LECTURE 20

Convolutional Neural Networks (CNNs)

Building CNNsin Using Keras

Data Science, Fall 2023 @ Knowledge Stream

Sana Jabbar

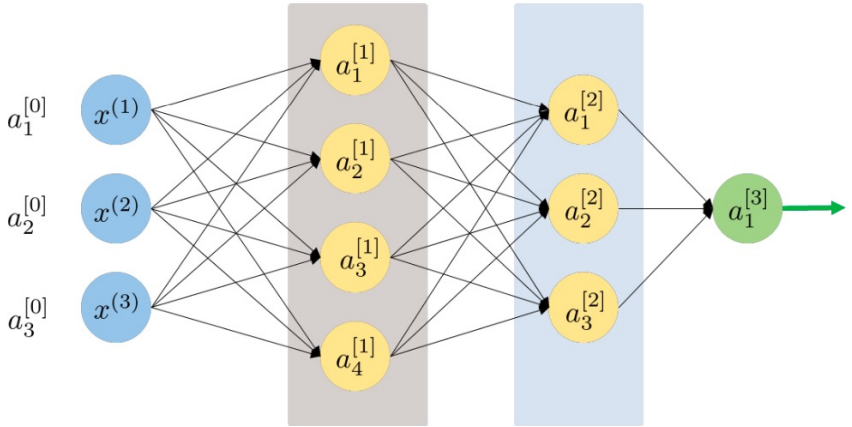
Review

Lecture 20

- Introduction to Neural Networks
- Neural network forward pass
- Activation functions
- Back Propagation

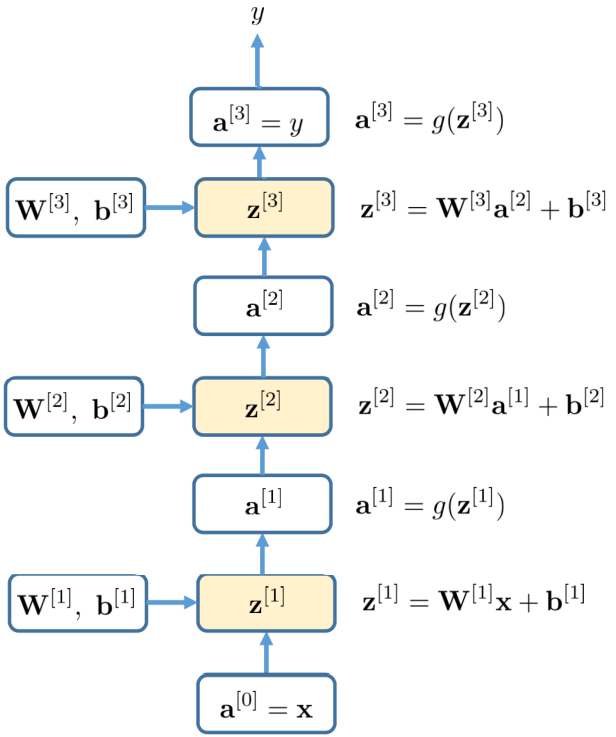
Back Propagation: Vectorization

- We compute loss function $\mathcal{L}$ using forward pass.



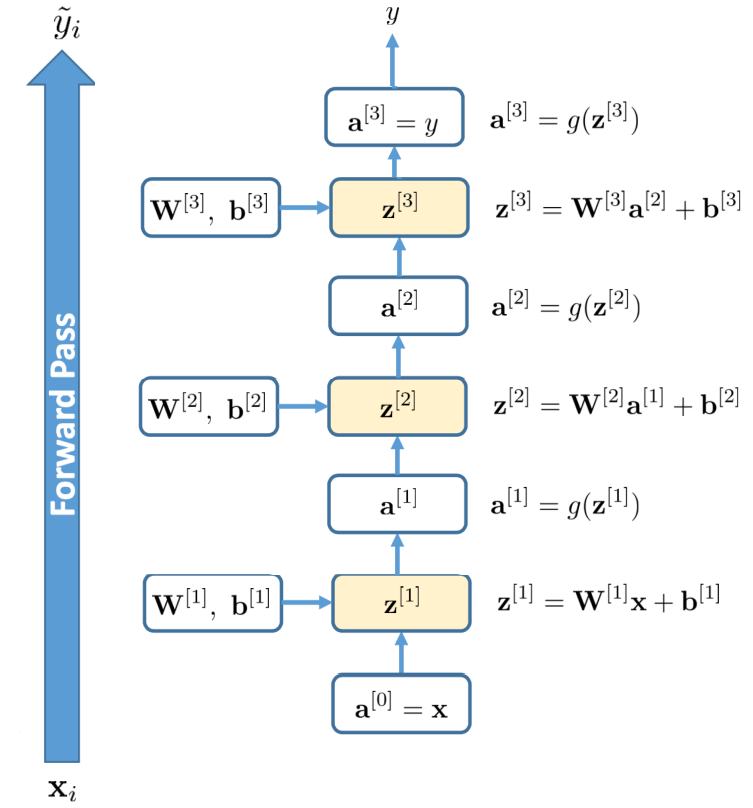
Back Propagation: Vectorization

- We compute loss function $\mathcal{L}$ using forward pass.



Back Propagation: Vectorization

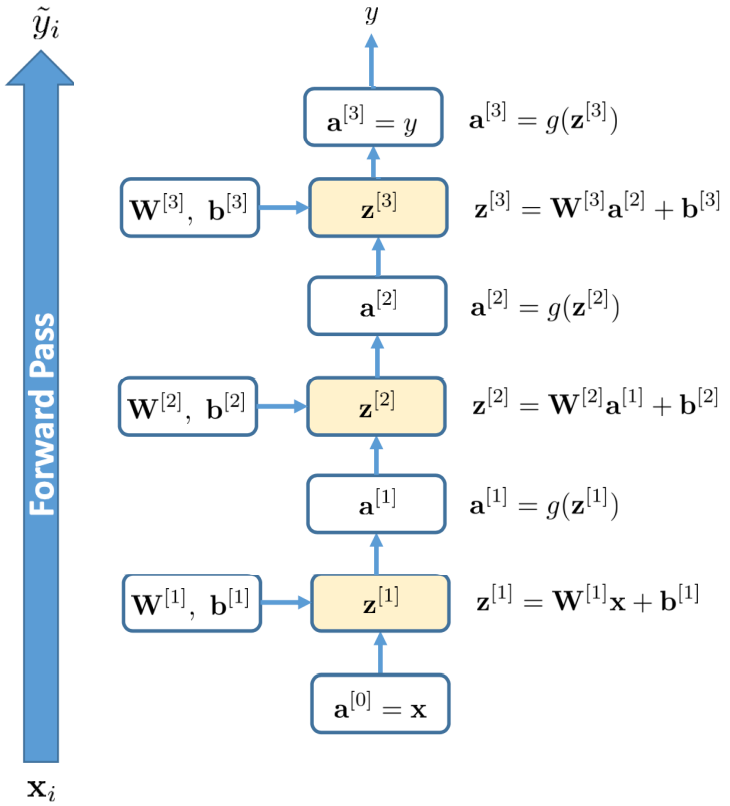
- We compute loss function $\mathcal{L}$ using forward pass.



