Contents

Chapter 1

Blur filters

The Fourier Transform is an indispensable tool for linear systems analysis, image analysis, and for efficient filter output computation. Among the benefits of the Fourier transform representation: it's easy to analyze images according to spatial frequency, and this represents some progress in interpreting the image over merely a pixel representation. But the Fourier transform has a major drawback as an image representation: it's too global! Every sinusoidal component covers the entire image. The Fourier transform tells us a little about *what* is happening in the image (based on the spatial frequency content), but it tells us nothing about *where* it is happening.

On the other hand, a pixel representation gives great spatial localization, but a pixel value by itself doesn't help us learn much about what's going on in the image. A Fourier representation tells us a bit about what's going on, but nothing about where it happens. We seek a representation that's somewhere in between those two extremes.

In the next two chapters we will analyze several important linear filters. We will study spatial filters and then temporal filters.

1.1 Human visual system and the contrast sensitivity function

Before we start describing different types of linear filters, let's start by gaining some subjective experience by playing with one: our own visual system. Although our visual system is clearly a non-linear system, linear filter theory can explain some aspects of our perception. Indeed, under certain conditions, the first stages of visual computation perform of the visual system can be approximated by a linear filter.

To experience the transfer function of our own visual system, let's build the following $N \times M$ image:

$$\mathbf{I}[n,m] = A[m] \sin(2\pi f[n]\, n/N) \tag{1.1}$$

with

$$A[m] = A_{min} \left(\frac{A_{max}}{A_{min}} \right)^{m/M} \tag{1.2}$$

and

$$f[n] = f_{min} \left(\frac{f_{max}}{f_{min}} \right)^{n/N} \tag{1.3}$$

This image is separable and it is composed of two factors: an amplitude, $A[m]$, that varies only along the vertical dimension, and a wave with a frequency, $f[n]$, that varies along the horizontal component. We set the amplitude so that it is A_{max} in the lowest part of the image and it decreases logarithmically to A_{min} at the top. And the frequency function is defined as an increasing function that starts from 0 and grows up to 60 (with $N = 2048$

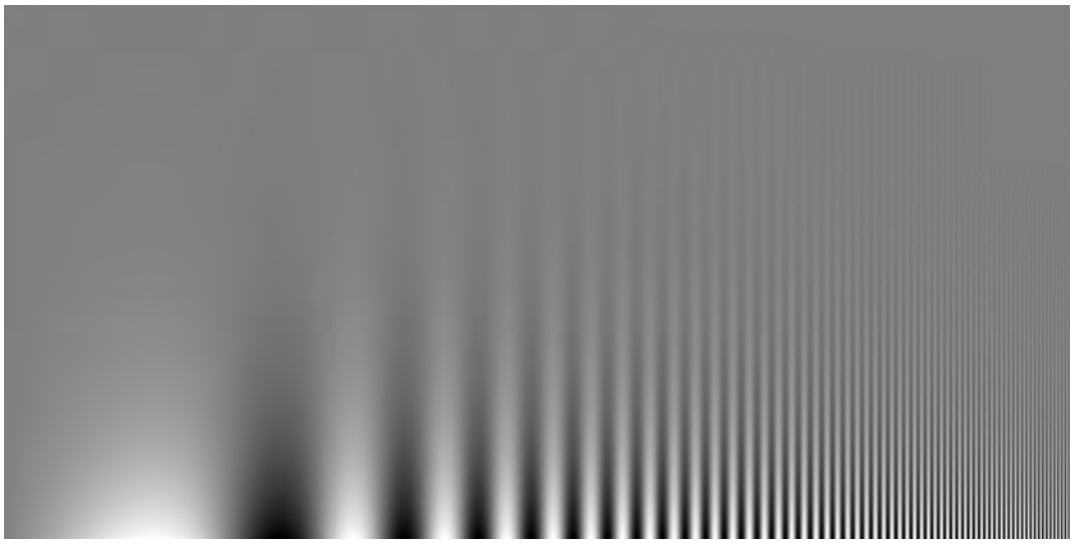

Figure 1.1: Contrast sensitivity function shown by the Campbell & Robson chart.

being the number of horizontal pixels in the image). This image is shown in figure 1.1 and it is also called the Campbell & Robson chart. It shows a signal with a wave that oscillates slow at the left and faster towards the right and that has high contrast a the bottom and loses contrast towards the top becoming invisible.

The first sign that our visual system is non-linear is that we do not perceive the amplitude as changing logarithmically from top to bottom. It feels more linear. This is because our photo-receptors compute the log of the incoming intensity (approximately).

What is interesting is that figure 1.1 is not perceived as being separable. If you trace the region where the sine wave seems to disappears you will trace a curve. In fact, your visual system is behaving as a band-pass filter: you are sensitive to middle spatial frequencies (with a pick around 6 cycles/degree) and you are less sensitive for very low spatial frequencies (left of the image) and to high-spatial frequencies (right of the image). This curve is called the contrast sensitivity function (CSF) in the psychophysics literature and is closely related to the transfer function of the filter.

The CSF is not a simple linear function but it can be approximated by one under certain conditions. However, the CSF changes depending of many factors such as overall intensity (the pick moves towards the left under low illumination.), adaptation (long exposure to one frequency reduces the sensitivity for that frequency), age, ...

The DFT and use of sine waves became very popular in the study of visual psychophysics.

1.2 Box filter

Let's start with a very simple filter, the box filter. In the box filter, each output pixel is the sum of the input pixels around its neighborhood. The box convolution kernel can be written as:

$$h_{N,M}[n,m] = \begin{cases} 1 & \text{if } -N \leq n \leq N \text{ and } -M \leq m \leq M \\ 0 & \text{otherwise} \end{cases} \tag{1.4}$$

The box filter is separable as it can be written as the convolution of two 1D kernels: $h_{N,M}[n,m] = h_{N,0} \circ h_{0,M}$.

Filtering an image with the box filter results in blurring the picture. Figure 1.2 shows some box filters and the corresponding output images. Figure 1.2 (a) shows an image convolved with a uniform, rectangular kernel. Each pixel is an average of the input pixels within the rectangle. Figure 1.2 (b) and (c) show the results of blurring in just one direction. Blur

Figure 1.2: Blurring with a square, and a horizontal and vertical line. Note the structures that are both averaged out (along the direction of blurring), and maintained (perpendicular to that direction) in each resulting image.

happens very often in real life. It happens when we look at something very far away, or at some detail inside a picture, or when we remove our eyeglasses (or wear some that are not ours).

The box filter is simple but it has a number of issues:

- If you convolve two boxes you get a triangle. You can easily check this. For instance, in the simple case where $N = 1, M = 0$:

$$h_{1,0} \circ h_{1,0} = [1, 1, 1] \circ [1, 1, 1] = [1, 2, 3, 2, 1] \tag{1.5}$$

  the output is a triangular filter with length $2 \times L - 1$, with $L = 3$ the length of the box filter. Although that is not a problem at first sight, it means that if you blur an image twice with box filters, what you get is not equivalent to blurring only once with a larger box filter.

- The box filter is not a perfect blurring filter. A blur filter should attenuate high spatial frequencies with stronger attenuation for higher spatial frequencies. However, if you consider the highest spatial frequency, which will be an oscillating signal that takes successively on the values 1 and -1: $[..., 1, -1, 1, -1, 1, -1, ...]$ when filtered with the box filter $h_{1,0}$ the results is the same signal! However, if you filter a wave with lower frequency such as $[..., 0.5, 0.5, -1, 0.5, 0.5, -1, ...]$ then the result is $[..., 0, 0, 0, 0, ...]$. Therefore, the attenuation is not monotonic with spatial frequency as it is shown in figure 1.3. This is not a desirable behavior for a blurring filter and it can cause artifacts to appear. This could be addressed using an even size box filter $[1, 1]$. However, an even box filter is not centered around the origin and the output image will have a half-pixel translation. Therefore, odd filter sizes are preferred.

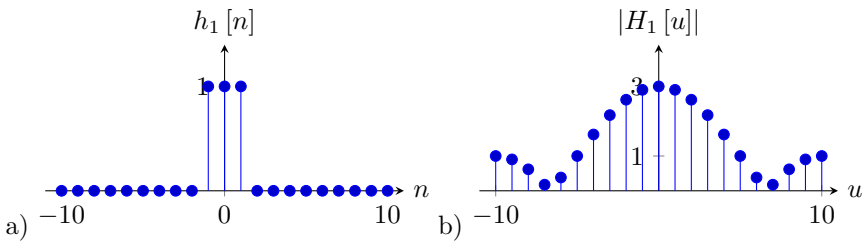

Figure 1.3: a) A one dimensional box filter ($[1, 1, 1]$) and b) its Fourier transform over 20 samples. Note that the frequency gain is not monotonically decreasing with spatial frequency.

