COMP322: Computer Vision & Image Processing

- Filtering
- Filtering properties
- Correlation & convolution
- Sampling & aliasing

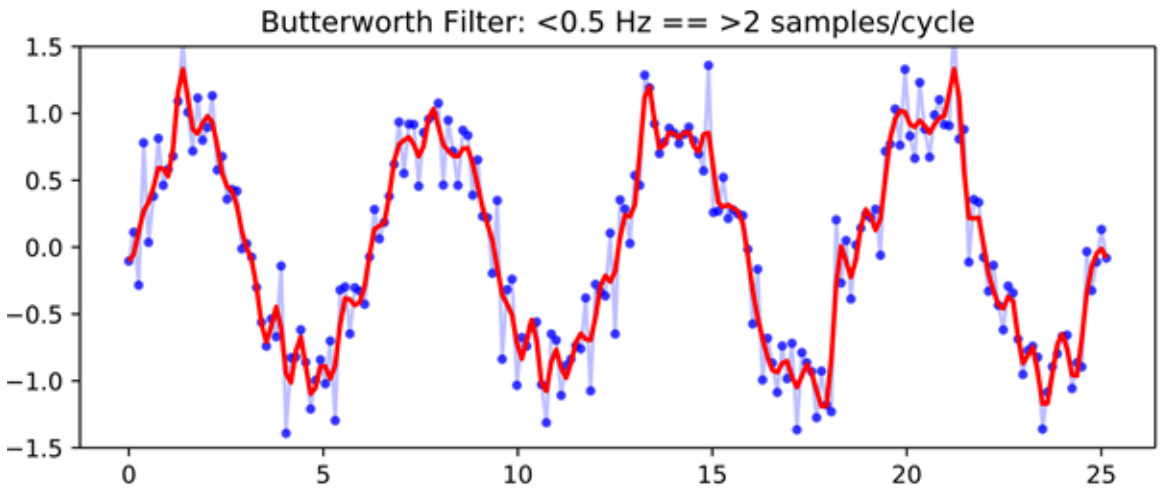
Modified by Hong Xu, mostly taken from slides by Betsy Sanders, in turn modified from James Tompkin's class.

Image Credit: Pixar Animation

Filtering

Definition

An operation that modifies a (measured) signal.

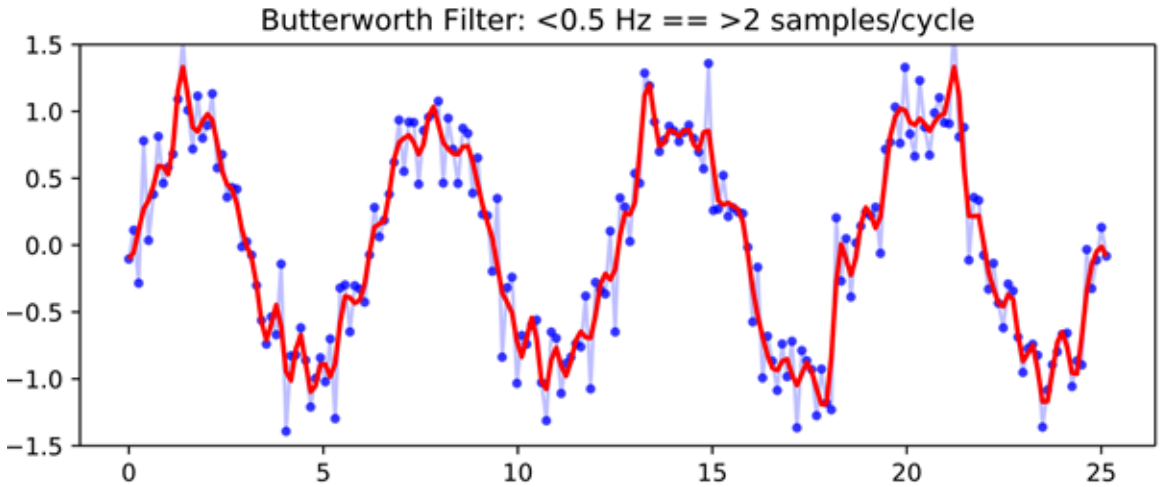


https://ggbaker.ca/data-science/media/butter4.svg

Filtering

Definition

An operation that modifies a (measured) signal.



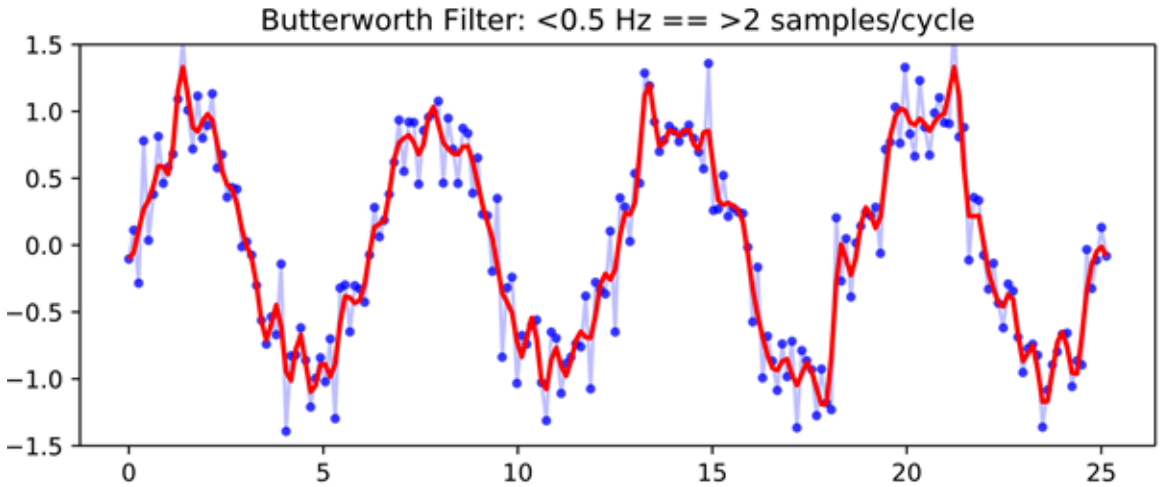
- Modifications include
  - Removing undesirable components
  - Transforming signal in a desirable way
  - Extract specific components of a signal

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

Filtering

Definition

An operation that modifies a (measured) signal.



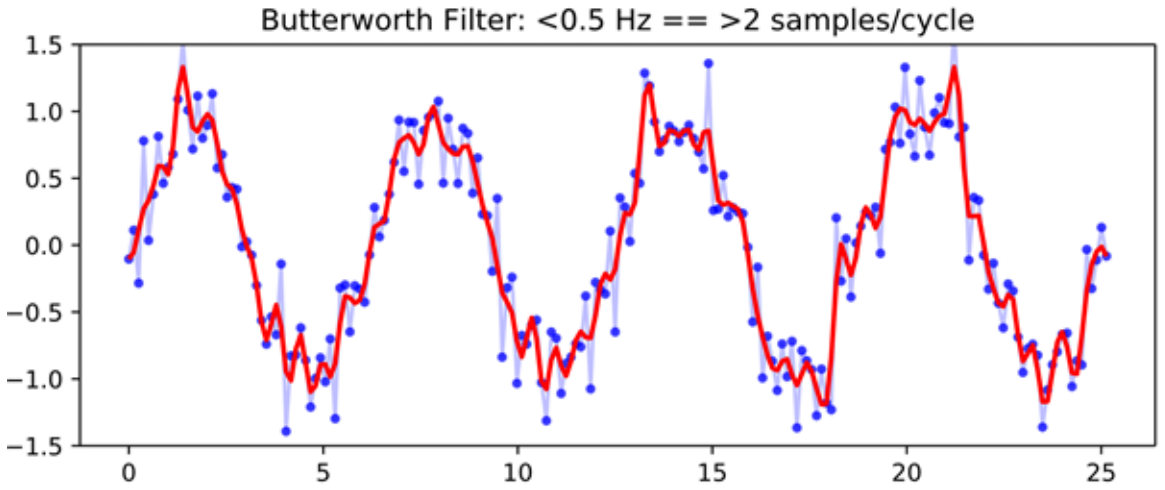
- Modifications include
  - Removing undesirable components
  - Transforming signal in a desirable way
  - Extract specific components of a signal
- Typically, discrete measured signals

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

Filtering

Definition

An operation that modifies a (measured) signal.



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

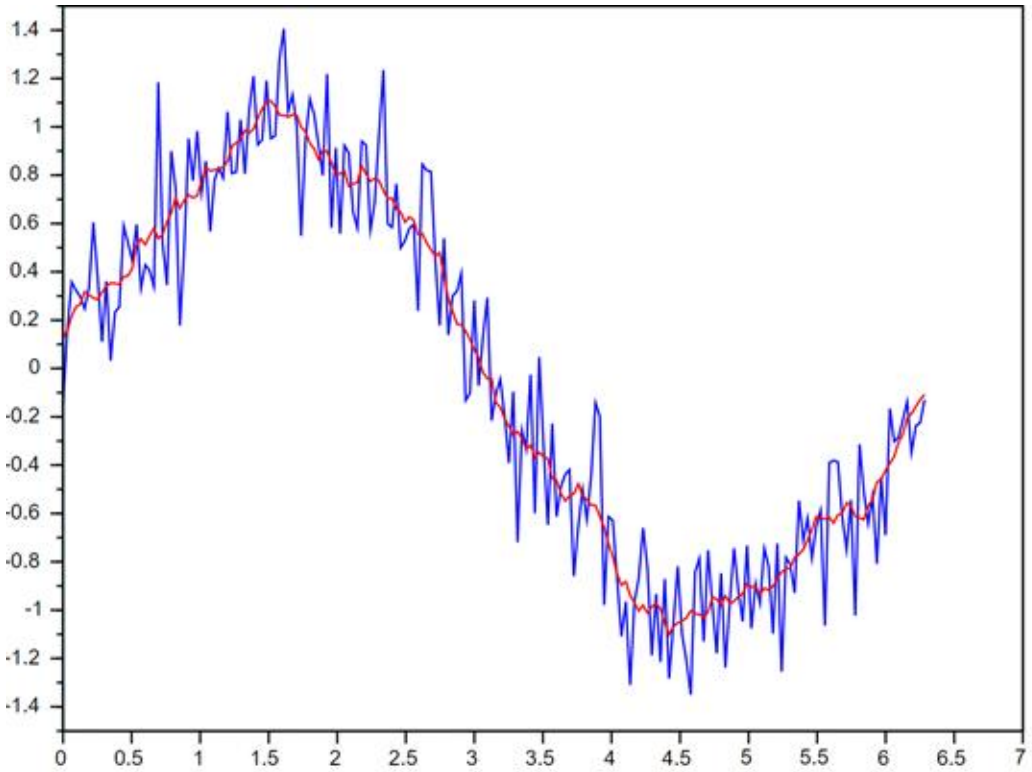
https://ggbaker.ca/data-science/media/butter4.svg

1D Filtering: Moving Average

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

1D Filtering: Moving Average

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



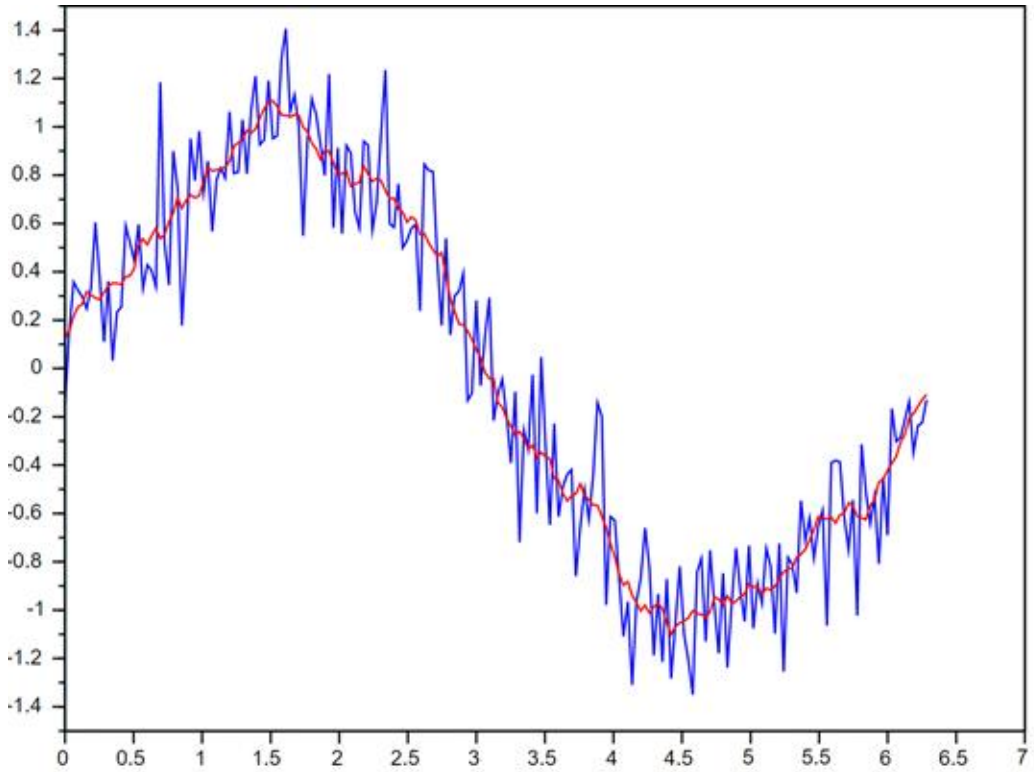
https://en.wikipedia.org/wiki/Moving_average

1D Filtering: Moving Average

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

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

Window size ‘k’



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

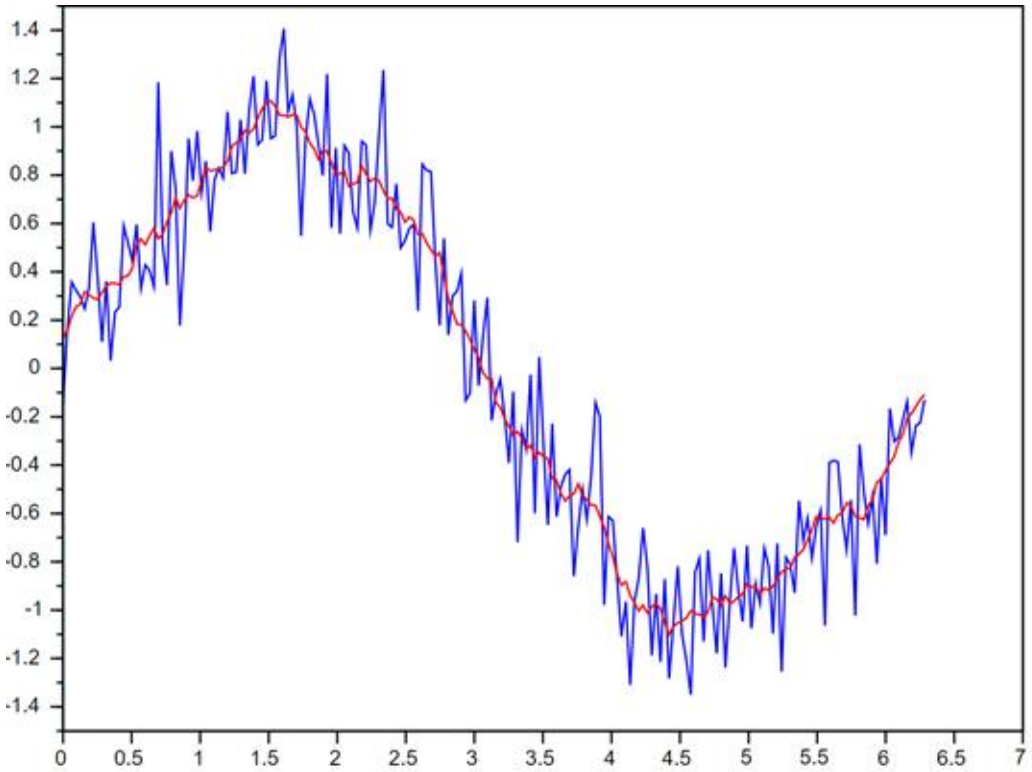
1D Filtering: Moving Average

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

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

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

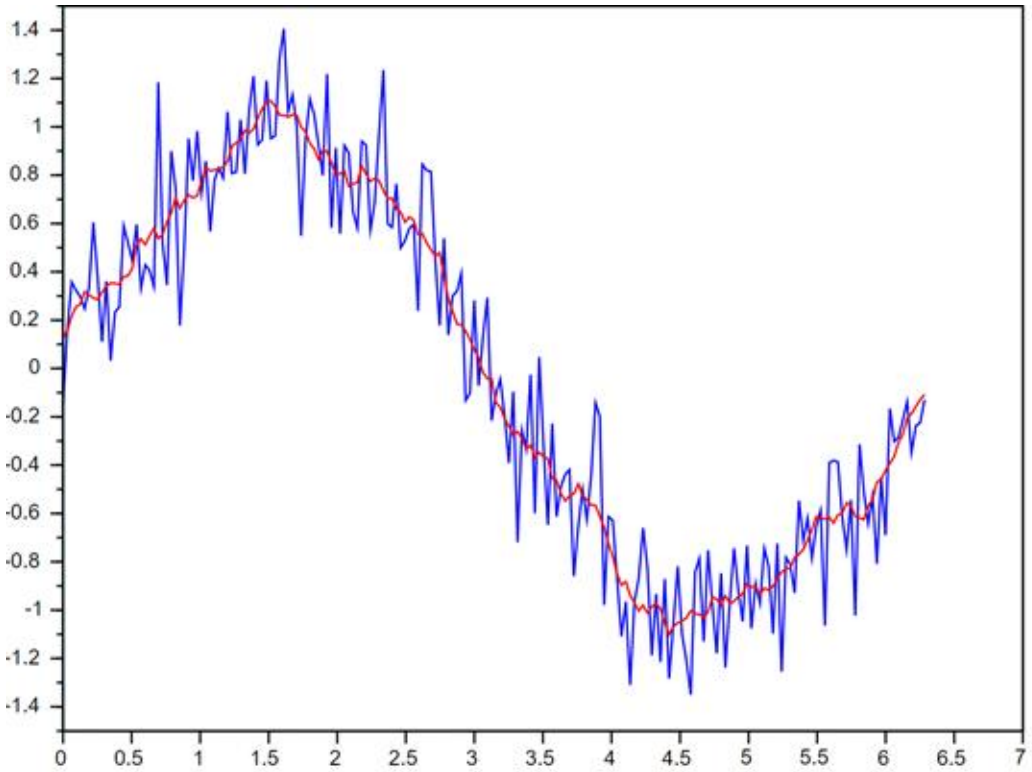
1D Filtering: Moving Average

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

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

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

1D Filtering: Moving Average

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

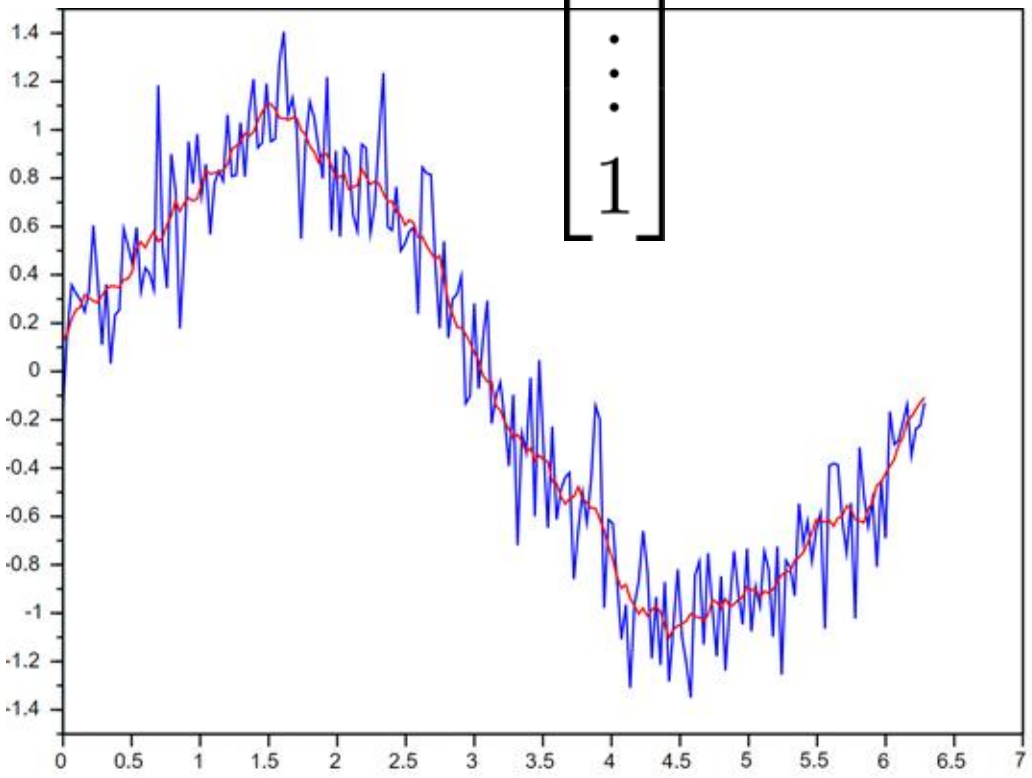
$$h[n] = \frac{1}{k} \sum_{i=n-k+1}^{n} I[i]$$

Window size ‘k’

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

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

Identity filter



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

1D Filtering: Moving Average

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

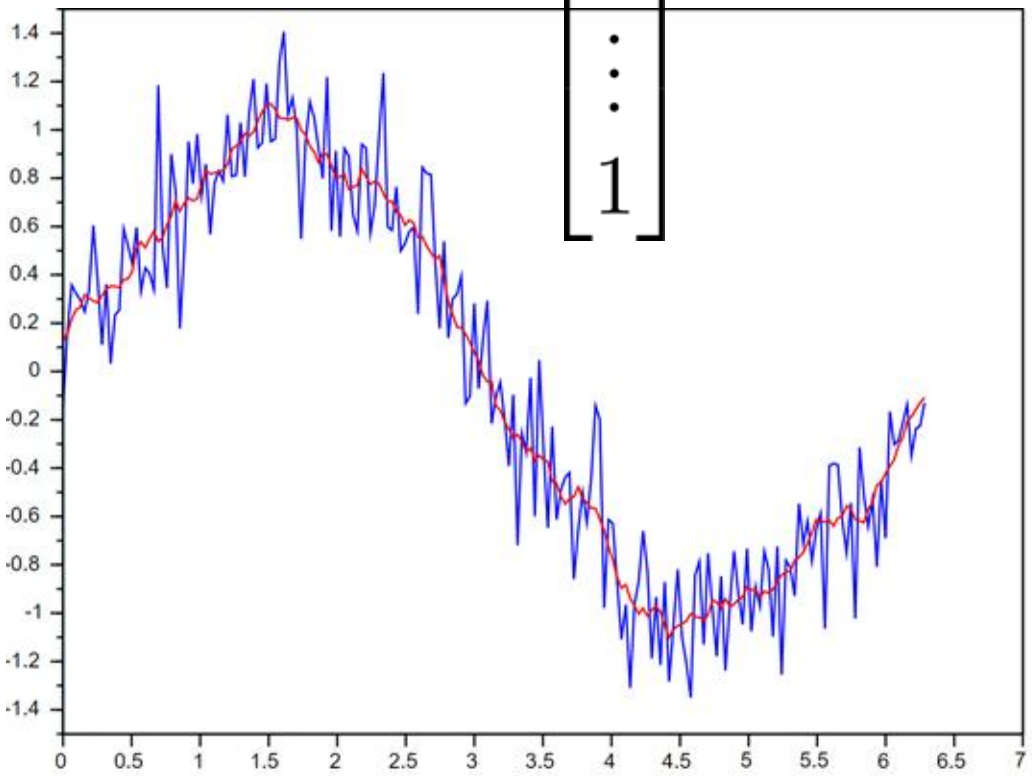
$$h[n] = \frac{1}{k} \sum_{i=n-k+1}^{n} I[i]$$

Window size ‘k’

