**Binary Images**

Binary images are grayscale images with only two possible levels of brightness for each pixel: **black** or **white**.

Binary images require little memory for storage and can be processed very quickly.

They are a good representation of an object if

- we are only interested in the **contour** of that object, and
- the object can be **separated** from the background and from other objects (no occlusion).

**Thresholding**

We usually create binary images from grayscale images through **thresholding**.

This can be done easily and perfectly if, for example, the brightness of pixels is lower for those of the object than for those of the background.

Then we can set a **threshold** $\theta$ such that $\theta$

- is greater than the brightness value of any object pixel and
- is smaller than the brightness value of any background pixel.

**Thresholding**

In that case, we can apply the threshold $\theta$ to the original image $A[i, j]$ to generate the **thresholded image** $A_\theta[i, j]$:

$$A_\theta[i, j] = 1 \quad \text{if} \quad A[i, j] \leq \theta$$

$$A_\theta[i, j] = 0 \quad \text{otherwise}$$

The convention for binary images is that pixels belonging to the object(s) have value 1 and all other pixels have value 0.

We usually display 1-pixels in black and 0-pixels in white.

**Thresholding**

If we know that the intensity of all object pixels is in the **range** between values $\theta_1$ and $\theta_2$, we can perform the following thresholding operation:

$$A_\theta[i, j] = 1 \quad \text{if} \quad \theta_1 \leq A[i, j] \leq \theta_2$$

$$A_\theta[i, j] = 0 \quad \text{otherwise}$$

If the intensities of all object pixels are **not in a particular interval**, but are still distinct from the background values, we can do the following:

$$A_Z[i, j] = 1 \quad \text{if} \quad A[i, j] \in Z$$

$$A_Z[i, j] = 0 \quad \text{otherwise},$$

Where $Z$ is the set of intensities of object pixels.

**Thresholding**

Here, the right image is created from the left image by thresholding, assuming that object pixels are darker than background pixels.

As you can see, the result is slightly imperfect (dark background pixels).

**Thresholding**

**How to find the optimal threshold?**
Thresholding

Intensity histogram

Thresholding result

Some Definitions

For a pixel \([i, j]\) in an image, …

\([i-1, j]\) \([i, j-1]\) \([i, j+1]\) \([i+1, j]\)

…these are its 4-neighbors (4-neighborhood).

\([i-1, j-1]\) \([i-1, j+1]\) \([i+1, j-1]\) \([i+1, j+1]\)

…these are its 8-neighbors (8-neighborhood).

Some Definitions

The set of all 1-pixels in an image is called the foreground and is denoted by \(S\).

A pixel \(p \in S\) is said to be connected to \(q \in S\) if there is a path from \(p\) to \(q\) consisting entirely of pixels of \(S\).

Connectivity is an equivalence relation, because

- Pixel \(p\) is connected to itself (reflexivity).
- If \(p\) is connected to \(q\), then \(q\) is connected to \(p\) (symmetry).
- If \(p\) is connected to \(q\) and \(q\) is connected to \(r\), then \(p\) is connected to \(r\) (transitivity).

Some Definitions

A path from the pixel at \([i_0, j_0]\) to the pixel \([i_n, j_n]\) is a sequence of pixel indices \([i_0, j_0], [i_1, j_1], \ldots, [i_n, j_n]\) such that the pixel at \([i_k, j_k]\) is a neighbor of the pixel at \([i_{k+1}, j_{k+1}]\) for all \(k\) with \(0 \leq k \leq n - 1\).

If the neighbor relation uses 4-connection, then the path is a 4-path; for 8-connection, the path is an 8-path.

Some Definitions

A set of pixels in which each pixel is connected to all other pixels is called a connected component.

The set of all connected components of \(\neg S\) (the complement of \(S\)) that have points on the border of an image is called the background. All other components of \(\neg S\) are called holes.

To avoid ambiguity, use 4-connectedness for foreground and 8-connectedness for background or vice versa.
Some Definitions
The **boundary** of S is the set of pixels of S that have 4-neighbors in \( \neg S \). The boundary is denoted by \( S' \).
The **interior** is the set of pixels of S that are not in its boundary. The interior of S is \( (S \setminus S') \).
Region T **surrounds** region S (or S is **inside** T), if any 4-path from any point of S to the border of the picture must intersect T.

