Exercise Sheet #2

Due: 19.05.2020, 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.

1 Downsampling an image (9+2pt)

In this exercise we will compare different methods to downscale the image `sign.jpg`. In order to make sure that no interpolation is performed automatically by `matplotlib`, do not forget to set the following option in your jupyter notebook:

```
matplotlib.pyplot.rcParams["image.interpolation"] = None
```

(a) Simple subsampling (1pt) First, load the image in python and plot it at the original resolution. We now want to downsample by a factor of 13 in both directions. As a first approach, start from the pixel at the top-left corner of the original image and then keep only one pixel in every 13 pixels along x and y. Plot and comment your output image.

(b) Shifting the image (1pt) Now repeat, but instead of starting from the pixel at the top-left corner (with coordinates $(0, 0)$ in the original image) start from the pixel with coordinates $(4, 4)$: is the result different? What if you start from the pixel with coordinates $(8, 8)$? Plot and comment your results.

Let us now filter the input image before subsampling it. The size of the filter kernel should depend on the downscaling factor.

(c) Box filter (2pt) We first consider a simple box filter. If the downscaling factor is N, then the box kernel will also have size N (see Fig. 1d and 1e for two examples with $N = 3$ and $N = 5$, respectively). Write a function that returns the box kernel for a given dowscaling factor N and then apply the filter to the image `sign.jpg` for $N = 13$. *Hint: use "valid" convolutions to avoid boundary artifacts; similarly to what you have done in the first exercise sheet, use the function `scipy.ndimage.convolve` and the fact that the kernel is separable.*

(d) (1pt) After applying the filter, subsample the image by keeping only one pixel in every 13 pixels starting from the top-left corner. Plot and compare the output with your previous results obtained in point (a). How does the output change if you start subsampling from the pixel with coordinates $(4, 4)$ in the filtered image?

(e) (1pt) When we apply the filter and then subsample the image, we are wasting some computations. Why? Have a look at the `torch.nn.functional.conv1d` function in the `pytorch` module: https://pytorch.org/docs/stable/nn.functional.html. Which argument of the `conv1d` function would you use to avoid the waste of computations?

(f) Lanczos filter (2pt) The box filter is very fast (because of its small kernel), but it is not a very good low-pass filter and its frequency response can sometimes lead to aliasing when downsampling. The sinc function is the ideal low-pass filter, but it is slower and gives ringing artifacts. Thus, we now consider a commonly used approximation given by the Lanczos windowed-sinc function:

$$\text{Lanczos}_\alpha(x) = \begin{cases} 1 & \text{if} \quad |x| = 0 \\ \frac{\sin(\pi x)}{\pi x} \frac{\sin(\frac{\pi x}{\alpha})}{\frac{\pi x}{\alpha}} & \text{if} \quad 0 < |x| < \alpha \\ 0 & \text{if} \quad |x| \geq \alpha. \end{cases} \tag{1}$$

The two most common choices for the parameter α are 2 and 3, giving a 2-lobed and 3-lobed function, respectively. Write a function computing the normalized Lanczos kernel for a given downscaling factor N and a parameter α. Before to normalize it, its elements should be given by:

$$\text{Lanczos}_\alpha\left(\frac{i}{N}\right) \qquad \text{with} \quad i \in \{-N\alpha, \ldots, N\alpha\} \tag{2}$$

In Fig. 1f, we show an example of a normalized kernel for $\alpha = 2$ and $N = 3$. More examples and details can be found on pages 10-11 of `http://www.realitypixels.com/turk/computergraphics/ResamplingFilters.pdf`. Using your implemented function, print the Lanczos kernel for $N = 9$ and $\alpha = 2$.

(g) (1pt) Apply the Lanczos filter to the image `sign.jpg` for $N = 13$ and $\alpha = 3$ and then subsample the image. Plot and comment your results.

