Assignment Template

Santilli Sofia - Matricola 1813509

March-April 2021

1 Exercise 1

Correlation - Symmetric padding

$$\text{image} = \begin{bmatrix} 0 & 0 & 0 & 5 \\ 4 & 3 & 4 & 1 \\ 1 & 0 & 5 & 4 \\ 3 & 4 & 1 & 1 \end{bmatrix}$$

Coordinates (2,2) (3,3)

$$\text{kernel} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 3 \\ 4 & 1 & 1 \end{bmatrix}$$

Discuss the possible problems and advantages of the particular padding strategy.

Correlation consists in multiplying element-wise the image by the filter; so, unlike convolution, we have not to double flip the filter.
Before computing correlation, I apply the symmetric padding to the image in input. Symmetric padding pads through the reflection of the pixels mirrored along the edge of the image. In the padding we have to add at each edge (up, down, left, right) of the image a padding of size 1 ($|p| = \lfloor \frac{k}{2} \rfloor = \lfloor \frac{3}{2} \rfloor = \lfloor 1.5 \rfloor = 1$, where k is the width and height of the kernel, $|p|$ is the size of the padding).

$$\text{imagePad} = \begin{bmatrix} 0 & 0 & 0 & 0 & 5 & 5 \\ 0 & 0 & 0 & 0 & 5 & 5 \\ 4 & 4 & 3 & 4 & 1 & 1 \\ 1 & 1 & 0 & 5 & 4 & 4 \\ 3 & 3 & 4 & 1 & 1 & 1 \\ 3 & 3 & 4 & 1 & 1 & 1 \end{bmatrix}$$

In this way, applying the kernel on *imagePad*, the output image will be of the same dimensions of the input one and the loss of information on the edges is avoided.
Now I can move the kernel across the image: as I have to compute only two coordinates of the resulting image, I simply center the kernel in the pixel of coordinates (2,2) and compute the multiplication element-wise between the kernel and the interested region of the image; then I repeat with coordinates (3,3).

Out(2,2) = 3*0+4*0+1*0+0*0+5*4+4*3+4*4+1*1+1*1 = 20+12+16+1+1 = 50
Out(3.3) = 5*0+4*0+4*0+1*0+1*4+1*3+1*4+1*1+1*1 = 4+3+4+1+1 = 13

Possible problems and advantages of the Symmetric Padding:
Unlike the zero padding and the constant one, the symmetric padding attempts to preserve the distribution of data by re-using the pixels present along the borders; it can be a good feature relating to certain tasks. On the other hand, for some tasks (image detection) this type of padding may represent an issue, as it could introduce unnecessary information, which can be misleading.

2 Exercise 2

Compute the Gaussian filter of dimension 3x3 given a $\sigma = 4$.

I sample the kernel values from the 2D Gaussian function $f(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$, substituting σ=4 and the indices (i,j), considering that the central pixel of the kernel has coordinates (0,0), while the other indices:

(-1,-1)	(-1,0)	(-1,1)
(0,-1)	(0,0)	(0,1)
(1,-1)	(1,0)	(1,1)

Doing the computations I obtain:

f(-1,-1) = f(-1,1) = f(1,-1) = f(1,1) = $\frac{1}{2\pi 4^2} e^{-\frac{1+1}{32}} = \frac{1}{100.48} e^{-\frac{2}{32}} = 0.0093$

f(0,0) = $\frac{1}{2\pi 4^2} e^{-\frac{0+0}{32}} = \frac{1}{100.48} e^{-0} = 0.0099$

f(-1,0) = f(0,-1) = f(0,1) = f(1,0) = $\frac{1}{2\pi 4^2} e^{-\frac{1+0}{32}} = \frac{1}{100.48} e^{-\frac{1}{32}} = 0.0096$

Finally I normalize and obtain the 3x3 kernel:

0.1088	0.1123	0.1088
0.1123	0.1158	0.1123
0.1088	0.1123	0.1088

3 Exercise 3

Given an image and a kernel, tell which filter is the provided one, apply it to the image and discuss the activated features.

$$\text{image} = \begin{bmatrix} 10 & 8 & 1 & 2 \\ 8 & 1 & 2 & 5 \\ 1 & 2 & 5 & 3 \\ 2 & 5 & 3 & 3 \end{bmatrix}$$

$$\text{kernel} = \begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix}$$

The original image presents same intensity values on the diagonals: some diagonals are more relevant (those of 10, 8 and 5 values), other less (1, 2, 3).
This kernel is an oblique (+45 degrees) line filter, that is a convolution mask to detect oblique (+45 degrees) lines in an image.
I decided to apply zero-padding to the original image (even if it implies a reduction of the 10-intensity pixel on the corner); I avoided using symmetric or reflect padding, because they would have mirrored non-zero-intensity pixels along the edges, also producing a new image without equal-intensity values along the diagonals.

$$\text{imagePad} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 8 & 1 & 2 & 0 \\ 0 & 8 & 1 & 2 & 5 & 0 \\ 0 & 1 & 2 & 5 & 3 & 0 \\ 0 & 2 & 5 & 3 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Now I apply convolution to the padded image. In this specific case, it isn't necessary to double flip the kernel, as it is symmetric. As confirm I have flipped the kernel along the x axis, obtaining k_x, and then along the y axis obtaining k_y (which is equal to the original kernel).

$$k_x = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix} \quad k_y = \begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix}$$

Applying convolution:

out(0,0) = 10*2 + 8*(-1) + 8*(-1) + 1*(-1) = 20-8-8-1 = 3
out(0,1) = 10*(-1) + 8*2 + 1*(-1) + 8*2 + 1*(-1) + 2*(-1) = -10+16-1+16-1-2 = 18
out(0,2) = 8*(-1) + 1*2 + 2*(-1) + 1*2 + 2*(-1) + 5*(-1) = -8+2-2+2-2-5 = -13
out(0,3) = 1*(-1) + 2*2 + 2*2 + 5*(-1) = -1+4+4-5 = 2
....

$$\text{out} = \begin{bmatrix} 3 & 18 & -13 & 2 \\ 18 & -29 & -11 & 12 \\ -13 & -11 & 16 & -3 \\ 2 & 12 & -3 & -5 \end{bmatrix}$$

Activated features: From this result it is almost recognizable how this filter highlights the most significant diagonals: it increases (in an approximately homogeneous way) the pixel values of the most relevant diagonals, while decreases the intensities on the other diagonals. For example, in output there is '18' where in input was 8; there are negative values on the diagonals where in the input had '1', '2' and '3'.

4 Exercise 4

Closure on a binary image. Discuss the results.

$$\text{image} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Closure consists in applying dilation and then erosion. In this way, we may be able to close missing gaps of pixels present in the image. Dilation consists in returning, for each 3x3 area of the image, a pixel of value '1' if it appears at least a '1' in that area. Erosion does the opposite: for each 3x3 area, returns '0' if that area presents at least a '0'. I applied a symmetric padding, in order to keep the size of the output unchanged from the input (not applying the padding, the final output would have been a scalar) and to avoid the loss of information.

$$\text{imagePad} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Here the matrix after the application of dilation and the final matrix, after erosion:

$$\text{image}_{dilated} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \quad \text{image}_{eroded} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Discuss the results: Here closure isn't able to close the missing gap (the central pixel in blue) and the output matrix is almost similar to the input one. So it isn't possible to have all 1-intensity pixels on the diagonal and we can't achieve this aim neither through the padding: it is due to the fact that the erosion step will always return a zero central pixel, as the footstep is applied to a 3x3 area that contains 0-pixels in (1,1) and (3,3).

5 Exercise 5

Dilation on a gray-scale image. Discuss the results of the operation.

$$\text{image} = \begin{bmatrix} 0 & 0 & 0 & 2 & 5 \\ 0 & 0 & 0 & 2 & 2 \\ -1 & 1 & 0 & 0 & 0 \\ 1 & 5 & 1 & 0 & 0 \\ 5 & 4 & 0 & 0 & 0 \end{bmatrix}$$

Dilation consists in the conjunction between the 3x3 footprint $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ and all the portions of pixels of the image along which the filter is shifted. The value of each output pixel consists in the maximum value among the pixels in its neighbourhood.
I applied a symmetric padding to the image, in order to avoid the loss of information (and not add, as would do a constant padding, misleading information pixels).

$$\text{imagePad} = \begin{bmatrix} 0 & 0 & 0 & 0 & 2 & 5 & 5 \\ 0 & 0 & 0 & 0 & 2 & 5 & 5 \\ 0 & 0 & 0 & 0 & 2 & 2 & 2 \\ -1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 5 & 1 & 0 & 0 & 0 \\ 5 & 5 & 4 & 0 & 0 & 0 & 0 \\ 5 & 5 & 4 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Applying dilation to the padded image I obtain:

$$\text{output} = \begin{bmatrix} 0 & 0 & 2 & 5 & 5 \\ 1 & 1 & 2 & 5 & 5 \\ 5 & 5 & 5 & 2 & 2 \\ 5 & 5 & 5 & 1 & 0 \\ 5 & 5 & 5 & 1 & 0 \end{bmatrix}$$

Discuss the results: Dilation makes objects more visible, expanding the shapes contained in the input image. It is recognizable by comparing the input and the output images: the latter contains only four pixels of intensity zero, compared to the fourteen zero-intensity pixels in the input. Besides, in the output the intensities of non-zero pixels are often higher than the ones in the original image.

