

Chapter 2

Video Preprocessing

Video text detection is one of the hot spots in pattern recognition, and a variety of approaches have been proposed for various real-life applications such as content-based retrieval, broadcast or game video analysis, license plate detection, and even address block location. However, in spite of such extensive studies, it is still not easy to design a general-purpose method [1]. This is because there are so many possibilities when extracting video texts that have variations in font style, size, color, orientation, and alignment or from a textured background with low contrast. Moreover, suppose the input of a text detection system is a sequence of video frames, the texts in the frames may or may not move, and the video itself can be in low resolution. These variations make the problem of automatic video text detection extremely difficult, and we thus need video preprocessing to reduce the complexity of the succeeding steps probably consisting of video text detection, localization, extraction, tracking, reconstruction, and recognition.

Figure 2.1 shows a few typical preprocessing stages in video text detection [2]. After inputting an original video frame (Fig. 2.1a), image segmentation techniques are employed to theoretically extract all the pixels belonging to text (Fig. 2.1b). Without knowing where and what the characters are, the aim of the segmentation step here is to initially divide the pixels of each frame from a video clip into two classes of regions which do not contain text and regions which potentially contain text. For other methods that consider video scene structures as scene contexts for text detection, a more general segmentation algorithm is potentially required to partition each video scene image into several meaningful parts such as street, building, and sky to reduce the searching space. After segmentation, each video frame can be considered to consist of homogeneous segments. By filtering the monochrome segments whose widths and heights are too large or too small to be instances of video characters (Fig. 2.1c), binarization and dilation are performed as shown in Fig. 2.1d. Sometimes motion analysis is also adopted to detect corresponding character candidate regions from consecutive frames since even stationary text in video may still move by some pixels around its original position from frame to

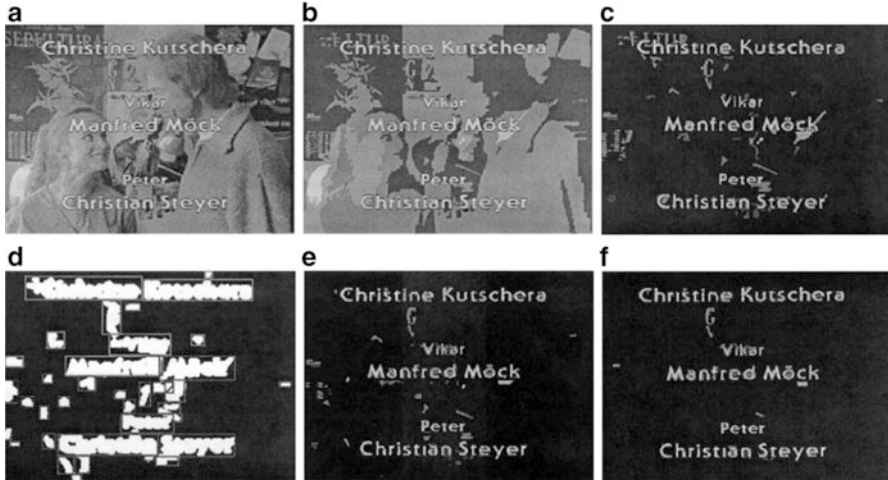


Fig. 2.1 Preprocessing stages for video text detection. (a) Original video frame, (b) image segmentation, (c) after size restriction, (d) after binarization and dilation, (e) after motion analysis, and (f) after contrast analysis and aspect ratio restriction [2]

frame (Fig. 2.1e). Moreover, to solve the problems like occlusion and arbitrary text directions, motion analysis is helpful by tracing text from temporal frames. Finally, image enhancement techniques such as contrast analysis and aspect ratio restriction (Fig. 2.1f) can be employed to obtain better preprocessing results for further video text detection or recognition. For example, the manually added graphics texts usually have a strong contrast between the character regions and their surrounding backgrounds for highlights and human reading. This property makes contrast analysis helpful to decide whether directly discarding a video segment or instead sending the segment for further video text detection and character recognition.

This chapter gives a brief overview of the abovementioned preprocessing techniques that are often used in video text detection. It is organized as follows. We first introduce some image preprocessing operators in Sect. 2.1. Sections 2.2 and 2.3 then discuss color-based and texture-based preprocessing techniques, respectively. Since image segmentation plays an important role in video text detection, we introduce several image segmentation approaches in Sect. 2.4. Next, Sect. 2.5 discusses motion analysis which is helpful to improve the efficiency or the accuracy of video text detection by tracing text from temporal frames. Finally, Sect. 2.6 concludes this chapter.

2.1 Preprocessing Operators

Image (or video frame) preprocessing is the term for operations on images (or video frames) at the lowest level of analysis. Generally, the aim of preprocessing operators for video text detection is an improvement of the image data by suppressing

undesired degradations and simultaneously enhancing specific text relevant features. In this section, we will introduce several typical preprocessing operators that are either important for enhancing image features or helpful in suppressing information that is not relevant to video text detection.

2.1.1 Image Cropping and Local Operators

Image cropping is generally the first step after inputting a video frame. In this step, irrelevant parts in the image will be cropped such that further processing focuses on the regions of interest and thereby the computational cost is reduced.

Next, image preprocessing transformations are performed on the rest of the regions. The simplest kind of image preprocessing transformations is *local operator* [3], where the value of each output pixel only depends on that of its corresponding input pixel. Let x be the position of a pixel in an image and $f(x)$ be the value of the pixel; in the continuous domain, a local operator can be represented by

$$h(x) = g(f(x)) \quad (2.1)$$

where the function of g operates over some range, which can either be scalar or vector valued (e.g., for color images or 2D motion). The commonly used local operators are multiplication and addition with constants [4]:

$$g(x) = af(x) + b \quad (2.2)$$

where the two parameters of a and b can be respectively regarded as the gain and the bias parameters. Examples of local operators include *brightness adjustment* and *contrast enhancement* as shown in Fig. 2.2, where the brightness of every pixel in Fig. 2.2a is recalculated by $g(x) = x + 20$ in Fig. 2.2b, while the contrast is enhanced by $g(x) = x * 1.5$ in Fig. 2.2c.

Another commonly used operator is the linear blend operator, which is designed to *cross-dissolve two video frames*:

$$g(x) = (1 - \alpha) f_0(x) + \alpha f_1(x) \quad (2.3)$$

By varying α from 0 to 1, it can essentially be considered as an image morphing algorithm. To automatically determine the best balance between the brightness and the gain control, one approach is to average all the intensity values in a video frame, turn the result to middle gray, and finally expand the range to fill all the displayable values. We can visualize the color of the frame by plotting the histogram of each individual color channel as well as the luminance values [5]. Based on these distributions, relevant statistics such as the minimum, the maximum, and the average intensity values can be computed. Accordingly, histogram equalization can be performed to find an intensity mapping function such that the resulting histogram

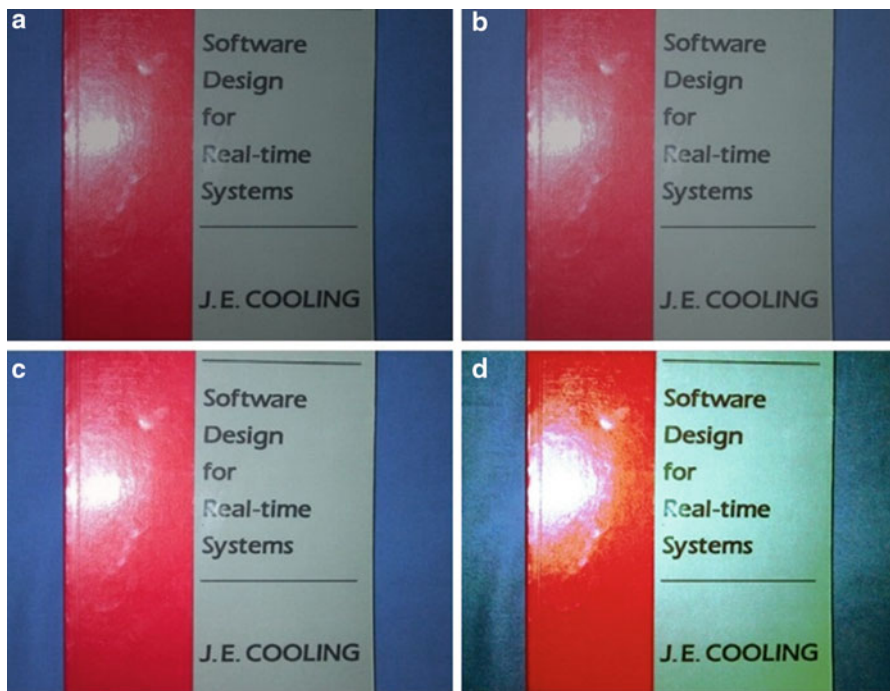


Fig. 2.2 Example local operators for image (video frame) preprocessing. (a) Original image, (b) the result by increasing the brightness on the original image $g(x) = x + 20$, (c) the result by increasing the contrast $g(x) = x * 1.5$, and (d) after histogram equalization

is flat. Figure 2.2d shows the result of applying *histogram equalization* on the original image in Fig. 2.2a. As we can see, the resulting image becomes relatively “flat” in the sense of the lacking of contrast.

