CSE 166: Image Processing, Spring 2022 – Assignment 2

Instructor: Ben Ochoa

- Due On: **Wednesday, April 13, 2022, 11:59 PM (Pacific Time)**.

Instructions

Please answer the questions below using Python in the attached Jupyter notebook and follow the guidelines below:

- This assignment must be completed **individually**. For more details, please follow the Academic Integrity Policy and Collaboration Policy on Canvas.
- This assignment contains both math and programming problems.
- All the solutions must be written in this Jupyter notebook.
- After finishing the assignment in the notebook, please export the notebook as a PDF and submit both the notebook and the PDF (i.e. the `.ipynb` and the `.pdf` files) on Gradescope. Please assign the pages of your PDF submission to the corresponding problems.
- You may use basic algebra packages (e.g. `NumPy`, `SciPy`, etc) but you are not allowed to use the packages that directly solve the problems. Feel free to ask the instructor and the teaching assistants if you are unsure about the packages to use.
- It is highly recommended that you begin working on this assignment early.

Late Policy: Assignments submitted late will receive a 15% grade reduction for each 12 hours late (i.e., 30% per day). Assignments will not be accepted 72 hours after the due date. If you require an extension (for personal reasons only) to a due date, you must request one as far in advance as possible. Extensions requested close to or after the due date will only be granted for clear emergencies or clearly unforeseeable circumstances.

Problem 1: Textbook problems (10 points)

a) Problem 3.5 (2 points)

your answer here

b) Problem 3.7 (2 points)

your answer here

c) Problem 3.11a (1 point)

your answer here

d) Problem 3.12a (1 point)

your answer here

e) Problem 3.24 (1 point)

your answer here

f) Problem 3.29 (2 points)

your answer here

g) Problem 3.44 (1 point)

your answer here

Problem 2: Programming: Intensity transformations and

spatial filtering (30 points)

Part 1: Piecewise linear transformation (5 points)

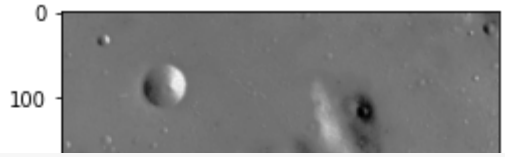
- Complete the function **piecewise_linear_transform** which computes an intensity transformed image for a given image and two additional parameters. The parameters are comprised of a set of input intensity values and a corresponding set of output intensity values, where the first intensity value is 0 and the last intensity value is 255 in both sets. The resulting intensity values in the output image are calculated from a piecewise linear transformation determined from the parameters. See Figure 3.10(a) in the textbook for an example of a piecewise linear transformation function corresponding to a set of input and output intensity values.
- Call the function **piecewise_linear_transform** for the moon image with input intensities $\{0, 100, 130, 255\}$ and output intensities $\{0, 70, 200, 255\}$ to calculate the intensity transformed image. Display the resulting output.
- Briefly discuss the qualitative differences between the input and output images.

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
import math
```

```
# Read and display image
img = data.moon()

plt.imshow(img, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7ff9ba37f850>
```




```
# function that transforms intensities of the image using piece-wise linear transfo
```

```
def piecewise_linear_transform(img, ip_intensities, op_intensities):
  """
  img: input image (H,W)
  ip_intensities: input intensities (N,)
  op_intensities: output intensities (N,)

  returns:
  tr_img: resulting image after piecewise linear transform (H,W)
  """
  # your code here

  return tr_img
```

```
"""
Call piecewise_linear_transform function for the moon image with the given input
and output intensities. Display the transformed image.
"""
ip_intensities = np.array([0, 100, 130, 255])
op_intensities = np.array([0, 70, 220, 255])
tr_img = piecewise_linear_transform(img, ip_intensities, op_intensities)

