Text Segmentation for Document Recognition

8

Nicola Nobile and Ching Y. Suen

Contents

N. Nobile
Centre for Pattern Recognition and Machine Intelligence (CENPARMI), Concordia University, Montréal, QC, Canada
e-mail: nicola@cenparmi.concordia.ca

C.Y. Suen
Department of Computer Science and Software Engineering, Centre for Pattern Recognition and Machine Intelligence (CENPARMI), Concordia University, Montréal, QC, Canada
e-mail: suen@encs.concordia.ca; suen@cenparmi.concordia.ca

D. Doermann, K. Tombre (eds.), *Handbook of Document Image Processing and Recognition*, DOI 10.1007/978-0-85729-859-1_8,

Abstract

Document segmentation is the process of dividing a document (handwritten or printed) into its base components (lines, words, characters). Once the zones (text and non-text) have been identified, the segmentation of the text elements can begin. Several challenges exist which need to be worked out in order to segment the elements correctly. For line segmentation, touching, broken, or overlapping text lines frequently occur. Handwritten documents have the additional challenge of curvilinear lines. Once a line has been segmented, it is processed to further segment it into characters. Similar problems of touching and broken elements exist for characters.

An added level of complexity exists since documents have a degree of noise which can come from scanning, photocopying, or from physical damage. Historical documents have some amount of degradation to them. In addition, variation of typefaces, for printed text, and styles for handwritten text bring new difficulties for segmentation and recognition algorithms.

This chapter contains descriptions of some methodologies, presented from recent research, that propose solutions that overcome these obstacles. Line segmentation solutions include horizontal projection, region growth techniques, probability density, and the level set method as possible, albeit partial, solutions. A method of angle stepping to detect angles for slanted lines is presented. Locating the boundaries of characters in historical, degraded ancient documents employs multi-level classifiers, and a level set active contour scheme as a possible solution. Mathematical expressions are generally more complex since the layout does not follow standard and typical text blocks. Lines can be composed of split sections (numerator and denominator), can have symbols spanning and overlapping other elements, and contain a higher concentration of superscript and subscript characters than regular text lines. Template matching is described as a partial solution to segment these characters.

The methods described here apply to both printed and handwritten. They have been tested on Latin-based scripts as well as Arabic, Dari, Farsi, Pashto, and Urdu.

Keywords

Anisotropic probability density • Broken • Characters • Contour • Curvilinear • Degradation • Features • Handwriting • Histogram • Historical • Level set • Lines • Line slant • Normalization • OCR • Pixel density • Projection • Recognition • Region growth • Segmentation • Template matching • Touching • Typefaces

Introduction

Document segmentation has been a difficult problem to solve through automation. The problem itself has evolved since its inception. Where initially, the problem was to recognize isolated printed words and characters, it has modernized itself over the

years in order to keep up with user needs. In later years, the research in this field became more complex. The progression continued by finding words and characters [29] in a line of text, then finding all text lines in a given printed document. As documents became more complex, it was then necessary to identify the regions of a document which are text and which are non-text – a common layout for newspapers and magazines.

Modern demands push the field into finding solutions for new problems. Such is the case with the Sarbanes-Oxley Act. This law requires every public company in the United States to electronically store all business records. Storing these document images is the easy part – searching them is much more difficult. These documents can have any layout ranging from standard business letters, receipts, purchase forms, and accounting tables to name a few, to charts, tables, and figures containing important textual information which need to be identified.

To handle this wide range of layouts, document processing systems [25] are required to identify the location of the text blocks and separate them from the non-text blocks. Of these text blocks, the lines of text need to be separated. The lines may be of different orientations from each other or a line may even be curved.

More recent applications include handwritten [15] documents usually co-existing with printed text in a document. Handwritten line segmentation and OCR are much more difficult [19–22, 28, 35] due to the greater diversity of handwriting styles compared with printed texts. Furthermore, machine-printed text lines are usually laid out straight and the skew and/or rotated texts are minimal, usually caused when it is – scanned or photocopied where the document is not aligned with the glass edge of the recording device. Handwritten documents are influenced mostly by proper line segmentation. The latest trend is geared towards page segmentation and OCR of Middle-Eastern languages such as Arabic [11, 30, 32], Farsi [12], and Urdu [23].

Other recent applications which use text segmentation and OCR technology are search engines used to search scanned documents. This is most useful for historical documents which would take too long to manually create an editable text version.

Word spotting is a relatively new field. The goal is to find specific words in documents. Depending on the system, accuracy relies greatly on the OCR [24] performance. Data mining can be used to find similar documents based on a user's reading patterns. Each has its own unique challenges and obstacles in OCR.

The remaining of this chapter discusses the latest research in line segmentation (section "Zone and Line Segmentation") and OCR will be discussed in section "Segmentation for Optical Character Recognition."

Zone and Line Segmentation

Document segmentation involves locating and separating the text from the non-text regions of a page. In ▸Chap. 5 (Page Segmentation Techniques in Document Analysis), the reader can find more information on document segmentation. In ▸Chap. 6 (Analysis of the Logical Layout of Documents), the reader can find details on how text blocks are located in complex document layouts.

Once a text region has been identified, it is passed to a line segmentation module which will locate and, if needed, separate the lines in that region. The scope of this chapter involves the processing of a region of text which contains no images and contains only text. The text may be arranged in an unnatural manner or it may contain noise. It is the job of the line segmentation module to locate the lines which have been naturally written and send this information to next module for further processing. Breaking down the problem from large objects (text and non-text regions) to smaller objects (characters or connected components) allows each module in the process to deal with fewer items, thereby speeding up the process and accuracy. However, any errors that occur in previous modules will affect subsequent tasks that follow. This has a negative effect on accuracy of all modules, be it OCR or word recognition.

Challenges

Documents come in a variety of layouts, printing media, and languages. Older documents may have been scanned at a lower resolution and may show signs of degradation. Similarly, historical documents usually are in poor condition, so high resolution scanning would not improve the quality.

Regardless of the layout, media, or language, all digitized documents contain a common feature – noise. The only difference is the degree of noise present in a digital document. Another common challenge to overcome is the flow of the text. Ideally, all text would be aligned in a straight organized direction and well-spaced. This is usually not the case since it is observed that most scanned documents are rotated and contain some skew. Furthermore, consecutive text lines can be overlapping or worse, touch each other. Touching lines should be separated before further processing can be applied. Other common hurdles include broken lines, lines that do not contain baseline information, curvilinear text, and/or touching/broken characters.

Algorithms were developed to address these issues and a few of them are discussed in this section. Section "Noisy Documents" discusses and describes the types of noise in documents and what can be done to clean it. In section "Historical Documents," a description of noise specific to historical documents is presented. In section "Line Segmentation," a few line segmentation techniques are explained.

Noisy Documents

Noise is the appearance of objects in a document image which do not belong to or are not part of the original document. These objects can be the result of poor paper quality, digitizing, or a physically damaged document (e.g., handwriting over text, coffee spills, creases, scratches).

Several types of noise can appear in an image but the most common type is "salt and pepper" noise which contains randomly white (salt) and black (pepper) pixels appearing throughout the image. Some causes of this kind of noise can come from dead pixels on the capture device, errors from converting analog to digital,

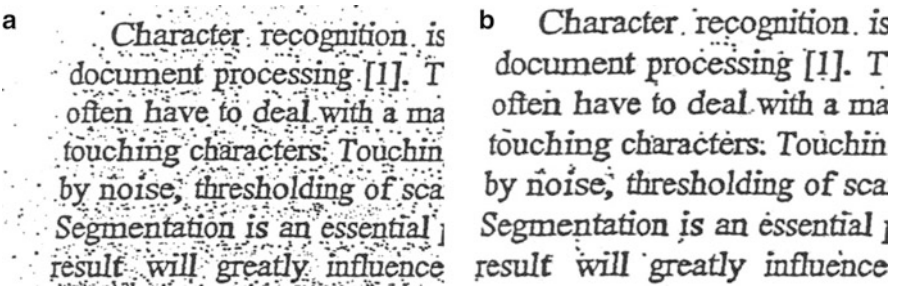

Fig. 8.1 Salt and pepper noise example (before (**a**) and after (**b**) cleaning)

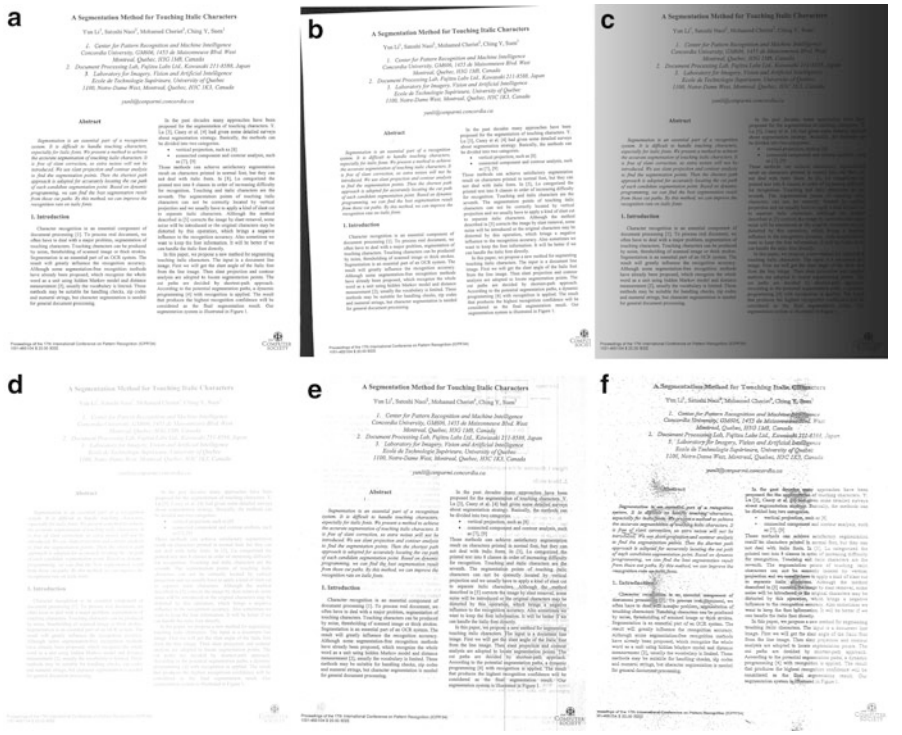

Fig. 8.2 Common photocopying artifacts; (**a**) original, (**b**) *black* borders, (**c**) uneven toner, (**d**) too Light, (**e**) too dark, (**f**) 10th generation copy

from quantization, using long shutter speeds or a high ISO in digital cameras, and transmission errors. This noise is commonly corrected by using a median filter. Figure 8.1 shows an example of an image containing salt and pepper noise before and after cleaning.