2.1.2 Neighborhood Operators

Neighborhood operators are often designed to remove noises, sharpen image details, or accentuate image edges. The most commonly used neighborhood operator is *linear filter*, in which the value of an output pixel depends on a weighted sum of the input pixel values by

$$g(i, j) = \sum_{k,l} f(i - k, j - l) h(k, l) \quad (2.4)$$

where $h(k, l)$ is called the weight kernel or the mask, k and l are two parameters for defining the range of neighbors.

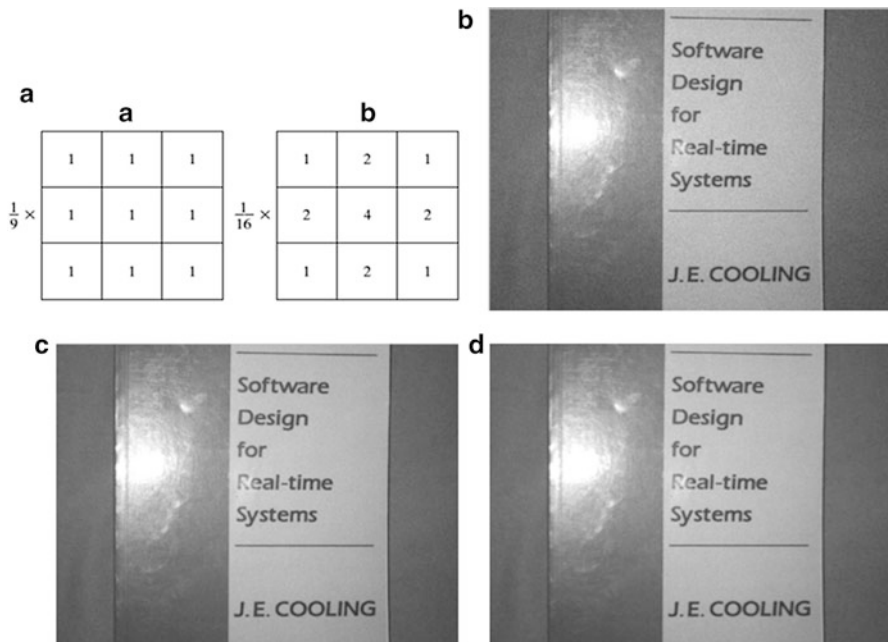


Fig. 2.3 Some example linear filters. (a) Two examples of average filters, (b) an example video frame with Gaussian noises, (c) the result after using the average filter with the second mask in (a), and (d) the result after using the median filter

The simplest linear filter used in video text detection is the *average filter*, which simply averages all the pixel values in a $K \times K$ window. It can be used for *noise reduction* or *image blurring* by filling the holes in lines or curves and thus remove unimportant details from an image. In Fig. 2.3, we show two example 3×3 average filters. The first filter *a* in Fig. 2.3a yields the standard average of the pixels under the mask

$$R = \frac{1}{9} \sum_{i=1}^9 Z_i \quad (2.5)$$

which is the average of the gray levels of the pixels in the 3×3 neighborhood. The second mask *b* in Fig. 2.3a yields a so-called weighted average, which assigns different importance weights to pixels by multiplying different coefficients. *Median filter* is another choice for smoothness, which selects the median value from the neighborhood as the output for each pixel. It can be used to filter certain types of random noises in video frame as well. Figure 2.3b gives an example video frame with Gaussian noises, and Fig. 2.3c shows the result after using the average filter with the mask *b* in Fig. 2.3a. The result after using the median filter is shown in Fig. 2.3d.

The principal objective of *image sharpening* is to highlight details in a video frame or enhance the details that have been blurred. A basic definition of the first-order derivative of a one-dimensional function $f(x)$ is based on the difference

$$\frac{\partial f}{\partial x} = f(x+1) - f(x) \quad (2.6)$$

Theoretically, the value of the derivative operator is proportional to the degree of discontinuity at the point x . In this way, the differentiation operator enhances edges and other discontinuities (e.g., noises), while the regions with slowly varying gray values are de-emphasized.

One commonly used first-order differentiation operator is proposed by Roberts [6] as

$$G_x = (z_9 - z_5) \quad (2.7)$$

$$G_y = (z_8 - z_6) \quad (2.8)$$

where the positions of pixels are shown in Fig. 2.4a. Generally, the *Roberts operator* is calculated by

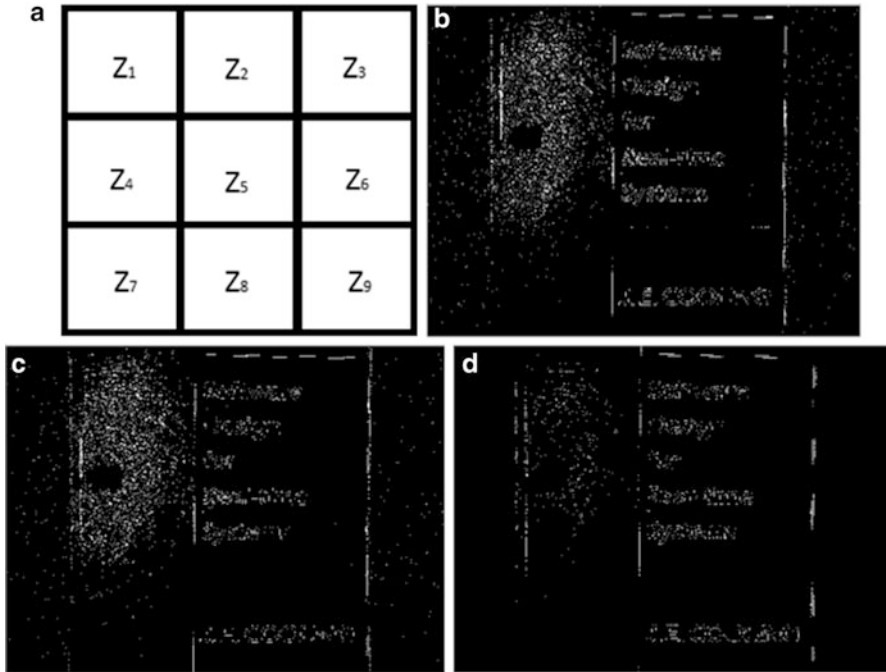


Fig. 2.4 Some filter examples. (a) One example table for demonstrating the filters, (b) the result example video frame (same with Fig. 2.3b) using Roberts filter, (c) the result using Sobel filter, and (d) the result using Laplacian filter

$$\nabla f \approx |z_9 - z_5| + |z_8 - z_6| \quad (2.9)$$

which is an approximation to the gradient.

Instead, another commonly used first-order *Sobel operator* [7] approximates the gradient by using absolute values at point z_5 with a 3*3 mask as follows:

$$\begin{aligned} \nabla f \approx & |(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)| \\ & + |(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)| \end{aligned} \quad (2.10)$$

where the idea behind is assigning a weight value to achieve smoothing and thereby giving more importance to the center point.

Following the similar idea of the differentiation operator, we can easily define the isotropic derivative operator in the second order by

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad (2.11)$$

For image or video frame processing, it can be calculated as the difference between pixels by

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x) \quad (2.12)$$

which is called the *Laplacian filter*. We show the results filtered from Fig. 2.4 using Roberts, Sobel, and Laplacian operators in Fig. 2.4b–d, respectively.

2.1.3 Morphology Operators

The morphological operators [8] change the shape of an underlying object to facilitate further video text detection or optical character recognition. To perform such an operation, we often need first convert a colorful video frame into a binary image, which has only two elements of either a real foreground or a background complementary with respect to the other. It can be produced after thresholding as follows:

$$\theta(f, c) = \begin{cases} 1 & \text{if } f > c \\ 0 & \text{else} \end{cases} \quad (2.13)$$

We then convolve the binary image with a binary structuring element and accordingly select a binary output value. The structuring element can be any shape, from a simple 3*3 box filter to more complicated disc structures. Let f be the binary image after thresholding, s be the structuring element, S be the size of the structuring element (number of pixels), and $c = f * s$ be the integer-valued count of the number

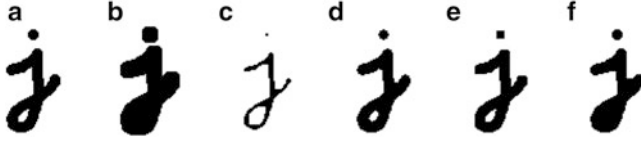


Fig. 2.5 Binary image morphology examples: (a) the original image f , (b) dilation, (c) erosion, (d) majority, (e) opening, (f) closing [4]

of 1s inside each structuring element as it is scanned over f ; the standard operators for binary morphology can be represented as follows [4]:

1. **Dilation:** $dilate(f, s) = \theta(c, 1)$
2. **Erosion:** $erode(f, s) = \theta(c, S)$
3. **Majority:** $maj(f, s) = \theta(c, S/2)$
4. **Opening:** $open(f, s) = dilate(erode(f, s), s)$
5. **Closing:** $close(f, s) = erode(dilate(f, s), s)$

Figure 2.5b–f shows the results after using the five binary image morphology operators on the original character image f in Fig. 2.5a with a $3 * 3$ structuring element s . As we can see from Fig. 2.5, the dilation operator grows (thickens) the character consisting of 1s, while the erosion operator shrinks (thins) it. The opening operator tends to smooth its contour and thus eliminate thin protrusions. The closing operator also tends to smooth contours but simultaneously strengthens thin gulfs and eliminates small holes by filling the gaps on the contour generally.

