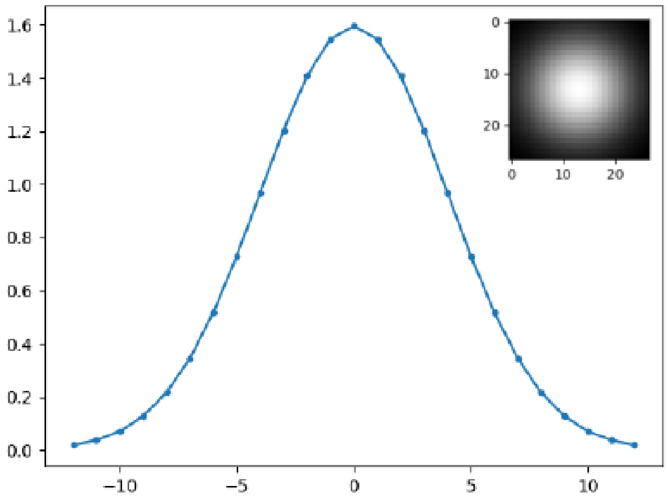
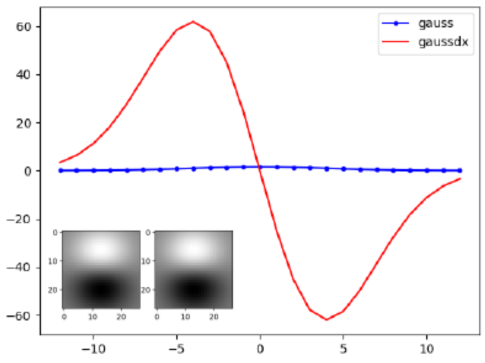
1. Report assignment 1

1.1 Exercise 1

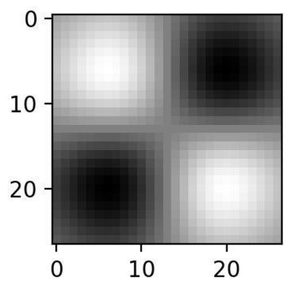
1.1.1 Point d



1. In the first convolution, in the first 1D Gaussian filter it applies it first on *x* direction and then on the *y* direction. That double operation is equivalent to a 2D Gaussian filter. With that filter combination gives us the possibility to apply a smooth on image. In our example we get a smoothed white point which reflects the bell-shaped function viewed from above. Hence applying the first convolution on the rows of the image we get black rows, apart the central row which get smoothed. In the second convolution the central row is smoothed in vertical, propagating the white pixels through all the image obtaining a radial view.
2. As for the mechanism explained above, *filter.py* applies first a normal Gaussian convolution parallel to the horizontal axes, but now the second convolution is the derivative *Dx* of the Gaussian filter. The derivative filter array has, in the first half, values tending to 1, while in the second half the values tend to 0. This means that applying the filter in vertical (*Dx.T*) the upper part of the image is lighter, while the bottom part is darker.

