BSB663
Image Processing

Pinar Duygulu

Slides are adapted from
Gonzales & Woods,
Emmanuel Agu
Suleyman Tosun

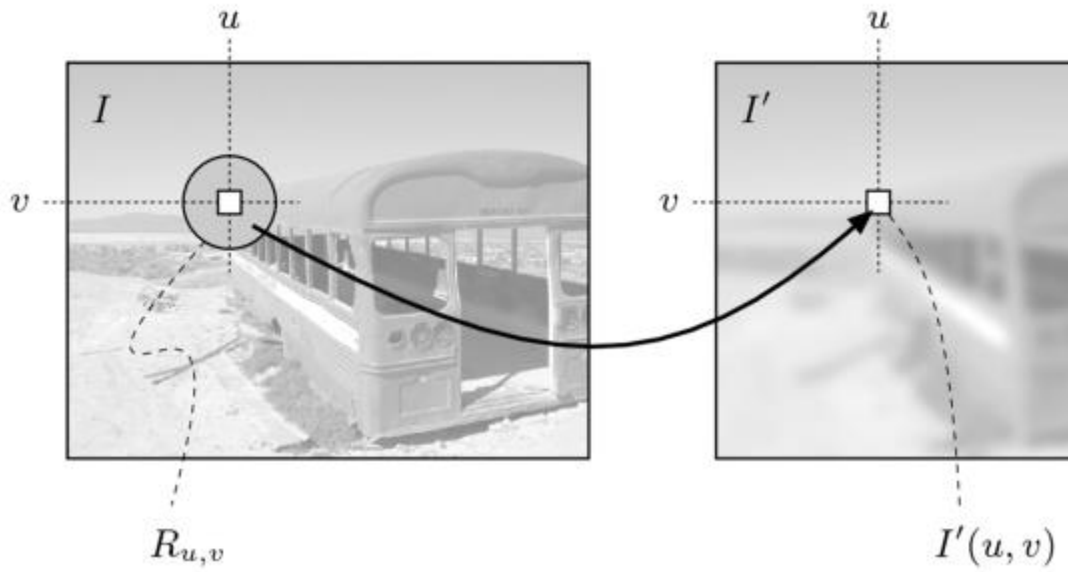
Filters

- Capabilities of point operations are limited
- **Filters:** combine **pixel's value** + **values of neighbors**
- **E.g blurring:** Compute average intensity of block of pixels

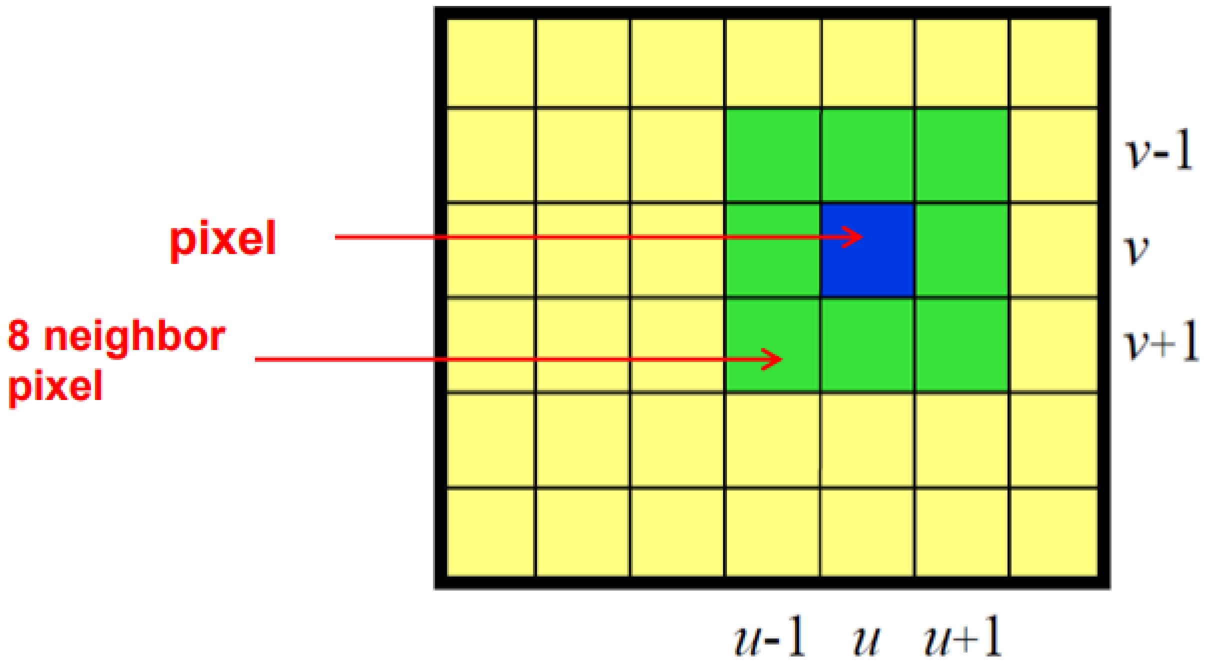
- Combining multiple pixels needed for certain operations:
  - Blurring, Smoothing
  - Sharpening

Spatial Filter

- An image operation that combines each pixel's intensity $I(u, v)$ with that of neighboring pixels
- **E.g:** average/weighted average of group of pixels



Average (Mean) of a 3x3 neighborhood

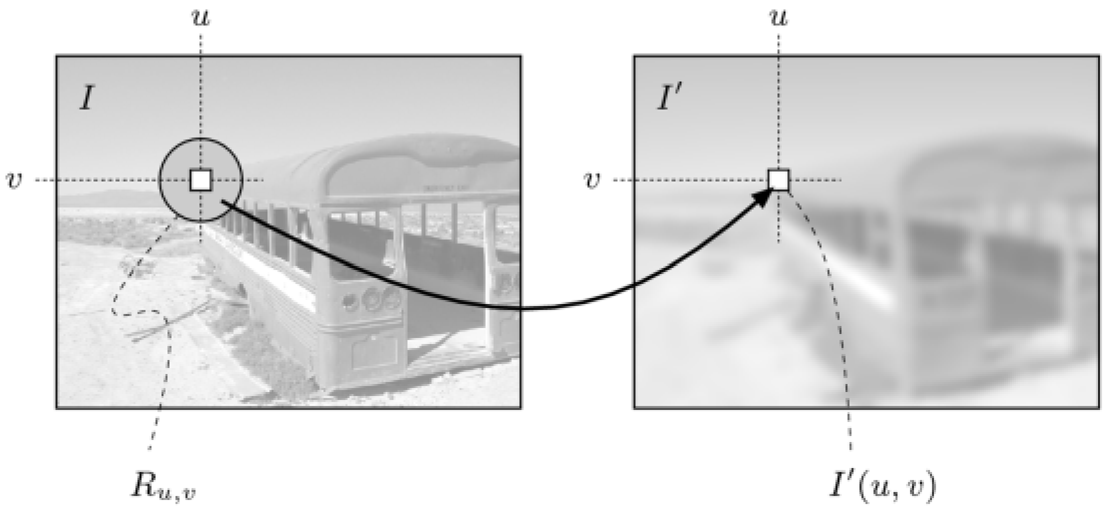


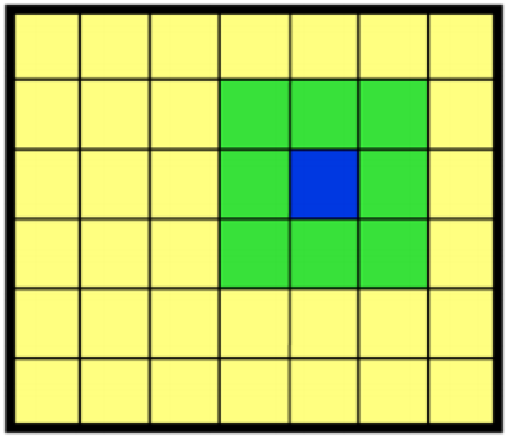
Blurring: Replace each pixel with AVERAGE Intensity of pixel + neighbors

Smoothing an image by averaging

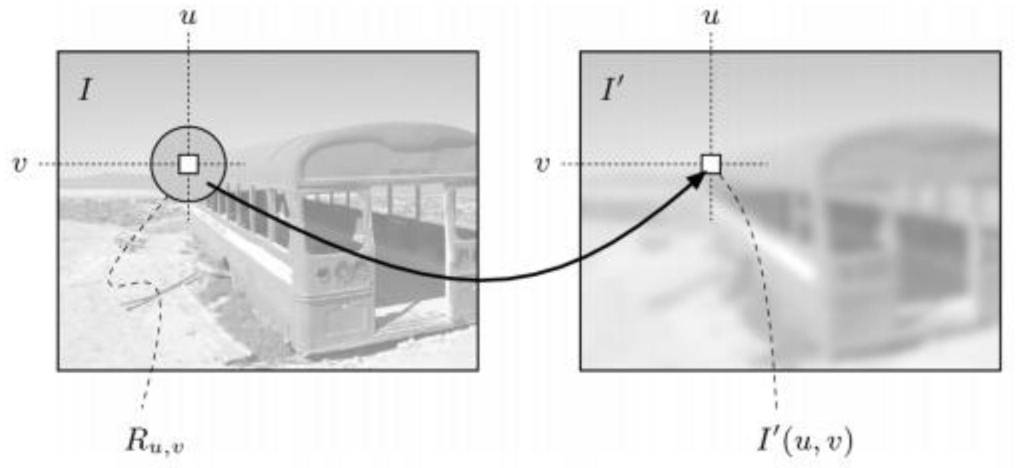
- Replace each pixel by average of pixel + neighbors
- For 3x3 neighborhood:

$$I'(u,v) \leftarrow \frac{p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8}{9}$$



