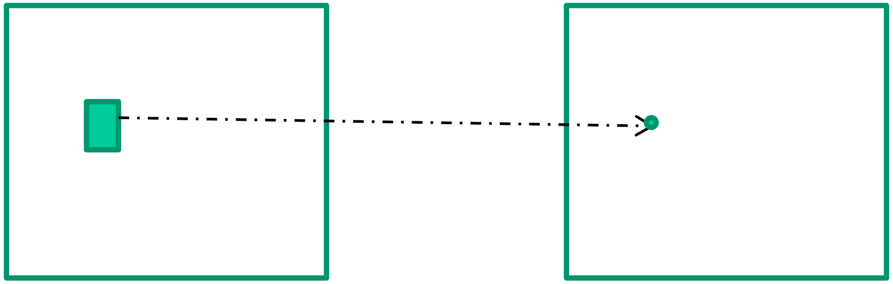
Neighbourhood processing



$$I_{out}[m_0,n_0] = I_{in}\,[m,n];\ [m,n] \in N[m_0,n_0]$$

Useful 2D functions

Unit impulse function in 2D:

$\delta[m,n] = 1$; $[m,n] = 0$

$= 0$ otherwise

Unit pulse (Rect) function in 2D:

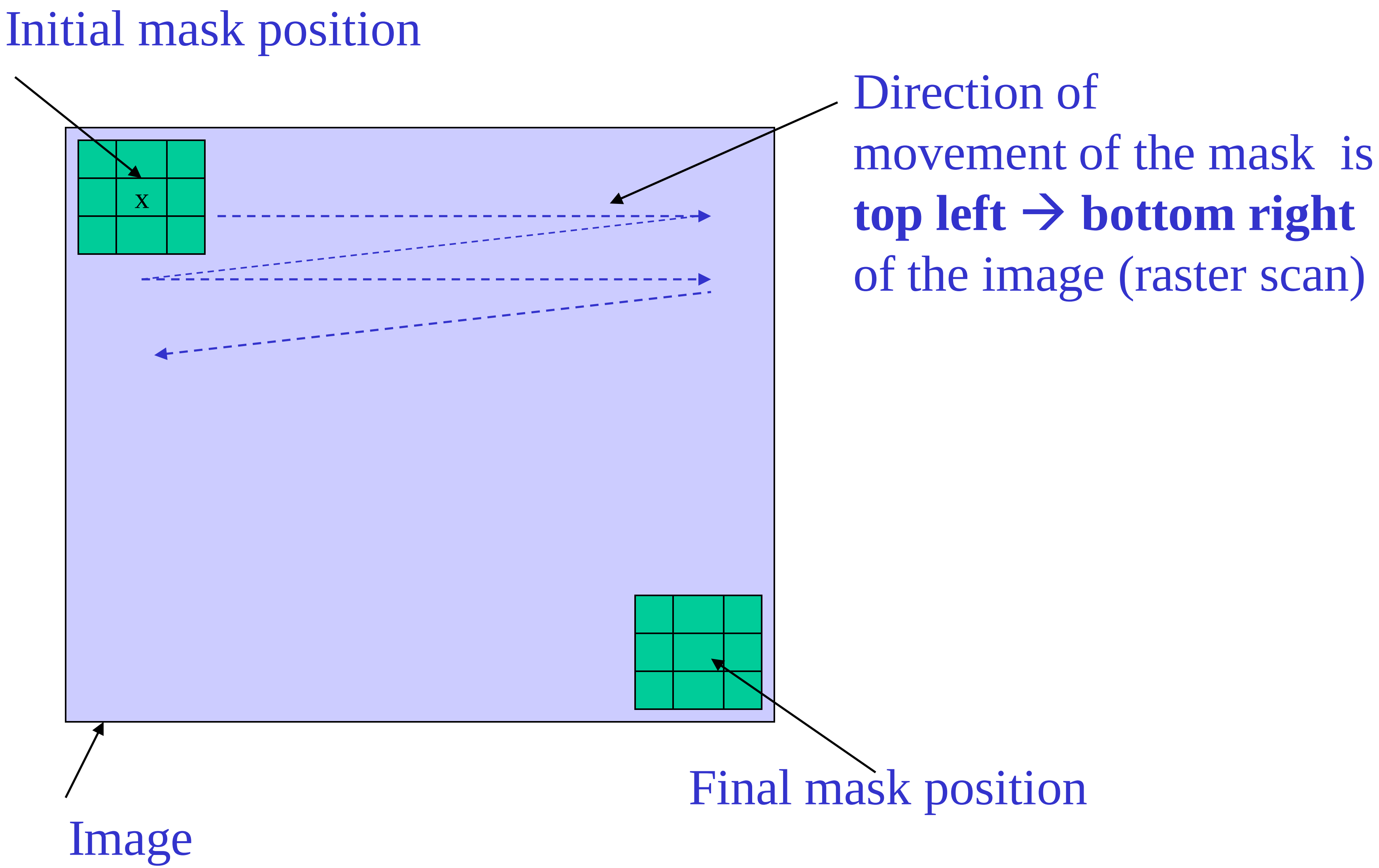
$\Pi[m,n] = 1$; $|m|$ or $|n| \leq 1$

$= 0$ otherwise

Neighbourhood (spatial) operations

- Output pixel value at p in the output image is determined by pixels in N_p in the input image
- This is a filtering operation
- Filter kernel is represented as a mask or window
- The operation performed can be linear or nonlinear

Filtering / Masking operation



Non-linear operations

Notations: input image (x), output image (y)

- **Ranking**
  - $y[m,n] = max \; .. \; min \; \ldots \; median \; \{x[m,n];\ [m,n] \in N[m,n]\}$

Neigbourhood of (m,n)

- **Extrema** – special case in ranking
  - $y[m,n] = \; min \; \{x[m,n];[m,n] \in N[m,n]\}$

  or $\qquad = max \; \{x[m,n];[m,n] \in N[m,n]\}$

  help find the local <u>max</u> or <u>min</u>

Linear mask operations

y: output image *x*: input image *h*: filter mask

y(m,n) = h(m,n) x(m,n) ; h* convolved with the input *x*

$$y[m,n] = \sum_i \sum_j x[i,j]h[m-i,n-j]$$

- output pixel is a weighted combination of the neighbouring pixels.
  - Weight is determined by the filter kernel *h* i.e. mask coefficients
- mask size is always finite ➔ FIR filtering

Linear filtering

What is linear filtering?

Filtered output (pixel) is a linear combination of input pixels

Common examples:

- Averaging (straight, weighted) or low pass filtering
- Finite difference or high pass filtering
- High boost filtering
- Bandpass filtering

Where is linear filtering used?

Common applications:

- Enhancement
  - Deblur, sharpen
- Denoising
  - By averaging
- Feature extraction