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

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

Identity filter



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

1D Filtering: Moving Average

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

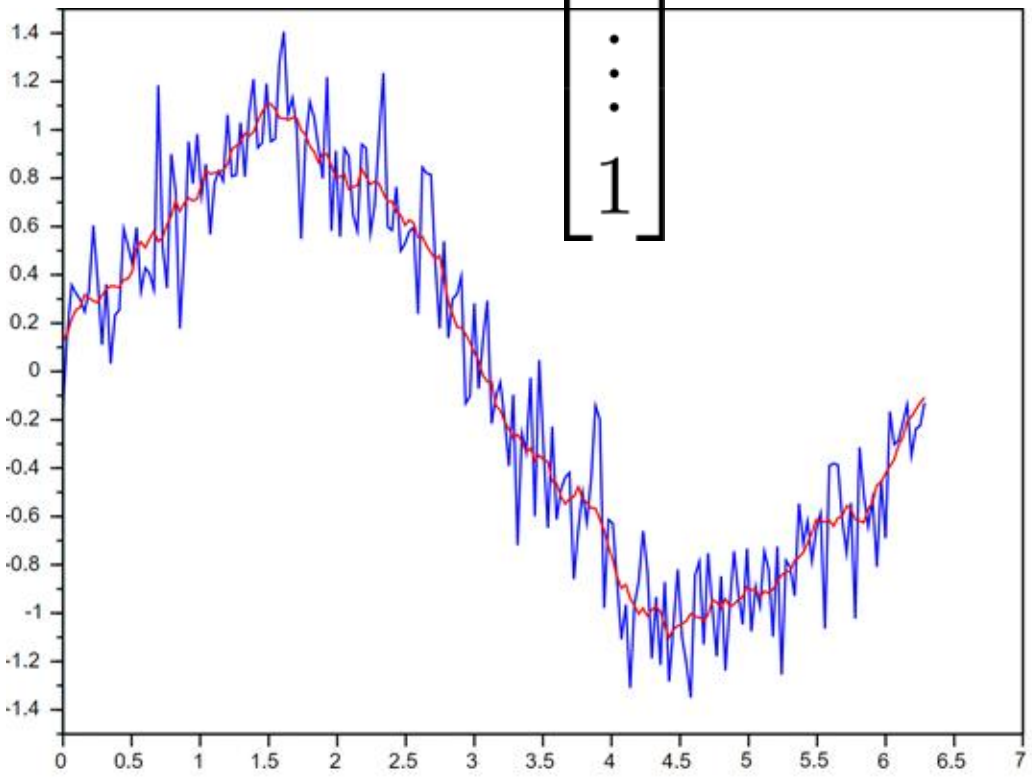
$$h[n] = \frac{1}{k} \sum_{i=n-k+1}^{n} I[i]$$

Window size ‘k’

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

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

Identity filter



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

Different window

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

Filtering Demo

https://cristal.univ-lille.fr/~casiez/1euro/InteractiveDemo/

2D 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’

2D Filtering

Compute function of local neighborhood at each position:

h=output f=filter I=image

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

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

[ ] [ ] [ ]

Note: Filter is often called the 'kernel'

Fundamental Equations of Computer Vision

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

1. Camera Geometry $$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\mathbf{X} \qquad x^T F x' = 0$$

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

1. Optical Flow $$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$$

Fundamental Equations of Computer Vision

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

1. Camera Geometry $$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\mathbf{X}$$ $$x^T F x' = 0$$

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

1. Optical Flow $$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$$

Example: Box Filter

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

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

Image Filtering

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

1	1	1
1	1	1
1	1	1

$I[\cdot,\cdot]$

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	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	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	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot,\cdot]$

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

$$m = 1, n = 1$$
$$k, l = [-1,0,1]$$

Image Filtering

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

1	1	1
1	1	1
1	1	1

$I[\cdot,\cdot]$

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	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	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	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot,\cdot]$

	0								

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

$$m = 1, n = 1$$
$$k, l = [-1,0,1]$$

Image Filtering

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

1	1	1
1	1	1
1	1	1

$I[\cdot,\cdot]$

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	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	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	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot,\cdot]$

	0	10							

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

$$m = 2, n = 1$$
$$k, l = [-1,0,1]$$

Image Filtering

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

$I[.,.]$

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	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	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	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20						

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

$$m=3, n=1$$
$$k,l=[-1,0,1]$$

Image Filtering

$f[\cdot,\cdot]$

1	1	1
1	1	1
1	1	1

$I[\cdot,\cdot]$

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	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	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	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot,\cdot]$

	0	10	20	30					

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

$$m = 4, n = 1$$
$$k, l = [-1,0,1]$$

Image Filtering

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

1	1	1
1	1	1
1	1	1

$I[\cdot,\cdot]$

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	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	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	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot,\cdot]$

	0	10	20	30	30				

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

$$m = 5, n = 1$$
$$k, l = [-1,0,1]$$

Image Filtering

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

1	1	1
1	1	1
1	1	1

$I[\cdot,\cdot]$

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	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	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	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot,\cdot]$

	0	10	20	30	30				

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

$$m = 4, n = 6$$
$$k, l = [-1,0,1]$$

Image Filtering

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

1	1	1
1	1	1
1	1	1

$I[\cdot,\cdot]$

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	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	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	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot,\cdot]$

	0	10	20	30	30				
						?			
				50					

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

$$m = 6, n = 4$$
$$k, l = [-1,0,1]$$

Image Filtering

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

$I[\cdot,\cdot]$

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	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	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	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot,\cdot]$

	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

What does it do?

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

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

Box Filter

What does it do?

- Replaces each pixel with an average of its neighborhood

- Achieve smoothing effect (remove ‘sharp’ features)

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

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

Box Filter

What does it do?

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

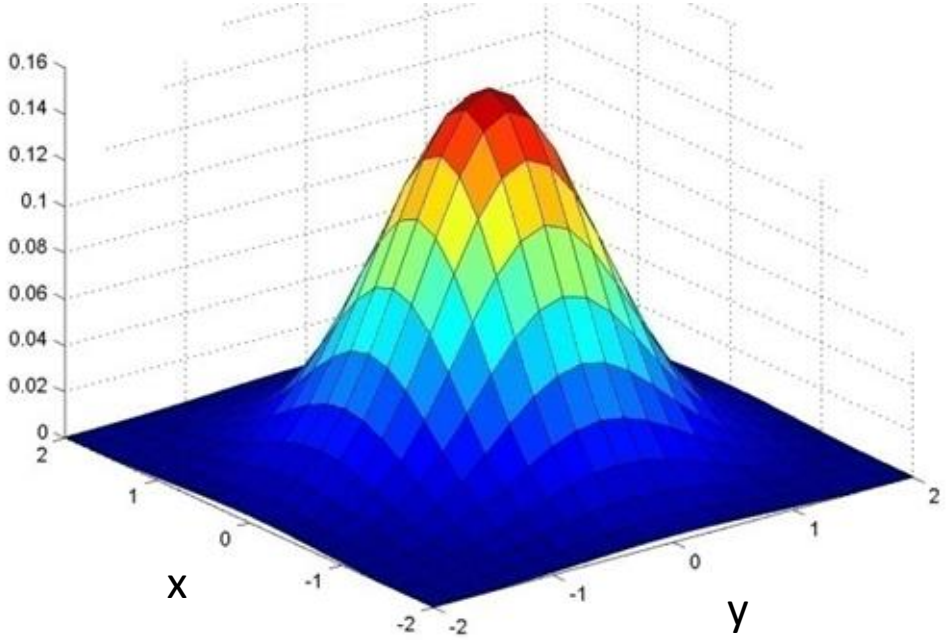
$$f[\cdot,\cdot]$$

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

Smoothing with Box Filter

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

Gaussian Filter



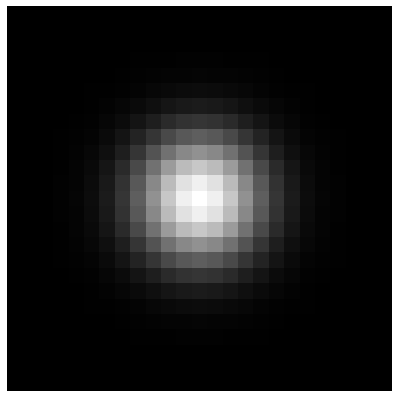
x

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

y

Kernel size 5 x 5,
Standard deviation σ = 1

Viewed from top

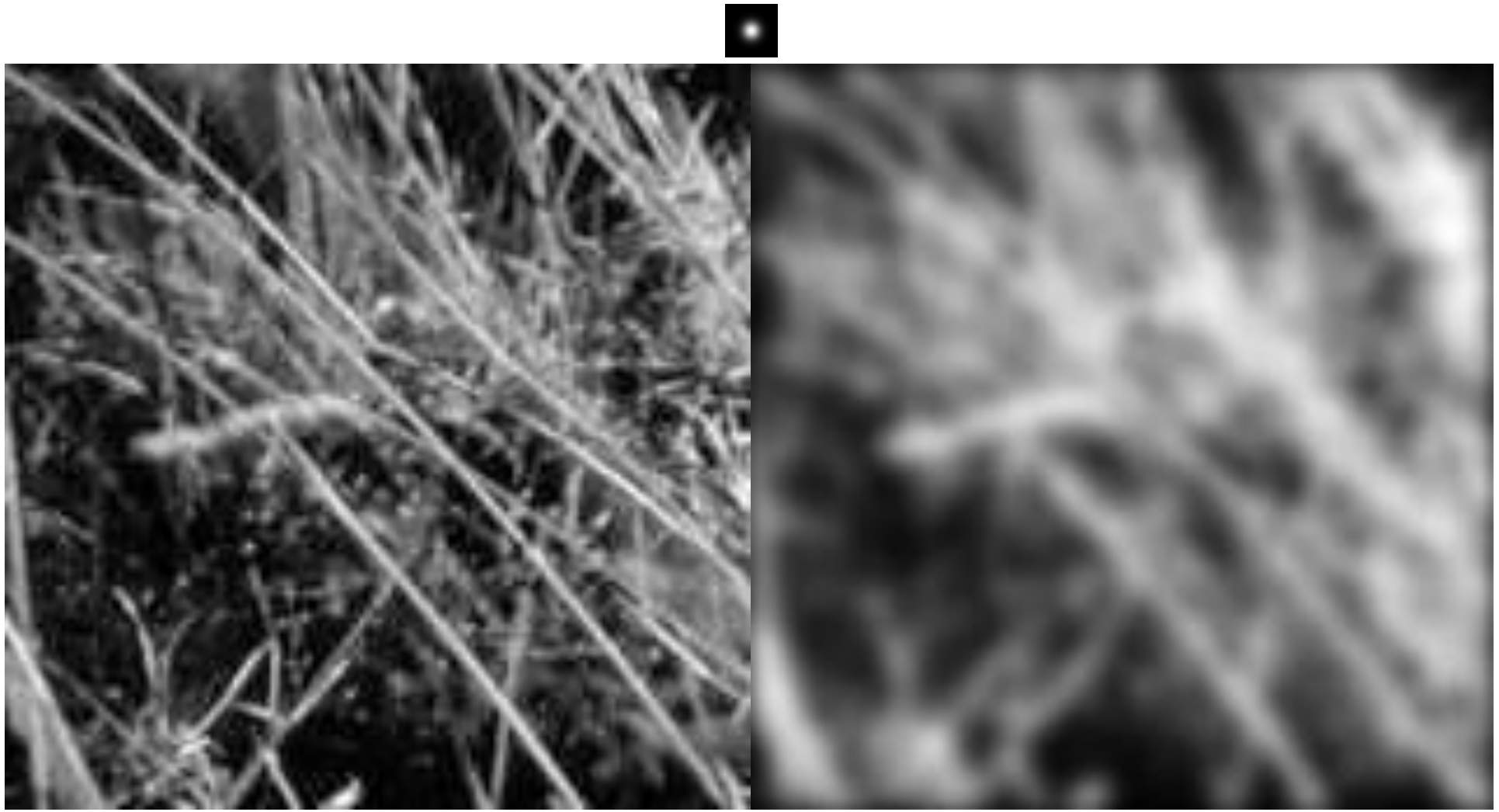


y

x

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

Smoothing with Gaussian Filter



Smoothing with Box Filter

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

Image Filtering

Compute function of local neighborhood
at each position:

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

Image Filtering

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.

Image Filtering

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.

Image Filtering

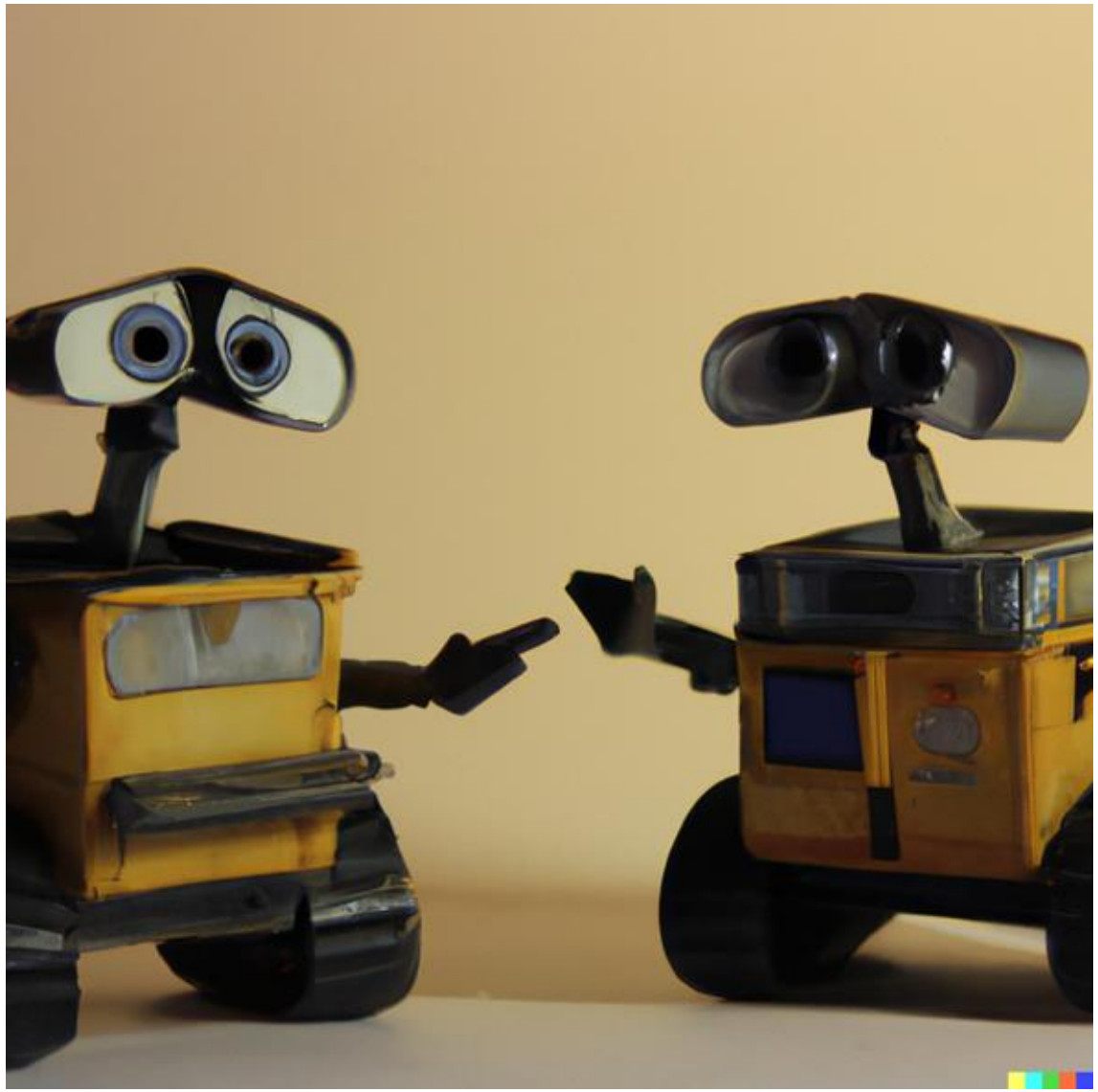
Compute function of local neighborhood
at each position:

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

Really important!

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

Let’s think about this



Dall-E: “wall-e having a discussion with another wall-e”

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

$-\frac{1}{9}$

1	1	1
1	1	1
1	1	1

1. Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

?

1. Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

Filtered
(no change)

2. Practice with linear filters

Original

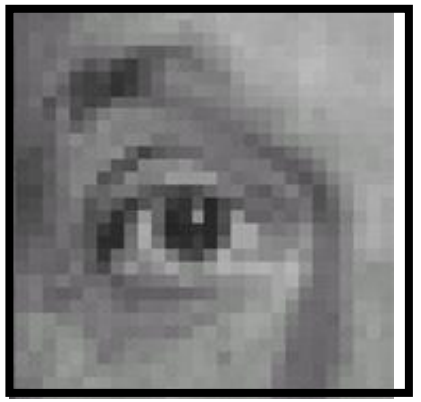
0	0	0
0	0	1
0	0	0

?

2. Practice with linear filters

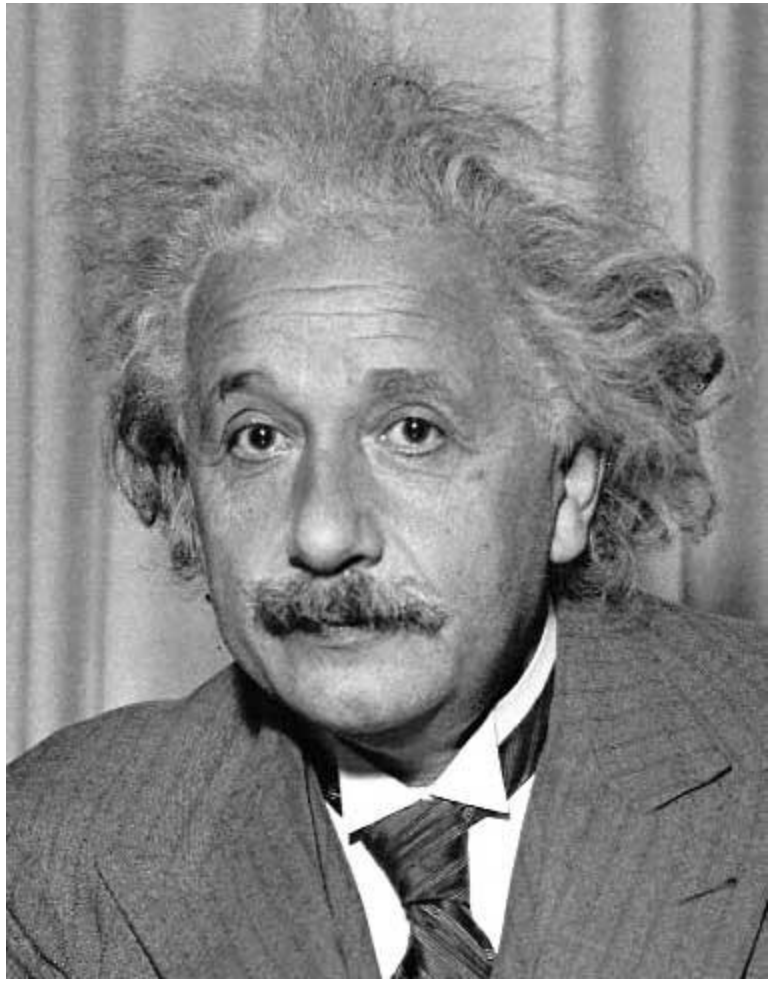
Original

0	0	0
0	0	1
0	0	0



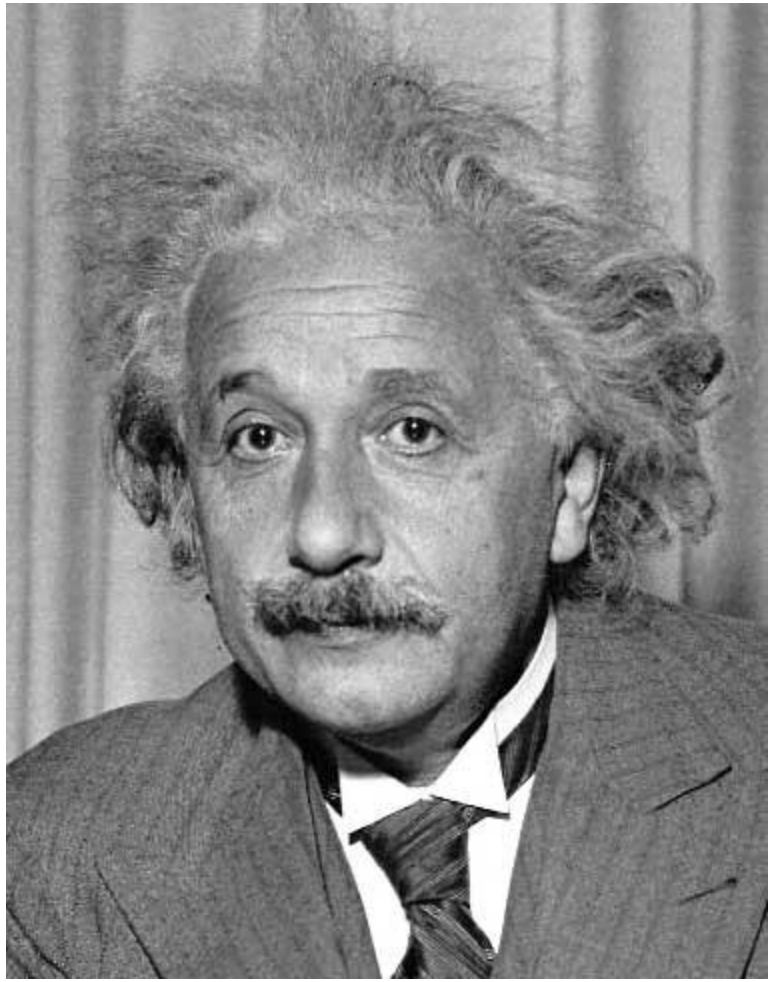
Shifted left
By 1 pixel

3. Practice with linear filters



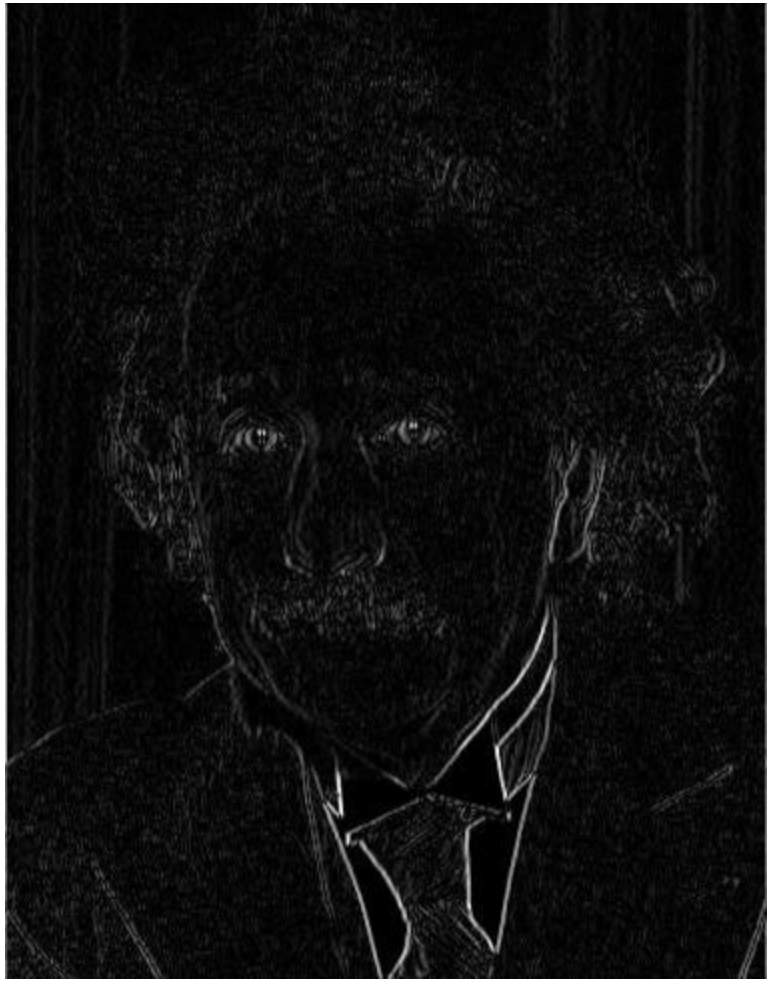
1	0	-1
2	0	-2
1	0	-1

3. Practice with linear filters



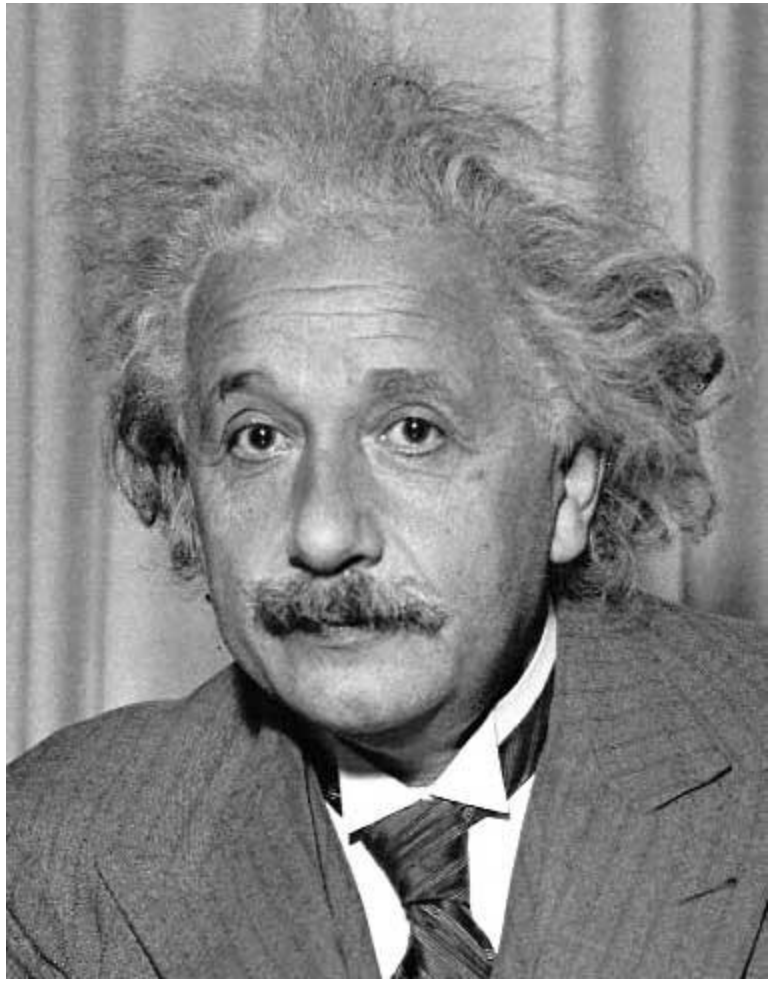
1	0	-1
2	0	-2
1	0	-1

Sobel



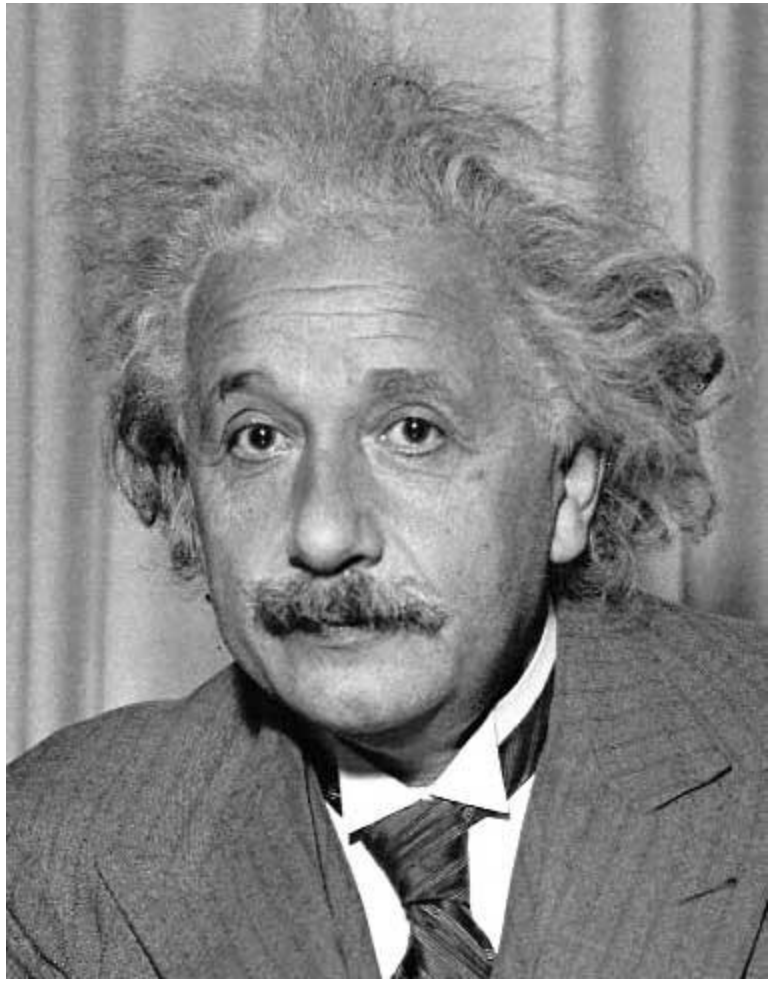
Vertical Edge
(absolute value)

