Chapter 3

Image filtering

Filtering an image involves transforming the values of the pixels by taking into account the values of the neighboring pixels. Filtering is a basic concept of signal and image processing, and it often forms the preprocessing step that transforms the raw input data into a more useful form for later algorithms.

3.1 Convolution

Suppose we have a one-dimensional discrete signal $f(x)$. The **convolution** of f with a **kernel** g is defined as

$$h(x) = f(x) * g(x) = \sum_{i=0}^{w-1} f(i)g(x-i), \tag{3.1}$$

where the kernel is of width w, so that $g(x) = 0$ for $x < 0$ or $x \geq w$. We can think of this computation as a system which operates on the input f to produce the output h. Convolution exactly describes the process of a linear shift-invariant system, in which case the system can be completely described by g, which is called the *impulse response*.

To ensure that the kernel has an unambiguous center, we will assume that the kernel has an odd number of elements. In such a case, the width w is related to the half-width $\tilde{w}$ by $w = 2\tilde{w} + 1$. For example, if g is a three-element kernel, then it is defined for $x \in \{0, 1, 2\}$, and the width and half-width are $w = 3$ and $\tilde{w} = 1$, respectively. Note that $\tilde{w}$ is also the zero-based index of the center element of the kernel. Implementing the one-dimensional convolution of an input signal f of width n with a kernel g of width w is straightforward:

```
CONVOLVE(f, g)
1  w̃ ← ⌊(w − 1)/2⌋
2  for x ← 0 to n − 1 do
3        val ← 0
4        for i ← 0 to w − 1 do
5              val ← val +g[i] ∗ f[x + w̃ − i]
6        h[x] ← val
7  return h
```

Notice that this code actually computes $g*f$, but it is easy to show that convolution is commutative, so that $f*g = g*f$. Be aware that the asterisk has a different meaning in the code, where $*$ denotes *multiplication*, than it does in the text, where $*$ denotes *convolution*. This reuse of notation is an unfortunate consequence of having a finite number of symbols at our disposal. Note also that we could just as easily have used the term *length* instead of *width* for the signal and kernel; with 1D functions

the terms are interchangeable. The floor operation in Line 1 is not necessary as long as the width of the kernel is odd.

Three additional points should be noted about this code. First, while g is actually defined for $n + w - 1$ values according to Equation (3.1), where n is the width of f, we adopt the common image processing practice of setting the size of the output to the same as that of the input. Secondly, near the two ends of the signal, the computation uses indices that exceed the size of f. Such out-of-bounds accessing must be avoided using one of the methods (zero pad the border, resize the kernel, or hallucinate out-of-bounds values) mentioned in Section 2.4.1 in the context of morphological operations; such details have been omitted to avoid cluttering the code. Finally, convolution must never be done in place: if f and h are stored in the same place in memory, then the values computed for h will corrupt those being read from f.

The extension to two dimensions is straightforward:

$$h(x,y) = f(x,y) * g(x,y) = \sum_{i=0}^{w-1} \sum_{j=0}^{h-1} f(i,j) g(x-i, y-j), \tag{3.2}$$

where w and h are the width and height of the kernel, respectively. To convolve an image with a 2D kernel, simply flip the kernel about both the horizontal and vertical axes, then slide the kernel along the image, computing the sum of the elementwise multiplication at each pixel between the kernel and the input image.

The relationship between the input f and the output h depends upon the type of the kernel g. Two types of kernels are common. *Smoothing kernels*, in which all the elements of the kernel sum to one, $\sum_i g_i = 1$, perform an averaging of the values in a local neighborhood and therefore reduce the effects of noise. *Differentiating kernels*, in which all the elements of the kernel sum to zero, $\sum_i g_i = 0$, accentuate the places where the signal is changing rapidly in value and are therefore useful to extract information. Smoothing kernels are therefore low-pass filters, whereas differentiating kernels are high-pass filters. We will consider these in turn.

3.2 Smoothing by convolving with a Gaussian

The simplest smoothing kernel is the box filter, in which every element of the kernel has the same value. Since the elements must sum to one, the elements of a box filter of length w are each $1/w$. Some examples of box filters are $\frac{1}{3}[1 \quad 1 \quad 1]$, and $\frac{1}{5}[1 \quad 1 \quad 1 \quad 1 \quad 1]$. In practice, kernels are often selected in practice to have an odd length to avoid undesired shifting of the output.

Convolving a box filter with itself yields an approximation to a Gaussian kernel. In this manner, the simplest non-trivial box filter leads to

$$\frac{1}{2}[1 \quad 1] * \frac{1}{2}[1 \quad 1] = \frac{1}{4}[1 \quad 2 \quad 1], \tag{3.3}$$

which is a kernel with $w = 3$, $\tilde{w} = 1$, $g[0] = \frac{1}{4}$, $g[1] = \frac{1}{2}$, and $g[2] = \frac{1}{4}$. This kernel approximates a Gaussian with $\sigma^2 = \sum_{i=0}^{2} g[i](i - \tilde{w})^2 = \frac{1}{4}\left(1 \cdot 1^2 + 0 + 1 \cdot 1^2\right) = \frac{1}{2}$. An additional iteration yields

$$\frac{1}{4}[1 \quad 2 \quad 1] * \frac{1}{4}[1 \quad 2 \quad 1] = \frac{1}{16}[1 \quad 4 \quad 6 \quad 4 \quad 1], \tag{3.4}$$

which approximates a Gaussian with $\sigma^2 = \sum_{i=0}^{4} g[i](i - \tilde{w})^2 = \frac{1}{16}\left(1 \cdot 2^2 + 4 \cdot 1^2 + 0 + 4 \cdot 1^2 + 1 \cdot 2^2\right) = 1$. These Gaussians can be easily remembered as the odd rows of Pascal's triangle (binomial triangle), with the $(2a+1)$th row approximating a Gaussian with $\sigma^2 = a/2$. Similarly, the $(a+1)$th row of the trinomial triangle approximates a Gaussian with $\sigma^2 = \frac{2a}{3}$. See Figure 3.1. For example, $\frac{1}{3}[1 \quad 1 \quad 1]$ approximates a Gaussian with $\sigma^2 = \frac{2}{3}$, while

$$\frac{1}{3}[1 \quad 1 \quad 1] * \frac{1}{3}[1 \quad 1 \quad 1] = \frac{1}{9}[1 \quad 2 \quad 3 \quad 2 \quad 1] \tag{3.5}$$

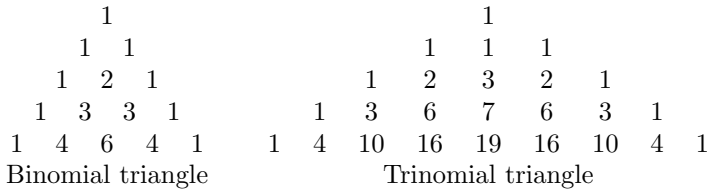

Figure 3.1: Binomial and trinomial triangles for constructing Gaussian kernels.

approximates a Gaussian with $\sigma^2 = \frac{4}{3}$.

The Gaussian function, which we shall represent with a capital G, is ubiquitous because of its many unique and desirable properties. The Gaussian shape accomplishes the optimal tradeoff between being localized in space and in frequency. The Fourier transform of a Gaussian is a Gaussian: $\mathcal{F}\{G\} = G$. The convolution of a Gaussian is a Gaussian. A Gaussian PDF has higher entropy than any other PDF with the same variance, therefore any operation on a PDF which discards information but conserves variance leads inexorably to a Gaussian. The central limit theorem — which says that the sum of any PDFs with finite variance tends toward a Gaussian — is the best known example of this.

We can construct a two-dimensional kernel by convolving two one-dimensional kernels (one vertical and one horizontal):

$$\frac{1}{3}\begin{bmatrix}1\\1\\1\end{bmatrix} * \frac{1}{3}\begin{bmatrix}1 & 1 & 1\end{bmatrix} = \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\1 & 1 & 1\\1 & 1 & 1\end{bmatrix}.$$

Although convolution itself is commutative, writing the vertical kernel first provides a helpful visual aid, because the result is the same as one would get by treating the operation as a matrix multiplication of the two vectors (i.e., the outer product of the two vectors).

