Spatial filtering

Semester 2, 2025

Kris Ehinger

Outline

- Introduction to convolution
- Commonly-used linear filters
- Filters in practice

Learning outcomes

- Explain the convolution and cross-correlation operations
- Identify commonly-used filters and their expected outputs
- Explain practical considerations in implementing filters (efficiency, border handling)

Introduction to convolution

Pixel operator

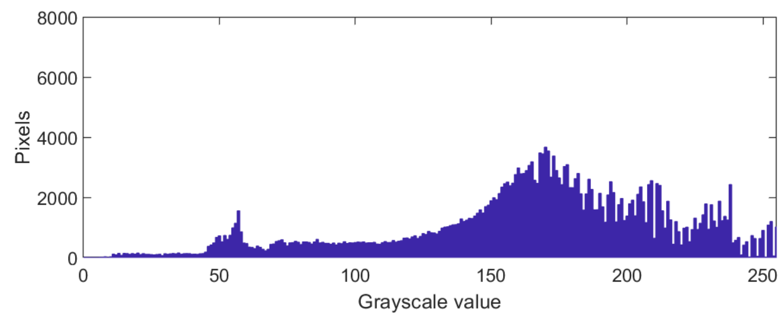
- Pixel operator: computes an output value at each pixel location, based on the input pixel value

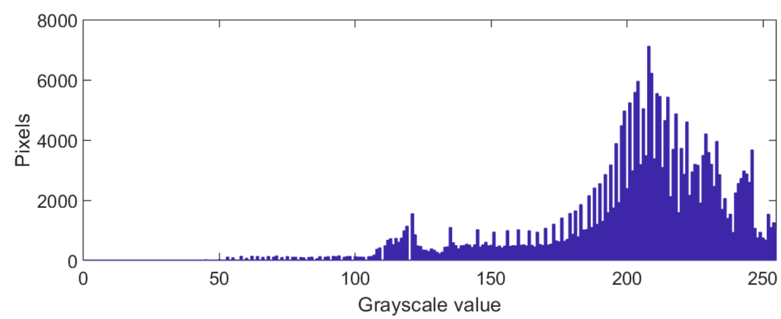
$$g(i,j) = h(f(i,j))$$

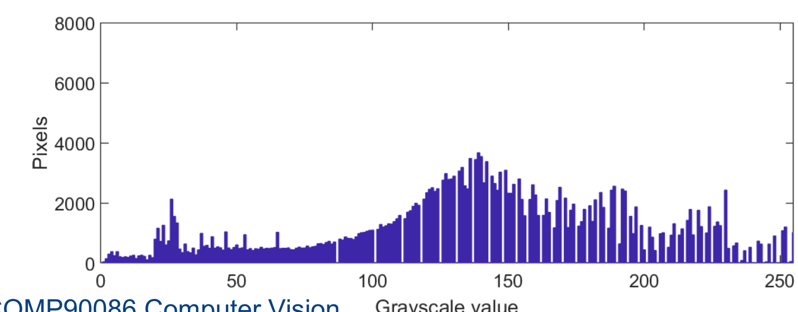
Output image g Input image f

- Example: $g(i,j) = 0.5(f(i,j))$

Gamma correction







Local operator

- Local operator: computes an output value at each pixel location, based on a neighbourhood of pixels around the input pixel
- Example: sharpening filter

Linear filtering

- Output pixel's value is a weighted sum of a neighbourhood around the input pixel

$$g(i,j) = h(u,v) \otimes f(i,j)$$

Output image g Kernel h Input image f

Cross-correlation convolution

$$g(i,j) = \sum_{u,v} f(i+u, j+v) h(u,v)$$

Linear filtering

- Output pixel's value is a weighted sum of a neighbourhood around the input pixel

$$g(i,j) = h(u,v) * f(i,j)$$

Output image g Kernel h Input image f

Convolution operator

$$g(i,j) = \sum_{u,v} f(i-u, j-v)h(u,v)$$

Linear filtering

- Consider a 3x4 image and 2x2 kernel

1	5	1	1
1	2	8	2
0	0	1	8

Image

1	0
0	1

Kernel

3		

Output

Position 1:

