Programming Assignment 2: Segmentation

CS 4670 Spring 2018

February 23, 2018

Key Information

Assigned:	Friday, February 23rd (Code is on CMS)
Due:	Friday, March 9th
Files to Submit:	`segment.py`, `readme.txt`

This project can be done **individually or in pairs**.

Overview

The goal of this assignment is to write several functions that segment images into perceptually distinct clusters of pixels. You will take three approaches. In Part A, you will implement an edge detection routine with signal processing methods. In Part B, you will implement a stochastic color-based segmentation algorithm using a k-Means clustering approximation. In Part C, you will implement a graph-based segmentation scheme.

Part A: Edge Detection

Background

Recall from lecture that finding segmenting an image is roughly equivalent to finding the boundaries or edges of an image, and that one easy way to do edge detection is by finding the gradient of an image.

One commonly used approximation for the directional derivative of an image is convolution with the Sobel filter. The Sobel filter is a linearly separable filter which approximates the difference of Gaussians. The Sobel filter in the positive-x direction has the following form:

+1	0	-1
+2	0	-2
+1	0	-1

You can derive a similar kernel for the +y direction.

Tasks

For Part A, you will write five functions. To calculate and display gradients (multidimensional derivatives) of images in `segment.py`. The function headers are as follows:

- `normalizeImage(cvImage, minIn, maxIn, minOut, maxOut)`
- `getDisplayGradient(gradientImage)`
- `takeXGradient(cvImage)`
- `takeYGradient(cvImage)`
- `takeGradientMag(cvImage)`

The `normalizeImage` and `getDisplayGradient` are visualization functions that will help you to make the results of the other three functions human-understandable. The other three functions should use a custom Sobel implementation effectively and efficiently. Make sure to separate any separable kernels, and call Numpy/Scipy/OpenCV functions where applicable.

Implementation notes

You may use most methods from any of the libraries mentioned in class, including convolution methods, but you may *not* use any third-party functions which construct a Sobel filter for you, filter the image with a Sobel kernel you did not create, or take a derivative through some other method. You will also lose points if you do not use the fact that the Sobel operator is linearly separable.

Part B: k-Means Clustering Segmentation

Background

Recall the k-means clustering problem from lecture:

k-Means Clustering Problem

Given some collection of points $(p_1, p_2, \cdots, p_n)$, partition the points into k disjoint segments $(S_1, S_2, \cdots, S_k)$ to minimize the variance of each segment. That is, find

$$\arg\min_{S_1,\cdots,S_k} \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2 .$$

where μ_i is the mean of all the points in segment S_i.

Recall also that by rewriting our image as a set of (R, G, B, x, y) tuples we can use a k-means solver for segmentation.

Tasks

For Part B, you will fill in pieces of a k-means segmentation solver in `segment.py`. You will work on two functions:

- `chooseRandomCenters(pixelList, k)`
- `kMeansSolver(pixelList, k, centers, eps, maxIterations)`

The `chooseRandomCenters` function should select k random rows from the input list. The `kMeansSolver` is partially done. You will have to fill in a few critical sections of code corresponding to important steps in the k-means solver algorithm. These sections are bracketed by `# TODO:PA2` and `# END TODO:PA2` comments.

Implementation notes

Your code should weight pixel R,G,B,x,y values so that the range of values that can be taken on by each channel is the same as the range of values that can be taken on by the x- and y-coordinates. k-Means is a randomized method, so you may have to run it a few times to get a nice result. The GUI (see below) has a checkbox that allows you to segment in Hue-Saturation-Value space instead fo RGB space. Our code handles this, so don't worry about it. As in Part A, feel free to use Numpy/Scipy/OpenCV functions as long as they do not trivialize the problem. For instance, you may not use the `scipy.cluster.vq` function to do do k-means segmentation for you.

Part C: Normalized Cuts Segmentation

Background

Another popular view of segmentation we saw in lecture is as a graph-cuts problem. The formulation is straightforward. Given an image with height m and width n, we construct a weighted graph $G = (V, E)$ with nm nodes, one for each pixel.