When a 2D kernel can be decomposed into the convolution of two 1D kernels, we say that the kernel is **separable**. Every 2D Gaussian kernel is separable, which can be seen by applying the law of exponents to the convolution of an arbitrary 2D signal $f(x, y)$ and a 2D Gaussian $G(x, y)$. Ignoring the normalization factor for simplicity, we see that convolving f with $G(x, y)$ is the same as convolving f with a vertical 1D Gaussian kernel $G(y)$, followed by a horizontal 1D Gaussian kernel $G^T(x)$ (or vice versa, since the order does not matter), where the superscript T denotes transpose:

$$\begin{aligned} f(x,y)*G(x,y) &= \sum_i \sum_j f(x-i, y-j) \exp\left\{\frac{-(i^2+j^2)}{2\sigma^2}\right\} && (3.6)\\ &= \sum_i \left[\sum_j f(x-i, y-j) \exp\left\{\frac{-j^2}{2\sigma^2}\right\}\right] \exp\left\{\frac{-i^2}{2\sigma^2}\right\} && (3.7)\\ &= [f(x,y)*G(y)]*G^T(x). && (3.8)\end{aligned}$$

As an example,

$$f * \left(\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{bmatrix}\right) = \left(f * \frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix}\right) * \left(\frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix}\right)$$

because

$$\frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix} * \frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix} = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{bmatrix}.$$

A discrete kernel is separable if and only if all of its rows and columns are linearly dependent (i.e., scalar multiples of one another), meaning that the kernel (viewed as a matrix) is rank 1.

If the 2D kernel is of size $w \times w$, then the amount of computation in separable convolution is $O(2w)$ rather than $O(w^2)$, which is a significant savings in computation over the full 2D convolution. Separable convolution is performed as follows, using the 1D kernel G, which is the same for both horizontal and vertical operations:

```
ConvolveSeparable(I, g_h, g_v)
 1  ; convolve horizontal
 2  for y ← 0 to height − 1 do
 3        for x ← w̃ to width − 1 − w̃ do
 4              val ← 0
 5              for i ← 0 to w − 1 do
 6                    val ← val + G[i] ∗ I(x + w̃ − i, y)
 7              tmp(x, y) ← val
 8  ; convolve vertical
 9  for y ← w̃ to height − 1 − w̃ do
10        for x ← 0 to width − 1 do
11              val ← 0
12              for i ← 0 to w − 1 do
13                    val ← val + G[i] ∗ tmp(x, y + w̃ − i)
14              out(x, y) ← val
15  return out
```

This code assumes that the width of both kernels is the same, so that the resulting 2D kernel is square, which is nearly always true in practice. Note that convolution requires a temporary image to store the result of the first convolution, since convolution must not be done in place. Note also that it is critical to convolve every row of the image for a horizontal kernel, and every column of the image for a vertical kernel. This is because the second convolution uses values computed in the first convolution. With a little extra work to handle out-of-bounds pixels, the values for all the pixels in the output image can be computed.

3.3 Constructing Gaussian kernels

To construct a one-dimensional Gaussian kernel with an arbitrary σ, we simply sample the continuous zero-mean Gaussian function $G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right)$ and then normalize by dividing each element by the sum $\sum_i G[i]$ of all the elements. This sum is the zeroth moment of the signal. Since $G(x)$ has zero mean, but our discrete Gaussian kernel $G[i]$ is centered around $i = \tilde{w}$, we must subtract $\tilde{w}$ from the index while constructing the kernel.

```
CreateGaussianKernel(σ)
1  w̃ ← GetKernelHalfWidth(σ)
2  w ← 2w̃ + 1
3  sum ← 0
4  for i ← 0 to w − 1 do
5        G[i] ← exp(−(i − w̃) ∗ (i − w̃)/(2 ∗ σ ∗ σ))
6        sum ← sum + G[i]
7  for i ← 0 to w − 1 do
8        G[i] ← G[i]/ sum
9  return G
```

Note that the continuous normalization factor $\frac{1}{\sqrt{2\pi\sigma^2}}$, which ensures $\int_{-\infty}^{\infty} G(x)\, dx = 1$ in the continuous domain, can be ignored since it disappears anyway when the discrete normalization step is performed.

kernel	discrete σ^2	continuous σ^2	error
$G_{0.25} = \frac{1}{8}[\,1 \quad 6 \quad 1\,]^T$	0.25	0.28	10.7%
$G_{0.333} = \frac{1}{6}[\,1 \quad 4 \quad 1\,]^T$	0.333	0.36	7.4%
$G_{0.375} = \frac{1}{16}[\,3 \quad 10 \quad 3\,]^T$	0.375	0.42	10.7%
$G_{0.5} = \frac{1}{4}[\,1 \quad 2 \quad 1\,]^T$	0.5	0.72	30.6%
$G_{1.0} = \frac{1}{16}[\,1 \quad 4 \quad 6 \quad 4 \quad 1\,]^T$	1.0	1.17	14.5%

Table 3.1: The discrete Gaussian kernel has, in general, a different variance from the underlying continuous function from which it was sampled. Shown are the differences for several different values of σ^2.

The discrete normalization step cannot be ignored because the continuous normalization factor alone will not ensure that $\sum_i G[i] = 1$, due to discretization effects. The property $\sum_i G[i] = 1$ is important to ensure that the overall brightness of the image does not change as a result of the smoothing. Another way to look at this is that the smoothing of a constant image should not change the image.

A reasonable implementation of the function in the first line of the code is given by the following.

```
GetKernelHalfWidth(σ)
1  return Round(2.5σ − 0.5)
```

To derive this expression, note that the central sample $G[\tilde{w}]$ in the discrete Gaussian approximates the region between $x = -0.5$ and $x = 0.5$, the adjacent sample $G[\tilde{w}+1]$ approximates the region between $x = 0.5$ and $x = 1.5$, and an arbitrary sample $G[\tilde{w}+k]$ approximates the region between $x = k - 0.5$ and $x = k + 0.5$. Since $w - 1 = 2\tilde{w} = \tilde{w} + \tilde{w}$, the final sample $G[w-1]$ approximates the region between $x = \tilde{w} - 0.5$ and $x = \tilde{w} + 0.5$. Therefore, a kernel of width w approximately captures the area

$$\int_{-\tilde{w}-0.5}^{\tilde{w}+0.5} G(x)\; dx$$

under the original continuous Gaussian function. Since 98.76% of the area under a Gaussian is captured from -2.5σ to 2.5σ, we set $\tilde{w} + 0.5 = 2.5\sigma$ to capture this area. This yields the desired expression, which results in a kernel that well approximates a Gaussian. However, if a less accurate approximation is acceptable, then the 2.5 factor multiplying the standard deviation can be reduced accordingly.

Let G_{σ^2} refer to a vertical 1D Gaussian kernel with variance σ^2. Then some common Gaussian kernels are as follows, where the superscript T indicates transpose:

$$\begin{aligned}
G_{0.25} &= \frac{1}{8}[\,1 \quad 6 \quad 1\,]^T \\
G_{0.333} &= \frac{1}{6}[\,1 \quad 4 \quad 1\,]^T \\
G_{0.375} &= \frac{1}{16}[\,3 \quad 10 \quad 3\,]^T \\
G_{0.5} &= \frac{1}{4}[\,1 \quad 2 \quad 1\,]^T \\
G_{1.0} &= \frac{1}{16}[\,1 \quad 4 \quad 6 \quad 4 \quad 1\,]^T .
\end{aligned}$$

Note that GetKernelHalfWidth returns the correct lengths for these variances, which are computed in the same manner described before. However, the variance of the discrete Gaussian kernel will in general be different from that of the underlying continuous Gaussian function that was sampled. From Table 3.1, we see that this difference in variance can be as high as 30%.

3.4 Evaluating Gaussian kernels*

Another way to look at this problem is to recognize that, in order to capture a Gaussian faithfully, the width of the kernel must be large enough — relative to σ — to capture most of the Gaussian. Some have argued [Trucco-Verri, p. 61] that it is not possible to build a faithful Gaussian kernel with just three samples ($w = 3$). The argument is based on conflicting constraints: Only a narrow (small σ) Gaussian will be accurately represented by just three samples, but a narrow Gaussian in the spatial domain leads to a wide Gaussian (large σ') in the frequency domain, leading to aliasing. This is because the Fourier transform of a Gaussian is $\mathcal{F}\left\{\exp(-\frac{x^2}{2\sigma^2})\right\} \propto \exp\left(-\frac{\omega^2}{2(1/\sigma)^2}\right)$, where ω is the angular frequency and $\sigma' = 1/\sigma$ is the standard deviation of the Fourier transformed signal. To see this numerically, recall from the definition of a Gaussian that 68.27% of the Gaussian is captured in the region $x \in [-\sigma, \sigma]$, 95.45% in the region $x \in [-2\sigma, 2\sigma]$, and 98.76% in the region $x \in [-2.5\sigma, 2.5\sigma]$. If we select the latter choice, then this yields the constraint $w \geq 5\sigma$, where w is the width of the kernel, since the region $[-2.5\sigma, 2.5\sigma]$ has a width of 5σ. Now because the sampling frequency is 1 sample per pixel, Nyquist's sampling theorem says the cutoff frequency is 0.5, implying a cutoff *angular* frequency of $2\pi(0.5) = \pi$. To keep 98.76% of the area of the Fourier transform between $-\pi$ and π, we must therefore (assuming w=3) set $2\pi \geq 5\sigma' = 5(1/\sigma)$, which leads to $\sigma \geq \frac{5}{2\pi} = 0.8$. Puting the spatial constraint $w \geq 5\sigma$ together with the frequency constraint $\sigma \geq 0.8$ implies $w \geq 5$, assuming w is odd.

