CHAPTER 5

Patterns, Smoothing and Filters

In this chapter, we introduce methods for obtaining descriptions of the appearance of a small group of pixels. These methods can be used to find patterns, to suppress noise, and to control aliasing.

5.1 LINEAR FILTERS AND CONVOLUTION

Section 3.2.2 showed improvements in downsampling obtained by replacing each image pixel with a weighted average of pixels. This useful trick is easily generalized into an important idea. The weights in that section were either uniform, or large at the pixel of interest, and falling off at distant pixels. Changing the weights leads to interesting outcomes.

5.1.1 Convolution and Filtering

For the moment, think of an image as a two dimensional array of intensities. Assume that images are infinite in extent (you can pad a finite image with zeros), so as to avoid mischief with indices that refer to locations outside the range of the image. Write $\mathcal{I}_{ij}$ for the pixel at position i, j.

Procedure: 5.1 *Convolution*

Convolution uses a source image $\mathcal{S}$, and forms a new image $\mathcal{N}$ from that source. The i, j'th pixel in $\mathcal{N}$ is now a weighted average of a $(2k-1)\times(2k-1)$ window of pixels in $\mathcal{S}$, centered on i, j. The weights are in a mask $\mathcal{M}$ and produce $\mathcal{N}$ from the original image and the mask, using the rule

$$\mathcal{N}_{ij} = \sum_{uv} \mathcal{I}_{i-u,j-v}\mathcal{M}_{uv}.$$

Convolution is sometimes written

$$\mathcal{N} = \mathcal{M} * \mathcal{I}.$$

The procedure works whatever the mask $\mathcal{M}$ (the mask is sometimes called a *kernel* or a *filter*). In some sources, you might see $\mathcal{M} * *\mathcal{I}$ (to emphasize the fact that the image is 2D). This operation is known as *convolution*, and $\mathcal{M}$ is often called the *kernel* of the convolution. You should look closely at the expression; the "direction" of the dummy variable u (resp. v) has been reversed compared with what you might expect unless you have a signal processing background. A variant is sometimes called *correlation* or *filtering*.

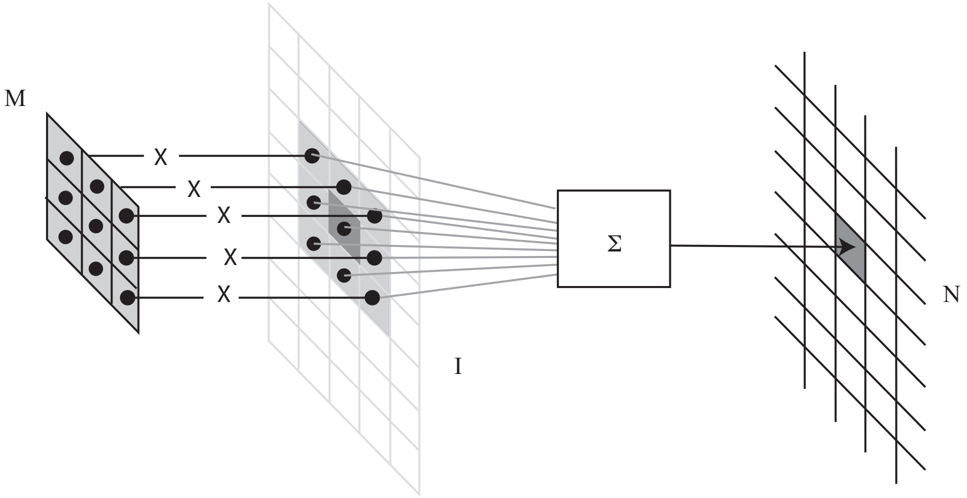

FIGURE 5.1: *To compute the value of $\mathcal{N} = \mathcal{W} * \mathcal{I}$. at some location, you shift a copy of $\mathcal{M}$ (the flipped version of $\mathcal{W}$) to lie over that location in $\mathcal{I}$; you multiply together the non-zero elements of $\mathcal{M}$ and $\mathcal{I}$ that lie on top of one another; and you sum the results. To compute the value of $\mathcal{N} = \mathtt{filter}(\mathcal{I}, \mathcal{W})$ at some location, just omit flipping $\mathcal{W}$.*