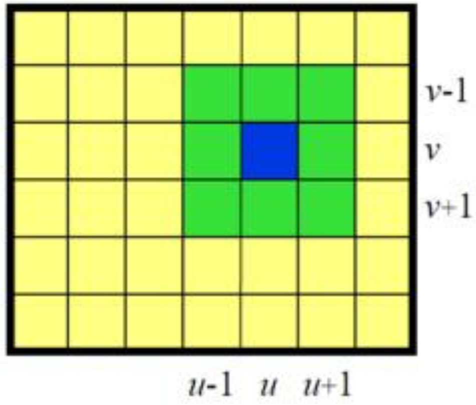


Smoothing an image by averaging



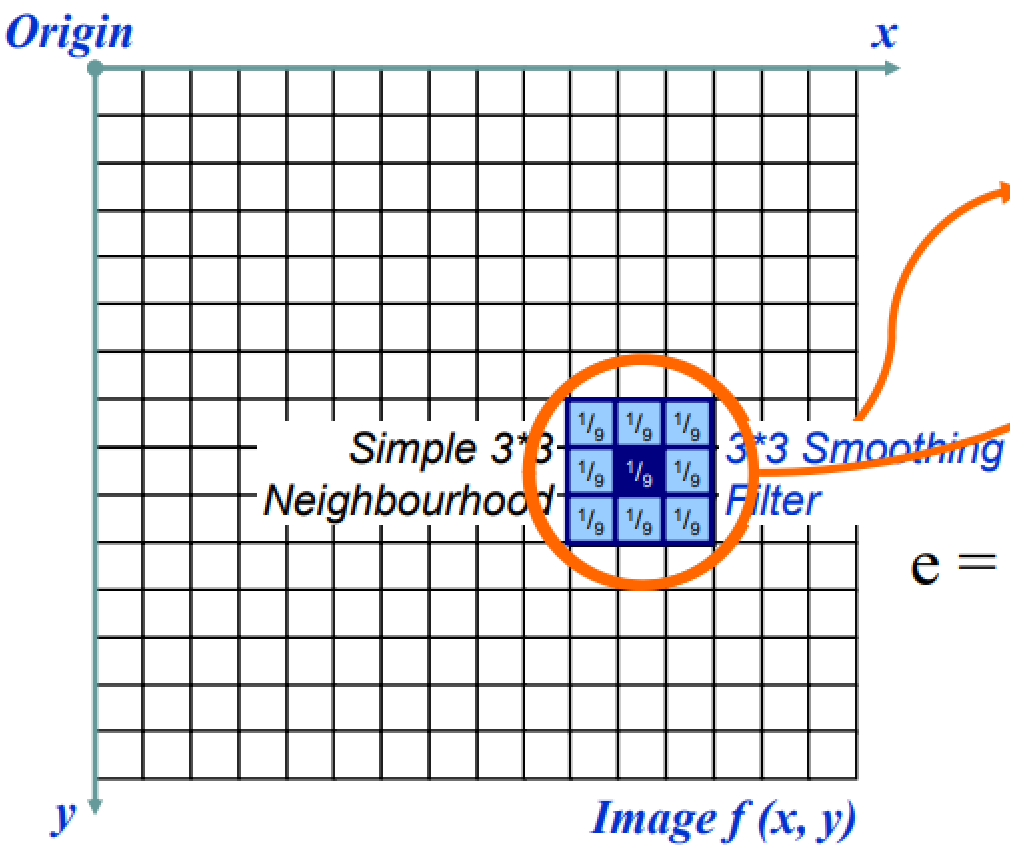
$$I'(u,v) \leftarrow \frac{p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8}{9}$$

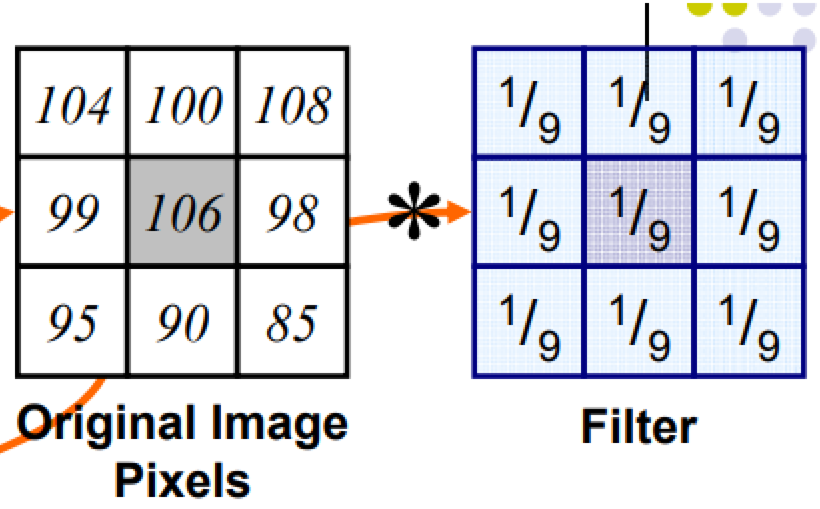
$$\begin{aligned} I'(u,v) \leftarrow \tfrac{1}{9} \cdot [\, & I(u-1,v-1) + I(u,v-1) + I(u+1,v-1) + \\ & I(u-1,v) + I(u,v) + I(u+1,v) + \\ & I(u-1,v+1) + I(u,v+1) + I(u+1,v+1) \,] \end{aligned}$$



$$I'(u,v) \leftarrow \frac{1}{9} \cdot \sum_{j=-1}^{1} \sum_{i=-1}^{1} I(u+i, v+j)$$

Smoothing as filtering

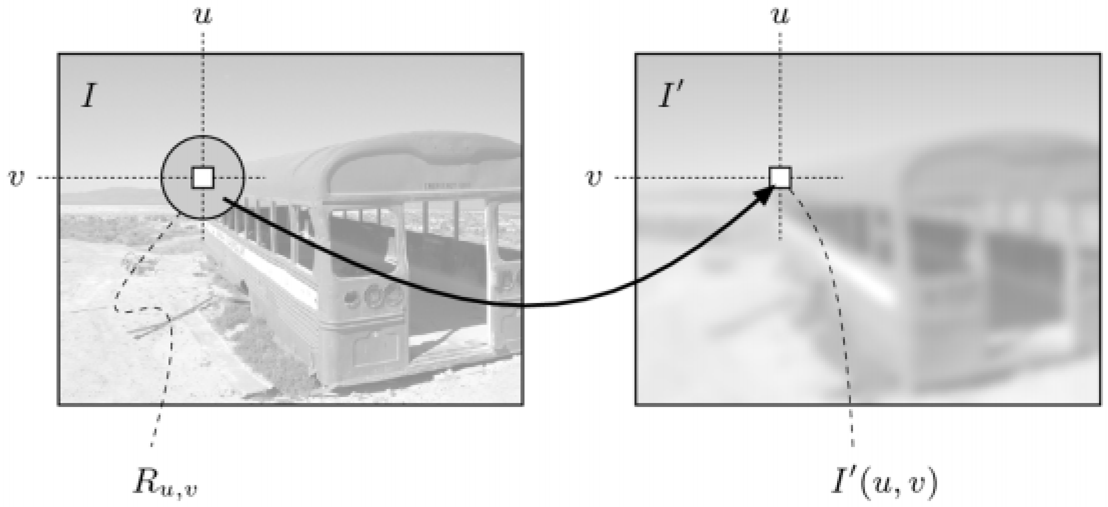


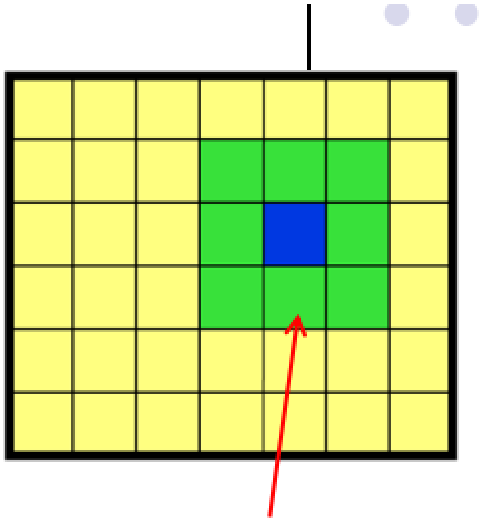


$$e = \tfrac{1}{9}*106 + \tfrac{1}{9}*104 + \tfrac{1}{9}*100 + \tfrac{1}{9}*108 + \tfrac{1}{9}*99 + \tfrac{1}{9}*98 + \tfrac{1}{9}*95 + \tfrac{1}{9}*90 + \tfrac{1}{9}*85 = 98.3333$$

The above is repeated for every pixel in the original image to generate the smoothed image

Smoothing as filtering





Previous example:
Filter size: 3x3

- Many possible filter parameters (size, weights, function, etc)
- **Filter size (size of neighborhood):** 3x3, 5x5, 7x7, ….,21x21,..
- **Filter shape:** not necessarily square. Can be rectangle, circle, etc
- **Filter weights:** May apply unequal weighting to different pixels
- **Filters function:** can be linear (a weighted summation) or nonlinear

Filter

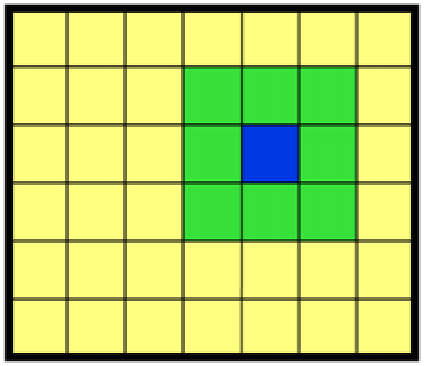
$$I'(u,v) \leftarrow \frac{p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8}{9}$$