However, such a conclusion is unwarranted. The 3-element kernel is widely used in practice, and for good reason. A more appropriate question to ask is, What is the best that a 3-element kernel can do? Instead of requiring that the kernel capture 98.76% of the area of the Gaussian, let us seek a value for σ that maximizes the area preserved in both the spatial and frequency domains. Let α represent the value such that this preserved area is in the region $x \in [-\alpha\sigma, \alpha\sigma]$, so that $w = 2\alpha\sigma$. The same area is captured in the frequency domain when $2\pi = 2\alpha\sigma'$, or $\sigma = 1/\sigma' = \alpha/\pi$. Since we are interested in the case $w = 3$, we can solve these equations $\alpha\sigma = 1.5$ and $\alpha/\sigma = \pi$ for the two unknowns to yield $\alpha = \sqrt{1.5\pi} \approx 2.17$ and $\sigma = \sqrt{1.5/\pi} \approx 0.69$. From the definition of the Gaussian, this value of $\alpha = 2.17$ implies that 97% of the Gaussian is captured in both the spatial and frequency domains, which is quite acceptable. Moreover, it is interesting to note that this particular value of $\sigma^2 = 0.48$ ($\sigma = 0.69$) is very close to the $\sigma^2 = 0.5$ of the 3×1 kernel obtained by the binomial triangle. Therefore, according to the criterion of balancing the area under the spatial- and frequency-domain signals, $\sigma = 0.69$ is the optimal 3×1 Gaussian kernel, which is closely approximated by $G_{0.5} = \frac{1}{4}[\,1 \quad 2 \quad 1\,]$.

Now that we have seen that the best 3-element Gaussian kernel is very close to the one given by the binomial triangle, let us analyze the other kernels in the triangle. Table 3.2 shows the first few of these kernels, along with the value for α (which indicates the preserved region $x \in [-\alpha\sigma, \alpha\sigma]$) and the area under the Gaussian curve (obtained using a Gaussian Q-function table). While the Gaussian is faithfully preserved in all cases, the binomial kernel is wider than it needs to be for increasing values of σ. For example, with $\sigma = 1.41$, a width of $w = 7$ would capture nearly 98.76% of the curve (since $5\sigma \approx 7$), but the binomial triangle uses $w = 9$ to represent this Gaussian. The trinomial triangle results in more compact Gaussians, as can be seen from the bottom portion of the table.

Now let us examine how faithfully the procedure GetKernelHalfWidth captures the corresponding Gaussian. The formula in the procedure is $\tilde{w} = \text{Round}(2.5\sigma - 0.5)$. Therefore, this procedure will output halfwidth $\tilde{w}$ if and only if $\tilde{w} - 0.5 \leq 2.5\sigma - 0.5 < \tilde{w} + 0.5$, assuming values halfway between integers round up. Solving for σ yields $\tilde{w}/2.5 \leq \sigma < (\tilde{w} + 1)/2.5$. Since $w = 2\tilde{w} + 1$, we can solve for $\alpha = w/2\sigma$, which corresponds to the preserved region $x \in [-\alpha\sigma, \alpha\sigma]$. Table 3.3 shows the minimum and maximum values of α for each odd width w, along with the minimum and maximum values of the area under the curve (obtained using a Gaussian Q-function table). Here we see the Gaussian is faithfully represented and compact. In fact, the width computed is the same as that of the trinomial triangle in all examples shown. For values of σ very near the lower end of each range, the kernels are not as compact as they could be. If this is a concern, the formula could be replaced by reducing the multiplicative factor, for example, $\tilde{w} = \text{Round}(2.2\sigma - 0.5)$.

a	$w = 2a + 1$	binomial kernel	$\sigma^2 = \frac{a}{2}$	σ	$\alpha = \frac{w}{2\sigma}$	area
1	3	$\frac{1}{4}[\,1 \quad 2 \quad 1\,]$	0.5	0.71	2.12	96.60%
2	5	$\frac{1}{16}[\,1 \quad 4 \quad 6 \quad 4 \quad 1\,]$	1.0	1.0	2.50	98.76%
3	7	$\frac{1}{64}[\,1 \quad 6 \quad 15 \quad 20 \quad 15 \quad 6 \quad 1\,]$	1.5	1.22	2.86	99.58%
4	9	$\frac{1}{256}[\,1 \quad 8 \quad 28 \quad 56 \quad 70 \quad 56 \quad 28 \quad 8 \quad 1\,]$	2.0	1.41	3.18	99.85%

a	$w = 2a + 1$	trinomial kernel	$\sigma^2 = \frac{2a}{3}$	σ	$\alpha = \frac{w}{2\sigma}$	area
1	3	$\frac{1}{3}[\,1 \quad 1 \quad 1\,]$	0.67	0.82	1.84	93.42%
2	5	$\frac{1}{9}[\,1 \quad 2 \quad 3 \quad 2 \quad 1\,]$	1.33	1.15	2.17	97.00%
3	7	$\frac{1}{27}[\,1 \quad 3 \quad 6 \quad 7 \quad 6 \quad 3 \quad 1\,]$	2.00	1.41	2.47	98.65%
4	9	$\frac{1}{81}[\,1 \quad 4 \quad 10 \quad 16 \quad 19 \quad 16 \quad 10 \quad 4 \quad 1\,]$	2.67	1.63	2.76	99.42%

Table 3.2: The area under the curve is well captured by Gaussian kernels given by the binomial and trinomial triangles. However, as σ increases, the kernels waste computation because they are much wider than necessary to faithfully capture the Gaussian to a reasonable amount.

$\tilde{w}$	$w = 2\tilde{w} + 1$	σ range	$\alpha = w/2\sigma$ range	area max	area min
1	3	$[0.4, 0.8)$	$[3.75, 1.88)$	–	93.99%
2	5	$[0.8, 1.2)$	$[3.13, 2.08)$	99.83%	96.25%
3	7	$[1.2, 1.6)$	$[2.92, 2.19)$	99.65%	97.15%
4	9	$[1.6, 2.0)$	$[2.81, 2.25)$	99.50%	97.56%

Table 3.3: The area under the curve is well captured by Gaussian kernels given by GetKernelHalfWidth. For standard deviations very near the lower end of each range, the kernels could be reduced in size while maintaining acceptable accuracy, but for the most part the representation is compact.

3.5 Nonlinear filters

The median filter is nonlinear. It is ideal for salt-and-pepper noise.

3.6 Gaussian derivative kernels

A high-pass filter is one that accentuates the differences in the input signal. The simplest approach is to compute finite differences, e.g., by convolving with $[\,1 \quad -1\,]$. Since this kernel has an even number of elements, the center of the kernel can be placed on either element, leading to the so-called *forward* and *backward* approximations. In the real world, however, the input signal is a combination of the underlying signal in which we are interested and noise that has been unfortunately added to (or combined with) the signal. Therefore, it is usually wise to perform at least some slight smoothing to the image before differentiating, to help reduce the effects of such noise. Thus, we could convolve the image with a Gaussian smoothing filter, then convolve the result with a differentiating filter. Because of the associativity of convolution, this is the same as convolving with a central-difference operator:

$$(f(x) * [\,1 \quad 1\,]) * [\,1 \quad -1\,] = f(x) * ([\,1 \quad 1\,] * [\,1 \quad -1\,]) = f(x) * [\,1 \quad 0 \quad -1\,]. \tag{3.9}$$

With larger smoothing kernels, however, computing finite differences between neighboring smoothed pixels will not yield favorable results. Instead, a more general procedure can be obtained by noting that differentiation and convolution are associative, so that differentiating the smoothed signal is equivalent to convolving the image with a smoothed differentiating kernel:

$$\frac{\partial}{\partial x}(f(x) * g(x)) = f(x) * \left(\frac{\partial}{\partial x} g(x)\right), \tag{3.10}$$

where f is the input and g is the smoothing kernel. The implication of this formula is that we can create a smoothed discrete differentiating kernel by sampling the continuous derivative of the smoothing function. Since the Gaussian is the most common smoothing kernel, we focus our attention on sampling the derivative of a Gaussian, which is easily shown in the continuous domain to be just a scaled version of the x coordinate times the Gaussian:

$$\frac{dG(x)}{dx} = -\frac{x}{\sigma^2\sqrt{2\pi\sigma^2}} \exp\left(\frac{-x^2}{2\sigma^2}\right) = -\frac{x}{\sigma^2} G(x). \tag{3.11}$$

To construct the 1D Gaussian derivative kernel, then, we follow a procedure similar to the one used to construct the 1D Gaussian kernel.

