

# **Introduction to Computer Vision**

---

**Kaveh Fathian**

Assistant Professor

Computer Science Department

Colorado School of Mines

**Lecture 3**



# Images in Python (import numpy)

$N \times M$  grayscale image “im”

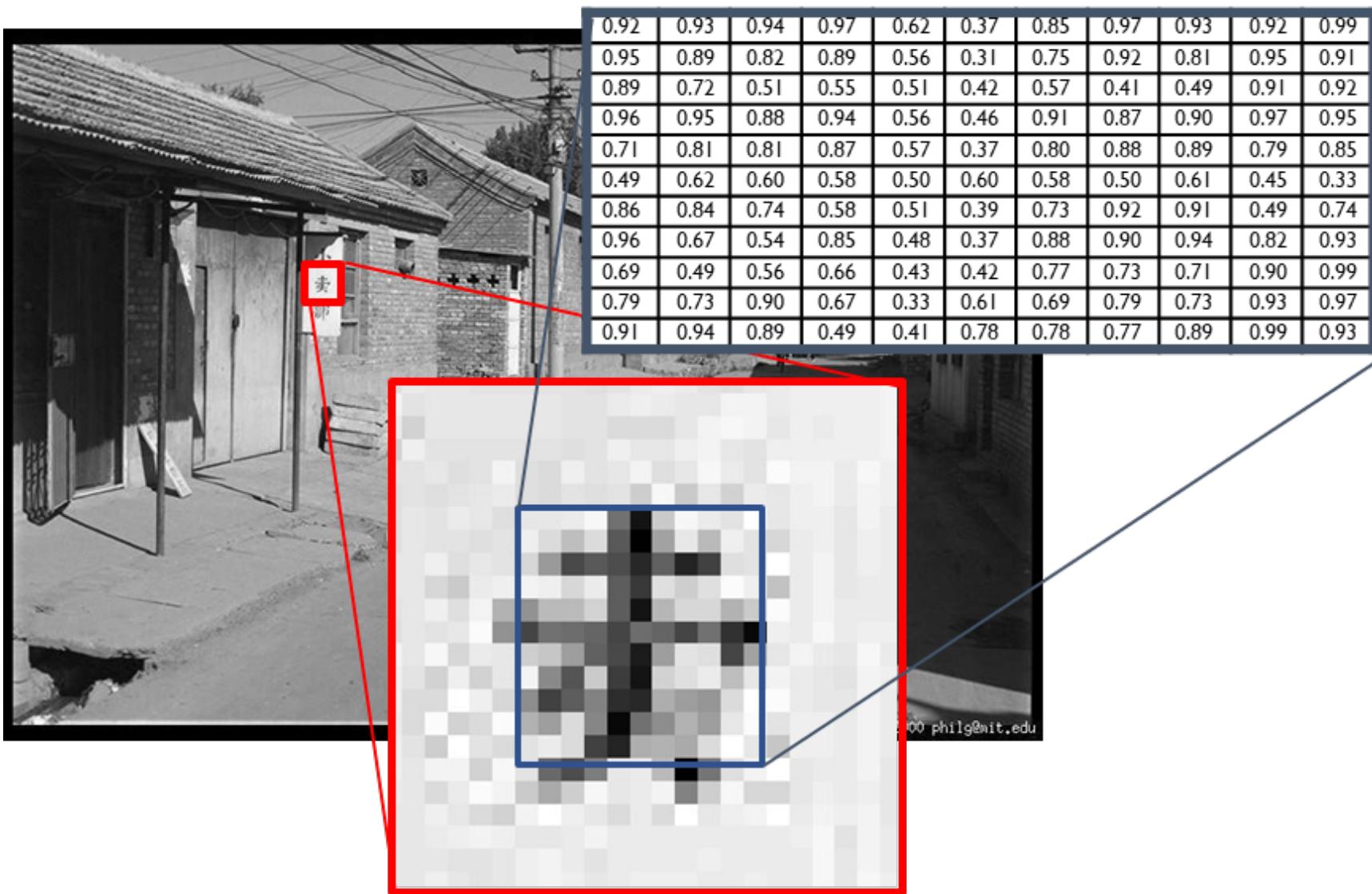
- $im[0, 0]$  = top-left pixel value
- $im[y, x]$  =  $y$  pixels down,  $x$  pixels to right
- $im[N-1, M-1]$  = bottom-right pixel

Row ↓      Column →

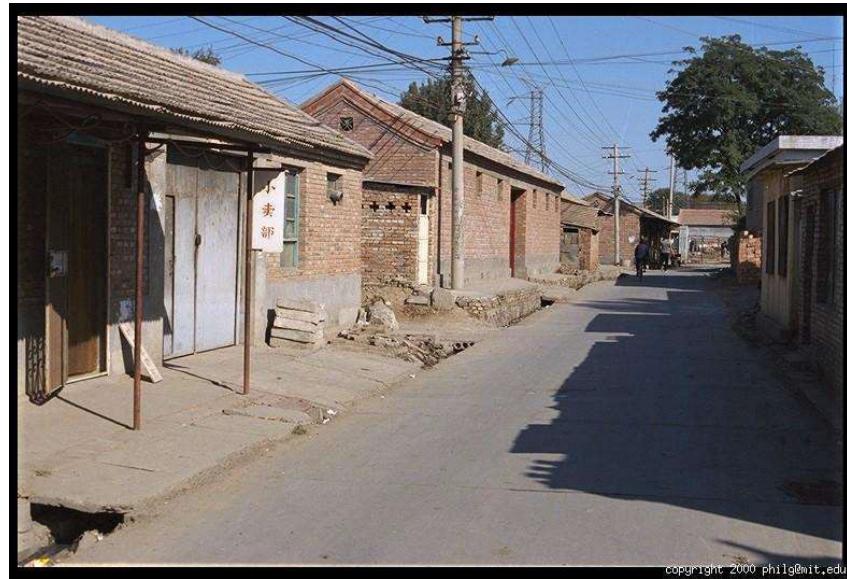
|      |      |      |      |      |      |      |      |      |      |      |
|------|------|------|------|------|------|------|------|------|------|------|
| 0.92 | 0.93 | 0.94 | 0.97 | 0.62 | 0.37 | 0.85 | 0.97 | 0.93 | 0.92 | 0.99 |
| 0.95 | 0.89 | 0.82 | 0.89 | 0.56 | 0.31 | 0.75 | 0.92 | 0.81 | 0.95 | 0.91 |
| 0.89 | 0.72 | 0.51 | 0.55 | 0.51 | 0.42 | 0.57 | 0.41 | 0.49 | 0.91 | 0.92 |
| 0.96 | 0.95 | 0.88 | 0.94 | 0.56 | 0.46 | 0.91 | 0.87 | 0.90 | 0.97 | 0.95 |
| 0.71 | 0.81 | 0.81 | 0.87 | 0.57 | 0.37 | 0.80 | 0.88 | 0.89 | 0.79 | 0.85 |
| 0.49 | 0.62 | 0.60 | 0.58 | 0.50 | 0.60 | 0.58 | 0.50 | 0.61 | 0.45 | 0.33 |
| 0.86 | 0.84 | 0.74 | 0.58 | 0.51 | 0.39 | 0.73 | 0.92 | 0.91 | 0.49 | 0.74 |
| 0.96 | 0.67 | 0.54 | 0.85 | 0.48 | 0.37 | 0.88 | 0.90 | 0.94 | 0.82 | 0.93 |
| 0.69 | 0.49 | 0.56 | 0.66 | 0.43 | 0.42 | 0.77 | 0.73 | 0.71 | 0.90 | 0.99 |
| 0.79 | 0.73 | 0.90 | 0.67 | 0.33 | 0.61 | 0.69 | 0.79 | 0.73 | 0.93 | 0.97 |
| 0.91 | 0.94 | 0.89 | 0.49 | 0.41 | 0.78 | 0.78 | 0.77 | 0.89 | 0.99 | 0.93 |

James Hays

# Grayscale Intensity



# Color



Red intensity



Green

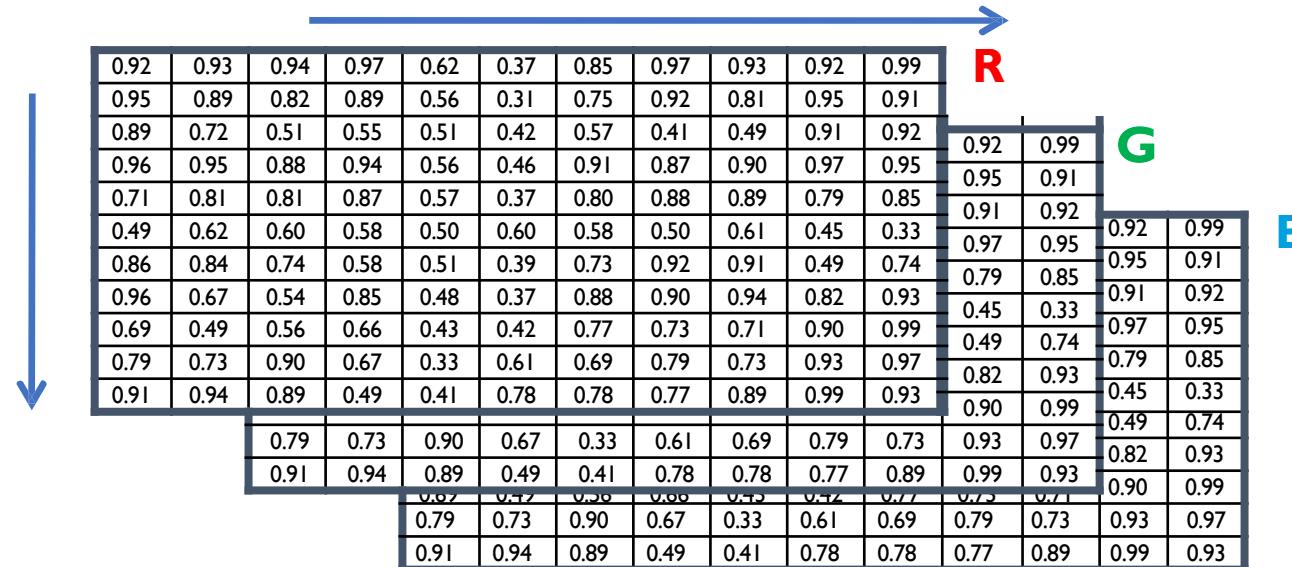


Blue

# Images in Python (import numpy)

$N \times M$  grayscale image “im”

- $im[0, 0, 0]$  = top-left pixel value, **red channel**
- $im[y, x, 1]$  =  $y$  pixels down,  $x$  pixels to right, **green channel**
- $im[N-1, M-1, 2]$  = bottom-right pixel, **blue channel**



James Hays

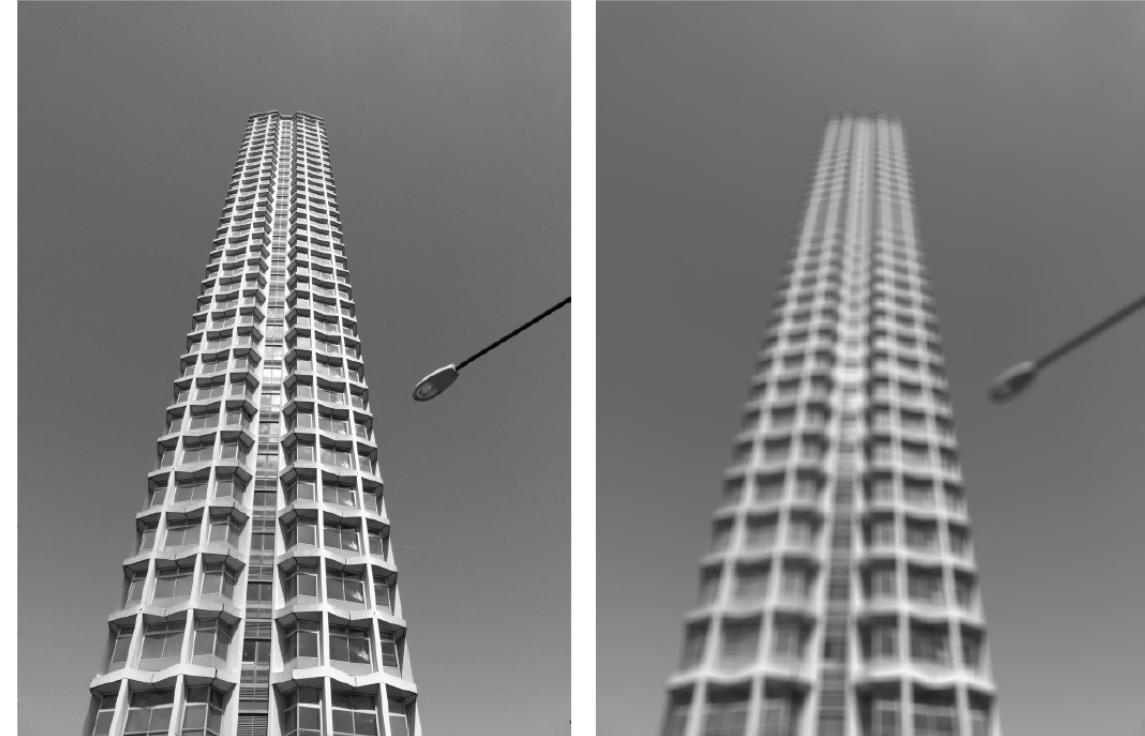
# Image Filtering



# Image Filtering

## Learning outcomes:

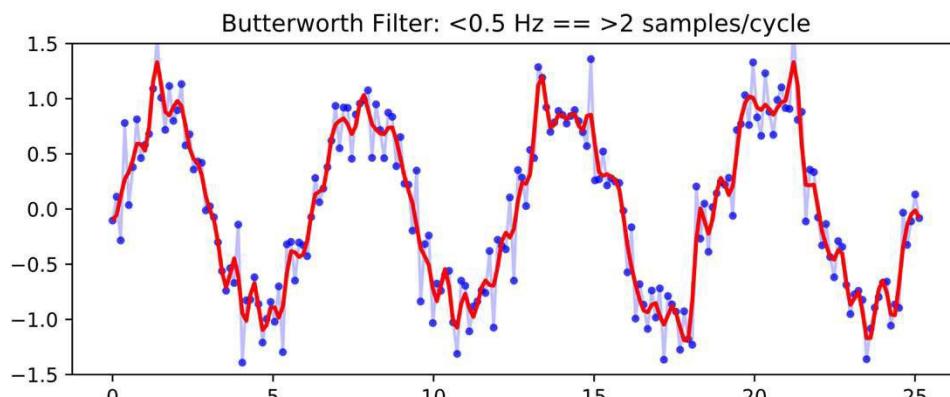
- Filter
- Image Filtering
- Low-pass Filter
- High-pass Filter
- Convolution
- Correlation
- Kernel
- Linearity and Shift Invariance
- Separability
- Sobel Filter
- Convolution
- Correlation & Template Matching
- Non-Linear Filters