The next blurring filter addresses both of these issues.

1.3 Gaussian filter

One of the important blurring (low-pass) filters in computer vision is the gaussian filter. The gaussian filter is important because it is a good model for many naturally occurring filters. It also has a number of properties, as we will discuss here, that make it unique.

The gaussian distribution is defined in continuous variables. In one dimension:

$$g(x;\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{x^2}{2\sigma^2} \tag{1.6}$$

and in 2 dimensions:

$$g(x,y;\sigma) = \frac{1}{2\pi\sigma^2} \exp -\frac{x^2+y^2}{2\sigma^2} \tag{1.7}$$

The parameter σ adjusts the spatial extend of the gaussian. The normalization constant is set so that the function integrates to 1.

In order to use this filter in practice we need to consider discrete locations and also approximate the function by a finite support function. In practice, we only need to consider samples within three standard deviations $x \in (-3\sigma, 3\sigma)$. At 3σ the amplitude of the gaussian is around 1% of its central value. Unfortunately, many of the properties of the gaussian that we will discuss later are only true in the continuous domain and are only approximated when using its discrete form.

For a given standard deviation parameter, σ, the discretized gaussian kernel is $g[m,n;\sigma]$:

$$g[m,n;\sigma] = \exp -\frac{m^2+n^2}{2\sigma^2} \tag{1.8}$$

We have removed the normalization constant as the sum of the discrete gaussian will be different from the integral of the continuous function. So here we prefer to use the form in which the value at the origin is 1. In practice, we should normalize the discrete gaussian by the sum of its values to make sure that the gain at DC is 1.

By adjusting the standard deviation, σ, of the Gaussian, it is possible to adjust the level of image detail that appears in the blurred image. Figure 1.4 shows the result of narrow and wider Gaussians applied to an image.

The multi-dimensional Gaussian filter has the additional computational advantage that it can be applied as a concatenation of 1-d Gaussian filters. This can be seen by writing the 2-d Gaussian, Eq. (1.8), in the convolution equation, Eq. (**??**). Letting g^x and g^y be the 1-d Gaussian convolution kernels in the horizontal and vertical directions, we have

$$\begin{aligned}
g[m,n] \circ f[m,n] &= \sum_{k,l} g[m-k, n-l] f[k,l] \\
&= \sum_{k,l} \exp -\frac{(m-k)^2+(n-l)^2}{2\sigma^2} \circ f[m,n] \\
&= \sum_{k} \exp -\frac{(m-k)^2}{2\sigma^2} \left(\sum_{l} \exp -\frac{(n-l)^2}{2\sigma^2} f[k,l] \right) \\
&= g^x \circ (g^y \circ f[m,n])
\end{aligned}$$

This can save quite a bit in computation time when applying the convolution of Eq. (1.9). If the 2-d convolution kernel is nxn samples, then a direct convolution of that 2-d kernel scales in proportion to n^2, since Eq. (1.9) requires one multiplication per image position per kernel sample. Using the cascade of two 1-d kernels, resulting in an equivalent 2-d filter of the same size, scales in proportion to $2n$.

Another application of blurring is to remove distracting high-resolution image details. Fig. 1.5 shows a Gaussian low-pass filter applied to remove unwanted image details (the blocky artifacts) from an image.

Properties of the Gaussian filter:

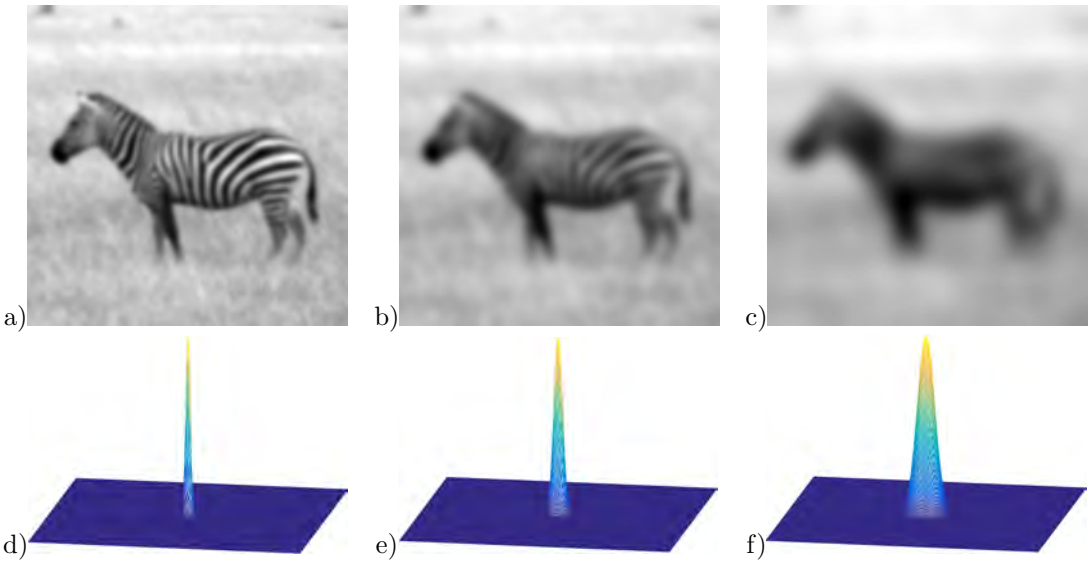

Figure 1.4: An image filtered with three gaussians with standard deviations: a) $\sigma = 2$, b) $\sigma = 4$, and c) $\sigma = 8$. Plots (d) - (f) show the three Gaussians over the same spatial support as the image. The discrete Gaussians are approximated by sampling the continuous Gaussian. The convolutions are perform with mirror boundary conditions.

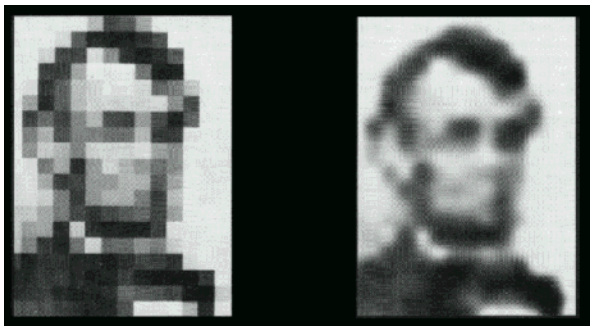

Figure 1.5: Left: input image. Right: blurred version. The left version has many spurious details introduced by the undersampling of the image. The right image has been blurred by a large Gaussian filter. ((image from http://acor.org/sgreene/hmsbeagle/html/content/17/recroom/artgalas.htm, after 1973 image by Bela Julesz and Leon Harmon)).

- The n-dimensional Gaussian is the only completely circularly symmetric operator that is separable.
- The continuous Fourier transform of a gaussian is also a gaussian.

$$G(u, v; \sigma) = \exp -2\pi^2(u^2 + v^2)\sigma^2 \tag{1.9}$$

  Note that this function is monotonically decreasing in magnitude for increasing frequencies, and it is also radially symmetric.
- The convolution of two n-dimensional gaussians is an n-dimensional gaussian.

$$g(x, y; \sigma_1) \circ g(x, y; \sigma_2) = g(x, y; \sigma_3) \tag{1.10}$$

  where the variance of the result is the sum $\sigma_3^2 = \sigma_1^2 + \sigma_2^2$. This is a remarkable property of gaussian filters and is the basis of the gaussian pyramid that we will see later. To prove this property, one can use the Fourier transform of the gaussian and the fact that the convolution is the product of Fourier transforms.
- The gaussian is the solution to the heat equation.
- Repeated convolutions of any function concentrated in the origin result in a gaussian (central limit theorem).
- In the limit $\sigma \to 0$ the Gaussian becomes an impulse. This property is shared by many other functions, but it is a useful thing to know.

However, many of these properties do not work for the discrete approximation $g[n, m; \sigma]$ obtained by directly sampling the values of the gaussian at discrete locations. To see this let's look at some examples. Let's consider a gaussian with variance $\sigma^2 = 1/2$. It can be approximated by 5 samples. We will call this approximation g_5:

$$g_5[n] = [0.0183,\ 0.3679,\ 1.0000,\ 0.3679,\ 0.0183] \tag{1.11}$$

first, note when convolved with a wave $[1, -1, 1, -1, ...]$ the result is not zero. This is to be expected from the form of the FT of the gaussian. You can check that if you compute the approximation for $\sigma^2 = 1$ by discretizing the gaussian, the result obtained is not equal to doing $g_5 \circ g_5$. Therefore, as you apply successive convolutions of gaussian the errors will accumulate.

