CMSC 426, Fall 2017
Problem Set 3
Due March 14

In this problem set you will implement a mincut approach to image segmentation. This algorithm has been discussed in class. The class web page also contains a reference to the paper by Boykov and Jolly which this project is based on.

To get started, download the maxflow code of Boykov and Kolmogorov from: http://vision.csd.uwo.ca/code/ The zip file you want is: maxflow-v3.01.zip. Note that this is C++ code.

Next, download Miki Rubenstein's Matlab interface to the maxflow code from: http://www.mathworks.com/matlabcentral/fileexchange/21310-maxflow. Note that per instructions in the readme file, you should place the C++ maxflow code in a subdirectory <lib_home>/maxflow-v3.0>.

Follow the instructions in Rubenstein's README file to make and test the maxflow code. In addition to the instructions given, you should add the flag

- largeArrayDims

to the end of the mex command in the make file.

You may see some warnings when you make the code, but you should be able to ignore these. To make sure that everyone is able to run the code on their machines, you are **required to download and make the code by Monday, March 6.** If you encounter any problems and report them to us by Monday, we will work hard to resolve them and give you an extension if necessary. If you do not report any problems in running the code by Monday, we will not be sympathetic if you encounter problems later.

Hint: The maxflow code is version 3.0.1. The Matlab interface expects it to be in a directory named maxflow-v3.0. Don't let this confuse you. As long as you name the directory maxflow-v3.0, they should work together fine.

For this assignment you will need to use sparse matrices. Look at the documentation for them: `help sparse`. In particular, you will need to create a sparse matrix that would be really huge if it wasn't sparse. You won't be able to do this by creating the full matrix and then making it sparse. The Matlab code you've downloaded contains examples that should guide you in this.

1) **10 points.** Let's try to use this code in the simplest way we can. Suppose we have a Source node, S, a Sink node, T, and three other nodes, A, B, C. The edge weights are given by:
   - w(A,B) = 8,
   - w(A,C) = 2,

- w(B,C) = 2.

This means that nodes A and B are likely to be grouped together, because it is expensive to cut the link between them. Suppose that the edge weights to the source and sink are given by:

- w(S,A) = w(S,B) = 0, w(S, C) = 10
- w(T,A) = 10, w(T,B) = w(T,C) = 0

This means that node C has a strong link to the source, while node A is strongly linked to the Sink. Intuitively, this should lead to a segmentation in which Node C winds up grouped with the source, while A and B are grouped with the sink. Set up and call the maxflow code to show that it achieves this result. Include your code and results in a write-up.

Hint: The test function test1.m, which is included in the Matlab code that you downloaded, shows how to do this.

2) **10 points.** We'll now create edge weights that relate to image cues about whether or not we want to connect two pixels together into the same group. We will be creating a graph in which every pixel corresponds to a node, and there is an edge between two pixels' nodes if they are 4-connected neighbors (that is, if they are neighbors horizontally or vertically; diagonal neighbors do not count). For example, I(x,y) and I(x+1,y) are neighbors, while I(x,y) and I(x+1,y+1) are not neighbors. I'll call these neighboring nodes A and B, and refer to their intensities as I(A) and I(B). We will define the edge weight between them as:

$$w(A,B) = e^{-\frac{(I(A)-I(B))^2}{2\sigma^2}}$$

σ is a parameter. Notice that I(A)-I(B) is essentially the image derivative as we go from pixel A to B (in this example, in the x direction).

Now you will write a function to compute these edge weights with the form: `A = image_edge_weights(I, sigma).` I is a grayscale image, and A is a sparse matrix that contains the weight between two neighboring pixels. To create a linear sequence of nodes for the pixels, we will number the pixels columnwise. That is, if we had a 2x3 image, the pixels would be numbered as:

1	3	5
2	4	6

Given an image, I that looked like:

40	30	20
60	50	20

and with $\sigma = 20$, `image_edge_weight` would produce an A like

```
A =
```

```
(2,1)       0.6065
(3,1)       0.8825
(1,2)       0.6065
(4,2)       0.8825
(1,3)       0.8825
(4,3)       0.6065
(5,3)       0.8825
(2,4)       0.8825
(3,4)       0.6065
(6,4)       0.3247
(3,5)       0.8825
(6,5)       1.0000
(4,6)       0.3247
(5,6)       1.0000
```

For example, in computing A(2,1), node 2 in the graph would refer to the pixel at location (2,1), while node 1 would refer to the pixel at (1,1). We would use the values 40 and 60 and compute:

$$w(2,1) = 1 * e^{-\frac{(40-60)^2}{2*20^2}} = .6065$$

Test your code by making sure it produces the appropriate result on the small image given above.

Hint: Look at the function test2, which comes with the maxflow code. This function has almost everything you need to complete this problem. In particular, it shows how to use the very convenient function, edges4connected, which is also included.

