Homework 4

ECE 253
Digital Image Processing

November 27, 2019

Make sure you follow these instructions carefully during submission :

- Homework 4 is due by 11:59 PM, December 12, 2019.
- You should avoid using loops in your MATLAB/Python code unless you are explicitly permitted to do so.
- Submit your homework electronically by following the two steps listed below:

  1. Upload a pdf file with your write-up on Gradescope. This should include your answers to each question and **relevant code snippet**. Make sure the report mentions your full name and PID. Finally, carefully read and include the following sentences at the top of your report:

     Academic Integrity Policy: Integrity of scholarship is essential for an academic community. The University expects that both faculty and students will honor this principle and in so doing protect the validity of University intellectual work. For students, this means that all academic work will be done by the individual to whom it is assigned, without unauthorized aid of any kind.

     By including this in my report, I agree to abide by the Academic Integrity Policy mentioned above.

  2. Upload a zip file with all your scripts and files on Gradescope. Name this file: **ECE_253_hw4_lastname_studentid.zip**. This should include all files necessary to run your code out of the box.

Problem 1. Hough Transform (10 points)

(The first two parts of this problem is borrowed from Professor Belongie's past CSE 166 class.)

(i) Implement the Hough Transform (HT) using the (ρ, θ) parameterization as described in GW Third Edition p. 733-738 (see 'HoughTransform.pdf' provided in the data folder). Use accumulator cells with a resolution of 1 degree in θ and 1 pixel in ρ.

(ii) Produce a simple 11×11 test image made up of zeros with 5 ones in it, arranged like the 5 points in GW Third Edition Figure 10.33(a). Compute and display its HT; the result should look like GW Third Edition Figure 10.33(b). Threshold the HT by looking for any (ρ, θ) cells that contains more than 2 votes then plot the corresponding lines in (x,y)-space on top of the original image.

(iii) Load in the image 'lane.png'. Compute and display its edges using the Sobel operator with default threshold settings, i.e.,

```
E = edge(I,'sobel')
```

Now compute and display the HT of the binary edge image E. As before, threshold the HT and plot the corresponding lines atop the original image; this time, use a threshold of 75% maximum accumulator count over the entire HT, i.e. `0.75*max(HT(:))`.

(iv) We would like to only show line detections in the driver's lane and ignore any other line detections such as the lines resulting from the neighboring lane closest to the bus, light pole, and sidewalks. Using the thresholded HT from the 'lanes.png' image in the previous part, show only the lines corresponding to the line detections from the driver's lane by thresholding the HT again using a specified range of θ this time. What are the approximate θ values for the two lines in the driver's lane?

Things to turn in:

- HT images should have colorbars next to them
- Line overlays should be clearly visible (adjust line width if needed)
- HT image axes should be properly labeled with name and values (see Figure 10.33(b) for example)
- 3 images from 2(ii): original image, HT, original image with lines
- 4 images from 2(iii): original image, binary edge image, HT, original image with lines
- 1 image from 2(iv): original image with lines
- θ values from 2(iv)
- Code for 2(i), 2(ii), 2(iii), 2(iv)

Problem 2. K-Means Segmentation (15 points)

In this problem, we shall implement a K-Means based segmentation algorithm from scratch. To do this, you are required to implement the following three functions -

- *features = createDataset(im)* : This function takes in an RGB image as input, and returns a dataset of features which are to be clustered. The output *features* is an $N \times M$ matrix where N is the number of pixels in the image *im*, and $M = 3$ (to store the RGB value of each pixel). You may not use a loop for this part.

- *[idx, centers] = kMeansCluster(features, centers)* : This function is intended to perform K-Means based clustering on the dataset *features* (of size $N \times M$). Each row in *features* represents a data point, and each column represents a feature. *centers* is a $k \times M$ matrix, where each row is the initial value of a cluster center. The output *idx* is an $N \times 1$ vector that stores the final cluster membership ($\in 1, 2, \cdots, k$) of each data point. The output *centers* are the final cluster centers after K-Means. Note that you may need to set a maximum iteration count to exit K-Means in case the algorithm fails to converge. You may use loops in this function.

  Functions you may find useful : *pdist2(), isequal()*.

- *im_seg = mapValues(im, idx)* : This function takes in the cluster membership vector *idx* ($N \times 1$), and returns the segmented image *im_seg* as the output. Each pixel in the segmented image must have the RGB value of the cluster center to which it belongs. You may use loops for this part.

  Functions that you may find useful : *mean(), cat()*.

With the above functions set up, perform image segmentation on the image *white-tower.png*, with the number of clusters, *nclusters = 7*. To maintain uniformity in the output image, please initialize clusters centers for K-Means as follows -

```
rng(5);
id = randi(size(features, 1), 1, nclusters);
centers = features(id, :);
```

Things to turn in:

- The input image, and the image after segmentation.
- The final cluster centers that you obtain after K-Means.
- All your code for this problem (in the Appendix).

Problem 3. Semantic Segmentation (20 points)

In this problem, we train fully convolutional networks [1] to do semantic segmentation. Most of the codes are provided. However, due to some incidents, part of the network is missing. You're required to complete and train the network using CityScape [2] dataset. Also, answer the following questions. Please check the **README.md** for training and testing commands.

1. Please complete the FCN network, the $fcn8s$ in **ptsemseg/models/fcn.py**. And briefly describe the model structure.
2. Do we use weights from pre-trained model? Or do we train the model from scratch?
3. Please train the network with CityScape dataset. Visualize the training curves with Tensorboard. Screenshot the training and validation curve. (config file: **configs/fcn8s_cityscapes.yml**)
4. What are the metrics used by the original paper? Do testing (**test.py**) on the dataset. Which class works better? Which class works worse?
5. Can you visualize your results, by plotting out the labels and prediction of the images? (**HINT:** check the unit test in **ptsemseg/loader/cityscapes_loader.py**)
6. Please take a photo of the city street, and show the output image from the model. Does the output image look reasonable?
7. Based on your analysis of the model and training, how can you get better results for prediction? (Give 2 possible options. Change the parameters? Change the network architecture? Or other thoughts? You can checkout the fcn paper [1] and the followup works.)

To be noted:

- Upload the zip file to the server. Follow the steps in **README.md** to install environments and requirements
- Training time is around 5 hours.
- When you're running the server, save the url so that you can access the tab later once you close it.
- You are encouraged to read the fcn paper [1] to know the details.

References

[1] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015, pp. 3431–3440, iSSN: 1063-6919.

[2] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The Cityscapes Dataset for Semantic Urban Scene Understanding," *arXiv:1604.01685 [cs]*, Apr. 2016, arXiv: 1604.01685. [Online]. Available: http://arxiv.org/abs/1604.01685