Using Max-Trees with Alternative Connectivity Classes in Historical Document Processing

Jaap Oosterbroek - S1657291
Department of Artificial Intelligence
University of Groningen

Supervisors:
Dr. Marco A. Wiering
Dr. Michael H.F. Wilkinson

May 24, 2012

Abstract

In mathematical morphology, connectivity classes are a formal way of describing the grouping of sets of elements in a graph. When applied to images, connectivity classes can easily be used to describe relations in binary images and generalizes to gray-scale by means of threshold super-positioning. Connectivity classes have long been of interest in image processing research, because they provided a basis for the invention and verification of connectivity based algorithms. Much work has been invested in finding structured ways of modifying connectivity classes with predictable results.

Here we present a method of using a combination of two known types of connectivity: mask and edge-based connectivity, in historical document processing. Max-Trees are data structures that can be used to express various connectivity classes. A system was built that uses a Max-Tree for all steps in the document processing chain, from preprocessing to feature generation and classification. The system aims to show that a combination of these two types of connectivity counteracts some of their mutual weaknesses. Two types of filters: the k-subtractive and k-absorption filters were used to remove noise and help with segmentation. Finally a class of features that can efficiently be computed in a Max-Tree, Normalized Central Moments, were used to classify the character zones resulting from this segmentation.

Contents

Chapter 1

Introduction

In almost all machine vision applications the perceptual grouping of pixels into meaningful entities, such as people, traffic signs, or characters, is an essential step in processing. Such perceptual groupings are most often based on what pixels in an image humans consider to be grouped together. Moreover, some of these groups are part of other groups which implies a hierarchy in this perceptual grouping, for instance from letters to words to sentences. In this thesis we will look at perceptual grouping from a mathematical morphology perspective and we will describe them formally in terms of *connectivity*.

Connection in the graph theoretical sense means that any number of elements might belong to a single set called a *connected component*. When we apply an operation called a *connectivity opening* onto any member of a connected component set the entire connected component is returned. In the context of two dimensional images this means that selecting one pixel will return the entire structure of which it is a member. This can be of great help when we want to segment interesting structures such as letters from a background such as the rest of a page.

This thesis attempts to employ the graph theoretical concept of connectivity in order to improve segmentation in the document processing of a historical hand-printed text: the *Rerum Frisicarum Historia* (Hereafter referred to as the *Rerum*) written by Ubbo Emmius and published in 1616. The Rerum is a compilation of several hand-printed books comprising over 1500 pages. It is the first of several such works residing in the library of the university of Groningen that has recently been digitized. The works are of interest to many scholars for a variety of reasons such as language research and a historical perspective into the works of the first Rector Magnificus of the University of Groningen. The ability to automatically convert these scans into a type of searchable text format, such as an ASCII or Unicode transcript, would further aid study of these works. The text in itself is also an interesting challenge for automated document processing systems: it contains too much variety for an off the shelf optical character recognition system to be effective, however it has been quite well preserved and it is easily read by non-expert researchers.

Earlier work by Van Laarhoven [54] on the same data set encountered several problems that were difficult to tackle with the methods that were employed. One of the most persistent and interesting of these problems was that of vertically separated connected components belonging to the same character. Van Laarhoven's thesis shows that although language models are a good way of coping with letters being split up or merged in a horizontal fashion, they have great difficulties achieving the same results in two dimensions. Therefore the system described in that thesis makes

many mistakes in characters comprised of vertically separated components. Letters such as the 'i', the 'j' and punctuation characters such as ':', ';' and '?' are the most common examples of such characters. However, the rarer cases of such characters also include letters using diacritics. This thesis focuses for a large part on these types of problems and tries to achieve a solution by using different *connectivity classes*.

A connectivity class is a formal method describing ways in which the elements of a graph or set are connected [19]. In our case, these graphs will be images and their elements will be pixels. The most common connectivity classes are of course the 4-connectivity in which pixels are connected to their orthogonal neighbors and the 8-connectivity, in which pixels are connected to their orthogonal and diagonal neighbors. However, as we shall show in chapter 3, there are many different ways of building connectivity classes.

We will use a data structure called a *Max-Tree* [39] to apply connectivity classes to images. The Max-Tree and its variants, the dual-input Max-Tree and the triple-input Max-Tree offer a relatively simple, intuitive and above all efficient way for representing connectivity classes in images. Our goals are to improve segmentation by means of filtering and the use of different connectivity classes and to improve our understanding of normalized central moment based attributes; a group of features that are 'native' to Max-Tree. Normalized central moments are 'native' to the Max-Tree in the sense that the speed at which they can be computed in a Max-Tree is dramatically increased by the way in which the Max-Tree represents its data. The rest of this introduction tries to give a short perspective on why we deemed these goals relevant and will also give an overview of the structure of this thesis.

1.1 Improving segmentation

The main goal of this thesis is to provide some footholds in improving segmentation in historical hand-print documents. The term *improving* implies that one segmentation is *better* than the other which in turn implies that there is such a thing as a *good* segmentation or even the optimal segmentation. In order to avoid a philosophical debate about segmentation, it is perhaps useful to take a few words to elaborate upon this issue.

As already stated by Ø. Trier in [52]: "performance evaluation of low-level image processing routines, such as binarization, segmentation, edge detection, thinning, etc., is inherently difficult". Such actions are always performed with a certain goal in mind. In the context of this thesis we will consider segmentation as a prerequisite to classification. Therefore a good segmentation should be regarded as: *a method of isolating regions of an image that benefits classification of those regions.* This means that a good segmentation has the following properties:

1. **A good segmentation isolates meaningful regions in an image.**

2. **A good segmentation is stable in relation to the features it generates.**

3. **A good segmentation is robust against changes in the context.**

In this thesis we will look at several ways to perform segmentation using the Max-Tree focusing on three different aspects related to this data structure. First of all we will investigate a group of features based on segmentation; normalized central moments are computed on the basis of a connected component ignoring the gray values of the elements in this component and the area

surrounding the connected component. More details on normalized central moments are provided in the next section.

Secondly, we will look at two hyper-connected filtering techniques: the k-subtractive and k-absorption filters [28]. Hyper-connectivity is an alternative connectivity notion that allows pixels to be part of multiple structures simultaneously. Using hyper-connected filters means that we can combine information from different gray levels, contrast information for instance, to filter on different levels at once. It also means that every pixel has several chances either to be filtered or preserved depending on its membership.

Finally, we will show comparative experiments with three different connectivity classes expressed in three different types of Max-Trees. The regular Max-Tree uses a common 4-connectivity function and is similar to the connected component operations used by Van Laarhoven. The dual-input Max-Tree uses a mask to combine the different image structures of the same letter into a single connected component. However, this mask sometimes allows merges of structures that should not be merged. The triple-input Max-Tree tries to counteract this by cutting these structures apart by severing the connection between the pixels of the structure: the edges. In chapter 3 we describe the basic theory of how these connectivities should work, while in chapter 4 we show how they can be implemented into a Max-Tree.

1.2 Examining normalized central moment attributes

Although several publications explain the concept of normalized central moments (abbreviated as NCMs) [15] and show some basic results on their use in classification [38] they have rarely been used in a context that uses more noisy data such as the text of the *Rerum*. NCMs therefore are of double interest to us. First of all we want to gain a better understanding of what moments should be used and in what fashion we can use them. Theory suggests that NCMs have several interesting properties: lower order moments should be more robust against noise than higher order moments and should contain more information. NCMs should also be scale invariant. Theoretically this would allow us to classify letters of different font sizes with the same training-set.

We also use NCMs as a way to evaluate the stability of our segmentation system. As mentioned above, we consider a segmentation good if it is stable in relation to the features it generates. This means that a nearest neighbor classifier based on features that are robust against small amounts of noise, such as NCMs, should be able to give us a reasonable classification score. In this way NCMs also give us a way to compare the stability of the different forms of connectivity that we will employ.

1.3 Research questions

Summarizing the previous sections we formulate three research questions:

1. Are normalized central moments reliable features for a character classifier in a historical document processing application?

2. Can hyper-connected filtering techniques help in a historical document processing application?

3. How can different connectivity classes be used to improve segmentation in a historical document processing application?

1.4 The structure of this thesis

This thesis is subdivided into five main chapters and some appendix material. We start with an introduction explaining how connectivity has been used in the past in optical character recognition applications and how this field has tried to cope with segmentation and classification problems in general. It gives a brief introduction to the history of this field and tries to show how connectivity has been handled in the past and how this relates to our current methods. It also tries to show how somewhat more expansive processing systems use language models and document scructure information to improve their classification scores while reducing our abiliy to determine where this increase comes from.

The third chapter covers most of the important theory relating to connectivity. It explains the core concepts of the connectivity class and shows extensions to second generation connectivity classes, hyper-connectivity classes and edge-based connectivity classes. It will do this from a mathematical perspective and methods of implementing these different forms of connectivity will be left for the next chapter.

Chapter 4 covers the methods used to implement the different connectivity classes described in chapter 3. It provides a short introduction to the Max-Tree data structure and its variants: the dual-input Max-Tree and the triple-input Max-Tree. It will also explain in more depth what NCMs are and how we can compute them efficiently in a Max-Tree. Finally, it provides a short discussion about our evaluation measures.

Chapter 5 explains the experimental results we got from examining the methods described in chapter three. For each of the experiments it will explain why we thought it necessary, how it was done and what the results were. It will also give a brief discussion of our perspective on the results. The final chapter contains a general evaluation of our research and suggestions for future work.

Chapter 2

Related work

Connectivity is an important concept in most text recognition systems. The most common example of this is the thresholding operation, but it can also be used more explicitly for instance by extracting connected components. This chapter tries to give a context of our research in the field of optical character recognition (OCR for short), while chapter 3 and 4 try to give more backgound information relating to other research in connectivity operations. The first part of this chapter contains a very brief history of OCR in general detailing how earlier research dealt with connectivity problems similar to those addressed in this thesis. The second part takes a more in depth look at how modern OCR systems are attempting to solve connectivity problems, focussing on the heavy influences of language models and other top down information concepts.

2.1 A short history of optical character recognition

2.1.1 The early beginnings of optical character recognition

The field of OCR is one of the oldest and most thoroughly researched subfields of image analysis. Some researchers claim it started as early as 1929 with the submission of a patent by Paul W. Handel [13]. A similar patent was later submitted by Gustav Tauschek [50]. In short both machines were mechanical template matching automata using photo-detectors and physical masks, both mechanical ways for performing connectivity operations. While neither concept ever received a full application, the idea of template matching has remained central in OCR until this very day.

After these initial tries OCR slumbered for several decades until the electronic computer revolution of the early 50's. David H. Shepard presented a major breakthrough in the field in 1953 by building a machine that converted characters read into machine language instead of some other physical medium as predecessors did [44]. While his machine was still an electro-mechanical implementation, digital implementations would quicky follow. By the end of the 50's there were many commercial digital OCR systems in operation [12].

A first deviation from the template matching approach was made by Glauberman in 1956 [11], his method used a histogram instead of a full mask. The histogram was constructed using a horizontal slit sliding vertically over the character while a photodetector registered the output. While this mapping from two dimensional information to one dimensional information was prompted by the electronic difficulties of digitizing images at that time it can be seen as the start of feature based

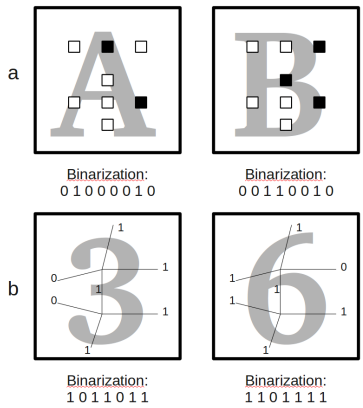

Figure 2.1: The image above shows the peephole and sonde-slit methods. **(a)** Shows how to convert an image into a binary feature vector using the peephole method: a set of critical areas of a single character are evaluated as either empty (0) or filled(1). These are then stored in an array. In this case read left to right top to bottom. The peephole method requires very stable fonts and segmentation and is therefore unsuitable for handwriting recognition applications. **(b)** Shows a method to convert an image into a binary feature vector using the sonde-slit method. Arabic digits are roughly drawn around two centers. By having lines extend from these centers at carefully chosen angles and measuring which of those cross the digit, a binary feature vector can be acquired. The sonde method is relatively stable against variation in digit shape and therefore found broad use by banks and postal companies in handwritten digit recognition applications.

approaches. The idea that not all information was needed to identify a character quickly became quite popular and many feature based methods appeared. Most important were the peephole, as described in [22], and Dimond's slit-sonde method [8]. Both are illustrated in figure 2.1.

In these early beginnings connectivity operations were often done by mechanical means, Handel, Tauschek, Glauberman and Dimond all used mechanical masks combined with photosensitive electronics as a means of getting discrete information about real world printed letters. These methods all show many similarities with mask-based connectivity which we shall discuss in section 3.2.6.

2.1.2 The field diversifies

Dimond's sonde method implementation marks a clear change in the field of OCR. While originally the field was mainly focused on modern western machine print recognition, other applications such as handwriting recognition machines or Asian font recognition now seemed feasible. Early handwriting recognition applications focused mainly on digit recognition but complete postal codes soon followed.

In the meanwhile machine print recognition made steady progress. A great deal of effort was put into reading machines for vision impaired members of society but the many challenges still undressed in both the classification and the speech synthesis part of the problem made broad deployment impossible. A comprehensive overview of methods and applications at the time is offered by [10].

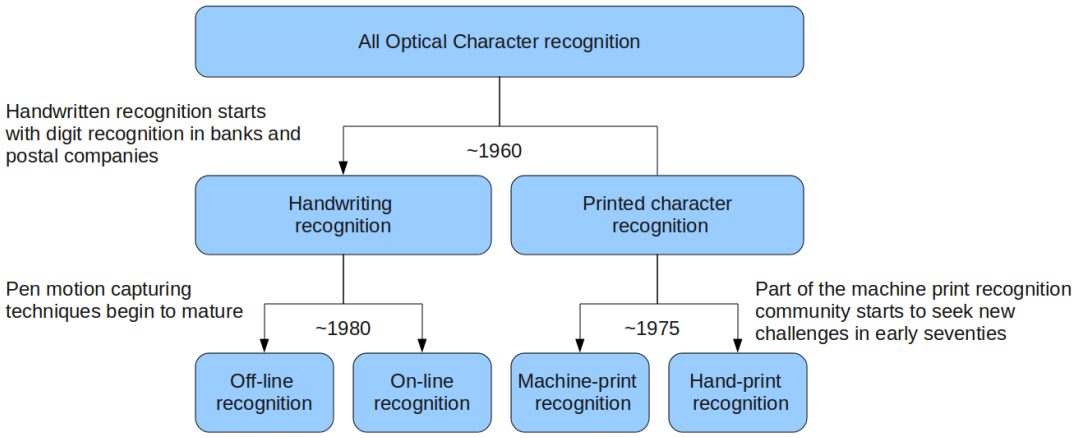

Figure 2.2: The image above shows how the field of OCR diversified over time.

In 1981 the first omni print system was produced by Kurzweil et. al. [14] and slowly but surely machine print was losing its appeal to OCR researchers. While many problems in the field remained unsolved, researchers were confident that they could design a system for any particular machine print font. Many lost interest in the remaining challenges. Over the course of the next decade many researchers therefore moved on to either harder, more noisy data sets or handwriting recognition. IBM was one of the first big research companies to shift their attention to character recognition in poor quality print documents [49] (hereafter referred to as *hand-print*). These days even western hand-print has lost much of its alure. While some research is still done in hand-print recognition it is more as a steppingstone to harder data sets such as hand writing or oriental hand-print.

Advances in pen motion capturing techniques in the early 80's finally makes real on-line handwriting recognition feasible [55]. It had long been a scientific ambition to also incorporate the timed data of the pen movements in the models, but handwriting is an extremely fast process. At least a 100hz sampling frequency is needed to do any sort of reliable online-recognition. At least twice that speed is required for writer or signature identification. Online handwriting data is a time series. Time series allow easy use of very different techniques than image data such as Hidden-Markov models (also used less effectively on image type data) and echo-state networks. Some researchers also tried to convert offline handwriting data into online data handwriting with varying results.

The diversification of the field shown in figure 2.2 resulted in very different methods of classfication. Offline handwriting applications for example are often based on word recognition, while online counterparts often try to learn more information from the strokes. Their connectivity operations have remained very similar. These are almost always based around a combination of a thresholding function with a connected components search. As we shall show in section 4.1 the use of a component tree such as the Max-Tree make these operations vastly simpler.

2.1.3 Arrival of the internet and the digital age

The penetration of internet into the mainstream in the late 90's has had a profound influence on OCR research and its applications. Although in some places prescient libraries had already begun to digitize works with systems as seen in [24], many companies, most notably Google, were now rapidly accelerating this process. Although most newly generated documents were now immediately digital there was a great backlog of documents that were not available in digital format. This prompted a renewed effort to perfect the existing OCR systems. An effort that could now be felt throughout society.

The impact of automated reading systems and their limitations can be clearly illustrated by the example of CAPTCHAs [56]. A CAPTCHA is a *Completely Automated Public Turing test to tell Computers and Humans Apart* that can be used to secure internet services from use or misuse by computer programs. They typically feature an image of some automatically generated pseudo word in heavily distorted font or presenting them in a format that makes segmentation extremely difficult. The user is then to prove his or her humanity by submitting its transcription in a text field on the site. The point is that while humans often have little trouble in reading these phrases, computers often can't make sense of them. These days CAPTCHAs are used as a security feature on almost every website providing a free on-line computation or storage feature. The use of CAPTCHAs is so abundant that the original researchers even proposed to use CAPTCHAs as a means of allowing people to help researchers annotate hard to read sections of text [57].

Because of their large interdependence with internet the use of PDAs and later smartphones can be seen as a byproduct of the internet. While many of these hand held computer systems initially employed some form of on-line handwriting recognition, most of the market now uses touchscreen QWERTY keyboards supported by simple language models. Either the on-line recognition software was not up to user standards or users simply prefer a keyboard of any kind above handwriting. Some users of hand held devices also desired to use their phone to digitize small fragments of text by means of the camera. The low quality of these cameras formed an obstruction which led to development of super-resolution image reconstruction techniques [29]. These allowed larger and better quality images to be constructed out of many low quality ones.

2.2 Modern methods in optical character recognition

In some ways not much has changed in recognition systems over the past half century. Preprocessing still attempts to convert the image to some binary representation and the most used method of recognition is still template matching. In other ways current methods are very different from those designed 30 years ago. As illustrated in figure 2.3 the basic concepts of character recognition have not changed much but their interconnectivity has changed dramatically. The most important here is the concept of using higher level information, such as information about language grammar, to supplement or disambiguate lower level information, such as character identity. This is referred to as the top-down approach in contrast with the normal bottom up approach. In some cases this means that modules support each other, in other cases they are executed simultaneously. This highly connected top-down and bottom up approach makes interpretation of work in this field difficult. It is hard to be sure what method or approach in a system amounted to what result. It is for this reason that in this thesis we avoid these interconnected methods, even though they can yield great benefits in terms of results.

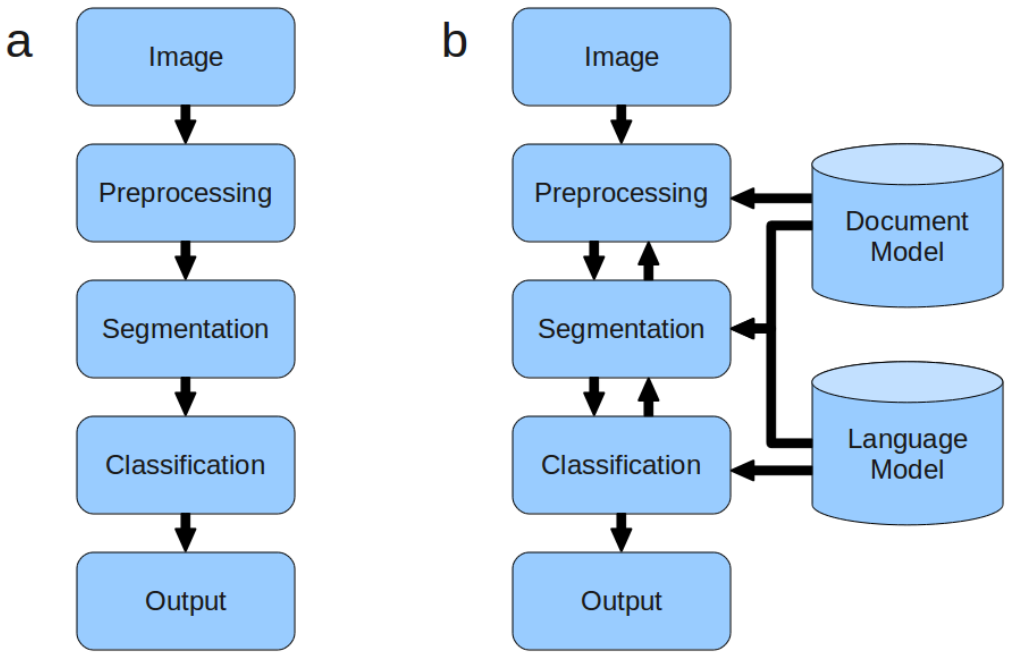

Figure 2.3: a: A typical recognition system as it was often seen in the seventies, every module has its own task. A new module is only activated if the previous module has completed its work. b: an example of a more modern interconnected system. There is more interaction between different modules top-down as well as bottom-up. Higher level information such as language and document models are also used to reinforce more valid hypothesis and detriment others.

2.2.1 Modern preprocessing

The core purpose of most preprocessing steps is still to remove as much noise as possible from the image. In practice this often means binarization: reducing the image to only black and white pixels. There are even contests dedicated to this processing step [32]. While binarization used to be motivated by the memory limitations of machines, these days this constraint is rarely an issue. As shown in section 3.2 many operations are simpler in a binary image.

