Computer Vision (CS 419/619)

Image filtering in Spatial Domain

Dr. Puneet Gupta

Image filtering

- Image filtering: compute function of local neighborhood at each position
  - Same function applied at each position
  - Output and input image are typically the same size

- Really important!
  - Enhance images
    - Denoise, resize, increase contrast, etc.
  - Extract information from images
    - Texture, edges, distinctive points, etc.
  - Detect patterns
    - Template matching
  - Deep Convolutional Networks

Image Denoising

Why would images have noise?

- Sensor noise: Sensors count photons: noise in count
- Camera Parameters
- And so on…

Let us assume noise at a pixel is

- independent of other pixels
- distributed according to a Gaussian distribution
    - i.e., low noise values are more likely than high noise values

Noise reduction

- Nearby pixels are likely to belong to same object
    - thus likely to have similar color
- Replace each pixel by average of neighbors

1D Filtering: Moving Average

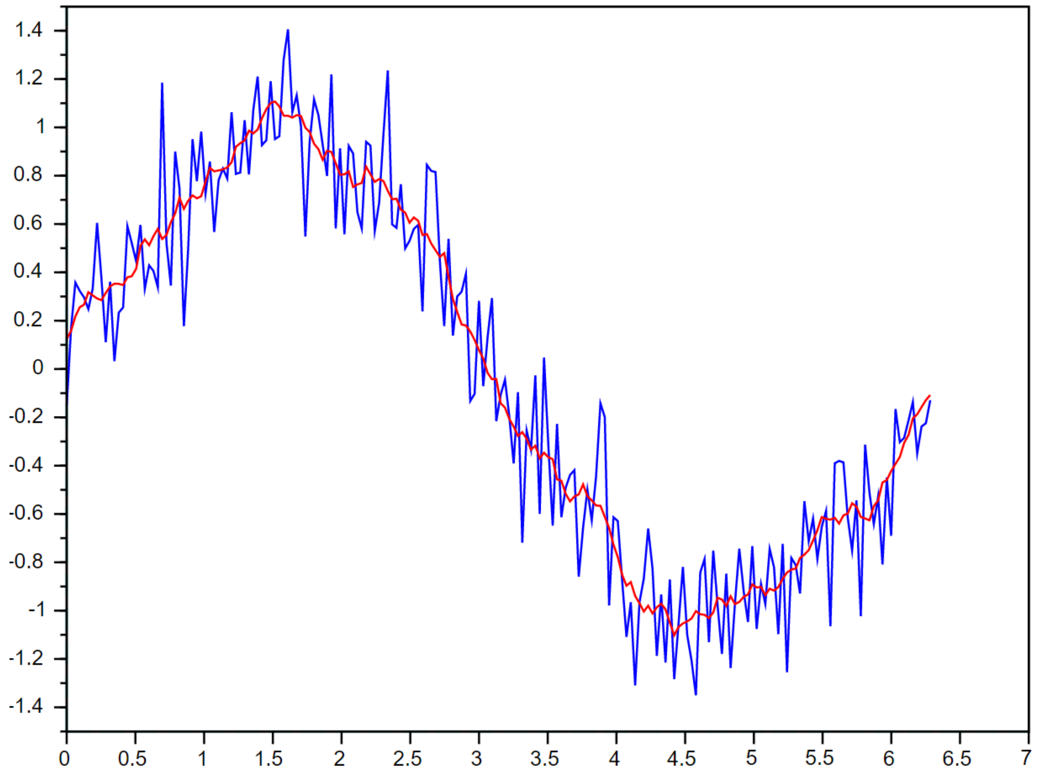
$$I \in \mathcal{R}^{m\times 1}$$

$$h[n] = \frac{1}{k} \sum_{i=n-k+1}^{n} I[i]$$

Ignore boundaries for the moment

`Window size ‘k’`

$$h[n] = \frac{1}{k} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}^T I[n-k+1:n]$$



$$I[n-k/2:n+k/2]$$

Different window

https://en.wikipedia.org/wiki/Moving_average

2D Filtering

Compute function of local neighborhood
at each position:

```
h=output          f=filter   I=image
```

$$h[m,n] = \sum_{k,l} f[k,l]\, I[m+k, n+l]$$

```
2d coords=k,l   2d coords=m,n
```

$$[\quad]\qquad [\ ]\qquad [\quad]$$

Note: Filter is often called the ‘kernel’

Image Kernels explained visually (setosa.io)

Box/Mean Filter

What does it do?

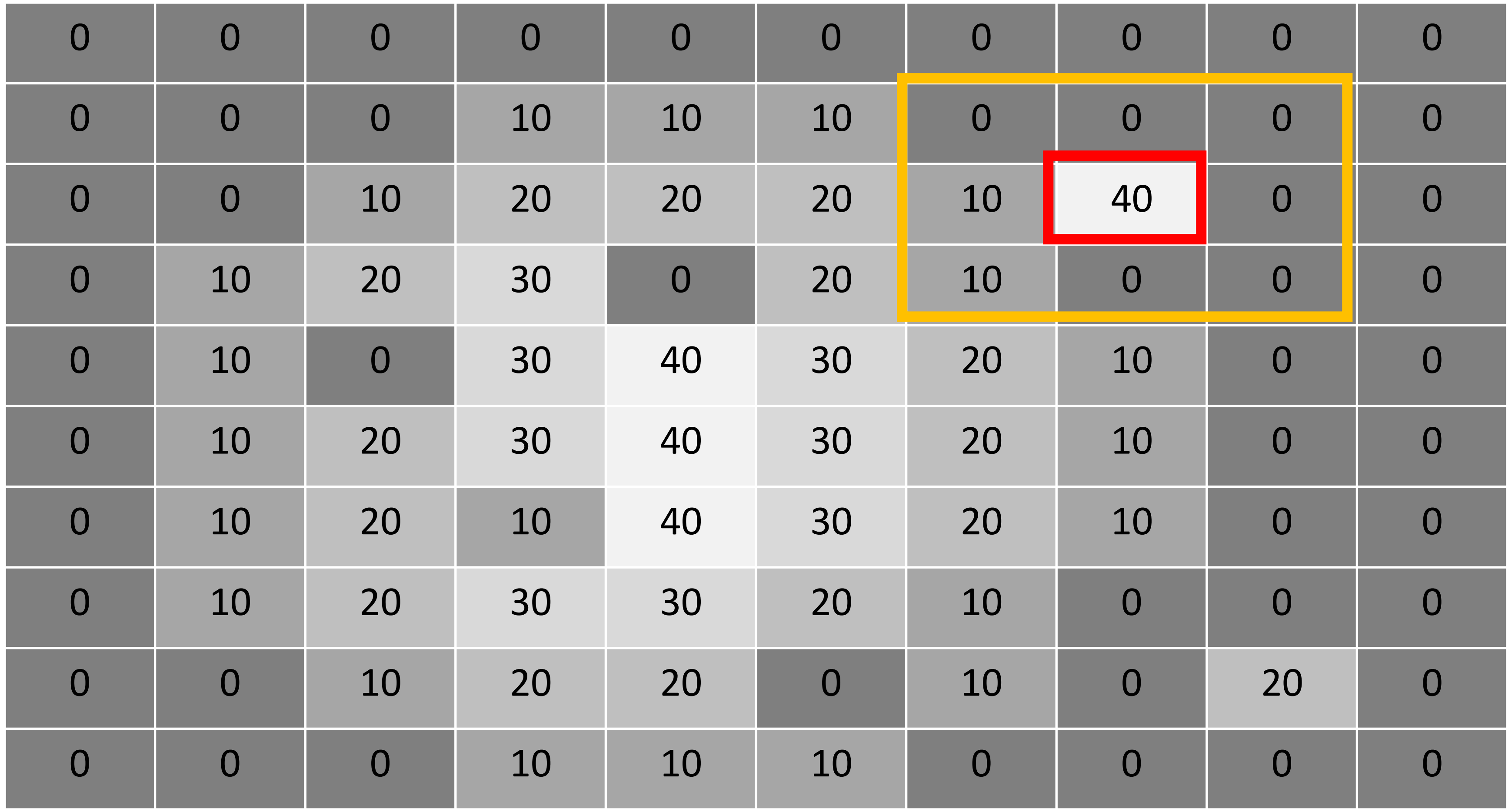
- Replaces each pixel with an average of its neighborhood
- Achieve smoothing effect (remove ‘sharp’ features)
- Why does it sum to one?

$$f[\cdot\,,\cdot\,]$$

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

David Lowe

Mean filtering



0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0
0	0	10	20	20	20	10	40	0	0
0	10	20	30	0	20	10	0	0	0
0	10	0	30	40	30	20	10	0	0
0	10	20	30	40	30	20	10	0	0
0	10	20	10	40	30	20	10	0	0
0	10	20	30	30	20	10	0	0	0
0	0	10	20	20	0	10	0	20	0
0	0	0	10	10	10	0	0	0	0

$$(0 + 0 + 0 + 10 + 40 + 0 + 10 + 0 + 0)/9 = 6.66$$

Mean filtering

0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0
0	0	10	20	20	20	10	40	0	0
0	10	20	30	0	20	10	0	0	0
0	10	0	30	40	30	20	10	0	0
0	10	20	30	40	30	20	10	0	0
0	10	20	10	40	30	20	10	0	0
0	10	20	30	30	20	10	0	0	0
0	0	10	20	20	0	10	0	20	0
0	0	0	10	10	10	0	0	0	0