6 Exercise 6

Upsample the given 1D vector by a factor of 2 (resulting in a 1x8 vector) using Linear Interpolation

$$\text{vector} = \begin{bmatrix} 4 & 8 & 8 & 5 \end{bmatrix}$$

Linear interpolation is used in order to resize a vector, by using known data to estimate values at unknown points. In fact here we want to upsample: obtain, from a 1x4 vector, a 1x8 vector.
In order to do that, we see our vector as a coordinate axis, where each element of the vector is one unit. So, as showed in *Figure 1*, we have 4 units: the values are centered in 0.5, 1.5, 2.5, 3.5.
As we want to double the vector, we have to split each 'cell' of the first vector in *Figure 1* in two parts and recompute all the values (the result is in the second vector in *Figure 1*), that this time will be centered in 0.25, 0.75, 1.25, etc.
In order to do this computation we rely on *distances*: to better explain this concept let's focus on the value we have to compute in correspondence of 0.75:

- $0.5<0.75<1.5$
- 0.75 has distance a=0.25 from 0.5 and has distance b=0.75 from 1.5.

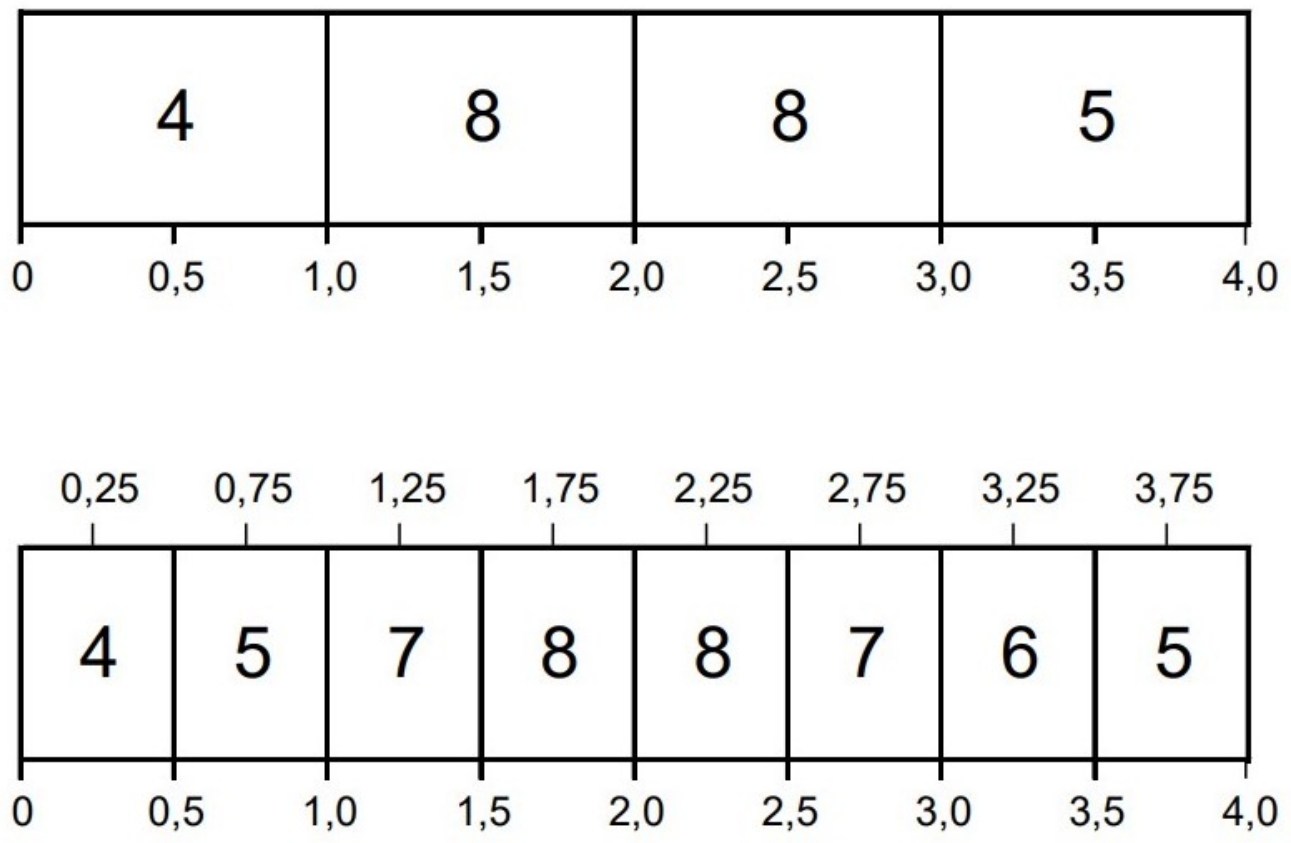

Figure 1: Exercise number 6

So, in order to compute the value correspondent to 0.75 I'll use the formula:

$$4 * (1 - a) + 8 * (1 - b) = 4 * (1 - 0.25) + 8 * (1 - 0.75) = 4 * 0.75 + 8 * 0.25 = 3 + 2 = 5 \quad (1)$$

In this way the result will be more influenced by the value 4, than 8, because 0.75 is closer to 0.5 with respect to 1.5. Doing the same reasoning it is possible to compute also the other values of the 1x8 array:

1.25: 0.5<1.25<1.5

$$4 * (1 - 0.75) + 8 * (1 - 0.25) = 4 * 0.25 + 8 * 0.75 = 1 + 6 = 7 \quad (2)$$

1.75: 1.5<1.75<2.5

$$8 * (1 - 0.25) + 8 * (1 - 0.75) = 8 * 0.75 + 8 * 0.25 = 6 + 2 = 8 \quad (3)$$

2.25: 1.5<2.25<2.5

$$8 * (1 - 0.75) + 8 * (1 - 0.25) = 8 * 0.25 + 8 * 0.75 = 2 + 6 = 8 \quad (4)$$

2.75: 2.5<2.75<3.5

$$8 * (1 - 0.25) + 5 * (1 - 0.75) = 8 * 0.75 + 5 * 0.25 = 6 + 1.25 = 7 \quad (5)$$

3.25: 2.5<3.25<3.5

$$8 * (1 - 0.75) + 5 * (1 - 0.25) = 8 * 0.25 + 5 * 0.75 = 2 + 3.75 = 6 \quad (6)$$

The values corresponding to 0.25 and 3.75 are immediate:

- **0.25**: 0<0.25<0.5. The correspondent value is 4.
- **3.75**: 3.5<3.75<4. The correspondent value is 5.

The resolution above is more intuitive. To solve the exercise, obtaining the same results, we could also apply the formula of the linear interpolation $y = y_0 \frac{x-x_1}{x_0-x_1} + y_1 \frac{x-x_0}{x_1-x_0}$, where y_i are the values we already have in the original vector, such that y will be placed between y_0 and y_1 in the new vector. x_i are the correspondent positions on the cartesian axis.

7 Exercise 7

Perform Histogram Equalization on the given image. Discuss the resulting effect.

$$\text{input image}_{4x4} = \begin{bmatrix} 10 & 8 & 6 & 6 \\ 6 & 8 & 10 & 1 \\ 8 & 1 & 3 & 5 \\ 0 & 9 & 5 & 0 \end{bmatrix}$$

The intensities of the pixels v in the *input image* are reported in the table in ascending order. For each pixel intensity I compute how many times it appears in the *input image* (count). The count of the minimum pixel intensity corresponds to the cdf_{min}=2.
Then I compute the cdf(v) ∀v. This is the cumulative distribution function, here approximated by summing the number of all the pixels with an intensity value lower or equal to v.
Finally, I apply the formula:

$$h(v) = round\left(\frac{cdf(v) - cdf_{min}}{(MxN) - cdf_{min}} x(L-1)\right) \tag{7}$$

where M = N = 4 the image dimensions, L = 256 amplitude of the range [0,255] of possible values of my image.

pixel intensity v	count	cdf(v)	h(v)	h ¬ round
0	2	2	0	0
1	2	4	36	36.4
3	1	5	55	54.64
5	2	7	91	91.07
6	3	10	146	145.71
8	3	13	200	200.35
9	1	14	219	218.57
10	2	16	255	255

Replacing the initial pixel intensities with the corresponding computed values h(v), we obtain the final image:

255	200	146	146
146	200	255	36
200	36	55	91
0	219	91	0

Discuss the results: Histogram equalization is a technique applied in order to spread the pixel intensities through the whole range of possible values [0,255]. As a result, it increases the contrast in the image. In fact, in this exercise we can see how, before applying this technique, the pixels had intensity values closer to each other; in reverse now, the intensity values go from 0 to 255.

8 Exercise 8

Discuss possible applications and advantages of using Entropy and Fourier Transform in combination on the provided images.

Figure 2: Exercise number 8. On the left "*Knight*"; on the right "*Temple*".