The simplest method for binarization is thresholding. The most popular way of choosing a (local)threshold is by Otsu's method [25] but many others exist. An overview of thresholding techniques and how they perform on various OCR data sets is offered by [43]. It is important to consider that binarization not only discards noise but also useful information. How much is lost largely depends on the type of binarization but also on the document itself.

Some systems skip binarization and use the gray-scale information. A system demonstrated in [62] uses edge strengths extracted from gray scale images to separate text areas and images in documents. Research in [51] shows a way of dealing with *bleed-through* and *show-through* effects in gray scale images. We try to tackle this problem using hyper-connected filters as described in [28]. More information on these filters can be found in section 3.2.7 and section 4.1.6.

2.2.2 Modern segmentation

In the classical sense segmentation was only about cutting out the separate characters and feeding them to some sort of classifier. This quickly evolved to include line and word segmentation. Modern segmentation is also about document processing. This entails dropping the assumption that all the text is written in a single block of uniformly distributed lines. The analysis of layout and typesetting then becomes an issue. Many techniques and features from segmentation and recognition systems have been reused in this problem. This problem area is still very much in movement as there is still a lot of discussion over what the aim of document processing could and should be. Research by Tang for instance [48], gives a survey on the logics and semantics of different text areas and how these can be connected.

Once the document has been divided up into text areas, headers, images and other components, the text areas can be split into lines. While the segmentation into lines is of course still an important step, this has never been a real challenge. An overview of many methods that can be used to segment the lines in a document is given by [20]. In our experiments we shall use a relatively simple line-splitter explained in [54], more on this in section 4.1.5.

After lines have been isolated they need to be split into words and characters. Most handwriting recognition systems forgo the level of characters altogether and only try to recognize words. These *holistic* recognizers either depend on knowing all possible words or employ a reject category to deal with new words. As implied earlier, many systems consider multiple mutual exclusive segmentations of words and characters. The segmentation module tries to find all possible segmentations for a line, word or character zone and the classifier then has the task of choosing the best fit taking into account both segmentation fits and higher level information such as classification results and grammar.

In many (hand)print recognition systems the objective is still to do character by character segmentation and classification. The most common and effective method of classification is still the use of a correlation of the complete image with a cutout of each character. Although converting the results of such a segementation to text is sometimes complicated and the cost of this operation increases rapidly with the number of classes present, they are simple to implement and yield great

results in many cases. As an alternative one could split the document up in loose characters using some sort of connectivity operation and classify these character by means of features.

Of great intrest in this case are so called splits and merges of characters. Characters are sometimes broken up into different parts by paper damage or an ink deficit, they are also sometimes merged together by noise or an overflow of ink. Especially the latter situation has received a lot of attention. Methods detailed in [58] for instance use a shortest path computation to split these characters. Most solutions for the other case, the merging of split characters involve some sort of post-processing. This thesis will attempt to solve this problem using image masks as detailed in section 3.2.6. For further reading [7] gives an overview of many other methods that have been used in character segmentation.

2.2.3 Modern classification

Classification is of course a central part in text recognition systems. It is especially on this part of text recognition systems that much attention has been focused. In the early stages features existed out of whatever was mechanically readable and most were implemented as hardware configurations. Using modern computer technology there is no limit to what features can be computed as long as they can be defined in a formal way. Some features however are more complex to compute than others, something that warrants attention when the volume of input data becomes bigger. In handwriting recognition the use of language models and especially dictionaries, has led to an emphasis on word recognition instead of character recognition. This has led to the development of so called holistic features, features that describe an entire word instead of a single letter. Some examples of strong holistic features are histograms of gradients [35], bottom profiling, template matching, Gabor-features and many others.

A dictionary can never contain every word one might encounter in a context free setting. Therefore some systems still use character based recognition as a backup in the reject category. Character segmentation is quite simple, in many machine print cases character based features still find application there as well. A whole range of classifiers and feature sets in the context of character recognition is evaluated by [46].

As a machine learning engine k-nearest neighbor is very popular, because it is fast in training and relatively fast in use. It also gives relatively good results regardless of the data set that is considered. Support vector machines, learning vector quantization and a host of neural networks are also used. These usually give better results but they are more complex to configure, and sometimes require long training times.

2.2.4 Language models

Language models are an important part of modern classification and segmentation systems. Although they will not be featured in this research, not mentioning them here would be a strange omission. The most common use of language models is computing the most likely a-priori (without the use of any data) classification of some portion of text. In this way one can give a significant boost to probability based classifiers such as Hidden Markov Models [6] or regular Bayesian models [31]. Research described in [40] also shows how this principle of using top down knowledge can be used to boost segmentation scores. By using a probabilistic path search algorithm to generate hypotheses for the segmentation system, it manages to boost its recognition scores significantly.

The most popular language models are bi- or tri-gram models as seen in [30], because they

are easy to make and to use. However, other models exist, such as max-entropy parsers [34] and parsers with more syntactic linguistic knowledge [2]. For a good overview of the most commonly used statistical language models see [37].

While the gains of language models have been large, they added a new dimension of complexity to evaluating text recognition research. As mentioned before interactions between segmentation, classification and language models are diverse and complex and they are often altered at the same time between papers. So while it is often possible to tell what the effects of a language model are within a specific system it is unclear what the effects are between systems. Even so there is no denial that the gains achieved by using language models are large and continue to increase.

2.3 Recent advances in connectivity frameworks

In the previous section readers will have noticed that much work has been put into the development of more useful language models and stronger features while relatively few publications have focused on preprocessing other than new thresholding techniques. This is mostly because the gains in that area have been slim for a long time. In contrast this thesis will focus exclusively on preprocessing and segmentation.

For the initial noise reduction process we will look at the use of some form of hyper-connected filtering. Hyper-connectivity as used in this thesis was first introduced by Serra in [41]. It entails a modification of the theory formed around regular connectivity frameworks that allows for overlapping components. K-flat hyper-connected filters can be used to segment large structures of star systems eliminating unwanted elements in both background and foreground. They have also been shown to be able to clear up bleed-through effects commonly found in historical documents [28]. Alternatively these attribute filters have also been used in [17] to clean up scan data of cranial arteries. Section 3.2.7 will give a deeper explanation of the theory behind hyper-connected components.

So-called second-generation connectivity classes are modified variants of regular connectivity classes. Especially mask-based second-generation connectivity seems to be extremely versatile since it can be used for both clustering (binding components together) and contracting (separating components that are normally connected) at the same time. This advantage is unfortunately overshadowed by practical limitations in the methods of contraction [59]. Section 3.2.6 will give a deeper explanation of the theory behind second generation mask-based connectivity.

Chapter 3

Theoretical background

Given some image, the question of which pixels are connected is almost a philosophical one. Maybe there is a big red car in the foreground and we should consider all those red pixels connected. But maybe there are multiple cars visible, then are they all connected to each other? Maybe they are all part of a big traffic jam and are more connected if they form a regular grid? The same questions can be asked of images showing buildings, people, galaxies, words on a paper and so on. *Connectivity classes* offer us a mathematical way of defining answers to these types of questions.

The research described in this thesis employs a combination of common and some less well known techniques in segmentation and preprocessing, all in some way related to *connectivity classes*. This chapter will try to provide the reader with the theoretical background necessary to understand the design considerations put forward in chapter 4. Readers that are familiar with the theory of connectivity classes or feel confident in their ignorance can simply skip this chapter and proceed to the next chapter.

We will start this chapter with a brief introduction into notation and the basic theory behind connectivity on graphs using the example of binarizations. After this we will look at *connectivity classes* and the most common connectivity operation: the *connectivity opening*. We also show the extensions of this theory: the *second generation connectivity classes* and *mask based connectivity*. Finally we will examine a more complicated generalization of *connectivity classes* named *hyper-connectivity classes* and their opening operation.

3.1 Notation

The paragraph below gives an enumeration of some of the most commonly used symbols in this chapter, and for each of them a brief explanation of how they are used.

1. D: The image domain that contains all possible images. The graph theoretical perspective on images explained in this chapter describes images as sets. When D is considered to be a binary image domain it is sometimes denoted as 2^n where n is the number of pixels or elements in D. In this chapter we for the most part consider situations where this is the case.

2. x: An element of D. x is used to represent the pixels in D. Since we are working in a set-theoretical framework we refer to them as elements.

3. X: An actual image, $X \subseteq D$. While D contains all x_i, X only contains some of them, therefore it holds that any X is a subset of D, $X \subseteq D$, and any X is an element of the powerset of D, $X \in \mathcal{P}(D)$. In visualizations of X elements in X are shown as white while elements in D but not in X are shown as black.

4. M: A masking image, $M \subseteq D$. M is similar to X but used for *mask based connectivity operations*. As seen in section 3.2.6 it is also possible that M is derived from X.

5. $f(x)$: The function that returns the value of an element $0 \leq f(x)$. Every element also has a value returned by $f(x)$, this is mostly used when dealing with gray-scale images. When dealing with binary images any $x \in X$ is considered to have the value 1, all other x are considered to have the value 0. $f(x)$ is explicitly used when we introduce k-flatzones in section 3.2.8.

The theories and the various definitions in this chapter have been taken from [4, 26–28, 42], these definitions were modified to conform to the same notational standard. Some of these sources also use definitions in terms of the power-set of complete lattices ($\mathcal{L}$) instead of the power-set of images as used in this thesis ($\mathcal{P}(D)$). The difference between the two is that a complete lattice also incorporates the infinite case while the $\mathcal{P}(D)$ only includes the finite ones. Because this consistently makes definitions, their notation and their relations more complicated this extension is not considered in this thesis. The use of lattices would be required however, if one would want to extend the theory beyond the limited domain of discrete finite binary images. Section 3.2.3 will show how we can use threshold superpositioning to apply this theory to gray-scale images regardless of this limitation.

3.2 Connectivity

The set-theoretical perspective can sometimes be confusing for people who are used to thinking about images in terms of grids and matrices. In this context it is often intuitive to think of pixels that are close together in a image grid as neighbors. This leads to the abundant use of 4 and 8-connectivity in which pixels are connected only orthogonally in the former case and orthogonally and diagonally in the latter case. Connectivity classes are an abstraction of the way in which pixels can be connected. This is useful when we either want to change the way in which they connect or when we try to find faster ways of finding out if pixels are connected. In both situations the use of connectivity classes helps us to make sure that the algorithms we define have no unintended side effects, such as generating new image structures during processing and work properly on fringe cases such as zero-size or infinite size images. Ultimately the framework of connectivity classes allows us to prove that the algorithms we build to perform connectivity openings are correct and do what we want them to do. As such they are an essential part of developing new tools in image processing.

3.2.1 Binarization

The mathematical theory behind connectivity classes quickly becomes complicated. We will therefore begin with a simple introduction using the example of binarizations. Binarization is the process of dividing a gray-scale image into two classes, often visualized as a black and a white class. In OCR this often means that the smaller class is the more interesting one containing the writing ink. The other class is deemed to contain the paper and, hopefully, most of the noise.

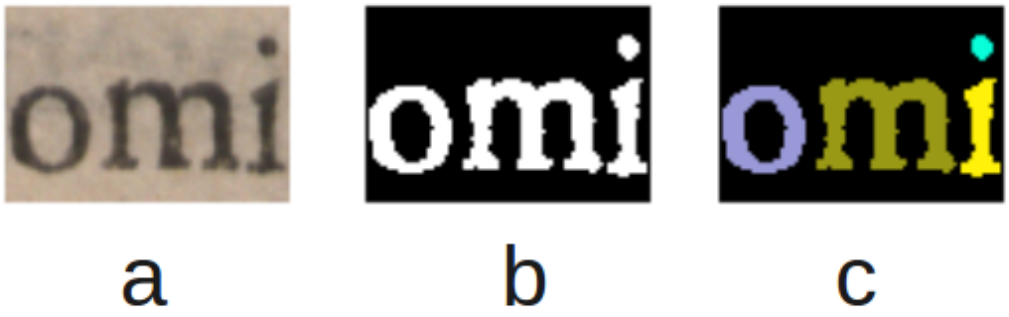

Figure 3.1: This figure shows the binarization of the letters 'omi'. **(a)** shows the original image. **(b)** shows the result of the thresholding operation. The image now only contains black and white pixels. **(c)** shows the image's connected components in 4-connectivity. Every connected component has a different color. Note that the letter 'i' exists out of two separate components.

Definition 1. *Adapted from [5] section 4.1. A family $\mathcal{F}$ forms a partition of image domain D if it satisfies:*

1. *For the set of all $A_i \in \mathcal{F}$ it holds that $\bigcup_i A_i = D$*
2. *For every $A \in \mathcal{F}$ and $B \in \mathcal{F}$ it holds that $A \cap B = \emptyset$ as long as $A \neq B$*

The most common type of binarization is thresholding. As mentioned in section 2.2.1 there are many ways to do thresholding binarizations. Figure 3.1 shows the binarization and the subsequent extraction of the connected components of some letters from our dataset. Let us name the white class X and the union of the black and the white class D. Binarization makes finding the individual letters simpler, if only because it makes it simpler to define where one begins and another ends. In figure 3.1 we show the different connected components as different colors.

This emphasizes the difference between the binarization and the connected component operation. The former only separates the image in two sets: white ($X \subseteq D$) and black ($D \setminus X$) while the latter splits it in 5 sets, one for every connected component and one for the background. The operation by which we extract connected components from the binarized image is called a *connectivity opening*. In section 3.2.4 we will look closer at the formal definition of a *connectivity opening*. In both the thresholding binarization and in the connected component representation it is the case that every pixel or element can only be part of exactly one set. This means that they both form *partitions* of the image. Formally a partition is defined as: Connectivity classes and as such binarizations and connected component operations create *partitions* of images. Dropping the second criterion of definition 1 leaves us with something called a *covering* of D. A covering merely ensures assignment of all elements in D but does not guarantee that that assignment is unique. In section 3.2.7 we will look at *hyper-connectivity classes*, these are a generalization over connectivity classes. Hyper-

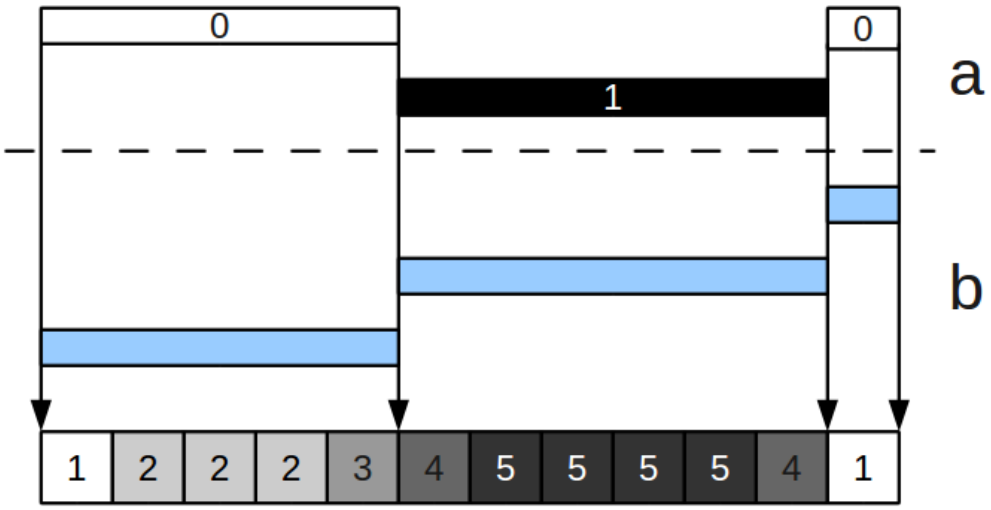

Figure 3.2: The bottom part of this image shows a 12 pixel 1-dimensional image, elements with a higher value are colored darker. The top part of this figure shows the binarization of an image and its associated connected component. The binarization shown in **(a)** is done by a thresholding operation, in this case with a threshold value of 4. All pixels of a gray value smaller than 4 are part of class 0, all pixels with a gray value equal or greater than 4 are part of class 1. In a binarized image connected components are simple to compute but this does not mean there are only two connected components. Part **(b)** shows the resulting three connected components.

connectivity classes result in coverings instead of partitions.

Figure 3.2 shows the difference between the binarization and the connected components operation in a more formal context. The figure shows a 12 pixel 1-dimensional image that we shall use as an example throughout this chapter. In this image each element has two neighbors if it is located in the middle but only one if it is near the border of the image. Again note that the binarization results in only two sets, while the connected component operation results in three.

3.2.2 Algebraic openings

In the context of mathematical morphology *connectivity classes* as described in [41] are often used as a way of defining in what way the elements of an image are connected. They provide a shared mathematical framework for a host of different connectivity operations including those mentioned in this section; most importantly *connectivity openings*. Connectivity openings are a very useful subset of *algebraic openings*:

Definition 2. *An algebraic opening is any set operator ψ with argument A that satisfies the following properties:*

1. *It is anti-extensive: $\psi(A) \subseteq A$. Our input is always a superset of the result of the operation. This means that while it can select a subset of the existing structures in A it can never create new ones.*

2. *It is idem-potent: $\psi(\psi(A)) = \psi(A)$. This means that it will not have any effect if it is applied more than once consecutively. This is important because without this property one would have to decide how often to apply it. An idem-potent operator should yield the same result no matter how many consecutive times it is applied on the same data.*

3. *It is order preserving: $A \subseteq B \Rightarrow \psi(A) \subseteq \psi(B)$. It will not alter an existing hierarchy of collections. This is important because it allows us to something called threshold superpositioning to generalize our methods to gray-scale images.*

Connectivity openings can be used for expanding a singleton element $x \in D$ to some large component $C_i \subseteq \mathcal{P}(D)$. In segmentation tasks it is often useful (and hard) to find some operator that can select an entire region of interest in this manner out of a larger image. The framework of connectivity classes is often used to ensure stability and reliability of such operators. The use of an operator that qualifies as an algebraic opening guarantees two operation properties that we deem essential in segmentation and a third property that is generally useful to us but not essential in its own right.

3.2.3 Threshold superpositioning

As mentioned in section 3.1 the theory presented in this chapter will mostly relate to binary images. In the example of binarization we converted our image to the binary domain by means of a thresholding operation. Since using a high threshold value will produce a binary image with fewer retained pixels than are made using a lower threshold value, carrying out such a threshold operation for every gray-scale level present in the image will provide us with a hierarchy of images. Any discrete gray scale image can be represented as the sum of such a hierarchy of nested binary images. This property is called *threshold super-positioning* [21].

When we consider the third property of definition 2, we can see that order in a threshold super-positioning system is preserved by algebraic openings, and thus by connectivity openings as well. This allows the use of computationally efficient methods such as the Max-Tree algorithm [39] explained in chapter 4. Threshold super-positioning allows the Max-Tree to use connectivity classes described in the binary domain on discrete gray-scale images.

3.2.4 Connectivity classes

Connectivity openings perform their task given a specific *connectivity class*. A connectivity class is an abstract definition of all possible results a connectivity opening can have given a certain image domain D. For instance how close two pixels need to be together to be considered connected and whether they need to be oriented orthogonally or diagonally and over what they can be connected. We will show some examples of simple connectivity classes later in this section. On the other hand, as we shall see in section 3.2.6, they can also include more complex structures.

To ensure the properties mentioned in definition 2 hold, connectivity classes as a whole must also satisfy some properties. For us to introduce the formal definition of a connectivity class we must first visit the notion of a family being *sup-generating* for an image space. This can essentially be seen as the constraint that no smaller parts can be missing from the family. If a family is *sup-generating* for an image space it means that every possible subset of that image space can be constructed as a supremum of elements in the *sup-generating* family. Formally:

Definition 3. *Inferred from text in [4]. For a family $\mathcal{F}$ to be sup-generating over an image space D it must satisfy:*

$$\textit{for any set: } C \in \mathcal{P}(D), C \textit{ can be constructed as: } C = \cup_i \{A_i \in \mathcal{F} | A_i \subseteq X\} \tag{3.1}$$

In section 3.2.7 we will see the definition of *chain-sup-completeness* which is essentially the other side of being sup-generating: that no bigger parts may be missing. This is however not a part of the definition of a connectivity class. With the concept of sup-generating defined we can now give the definition of a connectivity class:

Definition 4. *Adapted from definition 5.1 in [4]. Let D be an arbitrary non-empty set. A connectivity class $\mathcal{C}$ on D is any family that satisfies the following conditions:*

1. *$\emptyset \in \mathcal{C}$ and $\mathcal{C}$ is sup-generating for $\mathcal{P}(D)$*
2. *for any $\{A_i\} \subseteq \mathcal{C}$ for which holds $\cap_i A_i \neq \emptyset$ then $\cup_i A_i \in \mathcal{C}$*

The second criterion in definition 4 is of special interest in this thesis. Informally it says that if any number of sets in the class have a non zero intersection their union is also a member of the class. This is the most relaxed criterion that can be used; a single element is enough to form a bridge between two or more components, regardless of relative size or shape. In section 3.2.7 we will look at what happens if we make this overlap criterion stronger, requiring more than a single element overlap to connect two components.

As mentioned previously in this section the connectivity classes deal with the way in which neighbouring pixels can be connected. For this we would also need to define a neighbor relation that determines when two pixels are considered to be neighbors. Different neighbor relations result in different connectivity classes. Classical examples of this are the distinction between 4-connectivity often denoted as $\mathcal{C}_4$ in which only the orthogonal neighbors are considered connected and 8-connectivity ($\mathcal{C}_8$) where diagonal neighbors are also connected. Previous research [19, 36] shows that even this small distinction can have a large impact on a fundamental level in the context of finding the edges of components and translating the one into the other. However many other more extreme connectivity classes are possible. Figure 3.3 shows several variants of neighbor relations. Now, having described all required theory, we finally come to the definition of a connectivity opening:

Definition 5. *Adapted from definition 2 in [59]. Given a $x \in X$, a connectivity opening Γ_x of element x using the connectivity class $\mathcal{C}$ on D is defined as:*

$$\Gamma_x(X) = \cup_i \{A_i \in \mathcal{C} | x \in A_i, A_i \subseteq X\} \tag{3.2}$$

The result of such an opening is called a *connected component* or *connectivity grain* and is denoted as $\mathcal{C}_x$. While many subsets of $\mathcal{C}_x$ may exist within $\mathcal{C}$, Γ_x always yields the largest set in $\mathcal{C}$ of which x is a member. Because $\mathcal{C}$ forms a partition of D the result of Γ_x is unique.

