EECS 504 Foundations of Computer Vision: PS3

Term: Winter 2026
Instructor: Jason J. Corso, University of Michigan
Due Date: 3/20 23:59 Eastern Time

Starter code: The starter code can be found HERE We recommend editing and running your code in Google Colab, although you are welcome to use your local machine instead.

Submission Process:

1. **To Gradescope**:
   (a) a `pdf` file as your write-up, including your answers to all the questions. You don't need to include the result images from the code in the writeup. *Please do not handwrite*. The write-up must be electronic. You can use Word, Google Docs, LaTeX(try overleaf!), or any similar application.
   (b) Convert the .ipynb notebook to a pdf using the code in the last cell of notebook. Make sure it contains all the visualizations (i.e, the result images). Merge the notebook pdf to your write-up and upload your file to Gradescope as a single submission. While assigning pages for your Gradescope assign pages for the code as well.
2. **To Canvas**: The .ipynb file containing all the visualizations should be submitted to Canvas. Before submitting, please make sure to rename the file to EECS504_PS3_< your_uniquename>_<your_umid>.ipynb.

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 python questions.

Constraints: This assignment may be discussed with other students in the course but must be written independently. Programming assignments should be written in python. Over-the-shoulder 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.

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 the function `potts`. Then run the code to compare the computed energy with the true energy (given in the colab notebook).

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

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

   You will use the Potts energy of the image as $|\partial|$. Your function will be tested on the noisy image `f` and its estimation `f_hat`. Run the code and perform the sanity checks.

3. (5) In order to denoise f, we hope to find an $\hat{f}$ such that E_{PC} is minimized. Is the image `f_hat` provided in the colab notebook a minimizer of E_{PC}? Explain (one or two sentences max).

Problem 2 (50): 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.

We provide code for computing the superpixel maps with the SLIC superpixel method, and for obtaining the max flow with the Ford-Fulkerson algorithm. 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.

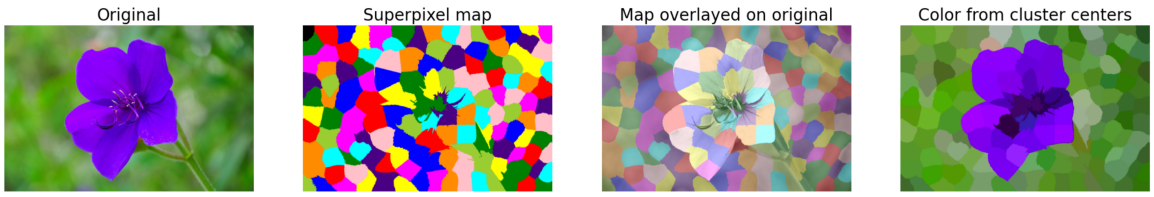

Figure 1: Flower superpixel representation

1.(8) Color Histogram Features. Implement the missing body of code in `color_histogram`, which creates a color histogram feature vector. Follow the comments in the code for the details.

2.(15) Superpixel adjacency

(a) (7) Implement the missing body of code in `adjacencyMatrix`, 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.

(b) (5) Implement the function `average_node_degree` to compute the average node degree. Then run the code and perform the error checks in the notebook.

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

3.(17) Graph-cuts

(a) (8) Implement the function `graph_cut`, 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.

(b) (5) 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.(10) Graph-cuts Study

(a) (3) Why are both boots being segmented in the `porch` image if we selected a superpixel corresponding to only one of the boots.

(b) (3) Why are multiple signs in the `veggies` image being segmented as the foreground if we selected the superpixel corresponding to only one of the signs.

(c) (4) Explain a solution that could potentially avoid the problem stated in 4.b and be able to segment only the sign selected in the `veggies` image.

Problem 3 (35): 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:

$$S_1 \neq S_2 \text{ and } weight(E(v1, v2)) < MInt(S_1, S_2).$$

Figure 2: Flower segmentation using graph-cut

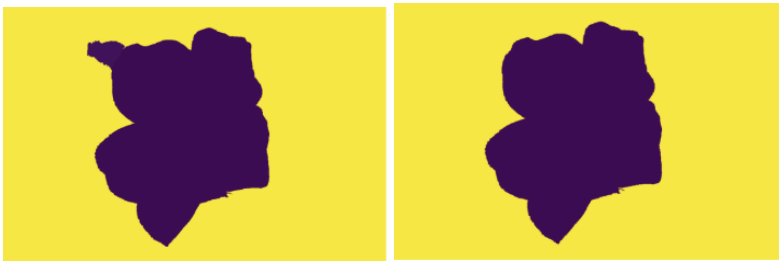

Figure 3: (Left) Segmentation using MSF, (Right) Segmentation using MSF after filtering.

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. (10) Implement the function `compute_MSF_edges` which will turn your adjacency matrix into a mx3 matrix where m is the number of edges. For each edge you will record the first node, the second node and the weight between the nodes. Note that the edge weight calculation is a little bit different from the graph-cut method. Follow the comments in the code for the details.

2. (15) Implement your minimum random forest segmentation algorithm. Fill in the missing parts in `MSF`. Follow the comments in the code for the details.

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

4. (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 in `filter_segments`. The function should 1) iterate through the edges in the original image graph, in ascending order, and 2) for each edge, if the edge connects two distinct segments, AND if at least one of those segments contains less than min_size pixels in it, merge the segments.