CSE166_FA23_assignment_6

November 29, 2023

1 CSE 166: Image Processing, Fall 2023 – Assignment 6

- Instructor: Ben Ochoa
- Due: Wed, Dec 6, 11:59 PM

Name:

PID:

1.1 Instructions

1. All solutions to the problems below must be written in this notebook. Programming aspects of the assignment must be completed using Python (preferably 3.10).
2. This assignment must be completed **individually**. For more details, please review the Academic Integrity Policy and Collaboration Policy on Canvas.
3. The packages you need to complete the assignment will be indicated in the problem descriptions.You are not allowed to use the packages that directly solve the problems. Feel free to ask the instructor and TAs if you are unsure about the packages to use.
4. It is highly recommended that you begin working on this assignment early.
5. After finishing the assignment in the notebook, please export the notebook as a PDF and a python source file. You need to **submit 3 files: the Notebook, the PDF and the python file** (i.e. the `.ipynb`, the `.pdf` and the `.py` files) on Gradescope.

- To convert the notebook to PDF, you can choose one way below:
  - You may first export the notebook as HTML, and then print the web page as PDF
    - e.g., in Chrome: File → Save and Export Notebook as → "HTML"; or in VS-code: Open the Command Palette by pressing Ctrl+Shift+P (Windows/Linux) or Cmd+Shift+P (macOS), search for Jupyter: Export to HTML
    - Open the saved web page and right click → Print... → Choose "Destination: Save as PDF" and click "Save")
  - If you have XeTex installed on your machine, you may directly export the notebook as PDF: e.g., in Chrome, File → Save and Export Notebook as → "PDF"
  - You may use nbconvert to convert the ipynb file to pdf using the following command `jupyter nbconvert --allow-chromium-download --to webpdf filename.ipynb`

- To convert the notebook to python file, you can choose one way below:
  - You may directly export the notebook as py: e.g., in Chrome, File → Save and Export Notebook as → "Executable script"; or in VScode: Open the Command Palette and search for Jupyter: Export to Python Script
  - You may use nbconvert to convert the ipynb file to python file using the following command `jupyter nbconvert --to script filename.ipynb`

6. Please make sure the content in each cell (e.g. code, output images, printed results, etc.) are clearly visible and are not cut-out or partially cropped in your final PDF file.
7. While submitting on gradescope, please make sure to assign the relevant pages in your PDF submission for each problem.

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.

2 Problem 1: Textbook problems (20 points)

Download the following textbook problems from the Resources tab on Piazza. **Answers to different versions of these problems will not be accepted.**

These textbook problems are conceptual and/or math problems, not programming problems. You are provided with a Markdown cell to answer each problem. Do not change this cell to a Code cell or create a new Code cell. **Answers in the form of code will not be accepted.**

2.1 a) Problem 9.8 (3 points)

Your answer here:

2.2 b) Problem 9.9 (3 points)

Your answer here:

2.3 c) Problem 9.10 (2 points)

Your answer here:

2.4 d) Problem 9.11 (2 points)

Your answer here:

2.5 e) Problem 9.23 (3 points)

Your answer here:

2.6 f) Problem 10.6 (1 point)

Your answer here:

2.7 g) Problem 10.10 (2 points)

Your answer here:

2.8 h) Problem 10.26 (2 points)

Your answer here:

2.9 i) Problem 10.41 (2 points)

Your answer here:

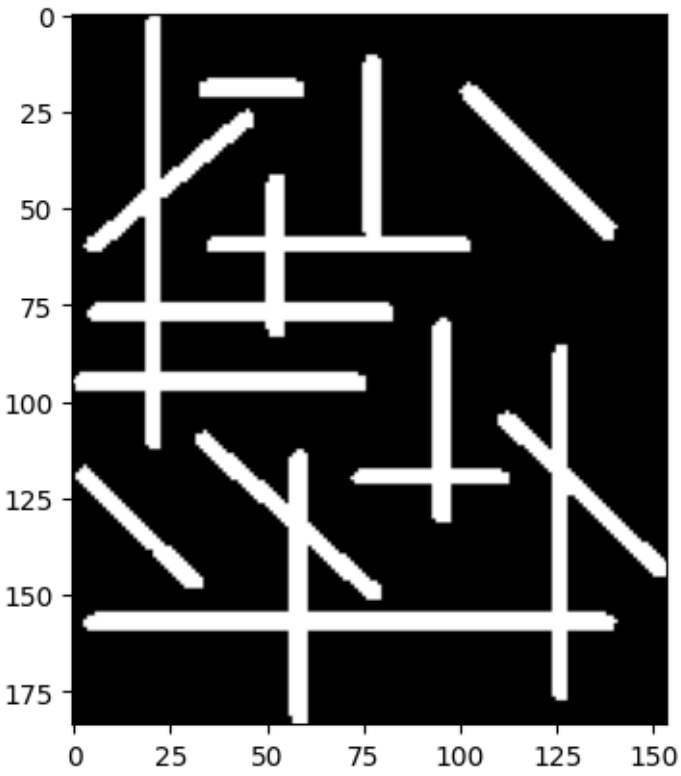
3 Problem 2: Programming (45 points)

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
from scipy.ndimage import gaussian_filter
from scipy.signal import convolve2d
import cv2
```

3.1 Part 1: Morphological Image Processing (15 points)

[ ]:
```
# Read and display image
th, img = cv2.threshold(plt.imread('lines.jpg'),128,255,cv2.THRESH_BINARY)

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



1) Complete the function `separate_lines` that separates out the vertical and horizontal lines from all lines in the image. The function takes an image as the input, and returns two images containing only vertical and horizontal lines, respectively.

Notes: - Use the opening operation with a suitable structuring element to remove undesired lines in the image. - You may use functions provided with OpenCV library. Documentation: https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.htm

[ ]:
```
# function that separates out the vertical and horizontal lines from the input image
def separate_lines(img):
  """
  img: input image (H,W)

  returns:
  img_v: image containing only vertical lines (H,W)
  img_h: image containing only horizontal lines (H,W)
  """
  # your code here
```

```
    return img_v, img_h
```

2. Call the function `separate_lines` with the lines image. Display the original and the two resulting images.

[ ]:
```
# your code here
```

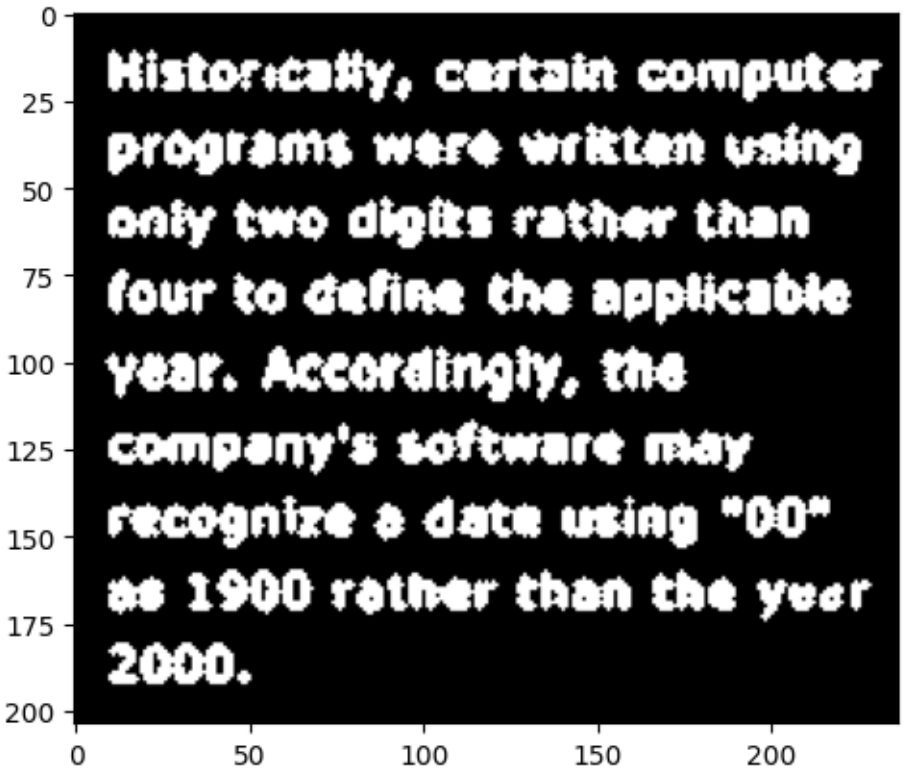
3) Improving text recognition using morphological image processing.

