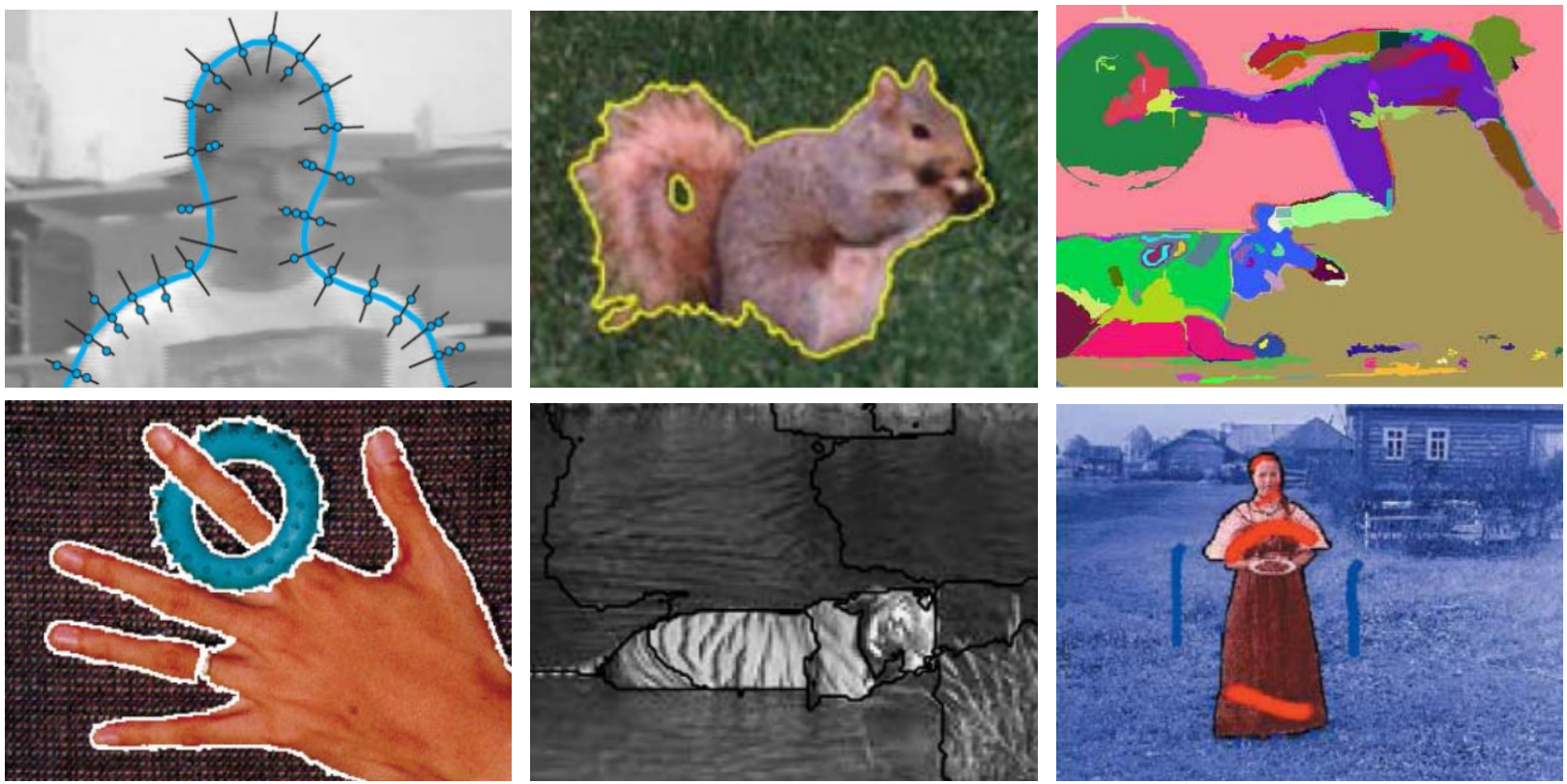
Segmentation
CSE 6367: Computer Vision

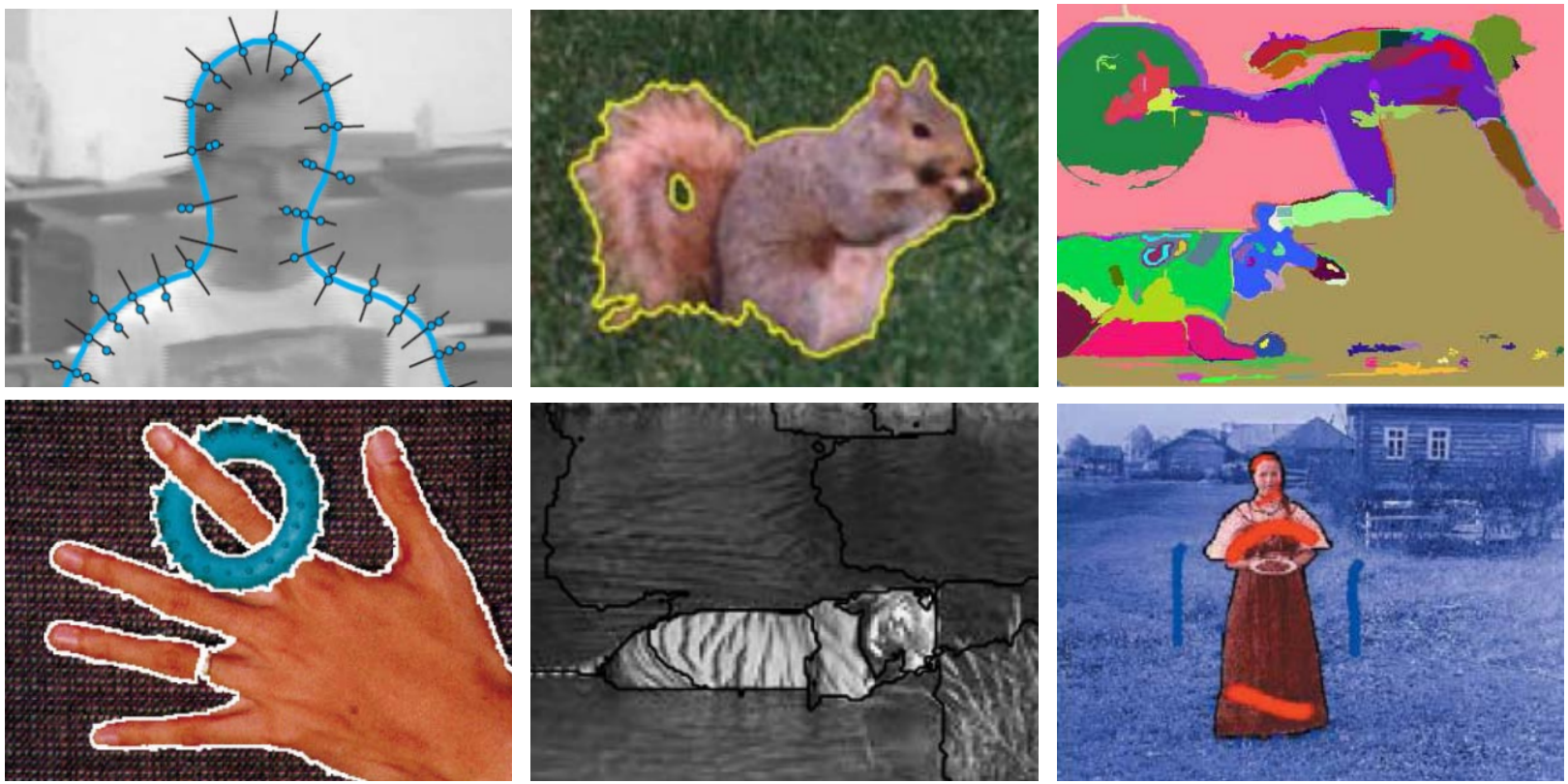
Instructor: William J. Beksi

Introduction



- Image segmentation is the task of finding groups of pixels that "go together"

Introduction



- Image segmentation is the task of finding groups of pixels that "go together"
- In statistics, this problem is known as **cluster analysis** and is a widely studied area with hundreds of different algorithms

Introduction

- In computer vision, image segmentation is one of the oldest and most widely studied problems

Introduction

- In computer vision, image segmentation is one of the oldest and most widely studied problems
- Early techniques tend to use region splitting or merging which correspond to *divisive* and *agglomerative* algorithms in the clustering literature

Introduction

- In computer vision, image segmentation is one of the oldest and most widely studied problems
- Early techniques tend to use region splitting or merging which correspond to *divisive* and *agglomerative* algorithms in the clustering literature
- Recent algorithms often optimize some global criterion, such as intra-region consistency and inter-region boundary lengths or dissimilarity

Locating Boundary Curves in Images

- While lines, vanishing points, and rectangles are commonplace in the man-made world, curves corresponding to object boundaries are even more common, especially in the natural environment

Locating Boundary Curves in Images

- While lines, vanishing points, and rectangles are commonplace in the man-made world, curves corresponding to object boundaries are even more common, especially in the natural environment
- Three approaches to locating such boundary curves in images are **snakes**, **scissors**, and **level set** techniques

Locating Boundary Curves in Images

- While lines, vanishing points, and rectangles are commonplace in the man-made world, curves corresponding to object boundaries are even more common, especially in the natural environment
- Three approaches to locating such boundary curves in images are **snakes**, **scissors**, and **level set** techniques
- All three of these are examples of **active contours** since these boundary detectors iteratively move towards their final solution under the combination of image and optional user-guidance forces

Snakes

- Snakes are a 2D generalization of the 1D energy-minimizing splines

$$\mathcal{E}_{\text{int}} = \int \alpha(s)||\mathbf{f}_s(s)||^2 + \beta(s)||\mathbf{f}_{ss}(s)||^2 ds$$

where s is the arc-length along the curve $f(s) = (x(s), y(s))$ and $\alpha(s)$ and $\beta(s)$ are first- and second-order continuity weighting functions

Snakes

- We can discretize this energy by sampling the initial curve position evenly along its length to obtain

$$E_{\text{int}} = \sum_i \alpha(i)||f(i+1) - f(i)||^2/h^2 + \beta(i)||f(i+1) - 2f(i) + f(i-1)||^2/h^4$$

where h is the step size which can be neglected if we resample the curve along its arc-length after each iteration

Snakes

- In addition to this *internal* spline energy, a snake simultaneously minimizes external image-based and constraint-based potentials