(0 + 0 + 0 + 0 + 0 + 10 + 0 + 0 + 0 + 0 + 20 + 10 + 40 + 0 + 0 + 20 + 10 + 0 + 0 + 0 + 30 + 20 + 10 + 0 + 0)/25 = 6.8

Mean filtering

0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0
0	0	10	20	20	20	10	40	0	0
0	10	20	30	0	20	10	0	0	0
0	10	0	30	40	30	20	10	0	0
0	10	20	30	40	30	20	10	0	0
0	10	20	10	40	30	20	10	0	0
0	10	20	30	30	20	10	0	0	0
0	0	10	20	20	0	10	0	20	0
0	0	0	10	10	10	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	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$(0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 10)/9 = 1.11$$

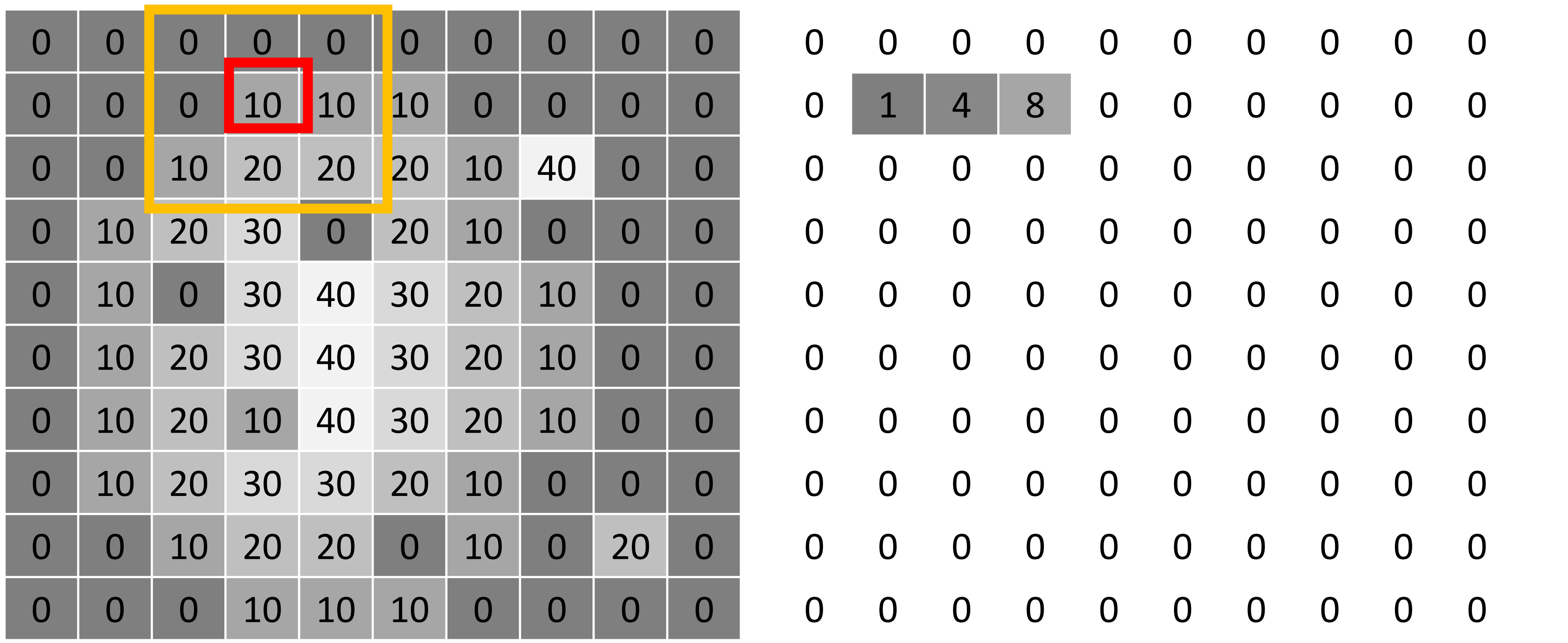
Mean filtering

0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0
0	0	10	20	20	20	10	40	0	0
0	10	20	30	0	20	10	0	0	0
0	10	0	30	40	30	20	10	0	0
0	10	20	30	40	30	20	10	0	0
0	10	20	10	40	30	20	10	0	0
0	10	20	30	30	20	10	0	0	0
0	0	10	20	20	0	10	0	20	0
0	0	0	10	10	10	0	0	0	0

0	0	0	0	0	0	0	0	0	0
0	1	4	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$(0 + 0 + 0 + 0 + 0 + 10 + 0 + 10 + 20)/9 = 4.44$$

Mean filtering

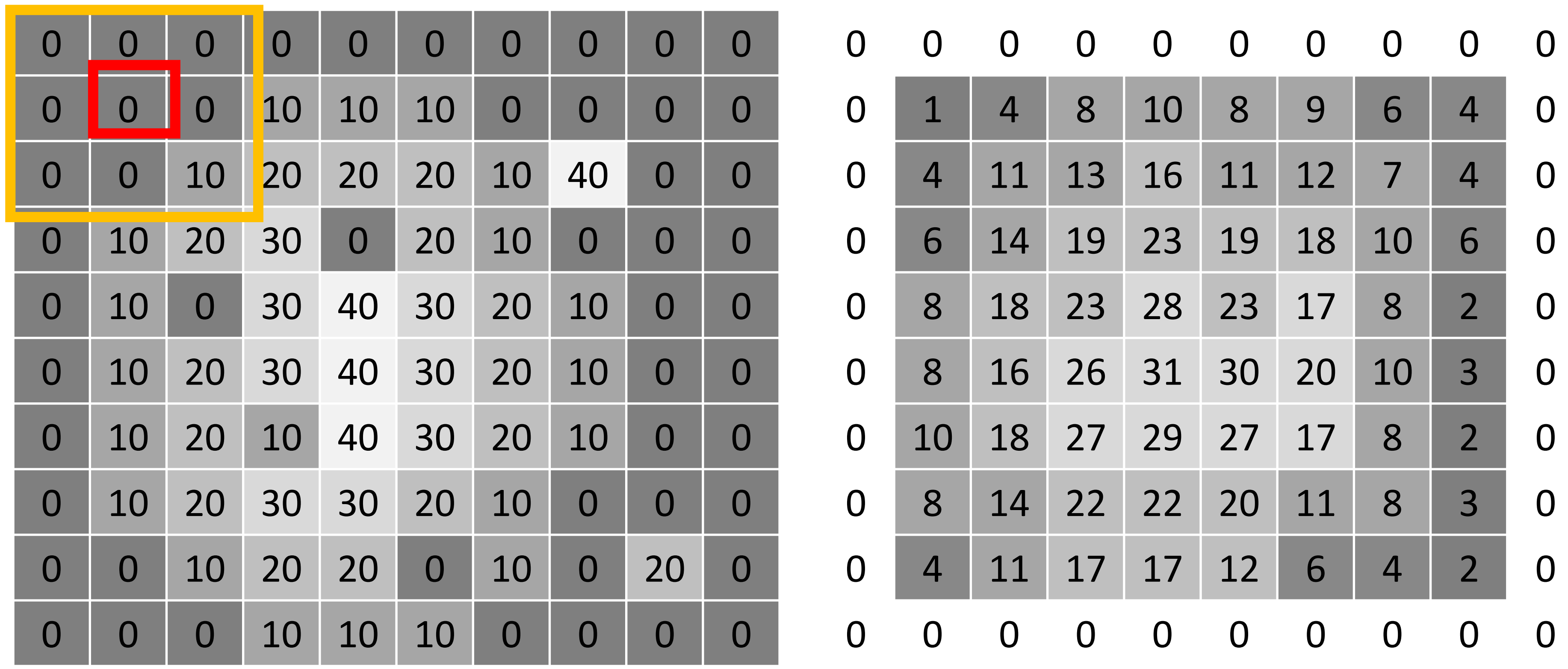


0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0
0	0	10	20	20	20	10	40	0	0
0	10	20	30	0	20	10	0	0	0
0	10	0	30	40	30	20	10	0	0
0	10	20	30	40	30	20	10	0	0
0	10	20	10	40	30	20	10	0	0
0	10	20	30	30	20	10	0	0	0
0	0	10	20	20	0	10	0	20	0
0	0	0	10	10	10	0	0	0	0

0	0	0	0	0	0	0	0	0	0
0	1	4	8	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$(0 + 0 + 0 + 0 + 10 + 10 + 10 + 20 + 20)/9 = 7.77$$