Noise can be produced by the way a document was handled. For example, poor photocopying is one source that results in noisy and degraded images. Figure 8.2 displays some common problems resulting in poor photocopying.

In Fig. 8.2b, the document image contains black borders around the perimeter. This usually occurs when the document is slanted and the cover of the scanner was left open during photocopying, or the photocopier has a dark background under the lid when it is closed. Although these borders do not usually interfere with the text of the document, the segmentation preprocessing must still identify these unwanted objects and remove them.

Another common problem that results from photocopying (or sometimes printing) is the uneven distribution of the toner which leaves part of the document lighter than the rest. Special care is needed to identify this problem and to correct it. If not identified, the lighter part of the document may disappear during the binarization process if a fixed threshold is used. This can be seen in Fig. 8.2c.

Some photocopiers have the option to make the copy lighter or darker than the original. Making the document too light may cause some characters to be too faint to recognize. Making the page darker may introduce a dark scanning "texture" as seen in Fig. 8.2e which interferes and overlaps the text giving the segmentation a difficult challenge to correct. Finally, another common routine people do is to make copies of copies. This can be repeated several times with each new generation inheriting the previous generation problems (i.e., noise) while introducing new unwanted artifacts. Furthermore, since a photocopy is not 100 % identical to the original, the quality of the document degrades with each new generation.

Figure 8.2f shows an example of a simple document which went through ten copy generations on the same photocopier. This shows the additive characteristic of nth-generation copying. Flaws in the photocopier(s) will show in the copy in addition to the previous flawed versions. Other copier functions can affect the quality such as resizing and resolution. Fax machines usually print in a lower resolution (usually between 100 and 200 DPI). Additionally, dirty copier or scanner glass or damaged capture devices will lead to noisier images.

The process of scanning itself can sometimes lead to degraded documents regardless of the quality of the document. Some examples of unwanted scanning items include scanning at low resolution, with a dirty or defective scanner, rotated or upside-down scans, and ghost images. Ghost images appear when the scanned page is double-sided or there is another page behind the scanned page during scanning. If the paper is thin enough, or the scanner lamp is too bright, ink from the second page will faintly appear on the scanned image. Also, some newer scanners can scan in duplex mode in such a way that both pages are scanned at the same time. This leads to the second page ink to "bleed" through the scan and appear in the image. Figure 8.3 shows examples of "ghost images."

Historical Documents

The current trend for digital libraries and online publishers is to make their books and magazines available online in digital form. This is not a problem with recent documents that have been written using word processors. However, old historical documents need to be scanned and sent through an OCR. The challenges with old

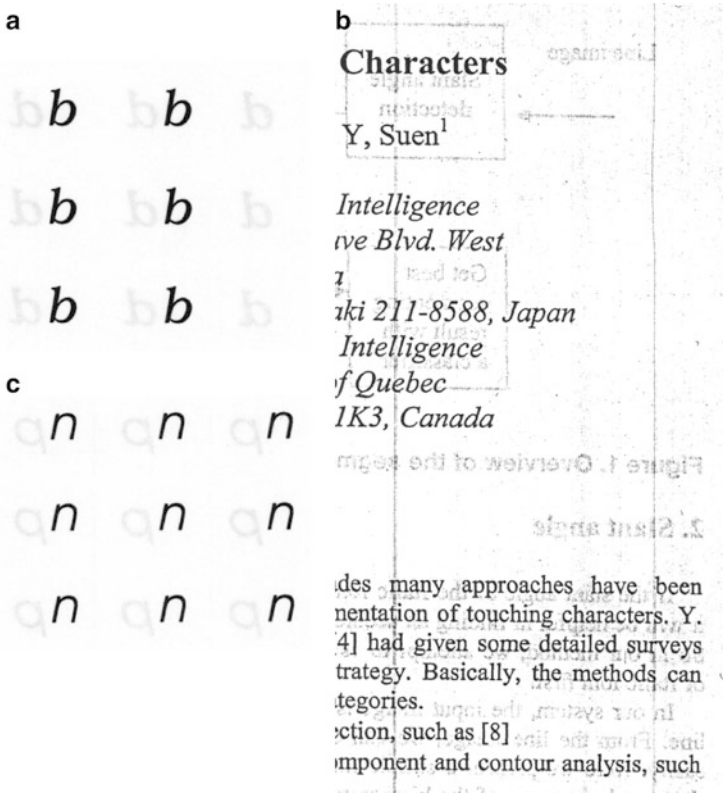

Fig. 8.3 Scanning ghost (show-through) examples

documents are usually related to the quality of the paper. Aged paper shows signs of fading, degrading, destruction from infestation, or degrading from environmental elements such as heat, oxygen, sunlight, humidity, and general paper decay. Any of these issues result in a poor image quality and therefore making it more difficult to segment characters. Figure 8.4 shows a few examples of aged-looking documents.

Line Segmentation

Given a digital text image, a line segmentation algorithm locates and extracts each text line from the image for further processing. Several types of software, such as OCR, word segmentation, and word spotting, make use of individual lines of text. The challenges for line segmentation are listed as follows:

1. Overlapping line boundaries
2. Touching lines
3. Broken lines
4. Lack of baseline information

Fig. 8.4 Examples of old historical books/documents

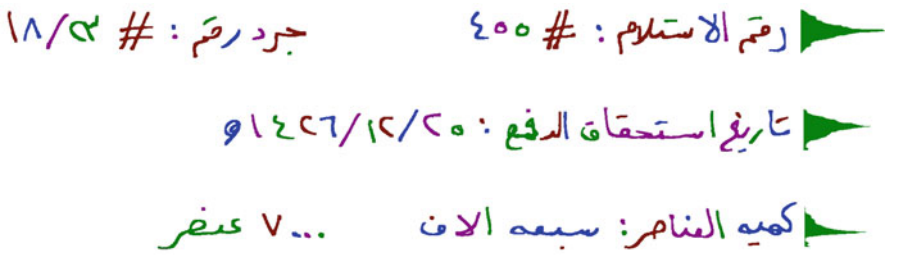

Fig. 8.5 Highlighted connected components of Arabic text (with horizontal projection)

5. Curvilinear text
6. Piecewise linear text
7. Touching characters and words within a line

Several approaches exist to solve these problems and they fall into one of two categories – top-down and bottom-up [1]. In a top-down approach, a line segmentation algorithm uses large features of a line in order to determine its boundaries.

Bottom-up starts from the smallest element of a document image – the pixel. By grouping touching pixels, connected components are generated. In Fig. 8.5, a sample Arabic handwritten text is shown with the different connected components displayed in various colors.

Researchers in image processing use connected components as the building blocks for analysis. In the case of text segmentation, they are used to locate characters and symbols having a specific relationship – they could be components on the same line or belonging to a complex mathematical formula (spanning several lines and containing symbols).

Segmenting the lines in Fig. 8.5 is a rather easy task. To the right of the three lines are the horizontal projections (in green). The gaps between the lines have no value in the projection and therefore can be used to determine where one line begins and another ends.

However, not all texts are written as neatly as the sample shown in Fig. 8.5. Most look like the example in Fig. 8.6 where handwritten text may begin in a straight line but taper off towards the end (right to left) [18]. This may be due to writers "drifting" off or it can be from a photocopy of the inner binding of a book. Figure 8.6 shows examples of overlapping lines as well as some touching lines.

Other elements that can add to the difficulties are broken lines, lack of baseline information, and unique language features. Languages from different regions make some techniques, which are successful with one family of languages, less effective on other types of languages. For example, using connected components on printed English text can help identify individual isolated characters in a line of text. However, for Arabic printed text, neighboring characters are frequently touching and not isolated. Additionally, for Arabic, the character shape/style changes depending on where in a word the character appears (beginning, middle, or end).

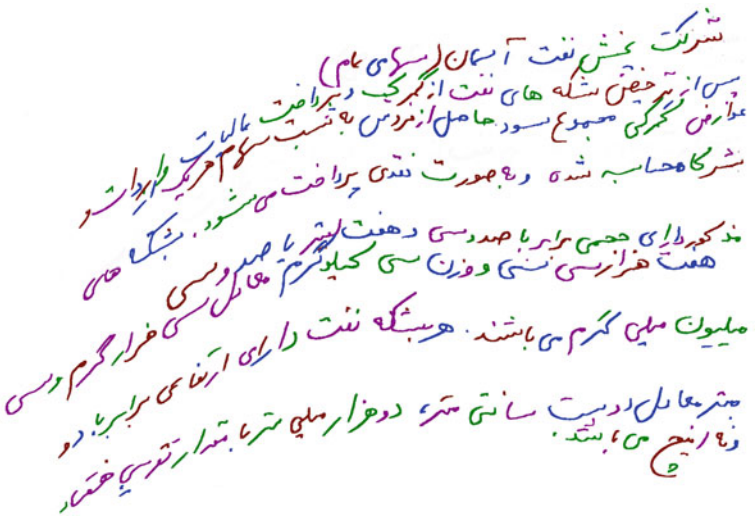

Fig. 8.6 Highlighted connected components in curvilinear, overlapping, and touching lines of handwritten Arabic text

COSUVWXZ
cosuvwxz

Fig. 8.7 English printed characters having same upper and lower case shapes

Lack of baseline information occurs when all the characters on the line have the same height and each of those characters begin and end at the same y-positions. Figure 8.7 shows the English printed characters which fall into this category.

