Lecture 4: Thresholding

Last modified on Wednesday, January 12, 2000 at 10:00 AM.

Reading

SH&B, Section 5.1

4.1 Introduction

Segmentation involves separating an image into regions (or their contours) corresponding to objects. We usually try to segment regions by identifying common properties. Or, similarly, we identify contours by identifying *differences* between regions (edges).

The simplest property that pixels in a region can share is intensity. So, a natural way to segment such regions is through *thresholding*, the separation of light and dark regions.

Thresholding creates binary images from grey-level ones by turning all pixels below some threshold to zero and all pixels about that threshold to one. (What you want to do with pixels at the threshold doesn't matter, as long as you're consistent.)

If $g(x, y)$ is a thresholded version of $f(x, y)$ at some global threshold T,

$$g(x,y) = \begin{cases} 1 & \text{if } f(x,y) \geq T \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

4.2 Problems with Thresholding

The major problem with thresholding is that we consider only the intensity, not any relationships between the pixels. There is no guarantee that the pixels identified by the thresholding process are contiguous.

We can easily include extraneous pixels that aren't part of the desired region, and we can just as easily miss isolated pixels within the region (especially near the boundaries of the region). These effects get worse as the noise gets worse, simply because it's more likely that a pixels intensity doesn't represent the normal intensity in the region.

When we use thresholding, we typically have to play with it, sometimes losing too much of the region and sometimes getting too many extraneous background pixels. (Shadows of objects in the image are also a real pain—not just where they fall across another object but where they mistakenly get included as part of a dark object on a light background.)

4.3 Local Thresholding

Another problem with global thresholding is that changes in illumination across the scene may cause some parts to be brighter (in the light) and some parts darker (in shadow) in ways that have nothing to do with the objects in the image.

We can deal, at least in part, with such uneven illumination by determining thresholds locally. That is, instead of having a single global threshold, we allow the threshold itself to smoothly vary across the image.

4.4 Automated Methods for Finding Thresholds

To set a global threshold or to adapt a local threshold to an area, we usually look at the histogram to see if we can find two or more distinct modes—one for the foreground and one for the background.

Recall that a histogram is a probability distribution:

$$p(g) = n_g/n \tag{4.2}$$

That is, the number of pixels n_g having greyscale intensity g as a fraction of the total number of pixels n.

Here are five different ways to look at the problem:

4.4.1 Known Distribution

If you know that the object you're looking for is brighter than the background and occupies a certain fraction $1/p$ of the image, you can set the threshold by simply finding the intensity level such that the desired percentage of the image pixels are below this value. This is easily extracted from the cumulative histogram:

$$c(g) = \sum_{0}^{g} p(g) \tag{4.3}$$

Simply set the threshold T such that $c(T) = 1/p$. (Or, if you're looking for a dark object on a light background, $c(T) = 1 - 1/p$.)

4.4.2 Finding Peaks and Valleys

One extremely simple way to find a suitable threshold is to find each of the modes (local maxima) and then find the valley (minimum) between them.

While this method appears simple, there are two main problems with it:

1. The histogram may be noisy, thus causing many local minima and maxima. To get around this, the histogram is usually smoothed before trying to find separate modes.

2. The sum of two separate distributions, each with their own mode, may not produce a distribution with two distinct modes. (See the plots on the right side of Figure 5.4 of your text.)

4.4.3 Clustering (K-Means Variation)

Another way to look at the problem is that we have two groups of pixels, one with one range of values and one with another. What makes thresholding difficult is that these ranges usually overlap. What we want to do is to minimize the error of classifying a background pixel as a foreground one or vice versa. To do this, we try to minimize the area under the histogram for one region that lies on the other region's side of the threshold. The problem is that we don't have the histograms for each region, only the histogram for the combined regions. (If we had the regions, why would we need to do segmentation?)

Understand that the place of minimum overlap (the place where the misclassified areas of the distributions are equal) is *not* is not necessarily where the valley occurs in the combined histogram. This occurs, for example, when one cluster has a wide distribution and the other a narrow one. (See the plots in Figure 5.4 of your text.)

