EECS 442 Computer Vision: HW1

Term: Fall 2018
Instructor: Jason J. Corso, EECS, University of Michigan
Due Date: 10/5 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 Python using the open-source library ETA. Over-the-shoulder Python 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: Test mathematical basics and deepen the understanding of images as functions.

Data: You need to download `hw1.zip` to complete this assignment. All paths in the assignment assume the data is off the local directory.

Problem 1 (16): Fitting Curves with Linear Least Squares
We discussed the least-square formulation for a slope-intercept model of a line during lecture.

$$E(m,b) = \sum_{i=1}^{n} [y_i - mx_i - b]^2 \tag{1}$$

Now, assume you obtain a few sample points from some experiments, i.e. `hw1_p1.npy` in the hw1p1 folder.

(a) (7) Say you want to fit the data to a cubic curve. Write down the least squares formulation (in a form similar to Eq.1) as well as the matrix formulation of it. Explain why this problem could be solved by using Linear Least Squares despite the fact that the curve is non-linear.

(b) (6) Fill in the missing part of the function `_fit_curve()` in file `hw1p1/hw1p1.py` to compute the coefficients of cubic curve using the LLS formulation you wrote in part(a). Report resulting coefficient values. Also, write the code for the function `_viz_curve()` to plot the original data and, on top of the data, the curve that fits the data. Include your code verbatim in your pdf submission. Increase the order of the curves to values such as 4, 5 and 6, and report their resulting plots as well.

(c) (3) Does the curve with higher orders fit the data better? Explain what you observe.

Problem 2 (15): Potts model

(a) (4) Convolution is a simple mathematical operation which is fundamental to many common image processing operations like blurring, edge-detection etc. Discrete convolution in 1-D is given by:

$$y[n] = x[n] * h[n] = \sum_{-\infty}^{\infty} x[n] \cdot h[n-m] \tag{2}$$

Think how would you perform convolution on the boundary values. Perform a 2-D convolution on the following image **A** with kernel **B**:

$$\mathbf{A} = \begin{bmatrix} 4 & 8 \\ 2 & 3 \\ 1 & 6 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 1 & 2 \\ 2 & 0 \end{bmatrix} \tag{3}$$

(b) (7) Recall the Potts model we discussed in the lecture. Based on that model, the energy of an image I is

$$E(I) = \beta \sum_{s=1}^{n-1} \sum_{t=1}^{n} (\mathbf{1}(I(s,t) \neq I(s+1,t))) + \beta \sum_{s=1}^{n} \sum_{t=1}^{n-1} (\mathbf{1}(I(s,t) \neq I(s,t+1))) \tag{4}$$

Implement a function to compute the energy of the image `data/Michigan_letter.jpg` using this model and submit the value of the resultant energy. Use $\beta = 1$ in your implementation. You need to fill in the missing parts of the functions `_create_x_derivative_kernel()`, `_create_y_derivative_kernel()` and `_convolve()` in `modules/convolution.py` and `_calculate_potts_energy()` in `modules/_calculate_potts_energy.py`.

You can run the the pipeline using the following command:
`eta build -r requests/potts_energy_request.json`

(c) (4) Perform a 2-D convolution on the given image using the `_create_gaussian_kernel()` in `modules/convolution.py` and calculate its Potts energy. You need to use the pipeline provided in `pipelines/potts_energy_gaussian.json` to do this. Explain the difference in energies for the original and final image. (Hint: Think in the direction as to what the given kernel is doing to the image).

You can run the the pipeline using the following command:
`eta build -r requests/potts_energy_gaussian_request.json`

Problem 3 (14): Transformation
Imagine you are in Lego world. You are given 3 blocks - red, blue and green. The blocks are arranged as shown in Fig 1. You have to rearrange and stack the lego blocks to form structure shown in Fig 2. The images are 20x20 and the top-left corner is (0,0). The edge-length (in number of pixels) of the blocks are provided in the figure.