CreateGaussianDerivativeKernel(σ)
1 $\tilde{w} \leftarrow$ GetKernelHalfWidth(σ)
2 $w \leftarrow 2\tilde{w} + 1$
3 $sum \leftarrow 0$
4 **for** $i \leftarrow 0$ **to** $w - 1$ **do**
5 $\quad G'[i] \leftarrow -(i - \tilde{w}) * \exp(-(i - \tilde{w}) * (i - \tilde{w})/(2 * \sigma * \sigma))$
6 $\quad sum \leftarrow sum + i * G'[i]$
7 **for** $i \leftarrow 0$ **to** $w - 1$ **do**
8 $\quad G'[i] \leftarrow G'[i]/\, sum$
9 **return** G'

Notice that normalization for differentiating kernels is different from normalization for Gaussian kernels. Since the former satisfy the property that the sum (zeroth moment) of their elements is zero, we cannot normalize by dividing by this sum as we did with the latter. Recall that Gaussian normalization was determined by imposing the constraint that convolution with a constant signal should not change the signal. In constrast, Gaussian derivative normalization is determined by imposing the constraint that convolution with a ramp should yield the slope (i.e., derivative) of the ramp. It is easy to verify that this goal is satisfied by dividing by the absolute value of the first moment: $\left|\sum_{i=0}^{w-1} iG'[i]\right|$, where $G'[i]$ is the ith element of the kernel.

Applying this procedure to the central difference operator leads to

$$G'_{0.5} = \frac{1}{2}\begin{bmatrix} 1 & 0 & -1 \end{bmatrix}^T, \tag{3.12}$$

since the first moment of $\begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$ is -2. Here we use G'_{σ^2} to denote the Gaussian derivative kernel with variance σ^2. However, there is no known way to compute the variance of a derivative kernel as there is for a smoothing kernel. And, in fact, no matter what variance is chosen, as long as $w = 3$ the resulting Gaussian derivative kernel will be identical because $G'_{0.5}$ is the *only* 3×1 Gaussian derivative kernel. For convenience, we have set the nominal variance here to $\sigma^2 = 0.5$. Also, keep in mind that although the normalization constant is needed in order to compute the slope of the function, in practice it can usually be ignored because the goal is often simply to compute the relative gradient at each pixel.

At first glance, it seems odd that the differentiating kernel ignores the central pixel. However, this is a natural consequence of the fact that the derivative of a Gaussian is zero at $x = 0$. Another way to see this is that, because the kernel sums to zero and is antisymmetric, its center pixel has to be zero. A straightforward interpretation of the centralized difference operator is that it is averaging the two slopes computed by the forward and backward differences. That is, if $h(x) = f(x) * G'_{0.5}$, then

$$h(x) = \frac{1}{2}\left(\frac{f(x) - f(x-1)}{1} + \frac{f(x+1) - f(x)}{1}\right) = \frac{f(x+1) - f(x-1)}{2}.$$

3.7 Computing the image gradient

The derivative of a function of one variable is defined as

$$\frac{df}{dx} = \lim_{m \to 0} \frac{f(x+m) - f(x)}{m}. \tag{3.13}$$

Because the image is defined over a two-dimensional domain, we must generalize the notion of derivative to 2D. This leads to the concept of the **gradient**, which is a vector whose elements are the partial derivatives of the function along the two axes:

$$\nabla f(x,y) = \left[\begin{array}{cc} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{array} \right]^T, \tag{3.14}$$

where the superscript T denotes transpose. If the image is viewed as a surface $z = f(x,y)$, where z is the height of the surface at any point, the gradient is a vector pointing uphill. If we let $\mathbf{d}_{(x_0,y_0)} = \nabla f(x,y)|_{(x_0,y_0)}$ be the gradient evaluated at a point (x_0,y_0) and $\mathbf{e}_{(x_0,y_0)} = \left[\begin{array}{cc} x - x_0 & y - y_0 \end{array} \right]^T$ be the vector from (x_0,y_0) to an arbitrary point (x,y), then the equation of the tangent plane to the surface at the point (x_0,y_0) is given by the inner product of the two vectors plus the value at that point: $\hat{z} = \mathbf{d}_{(x_0,y_0)}^T \mathbf{e}_{(x_0,y_0)} + f(x_0,y_0)$.

Thankfully, the partial derivative of a 2D Gaussian is separable:

$$\begin{aligned}
f(x,y) * \frac{\partial G(x,y)}{\partial x} &= \int\int -\frac{\xi}{\sigma^2} f(x-\xi, y-\eta) \exp\left\{ \frac{-(\xi^2+\eta^2)}{2\sigma^2} \right\} d\xi d\eta & (3.15)\\
&= \int\int -\frac{\xi}{\sigma^2} f(x-\xi, y-\eta) \exp\left\{ \frac{-\eta^2}{2\sigma^2} \right\} \exp\left\{ \frac{-\xi^2}{2\sigma^2} \right\} d\xi d\eta & (3.16)\\
&= \int \left[\int -\frac{\xi}{\sigma^2} f(x-\xi, y-\eta) \exp\left\{ \frac{-\eta^2}{2\sigma^2} \right\} d\eta \right] \exp\left\{ \frac{-\xi^2}{2\sigma^2} \right\} d\xi & (3.17)\\
&= (f(x,y) * G'(y)) * G^T(x), & (3.18)
\end{aligned}$$

where $G'(x) = \partial G / \partial x$. Similar analysis shows that the partial derivative in y is also separable:

$$f(x,y) * \frac{\partial G(x,y)}{\partial y} = (f(x,y) * G(y)) * G'^T(x). \tag{3.19}$$

Therefore, to compute the gradient of an image, we differentiate along the x and y axes by convolving with a smoothing 1D kernel and a differentiating kernel in the orthogonal direction. To compute the partial derivative with respect to x, we convolve with a horizontal derivative of a Gaussian, followed by a vertical Gaussian (or switch the order of these two, since convolution is commutative). To compute the partial derivative with respect to y, we convolve with a horizontal Gaussian, followed by a vertical derivative of a Gaussian.

The simplest 2D differentiating kernel is the Prewitt operator, which is obtained by convolving a 1D Gaussian derivative kernel with a 1D box filter in the orthogonal direction:

$$\begin{aligned}
Prewitt_x &= \frac{1}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} * \frac{1}{2} \left[\begin{array}{ccc} 1 & 0 & -1 \end{array} \right] = \frac{1}{6} \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}\\
Prewitt_y &= \frac{1}{3} \left[\begin{array}{ccc} 1 & 1 & 1 \end{array} \right] * \frac{1}{2} \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}.
\end{aligned}$$

It can be shown that applying the Prewitt operator is equivalent to solving for the plane that minimizes the least-squares-error over the 3×3 window, where all the pixels are treated equally. With this

equivalence, the Prewitt operator can be extended to other dimensions like 4×4, but they are not widely used.