One way that we can try to do this is to consider the values in the two regions as two *clusters*. Section 5.1.2 of your book describes a method for finding a threshold by clustering the histogram. The idea is to pick a threshold such that each pixel on each side of the threshold is closer in intensity to the mean of all pixels on that side of the threshold than the mean of all pixels on the other side of the threshold.

In other words, let $\mu_B(T)$ be the mean of all pixels less than the threshold and $\mu_O(T)$ be the mean of all pixels greater than the threshold. We want to find a threshold such that the following holds:

$$\forall g \geq T : |g - \mu_B(T)| > |g - \mu_O(T)|$$

and

$$\forall g < T : |g - \mu_B(T)| < |g - \mu_O(T)|$$

Algorithm 5.2 in your text shows an algorithm for finding such clustering. This is a variation of the *k-means clustering algorithm* used in pattern recognition and discussed in CS 521. The basic idea is to start by estimating $\mu_B(T)$ as the average of the four corner pixels (assumed to be background) and $\mu_O(T)$ as the average of everything else. Set the threshold to be halfway between $\mu_B(T)$ and $\mu_O(T)$ (thus separating the pixels according to how close their intensities are to $\mu_B(T)$ and $\mu_O(T)$ respectively). Now, update the estimates of $\mu_B(T)$ and $\mu_O(T)$ respectively

by actually calculating the means of the pixels on each side of the threshold. This process repeats until the algorithm converges.

This method works well if the spreads of the distributions are approximately equal, but it does not handle well the case where the distributions have differing variances.

4.4.4 Clustering (The Otsu Method)

Another way of accomplishing similar results is to set the threshold so as to try to make each cluster as tight as possible, thus (hopefully!) minimizing their overlap. Obviously, we can't change the distributions, but we can adjust where we separate them (the threshold). As we adjust the threshold one way, we increase the spread of one and decrease the spread of the other. The goal then is to select the threshold that minimizes the combined spread.

We can define the *within-class* variance as the weighted sum of the variances of each cluster:

$$\sigma^2_{\text{Within}}(T) = n_B(T)\sigma^2_B(T) + n_O(T)\sigma^2_O(T) \tag{4.4}$$

where

$$n_B(T) = \sum_{i=0}^{T-1} p(i) \tag{4.5}$$

$$n_O(T) = \sum_{i=T}^{N-1} p(i) \tag{4.6}$$

$$\sigma^2_B(T) = \text{the variance of the pixels in the background (below threshold)} \tag{4.7}$$

$$\sigma^2_O(T) = \text{the variance of the pixels in the foreground (above threshold)} \tag{4.8}$$

$$\tag{4.9}$$

and $[0, N-1]$ is the range of intensity levels.

Computing this within-class variance for each of the two classes for each possible threshold involves a lot of computation, but there's an easier way.

If you subtract the within-class variance from the total variance of the combined distribution, you get something called the between-class variance:

$$\begin{aligned} \sigma^2_{\text{Between}}(T) &= \sigma^2 - \sigma^2_{\text{Within}}(T) \\ &= n_B(T)\left[\mu_B(T) - \mu\right]^2 + n_O(T)\left[\mu_O(T) - \mu\right]^2 \end{aligned} \tag{4.10}$$

where σ^2 is the combined variance and μ is the combined mean. Notice that the between-class variance is simply the weighted variance of the cluster means themselves around the overall mean. Substituting $\mu = n_B(T)\mu_B(T) + n_O(T)\mu_O(T)$ and simplifying, we get

$$\sigma^2_{\text{Between}}(T) = n_B(T)n_O(T)[\mu_B(T) - \mu_O(T)]^2 \tag{4.11}$$

So, for each potential threshold T we

1. Separate the pixels into two clusters according to the threshold.
2. Find the mean of each cluster.
3. Square the difference between the means.
4. Multiply by the number of pixels in one cluster times the number in the other.

