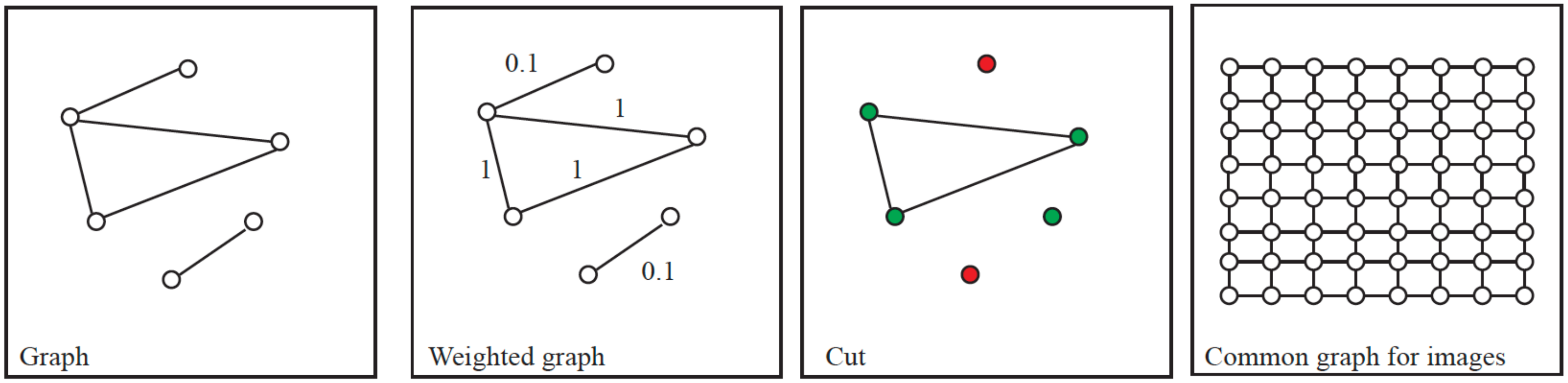
Graph based Image Segmentation

D.A. Forsyth,

University of Illinois at Urbana Champaign

Graphs

A *graph* is given by a set of vertices $\mathcal{V}$ and a set of edges (pairs of vertices, $\mathcal{E}$). The graph is a *weighted graph* if each edge has an associated weight. A *cut* partitions the graph into two sets of vertices (say $\mathcal{V}_A \subset \mathcal{V}$ and $\mathcal{V}_B \subset \mathcal{V}$) and two sets of edges ($\mathcal{E}_A \subset \mathcal{E}$ and $\mathcal{E}_B \subset \mathcal{E}$) such that $\mathcal{V}_A \cap \mathcal{V}_B = \emptyset$, $\mathcal{V}_A \cup \mathcal{V}_B = \mathcal{V}$, $\mathcal{E}_A$ contains only pairs of $\mathcal{V}_A$ vertices and $\mathcal{E}_B$ contains only pairs of $\mathcal{V}_B$ vertices. These ideas are quite visual (Figure 9.7).



Key idea of segmentation

- Break images/videos into large, useful pieces
  - internally coherent pieces
    - same color; same color and texture; etc
  - simplify
  - Identify key objects

There is a master recipe for image segmentation. This relies on *clustering*. a procedure that takes individual data items – for example, pixels, image patches – and produces blobs or *clusters* consisting of many similar data items.

Procedure: 9.1 *Image Segmentation: Master recipe*

Compute a feature vector at each pixel of the image, then cluster the feature vectors. Each segment consists of the pixels whose feature vectors are in the same cluster.