Mean filtering

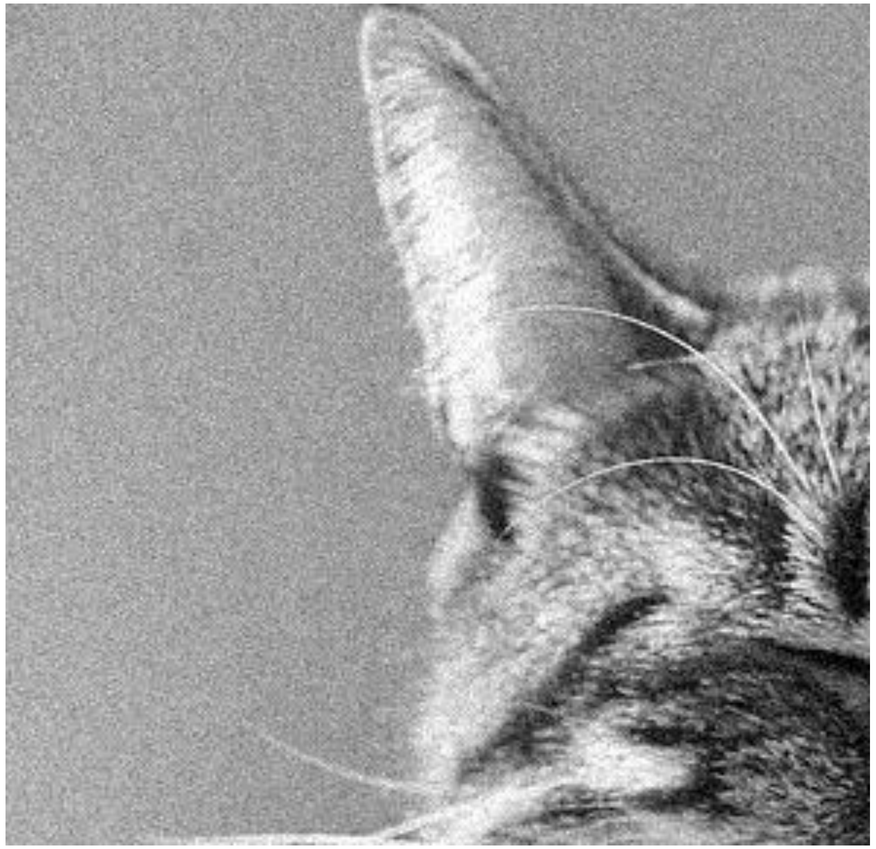


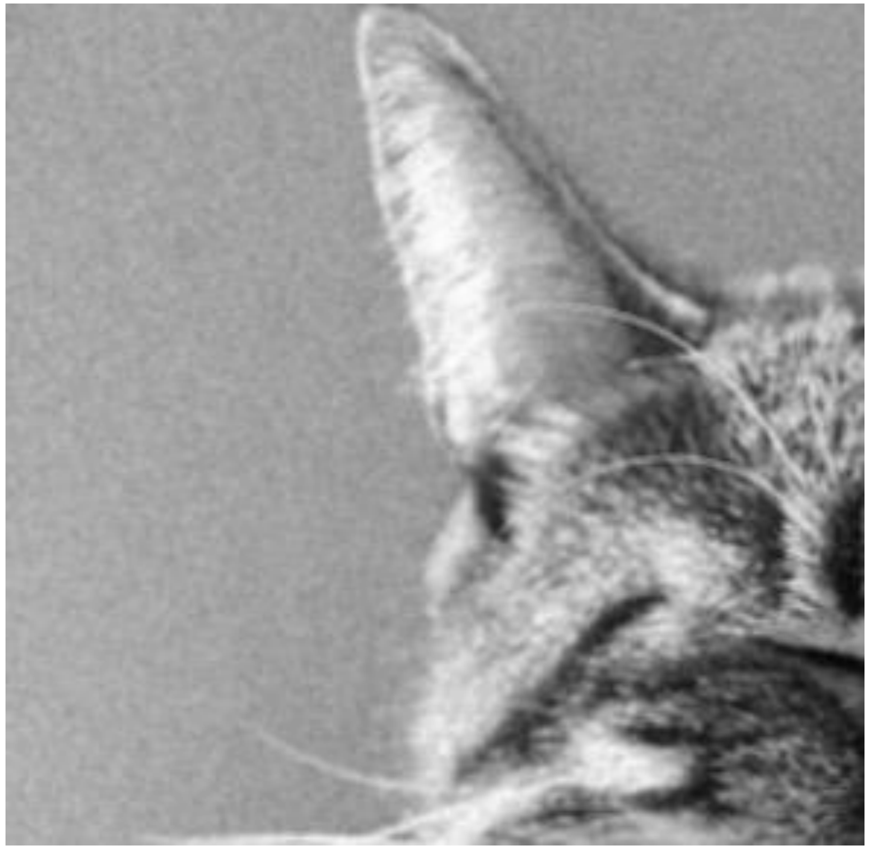
0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0
0	0	10	20	20	20	10	40	0	0
0	10	20	30	0	20	10	0	0	0
0	10	0	30	40	30	20	10	0	0
0	10	20	30	40	30	20	10	0	0
0	10	20	10	40	30	20	10	0	0
0	10	20	30	30	20	10	0	0	0
0	0	10	20	20	0	10	0	20	0
0	0	0	10	10	10	0	0	0	0

0	0	0	0	0	0	0	0	0	0
0	1	4	8	10	8	9	6	4	0
0	4	11	13	16	11	12	7	4	0
0	6	14	19	23	19	18	10	6	0
0	8	18	23	28	23	17	8	2	0
0	8	16	26	31	30	20	10	3	0
0	10	18	27	29	27	17	8	2	0
0	8	14	22	22	20	11	8	3	0
0	4	11	17	17	12	6	4	2	0
0	0	0	0	0	0	0	0	0	0

The box filter blurs the image because for each pixel, the values of its neighbors is averaged with it, smearing out the intensity values. Certain image filtering operations need to preserve mean image intensity. Box filter, a blurring filter, does that (reason why it sums to 1). Blurring only makes sense if the mean image intensity is preserved.