# Filtering

## Definition

**Filtering:** An operation that modifies a (measured) signal.



<https://ggbaker.ca/data-science/media/butter4.svg>

- Modifications include
  - Removing undesirable components
  - Transforming signal in a desirable way
  - Extract specific components of a signal
- Can be analog or discrete signals
- Can be in any dimension

# 1D Filtering: Moving Average

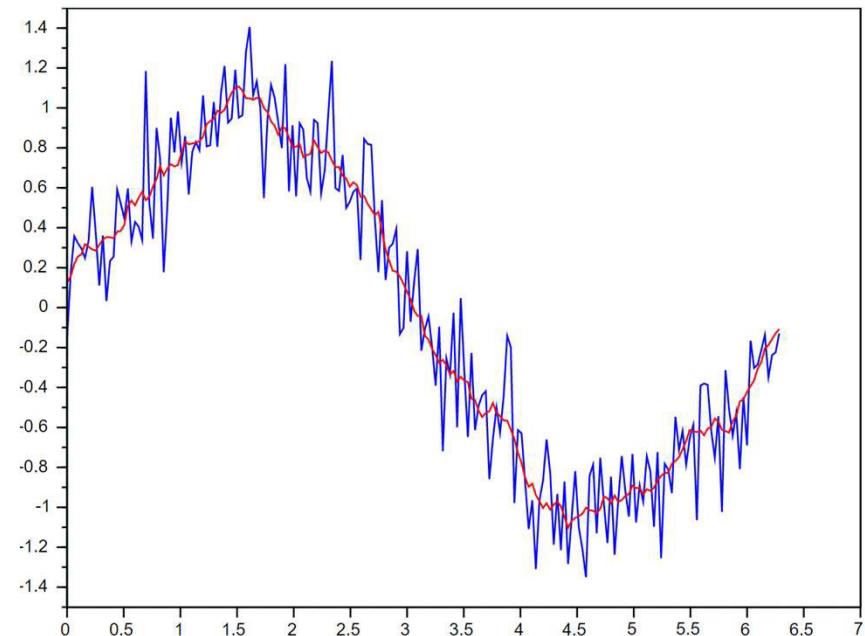
$$I \in \mathcal{R}^{m \times 1}$$

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

Window size 'k'

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

[https://en.wikipedia.org/wiki/Moving\\_average](https://en.wikipedia.org/wiki/Moving_average)



# 1D Filtering: Moving Average

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

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

Window size 'k'

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

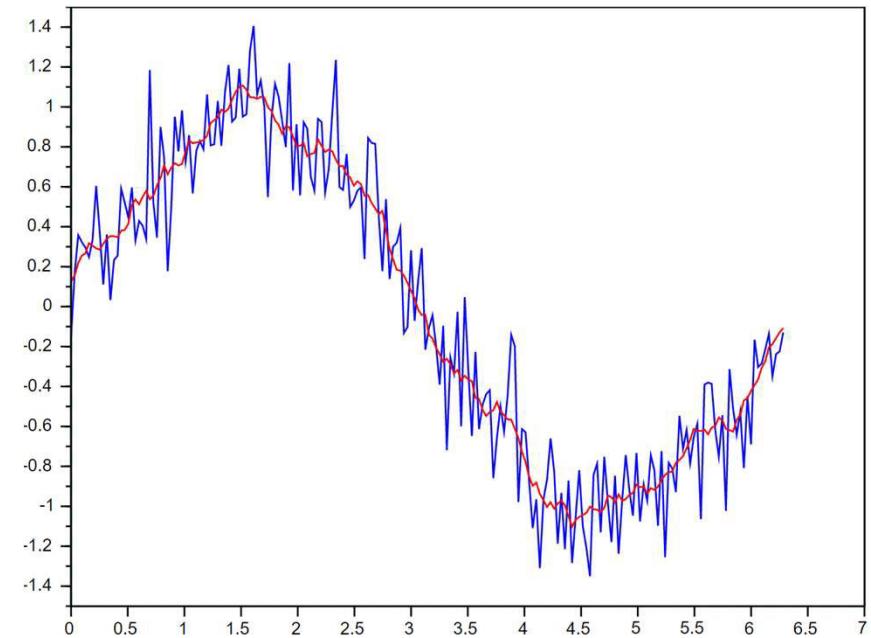
[https://en.wikipedia.org/wiki/Moving\\_average](https://en.wikipedia.org/wiki/Moving_average)

$$\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

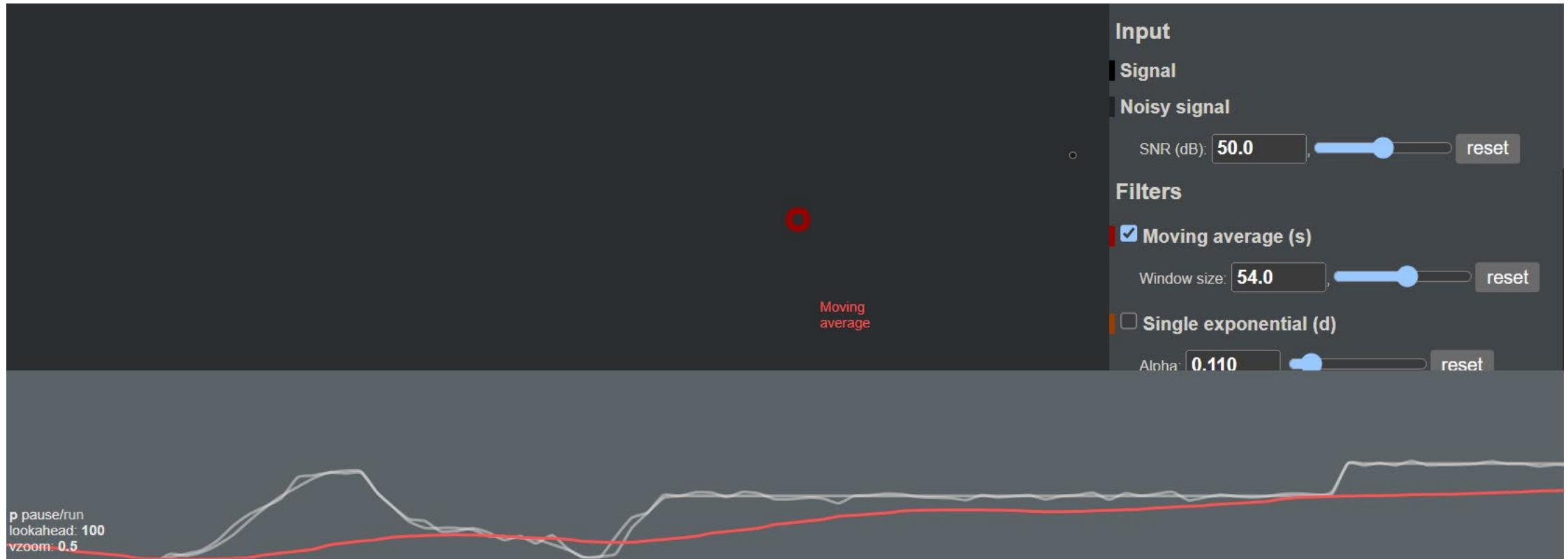
What is this filter?

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

What is the difference?



# Filtering Demo



<https://gery.casiez.net/1euro/InteractiveDemo/>

# Image Filtering

Compute function of local neighborhood at each position:

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

Window size 'k' and 'l'

# Image Filtering

Compute function of local neighborhood at each position:

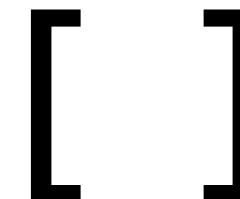
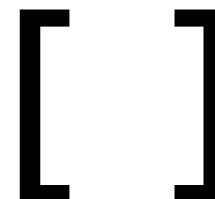
h=output

I=image

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

2d coords=k, l

2d coords=m, n



# Fundamental Equations of Computer Vision

1. Image Filtering

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

2. Optical Flow

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$$

3. Camera Geometry

$$\mathbf{x} = \mathbf{K}[\mathbf{R} \ \mathbf{t}] \mathbf{X} \quad x^T F x' = 0$$

4. Machine Learning

$$\arg \min_{\mathbf{S}} \sum_{i=1}^k \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2. \quad y = \varphi(\sum_{i=1}^n w_i x_i + b) = \varphi(\mathbf{w}^T \mathbf{x} + b)$$

# Example: Box Filter / Kernel

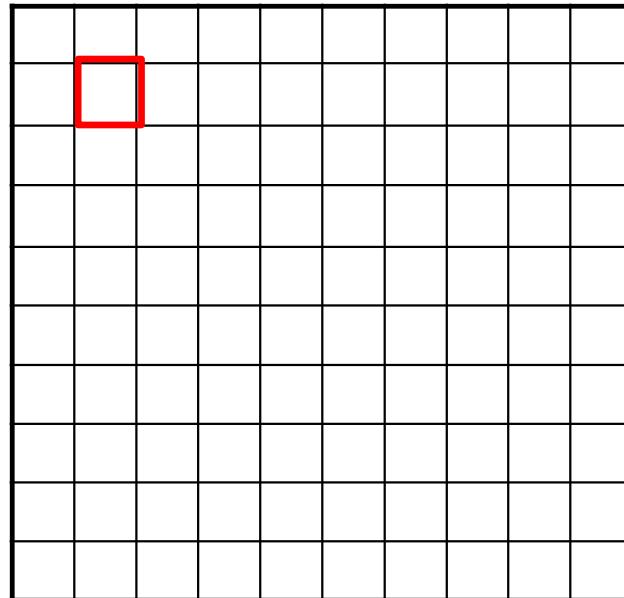
$$f[\cdot, \cdot]$$
$$\frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

# Image Filtering

$$f[\cdot, \cdot] \frac{1}{9}$$

$$I[ \cdot , \cdot ]$$

$h[.,.]$

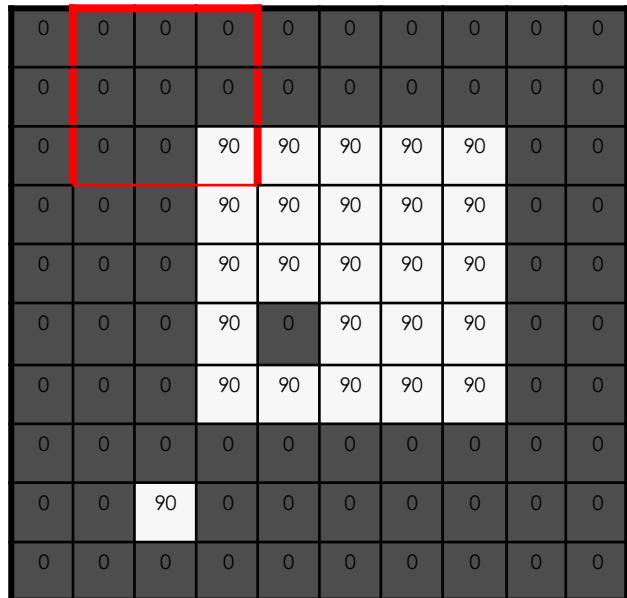


$$h[m, n] = \sum_{k, l} f[k, l] I[m + k, n + l] \quad \begin{matrix} m = 1, n = 1 \\ k, l = [-1, 0, 1] \end{matrix}$$

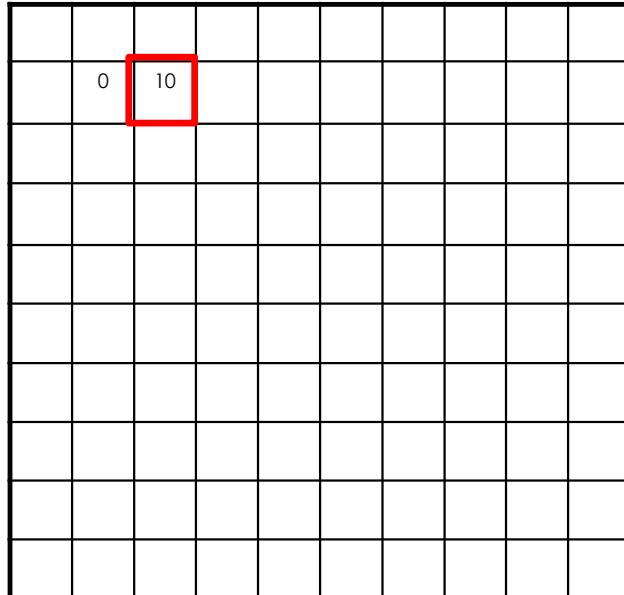
# Image Filtering

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

$I[.,.]$



$h[.,.]$

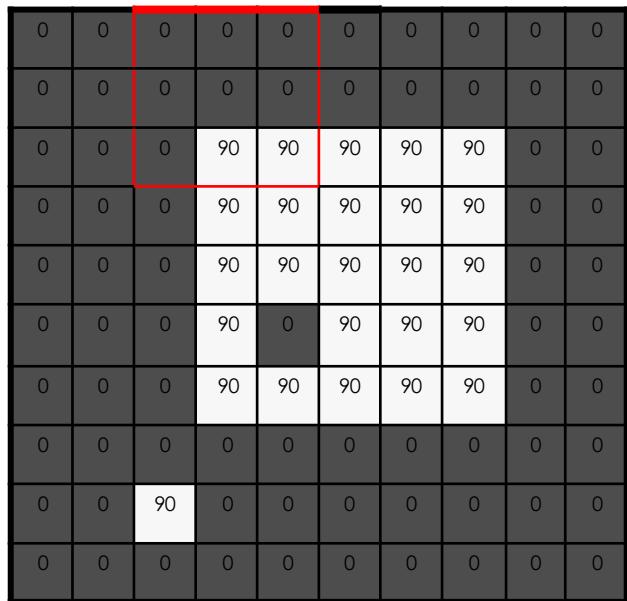


$$h[m, n] = \sum_{k, l} f[k, l] I[m + k, n + l]$$
$$\begin{aligned} m &= 2, n = 1 \\ k, l &= [-1, 0, 1] \end{aligned}$$

