11

Automatic Thresholding

Although techniques based on binary image regions have been used for a very long time, they still play a major role in many practical image processing applications today because of their simplicity and efficiency. To obtain a binary image, the first and perhaps most critical step is to convert the initial grayscale (or color) image to a binary image, in most cases by performing some form of thresholding operation, as described in Chapter 4, Sec. 4.1.4.

Anyone who has ever tried to convert a scanned document image to a readable binary image has experienced how sensitively the result depends on the proper choice of the threshold value. This chapter deals with finding the best threshold automatically only from the information contained in the image, i.e., in an "unsupervised" fashion. This may be a single, "global" threshold that is applied to the whole image or different thresholds for different parts of the image. In the latter case we talk about "adaptive" thresholding, which is particularly useful when the image exhibits a varying background due to uneven lighting, exposure, or viewing conditions.

Automatic thresholding is a traditional and still very active area of research that had its peak in the 1980s and 1990s. Numerous techniques have been developed for this task, ranging from simple ad-hoc solutions to complex algorithms with firm theoretical foundations, as documented in several reviews and evaluation studies [86, 178, 204, 213, 231]. Binarization of images is also considered a "segmentation" technique and thus often categorized under this term. In the following, we describe some representative and popular techniques in greater detail, starting in Sec. 11.1 with global thresholding methods and continuing with adaptive methods in Sec. 11.2.

11.1 Global Histogram-Based Thresholding

Given a grayscale image I, the task is to find a single "optimal" threshold value for binarizing this image. Applying a particular threshold q is equivalent to classifying each pixel as being either part

W. Burger, M.J. Burge, *Digital Image Processing*, Texts in Computer Science,
DOI 10.1007/978-1-4471-6684-9_11

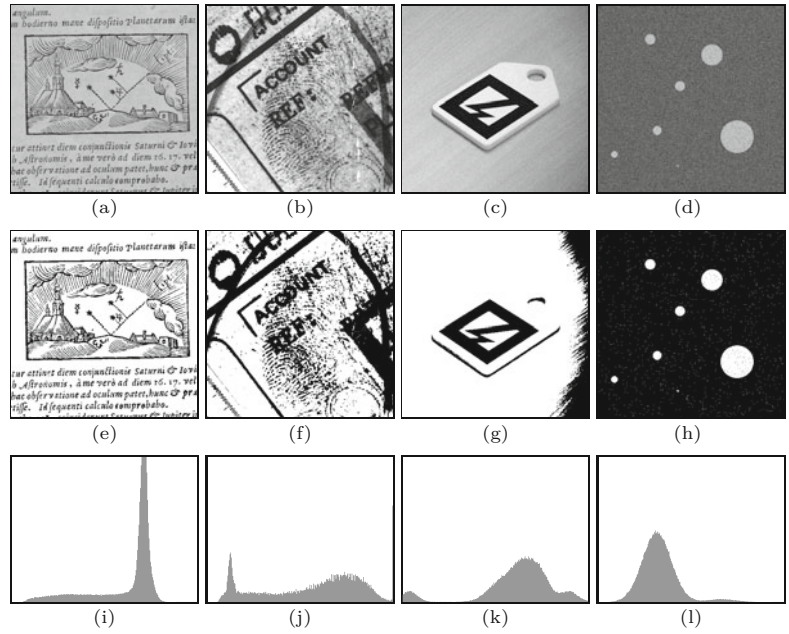


Fig. 11.1
Test images used for subsequent thresholding experiments. Detail from a manuscript by Johannes Kepler (a), document with fingerprint (b), ARToolkit marker (c), synthetic two-level Gaussian mixture image (d). Results of thresholding with the fixed threshold value $q = 128$ (e–h). Histograms of the original images (i–l) with intensity values from 0 (left) to 255 (right).

of the *background* or the *foreground*. Thus the set of all image pixels is partitioned into two disjoint sets $\mathcal{C}_0$ and $\mathcal{C}_1$, where $\mathcal{C}_0$ contains all elements with values in $[0, 1, \ldots, q]$ and $\mathcal{C}_1$ collects the remaining elements with values in $[q+1, \ldots, K-1]$, that is,

$$(u, v) \in \begin{cases} \mathcal{C}_0 & \text{if } I(u, v) \leq q \text{ (background)}, \\ \mathcal{C}_1 & \text{if } I(u, v) > q \text{ (foreground)}. \end{cases} \tag{11.1}$$

Of course, the meaning of *background* and *foreground* may differ from one application to another. For example, the aforementioned scheme is quite natural for astronomical or thermal images, where the relevant "foreground" pixels are bright and the background is dark. Conversely, in document analysis, for example, the objects of interest are usually the *dark* letters or artwork printed on a bright background. This should not be confusing and of course one can always *invert* the image to adapt to this scheme, so there is no loss of generality here.

Figure 11.1 shows several test images used in this chapter and the result of thresholding with a fixed threshold value. The synthetic image in Fig. 11.1(d) is the mixture of two Gaussian random distributions $\mathcal{N}_0, \mathcal{N}_1$ for the background and foreground, respectively, with $\mu_0 = 80$, $\mu_1 = 170$, $\sigma_0 = \sigma_1 = 20$. The corresponding histograms of the test images are shown in Fig. 11.1(i–l). Note that all histograms are normalized to constant area (not to maximum values, as usual), with intensity values ranging from 0 (left) to 255 (right).

The key question is how to find a suitable (or even "optimal") threshold value for binarizing the image. As the name implies, histogram-based methods calculate the threshold primarily from the information contained in the image's histogram, without inspecting the actual image pixels. Other methods process individual pixels for finding the threshold and there are also hybrid methods that rely both on the histogram and the local image content. Histogram-based

techniques are usually simple and efficient, because they operate on a small set of data (256 values in case of an 8-bit histogram); they can be grouped into two main categories: *shape-based* and *statistical* methods.

Shape-based methods analyze the structure of the histogram's distribution, for example by trying to locate peaks, valleys and other "shape" features. Usually the histogram is first smoothed to eliminate narrow peaks and gaps. While shape-based methods were quite popular early on, they are usually not as robust as their statistical counterparts or at least do not seem to offer any distinct advantages. A classic representative of this category is the "triangle" (or "chord") algorithm described in [261]. References to numerous other shape-based methods can be found in [213].

Statistical methods, as their name suggests, rely on statistical information derived from the image's histogram (which of course is a statistic itself), such as the mean, variance, or entropy. In the next section, we discuss a few elementary parameters that can be obtained from the histogram, followed by a description of concrete algorithms that use this information. Again there are a vast number of similar methods and we have selected four representative algorithms to be described in more detail: (a) iterative threshold selection by Ridler and Calvard [198], (b) Otsu's clustering method [177], (c) the minimum error method by Kittler and Illingworth [116], and (d) the maximum entropy thresholding method by Kapur, Sahoo, and Wong [133].

11.1.1 Image Statistics from the Histogram

As described in Chapter 3, Sec. 3.7, several statistical quantities, such as the arithmetic mean, variance and median, can be calculated directly from the histogram, without reverting to the original image data. If we *threshold* the image at level q $(0 \le q < K)$, the set of pixels is partitioned into the disjoint subsets $\mathcal{C}_0$, $\mathcal{C}_1$, corresponding to the background and the foreground. The number of pixels assigned to each subset is

$$n_0(q) = |\mathcal{C}_0| = \sum_{g=0}^{q} \mathsf{h}(g) \quad \text{and} \quad n_1(q) = |\mathcal{C}_1| = \sum_{g=q+1}^{K-1} \mathsf{h}(g), \tag{11.2}$$

respectively. Also, because all pixels are assigned to either the *background* set $\mathcal{C}_0$ or the *foreground* set $\mathcal{C}_1$,

$$n_0(q) + n_1(q) = |\mathcal{C}_0| + |\mathcal{C}_1| = |\mathcal{C}_0 \cup \mathcal{C}_1| = MN. \tag{11.3}$$

For any threshold q, the *mean* values of the associated partitions $\mathcal{C}_0, \mathcal{C}_1$ can be calculated from the image histogram as

$$\mu_0(q) = \frac{1}{n_0(q)} \cdot \sum_{g=0}^{q} g \cdot \mathsf{h}(g), \tag{11.4}$$

$$\mu_1(q) = \frac{1}{n_1(q)} \cdot \sum_{g=q+1}^{K-1} g \cdot \mathsf{h}(g) \tag{11.5}$$

and these quantities relate to the image's overall mean μ_I (Eqn. (3.9)) by[1]

$$\mu_I = \frac{1}{MN} \cdot \big[n_0(q) \cdot \mu_0(q) + n_1(q) \cdot \mu_1(q)\big] = \mu_0(K-1). \tag{11.6}$$

Analogously, the *variances* of the background and foreground partitions can be extracted from the histogram as[2]

$$\begin{aligned} \sigma_0^2(q) &= \frac{1}{n_0(q)} \cdot \sum_{g=0}^{q} (g - \mu_0(q))^2 \cdot \mathsf{h}(g) \\ \sigma_1^2(q) &= \frac{1}{n_1(q)} \cdot \sum_{g=q+1}^{K-1} (g - \mu_1(q))^2 \cdot \mathsf{h}(g). \end{aligned} \tag{11.7}$$

(Of course, as in Eqn. (3.12), this calculation can also be performed in a single iteration and without knowing $\mu_0(q), \mu_1(q)$ in advance.) The overall variance σ_I^2 for the whole image is identical to the variance of the background for $q = K-1$,

$$\sigma_I^2 = \frac{1}{MN} \cdot \sum_{g=0}^{K-1} (g - \mu_I)^2 \cdot \mathsf{h}(g) = \sigma_0^2(K-1), \tag{11.8}$$

that is, for all pixels being assigned to the background partition. Note that, unlike the simple relation of the means given in Eqn. (11.6),

$$\sigma_I^2 \neq \frac{1}{MN}\big[n_0(q) \cdot \sigma_0^2(q) + n_1(q) \cdot \sigma_1^2(q)\big] \tag{11.9}$$

in general (see also Eqn. (11.20)).

We will use these basic relations in the discussion of histogram-based threshold selection algorithms in the following and add more specific ones as we go along.

11.1.2 Simple Threshold Selection

Clearly, the choice of the threshold value should not be fixed but somehow based on the content of the image. In the simplest case, we could use the *mean* of all image pixels,

$$q \leftarrow \text{mean}(I) = \mu_I, \tag{11.10}$$

as the threshold value q, or the *median*, (see Sec. 3.7.2),

$$q \leftarrow \text{median}(I) = m_I, \tag{11.11}$$

or, alternatively, the average of the *minimum* and the *maximum* (mid-range value), that is,

$$q \leftarrow \frac{\max(I) + \min(I)}{2}\,. \tag{11.12}$$

[1] Note that $\mu_0(q), \mu_1(q)$ are meant to be functions over q and thus $\mu_0(K–1)$ in Eqn. (11.6) denotes the mean of partition $\mathcal{C}_0$ for the threshold $K-1$.
[2] $\sigma_0^2(q)$ and $\sigma_1^2(q)$ in Eqn. (11.7) are also functions over q.

Alg. 11.1 Quantile thresholding. The optimal threshold value $q \in [0, K-2]$ is returned, or -1 if no valid threshold was found. Note the test in line 9 to check if the foreground is empty or not (the background is always non-empty by definition).

QuantileThreshold(h, p)
Input: $\mathsf{h} : [0, K-1] \mapsto \mathbb{N}$, a grayscale histogram. p, the proportion of expected background pixels ($0 < p < 1$). Returns the optimal threshold value or -1 if no threshold is found.

1: $K \leftarrow \mathsf{Size}(\mathsf{h})$ ▷ number of intensity levels
2: $MN \leftarrow \sum_{i=0}^{K-1} \mathsf{h}(i)$ ▷ number of image pixels
3: $i \leftarrow 0$
4: $c \leftarrow \mathsf{h}(0)$
5: **while** $(i < K) \wedge (c < MN \cdot p)$ **do** ▷ quantile calc. (Eq. 11.13)
6: $i \leftarrow i + 1$
7: $c \leftarrow c + \mathsf{h}(j)$
8: **if** $c < MN$ **then** ▷ foreground is non-empty
9: $q \leftarrow i$
10: **else** ▷ foreground is empty, all pixels are background
11: $q \leftarrow -1$
12: **return** q

Like the image mean μ_I (see Eqn. (3.9)), all these quantities can be obtained directly from the histogram h.

Thresholding at the median segments the image into approximately equal-sized background and foreground sets, that is, $|\mathcal{C}_0| \approx |\mathcal{C}_1|$, which assumes that the "interesting" (foreground) pixels cover about half of the image. This may be appropriate for certain images, but completely wrong for others. For example, a scanned text image will typically contain a lot more white than black pixels, so using the median threshold would probably be unsatisfactory in this case. If the approximate fraction p ($0 < p < 1$) of expected background pixels is known in advance, the threshold could be set to that *quantile* instead. In this case, q is simply chosen as

$$q \leftarrow \min\{\, i \mid \sum_{j=0}^{i} \mathsf{h}(i) \geq M \cdot N \cdot p \,\}, \tag{11.13}$$

where N is the total number of pixels. We see that the *median* is only a special case of a quantile measure, with $p = 0.5$. This simple thresholding method is summarized in Alg. 11.1.

For the *mid-range* technique (Eqn. (11.12)), the limiting intensity values $\min(I)$ and $\max(I)$ can be found by searching for the smallest and largest non-zero entries, respectively, in the histogram h. The mid-range threshold segments the image at 50 % (or any other percentile) of the contrast range. In this case, nothing can be said in general about the relative sizes of the resulting background and foreground partitions. Because a single extreme pixel value (outlier) may change the contrast range dramatically, this approach is not very robust. Here too it is advantageous to define the contrast range by specifying pixel *quantiles*, analogous to the calculation of the quantities a'_{low} and a'_{high} in the modified auto-contrast function (see Ch. 4, Sec. 4.4).

In the pathological (but nevertheless possible) case that all pixels in the image have the *same* intensity g, all the aforementioned meth-

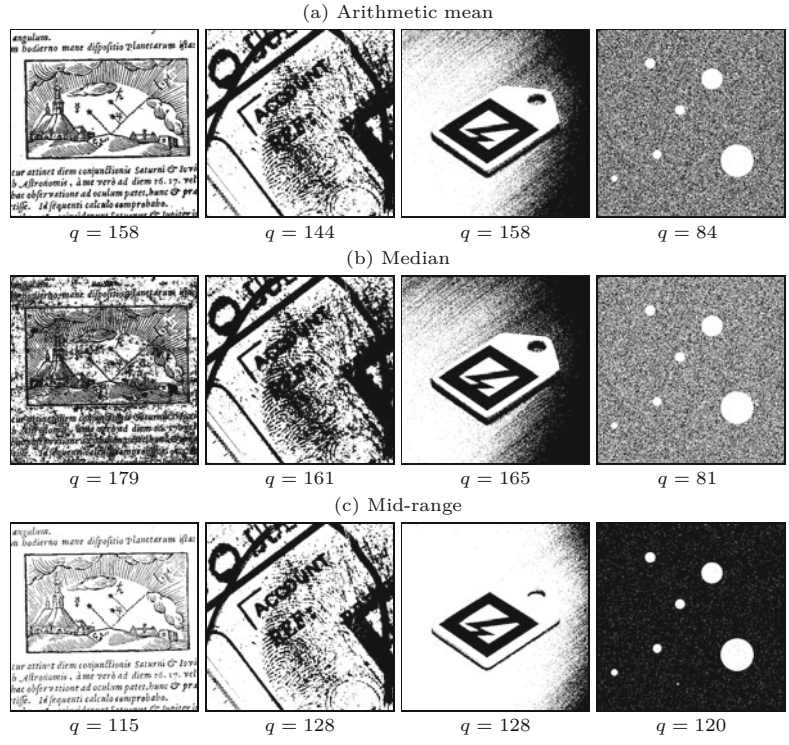


Fig. 11.2
Results from various simple thresholding schemes. Mean (a–d), median (e–h), and mid-range (i–l) threshold, as specified in Eqns. (11.10)–(11.12).

ods will return the threshold $q = g$, which assigns all pixels to the background partition and leaves the foreground empty. Algorithms should try to detect this situation, because thresholding a uniform image obviously makes no sense. Results obtained with these simple thresholding techniques are shown in Fig. 11.2. Despite the obvious limitations, even a simple automatic threshold selection (such as the quantile technique in Alg. 11.1) will typically yield more reliable results than the use of a fixed threshold.