When this situation occurs, the case of the characters cannot be determined – even though the letters are correct. This can be a problem later on when the segmented line is passed to an OCR.

Several methods exist to counter these challenges in order to correctly segment lines from a document. The following lists some techniques used:

1. Horizontal projection
2. Region growth techniques
3. Probability density
4. Level set method

Each has its own advantages and disadvantages and they are discussed in greater detail in the remaining of this chapter.

Horizontal Projection

Using a horizontal projection is probably the easiest way to segment lines [16]. A horizontal projection is a histogram, or count, of all the foreground pixels in a document for each horizontal scan line. Figure 8.5 shows horizontal projections to the right of the text. This method is ideal when the lines are well separated and

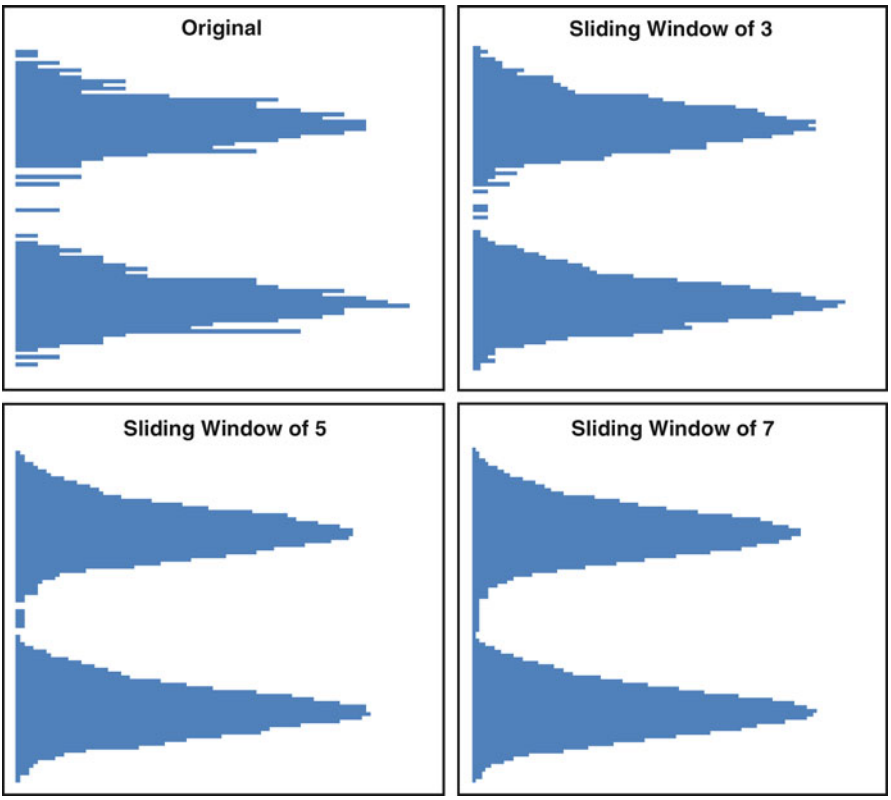

Fig. 8.8 Horizontal projection smoothing with different window sizes

are not overlapping or touching each other. In this situation, the projection will contain non-zero values for scan lines crossing a line of text. Gaps between lines will have values of zero thereby effectively marking the start and ends of the text line boundaries.

Small values may be present in the gaps if some noise is present. In this case, a threshold value or a smoothing algorithm can be applied to reduce the noise. Smoothing can also be used to fill in broken lines.

Figure 8.8 shows the horizontal projection for a short two line text block. The original contains some noise between the lines and the projection within each line is not smooth. There are too many high peak-to-valley values for adjacent scan lines. This is sometimes an indicator of broken text or some special noise, such as a cross-out [17], is present. When a sliding window of 3 pixels is applied, the peaks fall and the "holes" are filled in. A sliding window of seven eliminates all the influence of the gaps and noise within the lines. The peaks between the lines, which were caused by noise, have been reduced to small enough values which can be detected by a small threshold value.

The horizontal projection is best suited for printed text. Printed text is more consistent when it comes to maintaining gaps between the lines and for keeping the lines horizontally straight. Handwritten text can have curvilinear, touching, and overlapping text as seen in the example in Fig. 8.6.

Region Growth Techniques

Region growth techniques are methods which group neighboring pixels in an image into subregions. Pixels in a subregion share a common feature that no other pixel, in other subregions, has. A feature can be anything from the color of the pixel, the intensity, or texture, for example.

Several methods exist but all should obey the following rules:

1. Every pixel in the document must belong to a subregion.
2. Pixels in a subregion must be spatially connected with one another.
3. No two subregions have a common pixel member. That is, each pixel can belong to only one subregion.
4. Each pixel in a subregion must satisfy a logical predicate based on a feature. For example, each pixel color.
5. Applying a predicate function from one subregion to the pixels of another subregion should return false (or the empty set). That is, pixels not belonging to the first subregion cannot satisfy its predicate function.

The process of building the subregions is a bottom-up approach. It begins with "seedpoints." A seedpoint is an initial point, or pixel, in a digital document from which to begin to build a subregion from. More than 1 pixel can be added to a subregion's set of seedpoints. From a seedpoint, the neighboring pixels are examined and are added to the subregion based on the criterion established for subregion pixel membership, and they do not belong to any other subregion's set of pixels. Neighboring pixels are pixels which are immediately adjacent – either by 4-connectivity or 8-connectivity connection. The next step continues by following the same membership confirmation of neighboring pixels based around the newly added pixels from the previous iteration. This iterative process ends when no new pixels have been added to the set.

Choosing initial seedpoints is an important step and can be facilitated by using overall document statistics. For example, in our case, if text is needed to be removed from a background [27], the histogram of a grayscale image can be viewed and a value can be chosen where the foreground pixels are prevalent. The criterion for this subregion could be if the pixel grayscale value is less than a certain value. This value is shown as the red line in Fig. 8.9.

Of course this is not limited to just pixel counts. An alternative could be to assign a pixel on the edge of an object as our seedpoint and add neighboring pixels which are also edge pixels. This would give a set of points which form the outline of the object in which the initial seedpoint resides on. Or a criterion based on texture can be used. Applications using texture-based criteria are mainly used for image segmentation such as locating contaminated areas in meat products, defects in lumber moving through a sawmill, or cancerous regions in cell tissue, but can also be used for text segmentation.

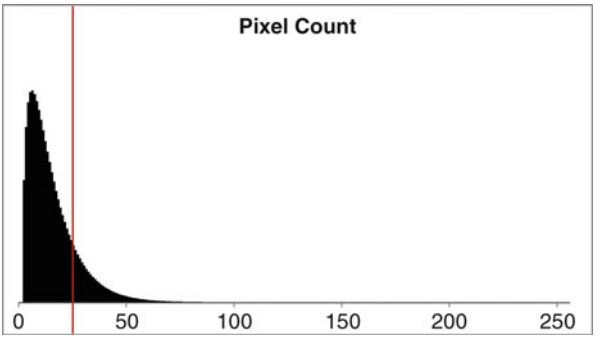

Fig. 8.9 Sample text document pixel intensity histogram

Note that choosing the initial seedpoints is important. This method will only find the pixels in the document which satisfy the criterion and are connected to the initial seedpoints. Points which meet the subregion's criterion but cannot be linked to any of the seedpoints will be neglected. In our example above, at least one point from each character in the text document will need to be added to our initial set of seedpoints in order to have full coverage. However, the advantage is that most will be filtered out and will not be included in the segmentation result.

These methods of segmentation rely heavily on a good choice of initial seedpoints. In addition, this method is highly computationally expensive.

Probability Density

A digital image can be considered as a two-dimensional array of pixel intensities or colors. From this array, algorithms can be applied to obtain features and statistical information [14] (such as a projection, density, and centroids). However, obtaining these features can be a time-consuming procedure and may not provide the best information for line segmentation.

Probability densities themselves can be considered a feature of a document. It is computed once and information can be obtained directly or they can be used for processing by other methodologies.

Peaks on a probability map of a text document correspond to the lines of text. Valleys correspond to the boundaries between consecutive text lines. In general, the probability density represents the distribution of the lines in a document. The values are higher (denser) within text lines and lower (thinner) in the spaces between the lines and in the margins.