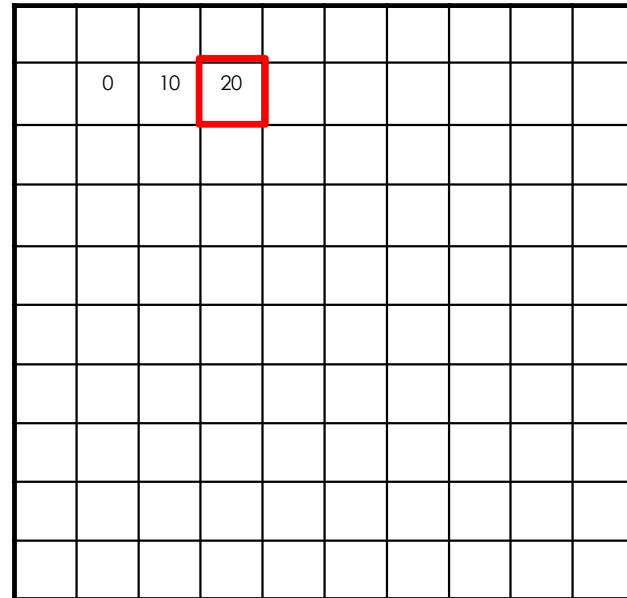
# Image Filtering

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

$I[.,.]$



$h[.,.]$

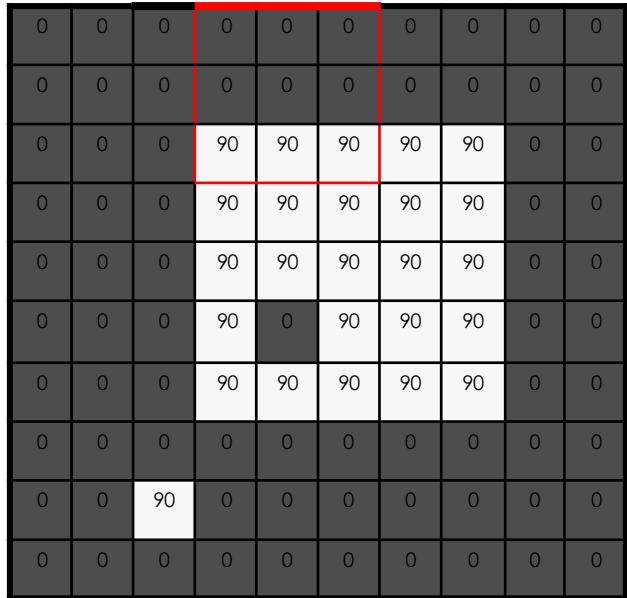


$$h[m, n] = \sum_{k, l} f[k, l] I[m + k, n + l] \quad m = 3, n = 1 \\ k, l = [-1, 0, 1]$$

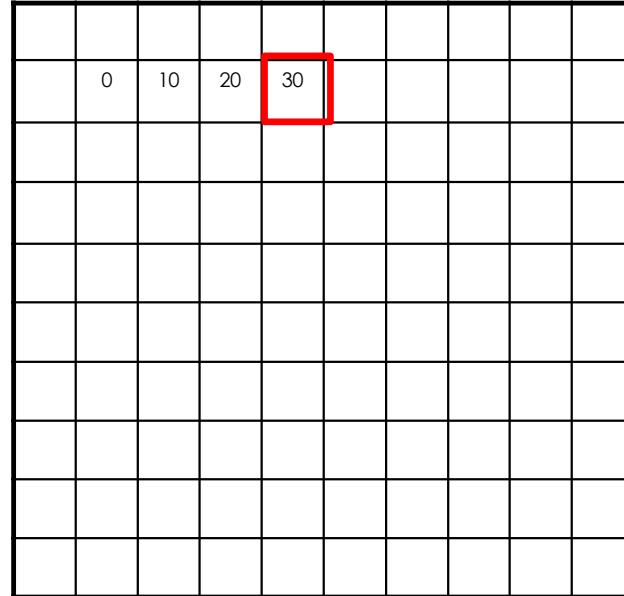
# Image Filtering

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

$I[.,.]$



$h[.,.]$



$$h[m, n] = \sum_{k, l} f[k, l] I[m + k, n + l] \quad m = 4, n = 1 \\ k, l = [-1, 0, 1]$$

# Image Filtering

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

$I[.,.]$

|   |   |    |    |    |    |    |    |    |   |   |
|---|---|----|----|----|----|----|----|----|---|---|
| 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  | 90 | 90 | 90 | 90 | 90 | 0  | 0 | 0 |
| 0 | 0 | 0  | 90 | 90 | 90 | 90 | 90 | 0  | 0 | 0 |
| 0 | 0 | 0  | 90 | 90 | 90 | 90 | 90 | 0  | 0 | 0 |
| 0 | 0 | 0  | 90 | 90 | 90 | 90 | 90 | 0  | 0 | 0 |
| 0 | 0 | 0  | 90 | 90 | 0  | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0  | 90 | 90 | 90 | 90 | 90 | 0  | 0 | 0 |
| 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0 | 0 |
| 0 | 0 | 90 | 0  | 0  | 0  | 0  | 0  | 0  | 0 | 0 |
| 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0 | 0 |

$h[.,.]$

|  |  |  |  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|--|--|--|
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

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

$$\begin{aligned} m &= 5, n = 1 \\ k, l &= [-1, 0, 1] \end{aligned}$$

# Image Filtering

$$f[\cdot, \cdot] \frac{1}{9}$$

$$I[ \cdot, \cdot ]$$

$$h[.,.]$$

$$h[m, n] = \sum_{k, l} f[k, l] I[m + k, n + l] \quad \begin{matrix} m = 4, n = 6 \\ k, l = [-1, 0, 1] \end{matrix}$$

# Image Filtering

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

$I[.,.]$

|   |   |    |    |    |    |    |    |   |   |   |
|---|---|----|----|----|----|----|----|---|---|---|
| 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  | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0  | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0  | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0  | 90 | 0  | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0  | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  | 0 | 0 | 0 |
| 0 | 0 | 90 | 0  | 0  | 0  | 0  | 0  | 0 | 0 | 0 |
| 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  | 0 | 0 | 0 |

$h[.,.]$

|  |   |    |    |    |    |  |  |  |  |  |
|--|---|----|----|----|----|--|--|--|--|--|
|  |   |    |    |    |    |  |  |  |  |  |
|  | 0 | 10 | 20 | 30 | 30 |  |  |  |  |  |
|  |   |    |    |    |    |  |  |  |  |  |
|  |   |    |    |    |    |  |  |  |  |  |
|  |   |    |    |    |    |  |  |  |  |  |
|  |   |    |    |    |    |  |  |  |  |  |
|  |   |    |    |    |    |  |  |  |  |  |
|  |   |    |    |    |    |  |  |  |  |  |
|  |   |    |    |    |    |  |  |  |  |  |
|  |   |    |    |    |    |  |  |  |  |  |
|  |   |    |    |    |    |  |  |  |  |  |

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

$$\begin{aligned} m &= 6, n = 4 \\ k, l &= [-1, 0, 1] \end{aligned}$$

# Image Filtering

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

|  |    |    |    |    |    |    |    |    |  |
|--|----|----|----|----|----|----|----|----|--|
|  |    |    |    |    |    |    |    |    |  |
|  | 0  | 10 | 20 | 30 | 30 | 30 | 20 | 10 |  |
|  | 0  | 20 | 40 | 60 | 60 | 60 | 40 | 20 |  |
|  | 0  | 30 | 60 | 90 | 90 | 90 | 60 | 30 |  |
|  | 0  | 30 | 50 | 80 | 80 | 90 | 60 | 30 |  |
|  | 0  | 30 | 50 | 80 | 80 | 90 | 60 | 30 |  |
|  | 0  | 20 | 30 | 50 | 50 | 60 | 40 | 20 |  |
|  | 10 | 20 | 30 | 30 | 30 | 30 | 20 | 10 |  |
|  | 10 | 10 | 10 | 0  | 0  | 0  | 0  | 0  |  |
|  |    |    |    |    |    |    |    |    |  |

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

# Box Filter / Kernel

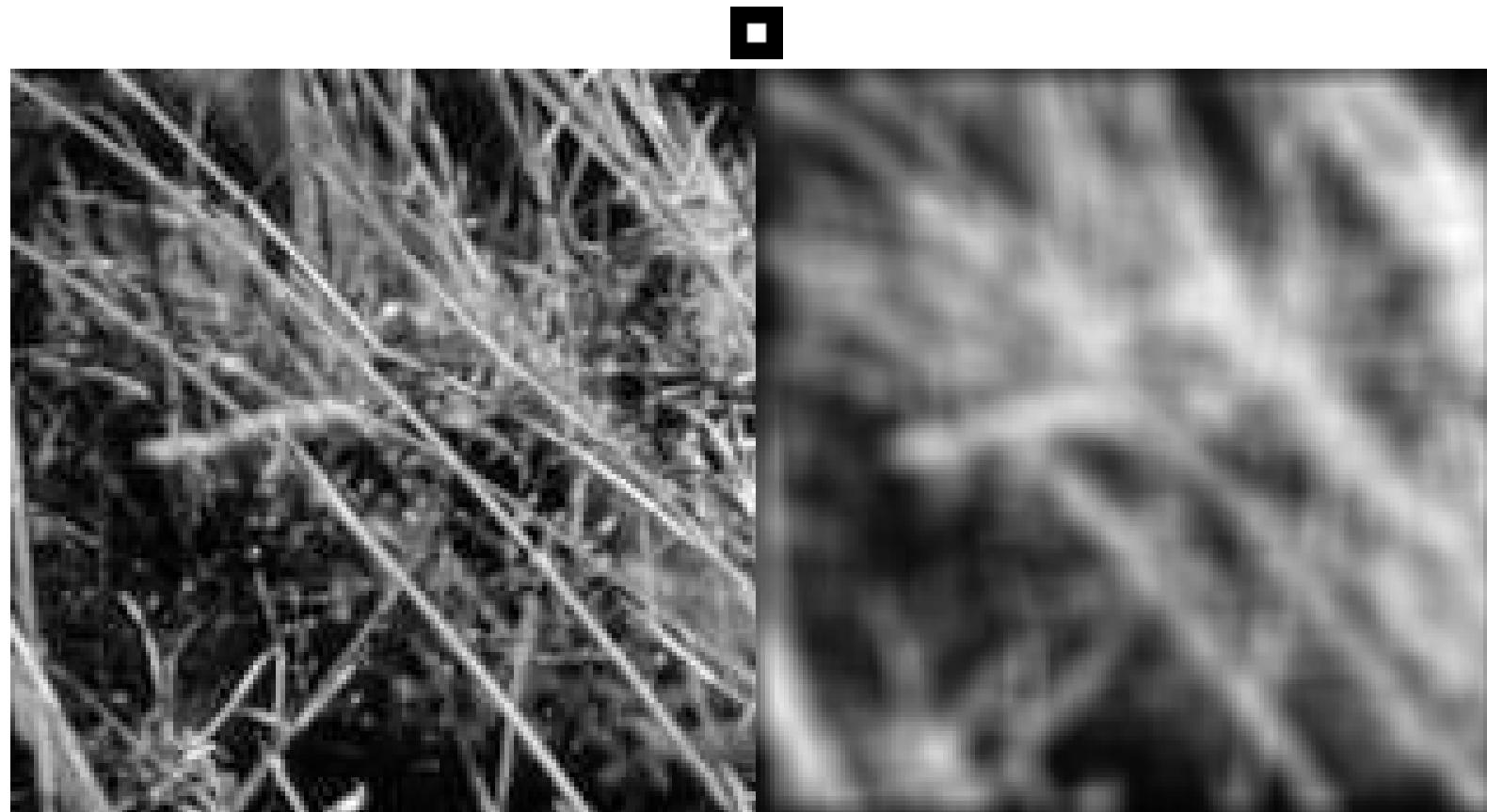
## What does it do?

