INTERNATIONAL INSTITUTE OF
INFORMATION TECHNOLOGY

HYDERABAD

CSE578:
Computer Vision

Professor : Anoop Namboodiri
Notes By : Sai Manaswini Reddy I

2021

IMAGE GROUPING

Segmentation as Labelling :

Process of grouping pixels into meaningful regions. We group similar pixels together, similarity is semantic similarity. Appearance similarity and semantic similarity are different.

Segmentation :

Dividing an image into semantically meaningful regions.


Looking at the first image we can not group them, but once if we notice this we keep on noticing. This is because the background has similar properties as that of the dog, so we are not able to segment initially. Later when we know that there is a dog present in the picture, our brain tries to group or find a border kind of thing that suits the dog.

So, we basically classify pixels as :

- Background : other than object of interest
- Foreground : object of interest

Types of Segmentation :

- Classification-based :
  - Label pixels based on region properties
  - Label each pixel based on object models/background model
- Region-based :
  - Region growing and splitting
  - Start with a small region and keep growing or start with a large and keep splitting until we reach the right size.
- Boundary-based :
  - Find edges in the image and use them as region boundary
- Motion-based
  - Group pixels that have consistent motion (e.g., move in the same direction)
  - pixels that move together are grouped together.

Thresholding :

Decide each pixel to be a part of an object or background depending on its gray value.

$$t(m,n) = \begin{cases} 1 & u(m,n) > T \\ 0 & u(m,n) \leq T \end{cases}$$

We decide a threshold, and if the value of intensity is greater then it belongs to the object otherwise it is a part of background.


For the example on the right threshold is 95. This is a manual way to find the threshold, we will look into automation of this.

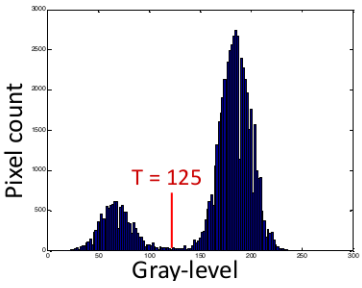
Types of thresholding :

- Global :
  - A single threshold for the whole image.
  - We find out the best threshold.
- Adaptive(local) :
  - Decide the threshold for every pixel depending on its neighbourhood.
  - We define a function for threshold.

Global thresholding :

A count of pixels of each gray level (or a range of gray levels) in an image (histogram) are computed.

Original



Histogram

Thresholded(T=125)

More number of Darker pixels belong to objects and lighter ones to background. Intermediate are low, so the threshold is in the valley between both the peaks.

Automatic thresholding :

1. Select an initial estimate of T.
2. Segment the image using T. Compute the mean gray values of the two regions, μ_1 and μ_2.
3. Set the new threshold T=(μ_1 + μ_2)/2.
4. Repeat 2 and 3 until T stabilizes.

Assumptions: normal distribution, low noise

This is something like k-means.

Extensions :

- Multiple Thresholds
  - Find multiple peaks and valleys in the gray level histogram
- Multi-spectral Thresholding
  - In color images, one could use different thresholds for each of the color channels

One might set all the background pixels to black, while leave the foreground at the original value so that the information is not lost.

Otsu's Method :

1. Compute histogram and probabilities of each intensity
2. Set up initial $\omega_i(0)$ and $\mu_i(0)$ and
3. Step through all possible thresholds t = 1..t max

    a. Update ω_i and μ_i
    b. Compute $\sigma_b^2(t)$ (this is inter-class variance)
4. Final threshold corresponds to the maximum $\sigma_b^2(t)$
5. If maximum is true for a region of contiguous pixels :
    a. Compute two maxima (and respective thresholds t_1 and t_2) using > and >= (first and last maxima)
    b. Desired threshold = $(t_1 + t_2)/2$

Adaptive thresholding :

Adaptive thresholding changes the threshold dynamically over the image. This can accommodate strong illumination gradients and shadows.

Here the threshold is the mean of pixels (gray values) in a neighborhood$_{(\text{say } 7\times7)}$. Darker regions will have a low threshold and lighter regions will comparatively have a higher threshold.

In regions with pure background (half the pixels are avg and rest are below), so we will see foreground and background appearing everywhere.

Thresholding using Mean-C

- Set cxc image regions of uniform gray level to background.
- If there is sufficient variation then we do thresholding.

Chow and Kaneko :

1. Apply the mean operator (low pass filter)
    a. Blur the whole image.
2. Subtract original image from the “mean”image
3. Threshold image in step 2
4. Invert the result

Optimal Thresholding :

The gray level histogram is approximated using a mixture of two gaussian distributions (bimodal fit) and set the threshold as their point of intersection to minimize the segmentation error.

Gaussian Mixture Estimation by EM

$$\text{Obj} = \text{N}(\mu_1, \sigma_1) = \frac{1}{\sigma_1\sqrt{2\pi}}\exp(\frac{(\text{x}-\mu_1)^2}{2\sigma_1^2})$$

$$\text{Bkg} = \text{N}(\mu_2, \sigma_2) = \frac{1}{\sigma_2\sqrt{2\pi}}\exp(\frac{(\text{x}-\mu_2)^2}{2\sigma_2^2})$$

- E-Step: Computed the expected pixel label assignments.
- M-Step: Computed Maximum-Likelihood estimates of the parameters: μ_1 , σ_1 , μ_2 , σ_2

