CSE166_WI23_assignment_7

March 8, 2023

1 CSE 166: Image Processing, Winter 2023 – Assignment 7

- Instructor: Ben Ochoa
- Due: Wednesday, March 15, 11:59 PM

1.1 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 review the Academic Integrity Policy and Collaboration Policy on Canvas.
- All the solutions must be written in the Jupyter notebook only.
- 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. While submitting the PDF on gradescope -
  - Make sure that the outputs generated are clearly visible and are not cut-off or partially cropped in the final PDF.
  - Make sure to assign the relevant pages in your PDF submission for each problem.
  - Do not export the jupyter notebook as a single page. Please see the detailed submission instructions at the end of this file.
- 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.

2 Problem 1: Textbook problems (7 points)

Note: As stated in the syllabus, you must use the 4th edition of the textbook, ISBN 9780133356724. Answers to different problems will not be accepted.

- The 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 10.6 (1 point)

Your answer here

2.2 b) Problem 10.10 (2 points)

Your answer here

2.3 c) Problem 10.26 (2 points)

Your answer here

2.4 d) Problem 10.41 (2 points)

Your answer here

3 Problem 2: Programming (30 points)

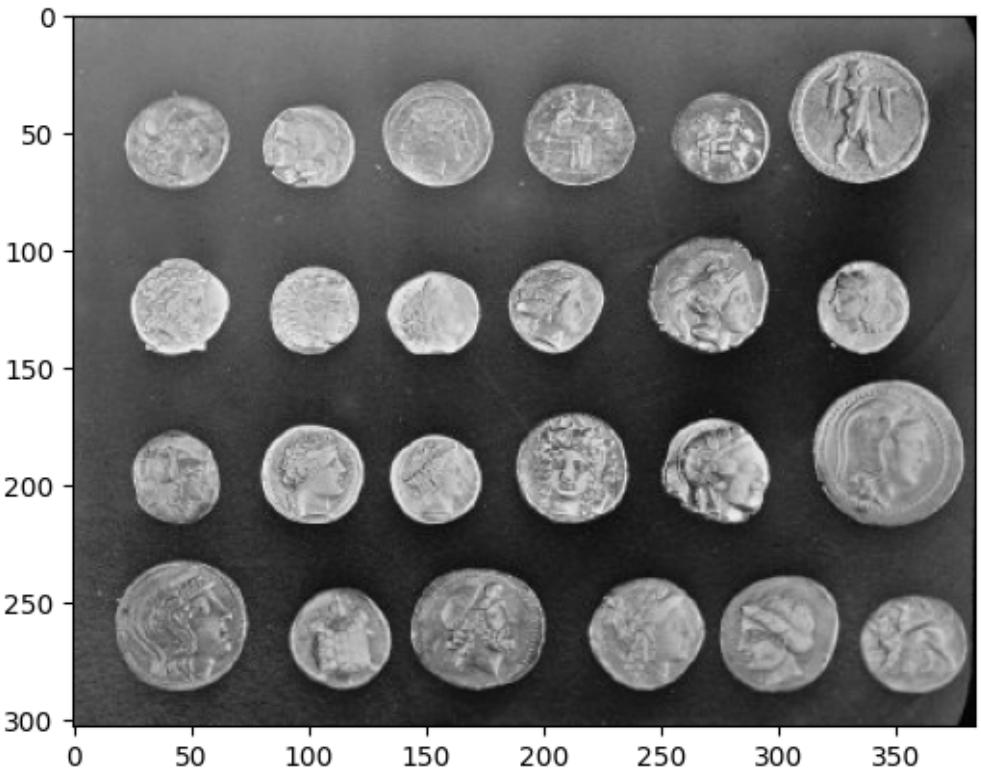
3.1 Part 1: Edge Detection (10 points)

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

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

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

[ ]:
```
<matplotlib.image.AxesImage at 0x1ad4e1f5730>
```



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 $\sigma = 0.5$
- Computes the gradient magnitude and angle images
- Applies non-maximal 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 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` with the coins 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.

