CSci 4968 and 6270
Computational Vision,
Fall Semester, 2011-2012
Lectures 6-8, Segmentation

The Segmentation Problem

- Problem: divide the image into regions of pixels that are somehow the same.
- This vague definition has more precise meaning in specific contexts:
  - Identify regions of airway, blood vessel, tissue, heart and bone in lung CT scans.
  - Segment cells in microscopy images
  - Identify moving objects in video.
- Humans do well at segmenting photographic images into natural/coherent objects; computer vision techniques still lag behind.

Simple Approach Using Thresholding

- Compute a threshold statistically or establish a threshold based on physical properties.
- Applying thresholding: pixels whose values are above the threshold are "foreground" pixels; others are "background".
  - Note that "value" does not have to be just the intensity of the pixel. It could be a texture measure or even a motion measure.
- Morphology and connected components labeling for extracting regions.

Critique of "Simple Approach"

- Decision is made pixel-by-pixel,
- Region information is only added during the morphology and connected-components stages
- Only a single threshold is used

Approaches Covered Here

- Three approaches:
  1. K-means clustering, and some variations
  2. Mean-shift clustering
  3. Graph-cuts binary segmentation
- The first two are "local" methods, whereas the third is "global" (image-wide).
- The most important method missing from our discussion is *active contours*. See Szelisk, Section 5.1

Readings

- K-means: Szeliski, Section 5.3
- Comaniciu and Meer, "Mean Shift: A Robust Approach Toward Feature Space Analysis", *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24:5, 2002, 603-619.
- The material on using graph cuts for segmentation is largely drawn from the paper, Boykov and Funka-Lea, Graph Cuts and Efficient N-D Image Segmentation, *International Journal of Computer Vision* 70:2, 109-131, 2006. In your first reading of this paper you should focus primarily on Sections 2 through 4 because Section 1 includes an extensive background discussion that will not make sense to beginners in the field.

Feature Vectors for Clustering

For clustering, the image will be described in terms of a discrete set of "feature vectors":

- $\mathbf{x}_i$ is a color vector at pixel index i — usually not RGB, but L*u*v or some other approximately uniform color space vector, or
- $\mathbf{x}_i$ is a concatenation of the color vector and the location vector at pixel i

K-means

Goal: Given a positive integer K, compute K "mean" vectors $\{\boldsymbol{\mu}_k\}$, also known as the "cluster centers", and determine the assignment of each pixel vector $\mathbf{x}_i$ to its nearest cluster center.

Iterative Procedure:

1. Generate K initial values of the mean vectors $\{\boldsymbol{\mu}_k\}$.
   - The simplest way to do this is to select k values from the data set. Sometimes this is done several times, with the following iterative procedure applied each time.
2. Repeat the following:
   (a) Assign each vector $\mathbf{x}_i$ to its closest mean vector $\boldsymbol{\mu}_k$. More precisely, form K sets
   $$\mathcal{X}_k = \{\mathbf{x}_i \mid k = \underset{1 \le j \le K}{\operatorname{argmin}} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2\} \tag{1}$$
   (b) Recompute the mean vectors for each k:
   $$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{\mathbf{x}_i \in \mathcal{X}_k} \mathbf{x}_i \tag{2}$$
   where $N_k = |\mathcal{X}_k|$.
3. Until the assignment of points to sets does not change from one iteration to the next.
4. The sets $\mathcal{X}_k$ form the clusters. The pixels belonging to a cluster form a segment.

We will work through some examples in class.

Fuzzy K-means

Instead of assigning each pixel to a class, make a "fuzzy" assignment by assigning a weight, $w_{i,k}$, in each iteration, where

- i is the pixel index, and
- k is the cluster index.

In the simplest form, $w_{i,k}$ is inversely proportional to the distance from the pixel to the cluster center.

Iterative Procedure:

- Generate K initial values of the mean vectors $\{\boldsymbol{\mu}_k\}$.
- Repeat the following:
  1. For each point i and for each cluster, k, compute the normalized weight

     $$w_{i,k} = \frac{1/(\epsilon + d_{i,k})}{\sum_{m=1}^{K} 1/(\epsilon + d_{i,m})}$$

     where

     $$d_{i,k} = \|\mathbf{x}_i - \boldsymbol{\mu}_k\|$$

     and ϵ is a small positive constant.
  2. Recompute the mean vectors for each k:

     $$\boldsymbol{\mu}_k = \frac{\sum_{i=1}^{N} w_{i,k}\mathbf{x}_i}{\sum_{i=1}^{N} w_{i,k}}$$

