Homework 3

ECE 253
Digital Image Processing

November 12, 2019

Make sure you follow these instructions carefully during submission :

- Homework 3 is due by 11:59 PM, November 22, 2019.
- All problems are to be solved using MATLAB/Python unless mentioned otherwise.
- 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_hw3_lastname_studentid.zip**. This should include all files necessary to run your code out of the box.

Problem 1. Canny Edge Detection (15 points)

In this problem, you are required to write a function that performs *Canny Edge Detection.* The function has the following specifications:

- It takes in two inputs: a grayscale image, and a threshold t_e.
- It returns the edge image.
- You are allowed the use of loops.

A brief description of the algorithm is given below. Make sure your function reproduces the each step as given.

1. **Smoothing**: It is inevitable that all images taken from a camera will contain some amount of noise. To prevent noise from being mistaken for edges, noise must be reduced. Therefore the image is first smoothed by applying a Gaussian filter. A Gaussian kernel with standard deviation $\sigma = 1.4$ (shown below) is to be used.

$$k = \frac{1}{159} \cdot \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

2. **Finding Gradients** The next step is to find the horizontal and vertical gradients of the smoothed image using the *Sobel* operators. The gradient images in the x and y-direction, G_x and G_y are found by applying the kernels k_x and k_y given below:

$$k_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, k_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

The corresponding gradient magnitude image is computed using:

$$|G| = \sqrt{G_x^2 + G_y^2},$$

and the edge direction image is calculated as follows:

$$G_\theta = arctan(\frac{G_y}{G_x}).$$

3. **Non-maximum Suppression (NMS)**: The purpose of this step is to convert the thick edges in the gradient magnitude image to "sharp" edges. This is done by preserving all local maxima in the gradient image, and deleting everything else. This is carried out by recursively performing the following steps for each pixel in the gradient image:

    - Round the gradient direction θ to nearest 45°, corresponding to the use of an 8-connected neighbourhood.

   - Compare the edge strength of the current pixel with the edge strength of the pixel in the positive and negative gradient direction i.e. if the gradient direction is north ($\theta = 90°$), then compare with the pixels to the north and south.
   - If the edge strength of the current pixel is largest; preserve the value of the edge strength. If not, suppress (remove) the value.

4. **Thresholding**: The edge-pixels remaining after the NMS step are (still) marked with their strength. Many of these will probably be true edges in the image, but some may be caused by noise or color variations. The simplest way to remove these would be to use a threshold, so that only edges stronger that a certain value would be preserved. Use the input t_e to perform thresholding on the non-maximum suppressed magnitude image.

Evaluate your canny edge detection function on *geisel.jpg* for a suitable value of t_e that retains the structural edges, and removes the noisy ones.

Things to turn in:

- The original gradient magnitude image, the image after NMS, and the final edge image after thresholding.
- The value for t_e that you used to produce the final edge image.
- Code for the function.

Problem 2. Butterworth Notch Reject Filtering in Frequency Domain (15 points)

This problem will follow Figure 4.64 in section 4.10.2 Notch Filters of Gonzalez & Woods 3rd Edition.

(i) Read in the image *Car.tif*, pad the image to 512x512 (using zero padding on all four sides of the image), and display the 2D-FFT log magnitude (after moving the DC component to the center with fftshift):
MATLAB:

```
imagesc(-256:255,-256:255,log(abs(imFFT))); colorbar;
xlabel('u'); ylabel('v');
```

Python: Please refer to this link and check *Fourier Transform in Numpy* section.

You should see symmetric "impulse-like" bursts which we suspect is the cause of the Moire Pattern (the dot pattern from the newspaper image). We would like to filter out this pattern in the frequency domain using a Butterworth Notch Reject Filter given by:

$$H_{NR}(u,v) = \prod_{k=1}^{K} \left[\frac{1}{1 + [D_0/D_k(u,v)]^{2n}} \right] \left[\frac{1}{1 + [D_0/D_{-k}(u,v)]^{2n}} \right] \quad (1)$$

where

$$D_k(u,v) = \left[(u-u_k)^2 + (v-v_k)^2\right]^{1/2} \tag{2}$$

$$D_{-k}(u,v) = \left[(u+u_k)^2 + (v+v_k)^2\right]^{1/2} \tag{3}$$

Here, we have slightly modified the definition from the textbook slightly in that we removed $M/2$ and $N/2$ from the equation so that the center of the DFT image is 0 rather than $(M/2, N/2)$.

The parameter **K** is the number of "impulse-like" bursts you would like to remove (excluding their symmetric counterparts since the H_{NR} equation already includes it), e.g. 4 for this image rather than 8. The parameters **n** and $\mathbf{D_0}$ are scalar values that you may choose to control the transition and radius of the notch filters. The parameter $\mathbf{u_k}$ and $\mathbf{v_k}$ is the location of the k^{th} "impulse-like" bursts in the 2D DFT magnitude image. **u** and **v** are all possible (u,v) coordinate pairs that you want to generate H_{NR} for:

```
MATLAB:
[u,v] = meshgrid(-256:255);

Python:
x_axis = np.linspace(-256,255,512)
y_axis = np.linspace(-256,255,512)
[u,v] = np.meshgrid(x_axis,y_axis)
```

(ii) Repeat for *Street.png*, except for K=2 and remove the burst along the $u = 0$ axis and the $v = 0$ axis.

Things to turn in:

- All images should have colorbars next to them
- All DFT magnitude images should have the DC frequencies in the center of the image
- 4 images from 2(i): 1 unpadded original image, the corresponding 2D DFT log-magnitude, the butterworth Notch Reject Filter in frequency domain $H_{NR}(u,v)$, the final filtered image
- 10 parameters for 2(i): n, D_0, u_1, v_1, ..., u_4, v_4
- 4 images from 2(ii): 1 unpadded original image, the corresponding 2D DFT log-magnitude, the butterworth Notch Reject Filter in frequency domain $H_{NR}(u,v)$, the final filtered image
- 6 parameters for 2(ii): n, D_0, u_1, v_1, u_2, v_2
- Code for 2(i), 2(ii)

Problem 3. PyTorch tutorial and questions (5 points)

After seeing some awe-inspiring machine learning results, do you want to try some yourselves? Let's start with some basic practice of machine learning framework: PyTorch. Please follow the tutorial for cifar10 classifier (cifar10 tutorial) and answer the following questions.

(i) Login to the server for CPU/GPU resources. (0 point)

- https://datahub.ucsd.edu/ (Jupyterhub)
- Select environment (with or without GPU. You don't need gpus in this homework.)
- Launch environment.

(ii) How many images and batches are used to train the network?

(iii) Do we normalize the images? What do we do in the example?

(iv) The losses are dropping! Can you plot out the training loss?

(v) Now the network is done training. Can you check some successful cases and some failure cases (show some images classified by the network)?

(vi) Can you visualize the output of the 1st layer of CNN using one image from the training set?

Congratulations! Pytorch can be more powerful than that! Feel free to go through some tutorials and get cool models!

References:

- deep learning 60min blitz
- pytorch with examples