11.1.3 Iterative Threshold Selection (Isodata Algorithm)

This classic iterative algorithm for finding an optimal threshold is attributed to Ridler and Calvard [198] and was related to Isodata clustering by Velasco [242]. It is thus sometimes referred to as the "isodata" or "intermeans" algorithm. Like in many other global thresholding schemes it is assumed that the image's histogram is a mixture of two separate distributions, one for the intensities of the background pixels and the other for the foreground pixels. In this case, the two distributions are assumed to be Gaussian with approximately identical spreads (variances).

The algorithm starts by making an initial guess for the threshold, for example, by taking the mean or the median of the whole image. This splits the set of pixels into a background and a foreground set, both of which should be non-empty. Next, the means of both sets are calculated and the threshold is repositioned to their average, that is, centered between the two means. The means are then re-calculated for the resulting background and foreground sets, and so on, until

Alg. 11.2 "Isodata" threshold selection based on the iterative method by Ridler and Calvard [198].

1: **IsodataThreshold**(h)
Input: $\mathsf{h} : [0, K-1] \mapsto \mathbb{N}$, a grayscale histogram.
Returns the optimal threshold value or -1 if no threshold is found.
2: $K \leftarrow \mathsf{Size}(\mathsf{h})$ ▷ number of intensity levels
3: $q \leftarrow \mathsf{Mean}(\mathsf{h}, 0, K-1)$ ▷ set initial threshold to overall mean
4: **repeat**
5: $n_0 \leftarrow \mathsf{Count}(\mathsf{h}, 0, q)$ ▷ background population
6: $n_1 \leftarrow \mathsf{Count}(\mathsf{h}, q+1, K-1)$ ▷ foreground population
7: **if** $(n_0 = 0) \vee (n_1 = 0)$ **then** ▷ backgrd. or foregrd. is empty
8: **return** -1
9: $\mu_0 \leftarrow \mathsf{Mean}(\mathsf{h}, 0, q)$ ▷ background mean
10: $\mu_1 \leftarrow \mathsf{Mean}(\mathsf{h}, q+1, K-1)$ ▷ foreground mean
11: $q' \leftarrow q$ ▷ keep previous threshold
12: $q \leftarrow \left\lfloor \frac{\mu_0 + \mu_1}{2} \right\rfloor$ ▷ calculate the new threshold
13: **until** $q = q'$ ▷ terminate if no change
14: **return** q

15: $\mathsf{Count}(\mathsf{h}, a, b) := \sum_{g=a}^{b} \mathsf{h}(g)$

16: $\mathsf{Mean}(\mathsf{h}, a, b) := \left[\sum_{g=a}^{b} g \cdot \mathsf{h}(g)\right] / \left[\sum_{g=a}^{b} \mathsf{h}(g)\right]$

the threshold does not change any longer. In practice, it takes only a few iterations for the threshold to converge.

This procedure is summarized in Alg. 11.2. The initial threshold is set to the overall mean (line 3). For each threshold q, separate mean values μ_0, μ_1 are computed for the corresponding foreground and background partitions. The threshold is repeatedly set to the average of the two means until no more change occurs. The clause in line 7 tests if either the background or the foreground partition is empty, which will happen, for example, if the image contains only a single intensity value. In this case, no valid threshold exists and the procedure returns -1. The functions $\mathsf{Count}(\mathsf{h}, a, b)$ and $\mathsf{Mean}(\mathsf{h}, a, b)$ in lines 15–16 return the number of pixels and the mean, respectively, of the image pixels with intensity values in the range $[a, b]$. Both can be computed directly from the histogram h without inspecting the image itself.

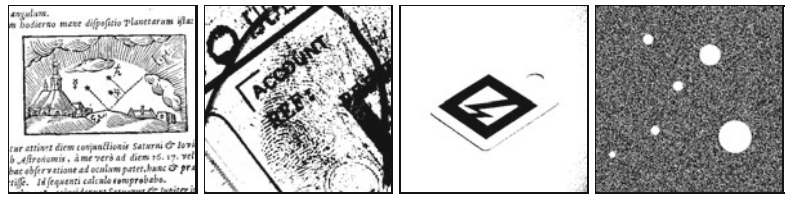
The performance of this algorithm can be easily improved by using tables $\boldsymbol{\mu}_0(q), \boldsymbol{\mu}_1(q)$ for the background and foreground means, respectively. The modified, table-based version of the iterative threshold selection procedure is shown in Alg. 11.3. It requires two passes over the histogram to initialize the tables $\boldsymbol{\mu}_0$, $\boldsymbol{\mu}_1$ and only a small, constant number of computations for each iteration in its main loop. Note that the image's overall mean μ_I, used as the initial guess for the threshold q (Alg. 11.3, line 4), need not be calculated separately but can be obtained as $\mu_I = \boldsymbol{\mu}_0(K-1)$, given that threshold $q = K-1$ assigns all image pixels to the background. The time complexity of this algorithm is thus $\mathcal{O}(K)$, that is, linear w.r.t. the size of the

Alg. 11.3 Fast version of "isodata" threshold selection. Precalculated tables are used for the foreground and background means $\boldsymbol{\mu}_0$ and $\boldsymbol{\mu}_1$, respectively.

FastIsodataThreshold(h)
Input: $\mathsf{h} : [0, K-1] \mapsto \mathbb{N}$, a grayscale histogram.
Returns the optimal threshold value or -1 if no threshold is found.

$K \leftarrow \mathsf{Size}(\mathsf{h})$ ▷ number of intensity levels
$\langle \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, N \rangle \leftarrow \mathsf{MakeMeanTables}(\mathsf{h})$
$q \leftarrow \lfloor \boldsymbol{\mu}_0(K-1) \rfloor$ ▷ take the overall mean μ_I as initial threshold
repeat
  if $(\boldsymbol{\mu}_0(q) < 0) \lor (\boldsymbol{\mu}_1(q) < 0)$ **then**
    return -1 ▷ background or foreground is empty
  $q' \leftarrow q$ ▷ keep previous threshold
  $q \leftarrow \left\lfloor \dfrac{\boldsymbol{\mu}_0(q) + \boldsymbol{\mu}_1(q)}{2} \right\rfloor$ ▷ calculate the new threshold
until $q = q'$ ▷ terminate if no change
return q

MakeMeanTables(h)
$K \leftarrow \mathsf{Size}(\mathsf{h})$
Create maps $\boldsymbol{\mu}_0, \boldsymbol{\mu}_1 : [0, K-1] \mapsto \mathbb{R}$
$n_0 \leftarrow 0, \quad s_0 \leftarrow 0$
for $q \leftarrow 0, \cdots, K-1$ **do** ▷ tabulate background means $\mu_0(q)$
  $n_0 \leftarrow n_0 + \mathsf{h}(q)$
  $s_0 \leftarrow s_0 + q \cdot \mathsf{h}(q)$
  $\boldsymbol{\mu}_0(q) \leftarrow \begin{cases} s_0/n_0 & \text{if } n_0 > 0 \\ -1 & \text{otherwise} \end{cases}$
$N \leftarrow n_0$
$n_1 \leftarrow 0, \quad s_1 \leftarrow 0$
$\boldsymbol{\mu}_1(K-1) \leftarrow 0$
for $q \leftarrow K-2, \cdots, 0$ **do** ▷ tabulate foreground means $\mu_1(q)$
  $n_1 \leftarrow n_1 + \mathsf{h}(q+1)$
  $s_1 \leftarrow s_1 + (q+1) \cdot \mathsf{h}(q+1)$
  $\boldsymbol{\mu}_1(q) \leftarrow \begin{cases} s_1/n_1 & \text{if } n_1 > 0 \\ -1 & \text{otherwise} \end{cases}$
return $\langle \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, N \rangle$



(a) $q = 128$ (b) $q = 125$ (c) $q = 94$ (d) $q = 90$

Fig. 11.3 Thresholding with the isodata algorithm. Binarized images and the corresponding optimal threshold values (q).

histogram. Figure 11.3 shows the results of thresholding with the isodata algorithm applied to the test images in Fig. 11.1.

11.1.4 Otsu's Method

The method proposed by Otsu [147, 177] also assumes that the original image contains pixels from two classes, whose intensity distributions are unknown. The goal is to find a threshold q such that the resulting background and foreground distributions are maximally separated, which means that they are (a) each as narrow as possi-

ble (have minimal variances) and (b) their centers (means) are most distant from each other.

For a given threshold q, the variances of the corresponding background and foreground partitions can be calculated straight from the image's histogram (see Eqn. (11.7)). The combined width of the two distributions is measured by the *within-class* variance

$$\sigma_{\mathrm{w}}^2(q) = \mathsf{P}_0(q) \cdot \sigma_0^2(q) + \mathsf{P}_1(q) \cdot \sigma_1^2(q) \tag{11.14}$$

$$= \frac{1}{MN} \cdot \big[n_0(q) \cdot \sigma_0^2(q) + n_1(q) \cdot \sigma_1^2(q)\big], \tag{11.15}$$

where

$$\mathsf{P}_0(q) = \sum_{i=0}^{q} \mathsf{p}(i) = \frac{1}{MN} \cdot \sum_{i=0}^{q} \mathsf{h}(i) = \frac{n_0(q)}{MN}, \tag{11.16}$$

$$\mathsf{P}_1(q) = \sum_{i=q+1}^{K-1} \mathsf{p}(i) = \frac{1}{MN} \cdot \sum_{i=q+1}^{K-1} \mathsf{h}(i) = \frac{n_1(q)}{MN} \tag{11.17}$$

are the class probabilities for $\mathcal{C}_0$, $\mathcal{C}_1$, respectively. Thus the within-class variance in Eqn. (11.15) is simply the sum of the individual variances weighted by the corresponding class probabilities or "populations". Analogously, the *between-class* variance,

$$\sigma_{\mathrm{b}}^2(q) = \mathsf{P}_0(q) \cdot \big(\mu_0(q) - \mu_I\big)^2 + \mathsf{P}_1(q) \cdot \big(\mu_1(q) - \mu_I\big)^2 \tag{11.18}$$

$$= \frac{1}{MN} \big[n_0(q) \cdot \big(\mu_0(q) - \mu_I\big)^2 + n_1(q) \cdot \big(\mu_1(q) - \mu_I\big)^2\big] \tag{11.19}$$

measures the distances between the cluster means μ_0, μ_1 and the overall mean μ_I. The total image variance σ_I^2 is the sum of the within-class variance and the between-class variance, that is,

$$\sigma_I^2 = \sigma_{\mathrm{w}}^2(q) + \sigma_{\mathrm{b}}^2(q), \tag{11.20}$$

for $q = 0, \ldots, K-1$. Since σ_I^2 is constant for a given image, the threshold q can be found by either *minimizing* the within-variance σ_{w}^2 or *maximizing* the between-variance σ_{b}^2. The natural choice is to maximize σ_{b}^2, because it only relies on first-order statistics (i.e., the within-class means μ_0, μ_1). Since the overall mean μ_I can be expressed as the weighted sum of the partition means μ_0 and μ_1 (Eqn. (11.6)), we can simplify Eqn. (11.19) to

$$\sigma_{\mathrm{b}}^2(q) = \mathsf{P}_0(q) \cdot \mathsf{P}_1(q) \cdot \big[\mu_0(q) - \mu_1(q)\big]^2 \tag{11.21}$$

$$= \frac{1}{(MN)^2} \cdot n_0(q) \cdot n_1(q) \cdot \big[\mu_0(q) - \mu_1(q)\big]^2. \tag{11.22}$$

The optimal threshold is finally found by *maximizing* the expression for the between-class variance in Eqn. (11.22) with respect to q, thereby *minimizing* the within-class variance in Eqn. (11.15).

Noting that $\sigma_{\mathrm{b}}^2(q)$ only depends on the means (and *not* on the variances) of the two partitions for a given threshold q allows for a very efficient implementation, as outlined in Alg. 11.4. The algorithm assumes a grayscale image with a total of N pixels and K intensity

Alg. 11.4 Finding the optimal threshold using Otsu's method [177]. Initially (outside the **for**-loop), the threshold q is assumed to be -1, which corresponds to the background class being empty ($n_0 = 0$) and all pixels are assigned to the foreground class ($n_1 = N$). The **for**-loop (lines 7–14) examines each possible threshold $q = 0, \ldots, K-2$. The factor $1/(MN)^2$ in line 11 is constant and thus not relevant for the optimization. The optimal threshold value is returned, or -1 if no valid threshold was found. The function MakeMeanTables() is defined in Alg. 11.3.

```
1:  OtsuThreshold(h)
        Input: h : [0, K−1] ↦ ℕ, a grayscale histogram. Returns the
        optimal threshold value or −1 if no threshold is found.
2:      K ← Size(h)                                  ▷ number of intensity levels
3:      (μ_0, μ_1, MN) ← MakeMeanTables(h)           ▷ see Alg. 11.3
4:      σ²_bmax ← 0
5:      q_max ← −1
6:      n_0 ← 0
7:      for q ← 0, ···, K−2 do ▷ examine all possible threshold values q
8:          n_0 ← n_0 + h(q)
9:          n_1 ← MN − n_0
10:         if (n_0 > 0) ∧ (n_1 > 0) then
11:             σ²_b ← 1/(MN)² · n_0 · n_1 · [μ_0(q) − μ_1(q)]²   ▷ see Eq. 11.22
12:             if σ²_b > σ²_bmax then                ▷ maximize σ²_b
13:                 σ²_bmax ← σ²_b
14:                 q_max ← q
15:     return q_max
```

levels. As in Alg. 11.3, precalculated tables $\boldsymbol{\mu}_0(q), \boldsymbol{\mu}_1(q)$ are used for the background and foreground means for all possible threshold values $q = 0, \ldots, K-1$.

Possible threshold values are $q = 0, \ldots, K-2$ (with $q = K-1$, all pixels are assigned to the background). Initially (before entering the main **for**-loop in line 7) $q = -1$; at this point, the set of background pixels ($\leq q$) is empty and all pixels are classified as foreground ($n_0 = 0$ and $n_1 = N$). Each possible threshold value is examined inside the body of the **for**-loop.

