Vision and Perception
Assignment 1: Image Processing

Maglianella Leandro - 1792507

April 11^{th} 2021

1 Exercise 1

Given an image I = $\begin{pmatrix} 0 & 0 & 8 & 5 \\ 4 & 2 & 2 & 1 \\ 0 & 8 & 5 & 4 \\ 2 & 2 & 1 & 0 \end{pmatrix}$ and a kernel K = $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 4 & 2 \\ 2 & 1 & 0 \end{pmatrix}$ apply symmetric padding and compute the correlation of the two coordinates (2, 2) and (3, 3) in the image.

Applying the symmetric padding I get:

$$\begin{pmatrix} 0 & 0 & 0 & 8 & 5 & 5 \\ 0 & 0 & 0 & 8 & 5 & 5 \\ 4 & 4 & 2 & 2 & 1 & 1 \\ 0 & 0 & 8 & 5 & 4 & 4 \\ 2 & 2 & 2 & 1 & 0 & 0 \\ 2 & 2 & 2 & 1 & 0 & 0 \end{pmatrix}$$

I now correlate K on the two highlighted coordinates and I obtain:

- g(2, 2) = 5*4 + 4*2 + 2*2 + 1*1 = 33
- g(3, 3) = 2*1 = 2

Symmetric padding is one of the most used padding techniques: it uses the own information of a image to pad it, therefore keeping the image's data distribution. However, the additional data included by the padding can have negative effects in certain tasks, for example in object detection where padding could make relevant some features not present in the original image.

2 Exercise 2

Compute the Gaussian filter of dimension 3 × 3 given a $\sigma = 8$.

Using the 2D Gaussian distribution $G_{\sigma=8}(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$, I will simply calculate it inserting in this equation the value of σ and, time by time, the values of i and j (remembering that the filter's center is the (0, 0) coordinate). Finally, I will normalise the resulting matrix so that the sum of its values is equal to 1. Here is the procedure (note that an approximation is needed):

- $G_{\sigma=8}(0, 0) = \frac{1}{2\pi*8^2} e^{-\frac{0}{2*8^2}} = 0.002487$
- $G_{\sigma=8}(-1, 0) = G_{\sigma=8}(0, -1) = G_{\sigma=8}(0, 1) = G_{\sigma=8}(1, 0) = \frac{1}{2\pi*8^2} e^{-\frac{1}{2*8^2}} = 0.002467$
- $G_{\sigma=8}(-1, -1) = G_{\sigma=8}(-1, 1) = G_{\sigma=8}(1, -1) = G_{\sigma=8}(1, 1) = \frac{1}{2\pi*8^2} e^{-\frac{2}{2*8^2}} = 0.002448$
- Normalisation: sum = 0.002487 + 0.002467*4 + 0.002448*4 = 0.022147
- Final result: $G = \frac{1}{sum} * \begin{pmatrix} 0.002448 & 0.002467 & 0.002448 \\ 0.002467 & 0.002487 & 0.002467 \\ 0.002448 & 0.002467 & 0.002448 \end{pmatrix} = \begin{pmatrix} 0.110534 & 0.111392 & 0.110534 \\ 0.111392 & 0.112296 & 0.111392 \\ 0.110534 & 0.111392 & 0.110534 \end{pmatrix}$

3 Exercise 3

Given a image I = $\begin{pmatrix} 2 & 2 & 5 & 2 \\ 2 & 5 & 2 & 1 \\ 5 & 2 & 1 & 7 \\ 2 & 1 & 7 & 7 \end{pmatrix}$ and an kernel K = $\begin{pmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{pmatrix}$, tell which filter is the provided one, apply it to the image, and discuss the activated features.