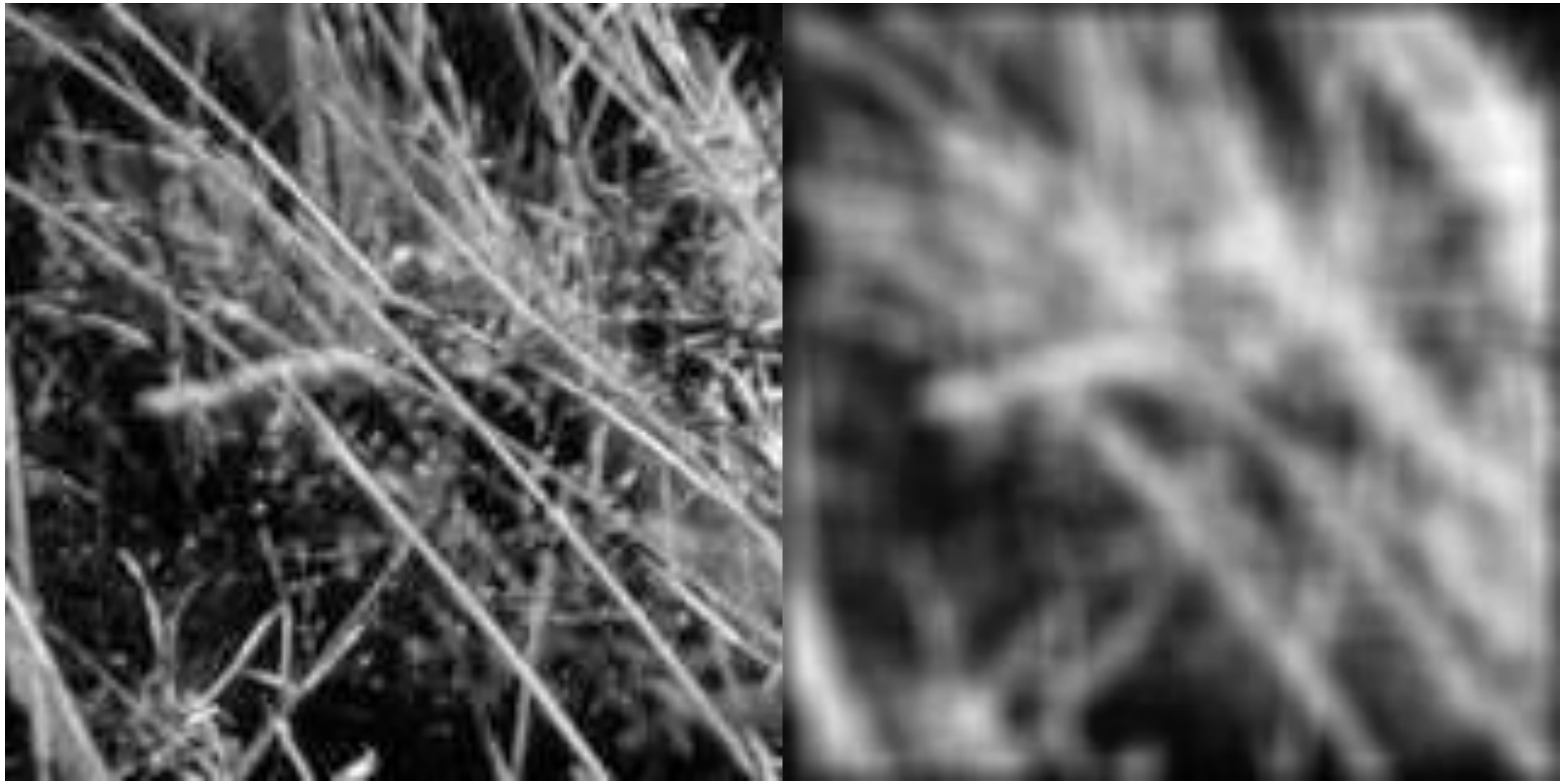
Noise reduction using mean filtering





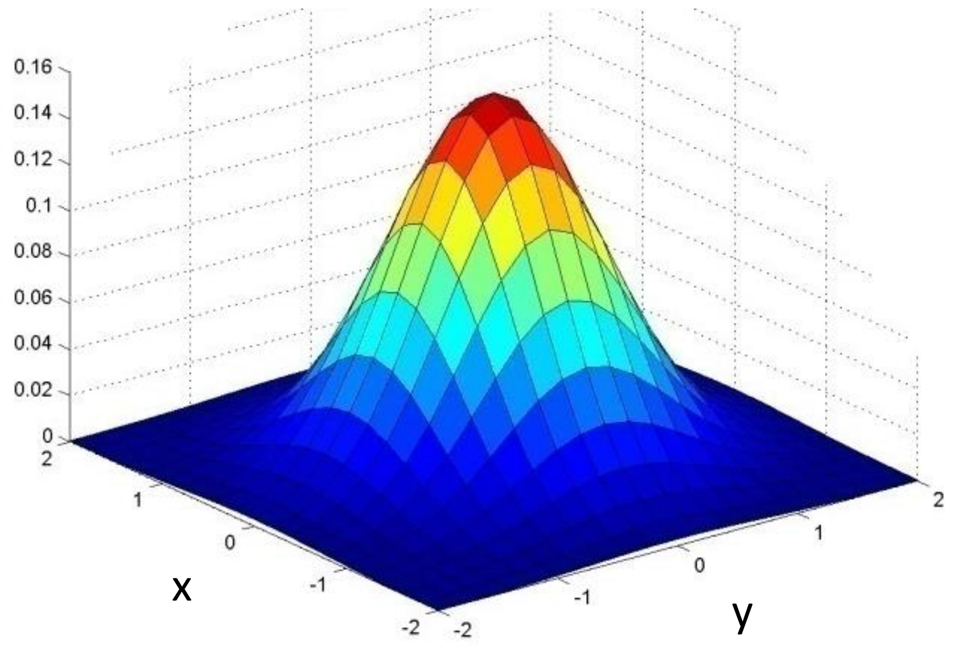
Smoothing with Box Filter

$$f[\cdot\,,\cdot\,]\,\frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline\end{array}$$



David Lowe

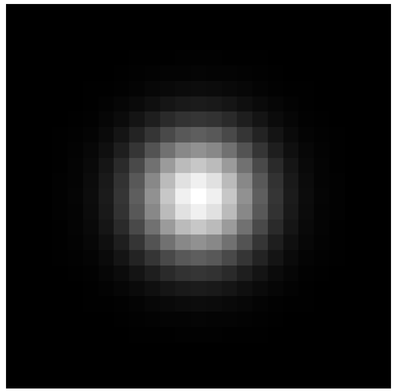
Gaussian Filter



$$\begin{array}{c|ccccc|c} & & & x & & & \\ \hline & 0.003 & 0.013 & 0.022 & 0.013 & 0.003 & \\ & 0.013 & 0.059 & 0.097 & 0.059 & 0.013 & \\ & 0.022 & 0.097 & 0.159 & 0.097 & 0.022 & y \\ & 0.013 & 0.059 & 0.097 & 0.059 & 0.013 & \\ & 0.003 & 0.013 & 0.022 & 0.013 & 0.003 & \\ \hline \end{array}$$

Kernel size 5 x 5,
Standard deviation σ = 1

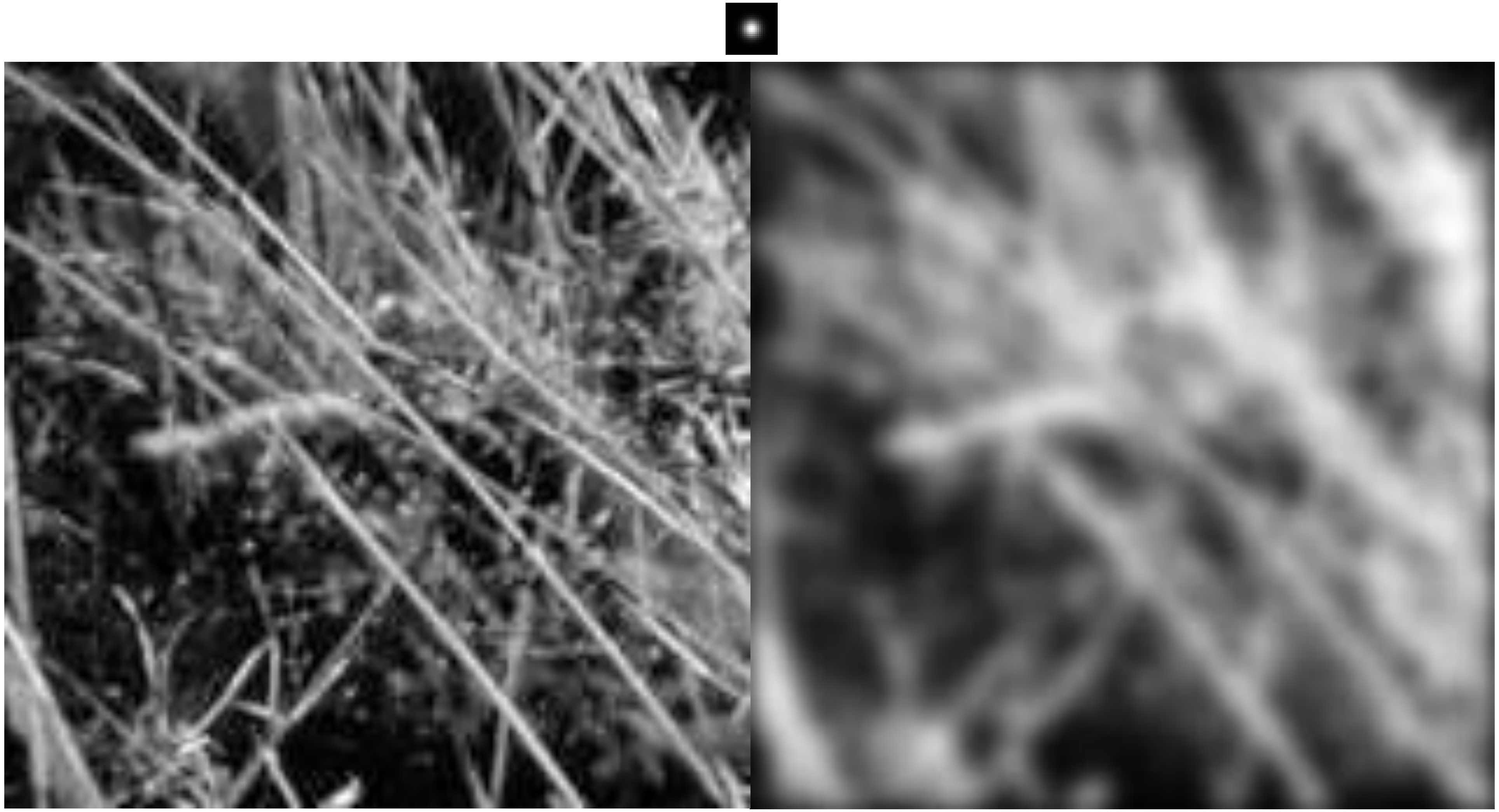
Viewed from top



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

Christopher Rasmussen

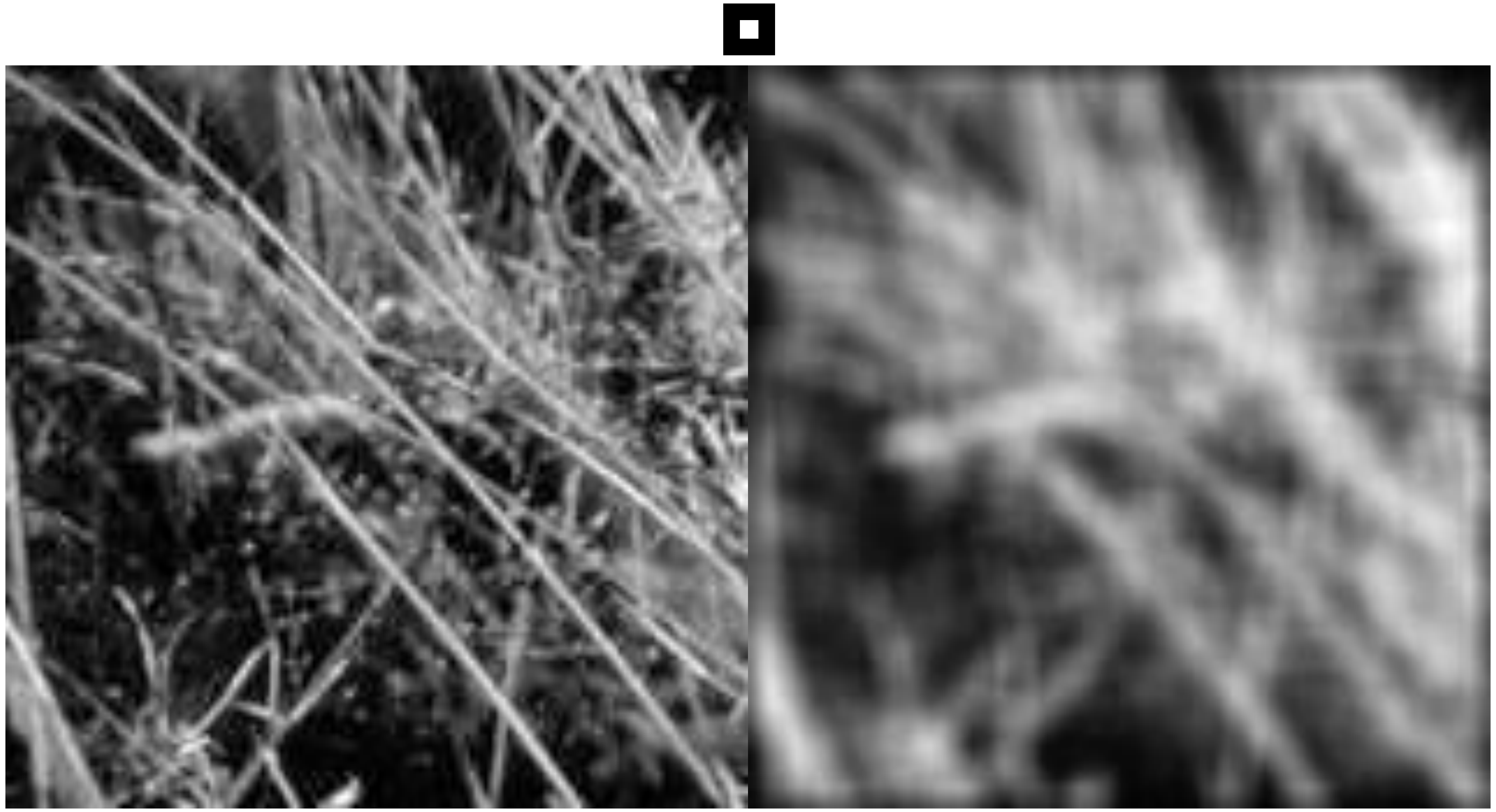
Smoothing with Gaussian Filter



James Hays

Smoothing with Box Filter

$$f[\cdot\,,\cdot\,]\ \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$



David Lowe

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

Filtered
(no change)

