

CS 2770: Filters

PhD. Nils Murrugarra-Llerena
nem177@pitt.edu



Topics

- Filters: motivation, math and properties
- Types of filters
 - Linear
 - Smoothing
 - Other
 - Non-linear
 - Median
- Applications of filters
 - Texture representation with filters
 - Anti-aliasing for image subsampling



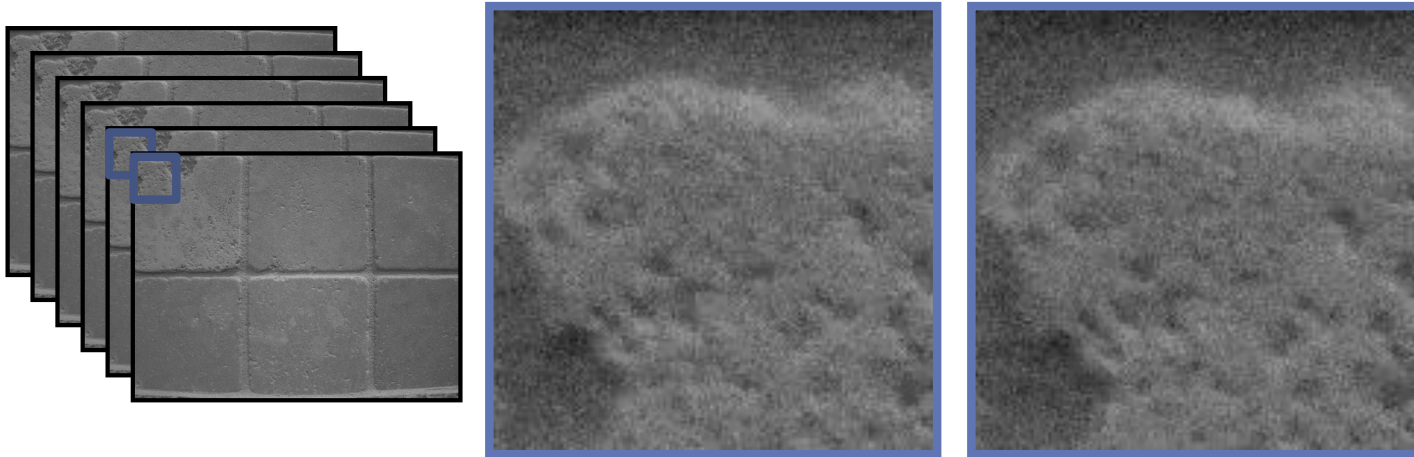
How images are represented

- Color images represented as a matrix with multiple channels (=1 if grayscale)
- Suppose we have a NxM RGB image called “im”
 - $\text{im}(0,0,0)$ = top-left pixel value in R-channel
 - $\text{im}(y, x, b)$ = y pixels **down (rows)**, x pixels **to right (cols)** in bth channel
 - $\text{im}(N, M, 3)$ = bottom-right pixel in B-channel
- `cv2.imread(filename)` returns a uint8 image (values 0 to 255)

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

Adapted from Derek Hoiem

Enter Noise



- The same object will look very different across images
- Even multiple images of same static scene won't be identical
- How could we reduce the noise, i.e., give an estimate of the true intensities?
- What if there's only one image?

Common types of noise

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



Original



Salt and pepper noise

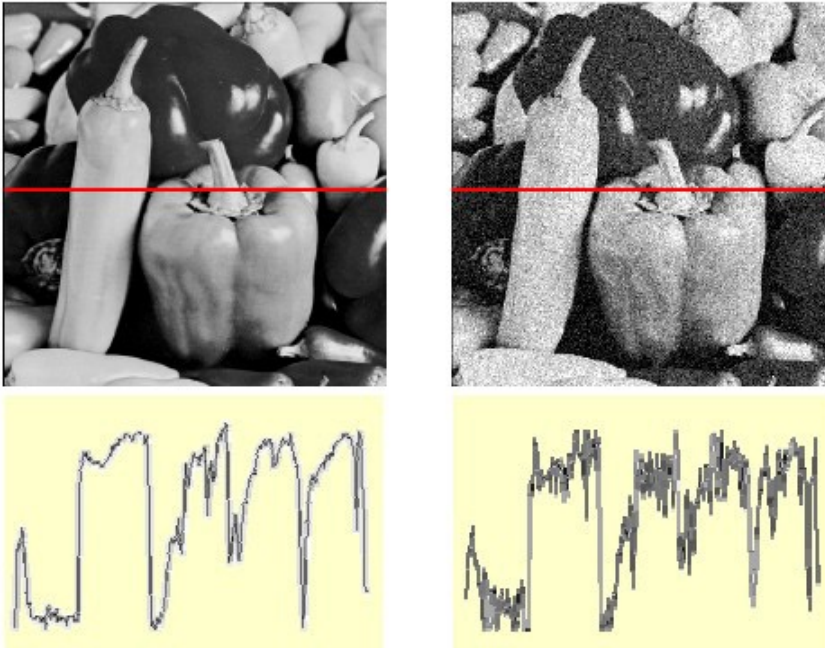


Impulse noise



Gaussian noise

Gaussian noise



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

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

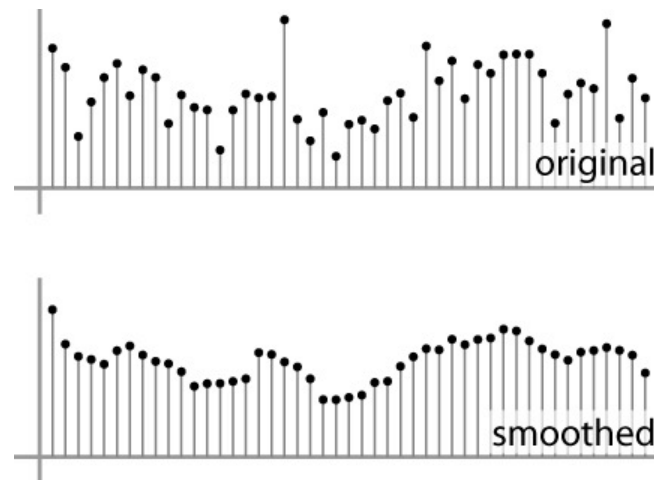
```
>> noise = np.random.rand( *im.shape ) * sigma
>> im_noise = im + noise
```

What is impact of the sigma?

[\[Github repo\]](#)

First attempt at a solution

- Let's replace each pixel with an **average of all the values in its neighborhood**
- Moving average in 1D:



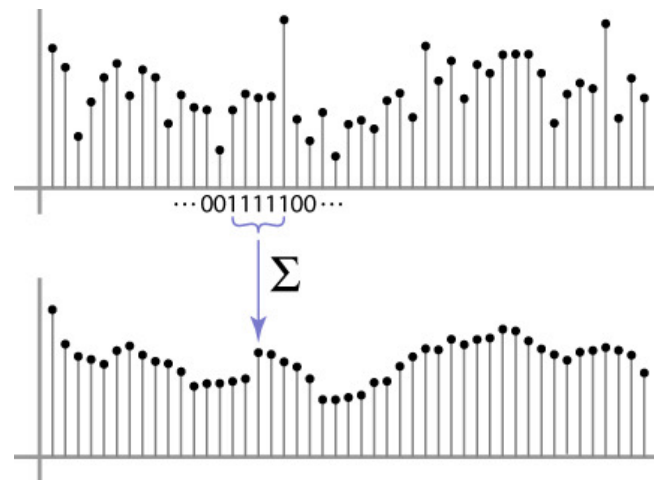
Source: S. Marschner

First attempt at a solution

- Let's replace each pixel with an average of all the values in its neighborhood
- Why/when will this work?
- Assumptions:
 - Expect pixels to be like their neighbors
 - Expect noise processes to be independent from pixel to pixel

Weighted Moving Average

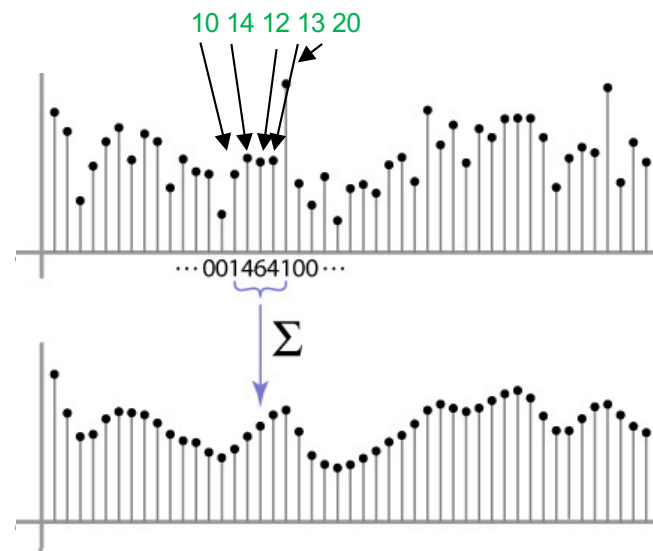
- Can add weights to our moving average
- *Weights* $[1, 1, 1, 1, 1] / 5$



Source: S. Marschner

Weighted Moving Average

- Non-uniform weights [1, 4, 6, 4, 1] / 16



Central pixel =
 $(10 \cdot 1 +$
 $14 \cdot 4 +$
 $12 \cdot 6 +$
 $13 \cdot 4 +$
 $20 \cdot 1) / 16$

Adapted from S. Marschner

Moving Average In 2D

$$F[x, y]$$

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

$$G[x, y]$$

0									

Moving Average In 2D

 $F[x, y]$

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

 $G[x, y]$

0	10								

Source: S. Seitz

Moving Average In 2D

$$F[x, y]$$

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

$$G[x, y]$$

	0	10	20						

Moving Average In 2D

$$F[x, y]$$

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

$$G[x, y]$$

	0	10	20	30					

Moving Average In 2D

$$F[x, y]$$

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

$$G[x, y]$$

	0	10	20	30	30				

Moving Average In 2D

 $F[x, y]$

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

 $G[x, y]$

	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	

In what ways is this output good/expected?
In what ways is it bad (what was lost)?

Source: S. Seitz

Image Filtering

- Compute a function of the local neighborhood at each pixel in the image
 - Function specified by a “filter” or mask saying how to combine values from neighbors
 - Element-wise multiplication of filter and image patch
- Uses of filtering:
 - Enhance an image (denoise, resize, etc)
 - Extract information (texture, edges, etc)
 - Detect patterns (template matching)



Adapted from Derek Hoiem

Correlation Filtering

Non-weighted, averaging window size is $2k+1 \times 2k+1$:

$$G[i, j] = \frac{1}{(2k+1)^2} \sum_{u=-k}^k \sum_{v=-k}^k F[i+u, j+v]$$

Attribute **uniform weight** to each pixel
Loop over all pixels in **neighborhood around image pixel $F[i,j]$**

Now generalize to allow **different weights** depending on neighboring pixel's relative position:

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k \underbrace{H[u, v]}_{\text{Non-uniform weights}} F[i+u, j+v]$$

Correlation Filtering

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i + u, j + v]$$

This is called **cross-correlation**, denoted

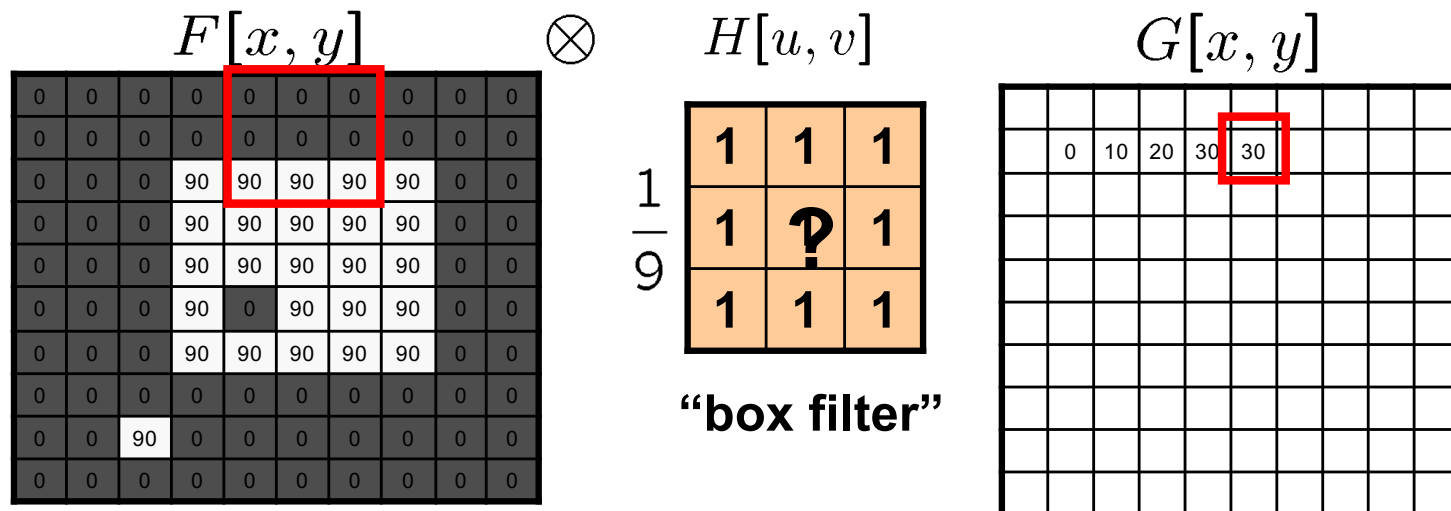
$$G = H \otimes F$$

Filtering an image: replace each pixel with a linear combination of its neighbors.

The filter “**kernel**” or “**mask**” $H[u, v]$ is the prescription for the weights in the linear combination.

Averaging Filtering

- What values belong in the kernel H for the moving average example?



$$G = H \otimes F$$

Smoothing by averaging



depicts box filter:
white = high value, black = low value



original

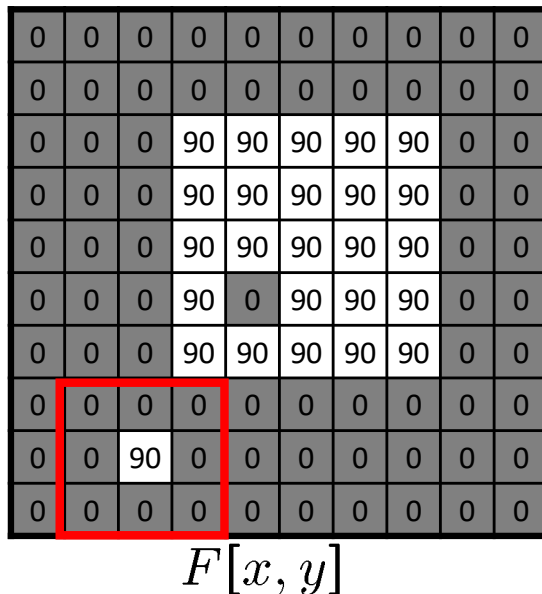


filtered

What if the filter size was 5 x 5 instead of 3 x 3?

Gaussian Filter

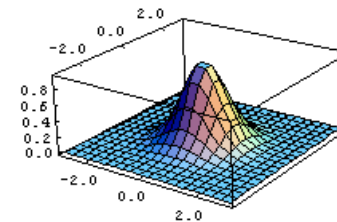
- What if we want nearest neighboring pixels to have the most influence on the output?



$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} H[u, v]$$

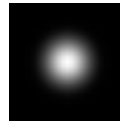
This kernel is an approximation of a 2d Gaussian function:

$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$



- Removes high-frequency components from the image (“low-pass filter”).

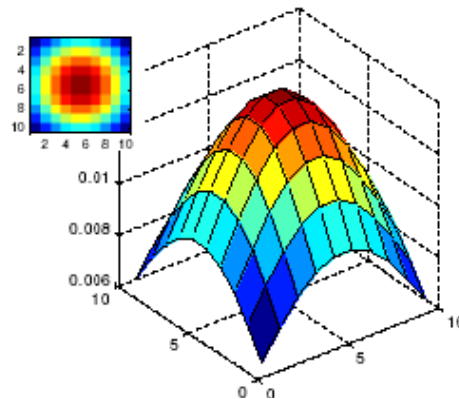
Smoothing with a Gaussian



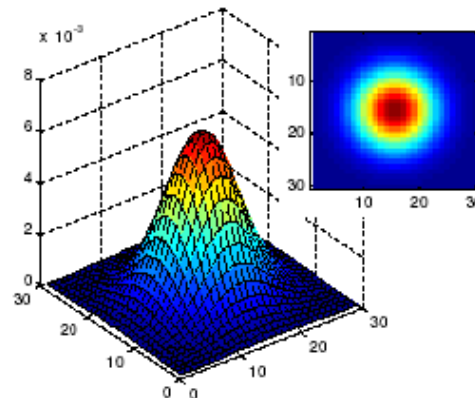
Vs box filter

Gaussian Filters

- What parameters matter here?
- **Size** of kernel or mask
 - Note, Gaussian function has infinite support, but discrete filters use finite kernels



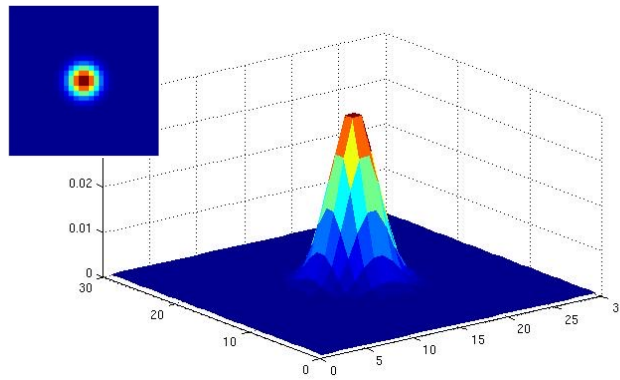
$\sigma = 5$ with
10 x 10
kernel



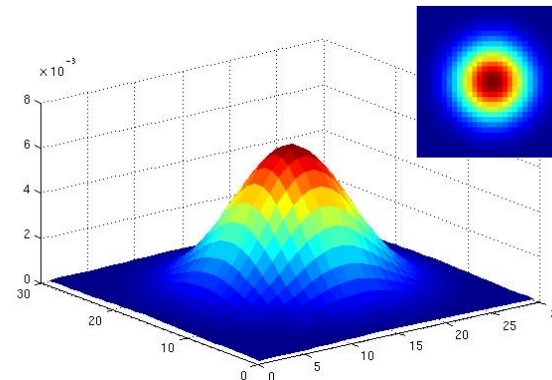
$\sigma = 5$ with
30 x 30
kernel

Gaussian Filters

- What parameters matter here?
- **Variance** of Gaussian: determines extent of smoothing



$\sigma = 2$ with
30 x 30
kernel

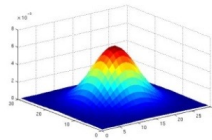


$\sigma = 5$ with
30 x 30
kernel

Gaussian Filters in Python

```
>> hsize = 10;  
>> sigma = 5;  
>> filter = fspecial_gauss(hsize, sigma)
```

```
>> plt.matshow(filter)
```



```
>> filt_im = ndimage.convolve(im, filter, mode='constant') #  
correlation  
>> cv2.imshow(outim);
```

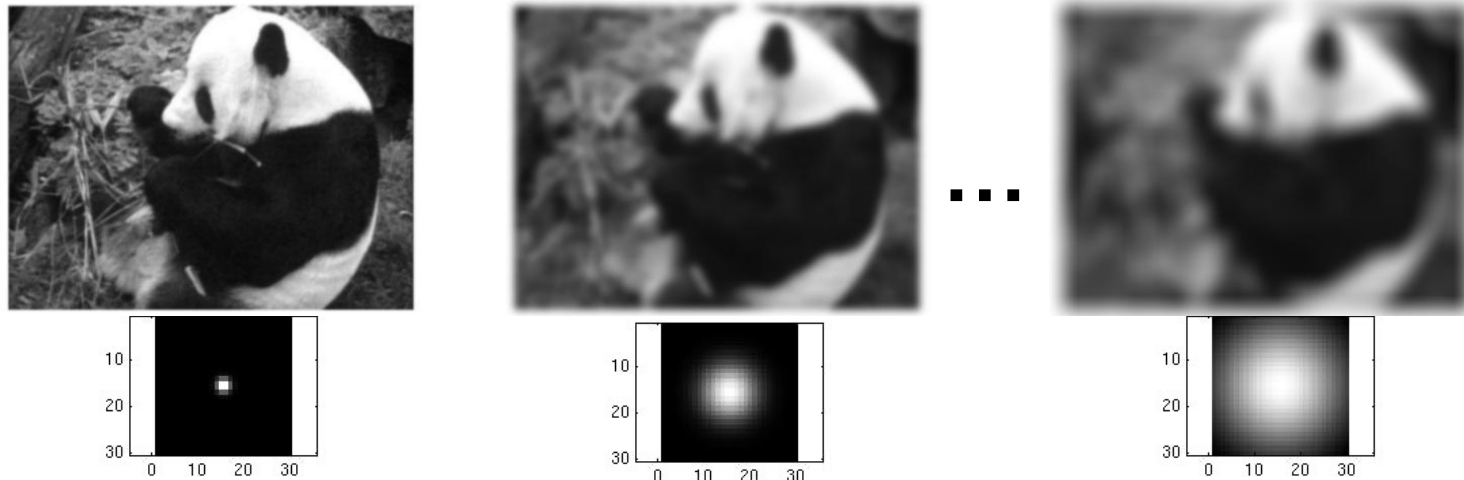


[Github repo]

outim

Smoothing with a Gaussian

Parameter σ is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.

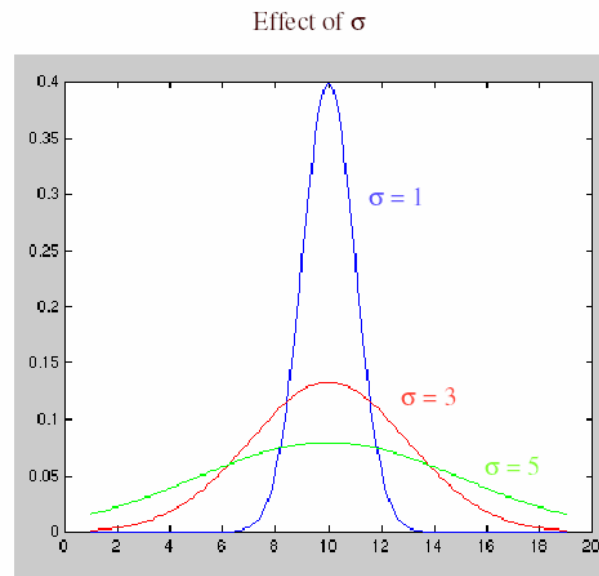


```
for sigma=1:3:10
    h = fspecial_gaus(hsize, sigma);
    out = ndimage.convolve(im, h);
    cv2.imshow(out);
end
```

Gaussian Filters

How big should the filter be?