The Sobel operator is more robust, as it uses the Gaussian ($\sigma^2 = 0.5$) for the smoothing kernel:

$$\begin{aligned} Sobel_x &= G_{0.5} * (G'_{0.5})^T = \frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix} * \frac{1}{2}\begin{bmatrix}1 & 0 & -1\end{bmatrix} = \frac{1}{8}\begin{bmatrix}1 & 0 & -1\\2 & 0 & -2\\1 & 0 & -1\end{bmatrix} \\ Sobel_y &= (G_{0.5})^T * G'_{0.5} = \frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix} * \frac{1}{2}\begin{bmatrix}1\\0\\-1\end{bmatrix} = \frac{1}{8}\begin{bmatrix}1 & 2 & 1\\0 & 0 & 0\\-1 & -2 & -1\end{bmatrix}. \end{aligned}$$

The Scharr operator is similar to the Sobel, but with a smaller variance ($\sigma^2 = 0.375$) in the smoothing kernel:

$$\begin{aligned} Scharr_x &= G_{0.375} * (G'_{0.5})^T = \frac{1}{16}\begin{bmatrix}3\\10\\3\end{bmatrix} * \frac{1}{2}\begin{bmatrix}1 & 0 & -1\end{bmatrix} = \frac{1}{32}\begin{bmatrix}3 & 0 & -3\\10 & 0 & -10\\3 & 0 & -3\end{bmatrix} \\ Scharr_y &= G_{0.375} * (G'_{0.5})^T = \frac{1}{16}\begin{bmatrix}3 & 10 & 3\end{bmatrix} * \frac{1}{2}\begin{bmatrix}1\\0\\-1\end{bmatrix} = \frac{1}{32}\begin{bmatrix}3 & 10 & 3\\0 & 0 & 0\\-3 & -10 & -3\end{bmatrix}. \end{aligned}$$

The advantage of the Scharr operator is that it is more rotationally invariant than other 3×3 Gaussian kernels.

For completeness, we mention the classic Roberts cross operator, which is the oldest pair of gradient kernels:

$$\begin{aligned} Roberts_1 &= \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix} \\ Roberts_2 &= \frac{1}{\sqrt{2}}\begin{bmatrix}0 & -1\\1 & 0\end{bmatrix}. \end{aligned}$$

The Roberts kernels suffer from two drawbacks. First, they compute derivatives along the diagonal directions rather than along the x and y axes, which is more of an inconvenience than any fundamental limitation. Secondly, because they have even dimensions, the kernels are not centered. That is, they do not compute the derivative at a pixel, but rather between pixels. However, for some applications two-point differences are better than three-point central differences, because they use all pixels in the computation (as opposed to ignoring the central pixel), and because they are more compact [9].

Note that Sobel is equivalently the convolution of rotated Roberts kernels with a 2×2 box filter:

$$\begin{aligned} Sobel_x &= \frac{1}{4}\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix} * \frac{1}{2}\begin{bmatrix}1 & -1\\1 & -1\end{bmatrix} = \frac{1}{8}\begin{bmatrix}1 & 0 & -1\\2 & 0 & -2\\1 & 0 & -1\end{bmatrix} \\ Sobel_y &= \frac{1}{4}\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix} * \frac{1}{2}\begin{bmatrix}1 & 1\\-1 & -1\end{bmatrix} = \frac{1}{8}\begin{bmatrix}1 & 2 & 1\\0 & 0 & 0\\-1 & -2 & -1\end{bmatrix}. \end{aligned}$$

Once we have computed the gradient of the image, it is often desirable to compute the magnitude of the gradient. A natural way to do this would be to compute the Euclidean norm of the gradient vector: $|\nabla f| = \sqrt{f_x^2 + f_y^2}$, where f_x and f_y are the two components of the gradient vector, i.e., $\nabla f = \begin{bmatrix} f_x & f_y \end{bmatrix}^T$. A computationally efficient approximation can be obtained by computing the sum of the absolute values: $|\nabla f| \approx |f_x| + |f_y|$. A third option is to select the maximum of the two absolute values: $|\nabla f| \approx \max(|f_x|, |f_y|)$. Note that these three choices are, respectively, the Euclidean, Manhattan, and

Figure 3.2: Intensity edges capture a rich representation of the scene. The scenes and objects in these line drawings are, with little difficulty, recognizable by the average human viewer. From [56]. For the original images, turn to Figure 3.3.

chessboard distances between the origin and the point (f_x, f_y), as discussed in Section 2.6.1. One might naturally assume that the Euclidean norm always yields the "most correct" answer, but in fact it suffers significantly from discretization effects. Consider, for example, a vertical (or horizontal) step edge between two intensities with values v_1 and v_2. Using a central difference operator $[\,1 \quad 0 \quad -1\,]^T$, the Euclidean norm yields a gradient magnitude of $|v_1 - v_2|$. But if we rotate the step edge by 45 degrees to create a diagonal step edge, the same operator will yield $\sqrt{2}|v_1 - v_2|$, which is a factor of $\sqrt{2}$ more. The Manhattan norm is even worse, yielding a factor of 2 between the diagonal magnitude and the vertical/horizontal magnitude. In contrast, with the chessboard norm the factor is 1 and the problem disappears. The chessboard norm, therefore, is to be preferred not only because it is more computationally efficient and its output fits within the 8 bits per pixel, but also because it is much more rotationally invariant than the other two. For these reasons, we will define the gradient magnitude as

$$|\nabla f| \equiv \max(|f_x|, |f_y|). \tag{3.20}$$

We are now ready to provide pseudocode for computing the image gradient. For each pixel in the image, the partial derivatives in x and y are computed. These values are then converted into a magnitude and phase representation.

```
ComputeImageGradient(I, σ)
1  G = CreateGaussianKernel(σ)
2  G' = CreateGaussianDerivativeKernel(σ)
3  G_x = ConvolveSeparable(I, G, G')
4  G_y = ConvolveSeparable(I, G', G)
5  for (x, y) ∈ I do
6        G_mag = max(|G_x(x, y)|, |G_y(x, y)|)
7        G_phase = atan2(G_y(x, y), G_x(x, y))
8  return G_mag, G_phase
```

3.8 Edge detection

Intensity edges are locations in the image where the intensity function changes rapidly. From an information-theoretic point of view, these are the locations that carry the most information because the graylevels at these pixels are the least predictable from the values of their neighbors [1]. Intensity edges retain a surprisingly large amount of information from the scene, as seen by the line drawings in Figure 3.2. Viewing only the line drawings, most human viewers can recognize these scenes effortlessly. Such demonstrations suggest that intensity edges are important for the human visual system and, therefore, that they should be important for artificial systems as well. In fact, some of the earliest work in computer vision focused on line drawing images of polyhedral objects (Roberts, Huffman line labeling).

There are four types of intensity edges that are generally considered important: Step edges, line edges, roof edges, and ridge edges. We will devote our attention primarily to step edges, which is the

Figure 3.3: The original images from which the line drawings shown in Figure 3.2 were obtained. From [56].

simplest and most widely used. Among the four types is also the one most closely tied to the derivative of the image.

The simplest way to find intensity edges is to compute the gradient magnitude and threshold. In fact, in some applications no thresholding is needed because a binary decision is not needed, and may even be undesirable.

Another way to compute edges is the approach of Frei-Chen, which uses a set of nine kernels. It considers a 3×3 subimage to be a vector in $\mathbb{R}^9$. An orthogonal basis for the 9-dimensional space of 3×3 subimages is shown below.

$$w_1 = \frac{1}{\sqrt{8}} \begin{bmatrix} -1 & -\sqrt{2} & -1 \\ 0 & 0 & 0 \\ 1 & \sqrt{2} & 1 \end{bmatrix} \qquad w_2 = \frac{1}{\sqrt{8}} \begin{bmatrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{bmatrix}$$

$$w_3 = \frac{1}{\sqrt{8}} \begin{bmatrix} 0 & -1 & \sqrt{2} \\ 1 & 0 & -1 \\ -\sqrt{2} & 1 & 0 \end{bmatrix} \qquad w_4 = \frac{1}{\sqrt{8}} \begin{bmatrix} \sqrt{2} & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & -\sqrt{2} \end{bmatrix}$$

$$w_5 = \frac{1}{2} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \qquad w_6 = \frac{1}{2} \begin{bmatrix} -1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \end{bmatrix}$$

$$w_7 = \frac{1}{6} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix} \qquad w_8 = \frac{1}{6} \begin{bmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{bmatrix}$$

$$w_9 = \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

(The normalization factors for w_5 through w_9 come from Shapiro and Stockman, p. 164.)

