CMT107 Visual Computing

Image Filtering

Jing Wu

School of Computer Science and Informatics

Cardiff University

Overview

- Linear filtering
- Convolution
- Box filtering
- Gaussian filtering
- Separable kernel
- Median filter
- Sharpening

Acknowledgement
The majority of the slides in this section are from Svetlana Lazebnik at University of Illinois at Urbana-Champaign

Image Filtering

- Filtering is a technique for modifying or enhancing an image
  - Emphasise certain features or remove other feature
- Filtering is a neighbourhood operation
  - The output value of any given pixel is determined by the values of the pixels in the neighbourhood of the corresponding input pixel

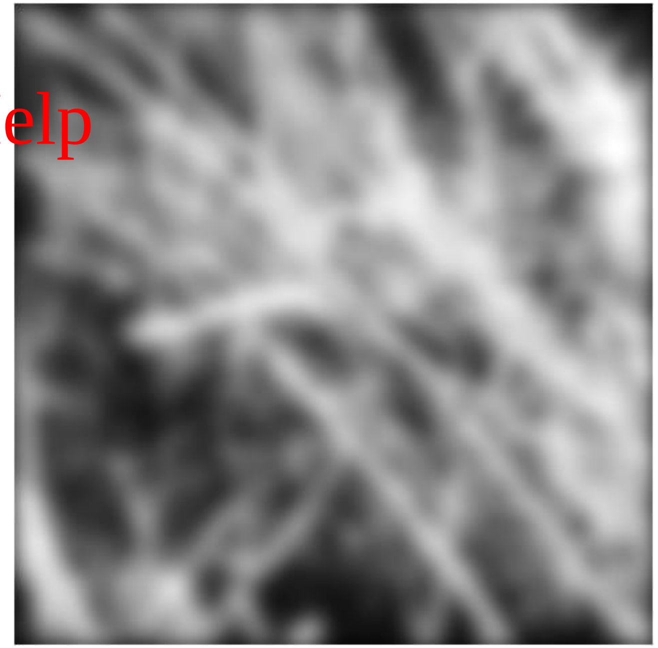
- Linear filtering is filtering in which the value of an output pixel is a linear combination (weighted average) of the values of the pixels in the input pixel's neighbourhood

  - Linear filtering can be represented by convolution

Linear Filtering

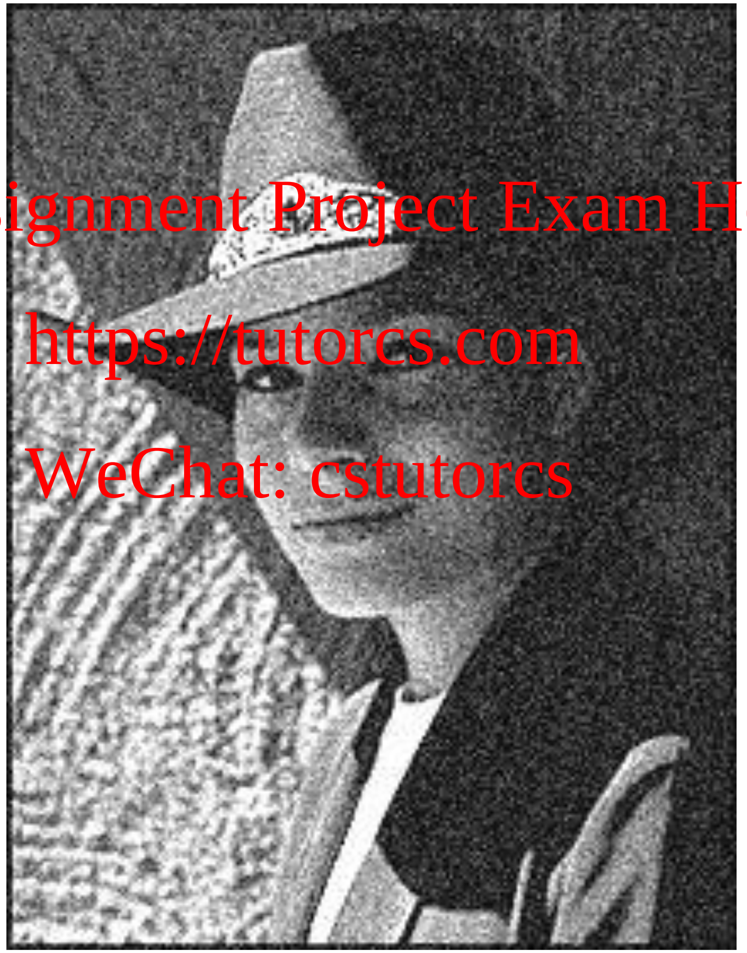
$*$

$=$



Motivation: Image Denoising

- How can we reduce noise in a photograph?



Moving Average

- Let's replace each pixel with a weighted average of its neighbourhood
- The weights are called the filter kernel

Source: D. Lowe

Moving Average

- Let's replace each pixel with a weighted average of its neighbourhood
- The weights are called the filter kernel
- What are the weights for the average of a 3x3 neighbourhood?

Source: D. Lowe

Moving Average

- Let’s replace each pixel with a weighted average of its neighbourhood
- The weights are called the filter kernel
- What are the weights for the average of a 3x3 neighbourhood?

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

“box filter”

Source: D. Lowe

Moving Average

- Let’s replace each pixel with a weighted average of its neighbourhood
- The weights are called the filter kernel
- What are the weights for the average of a 3x3 neighbourhood?

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & \boxed{10} & 1 \\ 1 & 1 & 1 \end{bmatrix} * \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} & & \\ & \boxed{?} & \\ & & \end{bmatrix}$$

“box filter”

Source: D. Lowe

Moving Average

- Let's replace each pixel with a weighted average of its neighbourhood
- The weights are called the filter kernel
- What are the weights for the average of a 3x3 neighbourhood?

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & \boxed{10} & 1 \\ 1 & 1 & 1 \end{bmatrix} * \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} & & \\ & \boxed{2} & \\ & & \end{bmatrix}$$

"box filter"

Source: D. Lowe

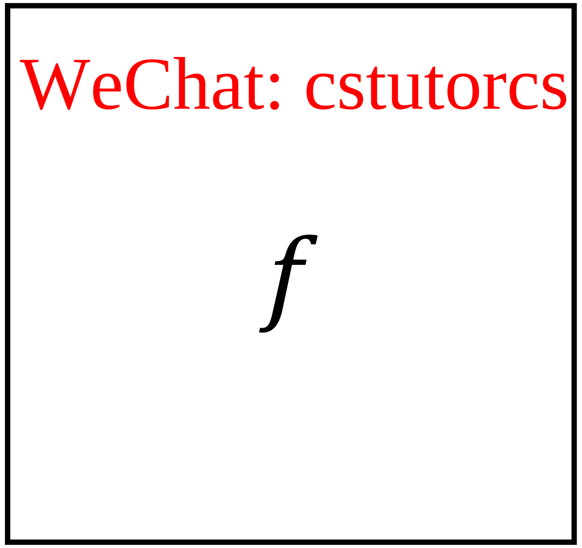
Convolution

- Let f be the image and g be the kernel. The output of convolving f with g is denoted $f * g$

Source: F. Durand

Convolution

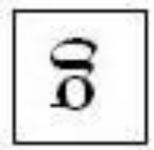
- Let f be the image and g be the kernel. The output of convolving f with g is denoted $f * g$



Source: F. Durand

Convolution

- Let f be the image and g be the kernel. The output of convolving f with g is denoted $f * g$



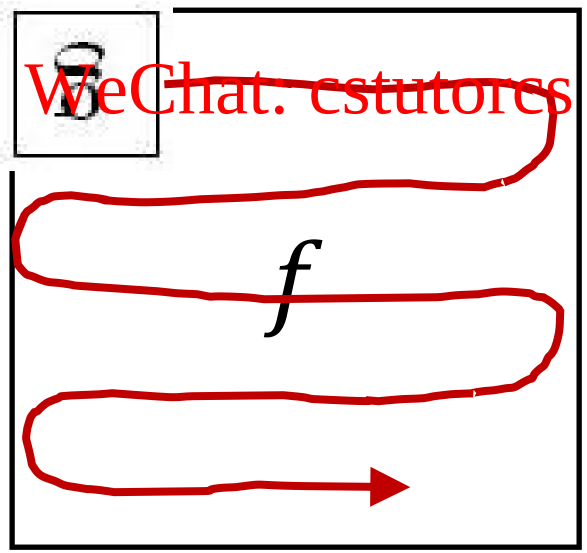
Convention: kernel is flipped for convolution

f

Source: F. Durand

- Let f be the image and g be the kernel. The output of convolving f with g is denoted $f * g$

