CHAPTER 4

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.

4.1 LINEAR FILTERS AND CONVOLUTION

Section 2.3.2 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.

4.1.1 Convolution and Filtering

For the moment, think of an image as a two dimensional array of intensities. Write $\mathcal{I}_{ij}$ for the pixel at position i, j. Recall that Section 2.3.4 gave a procedure to compute a smoothed image for subsampling.

Procedure: 4.1 *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*.

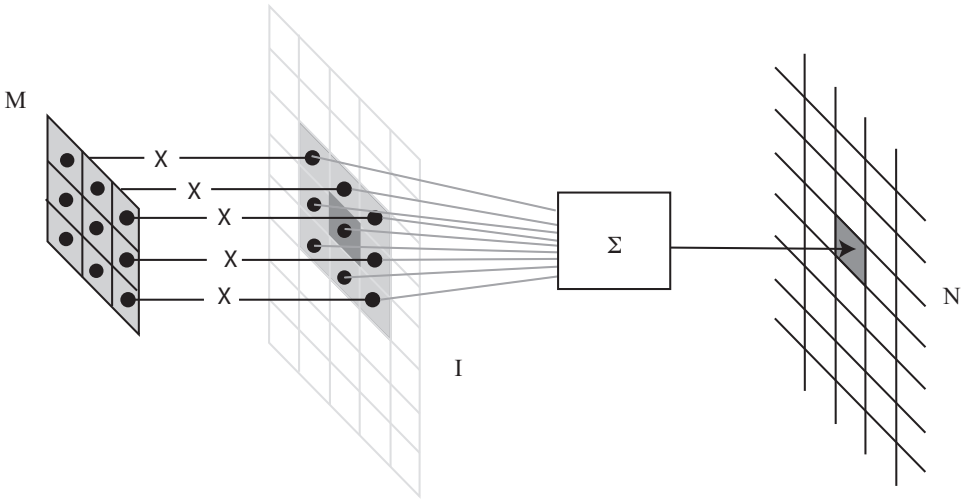

FIGURE 4.1: *To compute the value of $\mathcal{N} = \mathcal{W} * \mathcal{I}$. at some location, you shift a copy of $\mathcal{M}$ (the flipped version of $\mathcal{W}$) to lie over that location in $\mathcal{I}$; you multiply together the non-zero elements of $\mathcal{M}$ and $\mathcal{I}$ that lie on top of one another; and you sum the results. To compute the value of $\mathcal{N} = \texttt{filter}(\mathcal{I}, \mathcal{W})$ at some location, just omit flipping $\mathcal{W}$.*

Procedure: 4.2 *Filtering*

Filtering computes

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

This is sometimes written

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

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

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

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

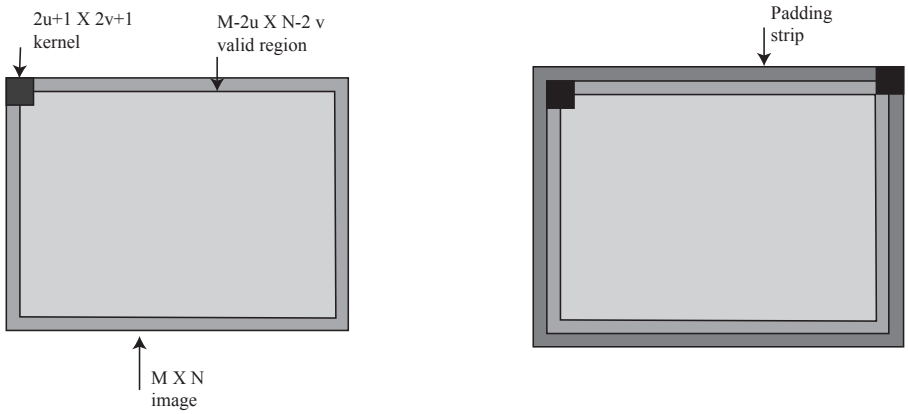

FIGURE 4.2: *To compute the value of the convolution for pixels at the edge of an image you will need to know the values of pixels outside the image. The set of valid pixels – those where you have all the image pixels you need to compute the convolution – is somewhat smaller than the original image. 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).*

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.

4.1.2 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.

4.1.3 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 4.2). 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 15.10). 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

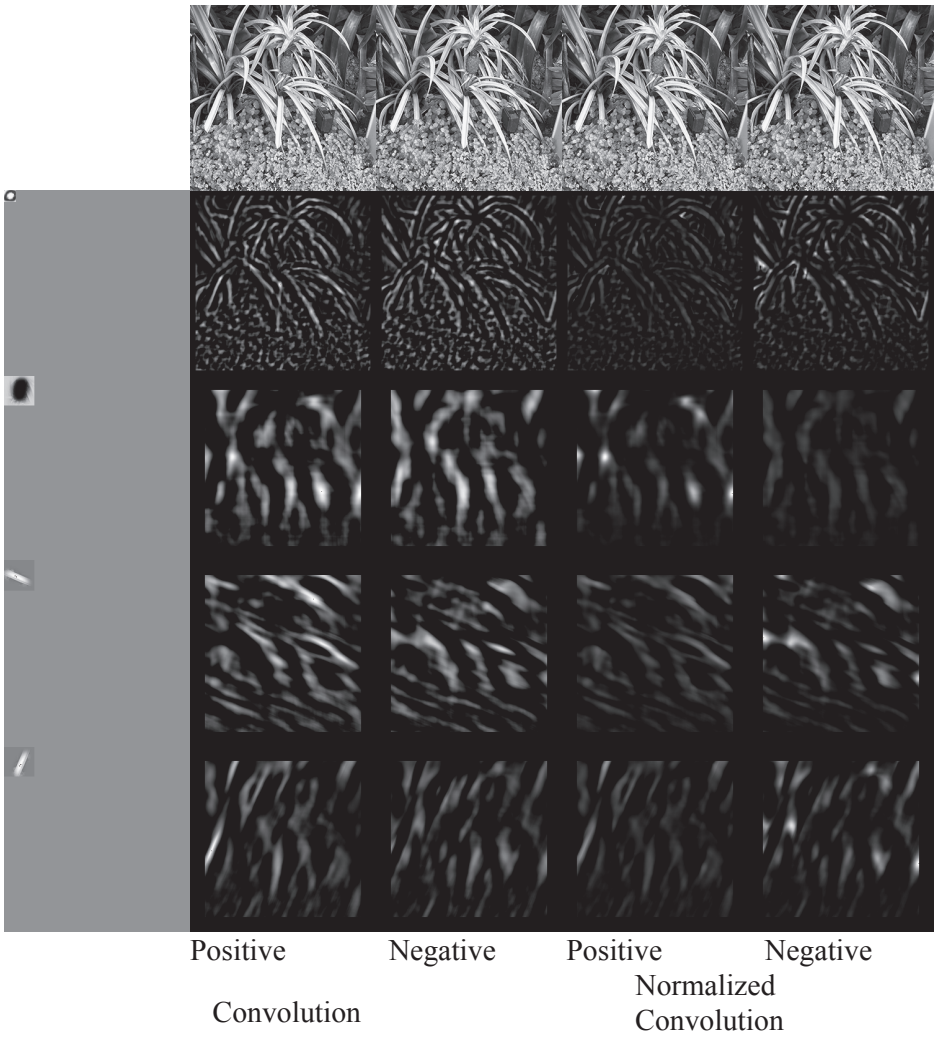

FIGURE 4.3: *Various zero-mean filters applied to a monochrome image of a pineapple plant (shown in the* **top row***, for reference), to show filters are simple pattern detectors. Details in the text.* Image credit: *Figure shows my photograph of a pineapple in the Singapore botanical garden.*

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.

4.1.4 Pattern Detection by Convolution

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 4.1). Reindex the two windows to be vectors, and the result is a dot product. This view explains why a convolution is interesting: it can be used as a very simple pattern detector.

Recall the properties of a dot product. Write $\mathbf{u}$ and $\mathbf{v}$ for two vectors, and $\mathbf{1}$ for a vector of ones. Then $\mathbf{u}^T(\mathbf{v} + c\mathbf{1}) = \mathbf{u}^T\mathbf{v} + c\mathbf{u}^T\mathbf{1}$. Interpreting $\mathbf{u}$ as the vector of kernel weights and $\mathbf{v}$ as the vector of image values suggests that using a filter with zero mean is a good idea. For a zero-mean filter, $\mathbf{u}^T\mathbf{1} = 0$, which means the value of the filter can't be changed by adding a constant to the image.

When you add a constant to the image, you want the smoothed image to change, so a zero-mean filter is no use for smoothing. It is very useful for pattern detection. Recall that the dot-product of two unit vectors $\mathbf{u}$ and $\mathbf{v}$ is largest when they are the same, and smallest when $\mathbf{u} = -\mathbf{v}$. This means that the zero-mean filter that gives the most positive response to an image pattern looks like that pattern, and the zero-mean filter that gives the most negative response looks like that pattern, but with contrast reversed (i.e. light swapped to dark and dark swapped to light).

> **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*

4.1.5 Normalized Convolution

If the mean of the kernel is zero, scaling the image will scale the value of the convolution. One strategy to build a somewhat better pattern detector is to normalize the result of the convolution to obtain a value that is unaffected by scaling the image. For $\mathcal{W}$ a zero mean kernel, $\mathcal{G}$ a gaussian kernel, and ϵ a small positive number compute

$$\frac{\mathcal{W} * \mathcal{I}}{\mathcal{G} * \mathcal{I} + \epsilon}.$$

Here the division is element by element, ϵ is used to avoid dividing by zero, and $\mathcal{G} * \mathcal{I}$ is an estimate of how bright the image is. This strategy, known as *normalized convolution* produces an improvement in the detector. Figure 4.3 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 15.10).

Useful Fact: *Normalized convolution divides by an estimate of intensity to produce a better pattern detector.*

4.1.6 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 4.3 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 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 and negative responses to produce a better pattern detector.*

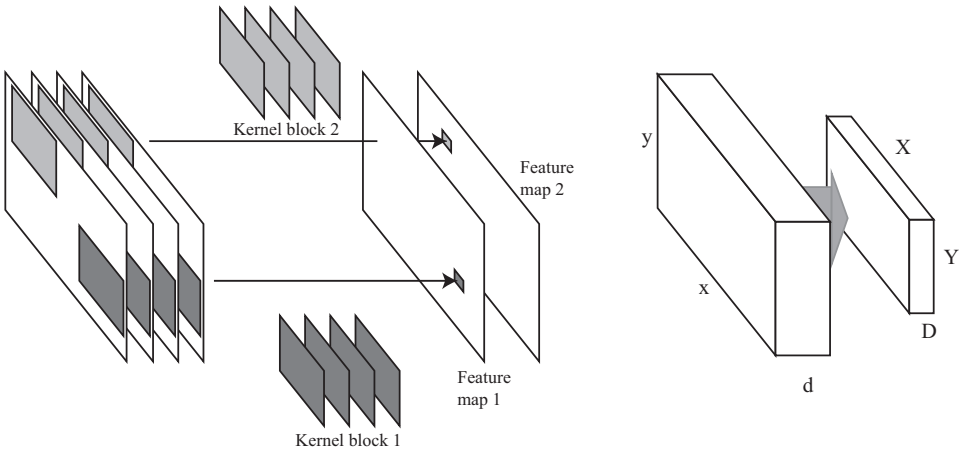

FIGURE 4.4: *On the* **left**, *two kernels (now 3D, as in the text) applied to a set of feature maps produce one new feature map per kernel, using the procedure of the text (the bias term isn't shown). Abstract this as a process that takes an* $x \times y \times d$ *block to an* $X \times Y \times D$ *block (as on the* **right**).

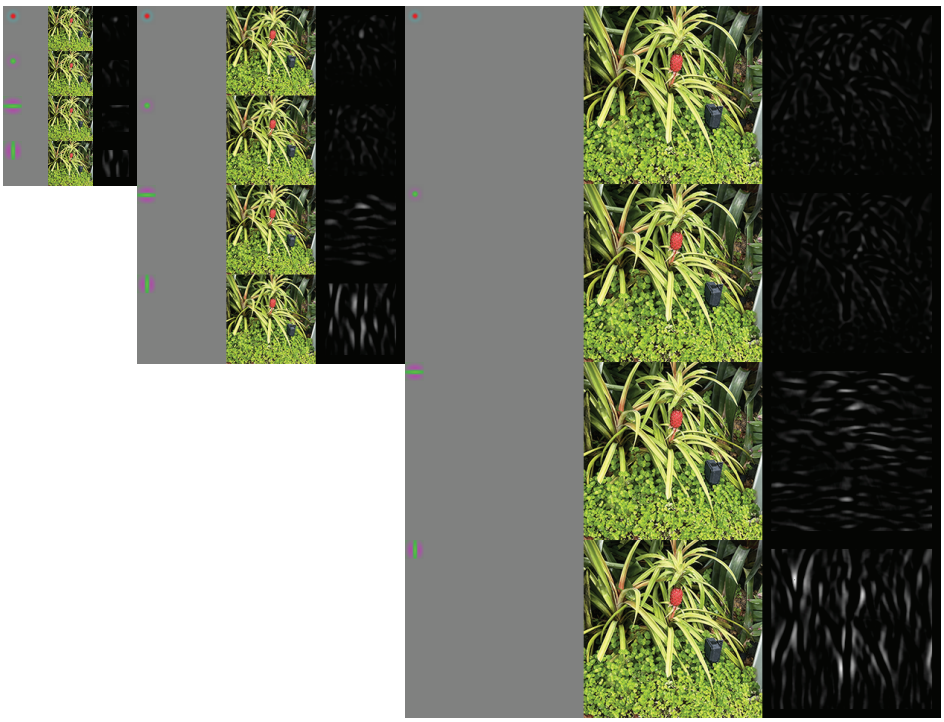

FIGURE 4.5: *Multichannel convolution easily yields a simple detector for colored patterns.* Image credit: *Figure shows my photograph of a pineapple in the Singapore botanical garden.*

4.1.7 Multi-Channel Convolution

The description of convolution anticipates monochrome images, and Figure 4.3 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 4.3 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 $\mathcal{N} = \mathcal{I} * \mathcal{K}$ 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 4.5). 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 $R \times U \times V$.*

4.2 YOU SHOULD

4.2.1 remember these facts:

Convolution and filtering are crucial building blocks. 53
Matrices: Trace 422
Matrices: Frobenius norm 423

4.2.2 remember these procedures:

Convolution 44
Filtering 45
RQ Factorization 423
K-Means Clustering 440

4.2.3 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.