Lecture 18: Segmentation (Region Based)

Last modified on March 3, 2000 at 6:00 PM

Contents

Reading

SH&B, 5.3
Castleman 18.8.1, 18.6

18.1 Introduction

We now turn from segmentation by finding boundaries (pixel differences) to segmentation by finding coherent regions (pixel similarities).

This approach has specific advantages over boundary based methods:

- It is guaranteed (by definition) to produce coherent regions. Linking edges, gaps produced by missing edge pixels, etc. are not an issue.
- It works from the inside out, instead of the outside in. The question of which object a pixel belongs to is immediate, not the result of point-in-contour tests.

However, it also has drawbacks:

- Decisions about region membership are often more difficult than applying edge detectors.

- It can't find objects that span multiple disconnected regions. (Whereas edge-based method can be designed to handle "gaps" produced by occlusion—the Hough transform is one example.)

The objectives of region-based approaches can be summarized as follows:

- Produce regions that are as large as possible (i.e., produce as few regions as possible).
- Produce coherent regions, but allow some flexibility for variation within the region.

Notice the inherent tradeoffs here. If we require that the pixels in a region be too similar, we get great coherency and probably won't span separate objects, but we *oversegment* the image into regions much smaller than the actual objects. If we allow more flexibility, we can produce larger regions that more likely fill entire objects, but they may cross multiple objects and "leak" across what should otherwise be boundaries. Remember: the real goal is to find regions that correspond to objects as a person sees them—not an easy goal.

18.2 Basic Idea of Region Growing

Suppose that we start with a single pixel p and wish to expand from that *seed pixel* to fill a coherent region. Let's define a similarity measure $S(i, j)$ such that it produces a high result if pixels i and j are similar and a low one otherwise. First, consider a pixel q adjacent to pixel p. We can add pixel q to pixel p's region iff $S(p, q) > T$ for some threshold T. We can then proceed to the other neighbors of p and do likewise.

Suppose that $S(p, q) > T$ and we added pixel q to pixel p's region. We can now similarly consider the neighbors of q and add them likewise if they are similar enough. If we continue this recursively, we have an algorithm analogous to a "flood fill" but which works not on binary data but on *similar* greyscale data.

Of course, we now have a few unanswered questions to address:

1. How do we define similarity measure S?
2. What threshold T do we use? Does it change or stay constant?
3. If we wish to add q's neighbor r, do we use $S(p, r)$, $S(q, r)$, or something else?

18.3 A Few Common Approaches

18.3.1 Similarity Measures

One obvious similarity measure is to compare individual pixel intensities. This can be sensitive to noise, though.

We can reduce our sensitivity to noise by comparing neighborhood characteristics between pixels. For example, we could compare the *average intensities* over a neighborhood around each pixel. Notice, though, that this is simply the same as applying an averaging kernel through convolution and then doing single-pixel comparison.

One can also gather other features from the neighborhood, such as texture (which we'll cover in a later lecture), gradient, or geometric properties.

18.3.2 Comparing to Original Seed Pixel

One approach is to always compare back to the seed point p by using $S(p, r)$ when considering adding pixel r to the growing region. This the advantage of using a single basis for comparison across all pixels in the region. However, it means that the region produced is very sensitive to the choice of seed pixel. Does it make sense that the region produced by growing pixel p is different than that produced by its neighbor q also in the same region?

18.3.3 Comparing to Neighbor in Region

One way of removing this effect is to compare pixel r to the neighboring pixel q already in the region of p: $S(q, r)$. In this way, each pixel that is already in the region can bring in neighbors who are like it.

The advantage of this method is that it produces transitive closures of similarity. If p is similar to q, and if q is similar to r, p and r end up in the same region.

Of course, this method can cause significant drift as one grows farther away from the original seed pixel. Indeed, the original seed is of no significance once one grows out more than one pixel. What started out finding green pixels ends up adding yellow ones if the transition is gradual enough.