When working with image processing algorithms, a good practice is to use in combination two methods: a main algorithm (here entropy) to extract the features from the image and a method to improve the features I extracted.
Entropy can be exploited for different tasks, for example *image segmentation*: through entropy we can highlight the contours and the curvature of the objects. In fact entropy is applied to the original image in order to remove, or at least reduce, the noise. While applying it, depending on the radius of the 'disk' shifting all over the image, we'll get better or worst results: if the radius is too small we'll obtain more detailed information, but also noisier; if it's too large, we'll have smoothed edges but the pattern remains still clear. But, also with the optimal value of the radius, entropy introduces some smoothing effect: for this reason a good practice is to exploit Fourier transform, applying to the current image some filters that contribute to sharpen the contours: the *Ideal high pass filter*, the *Butterworth high pass filter* or the *Gaussian high pass filter* (the Gaussian one is usually preferable as it doesn't introduce a *grid effect* in the output). In this way I can sharp then lines.
Two images were provided (*Figure 2*). We can apply entropy with an appropriate radius (step A), establish a threshold (step B), in order to extract only the most relevant features, and finally apply a high pass filter (step C) to the thresholded image. Comparing the outputs of step B and C, for the '*Knight*' I expect more sharpen contours from step C rather than step B (due to the fact that the original '*Knight*' image is quite defined). On the contrary for the '*Temple*', which is more homogeneous and less defined, I expect no big improvements applying the Fourier transforms.

9 Exercise 9

Discuss two methodologies to deal with noise in order to improve the performances of the studied algorithms.

Noise can be sparsely or homogeneously distributed over the whole image. *Salt-and-pepper noise* (also known as *impulse noise*), consists in small spurious bright and black spots on the images; it can be caused by sharp and sudden disturbances in the image signal. A valid methodology to deal with this type of noise, consists in applying erotion followed by dilation. Erotion, in fact, erases the isolated points in the image. It can be applied more than once on the image, if the quantity of sparse noise requires it. But applying erotion, we have to pay attention: in fact, while removing the noise, generally the rest of the image is subjected to a significant degrade. It is due to the fact that erosion also shrinks the boundaries, erases the borders and enlarges holes. For this reason, it is necessary to apply Dilation, in order reinstate the main shapes of the image. In fact dilation tends to expand boundaries and fill holes. In that way, we'll end up with an image that has the same pattern and features of the original one, but with a big reduction of noise. The process just described is nothing more than the **Opening** morphological operation.
Another methodology to reduce noise is to apply **Blurring filters**, as they smooth the images, reducing details and noise. In order to deal with impulse noise and with *speckle noise* (a granular interference on the whole image) we can apply the *Median filter*, that runs through the signal entry by entry, replacing each entry with the median of neighboring entries. Also this filter can be applied multiple times to the image if necessary. Unlike the other smoothing techniques, such as Gaussian filter, a very good feature of Median filter is that it manages in preserving the edges. Therefore, this filter is preferred to Gaussian in case of impulse noise and small or moderate levels of Gaussian noise.

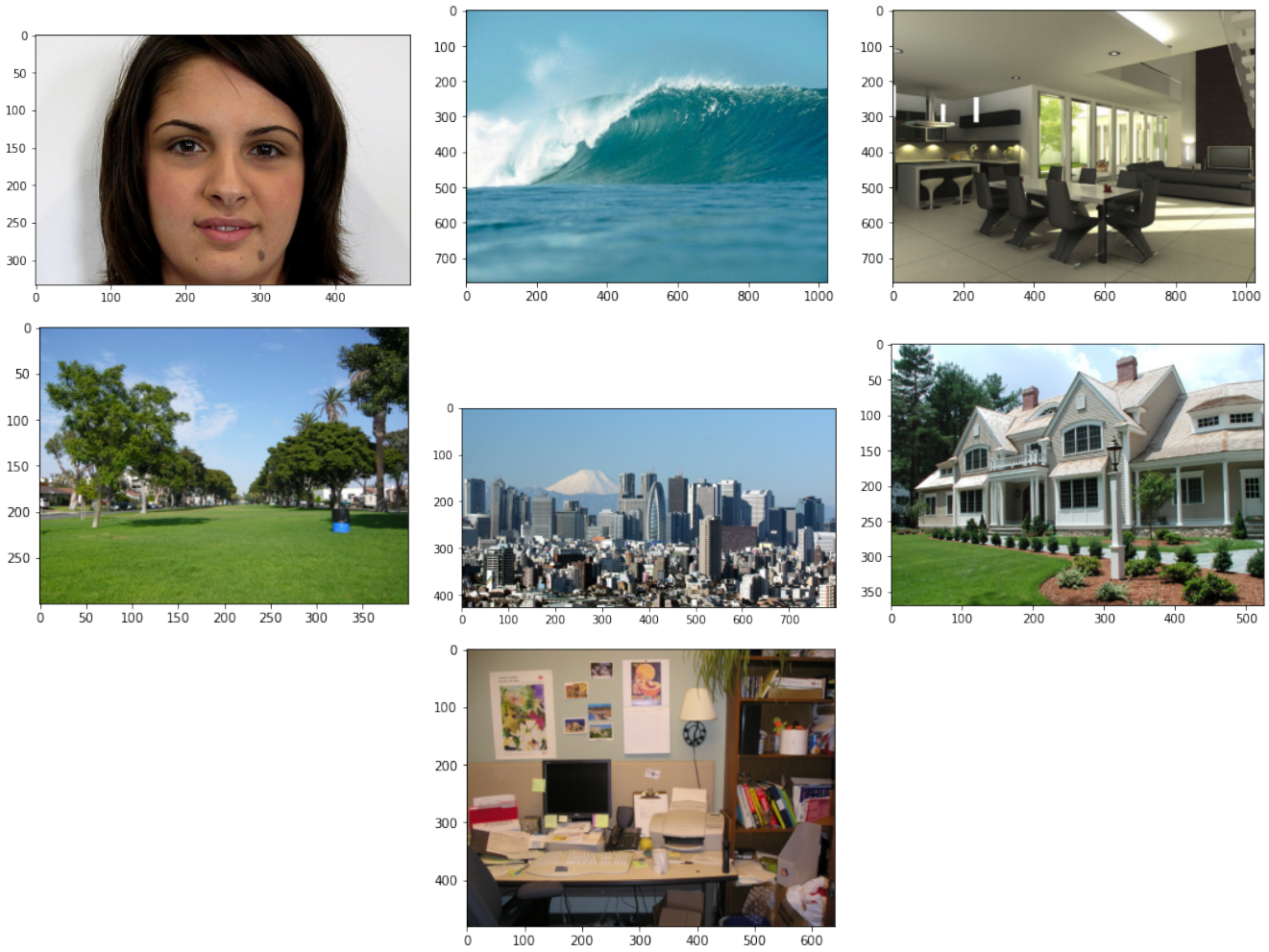

Figure 3: Exercise number 10 - Original images of the seven classes.

However, the **Gaussian blur** has better performances for high levels of noise, uniformly distributed on the image. Finally we remember (look at *exercise 8*) that entropy can be used to remove most of a noise that has a similar (/uniform) pattern all over the image: the probability of the noise is really high, so the entropy that we can extract from it is low. It's enough choose a radius large enough to understand the noise and an appropriate threshold.
All the previous methodologies can be applied for preprocessing the images, reducing the level of noise and so improving the results, for example, of a following edge-detection algorithm.

10 Exercise 10

Starting from the provided implementation of Bag of Visual Words algorithm with SIFT, implement Harris and another method at your choice (not explained during lectures). Compare SIFT, Harris and the new method. Report the features extraction phase and discuss the results. Show the extracted features using the three different methods on sample image from the dataset. Furthermore, the three confusion matrices as output of the BOW algorithm must be provided.

The first method is the Harris Corner Detector: here corners are the main feature we're interested in. In fact this method is based on the computation of image gradients over small regions: in this way, depending on the type of patch (flat, edge, corner) we'll work with different distributions of x and y derivatives. Corners are invariant to rotation, to intensity shifts and partially invariant to multiplicative intensity changes. A disadvantage of the method is that it is

