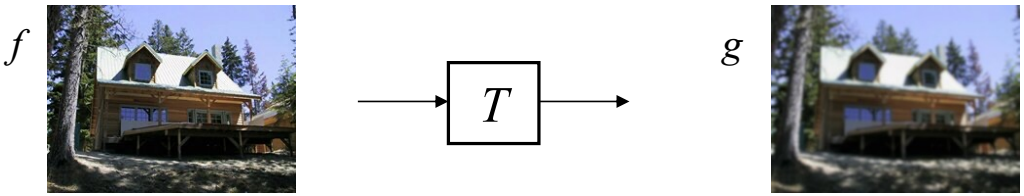
Image filtering

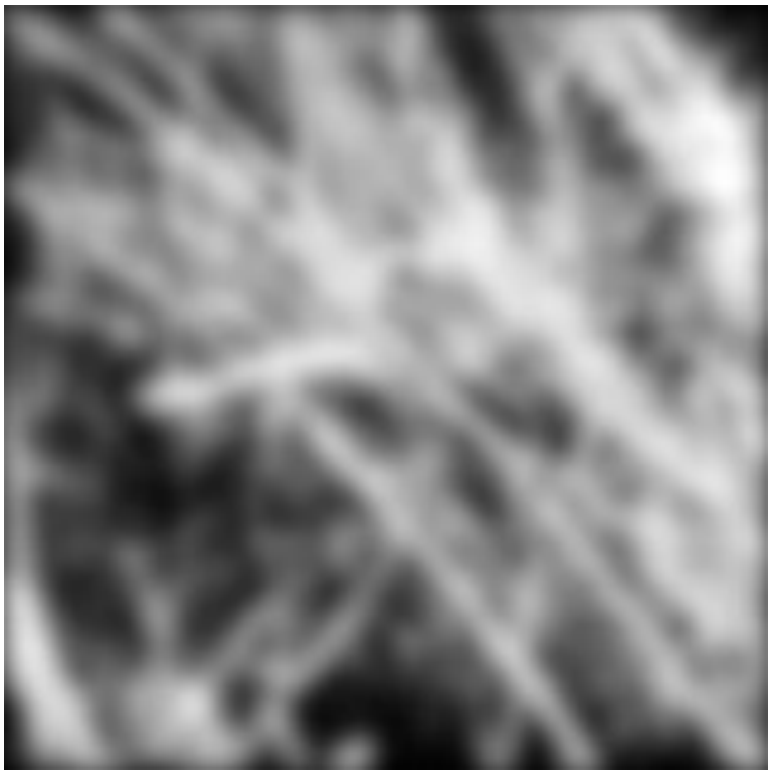
- Roughly speaking, replace image value at x with some function of values in its spatial neighborhood $N(x)$:

$$g(x) = T(f(N(x)))$$



- Examples: smoothing, sharpening, edge detection, etc.

Image filtering



Recall: Image transformations

- What are different kinds of image transformations?
  - Range transformations or point processing
  - Image warping
  - Image filtering

Image filtering: Outline

- Linear filtering and its properties
- Gaussian filters and their properties
- Nonlinear filtering: Median filtering
- Fun filtering application: Hybrid images

Sliding window operations

- Let's slide a fixed-size window over the image and perform the same simple computation at each window location
- Example use case: how do we reduce image noise?
  - Let's take the *average* of pixel values in each window
  - More generally, we can take a *weighted sum* where the weights are given by a *filter kernel*

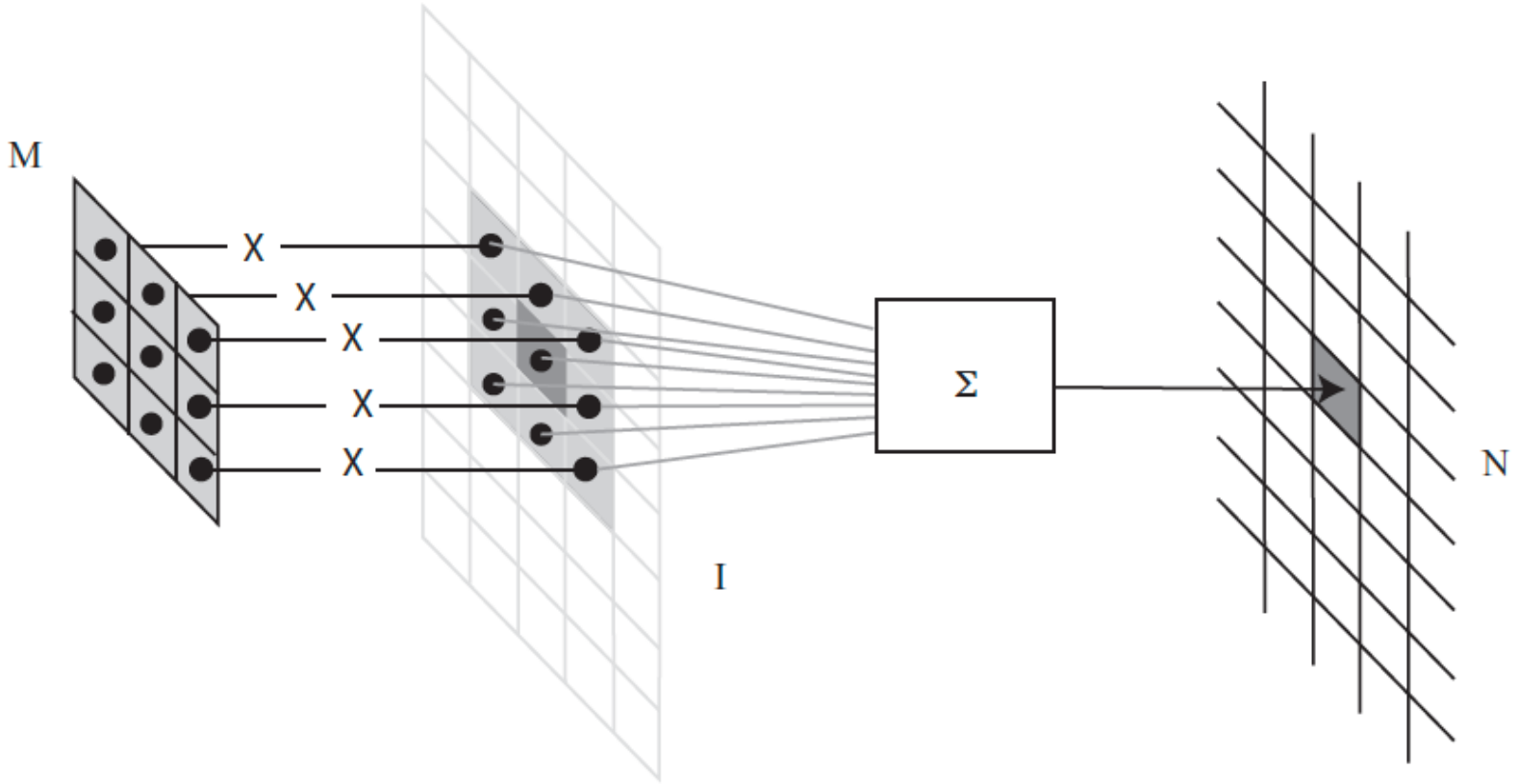
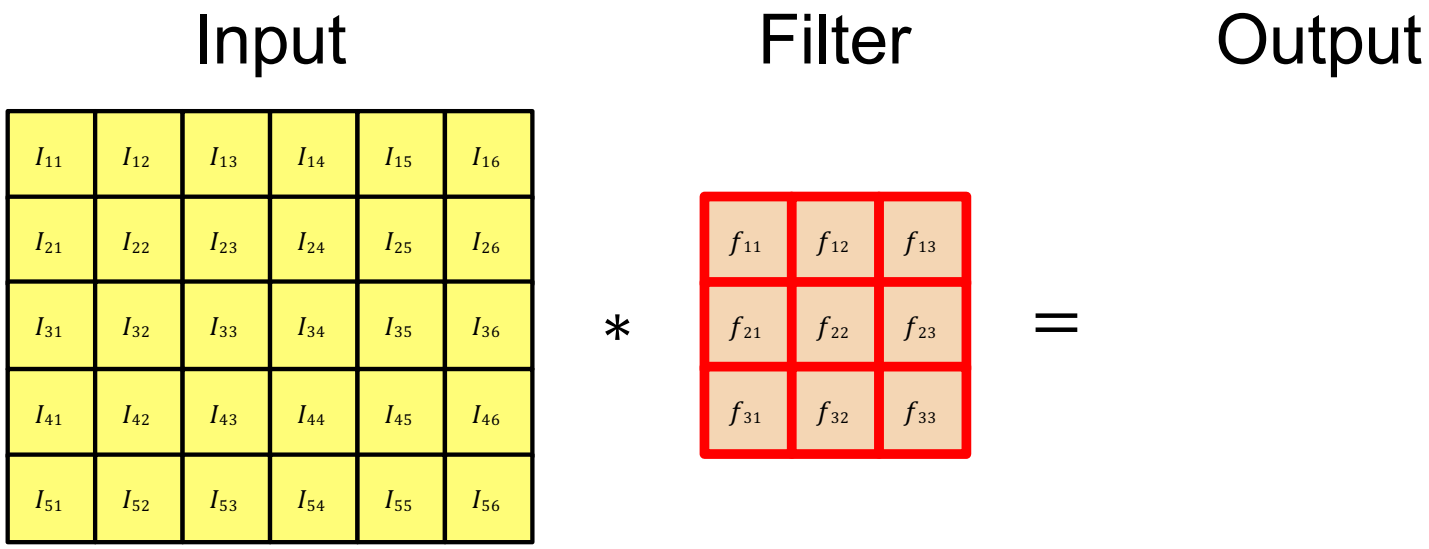


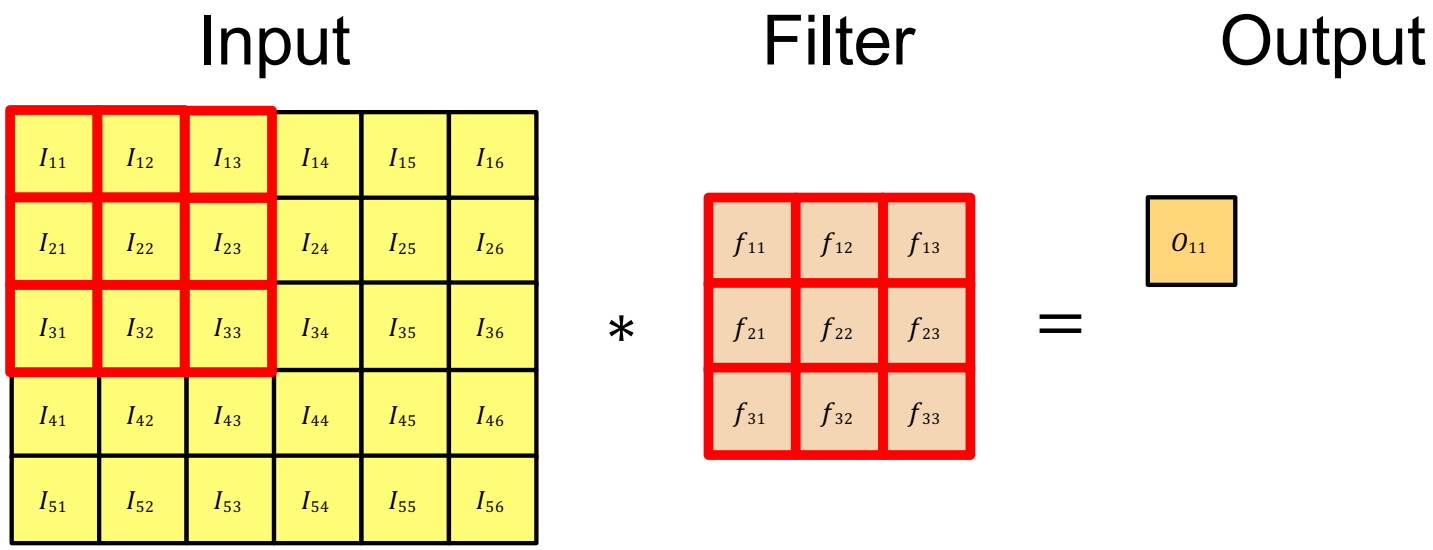
FIGURE 3.1: *To compute the value of $\mathcal{N}$ at some location, you shift a copy of $\mathcal{M}$ (the flipped version of $\mathcal{W}$) to lie over that location in $\mathcal{I}$; you multiply together the non-zero elements of $\mathcal{M}$ and $\mathcal{I}$ that lie on top of one another; and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

Applying a linear filter



Adapted from D. Fouhey and J. Johnson

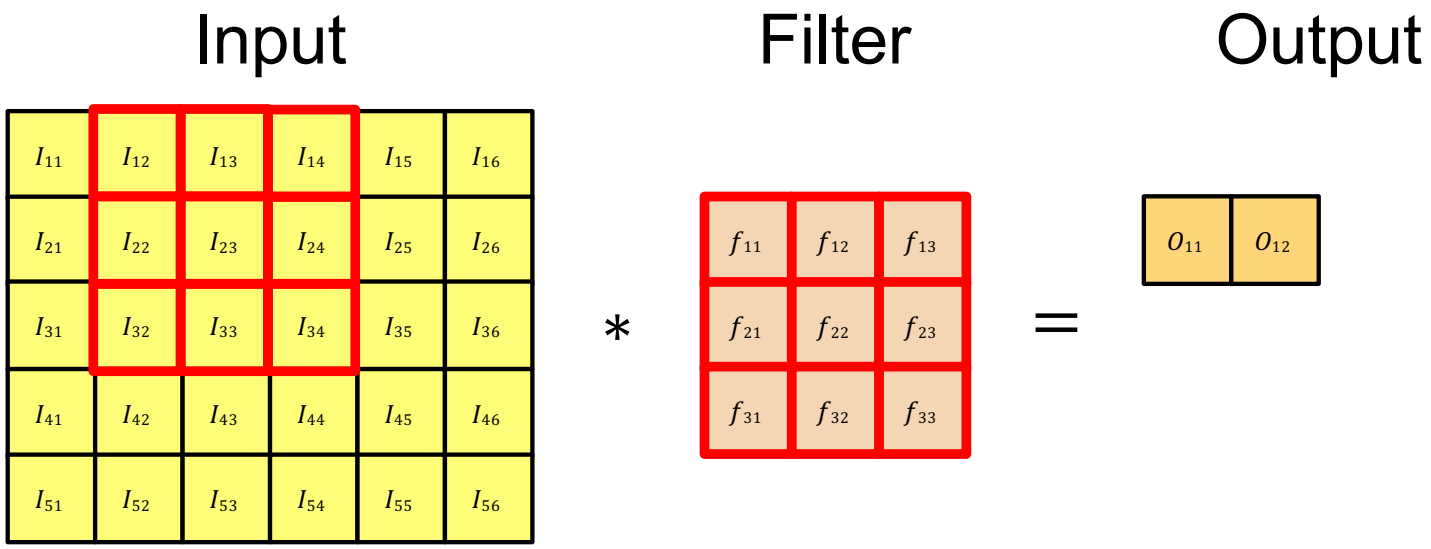
Applying a linear filter



$$O_{11} = I_{11} \cdot f_{11} + I_{12} \cdot f_{12} + I_{13} \cdot f_{13} + \ldots + I_{33} \cdot f_{33}$$