Back Propagation: Vectorization

- We compute loss function $\mathcal{L}$ using forward pass.
- We update $\mathbf{W}^{[\ell]}$ and $\mathbf{b}^{[\ell]}$ using gradient descent as:

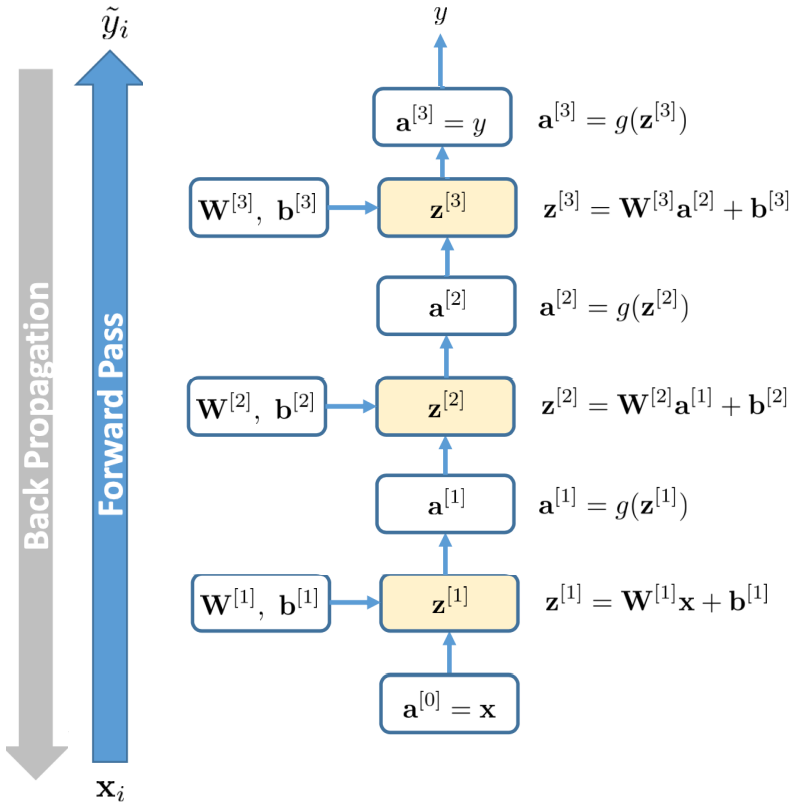
$$\mathbf{W}^{[\ell]} = \mathbf{W}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[\ell]}} \qquad \mathbf{b}^{[\ell]} = \mathbf{b}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[\ell]}}$$



Back Propagation: Vectorization

- We compute loss function $\mathcal{L}$ using forward pass.
- We update $\mathbf{W}^{[\ell]}$ and $\mathbf{b}^{[\ell]}$ using gradient descent as:

$$\mathbf{W}^{[\ell]} = \mathbf{W}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[\ell]}} \qquad \mathbf{b}^{[\ell]} = \mathbf{b}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[\ell]}}$$



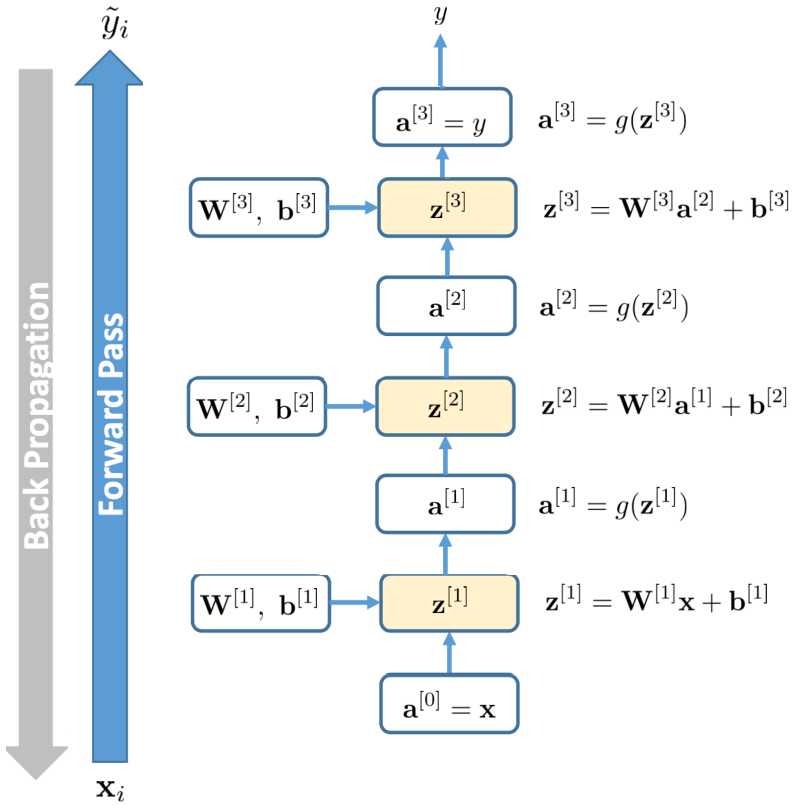
Back Propagation: Vectorization

- We compute loss function $\mathcal{L}$ using forward pass.
- We update $\mathbf{W}^{[\ell]}$ and $\mathbf{b}^{[\ell]}$ using gradient descent as:

$$\mathbf{W}^{[\ell]} = \mathbf{W}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[\ell]}} \qquad \mathbf{b}^{[\ell]} = \mathbf{b}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[\ell]}}$$

Partial Derivatives:

$$\frac{\partial \mathcal{L}}{\mathbf{W}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{W}^{[3]}} \qquad \frac{\partial \mathcal{L}}{\mathbf{b}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{b}^{[3]}}$$



Back Propagation: Vectorization

- We compute loss function $\mathcal{L}$ using forward pass.
- We update $\mathbf{W}^{[\ell]}$ and $\mathbf{b}^{[\ell]}$ using gradient descent as:

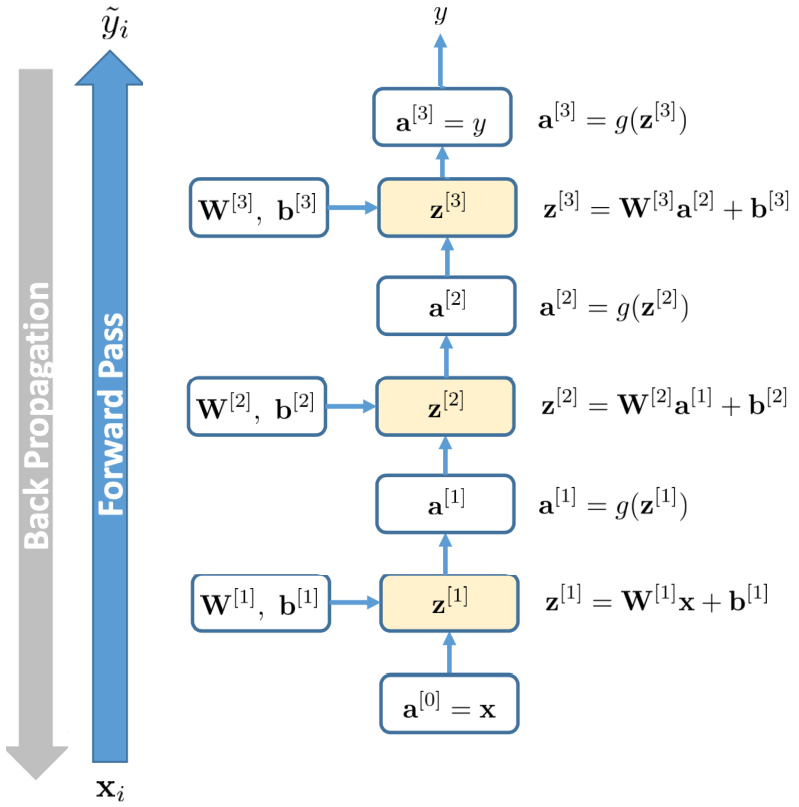
$$\mathbf{W}^{[\ell]} = \mathbf{W}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[\ell]}} \qquad \mathbf{b}^{[\ell]} = \mathbf{b}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[\ell]}}$$

Partial Derivatives:

$$\frac{\partial \mathcal{L}}{\mathbf{W}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{W}^{[3]}} \qquad \frac{\partial \mathcal{L}}{\mathbf{b}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{b}^{[3]}}$$

$$\frac{\partial \mathcal{L}}{\mathbf{W}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}}$$

$$\frac{\partial \mathcal{L}}{\mathbf{b}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{b}^{[2]}}$$