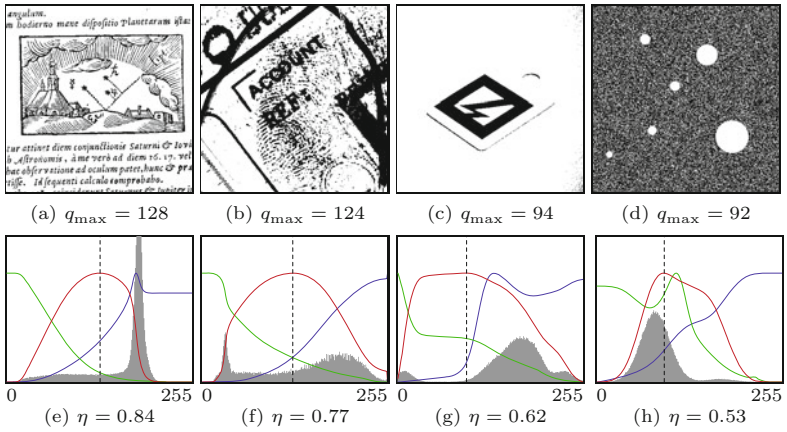
As long as any of the two classes is empty ($n_0(q) = 0$ or $n_1(q) = 0$),[3] the resulting between-class variance $\sigma_b^2(q)$ is zero. The threshold that yields the maximum between-class variance (σ_{bmax}^2) is returned, or -1 if no valid threshold could be found. This occurs when all image pixels have the same intensity, that all pixels are in either the background or the foreground class.

Note that in line 11 of Alg. 11.4, the factor $\frac{1}{N^2}$ is constant (independent of q) and can thus be ignored in the optimization. However, care must be taken at this point because the computation of σ_b^2 may produce intermediate values that exceed the range of typical (32-bit) integer variables, even for medium-size images. Variables of type `long` should be used or the computation be performed with floating-point values.

The absolute "goodness" of the final thresholding by q_{max} could be measured as the ratio

$$\eta = \frac{\sigma_b^2(q_{max})}{\sigma_I^2} \in [0, 1] \tag{11.23}$$

[3] This is the case if the image contains no pixels with values $I(u, v) \leq q$ or $I(u, v) > q$, that is, the histogram h is empty either below or above the index q.



(a) $q_{\max} = 128$ (b) $q_{\max} = 124$ (c) $q_{\max} = 94$ (d) $q_{\max} = 92$

(e) $\eta = 0.84$ (f) $\eta = 0.77$ (g) $\eta = 0.62$ (h) $\eta = 0.53$

Fig. 11.4
Results of thresholding with Otsu's method. Calculated threshold values q and resulting binary images (a–d). Graphs in (e–h) show the corresponding within-background variance σ_0^2 (green), the within-foreground variance σ_1^2 (blue), and the between-class variance σ_b^2 (red), for varying threshold values $q = 0, \ldots, 255$. The optimal threshold $q_{\max}$ (dashed vertical line) is positioned at the maximum of σ_b^2. The value η denotes the "goodness" estimate for the thresholding, as defined in Eqn. (11.23).

(see Eqn. (11.8)), which is invariant under linear changes of contrast and brightness [177]. Greater values of η indicate better thresholding.

Results of automatic threshold selection with Otsu's method are shown in Fig. 11.4, where $q_{\max}$ denotes the optimal threshold and η is the corresponding "goodness" estimate, as defined in Eqn. (11.23). The graph underneath each image shows the original histogram (gray) overlaid with the variance within the background σ_0^2 (green), the variance within the foreground σ_1^2 (blue), and the between-class variance σ_b^2 (red) for varying threshold values q. The dashed vertical line marks the position of the optimal threshold $q_{\max}$.

Due to the pre-calculation of the mean values, Otsu's method requires only three passes over the histogram and is thus very fast ($\mathcal{O}(K)$), in contrast to opposite accounts in the literature. The method is frequently quoted and performs well in comparison to other approaches [213], despite its long history and its simplicity. In general, the results are very similar to the ones produced by the iterative threshold selection ("isodata") algorithm described in Sec. 11.1.3.

11.1.5 Maximum Entropy Thresholding

Entropy is an important concept in information theory and particularly in data compression. It is a statistical measure that quantifies the average amount of information contained in the "messages" generated by a stochastic data source [99, 101]. For example, the MN pixels in an image I can be interpreted as a message of MN symbols, each taken independently from a finite alphabet of K (e.g., 256) different intensity values. Every pixel is assumed to be statically independent. Knowing the probability of each intensity value g to occur, entropy measures how likely it is to observe a particular image, or, in other words, how much we should be surprised to see such an image. Before going into further details, we briefly review the notion of probabilities in the context of images and histograms (see also Ch. 4, Sec. 4.6.1).

For modeling the image generation as a random process, we first need to define an "alphabet", that is, a set of symbols

$$Z = \{0, 1, \ldots, K-1\}, \tag{11.24}$$

which in this case is simply the set of possible intensity values $g = 0, \ldots, K-1$, together with the probability $p(g)$ that a particular intensity value g occurs. These probabilities are supposed to be known in advance, which is why they are called *a priori* (or *prior*) probabilities. The vector of probabilities,

$$\big(p(0), p(1), \ldots, p(K-1)\big),$$

is a *probability distribution* or *probability density function* (pdf). In practice, the *a priori* probabilities are usually *un*known, but they can be estimated by observing how often the intensity values actually occur in one or more images, assuming that these are representative instances of the images typically produced by that source. An estimate $\mathsf{p}(g)$ of the image's probability density function $p(g)$ is obtained by normalizing its histogram h in the form

$$p(g) \approx \mathsf{p}(g) = \frac{\mathsf{h}(g)}{MN}, \tag{11.25}$$

for $0 \le g < K$, such that $0 \le \mathsf{p}(g) \le 1$ and $\sum_{g=0}^{K-1} \mathsf{p}(g) = 1$. The associated *cumulative distribution function* (cdf) is

$$\mathsf{P}(g) = \sum_{i=0}^{g} \frac{\mathsf{h}(g)}{MN} = \sum_{i=0}^{g} \mathsf{p}(i), \tag{11.26}$$

where $\mathsf{P}(0) = \mathsf{p}(0)$ and $\mathsf{P}(K-1) = 1$. This is simply the normalized *cumulative histogram*.[4]

Entropy of images

Given an estimate of its intensity probability distribution $\mathsf{p}(g)$, the *entropy* of an image is defined as[5]

$$H(Z) = \sum_{g \in Z} \mathsf{p}(g) \cdot \log_b\Big(\frac{1}{\mathsf{p}(g)}\Big) = -\sum_{g \in Z} \mathsf{p}(g) \cdot \log_b\big(\mathsf{p}(g)\big), \tag{11.27}$$

where $g = I(u, v)$ and $\log_b(x)$ denotes the logarithm of x to the base b. If $b = 2$, the entropy (or "information content") is measured in *bits*, but proportional results are obtained with any other logarithm (such as ln or $\log_{10}$). Note that the value of $H()$ is always positive, because the probabilities $\mathsf{p}()$ are in $[0, 1]$ and thus the terms $\log_b[\mathsf{p}()]$ are negative or zero for any b.

Some other properties of the entropy are also quite intuitive. For example, if all probabilities $\mathsf{p}(g)$ are zero except for one intensity g', then the entropy $H(I)$ is *zero*, indicating that there is no uncertainty (or "surprise") in the messages produced by the corresponding data source. The (rather boring) images generated by this source will contain nothing but pixels of intensity g', since all other intensities are

[4] See also Chapter 3, Sec. 3.6.

[5] Note the subtle difference in notation for the cumulative histogram H and the entropy H.

impossible. Conversely, the entropy is a maximum if all K intensities have the same probability (uniform distribution),

$$\mathsf{p}(g) = \frac{1}{K}, \qquad \text{for } 0 \le g < K, \tag{11.28}$$

and therefore (from Eqn. (11.27)) the entropy in this case is

$$H(Z) = -\sum_{i=0}^{K-1} \frac{1}{K} \cdot \log_b\Big(\frac{1}{K}\Big) = \frac{1}{K} \cdot \underbrace{\sum_{i=0}^{K-1} \log_b(K)}_{K \cdot \log_b(K)} \tag{11.29}$$

$$= \frac{1}{K} \cdot \big(K \cdot \log_b(K)\big) = \log_b(K). \tag{11.30}$$

This is the maximum possible entropy of a stochastic source with an alphabet Z of size K. Thus the entropy $H(Z)$ is always in the range $[0, \log(K)]$.

Using image entropy for threshold selection

The use of image entropy as a criterion for threshold selection has a long tradition and numerous methods have been proposed. In the following, we describe the early (but still popular) technique by Kapur et al. [100, 133] as a representative example.

Given a particular threshold q (with $0 \le q < K-1$), the estimated probability distributions for the resulting partitions $\mathcal{C}_0$ and $\mathcal{C}_1$ are

$$\begin{aligned} \mathcal{C}_0 &: \Big(\tfrac{\mathsf{p}(0)}{\mathsf{P}_0(q)} \;\; \tfrac{\mathsf{p}(1)}{\mathsf{P}_0(q)} \;\; \dots \;\; \tfrac{\mathsf{p}(q)}{\mathsf{P}_0(q)} \;\; 0 \;\; 0 \;\; \dots \;\; 0 \Big), \\ \mathcal{C}_1 &: \Big(0 \;\; 0 \;\; \dots \;\; 0 \;\; \tfrac{\mathsf{p}(q+1)}{\mathsf{P}_1(q)} \;\; \tfrac{\mathsf{p}(q+2)}{\mathsf{P}_1(q)} \;\; \dots \;\; \tfrac{\mathsf{p}(K-1)}{\mathsf{P}_1(q)} \Big), \end{aligned} \tag{11.31}$$

with the associated cumulated probabilities (see Eqn. (11.26))

$$\mathsf{P}_0(q) = \sum_{i=0}^{q} \mathsf{p}(i) = \mathsf{P}(q) \qquad \text{and} \qquad \mathsf{P}_1(q) = \sum_{i=q+1}^{K-1} \mathsf{p}(i) = 1 - \mathsf{P}(q). \tag{11.32}$$

Note that $\mathsf{P}_0(q) + \mathsf{P}_1(q) = 1$, since the background and foreground partitions are disjoint. The entropies *within* each partition are defined as

$$H_0(q) = -\sum_{i=0}^{q} \frac{\mathsf{p}(i)}{\mathsf{P}_0(q)} \cdot \log\Big(\frac{\mathsf{p}(i)}{\mathsf{P}_0(q)}\Big), \tag{11.33}$$

$$H_1(q) = -\sum_{i=q+1}^{K-1} \frac{\mathsf{p}(i)}{\mathsf{P}_1(q)} \cdot \log\Big(\frac{\mathsf{p}(i)}{\mathsf{P}_1(q)}\Big), \tag{11.34}$$

and the *overall* entropy for the threshold q is

$$H_{01}(q) = H_0(q) + H_1(q). \tag{11.35}$$

This expression is to be maximized over q, also called the "information between the classes" $\mathcal{C}_0$ and $\mathcal{C}_1$. To allow for an efficient computation, the expression for $H_0(q)$ in Eqn. (11.33) can be rearranged to

$$H_0(q) = -\sum_{i=0}^{q} \frac{\mathsf{p}(i)}{\mathsf{P}_0(q)} \cdot \big[\log\big(\mathsf{p}(i)\big) - \log\big(\mathsf{P}_0(q)\big)\big] \tag{11.36}$$

$$= -\frac{1}{\mathsf{P}_0(q)} \cdot \sum_{i=0}^{q} \mathsf{p}(i) \cdot \big[\log\big(\mathsf{p}(i)\big) - \log\big(\mathsf{P}_0(q)\big)\big] \tag{11.37}$$

$$= -\frac{1}{\mathsf{P}_0(q)} \cdot \underbrace{\sum_{i=0}^{q} \mathsf{p}(i) \cdot \log\big(\mathsf{p}(i)\big)}_{S_0(q)} + \frac{1}{\mathsf{P}_0(q)} \cdot \underbrace{\sum_{i=0}^{q} \mathsf{p}(i)}_{=\mathsf{P}_0(q)} \cdot \log\big(\mathsf{P}_0(q)\big)$$

$$= -\frac{1}{\mathsf{P}_0(q)} \cdot S_0(q) + \log\big(\mathsf{P}_0(q)\big). \tag{11.38}$$

Similarly $H_1(q)$ in Eqn. (11.34) becomes

$$H_1(q) = -\sum_{i=q+1}^{K-1} \frac{\mathsf{p}(i)}{\mathsf{P}_1(q)} \cdot \big[\log\big(\mathsf{p}(i)\big) - \log\big(\mathsf{P}_1(q)\big)\big] \tag{11.39}$$

$$= -\frac{1}{1-\mathsf{P}_0(q)} \cdot S_1(q) + \log\big(1-\mathsf{P}_0(q)\big). \tag{11.40}$$

Given the estimated probability distribution $\mathsf{p}(i)$, the cumulative probability P_0 and the summation terms S_0, S_1 (see Eqns. (11.38)–(11.40)) can be calculated from the recurrence relations

$$\mathsf{P}_0(q) = \begin{cases} \mathsf{p}(0) & \text{for } q = 0, \\ \mathsf{P}_0(q-1) + \mathsf{p}(q) & \text{for } 0 < q < K, \end{cases}$$

$$S_0(q) = \begin{cases} \mathsf{p}(0) \cdot \log\big(\mathsf{p}(0)\big) & \text{for } q = 0, \\ S_0(q-1) + \mathsf{p}(q) \cdot \log\big(\mathsf{p}(q)\big) & \text{for } 0 < q < K, \end{cases}$$

$$S_1(q) = \begin{cases} 0 & \text{for } q = K-1, \\ S_1(q+1) + \mathsf{p}(q+1) \cdot \log\big(\mathsf{p}(q+1)\big) & \text{for } 0 \le q < K-1. \end{cases} \tag{11.41}$$

The complete procedure is summarized in Alg. 11.5, where the values $S_0(q), S_1(q)$ are obtained from precalculated tables $\mathsf{S}_0, \mathsf{S}_1$. The algorithm performs three passes over the histogram of length K (two for filling the tables $\mathsf{S}_0, \mathsf{S}_1$ and one in the main loop), so its time complexity is $\mathcal{O}(K)$, like the algorithms described before.

Results obtained with this technique are shown in Fig. 11.5. The technique described in this section is simple and efficient, because it again relies entirely on the image's histogram. More advanced entropy-based thresholding techniques exist that, among other improvements, take into account the spatial structure of the original image. An extensive review of entropy-based methods can be found in [46].

11.1.6 Minimum Error Thresholding

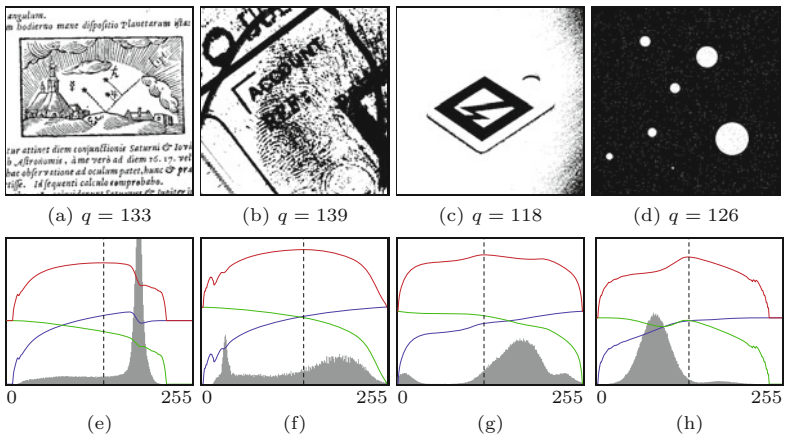
The goal of minimum error thresholding is to optimally fit a combination (mixture) of Gaussian distributions to the image's histogram. Before we proceed, we briefly look at some additional concepts from statistics. Note, however, that the following material is only intended

MaximumEntropyThreshold(h)
Input: $\mathsf{h} : [0, K-1] \mapsto \mathbb{N}$, a grayscale histogram. Returns the optimal threshold value or -1 if no threshold is found.