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

$$\frac{1}{9} f[\cdot, \cdot]$$

|   |   |   |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

# Smoothing with Box Filter

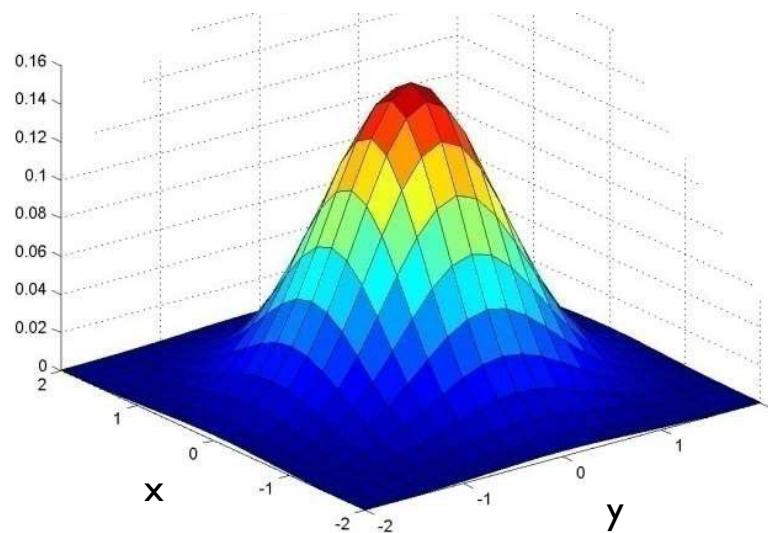


$$f[\cdot, \cdot] \frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

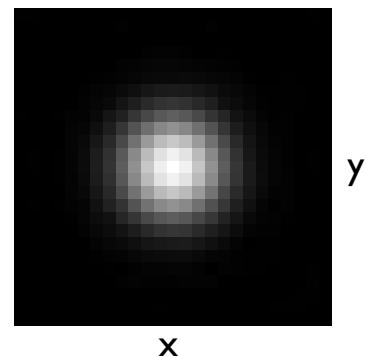
David Lowe

Kaveh Fathian

# Gaussian Filter / Kernel



Viewed  
from top

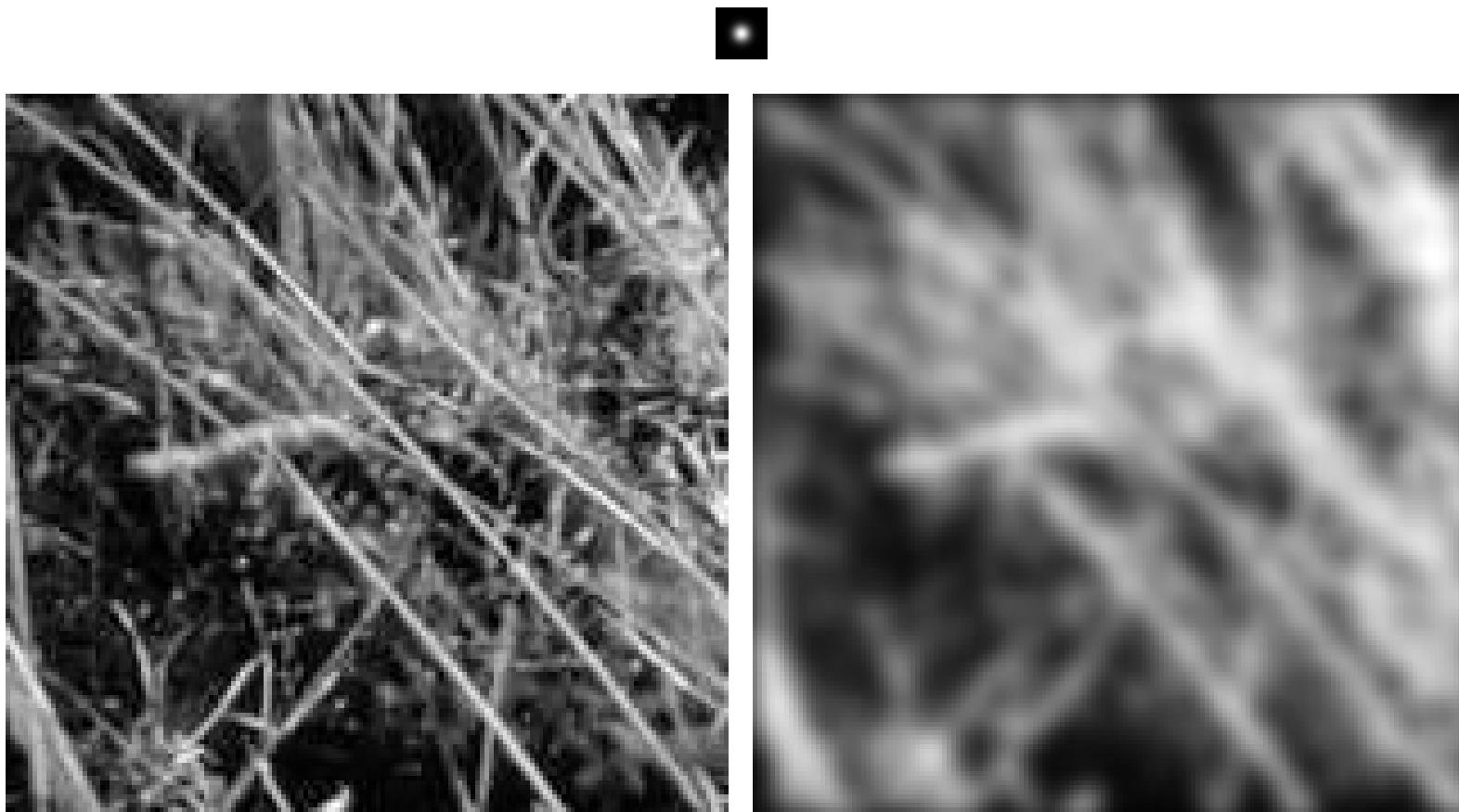


$$G_{\sigma} = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

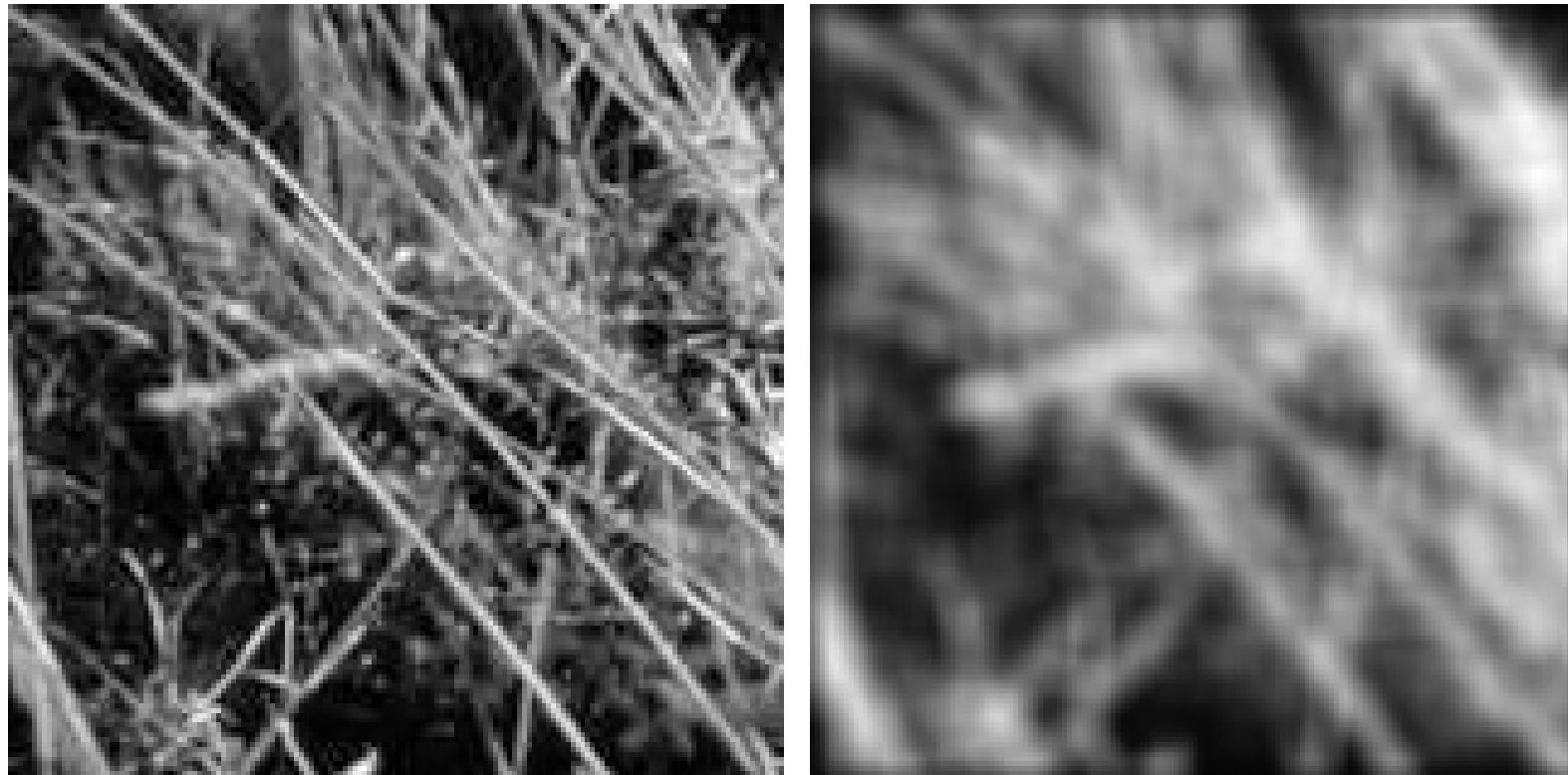
| x | 0.003 | 0.013 | 0.022 | 0.013 | 0.003 |
|---|-------|-------|-------|-------|-------|
| y | 0.013 | 0.059 | 0.097 | 0.059 | 0.013 |
| x | 0.022 | 0.097 | 0.159 | 0.097 | 0.022 |
| y | 0.013 | 0.059 | 0.097 | 0.059 | 0.013 |
| x | 0.003 | 0.013 | 0.022 | 0.013 | 0.003 |

Filter/Kernel size  $5 \times 5$ ,  
Standard deviation  $\sigma = 1$

# Smoothing with Gaussian Filter



# Smoothing with Box Filter



# Linear Filter Properties

## Linearity:

$$\text{imfilter}(I, f_1 + f_2) =$$

$$\text{imfilter}(I, f_1) + \text{imfilter}(I, f_2)$$

## Shift/translation invariance:

Same behavior given intensities regardless of pixel location m,n

$$\text{imfilter}(I, \text{shift}(f)) = \text{shift}(\text{imfilter}(I, f))$$

# Gaussian Filter Properties

**Gaussian convolved with Gaussian...**  
...is another **Gaussian**

- So can smooth with small-width kernel, repeat, and get same result as larger-width kernel
- Convoluting twice with Gaussian kernel of width  $\sigma$  is same as convoluting once with kernel of width  $\sigma\sqrt{2}$

**Separable kernel**

- Factors into product of two 1D Gaussians

# Image Filtering Applications

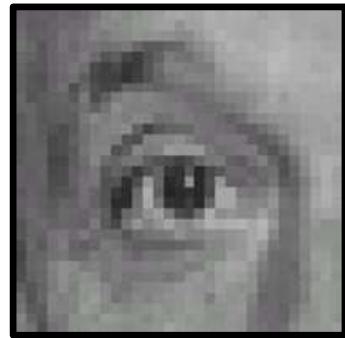
Compute function of local neighborhood at each position:

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

## Really important!

- **Enhance images**
  - Denoise, resize, increase contrast, etc.
- **Extract information** from images
  - Texture, edges, distinctive points, etc.
- **Detect patterns**
  - Template matching

# Practice with linear filters



1.

|   |   |   |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

2.

|   |   |   |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 0 | 0 |

3.

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

4.

|   |   |   |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 2 | 0 |
| 0 | 0 | 0 |

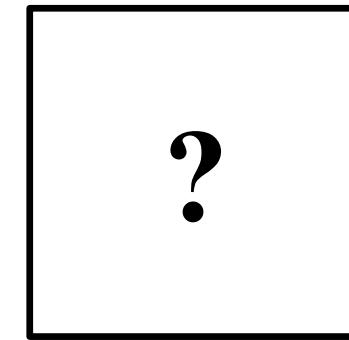
$$- \frac{1}{9} \begin{array}{|ccc|} \hline 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ \hline \end{array}$$

# Practice with linear filters

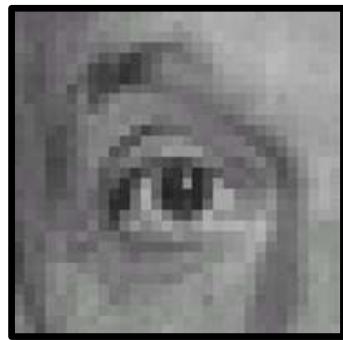


Original

|   |   |   |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

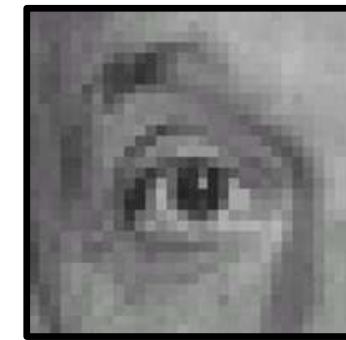


# Practice with linear filters



Original

|   |   |   |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |



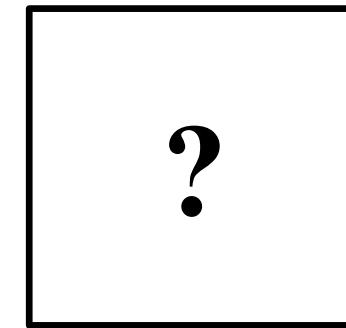
Filtered  
(no change)

