EECS 504 Foundations of Computer Vision: HW4

Term: Winter 2018
Instructor: Jason J. Corso, EECS, University of Michigan
Due Date: 3/11 23:59 Eastern Time

Constraints: This assignment may be discussed with other students in the course but must be written independently. Programming assignments should be written in Matlab. Over-the-shoulder Matlab coding is strictly prohibited. **Web/Google-searching for background material is permitted. However, everything you need to solve these problems is presented in the course notes and background materials, which have been provided already.**

Goals: Deepen understanding of image as graphs; integrate material discussed in the previous lectures.

Data: You need to download `hw4.zip` to complete this assignment. All paths in the assignment assume the data is off the local directory.

Problem 1 (15): Mumford-Shah Piecewise Constant

1. (5) Recall the Potts model we discussed when introducing the concept of image as functions. Based on that model, the energy of an image I is

$$E(I) = \beta \sum_{s=1}^{n-1} \sum_{t=1}^{n} \Big(\mathbf{1}\big(I(s,t) \neq I(s+1,t)\big)\Big) + \beta \sum_{s=1}^{n} \sum_{t=1}^{n-1} \Big(\mathbf{1}\big(I(s,t) \neq I(s,t+1)\big)\Big). \tag{1}$$

   Implement the Potts model in `potts.m` using **range map operations**. Run `potts_load.m` and it will load two images, call `potts.m` and display the results. Use $\beta = 1$ in your implementation. Note that the image should be converted to double format (with pixel values in [0,1]).

   Attach the output images and include your code as verbatim in your writeup. Also submit your `potts.m` to Canvas.

2. (5) Implement a discrete piecewise constant variant of the Mumford-Shah model:

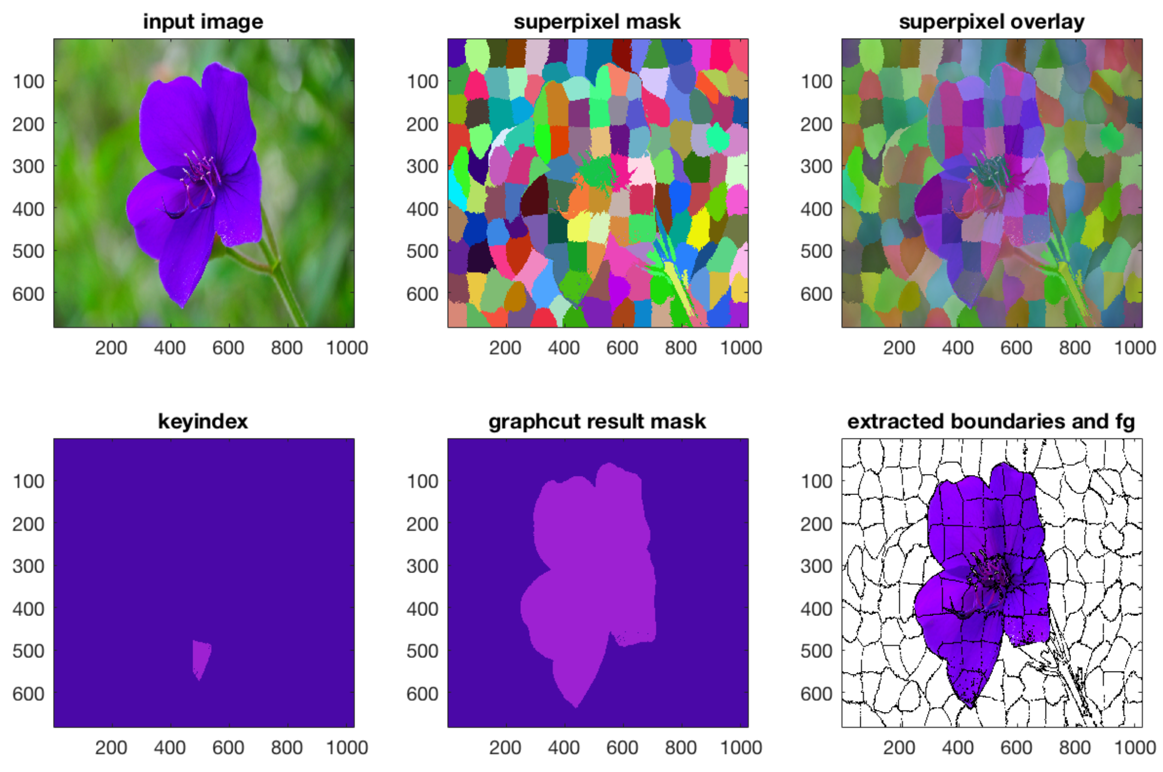
$$E_{\text{PC}}(\hat{f}, \partial) = \alpha \int\int_R (\hat{f} - f)^2 dxdy + \gamma |\partial| \ . \tag{2}$$