2.2 Color-Based Preprocessing

Gray or intensity plays an important role in video text analysis. For a binary video frame, it is relatively easy to cluster the pixels in the foreground into blocks which are adjacently linked together and thus segment them from the background for succeeding character recognition. However, for most video frames which can have several objects with different color or gray-level variations in illumination, color-based preprocessing techniques are necessary to reduce the complexity of succeeding video text detection and recognition.

A color image is built of several color channels, each of them indicating the value of the corresponding channel. For example, an RGB image consists of three primary color channels of red, green, and blue; an HSI model has three intuitive color characteristics as hue, saturation, and intensity; while a CMYK image has four independent channels of cyan, magenta, yellow, and black. These color representations are designed for specific devices and have no accurate definitions for a human observer. Therefore, color transformations are required to convert the representation of a color from one color space to another more intuitive one.

For video text detection task, the most common color transformation is the conversion from an RGB color model to a grayscale representation for performing other video preprocessing operators. For an RGB color, the lightness method calculates its grayscale value by

$$\text{gray} = \frac{\max(R, G, B) + \min(R, G, B)}{2} \quad (2.14)$$

The average method simply averages the RGB values by

$$\text{gray} = \frac{R + G + B}{3} \quad (2.15)$$

For better human perception, some methods give a weighted average on different RGB channels based on the theory that humans are more sensitive to green than other colors. For example, Hasan and Karam [9] present a morphological approach for text extraction. In their method, the RGB components of a color input image are combined to give

$$\text{gray} = 0.299R + 0.587G + 0.114B \quad (2.16)$$

Sometimes RGB color needs to be converted to the HIS space for text analysis. The RGB to HSI transformation is

$$\begin{cases} H = \arctan\left(\frac{\beta}{\alpha}\right) \\ S = \sqrt{\alpha^2 + \beta^2} \\ I = (R + G + B) / 3 \end{cases} \quad (2.17)$$

with

$$\begin{cases} \alpha = R - \frac{1}{2}(G + B) \\ \beta = \frac{\sqrt{3}}{2}(G - B) \end{cases} \quad (2.18)$$

Note that although color transformation is relatively simple and many researchers have adopted it to deal with color video frames, it still faces difficulties especially when the objects in the same video frame have similar grayscale values [1]. Figure 2.6 shows an example, where Fig. 2.6a is a color image and Fig. 2.6b is its corresponding grayscale image. Some text regions that are prominent in the color image become difficult to detect in the gray-level image after color transformation. To solve this problem, Shim et al. [10] use the homogeneity of intensity of text regions in images. Pixels with similar gray levels are merged into a group. After removing significantly large regions by regarding them as background, text regions are sharpened by performing a region boundary analysis based on the gray-level contrast.

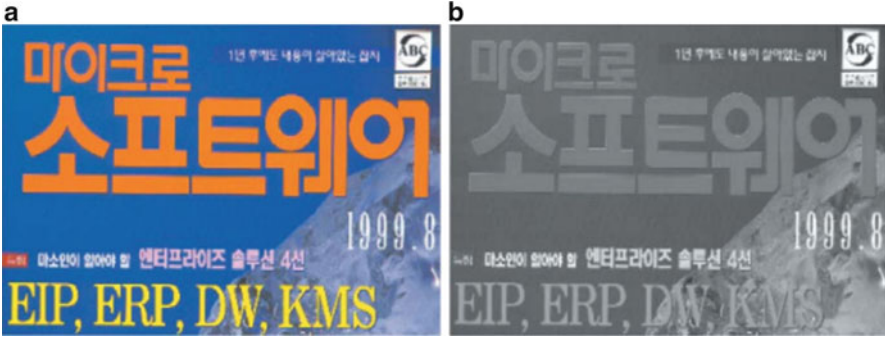


Fig. 2.6 Difference between color image and its grayscale image for video text detection: (a) color image, (b) corresponding grayscale image [1]



Fig. 2.7 A multicolored image and its element images: (a) an input color image, (b) nine element images [12]

Color clustering is another frequently adopted technique in video preprocessing. Kim [11] segments an image using color clustering by a color histogram in the RGB space. Non-text components, such as long horizontal lines and image boundaries, can be eliminated, potentially facilitating the tracking of horizontal text lines and text segments. Jain and Yu [12] apply bit dropping on a 24-bit color image to convert it to a 6-bit image and then quantize it by a color clustering algorithm. After the input image is decomposed into multiple foreground images, each foreground image goes through the same text localization stage. Figure 2.7 shows an example of the multi-valued image decomposition in their method.

Finally, other color-based preprocessing techniques like color reduction can also help enhance video frames for text analysis in some cases. For example, Zhong et al. [13] use color reduction to track the connected components (CCs) for text detection. They quantize the color space using the peaks in a color histogram in the RGB color space. This is based on the assumption that the text regions cluster together in this color space and occupy a significant portion of an image. Each text component goes through a filtering stage using heuristics, such as area, diameter, and spatial alignment.

2.3 Texture Analysis

Texture is an important visual clue for video analysis, and a number of texture features have been proposed in the past decades such as Gabor filters, wavelet, FFT, and spatial variance. These texture analysis techniques can potentially be helpful for video text detection by observing the fact that text regions in video frames generally have distinguished textural properties from their backgrounds. In this subsection, we prefer to introduce the situations of the recent research which are related to video text detection, rather than giving a more general introduction to texture analysis techniques.

Generally, along a horizontal scan line in a video frame, a text line consists of several interleaving groups of text pixels and background pixels. Accordingly, in [13], *spatial variances* along every horizontal line are used to distinguish text regions from the background. Another interesting observation on image text is that for a text line, the number of text-to-background and background-to-text transitions should be the same, and these transitions can be obtained by zero crossings of row gradient profiles. Figure 2.8 shows an example, where the left part of the row gradient profile related to a shrimp is irregular compared to the right representing gradients of the string of *that help you* [14]. Thereby, text lines can be found by extracting the rows having regularly alternate edges.

Based on the discussed observations, the *maximum gradient difference* (MGD) text feature for text detection is presented [14]. MGD is defined for any one pixel, as the difference between the max and the min values within a local $1 \times N$ window of the gradient image which can be obtained by any gradient extractor. Specifically, MGD is defined as follows:

$$Max(x, y) = \max_{t \in [-\frac{N}{2}, \frac{N}{2}]} g(x, y - t) \quad (2.19)$$

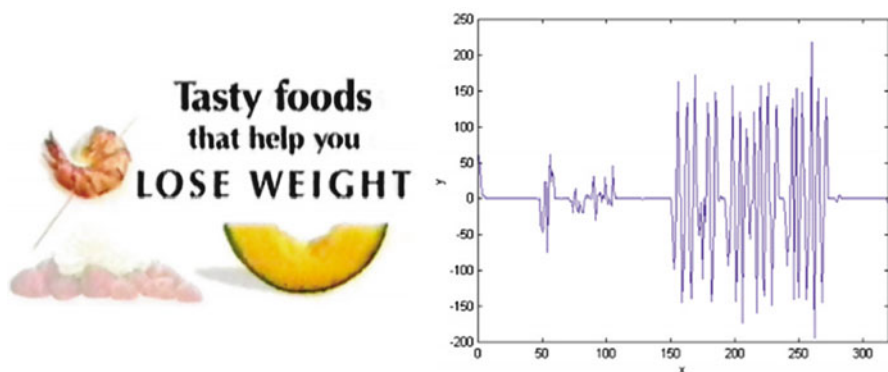


Fig. 2.8 An example image and its gradient profile of the horizontal scan line $x = 80$ [14]

$$Min(x, y) = \min_{t \in [-N/2, N/2]} g(x, y - t) \quad (2.20)$$

$$MGD(x, y) = Max(x, y) - Min(x, y) \quad (2.21)$$

Typically, pixels of text regions have large MGD values, while background regions have small MGD values. The computed MGD values will be helpful to search for potential text regions. For example, Shivakumara et al. [15] use k-means on the computed MGD values to classify pixels into two types of texts that have a higher cluster mean and non-texts that have a relatively lower cluster mean.

Intensity variations also play a useful role in text detection. If we scan horizontal texts row by row, the scan lines of text regions and background regions will be different in their distributions of intensity. Pixels in text regions manifest that the distribution of their intensities are separated with grayscale values of stroke pixels and between-strokes pixels having relatively high contrast, while those of regions below and above text are uniform in intensities within a specific domain. Thereby, the mean and the deviation of grayscale values of potential text segments can be used to merge neighboring top and bottom text lines to form candidate text regions by performing bottom-up and top-down scanning. Sometimes the intensities of raw pixels that make up the textural pattern can be directly fed to a classifier, e.g., the SVM due to its capability of handling high dimensional data. Kim et al. [16] further set a mask to use the intensities of a subset of the raw pixels as the feature vector and accordingly propose a classification-based algorithm for text detection using a sparse representation with discriminative dictionaries.