- Values at edges should be near zero
- Rule of thumb for Gaussian: set filter half-width (*hsize*) to 3σ



Source: Derek Hoiem

Properties of smoothing filters

- Smoothing
 - Values positive
 - Sum to 1 → overall intensity same as input
 - Amount of smoothing proportional to mask size
 - Remove “high-frequency” components; “low-pass” filter

Convolution

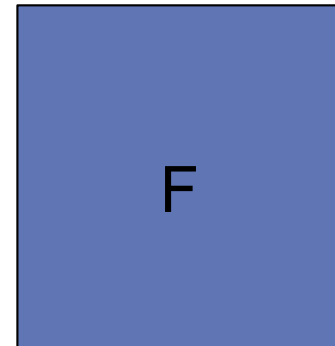
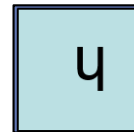
- Convolution:
 - Flip the filter in both dimensions (bottom to top, right to left)
 - Then apply cross-correlation

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i - u, j - v]$$

$$G = H \star F$$



*Notation for
convolution
operator*



Convolution vs correlation

Convolution

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i - u, j - v]$$

$$G = H \star F$$

Cross-correlation

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i + u, j + v]$$

$$G = H \otimes F$$

For a Gaussian or box filter, how will the outputs differ?

Convolution vs correlation

Cross-correlation

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i + u, j + v]$$

$$G = H \otimes F \quad u = -1, v = -1$$

F

5	2	5	4	4
5	200	3	200	4
1	5	5	4	4
5	5	1	1	2
200	1	3	5	200
1	200	200	200	1

(i, j)

Convolution

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i - u, j - v]$$

$$G = H \star F$$

H

.06	.12	.06
.12	.25	.12
.06	.12	.06

(0, 0)

Convolution vs correlation

Cross-correlation

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i + u, j + v]$$

$$G = H \otimes F$$

$$u = -1, v = -1 \\ v = 0$$

F

5	2	5	4	4
5	200	3	200	4
1	5	5	4	4
5	5	1	1	2
200	1	3	5	200
1	200	200	200	1

(i, j)

Convolution

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i - u, j - v]$$

$$G = H \star F$$

H

.06	.12	.06
.12	.25	.12
.06	.12	.06

(0, 0)

Convolution vs correlation

Cross-correlation

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i + u, j + v]$$

$$G = H \otimes F$$

$$u = -1, v = -1$$

$$v = 0$$

$$v = +1$$

F

5	2	5	4	4
5	200	3	200	4
1	5	5	4	4
5	5	1	1	2
200	1	3	5	200
1	200	200	200	1

(i, j)

Convolution

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i - u, j - v]$$

$$G = H \star F$$

H

.06	.12	.06
.12	.25	.12
.06	.12	.06

(0, 0)

Convolution vs correlation

Cross-correlation

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i + u, j + v]$$

$$G = H \otimes F$$

$$u = -1, v = -1$$

$$v = 0$$

$$v = +1$$

$$u = 0, v = -1$$

F

5	2	5	4	4
5	200	3	200	4
1	5	5	4	4
5	5	1	1	2
200	1	3	5	200
1	200	200	200	1

(i, j)

Convolution

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i - u, j - v]$$

$$G = H \star F$$

H

.06	.12	.06
.12	.25	.12
.06	.12	.06

(0, 0)

Convolution vs correlation

Cross-correlation

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i + u, j + v]$$

$$G = H \otimes F$$

$$u = -1, v = -1$$

F

5	2	5	4	4
5	200	3	200	4
1	5	5	4	4
5	5	1	1	2
200	1	3	5	200
1	200	200	200	1

(i, j)

Convolution

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i - u, j - v]$$

$$G = H \star F$$

H

.06	.12	.06
.12	.25	.12
.06	.12	.06

(0, 0)

Convolution vs correlation

Cross-correlation

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i + u, j + v]$$

$$G = H \otimes F$$

$$u = -1, v = -1 \\ v = 0$$

F

5	2	5	4	4
5	200	3	200	4
1	5	5	4	4
5	5	1	1	2
200	1	3	5	200
1	200	200	200	1

(i, j)

Convolution

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i - u, j - v]$$

$$G = H \star F$$

H

.06	.12	.06
.12	.25	.12
.06	.12	.06

(0, 0)

Convolution vs correlation

Cross-correlation

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i + u, j + v]$$

$$G = H \otimes F$$

$$u = -1, v = -1$$

$$v = 0$$

$$v = +1$$

F

5	2	5	4	4
5	200	3	200	4
1	5	5	4	4
5	5	1	1	2
200	1	3	5	200
1	200	200	200	1

(i, j)

Convolution

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i - u, j - v]$$

$$G = H \star F$$

H

.06	.12	.06
.12	.25	.12
.06	.12	.06

(0, 0)

Convolution vs correlation

Cross-correlation

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i + u, j + v]$$

$$G = H \otimes F$$

$$u = -1, v = -1$$

$$v = 0$$

$$v = +1$$

$$u = 0, v = -1$$

F

5	2	5	4	4
5	200	3	200	4
1	5	5	4	4
5	5	1	1	2
200	1	3	5	200
1	200	200	200	1

(i, j)

Convolution

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i - u, j - v]$$

$$G = H \star F$$

H

.06	.12	.06
.12	.25	.12
.06	.12	.06

(0, 0)

Properties of Convolution

- Commutative:

$$f * g = g * f$$

- Associative:

$$(f * g) * h = f * (g * h)$$

- Distributes over addition:

$$f * (g + h) = (f * g) + (f * h)$$


- Scalars factor out:


$$kf * g = f * kg = k(f * g)$$


- Identity:

$$\text{unit impulse } e = [\dots, 0, 0, 1, 0, 0, \dots]. \quad f * e = f$$

Predict the outputs using correlation filtering



$$* \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = ?$$


$$* \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} = ?$$


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

Practice with linear filters

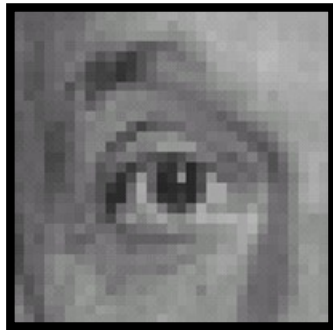


Original

0	0	0
0	1	0
0	0	0

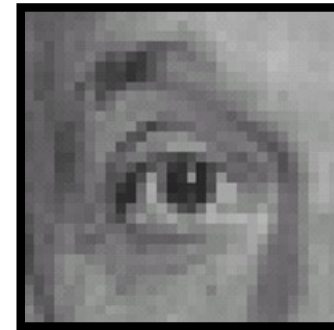
?

Practice with linear filters



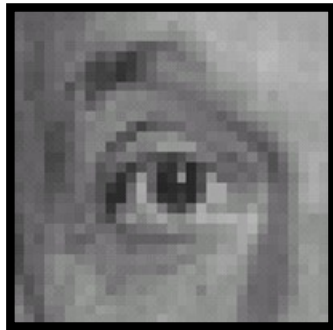
Original

0	0	0
0	1	0
0	0	0



**Filtered
(no change)**

Practice with linear filters

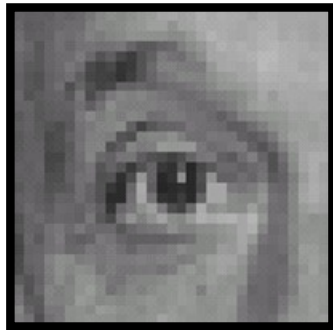


Original

0	0	0
0	0	1
0	0	0

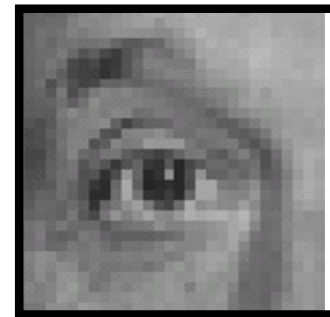
?

Practice with linear filters



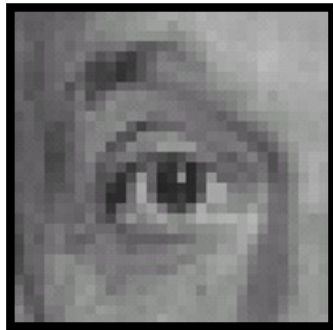
Original

0	0	0
0	0	1
0	0	0



**Shifted left
by 1 pixel
with
correlation**

Practice with linear filters



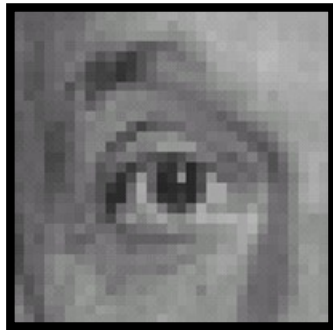
Original

$$\frac{1}{9}$$

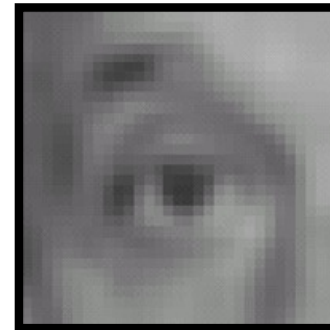
1	1	1
1	1	1
1	1	1

?

Practice with linear filters

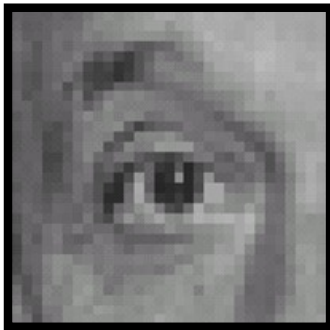


Original

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


**Blur (with a
box filter)**

Practice with linear filters



Original

0	0	0
0	2	0
0	0	0

—

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

?

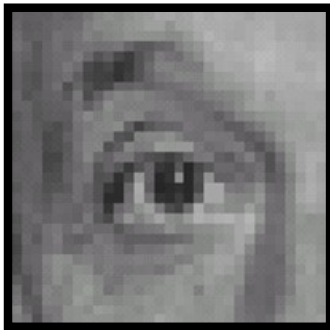
Practice with linear filters

$$\begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 2 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} - \frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} =$$

$$\underbrace{\begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array}}_{\text{Original image}} + \left(\begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} - \frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} \right)$$

Original image
Image minus blur = details

Practice with linear filters



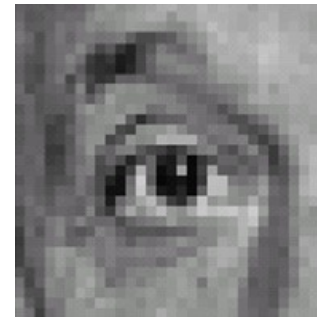
Original

0	0	0
0	2	0
0	0	0

−

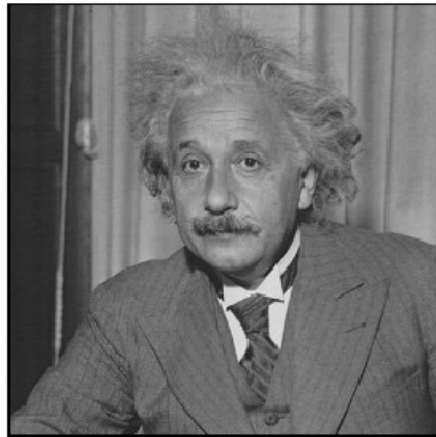
$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

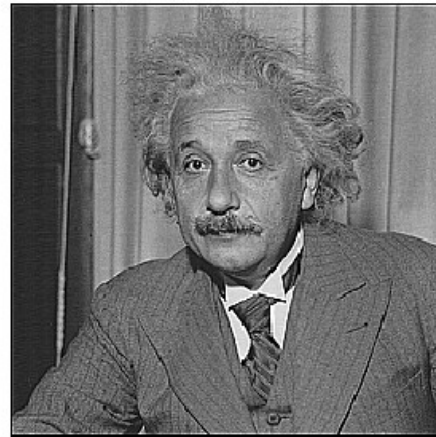


Sharpening filter:
accentuates differences with
local average

Sharpening



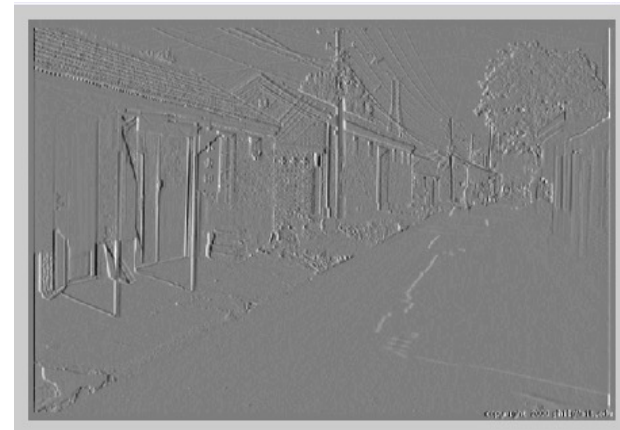
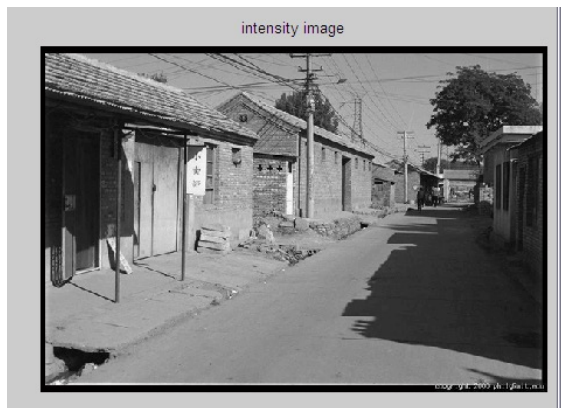
before



after

Filters for computing *gradients*

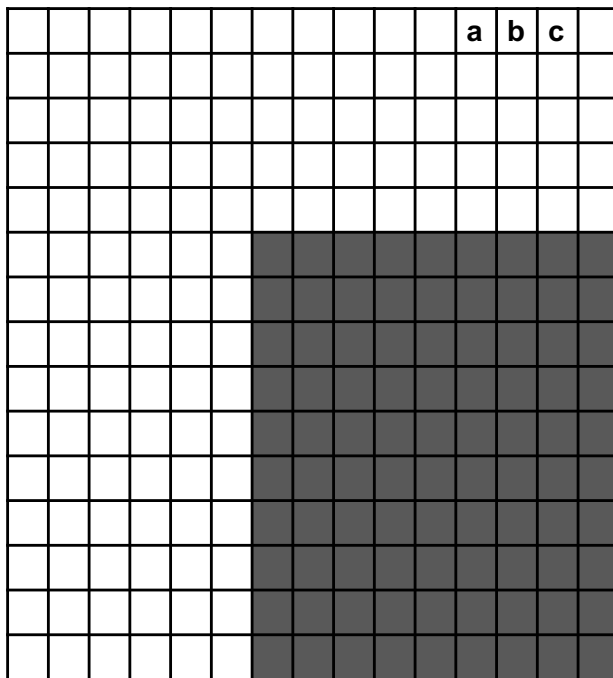
1	0	-1
2	0	-2
1	0	-1



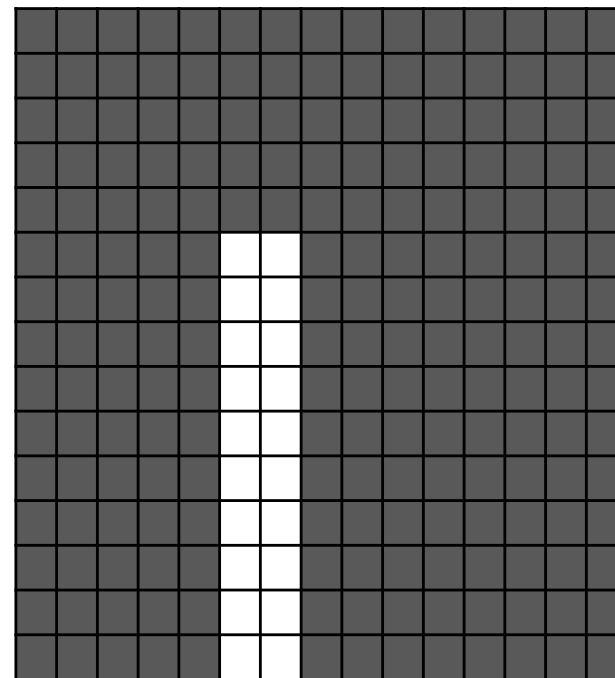
Adapted from Derek Hoiem

Filters for computing *gradients*

$$+1*a + 0*b + (-1)*c = a - c$$



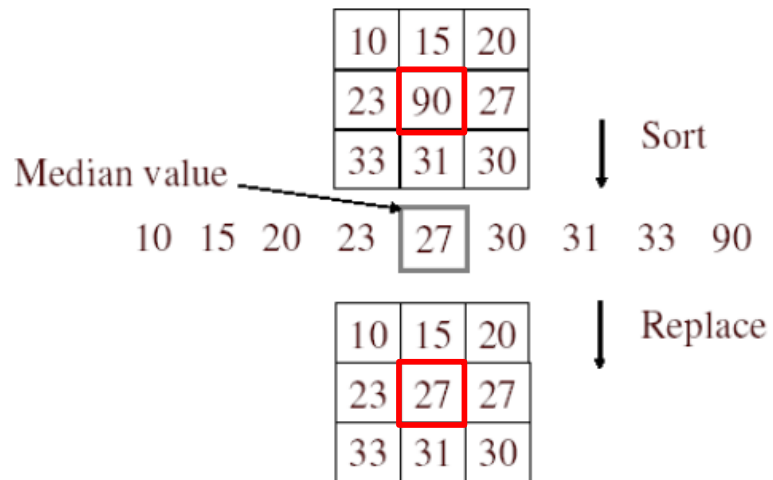
Input image
White = 1, black = 0



Filter: [+1 0 -1]

Output image

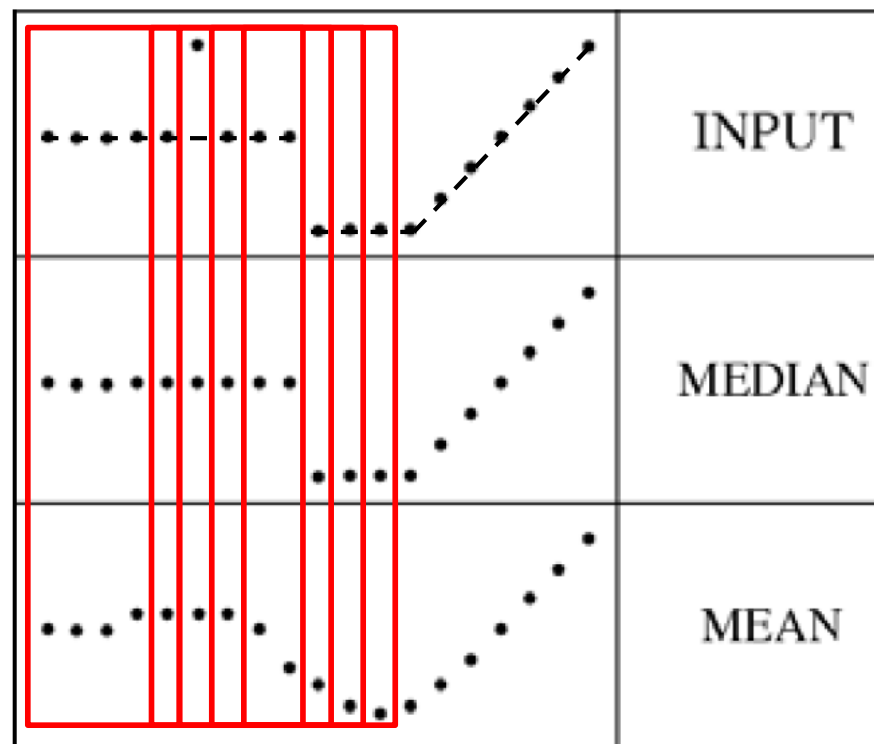
Median Filter



- No new pixel values introduced
- Removes spikes: good for impulse, salt & pepper noise
- Non-linear filter

Median Filter

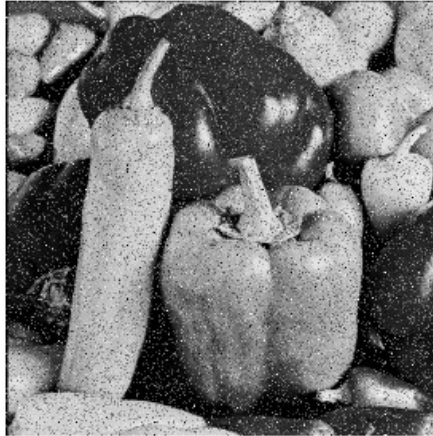
- Median filter is edge preserving



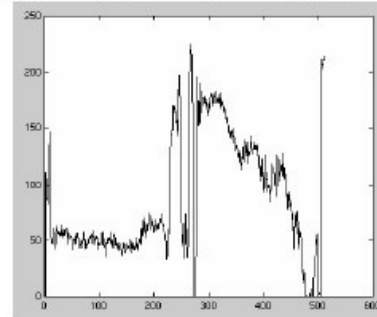
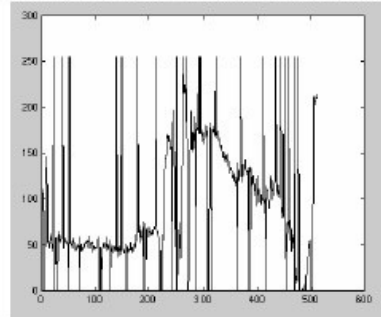
Adapted from Kristen Grauman

Median Filter

Salt and
pepper
noise



Median
filtered



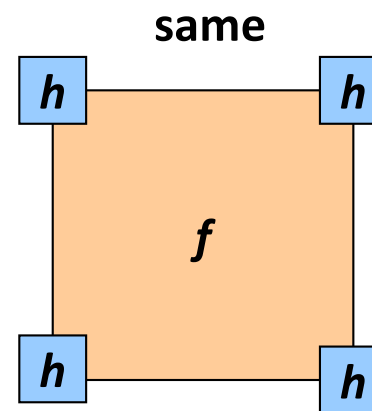
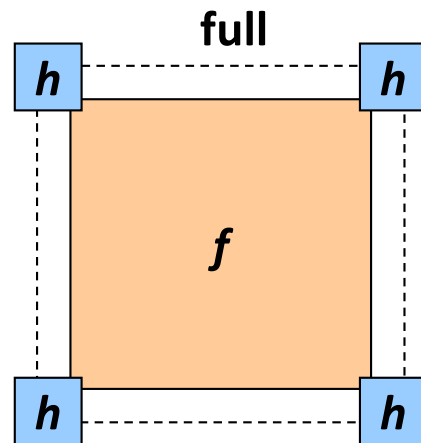
Plots of a row of the image

https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.median_filter.html#scipy.ndimage.median_filter