For $i = 1, \ldots, 9$, the vector $\mathbf{w}_i$ is obtained by stacking the elements of w_i into a column. The first four vectors $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3, \mathbf{w}_4$ form the basis for the edge subspace, while the next four $\mathbf{w}_5, \mathbf{w}_6, \mathbf{w}_7, \mathbf{w}_8$ form the basis for the line subspace. Edges are detected by projecting the 9-element vector $\mathbf{b}$, reshaped from a 3×3 subimage, onto the edge subspace:

$$\theta_{edge} = \arccos \sqrt{\frac{\sum_{i=1}^{4} (\mathbf{w}_i^T \mathbf{b})^2}{\sum_{i=1}^{9} (\mathbf{w}_i^T \mathbf{b})^2}}$$

The smaller value of the angle θ_{edge} between the two vectors, the more likely the pixel is an edge point. Equivalent results are obtained by

$$\frac{\sum_{i=1}^{4} (\mathbf{w}_i^T \mathbf{b})^2}{\mathbf{b}^T \mathbf{b}},$$

in which case a large value indicates an edge point. Frei-Chen can also be used to detect lines, by projecting the subimage vector onto the basis functions $\mathbf{w}_i, i = 5, \ldots, 8$ instead.

3.9 Canny edge detector

The Canny edge detector is a classic algorithm for detecting intensity edges in a grayscale image. The algorithm involves three steps. First the gradient of the image is computed. Next, any pixel whose gradient magnitude value is not a local maximum in the direction of the gradient direction (or phase) is set to zero. Finally, double-thresholding (or edge linking) is performed to discard pixels without much support. The result is a binary image B_{edges} that has a 1 for every edge pixel and a 0 everywhere else.

CANNY$(I, \sigma, \tau_{low}, \tau_{high})$
1 $G_{mag}, G_{phase} =$ COMPUTEIMAGEGRADIENT(I, σ)
2 $G_{localmax} =$ NONMAXSUPPRESSION(G_{mag}, G_{phase})
3 $B_{edges} =$ EDGELINKING$(G_{localmax}, \tau_{low}, \tau_{high})$
4 **return** B_{edges}

We have already described the code for computing the image gradient. For non-maximum suppression, we compare the gradient magnitude of each pixel with the two pixels along the gradient direction. If the pixel is not a local maximum, then its gradient magnitude value is set to zero.

NONMAXSUPPRESSION(G_{mag}, G_{phase})
1 $G_{localmax} \leftarrow G_{mag}$
2 **for** $(x, y) \in G_{mag}$ **do**
3 $\quad v \leftarrow G_{mag}(x, y)$
4 $\quad \theta \leftarrow G_{phase}(x, y)$
5 $\quad$ **if** $-\frac{\pi}{8} \leq \theta < \frac{\pi}{8}$ AND $(v < G_{mag}(x-1, y)$ OR $v < G_{mag}(x+1, y))$ **then**
6 $\quad\quad G_{localmax}(x, y) \leftarrow 0$
7 $\quad$ **elseif** $\frac{\pi}{8} \leq \theta < \frac{3\pi}{8}$ AND $(v < G_{mag}(x-1, y-1)$ OR $v < G_{mag}(x+1, y+1))$ **then**
8 $\quad\quad G_{localmax}(x, y) \leftarrow 0$
9 $\quad$ **elseif** $\frac{3\pi}{8} \leq \theta < \frac{5\pi}{8}$ AND $(v < G_{mag}(x, y-1)$ OR $v < G_{mag}(x, y+1))$ **then**
10 $\quad\quad G_{localmax}(x, y) \leftarrow 0$
11 $\quad$ **elseif** $\frac{5\pi}{8} \leq \theta < \frac{7\pi}{8}$ AND $(v < G_{mag}(x-1, y+1)$ OR $v < G_{mag}(x+1, y-1))$ **then**
12 $\quad\quad G_{localmax}(x, y) \leftarrow 0$
13 $\quad$ **return** $G_{localmax}$

Edge linking is basically double thresholding, which follows floodfill. The thresholds can be set manually or automatically. One way to set them automatically is, for example, to create a histogram of the gradients of the image, and to set τ_{high} to the value that forces $\alpha\%$ of the pixels to be edge pixels, and $\tau_{high} = \beta\tau_{low}$. Some reasonable values are $\alpha = 10$ and $\beta = 0.2$.

EDGELINKING$(G_{localmax}, \tau_{low}, \tau_{high})$
1 **for** $(x, y) \in G_{localmax}$ **do**
2 $\quad$ **if** $G_{localmax} \geq \tau_{high}$ **then**
3 $\quad\quad$ *frontier.push*(x, y)
4 $\quad\quad B_{edges}(q) \leftarrow$ ON
5 **while** NOT *frontier.isEmpty*() **do**
6 $\quad p \leftarrow$ *frontier.pop*()
7 $\quad$ **for** $q \in \mathcal{N}(p)$ **do**
8 $\quad\quad$ **if** $G_{localmax}(q) \geq \tau_{low}$
9 $\quad\quad$ **then** *frontier.push*(q)
10 $\quad\quad\quad B_{edges}(q) \leftarrow$ ON
11 **return** B_{edges}

One question that arises from the above discussion on computing the image gradient, is what sigma to use? A large sigma yields better signal-to-noise ratio (SNR), but a smaller sigma yields more accurate location. This is the well-known *localization-detection tradeoff*. Canny's landmark paper analyzed this tradeoff, deriving an optimal step detector that satisfies two criteria: 1) good detection, i.e., low false positive and false negative rates; and 2) good localization, i.e., the detected edge should be close to the true edge. To quantify these, let the true edge be given by an ideal step:

$$G(x) = \begin{cases} 0 & \text{if } x < 0 \\ A & \text{if } x \geq 0 \end{cases}$$

Let the actual edge in the image be the true edge plus noise: $G(x) + \xi(x)$, where $\xi(x) \sim \mathcal{N}(0, n_0^2)$. Let $f(x)$ be the impulse response of the filter we are trying to find. The good detection criterion is the signal-to-noise ratio (SNR), i.e., the ratio of the response of the filter to the true edge and the root-mean-square response of the filter to the noise:

$$SNR = \frac{A \left| \int_{-W}^{0} f(x)dx \right|}{n_0 \sqrt{\int_{-W}^{W} f^2(x)dx}} = \frac{A}{n_0} \Sigma(f)$$

where the filter has finite impulse response bounded by $[-W, W]$.

For the localization criterion, we use the reciprocal of the RMS distance of the marked edge from the center of the true edge. Skipping the mathematical derivation, which is too involved to cite here, we get

$$LOC = \frac{1}{\sqrt{E[x_0^2]}} = \frac{A\,|f'(0)|}{n_0 \sqrt{\int_{-W}^{W} f'^2(x)dx}} = \frac{A}{n_0} \Lambda(f')$$

where x_0 is the distance of the detected edge from the true edge, and $f' = \partial f / \partial x$ is the derivative of f. Notice that $\Sigma(f)$ and $\Lambda(f')$ are two measures of the performance of the filter, and that they depend only on the filter, not on the noise n_0 or the magnitude A of the true edge.

By substituting into the equations above it can be shown that scaling affects the performance in the following way:

$$\Sigma\left(f\left(\frac{x}{w}\right)\right) = \sqrt{w}\Sigma(f) \qquad \Lambda\left(f'\left(\frac{x}{w}\right)\right) = \frac{1}{\sqrt{w}}\Lambda(f')$$

In other words, if the filter is stretched to make it larger ($w > 1$), then the detection response increases. This makes sense, because a broader impulse response will have a larger SNR. At the same time, a larger filter reduces the localization performance. This is the *localization-detection tradeoff*. It makes sense to multiply the two criteria together to achieve combined performance that is independent of scale: $\Sigma(f)\Lambda(f')$. It can be shown that the filter f that maximizes this is

$$G(x) = \begin{cases} -1 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

In other words, the optimal 1D step edge detector is a simple difference operator, or box filter. The problem with a box filter is that it causes many local maxima to be detected. Therefore, we introduce the single response constraint, which says that the detector should return only one point for each true edge point. I.e., it must minimize the number of local maxima around the true edge created by noise. Solving this numerical optimization problem, we arrive at a signal whose shape is nearly the same as the derivative of a Gaussian. Because the Derivative of Gaussian is separable, and because the Gaussian has other nice properties, it is usually (always) used instead. Extended to 2D, we use the gradient computed from the partial derivatives of a 2D Gaussian, and steer the result to the appropriate 1D direction across the edge. This is the theory behind our use of the derivative of a Gaussian above.