Snakes

- In addition to this *internal* spline energy, a snake simultaneously minimizes external image-based and constraint-based potentials
- The image-based potentials are the sum of several terms

$$\mathcal{E}_{\text{image}} = w_{\text{line}}\mathcal{E}_{\text{line}} + w_{\text{edge}}\mathcal{E}_{\text{edge}} + w_{\text{term}}\mathcal{E}_{\text{term}}$$

where the **line** term attracts the snake to dark ridges, the **edge** term attracts it to strong gradients (edges), and the **term** term attracts it to line terminations

Snakes

- In practice, most systems only use the edge term, which can either be directly proportional to the image gradients

$$E_{\text{edge}} = \sum_i -||\nabla I(\mathbf{f}(i))||^2$$

or to a smoothed version of the image Laplacian

$$E_{\text{edge}} = \sum_i -|(G_\sigma * \nabla^2 I)(\mathbf{f}(i))|^2$$

Snakes

- In interactive applications, a variety of user-placed constraints can also be added, e.g. attractive (spring) forces towards anchor points

$$E_{\text{spring}} = k_i ||\mathbf{f}(i) - \mathbf{d}(i)||^2$$

as well as repulsive $1/r$ ("volcano") forces

Snakes

- In interactive applications, a variety of user-placed constraints can also be added, e.g. attractive (spring) forces towards anchor points

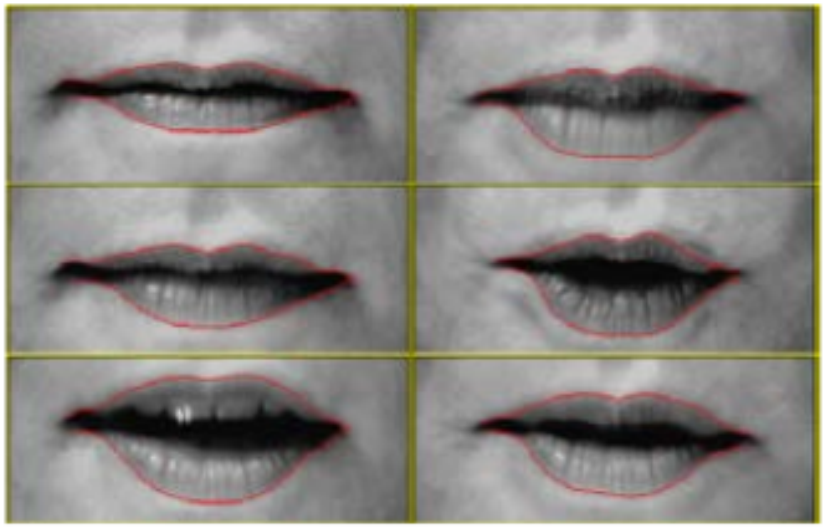
$$E_{\text{spring}} = k_i ||\mathbf{f}(i) - \mathbf{d}(i)||^2$$

  as well as repulsive $1/r$ ("volcano") forces

- As the snakes evolve by minimizing their energy, they often "wiggle" and "slither", which accounts for their popular name

Snakes

(a)



(b)

- Snakes: (a) the "snake pit" for interactively controlling shape; (b) lip tracking

Scissors

- Active contours allow a user to roughly specify a boundary of interest and have the system evolve the contour towards a more accurate location as well as track it over time

Scissors

- Active contours allow a user to roughly specify a boundary of interest and have the system evolve the contour towards a more accurate location as well as track it over time
- However, the results of this curve evolution may be unpredictable and require additional user-based hints to achieve the desired result

Scissors

- Active contours allow a user to roughly specify a boundary of interest and have the system evolve the contour towards a more accurate location as well as track it over time
- However, the results of this curve evolution may be unpredictable and require additional user-based hints to achieve the desired result
- **Intelligent scissors** is an alternative approach in which the contour is optimized in real time as the user is drawing

Scissors

- To compute the optimal curve path (live wire), the image is first preprocessed to associate low costs with edges that are likely to be boundary elements

Scissors

- To compute the optimal curve path (live wire), the image is first preprocessed to associate low costs with edges that are likely to be boundary elements
- Next, as the user traces a rough curve, the method continuously recomputes the lowest cost path between the starting **seed point** and the current mouse location using Dijkstra's algorithm

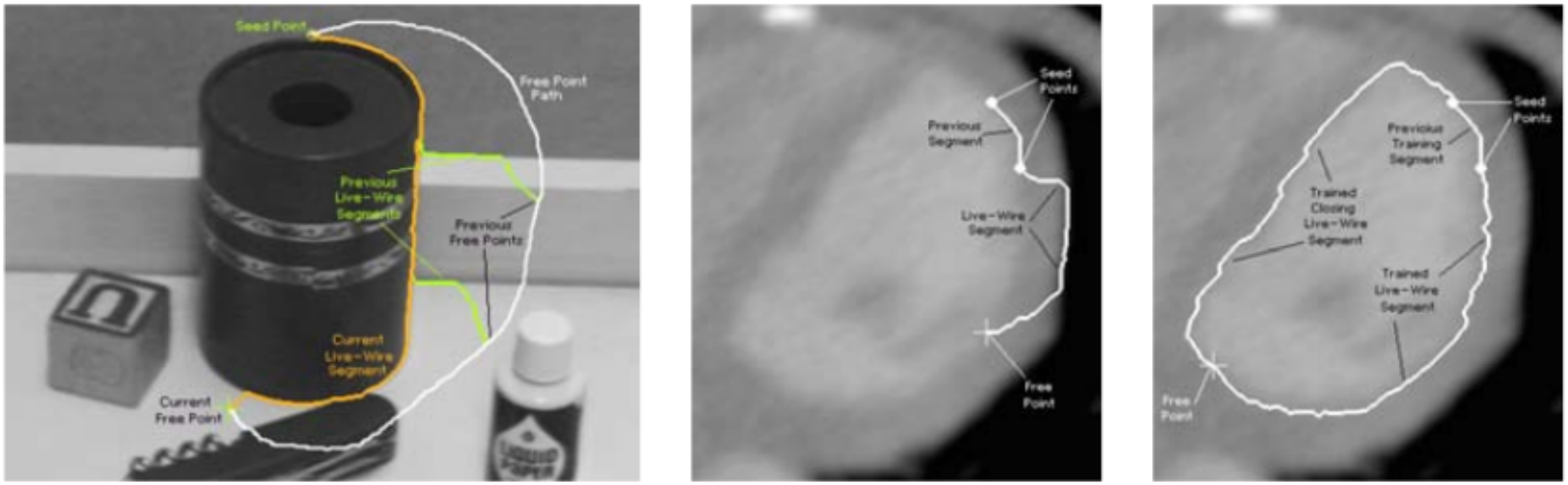
Scissors

- In order to prevent jumping around unpredictably, the curve will "freeze" (reset the seed point) after a period of inactivity

Scissors

- In order to prevent jumping around unpredictably, the curve will "freeze" (reset the seed point) after a period of inactivity
- To keep the live wire from jumping onto adjacent contours, the method "learns" the intensity profile under the current optimized curve, and uses this to preferentially keep the wire moving along the same boundary

Scissors



(a) (b) (c)

- Intelligent scissors: (a) as the mouse traces the white path, the scissors follow the orange path along the object boundary; (b) regular scissors can sometimes jump to a stronger (incorrect) boundary; (c) after training to the previous segment, similar edge profiles are preferred

Level Sets

- A limitation of active contours based on parametric curves of the form $\mathbf{f}(s)$ is that it is challenging to change the topology of the curve as it evolves

Level Sets

- A limitation of active contours based on parametric curves of the form $\mathbf{f}(s)$ is that it is challenging to change the topology of the curve as it evolves
- In addition, if the shape changes drastically, curve reparameterization may also be required

Level Sets

- An alternative representation for such closed contours is use a level set, where the zero crossing(s) of a characteristic function (or signed distance function) define the curve

Level Sets

- An alternative representation for such closed contours is use a level set, where the zero crossing(s) of a characteristic function (or signed distance function) define the curve
- Level sets evolve to fit and track objects of interest by modifying the underlying embedding function, $\phi(x, y)$, instead of the curve $\mathbf{f}(s)$

Level Sets

- An alternative representation for such closed contours is use a level set, where the zero crossing(s) of a characteristic function (or signed distance function) define the curve
- Level sets evolve to fit and track objects of interest by modifying the underlying embedding function, $\phi(x, y)$, instead of the curve $\mathbf{f}(s)$
- To reduce the amount of computation required, only a small strip (frontier) around the locations of the current zero crossings needs to be updated at each step

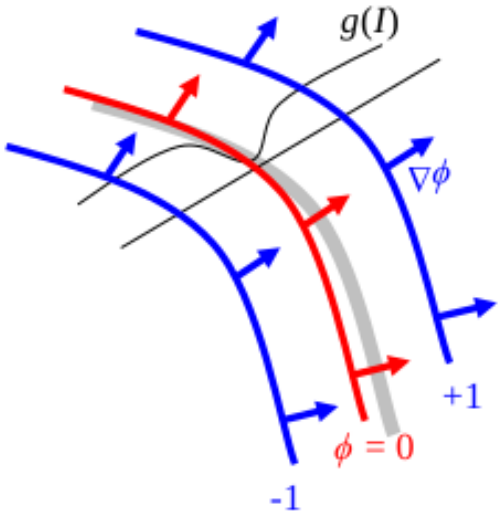
Level Sets

- An example of an evolution equation is the **geodesic active contour**

$$\frac{d\phi}{dt} = |\nabla\phi|\left(g(I)\frac{\nabla\phi}{|\nabla\phi|}\right)$$
$$= g(I)|\nabla\phi|\text{div}\left(\frac{\nabla\phi}{|\nabla\phi|}\right) + \nabla g(I)\cdot\nabla\phi \quad (1)$$

where $g(I)$ is a generalized version of the snake edge potential

Level Sets



- Level set evolution for a geodesic active contour; the embedding function ϕ is updated based on the curvature of the underlying surface modulated by the edge/speed function $g(I)$, as well was the gradient of $g(I)$, thereby attracting it to strong edges

Level Sets

- To get an intuitive sense of the curve's behavior, assume that the embedding function ϕ is a signed distance function away from the curve (e.g. $|\phi| = 1$)

Level Sets

- To get an intuitive sense of the curve's behavior, assume that the embedding function ϕ is a signed distance function away from the curve (e.g. $|\phi| = 1$)
- The first term in Equation (1) moves the curve in the direction of its curvature, i.e. it acts to straighten the curve under the influence of $g(I)$

Level Sets

- To get an intuitive sense of the curve's behavior, assume that the embedding function ϕ is a signed distance function away from the curve (e.g. $|\phi| = 1$)
- The first term in Equation (1) moves the curve in the direction of its curvature, i.e. it acts to straighten the curve under the influence of $g(I)$
- The second term moves the curve down the gradient of $g(I)$, encouraging the curve to migrate towards minima of $g(I)$