Adapted from D. Fouhey and J. Johnson

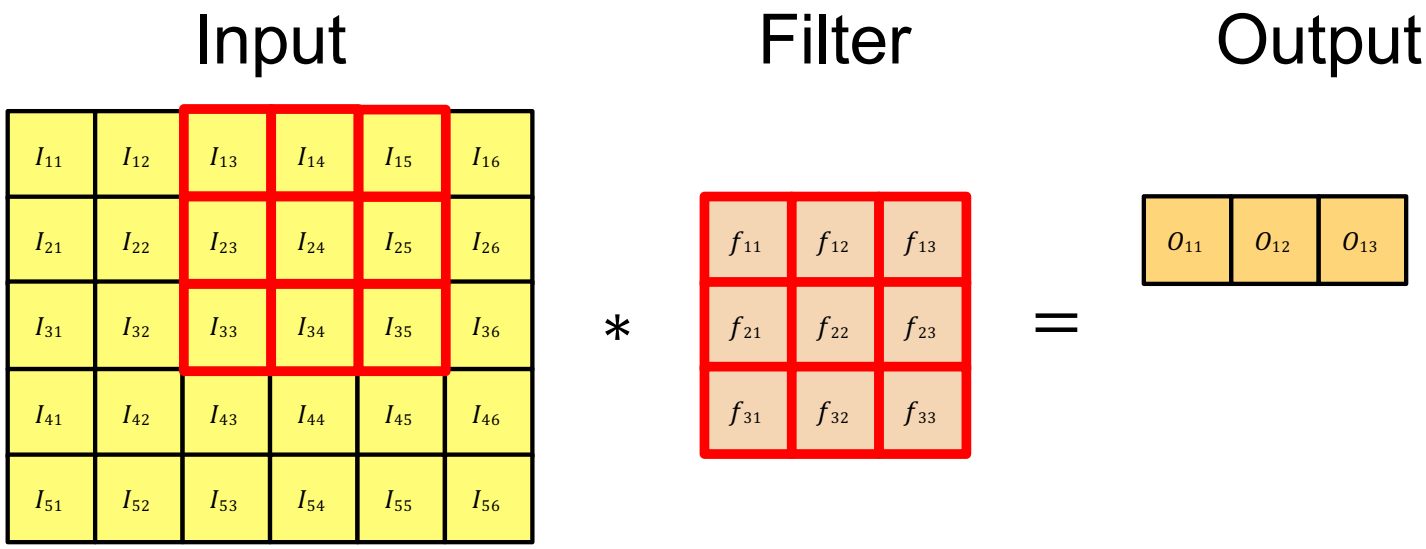
Applying a linear filter



$$O_{12} = I_{12} \cdot f_{11} + I_{13} \cdot f_{12} + I_{14} \cdot f_{13} + \ldots + I_{34} \cdot f_{33}$$

Adapted from D. Fouhey and J. Johnson

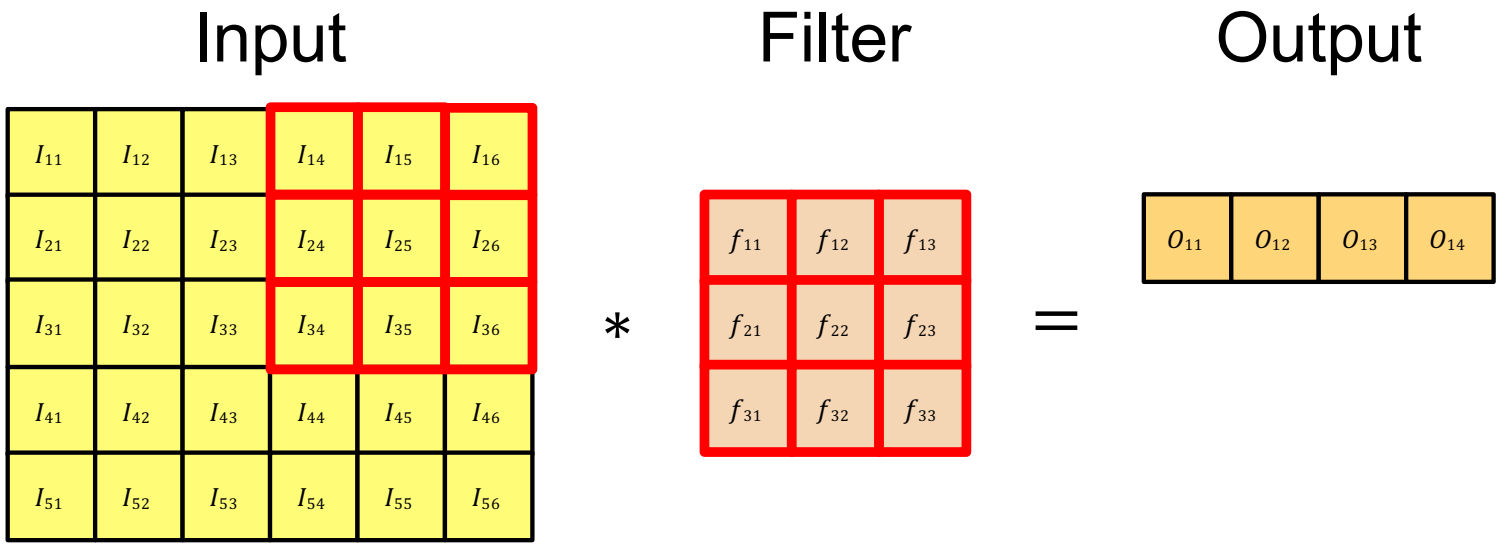
Applying a linear filter



$$O_{13} = I_{13} \cdot f_{11} + I_{14} \cdot f_{12} + I_{15} \cdot f_{13} + \ldots + I_{35} \cdot f_{33}$$

Adapted from D. Fouhey and J. Johnson

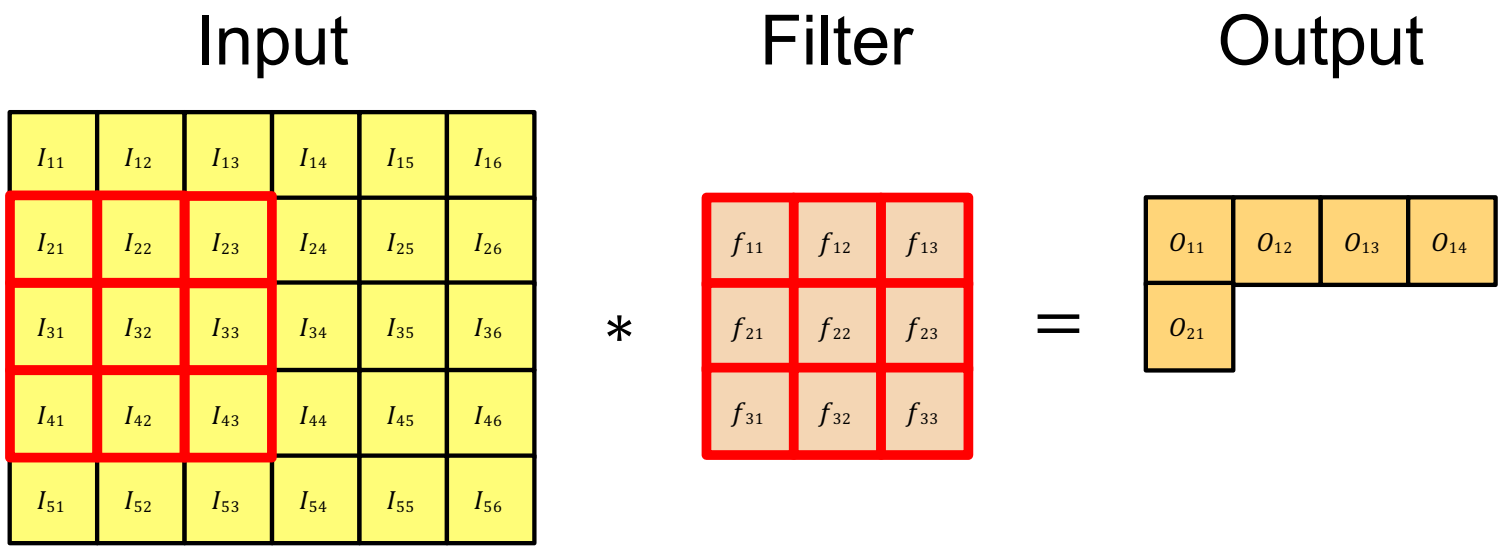
Applying a linear filter



$$O_{14} = I_{14} \cdot f_{11} + I_{15} \cdot f_{12} + I_{16} \cdot f_{13} + \ldots + I_{36} \cdot f_{33}$$

Adapted from D. Fouhey and J. Johnson

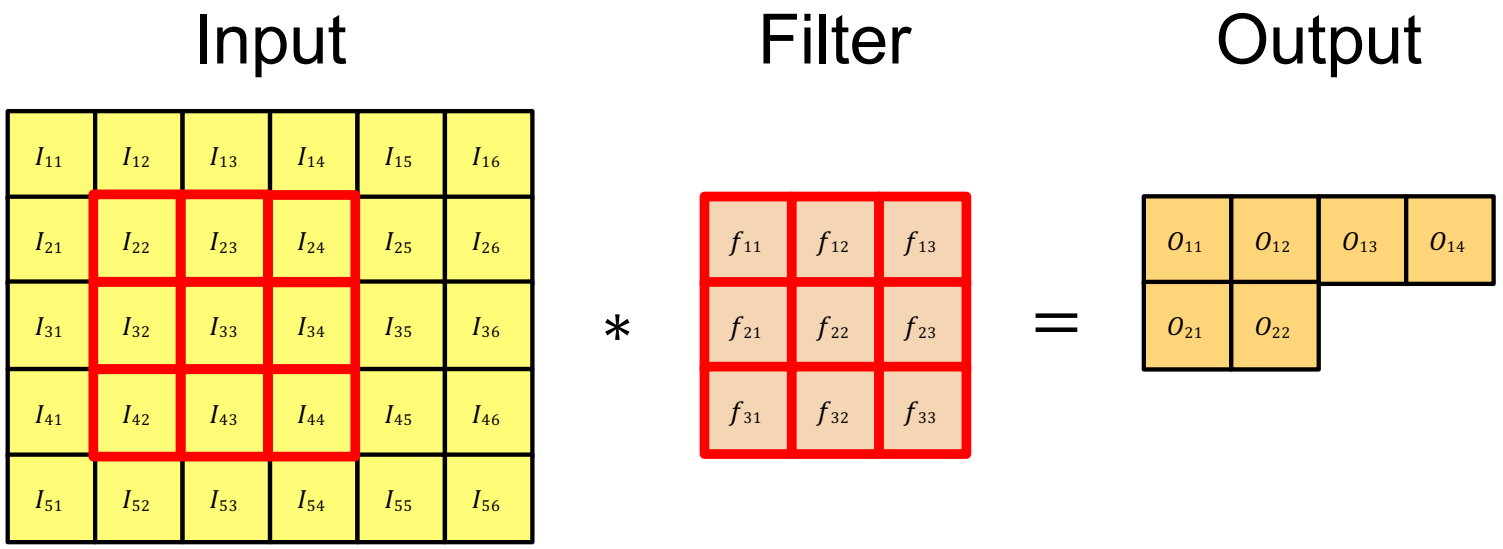
Applying a linear filter



$$O_{21} = I_{21} \cdot f_{11} + I_{22} \cdot f_{12} + I_{23} \cdot f_{13} + \ldots + I_{43} \cdot f_{33}$$

Adapted from D. Fouhey and J. Johnson

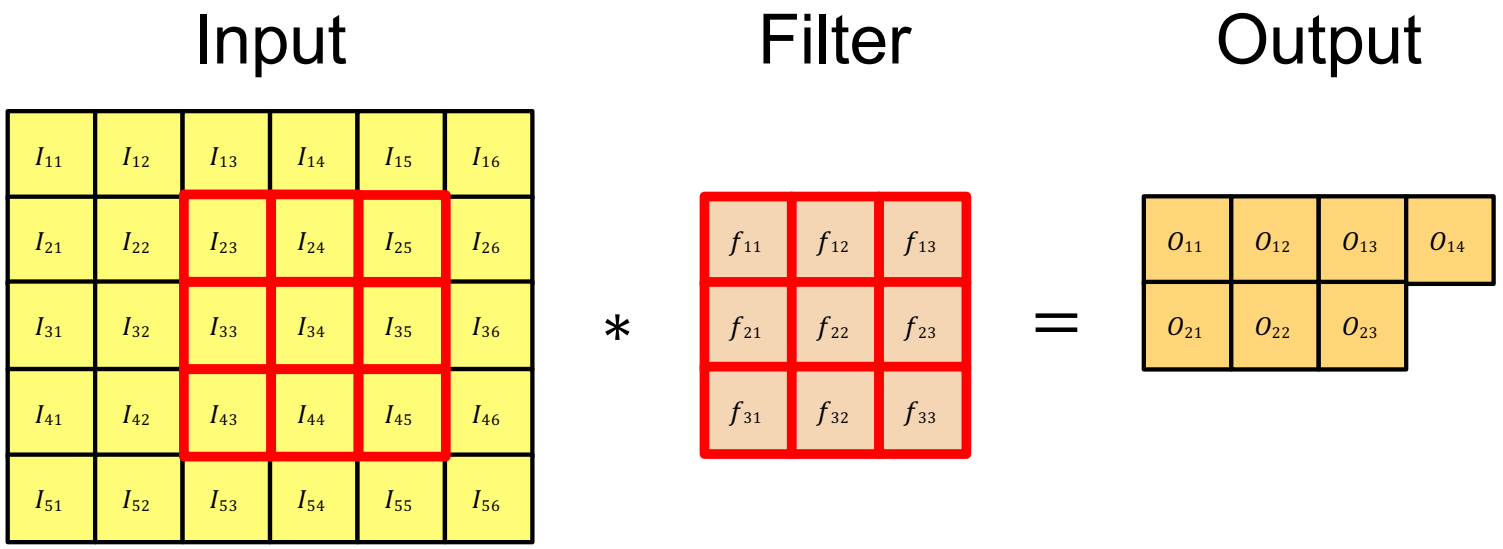
Applying a linear filter



$$O_{22} = I_{22} \cdot f_{11} + I_{23} \cdot f_{12} + I_{24} \cdot f_{13} + \ldots + I_{44} \cdot f_{33}$$

