

CS 1674: Intro to Computer Vision

Image Filtering and Texture

Prof. Adriana Kovashka
University of Pittsburgh
September 6, 2018

Plan for next two lectures

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

How images are represented (Matlab)

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

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

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

0.92

0.99

0.95

0.91

0.91

0.92

0.97

0.95

0.79

0.85

0.45

0.33

0.49

0.74

0.82

0.93

0.90

0.99

0.93

0.97

0.99

0.93

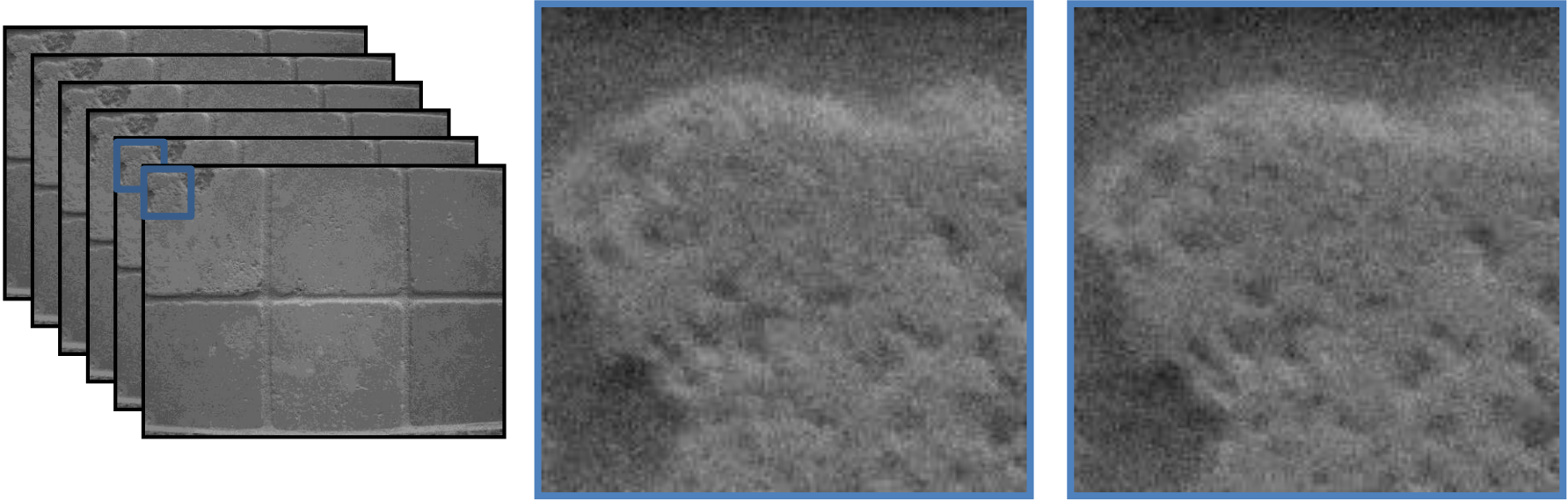
0.92

0.99

0.95

0.91

Enter: Noise



- We talked about how the same object will look very different across images
- Even multiple images of the same static scene will not 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

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



Original



Salt and pepper noise

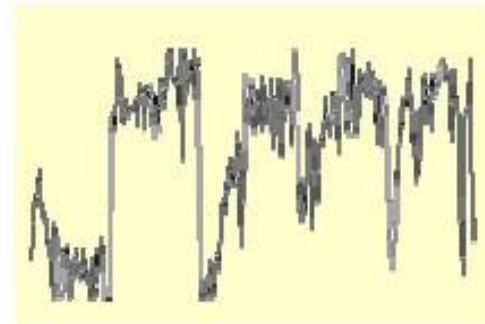
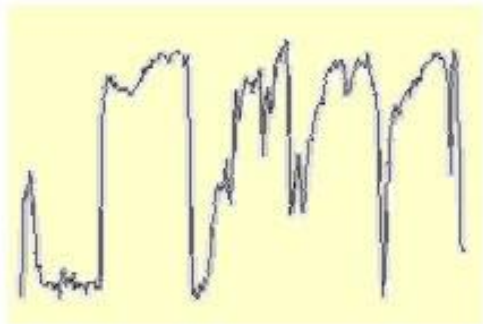
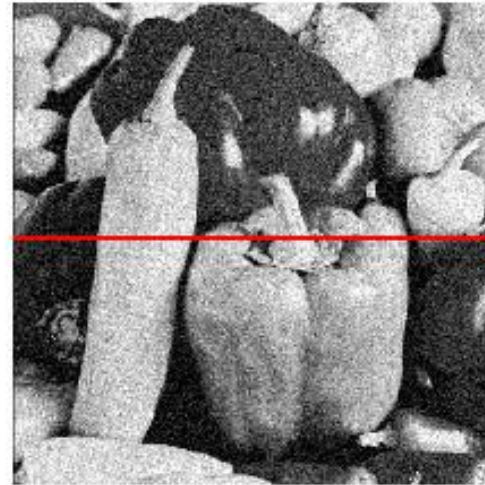


Impulse noise



Gaussian noise

Gaussian noise



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

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

```
>> noise = randn(size(im)).*sigma;  
>> output = im + noise;
```

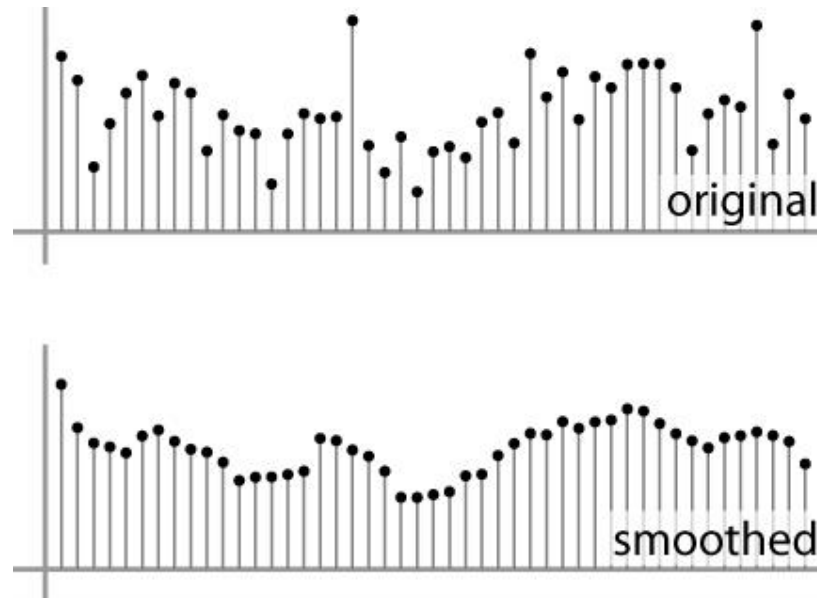
What is impact of the sigma?

First attempt at a solution

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

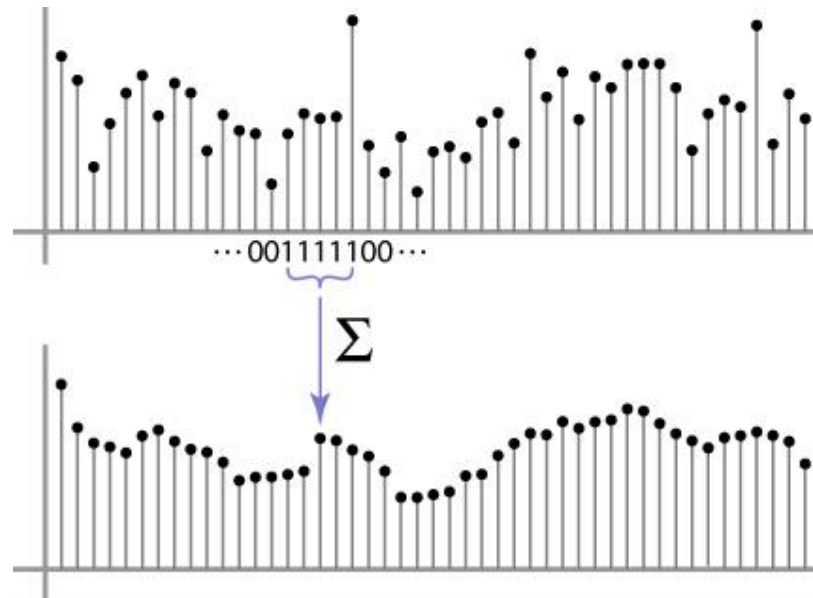
First attempt at a solution

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



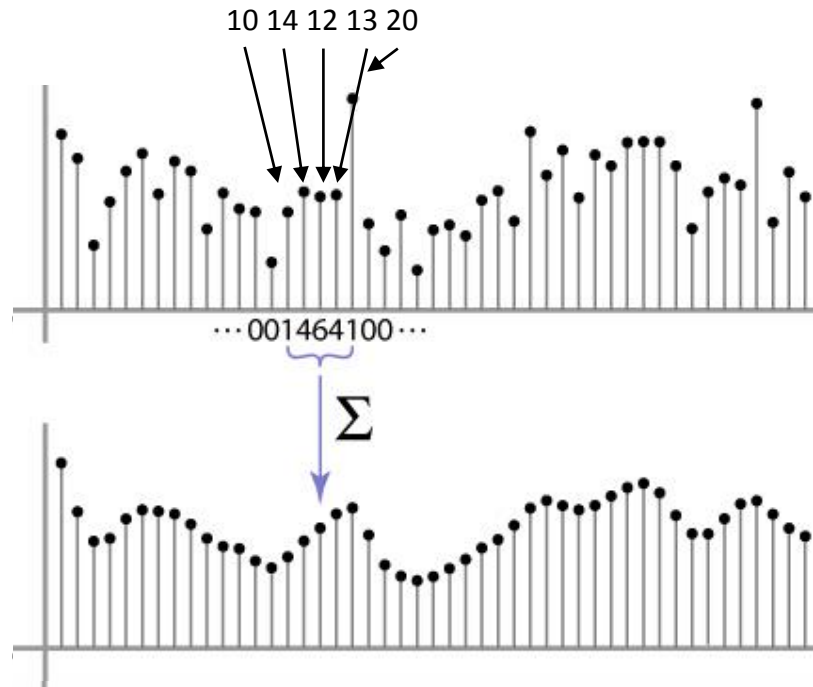
Weighted Moving Average

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



Weighted Moving Average

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



Central pixel =
 $10 \cdot 1 +$
 $14 \cdot 4 +$
 $12 \cdot 6 +$
 $13 \cdot 4 +$
 $20 \cdot 1$

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							

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	

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)

Correlation filtering

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

$$G[i, j] = \underbrace{\frac{1}{(2k+1)^2}}_{\text{Attribute uniform weight to each pixel}} \underbrace{\sum_{u=-k}^k \sum_{v=-k}^k F[i+u, j+v]}_{\text{Loop over all pixels in neighborhood around image pixel } F[i,j]}$$

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

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

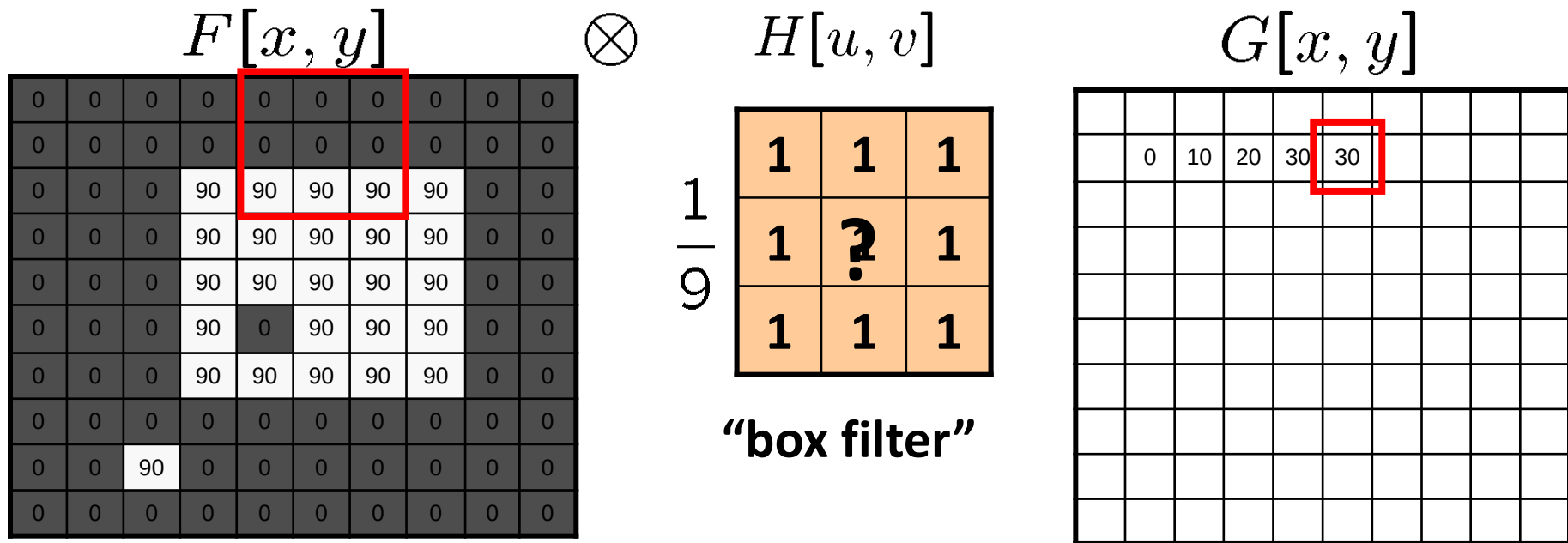
This is called **cross-correlation**, denoted $G = H \otimes F$

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 filter

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



$$G = H \otimes F$$

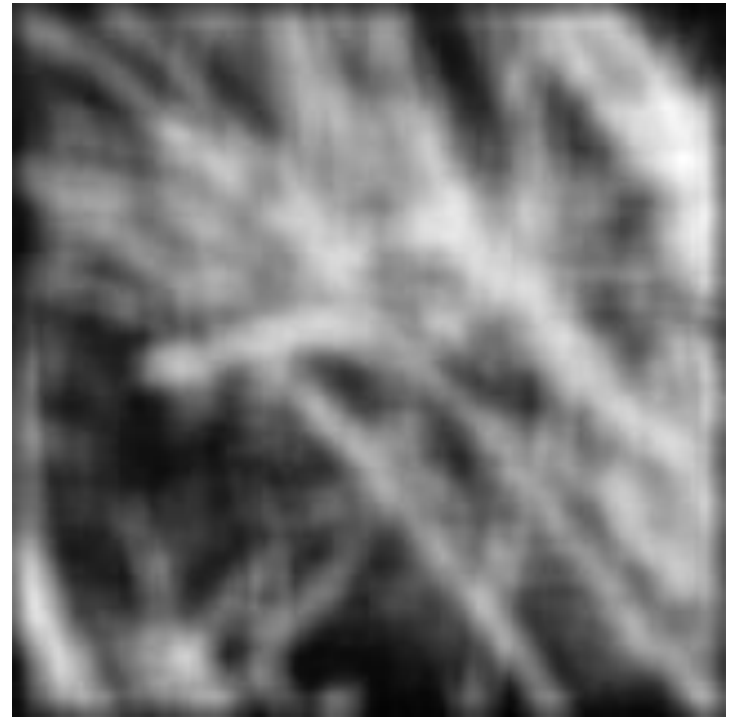
Smoothing by averaging



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?

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

$F[x, y]$

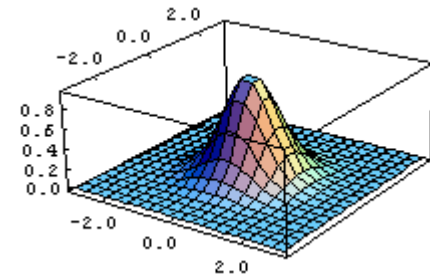
1	2	1
2	4	2
1	2	1

$\frac{1}{16}$

$H[u, v]$

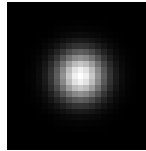
This kernel is an approximation of a 2d Gaussian function:

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



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

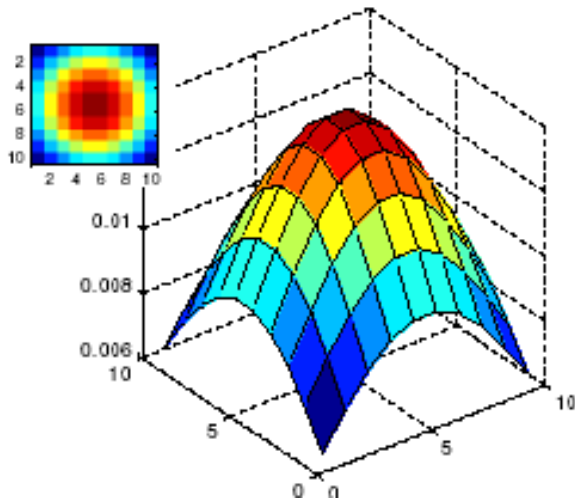
Smoothing with a Gaussian



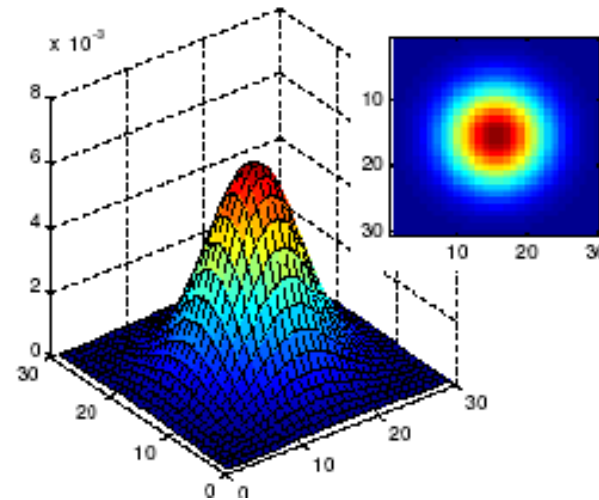
Vs box filter

Gaussian filters

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



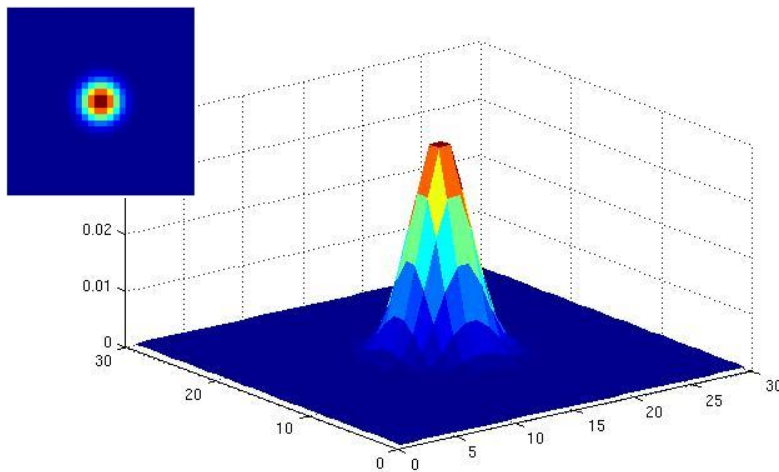
$\sigma = 5$ with 10
x 10 kernel



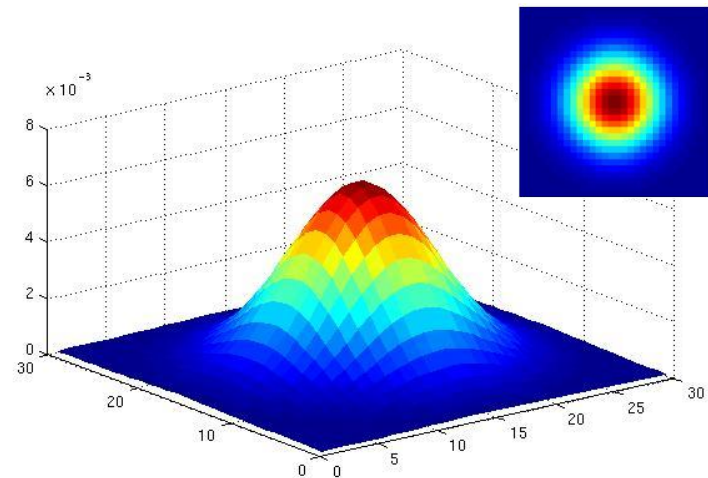
$\sigma = 5$ with 30
x 30 kernel

Gaussian filters

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



$\sigma = 2$ with 30
x 30 kernel

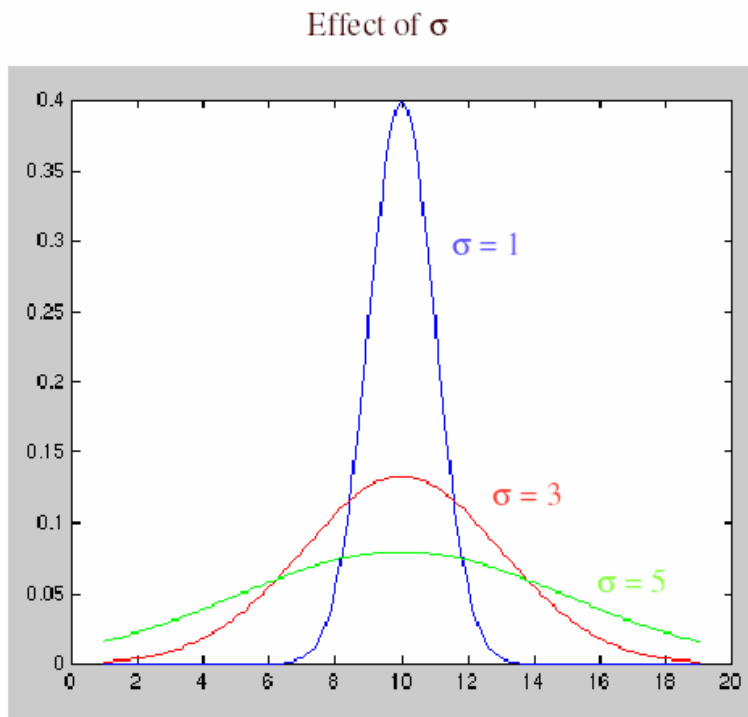


$\sigma = 5$ with 30
x 30 kernel

Gaussian filters

How big should the filter be?