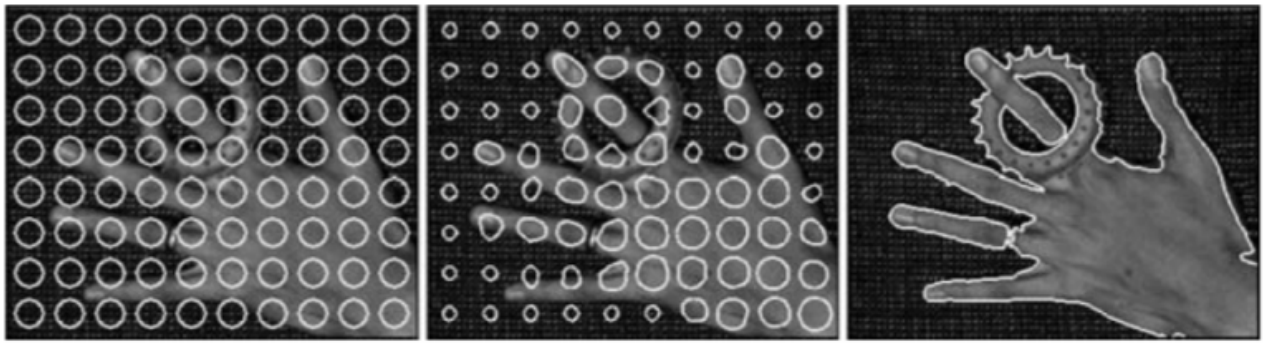
Level Sets

- While this level set formulation can readily change topology, it is still susceptible to local minima since it is based on local measurements such as image gradients

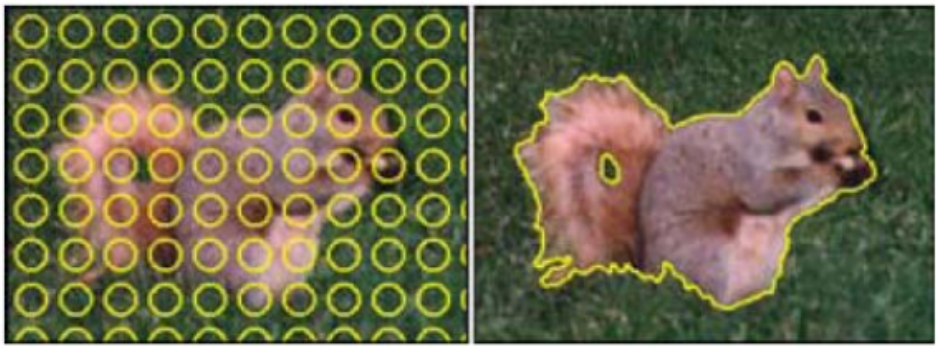
Level Sets

- While this level set formulation can readily change topology, it is still susceptible to local minima since it is based on local measurements such as image gradients
- An alternative is to recast the problem in a segmentation framework where the energy measures the consistency of the image statistics (e.g. color, texture, motion) inside and outside the segmented regions

Level Sets



(a)



(b)

- Level set segmentation: (a) grayscale image segmentation and (b) color image segmentation, initial circles evolve towards an accurate segmentation adapting their topology as they evolve

Splitting and Merging Images

- The simplest technique for segmenting a grayscale image is to select a threshold and then compute connected components

Splitting and Merging Images

- The simplest technique for segmenting a grayscale image is to select a threshold and then compute connected components
- Unfortunately, a single threshold is rarely sufficient for the whole image due to lighting and intra-object variations

Splitting and Merging Images

- The simplest technique for segmenting a grayscale image is to select a threshold and then compute connected components
- Unfortunately, a single threshold is rarely sufficient for the whole image due to lighting and intra-object variations
- As a result, a number of algorithms proceed either by recursively splitting the whole image into pieces based on region statistics, or conversely, merging pixels and regions together in a hierarchical fashion

Watershed

- A technique related to thresholding, since it operates on a grayscale image, is **watershed** computation

Watershed

- A technique related to thresholding, since it operates on a grayscale image, is **watershed** computation
- This technique segments an image into several **catchment basins**, which are regions of an image (interpreted as a height field or landscape) where rain would flow into the same lake

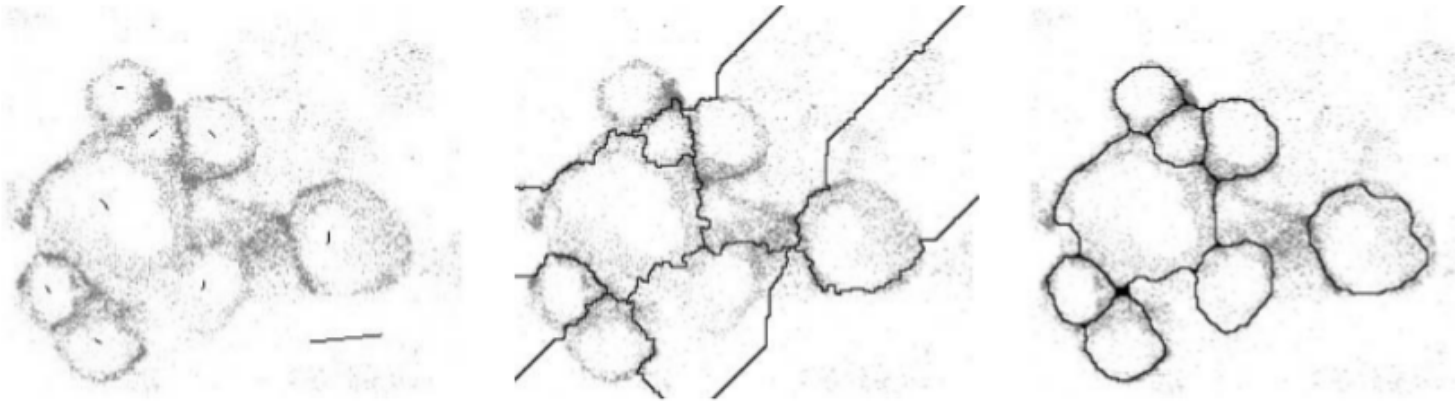
Watershed

- An efficient way to compute such regions is to start flooding the landscape at all of the local minima and to label regions wherever differently evolving components meet

Watershed

- An efficient way to compute such regions is to start flooding the landscape at all of the local minima and to label regions wherever differently evolving components meet
- The whole algorithm can be implemented using a priority queue of pixels and breadth-first search

Watershed



- Locally constrained watershed segmentation: (a) original confocal microscopy image with marked seeds (line segments); (b) standard watershed segmentation; (c) locally constrained watershed segmentation

Watershed

- Images rarely have dark regions separated by lighter ridges, therefore watershed segmentation is usually applied to a smoothed version of the gradient magnitude image (which also makes it usable with color images)

Watershed

- Images rarely have dark regions separated by lighter ridges, therefore watershed segmentation is usually applied to a smoothed version of the gradient magnitude image (which also makes it usable with color images)
- Since these boundaries are what active contours usually follow, a watershed segmentation can first be applied to the image followed by active contour segmentation

Watershed

- Unfortunately, watershed segmentation associates a unique region with each local minimum, which can lead to over segmentation

Watershed

- Unfortunately, watershed segmentation associates a unique region with each local minimum, which can lead to over segmentation
- Watershed segmentation is therefore often used as part of an interactive system, where the user first marks seed locations that correspond to the centers of desired components

Region Splitting (Divisive Clustering)

- Splitting an image into successively smaller regions is one of the oldest techniques in computer vision

Region Splitting (Divisive Clustering)

- Splitting an image into successively smaller regions is one of the oldest techniques in computer vision
- One such technique (Ohlander, Price, and Reddy, 1978) first computes a histogram for the whole image and then finds a threshold that best separates the large peaks in the histogram

Region Splitting (Divisive Clustering)

- Splitting an image into successively smaller regions is one of the oldest techniques in computer vision
- One such technique (Ohlander, Price, and Reddy, 1978) first computes a histogram for the whole image and then finds a threshold that best separates the large peaks in the histogram
- More recent splitting algorithms often optimize some metric of intra-region similarity and inter-region dissimilarity

Region Merging (Agglomerative Clustering)

- Region merging techniques also date back to the beginnings of computer vision

Region Merging (Agglomerative Clustering)

- Region merging techniques also date back to the beginnings of computer vision
- In data clustering, algorithms can link clusters together based on the distance between their closest points (single-link clustering), their farthest points (complete-link clustering), or something in between

Region Merging (Agglomerative Clustering)

- Region merging techniques also date back to the beginnings of computer vision
- In data clustering, algorithms can link clusters together based on the distance between their closest points (single-link clustering), their farthest points (complete-link clustering), or something in between
- Images can be segmented into **superpixels**, a useful preprocessing step, where adjacent regions whose average color difference is below a threshold are merged together

Graph-based Segmentation

- **Graph-based merging segmentation** (Felzenszwalb and Huttenlocher, 2004) uses *relative dissimilarities* between regions to determine which ones should be merged and produces an algorithm that provably optimizes a global grouping metric

Graph-based Segmentation

- **Graph-based merging segmentation** (Felzenszwalb and Huttenlocher, 2004) uses *relative dissimilarities* between regions to determine which ones should be merged and produces an algorithm that provably optimizes a global grouping metric
- It starts with a pixel-to-pixel dissimilarity measure $w(e)$ that measures, for example, intensity differences between $\mathcal{N}_8$ neighbors

Graph-based Segmentation

- For any region R, its *internal difference* is defined as the largest edge weight in the region's minimum spanning tree

$$Int(R) = \min_{e \in MST(R)} w(e)$$

Graph-based Segmentation

- For any region R, its *internal difference* is defined as the largest edge weight in the region's minimum spanning tree

$$Int(R) = \min_{e \in MST(R)} w(e)$$

- For any two adjacent regions with at least one edge connecting their vertices, the difference between these regions is defined as the minimum weight edge connecting the two regions

$$Dif(R_1, R_2) = \min_{e=(v_1,v_2)|v_1 \in R_1, v_2 \in R_2} w(e)$$

Graph-based Segmentation

- The algorithm merges any two adjacent regions whose difference is smaller than the minimum internal difference of these two regions

$$MInt(R_1, R_2) = \min(Int(R_1) + \tau(R_1), Int(R_2) + \tau(R_2))$$

where $\tau(R)$ is a heuristic region penalty set to $k/|R|$, but it can be set to any application-specific measure of region goodness