Procedure: 5.2 *Filtering*

Filtering computes

$$\mathcal{N}_{ij} = \sum_{uv} \mathcal{I}_{i+u,j+v} \mathcal{M}_{uv}$$

This is sometimes written

$$\mathcal{N} = \mathtt{filter}(\mathcal{M}, \mathcal{I}).$$

The difference between convolution and filtering isn't particularly significant, but if you forget that it is there, you compute the wrong answer. Notice that if you flip the mask $\mathcal{M}$ in both directions to form $\mathcal{W}$, so

$$\mathcal{W}_{uv} = \mathcal{M}_{-u,-v}$$

then

$$\mathtt{filter}(\mathcal{W}, \mathcal{I}) = \mathcal{M} * \mathcal{I}.$$

In each case, ignore the range of the sum, and assume that the sum is over a large enough range of u and v that all nonzero values are taken into account. Furthermore, assume that any values that haven't been explicitly specified are zero; this means that we can model the kernel as a small block of nonzero values in a sea of zeros. An important property of both convolution and filtering is that the result depends on the local pattern around a pixel, but not where the pixel is. Define the operation $\texttt{shift}(\mathcal{I}, m, n)$ which shifts an image so that the i, j'th pixel is moved to the $i - m$, $j - n$'th pixel, so

$$\texttt{shift}(\mathcal{I}, m, n)_{ij} = \mathcal{I}_{i-m,j-n}.$$

Ignore the question of the range, as `shift` just relabels pixel locations.

5.1.2 The Properties of Convolution

Most imaging systems have, to a good approximation, three significant properties. Write $R(f)$ for the response of the system to input f. Then the properties are:

- **Superposition:** the response to the sum of stimuli is the sum of the individual responses, so

  $$R(f + g) = R(f) + R(g);$$

- **Scaling:** the response to a scaled stimulus is a scaled version of the response to the original stimulus, so

  $$R(kf) = kR(f).$$

  An operation that exihibits superposition and scaling is *linear.*

- **Shift invariance:** In a shift invariant linear system, the response to a translated stimulus is just a translation of the response to the stimulus. This means that, for example, if a view of a small light aimed at the center of the camera is a small, bright blob, then if the light is moved to the periphery, the response is same small, bright blob, only translated.

A device that is linear and shift invariant is known as a *shift invariant linear system.* The operation represented by the device is a *shift invariant linear operation.*

Some systems accept a continuous signal and produce a continuous signal. A natural example is a lens, which takes a pattern of light and produces a pattern of light. Others accept a discrete signal (a vector; an array) and produce a discrete signal. A natural example would be smoothing with a Gaussian. Either kind of system can be linear, and either kind of system can be shift invariant. It turns out that any operation that is shift invariant and linear can be represented by a convolution, and this is true for either continuous or discrete signals.

Useful Fact: *Check that:*

- *Convolution is linear in the image, so*

$$\begin{aligned}\mathcal{W} * (k\mathcal{I}) &= k(\mathcal{W} * \mathcal{I}) \\ \mathcal{W} * (\mathcal{I} + \mathcal{J}) &= \mathcal{W} * \mathcal{I} + \mathcal{W} * \mathcal{J}\end{aligned}$$

  and filtering is linear in the image, too.

- *Convolution is linear in the mask, so*

$$\begin{aligned}(k\mathcal{W}) * \mathcal{I} &= k(\mathcal{W} * \mathcal{I}) \\ (\mathcal{W} + \mathcal{V}) * \mathcal{I} &= \mathcal{W} * \mathcal{I} + \mathcal{V} * \mathcal{I}\end{aligned}$$

  and filtering is linear in the mask, too.

- *Convolution is associative, so*

$$\mathcal{W} * (\mathcal{V} * \mathcal{I}) = (\mathcal{W} * \mathcal{V}) * \mathcal{I}$$

  and filtering is associative, too.

- *Convolution is shift-invariant, so*

$$\mathcal{W} * (\texttt{shift}(\mathcal{I}, m, n)) = \texttt{shift}(\mathcal{W} * \mathcal{I}, m, n)$$

  and filtering is shift-invariant, too.

5.1.3 Inconvenient Details

Now consider convolving an $M \times N$ image with a $2u + 1 \times 2v + 1$ kernel. Strips of the result of width v on the left and the right side and strips of height u at top and bottom contain values that are affected by pixels outside the image (Figure 5.2). Convolution could report only the values not affected by pixels outside the image – sometimes called the *valid region* of the convolution. This would turn an $M \times N$ image into an $M - 2u \times N - 2v$ image.

Attaching strips of width u on the left and right and height v on top and bottom would produce an image of size $M + 2u \times N + 2v$ – this is *padding*. Assuming the pixel values in these strips can be obtained somehow, convolving this padded image with the kernel would produce a $M \times N$ valid region. Padding like this is convenient, because there is no need to keep track of how much images have shrunk, but padding can have consequences (Section 21.3). Typically, API's make a variety of kinds of padding easy. One is *zero padding* (outside strips are zero); another is *reflection padding* (outside strips obtained by reflecting around the outer boundaries of the image).

The outer boundaries of an image mean that, in practice, convolution is not

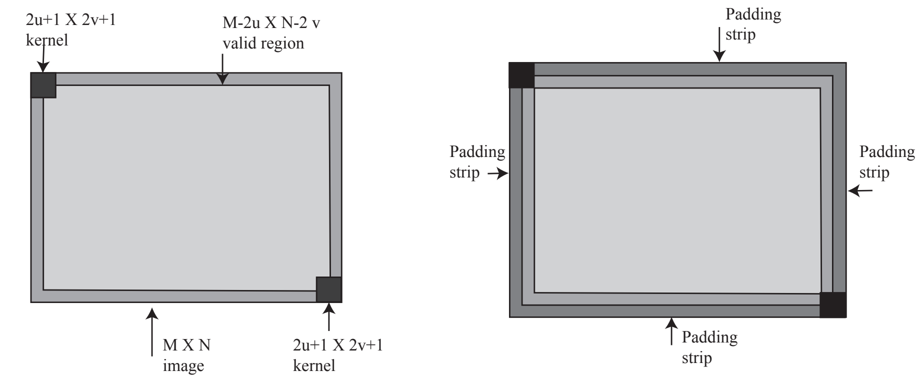

FIGURE 5.2: *To compute the value of the convolution for pixels at the edge of an image you will need to know the values of pixels outside the image. The set of valid pixels – those where you have all the image pixels you need to compute the convolution – is somewhat smaller than the original image. This inconvenience can be avoided by padding the image with values that lie outside the original block. The mid-gray box represents an $M \times N$ image, and the dark box a $2u+1 \times 2v+1$ kernel. The valid region is lighter gray. It can be constructed by placing the kernel at the top left and bottom right corners of the image, then constructing the box that joins their centers (***left***). A version of this construction reveals how the image should be padded to produce an $M \times N$ result. Place the center of the kernel at the bottom left and top right of the image, and construct the box that joins* their *outer corners (***right***). Most APIs offer a variety of procedures to put pixel values in these locations, including: zero-padding; constant padding; circular padding (join the edges of the image to form a torus); reflection padding (reflect image about the edges); and replication padding (make copies of the last row or column as required).*

shift-invariant. This is because, in practice, shifting an image is not just a matter of relabelling pixels. If one (say) pans a camera, some pixels "fall off" one edge of the image *and are lost*, and other new pixels with *new values* appear at the other edge. Convolution cannot be invariant to this operation, because the value of the convolution cannot be computed for unknown pixels.

5.1.4 Computing Image Gradients with Finite Differences

For an image $\mathcal{I}$, the gradient is

$$\nabla \mathcal{I} = (\frac{\partial \mathcal{I}}{\partial x}, \frac{\partial \mathcal{I}}{\partial y})^T,$$

which we could estimate by observing that

$$\frac{\partial \mathcal{I}}{\partial x} = \lim_{\delta x \to 0} \frac{\mathcal{I}(x+\delta x, y) - \mathcal{I}(x, y)}{\delta x} \approx \mathcal{I}_{i+1,j} - \mathcal{I}_{i,j}.$$