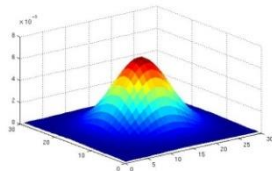
- Values at edges should be near zero ← important!
- Rule of thumb for Gaussian: set filter half-width to about 3σ



Gaussian filter in Matlab

```
>> hsize = 10;  
>> sigma = 5;  
>> h = fspecial('gaussian' hsize, sigma);
```

```
>> mesh(h) ;
```



```
>> imagesc(h) ;
```



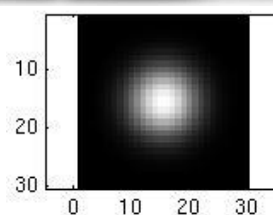
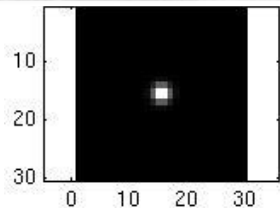
```
>> outim = imfilter(im, h) ; % correlation  
>> imshow(outim) ;
```



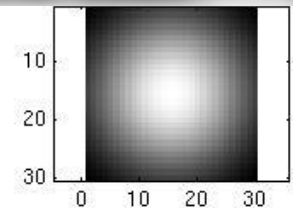
outim

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('gaussian', fsize, sigma);
    out = imfilter(im, h);
    imshow(out);
    pause;
end
```

Convolution

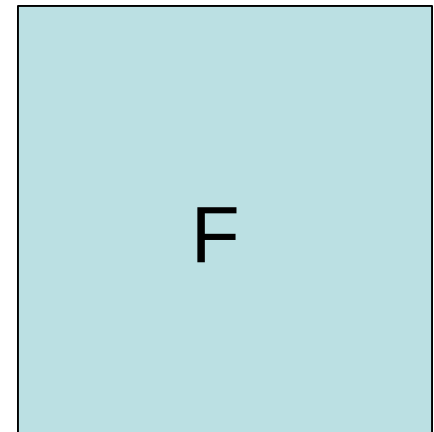
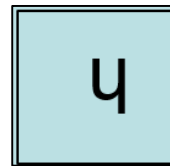
- Convolution:
 - Flip the filter in both dimensions (bottom to top, right to left)
 - Then apply cross-correlation

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

$$G = H \star F$$



*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

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

$$G = H \otimes F$$

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

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

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

$$G = H \otimes F$$

$$\begin{aligned} u &= -1, v = -1 \\ v &= 0 \end{aligned}$$

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

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

$$G = H \otimes F$$

$$\begin{aligned} u &= -1, v = -1 \\ v &= 0 \end{aligned}$$

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 smoothing filters

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

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

?

