

# Digital Image Processing - Lab Session 6

Sofia Noemi Crobeddu  
student number: 032410554

International Master of Biometrics and Intelligent Vision  
Université Paris-Est Créteil (UPEC)



December 22, 2024

# Contents

Introduction	3
1 Operators	3
2 Gradient operator	5
3 Spatial domain filtering	7

# Introduction

In this project we analyze different types and methods to filter an image in the context of image processing. In particular, we deepen different versions of Sobel and Prewitt kernels, gradients of an image and spatial domain filters.

## 1 Operators

In the first part of the task, we applied two filters on  $T$  image: **Sobel** and **Prewitt**. The functions used were *sobel\_v* and *prewitt\_v* for vertical gradients, to highlight horizontal intensity changes in the image. They are two horizontal edge-enhancing operators, and they were applied using *skimage.filters* library. Moreover, after the application of the filters in their default horizontal edge-emphasizing configuration, we created **Sobel kernels** adapted to all four primary directions: North, South, West, and East, using the transpose operator and *rot90* function.

The code to implement the process is shown below, and the resulting images are in Figure 1. As we can see, Sobel filter is less sensitive to noise compared to Prewitt, since it includes Gaussian smoothing.

```
1 import numpy as np
2 from PIL import Image
3 from skimage import filters
4
5 #Load image
6 image_T = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\assignment 6
    - IP\\IP6\\IP6\\T.png").convert('L')
7
8 #To array
9 image_array_T = np.array(image_T)
10
11 #Application of Sobel e Prewitt filters for horizontal labels
12 sobel_horiz_T = filters.sobel_v(image_array_T)
13 prewitt_horiz_T = filters.prewitt_v(image_array_T)
14
15 #Defining kernels for Sobel in the 4 directions
16 sobel_kernels = {
17     "North": filters.sobel_v(image_array_T),
18     "South": np.flip(filters.sobel_v(image_array_T)),
19     "West": np.rot90(filters.sobel_h(image_array_T), k=1),
20     "East": np.rot90(filters.sobel_h(image_array_T), k=-1)
21 }
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```

35 for i, (direction, result) in enumerate(sobel_kernels.items()):
36     axes[i].imshow(result, cmap='gray')
37     axes[i].set_title(f"Sobel {direction} for T image.")
38     axes[i].axis('off')
39
40 plt.tight_layout()
41 plt.show()