Adapted from D. Fouhey and J. Johnson

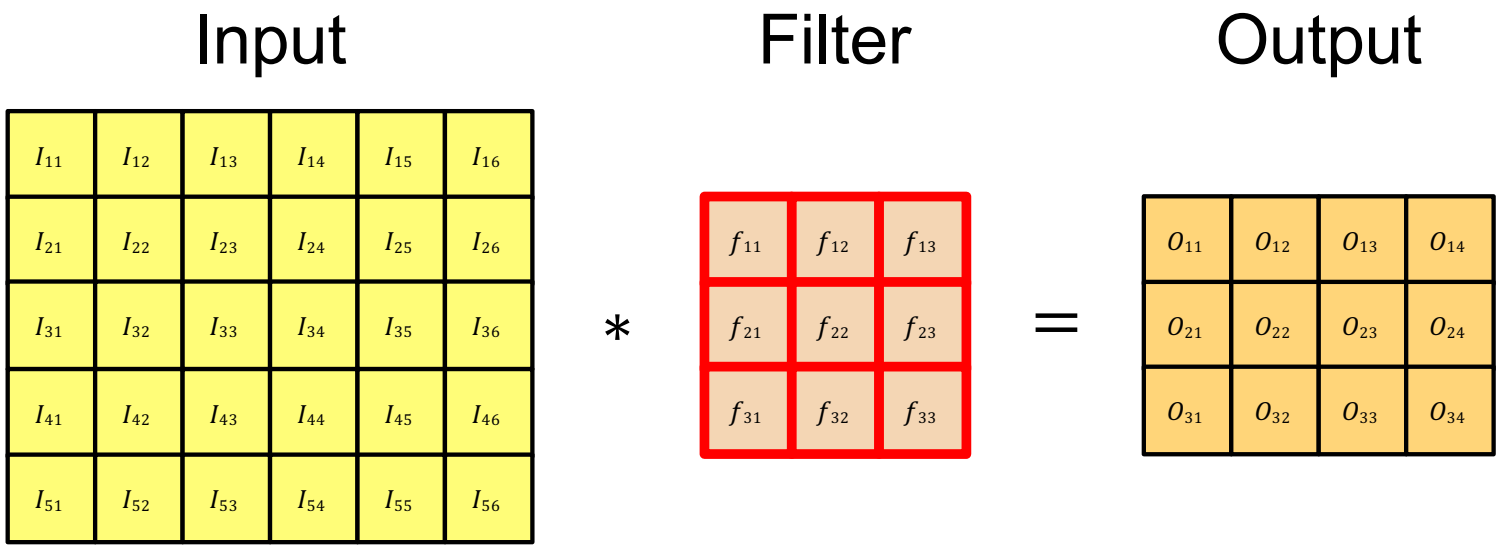
Applying a linear filter



$$O_{23} = I_{23} \cdot f_{11} + I_{24} \cdot f_{12} + I_{25} \cdot f_{13} + \ldots + I_{45} \cdot f_{33}$$

Adapted from D. Fouhey and J. Johnson

Applying a linear filter



What filter values should we use to find the average in a 3×3 window?

Adapted from D. Fouhey and J. Johnson

Convolution

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. We will construct a small array (a *mask* or *kernel*) $\mathcal{W}$, and compute a new image $\mathcal{N}$ from the image and the mask, using the rule

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

which we will write

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

In some sources, you might see $\mathcal{W} ** \mathcal{I}$ (to emphasize the fact that the image is 2D). We sum over all u and v that apply to $\mathcal{W}$; for the moment, do not worry about what happens when an index goes out of the range of $\mathcal{I}$. This operation is known as *convolution*, and $\mathcal{W}$ is often called the *kernel* of the convolution. You should

Filtering

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). What you might expect – sometimes called *correlation* or *filtering* – would compute

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

which we will write

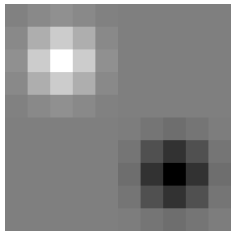
$$\mathcal{N} = \texttt{filter}(\mathcal{I}, \mathcal{W}).$$

This difference isn't particularly significant, but if you forget that it is there, you compute the wrong answer.

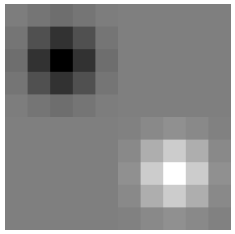
Note: Filtering vs. “convolution”

- In classical signal processing terminology, convolution is filtering with a *flipped* kernel, and filtering with an upright kernel is known as *cross-correlation*
  - Check convention of filtering function you plan to use!

Filtering or “cross-correlation”
(Kernel in original orientation)



“Convolution”
(Kernel flipped in x and y)



Adapted from D. Fouhey and J. Johnson

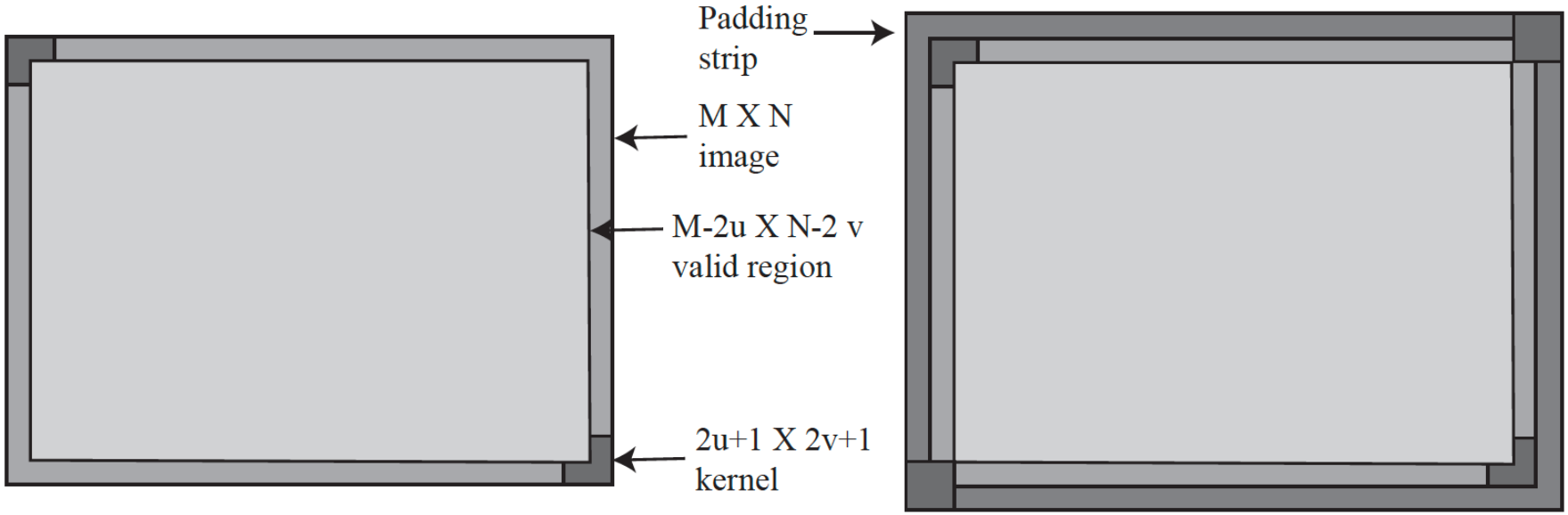


FIGURE 4.1: *The mid-gray box represents an $M \times N$ image, and the darker gray box a $2u + 1 \times 2v + 1$ kernel. The valid region is lighter gray. It can be constructed by placing the kernel at the top left and bottom right corners of the image, then constructing the box that joins their centers (***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***).*

Practical details: Dealing with edges

- To control the size of the output, we need to use *padding*
- What values should we pad the image with?
  - Zero pad (or clip filter)
  - Wrap around
  - Copy edge
  - Reflect across edge

Source: S. Marschner

important property of convolution 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. Check that:

- Convolution is linear in the image, so

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

- Convolution is linear in the mask, so

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

- Convolution is associative, so

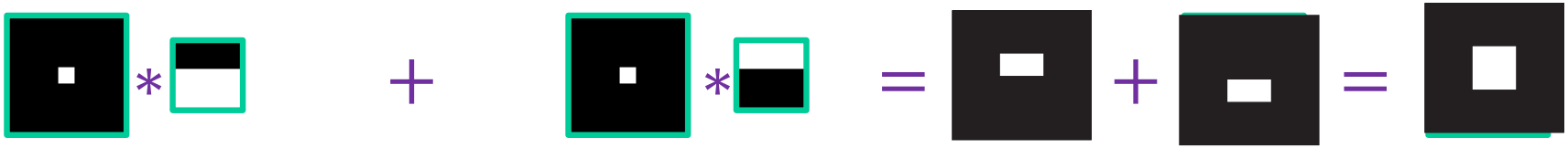
$$\mathcal{W} * (\mathcal{V} * \mathcal{I}) = (\mathcal{W} * \mathcal{V}) * \mathcal{I}.$$

- Convolution is shift-invariant, so

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

Properties: Linearity

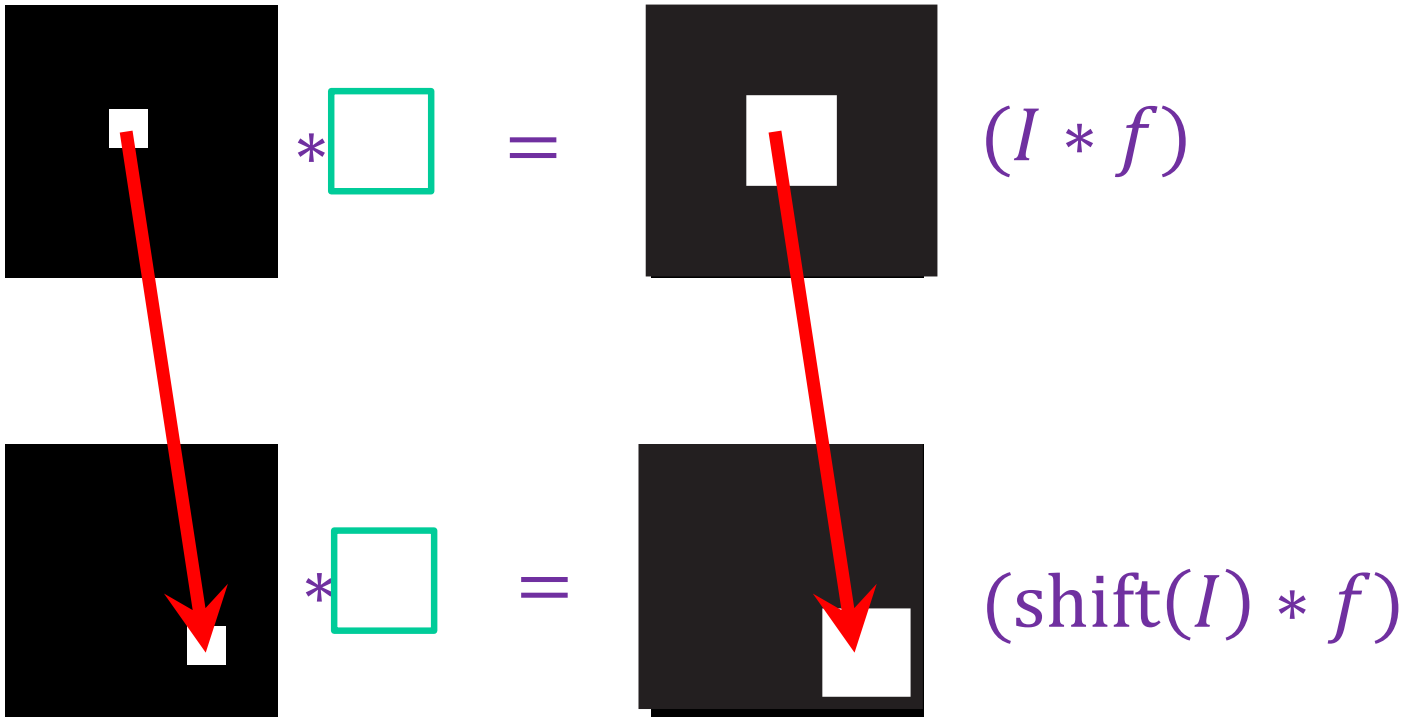
$$I * (f_1 + f_2) = I * f_1 + I * f_2$$



Adapted from D. Fouhey and J. Johnson

Properties: Shift-invariance

$$(\text{shift}(I) * f) = \text{shift}((I(* f))$$



Adapted from D. Fouhey and J. Johnson

More linear filtering properties

- Commutativity: $f * g = g * f$
  - For infinite signals, no difference between filter and signal
- Associativity: $f * (g * h) = (f * g) * h$
  - Convolving several filters one after another is equivalent to convolving with one combined filter:

$$\left((g * f_1) * f_2\right) * f_3 = g * (f_1 * f_2 * f_3)$$

- Identity: for *unit impulse* e, $f * e = f$

Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

?

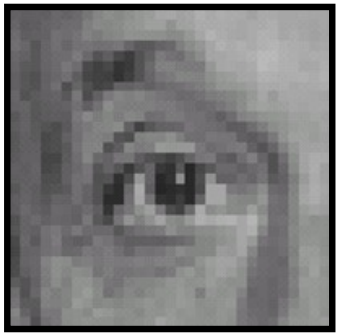
Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

One surrounded by zeros is the *identity filter*



Filtered (no change)

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0

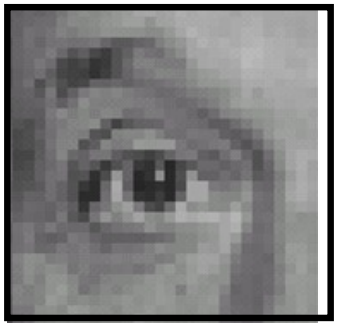
?

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0



Shifted *left*
By one pixel

Source: D. Lowe

Practice with linear filters

Original

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

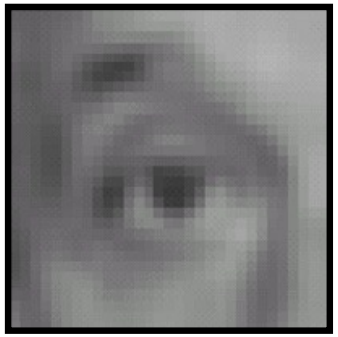
?