# Practice with linear filters

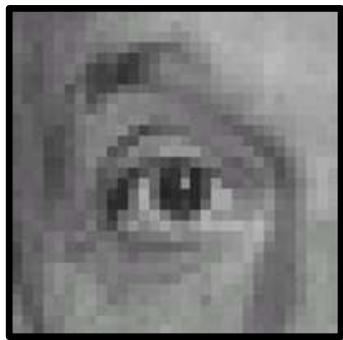


Original

|   |   |   |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 0 | 0 |

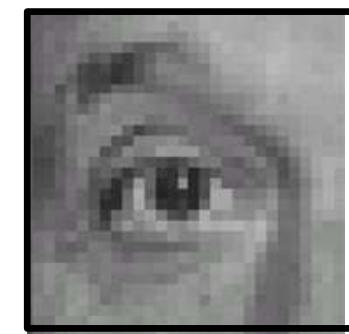


# Practice with linear filters



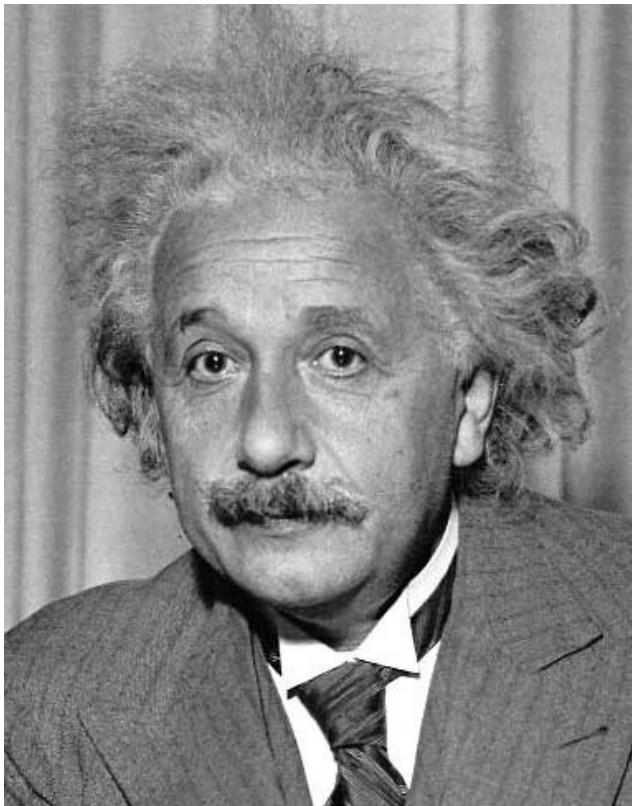
Original

|   |   |   |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 0 | 0 |



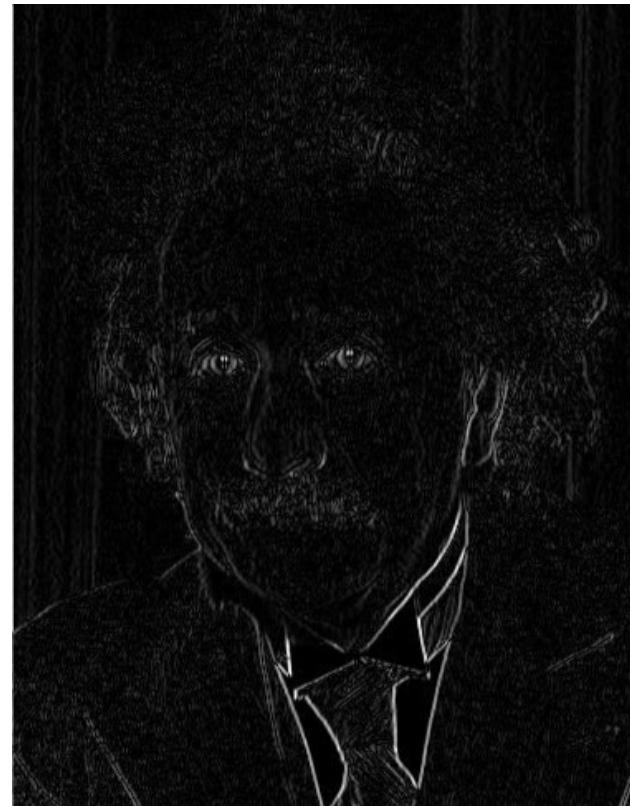
Shifted left  
By 1 pixel

# Practice with linear filters



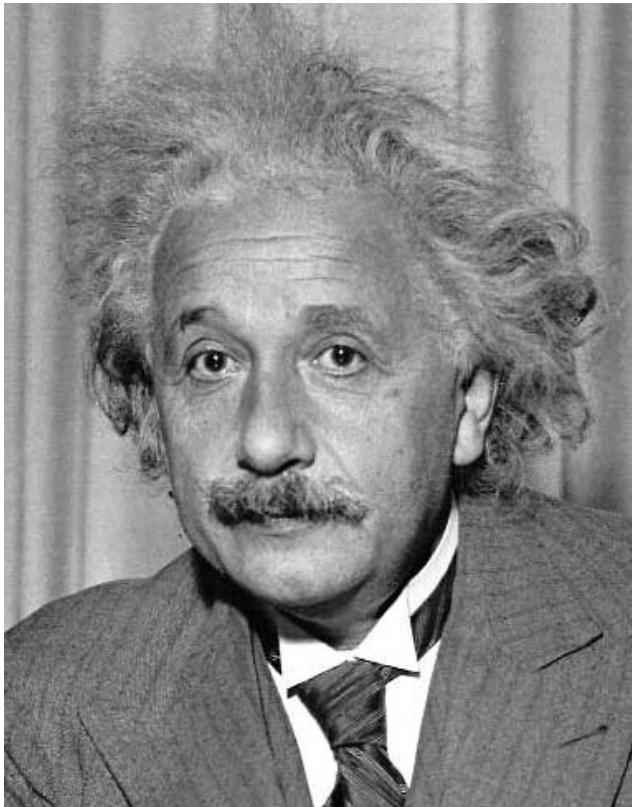
|   |   |    |
|---|---|----|
| 1 | 0 | -1 |
| 2 | 0 | -2 |
| 1 | 0 | -1 |

Sobel



Vertical Edge  
(absolute value)

# Practice with linear filters



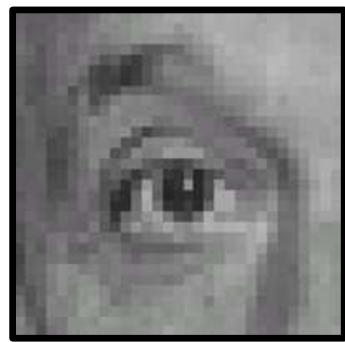
|    |    |    |
|----|----|----|
| 1  | 2  | 1  |
| 0  | 0  | 0  |
| -1 | -2 | -1 |

Sobel



Horizontal Edge  
(absolute value)

# Practice with linear filters



Original

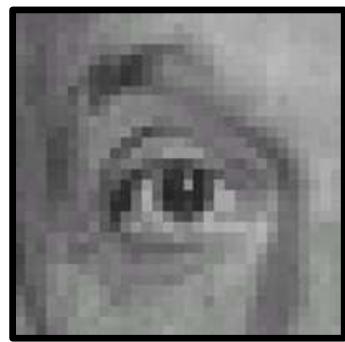
$$\begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 2 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array}$$

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

?

(Note that filter sums to 1)

# Practice with linear filters



Original

|   |   |   |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 2 | 0 |
| 0 | 0 | 0 |

-

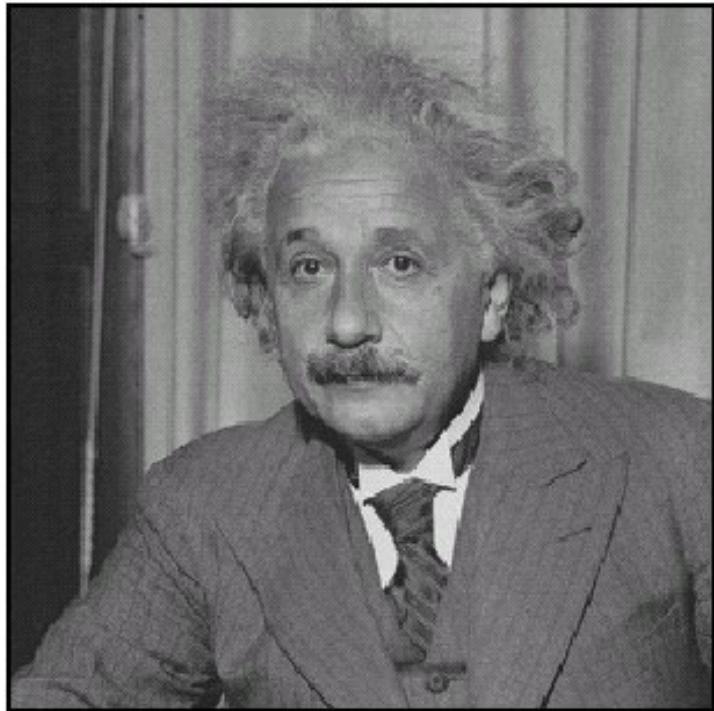
|               |   |   |   |
|---------------|---|---|---|
| $\frac{1}{9}$ | 1 | 1 | 1 |
| 1             | 1 | 1 | 1 |
| 1             | 1 | 1 | 1 |



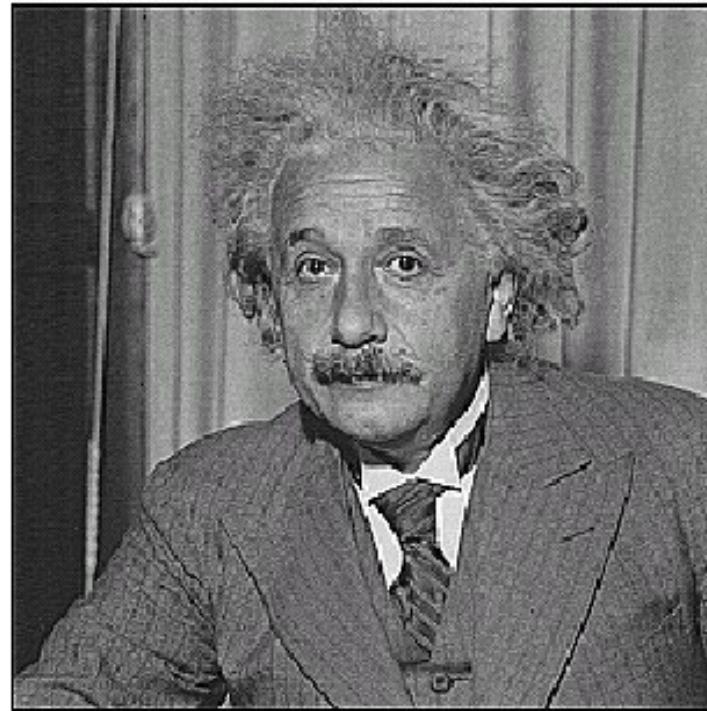
## Sharpening filter

- Accentuates differences with local average

# Practice with linear filters



**before**

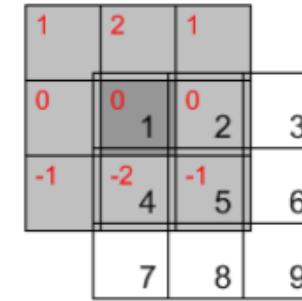
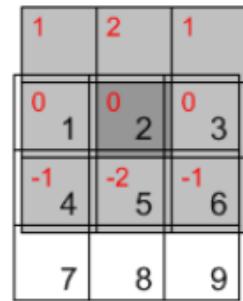
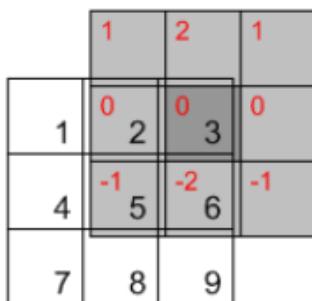


**after**

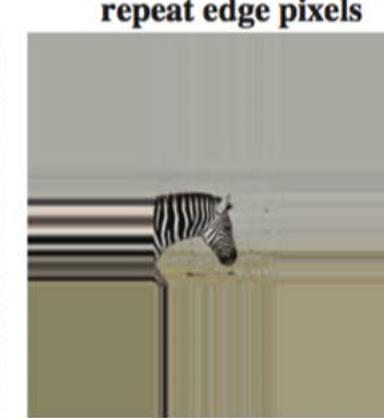
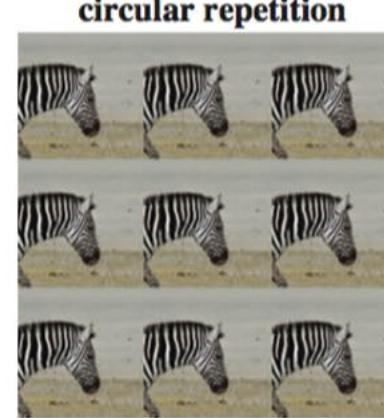
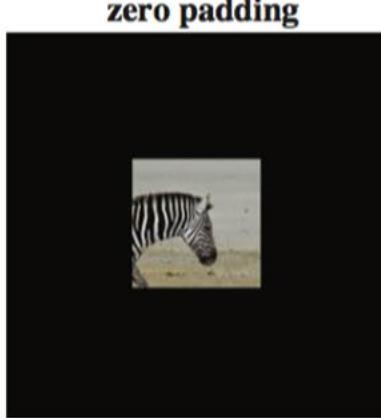
# Filters in Practice

## What about near the edge?

- The filter window falls off the edge of the image
- Need to extrapolate (padding)
- Methods:
  - clip filter (black / zero padding)
  - wrap around
  - copy edge
  - reflect across edge

|   |   |   |
|---|---|---|
|  |  |  |
|---|---|---|

Input



# Correlation and Convolution

## Definition

- **2D correlation**

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

e.g., `h = scipy.signal.correlate2d(f, I)`

\* Symmetric in the geometric sense, not in the matrix linear algebra sense.

# Filtering

## Definition

- 2D correlation

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

e.g., `h = scipy.signal.correlate2d(f, I)`

- 2D convolution

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

e.g., `h = scipy.signal.convolve2d(f, I)`

Convolution is the same as correlation with a  $180^\circ$  rotated filter kernel.  
Correlation and convolution are identical when the filter kernel is symmetric\*.

