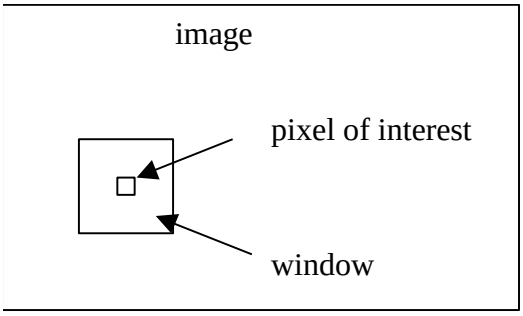
Lecture notes: Segmentation

Convolution is a method to extract information from a window of pixels:

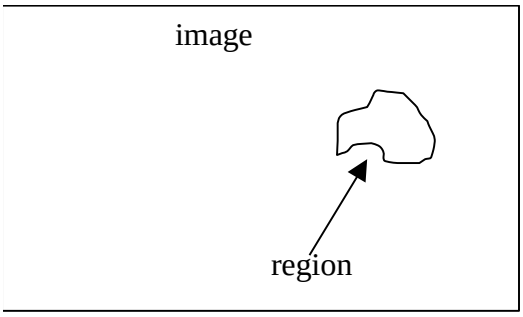
window(pixel) => information (e.g. avg color)



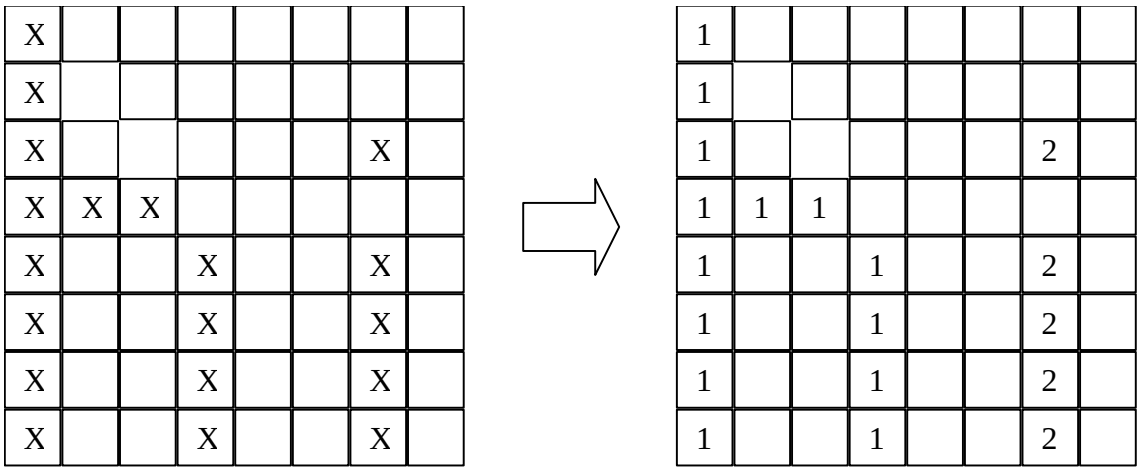
Methods that use convolution to find things are useful when the problem is to find instances of a **known shape** (e.g. locate enemy tanks, find tumors, find all letter t's). However, there are times when we are not sure what we are looking for. In this case, we need more generic methods to "find things".

Segmentation methods extract information from an arbitrarily shaped set of pixels:

region = predicate(set of pixels) => information (e.g. area, location, etc.)



The process of **segmentation** is **assigning pixels to groups.** For example:



In this example the pixels comprising the letter "h" are assigned to group "1", and the pixels comprising the letter "i" are assigned to group "2". Each such group is called a **region.**

Providing a more formal definition:

Let R represent the entire image region. We may view segmentation as a process that partitions **R** into n subregions $\mathbf{R_1}$, $\mathbf{R_2}$, … $\mathbf{R_n}$, such that

1. $\cup_{i=1}^{n} R_i = R$
2. R_i is a connected region, i=1,2,… n
3. $R_i \cap R_j = \emptyset$ for all $i \neq j$
4. $P(R_i) =$ TRUE for i=1,2,… n
5. $P(R_i \cup R_j) =$ FALSE for $i \neq j$, for R_i not adjacent to R_j

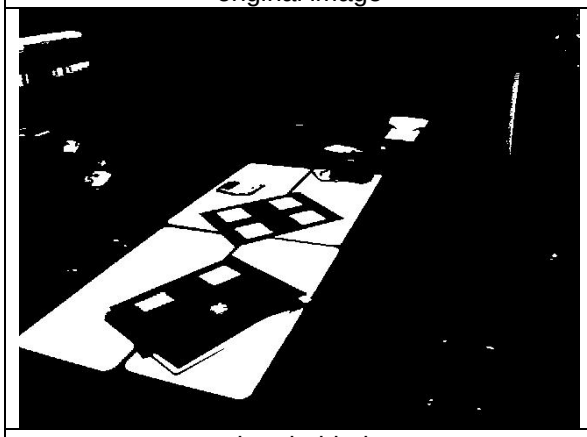
where $P(R_i)$ is a logical predicate over the points in set R_i and $\emptyset$ is the null set.