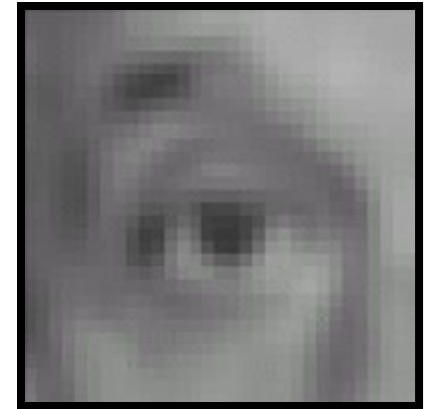
Practice with linear filters



Original

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

1	1	1
1	1	1
1	1	1



**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



Original

0	0	0
0	2	0
0	0	0

—

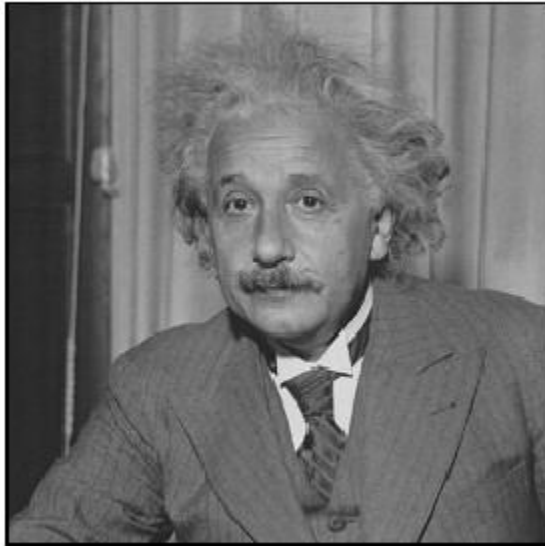
$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

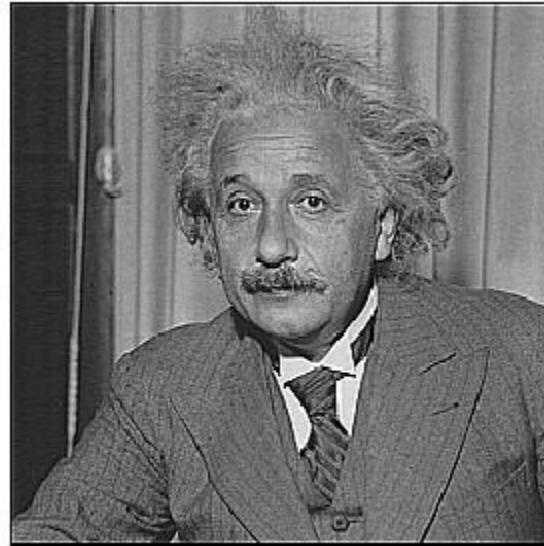


Sharpening filter:
accentuates differences with
local average

Sharpening



before

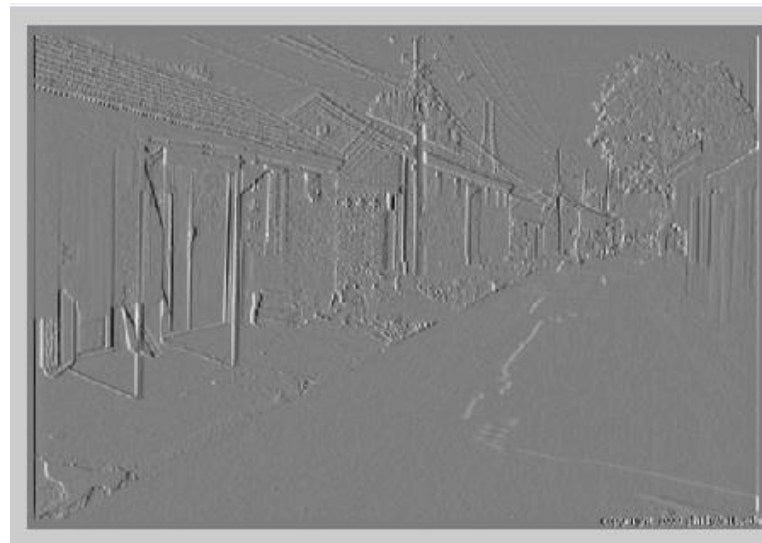


after

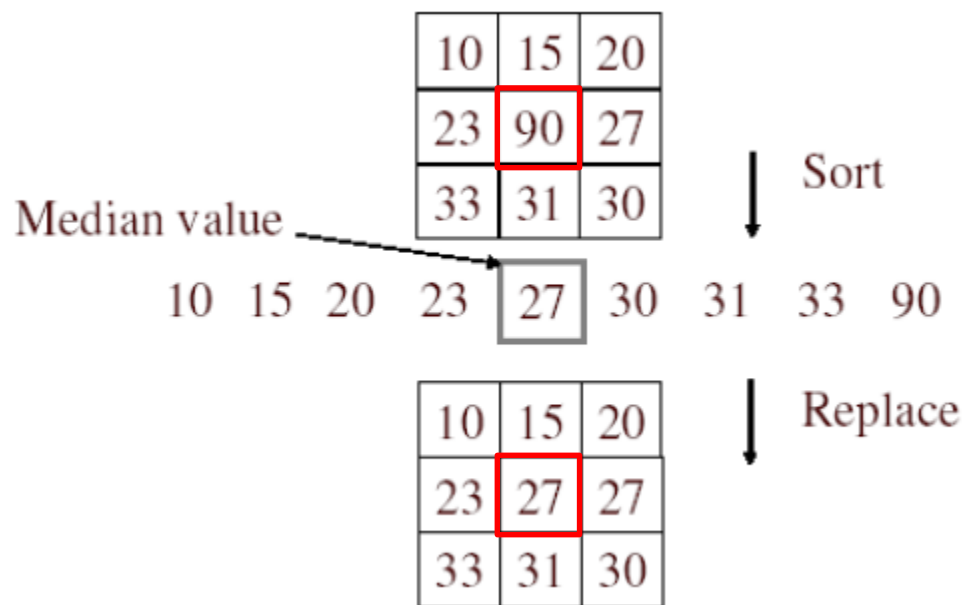
Filters for computing *gradients*

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

intensity image



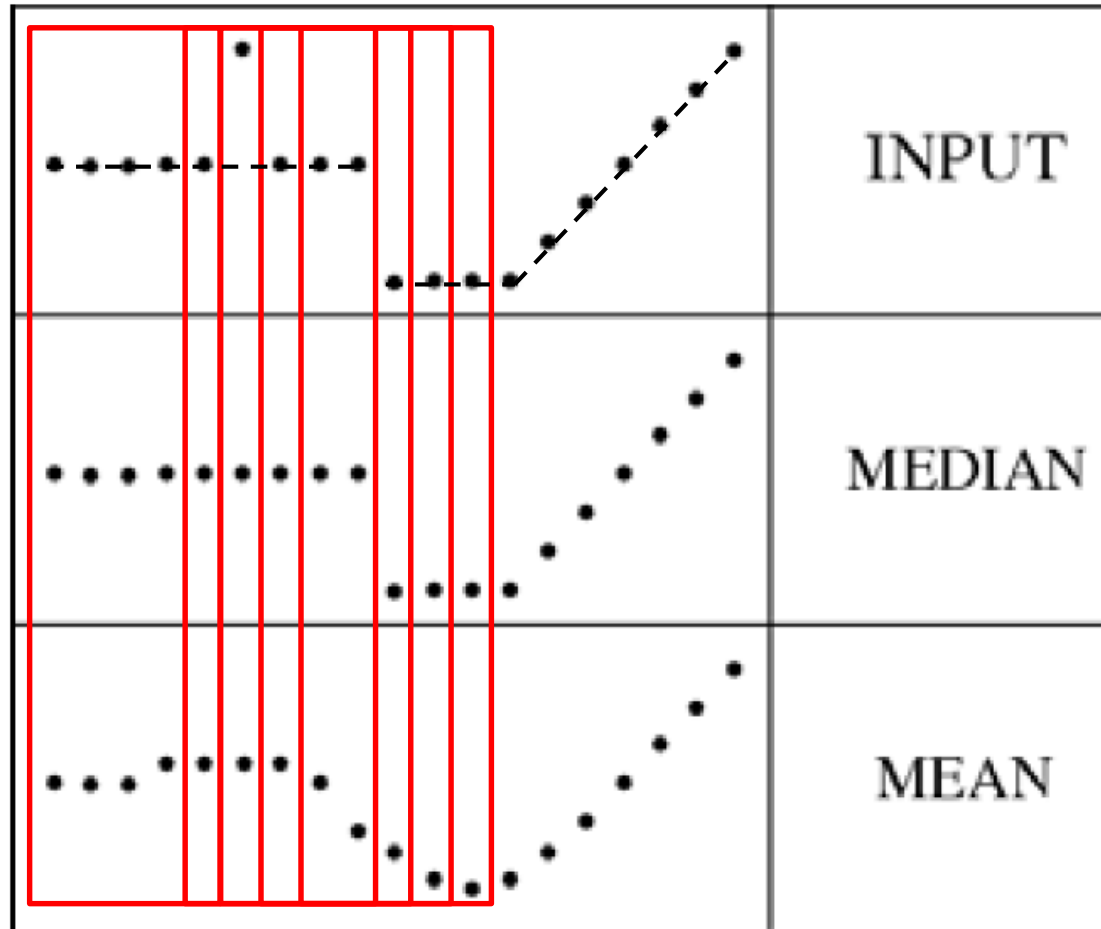
Median filter



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

Median filter

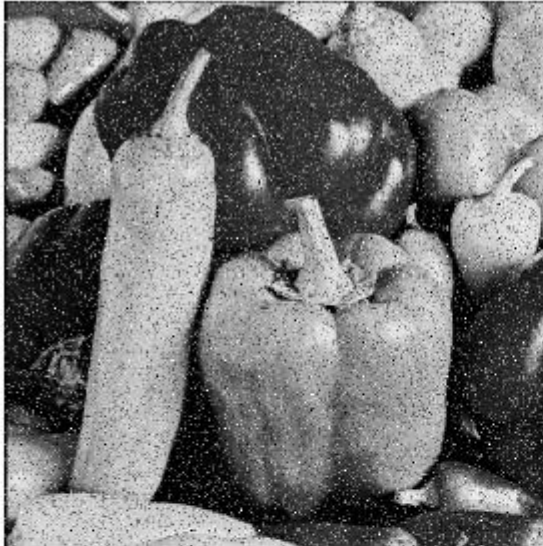
- Median filter is edge preserving



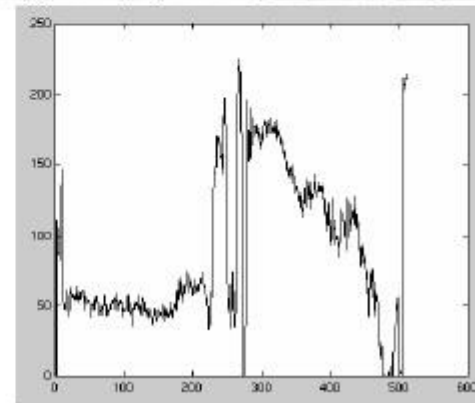
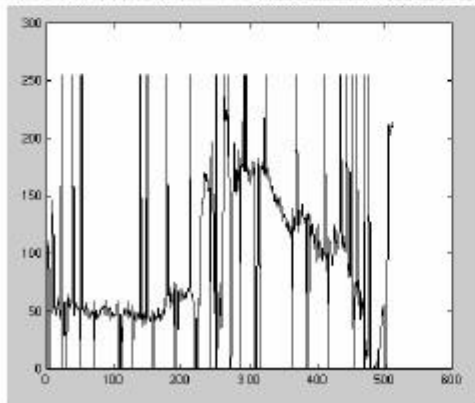
DEMO: filtering.m

Median filter

Salt and
pepper noise



Median
filtered



Plots of a row of the image

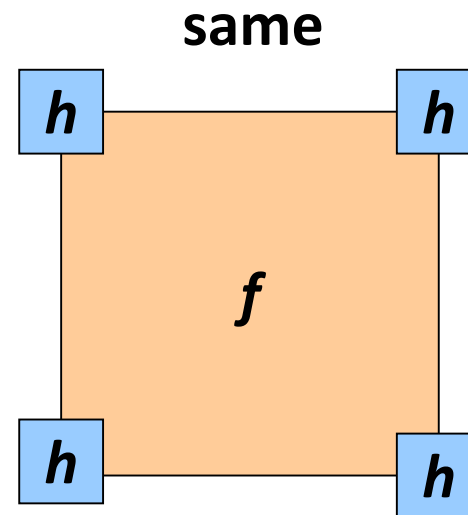
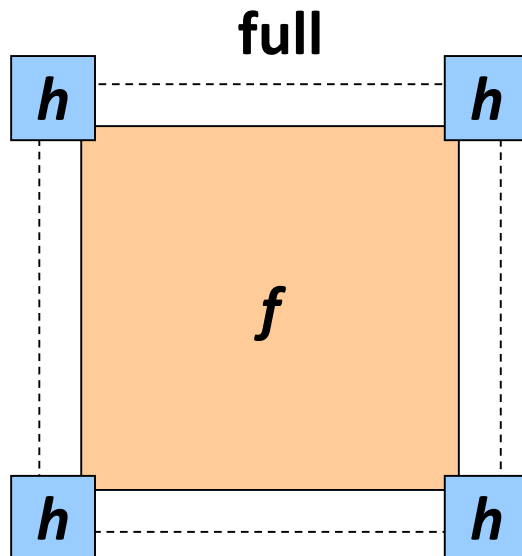
Matlab: `output_im = medfilt2(im, [h w]);`

f = image
 h = filter

Boundary issues

	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	

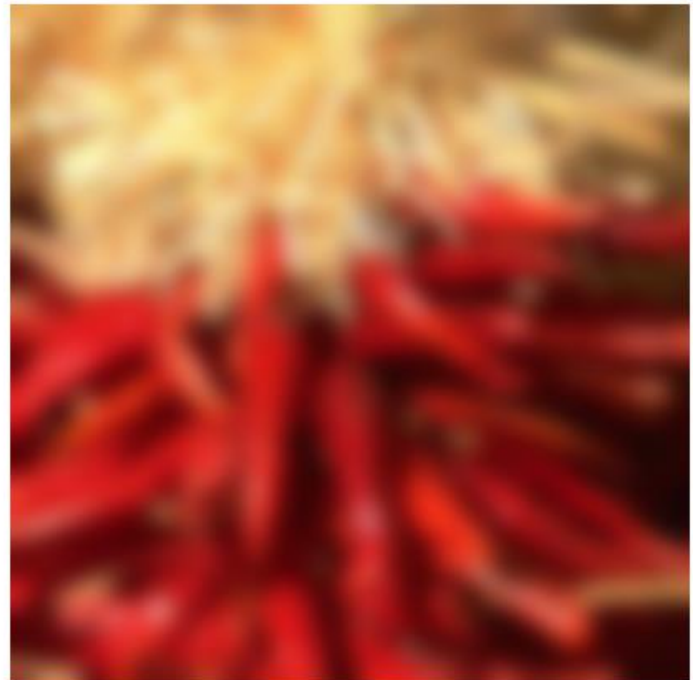
- What is the size of the output?
 - ‘full’: output size is larger than the size of f
 - ‘same’: output size is same as f



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



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:

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

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:

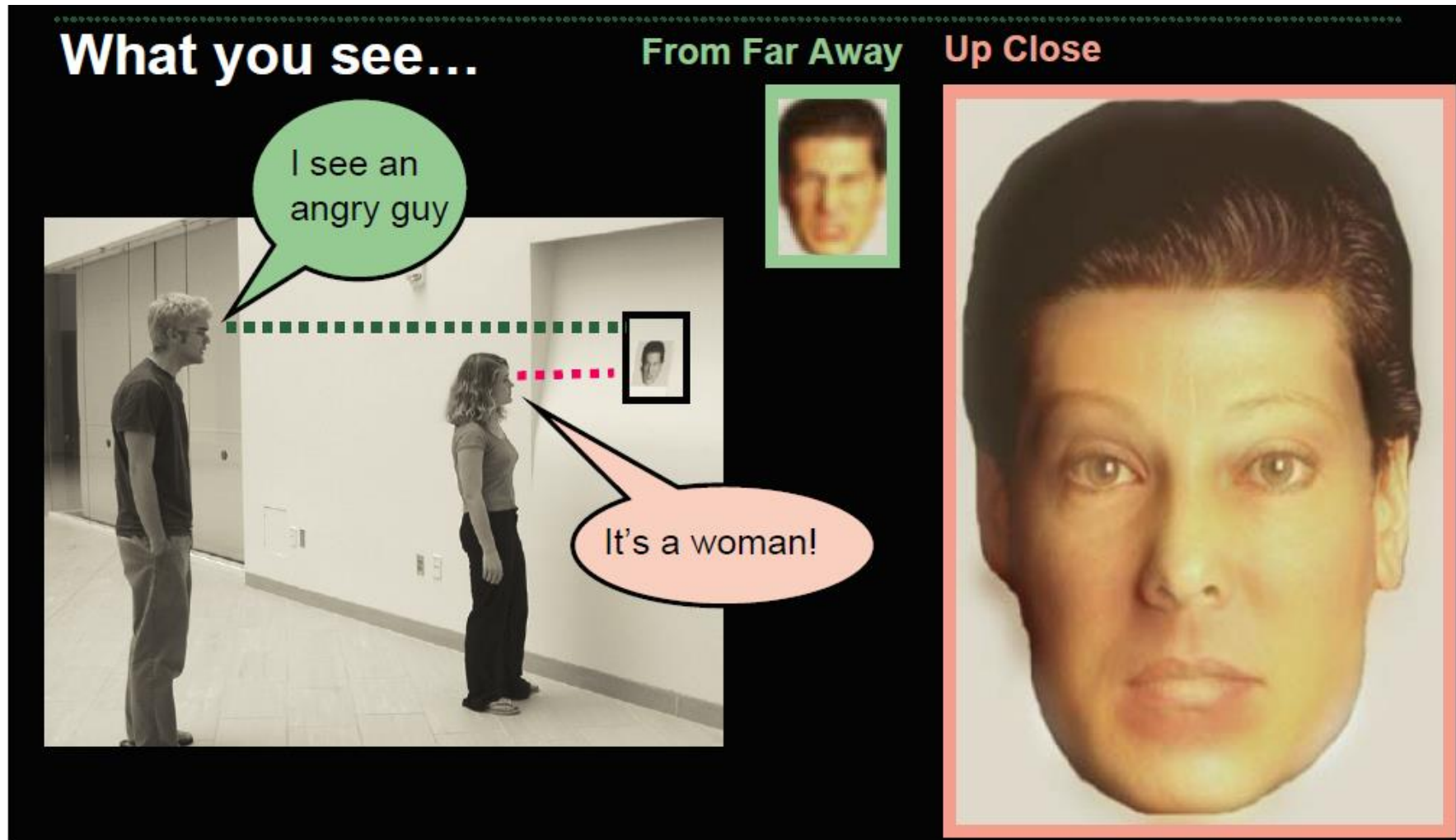