\* Symmetric in the geometric sense, not in the matrix linear algebra sense.

# Convolution Properties

**Commutative:**  $a * b = b * a$

- Conceptually no difference between filter and signal
- But filtering implementations might break this equality, e.g., image edges

**Associative:**  $a * (b * c) = (a * b) * c$

- Often apply several filters one after another:  $((a * b_1) * b_2) * b_3$
- This is equivalent to applying one filter:  $a * (b_1 * b_2 * b_3)$
- Why important?
- Correlation is **NOT** associative (rotation effect)

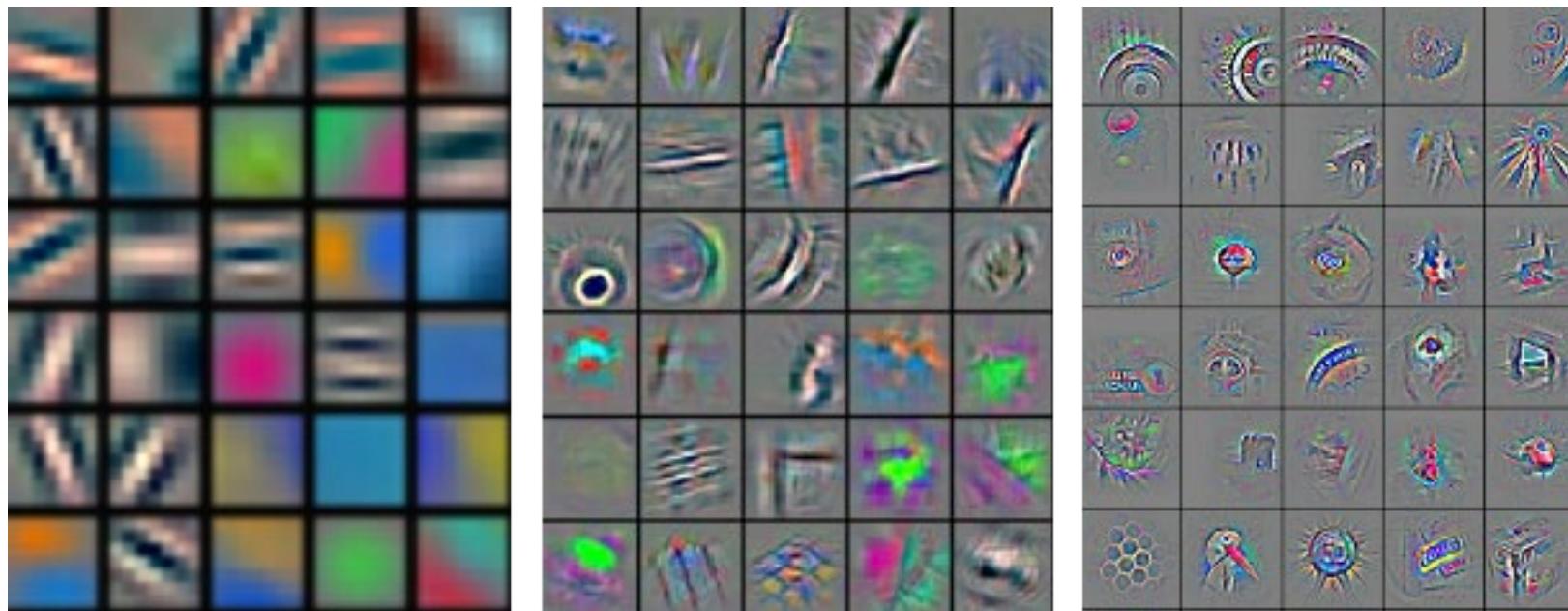
**Distributes over addition:**  $a * (b + c) = (a * b) + (a * c)$

**Scalars factor out:**  $ka * b = a * kb = k(a * b)$

**Identity:** unit impulse     $e = [0, 0, 1, 0, 0]$ ,  $a * e = a$

# Convolution in Convolutional Neural Networks

- Convolution is the basic operation in CNNs
- Learning convolution kernels allows us to learn which ‘features’ provide useful information in images.



# Linear Filters

## Linearity:

$$\text{imfilter}(I, f_1 + f_2) = \text{imfilter}(I, f_1) + \text{imfilter}(I, f_2)$$

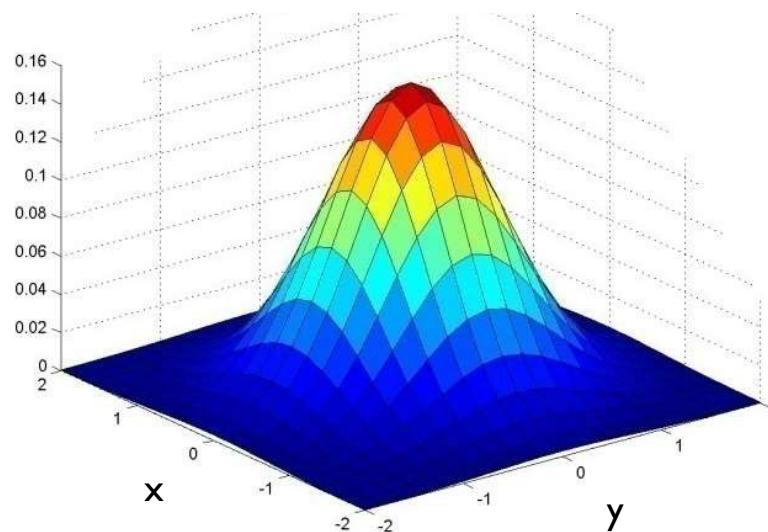
## Shift/translation invariance:

Same behavior given intensities regardless of pixel location m,n

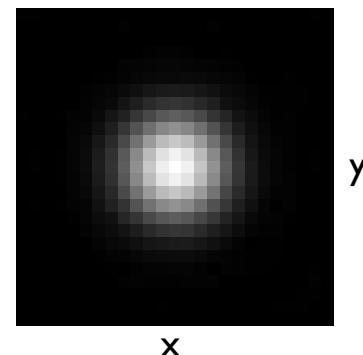
$$\text{imfilter}(I, \text{shift}(f)) = \text{shift}(\text{imfilter}(I, f))$$

**Any linear, shift-invariant operator can be represented as a convolution**

# Gaussian Filter / Kernel



Viewed  
from top



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

| x | 0.003 | 0.013 | 0.022 | 0.013 | 0.003 |
|---|-------|-------|-------|-------|-------|
| y | 0.013 | 0.059 | 0.097 | 0.059 | 0.013 |
| x | 0.022 | 0.097 | 0.159 | 0.097 | 0.022 |
| y | 0.013 | 0.059 | 0.097 | 0.059 | 0.013 |
| x | 0.003 | 0.013 | 0.022 | 0.013 | 0.003 |

Filter/Kernel size  $5 \times 5$ ,  
Standard deviation  $\sigma = 1$

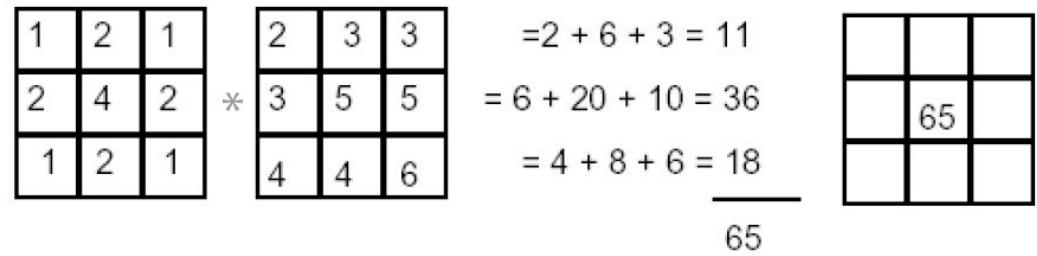
# Separability of the Gaussian Filter

- The 2D Gaussian can be expressed as the product of two functions, one a function of  $x$  and the other a function of  $y$
- In this case, the two functions are the (identical) 1D Gaussian:

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

# Separability Example

2D convolution (center location only):

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 2 & 3 & 3 \\ \hline 3 & 5 & 5 \\ \hline 4 & 4 & 6 \\ \hline \end{array} = \begin{array}{r} = 2 + 6 + 3 = 11 \\ = 6 + 20 + 10 = 36 \\ = 4 + 8 + 6 = 18 \\ \hline 65 \end{array}$$


The filter factors into a product of 1D filters:

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} = \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline \end{array}$$

Perform convolution along rows:

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 2 & 3 & 3 \\ \hline 3 & 5 & 5 \\ \hline 4 & 4 & 6 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 11 & & \\ \hline 18 & & \\ \hline 18 & & \\ \hline \end{array}$$

Followed by convolution along the remaining column:

$$\begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 11 & & \\ \hline 18 & & \\ \hline 18 & & \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & 65 & \\ \hline \end{array}$$

# Separability Use

MxN image, PxQ filter

- 2D convolution:  $\sim MNPQ$  multiply-adds
- Separable 2D:  $\sim MN(P+Q)$  multiply-adds

Speed up =  $PQ/(P+Q)$  9x9 filter =  $\sim 4.5x$  faster

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} = \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline \end{array}$$

# Sobel Filter

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

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

Sobel



# Sobel Filter

- What happens to negative numbers?

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

Sobel



# Sobel Filter

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



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



# Sobel Filter

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

```
>> I = img_to_float32( io.imread('luke.jpg') )  
>> h = convolve2d( I, sobelKernel )  
  
>> plt.imshow( h )  
  
>> plt.imshow( h + 0.5 )
```

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

Sobel



# Practice

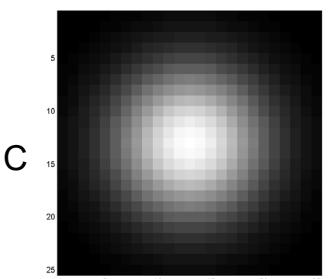
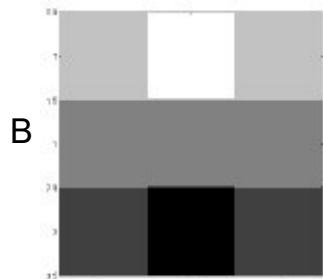
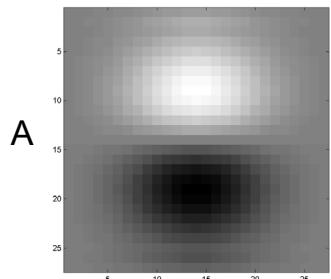
\* = Convolution operator

$$1) \underline{\quad} = D * B$$

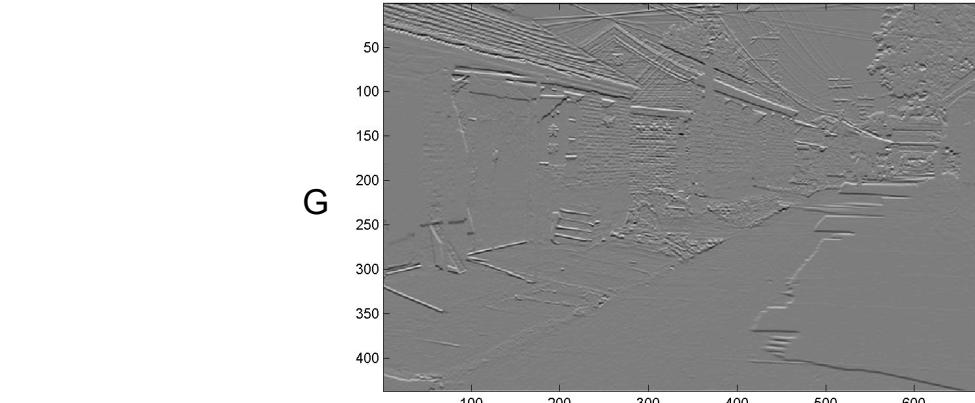
$$2) A = \underline{\quad} *$$

$$3) F = D * \underline{\quad}$$

$$4) \underline{\quad} = D * D$$



Derek Hoiem



Kaveh Fathian

# Practice

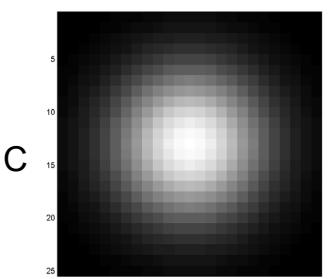
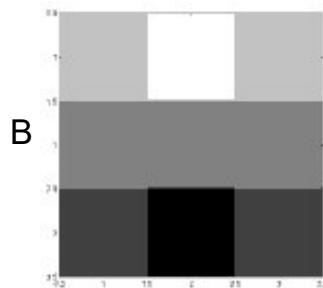
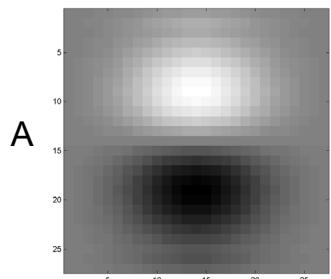
\* = Convolution operator

$$1) G = D * B$$

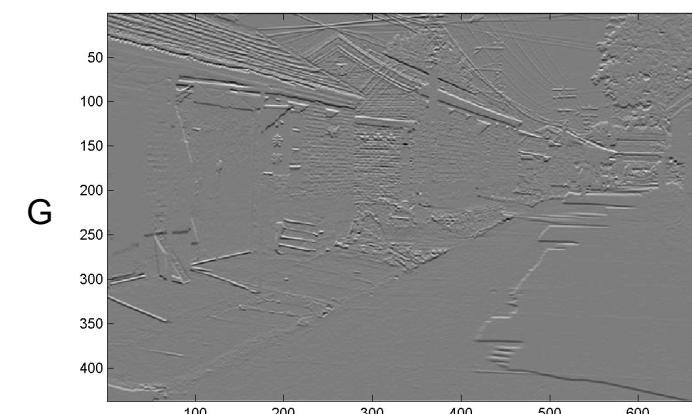
$$2) A = B * C$$

$$3) F = D * E$$

$$4) I = D * D$$



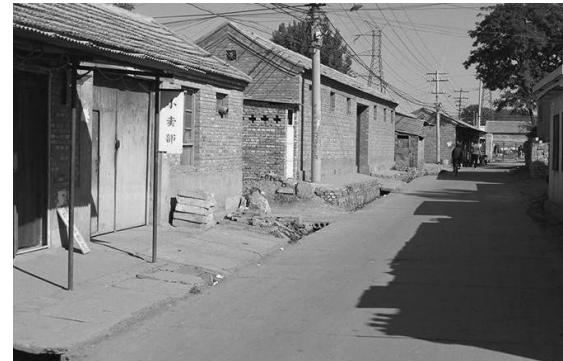
Derek Hoiem



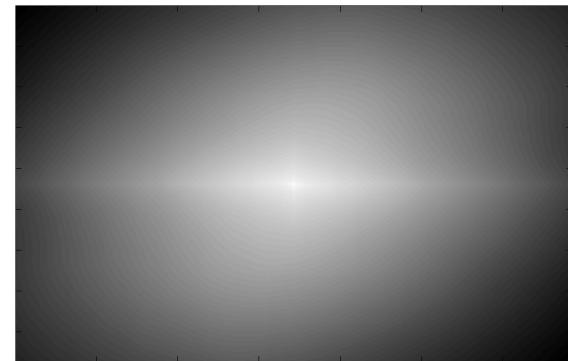
Kaveh Fathian

# Convolution

D (275 x 175 pixels)