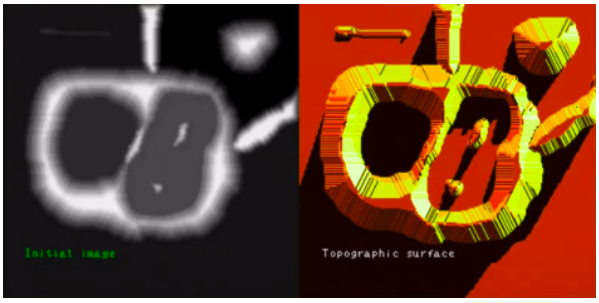
Segmentation as Optimal Labeling :

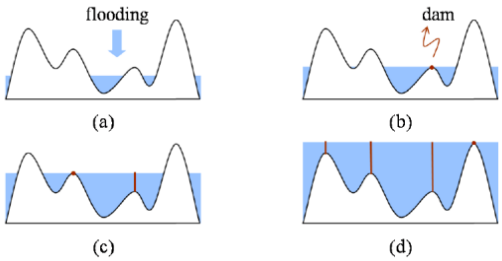
- Model knowledge about the world.
    - Here we modelled it as simple gaussians.
- Classify each pixel as belonging to a specific object.
    - Independent classification does not work (we see a lot of noise)
    - Need to incorporate neighborhood information.
- Consider a graph over the image
    - Each node in the graph needs to be labeled.
    - Edges in the graph represent neighborhood constraints.
- Define a cost function, Q(f), using the above.

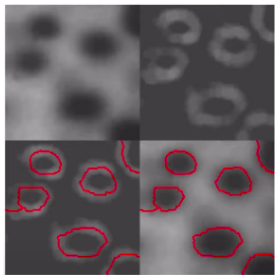
- Compute the optimal labeling wrt Q(f).

Watershed Segmentation :

Segments an image into several “catchment basins” or “regions”. Any gray scale image can be interpreted as a 3D topographical surface. Image can be segmented into regions where rainwater would flow into the same lake.

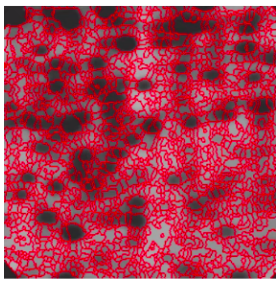






- Watershed algorithm segments image into regions by concept of rainwater flowing into the same lake. Groups where the water is stagnant forms the segments.
- Flood the landscape from local minima(low intense pixel) and prevent merging of water from different minima results in partitioning the image into “catchment basins” or “watershed lines”.
- Generally applied on image gradients rather than on the image.

Limitations :



- In practice, watershed segmentation does over-segmentation due to noise and irregularities in the image.
- Hence usually used as a part of an interactive system, where the user marks “centres” of each component, on which the flooding is done without worrying about the rest. (we also use celeski’s method to overcome this)

Region based segmentation :

Segmentation - Basic formulation :

$p(R_i)$ is a local predicate property defined over the points in set R_i.

1. $\bigcup_{i=1}^{n} R_i = R$
2. R_i is a connected region, i=1,2,3,...,n.
3. $R_i \cap R_j = \phi$, $\forall$ i≠j
4. $p(R_i)$ = TRUE , $\forall$ i = 1,2,3,..,n
5. $p(R_i \cup R_j)$ = FALSE , $\forall$ i≠j

Basic Region Growing :

If we threshold an entire image we get a bunch of disconnected regions, above the threshold. We use morphological image processing to select the regions that we want. Select a group of seed pixels within an image. Select a set of similarity criterion such as grey level intensity or color and set up a stopping rule. Grow regions by attaching each seed those have

predefined properties similar to seed pixels. Stop region growing when no more pixels met the criterion for inclusion in that region.

If we want “common pixels” (connected region) near one point (similar intensity to that point) :

Algorithm :

1. From INPUT image I(x,y) get a binary “SEED IMAGE” S(x,y) for locations that we are interested in (by thresholding). (shrink all these regions down to a single point - seed)
2. Reduce seed connected components to a single point. (erosion/region props)
3. Let T(x,y)=1 (processed image) if I(x,y) satisfies some predicate/condition else 0.
    a. Example : (x,y) is set of 8 points connected to seed point (x_i,y_i) and $|I(x,y)-(x_i,y_i)| \leq T$.

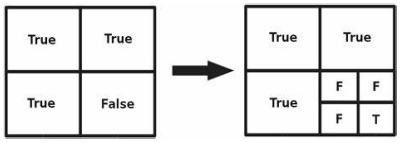
Region Split and Merge :

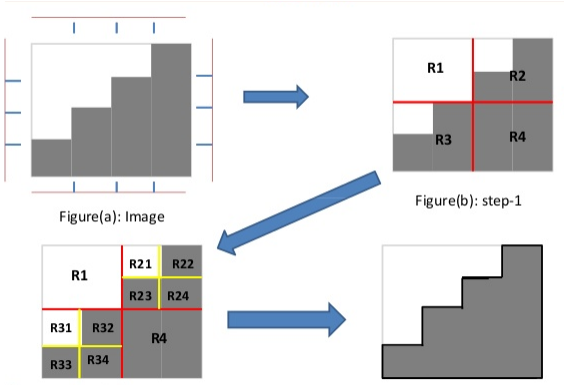
Specify a condition/ rule. In the previous case, region growing started from a set of seed points. Here we start with the whole image as a single region and subdivide the region that does not satisfy a condition of homogeneity.