2: $K \leftarrow \mathsf{Size}(\mathsf{h})$ ▷ number of intensity levels
3: $\mathsf{p} \leftarrow \mathsf{Normalize}(\mathsf{h})$ ▷ normalize histogram
4: $(\mathsf{S}_0, \mathsf{S}_1) \leftarrow \mathsf{MakeTables}(\mathsf{p}, K)$ ▷ tables for $S_0(q), S_1(q)$
5: $\mathsf{P}_0 \leftarrow 0$ ▷ $\mathsf{P}_0 \in [0, 1]$
6: $q_{\max} \leftarrow -1$
7: $H_{\max} \leftarrow -\infty$ ▷ maximum joint entropy
8: **for** $q \leftarrow 0, \cdots, K-2$ **do** ▷ check all possible threshold values q
9: $\mathsf{P}_0 \leftarrow \mathsf{P}_0 + \mathsf{p}(q)$
10: $\mathsf{P}_1 \leftarrow 1 - \mathsf{P}_0$ ▷ $\mathsf{P}_1 \in [0, 1]$
11: $H_0 \leftarrow \begin{cases} -\frac{1}{\mathsf{P}_0} \cdot \mathsf{S}_0(q) + \log(\mathsf{P}_0) & \text{if } \mathsf{P}_0 > 0 \\ 0 & \text{otherwise} \end{cases}$ ▷ BG entropy
12: $H_1 \leftarrow \begin{cases} -\frac{1}{\mathsf{P}_1} \cdot \mathsf{S}_1(q) + \log(\mathsf{P}_1) & \text{if } \mathsf{P}_1 > 0 \\ 0 & \text{otherwise} \end{cases}$ ▷ FG entropy
13: $H_{01} = H_0 + H_1$ ▷ *overall* entropy for q
14: **if** $H_{01} > H_{\max}$ **then** ▷ maximize $H_{01}(q)$
15: $H_{\max} \leftarrow H_{01}$
16: $q_{\max} \leftarrow q$
17: **return** $q_{\max}$

MakeTables(p, K)
Create maps $\mathsf{S}_0, \mathsf{S}_1 : [0, K-1] \mapsto \mathbb{R}$
$s_0 \leftarrow 0$
for $i \leftarrow 0, \cdots, K-1$ **do** ▷ initialize table S_0
if $\mathsf{p}(i) > 0$ **then**
$s_0 \leftarrow s_0 + \mathsf{p}(i) \cdot \log\big(\mathsf{p}(i)\big)$
$\mathsf{S}_0(i) \leftarrow s_0$
$s_1 \leftarrow 0$
for $i \leftarrow K-1, \cdots, 0$ **do** ▷ initialize table S_1
$\mathsf{S}_1(i) \leftarrow s_1$
if $\mathsf{p}(i) > 0$ **then**
$s_1 \leftarrow s_1 + \mathsf{p}(i) \cdot \log\big(\mathsf{p}(i)\big)$
return $(\mathsf{S}_0, \mathsf{S}_1)$

Alg. 11.5 Maximum entropy threshold selection after Kapur et al. [133]. Initially (outside the **for**-loop), the threshold q is assumed to be -1, which corresponds to the background class being empty ($n_0 = 0$) and all pixels assigned to the foreground class ($n_1 = N$). The **for**-loop (lines 8–16) examines each possible threshold $q = 0, \ldots, K-2$. The optimal threshold value $(0, \ldots, K-2)$ is returned, or -1 if no valid threshold was found.



(a) $q = 133$ (b) $q = 139$ (c) $q = 118$ (d) $q = 126$
(e) (f) (g) (h)

Fig. 11.5 Thresholding with the Maximum-entropy method. Calculated threshold values q and resulting binary images (a–d). Graphs in (e–h) show the *background* entropy $H_0(q)$ (green), *foreground* entropy $H_1(q)$ (blue) and *overall* entropy $H_{01}(q) = H_0(q) + H_1(q)$ (red), for varying threshold values q. The optimal threshold $q_{\max}$ is found at the maximum of H_{01} (dashed vertical line).

as a superficial outline to explain the elementary concepts. For a solid grounding of these and related topics readers are referred to the excellent texts available on statistical pattern recognition, such as [24, 64].

Bayesian decision making

The assumption is again that the image pixels originate from one of two classes, $\mathcal{C}_0$ and $\mathcal{C}_1$, or background and foreground, respectively. Both classes generate random intensity values following unknown statistical distributions. Typically, these are modeled as Gaussian distributions with unknown parameters μ and σ^2, as will be described. The task is to decide for each pixel value x to which of the two classes it most likely belongs. Bayesian reasoning is a classic technique for making such decisions in a probabilistic context.

The *probability*, that a certain intensity value x originates from a background pixel is denoted

$$p(x \mid \mathcal{C}_0).$$

This is called a "conditional probability".[6] It tells us how likely it is to observe the gray value x when a pixel is a member of the background class $\mathcal{C}_0$. Analogously, $p(x \mid \mathcal{C}_1)$ is the conditional probability of observing the value x when a pixel is known to be of the foreground class $\mathcal{C}_1$.

For the moment, let us assume that the conditional probability functions $p(x \mid \mathcal{C}_0)$ and $p(x \mid \mathcal{C}_1)$ are known. Our problem is reversed though, namely to decide which class a pixel most likely belongs to, given that its intensity is x. This means that we are actually interested in the conditional probabilities

$$p(\mathcal{C}_0 \mid x) \qquad \text{and} \qquad p(\mathcal{C}_1 \mid x), \tag{11.42}$$

also called *a posteriori* (or *posterior*) probabilities. If we knew these, we could simply select the class with the higher probability in the form

$$\mathcal{C} = \begin{cases} \mathcal{C}_0 & \text{if } p(\mathcal{C}_0 \mid x) > p(\mathcal{C}_1 \mid x), \\ \mathcal{C}_1 & \text{otherwise.} \end{cases} \tag{11.43}$$

Bayes' theorem provides a method for estimating these *posterior* probabilities, that is,

$$p(\mathcal{C}_j \mid x) = \frac{p(x \mid \mathcal{C}_j) \cdot p(\mathcal{C}_j)}{p(x)}, \tag{11.44}$$

where $p(\mathcal{C}_j)$ is the *prior* probability of class $\mathcal{C}_j$. While, in theory, the prior probabilities are also unknown, they can be easily estimated from the image histogram (see also Sec. 11.1.5). Finally, $p(x)$ in Eqn. (11.44) is the overall probability of observing the intensity value x,

[6] In general, $p(A \mid B)$ denotes the (conditional) probability of observing the event A in a given situation B. It is usually read as "the probability of A, given B".

which is typically estimated from its relative frequency in one or more images.[7]

Note that for a particular intensity x, the corresponding evidence $p(x)$ only *scales* the posterior probabilities and is thus not relevant for the classification itself. Consequently, we can reformulate the binary decision rule in Eqn. (11.43) to

$$\mathcal{C} = \begin{cases} \mathcal{C}_0 & \text{if } p(x \mid \mathcal{C}_0) \cdot p(\mathcal{C}_0) > p(x \mid \mathcal{C}_1) \cdot p(\mathcal{C}_1), \\ \mathcal{C}_1 & \text{otherwise.} \end{cases} \tag{11.45}$$

This is called Bayes' decision rule. It minimizes the probability of making a classification error if the involved probabilities are known and is also called the "minimum error" criterion.

Gaussian probability distributions

If the probability distributions $p(x \mid \mathcal{C}_j)$ are modeled as *Gaussian*[8] distributions $\mathcal{N}(x \mid \mu_j, \sigma_j^2)$, where μ_j, σ_j^2 denote the *mean* and *variance* of class C_j, we can rewrite the scaled posterior probabilities in Eqn. (11.45) as

$$p(x \mid \mathcal{C}_j) \cdot p(\mathcal{C}_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} \cdot \exp\Big(-\frac{(x-\mu_j)^2}{2\sigma_j^2}\Big) \cdot p(\mathcal{C}_j). \tag{11.46}$$

As long as the ordering between the resulting class scores remains unchanged, these quantities can be scaled or transformed arbitrarily. In particular, it is common to use the *logarithm* of the above expression to avoid repeated multiplications of small numbers. For example, applying the natural logarithm[9] to both sides of Eqn. (11.46) yields

$$\ln\big(p(x \mid \mathcal{C}_j) \cdot p(\mathcal{C}_j)\big) = \ln\big(p(x \mid \mathcal{C}_j)\big) + \ln\big(p(\mathcal{C}_j)\big) \tag{11.47}$$

$$= \ln\Big(\frac{1}{\sqrt{2\pi\sigma_j^2}}\Big) + \ln\Big(\exp\Big(-\frac{(x-\mu_j)^2}{2\sigma_j^2}\Big)\Big) + \ln\big(p(\mathcal{C}_j)\big) \tag{11.48}$$

$$= -\frac{1}{2} \cdot \ln(2\pi) - \frac{1}{2} \cdot \ln(\sigma_j^2) - \frac{(x-\mu_j)^2}{2\sigma_j^2} + \ln\big(p(\mathcal{C}_j)\big) \tag{11.49}$$

$$= -\frac{1}{2} \cdot \Big[\ln(2\pi) + \frac{(x-\mu_j)^2}{\sigma_j^2} + \ln(\sigma_j^2) - 2 \cdot \ln\big(p(\mathcal{C}_j)\big)\Big]. \tag{11.50}$$

Since $\ln(2\pi)$ in Eqn. (11.50) is constant, it can be ignored for the classification decision, as well as the factor $\frac{1}{2}$ at the front. Thus, to find the class $\mathcal{C}_j$ that maximizes $p(x \mid \mathcal{C}_j) \cdot p(\mathcal{C}_j)$ for a given intensity value x, it is sufficient to *maximize* the quantity

$$-\Big[\frac{(x-\mu_j)^2}{\sigma_j^2} + 2 \cdot \big[\ln(\sigma_j) - \ln\big(p(\mathcal{C}_j)\big)\big]\Big] \tag{11.51}$$

or, alternatively, to *minimize*

[7] $p(x)$ is also called the "evidence" for the event x.
[8] See also Sec. D.4 in the Appendix.
[9] Any logarithm could be used but the natural logarithm complements the exponential function of the Gaussian.

$$\varepsilon_j(x) = \frac{(x-\mu_j)^2}{\sigma_j^2} + 2\cdot\big[\ln(\sigma_j) - \ln\big(p(\mathcal{C}_j)\big)\big]. \tag{11.52}$$

The quantity $\varepsilon_j(x)$ can be viewed as a *measure of the potential error* involved in classifying the observed value x as being of class $\mathcal{C}_j$. To obtain the decision associated with the minimum risk, we can modify the binary decision rule in Eqn. (11.45) to

$$\mathcal{C} = \begin{cases} \mathcal{C}_0 & \text{if } \varepsilon_0(x) \le \varepsilon_1(x), \\ \mathcal{C}_1 & \text{otherwise.} \end{cases} \tag{11.53}$$

Remember that this rule tells us how to correctly classify the observed intensity value x as being either of the background class $\mathcal{C}_0$ or the foreground class $\mathcal{C}_1$, assuming that the underlying distributions are really Gaussian and their parameters are well estimated.

Goodness of classification

If we apply a threshold q, all pixel values $g \le q$ are implicitly classified as $\mathcal{C}_0$ (background) and all $g > q$ as $\mathcal{C}_1$ (foreground). The goodness of this classification by q over all N image pixels $I(u,v)$ can be measured with the criterion function

$$\mathsf{e}(q) = \frac{1}{MN} \cdot \sum_{u,v} \begin{cases} \varepsilon_0(I(u,v)) & \text{for } I(u,v) \le q \\ \varepsilon_1(I(u,v)) & \text{for } I(u,v) > q \end{cases} \tag{11.54}$$

$$= \frac{1}{MN} \cdot \sum_{g=0}^{q} \mathsf{h}(g) \cdot \varepsilon_0(g) \;+\; \frac{1}{MN} \cdot \sum_{g=q+1}^{K-1} \mathsf{h}(g) \cdot \varepsilon_1(g) \tag{11.55}$$

$$= \sum_{g=0}^{q} \mathsf{p}(g) \cdot \varepsilon_0(g) \;+\; \sum_{g=q+1}^{K-1} \mathsf{p}(g) \cdot \varepsilon_1(g), \tag{11.56}$$

with the normalized frequencies $\mathsf{p}(g) = \mathsf{h}(g)/N$ and the function $\varepsilon_j(g)$ as defined in Eqn. (11.52). By substituting $\varepsilon_j(g)$ from Eqn. (11.52) and some mathematical gymnastics, $\mathsf{e}(q)$ can be written as

$$\begin{aligned} \mathsf{e}(q) = 1 &+ P_0(q) \cdot \ln\big(\sigma_0^2(q)\big) + P_1(q) \cdot \ln\big(\sigma_1^2(q)\big) \\ &- 2 \cdot P_0(q) \cdot \ln\big(P_0(q)\big) - 2 \cdot P_1(q) \cdot \ln\big(P_1(q)\big). \end{aligned} \tag{11.57}$$

The remaining task is to find the threshold q that *minimizes* $\mathsf{e}(q)$ (where the constant 1 in Eqn. (11.57) can be omitted, of course). For each possible threshold q, we only need to estimate (from the image's histogram, as in Eqn. (11.31)) the "prior" probabilities $P_0(q)$, $P_1(q)$ and the corresponding within-class variances $\sigma_0(q), \sigma_1(q)$. The *prior* probabilities for the background and foreground classes are estimated as

$$P_0(q) \approx \sum_{g=0}^{q} \mathsf{p}(g) = \frac{1}{MN} \cdot \sum_{g=0}^{q} \mathsf{h}(g) = \frac{n_0(q)}{MN}, \tag{11.58}$$

$$P_1(q) \approx \sum_{g=q+1}^{K-1} \mathsf{p}(g) = \frac{1}{MN} \cdot \sum_{g=q+1}^{K-1} \mathsf{h}(g) = \frac{n_1(q)}{MN}, \tag{11.59}$$

where $n_0(q) = \sum_{i=0}^{q} \mathsf{h}(i)$, $n_1(q) = \sum_{i=q+1}^{K-1} \mathsf{h}(i)$, and $MN = n_0(q) + n_1(q)$ is the total number of image pixels. Estimates for background and foreground variances ()$\sigma_0^2(q)$ and $\sigma_1^2(q)$, respectively) defined in Eqn. (11.7), can be calculated efficiently by expressing them in the form

$$\begin{aligned}\sigma_0^2(q) &\approx \frac{1}{n_0(q)} \cdot \Big[\underbrace{\sum_{g=0}^{q} \mathsf{h}(g) \cdot g^2}_{B_0(q)} - \frac{1}{n_0(q)} \cdot \underbrace{\Big(\sum_{g=0}^{q} \mathsf{h}(g) \cdot g\Big)^2}_{A_0(q)}\Big] \\ &= \frac{1}{n_0(q)} \cdot \big[B_0(q) - \frac{1}{n_0(q)} \cdot A_0^2(q)\big], \end{aligned} \tag{11.60}$$

$$\begin{aligned}\sigma_1^2(q) &\approx \frac{1}{n_1(q)} \cdot \Big[\underbrace{\sum_{g=q+1}^{K-1} \mathsf{h}(g) \cdot g^2}_{B_1(q)} - \frac{1}{n_1(q)} \cdot \underbrace{\Big(\sum_{g=q+1}^{K-1} \mathsf{h}(g) \cdot g\Big)^2}_{A_1(q)}\Big] \\ &= \frac{1}{n_1(q)} \cdot \big[B_1(q) - \frac{1}{n_1(q)} \cdot A_1^2(q)\big], \end{aligned} \tag{11.61}$$

with the quantities

$$\begin{aligned} A_0(q) &= \sum_{g=0}^{q} \mathsf{h}(g) \cdot g, \qquad & B_0(q) &= \sum_{g=0}^{q} \mathsf{h}(g) \cdot g^2, \\ A_1(q) &= \sum_{g=q+1}^{K-1} \mathsf{h}(g) \cdot g, \qquad & B_1(q) &= \sum_{g=q+1}^{K-1} \mathsf{h}(g) \cdot g^2. \end{aligned} \tag{11.62}$$