1.4 Binomial filters

In practice, there are very efficient approximations to the Gaussian filter for certain σ values with nicer properties than when working with discretized gaussians. One common approximation of the gaussian filter is to use binomial coefficients [**?**, **?**]. Binomial coefficients provide a compact approximation of the gaussian coefficients using only integers. The binomial coefficients use the central limit theorem to approximate a gaussian as successive convolutions of a very simple function. The simplest low-pass filter is the box filter $[1, 1]$. The binomial coefficients form the Pascal's triangle as shown in figure 1.6.

The sum of all the coefficients for each binomial filter b_n is 2^n, and their variance is $\sigma^2 = n/4$. One remarkable property of the binomial filters is that $b_n \circ b_m = b_{n+m}$, and, therefore, $\sigma_n^2 + \sigma_m^2 = \sigma_{n+m}^2$, which is the analogous to the gaussian property in the continuous domain. Note that the values of b_2 are different from g_5 despite that both will be used as approximations to the same gaussian.

The simplest approximation to the Gaussian filter is the 3-tap kernel:

$$b_2 = [1, 2, 1] \tag{1.12}$$

b_0							1											$\sigma_0^2 = 0$
b_1						1		1										$\sigma_1^2 = 1/4$
b_2					1		2		1									$\sigma_2^2 = 1/2$
b_3				1		3		3		1								$\sigma_3^2 = 3/4$
b_4			1		4		6		4		1							$\sigma_4^2 = 1$
b_5		1		5		10		10		5		1						$\sigma_5^2 = 5/4$
b_6	1		6		15		20		15		6		1					$\sigma_6^2 = 3/2$
b_7		1		7		21		35		35		21		7		1		$\sigma_7^2 = 7/4$
b_8	1		8		28		56		70		56		28		8		1	$\sigma_8^2 = 2$

Figure 1.6: Binomial coefficients. To build the Pascal's triangle, each number is the sum of the number above to the left and the one above to the right.

This filter is interesting because it is even (so it can be applied to an image without producing any translation) and its DFT is:

$$B_2[u] = 2 + 2\cos(2\pi u/N) \tag{1.13}$$

it has a monotonic amplitude gain in frequency (there are no ripples) as shown in figure 1.7. All the even binomial filters can be written as successive convolutions with the kernel $[1, 2, 1]$. Therefore, their Fourier transform is a power of the Fourier transform of the filter $[1, 2, 1]$ and therefore they are also monotonic:

$$B_{2n}[u] = (2 + 2\cos(2\pi u/N))^n \tag{1.14}$$

For all the binomial filters b_n, when they are convolved with the wave $[1, -1, 1, -1, ...]$, the result is the zero signal $[0, 0, 0, 0, ...]$. This is a very nice property of binomial filters and will become very useful later when talking about downsampling an image (see section **??**).

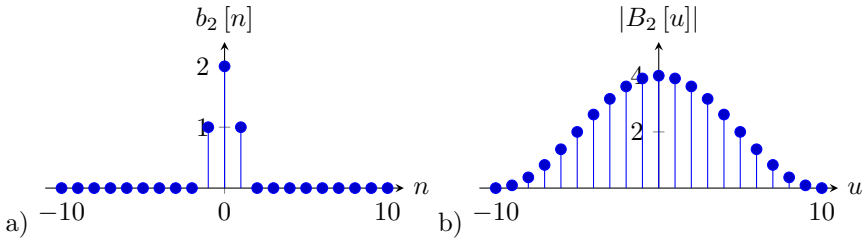

Figure 1.7: a) A one dimensional 3-tap approximation to the gaussian filter ($[1, 2, 1]$) and b) its Fourier transform over 20 samples. Note that the frequency gain is decreasing monotonically with spatial frequency and it becomes zero at the highest frequency, $G_3[10] = 0$.

The gaussian in 2D can be approximated, using separability, as the convolution of two binomial filters one vertical and another horizontal. For instance:

$$b_{2,2} = b_{2,0} \circ b_{0,2} = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \circ \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{1.15}$$

The gaussian and the binomial filter are extensively used in computer vision.

Chapter 2

Image derivatives

Computing image derivatives is an essential operator for extracting useful information from images. As we show in the previous chapter, image derivatives allowed us computing boundaries between objects and to have access to some of the 3D information lost when projecting the 3D world into the camera plane. Derivatives are useful because they give us information about where are happening the changes in the image and we expect those changes to be correlated with transitions between objects.

2.1 Image derivatives

If we had access to the continuous image, then image derivatives could be computed as: $\partial \mathbf{I}(x, y)/\partial x$. However, there are two reasons why we might not be able to apply this definition

- the first one is that we only have access to a sampled version of the input image: $\mathbf{I}[m, n]$
- even if we had access to the continuous image, the image could contain many non-derivable points and the gradient would not be defined. We will see how to address this issue later when we study gaussian derivatives.

for now, let's focus on the problem of approximating the continuous derivative with discrete operators. As the derivative is a linear operator, it can be approximated by a discrete linear filter. There are several ways in which image derivatives can be approximated.

Let's start with a simple approximation to the derivative operator that we have already played with: $d_0 = [1, -1]$. In one dimension, convolving a signal $f[n]$ with this filter results in:

$$f \circ d_0 = f[n] - f[n-1] \tag{2.1}$$

this approximates the derivative by the difference between consecutive values. Figure 2.1.c shows the result of filtering a 1-d signal (fig.2.1.a) convolved with $d_0[n]$ (fig.2.1.b). The output is zero wherever the input signal is constant and it is large in the places where there are variations in the input values. However note that the output is not perfectly aligned with the input. In fact there is a half a sample displacement to the right. This is due to the fact that $d_0[n]$ is not centered around the origin.

This can be addressed with a different approximation to the spatial derivative: $d_1 = [1, 0, -1]/2$. In one dimension, convolving a signal $f[n]$ with $d_1[n]$ results in:

$$f \circ d_1 = \frac{f[n+1] - f[n-1]}{2} \tag{2.2}$$

Figure 2.1.e shows the result of filtering the 1-d signal (fig.2.1.a) convolved with $d_1[n]$ (fig.2.1.d). Now the output shows the highest magnitude output in the mid point where there is variation in the input signal.

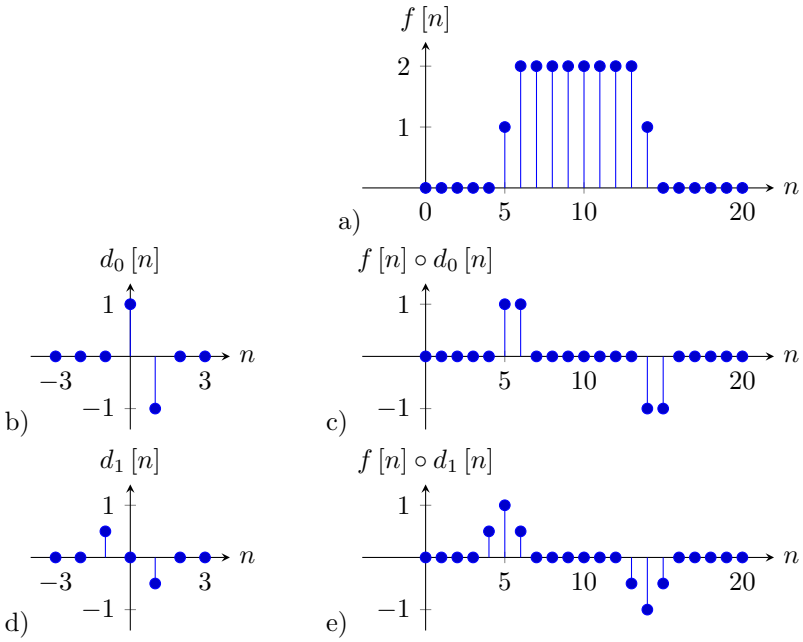

Figure 2.1: a) b).

It is also interesting to see the behavior of the derivative and its discrete approximations in the Fourier domain.

$$df(x)/dx \leftrightarrow jwF(w) \tag{2.3}$$

in the continuous Fourier domain, derivation can be done by multiplying by jw. The DFT of $d_0[n]$ is:

$$\begin{aligned} D_0[u] =& 1 - \exp\left(-2\pi j \frac{u}{N}\right) = \\ &= \exp\left(-\pi j \frac{u}{N}\right)\left(\exp\left(\pi j \frac{u}{N}\right) - \exp\left(-\pi j \frac{u}{N}\right)\right) \\ &= \exp\left(-\pi j \frac{u}{N}\right) 2j \sin(\pi u/N) \end{aligned} \tag{2.4}$$

the first term is a pure phase shift and it is responsible of the half a sample delay in the output. The second term is the amplitude gain and it can be approximated by a linear dependency on u for small u values.