```

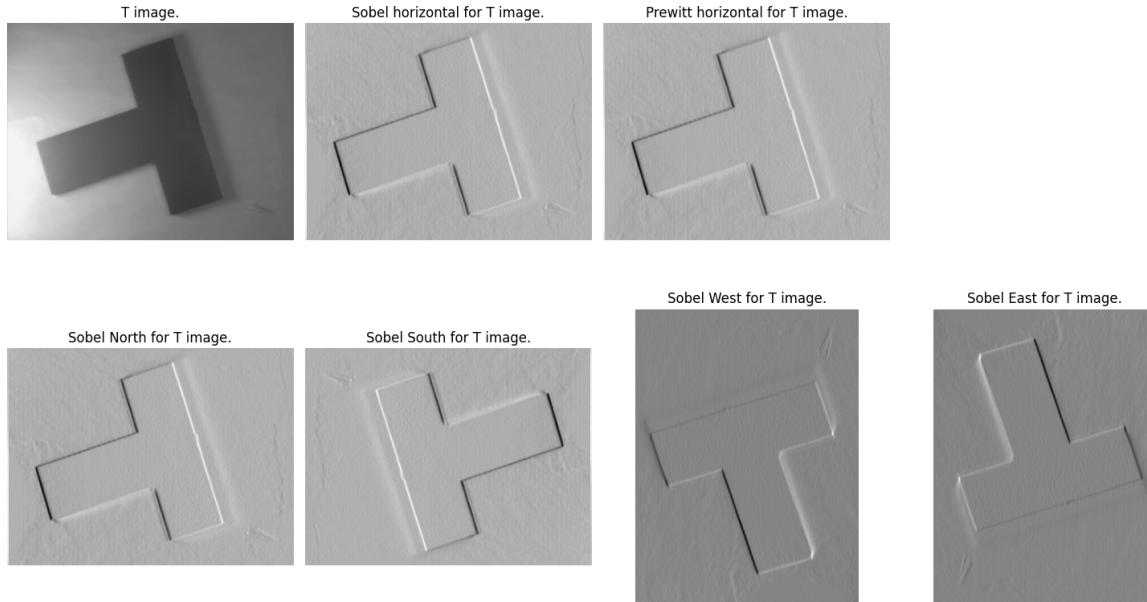


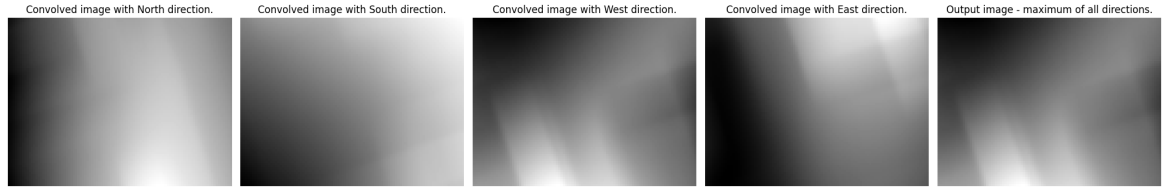
Fig. 1: Sobel with different directions and Prewitt filter on  $T$  image.

In the second part of the task  $T$  image was convolved with *Sobel kernels* in all the four directions, found before. This problem was addressed using the `mode='nearest'` option, that replicates edge values. The aim was to compute the maximum value at each pixel across all directions to produce the final output image. Moreover, due to memory constraints,  $T$  image was resized to half its original dimensions. The maximum value was calculated at each pixel across all directional convolution results using `reduce` function. The code is shown below and the resulting images are in Figure 2.

```

1 from scipy.ndimage import convolve
2 from skimage.transform import resize
3
4 #Resizing due to memory restrictions
5 image_array_T = resize(image_array_T, (image_array_T.shape[0] // 2, image_array_T.shape[1] // 2))
6 #Application of the convolution for each direction
7 convolved_img_near = {}
8 for direction, kernel in sobel_kernels.items():
9     convolved_img_near[direction] = convolve(image_array_T, kernel, mode='nearest')
10
11 #Combining results taking the maximum value on each pixel
12 output_img = np.maximum.reduce([img for img in convolved_img_near.values()])
13
14 #Plot
15 fig, axes = plt.subplots(1, 5, figsize=(20, 10))
16 #Convolution results
17 for i, (direction, result) in enumerate(convolved_img_near.items()):
18     axes[i].imshow(result, cmap='gray')
19     axes[i].set_title(f"Convolved image with {direction} direction.")
20     axes[i].axis('off')
21
22 axes[4].imshow(output_img, cmap='gray')
23 axes[4].set_title("Output image - maximum of all directions.")
24 axes[4].axis('off')
25
26 plt.tight_layout()
27 plt.show()

```



**Fig. 2:** Convolution of  $T$  image with different directions.

## 2 Gradient operator

In the first part of this task, we compute the gradient magnitude defined as:

$$magnitude = \sqrt{J_x^2 + J_y^2}$$

where  $J_x$  and  $J_y$  are the partial derivatives, representing gradients along the horizontal and vertical directions respectively.

After the calculation of the magnitude formula, the result is transformed into 8-bit integer type. The resulting magnitude of  $T$  image is shown in Figure 3, displayed using a grayscale colormap.

```

1 from skimage import img_as_float
2
3 #We need the image in float format in order to have a precise calculation of the gradients
4 image_float_T = img_as_float(image_array_T)
5
6 #Calculating the partial derivatives on x (horiz.) and y (vert.)
7 Jx, Jy = np.gradient(image_float_T)
8
9 #Magnitude of the gradient
10 magnitude_T = np.sqrt(Jx**2 + Jy**2)
11
12 #Normalizing and converting into uint8 type
13 magnitude_T_uint8 = np.uint8(np.clip(magnitude_T * 255 / np.max(magnitude_T), 0, 255))
14
15 #Plot
16 plt.figure(figsize=(6, 6))
17 plt.imshow(magnitude_T_uint8, cmap='gray')
18 plt.axis('off')
19 plt.show()