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0

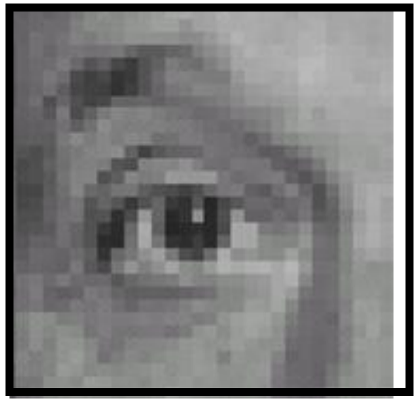
?

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0



Shifted left
By 1 pixel

Source: D. Lowe

Practice with linear filters

Original

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

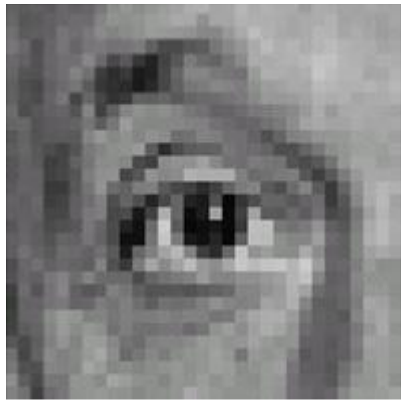
(Note that filter sums to 1)

Source: D. Lowe

Practice with linear filters

Original

$$\begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 2 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} - \frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$



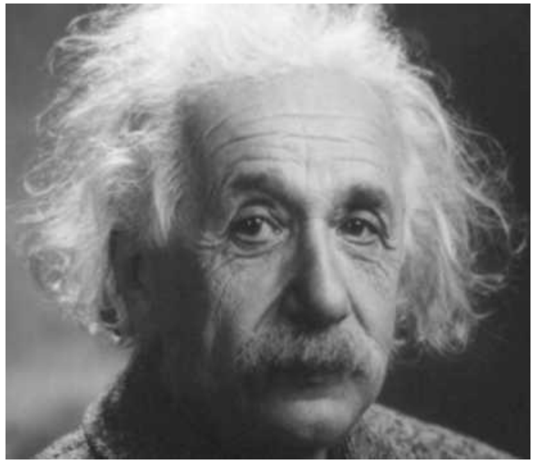
Sharpening filter

- Accentuates differences with local average

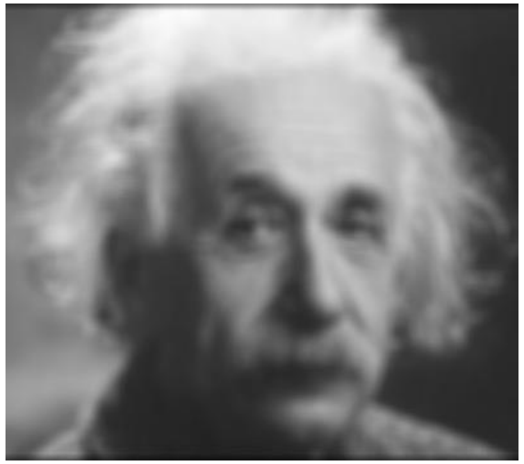
Source: D. Lowe

Sharpening

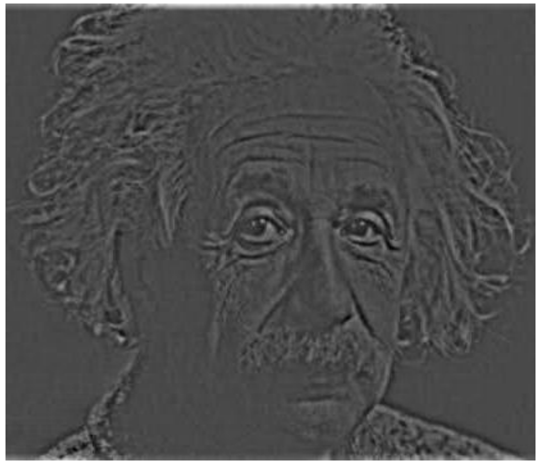
- What does blurring take away?



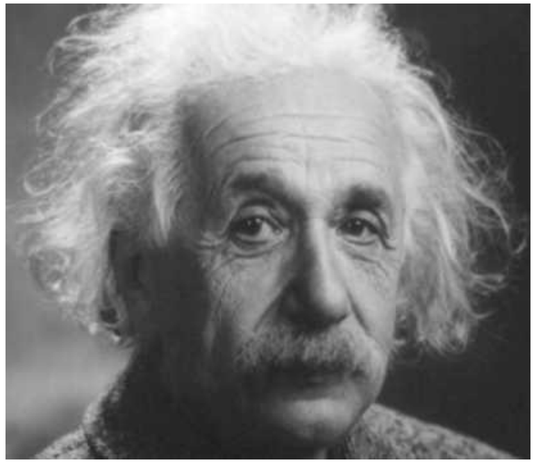
–



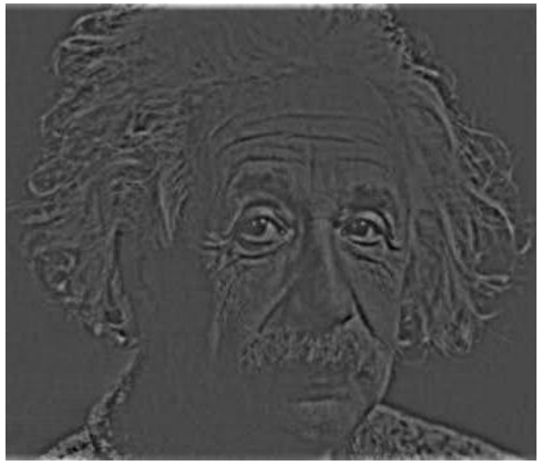
=



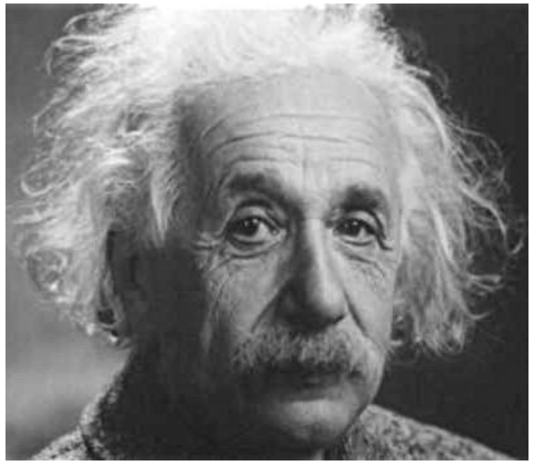
Let's add it back:



+ α



=



Sharpening

$$
\begin{aligned}
f_{sharp} &= f + \alpha(f - f_{blur}) \\
&= (1+\alpha)f - \alpha f_{blur} \\
&= (1+\alpha)(w * f) - \alpha(v * f)
\end{aligned}
$$

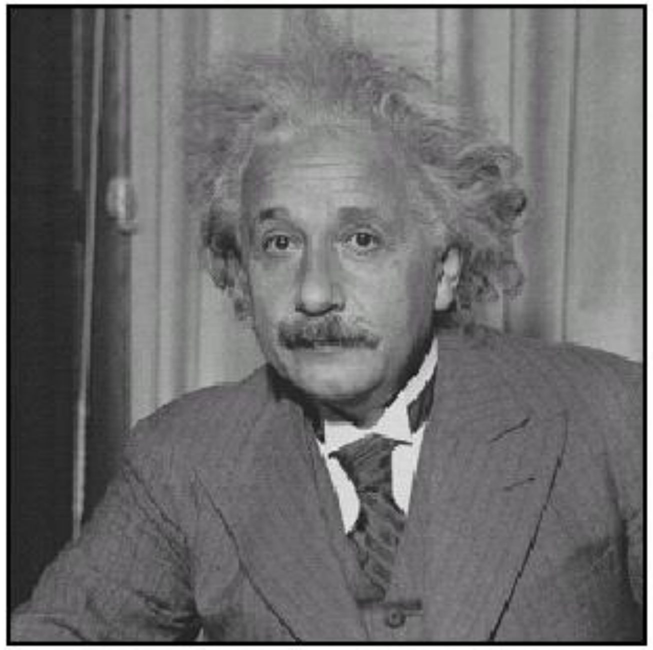
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

