Homework 2

Introduction to Signal and Image Processing

Handout: March 25th, 2020
Handin: April 8th, 2020 15:15

Instructions

This homework is an opportunity for you to have hands on experience manipulating and working with images and the material covered in class. Your hand-in will consist of a `.zip` archive named `hw2_firstName_lastName.zip` containing the following:

- Python source files named `hw2_exY_firstName_lastName.py`, where Y is the exercise number.
- All necessary files to run the above.
- Your report named `hw2_firstName_lastName.pdf`.

Your archive must be uploaded on ILIAS before the deadline.

Code

Code templates are provided to help you get started in solving the exercises. In the typical case, you will fill-in the missing function bodies and code blocks. Note that you may also make your own code from scratch.

IMPORTANT: In general, if you are not able to produce a functional code (i.e. your script crashes), comment out the concerned part, and if necessary give a short analysis in your report. Scripts that do not run will be penalized.

Report [4 points]

Along with the code, you are asked to provide a short report. Comment on all the questions on the report, show your results, including figures of all requested plots, and briefly explain what you did to obtain them. If you encountered problems and did not manage to finish the exercise, explain here what you did. On ILIAS you will find a LaTeX template to get started. Note that the use of LaTeX is NOT mandatory.

1 Linear Filtering - [7 Points]

1.1 [0.5 Points]

Write a function that returns a box filter kernel of size $[n \times n]$. Make sure the kernel values sum up to 1. If you are using the provided template you may fill in the `boxfilter(n)` function.

HINT: This can be done as a simple one-line function.

1.2 [1 Point]

Implement a 2D convolution function between an image $[m \times n]$ and a filter kernel $[k \times l]$. DO NOT USE any built-in convolution functions (such as from scipy or numpy). You should code your own version of the convolution, valid for both 2D and 1D filters. If you are using the provided template you may fill in the `myconv2(image,filt)` function.

IMPORTANT: The function should return a full convolution, meaning the output should be of size $(m+k-1) \times (n+l-1)$. Use any border filling approach you choose to ensure values at the border. For example, you could assume 0 for values outside the image (zero padding).

HINT: Other numpy functions, e.g. `np.newaxis()`, `np.flip()` and `np.pad()` are allowed for usage.

1.3 [0.5 Point]

In your script, create a boxfilter kernel of size 11 and convolve this kernel with a grayscale image (e.g., *cat.jpg*). Show your result in the report.

1.4 [1 Point]

Write a function that returns a 1D Gaussian filter kernel for a given value of sigma. The kernel should be a vector of length `filter_length`, if `filter_length` is odd. If `filter_length` is even, set the vector length to `filter_length + 1`, to ensure that you have a center element.

Each value of the filter kernel can be computed from the Gaussian function $f(x) = e^{-\frac{x^2}{2 \cdot \sigma^2}}$, where x is the distance of an array value from the center. This formula for the Gaussian ignores a constant factor, so you should normalize the values in the kernel so they sum to 1. If you are using the provided template you may fill in the `gauss1d(sigma, filter_length)` function.

HINT: For efficiency and compactness, it is best to avoid "for" loops. One way to do this is to first generate a vector of values for x, for example [-3 -2 -1 0 1 2 3] for a filter with length 7. These can then be used to calculate the Gaussian value corresponding to each element.

1.5 [1 Point]

Write a function that returns a 2D Gaussian filter kernel. Remember that a 2D Gaussian can be formed by convolution of a 1D Gaussian with its transpose. Use your function from 1.2 to perform the convolution. If you are using the provided template you may fill in the `gauss2d(sigma, filter_size)` function.

Display a 2D gaussian filter with sigma=3 and size 11x11 in the report.

1.6 [1 Point]

Implement 2D gaussian filtering for an image.

Convolve a grayscale image (e.g.,*cat.jpg*) with a Gaussian filter kernel with a sigma of 3 and display the result.

If you are using the provided template you may fill in the `gconv(image,sigma)` function.

1.7 [1 Point]

Convolution with a 2D Gaussian filter is not the most efficient way to perform Gaussian convolution with an image. In a few sentences, explain how this could be implemented more efficiently and why this would be faster.

Answer this question in your report.

HINT: How can we use 1D Gaussians?

1.8 [1 Point]

To show this fact create a plot of kernel size vs. computation time. In this plot, show in one color (e.g. blue) what this size-time relation is when using 2D box filter of increasing size (from 3 to 103), with steps of 5). Repeat the same for your solution to (1.7) and show it in another color.

HINT: Use `time.time()` to calculate the durations.