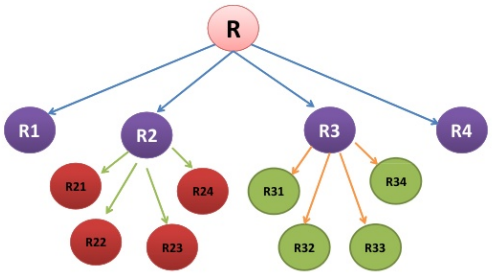
Start with small regions (2×2 or 4×4) and merge the regions that have similar characteristics (eg : gray level variance)

Algorithm :

1. Split any region if $p(R_i)$ = False
2. Merge any adjacent region R_i and R_i if $p(R_i \cup R_j)$ is True.
3. Stop when there is no further merge/split possible.







Clustering and Superpixels :

Clustering is an unsupervised learning task. Clustering is defined as a process of grouping objects based on attributes, so that objects with similar attributes lie in the same cluster.

A superpixel can be defined as a group of pixels that share common characteristics (like pixel intensity).

- They carry more information than pixels.
- Superpixels have a perceptual meaning since pixels belonging to a given superpixel share similar visual properties.
- They provide a convenient and compact representation of images that can be very useful for computationally demanding problems.

K - means Clustering (Hard Clustering) :

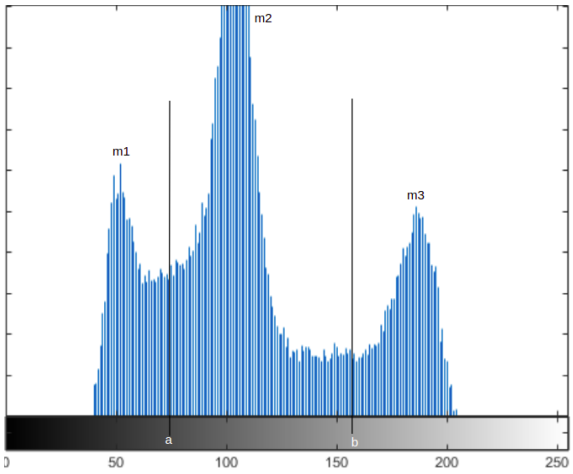
Hard clustering assumes that a pixel can only belong to a single cluster. There exist sharp boundaries between clusters. K-means clustering group n pixels of an image into k number of clusters, where k<n and k is a positive integer. The centroids of the predefined clusters are computed randomly. Clusters are formed on the basis of some similarity features like distance of pixel intensities and gray level intensity of pixels.

Algorithm :

1. Choose the number of clusters you want to find which is k.
2. Select at random K points, the centroids (m_1 , m_2 ,...., m_k) (not necessarily from the data set)
3. Assign each data point $x_i \in \mathbb{R}^n$ to the closest cluster ($x_i \in j$ if $||x_i - m_j|| < ||x_i - m_k|| \ \forall \ k \neq j$)
4. Update the means m_j for each cluster (average value of all x in j)
5. If there is a reassignment then keep repeating steps 3 and 4 until m_j stops changing.

Bad initialization leads to bad clusters, so we do different random initializations and choose the tightest clusters (tightness is measured using standard deviation of the points in the cluster).

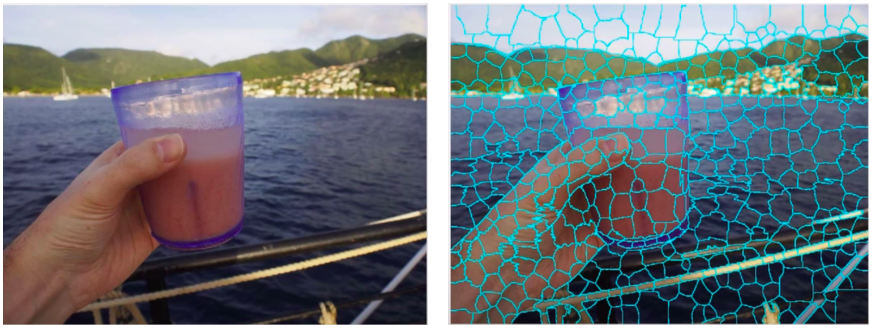
Example (Histogram) :



For k = 3 ; with peaks m_1, m_2 and m_3 as initial means, k-means might give all the pixels between 0 and a to cluster 1; between a and b to 2; b and 256 to 3.

Modification of K-means in image processing : Super pixels

Regions that are contiguous and have similar intensity/colour. Divide an image into 500 clusters.



All the clusters more or less have a similar colour. All the pixels in one cluster are superpixels.

Advantages :

- More compact (Thousands of superpixels could represent millions of mixels)
- Keeps things together better for subsequent segmentation ; computational efficiency.
- Graph cut algorithms benefit directly from using superpixels instead of pixels.

SLIC superpixels :

We also need to add spatial information into this to cluster. This can be done using SLIC(simple linear iterative clustering) superpixels. Clustering in 5D space [R G B x y] = [colour location].

algorithm :

1. Initialize superpixels centres by sampling N locations on a regular GRID in the image plane.
2. Move slightly within the 3 × 3 neighborhood to lie on the lowest gradient position (seeds should lie on the flattest locations possible, don't want to start with an edge).
3. For each cluster center m_i compute the distance between m_i and each pixel in the neighborhood of m_i. (s × s - this prevents super pixels from getting too large).
4. Assign pixels to cluster 'i' if it was better than its current value.
5. Update the cluster centres like in K-Means
6. Repeat until convergence.

For extra effects : replace each pixel intensity in each cluster with avg intensity of that cluster.