Probability densities are usually continuous for most applications. However, for document processing, the functions are not known. Therefore, a discrete estimate is performed. To generate a probability density for a document, a value is computed using an anisotropic Gaussian kernel defined as [2, 37]:

$$f(x, y) = Ae^{-\left[\frac{(x-b_x)^2}{2\sigma_x^2} + \frac{(y-b_y)^2}{2\sigma_y^2}\right]} \tag{8.1}$$

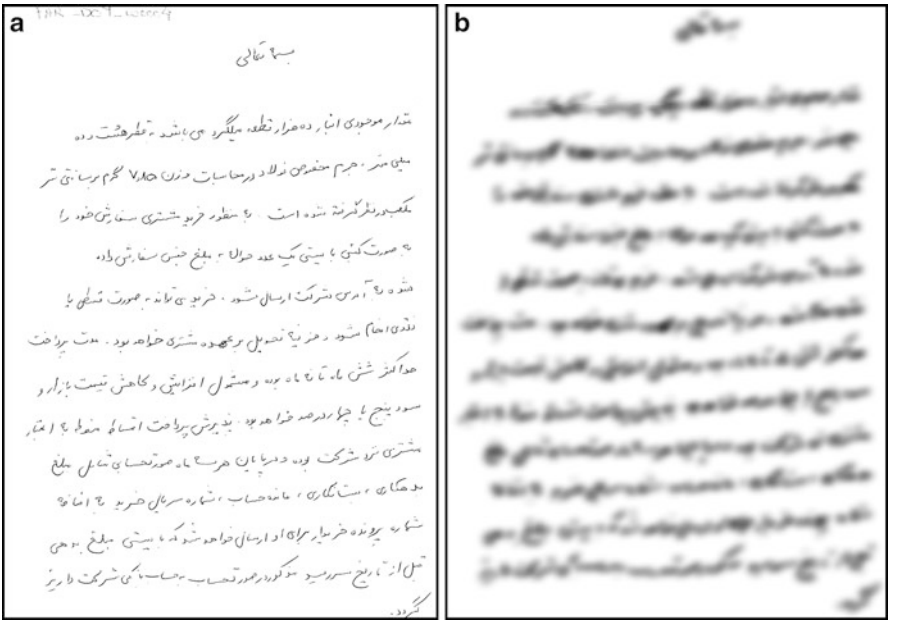

Fig. 8.10 (**a**) Original handwritten document, (**b**) probability density using anisotropic Gaussian probability density function

where b_x is the x-axis shift and b_y is the shift for the y-axis and σ_x and σ_y are the standard deviation-curve spread parameters in the x and y-directions, respectively. A is the amplitude of the function. Note that if $\sigma_x = \sigma_y$, then our Gaussian function will be isotropic. Having an isotropic Gaussian function is undesirable in the case of text line segmentation because text, printed or handwritten, is usually written horizontally with a little skew. This is the case with most languages. Having a large vertical spread will cause text lines, which are close to each other, to merge. Instead, it is preferred to use an anisotropic spread where the height (y-axis) is shorter and the horizontal reach (along the x-axis) is longer. This reduces the chances of consecutive lines being merged and increases the likelihood that pixels on consecutive characters, on the same line, will have a large influence on the probability density.

A new grayscale image is created from applying the Gaussian kernel. This image represents the intensity around the black pixels by expanding the area of influence of a pixel based on all the black pixels in its immediate neighborhood. The more black pixels in a neighborhood, the darker the grayscale image will be in that area. The surrounding pixel intensities will depend on the distance to the original pixel and for this reason, the Gaussian kernel generates a grayscale image representation with intensity values ranging from 0 to 255. The grayscale image is a representation of the probabilities that corresponding pixels in the original document belong to a text line or not. Figure 8.10 shows an original handwritten document and the grayscale

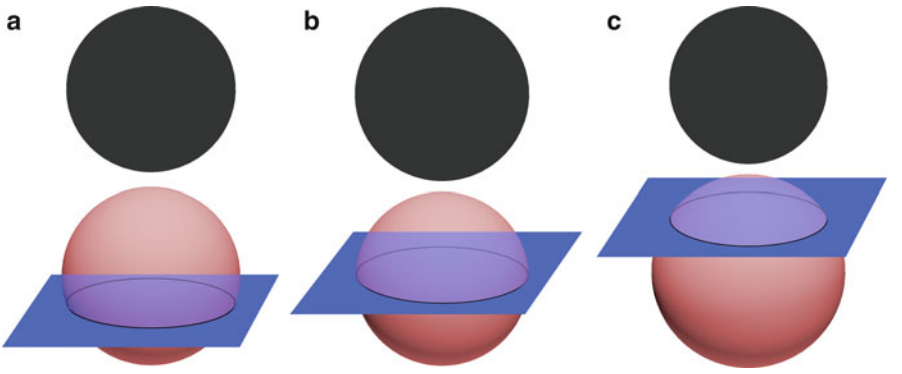

Fig. 8.11 Level set cross sections

image representing the probability density when applying an anisotropic Gaussian probability function.

It can be seen that the text line locations are more prevalent in the probability density image in Fig. 8.10. In addition, noise and pixels in the margins tend to disappear since they are isolated from the text and, therefore, have a little support from neighboring white pixels.

Level Set Method

The level set method [31] was developed by mathematicians Stanley Osher and James Sethian in 1988. The method was conceived to track moving objects and shapes in consecutive video frames and has been extended to image segmentation. It is used to separate foreground objects from the background. Visually, a level set is a cross section (in the *xy*-plane) of a two-dimensional object projected into a three-dimensional model as seen in Fig. 8.11. The corresponding three-dimensional graph is shown below each shape and is called the level set function defining the shape. The three-dimensional surface represents the motion of a two-dimensional shape.

The boundary of the shape is called the zero level set. The level set function is zero at the boundary points. The shape contains the points which are inside the boundary – where the level set function is positive. Any point inside the two-dimensional shape generates a positive level set function value. Any point outside will produce negative level set function values. As time advances, the shape can change form or topology as seen in Fig. 8.11b, c. Keeping track of the transformations of the original shape requires a great deal of work mostly for times when the shapes divide or join. Using the corresponding level set is much easier since it now only needs to track the zero level set (the cross section in the *xy*-plane).

The level set method can be used to separate the text regions from the background for text line segmentation. This is an iterative process where an initial zero level set is evolved according to a partial differential equation. The direction of growth

of the boundary over time is guided by its partial derivatives and an external vector field [3]:

$$\frac{\partial f}{\partial t} + \vec{S} + \nabla f + V_N |\nabla f| = b_k |\nabla f| \tag{8.2}$$

$\frac{\partial f}{\partial t}$ is the boundary movement depending on the vector field $\vec{S}$. The normal direction is represented by the gradient ∇f and the velocity V_N. The curvature is defined by b_k The zero level can therefore grow, shrink, or remain at rest based on Eq. 8.2. Experiments performed by Li et al. [4] showed that boundaries grew faster within text lines – where black pixel densities are large and slower when the gaps are approached. Taking advantage of the knowledge that text lines are horizontally written (for most languages), the boundaries can be set to grow faster in the horizontal direction. In addition, the curvature at the ends of a line is larger than the curvatures at the top and bottom. Knowing this, fewer iterations may be needed to reach the goal.

Segmentation for Optical Character Recognition (OCR)

Any character recognition system is highly dependent on the quality of the segmentation algorithm that precedes it. Segmentation is the process of dividing a document image into smaller elements. The purpose is to simplify a problem into a form which can be more easily manageable. In this chapter, research performed on segmentation of printed characters from a document is discussed. Characters include those found in the ASCII character set as well as some accented characters and symbols. Some research has also been done on segmenting mathematical symbols and characters found in non-Latin documents. Only English documents will be the language we will focus on but the techniques can be applied to most other languages.

Optical Character Recognition (OCR) [26, 38, 39] has been available since 1929. The idea of an OCR system or device has been around since the early twentieth century when several patents for OCR systems were granted. In the 1960s, the OCR-A font was designed to make it easier for both humans to read and for machines to perform OCR on the printed texts. OCR-A is a fixed-width Sans Serif font (see Fig. 8.12).

OCR-A was designed as a simple font with little complications. Each character was evenly spaced and had little extraneous features. Some companies would use this font to print serial numbers, airline tickets, and utility bills, to name a few examples. This limited OCR systems to the commercial environment. Existing printed material, such as literature, business, and legal documents, printed in a font other than OCR-A, was not accurately "read" by the existing OCR systems which were trained specifically on the OCR-A font.

In 1974, the Kurzweil Computer Products company developed an OCR which could read documents printed in any normal font. A few years later, the OCR system was sold commercially to the general public.

Fig. 8.12 OCR-A font sample

OCR has several uses. Some practical examples for OCR systems are license plate recognition, book scanning for digital libraries, mail address reader, text to speech, form processing, computer vision, and sign recognition.

OCR systems have improved greatly since the early pioneer years. However, even the current systems must deal with several obstacles that are inherent with any printed document. There are several problems which can occur that the segmentation procedure must overcome in order to obtain the best segmentation results possible. More on the challenges of segmentation of printed documents are described in section "Region Growth Techniques."

In this section, "Segmentation for Optical Character Recognition (OCR)," some recent research done to overcome the challenges in order to improve printed character segmentation is presented.

Challenges

Recognizing printed texts is not a trivial problem. Several obstacles can appear on a document which hinders the segmentation procedure. Because of the vast number of document types, fonts, paper quality, and even how the documents were digitized, this results in an almost endless number of variations for an OCR system to contend with. Since it is not possible to train on all these combinations, researchers instead train on common features which most documents contain.

Ideally, the characters on a printed page would appear clean and isolated, free from any obstructions or damages. However, this is rarely the case. Problems arise that cause the characters on a page to be fragmented, distorted [33], or even disappear. Below is a list of some of the common problems which a segmentation algorithm must overcome.

Touching Characters

The problems regarding poor document image capture can directly affect the quality of the printed characters on the document image. One common problem, which is of great importance for segmentation, is the separation of touching characters [36]. There are several reasons that lead to touching characters. They occur frequently either "naturally" or as a result of poor image capture. Documents that have text printed using a typewriter may have touching characters as a result of a malfunction such as a small-space advance, misaligned typebars, or even if character keys were pressed simultaneously.