$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} = \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline \end{array}$$

Perform filtering
along rows:

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

Followed by filtering
along the remaining column:

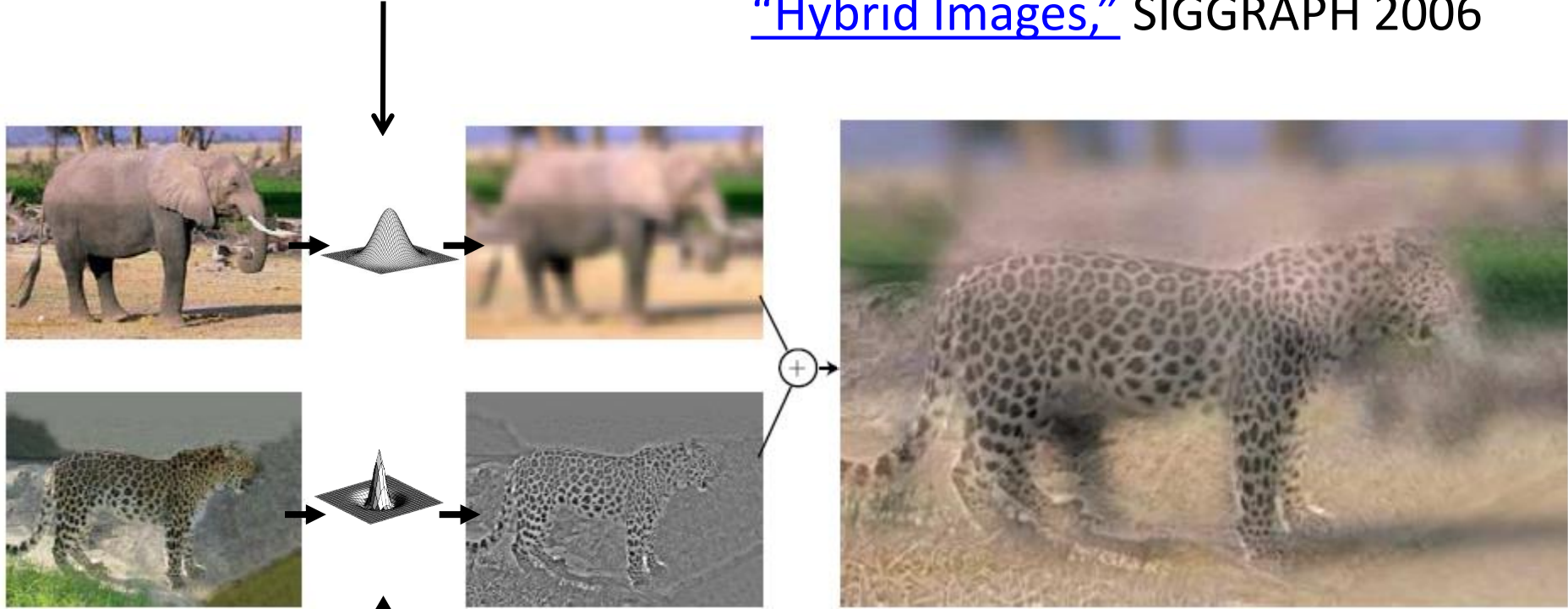
Application: Hybrid Images



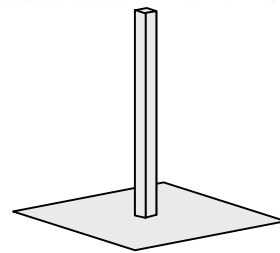
Application: Hybrid Images

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

Gaussian Filter

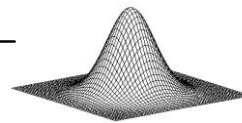


Laplacian Filter
(sharpening)



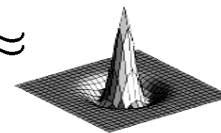
unit impulse

−



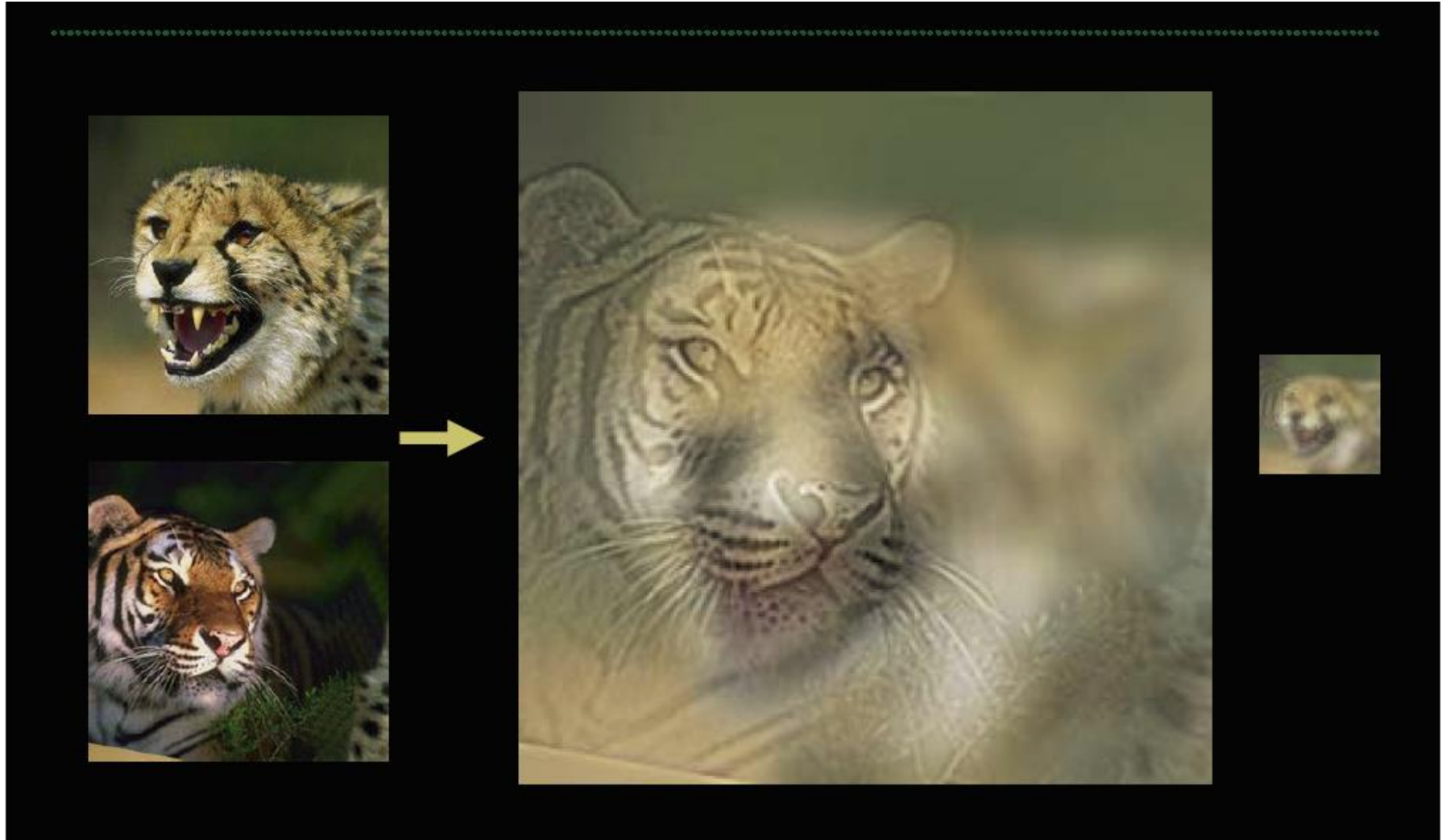
Gaussian

≈



Laplacian of Gaussian

Application: Hybrid Images



Application: Hybrid Images

Changing expression



SIGGRAPH2006

Sad



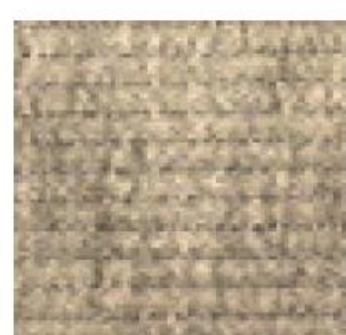
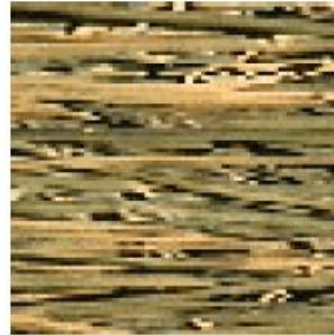
Surprised



Plan for next two lectures

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

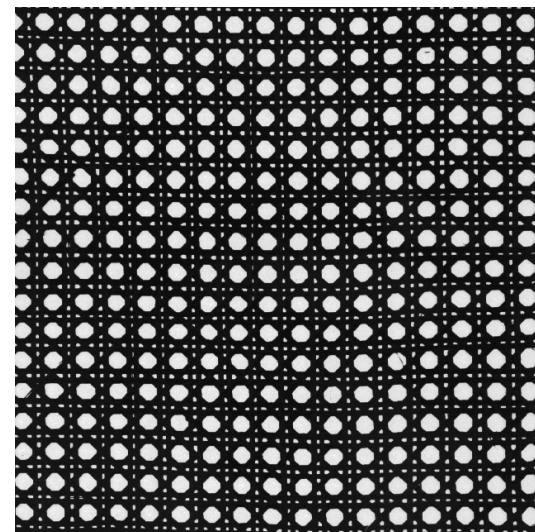
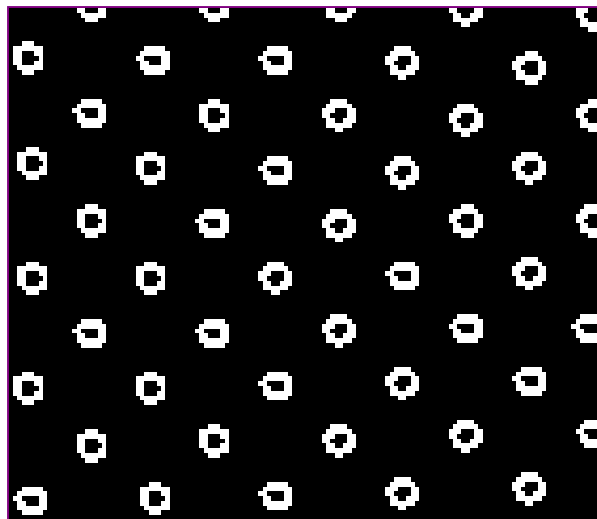
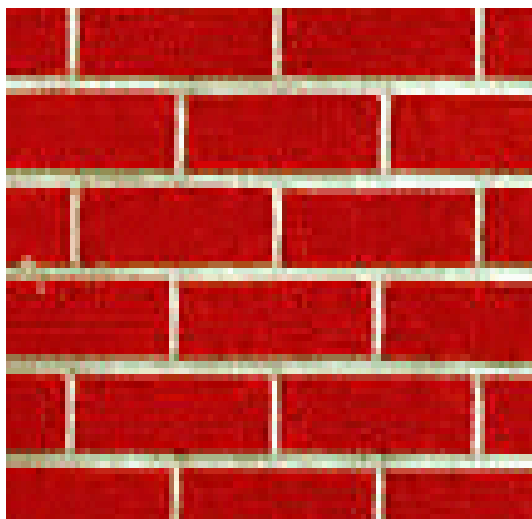
Texture



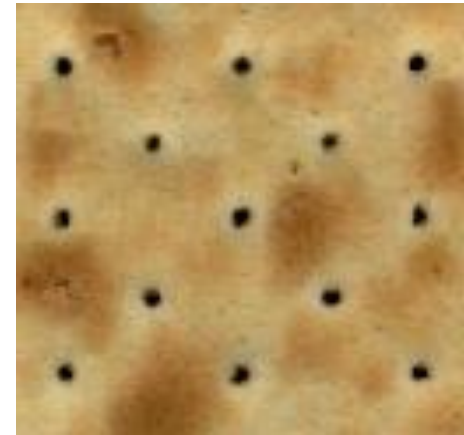
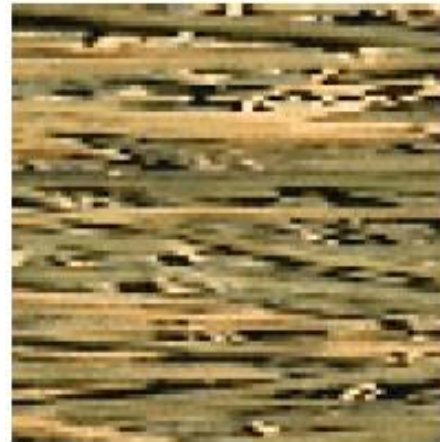
Due to:

Patterns, marks, etches, blobs, holes, relief, etc.

Includes: more regular patterns



Includes: more random patterns







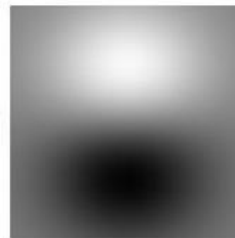
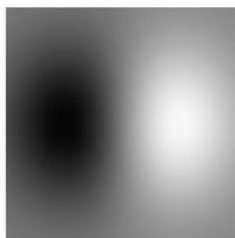
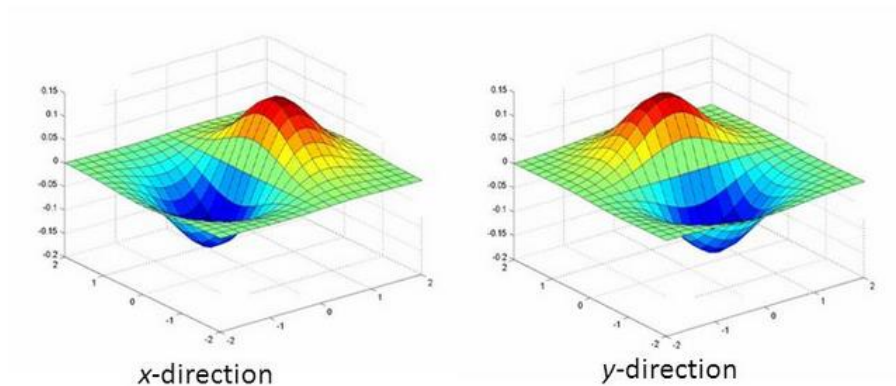
Why analyze texture?

- 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

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

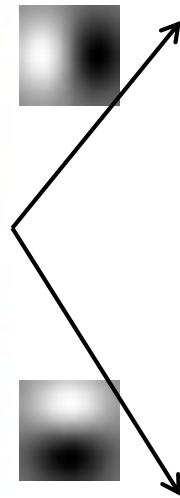
Derivative of Gaussian filter



Texture representation: example



original image



