

# Abstract

In this report a model for an image vision system is presented that is capable of recognizing car license plates independent from plate location, -size, - dimension, -color and -character style. We propose the application of a combined neocognitron type of neural network classifier in a generic car license plate recognition (CLPR) system.

The suggested system contains an image-processor, a segment-processor and five combined neocognitron network classifiers which act as a character recognizer. The presented model of the system depends neither on specific license plate image features nor on license plates character style and size. Combining neocognitron classifiers was motivated by the fact that manually tuning a training-set for a large neocognitron network is tedious. It is shown how training set tuning for a large neocognitron network can be avoided. By connecting small neocognitrons specifically trained for character classes that were frequently wrongly classified, the performance of the recognizer in our CLPR was improved significantly. The use of a neocognitron recognizer contributes significantly to the generality of a CLPR system. Besides, character recognition-rates over 98% are in reach using the proposed neocognitron configuration.

The only universal character recognizer is the human visual system. Mimic only some of the capabilities of human vision seemed not a trivial task. However it is demonstrated that the introduced techniques for image- processing, image segmentation and character recognition make the constructing of a generic CLRP system in software possible. Our prototype system needs further analysis and development but it currently exhibits an overall plate recovery-rate of 65% in laboratory conditions. The overall performance can be easily pushed towards rates that are claimed by commercial applications in this area.

This report is a description of my graduation project in the group Knowledge Based Systems of the faculty Information Technology and Systems of Delft University of Technology, headed by Prof. Dr. H. Koppelaar. The graduation committee members were Drs. Dr. L. Rothkranz, Prof. Dr. Ir. E. Kerckhoffs and Prof. Dr. H. Koppelaar.

Bas Cornet  
Rotterdam April 1, 2001

**ITS**  
Information Technology and Systems

Mekelweg 4  
2628 CD Delft  
The Netherlands



**TU Delft**  
Delft University of Technology

Intentionally left blank

# Preface

This report discusses a newly developed Car License Plate Recognition (CLPR) system, which performs recognition of numberplate characters from digital photo images of cars. The system consists of two major subsystems.

First, we have developed a prototype for image processing. This subsystem implements existing algorithms on image-enhancing and image-segmentation, to find the individual characters on the picture. We have managed to isolate individual license plate characters from low resolution grey-scaled images without applying so-called size constraints and without selecting the relevant objects by looking at certain intensity values of pixels of the number plate. Of course there are certain restrictions regarding the 'quality' of the image but our pre-processor does isolate individual characters of Dutch license plates with a reasonable success rate without using explicit knowledge on the numberplate size, -dimensions, -color and -location.

The second subsystem deals with recognizing the isolated segments as alphanumeric characters. For this subsystem, we have implemented a neocognitron simulator that runs on a Personal Computer. Where other CLPR systems do use multilayer perceptron (MLP) artificial neural classifiers which get their input as a derived set of specific features (e.g. principle components, moments or projections) of the presented image, we have selected to pass the character images directly to the recognizer. We make use of a multilayer neocognitron artificial type of network as a character recognizer. The neocognitron is one of the most complex neural network architectures. Nevertheless, this network is chosen because this network is successfully applied to similar problems (recognition of handwritings) and because the neocognitron can process input, independent from sizing and position of the object picture in the visual field.

There is a lot of material covered in this report and since not all readers would have the same background or interest, we have divided the text into three main parts.

- Part I covers an overview of the fundamental steps within an automated image understanding system. It gives an introduction to the main characteristics of image acquisition, image preprocessing, image segmentation and pattern recognition. Most of the image preprocessing algorithms described in this part are used later in our CLPR implementation. In chapter 2 an introduction on the architecture and functionality of a neocognitron neural network is given which is used as the character recognizer in our system.
- Part II provides a comprehensive description of our CLPR architecture. Chapter 3 gives the overall design of the CLPR system and chapter 4 deals with the procedures followed to configure and train the neocognitron neural network used.
- Part III gives the software implementation of the CLPR. An overview of the system software layers is given and, where applicable, the functionality of the software libraries is given. Of all relevant image processing and userinterface functions, implementation details are described in

chapter 5. In Chapter 6 the system is evaluated and some suggestions are made to improve the overall performance of the system.

The appendix contains:

- The software specification of the implemented neocognitron simulator.
- The used neocognitron network configurations and training-set patterns.
- The test results on the used neocognitron networks.
- A library of digital photographs used in the verification of the system.

The source code of both the CLPR prototype and the neocognitron simulator are available on a CD-ROM. This CD-ROM also contains a copy of this report, the appendices, the training-set pattern files, network configuration files and the photographs used for system verification and system evaluation.

# Acknowledgements

I thank all those who enabled and encouraged me to start my study Information Technology at Delft University of Technology and finally see me complete this study with this report.

First, I want to thank my wife Jacqueline and our children Sebastiaan and Christiaan for their understanding, patience, and tolerance during the months I spent preparing this report. I surely will find much more time to make up for the many postponed family trips and parties and to finish the many “LEGO™ -projects” my sons and I have started.

I owe a great deal to Drs.Dr. Leon Rothkrantz. He has guided me through the field of image understanding and the usage of artificial neural networks in a technical application. Without his coaching and support it would not have been possible to write this report and finish my study in such a relatively short period of time. I would also like to thank Ir. Michal Steuer. He let me use his software implementation and documentation of a neocognitron network simulator. The availability of his original design and source code saved me a lot of time when implementing the neocognitron on a PC for use in a car license plate recognition system. Finally, I would like to thank Dr. Harrie Hendrickx and for reviewing this report. His recommendations on the style and contents of this report was of great value.

Intentionally left blank

# Contents

<b>INTRODUCTION .....</b>	<b>9</b>
---------------------------	----------

## **PART I**

### CHAPTER 1

IMAGE UNDERSTANDING .....	13
1.1 Image acquisition.....	14
1.2 Digital image preprocessing .....	15
1.2.1 Image definition and basic transforms.....	15
1.2.2 Complex image transforms.....	17
1.2.3 Spatial domain image enhancements.....	19
1.2.4 Frequency domain filtering.....	21
1.3 Image segmentation.....	22
1.3.1 Discontinuity detection.....	23
1.3.2 Region-oriented segmentation.....	23
1.4 Pattern recognition.....	24
1.5 Interpretation of results.....	25

### CHAPTER 2

AN INTRODUCTION TO CHARACTER RECOGNITION USING A NEOCOGNITRON TYPE OF ARTIFICIAL NEURAL NETWORK .....	27
2.1 Description of the neocognitron .....	27
2.1.1 Cells.....	27
2.1.2 Planes.....	28
2.1.3 Layers .....	30
2.1.4 Network .....	32
2.2 Training the neocognitron network .....	34
2.2.1 Training individual neocognitron cells.....	34
2.2.2 An example on how to create a training-set.....	36
2.3. Design and dimension of a network for handwritten character recognition .....	42
2.4. Network recognition performance.....	43
2.5. Recognizing printed characters.....	46

## **PART II**

### CHAPTER 3

THE ARCHITECTURE FOR A LICENSE PLATE CHARACTER RECOGNIZER .....	51
3.1. Design background.....	52
3.1.1. The license plate paradigm .....	52
3.1.2. Implementation limitations.....	52
3.2. System concept.....	53
3.3 The image processor.....	53
3.4 The segment processor .....	54
3.5 Character recogniser.....	56
3.6 CLPR blue print.....	57

### CHAPTER 4

FINDING AN OPTIMAL NEOCOGNITRON CONFIGURATION TO RECOGNISE CHARACTERS OF CAR LICENSE PLATES .....	59
4.1 Reviewing the results using Fukisuma's original network .....	59
4.2 Redefining training patterns .....	61
4.2.1 Refining the existing training set .....	61
4.2.2 Modifying the existing training set .....	61
4.3 Modifying layer and plane configuration .....	64
4.4. A combined neocognitron network .....	66
4.4.1 Recognition performance increase measured on real-world photographs .....	66
4.4.2 Combining neocognitron classifiers .....	68
<b>PART III</b>	
CHAPTER 5	
SOFTWARE IMPLEMENTATION OF THE LICENSE PLATE RECOGNIZER .....	75
5.1 Systems software components .....	76
5.1.1 Software layers .....	76
5.1.2 Software modules .....	77
5.2 The image processor .....	78
5.2.1 Contrast stretching .....	79
5.2.2 Filtering .....	79
5.2.3 Binarization .....	79
5.2.4 Area filling .....	80
5.2.5 Segment locator .....	80
5.3 The segment processor .....	80
5.3.1 Segment scaling .....	81
5.3.2 Segment binarizer .....	81
5.3.3 Segment copying/pasting .....	82
5.3.4 Segment thinning .....	82
5.4 The character recognizer .....	82
5.5 The user-interface .....	83
CHAPTER 6	
EVALUATION .....	85
6.1 Systems performance .....	85
6.1.1 Performance definition .....	86
6.1.2 Image processor performance .....	87
6.1.3 Recognition performance .....	89
6.1.4 Post-processor performance .....	90
6.2 Future work .....	92
<b>REFERENCES .....</b>	<b>93</b>



# Introduction

Some years ago Car License Plate Recognition System technology received renewed interest from the Dutch government because of a political resolution to introduce automated tollgates around the 4 major cities. Commercial CLPR system applications are widely used in traffic law enforcement systems in order to process photos that record speed limit excess automatically. The number of photos to be processed by these systems is low compared to the numerous photos that will be produced by the proposed tollgate cameras. The availability of an accurate, robust and reliable CLPR system is mandatory to successfully introduce these tollgates that seems desirable by Dutch politicians.

A CLPR system is also the object of research in this graduation project. Initially this research was started to find a model for a universal character recognizer for photographs that show a wide range of objects that all could be identified by an alphanumerical character string. Car license plates, signposting, sea-containers and wagonnumbers on trains are only a few examples of objects that could be recognized.

Although intuitively we know such a universal system would be very hard or even impossible to implement, a model for such a system should not necessarily be very complex. Basically such a system needs only a number of independent well defined processing steps.

- Isolate all possible character image segments on the photograph.
- Extract features from the isolated segments.
- Pass all caught segments to a character recognizer.
- Apply domain knowledge to the string of alphanumerical characters found.

By using existing image processing techniques, it will not be a major problem to extract all possible character images of a photo. We can look for example at high frequency components in the Fourier transform of an image, or find objects of equal size or color that are arranged in line or that are surrounded by border frames. We can pass the isolated and pre-processed segments to a neocognitron network which is capable of classifying any kind of character style invariant of location and size. Finally we make use of a knowledge based system that may detect and even correct erroneous character classifications based on domain knowledge. However the only universal character recognizer is the human visual system, and only mimic some of the capabilities of human vision seemed not a trivial task.

In this report a model for an image vision system is presented that is capable of recognizing car license plates independent from plate location, size, dimension, color and character style. Of course, car license plates have standard formats. However if we want to generalize to other objects, we do not want to use this a priori-knowledge. By far our model is not a universal character recognizer system, however we've tried to introduce some techniques that are generally applicable in such a system. From this model, a software system has been implemented. Based on a license plate model, we have implemented a simple image processor to isolate individual license plate characters. For character recognition, we have implemented the neocognitron network in software as proposed by Fukisuma. Our motivation to use this type of artificial neural network was because of its capability to read

generalized character styles independent from size or location. This type of network would make many image preprocessing steps superfluous and therefore, it would make the system concept more general applicable.

In this report the reader will not find a description of a universal characters recognizer system nor a complete model for such a system, but instead a prototype and model for a CLPR is described based on a neocognitron type of artificial network. Our system needs further analysis and development but, based on the recognition rate it currently exhibits, we believe the recognition performance can be easily pushed towards recognition rates that are claimed by commercial applications in this area.

# Part I

Intentionally left blank

## Chapter 1

# Image understanding

To find a solution in the problem domain of image understanding, the following model is commonly used.

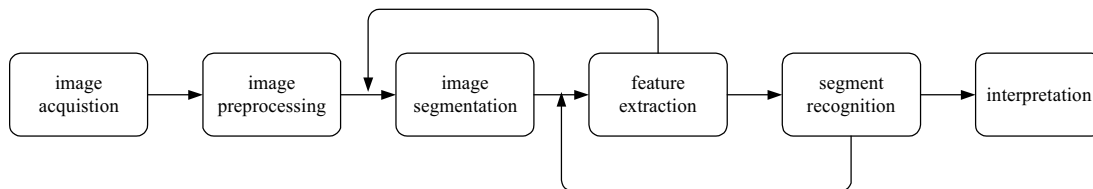


Fig. 1.0 A generalized image understanding system

- *Image acquisition* deals with the problem of getting a digital image representation of the scene of interest.
- *Pre-processing* are all image transformations that ‘improve’ the image in order to increase the change on successfully segmentate, and eventually understand, the image.
- *Segmentation* is defined as partitioning the pre-processed image such that areas of interest may easily be captured.
- *Feature extraction* is the process of determining specific features of the objects selected in the previous step in order to decrease the variety of all possible object appearances.
- *Recognition* deals with classification of the object according to its previous determined features.
- *Interpretation*, the last step in the image understanding model deals with verification of the results based on explicit knowledge of the objects recognized.

In order to increase the performance of image understanding systems, some stages in the process provide feedback to previous steps. The nature of feedback data depends mainly on the application area. The more complex the subject, the more feedback loops may be required to build a robust and reliable system.

In the next sections, the main topics of the different stages in the image understanding process are described in more detail. This chapter doesn’t pretend to give a complete overview on all possible techniques involved in the field of image understanding. Mainly those techniques and aspects are described which are used within our car license plate recognition system. Below in table 1.0 an overview is given on the subjects described in this chapter. Per section indications are given whether the subject was implemented in software and whether applied for license plate recovery. In our image recognition systems approach there was no need for a feature extraction stage in the processing pipeline. Therefore this subject is not described in this chapter.

Table 1.0 Stages in the image understanding process

Section	Subject	Impl <sup>*</sup>	Appl <sup>†</sup>	Remark
1.1 Image acquisition	Background information only	N	-	Used existing digital images
1.2 Digital image preprocessing	-Basic transforms	Y	Y	Inverse, grey-scaling
	-Complex transforms	Y	N	Fourier transforms
	-Spatial domain enhancements	Y	Y	Binarization, area-filling, filtering
	-Frequency domain enhancements	Y	N	Inverse Fourier transforms
1.3 Image segmentation	-Discontinuity detection	Y	N	Sobel operation
	-Region oriented segmentation	Y	Y	Pixel aggregation
1.4 Pattern recognition	-Multi Layer Perceptron	Y	N	SNNS MLP prototype was tried
	-Neocognitron	Y	Y	PC implementation neocognitron
1.5 Interpretation of results	Background information only	N	-	Typically post-processor aspects

<sup>\*</sup> indicates whether software functionality is written and available in the prototype CLPR

<sup>†</sup> indicates the application of the technique described to actually recover car license plates

## 1.1 Image acquisition

The first step in the image understanding process is image acquisition. Image acquisition encompasses a broad range of mechanisms and theoretically underpinnings that would go far beyond the scope of this report. Since, in our prototype CLPR system an image acquisition system has not been implemented, we confine with only a brief introduction on image acquisition.

The imaging device in an image understanding system would normally be a camera. A camera performs a perspective transformation. It projects the light reflected by three-dimensional objects onto a two-dimensional plane. Although the human eye does show some principal differences compared to a camera, they both consists of two basic components; a lens and discrete light receptor plane. In a camera, an optical lens focuses an image of the reflected light of the object or scene photographed onto a photographic film or electronic image chip. The light intensity values recorded by the film or the image chip forms the input information for an image understanding system. Within the human visual system light reflected by a scenery excitates the so-called *cones* and *rods* in the eye, which on their turn send electrical impulses to the visual cortex for human perception.

It would be very interesting to research how much the physical construction and mechanism in the eye contributes to the fact that humans can interpret images seemingly effortless. In automated image understanding systems, we shall have to make shift with mechanical cameras.

As stated in the previous section the task of the image acquisition module within an image understanding system is to get a digital image representation of the scene of interest.

The image acquisition module within an image understanding system should supply the preprocessing step with as much as possible information. It should deliver 'good quality' photographs under all conditions it was specified for. Image acquisition modules can be very complex because they have to compensate for all kind of physical recording limitations even modern cameras have. In a CLPR system used for law traffic enforcement for example the camera should not only deliver sharp images of moving cars under bad illumination conditions but also the image acquisition system should measure the speed and type of the cars.

In our prototype system there exists not an automated image acquisition module as such. In stead, we have used images produced by a digital pocket-camera and some digital scanned slide-images made available by the Dutch police.



Fig. 1.1 The imaging device for traffic law enforcement

## 1.2 Digital image preprocessing

Nowadays, digital image processing is applied in many different user areas. Enhancing pictures for human interpretation, automation of video animation and environment simulation in virtual reality systems are only a few examples where digital image processing is used extensively. Processing of image data in relation to computerized perception is another area of use. However, in all application areas the basic idea behind the processing of an image is *transformation*. In the field of human perception, the transformation of a picture serves for example sharpening; magnification or beautification. The goal of picture transformation for automated machine perception is to reduce the data of the image, such that features or areas of interest may be far easier isolated.

In literature, many different techniques and algorithms for image processing are described. The quintessence with respect to pattern recognition systems is to select those series of transformations that fit best to prepare the image for further steps that lead towards image interpretation. In the remainder of this section the theoretical background, terminology, and basic properties of some general image transformations are given.

### 1.2.1 Image definition and basic transforms

#### Image definition

An image may be represented by a *function* or a *mapping*:

$$z = f(x, y)$$

Equation 1.1

In this equation,  $z$  represents the brightness or color of an image at the spatial coordinates  $x$  and  $y$ . A digital image (both spatial and color variables have discrete values) is represented by a matrix :

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdot & a_{1m} \\ & & \cdot & \cdot \\ a_{n1} & & \cdot & a_{nm} \end{bmatrix}$$

Equation 1.2

Where matrix elements values indicate the brightness/color values of an image at the spatial coordinates  $n$  and  $m$ . Which representation is taken, is dependent on the characteristic of the transformation. In our case we will consider only grey-scaled images where the grey scale ranges from 0 to 255.  $Z$  in equation 1.1 is characterized by two components; the fraction of light received by objects ( $i$ ) on the scene and the fraction of light reflected by the object ( $r$ ) in the direction of the camera. Equation 1.1 may therefore be written as a product of  $i$  and  $r$ .

$$z = i(x, y)r(x, y)$$

Equation 1.3

### Inversion

The negative of an image is obtained by assigning each pixel in the image its 'negative' value. If on a grey-scale image the intensity values of pixels are in a range of [0,255]. The image negative is defined by:

$$z = -z + 255$$

Equation 1.4

### Contrast stretching

Within all segmentation procedures, detection of intensity values transitions plays an important role. In general, image contrast improvement will increase the success rate of segmentation especially when the image suffers poor illumination. Therefore a contrast stage in an image preprocessing procedure is often used. The following graph shows the grey-scale function which is applied on the image depicted in figure 1.3.

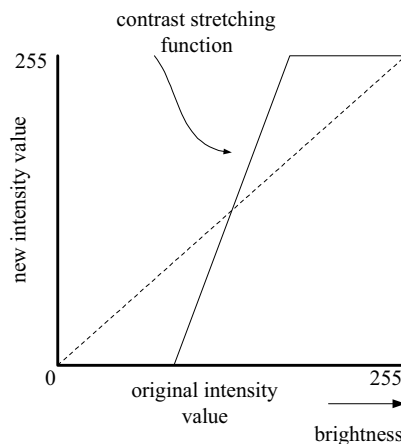


Fig. 1.2 Image contrast stretching function





Fig. 1.3 Left the original picture, right the contrasted stretched one

Both the negative image transform and contrast stretching transform are used in our application. There exist many more point processing transformation techniques the above mentioned alike; among others: grey-level slicing to highlight specific grey-level ranges, bitplane slicing to highlight or remove certain grey-levels or intensity value transforms based on the image color histogram. Since these are not used in our work, a description is not given here.

### 1.2.2 Complex image transforms

There exists many transformations on digital images that reveal characteristics that may be very useful in automated image recognition systems. The Fourier integral transform on a image may be useful in texture analysis. On a high-contrast image we will find more high frequency components and we will find mainly lower values for the frequency components in a low-contrast image. See figure 1.4. The Fourier transform is based on the fact that any continuous function defined on a finite interval could be expanded into a series of sine and cosine functions. Since an image may be represented by a function, we may transform the brightness values in the  $x$ - $y$  plane of the image.



Fig. 1.4 Left the original pictures, right the absolute value of the frequency components. White corresponds to the high values, black to low values

A Fourier transform of function  $f(x)$  is defined as:

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \omega} dx$$

Equation 1.5

In case of an image brightness function as described in section 1.1 the one-dimensional Fourier transform is extended to:

$$F(\nu, \omega) = \iint f(x, y) e^{-i2\pi(\nu x + \omega y)} dx dy$$

Equation 1.6

In practice we only have discrete functions f(x,y) the Fourier transform of an image will therefore be defined by:

$$F(\nu, \omega) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \frac{1}{MN} f(x, y) e^{-j2\pi\left(\frac{\nu x}{M} + \frac{\omega y}{N}\right)}$$

Equation 1.7

If we consider only images where N=M; equation (1.7) may be written as:

$$F(\nu, \omega) = \frac{1}{N} \sum_{x=0}^{N-1} F(x, \omega) e^{-j2\pi \nu x / N}$$

Equation 1.8

$$F(x, \omega) = \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi \omega y / N}$$

where

Equation 1.9

A Fourier transform of a 2-D image may easily be performed within two successive 1-D Fourier transforms. The pictures in figure 1.4 are created by implementing equations 1.8 and 1.9 in programming code together, using an existing 1-D fast Fourier implementation<sup>1</sup>.

A Fourier transformation as described above may be used in texture analysis of the image and for image enhancements and –segmentation techniques by filtering out specific frequency components.

---

<sup>1</sup> Written by Don Cross [dcross@intersrv.com](mailto:dcross@intersrv.com) available at <http://www.intersrv.com/~dcross/fft.html>

## 1.2.3 Spatial domain image enhancements

### 1.2.3.1 Point processing

Point processing techniques are based on the intensity value of individual pixels of the image. In the literature many techniques are described to enhance an image. In section 1.2.1, some basic transformations have been discussed. In this section some aspects of image binarization and area filling algorithms are given.

#### Binarization

Binarization or tresholding may be considered as a special form of high-pass filtering. The result of the following filter will highlight all pixels that have intensity values above a certain level.

$$\begin{array}{ll} \text{if } f(x, y) > T & \text{set pixel}(x, y) \text{ to BLACK} \\ \text{else} & \text{set pixel}(x, y) \text{ to WHITE} \end{array}$$

Equation 1.10

This type of filter is of practical use only in highly controlled environments. The problem with this kind of filter is selecting the threshold value  $T$  such that for all possible input images a binary image is created where the intended black-white relation is preserved. In literature, the binarization method described above is often labeled as global tresholding. As opposed to global tresholding several techniques, referred as local tresholding have been developed. These techniques have dynamic threshold values based on intensity values of neighboring pixels.

Figure 1.5 shows the results of the binarization procedures of equation 1.10 and 1.11. Figure 1.5.b shows the black/white image when 1.10 is applied with a fixed treshold value  $T$ . Figure 1.5.c shows the black/white image when the treshold value for each pixel is adjusted. The treshold value in the latter is assigned as the average value of the surrounding pixels.

$$T = (f(x, y) + f(x - 1, y) + f(x + 1, y) + f(x, y - 1) + f(x, y + 1)) / 5$$

Equation 1.11

Looking at equation 1.11 it is expected that this treshold applied would result in a very noising black and white image, because even small differences in intensity values of surrounding pixels may turn a pixel from bright to black or from dark to white. That is exactly what it does. As shown later in section 3.3 this transformation is very useful to highlight the numberplate boundary as well as the numberplate characters under particular conditions.



Fig. 1.5a,b,c Left the original picture, global tresholding applied on middle picture, right shows the result of the local tresholding procedure

### Areafilling algorithms

Various algorithms have been defined to fill image areas with a predefined color. The floodfill algorithm is the most practical when dealing with digital photographs. The floodfill algorithm starts at a pixel inside the interior of area that is to be filled with a specific color. If the intensity value of the starting pixel does not equal the value specified as the boundary, the color of the pixel is set to the specified fill color. The algorithm then continues by applying the same procedure for all the surrounding pixels. If the pixel under consideration has either the specified fill color or the specified boundary color the algorithm stops. Below a code sample of a recursive function is given that turns all pixels of a monochrome image to white except those that lie within an enclosed white area.

```
floodfill (int x, int y)
{
  if b[x][y]=BLACK
  {
    b[x][y] = WHITE
    floodfill (x+1,y)
    floodfill (x,y+1)
    floodfill (x-1,y)
    floodfill (x,y-1)
  }
}
```

#### 1.2.3.2 Spatial filtering

Spatial filtering refers to filtering techniques where pixel masks are used to enhance the image. Spatial filtering, as opposed, to point processing enhancements, considers the intensity values of neighbouring pixels to determine new intensity values.

