Foundations of Data Science
Image Filtering and Object Identification

Andrea Gasparini, Edoardo Di Paolo, Cirillo Atalla

October 2020

Contents

1 Image Filtering

1.1 Question 1.d

The effect of applying a filter can be studied by observing its *impulse response*. Executing the following snippet we created a test image (Figure 1) in which only the central pixel has a non-zero value:

```
img_imp = np.zeros([27,27])
img_imp[13, 13] = 1.0
plt.figure(1), plt.imshow(img_imp, cmap='gray')
```

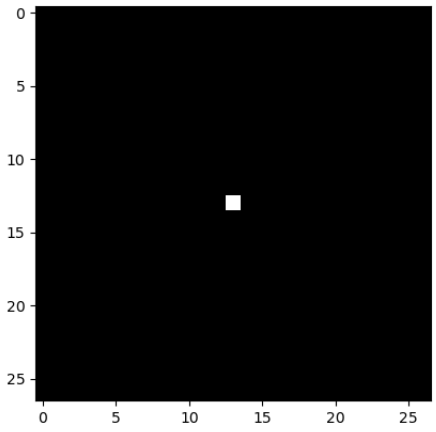

Figure 1: Test image

Executing the following snippet we created 1D Gaussian and Gaussian derivative kernels, `Gx` and `Dx` respectively.

```
sigma = 7.0
[Gx, x] = gauss_module.gauss(sigma)
[Dx, x] = gauss_module.gaussdx(sigma)
```

We applied the following filter combinations:

1. First `Gx`, then $\texttt{Gx}^T$
2. First `Gx`, then $\texttt{Dx}^T$
3. First $\texttt{Dx}^T$, then `Gx`
4. First `Dx`, then $\texttt{Dx}^T$
5. First `Dx`, then $\texttt{Gx}^T$
6. First $\texttt{Gx}^T$, then `Dx`

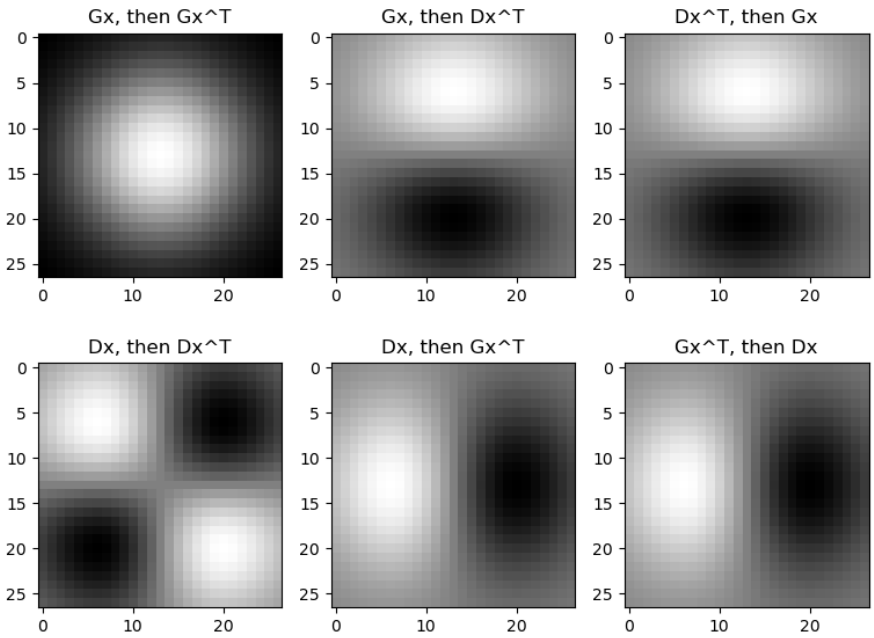

Figure 2: Applying filter combinations

As we can see in Figure 2, the first filter combination is the result of the gaussian filter applied first on the rows and then on the columns. So we compute two 1D convolution instead of one 2D convolution due to the separability of gaussian filter.
The second and third filter combinations are the same, there is no difference in applying `Gx` and then $\texttt{Dx}^T$ or viceversa. We find an edge when we apply the first derivative filter.
In the fourth filter combination there are some edges. We can see the changes from white to black and viceversa.
The fifth and sixth filter combinations are the same. This is the same case of 2nd and 3rd, but with inverted axis; there is no difference in applying `Dx` and then $\texttt{Gx}^T$ or viceversa.

1.2 Question 1.e

We implemented a `gaussderiv` method that takes an input image and generates three copies of it. The first two are smoothed according to a standard deviation σ and derived in the directions x and y respectively; the third is a combination of them, obtained as $f(img_x, img_y) = \sqrt{img_x^2 + img_y^2}$ where img_x and img_y are corresponding pixels of the two filtered images.

The results of applying `gaussderiv`, with $\sigma = 7.0$, to the provided example images (`graf.png` and `gantrycrane.png`) are shown in Figures 5 and 6.