18.3.4 Comparing to Region Statistics

A third approach is to compare candidate pixel r to the *entire* region already collected. Initially, this region consists of pixel p alone, so pixel p dominates. As the region grows, aggregate statistics are collected, and any new pixel r which may be added to the region is compared not to pixel p or to its already-included neighbor q, but to these aggregate statistics.

One simple such statistic is to keep an updated mean of the region pixels. As each new pixel is added, this mean is updated. Although gradual drift is still possible, the weight of all previous pixels in the region act as a damper on such drift. Some texts refers to this as *centroid* region growing.

18.3.5 Multiple Seeds

Another approach is to initialize the region with not only a single pixel but a small set of pixels to better describe the region statistics. With such initialization, not only a region mean is suggested but the *variance* as well. Candidate points can be compared to the region mean *with respect to the region variance*. This gives us at least some hope of producing identical regions under varying noise conditions.

Multiple seeds can be selected either through user interaction or by sampling a small area around the initial seed point. In other words, if we assume that no desired region is indeed smaller than, say, 5 pixels across, we can sample our initial statistics from a 5×5 area around the seed and grow outwards from there.

18.3.6 Cumulative Differences

Another approach is to use a similarity measure that involves not only comparison to a single pixel or to a region statistic, but by calculating cumulative differences as one follows a path from the seed to the candidate point.

In other words, if point q is a neighbor or seed p, and candidate point r is a neighbor of q, instead of using $S(p, r)$ or $S(q, r)$, we can use $S(p, q) + S(q, r)$. This is equivalent to finding the minimum-cost path from p to r and using this as the basis for the addition or rejection of point r.

18.3.7 Counterexamples

Yet another approach is to provide not only a seed pixel that *is* in the desired region but also counterexamples that *are not* in the region. This method allows us to use not only the similarity to the region but the dissimilarity to the exterior (the counterexample). The has the advantage of not requiring a predetermined threshold—the threshold is simply the value at which the candidate point becomes more similar to the background than to the foreground region. It does, however, require some prior knowledge to "train" the system in this way.

18.4 Multiple Regions

18.4.1 Selecting Seed Points

One way to select seed points is to do so interactively. A user can click a mouse inside a desired object and ideally fill the entire object.

But what about automatically segmenting an entire scene this way? Surely user-specified seed points are insufficient for this task. One way is to scatter seed points around the image and hope to fill all of the image with regions. If

the current seed points are insufficient, one can include additional seeds by selecting points not already in segmentated regions.

18.4.2 Scanning

Multiple regions can also be identified by scanning the image in a regular fashion adding pixels to growing regions and spawning new regions as needed. We can then make additional passes through the image resolving these regions. Notice that this is basically the same connected-component labelling that we saw earlier, only with a similarity measure instead of binary values.

18.4.3 Region Merging

The limit of the multiple-seed approach is to let *every* pixel be a seed. If two adjacent pixels are similar, merge them into a single region. If two adjacent regions are collectively similar enough, merge them likewise. This collective similarity is usually based on comparing the statistics of each region. Eventually, this method will converge when no further such mergings are possible.

18.4.4 Split and Merge Algorithm

Pure merging methods are, however, computationally expensive because they start from such small initial regions (individual points). We can make this more efficient by recursively splitting the image into smaller and smaller regions until all individual regions are coherent, then recursively merging these to produce larger coherent regions.

First, we must split the image. Start by considering the entire image as one region.

1. If the entire region is coherent (i.e., if all pixels in the region have sufficient similarity), leave it unmodified.
2. If the region is not sufficiently coherent, split it into four quadrants and recursively apply these steps to each new region.

The "splitting" phase basically builds a quadtree like we discussed earlier in Lecture 3. Notice that each of the regions (squares) so produced is now coherent. However, several adjacent squares of varying sizes might have similar characteristics.

We can thus merge these squares into larger regions from the bottom up, much as we merged regions earlier. Since we are starting with regions (hopefully) larger than single pixels, this method is more efficient.

18.5 Hybrid Edge/Region Approaches

One can extend the power of both region- and boundary-based segmentation methods by combining the strengths of the two. For example, we can make region-joining decisions based not only on pixel or neighborhood similarity but also on already-extracted edges and completion of these edges.