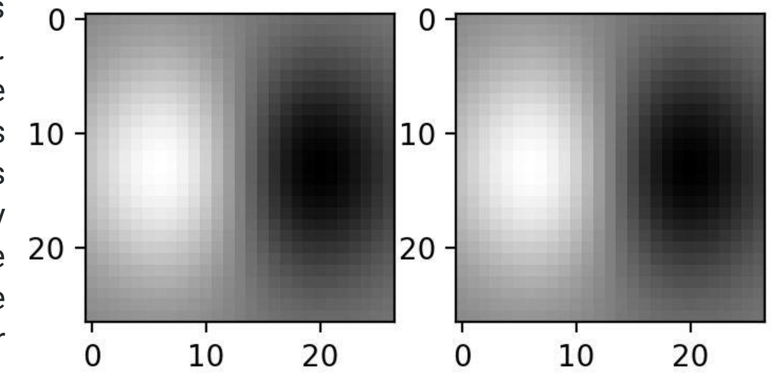


3. Now the convolution is first applied with *Dx.T* and then with *Gx*. The resulting image is the same as the second point, this means that the order in which the convolution is applied does not matter. Hence it seems that the convolution is commutative. The transformation could be different if the direction in which we apply the kernels changes, like for example *conv(conv(img, Gx.T), Dx)* (reference to points 5 and 6).
4. Here the system applies both the convolution with a kernel obtained by the derivative function *Dx*. The first convolution is applied horizontally, and we get an image where the first half is white and the second one is darker. The transpose kernel has the brighter pixels in the upper part of the array, while the darker ones in the bottom part. Applying the filter vertically, in the first half the brighter pixels are brought in the first quadrant and the darker ones in the third quadrant because it is applied on the



white ones. In the second half (the darker part), the pixels are brought to the second quadrant because it is applied on the darker ones. In conclusion, the derivative function yields the pixels on which it's applied in the direction where the function "goes up".

5. Like the point 2 or 3 the convolution is applied two times: the first application with the filter *Dx* and the last one with the filter *Gx.T.* The application of *Dx* create a situation where in the horizontal axes the left part of the pixels tending to *1.0*. The right part of the image, with this application, tending to *0.0* value. With the convolution on filter *Gx.T* the vertical axes create a specular situation of point 2: the Gaussian filter do a smooth on the lighter part and the dark one, respective. At the end of computation, the situation is a left light smoother part and the respective right part a smoother dark one. At the human vision is like 90° rotate of image at the second point.
6. At this point the application is that *conv(conv(img, Gx.T), Dx)*. For the same reason of the point 3 and 2 and also the thesis of the point 5 the result of this application explain the property of commutation and the property of differentiation; the inverted application of filter create the situation of rotation.

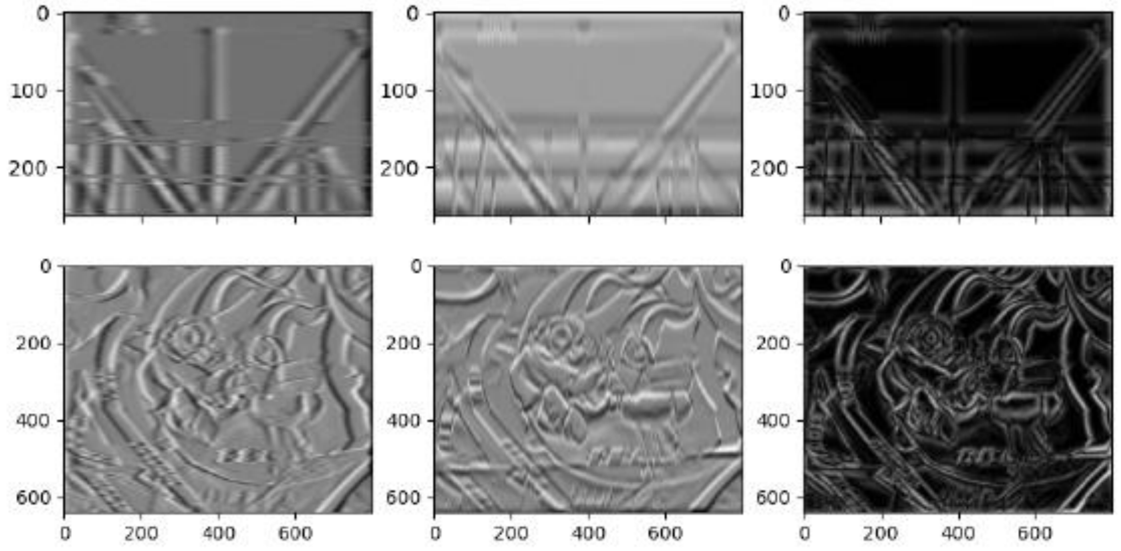


   With *Gx.T* application the image is smoothed on the vertical axes with the Gaussian filter: the pixels on the centre of arrays are lighted and the pixels at the borders darker. When *Dx* do its change, the situation is the same of the precedent point: the left part of image is lighted and the right one darker. The derivate of a filter array *x* has the left part values tending to *1.0* and the opposite ones tending to *0.0*.
   The differentiation rule is: $\frac{d}{dx}(f * g) = \frac{df}{dx} * g = f * \frac{dg}{dx}$. That, with the commutative one, create the association in point 2-3 and 5-6.