Figure 3: `graf.png`

Figure 4: `gantrycrane.png`

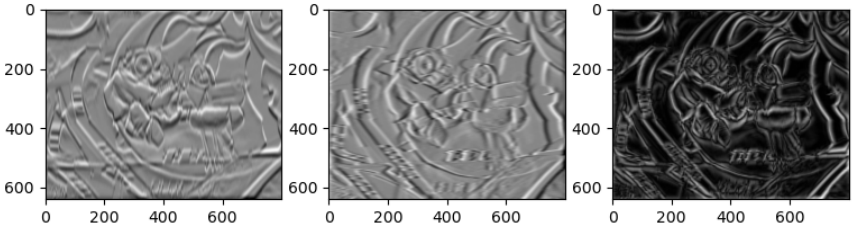

Figure 5: Results of applying `gaussderiv` on `graf.png`

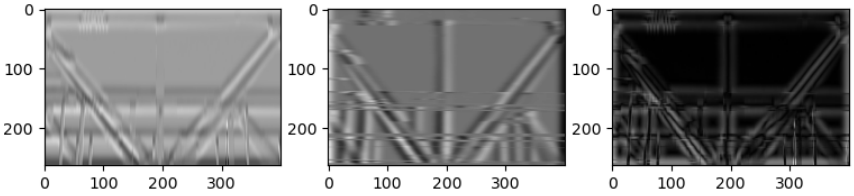

Figure 6: Results of applying `gaussderiv` on `gantrycrane.png`

Smoothing an image is important because with this process we can leave out the noise of the images.

From the left plot in Figures 5 and 6 we can see the edges on the vertical axis, while in the second one we can see the edges on the horizontal axis. The third one is the combination of the previous two.

2 Object Identification

2.1 Question 3.c

For the tests we used different combinations of distance type (χ^2, *intersect* and *l2*), histogram type (*grayvalue, rgb, rg* and *dxdy*) and number of bins.

These are the results we got:

Distance type	**Histogram type**	**Num bins**	**Correct**	**Rate**
intersect	rgb	25	81	0.910112
intersect	rgb	15	78	0.876404
intersect	rg	25	75	0.842697
intersect	rg	15	74	0.831461
intersect	rgb	30	72	0.808989
intersect	rgb	20	71	0.797753
intersect	rgb	10	70	0.786517
intersect	rg	20	65	0.730337
intersect	rg	30	65	0.730337
intersect	rg	10	62	0.696629
chi2	rgb	5	60	0.674157
chi2	rgb	10	59	0.662921
chi2	rg	5	57	0.640449
chi2	rgb	15	55	0.617978
l2	rgb	10	54	0.606742
chi2	rg	15	53	0.595506
l2	rg	16	53	0.595506
chi2	rg	10	52	0.584270
l2	rg	10	52	0.584270
l2	rg	15	52	0.584270
chi2	rg	25	50	0.561798
chi2	rg	20	48	0.539326
chi2	rgb	20	46	0.516854
intersect	dxdy	70	46	0.516854
intersect	grayvalue	20	46	0.516854
intersect	grayvalue	35	46	0.516854
chi2	rgb	25	45	0.505618
intersect	grayvalue	10	45	0.505618
intersect	grayvalue	30	45	0.505618
intersect	grayvalue	25	43	0.483146
intersect	grayvalue	70	43	0.483146
l2	rgb	15	42	0.471910
chi2	rg	30	41	0.460674
chi2	dxdy	150	41	0.460674
chi2	grayvalue	10	41	0.460674

intersect	dxdy	35	40	0.449438
intersect	grayvalue	15	40	0.449438
chi2	dxdy	70	39	0.438202
l2	dxdy	70	39	0.438202
l2	grayvalue	10	39	0.438202
intersect	dxdy	25	37	0.415730
intersect	dxdy	30	37	0.415730
chi2	rgb	30	36	0.404494
chi2	grayvalue	15	36	0.404494
chi2	grayvalue	25	36	0.404494
chi2	grayvalue	30	36	0.404494
l2	grayvalue	15	36	0.404494
l2	grayvalue	30	36	0.404494
intersect	dxdy	20	35	0.393258
chi2	grayvalue	5	35	0.393258
chi2	grayvalue	20	35	0.393258
chi2	dxdy	30	34	0.382022
chi2	dxdy	25	33	0.370787
l2	dxdy	30	33	0.370787
chi2	dxdy	20	32	0.359551
intersect	dxdy	15	31	0.348315
chi2	grayvalue	70	30	0.337079
chi2	dxdy	15	29	0.325483
l2	dxdy	15	29	0.325483
l2	grayvalue	70	28	0.314607
chi2	grayvalue	150	27	0.303371
chi2	dxdy	5	25	0.280899
chi2	dxdy	10	25	0.280899
l2	dxdy	10	25	0.280899
intersect	dxdy	10	23	0.258427

As we can see that the best measure is the intersect with *rgb* and *rg* histogram types but with the other types (*dxdy, grayvalue)* it is not very good. We can see also that the l2 distance is not good to recognize objects.

About the histograms the best ones are rg and rgb due to their robustness. The color histograms are really useful with the intersect distance while with the other distance types the recognition rate is similar to the dxdy and grayvalue histogram types. We tried with different number of bins in a range from 5 to 150.