Our goal is to split this graph into two segments of nodes, such that we keep related pixels together. Suppose between every two nodes i, j there is an edge with weight w_{ij} that reflects the likelihood that i and j appear in the same segment.

For this assignment we will use a simple exponential weighting.

$$w_{ij} = e^{\frac{-\|C_i - C_j\|_2^2}{\sigma_C^2}} \cdot \begin{cases} e^{\frac{-\|X_i - X_j\|_2^2}{\sigma_X^2}}, & \text{if } \|X_i - X_j\|_2^2 < r \\ 0, & \text{otherwise} \end{cases}$$

Here C_i and C_j are the colors of pixels i, and j. X_i and X_j are their respective image-coordinates, and r is a "neighborhood" radius, beyond which we do not bother to compare pixels.

The Normalized Cuts method uses this graph to find an approximate minimum to the Ncut error function,

$$\text{Ncut} = \frac{\text{cut}(S_1, S_2)}{\text{vol}(\text{S}_1)} + \frac{\text{cut}(S_1, S_2)}{\text{vol}(\text{S}_2)},$$

where

$$\text{cut}(S_1, S_2) = \sum_{i \in S_1} \sum_{j \in S_2} w_{ij},$$

and

$$\text{vol}(S) = \sum_{i \in S, j \in V} w_{ij}.$$

If we construct a matrix W to represent this graph, we can find an approximate minimum to the Ncut by finding the second smallest eigenvector of the Normalized Ggraph Laplacian. That is we find z such that

$$\left(I - D^{-1/2} W D^{-1/2}\right) z = \lambda_2 z$$

where D is the diagonal matrix of row sums of W and λ_2 is the second smallest eigenvalue of $\left(I - D^{-1/2} W D^{-1/2}\right)$. We then compute

$$y = D^{-\frac{1}{2}} z$$

and threshhold y at 0. Pixels with positive values go into segment S_1. The others go into S_2.

Tasks

This is the technique you will implement in Part C of this programming assignment. You will fill in the following four functions in `segment.py`:

- `getColorWeights`
- `getTotalNodeWeights`
- `approxNormalizedBisect`
- `reconstructNCutSegments`

In `getColorWeights`, you will construct W, the weighted adjacency matrix of the input image. In `getTotalNodeWeights`, you will construct D, the diagonal row-sum matrix. In `approxNormalizedBisect`, you will call `scipy.linalg.eigh` on an appropriately constructed matrix to calculate y. Finally, in `reconstructNCutSegments`, you will threshold y and color the original image to visualize the resultant segmentation.

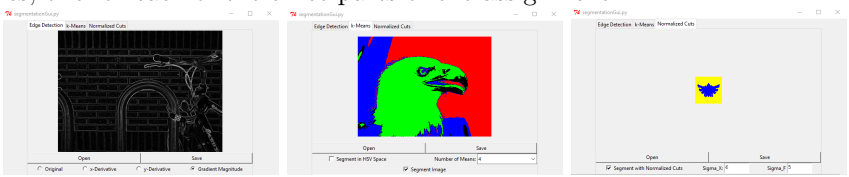
Implementation notes

This version of Normalized Cuts leaves out a lot of acceleration code. You should only try this method on small images (¡ 64x64 pixels). On our computers, constructing W is the limiting step, and can often take two minutes or more before moving on to the eigensolve. We only ask you to partition images into two segments with this method. The full method recursively segments the results, and can segment more complicated images. As in the first two parts, feel free to use Numpy/Scipy/OpenCV functions as long as they do not trivialize the problem.

