Chapter 16

Local Operations

$$g[x, y] = O\{f[x \pm \Delta x, y \pm \Delta y]\}$$

In many common image processing operations, the output pixel is a weighted combination of the gray values of pixels in the neighborhood of the input pixel, hence the term local neighborhood operations. The size of the neighborhood and the pixel weights determine the action of the operator. This concept has already been introduced when we considered image prefiltering during the discussion of realistic image sampling. It will now be formalized and will serve as the basis of the discussion of image transformations.

*Schematic of a local operation applied to the input image $f[x, y]$ to create the output image $g[x, y]$. The local operation weights the gray values in the neighborhood of the input pixel.*
16.1 Window Operators – Correlation

You probably have already been exposed to window operations in the course on linear systems. An example of a window operator acting on the 1-D continuous function \( f[x] \) is:

\[
\mathcal{O}\{f[x]\} = g[x] = \int_{-\infty}^{+\infty} f[\alpha] \gamma[\alpha - x] \, d\alpha
\]

The resulting function of \( x \) is the area of the product of two functions of \( \alpha \): the input \( f \) and a second function \( \gamma \) that has been translated (shifted) by the distance \( x \). Different results are obtained by substituting different functions \( \gamma[x] \).

The process may be recast in a different form by defining a new variable of integration \( u \equiv \alpha - x \):

\[
\int_{-\infty}^{+\infty} f[\alpha] \gamma[\alpha - x] \, d\alpha \rightarrow \int_{u=-\infty}^{u=+\infty} f[x + u] \gamma[u] \, du
\]

which differs from the first expression in that the second function \( \gamma[u] \) remains fixed in position and the input function \( f \) is shifted, now by \( -x \). If the amplitude of the function \( \gamma \) is zero outside some interval in this second expression, then the integral need be computed only over the region where \( \gamma[u] \neq 0 \). The region where the function \( \gamma[x] \) is nonzero is called the support of \( \gamma \), and functions that are nonzero over only a finite domain are said to exhibit finite or compact “support.”

The 2-D versions of these expressions are:

\[
\mathcal{O}\{f[x, y]\} = \iint_{-\infty}^{+\infty} f[\alpha, \beta] \gamma[\alpha - x, \beta - y] \, d\alpha \, d\beta
\]

\[
= \iint_{-\infty}^{+\infty} f[x + u, y + v] \gamma[u, v] \, du \, dv.
\]

The analogous process for sampled functions requires that the integral be converted to a discrete summation:

\[
g[n, m] = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} f[i, j] \gamma[i - n, j - m]
\]

\[
= \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} f[i + n, j + m] \gamma[i, j].
\]

In words, this process scales the shifted function by the values of the matrix \( \gamma \), and thus computes a weighted summation of gray values of the input image \( f[n, m] \). The operation defined by this last equation is called the crosscorrelation of the image with the window function \( \gamma[n, m] \). The correlation operation often is denoted by a
16.1 WINDOW OPERATORS – CORRELATION

five-pointed star (“pentagram”), e.g.,

\[ g[n,m] = f[n,m] \bigstar [n,m] \]

\[ = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} f[i,j] \ \gamma[i-n,j-m] \]

The output image \( g \) at pixel indexed by \([n,m]\) is computed by centering the window \( \gamma[n,m] \) on that pixel of the input image \( f[n,m] \), multiplying the window and input image pixel by pixel, and summing the products. This operation produces an output extremum at shifts \([n,m]\) where the gray-level pattern of the input matches that of the window.

In the common case where the sampled function \( \gamma \) is zero outside a domain with compact support of size \( 3 \times 3 \) samples, the function may be written in the form of a \( 3 \times 3 \) matrix or window function:

\[
\gamma[n,m] = \begin{bmatrix}
\gamma_{-1,1} & \gamma_{0,1} & \gamma_{1,1} \\
\gamma_{-1,0} & \gamma_{0,0} & \gamma_{1,0} \\
\gamma_{-1,-1} & \gamma_{0,-1} & \gamma_{1,-1}
\end{bmatrix}
\]

### 16.1.1 Examples of \( 3 \times 3 \) Crosscorrelation Operators

Consider the action of these \( 3 \times 3 \) window functions:

\[
\gamma_1[n,m] = \begin{bmatrix}
0 & 0 & 0 \\
0 & +1 & 0 \\
0 & 0 & 0
\end{bmatrix}
\]

\[
\gamma_2[n,m] = \begin{bmatrix}
0 & 0 & 0 \\
0 & +2 & 0 \\
0 & 0 & 0
\end{bmatrix}
\]

\[
\gamma_1[n,m] = \begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & +1 \\
0 & 0 & 0
\end{bmatrix}
\]

- \( \gamma_1 \) – the only pixel that influences the output \( g[n,m] \) is the identical pixel in the input \( f[n,m] \) – this is the identity operator.

- \( \gamma_2 \) – the output pixel has twice the gray value of the input pixel – this is a uniform contrast stretching operator.
• \( \gamma_3 \) - the output pixel is identical to its right-hand neighbor in the input image – this operator translates the image one pixel to the left.

Once the general crosscorrelation algorithm is programmed, many useful operations on the image \( f[n, m] \) can be performed simply by specifying different values for the window coefficients.

### 16.2 Convolution

A mathematically equivalent but generally more convenient neighborhood operation is the convolution, which has some very nice mathematical properties. The convolution of two 1-D continuous functions, the input \( f[x] \) and the impulse response (or kernel, or point spread function, or system function) \( h[x] \) is:

\[
  g[x] = f[x] * h[x] \equiv \int_{-\infty}^{\infty} f[\alpha] h[x - \alpha] \, d\alpha.
\]

where \( \alpha \) is a dummy variable of integration. As for the crosscorrelation, the function \( h[x] \) defines the action of the system on the input \( f[x] \). By changing the integration variable to \( u \equiv x - \alpha \), an equivalent expression for the convolution is found:

\[
  g[x] = \int_{-\infty}^{\infty} f[\alpha] h[x - \alpha] \, d\alpha
  = \int_{u=-\infty}^{u=+\infty} f[x - u] h[u] \, (-du)
  = \int_{-\infty}^{\infty} f[x - u] h[u] \, du
  = \int_{-\infty}^{\infty} h[\alpha] f[x - \alpha] \, d\alpha
\]

where the dummy variable was renamed from \( u \) to \( \alpha \) in the last step. Note that the roles of \( f[x] \) and \( h[x] \) have been exchanged between the first and last expressions, which means that the input function \( f[x] \) and system function \( h[x] \) can be interchanged.

The convolution of a continuous 2-D function \( f[x, y] \) with a system function \( h[x, y] \) is denoted by an asterisk “\(*\)” and defined as:

\[
  g[x, y] = f[x, y] * h[x, y]
  \equiv \iiint_{-\infty}^{\infty} f[\alpha, \beta] h[x - \alpha, y - \beta] \, d\alpha \, d\beta
  = \iiint_{-\infty}^{\infty} f[x - \alpha, y - \beta] h[\alpha, \beta] \, d\alpha \, d\beta
\]

Note the difference between the first forms for the convolution and the crosscorre-
16.2 CONVOLUTION

\[ f[x, y] \star [x, y] = \iiint_{-\infty}^{\infty} f[\alpha, \beta] \gamma[\alpha - x, \beta - y] \, d\alpha \, d\beta \]
\[ f[x, y] \ast h[x, y] \equiv \iiint_{-\infty}^{\infty} f[\alpha, \beta] h[x - \alpha, y - \beta] \, d\alpha \, d\beta \]

and between the second forms:
\[ f[x, y] \star [x, y] \equiv \iiint_{-\infty}^{\infty} f[x + u, y + v] \gamma[u, v] \, du \, dv \]
\[ f[x, y] \ast h[x, y] \equiv \iiint_{-\infty}^{\infty} f[x - \alpha, y - \beta] h[\alpha, \beta] \, d\alpha \, d\beta \]

The changes of the order of the variables in the first pair says that the function \( \gamma \) is just shifted before multiplying by \( f \) in the crosscorrelation, while the function \( h \) is flipped about its center (or equivalently rotated about the center by \( 180^\circ \)) before shifting. In the second pair, the difference in sign of the integration variables says that the input function \( f \) is shifted in different directions before multiplying by the system function \( \gamma \) for crosscorrelation and \( h \) for convolution. In convolution, it is common to speak of filtering the input \( f \) with the kernel \( h \). For discrete functions, the convolution integral becomes a summation:
\[ g[n, m] = f[n, m] \ast h[n, m] \]
\[ \equiv \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i, j] h[n - i, m - j] . \]

Again, note the difference in algebraic sign of the action of the kernel \( h[n, m] \) in convolution and the window \( \gamma_{ij} \) in correlation:
\[ f[n, m] \star [n, m] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i, j] \gamma[i - n, j - m] \]
\[ f[n, m] \ast h[n, m] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i, j] h[n - i, m - j] . \]

This form of the convolution has the very useful property that convolution of an input in the form of an impulse function \( \delta_d[n, m] \) with \( h[n, m] \) yields \( h[n, m] \), hence the name for \( h \) as the impulse response:
\[ \delta_d[n, m] \ast h[n, m] = h[n, m] \]
where the discrete Dirac delta function $\delta_{d}[n, m]$ is defined:

$$
\delta_{d}[n, m] \equiv \begin{cases} 
1 & \text{if } n = m = 0 \\
0 & \text{otherwise}
\end{cases}
$$

16.2.1 Evaluating Discrete Convolutions

1-D

Schematic of the sequence of calculations in 1-D discrete convolution. The 3-pixel kernel $h[n] = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$ is convolved with the input image that is 1 at one pixel and zero elsewhere. The output is a replica of $h[n]$ centered at the location of the impulse.
2-D

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{array}
\begin{array}{cccc}
1 & 1 & 1 \\
2 & 4 & 2 \\
1 & 2 & 1 \\
\end{array}
= 
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 \\
0 & 0 & 2 & 4 \\
0 & 0 & 1 & 2 \\
0 & 0 & 0 & 0 \\
\end{array}
\]

Schematic of 2-D discrete convolution with the 2-D kernel \(h[n, m]\).

\[
\delta [i - n, j - m] * h[n, m] = h[n, m]
\]

Examples of 2-D Convolution Kernels

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & +1 & 0 \\
0 & 0 & 0 \\
\end{array}
\Rightarrow \text{identity}
\]

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & +1 \\
0 & 0 & 0 \\
\end{array}
\Rightarrow \text{shifts image one pixel to right}
\]

Discrete convolution is linear because it is defined by a weighted sum of pixel gray values:

\[
f[n, m] * (h_1[n, m] + h_2[n, m]) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} f[i, j] \cdot (h_1[n - i, m - j] + h_2[n - i, m - j])
\]

\[
= \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} (f[i, j] \cdot h_1[n - i, m - j] + f[i, j] \cdot h_2[n - i, m - j])
\]

\[
= \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} f[i, j] \cdot h_1[n - i, m - j] + \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} f[i, j] \cdot h_2[n - i, m - j]
\]
The linearity of convolution allows new kernels to be created from sums or differences of other kernels. For example, consider the sum of three $3 \times 3$ kernels whose actions have already been considered:

$$h[n,m] = \frac{1}{3} \begin{bmatrix} 0 & 0 & 0 \\ 0 & +1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \frac{1}{3} \begin{bmatrix} 0 & +1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \frac{1}{3} \begin{bmatrix} 0 & 0 & 0 \\ 0 & +1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The output image $g[n,m]$ is the average of three images: the input and copies shifted one pixel up and down. Therefore, each pixel in $g[n,m]$ is the average of three pixels in a vertical line; $g[n,m]$ is blurred vertically. Note that the kernels have been normalized so that the sum of the elements is unity. This ensures that the gray level of the filtered image will fall within the dynamic range of the input image, but they may not be integers. The output of a lowpass filter must typically be requantized.

### 16.2.2 Convolutions – Edges of the Image

Because a convolution is the sum of weighted gray values in the neighborhood of the input pixel, there is a question of what to do near the edge of the image, i.e., when the neighborhood of pixels in the kernel extends “over the edge” of the image. The common solutions are:

1. consider any pixel in the neighborhood that would extend off the image to have gray value “0”;
2. consider pixels off the edge to have the same gray value as the edge pixel;
3. consider that the convolution in any such case to be undefined; and
4. define any pixels over the edge of the image to have the same gray value as pixels on the opposite edge.

On the face of it, the fourth of these alternatives may seem to be ridiculous, but it is simply a statement that the image is assumed to be periodic, i.e., that:

$$f[n,m] = f[n+kN,m+\ell M]$$

where $N$ and $M$ are the numbers of pixels in a row or column, respectively, and $k, \ell$ are integers. In fact, this is the most common case, and will be treated in depth when global operators are discussed.
Possible strategies for dealing with the edge of the image in 2-D convolution: (a) the input image is padded with zeros; (b) the input image is padded with the same gray values “on the edge;” (c) values “off the edge” are ignored; (d) pixels off the edge are assigned the values on the opposite edge, this assumes that the input image is periodic.

The 3 × 3 image $f[n, m]$ is outlined by the bold-face box and the assumed gray values of pixels off the edge of the image are shown in light face for four cases; the presence of an “x” in a convolutio kernel indicates that the output gray value is undefined.