#### Low-pass filtering

Low-pass filters are often used to clean the noise on an image. A side effect of noise reduction by applying low-pass spatial filters is image blurring. Generally, a mask describes filter operations. By moving the mask over the whole picture, each pixel in the image is assigned a new intensity value. The value  $z$  is set to the sum of the products of the masks weight values ( $w_1..w_9$ ) and the intensity values  $z_i$  of the pixels under the mask. The following spatial filter is often used to enhance noisy pictures.

$$1/5$$

$w_1$	$w_2$	$w_3$
$w_4$	$w_5$	$w_6$
$w_7$	$w_8$	$w_9$

Where

$$w_i = 1 \quad \text{and} \quad z_5 = (w_2z_2 + w_4z_4 + w_5z_5 + w_6z_6 + w_8z_8)/5$$

Equation 1.12

Below in figure 1.6 the effect of the filter is shown.



Fig. 1.6 Left the original image, right lowpass filtered image

### High-pass filtering

High-pass filters are, as opposed to their low-pass equivalent, used to sharpen the picture. High pass filters 'highlight' details while low contrast areas of an image remain unchanged. Edge enhancement is often implemented by applying a high-pass filter. The following figure gives an example of a special high boost filter.

$$1/9 \begin{array}{|c|c|c|} \hline w_1 & w_2 & w_3 \\ \hline w_4 & w_5 & w_6 \\ \hline w_7 & w_8 & w_9 \\ \hline \end{array}$$

Where

$$w_i = -1 \quad \forall i \neq 5 \quad w_5 = 9 \quad \text{and}$$

$$z_5 = (w_1 z_1 + w_2 z_2 + w_3 z_3 + w_4 z_4 + \dots + w_9 z_9) / 9$$

Equation 1.13



Fig. 1.7 Left the original image right the highboost filtered image

### 1.2.4 Frequency domain filtering

Image filtering in the frequency domain is in principle straightforward. First, the Fourier transform of the image is computed; the result is multiplied by a filter transfer function and finally the inverse Fourier transform is taken.

#### Lowpass filters

Sharp transitions in grey-levels between connected pixels do contribute to the high frequency components in the Fourier transform of the image. We recall equation 1.8. When multiplying this function by  $h(v, w)$  from equation 1.14 we can easily remove all sharp transition between pixels on the image just by taken the inverse transform. Figure 1.8 demonstrates the result.

$$h(v, w) = 0 \quad \text{if } \sqrt{v^2 + w^2} > D$$

$$h(v, w) = 1 \quad \text{elsewhere}$$

Equation 1.14 Function used in backward FFT



Fig. 1.8 Left original picture right the inverse transform using equation 1.14

### High-pass filters

As opposed to low pass filtering in order to smooth the image, the Fourier transform can be multiplied by 0 for all low frequency components (use equation 1.15). The inverse transform of the image will show only the sharp transitions of the original. See figure 1.9

$$h(\nu, w) = 0 \quad \text{if } \sqrt{\nu^2 + w^2} < D$$

$$h(\nu, w) = 1 \quad \text{elsewhere}$$

Equation 1.15 Function used in backward FFT to filter out low-frequency components



Fig. 1.9 Left original picture right the inverse transform using equation 1.15

## 1.3 Image segmentation

Within the pre-processing stage of an automated image understanding system, normally no information is extracted from the image. Generally, image pre-processing steps map an image from one disguise onto another.

Image segmentation, subdivides the image into parts that are considered to be the objects for recognition. The output of image segmentation as opposed to pre-processing, is not a transformed image but either cutouts of the original image or data related to segment properties.

Segmentation algorithms are based on two basic properties of the segment searching for:

- Local discontinuity in pixel intensity level.
- Pixel connection based on certain color similarities.

### 1.3.1 Discontinuity detection

In this section the most used line detection algorithm is presented; the Sobel operator. Similar operators can be used for point or edge detection. The Sobel operator to compute the derivative of image pixel intensity values in the y-direction is given by the following mask.

-1	-2	-1
0	0	0
1	2	1

The Sobel operator to compute the derivative of image pixel intensity values in the x-direction is given by the following mask.

-1	0	1
-2	0	2
-1	0	1

The black areas in the picture of figure 1.10b indicate that absolute values of the pixel intensity derivatives are above a certain level in either  $x$  or  $y$  direction.

$$\frac{\partial f}{\partial x} = G_x = -z_1 - 2z_2 - z_3 + z_7 + 2z_8 + z_9$$

$$\frac{\partial f}{\partial y} = G_y = -z_1 - 2z_4 - z_7 + z_3 + 2z_6 + z_9$$

Equation 1.16 Sobel operation



Fig. 1.10a,b Left original image, right the image after the Sobel operation of Equation 1.16 has been applied

### 1.3.2 Region-oriented segmentation

Discontinuity detection as described in the previous section does give us clues of what pixel areas to select in order to cutout the area of interest. Although the numberplate of the truck shown in figure 1.10 is surrounded by black pixels, further processing is required to cutout the actual license plate pixels. A technique to accomplish this is referred to as region growing by pixel aggregation. This procedure groups pixels that have the same color and that are connected into a region.

In order to reveal the absolute pixel coordinates of the edges of the numberplate we could work out the following algorithm.

1. Take the coordinates of utmost left bottom pixel of the image to start with.
2. Travel from this coordinate from left to right along each scan line of the image until a black pixel is hit.
3. This pixel will be the first pixel of the set of pixels belonging to the segment.
4. Add recursively all connected black pixels to the set.



5. Finally determine the maximum  $x$ - and  $y$ -coordinate en minimum  $x$ - and  $y$  coordinate of all pixels that belong to the set.
6. License plate edge coordinates will be  $(x_{min}, y_{min}), (x_{min}, y_{max}), (x_{max}, y_{min})$  and  $(x_{max}, y_{max})$ .
7. Redo from 2 to grab the next segment if necessary.

In our application we have implemented an algorithm as such to grab the areas in which individual characters of the number plate reside.

## 1.4 Pattern recognition

As with image processing, pattern recognition is applied in many different areas; speech processing early failure detection in machinery by trending characteristic parameters and share value forecasting are only a few examples.

Automated pattern recognition systems are hard to implement. In [6] Bishop stated: “*Often pattern recognition problems may seemingly be solved by humans effortless, computer implementations for this problem have in many cases proven to be immensely difficult however.*”

Traditional programming provides enough flexibility, expressive power, and theoretical foundation to solve the major problems in the image pre-processing, image segmentation, and feature extraction stage of an image understanding system relatively easy. There does not exist a serious hardware bottleneck in either pre-processing or in segmentating a digital image. However, solutions for pattern recognition problems in digital images are less obvious. The challenge within automated pattern recognition is; to program a machine such that instances of input patterns can be recognized with an expectable low misclassification rate.

Mathematically, a pattern recognition problem may be formulated as follows: “find a function which maps instances of the input sample to be recognized onto one of the classes that are defined on the pattern searched for”. Often it is convenient to denote the input sample as a vector  $\mathbf{x}$  and the output classes to vector  $\mathbf{y}$ .

$$f : R^N \rightarrow R^M \quad \mathbf{y} = f(\mathbf{x})$$

Equation 1.17

If there exists a “many to one”- relation between  $\mathbf{x}$  and  $\mathbf{y}$  and the input range and dimension are small numbers, then one can always find an implementation of the function in equation 1.17, just by simple constructing a mapping table by enumeration.

In practice however,  $N$  and  $M$  will not be small numbers and more seriously, we even might not have a clue to formulate a sensible relation between the input vector and the output vector.

Statistics form the framework to formulate solutions for pattern recognition problems. Neural networks have shown many powerful results in the field of pattern recognition and may be viewed as an extension on conventional techniques in statistics. We will use neural network pattern recognizers without given an introduction to statistical pattern recognition because this would be far beyond the scope of this report.

Neurocomputing as opposed to traditional programming, gives us tools to find functions like equation 1.17 that eventually may serve as a pattern recognizer function on a specific problem. Nowadays, often the term connectism is used for a class of algorithms that have a structure and behavior alike the model that currently exists on the functioning of the human brain. Typically, a neural network is trained to recognize pattern-classes in the image by supplying it with sufficient samples. If the recognition rate is acceptable after a certain time of training the network may be used as an approximation of the function, we were looking for, as specified in equation 1.17.

Most of the neural network applications require the original input sample to be classified to be pre-processed. This pre-processing, often referred as feature extraction, has often greatly improved the performance of the recognition system. This is mainly because we reduce the dimension  $N$  in formula 1.17. Suppose we want to recognize a character figure on a 256\*256 pixel image. We could use simply all 65.536 pixel values directly as input to the neural network.  $M$  would be 36 in this case if all



alphanumeric characters are considered; N would be 65,536 ! This is not necessarily a problem computational wise but the size of the training set would be huge. Therefore, we select only specific features on the input sample. Moments and principle components of the image to be recognized are often extracted before the image is fed to the neural network recognizer.

We have used two types of neural networks:

- A multiple layer perceptron network using 16\*20 binary pixels directly without preprocessing.
- A neocognitron type neural network.

Image segment samples to be recognised can be used directly in a neocognitron, provided some trivial preprocessing on the input segment is preformed like binarization. Viewed from the outside a neocognitron network type has the same processing function as any other type of artificial network. It provides a mapping like equation 1.17. Training/learning the network can be carried out explicitly as opposed to MLP's which have implicit training methods.

## 1.5 Interpretation of results

Humans can often recall complete information from partial or unclear observations. Everybody can interpret this text even if we dn't prnt th vwls bt nly th cnsnnts. Somehow, we have explicit knowledge about text. We know the syntax and grammar structure and rules applied to text and use this knowledge after having recognized the underlined printed text above. Another example of explicit knowledge: we would never classify an object as an elephant if observed as something big, high above us in the sky, since we know elephants do not fly usually. It would be more likely an airplane or an UFO.

The image processor, feature extractor and recognizer module in an image understanding system will never perform 100%. Misclassification will inevitably occur. Therefore, many image understanding systems have interpretation modules attached as final step in the processing chain. These post-processor modules do hold the explicit knowledge of the problem domain.

There is no general architecture of such a module since this highly depends on the domain of the image understanding system. In case of a CLPR for Dutch license plates, a hardcoded procedure of some syntax rules may be used to try to eliminate or correct recognition errors. If a silhouette recognizer for flying objects is the subject of our image understanding system, a knowledge base and an inference engine would be necessary to increase the understanding performance of the system.

In our system design, the post-processor is given a place in the architecture. We have not implemented any software module however, that acts as a post-processor.

Intentionally left blank

## Chapter 2

# An introduction to character recognition using a neocognitron type of artificial neural network

## 2.1 Description of the neocognitron

The basic function of the neocognitron is to act as a pattern recognition network. The neocognitron is mainly used to classify spatial binary character images. The neocognitron has been described comprehensively in many textbooks (i.e. Hecht-Nielsen[7] pp198-210) and many papers (i.e. Fukisuma [2][3]). Because describing the neocognitron in short but formal matter is a rather difficult task (because of the complexity on functionality and architecture), we will only briefly explain the items of the original neocognitron proposed by Fukisuma[2][3] which may be relevant for implementing the neocognitron in software and using the neocognitron as a character recognizer in a CLPR.

The neocognitron, developed between 1980 and 1990, is typically a hierarchical network, composed as cascaded connected layers. Neocognitron layers on their turn are composed of one or more slabs or planes. A plane is built as a two-dimensional array of cells. Often the neocognitron network is described top-down in literature. From a logical network model towards individual cells. In the next sections we have selected to describe the network bottom-up. That is starting with the cell description and ending with a complete network model.

### 2.1.1 Cells

The cells used in neocognitron networks are fundamentally simple objects. In the neocognitron network three types of cells are defined. S-cells, C-cells and V-cells. Cells may be compared to the classical perceptrons used in MLP artificial neural networks. Each type of cell is assigned to perform a typical function within the neocognitron.

The S-cells in the neocognitron network are the feature extracting cells. The excitation of a particular cell will be high only if a specific input pattern is presented (a particular feature) to its inputs. Which features a S-cell should detect is determined by training. Training is basically reinforcing the input connection weights of the cell according to certain rules. These rules are presented later. A S-cell is depicted in figure 2.1a. The excitation of a S-cell is calculated by equation 2.1a.

$$U_s = r\phi \left[ \frac{1 + \sum_{i \in A} U_{c_i} a_i}{1 + \frac{r}{r+1} b U_v} - 1 \right]$$

Equation 2.1a

In equation 2.1a  $A$  is the set of all cells in the connectable area. The connectable area will be explained later.  $\varphi$  is defined a function:  $\max(0, x)$ . The constant  $r$  is called the selectivity parameter and will also be explained later.

The V-cells are inhibitory cells. Their input connection weights are fixed, their output connections weight is reinforced by training. A V-cell is depicted in figure 2.1. The excitation of a V-cells is calculated by equation 2.1b.

The C-cells have an important task to make the network tolerant for small pattern distortions and deformations and translations of patterns. How this is accomplished will be explained in the next section. A C-cell is depicted in figure 2.1. The excitation of a C-cells is calculated by equation 2.1c. C-cells have predefined input connection weights which cannot be changed during training.

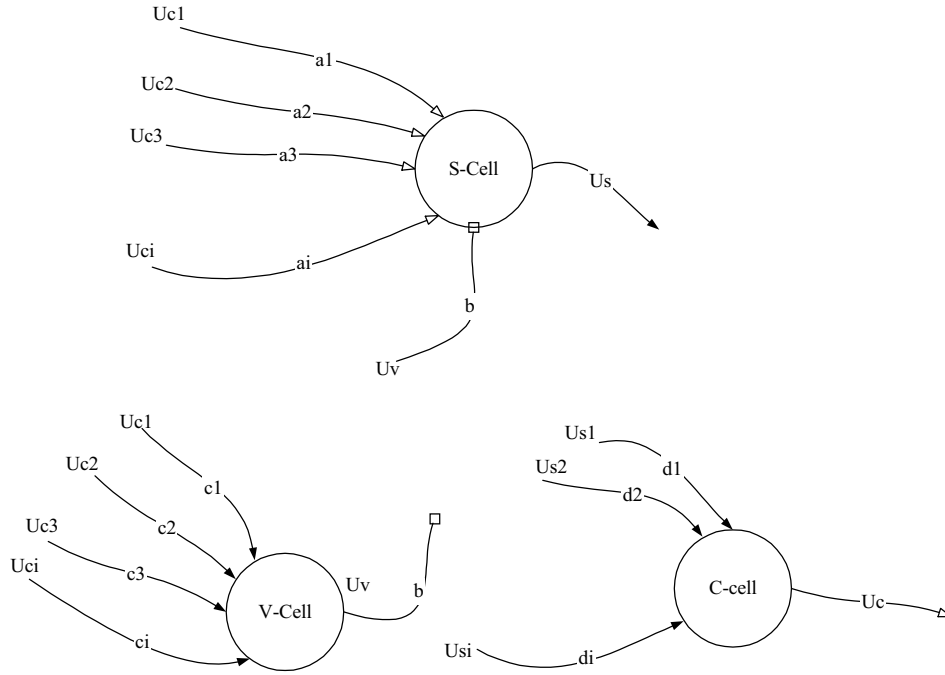


Fig. 2.1 Neocognitron type cells. a,b,c and d are the different cell weights values,  $Uc, Uv$  and  $Us$  are cell excitation values

$$Uv = \sqrt{\sum_{i \in A} U^2 c_i c_i}$$

Equation 2.1b

$$Uc = \varphi \left( \sum_{i \in A} Us_i d_i \right)$$

Equation 2.1c

Where  $\varphi$  is a function defined as follows:  $x / (1 + x)$ .

### 2.1.2 Planes

As there exist three kinds of cells in a neocognitron, there also exist three kinds of planes in a neocognitron architecture. Their names will not be very surprising: S-planes, C-planes, and V-planes. Planes hold two dimensional arrays of S-cells, C-cells or V-cells.

The number of plane outputs equals the number of cells in the plane and the number of inputs equals the number of cells times the number of input connections per cell. Per plane, all cells have exactly the same structure; their weights however will be different because of training, as we will see later. C-cells and V-cells are often modeled together on one plane because each C-cell is connected to exactly one V-cell and because their inputs are connected. This is shown in figure 2.2. A model of an S-plane is displayed in figure 2.3.

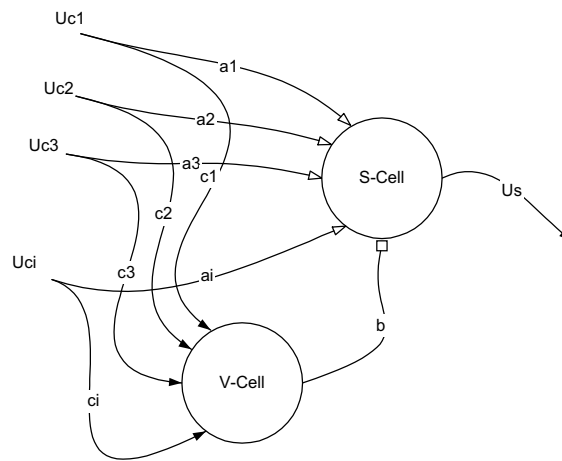


Fig. 2.2 V-S cell connection

Within a neocognitron network, two specific types of C-planes are defined: one input plane and several (more than one) output planes. The cells of the input plane only have output connections while the output C-plane typically has only one cell. There exists as much output planes, as there are different pattern-classes to be recognized by the network.

For example a neocognitron designed to recognize the numerical characters 0 through 9 from a 19 by 19 pixel binary image will have an input C-plane with 361 cells while it will have 10 output C-planes. Such a neocognitron basically implements a mapping as defined in equation 2.2.

$$f : R^{361} \rightarrow R^{10}$$

Equation 2.2

In the following section it will be explained how the ‘hidden’ planes and their interconnections are organized in so-called network layers.

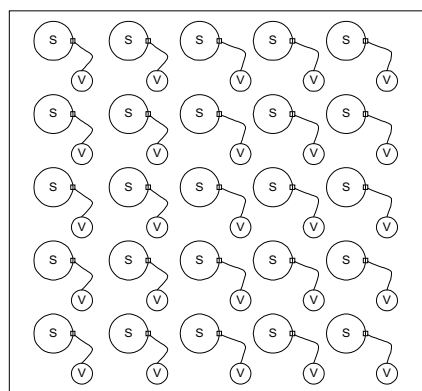


Fig. 2.3 A 5\*5 S-plane

### 2.1.3 Layers

A neocognitron network will have one or more hidden layers. Each hidden layer has two sublayers; these are called a C- and S-sublayer. C-sublayers hold a one dimensional array (a row) with one or more C-planes, S-sublayers hold one or more S-planes (also organized in a row) and exactly one V-plane. The number of C-planes is less or equal to the number of S-planes. Since we would have more than one plane per sublayer, we need additional indexes to indicate the output of a specific cell within a layer. Therefore we rewrite equations 2.1a, -b and -c by:

$$U_c \equiv U_{c(\eta,\kappa)}$$

$$U_s \equiv U_{s(\eta,\kappa)}$$

$$U_v \equiv U_{v(\eta)}$$

Equation 2.3

Where  $\eta$  is the  $(x,y)$  position of a cell within a plane and  $\kappa$  is the plane number within a layer. Note that the output of V-cell has no plane index  $\kappa$ , because there is only defined one V-plane per layer.

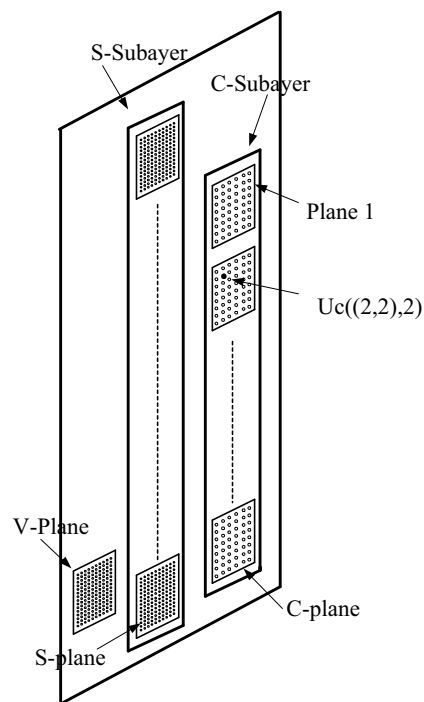


Fig. 2.4 The structure of a hidden layer of a neocognitron

Before explaining how signals are propagated between sublayers, we introduce two new concepts: *connectable area* and *neuron gap*. These two entities play an important role in the configuration of the network and determine the interconnections of cells.

Each neuron or cell takes input from several cells of planes in a previous (sub) layer. The source neurons all lie within a square area. This area is referred to as the connectable area of the cell considered. Figure 2.5 shows the connectable area of  $3 \times 3$  in the source plane for the center cell in the target plane.

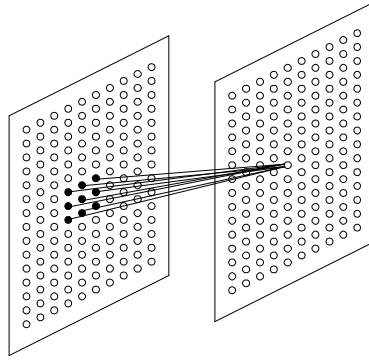


Fig. 2.5 The connectable area between two planes

In figure 2.5 both the source and target plane are of the same size and neuron density. The neuron gap for the target plane is 1. The neuron gap is defined as the distance of neurons in the layer considered compared to the distances between neurons in the source plane. Figure 2.6 illustrates the neuron gap parameter if we have different layouts of source and target planes. As shown in figure 2.6b we may use the neuron gap parameter to enlarge the area of the source plane that is visible by the target plane.

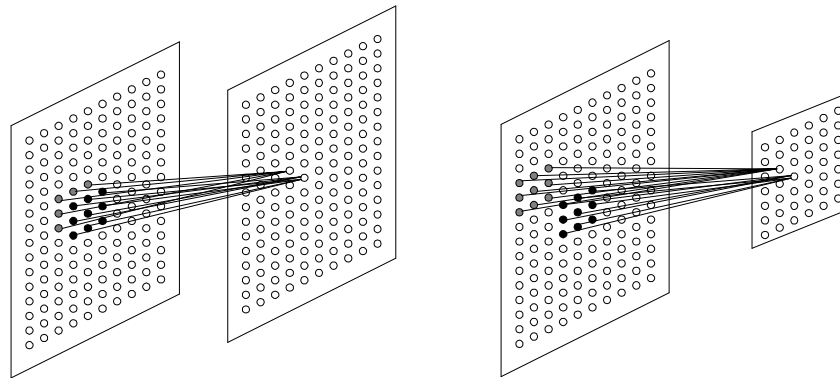


Fig. 2.6a Reachable area if neurongap=1      Fig. 2.6b when neurongap = 3

Layer intra-connections:

Within a layer the C-planes are connected to S-planes. A C-plane may be connected to one or more S-planes. The designer of the network determines these intra-connections. Which plane is connected to which is given by a discrete function:  $j(\kappa, \rho)$ .  $j=1$  if planes  $\kappa$  and  $\rho$  are connected;  $j=0$  otherwise. Basically, planes are connected through the cell connections shown in figure 2.6. Each cell in the target plane is linked to ‘many’ cells in the source plane. The size of the connectable area and the value of the neurongap parameter determine cell’s connection.

Layer inter-connections:

Overall a layer, all S-planes are connected to all C-planes in the previous layer. S-planes are connected to C-planes in the same way as layer intra-connections have been defined. The single V-plane in the layer is also connected by cell connections. Each V-plane cell is connected to all C-planes of the previous layer. The V-plane cell outputs are connected to a special inhibitory input of the cells in the S-plane of the same layer.

In figure 2.7 some of the intra and inter plane connections are drawn. Obviously painting a neocognitron in its full glory would result in a beautiful but very complex picture. In the next section, a more schematic picture is given of the neocognitron.

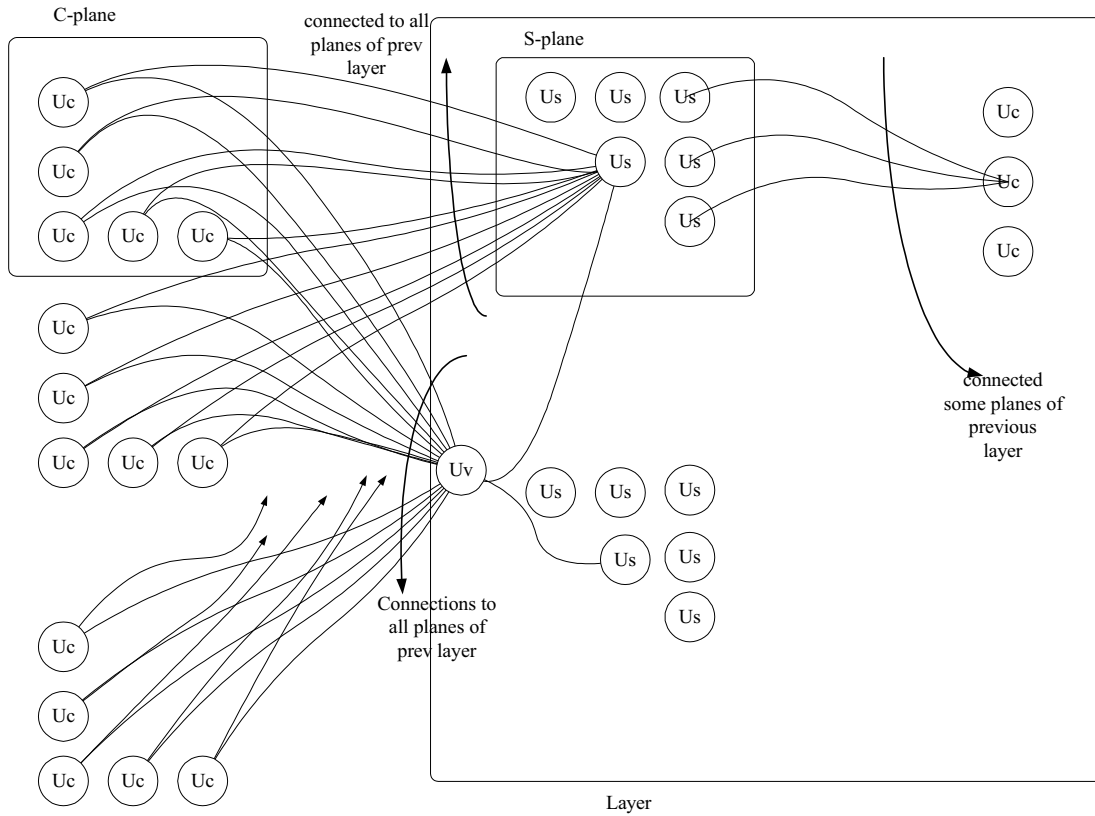


Fig. 2.7 Intra- and Inter layer connections

## 2.1.4 Network

Before the propagation functions per cell type are given in equation 2.7a, -c we present in table 2.1 the identifier symbols used in these equations.

Table 2.1 Formula identifiers used to describe the neocognitron

Identifier	Description
$U_s, U_c, U_v$	Output of a S-, C-, V-Cell
$\eta$	(x,y) absolute cell position in a plane
$\kappa$ and $\rho$	Index numbers of source and target plane
$\lambda$	Layer index number
$v$	(x,y) relative cell position in a connectable area
$r$	Selectivity parameter
$K$	Set of numbers of planes in a sublayer
$A$	Set of elements v e.g { (0,0), (0,1), (0,2), (1,0), (1,1), (1,2), .. ,(2,2) } for a 3*3 area
$q$	Trainings coefficient normally set aprox 100000
$a, b$	Reinforcable cell weights
$c, d$	Fixed cell weights (see equation 2.4)
$\gamma, \delta, \delta\text{-bar}$	Network constants determined by design to set the fixed cell weights
$j(\kappa, \rho)$	Connection function.
$\phi$	A maximum function (see equation 2.5)
$\psi$	A normalisation function (see equations 2.6)



$$c_\lambda(v) = \gamma_\lambda^{|v|}$$

$$d_\lambda(v) = \bar{\delta}_\lambda \delta_\lambda^{|v|}$$

Equation 2.4 Weight functions

$$\varphi(x) = 0 \quad x < 0,$$

$$\varphi(x) = 1 \quad x \geq 0,$$

Equation 2.5 The maximum function

$$\psi(x) = \frac{\varphi(x)}{1 + \varphi(x)}$$

Equation 2.6 The normalisation function

Equation 2.4 together with the knowledge that S-planes and C-planes are based on a connectable area and the neuron gap parameter lets us rewrite the output excitation function of C-, S- and V-plane cells as earlier defined in equation 2.3.

$$Uc \equiv Uc_{\lambda(\bar{\eta}, \rho)} = \psi \left[ \sum_{\kappa=1}^K j(\kappa, \rho) \sum_{\bar{v} \in A} d_\lambda(\bar{v}) Us_{\lambda(n, \bar{\eta} + \bar{v}, \kappa)} \right]$$

$$Uv \equiv Uv_{\lambda(\bar{\eta})} = \sqrt{\sum_{\kappa=1}^K \sum_{\bar{v} \in A} c_\lambda(\bar{v}) U^2 c_{\lambda-1(n, \bar{\eta} + \bar{v}, \kappa)}}$$

$$Us \equiv Us_{\lambda(\bar{\eta}, \rho)} = r_\lambda \varphi \left[ \frac{1 + \sum_{\kappa=1}^K \sum_{\bar{v} \in A} a_\lambda(\bar{v}, \kappa, \lambda) Uc_{\lambda-1(n, \bar{\eta} + \bar{v}, \kappa)}}{1 + \frac{r_\lambda}{r_\lambda + 1} b_\lambda(\rho) Uv_{\lambda(n, \eta)}} - 1 \right]$$

Equation 2.7a,b,c C-cell excitation, V-cell excitation and S-cell excitation

Using the equation above we can propagate the excitations of the input plane towards the output planes. These functions may not seem to be very trivial however one thing is clear, on whatever input, we won't get any other output values than 0 because the weights  $a$  and  $b$  are initially set to zero. The weight values  $a$  and  $b$  should be set to sensible values. This is accomplished by training the network. How this is done is described in the next section.

In figure 2.8, finally a schematic overview is given on a neocognitron configuration designed to recognize handwritten characters. The squares drawn within the plane boundaries designate the connectable areas. The circles on the last sublayer 'Uc4' designate the single cell planes from which finally the classification of the input image sample is derived.

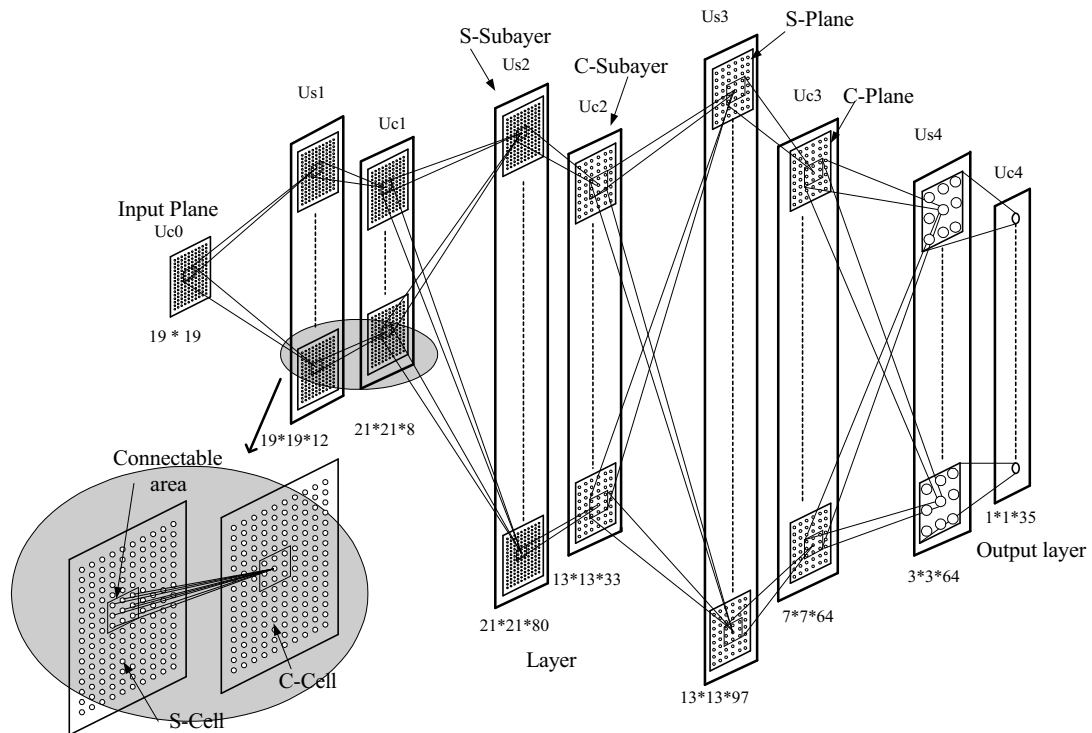


Fig. 2.8 Hierarchical network structure of neocognitron model. This model is used to implement Fukisuma's handwritten character recognizer

## 2.2 Training the neocognitron network

Training a neocognitron is a complex task. Fukisuma has formally defined how a neocognitron is to be trained. This is described in section 2.2.1. Unfortunately, there is no formal definition on how to create a suitable training set for a neocognitron. In section 2.2.2 an example procedure and some guidelines are given to implement a training-set to recognize simple objects. As will be demonstrated in section 2.2.2, training-set samples are found intuitively by studying the characteristic layout of the objects to be classified. A formal procedure to choose training features based on network response on the level of individual cell excitations in the hidden layers is not yet available.

### 2.2.1 Training individual neocognitron cells

The neocognitron network can be trained to classify its input sample images using a supervised method or through self-organization, also called unsupervised training. Although the neocognitron simulator used has functionality build in to perform weight sharing and unsupervised training, the procedures for unsupervised training are not described in this report, because in our application, we do not use the unsupervised training method.

The supervised training approach involves the use of "direct" inputs to the S-plane cells in each sublayer in the network. The following rules apply on a supervised training session:

- All training patterns are passed to the input plane of the network.
- Each S-plane is learned to recognize one specific input pattern.
- Each cell in a specific S-plane will have the same weights setting after training.
- Training set image samples are centered at the input plane.
- Supervised training starts with layer number 1, subsequent layers are trained after all planes in previous layers have finished training.
- Training-set patterns are binary images with intensity values 0 and 1.

As stated earlier, training the network is adjusting the weights  $a$  and  $b$  of the S- and V-cell respectively. Fukusuma has defined the following formula's to adjust the weight  $a$  and  $b$  during the supervised training session.

$$\Delta a_{\lambda}(v, \kappa, \rho) = q_{\lambda} c_{\lambda}(v) U c_{\lambda-1}(\eta+v, \kappa)$$

$$\Delta b_{\lambda}(\rho) = q_{\lambda} U v_{\lambda-1}(\eta)$$

Equation 2.8a, b a-weight adjusting, b-weight adjusting

Let us consider the following situation. We have created a new network. All weights are initialized to zero. The connectable area of the S-planes in the first sublayer is defined as a 3\*3 area. The input plane and the S-planes in the first sub-layer have the same size so the neuron gap parameter value is set to 1. Furthermore, we have set the  $\gamma$  parameter to a value of 0.9. Table 2.2 shows the weights of the S-cell and the V-cell after being reinforced. Obviously the weights are set conform the 'layout' of the input training-set pattern.

Table 2.2 The S-cell weights after training one input sample

Relative position	y↓	x→	Weight 'a' of the C-cell		
			1	0	-1
1	1		8615	0	0
0	0		0	10000	9000
-1	-1		0	0	0

Weight b of the v-cell 16618

Does the plane just trained above recognize the input image it was trained for? Yes, it is not difficult to prove that cells in this plane will be activated only if the learned input pattern is presented. Note that our starting point was 'all weights initialized to zero'. Hence  $\Delta a = a$  and  $\Delta b = b$ . We rewrite equations 2.8 and substitute these in 2.7c. We will get equation 2.10 for the output excitation value of cells in the trained plane.

$$\Delta a_{\lambda}(v, \kappa, \rho) = a_{\lambda}(v, \kappa, \rho) = q_{\lambda} c_{\lambda}(v) U c_{\lambda-1}(\eta+v, \kappa)$$

$$\Delta b_{\lambda}(\rho) = b_{\lambda}(\rho) = q_{\lambda} \sqrt{\sum_{\kappa=1}^K \sum_{\bar{v} \in A} c_{\lambda}(\bar{v}) U^2 c_{\lambda-1}(n, \bar{\eta}+\bar{v}, \kappa)}$$

Equation 2.9 Rewritten weight values

$$U_s \equiv U_{s_{\lambda(\bar{v}, \rho)}} = r_{\lambda} \varphi \left[ \frac{1 + \sum_{\kappa=1}^K \sum_{\bar{v} \in A} a_{\lambda}(\bar{v}, \kappa, \lambda) U c_{\lambda-1}(n, \bar{\eta}+\bar{v}, \kappa)}{1 + \frac{r_{\lambda}}{r_{\lambda} + 1} b_{\lambda}(\rho) U v_{\lambda}(n, \eta)} - 1 \right]$$

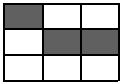
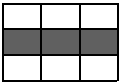

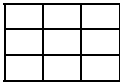
or

$$U_s \equiv U_{s_{\lambda(\bar{v}, \rho)}} = r_{\lambda} \varphi \left[ \frac{1 + Q_s}{1 + \frac{r_{\lambda}}{r_{\lambda} + 1} Q_v} - 1 \right]$$

Equation 2.10

Four different cases can be distinguished. Table 2.3 shows all four cases. From 2.10, the excitation value of the trained cell(s) in the S-plane can be derived.

Table 2.3 Trained S-cell excitations

Perfect match	Partly matched	Over matched	No single match
			
$Q_s=Q_v \gg 1 \Rightarrow$	$Q_s < Q_v \Rightarrow$	$Q_s \ll Q_v \Rightarrow$	$Q_s=Q_v=0$
Cell output = 1	Cell output $\approx$ 1	$0 < \text{Cell output} \ll 1$	Cell output = 0
Maximum cell output that can be reached		Inhibitory v-cell wins over excitory s-cell inputs	Trivial

In figure 2.9, the relative output values are given of a network plane trained with the sample in table 2.2. The input plane size and the S-plane size are 19 \* 19.

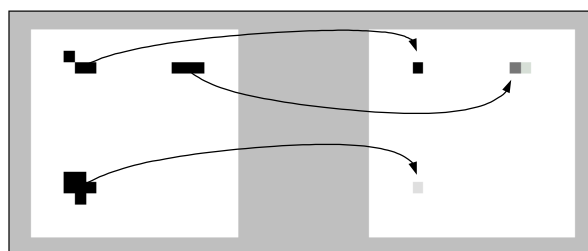


Fig. 2.9 Left: 3 segments in the input-plane network's Right: response on first layer S-plane

So far, we have described training the first plane in the first layer. Training the rest of the network however is more of the same, we train the network plane by plane, layer by layer. Constantly and consistently applying the formulas for cell excitation in equation 2.7 and the formulas for weight changing in equation 2.8 will be awarded by getting a trained neocognitron network for image classification. It will be unfeasible to analytically write down the behavior of the network during training in an understandable matter, but just that is one of the secrets of the neocognitron artificial network. In the next section an example is given on how a training-set is designed for recognition of images of arrow symbols.

## 2.2.2 An example on how to create a training-set

Suppose we are asked to design a neocognitron network that reads the direction to which an arrow symbol points. In order to keep the network small and simple, we require the network only to classify arrows pointing up or pointing down. Furthermore, the network only has to recognize arrows of a certain style. In figure 2.10 some examples of arrows to be recognized are given.

Before the network can be trained, a network configuration is to be determined. The design and configuration of the layers in the network however is depended on the training-set required. By

analyzing the basic properties of typical sample images of the objects to be recognized this chicken and egg situation is avoided. Property analysis applies to the following procedure:

1. Find the basic(smallest) line segments of the images to be recognized.
2. Find combinations of basic line segments that form parts of the object.
3. Find more complex combinations that specify characteristics (larger) parts of the objects.
4. Finally, find the typical structural layout of all object classes.

Step 3 is repeated depending on the complexity of the images and the number of classes to be recognized.

Thus, as described above, the first layer is trained with rather simple images, the successor layers are trained with more complex patterns of larger size. Normally the trainingset patterns of successor layers are combinations of patterns in previous layers. The last plane is trained with typical samples of the images finally to be classified.

The 4 patterns shown in figure 2.11 are designed to detect line segments at various orientations. In the typical samples shown in figure 2.10 we only find horizontal, vertical and 45 degree angle line segments.



Fig. 2.10 Samples of objects to be recognized in this example

Fig. 2.11 Basic line segments

In figure 2.12 all relevant combinations of basic line segments are shown. Note that only those combinations of basic line segments are chosen that actually form a part of the arrow images to be recognized. Finally, in figure 2.13 typical arrow samples are depicted.



Fig. 2.12 Combined basic line segments

Fig. 2.13 Typical object layout

Since the number of patterns (shown in figure 2.11 through 2.13) per layer are known, the network layout can be determined. In figure 2.12 the network layout of the 'arrow' recognizer is given. The input plane is defined as a 19\*19 cell plane. The output C-layer consists of two planes both having one cell, the upper cell indicates the arrow is pointing up, lower cell indicates the arrow is pointing down. Two hidden layers are defined. The first layer is capable of recognizing the small elementary line segments of arrow images, the second layer is trained to recognize the combinations of line segments that occur in the symbol images to be recognized. The last layer, the output layer is trained with

typical samples of the symbols to be recognized. In figure 2.14, the training-set samples are given for each layer. Note that a plane may be trained using more than one training-set sample. This superimposed training makes the network more robust against deformations and scale invariant.

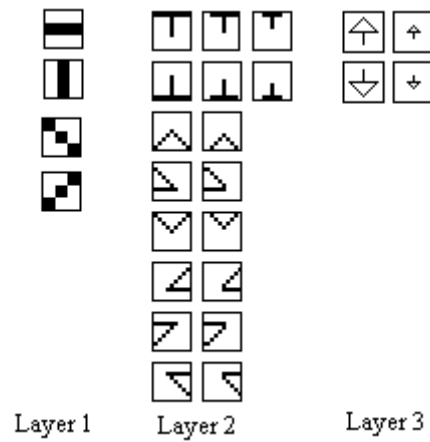


Fig. 2.14 The complete training set of the 'arrow' recognizer

In this case, the network parameters like selectivity, gamma and delta are set to the same values Fukisuma uses in his network for handwritten character recognition. The network parameter values do greatly influence the robustness and accuracy of the network, however in this example the exact values of these parameters are not very relevant. The same counts for the connectable areas. We have selected some sensible ranges, the exact sizes of the connectable areas are not very relevant in this example either. Figure 2.15 shows the network layout of the arrow-recognizer.

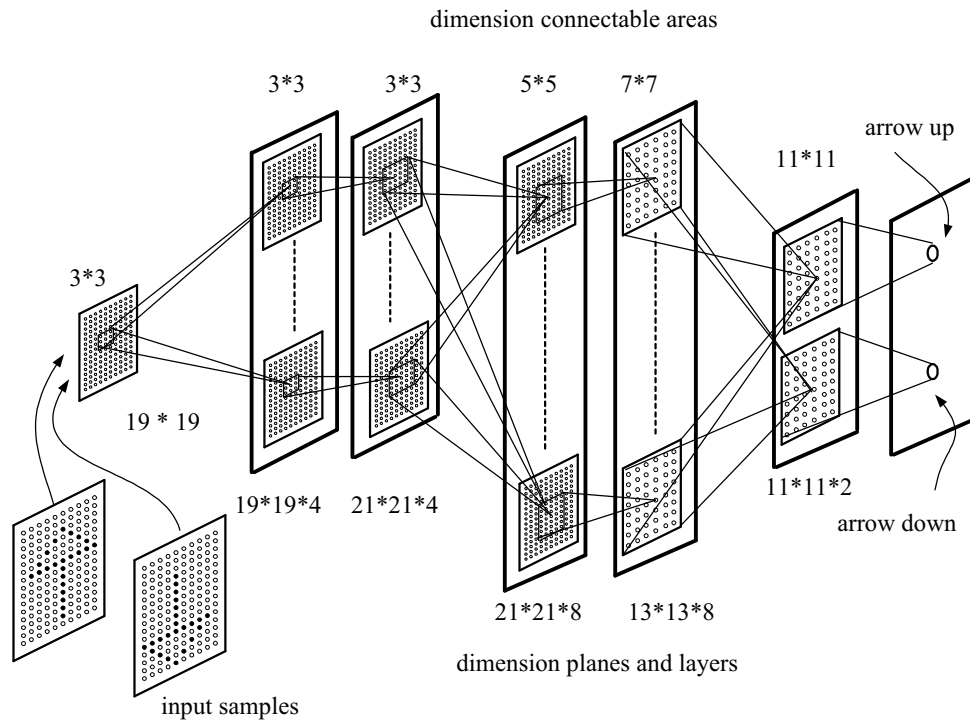


Fig. 2.15 The arrow recognizer

Before the total network is trained according to the training-set patterns defined in figure 2.14, we demonstrate the individual response per layer per plane of a partly trained network.

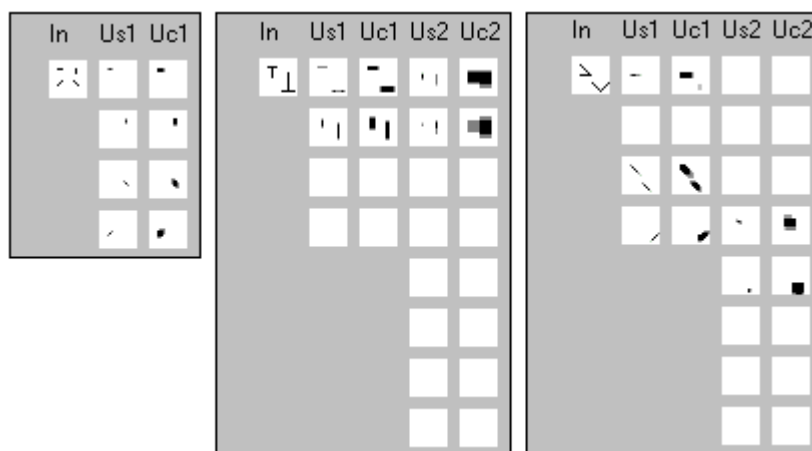


Fig. 2.16a-c Individual cell excitations on line segments and line combinations

In figure 2.16a the network is supplied an input sample with all possible basic line segments. In the planes of the layer labeled  $Us_1$ , the excitation of the cells is visible. Note that only those cells in the planes are excited for which it was trained. The first plane (counting top-down) was trained to detect horizontal lines segments. It clearly shows it has detected a horizontal line segment in the upper left corner of the input plane. The second plane was trained to detect vertical lines segments. Cells in this plane clearly shows a vertical line segment was detected in the upper right corner of the input plane.

In figure 2.16b and 2.16c the excitation of line combinations are given for both layer 1 and layer 2. In figure 2.16c, plane 4 in the layer labeled  $Us_2$  is trained to detect a combination of a horizontal and a  $-45$  degree angle line segments. Individual cells in the plane considered show that such a feature has been detected in the upper left corner of the input plane.

As stated in the previous section the network is trained plane by plane, layer by layer and all training samples are supplied through the input layer. Training the feature combination as described in the previous paragraph for fourth plane in layer  $Us_2$  is implicitly accomplished by the network. In fact we do train this plane by adjusting the its cell weights based on the excitation in the previous C-layer (the first plane and third plane in layer  $Uc_1$  in this case) and not based on the training-set sample as available from the input plane directly. In figure 2.17 it is shown that in this simple case it is relatively easy to trace the individual cell excitations through the network caused by a training-set sample. In a neocognitron network designed for character recognition we have for example 80 planes in layer 2 and 97 planes in layer 3 all connected to each other. In such a network is it would be very difficult to find the relation between individual cell excitations of the plane under training and any training-set images of previous planes.

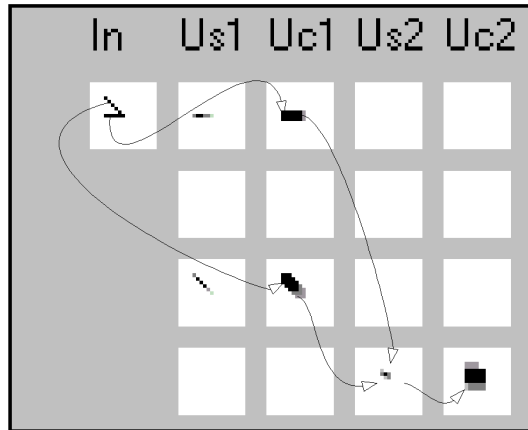


Fig. 2.17 Training-set sample propagation

Finally, in figure 2.18 it is demonstrated that the network and its training described in this section indeed classifies the direction of the arrow of the sample image correctly. The network exhibits robustness against image deformation and clearly the network recognizes object location invariant. Note the network response of the last network image in figure 2.18, both output cells have the same excitation. The network classifies this sample as 'up' and 'down'.