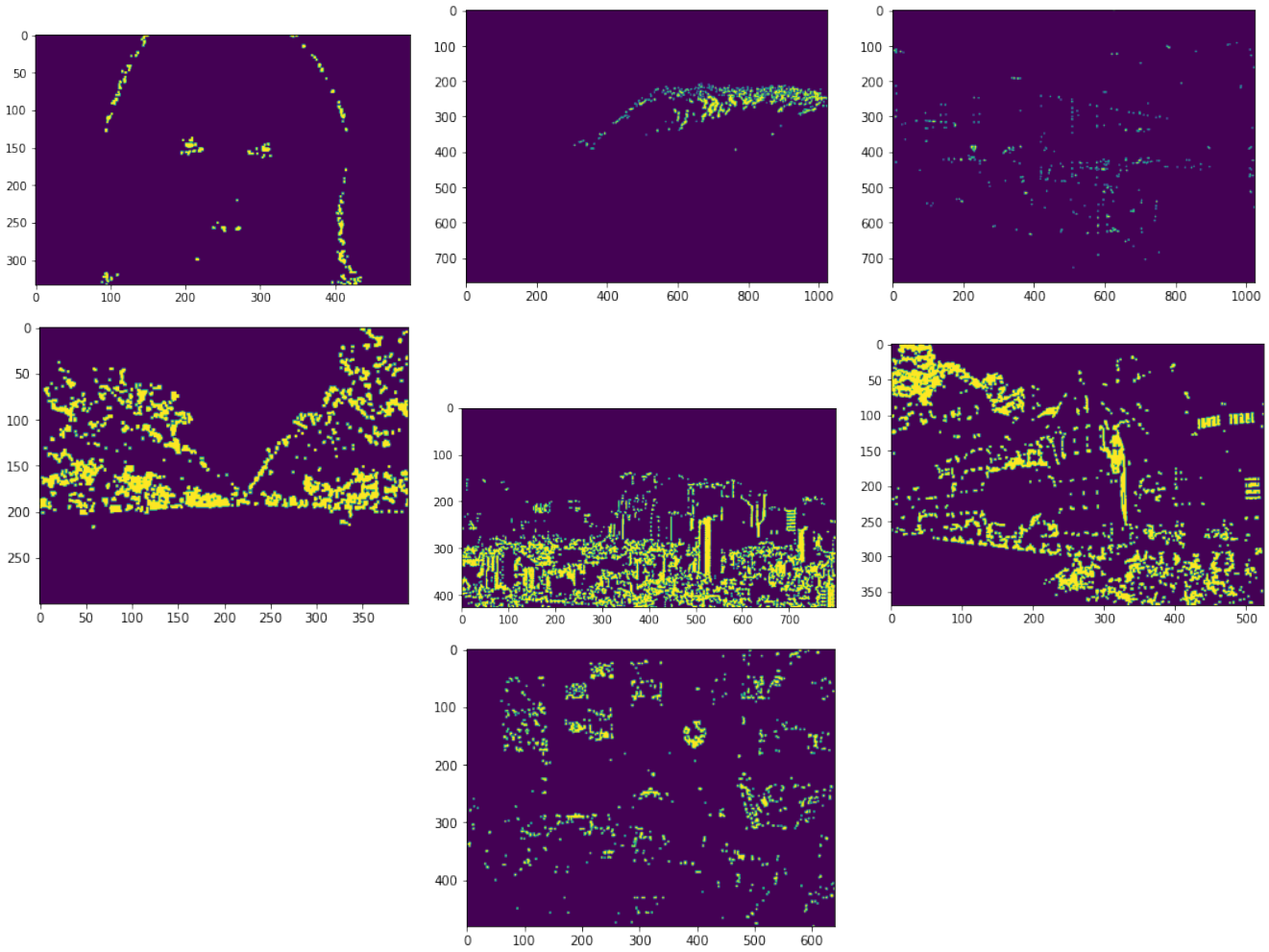

Figure 4: Exercise 10 - Features extracted through Harris on the seven classes.

not invariant to scale, as the window used to detect the keypoints on the image has the same dimension for the whole process; we have also to take into account that corners are only a small percentage of the image (e.g. it is evident from the wave image in *Figure 4*). For these reasons, the accuracy obtained on the test set is only 46.7%.
Sift manages in solving the problem of invariance with respect to scale. In fact through Log (here approximated with Difference od Gaussians), we detect blobs in various sizes due to change in σ and obtain a list of (x,y,σ) values which means there is a potential keypoint at (x,y) at σ scale. This allows to reach an accuracy of 64.1%, In *Figure 5*, we can see how the keypoints can be represented with circles with size of the keypoint and with an orientation .
The last method is Surf, a speeded-up version of Sift, that approximates LoG with Box Filter: in this way convolution is computed easily through integral images and can be done in parallel for different scales. Integral images, that make things easier, are also exploited for the orientation assignment and the feature description. Another improvement is the use of sign of Laplacian (trace of Hessian Matrix) for underlying interest point. It adds no computation cost and distinguishes bright blobs on dark backgrounds from the reverse situation: therefore during matching stage, we only compare features with the same type of contrast. The accuracy obtained is similar to the Sift case: 61.4%.
The features extracted from the training images through those methods, are used to identify the objects during the test step. The results obtained on the seven classes can be deduced by looking at the confusion matrices in *Figure 7, page 12*. We can observe that, for all the three methods, the smallest number of errors is made in correspondence of the class *face*, where the main features are concentrated around the eyes, mouth and unkempt hair. The worst result is obtained on *house indoor* samples, that are tipically confused with *office* images. Harris has appreciable performances only on the *city, face* and *sea* samples.

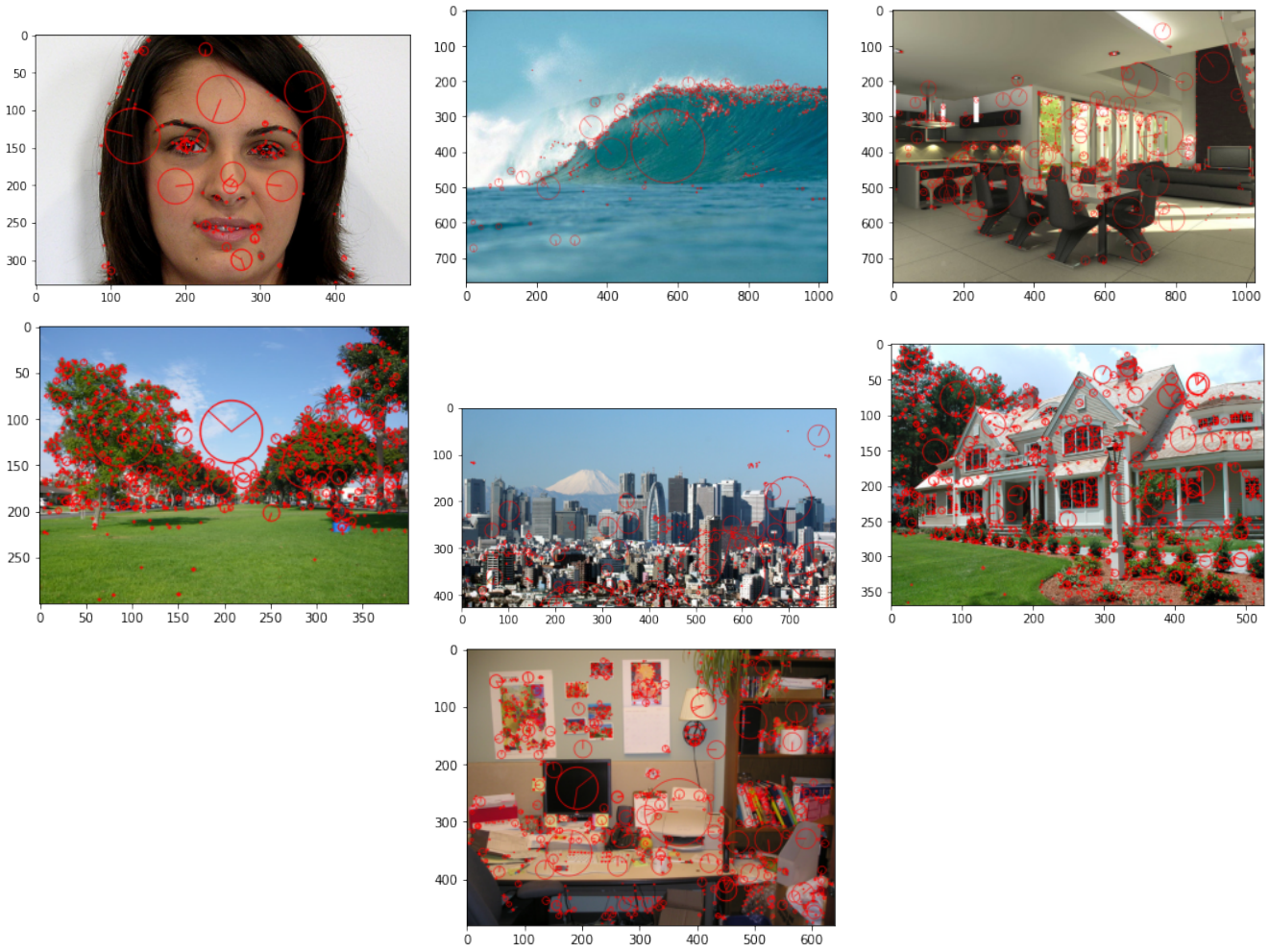

Figure 5: Exercise 10 - Features extracted through Sift on the seven classes.