Once we've identified two adjanct regions that are candidates for merging, we can examine the boundary between them. If the boundary is of sufficient strength (gradient magnitude, number of edge pixels according to some operator, etc.), we keep the regions separate. If the boundary between them is weak, we merge them.

Most such algorithms use a minimum fraction of edge pixels along the shared region boundary instead of just looking for holes in the boundaries. Thus, even if there is a small hole in the edge contour (noise, variation, etc.), the regions are still kept separate unless this hole or other holes form a sufficiently large fraction of the boundary.

18.6 Watersheds

18.6.1 Basic Definition

A *watershed region* or *catchment basin* is defined as the region over which all points flow "downhill" to a common point. The idea originates from geology but has been applied to vision as well. In geology, one might want to consider

the local region where all rainwater flows to a single lake or river. This might not seem to be applicable to intensity-based regions (why do we care about following intensity changes to a common dark point?) but it makes sense if we apply them to *gradient magnitude* images.

18.6.2 Watersheds of Gradient Magnitude

We talked before about how we'd like our edges to be locii of "maximal" gradient magnitude. One way of defining these maximal curves (ridges) is as the boundaries of watershed regions—everything on one side "flows" downhill to one side and everything on the other flows to the other side. Thus, as you cross from one watershed region to another, you've had to cross over some local ridge curve. Unfortunately, this definition of a ridge isn't based on purely local properties but instead requires building the regions first. But for segmentation, isn't that what we're after?

So, a common technique for segmentation is to use *gradient watershed regions*. First build a gradient magnitude image, then find the watershed regions in this image.

18.6.3 Basic Implementation

One way to find watersheds is to think of filling them from the bottom up and finding where different pools merge.

Suppose that the greylevel range is [0,255]. Start first with the 0 pixels. Clearly, nothing can be less than these, so these form the basis for new watersheds. Then add all pixels with intensity 1. If they are next to existing watersheds (0-intensity pixels), add them to these regions. Otherwise, start a new region for each pixel that is not next to an existing region.

This process repeats for each intensity k up to the maximum (255 in this example). Each k-intensity pixel that is next to an already-labeled watershed region is added to that region. Each k-intensity pixel that is *not* next to an already-labeled watershed region starts a new region.

One must be careful though: there may be multiple k-intensity pixels together, and if any one of these is adjacent to an already-labeled region, they all should be added.

This basic algorithm can be made faster by maintaining histogram-like data structures that index all pixels by their intensity. This makes it very fast to find all pixels with a particular intensity k.

18.6.4 Tobogganing

Another approach is to "slide" down a watershed to the common minimum. This approach links each pixel with the smallest of its neighbors. That one is linked to the smallest of its neighbors, that one to the smallest of its neighbors, etc. Eventually you get to points that have no smaller neighbors (local minima), which form the basis for a watershed region. The closure of all points that eventually link to the same minimum defines the watershed. This approach has been dubbed "tobogganing" because of its similarity to riding a sled down a mountainside.

18.7 Postprocessing

The major problem with region-based segmentation is that they usually undersegment (too few, too large regions) or oversegment the image (too many, too little regions). This is especially true for watersheds—even a little noise can make an otherwise homogeneous region "crinkle" into lots of small watersheds causing significant oversegmentation. Because of this, most implementations of region-based segmentation involve some form of postprocessing step that attempts to either split undersegmented regions or merge oversegmented ones.

The problem with this is that *the segmentation depends on what you are trying to do with the image*. If you are segmenting a tiger or a zebra, is it the stripes or the entire body that you're after? The only way of segmenting the image in any meaningful way is the one that considers images at these different levels.

This idea is similar to the one we talked about earlier for scale—indeed, some approaches to this problem do use multiscale segmentation. The idea is to build a *hierarchy* of regions that (hopefully!) are isolated regions at one level and parts of the same object at others.

Vocabulary

- Region growing
- Region splitting
- Region Merging
- Split and Merge Algorithm
- Watershed regions
- Tobogganing