1x1	5x0
1x0	2x1

Linear filtering

- Consider a 3x4 image and 2x2 kernel

1	5	1	1
1	2	8	2
0	0	1	8

Image

1	0
0	1

Kernel

3	13	

Output

Position 2:

5x1	1x0
2x0	8x1

Linear filtering

- Consider a 3x4 image and 2x2 kernel

1	5	1	1
1	2	8	2
0	0	1	8

Image

1	0
0	1

Kernel

3	13	3

Output

Position 3:

1x1	1x0
8x0	2x1

Linear filtering

- Consider a 3x4 image and 2x2 kernel

1	5	1	1
1	2	8	2
0	0	1	8

Image

1	0
0	1

Kernel

3	13	3
1		

Output

Linear filtering

- Consider a 3x4 image and 2x2 kernel

1	5	1	1
1	2	8	2
0	0	1	8

Image

1	0
0	1

Kernel

3	13	3
1	3	

Output

Linear filtering

- Consider a 3x4 image and 2x2 kernel

1	5	1	1
1	2	8	2
0	0	1	8

Image

1	0
0	1

Kernel

3	13	3
1	3	16

Output

Cross-correlation vs. convolution

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

F[x,y]

a	b	c
d	e	f
g	h	i

H[u,v]

		i	h	g		
		f	e	d		
		c	b	a		

$F[x,y] \otimes H[u,v]$

Cross-correlation: overlay filter on image

Image: S. Seitz

Cross-correlation vs. convolution

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

F[x,y]

a	b	c
d	e	f
g	h	i

H[u,v]

		a	b	c		
		d	e	f		
		g	h	i		

F[x,y] ∗ H[u,v]

Convolution: flip filter horizontally and vertically

Image: S. Seitz

Summary

- Pixel operator: transform pixel based on its value
- Local operator: transform pixel based on its neighbours
- Convolution (and cross-correlation): operations that apply a linear filter to an image

Common filters

Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

?

Slide: D. Lowe

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0

?

Slide: D. Lowe

Practice with linear filters

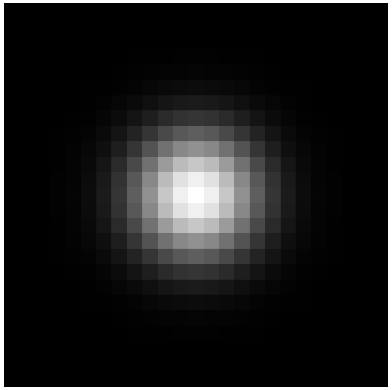
Original

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

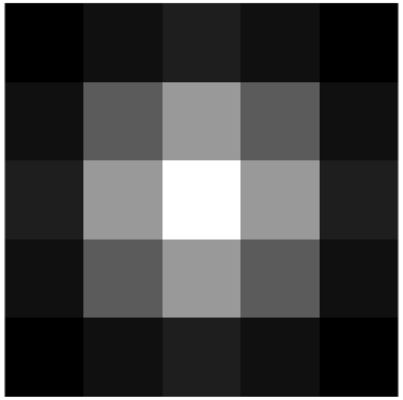
(Note that filter sums to 1)