### 16.2.3 Convolutions — Computational Intensity

Evaluating convolutions with large kernels in a serial processor used to be very slow. For example, convolution of a 512²-pixel image with an $M \times M$ kernel requires: $2 \cdot 512^2 \cdot M^2$ operations (multiplications and additions) for a total of $4.7 \cdot 10^6$ operations with a $3 \times 3$ kernel and $25.7 \cdot 10^6$ operations with a $7 \times 7$ (these operations generally are performed on floating-point data). The increase in computations as $M^2$ ensures that convolution of large images with large kernels is not very practical by serial brute-force means. In the discussion of global operations to follow, we will introduce an alternative method for computing convolutions via the Fourier transform that requires many fewer operations for large images.

### 16.2.4 Smoothing Kernels — Lowpass Filtering

If all elements of a convolution kernel have the same algebraic sign, then the operator $O$ sums weighted gray values of input pixels in the neighborhood; if the sum of the elements is one, then the process computes a weighted average of the gray values. Averaging reduces the variability of the gray values of the input image; it “smooths” the function:
Local averaging decreases the “variability” (variance) of pixel gray values

For a uniform averaging kernel of a fixed size, functions that oscillate over a period just longer than the kernel (e.g., short-period, high-frequency sinusoids) will be averaged more than slowly varying terms. In other words, local averaging attenuates the high sinusoidal frequencies while passing the low frequencies relatively undisturbed – local averaging operators are lowpass filters. If the kernel size doubles, input sinusoids with twice the period (half the spatial frequency) will be equivalently affected. This action was discussed in the section on realistic sampling; a finite detector averages the signal over its width and reduces modulation of the output signal to a greater degree at higher frequencies.

Local averaging operators are lowpass filters

Obviously, averaging kernels reduce the visibility of additive noise by spreading the difference in gray value of noise pixel from the background over its neighbors. By analogy with temporal averaging, spatial averaging of noise increases SNR by the square-root of the number of pixels averaged if the noise is random and the averaging weights are identical.

The action of an averager can be directional:

\[
h_1[n, m] = \frac{1}{3} \begin{bmatrix} 0 & +1 & 0 \\ 0 & +1 & 0 \\ 0 & +1 & 0 \end{bmatrix} \text{ averages vertically}
\]

\[
h_2[n, m] = \frac{1}{3} \begin{bmatrix} 0 & 0 & 0 \\ +1 & +1 & +1 \\ 0 & 0 & 0 \end{bmatrix} \text{ blurs horizontally}
\]

\[
h_3[n, m] = \frac{1}{3} \begin{bmatrix} +1 & 0 & 0 \\ 0 & +1 & 0 \\ 0 & 0 & +1 \end{bmatrix} \text{ blurs diagonally}
\]

The “rotation” or “reversal” of the convolution kernel means that the action of \(h_3[n, m]\) blurs diagonally along the direction at 90° that in the kernel.
The elements of an averaging kernel need not be identical, e.g.,

\[ h[n, m] = \frac{1}{3} \begin{bmatrix} +\frac{1}{4} & +\frac{1}{4} & +\frac{1}{4} \\ +\frac{1}{4} & +1 & +\frac{1}{4} \\ +\frac{1}{4} & +\frac{1}{4} & +\frac{1}{4} \end{bmatrix} \]

averages over the entire window but the output is primarily influenced by the center pixel; the output blurred less than in the case when all elements are identical.

Other 2-D discrete averaging kernels may be constructed by “orthogonal multiplication,” e.g., we can construct the common \(3 \times 3\) uniform averager via the product of two orthogonal 1-D uniform averagers:

\[
\begin{bmatrix} +\frac{1}{3} & +\frac{1}{3} & +\frac{1}{3} \\ +\frac{1}{3} \\ +\frac{1}{3} \end{bmatrix} \cdot \begin{bmatrix} +\frac{1}{3} \\ +\frac{1}{3} \\ +\frac{1}{3} \end{bmatrix} = \begin{bmatrix} +\frac{1}{9} & +\frac{1}{9} & +\frac{1}{9} \\ +\frac{1}{9} & +\frac{1}{9} & +\frac{1}{9} \\ +\frac{1}{9} & +\frac{1}{9} & +\frac{1}{9} \end{bmatrix} = \begin{bmatrix} +\frac{1}{3} & +\frac{1}{3} & +\frac{1}{3} \\ +\frac{1}{3} & +\frac{1}{3} & +\frac{1}{3} \\ +\frac{1}{3} & +\frac{1}{3} & +\frac{1}{3} \end{bmatrix}
\]

The associated transfer function is the orthogonal product of the individual 1-D transfer functions. Note that the 1-D kernels can be different, such as a 3-pixel uniform averager along the \(n\)-direction and a 2-pixel uniform averager along \(m\):

\[
\begin{bmatrix} +\frac{1}{3} & +\frac{1}{3} & +\frac{1}{3} \\ +\frac{1}{2} \\ +\frac{1}{4} \end{bmatrix} \cdot \begin{bmatrix} +\frac{1}{4} \\ +\frac{1}{4} \\ +\frac{1}{4} \end{bmatrix} = \begin{bmatrix} +\frac{1}{12} & +\frac{1}{12} & +\frac{1}{12} \\ +\frac{1}{12} & +\frac{1}{12} & +\frac{1}{12} \\ +\frac{1}{12} & +\frac{1}{12} & +\frac{1}{12} \end{bmatrix} = \begin{bmatrix} +\frac{1}{3} & +\frac{1}{3} & +\frac{1}{3} \\ +\frac{1}{3} & +\frac{1}{3} & +\frac{1}{3} \\ +\frac{1}{3} & +\frac{1}{3} & +\frac{1}{3} \end{bmatrix}
\]

Lowpass-Filtered Images

Examples of lowpass filtering: (a) \(f[n, m]\); (b) after uniform averaging over a \(3 \times 3\) neighborhood; (c) after uniform averaging over a \(5 \times 5\) neighborhood. Note that the “fine structure” (such as it is) becomes less visible as the neighborhood size increases.
16.2.5 Frequency Response of 1-D Averagers

The impulse response of the linear operator that averages uniformly is a unit-area rectangle:

$$h[x] = \frac{1}{|b|} RECT\left[\frac{x}{b}\right]$$

where $b$ is the width of the averaging region. The corresponding continuous transfer function is:

$$H[\xi] = SINC\left[\frac{\xi}{(1/b)}\right]$$

In the discrete case, the rectangular impulse response is sampled and the width $b$ is measured in units of $\Delta x$. If $\frac{b}{\Delta x}$ is even, the amplitudes of the endpoint samples are $\frac{b}{2}$. We consider the cases where $b = 2 \cdot \Delta x$, $3 \cdot \Delta x$, and $4 \cdot \Delta x$. The discrete impulse responses of uniform averagers that are two and three pixels wide have three nonzero samples:

For $b = 2 \cdot \Delta x$:

$$h_2[n] = \begin{array}{c}
+\frac{1}{4} \\
+\frac{1}{2} \\
+\frac{1}{4}
\end{array} = \frac{1}{4} (\delta_d[n + 1] + 2 \cdot \delta_d[n] + \delta_d[n - 1])$$

For $b = 3 \cdot \Delta x$:

$$h_3[n] = \begin{array}{c}
+\frac{1}{3} \\
+\frac{1}{3} \\
+\frac{1}{3}
\end{array} = \frac{1}{3} (\delta_d[n + 1] + \delta_d[n] + \delta_d[n - 1])$$

The linearity of the DFT ensures that the corresponding transfer function may be constructed by summing transfer functions for the identity operator and for translations by one sample each to the left and right. The resulting transfer functions may be viewed as discrete approximations to the continuous $SINC$ functions:

$$H_2[k] = \frac{1}{4} e^{-2\pi i k \cdot \Delta \xi} + \frac{1}{2} + \frac{1}{4} e^{+2\pi i k \cdot \Delta \xi}; \quad -\frac{N}{2} \leq k \leq \frac{N}{2} - 1$$

$$H_2[k] = \frac{1}{2} \left(1 + \cos \left[2\pi (k \cdot \Delta \xi)\right]\right) \equiv SINC_d\left[k; \frac{N}{2}\right]$$

$$H_3[k] = \frac{1}{3} e^{-2\pi i k \cdot \Delta \xi} + \frac{1}{3} + \frac{1}{3} e^{+2\pi i k \cdot \Delta \xi}$$

$$H_3[k] = \frac{1}{3} \left(1 + 2 \cos \left[2\pi (k \cdot \Delta \xi)\right]\right) \equiv SINC_d\left[k; \frac{N}{3}\right]$$

Note that both $H_2[k]$ and $H_3[k]$ have unit amplitude at the origin because of the discrete central ordinate theorem. The “zero-crossings” of the two-pixel averager would be located at $\xi = \pm \frac{1}{2}$ cycle per sample, i.e., at the positive and negative Nyquist frequencies, but the index convention admits only the negative index, $k = -\frac{N}{2}$. This sinusoid oscillates with a period of two samples and is thus averaged to zero by the two-pixel averager. The three-pixel averager should “block” any sinusoid with a period $3 \cdot \Delta x$, which can certainly be constructed because the Nyquist condition is satisfied. The zero crossings of the transfer function “should” be located at $\xi = \pm \frac{1}{3}$ cycle per sample, which “would” occur at noninteger indices $k = \pm \frac{N}{3}$. However,
because $\pm \frac{N}{3}$ is not an integer when $N$ is even, the zero crossings of the spectrum occur between samples in the discrete spectrum. In other words, the discrete transfer function of the three-pixel averager has no zeros and thus must “pass” all sampled and unaliased sinusoidal components (albeit with attenuation). This seems to be a paradox — a sinusoid with a period of three pixels can be sampled without aliasing but this function is not “blocked” by a three-pixel averager, whereas the two-pixel sinusoid is blocked by a two-pixel averager. The reason is because there are a noninteger number of periods of length $3 \cdot \Delta x$ in an array where $N$ is even. Thus there will be “leakage” in the spectrum of the sampled function. The resulting “spurious” frequency components pass reach the output. As a final observation, note also that the discrete transfer functions approach the edges of the array “smoothly” (without “cusps”) in both cases, as shown in the figure.

Comparison of 2- and 3-pixel uniform averagers for $N = 64$: (a) central region of the impulse response $h_2[n] = \frac{1}{2} \text{RECT} \left[ \frac{n}{2} \right]$; (b) discrete transfer function $H_2[k]$ compared to the continuous analogue $H_2[\xi] = \text{SINC} \left[ \frac{2 \xi}{3} \right]$ out to the Nyquist frequency $\xi = \frac{1}{2} \frac{\text{cycle}}{\text{sample}}$; (c) $h_3[n] = \frac{1}{3} \text{RECT} \left[ \frac{n}{3} \right]$, which has the same support as $h_2[n]$; (d) $H_3[k]$ compared to $H_3[\xi] = \text{SINC} \left[ \frac{3 \xi}{3} \right]$, showing the “smooth” transition at the edge of the array.

The discrete impulse response of the four-pixel averager has five nonzero pixels:
\[ b = 4 \cdot \Delta x : h_4[n] = \begin{bmatrix} \frac{1}{8} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{8} \end{bmatrix} \]

The linearity of the DFT ensures that the corresponding transfer function may be constructed by summing the transfer function of the three-pixel averager scaled by \( \frac{3}{4} \) with the transfer functions for translation by two samples each to the left and right:

\[
H_4[k] = \frac{1}{4} \left( \frac{1}{2} e^{-2\pi ik \cdot 2\Delta \xi} + e^{-2\pi ik \cdot \Delta \xi} + 1 + e^{2\pi ik \cdot \Delta \xi} + \frac{1}{2} e^{2\pi ik \cdot 2\Delta \xi} \right)
\]

\[
= \frac{1}{4} \left( 1 + 2 \cos \left[ 2\pi \left( k \cdot \Delta \xi \right) \right] + \cos \left[ 2\pi \left( k \cdot 2\Delta \xi \right) \right] \right)
\]

which also may be thought of as a discrete approximation of a \( SINC \) function: \( SINC_d \left[ k; \frac{N}{4} \right] \). This discrete transfer function has zeros located at \( \xi = \pm \frac{1}{4} \) cycle per sample, which correspond to \( k = \pm \frac{N}{4} \). Therefore the four-pixel averager “blocks” any sampled sinusoid with period \( 4 \cdot \Delta x \) from reaching the output. Again the transfer function has “smooth” transitions of amplitude at the edges of the array, thus preventing “cusps” in the periodic spectrum, as shown in the figure.
16.2 CONVOLUTION

The general expression for the discrete SINC function in the frequency domain suggested by these results for \(-\frac{N}{2} \leq k \leq \frac{N}{2} - 1\) is:

\[
SINC_d \left[ k; \frac{N}{w} \right] = \begin{cases} 
\frac{1}{w} \left( 1 + 2 \sum_{\ell=1}^{\frac{w-1}{2}} \cos \left[ 2\pi \left( k \cdot \ell \cdot \Delta \xi \right) \right] \right) & \text{if } w \text{ is odd} \\
\frac{1}{w} \left( 1 + \cos \left[ 2\pi \left( k \cdot \frac{w}{2} \cdot \Delta \xi \right) \right] + 2 \sum_{\ell=1}^{\frac{w-1}{2}} \cos \left[ 2\pi \left( k \cdot \ell \cdot \Delta \xi \right) \right] \right) & \text{if } w \text{ is even}
\end{cases}
\]

16.2.6 2-D Averagers

Effect of Lowpass Filtering on the Histogram

Because an averaging kernel reduces pixel-to-pixel variations in gray level (and hence the variance of additive random noise in the image), we would expect that clusters of pixels in the histogram of an averaged image to be taller and thinner than in the original image. It should be easier to segment objects based on average gray level from the histogram of an averaged image. To illustrate, we reconsider the example of the house-tree image. The image in blue light and its histogram before and after averaging with a \(3 \times 3\) kernel are shown below: Note that there are four fairly distinct clusters in the histogram of the averaged image, corresponding to the house, grass+tree, sky, and clouds+door (from dark to bright). The small clusters at the ends are more difficult to distinguish on the original histogram.
Effect of blurring on the histogram: the 64 × 64 color image, the histograms of the 3 bands, and the 3 2-D histograms are shown at top; the same images and histograms after blurring with a 3 × 3 kernel are at the bottom, showing the concentration of histogram clusters resulting from image blur.