- Until change in the weights is sufficiently small.
- Each pixel vector $\mathbf{x}_i$ chooses the cluster that has the maximum weight. Algebraically, this is

  $$\mathcal{X}_k = \{i | w_{i,k} \geq w_{i,m}, m = 1, \ldots, K\}.$$

Discussion

- Demos will be given in class.
- Computational cost is $O(KN)$ per iteration in each case.
- Each requires
  - An initial estimate
  - A specified number of clusters
- Solutions: multiple initializations, and automatic selection of K. Usually people just make sure K is set a bit too large
- Significance of distance:

  - Color space should be uniform
  - Mixing pixel location measurements and colors is particular difficult because the units are different; it can also be powerful.

- A more formal approach to Fuzzy K-means may be derived using Gaussian Mixture Models (covariance matrices in addition to mean vectors) and the Expectation Maximization (EM) algorithm.

Mean-Shift — Overview

- "Mode-seeking" algorithm: by this we mean that we pixel vector $\mathbf{x}_i$ attempts to "climb the hill" of its histogram until it reaches the peak (the "mode").
- All pixels that climb to the same peak are in the same segment. We will look at examples in class.
- The mean-shift algorithm does this by computing local weighted averages and "shifting" in the direction of this average, or "mean".
- The weighting functions are called "kernels".
- Preliminary comparison to K-means:
  - In K-means, the number of clusters is fixed and points pick their best cluster (as well as influencing the estimate of the position of the center of the cluster).
  - In mean-shift, the width of the weighting function is an important determinant of how many clusters there are — but the number of clusters still depends on the data!

Kernel Density Approximation

- At each data point we form a kernel function (or Parzen window)

$$K(\mathbf{x} - \mathbf{x}_i) = k\left(\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{h^2}\right) \tag{3}$$

  where
  - $k(r)$ is a window function such as a Gaussian that integrates to 1, and
  - h is a kernel width.
- This is usually used for approximating a probability function, The kernel width is typically quite narrow for an approximation, but is much larger for the mean-shift algorithm we use.
- The probability density function for N data points is approximated by

$$f(x) = \frac{1}{N}\sum_i K(\mathbf{x} - \mathbf{x}_i). \tag{4}$$

- In the mean-shift we will use these kernels more simply — as weighting functions!

Mode Seeking — The Mean Shift Algorithm

- Given the kernel density approximation, the goal is find the modes (peaks) of this distribution.
- This is quite difficult to do directly.
- Instead we compute the gradient of f and use this to derive a hill-climbing method:

  - each point will "climb" its hill to the local mode,
  - the stopping points for the points are the modes.

- We can show that the gradient is proportional to

$$\left[\sum_i G(\mathbf{x} - \mathbf{x}_i)\right]\mathbf{m}(\mathbf{x}), \tag{5}$$

  where

$$\mathbf{m}(\mathbf{x}) = \frac{\sum_i \mathbf{x}_i G(\mathbf{x} - \mathbf{x}_i)}{\sum_i G(\mathbf{x} - \mathbf{x}_i)} - \mathbf{x} \tag{6}$$

  is the "mean-shift". Note that this is **just a shift in the direction of the weighted average of the points in the neighborhood.**

- Here

$$g(r) = -k'(r) \tag{7}$$

  and

$$G(\mathbf{x} - \mathbf{x}_i) = g\left(\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{h^2}\right) \tag{8}$$

- Algorithmically, the mean-shift procedure is very simple. Our goal is to "climb the hill" starting at a given location, $\mathbf{y}_0$, to find the nearest mode. The update equation is just

$$\mathbf{y}_{k+1} = \frac{\sum_i \mathbf{x}_i G(\mathbf{y}_k - \mathbf{x}_i)}{\sum_i G(\mathbf{y}_k - \mathbf{x}_i)} \tag{9}$$

  This provably converges under mild conditions on K and therefore k and G and g.

- Typically,

$$k(r) = \max(0, 1 - r), \tag{10}$$

  which is called the "Epanechnikov" kernel, or

$$k(r) = \exp\left(\frac{-1}{2}r^2\right). \tag{11}$$

- Intuitively, the idea of mean shift is simply to iteratively replace the value of the mean by the weighted average of the nearby points.

