Homework 2

ECE 253
Digital Image Processing

October 24, 2023

Make sure you follow these instructions carefully during submission:

- Homework 2 is due by 11:59 PM, November 19, 2023.
- All problems are to be solved using Python.
- You should avoid using loops in your 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. You **must** use the Gradescope page selection tool to indicate the relevant pages of your report for each problem; failure to do so may result in no points. 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.

   Reports missing any of the above elements may receive no credit.
2. Upload a zip file with all of your scripts and files on Gradescope. Name this file: **ECE_253_hw2_lastname_studentid.zip**. This should include all files necessary to run your code out of the box.

Problem 1. Adaptive Histogram Equalization (10 points)

It is often found in image processing and related fields that real world data is unsuitable for direct use. This warrants the inclusion of pre-processing steps before any other operations are performed. An example of this is *histogram equalization* (HE) and its extension *adaptive histogram equalization* (AHE).

The goal of this problem is to implement a function for AHE as described in Chapter 1 of *Adaptive Histogram Equalization - A Parallel Implementation*[1]. The function has the following specifications:

(i) The desired function $AHE()$ takes two inputs: the image im and the contextual region size win_size.

(ii) Using the pseudocode in Algorithm 1 as a reference, compute the enhanced image after AHE.

(iii) You may use loops if necessary. You should not make use of any inbuilt Python functions for AHE or HE.

(iv) The function returns one output: the enhanced image after AHE.

Algorithm 1 Pseudocode for Adaptive Histogram Equalization of an Image

*Pre-requirement : Pad the image im on all 4 sides by mirroring intensity values so that the contextual region for edge pixels remains valid. This can be done in Python using *numpy.pad*() with the '*symmetric*' argument.

```
function AHE(im, win_size)
    for each pixel (x, y) in image im do
        rank ← 0
        contextual_region* ← (win_size × win_size) window centered around (x, y)
        for each (i, j) in contextual_region do
            if im(x, y) > im(i, j) then
                rank ← rank + 1
        output(x, y) ← rank × 255/(win_size × win_size)
    return output
```

Evaluate your function on the image *beach.png* for $win\_size = 33,\ 65,$ and 129. In your report, include the original image, the 3 images after AHE, and the image after simple HE (you may use Python inbuilt function for this final HE image only). Make sure to resize all images to ensure they do not take up too much space. Additionally, include your answers (no more than three sentences each) to the following questions:

- How does the original image qualitatively compare to the images after AHE and HE respectively?
- Which strategy (AHE or HE) works best for *beach.png* and why? Is this true for any image in general?

[1] You can find this article here.

Problem 2. Lloyd-Max Quantizer (10 points)

For this part of the homework, we study the *Lloyd-Max* quantizer. Before you get started, carefully study pages 602-603 of the book[2]. Then, proceed to implement the following:

(i) Write a function that takes as inputs a *greyscale* 8-bit (uint8) image, a scalar $s \in [1, 7]$ and performs uniform quantization over the entire range [0, 255] so that the output is quantized to an **s-bit** image. You may use loops for this part if necessary.

(ii) The Python script *lloyd_python.py* is found in the shared homework files. It performs Lloyd Max quantization (optimal quantization in the squared error sense). You can use this function as:

```
partition, codebook = lloyds(training_set, initcodebook)
```

The input training set is a vector of training data (which is used as an empirical measure of the probability mass function), so you will have to flatten your image (in other words, reshape the image from [M,N] to [M*N, 1].

For the images *lena512.tif* and *diver.tif*, calculate the MSE values for s $\in$ [1,7] using both your uniform quantizer and the Lloyd-Max quantizer (you may use loops for the Lloyd-Max quantizer as well). Plot the results (MSE versus number of bits). Show one plot for *lena512.tif* (with both uniform and Lloyd-Max quantization) and another plot for *diver.tif*. Compare the results for the different quantizers/images and explain them. That is, why does one quantizer outperform the other, and why is the performance gap larger for one image than for the other?

Do not make use of in-built Python functions to calculate the MSE.

(iii) Now use global histogram equalization on *lena512.tif* and on *diver.tif* to generate two new images.

Repeat part (ii) for these two new images. Compare them with the previous set of plots. What has happened to the gap in MSE between the two quantization approaches and why?

(iv) Why is the MSE of the 7-bit *Lloyd-Max* quantizer zero or near zero for the equalized images? One might have thought that equalization is not to the advantage of the *Lloyd-Max* quantizer, because equalizing the histogram should be flattening the distribution, making it more uniform, which should be to the advantage of the uniform quantizer. Explain this phenomenon.

Problem 3. Quantization with Dithering (5 points)

In this problem you will explore the impact of Dithering for image quantization. Specifically, you will be implementing uniform quantization and the algorithm you will be using for dithering is called *Floyd-Steinberg*.

[2]Included in the data folder.

(i) You will first implement a function that performs uniform quantization with different number of color bands. The function must quantize the image to 10 levels. You can handle each color dimension independently.

(ii) Then you will implement the *Floyd-Steinberg* shown in algorithm (2) dithering algorithm to study the effects of dithering in each quantized image. The *find_closest_palette_color()* function simply finds the nearest color value according to 10-level quantization. You can handle each color dimension independently.

Complete the above two steps on *geisel.jpg* and answer the questions below. You must include the result of both parts in your report.

1. What differences do you see between the two images?
2. Can you explain what caused these differences?

Algorithm 2 Floyd-Steinberg Dithering

```
function FSD(im)
    for each y from top to bottom do
        for each x from left to right do
            old_pixel ← im[x][y]
            new_pixel ← find_closest_palette_color()
            im[x][y] ← new_pixel
            quant_error ← old_pixel − new_pixel
            im[x + 1][y] ← im[x + 1][y ] + quant_error × 7 / 16
            im[x - 1][y + 1] ← im[x - 1][y + 1] + quant_error × 3 / 16
            im[x ][y + 1] ← im[x ][y + 1] + quant_error × 5 / 16
            im[x + 1][y + 1] ← im[x + 1][y + 1] + quant_error × 1 / 16
    return im
```

Problem 4. 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). Please refer to the links available here and here.

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:

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

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:

```
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)