Graph-based Segmentation

- The algorithm merges any two adjacent regions whose difference is smaller than the minimum internal difference of these two regions

$$MInt(R_1, R_2) = \min(Int(R_1) + \tau(R_1), Int(R_2) + \tau(R_2))$$

  where $\tau(R)$ is a heuristic region penalty set to $k/|R|$, but it can be set to any application-specific measure of region goodness

- By merging regions in decreasing order of the edges separating them, segmentations are provably produced that are neither too fine nor too course

Graph-based Segmentation

(a) (b) (c)

- Graph-based merging segmentation: (a) a grayscale image that is successfully segmented into three regions; (b) input grayscale image and (c) the resulting segmentation using an $\mathcal{N}_8$ pixel neighborhood

Probabilistic Aggregation

- A probabilistic algorithm (Alpert, Galun, Basri et al., 2007) uses two cues, gray-level and texture similarity, to perform merging

Probabilistic Aggregation

- A probabilistic algorithm (Alpert, Galun, Basri et al., 2007) uses two cues, gray-level and texture similarity, to perform merging
- The gray-level similarity between regions R_i and R_j is based on the *minimal external difference* from other neighboring regions

$$\sigma^+_{local} = \min(\Delta^+_i, \Delta^+_j)$$

where $\Delta^+_i = \min_k |\Delta_{ik}|$ and Δ_{ik} is the difference in average intensities between regions R_i and R_k

Probabilistic Aggregation

- This is compared to the *average intensity difference*

$$\sigma^{-}_{local} = \frac{\Delta^{-}_{i} + \Delta^{-}_{j}}{2}$$

where $\Delta^{-}_{i} = \sum_k(\tau_{ik}\Delta ik)/\sum_k(\tau_{ik})$ and τ_{ik} is the boundary length between regions R_i and R_k

Probabilistic Aggregation

- This is compared to the *average intensity difference*

$$\sigma^{-}_{local} = \frac{\Delta^{-}_{i} + \Delta^{-}_{j}}{2}$$

where $\Delta^{-}_{i} = \sum_k(\tau_{ik}\Delta ik)/\sum_k(\tau_{ik})$ and τ_{ik} is the boundary length between regions R_i and R_k

- The texture similarity is defined using relative differences between histogram bins of simple oriented Sobel filter respones

Probabilistic Aggregation

- This is compared to the *average intensity difference*

$$\sigma^{-}_{local} = \frac{\Delta^{-}_{i} + \Delta^{-}_{j}}{2}$$

  where $\Delta^{-}_{i} = \sum_k(\tau_{ik}\Delta ik)/\sum_k(\tau_{ik})$ and τ_{ik} is the boundary length between regions R_i and R_k

- The texture similarity is defined using relative differences between histogram bins of simple oriented Sobel filter respones

- The pairwise statistics σ^{+}_{local} and σ^{-}_{local} are used to compute the likelihoods p_{ij} that two regions should be merged

Probabilistic Aggregation

- Merging proceeds in a hierarchical fashion where a subset of nodes $C \subset V$ that are (collectively) *strongly coupled* to all of the original nodes (regions) are used to define the problem at a coarser scale, where strong coupling is defined as

$$\frac{\sum_{j \in C} p_{ij}}{\sum_{j \in V} p_{ij}} > \phi$$

with ϕ usually set to 0.2

Probabilistic Aggregation

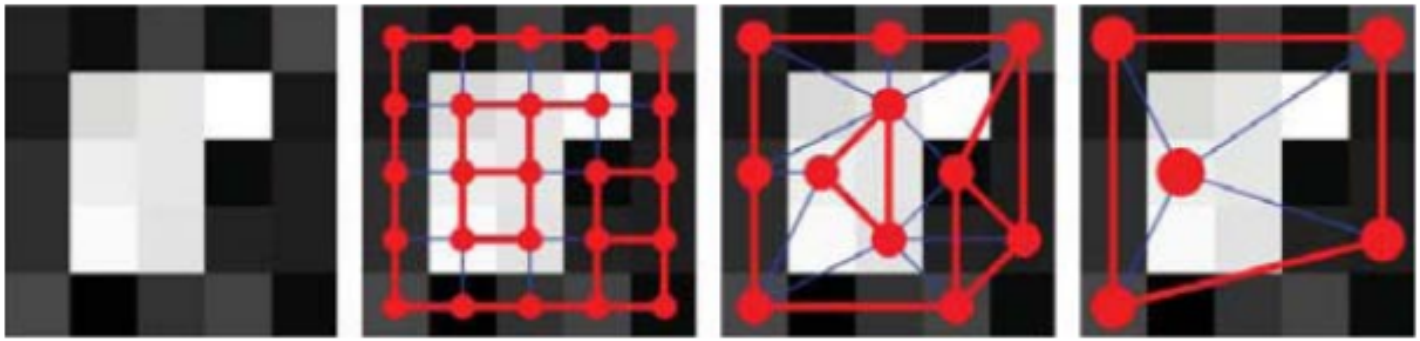
- Merging proceeds in a hierarchical fashion where a subset of nodes $C \subset V$ that are (collectively) *strongly coupled* to all of the original nodes (regions) are used to define the problem at a coarser scale, where strong coupling is defined as

$$\frac{\sum_{j \in C} p_{ij}}{\sum_{j \in V} p_{ij}} > \phi$$

  with ϕ usually set to 0.2

- After a segmentation has been identified at a coarser level, the exact memberships of each pixel are computed by propagating coarse-level assignments to their finer-level "children"

Probabilistic Aggregation



- Coarse to fine node aggregation in segmentation by weighted aggregation: (a) original gray-level pixel grid; (b) inter-pixel couplings where thicker lines indicate stronger couplings; (c) after one level of coarsening where each original pixel is strongly coupled to one of the coarse-level pixels; (d) after two levels of coarsening

Mode Finding Techniques

- **K-means** and **mixtures of Gaussians** are techniques that model the feature vectors associated with each pixel (e.g. color and position) as samples from an unknown probability density function and then try to find clusters (modes) in this distribution

Mode Finding Techniques

- **K-means** and **mixtures of Gaussians** are techniques that model the feature vectors associated with each pixel (e.g. color and position) as samples from an unknown probability density function and then try to find clusters (modes) in this distribution
- These techniques use a **parametric** model of the density function to find clusters in an image, i.e. they assume the density is the superposition of a small number of simpler distributions (e.g. Gaussians) whose locations (centers) and shape (covariance) can be estimated

K-Means

- K-means implicitly models the probability density as a superposition of spherically symmetric distributions, it does not require any probabilistic reasoning or modeling

K-Means

- K-means implicitly models the probability density as a superposition of spherically symmetric distributions, it does not require any probabilistic reasoning or modeling
- Instead, the algorithm is given the number of clusters k it is supposed to find; it then iteratively updates the cluster center location based on the samples that are closest to each center

K-Means

- K-means implicitly models the probability density as a superposition of spherically symmetric distributions, it does not require any probabilistic reasoning or modeling
- Instead, the algorithm is given the number of clusters k it is supposed to find; it then iteratively updates the cluster center location based on the samples that are closest to each center
- The algorithm can be initialized by randomly sampling k centers from the input feature vectors

Mixtures of Gaussians

- In mixtures of Gaussians, each cluster center is augmented by a covariance matrix whose values are re-estimated from the corresponding samples

Mixtures of Gaussians

- In mixtures of Gaussians, each cluster center is augmented by a covariance matrix whose values are re-estimated from the corresponding samples
- Instead of using nearest neighbors to associate input samples with cluster centers, a Mahalanobis distance is used

$$d(\mathbf{x}_i, \boldsymbol{\mu}_k; \Sigma_k) = ||\mathbf{x}_i - \boldsymbol{\mu}_k||_{\Sigma_k^{-1}} = (\mathbf{x}_i - \boldsymbol{\mu}_k)^T \Sigma_k^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_k)$$

where $\mathbf{x}_i$ are the input samples, $\boldsymbol{\mu}_k$ are the cluster centers, and Σ_k are their covariance estimates

Mixtures of Gaussians

- Samples can be associated with the nearest cluster center (a hard assignment of membership) or can be softly assigned to several nearby clusters

Mixtures of Gaussians

- Samples can be associated with the nearest cluster center (a hard assignment of membership) or can be softly assigned to several nearby clusters
- This latter, more commonly used, approach corresponds to iteratively re-estimating the parameters for a mixture of Gaussians density function

$$p(\mathbf{x} \mid \{\pi_k, \boldsymbol{\mu}_k, \Sigma_k\}) = \sum_k \pi_k \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \Sigma_k)$$

where π_k are the mixing coefficients, $\boldsymbol{\mu}_k$ and Σ_k are the Gaussian means and covariances, and

$$\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \Sigma_k) = \frac{1}{|\Sigma_k|} e^{-d(\mathbf{x}, \boldsymbol{\mu}_k; \Sigma_k)}$$

Mixtures of Gaussians

- To iteratively compute (a local) maximum likely estimate for the unknown mixture parameters $\{\pi_k, \boldsymbol{\mu}_k, \Sigma_k)\}$, the expectation maximization (EM) algorithm proceeds in two alternating stages

Mixtures of Gaussians

- To iteratively compute (a local) maximum likely estimate for the unknown mixture parameters $\{\pi_k, \boldsymbol{\mu}_k, \Sigma_k)\}$, the expectation maximization (EM) algorithm proceeds in two alternating stages

  1. The expectation stage estimates the responsibilities

$$z_{ik} = \frac{1}{Z_i}\pi_k \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \Sigma_k) \quad \text{with} \quad \sum_k z_{ik} = 1$$

     which are the estimates of how likely a sample $\mathbf{x}_i$ was generated from the kth Gaussian cluster

Mixtures of Gaussians

- To iteratively compute (a local) maximum likely estimate for the unknown mixture parameters $\{\pi_k, \boldsymbol{\mu}_k, \Sigma_k)\}$, the expectation maximization (EM) algorithm proceeds in two alternating stages

  1. The expectation stage estimates the responsibilities

$$z_{ik} = \frac{1}{Z_i}\pi_k \mathcal{N}(\mathbf{x} \,|\, \boldsymbol{\mu}_k, \Sigma_k) \quad \text{with} \quad \sum_k z_{ik} = 1$$

     which are the estimates of how likely a sample $\mathbf{x}_i$ was generated from the kth Gaussian cluster

  2. The maximization stage updates the parameter values

$$\boldsymbol{\mu}_k = \frac{1}{N_k}\sum_i z_{ik}\mathbf{x}_i, \quad \Sigma_k = \frac{1}{N_k}\sum_i z_{ik}(\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T, \quad \pi_k = \frac{N_k}{N}$$

     where $N_k = \sum_i z_{ik}$ is an estimate of the number of sample points assigned to each cluster