Source: D. Lowe

Practice with linear filters

Original

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



Blur (with a box filter)

Source: D. Lowe

Practice with linear filters

Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad ?$$

Source: D. Lowe

Practice with linear filters

Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

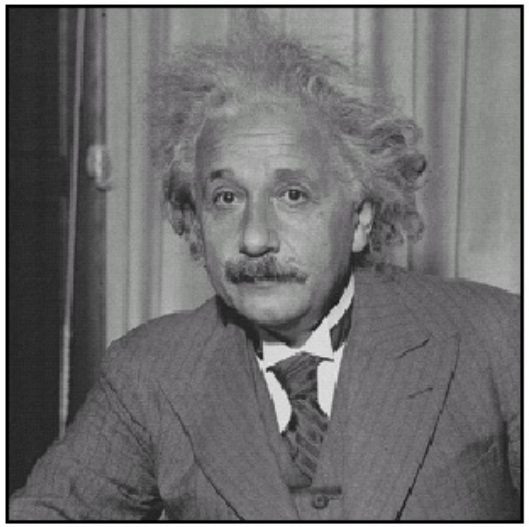
Sharpening filter:
Accentuates differences with local average

(Note that filter sums to 1)

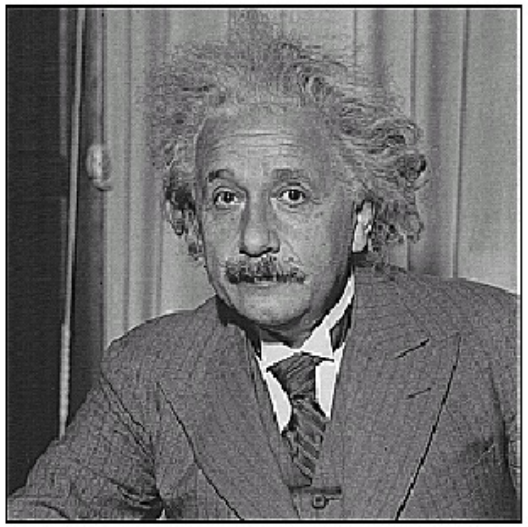
Sharpened

Source: D. Lowe

Sharpening



before



after

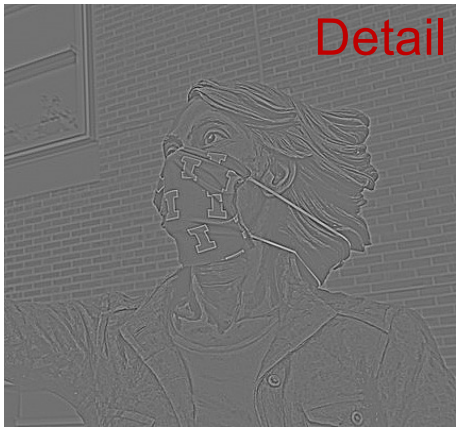
Source: D. Lowe

Sharpening

−

=

+



=

Source:
S. Gupta

Filters are dot products

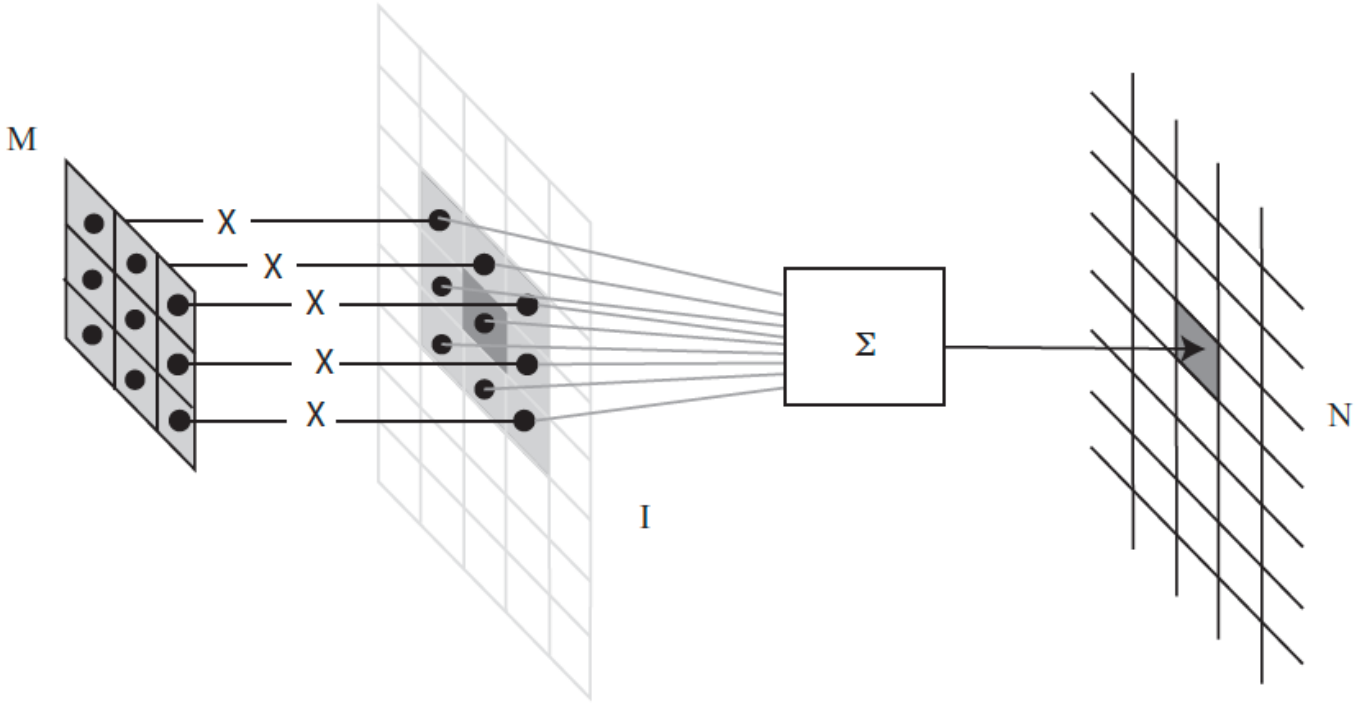


FIGURE 3.1: *To compute the value of $\mathcal{N}$ at some location, you shift a copy of $\mathcal{M}$ (the flipped version of $\mathcal{W}$) to lie over that location in $\mathcal{I}$; you multiply together the non-zero elements of $\mathcal{M}$ and $\mathcal{I}$ that lie on top of one another; and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

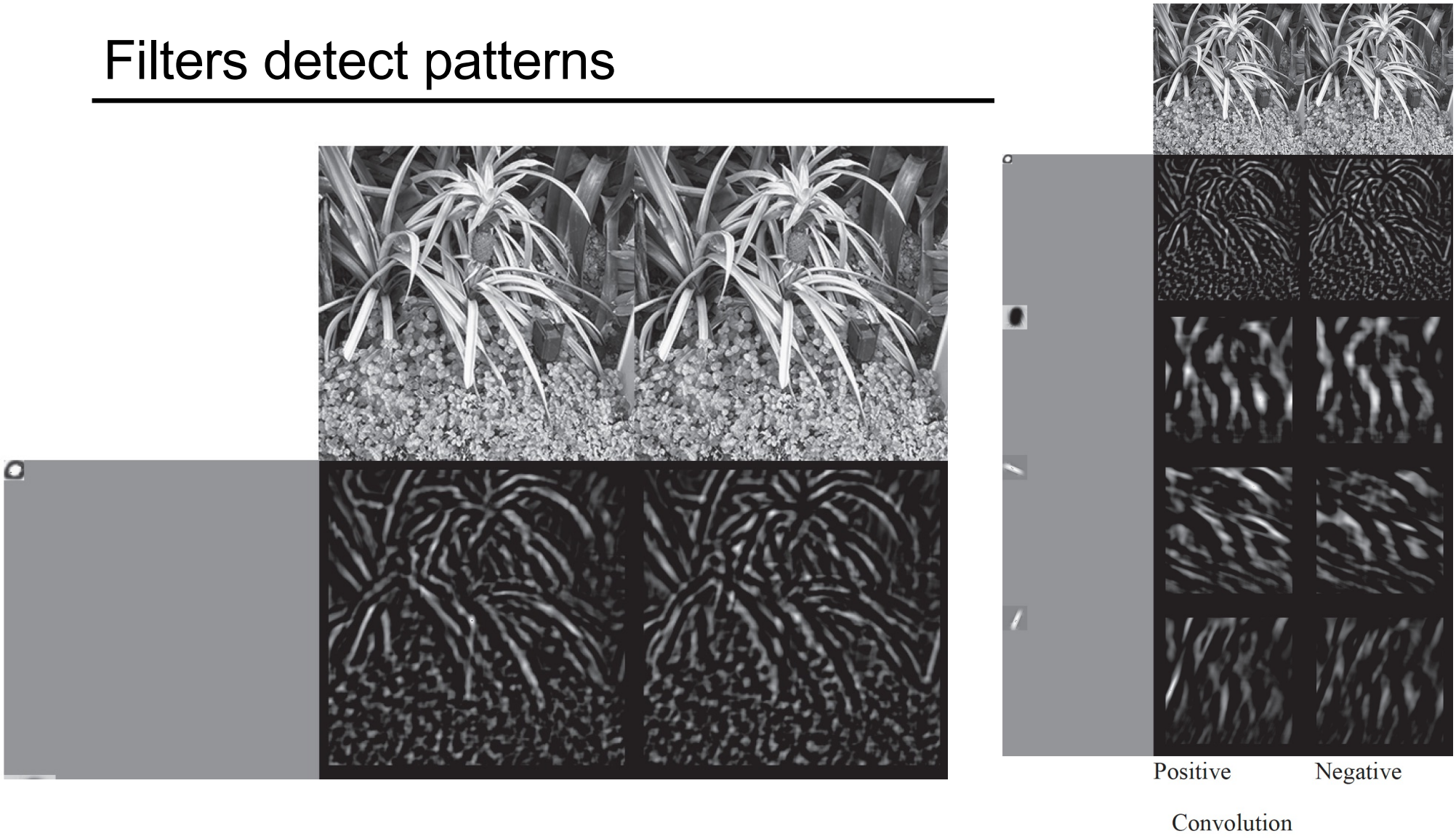
ReLUs

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

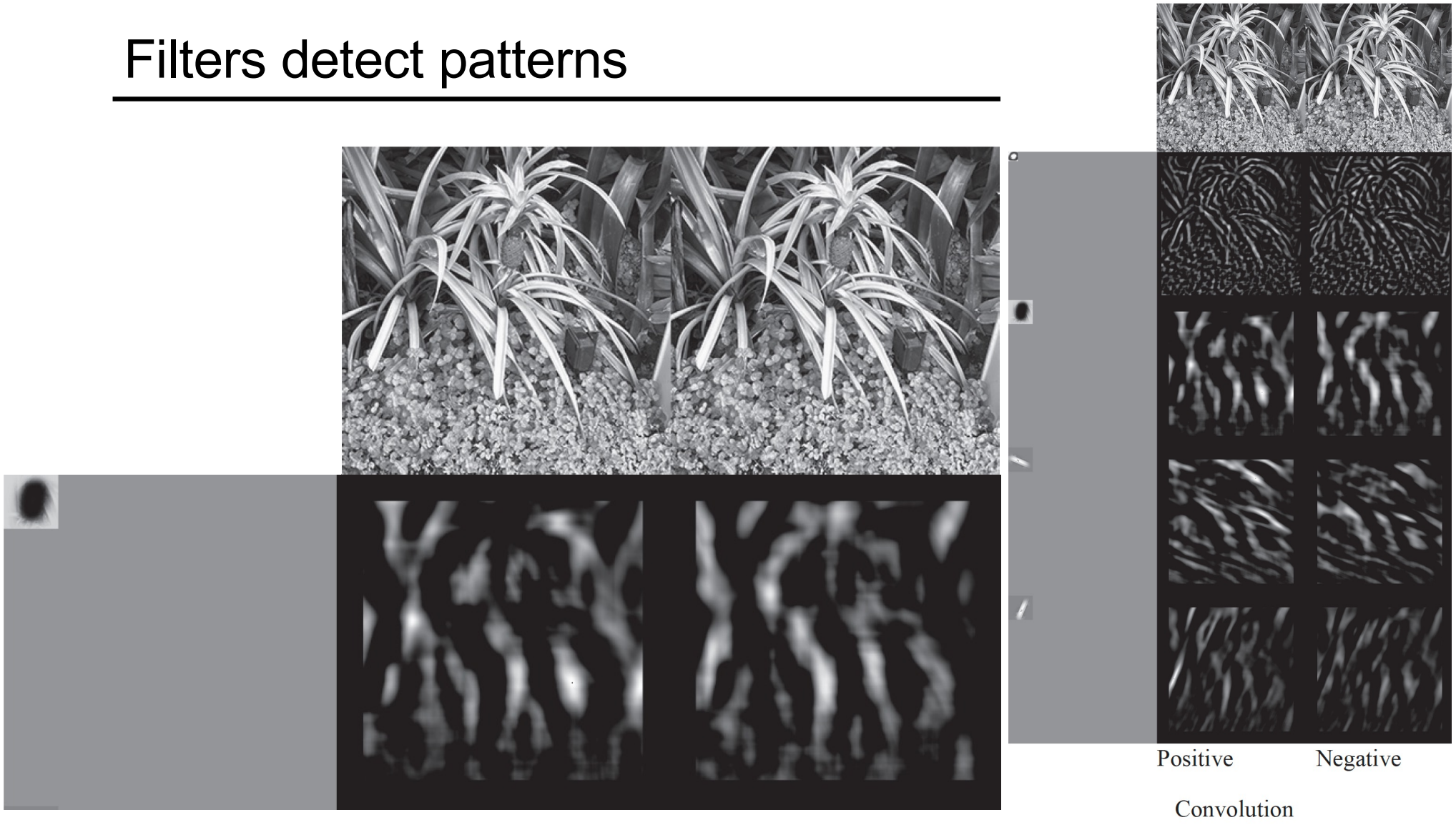
$$\texttt{relu}(x) = \begin{cases} x & \text{for } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