Grabcut

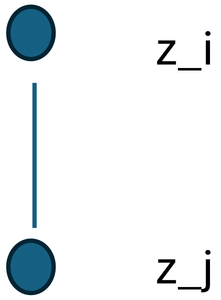
- Idea:
  - segment foreground from background
    - using rough model of appearance
    - ensure there are few/no isolated pixels
  - each pixel gets a binary variable
    - 1 for foreground, 0 for background
  - build probability model of features for both
    - P(feats[i]|z_i=1) and P(feats[i]|z_i=0)
    - many methods; normal distribution is one possibility
  - use graph to force pixels to “be like their neighbors”

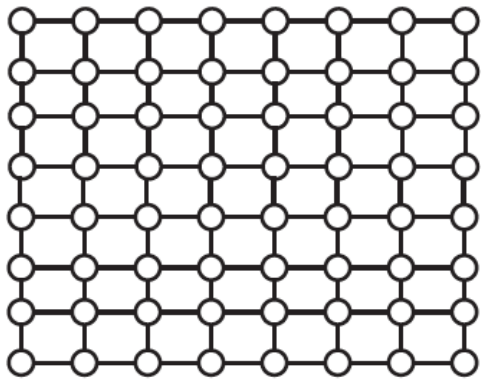
FG/BG only

- Choose z_i to minimize:

$$\sum_i \left(z_i\left[-\log P(\mathbf{f}_i|\text{foreground})\right] + (1-z_i)\left[-\log P(\mathbf{f}_i|\text{background})\right]\right).$$

If there are edges…





Common graph for images

- Cases:
  - (z_i, z_j) are:
    - (0, 0), (1, 1) - agree, both background/foreground
    - (1, 0), (0, 1) - disagree

$$z_i z_j E_{ij,11} + (1 - z_i) z_j E_{ij,01} + z_i (1 - z_j) E_{ij,10} + (1 - z_i)(1 - z_j) E_{ij,00}$$

Minimize

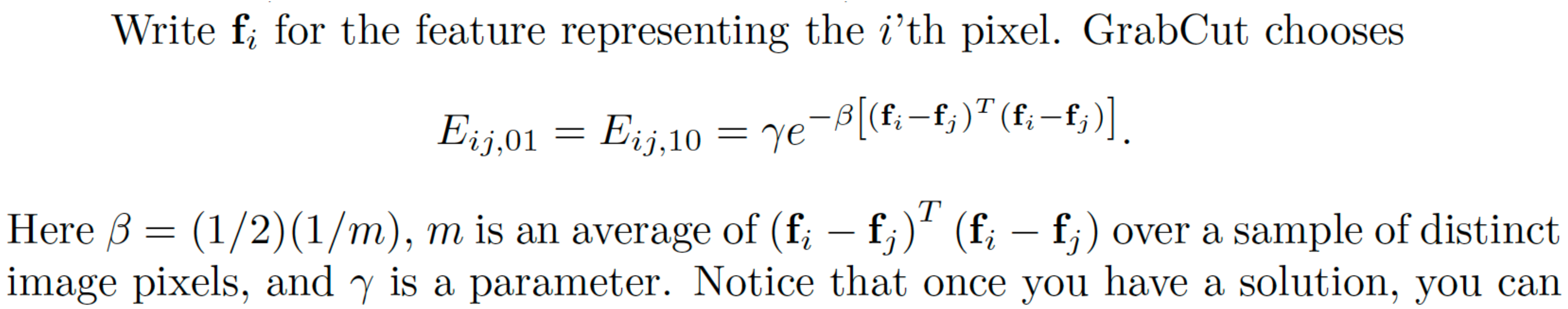
- by choice of z_i (V’s, E’s are known)

$$\sum_{i\in\mathcal{V}}(z_i V_{i,1} + (1-z_i)V_{i,0}) + \sum_{(i,j)\in\mathcal{E}}\left(\begin{array}{c} z_i z_j E_{ij,11}+ \\ (1-z_i)z_j E_{ij,01}+ \\ z_i(1-z_j)E_{ij,10}+ \\ (1-z_i)(1-z_j)E_{ij,00} \end{array}\right)$$

- General case v. nasty
- IF cheaper to agree than disagree
  - very fast algorithms are available (OOS)
  - this is the case we care about…

Setting up the cost function…

- E_ij00=E_ij11=0
- z_i neq z_j should be more expensive when their features are similar



Write $\mathbf{f}_i$ for the feature representing the i'th pixel. GrabCut chooses

$$E_{ij,01} = E_{ij,10} = \gamma e^{-\beta\left[(\mathbf{f}_i - \mathbf{f}_j)^T(\mathbf{f}_i - \mathbf{f}_j)\right]}.$$

