Individual Assignment : Week 4

February 27, 2023

This assignment must be solved and submitted **individually** and you are not allowed to work in groups. This assignment count towards your final grade and have to be submitted in order to pass the course. You must follow the report guidelines found in `guidelines.pdf`. The assignment contains questions on the topics we have covered in the course up to now including this week. The page limit for this assignment is **10 pages** including everything, i.e. illustrations and code snippets.

1. Image filtering

1.1. (1 point) Consider the following finite difference approximations of the image derivatives given by:

$$\frac{\partial I}{\partial x}(x,y) \approx \frac{I(x+1,y)-I(x-1,y)}{2}$$
$$\frac{\partial I}{\partial y}(x,y) \approx \frac{I(x,y+1)-I(x,y-1)}{2}$$

Deliverables: Assuming that we filter an image using correlation, what are the filter kernels that implements these approximate derivatives? How would the kernels look like if we instead want to filter the image using convolution? In both cases, where is the center pixel in the filter kernel? Why does it not matter if we use correlation or convolution for filters like Gaussian or mean?

1.2. (1 point) Based upon the linear separablility of the Prewitt and Sobel filters (see lecture slides), reinterpret the effect of the Prewitt and Sobel filters on images.
Deliverables: Write the two filters for y-derivatives in their separable forms. Explain in particular why they are more likely to be robust to noise than the simple finite difference approximation of the previous question.

2. Fourier transform

2.1. (1 point) The Fourier transform of a Gaussian function is also a Gaussian function. Using this fact and the convolution theorem, write a function `scale_fft`, which implements convolution with a Gaussian kernel parameterised by its standard deviation (its scale) σ in Fourier domain by constructing the filter kernel in the frequency space (i.e. without application of the discrete Fourier transformation on the kernel). Apply it to `trui.png` for a range of σ scales of your choice.

Deliverables: Include the code for your function in the report and explain how convolution can be implemented using the Fourier transform. Include illustrations of the function applied to `trui.png` at different σ scales.

3. **Histogram-based processing**

   3.1. (1 point) Implement a Python function that, for a given histogram of a grayscale image, computes its cumulative distribution function CDF (you may use the *numpy.cumsum* function and normalize the expression).

   Deliverables: Include the code for your Python function as a code snippet in the report. Display the result of computing the CDF for the image *pout.tif* using your function. What do the regions of fast increase of the function correspond to? And what about its flat regions?

   3.2. (1 point) Given an image I and its CDF C, write a Python function that computes the floating-point image $C(I)$ such that the intensity at each pixel (x, y) is $C(I(x, y))$.

   Deliverables: Include the code for your Python function. Show the result of applying this function to the *pout.tif* image.

   3.3. (1 point) The CDF is in general not invertible – why ? To overcome the problem of non-invertibility for histogram matching, we can consider instead a pseudo-inverse. For any CDF function $C(s)$ defined on the integer set $s \in \{0, .., 255\}$, we define its pseudo-inverse $C^{(-1)}$ as follows :

$$C^{(-1)}(l) = \min\{s \mid C(s) \geq l\} ,$$

   where $l \in [0, 1]$. Write a Python function that computes the pseudo-inverse of any given CDF (you may use *numpy.min* or *numpy.where* function).

   Deliverables: Include your answer to the first question and the code for your Python function as a code snippet.

   3.4. (1 point) Assume we have two gray scale images I_1 and I_2 and the corresponding CDFs C_1 and C_2, then histogram matching can be described by the mapping

$$J(x, y) = C_2^{-1} \left(C_1(I_1(x, y)) \right) .$$

   Based on this equation and on the previous questions, implement your own Python function to perform histogram matching between two images.

   Deliverables: Include the code for your Python function and show the two input and resulting images. Also plot and compare the cumulative histograms of the original image, the target and the one obtained by histogram matching.

4. **Segmentation by thresholding**

   4.1. (1 point) Perform foreground-background segmentation using intensity thresholding on the `pillsect.png` image separating the objects from the background. Start by converting the image to grayscale, then apply thresholding (e.g. using `np.where`) using the threshold value 100 or 100/256, depending on whether you

represent the grayscale image in unsigned 8 bit or floating point.
Deliverables: Show the original image, grayscale image and your binary segmentation image side-by-side, and include the essential code snippet in the report. Did we succeed in separating the foreground objects from the background?

4.2. (1 point) Plot the intensity histogram of the grayscale image of `pillsect.png` from the previous question (you can for instance use `matplotlib.pyplot.hist`). Was the choice of threshold value in the previous question appropriate? What happens in the two cases where we either lower or increase the threshold value?
Deliverables: Include histogram plot in the report and your brief answer to the questions

5. Morphology

5.1. (1 point) The segmentation you produced in question 4.1. contains spurious noisy pixels. Using binary morphology, write a program to remove the noise, but without destroying the shape of the foreground objects. You may use the `binary` functions from `skimage.morphology`.
Deliverables: Argue for your choice of morphological operation and structuring element. Provide the essential code snippet in the report. Illustrate the result of your solution by visualising the cleaned binary segmentation mask.