Wavelet transform techniques first compute texture features through various descriptors and then a classifier will be used to discriminate text and non-text or further respectively cluster them into text and non-text regions by measuring the Euclidian distances between the texture features. Histograms of wavelet coefficients are popular to capture the texture properties in detecting texts. As known, 1-D wavelet coefficients can effectively identify the sharp signals, while 2-D wavelet transformations can properly capture both signal variations and their orientations. For text detection, wavelet coefficients will be first calculated for all pixels in each video frame. Then for a potential text candidate region, the wavelet histogram is computed by quantizing the coefficients and counting the number of the pixels with their coefficients falling in the quantized bins. Next, the histogram is normalized by the value of each bin representing the percentage of the pixels whose quantized coefficient is equal to the bin. Comparing with the histogram of non-text area, generally, the average values of the histogram in text lines are large. As shown in [17], LH and HL bands are used to calculate the histogram of wavelet coefficients for all the pixels, and the bins at the front and tail of the histogram are found large in both vertical and horizontal bands, while the bins located in the middle parts are relatively small. This is potentially due to the contrast between text and its background.

Rather than histograms, the statistical features of wavelet coefficients are also employed to capture the texture property of text in video frames. Such statistical

features typically include energy, entropy, inertia, local homogeneity, mean, and the second-order (μ_2) and the third-order (μ_3) central moments of wavelet coefficients shown as follows:

$$energy = \sum_{i,j} W^2(i, j) \quad (2.22)$$

$$entropy = \sum_{i,j} W(i, j) \log W(i, j) \quad (2.23)$$

$$inertia = \sum_{i,j} (i - j)^2 W(i, j) \quad (2.24)$$

$$Homogeneity = \sum_{i,j} \frac{1}{1 + (i - j)^2} W(i, j) \quad (2.25)$$

$$mean = \frac{1}{N^2} \sum_{i,j} W(i, j) \quad (2.26)$$

$$\mu_2 = \frac{1}{N^2} \sum_{i,j} (W(i, j) - mean)^2 \quad (2.27)$$

$$\mu_3 = \frac{1}{N^2} \sum_{i,j} (W(i, j) - mean)^3 \quad (2.28)$$

where $w(i, j)$ is a wavelet coefficient at position (i, j) of the window with the size $N \times N$. Shivakumara et al. [18] use 2-D Haar wavelet decomposition to directly detect video texts based on three high-frequency sub-band images of LH, HL, and HH. After computing features for pixels, k-means algorithm is applied to classify the feature vectors into two clusters of background and text candidates. In [19], the entropy, μ_2 , and μ_3 are also used to define the texture feature for text detection. More methods which adopt different wavelets features can be found by referring to [20–22].

Gray-level co-occurrence matrix (GLCM) is another kind of texture feature which is computed over a small square window with size $N \times N$ centered at the pixel (x, y) . GLCM is a tabulation M encoding how often different combinations of pixel values (gray levels) occur in a frame. Specifically, $M(i, j)$ counts the number of two pixels having particular (co-occurring) gray values of i and j . The distance of these two pixels is d along a given direction θ . When divided by the total number of neighboring pixels along (d, θ) a video frame, the matrix becomes the estimate of the joint probability $p_{(d, \theta)}(i, j)$ of the two pixels. Figure 2.9 shows an example of the construction of the gray-level co-occurrence matrix for $d = 1$, $\theta = \{0^\circ, 180^\circ\}$, and $\theta = \{90^\circ, 270^\circ\}$ [52].

Correlation, *entropy*, and *contrast* of the gray-level co-occurrence matrix are further used in [19] to represent image texture. Correlation is used to model the similarity among the rows or the columns in the matrix, entropy essentially indicates the complexity of image texture by measuring the randomness of the matrix, while

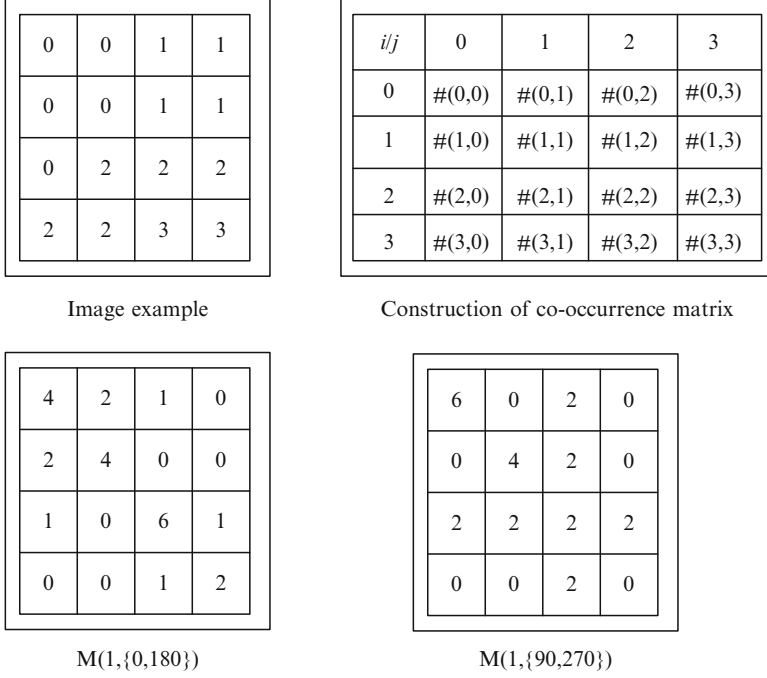


Fig. 2.9 The construction of the co-occurrence matrix [52]

the contrast encodes the variation of pixels in a local region which can be regarded as the intensity of the texture intuitively. The mentioned features are computed as follows:

$$correlation = \frac{\sum_{i=1}^N \sum_{j=1}^N (i \cdot j) P_d(i, j) - \mu_x \mu_y}{\sigma_x \sigma_y} \quad (2.29)$$

$$entropy = -\sum_{i=1}^N \sum_{j=1}^N P_d(i, j) \log P_d(i, j) \quad (2.30)$$

$$contrast = \sum_{n=0}^{n=1} n^2 \sum_{i=1}^N \sum_{j=1}^N P_d(i, j), |i - j| = n \quad (2.31)$$

where $P_d(i, j)$ is the gray-level co-occurrence matrix with the offset d along any direction. μ_x , σ_x , μ_y , and σ_y are the means and standard variances of $P_x(i)$ and $P_y(j)$, respectively. Hanif and Prevost [23] further use marginal probabilities of the joint distribution $p_{(d, \theta)}(i, j)$ to include texture information, shape, and spatial relationship between the pixels to discriminate text and non-text.

Gabor filter is also used to model texture in text detection. A 2-D Gabor filter is essentially a Gaussian kernel modulated by a sinusoidal carrier wave, as defined

below, which gives different responses for different positions in a window centered at (x, y)

$$g(x, y; \rho, \theta, \varphi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + y'^2}{2\sigma^2}\right) \cos\left(2\pi \frac{x'}{\rho} + \varphi\right)$$

$$x' = x \cos \theta + y \sin \theta, y' = -x \sin \theta + y \cos \theta \quad (2.32)$$

Given a proper window, the Gabor filter response is obtained by the convolution with a Gabor filter. Yi and Tian [24] define a suitability measurement to analyze the confidence of Gabor filters in describing stroke components and the suitability of Gabor filters. The k-means algorithm is applied to cluster the descriptive Gabor filters to form clustering centers defined as Stroke Gabor Words (SGWs), which provide a universal description of stroke components. When testing natural scene images, heuristic layout analysis is also applied to search for text candidates.

Qian et al. [25] put forward the DCT coefficients in text detection and localization. The DCT coefficients from an 8×8 image block $f(x, y)$ are expressed as

$$AC_{uv} = \frac{1}{8} C_u C_v \sum_{x=0}^7 \sum_{y=0}^7 f(x, y) \cos \frac{(2x+1)\pi\mu}{16} \times \cos \frac{(2y+1)\pi\nu}{16}$$

$$C_u, C_v = \begin{cases} \frac{1}{\sqrt{2}} & u, v = 0 \\ 1 & \text{others} \end{cases} \quad (2.33)$$

Three horizontal AC coefficients, three vertical AC coefficients, and one diagonal AC coefficient from an 8×8 block are finally selected to capture the horizontal, vertical, and diagonal textures. These AC coefficients are used to represent the texture intensity of the 8×8 block approximately by

$$I(i, j) = \sum_{u=1}^3 |AC_{u0}(i, j)| + \sum_{v=1}^3 |AC_{0v}(i, j)| + |AC_{11}(i, j)| \quad (2.34)$$

Generally, texts have sufficient contrast to the background. Thereby, a region-based smooth filter F is further adopted to distinguish the text blocks from the background:

$$F = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad (2.35)$$

Except for the mentioned usages, the limitations of texture-based transformations for video text preprocessing potentially include its big computational complexity due to the need of scanning of every video frame at several scales and the lack of precision due to the inherent fact that only small (or sufficiently scaled down) text exhibits the properties required by a video text detection algorithm. Additionally, texture-based preprocessing is typically unable to help detect sufficiently slanted texts.

2.4 Image Segmentation

A lot of techniques have been developed to perceive images, which in general consist of three layers of the so-called Image Engineering (IE): processing (low layer), analysis (middle layer), and understanding (high layer), as shown in Fig. 2.10 [26]. As the first step of IE, image segmentation greatly affects the subsequent tasks of video frame analysis including video text detection. Note that in the video case, every video frame is factorized and treated as an individual image in the segmentation step as introduced.

The objective of segmentation is to partition an image into separated regions which ideally correspond to interested objects (e.g., video texts) and accordingly obtain a more meaningful representation of the image. There are some general rules to be followed for the regions resulted from segmentation as [27]:

1. They should be uniform and homogeneous with respect to specific characteristics.
2. Their interiors should be simple enough and without too many small holes.
3. Adjacent regions should have significantly different values with respect to the characteristics on which they are uniform.
4. Boundaries of every segment should be simple, not ragged, and must be spatially accurate.