Note that the noise visible in uniform areas of the images (e.g., the sky in the blue image) has been noticeably reduced by the averaging, and thus the widths of the histogram clusters have decreased. The downside of this process is that pixels on the boundaries of objects now exhibit blends of the colors of both bounding objects, and thus will not be as easy to segment.

### 16.2.7 Differencing Kernels — Highpass Filters

A kernel with both positive and negative terms computes differences of neighboring pixels. From the previous discussion, it is probably apparent that the converse of the
The difference of adjacent pixels with identical gray levels will cancel out, while differences between adjacent pixels will be amplified. Since high-frequency sinusoids vary over shorter distances, differencing operators will enhance them and attenuate slowly varying (i.e., lower-frequency) terms.

Differencing operators are highpass filters because a differencing operator will “block” low-frequency sinusoids and “pass” those with high frequencies.

Subtraction of adjacent pixels can result in negative gray values; this is the spatial analogy of temporal differencing for change detection. The gray values of the output image must be biased “up” for display by adding some constant gray level to all image pixels, e.g., if \( g_{\text{min}} < 0 \), then the negative gray values may be displayed by adding the level \( |g_{\text{min}}| \) to all pixel gray values.

The discrete analogue of differentiation may be derived from the definition of the continuous derivative:

\[
\frac{df}{dx} \equiv \lim_{\tau \to 0} \left( \frac{f[x + \tau] - f[x]}{\tau} \right)
\]

In the discrete case, the smallest nonzero value of \( \tau \) is the sampling interval \( \Delta x \), and thus the discrete derivative is:

\[
\frac{1}{\Delta x} (f[(n+1) \cdot \Delta x] - f[n \cdot \Delta x]) = \frac{1}{\Delta x} f[n \cdot \Delta x] * (\delta[n + 1] - \delta[n])
\]

In words, the discrete derivative is the scaled difference of the value at the sample indexed by \( n + 1 \) and by \( n \). By setting \( \Delta x = 1 \) sample, the leading scale factor may be ignored. The 1-D derivative operator may be implemented by discrete convolution with a 1-D kernel that has two nonzero elements; we will write it with three elements to clearly denote the sample indexed by \( n = 0 \).

\[
f[n] * (\delta[n + 1] - \delta[n]) = f[n] * \begin{bmatrix} 1 & -1 & 0 \end{bmatrix} \equiv f[n] * \partial[n]
\]

where \( \partial[n] \equiv \begin{bmatrix} +1 & -1 & 0 \end{bmatrix} \) is the discrete impulse response of differentiation, which is perhaps better called a differencing operator. Note that the impulse response may be decomposed into its even and odd parts.

\[
\partial_{\text{even}}[n] = \begin{bmatrix} +1 & -1 & +1 \end{bmatrix} / 2 = \begin{bmatrix} +1 & +1 & +1 \end{bmatrix} / \frac{3}{2} - \begin{bmatrix} +1 & 0 & 0 \end{bmatrix} / \frac{3}{2}
\]

\[
\partial_{\text{odd}}[n] = \begin{bmatrix} +1 & 0 & -1 \end{bmatrix} / 2 = \begin{bmatrix} +1 & 0 & -1 \end{bmatrix} / 2
\]
The even part is a weighted difference of the identity operator and the three-pixel averager, while the odd part computes differences of pixels separated by two sample intervals.

The corresponding 2-D derivative kernel in the $x$-direction is:

$$h[n, m] = \begin{bmatrix} 0 & 0 & 0 \\ +1 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ +1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \equiv \partial_x$$

The resulting image is equivalent to the difference between an image translated one pixel to the left and an unshifted image, i.e.,

$$\frac{\partial}{\partial x} f[x, y] = \lim_{\Delta x \to 0} \frac{f[x + \Delta x, y] - f[x, y]}{\Delta x} \Rightarrow \frac{\partial}{\partial x} f[x, y] \equiv f[(n + 1) \cdot \Delta x, m \cdot \Delta y] - f[n \cdot \Delta x, m \cdot \Delta y]$$

$$\Rightarrow \partial_x * f[n, m] = f[n + 1, m] - f[n, m]$$

because the minimum nonzero value of the translation $\Delta x = 1$ sample. The corresponding discrete partial derivative in the $y$-direction is:

$$\partial_y * f[n, m] \equiv f[n, m + 1] - f[n, m]$$

The partial derivative in the $y$-direction is the difference of a replica translated “up” and the original:

$$\partial_y = \begin{bmatrix} 0 & +1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

This definition of the derivative effectively “locates” the edge of an object at the pixel immediately to the right or above the “crack” between pixels that is the actual edge.
16.2 CONVOLUTION

2-D Edge image and the first derivative obtained by convolving with the 3 × 3 first-derivative kernel in the x-direction, showing that the detected edge in the derivative image is located at the pixel to the right of the edge transition.

“Symmetric” (actually antisymmetric or odd) versions of the derivative operators are sometimes used that evaluate the difference across two pixels:

\[
\partial_x = \begin{bmatrix}
0 & 0 & 0 \\
+1 & 0 & -1 \\
0 & 0 & 0
\end{bmatrix}
\]

\[
\partial_y = \begin{bmatrix}
0 & +1 & 0 \\
0 & 0 & 0 \\
0 & -1 & 0
\end{bmatrix}
\]

These operators locate the edge of an object between two pixels symmetrically, but produce an output that is two pixels wide centered about the location of the edge:

2-D Edge image and the first derivative obtained by convolving with the 3 × 3 “symmetric” first-derivative kernel in the x-direction, showing that the edge is a two-pixel band symmetrically placed about the transition.
Higher-Order Derivatives

The kernels for higher-order derivatives are easily computed because convolution is associative. The convolution kernel for the 1-D second derivative is obtained by autoconvolving the kernel for the 1-D first derivative:

\[
\frac{\partial^2}{\partial x^2} f[x, y] = \frac{\partial}{\partial x} \left( \frac{\partial}{\partial x} f[x, y] \right) \\
= \frac{\partial}{\partial x} \left( \lim_{\Delta x \to 0} \frac{f[x + \Delta x, y] - f[x, y]}{\Delta x} \right) \\
= \lim_{\Delta x \to 0} \left( \lim_{\Delta x \to 0} \left( \frac{f[x + 2 \cdot \Delta x, y] - f[x + \Delta x, y]}{\Delta x} \right) - \lim_{\Delta x \to 0} \left( \frac{f[x + \Delta x, y] - f[x, y]}{\Delta x} \right) \right) \\
= \lim_{\Delta x \to 0} \left( \frac{f[x + 2 \cdot \Delta x, y] - 2f[x + \Delta x, y] + f[x, y]}{\Delta x} \right) \\
\Rightarrow \partial_x^2 \ast f[n, m] \equiv f[n + 2, m] - 2f[n + 1, m] - f[n, m]
\]

which may be evaluated by convolution with a five-element kernel, which is displayed in a $5 \times 5$ window to identify the center pixel

\[
\begin{array}{ccccc}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
+1 & -2 & +1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

This usually is “centered” in a $3 \times 3$ array by translating the kernel one pixel to the right and lopping off the zeros:

\[
\begin{array}{ccc}
0 & 0 & 0 \\
+1 & -2 & +1 \\
0 & 0 & 0 \\
\end{array}
\]

Except for the one-pixel translation, this operation generates the same image produced by the cascade of two first-derivative operators. The corresponding 2-D second
partial derivative kernels in the y-direction are:

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & +1 & 0 \\
0 & 0 & -2 & 0 \\
0 & 0 & +1 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & +1 & 0 \\
0 & -2 & 0 \\
0 & +1 & 0 \\
\end{array}
\]

The derivation may be extended to derivatives of still higher order by convolving kernels to obtain the kernels for the 1-D third and fourth derivatives. The third derivative may be displayed in a 7 × 7 kernel:

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
+1 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{array} \ast \begin{array}{cccc}
0 & 0 & 0 & 0 \\
+1 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{array} \ast \begin{array}{cccc}
0 & 0 & 0 & 0 \\
+1 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{array}
\]

which also is usually translated and truncated:

\[
\begin{array}{cccc}
0 & +1 & 0 & 0 \\
0 & -3 & +3 & -1 \\
0 & 0 & 0 & 0 \\
\end{array}
\]

The gray-level extrema of the image produced by a differencing operator indicate pixels in regions of rapid variation, e.g., at the edges of distinct objects. The visibility of these pixels can be further enhanced by a subsequent contrast enhancement or thresholding operation.
Because they compute weighted differences in pixel gray value, differencing operators also will enhance the visibility of noise in an image. Consider the 1-D example where the input image \( f[x] \) is a 3-bar chart with added noise, so that the signal-to-noise ratio (SNR) of 4. The convolution of the input \( f[x] \) with an averaging kernel \( h_1[x] = \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \) and with differencing kernel \( h_2[x] = \begin{bmatrix} -1 & 3 & -1 \end{bmatrix} \) are shown:

**Effect of averaging and sharpening on an image with noise:** (a) \( f[n] + \text{noise} \); (b) after averaging over a 3-pixel neighborhood with \( h_1[n] = \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \); (c) after applying a sharpening operator \( h_2[n] = \begin{bmatrix} -1 & 3 & -1 \end{bmatrix} \)

Note that the noise is diminished by the averaging kernel and enhanced by the differencing kernel.

### 16.2.8 Frequency Responses of 2-D Derivative Operators:

#### First Derivatives

The 2-D \( x \)-derivative operator denoted by \( \partial_x \):

\[
\partial_x = \begin{bmatrix}
0 & 0 & 0 \\
+1 & -1 & 0 \\
0 & 0 & 0
\end{bmatrix}
\]

has the associated discrete transfer function obtained via the discrete Fourier transform:

\[
\mathcal{F}_2 \{ \partial_x \} = H_{\partial_x}[k, \ell] = \left( -1 + e^{\pm 2\pi i \frac{k}{N}} \right) \cdot 1[\ell] \\
= \left( -1[k, \ell] + \cos \left( 2\pi \frac{k}{N} \right) \cdot 1[\ell] \right) + i \left( \sin \left( 2\pi \frac{k}{N} \right) \cdot 1[\ell] \right)
\]
16.2 CONVOLUTION

The differencing operator in the $y$-direction and its associated transfer function are obtained by rotating the expressions just derived by $+\frac{\pi}{2}$ radians. The kernel is:

$$\partial_y = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & +1 & 0 \end{bmatrix}$$

The corresponding transfer function is the rotated version of $F_2\{\partial_x\}$:

$$F_2\{\partial_y\} = H_{(\partial_y)}[k, \ell] = 1[k] \cdot \left( -1 + e^{+2\pi i \frac{\ell}{N}} \right)$$

We can also define 2-D derivatives along angles. We can also define differences along the diagonal directions:

$$\partial_{(\theta=+\frac{\pi}{4})} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ +1 & 0 & 0 \end{bmatrix}$$

$$\partial_{(\theta=+\frac{3\pi}{4})} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & +1 \end{bmatrix}$$

The angle in radians has been substituted for the subscript. Again, the continuous distance between the elements has been scaled by $\sqrt{2}$.