Back Propagation: Vectorization

- We compute loss function $\mathcal{L}$ using forward pass.
- We update $\mathbf{W}^{[\ell]}$ and $\mathbf{b}^{[\ell]}$ using gradient descent as:

$$\mathbf{W}^{[\ell]} = \mathbf{W}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[\ell]}} \qquad \mathbf{b}^{[\ell]} = \mathbf{b}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[\ell]}}$$

Partial Derivatives:

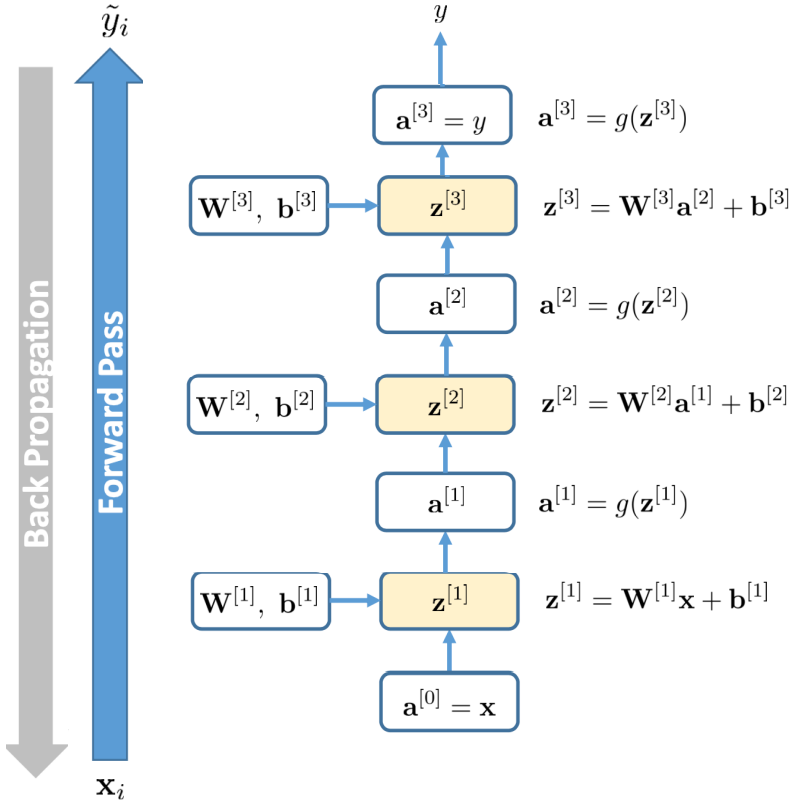
$$\frac{\partial \mathcal{L}}{\mathbf{W}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{W}^{[3]}} \qquad \frac{\partial \mathcal{L}}{\mathbf{b}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{b}^{[3]}}$$

$$\frac{\partial \mathcal{L}}{\mathbf{W}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}}$$

$$\frac{\partial \mathcal{L}}{\mathbf{b}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{b}^{[2]}}$$

$$\frac{\partial \mathcal{L}}{\mathbf{W}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[1]}} \frac{\partial \mathbf{a}^{[1]}}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{W}^{[1]}}$$

$$\frac{\partial \mathcal{L}}{\mathbf{b}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[1]}} \frac{\partial \mathbf{a}^{[1]}}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{b}^{[1]}}$$



Neural Networks In Keras

```
from keras.models import Sequential
from keras.layers import Dense

# built model
model = Sequential([
Dense(32, activation='relu', input_shape=(10,)),
Dense(32, activation='relu'),
Dense(1, activation='sigmoid'),
])

# compile your model
model.compile(optimizer='sgd',
loss='binary_crossentropy',
metrics=['accuracy'])
```

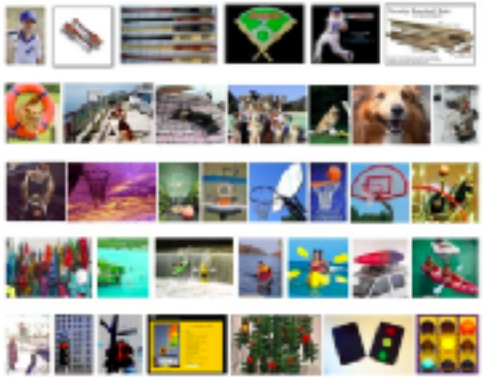
Neural Networks In Keras

```
# now train your model using fit
hist = model.fit(X_train, Y_train,
batch_size=32, epochs=100,
validation_data=(X_val, Y_val))
```

```
# evaluate your model using model.evaluate
model.evaluate(X_test, Y_test)[1]
```