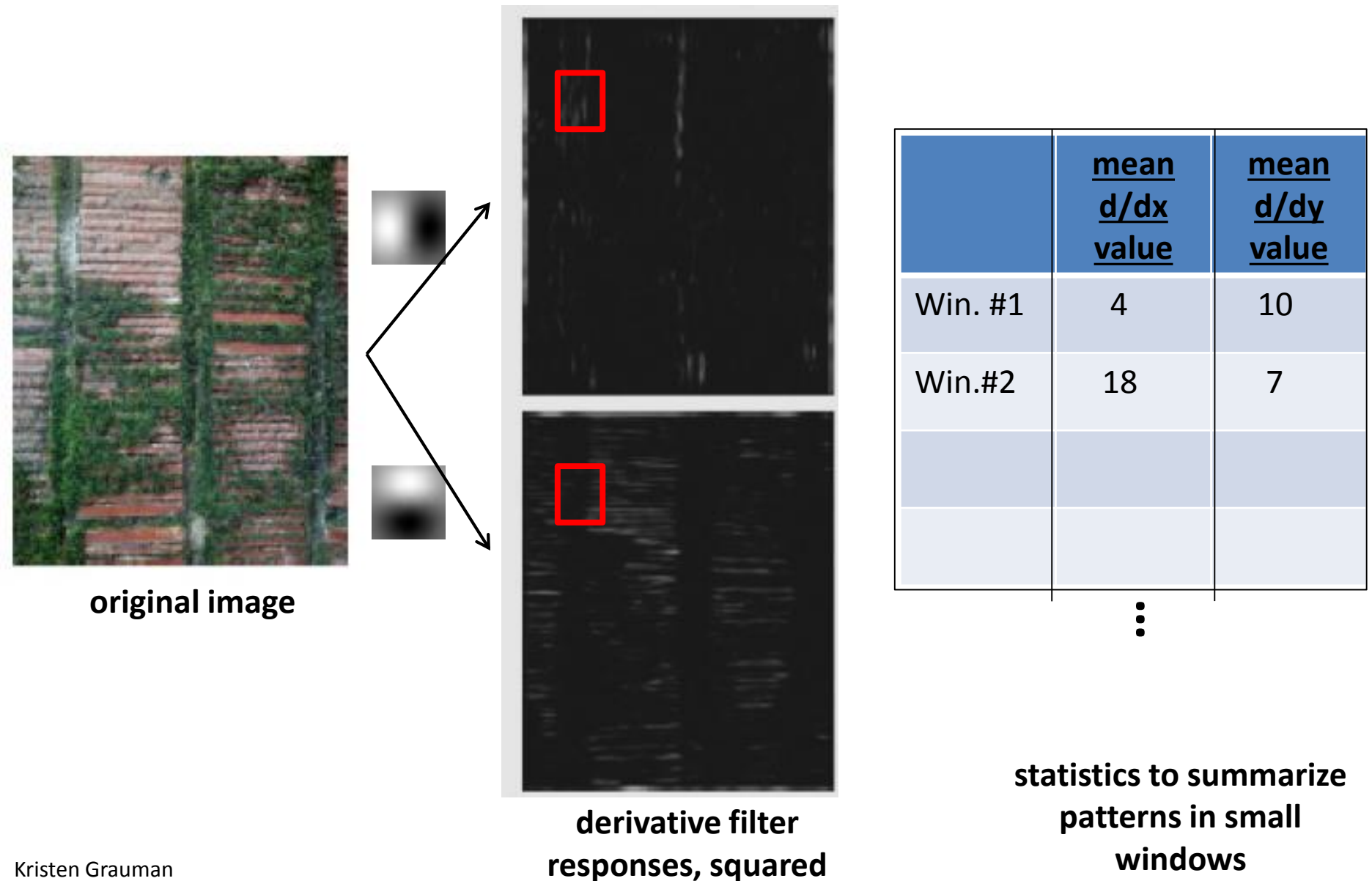
derivative filter
responses, squared

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

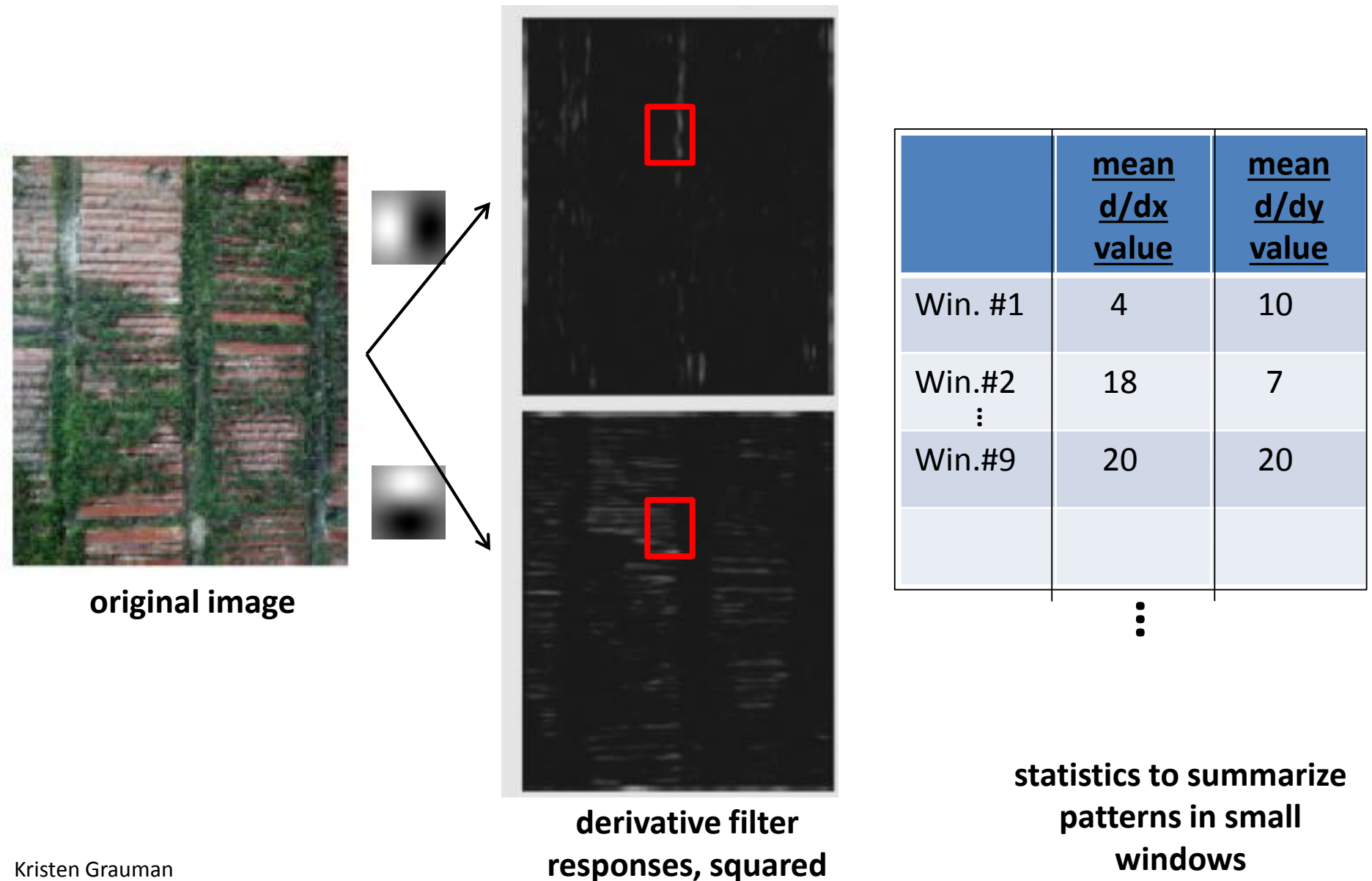
⋮

statistics to summarize patterns in small windows

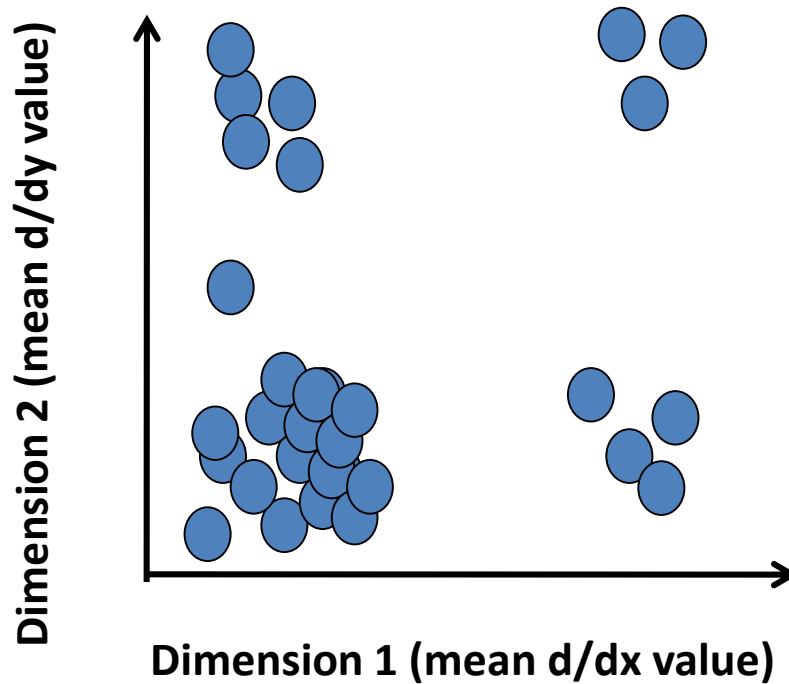
Texture representation: example



Texture representation: example



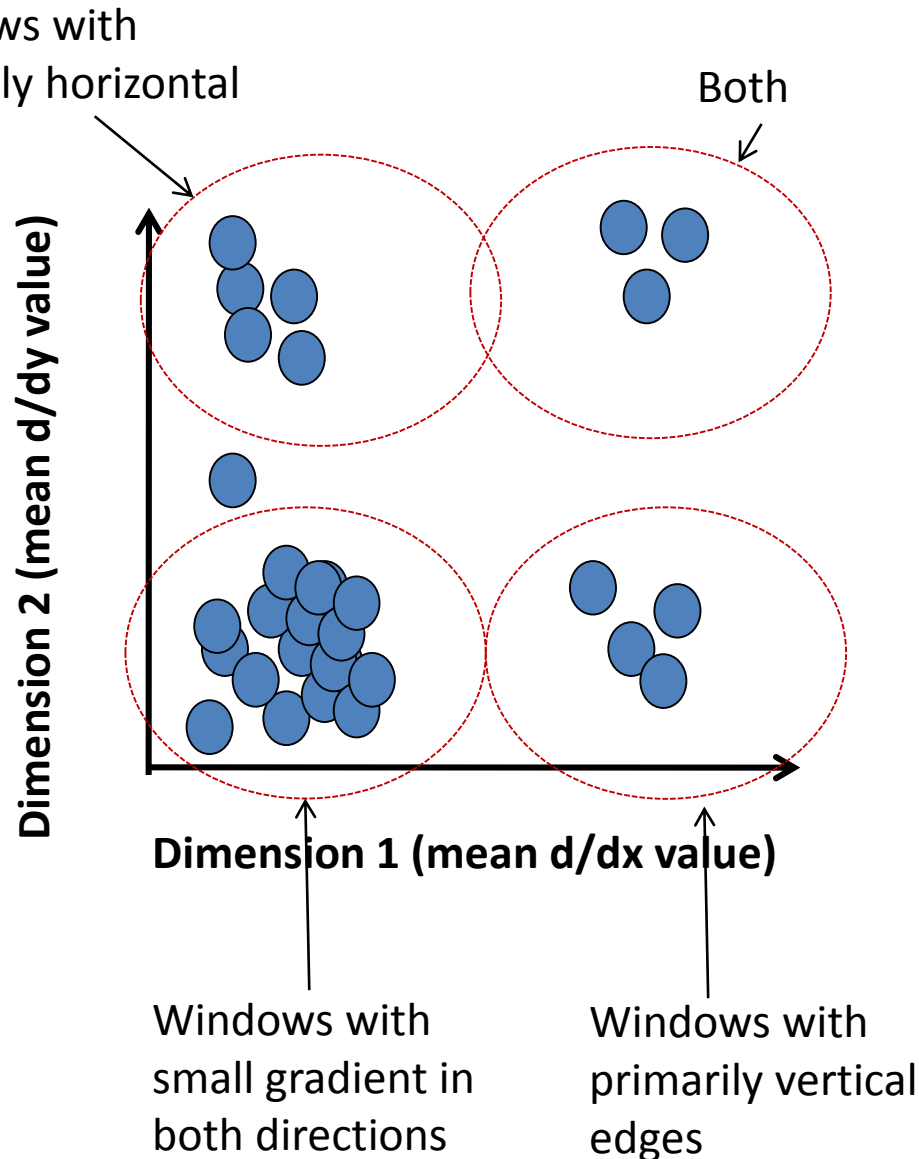
Texture representation: example



	<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



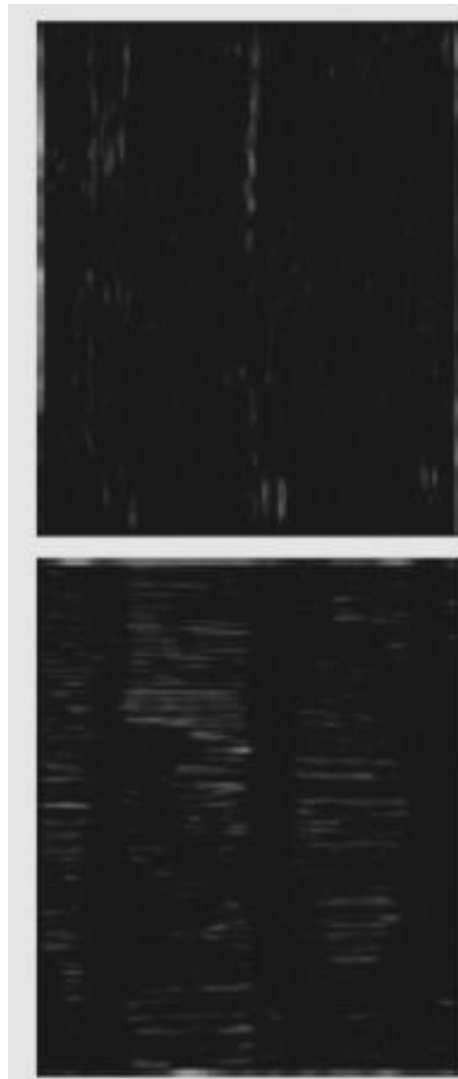
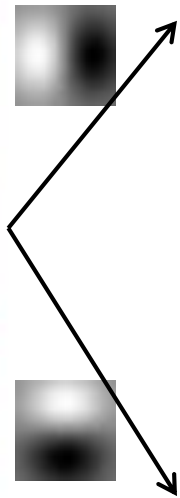
	<u>mean d/dx value</u>	<u>mean d/dy value</u>
Win. #1	4	10
Win.#2	18	7
⋮		
Win.#9	20	20
	⋮	

statistics to summarize
patterns in small
windows

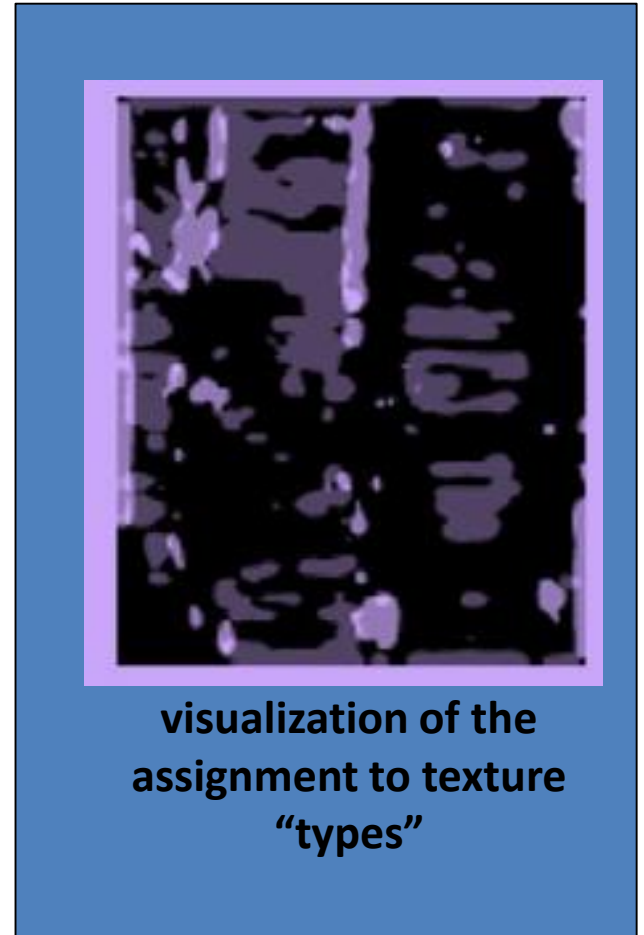
Texture representation: example



original image

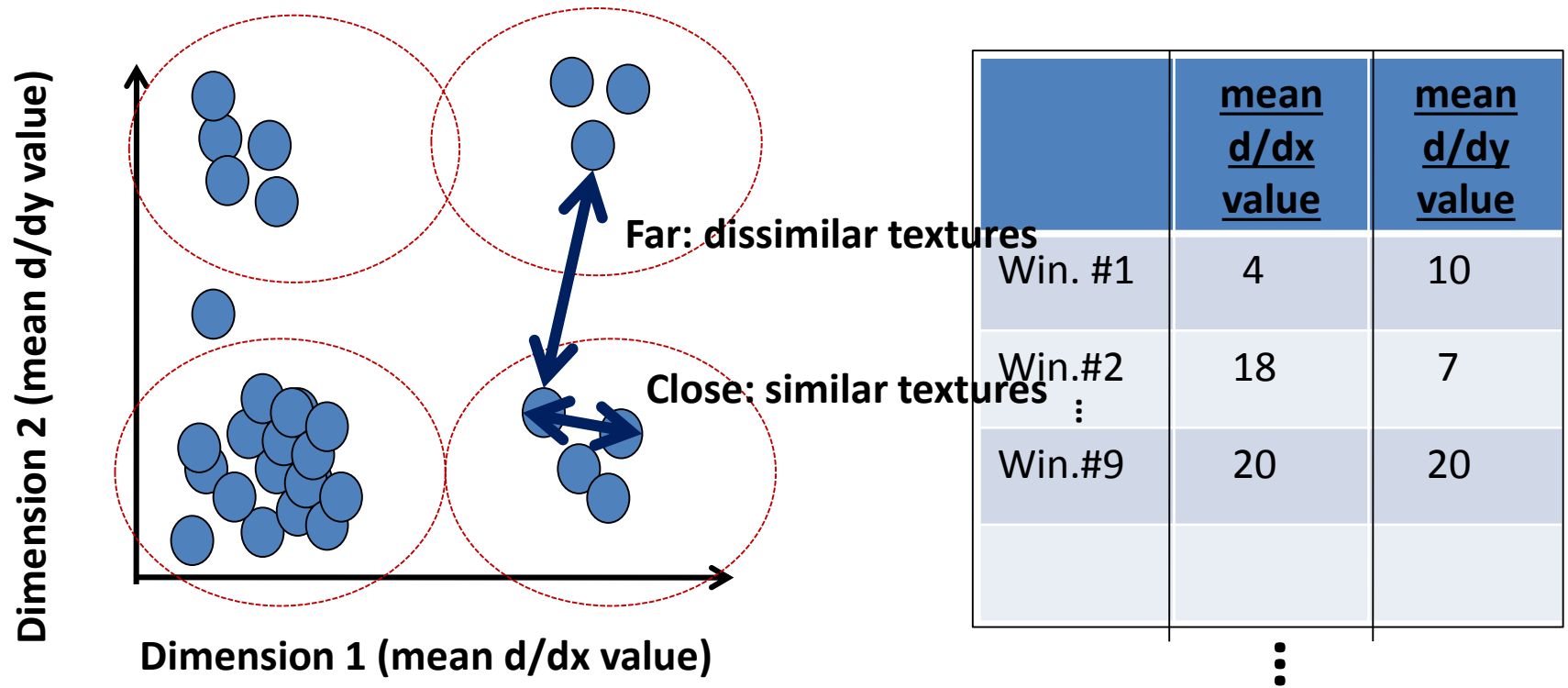


derivative filter
responses, squared



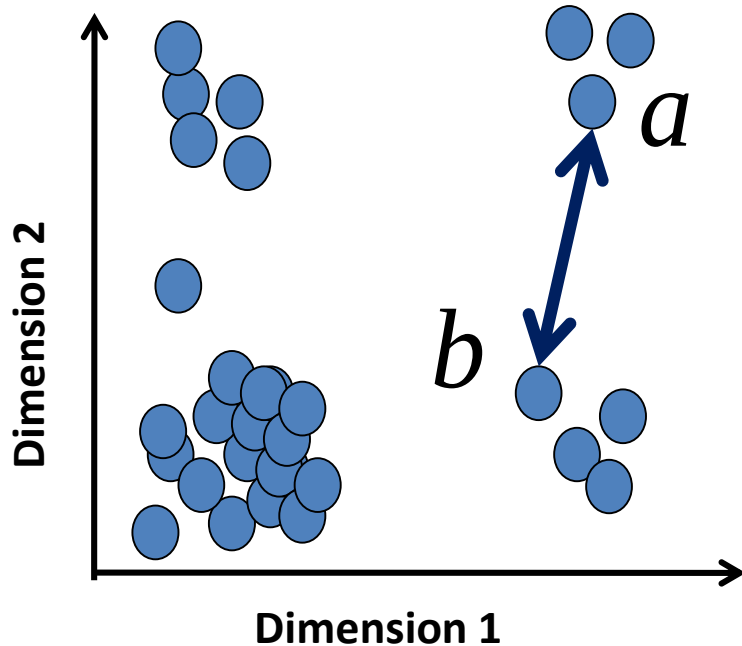
visualization of the
assignment to texture
“types”

Texture representation: example



statistics to summarize
patterns in small
windows

Computing distances using texture

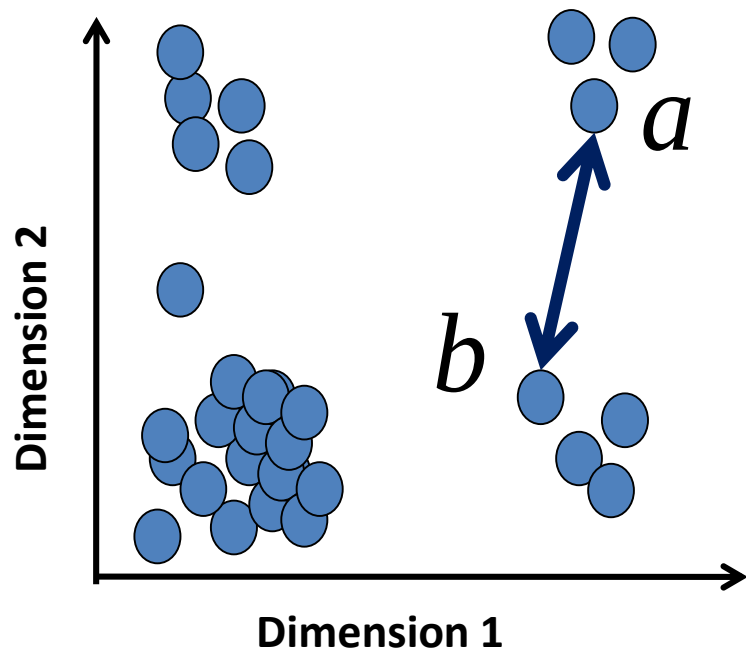


$$D(a, b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}$$

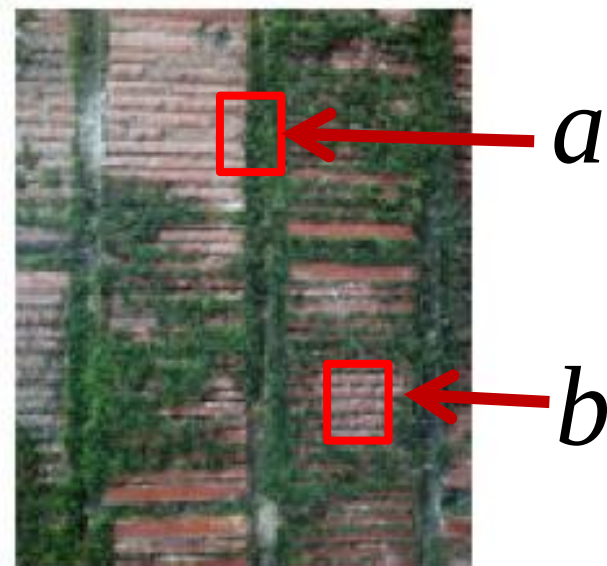
$$= \sqrt{\sum_{i=1}^d (a_i - b_i)^2}$$

Euclidean distance (L_2)

Texture representation: example



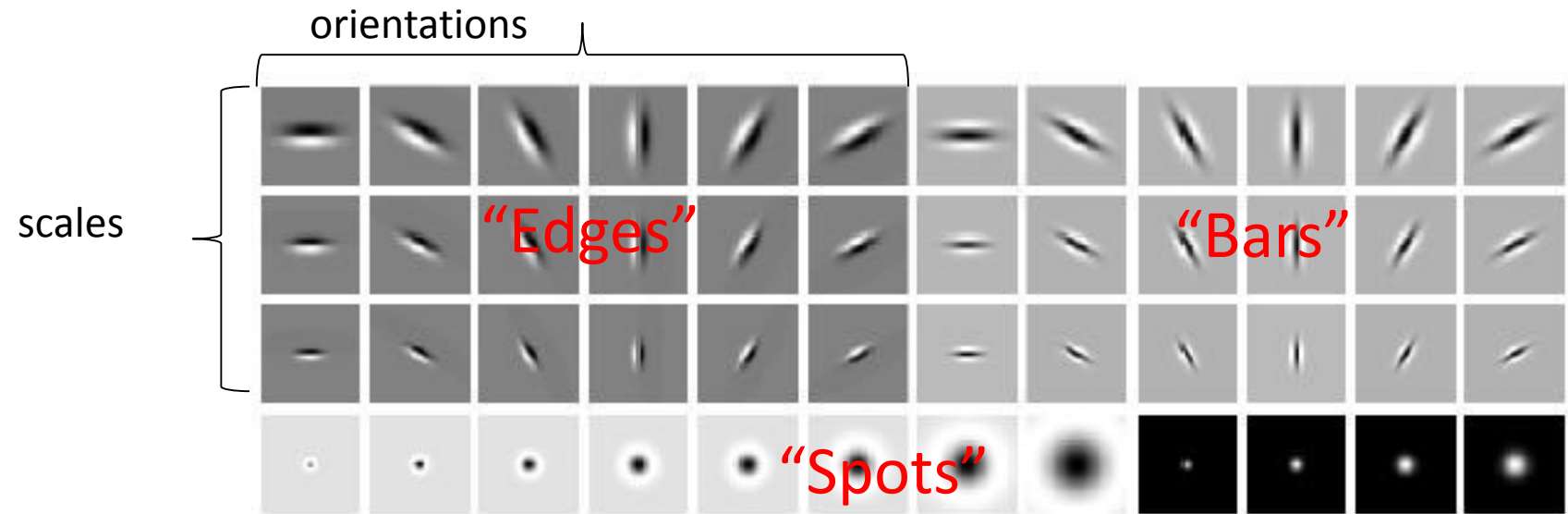
Distance reveals how dissimilar texture from window a is from texture in window b.



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

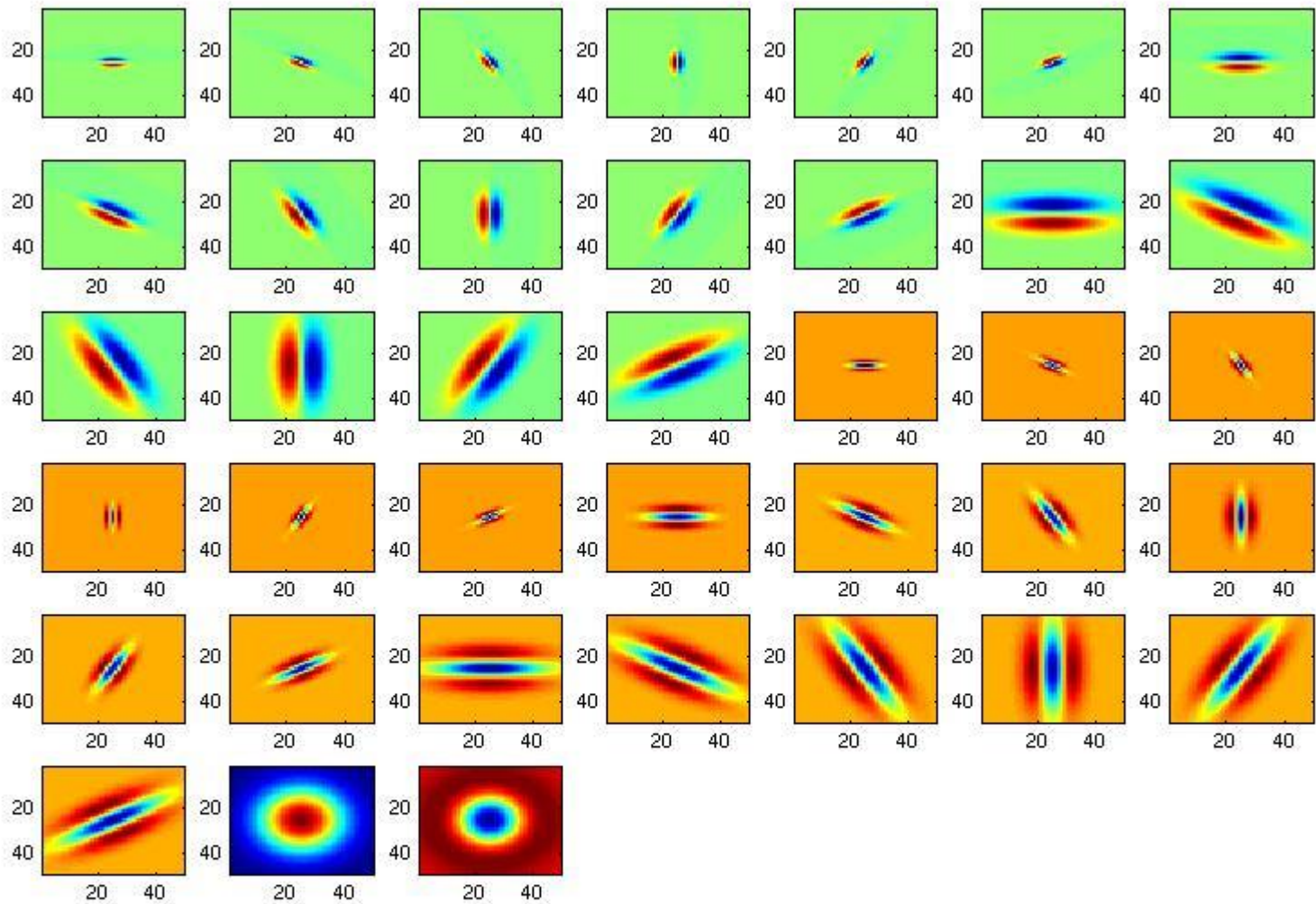


- What filters to put in the bank?
 - Typically we want a combination of scales and orientations, different types of patterns.

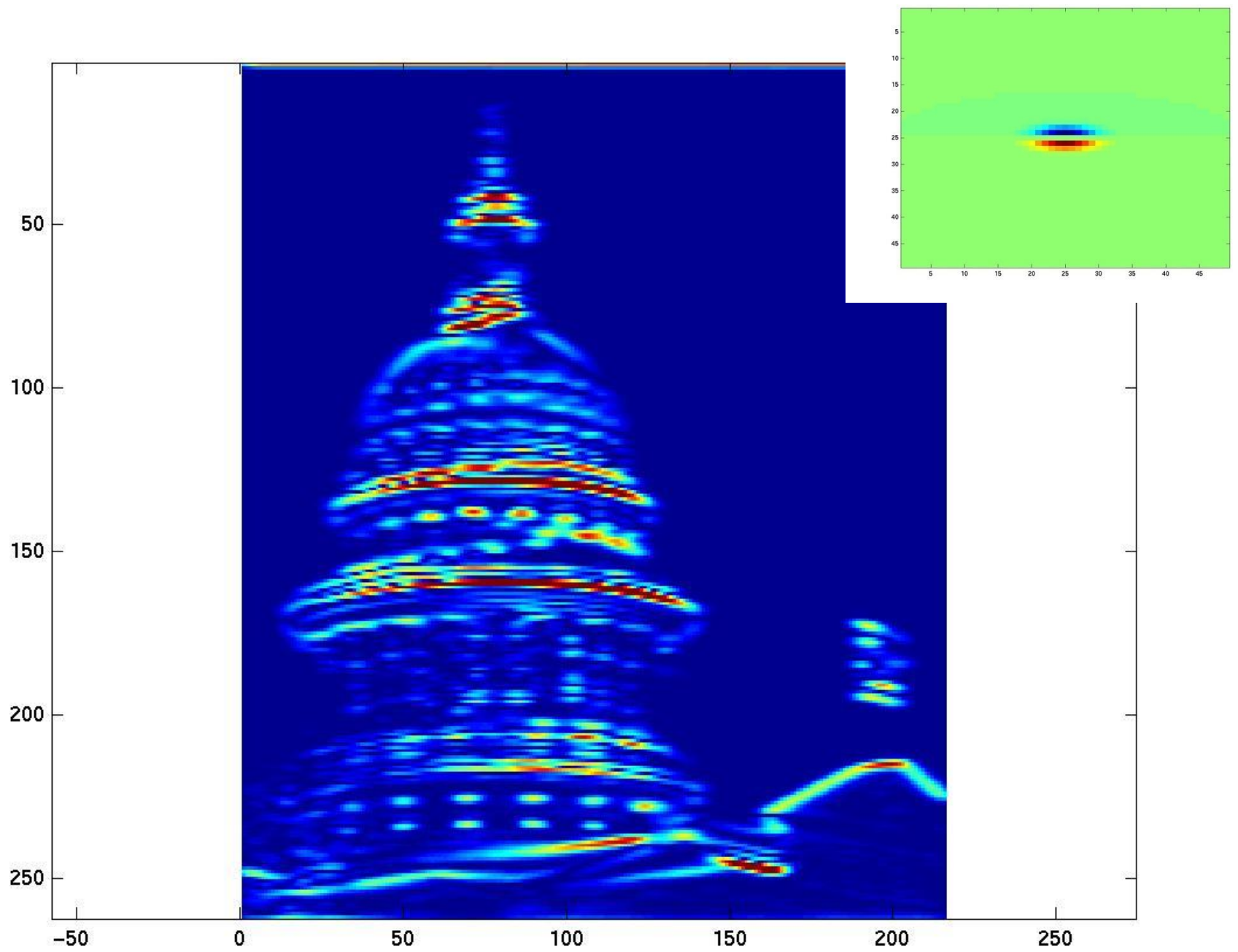