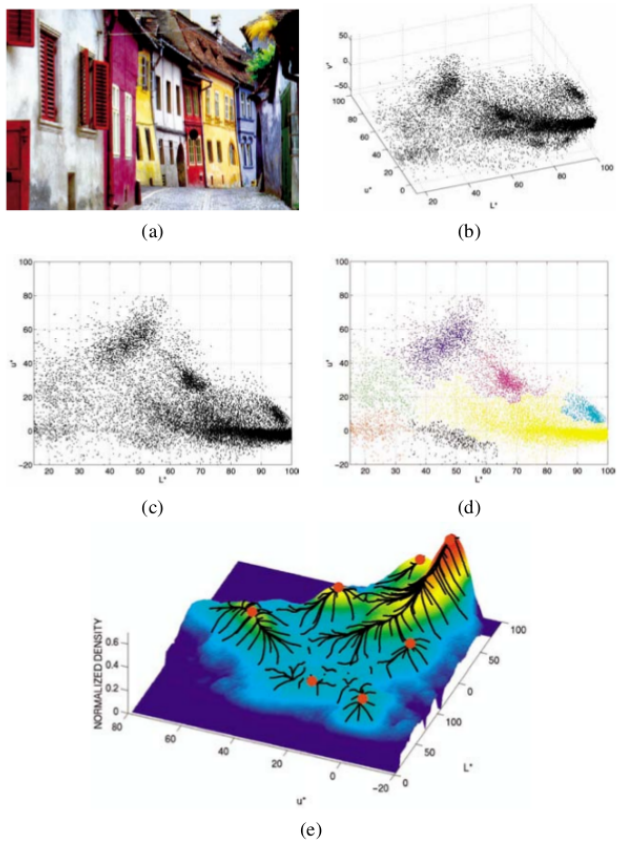
Mean Shift

- While k-means and mixtures of Gaussians use a parametric form to model the probability density function being segmented, **mean shift** implicitly models this distribution using a smooth continuous non-parametric model

Mean Shift

- While k-means and mixtures of Gaussians use a parametric form to model the probability density function being segmented, **mean shift** implicitly models this distribution using a smooth continuous non-parametric model
- The key to mean shift is a technique for efficiently finding peaks in this high-dimensional data without every computing the complete function explicitly

Mean Shift



- Mean-shift image segmentation: (a) input color image; (b) pixels plotted in L*u*v* space; (c) L*u* space distribution; (d) clustered results after 159 mean shift procedures; (e) corresponding trajectories with peaks marked as red dots

Mean Shift

- We can estimate the density function given a sparse set of samples by smoothing and convolving it with a fixed kernel of width h

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

where $\mathbf{x}_i$ are the input samples and $k(r)$ is the kernel function

Mean Shift

- We can estimate the density function given a sparse set of samples by smoothing and convolving it with a fixed kernel of width h

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

  where $\mathbf{x}_i$ are the input samples and $k(r)$ is the kernel function

- This is known as the **kernel density estimation** or **Parzen window** technique

Mean Shift

- We can estimate the density function given a sparse set of samples by smoothing and convolving it with a fixed kernel of width h

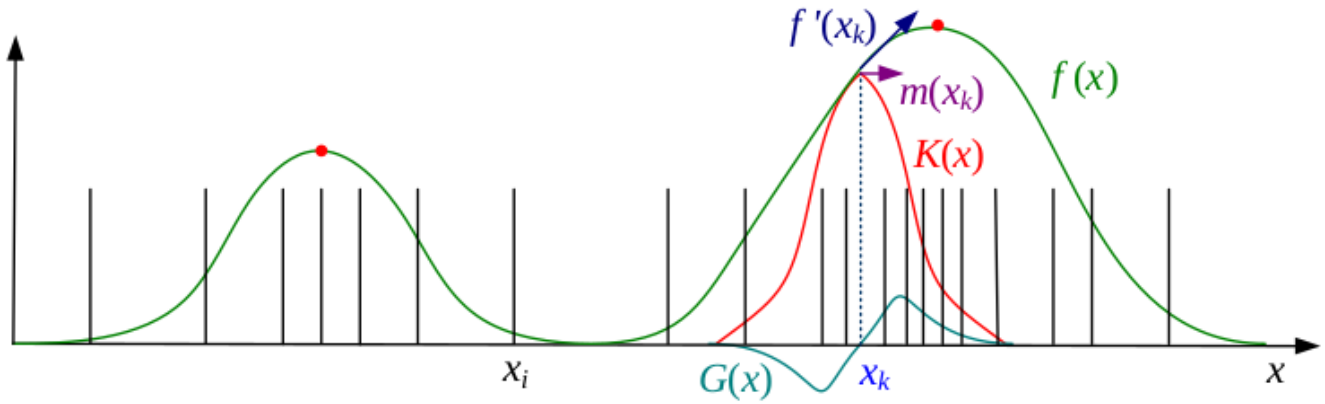
$$f(\mathbf{x}) = \sum_i K(\mathbf{x} - \mathbf{x}_i) = \sum_i k\left(\frac{||\mathbf{x} - \mathbf{x}_i||^2}{h^2}\right)$$

  where $\mathbf{x}_i$ are the input samples and $k(r)$ is the kernel function

- This is known as the **kernel density estimation** or **Parzen window** technique

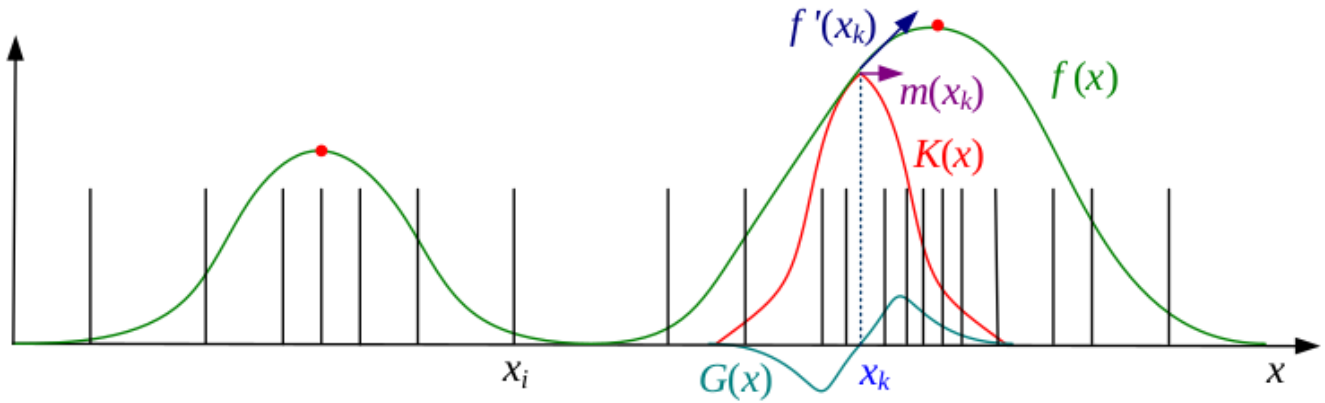
- Once we have computed $f(\mathbf{x})$ we can find its local maxima using gradient ascent or some other optimization technique

Mean Shift



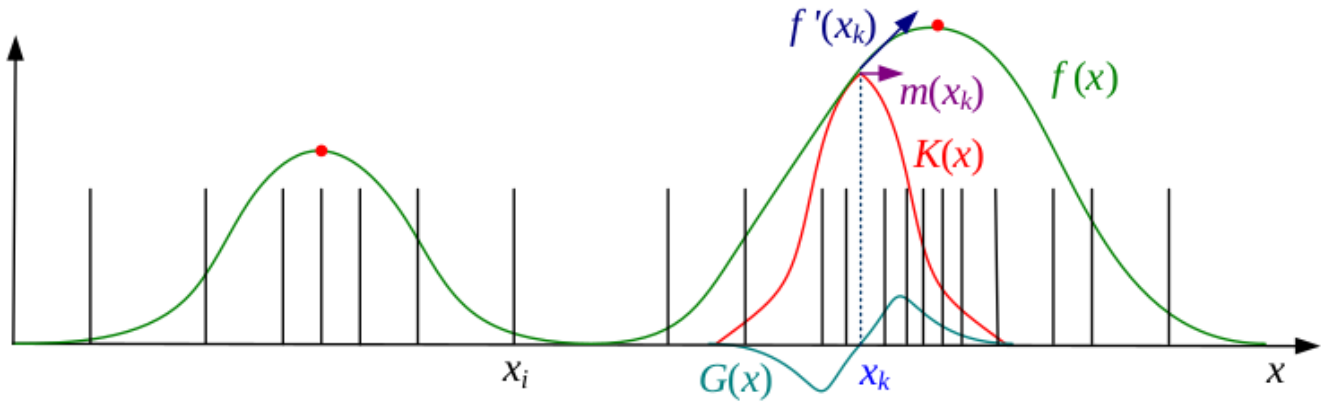
- The kernel density estimate $f(\mathbf{x})$ is obtained by convolving the sparse set of input samples $\mathbf{x}_i$ with the kernel function $K(\mathbf{x})$

Mean Shift



- The kernel density estimate $f(\mathbf{x})$ is obtained by convolving the sparse set of input samples $\mathbf{x}_i$ with the kernel function $K(\mathbf{x})$
- The derivative of this function, $f'(\mathbf{x})$, can be obtained by convolving the inputs with the derivative kernel $G(\mathbf{x})$

Mean Shift



- The kernel density estimate $f(\mathbf{x})$ is obtained by convolving the sparse set of input samples $\mathbf{x}_i$ with the kernel function $K(\mathbf{x})$
- The derivative of this function, $f'(\mathbf{x})$, can be obtained by convolving the inputs with the derivative kernel $G(\mathbf{x})$
- Estimating the local displacement vectors around a current estimate $\mathbf{x}_k$ results in the mean shift vector $\mathbf{m}(\mathbf{x}_k)$

Mean Shift

- The problem with this "brute force" approach is that for higher dimensions it becomes computationally prohibitive to evaluate $f(\mathbf{x})$ over the complete search space

Mean Shift

- The problem with this "brute force" approach is that for higher dimensions it becomes computationally prohibitive to evaluate $f(\mathbf{x})$ over the complete search space
- Instead, mean shift uses a technique known as *multiple restart gradient descent*