The DFT of $d_1[n]$ is:

$$\begin{aligned} D_1[u] =& 1/2 \exp\left(2\pi j \frac{u}{N}\right) - 1/2 \exp\left(-2\pi j \frac{u}{N}\right) = \\ &= j \sin(2\pi u/N) \end{aligned} \tag{2.5}$$

Figure 2.3 shows the magnitude of $D_0[u]$ and $D_1[u]$ and compares it with $|2\pi u/N|$ which will be the ideal approximation to the derivative. The amplitude of $D_0[u]$ provides a better approximation to the ideal derivative, but the phase of $D_0[u]$ introduces a small shift in the output. On the other hand, $D_1[u]$ has no shift, but it approximates the derivative over a smaller range of frequencies. The output to $D_1[u]$ is smoother than the output to $D_0[u]$, and, in particular, $D_1[u]$ gives a zero output when the input is the signal $[1, -1, 1, -1, ...]$. In fact, we can see that $[1, 0, -1] = [1, -1] \circ [1, 1]$, and, therefore $D_1[u] = D_0[u] B_1[u]$, where $B_1[u]$ is the DFT of the binomial filter $b_1[n]$.

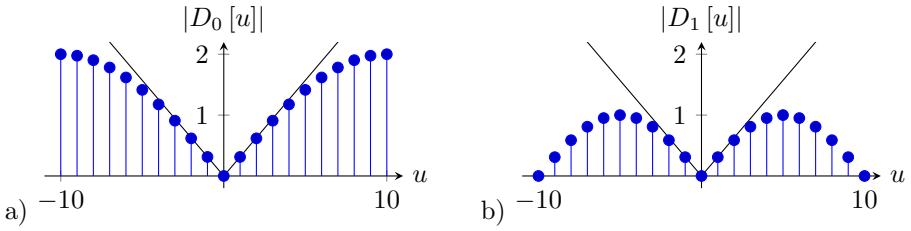

Figure 2.2: Magnitude of a) $D_0[u]$ and b) $D_1[u]$ and comparison with $|2\pi u/N|$, shown as a thin black line. Both DFT are computed over 20 samples.

Derivates have become an important tool to represent images and they can be used to extract a great deal of information from the image as it was shown in the previous chapter. One thing about derivatives is that it might seem as we are loosing information from the input image. An important question is: if we have the derivative of a signal, can we recover the original image? what information is being lost? Intuitively, we should be able to recover the input image by integrating its derivative, but it is an interesting exercise to look in detail how this integration can be performed. We will start with a one dimensional signal and then we will discuss the two dimensional case.

A simple way of seeing that we can recover the input from its derivatives is to write the derivative in matrix form. This is the matrix that corresponds to the convolution with the kernel $[1, -1]$ that we will call D_0. The next two matrices show the matrix D_0 and its inverse D_0^{-1} for a 1D image of length 5 pixels using zero boundary conditions:

$$D_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \qquad D^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{2.6}$$

We can see that the inverse D^{-1} is reconstructing each pixel as a sum of all the derivate values from the left-most pixel to the right. And the inverse perfectly reconstructs the input. But, this is cheating because the first sample of the derivative gets to see the actual value of the input signal and then we can integrate back the entire signal. That matrix is assuming zero boundary conditions for the signal and the boundary gives us the needed constraint to be able to integrate back the input signal.

But what happens if you only get to see differences and you remove any pixel that was affected by the boundary? In this case, the derivative operator in matrix form is:

$$D_0 = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \tag{2.7}$$

Let's consider the next 1D input signal:

$$g = [1, 1, 2, 2, 0] \tag{2.8}$$

then, the output of the derivative operator is:

$$f = D_0 g = [0, -1, 0, 2] \tag{2.9}$$

Note that this vector has one sample less than the input.

To recover g we can not invert D_0 as it is not a square matrix, but we can compute the pseudo inverse which turns out to be:

$$D_0^+ = \frac{1}{5}\begin{bmatrix} -4 & -3 & -2 & -1 \\ 1 & -3 & -2 & -1 \\ 1 & 2 & -2 & -1 \\ 1 & 2 & 3 & -1 \\ 1 & 2 & 3 & 4 \end{bmatrix} \tag{2.10}$$

It has a funny structure and it is easy to see how it can be written in the general form for signals of length N. Also note that this is not a convolution. Another important thing is that the inversion process is trying to recover more samples than there are observations. The trade off is that the signal that it will recover will have zero mean (so it looses one degree of freedom that can not be estimated). In this example, the reconstructed input is:

$$\hat{g} = D^+ f = [-0.2, -0.2, 0.8, 0.8, -1.2] \tag{2.11}$$

Note that $\sum \hat{g} = 0$ and that this is $\hat{g} = g - 1.2$, where 1.2 is the mean value of samples on g. Then, you still can recover the input signal up to the DC component.

In two dimensions things are a bit more complex. There are several ways in which partial image derivatives can be approximated. For instance, we can compute derivatives along the n and m components.

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad [1 \; -1] \tag{2.12}$$

Or we can use a rotated reference frame as it is done in the Robert-Cross operator, introduced in 1963 [] in a time when reading an image of 256×256 pixels into memory took several minutes:

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{2.13}$$

Although these operators are very old and better algorithms exist nowadays for edge extraction, when an efficient solution is needed, these simple operators are still very useful and had been used as key operators in modern computer vision descriptors such as HOG as we will see later.

2.2 Gaussian derivatives

In the previous section we studied how to discretize derivatives. However, computing derivatives in practice has several difficulties. First, derivatives are sensitive to noise. In the presence of noise, as images tend to vary slowly, the difference between two continuous pixel values will be dominated by noise. There are also situations in which the derivative of an image is not defined. For instance, consider an image in the continuous domain with the form: $\mathbf{I}(x, y) = 0$ if $x < 0$ and 1 otherwise. If we try to compute $\partial \mathbf{I}(x, y)/\partial x$ we will get 0 everywhere, but around $x = 0$ the value of the derivative is not defined. We avoided this issue in the previous section as for discrete images the approximation of the derivative is always defined.

Gaussian derivatives address these two issues. They where introduced by Koendering and Van Doorm [] as a model of the processing perform by neurons in the visual system.

Let's start with the following observation: For two functions defined in the continuous domain $f(x, y)$ and $g(x, y)$, we can write:

$$\frac{\partial f(x, y)}{\partial x} \circ g(x, y) = f(x, y) \circ \frac{\partial g(x, y)}{\partial x} \tag{2.14}$$

this is easy to prove in the Fourier domain. If our goal is to compute image derivatives and then blur the output using a low-pass filter, $g(x, y)$, then, instead of computing the

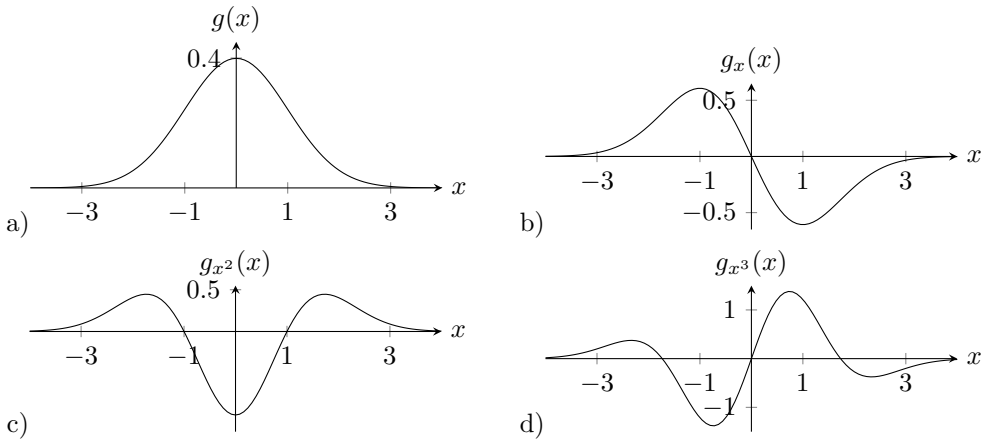

Figure 2.3: a) 1D Gaussian with $\sigma = 1$ and b-d) its derivatives up to order 3.

derivative of the image we can compute the derivatives of the filter kernel and convolve it with the image. Even if the derivative of f is not defined in some locations, we can always compute the result of this convolution.