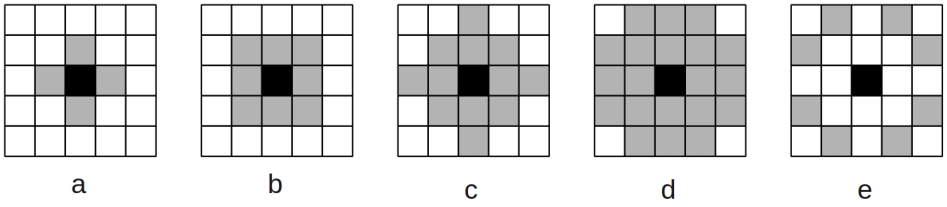

Figure 3.3: This figure shows several different neighbor relations that can be used to build connectivity classes. **(a)** and **(b)** show the neighbor relations for the classic 4 and 8-connectivity. **(c)** shows a Manhattan-distance-2 neighbor relation in which all pixels within a Manhattan-distance of **(2)** are connected. **(d)** shows a radial neighbor relation in which all pixels within an euclidean distance of the source pixel are considered neighbors to that pixel. **(a)**, **(b)** and **(c)** can all be expressed as a radial neighbor relation. **(e)** shows an unorthodox knight-leap neighbor relation, even though the purpose of a resulting connectivity class might seem hard to fathom there is no theoretical limitation to defining such a connectivity. Neighbor relations that are larger e.g. have more connections, or act over longer distances are also possible but might eventually lead to problems when implemented. More on this in chapter 4

3.2.5 Second-generation connectivity

Often the connectivity classes as they are described in section 3.2.4 are not powerful enough to provide the desired segmentation in an image. Second-generation connectivity classes provide a way to allow for more complicated connectivity classes by modifying the associated connectivity opening of an existing connectivity class. This produces a *child* connectivity class with a new connectivity opening operator. This operator denoted as Γ^{ψ} can be seen as a simple connectivity opening preceded by a structural operation ψ. This can either be an increasing operation in which case we consider the resulting connectivity opening a clustering operator or it can be a contracting operation in which case the resulting connectivity opening is called partitioning. Below are their formal definitions

Definition 6. *Taken from [27], definition 3. Given a $x \in D$, an image $X \subseteq D$, a connectivity class $\mathcal{C}$ and its associated connectivity opening Γ_x, a clustering second generation connectivity opening using a operator ψ is defined as:*

$$\Gamma_x^{\psi}(X) = \begin{cases} \Gamma_x(\psi(X)) \cap X, & \text{if } x \in X \\ \emptyset, & \text{otherwise} \end{cases} \tag{3.3}$$

Definition 7. *Taken from [27], definition 5. Given a $x \in D$, an image $X \subseteq D$, a connectivity class $\mathcal{C}$ and its associated connectivity opening Γ_x, a partitioning second generation connectivity opening using a operator ψ is defined as:*

$$\Gamma_x^{\psi}(X) = \begin{cases} \Gamma_x(\psi(X)), & \text{if } x \in \psi(X) \\ \{x\}, & \text{if } x \in X \wedge x \notin \psi(X) \\ \emptyset, & \text{otherwise} \end{cases} \tag{3.4}$$

3.2.6 Mask based connectivity

The biggest limitation to second-generation connectivity as it is described here is that it can either be clustering or partitioning, but never both at the same time. This limitation can be eliminated by using a mask to compute the connectivity rather than an operator. Mask based connectivity uses an additional image as a mask to compute a modified connectivity class on the original image. Structures in the original image that are part of the same structures in the mask image are clustered into a single connected component. Likewise, structures in the original image that are part of multiple structures in the masking image are contracted into separate images. This results in a new second generation connectivity class:

Definition 8. *Adapted from [27], definition 7. Let $\mathcal{C} \subseteq \mathcal{P}(D)$ be a connectivity class and $M \subseteq D$ be a connectivity mask for an image D. The mask-based second-generation connectivity class $\mathcal{C}^M$ is a connectivity class that in addition to the requirements listed in definition 4 satisfies:*

$$\text{for all } A, A \subseteq D \text{ where } \exists x \in D \text{ such that } A \subseteq \Gamma_x(M) \text{ it holds } A \in \mathcal{C}^M \tag{3.5}$$

Important to note in definition 8 is that the connectivity opening itself is used to define the class. This is of course possible because $\Gamma_x(M)$ is defined using $\mathcal{C}$ and M and not using $\mathcal{C}^M$. This means that $\mathcal{C}^M$ is not defined independently of the operation that extracts the actual connected components.

Definition 9. *Adapted from [27], proposition 1. Let $\mathcal{C} \subseteq \mathcal{P}(D)$ be a connectivity class, $X \subseteq D$ be an image and $M \subseteq D$ be a connectivity mask. We can now define the mask based connectivity opening Γ_x^M on X as:*

$$\Gamma_x^M(X) = \begin{cases} \Gamma_x(M) \cap X, & \text{if } x \in (X \cap M) \\ \{x\}, & \text{if } (x \in X) \wedge (x \notin M) \\ \emptyset, & \text{otherwise} \end{cases} \tag{3.6}$$

A useful observation in relation to definition 9 as noted in [27] is that the opening $\Gamma_x^M(X)$ is only idempotent as long as the mask M is computed independently of its own result. Even if M is derived from X we cannot derive M from the result of any operation using $\Gamma_x^M(X)$. We can, however, define mutiple masks $\{M_1, M_2, ...\}$ all derived from X and use them consecutively: $\Gamma_x^{M_2}(\Gamma_x^{M_1}(X))$. This would of course prioritise partitioning over clustering as any structures removed by an earlier mask can never be retrieved by a later one.

Although mask based connectivity solves some problems of regular second generation connectivity, it also leaves some problems unaddressed. As figure 3.4 illustrates, any part outside the mask is disassembled into singletons discarding any structure it might have. As discussed in [59] this is especially unfortunate when using filters based on area criteria as these singleton pixels are often the first to be removed.

Also, if we truly want to cut a connected component in X into two separate parts we will always need to have them separated by at least one single element, in the case of images. While in theory this separation margin can be infinitely small, in practice we will always be forced to throw away information in regions where it matters most.

3.2.7 Hyper-connectivity

Another way of extending connectivity classes is using hyper-connectivity. Instead of adapting the connectivity opening, this method changes a central property of connectivity classes themselves.

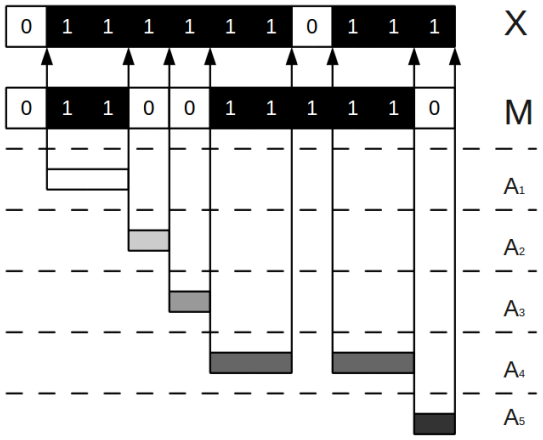

Figure 3.4: This figure shows how mask based connectivity works. Applying the connectivity opening $\Gamma_x^M(\mathrm{X})$ with mask M in image X yields the connected components A_1 to A_5 using definition 9. From this figure we can make several important observations. First of all A_2, A_3 and A_5 have been reduced to singletons. Intuitively it would make sense to keep A_2 and A_3 connected. Secondly it is not possible to make two larger components adjacent: we cannot connect A_2 to A_3 without also connecting them to A_1 and A_4

.

More specifically, it makes the second component in definition 4 more restricting. This allows for connectivity classes containing overlapping components without their union also being a member of that class. This makes it possible for elements to belong to two separate components at the same time. As one might imagine this will also make connectivity openings harder to define since they will no longer necessarily return a unique result. Several versions of hyper-connectivity exist in literature. In this thesis we will use hyperconnectivity as described in [28, 60].

In theory we could use hyper-connectivity to better describe situations in which structures in images overlap with each other. However, this way of applying hyper-connectivity has yet to be successfully applied. In this thesis we will use hyper-connectivity implicitly in our filtering technique, as explained in section 4.1.6. Before we can make a formal definition of hyper-connectivity classes we must first visit the notion of chain-sup-completeness:

Definition 10. *Adapted from [60], text section 2.1. For any family $\mathcal{F} \subseteq \mathcal{P}(D)$ to be chain-sup-complete it must satisfy:*

$$\text{for any non empty chain } \{A_i\} \subseteq \mathcal{F} \Rightarrow \bigcup_i A_i \in \mathcal{F} \tag{3.7}$$

As remarked in section 3.2.4 this is in some ways a counterpart to sup-generating. Informally it says that no bigger parts can be missing; if a set of components is in $\mathcal{F}$ its union is also in $\mathcal{F}$. Although it seems logical that this should also be in the definition of a connectivity class this property is also contained in the second part of definition 4. Since that is exactly the section we will modify, it makes sense to add it as an explicit requirement. The definition then of a hyper-connectivity class is as follows:

Definition 11. *Taken from [60] definition 6. A hyper-connectivity class $\mathcal{H}$ with an overlap criterion $\perp$ is a family in $\mathcal{P}(D)$ that satisfies*

1. *$\emptyset \in \mathcal{H}$ and $\mathcal{H}$ is sup-generating for $\mathcal{P}(D)$*
2. *for any $\{A_i\} \subseteq \mathcal{H}$ for which holds $\perp(\{A_i\}) = 1$ then $\bigcup_i A_i \in \mathcal{H}$*
3. *$\mathcal{H}$ is chain-sup-complete*

Members of a *hyper-connectivity class* are called *hyper-connected sets*. Hyper-connectivity classes might contain hyper-connected sets that are subsets of larger hyper-connected sets in the same hyper-connectivity class. Such subsets are deemed redundant and can be removed. To do this we derive the set of *hyper-connectivity components* of X, $\mathcal{H}^*_X$, from the original $\mathcal{H}$:

Definition 12. *Adapted from [28], equation 20. Given a hyper-connectivity class $\mathcal{H}$ the set of maximal hyper-connected components $\mathcal{H}^*_X$ is defined as:*

$$\mathcal{H}^*_X = \{A \in \mathcal{H} | (\nexists B \in \mathcal{H} : A \subset B) \wedge (A \subset X)\}$$

The members of $\mathcal{H}^*_X$ no longer contain subset members and are therefore maximal sets. We refer to these maximal hyper-connected sets as *hyper-connected components*. Because of the possibility that the components have overlapping regions that did not satisfy the overlap criterion, we are presented with a problem when we try to perform a connectivity opening on an element within such an overlapping region. When we would use an ordinary connectivity opening we would only return the union of all the possible hyper-connected components that contain the element making it less useful to use a hyper-connectivity class in the first place. Therefore we need to define a specific connectivity opening Γ^x_H.

Definition 13. *Taken from [60] definition 7. Given an image space D and a maximal hyper-connectivity class $\mathcal{H}^*_X$ on D, the associated hyper-connectivity opening $\Gamma^x_H(D)$ of element x is defined as:*

$$\Gamma^x_H(D) = \{A_i \in \mathcal{H}^*_X | x \in A_i\}$$

Note that the result of such an opening is not unique, it is a collection of sets.

3.2.8 K-flat zones

The previous section started with the assumption that we had some collection of overlapping sets that formed a covering of the image from which they where derived. There are of course many ways to define such a collection. One of the simpler ones is simply taking all connected components and dilating them by some structuring element larger than a singleton. In this research we will be using the k-flatzones.

The principle behind k-flatzones is rather simple in the discrete case; we partition a gray-scale image into connected components of a single gray level. Now we make a new collection of sets which are the union of one connected component and the connected components of the k gray levels below it. Also, if this connects us to other connected components within these k gray levels these are also included.

Definition 14. *Taken from [28], definition 4. A k-flatzone $F_{h,k}(x)$ at level h and depth k, is the set of all path-wise connected pixels marked by $x \in D$, with all intensities between h and $h-k$*

$$F_{h,k}(x) = \Gamma_x(\{p \in D | h - k \leq f(p) \leq k\})$$

k-flatzones can have overlap and can therefore be used to construct a hyper-connectivity class. Classes such as these will separate on strong edges without the need to define these in terms of gradients, Gabor filters or the like. Figure 3.5 shows an example of the depth 2 k-flatzones on an image, the hyperconnected sets they form given a overlap criterion of at least 3 elements and the resulting hyperconnected components. Note that neither k-flatzones nor hyper-connectivity classes necessarily form *partitions* of images.

3.3 Edge-based connectivity

All the connectivity openings described in the previous sections are based on the properties of an image $X \subseteq D$ in a binary case. In this sense the most remarkable concept was that of the k-flatzones that used the gray value $f(x)$ of an element x to determine its connectivity. We have also seen that even though mask-based connectivity classes offer us remarkable freedom in clustering, their contractions leave something to be desired. In this section we will address these and other issues by introducing edge-based connectivity. Connectivity based on the connectivity relation between two elements: the edge. We shall start with an explanation of the concept of edge-based connectivity and give some formal definitions. After this we will look at how edge-based connectivity can be used in connectivity openings.

3.3.1 Edge-based connectivity classes

When using edge-based connectivity we expand the image domain D to an extended image domain D^E that also includes the edges that express a certain neighbor relation. An image X^E now

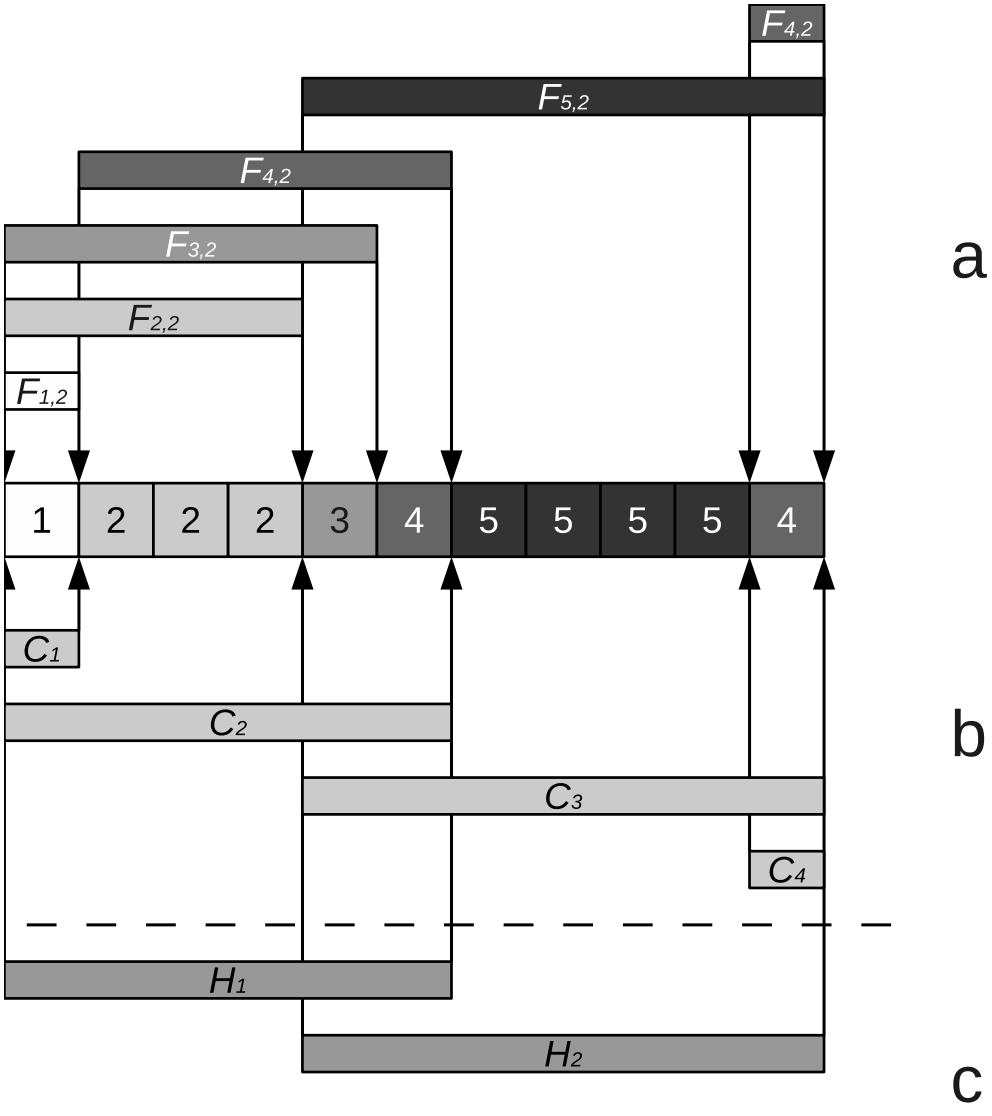

Figure 3.5: This figure shows the transformation of a one dimensional image in k-flatzones **(a)** with depth 2. $F_{i,j}$ signifies a k-flatzone of depth j at level i. In **(b)** these are reduced to four hyperconnected sets. $F_{2,2}$, $F_{3,2}$ and $F_{4,2}$ are merged into C_2 because they overlap 3 or more elements. The other k-flatzones are retained since they overlap 2 or less elements with any other k-flatzone. Finally in **(c)** the hyperconnected sets C_1 and C_4 are removed because C_2 and C_3 are supersets. These remaining hyperconnected sets form the hyperconnected componentes H_1 and H_2.

becomes a tuple of vertexes and edges $(V, R) \subseteq D^E$. The vertexes $V \subseteq \mathbb{Z}^2$ represent the pixels of the image, the edges $E \subseteq V^2$ represent a neighbor relation such as those shown in figure 3.3. Every edge $e_{1,2} \in E$ links a pair of two vertexes $(v_1, v_2) \subseteq V$. Using these we can define an edge-based connectivity class:

Definition 15. *Let D^E be an arbitrary non-empty set consisting of vertexes V and edges E. An edge-based connectivity class $\mathcal{C}^E$ on D^E is any family that satisfies the following conditions:*

1. *$\emptyset \in \mathcal{C}^E$ and $\mathcal{C}^E$ is sup-generating for $\mathcal{P}(D^E)$.*
2. *for any $\{A_i\} \subseteq \mathcal{C}^E$ for which holds $\cap_i A_i \neq \emptyset$ then $\cup_i A_i \in \mathcal{C}^E$.*
3. *$\{v_1, v_2, e_{1,2}\} \in \mathcal{C}^E$ if and only if $(v_1, v_2) \subseteq V$ are linked by edge $e_{1,2} \in E$.*

Edge-based connectivity is mainly an adaptation of the domain in which we consider connectivity. Since the connectivity relation itself remains mostly the same, the edge-based connectivity opening is almost identical to the regular connectivity opening in definition 5. The only modification is that it replaces D by D^E and that it can only be used to perform openings on vertexes and not on every element in X. The edge-based connectivity opening creates partitions of images and can be represented by a Max-tree data structure as shown in section 4.1.3.

Definition 16. *Given a $x \in D$, an edge-based connectivity opening Γ_x^E of element x (either a vertex or an edge) using the edge-based connectivity class $\mathcal{C}^E$ is defined as:*

$$\Gamma_x^E(D^E) = \cup_i \{A_i \in \mathcal{C}^E | v \in A_i, A_i \subseteq D^E\} \tag{3.8}$$

Note that the result of $\Gamma_x^E(D^E)$ migth include edges since they are part of the image X.

3.3.2 Edge functions

A stack of binary images with edges can represent a gray-scale image with valued edges by means of threshold superpositioning. In this way edge-based connectivity generalizes to gray scale in the same fashion as the mask based connectivity classes seen in section 3.2.6. As we shall see in section 4.1.4 this makes a combination of the two very practical. However, if we attempt to generalize edge-based connectivity to gray-scale we run into a slight problem: we have no values for the edges.

Threshold superpositioning for regular or mask based connectivity works on the basis of an image function $f(x)$ or a mask function $m(x)$, both with the domain D. Since we extended the domain to D^E we need to expand the image function as well. We do this by adding an edge function $\mathcal{E}(v_i, v_j)$ that returns a value based on two input vertexes. In practice, such an edge function can also be used to refer to a precomputed edge map. Instead of calculating the edge strength from the values from the properties of A_i and A_j, this map is constructed from any arbitrary information source. More on this in section 3.3.3.

One indication of the descriptive power of edge based connectivity is the fact that the use of different edge functions gives us different types of connectivity. For instance, we can use the maximal value edge function to emulate regular connectivity:

$$\mathcal{E}(v_i, v_j) = \max(f(v_i), f(v_j)) \tag{3.9}$$

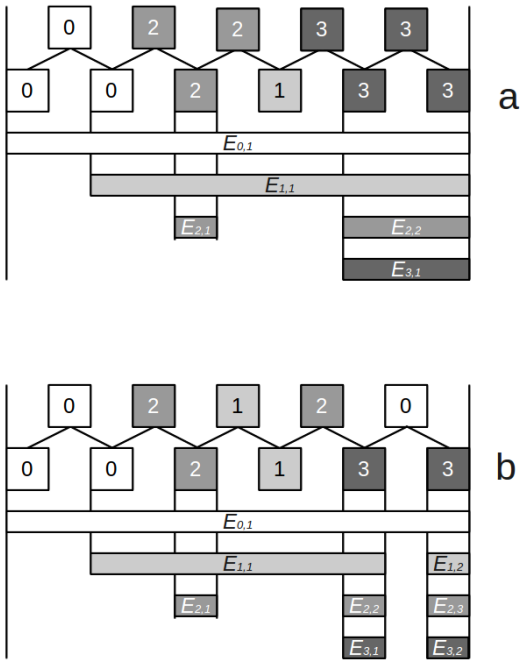

Figure 3.6: This figure shows the results of using the maximal value edge **(a)** and difference edge function **(b)**. Both sub figures show the signal on top divided in edges computed by using the function (top part of signal) and the vertexes which the edges connect (bottom part of signal). Below they show the various connected components that connectivity openings yield at various gray levels. They are labeled by level and by number on that level; for instance, $E_{2,3}$ would be the thirth component on the second level.