   and name your file as `mumford_shah.m`. Think about how this is related with last question. A noisy image `f.png` and its estimation `fhat.png` are provided. Evaluate the model on the given images with $\alpha = 1$ and $\gamma = 0.1$ and report the value. Include your code as verbatim in the writeup and submit the original file to Canvas. (Same with q1, convert the image to double format.)

3. (5) In order to denoise `f.png`, we hope to find an $\hat{f}$ such that E_{PC} is minimized. Is `fhat.png` a minimizer of E_{PC}? Demonstrate it and explain.

Problem 2 (55): Foreground-background graph-cut

We discussed the figure-ground graph-cut case study at length in class. You will implement this method for figure-ground extraction on color images using a superpixel graph and color histograms as the feature. A basic MRF is implemented in the graph. The figure below shows the expected results and facets of the assignment.



The provided code includes the core framework, an implementation of the SLIC superpixel method, and an implementation of the Ford-Fulkerson algorithm for computing the max-flow. You will implement the histogram feature representation and all aspects of taking the superpixel result and implementing the graph on top of it as well as reading out the results of the graph cut as a two-class segmentation.

The file `example.m` provides a full run through the whole method for you. Your results from running it will be submitted. You will also need to run additional methods.

Note that p-code is provided with a `j_` prefix on the files you need to implement. This is for your help and benefit, and it is used to compute accuracy for submission. You will not report results from the p-code as your own, nor will you make any attempt to deconstruct the p-code.

1.(8) Color Histogram Features

Implement the missing body of code in `histvec.m`, which creates a color histogram feature vector. Follow the comments in the code for the details. We call this with `C = reduce(im,C,S);`.

Run `q1.m`, which will generate an output image `q1_result.png`. Include this output in the writeup. Submit your `histvec.m` to Canvas.

Also, inspect the code in `q1.m` and answer the following question in one sentence: Describe the object in the image that is covered by the superpixel used to compute the histograms in `q1.m`.

2.(15) Superpixel adjacency

(a) (7) Implement the missing body of code in `segNeighbors.m`, which computes the adjacency matrix for the superpixel graph. Follow the comments in the code for the details. We call this in the graph cuts code.

Run `q2.m`, which will generate an output image `q2_result.png`. Include this output in the writeup. Submit your `segNeighbors.m` to Canvas.

(b) (5) Implement a small function to compute the average node degree. Report the average degree and include the code as plain text in the writeup. (No need to submit the original file.)

(c) (3) Why is the adjacency graph not a perfect banded diagonal matrix?

3.(17) Graph-cuts

(a) (8) Implement the missing two bodies of code in `graphcut.m`, which (1) creates the graph by defining the capacity matrix and (2) extract the results after running the max-flow/min-cut method. See the comments and refer to class notes for details.

Run `q3.m`, which will generate an output image `q3_result.png`. Include this output and submit your `graphcut.m` to Canvas.

(b) (5) Using the debugger, save an image of your capacity matrix before running graph cuts in `q3.m` and attach it to your writeup. In a few sentences, relate the adjacency matrix to the capacity image. Be sure to cover **all** nodes of the graph in your description.

(c) (4) Please explain why the capacity between adjacent nodes in the graph that have resulted from superpixels is downweighted with respect to the capacity between nodes in the graph and the special source and sink nodes.

4.(15) Graph-cuts Study

Use the `example.m` code here and change accordingly. You can change other parts of the code too if needed, but be clear to note it.

(a) (2) Run code as provided to run through a full example and show the resulting segmentation on the flower. Include the result in your writeup.