Serif fonts have a higher rate of touching characters because of their serifs. When the serifs of neighboring characters are facing each other, the chance of

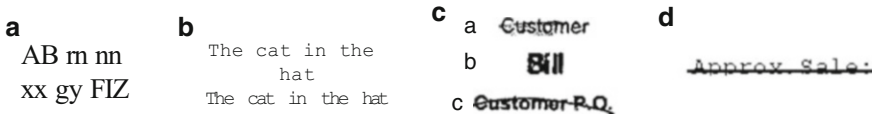

Fig. 8.13 Touching characters resulting from (**a**) Serif font, (**b**) kerning, (**c**) noise, (**d**) strikethrough

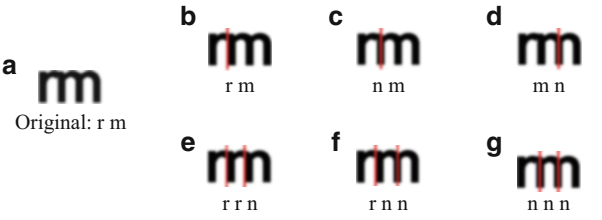

Fig. 8.14 Touching characters with potential cutting points

touching is higher as seen in Fig. 8.13a. Kerning, or the spacing between characters in proportional fonts, is another source of this problem. If the spacing is too small, characters will overlap. Most word processors allow the user to change the default kerning which can lead to more touching characters even for non-serif and fixed-spaced typefaces. Figure 8.13b shows an example of a sample text at regular spacing and when kerning is manually changed.

Noise is any unwanted component on the document image. This includes objects appearing as a result of scanning or photocopying or handwritten strokes overlapping the text. Figure 8.13c shows an example where regular pepper noise can cause characters to touch. The bottom image shows an example where a person crossed out some text causing the characters to be fully connected. Although striking out characters is not really considered as real noise, it has the same effect as the handwritten strike-out as previously mentioned. Figure 8.13d shows a strike-out example.

These examples show some sources responsible for touching characters which are a problem for the segmentation module because it is often difficult to accurately determine the cutting point to split them. Some common difficult cases are "rn" misclassified as the letter "m" and vice versa, "w" with "vv," "ri" and "n," "in" with "m," and "rm" with "nn," to name a few. Figure 8.14 shows the touching character pair "rm" and some potential segmentation cutting candidates.

More cutting points can be found in this example; however, Fig. 8.14 shows an example of the difficulty in printed touching character segmentation.

Broken Characters

Another major concern with segmentation is the case of broken characters. As with touching characters, several reasons can cause characters to be fragmented, for example, uneven ink, faded patches of text from old documents, or damaged paper.

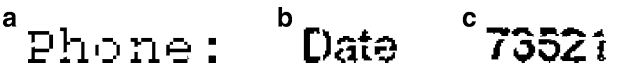

Fig. 8.15 Broken character examples

Fig. 8.16 Text line reference points

Some binarization algorithms during preprocessing will cause these uneven images to develop broken strokes if the binarization threshold is too low, for example.

Repairing broken characters involves an opposite procedure than that for segmenting touching characters. Here, separate components must be merged together to form a character. This could pose a bigger challenge than solving touching characters since some components may not be part of a character but just happened to be in the general area. Figure 8.15 shows some examples of broken characters. Figure 8.15a, b appear to have random breaks in the text. However, a consistent break across the text in Fig. 8.15c may be attributed to a scanner streak or defective scanner.

Lack of Baseline Information

The baseline is an imaginary line where most of the characters reside on. Figure 8.16 shows the location of the baseline and other reference points of a text line. Most of the time, a line will have a combination of characters with different heights which makes distinguishing between upper- and lowercase much easier. However, for lines where all the character heights are the same, distinguishing between uppercase and lowercase for some characters becomes difficult. The characters which have similar upper- and lowercase shapes are C, K, O, S, U, V, W, X, and Z. Additionally, digits "0" and "1," which are often misrecognized as "O" and "I," respectively, also fall into this category.

A line that contains any combination of these characters, mixed with symbols of the same height, may lead to misclassifications in character case. Some examples of case confusing strings are "ZOO" with "zoo," "COWS" with "cows," and "VOX" with "vox." Although the text is correctly recognized in terms of spelling, in some situations, the case may be important if a list of acronyms, codes, or passwords

appear on the document, for example. When all characters in a line have the same height, the segmentation module will not be able to provide the recognizer with the case information.

Typefaces

There are thousands of typefaces available for creating digital documents. There have been studies that show that some fonts are more legible than others using OCRs [5]. The main difficulty lies in the fact that it is time-consuming to train a system on every known typeface and for every character. Additionally, new typefaces are introduced every day, which would require such a system to be retrained. A more generic approach to segmentation is preferred to avoid this obstacle. However, such a system must deal with the variations in the fonts. Font features that a segmentation module must take into account are: Serif and Sans Serif, point size, proportional or fixed spacing, boldness, and italicized text.

As mentioned before, Serif fonts lead to an increase in touching characters. Section "Historical Documents" described in more detail the reasons why this is a problem for segmentation. Similarly, proportional spaced fonts, which have different spacing values for each character, also tend to have more cases of touching characters since the text is more compact than fixed-spaced fonts.

Point size can be a problem when the size is either too small or too large. If too small, then features may not appear as evident as with larger point sizes. Figure 8.17 shows two popular typefaces, Courier New and Times New Roman, at various point sizes. The smallest point size displayed (6) in either font shows how features are less pronounced than the larger characters. Some features which are important to segmentation and OCR systems include stroke widths, curves, loops, and stems for Serif fonts. These features are severely reduced in the smaller fonts and, consequently, provide less information to work with. Furthermore, feature calculations will not be as accurate for smaller fonts such as finding the curvature rate for some strokes.

A common practice in OCR is to binarize, normalize [13], and skeletonize a character image to a fixed bounding box size. Downscaling from a larger size to a smaller size may lead to a loss of information as pixels may be dropped. The bigger problem occurs when an image is scaled up from a small point size to the normalized size. This procedure introduces jaggy edges and blocky pixels. As seen in Fig. 8.18, the letter "a" produces a different shape when the smaller image is scaled up than the larger image. In fact, the normalized image derived from the smaller 6-point image does not resemble the letter "a" anymore. The skeletonized images of the normalized ones also show that the smaller "a" now looks more like the digit "8" or the letter "B" than it does "a." The larger 36-point "a" maintained the shape when scaled up to the normalized size. Even though the edges still appear jagged, the final skeleton appears correct.

Two more features that can lead to problems are bold and italicized fonts. Bold fonts have thicker strokes than the regular normal font. As previously mentioned,

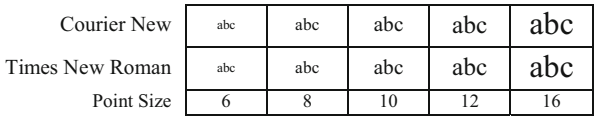

Fig. 8.17 Courier New and Times New Roman typefaces at different font sizes

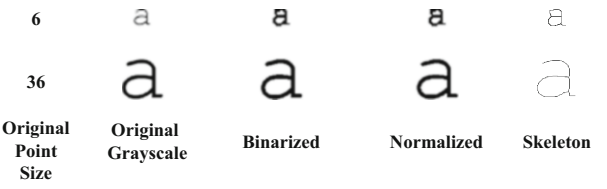

Fig. 8.18 Example of scaling problems due to normalization

Fig. 8.19 (**a**) Bold, and (**b**) italic examples leading to segmentation problems

this leads to more touching characters. Figure 8.19a shows both normal and bold fonts for the Courier New typeface. The normal font for this Serif typeface has no touching characters. When the same text is bolded, the serifs touch. This, as previously discussed, poses a problem for segmentation.

Italicized characters cause challenges for those segmentation algorithms that make use of vertical projection to perform character segmentation. In Fig. 8.19b, the normal font produces a vertical projection with discernible breaks which helps segmentation by easily identifying most character separation spaces. A red line is displayed if the projection in that column contains 1 pixel or less. Only one failed case exists in this example and it is the missed spacing between "f" and "g."

However, the italicized vertical projection loses most of the inter-character spacing which requires the segmentation algorithm to find other ways to separate the characters.

One last issue deals with nonstandard typefaces. Common typefaces are easier to segment because training is usually performed on them. However, when a less frequent or "artistic" typeface is used, segmentation will usually have trouble with character separation due to the fact that it has not seen (was not trained on) this typeface before. Table 8.1 shows examples of four commonly used typefaces (two Serif and two Sans Serif) at the top. The bottom row of the table shows

Table 8.1 Examples of four common typefaces and four rarely used typefaces

Segmentation Method	Segmentation Method	Segmentation Method	Segmentation Method
Times New Roman	**Courier New**	**Arial**	**Frutiger Linotype**
Segmentation Method	Segmentation Method	Segmentation Method	Segmentation Method
Letter Gothic	**Haettenschweiler**	**Impact**	**Harry Potter**

examples of more stylized fonts which are less legible. In fact, it was shown that typefaces with lower human legibility have higher OCR error rates [5].

Touching Italic Characters

In the paper "A Segmentation Method for Touching Italic Characters" [6], the authors describe a method of solving two common problems which occur together – touching and italic characters. Touching italic characters are the seventh most difficult case (out of eight) for the recognition of printed text [7]. The common practices of using vertical projection or connected component and contour analysis alone do not produce acceptable results and are not suitable for italic fonts. Vertical projection is not useful here as was described in Fig. 8.19 of section Typefaces. Slant correction is not used by the authors because, in general, it introduces noise and distorts the original characters. Instead, the authors proposed a combination of slant projection and contour analysis to segment touching italic characters. This proposed method is independent of slant correction and keeps the font information.

The algorithm accepts an image of a printed text line as input. After cleaning and other preprocessing, the authors perform a four-step procedure to segmentation. The first step is to find the slant angle. Again, this is done without the need to deskew the image. Second, the slant projection, combined with contour analysis, is used to find potential segmentation points. A shortest-path approach is used to determine all the potential cut paths. Based on these potential paths, dynamic programming using recognition is employed to find the paths with the highest recognition confidence values. These paths will be considered to be the final segmentation result. Following are more details on each step.

Slant Angle

To determine the slant angle of the italicized text line, the image is rotated by angles between 0° and 30° using steps of 1°. For each rotated angle, a vertical histogram is calculated. When finished, each histogram is normalized to values between 0.0 and 1.0. The variance for each histogram is computed, and the angle which contains the highest variance is considered to be the slant angle. Figure 8.20 shows some rotated images between this range (using 5° steps) and their associated vertical histograms. The red lines in the projection indicate a gap between projection peaks. This usually

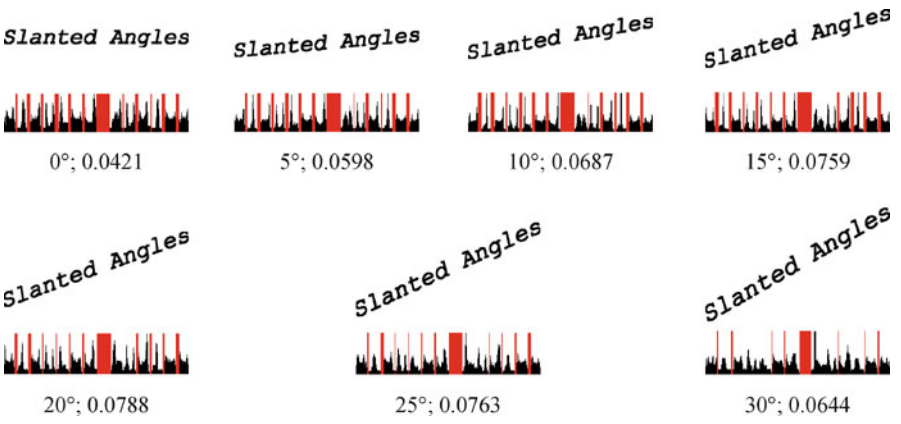

Fig. 8.20 Rotated line images with vertical histograms, rotation angle, and variance values

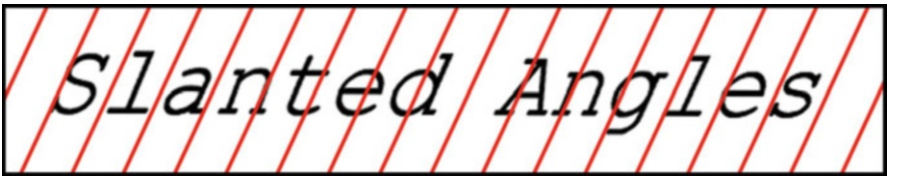

Fig. 8.21 Slant projection lines using angle obtained from Fig. 8.20

indicates a segmentation point. The variance value is computed for each histogram as well. In our limited example, the line image with the highest variance is for the angle at 20°.

Segmentation Points

The next step is to find all the connected components in the line. To determine if a connected component needs to be segmented, the aspect ratio and a classifier confidence output are used. If the aspect ratio is above a threshold value or the classifier confidence value is below a threshold value, the connected component is considered to contain touching characters and therefore segmentation will be performed. To find the segmentation points, the authors used two methods – slant projection and contour feature points.

Slant Projection

Since the slant angle has already been found (see Fig. 8.20), the next step is to compute the slant projection. This is similar to a vertical projection; however, the projection line is along the slant lines. Figure 8.21 shows some of the slant lines from our example of 20°.

The points along a slant line can be computed using the following equation:

$$X_i = X_0 - i^* tan\ \theta \qquad i = 1 \ldots n \tag{8.3}$$

where i is the y-coordinate of a point, θ is the slant angle, in our case 20°, and n is the height of the image. The points X_i that lie on the line starting with point $(X_0, 0)$ can be calculated for all i from $[0, n]$. All slant projection values are computed for $X_0 \epsilon [0, w]$, where $w =$ width of the image.

After the slant projection histogram is computed, it is smoothed and then the local minima points are found. They represent potential segmentation points. Redundant segmentation candidates can be removed if too many appear clustered together. If the distance between two candidates is less than a threshold value, the candidate with the smaller projection value is removed. Additionally, if a vertical line along a segmentation point crosses the character more than twice, it is likely to be incorrect and the candidate point can be removed.

Contour Feature Points

Some true segmentation points may be missed by the slant projection. This oversight usually occurs when two characters are heavily touching and as a result, will have large slant projection values at the points where the characters are touching. The true segmentation point will be missed because those values will exceed the threshold. To compensate for this, contour feature points are located and used to find more candidate segmentation points. Only the outer contour features are used in this step. To find these feature points requires the following steps:

1. Find the outer contour of each connected component.
2. A point on the upper contour is considered a candidate if, when joined with its two neighboring pixels, forms an upward-facing angle (i.e., a "v" shape).
3. A point on the lower contour is considered a candidate if, when joined with its two neighboring pixels, forms a downward facing angle (i.e., a "^" shape).

Figure 8.22 shows the upper and lower candidate feature points. Once the candidate points have been located, they are used to find the cut paths. The traditional method of vertical cutting will not give good performance for italic fonts. Therefore, a curved path is preferred. An ideal cut path is one that goes through the least number of black pixels and follows the slant angle of the italic font. To employ this methodology, penalty costs are used when building a path. Table 8.2 shows the penalty costs for building a segmentation cut path.

Starting from an initial point $(X_0, 0)$, points along the imaginary line which lies along the slant angle can be generated by points $(X_1, 1)$, $(X_2, 2)$, ..., (X_n, n). A normal directional move is a move from one point to a neighboring point that follows the slant angle. A deviated move is one that drifts away from the slant angle. The authors found that the best path can be found by only considering the points from $X_i - 3$ to $X_i + 3$ on any given row. Therefore, each segmentation point can have an associated cut path.

Fig. 8.22 Outer contour with candidate segmentation points

Table 8.2 Segmentation cut path penalty costs

Move to	Direction	Cost
White	Normal	0
Black	Normal	10
White	Deviated	1
Black	Deviated	14

After the connected components have been cut into two, dynamic programming matching is applied. Each segmented component is passed to a neural network classifier and the confidence value returned is used. Those components with high recognition confidence values are kept; the remaining ones are rejected.

Results

This method was applied to 50 strongly touching/italicized text lines collected by CENPARMI. This dataset consists of 311 touching italic connected components – several of which consist of more than 2 touching characters. The total number of characters is 1,969 from all 50 lines. All the true segmentation points were found as well as the cut paths corresponding to them except for two. Recognition results generated some errors due to the simplicity of the neural network used. Table 8.3 shows the recognition rates using different segmentation methods along with the new method proposed by the authors.

Each method sent the segmented connected components to the same simple neural network classifier in order to omit the classifier influence in the segmentation accuracy results. The new method shows great improvement over the others for touching italicized character segmentation.

The proposed method has both advantages and disadvantages; advantages include the fact that no slant correction is performed. This preserves the character images to be used in their original states without introducing noise and character distortion, as is sometimes the case when performing slant correction.

As a disadvantage, this method assumes the text line to be entirely italicized, which is rare in English documents. Therefore, the usefulness of the method is limited. Similarly, all the characters should be slanted at the same angle and written using the same typeface in order for the method to outperform other methods. Practically, the system may be too slow for commercial purposes. For one reason, slant angles are computed 30 times in order to find the best slant angle. This process

Table 8.3 Recognition rates using various methods

Segmentation method	Recognition rate (%)
Vertical projection	78.21
Contour analysis	83.24
Vertical projection and contour analysis	87.77
Method [6]	90.66

includes rotating the image and computing a vertical projection for each of the 30 angles. Secondly, calling a neural network classifier for each connected component also adds to the processing time, and therefore, slows down the entire system.

Segmentation of Degraded Characters

Several researchers have studied the difficult problem of segmenting characters from degraded documents. The source of the degradation could have originated from historical documents [34] or from recent documents that have gone through wear and tear (i.e., coffee spills, sun damage, heat, or other environmental factors) or just through the natural and gradual deterioration of the medium.

In the paper by Moghaddam et al. [8], a method was introduced using multi-level classifiers, and a level set active contour scheme was used to locate the boundaries of the characters in degraded ancient documents. This research aimed to correct two types of problems in segmentation – restoring degraded characters and fixing broken characters.

A degraded grayscale image of a character extracted from a document is input into the algorithm. This degraded image may contain missing pixels and elements from neighboring characters that may appear in the image. Additionally, the average stroke width, w, is provided to the program as well. The aim is to reconstruct the damaged character as close to the true shape as possible. This is done using the level-set method.

The first step is to obtain the average stroke width, w. Since the documents are degraded, the character images will contain noncontinuous and thinned strokes; therefore, the w is reduced to $w/2$ to compensate for this. The strokes are then converted into a structure usable by the level-set method. A Stroke Map (SM) [9] is created for this purpose. A SM contains all possible stroke pixels in an image by using a kernel method and the average stroke width, w.

After the SM has been computed, a Stroke Cavity Map (SCM), which is a representation of all the probable stroke pixels in an image, is generated. This includes all pixels in the SM with those that are between two SM pixels and less than w. The following binary kernel provided by the authors was used to create the SCM:

$$K_{r1,r2}(r) = \begin{cases} 1 & \|r_1 - r_2\| \leq w \;\&\; r \in R(r_1, r_2, t) \\ 0 & \text{otherwise} \end{cases} \tag{8.4}$$

where $R(r_1, r_2, t)$ is a rectangle from r_1 to r_2 of height t. To take advantage of the fact that the provided image is in the center, the SCM is computed by using a modified spatial decay transform [8]. This transform requires the average character width and height in the entire text from which the degraded character image was taken.