Theoretically, each image can be segmented into multiple separated regions using two strategies, namely, *bottom-up segmentation* and *top-down segmentation*. For the former, pixels are partitioned into regions if they have similar low-level features, in which image semantics are seldom considered. Super pixels are sometimes generated for the following analysis. For the latter, pixels are partitioned into regions by following strong semantics of the image in the tasks like object recognition and content understanding. We here mainly discuss the bottom-up algorithms which are considered as a step of video preprocessing without any predefined knowledge or semantics.

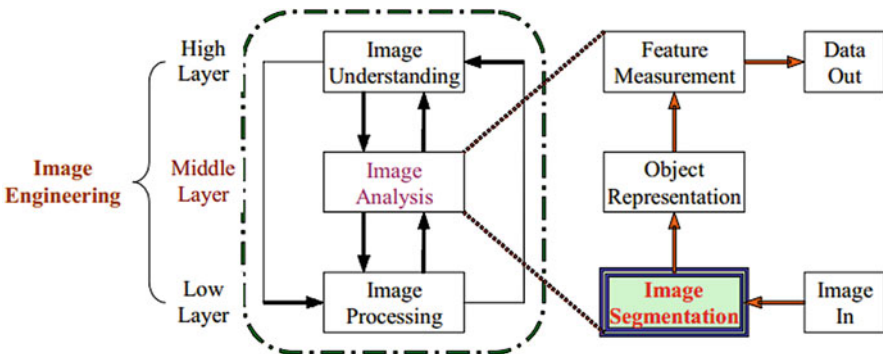


Fig. 2.10 The role of image segmentation [26]

Generally, the simplest way of bottom-up segmentation is using a proper threshold to *transform a grayscale image to a binary one*. The critical issue of this method is how to select such a proper threshold since spatial information is ignored and blurred region boundaries will bring havoc. *Histogram-based methods* are another simple way of bottom-up segmentation since they only require one pass through the pixels and therefore are efficient. In these methods, a histogram is first computed from all the pixels in the image, and then peaks and valleys in the histogram are detected and used to locate the clusters in the image for segmentation [28]. In some cases like for a monochrome image, the value between two histogram peaks can be used as the threshold of binarization, where the background can be defined by the maximum part of the histogram. Note that binarization of complex backgrounds through a single threshold value obtained from the histogram will potentially lead to a loss of object information against the background, e.g., the texts in Fig. 2.11a, b are lost after binarization using Otsu's method [29].

Comparatively, *edge-based methods* make use of edge operators to produce an “edginess” value at each pixel [30] to obtain edges. Then the regions within connected edges are considered as different segments since they lack continuity with other adjacent regions. Edge detection operators are also studied and implemented to find edges in images, and Fig. 2.12 gives some edge detection results using the Sobel operator. Generally, edges are useful to provide aids for other image segmentation algorithms especially in the refinement of segmentation results.

Image segmentation can be formulated as a graph partitioning and optimization problem. It transforms an image into a weighted indirect graph $G = (V, E)$, where the vertex set V represents image points in the feature space and the edge set E

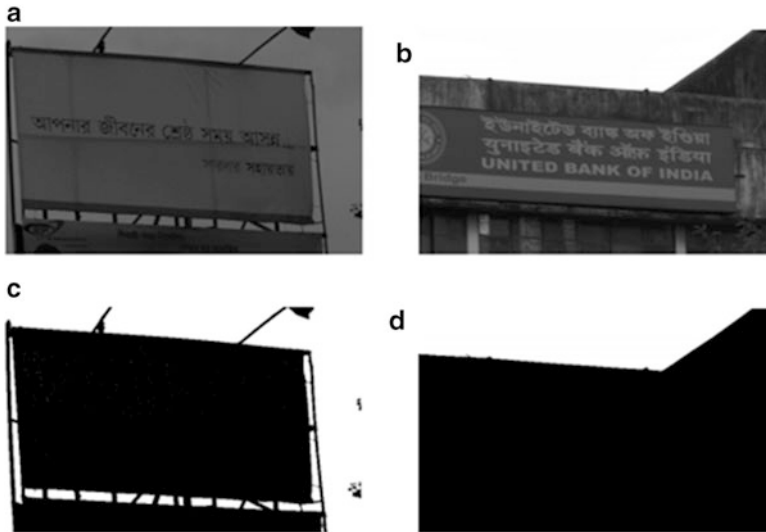


Fig. 2.11 Image binarization by a histogram: (a) and (b) are two scene image examples, (c) and (d) respectively illustrate their binarization results using Otsu's method [29]



Fig. 2.12 Edge detection examples with the Canny operator on the second row and Sobel operator on the last row

represents point connectivity. There in general exists a weight $w(i, j)$ for each edge to measure the similarity of vertex i and vertex j . Then the vertices are expected to be partitioned into disjoint sets of $V_1, V_2 \dots V_k$, where the similarity between the vertices in the same set V_i is relatively high by a specific measure, while that between the vertices across different sets V_i and V_j will be relatively low.

To partition such a graph in a meaningful way, a proper criterion should be selected to solve the optimization problem effectively and efficiently. A graph can be partitioned into two disjoint sets by using the edge cut technique which removes edges connecting two different parts. The degree of dissimilarity between these two pieces can be computed as the total weight of the edges that have been removed. In the *minimum cut* criteria, the quality of a bipartition can be measured by the following equation where a smaller value indicates a better partition:

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, v) \quad (2.36)$$

However, the minimum cut criteria still has a drawback of obtaining small sets of isolated nodes in the graph. The *normalized cut* criteria [34] make an improvement against it and introduce the size leverage of partition vertices by the following formulation:

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)} \quad (2.37)$$

$$assoc(A, V) = \sum_{u \in A, v \in V} W(u, v) \quad (2.38)$$

Against the former cut criteria, the normalized cut considers association within a group, and isolated points will no longer have a small cut value to avoid bias partitions. Formally, let x be a $|V|$ dimension indicating vector with $x_i = 1$ if v_i is partitioned into segment A , and -1 otherwise, W is a $|V| \times |V|$ matrix with $W_{ij} = -w(i,j)$ representing the distance between vertex i and j , and D is a $|V| \times |V|$ diagonal matrix with $d_i = \sum_j W(i,j)$. Then let

$$k = \frac{\sum_{x_i > 0} d_i}{\sum_i d_i}, \quad b = \frac{k}{1-k}, \quad y = (1+x) - b(1-x) \quad (2.39)$$

Finally, the image segmentation problem based on *normalized cut* criteria can be formulized as an optimization problem [34]:

$$\begin{aligned} \min_x Ncut(x) &= \min_y \frac{y^T (D-W)y}{y^T D y} \\ s.t. \quad &y^T D 1 = 0 \end{aligned} \quad (2.40)$$

which can be solved according to the theory of *generalized eigenvalue system*. Although there exists a major stumbling block that an exact solution to minimize the normalized cut is an NP-complete problem, approximate discrete solutions x can be achieved efficiently from y . Some examples of normalized cut are shown in Fig. 2.13.

Graph-based segmentation [33] is an efficient approach by performing clustering in the feature space. Such a method works directly on the data points in the feature

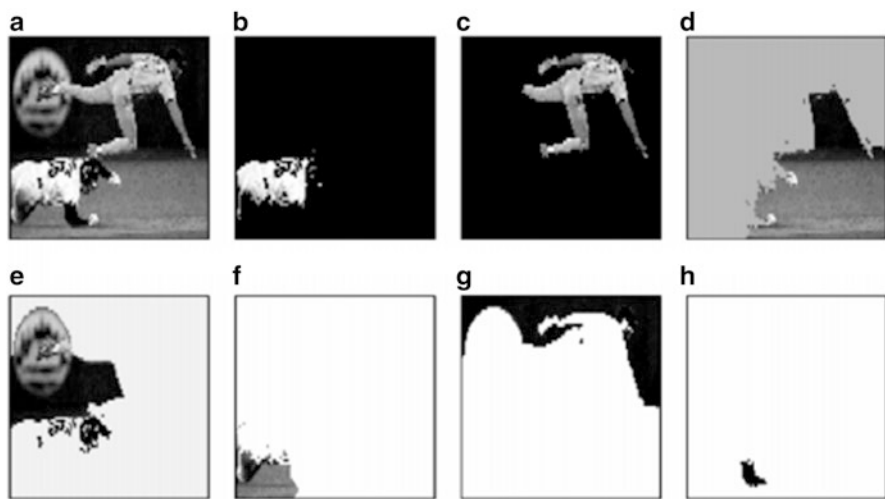


Fig. 2.13 Examples of normalized cut for image segmentation. (a) original image, (b–h) the segmented regions [34]

space, without a filtering processing step. The key to the success of these methods is using an adaptive threshold scheme. Formally, let $G = (V, E)$ represent the image and a weighting system w_E associating with edges. The final segmentation results $S = \{V_1, V_2, \dots, V_r\}$ can be achieved from the following algorithm:

1. Sort $E = (e_1, e_2, \dots, e_{|E|})$ according to $w_E = (w_1, w_2, \dots, w_{|E|})$.
2. Initial $S^0 = \{\{v_1\}, \{v_2\}, \dots, \{v_{|V|}\}\}$, where each cluster contains exactly one vertex.
3. For $t = 1, 2, \dots, |E|$
 - (a) $e_t = (v_i, v_j)$. $V^{t-1}(v_i)$ is the cluster containing v_i on the iteration $t-1$, and $l_i = \max \text{mst}(V^{t-1}(v_i))$ is the longest edge in the minimum spanning tree of $V^{t-1}(v_i)$. Likewise for l_j .
 - (b) Merge $V^{t-1}(v_i)$ and $V^{t-1}(v_j)$ if

$$W_{e_t} < \min \left\{ l_i + \frac{k}{|V^{t-1}(v_i)|}, l_j + \frac{k}{|V^{t-1}(v_j)|} \right\} \quad (2.41)$$

where k is a predefined empirical parameter.