2 Gradients and Edge Maps - [7 Points]

2.1 [1 Points]

Begin by defining the derivative operator in x direction ($dx = [-1, 0, 1]$). Using your functions from Exercise 1, convolve dx with a Gaussian filter with sigma = 1. Repeat the same for the derivative operator in y direction (dy).

2.2 [2 Points]

Using dx and dy from the previous question, construct an edge magnitude image (e.g., for the file *bird.jpg*). That is, for every pixel in the image, assign the magnitude of gradients:

$$|I_G| = \sqrt{I_x^2 + I_y^2}.$$

Also calculate for each pixel the gradient orientation (e.g., using the `np.arctan2()` function):

$$\theta = \arctan(I_y/I_x).$$

Display the gradient magnitude image next to the original one. If you are using the provided template you may fill in the `create_edge_magn_image(image, dx, dy)` function.

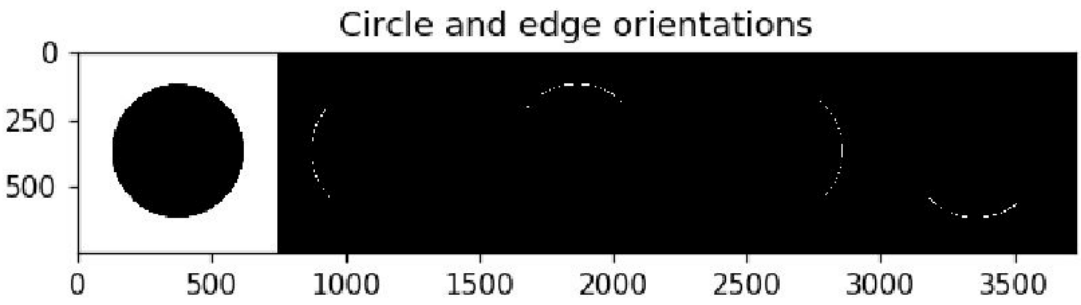

Figure 1: Edge maps of a circle for 4 different directions.

2.3 [4 Points]

You will now produce edge images of particular directions. An edge image is a binary image. In this case, you will produce 4 such images. For $m = 0, \ldots, 3$, an edge image e_m should have values 0 except if a pixel has an edge of magnitude greater than a threshold $|I_{G,edge}|$ and has a gradient orientation in the interval $[\frac{m \cdot 2\pi}{4} - \frac{\pi}{4}, \quad \frac{m \cdot 2\pi}{4} + \frac{\pi}{4}]$. In this case, the pixel should have value 255. Please find a value for $|I_{G,edge}|$ which is suitable for your image and describe your reasoning for this threshold in the report.

If you are using the provided template you may fill in the `make_edge_map(image, dx, dy)` function.

Show the result of these edge images by concatenating all of them with the original and magnitude edge image (similar to figure 1).

HINT 1: To verify your code, run it on image `circle.jpg`. You should generate edge images that cover the extent of the perimeter of the circle, similar to the maps shown in figure 1.

HINT 2: The output of the arctan-function is mapped to the interval $[-\pi, \pi]$. Make sure that you assign the direction intervals correctly.

3 Harris Corner detection - [7 Points]

For this exercise you can either use your functions from exercise 1, or `scipy.signal.convolve()`.

3.1 [3 Points]

Write a function which computes the harris corner for each pixel in the image. The function should return the R response at each location of the image.

If you are using the provided template you may fill in the `myharris(image, w_size, sigma, k)` function, where `w_size` is an integer denoting the window size over which the gradients will be summed, `sigma` is the parameter for the Gaussian smoothing filter and `k` is a constant factor required to compute the corner response measure R.

HINT: You may have to play with different parameters to have appropriate R maps. Try Gaussian smoothing with $sigma = 0.2$, gradient summing over a 5x5 region around each pixel and $k = 0.1$.

3.2 [1 Point]

Evaluate your function from question 3.1 on the chessboard image. Show the result.

3.3 [1 Point]

Repeat the same as (3.2), but this time rotate the chessboard image by 45 degrees (in either direction). Show the result.

You can use the built in `scipy.ndimage.rotate()` function.

3.4 [1 Point]

Repeat the same as (3.2), but this time downscale the chessboard image by a factor of a half. Show the result.

You can use the built in `scipy.misc.imresize()` function.

3.5 [1 Point]

Looking at the results from (3.2), (3.3) and (3.4) what can we say about the properties of Harris corners? What is maintained? What is it invariant to? Why is that the case?

Answer this question in your report.