When the SCM is completed, the pixel intensities are computed by using a normalized smoothed histogram. Finally, contours in the level set are found by matching those that intersect the surface at the zero level, $z = 0$. This is approximated as a signed distance function $\Phi(r)$. When the level set function $\Phi(r_i) >= 0$, this signifies that the pixel is part of the text; otherwise, it is labeled as a background pixel. The function is guided by a governing equation which attracts stroke pixels and repels background pixels.

The results for character restoration reported by the authors look very promising. Although it does not perform the segmentation on its own, it can provide a huge assistance to algorithms which attempt to locate and segment degraded characters in historical documents. The average stroke width and the pre-segmented character images must be provided in order for this method to function. In addition, the characters must be centered in the image. Testing needs to be improved – in particular, on more samples and on automated outputs. Although the experiments showed promising results, it was performed only on a few manually extracted sample images.

Segmentation of Mathematical Expressions

The methods used to segment regular printed text will provide little help when segmenting most mathematical expressions. Mathematical expressions are composed of several more characters, symbols, and valid overlapping components. The expressions include typefaces with superscript and subscript fonts which make recognition much more difficult due to the normalization problem. Table 8.4 shows some common mathematical formulae and some of the unique properties which make them different from regular printed text such that regular segmentation algorithms would not work on them.

Aside from the superscripts and subscripts, there are objects such as the summation symbol (Σ) and parentheses in Table 8.4(b) and (d) which span more than one row of text and may have some small components below and above it. There also exists some text in the middle of two lines such as ($a_n cos$) in Table 8.4(b). There are numerators and denominators vertically positioned and separated by a dividing line. In addition, new special symbols such as Σ, $\pm$, π, ∞, $\surd$, $\leq$, and $\geq$ and Greek letters are now part of the character set that will need to be recognized and therefore segmented. Mathematical expressions are two-dimensional as opposed to regular text which is just one-dimensional in nature. Segmentation algorithms must now be adapted to segment components in the vertical direction as is shown in some of the examples of Table 8.4. Therefore, methods such as projection will not be very useful for expression segmentation. The segmentation of overlapping and

Table 8.4 Common equation expressions

$a^2 + b^2 = c^2$	$f(x) = a_0 + \sum_{n=1}^{\infty} \left(a_n \cos \frac{n\pi x}{L} + b_n \sin \frac{n\pi x}{L}\right)$	$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
(a)	*(b)*	*(c)*
$(x + a)^n = \sum_{k=0}^{n} \binom{n}{k} x^k a^{n-k}$	$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots, \quad -\infty < x < \infty$	
(d)	*(e)*	

touching characters is the most difficult part of expression segmentation. According to the authors [10], more than half of the misclassifications are caused by touching characters.

A four-step procedure proposed in [10] begins by identifying expressions in regular text. Each located expression is then segmented into its connected components. Touching characters are detected and segmented. Finally, each extracted component is passed to a classifier for recognition.

The main part, the detection of touching characters, is done by passing the component to a classifier for an initial recognition result. The result from the classifier is used to compare the component with specific precomputed features of that recognized character. Some features used for comparison are the aspect ratio (*height/width*) and the peripheral features. If there is a difference in these feature values with the initial recognized touching pair, then the component will be considered as touching and will be passed to the segmentation module. Otherwise, it is regarded as one character and sent directly to the main classifier for recognition.

The segmentation will split a component into two characters. Again, an initial classifier is used to assist and guide the segmentation routine. Several touching character pairs were synthesized and trained on. The segmentation will pass the component to an initial classifier, trained on geometric features, to determine which touching pair it most closely matches with. That is, if the difference is larger than a pre-calculated threshold, it is considered to be a touching character pair candidate. Otherwise, it is believed to be a single character and passed directly to the main classifier. To overcome font variations, the current component is matched with previously recognized components from the same document.

Assuming the component is composed of two touching characters, segmentation is a four-step procedure:

1. Search for the first component character
2. Create a residual image
3. Identify the second component character
4. Verification

Searching for the first character begins by choosing an image from the document of a recognized character. This character is used as a template to see if a match is found within one of the four quadrant areas of the touching character image. If a match is found, then this is considered to be the first component character.

The residual image is created by subtracting a thickened version of the character image found in step one from the touching components. A thickened version of

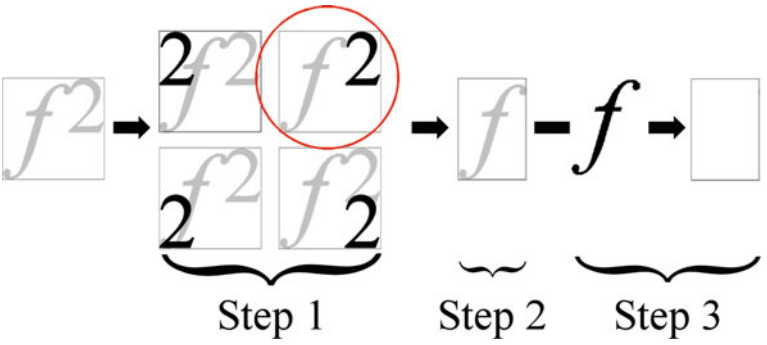

Fig. 8.23 Touching character segmentation using template matching

the known character is used in order to remove the small differences between the two images. This leaves the other character image as the remainder. Using this first residual image, the same step can be applied to find the identity of this remaining character. An image is taken from the set of single classified images from the same document and is matched with an unknown image. If a match is found, the procedure stops and proceeds to the verification step; otherwise, it continues with another known image and repeats the step. A match is found if subtracting the known character from the residual image leaves an empty image. Figure 8.23 shows the steps used to identify the touching characters.

The final step is to verify the result. Verification is done by creating a synthetic image of the touching characters that were identified in the first and third steps and by using two known characters and merging them. The original touching character image is then matched with the synthetic image by subtracting the two images. If the subtraction returns a blank image, then the verification step will have confirmed the segmentation result.

The verification step is useful to "undo" false-positives. If the difference between the touching characters image with the synthesized touching image does not yield a blank image, then the decision is rejected and a new search is performed to find the true touching pair.

If there exists more than one known component to match with, the representative component is used. The representative component is found by a sequential clustering method where each component is added to its closest cluster and clusters are dynamically split when the variance exceeds a predetermined threshold. Each component in a cluster will have the same result in both the detection and segmentation stages. Therefore, one representative (the centroid) component is used for comparison during both stages.

Results reported in [10] show that out of a test set of about 140,000 connected components, 2,978 were touching characters. The initial classifier, used to determine if a component contains touching characters, performed at 92.9 % – where 60 % of the misrecognitions were due to touching characters. Using the clustering method,

Fig. 8.24 Touching pair containing a broken character

the 140,000 components were grouped into 13,291 clusters. Of these, 909 clusters contained touching characters.

The performance was 96 % successful in detecting touching characters. For segmentation, about 51 % of the actual touching characters were successfully segmented into their two characters. Although not a high value, this does reduce the errors due to touching characters by half. Analysis showed that the three major reasons for the segmentation failure of the 49 % touching characters were (i) either one or both of the characters were not represented in the known isolated characters, (ii) the subtraction step of correct matching character images produced a small but significant remainder that was enough to declare a non-match, and (iii) touching characters were segmented into two incorrect characters (i.e., see Fig. 8.14). Of the three causes, the first was responsible for 40 % of the total errors.

Recognition was reported to be 95.1 %. This technique reduced the number of misrecognitions to 70 % from the initial classifier.

One advantage of this technique is the adaptiveness to the document font style and size. Characters used for matching come from the same document so excessive training is not required. However, this assumes that each character in a touching pair has a known isolated representative character. For short documents, the probability that an isolated representative character exists is smaller.

Further investigation showed that the speed increased the overall performance by one-tenth when not using clustering.

The limitation of the technique is bounded by the availability of known characters to match with those in the touching pairs. When a known character is missing from the document, this causes the technique to fail. The authors claim that the technique can be extended to segmentation of three touching characters. While this is possible, it adds an order of complexity to the system especially because of the large character set being used. Additionally, documents that contain broken characters will fail. For example, Fig. 8.24 shows the touching character pair "DO" with the "O" broken. Using the template-matching technique will not detect the "O" because subtracting a known identified "O" from this one will leave too many leftover pixels. Therefore, the "O" will not be classified, or worse, misclassified.

Conclusion

This chapter presented research and approaches to solving common problems associated with printed document zoning, line, and character segmentation. The solutions must overcome the vast diversity of document layouts, printing

media, and resolution. They must also be able to handle documents written in any language [19]. Historical documents have a set of difficulties such as degraded media and possibly physical missing sections from wear and tear. Wrinkles and creases in the paper will cause problems for segmentation algorithms.

Once these obstacles have been overcome and the text zones have been identified, text segmentation begins. Various techniques for performing this task were discussed. The structural-based procedure using horizontal projection performs very well but requires that the text lines not overlap each other and have a relatively horizontal orientation.

Region growth techniques are an alternative structural method which group neighboring pixels of an image into subregions. Pixels in a subregion share a common feature. Although region growth techniques are better for handling touching, skewed, and overlapping lines, these segmentation methods rely heavily on a good choice of initial seedpoints. Furthermore, they are highly computationally expensive. An advantage is that most noise will not be included in the segmentation result.

A common statistical approach is to use probability density features. By treating a document as a two-dimensional array of pixel intensities, algorithms are used to extract statistical information from this array. These features can be used to determine the most likely line and edge boundaries of text. Probability densities are good for eliminating noise and pixels which are distant from any text, such as pixels in the margins. They are also well suited for handling broken characters within a line. However, obtaining these features can be a time-consuming procedure and may not provide the best information for line segmentation.