4. $S = S^{|E|}$.

Specifically, during the clustering procedure, a minimum spanning tree of the data points is first generated, from which any length of edges within a data-dependent threshold indicates a merging process, in contrast to the mean shift algorithms which use a fixed threshold. The connected components are merged into clusters for segmentation. The merging criteria allows efficient graph-based clustering to be sensitive to edges in regions of low variability and less sensitive to them in regions of high variability, as shown in Fig. 2.14 [32]. On the other hand, a graph-based segmentation algorithm can well generate image regions for further processing.

The mean shift-based method introduced in [31] has also been widely used in image segmentation recently. It is one of the techniques under the heading of “feature space analysis.” The mean shift method comprises two basic steps: a mean shift filtering of the original image data and a subsequent clustering of the filtered data points. Specifically, the filtering step consists of analyzing the probability

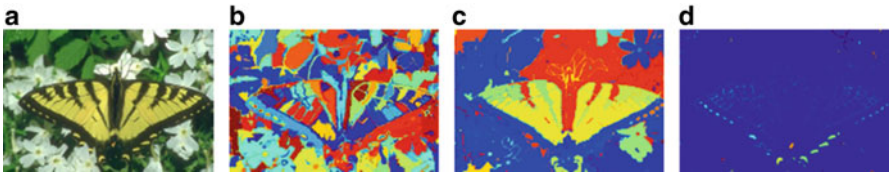


Fig. 2.14 Examples of efficient graph-based image segmentation: (a) original image, (b–d) the results of different granularities. Note that from fine to coarse, no granularity can generate a complete butterfly [32]

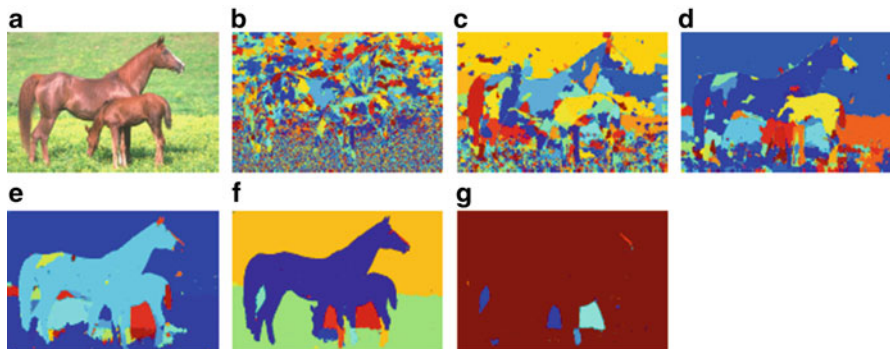


Fig. 2.15 Examples of mean shift segmentation: (a) original image, (b–g) the results with different granularities [31]

density function (pdf) of the image data in the feature space, then finding the modes of the underlying pdf in a sphere or window and associating with them any points in their basin of attraction. After mean shift filtering, every data point in the feature space will be replaced by its corresponding mode. Then clustering is described as a simple postprocessing step, in which the modes that are less than a specific kernel radius apart are grouped together and their basins of attraction are accordingly merged. This suggests using single linkage clustering, which effectively converts the filtered points into segmentation. Actually, the quantity of the sphere or windows decides the granularities during segmentation and Fig. 2.15 illustrates some examples. It can be found that a mean shift algorithm is quite sensitive to the mentioned window parameters, and a slight variation of the parameters will cause a large change in the granularity of the segmentation, as shown in Fig. 2.15b–g, which changes from over-segmentation to under-segmentation. This issue is a major stumbling block with respect to using mean shift segmentation as a reliable preprocessing step for other algorithms [32].

2.5 Motion Analysis

Motion analysis has been investigated in many video applications especially for object tracking or object recognition. It studies methods in which two or more video frames from an image sequence are processed. As a preprocessing technique, motion vector analysis from temporal frames is believed helpful for video text detection. It produces time-dependent information about the motions of both the foreground objects and their backgrounds to provide hints for identifying texts from consecutive video frames. Moreover, it potentially plays a key role in detecting and tracking video texts from multiple directions or with occlusions. For example, Palma et al. [35] present an automatic text extraction solution for digital video

based on motion analysis. In their motion analysis module, the correlation between frames is exploited to improve the results of video text detection. The major goals of motion analysis are to refine the text detection in terms of removing false alarms in individual frames, interpolating the location of accidentally missed text characters and words in individual frames, and temporally localizing the beginning and end of each word as well as its spatial location within each frame. Their method proposed for motion analysis is based on the comparison of regions in successive frames and includes five typical steps:

1. **Text Tracking:** This step is responsible for tracking the already detected text along the frames that constitute each text sequence targeting the formation of temporally related chains of characters. Each character chain here represents the same character during its existence in the video sequence and consists in a collection of similar regions, occurring in several contiguous frames. Every time a character region is detected for the first time, a position is stored and a signature is computed for that character region by using the features of luminance, size, and shape. Each frame contributes with one region classified as a character for the construction of a character chain.
2. **Text Integration:** This step groups the character chains in order to form words based on the spatial and temporal analysis of each character chain. Three temporal elements are adopted: (a) temporary coexistence, (b) duration, and (c) motion. The chains not included in words at this phase are considered as noise and are discarded.
3. **Character Recovery:** Video temporal redundancy is explored to complete the words with missing characters as least for some frames, e.g., due to noise or too textured background. In order to complete the words, they are extended to the size of their biggest chain of characters, and the characters missing in the chains are recovered by means of temporal interpolation with motion compensation. Thus, by using temporal redundancy, the text detection for each frame is improved by completing the words with missing characters for some frames.
4. **Elimination of Overlapped Words:** It is important to improve the performance for scene text. Usually, overlapping of words occurs when false words are detected, e.g., due to shadows or three-dimensional text. Every time two or more words overlap, more precisely their bounding boxes overlap at least for one frame; the words with the smaller areas are discarded.
5. **Text Rotation:** This step performs the rotation of the text to the horizontal position in order to be recognized OCR systems.

Figure 2.16 shows the benefit of motion analysis for the extraction of text from video sequences. The result of extracting text from a single frame is shown in Fig. 2.16b, while Fig. 2.16c shows the result of extracting text for the same frame but exploiting the temporal redundancy by motion analysis in the context of a video sequence. In [47], motion analysis is adopted to capture continuous changes between adjacent video frames, and then the dynamic attention algorithm is used to detect video caption blocks to search for video text candidates.



Fig. 2.16 Examples of the benefits of motion analysis for video text extraction: (a) original video frame, (b) text detection from a single video frame, and (c) better text detection for the same frame but using temporal redundancy for recovering lost characters [35]

Optical flow is the most common technique used in motion analysis [36–40] by exploiting two constraints from video frames: brightness constancy and spatial smoothness. The brightness constancy constraint is derived from the observation that surfaces usually persist over time, and hence, the intensity value of a small region in video remains the same despite its position changes [41, 42]. Using the brightness constancy, gradient methods can be used to analyze the derivatives of video intensity values to compute optical flows. As a supplement, the spatial smoothness constraint comes from the observation that neighboring pixels generally belong to the same surface and thus have nearly the same image motion characteristic.

Regardless of the difference between the two main types of methods, the majority are based on the following three stages in computing optical flows:

1. Preliminary processing using appropriate filters in order to obtain the desired signal structure and improve the signal-to-noise ratio
2. Calculating basic measurements such as partial derivatives after a certain time period or local correlation areas
3. Integrating the measurements in order to calculate a two-dimensional optical flow

As a representative approach, the Horn-Schunck (HS) method uses global constraint of smoothness to express a brightness variation in certain areas of the frames in a video sequence. HS is a specially defined framework to lay out the smoothness of the flow field, which can be described as follows [46]:

1. Calculate optical flows between two adjacent frames (after registration as needed).
2. For each pixel in the 2-D optical flow data, perform PCA for a local mask, and two eigenvalues are assigned to the central pixel.
3. Apply the Otsu's thresholding to the eigenvalues of all the pixels.