Here $\beta = (1/2)(1/m)$, m is an average of $(\mathbf{f}_i - \mathbf{f}_j)^T(\mathbf{f}_i - \mathbf{f}_j)$ over a sample of distinct image pixels, and γ is a parameter. Notice that once you have a solution, you can

Parameter

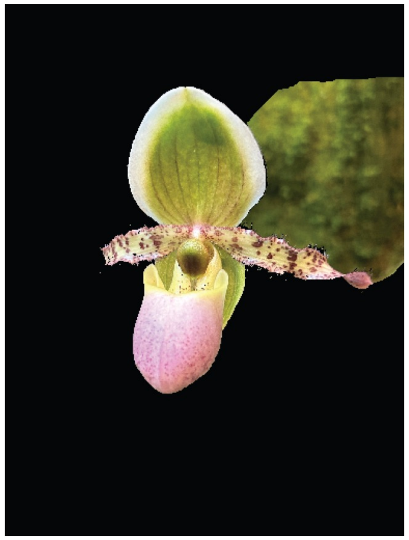
A large value of γ makes it very expensive for two pixels with an edge between them to have different z values. If you make the common choice of graph, then pixels are forced to agree with their neighbors in the absence of strong evidence they should disagree. This should remove isolated foreground or background labels. A large value of γ favors a shorter boundary between foreground and background, so too large a value of γ can result in an oversimplified boundary. As you reduce the value of γ, the boundary will become more complicated, but isolated pixels may appear.