3.10 Laplacian of Gaussian

Just as the finite difference operator approximates the first derivative, the difference between differences approximates the second derivative. This can be seen in 1D by convolving the function with the non-centralized difference operator, then convolving the result again with the same operator:

$$(f(x)*[\,1 \quad -1\,])*[\,1 \quad -1\,] = f(x)*([\,1 \quad -1\,]*[\,1 \quad -1\,]) = f(x)*[\,1 \quad -2 \quad 1\,], \tag{3.21}$$

which yields the second-derivative convolution kernel $[\,1 \quad -2 \quad 1\,]$. It turns out that, just as there is only one 3×1 kernel for computing the derivative of a Gaussian, this is the only 3×1 kernel for computing the second derivative of a Gaussian. It is also not hard to see that the normalization factor is 1 and is therefore already included in the kernel. We can express the first and second derivatives as $dI/dx \approx I(x) - I(x-1)$ and $d^2I/dx^2 \approx (I(x+1) - I(x)) - (I(x) - I(x-1)) = I(x+1) - 2I(x) + I(x-1)$, respectively.

The Laplacian operator ∇^2 is defined as the divergence of the gradient of a function. In 2D Cartesian coordinates, this leads to a natural extension of the second derivative to two dimensions:

$$\nabla^2 I = \nabla \cdot \nabla I = \left[\begin{array}{cc} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} \end{array}\right] \left[\begin{array}{cc} \frac{\partial I}{\partial x} & \frac{\partial I}{\partial y} \end{array}\right]^T = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}. \tag{3.22}$$

So the Laplacian of an image is the sum of the second derivative along the two axes.

To reduce the effects of noise, we generally smooth the image with a Gaussian, then compute the Laplacian. Because of associativity, this is the same as convolving the image with a **Laplacian of Gaussian (LoG)**:

$$\frac{\partial^2 (I*G)}{\partial x^2} + \frac{\partial^2 (I*G)}{\partial y^2} = I* \left(\frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2} \right). \tag{3.23}$$

The Laplacian of Gaussian function is also known as the inverted "Mexican hat" operator, because of its shape. The LoG is rotationally symmetric and is a center-surround filter because it consists of a central core of negative values surrounded by an annular ring of positive values. It therefore maintains some biological plausibility when compared with certain cells in the retina which perform center-surround operations. The LoG is a bandpass filter.

In the discrete domain, we sample the second derivative of a 2D Gaussian along the x-axis, and sample the second derivative of a 2D Gaussian along the y-axis. The sum of the two results is a LoG kernel. To create a 1D Gaussian second derivative, sample a 1D continuous Gaussian second-derivative function then divide by one-half the second centralized moment of the elements: $\frac{1}{2}\sum_{i=-h}^{h} i^2 k_i$, where h is the halfwidth of the kernel. The justification for this is that convolution with a parabola $y = x^2$ should yield the second derivative, which is two. Note that with the second derivative, it is important to use the centralized moment, whereas with the zeroth and first derivatives, either centralized or non-centralized produces the same result.

The 2D Gaussian second-derivative filter is separable:

$$\begin{aligned}
I(x,y)*\frac{\partial^2 G(x,y)}{\partial x^2} &= \int\int \frac{1}{\sigma^2}\left(1 - \frac{\xi^2}{\sigma^2}\right) I(x-\xi, y-\eta) \exp\left\{\frac{-(\xi^2+\eta^2)}{2\sigma^2}\right\} d\xi d\eta & (3.24)\\
&= \int\int \frac{1}{\sigma^2}\left(1 - \frac{\xi^2}{\sigma^2}\right) I(x-\xi, y-\eta) \exp\left\{\frac{-\xi^2}{2\sigma^2}\right\} \exp\left\{\frac{-\eta^2}{2\sigma^2}\right\} d\xi d\eta & (3.25)\\
&= \int \left[\int \frac{1}{\sigma^2}\left(1 - \frac{\xi^2}{\sigma^2}\right) I(x-\xi, y-\eta) \exp\left\{\frac{-\xi^2}{2\sigma^2}\right\} d\xi\right] \exp\left\{\frac{-\eta^2}{2\sigma^2}\right\} d\eta & (3.26)\\
&= [I(x)*G''(x)]*G(y), & (3.27)
\end{aligned}$$

where $G''(x) = \partial^2 G/\partial x^2$. Similarly for y. Therefore,

$$\frac{\partial^2 (I*G)}{\partial x^2} + \frac{\partial^2 (I*G)}{\partial y^2} = I* \left(\frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2} \right) = [I(x)*G''(x)]*G(y) + [I(x)*G''(y)]*G(x) \tag{3.28}$$

So the LoG itself is not separable, but it is the sum of two separable kernels. As a result, LoG is no more expensive than the first derivative.

A LoG kernel can be obtained by computing $\frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2}$ with the appropriate value for σ. For 3×3 kernels, the differentiating kernel is fixed as $[\,1 \quad -2 \quad 1\,]$ (because it is the only 3×1 second-derivative Gaussian kernel), and the smoothing kernel is determined by σ. For $\sigma^2 = 0.25$, for example, we have $\frac{1}{8}[\,1 \quad 6 \quad 1\,]$ as the smoothing kernel, leading to

$$\begin{aligned}
\frac{\partial^2 G_{0.25}}{\partial x^2} &= [\,1 \quad -2 \quad 1\,] * \frac{1}{8}\begin{bmatrix}1\\6\\1\end{bmatrix} = \frac{1}{8}\begin{bmatrix}1 & -2 & 1\\6 & -12 & 6\\1 & -2 & 1\end{bmatrix}\\
\frac{\partial^2 G_{0.25}}{\partial y^2} &= \frac{1}{8}[\,1 \quad 6 \quad 1\,] * \begin{bmatrix}1\\6\\1\end{bmatrix} = \frac{1}{8}\begin{bmatrix}1 & 6 & 1\\-2 & -12 & -2\\1 & 6 & 1\end{bmatrix}\\
LoG_{0.25} &= \frac{\partial^2 G_{0.25}}{\partial x^2} + \frac{\partial^2 G_{0.25}}{\partial y^2} = \frac{1}{4}\begin{bmatrix}1 & 2 & 1\\2 & -12 & 2\\1 & 2 & 1\end{bmatrix},
\end{aligned}$$

which is the kernel used by Horn-Schunck. Repeating this procedure for other values of σ^2 yields alternative 3×3 LoG kernels:

$\sigma^2 = 0.0$	$\sigma^2 = 0.167$	$\sigma^2 = 0.20$	$\sigma^2 = 0.25$	$\sigma^2 = 0.33$	$\sigma^2 = 0.5$
$[\,0 \quad 1 \quad 0\,]$	$\frac{1}{12}[\,1 \quad 10 \quad 1\,]$	$\frac{1}{10}[\,1 \quad 8 \quad 1\,]$	$\frac{1}{8}[\,1 \quad 6 \quad 1\,]$	$\frac{1}{6}[\,1 \quad 4 \quad 1\,]$	$\frac{1}{4}[\,1 \quad 2 \quad 1\,]$
$\begin{bmatrix}0 & 1 & 0\\1 & -4 & 1\\0 & 1 & 0\end{bmatrix}$	$\frac{1}{6}\begin{bmatrix}1 & 4 & 1\\4 & -20 & 4\\1 & 4 & 1\end{bmatrix}$	$\frac{1}{5}\begin{bmatrix}1 & 3 & 1\\3 & -16 & 3\\1 & 3 & 1\end{bmatrix}$	$\frac{1}{4}\begin{bmatrix}1 & 2 & 1\\2 & -12 & 2\\1 & 2 & 1\end{bmatrix}$	$\frac{1}{3}\begin{bmatrix}1 & 1 & 1\\1 & -8 & 1\\1 & 1 & 1\end{bmatrix}$	$\frac{1}{2}\begin{bmatrix}1 & 0 & 1\\0 & -4 & 0\\1 & 0 & 1\end{bmatrix}$

where the second row in the table shows the 1D smoothing kernel, while the third row shows the resulting LoG kernel. Of course, all the LoG kernels are sums of separable kernels, so that computing them is not any less efficient than computing the gradient. The center-surround property is revealed by seeing that in all cases the kernel is equivalent to a scalar times the difference between the average intensity of the neighbors and the intensity of the central pixel: $\nabla^2 I = h(\bar{I} - I)$, where $\bar{I}$ is the average intensity of the neighbors. The scalar h is the negative of the central kernel weight, so for $\sigma^2 = 0.25$, for example, $h = 3$, and so on. Note that the simplest Laplacian kernel, the one involving no smoothing, is simply the sum of the horizontal and vertical second-derivative kernels (considered as the middle row or column, respectively, of a square matrix whose remaining elements are zero):

$$[\,1 \quad -2 \quad 1\,] + \begin{bmatrix}1\\-2\\1\end{bmatrix} = \begin{bmatrix}0 & 1 & 0\\1 & -4 & 1\\0 & 1 & 0\end{bmatrix}.$$

Since the first derivative maxima are the second derivative zero crossings, the zero crossings of the LoG output can be used to detect edges. This is the well-known Marr-Hildreth operator. However, it is not as good as Canny at edge detection because it is isotropic, smoothing across as well as along edges. It is, however, biologically plausible and a useful precomputation, and the sign of the LoG (sLoG) has been shown to be a useful data reduction technique, which we shall revisit later.