```



**Fig. 3:** Magnitude of the gradient of  $T$  image.

Continuing with the task, the objective here was to extract edges from  $T$  image. *edge* function is actually from MatLab, but we can find an equivalent one in Python from *cv2* library, i.e. *Canny* function. In order to apply it, we need to define two thresholds, a lower and a higher one, for edge

detection: they were defined based on intensity percentiles. More specifically, the lower was set as the 10<sup>th</sup> percentiles of the intensity values in the normalized  $T$  image, while the higher one as the 30<sup>th</sup> percentiles. These values were taken from state-of-the-art ones, i.e. from literature settings. This algorithm first applies Gaussian smoothing to reduce noise, and after computes intensity gradients. Also, only the strongest edges are retained since there is the suppression of non-maximum points. The resulting edge map is displayed in grayscale in Figure 4.

```

1 import cv2
2
3 #Since edge() is a function of MatLab, here we use Canny.
4 #Calculation of the percentiles for the intensity values in normalized image
5 low_threshold = np.percentile(image_array_T, 10) #10%
6 high_threshold = np.percentile(image_array_T, 30) #30%
7
8 #Converting the image in uint8
9 image_T_uint8 = (image_array_T * 255).astype(np.uint8)
10
11 #Parameters for the function Canny
12 edges = cv2.Canny(image_T_uint8, threshold1=low_threshold, threshold2=high_threshold)
13
14 #Plot
15 plt.figure(figsize=(6, 6))
16 plt.imshow(edges, cmap='gray')
17 plt.axis('off')
18 plt.show()

```



Fig. 4: Edges detected in  $T$  image.

The last point of this task has the aim of visualizing the gradient vectors of  $T$  image, calculated before. For a better visualization, the step size of 10 written in the task was maintained, in order to subsample the gradient vectors and to reduce the number of arrows. Moreover, it was created a meshgrid in order to define the positions of the sampled points.

The gradient vectors at sampled points were plotted over the grayscale image through *quiver* function, where:

- the direction of each vector corresponds to the gradient direction;
- the magnitude of the vector represents the intensity of the gradient.

The final image with the legend of grayscale intensity is shown in Figure 5.

```

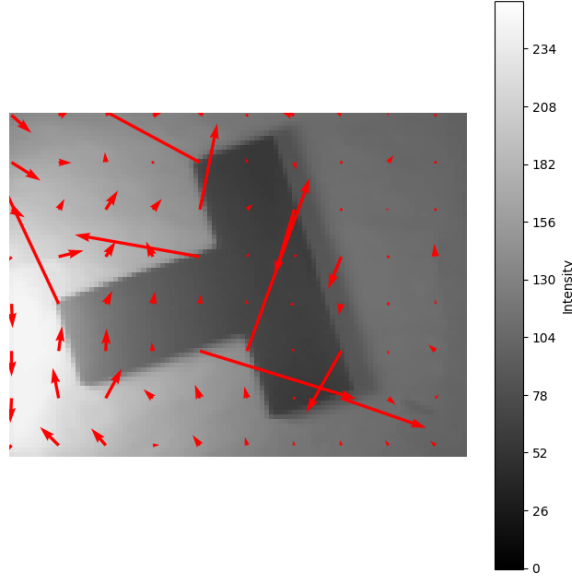
1 step=10 #Interval for the sampling (taken from the code of the task)
2
3 #Meshgrid for the points
4 x, y = np.meshgrid(np.arange(0, image_array_T.shape[1], step),
5                   np.arange(0, image_array_T.shape[0], step))
6
7 #Subsampling of the gradients
8 Jx_sub = Jx[::step, ::step]
9 Jy_sub = Jy[::step, ::step]
10

```

```

11 #Plot
12 plt.figure(figsize=(8, 8))
13 plt.imshow(image_array_T, cmap='gray', norm=NoNorm())
14 plt.colorbar(label="Intensity")
15 plt.axis('off')
16 #Adding the vectors of the gradients
17 plt.quiver(x, y, Jx_sub, -Jy_sub, color='red')
18 plt.show()

```



**Fig. 5:** Gradient vectors in  $T$  image.

### 3 Spatial domain filtering

In this task we filter  $T$  image using the convolution with three masks:

- **H1**, the **Averaging filter** for smoothing, defined as

$$H_1 = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

This filter smooths the image by averaging the intensity values of neighboring pixels. Hence, the image will have reduced noise and softened edges.

- **H2**, the **Vertical gradient filter** for edge detection, defined as

$$H_2 = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}.$$

In this filter, the positive and negative coefficients detect edges along the vertical axis. Hence, the image will have highlighted intensity changes along the vertical axis.