Distance function :

$$d_c = \left\| \begin{bmatrix} R \\ G \\ B \end{bmatrix}_i - \begin{bmatrix} R \\ G \\ B \end{bmatrix}_j \right\|_2 \quad \text{color}$$

$$d_s = \left\| \begin{bmatrix} x \\ y \end{bmatrix}_i - \begin{bmatrix} x \\ y \end{bmatrix}_j \right\|_2 \quad \text{spatial}$$

We can not directly add these things because they have different scales.

$$D = \sqrt{\left(\frac{dc}{c}\right)^2 + \left(\frac{ds}{s}\right)^2}$$

c = maximum colour distance

s = maximum spatial distance

We use c to tune tradeoff :

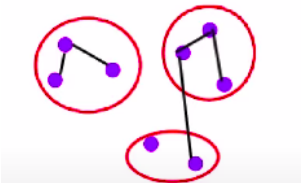
1. If c is big then colour doesn't play much role ; superpixels are more compact ;
2. Else superpixels are more tightly stuck to image boundaries.

Graph based Segmentation :

Graph based segmentation algorithm uses relative dissimilarities between regions to decide which ones to merge (region merging method)

An image = graph => G = (V,E) where adjacent pixels form vertices V and edges E.

A pixel-pixel dissimilarity metric w(e) is defined where edge e = (v_1,v_2) and v_1, v_2 are two pixels. This measures, for instance, the intensity difference between N_8 neighbors.



Ref : https://www.cse.iitb.ac.in/~meghshyam/seminar/SeminarReport.pdf (so good, go through it during leisure).

Minimum Spanning tree based segmentation :

- For a region C, its internal difference is defined as the largest edge weight in the region's minimum spanning tree:

$$\text{Int}(C) = \max_{e \in MST(C)} w(e)$$

- The minimum internal difference between two adjacent regions is defined as ($\tau(C)$ is a manually chosen region penalty) :

$$\text{Mint}(C_1, C_2) = \min(\text{Int}(C_1) + \tau(C_1), \text{Int}(C_2) + \tau(C_2))$$

- The threshold function τ controls the degree to which the difference between two components must be greater than their internal differences in order for there to be evidence of a boundary between them. Felzenszwalb et. al. used a threshold function based on the size of the component.

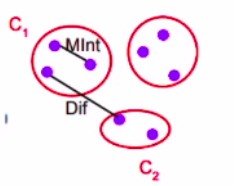
$$\tau(C) = k/|C|$$

where |C| denotes the size of C, and k is some constant parameter. So, if component size is small, it will require larger evidence for a boundary

- For any two adjacent regions with at least one edge connecting their vertices, difference between these two regions = minimum weight edge connecting these two regions.

$$Dif(C_1, C_2) = \min w(e)$$

$$e = (v_1, v_2) \mid v_1 \in C_1, v_2 \in C_2$$



- A predicate $D(C_1, C_2)$ for any two regions C_1, C_2 is defined as :

$$D(C_1,C_2) = \begin{cases} \text{true} & \text{if } Dif(C_1,C_2) > MInt(C_1,C_2) \\ \text{false} & \text{otherwise} \end{cases}$$

For any two regions, if the predicate D evaluates to **false**, regions are merged. Else, regions are considered separate.

This algorithm merges any two regions whose difference is smaller than the minimum internal difference of these two regions.

Algorithm :

Input :- Graph G(V, E) : |V | = n, |E| = m

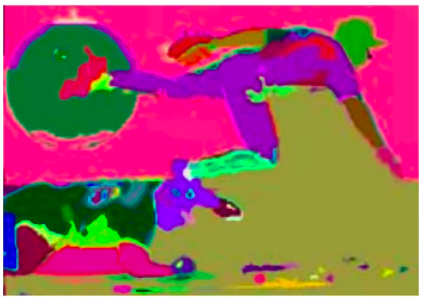
Output :- Set of Components $S = \{C_1, C_2, \ldots C_r\}$

1. Sort set of edges, E into $\pi = \{o_1, o_2, \ldots o_m\}$, by non-decreasing edge weight.
2. Let initial segmentation $S\,0 = \{\{v_i\}\} \ \forall \ i = 1 \ldots n$ i.e. each vertex is in its component.
3. Repeat step 4 for q = 1, ..., m.
4. Form next level segmentation S q from earlier segmentation S q−1 as follows
   - Let vi and vj be end vertices of edge oq i.e. oq = (vi , vj)
   - Let C_i^{q-1} = the component in S^{q-1} containing v_i
   - Similarly, C_j^{q-1} = the component in S^{q-1} containing v_j
   - If C_i^{q-1} , C_j^{q-1} and $w(oq) \leq MInt(C_i^{q-1}, C_j^{q-1})$ then :

     Merge C_i^{q-1} , C_j^{q-1} in S^q

     Otherwise $S^q = S^{q-1}$
   - Return $S = S^m$

In Grid graphs weights of edges can be given by difference of intensities between two pixels, while nearest neighbor graphs weights of edges are given by euclidean distance between two pixels represented in feature space such as (x, y, r, g, b). Currently, Graph cuts based methods have emerged as a preferred way to solve image segmentation problems.