3. Practice with linear filters



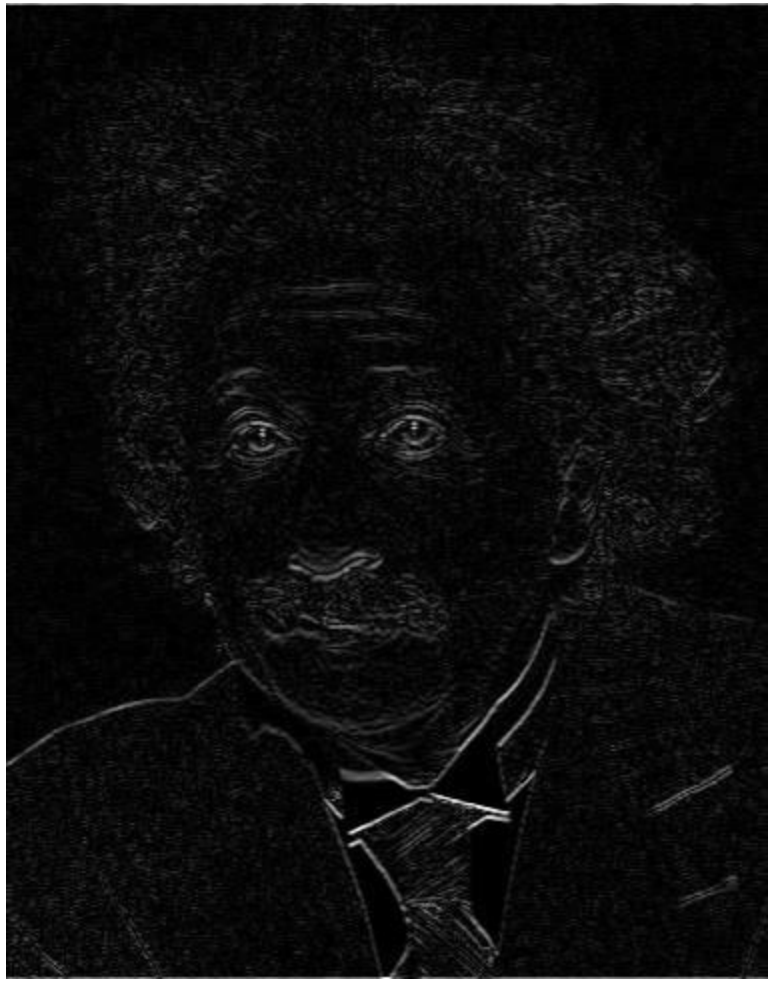
1	2	1
0	0	0
-1	-2	-1

3. Practice with linear filters



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

Sobel



Horizontal Edge
(absolute value)

4. Practice with linear filters

Original

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

(Note that filter sums to 1)

4. Practice with linear filters

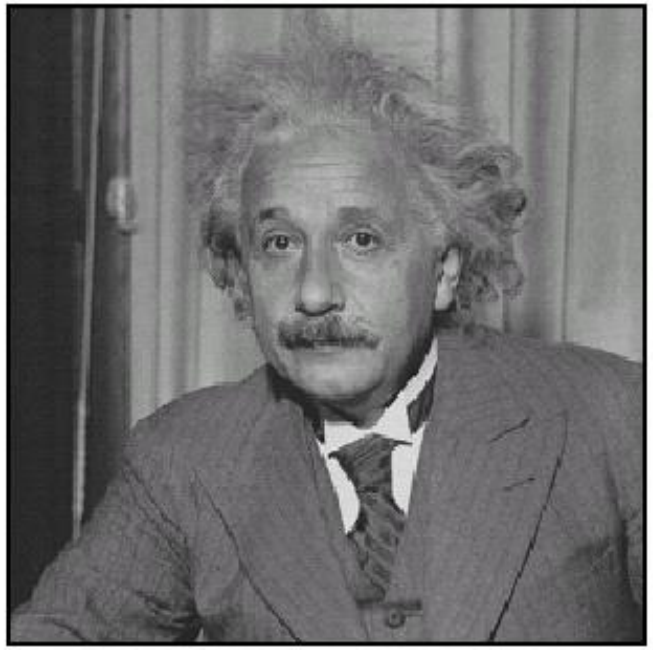
Original

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

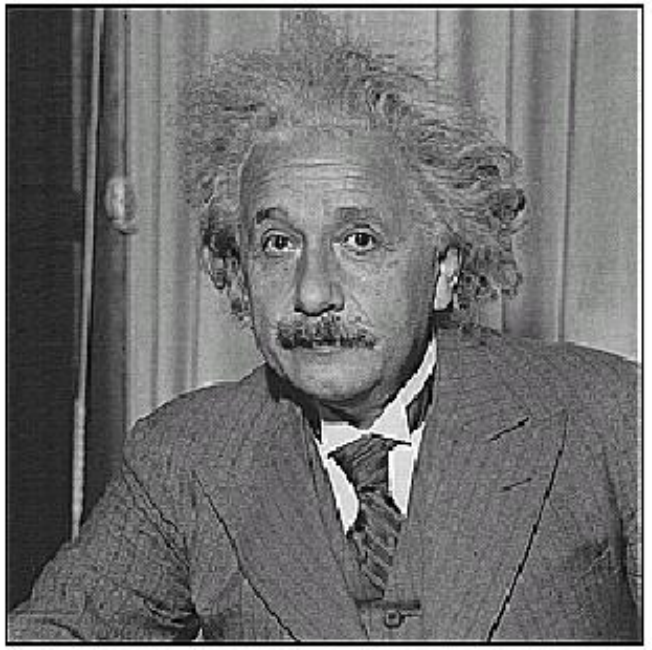
Sharpening filter

- Accentuates differences with local average

4. 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
- methods:
  - clip filter (black)
  - wrap around
  - copy edge
  - reflect across edge

Filters in Practice

What about near the edge?

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

Filters in Practice

What about near the edge?

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

Filters in Practice

What about near the edge?

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

Filters in Practice

What about near the edge?

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

Filters in Practice

What about near the edge?

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

Filters in Practice

What about near the edge?

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

Filters in Practice

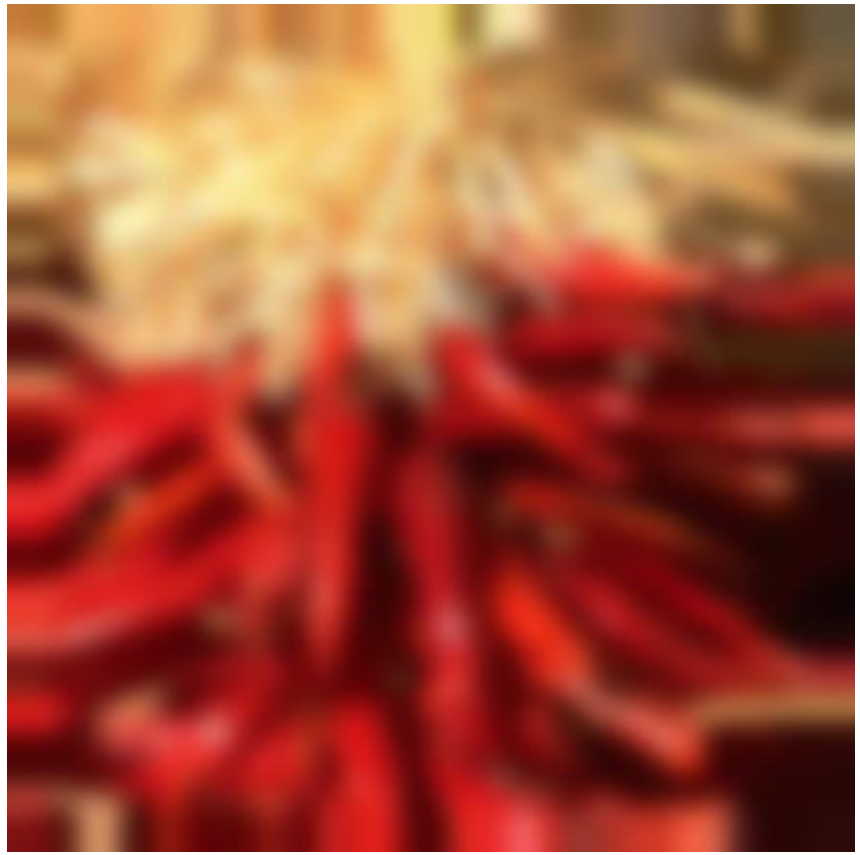
What about near the edge?

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

Filters in Practice

What about near the edge?

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



Filters in Practice

What about near the edge?

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

Filters in Practice

What about near the edge?

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

Filters in Practice

What about near the edge?

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

Filters in Practice

What about near the edge?

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

CS322

Properties of image filtering methods

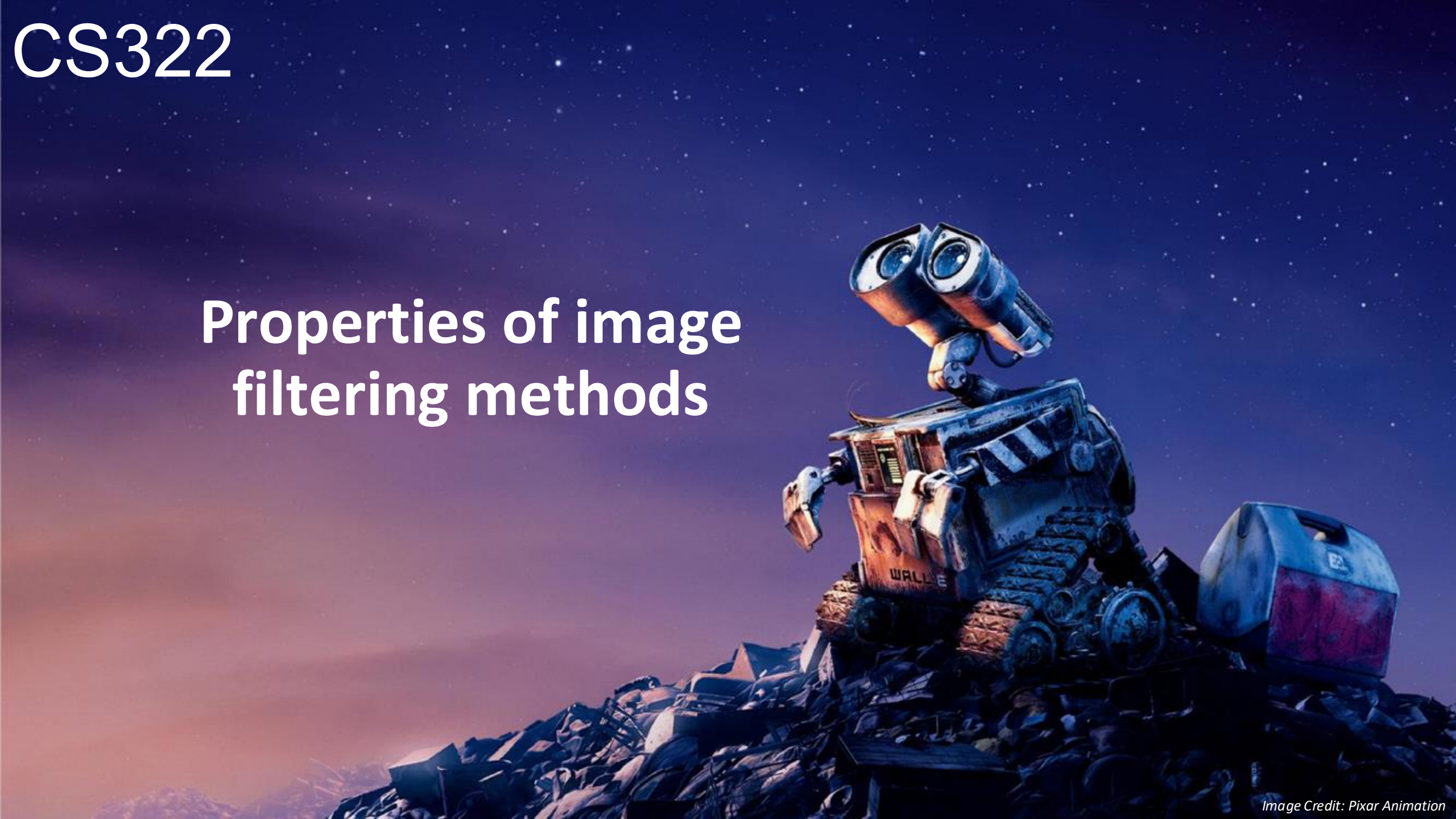


Image Credit: Pixar Animation

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)
```

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)
```

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° rotated filter kernel.
Correlation and convolution are identical when the filter kernel is rotationally symmetric*.

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

Linear Filter Properties

Linear Filter Properties

Linearity:

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

Linear Filter Properties

Linearity:

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

Shift/translation invariance:

Same behavior given intensities regardless of pixel location m,n

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

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
- Correlation is _not_ commutative (rotation effect) – produces rotated version of output.

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: $a * (b_1 * b_2 * b_3)$ -> computationally faster
- 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: $a * e = a$ when $e = [0, 0, 1, 0, 0]$,

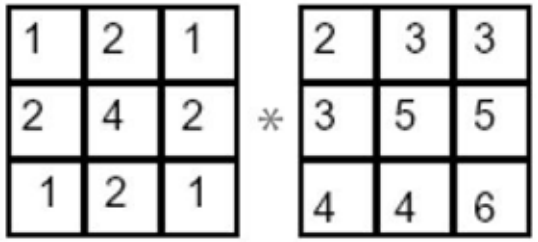
Convolution in Convolutional Neural Networks

- Convolution is the basic operation in CNNs

- Learning convolution kernels allows us to learn which `features' provide useful information in images.

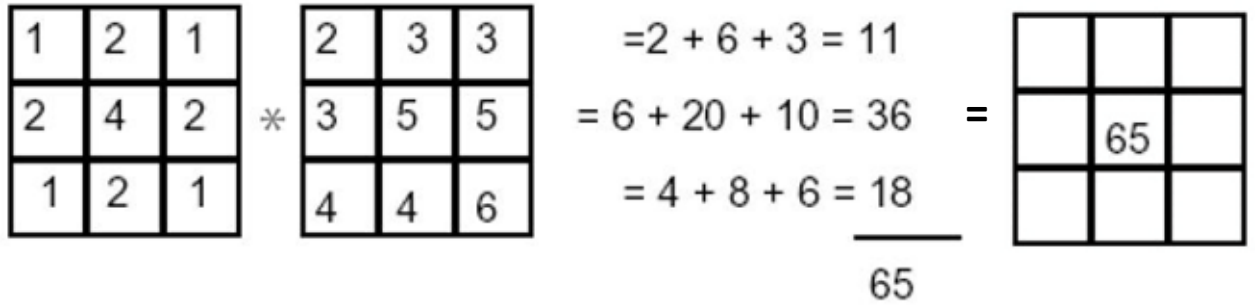
Separability: 2x 1D convolution = 2D convolution

2D convolution
(center location only)



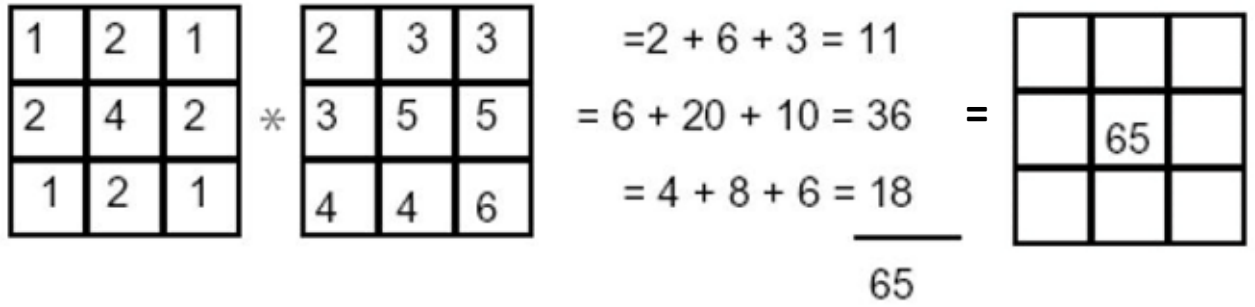
Separability Example

2D convolution
(center location only)



Separability Example

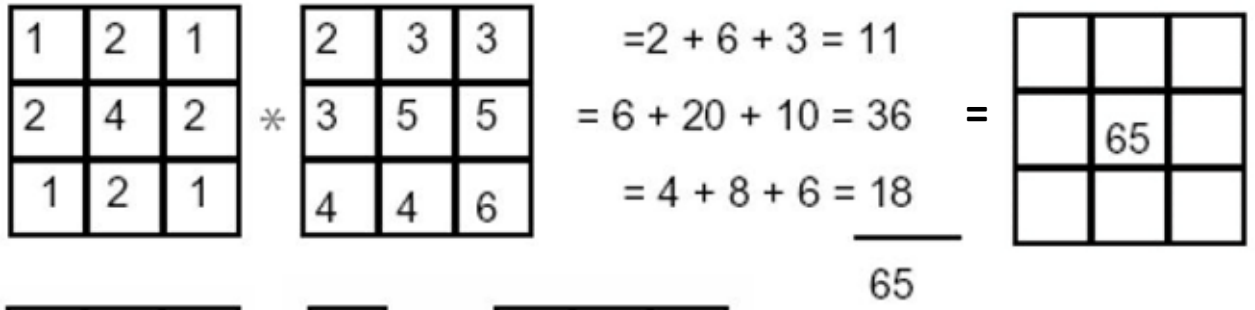
2D convolution
(center location only)



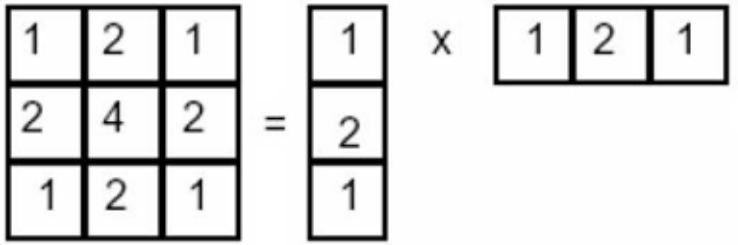
The filter factors into a product of 1D filters:

Separability Example

2D convolution
(center location only)

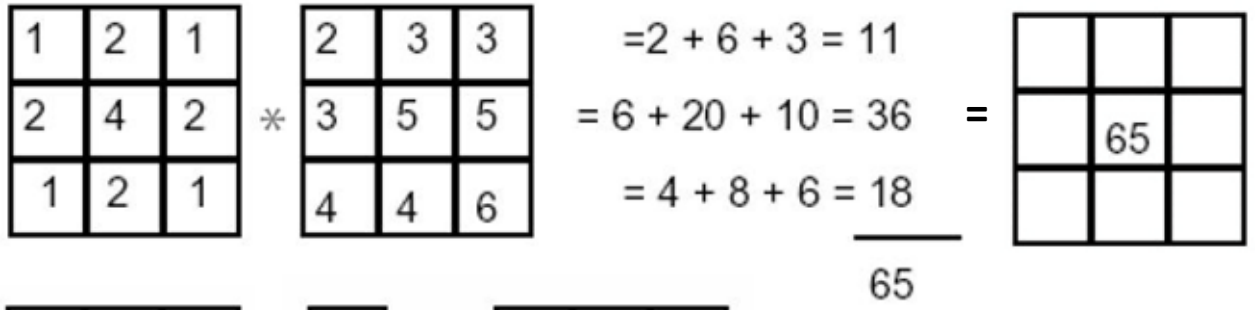


The filter factors
into a product of 1D
filters:

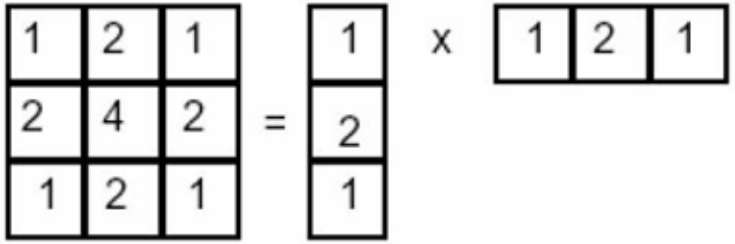


Separability Example

2D convolution
(center location only)



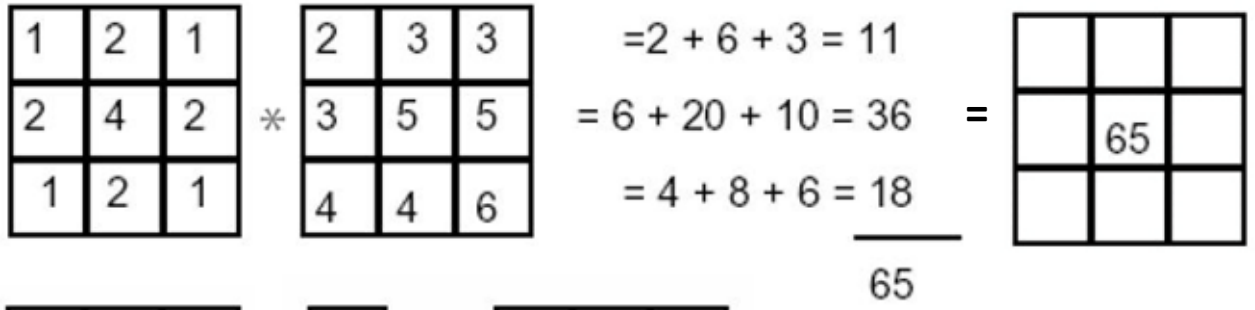
The filter factors into a product of 1D filters:



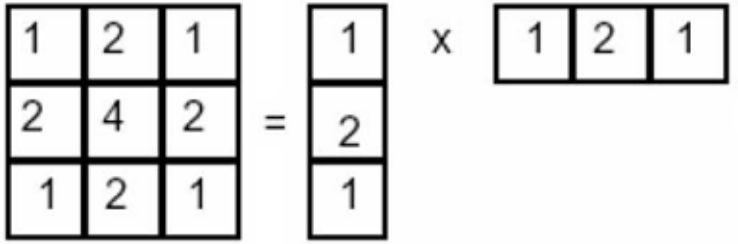
Perform convolution along rows:

Separability Example

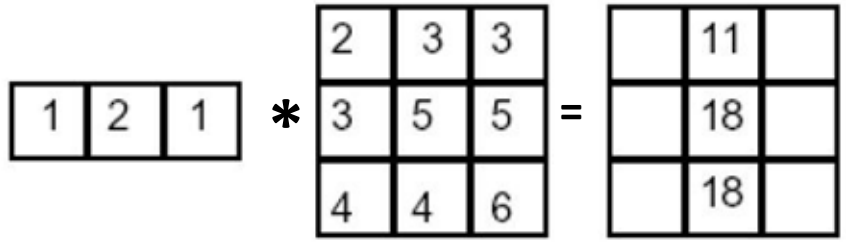
2D convolution
(center location only)



The filter factors
into a product of 1D
filters:

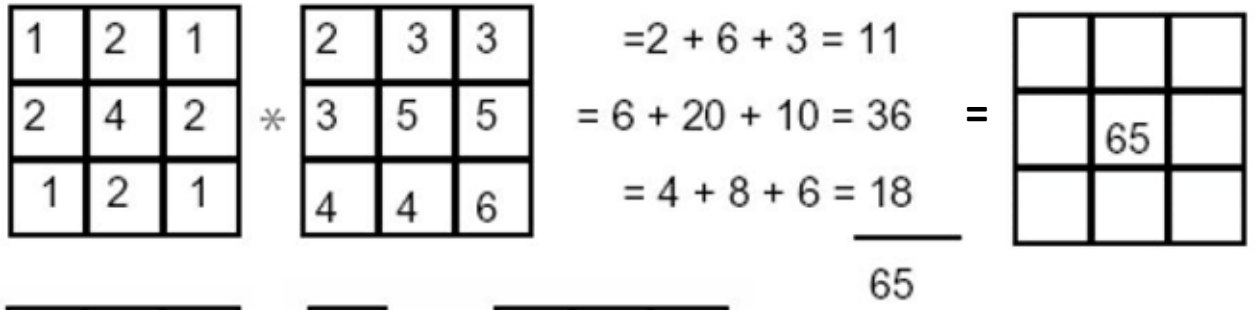


Perform convolution
along rows:

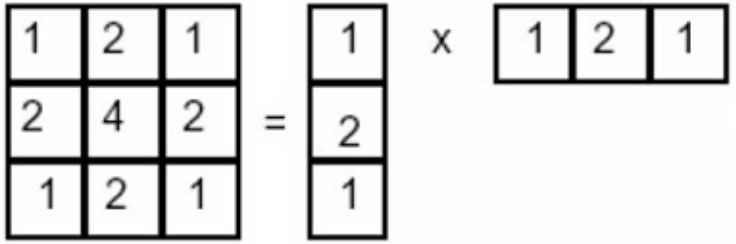


Separability Example

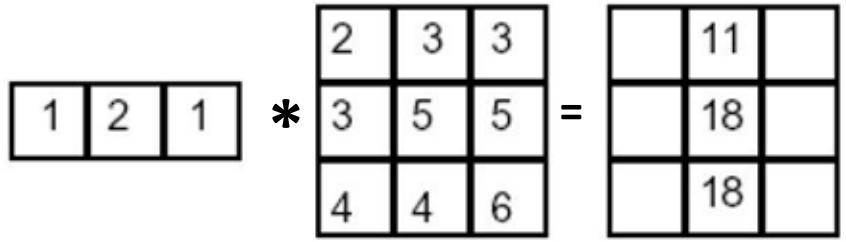
2D convolution
(center location only)



The filter factors into a product of 1D filters:



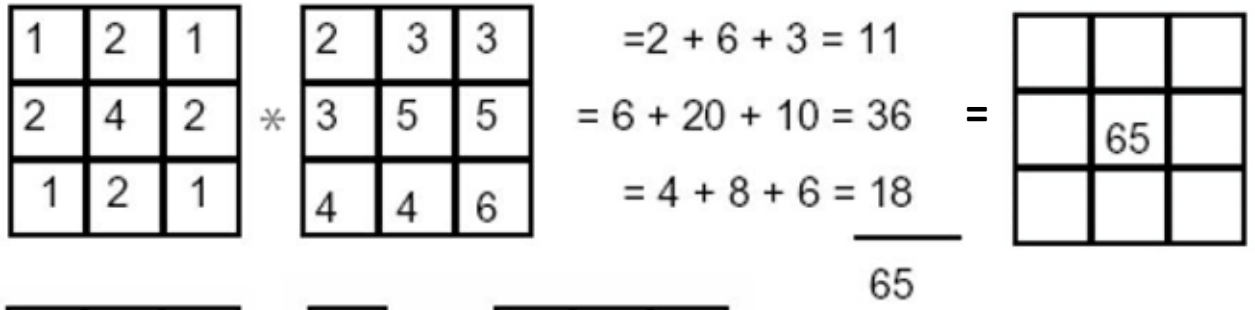
Perform convolution along rows:



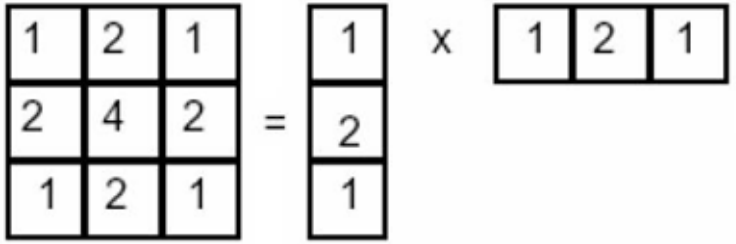
Followed by convolution along the remaining column:

Separability Example

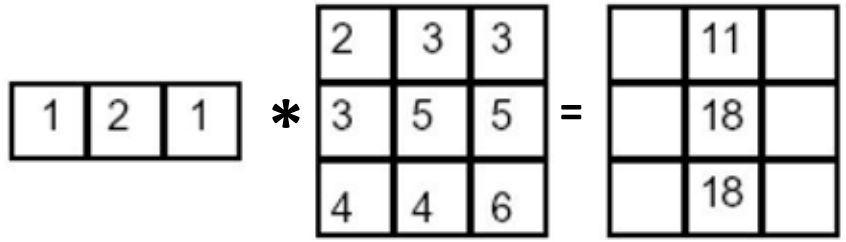
2D convolution
(center location only)



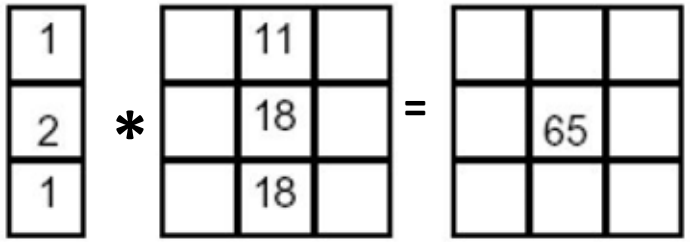
The filter factors
into a product of 1D
filters:



Perform convolution
along rows:



Followed by convolution
along the remaining
column:



Separability Use

MxN image, PxQ filter

- 2D convolution: ~MNPQ multiply-adds
- Separable 2D: ~MN(P+Q) multiply-adds

Separability Use

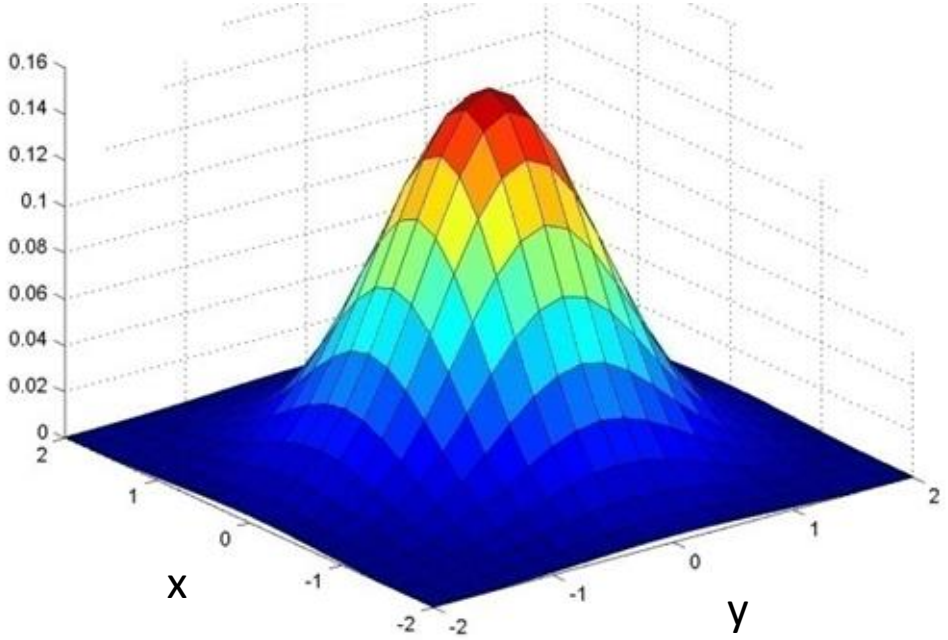
MxN image, PxQ filter

- 2D convolution: ~MNPQ multiply-adds
- Separable 2D: ~MN(P+Q) multiply-adds

Speed up = PQ/(P+Q)

9x9 filter = ~4.5x faster

Gaussian Filter



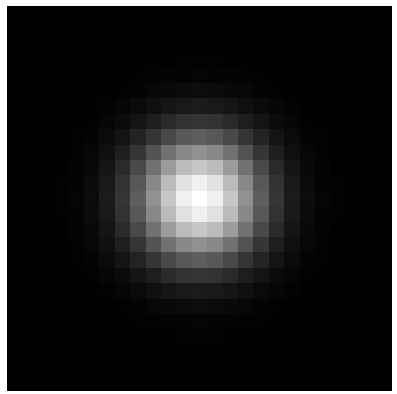
x

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

y

Kernel size 5 x 5,
Standard deviation σ = 1

Viewed from top



y

x

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

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
- Convolving twice with Gaussian kernel of width σ is same as convolving once with kernel of width $\sigma\sqrt{2}$

Separable kernel

- Factors into product of two 1D Gaussians

Separability of the Gaussian Filter

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

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

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

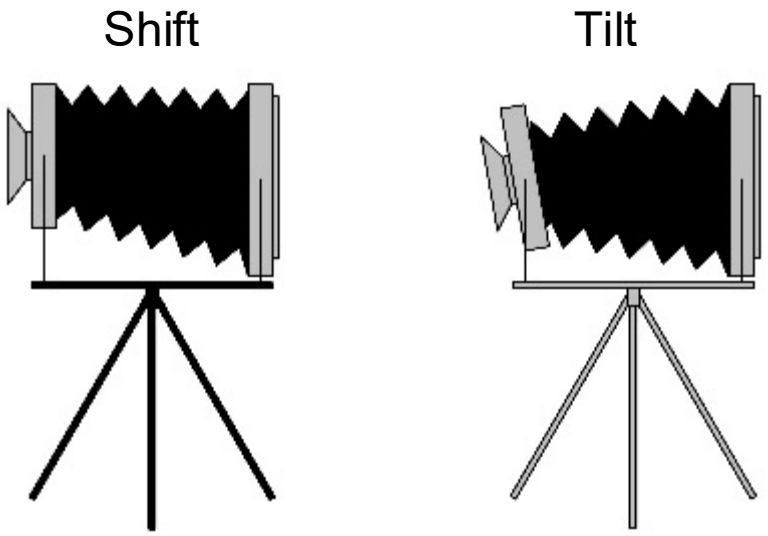
Intermission / Questions

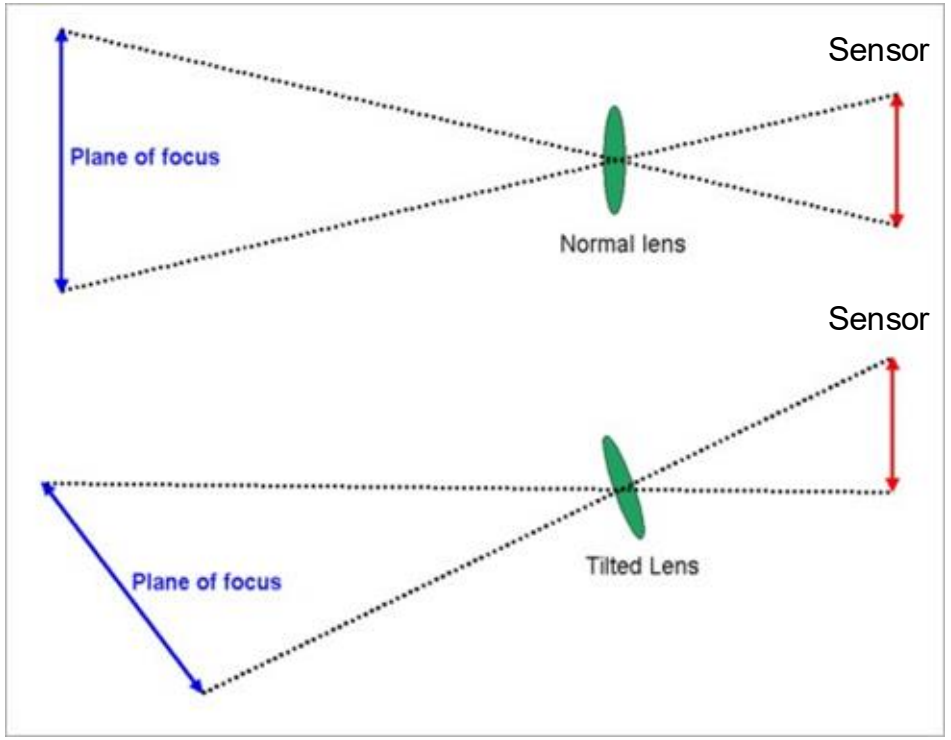
Image Credit: Pixar Animation

Ben Thomas

Tilt-shift photography

Tilt shift camera





Can we fake tilt shift?

We need to blur the image

- OK, now we know how to do that.

Can we fake tilt shift?

We need to blur the image

– OK, now we know how to do that.

We need to blur progressively more away from our ‘fake’ focal plane

But can I make it look more like a toy?

- Boost *saturation* – toys are very colorful
- We'll learn how to do this when we discuss color

Trust in Images

Left: Alexander Malchenko, P. Zaporozhets, and Anatoly Vaneyev (standing); Victor V. Starkov, Gleb Krzhizhanovsky, Vladimir Lenin, and Julius Martov (sitting). **Right: manipulated image** with Malchenko removed. [source]

Trust in Images

“In Event of Moon Disaster”

An MIT art installation to educate the public about deepfakes.

What technical and legal means can we develop and use to preserve **civil trust** on images that are publicly distributed?

What solutions have been proposed and what are potential pitfalls they might have?

Technical Solutions

Coalition for Content Provenance and Authenticity (C2PA)

Alliance between Adobe, ARM, BBC, Intel, Microsoft, Truepic and Twitter to address the prevalence of mis-leading information online through the development of technical stan-dards for certifying the source and history of media content. [source]

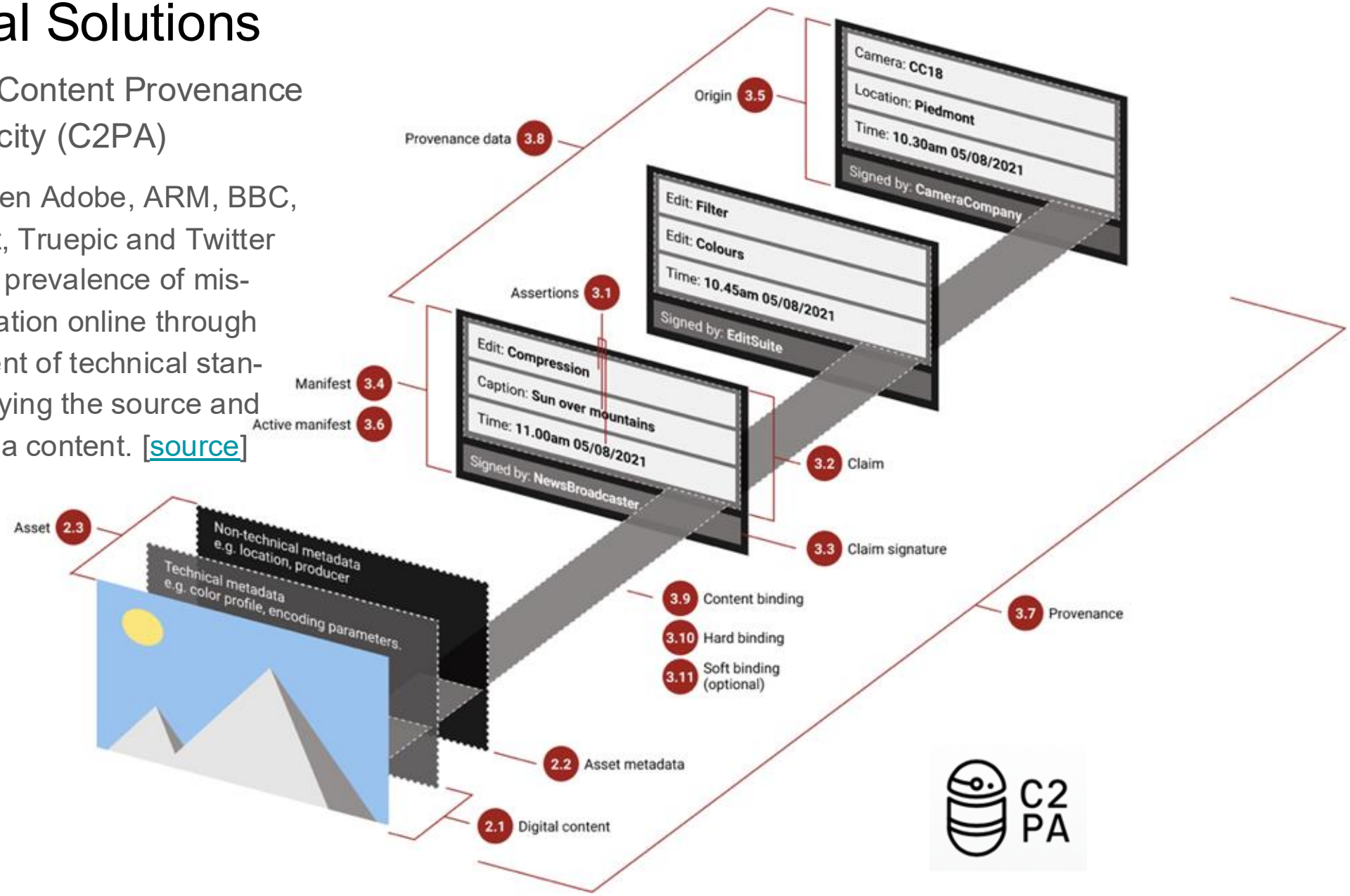
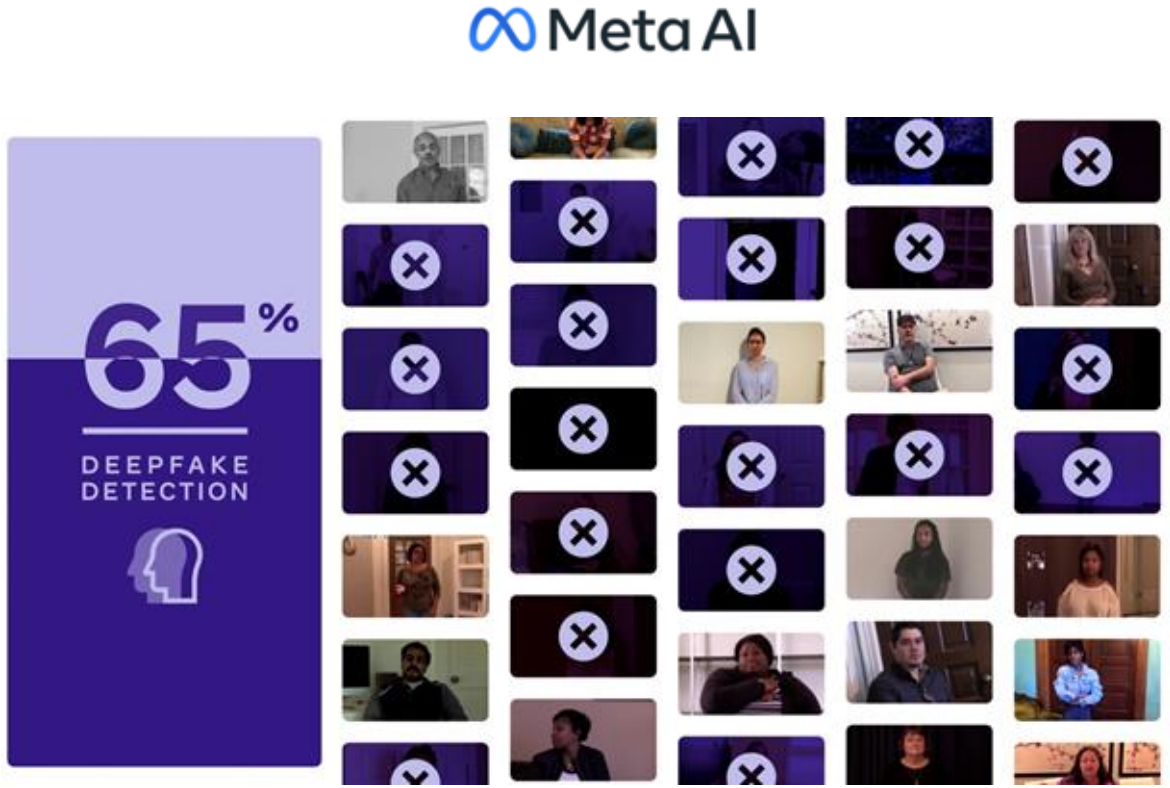


Figure 3. Elements of C2PA C2PA technical specifications

Technical Solutions

Deepfake Detection Challenge (DFDC)

Facebook invested $10m on deepfake detection technology in order to remove any media generated with artificial technology that represent misleading information. [source]



2,000 deepfake detection models submitted

Top model achieved 65.18% accuracy on dataset with real life examples.