Interactive segmentation

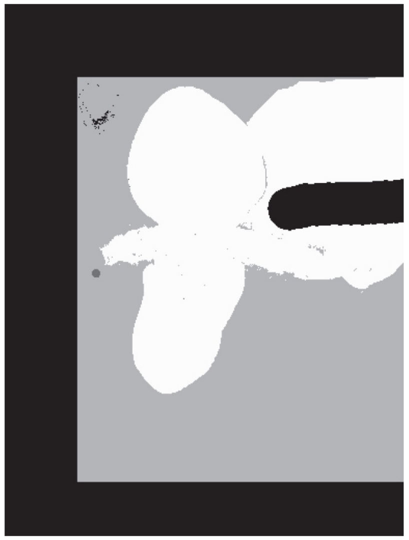
Image

Window

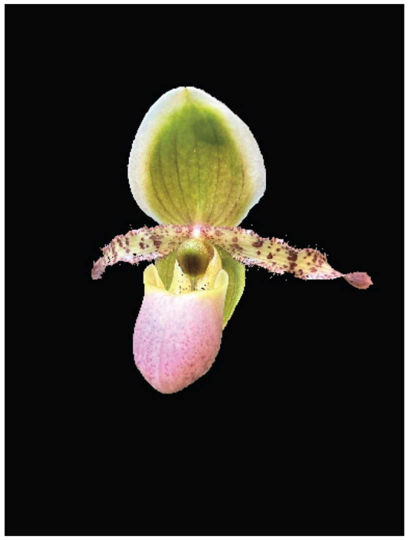
First segmentation



Strokes



Final segmentation



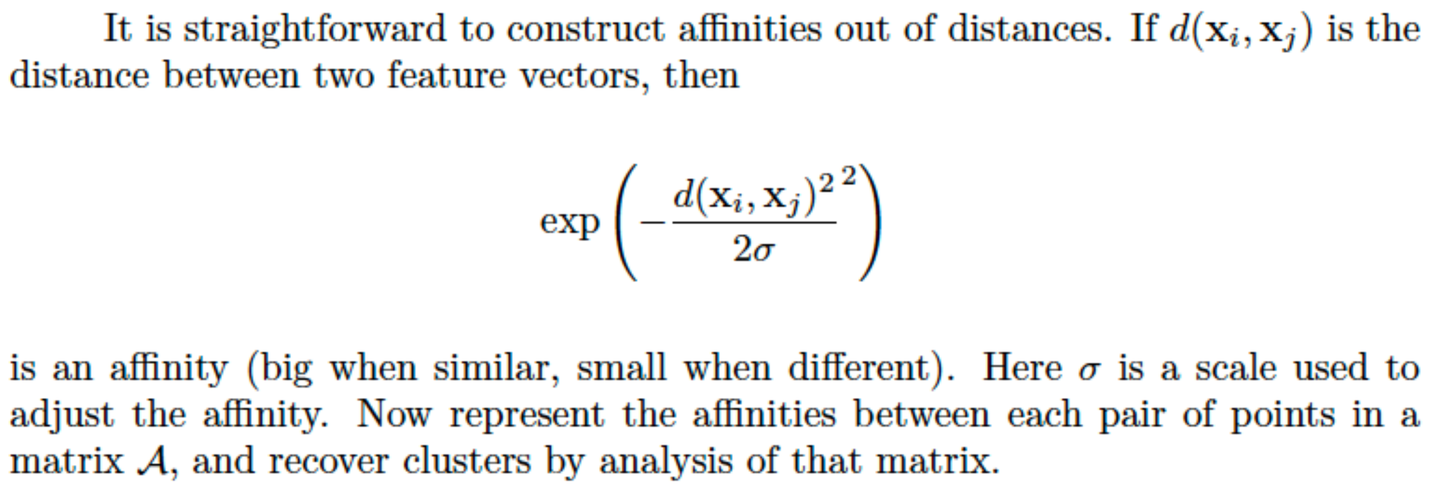
More segments

- GrabCut did two pieces
  - but needed foreground/background models

- To cluster
  - Build an affinity matrix
    - represents affinity between vertices
  - Use this to cut graph
    - components should
      - be internally coherent (high internal affinity)
      - be different (low affinity across cut)

Affinity matrices

- How similar is node i and node j?
  - typically each node is a pixel
  - large number – very similar
  - small number – very different
- Common case:
  - use distances