One common extension to this definition is to reserve a single label R_0 for all pixels that are "unlabeled" or noise or background. Region R_0 does not satisfy predicates 2 and 4. In essence, it satisfies the complement of predicate 4.

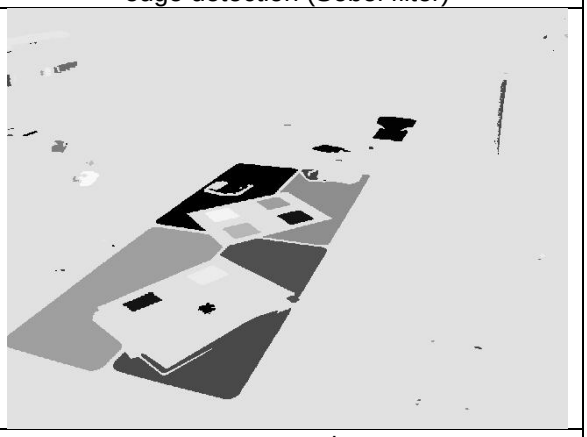
In some problems the definition of the grouping is clear. For example, the above groupings define individual letters. However, in the majority of images the grouping definition is vague and open to interpretation. For example, consider the following:

original image

edge detection (Sobel filter)



thresholded



segmentation

In this figure we see an original image (top left) after thresholding (bottom left), after Sobel edge detection (top right), and after segmentation (bottom right). Some of the regions seem obvious, such as those defining the squares on the table. Others are vague, such as in the background.

There is no such thing as a "generic" or "all-purpose" segmenter. Segmentation is a tool that must be tuned to the problem.

If each person in the class were to hand-segment the image, how many different answers would we get? There are a couple large data sets where exactly this question has been asked. One of the most popular is Berkley's data set (see the link provided at the web site).

How can an image be segmented? There are lots of ways. We will look at the three most common approaches (two in this lecture, and the most popular method next time).

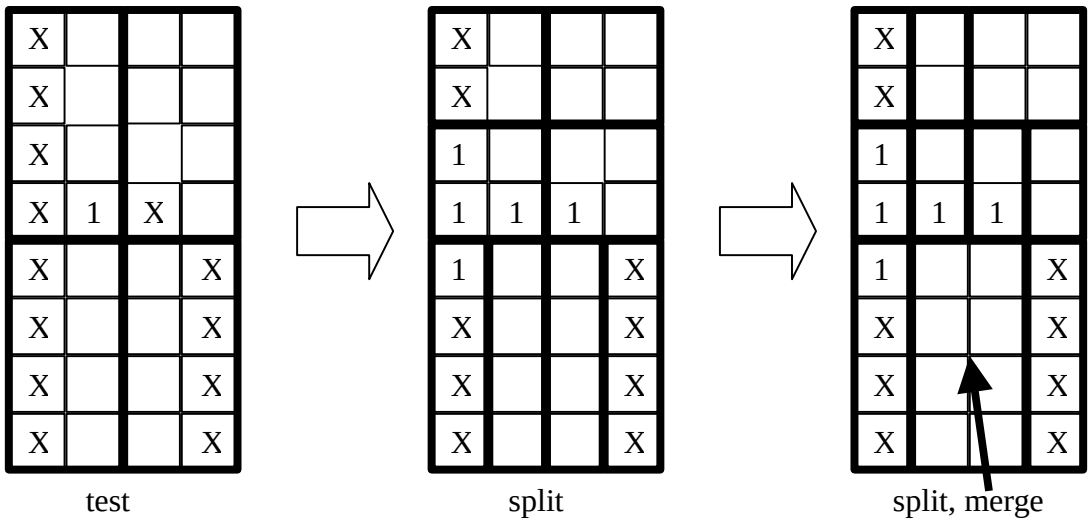
(1) Split-and-merge.

Initialize image as large regions.

a) **Similarity test(s).** Test all pixels within each region for similarity (segmentation predicate). If test passes, goto (c).

b) **Split.** Break up region into set of regions, iterate to (a) for each.

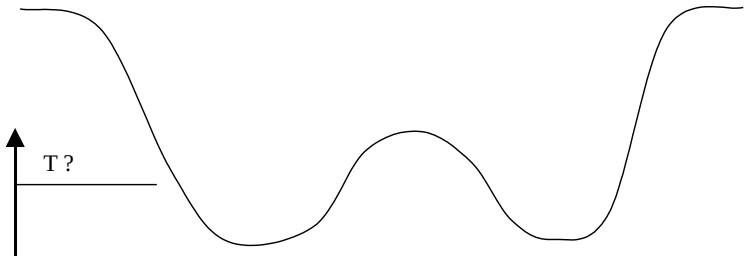
c) **Merge.** Test neighboring regions for similarity, combining if pass.



Splitting can be done using any method, but usually divides the regions into equal numbers of pixels. Usually all the splitting is done first, followed by all the merging. The nice thing about this approach is that the merging step can "fix" over-segmentation problems.

(2) Watershed.

If you know how many regions you want to obtain during a "split", then the watershed approach can provide a good segmentation. For example, imagine looking for two pools in the following 1D profile:



We can raise the threshold T until the two pools just merge. This is called watershed thresholding.