This means a convolution with

$$\begin{array}{cc} -1 & 1 \end{array}$$

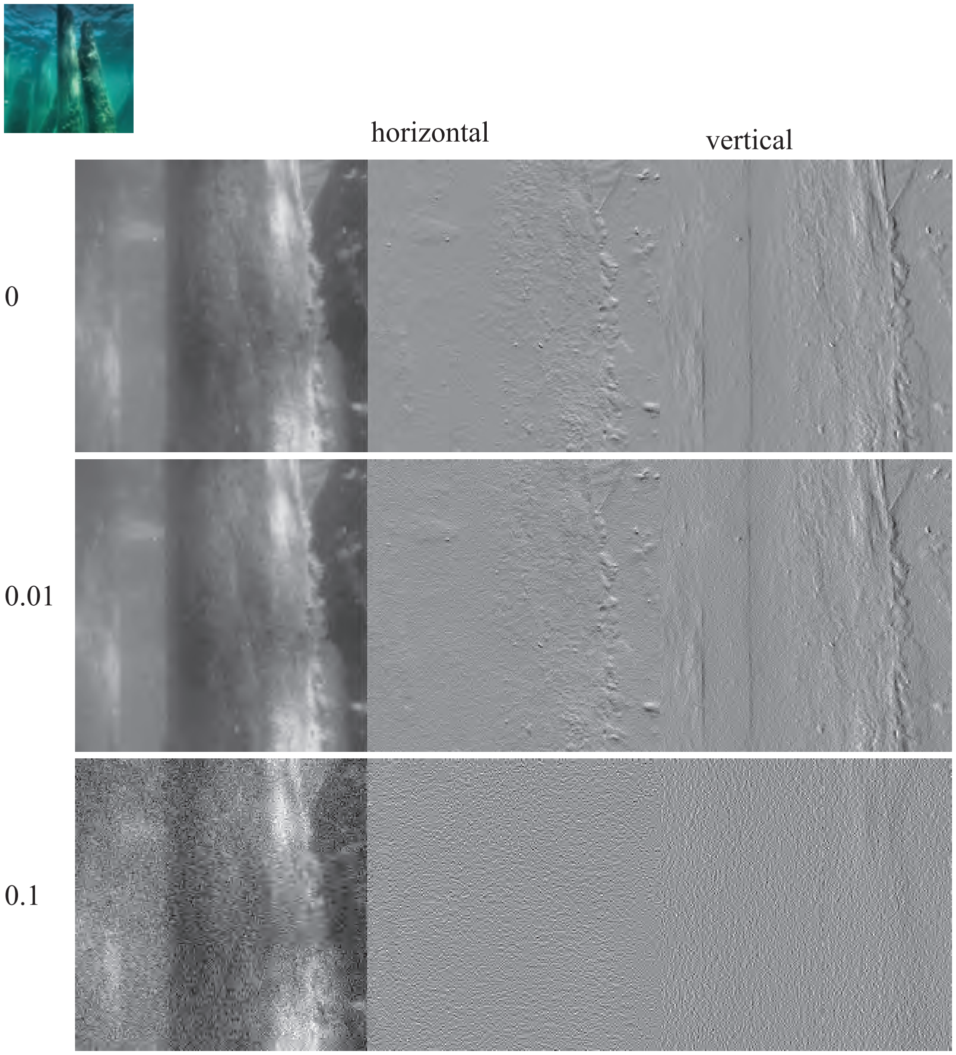

FIGURE 5.3: *Finite differences yield reasonable derivative estimates, but are strongly affected by noise.* **Top left** *shows the original image, from which a detail window is extracted and turned monochrome.* **Rows** *show image, horizontal derivative and vertical derivative, where derivatives are estimated by finite differences.* **First row** *is noise free image; others have additive Gaussian noise added, with standard deviation shown. Notice how this noise affects derivatives. The derivatives are scaled so that positive values are bright, negative values are dark, and 0 is mid-range. However the scale is chosen* per row, *which means the figure understates the effect of noise. In the noisy rows, the largest magnitude derivatives are much larger than in the clean row, which is why you can hardly see significant derivatives in the bottom row.* Image credit: *Figure shows Robert Forsyth's photograph of historical dock pilings in Lake Michigan.*

will estimate $\partial \mathcal{I}/\partial x$ (nothing in the definition requires convolution with a square kernel). Notice that this kernel "looks like" a dark pixel next to a light pixel, and will respond most strongly to that pattern. By the same argument,

$$\partial \mathcal{I}/\partial y \approx \mathcal{I}_{i,j+1} - \mathcal{I}_{i,j}.$$

These kinds of derivative estimates are known as *finite differences.* most unsatisfactory estimate of the derivative. This is because finite differences respond strongly (i.e., have an output with large magnitude) at fast changes, and fast changes are characteristic of noise. Roughly, this is because image pixels tend to look like one another. For example, if we had bought a discount camera with some pixels that were stuck at either black or white, the output of the finite difference process would be large at those pixels because they are, in general, substantially different from their neighbors. All this suggests that some form of noise suppression is appropriate before differentiation.

5.2 YOU SHOULD

5.2.1 remember these facts:

5.2.2 remember these procedures:

5.2.3 be able to:

- Convolve an image with a kernel using an API.
- Remember the properties of convolution.
- Recognize a filter as a simple pattern detector.
- Explain why a zero mean kernel is useful.
- Explain why normalized convolution is useful.
- Explain why a ReLU is useful.
- Visualize multi-channel convolution as a process that takes a block of data to another block of data.