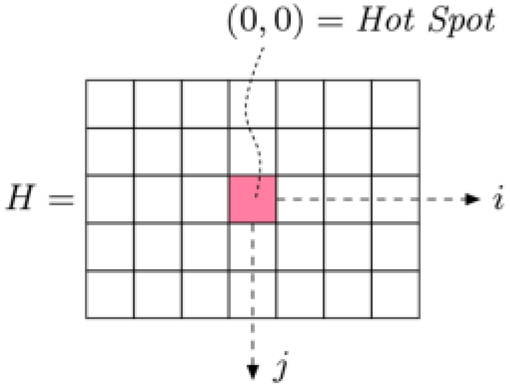
$$\begin{aligned} I'(u,v) \leftarrow \tfrac{1}{9} \cdot [\, & I(u-1, v-1) + I(u, v-1) + I(u+1, v-1) + \\ & I(u-1, v) + I(u, v) + I(u+1, v) + \\ & I(u-1, v+1) + I(u, v+1) + I(u+1, v+1) \,] \end{aligned}$$

$$H(i,j) = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

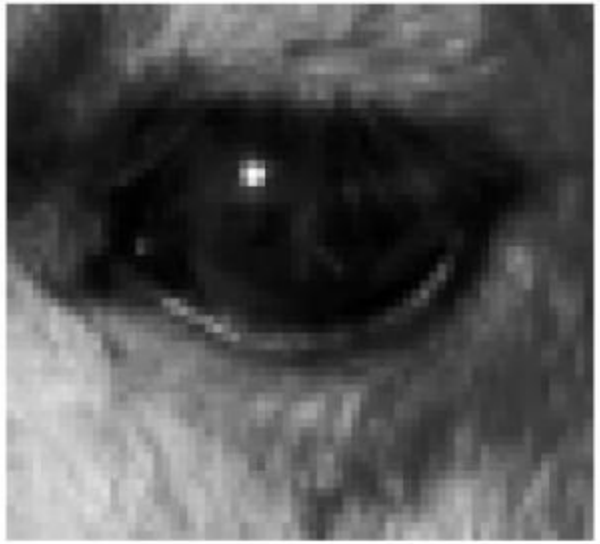
Filter operation can be expressed as a matrix
Example: averaging filter

Filter matrix also called filter mask ***H(i,j)***

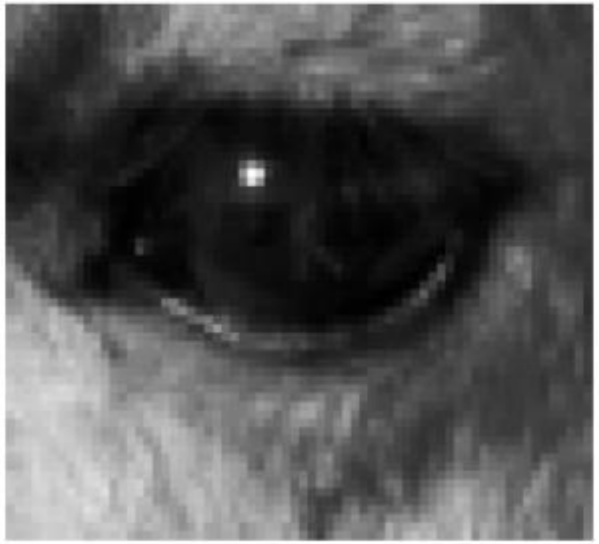




What does this filter do?

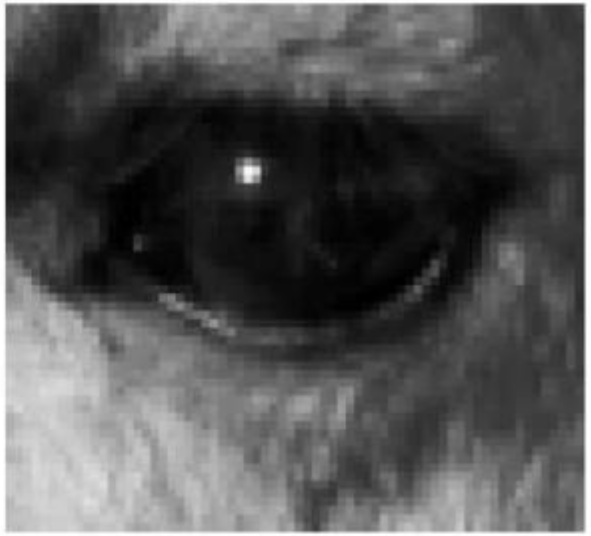


0	0	0
0	1	0
0	0	0

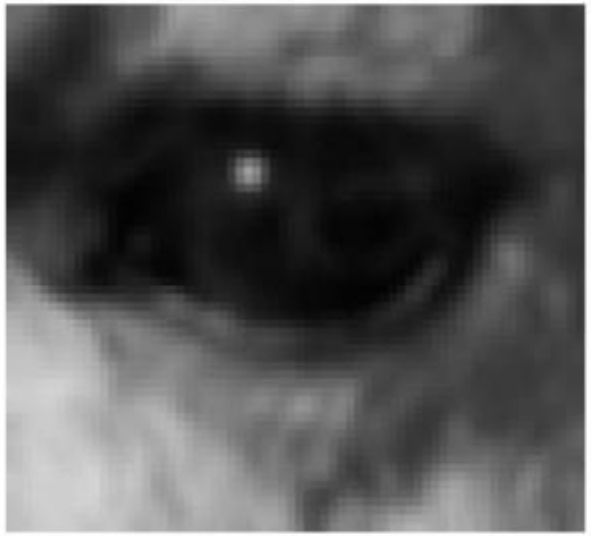


Identity function (leaves image alone)

What does this filter do?



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



Mean (averages neighborhood)

Mean filters: effect of filter size

Original

7×7

15×15

41×41

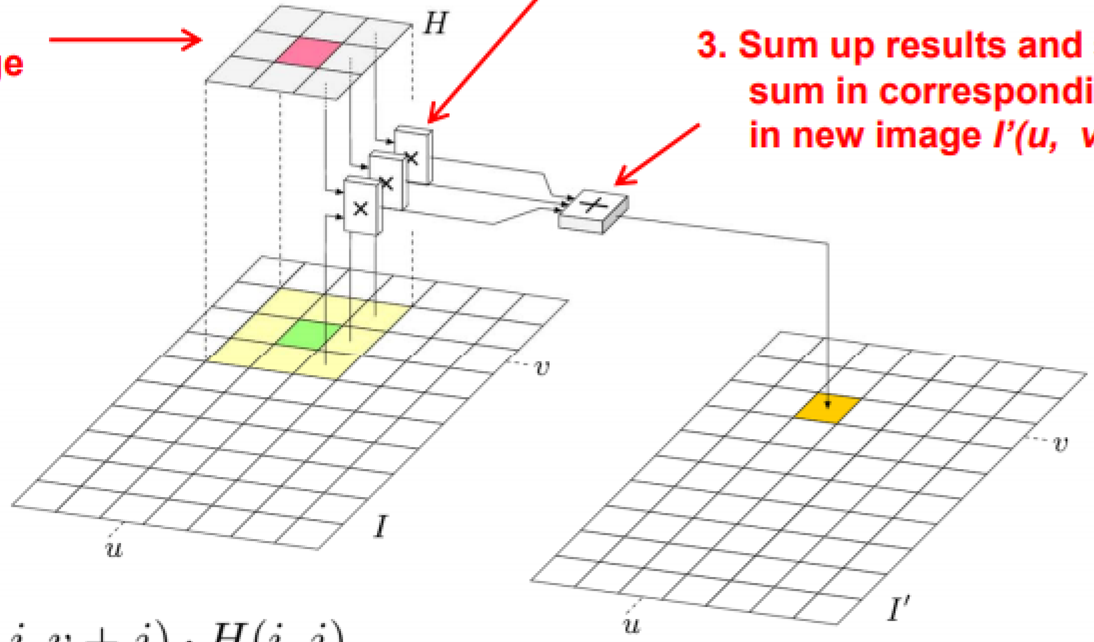
Convolution

For each image position *I(u,v)*:

1. Move filter matrix *H* over image such that *H(0,0)* coincides with current image position *(u,v)*

2. Multiply all filter coefficients *H(i,j)* with corresponding pixel *I(u + i, v + j)*

3. Sum up results and store sum in corresponding position in new image *I'(u, v)*



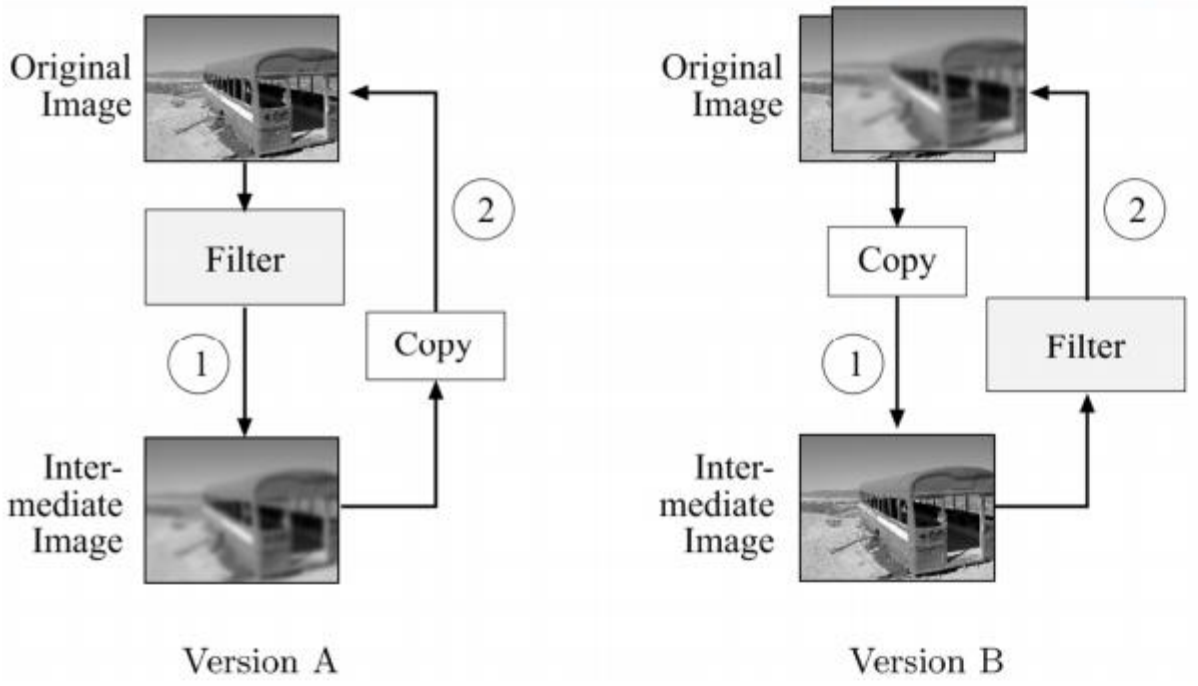
Stated formally:

$$I'(u,v) \leftarrow \sum_{(i,j)\in R_H} I(u+i, v+j) \cdot H(i,j)$$

R_H is set of all pixels Covered by filter.
For 3x3 filter, this is:

$$I'(u,v) \leftarrow \sum_{i=-1}^{i=1} \sum_{j=-1}^{j=1} I(u+i, v+j) \cdot H(i,j)$$

- Filter matrix ***H*** moves over each pixel in original image ***I*** to compute corresponding pixel in new image ***I'***
- Cannot overwrite new pixel value in original image ***I*** Why?



