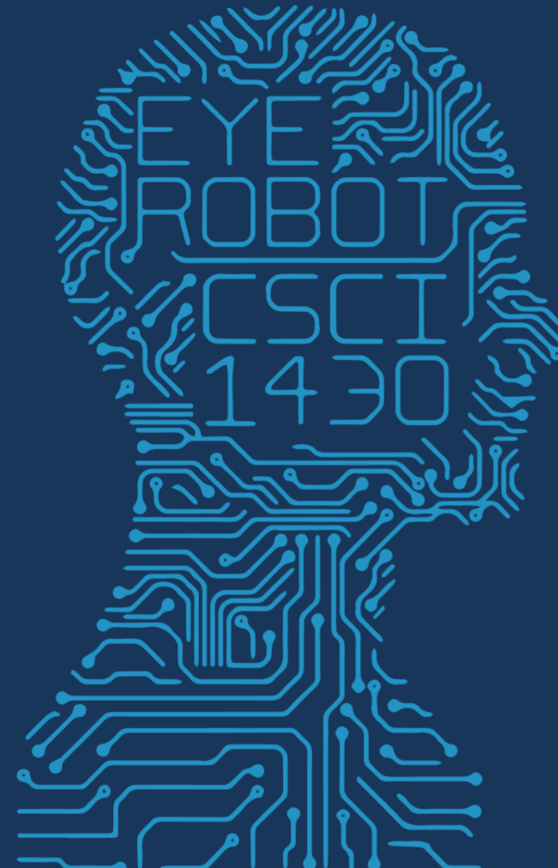
1950

Future Vision



2020

Computer Vision

WHAT IS AN IMAGE?

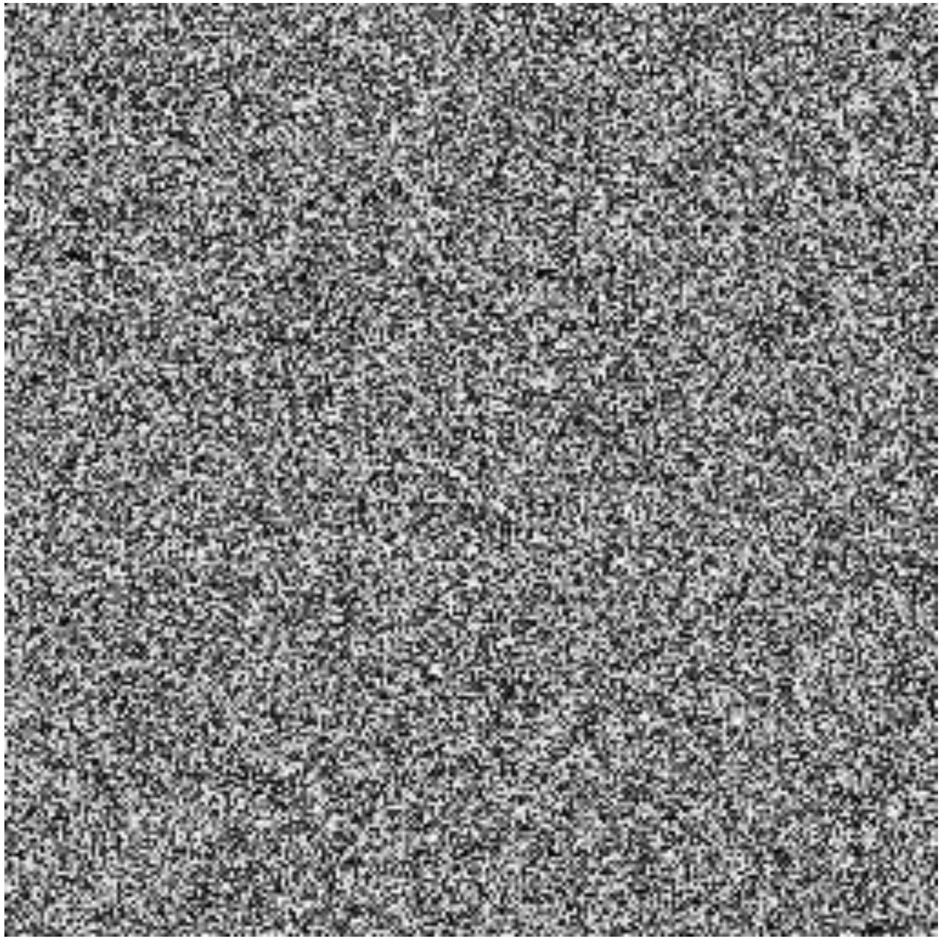
```
>>> from numpy import random as r
>>> I = r.rand(256,256);
```

Think-Pair-Share:

- What is this? What does it look like?
- Which values does it take?
- How many values can it take?

- Is it an image?

```
>>> from matplotlib import pyplot as p
>>> I = r.rand(256,256);
>>> p.imshow(I);
>>> p.show();
```



Danny Alexander

Dimensionality of an Image

- @ 8bit = 256 values ^ 65,536
  - Computer says ‘Inf’ combinations.

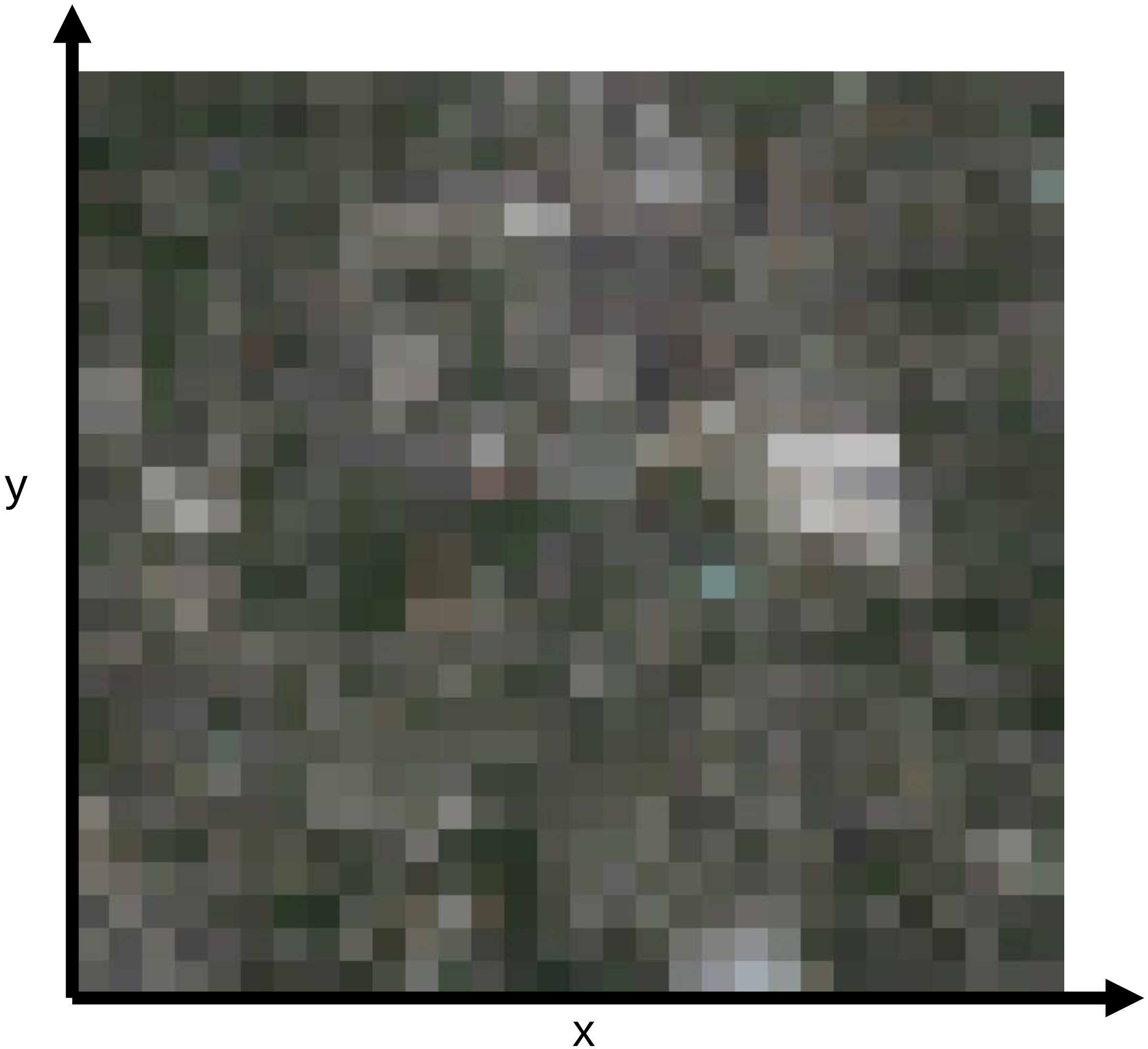
- Some depiction of all possible scenes would fit into this memory.

Dimensionality of an Image

- @ 8bit = 256 values ^ 65,536
  - Computer says 'Inf' combinations.

- Some depiction of all possible scenes would fit into this memory.
- Computer vision as making sense of an extremely high-dimensional space.
  - Subspace of 'natural' images.
  - Deriving low-dimensional, explainable models.

What is each part of an image?



What is each part of an image?

- Pixel -> picture element

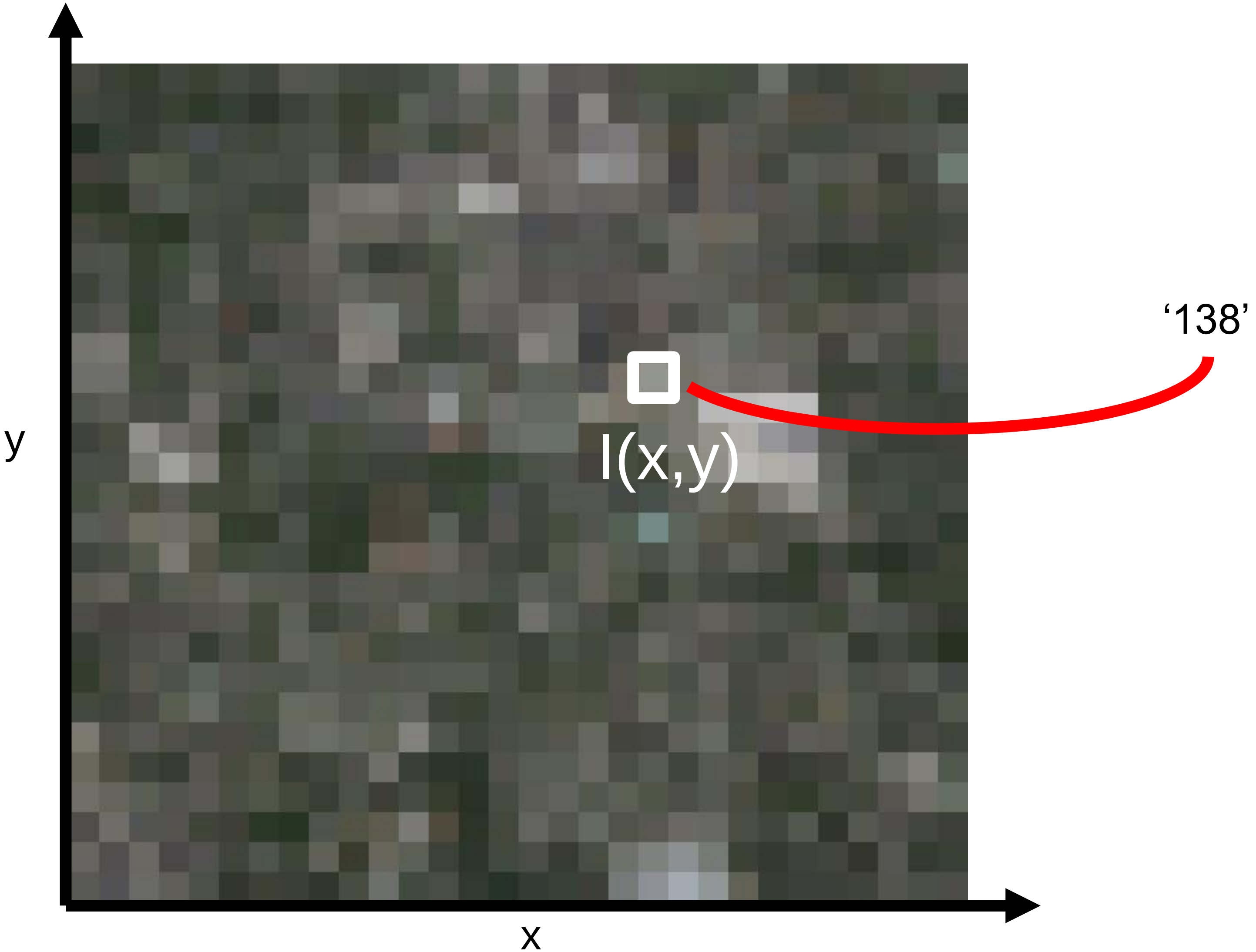


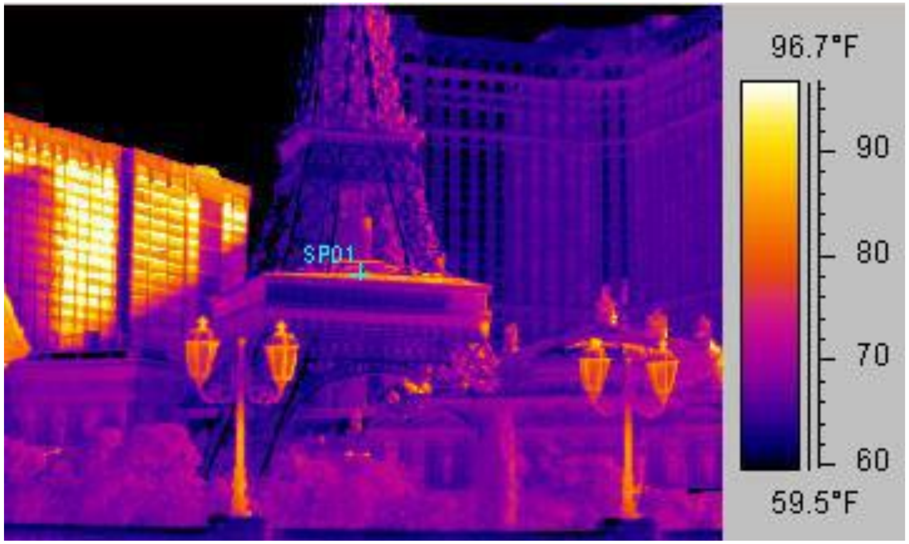
Image as a 2D sampling of signal

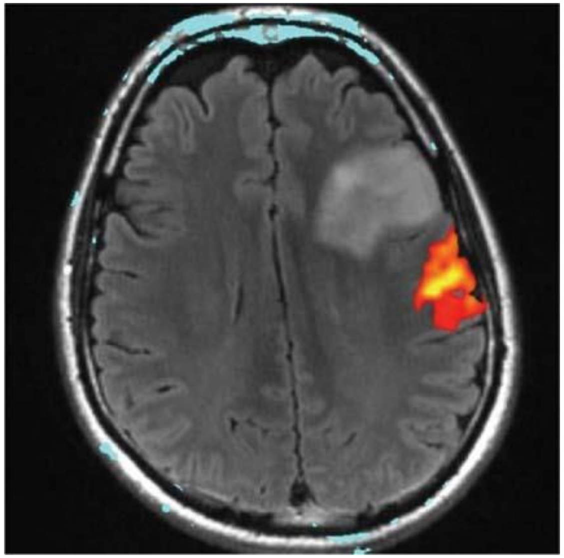
- Signal: function depending on some variable with physical meaning.