Relentless research on visual recognition

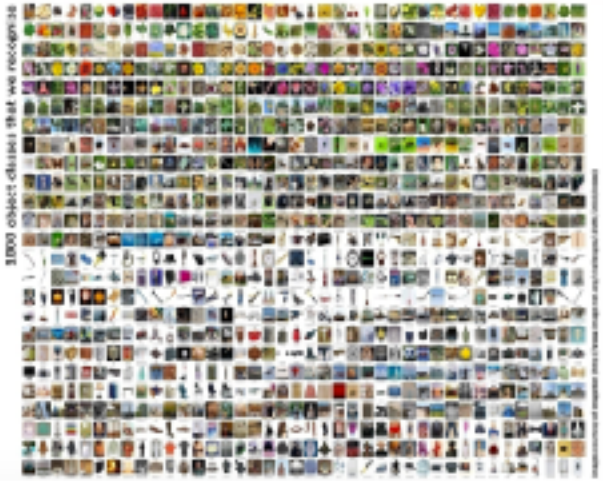
Caltech 101



80 Million Tiny Images

PASCAL VOC

ImageNet

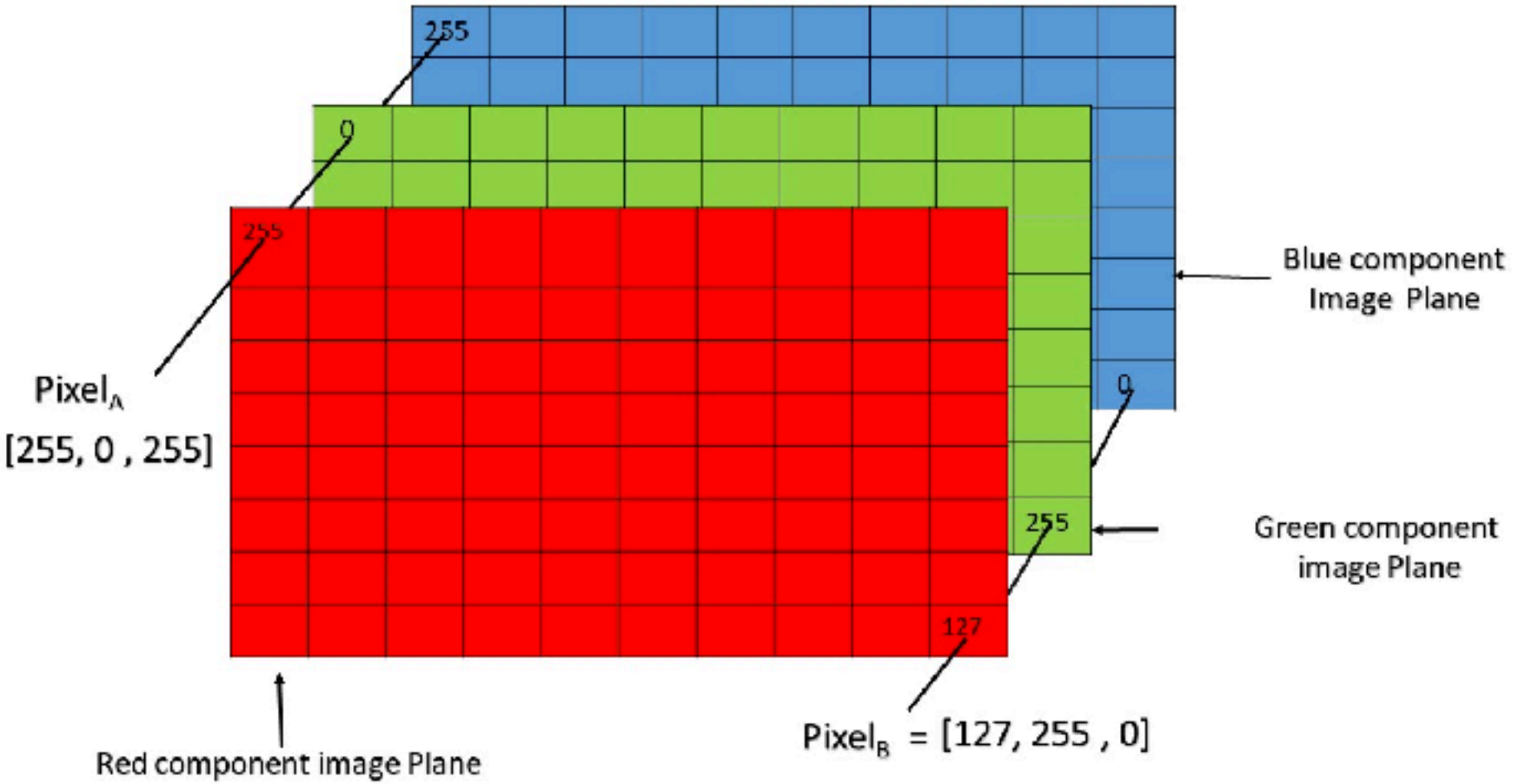


Convolution

Convolutional Neural Networks

- Automatic feature extraction.
- Highly accurate at image recognition & classification.
- Weight sharing.
- Minimizes computation.
- Ability to handle large datasets.
- Hierarchical learning

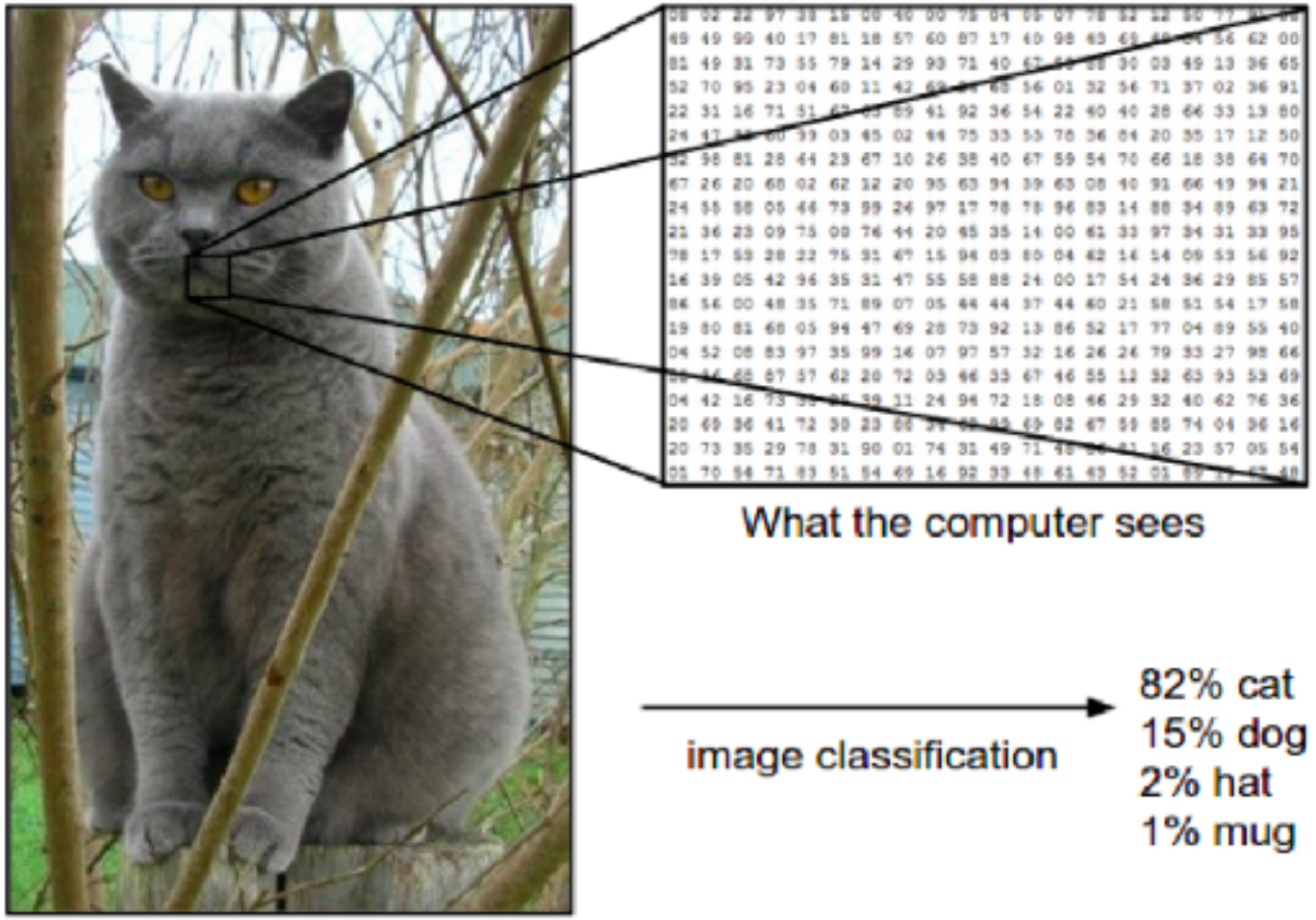
Digital Image



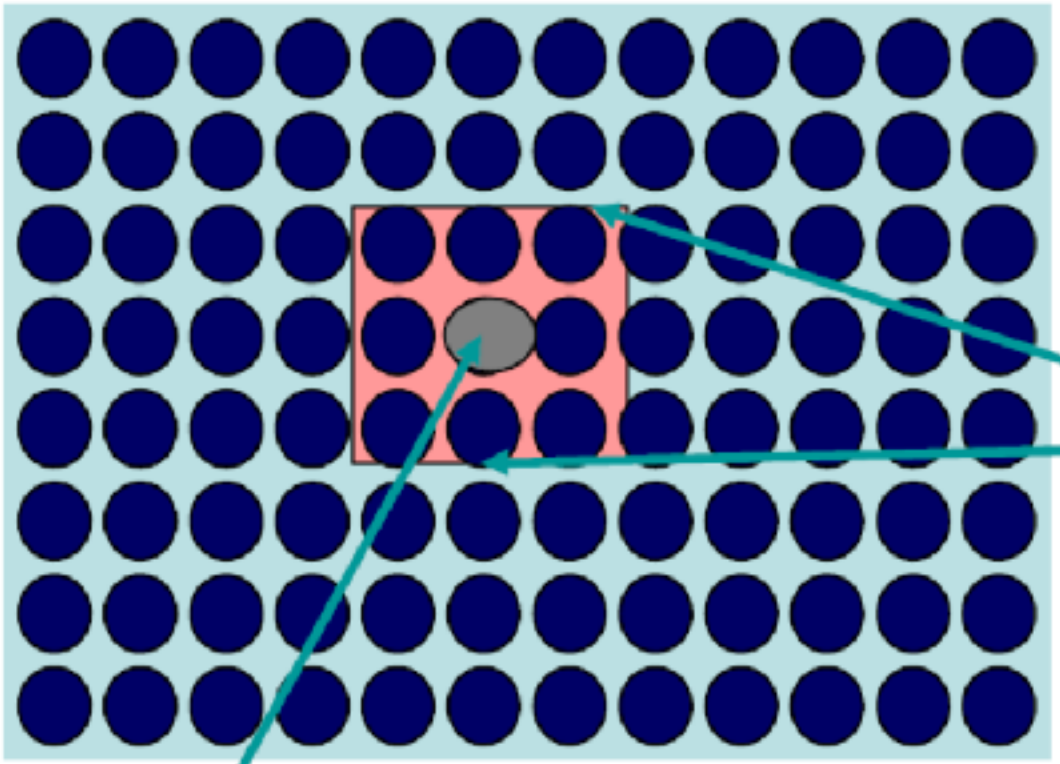
Pixel of an RGB image are formed from the corresponding pixel of the three component images