- **H3**, the **Laplacian filter** for enhancing edges and contours, defined as

$$H_3 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}.$$

In this filter, the matrix is designed in order to highlight all edges by emphasizing intensity changes in all directions.

For the convolution between the filter and the image, it was used *convolve2d* function. The resulting images are shown in Figure 6.

```

1 from scipy.signal import convolve2d
2
3 #Defining the three filters
4 #Averaging filter
5 H1 = (1/9)*np.array([[1, 1, 1],
6                      [1, 1, 1],
7                      [1, 1, 1]])
8 #Vertical gradient
9 H2 = np.array([[1, 0, -1],
10              [1, 0, -1],
11              [1, 0, -1]])
12 #Laplacian filter
13 H3 = np.array([[0, -1, 0],
14              [-1, 4, -1],
15              [0, -1, 0]])
16
17 #Applying filters using convolution
18 conv_H1 = convolve2d(image_array_T, H1, mode='same', boundary='symm')
19 conv_H2 = convolve2d(image_array_T, H2, mode='same', boundary='symm')
20 conv_H3 = convolve2d(image_array_T, H3, mode='same', boundary='symm')
21
22 #Plot
23 fig, axs = plt.subplots(1, 4, figsize=(20, 10))
24
25 axs[0].imshow(image_array_T, cmap='gray', norm=NoNorm())
26 axs[0].set_title("T image.")
27 axs[0].axis('off')
28
29 axs[1].imshow(conv_H1, cmap='gray')
30 axs[1].set_title("T image filtered with Averaging.")
31 axs[1].axis('off')
32
33 axs[2].imshow(conv_H2, cmap='gray')
34 axs[2].set_title("T image filtered with Vertical gradient.")
35 axs[2].axis('off')
36
37 axs[3].imshow(conv_H3, cmap='gray')
38 axs[3].set_title("T image filtered with Laplacian.")
39 axs[3].axis('off')
40
41 plt.tight_layout()
42 plt.show()

```

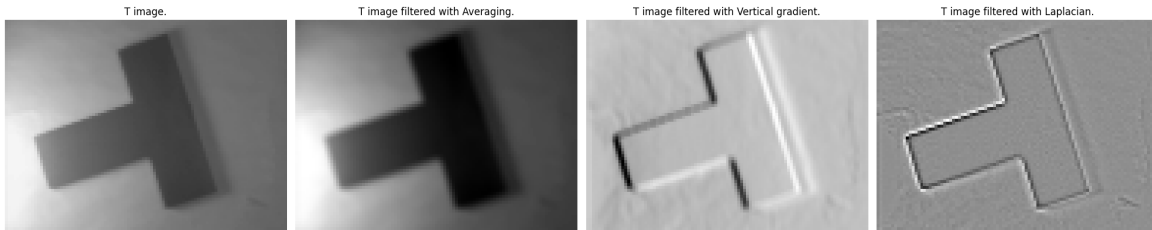


Fig. 6: Filtering in *T* image.

The second part of this task involves the application of a **median filter** to an image with **salt and pepper noise**. This noise was added using the function *random\_noise* and with a probability of 0.2, simulating random black and white pixels. After this, the median filter was applied to the noisy image, in order to remove the noise while preserving edges. This technique replaces each pixel's value with the median of its neighbors within a defined kernel.