Slide: D. Lowe

Common filters: Gaussian

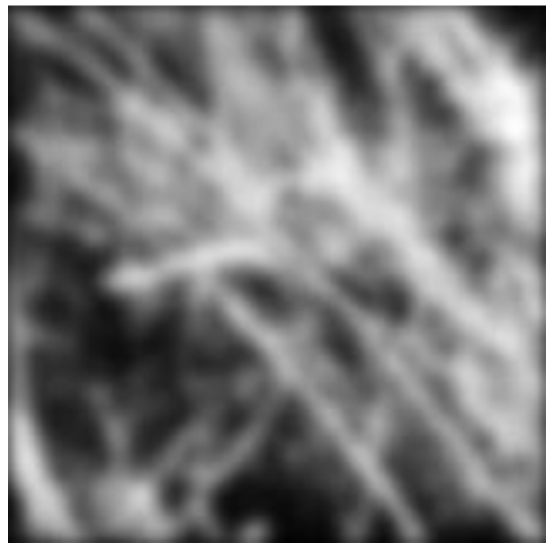


$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

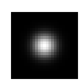


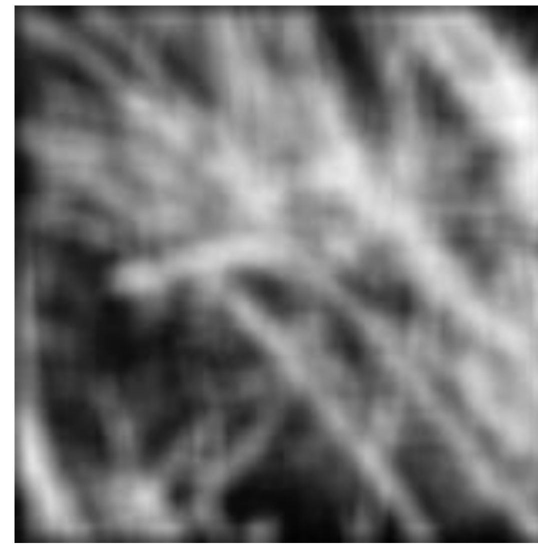
Gaussian kernel
Kernel size: 5 x 5 px
σ = 1