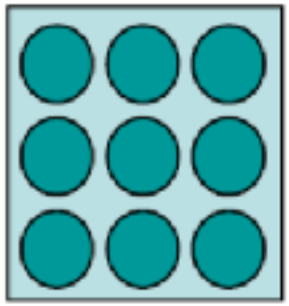
https://www.geeksforgeeks.org/matlab-rgb-image-representation/

Digital Image



Discrete 2D convolution



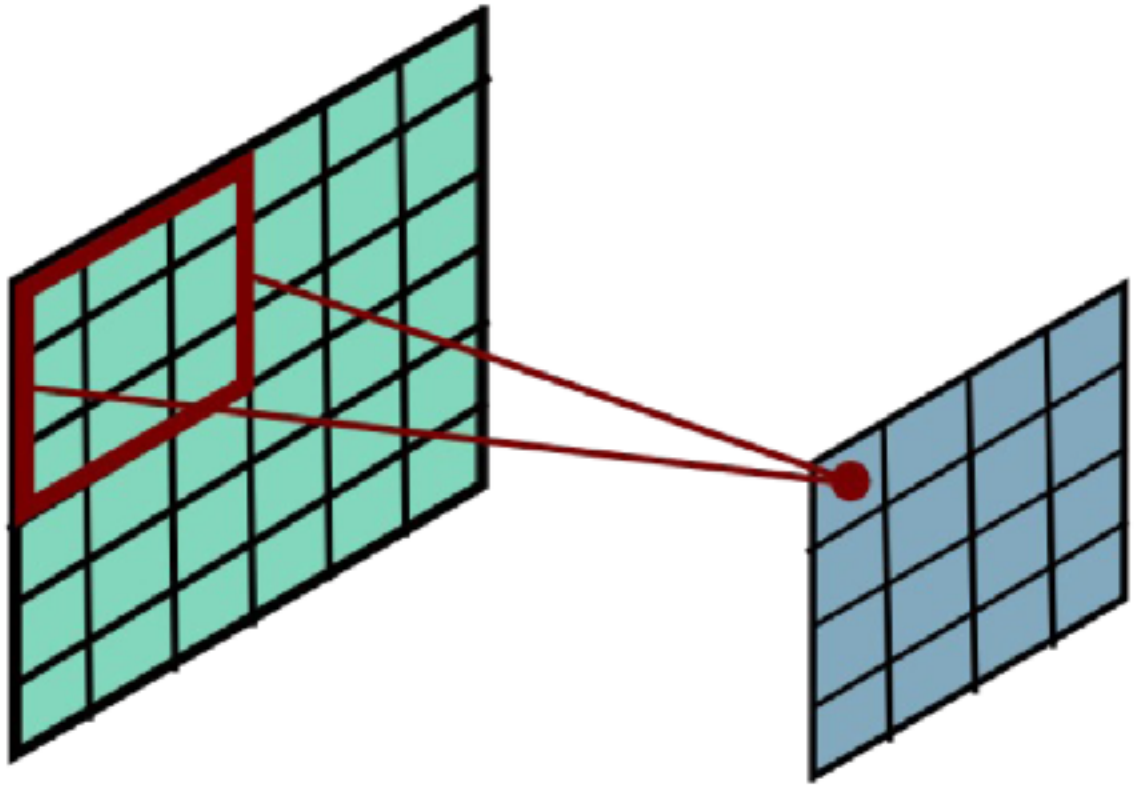


3X3 convolution kernel

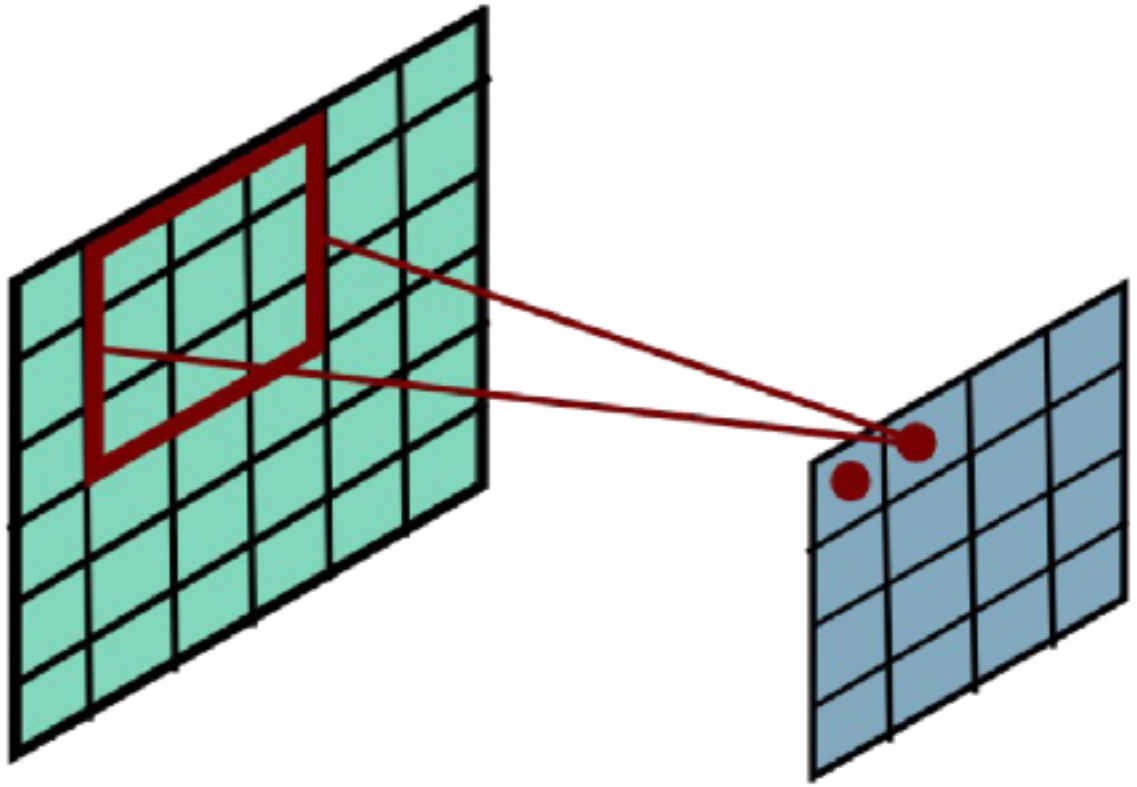
Pixels involved in the convolution process for the single pixel *p*