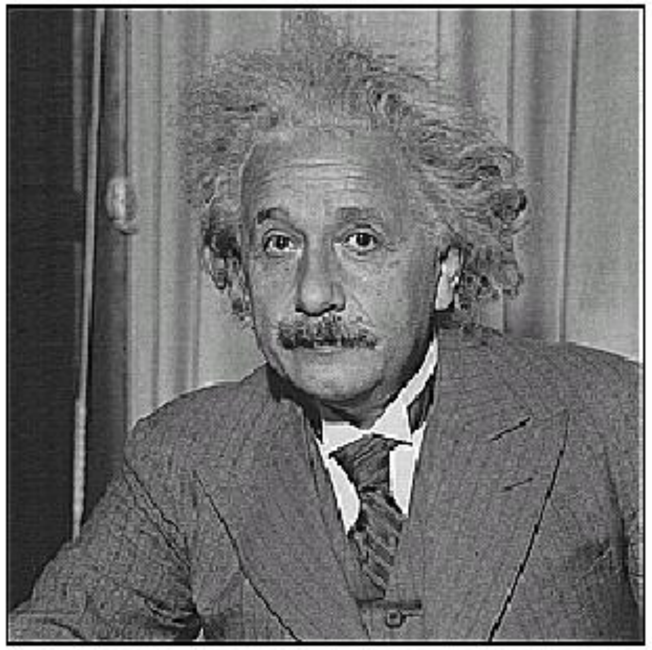
Our previous filter

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

$$= ((1+\alpha)w - \alpha v) * f$$

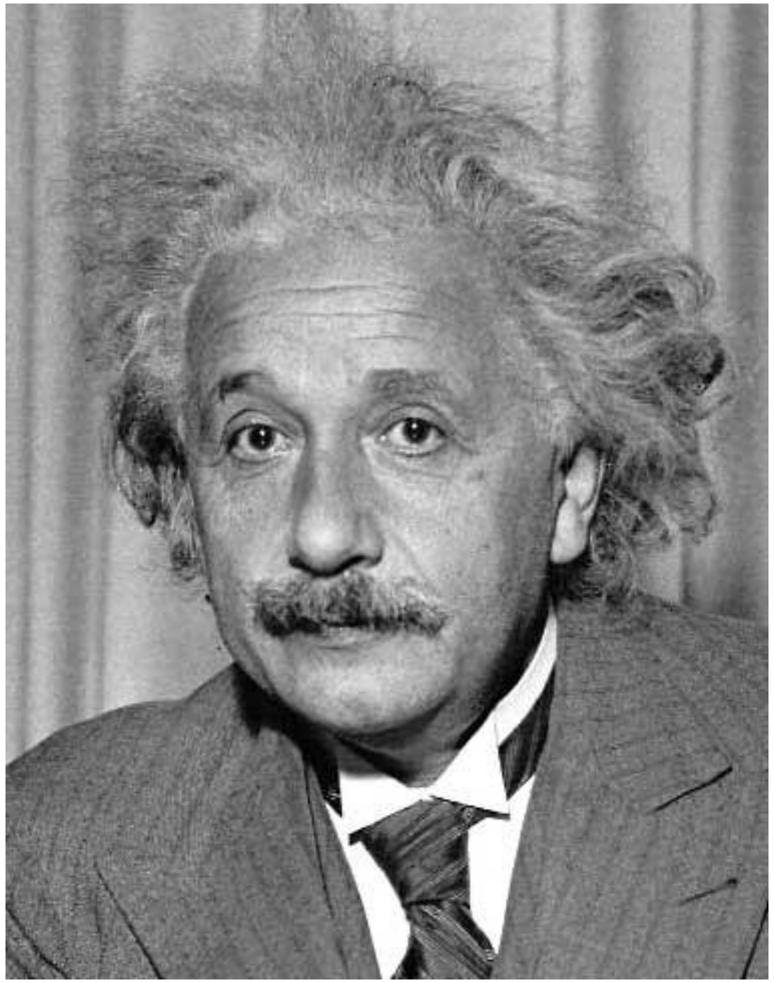
Sharpening


before


after

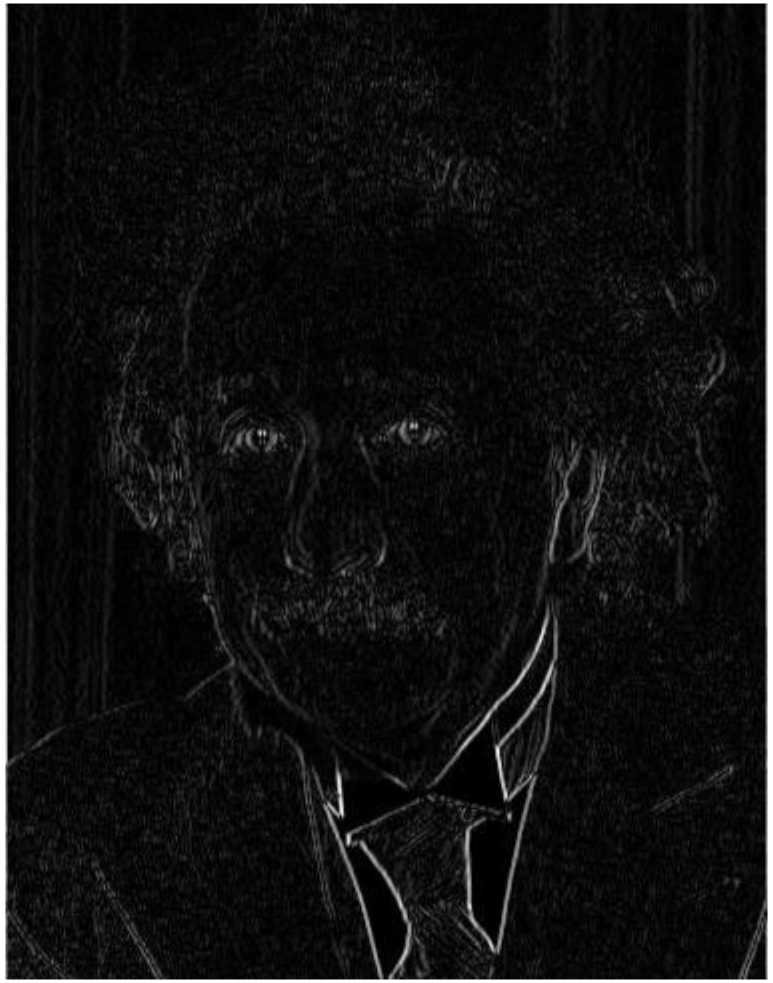
Source: D. Lowe

Edge filters



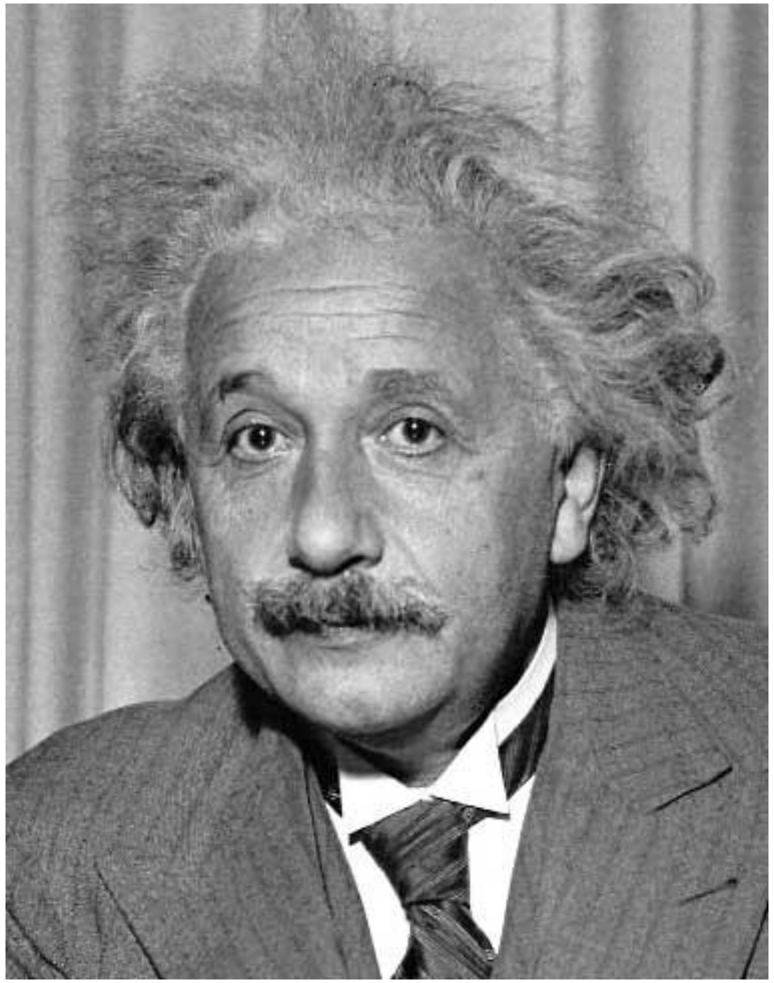
1	0	-1
2	0	-2
1	0	-1

Sobel


Vertical Edge
(absolute value)