Blur filters

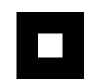


Gaussian kernel

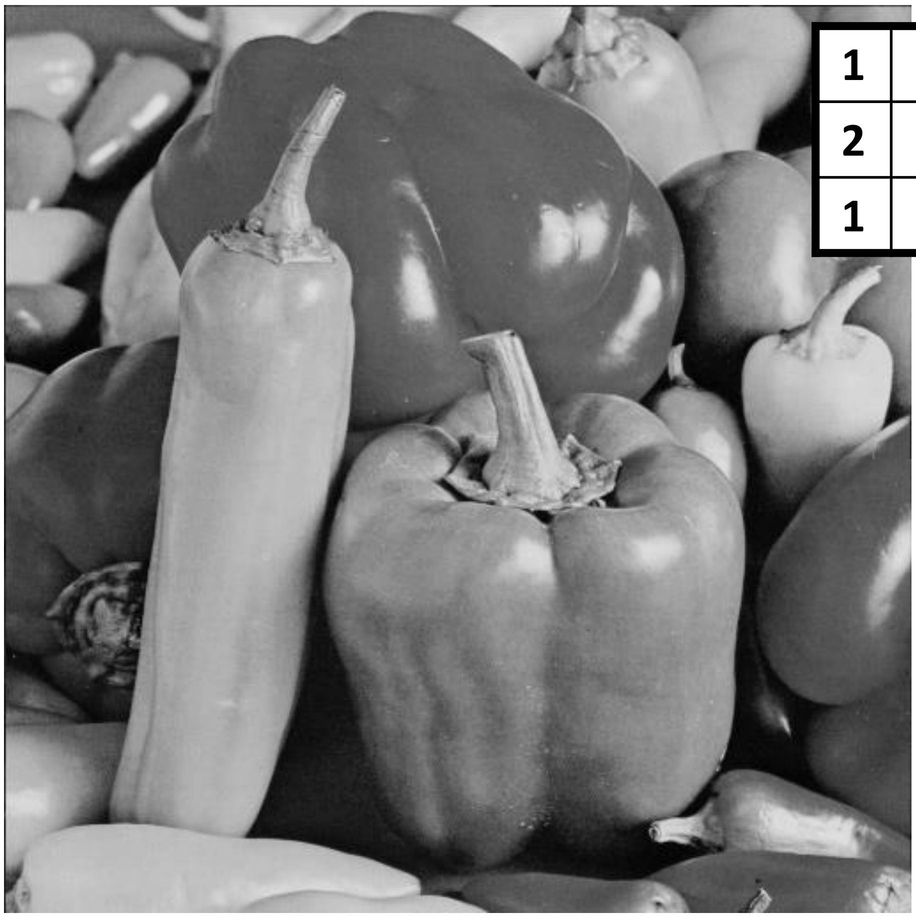




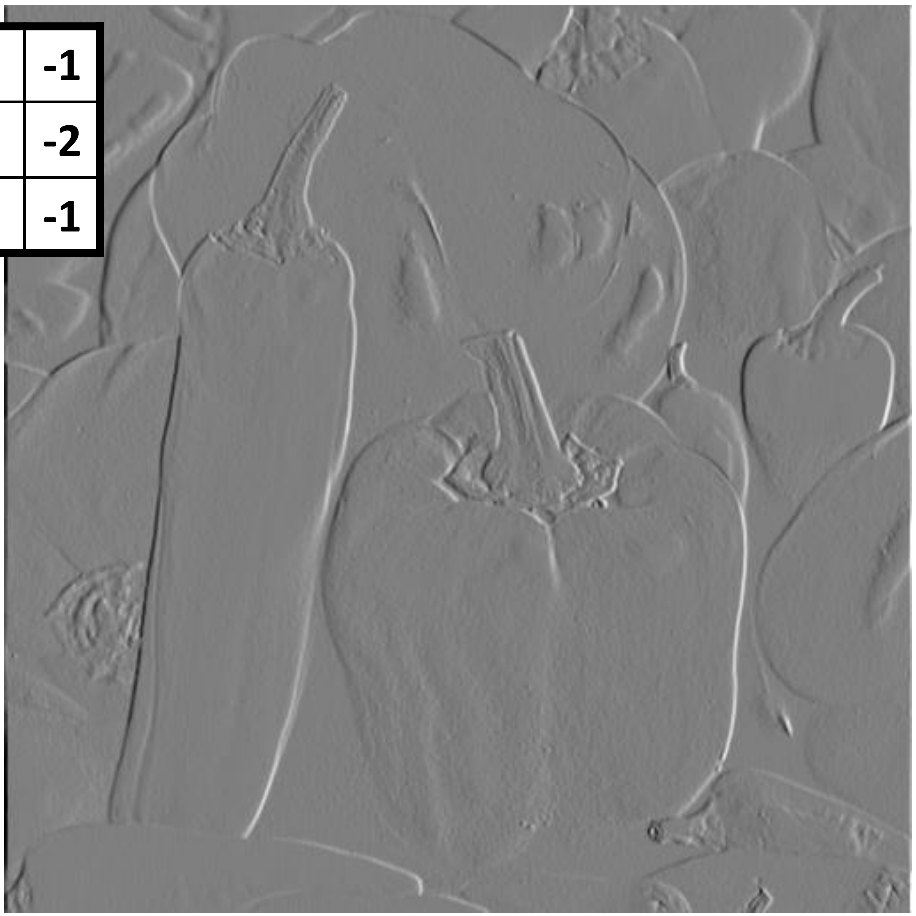
Average kernel



Common filters: Sobel

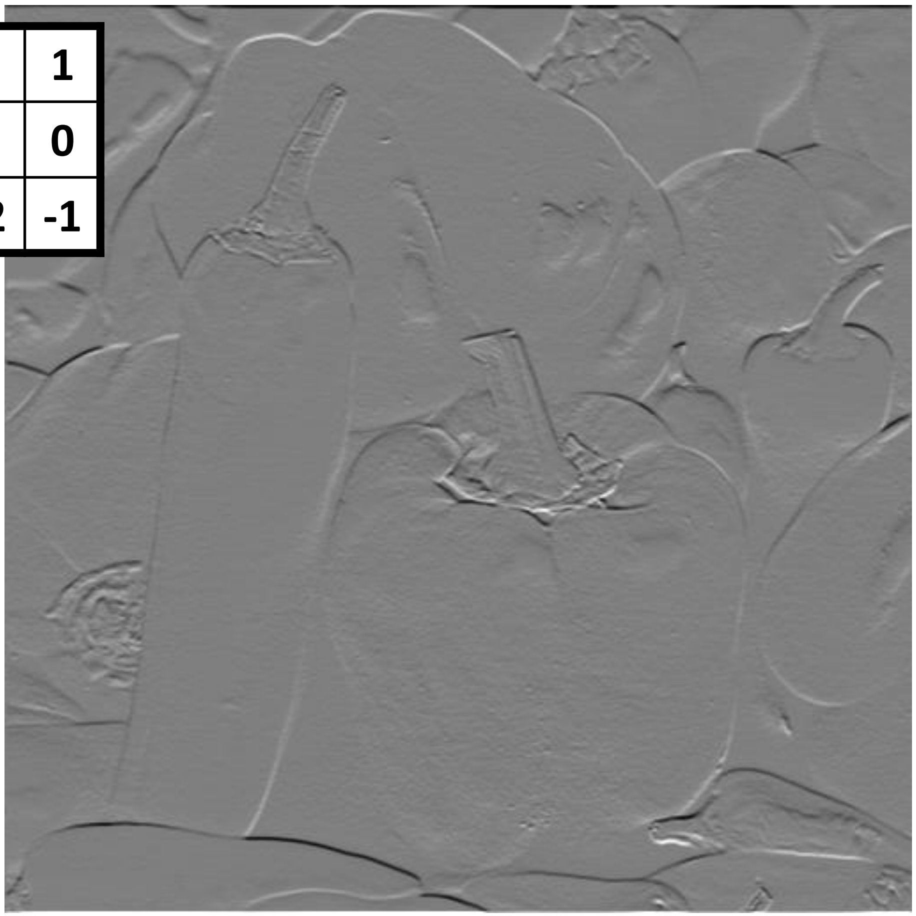


1	0	-1
2	0	-2
1	0	-1