- Image: sampling of that function.
  - 2 variables: xy coordinates
  - 3 variables: xy + time (video)
  - 'Brightness' is the value of the function for visible light

- Can be other physical values too: temperature, pressure, depth …

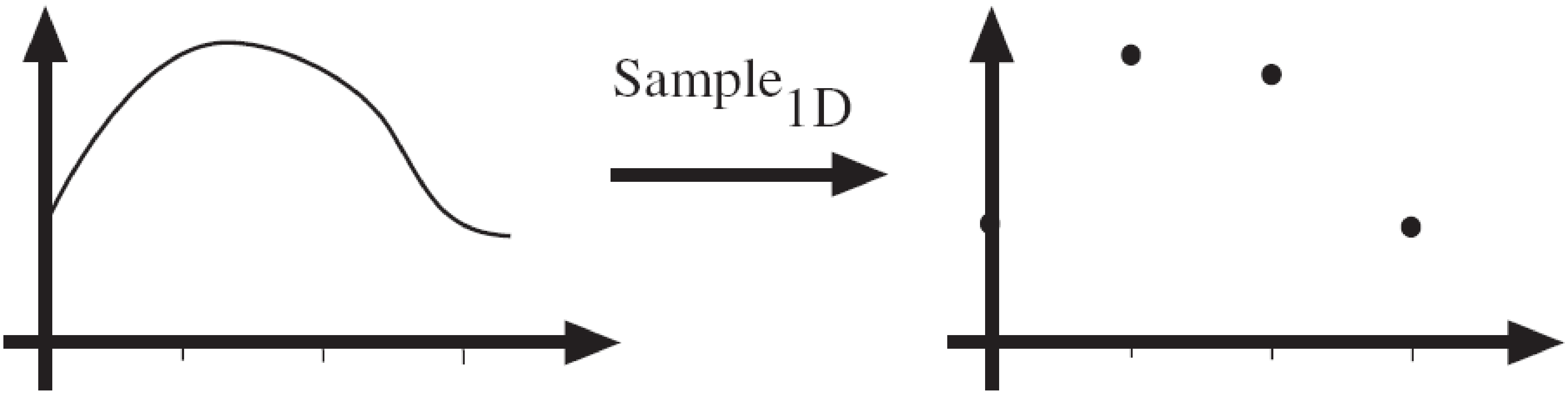
Example 2D Images





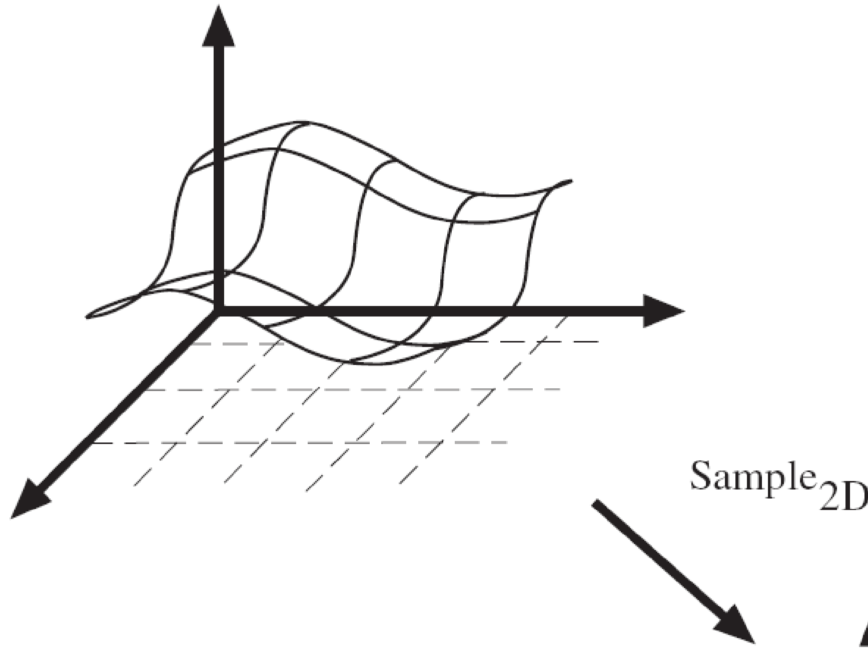
Danny Alexander

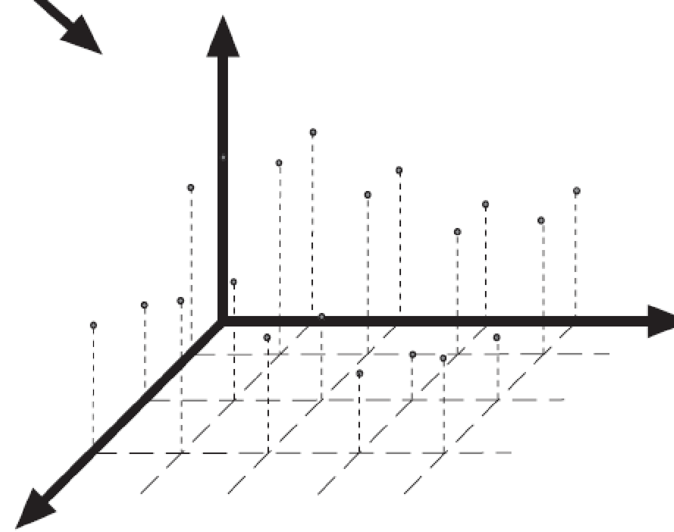
Sampling in 1D



Sampling in 1D takes a function and returns a vector whose elements are values of that function at the sample points.

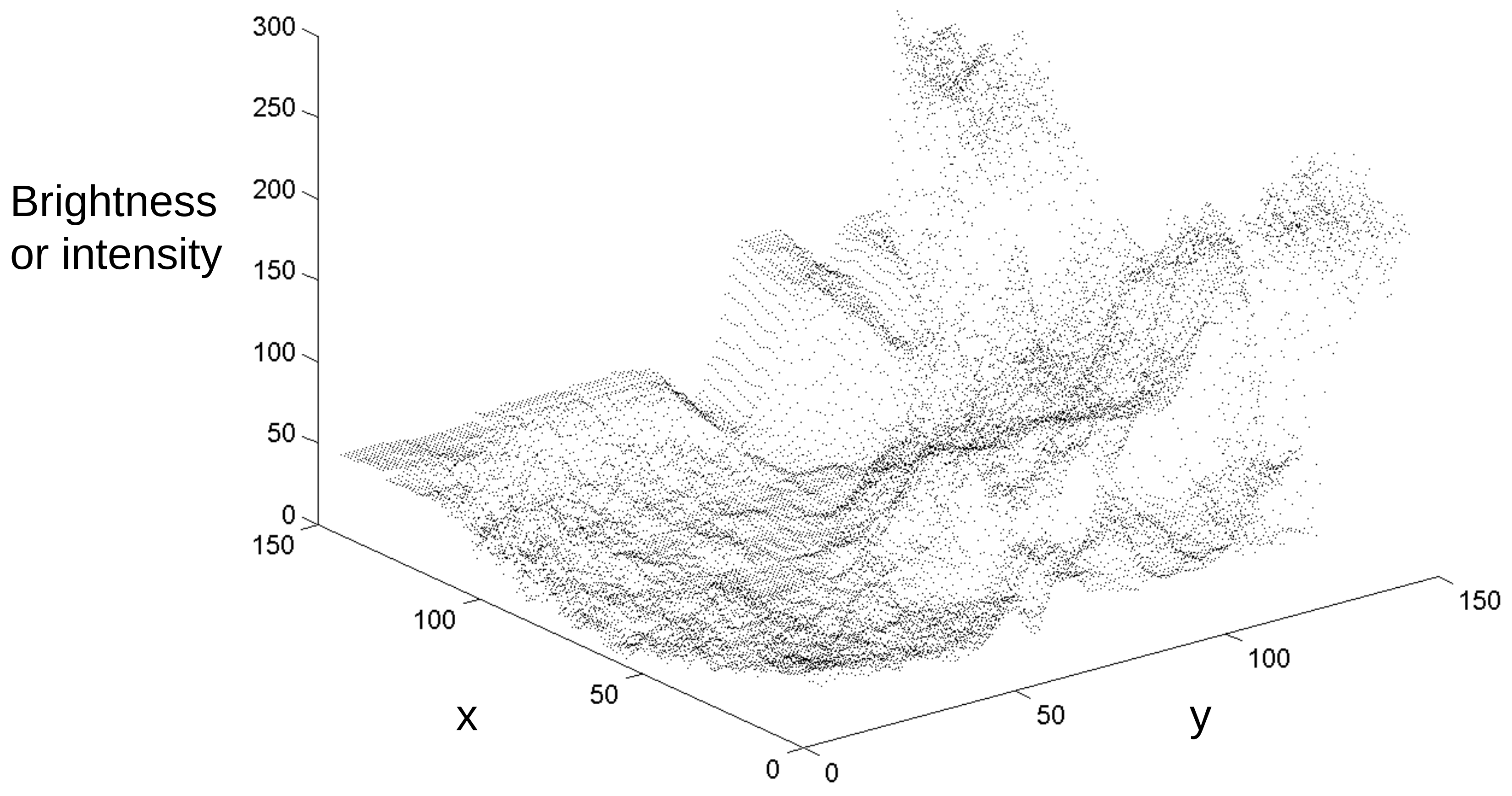
Sampling in 2D





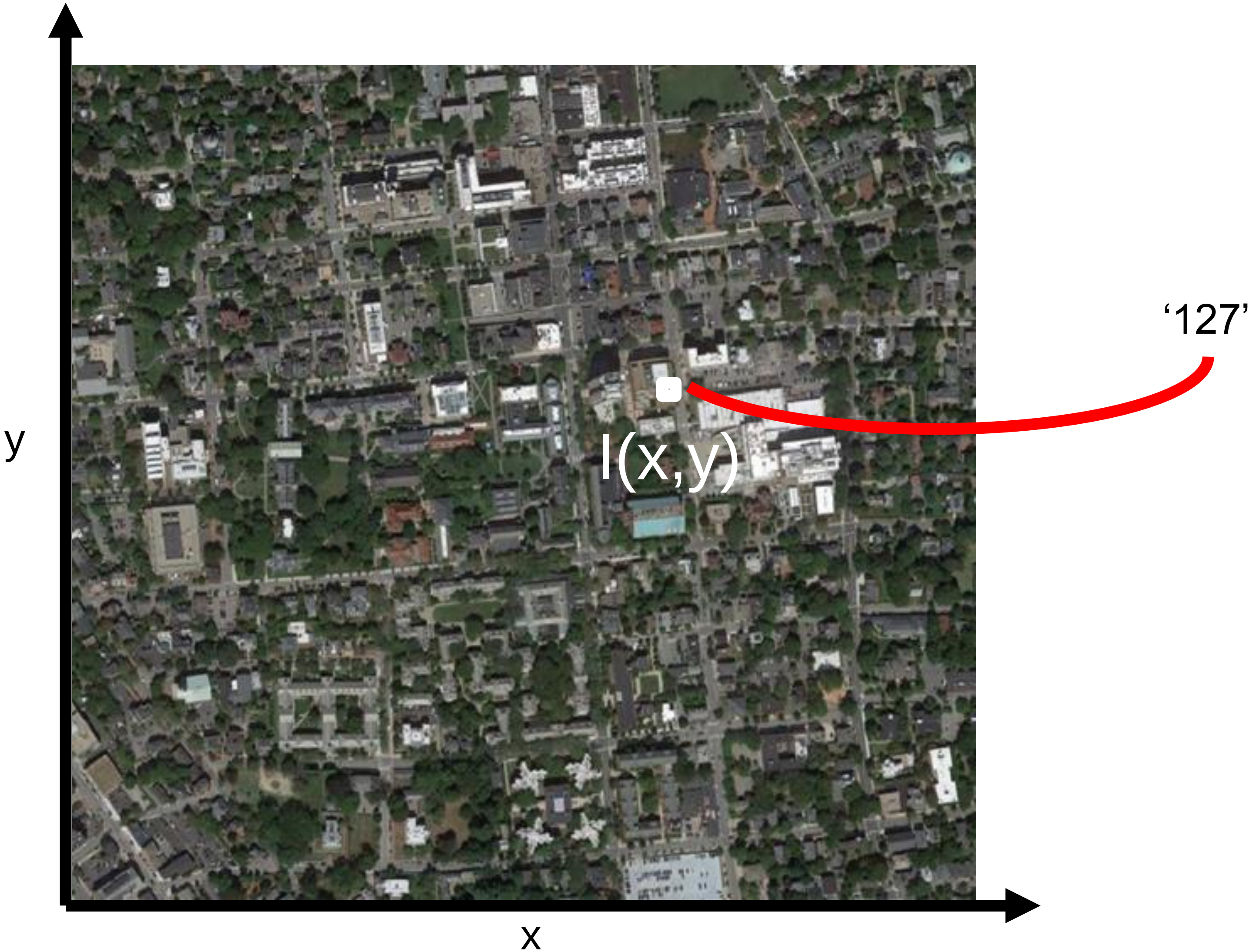
Sampling in 2D takes a function and returns a matrix.

Grayscale Digital Image

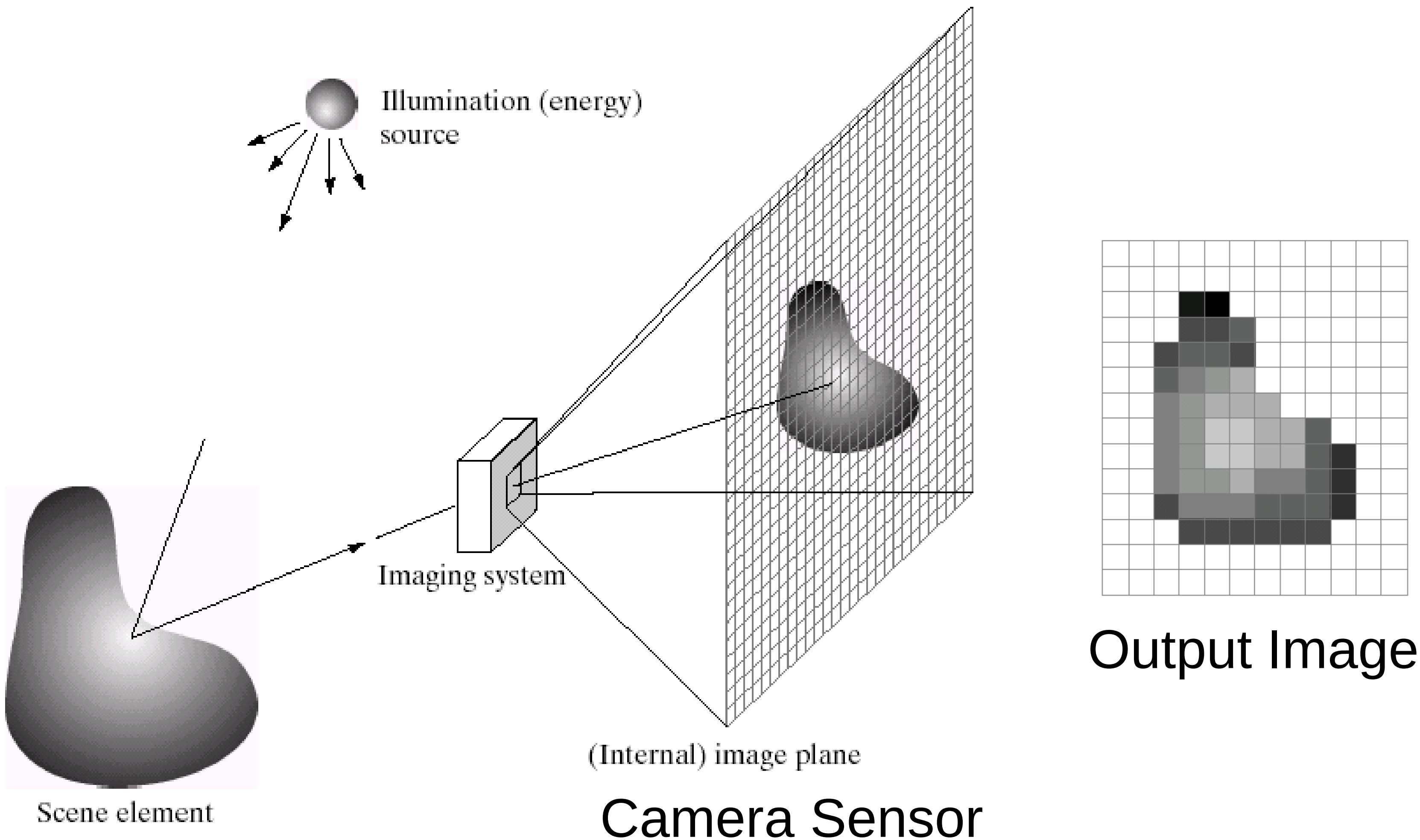


What is each part of a photograph?