Store results ***I'*** in intermediate image, then copy back to replace ***I***

Copy original image ***I*** to intermediate image, use it as source, then store results ***I'*** to replace original image

Weighted smoothing filters

- More effective smoothing filters can be generated by allowing different pixels in the neighbourhood different weights in the averaging function
  - Pixels closer to central pixel more important
  - Often referred to as a *weighted averaging*

$^1/_{16}$	$^2/_{16}$	$^1/_{16}$
$^2/_{16}$	$^4/_{16}$	$^2/_{16}$
$^1/_{16}$	$^2/_{16}$	$^1/_{16}$

Weighted averaging filter

- Instead of floating point coeffients, more efficient, simpler to use:

scaling factor + integer coefficients

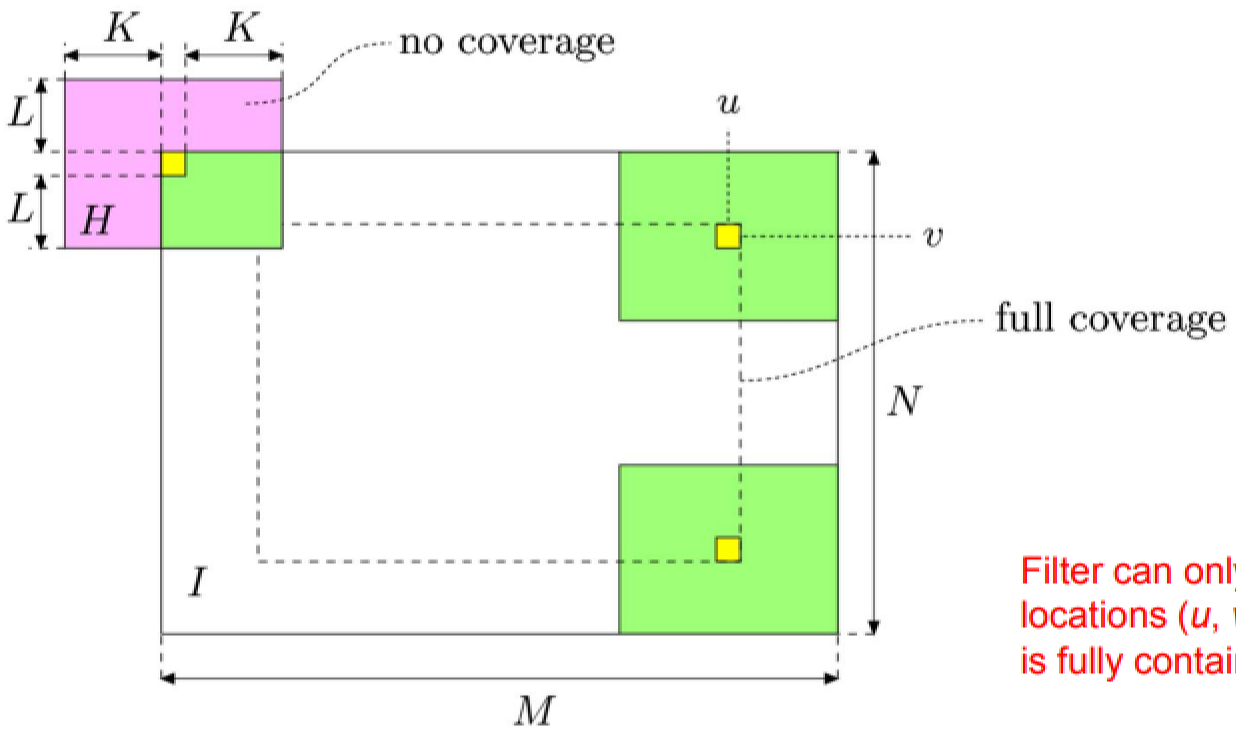
$$H(i,j) = \begin{bmatrix} 0.075 & 0.125 & 0.075 \\ 0.125 & \underline{0.200} & 0.125 \\ 0.075 & 0.125 & 0.075 \end{bmatrix} = \frac{1}{40} \begin{bmatrix} 3 & 5 & 3 \\ 5 & \underline{8} & 5 \\ 3 & 5 & 3 \end{bmatrix}$$

$$H(i,j) = s \cdot H'(i,j)$$

Computation range

- For a filter of size (2K+1) x (2L+1), if image size is MxN, filter is computed over the range:

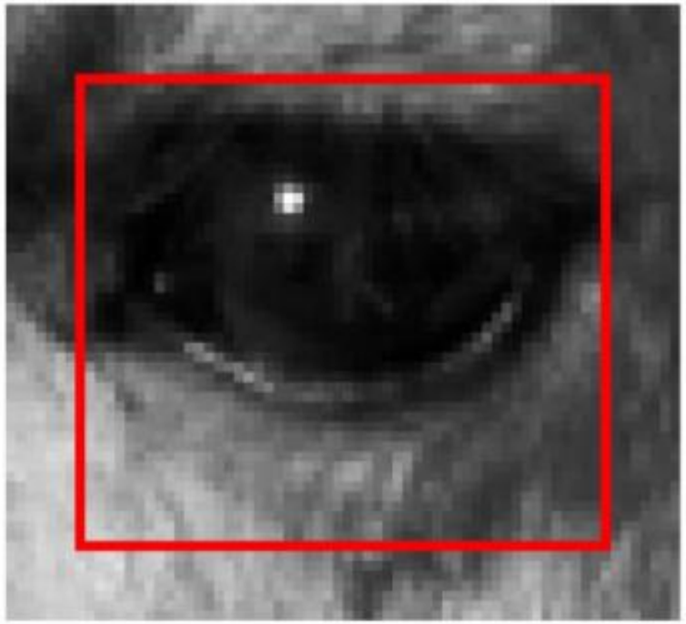
$$K \leq u' \leq (M-K-1) \qquad \text{and} \qquad L \leq v' \leq (N-L-1)$$



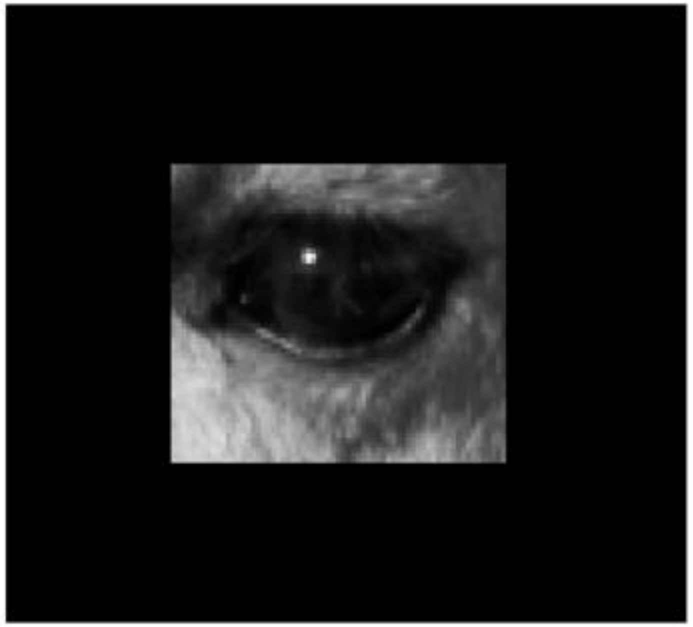
Filter can only be applied at image locations (*u*, *v*) where filter matrix *H* is fully contained in the image

What to do at image boundaries?