This kernel is a oblique line filter (+45 degrees), that it used to enhance the presence of oblique lines: in fact it provides a high response on diagonal lines. Convolving this kernel over the image, using a symmetric padding for the purpose of preserving the image dimension, I get this result:

$$\begin{pmatrix} -3 & 0 & 10 & -4 \\ 0 & 19 & -9 & -16 \\ 10 & -9 & -24 & 5 \\ -4 & -16 & 5 & 6 \end{pmatrix}$$

As we can see in the example, the operation has a highlighting effect on the oblique components on the original image: in fact it activates this feature increasing the value's difference between the three diagonals formed by 5s, 2s and 1s. This filter can be used by line detectors to help/increase their performance.
Just as an extra experimental proof of its validity, I tried applying it on Figure 1 and I obtained Figure 2 (note that horizontal and vertical lines are mostly discarded):

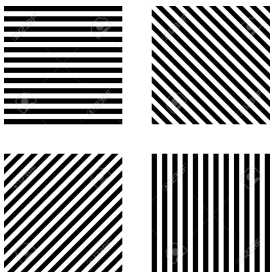

Figure 1: Exercise 3

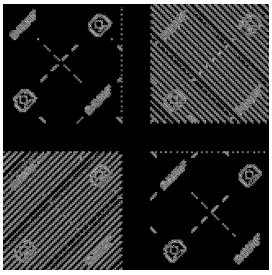

Figure 2: Exercise 3

4 Exercise 4

Compute the morphological operation of closure on the binary image I = $\begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$.

The closure operation is formed by a dilation followed by an erosion, it has the purpose of removing small imperfections/holes in images (in this case I assumed a 3 x 3 footprint for both):

- Dilation: a 0-padding is applied to maintain the image dimension and to help the dilation operation.
- Erosion: On dilation phase's result, a 1-padding is applied to maintain the image dimension and to help the erosion operation. This is the closure result.

$$Dilation\ result = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix} \quad Erosion\ result = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix} = Closure\ result$$

Finally, this closure operation was able to produce an image equal to the original one but with two recovered 1s next to the original clusters of 1s. Moreover, I would like to point out and re-emphasise the importance of padding: in fact, in this example, the whole procedure would not have worked with a different padding choice.

5 Exercise 5

Compute the morphological operation of dilation (3 x 3 footprint) on the gray-scale image I = $\begin{pmatrix} 0 & 0 & 0 & 3 & 5 \\ 0 & 0 & 1 & 5 & 1 \\ 0 & 0 & 0 & 1 & -1 \\ 2 & 3 & 0 & 0 & 0 \\ 5 & 2 & 0 & 0 & 0 \end{pmatrix}$.

A 0-padding is applied to maintain the image dimension and to help the dilation operation. The result is the following:

$$\begin{pmatrix} 0 & 1 & 5 & 5 & 5 \\ 0 & 1 & 5 & 5 & 5 \\ 3 & 3 & 5 & 5 & 5 \\ 5 & 5 & 3 & 1 & 1 \\ 5 & 5 & 3 & 0 & 0 \end{pmatrix}$$

Being a gray-scale dilation of an image, this procedure imply the allocation to each pixel of the image of the maximum value found in its neighbourhood (in this case a 3 x 3 area). This augments the shapes and features carried by the input image and it can be used, as in the previous exercise, as the beginning of a closure operation.

6 Exercise 6

Double the size of the 1D vector v = $\begin{pmatrix} 4 & 7 & 2 & 5 \end{pmatrix}$ using a Linear Interpolation.

Linear Interpolation is an upsampling technique where the new values are a weighted combination of their original pixel neighbours so that the nearest pixel have higher weights/influence on the corresponding value. Using the formula: $y = \frac{y_0(x_1-x)+y_1(x-x_0)}{x_1-x_0}$ where (x, y) are the coordinate and value of a new point, (x_0, y_0) are the coordinate and value of last original point and (x_1, y_1) are the coordinate and value of next original point. I also consider a "reflect" mode so that the first and the last element of the new vector are equal to the first and last of the original vector. In the following handwritten Figure 3, I represented graphically the old (black) and new (red) points and computed theirs values:

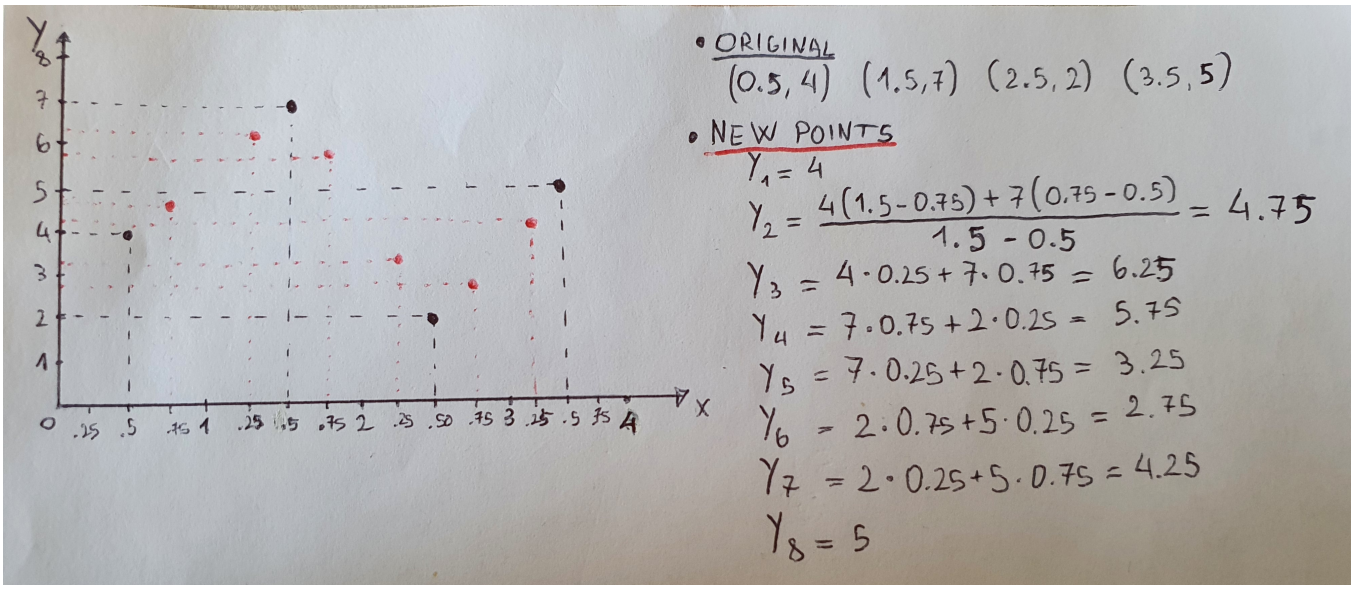

Figure 3: Exercise 6

Therefore, the new vector v' and its integer approximation are:

$$\begin{pmatrix}4 & 4.75 & 6.25 & 5.75 & 3.25 & 2.75 & 4.25 & 5\end{pmatrix} \rightarrow \begin{pmatrix}4 & 5 & 6 & 6 & 3 & 3 & 4 & 5\end{pmatrix}$$

7 Exercise 7

Perform Histogram Equalization on the image I = $\begin{pmatrix}8 & 7 & 14 & 4\\ 14 & 7 & 8 & 1\\ 7 & 9 & 2 & 5\\ 0 & 7 & 5 & 0\end{pmatrix}$.

Histogram Equalization is a contrast adjustment technique where the pixel intensities are evenly distributed in the full interval of assignable values, in our case from 0 to 255 and computed using the formula:

$$h(v) = round(\frac{cdf(v) - cdf_{min}}{M * N - cdf_{min}} * (L - 1))$$

Here v is denoting the intensity value in the input image, $cdf(v)$ is the cumulative density function of v, M and N refer to the image dimensions (in this case they are equal to 4) and L = 256.
The following table summarises all the performed computations for the given input image:

v	$count$	$cdf(v)$	$h(v)$
0	2	2	0
1	1	3	18
2	1	4	36
4	1	5	55
5	2	7	91
7	4	11	164
8	2	13	200
9	1	14	219
14	2	16	255

As we can see, $\text{cdf}_{min} = 2$ and the result image is the following:

$$\begin{pmatrix}200 & 164 & 255 & 55\\ 255 & 164 & 200 & 18\\ 164 & 219 & 36 & 91\\ 0 & 164 & 91 & 0\end{pmatrix}$$

I would like to note that the result matrix is obtained just by substituting the original values with their corresponding $h(v)$. As mentioned, the result's values are equalised across all the [0, 255] interval to increase the image's contrast, namely this procedure is used to better show small image's details.

8 Exercise 8

Discuss possible applications and advantages of using Entropy and Fourier Transform in combination on the provided images in max 15 lines.