If $g(x, y)$ is a blurring kernel it will smooth the derivatives reducing the output noise at the expense of a loss in spatial resolution. If g is a Gaussian, then the derivative is:

$$g_x(x, y; \sigma) = \frac{\partial g(x, y; \sigma)}{\partial x} = \frac{-x}{2\pi\sigma^4} \exp - \frac{x^2 + y^2}{2\sigma^2} = \frac{-x}{\sigma^2} g(x, y; \sigma) \tag{2.15}$$

and the second order derivative is:

$$g_{x^2}(x, y; \sigma) = \frac{x^2 - \sigma^2}{\sigma^4} g(x, y; \sigma) \tag{2.16}$$

Each derivative can be written as the product between a polynomial on x, with the same order as the derivative, times a gaussian. The family of polynomials that result on computing gaussian derivatives is called Hermite polynomials. The general expression for the n derivative of a gaussian is:

$$g_{x^n}(x; \sigma) = \frac{\partial^n g(x)}{\partial x^n} = \left(\frac{-1}{\sigma\sqrt{2}} \right)^n H_n \left(\frac{x}{\sigma\sqrt{2}} \right) g(x; \sigma) \tag{2.17}$$

the first Hermite polynomial is $H_0(x) = 1$, for $n = 0$ we have the original Gaussian. Figure 2.3 shows the 1D Gaussian derivatives.

In two dimensions, as the gaussian is separable, the partial derivatives result on the product of two Hermite polynomial, one for each dimension:

$$g_{x^n, y^m}(x, y; \sigma) = \frac{\partial^{n+m} g(x, y)}{\partial x^n \partial y^m} = \left(\frac{-1}{\sigma\sqrt{2}} \right)^{n+m} H_n \left(\frac{x}{\sigma\sqrt{2}} \right) H_m \left(\frac{y}{\sigma\sqrt{2}} \right) g(x, y; \sigma) \tag{2.18}$$

Figure 2.4 shows the 2D Gaussian derivatives, and Figure 2.5 shows the corresponding Fourier transforms.

The Gaussian derivatives share many of the properties of the Gaussian:

- The convolution of two Gaussian derivatives of order n and m and variances σ_1^2 and σ_2^2 results in another gaussian derivative of order $n + m$ and variance $\sigma_1^2 + \sigma_2^2$. Proving this property on the spatial domain can be tedious. However it is trivial to prove it in the Fourier domain.

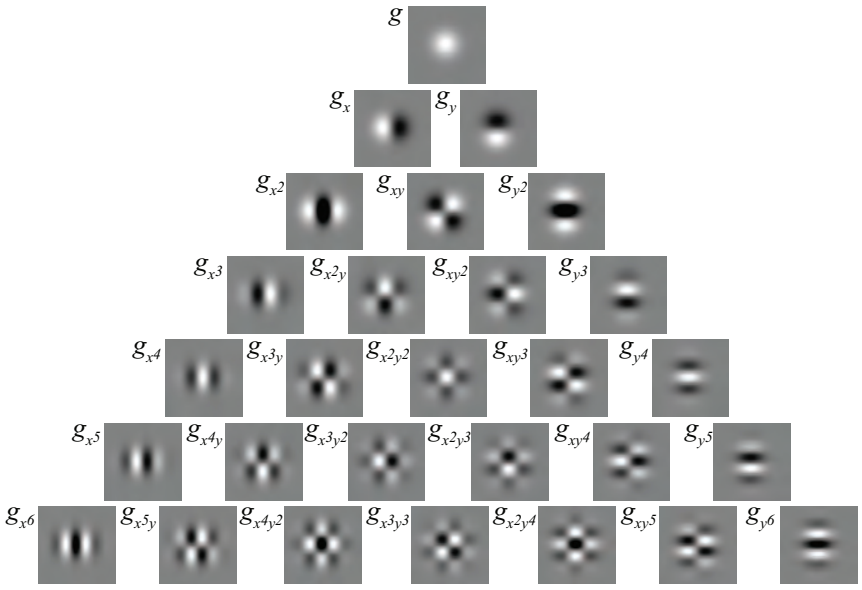

Figure 2.4: Gaussian derivatives up to order 6. All the kernels are separable. Although the seem similar to Fourier basis fig. 2.5 shows that they are different to sine and cosine waves, instead they look more like products of cosine and sine waves.

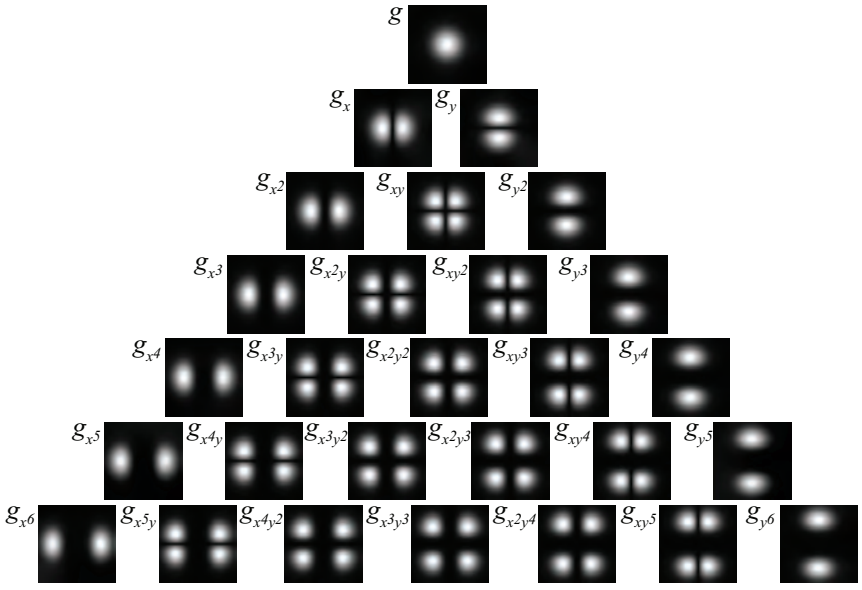

Figure 2.5: Fourier transform of the Gaussian derivatives shown in figure 2.4. Units are arbitrary.

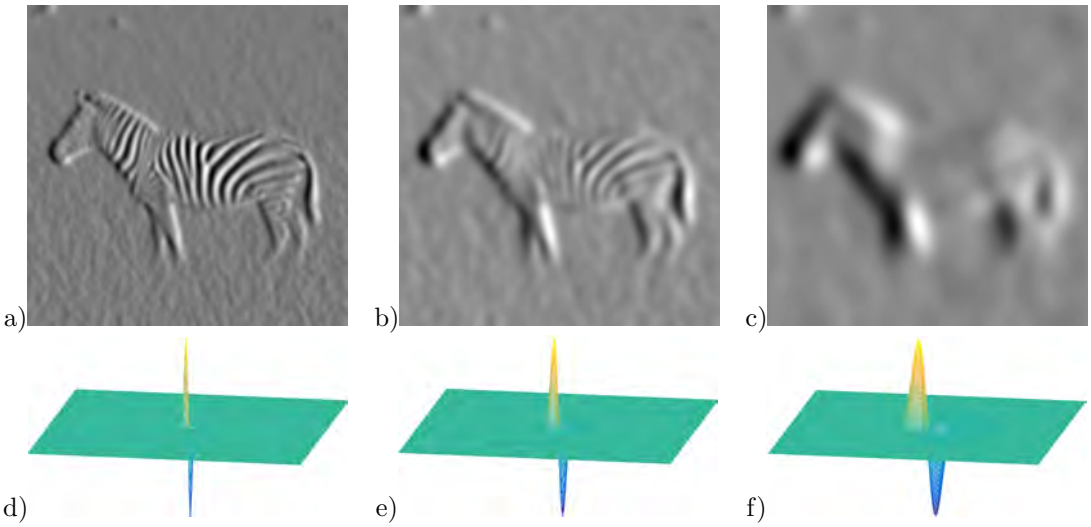

Figure 2.6: An image filtered with three Gaussian derivatives with standard deviations: a) $\sigma = 2$, b) $\sigma = 4$, and c) $\sigma = 8$. In the filtered images, bright pixels correspond to positive values and dark pixels correspond to negative values. Plots (d) - (f) show the three Gaussian derivatives over the same spatial support as the image. The discrete functions are approximated by sampling the continuous Gaussian derivatives. The convolutions are perform with mirror boundary conditions.

Figure 2.6 shows an image filtered with three gaussian derivatives with different widths. Interestingly, derivatives at different scales emphasize different aspects of the image. The fine-scale derivatives (fig. 2.6.a) highlight the bands in the zebra, while the coarse-scale derivatives (fig. 2.6.c) emphasize more the object boundaries. This multiscale image analysis will be studied in depth in the following chapter.