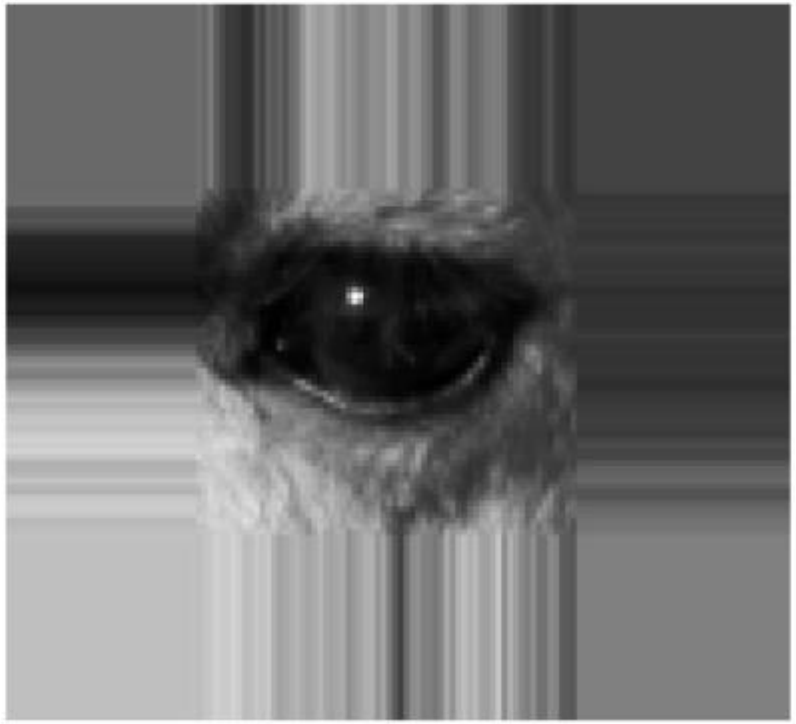
a) Crop



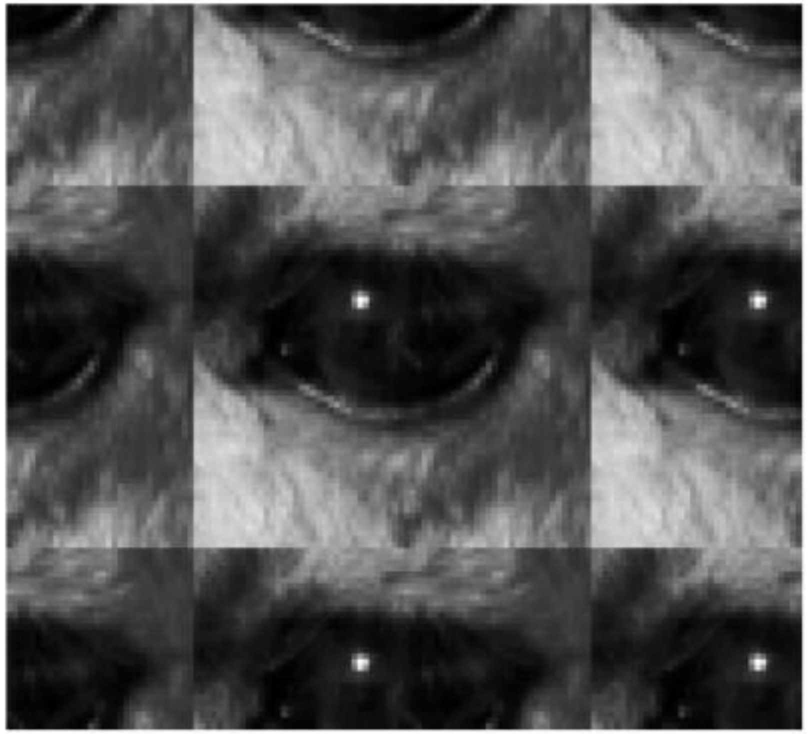
a) Crop
b) Pad



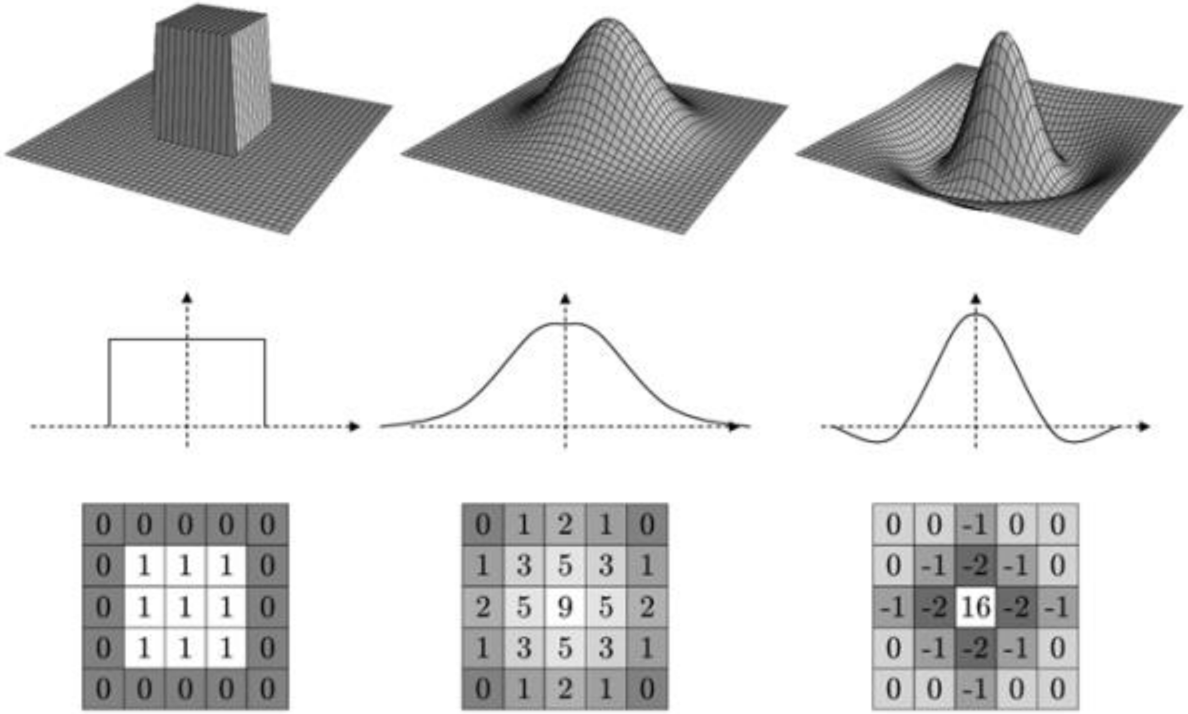
a) Crop
b) Pad
c) Extend



a) Crop
b) Pad
c) Extend
d) Wrap



- 2 main classes of linear filters:
  - **Smoothing:** +ve coeffients (weighted average). E.g box, gaussian
  - **Difference** filters: +ve and –ve weights. E.g. Laplacian

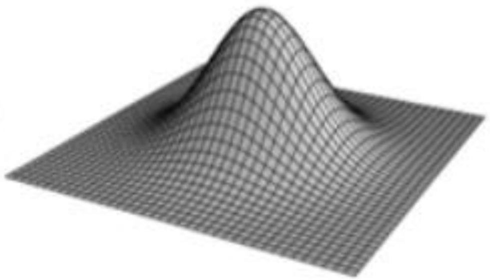


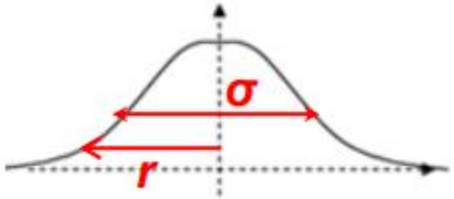
(a) **Box** (b) **Gaussian** (c) **Laplacian**

Gaussian Filter

$$G_\sigma(r) = e^{-\frac{r^2}{2\sigma^2}} \quad \text{or} \quad G_\sigma(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- where
  - ***σ*** is width (standard deviation)
  - ***r*** is distance from center



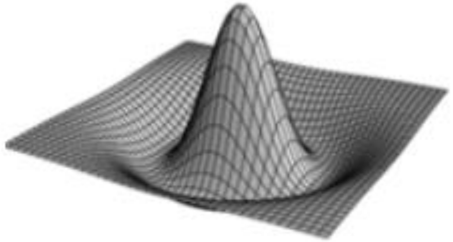


0	1	2	1	0
1	3	5	3	1
2	5	9	5	2
1	3	5	3	1
0	1	2	1	0

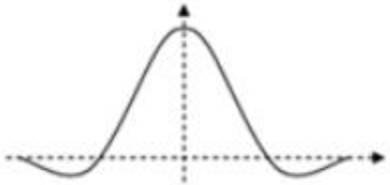
Gaussian filter

Difference Filters

- **Coefficients:** some +ve, some negative
- Example: Laplacian filter
- Computation is difference



$$\sum(+ve\ coeffients) - \sum(-ve\ coeffients)$$



$$I'(u,v) = \sum_{(i,j)\in R_H^+} I(u+i, v+j) \cdot |H(i,j)| - \sum_{(i,j)\in R_H^-} I(u+i, v+j) \cdot |H(i,j)|$$

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

Laplacian filter

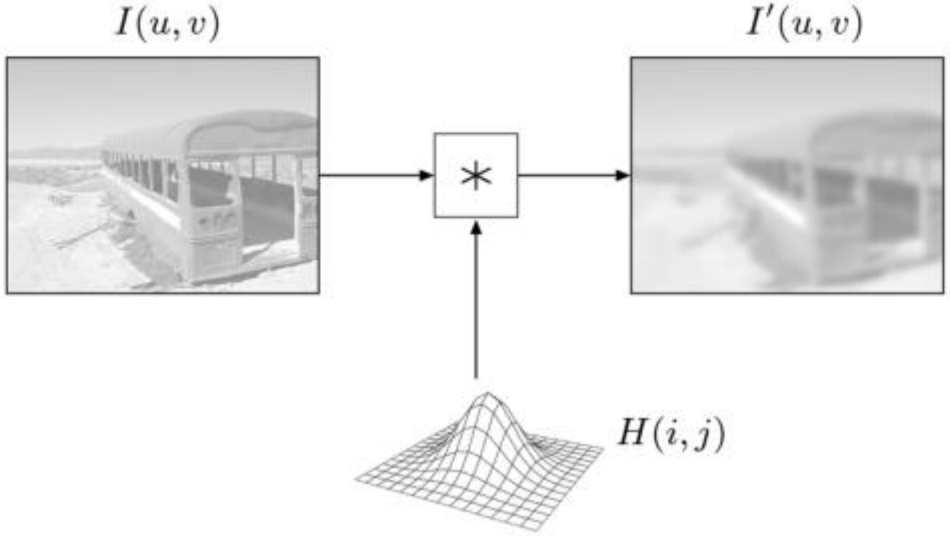
Mathematical Properties of convolution

- Applying a filter as described called ***linear convolution***
- For discrete 2D signal, convolution defined as:

$$I'(u,v) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(u-i, v-j) \cdot H(i,j)$$

Formal definition:
Sum to ± ∞

$$I' = I * H$$



- Commutativity

Same result if we convolve image with filter or vice versa

$$I * H = H * I$$

- Linearity

If image multiplied by scalar Result multiplied by same scalar

$$(s \cdot I) * H = I * (s \cdot H) = s \cdot (I * H)$$

$$(I_1 + I_2) * H = (I_1 * H) + (I_2 * H)$$

If 2 images added and convolve result with a kernel *H*, Same result if each image is convolved individually + added

(notice) $(b + I) * H \neq b + (I * H)$

- Associativity

$$A * (B * C) = (A * B) * C$$

Order of filter application irrelevant Any order, same result

- Separability

$$H = H_1 * H_2 * \ldots * H_n$$

$$\begin{aligned} I * H &= I * (H_1 * H_2 * \ldots * H_n) \\ &= (\ldots((I * H_1) * H_2) * \ldots * H_n) \end{aligned}$$

- If a kernel H can be separated into multiple smaller kernels

Applying smaller kernels H_1 H_2 … H_N H one by one computationally cheaper than apply 1 large kernel H

$$H = H_1 * H_2 * \ldots * H_n$$

Computationally More expensive

Computationally Cheaper

Separability

- Sometimes we can separate a kernel into **"vertical"** and **"horizontal"** components
- Consider the kernels

$$H_x = [1\ 1\ 1\ 1\ 1], \quad \text{and} \quad H_y = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Then

$$H = H_x * H_y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

- What is the number of operations for 3 x 5 kernel *H*

Ans: 15*wh*

$$H = H_x * H_y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

- What is the number of operations for H_x followed by H_y?

Ans: 3*wh* + 5*wh* = 8*wh*

$$H_x = [1\ 1\ 1\ 1\ 1], \quad \text{and} \quad H_y = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

- What is the number of operations for 3 x 5 kernel H

Ans: 15*wh*

- What is the number of operations for H_x followed by H_y?

Ans: 3*wh* + 5*wh* = 8*wh*

- What about M x M kernel?

$O(M^2)$ – no separability (M^2wh operations, grows quadratically!)

$O(M^2)$ – with separability ($2Mwh$ operations, grows linearly!)

Gaussian Kernel

- 1D

$$g_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

- 2D

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

Seperability of Gaussian

- 2D gaussian is just product of 1D gaussians:

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

Separable!

- Consequently, convolution with a gaussian is separable

$$I * G = I * G_x * G_y$$

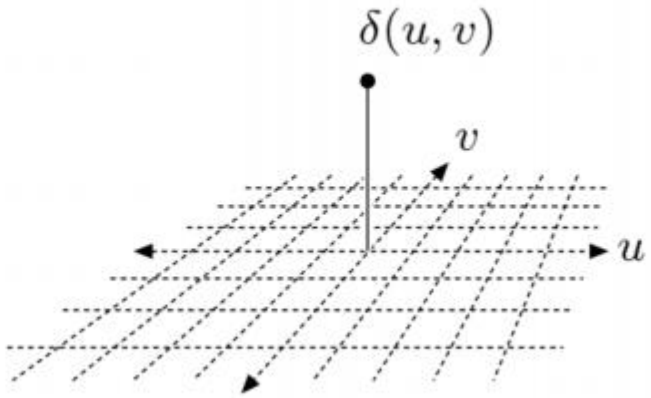
- Where *G* is the 2D discrete gaussian kernel;
- G_x is "horizontal" and G_y is "vertical" 1D discrete Gaussian kernels

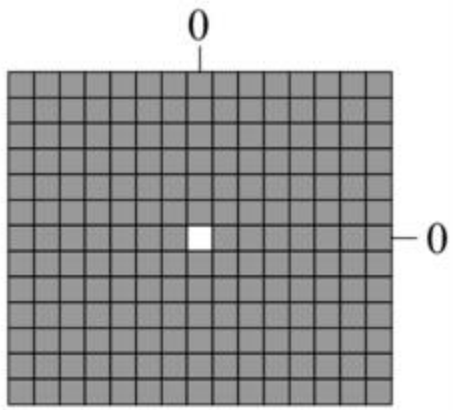
Impulse (or Dirac) Function

- In discrete 2D case, impulse function defined as:

$$\delta(u,v) = \begin{cases} 1 & \text{for } u = v = 0 \\ 0 & \text{otherwise.} \end{cases}$$

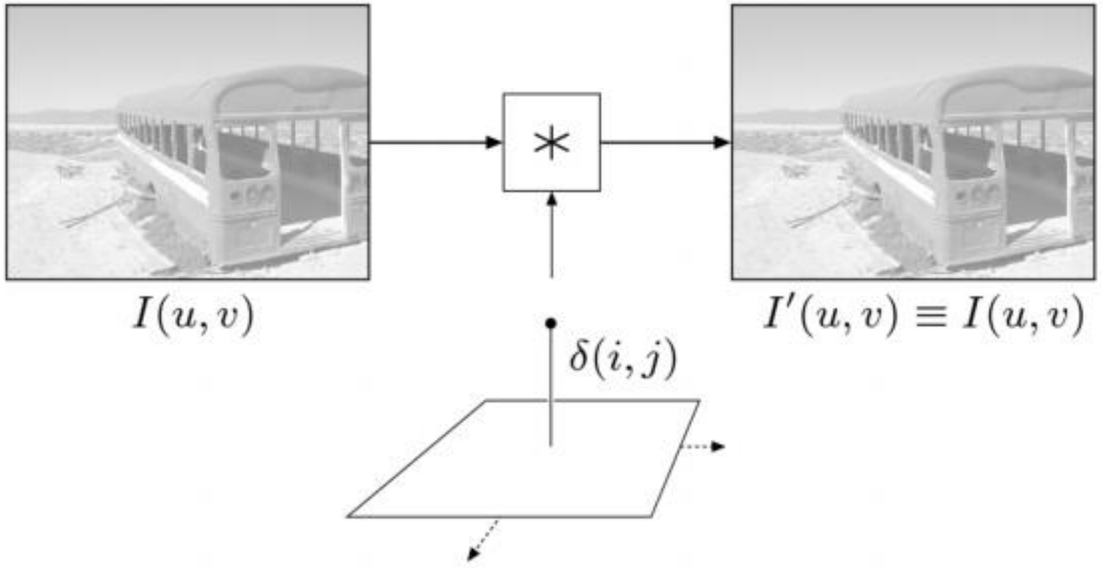
- Impulse function on image?
  - A white pixel at origin, on black background





- Impulse function neutral under convolution (no effect)
- Convolving an image using impulse function as filter = image

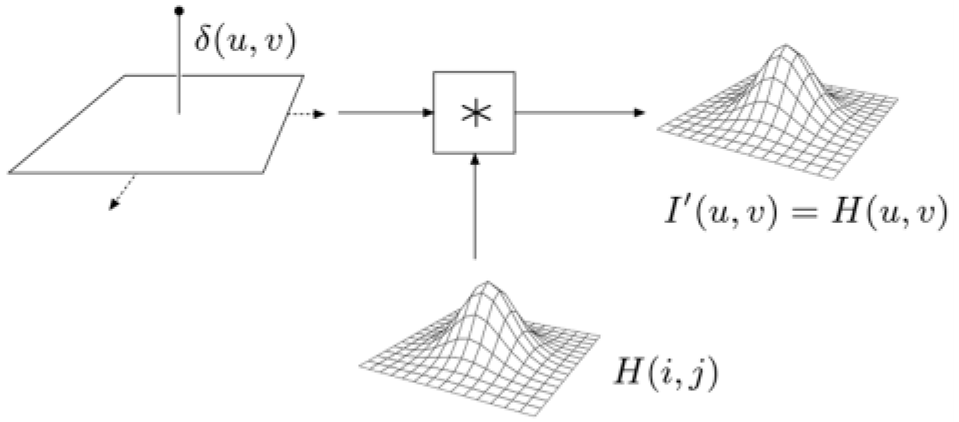
$$I * \delta = I$$



- Reverse case? Apply filter *H* to impulse function
- Using fact that convolution is commutative

$$H * \delta = \delta * H = H$$

- Result is the filter *H*



Noise

- While taking picture (during capture), noise may occur
- Noise? Errors, degradations in pixel values
- Examples of causes:
  - Focus blurring
  - Blurring due to camera motion
- Additive model for noise: $\boldsymbol{H} * \boldsymbol{I} + \boldsymbol{Noise}$

- Removing noise called **Image Restoration**
- Image restoration can be done in:
  - Spatial domain, or
  - Frequency domain

- **Type of noise determines best types of filters for removing it!!**
- **Salt and pepper noise:** Randomly scattered black + white pixels
- Also called **impulse noise, shot noise or binary noise**
- Caused by sudden sharp disturbance

(a) Original image (b) With added salt & pepper noise

Courtesy
Allasdair McAndrews

- **Gaussian Noise:** idealized form of white noise *added to* image, normally distributed $I + Noise$
- **Speckle Noise:** pixel values *multiplied* by random noise

$I(1 + Noise)$

(a) Gaussian noise

(b) Speckle noise

Courtesy Allasdair McAndrews

- **Periodic Noise:** caused by disturbances of a periodic nature

- Salt and pepper, gaussian and speckle noise can be cleaned using spatial filters

- Periodic noise can be cleaned using frequency domain filtering (later)

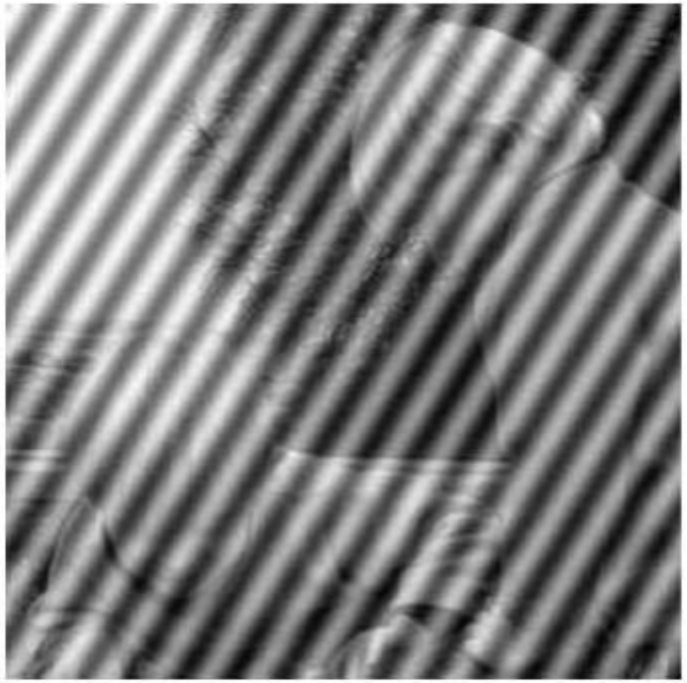
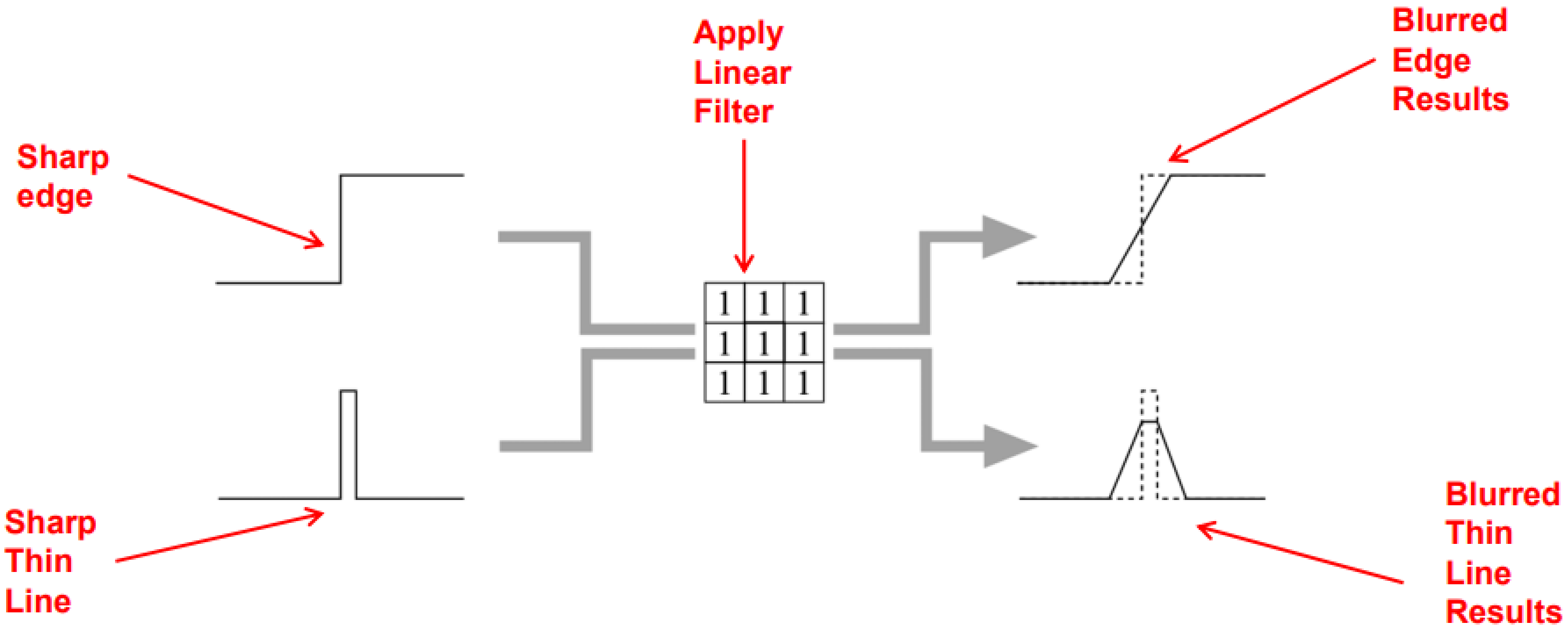


Figure 5.3: The twins image corrupted by periodic noise

Courtesy Allasdair McAndrews

Non-linear filters

- Linear filters blurs all image structures points, edges and lines, reduction of image quality (bad!)
- Linear filters thus not used a lot for removing noise



- **Example:** Using linear filter to clean salt and pepper noise just causes smearing (not clean removal)
- Try non-linear filters?

Courtesy Allasdair McAndrews

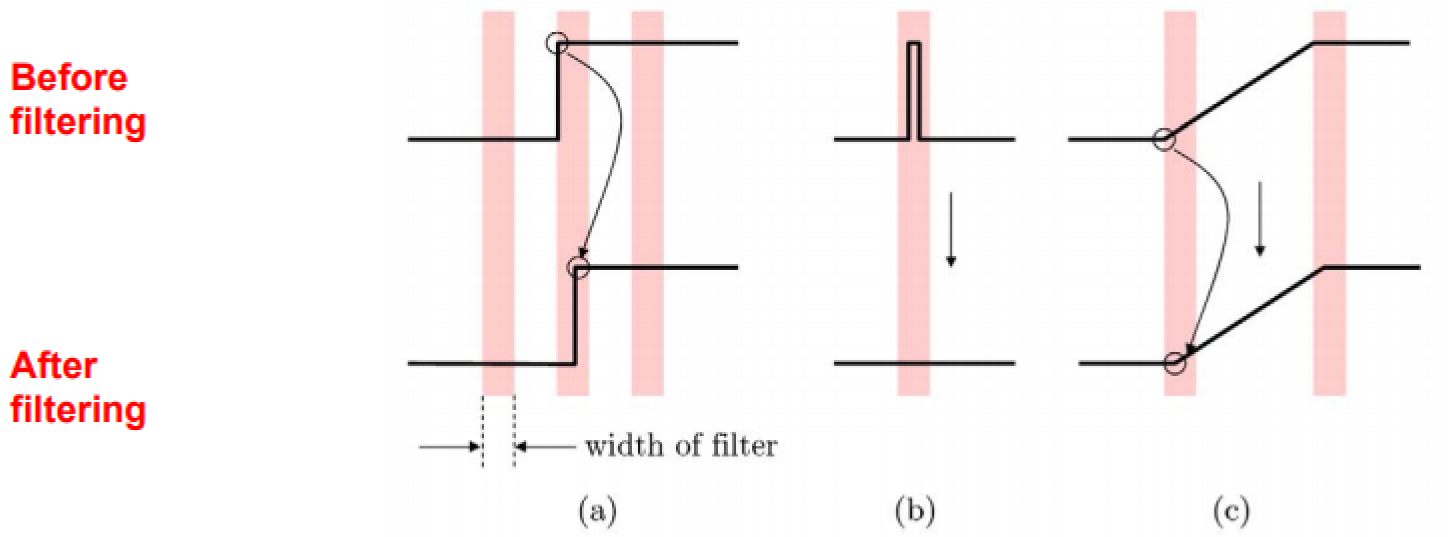
(a) 3×3 averaging

(b) 7×7 averaging

- Pixels in filter range combined by some non-linear function
- Simplest examples of nonlinear filters: Min and Max filters

$$I'(u,v) \leftarrow \min \{ I(u+i, v+j) \mid (i,j) \in R \}$$

$$I'(u,v) \leftarrow \max \{ I(u+i, v+j) \mid (i,j) \in R \}$$



Effect of Minimum filter

Step Edge (shifted to right)

Narrow Pulse (removed)

Linear Ramp (shifted to right)

(a) (b) (c)

Original Image with Salt-and-pepper noise

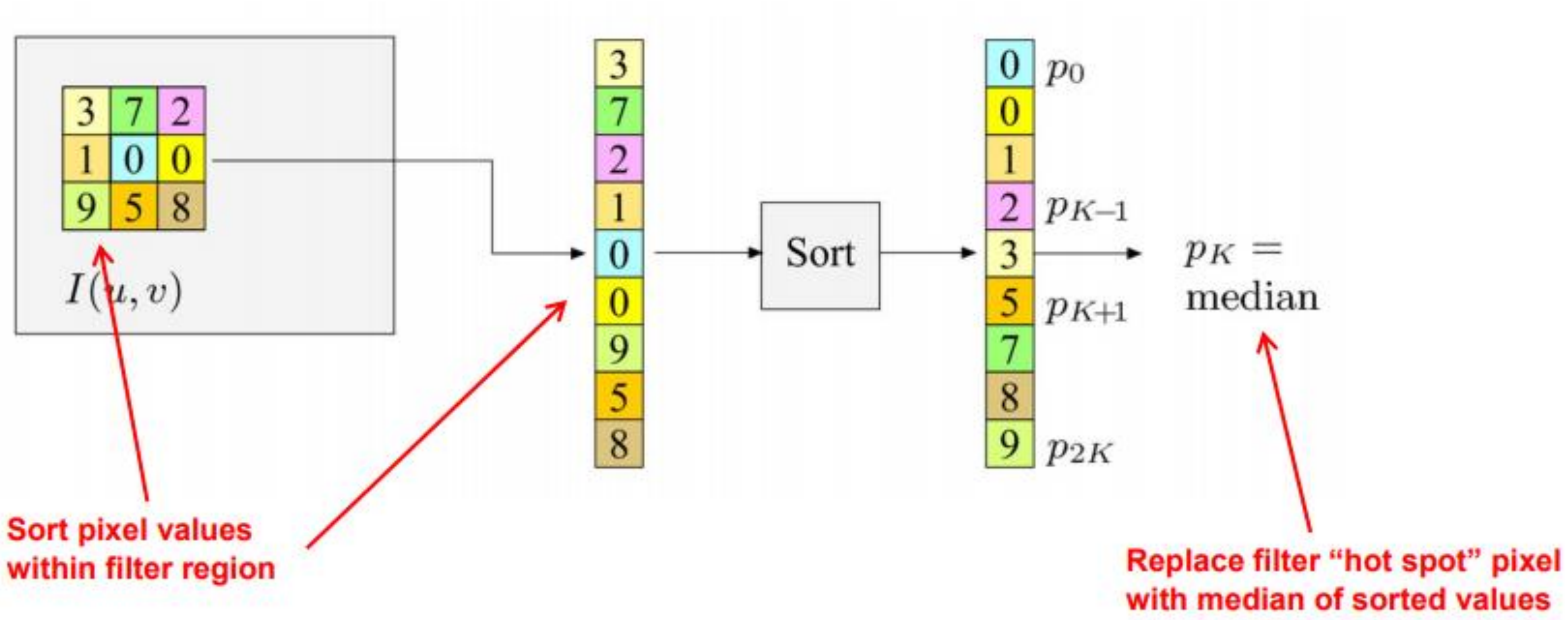
Minimum filter removes bright spots (maxima) and widens dark image structures