Substituting $f(x)$ for $m(x)$ in the previous edge function gives us the partitioning power of mask-based connectivity but not the clustering power. We can also use the difference edge function to split regions in areas of low contrast:

$$\mathcal{E}(v_i, v_j) = f(v_i) - f(v_j) \tag{3.10}$$

Figure 3.6 shows how these functions behave when using a one dimensional image.

3.3.3 Edge maps

Previously we considered only a simple localized version of $\mathcal{E}(A_i, A_j)$ that uses $f(x)$ to compute the edge strenght. We can also assume there exists some function $\mathcal{E}(A_i, A_j)$ that is based on a preconstructed map of all edge values. Such an *edge map* image could be computed by any means as long as it was static, thus no longer limiting us to using local information. This is exactly

analogue to the masks used in section 3.2.6 in the sense that for the final operation to be idempotent the edge-mask needs to be computed independently from the result of opening using the mask.

Chapter 4

System design

The content of this chapter shows how we used the theory from chapter 3 in our historical document processing system. We will start with an introduction to the Max-Tree data structure. The use of this data structure is an important part of this thesis as it provides us with a vital link between the theory of connectivity classes and the implementation of connectivity openings and filters based on connectivity openings and features of connected components, so called *attribute filters*. After this we will take a brief look at two such attribute filters: the k-subtractive and k-absorption filters. These are noise filtering techniques based on k-flatzones and the theory of hyper-connectivity seen in sections 3.2.7 and 3.2.8. Finally we will look at types of features that are very efficiently computed in Max-Trees, the normalized central moments. This chapter will only discuss the mathematics used to implement these components. Code listings can be found in the appendix. In the next chapter we will discuss experiments using these components in our image processing system.

4.1 Max-Tree

As discussed in chapter 3 the stacking of binary images with a decreasing binarization can be used in a process called threshold superpositioning to decribe a gray-scale image. Such stacks of binary images can be efficiently described by using the *Max-Tree* data structure as first described by Salembier et al. in [39]. Almost all of our operations will be performed in terms of these Max-Trees and it is therefore obvious that the algorithm itself deserves a thorough introduction.

A Max-Tree is primarily an image representation in which each pixel is assigned to a certain node. These *max-tree nodes* are not connected components themselves. Since threshold superpositioning connected components are nested they would contain a lot of redundant information. Instead, they contain only the elements that are added to the component on that specific level and a value that indicates what level that is. In addition they contain references of their children. The children of a Max-Tree node are all nested Max-Tree nodes that contain elements of a higher value. Figure 4.1 shows an example of a Max-Tree with nine elements spread over five nodes. There are many algorithms to construct these Max-Trees [16, 23, 39, 61], the version used in this research is listed in appendix A.

The use of the Max-Tree has some advantages when performing image operations. The main advantage of the Max-Tree representation is giving the user a very fast way to perform connectivity openings. The greatest cost of using a Max-Tree is incurred during the building of a Max-Tree;

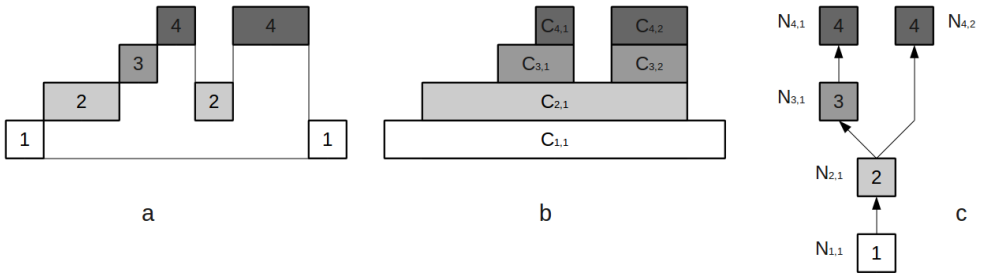

Figure 4.1: This figure shows how a Max-Tree represents a 1-dimensional signal: **(a)** shows what the signal looks like in a profile representation. Each of the elements shown is only present once in the Max-Tree. **(b)** shows what connected components should be returned at different levels on connectivity openings. Levels 1 and 2 have only a single component, levels 3 and 4 both have two components. **(c)** shows a graph representation of the resulting Max Tree. The bottom level node is node $N_{1,1}$ and contains both level 1 elements located at the beginning and the end of the signal. It has only one child: $N_{2,1}$. $N_{2,1}$ contains the three elements of level 2 located at the beginning and the center of the signal. It has two children: $N_{3,1}$ and $N_{4,2}$. There is no node $N_{3,2}$ since all the elements that would result from an opening on component $C_{3,2}$ are already represented in $N_{4,2}$. Node $N_{3,1}$ has one element and one child. The nodes $N_{4,1}$ and $N_{4,2}$ are peak nodes. They contain elements but do not have children.

performing connectivity openings in a Max-Tree is as simple as selecting the node that contains the opening element and collecting the union of all its elements and all those contained by its descendants. The other advantage of the Max-Tree is that it allows efficient computation of some types of connected component attributes. As connected components are nested in a Max-Tree, information from nested children components can be reused by their parents. It is for these reasons that it is preferable to perform as many operations as possible on using a single Max-Tree. Often the need to recompute the Max-Tree after a operation can be avoided by modifying the Max-Tree nodes themselves as shown in [28]. A slight disadvantage of the Max-Tree is that operations such as adding a single neighboring pixel to a tree or merging Max-Trees of two neighboring images are not trivial, as shown in [61].

4.1.1 Virtual Max-Tree elements

The Max-Tree building algorithm used in this research has no preference for topology. This means that it does not matter what connectivity class is used on the flooded image. We can use 4-connectivity, 8-connectivity or even more unusual forms such as the knight-connectivity briefly discussed in section 3.2.4. This is as simple as modifying the neighbor function which must comply to two criteria:

1. *symmetric* If A is a neighbor of B, B must be a neighbor of A.

2. *traversable* For every $x \in D$. There must exist a path from x to every other element of D.

As long as we comply by these criteria we can even mix different neighbor functions within the same image. For instance, one region of an image can use 4-connectivity while another region uses 8-connectivity. This would create a border area in which a mixed connectivity would be required to maintain the symmetry of the neighbor function.

An interesting way to use this property is by extending the image with a so called virtual image space [27]. This image space contains additional elements that have their own values and topology. The complete image topology together with the virtual image space define what sort of connectivity class we use. In the following subsections we will show two examples of how this works. The main difference between regular elements and the virtual image elements is that virtual elements are not considered when we compute attributes of a Max-Tree node. Only elements inside the original image can contribute to these. This makes it possible to compare attributes between different versions of the Max-Tree. It also means that we cannot compare Max-Tree nodes that contain only virtual elements.

4.1.2 Dual-input Max-Tree

In order to implement the mask based connectivity discussed in section 3.2.6 we use the dual-input Max-Tree demonstrated in [26]. The dual-input Max-Tree is built using two images instead of one, hence the name *dual*-input Max-Tree. One image represents the original image on which the connectivity openings are performed, the other represents the mask that modifies this connectivity. We place the image X in the same image space as the mask M. M is represented by virtual elements. We also modify the neighbor function; instead of connecting to each other the elements in X only connect to their corresponding element in M in the virtual image space. As shown in figure 4.2a, elements in M connect to each other in regular 4-connectivity, and of course back to the original pixel in X, remember the neighbor function needs to be symmetric. Elements in X can now only connect to each other through the mask. For more information on the dual-input Max-Tree see [26].

4.1.3 Edge-based Max-Tree

In order to implement the edge-based connectivity discussed in section 3.3 we can use the same scheme as in the previous section. We extend the image space with a virtual space, but instead of using a mask we fill it with edge weights essentially building an edge map. Elements in the original image now no longer connect to each other, nor do they connect to themselves, but as one can see in figure 4.2 they connect to edge elements. Every edge element connects to two elements in the original image but not to another edge. To express the edge function we use an edge map computed on the basis of X in a similar fashion to M in the previous section.

The size of the edge-map in the virtual image space grows quite rapidly with increasing connectivity with a worst case of $n^2/2$, where n is the number of pixels in the original image. For the more usual smaller cases every C edges in a connectivity, the image space needs to be expanded by $C/2$ times the original image to accommodate the entire edge map. The edge map is slightly smaller than the original image as edges that connect to elements outside the image space are of course forbidden. On large images, however, this slight saving in memory becomes insignificant. These memory issues somewhat constrain the amount of edges we can use. A grave disadvantage if one would want to use edges to connect structures at longer distances from each other. One option would be to make long distance edges sparse. We felt, however, that it would be difficult to make a

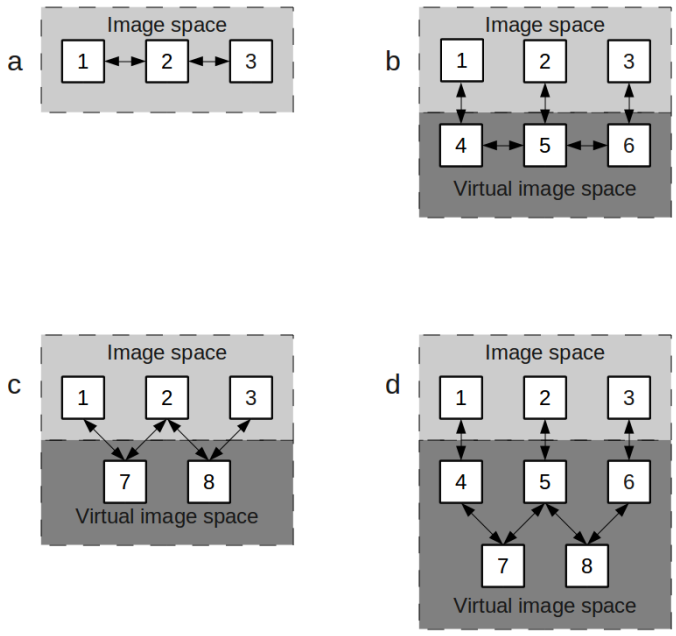

Figure 4.2: This figure shows the 4 variants of Max-Tree topology considered in this thesis using an example of a one dimensional three element pixel: elements 1, 2 and 3. **(a)** shows the neighbor relations in a regular connectivity. Each pixel is connected only to its neighbors. **(b)** shows the neighbor relations in masked connectivity. Elements in the image space, elements 1, 2 and 3, do not connect to each other but connect to their corresponding elements in the mask, namely 4, 5, and 6. **(c)** shows the edge-based connectivity as discussed in this chapter. Elements in the image do not connect to each other but only to edges. Elements near the end of the image, in this case elements 1 and 3 only connect to a single edge. Each of the edges connects to exactly two elements in the original image. **(d)** shows a combination of mask-based connectivity and edge-based connectivity. In this scheme every element in the image connects to the mask elements. The mask elements connect to each other by means of edges. This means that we can go from element 1 to element 3 without passing trough element 2 as would be the case in edge-based connectivity. It also means that we can cut off element 1 and 2 from element 3 simply by making the edge element 8 inaccessible, by giving it a lower value than elements 1,2 and 3. This will leave the mask elements unaffected.

segmentation based on this scheme stable. Another option would be to combine it with a different connectivity that has shown to have very little trouble in making connections over longer distances, for instance by combining local neighbor edges with masks.

4.1.4 Triple-input Max-Tree

As discussed in section 3.2.6 a disadvantage of mask based connectivity is its poor capabilities in contraction, specifically, that it cannot avoid removing a line of pixels when separating two connected components. The mask-based approach in clustering, however, is very elegant and works in most situations. In this light it makes sense to forgo contraction using mask-based connectivity and use it only for clustering. This means the mask image function $m(x)$ should always be larger than the image it is used on, i.e., $f(x) \leq m(x)$.

Section 3.3 has shown that edge-based connectivity works great with contraction and does not share the disadvantages that mask based connectivity has when splitting connected components. However, as shown in the previous section, long distance clusterings that are not sparse are somewhat of a problem in practice.

In order to combine both the properties of the mask-based dual-input Max-Tree and the edge-based Max-Tree, we designed a data structure called the triple-input Max-Tree. This is simply a combination of the two Max-Tree variants shown in the previous sections. In the triple Max-Tree the image is extended by two virtual image spaces: one that contains the mask, another that contains the edge weights. Note that while the mask is always the same size as the original image, the edge map is at least twice as big as the original image and could be much larger. Elements in the original image do not connect to each other but only to their corresponding virtual elements in the mask. The virtual elements in the mask, of course, connect back to their corresponding elements in the original image but they do not connect to each other. Instead they connect to edges in the second part of the virtual image space. These edges can represent any type of topology that maintains the two constraints we put earlier on the neighbor function. As before, an edge always has precisely two neighbors, however, a mask element can have many edges. We only examined the situation in which every mask element has four edges as memory and processing time constraints limit us in how many edges we use.

4.1.5 Line splitter edge map

The previous section does not explain how we give the edges their weights. In the next chapter we will use an algorithm to split lines as described in [54]. In the vertical profile, or histogram, of an image we use an algorithm that finds the peak intervals around which text lines are centered. A peak interval is an interval in which all values are at least a certain fraction α of the maximum of this interval. By means of hand tuning we determined that 0.1 is a good value for α in relation to our data set. Now using these peak intervals we find every line of vertical edges that is located in the middle between two peak intervals. If more than one line of vertical edges qualifies we take the upper one. These selected lines of edges are made impassable by setting their value in the edge map to 0. All other edges are made alway passable by setting their value to 255. Figure 4.3 shows how the interval finding algorithm was used.

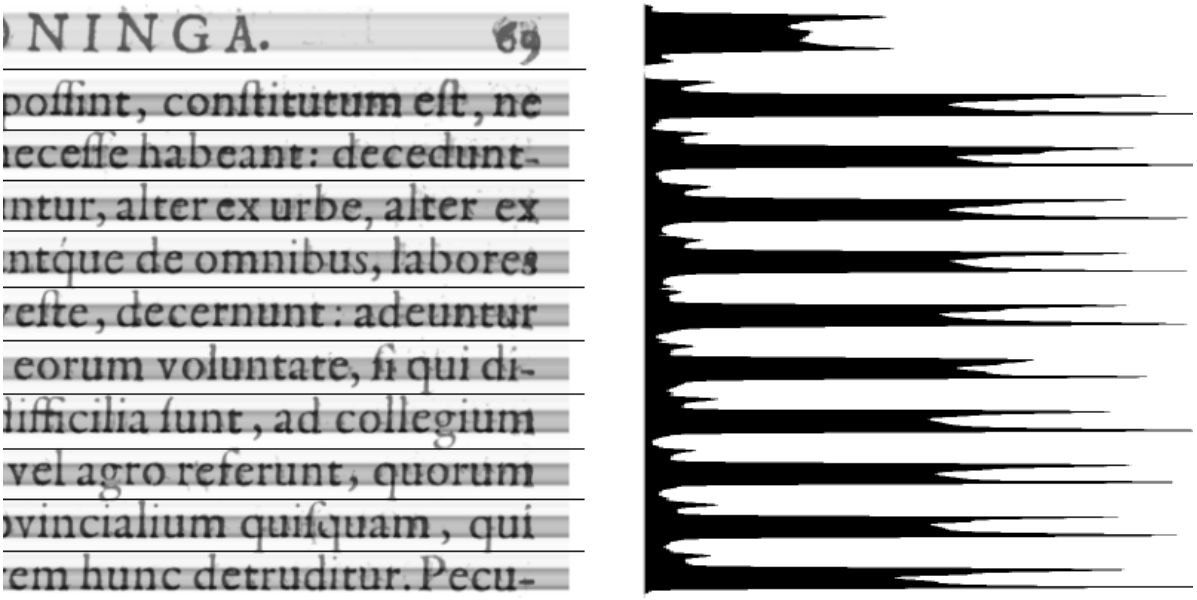

Figure 4.3: This figure demonstates the use of the peak interval as applied to a small section of a page. The right side of the figure shows a histogram of the amount of pixels on that line. Using the peak interval algorithm we find the line intervals, shown as gray background areas in the left of the figure, in the middle between every two line intervals we split the line. These splits are shown by black lines in the figure. Such a split is made in the edge map of the image and not in the images itself. Figure was taken from [54] and adapted for use here.

4.1.6 K-flat filters

In addition to the virtual Max-Tree elements mentioned in the previous sections, we also used a hyper-connected attribute filter based on k-flatzones. In practice, k-flatzones can be represented in a Max-Tree by implicitly or explicitly making groups of Max-Tree nodes. We could do this explicitly by taking every Max-Tree node and adding all its descendants that have gray values that are no more than k greater than the initial node. Each of these would result in a new k-flatzone. This means that a Max-Tree has an equal number of k-flatzones and nodes. We could also do this implicitly by using a recursive algorithm that determines in which k-flat zone it is working during the processing itself. Work published in [28] provides an efficient way to do this implicitly; we used the method described there thus avoiding having to reimplement our Max-Tree data structure and algorithm.

Using the k-flatzones explained in 3.2.8 it is possible to build attribute filters that can be used to filter background noise. We investigated two of these filters: the k-subtractive and k-absorption filter that are particularly good in filtering noise. These operations modify the k-flatzones in a Max-Tree based on two criteria: the peak criterion and an attribute criterion. The peak criterion is true if a k-flatzone contains at least one Max-Tree node whose descendants have at least one peak that is not part of the k-flatzone and has a gray value of at least k. In order to quickly determine this we extended our Max-Tree with an extra data member that contains the maximum gray value of all its descendants or its own if it has none.

The second criterion can be based on any attribute computed on a k-flatzone. It can even be an attribute based on a collection of other attributes. In this thesis we shall use an area attribute. This area criterion is true if the cumulative area of the Max-Tree k-flatzone lies between a minimum and a maximum vale. We defined the cumulative area of a k-flatzone as the area of the Max-Tree node with the smallest gray value in the k-flatzone together with the area of all its descendants, including those that are not part of the k-flatzone. This value is computed during the construction of the Max-Tree and as such is readily available. Figure 4.4 shows the results of the use of a k-absorption filter taken from [28].

The difference between the k-subtractive filter and the k-absorption is subtle but important. The k-subtractive *retains* every k-flatzone completely that satisfies both the peak criterion and the attribute criterion. Since k-flatzones overlap this means that some k-flatzones are only partially preserved. The elements that belong to Max-Tree nodes that are removed are added to their parents.

The k-absorption *removes* all k-flatzones completely that satisfy the peak criterion but fail the attribute criterion. After this it retains every k-flatzone that satisfies both the peak criterion and the attribute criterion. Zones that satisfy both the peak criterion and the attribute criterion but overlap with zones that fail their attribute criterion are therefore only partially preserved. Previous research showed that especially the k-absorption filter produces promising results in the context of document reconstruction.

4.2 Moments

As a way to measure the stability of our segmentation we will use a feature based recognition framework. As our main feature we will use *normalized central moments* as presented in [15] and [9]. Moments are a statistical measure of a geometric shape or a set of points. There are a great many different moments available, [9] describes geometric moments, central moments, normalized central

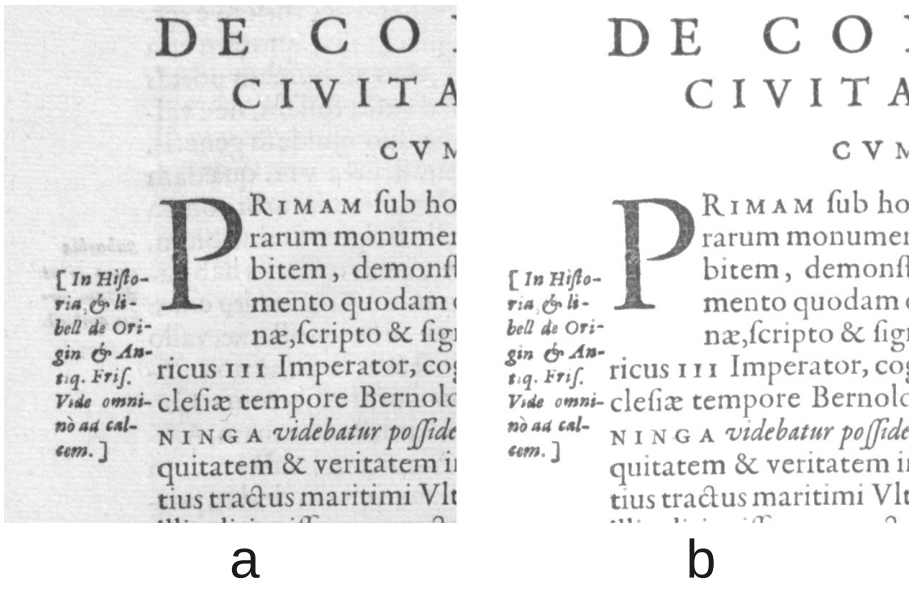

Figure 4.4: Processing a historical document: **(a)** shows the original image with much detail in the background. Filtering the image with the k-absorption rule at $k = 30$ results in **(b)**. This allows the suppression of details which touch the desired structures.

moments and affine invariant moments. They all share two characteristics: there are infinitely many of them and they can be used to reconstruct the original image. Although reconstruction algorithms exist for most of them, a complete reconstruction often requires an infinite amount of moments. It is this reconstruction property that makes them inserting features; we are guaranteed that if we have enough of them we have enough information to classify our geometric shape.