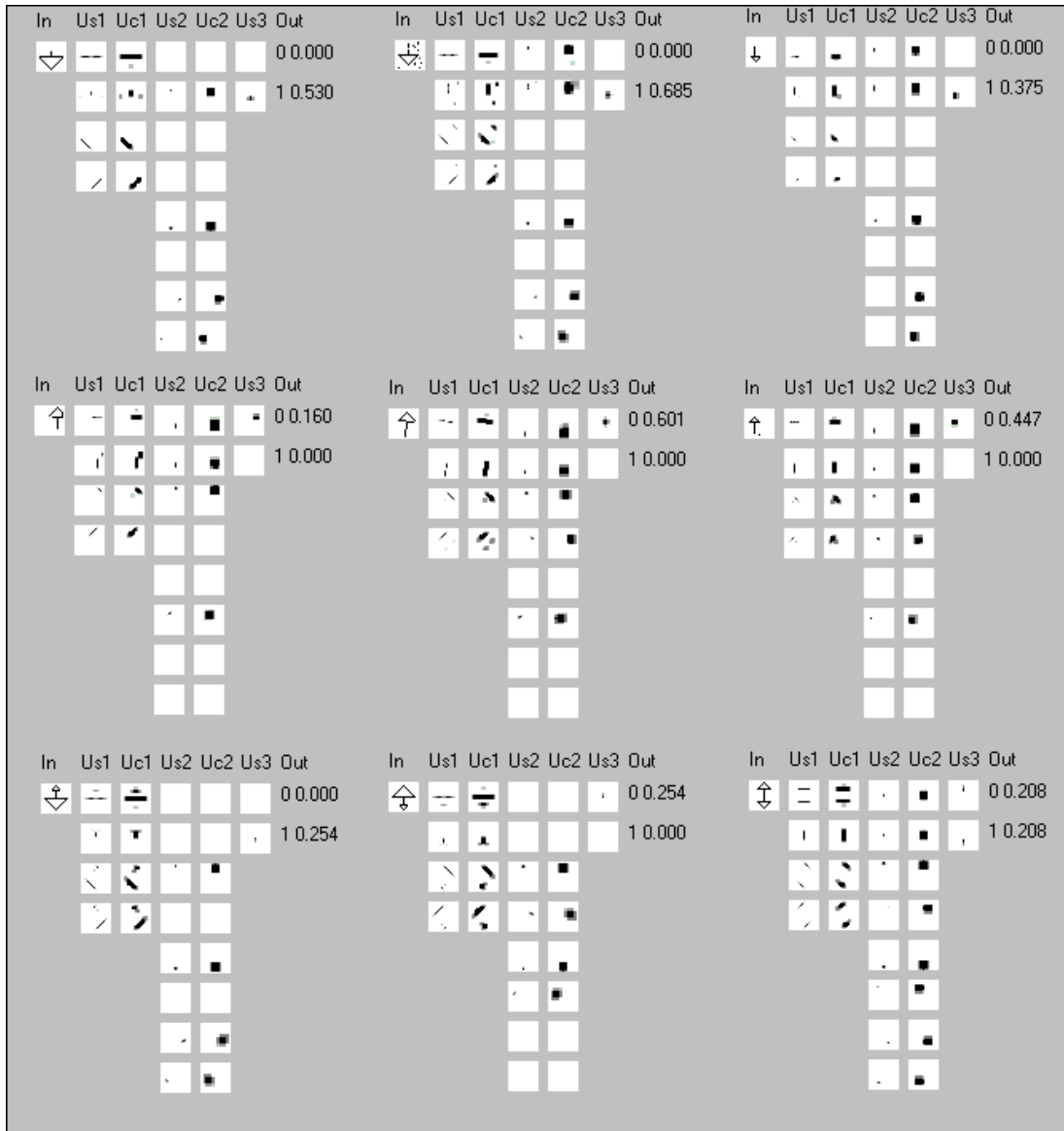


Fig. 2.18 Network response on different 'arrow' image samples. (Out 0 =up, Out 1=down)

## 2.3. Design and dimension of a network for handwritten character recognition

Fukisuma has proposed a 4-layer neocognitron network to classify handwritten characters. As stated in [3], it is difficult to show precisely how to choose the network configuration and how to scale parameters, however a few guidelines can be given. If the complexity of the patterns to be recognized is high, the size of the connectable area would be small, while the number of layers in the network would be high. In other words the more complex the patterns the finer individual cell tune in on features to be recognized and the ‘wider’ the network becomes. If a network has to recognize between a large number of classes the network becomes ‘higher’ that is the layers would have a larger number of planes.

Another design considerations are the robustness of the network against deformations and its generalization capabilities. These network characteristics are mainly depended on the size of the connectable area. Large sizes of the connectable area result in less robustness against deformations, small connectable areas serve good deformation tolerance.

As with the configuration of the network, it seemed also difficult to determine ‘good’ training-sets for the network. In the lower stages and the last stage, the trainingset images may look obvious. Determining the training set patterns for the hidden layers however is, using Fukisuma’s words, hard labor. We have the same experience. Although insight in the architecture of the network and the basic functionality is mandatory, one needs patience, some feeling and above all good-luck to find an optimal training-set pattern for the recognition problem which has to be resolved.

A formal description of Fukisuma original network is given in appendix 2.1. Below the main characteristics of Fukisuma’s network are given on the trainable planes in the S-sublayers.

Table 2.4 Fukisuma’s original network main characteristics

layer	#planes	Plane /Training size	Connect-able area	$\gamma$	Purpose
input	1	19*19	-	-	Receptable of input image for training or recognition.
1	12	19*19/ 3*3	3*3	0.9	Extraction of simple and small straight line components. Single pattern training.
2	80	21*21/ 9*9	5*5	0.9	Extracting local features of characters like corners, curvatures, intersections, line endpoints. Superimposed training on all planes.
3	97	13*13/ 19*19	5*5	0.9	Global feature extractor. In this layer local image features are combined to form complete characters or major characteristic parts of a specific characters. Superimposed training on each plane.
4	47	3*3/ 19*19	5*5	0.8	This layer is trained with typical samples of alphanumerical characters. For most alphanumerical characters there is defined more than one plane to reflect the different styles there exists for character ‘prints’. Superimposed training-set samples are used to train this plane on different sizes of the character.
output	35	-	3*3	1.0	Single cell planes which output excitations are used to finally classify the input sample .

## 2.4. Network recognition performance

In appendix 2.1, the specification and training-set is given of the network described in the previous section. This network has been used to verify our neocognitron simulator software implementation. In this section we will demonstrate that our implementation indeed does the things that Fukisuma has intended its recognizer should do.

Although we have tried to implement the neocognitron simulator in a structured manner the program eventually becomes seemingly large and complex. Both aspects are inherent to the nature of the neocognitron network. In order to let the software give reasonable response times, we have not always followed the rules according to which perfect OO-system should be build. The neocognitron output results given in this section will sufficiently sustain the claim that our neocognitron is implemented correctly. At least our neocognitron is implemented according to Fukisuma's original design.

In figure 2.19 an example is given of the response of the network when the input is supplied with a binary image of the character 'A'. First of all the network indeed recognizes the letter correctly but also the responses of individual cells in each plane compares correctly to Fukisuma[3, pp. 362 Fig 9.]

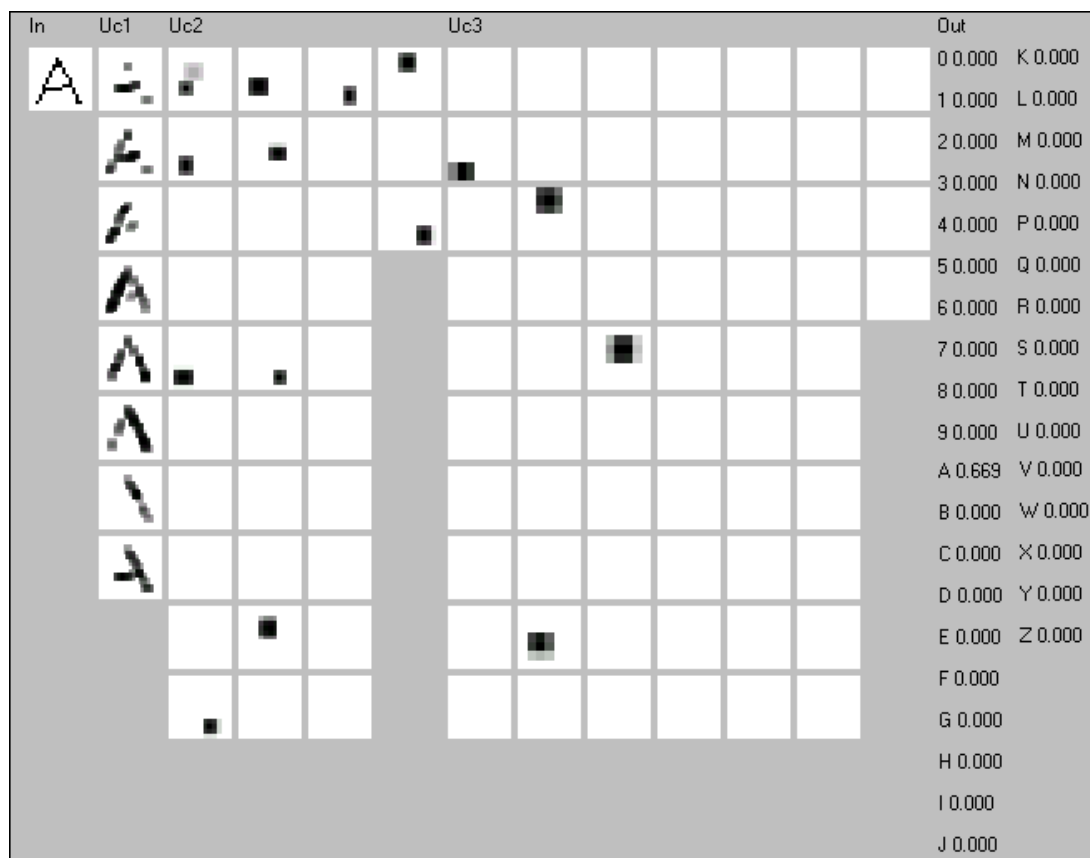


Fig. 2.19 An example of the response of the network in our neocognitron simulator. Only the C-cells are displayed. Note that the planes have different 'cell-resolutions' per layer

Figure 2.20 shows the example character images we supplied to the network to recognize. This sample compares to the input sample Fukisuma has also verified his network with Fukisuma[3, pp. 363 Fig.10] . The result of our verification test is shown in table 2.5. Obviously our network does not recognize all characters correctly. Only about 90% of the input sample images have been classified correctly.

At this point we must conclude that our network does not exactly implements Fukushima's original design since the result of our last test contradicts with Fukisuma's claim that all these samples are correctly recognized. Extensive code reviews and debugging of our s/w implementation does not reveal any significant mismatches with Fukisuma's architecture and design of the network. There are however four aspects that may clarify the different recognition rates.

1. In our implementation, we cannot have different selectivity values of individual planes in a sublayer. All planes in a sublayer in our network have the same selectivity value assigned at initialization. Fukisuma's system is designed so that adjustment of the selectivity parameter can be done per plane. In most layers there was no apparent need for having different values however Fukisuma does specify slight different selectivity values for some S-cell planes in the second sublayer. Where we use 4.0 for all planes, Fukisuma uses 4.0 for all planes except some are assigned a value 3.8. From its publication [3] it remains unclear why different selectivity values are chosen within one layer.
2. It was observed that our training patterns used for layer-2 through layer 4 do not exactly match the training-set patterns Fukisuma has used. The reason for this mismatch is because we had to input the patterns manually, as read from a largely magnified photocopy of a print of Fukisuma's paper. During this labor-intensive activity, visual misinterpretations and typing errors have caused our training-sets to be slightly different from the training-set Fukisuma had intended.
3. The input examples as depicted in figure 2.8 do not exactly match Fukisuma's example set. Our sample files are created by optically scanning the print of the paper and processing the bitmap file of the scan with a paint-program to adjust scale and color values. It was not possible to correct all deviations in the scanned image.
4. There may be another implementation aspect that could clarify slightly different recognized values. We have implemented real-numerical values as float datatypes. If, instead we had used the double type for floating point numbers, the output values of the trained network could have been just be a little different because of rounding errors in the numerous successively floating point calculations during training and using of the network.



Fig. 2.20 Some examples of deformed input patterns with which the neocognitron has been verified against

Table 2.5 Results of recognition of the “handwritten characters” from figure 2.20 using the original Fukisuma network

Character	Column 1/2	Column 3/4	Column 5/6	Column 7/8	Recognised Correctly
0	0CS6Q	0SC69	0CS69	0C6S0	100%
1	1J400	14000	1I000	1J4U0	100%
2	2S000	2Z000	2QS00	2S000	100%
3	30000	38000	?????	30C00	75%
4	40000	4L000	40000	40000	100%
5	5FE00	FE000	5DFE0	50000	75%
6	S6000	60S00	?????	60S00	50%
7	7T000	70000	7T000	7T000	100%
8	80000	BQ800	80000	80000	75%
9	9QSC0	9Q000	9Q0C0	9Q000	100%
A	AM400	AM000	A0000	A0000	100%
B	B0000	PRD00	DE000	D0000	25%
C	C0000	CQS90	CG000	C0000	100%
D	DPSR0	DPRS0	D0000	DPR00	100%
E	EF000	EFL00	E0000	EFL00	100%
F	FE000	FE000	FE500	FE000	100%
G	QGC09	GQC09	CG000	GC000	50%
H	H4100	41H00	H4100	4H000	50%
I	I1000	I0000	I1000	I1000	100%
J	J1000	VNW00	J4100	JU000	75%
K	K1400	K0000	K0000	K1000	100%
L	L1E00	LE000	LE000	L0000	100%
M	MA400	M0000	M0000	M1000	100%
N	NVW00	NV100	N0000	N4000	100%
P	PRD00	PRD00	P0000	PRD00	100%
Q	Q0000	Q9S00	Q90CS	Q9C0S	100%
R	RPDFB	RP000	RPD00	RP000	100%
S	S06D0	S0000	S6D00	S0000	100%
T	T0000	T0000	T0000	TF000	100%
U	U1400	U0000	U1000	UJ000	100%
V	VWN00	VWN00	VNW00	VWN00	100%
W	WVN00	WVN00	NWV00	WVN00	75%
X	XK000	X0000	X0000	XK000	100%
Y	Y0000	Y0000	Y0000	Y0000	100%
Z	Z2F00	Z0000	Z0000	Z2000	100%
<b>Recognised Correctly</b>	94%	86%	86%	94%	90%

Finally, a brief remark on response times performance. The time to train the network as described above is about 30 minutes on an ordinary Personal Computer. As stated earlier the network was trained the supervised way only. The CPU-time that is needed for the network to train only depends on the number of layers, the number of planes and the size of the connectable area's. Unlike a MLP training times are highly predictable. The time needed for the network to recognize an input image sample is about 2 seconds. Although these response times do not compare to response times we would measure when using a MLP network, there is no reason not to use a neocognitron in a CLPR system when only running times are considered.

## 2.5. Recognizing printed characters

Obviously Fukisuma had not hired a person specialized in calligraphy to produce the input sample as shown in figure 2.20. In general, handwritings may be classified as a generalization of a formally defined character font. One should expect the neocognitron defined in by Fukisuma[3] will have comparable recognition rates for characters produced by beautiful and perfect handwriting. We did not hire a calligrapher either but instead we supplied the network with printed characters from a predefined font. Table 2.6 shows the results of these tests.

Table 2.6 Percentage of characters recognized correctly using the original Fukisuma training-set.

Font/Style	22pt	20pt	18pt	16pt	14pt	12pt	10pt	8pt	average
Arial	-	62	65	62	71	88	65	62	
<b>Arial Bold</b>	-	47	71	68	76	62	44	29	
Veranda	-	59	65	65	62	88	53	62	
<b>Veranda Bold</b>	-	50	44	59	41	62	50	38	
Eurostile	-	47	59	65	56	50	50	41	
<b>Eurostile Bold</b>	-	56	56	53	56	44	35	26	
License Plate	52	-	52	52	56	44	26	-	
<b>Total</b>	52	53	59	60	59	62	46	43	54

The table 2.6 is summarized from the results as presented in appendix 3.1 (table 3.1.1. till 3.1.7). The **Arial**, **Veranda** and **EuroStile** character font samples are created by a paint-program while the license plate font characters are reproduced from 27 original high resolution digital photo images taken from Dutch license plates. It may be clear that the original Fukisuma's network configuration cannot be used in our CLPR system due to poor recognition performance.

A quick review of table 3.1.8 in appendix 3.1 however reveals that the spread in recognition rate per character is high; some characters are recognized with acceptable high rate's e.g. 'E', 'F', 'L' and 'T' and some are recognized with dramatic low rates e.g. '3', 'X' and 'B'. It is expected that a refined training-set would increase the performance of recognition of printed characters.

In chapter 4 we will clarify the poor results described above and explain our procedure followed to increase the performance of a neocognitron recognizing license plate font characters up to 95 %.

Fukisuma must have had foresight not to hire a calligrapher, since those people often beautify their characters with serifs. Just out of curiosity, we have tested our network supplying it with a serif font like TimesRoman 14pt. The recognition rate found at this test was very poor: about 30 %! Below the input samples used in our recognition system representing the Dutch license plate character font.



Fig. 2.21 Reproduced Dutch license plates characters from original high resolution digital photo images

Intentionally left blank



## **Part II**

Intentionally left blank

## Chapter 3

# The architecture for a license plate character recognizer

There is no such thing as a universal car license plate recognition (CLPR) system. If we recall, figure 1.0. in chapter 1. each step in the image understanding process must be tailored to the input image and to the object characteristics we are looking for.

- First of all each CLPR application will be equipped with a specific *image acquisition system*; camera types may vary, camera position with respect to distance and direction from which the image is photographed, image resolution and illumination conditions will be different.
- The required transformations in the *image preprocessing* steps will be depended on the quality of the input image and the size in pixels of the character objects to segment.
- The implementation of the *segmentation step* will be depended on character features and the number of characters to segment.
- Although the *character recognition* is implemented using neural computing, it is quite different whether to recognize a Japanese symbol string of variant length or a fixed length Dutch license plate character string with a predefined font.
- Finally, the *interpretation step* may vary due to the syntax of the license plate text.

In this chapter the overall design of our CLPR is given. Starting point in the design is to construct the system with a minimum of assumptions regarding its input. Our system is invariant to image size, license plate dimensions, license plate color and location. Figure 3.1 depicts the context diagram of our character recognition system.

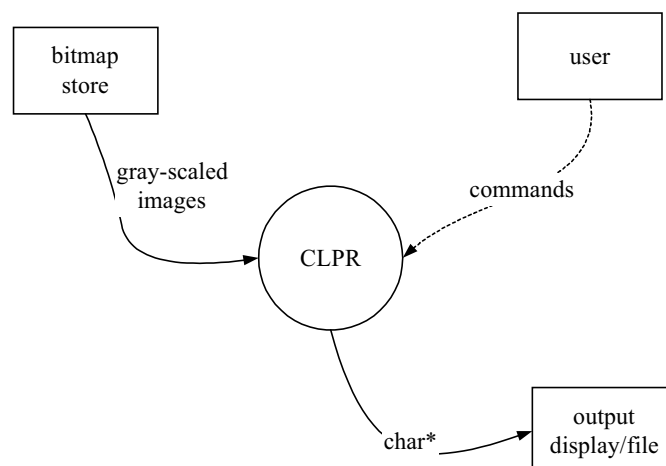


Fig. 3.1 Top level diagram of the CLPR

## 3.1. Design background

### 3.1.1. The license plate paradigm

In the real world, it may be observed that:

1. The color of the border of a numberplate is different from the object it is attached to (either the cars chassis or the cars bumpers).
2. The background color of license plate and the color of the characters reproduced on it are different.

The latter may be trivial, even for humans it would be impossible to discriminate characters printed in the same color as their background. Practically, the first observation would not always be true. However, even if numberplates don't have a physical border or different color from its background it is attached to, a difference in intensity levels between the physical number plate and its background on the picture would always exist.

Since numberplates are made of specific retroflex material with different reflection components as their background one may conclude that intensity levels will always exist (recall the image definition equation 2.3 in section 2.1).

Figure 3.2.a-d. shows some typical examples of either images that confirm the above definition of numberplates. The numberplate definition above will be the starting point for the image processor module design.

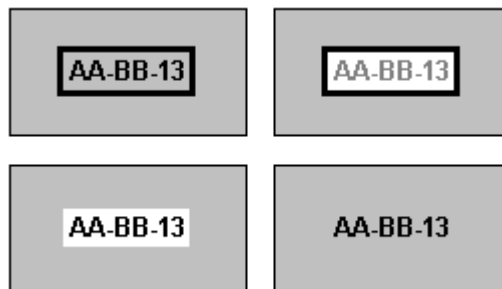


Fig. 3.2a-c number plates Fig. 3.2d (super)inscription

### 3.1.2. Implementation limitations.

The system is designed as a prototype CLPR. The main purpose of the system's implementation is to demonstrate different techniques on image processing and pattern recognition in software. There is no formal specification given on the CLPR system however, the starting points, which our design are based upon, are listed below:

- Without losing generality the input is restricted to series of 8-bit grey-scaled images of less than 640\*480 pixel format. Any picture with different specifications needs to be reformatted.
- The systems userinterface will run on a standard PC under Windows98/NT.
- Although the image analyzer, segment analyzer, and character recognizer are designed O/S independent, in our CLPR system these modules will also run a PC under Windows98/NT.
- The minimum size of characters recognized would be 10 pixels height.

In the design of our CLPR, reusability of code is accounted for as long as it will not affect the time needed to implement certain modules too much. The neural network software implementation is generated by either the SNNS workbench [9] (for MLP usage) or our refactored Sun/N-cube C/C++ version of the neocognitron implementation [12] is used.

### 3.2. System concept

The system should be invariant to the position of the license plate on the image, the color of the license plate background, the color of the characters on the plate and the dimensions of the plate. Of course there are input resolution restrictions; recognizing characters of dirty or vague plate with too small size, will surely fail whatever techniques used.

The basic procedure to automatically read numberplate characters from a digital image is performed in the following steps:

1. Binarize the image such that at least the individual characters form connected segments.
2. Successively take all connected segments produced in the previous step and determine whether it could be a character by applying dimension checks.
3. Apply a character recognition procedure on all candidate segments.
4. Determine whether the recognized characters in step 3 form a valid character numberplate string.

In the following sections of this chapter, it will be discussed how to overcome the most trivial problems that surely will be encountered when implementing the procedure above:

- Which threshold has to be applied to binarize the image correctly ?
- How to keep the number of segments in step 2 small in order limit processing time?
- Will the recognizer accept all real characters from the segments ?

In figure 3.3 the DFD of our CLPR is given depicting the main processor modules and the main data.

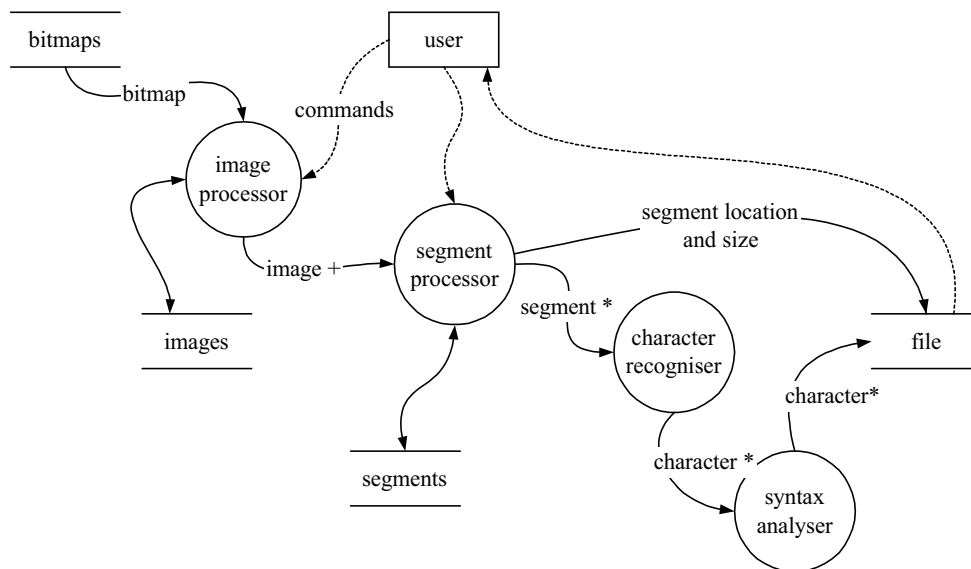


Fig. 3.3 The dataflow diagram the CLPR

### 3.3 The image processor

The problem to be solved by our image analysis module is finding all license plate characters without considering license plate properties. We have used a Belgium numberplate as an example in the text of section 3.3 and section 3.4 to show that the proposed image processor mechanisms and concepts indeed remains independent on size, location and color of the numberplate.

The main function of the image processor is to binarise the image such that the intended 'black-white' relation of the numberplate background and the numberplate characters is preserved. The image processor also should remove all possible non-character areas on the binarized image.