When processing images we have to use discrete approximations for the Gaussian derivatives. After discretization, many of the properties of the continuous Gaussian will not hold exactly.

There are many discrete approximations. For instance, we can take samples of the continuous functions. In practice it is common to use the discrete approximation given by the binomial filters. Figure 2.7 shows the result of convolving the binomial coefficients with: $b_n \circ [1, -1]$.

$$
\begin{array}{ll}
d_0 & \qquad\qquad\qquad\qquad\qquad\qquad 1 \qquad -1 \\
d_1 & \qquad\qquad\qquad\quad 1 \qquad\quad 0 \qquad -1 \\
d_2 & \qquad\qquad\quad 1 \qquad\quad 1 \qquad -1 \qquad -1 \\
d_3 & \qquad\qquad 1 \qquad\quad 2 \qquad\quad 0 \qquad -2 \qquad -1 \\
d_4 & \qquad\quad 1 \qquad\quad 3 \qquad\quad 2 \qquad -2 \qquad -3 \qquad -1 \\
d_5 & \qquad 1 \qquad\quad 4 \qquad\quad 5 \qquad\quad 0 \qquad -5 \qquad -4 \qquad -1
\end{array}
$$

Figure 2.7: Derivative of binomial coefficients resulting from convolving $b_n \circ [1, -1]$. The first two filters are the ones we have studied in detail in the previous section.

In two dimensions, we can use separable filters and build a partial derivative as:

$$Sobel_x = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \circ \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{2.19}$$

$$Sobel_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.20}$$

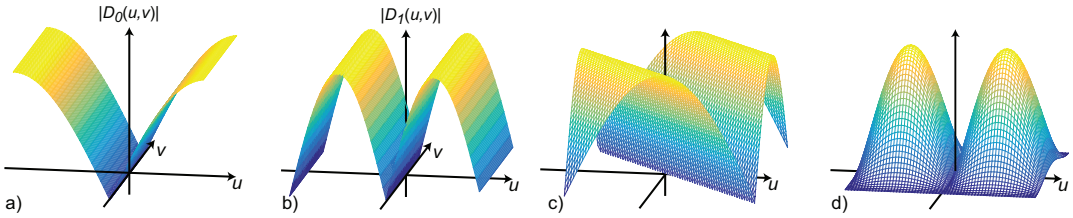

Figure 2.8: Magnitude of the DFT of four different discretization of Gaussian derivatives: a) d_0, b) d_1, c) Cross-Robert operator and d) Sobel-Feldman operator. As d_0 and d_1 are one dimensional their 2D-DFT vary only along one dimension. The Cross-Robert operator is similar to a rotated version of d_1. The Sobel-Feldman operator has the profile of D_1 along the axis $u = 0$ and it is proportional to the profile of B_2 along any section $v = constant$.

This particular filter is called the Sobel-Feldman operator. The goal of this operator was to be compact and to be as isotropic as possible. The sobel-feldman operator can be implemented very efficiently as it can be written as the convolution with 4 small kernels: $Sobel_x = b_1 \circ d_0 \circ b_1^T \circ b_1^T$. The DFT is:

$$Sobel_x [u, v] = D_1 [u] B_2 [v] = j \sin (2\pi u/N) (2 + 2 \cos (2\pi v/N)) \tag{2.21}$$

and it is shown in fig. 2.8.d. $N \times N$ is the extension of the domain (the operator is zero padded).

Fig. 2.8 compares the DFT of the 4 types of approximations of the derivatives that we have discussed. These operators are still very popular. *Sobel* has the best tolerance to noise due to its band-pass nature. The kernel d_0 is the one that provides the highest resolution in the output. Fig. 2.9 shows the output of different derivative approximations to a simple input image containing a circle. In the next section we will discuss now to use these derivatives to extract other interesting quantities.

2.3 Image gradient and directional derivatives

As we saw in chapter one, an important image representation is given by the image gradient. From the image derivatives we can define also the image gradient as the vector:

$$\nabla \mathbf{I} = \left(\frac{\partial \mathbf{I}}{\partial x}, \frac{\partial \mathbf{I}}{\partial y} \right) \tag{2.22}$$

For each pixel, the output is a two dimensional vector. In the case of using gaussian derivatives, we can write:

$$\nabla \mathbf{I} \circ g = (g_x(x, y), g_y(x, y)) \circ \mathbf{I} = \nabla g \circ \mathbf{I} \tag{2.23}$$

Although we have mostly computed derivatives along the x and y variables, we can obtain the derivative on any orientation as a linear combination of the two derivatives along the main axes. With $\mathbf{t} = (\cos(\theta), \sin(\theta))$, we can write the directional derivative a long the vector $\mathbf{t}$ as:

$$\frac{\partial \mathbf{I}}{\partial \mathbf{t}} = \nabla \mathbf{I} \cdot \mathbf{t} = \cos(\theta) \frac{\partial \mathbf{I}}{\partial x} + \sin(\theta) \frac{\partial \mathbf{I}}{\partial y} \tag{2.24}$$

In the gaussian case:

$$\frac{\partial \mathbf{I}}{\partial \mathbf{t}} \circ g = (\cos(\theta) g_x(x, y) + \sin(\theta) g_y(x, y)) \circ \mathbf{I} = (\nabla g \cdot \mathbf{t}) \circ \mathbf{I} = g_\theta(x, y) \circ \mathbf{I} \tag{2.25}$$

with $g_\theta(x, y) = \cos(\theta) g_x(x, y) + \sin(\theta) g_y(x, y)$. However, to compute the derivate a long any arbitrary angle θ does not require doing new convolutions. Instead, we can compute any

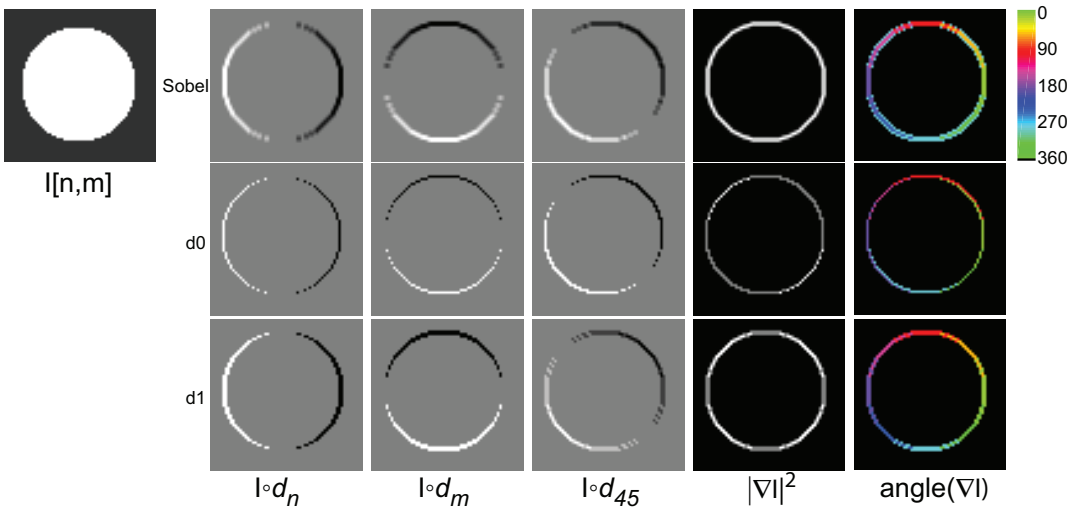

Figure 2.9: Image containing a circle filtered along the directions n, m and 45 degrees. And also the magnitude and the angle of the gradient. The angle is shown only where the magnitude is > 0. The derivative output along 45 degrees is obtained as a linear combination of the derivatives outputs along n and m. Check the differences among the different kernels. The Sobel operator gives the most rotationally invariant gradient magnitude at the expense of a bit of blurring of the output.

derivative as a linear combination of the output of convolving the image with $g_x(x, y)$ and $g_y(x, y)$:

$$\frac{\partial \mathbf{I}}{\partial \mathbf{t}} \circ g = \cos(\theta) g_x(x, y) \circ \mathbf{I} + \sin(\theta) g_y(x, y) \circ \mathbf{I} \tag{2.26}$$

When using discrete convolutional kernels $d_n[n, m]$ and $d_m[n, m]$ to approximate the derivatives along n and m, it can be written as:

$$\nabla \mathbf{I} = (d_n[n, m], d_m[n, m]) \circ \mathbf{I} \tag{2.27}$$

and

$$\nabla \mathbf{I} \cdot \mathbf{t} = d_\theta[n, m] \circ \mathbf{I} \tag{2.28}$$

with $d_\theta[n, m] = \cos(\theta) d_n[n, m] + \sin(\theta) d_m[n, m]$. We expect that the linear combination of these two kernels should approximate the derivative in the direction θ. The quality of this approximation will vary for the different kernels we have seen in the previous sections.