Pixel *p* to be processed

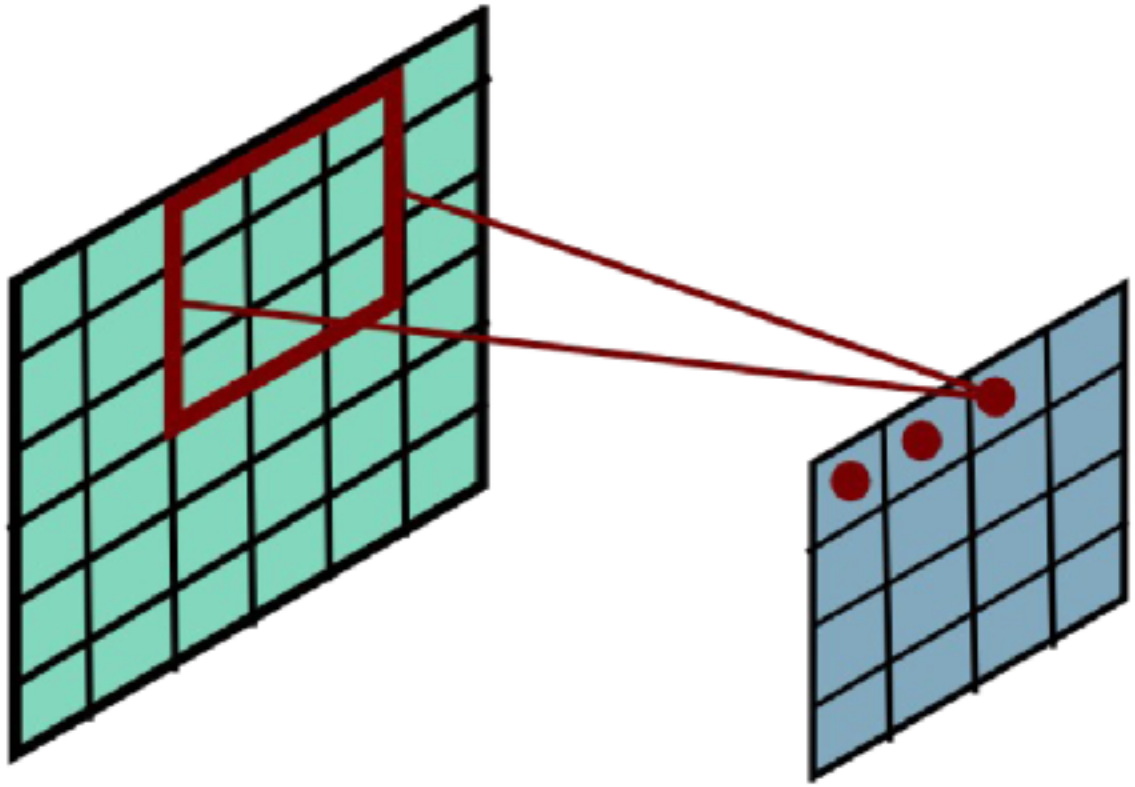
Convolution procedure



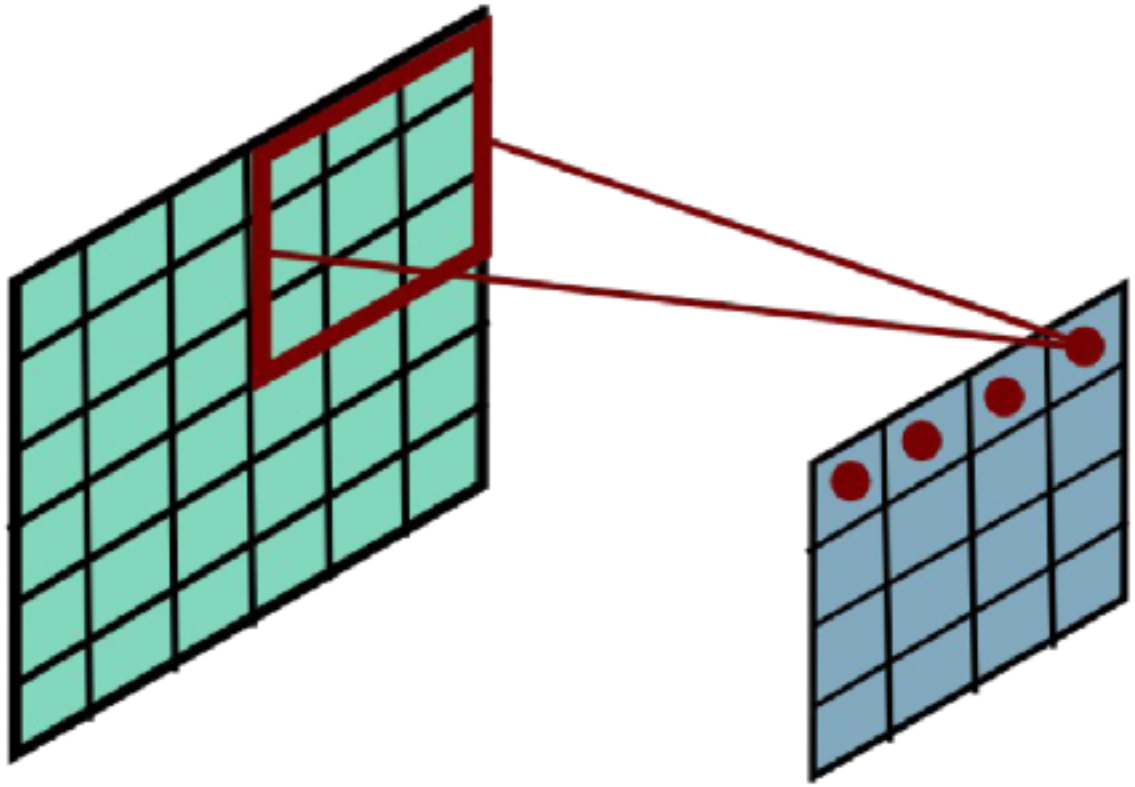
Convolution procedure



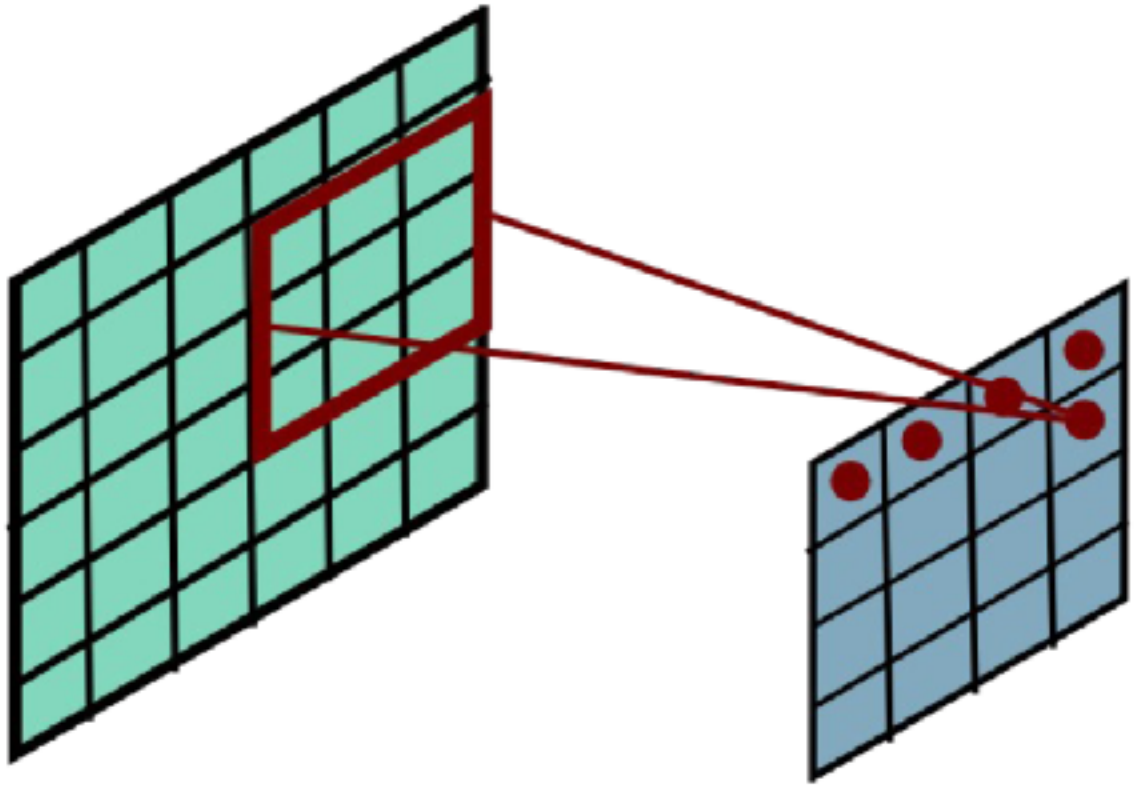
Convolution procedure



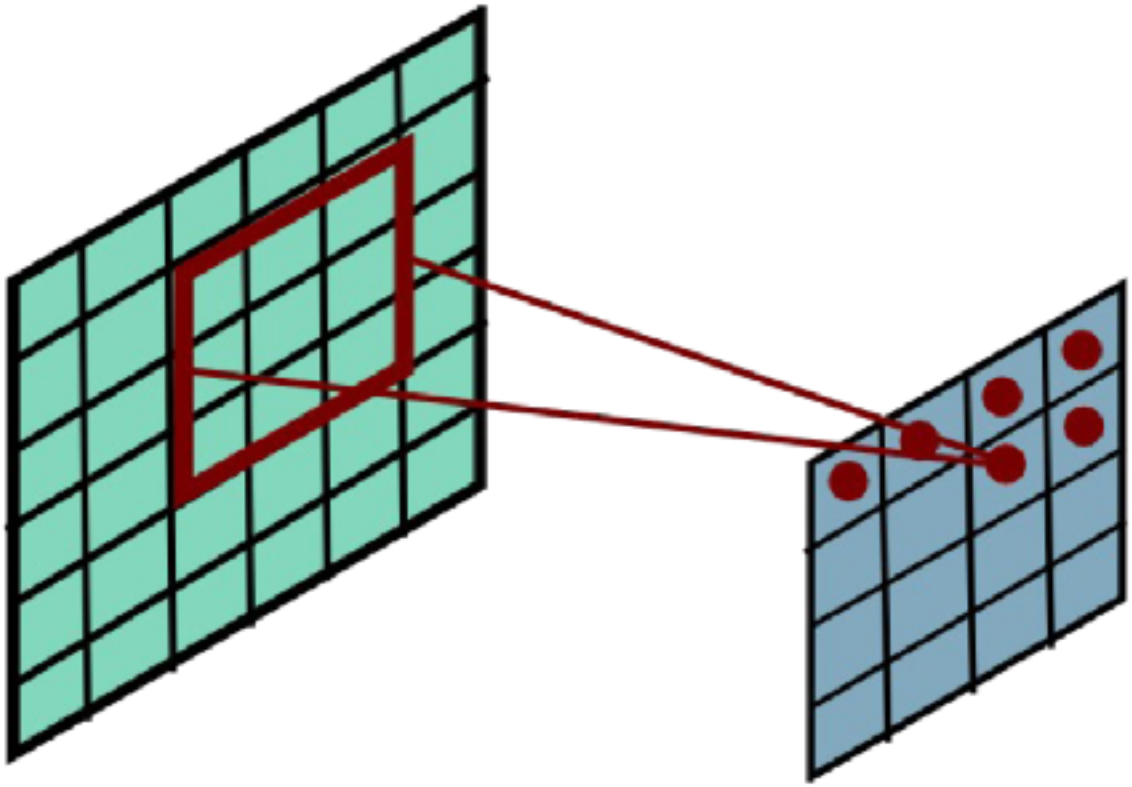
Convolution procedure



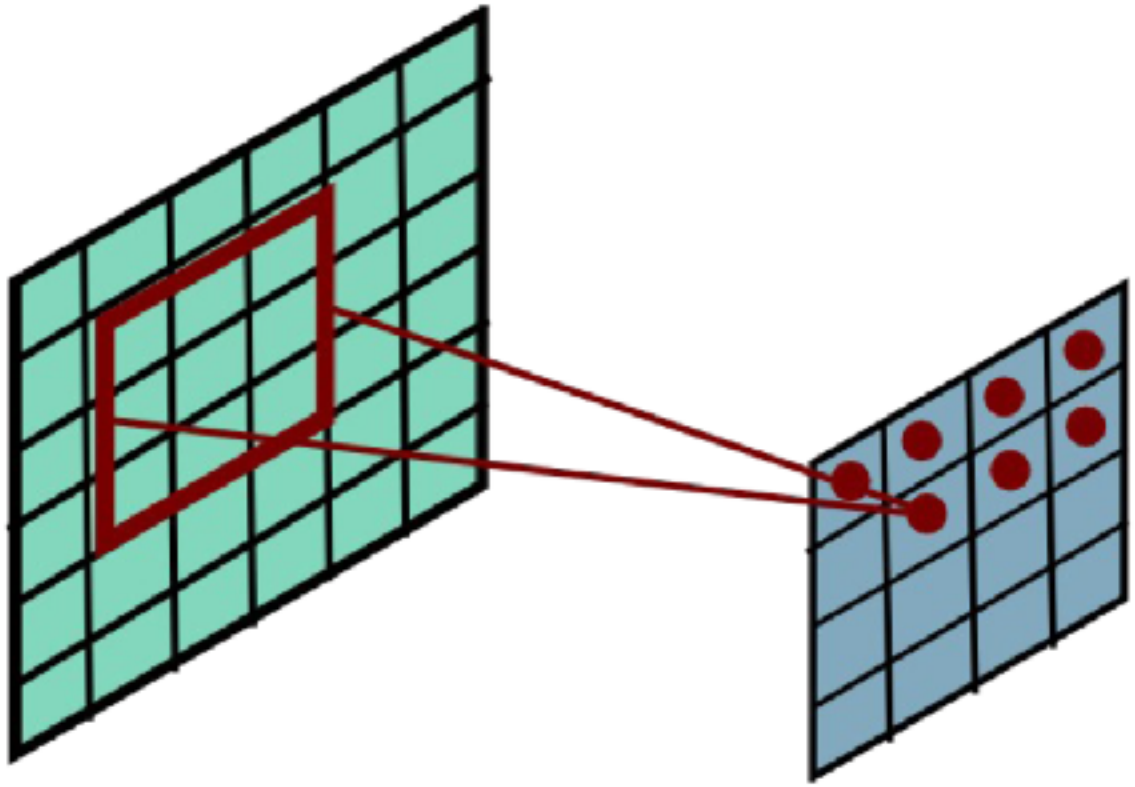
Convolution procedure