3) **30 points.** Next we will use these edge weights to segment an image. Write a function of the form:

```
S = segment_image_gray(I, F, B, sigma)
```

I will be a grayscale image. F and B will be binary images of the same size as I. Where F is equal to 1, a pixel is definitely known to belong to the foreground. Where B is 1, the pixel is definitely background. Other pixels are up for grabs. You can encode these constraints in the graph by placing a very strong weight on edges that connect the foreground pixels to the source, and edges that connect the background pixels to the sink. Your function should return a binary image, S, also of the same size as I. Where S has the value 1, there is a background pixel; where S is 0 there is a foreground pixel. Test your function using the following example of I, F, and B, and with σ=20. The resulting S is also shown.

```
I =

    10    15    10
    12    42    44
    48    50    50


F =
```

```
     0     0     0
     0     1     0
     0     0     0

B =

     1     0     0
     0     0     0
     0     0     0

S =

     1     1     1
     1     0     0
     0     0     0
```

To further test your program, run it on the grayscale Swan image on the class web page. Assume that all pixels in the rectangle with corners at (200,300) and (250,400) are part of the foreground, and all pixels in the rectangle with corners at (1,1) and (100,100) are part of the background. The image swan_segmented shows the results of my program on this image.
Hint: In this problem and the next, if your program is slow, feel free to shrink the image and run on a smaller version of it. However, you will also have to appropriately shrink the rectangles that specify foreground and background pixels.

4) **40 points.** This works pretty well on the swan because its boundary has pretty strong intensity gradients. Now you will use color information to improve performance on color images. The basic idea (as described in the Boykov and Jolly paper) is to tailor the weights between a pixel's node in the graph and the source and sink based on whether the color indicates that this particular pixel is foreground or background.

   To be more specific, in the last problem segment_image_gray was called with a matrix F which indicated pixels that are known to be foreground. Using only these pixels, we create a color histogram. First I'll describe how to create a histogram for gray scale images, and then how to do this for color images.

   To create a histogram for gray scale images, you need to choose a bin size. Let's say the bin size is 20. Then the first bin contains a count of all the pixels that have intensity between 0 and 19. The next bin counts all pixels between 20 and 39. Keep doing this until you reach a bin with intensities between 240 and 255 (this bin will be smaller, but that's ok.

   To create a color histogram, you do the same thing, but using all three color channels. Remember, a color image is just a 3D Matlab matrix. So, for example, if I is a color image and you want to get a matrix of the red values for every pixel, you could say: `R = I(:,:,1).` For a color histogram, the first bin will contain a count of all the pixels in which the red value is between 0 and 19, the green value is between 0 and 19, and the blue value is between 0 and 19. The

next bin could have red values between 20 and 39, green values between 0 and 19, and blue values between 0 and 19. With a bin size of 20, the gray scale histogram had 13 bins. In color, with a bin size of 20, we would have 13^3 bins. That's quite a few, so you might want to use a larger bin size. Check your code by making sure that the sum of the values in all the bins equals the number of the pixels in the image. Every pixel should add to exactly one bin.

We can then normalize this histogram so that the value in the bins sum to 1 (just divide the value in each bin by the number of foreground pixels). Now, let's suppose that a particular pixel, p=(i,j), is not known to be foreground or background (ie., F(i,j)=0, B(i,j) = 0). Then, if we find the bin that the color of p belongs to, we will get the probability that the foreground would produce the color that we see in p. In the same way, we can figure out the probability that the foreground will produce this color.

Let's look at an example. For simplicity, I'll suppose images are grayscale, but you will need to follow the same idea with color. If our image looked like:

30	30	20
40	30	30

F looked like:

1	0	0
1	0	0

and B looked like:

0	0	0
0	0	1

And we were trying to classify one of the middle pixels, which has an intensity of 30, we would say that the foreground produces this intensity with probability ½, while the background produces this intensity with a probability of 1.

There is one problem with this. If we were trying to classify the pixel in the top right, which has an intensity of 20, both probabilities would be 0. This will cause our approach to blow up. So we *smooth* the histogram before normalizing it, by adding a constant value to every bin in the histogram. I used a constant that is equal to the number of pixels in the image divided by 20 times the number of bins.

Finally, for these pixels that aren't known to belong to the foreground or background, we connect them to the source and sink using weights of:
$-\lambda log(Pr(color|Object))$ and $-\lambda log(Pr(color|Background))$
Pr(color|Object) here is just the value you get from looking up the pixel in the normalized histogram, and similarly for the background probability. Here λ is another parameter. You can just set it to 1 and get good results, or you can play around with it a little.

Implement a color segmentation algorithm of the form:

```
S = segment_image_color(I, F, B, lambda, sigma);
```

Use your code to segment the dog image on the class web page, using foreground and background defined as:

```
F = zeros(size(I,1), size(I,2));
F(100:200, 130:280) = 1;
B = zeros(size(I,1), size(I,2));
B(:, 1:50) = 1;
```

Here I is the dog image. The results of running my program are shown on the class web page.

5) **10 points.** Finally, integrate your code with the GUI.m provided on the class web page, looking at the documentation in the gui code. The gui will allow you to interactively mark rectangles in an image as foreground or background. Using this, segment a new image of your own choosing. You may find it helpful to play around with the parameters of the segmentation method to get better results. For example, using lambda = .2 seems to produce a cleaner segmentation. As always, you may want to experiment with smaller images first. Show the results of your segmentation. Briefly describe the advantages and disadvantages of this segmentation tool.
6) **Challenge Problem (up to 20 points):** Use this segmentation tool to combine two or more images. To produce a good effect you may need to resize images, or even to rotate them (see imrotate). You might also want to manipulate the colors or intensities in an image. You can also try to get two images to blend together better by smoothing the boundary between them. Feel free to ask me if you want some hints on how to do this, or other pointers to references on creating good images. Document all steps you took in creating your image. A prize will be given for the best image.