  - Detect edges

Spatial averaging

- Output pixel value at $[i,j]$ = average of all pixels in $N\,[i,j]$
- Mask weights = $h\,[i,j]$
- Mask is chosen to be odd sized 3x3, 7x7,…etc.
  - Why?

- Visual effect: *smoothing*
  - <u>Degree</u> of smoothing $\propto$ mask size
  - <u>Type</u> of smoothing determined by mask shape, and weights. $h(i,j)$

h[-1,-1]	**h[-1,0]**	**h[-1,1]**
h[0,-1]	**h[0,0]**	**h[0,1]**
h[1,-1]	**h[1,0]**	**h[1,1]**

m x m mask representing $N(i,j)$

Straight averaging

- Equal mask weights (Ideal low pass filter)
  - For $m \times m$ sized kernel, $h[i,j] = 1/m^2 \;\; \forall \; i,j$
  - Equivalent to a running average filter in 1D

- Equivalent to convolution with $\Pi[m,n]$

  2D *rect* function or a *box* function
  - Can lead to problems along sharp edges

12	**10**	**17**
17	**12**	**17**
35	**16**	**184**

$x(m,n) = 12$

$\Rightarrow y(m,n) = 35$

Smoothing using averaging

Before

After

After repeated averaging

Check: Is repeated averaging by a filter the same as filtering with a larger kernel?

Weighted averaging

Gaussian smoothing : Mask weights are samples of a Gaussian function (distant neighbours have less weights)

$$G(x,y) = \frac{1}{2\pi\sigma^2} \exp\{-(x^2 + y^2)/2\sigma^2\}$$

- Used to obtain controlled smoothing
  - Small σ means less smoothing and retaining fine detail

- Mask size increases rapidly with σ

N_x

12	**10**	**17**
17	**12**	**17**
35	**16**	**184**

$x(m,n) = 12$

h $\sigma = 0.391$ pixels

1	**4**	**1**
4	**12**	**4**
1	**4**	**1**

$y(m,n) = ?$

Controlled smoothing – 2 other alternatives

There are other ways to have controlled smoothing

1. Apply averaging selectively

Example

- include a neighbouring pixel in the averaging only if it is in a specified range
- output pixel is valid only if it is in a specified range,
- etc.

Controlled smoothing...contd.

2. Use a rotating mask and detect homogeneous region around a pixel of interest and average this region only

9 rotated versions of a 3x3 mask

x : pixel of interest

Non-local Mean filter

- Non-local Means [Buades et al. 2005] – proposed for denoising

$$g_{NL}[r] = \sum_{q \in I} w[r,q] I[r]$$

$$0 \leq w \leq 1; \sum w = 1$$

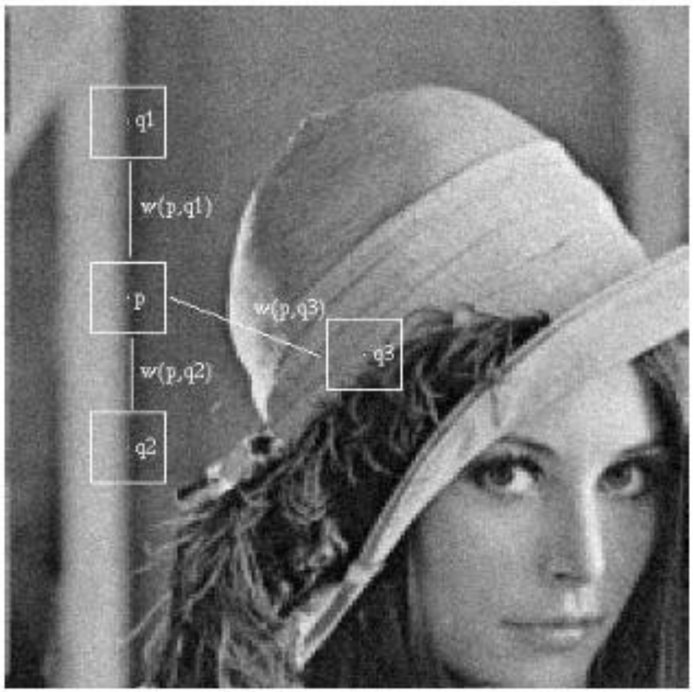
r, *q* are pixel locations;

I[*r*] is the pixel value at *r*

- *w* is the weight based on <u>similarity</u> between pixel values in the neighbourhoods of *r* and *q*.

Basic strategy: similar neighbourhoods should lead to large weights and vice versa

NLM filter contd.



From Buades et al. 2005

Popular choice for w is a Gaussian weighted (Euclidean) distance between r,q

$$w(r,q) = \frac{1}{K(r)} e^{-\frac{N\left(\left\| I(\Omega_r) - I(\Omega_q) \right\|^2, a\right)}{h^2}}$$

$$K(r) = \sum_q e^{-\frac{N\left(\left\| I(\Omega_r) - I(\Omega_q) \right\|^2, a\right)}{h^2}}$$

$N(\mu,\sigma)$ denotes Normal distribution

Ω_x denotes the neighbourhood about x

Sample result from Buades *et al*

Noisy input

Denoised results

Gaussian averaging

NLM

Other types of spatial domain filtering

- Other common linear filtering operations
  - High-pass
  - Band-pass

What are the mask coefficients *h[m,n]*?

Clue: Any spectrum can be divided into a lowpass band + highpass band

➔ An allpass filter can be thought of as a sum of LPF and HPF

Check: What is the *filter kernel* h_{AP} for an allpass filter?

HPF Filter derivation

- Use LPF as the prototype

HPF: $h_{HP}[m,n] = \delta[m,n] - h_{LP}[m,n]$

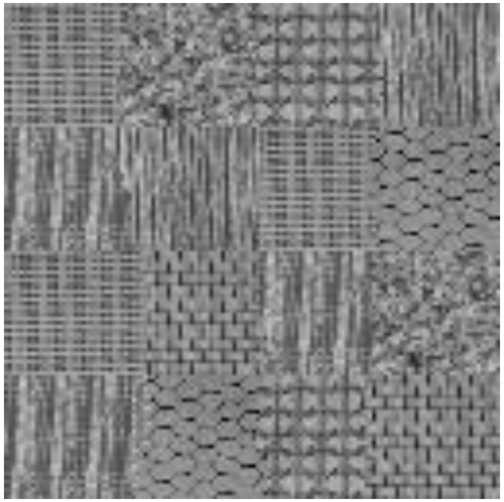
$$y_{HP}[m,n] = x[m,n] * h_{HP}[m,n]$$
$$= x[m,n] * [\delta[m,n] - h_{LP}[m,n]]$$

Using distributive property of convolution, rewrite as

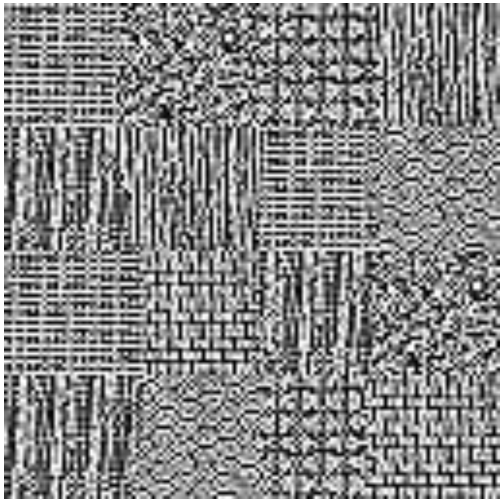
$$\boxed{y_{HP}[m,n] = x[m,n] - y_{LP}[m,n]}$$

where $y_{LP}[m,n] = x[m,n] * h_{LP}[m,n]$

Examples



After HPF ➔



Quilt

BPF filter derivation

BPF: $h_{BP}[m,n] = h_{LP2}[m,n] - h_{LP1}[m,n]$

$$y_{BP}[m,n] = x[m,n] * (h_{LP2}[m,n] - h_{LP1}[m,n])$$

$$\boxed{y_{BP}[m,n] = y_{LP2}[m,n] - y_{LP1}[m,n]}$$

Filter masks

1/9	**1/9**	**1/9**
1/9	**1/9**	**1/9**
1/9	**1/9**	**1/9**

LPF mask

-1/9	**-1/9**	**-1/9**
-1/9	**8/9**	**-1/9**
-1/9	**-1/9**	**-1/9**

HPF mask

What is the sum of mask coefficients?

What is their significance?

Filter applications

- LPF
  - (impulse) noise smoothing,
- HPF
  - extract edges/details
- BPF
  - edge enhancement, sharpen noisy images

Other filters

- LPF smoothes and HPF retains only details in images

What if we want to only sharpen an image?

- Unsharp masking or high boost filtering
  - Derived from printing industry practice
  - Performs high frequency emphasis

$$y[\mathrm{m,n}] = \alpha\, x[\mathrm{m,n}] - x_{LP}[\mathrm{m,n}] = (\alpha - 1)\, x + x_{HP}\ ;\ \alpha \geq 1$$

$\alpha = 1$ results in HPF (y has only details)

$\alpha > 1$ indirectly controls the degree of sharpening

Filter masks ……contd

w = ?

-1/9	**-1/9**	**-1/9**
-1/9	**w**	**-1/9**
-1/9	**-1/9**	**-1/9**

High Boost Filter mask

Examples – HPF vs HBF

Input : Cheetah

3x3

5x5

7x7

15x15

HPF results for different mask sizes

Input : Cheetah

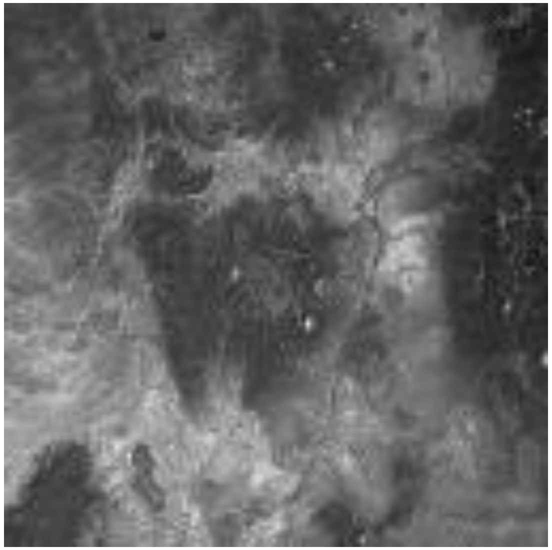
High Boost filter results:
Effect of α and mask size

3x3 mask with α = 1 (HPF)

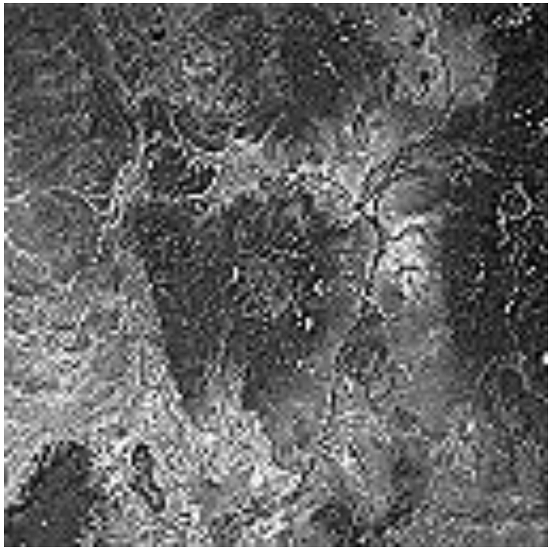
3x3 mask with α = 1.5

3x3 mask with α = 2

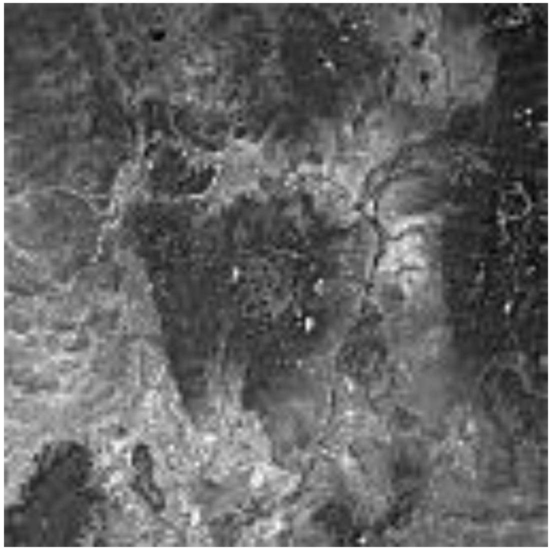
7x7 mask with α = 1.3



Santa Fe -
Remote sensing



Santa Fe - HPF



Santa Fe - HBF

Examples

Face

Face - HPF

Face - HBF

Non-linear filtering – some applications

Median filtering

- Output pixel = median value of all pixels within the $N(i,j)$
- *Effects*
  - Excellent in removing salt and pepper noise
  - Does not shift boundaries
  - Does not affect brightness change across steps
- Computationally intensive as processing an image with M pixels requires M sorting operations

Median filtering - example

12	**10**	**17**
17	**12**	**17**
35	**16**	**184**

Pixels within a 3x3 window

Ranked list: 184 35 17 17 17 16 12 12 10

output pixel value = Median = 17

Comparison of different filters

Input image patch

12	**10**	**17**
17	**12**	**17**
35	**16**	**184**

Output pixel value

Mean filter: 35 **Gaussian weighted mean** filter : 28
Median filter: 17

Mean after ignoring the 'noisy pixel' = 17

Application 1: Reducing noise

Noisy original

After averaging

After median filtering

Application 2: Finding extrema

Task: find extrema points in an image.

Solution 1 **Max (Min) filter**

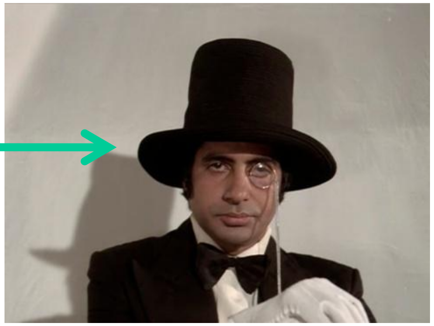
- Output pixel at $p = \max\{I_{in}[m,n]; [m,n] \in N_p\}$
  - ➢ Simple ranking
  - ➢ Does not discriminate between extrema of different kinds

Solution 2 **Top (bottom) hat filter**

- Output pixel at $p = I_p$ if $I_p >$ pixel level in its background
  - ➢ *Context-aware* max finder

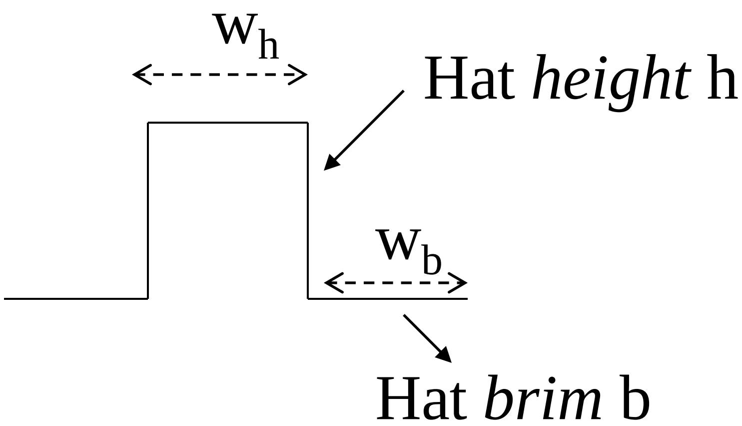
- Uses 2 neighbourhoods (1 each for estimating background and foreground)

Top hat filter



1-D case: finds <u>local</u> peaks/maxima

- Low or broad peaks are skipped



Top hat function

For the local peak of interest the control parameters are:

w_h: peak *width*

w_h: specifies the *extent* of the context

h: peak *height*

Top hat filter

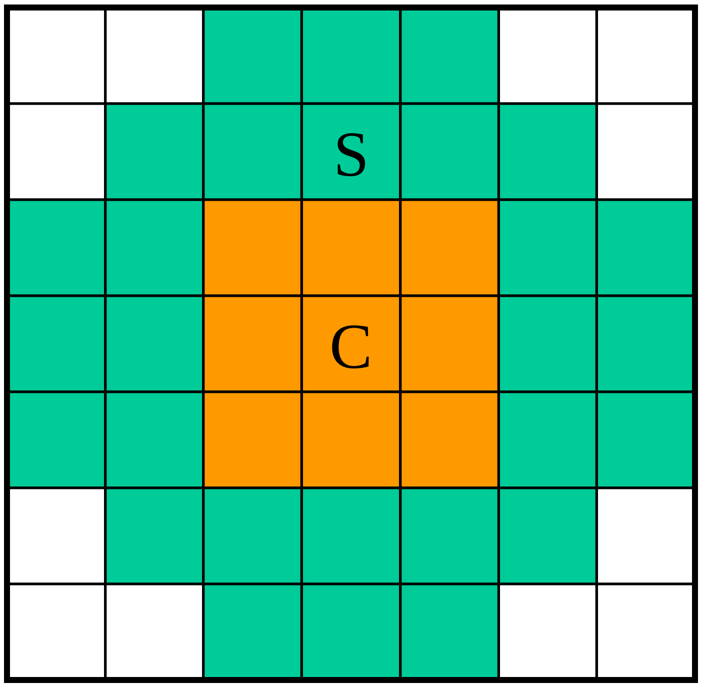
- "Interesting" point finder
  - w_h specifies the spread of the peak of interest
- Many versions exist

Example: to find the bright points in an image

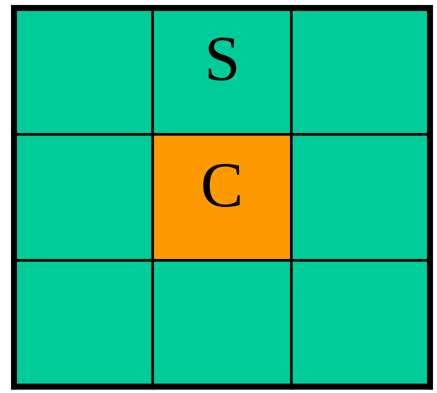
A method:

1. Define a central and a surround region about a pixel (with a window)
2. Rank the pixels in the two regions and find the max values
3. Find $g_{cmax} - g_{smax} = g_{diff}$
4. If g_{diff} is above a threshold **retain** centre pixel.
   Else set it to 0

Neighbourhood patterns



Octagonal



Square

Top hat filtering

Effects:

- ➢ Only " interesting pixels" are retained

  - is a suppression operator

- ➢ Centre pixel is retained even if it is not the brightest pixel (as the criterion is $g_{diff} > t$)

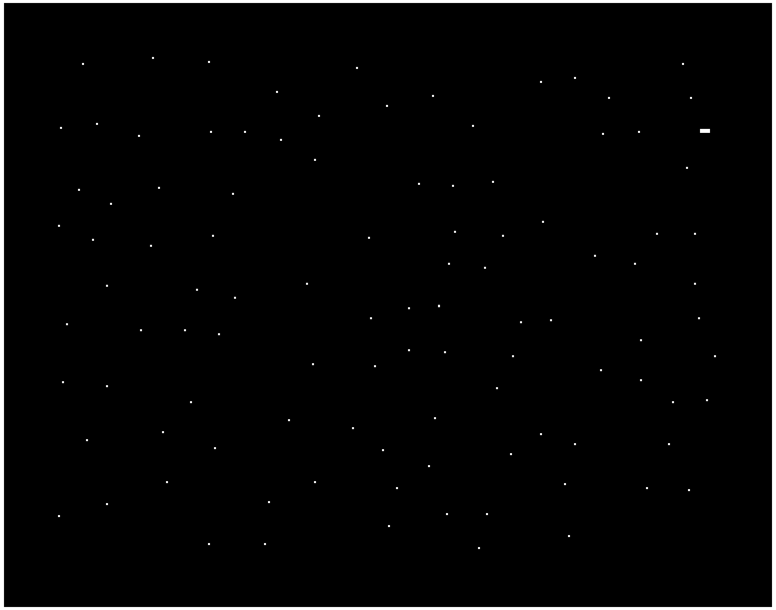
  - produces small "halo" around bright points

- Matlab also has a *Bottom hat* function

Stars in Night Sky

Top Hat Filter Result

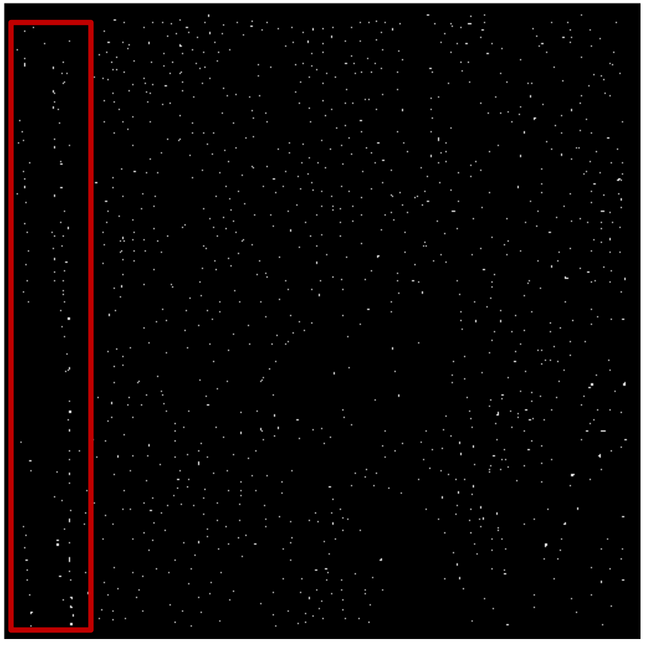
Max Filter Result



Lena Image

Top Hat Filter Result

Max Filter Result



Neighbourhood processing –

implementation issues

Issues

Handling border pixels: what are their neighbours?

Option 1

- Don’t compute the output for such pixels
  - Output image is smaller than input image

Option 2

- Extend the image size by zero padding before processing

Option 3

- Extend the image by one of many ways (replicate, wrapped around, etc.) before processing

Implementation issues.. contd.

Computational load – load increases with mask size as well as type of filtering

- Pixel addressing load as well as processing load

Solution 1

- Implement in frequency domain
  - Exploit the property that convolution in spatial domain is equivalent to multiplication in frequency domain

Check: what is the max kernel size above which it is not efficient to implement filtering in spatial domain?

Solution 2

- efficient hardware implementation

When to use what type of proc.?

Point

- When type of image is *fixed*
- Intensity characteristics of object/region of interest is *known*

Ex. Medical images of specific organs

- One time parameter tuning

Neighbourhood

- More *generic* and *customizable*
- *Demands skill* in modeling the neighborhood interaction

Global

- When type of image is *unknown*
- Objects/regions of interest are *unknown* so intensity characteristics
- i.e. Natural images/indoor/outdoor