(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

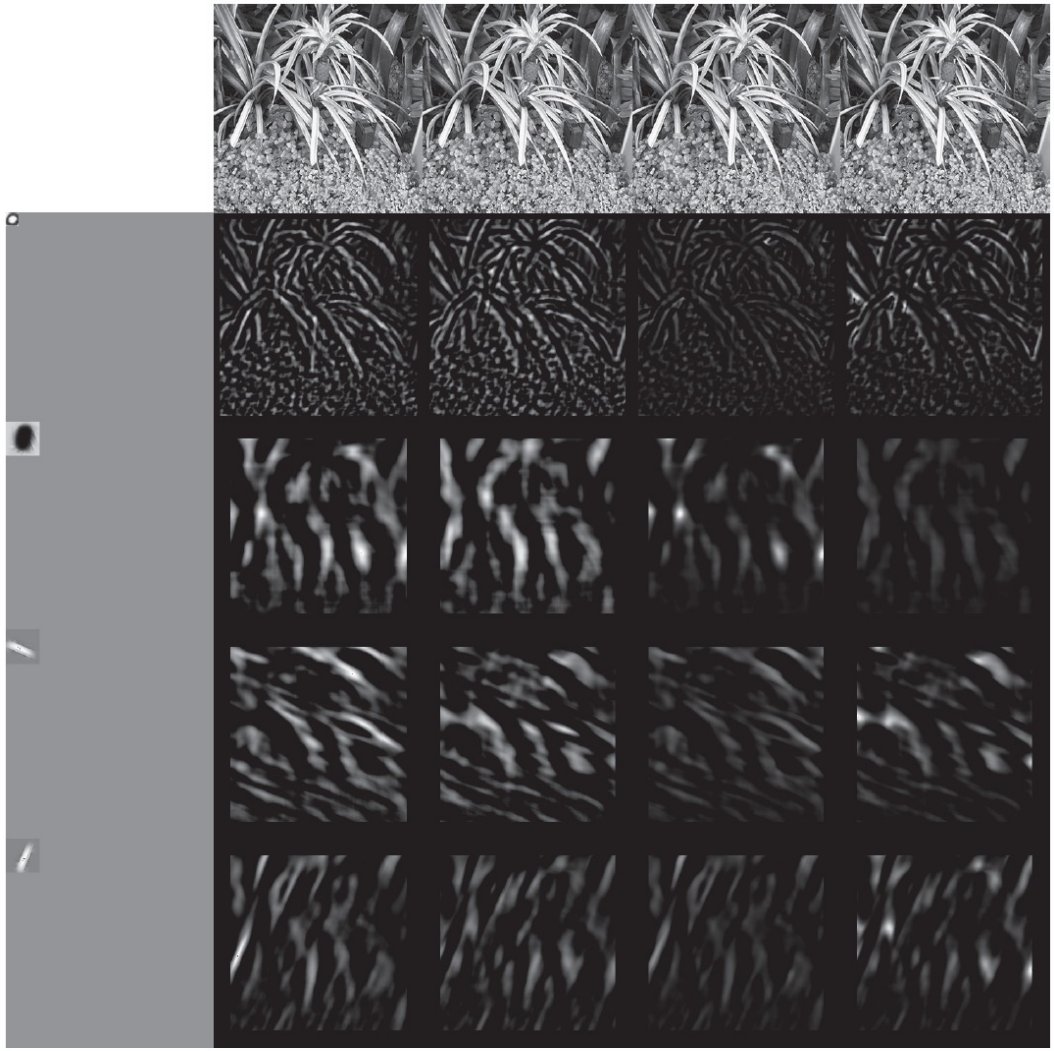


The dot-product analogy reveals some reasons that convolution is not a particularly good pattern detector. Assume that the mean of the kernel is not zero. In this case, adding a constant offset to the image will change the value of the convolution, so you cannot rely on the value. This can be dealt with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

$$-1 \quad 1$$

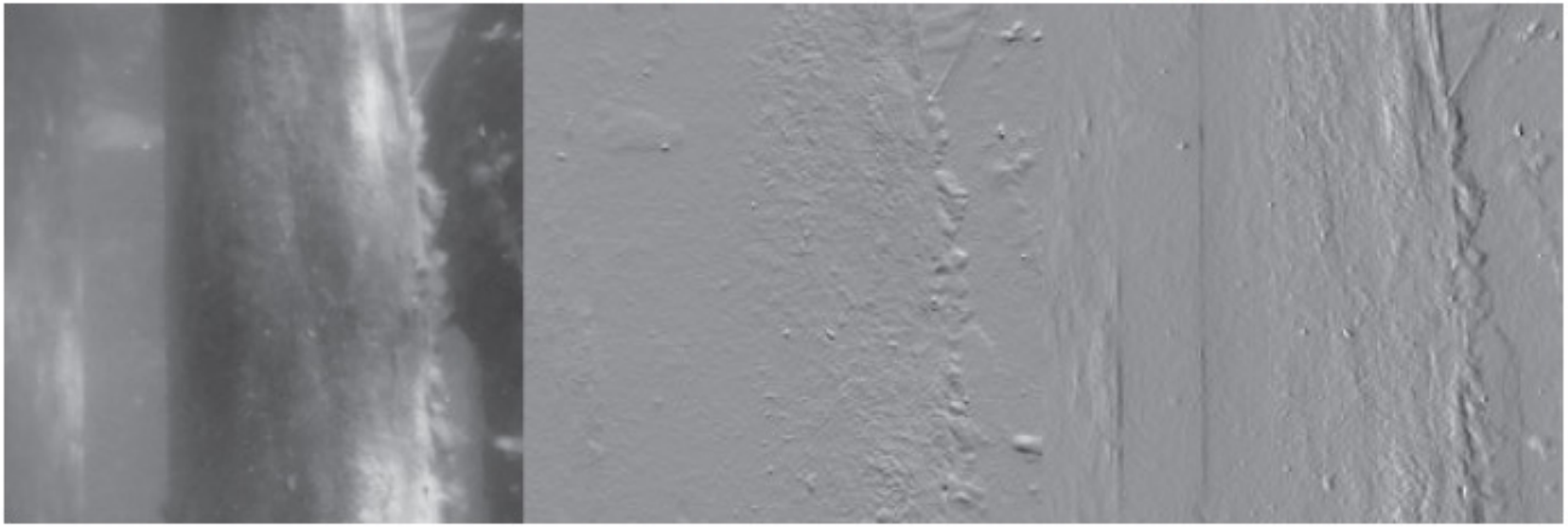
will estimate $\partial \mathcal{I}/\partial x$ (nothing in the definition requires convolution with a square kernel). Notice that this kernel "looks like" a dark pixel next to a light pixel, and will respond most strongly to that pattern. By the same argument, $\partial \mathcal{I}/\partial y \approx \mathcal{I}_{i,j+1} - \mathcal{I}_{i,j}$. These kinds of derivative estimates are known as *finite differences*.

Image derivatives with finite differences

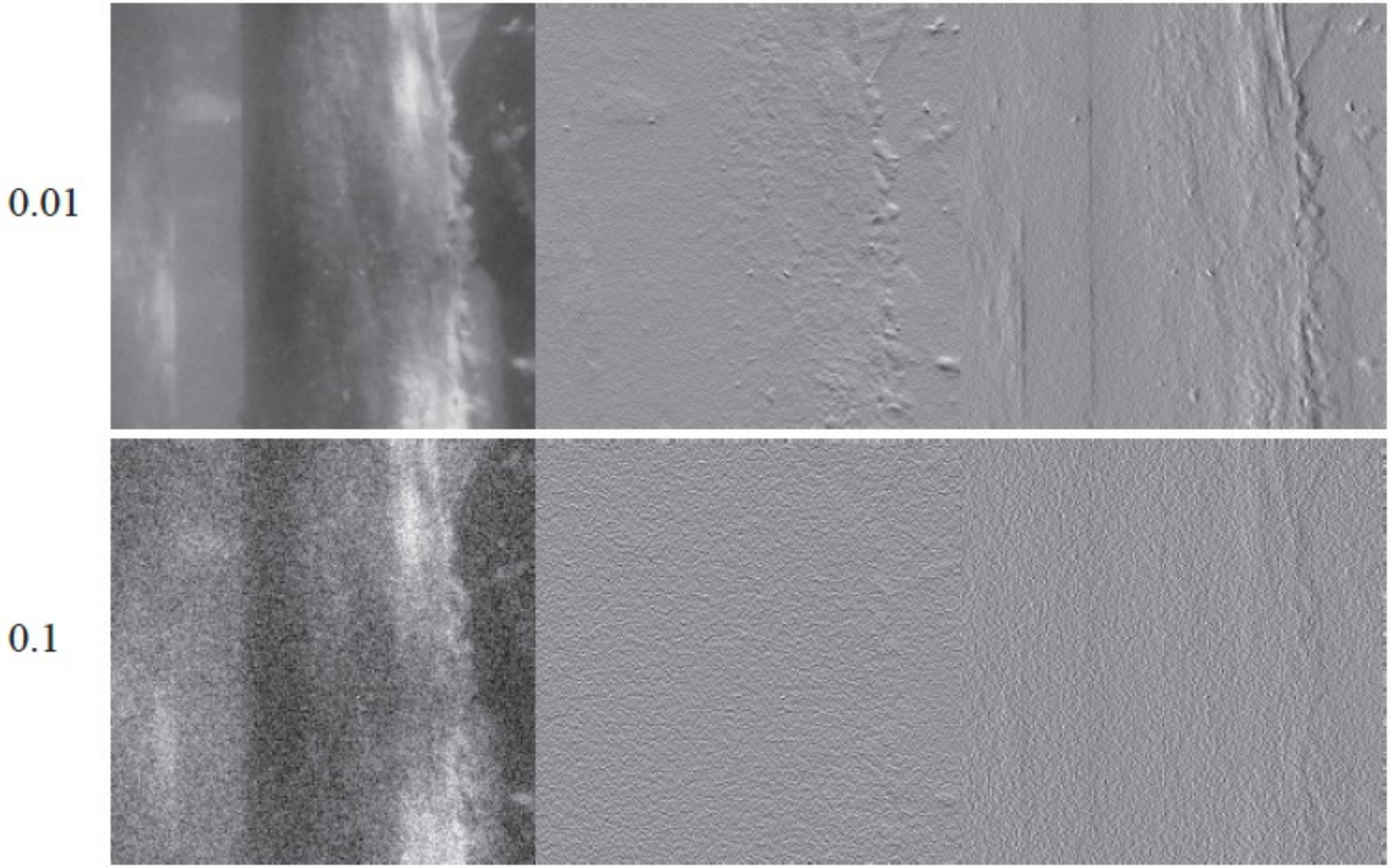
horizontal

vertical

0



Finite differences are overexcited by noise



Denoising with filters

A crucial property of images

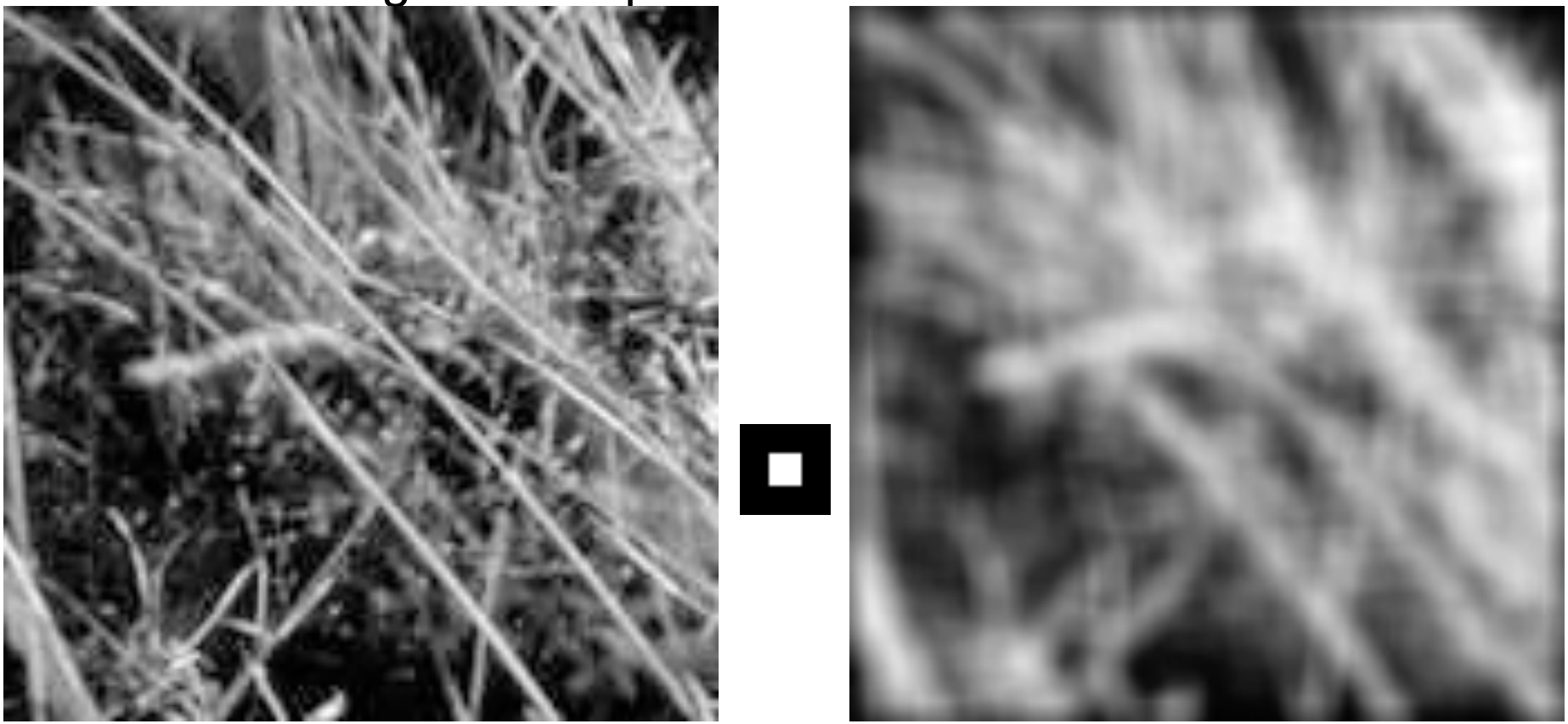
Pixels are like their neighbors

mostly, for most pixels

Imagine you wish to denoise an image. You could do so by averaging neighbors (a filter!).

Smoothing with box filter revisited

- What's wrong with this picture?



Source: D. Forsyth

A crucial property of images

Pixels are like their neighbors

mostly, for most pixels

the closer the neighbor the more alike

Imagine you wish to denoise an image. You could do so by averaging neighbors (a filter!). Weighting the neighbors so nearby neighbors get heavier weights is a good move.