As stated in the 'license plate paradigm' section 3.1.1, the pixels belonging to the prints of license plate characters have different intensity values as the pixels belonging to the background area of the characters. By applying a binarization function over the total image we divide the image in two parts. One part does hold only regions of white pixels connected; the other part does hold regions with only black connected pixels (refer figure 1.5). Since we do not know in what color a license plate background and its characters appear, we need to apply a local binarization method.

All black pixel-connected regions are candidate license plate characters. However, as shown in figure 1.5 there may be too many of these areas. The 'license plate paradigm' also states that there always would be some kind of boundary around license plates characters. If we implement the local binarization function as kind of etching filter, this boundary would also be an area of black connected pixels. Figure 3.4 shows the original picture in which a part of it is locally binarized. Obvious we see the characters as connected black pixel regions as well as a boundary enclosing the license plate characters.



Fig. 3.4 A cutout of an original 640\*480 8-bit bitmap image

If we only select the regions of black pixels connected within an enclosed area of black connected pixels we would have all license plate character segments isolated from the original picture. The main trick is how to dimension the etching filter such that characters are formed as connected black pixel regions and the license plate characters are enclosed by a black-pixel border. The implementation of this function is given in section 5.2.3.

Once we have defined the filter, isolating the characters from the original image is not a problem any more. Finally the image processor should have the following basic image transformation functions:

- Contrast stretching (to remove large equal intensity areas on the image).
- Low pas filtering (to blur the image and make is less noisy).
- Local binarization (to highlight license plate characters and boundaries).
- Area filling functions (to remove all segments which do not lie within an enclosed area).

### 3.4 The segment processor

Figure 3.5 shows a part of the image after the area filling procedures have been applied on the binarized section of figure 3.4. As shown the number of segments have been decreased significantly. The segment processor should take all connected black-pixel regions; determine the rectangular area of the original location, catch, and cutout the segment from the original image. A procedure to do so is

described in section 1.4.2. As shown in figure 3.5 we enlarge the connected region one pixels on all sides. As described later in this section this boundary extension will be very helpful in processing the segment.

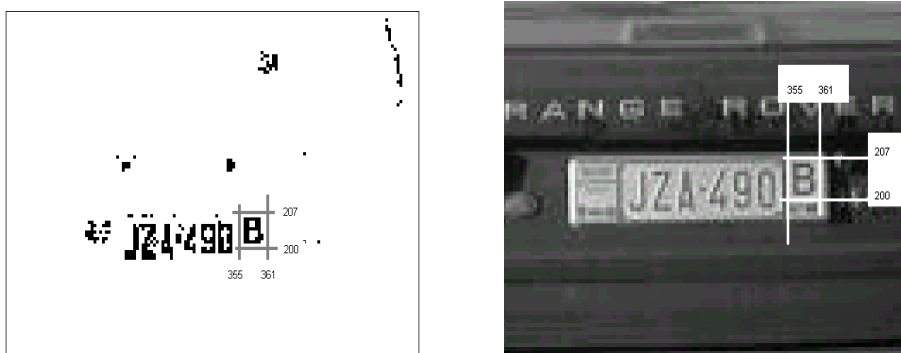


Fig. 3.5 The work of the segment analyser

Notice that the total number of segments in figure 3.5 is still relatively high compared to the number of segments that actually form the characters of the license plate. Moreover, we have depicted only a small part of the analyzed image. The segment processor should remove or ignore all segments that could not be considered character images. Before the isolated segment is send to the recognizer module a validation procedure should be applied. The segment processor should perform one other function. Since we select rectangle areas form the original picture we should binarized the segment again. The procedure to binarize an isolated segment taken form the original image is as follows:

- Determine the average pixel intensity values of all boundary pixels of the segment.
- Determine the standard deviation of the pixel intensity values set.
- Assign the average pixel intensity value - 2\*standard deviation to the treshold variable.
- Turn all pixels white that have intensity values higher or equal then the treshold.
- Turn all pixels black that have intensity values less then the treshold.

Figure 3.6 shows the original segment and the binarized segment after the procedure described above has been applied.

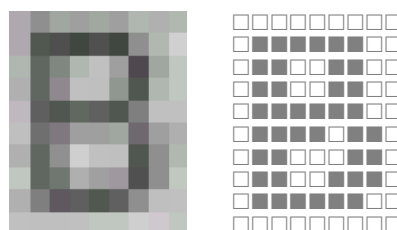


Fig. 3.6 The segment binarizer

As stated above the segment analyzer also has the task to remove or ignore all segments that do not depict a character. The segment analyzer checks binarized segments like in figure 3.6 above against:

- Min and Max height in pixels. (adjustable by  $9 < \text{system constant} < 50$  )
- Min and Max width in pixels. (adjustable by  $1 < \text{system constant} < 40$  )
- Height/width ratio. (adjustable by a system constant  $> 1$ )
- Ratio of white/black pixels in the segment.

As stated in section 1.5 we have used a neocognitron type of network as character recognizer. Since the neocognitron only uses skeletonised image samples directly on a fixed size input plane, three other functions are defined for the segment processor module:

- Scaling the cutout segment towards the standard height (not necessary for the neocognitron).
- Copy/paste the cutout segment onto a fixed 19\*19 pixel plane.
- Thinning the segment (necessary for the neocognitron)

Rather than defining the algorithms here, which perform the functions listed above, in the text we illustrate only the workout of these functions in figure 3.7a and 3.7b. In 3.7a the original 9\*10 pixel cutout is scaled to a standard height segment of 17\*19 pixels. In 3.7b the binarized segment is pasted on a 19\*19 plane and successively skeletonized. The function definitions and implementations can be found in section 5.3.

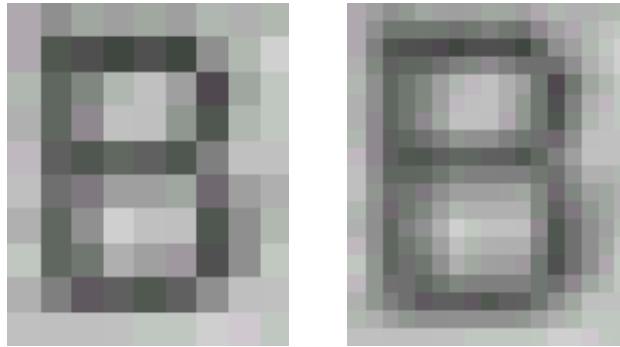


Fig. 3.7a The scaling operation on segments to a standard height

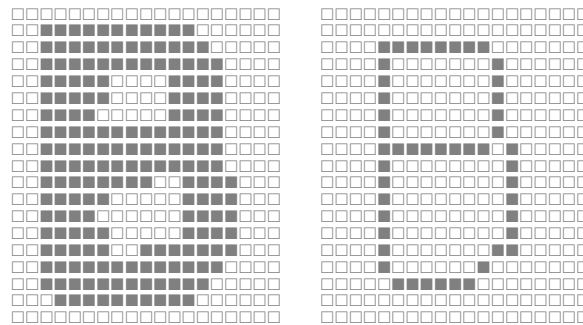


Fig. 3.7b The copy/paste and the thinning operation on an isolated segment

### 3.5 Character recogniser

The main character recognizer module in our CLPR is a neocognitron neural network. The architecture of our CLPR system provides for any type of neural net as long as it will use binary image inputs directly. The character recognizer takes one segment at the time as input, and outputs a series of character values on each segment. These character values are sorted in the order of the neuron's output excitation. For example a character 'B' may be classified as "B-0.921" and "8-0.624". These values are taken directly from the output neuron of the classifier. The output of the neuron trained to classify a B has value 0.921 in this case, and the output of the neuron trained to classify an 8 has value 0.624 in this case. Note that the output values are only relative numbers.

It is for the character recognizer to find out which segments are actual characters and which segments should be considered isolated erroneously from the image. A post processor should remove all faulty recognized segments. For post-processing purposes the character recognizer will not only output the mostly likely character class per segment but also output the actual size and location of the segment in the original picture. Unclassified characters are assigned a '?' value. These segments may be considered rejected characters. After all segments on the image have been processed by the recognizer, an output formatter will sort the segments such that the sequence in the output string would reflect the same sequence as characters on the actual plate. The dataflow diagram of the character recognizer is depicted in figure 3.8.



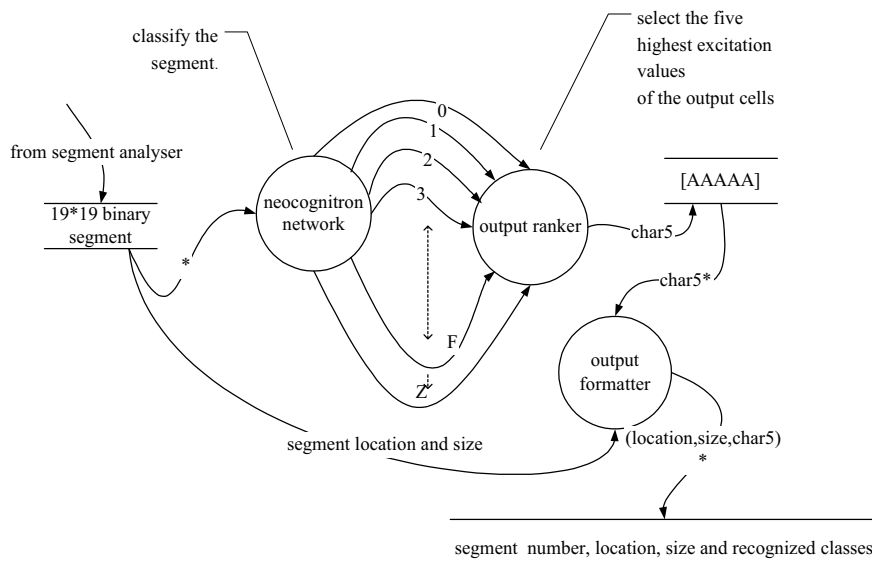


Fig. 3.8 The recogniser dataflow

### 3.6 CLPR blue print

Figure 3.9 shows the conceptual software model of our CLPR system. Within each building block, the main functions required are given in the bullet lists. If this blueprint of our CLPR is compared against the model depicted in figure 1.0, we will find the image acquisition, feature extraction, and syntax analysis building blocks missing. The reasons for this are:

- The images to be processed are considered to be available from a datastore.
- Feature extraction function seemed not be necessary in our setup.
- Syntax analysis is left undefined.

As will be described in chapter 5, the building blocks in figure 3.9 are implemented as physically separated software modules except for the (sub)blocks 'bitmap xformer', 'segment catcher' and 'output formatter'. In order to make the CLPR software platform independent we have selected to incorporate the 'bitmap xformer', 'segment catcher', and 'output formatter' in the user interface. The bitmap xformer translates a MFC bitmap structure into a simpler format that is more easily to manipulate. The segment catcher and the output formatter heavily rely on the GUI since both building blocks are using display windows as output device during processing.

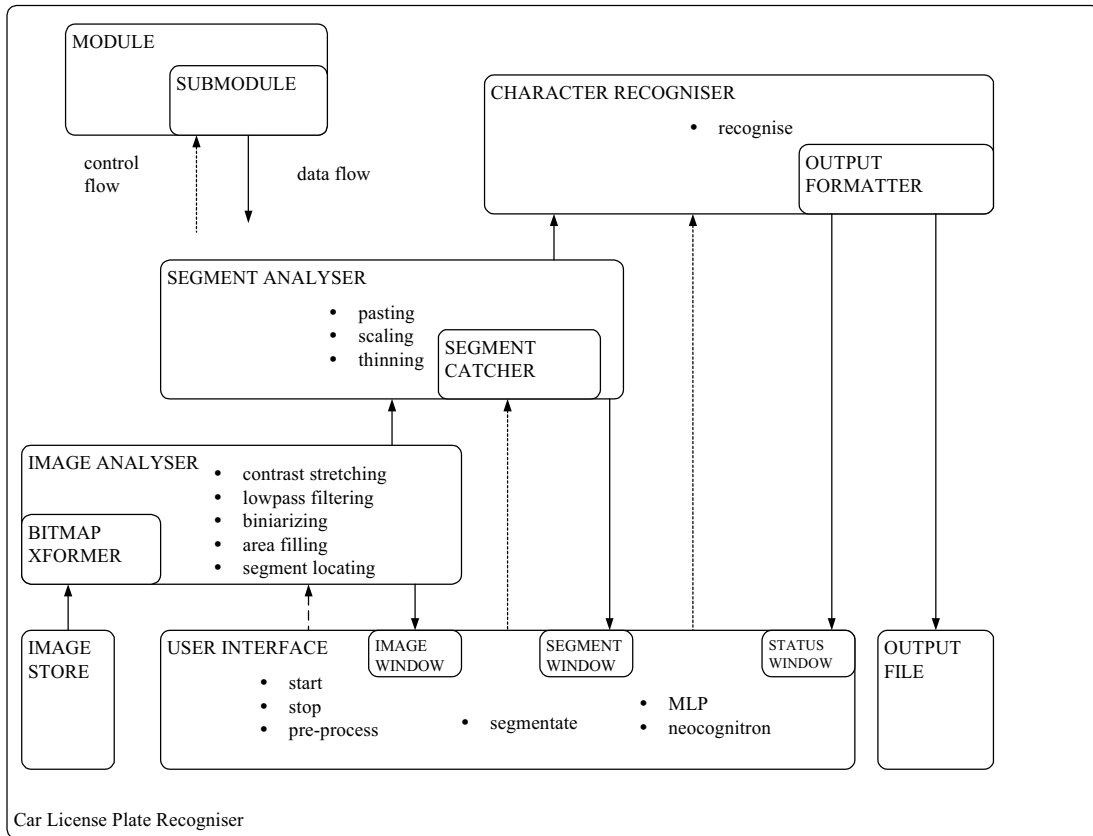


Fig. 3.9 The CLPR blueprint

## Chapter 4

# Finding an optimal neocognitron configuration to recognise characters of car license plates

## 4.1 Reviewing the results using Fukisuma's original network.

The results of the verification of our software implementation of Fukisuma's neocognitron as presented in chapter 2 do not harm his claim that the neocognitron proposed in [3] is capable of deformation-, location-, and size-invariant visual character recognition. However, it sounds contradictory that this specific network designed to recognize handwritten characters performs so poorly on printed characters. Table 4.1 illustrates this poor performance on a number of characters in the Eurostile font that were used in our earlier tests (see also appendix 3.1, table 3.1.5).

Table 4.1 Typical Eurostile font characters

EuroStile font	O	2	B	M	R	S	W
Recognition rate	29%	29%	14%	0 %	14%	14%	29%

The characters shown in table 4.1 are not randomly selected from the result tables in the appendix 3.1. We took these characters with the intention to clarify the fact they are recognized with such a poor performance. If we compare the characteristic layout of these input sample characters to the training-sets Fukisuma's proposed for the third and fourth layer we find noticeable differences:

- The '0' has more the form of a square rather than that of a circle or ellipse.
- Notice the perpendicular line segment crossing in the lower left corner of the character '2'.
- Both humps of the letter 'B' have the same width each.
- The three 'legs' of letter 'M' do end at the same bottom line.
- The normally oblique left little leg of character 'R' is parallel to the line segment on right side of the character.
- Notice the amidships part of the letter 'S' which is a pure horizontal line segment.
- Finally, letter 'W' which has also three legs of the same length.

The training-set patterns of all layers used in Fukisuma's original network are depicted in the appendix 2.1. Figure 4.1 illustrates our findings described above by showing the specific training character images used to train layer 4. Obviously, these elements of the training-set and the characters listed in table 4.1 have different styles.

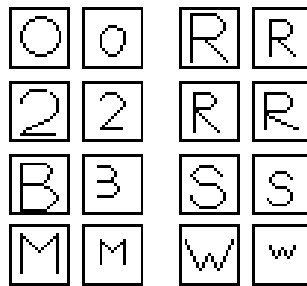


Fig. 4.1 Layer 4 training-set images on the characters of table 4.1

Of course, the fourth layer does not solely determine the recognition performance on these characters. The training-set patterns of previous layers may be even more important. However, we found out that if you do not fix these mismatches between the character style and the training-set style, it is very difficult to increase recognition performance.

Things can get even worse with respect to character styles. The Fukisuma network used is designed only to accept character patterns consisting of line segments of one pixel width. All input character images should therefore be 'thinned' before they can be fed to the neocognitron's input plane. Thinning the input character images is not the problem, however thinning will deform the characters to be recognized even more. The 'character style problem' described above gets even worse due to thinning.

The only way to solve this problem is to refine the training-sets and adapt them to the styles we could expect of the skeletons of printed characters. Figure 4.2 illustrates the deformation and/or style changes that occur when samples of the license plate font characters are skeletonized. Do note the high deformation caused by thinning on the letters 'K' and 'X' in the first series of characters in figure 4.2. The thinning algorithm used can even deform the characters such that it makes it even very hard for humans to interpret the image correctly. Note for example the letters '3', 'N', and 'T' in the second series of characters in figure 4.2.

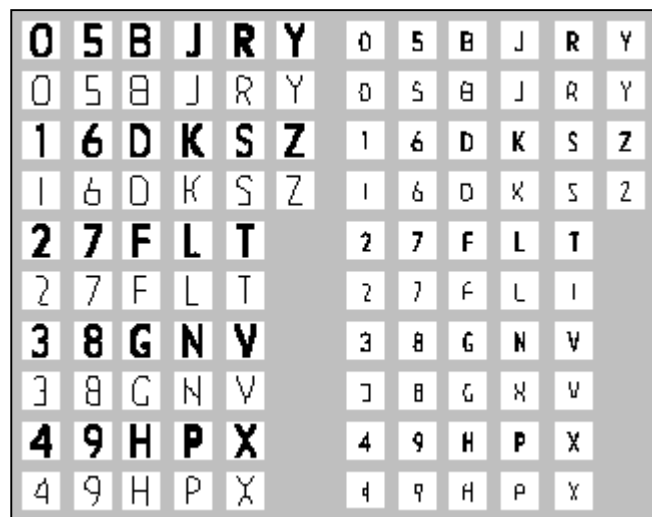


Fig. 4.2 Thinned characters of the license plate sample of figure 2.9. on a 19\*19 pixel raster. Only 18pt and 10pt characters are shown

## 4.2 Redefining training patterns

### 4.2.1 Refining the existing training set

In the previous section we demonstrated that the only way to improve the performance on printed characters is to refine the training-set and adapt them to the style we could expect on skeletons of characters that should be recognized.

- First of all we passed all image samples as reproduced from Dutch license plates characters from high resolution digital photo images (see figure 2.9) through our thinning algorithm.
- Then we studied the skeletons we got.
- Finally we tried to adapt the training-set for layer 3 and layer 4 of the network such that the network will become invariant for the deformations as shown in figure 4.2 on certain characters.

We changed those training patterns in layers 3 and layers 4 of which we think they contribute most to recognize the ‘3’, ‘N’, ‘X’ and ‘K’. Furthermore, we made some minor changes to several trainingset patterns on characters whose style will not ever appear on numberplates. For example, the second style of ‘4’ and the letter ‘R’ as defined in figure 2.1.3 in appendix 2.1.

Figure 2.2.2 and 2.2.3 in the appendix show the new trainingset for layer 3 and 4. Note that we have not changed the training-set of layer 2, neither we have changed the network constants selectivity, nor the gamma or deltas.

How our new network performs on the same testset as used in chapter 2 is shown in table 4.2. On the average, we got a recognition rate improvement of about 3 % only. On license plate character samples, we got a 17 % increase on recognition rate, most likely this is because we focused on the training-set patterns of this font specifically. If we take only into account the license plate characters with a size greater then 10 pt we got an increase in performance of 20 % (71.2 versus 51.2). This 20 % improvement confirms that the refined trainingsset pattern has sorted some effect.

Table 4.2 Percentage of characters recognized correctly using a modified Fukisuma training-set network specified in appendix 2.2

Font/Style	22pt	20pt	18pt	16pt	14pt	12pt	10pt	8pt	total
Arial	-	64(62) <sup>2</sup>	70(65)	64(62)	61(71)	91(88)	67(65)	55(62)	
<b>Arial Bold</b>	-	55(47)	64(71)	64(68)	76(76)	67(62)	48(44)	24(29)	
Veranda	-	67(59)	52(65)	70(65)	58(62)	76(88)	61(53)	55(62)	
<b>Veranda Bold</b>	-	55(50)	52(44)	55(59)	52(41)	48(62)	55(50)	33(38)	
Eurostile	-	45(47)	45(59)	48(65)	52(56)	42(50)	45(50)	45(41)	
<b>Eurostile Bold</b>	-	45(56)	52(52)	55(53)	60(56)	51(44)	42(35)	33(26)	
License Plate	78(52)	-	70(52)	67(52)	74(56)	67(44)	33(26)	-	64(47)
<b>total</b>	78(52)	54(53)	57(59)	60(60)	61(59)	63(62)	50(46)	40(43)	57(54)

It may be clear that a network which exhibits a 64 % recognition rate on individual characters is absolutely not usable in a CLPR. When using this network the chance to read a Dutch license plate correctly would be less than 8 % (0.65  $\uparrow$  6).

### 4.2.2 Modifying the existing training set

Therefor we started a second round to improve the performance by modifying the network training-set patterns. Since it becomes clear that training-set patterns significantly determine the recognition rate

<sup>2</sup> In brackets the recognition rate of Fukisuma original network is given.

on certain character fonts, we decide that from now on we concentrate on gaining recognition rates improvements on license plate characters only. This may be justified because constructing an artificial network for recognizing license plate characters is our objective. Besides it would make the training-set smaller and the network configuration a fraction less complex computational wise.

We carried out the following modifications in our second try to get a better performing network on license plate characters:

- The network gets only 27 output neurons each specifically trained to recognize one character in the set: { 0,1,2,3,4,5,6,7,8,9,B,C,D,E,F,G,H,J,K,L,N,P,R,S,T,V,X,Y,Z }.
- We change the configuration of layer 3. Instead of having multiple S-planes connected to one C-plane we have connected all S-planes to exactly one C-plane in this configuration.<sup>3</sup>
- By studying results shown the table 3.1.8 en 3.2.8 in the appendix we have tried to improve the training-set by decreasing the misclassification rate on certain characters. See table 4.3. below.

Table 4.3 Highest misclassifications rates

	Misclassification rate	Relatively high rate to be misclassified as
<b>0</b>	66.7 %	Q or D
<b>6</b>	70.8 %	5
<b>8</b>	64.6 %	Q or B
<b>9</b>	64.6 %	Q
<b>B</b>	45,8 %	D or P
<b>C<sup>†</sup></b>	64.3 %	F
<b>G</b>	72.9 %	C
<b>J</b>	60.4 %	1
<b>K<sup>†</sup></b>	56.3 %	1
<b>N</b>	43,8 %	1
<b>R</b>	45.8 %	P or D
<b>V</b>	68.8 %	1
<b>W<sup>†</sup></b>	52.4 %	N
<b>Z</b>	43.8 %	2

<sup>†</sup>Characters will not appear on Dutch license plates.

After several iterations, we found a network trainingset that produces comparable results to the previous one. It seemed hard to improve the recognition performance using the configuration and basic training-set Fukisuma has proposed initially. We experienced that one can increase the recognition rate of some individual characters just by matching the expected input patterns to training-set patterns and studying the excitation of individual planes. However increasing the recognition rate of the numerical ‘0’ could easily lead to a significant drop of recognition rate of e.g characters like ‘P’, ‘B’ or ‘R’.

Since the neocognitron has so many connections it makes it almost impossible to trace why exactly there exist such an observed negative correlation in recognition rate of some characters. In total we tried about 4 different trainingset patterns and network configurations, all alike Fukisuma’s original one. There seems some maximum to exists in overall recognition rate of about 70%.