This depends only on the difference between the means of the two clusters, thus avoiding having to calculate differences between individual intensities and the cluster means. The optimal threshold is the one that maximizes the between-class variance (or, conversely, minimizes the within-class variance).

This still sounds like a lot of work, since we have to do this for each possible threshold, but it turns out that the computations aren't independent as we change from one threshold to another. We can update $n_B(T)$, $n_O(T)$, and the respective cluster means $\mu_B(T)$ and $\mu_O(T)$ as pixels move from one cluster to the other as T increases. Using simple recurrence relations we can update the between-class variance as we successively test each threshold:

$$\begin{aligned} n_B(T+1) &= n_B(T) + n_T \\ n_O(T+1) &= n_O(T) - n_T \\ \mu_B(T+1) &= \frac{\mu_B(T)n_B(T) + n_T T}{n_B(T+1)} \\ \mu_O(T+1) &= \frac{\mu_O(T)n_O(T) - n_T T}{n_O(T+1)} \end{aligned}$$

This method is sometimes called the *Otsu* method, after its first publisher.

4.4.5 Mixture Modeling

Another way to minimize the classification error in the threshold is to suppose that each group is Gaussian-distributed. Each of the distributions has a mean (μ_B and μ_O respectively) and a standard deviation (σ_B and σ_O respectively) *independent of the threshold we choose*:

$$h_{\text{model}}(g) = n_B \ e^{-(g-\mu_B)^2/2\sigma_B^2} + n_O \ e^{-(g-\mu_O)^2/2\sigma_O^2}$$

Whereas the Otsu method separated the two clusters according to the threshold and tried to optimize some statistical measure, *mixture modeling* assumes that there already exists two distributions and we must find them. Once we know the parameters of the distributions, it's easy to determine the best threshold.

Unfortunately, we have six unknown parameters (n_B, n_O, μ_B, μ_O, σ_B, and σ_O), so we need to make some estimates of these quantities.

If the two distributions are reasonably well separated (some overlap but not too much), we can choose an arbitrary threshold T and assume that the mean and standard deviation of each group approximates the mean and standard deviation of the two underlying populations. We can then measure how well a mix of the two distributions approximates the overall distribution:

$$F = \sum_{0}^{N-1} \Big[h_{\text{model}}(g) - h_{\text{image}}(g) \Big]^2$$

Choosing the optimal threshold thus becomes a matter of finding the one that causes the mixture of the two estimated Gaussian distributions to best approximate the actual histogram (minimizes F). Unfortunately, the solution space is too large to search exhaustively, so most methods use some form of gradient descent method. Such gradient-descent methods depend heavily on the accuracy of the initial estimate, but the Otsu method or similar clustering methods can usually provide reasonable initial estimates.

Mixture modeling also extends to models with more than two underlying distributions (more than two types of regions). For example, segmenting CT images into grey matter, white matter, and cerebral spinal fluid (CSF).

4.5 Multispectral Thresholding

Section 5.1.3 of your text describes a technique for segmenting images with multiple components (color images, Landsat images, or MRI images with T1, T2, and proton-density bands). It works by estimating the optimal threshold in one channel and then segmenting the overall image based on that threshold. We then subdivide each of these regions independently using properties of the second channel. We repeat it again for the third channel, and so on, running through all channels repeatedly until each region in the image exhibits a distribution indicative of a coherent region (a single mode).

4.6 Thresholding Along Boundaries

If we want our thresholding method to give stay fairly true to the boundaries of the object, we can first apply some boundary-finding method (such as edge detection techniques that we'll cover later) and then sample the pixels only where the boundary probability is high.

Thus, our threshold method based on pixels near boundaries will cause separations of the pixels in ways that tend to preserve the boundaries. Other scattered distributions within the object or the background are of no relevance.

However, if the characteristics change along the boundary, we're still in trouble. And, of course, there's still no guarantee that we'll not have extraneous pixels or holes.

Vocabulary

- Thresholding
- Local Thresholding
- Clustering
- Otsu Method
- Mixture Modeling