1-D Antisymmetric Differentiation Kernel

We can also construct a discrete differentiator with odd symmetry by placing the components of the discrete “doublet” at samples $n = \pm 1$:

$$(\partial_x)_2 = \begin{bmatrix} +1 & 0 & -1 \end{bmatrix}$$

This impulse response is proportional to the odd part of the original 1-D differentiator. The corresponding transfer function is again easy to evaluate via the appropriate combination of translation operators. Because $(\partial_x)_2$ is odd, the real part of the discrete transfer function is zero, as shown in the figure:
\[ H[k] = \exp \left[ +2\pi i \left( \frac{k}{N} \right) \right] - \exp \left[ +2\pi i \left( -\frac{k}{N} \right) \right] = 2i \sin \left[ 2\pi \frac{k}{N} \right] \]

\[ |H[k]| = 2 \left| \sin \left[ 2\pi \frac{k}{N} \right] \right| \]

\[ \Phi \{ H[k] \} = +\frac{\pi}{2} \left( \text{SGN} [k] - \delta_d \left[ k + \frac{N}{2} \right] \right) \]

Note that this transfer function evaluated at the Nyquist frequency is:

\[ \left| H \left[ k = \frac{-N}{2} \right] \right| = 2i \sin \left[ -\pi \right] = 0 \]

which means that this differentiator also “blocks” the Nyquist frequency. This may be seen by convolving \((\partial_x)^2\) with a sinusoid function that oscillates with a period of two samples. Adjacent positive extrema are multiplied by ±1 in the kernel and thus cancel. Also note that the transfer function amplifies lower frequencies more and larger frequencies less than the continuous transfer function.
1-D discrete antisymmetric derivative operator and its transfer function: (a) 1-D impulse response \( h[n] \); Because \( h[n] \) is odd, \( H[k] \) is imaginary and odd. The samples of the discrete transfer function are shown as (b) real part, (c) imaginary part, (d) magnitude, and (e) phase, along with the corresponding continuous transfer function \( H[\xi] = i2\pi\xi \). Note that the magnitude of the discrete transfer function is attenuated near the Nyquist frequency and that its phase is identical to that of the transfer function of the continuous derivative.
1-D Second Derivative

The impulse response and transfer function of the continuous second derivative are easily obtained from the derivative theorem:

\[ h[x] = \delta''[x] \]
\[ H[\xi] = (2\pi i \xi)^2 = -4\pi^2 \xi^2 \]

Again, different forms of the discrete second derivative may be defined. One form is obtained by differentiating the first derivative operator via discrete convolution of two replicas of \( \partial_x \). The result is a five-pixel kernel including two null weights:

\[
\partial_x * \partial_x = \begin{bmatrix} +1 & -1 & 0 & +1 & -1 & 0 \\
 0 & +1 & 0 & -2 & +1 & 0 \\
 \end{bmatrix}
\]

The corresponding discrete transfer function is obtained by substituting results from the translation operator:

\[
H[k] = e^{+2\pi i \frac{k}{N}} - 2 e^{+2\pi i \frac{k}{N}} + 1[k]
\]
\[
= e^{+2\pi i \frac{k}{N}} \left( e^{+2\pi i \frac{k}{N}} - 2 e^{-2\pi i \frac{k}{N}} \right)
\]
\[
= 2 e^{+2\pi i \frac{k}{N}} \left( \cos \left( \frac{2\pi k}{N} \right) - 1 \right)
\]

The leading linear phase factor usually is discarded to produce the real-valued and symmetric discrete transfer function:

\[
H[k] = 2 \left( \cos \left( \frac{2\pi k}{N} \right) - 1 \right)
\]

Deletion of the linear phase is the same as translation of the original discrete second derivative kernel by one pixel to the right. The discrete impulse response for this symmetric discrete kernel is also real valued and symmetric:

\[ h[n] = \partial_x^2 \equiv \begin{bmatrix} 0 & +1 & -2 & +1 & 0 \end{bmatrix} = \begin{bmatrix} +1 & -2 & +1 & +1 \end{bmatrix} \]

and the magnitude and phase of the transfer function are:
$|H[k]| = 2 \left( 1 - \cos \left( \frac{2\pi k}{N} \right) \right)$

$\Phi \{H[k]\} = \begin{cases} 
-\pi & \text{for } k \neq 0 \\
0 & \text{for } k = 0 
\end{cases} = \pi (-1 + \delta_d[k])$

as shown in the figure. The amplitude of the discrete transfer function at the Nyquist frequency is:

$H \left[ k = -\frac{N}{2} \right] = 2 \cdot (\cos [-\pi] - 1) = -4$

while that of the continuous transfer function is $-4\pi^2 \left( -\frac{1}{2} \right)^2 = -\pi^2 \cong -9.87$, so the discrete second derivative does not amplify the amplitude at the Nyquist frequency as much as the continuous second derivative. The transfer function is a discrete approximation of the parabola and again approaches the edges of the array to ensure smooth periodicity.

Higher-order discrete derivatives may be derived by repeated discrete convolution of $\partial_x^2$, after discarding any linear-phase factors.

1-D Discrete second derivative: (a) Impulse response $\partial_x^2$; (b) comparison of discrete and continuous transfer functions.

### 16.2.9 Laplacian Operator

The Laplacian operator for continuous functions was introduced in the discussion of electromagnetism. It is the sum of orthogonal second partial derivatives:

$\nabla^2 f [x, y] \equiv \left( \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) f [x, y]$
The associated transfer function is the negative quadratic that evaluates to 0 at DC, which again demonstrates that constant terms are blocked by differentiation:

\[ H [ \xi, \eta] = -4\pi^2 (\xi^2 + \eta^2) \]

The discrete Laplacian operator is the sum of the orthogonal 2-D second-derivative kernels:

\[
\begin{align*}
\partial_x^2 + \partial_y^2 \equiv \nabla_d^2 &= \begin{bmatrix} 0 & 0 & 0 \\ +1 & -2 & +1 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & +1 & 0 \\ 0 & -2 & 0 \\ 0 & +1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & +1 & 0 \\ +1 & -4 & +1 \\ 0 & +1 & 0 \end{bmatrix}
\end{align*}
\]

The discrete transfer function of this “standard” discrete Laplacian kernel is:

\[
H [k,c] = 2 \left( \cos \left( \frac{2\pi k}{N} \right) - 1 \right) + 2 \left( \cos \left( \frac{2\pi c}{N} \right) - 1 \right)
= 2 \left( \cos \left( \frac{2\pi k}{N} \right) + \cos \left( \frac{2\pi c}{N} \right) - 2 \right)
\]

The amplitude at the origin is \( H [k = 0, \ell = 0] = 0 \) and decays in the horizontal or vertical directions to \(-6\) at the “edge” of the discrete array and to \(-8\) at its corners.

**Rotated Laplacian**

The sum of the second-derivative kernels along the diagonals creates a rotated version of the Laplacian, which is “nearly” equivalent to rotating the operator \( \nabla_d^2 \) by \( \theta = +\frac{\pi}{4} \) radians:

\[
\left( \partial_x^2 \left( +\frac{\pi}{4} \right) + \partial_y^2 \left( +\frac{3\pi}{4} \right) \right) = \nabla_d^2 \left( +\frac{\pi}{4} \right) = \begin{bmatrix} 0 & 0 & +1 \\ 0 & -2 & 0 \\ +1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} +1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & +1 \end{bmatrix}
= \begin{bmatrix} +1 & 0 & +1 \\ 0 & -4 & 0 \\ +1 & 0 & +1 \end{bmatrix}
\]

Derivation of the transfer function is left to the student; its magnitude is zero at the origin and its maximum negative values are located at the horizontal and vertical edges, but the transfer function zero at the corners.
The real and symmetric transfer functions of Laplacian operators: (a) “normal” Laplacian from Eq.(20.50); (b) rotated Laplacian from Eq.(20.51), showing that the amplitude rises back to 0 at the corners of the array.

A 2-D example is shown in the figure, where the input function is nonnegative and the bipolar output $g[n, m]$ is displayed as amplitude and as magnitude $|g[n, m]|$, which shows that the response is largest at the edges and corners.
Action of the 2-D discrete Laplacian. The input amplitudes are nonnegative in the interval \(0 \leq f \leq 1\), and the output amplitude is bipolar in the interval \(-2 \leq g \leq +2\) in this example. As shown in the magnitude image, the extrema of output amplitude occurs at the edges and corners of the input.
Isotropic Laplacian:

A commonly used “isotropic” Laplacian is obtained by summing the original and rotated Laplacian kernels:

\[
\left( \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) + \left( \frac{\partial^2}{\partial \left(\frac{\pi}{4}\right)} + \frac{\partial^2}{\partial \left(\frac{3\pi}{4}\right)} \right) = \\
\begin{bmatrix}
0 & +1 & 0 \\
+1 & -4 & +1 \\
0 & +1 & 0 \\
\end{bmatrix} + \\
\begin{bmatrix}
0 & -4 & 0 \\
+1 & 0 & +1 \\
+1 & +1 & +1 \\
+1 & +1 & +1 \\
\end{bmatrix}
\]

The linearity of the DFT ensures that the transfer function of the isotropic Laplacian is the real-valued and symmetric sum of the “normal” and rotated Laplacians.

Generalized Laplacian

The isotropic Laplacian just considered may be written as the difference of a $3 \times 3$ average and a scaled discrete delta function:

\[
\begin{bmatrix}
+1 & +1 & +1 \\
+1 & -8 & +1 \\
+1 & +1 & +1 \\
\end{bmatrix} = \\
\begin{bmatrix}
+1 & +1 & +1 \\
+1 & +1 & +1 \\
+1 & +1 & +1 \\
\end{bmatrix} - 9 \cdot \\
\begin{bmatrix}
0 & 0 & 0 \\
0 & +1 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\]

which suggests that the Laplacian operator may be generalized to include any operator that computes the difference of a scaled original image and replicas that were blurred by some averaging kernel. For example, the impulse response of the averager may be the 2-D circularly symmetric continuous Gaussian impulse response:

\[
h [x, y] = A \exp \left[ -\pi \left( \frac{x^2 + y^2}{b^2} \right) \right]
\]

where the decay parameter $b$ determines the rate at which the values of the kernel decrease away from the center and the amplitude parameter $A$ often is selected to normalize the sum of the elements of the kernel to unity, thus ensuring that the process computes a weighted average. A normalized discrete approximation of the Gaussian kernel with $b = 2 \cdot \Delta x$ is:
The corresponding generalized Laplacian operator is this difference of this quantized Gaussian and the $5 \times 5$ identity kernel:

where $\alpha$ is the weighting of the identity image. Note that the sum of the elements of this generalized Laplacian kernel is zero because of the normalization of the Gaussian kernel, and so this operator applied to an input with uniform gray value will yield a null image.
16.2 CONVOLUTION

16.2.10 Discrete “Sharpening” Operators

1-D Case

A “sharpening” operator passes all sinusoidal components with no change in phase while amplifying those with large spatial frequencies. This action will tend to compensate for the effect of lowpass filtering. One example of a 1-D continuous sharpener is constructed from the second derivative; the amplitude of its transfer function is unity at the origin and rises as \( \xi^2 \) for larger spatial frequencies:

\[
H[\xi] = 1 + 4\pi^2\xi^2
\]

The corresponding continuous 1-D impulse response is the difference of the identity and second-derivative kernels

\[
h[x] = \delta[x] - \delta''[x]
\]

A discrete version of the impulse response may be generated by substituting the discrete Dirac delta function and the “centered” discrete second derivative operator:

\[
h[n] = \delta_d - \partial_x^2 = \begin{bmatrix} 0 & +1 & 0 & -1 & -2 & +1 \\ -1 & +3 & -1 & \end{bmatrix} = \delta_d[n] - RECT\left[\frac{n}{3}\right]
\]

The transfer function of the discrete sharpener is:

\[
H[k] = 4 \cdot 1[k] - \left(1 + 2 \cos \left[\frac{2\pi k}{N}\right]\right) = 3 - 2 \cos \left[\frac{2\pi k}{N}\right]
\]

The amplitudes of the transfer function at DC and at the Nyquist frequency are:

\[
H[k = 0] = +1
\]

\[
H\left[k = -\frac{N}{2}\right] = +5
\]

In words, the “second-derivative sharpener” amplifies the amplitude of the sinusoidal component that oscillates at the Nyquist frequency by a factor of 5.

The action of this sharpening operator on a “blurry” edge is shown in the figure. The slope of the edge is “steeper” after sharpening, but the edge also “overshoots” the correct amplitude at both sides.
CHAPTER 16 LOCAL OPERATIONS

Action of 1-D 2nd-derivative sharpening operator on a “blurry” edge. The angle of the slope of the sharpened edge is “steeper”, but the amplitude “overshoots” the correct value on both sides of the edge. The output is not the ideal sharp edge, so this operator only approximates the ideal inverse filter.

This interpretation may be extended to derive other 1-D sharpeners by computing the difference between a scaled replica of the “original” (blurred) input image and an image obtained by passing through a different lowpass filter.

2-D Sharpening Operators

We can generalize the 1-D discussion to produce a 2-D sharpener based on the Laplacian. The discrete version often is used to sharpen digital images that have been blurred by unknown lowpass filters. The process is ad hoc; it is not “tuned” to the details of the blurring process and so is not an “inverse” filter. It cannot generally reconstruct the original sharp image, but it “steepens” the slope of pixel-to-pixel changes in gray level, thus making the edges appear “sharper.” The impulse response of the 2-D continuous Laplacian sharpener is:

\[ f[x, y, 0] \cong f[x, y, z] - \alpha \cdot \nabla^2 f[x, y, z] \]

where \( \alpha \) is a real-valued free parameter that allows the sharpener to be “tuned” to the amount of blur. Obviously, the corresponding discrete solution is:

\[ g[n, m] = f[n, m] - \alpha \cdot \nabla^2_d f[n, m] \]

\[ = (\delta_d[n, m] - \alpha \cdot \nabla^2_d) \ast f[n, m] \]

where \( \nabla^2_d[n, m] \) is a Laplacian kernel that may be selected from the variants already considered. A single discrete sharpening kernel \( h[n, m] \) may be constructed from the
simplest form for the Laplacian:

\[
\begin{align*}
    h_1[n,m;\alpha] & = 0 \quad 0 \quad 0 \\
                      & -\alpha \cdot 0 +1 \quad -4 \quad 1 \\
                      & 0 \quad 0 \quad 0 \\
    & = 0 \quad -\alpha \quad 0 \\
                      & -\alpha \quad 1 - 4\alpha \quad -\alpha \\
                      & 0 \quad -\alpha \quad 0
\end{align*}
\]

The parameter \( \alpha \) may be increased to enhance the sharpening by steepening the slope of the edge profile and also increasing the “overshoot.” Selection of \( \alpha = +1 \) produces a commonly used sharpening kernel:

\[
\begin{align*}
    h_1[n,m;\alpha = +1] & = 0 \quad -1 \quad 0 \\
                      & -1 \quad +5 \quad -1 \\
                      & 0 \quad -1 \quad 0
\end{align*}
\]

The weights in the kernel sum to unity, which means that the average gray value of the image is preserved. In words, this process amplifies differences in gray level of adjacent pixels while preserving the mean gray value.