Mean Shift

- Starting at some guess for a local maximum, $\mathbf{y}_k$, which can be a random input data point $\mathbf{x}_i$, mean shift computes the gradient of the density estimate $f(\mathbf{x})$ at $\mathbf{y}_k$ and takes an uphill step in that direction

Mean Shift

- Starting at some guess for a local maximum, $\mathbf{y}_k$, which can be a random input data point $\mathbf{x}_i$, mean shift computes the gradient of the density estimate $f(\mathbf{x})$ at $\mathbf{y}_k$ and takes an uphill step in that direction
- The gradient is given by

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

where $g(r) = -k'(r)$ and $k'(r)$ is the first derivative of $k(r)$

Mean Shift

- We can rewrite the gradient of the density function as

$$\nabla f(\mathbf{x}) = \left[\sum_i G(\mathbf{x} - \mathbf{x}_i) \right] \mathbf{m}(\mathbf{x})$$

where the vector

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

is called the *mean shift* since it is the difference between the weighted mean of the neighbors $\mathbf{x}_i$ around $\mathbf{x}$ and the current value of $\mathbf{x}$

Mean Shift

- In the mean-shift procedure, the current estimate of the mode $\mathbf{y}_k$ at iteration k is replaced by its locally weighted mean

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

Mean Shift

- In the mean-shift procedure, the current estimate of the mode $\mathbf{y}_k$ at iteration k is replaced by its locally weighted mean

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

- Convergence to a local maximum of $f(\mathbf{x})$ is achieved under reasonable weak conditions on the kernel $k(r)$, i.e. that it is monotonically decreasing

Mean Shift

- The simplest way to apply mean shift is to start a separate mean-shift mode estimate $\mathbf{y}$ at every input point $\mathbf{x}_i$ and to iterate for a fixed number of steps or until the mean-shift magnitude is below a threshold

Mean Shift

- The simplest way to apply mean shift is to start a separate mean-shift mode estimate $\mathbf{y}$ at every input point $\mathbf{x}_i$ and to iterate for a fixed number of steps or until the mean-shift magnitude is below a threshold
- A faster approach is to randomly resample the input points $\mathbf{x}_i$ and to keep track of each point's temporal evolution, the remaining points can then be classified based on the nearest evolution path

Mean Shift

- Color segmentation can be done by clustering in the *joint domain* of color and location

Mean Shift

- Color segmentation can be done by clustering in the *joint domain* of color and location
- In this approach, the spatial coordinates of the image $\mathbf{x}_s = [x, y]$ (spatial domain) are concatenated with the color values $\mathbf{x}_r$ (range domain) and mean-shift clustering is applied

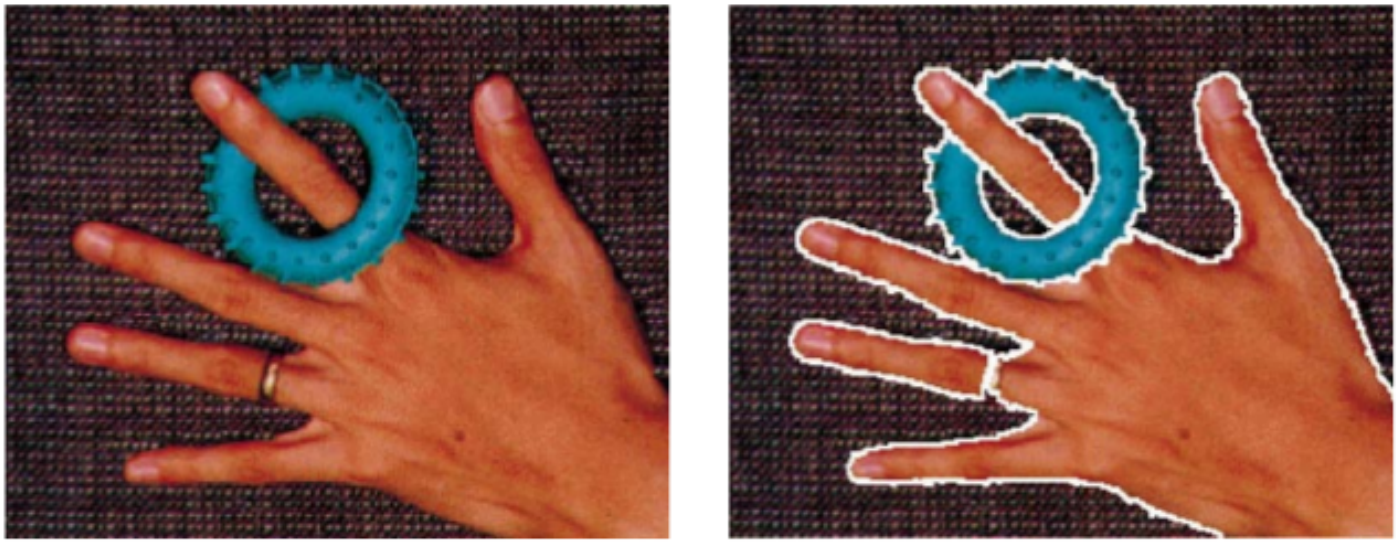
Mean Shift

- Color segmentation can be done by clustering in the *joint domain* of color and location
- In this approach, the spatial coordinates of the image $\mathbf{x}_s = [x, y]$ (spatial domain) are concatenated with the color values $\mathbf{x}_r$ (range domain) and mean-shift clustering is applied
- Since location and color may have different scales, the kernels are adjusted accordingly, e.g.

$$K(\mathbf{x}_j) = k\left(\frac{||\mathbf{x}_s||^2}{h_s^2}\right) k\left(\frac{||\mathbf{x}_r||^2}{h_r^2}\right)$$

where parameters h_s and h_r are used to control the spatial and range bandwidths of the filter kernels

Mean Shift



- Mean-shift color image segmentation with parameters $(h_s, h_r, M) = (16, 19, 40)$

Normalized Cuts

- The **normalized cuts** method examines the *affinities* (similarities) between nearby pixels and tries to separate groups that are connected by weak affinities

Normalized Cuts

- The **normalized cuts** method examines the *affinities* (similarities) between nearby pixels and tries to separate groups that are connected by weak affinities
- The cut between two groups A and B is defined as the sum of all the weights being cut

$$cut(A, B) = \sum_{i \in A, j \in B} w_{ij}$$

where the weights between two pixels (or regions) i and j measure their similarity

Normalized Cuts

- However, using a minimum cut as a segmentation criterion does not result in reasonable clusters since the smallest cuts usually involve isolating a single pixel

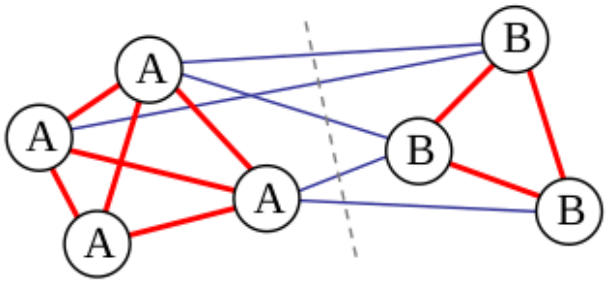
Normalized Cuts

- However, using a minimum cut as a segmentation criterion does not result in reasonable clusters since the smallest cuts usually involve isolating a single pixel
- A better measure of segmentation is the normalized cut, which is defined as

$$Ncut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)}$$

where $assoc(A,A) = \sum_{i \in A, j \in A} w_{ij}$ is the association (sum of all the weights) within a cluster and $assoc(A,V) = assoc(A,A) + cut(A,B)$ is the sum of *all* the weights associated with nodes in A

Normalized Cuts



(a)

	A	B	sum
A	$assoc(A, A)$	$cut(A, B)$	$assoc(A, V)$
B	$cut(B, A)$	$assoc(B, B)$	$assoc(B, V)$
sum	$assoc(A, V)$	$assoc(B, v)$	

(b)

- Sample weighted graph and its normalized cut: (a) a small sample graph and its smallest normalized cut; (b) tabular form of the associations and cuts for this graph

Normalized Cuts

- The cuts and associations can be thought of as area sums in the weight matrix $W = [w_{ij}]$, where the entries of the matrix have been arranged so that the nodes in A come first and the nodes in B come second

Normalized Cuts

- The cuts and associations can be thought of as area sums in the weight matrix $W = [w_{ij}]$, where the entries of the matrix have been arranged so that the nodes in A come first and the nodes in B come second
- Dividing each of these areas by the corresponding row sum results in the normalized cut and association values

Normalized Cuts

- The cuts and associations can be thought of as area sums in the weight matrix $W = [w_{ij}]$, where the entries of the matrix have been arranged so that the nodes in A come first and the nodes in B come second
- Dividing each of these areas by the corresponding row sum results in the normalized cut and association values
- These normalized values better reflect the fitness of a particular segmentation since they look for collections of edges that are weak relative to all of the edges both inside and emanating from a particular region

Normalized Cuts

- Unfortunately, computing the optimal normalized cut is NP-complete and instead we compute a real-valued assignment of nodes to groups

Normalized Cuts

- Unfortunately, computing the optimal normalized cut is NP-complete and instead we compute a real-valued assignment of nodes to groups
- Let $\mathbf{x}$ be the *indicator vector* where $x_i = +1$ iff $i \in A$ and $x_i = -1$ iff $i \in B$

Normalized Cuts

- Unfortunately, computing the optimal normalized cut is NP-complete and instead we compute a real-valued assignment of nodes to groups
- Let $\mathbf{x}$ be the *indicator vector* where $x_i = +1$ iff $i \in A$ and $x_i = -1$ iff $i \in B$
- Let $\mathbf{d} = W\mathbf{1}$ be the row sums of the symmetric matrix W and $D = \text{diag}(\mathbf{d})$ be the corresponding diagonal matrix

Normalized Cuts

- Minimizing the normalized cut over all possible indicator vectors $\mathbf{x}$ is equivalent to minimizing

$$\min_{\mathbf{y}} \frac{\mathbf{y}^T(D - W)\mathbf{y}}{\mathbf{y}^T D\mathbf{y}}$$

where $\mathbf{y} = ((\mathbf{1} + \mathbf{x}) - b(\mathbf{1} - \mathbf{x}))/2$ is a vector consisting of all 1s and $-b$s such that $\mathbf{y} \cdot \mathbf{d} = 0$

Normalized Cuts

- Minimizing this *Rayleigh quotient* is equivalent to solving the generalized eigenvalue system

$$(D - W)\mathbf{y} = \lambda D\mathbf{y}$$

which can be turned into a regular eigenvalue problem

$$(I - N)\mathbf{z} = \lambda\mathbf{z}$$

where $N = D^{-1/2}WD^{-1/2}$ is the *normalized* affinity matrix and $\mathbf{z} = D^{1/2}\mathbf{y}$