Using the Mean-Shift Algorithm in Segmentation

- As we discussed earlier, the data values $\mathbf{x}_i$ are 5-component vectors.
- Intuitively, the idea is to run mean-shift starting from each pixel and then iterate until convergence. All pixels that converge to the same mode are in the same segment.
- Faster algorithms have been proposed.
- In order to combine pixel values and color values, which have different units and different statistics, Comaniciu and Meer split the kernel into a product of a pixel location kernel and a color kernel.

- Sizes of the kernels are controlled by two separate parameters, with a third parameter to determine a minimum region segment size to allow.
  - Smaller parameter values yield smaller segments because the kernel is smaller and the resulting density is less smooth.
- Mean-shift has also been used for tracking and for stereo
- Sometimes small values of h are given to produce an oversegmentation of the data to generate "super-pixels" as the starting point for other algorithms.

Overview of Graph-Cuts for Segmentation

- First intuition is based on a quick comparison to edge detection: we will focus primarily on the region instead of the boundary. More specifically, like thresholding, graph-cuts will label which pixels correspond to the object and which correspond to the background. The boundary, if desired, is found by looking for neighboring pixels having different labels.
- The problem will be formulated as a discrete / combinatorial problem: we consider only image pixels and consider two possible labels per pixel.
- The exact solution is computed using max-flow / min-cut algorithms.
- Our discussion:
  - Algorithmic notion of a graph cut
  - Energy formulation
  - Map to s/t graph, and solve using min-cut
  - Show that the solution is correct.
  - Impose hard constraints
  - Additional techniques and applications

Graph-Cuts

- Given is a connected graph $G = (V, E)$, where V is a set of vertices, and E is a set of edges with associated weights.
  - The weights are $w(v_i, v_j) > 0$, for $v_i \in V$, $v_j \in V$ and $(v_i, v_j) \in E$.
- A "cut" in the graph is a subset of edges $E' \subset E$ such that the graph $G' = (V, E - E')$ has two connected components.
- A "min-cut" is a cut such that the sum of the weights E' is minimal over all possible cuts. A min-cut is not necessarily unique.
- A source/terminal or "s/t" graph is one where there are two special vertices marked as the source, s, and as the terminal, t.
- A cut of an s/t graph is one where $s \in V_1$ and $t \in V_2$. Stated another words, it is a cut where there is no longer a path from s to t in the graph.
- Minimizing the cut in an s/t graph is equivalent to maximizing the "flow" from s to t, where
  - The "flow" along any edge $(v_i, v_j) \in E$ must be $\leq w(v_i, v_j)$,
  - Except for s and t, the "flow" into a vertex must be equal to the flow out of a vertex. This introduces a directionality to the edges.
  - The goal is to maximize the total flow into t.
- We will tend to think of our problem in terms of minimizing the cut.

Energy Formulation for Segmentation

Energy (objective function) minimization problems are very common in computer vision. We will look at an energy formulation for segmentation and then show how to turn this into a graph-cuts formulation.

- Think of the image as a set, $\mathcal{P}$, of pixel locations, p.
- A segmentation is an assignment of labels to each pixel p. In other words:

$$a(p) = \begin{cases} 1 & p \text{ is an object pixel} \\ 0 & p \text{ is a background pixel} \end{cases} \tag{12}$$

- We form a vector $\mathbf{a}$ of these labelings, with one entry in the vector per pixel.
- We can write an "energy", or "objective function" for this labeling as

$$E(\mathbf{a}) = \lambda \sum_{p \in \mathcal{P}} R_p(a(p)) \; + \sum_{(p,q) \in \mathcal{N}} B_{p,q} \delta_{a(p),a(q)}. \tag{13}$$

  where

  - R_p is a cost function for assigning label $a(p)$ to pixel p,
  - $\mathcal{N}$ is a set of neighboring pixels,
  - $\delta_{a(p),a(q)}$ is defined as

$$\delta_{a(p),a(q)} = \begin{cases} 0 & a(p) = a(q) \\ 1 & a(p) \neq a(q) \end{cases} \tag{14}$$

    In other words, $\delta_{a(p),a(q)}$ is the energy cost of having two adjacent pixels with different labels.

  - $B_{p,q}$ is the cost of placing a boundary between pixel locations p and q.
- This type of energy formulation is quite new to us, so we will spend time discussing it in class.

Approach to Formulating the Energy Terms Based on Statistics

This is a bit of an aside

- We sometimes talk about the "probability" (conditional probability) of an image "event".
  - For example we might want to know the probability that a particular voxel intensity, i, is from bone in a CT scan.
- We can "learn" this probability in several ways:
  - By computing a histogram of all pixels that are bone and normalizing it to form a probability function.