Component Labeling
Component labeling is one of the most fundamental operations on binary images.
It is used to distinguish different objects in an image, for example, bacteria in microscopic images.
We find all connected components in an image and assign a unique label to all pixels in the same component.

Component Labeling
A simple algorithm for labeling connected components works like this:
1. Scan the image to find an unlabeled 1-pixel and assign it a new label \( L \).
2. Recursively assign a label \( L \) to all its 1-pixel neighbors.
3. Stop if there are no more unlabeled 1-pixels.
4. Go to step 1.
However, this algorithm is very **inefficient**.
Let us develop a more efficient, non-recursive algorithm.

Size Filter
We can use component labeling to **remove noise** in binary images.
For example, when we want to perform optical character recognition (OCR), it often happens that there are small groups of 1-pixels outside the actual characters.
Since these are usually very small, isolated blobs, we can remove them by applying a **size filter**, that is,
- labeling all components,
- computing their size, and
- for all components smaller than a threshold \( \tau \), setting all of their pixels to 0.

Component Labeling
1. Scan the image left to right, top to bottom.
2. If the pixel is 1, then
   - If only one of its upper and left neighbors has a label, then copy the label.
   - If both have the same label, then copy the label.
   - If both have different labels, then copy the upper neighbor’s label and enter both labels in the equivalence table as equivalent labels.
   - Otherwise assign a new label to this pixel and enter this label in the equivalence table.
3. If there are more pixels to consider, then go to Step 2.
4. Find the lowest label for each equivalence set in the equivalence table.
5. Scan the picture. Replace each label by the lowest label in its equivalence set.

Size Filter
Here, for \( \tau = 10 \), the size filter perfectly removes all noise in the input image.
However, if our threshold is too high, “accidents” may happen.

In the case we had only “positive noise,” that is, there were some 1-pixels in places that should have contained 0-pixels,

Often, we also have “negative noise,” which means that we have 0-pixels in places that should contain 1-pixels.

To remove negative noise, we could define a “hole size filter” that removes all holes that are smaller than a certain threshold.

A common, efficient method of removing both kinds of noise is to apply sequences of expanding and shrinking.

As we have just seen, using a size filter is one method for preprocessing images for subsequent character recognition.

Another common way of achieving this is called expanding and shrinking.

Expanding operation: For all pixels in the image, change a pixel from 0 to 1 if any neighbors of the pixel are 1.

Shrinking operation: For all pixels in the image, change a pixel from 1 to 0 if any neighbors of the pixel are 0.

Expanding followed by shrinking can be used for filling undesirable holes.

Shrinking followed by expanding can be used for removing isolated noise pixels.

If the resolution of the image is sufficiently high and the noise level is low, expanding – shrinking – shrinking – expanding sequence may be able to do both tasks.

Of course we always have to perform the same total number of expanding and shrinking operations.
Compactness

For a two-dimensional continuous geometric figure, its compactness is measured by the quotient $P^2/A$, where $P$ is the figure’s perimeter and $A$ is its area.

For example, for a square of height $s$ we have $P = 4s$ and $A = s^2$, so its compactness is 16.

For a circle of radius $r$ we have $P = 2\pi r$ and $A = \pi r^2$, so its compactness is $4\pi \approx 12.56$.

No figure is more compact than a circle, so $4\pi$ is the minimum value for compactness.

Notice: The more compact a figure is, the lower is its compactness value.

Geometric Properties

Let us say that we want to write a program that can recognize different types of tools in binary images.

Then we have the following problem:

The same tool could be shown in different

• sizes,
• positions, and
• orientations.

However, that would be a very inefficient and inflexible approach.

Instead, it is much simpler and more efficient to standardize the input before performing object recognition.

We can scale the input object to a given size, center it in the image, and rotate it towards a specific orientation.
Computing Object Size

The size $A$ of an object in a binary image $B$ is simply defined as the number of black pixels ("1-pixels") in the image:

$$A = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} B[i,j]$$

$A$ is also called the zeroth-order moment of the object.

In order to standardize the size of the object, we expand or shrink the object so that its size matches a predefined value.

Computing Object Position

We compute the position of an object as the center of gravity of the black pixels:

$$X = \frac{\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} i B[i,j]}{A}$$

$$Y = \frac{\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} j B[i,j]}{A}$$

These are also called the first-order moments of the object.

In order to standardize the position of the object, we shift its position so that it is in the center of the image.