The resulting images are shown in Figure 7. As we can notice, the filter effectively removed noise while maintaining edge sharpness.

```

1 from skimage.util import random_noise
2 from scipy.ndimage import median_filter

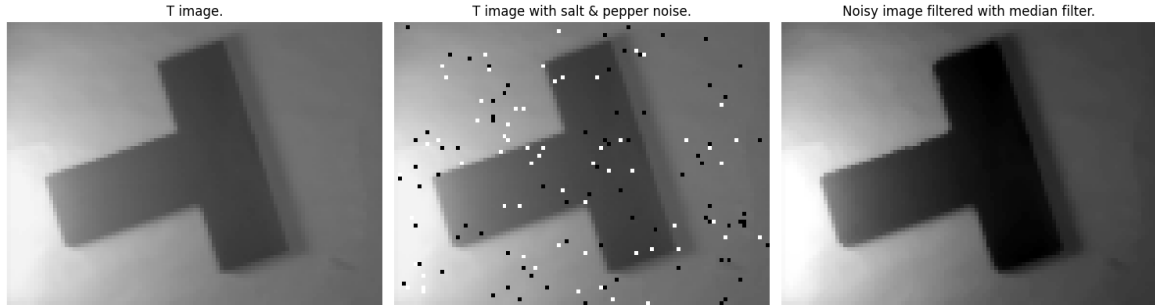
```



```

3
4 #Adding 'salt & pepper' noise
5 noisy_image_T = random_noise(image_T_uint8, mode='s&p', amount=0.02) #amount=0.02 to simulate the
   noise
6 noisy_image_T = (noisy_image_T * 255).astype(np.uint8) #Conversion in uint8
7
8 #Applying the median filter
9 med_filt_image_T = median_filter(noisy_image_T, size=3)
10
11 #Plot
12 plt.figure(figsize=(15, 5))
13 plt.subplot(1, 3, 1)
14 plt.title("T image.")
15 plt.imshow(image_array_T, cmap='gray', norm=NoNorm())
16 plt.axis('off')
17
18 plt.subplot(1, 3, 2)
19 plt.title("T image with salt & pepper noise.")
20 plt.imshow(noisy_image_T, cmap='gray')
21 plt.axis('off')
22
23 plt.subplot(1, 3, 3)
24 plt.title("Noisy image filtered with median filter.")
25 plt.imshow(med_filt_image_T, cmap='gray')
26 plt.axis('off')
27
28 plt.tight_layout()
29 plt.show()

```



**Fig. 7:** Median filter on  $T$  image.

For this last part of the task, we analyze four grayscale images of the Lena dataset.

- In *lena1* image, a **Gaussian filter** was applied in order to reduce noise and smooth the image. The function used was *gaussian\_filter*, with a sigma value of 1.2. This operation blurs the image, softening the transitions between the pixel values.
- In *lena2* image, we first apply a **median filter**, in order to reduce salt and pepper noise. The kernel size for this filter was set to 5. After this, a sharpness kernel was applied in order to enhance the edges of the image. The sharpness kernel used is a 3x3 kernel with negative values surrounding a central positive value, defined as:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}.$$

This kernel was applied using *filter2D* function, that enhances the edges of the image.

- In *lena3* image, first we apply the sharpness kernel used also before, and after the **averaging filter H1** defined in the initial part of task 3. It helps in smoothing noise by averaging pixel values within a specified neighborhood. The function used was *convolve2d*.
- *lena4* image was denoised using the **Non Local Means (NLM) algorithm** through the function *fastNLMMeansDenoising*. It works by averaging the pixels within a local neighborhood, while considering the similarity between the regions of the image. It is effective especially for removing noise