(h) High-frequency image (2pt - Bonus) Downscale the image `high-frequency.png` by a factor of $N = 5$ with the three methods you implemented previously: sub-sampling using the box filter, the Lanczos filter and no filter at all. Save the three outputs as images and comment why the results are different. *Note that we ask you to save your outputs because if you plot your results with the matplotlib option "`image.interpolation`" set to `None`, matplotlib will likely perform a basic subsampling during the visualization and this could result in plots with aliasing artifacts because of the high frequency content of this image.*

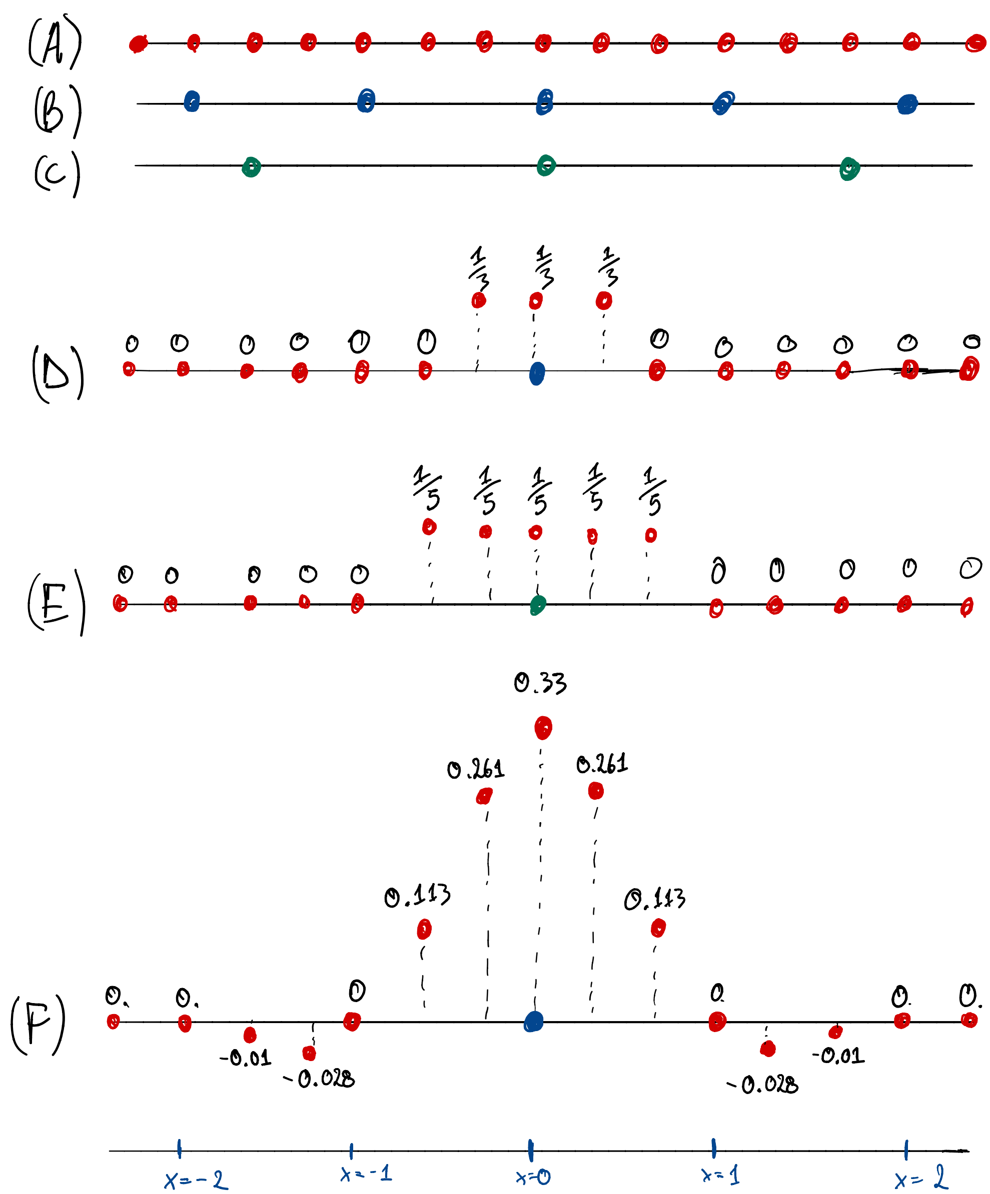

Figure 1: **(a)** Representation of the points/pixels in the input 1D-image at the original resolution. **(b)** Pixels in the output 1D-image downscaled with a factor $N = 3$. **(c)** Pixels in the output 1D-image downscaled with a factor $N = 5$. **(d)** Example of normalized box kernel (applied to the central pixel) for $N = 3$. **(e)** Example of normalized box kernel (applied to the central pixel) for $N = 5$. **(f)** Example of normalized Lanczos kernel (applied to the central pixel) for $N = 3$ and $\alpha = 2$.