Convention: kernel is flipped for convolution



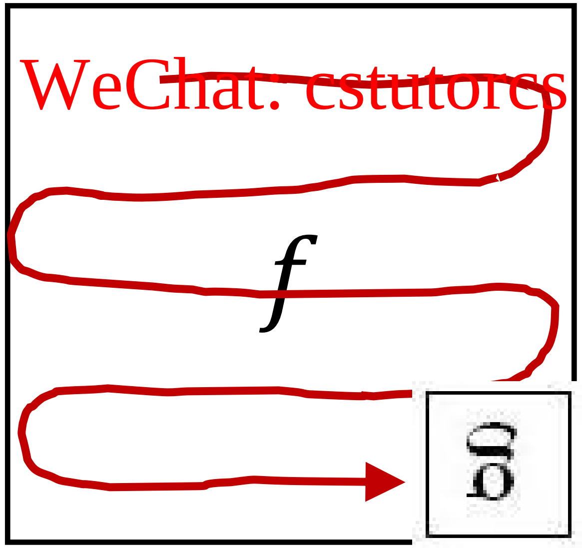
Source: F. Durand

Convolution

- Let f be the image and g be the kernel. The output of convolving f with g is denoted $f * g$

$$(f * g)[x, y] = \sum_{i=-k}^{k} \sum_{j=-l}^{l} f[x-i, y-j]\, g[i, j]$$

Convention: kernel is flipped for convolution



Source: F. Durand

Linear Filter: Key Properties

- Linearity: filter($f_1 + f_2$) = filter(f_1) + filter(f_2)
- Shift invariance: same behaviour regardless of pixel location

  filter(shift(f)) = shift(filter(f))

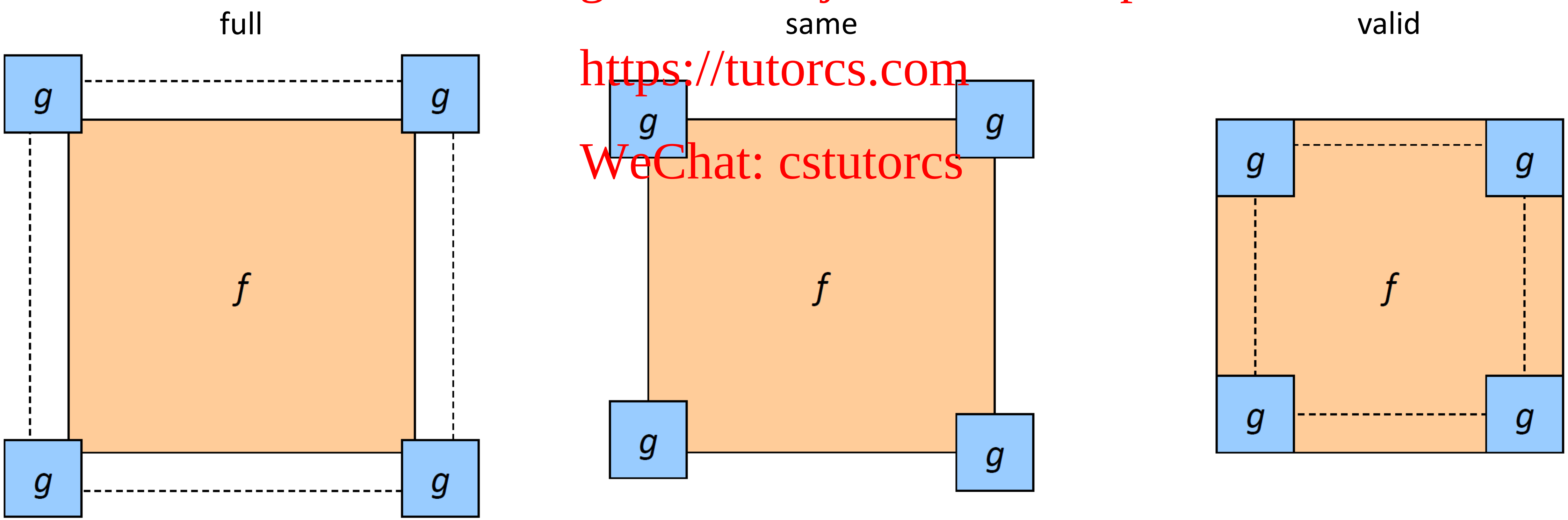
- Theoretical result: any linear shift-invariant operator can be represented as a convolution

Linear Filter: More Properties

- Commutative: $a * b = b * a$
  - Conceptually no difference between filter and signal
- 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: $a * (b_1 * b_2 * b_3)$
- Distributive over addition: $a * (b + c) = (a * b) + (a * c)$
- Scalars factor out: $ka * b = a * kb = k(a * b)$
- Identity: unit pulse $e = [\dots, 0, 0, 1, 0, 0, \dots]$, $a * e = a$

Size of the Output

- "full": output size is the sum of sizes of f and g minus 1
- "same": output size is the same as the size of f
- "valid": output size is the difference of the sizes of f and g



Boundary Pixels

- What about near the edge?
  - The filter window falls off the edge of the image
  - Need to extrapolate
  - Method
    - Clip filter (black)
    - Wrap around
    - Copy edge
    - Reflect across edge

Source: S. Marschner

Boundary Pixels

- Clip filter (black)

Source: S. Marschner

Boundary Pixels

- Wrap around

Source: S. Marschner

Boundary Pixels

- Copy edge

Source: S. Marschner

Boundary Pixels

- Reflect across edge

Source: S. Marschner

Practice with Linear Filter

Original

*

0	0	0
0	1	0
0	0	0

=

?

Source: D. Lowe

Practice with Linear Filter

Original

$*$

0	0	0
0	1	0
0	0	0

$=$

Filtered
(no change)

Source: D. Lowe

Practice with Linear Filter

Original

*

0	0	0
1	0	0
0	0	0

=

?

Source: D. Lowe

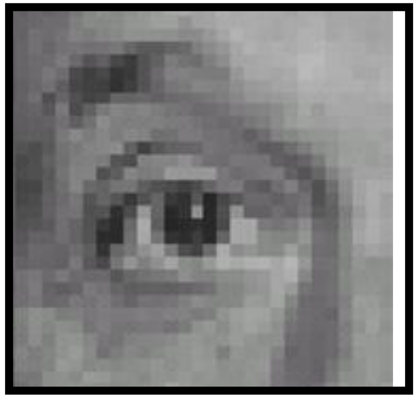
Practice with Linear Filter

Original

$*$

0	0	0
1	0	0
0	0	0

$=$



Shifted left
by 1 pixel

Source: D. Lowe

Practice with Linear Filter

$$* \quad \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \quad ?$$

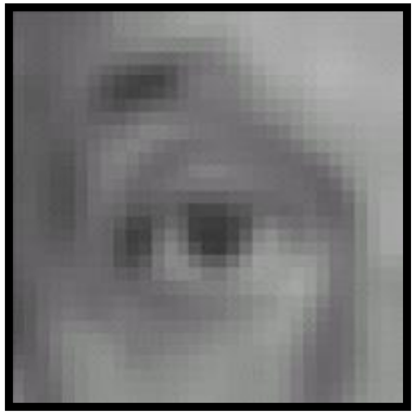
Original

Source: D. Lowe

Practice with Linear Filter

Original

$$* \quad \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad =$$



Blur
(with a box filter)

Source: D. Lowe

Practice with Linear Filter

$$* \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \right) = \ ?$$

Original

(Note that filter weights sum to 1)