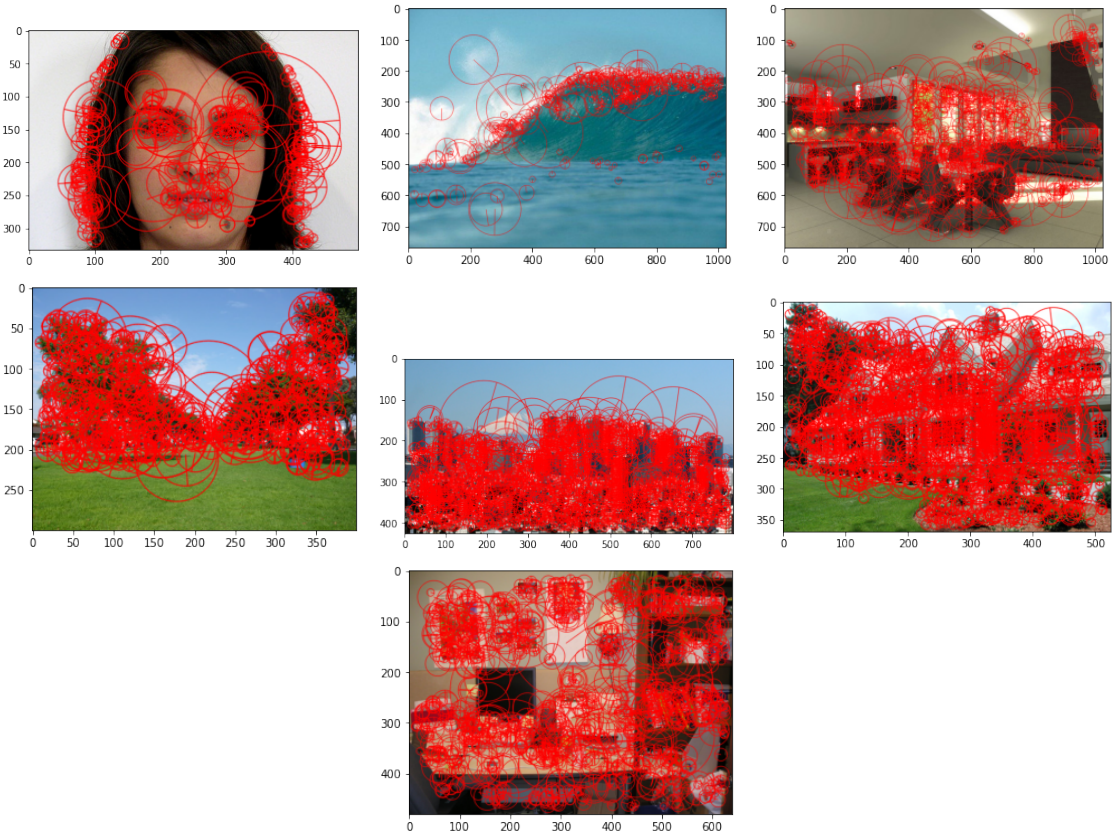

Figure 6: Exercise 10 - Features extracted through Surf on the seven classes.

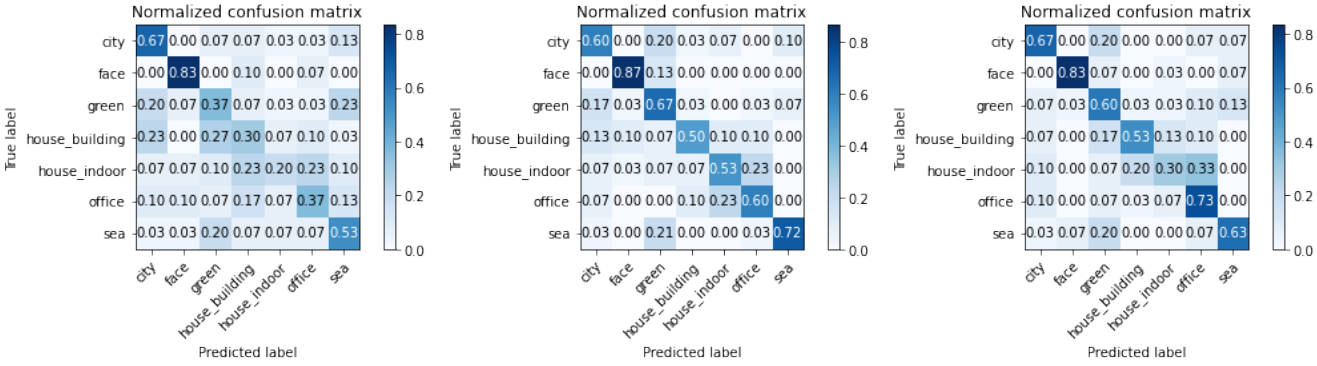

Figure 7: Exercise 10 - Normalized confusion matrices. Left: Harris, center: Sift, right: Surf.

11 Exercise 11

Write a code to segment simple shapes, showing the reasoning behind your choices. Discuss the limits and strengths of your proposed approach. Then exploiting the Optical Flow algorithm on the given images, define which object that

you segment with the previous code is moving to the center and which is diverging to the borders. Discuss and report the estimated optical flow usage.

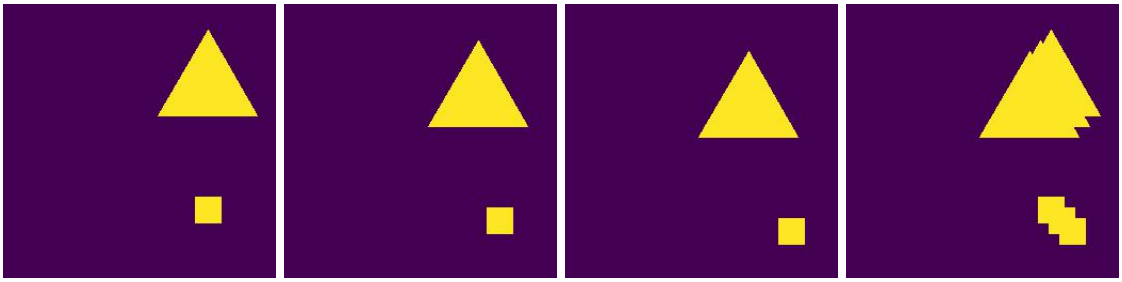

Figure 8: Exercise number 11. From left to right: fragment1, fragment2, fragment3, fragments.

The three first images given consist in a purple background, a yellow triangle and a yellow square. Being very simple images, where the main objects differentiate from the background only relying on the color, I decided to perform image segmentation using the K-Means clustering algorithm. This process results in partitioning the image into multiple distinct regions containing pixels with similarity in the data: here the color of the pixel. In this way we are able to segment the interested area (see *Figure 9 (left)*) from the background, by simply choosing the number of clusters K=2. In this way we also managed in eliminating a small noise that was located around the triangle (because noise has a lighter purple color and we choose to detect only two different colors). On the other hand, depending on the complexity of the image and with respect to which feature we want to segment, the K-means can have good or bad performances: e.g. if we wanted to segment image in *Figure 9 (right)* with respect to the shape of the objects, K-means isn't the appropriate method. In fact there are four cars of same shape but different colors, and it is evident how the K-means clustering, with K=2, isn't able to gather all the cars in one cluster: two cars will be yellow, two black.

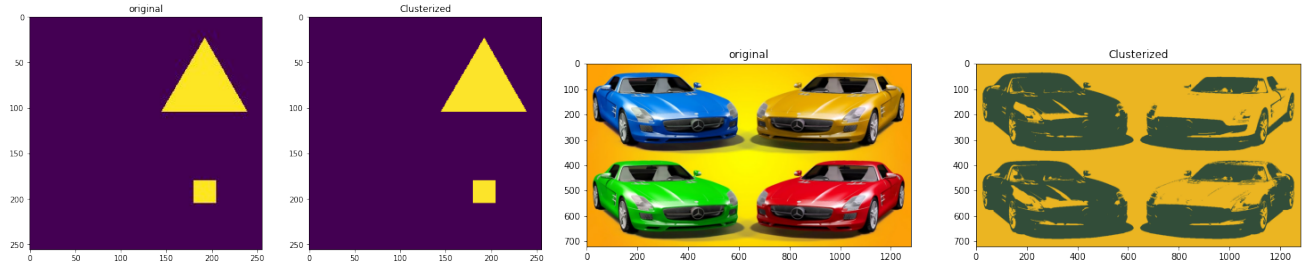

Figure 9: Exercise number 11. Left: segmentation of triangle and square. Right: segmentation of cars.

I exploited two methods for Optical flow (both applied to the segmented fragment1, fragment2, fragment3, assuming them as 3 frames from a video). With both methods we can conclude that the triangle is moving to the center and the square is diverging to the right bottom corner (see *Figure 10*). In the case of Lucas-Kanade, the colored lines show us the path followed by the two figures, while the point is in correspondence of the position occupied by the figure in the last frame. Dense optical flow shows a trail, not just of the vertices, that results in colored areas: the triangle in step 2 is is more central than in step 1; the square is more external in step 2.

12 Exercise 12

Taking two given images, compute and discuss the texture gradients. Which filters perform better w.r.t. on which effects?