1.1.2 Point e

The derivative filter is used to detect edges. Ideally, we should first smooth the image in order to remove the noise, in fact a smoothed image will remove small details. The effect to apply the derivative filter is to increase the change of colours, so it points out the differences. In our case the derivative filter is applied without smoothing the image first. Therefore in graf.png the filter highlights small details like the antenna of the car. In our case σ is settled at 7.0. Edges are large, less defined and strength. Decreasing σ has the effect to decreasing the edges but obtaining more defined lines. Applying the filter parallel to the *x* axes, we highlight the vertical edges. For example, in gantrycrane.png the central pillar is well defined in *imgDx*, while it almost disappears in *imgDy*. On the other hand, applying the filter parallel to the *y* axes, the horizontal edges are spotted.

The last image is obtained by calculating the magnitude of the gradient. This method highlights the edges in every direction.



In the image there are the two cases of the application of *gaussderiv* in two differences images. The left ones are the application on the first dimension, the cantered ones the application on the other dimension and, at right, there are the result of merging. The merge of the two-image derivate in *x* and in *y* explains the power-up of the edge on *2D*.

1.2 Exercise 3

1.2.1 Point c

The experiment it was conduct on a group of coloured histograms with different type, different type of distance and with different number of bins. In this experiment emerge the power of *intersect* distant function. In fact, this function explains the best result in computation. At the same, the *RGB* histogram has the best result. The best combination with a middle-large number of bins, for example *30*, is that: *intersect* + *RGB*. Another good function for the distant is the *chi2*, that have result like the previous one with many bins. The worst function for the histogram is *dxdy*. It with all function of distance return a bad result. Another not good function is the distance function *l2*, but that with a not many bins result a good compromise. In the table in the next page there are some of many tests carried out. They are printed by the match number in reserve order. At the top there is the best combination of function for this group of tests.

dist_type	hist_type	num_bins	match_number	match_percentage
intersect	rgb	30	79	0.887640
intersect	rgb	15	76	0.853933
intersect	rgb	35	76	0.853933
intersect	rg	35	74	0.831461
chi2	rg	35	73	0.820225
chi2	rg	30	72	0.808989
intersect	rg	30	72	0.808989
chi2	rgb	30	71	0.797753
chi2	rgb	15	70	0.786517
intersect	rg	15	67	0.752809
chi2	rgb	35	65	0.730337
chi2	rg	15	61	0.685393
intersect	rgb	5	56	0.629213
l2	rgb	15	56	0.629213
l2	rgb	5	55	0.617978
l2	rg	15	52	0.584270
intersect	rg	5	50	0.561798
l2	rg	5	50	0.561798
chi2	rgb	5	49	0.550562
l2	rg	35	49	0.550562
l2	rg	30	48	0.539326
chi2	rg	5	42	0.471910
l2	rgb	35	42	0.471910
l2	rgb	30	39	0.438202
chi2	dxdy	5	38	0.426966
intersect	dxdy	5	32	0.359551
intersect	dxdy	35	32	0.359551
intersect	dxdy	30	30	0.337079
intersect	dxdy	15	29	0.325843
l2	dxdy	15	26	0.292135
l2	dxdy	5	26	0.292135
l2	dxdy	30	23	0.258427
l2	dxdy	35	22	0.247191
chi2	dxdy	35	10	0.112360
chi2	dxdy	30	8	0.089888
chi2	dxdy	15	4	0.044944