Normalized Cuts

- Minimizing this *Rayleigh quotient* is equivalent to solving the generalized eigenvalue system

$$(D - W)\mathbf{y} = \lambda D\mathbf{y}$$

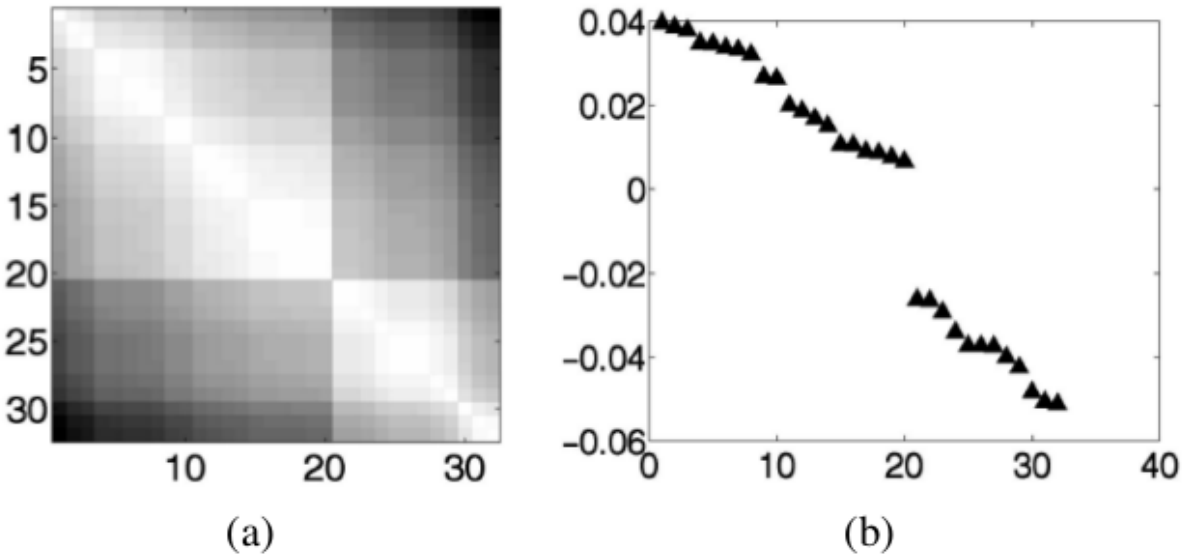
which can be turned into a regular eigenvalue problem

$$(I - N)\mathbf{z} = \lambda\mathbf{z}$$

where $N = D^{-1/2}WD^{-1/2}$ is the *normalized* affinity matrix and $\mathbf{z} = D^{1/2}\mathbf{y}$

- Since these eigenvectors can be interpreted as the large modes of vibration in a spring-mass system, normalized cuts is an example of a **spectral method** for image segmentation

Normalized Cuts



- Sample weight table and its second smallest eigenvector: (a) sample 32×32 weight matrix W; (b) eigenvector corresponding to the second smallest eigenvalue of the generalized eigenvalue problem $(D - W)\mathbf{y} = \lambda D\mathbf{y}$

Normalized Cuts

- After this real-valued eigenvector is computed, the variables corresponding to positive and negative eigenvector values are associated with the two cut components

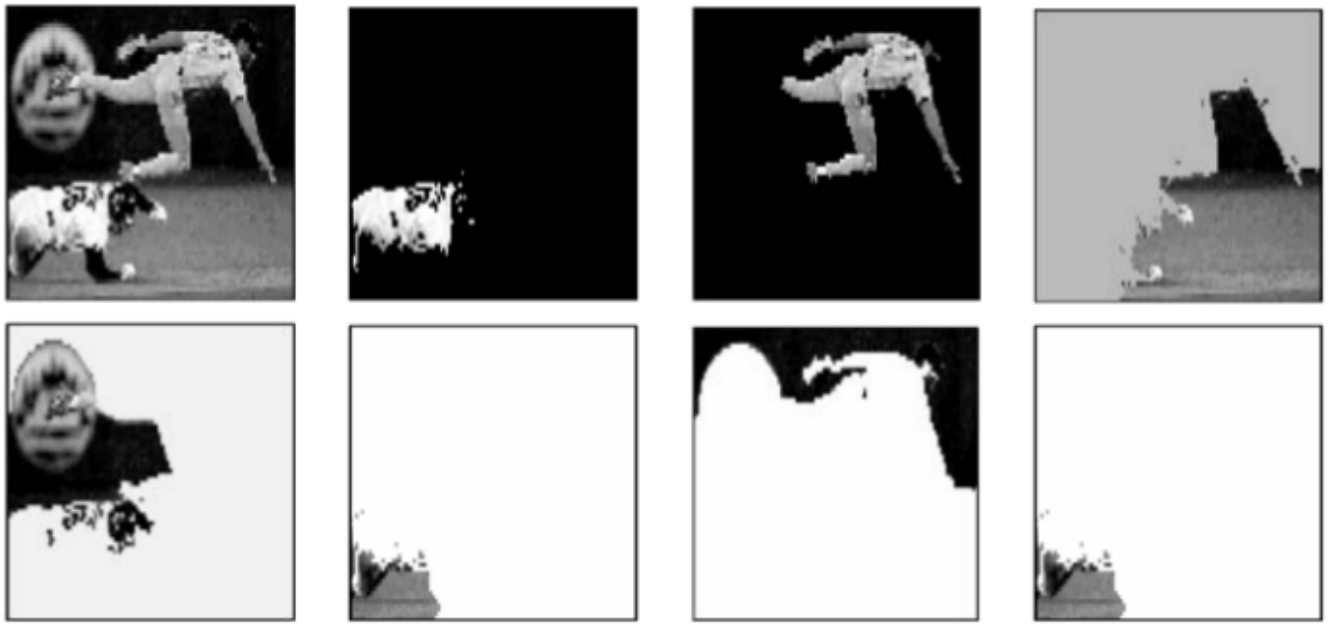
Normalized Cuts

- After this real-valued eigenvector is computed, the variables corresponding to positive and negative eigenvector values are associated with the two cut components
- This process can be further repeated to hierarchically subdivide an image

Normalized Cuts

- After this real-valued eigenvector is computed, the variables corresponding to positive and negative eigenvector values are associated with the two cut components
- This process can be further repeated to hierarchically subdivide an image
- Normalized cuts can be quite slow since it requires the solution of large sparse eigenvalue problems, however much research has been done to accelerate the computations

Normalized Cuts



- Normalized cuts segmentation: the input image and the components returned by the normalized cuts algorithm

Superpixels

- Superpixel algorithms group pixels into perceptually meaningful atomic regions which can be used to replace the rigid structure of a pixel grid

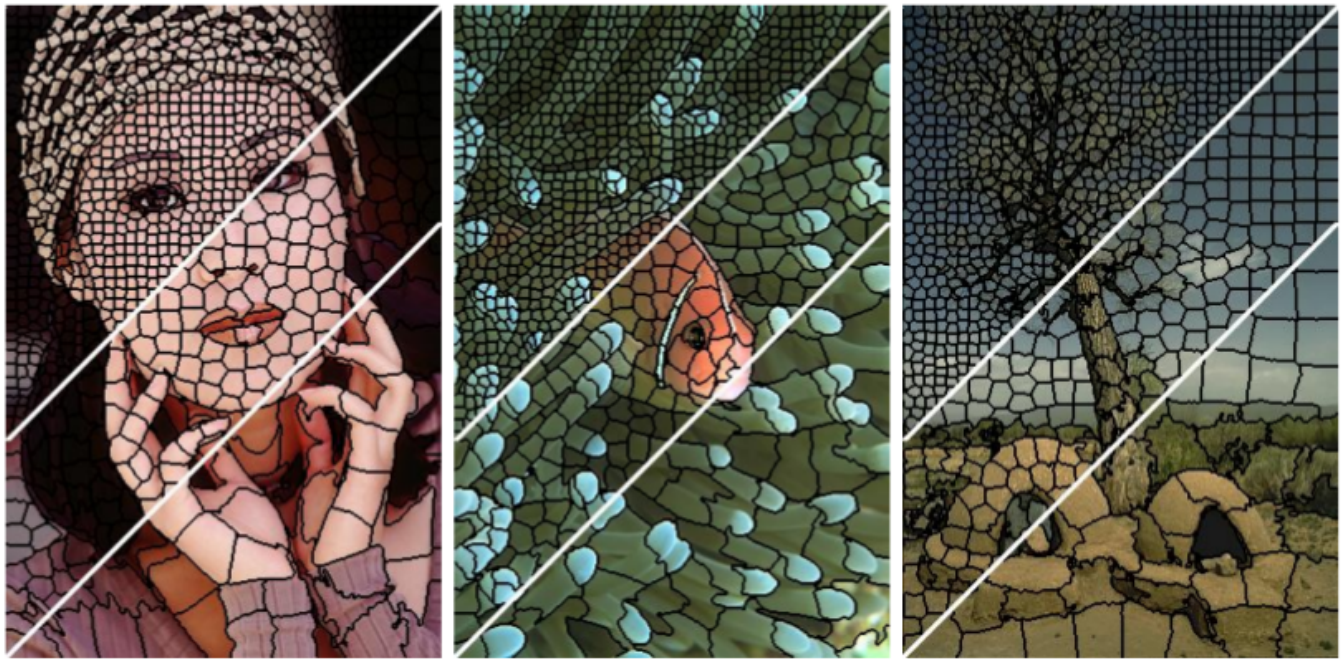
Superpixels

- Superpixel algorithms group pixels into perceptually meaningful atomic regions which can be used to replace the rigid structure of a pixel grid
- They capture image redundancy, provide a convenient primitive from which to compute image features, and greatly reduce the complexity of subsequent image processing tasks

Superpixels

- Superpixel algorithms group pixels into perceptually meaningful atomic regions which can be used to replace the rigid structure of a pixel grid
- They capture image redundancy, provide a convenient primitive from which to compute image features, and greatly reduce the complexity of subsequent image processing tasks
- Superpixels are useful for applications such as depth estimation, image segmentation, skeletonization, body model estimation, and object localization

Superpixels



- Images segmented using into superpixels of size 64, 256, and 1024 pixels

Superpixel Properties

- The following properties are generally desirable for generating superpixels:

Superpixel Properties

- The following properties are generally desirable for generating superpixels:
  1. Superpixels should adhere well to image boundaries