Source: D. Lowe

Practice with Linear Filter

$$* \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \right) =$$

Original

(Note that filter weights sum to 1)

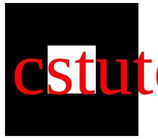
Sharpening filter:
Accentuates differences with local average

Source: D. Lowe

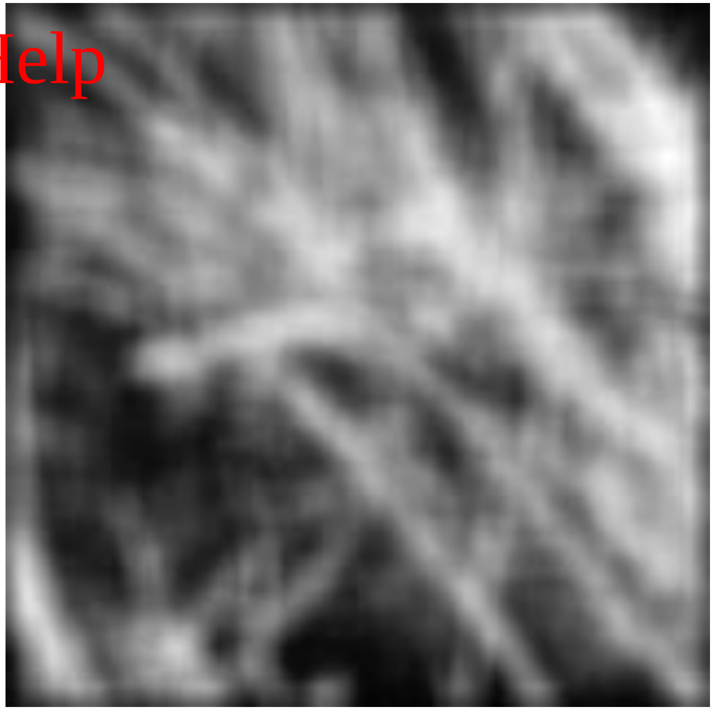
Smoothing with Box Filter revisited

- What’s wrong with this picture?
- What’s the solution?

*



=



Source: D. Forsyth

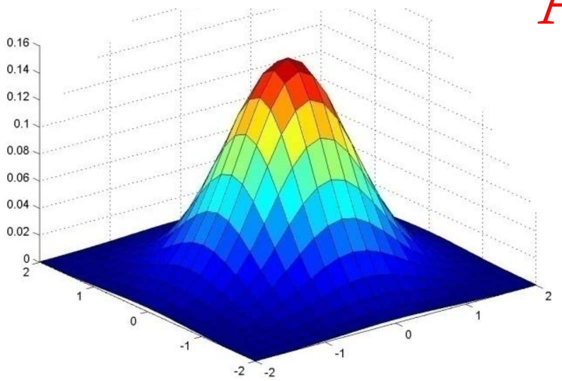
Smoothing with Box Filter revisited

- What’s wrong with this picture?
- What’s the solution?
    - To eliminate edge effect, weight contribution of neighbourhood pixels according to their closeness to the centre.

“fuzzy blob”

Gaussian Kernel

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



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
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

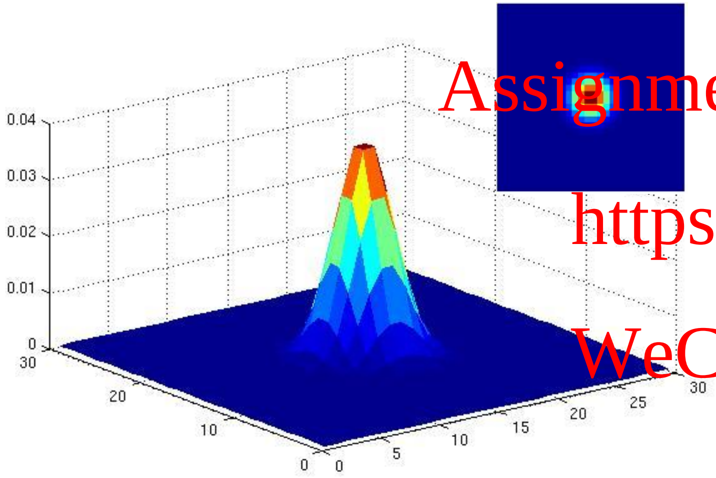
5 x 5, σ = 1

- Constant factor at front makes volume sum to 1 (can be ignored when computing the filter values, as we should renormalize weights to sum to 1 in any case

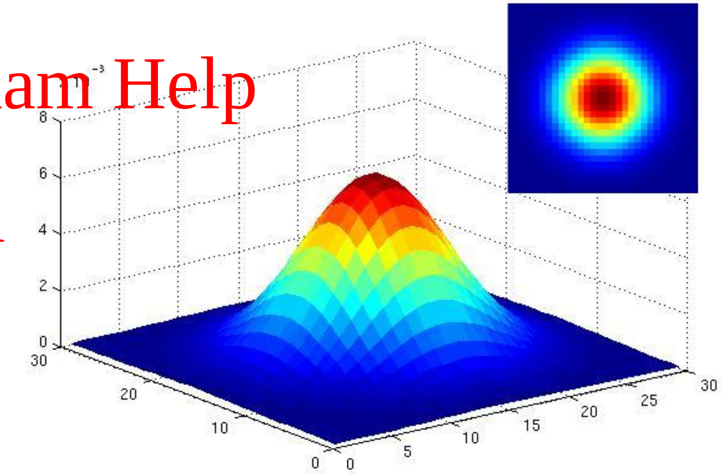
Source: C. Rasmussen

Gaussian Kernel

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



σ = 2 with 30 x 30 kernel



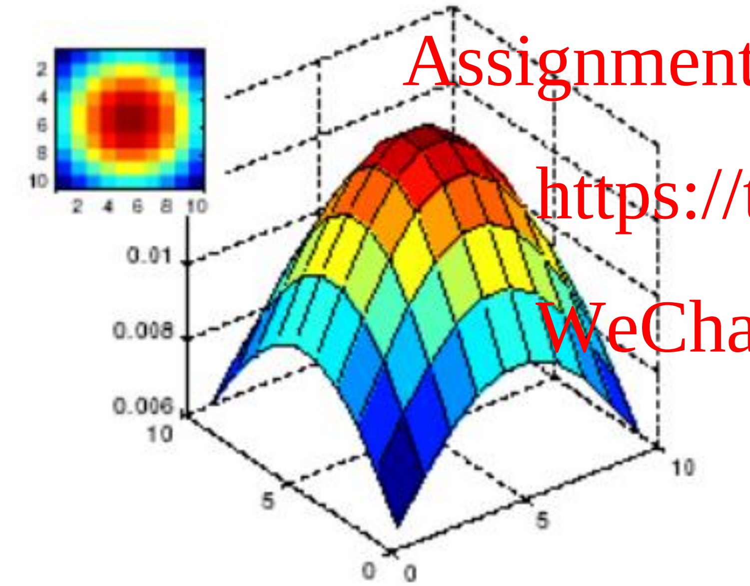
σ = 5 with 30 x 30 kernel

- Standard deviation σ determines the extent of smoothing

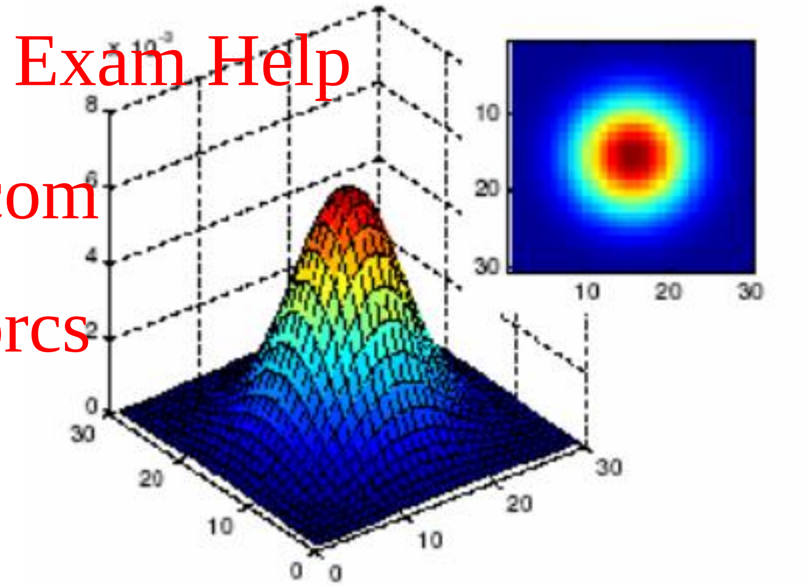
Source: K. Grauman

Choosing Kernel Width

- The Gaussian function has infinite support, but discrete filters use finite kernels



