Homework 2

Prof. Raphael Sznitman
Introduction to Signal and Image Processing

Handout: March 21, 2018
Handin: April 11, 2018, 14:15

Instructions

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 [7 points]

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 [3 points]

Along with the code, you are asked to provide a short report (max. 2 pages). The questions will give you indications on its content. On ILIAS you will find a latex template to get started. Note that the use of latex is not mandatory.

1 Gaussian Filtering [2.5 points]

For this question, you will have to create smoothed images through Gaussian Filtering.

1.1 Write a function `myconv(signal, filt)` that takes 1-dimensional signal `signal` and 1-dimensional filter `filt` and outputs the filtered signal, i.e. the convolution of the signal with a given filter. The output of the convolution should be full size.

1.2 Write a function `myconv2(img, filt)` that takes an image `img` and 2-dimensional filter `filt` and outputs the filtered image. The output of the convolution should be full size.

1.3 Write a function `gconv(img, sigma, filter_size)` that convolves the image `img` with a gaussian filter of standard deviation of `sigma` and of size `filter_size`. Use your function `myconv2` for convolution. You can use the given function `gauss2d()` to create 2-dimensional gaussian filter.

1.4 Convolve the image `ex1.img` with a 2-dimensional gaussian filter with different standard deviations and filter sizes of your preference in order to obtain smoothed images. Use your function `gconv()`.

1.5 What do you observe if you increase the standard deviation/filter size? Comment on your observations.

2 Edge Map [1.5 points]

In this section, you are going to implement the Difference of Gaussians (DoG) operator that is well used in computer vision applications. The goal of this exercise is to create an edge map for a given image by DoG operation. DoG computes the difference of two blurred versions of a given image.

2.1 Implement a function `DoG(img, sigma_1, sigma_2, filter_size)` that takes an image `img`, two different standard deviations `sigma_1, sigma_2` and a filter size `filter_size` as inputs and creates the DoG for the given image. Use the function `gconv()` that you implemented in the previous question. In the case where you could not implement convolution in question 1.3, use the built-in `scipy.signal.convolve2d()`.

2.2 Create DoG for the image `ex2_img.png` with the following parameters: `sigma_1` $= 1,$ `sigma_2` $= 1.5,$ `filter_size` $= 30$. Show your results in the form of Fig. 1.

2.3 Create DoG for the same image with the following parameters: `sigma_1` $= 1,$ `sigma_2` $= 25,$ `filter_size` $= 30$. Show the result.

2.4 What did you observe for the two different DoGs? Did you observe edges? Why and how DoG will (or will not) help creating edges? Comment on your findings and observations.

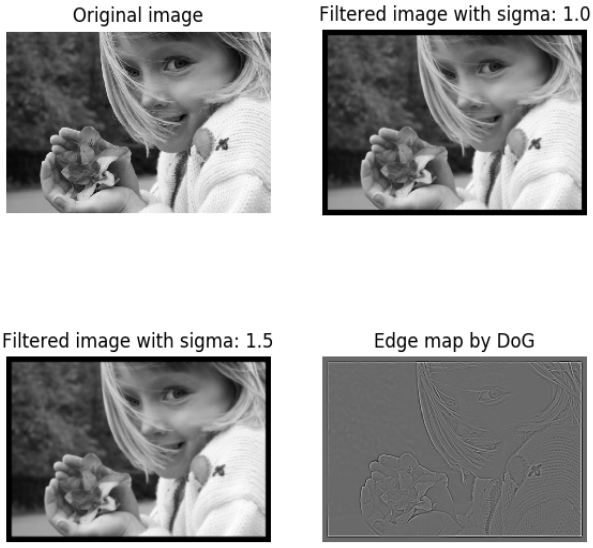

Figure 1: Filtered images and DoG of the original image

3 Face Detector [3 points]

In this exercise, you will implement a face detector by means of template matching. This approach consists in searching a given example template within an image. In practice, the object of interest may exist at different scales, i.e. be bigger or smaller than the template. Therefore we take into account different scales of image and/or template. The following steps will help you perform a multi-resolution template matching technique to detect faces in an image.

3.1 A Gaussian Pyramid is a multi-scale representation of an image by subsequently burring and sub-sampling the image. Every scale of the pyramid is called *level* as seen in Fig. 3. This is a well-known technique to make multi-resolution image analysis, which is aligned with the purpose of this exercise.
You will implement a function `generate_gaussian_pyramid(img, sigma, filter_size, scale, num_levels)` that creates a Gaussian Pyramid of the given image `img` for standard deviation `sigma`, filter size `filter_size` and number of levels `num_levels` as given by the following steps:

a. First write a function `blur_and_downsample(img, sigma, filter_size, scale)` which filters the image `img` with a Gaussian filter of standard deviation `sigma` and size `filter_size` and downscales it with scaling factor `scale`. It returns the blurred and down-scaled image. You may use `scipy.misc.imresize()` for the down-scaling operation.