Level set methods are used to separate foreground objects from the background in a document. This is ideal for text line segmentation to separate text lines in front of other elements such as a photo or ghost images. The procedure is an iterative process beginning with an initial zero level set and is evolved according to a partial differential equation. This has the disadvantage of having a high computational requirement. The advantage is shown where it is observed that the boundary of a level set grew faster inside text lines that is, where black pixel densities are large. Slower growth was observed when the gaps were approached.

In addition, the segmentation challenges faced for character recognition was described. It was mentioned that several factors affect segmentation such as the typeface, font size, and bold and italicized text. Because of these and other features, this leads to structural problems such as broken and touching characters. A lack of baseline information (when all characters in a text line are the same height) can also impede segmentation. Several algorithms to solve these types of problems as well as slanted lines were pointed out. Degraded characters, which are prevalent in historical documents or documents which have gone through a great deal of wear and tear, need to be specially handled in order to reconstruct missing and damaged characters. Alternative special cases are mathematical expressions. The text layout is not in the form as a regular text block. More than one line can be contained within a larger line. Subscripts, superscript, symbolic characters, and mathematical symbols are introduced. In addition, the kerning (spacing between the characters) is

not predictable as it is with a known typeface. Because of the variation of kerning and font sizes, the majority of misclassifications are caused by touching characters.

There has been a great amount of research on segmentation. Several solutions to common segmentation difficulties were presented. It is unlikely any one solution will achieve a complete result. It is common practice that more than one algorithm is used to obtain the best segmentation possible.

Cross-References

▸Analysis of the Logical Layout of Documents
▸Imaging Techniques in Document Analysis Process
▸Machine-Printed Character Recognition
▸Page Segmentation Techniques in Document Analysis

References

1. Li Y, Zheng Y, Doermann D, Jaeger S (2008) Script-Independent text line segmentation in freestyle handwritten documents. IEEE Trans Pattern Anal Mach Intell 30(8):1313–1329
2. Brodić D (2010) Optimization of the anisotropic Gaussian kernel for text segmentation and parameter extraction. In: Theoretical computer science. Springer, Brisbane, pp 140–152
3. Sumengen B (2004) Variational image segmentation and curve evolution on natural images. Ph. D. Thesis, University of California, Santa Barbara
4. Li Y, Zheng Y, Doermann D, Jaeger S (2006) A new algorithm for detecting text line in handwritten documents. In: Tenth international workshop on frontiers in handwriting recognition, La Baule, pp 35–40
5. Suen C, Nikfal S, Li Y, Zhang Y, Nobile N (2010) Evaluation of typeface legibility. In: ATypI, Dublin, Sept 2010
6. Li Y, Naoi S, Cheriet M, Suen C (2004) A segmentation method for touching Italic characters. In: International conference on pattern recognition (ICPR), Cambridge, pp 594–597, Aug 2004
7. Lu Y (1995) Machine printed character segmentation – an overview. Pattern Recognit 28: 67–80
8. Moghaddam R, Rivest-Hénault D, Cheriet M (2009) Restoration and segmentation of highly degraded characters using a shape-independent level set approach and multi-level classifiers. In: International conference on document analysis and recognition (ICDAR), Barcelona, pp 828–832, July 2009
9. Moghaddam R, Cheriet M (2009) RSLDI: restoration of single-sided low-quality document images. Pattern Recognit 42(12):3355–3364
10. Nomura A, Michishita K, Uchida S, Suzuki M (2003) Detection and segmentation of touching characters in mathematical expressions. In: Seventh international conference on document analysis and recognition – ICDAR2003, Edinburgh, pp 126–130
11. Ball G, Srihari S, Srinivasan H (2006) Segmentation-Based and segmentation-free approaches to Arabic word spotting. In: Proceedings of the international workshop on frontiers in handwriting recognition (IWFHR-10), La Baule, pp 53–58, Oct 2006
12. Liu C, Suen C (2008) A new benchmark on the recognition of handwritten Bangla and Farsi numeral characters. In: Proceedings of eleventh international conference on frontiers in handwriting recognition (ICFHR 2008), Montreal, pp 278–283
13. Liu C, Nakashima K, Sako H, Fujisawa H (2004) Handwritten Digit Recognition: Investigation of Normalization and Feature Extraction Techniques. Pattern Recognition 37(2):265–279

14. McLachlan G (1992) Discriminant Analysis and Statistical Pattern Recognition. Wiley Interscience, New York
15. Shi M, Fujisawa Y, Wakabayashi T, Kimura F (2002) Handwritten Numeral Recognition Using Gradient and Curvature of Gray Scale Image. Pattern Recognition 35(10):2051–2059
16. Li Y, Zheng Y, Doermann D (2006) Detecting Text Lines in Handwritten Documents. In: International Conference on Pattern Recognition, Hong Kong, vol 2, pp 1030–1033
17. Likforman-Sulem L, Vinciarelli A (2008) HMM-based Offline Recognition of Handwritten Words Crossed Out with Different Kinds of Strokes. In: Eleventh International Conference on Frontiers in Handwriting Recognition, Montreal, pp 70–75
18. Zheng D, Sun J, Naoi S, Hotta Y, Minagawa A, Suwa M, Fujimoto K (2008) Handwritten Email address recognition with syntax and lexicons. In: Eleventh international conference on frontiers in handwriting recognition, Montreal, pp 119–124
19. Kessentini Y, Paquet T, Benhamadou A (2008) A multi-stream HMM-based approach for off-line multi-script handwritten word recognition. In: Eleventh international conference on frontiers in handwriting recognition, Montreal, pp 147–152
20. Fei Y, Liu C-L (2008) Handwritten text line segmentation by clustering with distance metric learning. In: Eleventh international conference on frontiers in handwriting recognition, Montreal, pp 229–234
21. Roy P, Pal U, LLados J (2008) Morphology based handwritten line segmentation using foreground and background information. In: Eleventh international conference on frontiers in handwriting recognition, Montreal, pp 241–246
22. Du X, Pan W, Bui T (2008) Text line segmentation in handwritten documents using Mumford-Shah model. In: Eleventh international conference on frontiers in handwriting recognition, Montreal, pp 253–258
23. Liu C-L, Suen C (2008) A new benchmark on the recognition of handwritten Bangla and Farsi numeral characters. In: Eleventh international conference on frontiers in handwriting recognition, Montreal, pp 278–283
24. Mori S, Nishida H, Yamada H (1999) Optical character recognition. Wiley-Interscience, New York
25. Chaudhuri B (2007) Digital document processing: major directions and recent advances. Springer, London
26. Bunke H, Wang P (1997) Handbook of character recognition and document image analysis. World Scientific, Singapore
27. Garain U, Paquet T, Heutte L (2006) On foreground – background separation in low quality document images. Int J Doc Anal Recognit 8(1):47–63
28. Morita M, Sabourin R, Bortolozzi F, Suen C (2004) Segmentation and recognition of handwritten dates: an HMM-MLP hybrid approach. Int J Doc Anal Recognit 6(4):248–262
29. Hase H, Yoneda M, Tokai S, Kato J, Suen C (2004) Color segmentation for text extraction. Int J Doc Anal Recognit 6(4):271–284
30. Sarhan A (2009) Arabic character recognition using a combination of k-means and k-NN algorithms. Int J Comput Process Lang 22(4):305–320
31. Karthik S, Hemanth V, Balaji V, Soman K (2012) Level set methodology for Tamil document image binarization and segmentation. Int J Comput Appl 39(9):7–12
32. Ouwayed N, Belaïd A (2008) Multi-Oriented text line extraction from handwritten Arabic documents. In: Eighth IAPR international workshop on document analysis systems, Nara, pp 339–346
33. Pan P, Zhu Y, Sun J, Naoi S (2011) Recognizing characters with severe perspective distortion using hash tables and perspective invariants. In: International conference on document analysis and recognition, Beijing, pp 548–552
34. Silva G, Lins R (2011) An automatic method for enhancing character recognition in degraded historical documents. In: International conference on document analysis and recognition, Beijing, pp 553–557
35. Saabni R, El-Sana J (2011) Language-Independent text lines extraction using seam carving. In: International conference on document analysis and recognition, Beijing, pp 563–568

36. Kang L, Doermann D (2011) Template based segmentation of touching components in handwritten text lines. In: International conference on document analysis and recognition, Beijing, pp 569–573
37. Bukhari S, Shafait F, Breuel T (2011) Text-Line extraction using a convolution of isotropic Gaussian filter with a set of line filters. In: International conference on document analysis and recognition, Beijing, pp 579–583
38. Marinai S, Fujisawa H (eds) (2010) Machine learning in document analysis and recognition, 1st edn. Studies in computational intelligence, vol 90. Springer, Berlin
39. Cheriet M, Kharma N, Liu C-L, Suen C (2007) Character Recognition systems: a guide for students and practitioners. Wiley, Hoboken

Further Reading

Bunke H, Wang P (1997) Handbook of character recognition and document image analysis. World Scientific, Singapore

Chaudhuri B (2007) Digital document processing: major directions and recent advances. Springer, London

Cheriet M, Kharma N, Liu C-L, Suen C (2007) Character recognition systems: a guide for students and practitioners. Wiley, Hoboken

Li H, Doermann D, Zheng Y (2008) Handwritten document image processing: identification, matching, and indexing of handwriting in noisy document images. VDM, Saarbrücken

Marinai S, Fujisawa H (eds) (2010) Machine learning in document analysis and recognition, 1st edn. Studies in computational intelligence, vol 90. Springer, Berlin