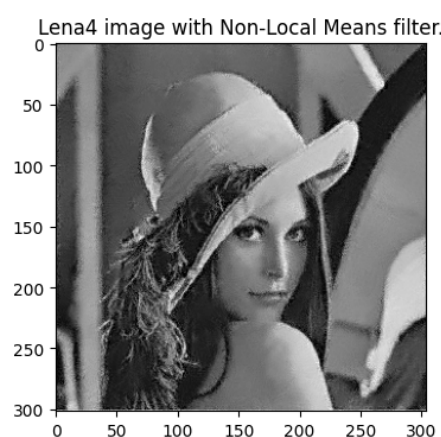
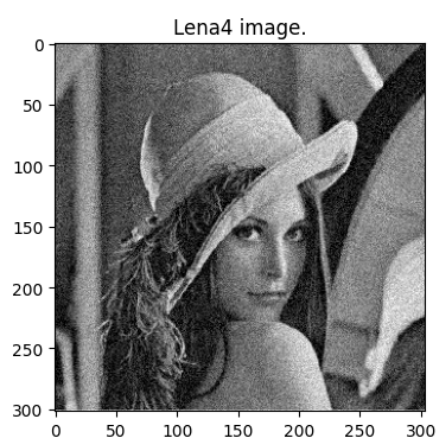
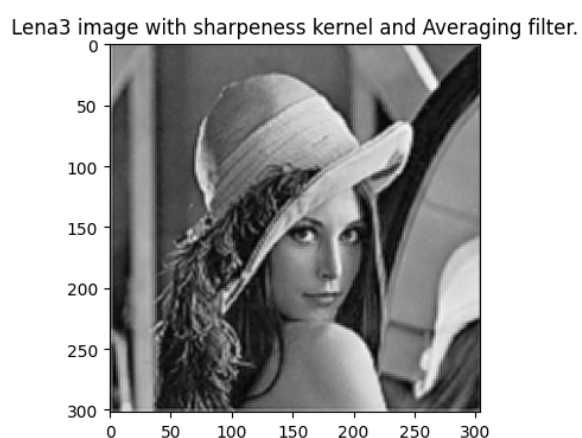
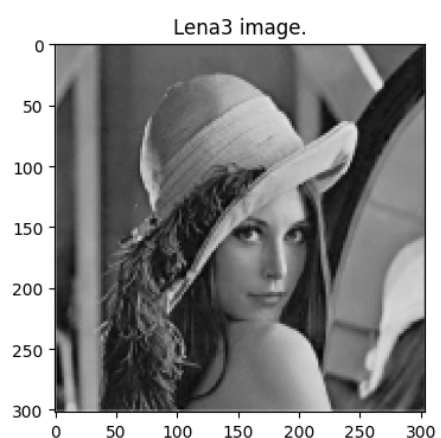
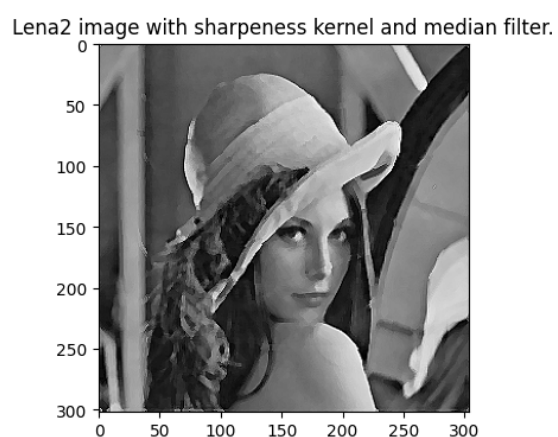
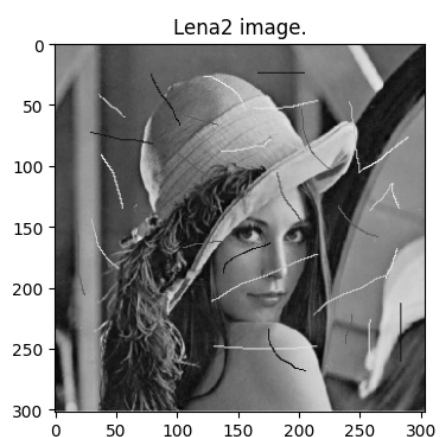
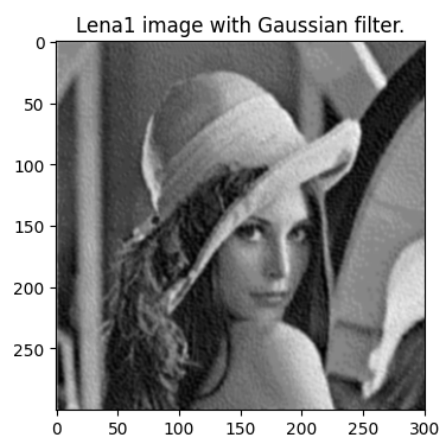
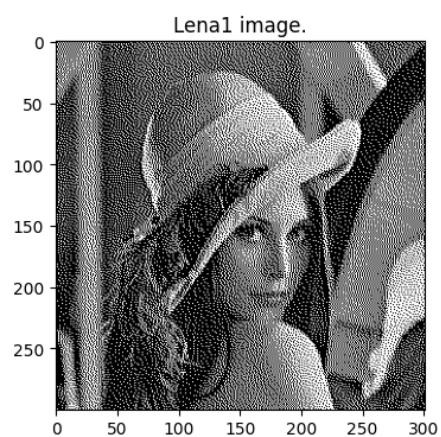
without eliminating details. The hyperparameter  $h$  of the function was set to 14 after some trials, in order to allow the use of more dissimilar pixels. A lower value is usually used for more similar ones.