Work by [38] has shown these NCMs features give reasonable recognition results in the context of character recognition. As we shall show in the following subsections there exists a rather efficient way of computing these NCMs in the context of a Max-Tree. As seen in the definitions below there are two parameters to every geometric moment, in our case denoted as p and q. These are powers, taken in both the x and the y dimension respectively. The sum of p and q is referred to as the order of the moment. Of each order o there exist $o + 1$ different moments. A moment of order $o = 0$ is called to as the zeroth order moment. The moments we used were all originally used on continuous functions. For an implementation on images built out of pixels instead of mathematical geometry they needed to be converted for use on discrete image functions. Definitions in this section have been adapted accordingly. What the influences of this discretization are is reviewed in depth in section 5.2.

4.2.1 Geometric moments

The most basic building block of the NCMs are the geometric moments. Since geometric moments are neither translational nor scale invariant, they are features poorly suited for classification but as we shall see in the next subsections they can help in building features that do have these properties.

Definition 17. *Given some discrete binary image function f(x,y), the geometric moment $M_{p,q}$ is defined as:*

$$M_{p,q} = \sum_x \sum_y x^p y^q f(x, y) \tag{4.1}$$

Since Max-Trees describe nested binary images, there is a way to speed up the computation of geometric moments. Max-tree nodes are nested, this means that for Max-Tree node m the $M_{p,q}$ on m will be the sum of $M_{p,q}$ on its member elements added to the sum of $M_{p,q}$ on all its children. This makes computing geometric moments relatively simple and cheap within the context of a Max-Tree.

4.2.2 Central moments

On their own geometric moments are very poor OCR features since they are not translational invariant: the location of the described structure in the image space matters to the features. From the geometric moments, however, we can derive a type of moments that are translational invariant, the so called central moments. Central moments are geometric moments translated to center around their mean, thus making them translational invariant.

Definition 18. *Given some binary image function f(x,y), the central moment $\mu_{p,q}$ is defined as:*

$$\mu_{p,q} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q f(x, y) \tag{4.2}$$

The expression $(x-\bar{x})^p(y-\bar{y})^q$ produces a polynomial that expands in a predictable way. Using Newtons binomium we can rewrite the central moment definition in a way so that it is based on a

known collection of geometric moments. This new formula would be much faster to compute if we assume the geometric moments are known.

$$\mu_{p,q} = \sum_{k=0}^{p}\sum_{l=0}^{q}((-\bar{x})^{p-k} * (-\bar{y})^{q-l} M_{k,l})\binom{p}{k}\binom{q}{l} \tag{4.3}$$

We can even eliminate the need for computing the $\bar{x}$ and $\bar{y}$, which would require iteration through all members of $f(x, y)$, by computing them on a basis of the zeroth and first order geometric moments, which would be their sum and the cardinality of the set of elements on which they are based.

$$\mu_{p,q} = \sum_{k=0}^{p}\sum_{l=0}^{q}((-\frac{M_{1,0}}{M_{0,0}})^{p-k} * (-\frac{M_{0,1}}{M_{0,0}})^{q-l} M_{k,l})\binom{p}{k}\binom{q}{l} \tag{4.4}$$

The computational cost of this new equation grows only in the size of the powers p and q, instead of in the size of the image considered, and is therefore much more efficient to compute in the context of a Max-Tree.

4.2.3 Normalized central moments

We can expand the translational invariant central moments to be scale invariant as well. This means that if a letter is scaled to a different size or is acquired on a different resolution the resulting moments should be the same, although as we shall see in section 5.2 there are practical limits to this. Our data set contains only scans of a single resolution so there is no real need for this scale invariance property. It does contain various font sizes and capital letters. We hope that the scale invariance of the NCMs allows us to classify the different font sizes indiscriminately while not mixing up capital letters and their lower case counterparts.

Definition 19. *Given some central moment $\mu_{p,q}$ and the zeroth order central moment $\mu_{0,0}$ the normalized central moment $\nu_{p,q}$ is defined as:*

$$\nu_{p,q} = \frac{\mu_{p,q}}{\mu_{0,0}^{w}} \quad \textit{where } w = \frac{p+q}{2} + 1 \tag{4.5}$$

4.2.4 Invariant moments

Although there exists research that expands the theory behind the scale invariant NCMs to rotation invariant moments [15] or even to affine invariant moments [9], for our application these are not desirable properties. Rotation or even affine invariance would reduce the difference between certain characters and make some letters extremely hard to distinguish. For the 'p' and the 'd' for instance, the differences, if any, would be minimal. In his paper from 1994 [9], Flusser only describes four invariant moments but his derivation leaves room for finding more and many have improved upon his work in different ways.

4.3 Evaluation measures

Our system, given an input image, creates an annotation: a collection of labeled zones in an image. Our central interest is not only how many of those labels are correct, but how stable the character

zones that carry these labels are. Therefore we use metrics on the classification as a measure for this stability. We are also interested in over- and under-segmentation; is a zone bigger or smaller than it should be? In the context of segmentation of text, under-segmentation, the incomplete selection of a character, is a bigger problem than over-segmentation, the segmentation of more pixels than we think a character should include. Both situations would render a character harder to recognize. However, in the case of over-segmentation we can correct afterwards by discarding even more pixels, while in the case of under segmentation this is harder since we need to go back to the source image to reacquire lost information. Over- and under-segmentation often means that only portions of characters are missing and such errors should thus be measured in pixels. Therefore our recognition and data set statistics should be measured in pixels as well. When comparing two annotations we put every pixel in the image in one of four classes: Our system, given an input image, creates an annotation: a collection of labeled zones in an image. Our central interest is not only how many of those labels are correct, but how stable the character zones that carry these labels are. Therefore we use metrics on the classification as a measure for this stability. We are also interested in over- and under-segmentation; is a zone bigger or smaller than it should be? In the context of segmentation of text, under-segmentation, the incomplete selection of a character, is a bigger problem than over-segmentation, the segmentation of more pixels than we think a character should include. Both situations would render a character harder to recognize. However, in the case of over-segmentation we can correct afterwards by discarding even more pixels, while in the case of under segmentation this is harder since we need to go back to the source image to reacquire lost information. Over- and under-segmentation often means that only portions of characters are missing and such errors should thus be measured in pixels. Therefore our recognition and data set statistics should be measured in pixels as well. When comparing two annotations we put every pixel in the image in one of four classes:

1. A pixel that is annotated in both the ground truth and the computed annotation is considered a **true-positive**.

2. A pixel that is not annotated in the ground truth but is annotated in the computed annotation is considered a **false-positive**.

3. A pixel that is annotated neither in the ground truth nor in the computed annotation is considered a **true-negative**.

4. A pixel that is annotated in the ground truth but not in the computed annotation is considered a **false-negative**.

For the most important experiments in the next chapter we will use the F-measure static. The F-measure, or F_β-measure, is a statistic mainly used in two class problems in the field of information retrieval. It incorporates not only what fraction of the submitted inputs is labeled correctly but also how many false positives are returned. As such it is a great measure to define the quality of a segmentation. The F-measure is based on two other metrics, namely precision and recall:

Definition 20. *Given the number of true-positives, tp, false-positives, fp, and false-negatives, fn, the values precision p and recall r are defined as:*

$$p = \frac{tp}{tp + fp} \tag{4.6}$$

$$r = \frac{tp}{tp + fn} \tag{4.7}$$

Recall is also known as *hit-rate*, since it gives the fraction between true-positives, or *hits*, and all submitted positives comprised of true-positives and false-negatives. The result of a Leave One Out Cross Validation (abbreviated as LOOCV) with our classifier also yields a hit rate and can therefore be compared to the recall of our complete system. Using precision and recall we can define the F-measure:

Definition 21. *Given some precision p, recall r and value for β, the F-measure is defined as:*

$$F_\beta = \frac{(1 + \beta^2) * p * r}{(\beta^2 * p) + r} \tag{4.8}$$

Defining what a false-positive is becomes more complicated in multi-class problems. Therefore it is important to make a distinction between class-sensitive and class-insensitive cases when looking at the F-measures we receive from comparing two annotations. In the class-insensitive case we do not consider the labels for computing an f-measure. In the class-sensitive case, when a pixel appears in the ground truth and also in the computed annotation but has a different labeling there, it is considered a false-negative instead.

Most of a document image consists of pixels that are not part of any letter, this holds true regardless of font, language or even alphabet. In our scans of the Rerum only one in ten up to one in eight pixels on a page can be considered to be part of a letter. Therefore when considering a single pixel guessing it is empty will have a reasonable chance leading to a correct-negative. Secondly, in the context of document processing it is often more important to have a good recall than a good precision. If we incorrectly keep information that is actually irrelevant we can always discard it in a later step or some form of postprocessing. Based on these consideration we have chosen a beta value of 2 to be suitable for comparing results from our different experiments.

Chapter 5

Experiments

This chapter contains all the information about the experiments we did in our research. After some explorative trials we did several structured experiments to learn more about specific parts of our final recognition system. We start by looking a little more in depth at our data set. After this we describe two experiments where we attempt to learn more about the normalized central moments defined in definition 19. The first experiment deals with the scale invariance of these moments and how well this is maintained in discrete cases. The second one deals with the use of these moments in classifying characters. After this we will briefly examine if our training-set is big enough to generalize over to the test-set. In section 5.6 we will look at the use of k-flat filters as a means of segmentation. In the final section of this chapter we shall look at the combination of the k-flatfilters with different connectivity classes. We attempt to measure the stability of the resulting segmentations with our nearest-neighbor classifier that uses NCMs as features. Each of these sections starts with a description of the experiment and its context. This is followed by a report of the outcomes of the experiments and a numerical analysis of these. We conclude each section with a discussion of the results and their relevance.

5.1 The data set

5.1.1 Overview

For the purpose of the experiments listed in this chapter we established a ground truth for a limited portion of the Rerum. This was done by manually annotating a part of the Rerum using a Max-Tree based annotation tool. Every example in this data set is therefore either an original Max-Tree node or a combination of Max-Tree nodes. The annotated data set was split into two halves: a training-set, and a test-set. Both contain 10 pages. To avoid fitting on the test-set, all experiments except for the final one in section 5.7 are run on the training-set often using a LOOCV design. Figure 5.1 shows an example of one of the images from the training-set.

5.1.2 Quantitative analysis

One of the most important properties of any multi class data set is the ratio between types and tokens and their relative occurrence. A classifier cannot correctly classify examples of classes that

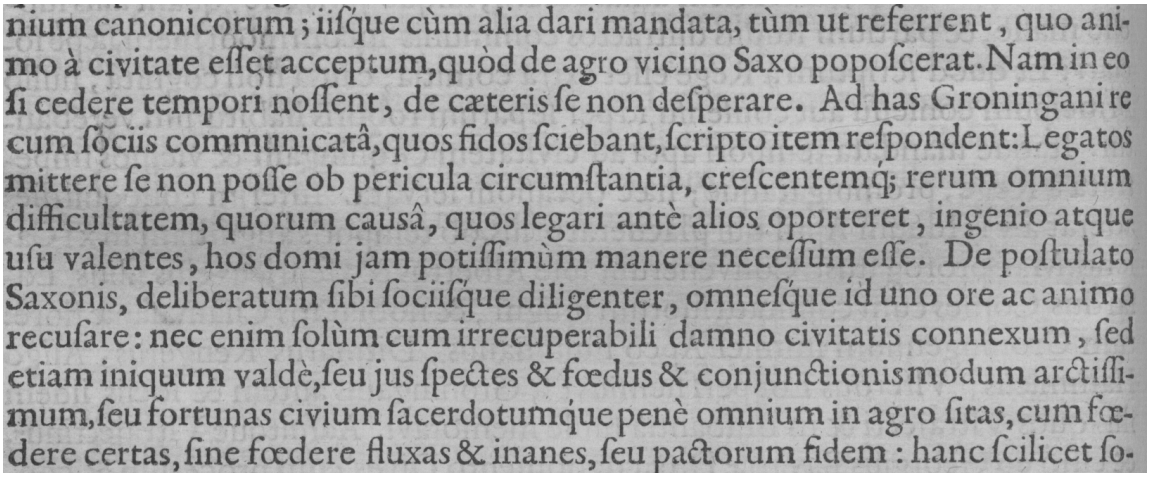
nium canonicorum; iiſq́ue cùm alia dari mandata, tùm ut referrent, quo animo à civitate eſſet acceptum, quòd de agro vicino Saxo popoſcerat. Nam in eo ſi cedere tempori noſſent, de cæteris ſe non deſperare. Ad has Groningani re cum ſo͂ciis communicatâ, quos fidos ſciebant, ſcripto item reſpondent: Legatos mittere ſe non poſſe ob pericula circumſtantia, creſcentemq́; rerum omnium difficultatem, quorum causâ, quos legari antè alios oporteret, ingenio atque uſu valentes, hos domi jam potiſſimùm manere neceſſum eſſe. De poſtulato Saxonis, deliberatum ſibi ſociiſq́ue diligenter, omneſq́ue id uno ore ac animo recuſare: nec enim ſolùm cum irrecuperabili damno civitatis connexum, ſed etiam iniquum valdè, ſeu jus ſpectes & fœdus & conjunctionis modum arctiſſimum, ſeu fortunas civium ſacerdotumq́ue penè omnium in agro ſitas, cum fœdere certas, ſine fœdere fluxas & inanes, ſeu pactorum fidem: hanc ſcilicet ſo-

Figure 5.1: This figure shows an example of one of the images from the training-set. Note the clear bleed-through effect from the back side of the page, and the smudges around some of the larger letter caused by the flow of ink during print. The image also contains many characters comprised of two or more parts and characters with diacritics.

do not occur in the training-set. The reject or noise class is an exception of course. Figure 5.2 shows the relative occurrence of characters in the training-set. All the characters together form 2431128 pixels of annotated text. Of those, 10 percent are part of characters that consist of two or more 'parts' such as 'i', 'j' , ':' '?' and letters with diacritics such as 'ò' or 'q́'. Over half of these pixels in *characters of interest* are the letter 'i'. This letter makes up more than 6 percent of our data measured in both pixels and tokens. Figure 5.2 also shows two multi letter ligatures containing the letter 'i' namely 'si' and 'fi'. In terms of tokens the letter 'i' is the second most common letter in our training-set. In terms of pixels it is the sixth most common letter in our training-set and probably in the *Rerum* as a whole.

5.2 Examination of scale invariance property

5.2.1 Context

During initial tests we noticed that the NCMs were not behaving as we expected. We decided to do some more in depth review of the scale invariance property of the NCMs. Note that NCMs were only derived in the continuous case in [9] and [15]. This means that a discrete implementation is bound to suffer from some numerical error. We wanted to investigate if these numerical errors led to numerical instability. We investigated the moments of orders two onto seven. We focus on these moments because they are most often used in literature, [3, 9, 15, 38] all deal with moments of these orders. NCMs of the zeroth and first order are irrelevant since they are always normalized to 0 and 1 respectively.

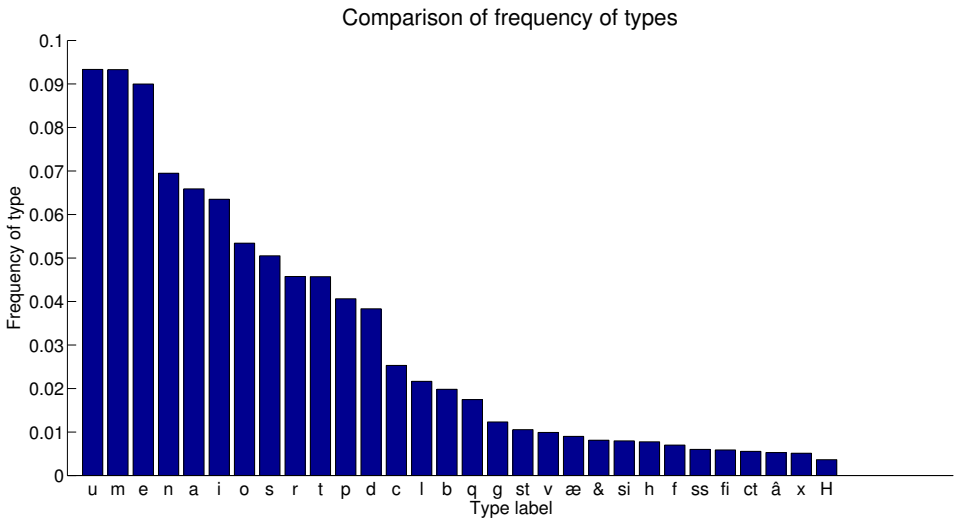

Figure 5.2: This figure shows the relative occurrence of the 30 most common characters in the training-set measured in pixels as a portion of the entire training-set. Types are sorted in order of frequency.

5.2.2 Results

To examine the scale invariance property we scaled originally high resolution images of characters from our data set to various sizes between 20 by 20 and 400 by 400 pixels. Then we thresholded them at a value that would clearly show the shape of the letter, in our case this was a threshold value of 170. For each image and each selected size we then computed the NCMs, and compared these to three reference sizes, 40 by 40, 100 by 100 and 200 by 200 pixels. We took the absolute difference between a moment and a reference moment devided by that reference moment as the absolute normalized error.

Figure 5.3 show the averages of the absolute normalized errors of all moments. Errors across the board are quite large, between 2 and 30 times the reference moment size. This is because the normalization used to compute NCMs tends to make the NCMs quite small and small absolute differences make a bigger impact; 95 percent of all NCMs generated by our data set lie between 0.001 and 0.1. Upon closer inspection most of the variance in the lower scale levels seems to be caused by fifth and seventh order moments. Figure 5.4 shows a closer inspection of all fifth order moments.

Figure 5.4 shows that within the fifth order the moments, $\nu_{2,3}$ is the main source of error. The other five moments are relatively stable. The large error seen in these fifth order moments led us to also investigate the differences in other order moments. The results of this can be seen in figure 5.5. Figure 5.5 shows a very mixed picture; the fifth and seventh order moments are the least stable, the second order moments are the most stable. The other moments all show similar levels of stability.

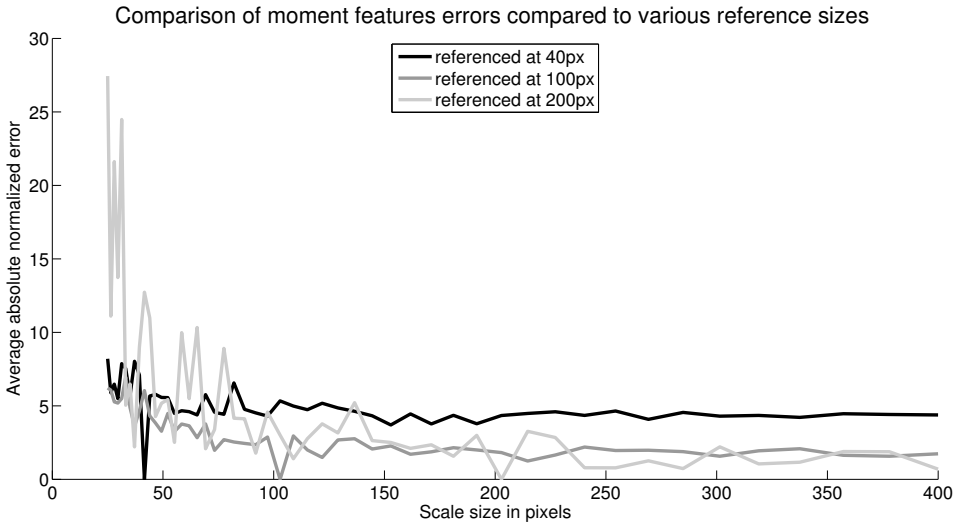

Figure 5.3: This figure shows the averages of the absolute normalized errors of all moments, compared to reference moments from three sizes; 40 by 40, 100 by 100 and 200 by 200 pixels.

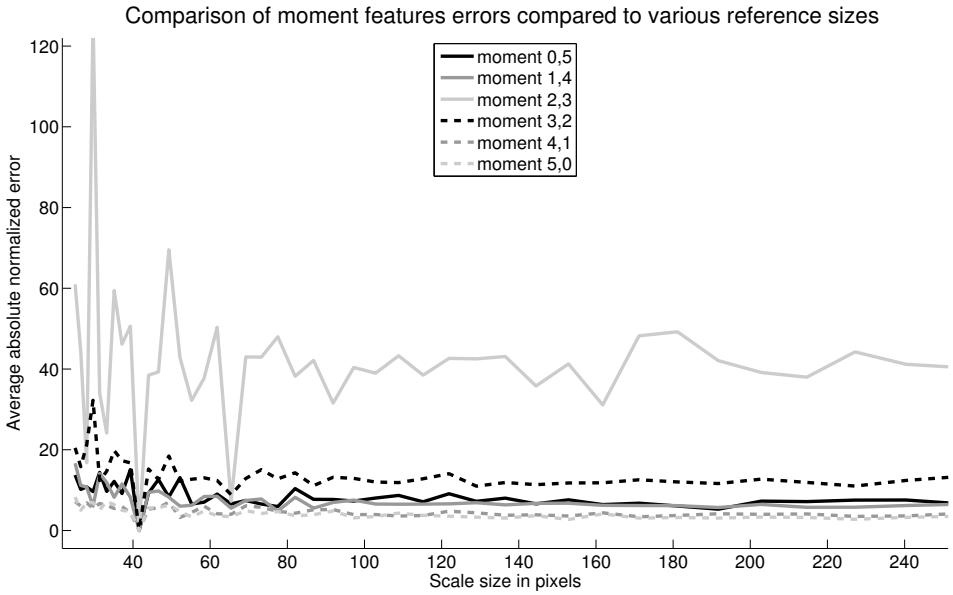

Figure 5.4: This figure shows the averages of the absolute normalized errors of only the fifth order moments, compared to reference moments from 40 by 40 pixels.