σ = 5 with 10 x 10 kernel

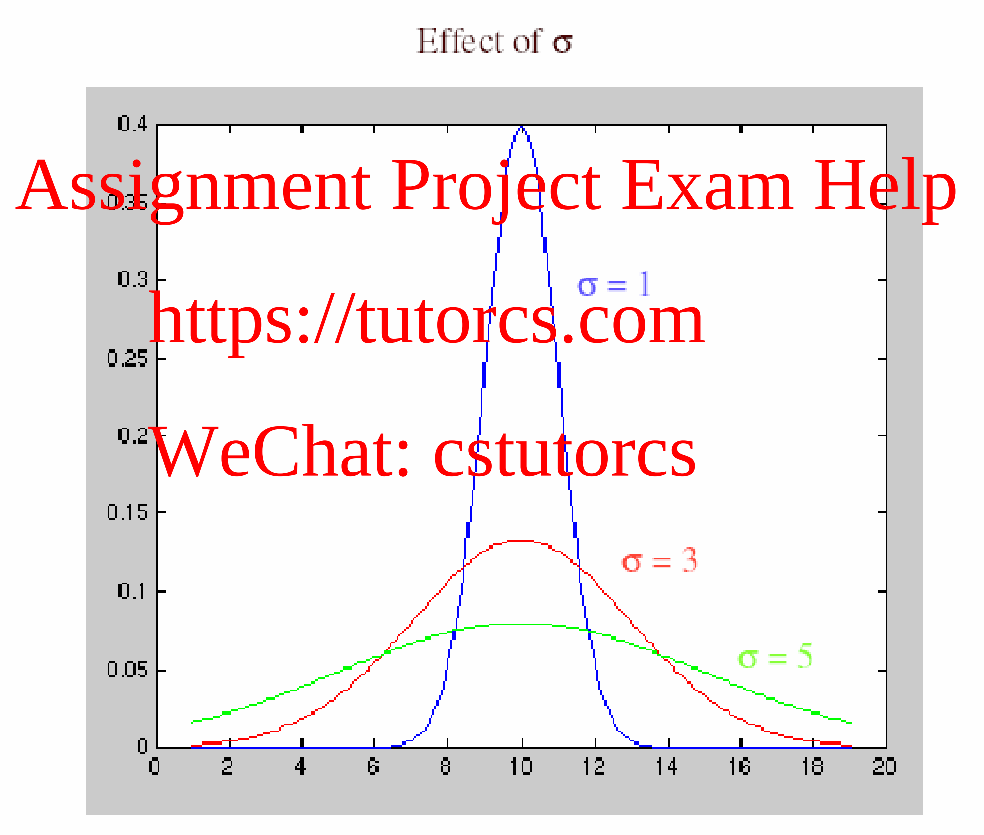


σ = 5 with 30 x 30 kernel

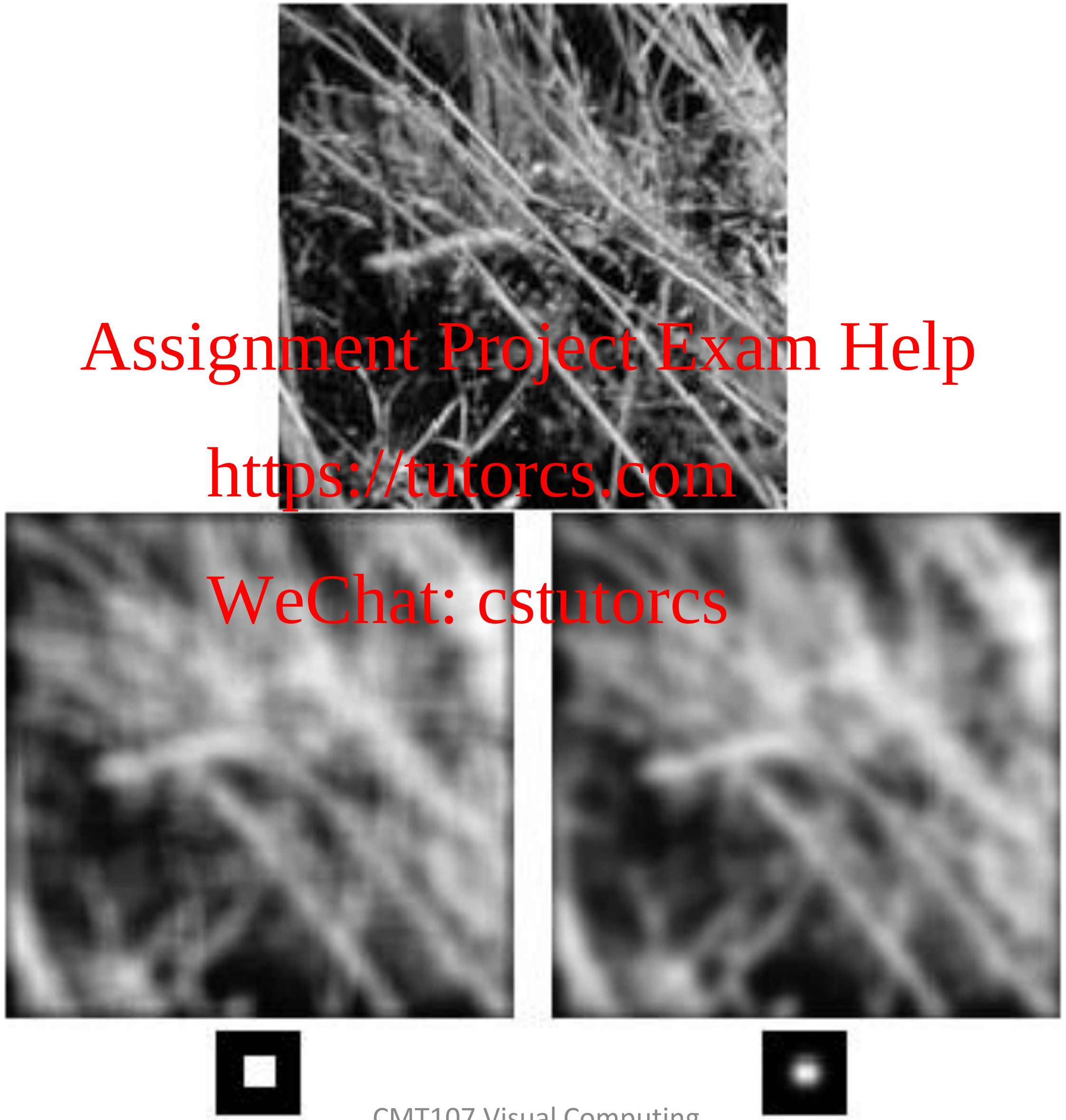
Source: K. Grauman

Choosing Kernel Width

- Rule of thumb: set filter half width to about 3σ



Gaussian vs. Box Filtering



Gaussian Filters

- Remove "high frequency" component from the image (low-pass filter)
- Convolution with self is another Gaussian
  - So can smooth with small-σ kernel, repeat, and get same result as large-σ kernel would have
  - Convolving two times with Gaussian kernel with standard deviation σ is the same as convolving one with Gaussian kernel with standard deviation $\sigma\sqrt{2}$
- Separable kernel
  - Factors into product of two 1D Gaussians

Source: K. Grauman

Separability of the Gaussian Filter

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

- The 2D Gaussian can be expressed as the product of two 1D functions: one is a function of x, and the other is a function of y.

- In this case, the two functions are the identical 1D Gaussian.