fig, ax = plt.subplots(1,2, figsize=(10,10))
ax[0].imshow(img, cmap='gray')
ax[1].imshow(tr_img, cmap='gray')
```

Briefly discuss the qualitative differences between the input and output images.

your answer here

Part 2: Histogram equalization (5 points)

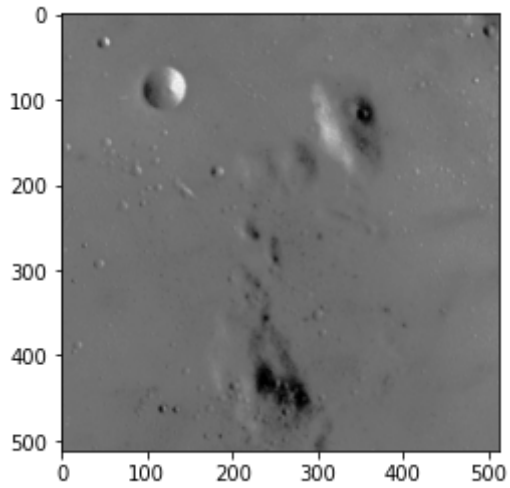
- Complete the function **histogram_equalization** which calculates a histogram equalized image corresponding to the given image. The function input is a grayscale image and the function output is the histogram equalized image corresponding to the input image.
- Call the function **histogram_equalization** for the moon image and display the resulting output.
- Plot the histogram of the intensities of the input image and the histogram-equalized image side-by-side

```
# Read and display the image along with its histogram of intensities
img = data.moon()

plt.imshow(img, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f2053008f90>
```

<matplotlib.image.AxesImage at 0x7f2053008190>




```
# function that transforms the input image into histogram-equalized image
def histogram_equalization(img):
  """
  img: input image

  returns:
  heq_img: histogram-equalized image
  """
  # your code here

  return heq_img
```

```
"""
Call histogram_equalization for the moon image. Display the output
"""
heq_img = histogram_equalization(img)

fig, ax = plt.subplots(1,2, figsize=(10,10))
ax[0].imshow(img, cmap='gray')
ax[1].imshow(heq_img, cmap='gray')
```

```
"""
plot the histogram of the intensities of the input image and the histogram-equalize
image side-by-side
"""
# your code here
```

Part 3: Sharpening using the second derivative (10 points)

1. Complete the function **sharpen_image** that calculates a sharpened image corresponding to a given image. The function input is a grayscale image and the function output is the sharpened image corresponding to the input image. The function must use the Laplacian filter (without

diagonal directions). Note that $g(x, y) = f(x, y) - \nabla^2 f(x, y)$, where $f(x, y)$ is the input image and $g(x, y)$ is the output sharpened image. Filtering must be implemented without using Python functions like *scipy.ndimage.convolve*.

2. Call the function **sharpen_image** for the checkerboard image. Display the input image and the sharpened image, side by side.
3. Briefly discuss your results.
4. **Optional**: sharpen the image using the Laplacian filter with diagonal directions and briefly discuss the qualitative differences between these results and those obtained using the filter without diagonal directions.

Notes:

- While performing sharpening, the output pixels can have values outside the [0,255]. You can handle this by working with np.int32 data type matrices.
- Clamp the output sharpened image such that all intensities less than zero is set to 0 and all intensities greater than 255 is set to 255.

```
# function that sharpens the given image
def sharpen_image(img):
  """
  img: input image (H,W)

  returns:
  sh_img: image after sharpening (H,W)
  """
  # your code here

  return sh_img
```

```
"""
Call the sharpen_image function with the checkerboard image.
Display the images before and after sharpening.
"""
img = data.checkerboard()
# your code here

```

Briefly discuss your results

your answer here

Part 4: Magnitude of gradient (10 points)

1. Complete the function **gradient_magnitude** that calculates the magnitude of gradient vector image corresponding to a given image. The function input is a grayscale image and the function output is the gradient magnitude image corresponding to the input image. The function must use the Sobel approximation to the derivative. Filtering must be implemented without using Python functions like *scipy.ndimage.convolve*.
2. Call the function **gradient_magnitude** with the camera image, and display the output.

Notes:

- While computing magnitude of gradient, the output pixels can have values outside the [0,255] range and/or floating-point numbers. You can handle this by working with np.float32 data type matrices.
- Scale (only) the output image such that zero is black and the maximum gradient magnitude value in the image is white.

```
# function that computes the magnitude of gradient for the given image
def gradient_magnitude(img):
  """
  img: input image (H,W)

  returns:
  mg_img: image after applying magnitude of gradient (H,W)
  """
  # your code here

  return mg_img
```

```
"""
Call the gradient_magnitude function with the camera image.
Display the resulting image.
"""
img = data.camera()
# your code here

```