Sobel Filter

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

Sobel

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

Sobel Filter

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

Sobel

- What is the 1 2 1 pattern?
  - Binomial distribution or “triangle filter”
  - Discrete approximate to Gaussian; fast for integer mathematics
  - Smooths along edge
- What happens to negative numbers?
- For visualization:
  - Shift image + 0.5
  - If gradients are small, scale edge response

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

Sobel

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

>> plt.imshow( h )
```

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

Sobel

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

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

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

Think-Pair-Share

1) $\_ = D * B$

2) $\bar{A} = \_ * \_$

3) $F = D * \_$

4) $\_ = D * D$

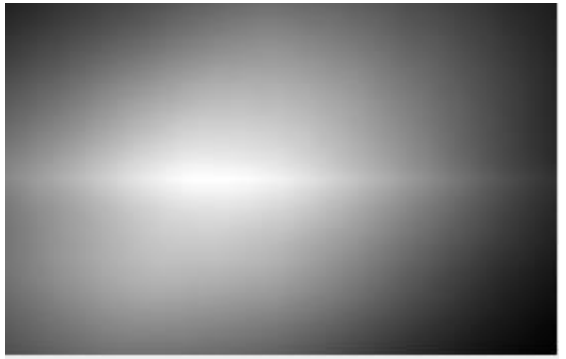
* = Convolution operator

D

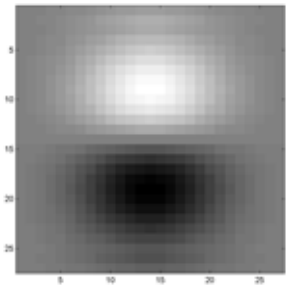
H

F

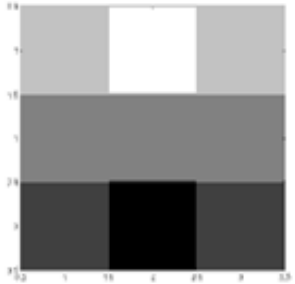
I



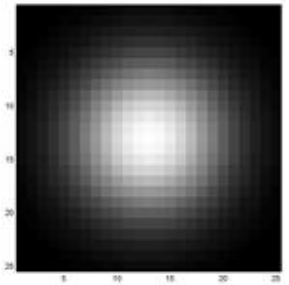
A



B

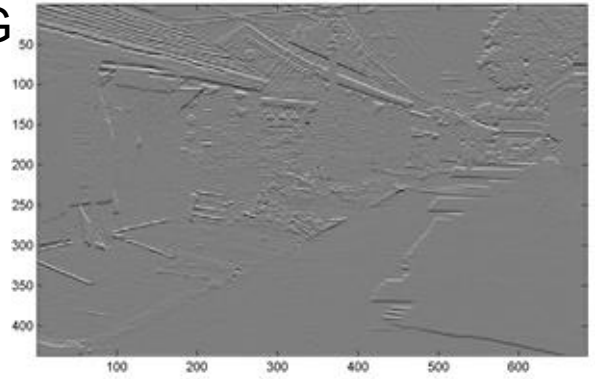


C



E

G



Think-Pair-Share

1) G = D * B
2) A = _ * _
3) F = D * _
4) _ = D * D

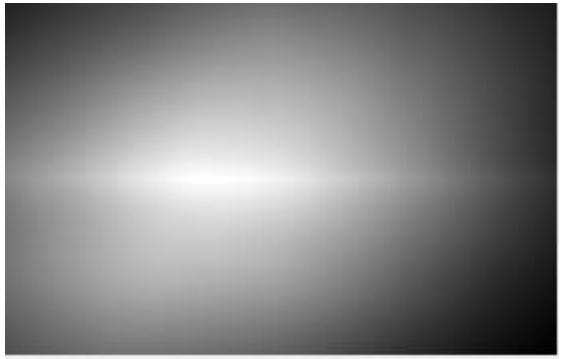
* = Convolution operator

D

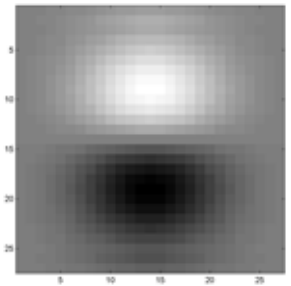
H

F

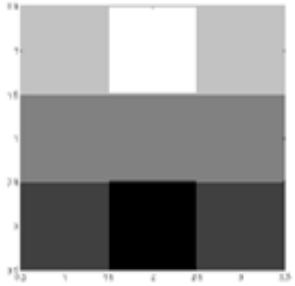
I



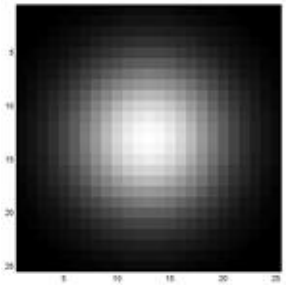
A



B

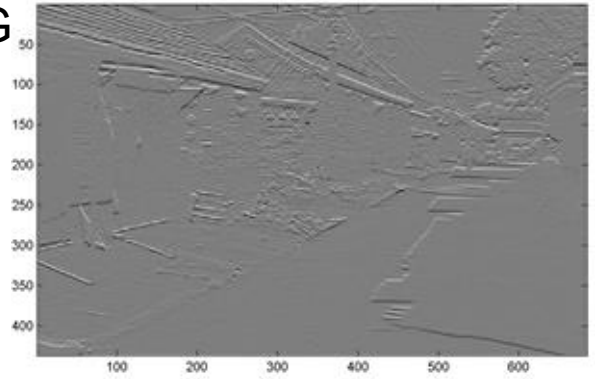


C



E

G



Think-Pair-Share

1) G = D * B
2) A = B * C
3) F = D * $\overline{\_}$
4) $\overline{\_}$ = D * $\overline{D}$

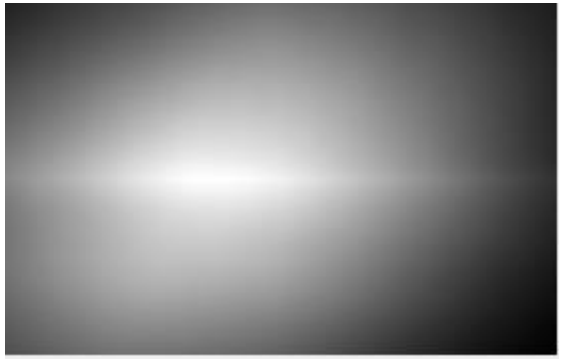
* = Convolution operator

D

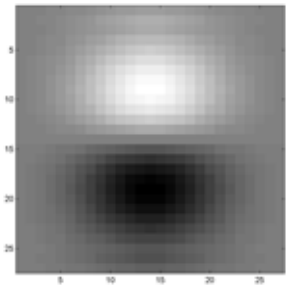
H

F

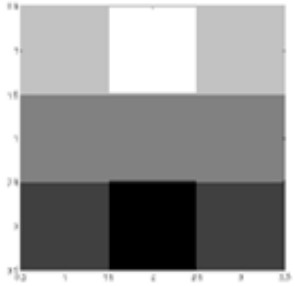
I



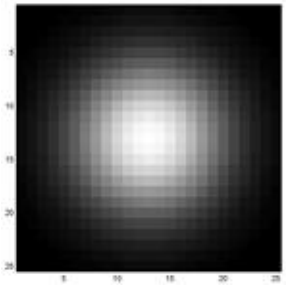
A



B

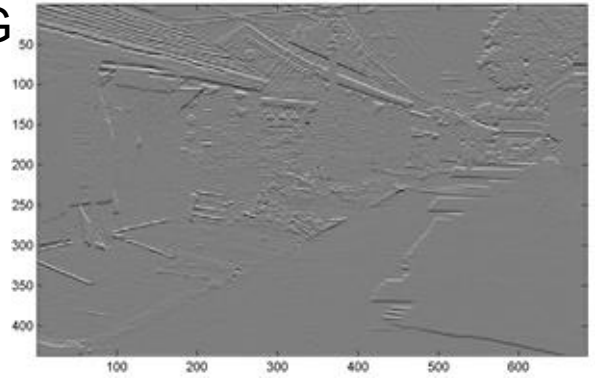


C



E

G



Think-Pair-Share

1) G = D * B
2) A = B * C
3) F = D * E
4) _ = D * D

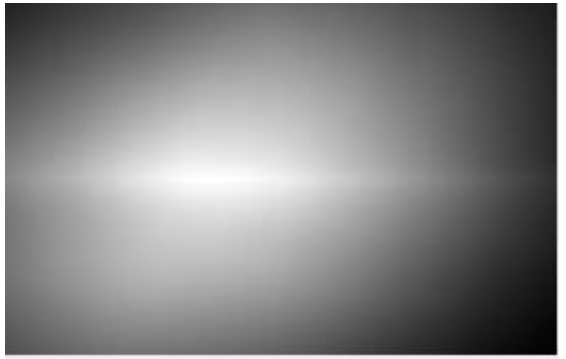
* = Convolution operator

D

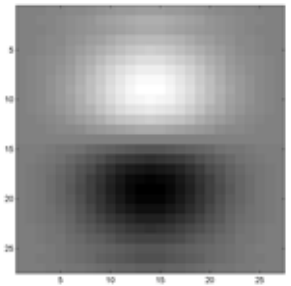
H

F

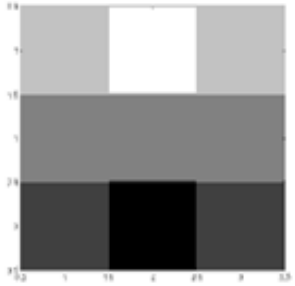
I



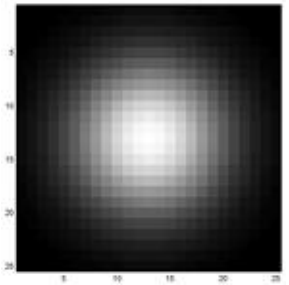
A



B

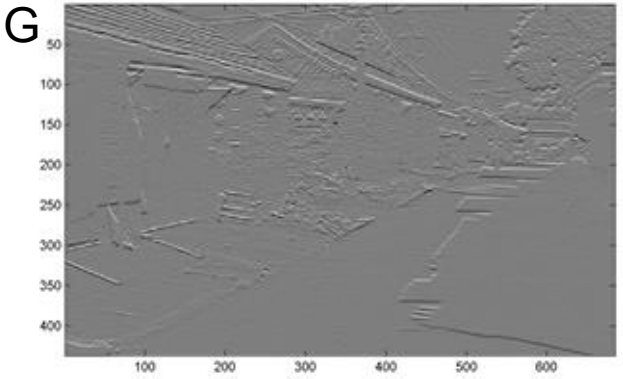


C



E

G



Think-Pair-Share

1) G = D * B
2) A = B * C
3) F = D * E
4) I = D * D

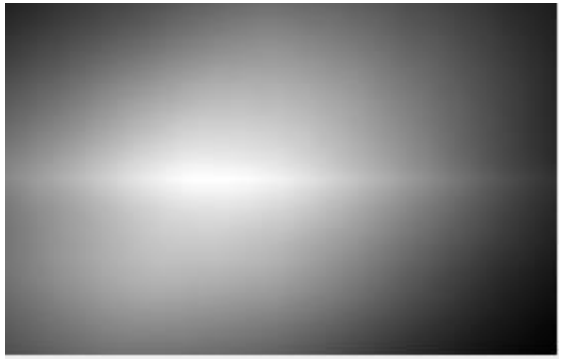
* = Convolution operator

D

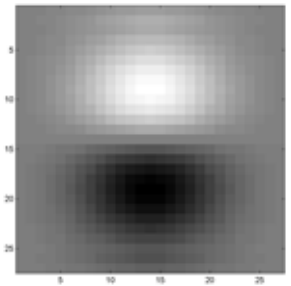
H

F

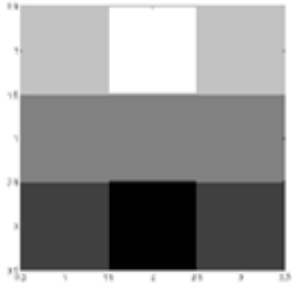
I



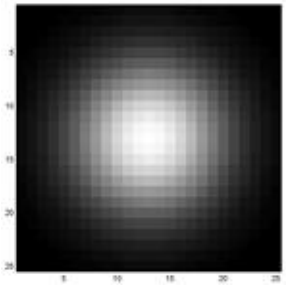
A



B

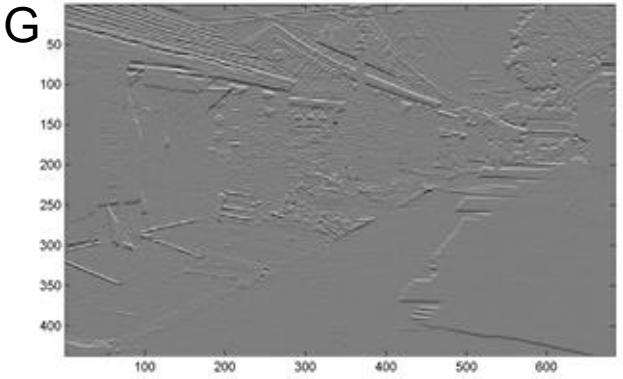


C



E

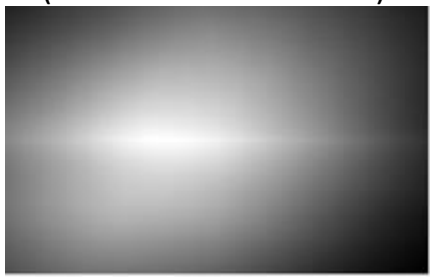
G



I = D * D

D (275 x 175 pixels)

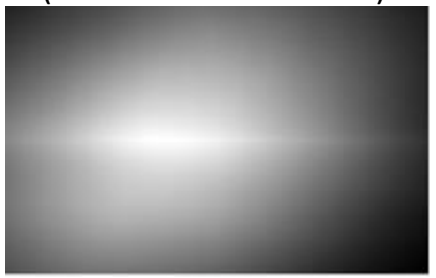
I (from slide – 275 x 175)



I = D * D

D (275 x 175 pixels)

I (from slide – 275 x 175)

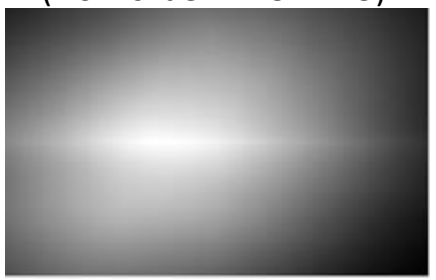


"…something to do with lack of content (black) at edges…"

I = D * D

D (275 x 175 pixels)

I (from slide – 275 x 175)



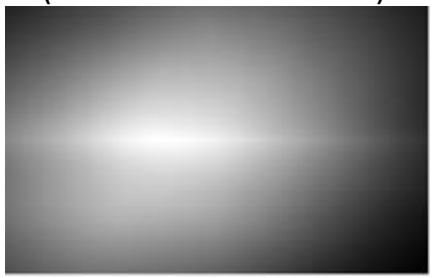
```
>> D = img_to_float32( io.imread( ‘convexample.png’ ) )
>> I = convolve2d( D, D )
>> np.max(I)
1.1021e+04

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

I = D * D

D (275 x 175 pixels)

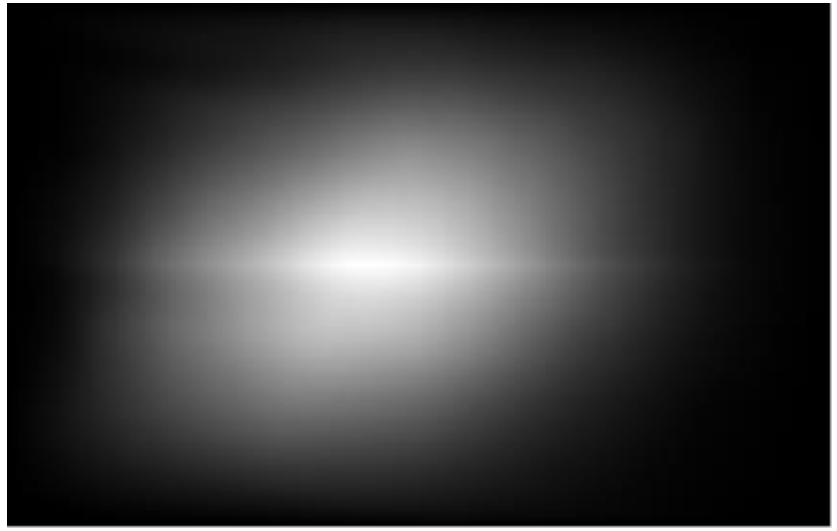
I (from slide – 275 x 175)



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

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

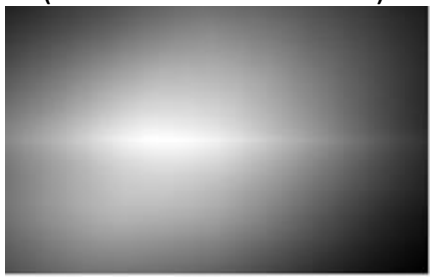
I_norm



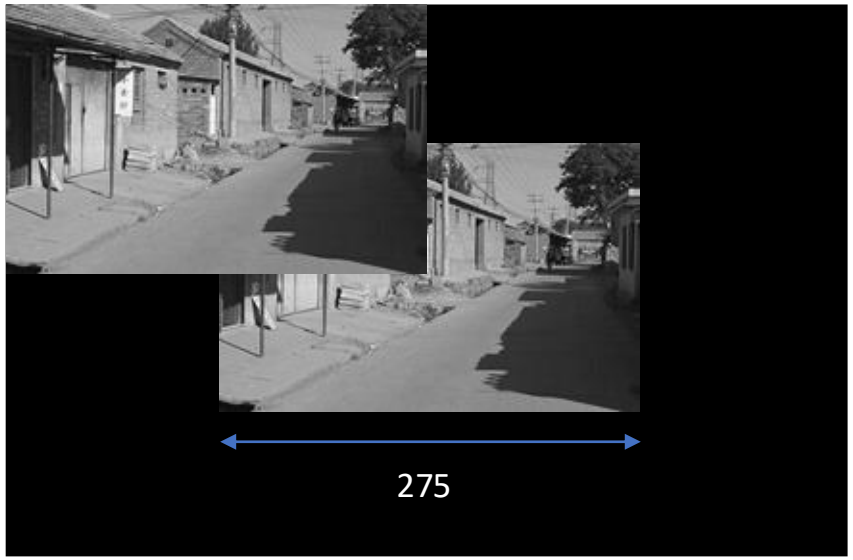
I = D * D

D (275 x 175 pixels)

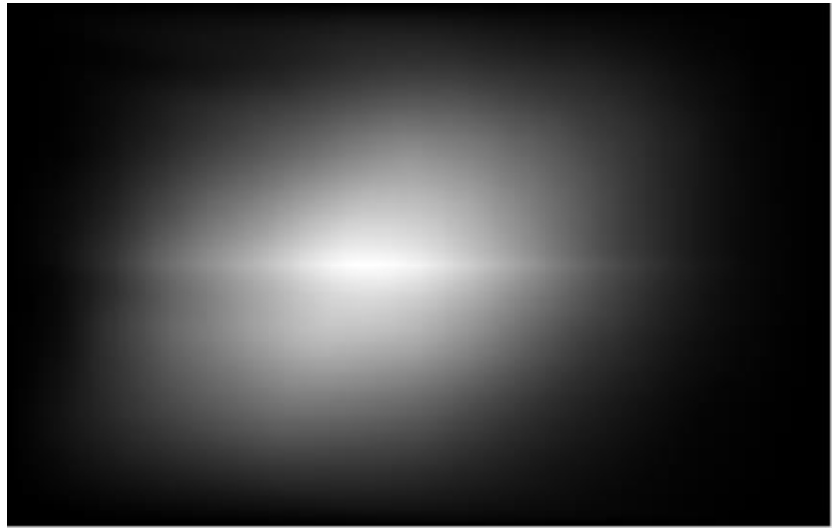
I (from slide – 275 x 175)



(275-1)/2



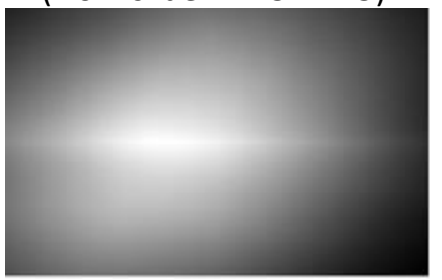
I_norm



I = D * D

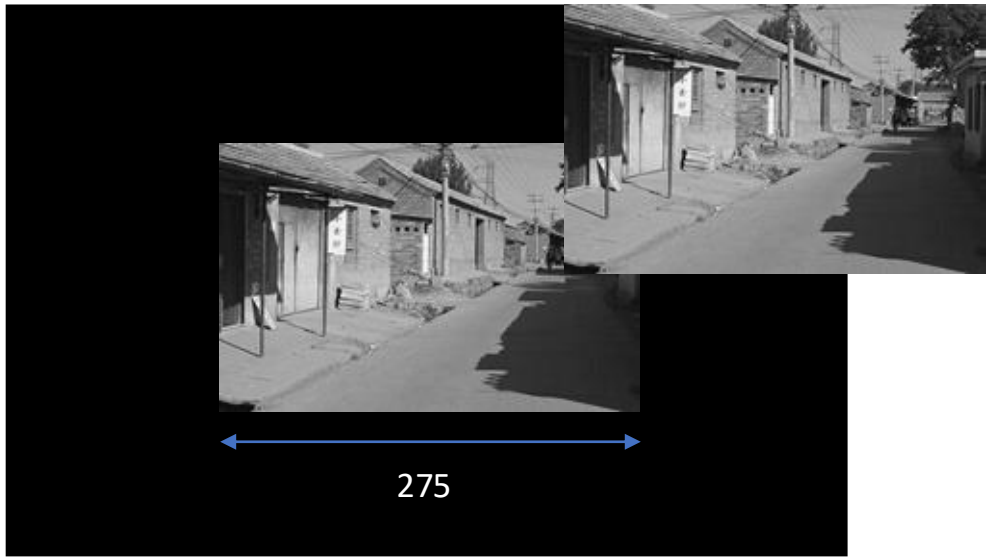
D (275 x 175 pixels)

I (from slide – 275 x 175)

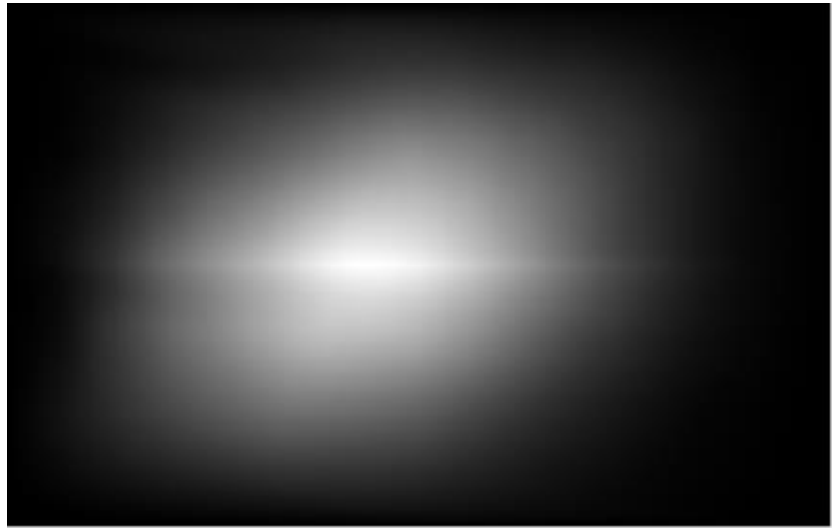


(275-1)/2

(275-1)/2



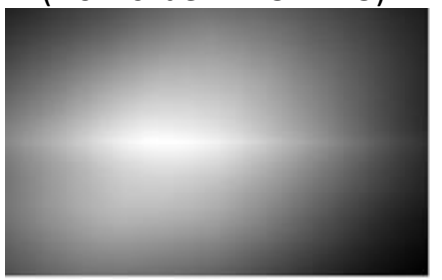
I_norm



I = D * D

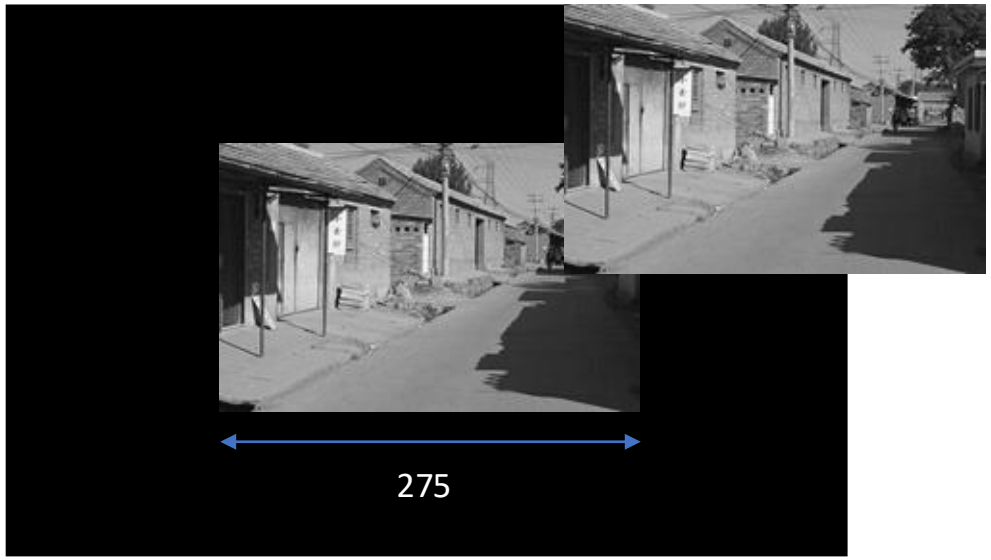
D (275 x 175 pixels)

I (from slide – 275 x 175)



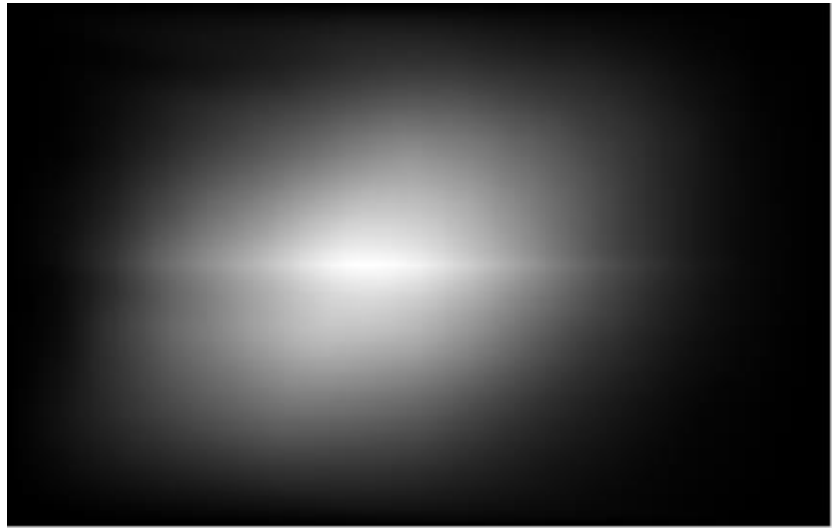
(275-1)/2

(275-1)/2



For x: 275 + (275-1)/2 + (275-1)/2
= 549

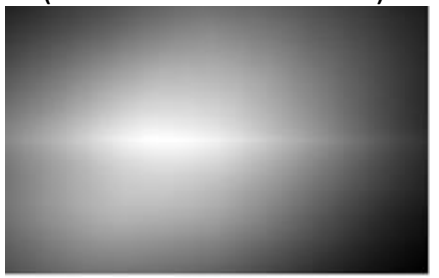
I_norm



I = D * D

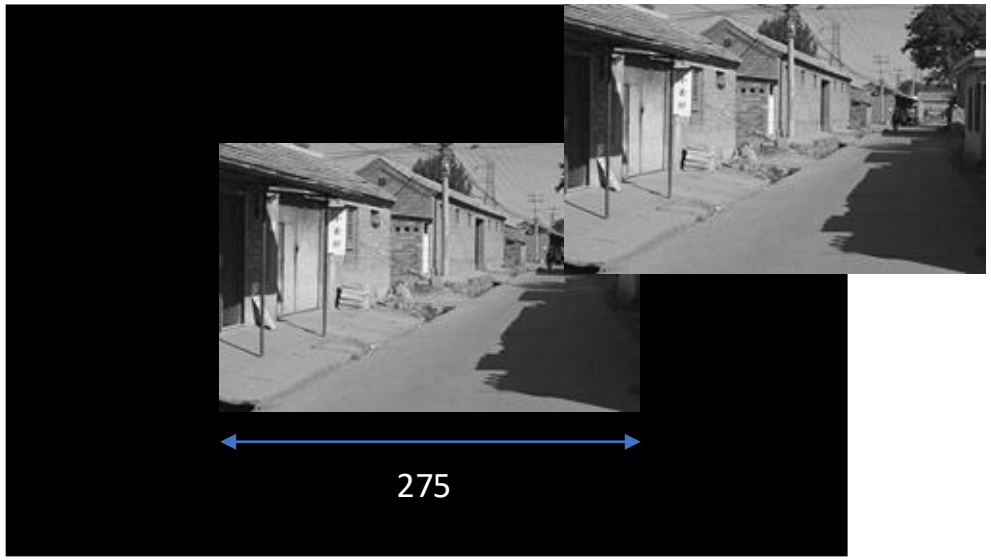
D (275 x 175 pixels)