Matlab code available for these examples:

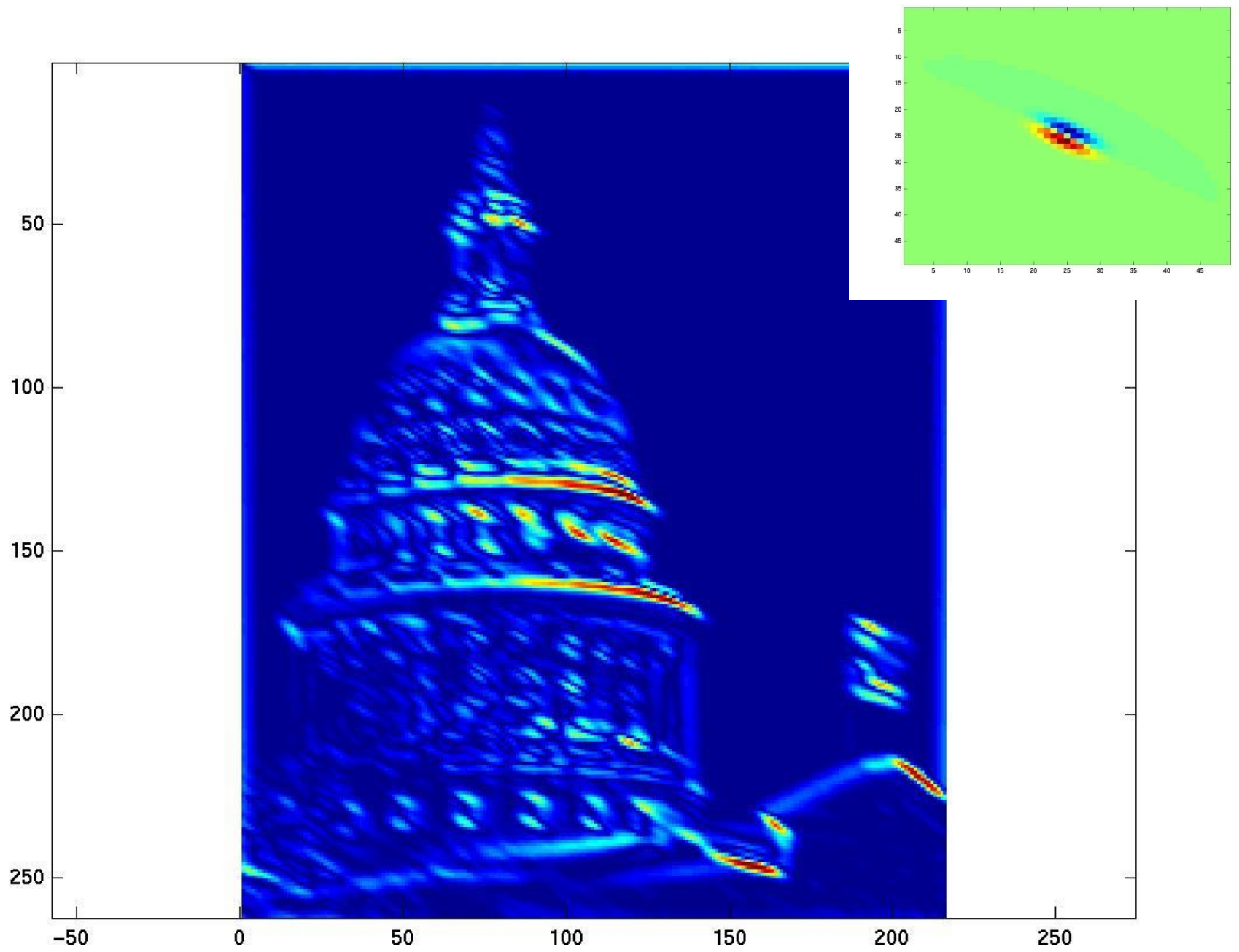
<http://www.robots.ox.ac.uk/~vgg/research/texclass/filters.html>

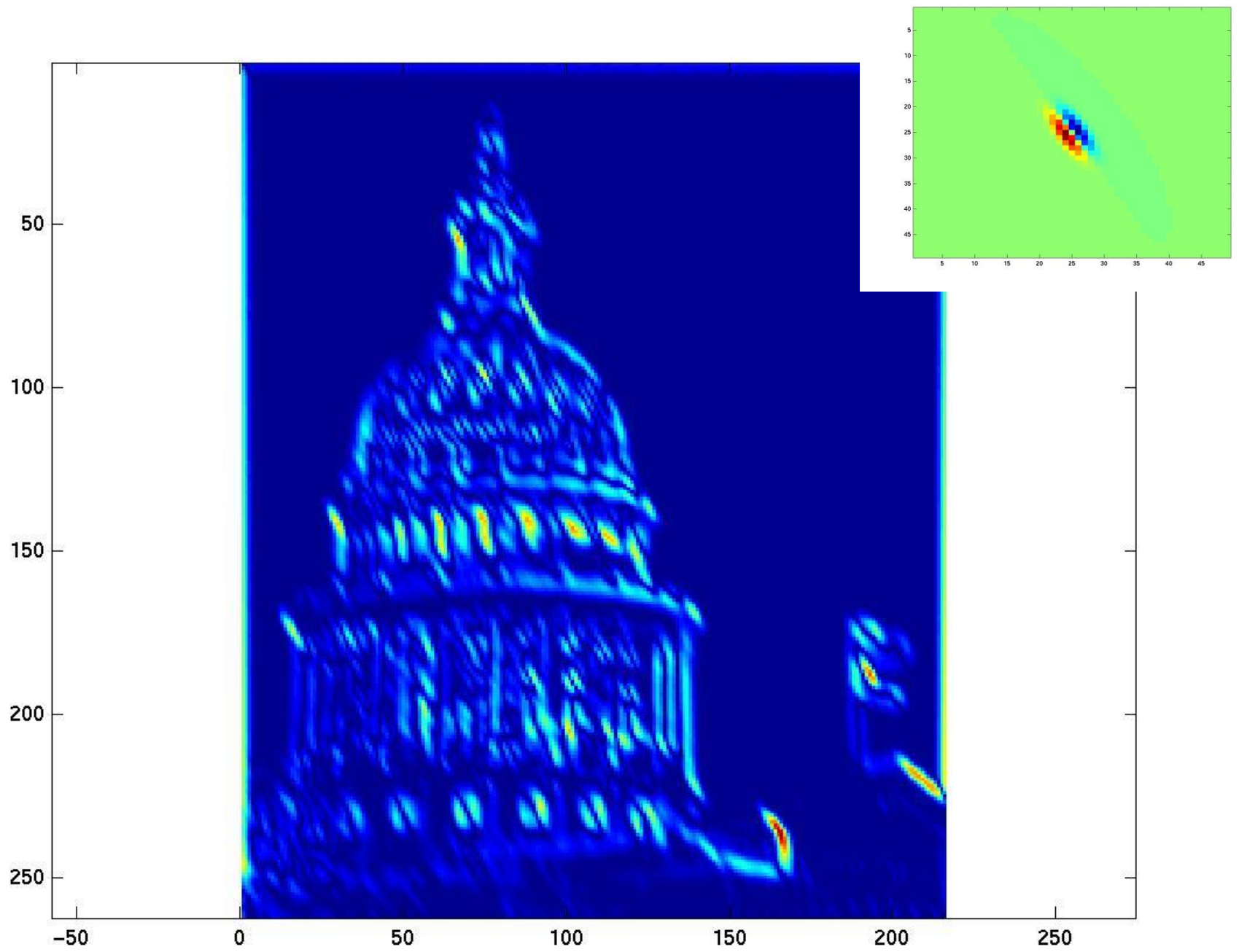
Filter bank

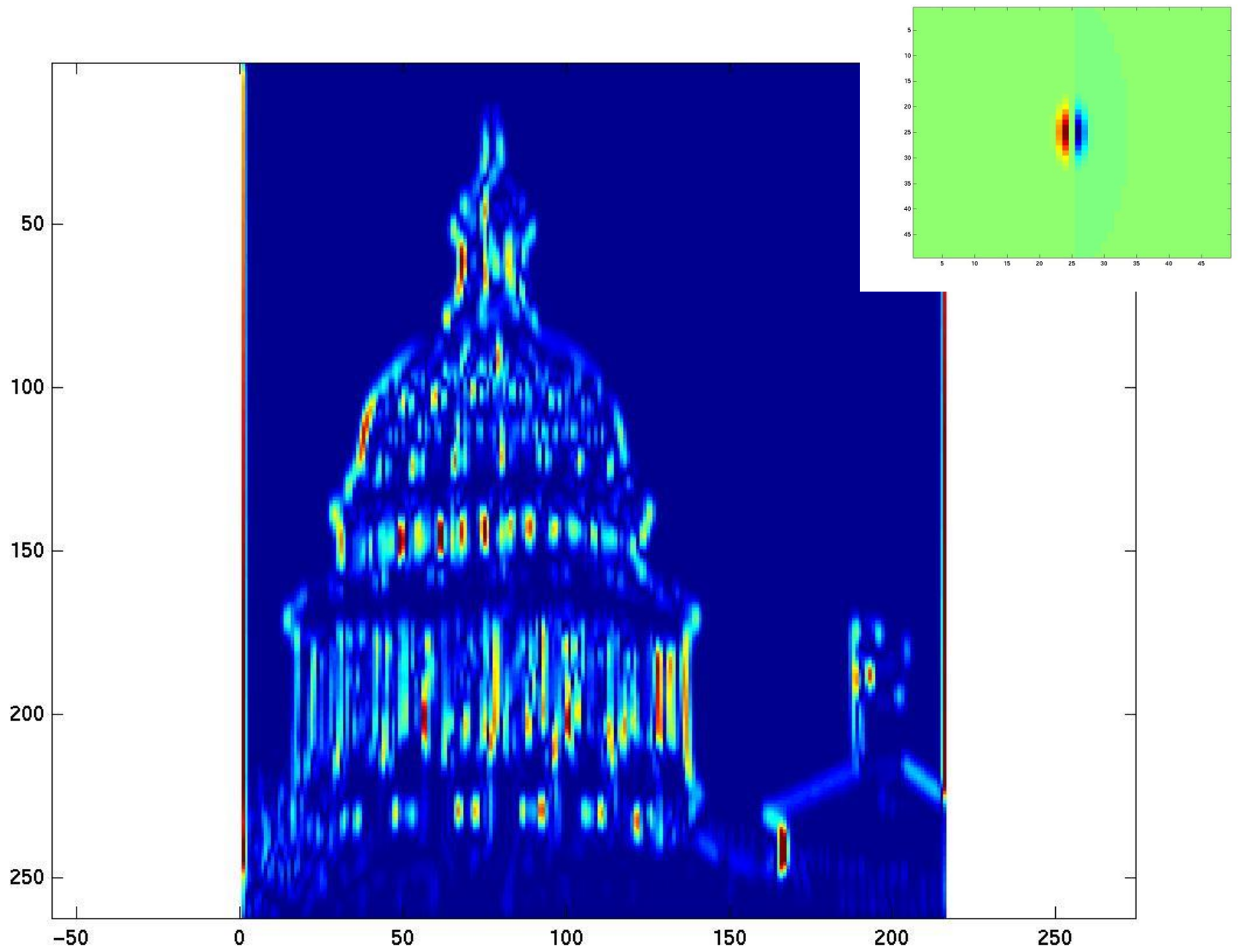


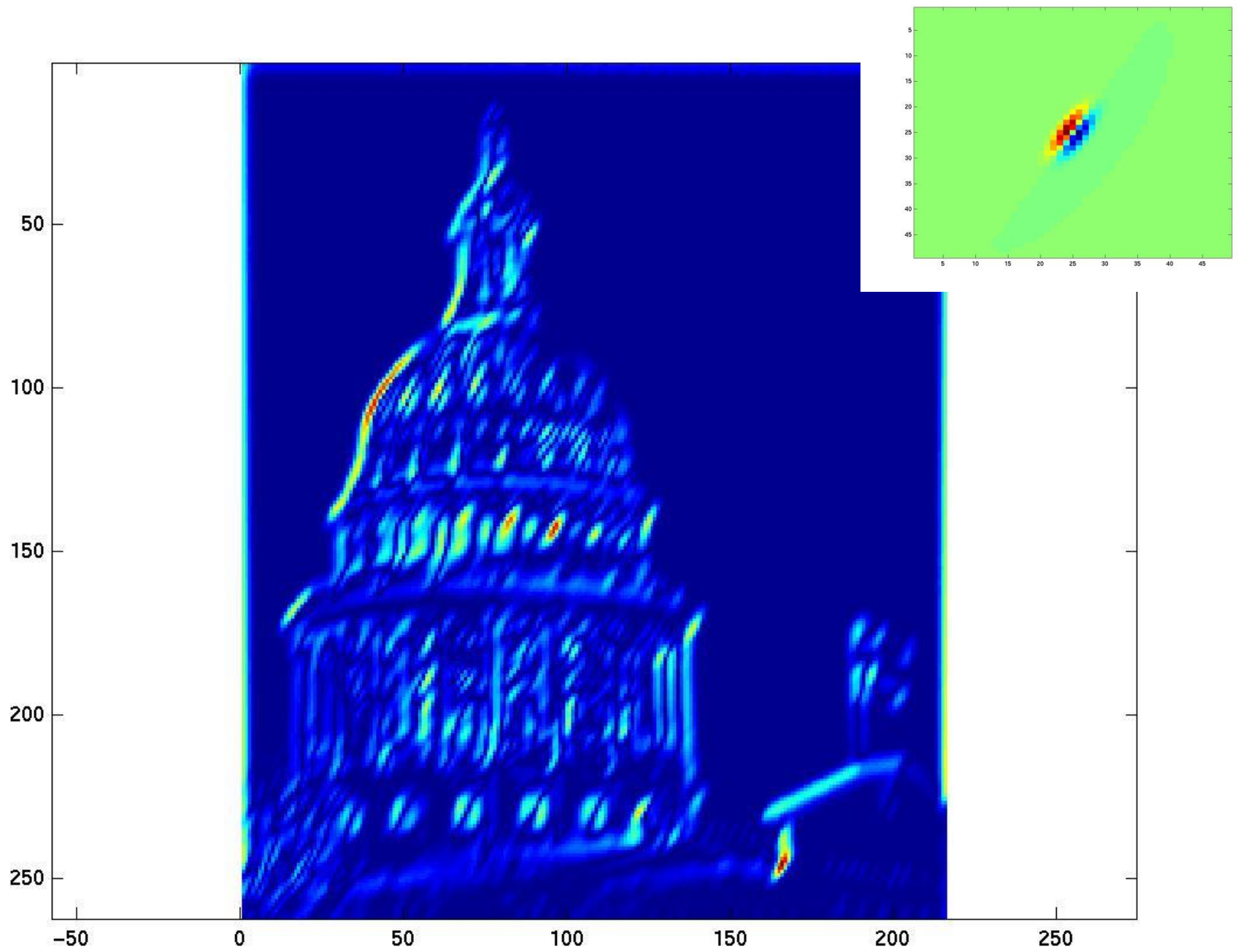


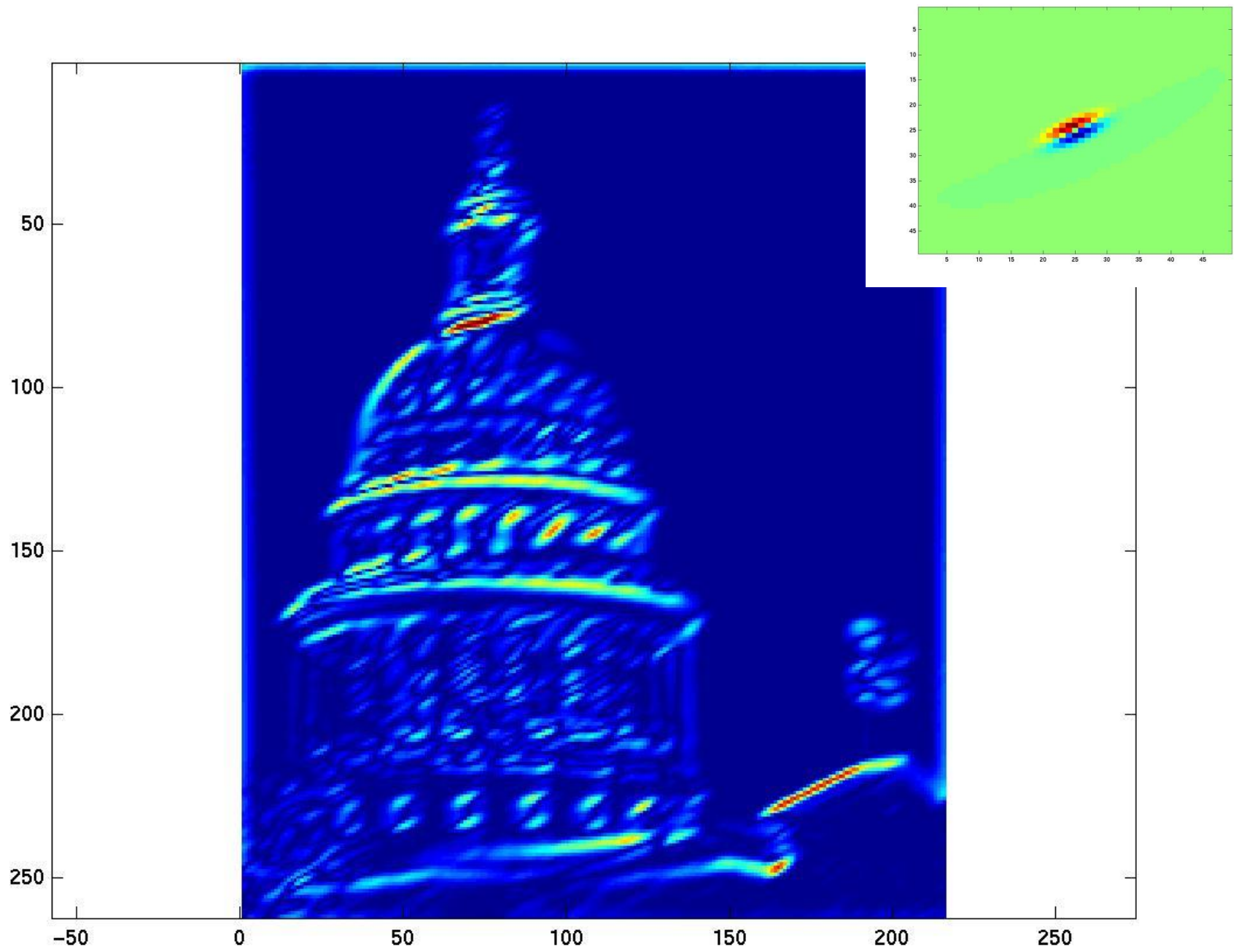


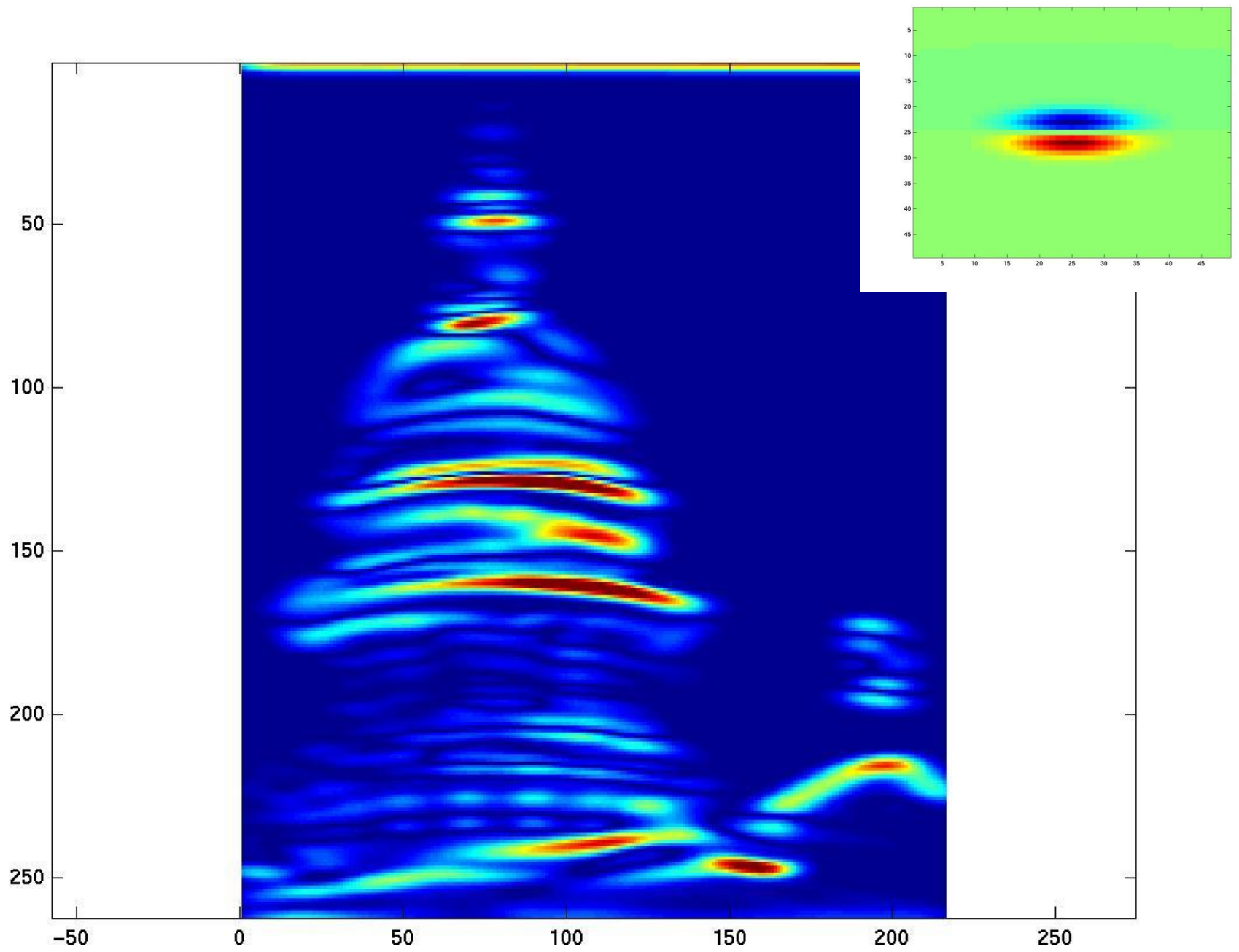


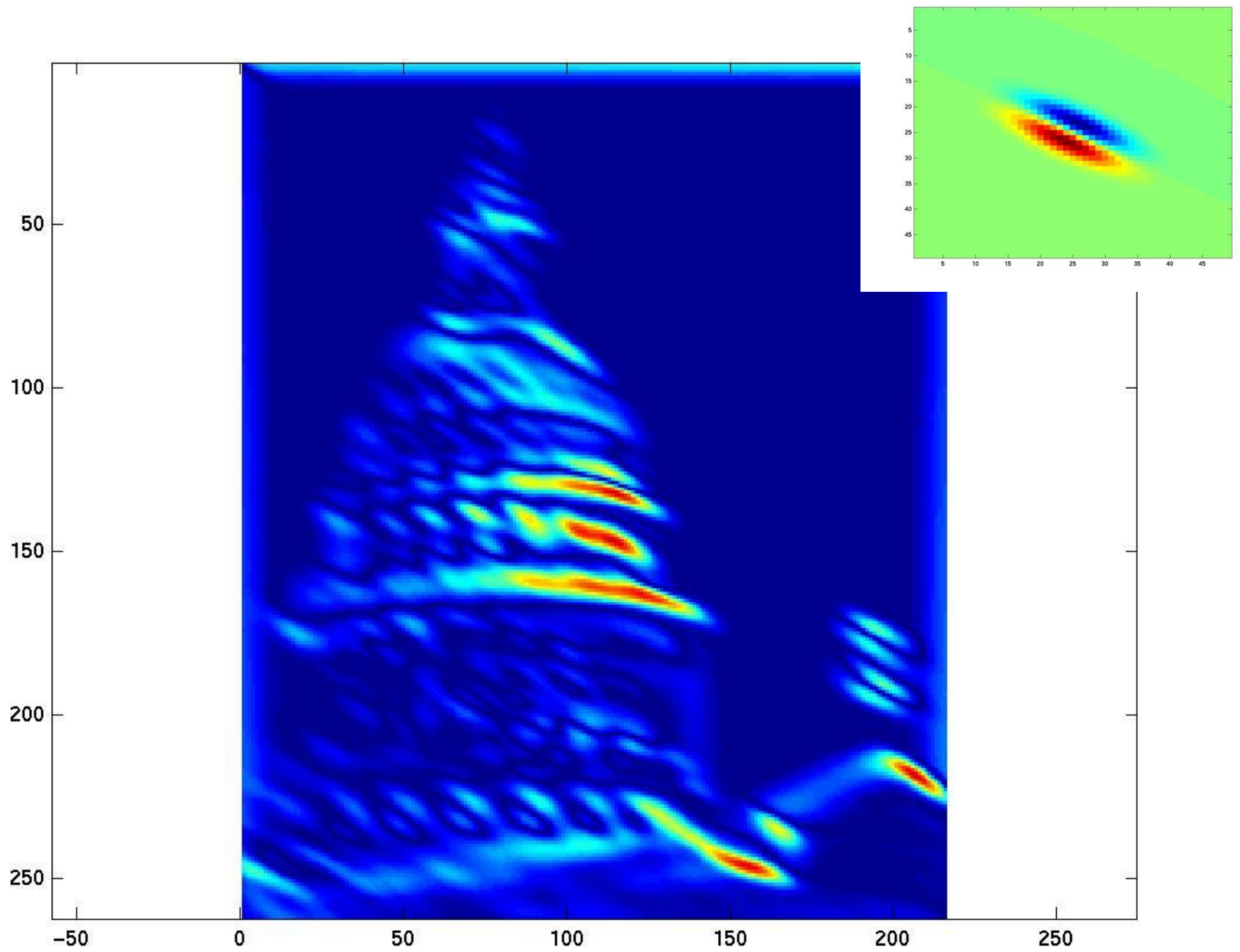


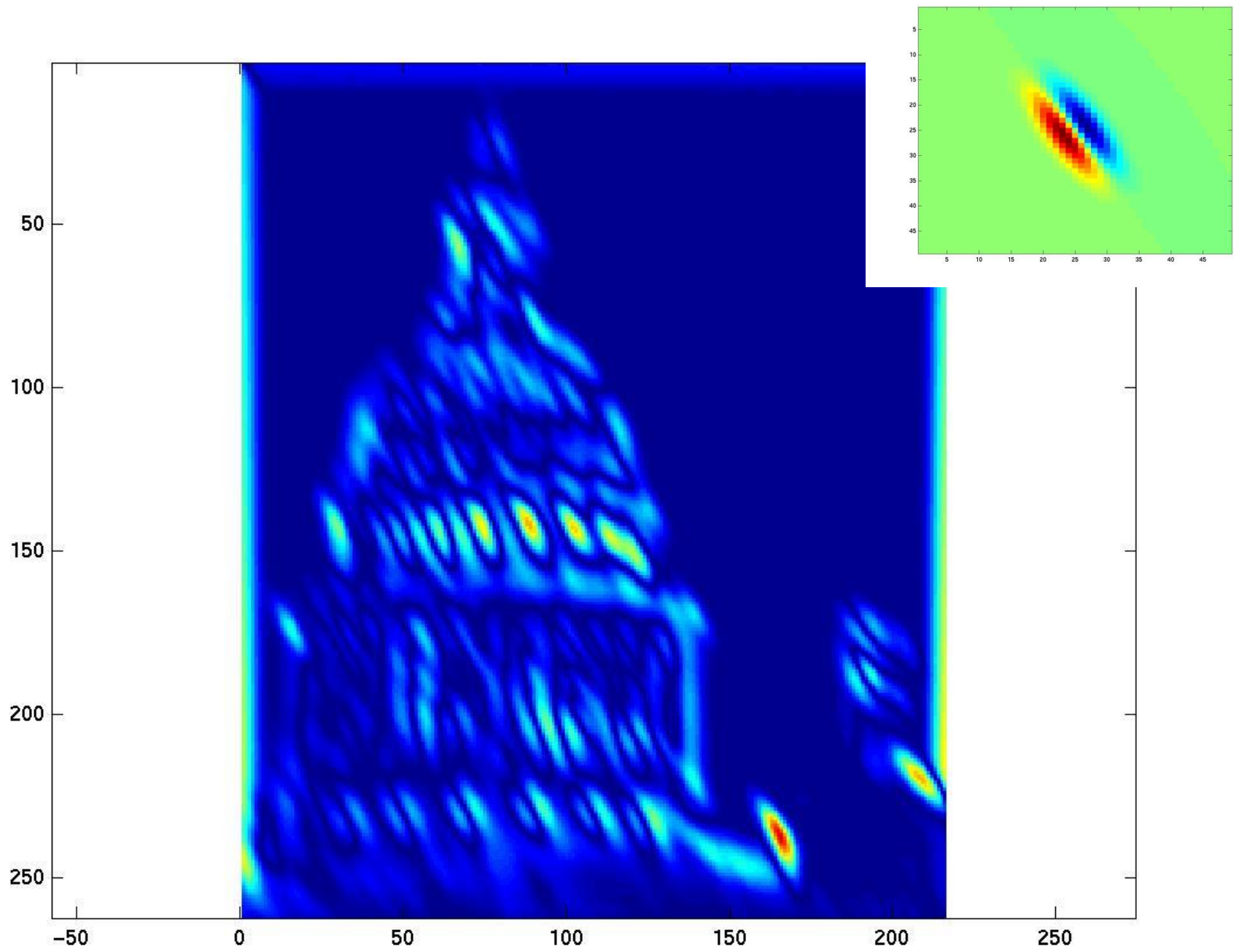


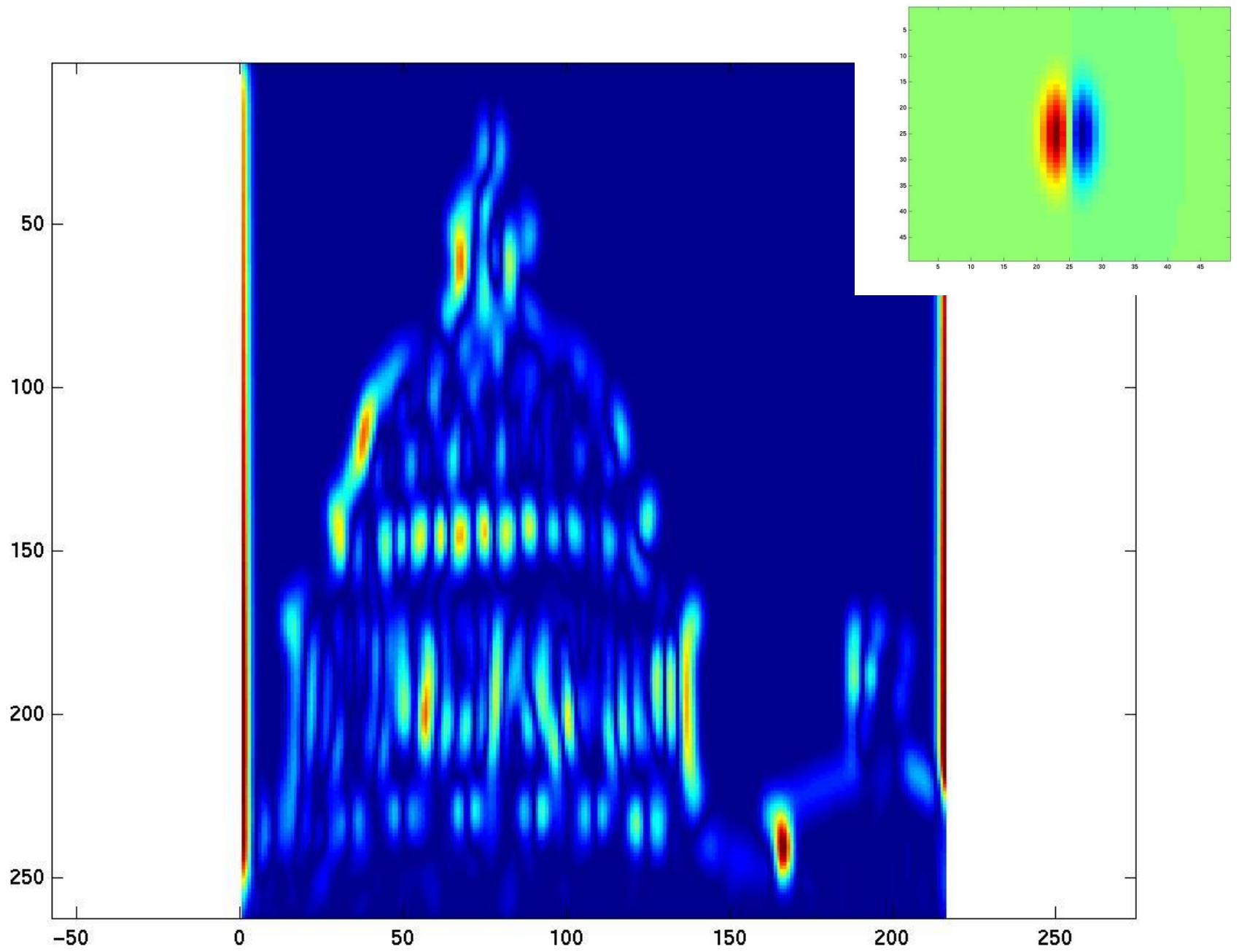


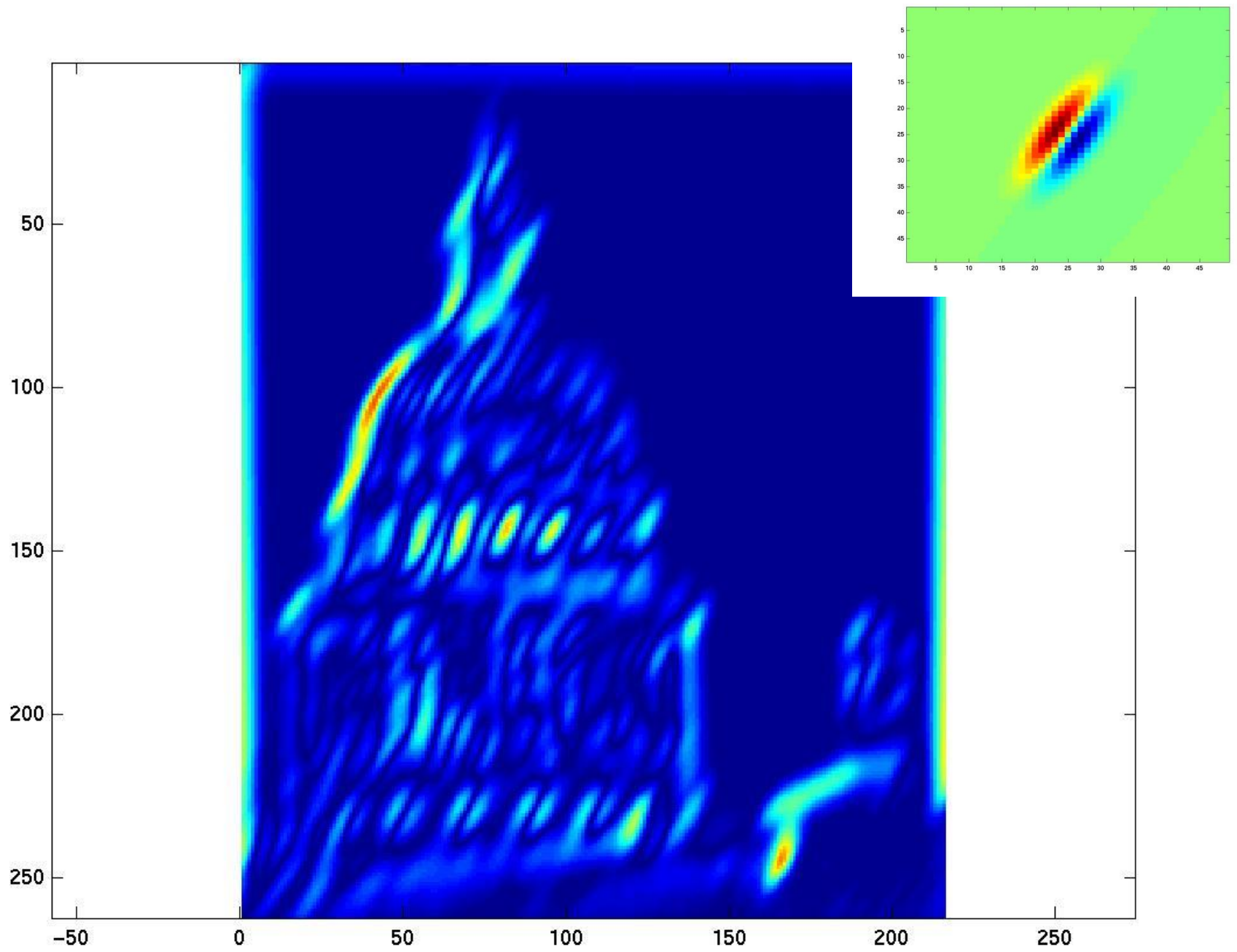


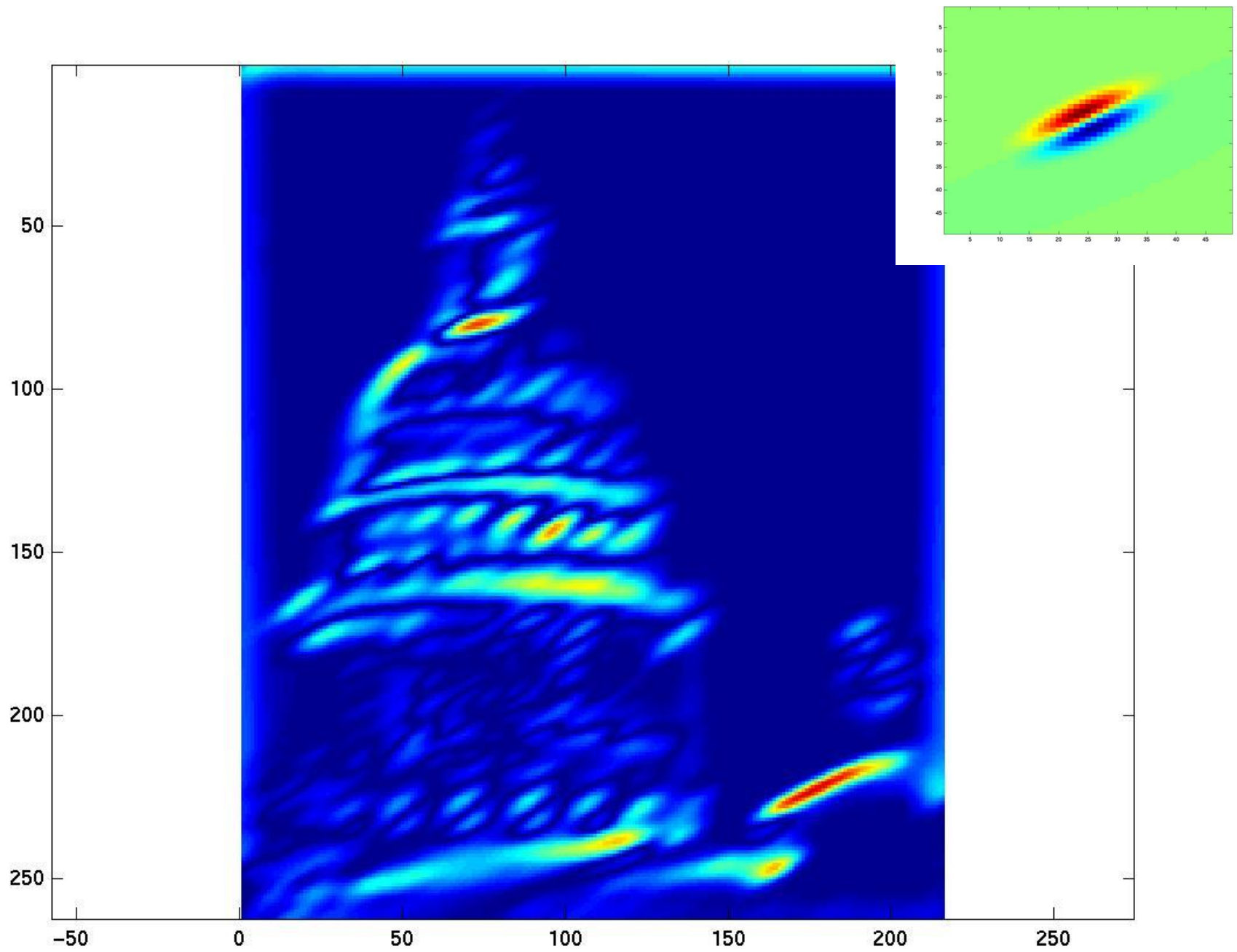


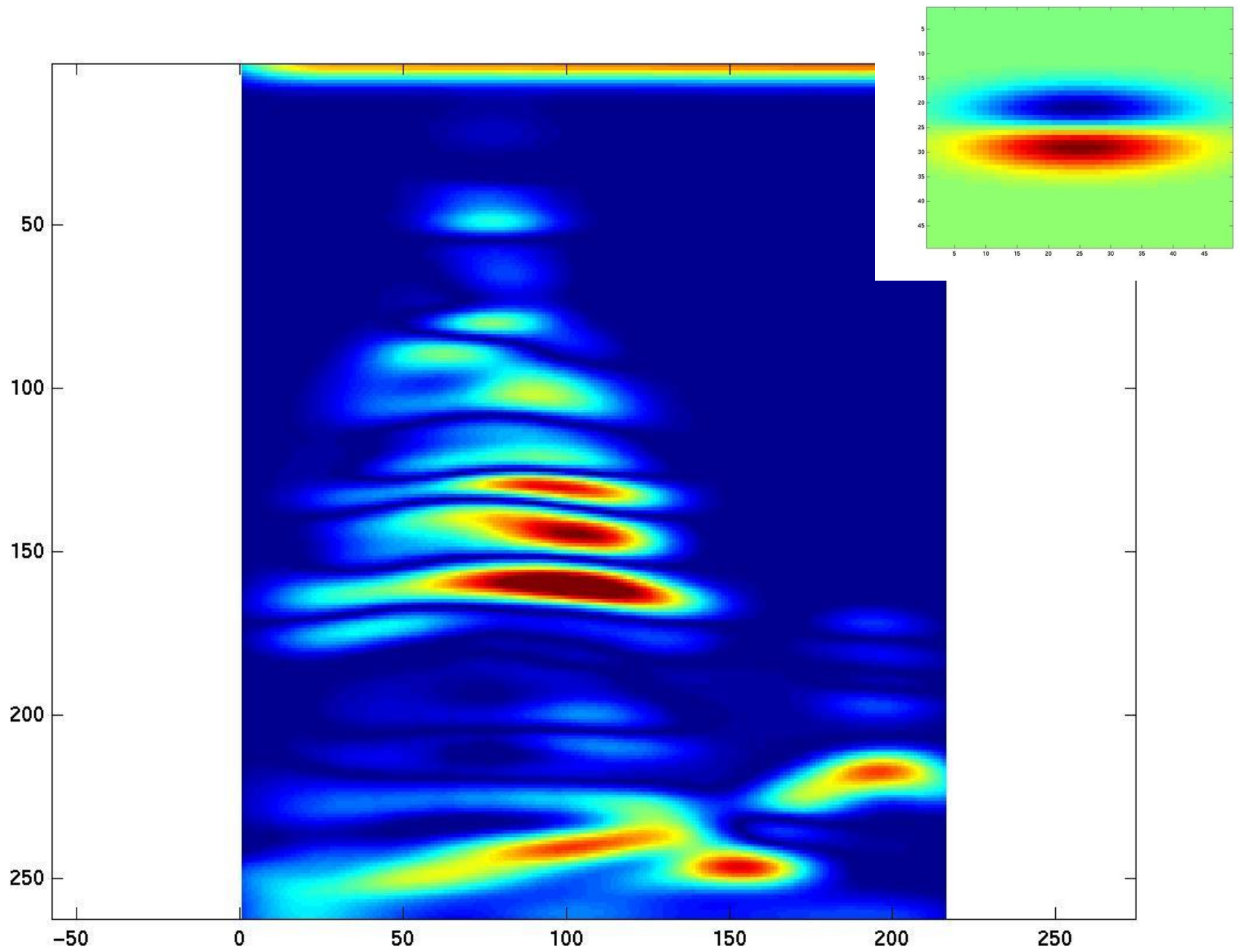


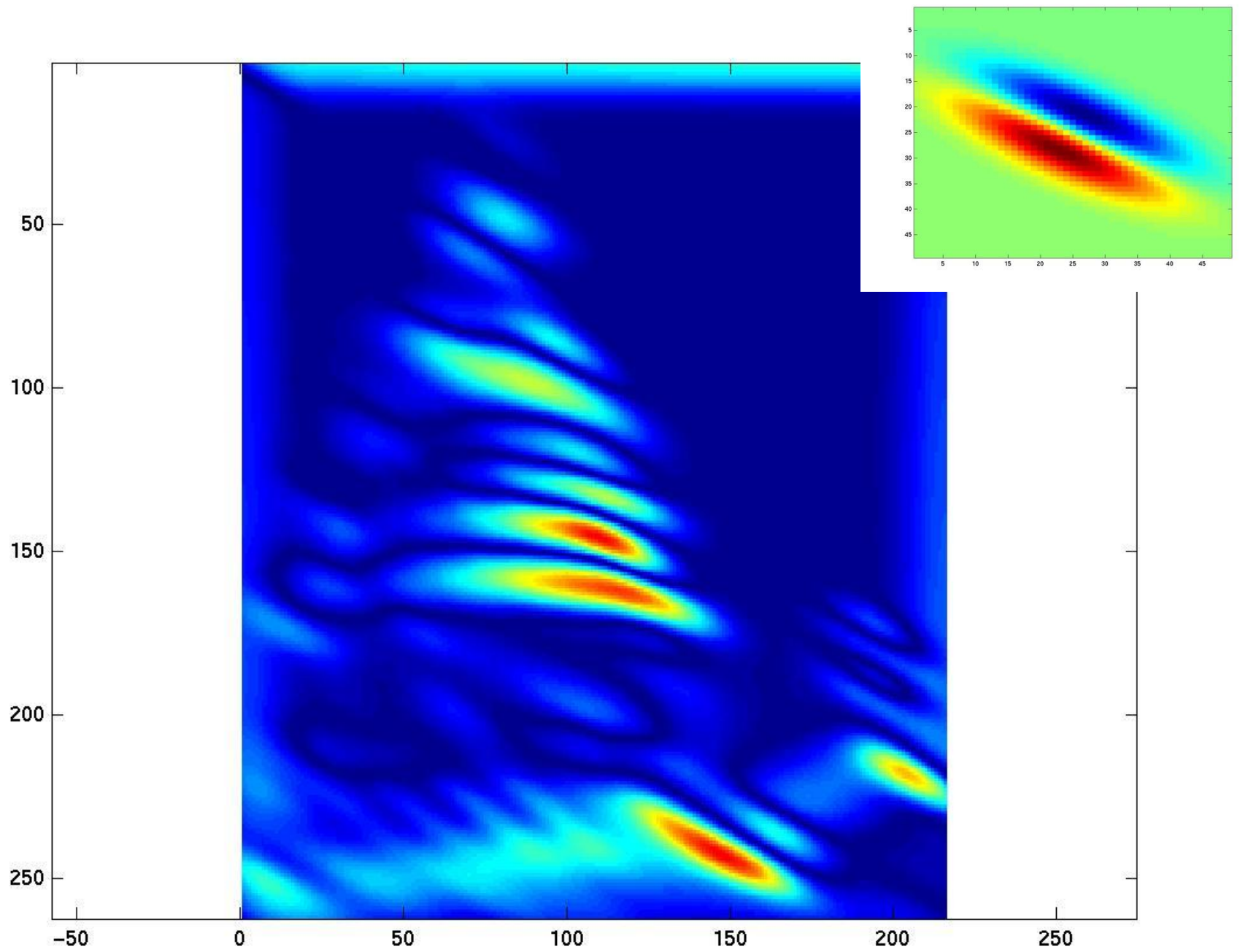


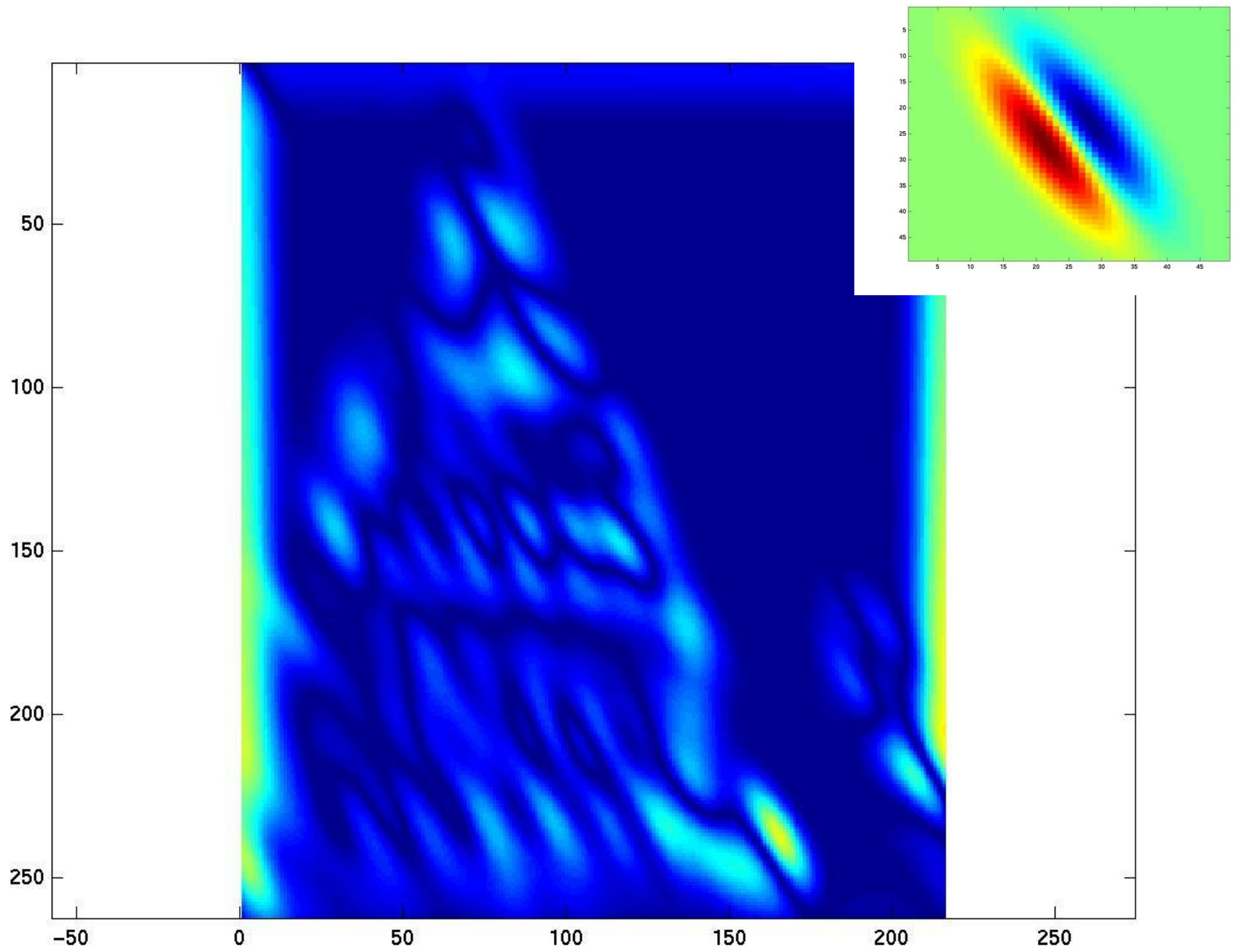


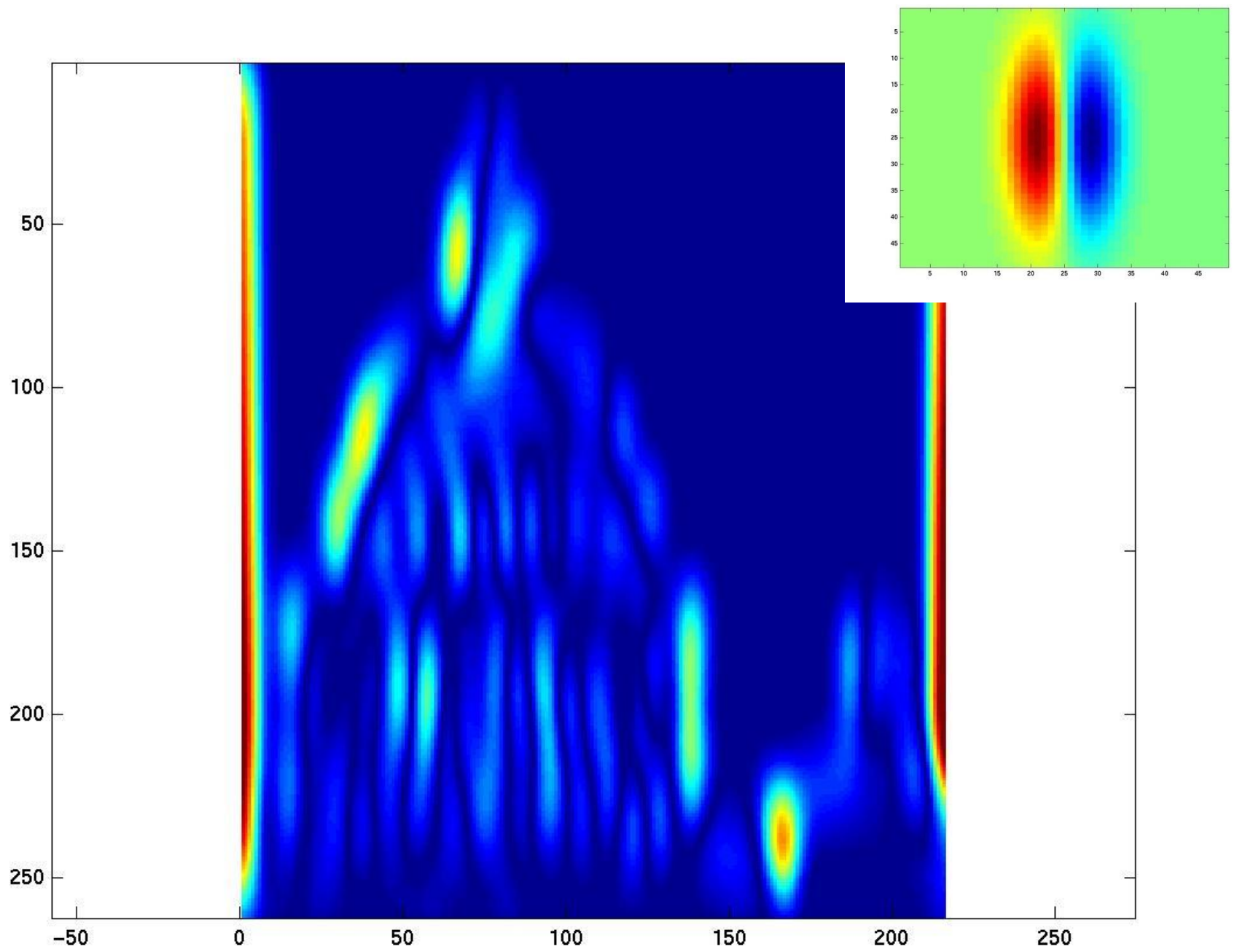


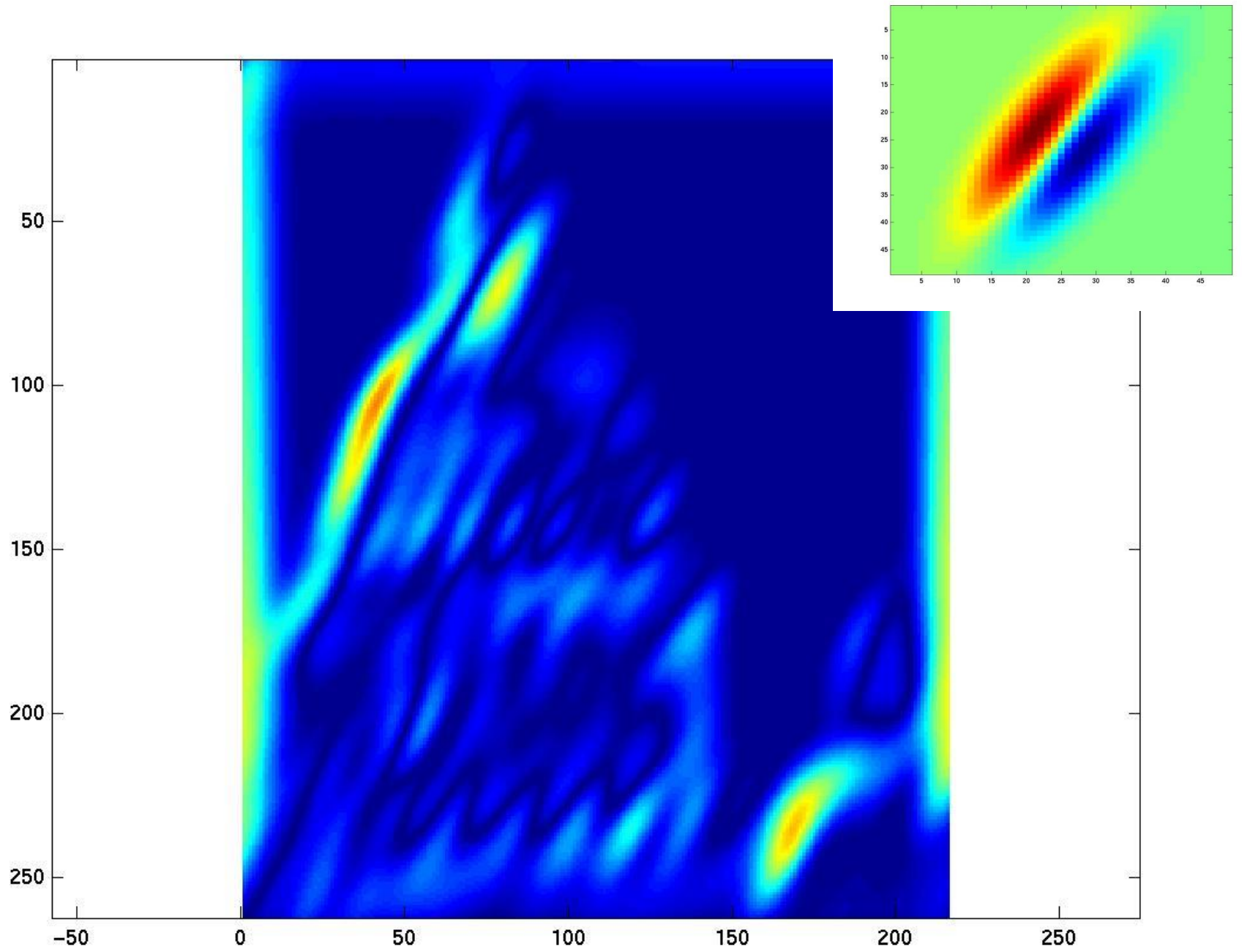


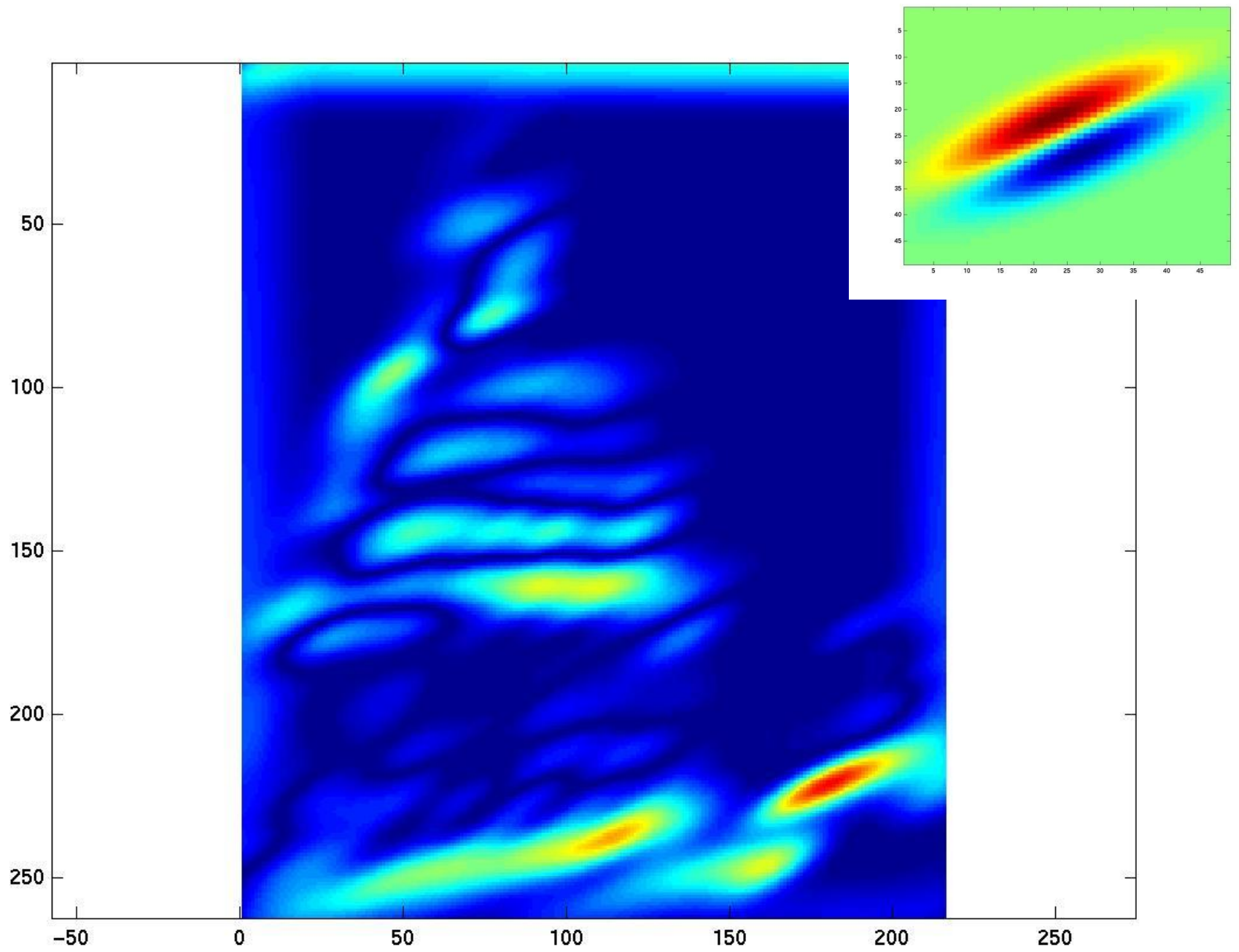


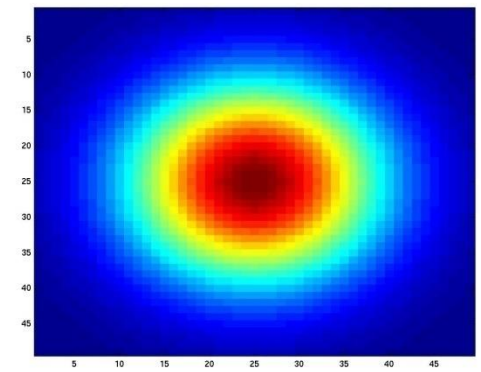
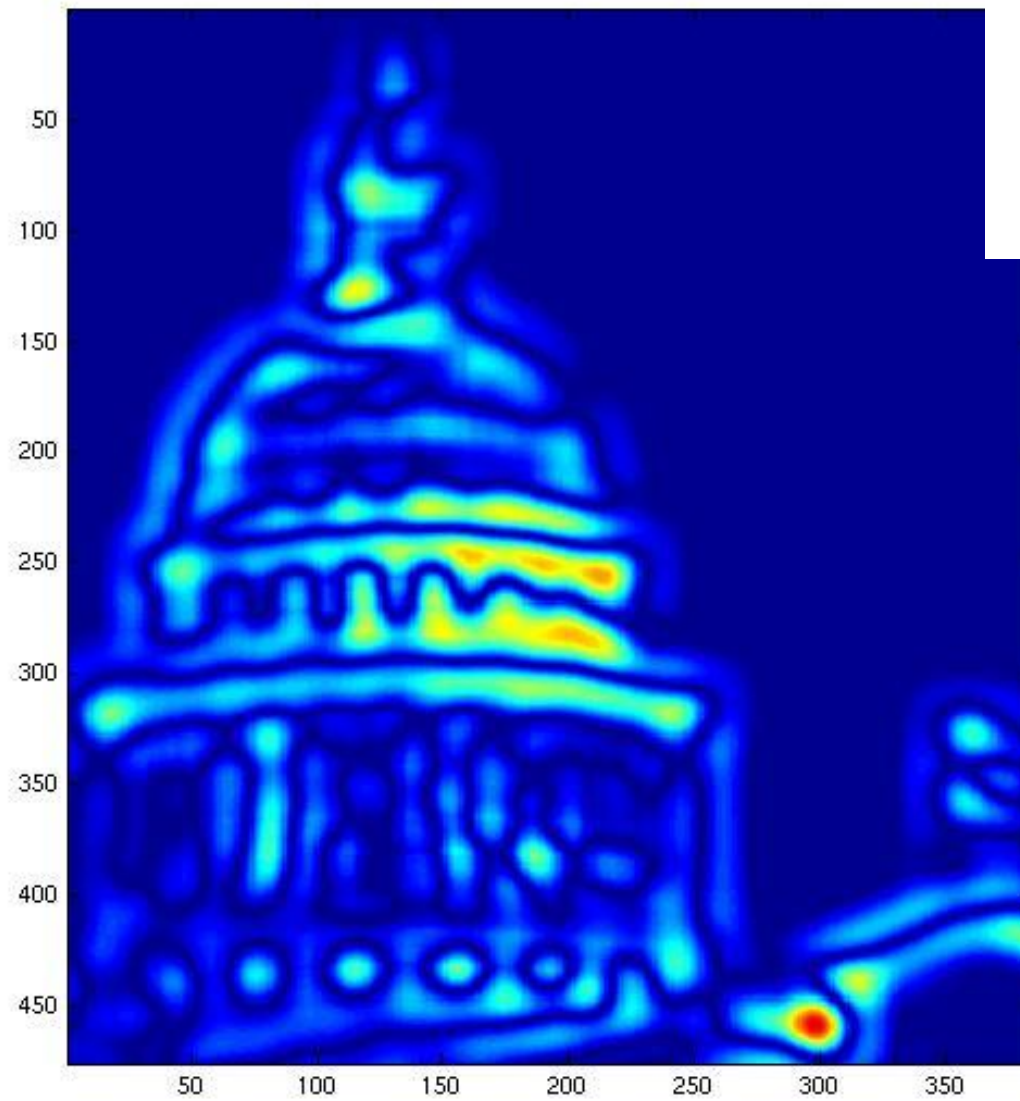




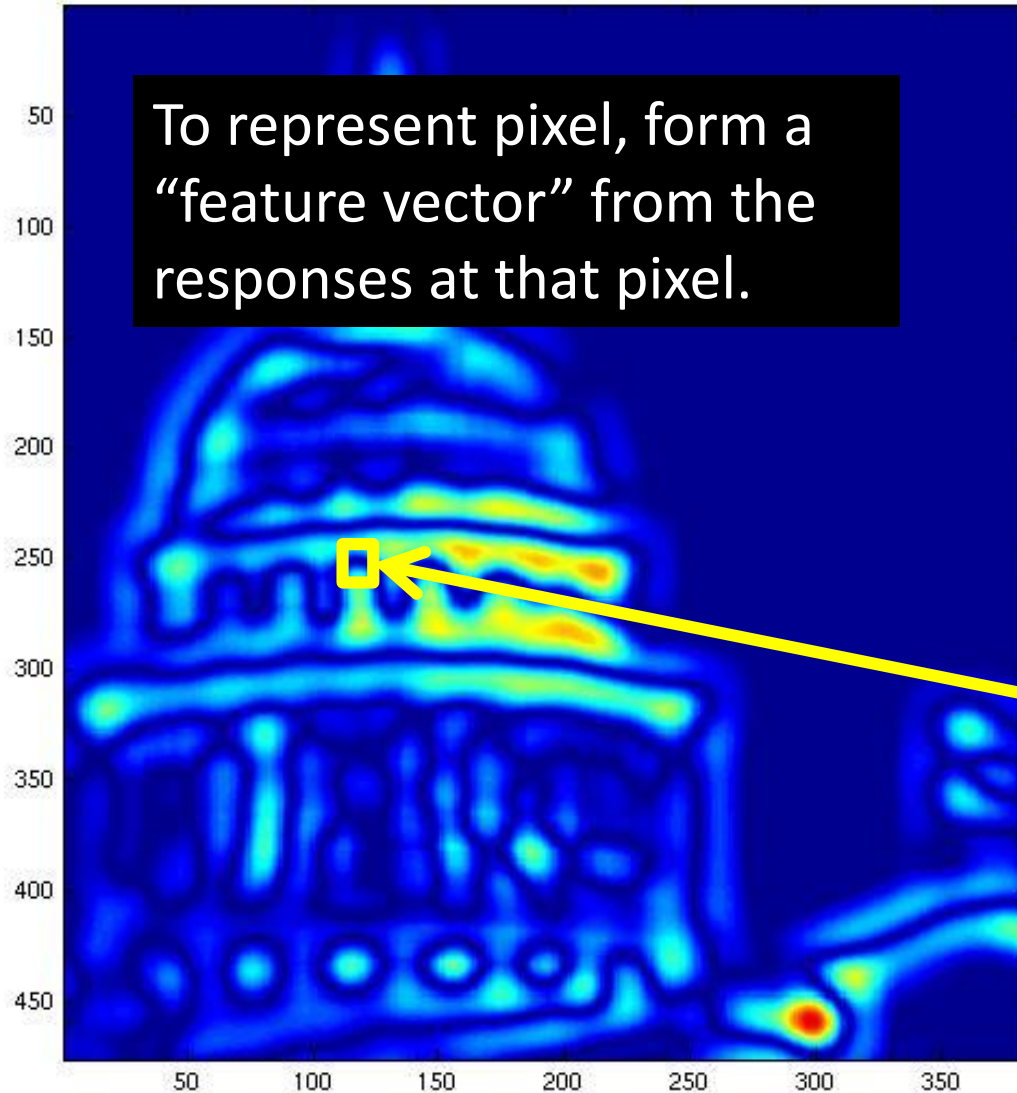




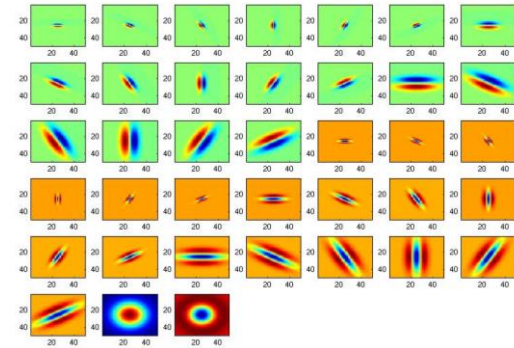




Vectors of texture responses

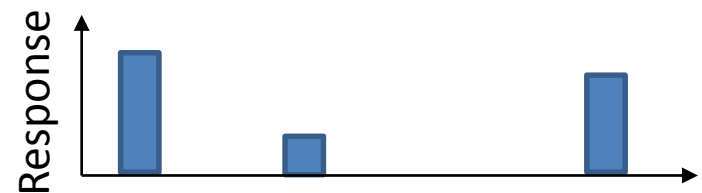


38 filters



1x38 vector
representation
feature

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



Vectors of texture responses

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

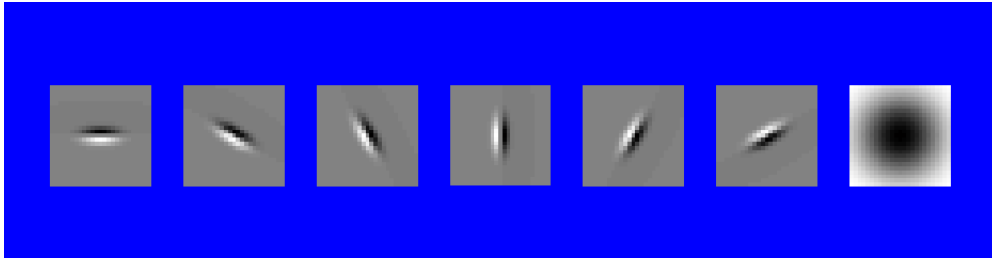
To represent *image*, compute statistics over all pixel feature vectors, e.g. their mean.

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

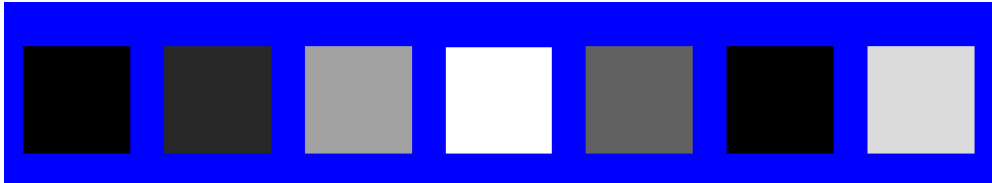
$$[\text{mean}(r_{(:,1)}^1), \text{mean}(r_{(:,2)}^2), \dots, \text{mean}(r_{(:,38)}^{38})]$$

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

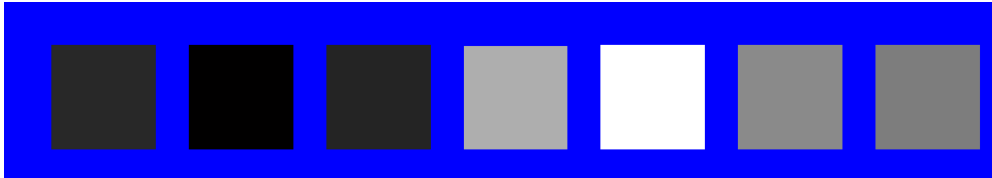