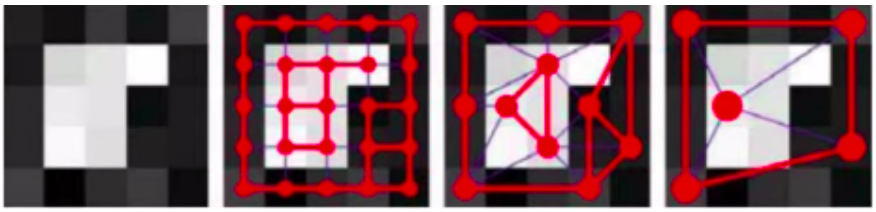
Probabilistic bottom up Merging Algorithm :

Image segmentation based on aggregating two cues - *gray level similarity* and *texture similarity*

- Initially consider each pixel as a region, assign merging likelihood P_{ij} based on intensity and texture similarities to each pair of neighboring regions.
- Given a graph $G^{[s-1]} = (V^{[s-1]}, E^{[s-1]})$, $G^{[s]}$ is constructed by selecting subset of seed nodes $C \subset V^{[s-1]}$, we merge the nodes/regions if they are strongly coupled to region C.
- Strong coupling :

$$\frac{\sum_{j \in C} P_{ij}}{\sum_{j \in V} P_{ij}} > \text{threshold (usally set to 0.2)}$$

- Once a segmentation is identified at a coarser level, assignments are propagated to their finer level “children”, followed by further coarsening.

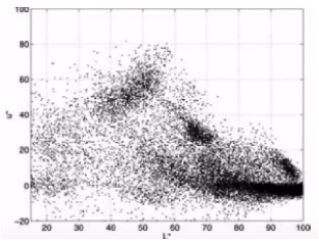


Mean Shift Segmentation :

A mode finding technique based on non-parametric density estimation

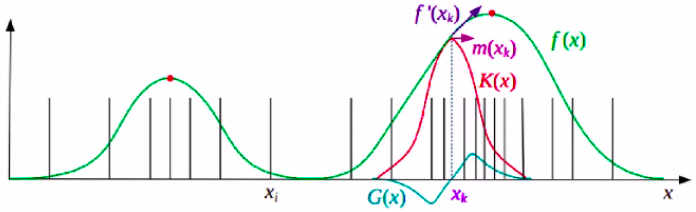
- Feature vectors of each pixel in the image are assumed to be samples from an unknown probability distribution.
- We estimate p.d.f using non-parametric estimation and find its nodes.

- Image is segmented pixel-wise by considering every set of pixels which climb to the same mode as a consistent segment.
- Consider the example image below on the left. The graph on the right shows the distribution of L*u* features of each pixel (in the L*u*v /CIELUV space)



- Our aim is to obtain modes of distribution on the right, without actually explicitly computing the density function!

- 1D visualisation to find the mode finding approach. We can extrapolate this 1D approach to 2D.



- Best and popular way to estimate PDF is through kernel density. Estimate the density function by convolving the data (observations) with the kernel of width h, where k is the kernel function.

$$f(x) = \sum_i k(\frac{||x - x_i||}{h^2})$$

- To find the modes (peaks), mean shifts uses a gradient ascent method(we want peaks) with multiple restarts.
- First, we pick a guess y_0 for a local maximum which can be a random input data x_i.
- Then, we calculate the gradient density estimate f(x) at y_0 and take an ascent step in that direction.

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

  Where g(.) = -k'(.), derivative of kernel k
- The gradient of the density function can be re-written as :

$$\nabla f(x) = \left[\sum_i G(x_i - x)\right] m(x) \quad \text{where } m(x) = \frac{\sum_i x_i G(x_i - x)}{\sum_i G(x_i - x)} - x$$

$$\text{where } m(x) = \frac{\sum_i x_i G(x_i - x)}{\sum_i G(x_i - x)}$$

$$G(x_i - x) = g\left(\frac{||x_i - x||}{h^2}\right)$$

- How did we get this ? solve
- The vector m(x) is known as the mean shift, the difference between x and the gradient weighted mean of the neighbors around x $G(x-x_i)$.
- Current estimate of the mode y_k at iteration k is replaced by its locally weighted mean :

$$y_{k+1} = y_k + m(y_k) = \frac{\sum_i x_i G(y_k - x_i)}{\sum_i G(y_k - x_i)}$$

- Keep doing this until we reach the mode of the distribution.
- Find out all the neighboring points that by climbing would lead to this mode will form one segment, some other points that form some other mode lead to other segments.
- This method relies on selecting a suitable kernel width.

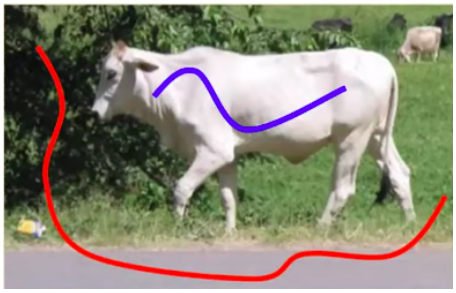
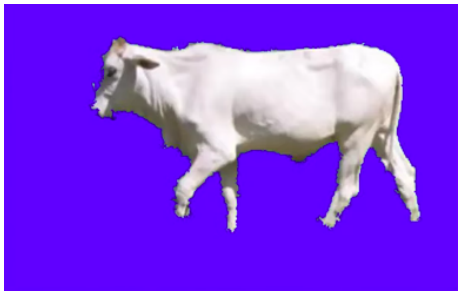
Above description is strictly color based, however, better results can be obtained by working with feature vectors which include both color and location.

Graph Cut for image segmentation :

Region splitting method where a graph representing pixels in an image is successively split into parts.

We define an energy function and minimize its cost. Energy function is defined in terms of energy or the cost of assigning a label to a node. Cost function essentially captures our understanding of the world (models our knowledge about natural images) .

The algorithm should have an idea about what a cow is to separate a cow. In this we capture the understanding of the objects by the cost function. This information can be obtained by taking the user input. From these example pixels.