Complete the function `morphological_processing` that performs morphological image processing on a given image to remove noise in the text.

Note: - There is no unique way to implement morphological operations for improvements in the problem. You may use the erosion, dilation, opening, and closing functions provided with OpenCV library. Your results should visibly improve the quality of the letters.

[ ]:
```
# Read and display image
img = plt.imread('text.tif')

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



```
[ ]: # function that performs morphological image processing on the image to remove␣
      ↪noise in the text.
     def morphological_processing(img):
       """
       img: input image (N,N)

       returns:
       out: output image after morphological processing (N,N)
       """
       # your code here


       return out
```

4) Call the function `morphological_processing` with the text.tif image. Display the original and resulting images.

```
[ ]: """
     Call the function morphological_processing with the text.tif image
     Display the original and resulting images.
     """
     # your code here
```

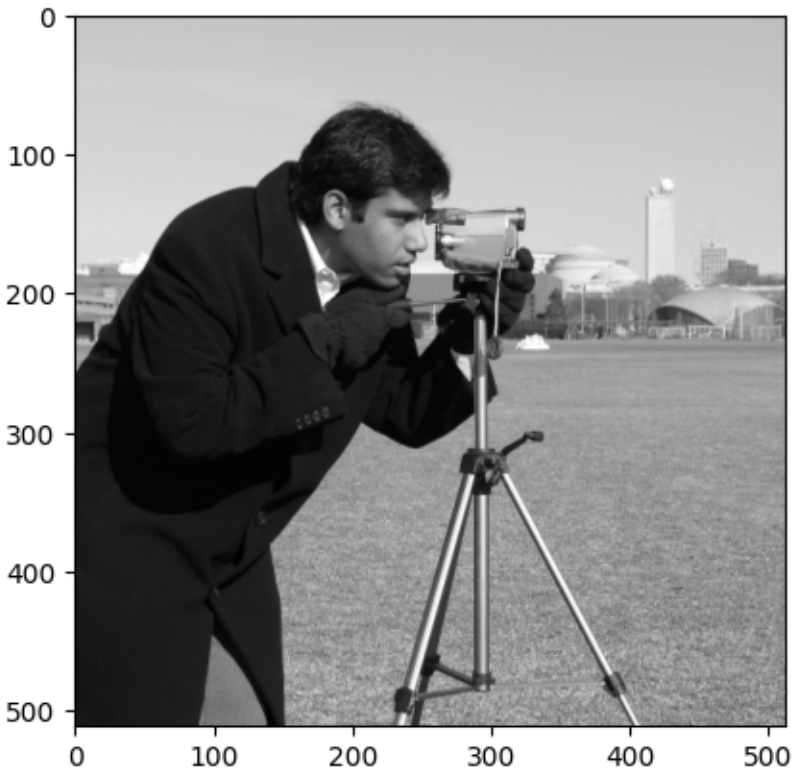
5) Briefly describe your methodology.

Your answer here:

3.2 Part 2: Edge Detection (10 points)

```
[ ]: # Read and display image
     img = data.camera()

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



1) Complete the function `edge_detection` that takes an image as input and performs the following steps:

- Normalizes the image to [0,1] range
- Applies Gaussian smoothing with standard deviation 0.5
- Computes the gradient of the image with the central difference operators
- Computes the gradient magnitude and angle images
- Applies nonmaximal suppression to the gradient magnitude image. For non-maximum suppression, discretize the angle into 8 directions (or bins), where each bin accounts for 45 degrees. For example, first bin would range from [-22.5,22.5] degrees, second bin would range from [22.5,67.5] degrees, and so on. Please refer to Lecture 14, slide 31 for a graphical description.
- Returns the gradient magnitude image, angle image, and gradient magnitude image after non-maximal suppression

Note: You may use `scipy.ndimage.gaussian_filter` and `scipy.signal.convolve2d` functions for performing gaussian filtering and convolution operations, respectively.

[ ]:
```
# function that performs edge detection
def edge_detection(img):
  """
  img: input image (H,W)
```

```
    returns:
    gmag: gradient magnitude image (H,W)
    ang: angle image (H,W)
    gmag_nms: gradient magnitude image after non-maximal suppression (H,W)
    """
    # your code here


    return gmag, ang, gmag_nms
```

2) Call the function `edge_detection` function with the cameraman image. Display the input image and all three output images from your `edge_detection` function.

[ ]:
```
"""
Call the function edge_detection function with the cameraman image.
Display the input image and all three output images from your edge_detection␣
↪function.
"""
# your code here
```

3) Briefly discuss the resulting images and how you might implement edge linking techniques using these outputs. Explain your reasoning.

Your answer here:

3.3 Part 3: Region segmentation using k-means clustering (20 points)

[ ]:
```
# Read and display image
img = data.rocket()
plt.imshow(img)
plt.show()
```

1) Complete the funciton `region_segmentation` that performs region-based segmentation using k-means clustering. The function that takes an input image, number of clusters k, nonnegative convergence threshold T, and maximum number of iterations N as inputs and performs the following steps:

- Normalizes the input image to [0,1] range.
- Randomly initializes a set of means
- Then iteratively assigns samples to clusters using Euclidean distance followed by cluster mean m_i updates. The script must perform a test for convergence

$$\sum_{i=1}^{k} \|m_i^{(t)} - m_i^{(t-1)}\| < T$$

where T is the specified nonnegative convergence threshold T, but must also stop iterating after a specified maximum number of iterations N.

- Returns the output image with region-based segmentation, and the number of iterations t. The output image must assign each pixel to the mean m_i value corresponding to its cluster.

Notes: - Your function may take a long time to converge if you do not vectorize your code. - You may use the function `np.random.rand` for generating random numbers

[ ]:
```
# function that performs region-based segmenation using k-means clustering
def region_segmentation(img, k, N, T):
  """
  img: input image(H,W)
```

```
    k: number of clusters (integer)
    N: maximum number of iterations (integer)
    T: non-negative convergence threshold (floating-point)

    returns:
    out: output image after region-based segmentation (H,W)
    t: number of iterations
    """
    # your code here


    return out, t
```

2) Call the function `region_segmentation` with the rocket image, $N = 80$, $T = 0.01$ and two different values of $k = 3$ and $k = 6$. For each value of k, run the `region_segmentation` function twice.

Display the input image, along with output images for $k = 3$ and $k = 6$ (4 output images in total: 2 runs * 2 values of k). Also, print the number of iterations t taken by your function for different values of k.

Note:

- You may get different outputs for each run since the means are initiatized randomly.

[ ]:
```
"""
Call the function region_segmentation with the rocket image, N = 80, T=0.01 and␣
 ↪two different values of k=3 and k=6. For each value of k, run the␣
 ↪region_segmentation function twice.
Display the input image, along with output images for k = 3 and k = 6 (4 output␣
 ↪images in total: 2 runs * 2 values of k).
Also, print the number of iterations t taken by your function for different␣
 ↪values of k.
"""
# your code here
```

4) Briefly describe the differences between the two images and how k impacts the result.

Your answer here: