Filters are dot products

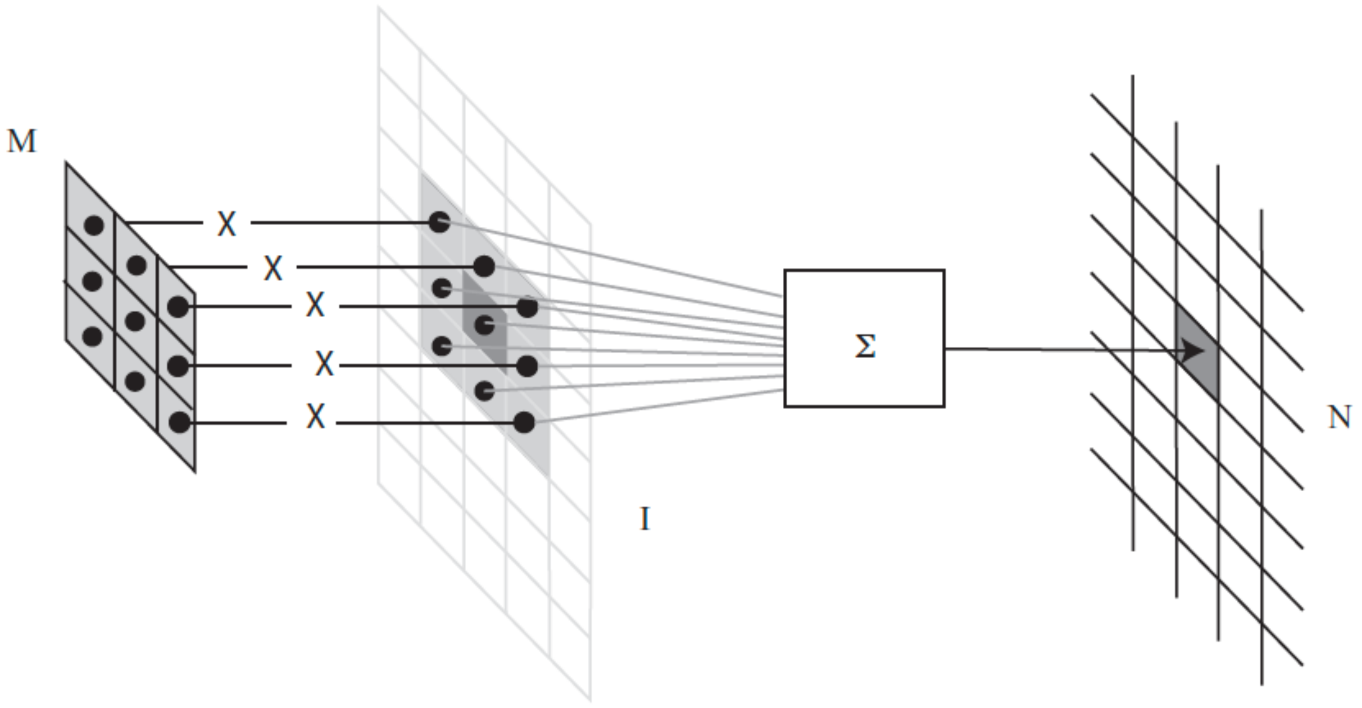


FIGURE 3.1: *To compute the value of $\mathcal{N}$ at some location, you shift a copy of $\mathcal{M}$ (the flipped version of $\mathcal{W}$) to lie over that location in $\mathcal{I}$; you multiply together the non-zero elements of $\mathcal{M}$ and $\mathcal{I}$ that lie on top of one another; and you sum the results.*

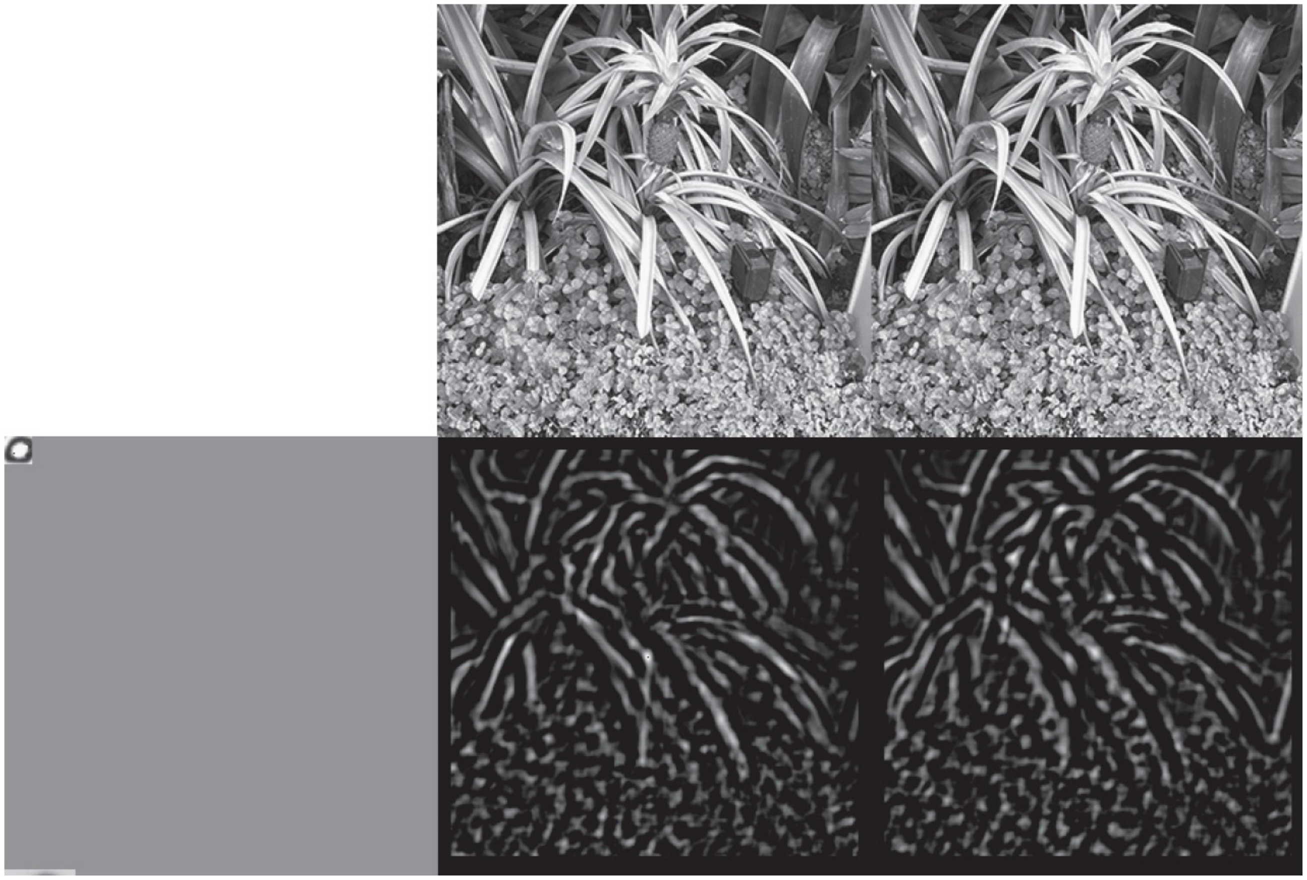
ReLUs

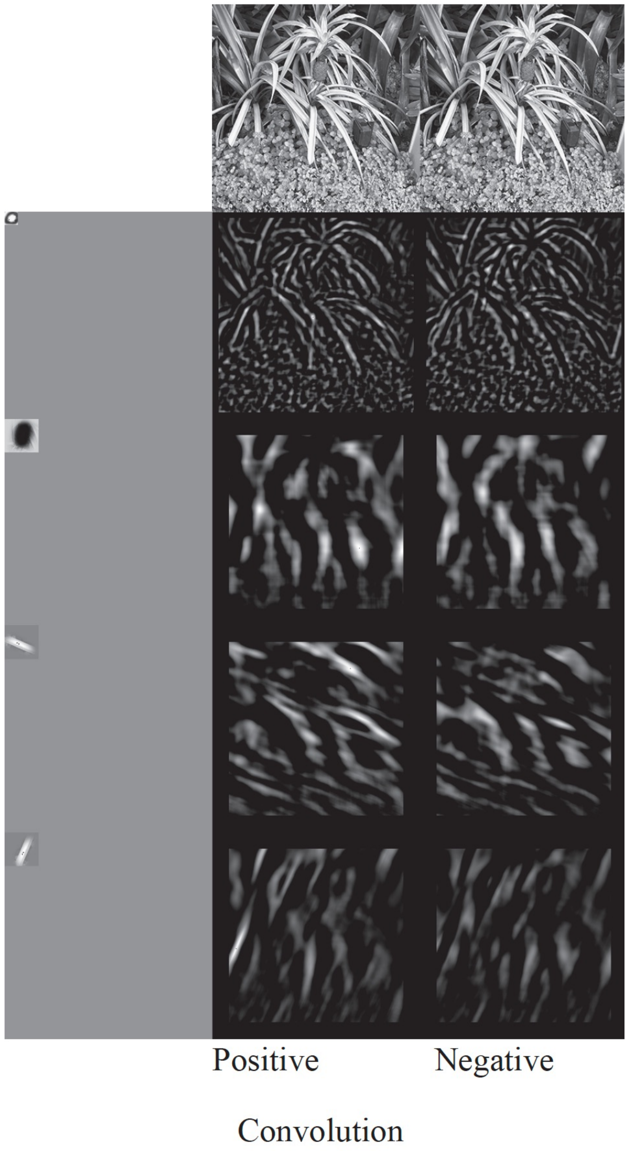
Write $\mathcal{W}$ for a kernel representing some pattern you wish to find. Assume that $\mathcal{W}$ has zero mean, so that the filter gives zero response to a constant image. Notice that $\mathcal{N} = \mathcal{W} * \mathcal{I}$ is strongly positive at locations where $\mathcal{I}$ looks like $\mathcal{W}$, 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

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

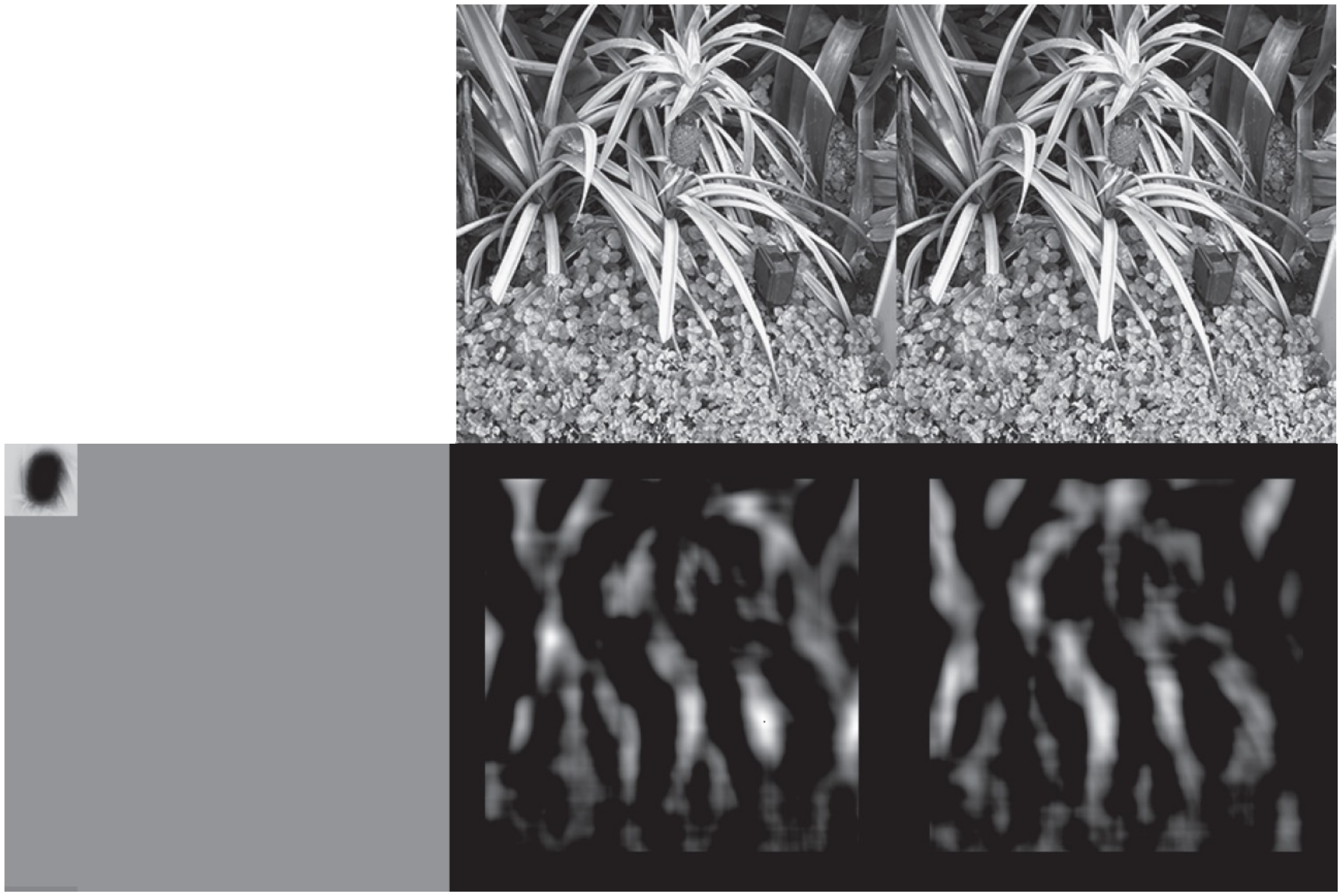
(often called a *Rectified Linear Unit* or more usually *ReLU*). Then $\texttt{relu}\mathcal{W} * \mathcal{I}$ is a measure of how well $\mathcal{W}$ matches $\mathcal{I}$ at each pixel, and $\texttt{relu}-\mathcal{W} * \mathcal{I}$ is a measure of how well $\mathcal{W}$ matches a contrast reversed $\mathcal{I}$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

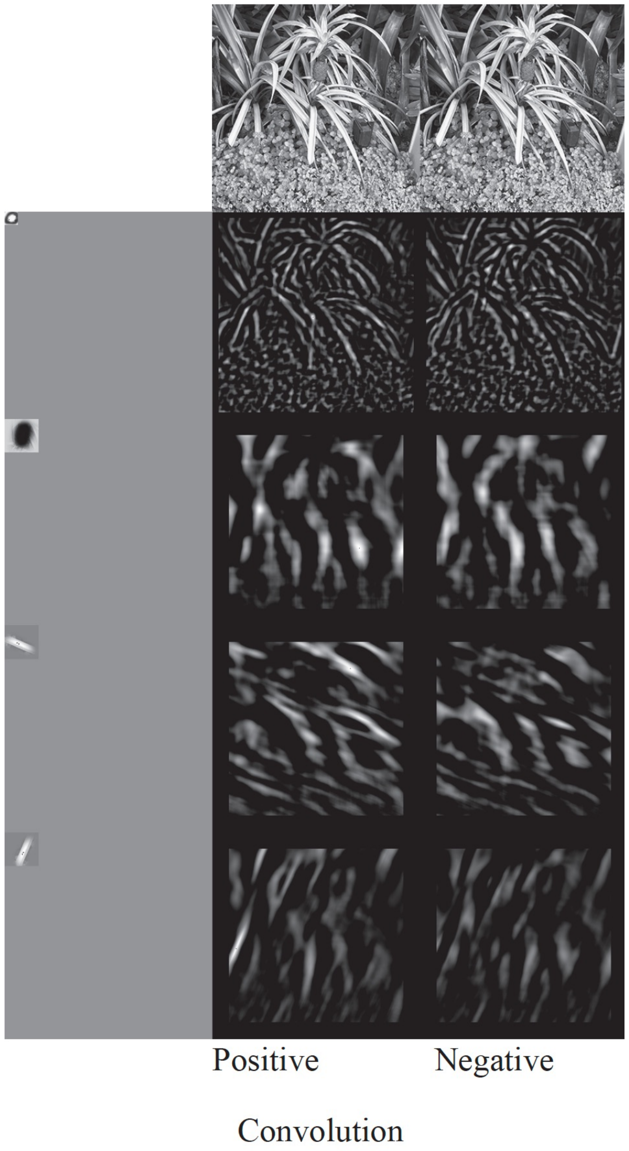
Filters detect patterns





Filters detect patterns



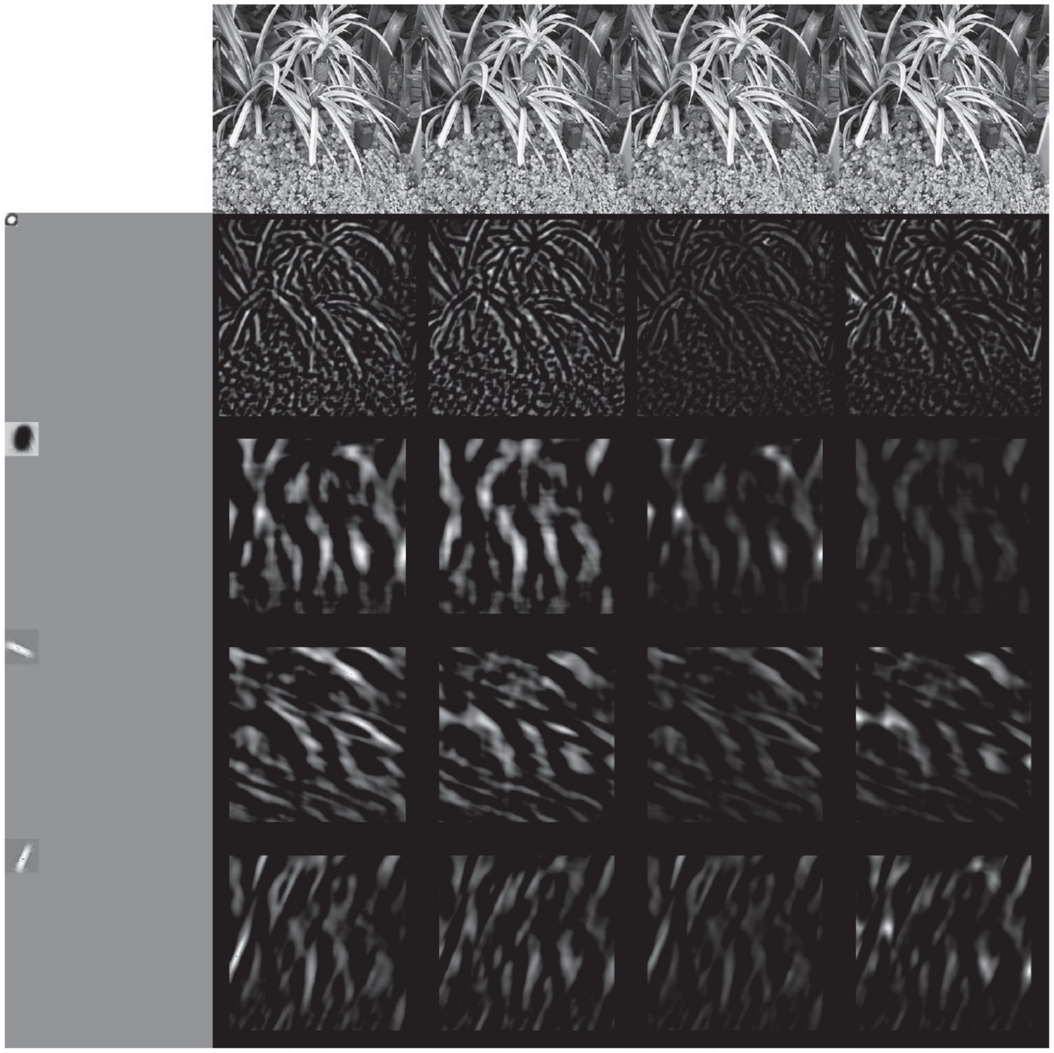


The dot-product analogy reveals some reasons that convolution is not a particularly good pattern detector. Assume that the mean of the kernel is not zero. In this case, adding a constant offset to the image will change the value of the convolution, so you cannot rely on the value. This can be dealt with by subtracting the mean from the kernel.

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


Positive Negative Positive Negative

Convolution

Normalized Convolution

Applications: Gradient estimates

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

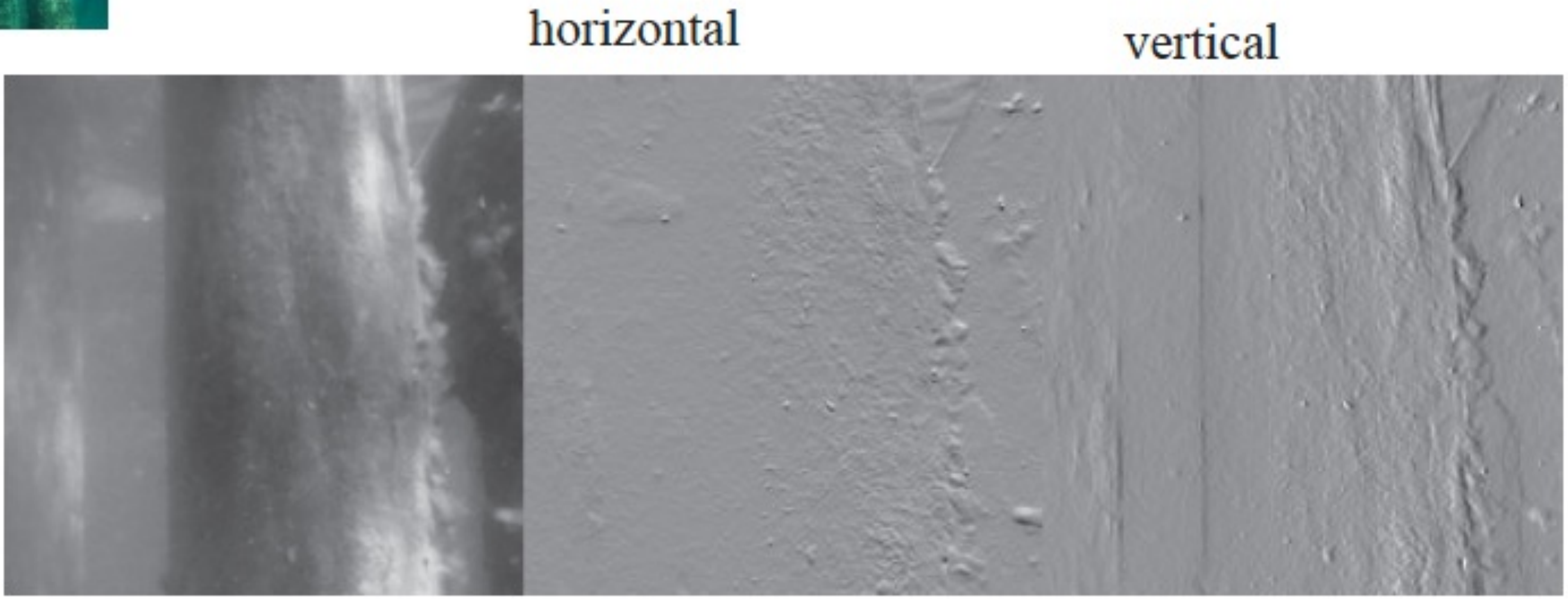
$$\frac{\partial \mathcal{I}}{\partial x} = \lim_{\delta x \to 0} \frac{\mathcal{I}(x + \delta x, y) - \mathcal{I}(x, y)}{\delta x} \approx \mathcal{I}_{i+1,j} - \mathcal{I}_{i,j}.$$

This means a convolution with

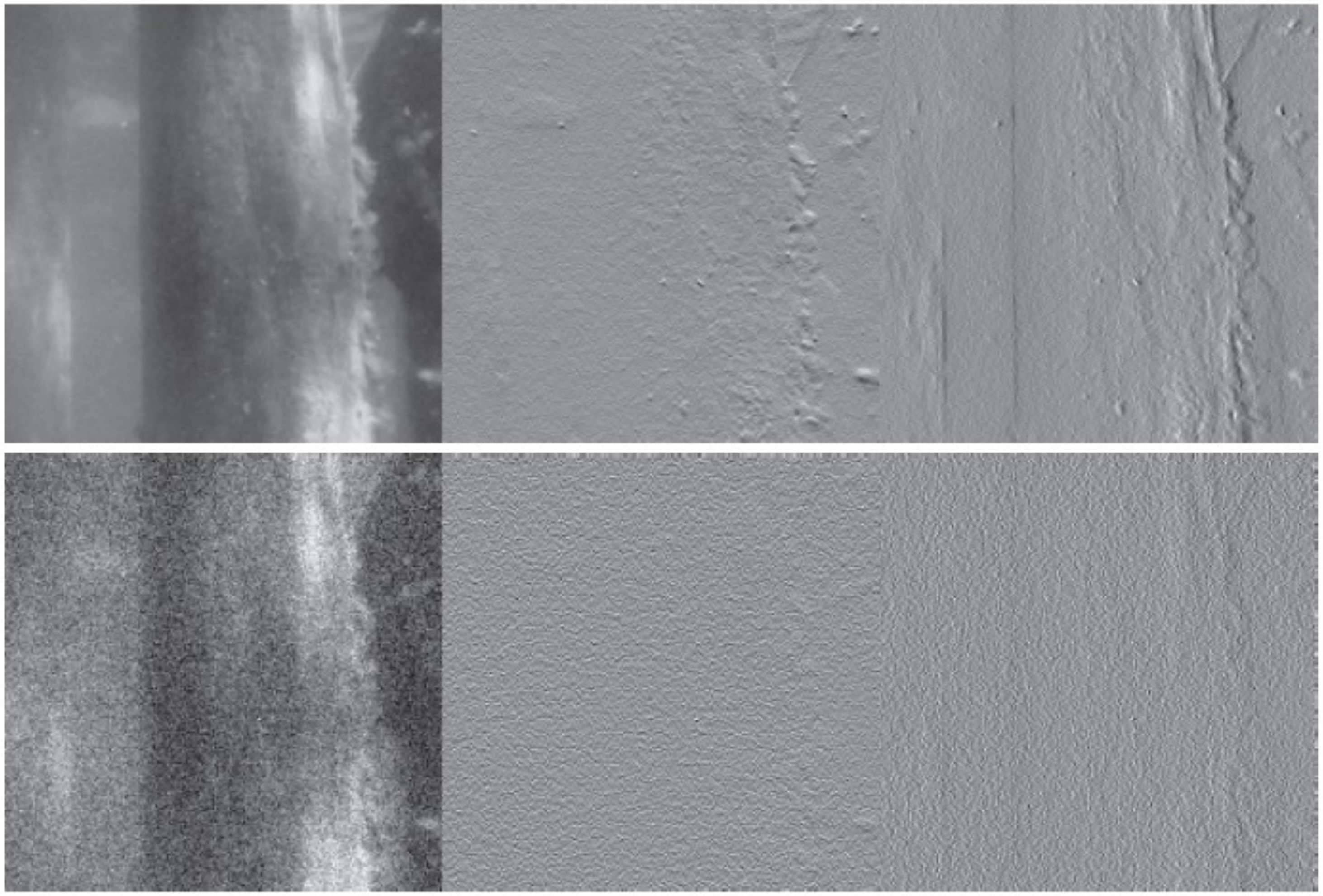
$$-1 \quad 1$$

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*.

Image derivatives with finite differences



Finite differences are overexcited by noise



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).

Multi-channel Convolution

For a color image $\mathcal{I}$, write $\mathcal{I}_{k,ij}$ 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

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

Multi-channel convolution

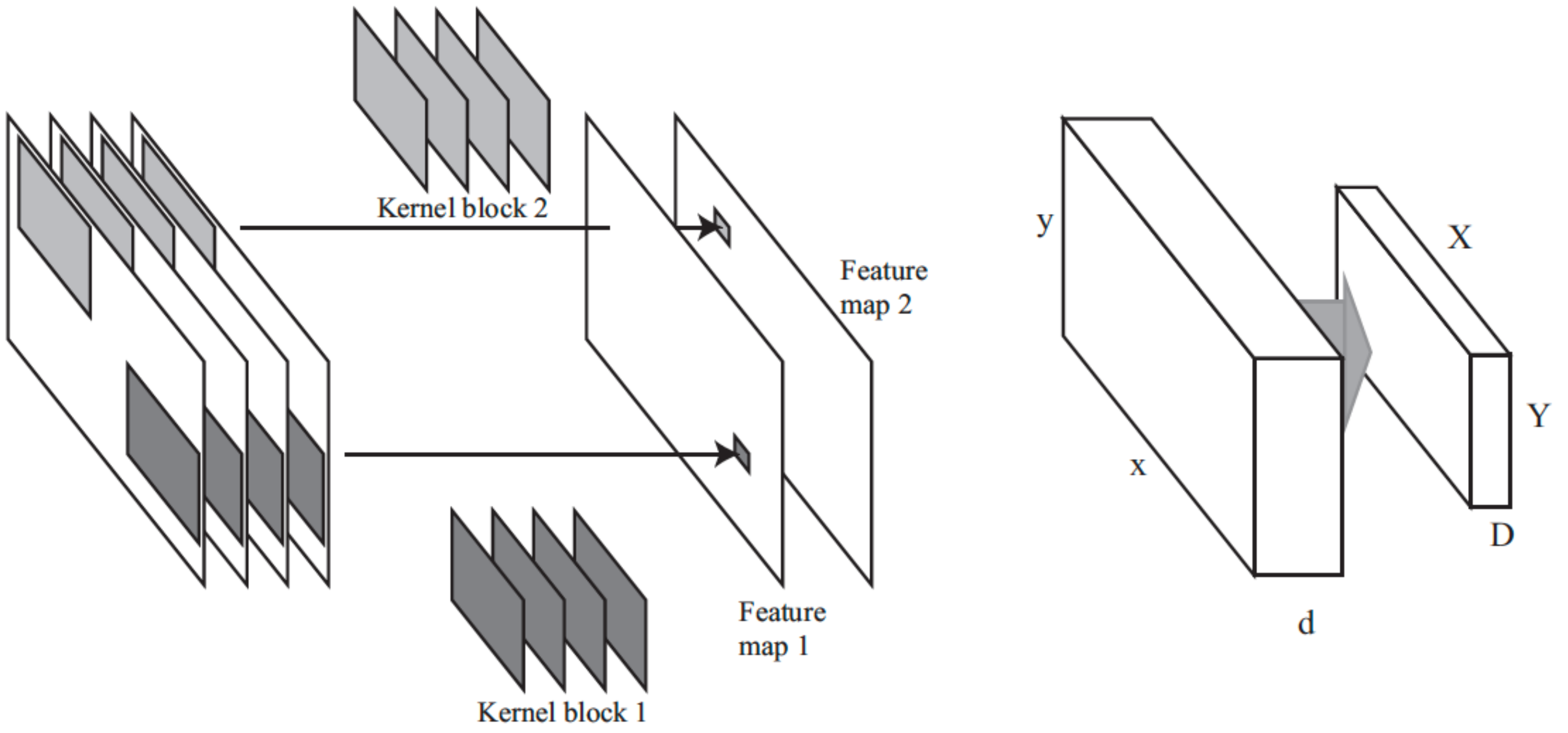
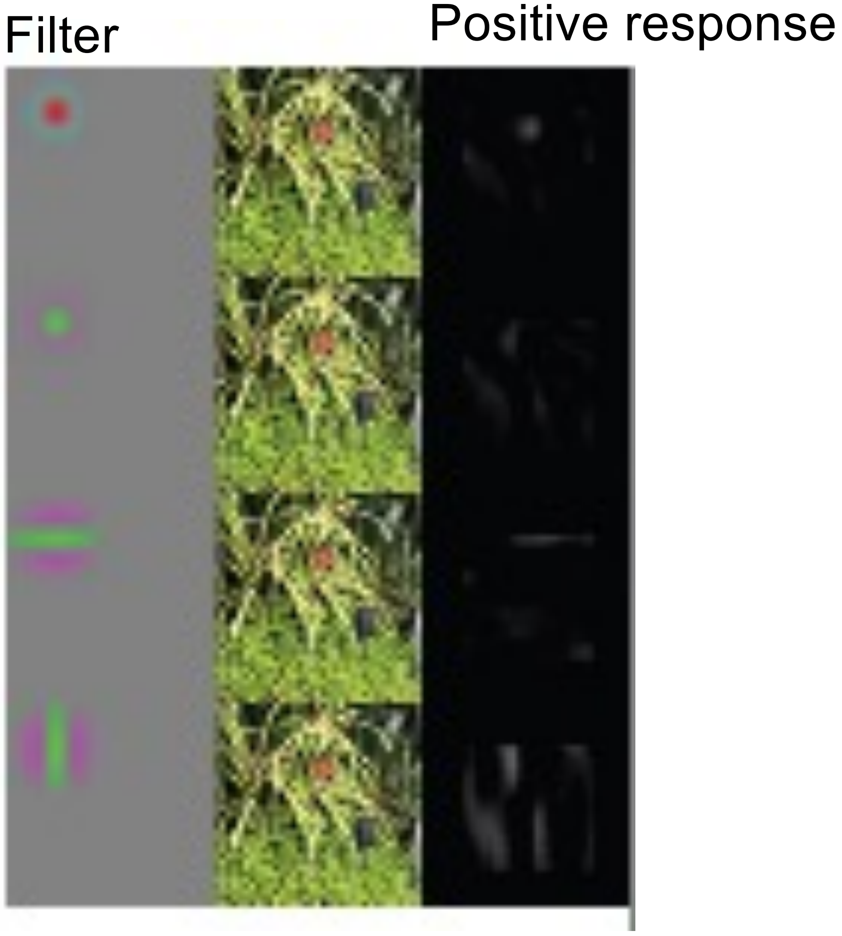
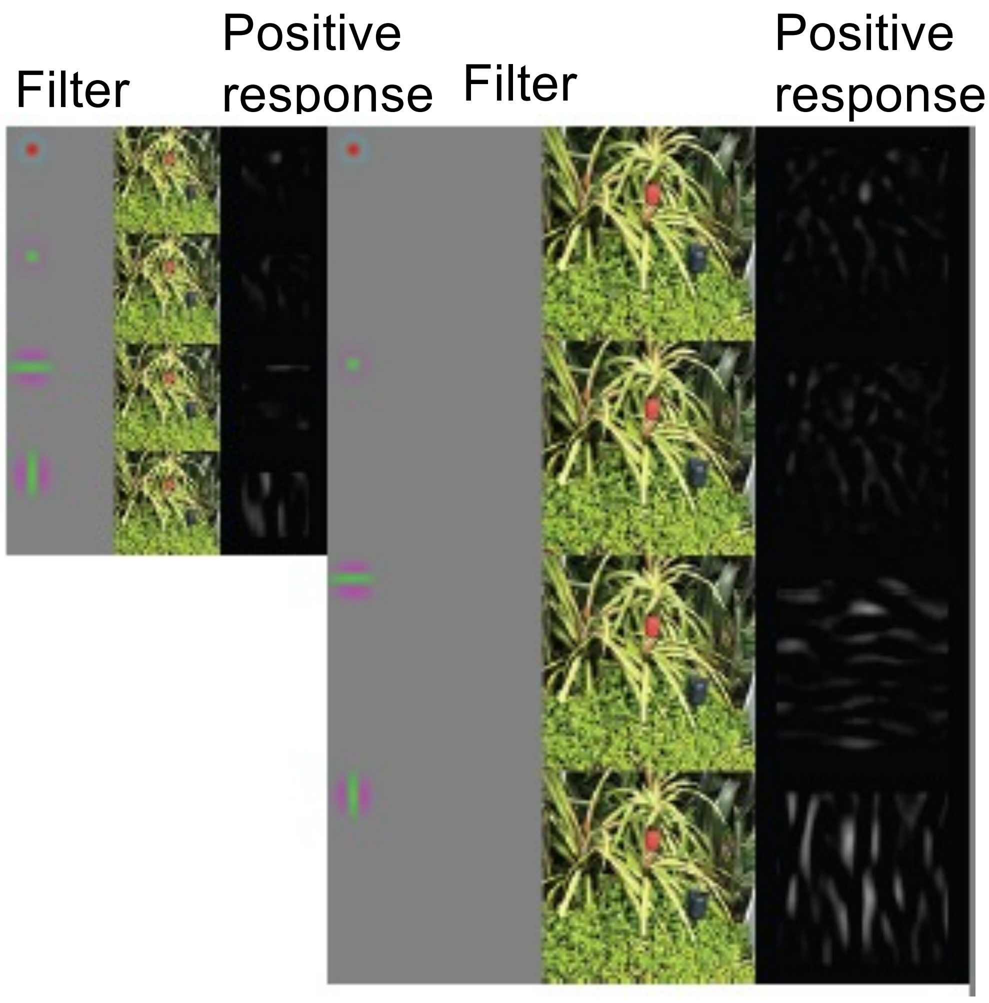


FIGURE 4.4: *On the* **left**, *two kernels (now 3D, as in the text) applied to a set of feature maps produce one new feature map per kernel, using the procedure of the text (the bias term isn't shown). Abstract this as a process that takes an $x \times y \times d$ block to an $X \times Y \times D$ block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

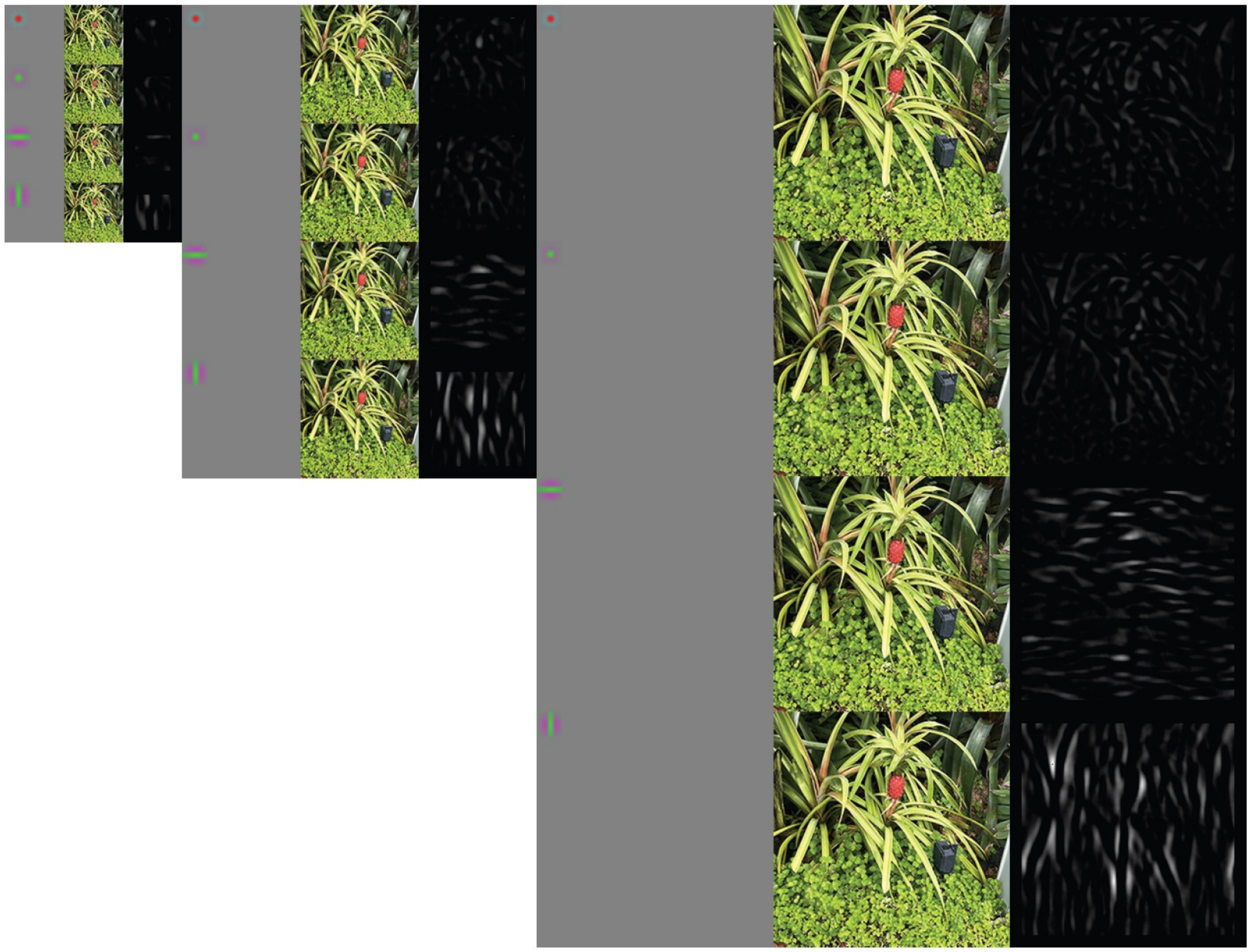
Representing Images with Filter Banks



Representing Images with Filter Banks at scales



Representing Images with Filter Banks at scales



But which filters should I use?

Up till about 2012:

- choose some, mostly spots and bars

After 2012:

- lots; choose ones that work well in your application
  using an optimization procedure