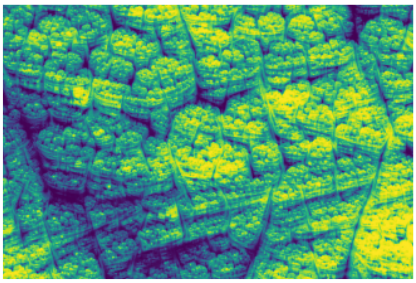

Figure 4: Exercise 8

Figure 5: Exercise 8

Figure 6: Exercise 8

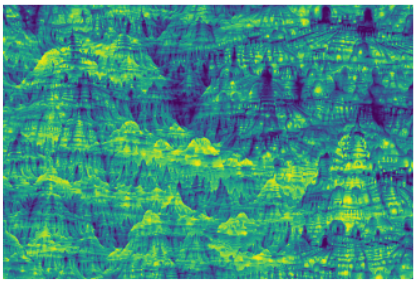

Figure 7: Exercise 8

Figure 8: Exercise 8

Figure 9: Exercise 8

Entropy and Fourier Transform (FT) are important tools used in a wide range of applications in image processing. The Entropy notion is based on the idea of measuring the uncertainties and information's complexity in a given entity. It can be used in compression algorithms or to quantitatively evaluate image's details: before the recent revival of neural networks, it was commonly used as a feature extractor as it can highlight contours and curvatures and define clusters of similar visual elements, therefore showing their properties. Moreover, Entropy not only highlight images' edges (as for example the Canny edge detector does), but also shows important information with reference to the specific performed task (for instance, colours are conserved therefore facilitating a semantic segmentation task). The FT is used to decompose an input represented in the spatial domain into its frequency domain representation, where images are invariant to geometric (translation, rotation, scale) and photometric transformations. Using the convolution theorem, in frequency domain we can implement filtering operations more efficiently as multiplications: for instance, blurring can be performed with a high pass filter and an edge detection with a low pass filter. Just as a support to this theoretical discussion, I applied the Entropy operation to the starting images (Figures 4 and 7), obtaining Figures 5 and 8. As can be seen, the process derives clusters of visual elements from the images but Entropy alone cannot describe them perfectly because the images are too detailed. Therefore, I combined FT and Entropy: I smoothed the images with a high pass filter to simplify the images and then used Entropy to obtain Figures 6 and 9. In this way better results are achieved.

9 Exercise 9

Discuss two methodologies to deal with noise in order to improve the performances of the studied algorithms (e.g. Sobel, downsampling and so on) in max 20 lines.

Despite the technological advancement of image systems' (for instance digital cameras) sensors, it is often impossible to produce completely noise-free images. This is due to the intrinsic and necessary transposition from a 3D world to a 2D one to produce images, carried out by the phases of the in-camera image processing pipeline: a loss of information is inevitable and this hinders a good feature extraction. One of the most used methodologies to reduce image noise is the application of a Gaussian filter, which has a smoothing effect on images: in particular this method is very commonly used in edge detection algorithms that are more sensitive to noise. For example, the Sobel filter (which computes image's derivatives with respect to x and y and forms the image gradient) is particularly used in edge detection algorithms and is intrinsically related to smoothing. In fact, let us consider for example the vertical Sobel filter S_y which captures horizontal lines; it can be seen that it is decomposable in a 1D derivative filter $\begin{pmatrix}1 & 0 & -1\end{pmatrix}$ and a 1D Gaussian filter $\begin{pmatrix}1 & 2 & 1\end{pmatrix}$: therefore the Sobel filter, due to the said noise cancelling reasons, actually find the smoothed image's gradient (Figure 12).

$$\begin{pmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{pmatrix} = \begin{pmatrix}1\\ 0\\ -1\end{pmatrix} * \begin{pmatrix}1 & 2 & 1\end{pmatrix}$$

The exact same concept of Sobel is used in the Laplacian of Gaussian filter (Figure 13), another very common edge extractor which catches edges as zero crossings: the only difference is that the second order derivatives of the smoothed image is computed here, therefore getting the divergence of the gradient. Notice how noisy applying only the Laplacian (Figure 11) is with reference to Figures 12 and 13:

Figure 10: Exercise 9

Figure 11: Exercise 9

Figure 12: Exercise 9

Figure 13: Exercise 9

In the Canny edge detector, other than the derivative-of-Gaussian approach, also another noise reduction methodology is used: double and hysteresis thresholding. Two thresholds (high and low) are chosen and each pixel evaluated as "strong edge" if its intensity value is greater than the high threshold, "no edge" if less than the low threshold or "weak edge" if between the two thresholds; subsequently the "weak edges" are included or not in the edge only if there is at least one "strong edge" in their neighbourhood. The result of this step is an image with only two pixel intensity values and is usually completely noise-free. Finally, even image pyramids can be considered a denoising methodology: in fact downsampling and reupsampling an image to its original size is able to remove high-frequency noises, though loosing some details.