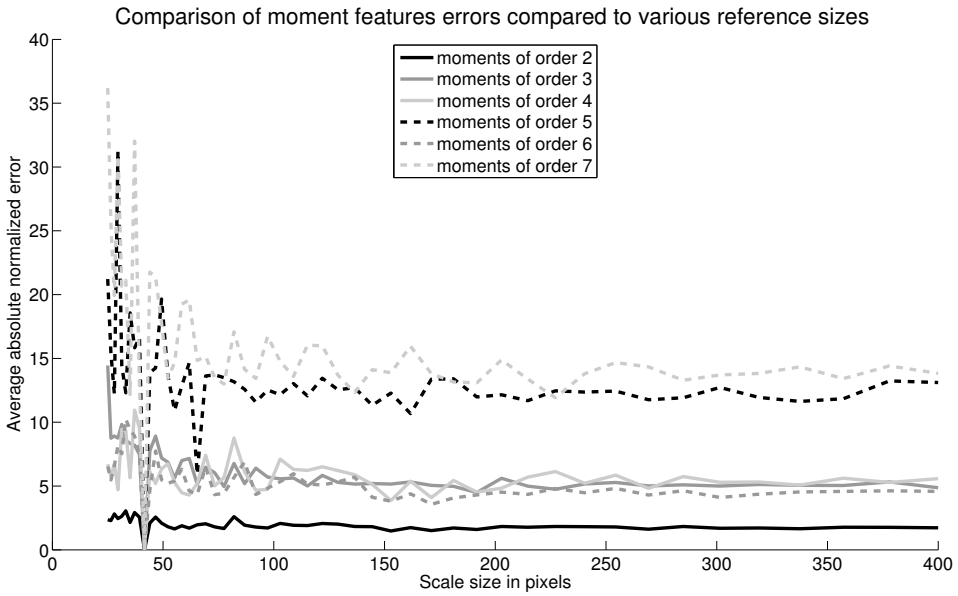

Figure 5.5: This figure show the averages of the absolute normalized errors of the moments of the second onto the seventh order compared to reference moments from 40 by 40 pixels.

5.2.3 Evaluation

There are several important lessons to be gleaned from these experiments. The most important of these is that the scale invariance of NCMs varies greatly between moments, but is very poor in general and especially so at the lower resolutions. Although the errors listed above are relative to the size of the reference moments, an error factor of 30 is too great on a value interval from 0.001 to 0.1 if there are many classes. In general, moments calculated on a high scale level tend to generalize better than those from lower scale levels. Put otherwise, it is easier to classify a high resolution character based on features from a small one than vice versa. This might be contrary to what one would expect since a large character would contain redundant information. Note that most of our data set contains letters of roughly 40 by 40 pixels. Around that scale level all moment orders, with the exception of the second order moments, have error levels of 10 to 30 times the reference moments. This means that unless we use only second order moments we cannot depend on the NCMs to be truly scale invariant.

5.3 Feature set examination

5.3.1 Context

Having established the limits of the scale invariance property of NCMs we wanted to have some idea how distinctive NCMs are as features. Earlier reports suggested recognition rates reached up to 93 percent in near optimal conditions [38]. Different research on traffic sign recognition in natural

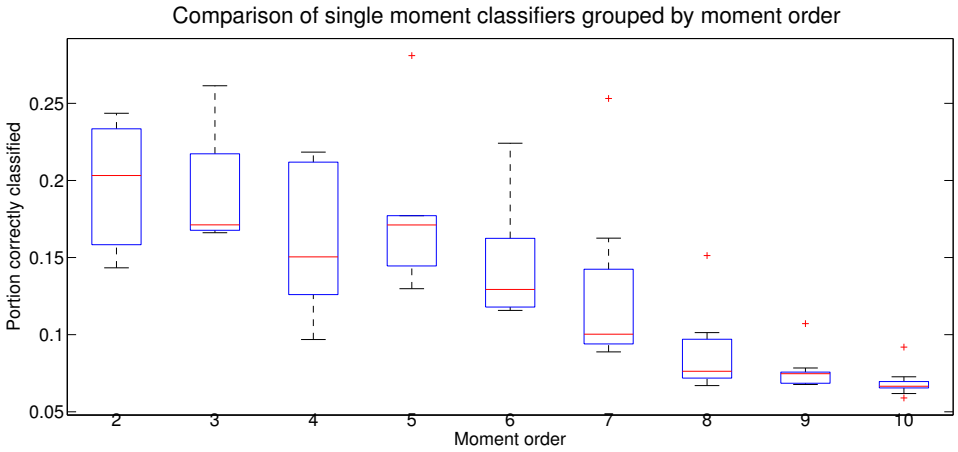

Figure 5.6: This figure shows results of LOOCV using a nearest neighbor classifier based on a single moment. The results were thereafter grouped by order of the moments. The results from each of these groups is represented by a box plot.

scene images [3] reported recognition rates from around 50 percent to up to 80 percent using similar methods, but higher order moments. This emphasizes the importance of good segmentation when using moment based features in complicated images.

Even though the Rerum is not the most difficult case of hand-print recognition, we were worried that in less than optimal conditions NCMs might not be strong enough features to produce any interpretable results. In other words we were worried that the within class variance would be much bigger than the variance between classes, thus preventing us from getting clear differences between the different types of connectivity.

The NCMs are interesting on a more fundamental level as well. Since most moments can be computed effectively in the context of a Max-Tree, they are of key interest to users of this data structure. We were hoping to learn more about how they performed outside the optimal conditions represented in [9,38] and how they can best be used. NCMs are individually normalized in scale and all lie between -1 and 1. Within these bounds, however, they vary greatly in scale; some are close to 0.5 in size but most of them are of sizes between 0.001 and 0.1 (positive). Since our nearest-neighbor classifier relies on euclidean distance within a kd-tree for efficiency reasons, it benefits from another normalizing feature space transformation. For the remainder of the experiments in this chapter, including the one described in this section, we used a percentile normalization that linearly transforms the interval between the 10th and 90th percentile to the unit interval. This ensures that features cannot dominate on basis of scale alone.

5.3.2 Results

Looking at figure 5.6, moments seem to become weaker features (on average) as the order of the moments increases. The relation between moment order and classification score cannot be linear, because the moment score is limited between 1 and 0 but there are an infinite number of moment

orders. However, a Pearson product reveals that there is in fact a very significant correlation between the two ($df = 61, t = -9.53, p < 0.01$). The strongest moment in general was not a second order moment but a fifth order moment, $\nu_{0,5}$. A classifier built with only $\nu_{0,5}$ scored more than 28% correct with LOOCV; as such it was the first moment chosen when building our optimal moment set as shown in the next experiment.

5.3.3 Evaluation

Contrary to what we saw in our examination of the scale invariance property, large moments seem to vary more when evaluated on classification. In general the lower moments produce the greatest hit rates. These results make a lot of sense if looked at from an image reconstruction perspective. As mentioned before in section 4.2, any image can be completely reconstructed from its moments, in the discrete case this can even be a finite set of moments. When building such a reconstruction the moments with lower order will represent the coarse grained structure of the image and the higher order moments represent the finer details. This would indeed mean that a classifier based on only lower order moments would yield a higher hit-rate than one that uses only higher order moments.

5.4 Building an optimal moment set

5.4.1 Context

The previous experiment confirmed our expectation that lower moments were stronger in general. It might seem intuitively correct to build a classifier with only lower order moments. A classifier with only lower order moments might, however, contain redundant information. We wondered what would be the set of moments that would give us the best classification. Even for a limited moment order m this would require us to try all different combinations of moments. The number of possible classifiers would be in the order of $2^{((m+1)^2/2)}$. Trying them all is obviously impractical for even small values of m. As an alternative we used a greedy search algorithm. The algorithm starts with a classifier based on the strongest single moment and keeps on adding the moment that gives it the greatest increase in its performance. It does this until no single moment can be found that improves the classifier. We again used LOOCV as a performance measure.

5.4.2 Results

We ran the greedy search algorithm over using all moments of orders 2 onto 10. The resulting moment set is listed in table 5.1. The first four moments added already give us a performance of over 80 percent. After that, performance gains quickly diminish. Note that the last 5 moments together do not add more than a single percent to our LOOCV score. Because the greedy moment search algorithm yielded mostly moments of orders two, three and four (11 out of 15) we also tried a classifier with only those moments which got a fraction of 0.9356 correct, only slightly less than our greedy-search moment set.

5.4.3 Evaluation

Although we cannot guarantee that this procedure would yield the truly *optimal* moment set, we believe the set of moments shown in Table 5.1 is close to the limit what a simple nearest-neighbor

number to be added	moment	score
1	$\nu_{0,5}$	0.2809
2	$\nu_{2,0}$	0.5047
3	$\nu_{1,3}$	0.7131
4	$\nu_{2,1}$	0.8205
5	$\nu_{0,2}$	0.8758
6	$\nu_{4,2}$	0.9163
7	$\nu_{3,0}$	0.9230
8	$\nu_{1,1}$	0.9269
9	$\nu_{2,2}$	0.9295
10	$\nu_{0,3}$	0.9315
11	$\nu_{4,0}$	0.9338
12	$\nu_{3,1}$	0.9346
13	$\nu_{3,2}$	0.9359
14	$\nu_{3,3}$	0.9362
15	$\nu_{0,4}$	0.9366

Table 5.1: The result of our greedy moment search

classifier can reach using NCMs. If we look at the moments selected it may be unsurprising that they contain all moments from second onto fourth order except for $\nu_{1,2}$. In the first 10 selected moments the only moments of an order larger than 4 are $\nu_{0,5}$ and $\nu_{4,2}$. This falls in line with our earlier evidence that lower moments are stronger features. Another striking trend is that the selected moments seem to alternate between moments that contain mostly vertical information, such as $\nu_{0,5}$ or $\nu_{1,3}$, and moments that contain mostly horizontal information, such as $\nu_{2,0}$ or $\nu_{2,1}$. These categories should supplement each other in theory but the strict alternation in the first six moment was unexpected, because there is no fundamental reason why letters should contain as much information in both dimensions.

As we noted in section 4.2.3, in order to compute a moment of the order m we require all geometric moments of the orders 0 onto m. Computing only the geometric and scale invariant central moments up to order 4 makes the process less computationally intense and makes our results in the next sections more transparent. For the rest of this thesis we shall use a classifier based on all the NCMs of orders two, three and four.

5.5 Examination of training-set size

5.5.1 Context

The number of classes and their frequency distribution shown in the first section of this chapter made us worried that we might not have enough data to train our classifier. A classifier with insufficient data might do well on data it is trained on but its performance would degrade rapidly once it encounters new data. This new data might contain classes it was not trained on. Although unknown classes cannot be classified correctly, the fact that they are not present in the training-set makes it likely that they are exceedingly rare. This would seem to be even more true in the case of

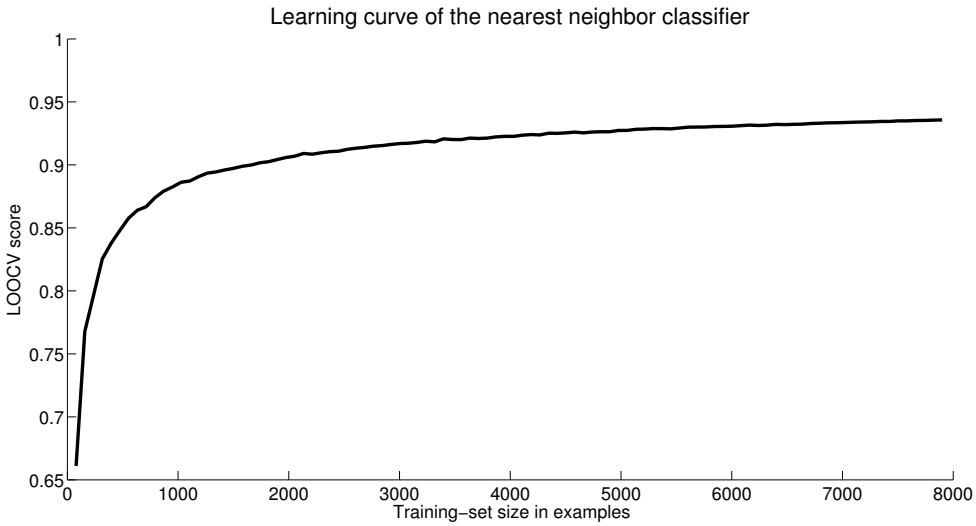

Figure 5.7: This figure shows the result of LOOCV using training-sets of different sizes.

hand-print recognition. There were only a limited number of characters available to the typesetters, and since our training-set and test-set were taken from random portions of the Rerum there should be a good chance that the training-set contains most of them. During the annotation process we were surprized by the large variety of extremely rare or seemingly single use characters that were present in the Rerum. This is emphasized by the fact that for more than half of our classes we have 10 or fewer examples.

A good way to examine the unknown classes problem is to look at the change of the LOOCV performance of the classifier as we change the amount of data it is trained with. A classifier that is still rapidly improving when it was trained on the full training-set is often prone to rapidly degrade its performance when it encounters new data. If its performance is stable at that point there is apparently very little left that it has not seen, and it can most likely generalize from the training-set to the test-set.

In order to find if our training-set was large enough to generalize we used LOOCV on random subsets of the training-set. In small increments we slowly increased the data used to train the classifier from nothing to the full training-set containing 7982 tokens. The classifier used the second, third and fourth order moments to classify the examples.

5.5.2 Results and evaluation

Figure 5.7 shows how our classifier improves as we train it with more data. The plot shows that after 4000 examples hardly any improvement occurs. Although our classifier is not fully saturated and could benefit from more data, we deemed its accuracy sufficient for our research purposes. We were also satisfied to see that our classifier reached the 0.93 hit-rate attained in [38]. This also means that 93 percent is our upper limit of recall in our experiments with connectivity classes later in this chapter.

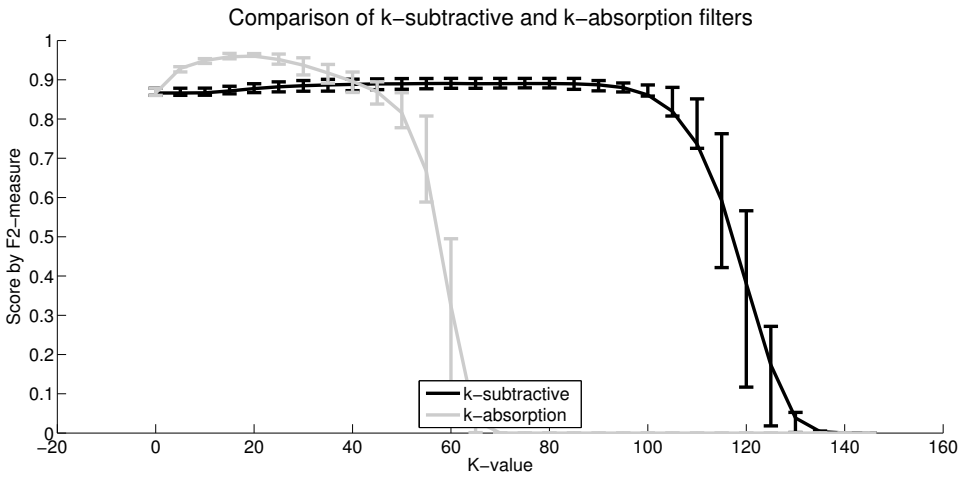

Figure 5.8: This figure shows results of both k-flat filters with k-values from 0 onto 150. Error bars show the ends of the first and third quartile.

5.6 Examination of k-flat filters

5.6.1 Context

As a means of segmentation we tried the two hyper-connected filtering methods described in section 4.1.6. Using the algorithms described in [28] both the k-subtractive and k-absorption filters can be implemented in an ordinary Max-Tree data structure (barring the addition of some extra data members in the node).

In this experiment we had two questions: we wanted to know which filter was better suited for segmentation of our data, the k-subtractive or the k-absorption filter, and we wanted to know what value of k would give the best segmentation. Since we only want to retain nodes that could possibly be characters we also filtered all nodes bigger than the biggest character in the training-set and all nodes smaller than the smallest character in the training-set.

After filtering we rebuilt a new Max-Tree based on the filtered image. We did this because our implementation changes the output values and not the structure of the Max-Tree (see [28]). In this newly constructed Max-Tree all direct children of the bottom level node are considered to be characters. In order to quantify the quality of this segmentation we make a class-insensitive comparison between the newly found character nodes to the hand made annotation and compute an F2-measure.

5.6.2 Results

Figure 5.8 shows the results of our filters used on the training-set. The variation in score was shown in a percentile measure, means are calculated over the whole training-set of 10 pages. The k-absorption filter has a maximum performance at a k-value of 14, scoring an F2-measure of over 0.99, the k-subtractive filter is best at a k-value of 86 where it scores 0.88 on the F2-measure.

5.6.3 Evaluation

In the introduction we noted that results of these filters in other fields of application seemed quite promising, and indeed they perform quite reasonable on our data as well. From figure 5.8 it is quite clear that the k-absorption filter outperforms the k-subtractive filter by a fair margin. We believe this is because the k-absorption filter is more context sensitive than the k-subtractive filter. As noted in section 4.1.6 the k-absorption filter also propagates the reject decisions while the k-subtractive filter only propagates the restorations. These results mostly confirm the findings in [28] although optimal values of k vary between data sets. In our experiments the k-absorption was so much stronger than the k-subtractive filter that we decided the second one was no longer interesting for further testing. It was therefore not considered in the final experiment.

5.7 Comparing connectivity classes in a complete image processor

5.7.1 Context

In the previous experiments each time we looked at individual portions of our system. We showed that we could not depend on the scale invariance property at our scale level. After this we established that a combination of all second, third and fourth order moments yields a classifier that is close to optimal. We then showed that our training-set was large enough to generalize and that a combination of an area filter with k-absorbtion filter preprocessing will give us a good segmentation. Now we wanted to build a complete image processor in which we could compare different image topologies.

For these experiments we used a nearest-neighbor classifier trained with the annotated examples in our training-set. As a preprocessing step we filtered the image using the k-absorption filter with a k-value of 14.0 and an area attribute criterion based on the smallest and largest examples in the training-set. This filter was used on the specific Max-Tree type used for that experiment, either the dual-input, the triple-input or the regular Max-Tree. We then segmented our character zones by selecting all children of the bottom level Max-Tree. Labeling these character zones using the before mentioned classifier then results in a computed annotation. This annotation contains a class labeling for every pixel that is part of a character. Pixels not labeled in the computed annotation are considered to be part of the background. This computed annotation is then compared to a manual ground truth annotation as described in section 4.3. A visual representation of the entire process used in this experiment can be seen in figure 5.9.

Our aim was to compare three different connectivity classes as they are expressed by various Max-Trees:

1. **The vanilla Max-Tree**. This is a simple Max-Tree with no added virtual elements. This will be the baseline for our experiments. It has no parameters other than the parameters we use for the area filter and the moment transform.

2. **The dual-input Max-Tree**. The dual-input Max-Tree uses a mask-based second generation connectivity described in section 3.2.6. As a mask we shall use a structurally dilated version of the original image; for the dilation we shall use a vertical line with the origin located on the bottom most pixel as a structuring element. Our hope is that dilation will be able to connect the various components of our separated characters. The size of the line is a parameter in this

experiment, we expected it to have a sweet spot where it is large enough to connect letters to their dots and diacritics but not yet so large that it connects different lines. Beyond that, the score would be decreased by the influence of unwanted merges.

3. **The triple-input Max-Tree**. This is in many ways the same structure as the dual-input Max-Tree. It has an image that does not connect locally but through a mask that is formed by the dilation with a line shaped structuring element. However, instead of having the mask connect directly to neighboring elements in the mask it does this through edges. The edge weights are determined by the algorithm discussed in section 4.1.5. We expected this structure to prevent unwanted merges between lines, allowing us to further increase the dilation without those merges occurring.

5.7.2 Results

We first used the 'vanilla' max tree to establish a baseline. This yielded a recall score of 0.81. After this we tried the dual-input Max-Tree with a mask built by a dilation of the original image with a structuring line element. We tried line elements of different sizes. The optimum was found at a line element size of 7 pixels where it scored an F2 score of 0.83 and a recall score of 0.81. We wanted to compare this against the triple-input Max-Tree for the same dilation sizes. This comparison is shown in figure 5.10.

We were of course also interested in how these Max-Trees influence the segmentation of our *characters of interest*: the letters that are comprised of two or more vertically separated parts. We will do this by removing all letters that are comprised of only a single component from the computed annotation and the manual annotation and compare these stripped annotations to compute a recall score. Since both annotations are now almost completely empty, a precision statistic becomes even less meaningful than it is in the case of the full annotations (for more information, see section 4.3). Table 5.2 therefore only lists recalls on various conditions. It shows that, not considering classes, the recall is nearly perfect, meaning that there is very little information lost in preprocessing.

Aside from the numerical results it is also important to visually inspect the computed annotations and attempt to find the most prominent successes and mistakes. The next section will begin with an inspection of our intended target: the letters that have multiple vertically separated components. After this we will look at some of the problems that remain unaddressed by our methods. When we look specifically at the letters that first prompted us to look for alternative connectivity classes we see some very promising results. Figure 5.11 shows how the different Max-Trees do indeed make it possible to classify the vertically separated letters correctly.

Vertically separated components belonging to the same character are not the only issue faced when segmenting the Rerum. As a result of our method we also saw the rise of some unintentional merges and splits that were not solved by the triple-input Max-Tree. Figure 5.12 tries to illustrate some of the most common problems that remain unaddressed by our methods.