Smoothing with box filter revisited

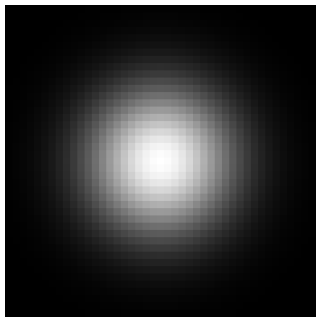
- What's wrong with this picture?
- What's the solution?
  - To eliminate edge effects, weight contribution of neighborhood pixels according to their closeness to the center

"proportional to"
(renormalize values to sum to 1)

$$G(x, y) \propto \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

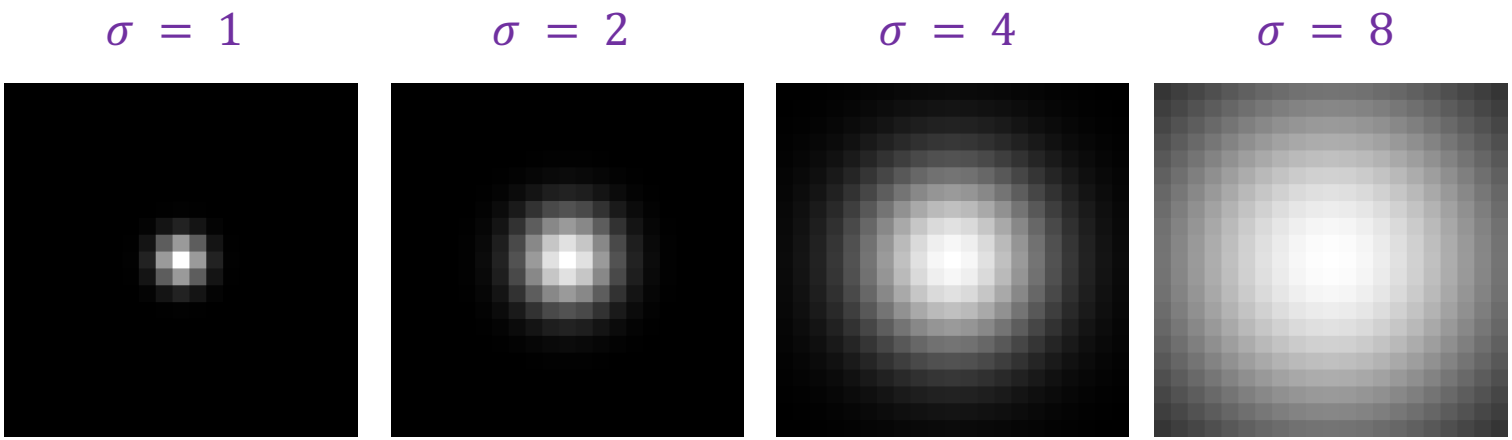
standard deviation
(determines size of "blob")

Gaussian filter



Adapted from D. Fouhey and J. Johnson

Gaussian filters



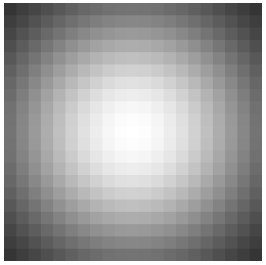
Filter size: 21×21

Adapted from D. Fouhey and J. Johnson

Choosing filter size

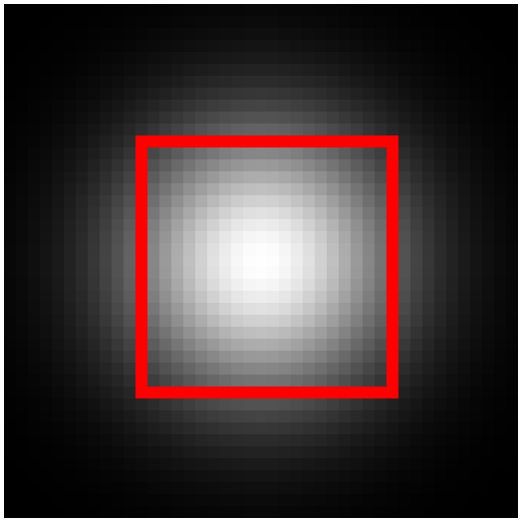
- Rule of thumb: set filter width to about 6σ (captures 99.7% of the energy)

$\sigma = 8$
$\text{Width} = 21$



Too small!

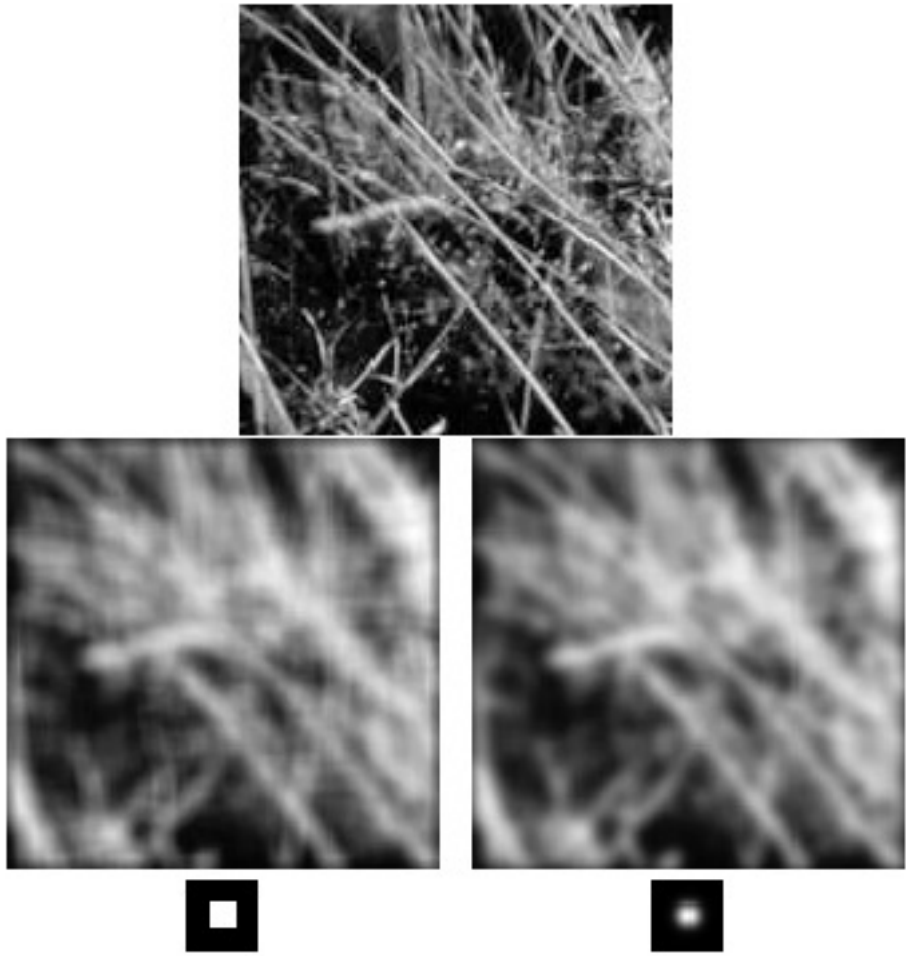
$\sigma = 8$
$\text{Width} = 43$



A bit small (might be OK)

Adapted from D. Fouhey and J. Johnson

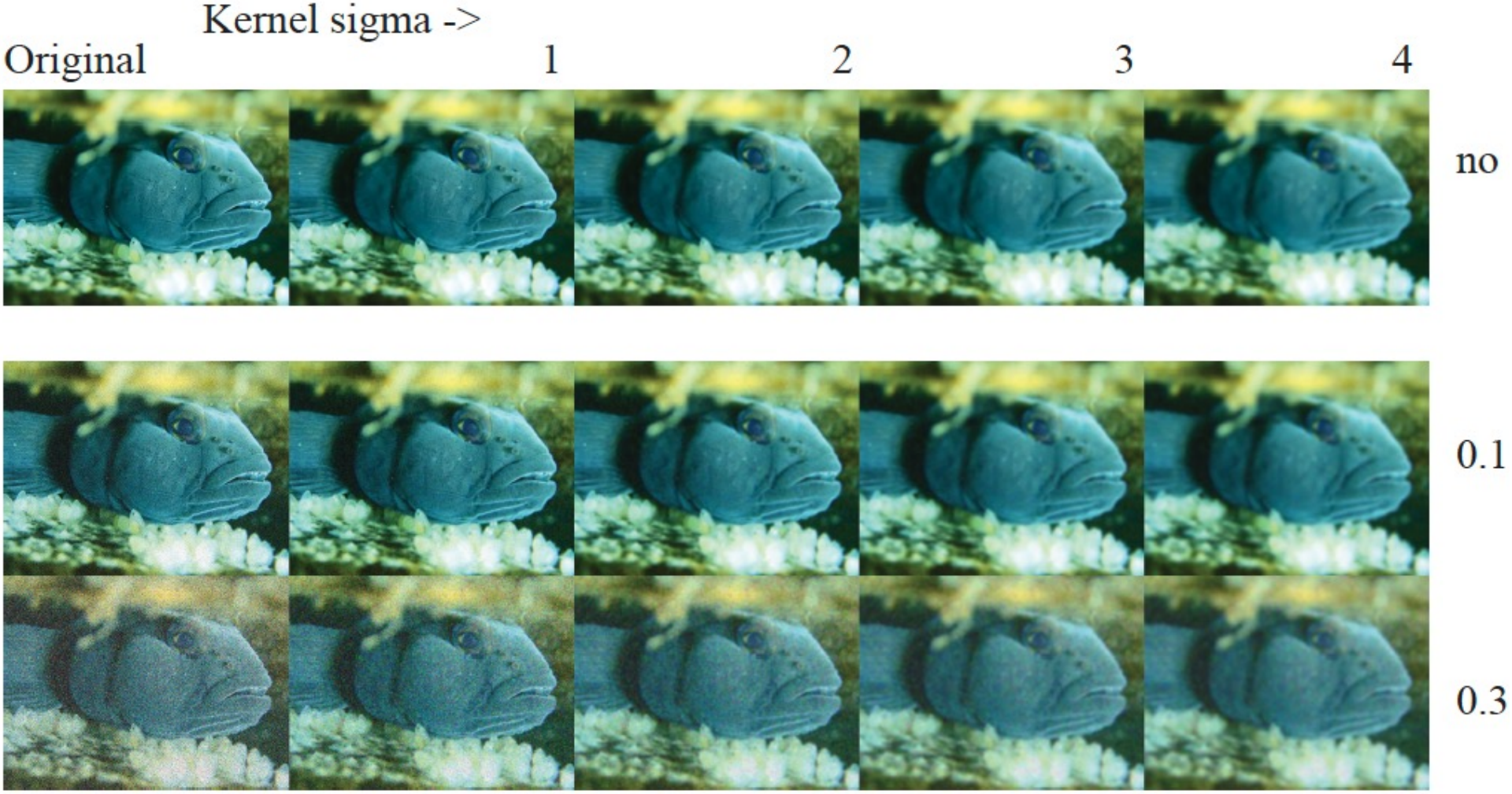
Gaussian vs. box filtering



Gaussian noise

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 4.6 shows some examples of additive stationary Gaussian noise.

Gaussian smoothing of Gaussian noise

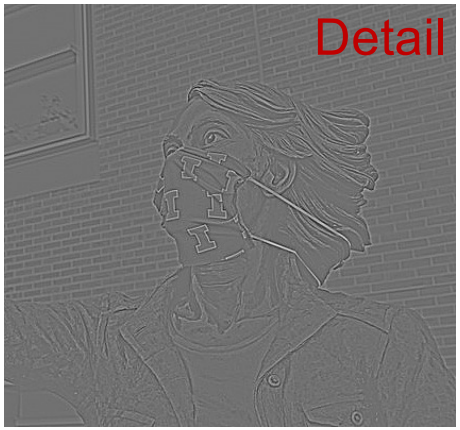


Smoothing by how much?

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.

Sharpening

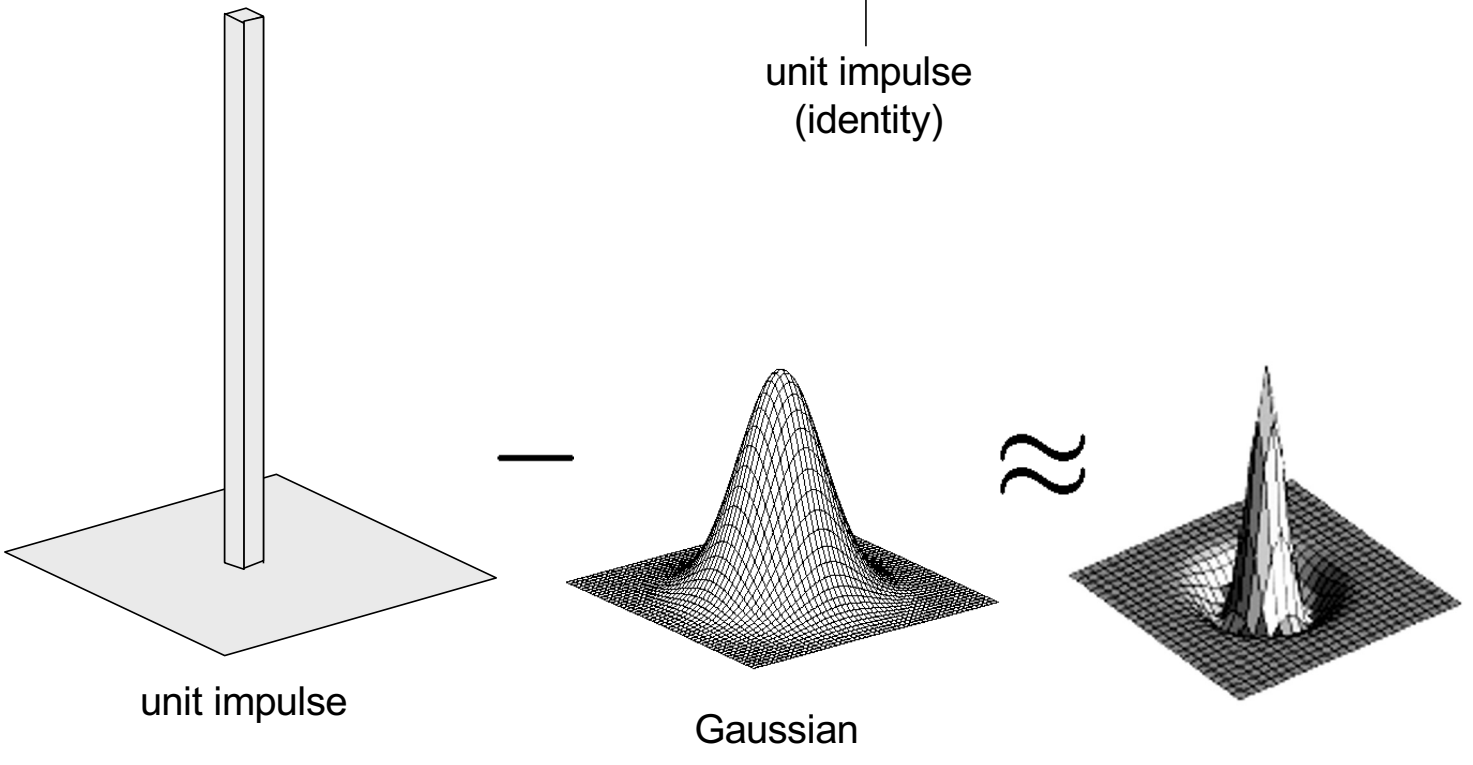


Source:
S. Gupta

“Detail” filter follows from linearity

$$I - I * g = I * (e - g)$$

unit impulse
(identity)