(b) (5) Run the code using the `flag1.jpg` image. Manually select a stripe on the flag. Are you able to get a full segmentation of the stripe and no other regions? If so, explain what you did to make this possible. If not, explain why this is hard. Include an rendering of the figure to substantiate your explanation either way.

(c) (5) Run the code using the `porch1.png` image. Are you able to segment the boots perfectly? If so, explain what you did to make this possible. If not, explain why this is hard. Include an rendering of the figure to substantiate your explanation either way.

(d) (3) On the `porch1.png` image again, are you able to segment either of the baskets perfectly? My guess is no. Can you describe (but do not implement) a way to change this system to make this more possible?

Problem 3 (30): Segmentation with minimum spanning forest

In problem 2, we build a graph from superpixels and partition the graph into two parts by finding the max-flow. For the same graph, now we try another way to do partition using Felzenszwalb-Huttenlocher algorithm, which can be applied to multi-object segmentation.

The basic idea of F-H algorithm is to find the minimum spanning forest in a graph. Recall the Kruskal's algorithm which finds the minimum spanning tree (MST). MST contains all the nodes in the graph while keeping the sum of edge weights minimal. As a variant, F-H doesn't force all the nodes to be connected; it requires as many as similar nodes to be grouped together but rejects the connection between nodes that are unlikely to be the same kind. F-H is different from the Kruskal's algorithm in that it adds an extra condition when merging two segments. The condition to accept an edge $E(v1, v2)$ as a part of the forest and merge two segments is following:

$$S_1 \neq S_2 \ AND \ weight(E(v1, v2)) < MInt(S_1, S_2).$$

S_1 and S_2 are the segments node v_1 and v_2 belong to, and

$$MInt(S_1, S_2) = \min(Int(S_1) + \frac{k}{|S_1|}, Int(S_2) + \frac{k}{|S_2|}),$$

where $Int(S)$ is the maximum edge weight in the segment S, $|S|$ is the number of nodes in S and k is a hyperparameter settled before program runs. Simply speaking, one edge can be accepted only when the nodes at two ends of the edge are in different segments and the edge weight satisfies the criterion above. After iterating all the edges ordered from lowest cost to highest cost, the remaining (unconnected) segments forms a partition of the graph.

1. (16) Build the graph in the similar way as you did in previous problem and fill the missing part in `fh_imgseg.m`. Note that the edge weight calculation is a little bit different. See more details in the comments. Feel free to use `j_` files or your own implementation in problem 2 to help you solve this problem.

   Then implement F-H algorithm in `fh.m`. `fh_imgseg.m` will call your function and visualize the segmentation for the purple flower image.

   Submit your `fh_imgseg.m` and `fh.m` to Canvas. And include the output image in your report.

2. (4) Explain why we calculate the weights in different ways for graph-cut and for F-H?

3. (6) A common postprocessing step used with F-H is to merge all of the small, unnecessary components that result from uneven regions. Implement this as another function within `fh.m`, which 1) iterates through the edges in the original image graph, in ascending order, and 2) for each edge, if it connects two distinct segments, AND if at least one of those segments contains less than min_size pixels in it, merge the segments.

   Try different values of min_size. Pick one that removes small fragments from the previous result and report it in your writeup. Attach the output image and your implementation of this function as verbatim in the writeup.

4. (4) Comment out the postprocessing function in `fh.m`. Run the program with $k = 1$ and $k = 20$ and include the output images in your writeup. Could you describe what role k is playing in this algorithm? (Think but no need to answer: what is the intuition behind the merge criterion?)

Submission Process: Submit a single pdf with your answers to these problems, including all plots and discussion. Submit the pdf to Gradescope. The entry code of this course is **9GEG8X**.

For coding assignments, include your code verbatim in your writeup. Put original program files in separate folders corresponding to problems and name your folders as `hw4p1`, `hw4p2`, etc. Pack all the folders into one zip file named as `hw4_<unique_name>.zip` and upload it to Canvas. The problem description will clarify whether you need to attach your code verbatim or turn in the original files for that problem, or both. **Code should be well commented for grading**.

Grading and Evaluation: The credit for each problem in this set is given in parentheses at the stated question (sub-question fraction of points is also given at the sub-questions). Partial credit will be given for both paper and Matlab questions. For Matlab questions, if the code does not run, then limited or no credit will be given.