5.7.3 Evaluation

The results of the comparison between the dual-input and the triple-input Max-Tree shown in figure 5.10 are interesting in several ways. The most striking feature in this plot is the critical point at a dilation size of 15 pixels. This is the distance at which ascenders and descenders start to merge with regular characters on different lines. Ascenders merge with descenders much before this point but a

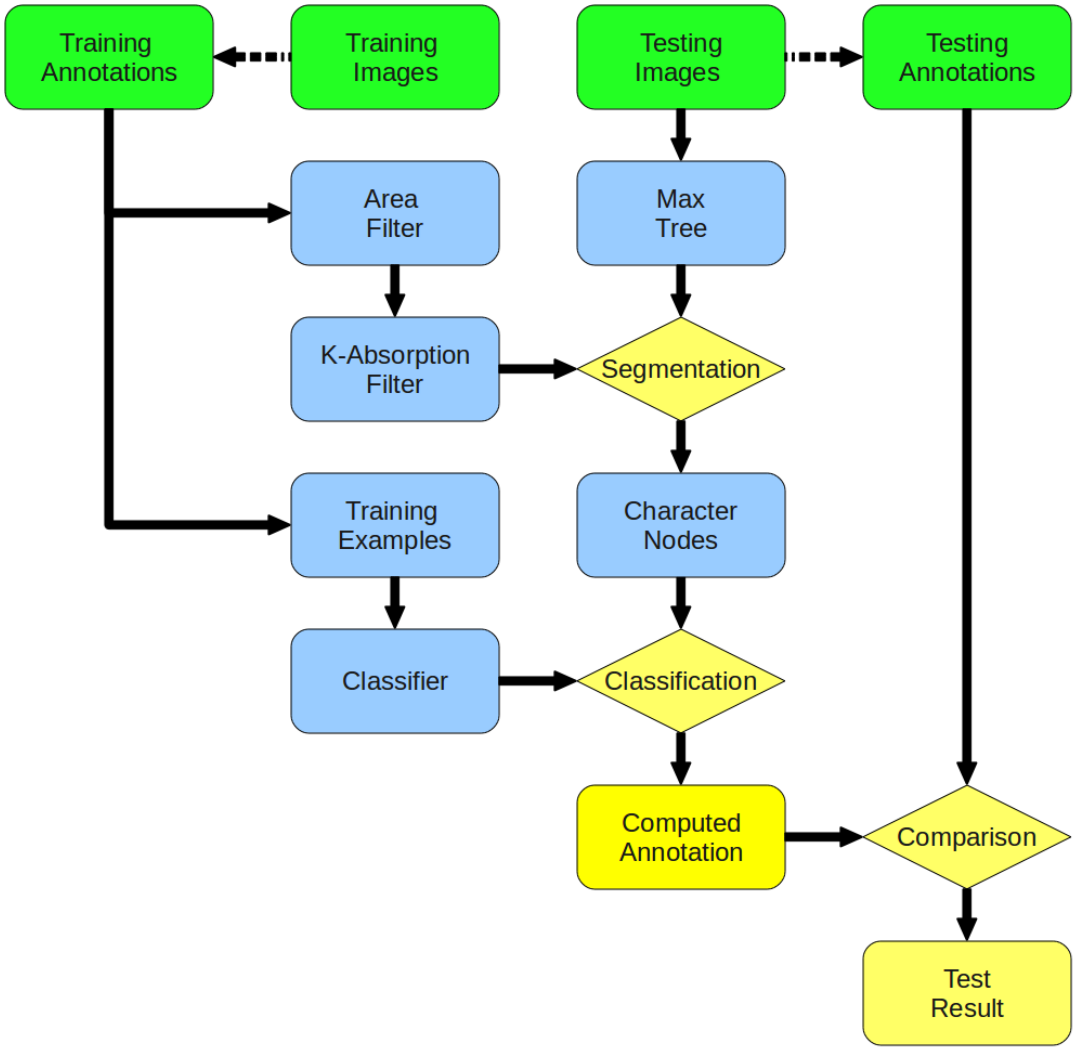

Figure 5.9: This figure shows the entire process used to produce the scores as presented in table 5.2. Dotted arrows mean the process was carried out by hand.

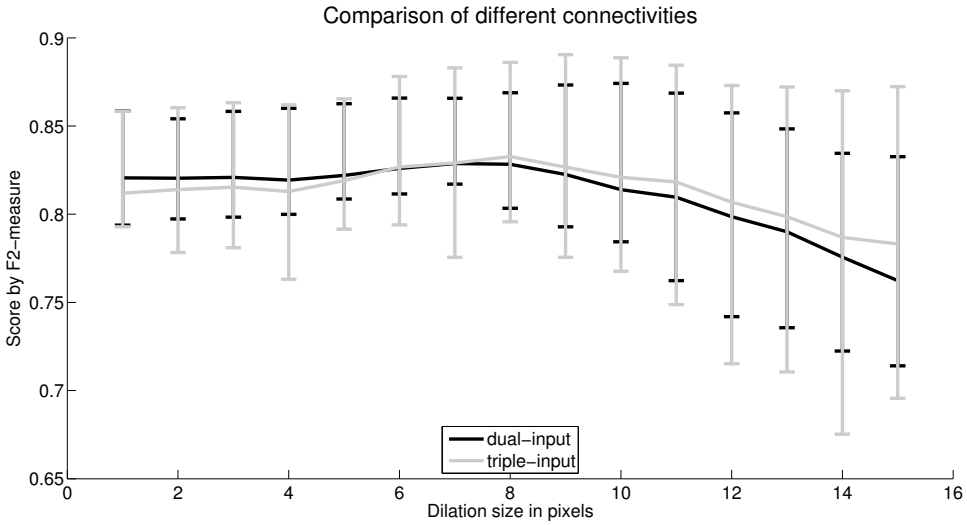

Figure 5.10: This figure shows a comparison of the F2-scores of the dual-input Max-Tree and the triple-input Max-Tree on the test-set with dilation sizes from 1 onto 15. Error bars show the limits of first and third quartile.

Max-Tree type	dil. size	classless recall	classed recall	'i' recall	COI recall
vanilla	-	**0.99 (0.99-1.00)**	0.81 (0.78-0.85)	0.33 (0.13-0.44)	0.25 (0.10-0.35)
dual-input	4	0.99 (0.99-0.99)	0.80 (0.78-0.84)	0.33 (0.29-0.41)	0.31 (0.27-0.41)
dual-input	8	0.99 (0.99-0.99)	0.81 (0.78-0.86)	0.73 (0.65-0.86)	0.63 (0.49-0.73)
dual-input	15	0.97 (0.97-0.99)	0.73 (0.67-0.80)	**0.81 (0.72-0.91)**	0.74 (0.67-0.88)
triple-input	4	0.99 (0.99-0.99)	0.80 (0.73-0.86)	0.33 (0.27-0.41)	0.30 (0.24-0.41)
triple-input	8	0.99 (0.98-0.99)	**0.82 (0.77-0.88)**	0.70 (0.58-0.86)	0.61 (0.50-0.73)
triple-input	15	0.97 (0.97-0.99)	0.75 (0.65-0.85)	0.80 (0.59-0.93)	**0.74 (0.53-0.91)**

Table 5.2: This table shows the results of our experiments with the complete document processing system. The numbers in this table are recall scores. Every triplet starts with the mean value followed by the 1st and 3th quartile as an indication of the spread. Bold scores are the highest in their column. The first column indicates what kind of Max-Tree was used. The second one indicates the vertical size of the structuring line element used for dilation (if any). The third column shows the recall rates regardless of class. The fourth column shows the recall rate if we also include classifications in the results. The final two columns show the recall scores for both the 'i' characters individually, and all *characters of interest* respectively.

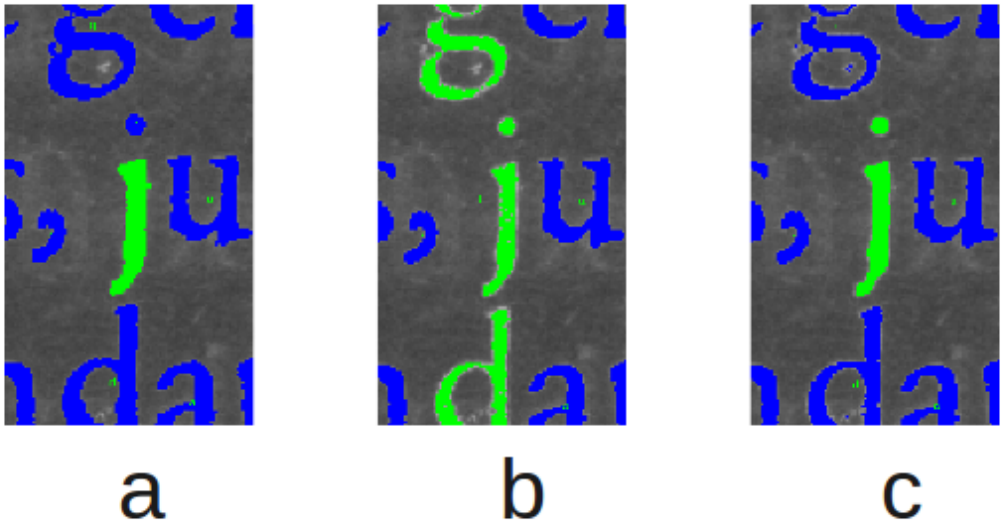

Figure 5.11: This figure shows a comparison of the same section of an image containing the letter 'j' as it is annotated by the three different Max-Trees. **(a)** shows the result of the ordinary Max-Tree. It has no way to connect the dot and the rest of the letter 'j' and classifies the dot as a period and the rest correctly as the letter 'j'. Even without its dot the letter 'j' is apparently special enough to get classified correctly. **(b)** shows what happens when we use the dual-input Max-Tree on the same region. Because this area of the image contains vertically aligned ascenders and descenders, merges between lines arise which makes correct classification impossible. **(c)** shows how the problems of the dual-input Max-Tree are counteracted by cutting the line apart by means of edges.

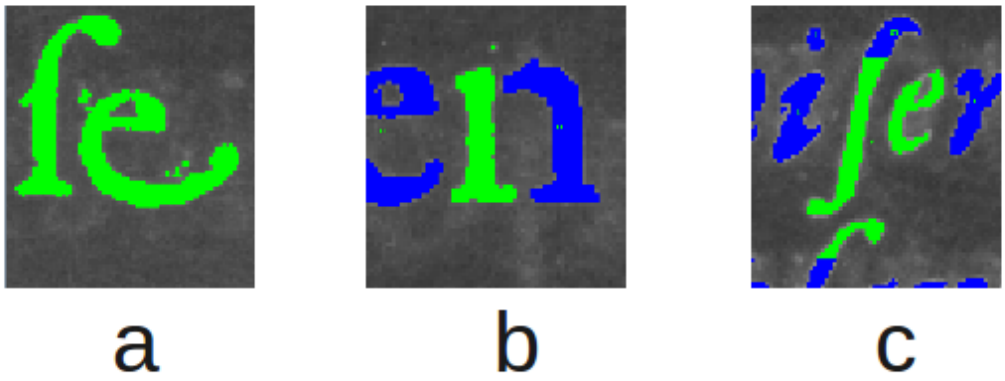

Figure 5.12: This figure shows examples of some of the segmentation problems that the system is not equipped to solve. **(a)** shows the result of an unintentional clustering of two characters. In this case the dilation merges the letters 's' and 'e'. This happens because some letters 'invade' other letter's vertical space, this is common with the letters 'f' and 's' and in the italic fonts. **(b)** shows an unintentional split in the letter 'n'. As a result of the poor quality of the print of the Rerum, letters sometimes get split in two or more horizontally separated portions. This happens most often in the letters 'm', 'n' 'u' 'c' and 'h'. Is is also aggrevated by the k-absorption filtering. **(c)** shows a problem that arises when the scanned image of a page has a slight rotation. In these cases our line splitting algorithm has problems finding the separations between lines. Moreover, it cannot find diagonal line splits and many letters are split into several portions often resulting in misclassifications. This problem is worse on the italic fonts.

vertical alignment as shown in figure 5.11 is extremely rare. Both Max-Trees seem to perform best with dilations between 1 and 10 pixels. On the interval of dilation sizes of 1 to 6 pixels a Wilcoxon summed rank test finds no significant differences between the two populations ($N = 50, \alpha = 0.05$). At 6 pixels the triple-input Max-Tree is better for the first time and on the interval from 6 to 10 pixels a Wilcoxon summed rank test finds it is significantly better than the dual-input Max-Tree ($N = 50, \alpha = 0.05$). Beyond this point both seem to run into problems and performance quickly degrades.

Table 5.2 shows very mixed recall rates on all different Max-Trees and dilations. On the optimal dilation (8 pixels) for both the dual-input and the triple-input Max Trees, the triple-input Max-Tree beats the dual-input Max-Tree by a small margin (although the triple-input Max-Tree's variance is higher) but if we look at our *characters of interest* at that dilation level, the errors made while cutting the line apart causes enough damage to push its recall under the rate of the dual-input Max Tree at that level. If we increase dilation to the point where it becomes a serious problem when considering all classes (15 pixels), they perform more or less equal.

Two important conclusions can be drawn from these observations: Most importantly, mask based connectivity can help a lot in the segmentation of letters that are comprised of multiple components. Recall rates are dramatically increased over the vanilla variants by means of the dilation. Even using the simplest operation to create a mask yields decent results. Secondly, our edge-based cutting technique does not improve very much upon this mask based scheme. While it protects us ever so slightly from unintentional merges between lines, the damage it does in cases such as the one seen in figure 5.12.c is so great, that the cost almost outweighs the gains. This is without factoring in the extra cost of the triple-input Max-Tree compared to the dual-input Max-Tree in terms of memory and computation.

When looking from a qualitative perspective one of the biggest problems that remained unaddressed was that of the unintentional merges caused by the letters 'f' and 's' as shown in figure 5.12. There are a great many combinations in which they merge with the neighboring letters. For the letter 'i' we made special combined annotations for this. We did not, however, do this for all combinations because this would result in an explosion of extremely rare classes. For more information on the used annotation policy, see appendix B.

Chapter 6

Conclusion and future work

Overall segmentation and classification by means of a Max-Tree data structure seems promising. Unfortunately, in its native form the Max-Tree has a somewhat limited domain of application, namely the situation in which there exists a background and intensity information that contrasts with this background. This domain includes: most cases of print recognition, astronomical data, meteorological data and all sorts of medical scans. Figure 6.1 shows three examples of problems that would be hard to solve using the methods demonstrated in this thesis. Although none of these problems are truly relevant in our data set, they do show some theoretical limits of our approach.

Images such as the one in figure 6.1**(a)** show how easily this paradigm of contrasting information can be broken. We can use a Max-Tree and the top '0' will be represented as a node, the bottom '1' will be part of the same node as the background around the '0' because it is of a lower value than the surrounding pixels. We could invert the image but that would only reverse the problem. Situations such as these are extremely rare in historic documents but quite common in modern print work. They are also quite common in natural scene images such as the ones used in [3]. In cases such as these, the use of constraint connectivity representations such as α-partition trees and α-ω-trees described in [45] can provide a better solution.

Figure 6.1**(b)** shows a problem that does occur quite often in historical documents. The ligature shown here clearly consists of three letters that any person with some knowledge of Latin scripts could recognize. In practice, one could easily solve this problem by annotating the ligatures as a single character. This is defensible because such a ligature would need to be a single piece of metal in the setting of the text. However, if for some reason we would want to split these apart we would find that no connectivity class exists that can achieve this. This is because the letters overlap and some pixels belong to multiple letters. Connectivity classes form partitions, in which every pixel is only a member of a single component. Clearly some sort of hyper-connectivity class is required that could provide each of the 3 letters with their own hyper-connected component.

The situation shown in figure 6.1**(c)** is even more problematic. Even when using mask-based connectivity, there exists no mask that would connect the body of the 'i' to its dot without upsetting the 'g'. Luckily, the nature of historical print work ensures this situation will never occur in practice, but in other applications such as handwriting recognition this type of situation might require a solution. In theory, edge-based connectivity could connect both parts by means of long distance edges. The edge function used would, however, need to be very complex to prevent unwanted merges.

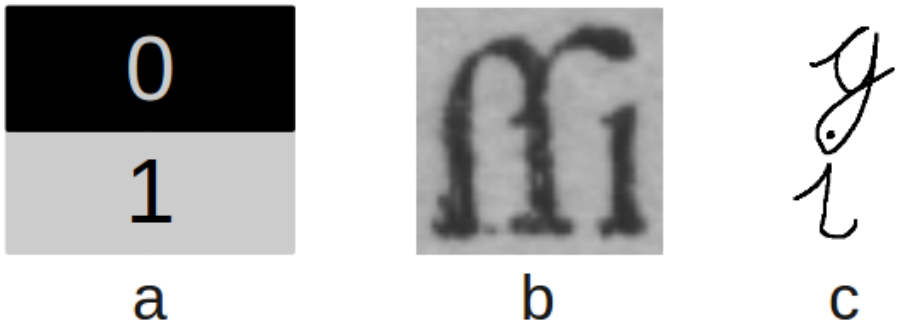

Figure 6.1: This figure shows three situations in which any Max-Tree based system would have trouble. Part **(a)** shows a complicated contrast situation in which only one of the numbers can be represented as a node, the other will fall into the background. Part **(b)** shows a ligature comprised of three letters, two times 's' and a 'i'. A human reader would recognize them as such but a computer has trouble pulling them apart. Part **(c)** shows a situation in which the dot of an 'i' is surrounded by the descender of the letter 'g'. Although this is a situation that does not occur in print-work, a Max-Tree has great problems in connecting the two parts of the 'i'

6.1 System improvements

6.1.1 Advanced map builders

Many of the problems highlighted in section 5.7.2 could be addressed by using more advanced mask and edge map building algorithms. This section will list several ways of doing this.

First of all we could build an edge-map based on a better line separator. On a first glance a lot of our difficulties with splitting the lines could be counteracted by some sort of image rotation alignment algorithm. Lines however are never exactly straight, although the *Rerum* is somewhat accommodating in this aspect. A line separator would need to be somewhat adaptive. The complete method used in [54], of which our algorithm is a simplified version, might provide some starting point but it is in essence a histogram based method that is doomed to fail on more distorted print work. For alternatives such as the drop fall method or the extended drop fall method, see [33]. Research by [20] offers a whole range of alternatives but is somewhat less verbose in how they should be implemented.

Alternatively, a lot can be gained by improving the mask generator used. Initially we considered using the tensor based smoothing system discussed in [47], but it was dropped because it was too complicated to combine with our other endeavors. Many possible alternatives exist, such as dilations based on run length or other morphological filters. For any alternative it is important that the mask should be an increase upon the original image, e.g $f(x) \leq m(x)$. We want to delegate any contractions towards the edge based connectivity because the mask-based connectivity dissolves any structures outside the mask to singleton elements making us lose any information contained in them.

6.1.2 Classifiers

The system currently uses a very simple nearest neighbor classifier based on the second, third and fourth order moments. Adding more moments in many cases actually decreases our recall rates. This either means that these moments contain more noise than information or that our classifier simply cannot effectively use the information contained in these moments. The latter seems more likely, as some of our better moments in the context of the single moment classifiers used in section 5.3 were moments of the fifth and sixth order. Replacing our classifier by a more advanced algorithm such as a support vector machine [1, Chapter 10] or a form of LVQ [18] should improve performance.

6.1.3 Text generation and language models

Our annotation currently does not include any information on the sequence in which the characters should be read. It contains only spatial information on the pixels of which the character annotations are comprised. When provided with a small amount of external knowledge, such as average line width, these annotations could be converted into a searchable text document. It would also be possible to improve recognition rates by doing some sort of post processing during this step using a language model. Preferably, however, a language model is used in conjunction with the classifier, where a classifier provides prior probabilities for all classes and the language model combines these with context information to select the correct alternatives. This would require a small modification of our current classifier and annotation system.

6.2 Theoretical extensions

6.2.1 More and longer edges

The edge-based connectivity used in our experiments with the triple-input Max-Tree expressed a regular 4-connectivity within the mask. There is no theoretical reason why we should be limited to 4 or even 8 edges per element. The practical limitation with our current implementation is that this would require vast amounts of memory in which to store the edge map.

Replacing the edge map with an edge function that computes edge weights on the fly would allow us to eliminate this memory requirement. Although removing the edge map is hardly any trouble, no longer counting the edges as elements would result in the need for some adaptation of our Max-Tree algorithm and probably increase the computational cost of building a Max-Tree.

It is also possible to include longer edges. Instead of only considering edges that connect within a limited distance edges could connect pixels several rows and columns apart. This will eliminate the need for our dilated mask as merges of components can be made through these longer edges instead. One could even attempt to solve the problem shown in figure 6.1**c**. An edge could in theory connect the body and the dot of the 'i' by 'jumping over' the descender of the 'g' although this would mean using a very complicated edge function to prevent unwanted merges.

One of the greater problems of these longer edges is that edges need to be symmetric to form a connectivity class (see definition 4); if every pixel is connected by an edge to a pixel 10 rows above it, they are also connected to a pixel 10 rows below it. This will allow even more merges between lines than those already covered by our current edge function since they could happen over longer distances. In the mask based dilation used in our experiment these merges are less of an issue since it only increases merges upwards and not in both directions.

6.2.2 Attribute merges

In theory the algroritm for merging Max-Tree trees described in [61] can also be used to merge different nodes in the same Max-Tree. This would provide us with an alternative way of merging multi component characters that does not depend on using edges and masks. Such merges could be based on the attributes of nodes. Nodes with certain attributes such as shape, location or size could be merged with other nearby nodes yielding new Max-Tree nodes with new attributes. These atttribute merges would take place after building a Max-Tree. After each merge attributes would have to be recomputed for the resulting merged nodes, a process that could lead to even more merges. Such a cascade is potentially costly but this could be mitigated by choosing attributes that can be computed efficiently on the basis of the merged nodes, such as NCMs.

6.3 Research questions

Summarizing the previous sections we can formulate answers to our three research questions:

1. **Are normalized central moments reliable features for a character classifier in a historical document processing application?** In general NCMs seem to be good features for classifying letters. We noted some issues with their scale invariance on small scales, however we did not depend on this particular property in our research.

2. **Can hyper-connected filtering techniques help in a historical document processing application?** Hyper-connected filters and the k-absorption filter in particular gave very good results in testing. Although their effectiveness depends on the underlying connectivity class their filtering was so good that we found it safe to assume that everything that was not filtered by the k-absorption filter was a character node.

3. **How can different connectivity classes be used to improve segmentation in a historical document processing application?** We have demonstrated methods for implementing various connectivity classes. We have also shown a comparison of results from three different classes in a historical document processing system. Although in this area much work remains to be done we can certainly say that we made progress on the problem of characters with multiple components.