– By fitting a probability model such as a Gaussian:

$$\text{pr}(i) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(i-i_0)^2}{2\sigma^2}}$$

where i_0 and σ are the parameters of the distribution.

- The "energy" we will formulate is proportional to the negative logarithm of this probabilty.
  - For example, with our Gaussian probabilty the energy is proportional to

$$E(i) = \frac{(i-i_0)^2}{2\sigma^2} + \frac{\log(2\pi)}{2} + \log\sigma$$

The key here is the $(i-i_0)^2$ term which is the same form as the "error" penalty we "pay" in the least-squares estimate.

- The relationship between energy and probabilty is deep and important. We tend to "learn" probabilities and "estimate" with energies.

Formulating the Energy Terms

Recall our energy equation for labeling **a**

$$E(\mathbf{a}) = \lambda \sum_{p\in\mathcal{P}} R_p(a(p)) \; + \sum_{(p,q)\in\mathcal{N}} B_{p,q}\delta_{a(p),a(q)}.$$

- To formulate $R_p(a(p))$ we need a statistical model of the object appearance and of the background appearance. We write these as probability functions of the intensities:
  - For the object it will be $Pr(I(p) \mid a(p) = 1)$.
  - For the background it will be $Pr(I(p) \mid a(p) = 0)$.
- As just described, these probability models can be learned, which is often the case in medical image segmentation, or they can be gathered from simple manual selection of example objects and background regions and computing histograms
- Once we have these terms, we write

$$R_p(1) = -\ln Pr(I(p) \mid a(p) = 1) \tag{15}$$

$$R_p(0) = -\ln Pr(I(p) \mid a(p) = 0) \tag{16}$$

- For $B_{p,q}$, Boykov and Funka-Lea suggest

$$B_{p,q} = \exp\left(\frac{-(I(p)-I(q))^2}{2\sigma^2}\right)\frac{1}{\text{dist}(p,q)} \tag{17}$$

which encourages discontinuities at locations where intensities differ significantly.

Looking At Some Examples

We will construct three simple examples in class to help illustrate this:

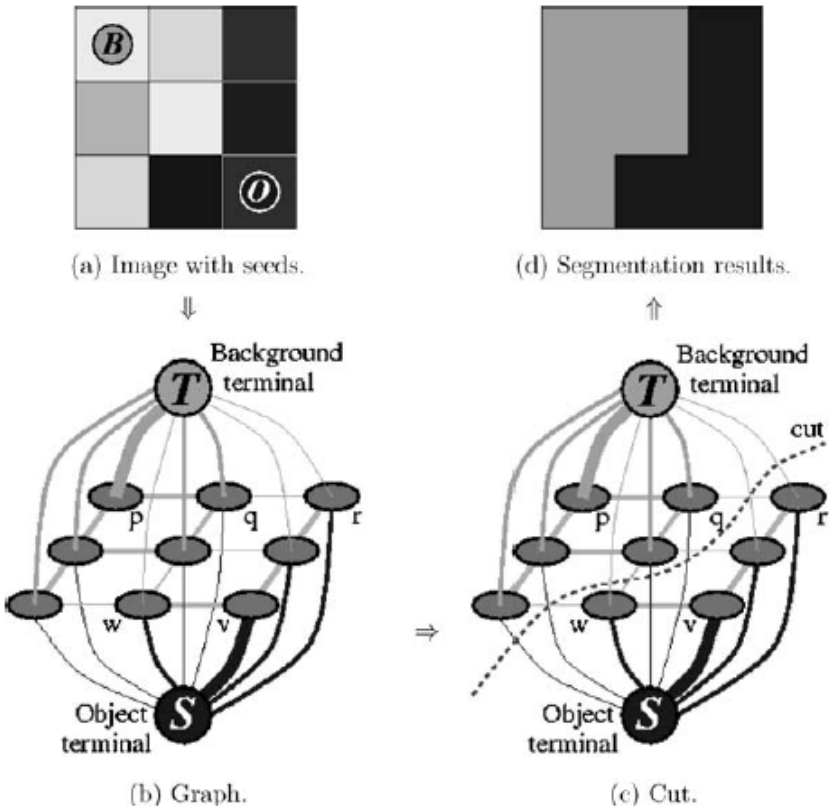
- No boundary energy, which reduces the problem to thresholding.
- A one dimensional image.
- A two dimensional region with a small hole in it.