Cost function :

Quality of segmentation depends on the cost function. Optimization of cost function depends on Cost function. Denoted by Q(f).

Binary Image Segmentation :

We can create a segmented image by computing the colour histogram or colour density function of foreground and background and by computing how foreground like or how background like the pixel is we assign.

We define a four connectivity graph G = (V,E) for all the pixels

Assign a label to each vertex from L = {obj,bkg} (object and background).

Cost of labelling is f : V → L (every vertex has to be mapped)

Here, what we want is :

Consider a pixel in the body of the cow, by looking at the colour values there : cost of label 'obj' should be low and 'bkg' should be high and the other way round for the pixel in green (initially marked as background).

Terms in the cost function :

Unary term :

First term in the cost function is the unary term. Independently we assign a cost for assigning every node to either label a or label b and this is independent only for single pixel (does not depend on neighborhood)

Pairwise term :

We will take 2 vertices at a time and consider :

- Cost of assigning same label to both
- Cost of assigning different label to both

Nearby pixels usually tend to have similar labels (not at the boundary). We give cost to the pixels such that if two pixels are nearby, if their appearances are different then you should have a different label (high) else low.

Conclusion :

Every pixel will have independent cost depending on its appearance and a pairwise cost depending on the similarity of appearance.

Cost function is Q(f) *and* h *number of segments implies* L = { 1,2,3,...,h } . We wanna minimize the energy function on the graph i.e., minimum cost labelling i.e., if f is a labelling then we want the f for which Q(f) is minimized. Let it be f*.

$$f^* = \arg\min Q(f)$$

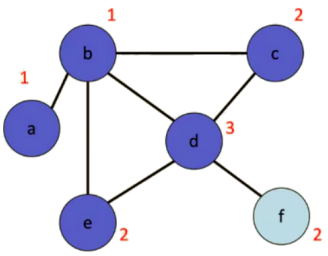
Energy function Formulation :

Let us consider 4 nearby vertices :

Since, this is binary segmentation,

$$L = \{ l_0 , l_1 \}$$

Random variables $V = \{ V_a , V_b , \dots \}$



Data points (pixels) $D = \{ D_a , D_b , ... \}$

Labelling $f : \{ a,b,.. \} \rightarrow \{ 0,1,.. \}$

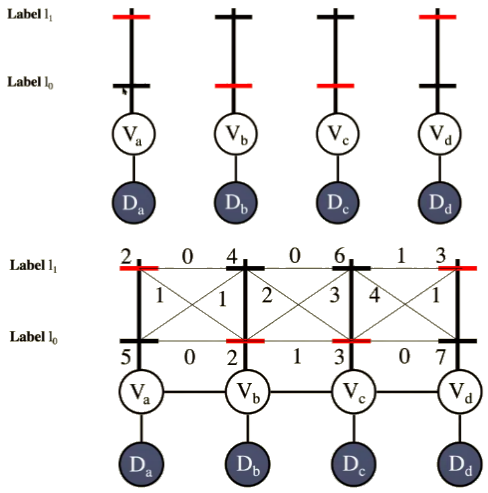
$Q(f)$ = unary term + pairwise term

$= \Sigma_a \theta_{a;f(a)}$ + pairwise term

f(a) is labelling, θ is the cost of assigning.

$E : (a,b) \in E$ if V_a and V_b are neighbors

$$Q(f ; \theta) = \Sigma_a \theta_{a;f(a)} + \Sigma_{(a,b)} \theta_{ab;f(a)f(b)}$$



θ_{ab} depends on similarity of a and b.

Map estimation :

$$q^* = \min Q(f ; \theta) = Q(f^* ; \theta)$$

$$f^* = \arg\min Q(f ; \theta)$$

f(a)	f(b)	f(c)	f(d)	Q(f; θ)
0	0	0	0	18
0	0	0	1	15
0	0	1	0	27
0	0	1	1	20
0	1	0	0	22
0	1	0	1	19
0	1	1	0	27
0	1	1	1	20

f(a)	f(b)	f(c)	f(d)	Q(f; θ)
1	0	0	0	16
1	0	0	1	13
1	0	1	0	25
1	0	1	1	18
1	1	0	0	18
1	1	0	1	15
1	1	1	0	23
1	1	1	1	16

$$f^* = (1, 0, 0, 1 \}$$

$$q^* = 13$$

Computational complexity :

Total possible labellings = $2^{|v|}$

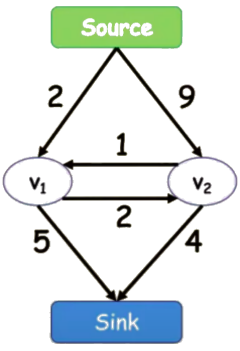
$|v|$ = number of pixels

Brute force solution is way too difficult to compute.

MAP Estimation in general is NP-hard !!

There are a few algos like st - mincut for a particular class of problems.

st - Mincut :



There is a “source” and a “sink/target” (foreground and background terminal). Each pixel in the image (graph) is connected to both the foreground and background terminal.

Unary term is weight of the edges between pixel and source/sink

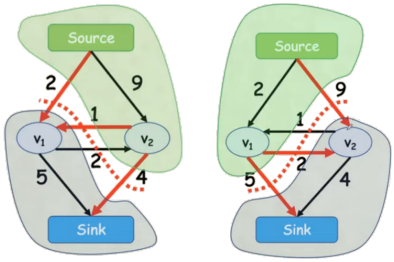
Graph (V,E,C)

Vertices $V = \{ v_1, v_2, ..., v_n \}$

Edges $E = \{ (v_1, v_2), ... \}$

St-cut cuts some of these edges where the image gets divided into 2 parts such that one chunk belongs to background and other belongs to foreground. Anything that separates foreground and background is an st-cut.