6.4 Conclusion

Currently our system is not competitive in recognition rates with simple correlation based systems. Considering the fact that our classifier reaches a recall rate of 97 percent (section 5.4) and taking into account the great success of the k-absorption filter reaching over 99 percent recall (section 5.7.2) one might expect the classed recall rates to be similarly high.

However, as seen in the classed recall rate column in table 5.2, the combination of letter zones segmented by our alternative connectivities and the classifier fails to show a marked improvement over our vannila Max-Tree. As we remarked in the introduction a segementation should be measured by the succes of its goal and in that measure our system still has a long way to go. In the end, the connectivity classes used distorted the letter zones too much for the system to reach its full potential. Especially, a better line splitting algorithm and more attention for horizontal merges and splits would go a long way in improving the recognition rates.

In this chapter we have offered several angles from which such improvements can be made to the system. We have developed a framework in which these improvements would be easy to incorporate and test. We have also succeeded in attacking one of the more persistent problems noted in [54], namely that of vertically separated components. Further research will have to show if a Max-Tree based system such as the one descibed in this thesis can truly provide an accurate and reliable digitization of historical documents.

Appendix A

Appendix: The Max-Tree algorithm

This appendix contains the listing of the Max-Tree algorithm used in this thesis. The *build_max_tree* function is called to build a Max-Tree. Changing the *neighbor_function* argument modifies the type of connectivity used. If a modified connectivity is used (such as in the dual-input or triple-input Max-Tree) *elements_to_values* is assumed to also include the virtual elements at this point.

```
public class MaxTreeBuilder
{

  ElementValueComparator                          d_element_comparator;
  NeighborFunction                                d_neigbor_function;
  int []                                          d_elements_to_values;
  boolean []                                      d_elements_to_processed;

  Stack<Integer>                                  d_stack;
  PriorityQueue<Integer>                          d_queue;

  Stack<Pair<Integer, List<MaxTreeNode>>>         d_component_stack;
  MaxTreeNode                                     d_bottom_level_node;



  protected MaxTree build_max_tree(
          NeighborFunction neigbor_function,
          int [] elements_to_values,
          int image_width,
          int image_height)
  {

    d_neigbor_function      = neigbor_function;
    d_elements_to_values    = elements_to_values;
    d_element_comparator    = new ElementValueComparator(elements_to_values);
    d_elements_to_processed = new boolean[d_elements_to_values.length];
    d_bottom_level_node     = null;

    d_queue           = new PriorityQueue<Integer>(1, d_element_comparator);
    d_stack           = new Stack<Integer>();
    d_component_stack = new Stack<Pair<Integer, List<MaxTreeNode>>>();

    // build tree
    d_queue.add(0);
    d_elements_to_processed[0] = true;
    while (!d_queue.isEmpty() || !d_stack.isEmpty())
      process();

    return new MaxTree(image_width, image_height,
                       d_elements_to_values.length, d_bottom_level_node);
  }
```

```
private void process()
{
  // System.out.println("process!");  //debug
  if(d_queue.isEmpty())
  {
    build_component();
    return;
  }

  if(d_stack.isEmpty())
  {
    queue_to_stack();
    check_processing_level();
    return;
  }

  switch(d_element_comparator.compare(d_stack.peek(), d_queue.peek()))
  {
    case  1:
      // queue value is bigger
      queue_to_stack();
      check_processing_level();
      return;
    case  0:
      // value equal
      queue_to_stack();
      return;
    case -1:
      // stack value is bigger
      build_component();
      return;
  }
}
```

```
private void queue_to_stack()
{
  /* get current_element */
  int current_element = d_queue.poll();

  /* add neighbors */
  int [] neighbors = d_neigbor_function.get_neighbors(current_element);
  for (int neighbor : neighbors)                // for all returned neighbors
  {
    if (neighbor != -1)                         // if neighbor exists
    {
      if (!d_elements_to_processed[neighbor])   // and if it is unprocessed
      {
        d_elements_to_processed[neighbor] = true;
        d_queue.add(neighbor);
        // push current_element back into the queue
        if (d_elements_to_values[current_element] <
            d_elements_to_values [neighbor])
        {
          d_queue.add(current_element);
          return;
        }
      }
    }
  }
  /* transfer current_element to stack*/
  d_stack.push(current_element);

}
```

```
/* function for building the max tree nodes*/
private void build_component()
{
  int value = d_elements_to_values[d_stack.peek()];

  /* gather elements*/
  List<Integer> element_list = new Vector<Integer>();
  do
    element_list.add(d_stack.pop());
  while (!d_stack.isEmpty() && d_elements_to_values[d_stack.peek()] == value);

  /* gather child nodes*/
  Pair<Integer, List<MaxTreeNode>> level  = d_component_stack.pop();
  MaxTreeNode new_node = new MaxTreeNode(
          value,
          CollectionTools.convert_to_integer_array(element_list),
          CollectionTools.convert_to_max_tree_node_array(level.get_object2()));

  /* either add it to the component stack or make it the bottom level node*/
  if(check_processing_level())
    d_bottom_level_node = new_node;
  else
    d_component_stack.peek().get_object2().add(new_node);
}
```

```
/* function for managing the component stack                    */
/* returns true if the both the stack and the queue are empty*/
private boolean check_processing_level()
{
  int queue_level = -1;
  int stack_level = -1;
  if (d_queue.isEmpty())
  {
    if (d_stack.isEmpty())
      return true;
    else
      stack_level = d_elements_to_values[d_stack.peek()];
  }
  else
  {
    queue_level = d_elements_to_values[d_queue.peek()];
    if (!d_stack.isEmpty())
        stack_level = d_elements_to_values[d_stack.peek()];
  }

  int processing_level_target = Math.max(queue_level, stack_level);
  if (d_component_stack.isEmpty())
    d_component_stack.push(new Pair<Integer, List<MaxTreeNode>>(
            processing_level_target,
            new Vector<MaxTreeNode>()));
  else
    if (d_component_stack.peek().get_object1() < processing_level_target)
        d_component_stack.push(new Pair<Integer, List<MaxTreeNode>>(
                processing_level_target,
                new Vector<MaxTreeNode>()));
  return false;
}
```

```
  /* private comparator class for sorting the queue */
  private class ElementValueComparator implements Comparator<Integer>
  {
    int [] d_elements_to_values;

    public ElementValueComparator(int [] elements_to_values)
    {
      d_elements_to_values = elements_to_values;
    }


    @Override
    public int compare(Integer element_index1, Integer element_index2)
    {
      int value1 = d_elements_to_values[element_index1];
      int value2 = d_elements_to_values[element_index2];

      if (value1 == value2)
        return 0;

      if (value1 < value2)
        return 1;  // comparator is in reversed
      else
        return -1;
    }
  }
}
```

Appendix B

Appendix: Annotation

This appendix is included for anyone who would like to use the data set contructed in this thesis for further research, it details policies followed in annotation and observation about the data in the *Rerum*. Note that the annotation is not an expert annotation. It will undoubtedly contain mistakes and inconsistensies. It is also quite small. As a result of the considerable effort required to make an acurate pixel level annotation, we only annotated around 0.5% of the entire *Rerum*.

B.1 About the images

The images used are sections of scans used in previous research. In order to keep the data set managable, we cut selected pages into four sections, randomly discarded two and added from the remaining one section to training-set and the other to the test-set. Pages were randomly selected from different portions of the book that were neither title pages (pages containing only a few very large font characters) or table pages (pages containing large layout structures such as tables or graphs). We also included margin notations and page numberings in our images wherever they appeared. These were also annotated.

Some of the pages are scanned in a rotated position (-5 to 5 degrees of rotation). Although it heavily influences our results, we made no effort to counteract this rotation because it was outside the scope of this research. This rotation causes some lines to be only partially present in our images; in these cases they were annotated when a recognizable letter was observable and left unannotated when only parts of the characters were visible.

B.2 Annotation policy

In annotation we strove to be consistent and detailed. In retrospect, however, we made wrong decisions. Initially, the annotation used unicode strings to represent the labellings of the different letter zones. At some point, however, we encountered characters that were not present in the unicode set (such as a q with an acent grave), and we therefore took to the convention used in the monk project [53], where any letter outside the ASCII set is annotated by a '@' followed by an indentifing string of characters. This convention also allows for simpler to read file formats. The annotation we made was semantic: we annotated what letters the characters meant, not what they

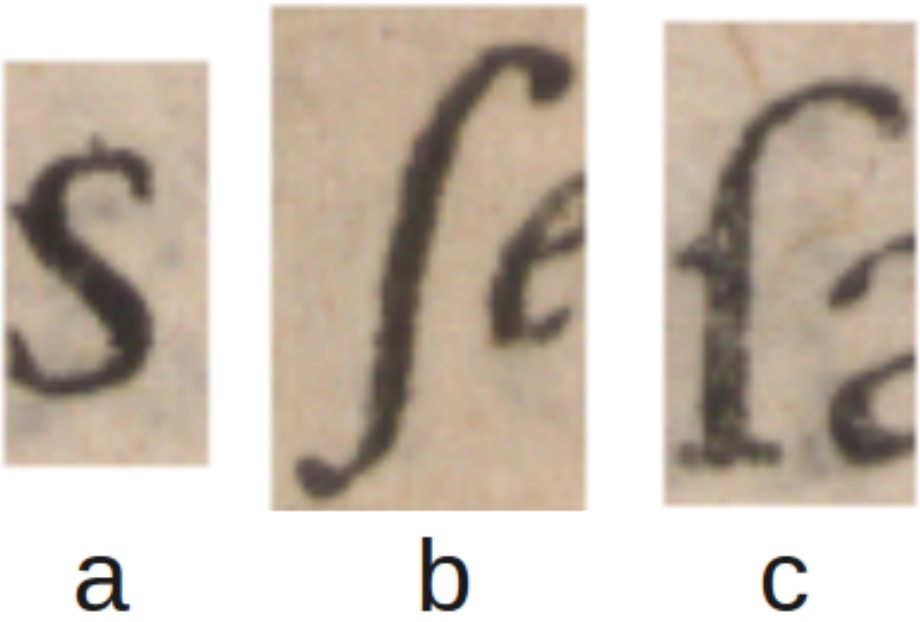

Figure B.1: This image shows three different shapes the letter 's' can take in the Rerum. **(a)**, the regular short 's'. **(b)**, the long 's' or descending 's', used at the beginning or in the middle of a word (the Rerum is not completely consistent in this), **(c)** the italic version of the long 's'. All of these were annotated as the same letter.

looked like. This means that there are sometimes multiple ortographic representations of the same letter labeled in the same way. The most striking example of this is the 's' shown in figure B.1.

The text sometimes contains complex ligatures, or otherwise consistent combinations of letters. These were annotated as the combinations of letters they are supposed to indicate. It is sometimes difficult for a non-expert viewer to label these ligatures and some of them have special meanings. These were, however, seen as irrelevant to this thesis. It is also sometimes the case that characters invade each others' space, most notably the s and the f. We deemed it unlikely, but not impossible, that these were all single form ligatures, because there were simply too many different combinations. We therefore annotated these cases as separate letters. The most notable execptions to this are the 'si' and the 'fi' combinations. In these cases the first letter absorbes the dot of the second letter, making it evident that in these cases a ligature, such as shown in figure B.2, was used. This also means that the intrustion makes the other letter more difficult to recognise. We therefore annotated these combinations as a single 'si' combination.

About a tenth of the *Rerum* is written in a separate italic font. Although some of the characters

Figure B.2: This image shows a ligature that could be used for the 'si' combination. The 'si' combination is by far the most common in the Rerum. In our relatively small sample we found over 20 different combinations of the s, the f and other letters that invaded each other's letterspace but did not connect. Typesets such as the one used to print the Rerum were quite expensive and we found it hard to assume that all these combinations were available as ligatures. We therefore annotated them as separate characters. *Image released under the GNU Free Documentation License.*

```
File := {Header|Data}
  Header := {source_image_file_name "\n" | integer source_image_width
                                         | integer source_image_height | "\n"}
  Data   := {*DataEntry}
    DataEntry   := {AreaDescription|Labeling}
      AreaDescription := {"box"    | double top_left_x | double top_left_y
                                   | double box_width  | double box_height
                                   | double confidence_score | "\n"}
      AreaDescription := {"element"| {*integer image_element} | "\n"}
      Labeling        := {String | "\n"}
```

differ significantly from their regular font counterparts, they were annotated as the same letters. Uppercase letters are annotated as uppercase letters. Sometimes they only differ from their lowercase counterparts in size. As a result of the fact that our features are scale invariant they are often confused with one another when the differences are small. In theory, this could be resolved by adding some sort of a scale invariant area feature, perhaps normalized by the line height. Since capital letters differ very little in shape but are often bold and as such cover relatively more area than their lower case counterparts, such a feature would be a good way of telling them apart.

B.3 File format

Internally annotations were saved as serialized java objects. However for better cooperation with other researchers a file format was defined. The grammar of this file format (extension aff) is listed in figure B.3 :

Bibliography

[1] E. Alpaydin. *Introduction to machine learning*. The MIT Press, 2004.

[2] A. Belaid and J.P. Haton. A syntactic approach for handwritten mathematical formula recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (1):105–111, 1984.

[3] F. Bracci. Automatic traffic sign recognition. Master's thesis, Rijksuniversiteit Groningen, 2010.

[4] U. Braga-Neto and J. Goutsias. A theoretical tour of connectivity in image processing and analysis. *Journal of Mathematical Imaging and Vision*, 19(1):5–31, 2003.

[5] U. Braga-Neto and J. Goutsias. Object-based image analysis using multiscale connectivity. *IEEE transactions on pattern analysis and machine intelligence*, pages 892–907, 2005.

[6] H. Bunke, S. Bengio, and A. Vinciarelli. Offline recognition of unconstrained handwritten texts using HMMs and statistical language models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(6):709–720, 2004.

[7] R.G. Casey and E. Lecolinet. A survey of methods and strategies in character segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(7):690–706, 1996.

[8] T. L. Dimond. Devices for reading handwritten characters. In *Papers and discussions presented at the December 9-13, 1957, eastern joint computer conference: Computers with deadlines to meet*, pages 232–237, 1957.

[9] J. Flusser and T. Suk. Affine moment invariants: a new tool for character recognition. *Pattern Recognition Letters*, 15(4):433–436, 1994.

[10] H. Freiberger and E. F. Murphy. Reading machines for the blind. *Human Factors in Electronics, IRE Transactions on*, (1):8–19, 1961.

[11] M. H. Glauberman. Character recognition for business machines. *Electronics*, 29(2):132–136, 1956.

[12] E. C. Greanias. Some important factors in the practical utilization of optical character readers. *Optical Character Recognition*, pages 129–146, 1962.

[13] P.W. Handel. HANDEL, 1933. US Patent 1,915,993.

[14] S. Hockey. OCR: the Kurzweil data entry machine. *Literary and Linguistic Computing*, 1(2):63–67, 1986.

[15] M.K. Hu. Visual pattern recognition by moment invariants. *Information Theory, IRE Transactions on*, 8(2):179–187, 1962.

[16] R. Jones. Connected filtering and segmentation using component trees. *Computer Vision and Image Understanding*, 75(3):215–228, 1999.

[17] Fred. N. Kiwanuka and Michael H.F. Wilkinson. Automatic attribute threshold selection for blood vessel enhancement. *Pattern Recognition, International Conference on*, pages 2314–2317, 2010.

[18] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[19] T.Y. Kong and A. Rosenfeld. Digital topology: introduction and survey. *Computer Vision, Graphics, and Image Processing*, 48(3):357–393, 1989.

[20] L. Likforman-Sulem, A. Zahour, and B. Taconet. Text line segmentation of historical documents: a survey. *International journal on document analysis and recognition*, 9(2):123–138, 2007.

[21] P. Maragos and R.D. Ziff. Threshold superposition in morphological image analysis systems. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12(5):498–504, 1990.

[22] S. Mori, C.Y. Suen, and K. Yamamoto. Historical review of OCR research and development. *Proceedings of the IEEE*, 80(7):1029–1058, 1992.

[23] L. Najman and M. Couprie. Building the component tree in quasi-linear time. *Image Processing, IEEE Transactions on*, 15(11):3531–3539, 2006.

[24] L. O'Gorman. Image and document processing techniques for the RightPages electronic library system. In *Pattern Recognition, 1992. Vol. II. Conference B: Pattern Recognition Methodology and Systems, Proceedings., 11th IAPR International Conference on*, pages 260–263. IEEE, 1992.

[25] N. Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11:285–296, 1975.

[26] G. K. Ouzounis and M. H. F. Wilkinson. Second-order connected attribute filters using max-trees. *Mathematical Morphology: 40 Years On*, pages 65–74, 2005.

[27] G. K. Ouzounis and M. H. F. Wilkinson. Mask-based second generation connectivity and attribute filters. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(6):990–1004, 2007.

[28] G. K. Ouzounis and M. H. F. Wilkinson. Hyperconnected attribute filters based on k-flat zones. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(2):224–239, 2011.

[29] S.C. Park, M.K. Park, and M.G. Kang. Super-resolution image reconstruction: a technical overview. *Signal Processing Magazine, IEEE*, 20(3):21–36, 2003.

[30] F. Peng and D. Schuurmans. Combining naive Bayes and n-gram language models for text classification. *Advances in Information Retrieval*, pages 547–547, 2003.

[31] F. Peng, D. Schuurmans, and S. Wang. Augmenting naive Bayes classifiers with statistical language models. *Information Retrieval*, 7(3):317–345, 2004.

[32] I. Pratikakis, B. Gatos, and K. Ntirogiannis. H-DIBCO 2010-Handwritten Document Image Binarization Competition. In *2010 12th International Conference on Frontiers in Handwriting Recognition*, pages 727–732. IEEE, 2010.

[33] J. Punnoose. *An improved segmentation module for identification of handwritten numerals*. PhD thesis, Massachusetts Institute of Technology, 1999.

[34] A. Ratnaparkhi. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34(1):151–175, 1999.

[35] J.A. Rodrıguez and F. Perronnin. Local gradient histogram features for word spotting in unconstrained handwritten documents. In *Int. Conf. on Frontiers in Handwriting Recognition*, 2008.

[36] A. Rosenfeld. Connectivity in digital pictures. *Journal of the ACM (JACM)*, 17(1):146–160, 1970.

[37] R. Rosenfeld. Two decades of statistical language modeling: Where do we go from here? *Proceedings of the IEEE*, 88(8):1270–1278, 2000.

[38] R. Moelker S. Bouma, W. Buck. Vector-attribute character recognition. Master's thesis, Rijksuniversiteit Groningen, 2011.

[39] P. Salembier, A. Oliveras, and L. Garrido. Antiextensive connected operators for image and sequence processing. *Image Processing, IEEE Transactions on*, 7(4):555–570, 1998.

[40] M. Schenkel, H. Weissman, I. Guyon, C. Nohl, and D. Henderson. Recognition-based segmentation of on-line hand-printed words. *Advances in Neural Information Processing Systems*, pages 723–723, 1993.

[41] J. Serra. Image analysis and mathematical morphology, vols. 1, 2. *Academic, London (1982, 1988)*.

[42] J. Serra. Connectivity on complete lattices. *Journal of Mathematical Imaging and Vision*, 9:231–251, 1998.

[43] M. Sezgin and B. Sankur. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic imaging*, 13:146, 2004.

[44] D.H. Shepard. Apparatus for reading, 1953. US Patent 2,663,758.

[45] P. Soille and J. Grazzini. Advances in constrained connectivity. In *Discrete Geometry for Computer Imagery*, pages 423–433. Springer, 2008.

[46] S.N. Srihari. Recognition of handwritten and machine-printed text for postal address interpretation. *Pattern recognition letters*, 14(4):291–302, 1993.

[47] C.K. Tang and G. Medioni. Curvature-augmented tensor voting for shape inference from noisy 3d data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 858–864, 2002.

[48] Y.Y. Tang, S.W. Lee, and C.Y. Suen. Automatic document processing: a survey. *Pattern Recognition*, 29(12):1931–1952, 1996.

[49] C.C. Tappert, C.Y. Suen, and T. Wakahara. Online handwriting recognition-a survey. In *Pattern Recognition, 1988, 9th International Conference on*, pages 1123–1132. IEEE, 1988.

[50] G. Tauschek. BEADING MACHINE, 1935. US Patent 2,026,329.

[51] A. Tonazzini, E. Salerno, and L. Bedini. Fast correction of bleed-through distortion in grayscale documents by a blind source separation technique. *International Journal on Document Analysis and Recognition*, 10(1):17–25, 2007.

[52] Ø.D. Trier and A.K. Jain. Goal-directed evaluation of binarization methods. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(12):1191–1201, 1995.

[53] T. Van der Zant, L. Schomaker, and K. Haak. Handwritten-word spotting using biologically inspired features. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(11):1945–1957, 2008.

[54] T. M. van Laarhoven. Text recognition in printed historical documents. Master's thesis, Rijksuniversiteit Groningen, 2010.

[55] P. Viviani and C. Terzuolo. Trajectory determines movement dynamics. *Neuroscience*, 7(2):431–437, 1982.

[56] L. Von Ahn, M. Blum, N. Hopper, and J. Langford. CAPTCHA: Using hard AI problems for security. *Advances in CryptologyEUROCRYPT 2003*, pages 646–646, 2003.

[57] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465, 2008.

[58] J. Wang and J. Jean. Segmentation of merged characters by neural networks and shortest path. *Pattern Recognition*, 27(5):649–658, 1994.

[59] M.H.F. Wilkinson. Attribute-space connectivity and connected filters. *Image and Vision Computing*, 25(4):426–435, 2007.

[60] M.H.F. Wilkinson. Hyperconnections and openings on complete lattices. *Mathematical Morphology and Its Applications to Image and Signal Processing*, pages 73–84, 2011.

[61] M.H.F. Wilkinson, H. Gao, W.H. Hesselink, J.E. Jonker, and A. Meijster. Concurrent computation of attribute filters on shared memory parallel machines. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(10):1800–1813, 2008.

[62] Q. Yuan and C. L. Tan. Text extraction from gray scale document images using edge information. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, pages 302–306. IEEE, 2001.