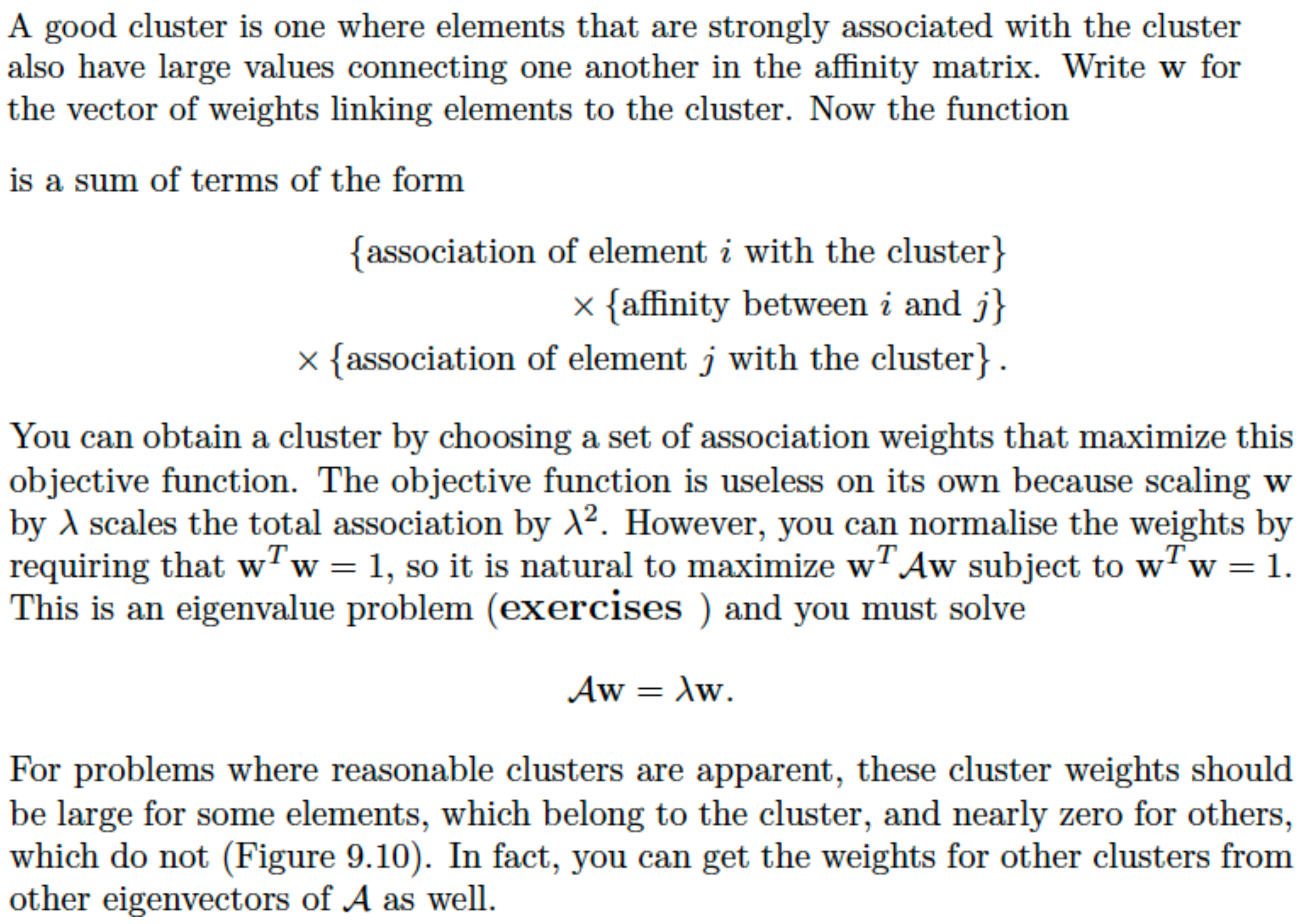


It is straightforward to construct affinities out of distances. If $d(\mathbf{x}_i, \mathbf{x}_j)$ is the distance between two feature vectors, then

$$\exp\left(-\frac{d(\mathbf{x}_i, \mathbf{x}_j)^{2}}{2\sigma}{}^{2}\right)$$

is an affinity (big when similar, small when different). Here σ is a scale used to adjust the affinity. Now represent the affinities between each pair of points in a matrix $\mathcal{A}$, and recover clusters by analysis of that matrix.

The simplest clustering...



A good cluster is one where elements that are strongly associated with the cluster also have large values connecting one another in the affinity matrix. Write **w** for the vector of weights linking elements to the cluster. Now the function

is a sum of terms of the form

$$\begin{aligned}&\{\text{association of element } i \text{ with the cluster}\}\\ &\quad\times\{\text{affinity between } i \text{ and } j\}\\ &\quad\times\{\text{association of element } j \text{ with the cluster}\}.\end{aligned}$$

You can obtain a cluster by choosing a set of association weights that maximize this objective function. The objective function is useless on its own because scaling **w** by λ scales the total association by λ^2. However, you can normalise the weights by requiring that $\mathbf{w}^T\mathbf{w} = 1$, so it is natural to maximize $\mathbf{w}^T\mathcal{A}\mathbf{w}$ subject to $\mathbf{w}^T\mathbf{w} = 1$. This is an eigenvalue problem (**exercises**) and you must solve

$$\mathcal{A}\mathbf{w} = \lambda\mathbf{w}.$$

For problems where reasonable clusters are apparent, these cluster weights should be large for some elements, which belong to the cluster, and nearly zero for others, which do not (Figure 9.10). In fact, you can get the weights for other clusters from other eigenvectors of $\mathcal{A}$ as well.