Source: D. Lowe

Separability Example

2D convolution
(centre location only)

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

Source: K. Grauman

Separability Example

2D convolution
(centre location only)

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

$$= 1 \times 2 + 2 \times 3 + 1 \times 3 = 11$$
$$= 2 \times 3 + 4 \times 5 + 2 \times 5 = 36$$
$$= 1 \times 4 + 2 \times 4 + 1 \times 6 = \underline{18}$$
$$65$$

Source: K. Grauman

Separability Example

2D convolution
(centre location only)

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

$$= 1 \times 2 + 2 \times 3 + 1 \times 3 = 11$$
$$= 2 \times 3 + 4 \times 5 + 2 \times 5 = 36$$
$$= 1 \times 4 + 2 \times 4 + 1 \times 6 = \underline{18}$$
$$65$$

The filter factors into a product of 1D filters

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

Source: K. Grauman

Separability Example

2D convolution
(centre location only)

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

$$\begin{aligned} &= 1 \times 2 + 2 \times 3 + 1 \times 3 = 11 \\ &= 2 \times 3 + 4 \times 5 + 2 \times 5 = 36 \\ &= 1 \times 4 + 2 \times 4 + 1 \times 6 = \underline{18} \\ &\phantom{= 1 \times 4 + 2 \times 4 + 1 \times 6 = {}} 65 \end{aligned}$$

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

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

Source: K. Grauman

Separability Example

2D convolution
(centre location only)

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

$$\begin{aligned} &= 1 \times 2 + 2 \times 3 + 1 \times 3 = 11 \\ &= 2 \times 3 + 4 \times 5 + 2 \times 5 = 36 \\ &= 1 \times 4 + 2 \times 4 + 1 \times 6 = \underline{18} \\ & \qquad\qquad\qquad\qquad\qquad\quad 65 \end{aligned}$$

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

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

Source: K. Grauman

Separability Example

2D convolution
(centre location only)

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

$$\begin{aligned} &= 1 \times 2 + 2 \times 3 + 1 \times 3 = 11 \\ &= 2 \times 3 + 4 \times 5 + 2 \times 5 = 36 \\ &= 1 \times 4 + 2 \times 4 + 1 \times 6 = \underline{18} \\ &\qquad\qquad\qquad\qquad\qquad\quad\; 65 \end{aligned}$$

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

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

Source: K. Grauman

Separability Example

2D convolution
(centre location only)

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

$$= 1 \times 2 + 2 \times 3 + 1 \times 3 = 11$$
$$= 2 \times 3 + 4 \times 5 + 2 \times 5 = 36$$
$$= 1 \times 4 + 2 \times 4 + 1 \times 6 = \underline{18}$$
$$65$$

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

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

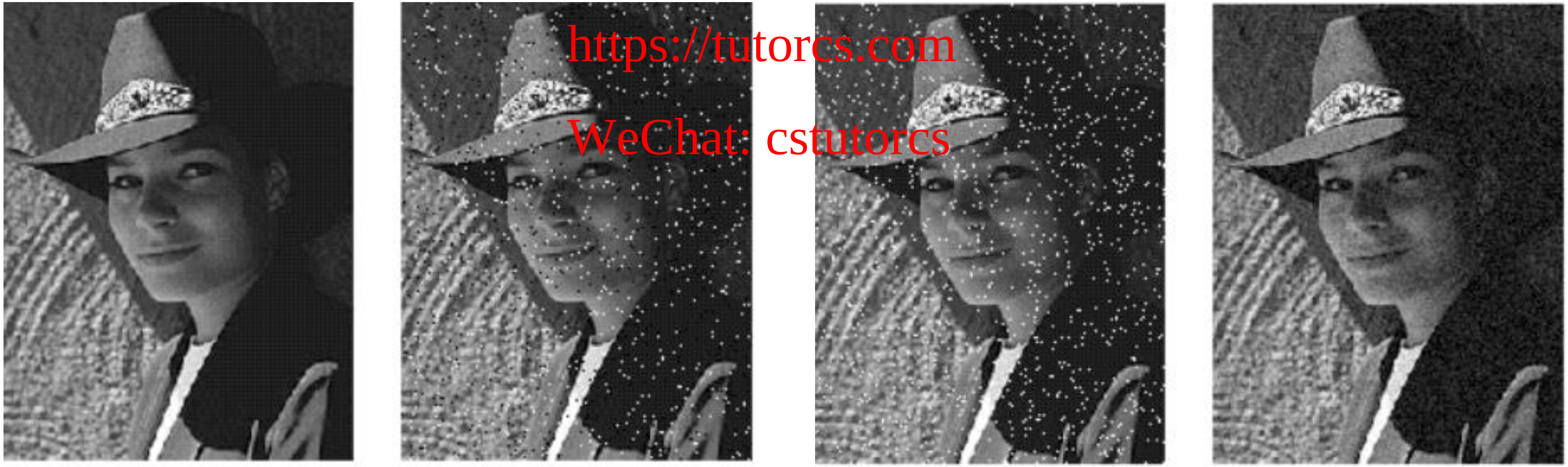
Source: K. Grauman

Why is Separability Useful

- What is the complexity of filtering an $n \times n$ image with an $m \times m$ kernel?
  - $O(n^2 \times m^2)$
- What if the kernel separable?
  - $O(n^2 \times m)$

Noise

- **Salt and pepper noise:** contains random occurrences of black and white pixels
- **Impulse noise:** contains random occurrences of white pixels
- **Gaussian noise:** variations in intensity drawn from a gaussian normal distribution



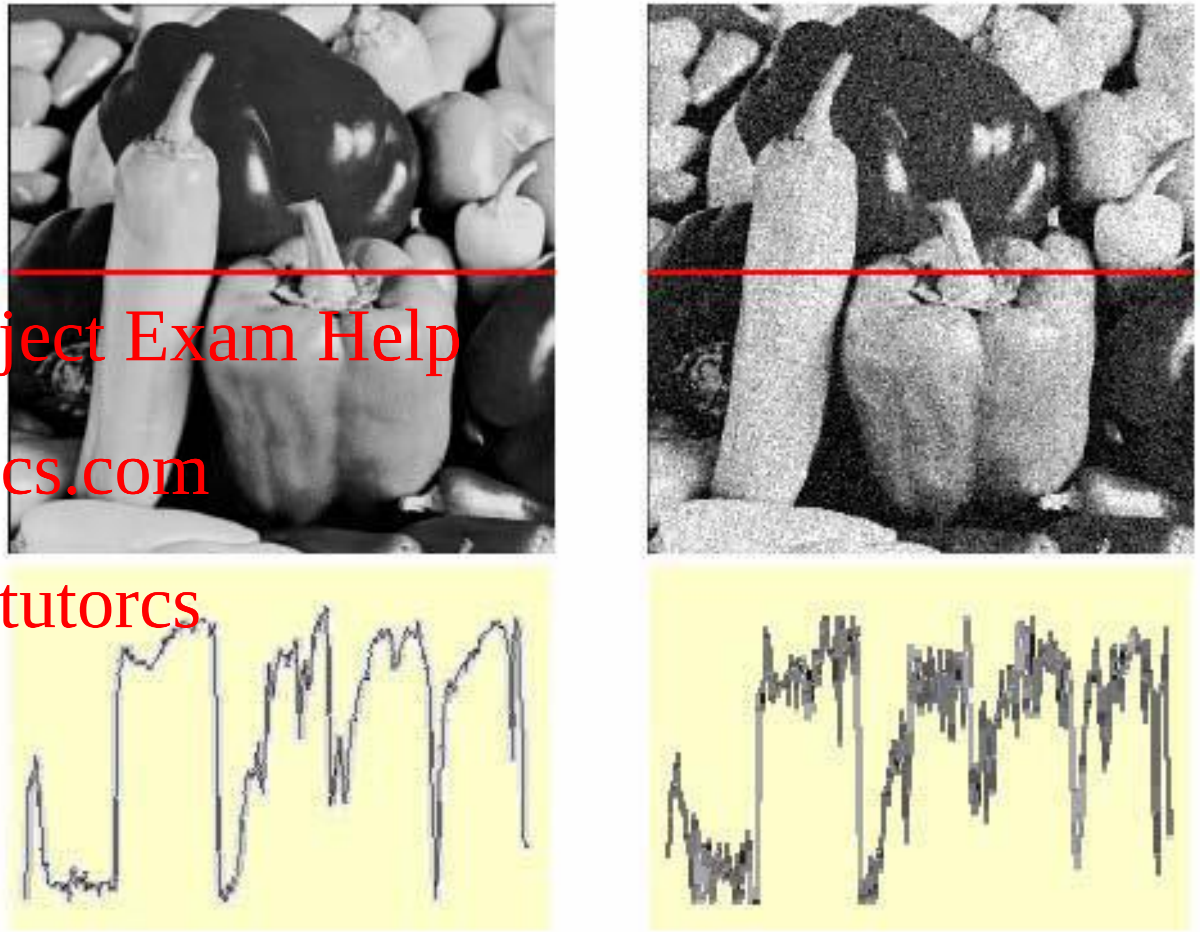
Original | Salt and pepper noise | Impulse noise | Gaussian noise