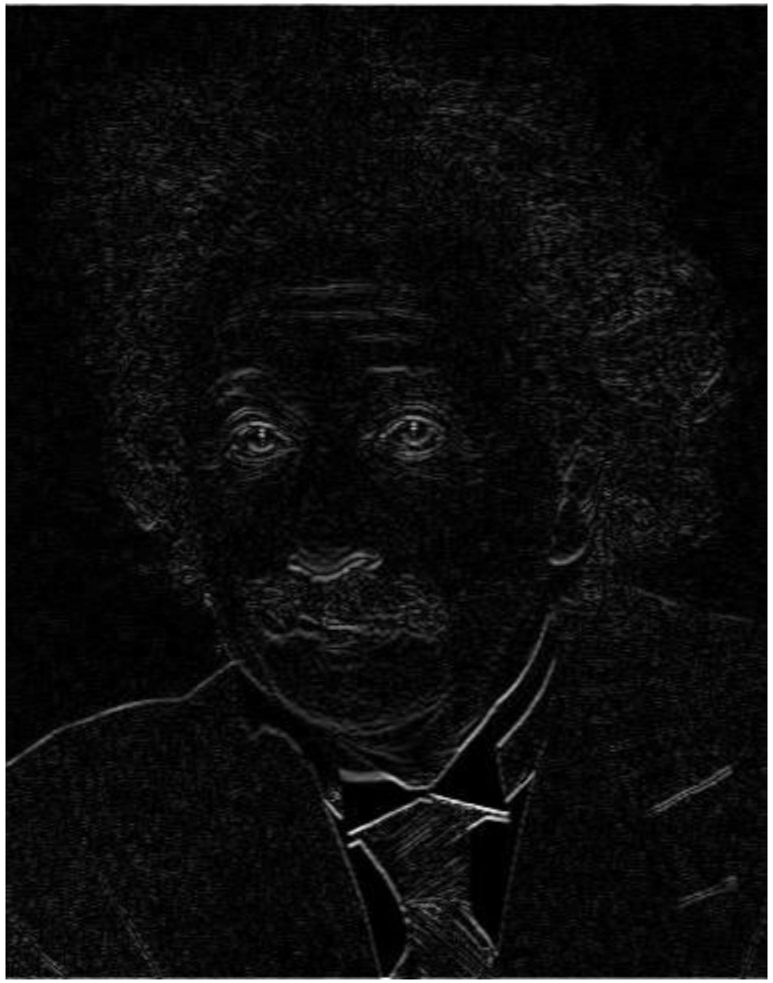
Notice that this filter sums to 0.

Edge filters



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

Sobel



Horizontal Edge
(absolute value)

Sobel Filter

- What is the 1 2 1 pattern?
  - Binomial distribution or “triangle filter”
  - Discrete approximate to Gaussian; fast for integer mathematics
  - Smooths along edge

- What happens to negative numbers?
  - let’s take the absolute value (the magnitude)

- For visualization:
  - Shift image + 0.5
  - If gradients are small, scale edge response

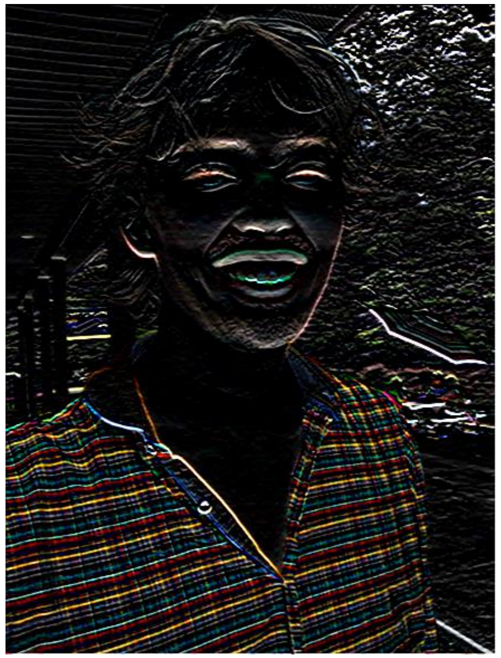
1	2	1
0	0	0
-1	-2	-1

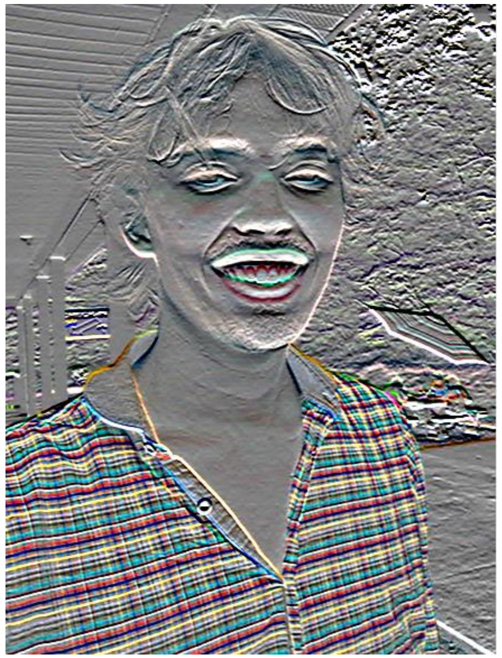
Sobel

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

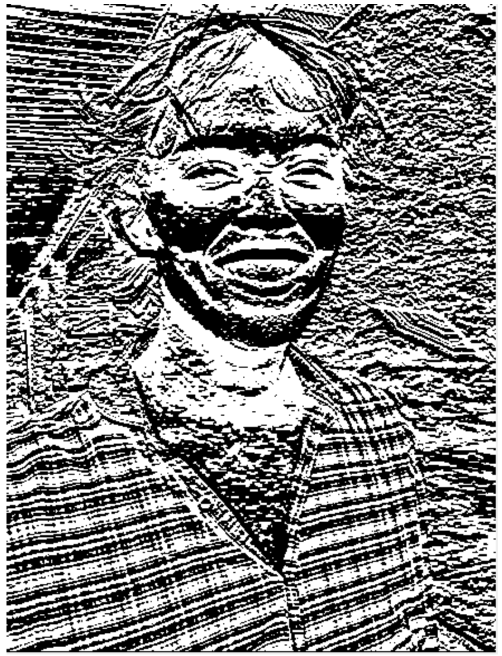
Sobel

```
>> I = img_to_float32( io.imread( 'luke.jpg' ) )
>> h = convolve2d( I, sobelKernel )

>> plt.imshow( h )

>> plt.imshow( h + 0.5 )
```

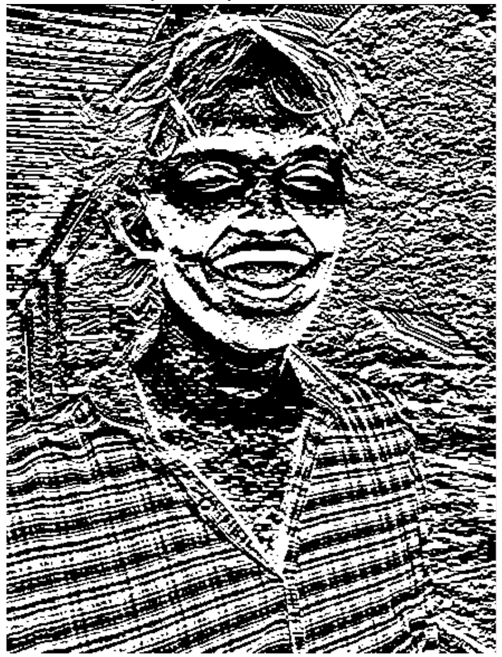




`h(:,:,1) < 0`



`h(:,:,1) > 0`



Another basic gradient/edge filters

Horizontal Gradient

0	0	0
-1	0	1
0	0	0

or

-1	0	1

Vertical Gradient

0	-1	0
0	0	0
0	1	0

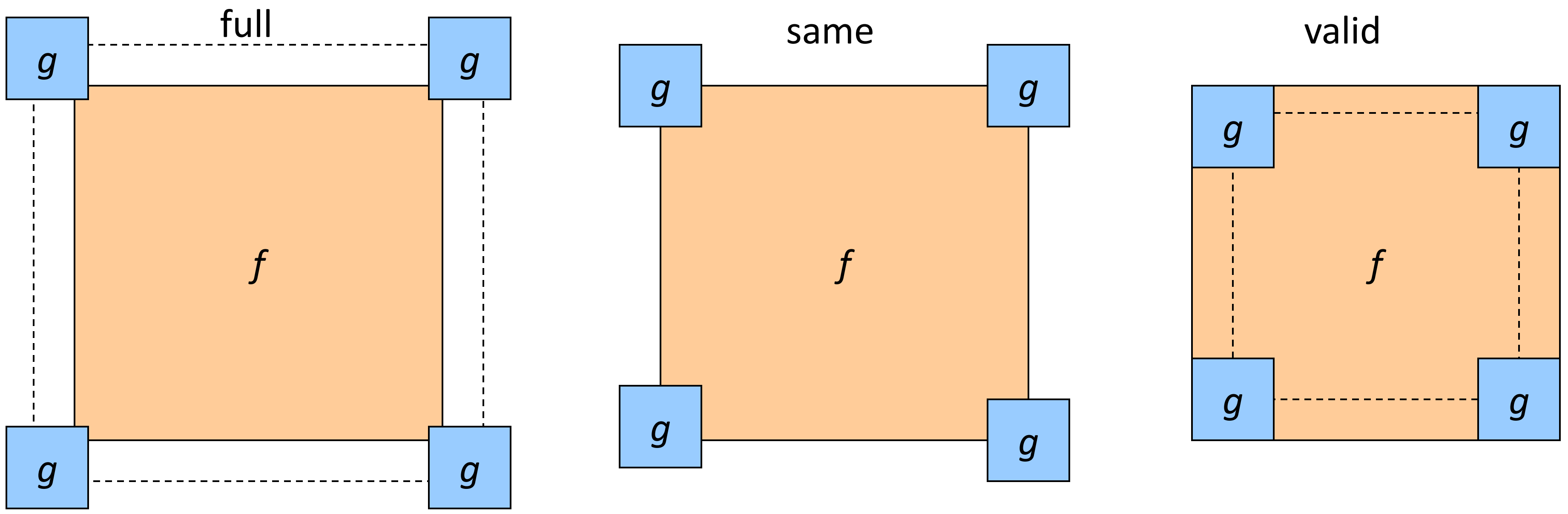
or

-1
0
1

Filters in Practice

What is the size of the output?