Furthermore, the values $\sigma_0^2(q)$, $\sigma_1^2(q)$ can be tabulated for every possible q in only two passes over the histogram, using the recurrence relations

$$A_0(q) = \begin{cases} 0 & \text{for } q = 0, \\ A_0(q-1) + \mathsf{h}(q) \cdot q & \text{for } 1 \le q \le K-1, \end{cases} \tag{11.63}$$

$$B_0(q) = \begin{cases} 0 & \text{for } q = 0, \\ B_0(q-1) + \mathsf{h}(q) \cdot q^2 & \text{for } 1 \le q \le K-1, \end{cases} \tag{11.64}$$

$$A_1(q) = \begin{cases} 0 & \text{for } q = K-1, \\ A_1(q+1) + \mathsf{h}(q+1) \cdot (q+1) & \text{for } 0 \le q \le K-2, \end{cases} \tag{11.65}$$

$$B_1(q) = \begin{cases} 0 & \text{for } q = K-1, \\ B_1(q+1) + \mathsf{h}(q+1) \cdot (q+1)^2 & \text{for } 0 \le q \le K-2. \end{cases} \tag{11.66}$$

The complete minimum-error threshold calculation is summarized in Alg. 11.6. First, the tables $\mathsf{S}_0, \mathsf{S}_1$ are set up and initialized with the values of $\sigma_0^2(q), \sigma_1^2(q)$, respectively, for $0 \le q < K$, following the recursive scheme in Eqns. (11.63–11.66). Subsequently, the error value $\mathsf{e}(q)$ is calculated for every possible threshold value q to find the global minimum. Again $\mathsf{e}(q)$ can only be calculated for those values of q, for which both resulting partitions are non-empty (i.e., with $n_0(q), n_1(q) > 0$). Note that, in lines 27 and 37 of Alg. 11.6, a small constant ($\frac{1}{12}$) is added to the variance to avoid zero values when the corresponding class population is homogeneous (i.e., only

contains a single intensity value).[10] This ensures that the algorithm works properly on images with only two distinct gray values. The algorithm computes the optimal threshold by performing three passes over the histogram (two for initializing the tables and one for finding the minimum); it thus has the same time complexity of $\mathcal{O}(K)$ as the algorithms described before.

Figure 11.6 shows the results of minimum-error thresholding on our set of test images. It also shows the fitted pair of Gaussian distributions for the background and the foreground pixels, respectively, for the optimal threshold as well as the graphs of the error function $\mathsf{e}(q)$, which is minimized over all threshold values q. Obviously the error function is quite flat in certain cases, indicating that similar scores are obtained for a wide range of threshold values and the optimal threshold is not very distinct. We can also see that the estimate is quite accurate in case of the synthetic test image in Fig. 11.6(d), which is actually generated as a mixture of two Gaussians (with parameters $\mu_0 = 80$, $\mu_1 = 170$ and $\sigma_0 = \sigma_1 = 20$). Note that the histograms in Fig. 11.6 have been properly normalized (to constant area) to illustrate the curves of the Gaussians, that is, properly scaled by their prior probabilities (P_0, P_1), while the original histograms are scaled with respect to their maximum values.

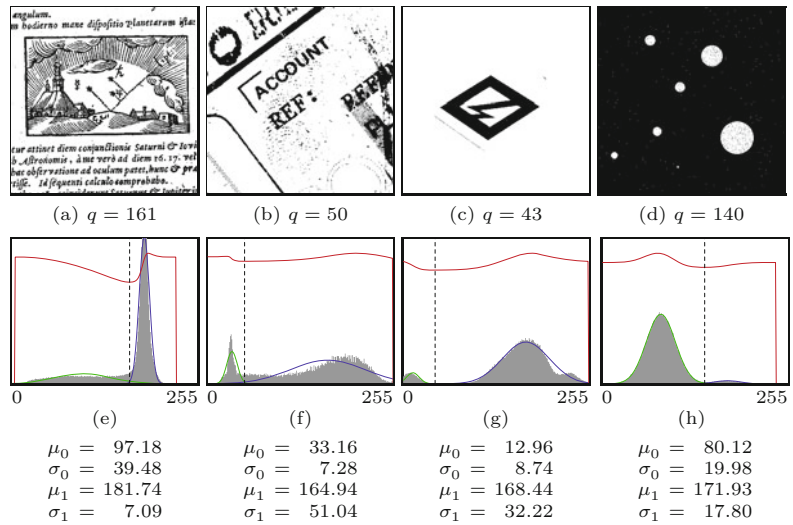


Fig. 11.6 Results from minimum-error thresholding. Calculated threshold values q and resulting binary images (a–d). The green and blue graphs in (e–h) show the fitted Gaussian background and foreground distributions $\mathcal{N}_0 = (\mu_0, \sigma_0)$ and $\mathcal{N}_1 = (\mu_1, \sigma_1)$, respectively. The red graph corresponds to the error quantity $\mathsf{e}(q)$ for varying threshold values $q = 0, \ldots, 255$ (see Eqn. (11.57)). The optimal threshold $q_{\min}$ is located at the *minimum* of $\mathsf{e}(q)$ (dashed vertical line). The estimated parameters of the background/foreground Gaussians are listed at the bottom.

A minor theoretical problem with the minimum error technique is that the parameters of the Gaussian distributions are always estimated from *truncated* samples. This means that, for any threshold q, only the intensity values smaller than q are used to estimate the parameters of the background distribution, and only the intensities greater than q contribute to the foreground parameters. In practice, this problem is of minor importance, since the distributions are typically not strictly Gaussian either.

[10] This is explained by the fact that each histogram bin $\mathsf{h}(i)$ represents intensities in the continuous range $[i \pm 0.5]$ and the variance of uniformly distributed values in the unit interval is $\frac{1}{12}$.

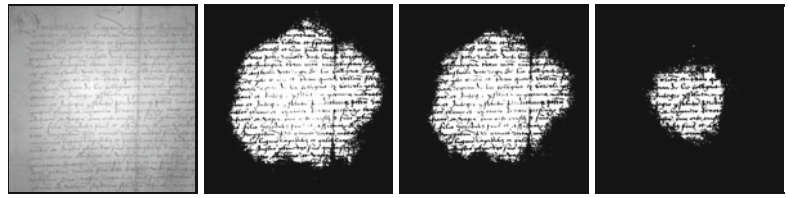
Alg. 11.6 Minimum error threshold selection based on a Gaussian mixture model (after [116]). Tables S_0, S_1 are intialized with values $\sigma_0^2(q)$ and $\sigma_1^2(q)$, respectively (see Eqns. (11.60)–(11.61)), for all possible threshold values $q = 0, \ldots, K-1$. N is the number of image pixels. Initially (outside the **for**-loop), the threshold q is assumed to be -1, which corresponds to the background class being empty ($n_0 = 0$) and all pixels assigned to the foreground class ($n_1 = N$). The **for**-loop (lines 8–16) examines each possible threshold $q = 0, \ldots, K-2$. The optimal threshold is returned, or -1 if no valid threshold was found.

1: **MinimumErrorThreshold**(h)
Input: $\mathsf{h} : [0, K-1] \mapsto \mathbb{N}$, a grayscale histogram. Returns the optimal threshold value or -1 if no threshold is found.
2: $K \leftarrow \mathsf{Size}(\mathsf{h})$
3: $(\mathsf{S}_0, \mathsf{S}_1, N) \leftarrow \mathsf{MakeSigmaTables}(\mathsf{h}, K)$
4: $n_0 \leftarrow 0$
5: $q_{\min} \leftarrow -1$
6: $\mathsf{e}_{\min} \leftarrow \infty$
7: **for** $q \leftarrow 0, \cdots, K-2$ **do** ▷ evaluate all possible thresholds q
8: $n_0 \leftarrow n_0 + \mathsf{h}(q)$ ▷ background population
9: $n_1 \leftarrow N - n_0$ ▷ foreground population
10: **if** $(n_0 > 0) \wedge (n_1 > 0)$ **then**
11: $P_0 \leftarrow n_0 / N$ ▷ prior probability of $\mathcal{C}_0$
12: $P_1 \leftarrow n_1 / N$ ▷ prior probability of $\mathcal{C}_1$
13: $\mathsf{e} \leftarrow P_0 \cdot \ln(\mathsf{S}_0(q)) + P_1 \cdot \ln(\mathsf{S}_1(q)) - 2 \cdot (P_0 \cdot \ln(P_0) + P_1 \cdot \ln(P_1))$ ▷ Eq. 11.57
14: **if** $\mathsf{e} < \mathsf{e}_{\min}$ **then** ▷ minimize error for q
15: $\mathsf{e}_{\min} \leftarrow \mathsf{e}$
16: $q_{\min} \leftarrow q$
17: **return** $q_{\min}$

18: **MakeSigmaTables**(h, K)
19: Create maps $\mathsf{S}_0, \mathsf{S}_1 : [0, K-1] \mapsto \mathbb{R}$
20: $n_0 \leftarrow 0$
21: $A_0 \leftarrow 0$
22: $B_0 \leftarrow 0$
23: **for** $q \leftarrow 0, \cdots, K-1$ **do** ▷ tabulate $\sigma_0^2(q)$
24: $n_0 \leftarrow n_0 + \mathsf{h}(q)$
25: $A_0 \leftarrow A_0 + \mathsf{h}(q) \cdot q$ ▷ Eq. 11.63
26: $B_0 \leftarrow B_0 + \mathsf{h}(q) \cdot q^2$ ▷ Eq. 11.64
27: $\mathsf{S}_0(q) \leftarrow \begin{cases} \frac{1}{12} + (B_0 - \mathsf{A}_0^2/n_0)/n_0 & \text{for } n_0 > 0 \\ 0 & \text{otherwise} \end{cases}$ ▷ Eq. 11.60
28: $N \leftarrow n_0$
29: $n_1 \leftarrow 0$
30: $A_1 \leftarrow 0$
31: $B_1 \leftarrow 0$
32: $\mathsf{S}_1(K-1) \leftarrow 0$
33: **for** $q \leftarrow K-2, \cdots, 0$ **do** ▷ tabulate $\sigma_1^2(q)$
34: $n_1 \leftarrow n_1 + \mathsf{h}(q+1)$
35: $A_1 \leftarrow A_1 + \mathsf{h}(q+1) \cdot (q+1)$ ▷ Eq. 11.65
36: $B_1 \leftarrow B_1 + \mathsf{h}(q+1) \cdot (q+1)^2$ ▷ Eq. 11.66
37: $\mathsf{S}_1(q) \leftarrow \begin{cases} \frac{1}{12} + (B_1 - \mathsf{A}_1^2/n_1)/n_1 & \text{for } n_1 > 0 \\ 0 & \text{otherwise} \end{cases}$ ▷ Eq. 11.61
38: **return** $(\mathsf{S}_0, \mathsf{S}_1, N)$

11.2 Local Adaptive Thresholding

In many situations, a fixed threshold is not appropriate to classify the pixels in the entire image, for example, when confronted with stained backgrounds or uneven lighting or exposure. Figure 11.7 shows a typical, unevenly exposed document image and the results obtained with some global thresholding methods described in the previous sections.



(a) Original (b) Otsu (c) Max. entropy (d) Min. error

Fig. 11.7 Global thresholding methods fail under uneven lighting or exposure. Original image (a), results from global thresholding with various methods described above (b–d).

Instead of using a single threshold value for the whole image, adaptive thresholding specifies a *varying* threshold value $Q(u, v)$ for each image position that is used to classify the corresponding pixel $I(u, v)$ in the same way as described in Eqn. (11.1) for a global threshold. The following approaches differ only with regard to how the threshold "surface" Q is derived from the input image.

11.2.1 Bernsen's Method

The method proposed by Bernsen [23] specifies a dynamic threshold for each image position (u, v), based on the minimum and maximum intensity found in a local neighborhood $R(u, v)$. If

$$I_{\min}(u, v) = \min_{\substack{(i,j)\in \\ R(u,v)}} I(i, j), \tag{11.67}$$

$$I_{\max}(u, v) = \max_{\substack{(i,j)\in \\ R(u,v)}} I(i, j) \tag{11.68}$$

are the minimum and maximum intensity values within a fixed-size neighborhood region R centered at position (u, v), the space-varying threshold is simply calculated as the *mid-range* value

$$Q(u, v) = \frac{I_{\min}(u, v) + I_{\max}(u, v)}{2}. \tag{11.69}$$

This is done as long as the local contrast $c(u, v) = I_{\max}(u, v) - I_{\min}(u, v)$ is above some predefined limit $c_{\min}$. If $c(u, v) < c_{\min}$, the pixels in the corresponding image region are assumed to belong to a single class and are (by default) assigned to the background.

The whole process is summarized in Alg. 11.7. Note that the meaning of "background" in terms of intensity levels depends on the application. For example, in astronomy, the image background is usually darker than the objects of interest. In typical OCR applications, however, the background (paper) is brighter than the foreground objects (print). The main function provides a control parameter bg to select the proper default threshold $\bar{q}$, which is set to K in case of a dark background ($bg = \mathsf{dark}$) and to 0 for a bright background ($bg = \mathsf{bright}$). The support region R may be square or circular, typically with a radius $r = 15$. The choice of the minimum contrast limit $c_{\min}$ depends on the type of imagery and the noise level ($c_{\min} = 15$ is a suitable value to start with).

Figure 11.8 shows the results of Bernsen's method on the uneven test image used in Fig. 11.7 for different settings of the region's radius r. Due to the nonlinear min- and max-operation, the resulting

Alg. 11.7 Adaptive thresholding using local contrast (after Bernsen [23]). The argument to *bg* should be set to dark if the image background is darker than the structures of interest, and to bright if the background is brighter than the objects.