The corresponding discrete transfer function for the parameter \( \alpha \) is:

\[
H_1[k,\ell;\alpha] = (1 + 4\alpha) - 2\alpha \left( \cos \left[ \frac{2\pi k}{N} \right] + \cos \left[ \frac{2\pi \ell}{N} \right] \right)
\]

In the case \( \alpha = +1 \), the resulting transfer function is:

\[
H_1[k,\ell;\alpha = 1] = 5 - 2 \left( \cos \left[ \frac{2\pi k}{N} \right] + \cos \left[ \frac{2\pi \ell}{N} \right] \right)
\]

which has its maximum amplitude of \((H_1)_{\text{max}} = 9\) at the corners of the array.

A sharpening operator also may be derived from the isotropic Laplacian:

\[
\begin{align*}
    h_2[n,m;\alpha] & = -\alpha \quad -\alpha \quad -\alpha \\
                      & -\alpha \quad 1 + 8\alpha \quad -\alpha \\
                      & -\alpha \quad -\alpha \quad -\alpha
\end{align*}
\]

Again, the sum of the elements in the kernel is unity, ensuring that the average gray value of the image is preserved by the action of the sharpener. If the weighting factor is again selected to be unity, the kernel is the difference of a scaled original and a
$3 \times 3$ blurred copy:

$$h_2[n, m; 1] = \begin{bmatrix} -1 & -1 & -1 \\ -1 & +9 & -1 \\ -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & +10 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} +1 & +1 & +1 \\ +1 & +1 & +1 \\ +1 & +1 & +1 \end{bmatrix}$$

This type of process has been called *unsharp masking* by photographers. A sandwich of transparencies of the original image and a blurred negative produces a sharpened image of the original. This difference of the blurred image and the original is easily implemented in a digital system as a single convolution.

An example of 2-D sharpening is shown in the figure. Note the “overshoots” at the edges in the sharpened image. The factor of +9 ensures that the dynamic range of the sharpened image can be as large as from +9 to −8 times the maximum gray value, or $-2040 \leq \hat{f} \leq +2295$ for an 8-bit image. This would only happen for an isolated bright pixel at the maximum surrounded by a neighborhood of black pixels, and vice versa. In actual use, the range of values is considerably smaller. The image gray values either have to be biased up and rescaled or “clipped” at the maximum and minimum, as was done here.
Action of the 2-D sharpening operator based on the Laplacian. The original image \( f[n,m] \) has been blurred by a \( 3 \times 3 \) uniform averager to produce \( g[n,m] \). The action of the \( 3 \times 3 \) Laplacian sharpener on \( g[n,m] \) produced the bipolar image \( \hat{f}[n,m] \), which was clipped at the original dynamic range. The “overshoots” at the edges give the impression of a sharper image.

### 16.2.11 2-D Gradient

The *gradient* of a 2-D continuous function \( f[x,y] \) also was defined in the discussion of electromagnetism. It constructs a 2-D vector at each location in a scalar (gray-scale) image. The \( x \)-and \( y \)-components of the vector at each location are the \( x \)- and \( y \)-derivatives of the image:

\[
g[x,y] = \nabla f[x,y] = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]
\]

The image \( f[n,m] \) is a **scalar** function which assigns a numerical gray value \( f \) to each coordinate \( [n,m] \). The gray value \( f \) is analogous to terrain “elevation” in a
map. This process calculates a vector at each coordinate \([x, y]\) of the scalar image whose Cartesian components are \(\frac{\partial f}{\partial x}\) and \(\frac{\partial f}{\partial y}\). Note that the 2-D vector \(\nabla f\) may be represented in polar form as magnitude \(|\nabla f|\) and direction \(\Phi \{\nabla f\}\):

\[
|\nabla f [x, y]| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}
\]

\[
\Phi \{\nabla f [n, m]\} = \tan^{-1}\left[\frac{\left(\frac{\partial f}{\partial y}\right)}{\left(\frac{\partial f}{\partial x}\right)}\right]
\]

The 2-D vector at each location has the values:

\[
g [n, m] = \nabla f [n, m] = \begin{bmatrix} \partial_x * f [n, m] \\ \partial_y * f [n, m] \end{bmatrix}
\]

This vector points “uphill” in the direction of the maximum “slope” in gray level. The magnitude \(|\nabla f|\) is the “slope” of the 3-D surface \(f\) at pixel \([n, m]\). The azimuth angle (often called the phase by analogy with complex numbers) of the gradient \(\Phi \{\nabla f [n, m]\}\) is the compass direction toward which the slope points “uphill.”

The discrete version of the gradient magnitude also is a useful operator in digital image processing, as it will take on extreme values at edges between objects. The magnitude of the gradient often is approximated as the sum of the magnitudes of the components:

\[
|\nabla f [n, m]| = \sqrt{(\partial_x * f [n, m])^2 + (\partial_y * f [n, m])^2}
\approx |\partial_x * f [n, m]| + |\partial_y * f [n, m]|
\]

The gradient is not a linear operator, and thus can neither be evaluated as a convolution nor described by a transfer function. The largest values of the magnitude of the gradient correspond to the pixels where the gray value “jumps” by the largest amount, and thus the thresholded magnitude of the gradient may be used to identify such pixels. In this way the gradient may be used as an “edge detection operator.”
Example of the discrete gradient operator $\nabla f[n,m]$. The original object is the nonnegative function $f[n,m]$ shown in (a), which has amplitude in the interval $0 \leq f \leq +1$. The gradient at each pixel is the 2-D vector with components bipolar $[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$. The two component images are shown in (b) and (c). These also may be displayed as the magnitude $\sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$ and the angle $\phi = \tan^{-1}\left[\frac{\partial f}{\partial y}/\partial f/\partial x}\right]$. The extrema of the magnitude are located at corners and edges in $f[n,m]$.

### 16.2.12 Pattern Matching

It often is useful to design kernels to locate specific gray-level patterns, such as edges at particular orientations, corners, isolated pixels, particular shapes, or what have you. Particularly in the early days of digital image processing when computers were less capable than they are today, the computational intensity of the calculation often
was an important issue. It was desirable to find the least intensive method for common
tasks such as pattern detection, which generally meant that the task was performed
in the space domain using a small convolution kernel rather than calculating a better
approximation to the ideal result in the frequency domain. That said, the process of
designing and applying a pattern-matching kernel illuminates some of the concepts
and thus is worth some time and effort.

A common technique for pattern matching convolves the input image with a kernel
that is the same size as the “reference” pattern. The process and its limitations will
be illustrated by example. Consider an input image $f[n,m]$ that is composed of two
replicas of some real-valued nonnegative pattern of gray values, $p[n,m]$, centered at
coordinates $[n_1,m_1]$ and $[n_2,m_2]$ with respective amplitudes $A_1$ and $A_2$. The image
also includes a bias $b \cdot 1[n,m]$

$$f[n,m] = A_1 \cdot p[n-n_1,m-m_1] + A_2 \cdot p[n-n_2,m-m_2] + b \cdot 1[n,m]$$

The appropriate kernel of the discrete filter is:

$$\hat{m}[n,m] = p[-n,-m]$$

which also is real valued and nonnegative within its region of support. The output
from this matched filter autocorrelation of the pattern centered at those coordinates:

$$g[n,m] = f[n,m] \ast \hat{m}[n,m]$$

$$= A_1 \cdot p[n,m] \star p[n,m]|_{n=n_1,m=m_1} + A_2 \cdot p[n,m] \star p[n,m]|_{n=n_2,m=m_2}$$

$$+ b \cdot (1[n,m] \ast p[-n,-m])$$

$$= A_1 \cdot p[n,m] \star p[n,m]|_{n=n_1,m=m_1} + A_2 \cdot p[n,m] \star p[n,m]|_{n=n_2,m=m_2}$$

$$+ b \cdot \sum_{n,m} p[n,m]$$

The last term is the constant output level from the convolution of the bias with the
matched filter, which produces the sum of the product of the bias and the weights at
each sample. The spatially varying autocorrelation functions rest on a bias propor-
tional to the sum of the gray values $p$ in the pattern. If the output bias is large, it
can reduce the “visibility” of small (but significant) variations in the autocorrelation
in exactly the same way as small modulations of a nonnegative sinusoidal function
with a large bias are difficult to see. It is therefore convenient to construct a matched
filter kernel whose weights sum to zero. It only requires subtraction of the average
value from each sample of the kernel:

$$\hat{m}[n,m] = p[-n,-m] - p_{\text{average}}$$

$$\Rightarrow \sum_{n,m} \hat{m}[-n,-m] = \sum_{n,m} \hat{m}[n,m] = 0$$

Thus ensuring that the constant bias vanishes. This result determines the strategy
for designing convolution kernels that produce outputs that have large magnitudes at
pixels centered on neighborhoods that contain these patterns and small magnitudes
in neighborhoods where the feature does not exist. For example, consider an image
containing an “upper-right corner” of a brighter object on a darker background:

\[
f[n, m] = \begin{pmatrix}
\ldots & \vdots & \vdots & \vdots & \vdots & \ldots \\
\ldots & 50 & 50 & 50 & 50 & \ldots \\
\ldots & 50 & 50 & 50 & 50 & \ldots \\
\ldots & 100 & 100 & 100 & 50 & \ldots \\
\ldots & 100 & 100 & 100 & 50 & \ldots \\
\ldots & \vdots & \vdots & \vdots & \vdots & \ldots 
\end{pmatrix}
\]

The task is to design a 3 × 3 kernel for “locating” this pattern:

\[
p[n, m] = \begin{pmatrix}
50 & 50 & 50 \\
100 & 100 & 50 \\
100 & 100 & 50 
\end{pmatrix}
\]

In other words, we want to construct an operator that produces a large output value
when it is centered over this “upper-right corner” pattern. The recipe for convolution
tells us to rotate the pattern by \( \pi \) radians about its center to create \( p[-n, -m] \):

\[
p[-n, -m] = \begin{pmatrix}
50 & 100 & 100 \\
50 & 100 & 100 \\
50 & 50 & 50 
\end{pmatrix}
\]

The average weight in this 3 × 3 kernel is \( \frac{650}{9} \approx 72.222 \), which is subtracted from each element:

\[
\begin{pmatrix}
-200/9 & +250/9 & +250/9 \\
-200/9 & +250/9 & +250/9 \\
200/9 & -200/9 & -200/9 
\end{pmatrix} \approx (+22.222)
\]

\[
\begin{pmatrix}
-1 & +1.25 & +1.25 \\
-1 & +1.25 & +1.25 \\
-1 & -1 & -1 
\end{pmatrix}
\]

The multiplicative factor may be ignored since it just scales the output of the convo-
lution by this constant. Thus one realization of the unamplified 3 × 3 matched filter
for upper-right corners is:
Though it is not really an issue any longer (given the advanced state of computing technology), it was once more convenient to restrict the weights in the kernel to integer values so that all calculations were performed by integer arithmetic. This may be achieved by redistributing the weights slightly. In this example, the fraction of the positive weights often is concentrated in the center pixel to produce the Prewitt corner detector:

Note that that the upper-right corner detector contains a bipolar pattern that looks like a lower-left corner because of the rotation (“reversal”) inherent in the convolution. Because \( \hat{m} \) is bipolar, so generally is the output of the convolution with the input \( f[n,m] \). The linearity of convolution ensures that the output amplitude at a pixel is proportional to the contrast of the feature.

If the contrast of the upper-right corner is large and “positive,” meaning that the corner is much brighter than the dark background, the output at the corner pixel will be a large and positive extremum. Conversely, a dark object on a very bright background will produce a large negative extremum. The magnitude of the image shows the locations of features with either contrast. The output image may be thresholded to specify the pixels located at the desired feature.

