CHAPTER 6

Applications of Convolution

6.1 FINDING PATTERNS

Recall convolution takes a source image and produces a new image, notation in Section 5.1.1. You should think of the value of $\mathcal{N}_{ij}$ as a dot-product. Convolution with a mask is the same as filtering with a flipped mask. But when you filter, to compute the value of $\mathcal{N}$ at some location, you place the flipped version of the mask at some location in the image; you multiply together the elements of the image and the mask that lie on top of one another, ignoring everything in the image outside the mask; then you sum the results (Figure 5.1). Reindex the two windows to be vectors, and the result is a dot product.

6.1.1 Pattern Detection by Convolution

The properties of a dot product explain why a convolution is interesting: it can be used as a very simple pattern detector. Recall that if $\mathbf{u}$ and $\mathbf{v}$ are unit vectors, then $\mathbf{u}^T\mathbf{v}$ is maximized when $\mathbf{u} = \mathbf{v}$ and minimized when $\mathbf{u} = -\mathbf{v}$. Interpreting $\mathbf{u}$ as a vector of kernel weights and $\mathbf{v}$ as a vector of image values suggests the rough rule of thumb: filters respond most strongly to image patterns that look like the filter kernel.

The mean of $\mathbf{v}$ presents an issue. Write $\mathbf{1}$ for a vector of ones. Then $\mathbf{u}^T(\mathbf{v} + c\mathbf{1}) = \mathbf{u}^T\mathbf{v} + c\mathbf{u}^T\mathbf{1}$, so you can increase or decrease the response of the filter by adding a constant to the image window *unless* $\mathbf{u}^T\mathbf{1} = 0$. This suggests that the best pattern detection is obtained by using a filter with zero mean. If $\mathbf{u}^T\mathbf{1} = 0$, the magnitude of $\mathbf{u}$ just changes the scale of the response to the filter. The local maxima (or minima) of the response are what is important – these signal where a pattern is present – and so the magnitude of $\mathbf{u}$ doesn't really matter.

Useful Fact: *A zero-mean filter is a pattern detector that responds positively to image patches that look like it, and negatively to patches that look like it with a contrast reversal*

6.1.2 Normalized Convolution

If the mean of the kernel is zero, scaling the image will scale the value of the convolution. If you test the convolution value against a threshold to find a pattern, you will find more instances when the image gets brighter, and fewer when it gets darker, which is usually inconvenient. One strategy to build a somewhat better

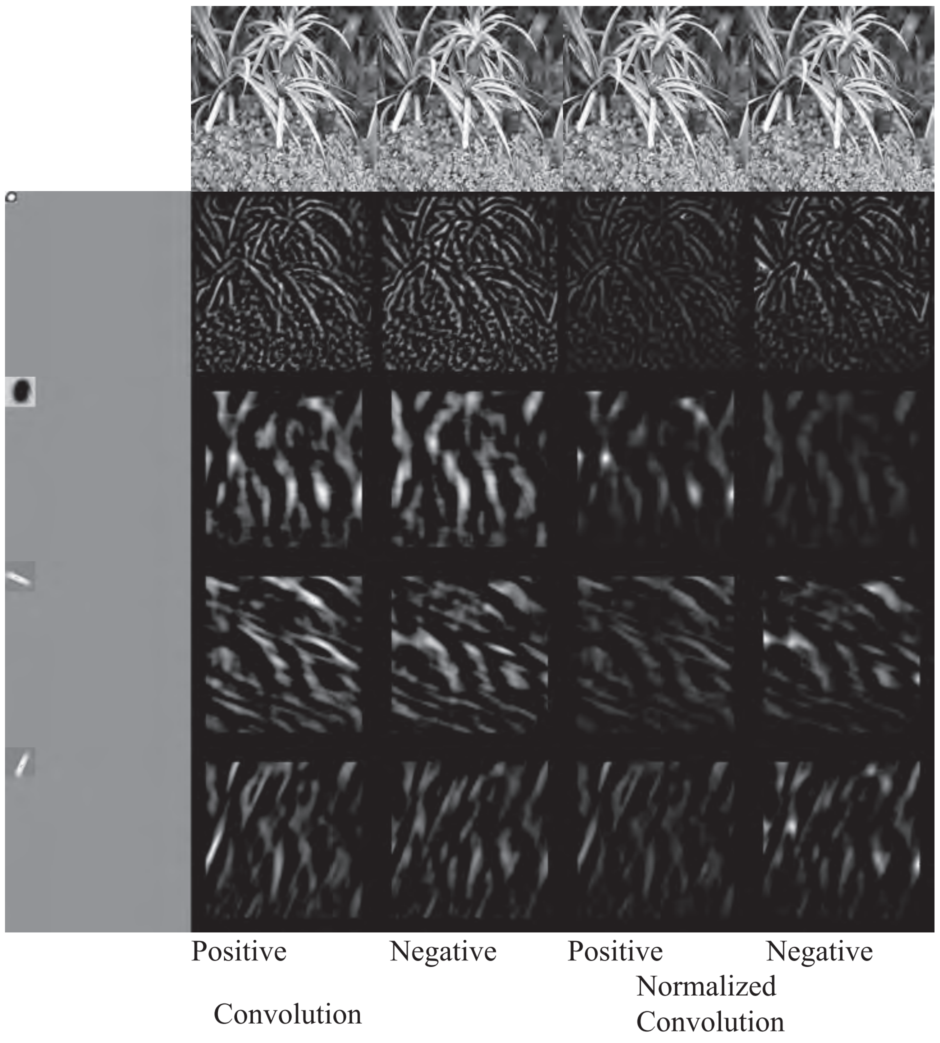

FIGURE 6.1: *Various zero-mean filters applied to a monochrome image of a pineapple plant (shown in the* **top row***, for reference), to show filters are simple pattern detectors. Details in the text.* Image credit: *Figure shows my photograph of a pineapple in the Singapore botanical garden.*

pattern detector is to normalize the result of the convolution to obtain a value that is unaffected by scaling the image. For example, smooth the image with a Gaussian to obtain $\mathcal{G}$, then form

$$\mathcal{C}_{ij} = \frac{\mathcal{N}_{ij}}{\mathcal{G}_{ij} + \epsilon}$$

(remember, $\mathcal{N}_{ij}$ was obtained by convolving the image with some zero-mean kernel). Here $\mathcal{G}$ is an estimate of how bright the image is. Most images have all positive pixels (a zero pixel value is usually a sign of camera problems) so using ϵ to avoid dividing by zero isn't essential. But note that $\epsilon > 0$ causes the score to saturate if the image is very dark. This makes sense because a group of very dark pixels is more likely to have a pattern present through thermal noise. The process that produces $\mathcal{C}$ is known as *normalized convolution*, and produces an improvement in the detector. Figure 6.1 compares normalized convolution to convolution. The right two frames show the positive and negative components of the normalized convolution (divide the filter responses by an estimate of image intensity). The normalized convolution is more selective. Responses are shown on a scale where zero is dark and a strong response is light. It is now more usual to manage these difficulties by learning kernels that behave well (Section 21.3).

Useful Fact: *Normalized convolution normalizes the score for a pattern using an estimate of intensity to produce a better pattern detector.*

6.1.3 ReLU's

Write $\mathcal{W}$ for a kernel representing some pattern you wish to find. Assume that $\mathcal{W}$ has zero mean, so that the filter gives zero response to a constant image. Notice that $\mathcal{N} = \mathcal{W} * \mathcal{I}$ is strongly positive at locations where $\mathcal{I}$ looks like $\mathcal{W}$, and strongly negative when $\mathcal{I}$ looks like a contrast reversed (so dark goes to light and light goes to dark) version of $\mathcal{W}$. Usually, you would want to distinguish between (say) a light dot on a dark background and a dark dot on a light background. Write

$$\texttt{relu}(x) = \begin{cases} x & \text{for } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

(often called a *Rectified Linear Unit* or more usually *ReLU*). Then $\texttt{relu}(\mathcal{W} * \mathcal{I})$ is a measure of how well $\mathcal{W}$ matches $\mathcal{I}$ at each pixel, and $\texttt{relu}(-\mathcal{W} * \mathcal{I})$ is a measure of how well $\mathcal{W}$ matches a contrast reversed $\mathcal{I}$ at each pixel. The ReLU will appear again.

Figure 6.1 give some examples. The filters are shown on the far left, each in the top left hand corner of a field of zeros the same size as the image; this gives some sense of spatial scale. The lightest value is the largest value of the filter, the darkest is the smallest. The left two frames show the positive and negative components of the response to the filter. The positive responses occur where (rather roughly) the image "looks like" the filter. Similarly, negative responses occur where the image "looks like" a contrast reversed version of the filter. Notice how the filters really are pattern detectors (the big dark blob gets responses from big dark blobs, and the small bright blob gets responses from small bright blobs), but they are not very good pattern detectors. Something that causes a bar filter to response will often also get a response from a blob filter. Further, the region of small bright leaves on the bottom of the image produces strong positive responses. The filter is linear, so bright patterns that don't look like the filter tend to give responses as strong as

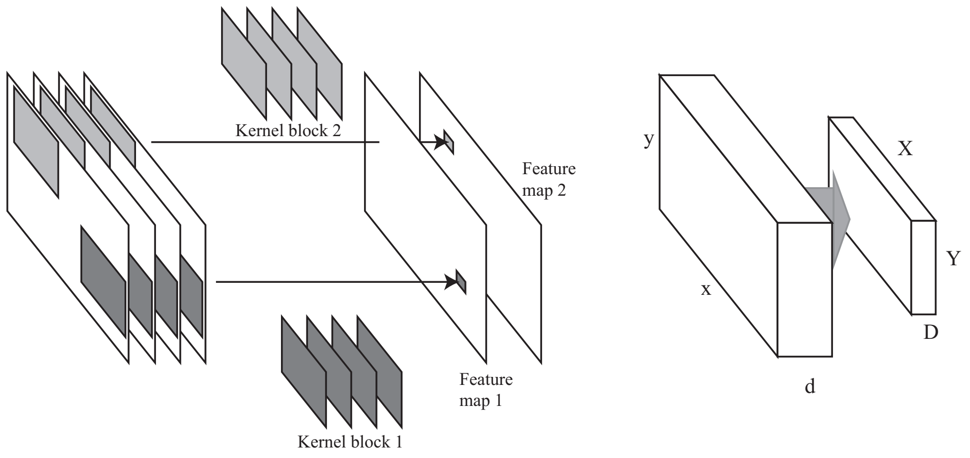

FIGURE 6.2: *On the* **left**, *two kernels (now 3D, as in the text) applied to a set of feature maps produce one new feature map per kernel, using the procedure of the text (the bias term isn't shown). Abstract this as a process that takes an $x \times y \times d$ block to an $X \times Y \times D$ block (as on the* **right**).

dark patterns that do. It can be useful to suppress small responses, and it is easy to do so by subtracting a small constant from the response before applying the ReLU (**exercises**).

Useful Fact: *A ReLU can be used to separate positive (use* `relu`(x)*) and negative responses (use* `relu`$(-x)$*) to produce a better pattern detector.*

6.1.4 Multi-Channel Convolution

The description of convolution anticipates monochrome images, and Figure 6.1 shows filters applied to a monochrome image. Color images are naturally 3D objects with two spatial dimensions (up-down, left-right) and a third dimension that chooses a slice or *channel* (R, G or B for a color image). Color images are sometimes called *multi-channel images*. Multi-channel images offer a natural for representations of image patterns, too — two dimensions that tell you where the pattern is and one that tells you what it is. For example, the results in Figure 6.1 can be interpreted as a block consisting of eight channels (four patterns, original contrast and contrast reversed). Each slice is the response of a pattern detector *for a fixed pattern*, where there is one response for each spatial location in the block, and so are often called *feature maps* (it is entirely fair, but not usual, to think of an RGB image as a rather uninteresting feature map).

For a color image $\mathcal{I}$, write $\mathcal{I}_{kij}$ for the k'th color channel at the i, j'th location, and $\mathcal{K}$ for a color kernel – one that has three channels. Then interpret $\mathcal{N} = \mathcal{I} * \mathcal{K}$

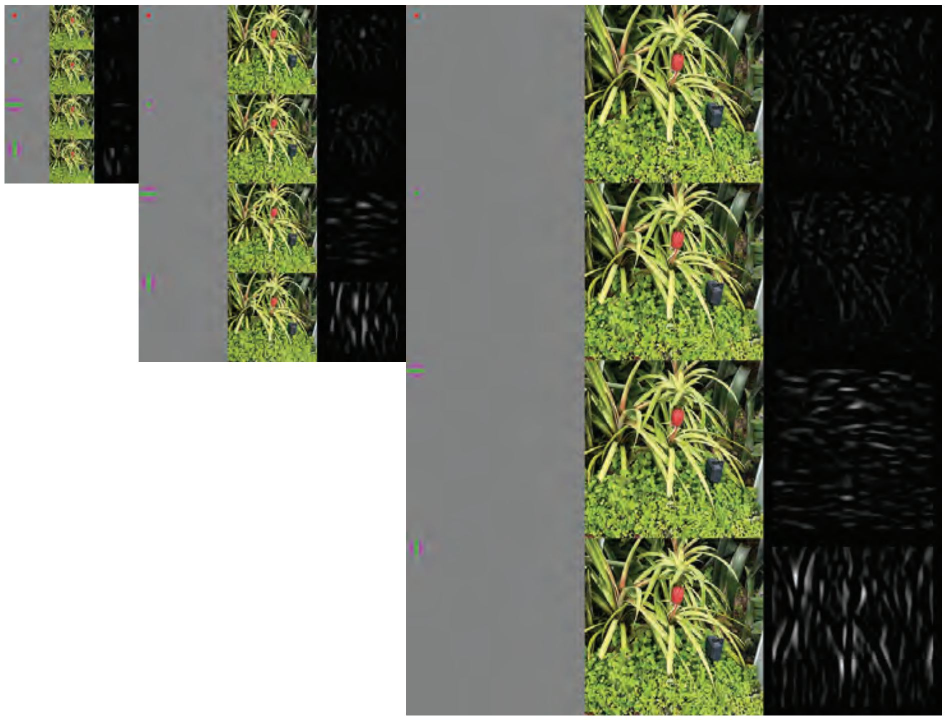

FIGURE 6.3: *Multichannel convolution easily yields a simple detector for colored patterns.* Image credit: *Figure shows my photograph of a pineapple in the Singapore botanical garden.*

as

$$\mathcal{N}_{ij} = \sum_{kuv} \mathcal{I}_{k,i-u,j-v} \mathcal{K}_{kuv}$$

which is an image with a single channel. This $\mathcal{N}$ is a single channel image that encodes the response to a single pattern detector. Much more interesting is an encoding of responses to multiple pattern detectors, and for that you must use multiple kernels (often known as a *filter bank*). Write $\mathcal{K}^{(l)}$ for the l'th kernel, and obtain a feature map

$$\mathcal{N}_{l,ij} = \sum_{kuv} \mathcal{I}_{k,i-u,j-v} \mathcal{K}^{(l)}_{kuv}.$$

This notation is quite clunky, because it isn't a three dimensional convolution (look at the directions of the indices). This never matters for our purposes. Another clunky feature of the notation is that applying the same kernel to each layer of a color image requires a fairly odd set of kernels (**exercises**). It has two enormous virtues. First, convolution can be used to detect colored patterns (Figure 6.3). Second, convolution becomes an operation that turns a three dimensional object – a stack of channels, a multi-channel image or a feature map, according to taste – into another such object, so you can apply a convolution to the results of a convolution.

Remember this: *Convolution and filtering are largely equivalent. Convolution can be used to smooth images and to detect patterns. ReLU's can be used to manage contrast reversal effects. Multichannel convolution can be used to build pattern detectors for color images. Multichannel convolution with R kernels will map a block of data structured as $C \times M \times N$ (by convention, C is the number of channels and M and N are spatial dimensions) to a block of data structured as $R \times U \times V$.*

6.1.5 Representing Images with Filter Banks

In the image in Figure 6.1, the leaves of the pineapple plant look like disorganized thick stripes. The leaves of the plant at its base are quite different, and look more like repeated small spots. These are examples of textures – somewhat unstructured patterns that are quite characteristic. Textures are widespread and quite distinctive – a field of pebbles looks quite different from a stand of corn; a cluster of pine needles looks very different from an expanse of bark; and so on.

Figure 6.1 also suggests a way to represent textures, and so images. Think of a texture as a collection of small patterns, arranged in some distinctive way. An image region showing a field of pebbles would have many spots, some small, some large and most medium, but very few thin bars. In contrast, an image region showing a cluster of pine needles would have many thin bars, pointing in about the direction, but very few small or large spots. Then to build an image representation: (a) construct a vocabulary of patterns; (b) find out which patterns are present at which pixel; and then (c) building a summary of which patterns are present in a region.

Because the patterns are likely so variable, an elaborate or detailed pattern detector is likely to be unhelpful – something that precisely detects a pine needle would need to be tuned to exactly the right angle, which would be a nuisance – so it is natural to use filters as pattern detectors. However, it is helpful to distinguish between, say, light thin bars on dark backgrounds (possible pine needles) and dark thin bars on light backgrounds (possible gaps between needles).

For the moment, assume the vocabulary of patterns is given, represented as a filter bank. Then the next two steps are straightforward. To find the patterns in an image, construct the response of all the filters at all points and apply a ReLU. Stack these responses into a multi-channel image. To compute a summary, construct a local weighted average of each channel of the multi-channel image at each pixel.

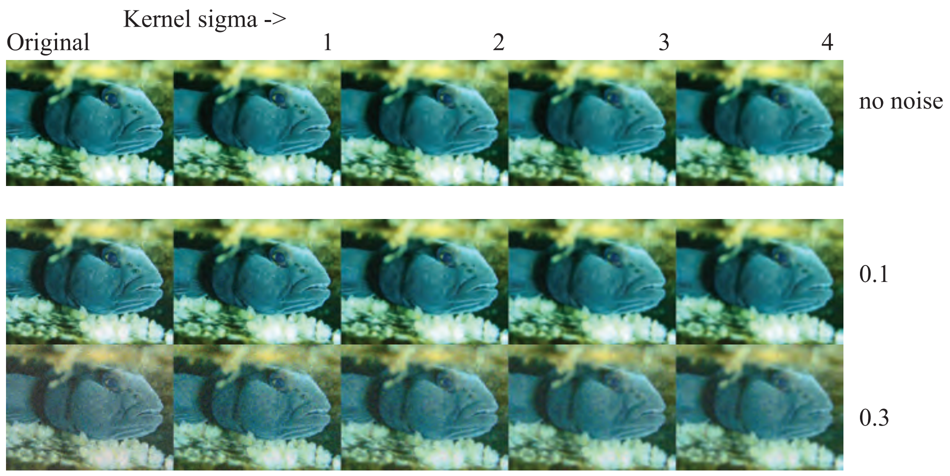

FIGURE 6.4: *Smoothing an image with a gaussian kernel is an effective way to suppress additive Gaussian noise.* **Left column** *the original image, followed by versions smoothed with a gaussian kernel with $\sigma = 1$, $\sigma = 2$, $\sigma = 3$ and $\sigma = 4$.* **Top row** *shows results on a noise free image;* **middle row** *shows results on an image with additive stationary gaussian noise with standard deviation 0.01, where the value of a pixel ranges from 0 to 1;* **bottom row** *shows results on an image with additive stationary gaussian noise with standard deviation 0.1. Notice how (a) smoothing blurs the original image; (b) more smoothing leads to more blur; (c) smoothing suppresses noise (so a smoothed version of a noisy image is close to the smoothed version of the original); and (d) more smoothing suppresses more noise.* Image credit: *Figure shows Robert Forsyth's photograph of a goby on its nest in Lake Michigan.*

Remember this: *Images can be represented using the outputs of multiple filters, formed into a bank. Convolutions with simple filter kernels will find the x- and y-derivatives of an image, yielding the image gradient. This gradient estimate is significantly affected by image noise.*

6.2 DENOISING

6.2.1 Suppressing Noise with Filters

The simplest model of image noise is the *additive stationary Gaussian noise* (or *Gaussian noise*) model, where each pixel has added to it a value chosen independently from the same normal (Gaussian – same Gauss, different sense) probability distribution. This distribution almost always has zero mean. The standard deviation is a parameter of the model. Figure 6.4 shows some examples of additive

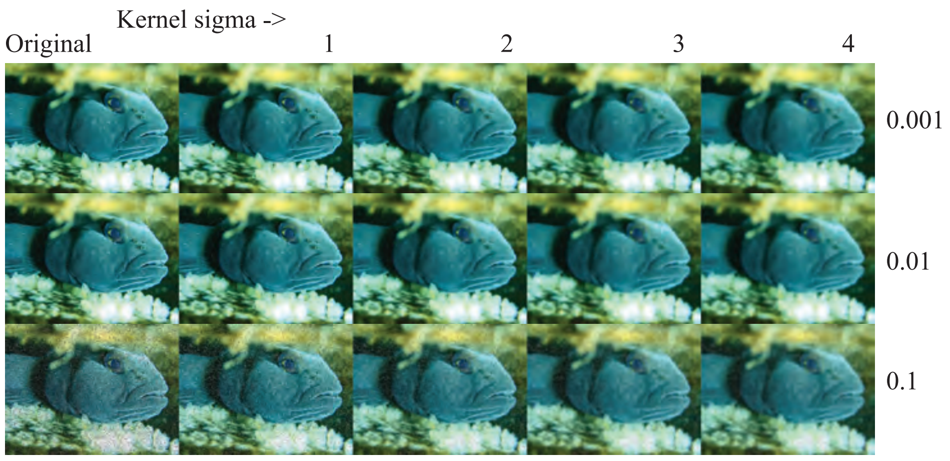

FIGURE 6.5: *Images at various noise levels smoothed with various gaussian kernels. The noise here involves picking pixel locations uniformly at random in the image, then flipping them either full light or full dark. The number on the* **far right** *shows the probability of a pixel being flipped (so at* 0.001*, a* 30×30 *window should have about one flipped pixel in it; at* 0.01*, a* 10×10 *window should have about one flipped pixel in it; and at* 0.1*, a* 3×3 *window should have one flipped pixel in it).* **Left** *the original image, followed by versions smoothed with* $\sigma = 1$*,* $\sigma = 2$*,* $\sigma = 3$ *and* $\sigma = 4$*. Notice how (a) smoothing blurs the original image; (b) more smoothing leads to more blur; (c) smoothing suppresses noise (so a smoothed version of a noisy image is close to the smoothed version of the original); and (d) more smoothing suppresses more noise. The noise-free image is top left in Figure 6.4.* Image credit: *Figure shows Robert Forsyth's photograph of a goby on its nest in Lake Michigan.*

stationary Gaussian noise.

Images can be quite effectively denoised because "pixels look like their neighbors". This important and very reliable slogan is in scare quotes because, while it is an extremely important practical guide, making it precise is neither easy nor particularly useful. Generally, pictures show objects which are span a large number of pixels, and where the shading changes relatively slowly over the surface of the object. This means that the value at a pixel is likely to be close to the value at its neighbor. Although this isn't true of every pixel – otherwise there wouldn't be edges in images – it is true of most pixels. If you have a pixel whose value is unknown, looking at its neighbors will almost always yield a good estimate. A pixel that does not look like its neighbors is suspect.

Check that the smoothing used in the downsampling strategy of Section 3.2.4 is a convolution with a gaussian kernel. This procedure is called *gaussian smoothing* or very often just *smoothing*. It turns out that this procedure is very good at suppressing many kinds of image noise. Figure 6.4 shows examples of suppressing additive Gaussian noise, and the exercises **exercises** explore some details. Gaussian smoothing can suppress the effects of other noise processes, too (Figure 6.5).

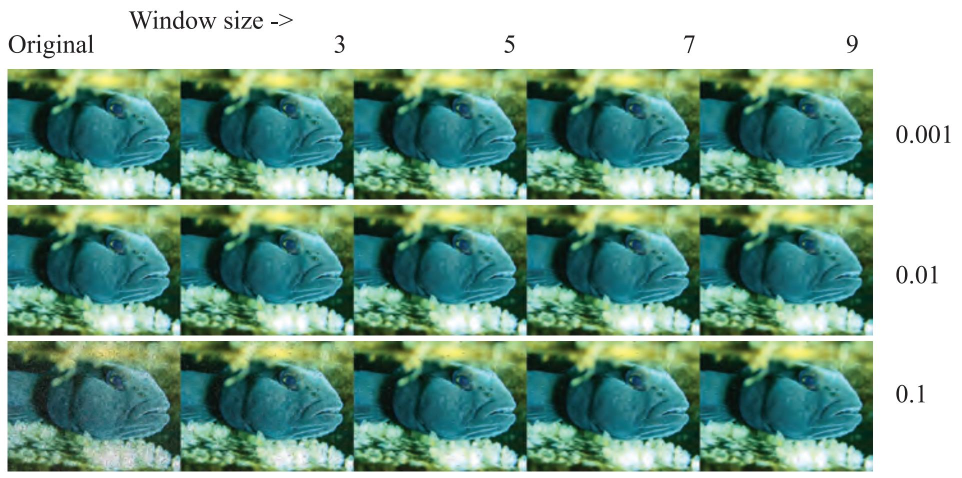

FIGURE 6.6: *Images at various noise levels smoothed with a median filter. The noise here involves picking pixel locations uniformly at random in the image, then flipping them either full light or full dark. The number on the* **far right** *shows the probability of a pixel being flipped (so at* 0.001*, a* 30×30 *window should have about one flipped pixel in it; at* 0.01*, a* 10×10 *window should have about one flipped pixel in it; and at* 0.1*, a* 3×3 *window should have one flipped pixel in it.* **Left** *the original image, followed by versions where the median is taken in windows of different sizes. Notice how (a) the median filter preserves edges rather well, even over big windows; (b) bigger windows lead to more noise suppression; and (c) texture details are suppressed by the median, with bigger windows suppressing more.* Image credit: *Figure shows Robert Forsyth's photograph of a goby on its nest in Lake Michigan.*

The choice of σ (or scale) for the Gaussian follows from the following considerations:

- If the standard deviation of the Gaussian is very small—say, smaller than one pixel—the smoothing will have little effect because the weights for all pixels off the center will be very small.
- For a larger standard deviation, the neighboring pixels will have larger weights in the weighted average, which in turn means that the average will be strongly biased toward a consensus of the neighbors. This will be a good estimate of a pixel's value, and the noise will largely disappear at the cost of some blurring.
- Finally, a kernel that has a large standard deviation will cause much of the image detail to disappear, along with the noise.

6.2.2 Median Filters

Most image noise tends to result in pixels not looking like their neighbors. However, gaussian smoothing is not always effective at estimating the true value of noisy

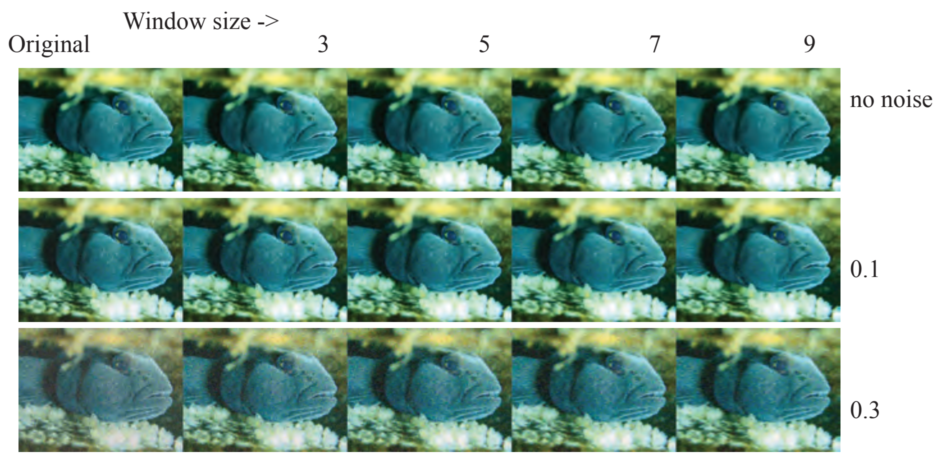

FIGURE 6.7: *Images at various noise levels smoothed with a median filter. The noise here is additive Gaussian noise.* **Left** *the original image, followed by versions where the median is taken in windows of different sizes.* **Top row** *shows results on a noise free image;* **middle row** *shows results on an image with additive stationary gaussian noise with standard deviation 0.01 (where the value of a pixel ranges from 0 to 1);* **bottom row** *shows results on an image with additive stationary gaussian noise with standard deviation 0.1. Notice how (a) the median filter preserves edges rather well, even over big windows; (b) bigger windows lead to more noise suppression; and (c) texture details are suppressed by the median, with bigger windows suppressing more.* Image credit: *Figure shows Robert Forsyth's photograph of a goby on its nest in Lake Michigan.*

pixels. For example, look closely at Figure 6.5. The noise process – a *Poisson noise process*, sometimes called *salt and pepper noise* – picks pixel locations uniformly at random in the image, then flips the result either full light or full dark. This means that a noisy pixel contains no information, and might be very different from its neighbors. If you compute a weighted average in a region that contains a noisy pixel, that weighted average might be severely disrupted by the noise, even if the center is a clean pixel. For example, think of a dark neighborhood on the goby where noise has turned one pixel bright – the bright pixel will dominate the average unless it contains a very large number of pixels with quite large weights. And in that case, the image will be blurry.

This suggests the entirely natural alternative of computing a median in a neighborhood as an estimate of the value at a pixel. As Figure 6.6 shows, this can be very effective at suppressing noise. Notice an attractive feature of the median filter – it tends not to blur edges, even when it strongly smoothes the interior of image regions. Median filters are somewhat more expensive computationally than smoothing, but deal fairly well with additive gaussian noise as well as salt and pepper noise (Figure 6.7)

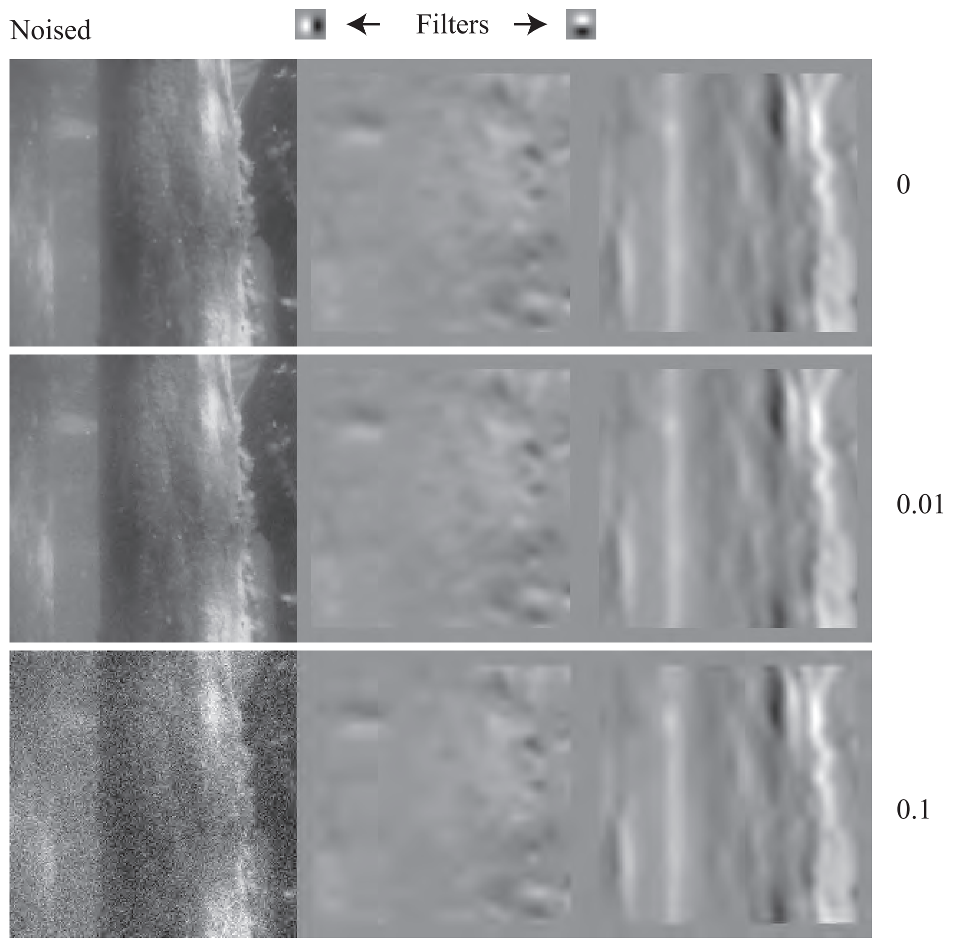

FIGURE 6.8: *Derivative of gaussian filters, equivalent to applying a finite difference to a smoothed image, very significantly improve estimates of derivatives. Compare this figure with Figure 5.3.* **Rows** *show image, horizontal derivative and vertical derivative, where derivatives are estimated by convolution with difference of gaussian filters. The filters are shown at the top. As you should expect, one looks like a dark bar next to a light bar, the other looks like a dark bar below a light bar. As should be apparent from the filters, the smoothing is the same in each direction.* **First row** *is noise free image; others have additive Gaussian noise added, with standard deviation shown on the* **right**. *Notice how this noise hardly affects derivatives. The derivatives are scaled so that positive values are bright, negative values are dark, and 0 is mid-range. Although the scale is chosen* per row, *the derivative images look the same from row to row – this means that each row has about the same largest magnitude value.* Image credit: *Figure shows Robert Forsyth's photograph of historical dock pilings in Lake Michigan.*

6.2.3 Application: Derivative of Gaussian Filters

Because convolution is associative, smoothing an image and then differentiating it is the same as convolving it with the derivative of a smoothing kernel. First, differentiation is linear and shift invariant. This means that there is some kernel that differentiates. Given a function $I(x, y)$,

$$\frac{\partial I}{\partial x} = K_{(\partial/\partial x)} * I.$$

Write the convolution kernel for the smoothing as S. Now

$$(K_{(\partial/\partial x)} * (S * I)) = (K_{(\partial/\partial x)} * S) * I = (\frac{\partial S}{\partial x}) * I.$$

Usually, the smoothing function is a gaussian, so an estimate of the derivative can be obtained by convolving with the derivative of the gaussian (rather than convolve and then differentiate), yielding

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

As they should (Section **??**), derivative of gaussian filters look like the effects they are intended to detect. The x-derivative filters look like a vertical light blob next to a vertical dark blob (an arrangement where there is a large x-derivative), and so on.

Large gradients in images are interesting (Chapters **??** and **??**) because they tend to occur on the outlines of objects, at shadow boundaries, and so on. Generally, if there is a large x derivative at a pixel, there will be a large x derivative at neighboring pixels. Smoothing *across* the direction of the derivative may result in smeared or blurred derivatives; but smoothing *along* the direction of the derivative will tend to average the value at points with similar derivatives and improve the noise resistance. It is quite usual to use

$$\begin{aligned}\frac{\partial g_\sigma}{\partial x} &= \frac{1}{2\pi\sigma_s\sigma_b}\left[\frac{-x}{2\sigma_s^2}\right]\exp -\left(\frac{x^2}{2\sigma_b^2}+\frac{y^2}{2\sigma_s^2}\right)\\ \frac{\partial g_\sigma}{\partial y} &= \frac{1}{2\pi\sigma_s\sigma_b}\left[\frac{-y}{2\sigma_s^2}\right]\exp -\left(\frac{x^2}{2\sigma_b^2}+\frac{y^2}{2\sigma_s^2}\right)\end{aligned}$$

where $\sigma_b > \sigma_s$. Smoothing results in much smaller noise responses from the derivative estimates, and more smoothing yields less noisy, but more blurry, gradients (Figure 6.8 and 6.9).

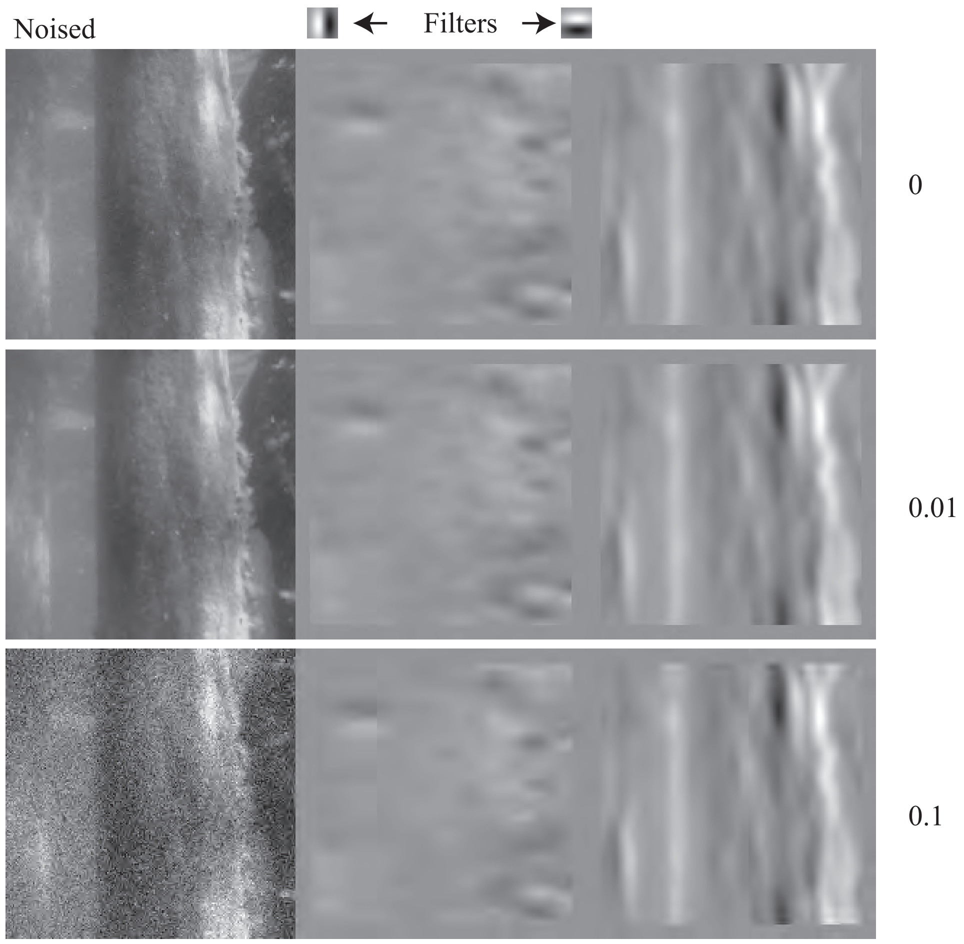

FIGURE 6.9: *Derivative of gaussian filters, equivalent to applying a finite difference to a smoothed image, very significantly improve estimates of derivatives. Compare this figure with Figure 5.3. As should be apparent from the filters, the smoothing is over a much larger range along the derivative direction than across it (compare Figure 6.8).* Image credit: *Figure shows Robert Forsyth's photograph of historical dock pilings in Lake Michigan.*

Remember this: *Smoothing with a Gaussian will denoise images. An alternative that works better for some kinds of noise is a median filter, which is not linear and is not a convolution. Smoothing with a Gaussian then differentiating will obtain an image gradient estimate that is less sensitive to noise. This process is better represented as convolution with the derivative of Gaussian.*

6.3 YOU SHOULD

6.3.1 remember these terms:

A zero-mean filter is a pattern detector. 89
Normalized convolution is better for pattern detection than convolution. 91
ReLU's help in pattern detection. 92
Convolution and filtering are crucial building blocks. 94
Applications of convolution include representing images with a filter bank and computing the image gradient. 95
Applications of convolution further include denoising images and obtaining a low-noise gradient estimate. 102

6.3.2 remember these facts:

A zero-mean filter is a pattern detector. 89
Normalized convolution is better for pattern detection than convolution. 91
ReLU's help in pattern detection. 92
Convolution and filtering are crucial building blocks. 94
Applications of convolution include representing images with a filter bank and computing the image gradient. 95
Applications of convolution further include denoising images and obtaining a low-noise gradient estimate. 102

6.3.3 remember these procedures:

6.3.4 be able to:

- Recognize a bank of filters as a way to represent small patterns in images.
- Denoise an image by smoothing with either Gaussian or median filters.
- Form a gradient estimate using derivative of Gaussian filters.
- Use the model of sampled functions in simple calculations.
- Recognize interpolation as a convolution that passes from a sampled function to a continuous function.

EXERCISES

QUICK CHECKS

6.1. Why would a zero-mean filter be a poor choice for smoothing Gaussian noise?
6.2. Why is a non zero-mean filter a poor choice of pattern detector?
6.3. Why is a normalized convolution useful?
6.4. Why does "subtracting a small constant from the response before applying the ReLU" (Section 6.1.3) help suppress small responses to a pattern detector?
6.5. Is normalized convolution linear in the convolution kernel?
6.6. Is normalized convolution linear in the image?
6.7. Is multichannel convolution linear in the convolution kernel?
6.8. Is multichannel convolution linear in the image?
6.9. Can you normalize multichannel convolution?
6.10. Can you construct a zero-mean kernel for multichannel convolution?

LONGER PROBLEMS

6.11. This exercise explores smoothing of additive Gaussian noise. Write Ξ for a noise image (i.e. the image whose i, j'th component is ξ_{ij}), where ξ_{ij} is an independent, identically distributed sample from a standard normal distribution (this has mean 0 and standard deviation 1). Write $\mathcal{K}$ for some $(2k+1)\times(2k+1)$ filter kernel.