1: **BernsenThreshold**$(I, r, c_{\min}, bg)$
Input: I, intensity image of size $M \times N$; r, radius of support region; $c_{\min}$, minimum contrast; bg, background type (dark or bright). Returns a map with an individual threshold value for each image position.
2: $(M, N) \leftarrow \text{Size}(I)$
3: Create map $Q : M \times N \mapsto \mathbb{R}$
4: $\bar{q} \leftarrow \begin{cases} K & \text{if } bg = \text{dark} \\ 0 & \text{if } bg = \text{bright} \end{cases}$
5: **for all** image coordinates $(u, v) \in M \times N$ **do**
6: $R \leftarrow \text{MakeCircularRegion}(u, v, r)$
7: $I_{\min} \leftarrow \min_{(i,j)\in R} I(i, j)$
8: $I_{\max} \leftarrow \max_{(i,j)\in R} I(i, j)$
9: $c \leftarrow I_{\max} - I_{\min}$
10: $Q(u, v) \leftarrow \begin{cases} (I_{\min} + I_{\max})/2 & \text{if } c \geq c_{\min} \\ \bar{q} & \text{otherwise} \end{cases}$
11: **return** Q

12: **MakeCircularRegion**(u, v, r)
Returns the set of pixel coordinates within the circle of radius r, centered at (u, v)
13: **return** $\{(i, j) \in \mathbb{Z}^2 \mid (u - i)^2 + (v - j)^2 \leq r^2\}$

threshold surface is not smooth. The minimum contrast is set to $c_{\min} = 15$, which is too low to avoid thresholding low-contrast noise visible along the left image margin. By increasing the minimum contrast $c_{\min}$, more neighborhoods are considered "flat" and thus ignored, that is, classified as background. This is demonstrated in Fig. 11.9. While larger values of $c_{\min}$ effectively eliminate low-contrast noise, relevant structures are also lost, which illustrates the difficulty of finding a suitable global value for $c_{\min}$. Additional examples, using the test images previously used for global thresholding, are shown in Fig. 11.10.

What Alg. 11.7 describes formally can be implemented quite efficiently, noting that the calculation of local minima and maxima over a sliding window (lines 6–8) corresponds to a simple nonlinear filter operation (see Ch. 5, Sec. 5.4). To perform these calculations, we can use a *minimum* and *maximum* filter with radius r, as provided by virtually every image processing environment. For example, the Java implementation of the Bernsen thresholder in Prog. 11.1 uses ImageJ's built-in `RankFilters` class for this purpose. The complete implementation can be found on the book's website (see Sec. 11.3 for additional details on the corresponding API).

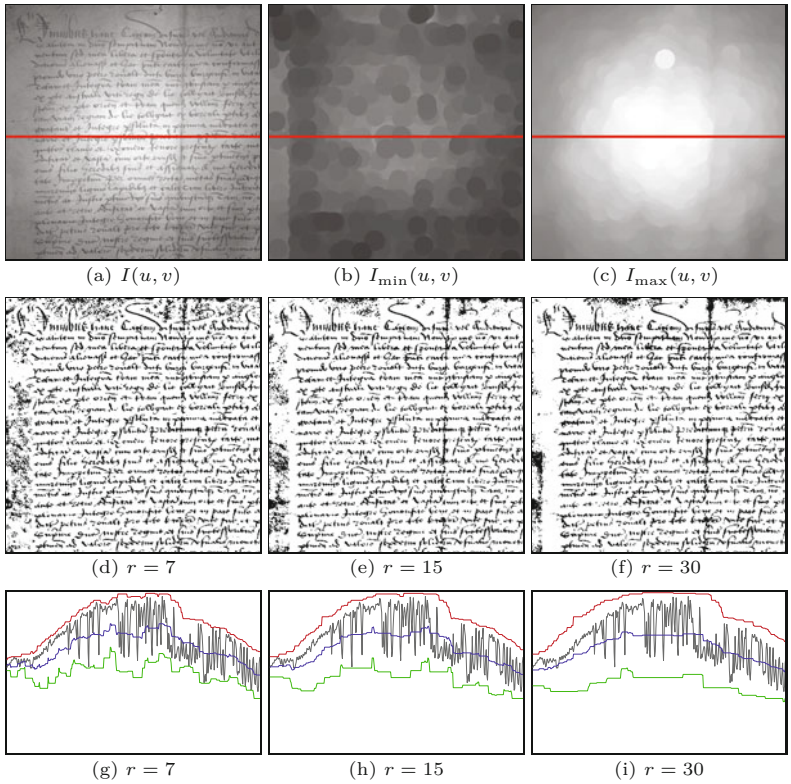
11.2.2 Niblack's Method

In this approach, originally presented in [172, Sec. 5.1], the threshold $Q(u, v)$ is varied across the image as a function of the local intensity average $\mu_R(u, v)$ and standard deviation[11] $\sigma_R(u, v)$ in the form

[11] The standard deviation σ is the square root of the variance σ^2.

Prog. 11.1 Bernsen's thresholder (ImageJ plugin implementation of Alg. 11.7). Note the use of ImageJ's `RankFilters` class (lines 30–32) for calculating the local minimum (`Imin`) and maximum (`Imax`) maps inside the `getThreshold()` method. The resulting threshold surface $Q(u, v)$ is returned as an 8-bit image of type `ByteProcessor`.

```
1  package imagingbook.pub.threshold.adaptive;
2  import ij.plugin.filter.RankFilters;
3  import ij.process.ByteProcessor;
4  import imagingbook.pub.threshold.BackgroundMode;
5
6  public class BernsenThresholder extends AdaptiveThresholder
       {
7
8    public static class Parameters {
9      public int radius = 15;
10     public int cmin = 15;
11     public BackgroundMode bgMode = BackgroundMode.DARK;
12   }
13
14   private final Parameters params;
15
16   public BernsenThresholder() {
17     this.params = new Parameters();
18   }
19
20   public BernsenThresholder(Parameters params) {
21     this.params = params;
22   }
23
24   public ByteProcessor getThreshold(ByteProcessor I) {
25     int M = I.getWidth();
26     int N = I.getHeight();
27     ByteProcessor Imin = (ByteProcessor) I.duplicate();
28     ByteProcessor Imax = (ByteProcessor) I.duplicate();
29
30     RankFilters rf = new RankFilters();
31     rf.rank(Imin,params.radius,RankFilters.MIN); // I_min(u,v)
32     rf.rank(Imax,params.radius,RankFilters.MAX); // I_max(u,v)
33
34     int q = (params.bgMode == BackgroundMode.DARK) ?
35               256 : 0;
36     ByteProcessor Q = new ByteProcessor(M, N);  // Q(u,v)
37
38     for (int v = 0; v < N; v++) {
39       for (int u = 0; u < M; u++) {
40         int gMin = Imin.get(u, v);
41         int gMax = Imax.get(u, v);
42         int c = gMax - gMin;
43         if (c >= params.cmin)
44           Q.set(u, v, (gMin + gMax) / 2);
45         else
46           Q.set(u, v, q);
47       }
48     }
49     return Q;
50   }
51 }
```



(a) $I(u,v)$ (b) $I_{\min}(u,v)$ (c) $I_{\max}(u,v)$

(d) $r = 7$ (e) $r = 15$ (f) $r = 30$

(g) $r = 7$ (h) $r = 15$ (i) $r = 30$

Fig. 11.8
Adaptive thresholding using Bernsen's method. Original image (a), local minimum (b), and maximum (c). The center row shows the binarized images for different settings of r (d–f). The corresponding curves in (g–i) show the original intensity (gray), local minimum (green), maximum (red), and the actual threshold (blue) along the horizontal line marked in (a–c). The region radius r is 15 pixels, the minimum contrast $c_{\min}$ is 15 intensity units.

(a) $c_{\min} = 15$ (b) $c_{\min} = 30$ (c) $c_{\min} = 60$

Fig. 11.9
Adaptive thresholding using Bernsen's method with different settings of $c_{\min}$. Binarized images (top row) and threshold surface $Q(u,v)$ (bottom row). Black areas in the threshold functions indicate that the local contrast is below $c_{\min}$; the corresponding pixels are classified as background (white in this case).

$$Q(u,v) := \mu_R(u,v) + \kappa \cdot \sigma_R(u,v). \tag{11.70}$$

Thus the local threshold $Q(u,v)$ is determined by adding a constant portion ($\kappa \geq 0$) of the local standard deviation σ_R to the local mean μ_R. μ_R and σ_R are calculated over a square support region R centered at (u,v). The size (radius) of the averaging region R should be as large as possible, at least larger than the size of the structures to be detected, but small enough to capture the variations (unevenness)

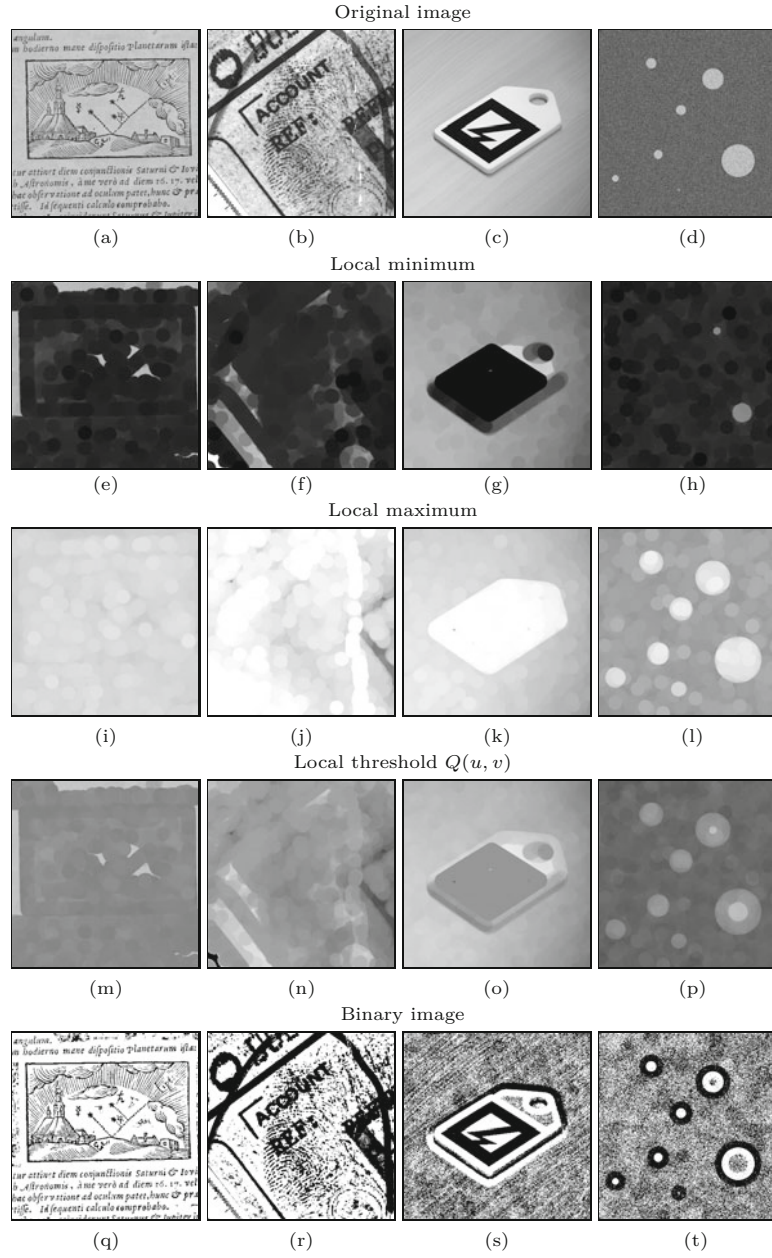


Fig. 11.10
Additional examples for Bernsen's method. Original images (a–d), local minimum $I_{\min}$ (e–h), maximum $I_{\max}$ (i–l), and threshold map Q (m–p); results after thresholding the images (q–t). Settings are $r = 15$, $c_{\min} = 15$. A bright background is assumed for all images ($bg =$ bright), except for image (d).

of the background. A size of 31×31 pixels (or radius $r = 15$) is suggested in [172] and $\kappa = 0.18$, though the latter does not seem to be critical.

One problem is that, for small values of σ_R (as obtained in "flat" image regions of approximately constant intensity), the threshold will be close to the local average, which makes the segmentation quite sensitive to low-amplitude noise ("ghosting"). A simple improvement is to secure a minimum distance from the mean by adding a constant offset d, that is, replacing Eqn. (11.70) by

$$Q(u, v) := \mu_R(u, v) + \kappa \cdot \sigma_R(u, v) + d, \tag{11.71}$$

with $d \geq 0$, in the range $2, \ldots, 20$ for typical 8-bit images.

The original formulation (Eqn. (11.70)) is aimed at situations where the foreground structures are *brighter* than the background (Fig. 11.11(a)) but does not work if the images are set up the other way round (Fig. 11.11(b)). In the case that the structures of interest are *darker* than the background (as, e.g., in typical OCR applications), one could either work with inverted images or modify the calculation of the threshold to

$$Q(u,v) := \begin{cases} \mu_R(u,v) + (\kappa \cdot \sigma_R(u,v) + d) & \text{for } \textit{dark} \text{ BG}, \\ \mu_R(u,v) - (\kappa \cdot \sigma_R(u,v) + d) & \text{for } \textit{bright} \text{ BG}. \end{cases} \tag{11.72}$$

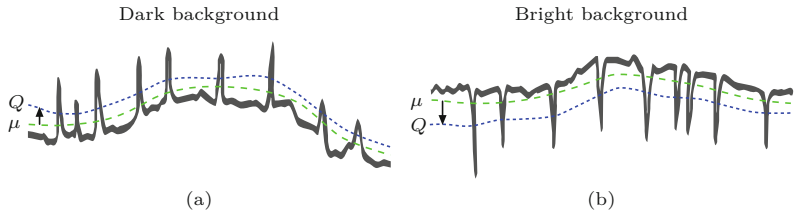


Fig. 11.11
Adaptive thresholding based on average local intensity. The illustration shows a line profile as typically found in document imaging. The space-variant threshold Q (dotted blue line) is chosen as the local average μ_R (dashed green line) offset by a multiple of the local intensity variation σ_R. The offset is chosen to be *positive* for images with a dark background and bright structures (a) and *negative* if the background is brighter than the contained structures of interest (b).

The modified procedure is detailed in Alg. 11.8. The example in Fig. 11.12 shows results obtained with this method on an image with a bright background containing dark structures, for $\kappa = 0.3$ and varying settings of d. Note that setting $d = 0$ (Fig. 11.12(d, g)) corresponds to Niblack's original method. For these examples, a circular window of radius $r = 15$ was used to compute the local mean $\mu_R(u,v)$ and variance $\sigma_R(u,v)$. Additional examples are shown in Fig. 11.13. Note that the selected radius r is obviously too small for the structures in the images in Fig. 11.13(c, d), which are thus not segmented cleanly. Better results can be expected with a larger radius.