Table 4.4 Car license plate character recognition rates in % on different configurations of Fukisuma’s original network

Network	22pt	18pt	16pt	14pt	12pt	10pt	total
Fukisuma (refer section 2.5)	52	52	52	56	44	26	47
First attempt (refer section 4.2.1)	78	70	67	74	67	33	64
Second attempt (refer section 4.2.2)	78	66	63	70	70	40	65

<sup>3</sup> With this ‘type’ of layer one could better study which neurons have large excitations on exactly what specific input pattern

One other thing we experienced was that tuning, on more than one of the parameters like selectivity, gammas, or deltas at the time, could lead to very poor performance rates. The new network configuration suggested in this section is described in appendix 2.3

Table 4.5 Comparison of recognition rates of two different networks per individual character

Recognition Rate in %	Our network in section 4.2.1			Fukisuma's original network from Section 2.5		
	classified correctly	Mis-classified	Un-classified	classified correctly	Mis-classified	Un-classified
input						
<b>0</b>	23	67	10	25	62	13
<b>1</b>	31	40	30	19	54	27
<b>2*</b>	73	25	2	56	35	8
<b>3*</b>	56	19	25	6	29	65
<b>4*</b>	71	8	21	40	17	44
<b>5</b>	77	15	8	77	15	8
<b>6</b>	25	71	4	33	56	10
<b>7</b>	100	0.0	0.0	96	2	2
<b>8</b>	17	65	19	19	52	29
<b>9</b>	15	65	21	21	52	27
<b>A†</b>	41	43	17	62	17	21
<b>B</b>	44	46	10	10	65	25
<b>C</b>	36	64	0	71	24	5
<b>D*</b>	90	8	2	75	23	2
<b>E</b>	96	5	0	88	0	12
<b>F</b>	96	2	2	92	2	6
<b>G</b>	21	73	6	33	63	4
<b>H*</b>	94	6	0	67	29	4
<b>I</b>	-	-	-	7	93	0
<b>J</b>	38	60	2	38	33	29
<b>K†</b>	42	56	2	63	35	2
<b>L</b>	100	0	0	96	4	0
<b>M</b>	45	48	7	38	50	12
<b>N</b>	54	44	2	44	52	4
<b>P</b>	90	8	2	88	4	8
<b>R</b>	46	46	8	58	23	19
<b>S*</b>	54	42	4	25	54	21
<b>T</b>	90	10	0	92	8	0
<b>U</b>	60	40	0	91	9	0
<b>V†</b>	31	69	0	56	44	0
<b>W</b>	36	52	12	38	54	7
<b>X*</b>	58	31	10	31	56	13
<b>Y</b>	79	17	4	83	17	0
<b>Z</b>	56	44	0	60	38	2
total	57	36	7	53	35	13

\*significantly improved recognition rates

†significantly decreased recognition rates

Finally, we show in table 4.5 (derived from tables 3.1.8 and 3.2.8 in appendix 3 ). the overall recognition rate per character. The only conclusion we can draw so far is that in general we can only increase performance slightly, on a network considered to recognize different character fonts and sizes, simultaneously. We managed to do so on one specific font, the license plate character font, and also we managed to increase the recognition rate of certain characters (see the annotated characters in table 4.5) but overall performance remains poor. In the next section, we suggest a different network layout with more drastical changes to the network training-set patterns: a neocognitron network that shows significant better results. We refer to appendix section 3.1 and 3.2 specifically table 3.1.8 and table 3.2.8 for comprehensive data regarding the recognition behaviour of the networks described in this section.

### 4.3 Modifying layer and plane configuration

As stated earlier the neocognitron is capable of deformation-, location-, and size-invariant visual character recognition. However, do we all need these features ?

- The character images of license plates to be recognized are to some extent deformed but they are all of the same style.
- We could easily fix the location<sup>4</sup>.
- Adjusting the size of too small characters is not a problem in software.

In the previous section, we came to the conclusion that somehow the interconnections between certain planes are too tied after training. Since changing a training-set of a plane obviously mend as a keyplayer in recognizing the numeral '0' appeared to worsen the recognition rate of for example the character 'P'. The latest may be plausible but hard to proof and even more difficult to fix.

Taken all these facts into account we designed a neocognitron network with only 3 layers. The first two layers comparable to Fukisuma's configuration. Layer 1 serves as a line extractor, layer 2 is trained to detect parts of any alphanumerical characters. We added a number of planes to layer 2. The added planes are trained on parts of the characters '6', 'V', 'X' and 'Z'. These characters have been proven earlier to be recognized with difficulty.

Layer 3 is trained directly with character images instead of using this layer as combined feature extractor as Fukisuma does. Training a plane with more than one pattern increases its ability to extract deformed features. Since we do not expect our characters have too much deformations we train the plane in layer 3 with only one training sample per plane. The required robustness against deformation is set with the selectivity parameter of the planes in this layer. The formal description of this new network can be found in appendix 2.4.

The network described above has indeed lost some of it functional features with respect to size and location constraints but performs quite well on the test set containing the license plates characters we used in section 2.5 as shown in table 4.6. below.

At this point, we want to recall our previous results in table 4.4. If the recognition rate of our neocognitron is plotted against the size of the characters, we get the following graph figure 4.3.

Obviously, the network recognition performance starts to degrade significantly at characters less then 12 pixels height on each configuration based on the original Fukisuma's network. The upper line in the graph depicted in figure 4.3 shows the recognition rate of the neocognitron described in this section to drop significantly at about 15 pt's. This is very plausible since we do not explicitly train the network on the small characters anymore.

In order to increase the recognition rate performance of our network on all character sizes we have carried out two other experiments. The results are shown in table 4.7.

1. First of all we magnified all character images to a standard size of 19 pixels height.
2. Secondly, to get the best of both worlds, we magnified only the character images to 19 pixels that had a height equal or less than 15 pixels. As shown in table 4.7. we achieved a recognition rate on individual characters of out of our test set of about 90 %.

---

<sup>4</sup> In fact, they are already positioned at a fix location in the input plane of the neocognitron. Our segmentation routine implicitly centers the image on the input plane.



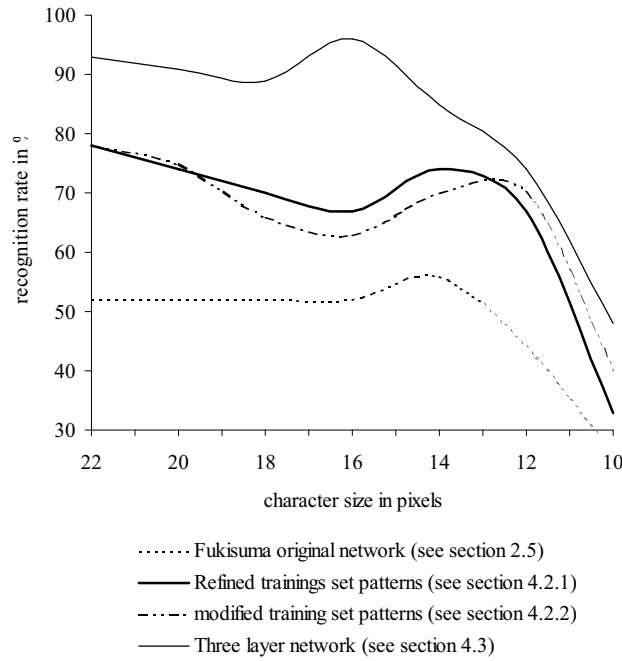


Fig. 4.3 Recognition rate versus characters size

Table 4.6 Recognition of standardised car license plate characters by the network suggested in section 4.3

Font Size Character	License Plate						Recognised Correctly
	22px	18px	16px	14px	12px	10px	
0	0DG83	0D8G0	0G830	0D900	?????	40000	67
1	1YVN0	1YVN0	1YVN0	T0000	10000	10000	83
2	2Z000	2Z000	2Z000	?????	2LZ00	?????	67
3	3J000	3J000	3J000	3J000	30000	70000	83
4	40000	40000	40000	40000	40000	40000	100
5	5B000	5S000	5S000	50000	50000	?????	83
6	60000	60000	60000	60000	?????	40000	67
7	7TZ00	7TZ00	70000	7T000	70000	70000	100
8	B8G30	B8D60	B8000	B9000	?????	?????	0
9	90000	90000	90000	94000	40000	?????	67
B	B8DP0	BD8PR	BDP80	B8P60	B8600	B0000	100
D	D0B8G	DB000	DB000	DB200	D0000	D0000	100
F	FP500	FP000	FP000	FP000	F0000	F0000	100
G	G0000	G0000	G0000	G0000	?????	?????	67
H	HKN00	HNKV0	HKN00	HK000	HK000	HK000	100
J	J3000	J3000	J3000	J0000	J0000	J0000	100
K	HK1N0	KN000	KNV00	KN000	KN000	K0000	83
L	L2000	L2000	L2000	L0000	L0000	L0000	100
N	NVKH0	NVHKX	NVKH0	NV000	NK000	NK000	100
P	PR000	PR000	PR000	PR000	PR000	?????	83
R	RP000	RP000	RP000	RP800	RP000	?????	83
S	S0000	SG000	S5000	S0000	50000	?????	67
T	T7000	10000	T0000	T0000	10000	10000	50
V	VNYX0	VNY1X	VNY00	VN000	VK000	?????	83
X	XY000	YX000	XYKNV	YXVNK	XY000	KY000	50
Y	YVX10	YVX10	YVXN0	YVXK0	YV000	YVK00	100
Z	Z2700	Z2700	Z2000	Z2000	Z2L00	Z2L00	100
<b>Recognised Correctly</b>	93	89	96	85	74	48	<b>81</b>

Table 4.7 Recognition rate after sizing the character image samples

Sizing the input image	22pt	18pt	16pt	14pt	12pt	10pt	total
Only when height or width > 19 pixels	93	89	96	85	74	48	81
All images sized to 19*19 pixels	93	89	81	85	85	85	86
Only size to 19*19 if >19 or <15 pixels	93	89	96	89	85	85	90

So far, we have only tested the network with samples produced from standardized images of individual characters. How this network performs on character images isolated from real photographs of cars on the road is given in the next section. A final remark on this section. If you look at the results on recognition of the numerical '8' in table 4.6. you will find either it is recognized as 'B' or it remains unrecognized. We will fix this problem in the next section.

## 4.4. A combined neocognitron network

### 4.4.1 Recognition performance increase measured on real-world photographs

Table 4.8. gives an overview on the recognition rates obtained using the various described networks on character segment images isolated from real photographs. As expected the table below shows a slightly lower recognition rate than the recognition rates observed in the previous tests with the various network configurations using a standardized character set. This is caused by the fact that the segments isolated from real photographs show more irregularities and noise than the segments isolated from the binary image shown in figure 2.9. The images referred to in table 4.8. are 640 \* 480 8 bit grey-scaled pictures produced by an Olympus CAMEDIA digital camera C-840L. In appendix IV find of the photographs used are presented.

Table 4.8 Recognition of car license plates using the various networks

ref	image	plate	char	Network configuration				
				#	ident	size in px	Fukisuma's network section 2.5	Network in section 4.2.1
1	004	<b>02-DR-GR</b>	12	Q?D9?9	Q2D9GQ	02D9G?	02DRGR	02DRGR
2	005	<b>02-DR-GR</b>	12	9LDR4P	02DRGP	02DRGP	02DRGR	02DRGR
3	006	<b>02-DR-GR</b>	11	CZD9?R	02D9FR	02D?FR	02DRGR	02DRGR
4	007	<b>02-DR-GR</b>	11	C2DR??	C2DR1X	02DR?X	02DRGR	02DRGR
5	008	<b>02-DR-GR</b>	13	02SRCR	02DRCR	02DRGR	02DRGR	02DRGR
6	009	<b>02-DR-GR</b>	13	0LD?CR	02D?GR	02D?GR	020RGR*	02DRGR
7	012	<b>02-DR-GR</b>	16	QLDR?R	Q2DRFR	02DXFR	02DRGR	02DRGR
8	013	<b>LS-BZ-92</b>	17	L5DL?2	LSDZ?2	LSBZ92	LSBZ92	LSBZ92
9	015	<b>PG-ZG-50</b>	14	RCL?E0	PGZF50	PGZF50	PGZG50	PGZG50
10	016	<b>HJ-XT-50</b>	10	H4KFE2	H1X?5Q	11XF50	HJXT50	HJXT50
11	021e	<b>71-FF-NN</b>	16	71FFNN	71FFNN	71FFNN	71FFNN	71FFNN
12	022	<b>71-FF-NN</b>	14	7K?FNN	7XFNN	71FFNN	7?FFNN	7?FFNN
13	023	<b>RS-ZF-23</b>	15	R?LF2?	RS2F23	RS2F23	RSZF23	RSZF23
14	024	<b>68-DB-BT</b>	14	??DDQT	6?DDDT	6B?DBT	6BDBBT*	6BDBBT
15	028	<b>PD-21-DJ</b>	12	RSZ1S1	PQ31D1	P?3101	P021DJ*	PD21DJ
16	029	<b>TF-TF-85</b>	12	TFF?D5	TFF?B?	TFFFD5	T??F?5	T??F?5
17	031e	<b>NX-NP-31</b>	9	4?P??1	1K1Y?1	1K?Y?1	NXNP21†	NXNP21
18	032	<b>VL-29-DN</b>	9	VLL?1?	KLL?1?	?LL11J	?LZ9DN†	?LZ9DN
19	035	<b>JS-PT-84</b>	9	4FP7A?	15P???	15P1??	JSPT84†	J5PT84
20	036	<b>VG-47-GD</b>	17	44474S	1GH7GD	1GH7G0	VG47G0*	VG47GD
21	037e	<b>LT-TL-53</b>	20	L?TL5?	LXTL53	LXFL53	LTTL53	LTTL53
22	038	<b>HP-BG-77</b>	15	49DC77	HQDG77	10BG77	HPBG77	HPBG77
23	039	<b>PX-HT-87</b>	17	PKHTQ7	PXHTQ7	PXHT07	PXHT07	PXHT07
24	040e	<b>VR-18-NK</b>	9	4PK1?	1PK1?K	??11?	VR18?K†	VR18?K
25	041	<b>70-VH-RT</b>	10	7QYHRT	7QYHRT	701HRF	70VHRT	70VHRT
26	043e	<b>HN-LX-85</b>	17	H1L4Q5	H1LX8S	HNLX85	HNLXBS*	HNLXB5
27	044	<b>JG-XG-98</b>	19	14??93	1GXFQ8	1GXF98	1GXG90	JGXG90
28	045e	<b>KZ-96-XR</b>	15	1296KR	1296XR	1296XR	KZ96XR	KZ96XR
29	046	<b>P-44-RD</b>	13	P??RD	P44RD	P4?RD	P44RD	P44RD
30	047	<b>VS-44-ZT</b>	12	V??ZF	K3??ZF	1S??ZF	YS44ZT	YS44ZT
31	048	<b>BB-PB-95</b>	14	D?SB95	BBPBQ5	BBPD95	BBRB95	BBRB95
32	049e	<b>BB-NH-08</b>	18	DBKH0?	P?KH0?	P?NH0B	BBNH08	BBNH08
33	050	<b>BG-ZG-21</b>	12	PCL??1	BCZG21	DGZG21	8GZG21*	8GZG21
34	051	<b>VX-RZ-10</b>	11	VV7?MQ	N1?ZMD	?1?Z1?	?XRZ10	?XRZ10
35	053	<b>ZF-54-FY</b>	14	LF5?FY	2F54FY	2F54FY	ZF54FY	ZF54FY
36	054	<b>50-VG-ST</b>	19	5SV45T	5SVGST	501G5T	50VGST	50VGST
37	055e	<b>JX-71-RT</b>	15	1111R7	1X11R7	1X11X7	JXH1RT	JXH1RT
38	056	<b>SJ-GN-95</b>	11	?W4ND5	51?NB5	51?N?5	SJGN9S*	SJGN95
39	057e	<b>HZ-LH-47</b>	12	4L1457	1Z11?7	N211?7	NZLH47	NZLH47
40	058	<b>XN-ZD-18</b>	13	?KL116	X1ZD16	XXZD10	XKZD1B	XKZD18
41	059	<b>PN-NG-01</b>	14	PN1C01	PN1G01	PN?G01	PNNG01	PNNG01
42	060	<b>RS-BJ-46</b>	13	R5D146	RSD146	RSD146	RSBJ46	RSBJ46
43	061	<b>XJ-PJ-42</b>	16	K1R142	X1RJ42	X1PJ42	X1PJ42*	XJPJ42
character recognition rate				38.91	60.70	63.81	89.88	92.61
plate recognition rate				2.33	4.65	11.63	53.49	65.12

\*0/D, 5/S, 8/B, 2/Z or 1/J problem

† Sizes less than 9 pixels

Although we have managed to increase the recognition rate of a binary character image from 39 %, using Fukisuma's original neocognitron, up to almost 90% using a 3 layer neocognitron trained for the license plate character font specifically, the overall recognition rate on plate level is still poor about 53%.

It should be noted that many of the unclassified characters marked as '?'s in table 4.8 are caused by shortcomings of the segmentation- or thinning processing procedures. However, even if the image pre-processing and segment enhancement routines would carry out the job perfectly, we would still

observe many character misclassifications. In figure 4.4, an example is given on how a character is deformed too much as a result of failure in the skeletonizing procedure.

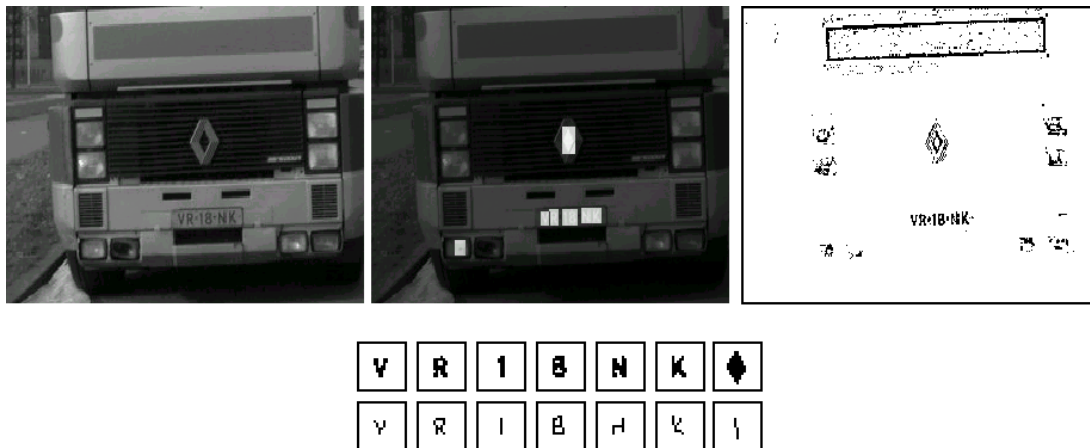


Fig. 4.4 Not all segments are processed correctly. The image left is a 300\*250 pixel cutout of original picture. The character size in this case +/- 9 pixels. See image ref 24 on table 4.8

#### 4.4.2 Combining neocognitron classifiers

As with many other character image recognizers we observe that some character images are hard to distinct from each other. Typical examples are the numerical '0' and the alpha character 'O' or the numerical '1' and the alpha character 'I'. To a lesser extend we observed misclassifications between the ambiguous characters 0/D, 2/Z, 5/S, 8/B and even 1/J. If we could eliminated all these errors in recognition, we could improve the recognition rate on character level up to 94 % and license plate recognition rate up to 67%.

When only considering Dutch license plates we could parse the recognized string, apply a syntax check, and correct the output according to certain rules. We show some typical examples in table 4.9 to illustrate this.

Table 4.9 Applying syntax rules to correct misclassifications by the neocognitron

ref	Image #	Plate ident	Recognised string	Corrected string
14	024	<b>68-DB-BT</b>	6BDBBT	68DBBT
15	028	<b>PD-21-DJ</b>	P021DJ	PD21DJ
26	043e	<b>HN-LX-85</b>	HNLXBS	HNLXBS or HNLX85

We suggest a different method however to correct for neocognitron misclassifications on the alphanumerical characters {0, 1, 2, 5, 8, B, D, J, S, Z}. In case the output of the neocognitron is one of the alphanumerical characters in the set mentioned above we let a dedicated neocognitron network confirm or correct the output of the 'main' neocognitron.

Why don't we improve the training of the main neocognitron such that these misclassifications are prevented to happen? This seemed very hard to model. If we try to tune the recognition rates of '0' and 'D' by adjusting the training-set patterns in layer 2 of the neocognitron (specified in section 2.4. of the appendix) we observe severe decrease in recognition rates on other alphanumerical characters like for example 8,R or P. This is because specific planes in layer 2 do not only determine the recognition of 0/D but also may contribute significantly to the recognition of the 8, R or P.

If we build specialized networks which are only used to discriminate between a '0' or 'D' we could freely adjust training-set samples in each plane of the lower layers without disturbing the behavior of

the main network. Which rules to apply exactly for correction and or confirmation in order to have the best overall performance, should be determined by verification testing. We have selected the architecture depicted in figure 4.5. Implementation of this model is not difficult; since our implementation is Object Oriented. We just instantiate a number of additional objects of the CNetwork class and add some if-statements to include the logic we suggest. What is left is defining the configuration and training-set patterns of the dedicated networks.

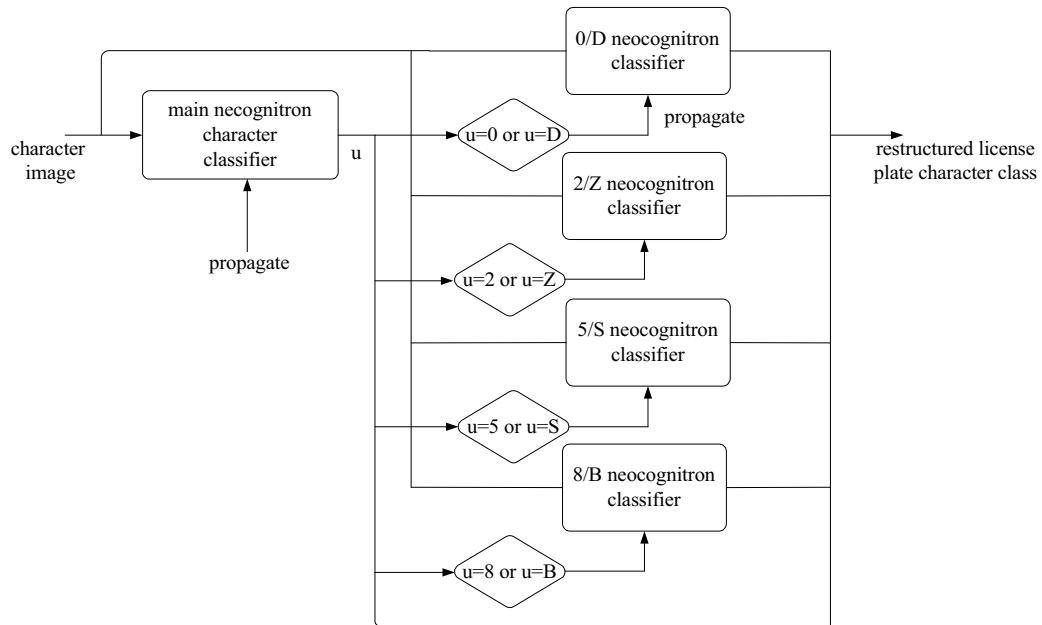


Fig. 4.5 Connected neocognitrons.  $u$  is the character class with the highest excitation value

In figure 4.6. the configuration and the training-set patterns of a network to discriminate between '0' and 'D' are given. This network serves only as an example. Building a network that has to discriminate between only two classes is much easier than building a network that should recognize 27 different classes. Mainly because we must only concentrate on a small number of specific characteristics of the characters to be recognized. Besides that, the dedicated neocognitrons are small, their training-set is easy to manipulate, and training times are short. All this makes the manual process of iteratively reconfiguring the network in order to find an optimal performing network apparently easier. The exact specification of this network can be found in appendix 2.5.1.

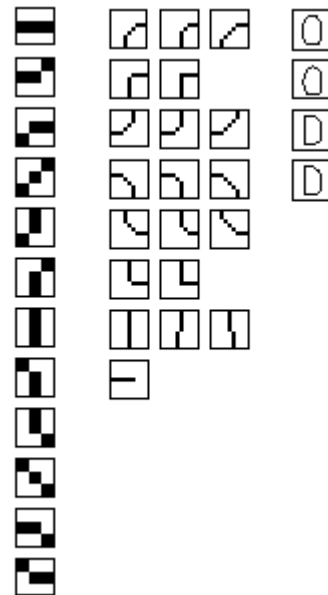
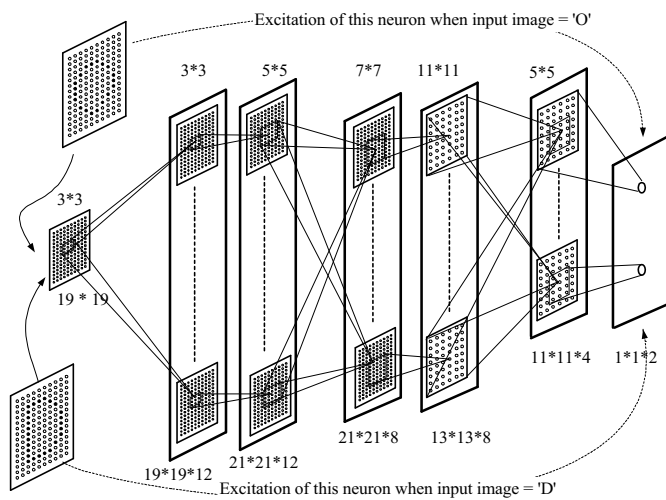


Fig. 4.6a An 0/D discriminator neocognitron network

Fig. 4.6b Corresponding training set

Does it work, the mini-neocognitrons? Below we will demonstrate that the specific '0/D' discriminator neocognitron does indeed correctly confirm any 'D' characters found on the numberplates of the cars in picture 28 and picture 36 of table 4.8.

In table 4.10 the five highest output cell excitations and their associated character classes of the main neocognitron are given for any 'D' character in both plates considered. Obviously the first 'D' in plate "PD-21-DJ" and the 'D' in plate "VG-47-GD" are misclassified as '0' by the main neocognitron.