Filters



1



2

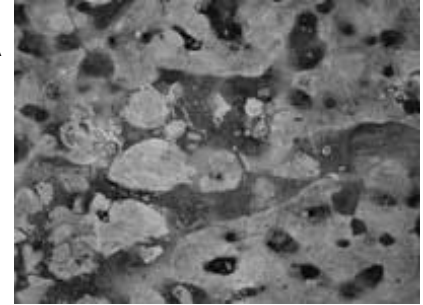


3



Mean abs responses

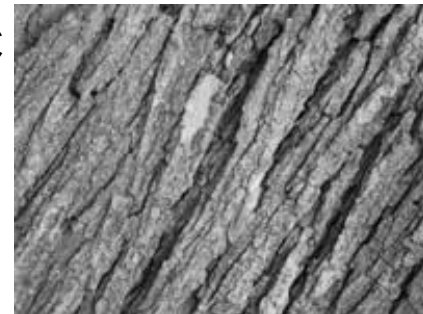
A



B

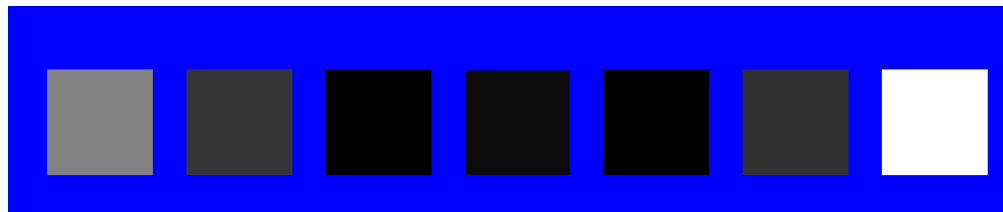
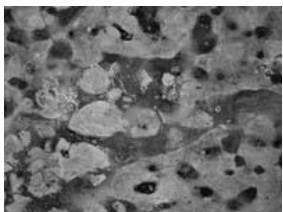
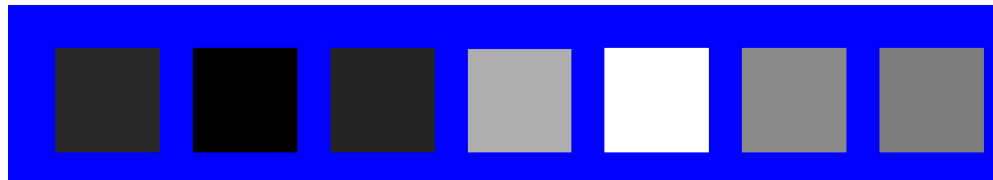
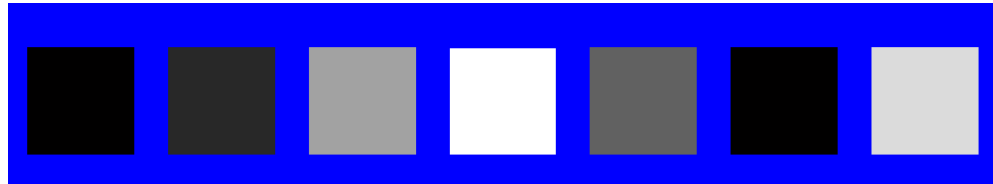
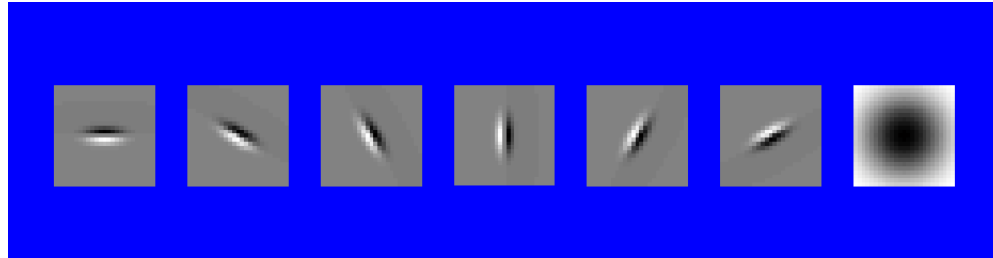


C



Representing texture by mean abs response

Filters



Mean abs responses

Classifying materials, “stuff”



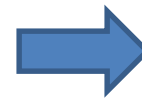
Figure by Varma & Zisserman

Plan for next two lectures

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

Sampling

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



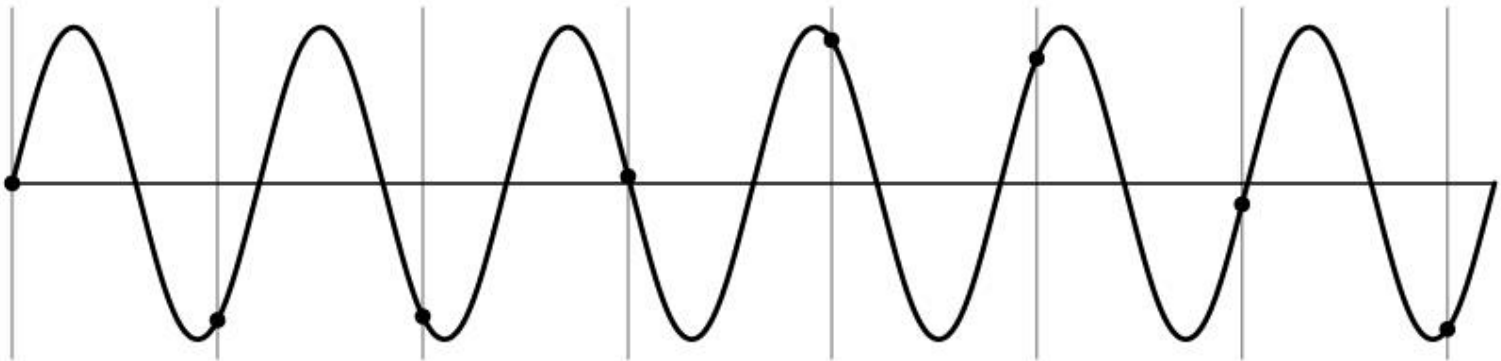
Subsampling by a factor of 2



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

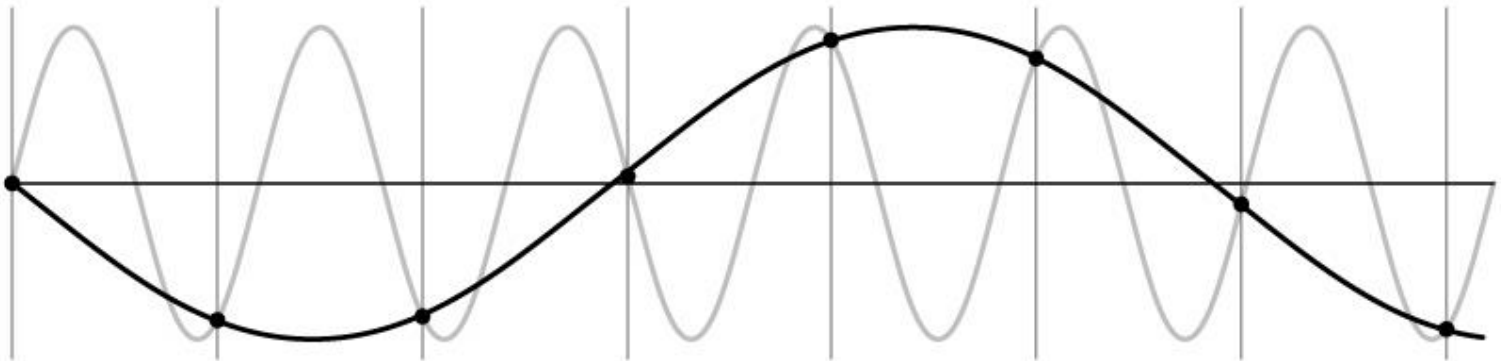
Aliasing problem

- 1D example (sinewave):



Aliasing problem

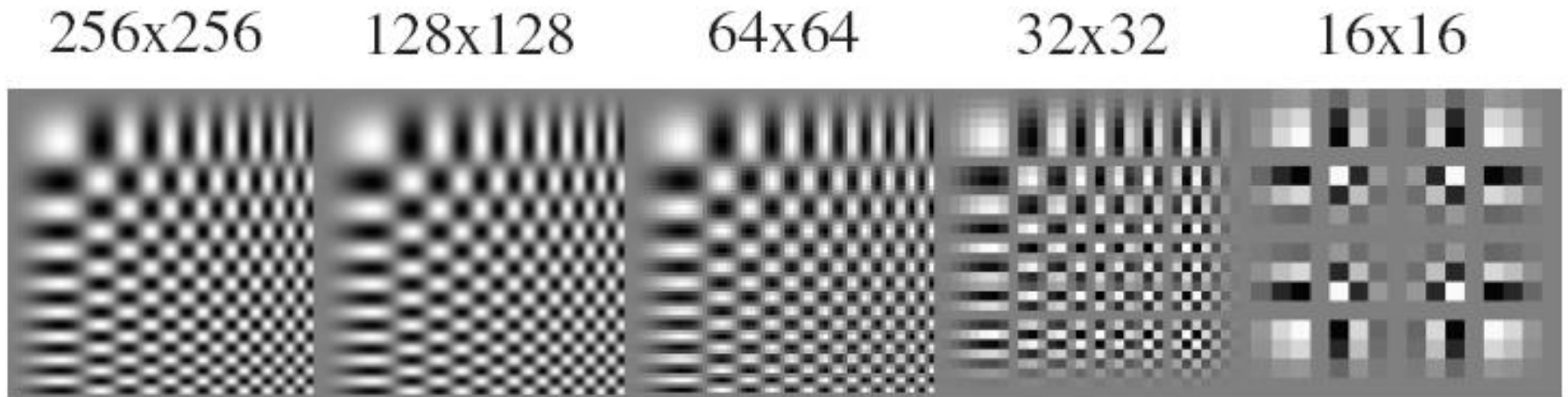
- 1D example (sinewave):



Aliasing problem

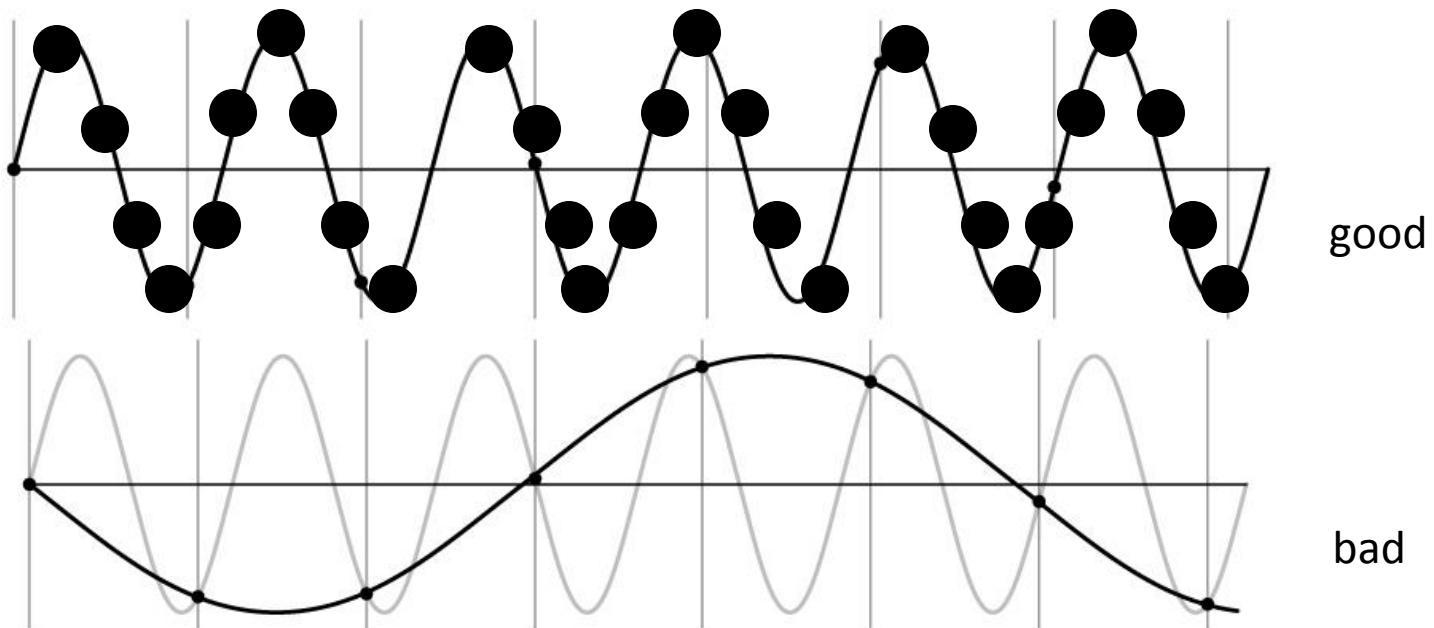
- Sub-sampling may be dangerous....
- Characteristic errors may appear:
 - “Wagon wheels rolling the wrong way in movies”
 - “Striped shirts look funny on color television”

Sampling and aliasing



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 will allow to reconstruct the original perfectly from the sampled version



Anti-aliasing

Solutions:

- Sample more often
- Get rid of high frequencies
 - What are these in the case of images?
 - Will lose information, but it's better than aliasing
 - Apply a smoothing filter

Algorithm for downsampling by factor of 2

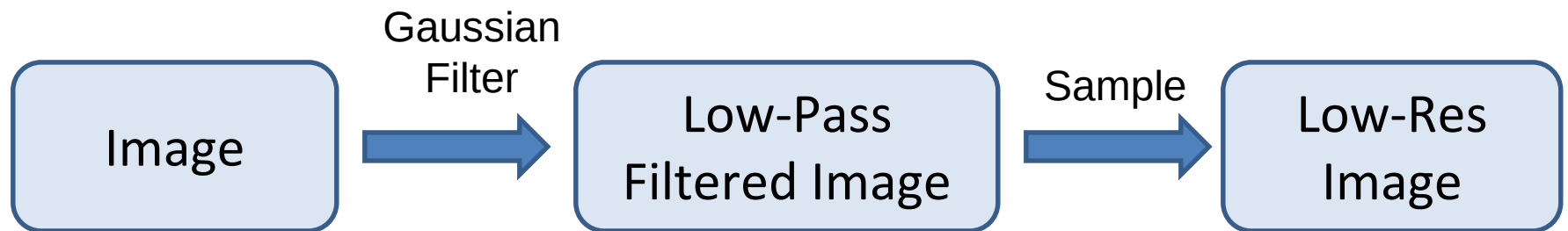
1. Start with image(h, w)

2. Apply low-pass filter