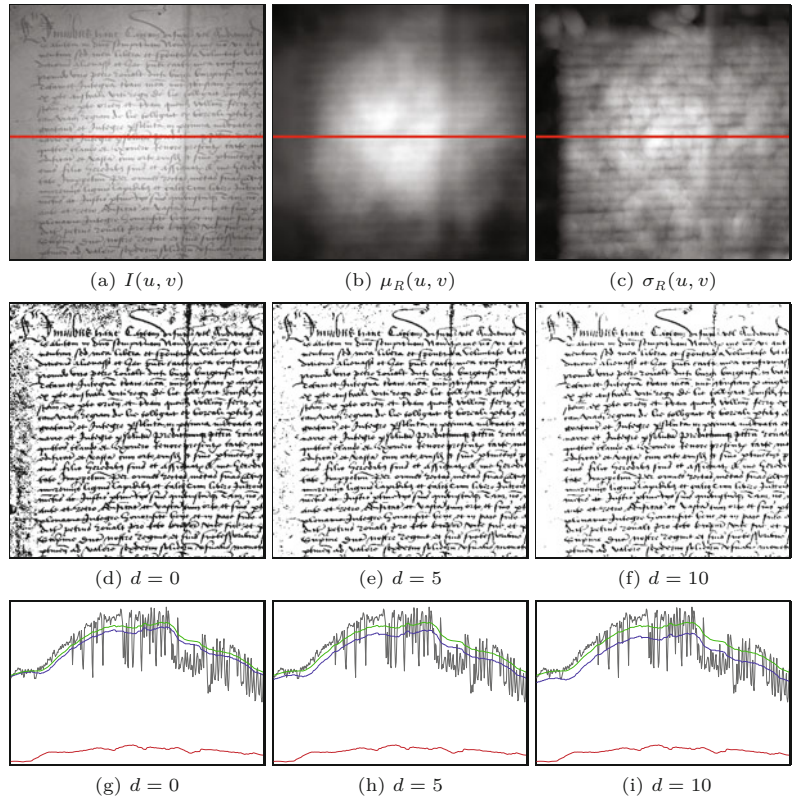
With the intent to improve upon Niblack's method, particularly for thresholding deteriorated text images, Sauvola and Pietikäinen [207] proposed setting the threshold to

$$Q(u,v) := \begin{cases} \mu_R(u,v) \cdot \left[1 - \kappa \cdot \left(\frac{\sigma_R(u,v)}{\sigma_{\max}} - 1\right)\right] & \text{for } \textit{dark} \text{ BG}, \\ \mu_R(u,v) \cdot \left[1 + \kappa \cdot \left(\frac{\sigma_R(u,v)}{\sigma_{\max}} - 1\right)\right] & \text{for } \textit{bright} \text{ BG}, \end{cases} \tag{11.73}$$

with $\kappa = 0.5$ and $\sigma_{\max} = 128$ (the "dynamic range of the standard deviation" for 8-bit images) as suggested parameter values. In this approach, the offset between the threshold and the local average not only depends on the local variation σ_R (as in Eqn. (11.70)), but also on the magnitude of the local *mean* μ_R! Thus, changes in absolute brightness lead to modified relative threshold values, even when the image contrast remains constant. Though this technique is frequently referenced in the literature, it appears questionable if this behavior is generally desirable.

Calculating local mean and variance

Algorithm 11.8 shows the principle operation of Niblack's method and also illustrates how to efficiently calculate the local average and



(a) $I(u,v)$ (b) $\mu_R(u,v)$ (c) $\sigma_R(u,v)$

(d) $d = 0$ (e) $d = 5$ (f) $d = 10$

(g) $d = 0$ (h) $d = 5$ (i) $d = 10$

Fig. 11.12
Adaptive thresholding using Niblack's method (with $r = 15$, $\kappa = 0.3$). Original image (a), local mean μ_R (b), and standard deviation σ_R (c). The result for $d = 0$ in (d) corresponds to Niblack's original formulation. Increasing the value of d reduces the amount of clutter in regions with low variance (e, f). The curves in (g–i) show the local intensity (gray), mean (green), variance (red), and the actual threshold (blue) along the horizontal line marked in (a–c).

variance. Given the image I and the averaging region R, we can use the shortcut suggested in Eqn. (3.12) to obtain these quantities as

$$\mu_R = \frac{1}{n} \cdot A \qquad \text{and} \qquad \sigma_R^2 = \frac{1}{n} \cdot \left(B - \frac{1}{n} \cdot A^2\right), \tag{11.74}$$

with

$$A = \sum_{(i,j)\in R} I(i,j), \qquad B = \sum_{(i,j)\in R} I^2(i,j), \qquad n = |R|. \tag{11.75}$$

Procedure GetLocalMeanAndVariance() in Alg. 11.8 shows this calculation in full detail.

When computing the local average and variance, attention must be paid to the situation at the image borders, as illustrated in Fig. 11.14. Two approaches are frequently used. In the first approach (following the common practice for implementing filter operations), all outside pixel values are replaced by the closest inside pixel, which is always a border pixel. Thus the border pixel values are effectively replicated outside the image boundaries and thus these pixels have a strong influence on the local results. The second approach is to perform the calculation of the average and variance on only those image pixels that are actually covered by the support region. In this case, the number of pixels (N) is reduced at the image borders to about 1/4 of the full region size.

Although the calculation of the local mean and variance outlined by function GetLocalMeanAndVariance() in Alg. 11.8 is definitely more

Alg. 11.8 Adaptive thresholding using local mean and variance (modified version of Niblack's method [172]). The argument to bg should be dark if the image background is darker than the structures of interest, bright if the background is brighter than the objects. The function MakeCircularRegion() is defined in Alg. 11.7.

1: **NiblackThreshold**(I, r, κ, d, bg)
Input: I, intensity image of size $M \times N$; r, radius of support region; κ, variance control parameter; d, minimum offset; $bg \in \{\text{dark}, \text{bright}\}$, background type. Returns a map with an individual threshold value for each image position.
2: $(M, N) \leftarrow \text{Size}(I)$
3: Create map $Q : M \times N \mapsto \mathbb{R}$
4: **for all** image coordinates $(u, v) \in M \times N$ **do**
Define a support region of radius r, centered at (u, v):
5: $(\mu, \sigma^2) \leftarrow \text{GetLocalMeanAndVariance}(I, u, v, r)$
6: $\sigma \leftarrow \sqrt{\sigma^2}$ ▷ local std. deviation σ_R
7: $Q(u, v) \leftarrow \begin{cases} \mu + (\kappa \cdot \sigma + d) & \text{if } bg = \text{dark} \\ \mu - (\kappa \cdot \sigma + d) & \text{if } bg = \text{bright} \end{cases}$ ▷ Eq. 11.72
8: **return** Q

9: **GetLocalMeanAndVariance**(I, u, v, r)
Returns the local mean and variance of the image pixels $I(i, j)$ within the disk-shaped region with radius r around position (u, v).
10: $R \leftarrow \text{MakeCircularRegion}(u, v, r)$ ▷ see Alg. 11.7
11: $n \leftarrow 0$
12: $A \leftarrow 0$
13: $B \leftarrow 0$
14: **for all** $(i, j) \in R$ **do**
15: $n \leftarrow n + 1$
16: $A \leftarrow A + I(i, j)$
17: $B \leftarrow B + I^2(i, j)$
18: $\mu \leftarrow \frac{1}{n} \cdot A$
19: $\sigma^2 \leftarrow \frac{1}{n} \cdot (B - \frac{1}{n} \cdot A^2)$
20: **return** (μ, σ^2)

efficient than a brute-force approach, additional optimizations are possible. Most image processing environments have suitable routines already built in. With ImageJ, for example, we can again use the `RankFilters` class (as with the *min*- and *max*-filters in the *Bernsen* approach, see Sec. 11.2.1). Instead of performing the computation for each pixel individually, the following ImageJ code segment uses predefined filters to compute two separate images `Imean` (μ_R) and `Ivar` (σ_R^2) containing the local mean and variance values, respectively, with a disk-shaped support region of radius 15:

```
ByteProcessor I; // original image I(u,v)
int radius = 15;

FloatProcessor Imean = I.convertToFloatProcessor();
FloatProcessor Ivar =  Imean.duplicate();

RankFilters rf = new RankFilters();
rf.rank(Imean, radius, RankFilters.MEAN);       // μ_R(u,v)
rf.rank(Ivar, radius, RankFilters.VARIANCE);    // σ²_R(u,v)
...
```

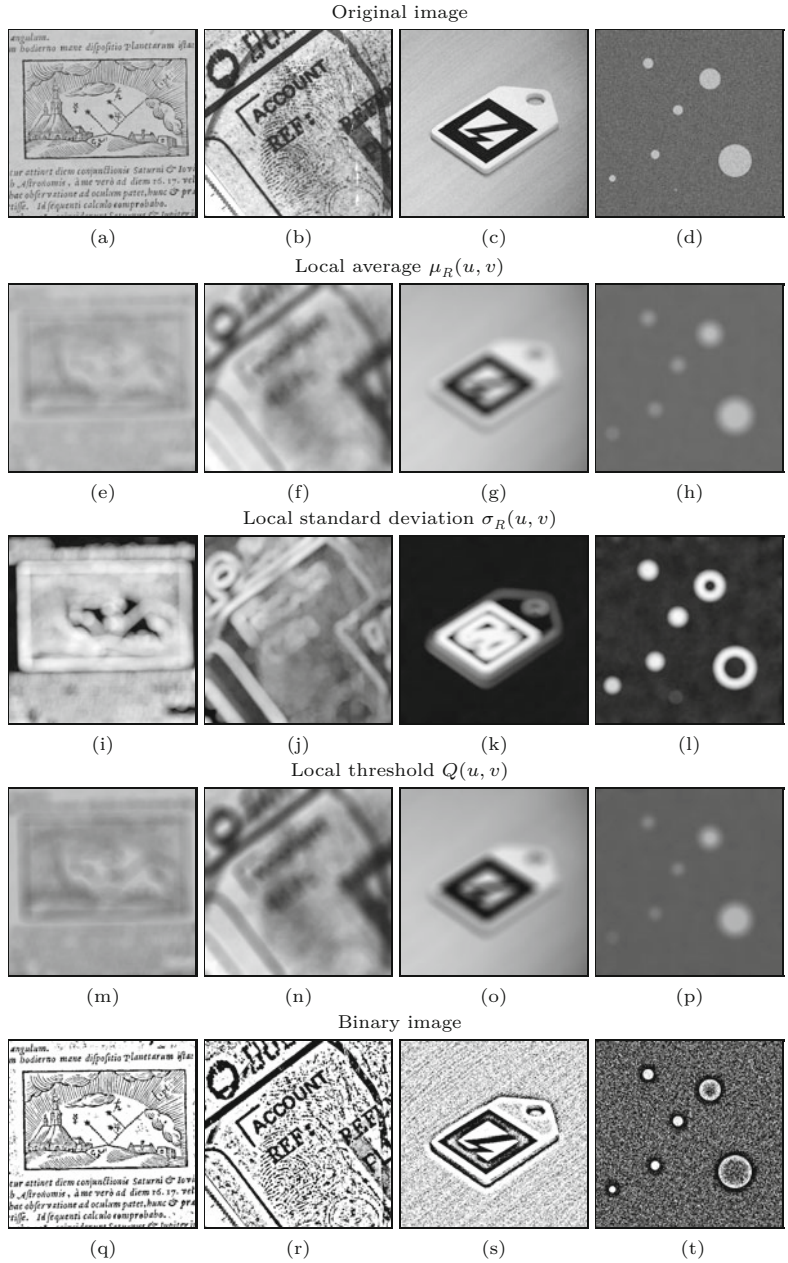


Fig. 11.13 Additional examples for thresholding with Niblack's method using a disk-shaped support region of radius $r = 15$. Original images (a–d), local mean μ_R (e–h), std. deviation σ_R (i–l), and threshold Q (m–p); results after thresholding the images (q–t). The background is assumed to be brighter than the structures of interest, except for image (d), which has a dark background. Settings are $\kappa = 0.3$, $d = 5$.

See Sec. 11.3 and the online code for additional implementation details. Note that the filter methods implemented in `RankFilters` perform replication of border pixels as the border handling strategy, as discussed earlier.

Local average and variance with Gaussian kernels

The purpose of taking the local average is to smooth the image to obtain an estimate of the varying background intensity. In case of a square or circular region, this is equivalent to convolving the image with a box- or disk-shaped kernel, respectively. Kernels of this

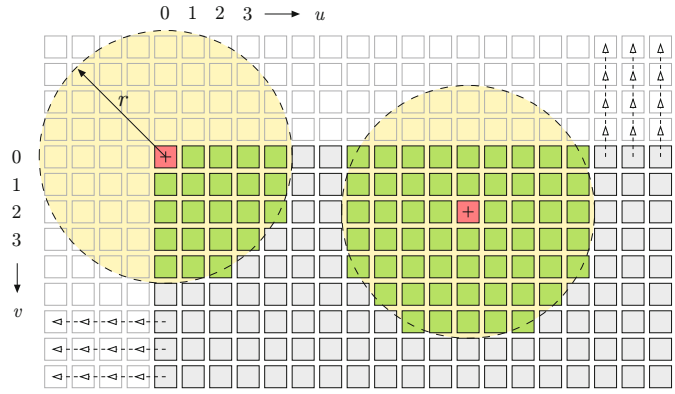


Fig. 11.14
Calculating local statistics at image boundaries. The illustration shows a disk-shaped support region with radius r, placed at the image border. Pixel values outside the image can be replaced ("filled-in") by the closest border pixel, as is common in many filter operations. Alternatively, the calculation of the local statistics can be confined to include only those pixels inside the image that are actually covered by the support region. At any border pixel, the number of covered elements (N) is still more than $\approx 1/4$ of the full region size. In this particular case, the circular region covers a maximum of $N = 69$ pixels when fully embedded and $N = 22$ when positioned at an image corner.

type, however, are not well suited for image smoothing, because they create strong ringing and truncating effects, as demonstrated in Fig. 11.15. Moreover, convolution with a box-shaped (rectangular) kernel is a non-isotropic operation, that is, the results are orientation-dependent. From this perspective alone it seems appropriate to consider other smoothing kernels, Gaussian kernels in particular.

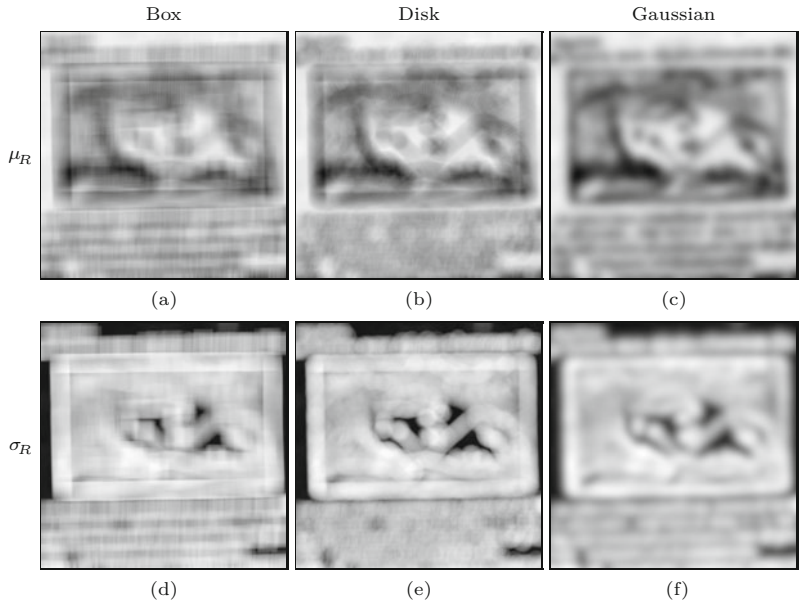


Fig. 11.15
Local average (a–c) and variance (d–f) obtained with different smoothing kernels. 31×31 box filter (a, d), disk filter with radius $r = 15$ (b, e), Gaussian kernel with $\sigma = 0.6 \cdot 15 = 9.0$ (c, f). Both the box and disk filter show strong truncation effects (ringing), the box filter is also highly non-isotropic. All images are contrast-enhanced for better visibility.