Testing and Debugging

The GUI

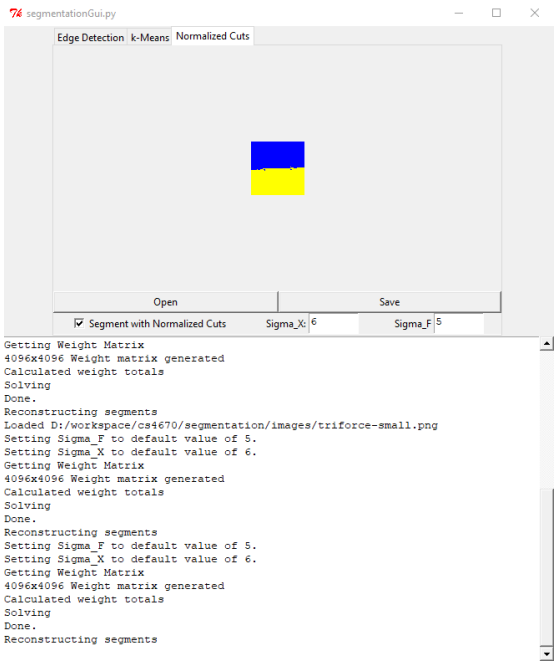
You can test your code with GUI in `segmentationGui.py`. The GUI has three panes, one for each of the three parts of the assignment.



Whenever you access your code from the GUI, it will automatically reload `segment.py`, so you don't have to restart the GUI whenever you update your code.

The Log

While running the GUI, your print statements and errors will be re-routed to the log in the bottom half of the window.



If you don't like this behavior , you can deactivate it in `segmentationGui.py` by commenting out the lines that say:

```
sys.stdout = LogRedirector(self.log)
sys.stderr = LogRedirector(self.log)
```

If you use `pdb`, the python debugger, you will need to comment these lines out.

Automated Tests

Automated tests will be released early next week with instructions for use.

Deliverables

You should submit two files, `segment.py`, with all of your work, and `readme.txt`, explaining your submission. You should fill in every function marked with `# TODO:PA2` in `segment.py`.

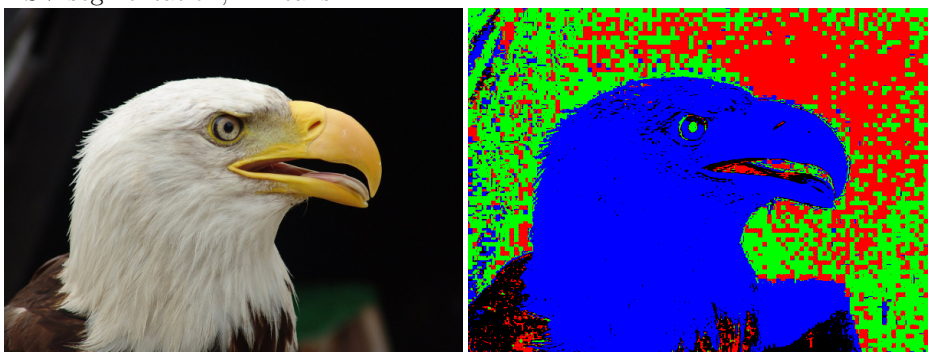
Expected outputs

Gradients

k-Means

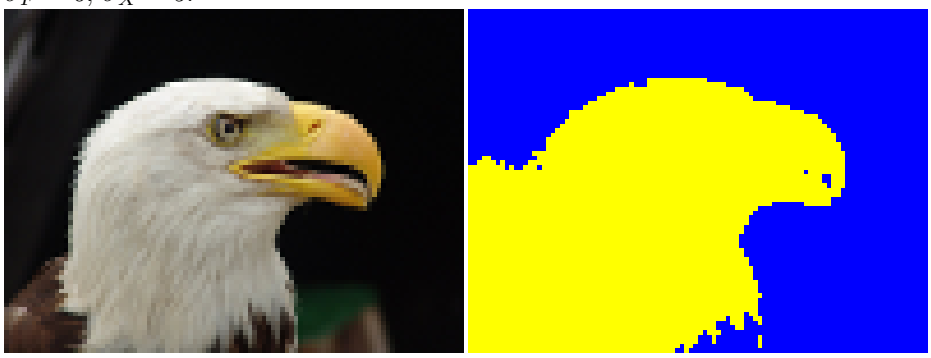
RGB segmentation, 12 means.

HSV segmentation, 4 means.



Normalized cuts

$\sigma_F = 6$, $\sigma_X = 5$.



$\sigma_F = 6$, $\sigma_X = 5$.