I (from slide – 275 x 175)



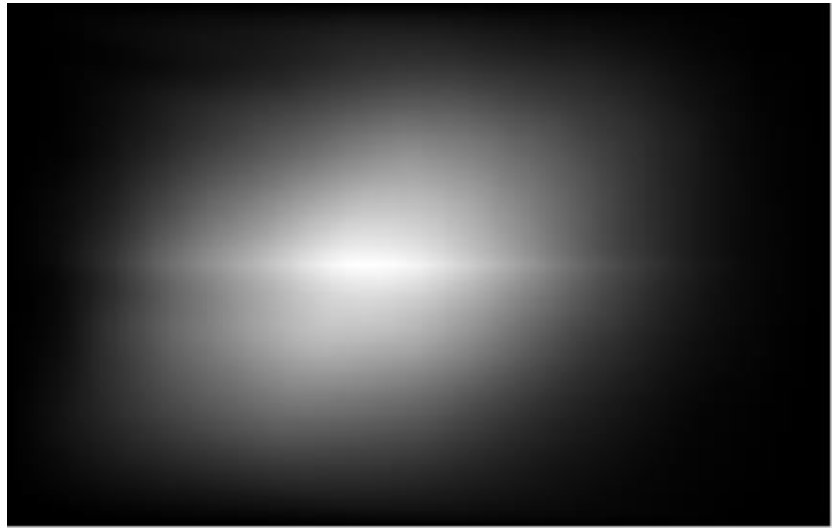
(275-1)/2

(275-1)/2



For x: 275 + (275-1)/2 + (275-1)/2
= 549

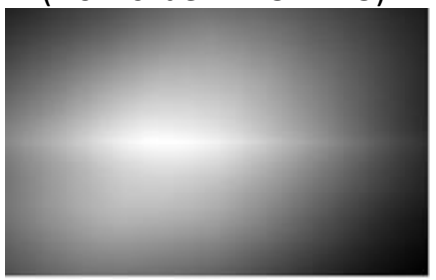
I_norm (549 x 349 pixels)



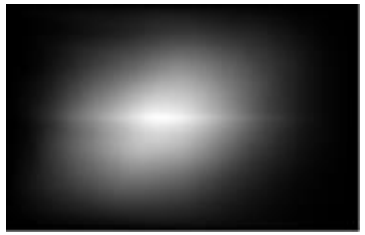
I = D * D

D (275 x 175 pixels)

I (from slide – 275 x 175)



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

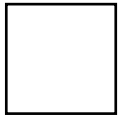


549 x 349

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

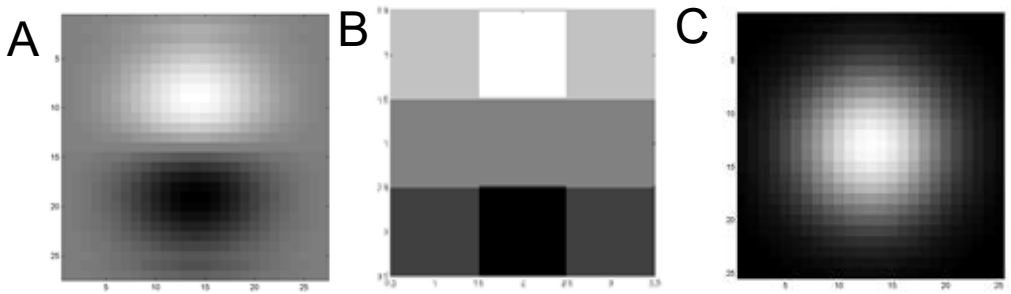
275 x 175

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



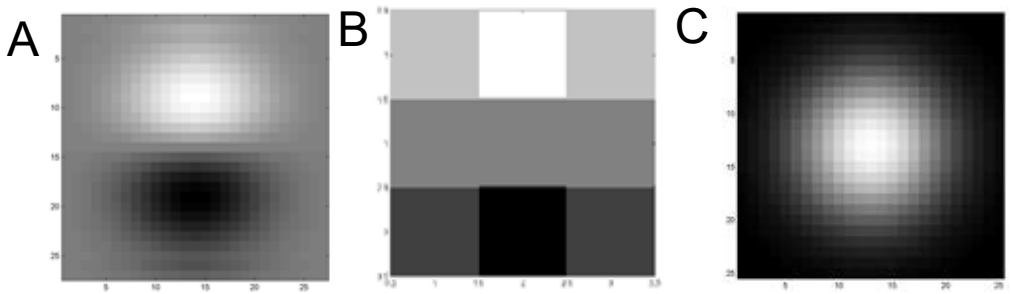
Value = 10528.3
1x1

A = B * C



When the filter ‘looks like’ the image = ‘template matching’

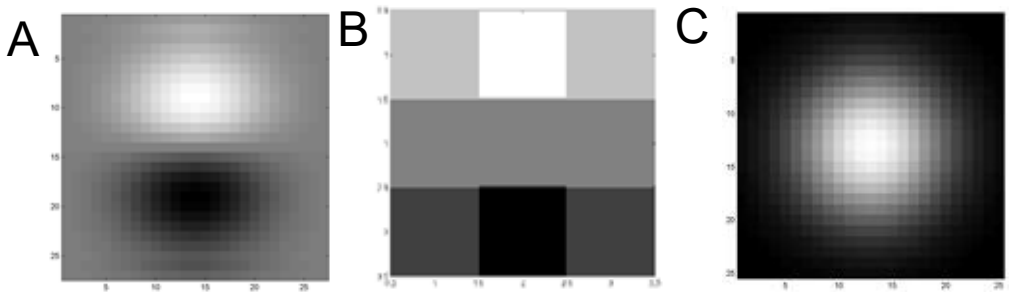
A = B * C



C is a Gaussian kernel and we know that it ‘blurs’.

When the filter ‘looks like’ the image = ‘template matching’

A = B * C

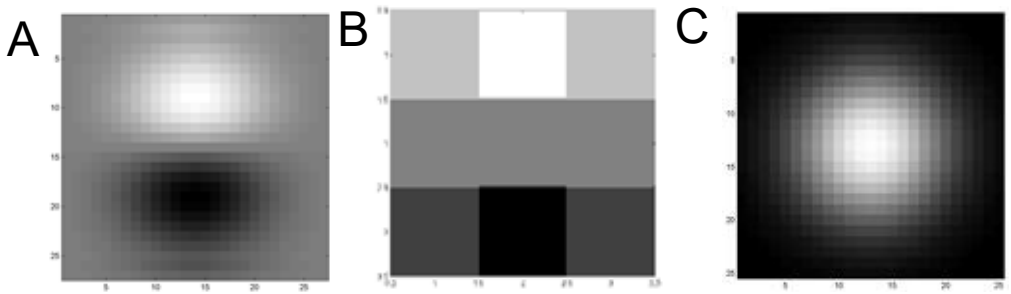


C is a Gaussian kernel and we know that it ‘blurs’.

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.

A = B * C



C is a Gaussian kernel and we know that it ‘blurs’.

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.

Let’s see if we can use correlation to ‘find’ the parts of the image that look like the kernel.

D (275 x 175 pixels)

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

D (275 x 175 pixels)

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

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

D (275 x 175 pixels)

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

f
61 x 61

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

D (275 x 175 pixels)

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

Expect response 'peak' in middle of I

f
61 x 61

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

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' )
```

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

D (275 x 175 pixels)

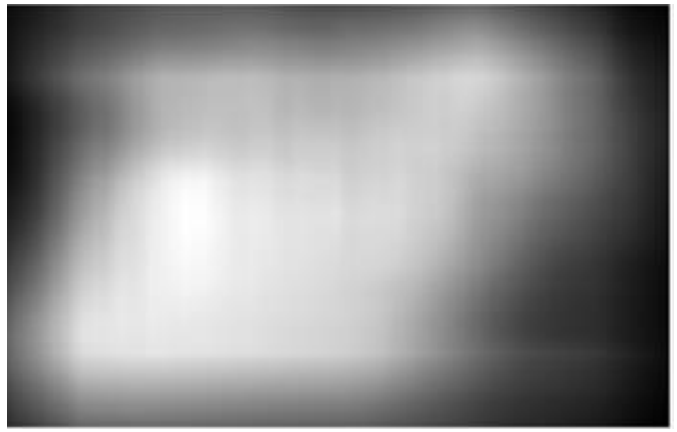
```
>> f = D[ 57:117, 107:167 ]
```

Expect response 'peak' in middle of I

f
61 x 61

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

I



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

D (275 x 175 pixels)

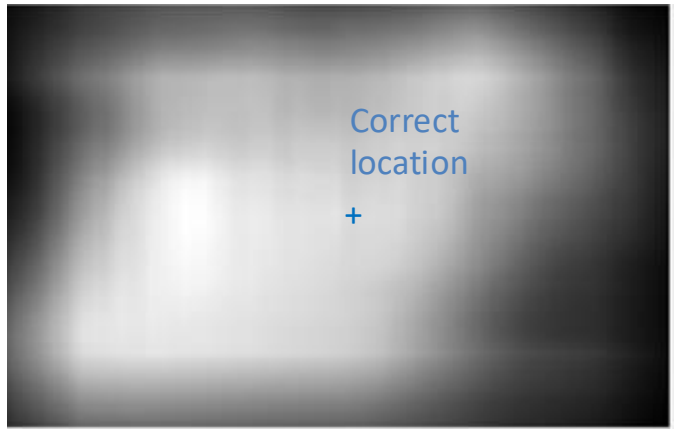
```
>> f = D[ 57:117, 107:167 ]
```

Expect response 'peak' in middle of I

f
61 x 61

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

I



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

D (275 x 175 pixels)

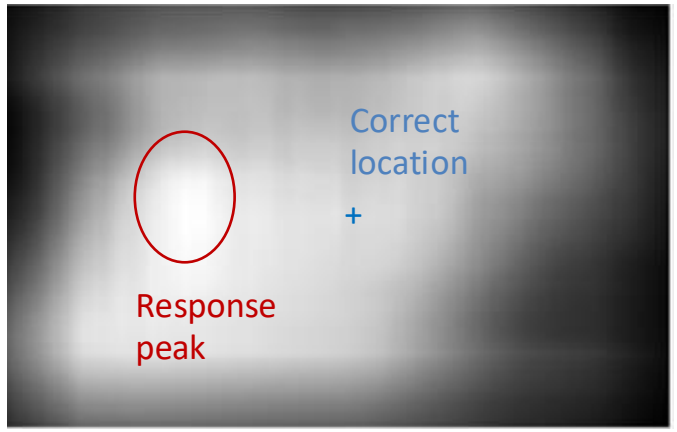
```
>> f = D[ 57:117, 107:167 ]
```

Expect response 'peak' in middle of I

f
61 x 61

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

I



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

D (275 x 175 pixels)

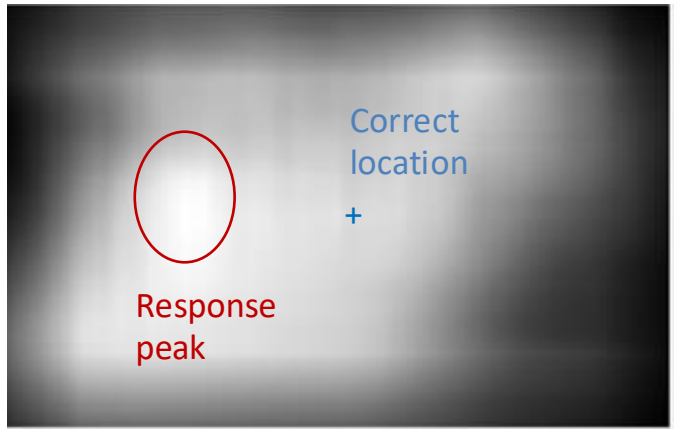
```
>> f = D[ 57:117, 107:167 ]
```

Expect response 'peak' in middle of I

f
61 x 61

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

I



Hmm…
That didn't work – why not?

Correlation

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

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

Correlation

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

```
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.

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

OK, so let’s subtract the mean

OK, so let's subtract the mean

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

OK, so let's subtract the mean

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

OK, so let's subtract the mean

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

f2
61 x 61

OK, so let's subtract the mean

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

f2
61 x 61

OK, so let's subtract the mean

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

D2 (275 x 175 pixels)

f2
61 x 61

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.

D2 (275 x 175 pixels)

f2
61 x 61

OK, so let's subtract the mean

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

D2 (275 x 175 pixels)

f2
61 x 61

Now zero centered.
Score is higher only when dark parts match and when light parts match.

```
>> I2 = correlate2d( D2, f2, ‘same’ )
```

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

I2

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!

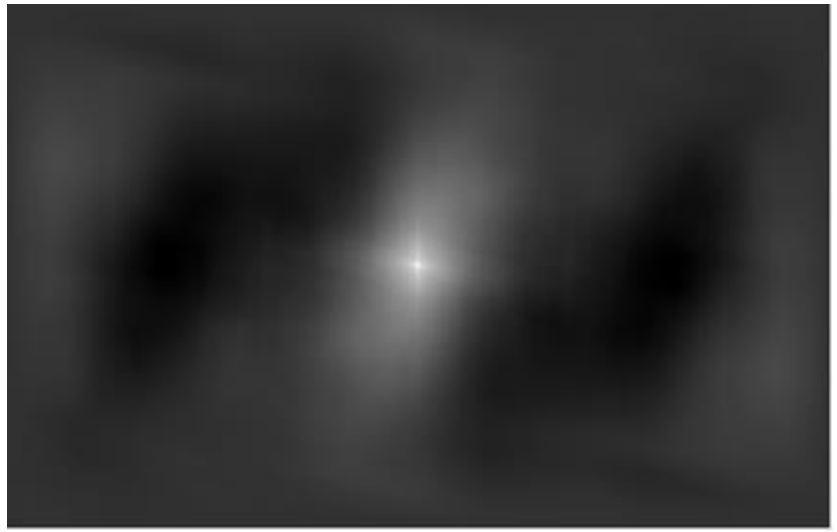
I2

```
>> I3 = correlate2d( D2, D2, ‘full’ )
```

Or our original example: D2 correlated with itself

```
>> I3 = correlate2d( D2, D2, ‘full’ )
```

I3

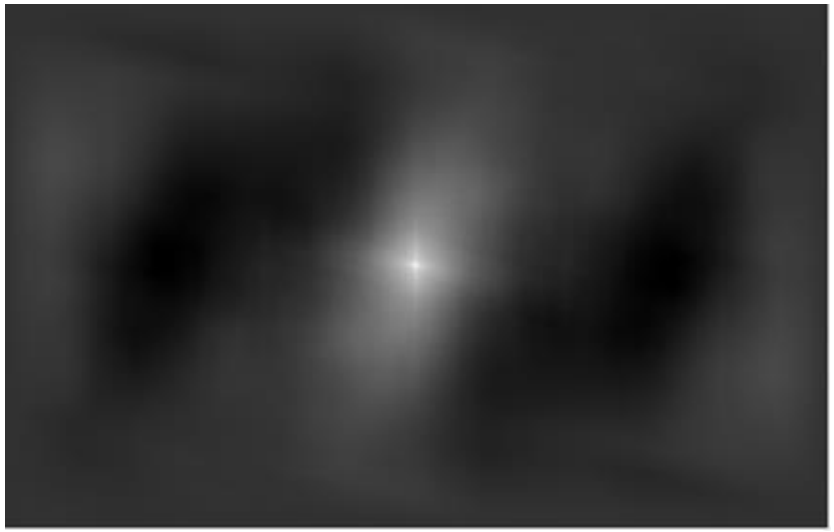


Or our original example: D2 correlated with itself

D2 (275 x 175 pixels)

```
>> I3 = correlate2d( D2, D2, ‘full’ )
```

I3



Or our original example: D2 correlated with itself

What happens with convolution?

What happens with convolution?

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

What happens with convolution?

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

What happens with convolution?

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

f2
61 x 61

What happens with convolution?

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

f2
61 x 61

What happens with convolution?

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

D2 (275 x 175 pixels)

f2
61 x 61

What happens with convolution?

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

>> I2 = convolve2d( D2, f2, ‘same’ )
```

D2 (275 x 175 pixels)

f2
61 x 61

What happens with convolution?

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

>> I2 = convolve2d( D2, f2, ‘same’ )
```

D2 (275 x 175 pixels)

f2
61 x 61

I2

CS322

Non-linear Filters

Image Credit: Pixar Animation

Non-Linear Filter: Median Filter

- Operates over a window by selecting the median intensity in the window.

- ‘Rank’ filter as based on ordering of gray levels
  - E.G., min, max, range filters

Median Filter

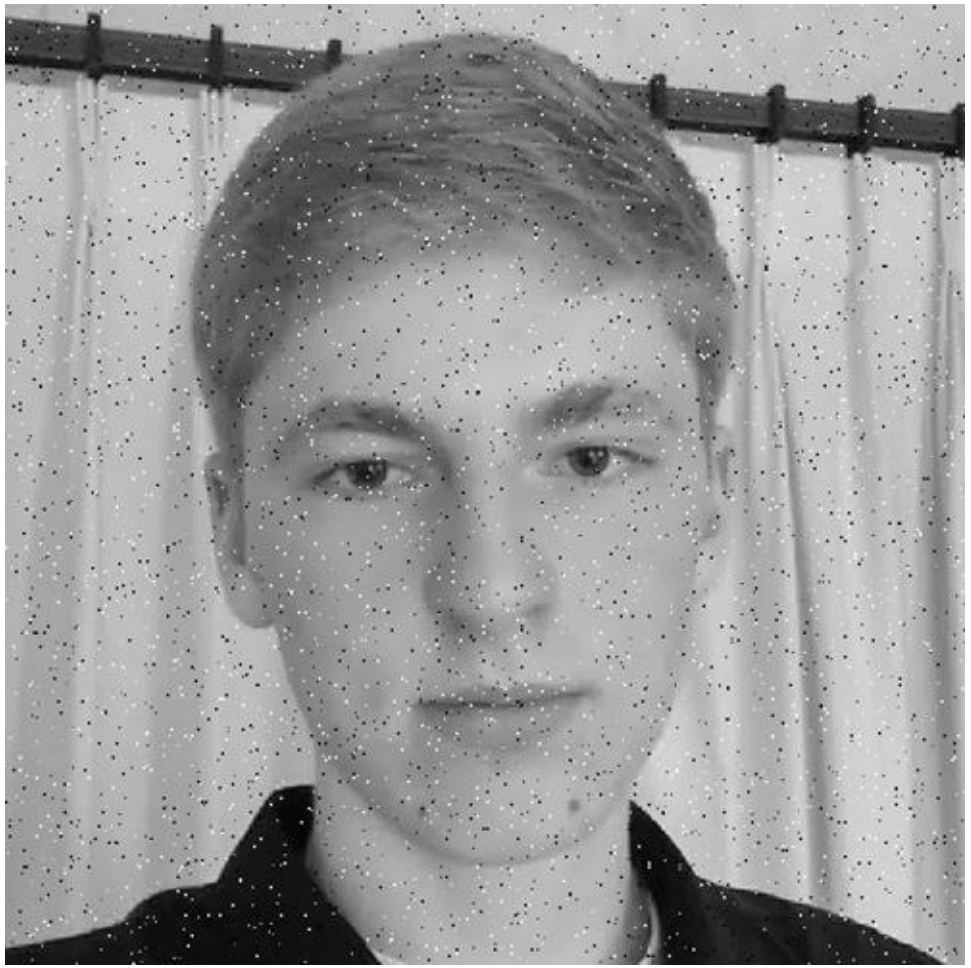
$I[.,.]$

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	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	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	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

				?					

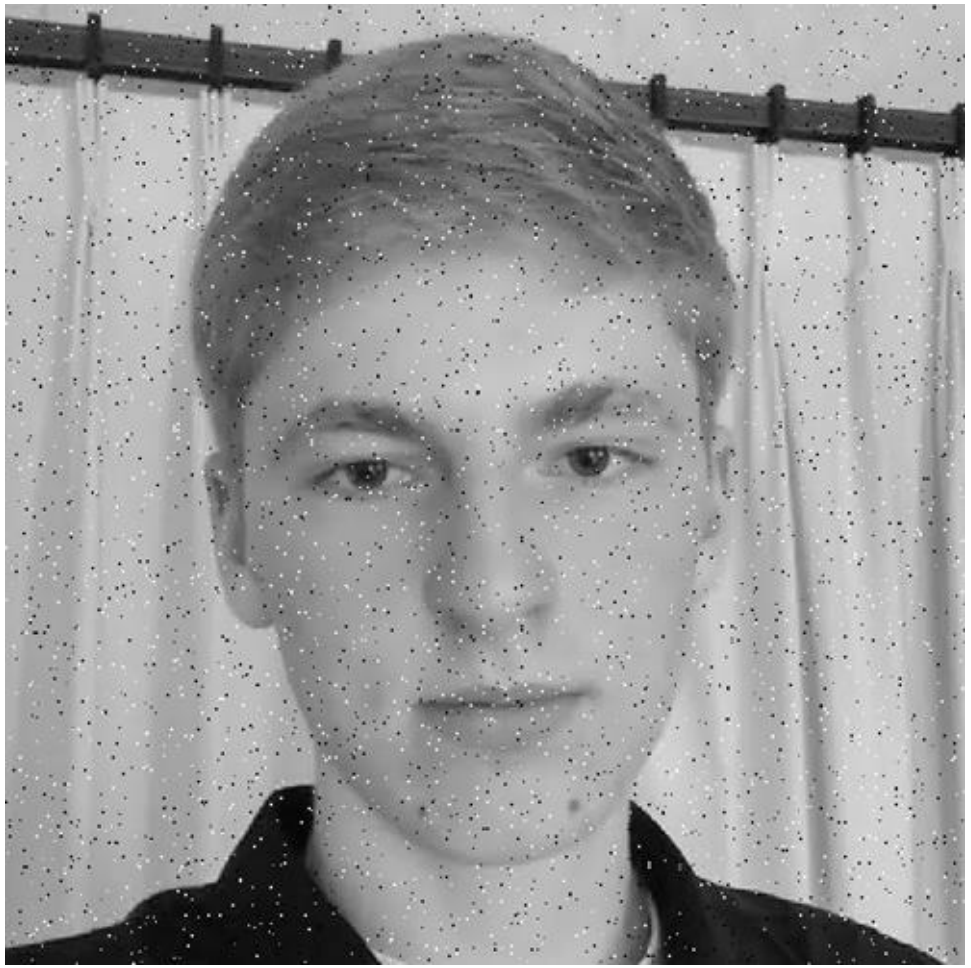
Salt and Pepper Noise



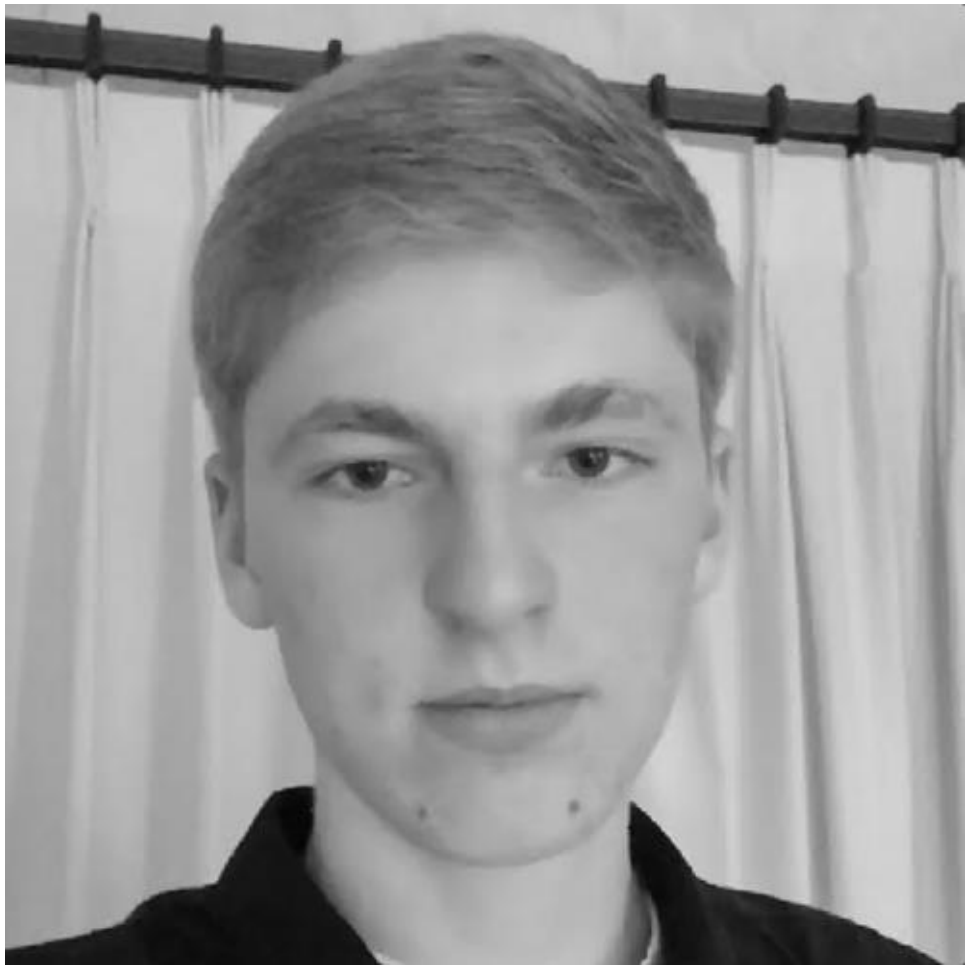
3 x 3 Mean Filter

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.