Table 4.10 Recognition of '0' instead of 'D'

ref	image #	Plate Ident	Output main network	Network configuration as specified in section 4.3					
				Plate recognized as	Five highest values of excitation of the output planes				
15	28	<del>PD-21-DJ</del>	[0D8G-]	P021DJ	0	8	D	G	-
					0.711	0.445	0.493	0.067	0.000
15	28	<del>PD-21-DJ</del>	[D0B8G]	P021DJ	0	8	B	D	G
					0.583	0.223	0.250	0.784	0.093
20	36	<del>VG-47-GD</del>	[0DGB8]	VG47G0	0	8	B	D	G
					0.698	0.125	0.126	0.621	0.269

As listed in the table above, discrimination between 0 and D is a close race. In figure 4.7 however the specialized '0/D'-discriminator neocognitron network does pronounce its judgement on what characters is finally recognized.

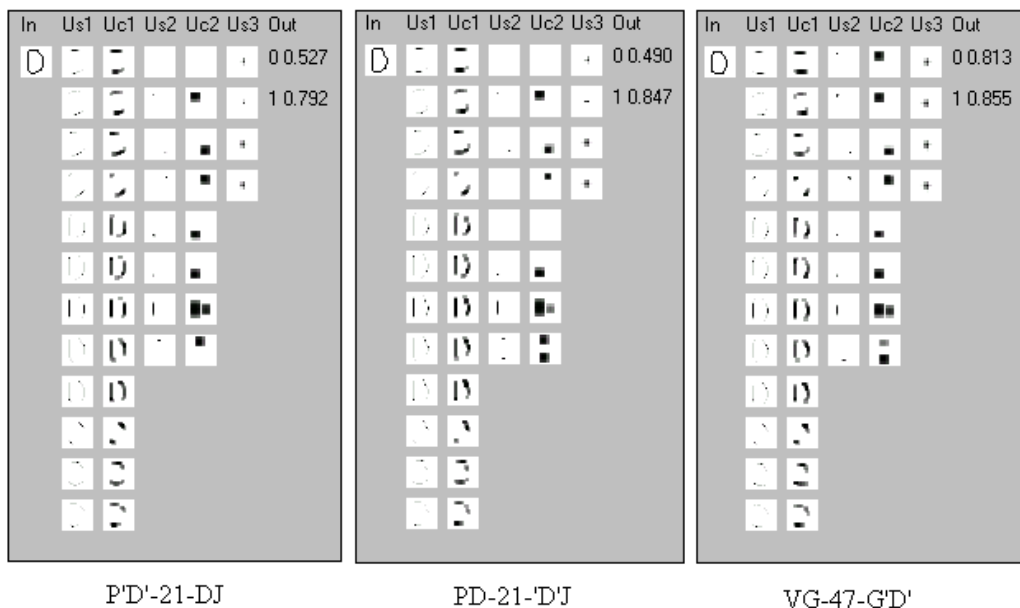


Fig. 4.7 Outputs of O/D classifier in the combined neocognitron network for the 'D' characters of the plates considered

According to the outputs of the neocognitron shown above; the first letter 'D' in plate PD-21-DJ should be classified as 'D' and not as '0', the second letter 'D' in the same plate is confirmed as 'D' obviously and finally the last character in plate 'VG-47-GD' is –succeeded by the skin of one teeth–classified as 'D'. Finally, the plates do read PD21DJ and VG47GD successively, which is correct after all.

In table 4.8 last column the recognition results are listed after we have corrected the output using mini neocognitron correctors for the character combinations O/D, 2/Z, 5/S, 8/B and 1/J. On our verification set of images listed in table 4.8 we got a plate recognition rate of 65 %. In chapter 6 the recognition results are given on a larger number of photographs taken on the TU-Delft parking lots.

Intentionally left blank



## **Part III**

Intentionally left blank

## Chapter 5

# Software implementation of the license plate recognizer

Preceding the description of the model of the software of the implemented prototype CLPR, figure 5.1 gives the overall block structure and figure 5.2 shows a typically setup of the current user interface.

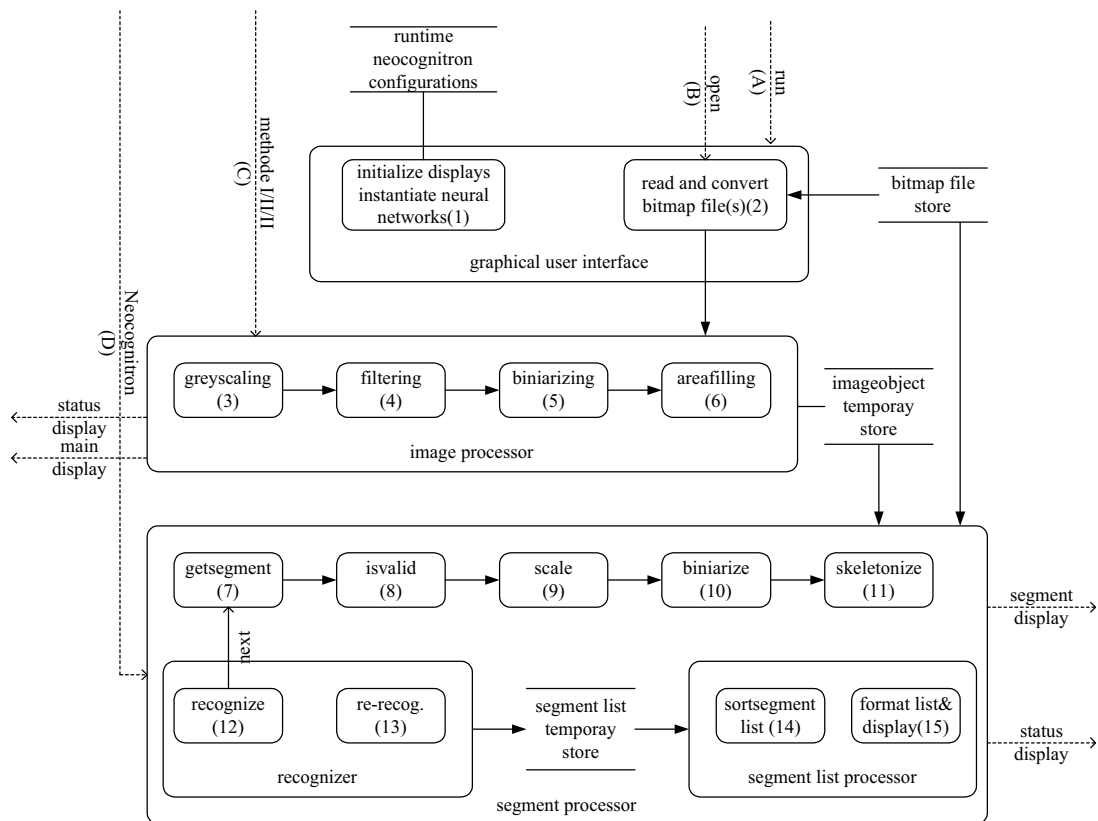


Fig. 5.1 The CLPR building blocks as implemented in software

A car license plate is recovered from an image by the following user-actions: A) the system is loaded, B) select and open a bitmap file C) initiate image pre-processing and finally D) trigger image segmentation and recognition. Steps B, C and D are preformed automatically when processing series of bitmap files. Internally the system executes the functions 1 through 15 successively as given in the block diagram above. Note that function 7 until 13 are repeated for each segment caught.

In the remainder of the chapter the overall software structure of the CLPR prototype is given and the descriptions of the main functions of the image processor, segment processor and recognizer are

given. The prototype CLPR is written in C++. All text in remainder of this chapter printed in the courier new font are to be considered implicit references to the source code.

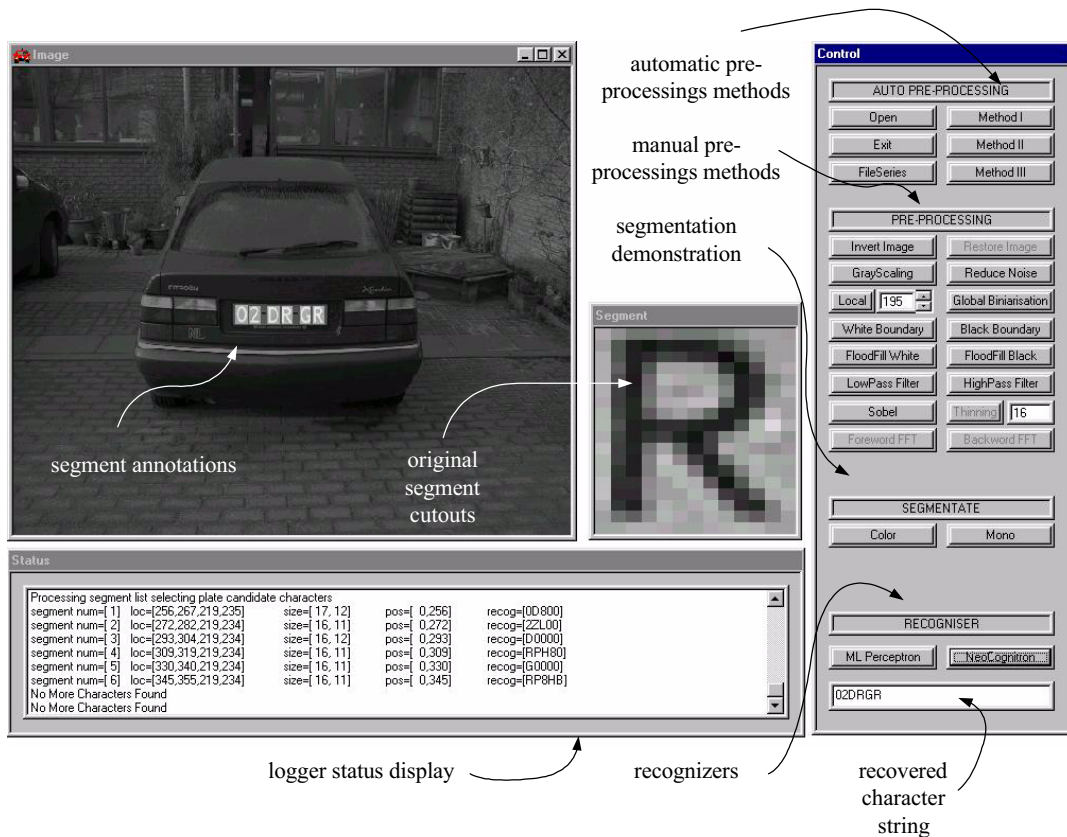


Fig. 5.2 The userinterface of the CLPR system prototype

## 5.1 Systems software components

### 5.1.1 Software layers

Figure 5.3 shows the layered structure according to which the software of the CLPR is constructed. The userinterface layer provides for some displays to paint images and segments, it provides for buttons to start/stop the application, to manually select individual image preprocessing functions and manually start segment recognition. The userinterface is named SysUI and is build using the MicroSoft Foundation Class library (MFC).

Above the userinterface, two object-view layers are available. The SysOV layer holds all declarations and definitions of the classes relevant for image processing and image segmentation. These classes are CImage, CSegment and CSegmentList. The NeoOV layer holds the class CNetwork and all its derived classes like CLayer and CPlane. Finally, a utility layer is used. The NeoUT layer holds some classes helpful for logging and debugging while the SysUT holds a collection of different functions which were not implemented in C++. Among others, a recursive filling function and a MLP character recognizer function. The NeoOV and NeoUI are described comprehensively in appendix I.

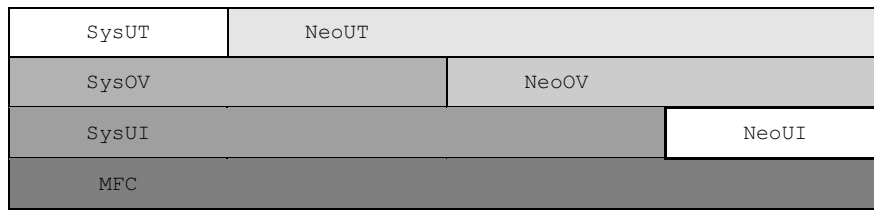


Fig. 5.3 The software layers of a license plate character recognizer

### 5.1.2 Software modules

The CLPR is build as one executable program called sysui.exe. The program is build according to figure 5.4 below. The two object views SysOv and NeoOv are available from a library and statically linked onto the executable. The same counts for both UT-libraries. MFC classes are linked dynamically using a MicroSoft DLL. The graphical user-interface (gui) is programmed in an evolutionary way, using the VisualStudio Wizard. A formal design and specification on gui-class members is therefore not given in this chapter. In the remainder of this chapter we will only briefly describe the main characteristics of the classes in the image- and segment processor module. Because a comprehensive description of the functionality provided by the NeoOv.Lib can be read in the appendix 1.1 through 1.7, we will only indicate how the neocognitron simulator is used in the s/w of our CLPR application.

## Logical View

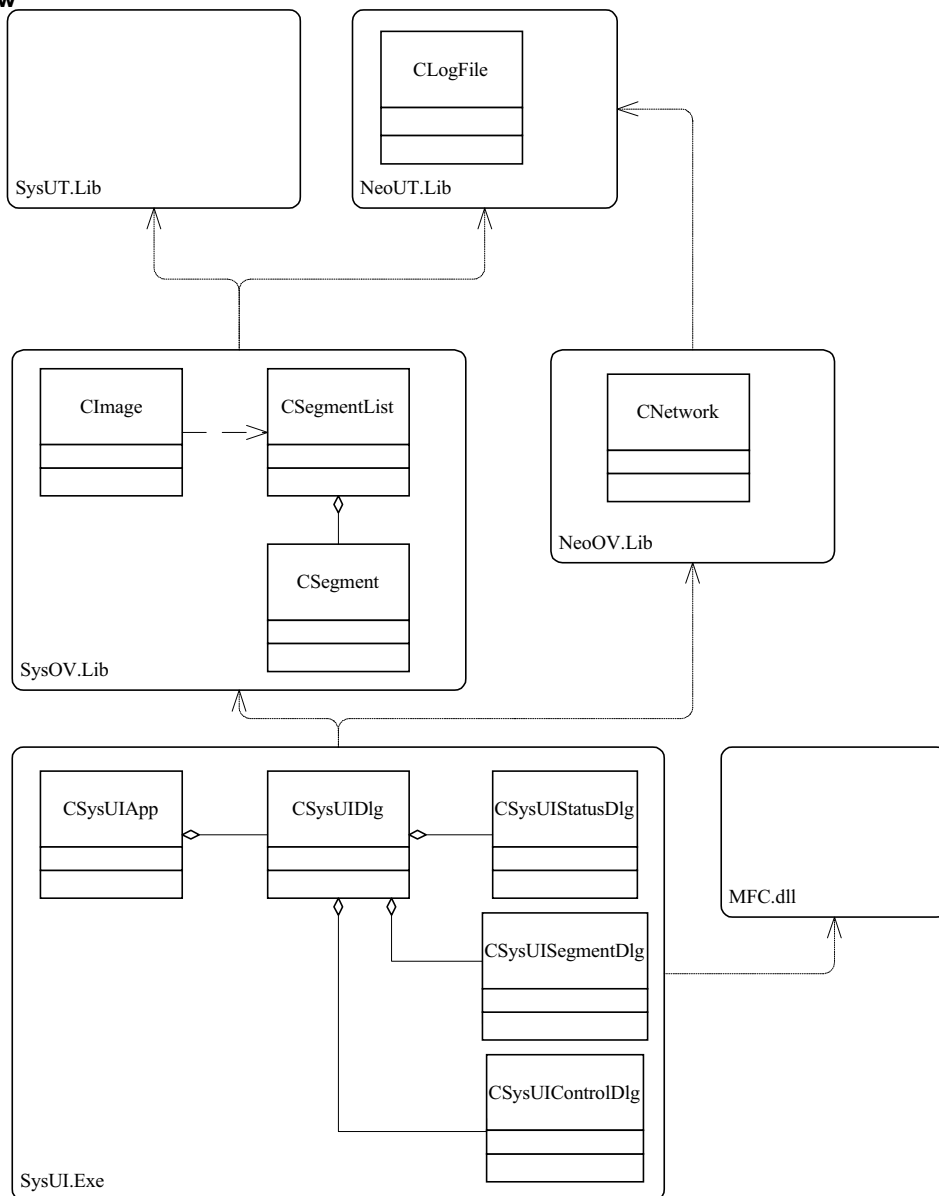


Fig. 5.4 The class diagram of the CLPR

## 5.2 The image processor

The image processor functions are all captured the `CImage` class. The size of the image is given by the private datamembers `m_Height` and `m_Width`. The pixel intensity values are stored in the `m_Values` datamember. The datatype of `m_Values`, a pointer to an array of unsigned char, restricts us to only 8-bit grey-scaled images. In the sections that follow some details of specific implementations on the class operations used within our CLPR are described.

<b>CImage</b>	
- unsigned char	*m_Values
- int	m_Height
- int	m_Width
<pre> + CImage() + CImage(const unsigned char *b, const int h, const int w)  + ~CImage()  + int  GetHeight  (void) + int  GetWidth   (void) + int  GetSize    (void) + bool GetSegment (int, CSegment *)  + void SetImage (const unsigned char *b, const int h, const int w)  // image transformations + bool Invert      (void) + bool GrayScale   (void) + bool FilterLow   (void) + bool FilterHigh  (void) + bool BlackWhite  (int GLOBAL    LOCAL, int level=200) + bool Boundary    (int BLACK    WHITE) + bool FloodFill   (int BLACK    WHITE) + bool Noise       (int BLACK    WHITE) + bool Sobel       (void) </pre>	

Fig. 5.5 The image class

### 5.2.1 Contrast stretching

Contrast stretching is performed by the operation `CImage::GrayScale()`. Basically, this operation performs the function as depicted in figure 1.2. The grey-scale boundaries are given by two systems constants defined in a header file. Currently these values are set to:

```

#define LOWERSLICEBOUND 40
#define UPPERSLICEBOUND 210

```

### 5.2.2 Filtering

The filtering procedure to blur the image is a 3\*3 mask according equation 1.12. Filtering the image is performed by `CImage::FilterLow()`. We have experienced that a larger filter yields better processing results. Within the filtering block in figure 5.1 this method is called twice.

### 5.2.3 Binarization

The implementation of the local binarization routine deviates slightly from the algorithm presented in section 1.2.3. The local binarization procedure is applied by calling the `CImage::BlackWhite()` method with parameter `GLOBAL`. Basically, the functions defined in equation 1.10 and 1.11 are implemented. The threshold value used to set each pixel of the image to either black or white is determined by taken the average of the five pixels. Four surrounding pixels plus the pixel under consideration.

If the average value however is greater then a certain level, defined by the `UPPERGRAYLEVEL` constant, and the difference between the maximum and minimum value of the pixel intensity values of the five pixels considered is above a certain value, defined by `UPPERGRAYRANGE`. The threshold value is adjusted to:  $\text{minimum value} + (\text{maximum value} - \text{minimum value})/2$ .

Another situation where the threshold value is adjusted is, if the average value is less then a certain level (defined by the `LOWERGREYLEVEL` constant) and the difference between the maximum and minimum value of the pixel intensity values of the five pixels considered is above a certain value (defined by `LOWERGREYRANGE`). The threshold value is also adjusted to:  $\text{minimum value} + (\text{maximum value} - \text{minimum value})/2$ . This threshold adjusting based on the maximum and minimum intensity values of the surrounding pixels described above prevents the binary image to become too noisy. The

biniarization constants are given by four systems constants defined in a header file. Currently these values are set to:

```
#define UPPERGRAYLEVEL 180
#define LOWERGRAYLEVEL 70
#define UPPERGRAYRANGE 30
#define LOWERGRAYRANGE 20
```

### 5.2.4 Area filling

After the image has been biniarized, filling operations are applied to get rid of all segments which are not potential number plates characters. Areafilling is performed by the `CImage::FloodFill()` method. The operation `CImage::FloodFill(int)` is called with the filling color required. Either the parameter is `BLACK` or the parameter is `WHITE`. The `CImage::FloodFill()` method calls a recursive c-function from the SysUT library to perform the actual filling.

### 5.2.5 Segment locator

The `CImage::GetSegment(int, Csegment *)` does locate the segments on the biniarized image (note that segments are considered groups of black connected pixels). The integer parameter gives the start location for searching on the image. The `Csegment` pointer is the output parameter. The `CImage::GetSegment()` method calls a recursive c-function from the SysUT library to perform the region-oriented segmentation.

## 5.3 The segment processor

The central object in the image processor step of our image CLPR system is an instance from the segment class. As stated earlier a segment is the 'raw' cutout from the original image. The segment locator discussed in the previous section gives use the exact location. The segment catcher as part of the gui actually cuts out the segment from the original image preserving the original pixel intensity values. Figure 5.6 gives all data members and operations available on the segment class. In this section, we only briefly describe the most relevant operations on segments.



<b>Csegment</b>	
<pre> - int m_Height - int m_Width - int m_Line - int m_Pos - char m_char[6] </pre>	
<pre> + CSegment() + ~CSegment() + CSegment&amp; operator=( const CSegment&amp; rhs )  + void SetSegment      ( const int maxx, const int maxy,                         const int minx, const int miny ) + void SetSegmentWidth ( const int width ) + void SetSegmentHeight( const int height ) + void SetSegmentPixels( void ) + void SetSegmentPixels( int , int ) + void SetSegmentChar  ( char * ) + void SetPos          ( int ) + void SetLine         ( int )  + int      GetSegmentWidth  ( void ) + int      GetSegmentHeight ( void ) + int      GetLine          ( void ) + int      GetPos           ( void ) + char*    GetSegmentChar   ( void ) + unsigned char* GetSegmentPixels ( void ) + bool     IsValidSegment   ( void )  // segment transformations + void EnLargeSegment      (void) + void RemoveBorder        (void) + void SetCorrectDimension (void) + void SetCorrectScale     (int,int) + bool Biniarise           (float level=1.6) + void Thinning            (bool)  + bool operator == (const CSegment&amp; seg) const + bool operator != (const CSegment&amp; seg) const + bool operator &gt; (const CSegment&amp; seg) const + bool operator &lt; (const CSegment&amp; seg) const  + int      m_Maxx + int      m_Maxy + int      m_Minx + int      m_Miny + unsigned char* m_Pixels </pre>	

Fig. 5.6 The segment class

### 5.3.1 Segment scaling

`CSegment::SetCorrectScale(int,int)` reformats the segment as cutout from the image. This operation is only performed if a segment is too large to fit on a 19\*19 plane or if the segment to be recognized is too small (refer to section 4.3 for our motivation). The original ratio of the segment is preserved in order to prevent character deformation. E.g., a segment holding an image of the numerical character '1' of size 38\*4 will be resized to a segment of 19\*2. The scaling operation does change the segment's resolution. The segment is either magnified or reduced, in both cases new pixels intensity values are calculated by linear interpolation.

### 5.3.2 Segment binarizer

After the segment has been scaled, the segment is binarized according the procedure explained in section 3.4. During binarization the number of black and white pixels are counted. The binarization operation validates the segment against two system constants: `MINPERCENTAGEBLACK` and `MAXPERCENTAGEBLACK` Currently these values are set to:

```

#define MINPERCENTAGEBLACK (double)0.06
#define MAXPERCENTAGEBLACK (double)0.90

```

This validation prevents segments, caught by previous steps, to be considered as characters erroneously.

### 5.3.3 Segment copying/pasting

`CSegment::SetSegmentPixels()` is the operation which is called after the segment has been binarized and validated. Basically this operation pastes the binary character image onto a 19\*19 plane to be used directly by the neocognitron recognizer. In our CLPR the segment pixels will be placed centrally on the 19\*19 plane; however an overloaded operation

`CSegment::SetSegmentPixels(int,int)` is available to place the segment on any location on the plane (this operation may be used to verify the location invariance of the neocognitron recognizer).

### 5.3.4 Segment thinning

As stated earlier the neocognitron used is designed to recognize features that have only line segments of one pixel thick, therefore a thinning operation is applied on the segment before the segment image is supplied to the recognizer. We have implemented the 'Hildditch' algorithm to skeletonise the segment. We have slightly changed the procedure described in Woods[1, pp 491] to prevent certain line segments to be wiped out completely. Later in chapter 6 it will be mentioned that finding a better skeletonizing operation will improve the overall performance of our CLPR system. Skeletonising is performed by calling the `CSegment::Thinning()` method.

## 5.4 The character recognizer

Within the CLPR, two types of neural networks can be used: a MLP using direct input and a neocognitron network. The MLP is used only for verification and not described in this document. The neocognitron simulator software is described in the appendix. The incorporation of the neocognitron recognizer in the CLPR is straightforward. What is important for the CLPR software implementation may best be explained by the code fragment listed below.

```
#include "network.h"

float* z;          // pointer to array of all networks output cells
int    max_layer; // integer defining the layer number

// instantiate the network object
m_NetWork = new CNetwork();

// load the network configuration
m_NetWork->Install(SupervisedFukushima, "c:\MyNetworkFile.net");

// let the network recognise an input sample
m_NetWork->Test(a pointer to an array containing the image sample);

// get the networks output
z = m_NetWork->GetPlaneOutput(5, 'C');
```

Fig 5.7 Sample code to include the neocognitron in the application

In the main program of the CLPR, a pointer to the network object is created. Once the network has been created, the network constants and weights are installed from an existing network configuration file: `m_NetWork->Install()`. After installation, the network is supplied with a character array holding the image segment to be recognized: `m_NetWork->Test()`. What follows is retrieving the output after the network has propagated the output neuron excitations. This is accomplished by the method `m_NetWork>GetPlaneOutput()`.