Eigenvectors are natural

- They’re permutation covariant

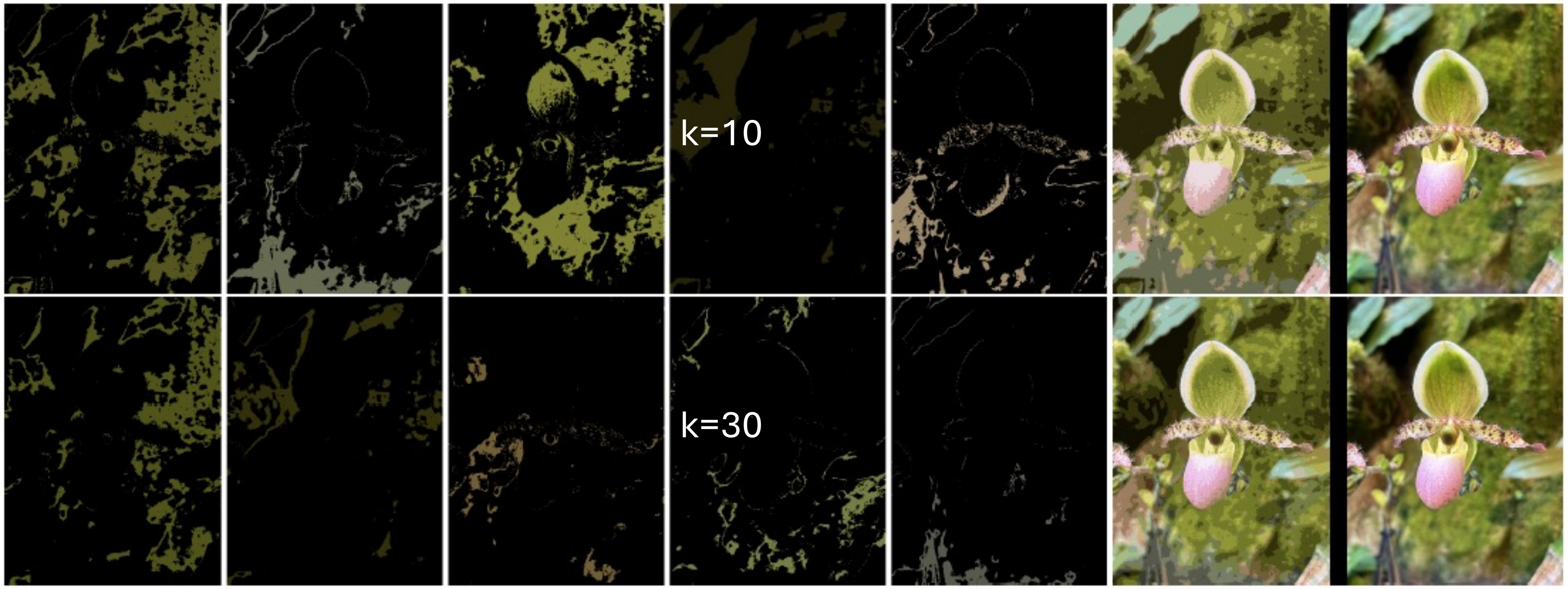
Normalized cuts

Formalize this as decomposing a weighted graph V into two components A and B and scoring the decomposition with