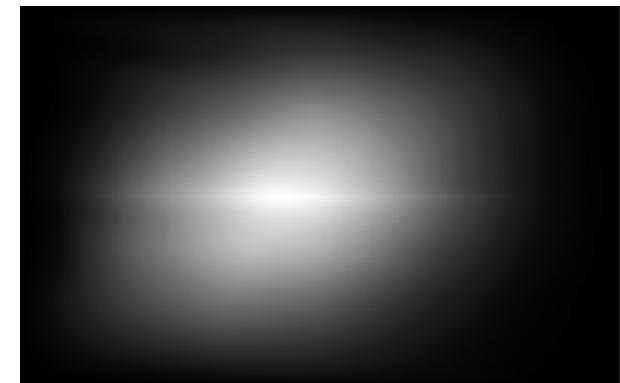
I (275 x 175 pixels)



```
>> D = img_to_float32( io.imread( 'convexample.png' ) )
>> I = convolve2d( D, D )
>> np.max(I) 1.1021e+04
```

I\_norm

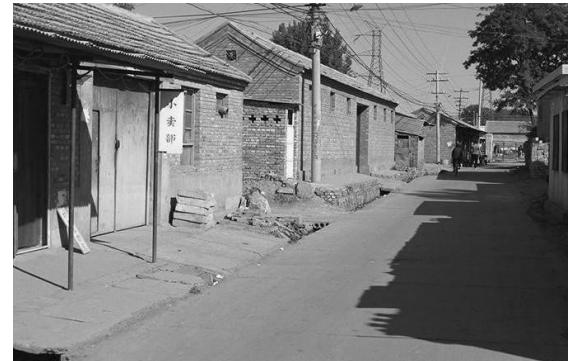
```
# Normalize for visualization
>> I_norm = (I - np.min(I)) / (np.max(I) - np.min(I))
>> plt.imshow( I_norm )
```



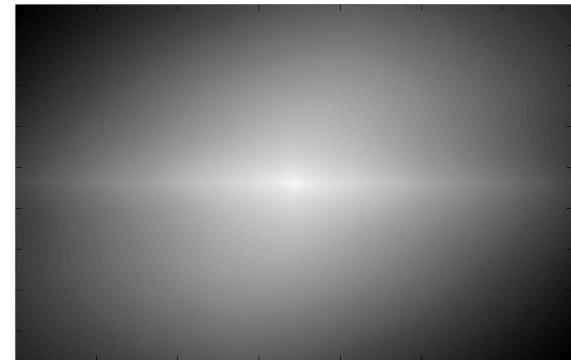
Zero padding causes black regions

# Convolution

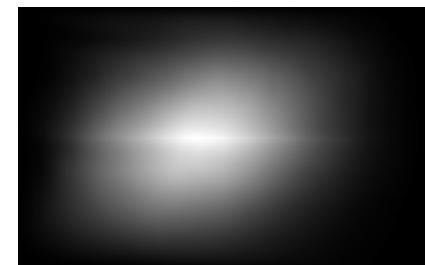
D (275 x 175 pixels)



I (275 x 175 pixels)



```
>> I = convolve2d( D, D, mode='full' )  
(Default; pad with zeros)
```



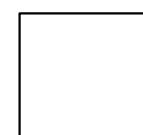
549 x 349

```
>> I = convolve2d( D, D, mode='same' )  
(Return same size as D)
```



275 x 175

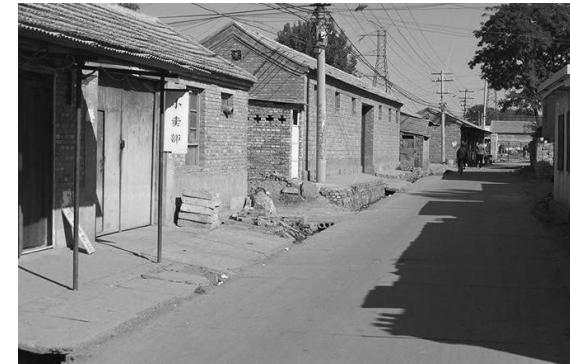
```
>> I = convolve2d( D, D, mode='valid' )  
(No padding)
```



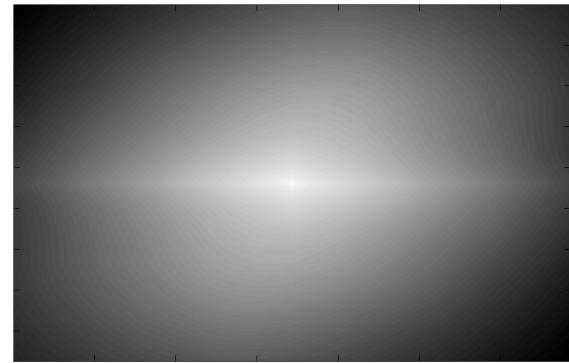
Value = 10528.3 1x1

# Convolution

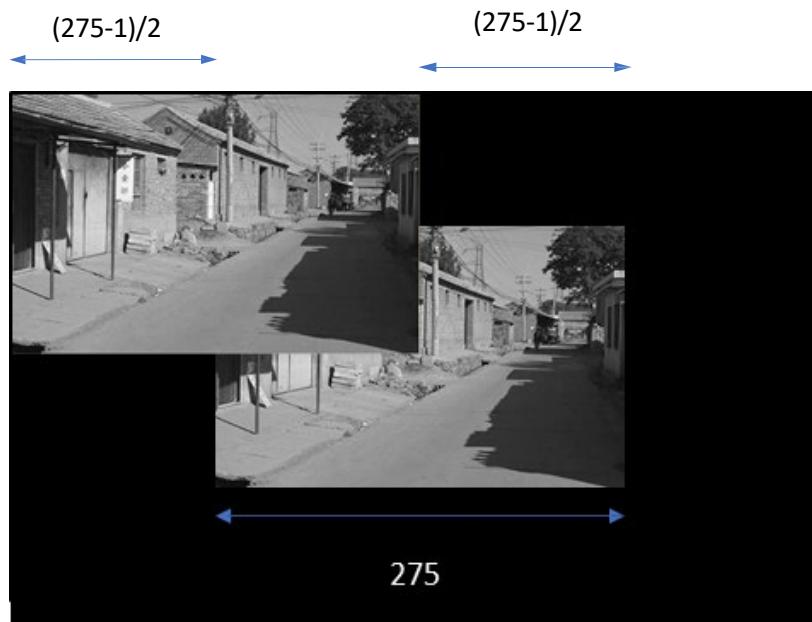
D (275 x 175 pixels)



I (275 x 175 pixels)



```
>> I = convolve2d( D, D, mode='full' )  
(Default; pad with zeros)
```



$$\text{For } x: 275 + (275-1)/2 + (275-1)/2 = 549$$

I\_norm (549 x 349 pixels)



# Correlation

- When the filter ‘looks like’ the image = ‘template matching’
- Filtering viewed as comparing an image of what you want to find against all image regions.
- For **symmetric** filters: use either convolution or correlation.
- For **nonsymmetric** filters: correlation is template matching.

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

e.g., `h = scipy.signal.correlate2d(f, I)`

As brightness in  $I$  increases, the response in  $h$  will increase, as long as  $f$  is positive.



# Correlation

Let's see if we can use correlation to 'find' the parts of the image that look like the filter

D ( $275 \times 175$  pixels).



```
>> f = D[ 57:117, 107:167 ]
```

*Expect response 'peak' in middle of I*

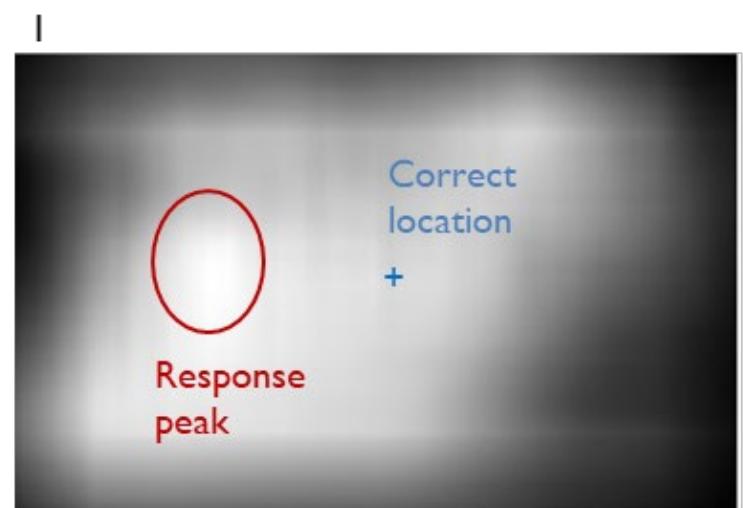
```
>> I = correlate2d( D, f, 'same' )
```

Hmm...

That didn't work – why not?



f  
 $61 \times 61$



# Correlation

Let's see if we can use correlation to 'find' the parts of the image that look like the filter

D (275 x 175 pixels).



```
>> f = D[ 57:117, 107:167 ]
```

*Expect response 'peak' in middle of I*



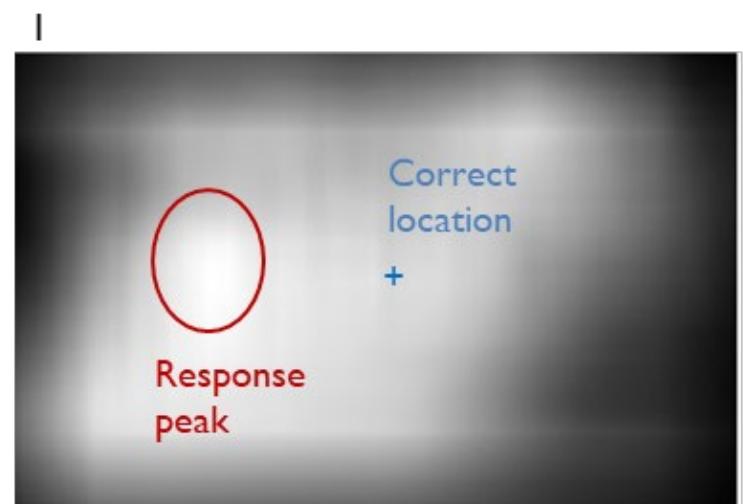
f  
61 x 61

```
>> I = correlate2d( D, f, 'same' )
```

Hmm...

That didn't work – why not?

Overall brighter regions will give higher correlation response  
-> not useful!



# Correlation

OK, so let's subtract the mean

```
>> f = D[ 57:117, 107:167 ]  
>> f2 = f - np.mean(f)  
>> D2 = D - np.mean(D)
```

Now zero centered.

Score is higher only when dark parts match and when light parts match.

```
>> I2 = correlate2d( D2, f2, 'same' )
```

D2 (275 x 175 pixels)



f2  
61 x 61

Remember –  
float data type –  
goes negative!