The code to implement the filtering is shown below, and the final images both original and filtered, are shown in Figure 8.

```

1 from scipy.ndimage import gaussian_filter
2
3 #Load images
4 image_lena1 = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\
    assignment 6 - IP\\IP6\\IP6\\lena1.tif").convert('L')
5 image_lena2 = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\
    assignment 6 - IP\\IP6\\IP6\\lena2.tif").convert('L')
6 image_lena3 = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\
    assignment 6 - IP\\IP6\\IP6\\lena3.tif").convert('L')
7 image_lena4 = Image.open("C:\\Users\\sofyc\\OneDrive\\Desktop\\UPEC\\Pattern recognition\\
    assignment 6 - IP\\IP6\\IP6\\lena4.tif").convert('L')
8
9 #To array
10 image_array_lena1 = np.array(image_lena1)
11 image_array_lena2 = np.array(image_lena2)
12 image_array_lena3 = np.array(image_lena3)
13 image_array_lena4 = np.array(image_lena4)
14
15 #To lena1: gaussian filter
16 gauss_lena1 = gaussian_filter(image_array_lena1, sigma=1.2)
17 #To lena2: sharpeness kernel and median filter --> the image has salt & pepper noise.
18 med_lena2 = median_filter(image_array_lena2, 5) #median filter
19 kernel_sharpness = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]]) #kernel
20 med_lena2 = cv2.filter2D(med_lena2, -1, kernel) #sharpen the image
21 #To lena3: sharpeness kernel and averaging filter
22 sharpened_lena3 = cv2.filter2D(image_array_lena3, -1, kernel) #sharpen the image
23 conv_lena3 = convolve2d(sharpened_lena3, H1, mode='same', boundary='symm') #averaging filter
24 #To lena4: NLM algorithm
25 conv_H2_lena4 = cv2.fastNlMeansDenoising(image_array_lena4, h=14)
26
27 #Plot
28 fig, axs = plt.subplots(4,2, figsize=(12, 15))
29
30 axs[0,0].imshow(image_lena1, cmap='gray', norm=NoNorm())
31 axs[0,0].set_title("Lena1 image.")
32 axs[0,1].imshow(gauss_lena1, cmap='gray')
33 axs[0,1].set_title("Lena1 image with Gaussian filter.")
34
35 axs[1,0].imshow(image_lena2, cmap='gray', norm=NoNorm())
36 axs[1,0].set_title("Lena2 image.")
37 axs[1,1].imshow(med_lena2, cmap='gray')
38 axs[1,1].set_title("Lena2 image with sharpeness kernel and median filter.")
39
40 axs[2,0].imshow(image_lena3, cmap='gray', norm=NoNorm())
41 axs[2,0].set_title("Lena3 image.")
42 axs[2,1].imshow(conv_lena3, cmap='gray')
43 axs[2,1].set_title("Lena3 image with sharpeness kernel and Averaging filter.")
44
45 axs[3,0].imshow(image_lena4, cmap='gray', norm=NoNorm())
46 axs[3,0].set_title("Lena4 image.")
47 axs[3,1].imshow(conv_H2_lena4, cmap='gray')
48 axs[3,1].set_title("Lena4 image with Non-Local Means filter.")
49
50 plt.tight_layout()
51 plt.show()

```



**Fig. 8:** Filtering on *lena* images.

## References

- [1] Geveo: Image Smoothing Algorithms. <https://blog.geveo.com/Image-Smoothing-Algorithms> (2024)
- [2] Dilhani, S.: Digital Image Processing Filters. <https://medium.com/@shashikadilhani97/digital-image-processing-filters-832ec6d18a73> (2024)
- [3] YouTube: Image of YouTube Video. <https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DB7FIuz8vAkE&psig=AOvVaw39hZ5T4yQmQlxTA4LAAaq8&ust=1734978308724000&source=images&cd=vfe&opi=89978449&ved=0CBcQjhxqFwoTCMDgoIKAvIoDFQAAAAAdAAAAABAE> (2024)
- [4] YouTube: B7FIuz8vAkE Video on YouTube. <https://www.youtube.com/watch?v=B7FIuz8vAkE> (2024)
- [5] User, S.O.: How do I to implement Matlab's 'edge' function in Python (OpenCV or skimage)? <https://stackoverflow.com/questions/61272291/how-do-i-to-implement-matlabs-edge-function-in-python-opencv-or-skimage> (2020)
- [6] Majidzadeh, F.: Digital Image Processing and Pattern Recognition, (2023)
- [7] GeeksforGeeks: Image Enhancement Techniques Using OpenCV Python. <https://www.geeksforgeeks.org/image-enhancement-techniques-using-opencv-python/> (2024)
- [8] Vidhya, A.: Sharpening an Image Using OpenCV Library in Python. <https://www.analyticsvidhya.com/blog/2021/08/sharpening-an-image-using-opencv-library-in-python/> (2021)