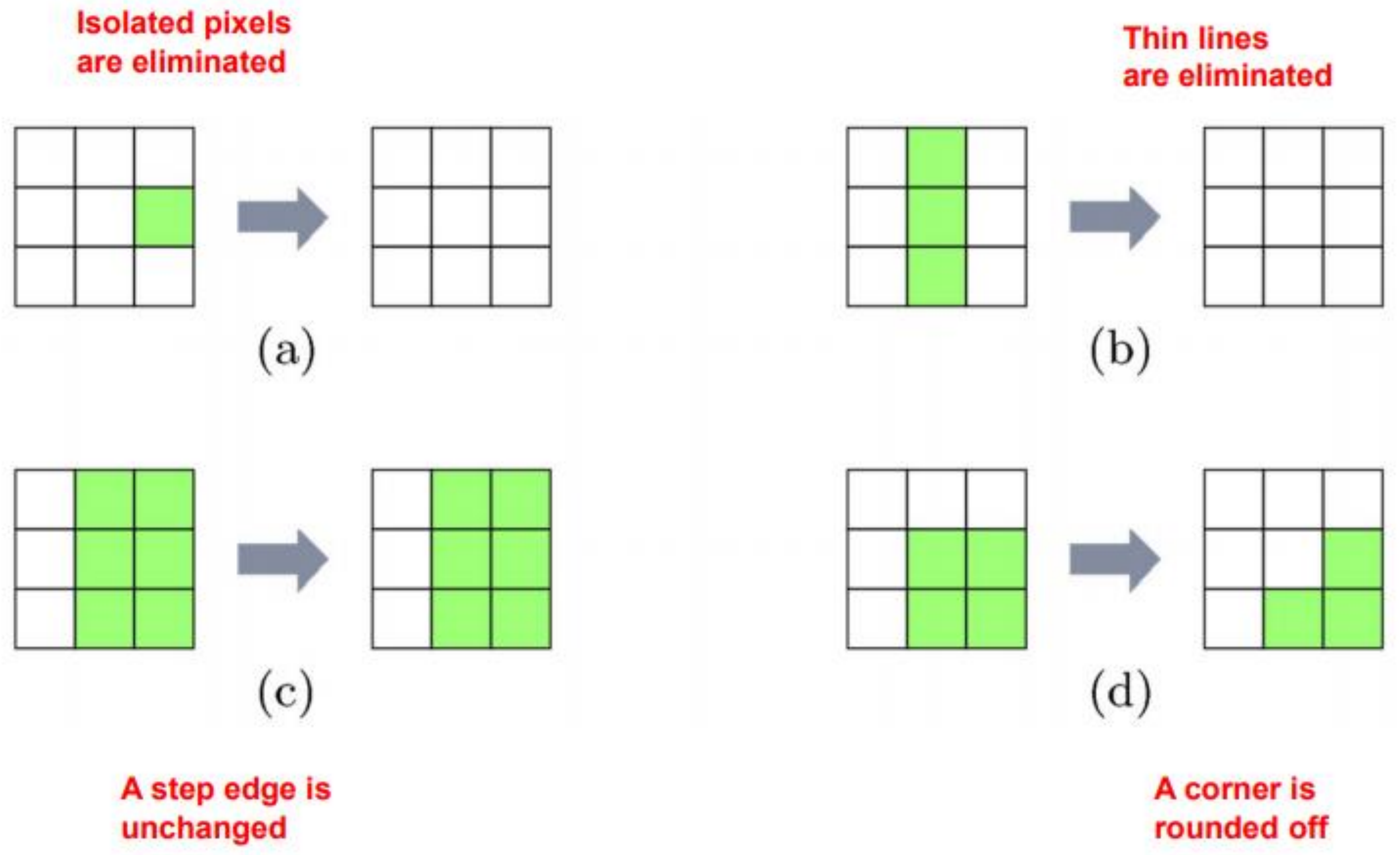
Maximum filter (opposite effect): Removes dark spots (minima) and widens bright image structures

Median Filter

- Much better at removing noise and keeping the structures

$$I'(u,v) \leftarrow \text{median}\,\{I(u+i, v+j) \mid (i,j) \in R\}$$




Isolated pixels
are eliminated
Thin lines
are eliminated
(a)
(b)
(c)
(d)
A step edge is
unchanged
A corner is
rounded off

(a)

Original Image with Salt-and-pepper noise

(b)

Linear filter removes some of the noise, but not completely. Smears noise

(c)

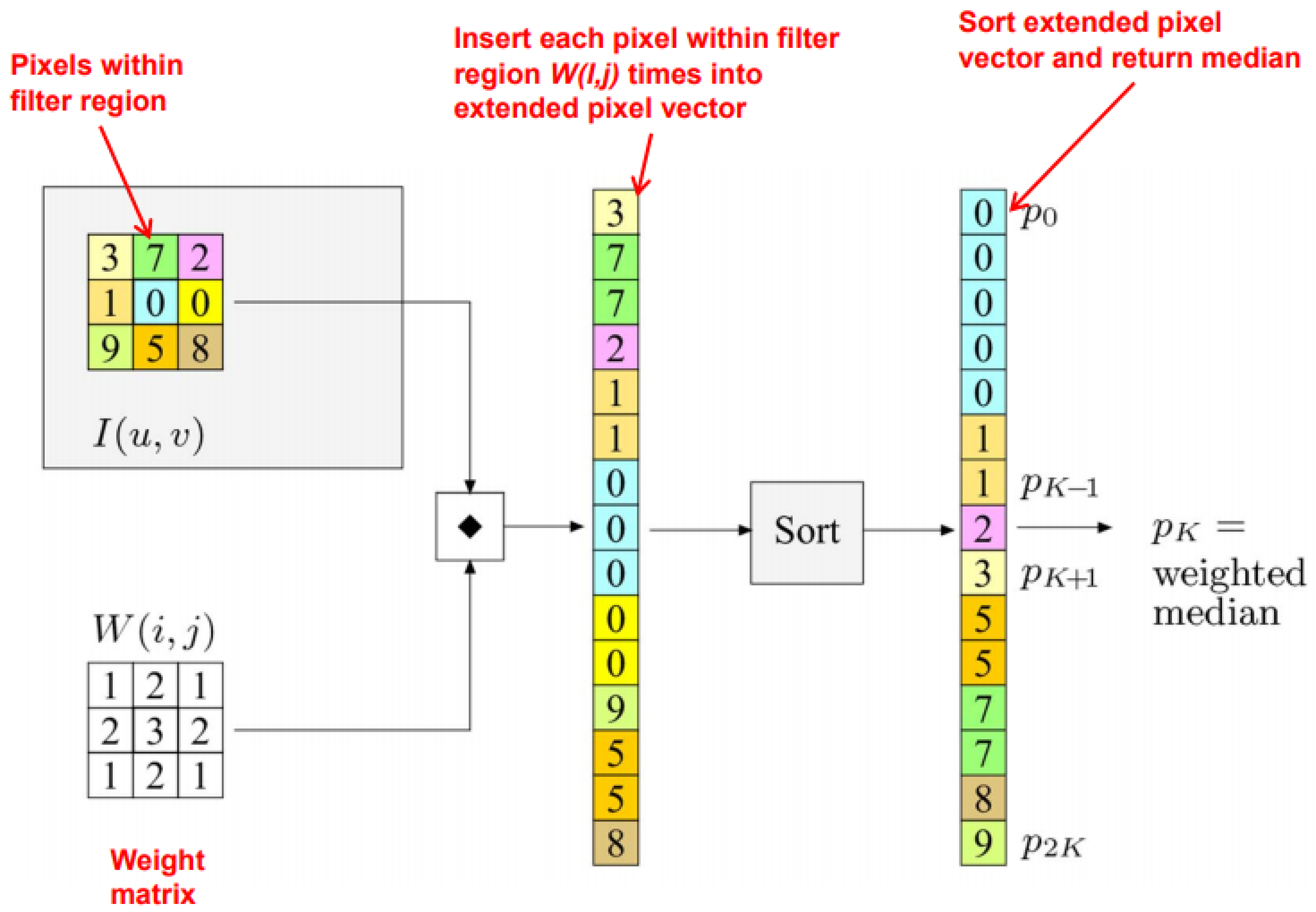
Median filter salt-and-pepper noise and keeps image structures largely intact. But also creates small spots of flat intensity, that affect sharpness

Weighted Median Filter

- Color assigned by median filter determined by colors of "the majority" of pixels within the filter region
- Considered robust since single high or low value cannot influence result (unlike linear average)
- Median filter assigns weights (number of "votes") to filter positions

$$W(i,j) = \begin{bmatrix} 1 & 2 & 1 \\ 2 & \mathbf{3} & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

- To compute result, each pixel value within filter region is inserted *W(i,j)* times to create **extended pixel vector**
- Extended pixel vector then sorted and median returned



Note: assigning weight to center pixel larger than sum of all other pixel weights inhibits any filter effect (center pixel always carries majority)!!

- More formally, **extended pixel vector** defined as

$$Q = (p_0, \ldots, p_{L-1}) \quad \text{of length} \quad L = \sum_{(i,j) \in R} W(i,j)$$

- For example, following weight matrix yields extended pixel vector of length 15 (sum of weights)

$$W(i,j) = \begin{bmatrix} 1 & 2 & 1 \\ 2 & \mathbf{3} & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

- Weighting can be applied to non-rectangular filters
- Example: *cross-shaped* median filter may have weights

$$W^{+}(i,j) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & \mathbf{1} & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Outlier removal

- Median filter does sorting per pixel (computationally expensive)
- Alternate method for removing salt-and-pepper noise
  - Define noisy pixels as **outliers** (different from neighboring pixels by an amount > D)
- Algorithm:
  - Choose threshold value D
  - For given pixel, compare its value p to mean m of 8 neighboring pixels
  - If $|p - m| > D$, classifiy pixel as noise, otherwise not
  - If pixel is noise, replace its value with m; Otherwise leave its value unchanged
- Method not automatic. Generate multiple images with different values of D, choose the best looking one

- Effects of choosing different values of *D*

(a) $D = 0.2$

D value too small: removes noise from dark regions

(b) $D = 0.4$

D value too large: removes noise from light regions

Courtesy Allasdair McAndrews

- *D* value of 0.3 performs best
- Overall outlier method not as good as median filter