unit impulse

Gaussian

Poisson noise

For each pixel location, flip a biased coin

if it comes up heads, move on

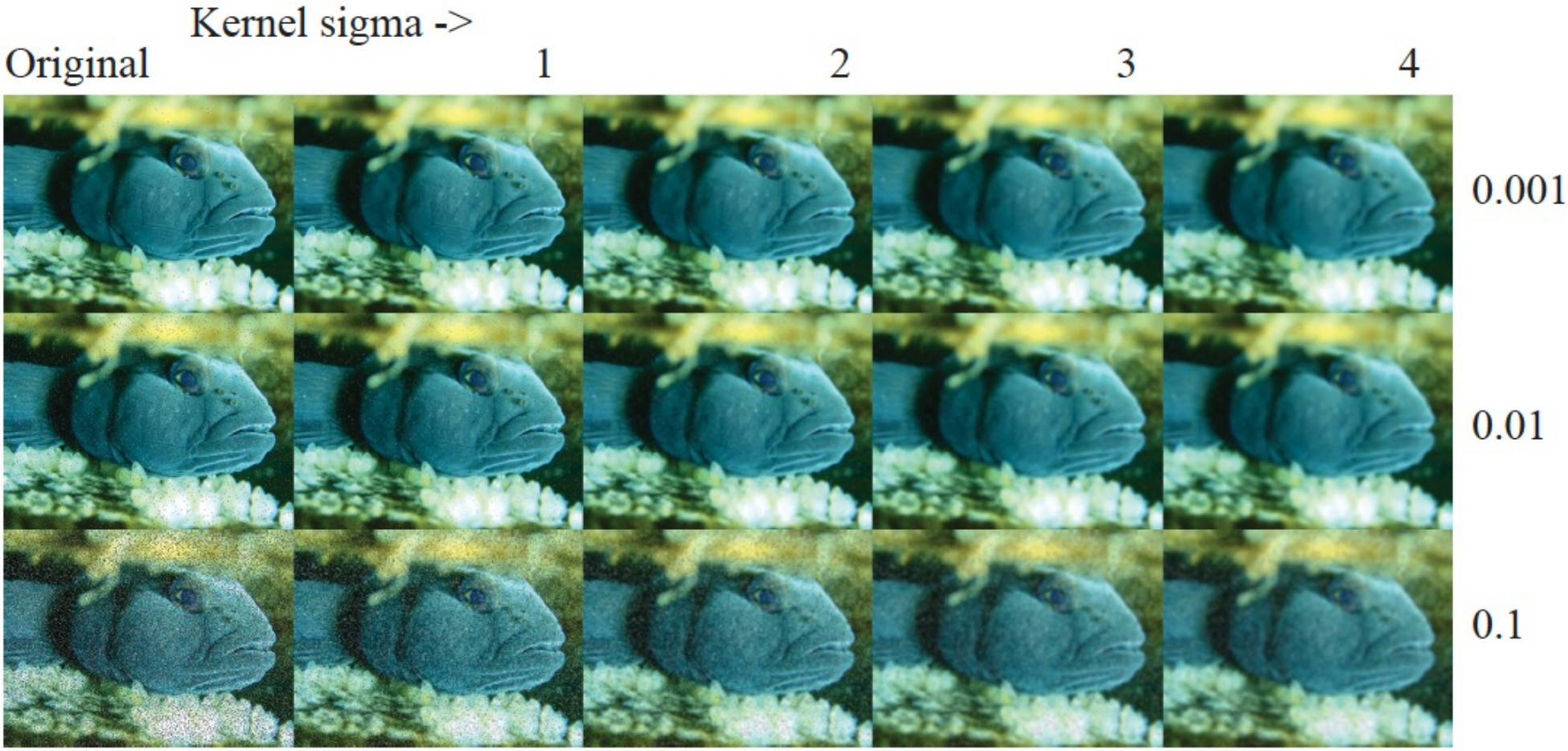
if it comes up tails, flip a fair coin

if that is heads, pixel -> full bright

tails, pixel -> full dark

Models device damage, manufacturing failures, some kinds of transmission error, etc.

Smoothing Poisson noise with a gaussian filter



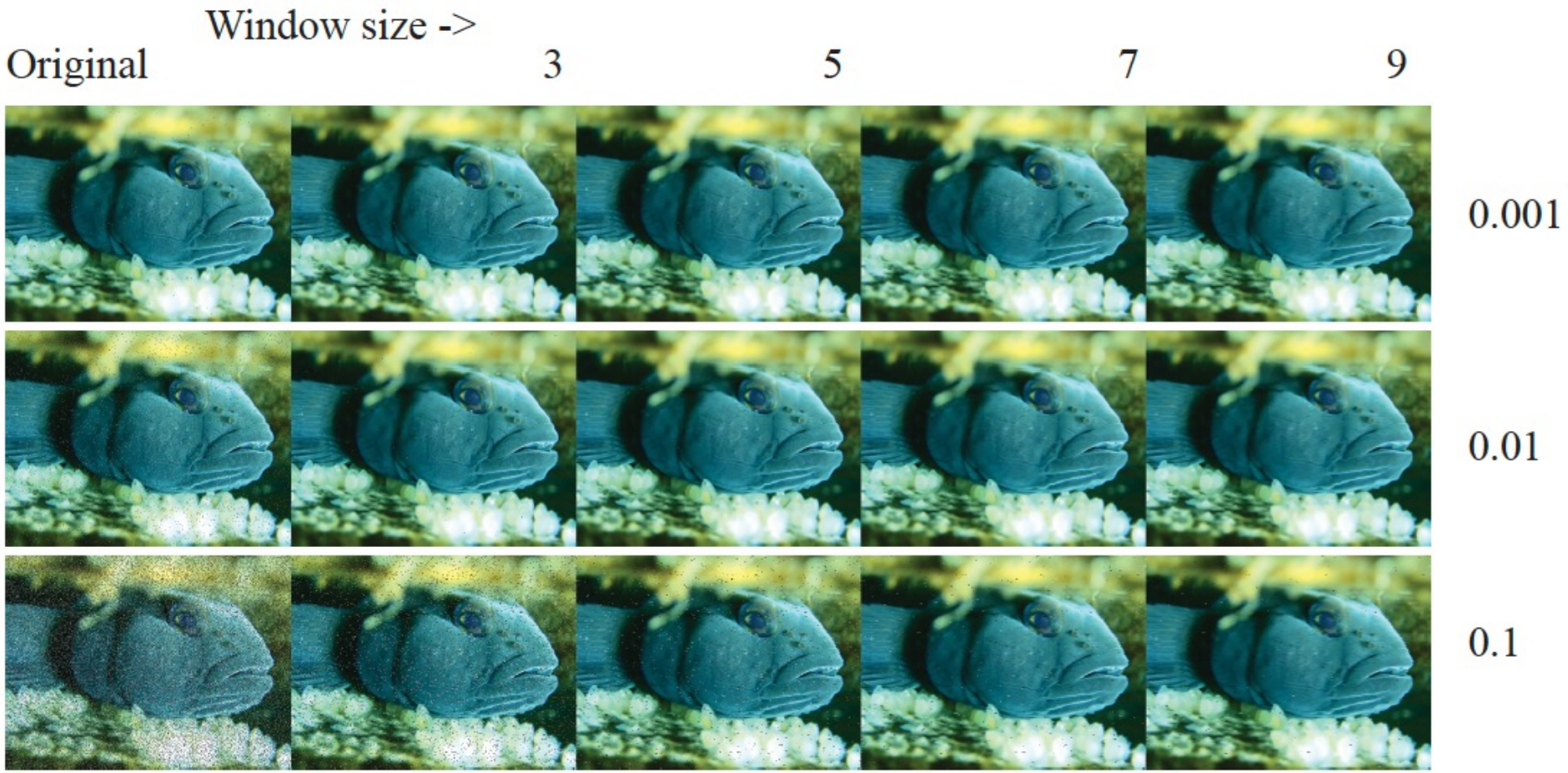
The median filter

N_{ij} = median(Neighborhood(O_{ij}))

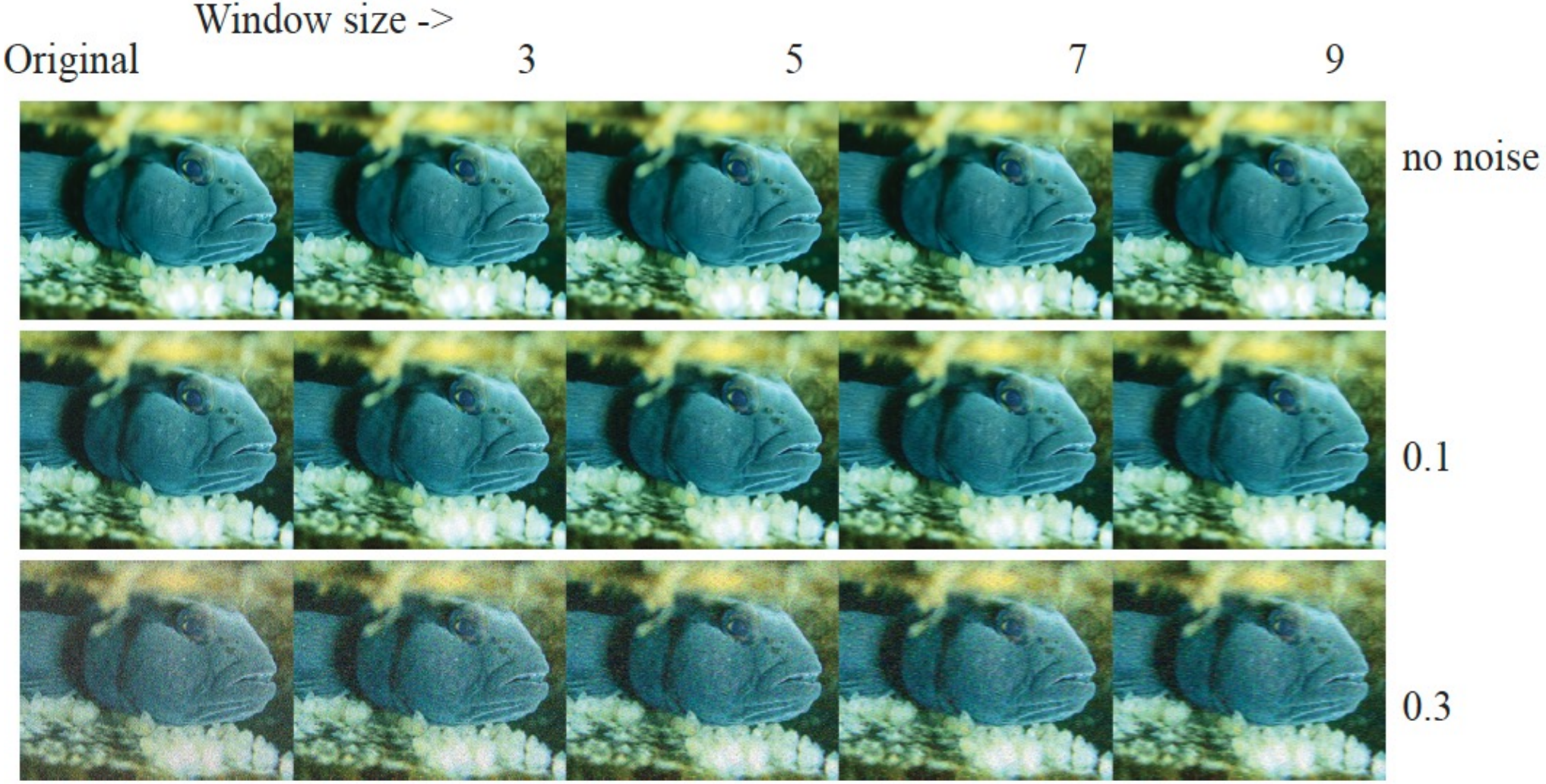
THIS ISN'T LINEAR!

(check you're sure of this)

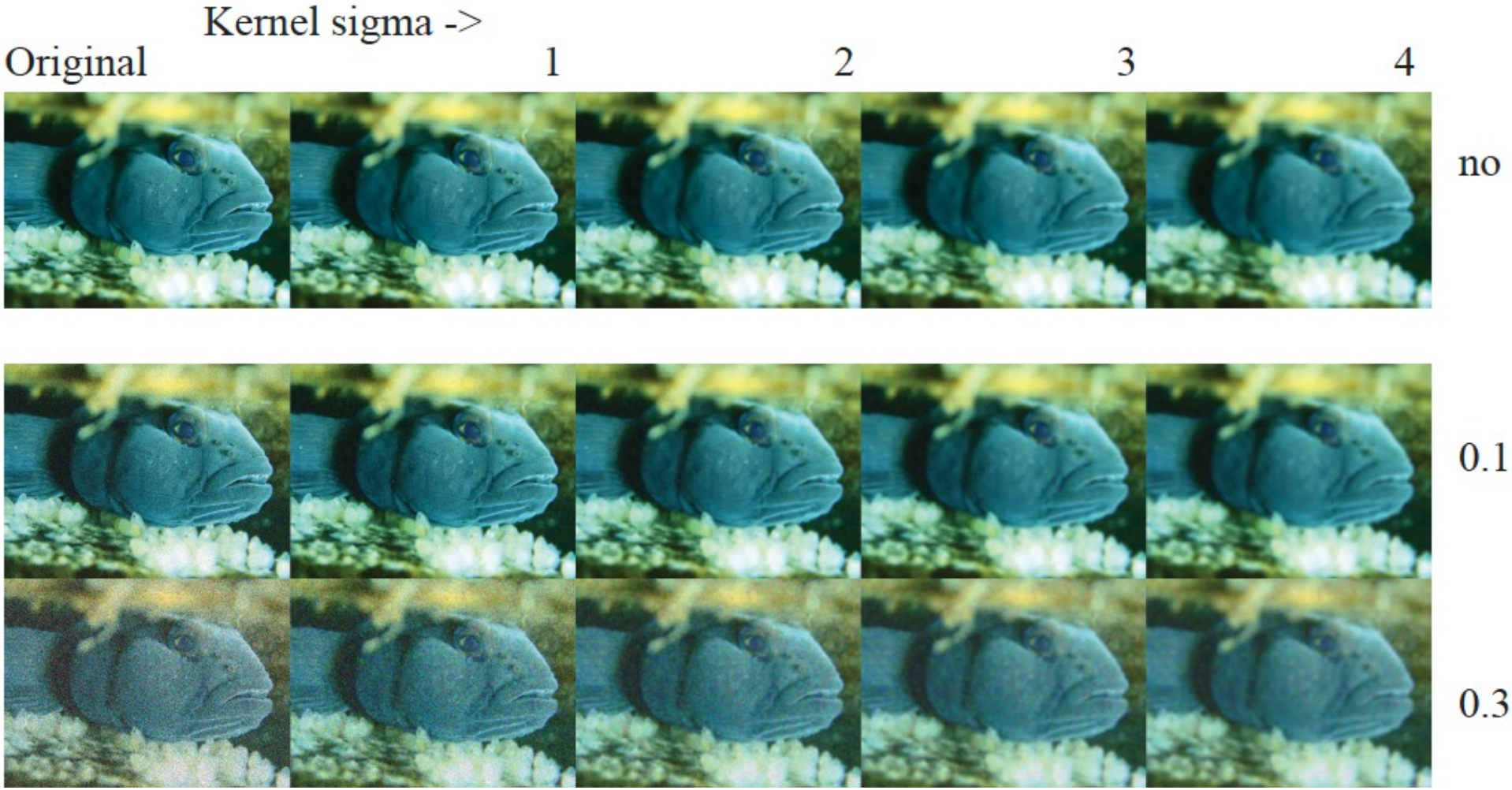
Smoothing Poisson noise with a median filter