2.4 Image laplacian

The Laplacian filter was made popular by Marr and Hildreth in 1980 [] in the search for operators that locate the boundaries between objects. The Laplacian is a common operator from differential geometry to measure the divergence of the gradient and it appears frequently in modeling fields in physics. One cool example is the paper "can one hear the shape of a drum" [] where the laplacian is used for modeling vibrations in a drum and the sounds it produces as a function of its shape, those results have applications also in spectral methods for image segmentation as we will see later.

The Laplacian operator is defined as the sum of the second order partial derivatives of a function:

$$\nabla^2 \mathbf{I} = \frac{\partial^2 \mathbf{I}}{\partial x^2} + \frac{\partial^2 \mathbf{I}}{\partial y^2} \tag{2.29}$$

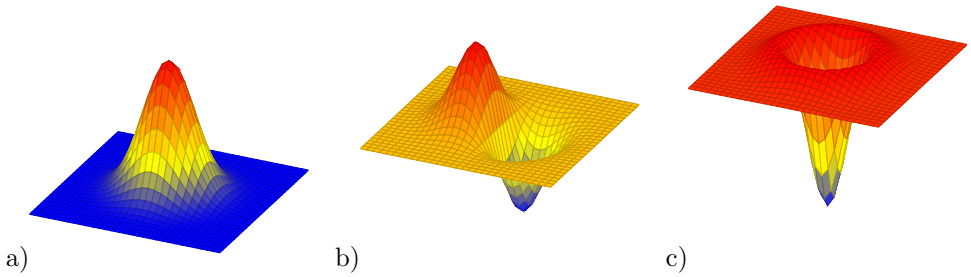

Figure 2.10: Three 2D gaussian-related convolutional kernels: a) 2D gaussian with $\sigma = 1$, b) its x derivative, and c) its laplacian.

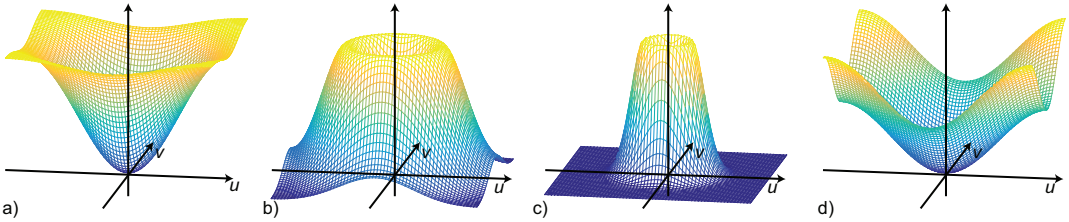

Figure 2.11: Magnitude of the DFT of the Gaussian Laplacian with three different widths and a discrete approximation: a) $\sigma = 1/2$, b) $\sigma = 1$, c) $\sigma = 2$ and d) DFT of the five-point discrete approximation to the laplacian.

The laplacian is more sensitive to noise than the first order derivatives. Therefore, in the presence of noise, it is useful to filter the output with a gaussian. We can write:

$$\nabla^2 \mathbf{I} \circ g = \nabla^2 g \circ \mathbf{I} \tag{2.30}$$

Where

$$\nabla^2 g = \frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} g(x, y) \tag{2.31}$$

Figure 2.10 compares the 2D gaussian laplacian (fig. 2.10.c) with the first order derivative (fig. 2.10.b) of a 2D gaussian (fig. 2.10.a). Due to its shape, the laplacian is also called the inverted mexican hat wavelet. Despite that visually it might seem as if the laplacian has a negative mean the mean is actually zero as the positive side is wider than the negative. Figure 2.11 shows the DFT of the gaussian laplacian for three different values of σ. For values $\sigma > 1$ the resulting filter is band-pass.

In the discrete domain there are several approximations to the laplacian filter. Lindeberg [] explored different approximations to the image Laplacian in order to optimize its properties. In one dimension, the Laplacian can be approximated by $[1, -2, 1]$ which is the result of the convolution of $[1, -1] \circ [1, -1]$. In two dimensions, the most popular approximation is the five-point formula which consists in convolving the image with the kernel:

$$\nabla^2_5 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{2.32}$$

It involves the central pixel and its four nearest neighbors. This is a sum-separable kernel: it corresponds to approximating the second order derivative for each coordinate and summing the result (i.e., convolve the image with $[1, -2, 1]$ and also with its transpose and summing the two outputs. It is important to shift by one pixel, on the appropriate dimension, the two results to make sure that the sum results in the Laplacian.) Figure 2.10.d shows the DFT

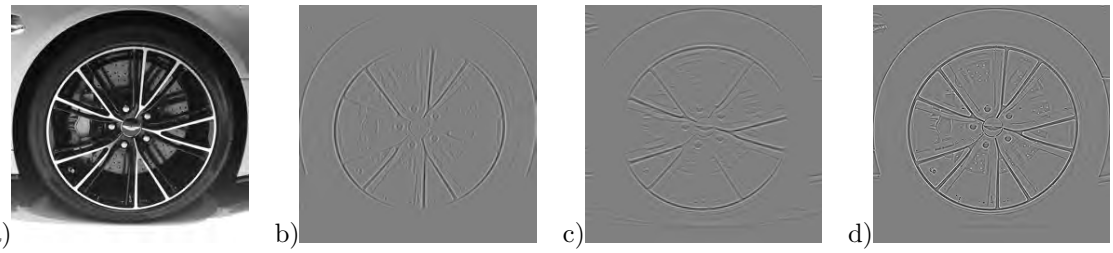

Figure 2.12: a) Input image, b) second order derivative along x, c) second order derivative along y and d) the sum of the two second order derivatives, which results in the laplacian output.

of this approximation. It also has a quadratic from near the origin of the frequency domain, but it has a high-pass shape similar to the one obtained for a small value of σ.

Note that when summing two first order derivatives along x and y, what we obtain is a first order derivative along the 45 degrees angle. However, when summing up the two second order derivatives we obtain a rotationally invariant kernel. This is shown in fig. 2.12

The discrete approximation also provides a better intuition of how it works than its continuous counterpart. The Laplacian filter has a number of advantages with respect to the gradient:

- It is rotationally invariant: it is a linear operator that responds equally to edges in any orientation (this is only approximate in the discrete case).
- It measures curvature: if the image contains a linear trend the derivative will be non-zero despite being no boundaries, while the laplacian will be zero.
- Edges can be located as the zero-crossings in the Laplacian output. However, this way of detecting edges is not very reliable.
- Zero crossings of an image form closed contours.

Marr-Hildreth [] used zero-crossings of the laplacian output to compute edges, but this method is not used nowadays for edge detection. Instead the laplacian filter is widely used in a number of other image representations. It is used to build image pyramids (multiscale representations, chapter **??**), to detect points of interest in images (it is the basic operator used to detect keypoints where to compute SIFT descriptors, which we will discuss in chapter **??**).

Figure 2.13 compares the output of the first order derivative (Fig. 2.13.b) and the laplacian (Fig. 2.13.d) on a simple 1D signal (Fig. 2.13.a). The local maximum of the derivative output (Fig. 2.13.c) and the zero crossings of the laplacian output (Fig. 2.13.e) are aligned with the transitions (boundaries) of the input signal (Fig. 2.13.a).

The Laplacian filter can also be used as a coarse approximation of the behavior of the early visual system. When looking at the magnitude of the DFT of the laplacian with $\sigma = 1$ (fig. 2.11), the shape seems reminiscent of our subjective evaluation of our own visual sensitivity to spatial frequencies when we look at the Campbell & Robson chart (fig. 1.1). As the visual filter does not seem to cancel exactly the very low spatial frequencies as the laplacian does, a better approximation is:

$$h = \nabla^2 g + \lambda g \tag{2.33}$$

where h is the approximate impulse response of the human visual system, λ is a small constant that is equal to the DC gain of the filter. This particular form of the human sensitivity function helps to explain some visual illusions. One of them is shown in figure 2.14. This visual illusion is called the Vasarely visual illusion. Images (a) and (d) show two gray-scale pyramids formed by superimposing squares of increasing (or decreasing) intensity.