- Pixel -> picture element



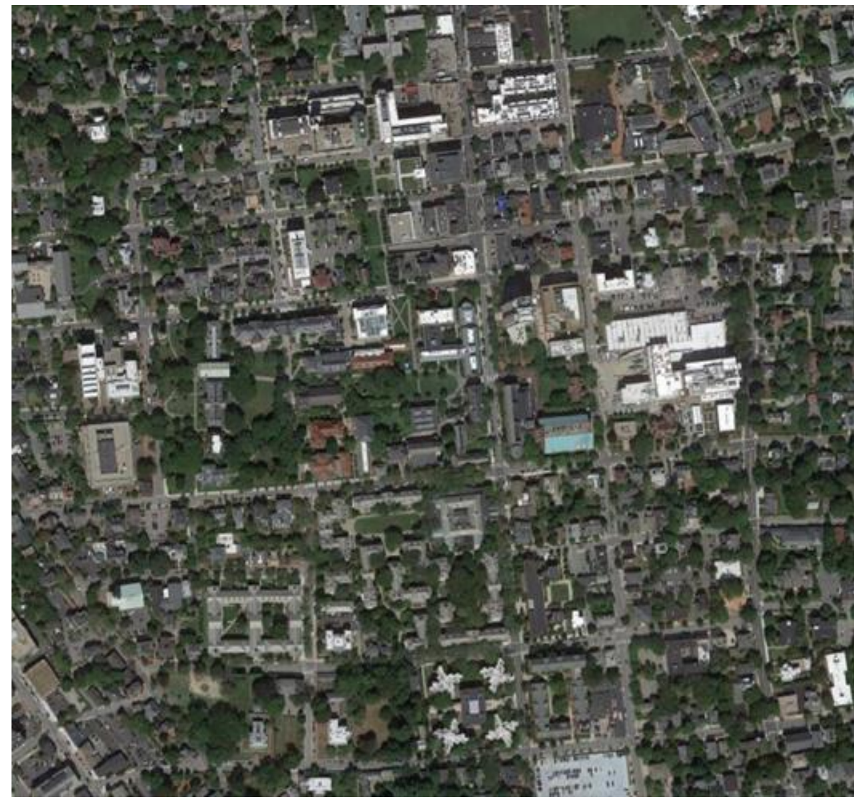
Integrating light over a range of angles.



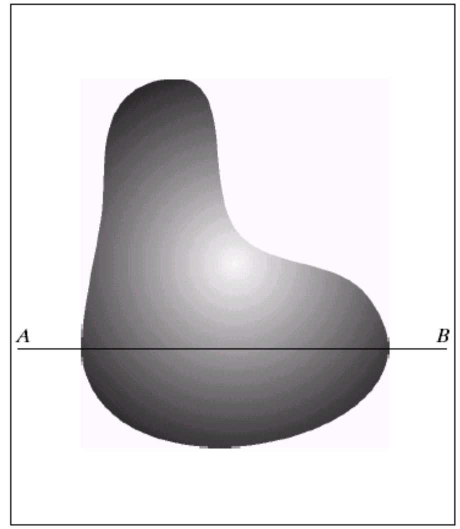
James Hays

Resolution – geometric vs. spatial resolution

Both images are ~500x500 pixels

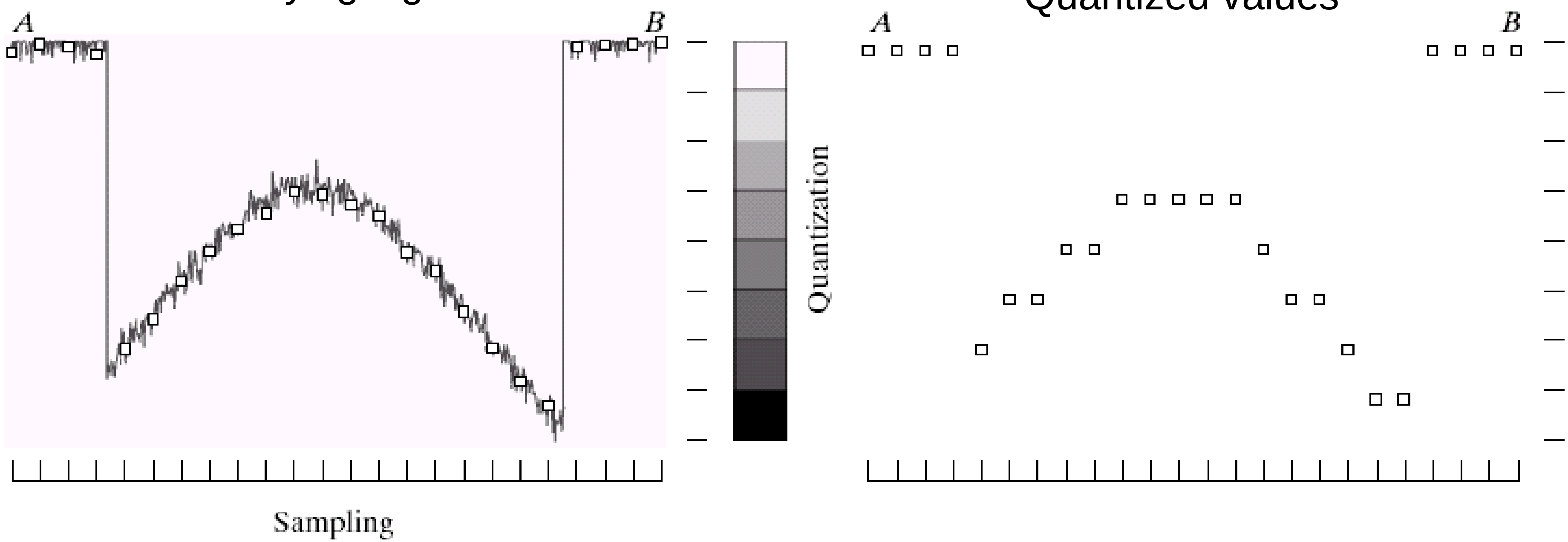


Quantization

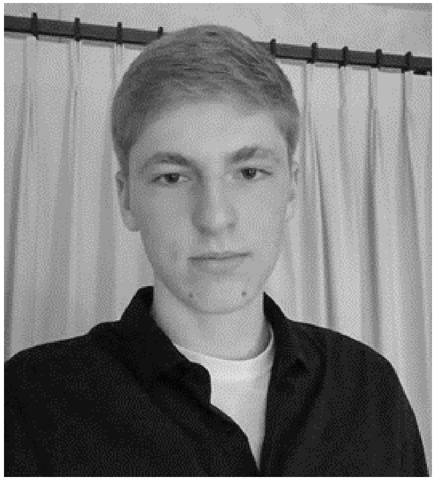


Underlying signal

Quantized values

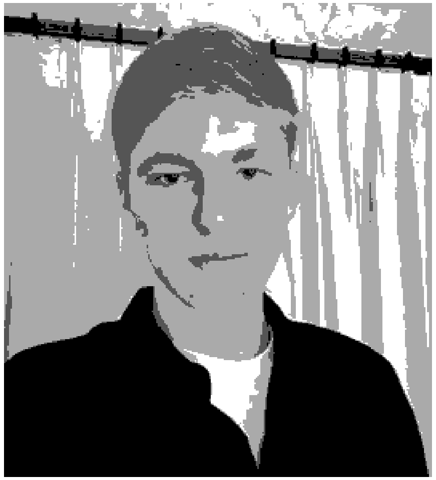


Quantization Effects – Radiometric Resolution



8 bit – 256 levels

4 bit – 16 levels



2 bit – 4 levels

1 bit – 2 levels

Images in Python Numpy

N x M grayscale image "im"

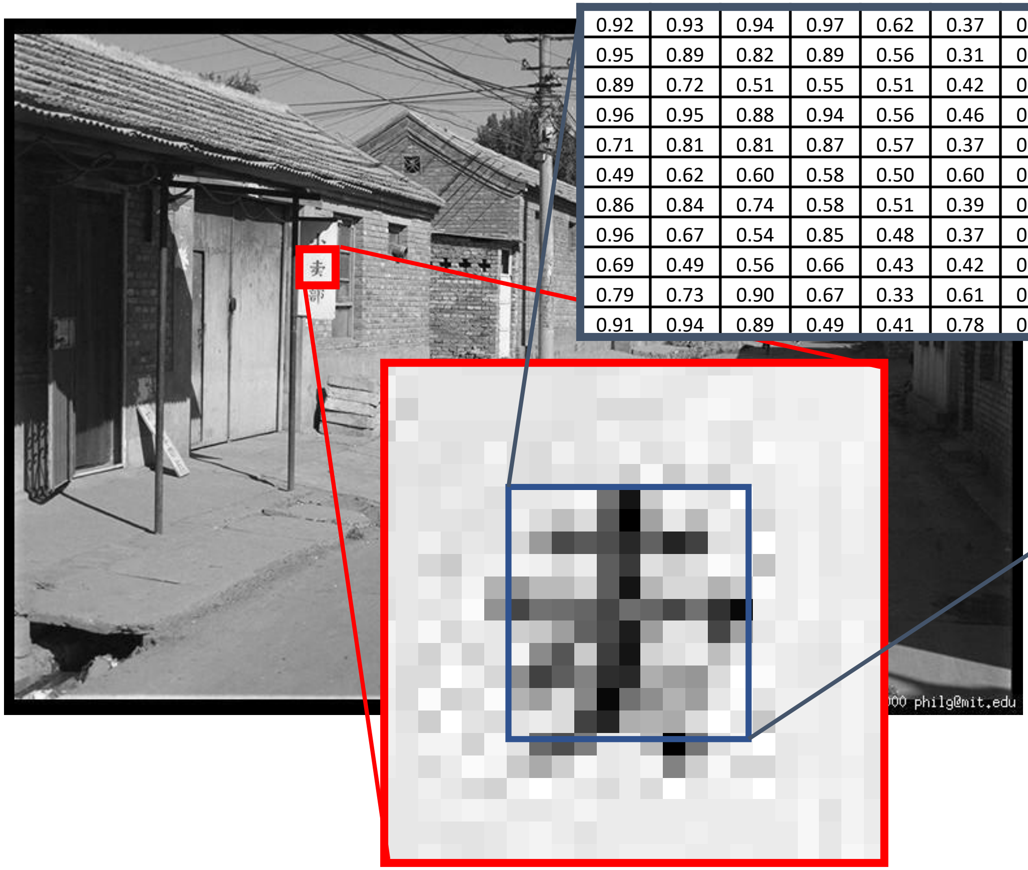
– im[0,0] = top-left pixel value
– im[y, x] = y pixels down, x pixels to right
– im[N-1, M-1] = bottom-right pixel

Row ↓ / **Column** →

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

James Hays

Grayscale intensity



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

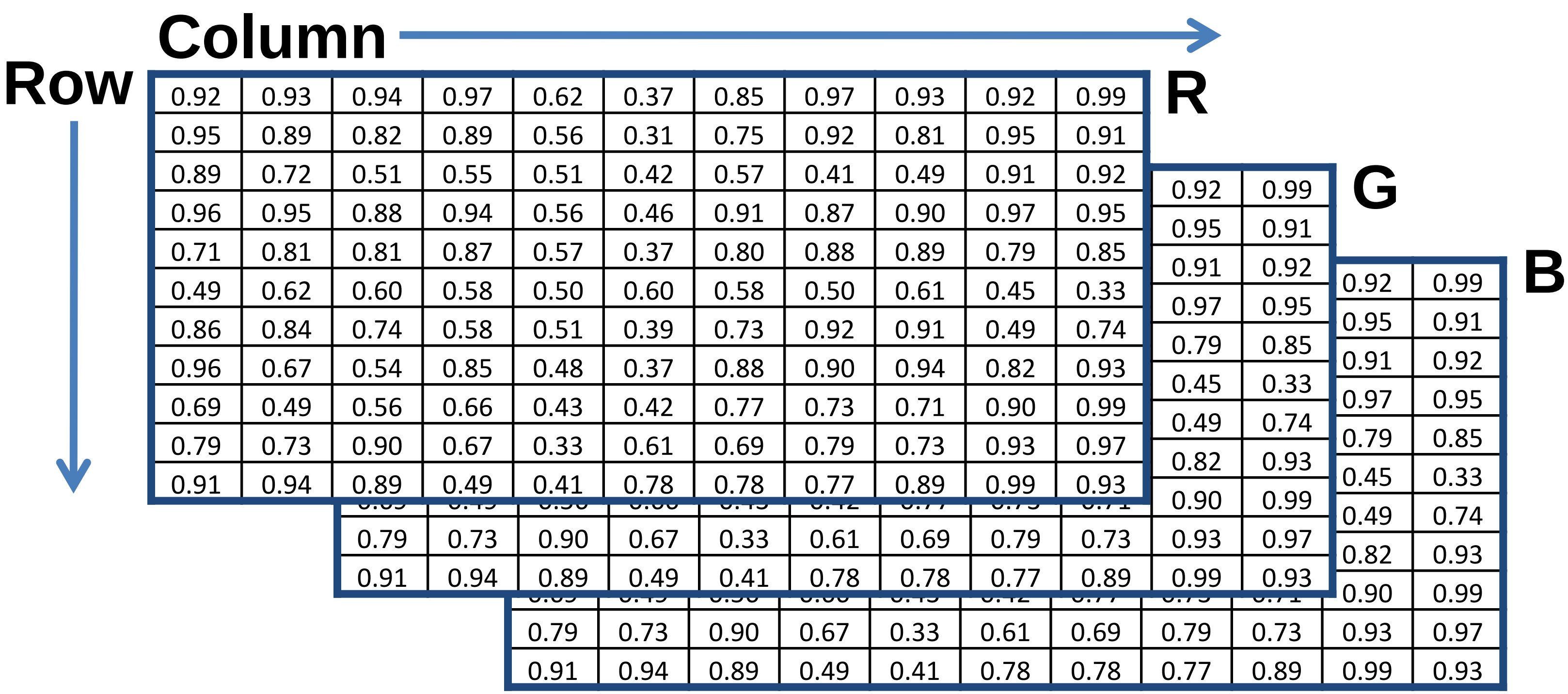
Color

James Hays

Images in Python Numpy

N x M RGB image "im"

- im[0,0,0] = top-left pixel value in R-channel
- Im[x, y, b] = x pixels to right, y pixels down in the b^{th} channel
- Im[N-1, M-1, 3] = bottom-right pixel in B-channel



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

Images in Python Numpy

Take care between types!