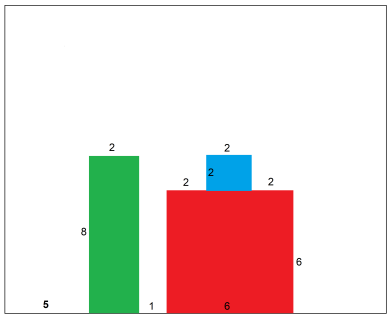

Figure 1: Current state

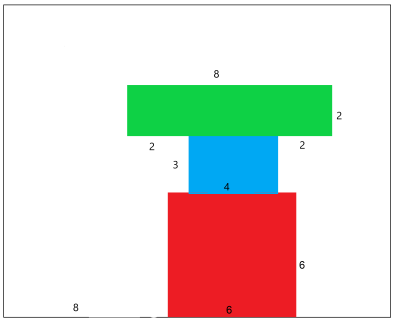

Figure 2: Final state

(a) (10) Explain your approach in detail. Provide the transformation matrix (matrices) you have created to achieve Fig 2.

(b) (4) State which transformation(s) is (are) used in part (a). Do these transformation matrix(matrices) preserve orientation, angle and parallelism of line segments?

Problem 4 (25): Edge Detection
An edge is a place where image intensity changes abruptly. Edge detection is the method of finding these boundaries of objects within images. Through this problem you are going to implement a well-known algorithm called Canny Edge Detector. You need to run the detector on `data/Canny_input.jpg` by using `pipelines/edge_detection.json`.

You can run the the pipeline using the following command:
`eta build -r requests/edge_detection_request.json`

(a) (4) The first step in the Canny edge detector is to get the gradient images. One of the most popular ways to compute the gradient images is by using vertical and horizontal sobel operators. Fill out the missing part of the functions `_create_sobel_horizontal_kernel()` and `_create_sobel_vertical_kernel()` in file `modules/convolution.py` to create the kernels, and function `_create_intensity_orientation_matrices()` in file `modules/canny_edge_detector.py` to compute the intensity and orientation matrices of the gradients.
Report the gradient intensity image in your writeup. Observe the edges extracted from this. Does it give a sharp edge? Explain your observation.

(b) (7) Implement an algorithm to get a sharp edge response from the output of part (a). This process is called Non-Maximum Suppression. Fill in the missing part of the function `_non_maximum_suppression()` in the file `modules/canny_edge_detector.py` and report the output image in your writeup.

(c) (10) Observe the output from part (b). The edge-pixels can be considered as strong, weak, or suppressed based on the intensity of their gradients. You should then perform Edge Tracking by connecting each weak-edged pixel to any neighboring strong-edged pixel (if any) to get the complete edge. Fill in the missing part of the code for the functions `_double_thresholding()` and `_hysteresis()` in the file `modules/canny_edge_detector.py` to implement double thresholding and perform edge tracking by hystersis, respectively. Include the output image in your report.

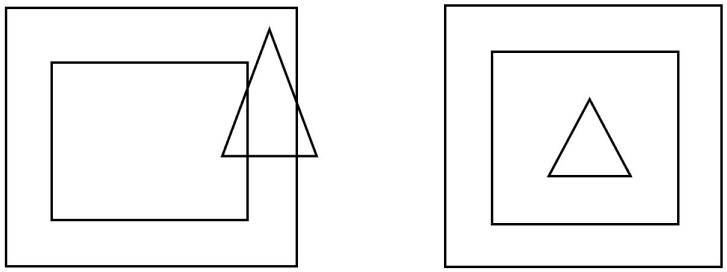

Figure 3: Canny_input.jpg

(d) (4) Is the Canny Edge Detector rotation invariant? Provide a mathematical proof or counter argument.

(e) (**Extra Credits: 10**) Observe the line-segments formed by intersecting edges in the output of part (c). Write a function to get the coordinates of these line-segments. Be creative! For this problem, you can use `pipelines/line_segments.json` and modify `modules/find_line_segments.py`. You can run the the pipeline using the following command:

```
eta build -r requests/line_segments_request.json
```

You should provide the output as a .json file called `output_points.json` (You can use `eta.core.serial.write_json()` to write into .json format). Your `output_points.json` should be in the following format:

```
{
    "Line_segments": {
                        "No": 1
                        "coordinates": [(x1, y1), (x1',y1')}
                      }
                      {
                        "No": 2
                        "coordinates": [(x2, y2), (x2',y2')}
                      }
                      .
                      .
                      .
}
```

Submission Process: Submit a single pdf with your answers to these problems, including all plots and discussion. Submit the pdf to Gradescope.

For coding assignments, include your code verbatim in your writeup. Pack the original program files into one zip file 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 python questions. For python questions, if the code does not run, then limited or no credit will be given.