This method of feature detection is not ideal. The output of this unamplified filter at a corner is the autocorrelation of the feature rather than the ideal 2-D discrete Dirac delta function. If multiple copies of the pattern with different contrasts are present in the input, it will be difficult or impossible to segment the desired features by thresholding the convolution alone. Another consequence of the unamplified matched filter is that features other than the desired pattern produce nonnull outputs, as shown in the output of the corner detector applied to a test object consisting of “E” at two different amplitudes as shown in the figure. The threshold properly locates the upper-right corners of the bright “E” and one point on the sampled circle, but misses the corners of the fainter “E”. This shows that corners of some objects are missed (false negatives). If the threshold were set at a lower level to detect the corner of the fainter “E”, other pixels will be incorrectly identified as corners (false positives). A simple method for reducing misidentified pixels is considered in the next section.
Threshooing to locate features in the image: (a) $f[n,m]$, which is the nonnegative function with $0 \leq f \leq 1$; (b) $f[n,m]$ convolved with the “upper-right corner detector”, producing the bipolar output $g[n,m]$ where $-5 \leq g \leq 4$. The largest amplitudes occur at the upper-right corners, as shown in the image thresholded at level 4, shown in (c) along with the “ghost” of the original image. This demonstrates that the upper-right corners of the high-contrast “E” and of the circle were detected, but corner of the low-contrast “E” was missed.
16.2.13 Normalization of Contrast of Detected Features

The recipe just developed allows creation of kernels for detecting pixels in neighborhoods that are “similar” to some desired pattern. However, the sensitivity of the process to feature contrast that was also demonstrated can significantly limit its usefulness. A simple modification to “normalize” the correlation measure can improve the classification significantly. Ernest Hall defined the normalized correlation measure $R[n, m]$:

$$R[n, m] = \frac{f[n, m] * h[n, m]}{\sqrt{\sum_{n,m} (f[n, m])^2 \sum_{n,m} (h[n, m])^2}}$$

where the sums in the denominator are over ONLY the elements of the kernel. The sum of the squares of the elements of the kernel $h[n, m]$ results in a constant scale factor $k$ and may be ignored, thus producing the formula:

$$R[n, m] = k \left( \frac{f[n, m]}{\sum_{n,m} (f[n, m])^2} \right) * h[n, m]$$

where again the summation is ONLY over the size of the kernel. In words, this operation divides the convolution by the geometric sum of gray levels under the kernel and by the geometrical sum of the elements of the kernel. The modification to the filter makes the entire process shift variant and thus may not be performed by a simple convolution. The denominator may be computed by convolving $(f[n, m])^2$ with a uniform averaging kernel of the same size as the original kernel $h[n, m]$ and then evaluating the square root

$$R[n, m] = k \frac{f[n, m] * h[n, m]}{\sqrt{(f[n, m])^2 * s[n, m]}}$$

The upper-right corner detector with normalization is shown in the figure, where the features of both “E”s are located with a single threshold.
16.3 NONLINEAR FILTERS

The action of the nonlinear normalization of detected features using the same object: (a) \( f[n,m] \) convolved with the upper-right corner detector with normalization by the image amplitude, producing the bipolar output \( g[n,m] \) where \(-0.60 \leq g \leq +0.75\); (b) image after thresholding at \( g = 0.6 \), with a “ghost” of the original image, showing the detection of the upper-right corners of both “E”s despite the different image contrasts.

16.3 Nonlinear Filters

Any filtering operation that cannot be evaluated as a convolution must be either nonlinear or space variant, or both.

16.3.1 Median Filter

Probably the most useful nonlinear statistical filter is the local median, i.e., the gray value of the output pixel is the median of the gray values in a neighborhood, which is obtained by sorting the gray values in numerical order and selecting the middle value. To illustrate, consider the \( 3 \times 3 \) neighborhood centered on the value “3” and the 9 values sorted in numerical order; the median value of “2” is indicated by the
box and replaces the “3” in the center of the window:

\[
\begin{array}{ccc}
1 & 2 & 6 \\
3 & 2 & 5 \\
1 & 5 & 2
\end{array}
\]

\[\Rightarrow\text{ ordered sequence is } 1 \ 1 \ 2 \ 2 \ 3 \ 5 \ 5 \ 6\]

The nonlinear nature of the median can be recognized by noting that the median of the sum of two images is generally not equal to the sum of the medians. For example, the median of a second \(3 \times 3\) neighborhood is “3”

\[
\begin{array}{ccc}
4 & 5 & 6 \\
3 & 1 & 2 \\
2 & 4 & 3
\end{array}
\]

\[\Rightarrow 1 \ 2 \ 2 \ 3 \ 3 \ 4 \ 4 \ 5 \ 6\]

The sum of the two medians is \(2 + 3 = 5\), but the sum of the two \(3 \times 3\) neighborhoods produces a third neighborhood whose median of “6”:

\[
\begin{array}{ccc}
5 & 7 & 12 \\
6 & 3 & 7 \\
3 & 9 & 5
\end{array}
\]

\[\Rightarrow 3 \ 3 \ 5 \ 5 \ 6 \ 7 \ 7 \ 9 \ 12\]

confirming that the median of the sum is not the sum of the medians and that the median is a nonlinear operation.

The median requires sorting and thus may not be computed as a convolution. Its computation typically requires more time than a mean filter, but it has the advantage of reducing the modulation of signals that vary or oscillate over a period less than the width of the window while preserving the gray values of signals which are constant or monotonically varying on a scale larger than the window size. This implies that the variance of additive noise will be reduced by the median in a fashion similar to the mean filter, while preserving sharp transitions in gray value. Also note that, unlike the mean filter, all gray values generated by the median exist in the original image, thus obviating the need for requantization.

The statistics of the median-filtered image depend on the probability density function of the input signal, including the deterministic part and any noise. Thus predictions of the effect of the filter cannot be as specific as for the mean filter, i.e., given an input image with known statistics (mean, variance, etc.), the statistics of the output image are more difficult to predict. However, Frieden [7, Probability, Statistical Optics, and Data Testing, Springer-Verlag, 1983, pp. 254-258] has analyzed the statistical properties of the median filter by modeling it as a limit of a large number of discrete trials of a binomial probability distribution (Bernouilli trials). The median of \(N\) samples (odd number) for a set of gray values \(f_i\) taken from an input distribu-
tion with probability law (i.e. histogram) \( p_f[x] \) must be determined. Frieden applied the principles of Bernoulli trials to determine the probability density of the median of several independent sets of numbers. In other words, he sought to determine the probability that the median of the \( N \) numbers \( \{f_n\} \) is \( x \) by evaluating the median of many independent such sets of \( N \) numbers selected from a known probability distribution \( p_f[x] \). Frieden reasoned that, for each placement of the median window, a specific amplitude \( f_n \) of the \( N \) values is the median if three conditions are satisfied:

1. one of the \( N \) numbers satisfies the condition \( x \leq f_n < x + \Delta x \)
2. of the remaining \( N - 1 \) numbers, \( \frac{N-1}{2} \) exceed \( x \), and
3. \( \frac{N-1}{2} \) of the remaining numbers are less than \( x \).

The probability of the simultaneous occurrence of these three events is the probability density of the output of the median window. For an arbitrary \( x \), any one value \( f_n \) must either lie in the interval \( (x \leq f < x + \Delta x) \), be larger than \( x \), or less than \( x \). In other words, each trial has three possible outcomes. These conditions define a sequence of Bernoulli trials with three outcomes, which is akin to the task of flipping a “three-sided” coin where the probabilities of the three outcomes are not equal. In the more familiar case, the probability that \( N \) coin flips with two possible outcomes that have associated probability \( p \) and \( q \) will produce \( m \) “successes” (say, \( m \) heads) is:

\[
P_N[m] = \frac{N!}{(N-m)!m!} p^m (1-p)^{N-m}
\]

The formula is easy to extend to the more general case of three possible outcomes; the probability that the result yields \( m_1 \) instances of the first possible outcome (say, “head #1), \( m_2 \) of the second outcome (“head #2”) and \( m_3 = N - (m_1 + m_2) \) of the third (“tails”) is

\[
P_N[m_1, m_2, m_3] = \frac{N!}{m_1!m_2!(N - (m_1 + m_2))!} p_1^{m_1} p_2^{m_2} p_3^{m_3}
\]

where \( p_1, p_2, \) and \( p_3 = 1 - (p_1 + p_2) \) are the respective probabilities of the three outcomes.

When applied to one sample of data, the median filter has three possible outcomes whose probabilities are known:

1. the sample amplitude may be the median (probability \( p_1 \)),
2. the sample amplitude may be smaller than the median (probability \( p_2 \)), and
3. it may be larger than the median (probability \( p_3 \)).
\[ p_1 = P[x \leq f_n \leq x + \Delta x] = pf[x] \]
\[ p_2 = P[f_n < x] = Cf[x] \]
\[ p_3 = P[f_n > x] = 1 - Cf[x] \]

where \( Cf[x] \) is the cumulative probability distribution of the continuous probability density function \( pf[x] \):
\[
Cf[x] = \int_{-\infty}^{x} pf[\alpha] \ d\alpha
\]

In this case, the distributions are continuous (rather than discrete), so the probability is the product of the probability density function \( pf[x] \) and the infinitesimal element \( dx \). We substitute the known probabilities and the known number of occurrences of each into the Bernoulli formula for three outcomes:
\[
p_{med}[x] \ dx = \frac{N!}{((N-1)/2)! \cdot (N-1)! \cdot 1!} (Cf[x])^{\frac{N-1}{2}} \cdot (1 - Cf[x])^{\frac{N-1}{2}} \cdot pf[x] \ dx
\]

If the window includes \( N = 3, 5, \) or 9 values, the following probability laws for the median result:

\[ N = 3 \implies p_{med}[x] \ dx = \frac{3!}{(1!)} (Cf[x]) \cdot (1 - Cf[x]) \cdot pf[x] \ dx \]

\[ = 6(Cf[x]) \cdot (1 - Cf[x]) \cdot pf[x] \ dx \]

\[ N = 5 \implies p_{med}[x] \ dx = \frac{5!}{(2!)} (Cf[x])^2 \cdot (1 - Cf[x])^2 \cdot pf[x] \ dx \]

\[ = 30(Cf[x])^2 \cdot (1 - Cf[x])^2 \cdot pf[x] \ dx \]

\[ N = 9 \implies p_{med}[x] \ dx = 630(Cf[x])^4 \cdot (1 - Cf[x])^4 \cdot pf[x] \ dx \]

**Example: Median Filter Applied to Uniform Distribution**

The statistical properties of the median will now be demonstrated for some simple examples of known probabilities. If the original pdf \( pf[x] \) is uniform over the interval \([0, 1]\), then it may be written as a rectangle function:
\[
p_f[x] = RECT \left[ x - \frac{1}{2} \right]
\]
pdf of noise that is uniformly distributed over the interval $[0, 1]$ and its associated cumulative probability distribution $F_c[x] = x \cdot \text{RECT} \left[ x - \frac{1}{2} \right] + \text{STEP} [x - 1]$

The associated cumulative probability distribution may be written in several ways, including:

$$C_f[x] = \int_{-\infty}^{x} p_f(\alpha) d\alpha$$

$$= x \cdot \text{RECT} \left[ x - \frac{1}{2} \right] + \text{STEP} [x - 1]$$

so the product of the cumulative distribution and its complement is windowed by the rectangle to yield:

$$p_{\text{med}}[x] \ dx = \frac{N!}{((N-1)!)^2} \left( x^{\frac{N-1}{2}} \cdot (1 - x)^{\frac{N+1}{2}} \right) \ \text{RECT} \left[ x + \frac{1}{2} \right] \ dx$$

The pdfs of the output of median filters for $N = 3, 5,$ and $9$ are:

$N = 3 \implies p_{\text{median}}[x] \ dx = 6 (x - x^2) \ \text{RECT} \left[ x - \frac{1}{2} \right] \ dx$

$N = 5 \implies p_{\text{median}}[x] \ dx = 30 \left( x^4 - 2x^3 + x^2 \right) \ \text{RECT} \left[ x - \frac{1}{2} \right] \ dx$

$N = 9 \implies p_{\text{median}}[x] \ dx = 630 \cdot (x^8 - 4x^7 + 6x^6 - 4x^5 + x^4) \ \text{RECT} \left[ x - \frac{1}{2} \right] \ dx$

are compared to the pdfs of the output of the mean filters in the figure:
Comparison of pdfs of mean and median filter for uniform probability density function \( p_f(x) = \text{RECT}(x + \frac{1}{2}) \) for \( N = 3, 5, \) and 9. Note that the pdf of the mean filter is “taller” and “skinnier” in all three cases, showing that it will reduce the variance more than the median filter.

Just like the mean filter, the maximum value of \( p_{\text{median}}(x) \) increases and its “width” decreases as the number of input values in the median window increases (as \( N \uparrow \)). The calculated pdfs for the median and mean filters over \( N = 3 \) and \( N = 5 \) samples for input values from a uniform probability distribution are shown below to the same scale. Note that the output distributions from the mean filter are taller than for the median, which indicates that the median filter does a poorer job over averaging noise than the mean (Frieden determined that the SNR of the median filter is smaller than that of the mean by a factor of \( \log_2[2] \approx 0.69 \), so that there is a penalty in SNR of about 30% for the median filter relative to the averaging filter. Put another way, the standard deviation of the median of \( N \) samples decreases as \( \left( \sqrt{N} \cdot \log_2[2] \right)^{-1} \approx \frac{0.69}{\sqrt{N}} \) instead of as \( \frac{1}{\sqrt{N}} \). The lesser noise reduction of the median filter is offset by its ability to preserve the sharpness of edges.
Comparison of mean and median filter: (a) bitonal object $f[m]$ defined over 1000 samples; (b) mean of $f[m]$ over 25 samples, showing reduction in contrast with increasing frequency; (c) median of $f[m]$ over 25 samples, which is identical to $f[m]$; (d) $f[m] + n[m]$, which is uniformly distributed over interval $[0, 1]$; (e) mean over 25 samples; (f) median over 25 samples. Note that the highest-frequency bars are better preserved by the median filter.