3.11 Difference of Gaussians

Recall that a 1D Gaussian is given by

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-x^2}{2\sigma^2}\right), \tag{3.29}$$

where the normalizing factor ensures that $\int_{-\infty}^{\infty} G(x) = 1$. It is easy to show that the first derivative is

$$\begin{aligned} \frac{dG}{dx} &= \frac{d}{dx}\left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-x^2}{2\sigma^2}\right)\right] &(3.30)\\ &= -\frac{x}{\sigma^2} G(x). &(3.31)\end{aligned}$$

Therefore, the second derivative is

$$\begin{aligned} \frac{d^2G(x)}{dx^2} &= \frac{d}{dx}\left[\frac{dG(x)}{dx}\right] \\ &= \frac{d}{dx}\left[-\frac{x}{\sigma^2}G(x)\right] \\ &= -\frac{1}{\sigma^2}\left[G(x) + x\frac{dG(x)}{dx}\right] \\ &= -\left[\frac{1}{\sigma^2} - \frac{x^2}{\sigma^4}\right] G(x). \end{aligned}$$

Now let ua differentiate $G(x)$ with respect to the scale parameter σ:

$$\begin{aligned} \frac{dG(x)}{d\sigma} &= \frac{d}{d\sigma}\left[\frac{1}{\sqrt{2\pi}}\sigma^{-1}e^{-\frac{1}{2}x^2\sigma^{-2}}\right] \\ &= \frac{1}{\sqrt{2\pi}}\left[-\sigma^{-2} + \sigma^{-1}(x^2\sigma^{-3})\right] e^{-\frac{x^2}{2\sigma^2}} \\ &= \frac{1}{\sqrt{2\pi}}\left[-\frac{1}{\sigma^2} + \frac{x^2}{\sigma^4}\right] e^{-\frac{1}{2}x^2\sigma^{-2}} \\ &= -\sigma\left[\frac{1}{\sigma^2} - \frac{x^2}{\sigma^4}\right] G(x) \end{aligned}$$

Dividing the two ezpressions yields a surprisingly simple result:

$$\frac{dG(x)}{d\sigma} = \sigma\frac{d^2G(x)}{dx^2}.$$

From the definition of derivative, the Taylor series expansion to the first order of a function is

$$f(x+a) \approx f(x) + a\left.\frac{df}{dx}\right|_x.$$

Applied to our problem we get

$$G(x;\sigma+\Delta_\sigma) \approx G(x;\sigma) + \Delta_\sigma\left.\frac{dG}{d\sigma}\right|_\sigma.$$

In other words, a discrete difference of Gaussians (DoG) is

$$G(x;\sigma+\Delta_\sigma) - G(x;\sigma) \approx \Delta_\sigma\sigma\frac{d^2G}{dx^2}.$$

If we let $\sigma_1 = \sigma$ and $\sigma_2 = \sigma + \Delta_\sigma$, and if we set $\rho = \sigma_2/\sigma_1$, then we have $\rho = 1 + \frac{\Delta_\sigma}{\sigma}$, or $\Delta_\sigma = \sigma(\rho - 1)$. Substituting yields

$$G(x; \rho\sigma) - G(x; \sigma) \quad \approx \quad (\rho - 1)\sigma^2 \frac{d^2 G}{dx^2}.$$

or

$$DoG(x; \sigma) \approx (\rho - 1)\sigma^2 LoG(x; \sigma).$$

If $\rho = 1.6$, this leads to

$$DoG(x; \sigma) \approx 0.6\sigma^2 LoG(x; \sigma)$$

So the difference between two Gaussians, with similar sigmas, is approximately the same as the LoG except for a constant scale factor. Thus the DoG operator is invariant across scales, while the LoG operator has to be normalized by σ^2. DoG / LoG $\approx$ constant. The LoG can be well approximated, up to the scale factor, with a difference of two centered Gaussians with scales related by $\sigma_2/\sigma_1 = 1.6$.

Now let us try the 2D case. The multidimensional Gaussian in n dimensions is

$$G(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu)\right)$$

so in 2D, a zero-mean isotropic Gaussian with $\Sigma = diag(\sigma^2, \sigma^2)$ is

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right).$$

Therefore,

$$\begin{aligned}
\frac{\partial^2 G(x,y)}{\partial x^2} &= \frac{\partial^2}{\partial x^2}\left[\frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)\right] \\
&= \frac{\partial}{\partial x}\left[\frac{-x}{2\pi\sigma^4}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)\right] \\
&= \frac{-x}{2\pi\sigma^4}\frac{\partial}{\partial x}\left[\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)\right] + \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)\frac{\partial}{\partial x}\left[\frac{-x}{2\pi\sigma^4}\right] \\
&= \left(\frac{-x}{2\pi\sigma^4}\right)\left(\frac{-x}{\sigma^2}\right)\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) + \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)\left(\frac{-1}{2\pi\sigma^4}\right) \\
&= \left[\frac{x^2}{2\pi\sigma^6} - \frac{1}{2\pi\sigma^4}\right]\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \\
&= \frac{1}{2\pi\sigma^4}\left[1 - \frac{x^2}{\sigma^2}\right]\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \\
&= \frac{1}{\sigma^2}\left[1 - \frac{x^2}{\sigma^2}\right]G(x, y).
\end{aligned}$$

By symmetry,

$$\frac{\partial^2 G(x,y)}{\partial y^2} \quad = \quad \frac{1}{\sigma^2}\left[1 - \frac{y^2}{\sigma^2}\right]G(x, y).$$

Putting these together yields

$$LoG(x, y; \sigma) = \frac{\partial^2 G(x,y)}{\partial x^2} + \frac{\partial^2 G(x,y)}{\partial y^2} \quad = \quad \left[\frac{x^2 + y^2 - 2\sigma^2}{2\pi\sigma^6}\right]\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

$$
\begin{aligned}
&= -\frac{1}{\pi\sigma^4}\left[\frac{2\sigma^2 - x^2 - y^2}{2\sigma^2}\right]\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \\
&= -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right]\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \\
&= -\frac{2}{\sigma^2}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right]G(x,y) \\
&= \left[\frac{x^2+y^2}{\sigma^4} - \frac{2}{\sigma^2}\right]G(x,y).
\end{aligned}
$$

Now let us differentiate with respect to σ:

$$
\begin{aligned}
\frac{dG(x,y)}{d\sigma} &= \frac{d}{d\sigma}\left[\frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)\right] \\
&= \frac{1}{2\pi\sigma^2}\frac{d}{d\sigma}\left[\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)\right] + \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)\frac{d}{d\sigma}\left[\frac{1}{2\pi\sigma^2}\right] \\
&= \frac{1}{2\pi\sigma^2}\frac{d}{d\sigma}\left[\exp\left(-\frac{1}{2}(x^2+y^2)\sigma^{-2}\right)\right] + \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)\frac{d}{d\sigma}\left[\frac{\sigma^{-2}}{2\pi}\right] \\
&= \frac{(x^2+y^2)\sigma^{-3}}{2\pi\sigma^2}\exp\left(-\frac{1}{2}(x^2+y^2)\sigma^{-2}\right) + \frac{-2\sigma^{-3}}{2\pi}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \\
&= \left[\frac{(x^2+y^2)}{2\pi\sigma^5} - \frac{1}{\pi\sigma^3}\right]\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \\
&= -\frac{1}{\pi\sigma^3}\left(1 - \frac{x^2+y^2}{2\sigma^2}\right)\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \\
&= -\frac{2}{\sigma}\left(1 - \frac{x^2+y^2}{2\sigma^2}\right)G(x,y).
\end{aligned}
$$

Therefore, in 2D we have

$$
\frac{dG(x,y)}{d\sigma} = \sigma\left[\frac{\partial^2 G(x,y)}{\partial x^2} + \frac{\partial^2 G(x,y)}{\partial y^2}\right].
$$

Happily, this is the same result that we obtained in 1D.