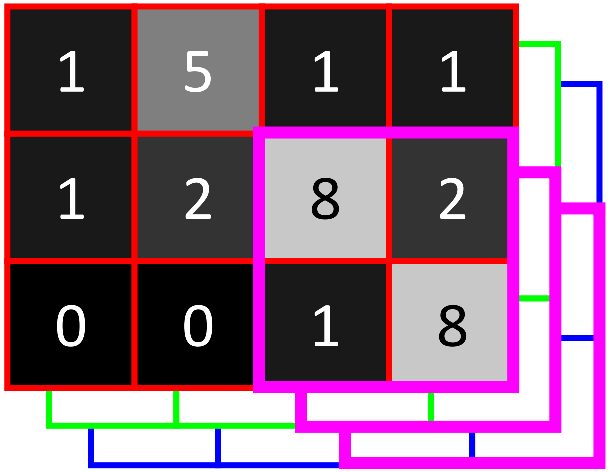
Common filters: Sobel

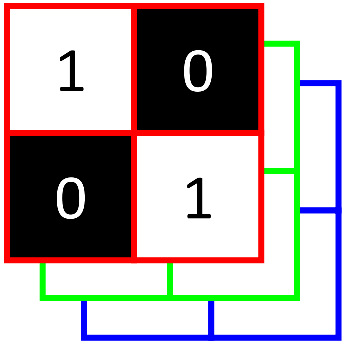
1	2	1
0	0	0
-1	-2	-1



What about colour?

- Consider a 3x4x3 image and 2x2x3 kernel



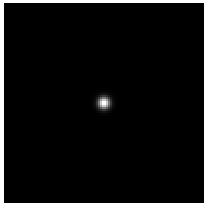


Convolution output?

Filter size vs. filter support

Convolved with:

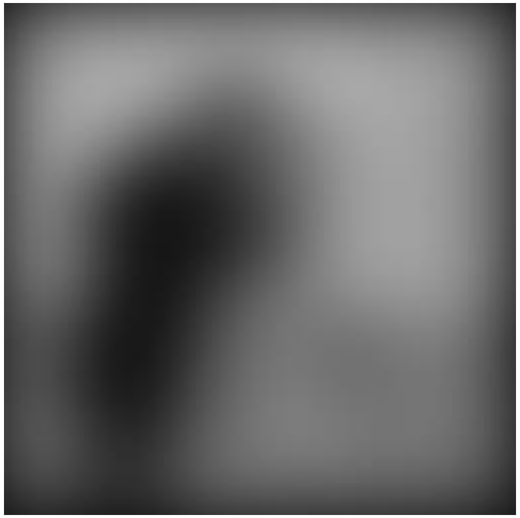
100 x 100 , σ = 2



100 x 100 , σ = 20

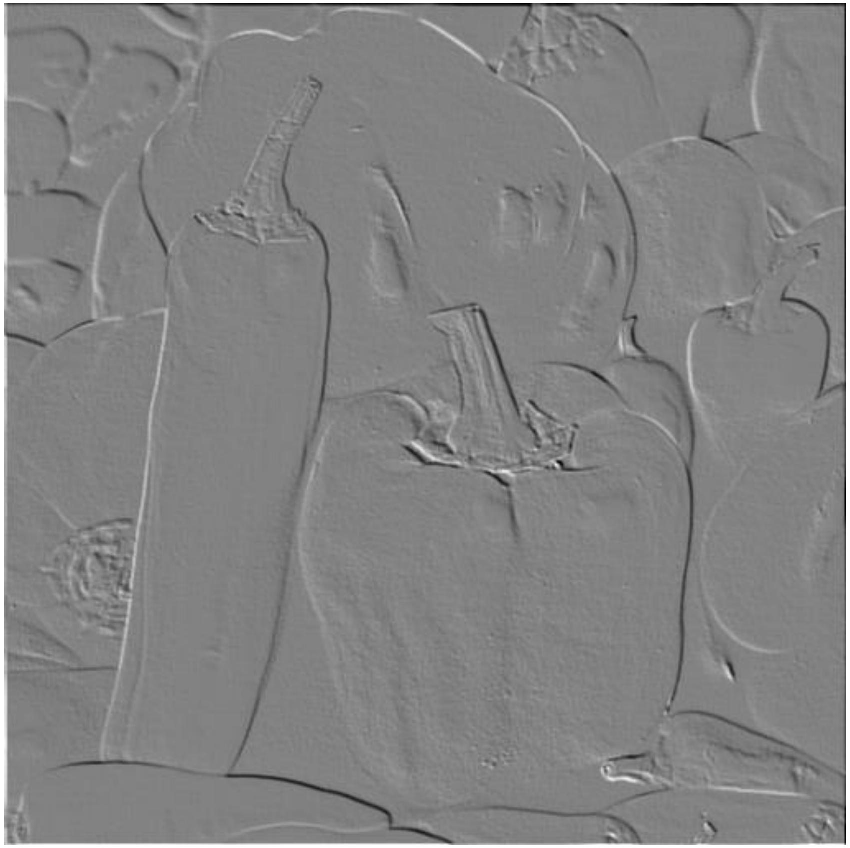
10 x 10, σ = 2

?



Designing filters

- How could you detect diagonal edges?



Designing filters

- How could you simulate (linear) motion blur?

Common filters

- Average/blur filters: average pixel values, blur the image
- Sharpening filters: subtract pixel from surround, increase fine detail
- Edge filters: compute difference between pixels, detect oriented edges in image

Filters in practice

Properties of linear filters

- Commutative: $f * h = h * f$
  - Theoretically, no difference between kernel and image
  - But most implementations do care about order
- Associative: $(f * h1) * h2 = f * (h1 * h2)$
  - Usually one option is faster than the other – allows for more efficient implementations
- Distributive over addition
  - $f * (h1 + h2) = (f * h1) + (f * h2)$
- Multiplication cancels out
  - $kf * h = f * kh = k(f * h)$

Efficient filtering

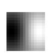
- Multiple filters: generally more efficient to combine 2D filters ($h1 * h2 * h3 \ldots$) and filter image just once

$*$  $=$

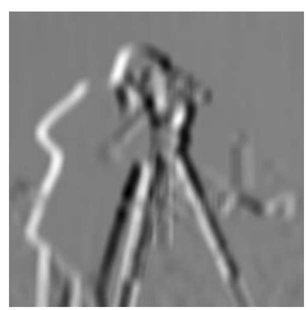
Gaussian blur filter

$*$  $=$

Horizontal derivative filter

$*$  $=$

Derivative-of-Gaussian filter



Efficient filtering

- Separable filters: generally more efficient to filter with two 1D filters than one 2D filter
- For example, the 2D Gaussian can be expressed as a product of two 1D Guassians (in x and y)

$$
\begin{aligned}
G_\sigma(x, y) &= \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2 + y^2}{2\sigma^2}} \\
&= \left(\frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{x^2}{2\sigma^2}}\right) \left(\frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{y^2}{2\sigma^2}}\right)
\end{aligned}
$$