(a) Form $\mathcal{M} = \mathcal{K} * \Xi$. Show that each pixel of this image is a sample of a normal distribution whose mean is 0 and whose standard deviation is

$$\sqrt{\sum_{ij} \mathcal{K}_{ij}^2}.$$

(b) Assume $k >= 1$. Why are m_{ij} and $m_{i+1,j}$ not independent?
(c) What condition on k ensures every pixel of $\mathcal{M}$ is independent of every other?
(d) For given k, characterize the pixels that are guaranteed to be independent of one another?
(e) What is the covariance of $\mathcal{M}_{ij}$ and $\mathcal{M}_{i+r,j+s}$ for given r and s?
(f) Assume $K_{ij} \geq 0$ and $\sum_{ij} K_{ij} = 1$. What can you say about

$$\sqrt{\sum_{ij} \mathcal{K}_{ij}^2}?$$

6.12. This exercise explores smoothing of additive Gaussian noise. Write $\mathcal{I}$ for an image whose i, j'th entry is $\mathcal{I}_{ij}$. Form $\mathcal{N}_{ij} = \mathcal{I}_{ij} + \sigma\xi_{ij}$, where ξ_{ij} is an independent, identically distributed sample from a standard normal distribution (this has mean 0 and standard deviation 1).

(a) Use the results of the previous exercise to argue that Gaussian smoothing suppresses Gaussian noise.

6.13. This exercise explores smoothing of another noise model. Write $\mathcal{I}$ for an image whose i, j'th entry is $\mathcal{I}_{ij}$. Form $\mathcal{N}_{ij} = \mathcal{I}_{ij} + \sigma\delta_{ij}$, where δ_{ij} is an independent, identically distributed sample from a Bernoulli distribution with parameter p (so with probability p, $\delta_{ij} = 1$ and with probability $1-p$, $\delta_{ij} = 0$). Equivalently, at each pixel in the image, you flip a biased coin. If it comes up heads (probability p), add σ to the pixel value. Write $\mathcal{K}$ for some $k \times k$ filter kernel, and Δ for the noise image (i.e. the image whose i, j'th component is δ_{ij}).

(a) Assume p is very small. Describe $\mathcal{M} = \mathcal{K} * \Delta$ qualitatively. What is the mean of each pixel? What is the variance of each pixel?

(b) Use the results of the previous exercise to argue that Gaussian smoothing suppresses this noise.

6.14. Write $\mathcal{I}$ for an image whose i, j'th entry is $\mathcal{I}_{ij}$. Form $\mathcal{N}_{ij} = \mathcal{I}_{ij} + \sigma\xi_{ij}$, where ξ_{ij} is an independent, identically distributed sample from a standard normal distribution (this has mean 0 and standard deviation 1). This means that, at each pixel in the image, you draw a sample from a standard normal distribution, scale it by σ, then add it to the pixel value. Write Ξ for the noise image (i.e. the image whose i, j'th component is ξ_{ij}) and $\mathcal{D}$ for the filter kernel

$$-1 \quad 1 \ .$$

(a) Form $\mathcal{M} = \mathcal{D} * \Xi$. Show that each pixel of this image is a sample of a normal distribution whose mean is 0 and whose standard deviation is $\sqrt{2}$. Use this to explain why finite difference filters yield poor estimates of gradients in the presence of noise.

(b) Write $\mathcal{G}$ for a 7×7 Gaussian kernel with scale 1. Form $\mathcal{M} = \mathcal{D} * \mathcal{G} * \Xi$. Show that each pixel of this image is a sample of a normal distribution whose mean is 0. What is the standard deviation of this distribution? Use this to explain why finite difference filters yield better estimates of gradients in the presence of noise when the image is smoothed.

PROGRAMMING EXERCISES

6.15. This exercise seeks N filters that are "most useful" describing a set of image windows. Collect a set of at least 500 natural images from the internet (for example, you could use the Berkeley Segmentation Dataset at `https://github.com/BIDS/BSDS500`). Reduce these images to intensity images by averaging the R, G, and B components. From each image, obtain 10 windows of size 5×5, selected at randomly chosen locations (use different random locations for different images). Straighten these windows into 25 dimensional vectors (write $\mathbf{x}_i$ for the i'th such). Compute the mean $\mathbf{m}$ of these vectors, and subtract the mean from each (write $\mathbf{z}_i = \mathbf{x}_i - \mathbf{m}$ for the i'th vector resulting).

(a) Find the filter that is "best" at describing the vectors $\mathbf{z}_i$. Define this to be a vector $\mathbf{f}$ such that (a) $\mathbf{f}^T\mathbf{f} = 1$ and (b) $\sum_i \left(\mathbf{f}^T\mathbf{z}_i\right)^2$ is maximised. Show that you can find this vector by forming

$$\mathcal{S} = \sum_i \mathbf{z}_i\mathbf{z}_i^T,$$

then finding the eigenvector with largest eigenvalue. Rearranging this vector into a filter kernel will give the filter.

(b) Now consider the collection of vectors obtained by projecting off the component in the direction of $\mathbf{f}$ from each of the original $\mathbf{z}_i$, equivalently $\mathbf{z}_i' = \mathbf{z}_i - (\mathbf{z}_i^T\mathbf{f})\mathbf{f}$. You could find the filter that is "best" at describing this new set of vectors using the procedure of the previous exercise. Show that you can find this filter by finding the eigenvector of $\mathcal{S}$ with second largest eigenvalue and rearranging the vector into a filter.

(c) Find the 10 "best" filters using the procedure above. Visualize them by arranging all 10 into a single image where 0 is shown as mid gray, negative values are darker, and positive values are lighter. You may need to scale the image to see effects. Do you see anything interesting in the appearance of the filter kernels?

(d) Repeat the exercise above for color windows and multichannel convolution. Again, do you see anything interesting in the appearance of the filter kernels?

6.16. This exercise highlights the spatial changes in noise that occur as you smooth it. Write Ξ for a noise image (i.e. the image whose i, j'th component is ξ_{ij}), where ξ_{ij} is an independent, identically distributed sample from a standard normal distribution (this has mean 0 and standard deviation 1). Make a 256×256 image of independent, identically distributed Gaussian noise. Produce a set of versions of this image smoothed with a Gaussian of $\sigma \in [1, 2, 4, 8, 16, 32, 64]$, padding as required to ensure you have a 256×256 image of valid pixels.

(a) Stack the results together and look at them. What do you see?

(b) One procedure for obtaining reasonable looking stochastic textures, known as *Perlin noise*, involves constructing a weighted combination of these noise images, then remapping the result to the range $[0, 1]$. Can you construct a set of weights so that the result looks like marble? (I used small weights for all but the largest σ, and a large weight for that).