10 Exercise 10

Starting from the provided implementation of Bag of Visual Words algorithm with SIFT, implement Harris corner detector and another method at your choice (not explained during lectures). Compare SIFT with Harris, and SIFT with the new method. The dataset is provided in the assignment zip. 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 (max 20 lines).

I have implemented Harris corner detector and BRISK: let us compare them with SIFT reporting their different feature extraction algorithms. Harris corner detector's extracted features are the positions of the corners inside images. This is done by shifting a small window across the image and computing the covariance matrix M and the corner detector's response R at each pixel. A threshold is applied: if a pixel's R is high enough, it is considered a corner. This method is rotation invariant because these computations are as well. It is also invariant to additive intensity changes, partially to multiplicative ones and is not scale invariant because it only works on a single scale. These limitations are overcome by SIFT: it is scale, intensity and rotation invariant. In summary, in SIFT's feature detection we find the local maxima (potential keypoints) across different scales of a Difference of Gaussian pyramid (scale invariant) by comparing each pixel to its neighbours on the same, previous and next level; of these, only the points with a sufficient intensity are accepted (intensity invariant) and are assigned an orientation. Gradient directions and locations are rotated corresponding to the orientation of the keypoints (rotation invariant). SIFT descriptors are then created using some computed orientation histograms. BRISK builds on SIFT and other methods and is comparable to SIFT's invariance characteristics. Its keypoint description is done sampling and processing local intensity gradients, finding point's patterns and determining their characteristic directions. These data are then processed and assembled into the binary BRISK descriptors. Figures 14, 15 and 16 visualise the features extracted respectively by SIFT, Harris and BRISK from a sample image: all three successfully find useful features along the edges of the face and near its characteristic points (eyes, nose, lips and ears). All of these extracted features have been used in the provided BOVW pipeline and the obtained results are described respectively by the confusion matrices in Figures 17, 18 and 19. The performances and the training time of all three methods are similar (accuracy just above 0.5 and a few minutes of training time). Concluding, the most robust method with the best accuracy/time ratio is confirmed to be SIFT.

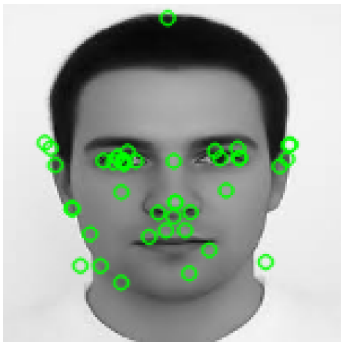

Figure 14: Exercise 10

Figure 15: Exercise 10

Figure 16: Exercise 10

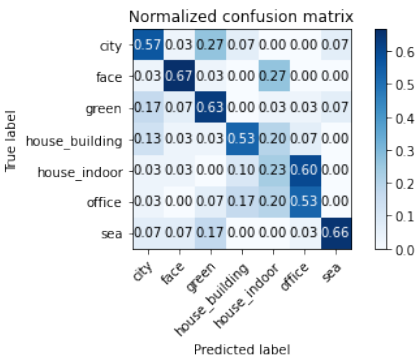

Figure 17: Exercise 10

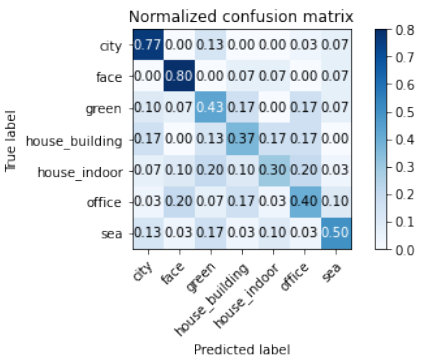

Figure 18: Exercise 10

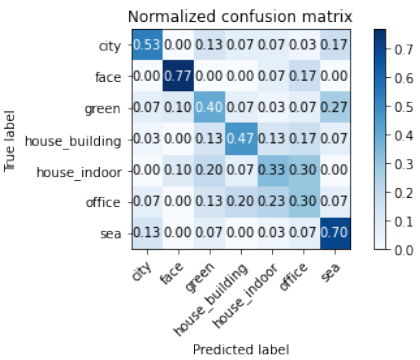

Figure 19: Exercise 10

11 Exercise 11

Write a code to segment simple shapes (the images are given) showing the reasoning behind your choices (max 5 lines). Discuss the limits and strengths of your proposed approach (max 5 lines). 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 (max 5 lines).