Figure: D. Lowe

Separable filters

2D convolution
(center location only)

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{bmatrix}$$

The filter factors into a product of 1D filters:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Perform convolution along rows:

$$\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{bmatrix} = \begin{bmatrix} & 11 & \\ & 18 & \\ & 18 & \end{bmatrix}$$

Followed by convolution along the remaining column:

$$\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} & 11 & \\ & 18 & \\ & 18 & \end{bmatrix} = \begin{bmatrix} & & \\ & 65 & \\ & & \end{bmatrix}$$

Slide: K. Grauman

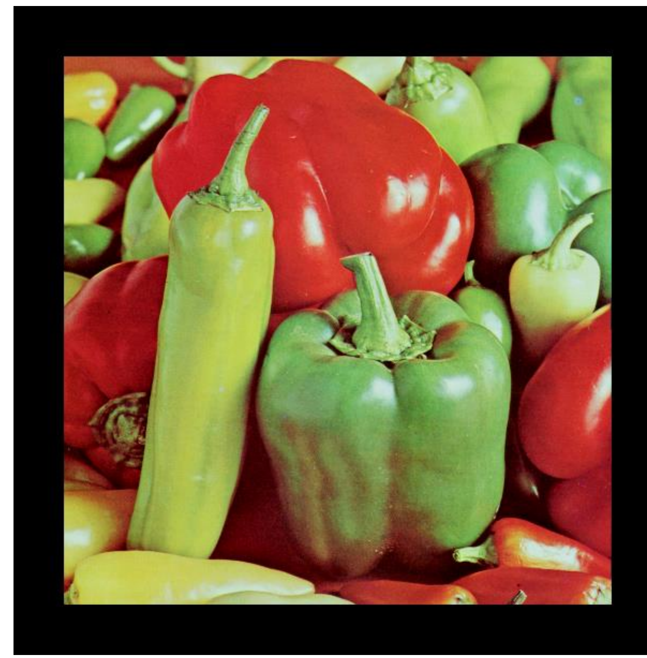
Convolution output size

- Valid convolution: the output image is smaller than the input image
- Why?

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 255 & 255 & 255 & 0 & 0 \\
0 & 255 & 0 & 0 & 0 & 255 & 0 \\
0 & 255 & 0 & 0 & 0 & 0 & 0 \\
0 & 255 & 0 & 0 & 0 & 255 & 0 \\
0 & 0 & 255 & 255 & 255 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
*
\begin{bmatrix}
1/9 & 1/9 & 1/9 \\
1/9 & 1/9 & 1/9 \\
1/9 & 1/9 & 1/9
\end{bmatrix}
=
\begin{bmatrix}
57 & 85 & 85 & 85 & 57 \\
85 & 113 & 85 & 85 & 57 \\
85 & 85 & 0 & 57 & 57 \\
85 & 113 & 85 & 85 & 57 \\
57 & 85 & 85 & 85 & 57
\end{bmatrix}
$$

Border handling

- Pad with constant value



 Filter: Gaussian blur

Border handling

- Wrap image



 Filter: Gaussian blur

Border handling

- Clamp / replicate the border value

 Filter: Gaussian blur

Border handling

- Reflect image

 Filter: Gaussian blur

Practical considerations

- Think about how to implement filters efficiently
  - Images are big, so efficient filtering can save a lot of time!
- Think about how to handle borders
  - No one-size-fits-all solution
  - Wrap is ideal for tiling textures (but not photos)
  - Clamp/replicate tends to work well for photos

Summary

- Linear filters: first step of almost all computer vision systems
- Linear filters are just a first step – you can’t build complex feature detectors from just linear filters