In image gradients, each pixel of a gradient image measures the change in intensity of that point in the original image, in a given direction. Gradient images have been generated by the convolution of the original image with a filter. Let's start with Sobel, Prewitt, Scharr and Roberts filters. In *Figure 12-13*, the gradient images in the x and y directions and the magnitude are reported. By looking at x and y gradients, it is recognizable how horizontal filters highlight the vertical elements, while vertical filters highlight horizontal ones: it is clear by looking at *img2*, where an horizontal

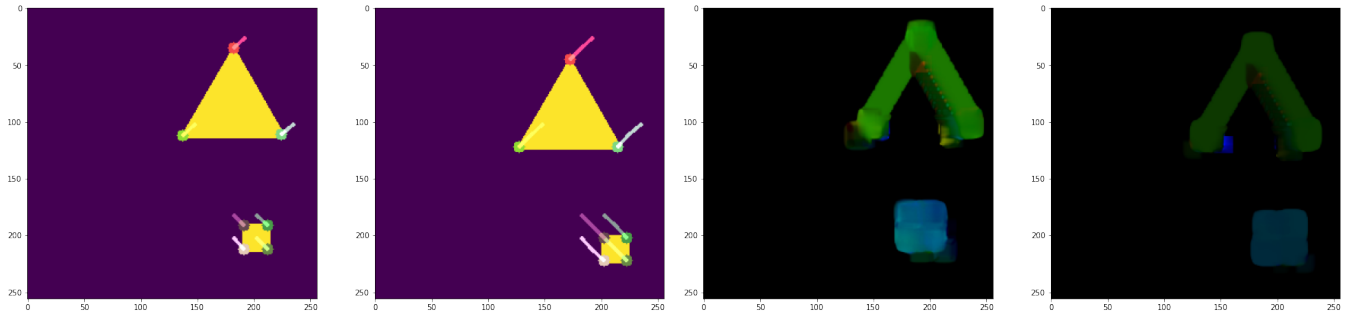

Figure 10: Exercise number 11. Left: step 1 and 2 of Lucas-Kanade. Right: step 1 and 2 of Dense.

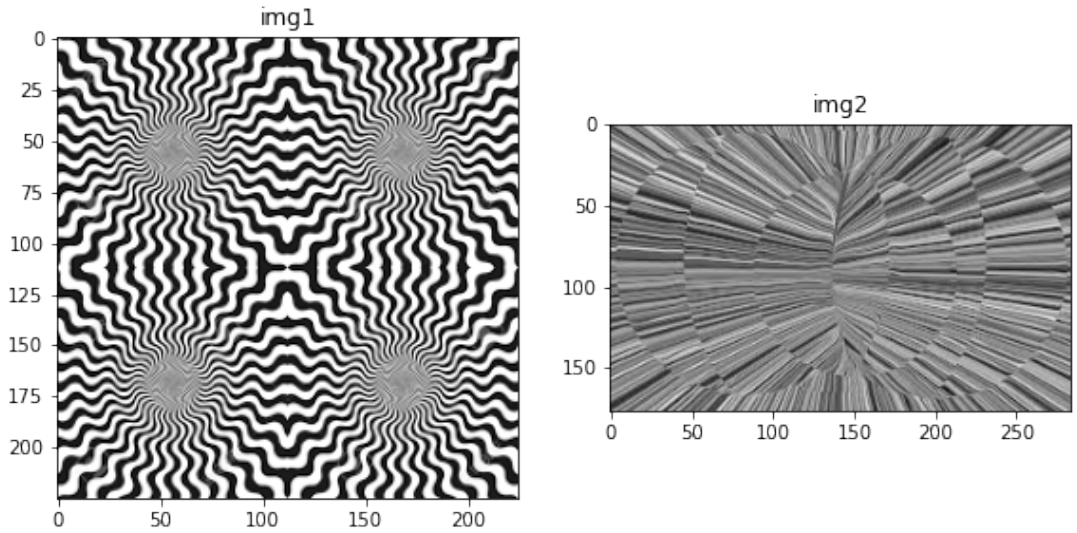

Figure 11: Exercise 12 - original images

filter isn't able to detect the lines in the central area (mostly horizontal), resulting in an almost flat and smooth effect in that region. Now let's concentrate on magnitude, from which we can make deductions on the performances of these filters in edge detection. By focusing on *img1* it is easy to conclude that the Sobel filter introduces a lot of noise: its result is brighter than the others, as the edges computed are thick, cutting off a lot of information. Prewitt and Scharr have a similar behaviour: edges are almost thin, but they introduce some noise in the area around the darker center of the image. The best result is achieved through the Roberts filter, with thinner and definite contours (robertX and robertY highlight more the elements on one of the diagonals). Similar considerations can be done about *img2* that, as the central area of *img1*, is more difficult to analyse, because characterised by very thin shapes, in different gray scales.
Now, let's observe the performances of the Laplacian and Log filters (*Figure 14*). With both, the edges are more defined (with respect to the previous filters), but we have a flatten effect: the depth effect has disappeared. Then, the outcomes of Log present higher contrast, making more clear the differences in intensities.

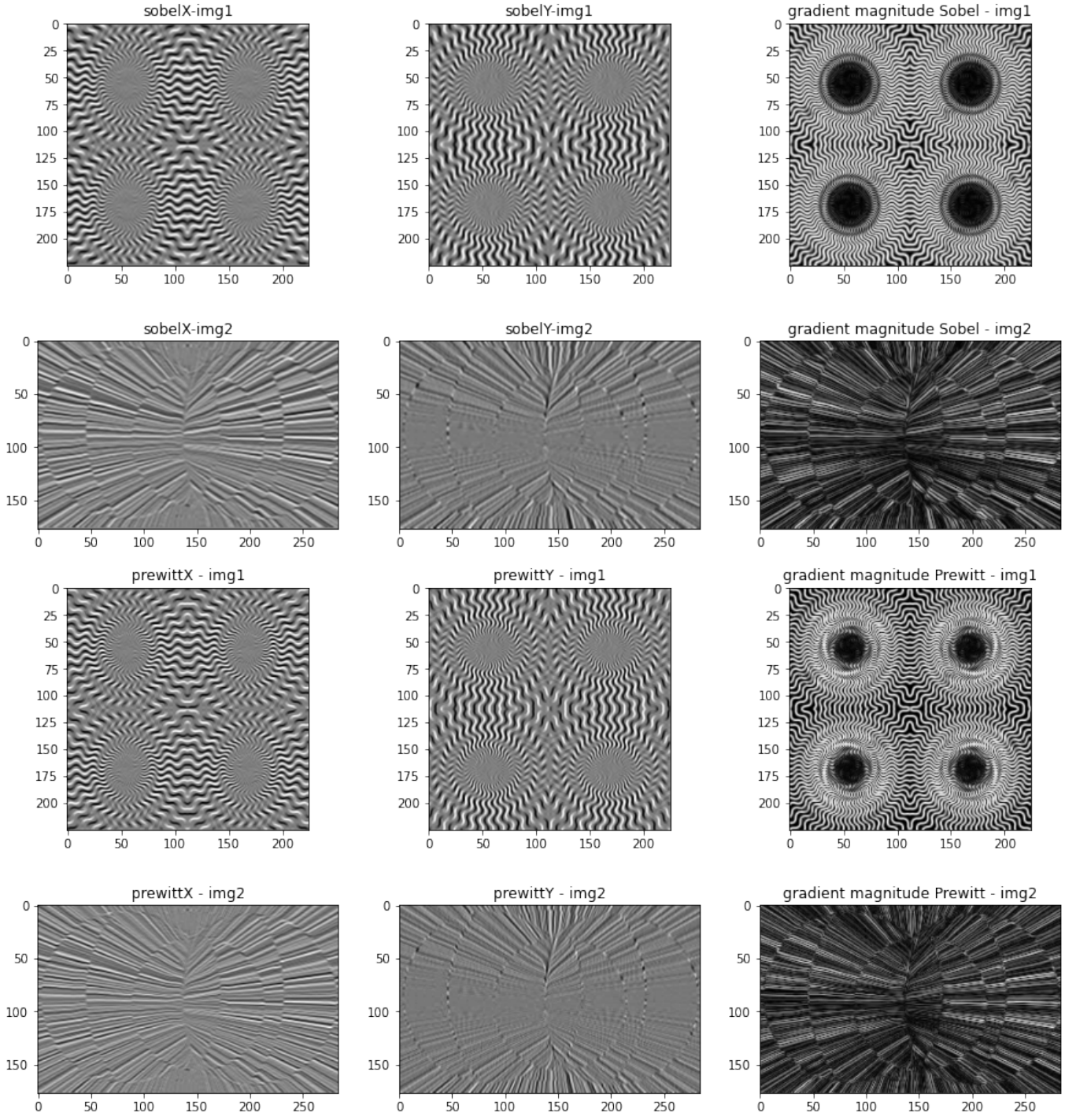

Figure 12: Exercise 12 - Sobel and Prewitt filters.

