Computer vision: Foundations
Summer Semester 2020

Exercise Sheet #1

Due: 12.05.2020, 15:15

Fred Hamprecht, Alberto Bailoni
`cvflecture.s20@gmail.com`

General Regulations.

- Hand in your solutions in groups of two or three people (preferred).
- Practical exercises should be implemented in python and submitted as a jupyter notebook (`.ipynb`). Always provide both the (commented) code as well as the output, and don't forget to explain/interpret the latter.
- If possible, theoretical exercises should be typeset, either by providing your answers in the provided jupyter notebook (use markdown blocks and LaTeX code for equations) or by submitting a separate `.pdf` file created with LaTeX. Alternatively, a scan of handwritten notes is also accepted.
- Submit all your files in a single `.zip` archive and upload it using the Physics platform https://uebungen.physik.uni-heidelberg.de/h/1176. **Only one person in the group should upload the zip archive**: in the upload interface you will see the option to indicate your exercise-partners.
- Note that this first exercise sheet is longer and the due date is in two weeks.

Prerequisite: Working with Jupyter Notebooks

- We suggest to download `miniconda` https://docs.conda.io/en/latest/miniconda.html to manage virtual environments. To make the changes take effect after installing it, make sure to close and then re-open your terminal window. To test your installation: in your terminal window, run the command `conda list`. A list of installed packages appears if it has been installed correctly.
- **The necessary dependencies for completing the exercises of this first sheet can be found in the `requirements.yml` file**. You can easily create a new environment for the lecture with all the dependencies by moving to the assignment folder and then running the command `conda env create -n CVF20 -f requirements.yml`
- Every time you run your scripts or install new packages, you should first activate your environment with `conda activate CVF20` (and deactivate it at the end with `conda deactivate`)
- To start working on the exercise sheet, start the notebook with the `jupyter notebook` command and then open http://localhost:8888 in your browser. If you are not familiar with jupyter notebooks, you can have a look at a tutorial online.

1 Plot an image in Python (1pt)

In the provided Jupyter Notebook `ex01.ipynb`, write a short python script that does the following:

(a) Load the image `data/coral.jpg` into a `numpy` array

(b) Plot the three color channels separately using `matplotlib`

2 Convolution (12pt)

(a) Commutative property (1pt) Let's consider two periodic 1D signals f and h of length N. Then their convolution is defined as:

$$(h * f)[x] = \sum_{i=0}^{N-1} h[i]f[x-i] \qquad \forall x \in [0, N-1]. \tag{1}$$

Prove that the convolution operator is commutative, i.e. $f * h = h * f$.

(b) Boundary conditions (2pt) In computer vision, we usually do not work with periodic signals. Let's instead consider a 1D signal f of length N and a kernel h with odd length $2K+1$, so that its elements are indexed from $-K$ to K and the central element is $h(0)$. Their convolution is then given by:

$$(h * f)[x] = \sum_{i=-K}^{K} h[i]f[x-i] \qquad \forall x \in [K, N-1-K]. \tag{2}$$

Note that this formulation does not extend to the boundaries ("valid" convolution). Another common approach consists in padding the input signal f with additional zeros at the boundaries, so that the convolution $h*f$ in (2) has the same length of the input and is defined in $[0, N-1]$ ("same" convolution). Now consider the kernel $h[x] = [-1, 0, 1]$ and the signal $f[x] = [1, 1, 0, 0, 1, 0, 1, 0, 0]$: What is the "valid" convolution $h * f$? What is instead the result of the convolution with the zero-padding boundary condition?

(c) Implementation (5pt) The definition of convolution in (2) easily generalizes to 2D images. Finish implementing the `conv_2D()` function in `filters.py`, which should compute the convolution for a 2D kernel and a 2D image using the zero-padding boundary condition. Then test it with the code provided in the jupyter notebook.

(d) Separable kernels (4pt) Let's consider a 3D image with shape $(100, 800, 800)$ produced by a CT scan. We now apply a $9 \times 25 \times 25$ Gaussian blur filter (see e.g. https://en.wikipedia.org/wiki/Kernel_(image_processing)) using zero-padding. If we naively compute a 3D-convolution, how many multiplication and addition operations do we need to perform? What is the speedup if we note that the Gaussian kernel is separable and we can then apply three 1D-convolutions?

3 Scharr Filter and Edge Detection (6pt)

Starting here, we will work with images of yeast cells from an ongoing project[1] and solve increasingly difficult problems.

(a) (1pt) Load `yeast_cells_cropped.tif` in python as a numpy `uint8` array. Make sure to rescale the values in the image and obtain a `float32` array with values in $[0, 1]$.

(b) (2pt) Set up a 1D filter $[+1, 0, -1]$ and convolve it with every row of the image, so that we approximate a horizontal derivative (use `scipy.ndimage.convolve`). Then consider a 1D smoothing filter $[3, 10, 3]$ and convolve it with every column of the previous output. Plot your results.

(c) (2pt) The Scharr filters are given by:

$$S_x = \begin{bmatrix} +3 & 0 & -3 \\ +10 & 0 & -10 \\ +3 & 0 & -3 \end{bmatrix} \qquad S_y = \begin{bmatrix} +3 & +10 & +3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix} \tag{3}$$

In the previous point, you applied the S_x filter by using the fact that its kernel is separable. Now apply S_y to the image and plot your results. Finally, compute the magnitude of the gradient. Plot and comment your results.

[1]This yeast-cells dataset was kindly made available by Joel Goodman, *UT Southwestern*, Dallas, Texas.

(d) (1pt) Now repeat the exercise again but smooth the original image with a Gaussian blur filter (`scipy.ndimage.gaussian_filter`) before to apply the Scharr filter. What does it change when you increase the sigma parameter of the smoothing filter? Plot and comment your results.

4 Bonus: Segmentation of Yeast Cells (8pt - bonus)

Now consider the full image `yeast_cells.tif`. In this exercise we want to perform a segmentation of the yeast cells and separate them from the background.

(a) (1pt) Load the image in python and try to segment the cells by thresholding the gray values in the image. Plot your results. Why is it difficult to obtain a good segmentation with this method?

(b) (4pt) Considering what you learned in this first lecture, can you think about a smarter method to obtain better segmentation results?

(c) (3pt) Implement your idea and plot your results.

5 Segmentation of Yeast Cells with ilastik (8pt)

In this exercise we will try again to segment yeast cells by training a simple classifier in ilastik.

(a) Download ilastik and read the tutorial on pixel classification `https://www.ilastik.org/documentation/pixelclassification/pixelclassification`

(b) Load the `yeast_cells.tif` image used in the previous exercise.

(c) (5pt) Try an "active learning" strategy to train the classifier: label a set of 10 randomly chosen background and foreground points/strokes; then turn on the *Live update* mode and choose other 10 points that are misclassified. Show (screenshot or export) the labels of your final segmentation.

(d) (3pt - bonus) Play with different choices of *Feature selection* and comment your results.