Source: S. Seitz

Gaussian Noise

- Mathematical model: sum of many independent factors
- Good for small standard deviations
- Assumption: independent, zero-mean noise

Image Noise

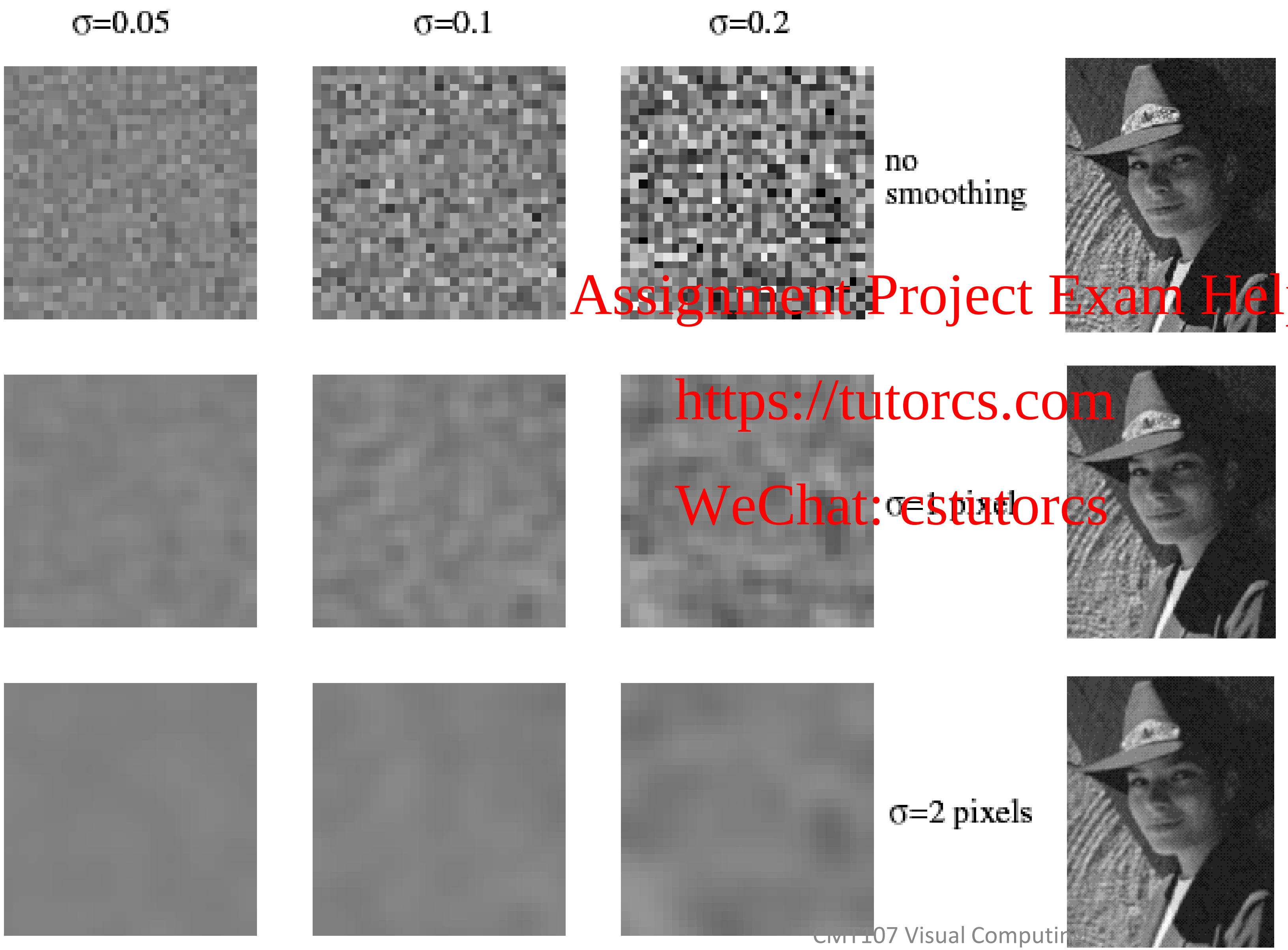


$$f(x,y) = \overbrace{\tilde{f}(x,y)}^{\text{Ideal Image}} + \overbrace{\eta(x,y)}^{\text{Noise process}}$$