CS322

Sampling

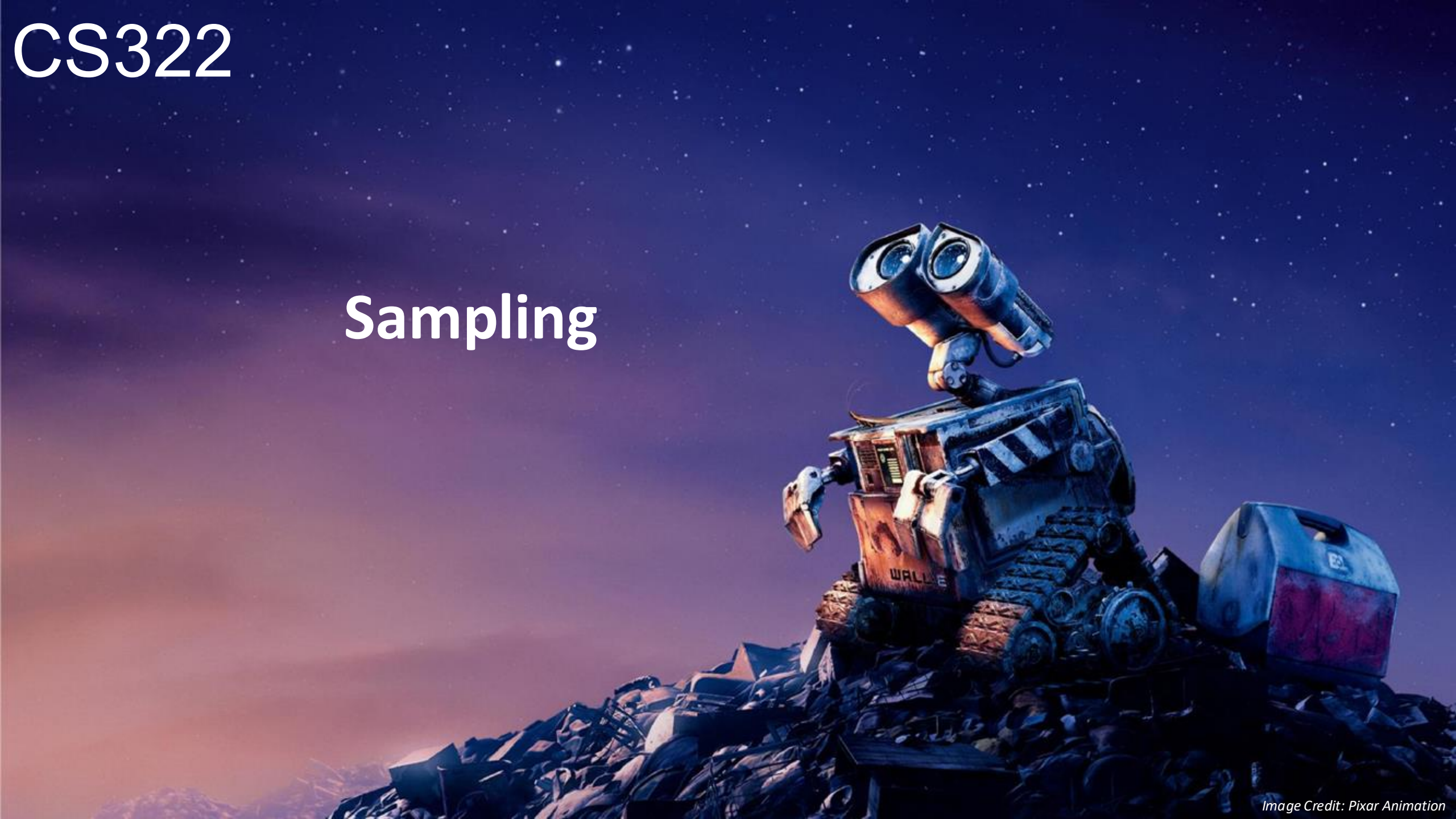


Image Credit: Pixar Animation

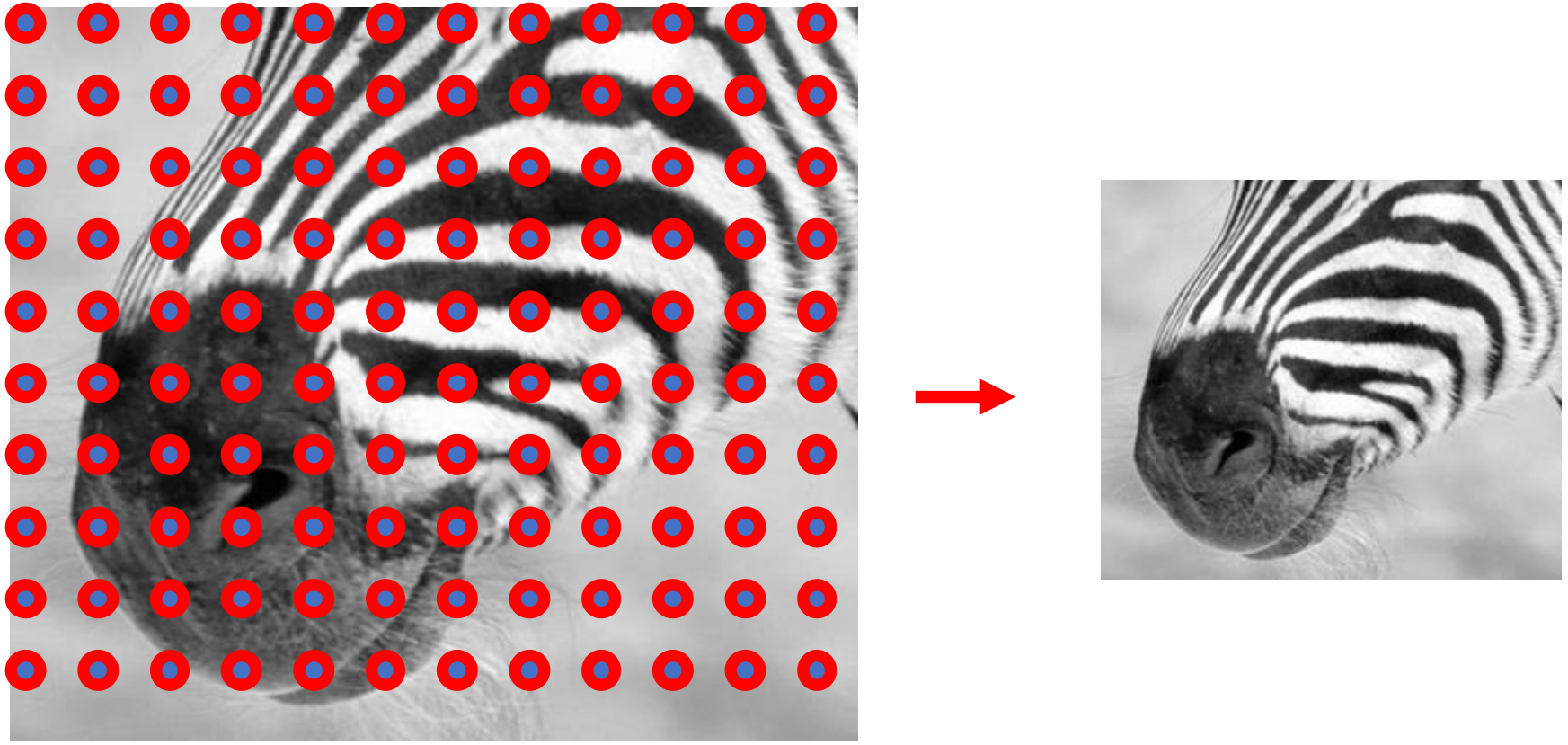
Why does a lower resolution image still make sense to us?
What information do we lose?

Slide: Derek Hoiem, Image: http://www.flickr.com/photos/igorms/136916757/

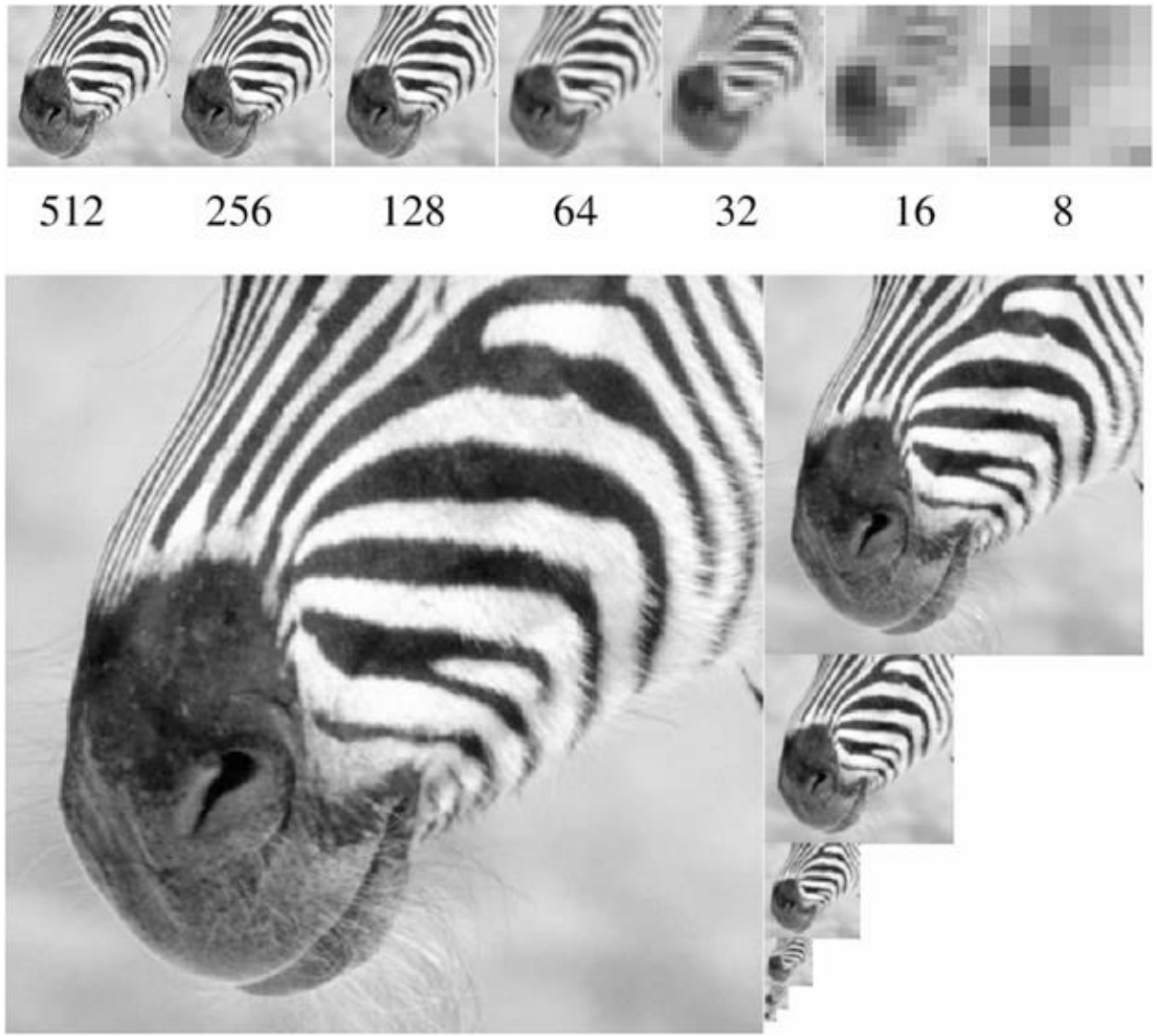
Making lower resolution images: Subsampling by a factor of 2

Throw away every other row and column to create a 1/2 size image

Making lower resolution images: Subsampling by a factor of 2



Throw away every other row and column to create a 1/2 size image



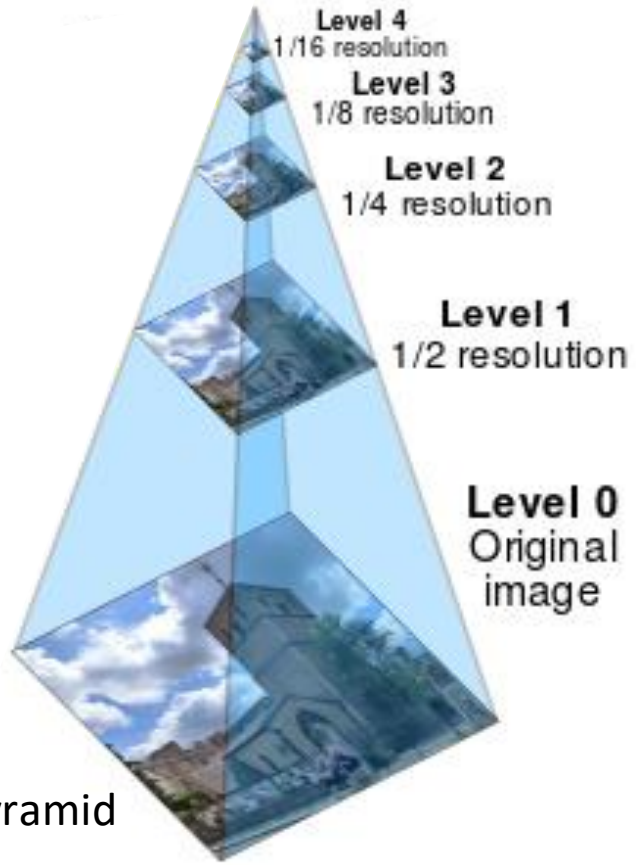


Image Pyramid

Downsampling

Algorithm

1. Start with image of w x h
2. Sample every other pixel
3. `im_small = image[ ::w, ::w ]`

1. To build a pyramid,
   repeat Steps 1 & 2
   until `im_small` is 1 pixel large.

Downsampling

Algorithm

1. Start with image of w x h
2. Sample every other pixel
3. `im_small = image[ ::w, ::w ]`

1. To build a pyramid,
   repeat Steps 1 & 2
   until `im_small` is 1 pixel large.

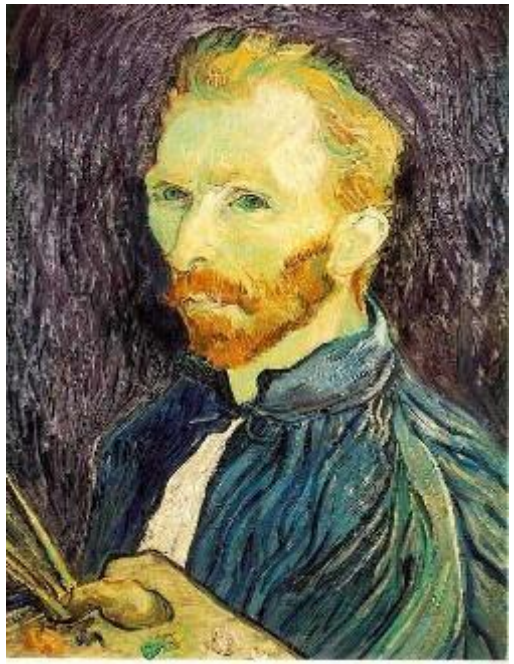
NumPy Syntax

w = 2
::2 -> start at 0,
end at 'end',
increase every 2,
until the end.

e.g.,
0,2,4,6,…,w

(if w is not even, then
this goes to w-1)

Image Subsampling

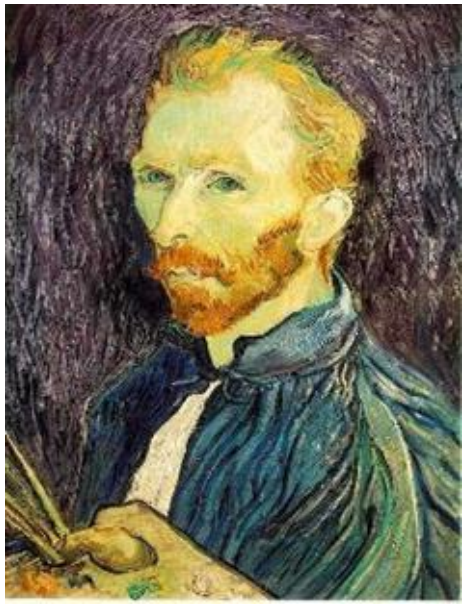


Throw away every other row and column to create a 1/2 size image.

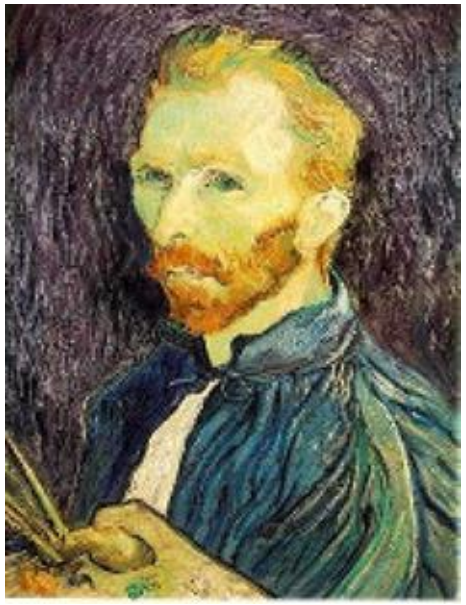
1/4

1/8

Subsampling without Filtering



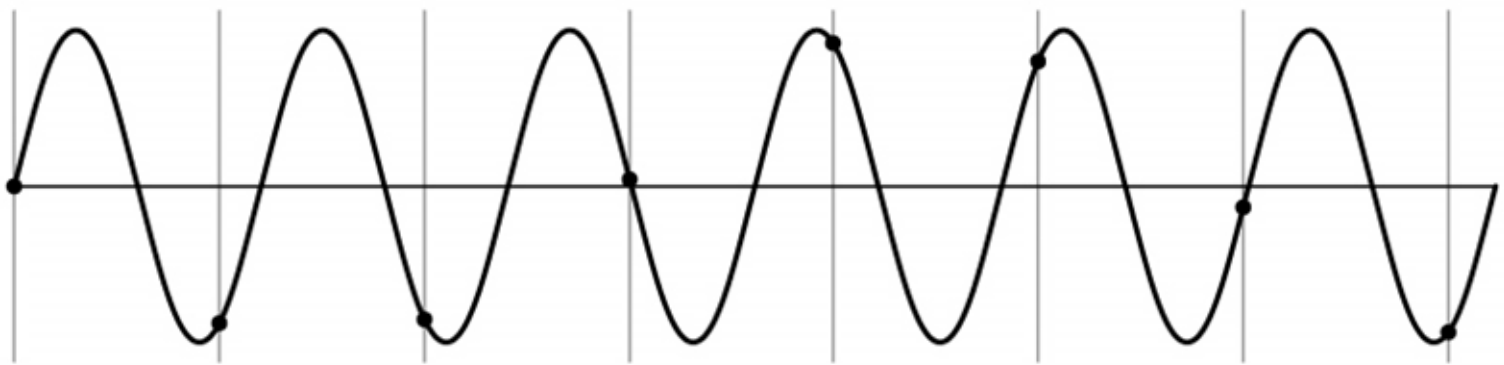
1/2



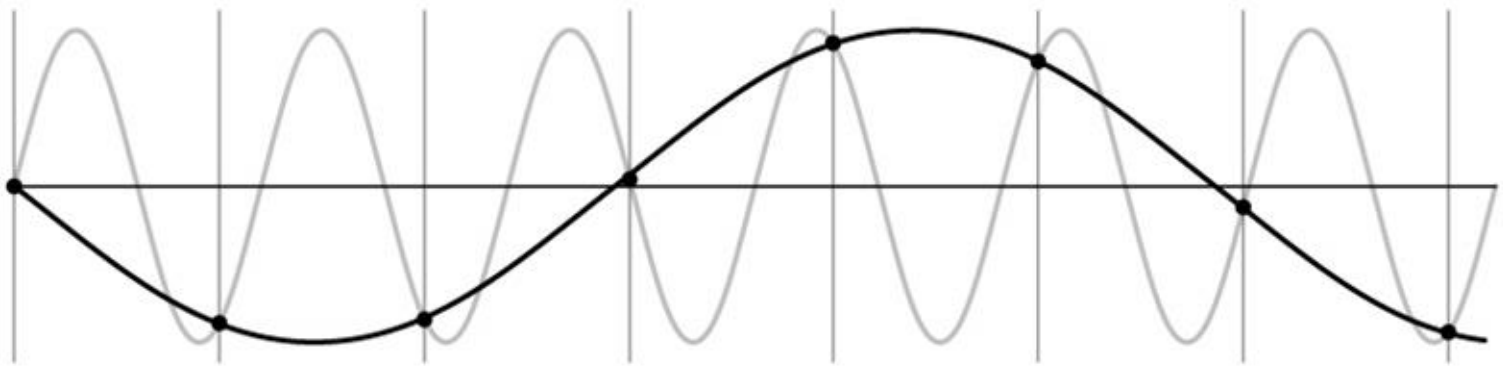
1/4 (2x subsample)

1/8 (4x subsample)

Aliasing

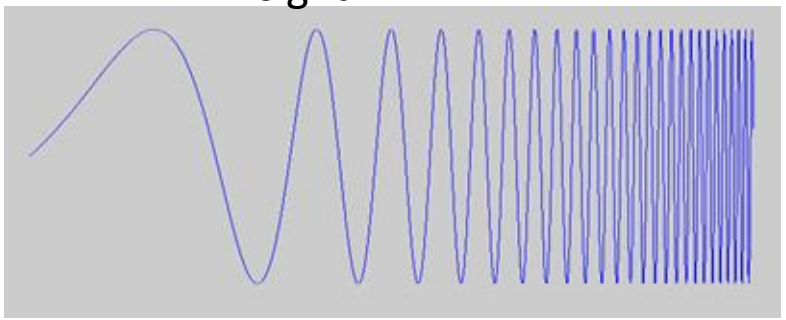


Aliasing



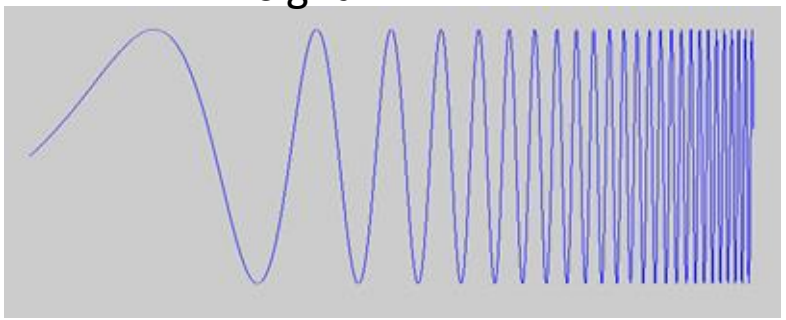
What's happening?

Input signal:

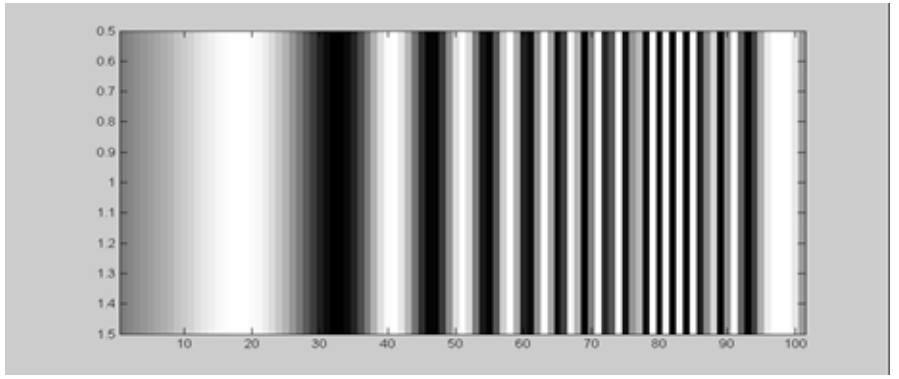


What's happening?