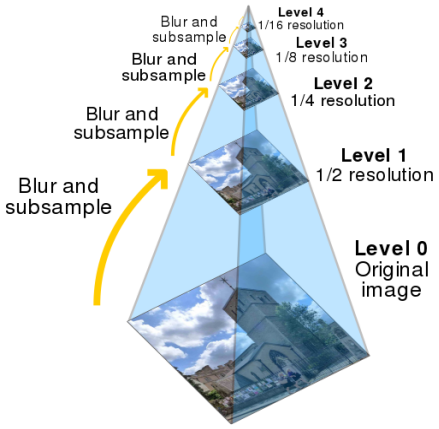

Figure 2: Image pyramid with 5 levels [from en.wikipedia.org/wiki/Pyramid_(image_processing)]

b. Implement `generate_gaussian_pyramid(img, sigma, filter_size, scale, num_levels)` that creates a gaussian pyramid of `num_levels` by repetitively blurring and down-sampling the image. *Please keep in mind that a blurring and down-scaling at level n is performed on previously blurred and down-scaled image at level $n-1$.* Also note that level-0 of the gaussian pyramid is considered to be the original image (before any downsampling and filtering).

c. Create the Gaussian pyramid for an image with parameters of your choice. Show your results.

3.2 To detect faces, you will use a sample face from the image as a template and search for the image patches that have high similarity with it. As discussed earlier, in order to capture every different sized face, we need to perform template matching at different scales. Accordingly, create a Gaussian Pyramid of the template `ex3_template.jpg` using `generate_gaussian_pyramid()` with parameters `sigma` $= 3$, `filter_size` $= 5$, `scale` $= 0.8$, `num_levels` $= 2$.

3.3 To quantify the similarity between a template and an image patch, one can use Normalized Cross Correlation (NCC) given by

$$NCC(u,v) = \frac{\sum_{x,y}(I(x,y) - \bar{I}_{u,v})(t(x-u,y-u) - \bar{t})}{\sqrt{\sum_{x,y}(I(x,y) - \bar{I}_{u,v})^2 \sum_{x,y}(t(x-u,y-u) - \bar{t})^2}},$$

where $I(x,y)$ is the intensity at location (x,y), t is the template, $\bar{t}$ is the mean intensity of the template and $\bar{I}_{u,v}$ is the mean of the I within the area of the template t shifted to location (u,v). Compute the NCC matrix between the image `ex3_img.jpg` and each level of the Gaussian Pyramid of the template `ex3_template.jpg`. You can use `skimage.feature.match_template()` that performs NCC for a given image and template. Set `pad_input=True` for the function to obtain the NCC matrix of the same size with the original image.

3.4 NCC calculates a similarity measure between the template and each overlapping patch in the image. Higher values would therefore mean higher similarity with the template. In order to locate image patches with high similarity with the template, we will threshold the NCC matrices using a pre-defined threshold value.
Create binary images I_b^n for each level n that are of the same size with the NCC matrices, containing only values from $\{0, 1\}$ as in the following:

$$I_b^n(u,v) = \begin{cases} 1 & \text{if } NCC(u,v) > \alpha, \\ 0 & \text{otherwise} \end{cases}$$

where α is the threshold value. Use $\alpha = 0.56$ for this exercise.

- Explain what happens when you increase/decrease α.

3.5 The binary images obtained for every level of Gaussian pyramid reflect the locations that are detected to be similar to the template, i.e., they are assumed to be *faces*. We accordingly put a rectangle around the detected locations. Show the detection results that are found for every level, i.e. every scale of the template (Note that this is already coded in the provided code template. If you do not use the template, you can use the part that draws rectangle around the most correlated locations). You should get a similar plot to Fig. 3.

1. What do you observe in the results? Are your detections accurate? What are potential pitfalls with the implemented scheme? Comment on your observations.

2. What would you recommend in order to improve the detection accuracy?

3. You implemented a multi-scale template matching by building Gaussian Pyramid of the template and correlating them with the image. What could be an alternative to this? What is the potential issue with this alternative?

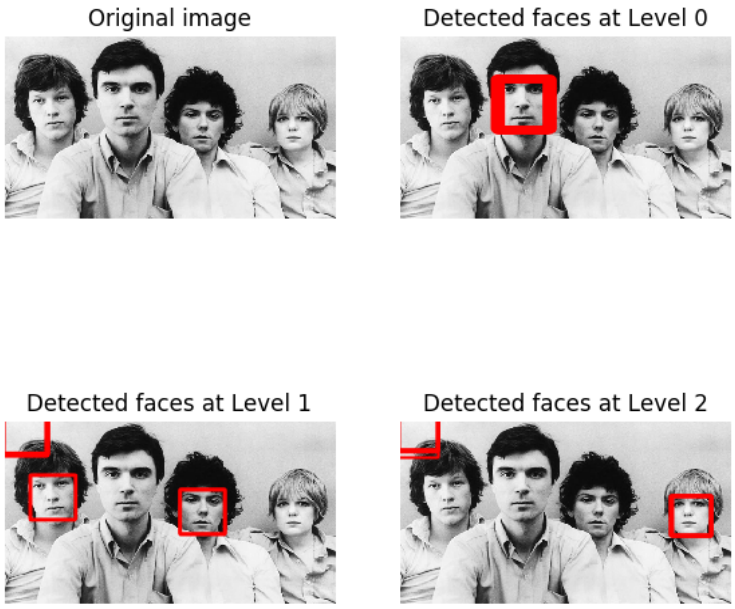

Figure 3: Detected faces obtained by template matching at different scales.