Image Segmentation by Cluster Analysis

We could also think of regions in images as clusters of pixels that are close to each other spatially and in terms of their intensity (or other features).

There are numerous clustering algorithms such as k-means or hierarchical techniques.

For image segmentation, one such approach, mean shift clustering, has proven to be particularly powerful.

Its basic idea is to, starting at each data point, follow the density gradient to a local density maximum. Those points being associated with the same maximum are assigned to the same cluster.

Mean Shift Clustering

Algorithm:
1. Place point $q$ on one of the data points.
2. Find the center of gravity $c$ (i.e., the mean) of all data points within a radius $r$ from $q$.
3. Move $q$ to point $c$.
4. Repeat steps 2 and 3 until the motion is smaller than one pixel per iteration.
5. Repeat steps 1 to 4 for each pixel in the image.
6. Assign all pixels for which $q$ ends up at the same point (within radius $r$) to the same cluster.
In order to use the mean shift algorithm for segmenting images, we should consider each pixel as a 3D point (row, column, intensity). Regions form clusters in this 3D space. To apply mean shift clustering, we can use a sphere instead of a circle.

Let us consider a one-dimensional image (or a horizontal cut through a 2D image) to illustrate this.
Mean Shift Segmentation

Better results can be obtained by giving greater weight to points near the center of the sphere than to those further away from it. We can use Gaussian functions for computing a weighted mean. Also, we may want to give different amounts of tolerance to the spatial and intensity dimensions.

The center of gravity $c$ for the $P$ pixels (points) in the image window can then be computed as follows:

$$
\begin{align*}
\mathbf{c} &= \frac{\sum_{i=1}^{P} \mathbf{p}_i \cdot e^{-\frac{1}{h_i^2} \left( \frac{(p_{i,s} - q_s)^2}{h_s^2} + \frac{(p_{i,r} - q_r)^2}{h_r^2} \right)}}{\sum_{i=1}^{P}}
\end{align*}
$$

• $\mathbf{p}_i$ denotes the 3D coordinates of point number $i$,
• $\mathbf{p}_{i,s}$ indicates the 2 spatial coordinates of that point,
• $\mathbf{p}_{i,r}$ stands for the intensity (range) value of the point,
• $q_s$ and $q_r$ are the current spatial and range coordinates, resp., of the window, and
• $h_s$ and $h_r$ are the corresponding window sizes.

Practical Considerations

• Large, homogeneous regions do no have a single point of convergence.
• In such cases, the mean shift clustering can be followed by region merging and in split-and-merge.
• Unlike split-and-merge, mean shift does not always generate contiguous regions.
• Thus, after mean shift segmentation, we should remove all regions that are smaller than (i.e., contain fewer pixels than) a certain threshold $\theta$.
• These regions should be merged with their neighbor that is most similar to it in its average intensity.
• (This is also useful for split-and-merge.)

Examples

Depending on the purpose of our system, we may want to **represent** the regions that we detected in a specific way. There are three different basic **classes** of representation:

• array representation,
• hierarchical representation, and
• symbolic representation.

We will now discuss these different approaches.
Array Representation

The simplest way to represent regions is to use a two-dimensional array of the same size as the original image. Each entry indicates the label of the region to which the pixel belongs. This technique is similar to the result of component labeling in binary images. If we have overlapping regions, we can provide a mask for each region. A mask is a binary array in which 1-pixels belong to the region in question and 0-pixels do not.

Hierarchical Representation

Hierarchical representation of images allows us to represent the visual information at multiple resolutions. This is useful for a variety of tasks in computer vision. For example, the presence and number of certain objects can be more efficiently detected at low resolution. Then object identification can be performed at high resolution. We will look at two techniques: pyramids and quad trees.

Pyramids

A pyramid is a set of pixel arrays, each of which shows the same image at a different resolution. The level 0 image at the top of the pyramid consists of only 1 pixel. Each successive level has twice as many pixels in x and y direction as the level above it. Therefore, an n×n image is represented by levels 0 to log₂n. The following slide shows the pyramid representation of a 512×512 image.

Quad Trees

Quad trees are a method for hierarchically representing binary images. The root of a quad tree stands for the entire image, and its 4 children represent the 4 quadrants of the image. If a quadrant is completely white or completely black, the corresponding node is marked white or black, respectively. Otherwise, the node is expanded, and its children represent the quadrants within the original quadrant. This process is repeated recursively until all pixels are represented.
Symbolic Representation

For each region, represent information such as:
- enclosing rectangle
- moments
- compactness
- mean and variance of intensity
- etc.

In other words, symbolic representation of a region means that we describe its characteristics using a set of (typically numerical or categorical) parameters.

Data Structures

How can we represent regions in computer programs?
We will look at two different approaches: **Region adjacency graphs, picture trees, and super grids.**

Region adjacency graphs are just undirected graphs, where each vertex represents a region. The edges in the graph indicate which regions are adjacent to each other.

Region Adjacency Graphs

[Image of region adjacency graphs]

Picture Trees

**Picture trees** are a hierarchical structure for storing regions. The rule for representation is the "is-included-in" relationship: All child regions are included in their parent region.

Super Grids

If we want to represent region boundaries in an array, it is unclear to which region the pixels on the boundary belong (left: image; center: boundary). If we expand our grid (right image), the boundary can be indicated without ambiguity.

Chain Codes

An interesting way of describing a contour is using **chain codes.**
A chain code is an ordered list of **local orientations** (directions) of a contour.
These local directions are given through the locations of neighboring pixels, so there are only **eight different possibilities.**
We assign each of these directions a code, e.g.:

```
<table>
<thead>
<tr>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>
```
Chain Codes

Then we start at the first edge in the list and go **clockwise** around the contour.
We add the code for each edge to a list, which becomes our chain code.

What happens if in our chain code for a given contour we replace every code \( n \) with \( (n \mod 8) + 1 \)?
The contour will be (approximately) **rotated** clockwise by 45 degrees.
We can also compute the derivative of a chain code, also referred to as **difference code**.
Difference codes are **rotation-invariant** descriptors of contours.
Some features of regions, such as their corners or areas, can be directly computed from chain or difference codes.

Chain Codes

Slope Representation

The **slope representation** of a contour (also called the \( \Psi \)-s plot) is like a chain code for continuous data.
Along the contour, we plot the **tangent** \( \Psi \) versus **arc length** \( s \).
Then horizontal line segments in the \( \Psi \)-s plot correspond to straight line segments in the contour.
Straight line segments at other orientations in the \( \Psi \)-s plot correspond to circular arcs in the contour.
Other segments in the \( \Psi \)-s plot correspond to other curve primitives in the contour.

Note that in this plot not the actual arc length is used, but its horizontal projection:

Slope and Curvature Density Functions

The **slope density function** is the histogram of all slopes (tangent angles) in a contour.
This function can be used for recognizing objects.
We can also use the derivative of the slope representation, which we can call the **curvature representation**.
Its histogram is the **curvature density function**.
The curvature density function can also be used to recognize objects.
Its advantage is its **rotation invariance**, i.e., matching two curvature density functions determines similarity even if the two objects have different orientations.

Examples of Slope Density Functions