Gaussian i.i.d. ("white") noise: $\eta(x,y) \sim \mathcal{N}(\mu,\sigma)$

Source: M. Hebert

Reducing Gaussian Noise

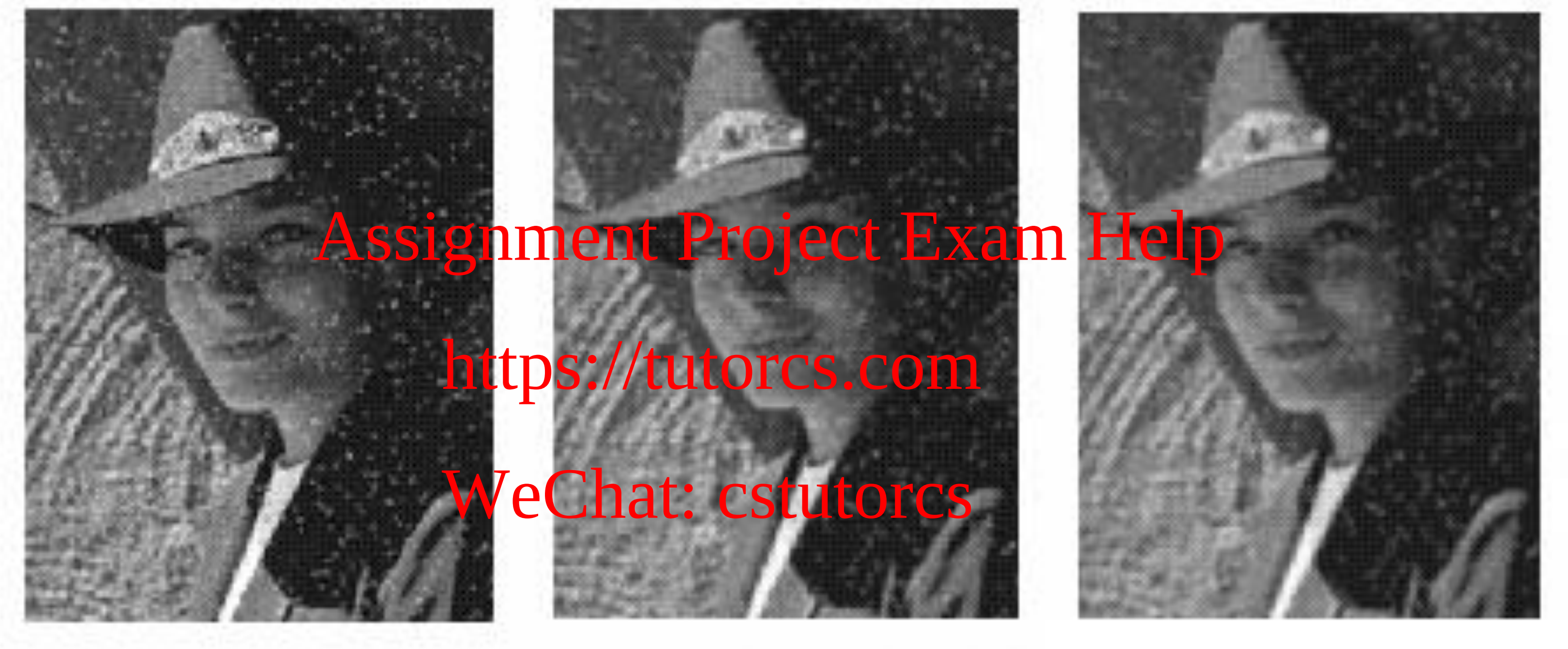


Noise supressed

Image blurred

Reducing Salt and Pepper Noise

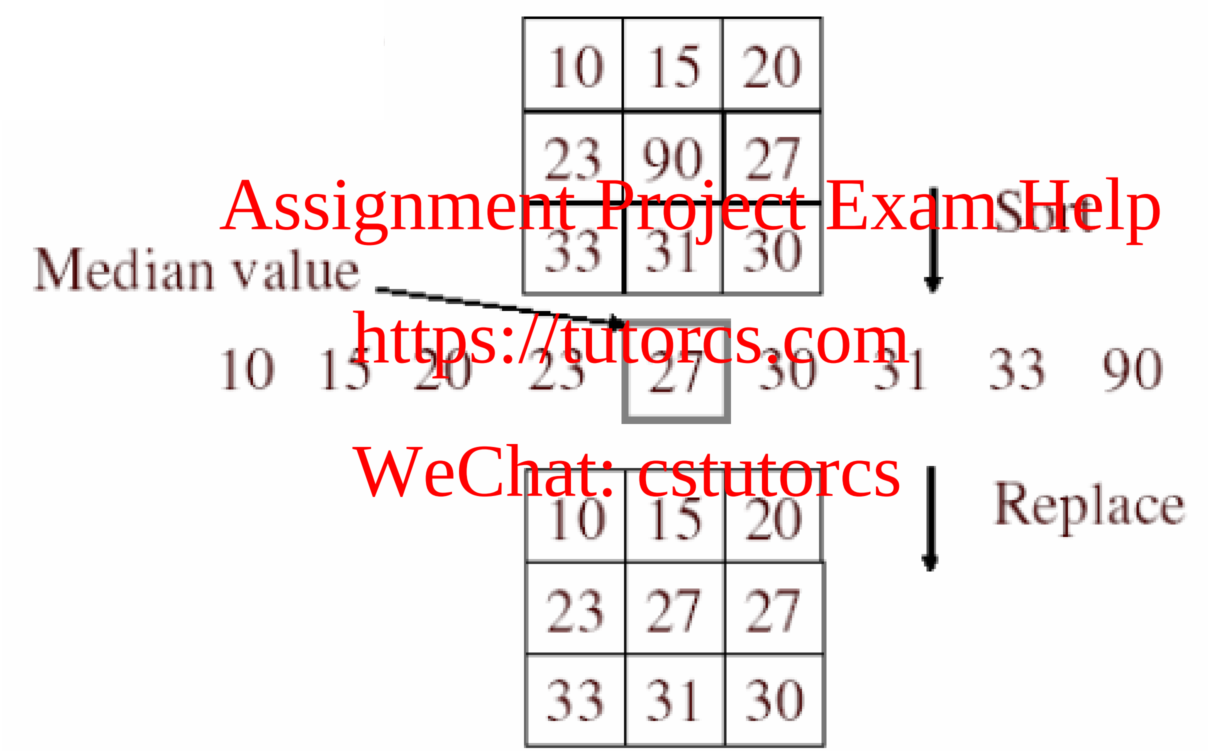
3x3 5x5 7x7



- What's wrong with the results?

Alternative Idea: Median Filtering

- A median filter operates over a window by selecting the median intensity in the window



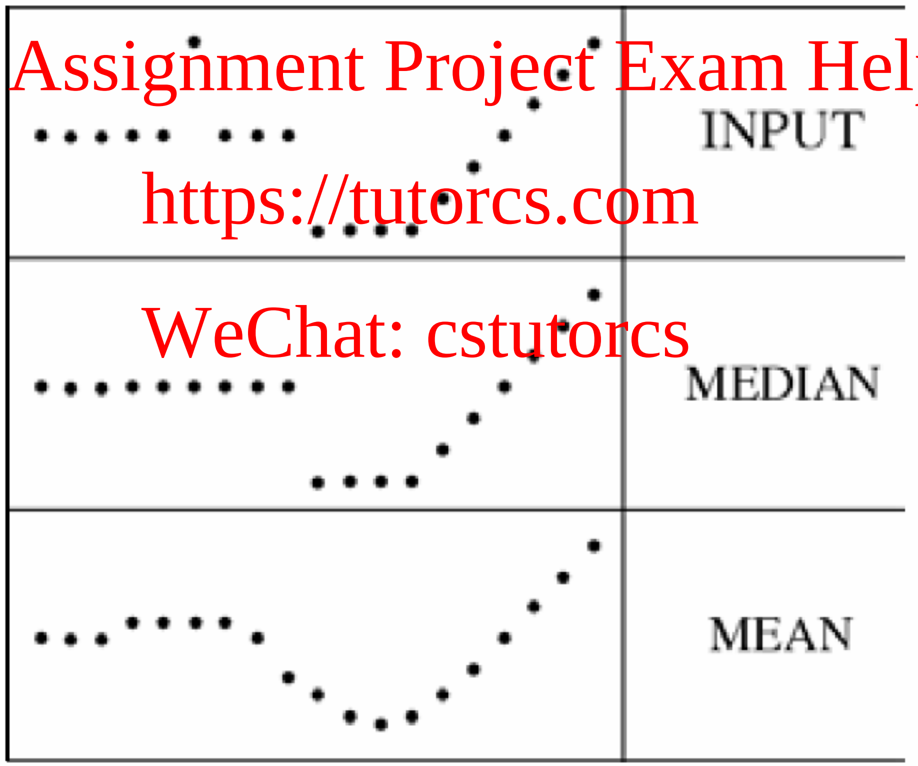
- Is median filtering linear?

Source: K. Grauman

Median Filter

- What advantage does median filtering have over Gaussian filtering?
  - Robustness to outliers

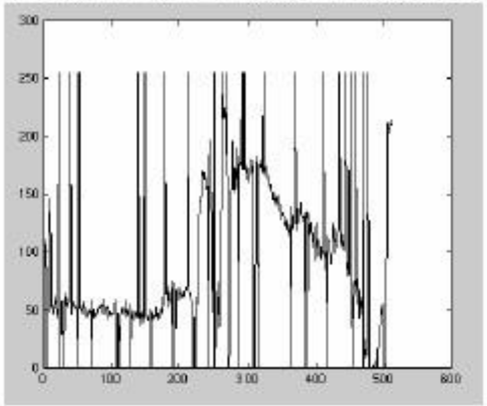
filters have width 5 :