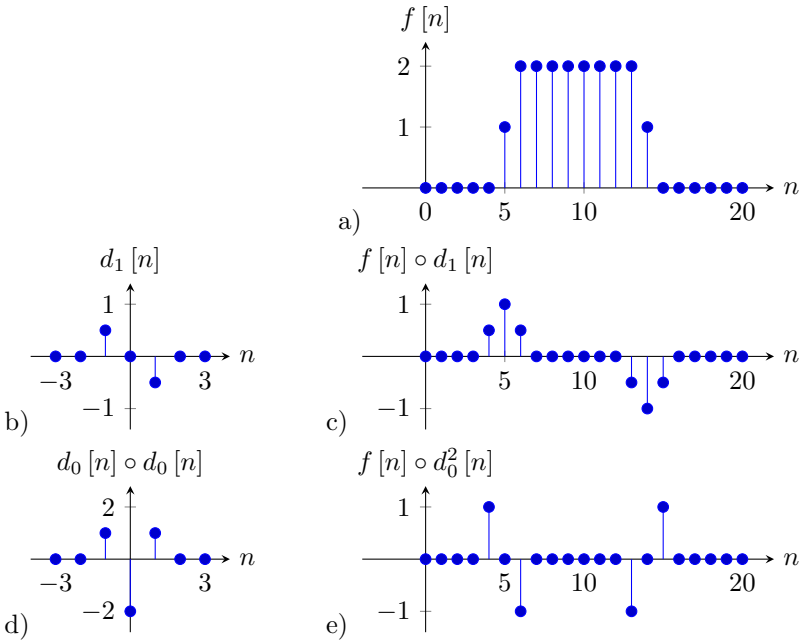

Figure 2.13: Comparison between the output of a first order derivative and the Laplacian of 1D signal. a) Input signal, b) kernel d_1, c) output of the derivative (convolution of (a) and (b)), d) discrete approximation of the Laplacian and e) output of convolving the signal (a) with the laplacian kernel (d).

When looking at them we perceive the diagonal directions as being brighter (a) or darker (d) than their neighborhood. This is an illusion because there is not such a difference in image intensity. One explanation consists in saying that the image that we perceive is the output of a filter (as shown in eq. 2.33). Images (b) and (e) show the output of such a filter. We can see the diagonals again as being brighter or darker but now this effect is not an illusion, the brighter and darker diagonals are really part of the filtered image. In fact, (c) shows in blue an horizontal section at one quarter of image (b). The red curve is the section of the input image (a). We can see that the output really contains a pick in intensity on the diagonals of image (b). The plot in figure (f) shows the same for images (d) and (e).

2.5 Sharpening filter

One example of a simple but very useful filter is a sharpening filter. The goal of a sharpening filter is to transform an image so that it appears sharper. This can be achieved by amplifying the amplitude of the high-spatial frequency content of the image. We can achieve this with a combination of filters that we have already discussed in this section.

A simple way to design a *sharpening* filter is to de-emphasize the blurry components of an image. By the linearity of the convolution operator, we're allowed to add and subtract kernels to make a new kernel that would give us the same filtered image as if we had added and subtracted the filtered outputs of each of the component kernels. For this example, we start with twice the original image (sharp plus blurred parts), then subtract away the blurred components of the image:

$$\text{sharpening filter} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.34}$$

Note that the DC gain of this sharpening filter is 1. That would leave one original image in

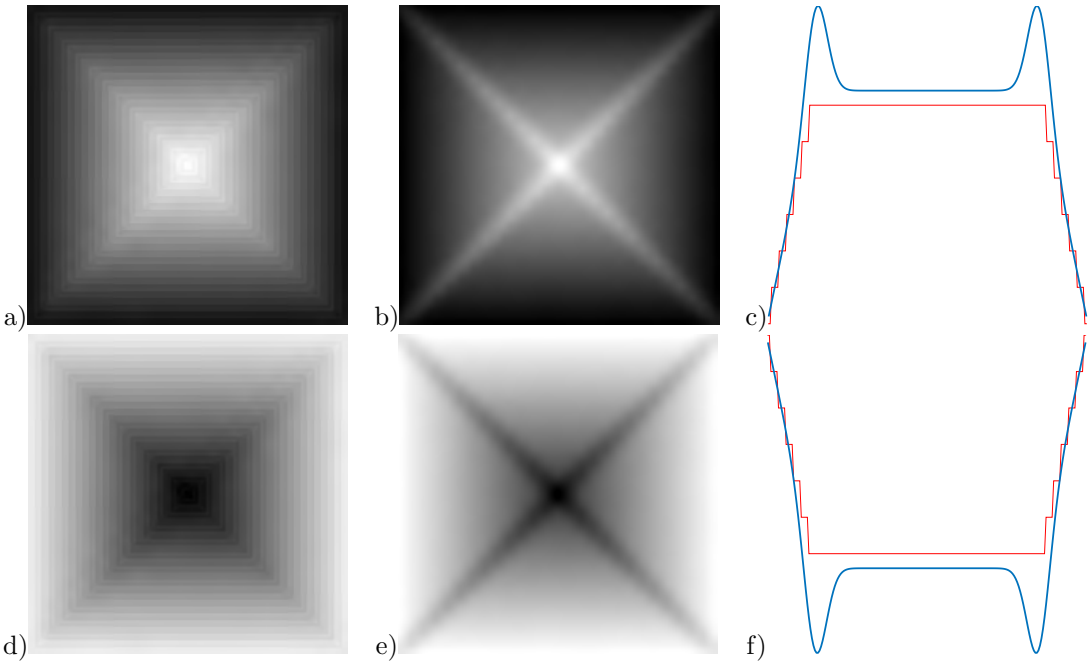

Figure 2.14: Vasarely visual illusion. Images (a) and (d) show two gray-scale pyramids. When looking at them we perceive the diagonal directions as being brighter (a) or darker (d) than their neighborhood. Images (b) and (e) show the output of the human model given by the filter from eq. 2.33. The plots (c) and (f) show, in blue, an horizontal section at one quarter of image (b) and (e) showing the intensity profile as a function of x. The red curve is the section of the corresponding input image. We can see that the output really contains a pick in intensity on the diagonals of the input image.

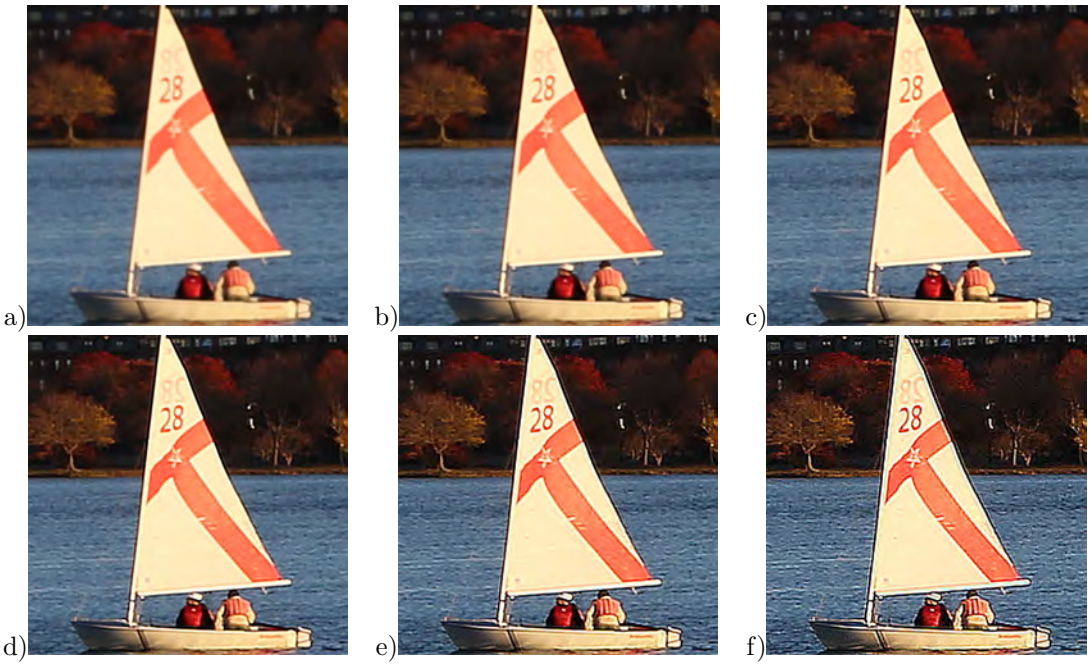

Figure 2.15: Sharpening achieved by subtraction of blurred components. a) Original image, b) sharpened once by filtering with kernel from eq. 2.34. Each color channel is filtered independently. c) - f) the same filter is applied successively to the previous output. In the last image the sharpening filter has been applied 5 times to the input image. The last image seems looks substantially sharper than the original image, but close inspection will reveal some artifacts.

there, plus an additional component of the sharp details. The perceptual result is that of a sharpened image, Fig. 2.15. We can apply this filter successively in order to further enhance the image details. If too much sharpening is applied we might end up enhancing noise and introducing image artifacts.