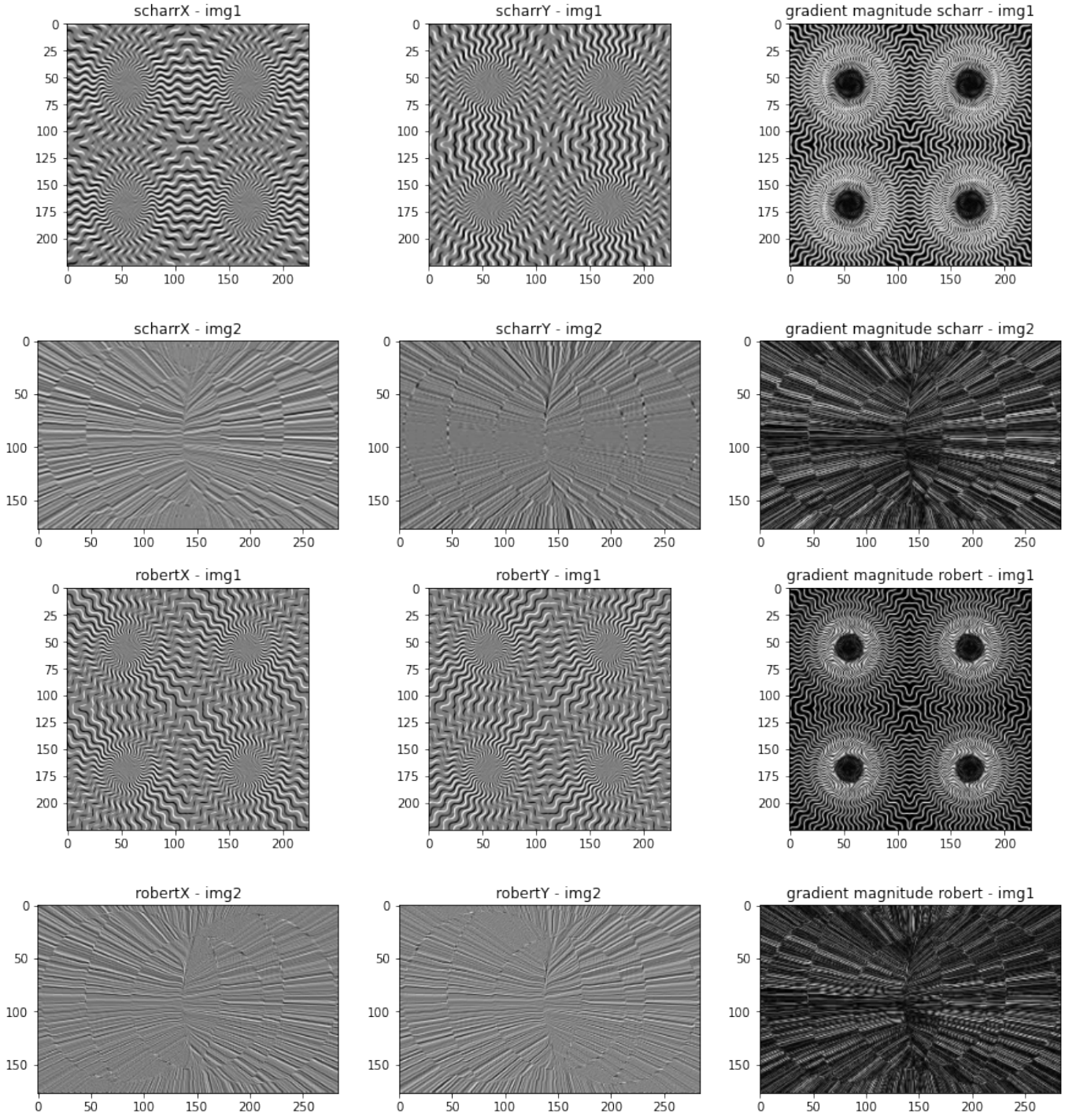

Figure 13: Exercise 12 - Scharr and Roberts filters.

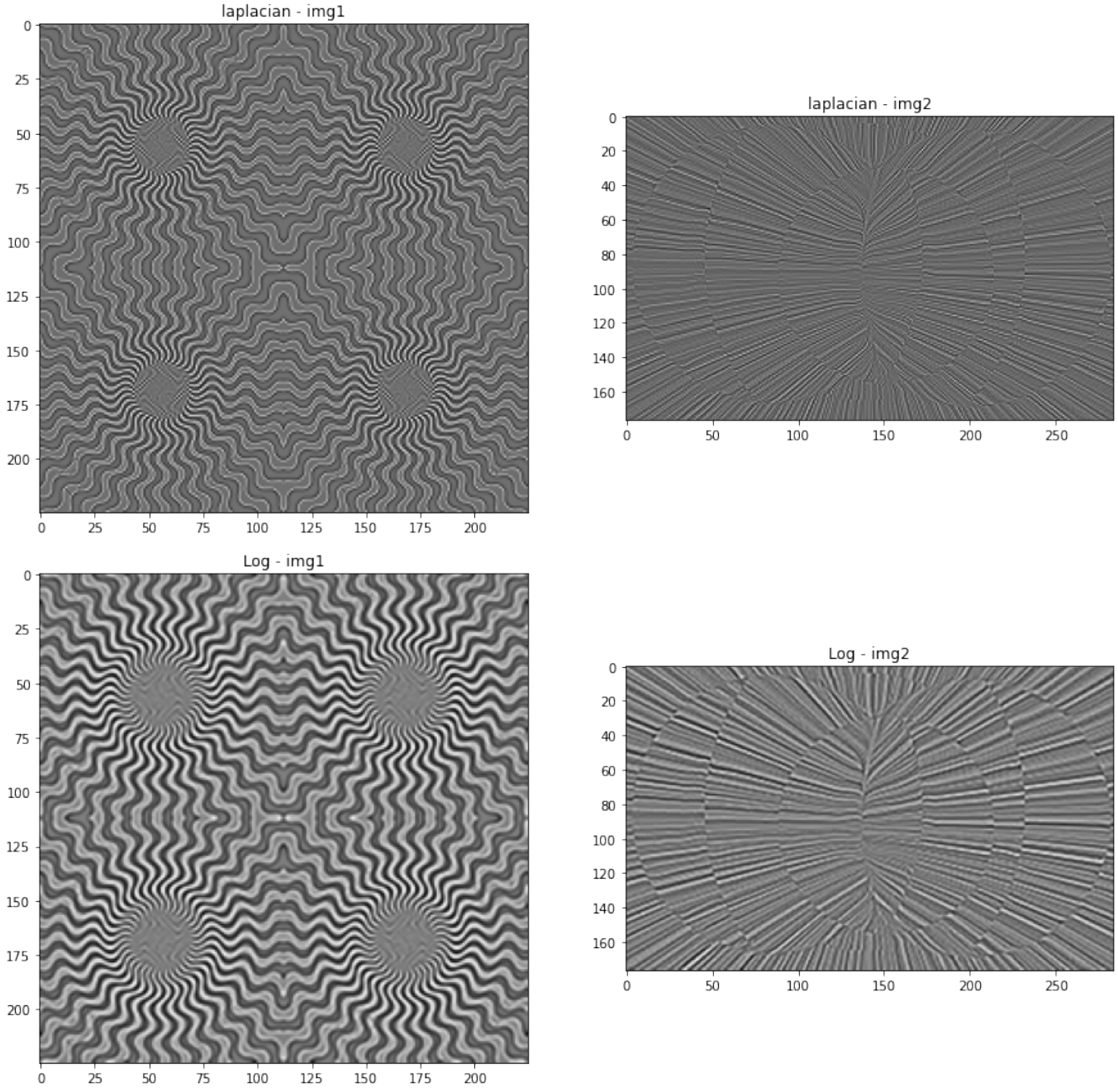

Figure 14: Exercise 12 - Laplacian and Laplacian of Gaussian.

13 Exercise 13

Discuss and implement at least two algorithms to decrease the size of an image with a minimum content loss and discuss each. How can you estimate the content loss before applying your functions?

Image downsampling consists in resizing a digital image, by discarding some of the rows and/or columns from the original image. It results in a reduction in the dimensionality of the data, with a loss of information that we want to minimize. For all the algorithms presented, starting from a RGB 378x650 image, I performed 8 downsamplings, obtaining a 3x6 outcome image at the end of the process.
Downsampling by simply removing half of the columns and half of the rows (Naive), will result in a big content loss and often in aliasing: high frequency changes (such as alternating light/dark bands) will convert to low frequencies (such as constant light or dark). Only as demonstration, I implemented the **Naive downsampling**: by looking at *Figure 16* is easy to deduce that this method has to be avoided, as the results obtained are really bad, as we lost a lot of information. By doing a comparison with the following results, it is evident how Naive has different outcomes, while all the next results are pretty similar.
In order to avoid the previous big amount of loss, I decided to implement the **Box filter**. In fact a good practice consists in applying an average filter before downscaling. We obtain better smaller scale images, because of the smoothed effect introduced by the use of this filter. The outcomes can be seen in *Figure 17*.
A better reduction in scale can be implemented through the **Gaussian Pyramids**. It consists in a loop where, at each step, we have the image of a certain dimension in input: we smooth it by applying a Gaussian filter (as the openCV preimplemented functions use a 5x5 kernel, also I used a 5x5 Gaussian filter) and we downsample by removing half of the rows and half of the columns. As recognizable from *Figure 18*. Comparing these outcomes with the ones of the Box filter downsampling, we can observe that the current implementation causes a smaller loss of information. This is due to the fact that the Gaussian filter doesn't weight all the pixels as the same way. For this reason Gaussian pyramids downsampling returns less smoothed scaled images, as it is recognizable by comparing the 12x21 images: the one returned by the Box method have pixels with more homogeneous colors, that makes a bit more difficult to recognize the components of the original image.
The only disadvantage of using Gaussian pyramids is that it isn't possible to reconstruct the original image, given the most downscaled one. Therefore, I implemented the **Laplacian pyramids downsampling**, which computes two pyramids: a gaussian pyramid (equal to the one obtained in *Figure 18*) and a laplacian pyramid, reported in *Figure 19*, that has a level in common with the gaussian one, while the others are the residuals obtained from the subtraction between two consecutive levels of the gaussian pyramid. So, known the residuals and the upper (= the most downsampled) level of the gaussian pyramid, it is possible to reconstruct the original image. Reconstruction is reported in *Figure 20*.