Median filters vs Gaussian noise



Gaussian smoothing of Gaussian 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

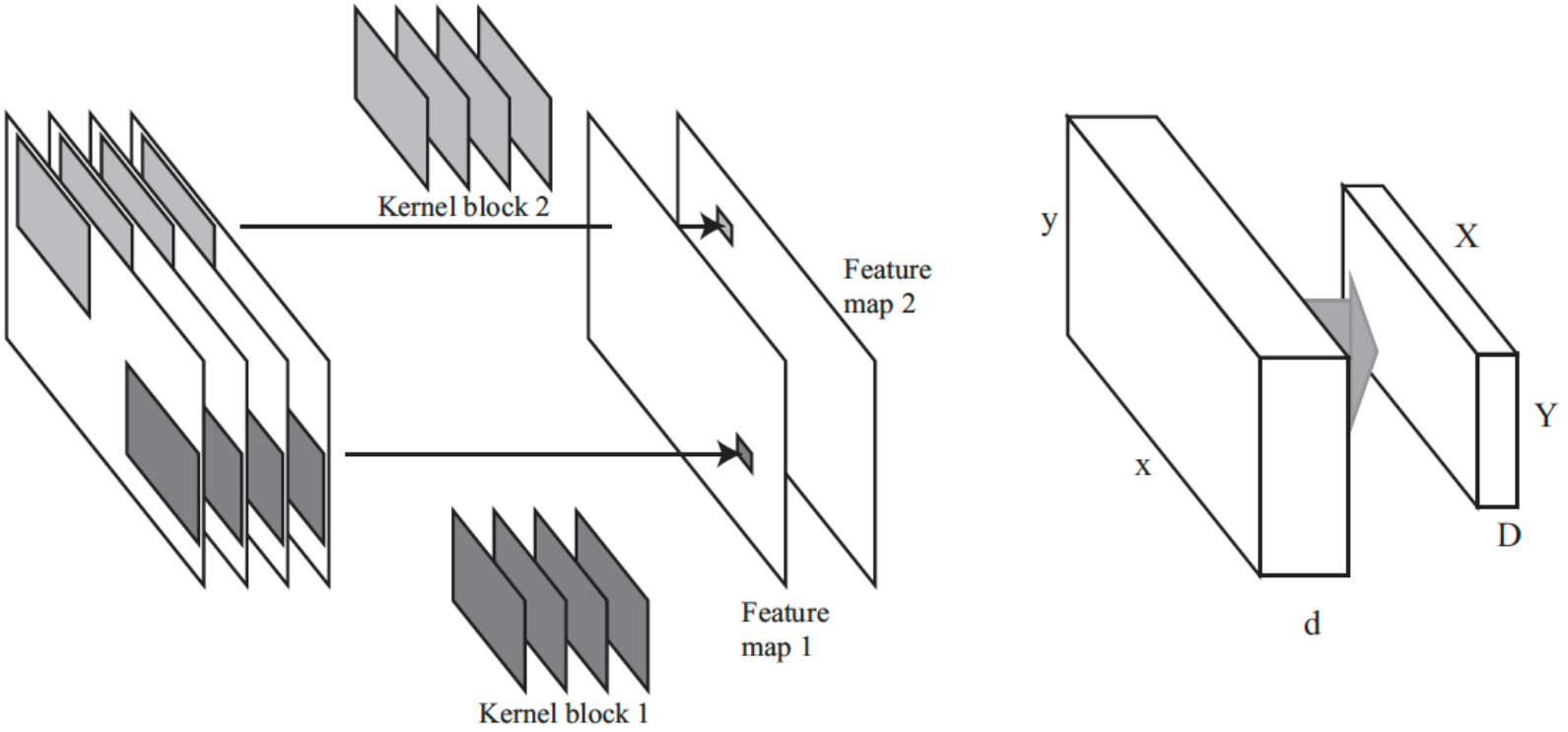
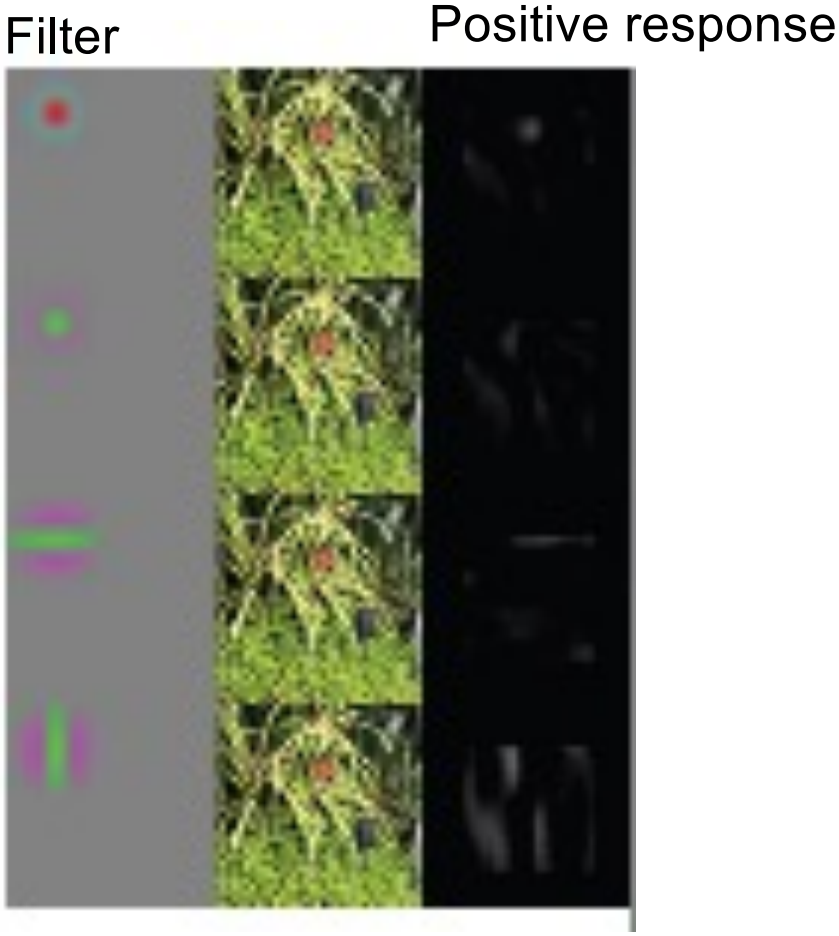
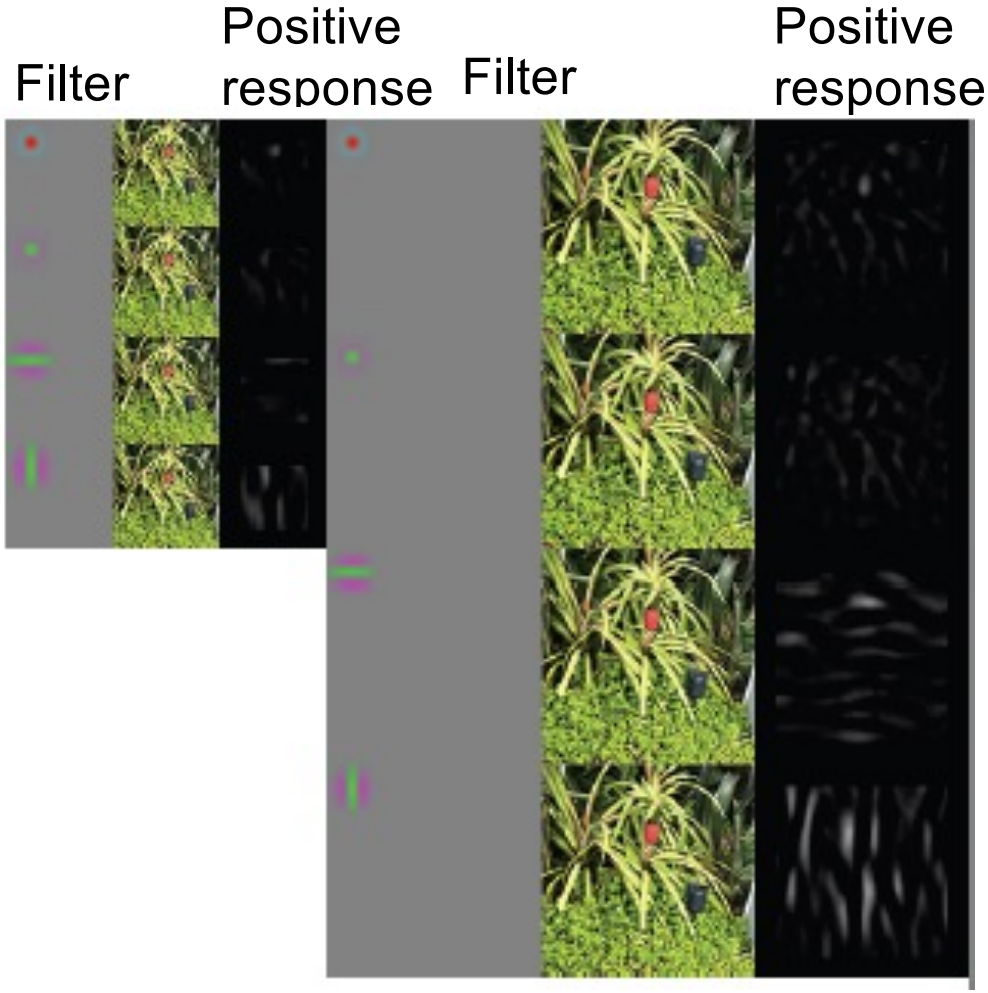


FIGURE 4.4: *On the* **left**, *two kernels (now 3D, as in the text) applied to a set of feature maps produce one new feature map per kernel, using the procedure of the text (the bias term isn't shown). Abstract this as a process that takes an* $x \times y \times d$ *block to an* $X \times Y \times D$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

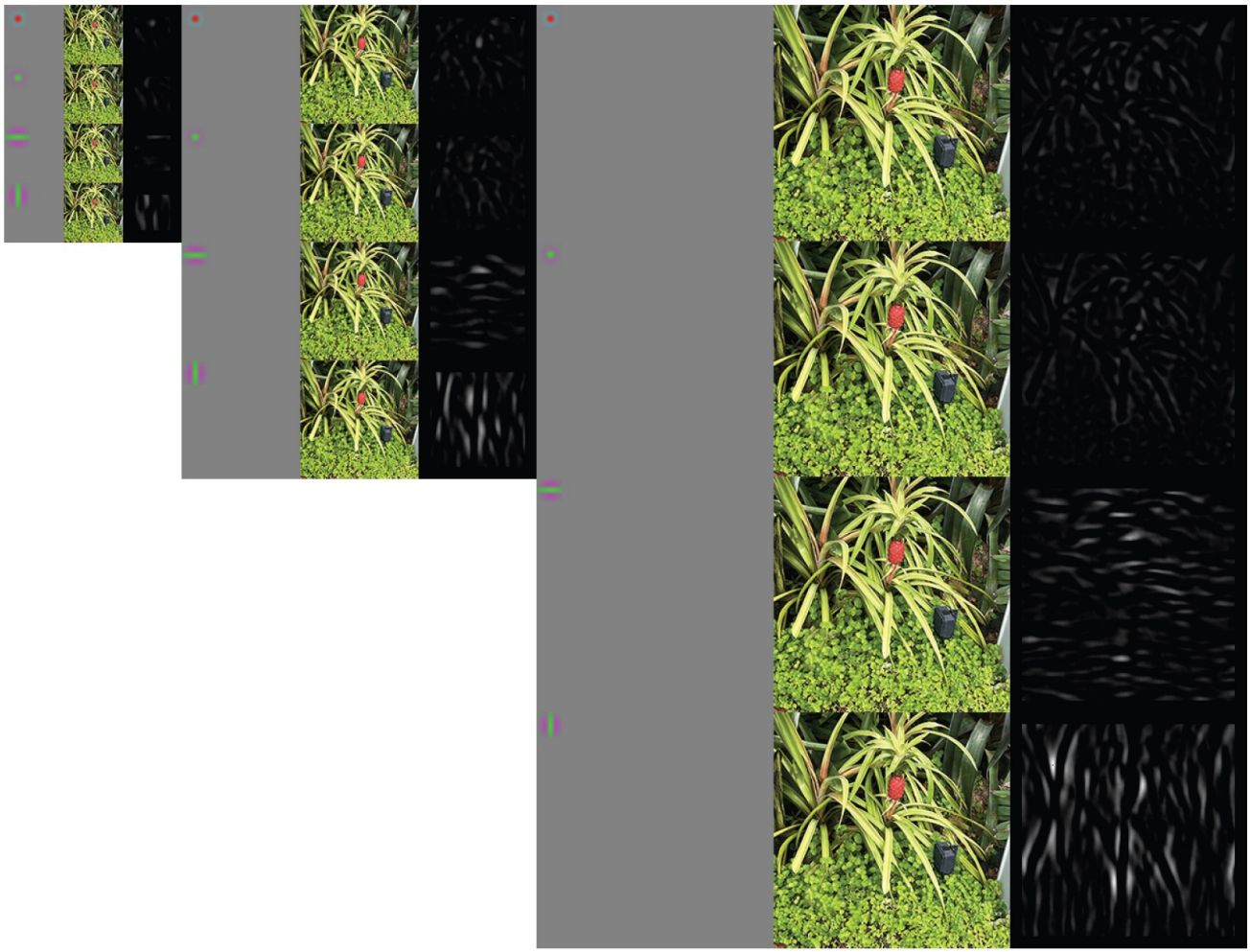
Representing Images with Filter Banks



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