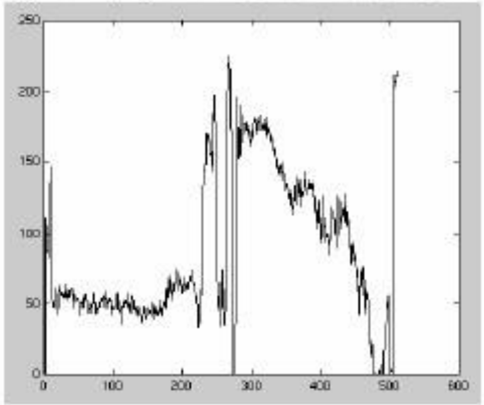
Source: K. Grauman

Median Filter

Salt-and-pepper noise



Median filtered



Source: M. Hebert

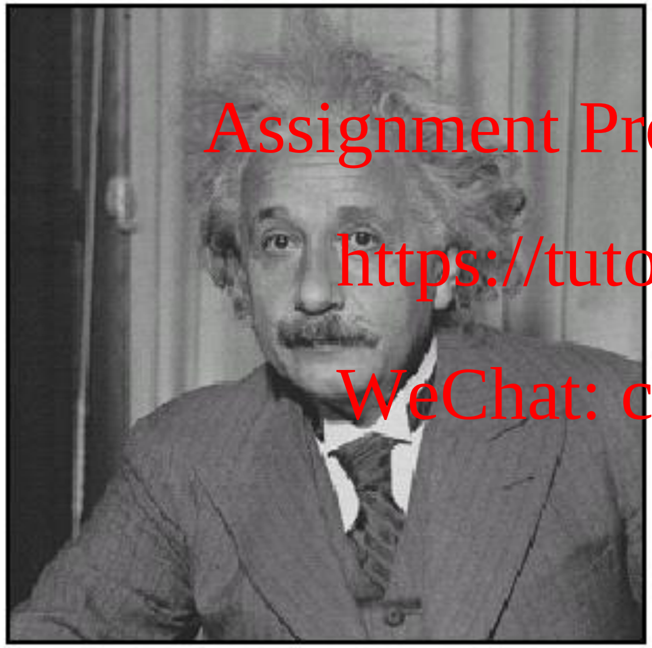
Gaussian vs. Median Filter

3x3 5x5 7x7

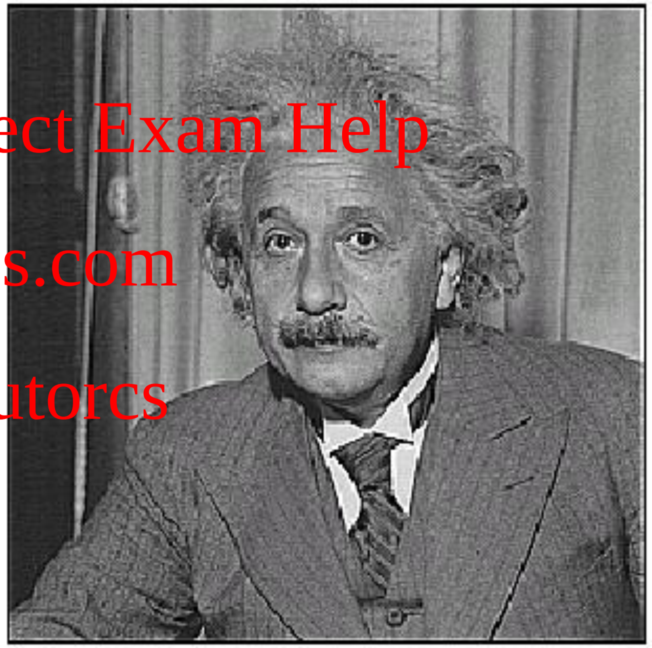
Gaussian

Median

Sharpening revisited



before



after

Source: D. Lowe

Sharpening revisited

- What does blurring take away?

–

Sharpening revisited

- What does blurring take away?

–

=

Sharpening revisited

- What does blurring take away?

−

=

- Let’s add it back

+ α

Sharpening revisited

- What does blurring take away?

−

=

- Let’s add it back

+ α

=

Unsharp Mask Filter

$$f + \alpha(f - f * g) = (1 + \alpha)f - \alpha f * g = f * ((1 + \alpha)e - \alpha g)$$

image

blurred image

unit impulse (identity)

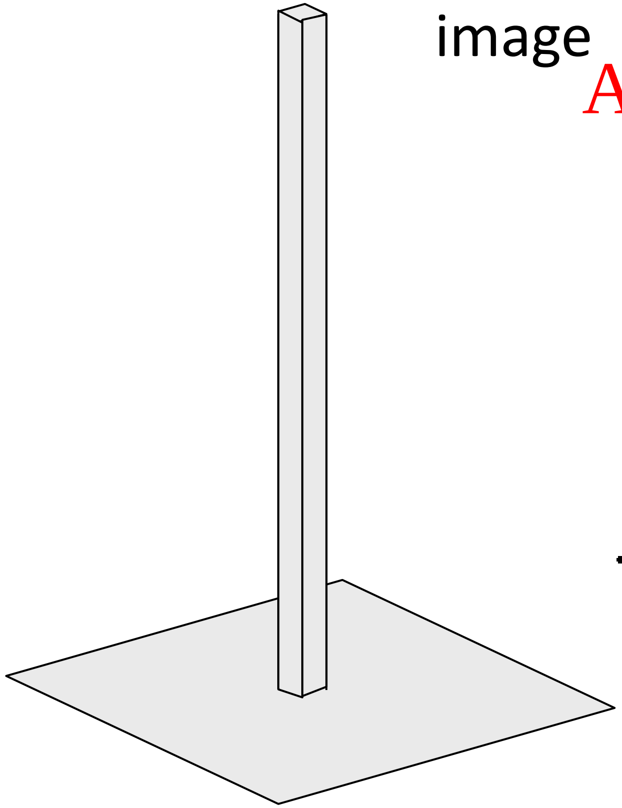
Unsharp Mask Filter

$$f + \alpha(f - f * g) = (1 + \alpha)f - \alpha f * g = f * ((1 + \alpha)e - \alpha g)$$

image

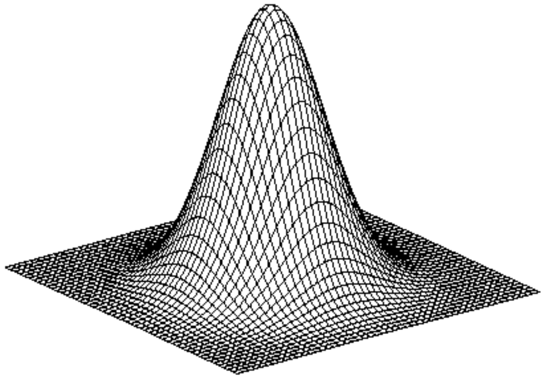
blurred image

unit impulse (identity)



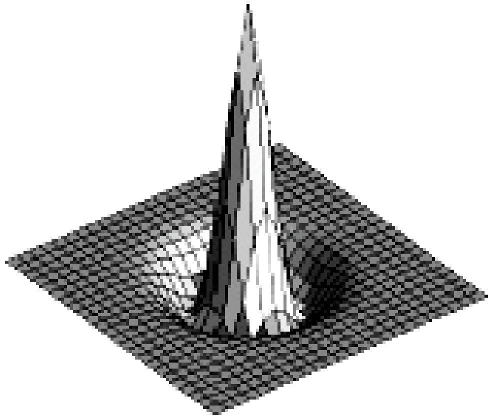
unit impulse

$-$



Gaussian

$\approx$



Laplacian of Gaussian

Image Filtering with Java

- Use filter() in BufferedImageOp
- Implement filtering without using filter() function

Use filter() Function

- Define a filter kernel

```
float[] km = {          // low-pass filter kernel
    0.1f, 0.1f, 0.1f,   // Suppose the matrix has been flipped
    0.1f, 0.2f, 0.1f,
    0.1f, 0.1f, 0.1f
};
Kernel kernel = new Kernel(3, 3, km);
```


ConvolveOp(Kernel kernel, int edgeCondition, RenderingHints hints)

- edgeCondition: ConvolveOp.EDGE_NO_OP or ConvolveOp.EDGE_ZERO_FILL

- Define an operator

```
BufferedImageOp op = null;
op = new ConvolveOp(kernel, ConvolveOp.EDGE_NO_OP, null);
```

- Call the filter() function

```
out = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
op.filter(in, out);
```

Not Use filter() Function

- Define a filter kernel matrix

```
float[] km = {          // low-pass filter kernel
    0.1f, 0.1f, 0.1f,   // Suppose the matrix has been flipped
    0.1f, 0.2f, 0.1f,
    0.1f, 0.1f, 0.1f
};
```


- Calculate convolution on each pixel

```
int[] rArray = new int[width*height]; //
for each pixel {
    get the neighbourhood colours of the pixel
    calculate the colour according to the convolution formula
    set the pixel colour in the output image
}
```

  - More details in Lab session 6

Summary

- What is filtering? What is linear filtering?
- What is convolution?
- How to do sharpening of image?
- What is box filtering, Gaussian filtering, and median filtering?
- What is separable kernel? Why use separable kernel?