In Image Processing, the image segmentation process consists in dividing an image into multiple regions, each one containing only sets of pixels with similar attributes. Let us consider the starting images:

Figure 20: Exercise 11

Figure 21: Exercise 11

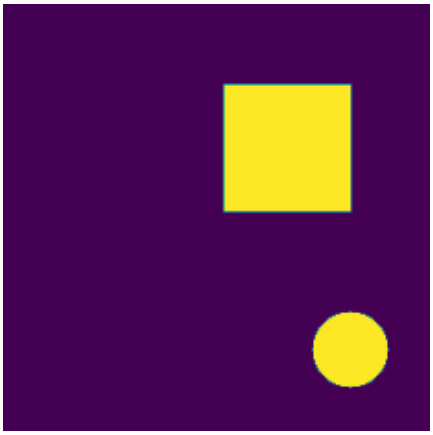

Figure 22: Exercise 11

As can be seen, these images contains particularly simple shapes, the only barely noticeable defect is a really small noise along their edges. I decided to segment them from the background just using their colour: to do so I used a K-means algorithm with the number of clusters set to two and obtained results with noise-free shapes. Other than this, my approach's strengths are making the shapes' edges sharper and being an easy and fast technique appropriate to the situation. The limitation of this segmentation is being only based on colours, therefore not being able to segment the two shapes separately (since they are both yellow), which in other applications could have been a fundamental detail. At this point I used Optical Flow to track the shifting of the shapes between the three frames. This method is widely employed to recover images' pixels motion from image brightness variations in space during time. In particular I used two types of optical flow: Lucas-Kanade and Gunnar-Farneback. In the first, using a corner detection algorithm before the optical flow, I was able to follow the movement of the shape converging towards the center: the square. Figure 23 is the result obtained. The second Dense Optical Flow was added to better visualise the displacements and to consider the circle (since the previous corner detector was rightly unable to locate corners along it). Figure 24 documents the result. Finally, I have added arrows in Figure 25 to highlight the flow: it is now clearly visible that the square is moving to the center while the circle is diverging to the borders.

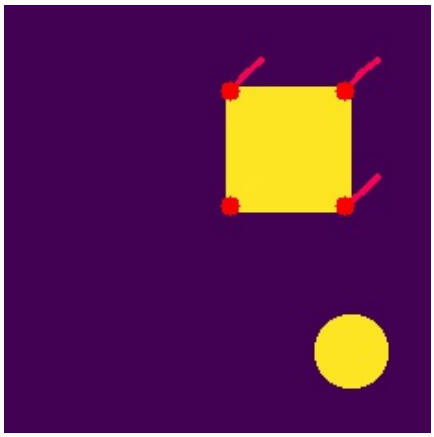

Figure 23: Exercise 11

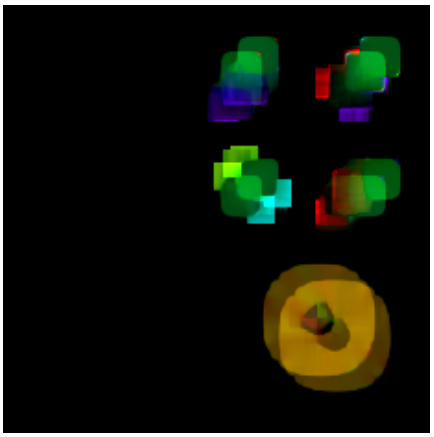

Figure 24: Exercise 11

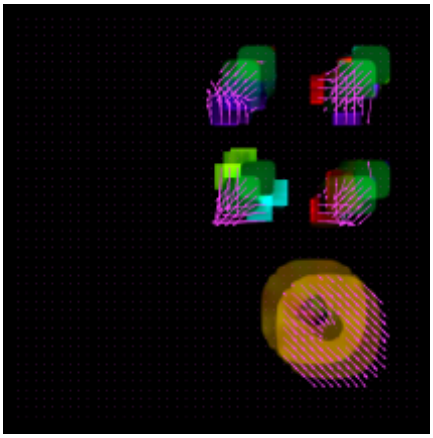

Figure 25: Exercise 11

12 Exercise 12

Taking two given images, compute and discuss the image gradients. Which filters perform better w.r.t. on which effects? (max 15 lines).