- uint8 (values 0 to 255) – io.imread("file.jpg")
- float32 (values 0 to 255) – io.imread("file.jpg").astype(np.float32)
- float32 (values 0 to 1) – img_as_float32(io.imread("file.jpg"))

Column →

Row ↓

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

Ben Thomas

IMAGE FILTERING

Image filtering

Compute function of local neighborhood at each position:

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

James Hays

Image 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

[ ] [ ] [ ]

Example: box filter

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

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

Image filtering

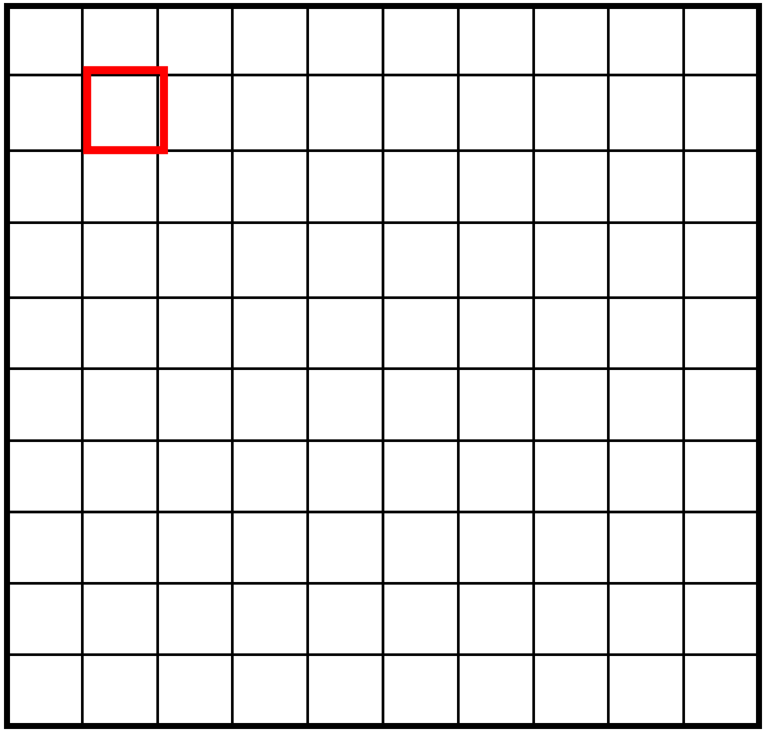
$f[\cdot,\cdot]\ \frac{1}{9}$

1	1	1
1	1	1
1	1	1

$I[.,.]$

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

$h[.,.]$



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

$m = 1, n = 1$

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

Credit: S. Seitz

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							

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

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

Credit: S. Seitz

Image filtering

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

1	1	1
1	1	1
1	1	1

$I[.,.]$

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						

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

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

Credit: S. Seitz

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	30					

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

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

Credit: S. Seitz

Image filtering

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

1	1	1
1	1	1
1	1	1

$I[.,.]$

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	30	30				

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

$m = 5, n = 1$

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

Credit: S. Seitz

Image filtering

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

1	1	1
1	1	1
1	1	1

$I[.,.]$

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	30	30				
				?					

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

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

Credit: S. Seitz

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	30	30				
						?			
				50					

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

$m = 6, n = 4$

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

Credit: S. Seitz

Image filtering

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

$I[.,.]$

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	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	

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

Credit: S. Seitz

Box Filter

What does it do?

- Replaces each pixel with an average of its neighborhood

- Achieve smoothing effect (remove sharp features)

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

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

Slide credit: David Lowe (UBC)

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?

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

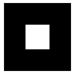
$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Slide credit: David Lowe (UBC)

Smoothing with box filter

David Lowe

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



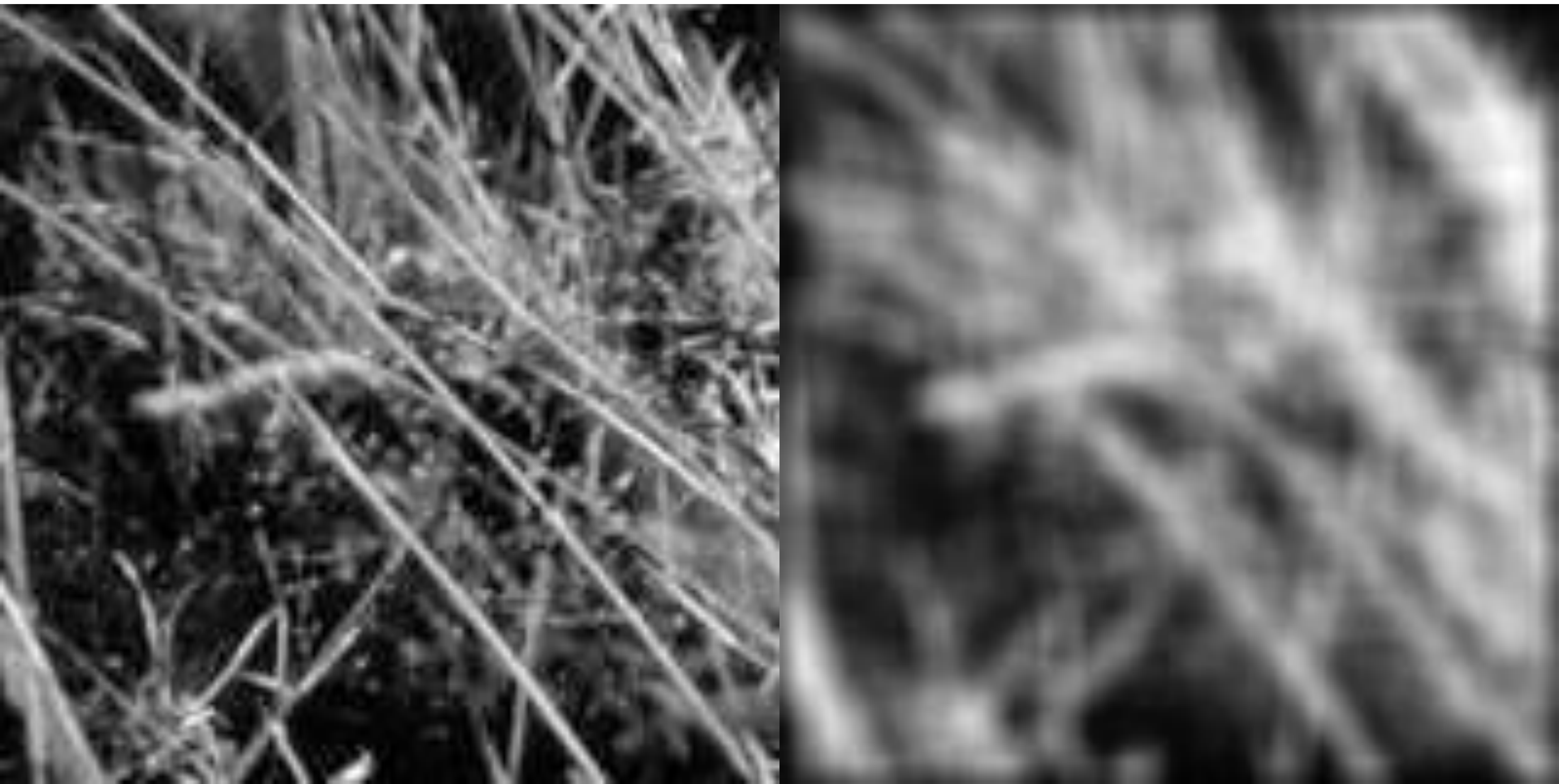


Image filtering

Compute function of local neighborhood
at each position:

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

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

James Hays

Think-Pair-Share time

1. $$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

2. $$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

3. $$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

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

1. Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

?

Source: D. Lowe

1. Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

Filtered
(no change)

Source: D. Lowe

2. Practice with linear filters

Original

0	0	0
0	0	1
0	0	0

?

Source: D. Lowe

2. Practice with linear filters

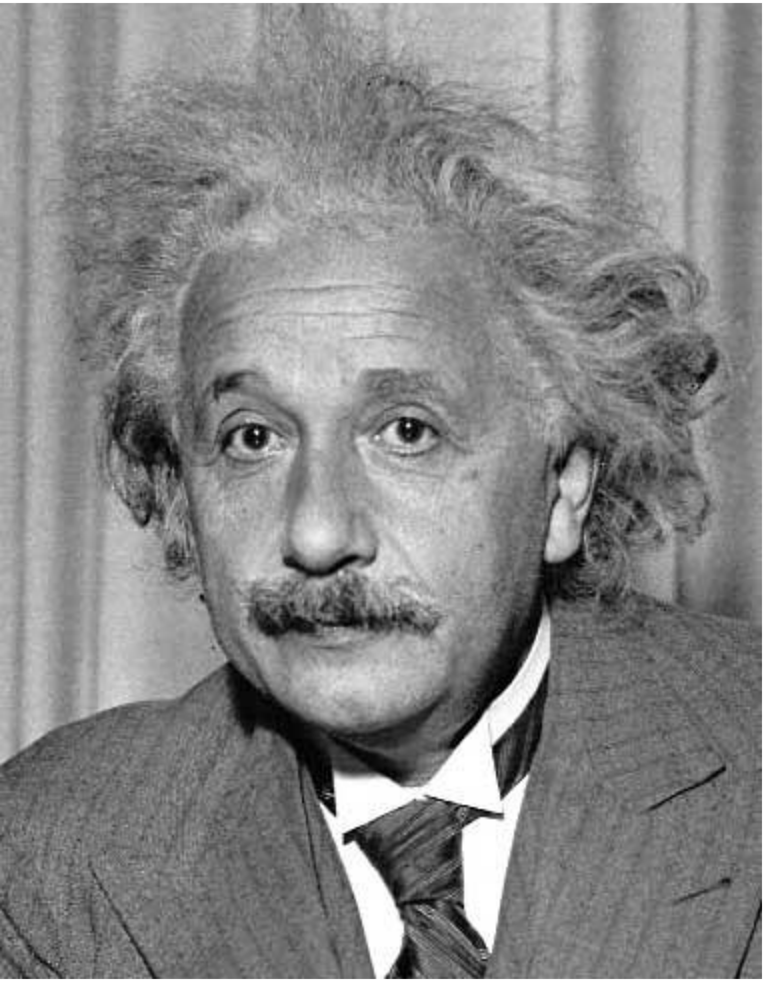
Original

0	0	0
0	0	1
0	0	0

Shifted left
By 1 pixel

Source: D. Lowe

3. Practice with linear filters



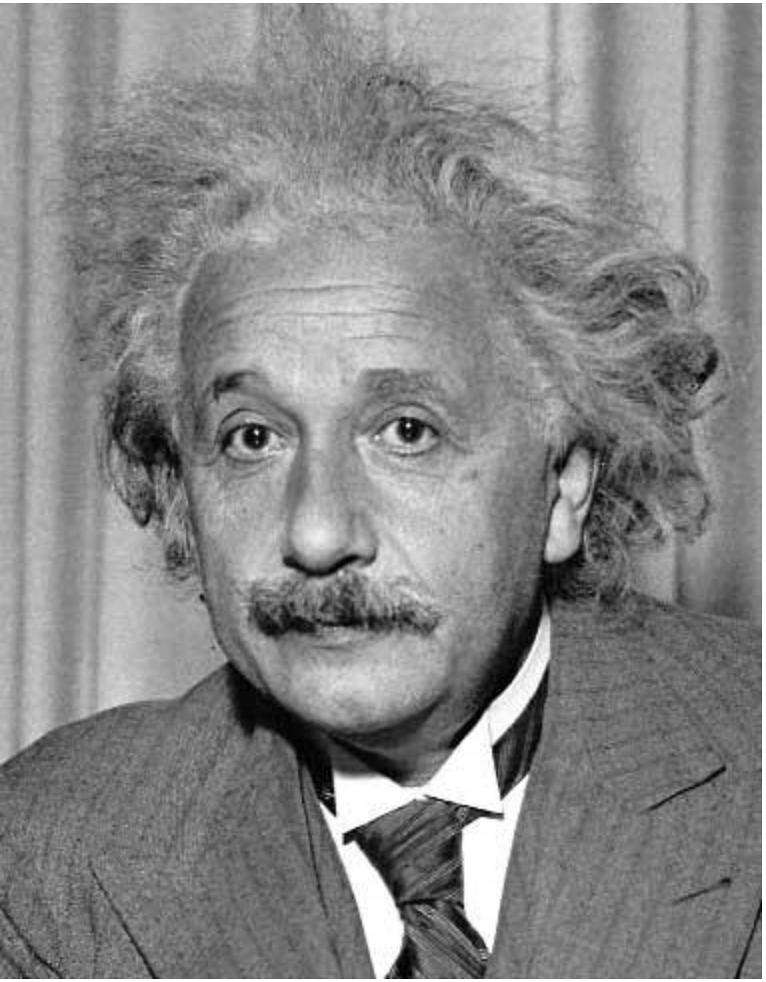
1	0	-1
2	0	-2
1	0	-1

Sobel

Vertical Edge
(absolute value)

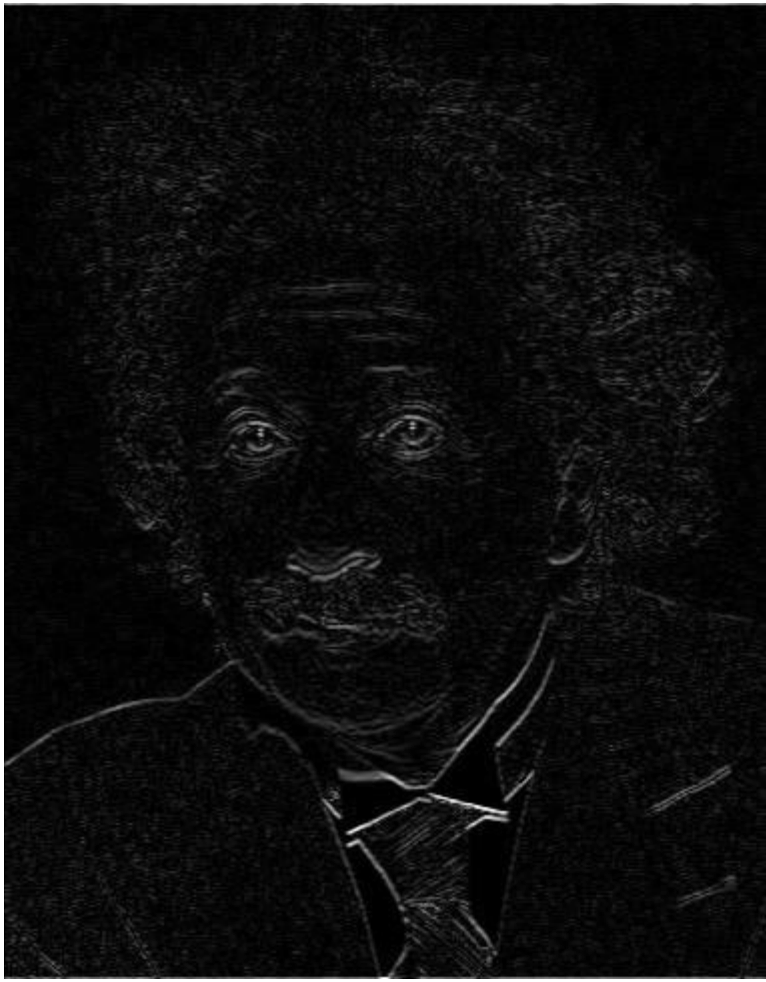
David Lowe

3. Practice with linear filters



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

Sobel



Horizontal Edge
(absolute value)

David Lowe

4. Practice with linear filters

Original

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

?

(Note that filter sums to 1)

Source: D. Lowe

4. Practice with linear filters

Original

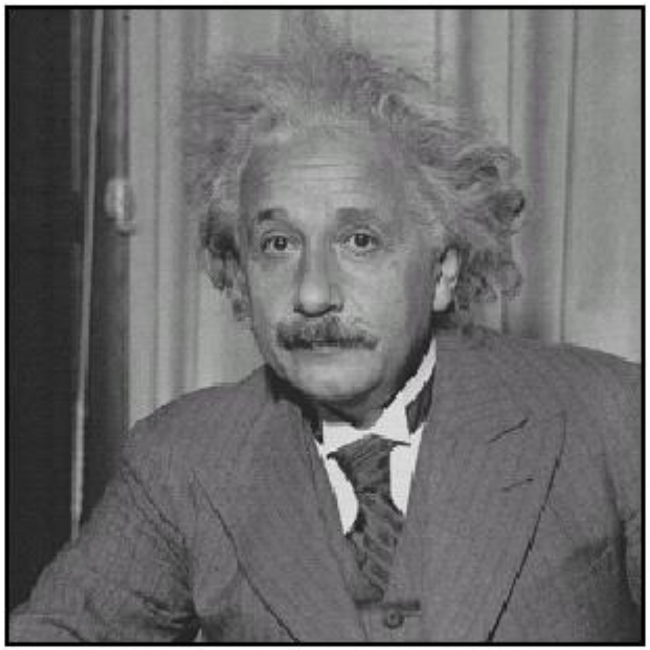
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sharpening filter

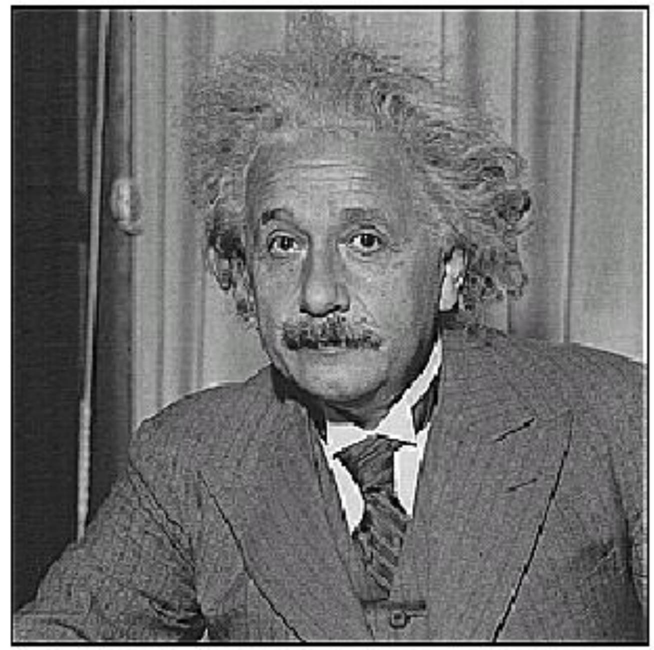
- Accentuates differences with local average

Source: D. Lowe

4. Practice with linear filters



before



after

Source: D. Lowe

Correlation and Convolution

- 2d correlation

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

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

James Hays

Correlation and Convolution

- 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 symmetric.

James Hays

Key properties of linear filters

Linearity:

```
imfilter(I, f₁ + f₂) =
   imfilter(I,f₁) + imfilter(I,f₂)
```

Shift invariance:

Same behavior given intensities regardless of pixel location m,n

```
imfilter(I,shift(f)) = shift(imfilter(I,f))
```

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

S. Lazebnik

Convolution properties

Commutative: $a * b = b * a$

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

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

- Often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
- This is equivalent to applying one filter: a * $(b_1 * b_2 * b_3)$

Source: S. Lazebnik

Convolution properties

Commutative: $a * b = b * a$

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

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

- Often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
- This is equivalent to applying one filter: a * $(b_1 * b_2 * b_3)$
- Correlation is _not_ associative (rotation effect)
- Why important?

Source: S. Lazebnik

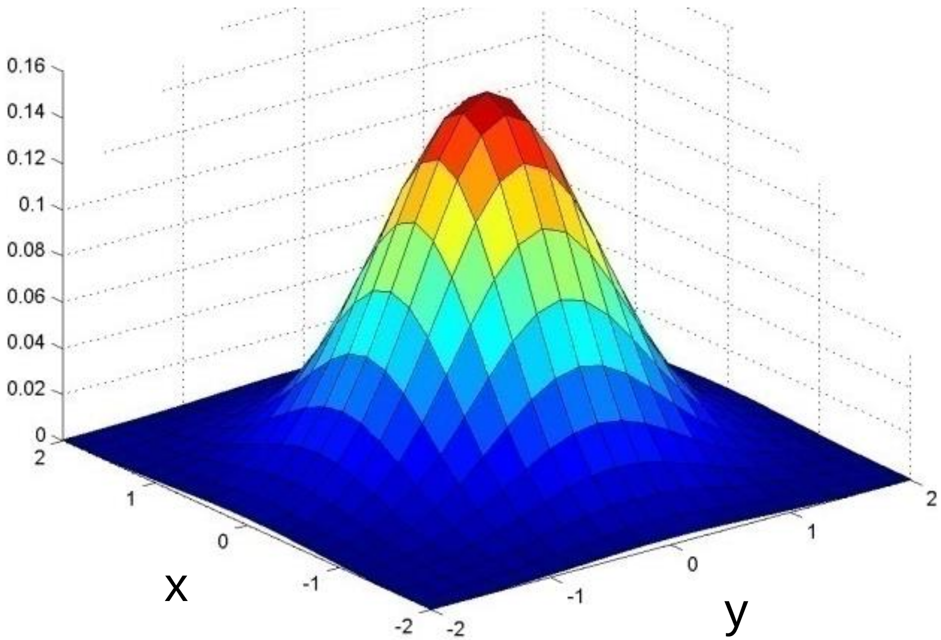
Convolution properties

- Commutative: $a * b = b * a$
  - Conceptually no difference between filter and signal
  - But particular filtering implementations might break this equality, e.g., image edges
- Associative: $a * (b * c) = (a * b) * c$
  - Often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
  - This is equivalent to applying one filter: a * $(b_1 * b_2 * b_3)$
  - Correlation is _not_ associative (rotation effect)
  - Why important?
- Distributes over addition: $a * (b + c) = (a * b) + (a * c)$
- Scalars factor out: $ka * b = a * kb = k (a * b)$
- Identity: unit impulse $e = [0, 0, 1, 0, 0]$, $a * e = a$

Source: S. Lazebnik

Important filter: Gaussian

Weight contributions of neighboring pixels by nearness



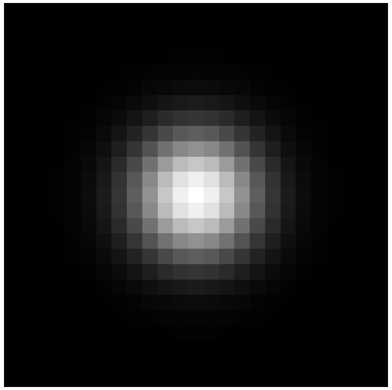
x

y

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

Kernel size 5 x 5,
Standard deviation $\sigma = 1$

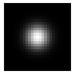
Viewed from top

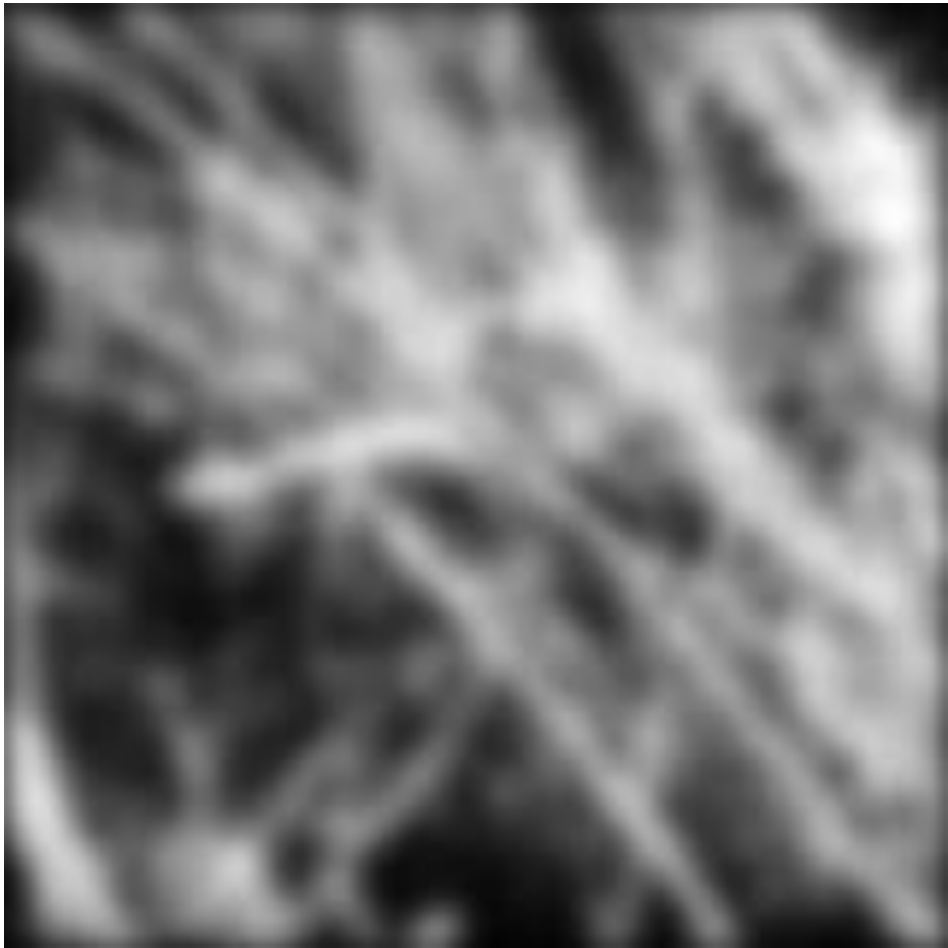


$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Slide credit: Christopher Rasmussen

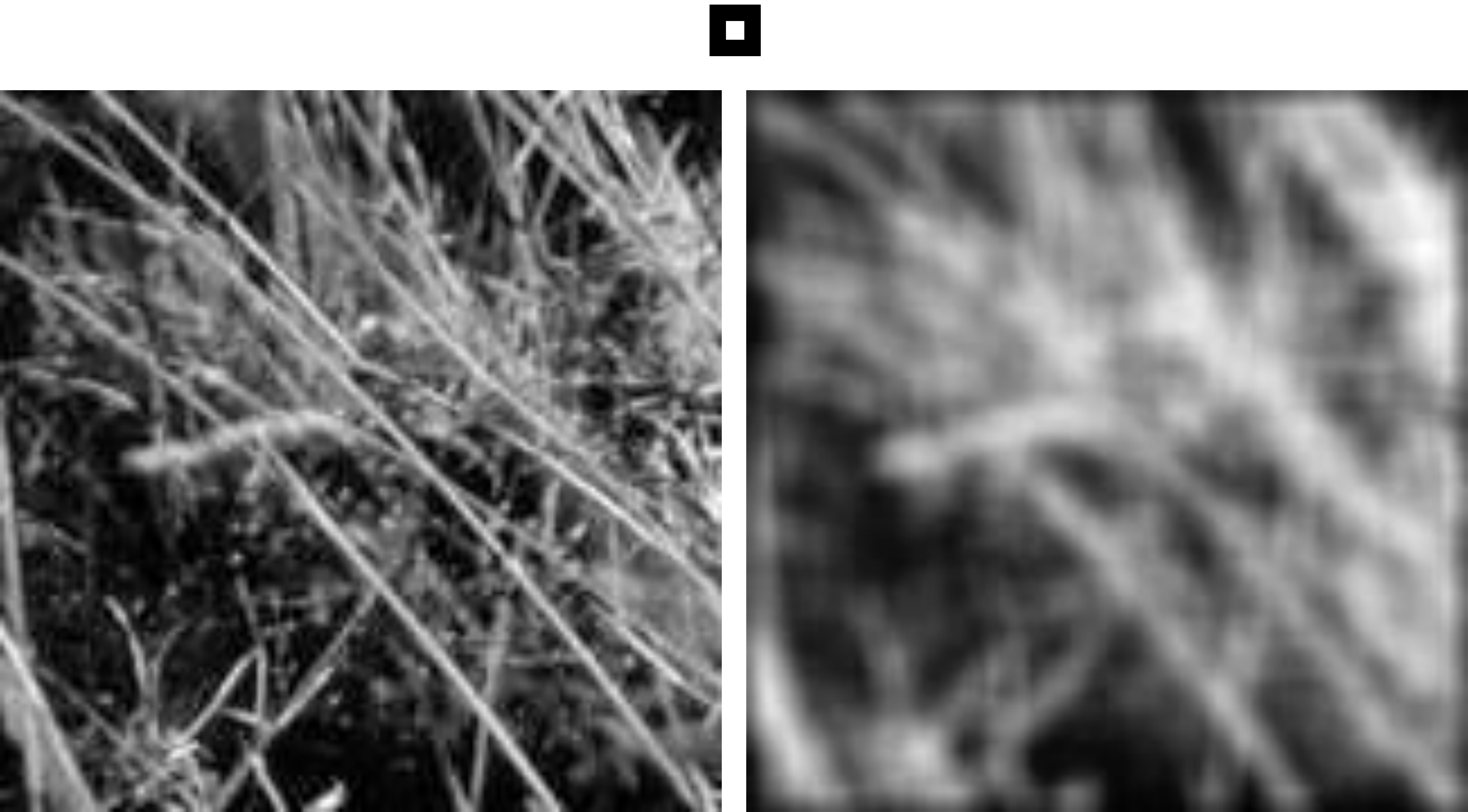
Smoothing with Gaussian filter





James Hays

Smoothing with box filter



Gaussian filters

- Remove “high-frequency” components from the image (a low-pass filter)
  – Images become more smooth
- Gaussian convolved with Gaussian…

  …is another Gaussian
  – So can smooth with small-width kernel, repeat, and get same result as larger-width kernel would have
  – 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

Source: K. Grauman

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

Source: D. Lowe

Separability example

2D convolution (center location only)

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

The filter factors into a product of 1D filters:

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

Perform convolution along rows:

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

Followed by convolution along the remaining column:

Source: K. Grauman

Separability

Why is separability useful in practice?

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

Practical matters

How big should the filter be?

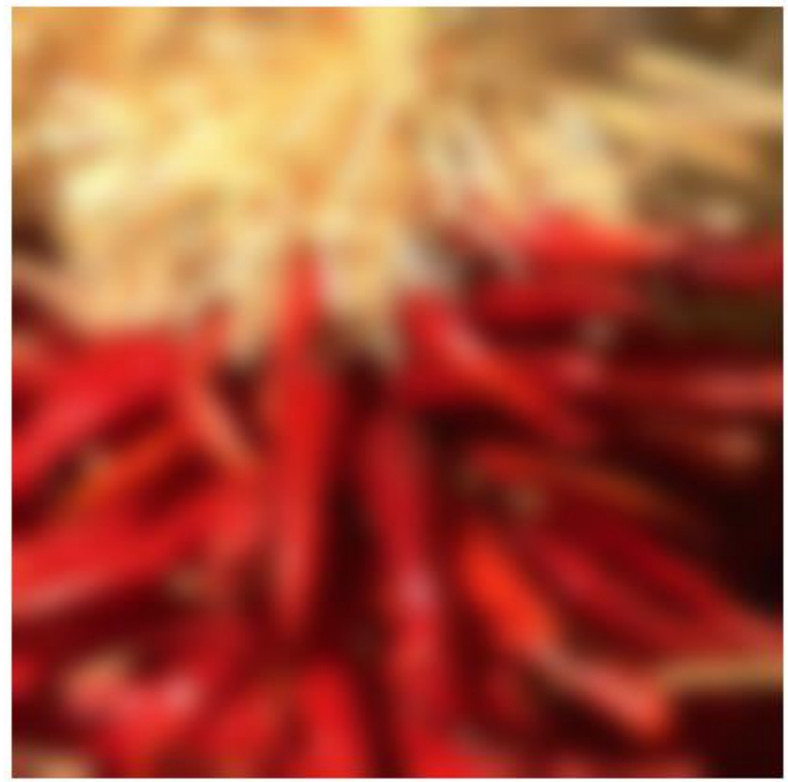
- Values at edges should be near zero
- Gaussians have infinite extent…
- Rule of thumb for Gaussian: set filter half-width to about 3 σ

James Hays

Practical matters

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



Source: S. Marschner

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.

Sobel filter visualization

- What happens to negative numbers?

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

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

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

Sobel

`plt.imshow( h );`

`plt.imshow( h + 0.5 );`

h(:,:,1) < 0

h(:,:,1) > 0

Think-Pair-Share

* = Convolution operator

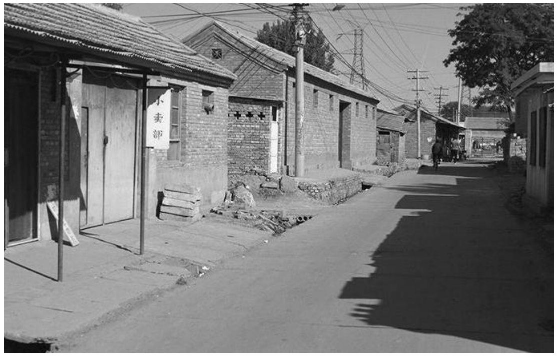
a) _ = D * B

b) A = _ * _

c) F = D * _

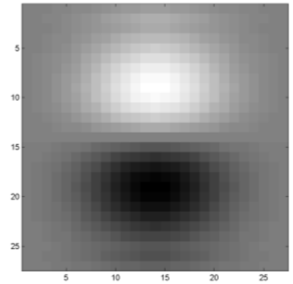
d) _ = D * D

D

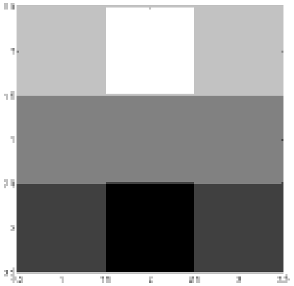


H

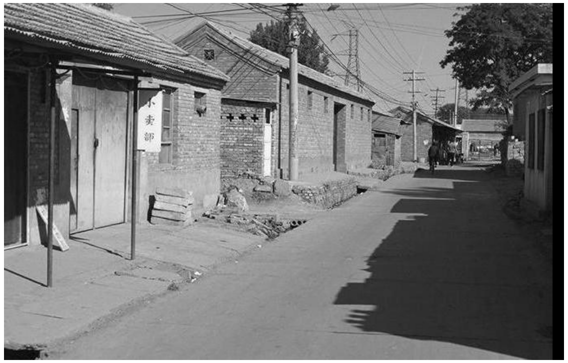
A



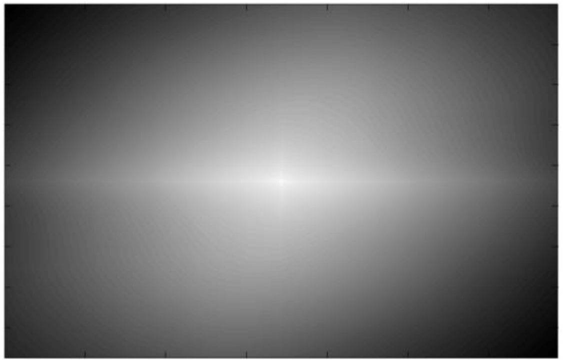
B



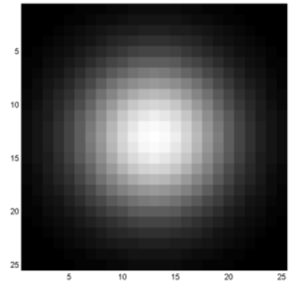
F



I

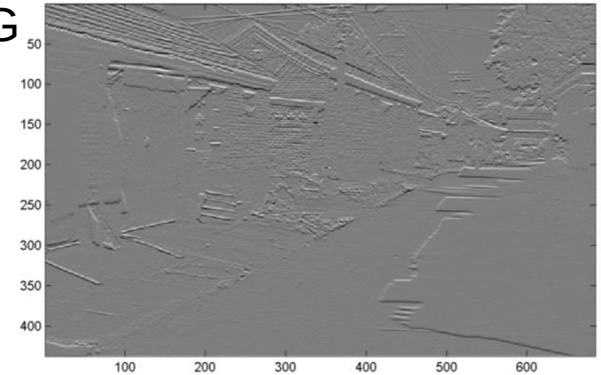


C



E

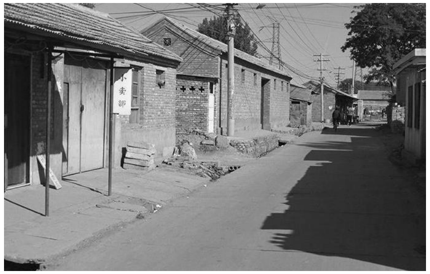
G



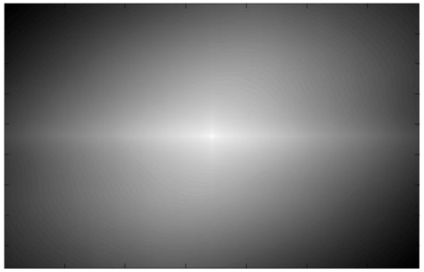
Hoiem

I = D * D

D (275 x 175 pixels)



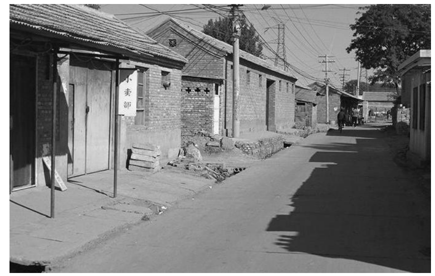
I (from slide – 275 x 175)



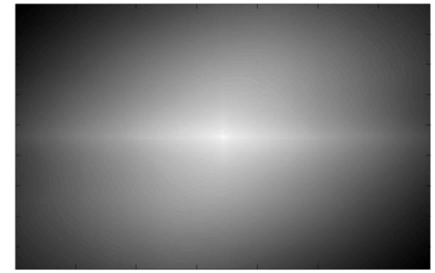
“…something to do with lack of content (black) at edges…”

I = D * D

D (275 x 175 pixels)

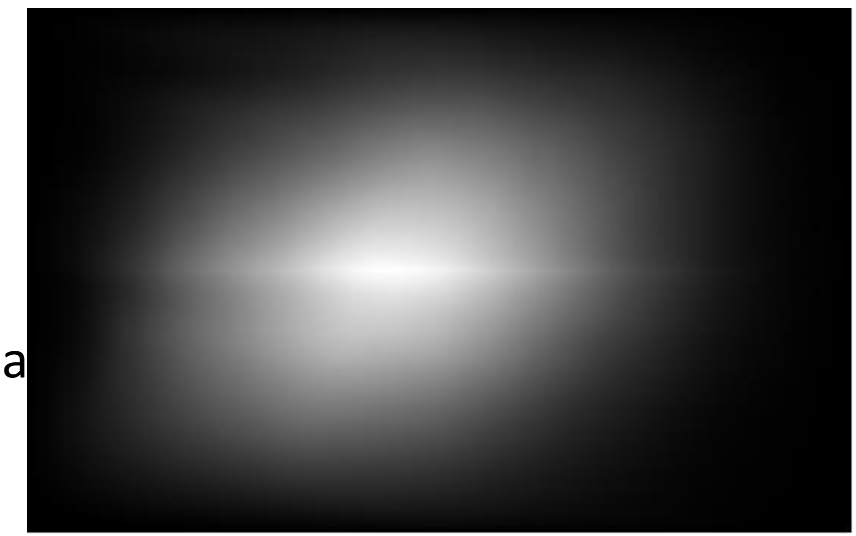


I (from slide – 275 x 175)



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

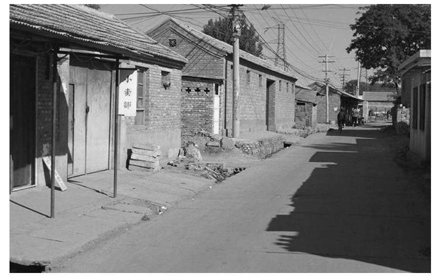
I_norm



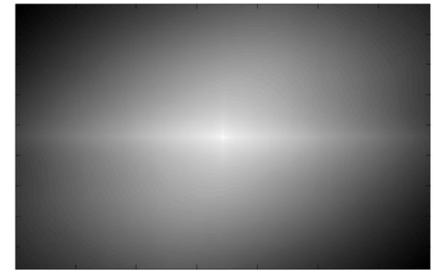
```
# Normalize for visualization
>> I_norm = (I – np.min(I)) / (np.ma
>> plt.imshow( I_norm )
```

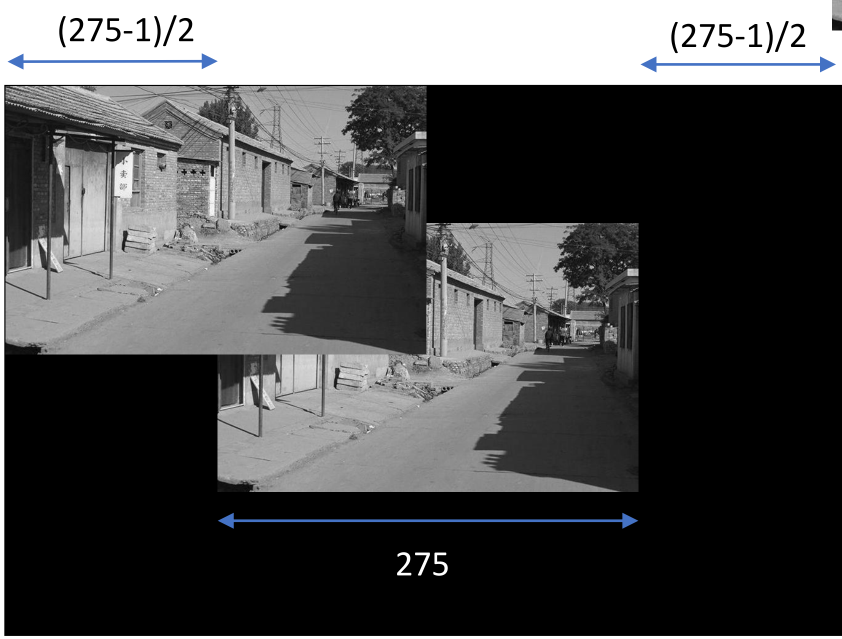
$I = D * D$

D (275 x 175 pixels)



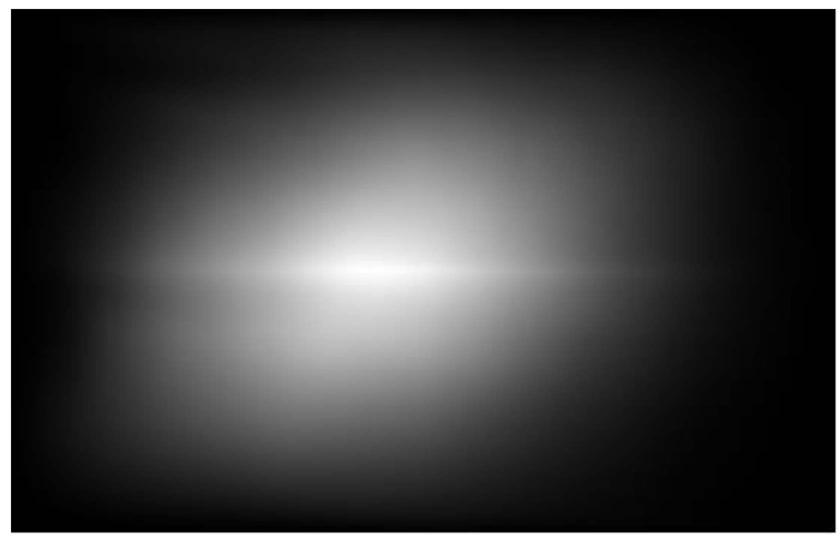
I (from slide – 275 x 175)





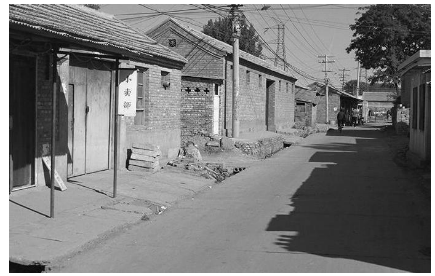
For x: 275 + (275-1)/2 + (275-1)/2
= 549

I_norm (549 x 349 pixels)

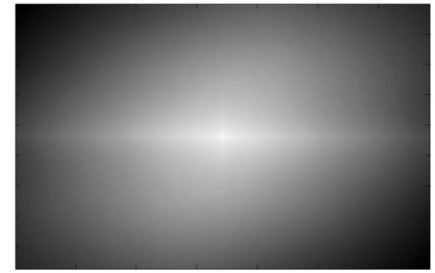


I = D * D

D (275 x 175 pixels)

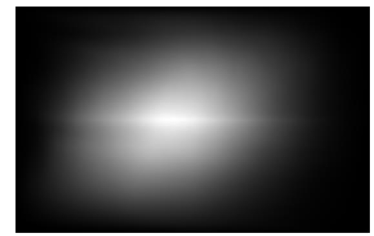


I (from slide – 275 x 175)



>> I = convolve2d(D, D, mode='full')

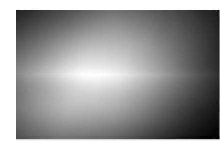
(Default; pad with zeros)



549 x 349

>> I = convolve2d(D, D, mode='same')

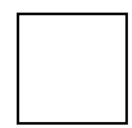
(Return same size as D)



275 x 175

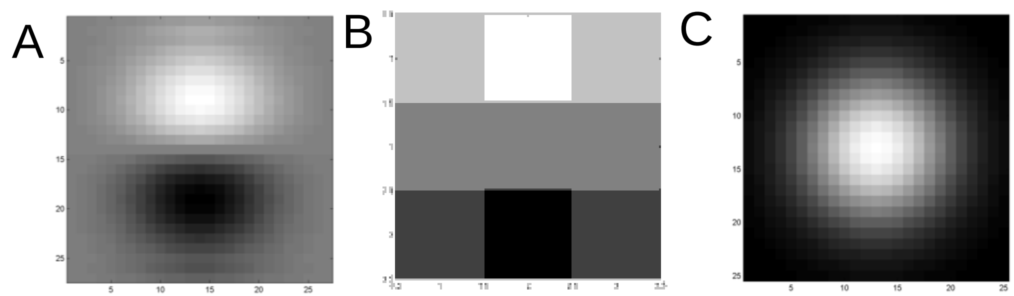
>> I = convolve2d(D, D, mode='valid')

(No padding)



Value = 10528.3
1x1

`A = B * C` – “because it kind of looks like it.”



C is a Gaussian filter
(or something close to it it),
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.

Filtering: Correlation and Convolution

- 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

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

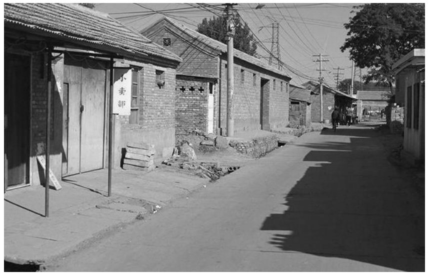
```
e.g., h = scipy.signal.convolve2d(f,I)
```

Convolution is the same as correlation with a 180° rotated filter kernel.
Correlation and convolution are identical when the filter kernel is symmetric.

James Hays

OK, so let's test this idea. Let's see if we can use correlation to 'find' the parts of the image that look like the filter.

D (275 x 175 pixels)



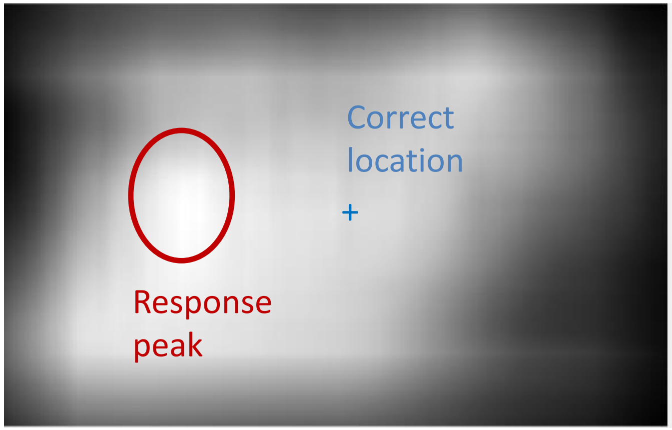
```
>> f = D[ 57:117, 107:167 ]
```

Expect response 'peak' in middle of I

f
61 x 61

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

I



Hmm…
That didn't work – why not?

[Thanks to Robert Collins @ Penn State]

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

```
>> 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

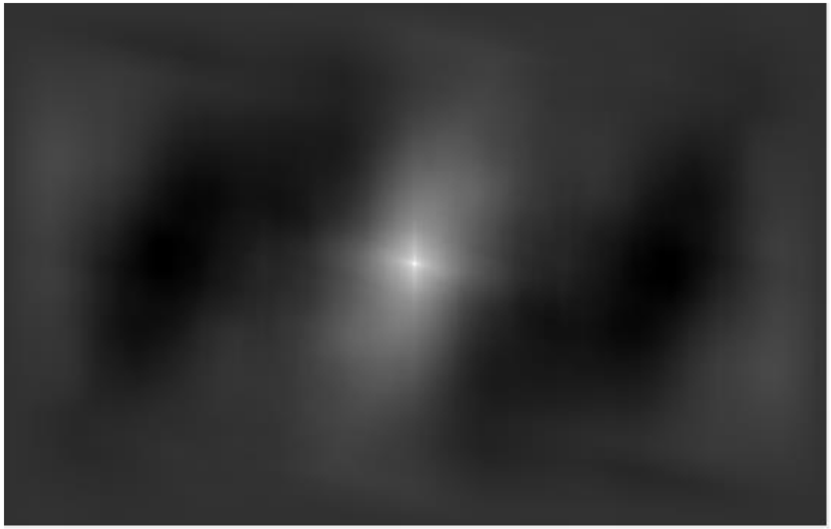
I2

Or even for our original example

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

D2 (275 x 175 pixels)

I3



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

NON-LINEAR FILTERS

Median filters

- 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

Image filtering - mean

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

1	1	1
1	1	1
1	1	1

$I[.,.]$

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	30	30				
				?					

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

Credit: S. Seitz

Image filtering - mean

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

$I[.,.]$

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

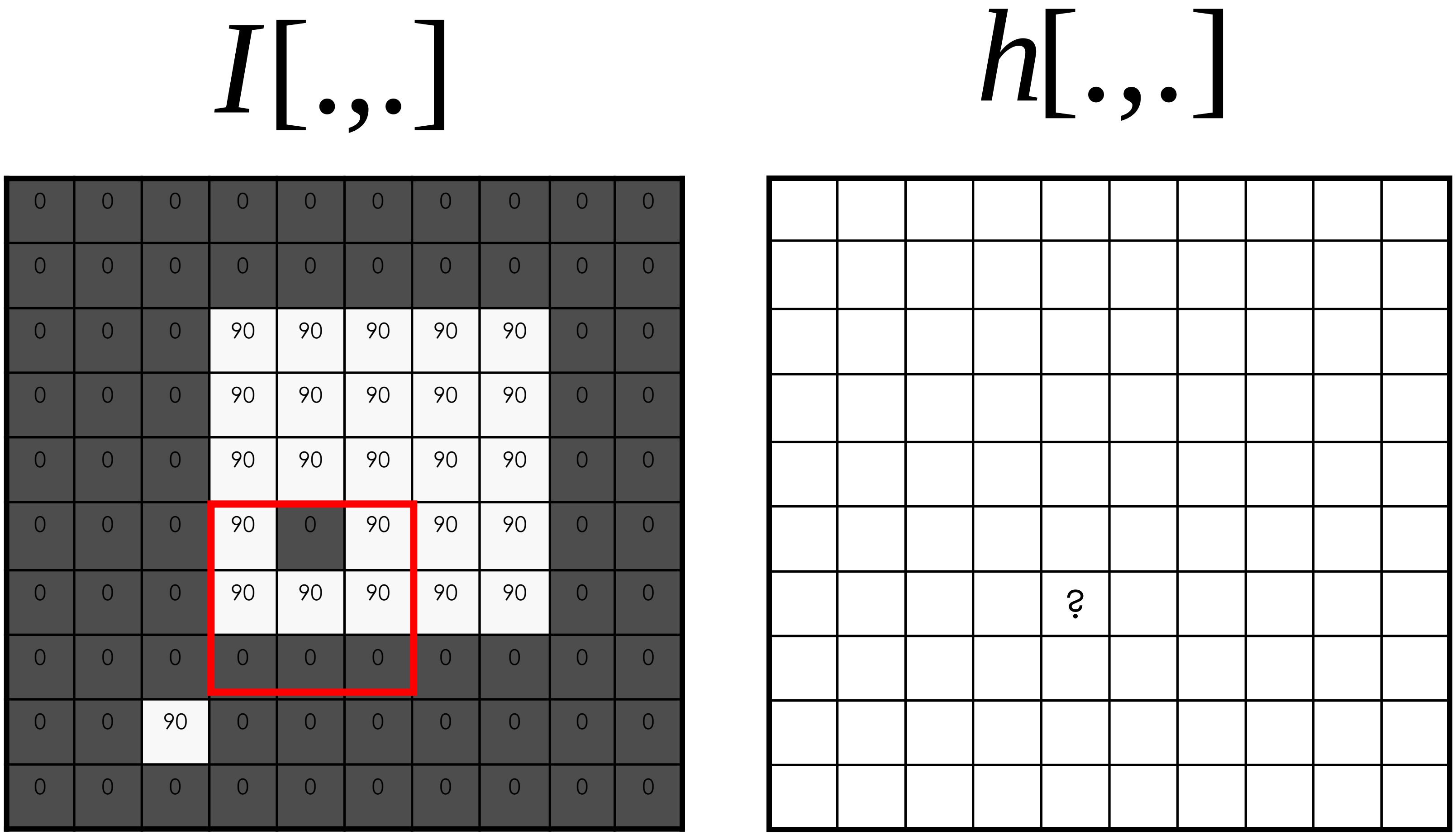
$h[.,.]$

	0	10	20	30	30				
				50					

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

Credit: S. Seitz

Median filter?

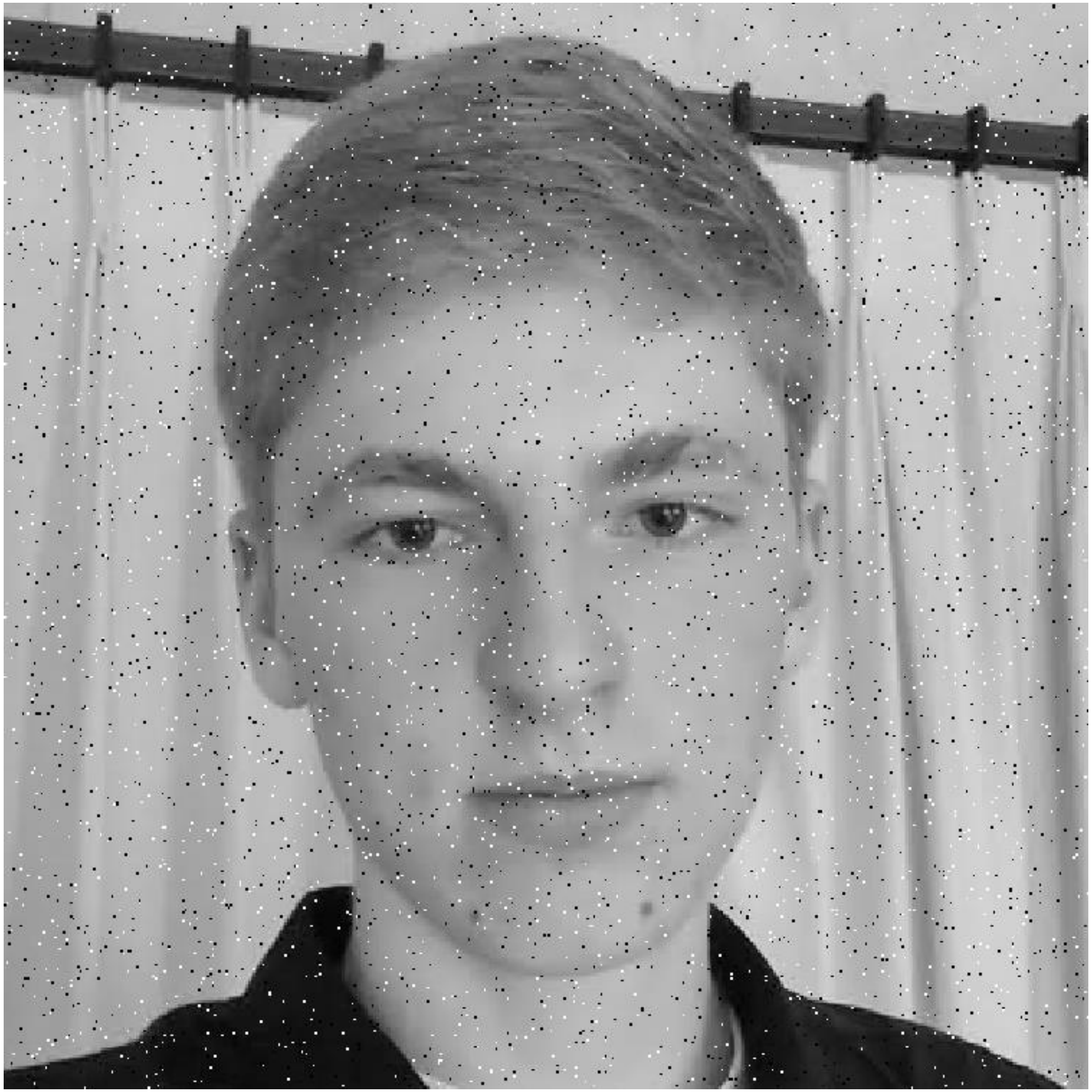


Credit: S. Seitz

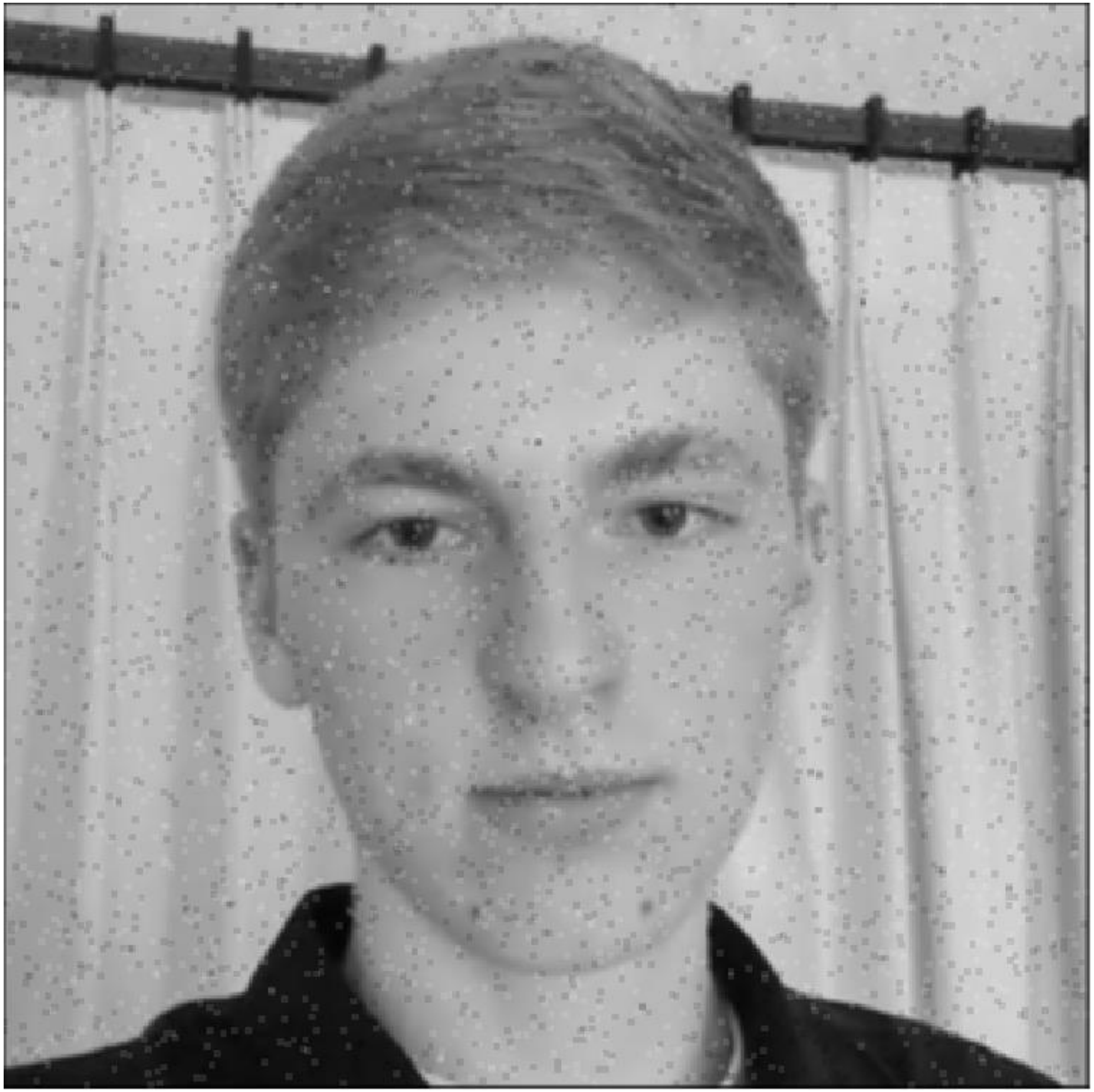
Median filters

- Operates over a window by selecting the median intensity in the window.
- What advantage does a median filter have over a mean filter?

Noise – Salt and Pepper Jack

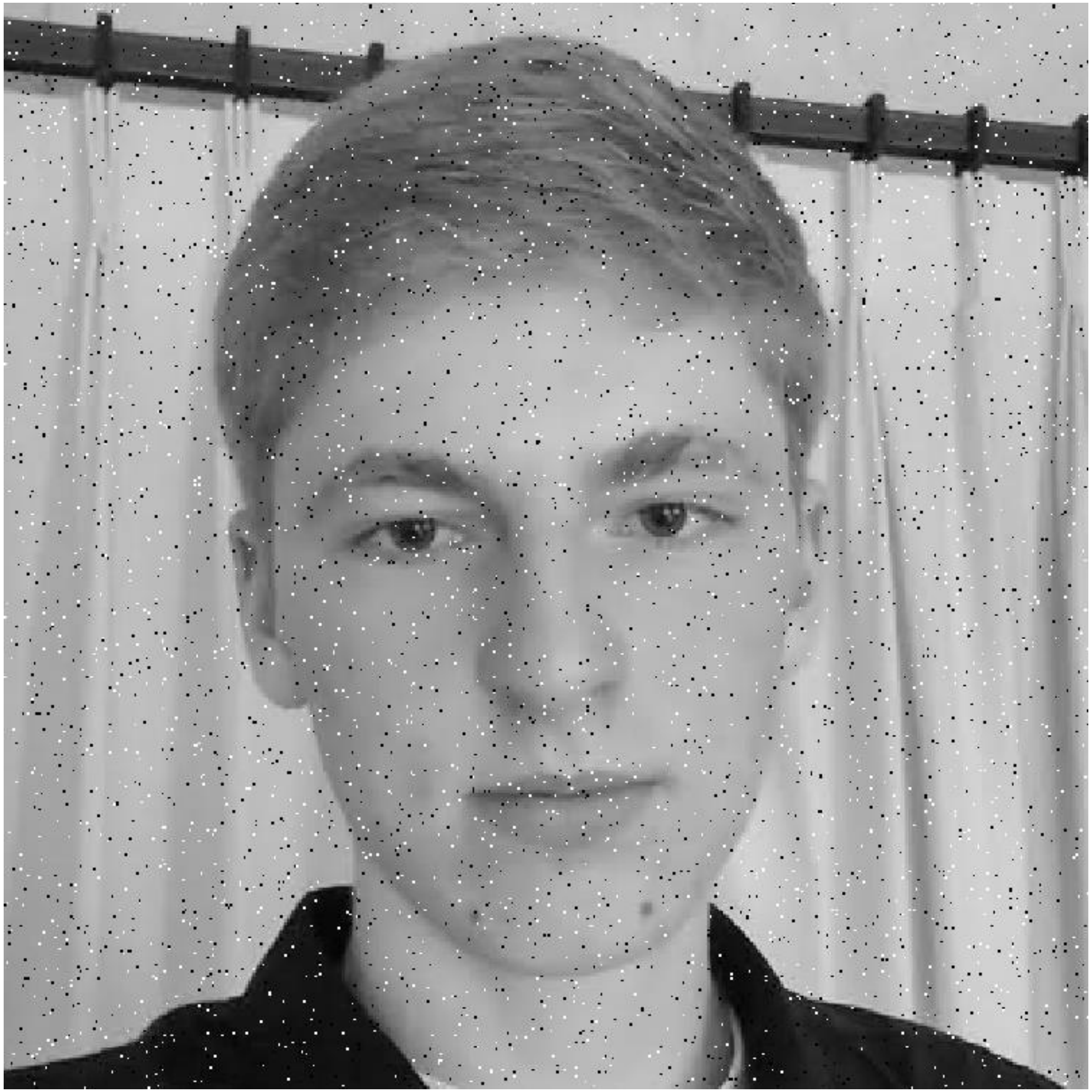


Mean Jack – 3 x 3 filter

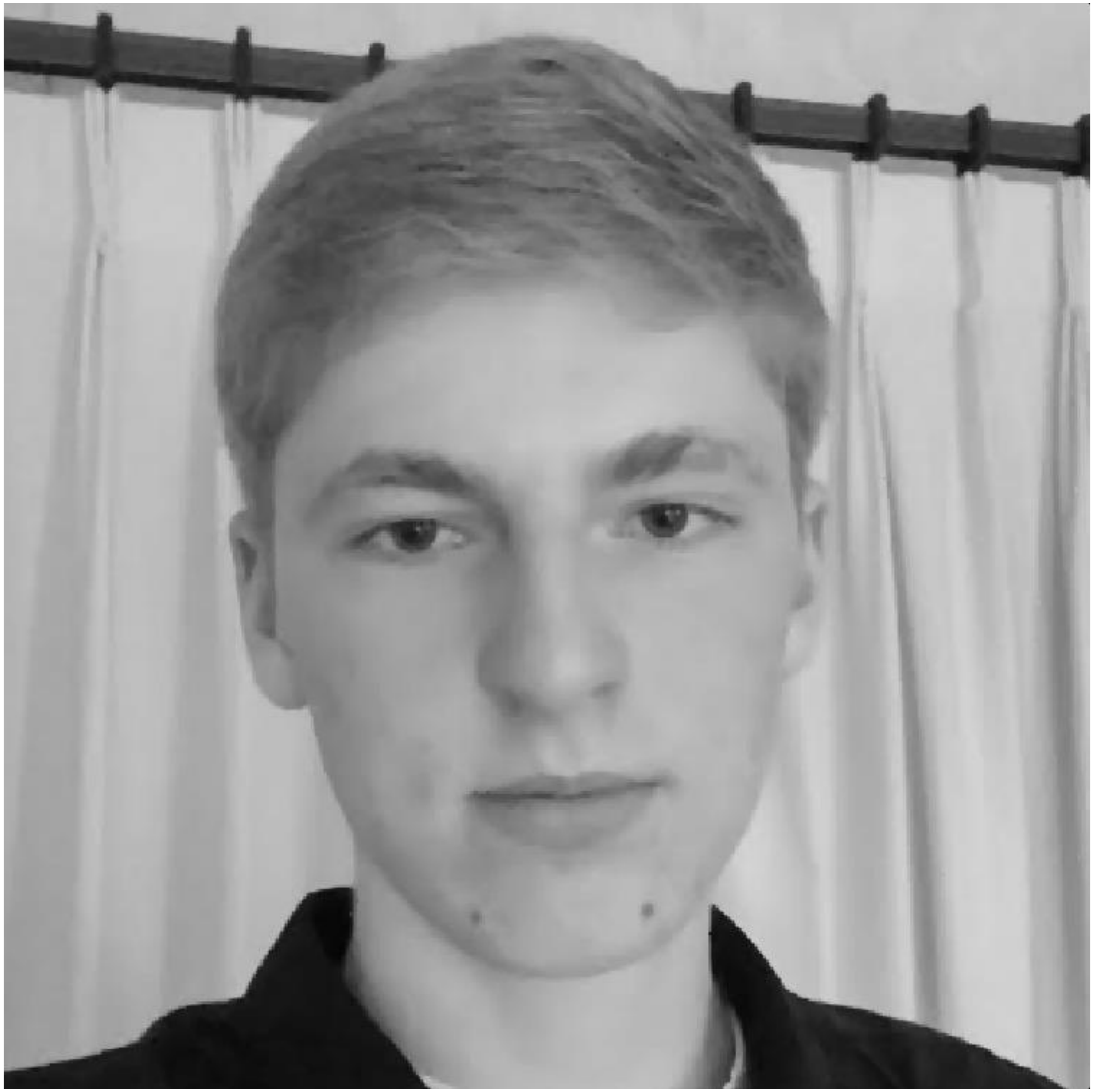


Very Mean Jack – 11 x 11 filter

Noisy – Salt and Pepper Jack



Median Jack – 3 x 3



Very Median Jack – 11 x 11

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?

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?

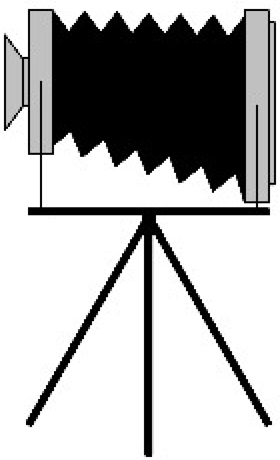
Interpretation: Median filtering is sorting.

Ben Thomas

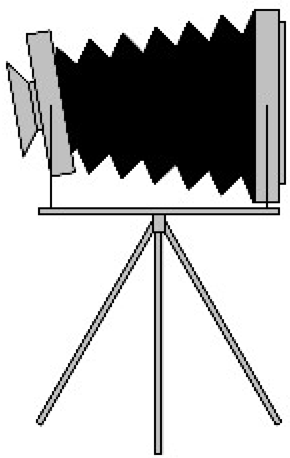
Tilt-shift photography

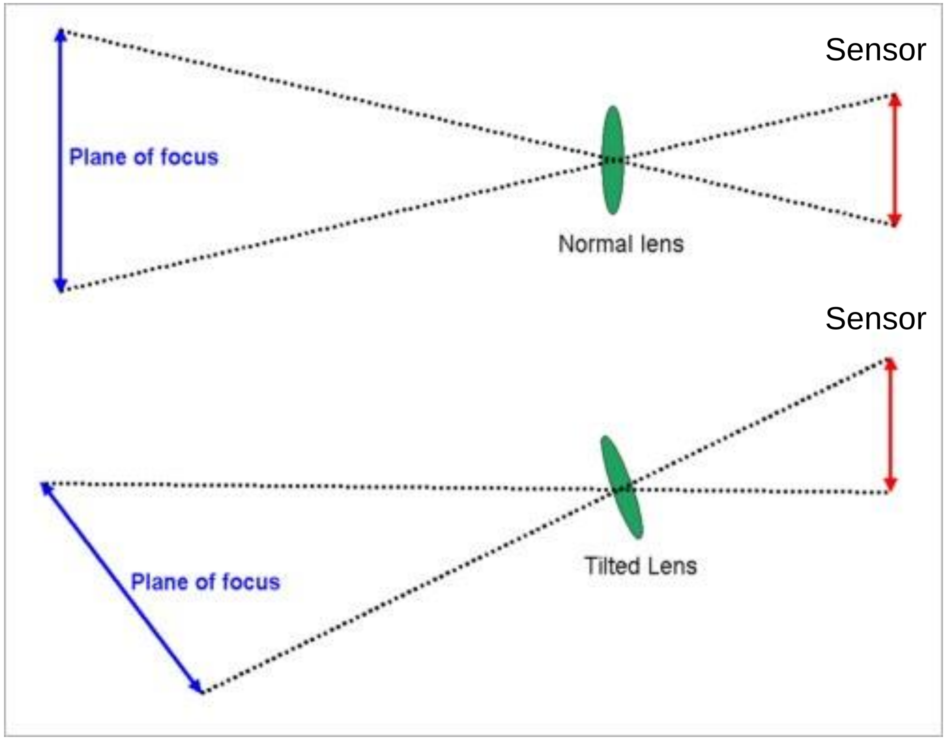
Tilt shift camera

Shift

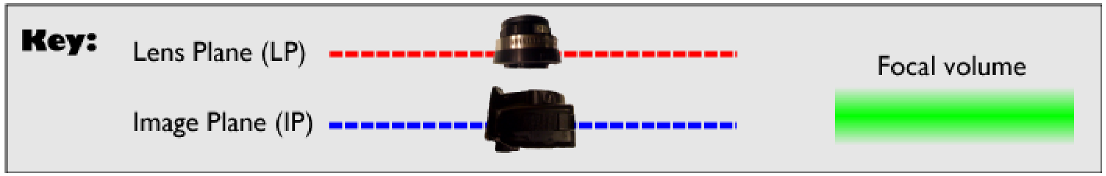


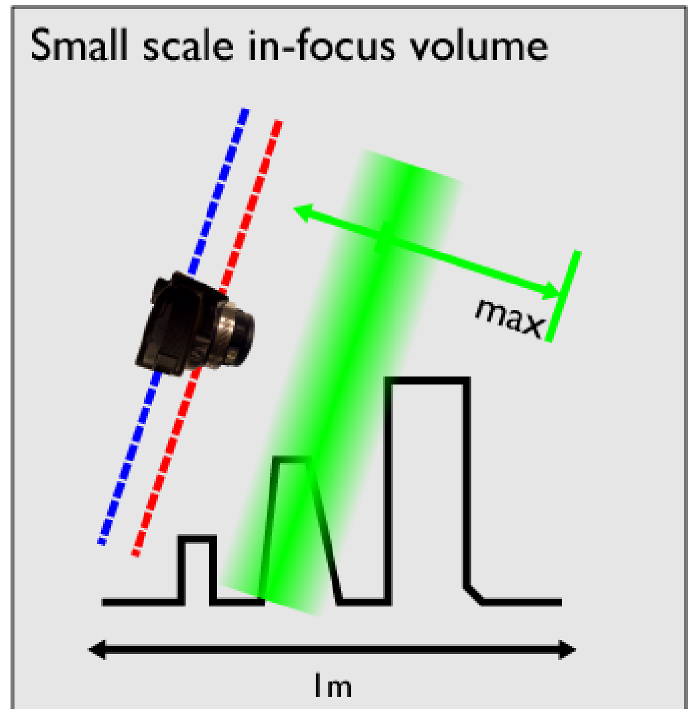
Tilt





Macro photography





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 point


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
- For now: transform to Hue, Saturation, Value instead of RGB

Next class: Thinking in Frequency