```
im_blur = imfilter(image, fspecial('gaussian', 7, 1))
```

3. Sample every other pixel

```
im_small = im_blur(1:2:end, 1:2:end);
```



Anti-aliasing

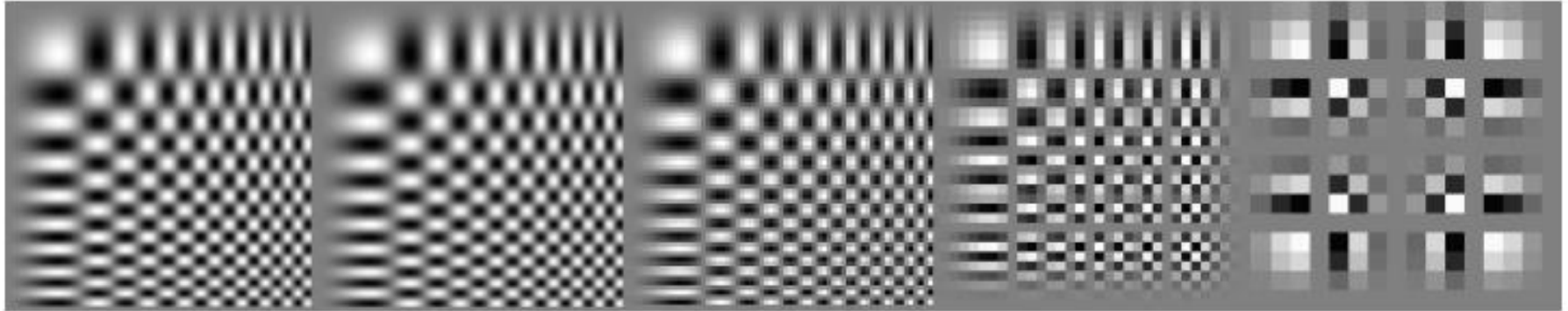
256x256

128x128

64x64

32x32

16x16



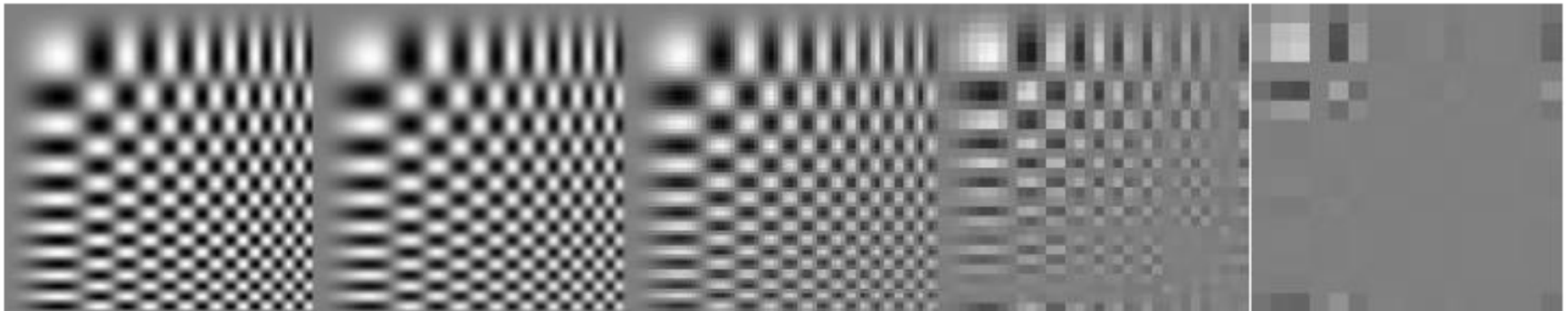
256x256

128x128

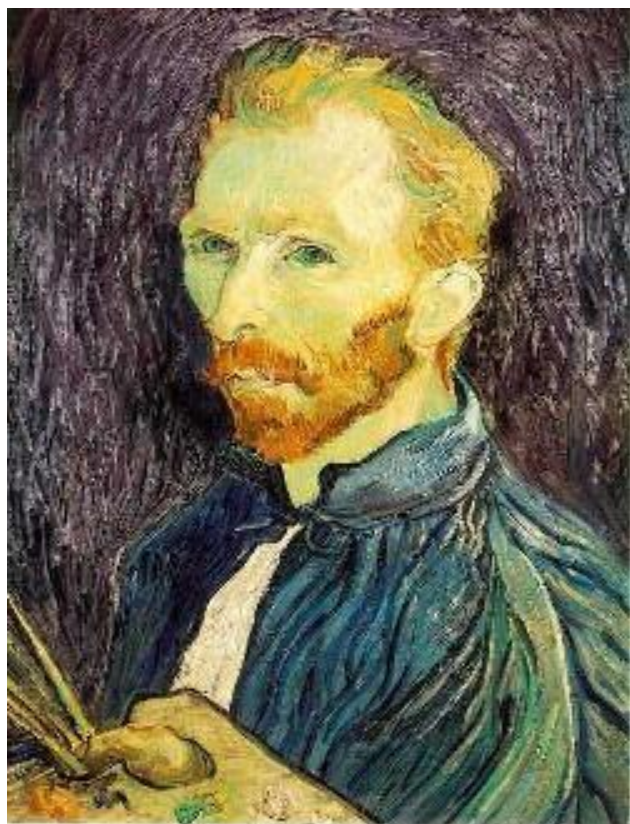
64x64

32x32

16x16



Subsampling without pre-filtering



$1/2$

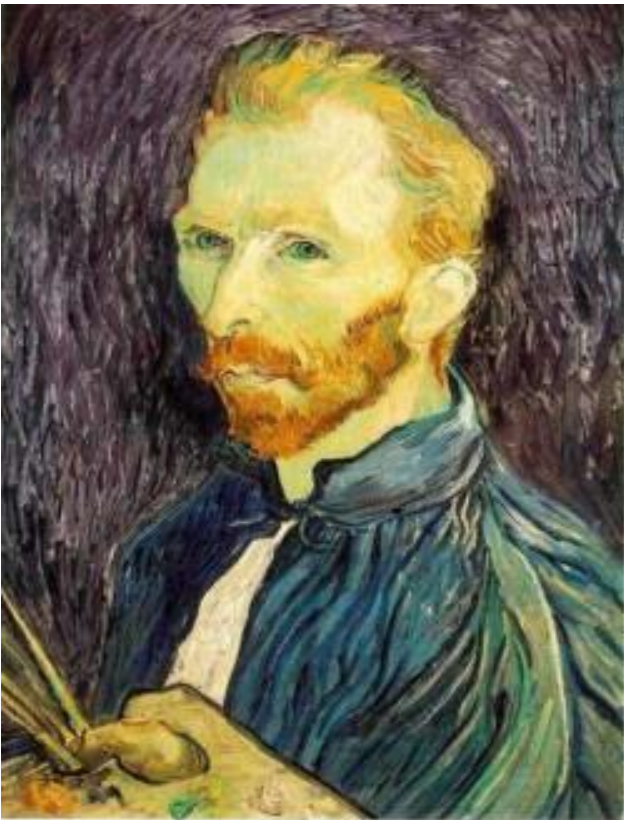


$1/4$ (2x zoom)



$1/8$ (4x zoom)

Subsampling with Gaussian pre-filtering



Gaussian $1/2$

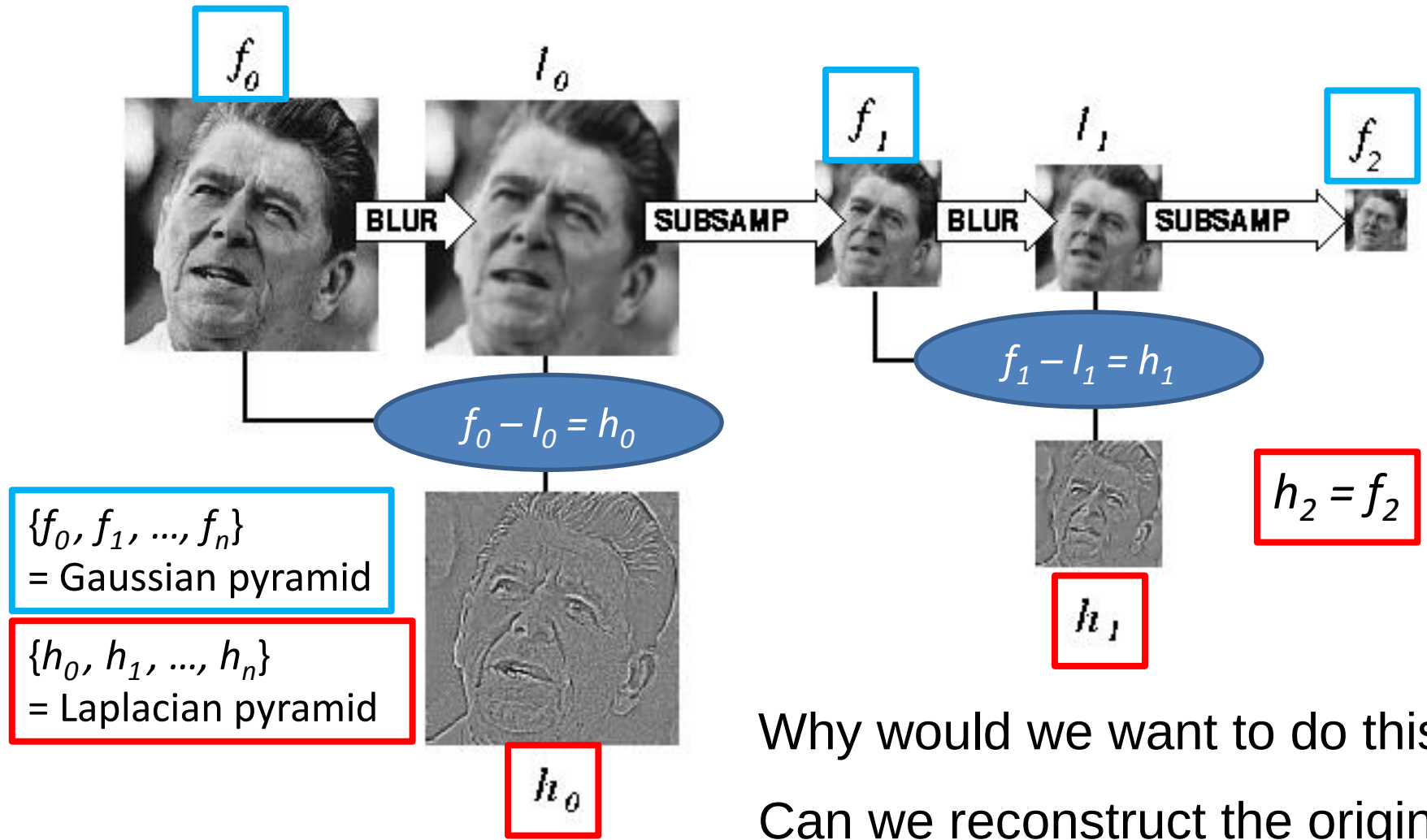


G $1/4$



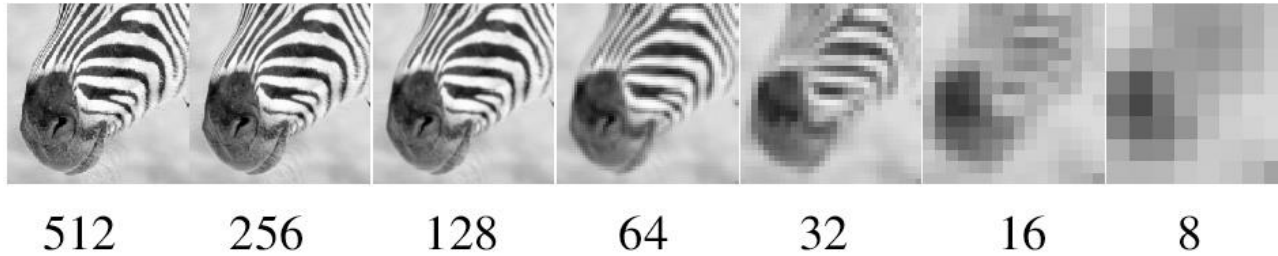
G $1/8$

Subsampling away...

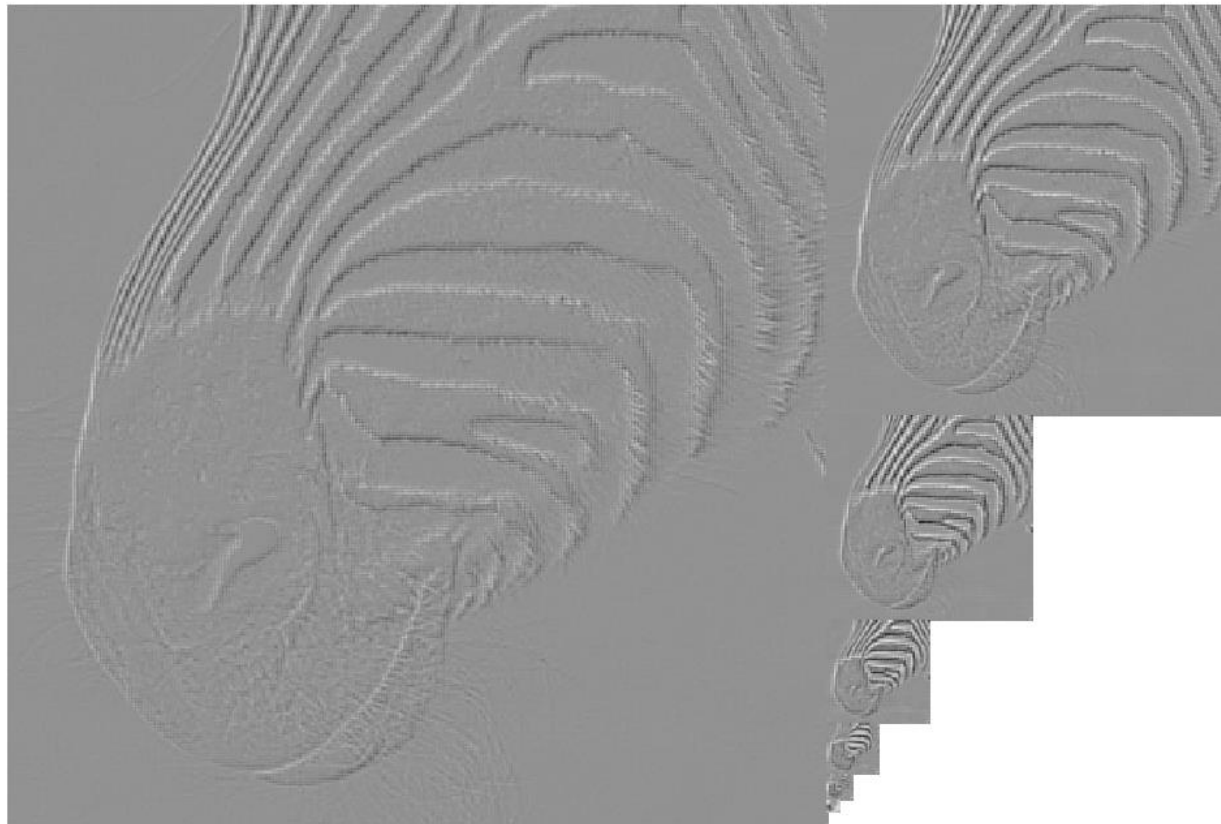
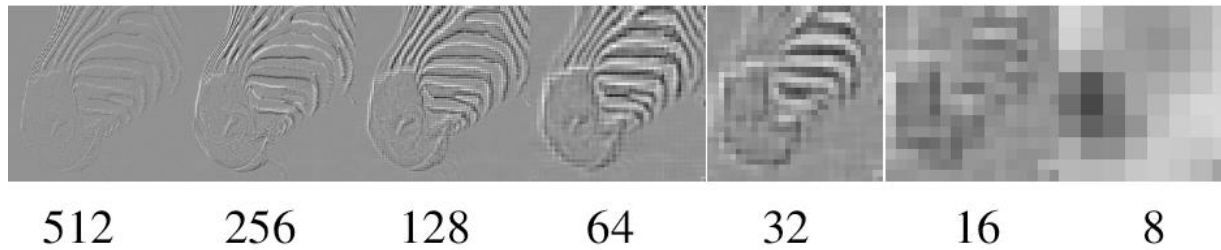


Why would we want to do this?
Can we reconstruct the original
from the Laplacian pyramid?

Gaussian pyramid



Laplacian pyramid



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
- Can use filtering to reduce the effects of subsampling