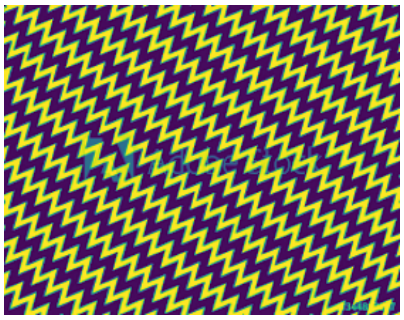

Figure 26: Exercise 12

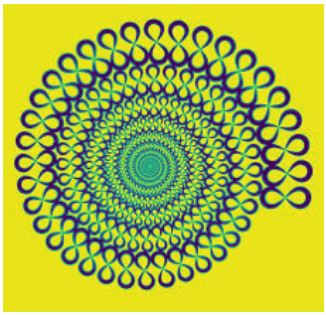

Figure 27: Exercise 12

As I began to discuss in the answer to exercise 9, in edge detection algorithms it is common to use filters that are able to find the images' gradient. The best known are (considering derivatives with respect to x and y):

- $Sobel_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$ $Sobel_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$
- $Scharr_x = \begin{pmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{pmatrix}$ $Scharr_y = \begin{pmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{pmatrix}$
- $Prewitt_x = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}$ $Prewitt_y = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}$
- $Roberts_x = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ $Roberts_y = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$

The x-derivative filters are called "horizontal filters" and captures vertical lines in images, while y-derivative filters are called "vertical filters" and captures horizontal lines. In particular, the operators of Sobel and Scharr act by smoothing the image and calculating its derivatives, while the Prewitt one does a pixels averaging and derivatives computation. Once that the derivatives D_x and D_y with respect to x and y have been determined, the gradient's amplitude G is well approximated by the formula $\sqrt{D_x^2 + D_y^2}$. I have computed these four gradients on the starting images and obtained the following results (respectively, Gradient with Sobel, Scharr, Prewitt and Roberts):