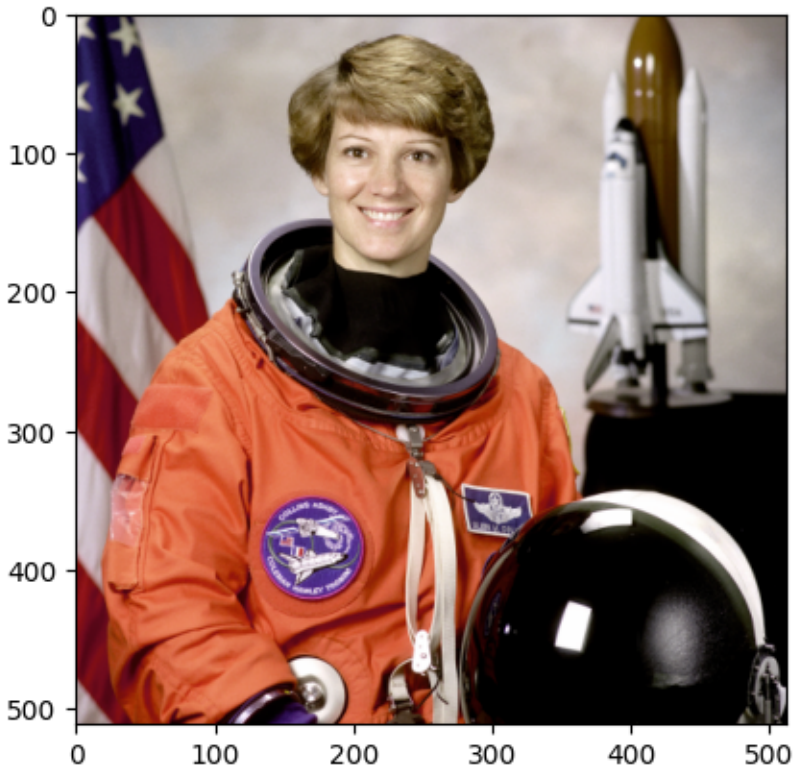
Your answer here

3.2 Part 2: Region segmentation using k-means clustering (20 points)

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

plt.imshow(img)
```

[ ]:
```
<matplotlib.image.AxesImage at 0x1ad4e23af10>
```



1) Complete the function `region_segmentation` that performs region-based segmentation using k-means clustering algorithm. The function takes an input image, number of clusters k, non-negative 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 based on Euclidean distance followed by cluster mean m_i updates. The script must perform the following test for convergence but must also stop iterating after a specified maximum number of iterations N

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

  where T is the specified non-negative convergence threshold.
- 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 astronaut image, $N = 100$, $T = 0.001$ and two different values of $k = 4$ and $k = 8$. For each value of k, run the `region_segmentation` function twice.

- **Note**: You may get different outputs for each run since the means are initialized randomly.

3) Display the input image, along with output images for $k = 4$ and $k = 8$ (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 output images (for $k = 4$ and $k = 8$) and how k impacts the result.

Your answer here

4 Submission Instructions

1. Remember to submit **both** the Jupyter notebook file (`.ipynb`) and the PDF version (`.pdf`) of this notebook to Gradescope.
2. 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.
3. While submitting on gradescope, please make sure to assign the relevant pages in your PDF submission for each problem.
4. Do not export the jupyter notebook as a single page.

To convert the notebook to PDF, you can choose one way below:

1. You can print the web page and save as PDF (e.g. Chrome: Right click the web page → Print… → Choose "Destination: Save as PDF" and click "Save").
2. You can find the export option in the header: File → Download as → "PDF via LaTeX"
3. You can use nbconvert (https://nbconvert.readthedocs.io/en/latest/install.html) to convert the ipynb file to pdf using the following command

```
jupyter nbconvert --allow-chromium-download --to webpdf <ipynb filename>
```