# Introduction to Computer Vision

Alvin Lin

August 2018 - December 2018

## Segmentation

Image processing involves an image as input, with the output being another image. Image analysis outputs measurements from an input image, and image understanding outputs a high level description from an input image. Humans tend to identify regions of images in groups, so the goal of segmentation is to identify groups of pixels or regions that are related and go together perceptually.

#### Goals of Segmentation

- Obtaining primitives for other tasks
- Perceptual organization/recognition
- Image manipulation for graphics related tasks.

Segmentation can be used to group together similar pixels for efficiency of further processing (superpixels). It can also separate images into coherent objects. Oversegmentation and undersegmentation is always an issue here.

### High Level Approaches

Bottom-up segmentation involves grouping tokens with similar features, while top-down segmentation involves grouping tokens that likely belong to the same object.

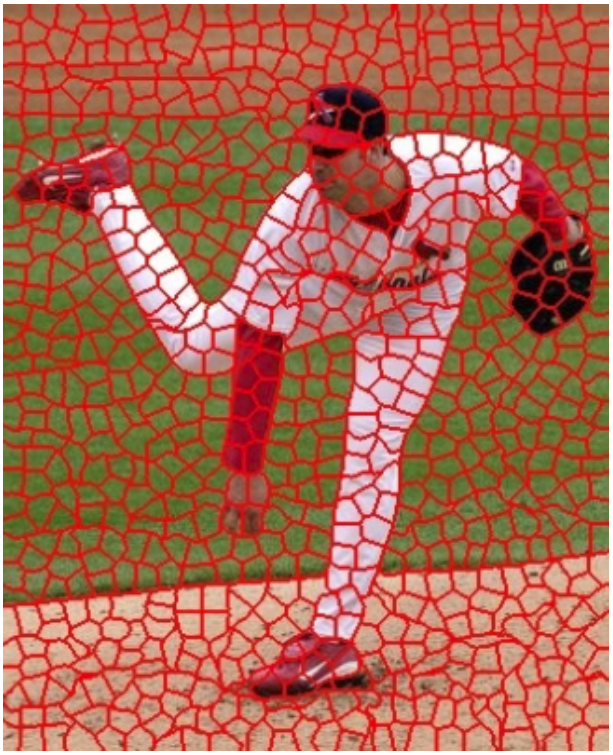

Figure 1: An example of a bottom-up unsupervised approach

An alternative approach to this is recognition, or being able to separate the image into coherent "objects". As humans, we can infer a lot of context using our knowledge to separate out the different elements in an image to identify the foreground objects. Methods that take the recognition route use a top-down approach of grouping tokens that likely belong to the same object.

## K-means

Segmentation can be performed through clustering, graph partitioning, and labeling. With clustering, similar points are grouped together and represented with a single token. The K-means clustering algorithm can be used to cluster together similar regions. This will usually find cluster centers that minimize conditional variance and is simple to implement, however it is prone to local minima and it is often difficult to choose $k$.

Traditionally, Euclidean distance is used for the K-means clustering algorithm, but various other metrics can be used.

- L1 norm
$$d(\vec{x}, \vec{y}) = \sum_{i=1}^{n} \|x_i - y_i\|$$
- L2 norm
$$d(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$
- L-infinity
$$d(\vec{x}, \vec{y}) = max(|x_i - y_i|, \ldots, |x_n - y_n|)$$
- Mahalanobis distance
$$d(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{n} \frac{(x_i - y_i)^2}{\sigma_i^2}}$$
- Cosine distance
$$d(\vec{x}, \vec{y}) = \cos(\theta) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \|\vec{y}\|}$$

K-means is usually simple to implement but can be very slow since it is $O(kNd)$ for $Nd$-dimensional points clustered into $k$ centers.

## K-medoids

Instead of representing a cluster with the mean of its members, we select a member of the cluster to represent it and minimize cluster dissimilarity. This is most applicable when the mean is not meaningful. This is much less sensitive to outliers than k-means.

This is a simple and fast alternative to K-means that is easy to implement, but again depends very heavily on choosing $k$ correctly. It is rarely used for pixel segmentation.

## Mean Shift Clustering

The mean shift algorithm seeks modes of the given set of points.

1. Choose a kernel and bandwidth.

2. For each point, center a window on that point and compute the mean of the data in the search window. Center the search window at the new mean location and repeat until the window convergess.

3. Assign points that lead to nearby modes to the same cluster.

In addition to storing each pixel's position and color, we can store additional features such as gradients and texture when performing the mean shift algorithm. This is typically a very good general purpose segmentation algorithm that is flexible in the number and shape of regions while being robus to outliers.

However, a kernel size must be chosen in advance and it is usually not suitable for use with high dimensional features. Thus, it is generally used for multiple segmentations, tracking, clustering, and filtering applications.

## Kernel Density Estimation

Also known as the Parzen-Rosenblatt window method, this is a non-parametric way to estimate the probability density function of a random variable. Inferences about a population are made based only on a finite data sample.

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

Using the function described above with kernel $K$ and bandwidth $h$, we can generate a relatively smooth curve that very closely models the distribution of the image.

## Segmentation as Graph Partitioning

If we create a graph with a node for each pixel in an image, we can link nodes corresponding to adjacent pixels with edges weighted according to the similarity or affinity of the two nodes. There are a few ways to determine affinity based on different attributes.

- Distance
$$d(\vec{x}, \vec{y}) = exp\left\{-\frac{(\vec{x} - \vec{y})^t(\vec{x} - \vec{y})}{2\sigma_d^2}\right\}$$

- Intensity ($I(x)$ is the intensity of the pixel at $\vec{x}$)
$$d(\vec{x}, \vec{y}) = exp\left\{-\frac{(I(\vec{x}) - I(\vec{y}))^t(I(\vec{x}) - I(\vec{y}))}{2\sigma_I^2}\right\}$$

- Color ($c(x)$ is the color of the pixel at $\vec{x}$)

$$d(\vec{x}, \vec{y}) = exp\left\{-\frac{(dist(c(\vec{x}), c(\vec{y})))^2}{2\sigma_c^2}\right\}$$

- Texture ($f(x)$ is a vector of filter outputs at $\vec{x}$)

$$d(\vec{x}, \vec{y}) = exp\left\{-\frac{(f(\vec{x}) - f(\vec{y}))^2(f(\vec{x}) - f(\vec{y}))}{2\sigma_I^2}\right\}$$

After choosing an appropriate feature vector for each pixel and defining an appropriate distance function, we can break the graph into segments by deleting links that have low affinity. Similar pixels should be in the same segments while dissimilar pixels should be in different segments. A graph cut allows us to make a good segmentation by removing enough edges to fully disconnect discrete segments.

One method of doing this is by finding the minimum cut of the graph (for which fast algorithms already exist). This tends to cut off small and isolated segments however, so we can fix this by using a normalized cut where we normalize the cut by the weight of all the edges incident to the segment. We compute a new normalized cut cost

$$cost(A, B) = \frac{w(A, B)}{w(A, V)} + \frac{w(A, B)}{W(B, V)}$$

where $w(A, B)$ is the sum of the weights of all edges between $A$ and $B$. Finding the globally optimal cut is an NP-complete problem, but a relaxed version can be solved using a generalized eigenvalue problem. This is a generic framework that can be used with many different features and affinity formulations but has a high storage requirement and time complexity due to solving a generalized eigenvalue problem of size $n \times n$ where $n$ is the number of pixels.