- *shape* = 'full': output size is sum of sizes of f and g
- *shape* = 'same': output size is same as f
- *shape* = 'valid': output size is difference of sizes of f and g



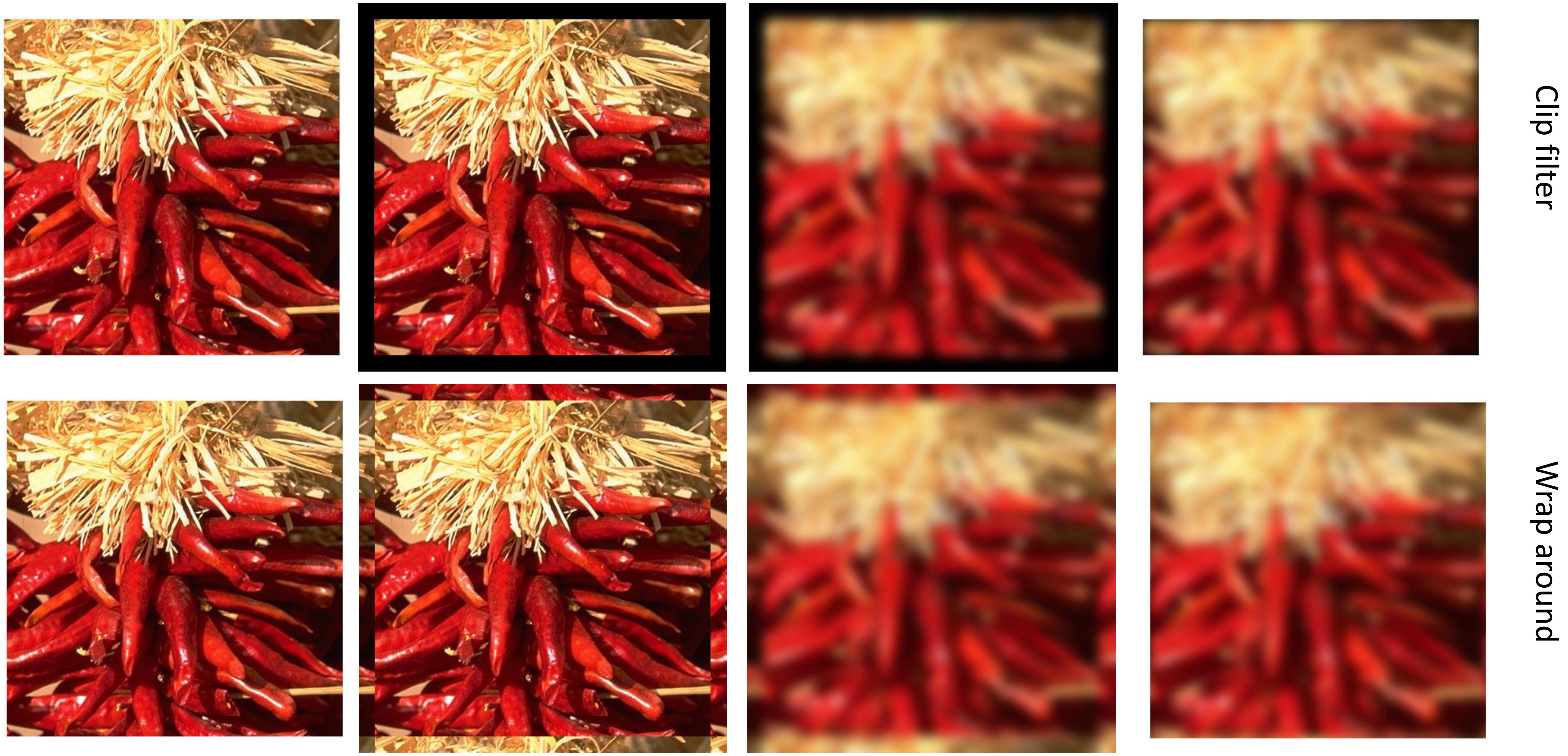
Source: S. Lazebnik

Filters in Practice

What about near the edge?

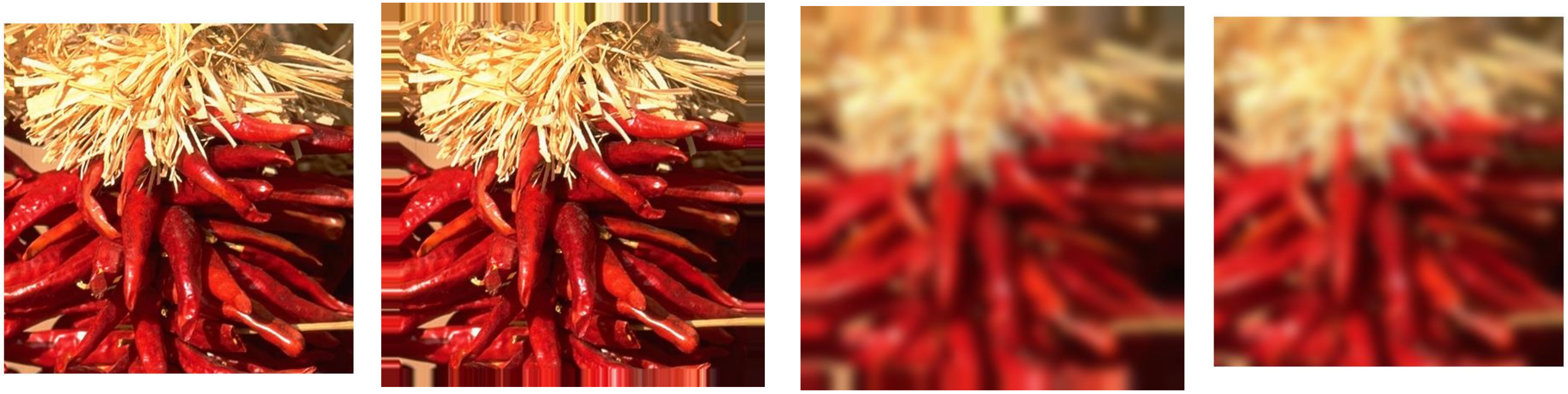
- The filter window falls off the edge of the image
- Need to extrapolate
- Methods:
  - Clip filter: any value outside of the image is set to 0
  - Wrap around: copy the values near the opposite edge (e.g. copying the bottom portion of the image to the outside of the top boundary)
  - Copy edge: copy the pixel value of the nearest edge
  - Reflect across edge: treat out-of-bounds regions as ‘mirrors’ that reflect near-surface image pixels in reverse order

Filters in Practice

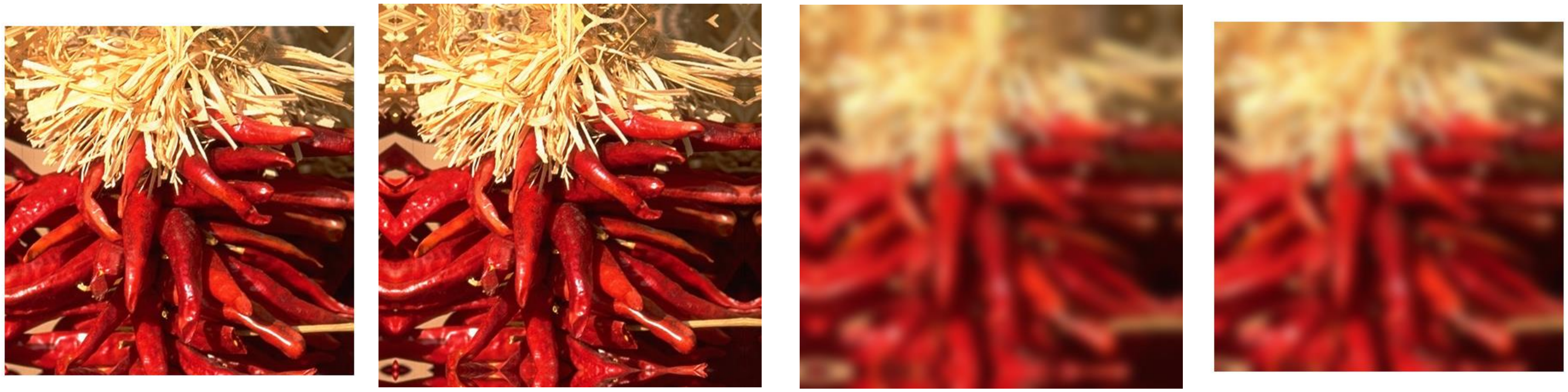


S. Marschner

Filters in Practice



Copy edge



Reflect across edge

S. Marschner