Mapping Into A Graph

- We want to map this into s/t (source/terminal) graph for solution by a min-cut / max-flow algorithm.
- Form a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.
- Vertices: form one vertex for each pixel; add two special vertices, S and T:
  - S is the "source" and corresponds to the object
  - T is the "terminal" and corresponds to the background
  - Our cut must separate S and T — placing them into two separate connected components.

  Therefore,
  $$\mathcal{V} = \mathcal{P} \bigcup \{S, T\}. \tag{18}$$
- Edges in this undirected graph:
  - For each $p \in \mathcal{P}$, connect p to S and connect p to T. These edges are called *t-links*.
  - Each neighbor relation $\{p, q\}$ forms an edge. These edges are called *n-links*.

  Therefore,
  $$\mathcal{E} = \mathcal{N} \bigcup_{p \in \mathcal{P}} \{\{p, S\}, \{p, T\}\} \tag{19}$$
- An example is shown in the following figure from the Boykov and Funka-Lea paper:



(a) Image with seeds. (d) Segmentation results.

(b) Graph. (c) Cut.

- Now, the all-important edge weights:

edge	**weight (cost)**	**for**
$\{p, q\}$	$B(p, q)$	$\forall \{p, q\} \in \mathcal{N}$
$\{p, S\}$	$-\lambda \ln Pr(I(p) \mid a(p) = 0)$	$\forall p \in \mathcal{P}$
$\{p, T\}$	$-\lambda \ln Pr(I(p) \mid a(p) = 1)$	$\forall p \in \mathcal{P}$

  Note the reversal here: with the cost of the connection between p and the background terminal being the cost, in the energy function, of labeling p as a foreground node.

Max-Flow / Min-Cut to Solve the Segmentation Problem

- As described above, a cut $\mathcal{C} \subset \mathcal{E}$ of an s/t graph is a subset of the edges such that S and T are in different connected components.
- The weight of a cut is

$$|C| = \sum_{e \in \mathcal{C}} w(e) \tag{20}$$

- Getting a bit more technical, a cut $\mathcal{C}$ is said to be *feasible* if
  - for each $p \in \mathcal{P}$ exactly one t-link is in $\mathcal{C}$;
  - for each $\{p, q\} \in \mathcal{N}$, $\{p, q\} \in \mathcal{C}$ if and only if p and q are t-linked to different terminals.
- Note how a feasible cut $\mathcal{C}$ defines our segmentation quite simply:

  - if p is connected to T following the cut, p is a background pixel;
  - if p is connected to S following the cut, p is an object pixel.
- $\hat{\mathcal{C}}$ is a *min-cut* if no other cut has lower weight.
- The Ford-Fulkerson algorithm can be used to solve for the min-cut in polynomial time.
- We then need to prove:
  - $\hat{\mathcal{C}}$ is feasible, which means it determines our segmentation.
  - $|\hat{\mathcal{C}}|$ yields the minimum energy segmentation.
- To be sure that you are comfortable with what is happening here, we will look at the case where the neighborhood cost $B_{p,q} = 0$.

Adding Hard Constraints

- In an interactive scenario, we can identify a set of pixels $\mathcal{O}$ that are definitely inside the segmented object and a set of pixels $\mathcal{B}$ that are definitely outside the segmented object.
- These can be used to establish object and background intensity statistics.
- These pixels are easily added to the s/t graph:
  - For $p \in \mathcal{B}$, $w(\{p, T\}) = \infty$ and $w(\{p, S\}) = 0$.
  - For $p \in \mathcal{O}$, $w(\{p, T\}) = 0$ and $w(\{p, S\}) = \infty$.

  Note that the weight of the edge to the terminal that p must remain connected to gets infinite weight so that it can not be cut.
- It is easily shown that there must be a closed "contour" of n-links cut that separate the vertices in $\mathcal{B}$ from the vertices in $\mathcal{O}$.

Applications, Additional Techniques and Limitations

- Interactive segmentation for photo editing and medical image segmentation.
- Add directed edges. These can be used to enforce different types of weights based on intensity properties.
- Fast editing, where the entire max-flow / min-cut need not be re-estimated.
- Multi-way segmentation:
  - Intuitively we can think of applying the graph-cuts algorithm for each type of label vs. all other labels and then picking the best.
  - The trick is adjudicating between the choices when more than one label seems to apply (as opposed to background).
  - The core graph-cuts algorithm is used several-times over as the central computation engine.