An st-cut (S,T) divides the nodes between source and sink.



Sum of cost of all the edges going from S to T.

A cut is a set of edges when removed separates F and B terminals (we get 2 disjoint sets A and B) . We assign a weight to each edge (pixel-pixel and pixel-terminal) based on intensities.

st-Mincut is the st-cut with minimum cost.

From the Mincut-Maxflow theorem we know that Mincut has the dual problem Maxflow.

Maxflow :

Let's consider the situation where the water flows from source to sink and edge weights represent the capacity of the pipes.

Maximum amount of water that can flow from source to sink is maxflow.

Constraints :

Edges : flow < capacity

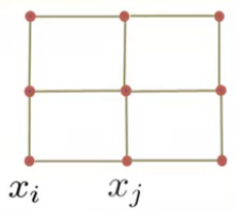
Nodes : Flow in = Flow out

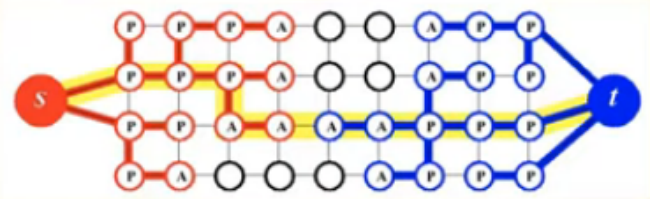
Augmenting path based algorithms :

1. Find the path from source to sink with positive capacity
2. Push maximum flow through this path
3. Repeat until no path can be found.

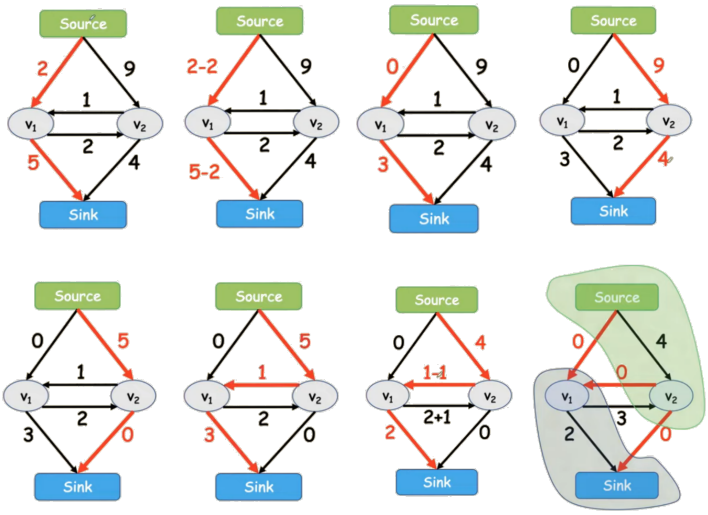
Maxflow in CV :

- Specialized algorithms for grid problems.
  - Grid graphs
  - Low connectivity (m ~ O(n))
- Dual search tree augmenting path algorithm
  - Finds the approximate shortest augmenting paths efficiently.
  - High - worst case complexity
  - Empirically outperforms other algos on vision problems.





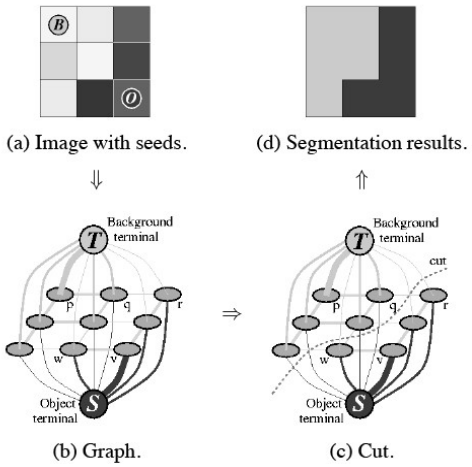
Example :



Mincut :

We want to find the minimum st-cut i.e., let C be a cut (set of edges) and sum of all the weights of the edges in C are minimized.

$$\text{Minimize} \sum_{i \in A, j \in B} W_{ij}$$



Pixels that relatively have the same intensity should be kept together, cut the edges that are between the pixels that have large intensity differences between them.

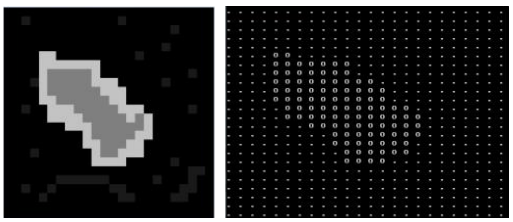
Edge exists between adjacent pixels ; weight of the edge is :

$$W_{ij} = \frac{1}{\text{dist}(i,j)} e^{-\frac{1}{2\sigma^2}||I_i - I_j||^2}$$

$$(I_i \approx I_j) \Rightarrow e^0 = 1$$

$$(I_i \not\approx I_j) \Rightarrow e^{-big} = 0$$

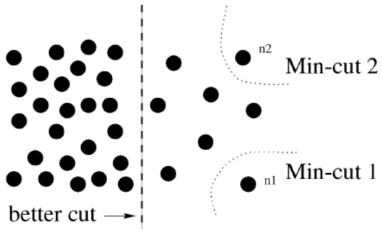
For neighbours with low weight, graph cut algorithm inclines towards cutting the edge. To teach how similar we are to background or foreground, we let the user scribble on the image to denote some initial foreground and background pixels. Those scribbles basically form probability distribution. F_F (colour), F_B (colour)



1. Scribbled pixels are forced to stick with one terminal i.e., $W_iF = \infty$; $W_iB = 0$.
2. Other pixels :
   a. $W_{iF} = -\lambda \log(F_B(i))$
   b. $W_{iF} = -\lambda \log(F_B(i))$
3. F_B is low implies its log value becomes -infinity. So we leave it with foreground.