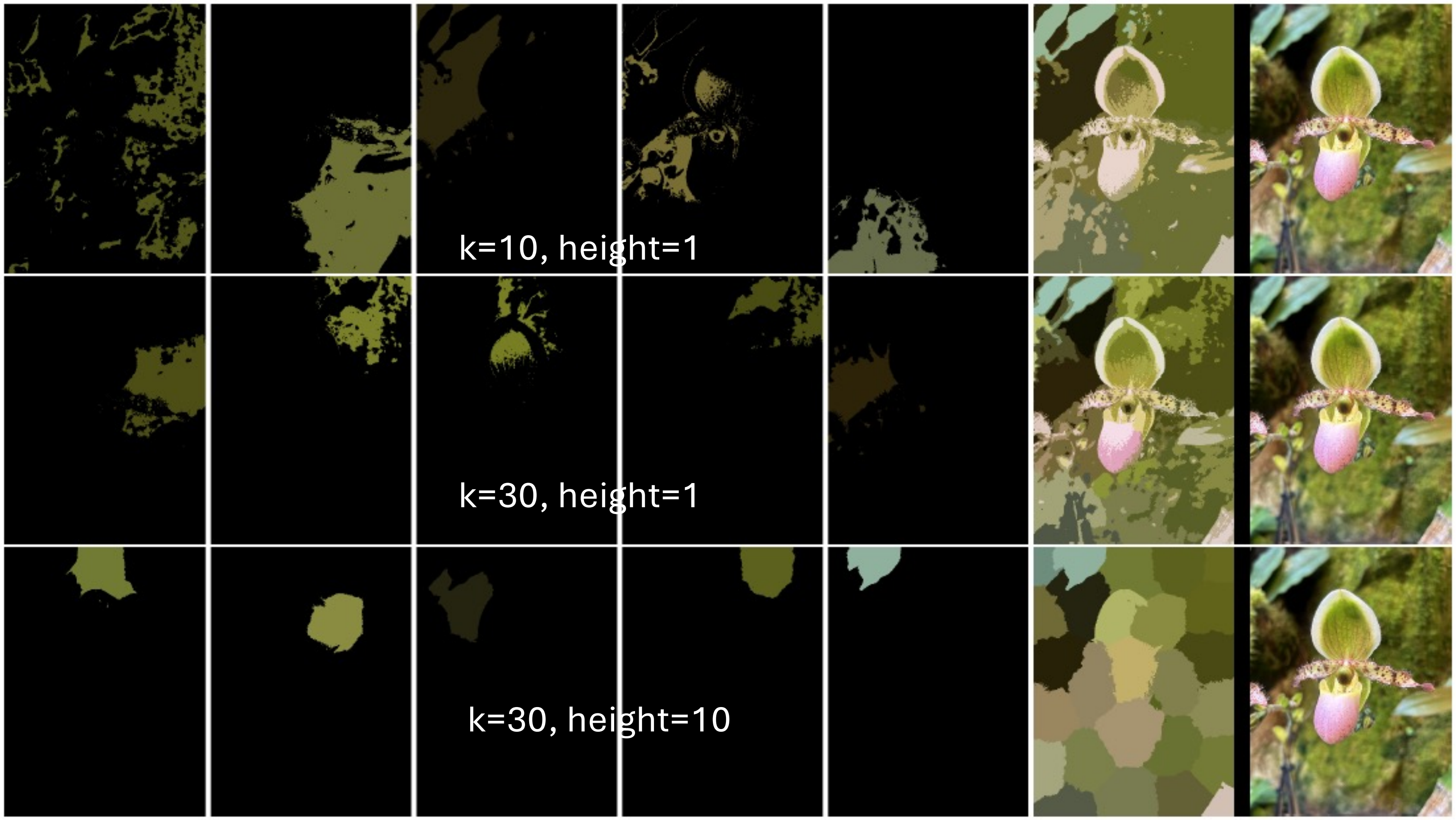
$$\frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)}$$

(where $cut(A,B)$ is the sum of weights of all edges in V that have one end in A and the other in B, and $assoc(A,V)$ is the sum of weights of all edges that have one end in A). This score is small if the cut separates two components that have few edges of low weight between them and many internal edges of high weight. One seeks the cut with the minimum value of this criterion, called a *normalized cut*. Actually finding this cut is algorithmically tricky. Reasonable approximation procedures are known, but are out of scope. The criterion is successful in practice (Figure 9.11 and 9.12). At least one efficient implementation is available for download (`https://ncut-pytorch.readthedocs.io/en/latest/`).

Normalized cut segmentations



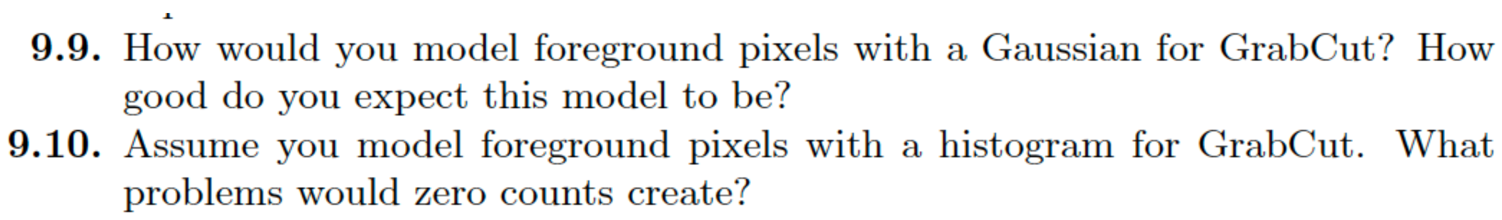
Normalized cut segmentations



Evaluating segmentations

- Compare to human segmentations of scenes
- Detailed comparison requires care
  - humans don’t always agree
  - boundaries may not be on exactly the right spot
- There is now an established procedure (notes)

Things to think about



9.9. How would you model foreground pixels with a Gaussian for GrabCut? How good do you expect this model to be?

9.10. Assume you model foreground pixels with a histogram for GrabCut. What problems would zero counts create?