I2



# Correlation

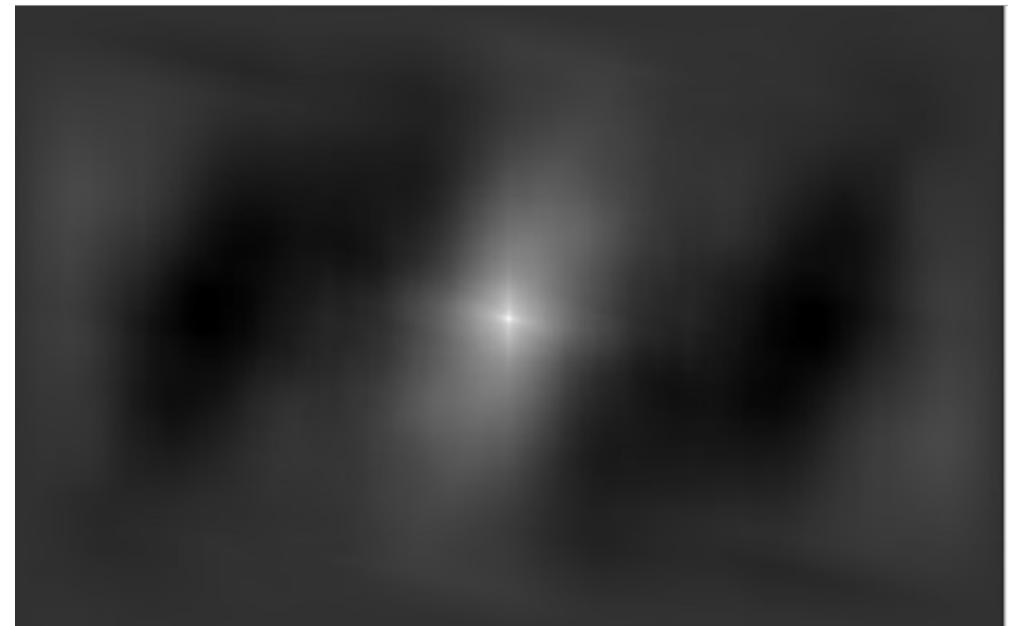
Or our original example: D2 correlated with itself

```
>> I3 = correlate2d( D2, D2, 'full' )
```

D2 (275 x 175 pixels)



I3



# Correlation vs. Convolution

What happens with convolution?

```
>> f = D[ 57:117, 107:167 ]  
>> f2 = f - np.mean(f)  
>> D2 = D - np.mean(D)           I2  
>> I2 = convolve2d( D2, f2, 'same' )
```

D2 (275 x 175 pixels)



f2  
61 x 61



# Non-Linear Filters

- **Rank filters** are non-linear filters based on ordering of gray levels
  - e.g., median, min, max, range filters
- **Median filter** operates over a window by selecting the median intensity in the window

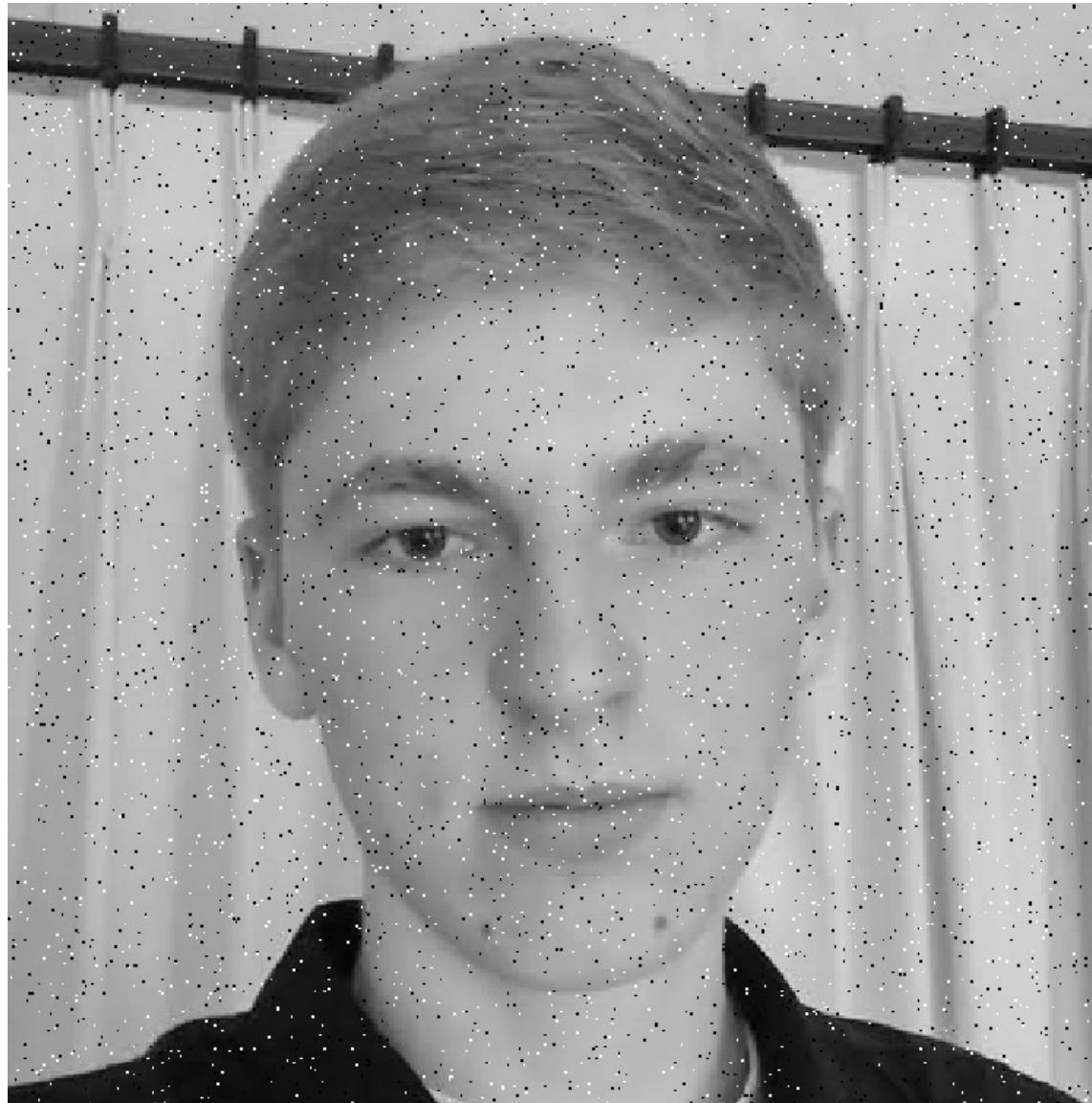
 $I[.,.]$ 

|   |   |    |    |    |    |    |    |   |   |   |
|---|---|----|----|----|----|----|----|---|---|---|
| 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  | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0  | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0  | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0  | 90 | 0  | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0  | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  | 0 | 0 | 0 |
| 0 | 0 | 90 | 0  | 0  | 0  | 0  | 0  | 0 | 0 | 0 |
| 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  | 0 | 0 | 0 |

 $h[.,.]$ 

|  |  |  |  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|--|--|--|
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

# Salt and Pepper Noise



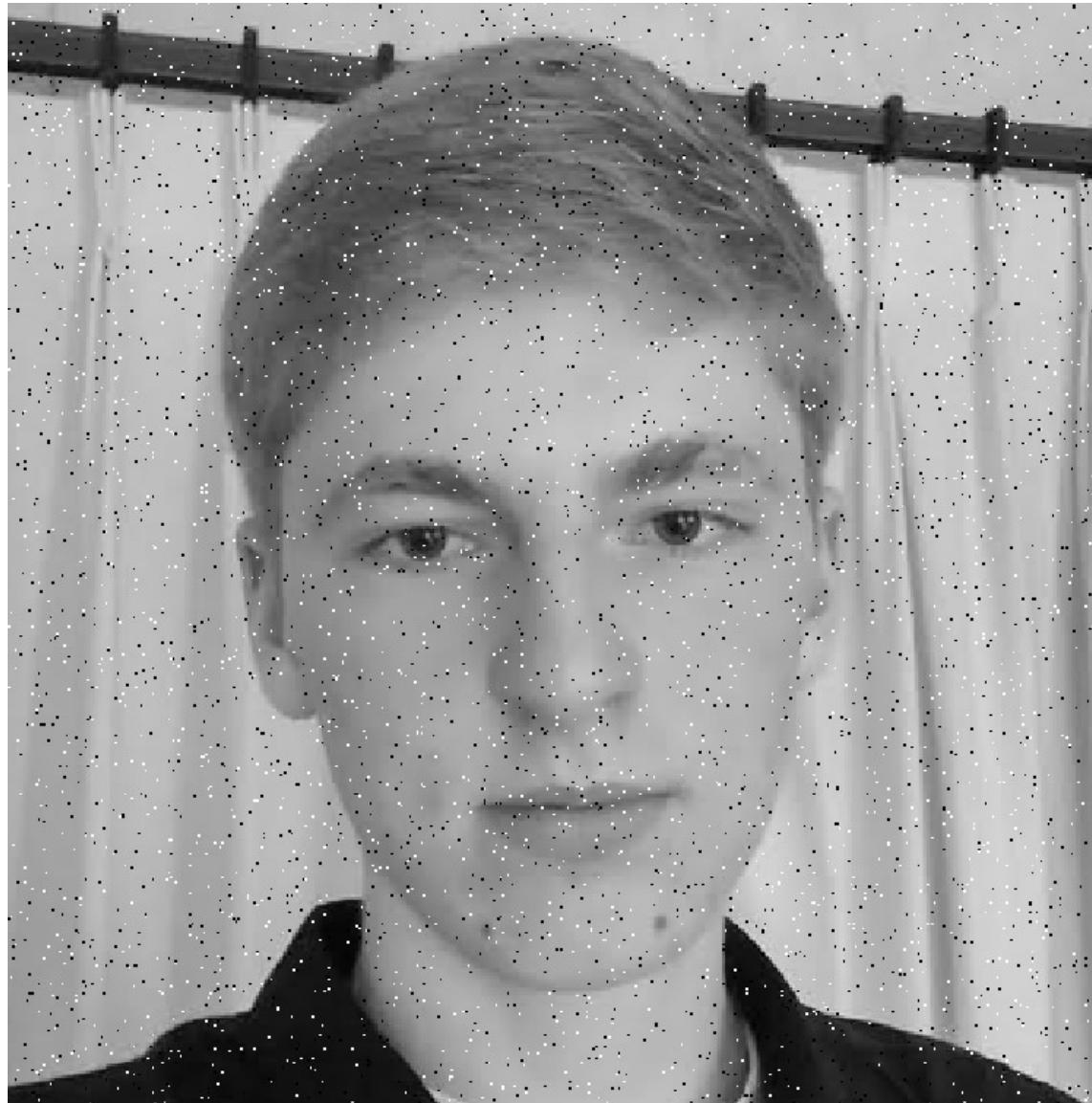
# **3 x 3 Mean Filter**



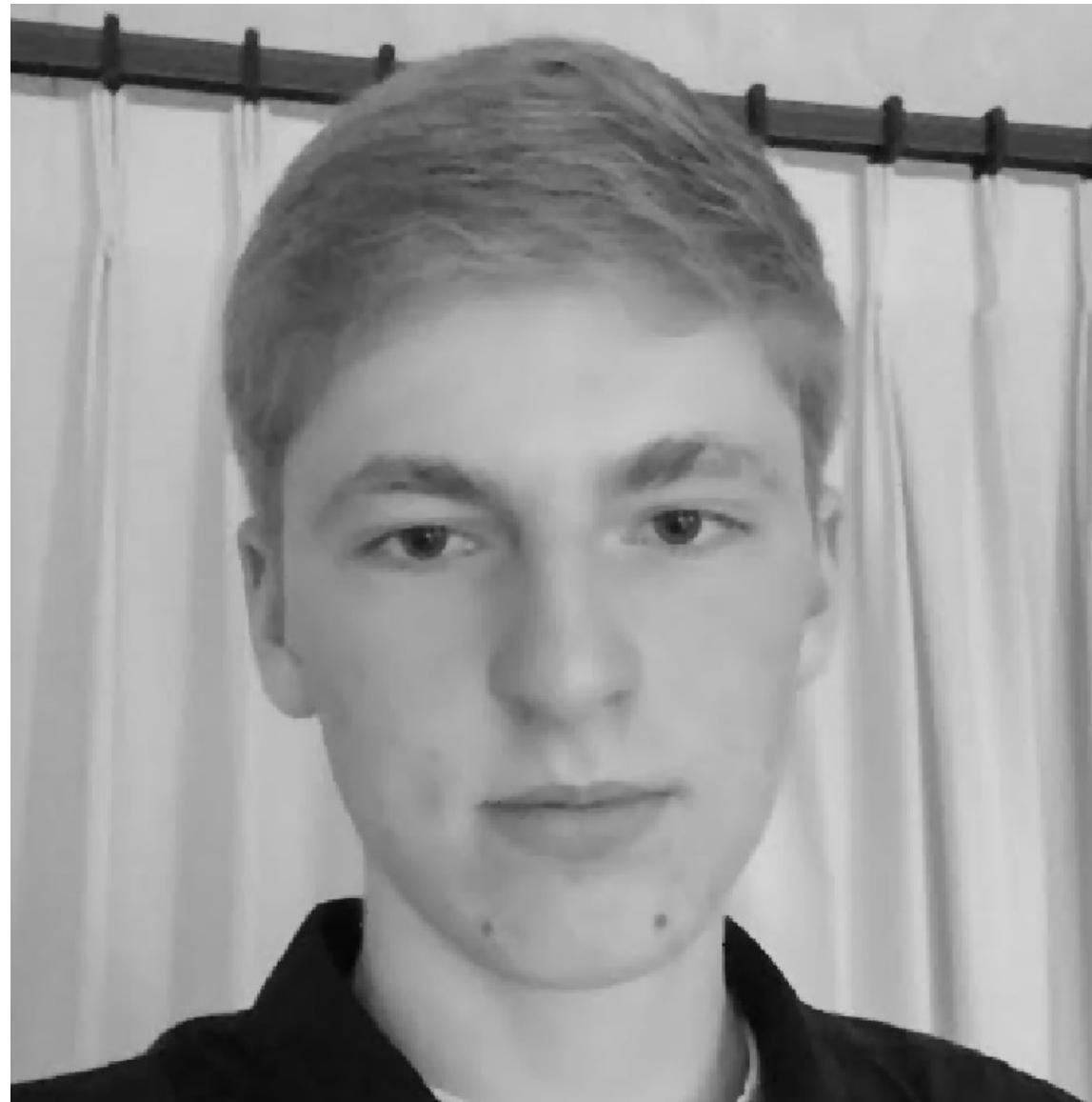
# **11 x 11 Mean Filter**



# Salt and Pepper Noise



# **3 x 3 Median Filter**



# 11 x 11 Median Filter



# Median filters

- Operates over a window by selecting the median intensity in the window
- What advantage does a median filter have over a mean filter?
- Is a median filter a kind of convolution?
- Interpretation: Median filtering is sorting