Input signal:



Plot as image:

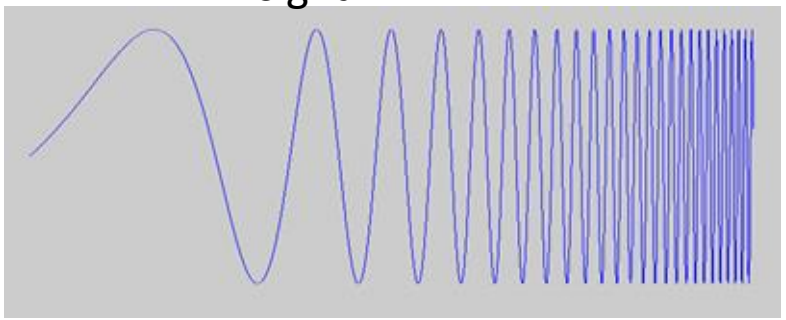



```
x = 0:.05:5;  plt.imshow(sin((2.^x).*x))
```

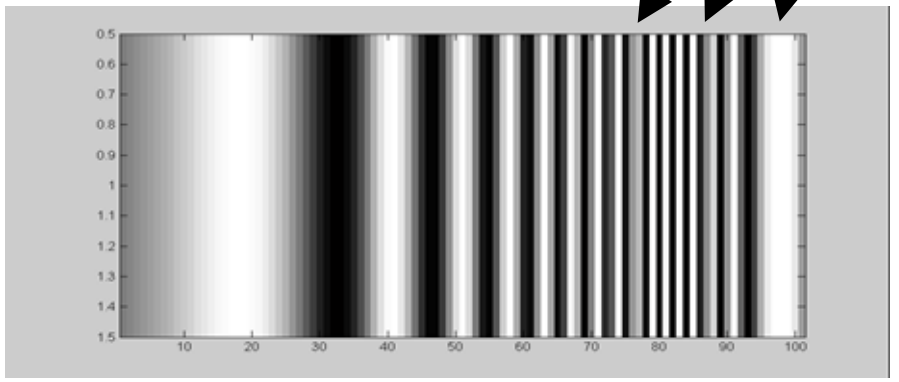
What's happening?

Aliasing!
Not enough
samples

Input
signal:



Plot as image:

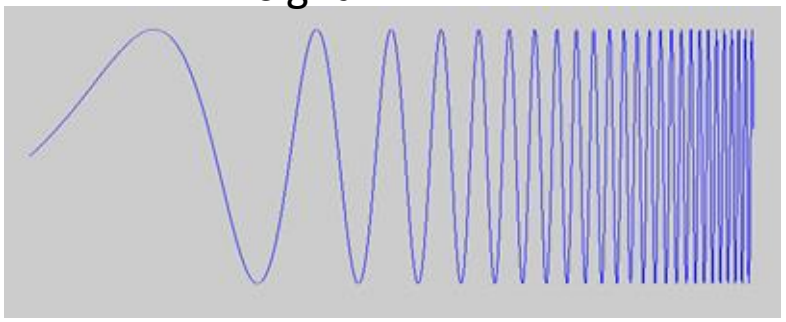



```
x = 0:.05:5;  plt.imshow(sin((2.^x).*x))
```

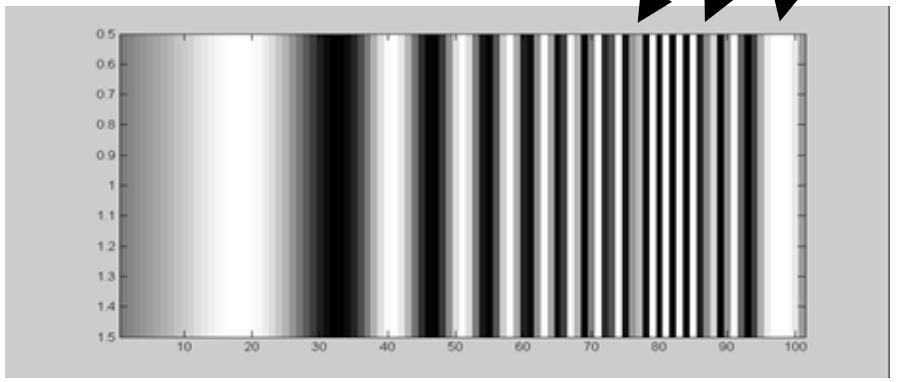
What's happening?

Aliasing!
Not enough
samples

Input
signal:



Plot as image:

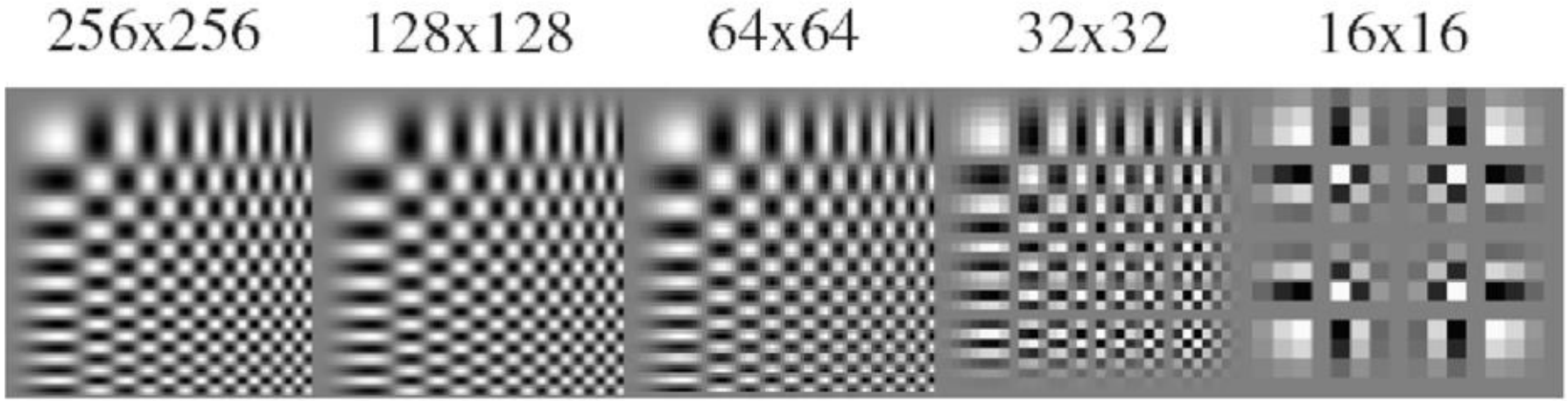



```
x = 0:.05:5;  plt.imshow(sin((2.^x).*x))
```

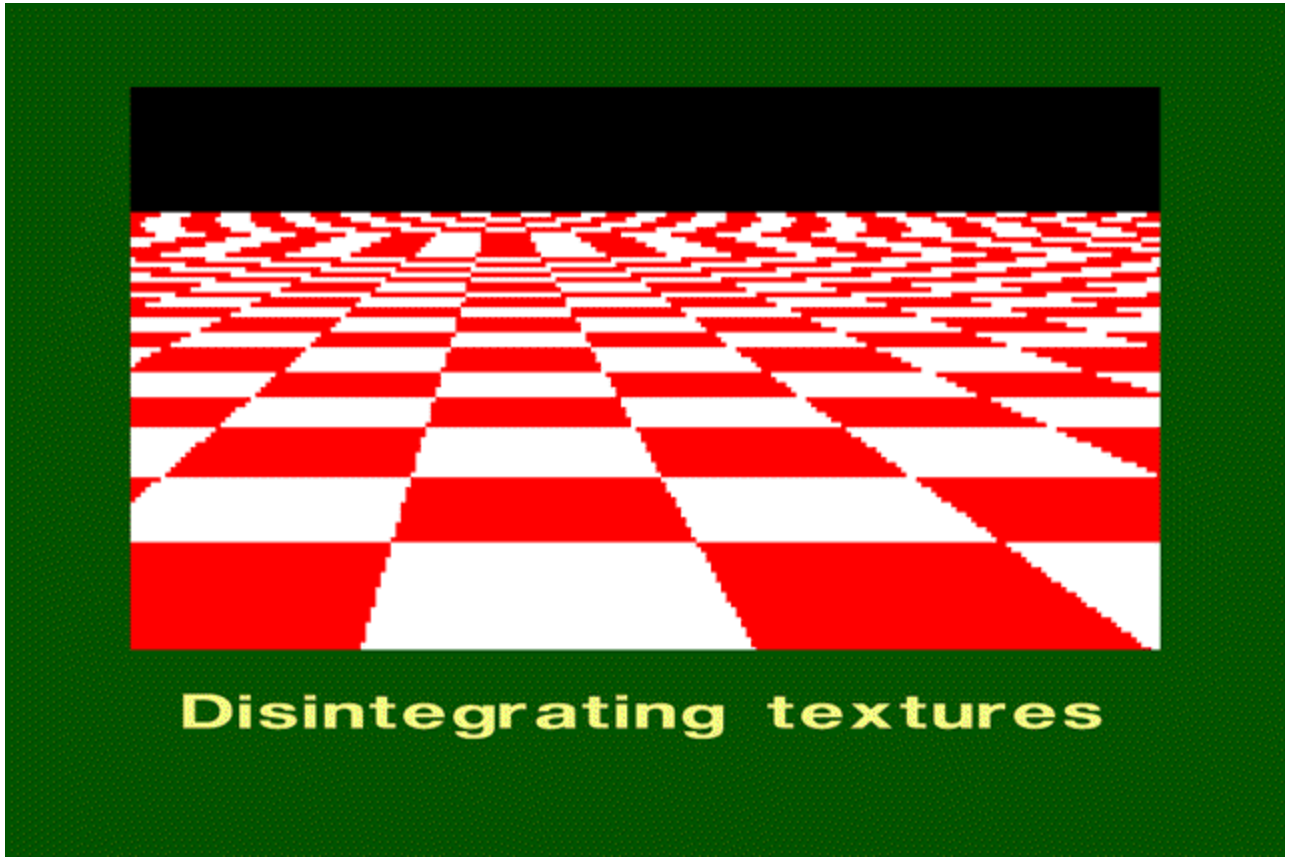
Aliasing:

- When two signals become indistinguishable from one another due to sampling.
- They are 'aliases' of one another.

Sampling and aliasing



Aliasing in Graphics



Temporal Aliasing in Videos

[YouTube; JoinBuzzirk; phrancque]

Temporal Aliasing in Videos

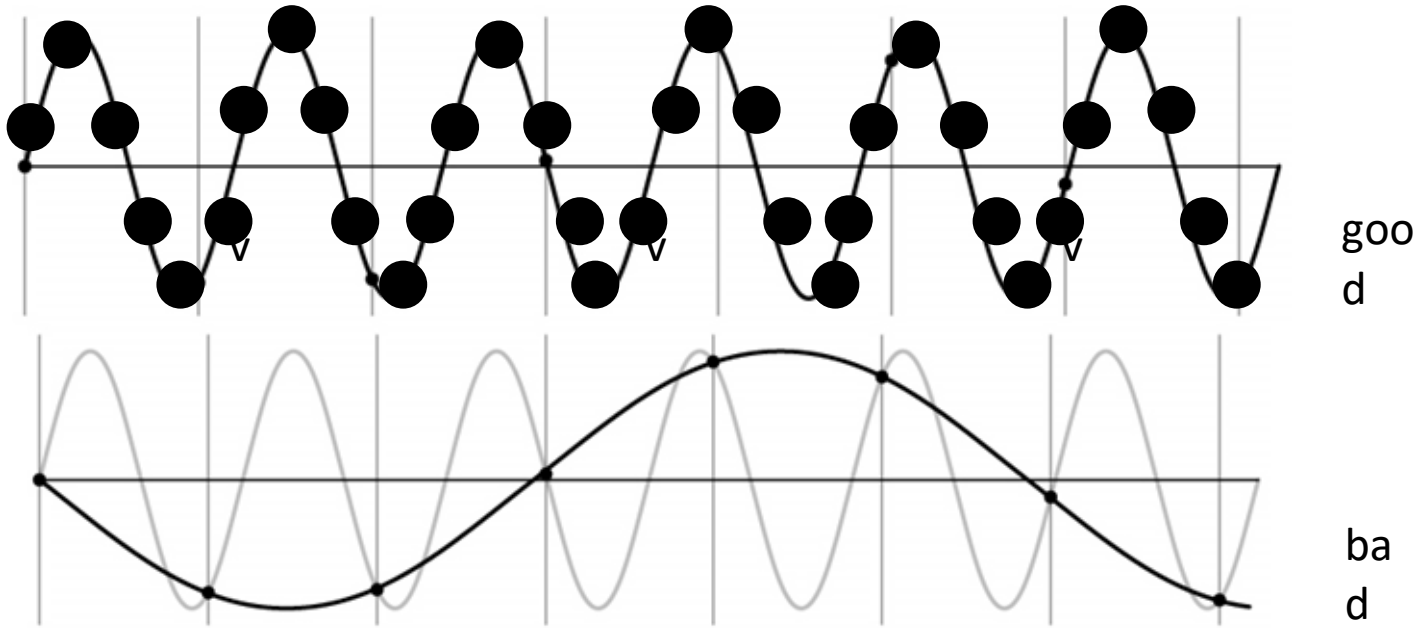
[YouTube; JoinBuzzirk; phrancque]

Temporal Aliasing in Videos

[YouTube; JoinBuzzirk; phrancque]

Nyquist-Shannon Sampling Theorem

- When sampling a signal at discrete intervals, the sampling frequency must be $\geq 2 \times f_{max}$
- f_{max} = max frequency of the input signal
- This allows us to reconstruct the original perfectly from the sampled version



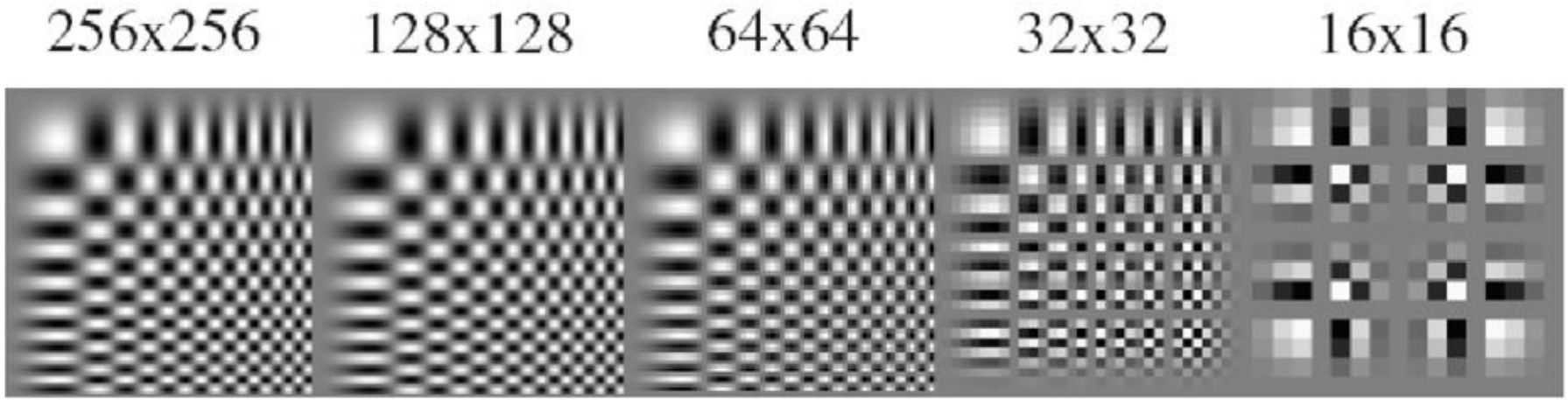
Anti-aliasing

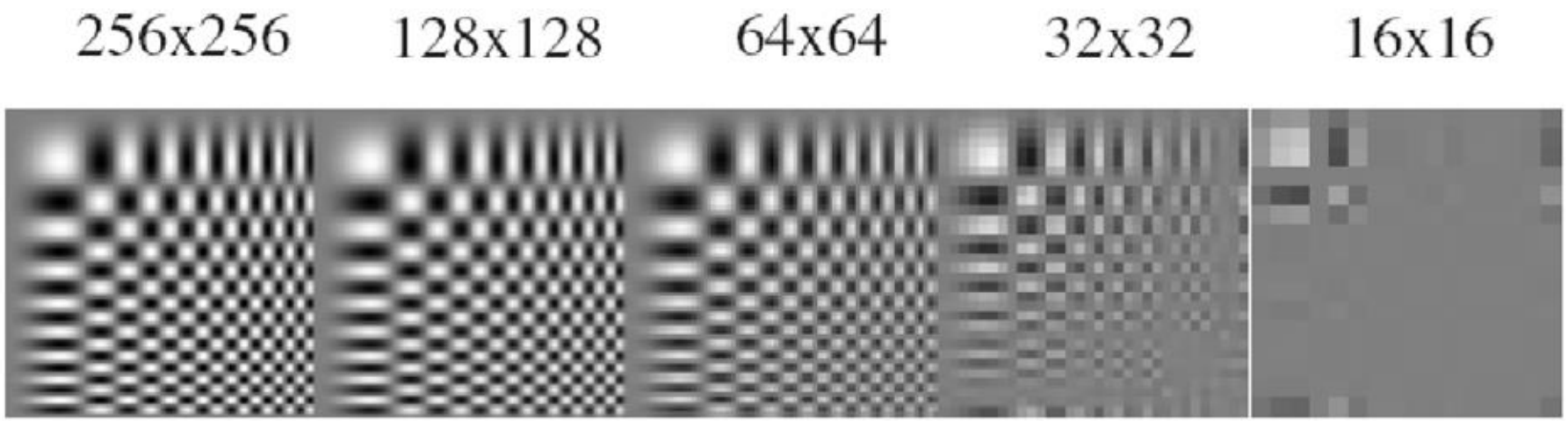
Solutions:

- Sample more often

- Remove all frequencies that are greater than half the new sampling frequency
  - Will lose information
  - But it's better than aliasing
  - How?

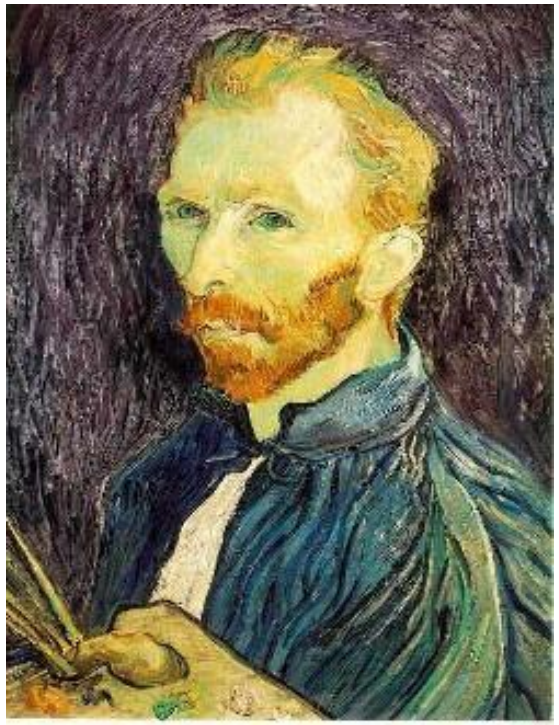
Anti-aliasing



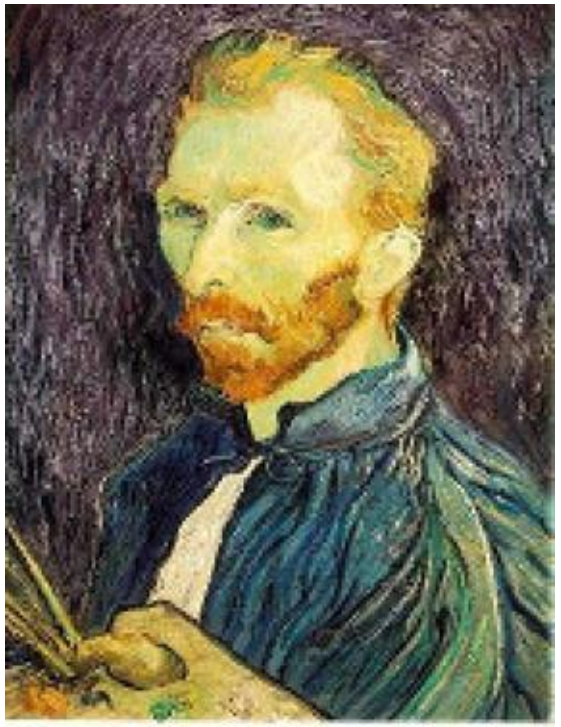


Forsyth and Ponce 2002

Subsampling without Filtering



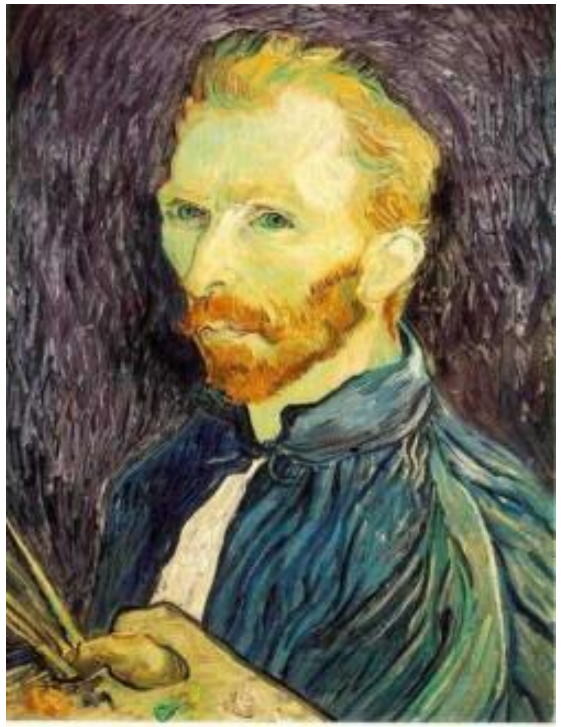
1/2



1/4 (2x subsample)

1/8 (4x subsample)

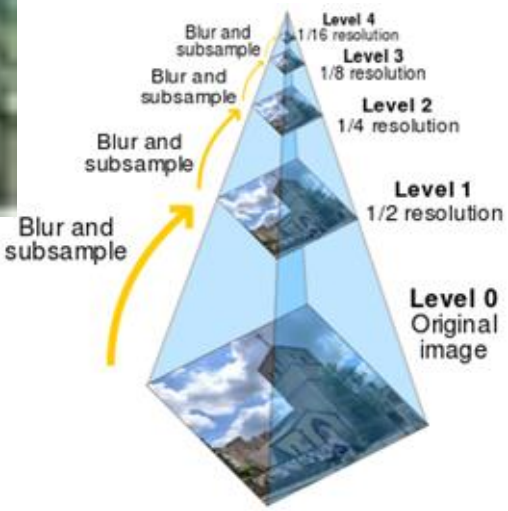
Subsampling with Gaussian Pre-filtering



Gaussian
1/2

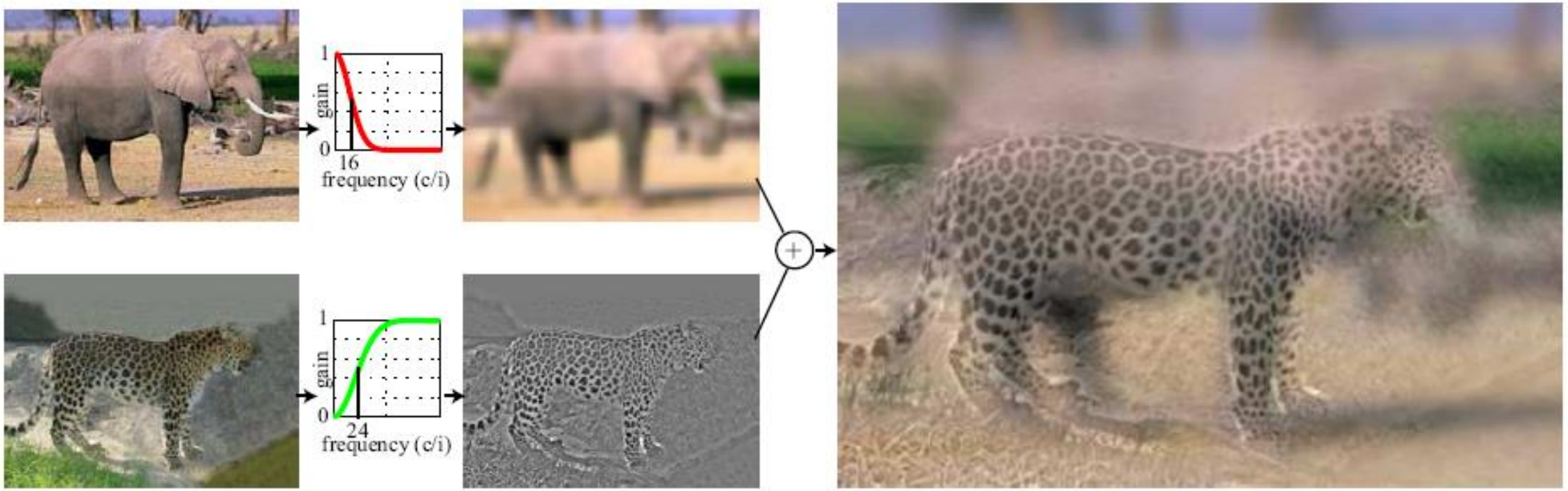
G 1/4

G 1/8



Gaussian Pyramid. Burt and Adelson, 1983

Hybrid Images



Hybrid Images. A. Oliva, A. Torralba, P.G. Schyns

CSCI 1430: COMPUTER VISION

Next episode...

- Frequency Domain
- Hybrid Images
- Fourier Transforms
- FT as filtering

Image Credit: Wallpapercrafter.com