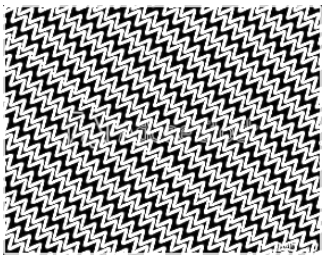

Figure 28: Exercise 12

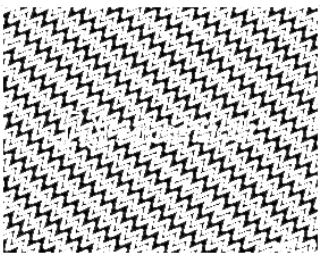

Figure 29: Exercise 12

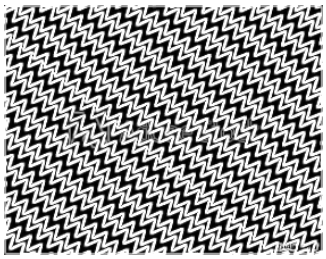

Figure 30: Exercise 12

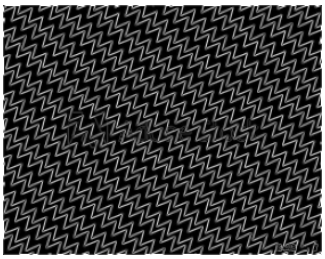

Figure 31: Exercise 12

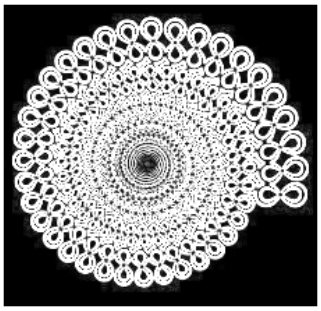

Figure 32: Exercise 12

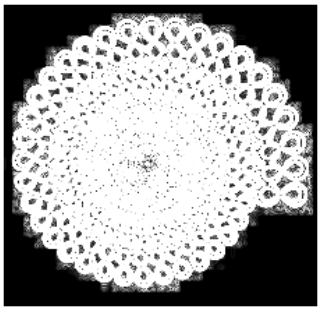

Figure 33: Exercise 12

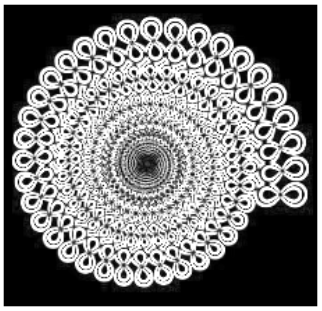

Figure 34: Exercise 12

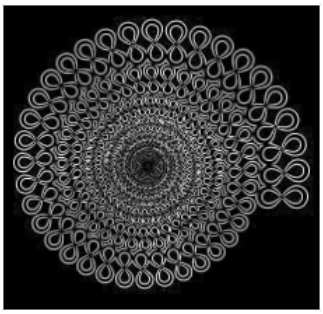

Figure 35: Exercise 12

For both sets of images, the gradients computed by Sobel and Prewitt are very similar, while in Scharr's ones the edges appears extremely marked (because of the higher smoothing values in the filter) and in Robert's ones they are the least marked. As for the first image, I'd say the best performing filter is Sobel (or the very similar Prewitt). The image does not have very small details, so the Scharr gradient is still acceptable; the result of Roberts is good as well, however its low intensity could make it less effective than the others in edge detection tasks. Similarly, for the second image the best filters are Sobel and Prewitt, and Roberts has a low intensity. The difference to note here is in Scharr's result: the center of the image has too many small details that fail to be preserved by the smoothing.

13 Exercise 13

Discuss and implement at least two algorithms to decrease the size of an image with a minimum content loss and discuss in max 8 lines each. How can you estimate the content loss before applying your functions? (max 5 lines).

I have implemented Naive downsampling, a box-smoothing+downsampling and Gaussian and Laplacian Pyramids and tested them on Figure 36: let us discuss the results. The Naive down sampling is the worst one, it is just done by deleting half rows and columns: a lot of image content/information is lost and an aliasing effect can form in the result (Figure 37). To improve this result and lose less content, I used the method of smoothing images before downsampling them, for instance using a box filter $= \frac{1}{9}\begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{pmatrix}$ (figure 38).

Figure 36: Exercise 13

Figure 37: Exercise 13

Figure 38: Exercise 13

An even better technique to lose the least amount of content is to employ a Gaussian filter combined with a multi scale downsampling: this is the main method adopted in the two algorithms of Gaussian and Laplacian Pyramids. In the first type of pyramid, a Gaussian filter and downsampling are applied to an image repeatedly: each one of these iterations form a layer of the pyramid (Figures 39 to 41, notice that these are the second, third and forth layers because the first one is actually the original image Figure 36).

Figure 39: Exercise 13

Figure 40: Exercise 13

Figure 41: Exercise 13

Instead, in Laplacian Pyramids are stored only the residuals, one for each layer and computed as $residual_i = smoothed_{\text{i-1}} - smoothed_i$, and the smallest smoothed image (Figures 42 to 44).

Figure 42: Exercise 13

Figure 43: Exercise 13

Figure 44: Exercise 13

The peculiarity of the Laplacian pyramids is that each level of the pyramid can be obtained by adding between them the levels above it (those smaller than it), upsampling when necessary. These characteristics of image pyramids make them a very important tool in various fields such as image compression, denoising, image blending and multi-scale detection and registration. To now estimate the loss of content before applying one of the said downsampling algorithms, assuming that downsampling an image inevitably involves a loss, it is possible to consider the Nyquist-Shannon sampling theorem. This theorem is the theoretical basis for the techniques described so far: it states that each function can be downsampled while still remaining completely determined if the sampling operation is done with a frequency f equal to at least twice the maximum frequency f_m contained in the function. But, considering that every time that a Gaussian smoothing is applied the f_m of the image decreases, the use of this filter always assures us that the loss of content in the image will be minimal.