Example: Median Filter Applied to Gaussian Noise

Probably the most important application of the median filter is to attenuate Gaussian noise (i.e., the gray values are selected from a normal distribution with zero mean) without blurring edges. The central limit theorem indicates that the statistical character of noise which has been generated by summing random variables from different distributions will be Gaussian in character. The probability distribution function is the Gaussian with mean value $\mu$ and variance $\sigma^2$ normalized to unit area:

$$p_f[x] = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[ -\frac{(x - \mu)^2}{2\sigma^2} \right]$$
The cumulative probability density of this noise is the integral of the Gaussian probability law, which is proportional to the error function:

\[
\text{erf} [x] \equiv \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt
\]

We can evaluate the cumulative density in terms of \( \text{erf} [x] \):

\[
C_f [x] = \int_{-\infty}^x p_c [x] \, dx = \frac{1}{2} - \frac{1}{2\sqrt{2\sigma}} \text{erf} \left[ \frac{x - \mu}{\sqrt{2\sigma}} \right] \text{ for } x \leq \mu
\]

\[
C_f [x] = \int_{-\infty}^x p_c [x] \, dx = \frac{1}{2} + \frac{1}{2\sqrt{2\sigma}} \text{erf} \left[ \frac{x - \mu}{\sqrt{2\sigma}} \right] \text{ for } x \geq \mu
\]

Therefore the probabilities of the different outcomes of the median filter are:

\[
p_{\text{med}} [x] \, dx = \frac{N!}{((N-1)!)^2} \left( \frac{1}{2} + \frac{1}{2\sqrt{2\sigma}} \text{erf} \left[ \frac{x - \mu}{\sqrt{2\sigma}} \right] \right)^{\frac{N-1}{2}} \cdot \left( 1 - \frac{1}{2\sqrt{2\sigma}} \text{erf} \left[ \frac{x - \mu}{\sqrt{2\sigma}} \right] \right)^{\frac{N-1}{2}} \cdot \frac{1}{\sqrt{2\pi\sigma}} \exp \left[ -\frac{x^2}{2\sigma^2} \right] \, dx
\]

The error function is compiled and may be evaluated to plot the probability \( p_{\text{med}} [x] \)

\[\text{pdf of Gaussian noise with } \mu = 1, \sigma = 2 \text{ (black) and of the median for } N = 3 \text{ (red), } \quad N = 9 \text{ (blue).}\]

The graphs illustrate the theoretical averaging effects of the mean and median filters on Gaussian noise. The graphs are plotted on the same scale and show the pdf of the original Gaussian noise (on the left) and the output resulting from mean and median.
filtering over 3 pixels (center) and after mean and median filtering over 5 pixels (right). The calculated mean gray value and standard deviation for 2048 samples of filtered Gaussian noise yielded the following values:

\[
\begin{align*}
\mu_{in} &= 0.211 \\
\sigma_{in} &= 4.011 \\
\mu_3 - \text{mean} &= 0.211 \\
\sigma_3 - \text{mean} &= 2.355 \\
\mu_3 - \text{median} &= 0.225 \\
\sigma_3 - \text{median} &= 2.745
\end{align*}
\]

**Effect of Window “Shape” on Median Filter**

In the 2-D imaging case, the shape of the window over which the median is computed also affects the output image. For example, if the 2-D median is computed over a \(5 \times 5\) window at the upper-right corner of a dark object on a bright background, the median will be the background value:

\[
\begin{array}{cccccc}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
+1 & +1 & +1 & 0 & 0 \\
+1 & +1 & +1 & 0 & 0 \\
+1 & +1 & +1 & 0 & 0
\end{array}
\]

The median calculated over a full square window (\(3 \times 3\), etc.) will convert bright pixels at outside corners of a bright object to dark pixels, i.e., the corners will be clipped; it will also convert a dark background pixel at the inside corner of a bright object to a bright pixel. It will also eliminate lines less than half as wide as the window. Corner clipping may be prevented by computing the median over a window that only includes 9 values arrayed along horizontal and vertical lines:

\[
\begin{array}{cccc}
- & - & +1 & - \\
- & - & +1 & - \\
+1 & +1 & +1 & +1 \\
- & - & +1 & - \\
- & - & +1 & - \\
\end{array}
\]
If applied to the pixel in the corner, we obtain
\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
+1 & +1 & +1 & 0 \\
+1 & +1 & +1 & 0 \\
+1 & +1 & +1 & 0 \\
\end{array} \times \begin{array}{cccc}
- & - & +1 & - \\
- & - & +1 & - \\
+1 & +1 & +1 & +1 \\
- & - & +1 & - \\
- & - & +1 & - \\
\end{array} = \begin{array}{cccc}
- & - & 0 & - \\
- & - & 0 & - \\
+1 & +1 & +1 & 0 \\
- & - & +1 & - \\
- & - & +1 & - \\
\end{array} = +1
\]

This pattern also is effective when applied to thin lines without eliminating them:
\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
+1 & +1 & +1 & +1 \\
+1 & +1 & +1 & +1 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{array} \times \begin{array}{cccc}
- & - & +1 & - \\
- & - & +1 & - \\
+1 & +1 & +1 & +1 \\
- & - & +1 & - \\
- & - & +1 & - \\
\end{array} = \begin{array}{cccc}
- & - & 0 & - \\
- & - & 0 & - \\
+1 & +1 & +1 & +1 \\
- & - & 0 & - \\
- & - & 0 & - \\
\end{array} = +1
\]

Other patterns of medians are also useful [?, Castleman, Digital Image Processing, Prentice-Hall, 1996, p. 249].

### 16.3.2 Other Statistical Filters (Mode, Variance, Maximum, Minimum)

The statistical mode in the window (i.e., the most common gray level) is a useful operator on binary images corrupted by isolated noise pixels (“salt-and-pepper noise”).
The mode is found by computing a mini-histogram of pixels within the window and assigning the most common gray level to the center pixel. Rules must be defined if two or more gray levels are equally common, and particularly if all levels are populated by a single pixel. If two levels are equally populated, the gray level of center pixel is usually retained if it is one of those levels; otherwise one of the most common gray levels may be selected at random.

The variance filter \( \sigma^2 \) and standard deviation filter \( \sigma \) replace the center pixel with the variance or standard deviation of the pixels in the window, respectively. The variance filtering operation is

\[
g(x, y) = \sum_{\text{window}} (f(x, y) - \mu)^2
\]

where \( \mu \) is the mean value of pixels in the window. The output of a variance or standard deviation operation will be larger in areas where the image is busy and small where the image is smooth. The output of the \( \sigma \)-filter resembles that of the isotropic Laplacian, which computes the difference of the center pixel and the average of the eight nearest neighbors.

The Maximum or Minimum filter obviously replace the gray value in the center with the highest or lowest value in the window. The MAX filter will dilate bright objects, while the MIN filter erodes them. These provide the basis for the so-called morphological operators. A “dilation” (MAX) followed by an “erosion” (MIN) defines the morphological “CLOSE” operation, while the opposite (erosion followed by dilation) is an “OPEN” operation. The “CLOSE” operation fills gaps in lines and removes isolated dark pixels, while OPENING removes thin lines and isolated bright pixels. These nonlinear operations are useful for object size classification and distance measurements.

### 16.4 Adaptive Operators

In applications such as edge enhancement or segmentation, it is often useful to “change”, or “adapt” the operator based on conditions in the image. One example has already been considered: the nonlinear normalization used while convolving with a bipolar convolution kernel. For another example, it is possible to enhance differences in the direction of the local gradient (e.g. via a 1-D Laplacian) while averaging in the orthogonal direction. In other words, the operator used to enhance the edge information is determined by the output of the gradient operator. As another example, the size of an averaging neighborhood could be varied based on the statistics (e.g., the variance) of gray levels in the neighborhood.

In some sense, these adaptive operators resemble cascaded convolutions, but the resulting operation is not space invariant and may not be described by convolution with a single kernel. By judicious choice of algorithm, significant improvement of image quality may be obtained.
16.5 Convolution Revisited – Bandpass Filters

The parameters of a filter that determine its effect on the image are the size of the kernel and the algebraic sign of its coefficients. Kernels whose elements have the same algebraic sign are lowpass filters that compute spatial averages and attenuate the modulation of spatial structure in the image. The larger the kernel, the greater the attenuation. On the other hand, kernels that compute differences of neighboring gray levels will enhance the modulation of spatially varying structure while attenuating the brightness of constant areas. Note that the largest number of elements in a kernel with different algebraic signs is two; the spatial first derivative is an example.

We will now construct a hybrid of these two extreme cases that will attenuate the modulation of image structure that varies more slowly or rapidly than some selectable rate. In other words, the filter will pass a band of spatial frequencies and attenuate the rest of the spectrum; this is a bandpass filter. The bandpass filter will compute differences of spatial averages of gray level. For example, consider a 1-D image:

\[ f[n] = 1 + \sum_{i=0}^{2} \cos \left[ \frac{2\pi n(2^i)}{128} \right] = 1 + \cos \left[ \frac{2\pi n}{\infty} \right] + \cos \left[ \frac{2\pi n}{64} \right] + \cos \left[ \frac{2\pi n}{32} \right] \]

\[ = 2 + \cos \left[ \frac{2\pi n}{128} \right] + \cos \left[ \frac{2\pi n}{64} \right] + \cos \left[ \frac{2\pi n}{32} \right] \]

The spatial frequencies of the cosines are:

- \( \xi_0 = \frac{1}{\infty} = 0 \) cycles per sample \( \implies X_0 = \infty \)
- \( \xi_1 = \frac{1}{128} \approx 7.8 \cdot 10^{-2} \) cycles per sample \( \implies X_1 = 128 \) samples
- \( \xi_2 = \frac{1}{64} \) cycles per sample \( \implies X_2 = 64 \) samples
- \( \xi_3 = \frac{1}{32} \) cycles per sample \( \implies X_3 = 32 \) samples

This function is periodic over 128 samples, which is a common multiple of all of the finite-period cosines. The extreme amplitudes are +5 and +0.2466. Consider convolution of \( f[n] \) with several kernels; the first set are 3-pixel averagers whose weights sum to unity, therefore preserving the mean gray level of \( f[n] \):

\[ h_1[n] = \begin{bmatrix} 0 & +1 & 0 \end{bmatrix} \]
\[ h_2[n] = \begin{bmatrix} +\frac{1}{4} & +\frac{1}{2} & +\frac{1}{4} \end{bmatrix} \]
\[ h_3[n] = \begin{bmatrix} +\frac{1}{3} & +\frac{1}{3} & +\frac{1}{3} \end{bmatrix} \]

Obviously, \( h_1[n] \) is the identity kernel, \( h_2 \) is a tapered averager that applies more weight to the center pixel, while \( h_3 \) is a uniform averager. Based on our experience with averaging filters, we know that \( g_1[n] = f[n] * h_1[n] \) must be identical to \( f[n] \), while the modulation of the output from \( h_2[n] \) will be reduced a bit in \( g_2 \) and somewhat
more in $g_3$. This expectation is confirmed by the computed maximum and minimum values:

<table>
<thead>
<tr>
<th></th>
<th>$f_{\text{max}} = 5$</th>
<th>$(g_1)_{\text{max}} = 5$</th>
<th>$(g_2)_{\text{max}} \approx 4.987$</th>
<th>$(g_3)_{\text{max}} \approx 4.983$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{\text{min}} \approx 0.2466$</td>
<td>$(g_1)_{\text{min}} \approx 0.2466$</td>
<td>$(g_2)_{\text{min}} \approx 0.2564$</td>
<td>$(g_3)_{\text{min}} \approx 0.2596$</td>
<td></td>
</tr>
</tbody>
</table>

The mean gray values of these images are identical:

$$ \langle f \rangle = \langle g_1 \rangle = \langle g_2 \rangle = \langle g_3 \rangle = 2 $$

We can define a contrast factor based on these maximum and minimum values that is analogous to the modulation, except that the image is not sinusoidal:

$$ c_f \equiv \frac{f_{\text{max}} - f_{\text{min}}}{f_{\text{max}} + f_{\text{min}}} $$

The corresponding factors are:

<table>
<thead>
<tr>
<th>$c_f$</th>
<th>$c_4$</th>
<th>$c_5$</th>
<th>$c_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.906</td>
<td>0.906</td>
<td>0.9022</td>
<td>0.9019</td>
</tr>
</tbody>
</table>

which confirms the expectation.

Now consider three 5-pixel avergers:

$$ h_4[n] = \begin{bmatrix} 0 & 0 & +1 & 0 & 0 \end{bmatrix} $$

$$ h_5[n] = \begin{bmatrix} \frac{1}{9} & \frac{2}{9} & \frac{3}{9} & \frac{2}{9} & \frac{1}{9} \end{bmatrix} $$

$$ h_6[n] = \begin{bmatrix} \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \end{bmatrix} $$

Again, $h_4$ is the identity kernel that reproduces the modulation of the original image, $h_2$ is a tapered averager, and $h_6$ is a uniform averager. The computed maximum and minimum values for the images are:

<table>
<thead>
<tr>
<th></th>
<th>$f_{\text{max}} = 5$</th>
<th>$(g_4)_{\text{max}} = 5$</th>
<th>$(g_5)_{\text{max}} \approx 4.967$</th>
<th>$(g_6)_{\text{max}} \approx 4.950$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{\text{min}} \approx 0.2466$</td>
<td>$(g_4)_{\text{min}} \approx 0.2466$</td>
<td>$(g_5)_{\text{min}} \approx +0.272$</td>
<td>$(g_6)_{\text{min}} \approx 0.285$</td>
<td></td>
</tr>
<tr>
<td>$\langle f \rangle = 2$</td>
<td>$\langle g_4 \rangle = 2$</td>
<td>$\langle g_5 \rangle = 2$</td>
<td>$\langle g_6 \rangle = 2$</td>
<td></td>
</tr>
<tr>
<td>$c_f = 0.906$</td>
<td>$c_4 = 0.906$</td>
<td>$c_5 = 0.896$</td>
<td>$c_6 = 0.891$</td>
<td></td>
</tr>
</tbody>
</table>

The average over a larger number of samples reduces the modulation further but does not affect the mean gray values.
Outputs from application of various averaging kernels to $f[x]$. At this scale, the differences are not noticeable.