Let $I(x,y,t)$ denote the brightness of a pixel at position (x, y) and the t th frame; the image constraint at $I(x,y,t)$ with Taylor series can be expressed by

$$\frac{\partial I}{\partial x} \partial x + \frac{\partial I}{\partial y} \partial y + \frac{\partial I}{\partial t} \partial t = 0 \quad (2.42)$$

which results in

$$I_x u + I_y v + I_t = 0 \quad (2.43)$$

where $u = \partial x / \partial t$ and $v = \partial y / \partial t$ are the x and y components of the velocity or optical flow of $I(x, y, t)$, respectively. $I_x = \partial I / \partial x$, $I_y = \partial I / \partial y$, and $I_t = \partial I / \partial t$ are the derivatives of the image at (x, y, t) in the corresponding directions. A constrained minimization problem can then be formulated to calculate optical flow vector (u^{k+1}, v^{k+1}) for the $(k+1)$ th frame by

$$u^{k+1} - \bar{u}^k = I_x \cdot \frac{I_x \bar{u}^k + I_y \bar{v}^k + I_t}{\alpha^2 + I_x^2 + I_y^2} \quad (2.44)$$

$$v^{k+1} - \bar{v}^k = I_y \cdot \frac{I_x \bar{u}^k + I_y \bar{v}^k + I_t}{\alpha^2 + I_x^2 + I_y^2} \quad (2.45)$$

where \bar{u}^k and \bar{v}^k are the estimated local average optical flow velocities and α is a weighting factor. Based on the norm of an optical flow, one can determine if the motion exists or not, while the direction of this vector provides the motion orientation.

Next, two optical flow images can be constructed by pixel optical flow vector (u, v) . A mask of size $n \times n$ slides through these u and v images. At location (i, j) , a two-dimensional data matrix \mathbf{X} can be constructed, which includes all the 2-D vectors covered by the mask. The covariance matrix is calculated as

$$\Sigma = \bar{\mathbf{X}}^T \bar{\mathbf{X}} \quad (2.46)$$

where $\bar{\mathbf{X}}$ is the optical flow matrix after mean removal. After eigendecomposition, two eigenvalues (λ_1, λ_2) are assigned to the central pixel of the mask and motion detection will be accomplished by analyzing or thresholding the eigenvalues. Since λ_1 is the major flow component and λ_2 is the minor flow component, it may be more efficient to consider (λ_1, λ_2) than the values in the original (u, v) space. Note that although only λ_1 is intuitively needed to be considered because it corresponds to the major flow component while λ_2 corresponds to the minor flow component or even turbulence, in practice λ_2 is often considered as well since pixels inside object boundaries usually have quite large λ_2 but not λ_1 . Thus, thresholding may need to be taken on the λ_2 histogram determined by using the Ostu's method [29], and a pixel is claimed to have motion if either λ_1 or λ_2 is above the corresponding thresholds.

Figure 2.17a shows the framework of the HS algorithm with a 3×3 mask and resulting 2×9 data matrices [44]. In Fig. 2.17b, two video frames are input, and the result after using the HS optical flow method is shown in Fig. 2.17c. The final result is given in Fig. 2.17d after Kalman filtering.

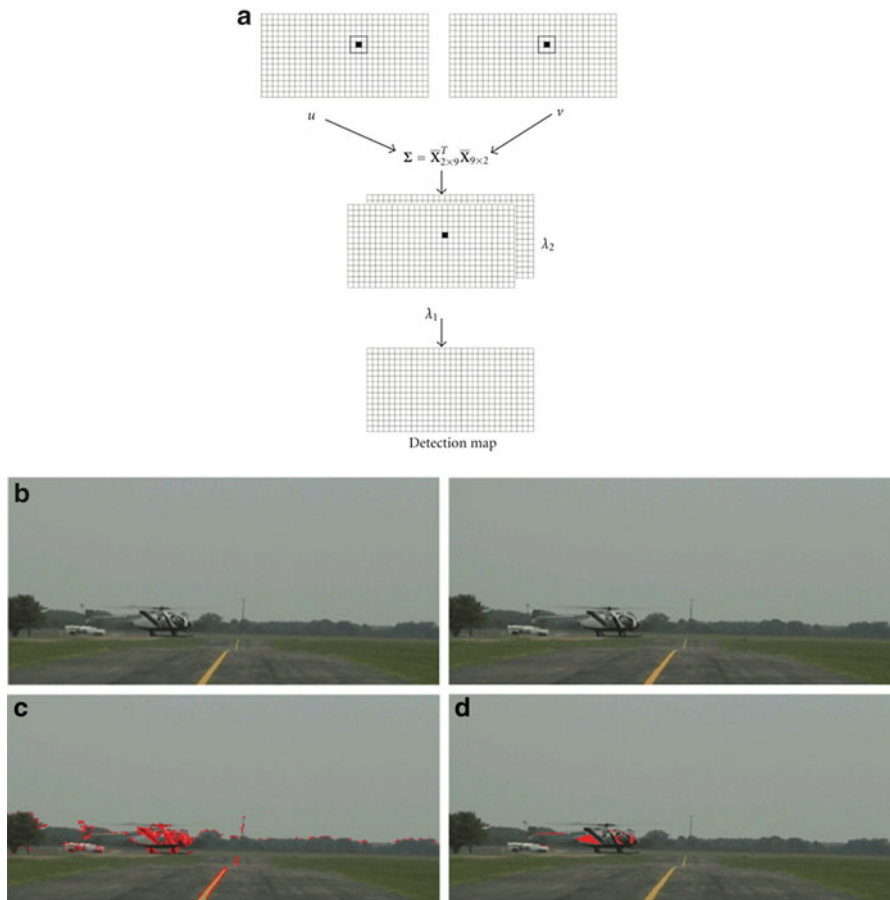


Fig. 2.17 The framework of the HS algorithm. (a) The framework of optical flow computation, (b) two input frames from video, (c) the result after using the HS optical flow method, and (d) the final result after Kalman filtering [44]

There are two other classic optical flow algorithms, namely, the Lucas-Kanade method [45] and the Brox method [36]. The Lucas-Kanade method assumes that the flow is essentially constant in a local neighborhood of the pixel. Thus, the goal of the Lucas-Kanade method is to minimize the sum of squared error between two video frames. Since the pixel values are essentially unrelated to its coordinates, the Lucas-Kanade algorithm assumes that a current estimate of vector p is known and then iteratively solves for increments to the parameters Δp for optimization. The Brox method relies on the brightness and gradient constancy assumptions, using the information of video frame intensities and the gradients to find correspondences. With respect to the Horn-Schunck method, it is more robust to the presence of outliers and can cope with constant brightness changes. Głowacz et al. [43] compare

Table 2.1 Optical flow computation: results of preliminary experiments [43]

Method	Image size	Mean calculation time for a single frame (s)	Range (for frames with no vibration)	Implementation language
Horn-Schunck $\alpha = 31, iter = 8$	640×480	0.09	[0;2.67]	C
Horn-Schunck $\alpha = 33, iter = 9$	640×480	0.10	[0;3.36]	C
Lucas-Kanade $\tau = 32$	640×480	0.09	[0;6.64]	C
Brox	640×480	36.00	–	MATLAB
	320×240	7.70	[0;0.88]	MATLAB

the Horn-Schunck method with these two methods in computing motions from video. They find that taking into account the effectiveness and calculation time the Horn-Schunck algorithm is effective at detecting video objects when they are subject to binarization using a fixed threshold. Table 2.1 presents basic information on the tests conducted. It can be found that Horn-Schunck and Lucas-Kanade have similar mean calculation time costs for a single frame. Figure 2.18 gives the results by respectively using the three optical flow computation methods.

2.6 Summary

This chapter gives a brief overview of preprocessing techniques related to video text detection. We introduce image preprocessing operators and color-based and texture-based preprocessing techniques which are most frequently used in video analysis. Since image segmentation potentially helps reduce the searching space and more important, provides contextual hints in complex video scenes for video graphics text and scene text detection, we then give a brief introduction on automatic segmentation technique. Finally, considering the necessity of temporal redundancy analysis in detecting texts from video sequences, we introduce how to compute motions from video frames and how video text detection benefits from motionanalysis.

Most of the introduced preprocessing operators and methods have been realized by MATLAB or OpenCV (Open Source Computer Vision Library), so readers can make use of these open sources for practice. A more systematical and detailed introduction to the discussed techniques may be found in the references for image processing and machine vision in [48–51].