Superpixel Properties

- The following properties are generally desirable for generating superpixels:
  1. Superpixels should adhere well to image boundaries
  2. When used to reduce computational complexity as a preprocessing step, superpixels should be fast to compute, memory efficient, and simple to use

Superpixel Properties

- The following properties are generally desirable for generating superpixels:
  1. Superpixels should adhere well to image boundaries
  2. When used to reduce computational complexity as a preprocessing step, superpixels should be fast to compute, memory efficient, and simple to use
  3. When used for segmentation purposes, superpixels should both increase the speed and improve the quality of the results

Simple Linear Iterative Clustering (SLIC)

- The **simple linear iterative clustering (SLIC)** algorithm generates superpixels by clustering pixels based on their color similarity and proximity in the image plane

Simple Linear Iterative Clustering (SLIC)

- The **simple linear iterative clustering (SLIC)** algorithm generates superpixels by clustering pixels based on their color similarity and proximity in the image plane
- This is done in five-dimensional [*labxy*] space, where [*lab*] is the pixel color in the CIELAB color space (known for its perceptually uniform color distances) and *xy* is the pixel position

Simple Linear Iterative Clustering (SLIC)

- The maximum possible distance between two colors in CIELAB space is limited, and the spatial distance in the xy plane depends on the image size

Simple Linear Iterative Clustering (SLIC)

- The maximum possible distance between two colors in CIELAB space is limited, and the spatial distance in the xy plane depends on the image size
- Therefore, it is not possible to simply use the Euclidean distance in this 5D space without normalizing the spatial distances

Simple Linear Iterative Clustering (SLIC)

- SLIC utilizes a distance measure that considers superpixel size

Simple Linear Iterative Clustering (SLIC)

- SLIC utilizes a distance measure that considers superpixel size
- It enforces color similarity as well as pixel proximity in 5D space such that the expected cluster sizes and their spatial extent are approximately equal

SLIC Distance Measure

- SLIC takes as input a desired number of approximately equally-sized superpixels K

SLIC Distance Measure

- SLIC takes as input a desired number of approximately equally-sized superpixels K
- For an image with N pixels, the approximate size of each superpixel is N/K pixels

SLIC Distance Measure

- SLIC takes as input a desired number of approximately equally-sized superpixels K
- For an image with N pixels, the approximate size of each superpixel is N/K pixels
- Given roughly equally sized superpixels, there is a superpixel center at every grid interval $S = N/K$

SLIC Distance Measure

- Initially K superpixel cluster centers are chosen, $C_k = [l_k, a_k, b_k, x_k, y_k]^T$, with $k = [1, K]$ at regular grid intervals S

SLIC Distance Measure

- Initially K superpixel cluster centers are chosen, $C_k = [l_k, a_k, b_k, x_k, y_k]^T$, with $k = [1, K]$ at regular grid intervals S
- Since the spatial extent of any superpixel is S^2 (approximate area of a superpixel), we can safely assume pixels associated with this cluster center lie within a $2S \times 2S$ area around the superpixel center on the xy plane

SLIC Distance Measure

- Initially K superpixel cluster centers are chosen, $C_k = [l_k, a_k, b_k, x_k, y_k]^T$, with $k = [1, K]$ at regular grid intervals S
- Since the spatial extent of any superpixel is S^2 (approximate area of a superpixel), we can safely assume pixels associated with this cluster center lie within a $2S \times 2S$ area around the superpixel center on the xy plane
- This becomes the search area for the pixels nearest to each cluster center

SLIC Distance Measure

- Euclidean distances in CIELAB color space are perceptually meaningful for small distances

SLIC Distance Measure

- Euclidean distances in CIELAB color space are perceptually meaningful for small distances
- If spatial pixel distances exceed this perceptual color distance limit, then they begin to outweigh pixel color similarities

SLIC Distance Measure

- Euclidean distances in CIELAB color space are perceptually meaningful for small distances
- If spatial pixel distances exceed this perceptual color distance limit, then they begin to outweigh pixel color similarities
- This results in superpixels that do not respect region boundaries, only proximity, in the image plane

SLIC Distance Measure

- Thus, instead of using the Euclidean norm in 5D space, a distance measure D_s defined as follows

$$d_{lab} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2}$$

$$d_{xy} = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2}$$

$$D_s = d_{lab} + \frac{m}{S} d_{xy}$$

where D_s is the sum of the *lab* and *xy* plane distance normalized by the grid interval S

SLIC Distance Measure

- Thus, instead of using the Euclidean norm in 5D space, a distance measure D_s defined as follows

$$d_{lab} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2}$$

$$d_{xy} = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2}$$

$$D_s = d_{lab} + \frac{m}{S} d_{xy}$$

  where D_s is the sum of the *lab* and *xy* plane distance normalized by the grid interval S

- The variable $m \in [1, 20]$ controls the compactness of a superpixel, e.g. the greater the value of m, the more spatial proximity is emphasized and the more compact the cluster

SLIC Algorithm

- The algorithm begins by sampling K regularly spaced cluster centers and moving them to seed locations corresponding to the lowest gradient position in a 3×3 neighborhood

SLIC Algorithm

- The algorithm begins by sampling K regularly spaced cluster centers and moving them to seed locations corresponding to the lowest gradient position in a 3×3 neighborhood
- This is done to avoid placing them at an edge and to reduce the chances of choosing a noisy pixel

SLIC Algorithm

- The image gradients are computed as

$$G(x,y) = ||\mathbf{I}(x+1,y)-\mathbf{I}(x-1,y)||^2+||\mathbf{I}(x,y+1)-\mathbf{I}(x,y-1)||^2$$

where $\mathbf{I}(x,y)$ is the *lab* vector corresponding to the pixel at position (x,y) and $||\cdot||$ is the L_2 norm

SLIC Algorithm

- The image gradients are computed as

$$G(x,y) = ||\mathbf{I}(x+1,y)-\mathbf{I}(x-1,y)||^2+||\mathbf{I}(x,y+1)-\mathbf{I}(x,y-1)||^2$$

  where $\mathbf{I}(x,y)$ is the *lab* vector corresponding to the pixel at position (x,y) and $||\cdot||$ is the L_2 norm

- This takes into account both color and intensity information

SLIC Algorithm

- Each pixel in the image is associated with the nearest cluster center whose search area overlaps this pixel

SLIC Algorithm

- Each pixel in the image is associated with the nearest cluster center whose search area overlaps this pixel
- After all the pixels are associated with the nearest cluster center, a new center is computed as the average *labxy* vector of all the pixels belonging to the cluster

SLIC Algorithm

- Each pixel in the image is associated with the nearest cluster center whose search area overlaps this pixel
- After all the pixels are associated with the nearest cluster center, a new center is computed as the average *labxy* vector of all the pixels belonging to the cluster
- The process of associating pixels with the nearest cluster center, and recomputing the cluster center, is iteratively repeated until convergence

SLIC Algorithm

- At the end of this process, there can be pixels in the vicinity of a larger segment that may have the same label but are not connected to it

SLIC Algorithm

- At the end of this process, there can be pixels in the vicinity of a larger segment that may have the same label but are not connected to it
- Although this situation is rare, it may arise despite the spatial proximity measure since clustering does not explicitly enforce connectivity

SLIC Algorithm

- At the end of this process, there can be pixels in the vicinity of a larger segment that may have the same label but are not connected to it
- Although this situation is rare, it may arise despite the spatial proximity measure since clustering does not explicitly enforce connectivity
- Therefore, connectivity is enforced in the last step of the algorithm by relabeling disjoint segments with the labels of the largest neighboring cluster

SLIC Algorithm

Algorithm 1 SLIC superpixel segmentation

/ Initialization */*

Initialize cluster centers $C_k = [l_k, a_k, b_k, x_k, y_k]^T$ by sampling pixels at regular grid steps S.

Move cluster centers to the lowest gradient position in a 3×3 neighborhood.

Set label $l(i) = -1$ for each pixel i.

Set distance $d(i) = \infty$ for each pixel i.

repeat

  / Assignment */*

  for each cluster center C_k **do**

    for each pixel i in a $2S \times 2S$ region around C_k **do**

      Compute the distance D between C_k and i.

      if $D < d(i)$ **then**

        set $d(i) = D$

        set $l(i) = k$

      end if

    end for

  end for

  / Update */*

  Compute new cluster centers.

  Compute residual error E.

until $E \leq$ threshold

SLIC Complexity

- The idea of iteratively evolving local clusters and cluster centers is a special case of k-means adapted to the task of generating superpixels

SLIC Complexity

- The idea of iteratively evolving local clusters and cluster centers is a special case of k-means adapted to the task of generating superpixels
- SLIC's distance measure is able to localize pixel search to an area $(2S \times 2S)$ on the image plane that is inversely proportional to the number of superpixels K

SLIC Complexity

- The idea of iteratively evolving local clusters and cluster centers is a special case of k-means adapted to the task of generating superpixels
- SLIC's distance measure is able to localize pixel search to an area $(2S \times 2S)$ on the image plane that is inversely proportional to the number of superpixels K
- In practice, a pixel falls in the local neighborhood of no more than 8 cluster centers and experiments show that it suffices to run the algorithm for 4 to 10 iterations

SLIC Complexity

- The time complexity for k-means is $O(NKI)$ where N is the number of data points (pixels in the image), K is the number of clusters (or seeds), and I is the number of iterations required for convergence

SLIC Complexity

- The time complexity for k-means is $O(NKI)$ where N is the number of data points (pixels in the image), K is the number of clusters (or seeds), and I is the number of iterations required for convergence
- SLIC achieves $O(N)$ complexity since it needs to compute distances from any point to no more than 8 cluster centers and the number of iterations is constant

SLIC Complexity

- The time complexity for k-means is $O(NKI)$ where N is the number of data points (pixels in the image), K is the number of clusters (or seeds), and I is the number of iterations required for convergence
- SLIC achieves $O(N)$ complexity since it needs to compute distances from any point to no more than 8 cluster centers and the number of iterations is constant
- Thus, SLIC is specific to the problem of superpixel segmentation and unlike k-means avoids several redundant distance calculations

Summary

- Image segmentation is closely related to clustering techniques

Summary

- Image segmentation is closely related to clustering techniques
- Since the number of segmentation methods is vast, a good way to get a handle on some of the better algorithms is to look at experimental comparisons on human-labeled datasets

Summary

- Image segmentation is closely related to clustering techniques
- Since the number of segmentation methods is vast, a good way to get a handle on some of the better algorithms is to look at experimental comparisons on human-labeled datasets
- The best known of these datasets is the Berkeley Segmentation Dataset and Benchmark, which consists of 1000 images from a Corel image dataset that were hand-labeled by 30 human subjects