Using a Gaussian kernel H^{G} for smoothing is equivalent to calculating a weighted average of the corresponding image pixels, with the weights being the coefficients of the kernel. Thus calculating this weighted local average can be expressed by

$$\mu_{\mathrm{G}}(u,v) = \frac{1}{\Sigma H^{\mathrm{G}}} \cdot \big(I * H^{\mathrm{G}}\big)(u,v), \tag{11.76}$$

where ΣH^{G} is the sum of the coefficients in the kernel H^{G} and $*$ denotes the linear convolution operator.[12] Analogously, there is also

[12] See Chapter 5, Sec. 5.3.1.

a weighted *variance* σ_G^2 which can be calculated jointly with the local average μ_G (as in Eqn. (11.74)) in the form

$$\mu_G(u,v) = \frac{1}{\Sigma H^G} \cdot A_G(u,v), \tag{11.77}$$

$$\sigma_G^2(u,v) = \frac{1}{\Sigma H^G} \cdot \big(B_G(u,v) - \frac{1}{\Sigma H^G} \cdot A_G^2(u,v)\big), \tag{11.78}$$

with $A_G = I * H^G$ and $B_G = I^2 * H^G$.

Thus all we need is two filter operations, one applied to the original image ($I * H^G$) and another applied to the squared image ($I^2 * H^G$), using the same 2D Gaussian kernel H^G (or any other suitable smoothing kernel). If the kernel H^G is *normalized* (i.e., $\Sigma H^G = 1$), Eqns. (11.77)–(11.78) reduce to

$$\mu_G(u,v) = A_G(u,v), \tag{11.79}$$

$$\sigma_G^2(u,v) = B_G(u,v) - A_G^2(u,v), \tag{11.80}$$

with A_G, B_G as defined already.

This suggests a very simple process for computing the local average and variance by Gaussian filtering, as summarized in Alg. 11.9. The width (standard deviation σ) of the Gaussian kernel is set to 0.6 times the radius r of the corresponding disk filter to produce a similar effect as Alg. 11.8. The Gaussian approach has two advantages: First, the Gaussian makes a much superior low-pass filter, compared to the box or disk kernels. Second, the 2D Gaussian is (unlike the circular disk kernel) separable in the x- and y-direction, which permits a very efficient implementation of the 2D filter using only a pair of 1D convolutions (see Ch. 5, Sec. 5.2).

For practical calculation, A_G, B_G can be represented as (floating-point) images, and most modern image-processing environments provide efficient (multi-scale) implementations of Gaussian filters with large-size kernels. In ImageJ, fast Gaussian filtering is implemented by the class `GaussianBlur` with the public methods `blur()`, `blurGaussian()`, and `blurFloat()`, which all use normalized filter kernels by default. Programs 11.2–11.3 show the complete ImageJ implementation of Niblack's thresholder using Gaussian smoothing kernels.

11.3 Java Implementation

All thresholding methods described in this chapter have been implemented as part of the `imagingbook` library that is available with full source code at the book's website. The top class in this library[13] is `Thresholder` with the sub-classes `GlobalThresholder` and `AdaptiveThresholder` for the methods described in Secs. 11.1 and 11.2, respectively. Class `Thresholder` itself is abstract and only defines a set of (non-public) utility methods for histogram analysis.

[13] Package `imagingbook.pub.threshold`.

Alg. 11.9 Adaptive thresholding using Gaussian averaging (extended from Alg. 11.8). Parameters are the original image I, the radius r of the Gaussian kernel, variance control k, and minimum offset d. The argument to bg should be dark if the image background is darker than the structures of interest, bright if the background is brighter than the objects. The procedure MakeGaussianKernel2D(σ) creates a discrete, normalized 2D Gaussian kernel with standard deviation σ.

AdaptiveThresholdGauss(I, r, κ, d, bg)
Input: I, intensity image of size $M \times N$; r, support region radius; κ, variance control parameter; d, minimum offset; $bg \in \{\mathsf{dark}, \mathsf{bright}\}$, background type.
Returns a map Q of local thresholds for the grayscale image I.

$(M, N) \leftarrow \text{Size}(I)$
Create maps $\mathsf{A}, \mathsf{B}, \mathsf{Q} : M \times N \mapsto \mathbb{R}$
for all image coordinates $(u, v) \in M \times N$ **do**
  $\mathsf{A}(u, v) \leftarrow I(u, v)$
  $\mathsf{B}(u, v) \leftarrow (I(u, v))^2$
$H^{\mathrm{G}} \leftarrow \mathsf{MakeGaussianKernel2D}(0.6 \cdot r)$
$\mathsf{A} \leftarrow \mathsf{A} * H^{\mathrm{G}}$ ▷ filter the original image with H^{G}
$\mathsf{B} \leftarrow \mathsf{B} * H^{\mathrm{G}}$ ▷ filter the squared image with H^{G}
for all image coordinates $(u, v) \in M \times N$ **do**
  $\mu_{\mathrm{G}} \leftarrow \mathsf{A}(u, v)$ ▷ Eq. 11.79
  $\sigma_{\mathrm{G}} \leftarrow \sqrt{\mathsf{B}(u, v) - \mathsf{A}^2(u, v)}$ ▷ Eq. 11.80
  $\mathsf{Q}(u, v) \leftarrow \begin{cases} \mu_{\mathrm{G}} + \big(\kappa \cdot \sigma_{\mathrm{G}} + d\big) & \text{if } bg = \mathsf{dark} \\ \mu_{\mathrm{G}} - \big(\kappa \cdot \sigma_{\mathrm{G}} + d\big) & \text{if } bg = \mathsf{bright} \end{cases}$ ▷ Eq. 11.72
return Q

MakeGaussianKernel2D(σ)
Returns a discrete 2D Gaussian kernel H with std. deviation σ, sized sufficiently large to avoid truncation effects.
$r \leftarrow \max(1, \lceil 3.5 \cdot \sigma \rceil)$ ▷ size the kernel sufficiently large
Create map $H : [-r, r]^2 \mapsto \mathbb{R}$
$s \leftarrow 0$
for $x \leftarrow -r, \ldots, r$ **do**
  for $y \leftarrow -r, \ldots, r$ **do**
    $H(x, y) \leftarrow e^{-\frac{x^2+y^2}{2 \cdot \sigma^2}}$ ▷ unnormalized 2D Gaussian
    $s \leftarrow s + H(x, y)$
for $x \leftarrow -r, \ldots, r$ **do**
  for $y \leftarrow -r, \ldots, r$ **do**
    $H(x, y) \leftarrow \frac{1}{s} \cdot H(x, y)$ ▷ normalize H
return H

11.3.1 Global Thresholding Methods

The thresholding methods covered in Sec. 11.1 are implemented by the following classes:

- `MeanThresholder`, `MedianThresholder` (Sec. 11.1.2),
- `QuantileThresholder` (Alg. 11.1),
- `IsodataThresholder` (Alg. 11.2–11.3),
- `OtsuThresholder` (Alg. 11.4),
- `MaxEntropyThresholder` (Alg. 11.5), and
- `MinErrorThresholder` (Alg. 11.6).

These are sub-classes of the (abstract) class `GlobalThresholder`. The following example demonstrates the typical use of this method for a given `ByteProcessor` object `I`:

```
...
GlobalThresholder thr = new IsodataThresholder();
int q = thr.getThreshold(I);
```

Prog. 11.2
Niblack's thresholder using Gaussian smoothing kernels (ImageJ implementation of Alg. 11.9, part 1).

```
1  package threshold;
2
3  import ij.plugin.filter.GaussianBlur;
4  import ij.plugin.filter.RankFilters;
5  import ij.process.ByteProcessor;
6  import ij.process.FloatProcessor;
7  import imagingbook.pub.threshold.BackgroundMode;
8
9  public abstract class NiblackThresholder extends
       AdaptiveThresholder {
10
11   // parameters for this thresholder
12   public static class Parameters {
13     public int radius = 15;
14     public double kappa =  0.30;
15     public int dMin = 5;
16     public BackgroundMode bgMode = BackgroundMode.DARK;
17   }
18
19   private final Parameters params;  // parameter object
20
21   protected FloatProcessor Imean;   // = μ_G(u,v)
22   protected FloatProcessor Isigma;  // = σ_G(u,v)
23
24   public ByteProcessor getThreshold(ByteProcessor I) {
25     int w = I.getWidth();
26     int h = I.getHeight();
27
28     makeMeanAndVariance(I, params);
29     ByteProcessor Q = new ByteProcessor(w, h);
30
31     final double kappa = params.kappa;
32     final int dMin = params.dMin;
33     final boolean darkBg =
34         (params.bgMode == BackgroundMode.DARK);
35
36     for (int v = 0; v < h; v++) {
37       for (int u = 0; u < w; u++) {
38         double sigma = Isigma.getf(u, v);
39         double mu = Imean.getf(u, v);
40         double diff = kappa * sigma + dMin;
41         int q = (int)
42             Math.rint((darkBg) ? mu + diff : mu - diff);
43         if (q < 0)   q = 0;
44         if (q > 255) q = 255;
45         Q.set(u, v, q);
46       }
47     }
48     return Q;
49   }
50
51   // continues in Prog. 11.3
```

```
52 // continued from Prog. 11.2
53
54   public static class Gauss extends NiblackThresholder {
55
56     protected void makeMeanAndVariance(ByteProcessor I,
       Parameters params) {
57       int width = I.getWidth();
58       int height = I.getHeight();
59
60       Imean = new FloatProcessor(width,height);
61       Isigma = new FloatProcessor(width,height);
62
63       FloatProcessor A = I.convertToFloatProcessor(); // = I
64       FloatProcessor B = I.convertToFloatProcessor(); // = I
65       B.sqr();     // = I^2
66
67       GaussianBlur gb = new GaussianBlur();
68       double sigma = params.radius * 0.6;
69       gb.blurFloat(A, sigma, sigma, 0.002); // = A
70       gb.blurFloat(B, sigma, sigma, 0.002); // = B
71
72       for (int v = 0; v < height; v++) {
73         for (int u = 0; u < width; u++) {
74           float a = A.getf(u, v);
75           float b = B.getf(u, v);
76           float sigmaG =
77               (float) Math.sqrt(b - a*a); // Eq. 11.80
78           Imean.setf(u, v, a);       // = μ_G(u,v)
79           Isigma.setf(u, v, sigmaG); // = σ_G(u,v)
80         }
81       }
82     }
83   }  // end of inner class NiblackThresholder.Gauss
84 }  // end of class NiblackThresholder
```

Prog. 11.3
Niblack's thresholder using Gaussian smoothing kernels (part 2). The floating-point images `AG` and `BG` correspond to the maps A_G (filtered original image) and B_G (filtered squared image) in Alg. 11.9. An instance of the ImageJ class `GaussianBlur` is created in line 67 and subsequently used to filter both images in lines 69–70. The last argument to the ImageJ method `blurFloat` (0.002) specifies the accuracy of the Gaussian kernel.

```
if (q > 0) I.threshold(q);
else ...
```

Here `threshold()` is the built-in ImageJ's method defined by class `ImageProcessor`.

11.3.2 Adaptive Thresholding

The techniques described in Sec. 11.2 are implemented by the following classes:

- `BernsenThresholder` (Alg. 11.7),
- `NiblackThresholder` (Alg. 11.8, multiple versions), and
- `SauvolaThresholder` (Eqn. (11.73)).

These are sub-classes of the (abstract) class `AdaptiveThresholder`. The following example demonstrates the typical use of these methods for a given `ByteProcessor` object `I`:

```
...
AdaptiveThresholder thr = new BernsenThresholder();
```

```
ByteProcessor Q = thr.getThreshold(I);
thr.threshold(I, Q);
...
```

The 2D threshold surface is represented by the image `Q`; the method `threshold(I, Q)` is defined by class `AdaptiveThresholder`. Alternatively, the same operation can be performed without making `Q` explicit, as demonstrated by the following code segment:

```
...
// Create and set up a parameter object:
Parameters params = new BernsenThresholder.Parameters();
params.radius = 15;
params.cmin = 15;
params.bgMode = BackgroundMode.DARK;

// Create the thresholder:
AdaptiveThresholder thr = new BernsenThresholder(params);

// Perform the threshold operation:
thr.threshold(I);
...
```

This example also shows how to specify a parameter object (`params`) for the instantiation of the thresholder.

11.4 Summary and Further Reading

The intention of this chapter was to give an overview of established methods for automatic image thresholding. A vast body of relevant literature exists, and thus only a fraction of the proposed techniques could be discussed here. For additional approaches and references, several excellent surveys are available, including [86, 178, 204, 231] and [213].

Given the obvious limitations of global techniques, adaptive thresholding methods have received continued interest and are still a focus of ongoing research. Another popular approach is to calculate an adaptive threshold through image decomposition. In this case, the image is partitioned into (possibly overlapping) tiles, an "optimal" threshold is calculated for each tile and the adaptive threshold is obtained by interpolation between adjacent tiles. Another interesting idea, proposed in [260], is to specify a "threshold surface" by sampling the image at specific points that exhibit a high gradient, with the assumption that these points are at transitions between the background and the foreground. From these irregularly spaced point samples, a smooth surface is interpolated that passes through the sample points. Interpolation between these irregularly spaced point samples is done by solving a Laplacian difference equation to obtain a continuous "potential surface". This is accomplished with the so-called "successive over-relaxation" method, which requires about N scans over an image of size $N \times N$ to converge, so its time complexity is an expensive $\mathcal{O}(N^3)$. A more efficient approach was proposed in [26], which uses a hierarchical, multi-scale algorithm for interpolating the threshold surface. Similarly, a quad-tree representation

was used for this purpose in [49]. Another interesting concept is "kriging" [175], which was originally developed for interpolating 2D geological data [190, Ch. 3, Sec. 3.7.4].

In the case of color images, simple thresholding is often applied individually to each color channel and the results are subsequently merged using a suitable logical operation. Transformation to a non-RGB color space (such as HSV or CIELAB) might be helpful for this purpose. For a binarization method aimed specifically at vector-valued images, see [159], for example. Since thresholding can be viewed as a specific form of segmentation, color segmentation methods [50, 53, 85, 216] are also relevant for binarizing color images.

11.5 Exercises

Exercise 11.1. Define a procedure for estimating the minimum and maximum pixel value of an image from its histogram. Threshold the image at the resulting mid-range value (see Eqn. (11.12)). Can anything be said about the size of the resulting partitions?

Exercise 11.2. Define a procedure for estimating the median of an image from its histogram. Threshold the image at the resulting median value (see Eqn. (11.11)) and verify that the foreground and background partitions are of approximately equal size.

Exercise 11.3. The algorithms described in this chapter assume 8-bit grayscale input images (of type `ByteProcessor` in ImageJ). Adopt the current implementations to work with 16-bit integer image (of type `ShortProcessor`). Images of this type may contain pixel values in the range $[0, 2^{16}-1]$ and the `getHistogram()` method returns the histogram as an integer array of length 65536.

Exercise 11.4. Implement simple thresholding for RGB color images by thresholding each (scalar-valued) color channel individually and then merging the results by performing a pixel-wise AND operation. Compare the results to those obtained by thresholding the corresponding grayscale (luminance) images.

Exercise 11.5. Re-implement the Bernsen and/or Niblack thresholder (classes `BernsenThresholder` and `NiblackThresholder`) using integral images (see Ch. 3, Sec. 3.8) for efficiently calculating the required local mean and variance of the input image over a rectangular support region R.