Source: M. Hebert

Boundary Issues

- What is the size of the output?
 - 'full': output size is larger than the size of f
 - 'same': output size is same as f (**Default for Python**)



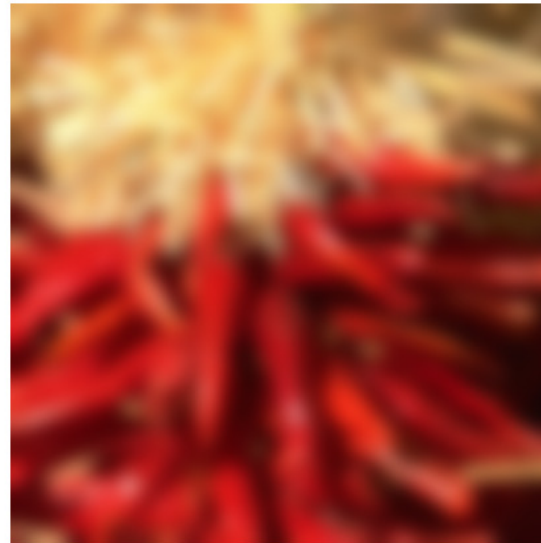
f = image
 h = filter

	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	

Adapted from S. Lazebnik

Boundary Issues

- What about near the edge?
 - the filter window might fall off the edge of the image (in 'same' or 'full')
 - need to extrapolate
 - methods:
 - clip filter (black)
 - wrap around
 - copy edge
 - reflect across edge



Source: S. Marschner

Separability

- In some cases, filter is separable, and we can factor into two steps:
 - Convolve all rows
 - Convolve all columns

Separability Example

2D filtering
(center location only)

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 2 & 3 & 3 \\ \hline 3 & 5 & 5 \\ \hline 4 & 4 & 6 \\ \hline \end{array}$$

filter

The filter factors
into an *outer* product
of 1D filters:

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} = \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline \end{array}$$

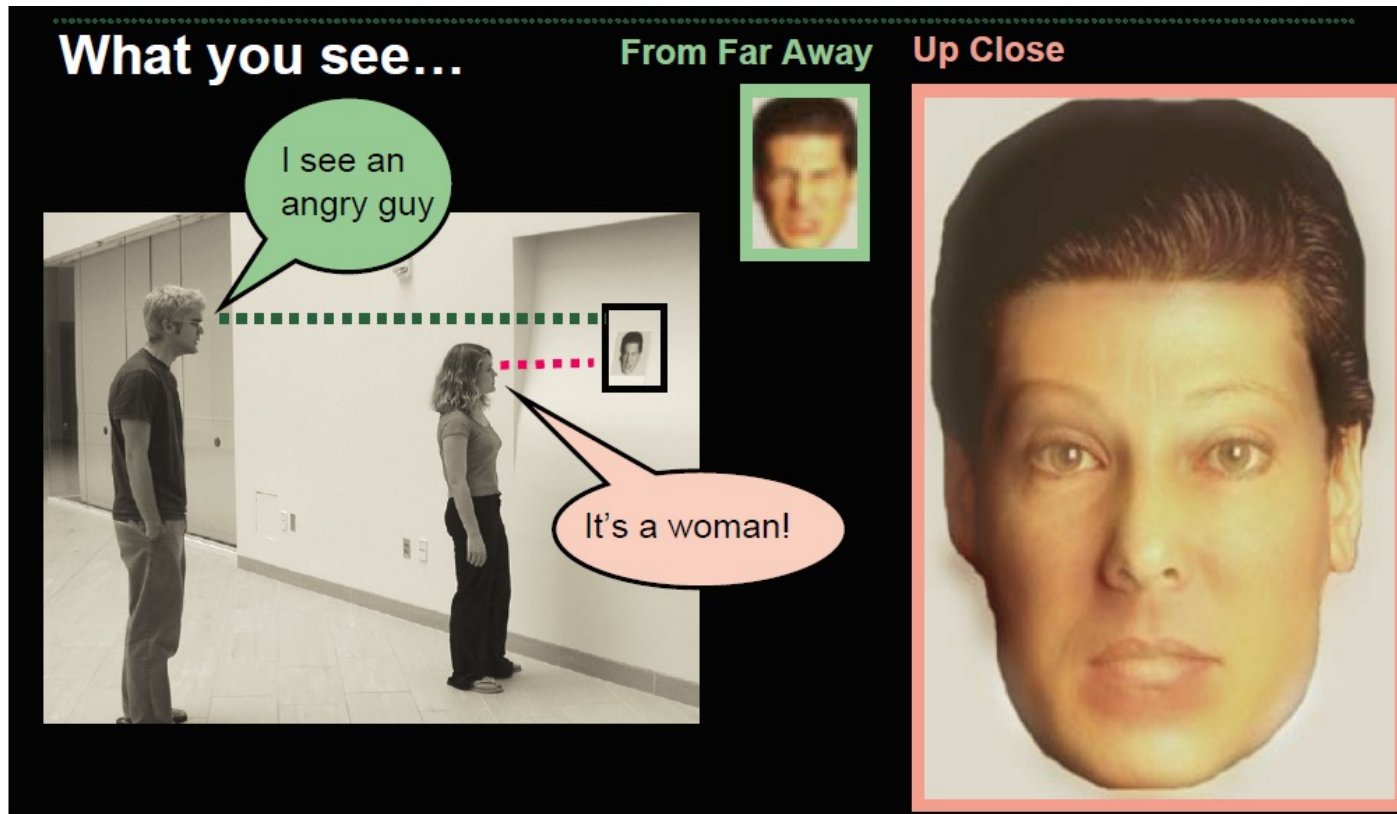
Perform filtering
along rows:

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 2 & 3 & 3 \\ \hline 3 & 5 & 5 \\ \hline 4 & 4 & 6 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 11 & 18 & 18 \\ \hline \end{array}$$

Followed by filtering
along the remaining column:

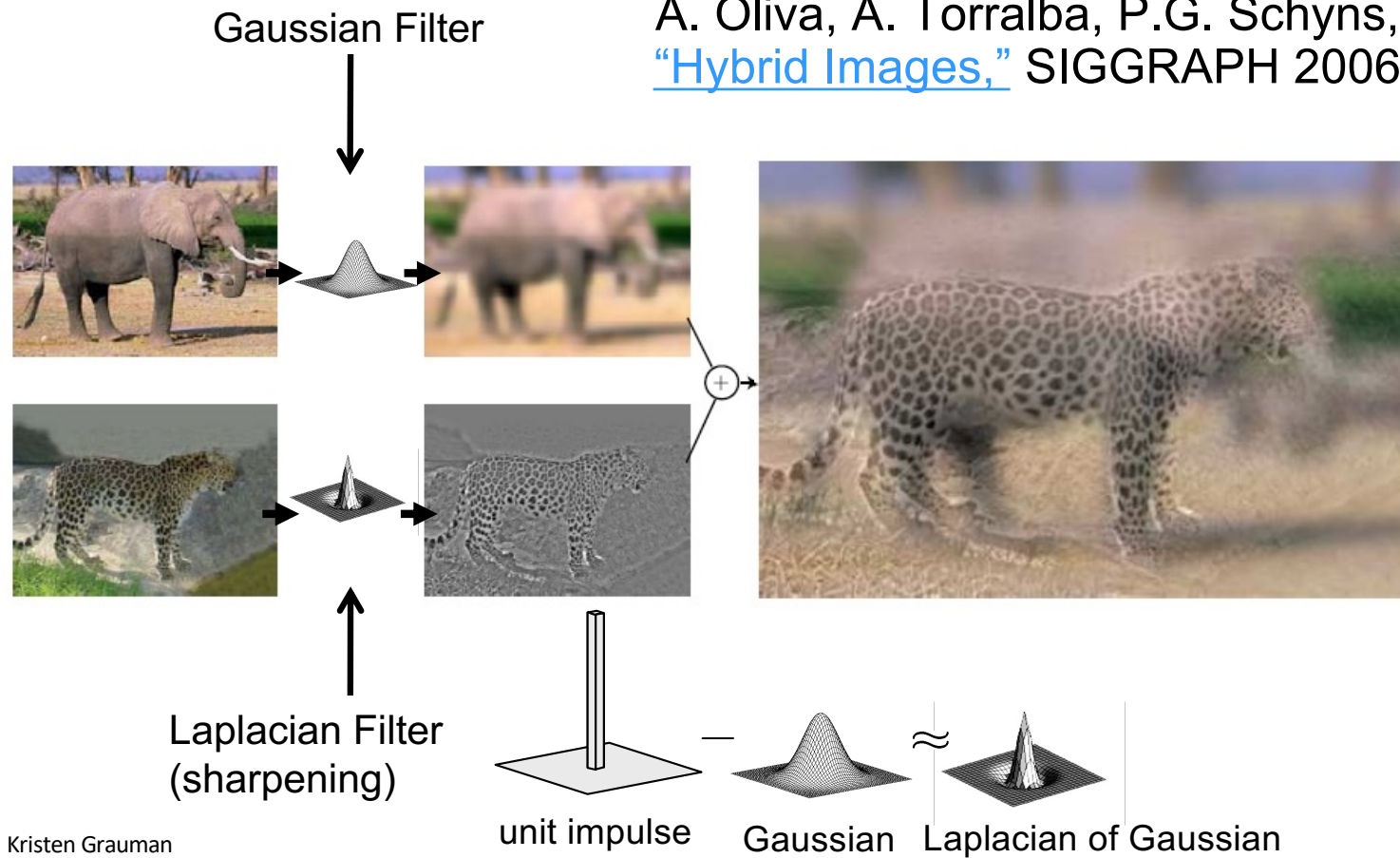
Asymptotic cost for 2D vs 1D filtering? Let image be $P \times P$, filter be $N \times N$

Application: Hybrid Images

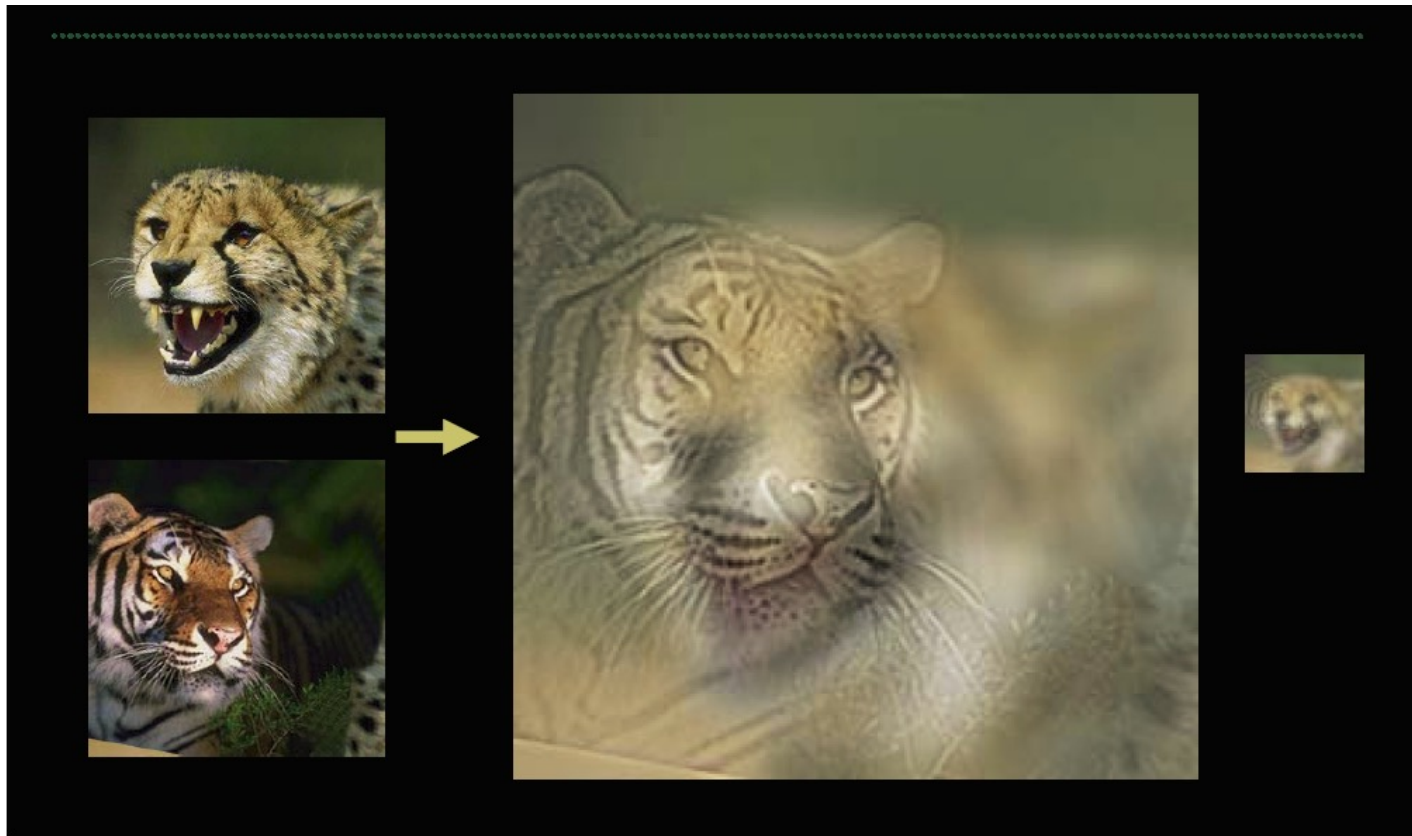


Application: Hybrid Images

A. Oliva, A. Torralba, P.G. Schyns,
["Hybrid Images,"](#) SIGGRAPH 2006



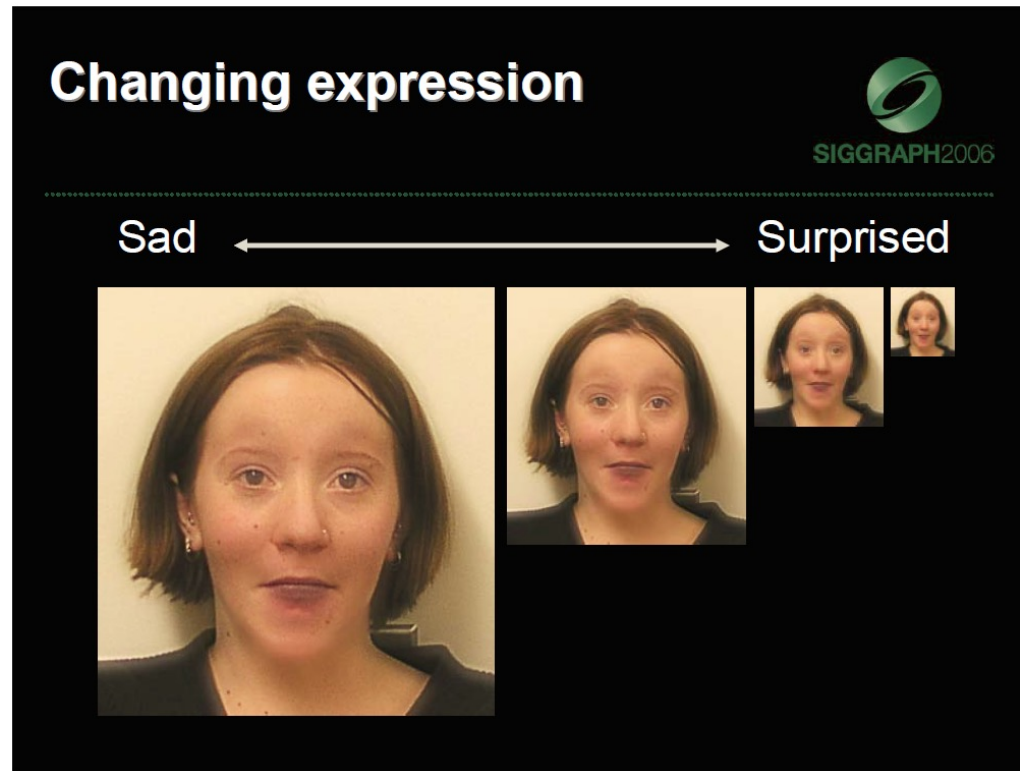
Application: Hybrid Images



Kristen Grauman

Aude Oliva & Antonio Torralba & Philippe G Schyns, SIGGRAPH 2006

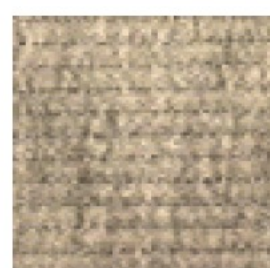
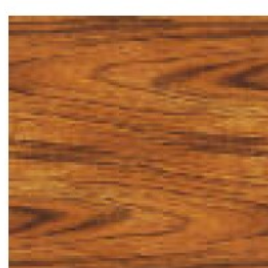
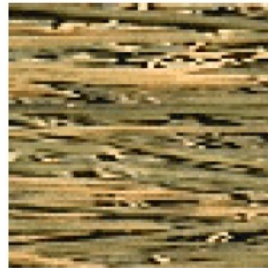
Application: Hybrid Images



Plan for next two lectures

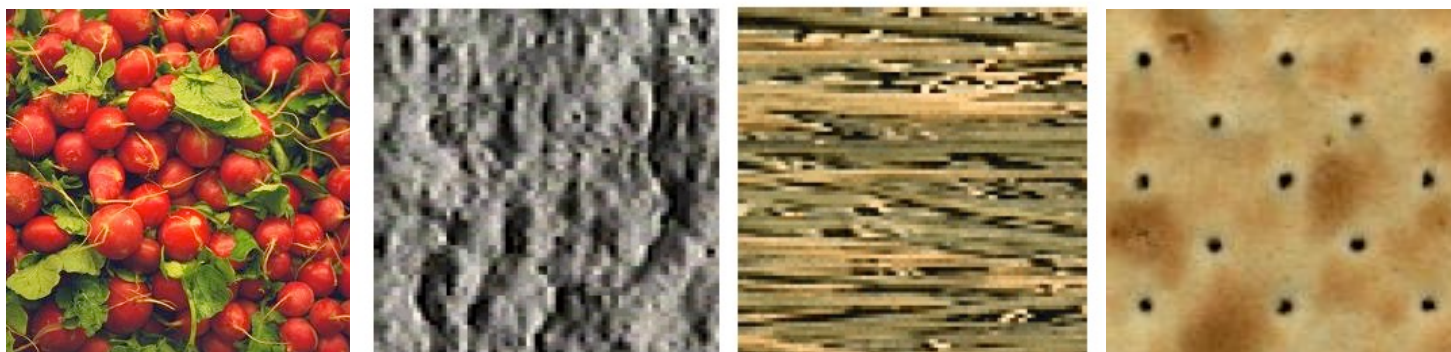
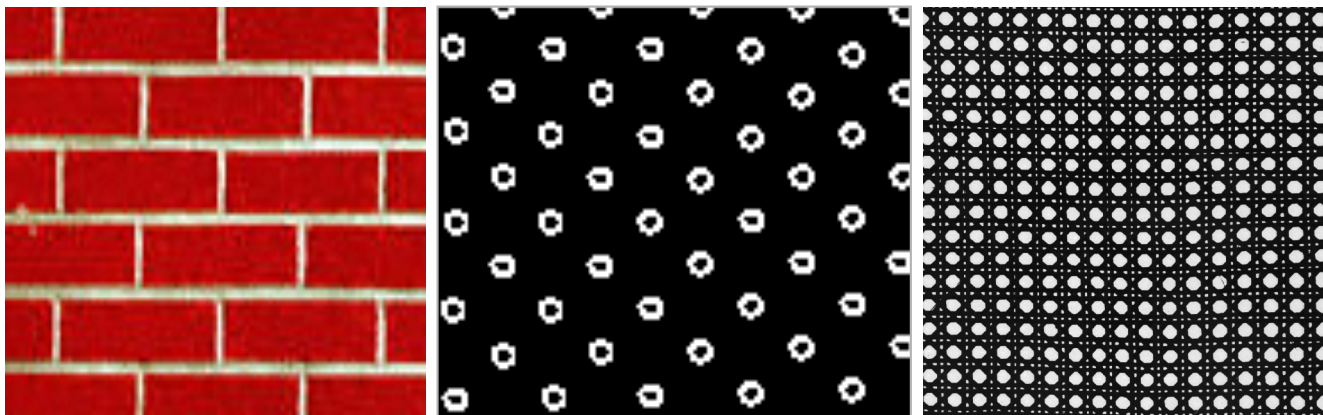
- Filters: math and properties
- Types of filters
 - Linear
 - Smoothing
 - Other
 - Non-linear
 - Median
- Applications
 - Texture representation with filters
 - Anti-aliasing for image subsampling

Texture



Due to:
Patterns, marks, etches, blobs, holes, relief, etc.

Regular (top), random (bottom) patterns



Why analyze texture?