## 5.5 The user-interface

As shown in figure 5.2 there are many buttons to push on the control panel of the userinterface. We will not explain the working out of each button in this report because many are self-explaining and because such a description should be given in a user-manual. Basically, the system can be operated in two ways. Either all processing steps including loading the image are initiated separately by successively pushing the buttons that call their corresponding operations or the system is presented a file containing a list of images to be processed 'automatically'.

By pushing the 'FileSeries' button, a FileDialogBox is presented to select a file containing all files to be processed. After the file has been selected the system starts processing all image listed in the file automatically. After all images listed in the file have been processed, an output file will be available with the recognition results. Below some lines of an output file are listed as an example.

```
..
FILE: C:\Afstudeer\PIC00032.BMP      READ: [?LZ9DN]
FILE: C:\Afstudeer\PIC00035.BMP      READ: [JSPT84]
FILE: C:\Afstudeer\PIC00036.BMP      READ: [VG47G0]
FILE: C:\Afstudeer\PIC00037e.BMP     READ: [LTTL53]
FILE: C:\Afstudeer\PIC00038.BMP      READ: [HPBG77]
..
```

Fig 5.8 File sample containing the analyzed bitmap images and the recovered car license plate

Intentionally left blank

## Chapter 6

# Evaluation

## 6.1 Systems performance

Below, in figure 6.1, it is demonstrated that our CLPR does recognize license plates on photographs produced by automated road cameras operated by the Dutch police. It would have been beyond the scope of this research to verify our system against a large number of these types of photographs in order to measure the recognition success rate of the prototype described in this report. In stead, for evaluation of our prototype system we have produced a number of photographs of cars parked on several parking lots of the Delft University campus. The pictures were taken using a digital pocket camera equipped with a 36-mm lens. All images taken have a 640\*480 pixel 8-bit grey scale format. Figure 6.2 shows a representative sample of our evaluation set.



Fig. 6.1 A processing example of a photograph produced by an automated camera operated by the Dutch police. The image cut-out is 320\*280 pixels 8-bit grey scaled



Fig. 6.2 A representative sample of the evaluation set used

### 6.1.1 Performance definition

Obviously, our system is not a ready-to-use automated CLPR to be used for traffic law enforcement. Only certain aspects of image processing and character recognition have been studied and implemented in software. Nevertheless, it is possible to compare the performance of our CLPR to other system-architectures described in scientific articles or other commercial systems. In this chapter we will present the results of our system regarding the image processor and the recognition rate of the neocognitron, based on the definitions described below.

Our prototype CLPR is based on the block diagram of a general image understanding system as presented in figure 1.0. However, not all building blocks have been implemented.

- There is no automated image acquisition.
- Feature extraction seemed superfluous in our setup.
- A post-processor has been defined but was not implemented.
- Finally, there exists no feedback in the processing chain of our prototype.

Basically, our system contains only two components:

1. An image processor that transforms an input image into a series of binary image cutouts representing individual license plate characters.
2. A character recognizer that classifies the binary image cutouts into alphanumeric characters.

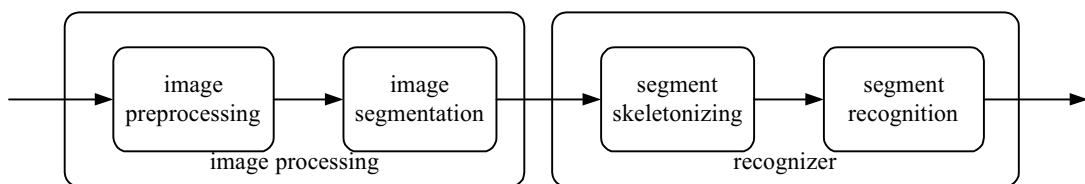


Fig. 6.3 The model of our system used to define performance measures

Below two measures are defined which are used to qualify the system's overall performance. In the next two sections these measures are explained and quantified for the CLPR prototype described in this report.

- Supply the image-processor an image, how well does it segmentate the license plate characters? This measure has been defined as the image processor success rate: *PRSC*.
- Given a binary segment cutout representing a license plate character is it recognized correctly? This measure has been defined the recognition rate: *RR*

## 6.1.2 Image processor performance

Ideally, the image processor would return exactly the number of character cutouts the license plate has, of each bitmap photo presented to the system. Our image processor does not. In stead it can return:

1. Not a single valid segment.
2. Many segment but none of them may be considered character cutouts.
3. A set of segments containing not all license plate character cutouts.
4. A set of six or more segment, among which are all license plate character cutouts.

It will be clear that a CLPR only can recover the car's license plate if the image processor returns a value that fall within category 4 of the list given above.<sup>5</sup> If the image processor returns at least all license plate character cutouts, the image processor step is considered successful. In all other cases, the image processor has failed, and eventually the CLPR system would not read the license plate. Our image processor performance is defined by the successrate achieved when processing the image:

$$PRSC = \frac{\#images\ at\ least\ all\ characters\ isolated}{total\ \#images}$$

Equation 6.1 Image-processor succes rate

We have taken in total 162 pictures. Nineteen of them seemed unusable. Either the picture was over exposed, suffered fuzziness caused by movement or the car has been photographed from a distance too far away. Finally we ended up with 144 photographs, 125 of them were pre-processed successfully. This yields a image processor success rate of 87%. Of course, this number may vary depending on the characteristics of the photographs in the testset. It will go up in a highly controlled environment, and it will surely be much lower under bad lighting conditions. However it gives a good indication that the techniques and algorithms used in the pre-processor may work quite satisfactory in this kind of software application.

Unfortunately in most cases the image processor returns more than six segments. As will be demonstrated in section 6.1.4 this should not cause any performance degradation on the total system but it makes it mandatory to include a post-processor in the CLPR. Another problem with the many erroneously isolated segments is related to run-time performance. Currently the setup of our system is such that all isolated segments are considered candidate license plate characters. The neocognitron recognizer will therefor analyze all of them. Since the neocognitron network simulator is by far the most CPU-cycles consuming part of the system, long processing times images should be anticipated. A recognition cycle for one segment takes about 2 seconds on an 1400 Mhz Pentium PC using a release version of the neocognitron simulator. Below in figure 6.4 a histogram is given showing the frequency of the number of segments isolated per image.

---

<sup>5</sup> remember we do not catch complete license plates but consider individual characters in our segmentation procedures

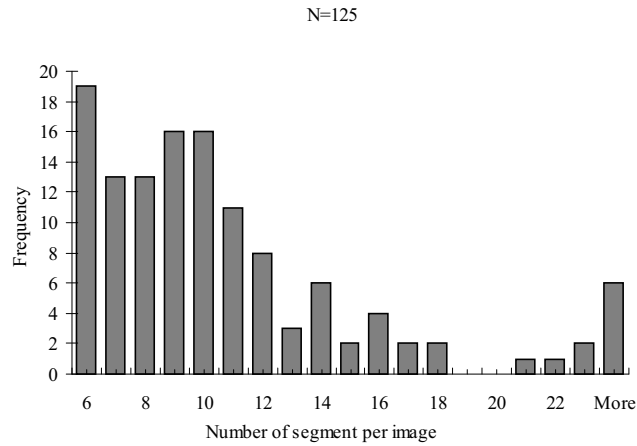


Fig. 6.4 The frequency on the number of segments caught

What goes wrong in the image processor? The main reason the pre-processor fails to isolate license plate characters is because the etching filter used in the binarization method seems not to enclose the license plate characters completely. The filling operations applied after the image has been binarized do wipe out completely the characters. See figure 6.5. This happens when the license plate color is too much alike the color of the cars chassis.



Fig. 6.5 left: 240\*190 pixel cutout from original picture #154. middle: binarized image right: no plate left.

Another reason why segmentation fails is that individual characters become mutual connected or become connected to the plate boundary. The filling operation in this case does wipe out the not completely isolated characters. See figure 6.6. This problem also does occur on dirty numberplates or when for example too big screws have been used to attach the numberplate.

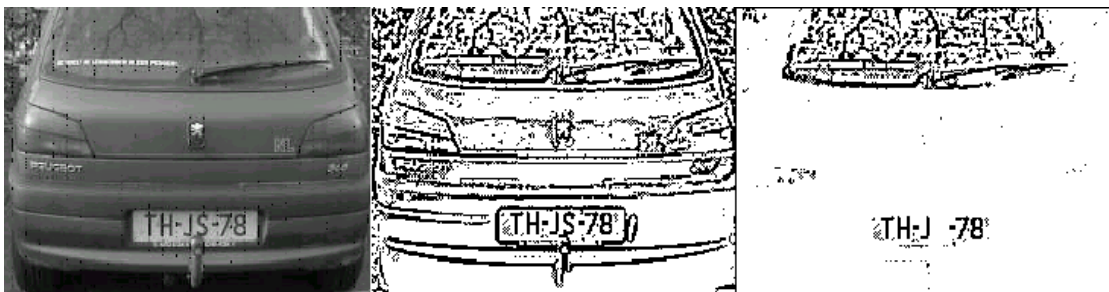


Fig. 6.6 left: 240\*190 pixel cutout from original picture #015. middle: binarized image right: missing 'S' char



### 6.1.3 Recognition performance

The recognition performance of the system is defined by equations 6.2.

$$RR_{character\ level} = \frac{\#correctly\ read\ characters}{\#characters\ isolated}$$

$$RR_{licenseplate\ level} = \frac{\#correctly\ read\ plates}{\#images\ at\ least\ all\ characters\ isolated}$$

Equation 6.2 recognition succes rate

It should be noted that recognition success rate ( $RR$ ) is determined by both the segment skeletonizer and the neocognitron network. Refer figure 6.3. From the set of 125 pictures successfully pre-processed, we leave out 4 pictures that have license plate character images of less then 10 pixels height. The remaining 726  $= (6 * 121)$  characters were passed through the neocognitron recognizer. 13 out of 726 were not classified at all, 34 out of 726 were misclassified, refer table 6.3. On character level, the recognizer part of the CLPR system exhibits the following performance figures.

Table 6.1 Character level recognizer performance

Recognition on character level	$RR_{charlevel}$
Recognition rate (correctly classified segments)	93.9 %
Error rate (misclassified segments)	4.4 %
Rejection rate (unclassified segments)	1.7 %

The figures above are derived from table 3.4.2 in the appendix. Tables 6.2 lists the recognition rate performance on license plate level. It shows the percentage of images from which all six alphanumerical characters of the numberplate have been recognized correctly (recognition rate), the percentage of image where not all characters have been recognized correctly (error rate) and the percentage of numberplates that show one unclassified character (rejection rate) at least.

Table 6.2 Character level recognizer performance

Recognition on license plate level	$RR_{platelevel}$
Recognition rate (correctly classified license plates)	72.7 %
Error rate (misclassified plates)	16.6 %
Rejection rate (unclassified plates)	10.7 %

In table 6.3 a matrix is given that shows the the recognition results on all characters that appear in the testset. What is causing the rejection or misclassification in the recognizer module of the system ? 47 out 726 characters are either rejected or wrongly classified. A brief analysis showed the failures of the recognizer fall in either one of three categories below:

1. Regrettably, the 'mini' discriminator neocognitrons for 0/D, 2/Z, .. ,8/B suggested in section 4.4 let us down in some cases. In section 6.1.4 we will comment on this.
2. The thinning operation on the segments seems not to be able to process certain input patterns correctly. In figure 6.7a some typical examples are given of skeletonizer errors.
3. Finally, it was observed that the rest of the failures is due to the neocognitron failure. Seemingly the trainingset patterns for the network and/or its configuration were not sufficient to correctly

classify the input samples. Figure 6.7b gives some examples of segments that belong to the this category.

Below in figure 6.7a four character segments are depicted of respectively the pictures #20, 141, 154 and 27 of the testset. Refer appendix table 3.4.2. The recognizer respectively reads “?”,”?”, “H” and “Y”. This seems very plausible. The segments shown in figure 6.7b are isolated from image # 51, 89, 58 and 27 respectively. The neocognitron rejects the J,L,G and misclassifies the character K as R.

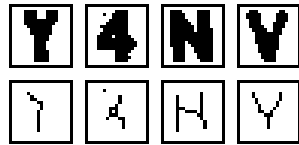


Fig. 6.7a Thinning failures

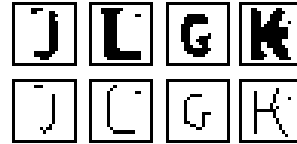


Fig. 6.7b Recognizer failures

Table 6.3 The character confusion matrix observed on the evaluation photo set

→ Recognized as																																
Input character																																
↓	0	1	2	3	4	5	6	7	8	9	B	D	F	G	H	J	K	L	N	P	R	S	T	V	X	Y	Z	?*	!†	#		
0	16	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	20	
1	-	26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	26	
2	-	-	23	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3	-	3	26		
3	1	-	-	25	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	27	
4	-	-	-	-	17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	18		
5	-	-	-	-	-	25	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	1	26	
6	-	-	-	1	-	-	29	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	30	
7	-	-	-	-	-	-	-	18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	18	
8	-	-	-	-	-	-	-	-	24	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3	1	28		
9	-	-	-	-	-	-	-	-	-	22	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	23	
B	-	-	-	-	-	-	-	-	-	-	2	-	25	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	2	28	
D	-	-	-	-	-	-	-	-	-	-	-	1	44	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	45
F	-	-	-	-	-	-	-	-	-	-	-	-	-	38	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	38	
G	-	-	-	-	-	-	-	-	-	-	-	-	-	-	34	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	35	
H	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	31	-	-	-	-	-	-	-	-	-	-	-	-	-	-	31	
J	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	17	-	-	-	-	-	-	-	-	-	-	-	1	-	18	
K	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	8	-	-	1	-	-	-	-	-	-	-	-	1	1	10	
L	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	28	-	-	-	-	-	-	-	-	-	-	1	1	29	
N	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-	-	-	28	-	-	-	-	-	-	-	-	-	-	5	33	
P	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	26	-	-	-	-	-	-	-	-	-	-	26	
R	-	-	-	-	-	-	-	-	-	-	1	-	-	-	1	-	-	-	-	-	36	-	-	-	-	-	-	-	-	2	38	
S	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	26	-	-	-	-	-	-	-	2	28		
T	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	26	-	-	-	-	1	-	27		
V	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	26	-	23	-	4	-	5	28		
X	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	22	2	-	1	2	25		
Y	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	17	-	1	-	18		
Z	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	26	-	1	27	
#	17	26	23	26	17	27	29	19	26	22	27	49	38	34	37	17	8	29	29	26	37	27	26	23	22	23	29	13	34	726		

\* ? = rejected characters

† ! = misclassified characters

### 6.1.4 Post-processor performance

In section 6.1.2 it was mentioned that a post-processor becomes necessary in the current system setup because of the large number of non character segments are isolated from the input images. As described in section 3.5 of this report the character recognizer does not only outputs the most likely character class the segment belongs to, but also returns the exact pixel location and size of the segment. It is for the post-processor to select just these six license plate segments from the segment list. Below the systems output is given produced when processing image #111 from the testset.



Fig. 6.9 Image number 111 of the testset

```

15:36:32 New Bitmap Loaded      File=C:\new\PIC00111.bmp      Size=640*480
15:36:34 Bitmap Preprocessed using method I
15:36:39 Start recognizing picture segments
15:36:42 analysing segment num=[ 1]   loc=[328,338,162,178]       NeoCognitron Output=[S5000]
15:36:44 analysing segment num=[ 2]   loc=[341,352,162,178]       NeoCognitron Output=[R0000]
15:36:47 analysing segment num=[ 3]   loc=[359,370,163,179]       NeoCognitron Output=[D0000]
15:36:49 analysing segment num=[ 4]   loc=[371,379,163,180]       NeoCognitron Output=[J0000]
15:36:52 analysing segment num=[ 5]   loc=[387,398,164,180]       NeoCognitron Output=[2Z000]
15:36:54 analysing segment num=[ 6]   loc=[397,407,164,180]       NeoCognitron Output=[7T000]
15:36:57 analysing segment num=[ 7]   loc=[270,275,222,232]       NeoCognitron Output=[10000]
15:36:57 InValid segment num=[ 8]     loc=[282,296,224,236]       skipped
15:37:00 analysing segment num=[ 9]   loc=[114,124,369,386]       NeoCognitron Output=[?????]
15:37:00 InValid segment num=[10]     loc=[123,134,370,386]       skipped
15:37:00 InValid segment num=[11]     loc=[ 28, 34,386,396]       skipped
15:37:00 InValid segment num=[12]     loc=[570,583,417,432]       skipped
15:37:00 InValid segment num=[13]     loc=[ 51, 64,441,458]       skipped
15:37:00
15:37:00 End recognizing picture segments
15:37:00 Processing segment list selecting plate candidate characters
15:37:00 segment num=[ 1]   loc=[115,123,370,385]       size=[ 16, 9]   pos=[ 20,115]   recog=[?????]
15:37:00 segment num=[ 2]   loc=[271,274,223,231]       size=[ 9, 4]   pos=[ 6,271]   recog=[10000]
15:37:00 segment num=[ 3]   loc=[329,337,163,177]       size=[ 15, 9]   pos=[ 0,329]   recog=[S5000]
15:37:00 segment num=[ 4]   loc=[342,351,163,177]       size=[ 15, 10]  pos=[ 0,342]   recog=[R0000]
15:37:00 segment num=[ 5]   loc=[360,369,164,178]       size=[ 15, 10]  pos=[ 0,360]   recog=[D0000]
15:37:00 segment num=[ 6]   loc=[372,378,164,179]       size=[ 16, 7]   pos=[ 0,372]   recog=[J0000]
15:37:00 segment num=[ 7]   loc=[388,397,165,179]       size=[ 15, 10]  pos=[ 0,388]   recog=[2Z000]
15:37:00 segment num=[ 8]   loc=[398,406,165,179]       size=[ 15, 9]   pos=[ 0,398]   recog=[7T000]
15:37:00 No More Characters Found

```

Fig. 6.10 The output log produced by the system when processing the image number 111 above

A program to select only the last 6 segments of the list of 8 segments initially analysed, and mark them as the license plate characters would not be very difficult. No performance degradation would occur by this step if added to the CLPR.

Clearly a post-processor must be included in our system setup because non character segments are to be removed from the output string based on their location and size. This post-processor could also perform a syntactical analysis on the output string. Although the use of a post-processor as syntax forcer is not favourable in a CLPR because it makes it less general applicable, it must be mentioned that the post-processor in our system could increase the total performance significantly. Table 6.3 shows 12 misclassifications on 0/D, 2/Z, 5/S or 8/B characters. From table 3.4.2 in the appendix it is derived that correcting this misclassification by using the Dutch license plate syntax rules we would recover an additional number of 9 license plates.

Our system definitely needs real life experimental verification, but recognition rates on plate level of 80 % can be achieved in this setup using a post-processor. Taken the image processor into account we finally would have an overall succesrate of 70%<sup>6</sup>, on images that have character sizes as small as 10 pixels.

<sup>6</sup> = PRSC \* RR<sub>platelevel</sub> = 0.87 \* .80

## 6.2 Future work

The CLPR described in this report is a prototype. *A prototype is the first model that is made of something. The prototype is used as a basis for later improved models.* [Collins cobuild English language dictionary]. Unfortunately it is not common practice in the software development industry to through away the implemented prototype and starting from scratch only reusing the prototype's concepts.

However, we believe the demonstrated image processor concept is usable for this kind of software application but needs further analysis. Especially the binarization method described in section 5.2.3. This algorithm relies too much on practical experience rather than a theoretical foundation. The neocognitron on the other hand is a proven concept and it has been demonstrated that it does work quite well in this application also. The neocognitron simulator we have build should be analyzed regarding it response times, in order to let it compete with MLP artificial networks or template matchers in this kind of application.

An overall performance of about 90% should be achievable using the concepts of our CLPR design by increasing image processor and recognizer performance.

The image processor performance can be increased by:

1. Using higher resolution digital photographs.
2. Developing a better local binarization method.
3. Tuning the segment validator to decreased the number of erroneously isolated segments.

The recognizer performance can be increased by:

1. Modifying the segment-thinning algorithm by taken the original grey-scales of the binarized segment into account.
2. Enlarging the neocognitron input plane to 38\*38 pixels. Larger character image samples would suffer less from the thinning deformations demonstrated in figure 6.7a.
3. Try to train the neocognitron the unsupervised way to make it more robust against character deformations as depicted in figure 6.7b.
4. Adding a post-processor to include syntax forcing and plate validation to reduced the error rate.

# References

- [1] Gonzales R.C. and Woods R.E. (1992) *Digital image processing*, Addison-wesley ISBN 0-201-5083-6.
- [2] Shouno H., Fukisuma K. and Okada M. "Recognition of handwritten digits in the real world by a neocognitron", (1999) *Knowledge-based intelligent techniques in character recognition*, pp.17-47 ISBN 0-8493-9807-X.
- [3] Fukisuma K. and Wake N. "Handwritten alphanumeric character recognition by the neocognitron", (1991) *IEEE Transactions on neural networks VOL 2. No 3. May 1991*, pp.355-365.
- [4] Terbrugge M.H., Nijhuis J.A.G. , Spaanenburg L. and Stevens J.H. "License plate recognition", (1999) *Knowledge-based intelligent techniques in character recognition*, pp.263-296 ISBN 0-8493-9807-X.
- [5] Kulkarni A.D. (1994) *Artificial neural networks for image understanding*, Van Nostrand Reinhold ISBN 0-4222-00921-6.
- [6] Bishop C.M., (1998) *Neural networks for pattern recognition*, Oxford Press ISBN 0-19-853864-2.
- [7] Hecht-nielsen R. (1989) *Neurocomputing*, Addison-wesley ISBN 0-201-09355-3.
- [8] Zell A. Mamier G. Vogt M. (1995) , SNNS Stuttgart Neural Network Simulator User Manual Version 4.1. Report No. 6/95
- [9] Steuer M., (1996) Parallely Implemented Neocognitron-like Neural Network, *master's thesis report*, Delft University of Technology.
- [10] Kanayama K., Fujikawa Y., Fujimoto K., and Horino M. (1991) "Development of vehicle-license number recognition using real time image processing and its application to travel time measurement," *Proceedings of the 41<sup>st</sup> IEEE Vehicular technology Conference (ST. Louis)*, pp. 798-804
- [11] Fohr R. and Raus M. (1994), "Automatisch lesen amtlicher Kfz-Kennzeichen," *Elektronik* No. 1, pp. 60-64

- [12] Davies P., Emmott N., and Ayland N. (1990), "License Plate Recognition Technology for Toll Violation Enforcement," in: *Proceedings of IEEE Colloquium on image analysis for transport applications*, Vol. 35, pp 7/1-7/5.
- [13] Hwang C., Shu S., Chen W., Chen Y. and Wen K. (1992), "A PC-based License Plate Reader." *Proceedings Machine Vision Applications, Architecture and Systems Integration*, Vol. SPIE1823 (Boston, MA), pp.272-283.
- [14] Tanabe K., Marubayashi E., Kawashima H., Nakanishi T., and Shio A. (1994), "PC-based Car License Plate Reading," *Image and Video Processing*, Vol. SPIE-2182, pp.220-231.
- [15] Lisa F., Carrabina J., Perez-Vincente C. Avellana N., and Valderrame E. (1993) "Two-bit Weights are enough to solve Vehicle License Plate Recognition Problem." , *Proceedings of the International Conference on Neural Networks*, Vol. 3 (San Francisco, CA), pp.1242-1246.
- [16] Williams P., Kirby H., Montgomery F., and Boyle R. (1989), "Evaluation of video-recognition equipment for number-plate matching", *Proceedings of the 2<sup>nd</sup> International Conference on Road Traffic Monitoring* (London U.K.), pp. 89-93.
- [17] Lotufo R.A., Morgan A.D., and Johnson A.S. (1994), "Automatic Number-plate Recognition," *Proceedings of the IEEE colloquium on Image Analysis for transport applications*, Vol. 35, pp.6/1-6/6
- [18] Yoo J., Chun B., and Shin D. (1994), " A neural network for recognizing characters extracted from moving vehicles," *Proceedings of world congress on Neural networks*, Vol. 3 (San Diego, CA), pp. 162-166.
- [19] Kertesz A. Kertesz V. and Muller, T. (1994) "An On-line Image Processing System for registration number identification," *Proceedings of IEEE International Conference on Neural Networks*, Vol. 6 (Orlando, FL), pp. 4145-4147.