EXERCISES

QUICK CHECKS

5.1. Show that convolution is linear in the image, so

$$\begin{aligned}\mathcal{W} * (k\mathcal{I}) &= k(\mathcal{W} * \mathcal{I}) \\ \mathcal{W} * (\mathcal{I} + \mathcal{J}) &= \mathcal{W} * \mathcal{I} + \mathcal{W} * \mathcal{J}.\end{aligned}$$

5.2. Show that filtering is linear in the image.

5.3. Show that convolution is linear in the mask, so

$$\begin{aligned}(k\mathcal{W}) * \mathcal{I} &= k(\mathcal{W} * \mathcal{I}) \\ (\mathcal{W} + \mathcal{V}) * \mathcal{I} &= \mathcal{W} * \mathcal{I} + \mathcal{V} * \mathcal{I}\end{aligned}$$

5.4. Show that filtering is linear in the mask.

5.5. Show that convolution is associative, so

$$\mathcal{W} * (\mathcal{V} * \mathcal{I}) = (\mathcal{W} * \mathcal{V}) * \mathcal{I}$$

5.6. Show that filtering is associative.

5.7. Show that convolution is shift-invariant, so

$$\mathcal{W} * (\texttt{shift}(\mathcal{I}, m, n)) = \texttt{shift}(\mathcal{W} * \mathcal{I}, m, n)$$

5.8. Show that filtering is shift-invariant.

5.9. Section 5.1.3 has “The outer boundaries of an image mean that, in practice, convolution is not shift-invariant. ” Explain

LONGER PROBLEMS

5.10. You can write the procedure that computes

$$\mathcal{N}_{ij} = \mathcal{I}_{i,j+1} - \mathcal{I}_{i,j-1}$$

as a convolution.

(a) What is the convolution kernel?

(b) Show that $\mathcal{N}$ gives an estimate of a partial derivative (this estimate is known as a *symmetric finite difference*).

(c) Find a pattern that (a) is not a constant and (b) gives a zero response from this derivative estimate.

(d) Why would you not use this procedure to estimate an image gradient.

5.11. You can write the procedure that computes

$$\mathcal{N}_{ij} = \mathcal{I}_{i,j+1} - 2\mathcal{I}_{ij} + \mathcal{I}_{i,j-1}$$

as a convolution.

(a) What is the convolution kernel?

(b) Show that $\mathcal{N}$ gives an estimate of a second partial derivative (this estimate is known as a *symmetric finite difference*).

5.12. What would you use the following filter kernels for?

(a)

$$\begin{matrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$$

(b)

$$\begin{array}{ccc} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{array}$$

(c)

$$\begin{array}{ccc} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{array}$$

(d)

$$\begin{array}{ccc} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{array}$$

PROGRAMMING EXERCISES

5.13. Obtain an image of text with black ink on a white background using a roman alphabet (as opposed to, say, greek letters; or bangla letters; etc. – if it is helpful, this text is in the roman alphabet).

(a) Build two filters, one to detect vertical stripes and another to detect horizontal stripes, using a Gaussian. If you write this filter as $ke^{-(\frac{1}{2})(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_y^2})}$, then you can detect horizontal stripes using a large value of σ_x and a small value of σ_y. Normalize the filters so the weights sum to one.

(b) Use this image to compare the original filters with zero-mean filters. You can obtain zero-mean filters by just subtracting the mean of the weights.

(c) Build detectors for vertical and horizontal dark bars on light backgrounds out of these filters and ReLU's. For images of text, is there any advantage to using normalized convolution?

(d) Now add detectors for vertical and horizontal light bars on dark backgrounds out of these filters and ReLU's. For images of text, is there any advantage to using these contrast reversed detectors as well as the originals?

5.14. Obtain a collection of 10 color images of small red fruits on a green leafy background (I searched an image search site for "redcurrants").

(a) Build a filter that detects circular red blobs on green backgrounds using multichannel convolution and Gaussians. Use this filters and ReLU's to build a very simple small red fruit detector. There are two steps. First, build a score for the "red fruitness" of each pixel using the filters and ReLU's. Next, count the red fruits. Do this using a procedure known as non-maximum suppression. For each pixel where the "red fruitness" score is larger than a threshold (which you set by experiment), check if the score is bigger than that at all eight neighboring pixels. If it is, then the pixel is a local maximum. Mark that pixel and erase its neighbors (at least eight, but possibly more). Continue until all pixels that are over threshold have been either erased or marked. Every marked location is a detector response. Use three of the 10 images to set good values of the threshold, the number of neighbors, and to select the filter. Further, check whether normalizing the convolution or using a zero-mean filter helps.

(b) Evaluate your detector on the seven remaining images. You should compute how many fruits it detects that are not there (the false positive rate) and how many fruits it misses (the false negative rate).

(c) Can you improve your detector by using more filters? The most likely improvement is obtained by using more than one scale. The sensible way to do this is to smooth and subsample each image, and apply the original filter to the smaller image as well as to the original. In this case, how can you ensure you don't detect the same fruit twice?