- Important for how we perceive objects
- Can be an important appearance cue that allows us to distinguish objects, especially if shape is similar across objects

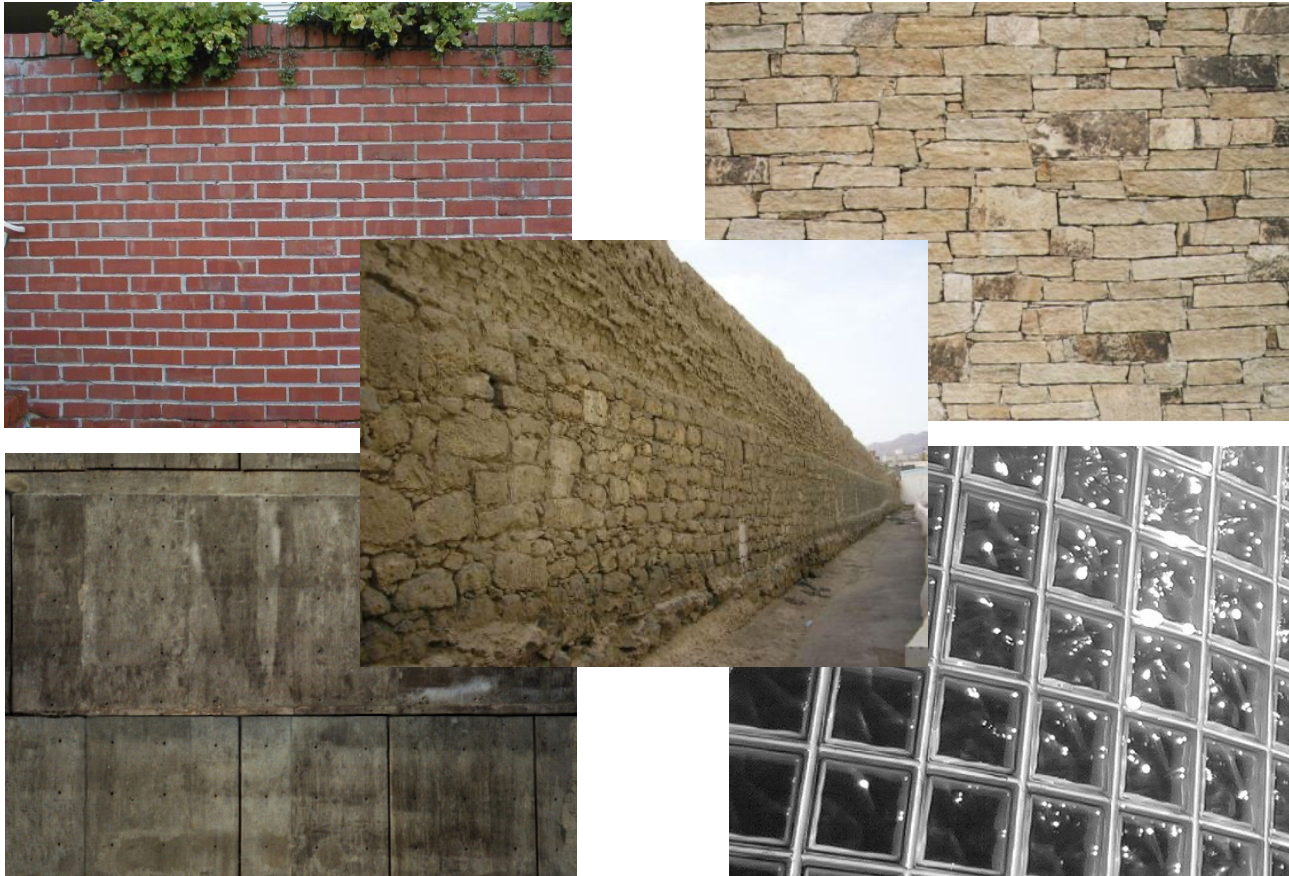
Same shape, different texture/object



Kristen Grauman

<http://animals.nationalgeographic.com/>

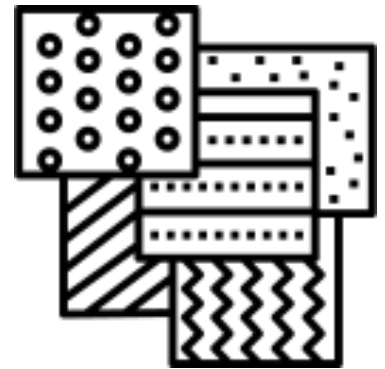
Same object, different texture



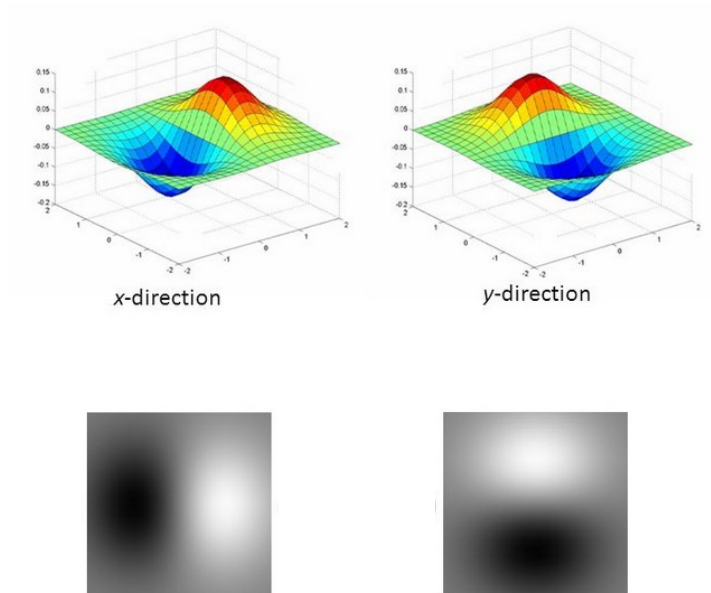
Kristen Grauman

Texture Representation

- Textures are made up of repeated local patterns, so:
 - Find the patterns
 - Use filters that look like patterns (spots, bars, raw patches...)
 - Consider magnitude of response
 - Describe their statistics within each local window
 - E.g. mean, standard deviation, histogram



Derivative of Gaussian filter

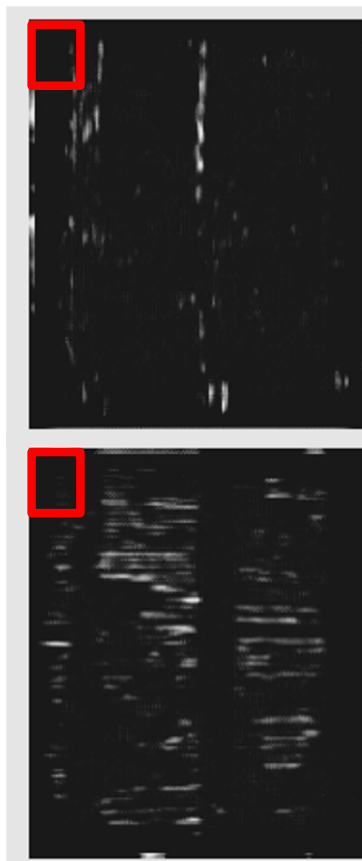
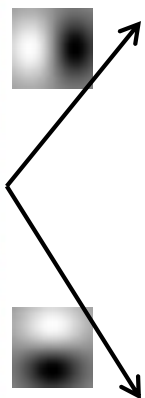


Figures from Noah Snavely

Texture representation: Example



original image



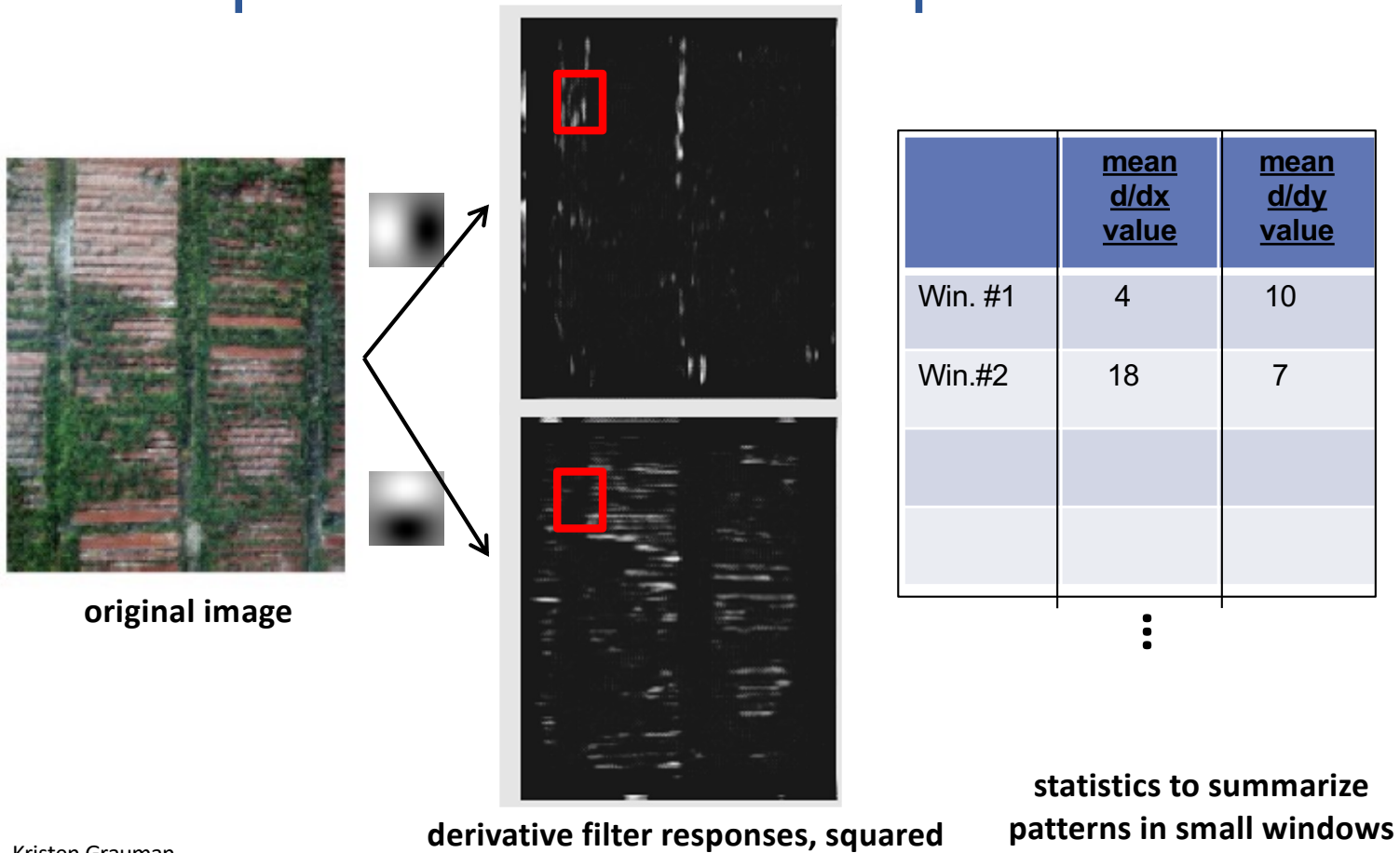
derivative filter responses, squared

	<u>mean d/dx value</u>	<u>mean d/dy value</u>
Win. #1	4	10

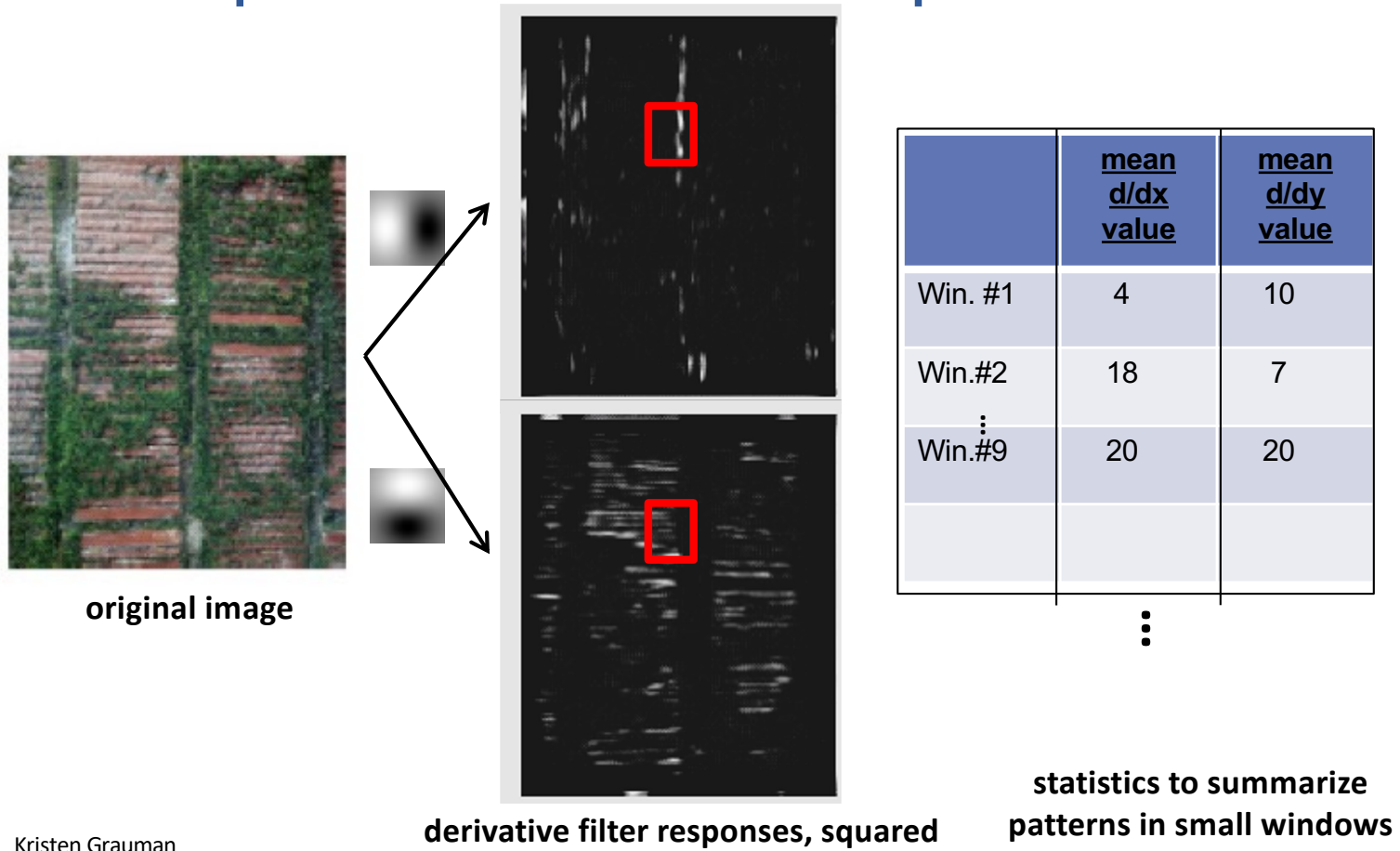
⋮

statistics to summarize
patterns in small windows

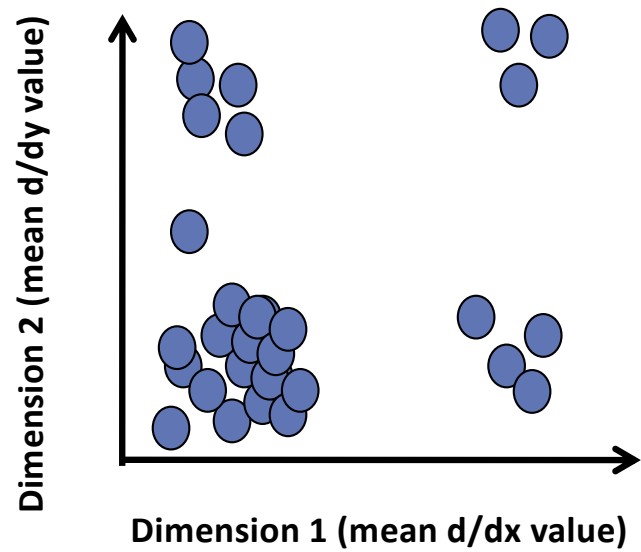
Texture representation: Example



Texture representation: Example



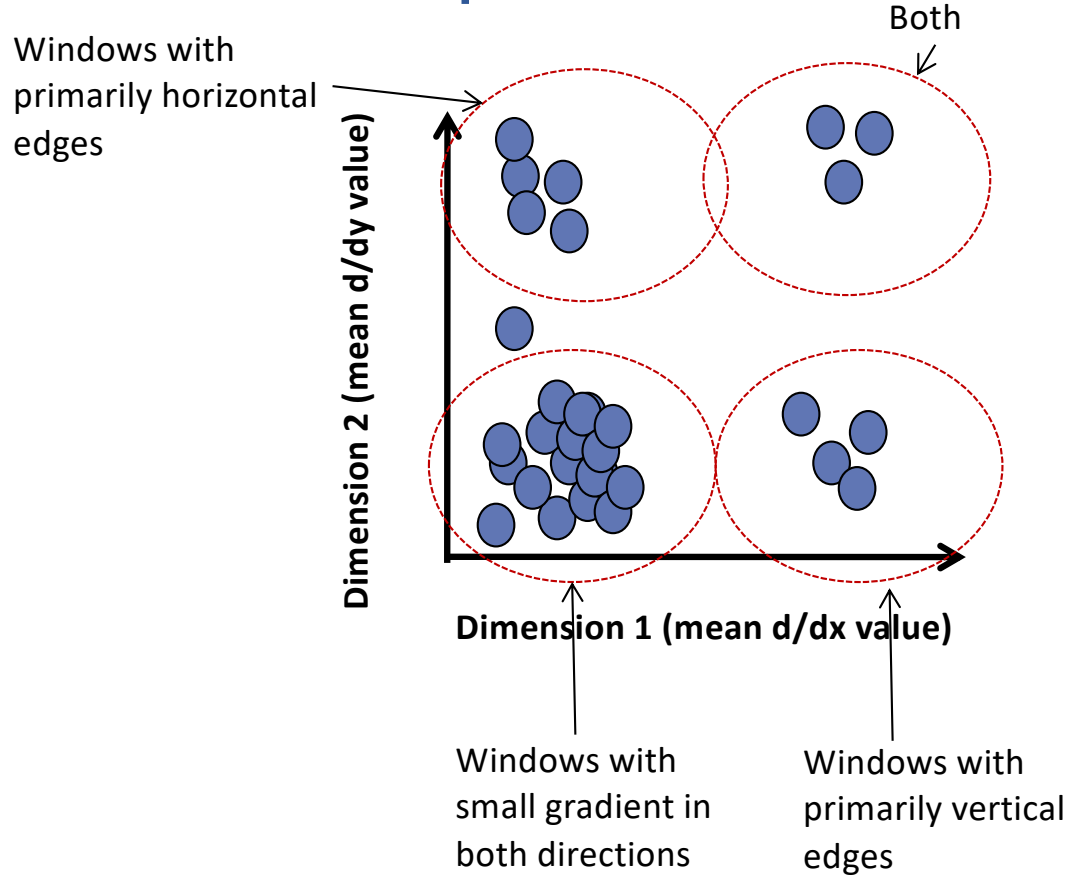
Texture representation: Example



	<u>mean d/dx value</u>	<u>mean d/dy value</u>
Win. #1	4	10
Win.#2	18	7
⋮		
Win.#9	20	20
	⋮	

statistics to summarize
patterns in small
windows

Texture representation: Example



	<u>mean d/dx value</u>	<u>mean d/dy value</u>
Win. #1	4	10
Win.#2	18	7
⋮		
Win.#9	20	20

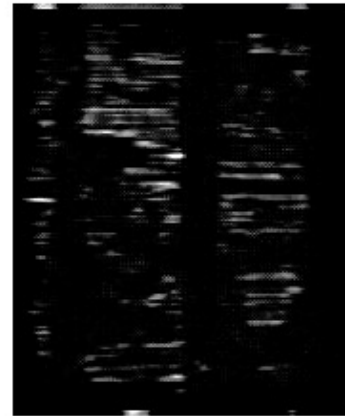
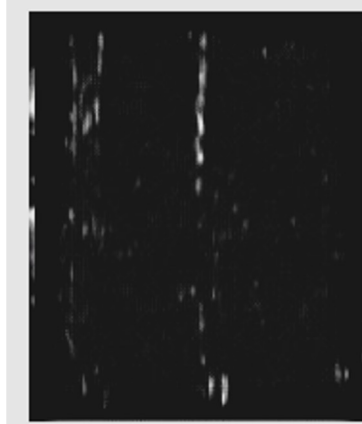
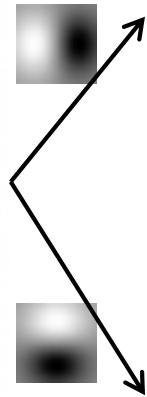
⋮

statistics to summarize patterns in small windows

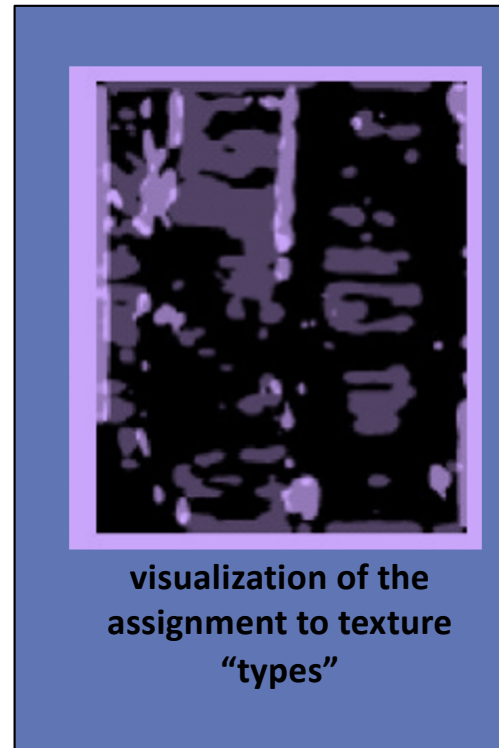
Texture representation: Example



original image

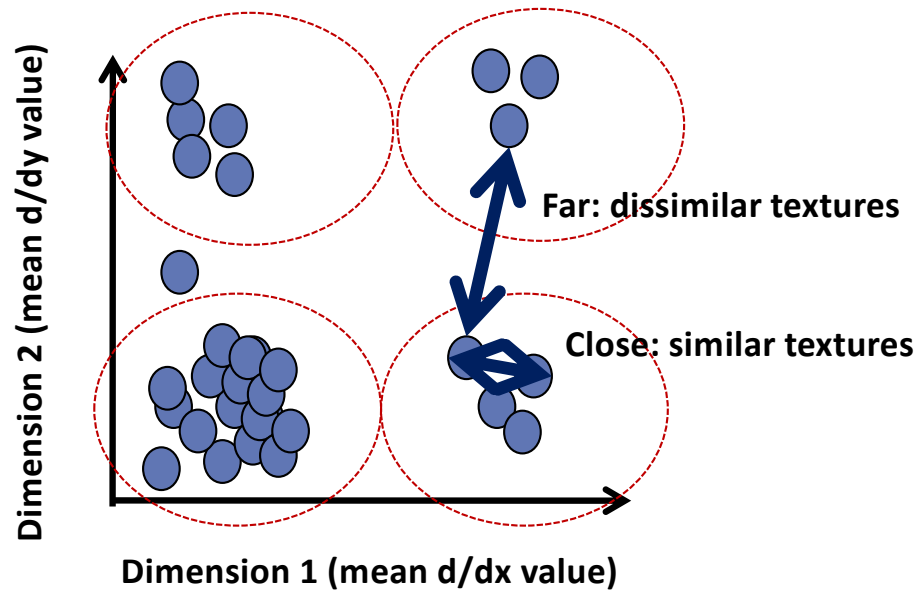


derivative filter responses, squared



visualization of the
assignment to texture
"types"

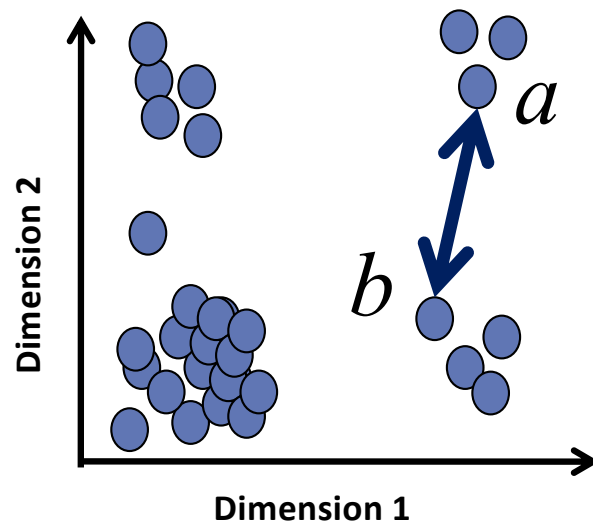
Texture representation: Example



	<u>mean d/dx value</u>	<u>mean d/dy value</u>
Win. #1	4	10
Win. #2	18	7
⋮		
Win. #9	20	20
	⋮	

**statistics to summarize
patterns in small
windows**

Computing Distance using Texture

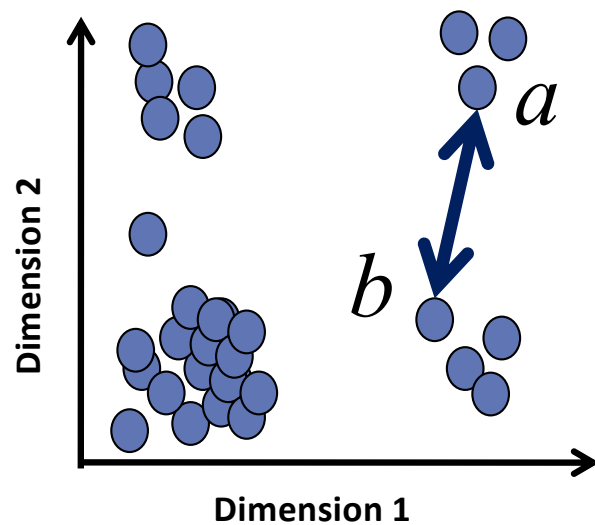


$$D(a, b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}$$

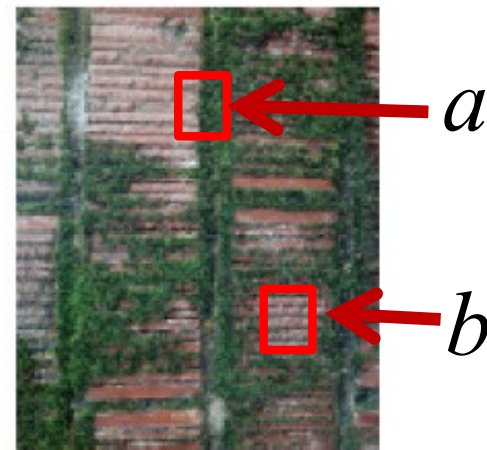
$$= \sqrt{\sum_{i=1}^d (a_i - b_i)^2}$$

Euclidean distance (L_2)

Texture Representation: Example



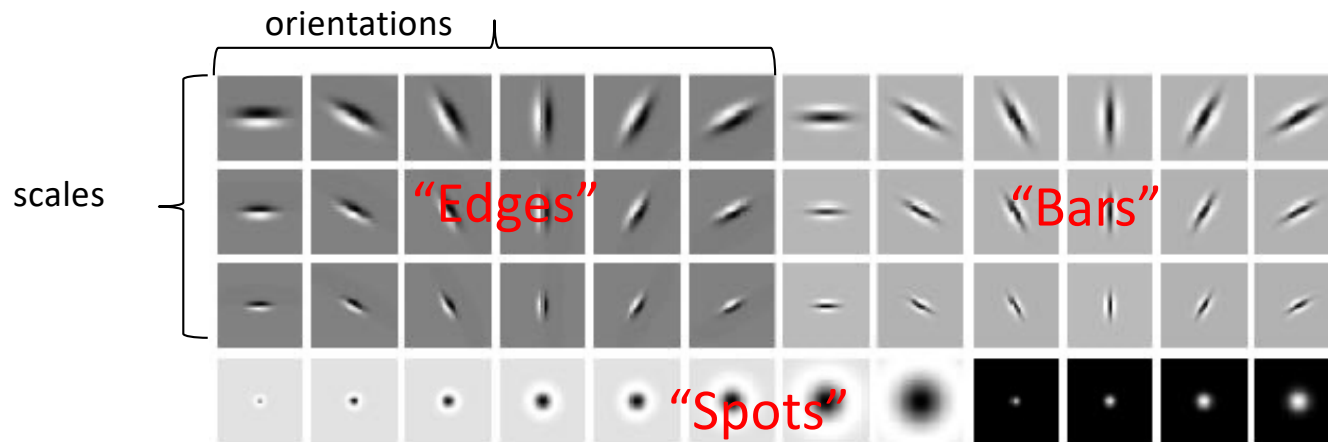
Distance reveals how dissimilar texture from window a is from texture in window b.



Filter banks

- Our previous example used two filters, and resulted in a 2-dimensional feature vector to describe texture in a window.
 - x and y derivatives revealed something about local structure.
- We can generalize to apply a collection of multiple (d) filters: a “filter bank”.
- Then our feature vectors will be d -dimensional.

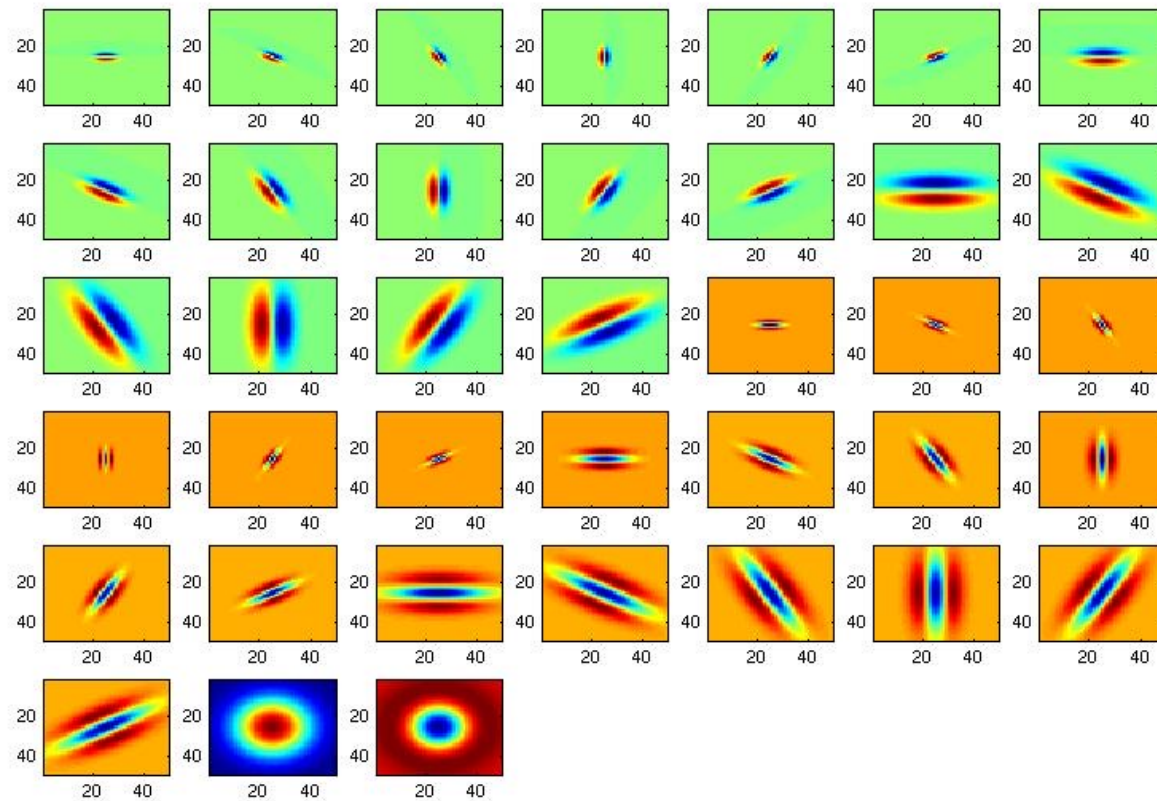
Filter banks



- What filters to put in the bank?
 - Typically we want a combination of scales and orientations, different types of patterns.

Which filters would you use to distinguish buildings from animals? Cheetahs from tigers? Ladybugs from dalmatians?

Filter bank



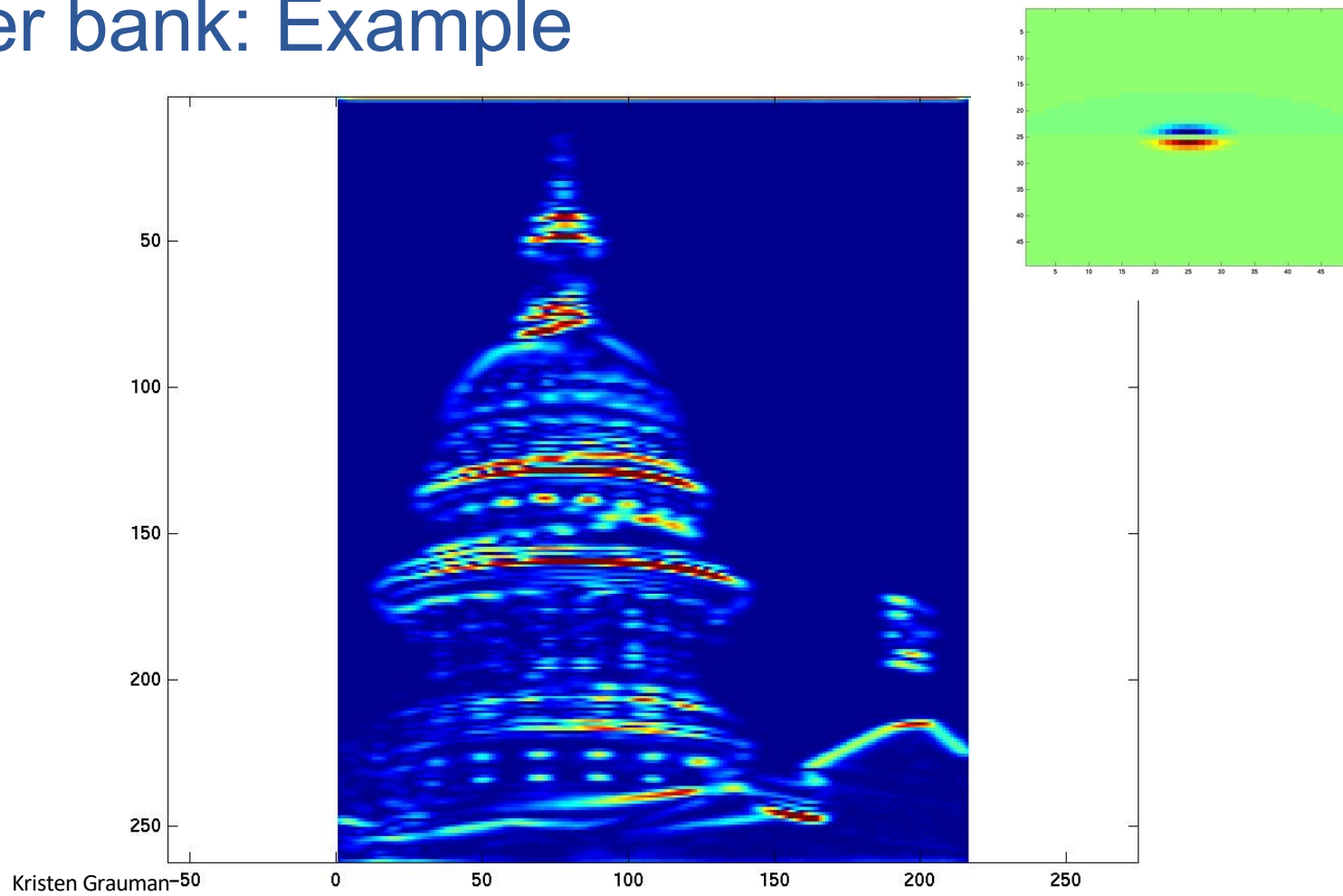
Filter bank: Example

Image from <http://www.texasexplorer.com/austincap2.jpg>

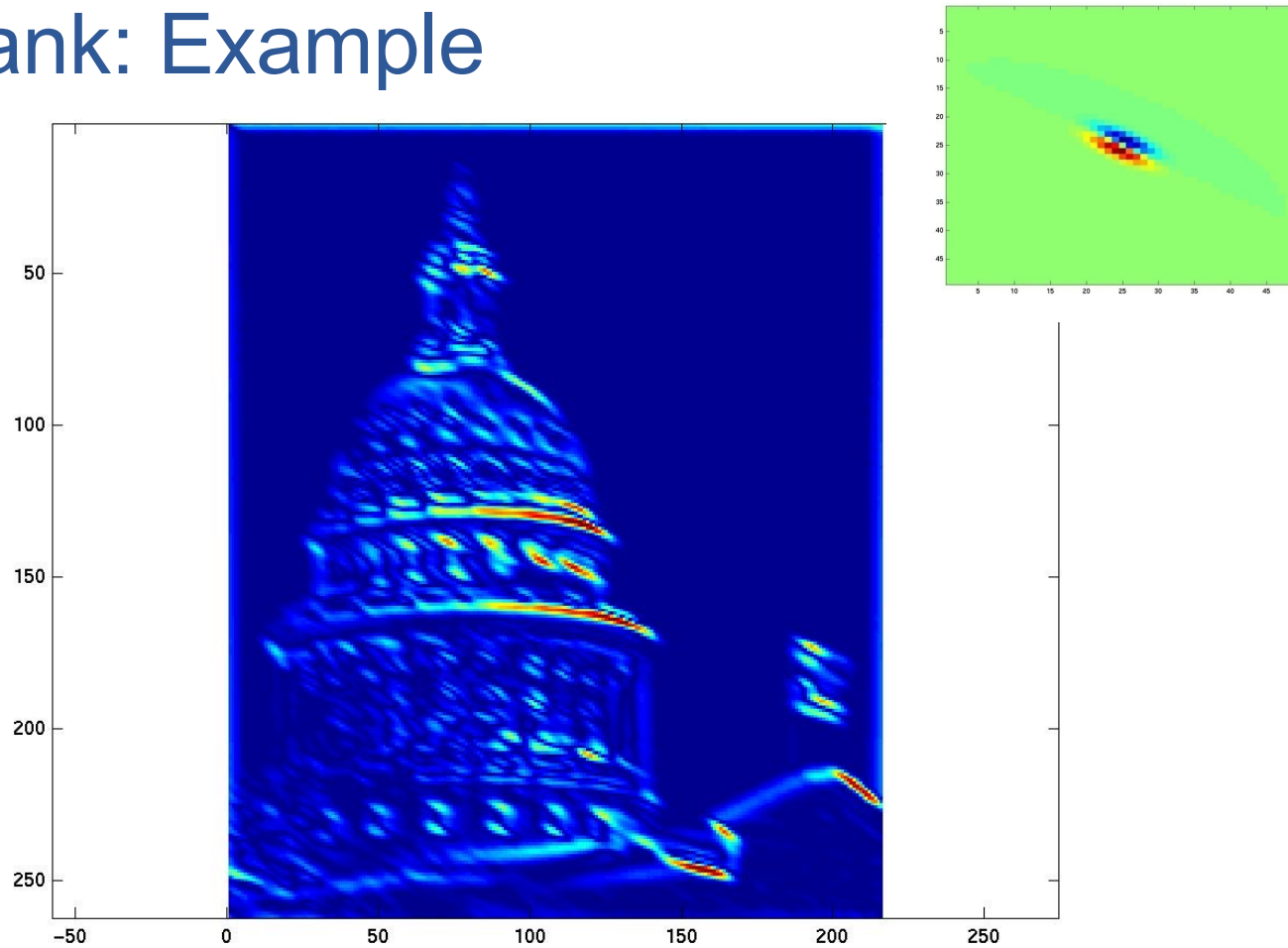


Kristen Grauman

Filter bank: Example

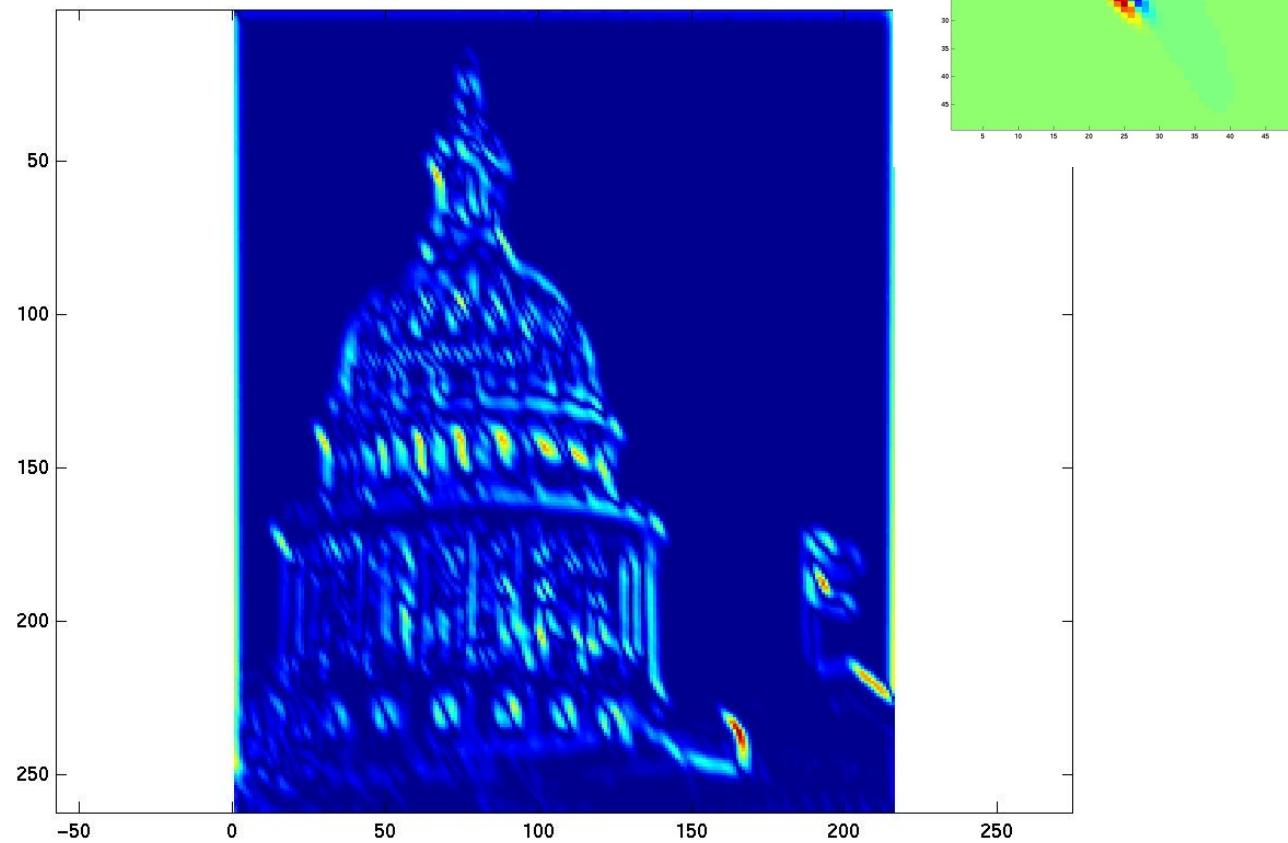


Filter bank: Example



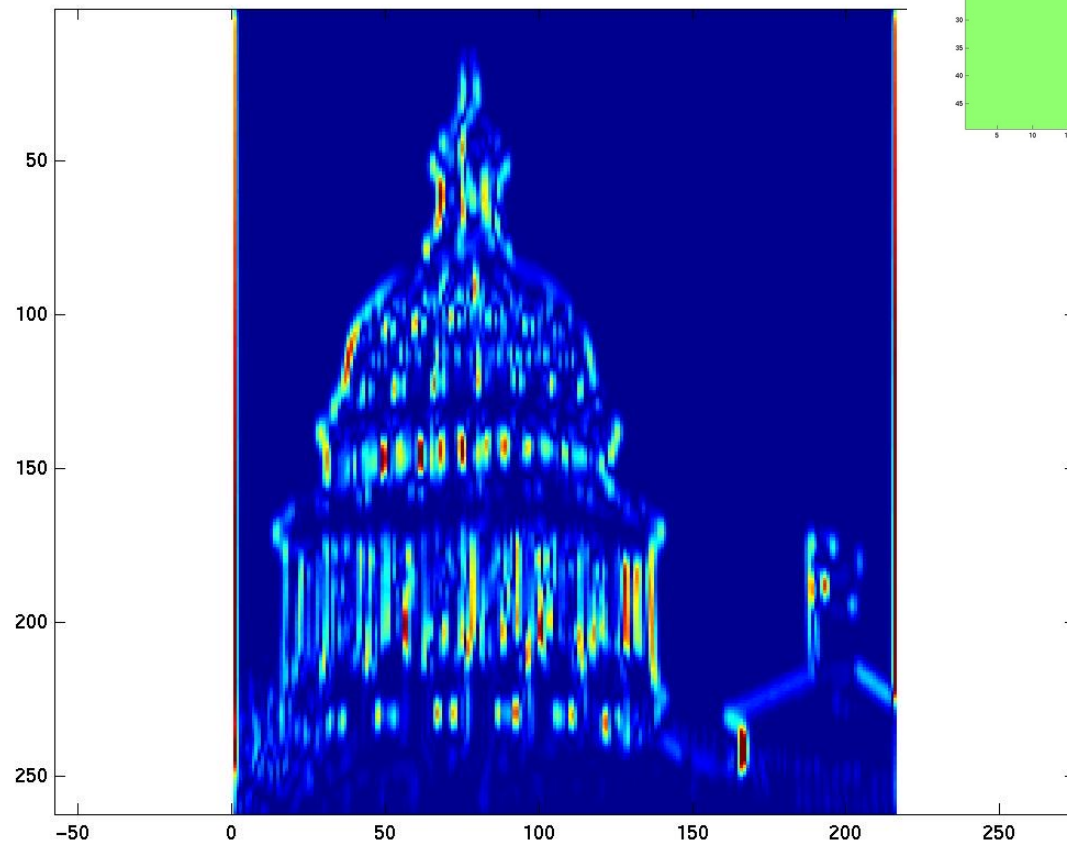
Kristen Grauman

Filter bank: Example



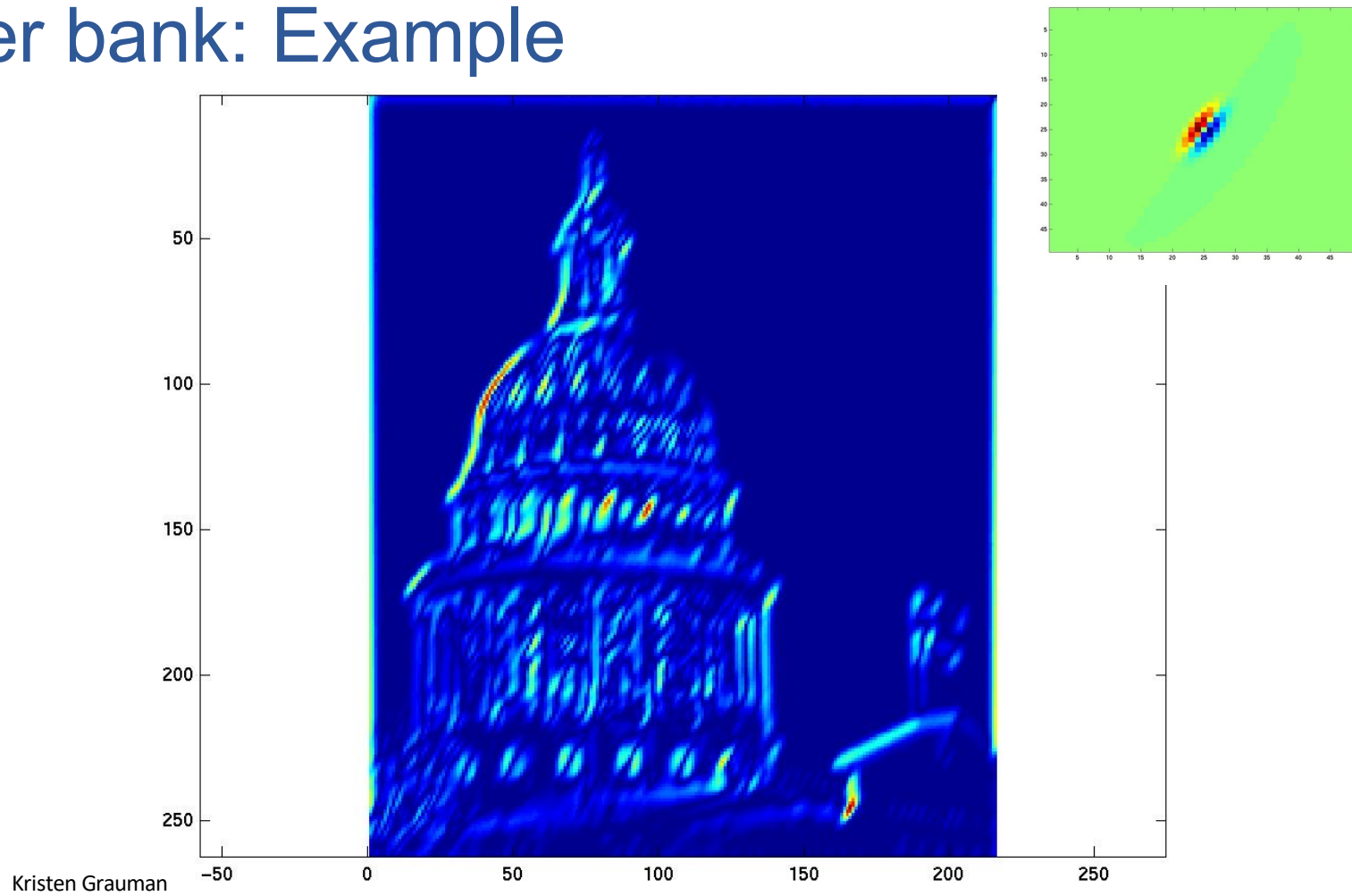
Kristen Grauman

Filter bank: Example

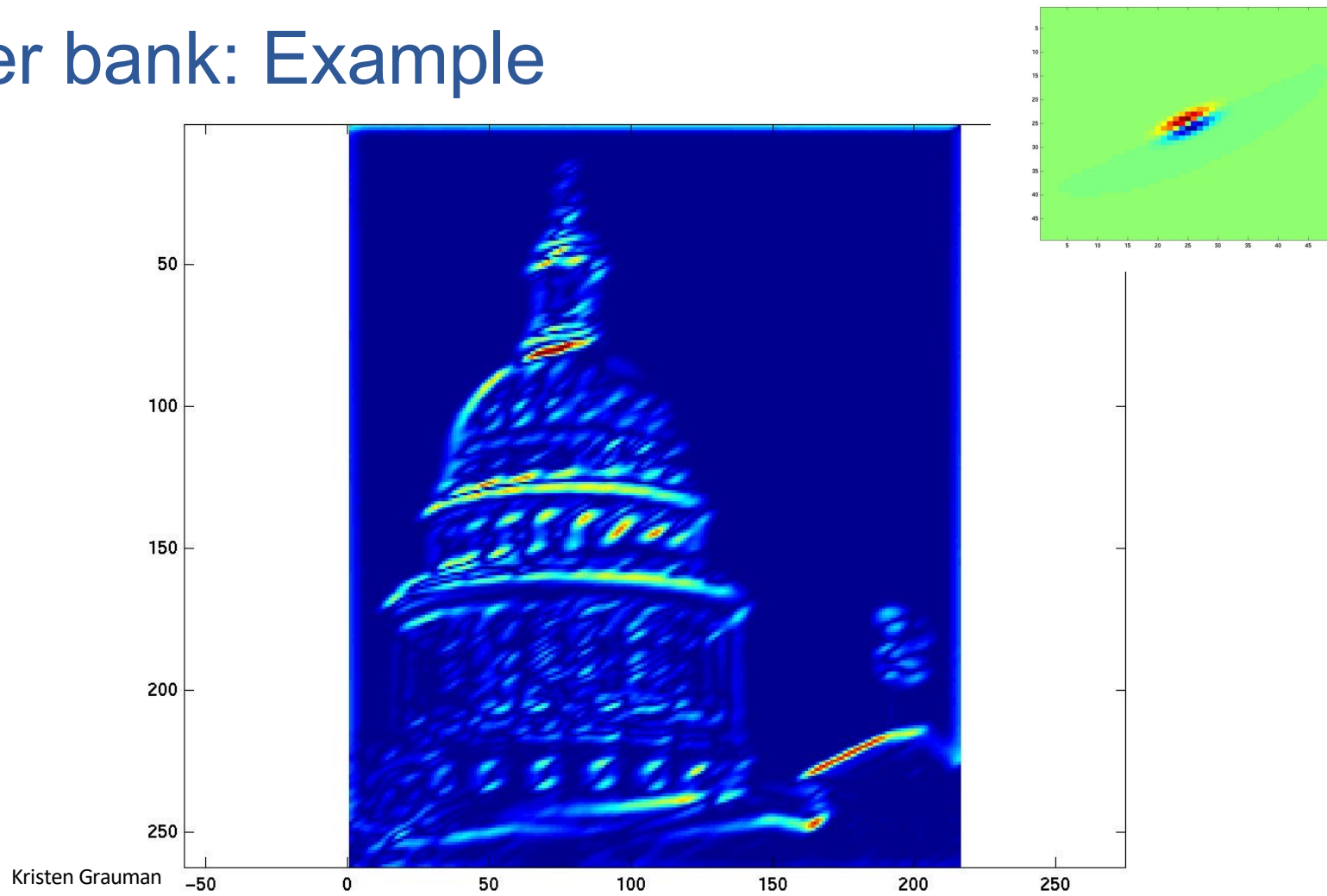


Kristen Grauman

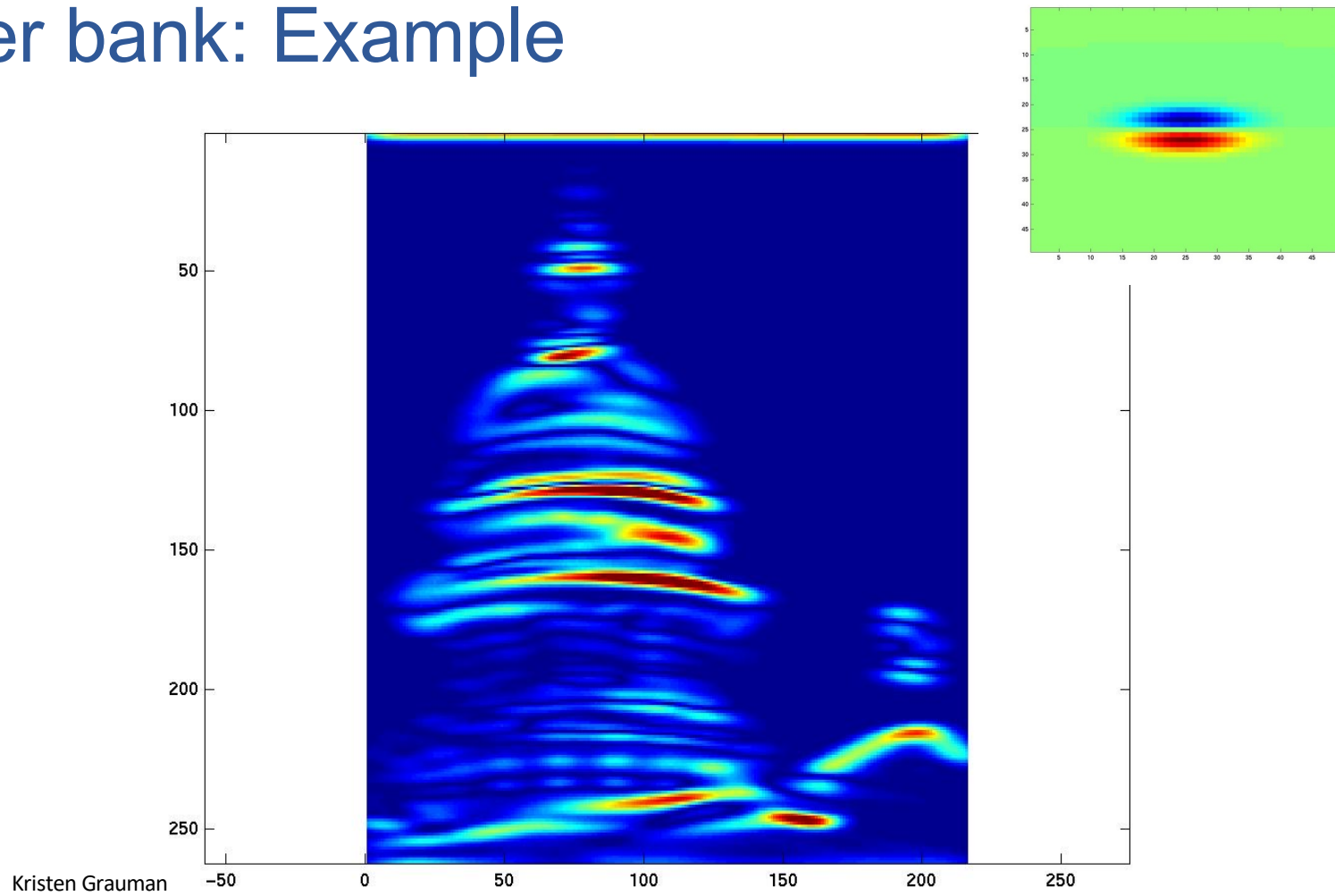
Filter bank: Example



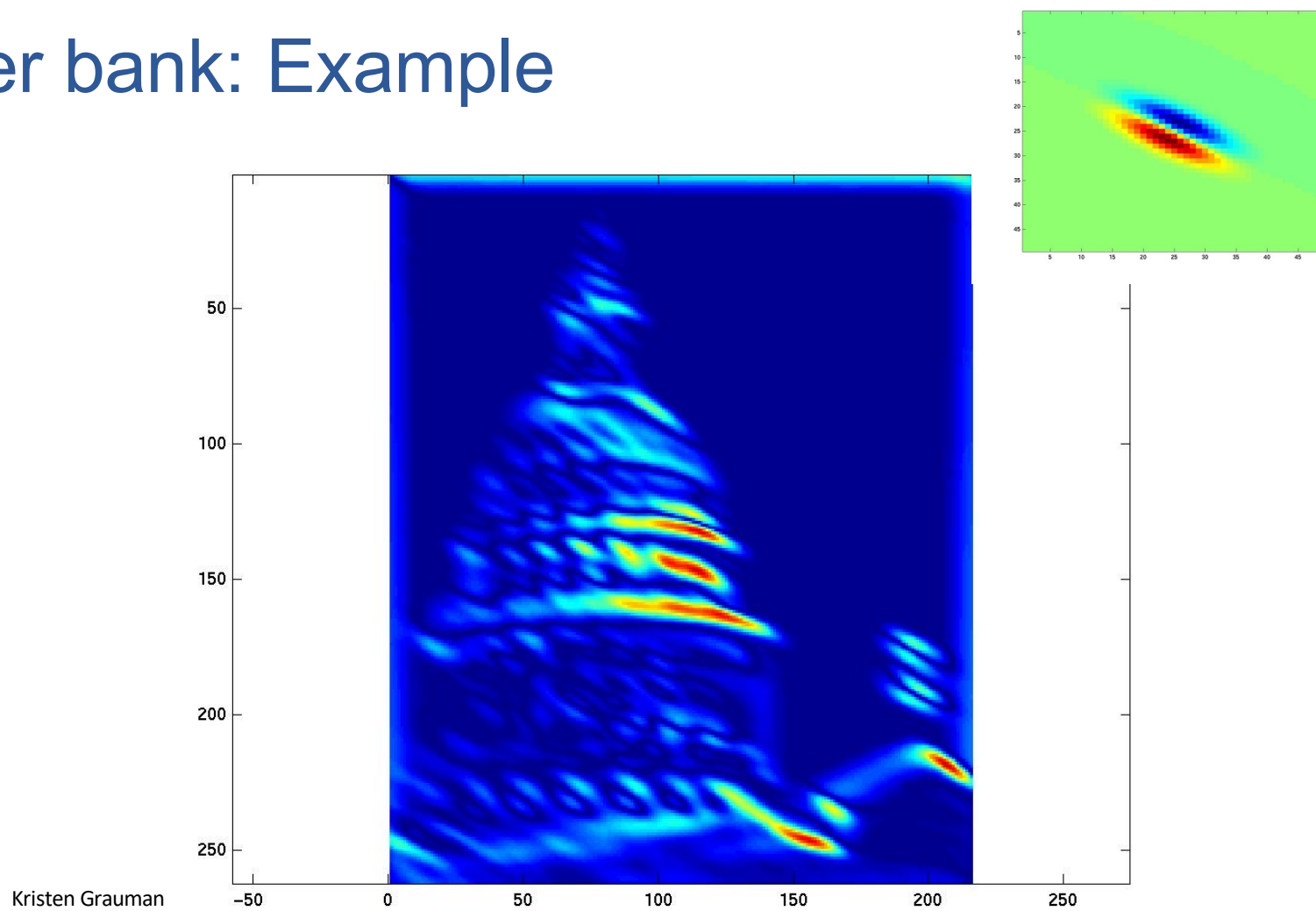
Filter bank: Example



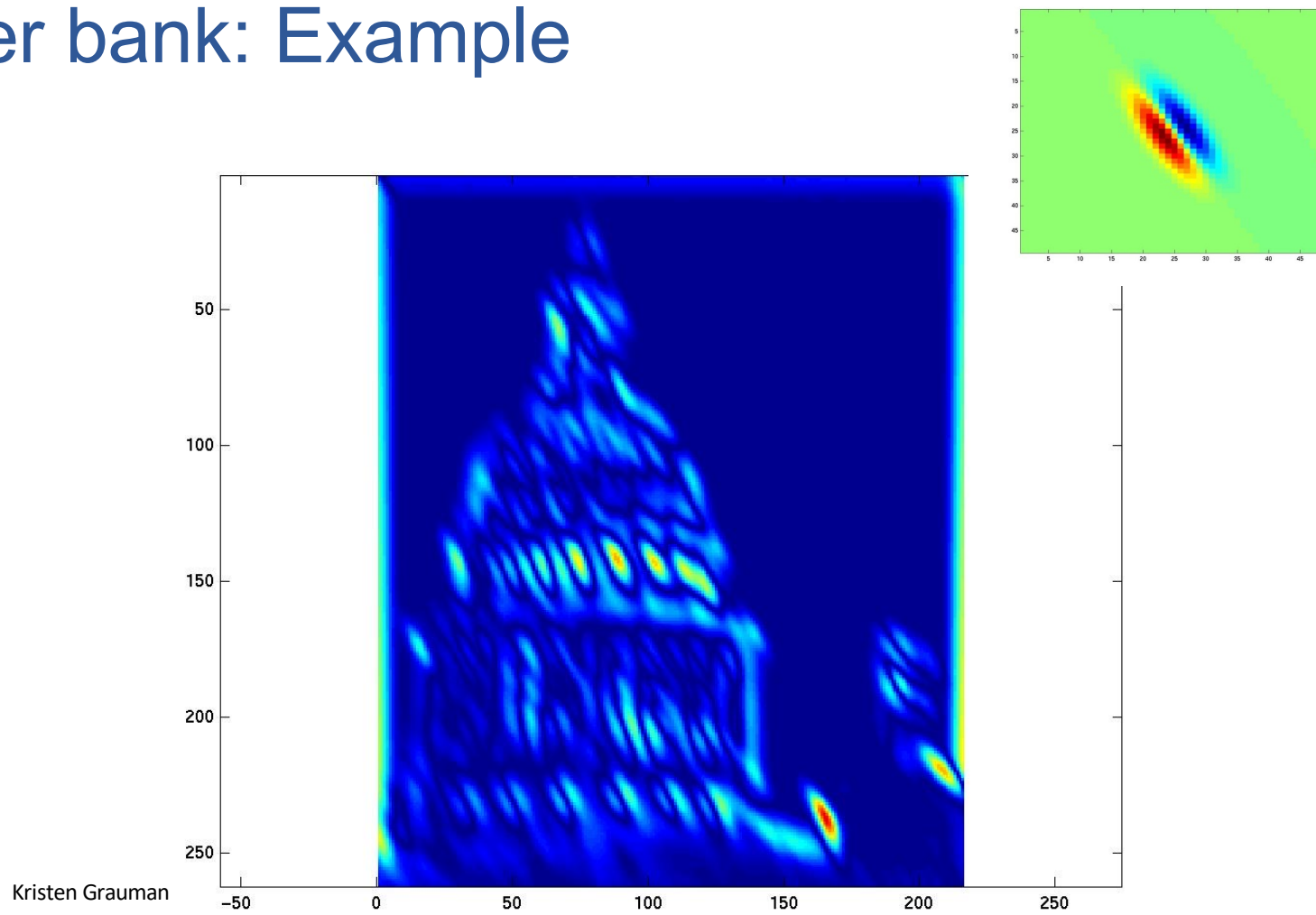
Filter bank: Example



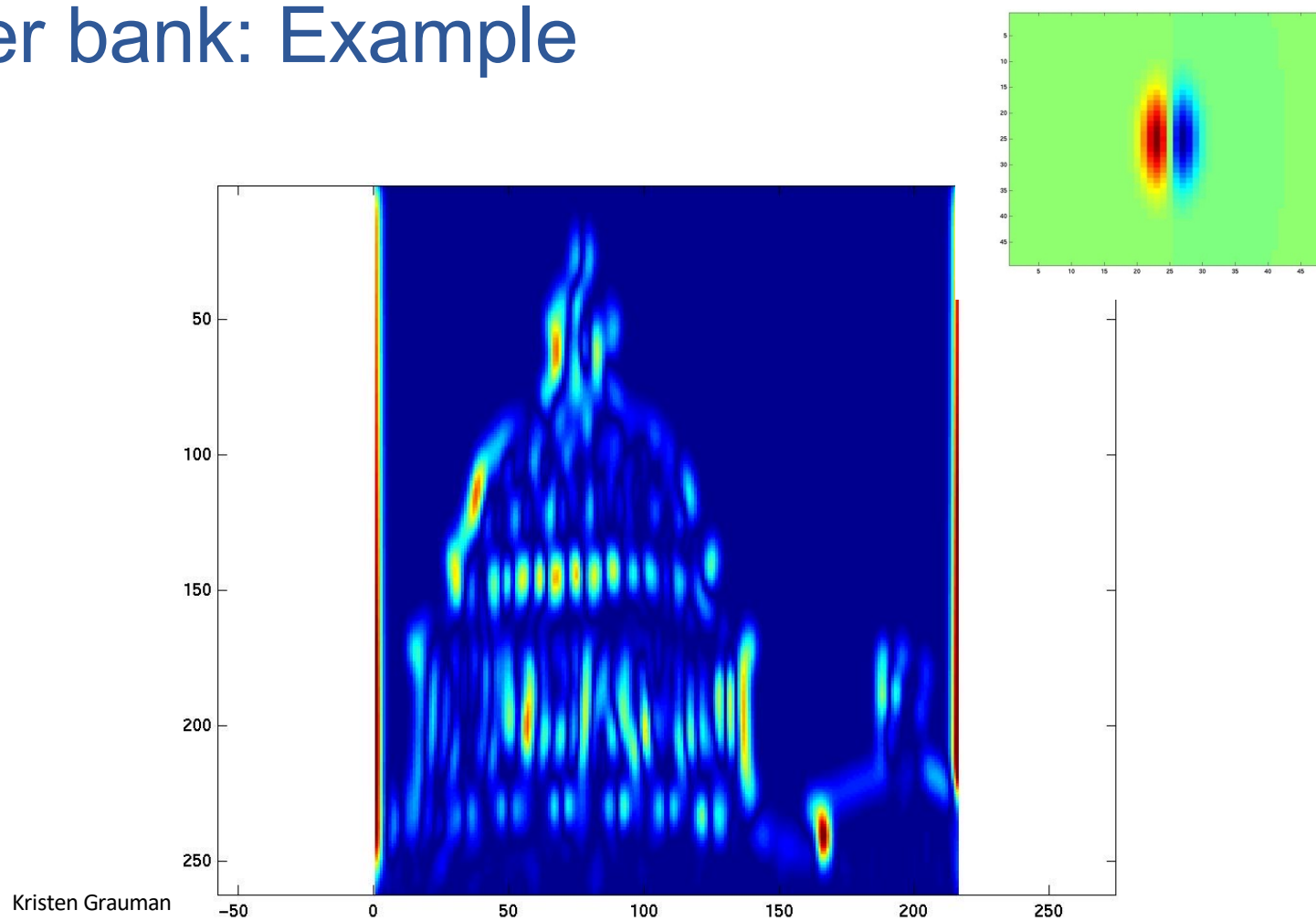
Filter bank: Example



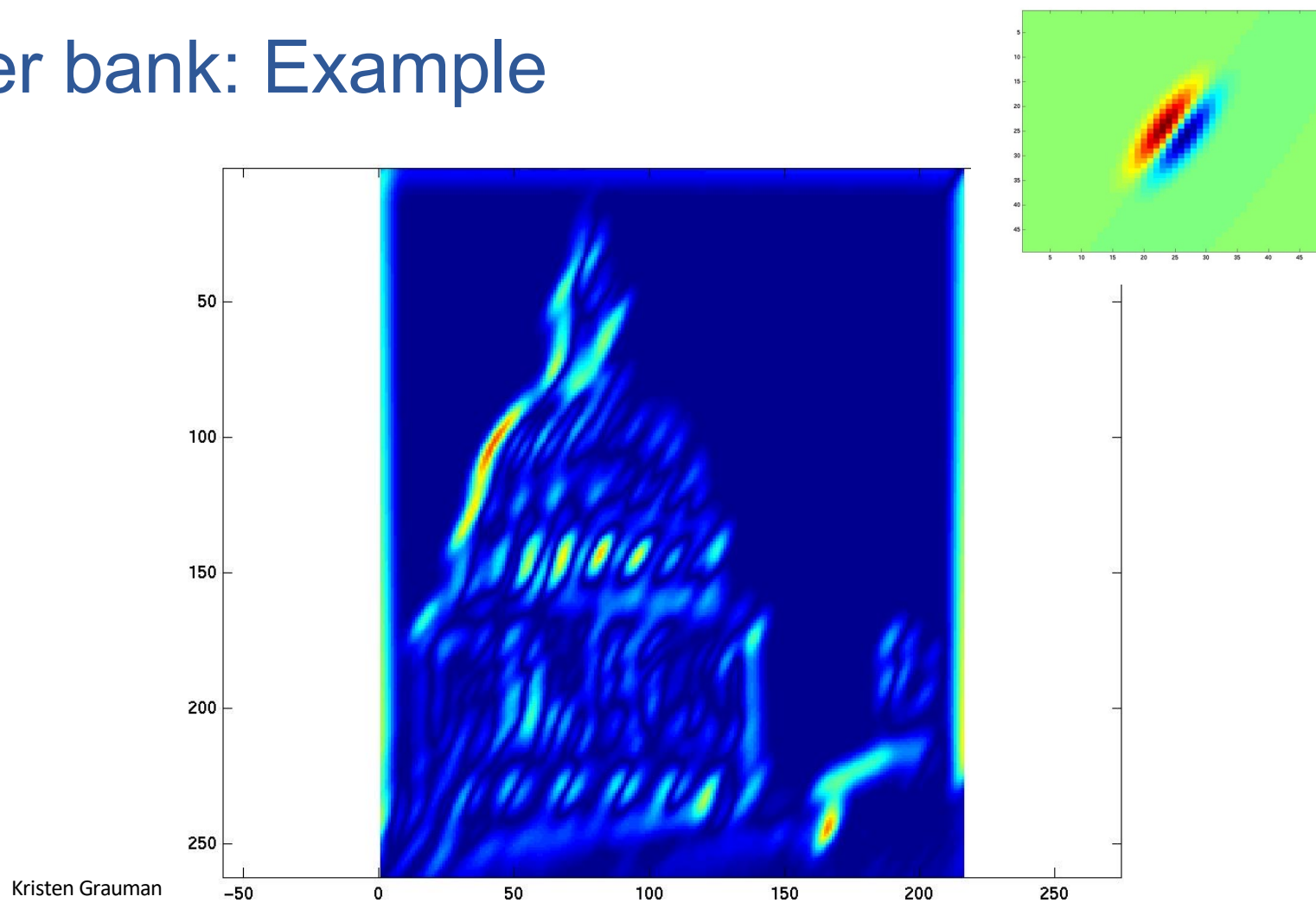
Filter bank: Example



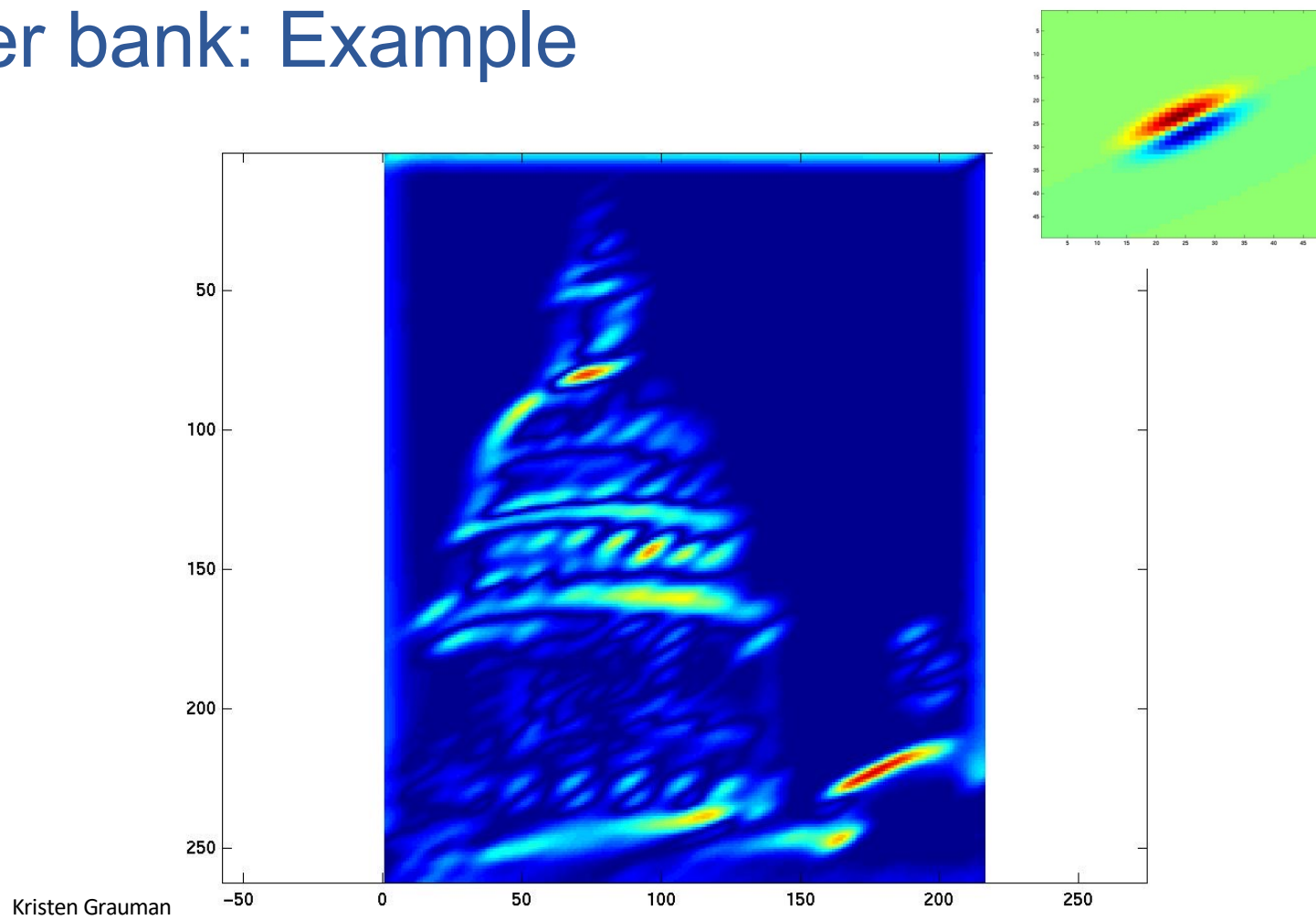
Filter bank: Example



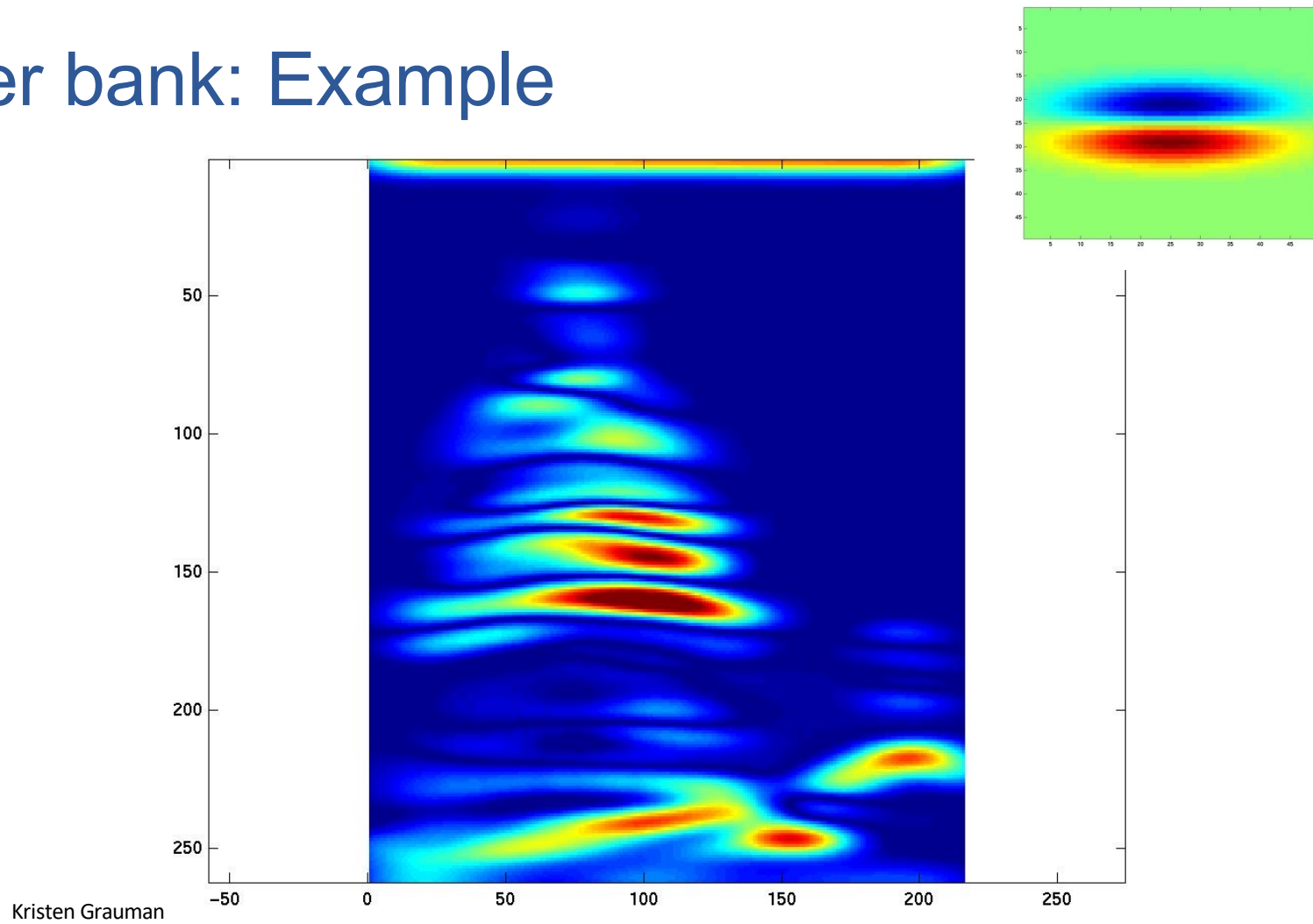
Filter bank: Example



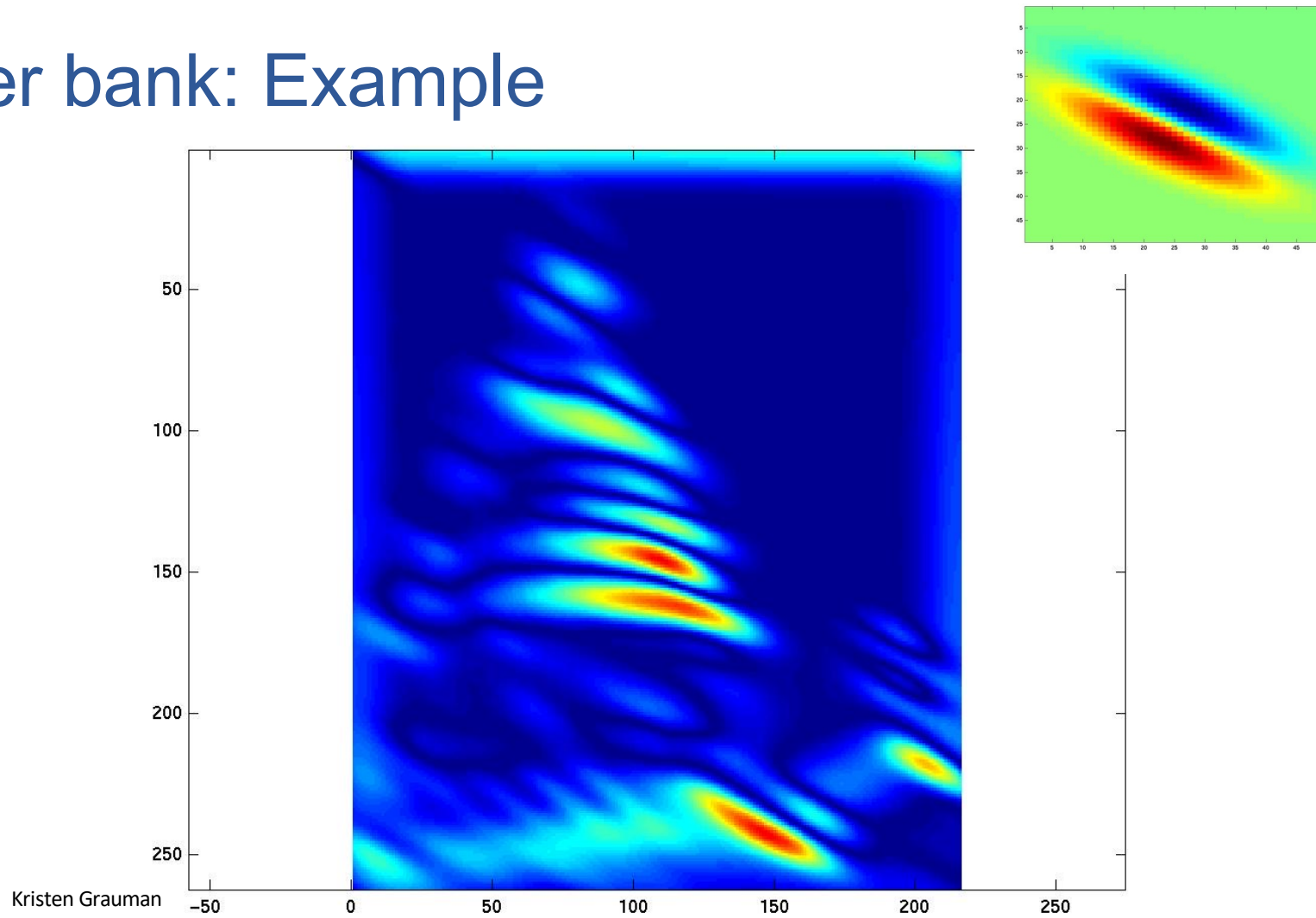
Filter bank: Example



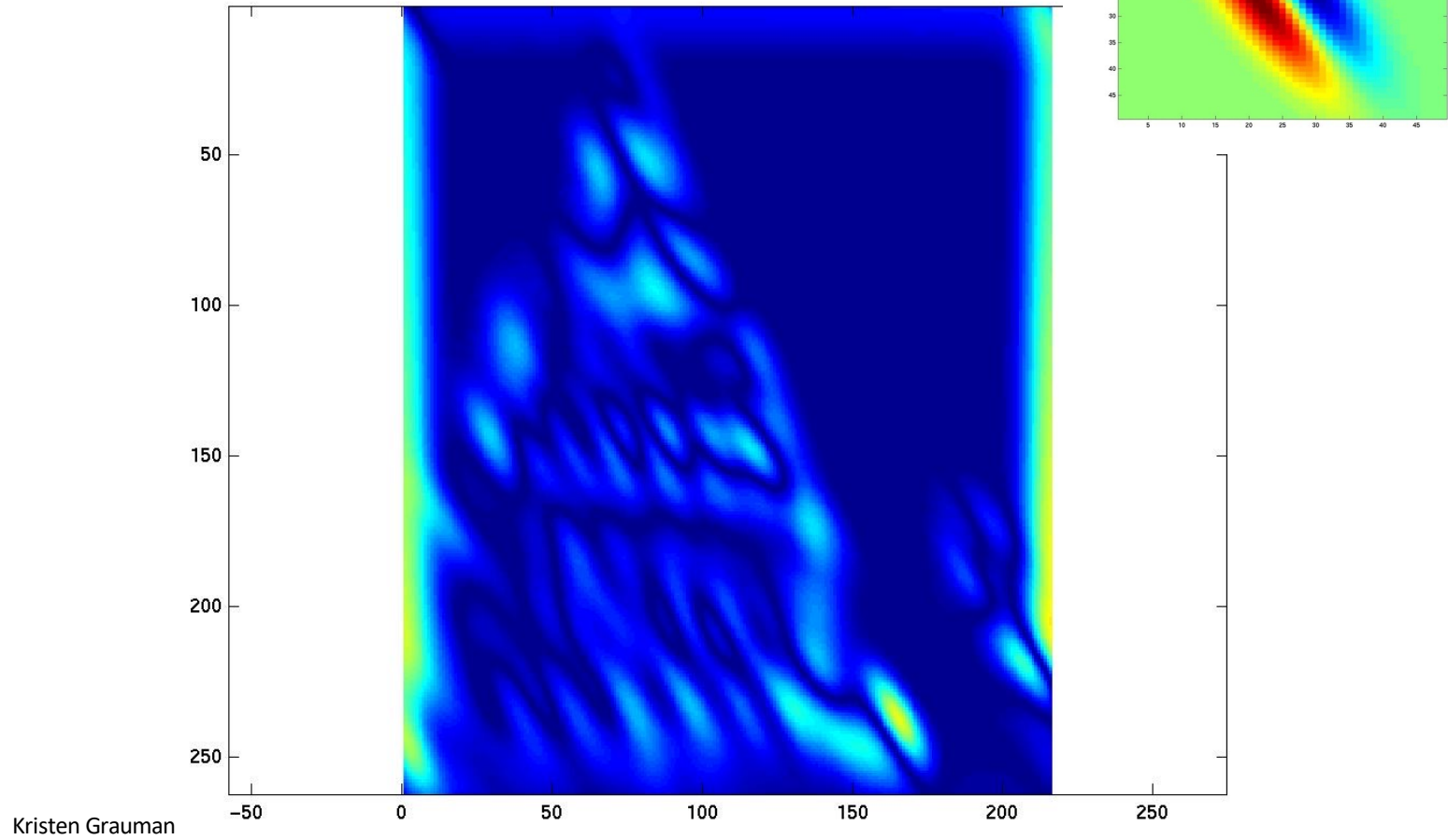
Filter bank: Example



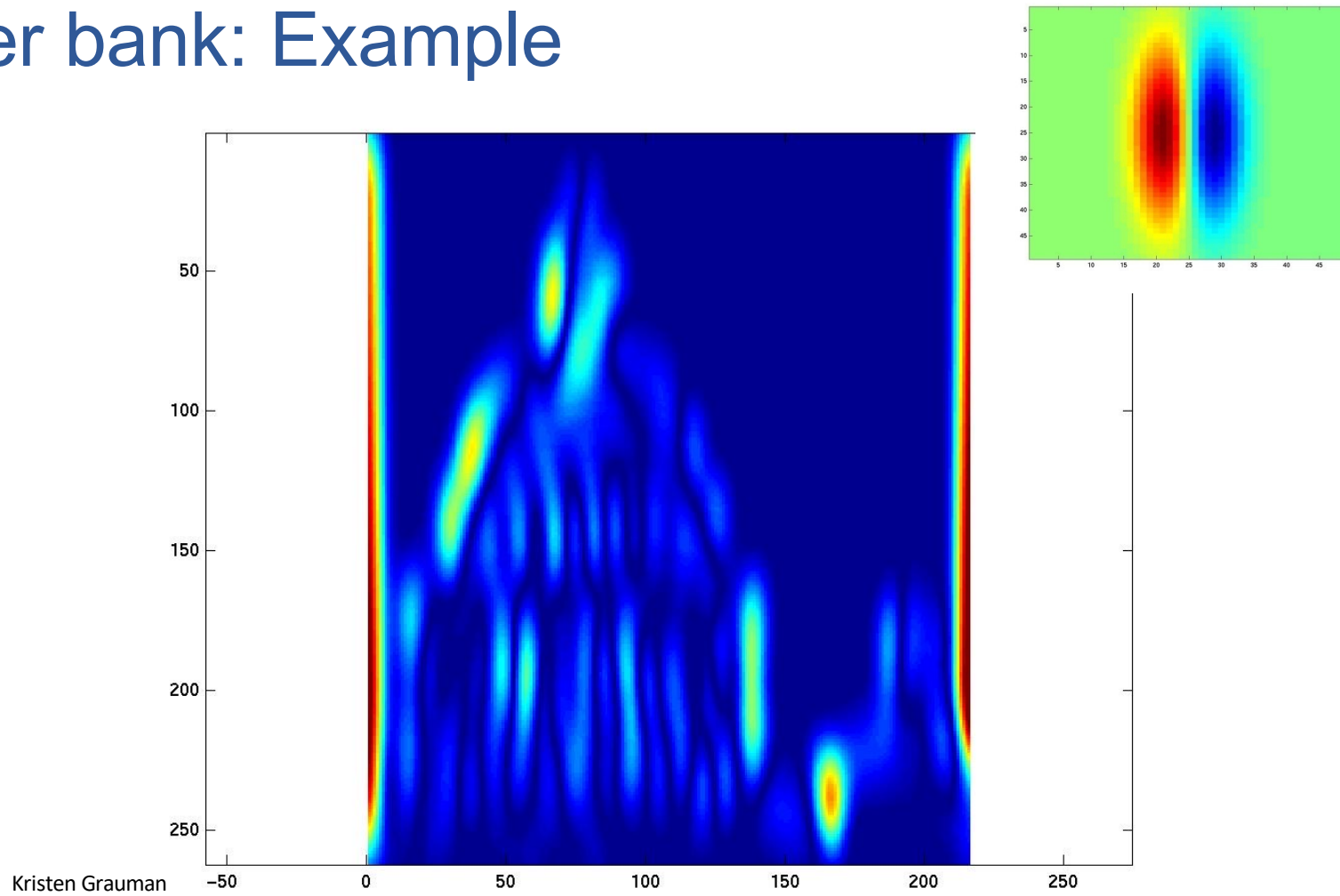
Filter bank: Example



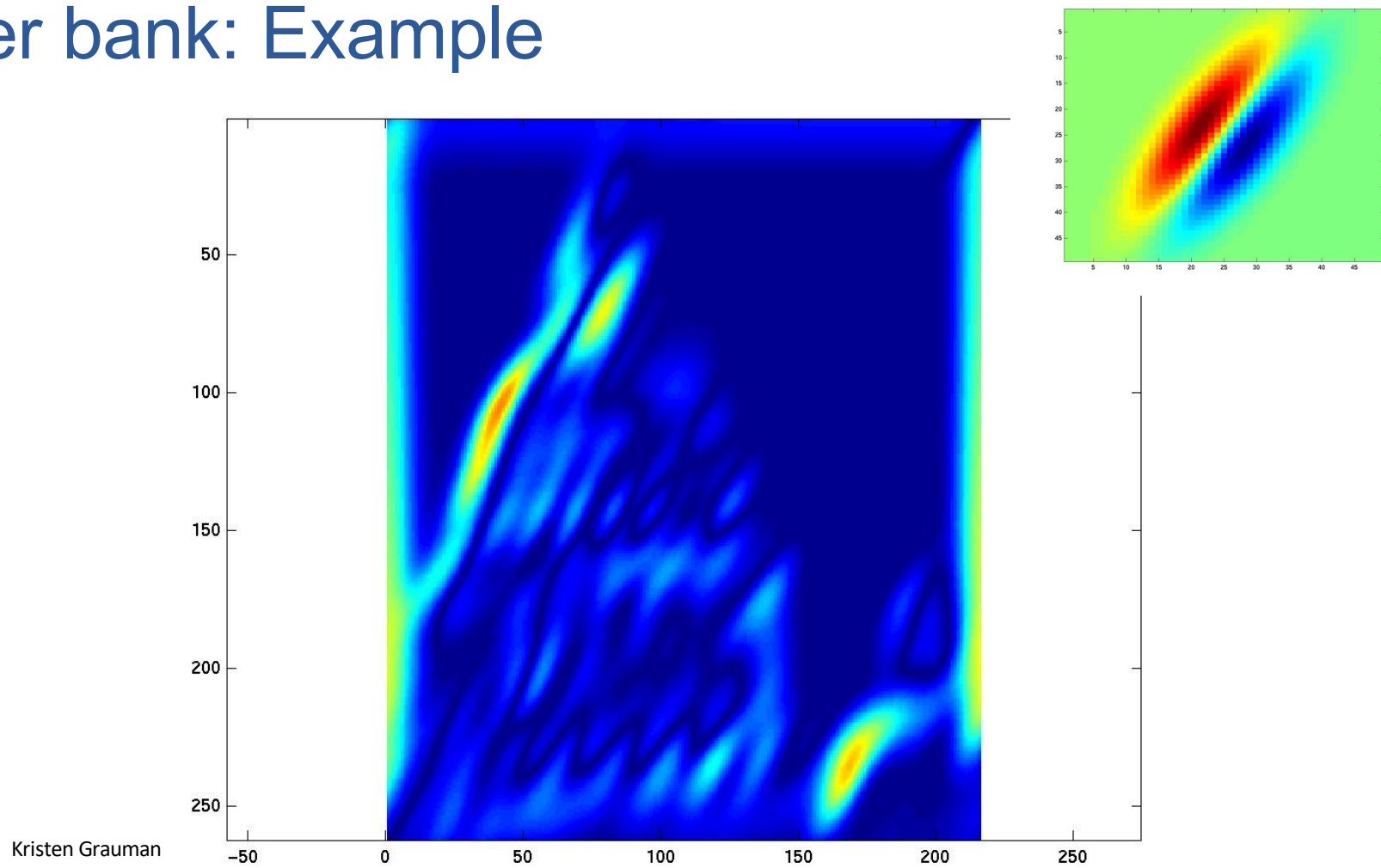
Filter bank: Example



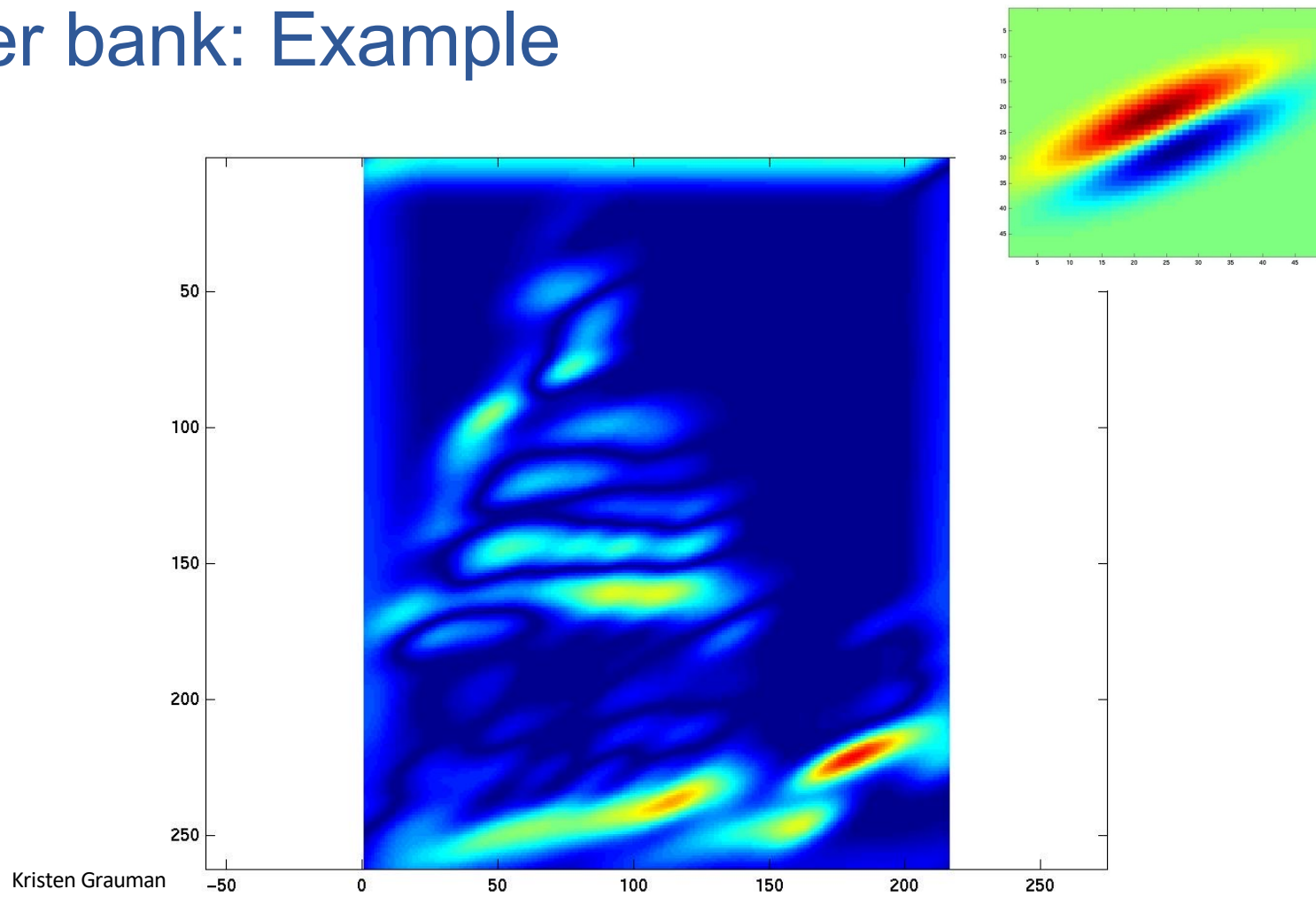
Filter bank: Example



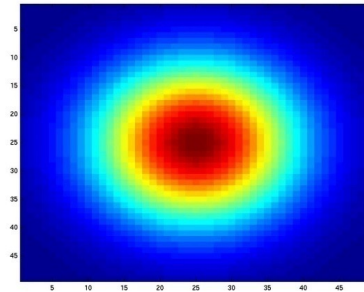
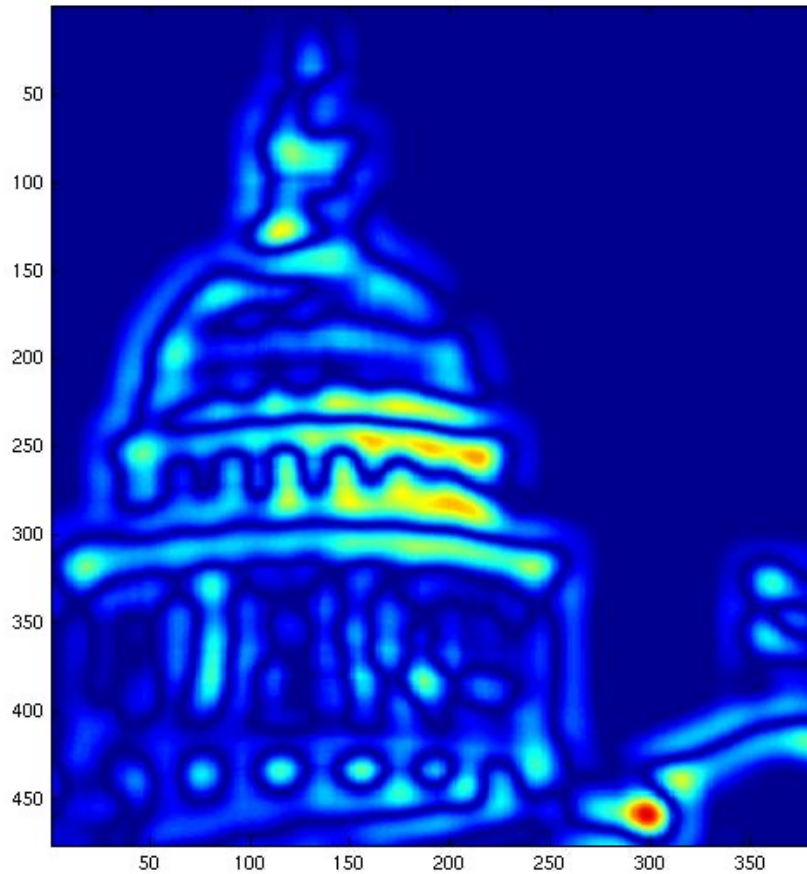
Filter bank: Example



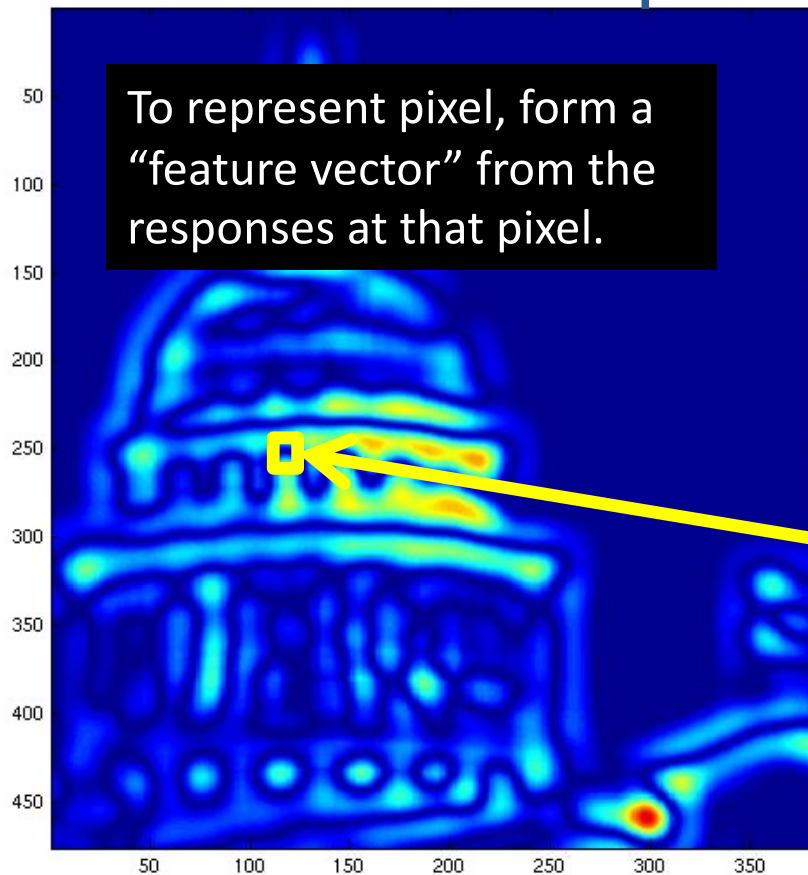
Filter bank: Example



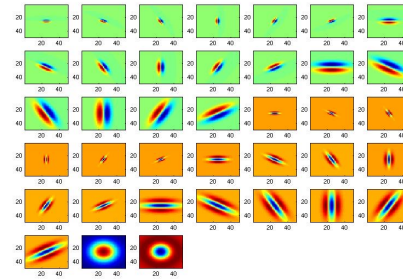
Filter bank: Example



Vectors of texture responses



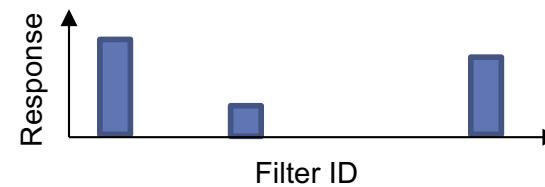
38 filters



1x38

vector
representation
feature

$[r_1, r_2, \dots, r_{38}]$



Adapted from Kristen Grauman

Vectors of texture responses

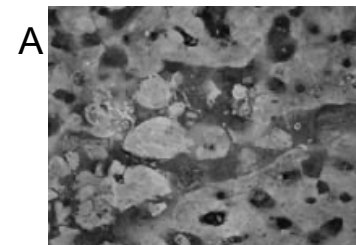
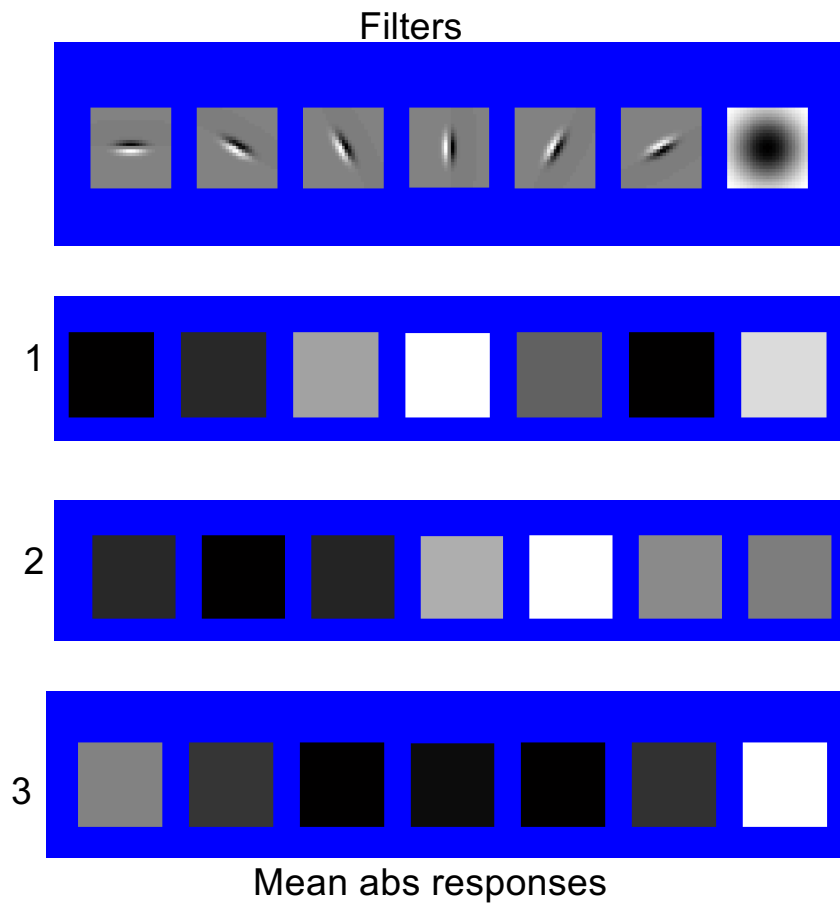
To represent pixel, form a “feature vector” from the responses at that pixel.

To represent *image*, compute statistics over all pixel feature vectors, e.g. their mean.

$$\begin{array}{c}
 [r_{(1,1)}^1, r_{(1,1)}^2, \dots, r_{(1,1)}^{38}] \\
 \uparrow \quad \swarrow \\
 [r_{(1,2)}^1, r_{(1,2)}^2, \dots, r_{(1,2)}^{38}] \\
 \text{Pixel location} \quad \text{Filter ID} \\
 \text{(row, column)} \\
 \dots \\
 [r_{(W,H)}^1, r_{(W,H)}^2, \dots, r_{(W,H)}^{38}]
 \end{array}$$

$$[\text{mean}(r_{(:,1)}^1), \text{mean}(r_{(:,2)}^2), \dots, \text{mean}(r_{(:,38)}^{38})]$$

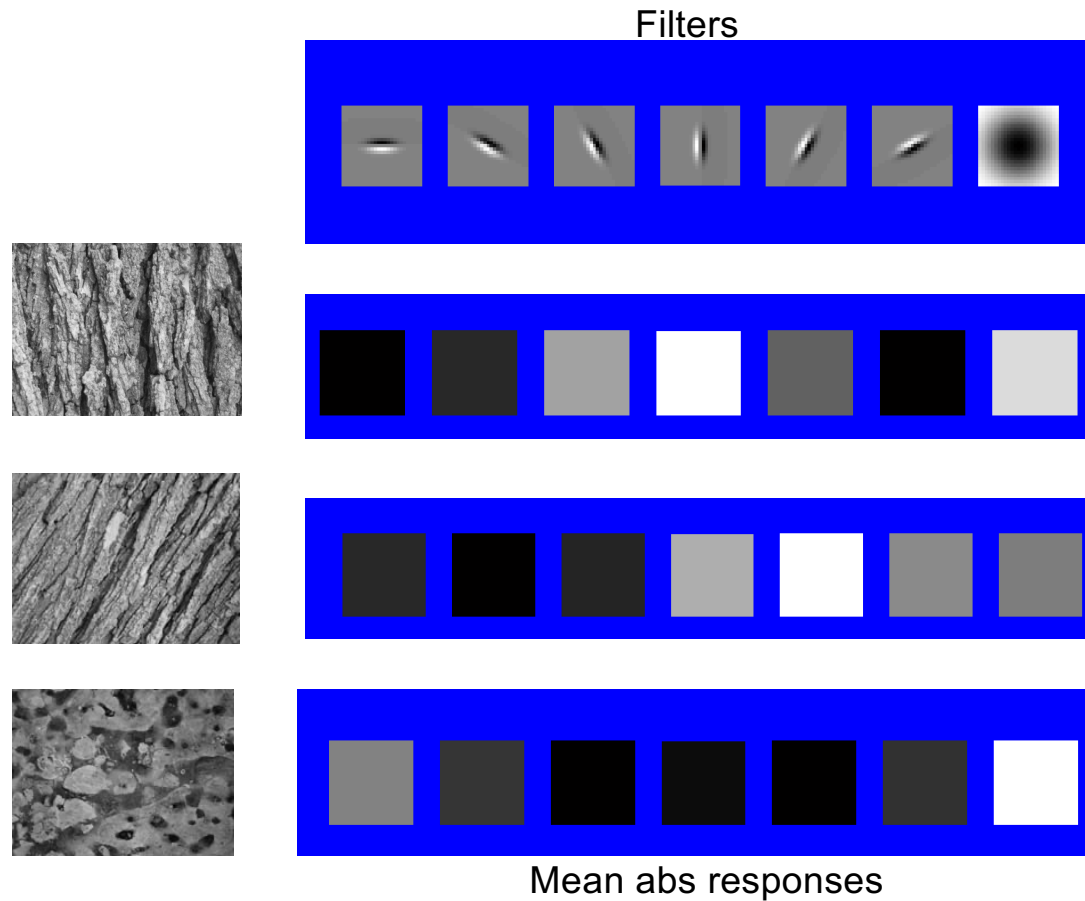
You try: Can you match the texture to the response?



White color
means higher
response

Derek Hoiem

Representing textures by mean absolute response



Derek Hoiem

Classifying materials, “stuff”

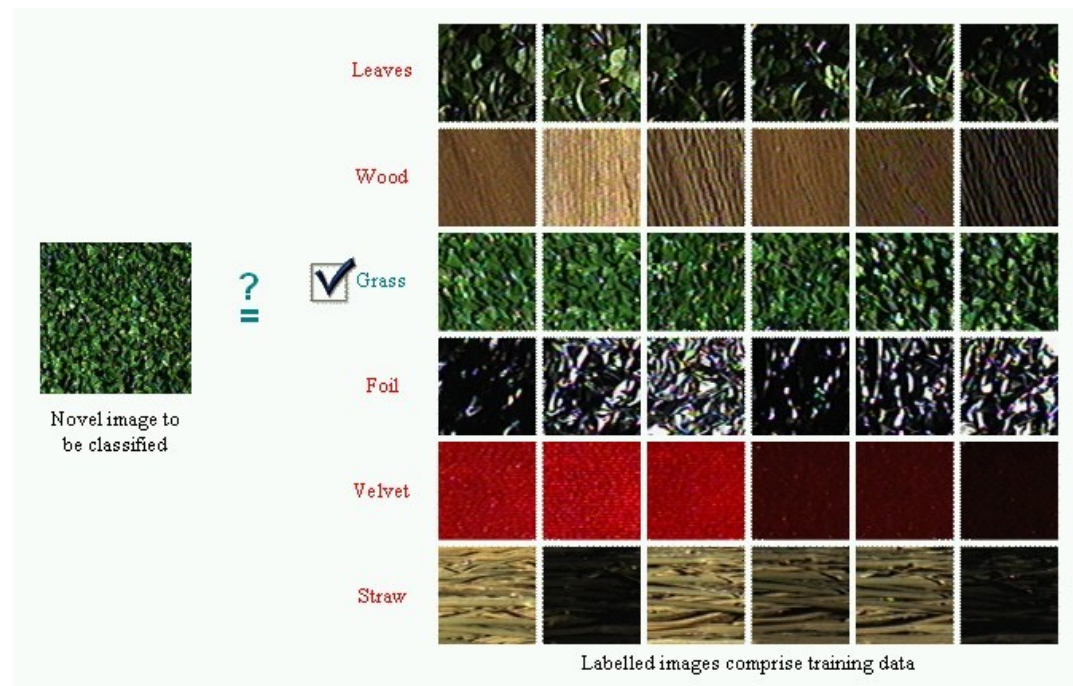


Figure by Varma & Zisserman

Summary

- Filters useful for
 - Enhancing images (smoothing, removing noise), e.g.
 - Box filter (linear)
 - Gaussian filter (linear)
 - Median filter
 - Detecting patterns (e.g. gradients)
- Texture is a useful property that is often indicative of materials, appearance cues
 - Texture representations summarize repeating patterns of local structure