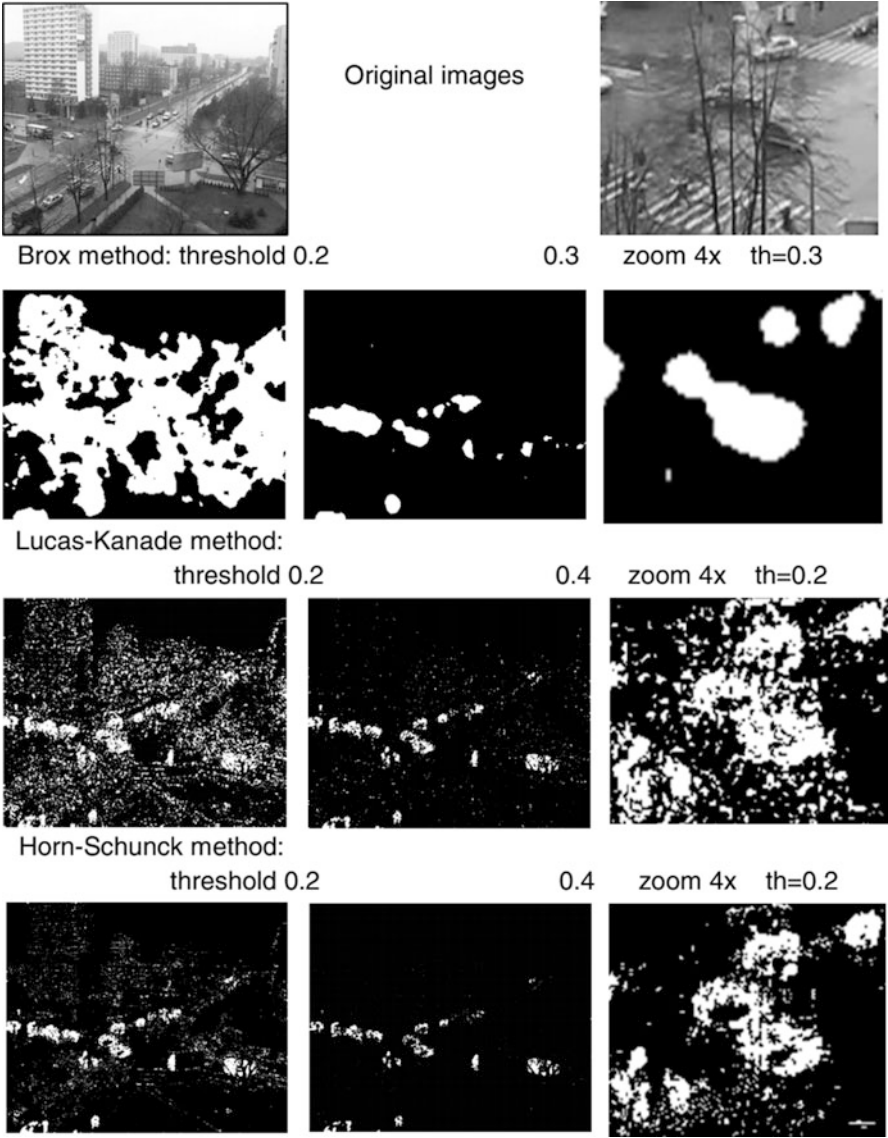


Fig. 2.18 Effects of the optical flow binarization calculated using three methods [43]

References

1. Jung K, In Kim K, Jain AK (2004) Text information extraction in images and video: a survey. *Pattern Recog* 37(5):977–997
2. Lienhart RW, Stuber F (1996) Automatic text recognition in digital videos. *Proc SPIE* 2666(3):180–188
3. Crane R (1996) Simplified approach to image processing: classical and modern techniques in C. Prentice Hall PTR. 317
4. Szeliski R (2010) Computer vision: algorithms and applications. Springer, New York
5. Kopf J et al (2007) Capturing and viewing gigapixel images. *ACM Trans Graph* 26(3):93
6. Roberts LG (1963) Machine perception of three-dimensional solids, DTIC Document
7. Engel K et al (2006) Real-time volume graphics: AK Peters, Limited
8. Ritter GX, Wilson JN (1996) Handbook of computer vision algorithms in image algebra, vol 1. Citeseer
9. Hasan YMY, Karam LJ (2000) Morphological text extraction from images. *IEEE Trans Image Process* 9(11):1978–1983
10. Jae-Chang S, Dorai C, Bolle R (1998) Automatic text extraction from video for content-based annotation and retrieval. In: *Proceedings of the fourteenth international conference on pattern recognition*, 1998
11. Kim H-K (1996) Efficient automatic text location method and content-based indexing and structuring of video database. *J Vis Commun Image Represent* 7(4):336–344
12. Jain AK, Yu BIN (1998) Automatic text location in images and video frames. *Pattern Recogn* 31(12):2055–2076
13. Zhong Y, Karu K, Jain AK (1995) Locating text in complex color images. *Pattern Recogn* 28(10):1523–1535
14. Wong EK, Chen M (2003) A new robust algorithm for video text extraction. *Pattern Recogn* 36(6):1397–1406
15. Shivakumara P, Trung Quy P, Tan CL (2011) A laplacian approach to multi-oriented text detection in video. *IEEE Trans Pattern Anal Mach Intell* 33(2):412–419
16. Kim KI, Jung K, Kim JH (2003) Texture-based approach for text detection in images using support vector machines and continuously adaptive mean shift algorithm. *IEEE Trans Pattern Anal Mach Intell* 25(12):1631–1639
17. Ye Q et al (2007) Text detection and restoration in natural scene images. *J Vis Commun Image Represent* 18(6):504–513
18. Shivakumara P, Trung Quy P, Tan CL (2009) A robust wavelet transform based technique for video text detection. In: *ICDAR '09. 10th international conference on document analysis and recognition*, 2009
19. Zhong J, Jian W, Yu-Ting S (2009) Text detection in video frames using hybrid features. In: *International conference on machine learning and cybernetics*, 2009
20. Zhao M, Li S, Kwok J (2010) Text detection in images using sparse representation with discriminative dictionaries. *Image Vis Comput* 28(12):1590–1599
21. Shivakumara P, Trung Quy P, Tan CL (2010) New Fourier-statistical features in RGB space for video text detection. *Circ Syst Video Technol IEEE Trans* 20(11):1520–1532
22. Rongrong J et al (2008) Directional correlation analysis of local Haar binary pattern for text detection. In: *IEEE international conference on multimedia and expo*, 2008
23. Hanif SM, Prevost L (2007) Text detection in natural scene images using spatial histograms. In: *2nd workshop on camera based document analysis and recognition*, Curitiba
24. Chucai Y, YingLi T (2011) Text detection in natural scene images by Stroke Gabor Words. In: *International conference on document analysis and recognition (ICDAR)*, 2011
25. Qian X et al (2007) Text detection, localization, and tracking in compressed video. *Signal Process Image Commun* 22(9):752–768
26. ZHANG YJ (2002) Image engineering and related publications. *Int J Image Graph* 02(03):441–452

27. Haralick RM, Shapiro LG (1985) Image segmentation techniques. *Comp Vis Graph Image Process* 29(1):100–132
28. Shapiro LG, Stockman GC (2001) *Computer vision*. Prentice-Hall, New Jersey, pp 279–325
29. Otsu N (1975) A threshold selection method from gray-level histograms. *Automatica* 11(285–296):23–27
30. Saraf Y (2006) *Algorithms for image segmentation*, Birla Institute of Technology and Science
31. Comaniciu D, Meer P (2002) Mean shift: a robust approach toward feature space analysis. *Pattern Anal Mach Intell IEEE Trans on* 24(5):603–619
32. Pantofaru C, Hebert M (2005) A comparison of image segmentation algorithms. *Robotics Institute*, p 336
33. Felzenszwalb P, Huttenlocher D (2004) Efficient graph-based image segmentation. *Int J Comp Vis* 59(2):167–181
34. Shi J, Malik J (2000) Normalized cuts and image segmentation. *Pattern Anal Mach Intell IEEE Trans* 22(8):888–905
35. Palma D, Ascenso J, Pereira F (2004) Automatic text extraction in digital video based on motion analysis. In: Campilho A, Kamel M (eds) *Image analysis and recognition*. Springer, Berlin, pp 588–596
36. Brox T et al (2004) High accuracy optical flow estimation based on a theory for warping. In: Pajdla T, Matas J (eds) *Computer vision – ECCV 2004*. Springer, Berlin, pp 25–36
37. Beauchemin SS, Barron JL (1995) The computation of optical flow. *ACM Comput Surv* 27(3):433–466
38. Anandan P (1989) A computational framework and an algorithm for the measurement of visual motion. *Int J Comp Vis* 2(3):283–310
39. Weickert J, Schnörr C (2001) A theoretical framework for convex regularizers in PDE-based computation of image motion. *Int J Comp Vis* 45(3):245–264
40. Mémin E, Pérez P (2002) Hierarchical estimation and segmentation of dense motion fields. *Int J Comp Vis* 46(2):129–155
41. Sun D et al (2008) Learning optical flow. In: Forsyth D, Torr P, Zisserman A (eds) *Computer vision – ECCV 2008*. Springer, Berlin, pp 83–97
42. Black MJ, Anandan P (1996) The robust estimation of multiple motions: parametric and piecewise-smooth flow fields. *Comp Vis Image Underst* 63(1):75–104
43. Głowacz A, Mikrut Z, Pawlik P (2012) Video detection algorithm using an optical flow calculation method. In: Dziech A, Czyżewski A (eds) *Multimedia communications, services and security*. Springer, Berlin, pp 118–129
44. Horn BKP, Schunck BG (1981) Determining optical flow. *Artif Intell* 17(1–3):185–203
45. Lucas BD, Kanade T (1981) An iterative image registration technique with an application to stereo vision. In: *IJCAI*
46. Kui L et al (2010) Optical flow and principal component analysis-based motion detection in outdoor videos. *EURASIP J Adv Signal Proc*
47. Zhao Y et al (2011) Real-time video caption detection. In: *Proceedings of the ninth IAPR international workshop on graphics recognition*
48. Gonzalez RC, Woods RE, Eddins SL *Digital image processing using matlab*. Prentice Hall
49. Nixon MS, Aguado AS *Feature extraction & image processing for computer vision*, 3rd edn. Academic
50. Forsyth DA, Ponce J *Computer vision: a modern approach*, 2nd edn. Prentice Hall Press
51. Petrou M, Petrou C *Image processing: the fundamentals*. Wiley
52. Haralick RM, Shanmugam K (1973) Its’ Hak Dinstein. Textual features for image classification. *IEEE Trans Syst Man Cybern* 3(6):610–621

Video Text Detection

Lu, T.; Palaiahnakote, S.; Tan, C.L.; Liu, W.

2014, X, 264 p. 160 illus., Hardcover

ISBN: 978-1-4471-6514-9