In order to get an idea of my content loss before applying the presented algorithms, a possible solution could be exploiting the Nyquist-Shannon sampling theorem: when downsampling, aliasing does not occur if the frequency of downsampling is equal to the Nyquist frequency or higher. In this way we'll avoid also big loss of information, that takes place if we use few samples.

Figure 15: Exercise 13 - Original image.

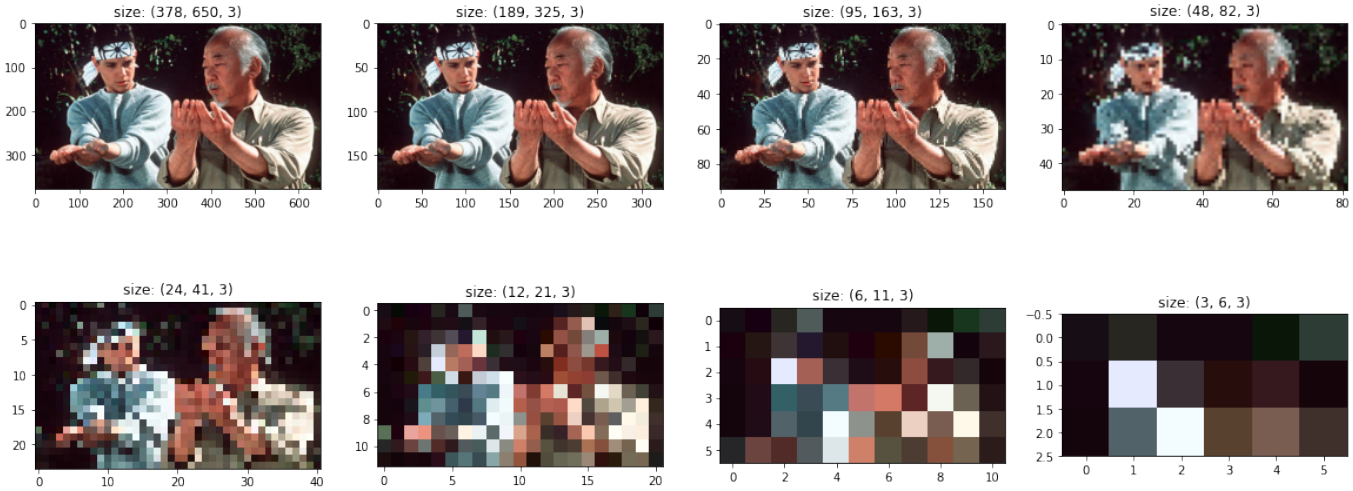

Figure 16: Exercise 13 - Naive downsampling.

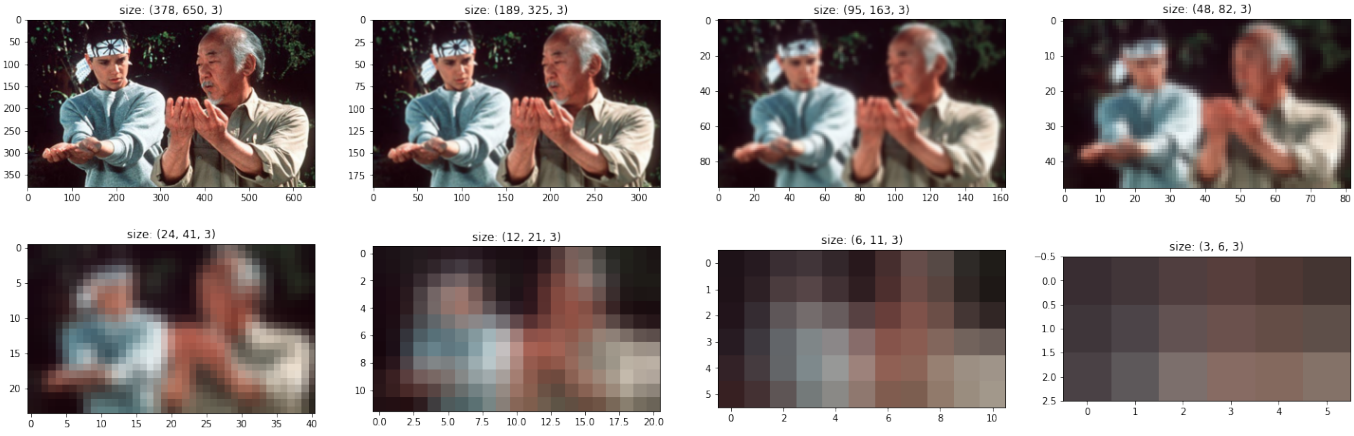

Figure 17: Exercise 13 - Box filter downsampling.

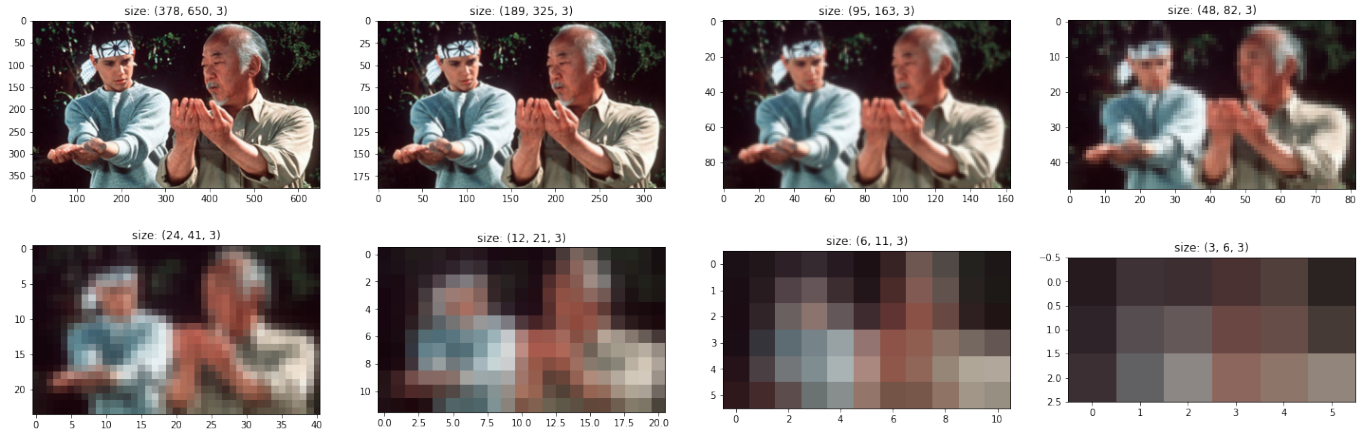

Figure 18: Exercise 13 - Gaussian pyramid - from the base to the top.

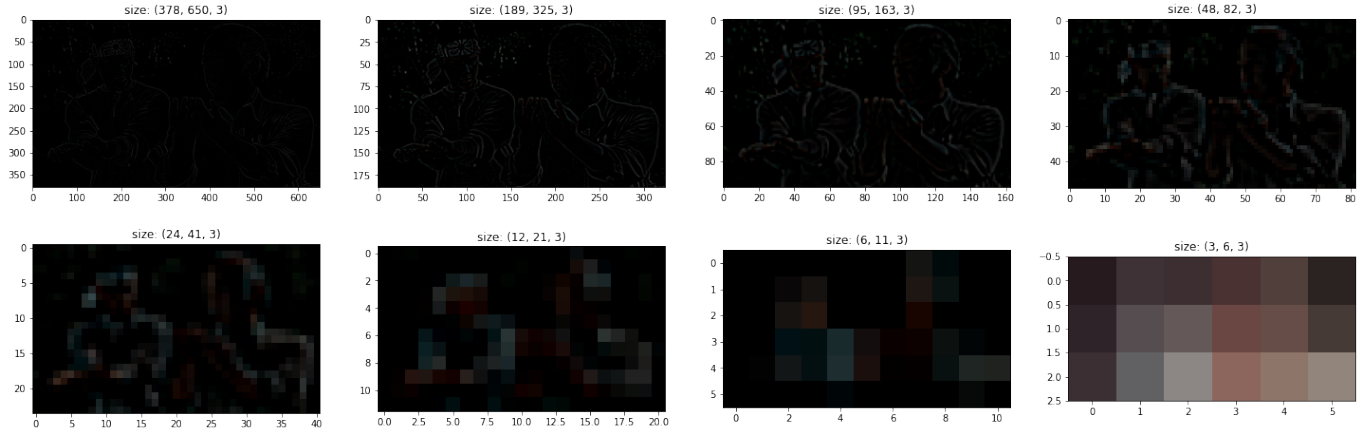

Figure 19: Exercise 13 - Laplacian pyramid (residuals) - from the last to the first level of the pyramid.

Figure 20: Exercise 13 - Laplacian pyramid - reconstruction of the original image.