Min-cut algorithm can directly result in trivial solutions such as isolating single pixel.

When A consists of say one pixel and B consists of another region of pixels, this summation may be restricted to a small set and hence may have a smaller value when we sum all of them. This means min-cut may favor regions where there is only one pixel which may be an outlier, this does not result in a good segmentation for the image. We use normalised cut to overcome this.



Normalised Cuts for Segmentation :

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

This improves the min-cut problem.

We define assoc(A,V) = assoc(A,A) + assoc(A,B) as the sum of all weights associated with vertices in A where :

$$\text{assoc}(A,B) = \sum_{i \in A, j \in B} W_{ij}$$

assoc(A,V) is to ensure that denominator contains the number of pixels in each of these regions. If one of the regions has only one pixel, then the corresponding quantity becomes very high, then the term corresponding to the other region will get pulled down. This penalizes large segments.

While computing an optimal normalized cut is NP-Complete, there exist appropriate solutions.

Recursive Normalized Cuts :

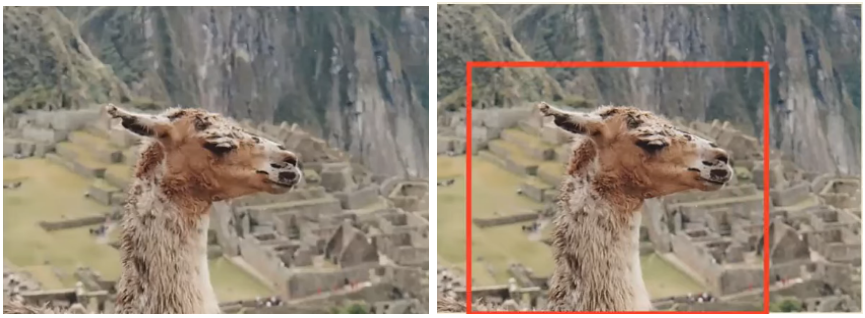
> When the image has to be segmented into multiple parts/segments we do recursive normalized cuts

1. Given an image or image sequence, set up a weighted graph: G=(V,E)
   a. Vertex for each pixel
   b. Edge weight for nearby pairs of pixels
2. Solve for eigenvectors with the smallest eigenvalues: (D –W)y=2Dy, where
   a. Use the eigenvector with the second smallest eigenvalue to bipartition the graph
   b. Note: this is an approximation
3. Recursively repartition the segmented parts if necessary.

Example :

GrabCut :

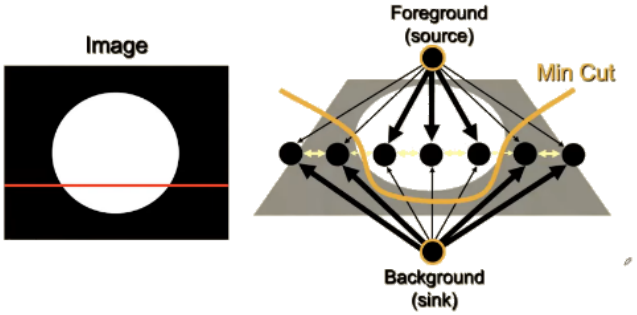
This is a user interactive foreground extraction using iterated Graph cuts. We define energy function based on user input.



Advantages :

1. Less user input : only rectangle
2. Handle colour
3. Extract matte as post-process

Use basic graph cut for segmentation :



Graph cuts for Foreground Extraction :

Assume we know that the foreground is white and the background is black.

Data term = whiteness

(cost of assigning label)

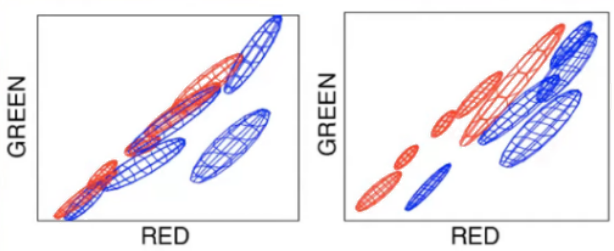
Regularization = colour match

(cost of separating neighbours)

Colour Data term :

Unary term here is defined based on colour data term.

A model of the foreground is created by creating a gaussian mixture model that fits inside the rectangle. So we choose a bunch of foreground pixels and do the clustering using GMM fitting and these gaussians act as a pdf of foreground of the image.

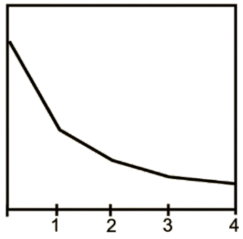


Iterated Graph Cuts :

1. User initialization
    a. Every pixel outside of rectangle is background
2. Learn foreground colour model for every pixel inside the rectangle
3. We can assign foreground and background cost based on this/pairwise cost
4. Graph cuts to infer the background
5. With new foreground model repeat from step 2

Example - iterations :

Energy convergence :



Background matting :

We might get a jagged edge for a particular class of problems. So, we use matte regularizer for a smooth transition between foreground and background.

In alpha matting we do not classify the boundary pixel as foreground or background instead we define a membership function (percentage of foregroundness/backgroundness)