Now consider a kernel that is significantly wider:

$$h_7[n] = A_7 \cos \left[ \frac{2\pi n}{64} \right] \cdot \text{RECT} \left[ \frac{n}{32} \right]$$

where the scale factor $A_7$ is usually used to normalize $h_7[n]$ to unit area. The $\text{RECT}$ function limits the support of the cosine to a finite width of 32 pixels, which is half the period of the cosine function. The kernel is wide and nonnegative; this is another example of a tapered averaging kernel that weights nearby pixels more heavily than more distant pixels. The corresponding uniform averaging kernel is:

$$h_8[n] = \frac{1}{32} \text{RECT} \left[ \frac{n}{32} \right]$$

To simplify comparison of the results of $h_7$ and $h_8$, we will set $A_7 = \frac{1}{32}$ instead of a factor that ensures a unit area. The exact value of the scale factor will affect the output amplitudes and not the modulation. Based on the experience gained for $h_1, h_2, \ldots, h_6$, we expect that both $h_7$ and $h_8$ will diminish the modulation of spatially varying patterns, and that $h_8$ will have the larger effect. In fact, because the width of $h_8$ matches the period $X_3 = 32$, this cosine term will be attenuated to null amplitude. The effects on the amplitudes of the array are:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{\text{max}}$</td>
<td>5</td>
</tr>
<tr>
<td>$(g_7)_{\text{max}}$</td>
<td>$\approx 2.585$</td>
</tr>
<tr>
<td>$(g_8)_{\text{max}}$</td>
<td>$\approx 3.536$</td>
</tr>
<tr>
<td>$f_{\text{min}}$</td>
<td>$0.2466$</td>
</tr>
<tr>
<td>$(g_7)_{\text{min}}$</td>
<td>$\approx 0.593$</td>
</tr>
<tr>
<td>$(g_8)_{\text{min}}$</td>
<td>$\approx +1.205$</td>
</tr>
<tr>
<td>$\langle f \rangle$</td>
<td>2</td>
</tr>
<tr>
<td>$\langle g_7 \rangle$</td>
<td>2</td>
</tr>
<tr>
<td>$\langle g_8 \rangle$</td>
<td>2</td>
</tr>
<tr>
<td>$c_f$</td>
<td>0.906</td>
</tr>
<tr>
<td>$c_7$</td>
<td>$\approx 0.627$</td>
</tr>
<tr>
<td>$c_8$</td>
<td>$\approx 0.492$</td>
</tr>
</tbody>
</table>

which again confirms the expectation that the uniform averager reduces the contrast to a greater degree than the tapered averager, but neither affects the mean gray value.
16.5 CONVOLUTION REVISITED — BANDPASS FILTERS

If the width of the $RECT$ function in the tapered averaging kernel $h_7$ is increased still further while the period of the constituent cosine function is retained, the resulting kernel includes some negative weights. For example:

$$h_9[n] = A_9 \cos \left[ 2\pi \frac{n}{64} \right] RECT \left[ \frac{n}{48} \right]$$

The constant $A_9$ may be chosen so that the area of $h_9$ is unity ($A_9 \simeq 0.06948$), or $A_9$ may be set to $\frac{1}{48}$, matching the normalization factor for the uniform averager:

$$h_{10}[n] = \frac{1}{48} RECT \left[ \frac{n}{48} \right]$$

which simplifies comparison of the resulting amplitudes. Kernel $h_9$ computes the same weighted average as $h_7$ in the neighborhood of the pixel, but then subtracts a weighted average of distant pixels from it; it computes differences of average amplitudes. The effects of these operations on the extrema are:

<table>
<thead>
<tr>
<th>$f_{\text{max}} = 5$</th>
<th>$(g_9)_{\text{max}} \simeq 1.532$</th>
<th>$(g_{10})_{\text{max}} \simeq 2.940$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{\text{min}} \simeq 0.2466$</td>
<td>$(g_9)_{\text{min}} \simeq 0.118$</td>
<td>$(g_{10})_{\text{min}} \simeq 1.420$</td>
</tr>
<tr>
<td>$\langle f \rangle = 2$</td>
<td>$\langle g_9 \rangle \simeq 0.5997$</td>
<td>$\langle g_{10} \rangle = 2$</td>
</tr>
<tr>
<td>$c_f = 0.906$</td>
<td>$c_9 \simeq 0.857$</td>
<td>$c_{10} \simeq 0.349$</td>
</tr>
</tbody>
</table>

Kernel $h_{10}$ retained the mean value but further attenuated the contrast by pushing the amplitudes toward the mean. However, the difference-of-averages kernel $h_9$ actually increased the contrast and decreased the mean value.
Outputs from filters with impulse responses \( h_9[n] = \frac{1}{48} \text{RECT} \left( \frac{n}{48} \right) \cdot \cos \left( \frac{2\pi n}{64} \right) \), which now has some negative amplitudes.

This trend may be continued by increasing the width of the RECT and using equal scale factors:

\[
\begin{align*}
    h_{11}[n] &= \frac{1}{64} \cos \left( \frac{2\pi n}{64} \right) \text{RECT} \left( \frac{n}{64} \right) \\
    h_{12}[n] &= \frac{1}{64} \text{RECT} \left( \frac{n}{64} \right)
\end{align*}
\]

Because the width of the RECT matches the period of the cosine in \( h_1 \), it may not be normalized to unit area. The extrema of these two processes are:

\[
\begin{array}{|c|c|c|}
\hline
f_{\text{max}} &= 5 & (g_{11})_{\text{max}} & \approx 0.712 \\
\kern1cm f_{\text{min}} & \approx 0.2466 & (g_{11})_{\text{min}} & \approx -0.511 \\
\kern1.5cm & \quad (g_{12})_{\text{max}} & = 2 + \frac{2}{\pi} & \approx 2.637 \\
\kern1cm & \quad (g_{12})_{\text{min}} & \approx +1.364 \\
\hline
\langle f \rangle & = 2 & \langle g_{11} \rangle & \approx 0 & \langle g_{12} \rangle & = 2 \\
\kern1cm & \kern1cm c_f = 0.906 & \kern1cm c_{11} & \approx 6.08 \quad (?!?) & \kern1cm c_{12} & \approx 0.318 \\
\end{array}
\]

The uniform averager \( h_1 \) continues to push the amplitudes toward the mean value of 2 and decreases the contrast, while the mean amplitude generated by the difference of averages kernel is now zero, which means that the minimum is less than zero.

Note that the output \( g_{11}[n] \) looks like the kernel \( h_{11}[n] \); in other words, the portion of \( f[n] \) that was transmitted to \( g_{11}[n] \) largely is a cosine of period 64. The distortion is due to cosines at other frequencies.
Outputs from convolutions with $h_{11}[n] = \frac{1}{64} \text{RECT} \left[ \frac{n}{64} \right] \cdot \cos \frac{2\pi n}{64}$ and $h_{12}[n] = \frac{1}{64} \text{RECT} \left[ \frac{n}{64} \right]$. The bipolar impulse response has zero area and “blocks” the constant part of the signal.

Now, consider filters whose widths are equal to the period of $f[n]$: 

$$h_{13}[n] = \cos \left[ 2\pi \frac{n}{64} \right] \cdot \frac{1}{128} \text{RECT} \left[ \frac{n}{128} \right]$$

$$h_{14}[n] = \frac{1}{128} \text{RECT} \left[ \frac{n}{128} \right].$$

The figures of merit for the gray values of these arrays are:

<table>
<thead>
<tr>
<th></th>
<th>$f_{\text{max}} = 5$</th>
<th>$(g_{13})_{\text{max}} = 0.5$</th>
<th>$(g_{14})_{\text{max}} = +2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{\text{min}} \approx 0.2466$</td>
<td>$(g_{13})_{\text{min}} \approx -0.5$</td>
<td>$(g_{14})_{\text{min}} = +2$</td>
<td></td>
</tr>
<tr>
<td>$\langle f \rangle = 2$</td>
<td>$\langle g_{13} \rangle = 0$</td>
<td>$\langle g_{14} \rangle = 2$</td>
<td></td>
</tr>
<tr>
<td>$c_f = 0.906$</td>
<td>$c_{13} = \infty$</td>
<td>$c_{14} = 0.0$</td>
<td></td>
</tr>
</tbody>
</table>

The output of the bipolar kernel $h_{13}$ is a sinusoid with period 64 and zero mean, while that of the averager $h_{14}$ is the constant average value of $f[n]$. Note that the contrast parameter $c_{13}$ is undefined because $f_{\text{min}} = -f_{\text{max}}$, while $c_{14} = 0$. 
Outputs resulting from convolution with $h_{13} = \frac{1}{128} \text{RECT} \left[ \frac{n}{128} \right] \cdot \cos \left[ 2\pi \frac{n}{64} \right]$ and $h_{14} = \frac{1}{128} \text{RECT} \left[ \frac{n}{128} \right]$; the former is a bandpass filter and the latter is a lowpass filter.

To summarize, kernels that compute differences of averages are wide and bipolar, and typically yield bipolar outputs. As the width of a difference-of-averages kernel is increased, the output resembles the kernel itself to a greater degree, which is bipolar with zero mean. On the other hand, increasing the width of an averaging operator results in outputs that approach a constant amplitude (the average value of the input); this constant is a cosine with infinite period. The difference-of-averages operator rejects BOTH slowly and rapidly varying sinusoids, and preferentially passes a particular sinusoidal frequency or band of frequencies. Thus, differences of averages operators are called bandpass filters.

Kernels of bandpass filters are wide, bipolar, and resemble the signal to be detected.

16.6 Implementation of Filtering

16.6.1 Nonlinear and Shift-Variant Filtering

Special-purpose hardware (array processors) is readily available for implementing linear operations. These contain several memory planes and can store multiple copies of the input. By shifting the addresses of the pixels in a plane and multiplying the values by a constant, the appropriate shifted and weighted image can be generated. These are then summed to obtain the filtered output. A complete convolution can be computed in a few clock cycles.

Other than the mean filter, the statistical filters are nonlinear, i.e., the gray value of the output pixel is obtained from those of the input pixel by some method other than multiplication by weights and summing. In other words, they are not convolutions and thus cannot be specified by a kernel. Similarly, operations that may be linear (output is a sum of weighted inputs) may use different weights at different locations in the image. Such operations are shift-variant and must be specified by
different kernels at different locations. Nonlinear and shift-variant operations are computationally intensive and thus slower to perform unless special single-purpose hardware is used. However, as computer speeds increase and as prices fall, this is becoming less of a problem. Because of the flexibility of operations possible, nonlinear shift-variant filtering is a very active research area.

16.7 Neighborhood Operations on Multiple Images

16.7.1 Image Sequence Processing

It should now be obvious that we can combine the gray levels in neighborhoods of the input pixel in multiple images to obtain the output image $g[x, y]$. The multiple copies of $f[x, y]$ may have been spread over time (e.g. video), over wavelength (e.g. RGB images), or some other parameter. In these cases, the image and kernel are functions of three coordinates.

Schematic of neighborhood operation on multiple images that differ in some other characteristic (time, wavelength, or focus depth).

16.7.2 Spectral + Spatial Neighborhood Operations

Additive noise is a common corrupter of digital images and may disrupt classification algorithms based on gray-level differences, e.g. in multispectral differencing to segment remote-sensing images. The noise can be attenuated by combining spatial averaging and spectral differencing, i.e.

$$g[x, y] = f[x, y, \lambda_1] * h[x, y] - f[x, y, \lambda_2] * h[x, y]$$
where
\[ h[x, y] = \frac{1}{9} \begin{bmatrix} +1 & +1 & +1 \\ +1 & +1 & +1 \\ +1 & +1 & +1 \end{bmatrix} \]

Since convolution and subtraction are linear, the order of operations can be inter-changed:
\[ g[x, y] = (f[x, y, \lambda_1] - f[x, y, \lambda_2]) * h[x, y] \]

These can be combined into a single 3-D operation using a 3-D kernel \( h[x, y, \lambda] \)
\[ g[x, y] = f[x, y, \lambda_i] * h[x, y, \lambda_i] \]

where
\[
\begin{align*}
  h[x, y, \lambda_i] &= -\frac{1}{9} \begin{bmatrix} +1 & +1 & +1 \\ +1 & +1 & +1 \\ +1 & +1 & +1 \end{bmatrix} \\
  & \quad \begin{bmatrix} +1 & +1 & +1 \\ +1 & +1 & +1 \\ +1 & +1 & +1 \end{bmatrix} \\
  & \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}
\end{align*}
\]