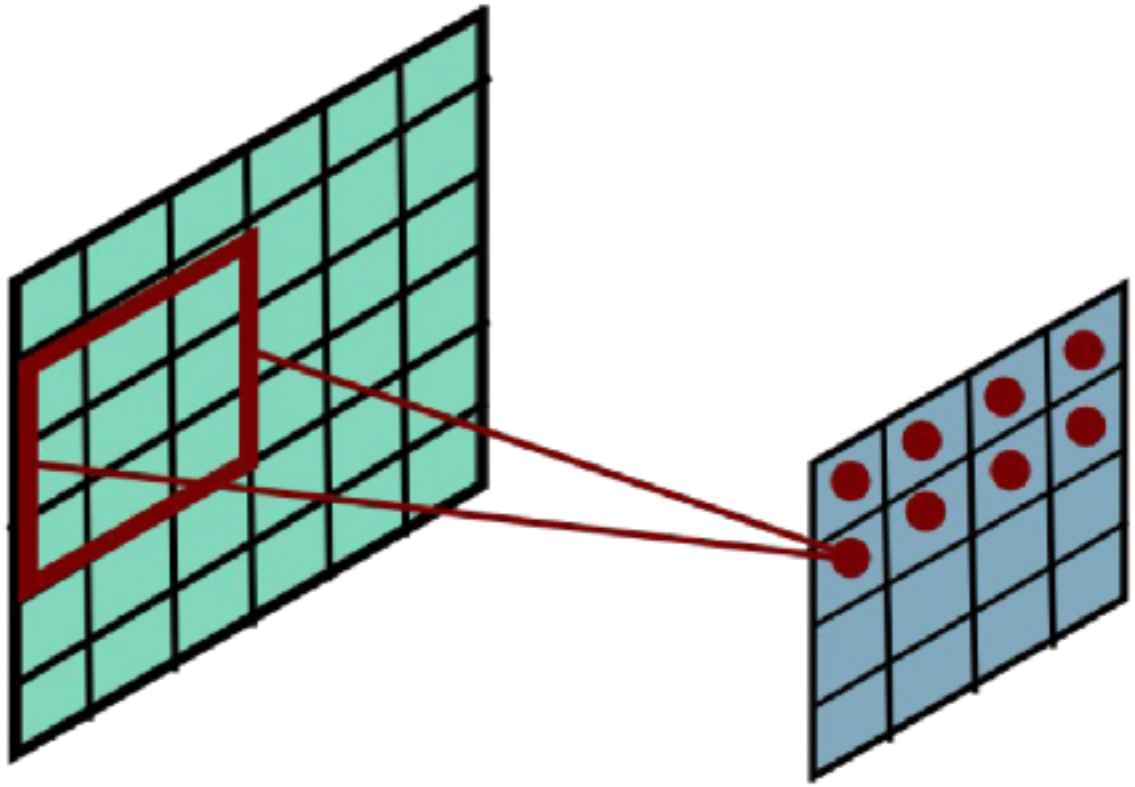
Convolution procedure



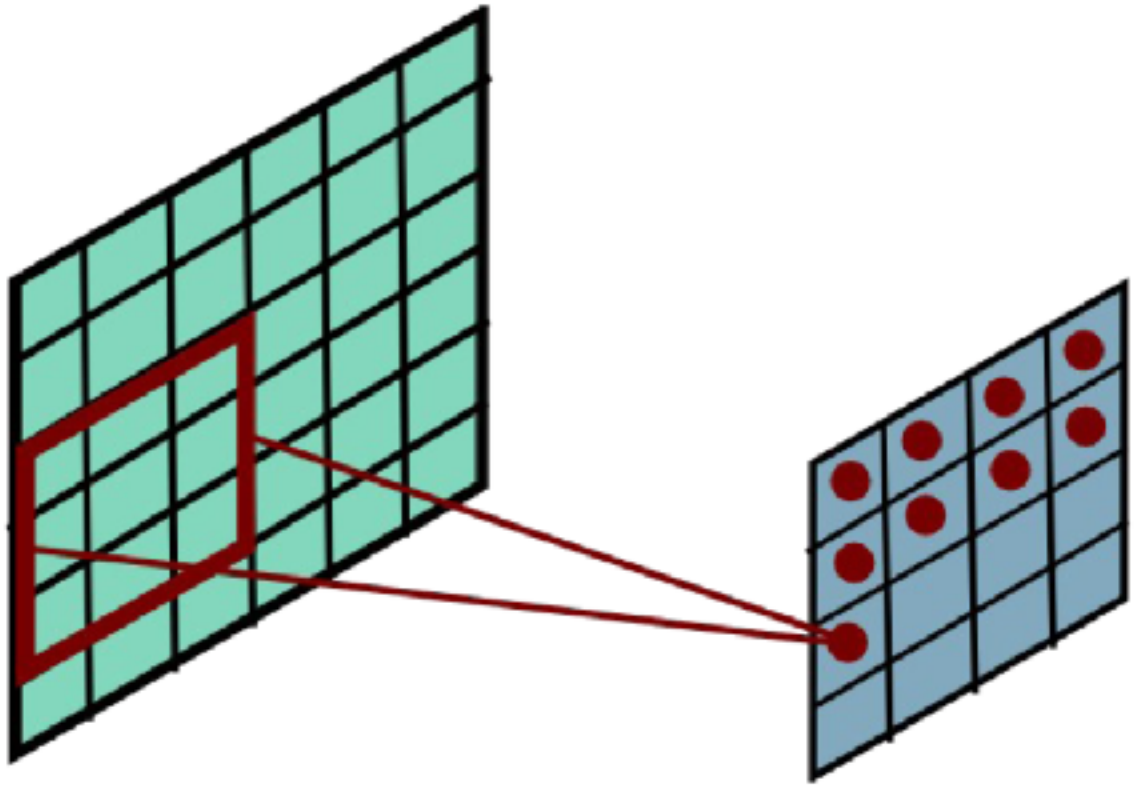
Convolution procedure



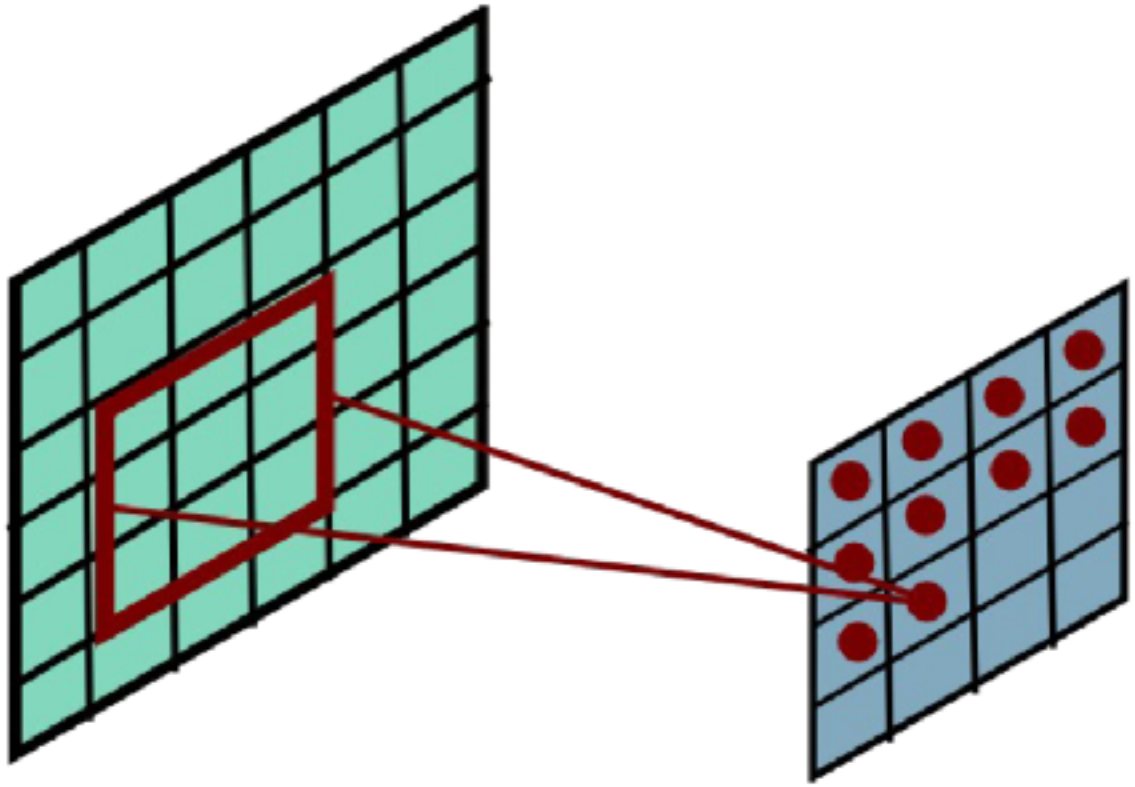
Convolution procedure



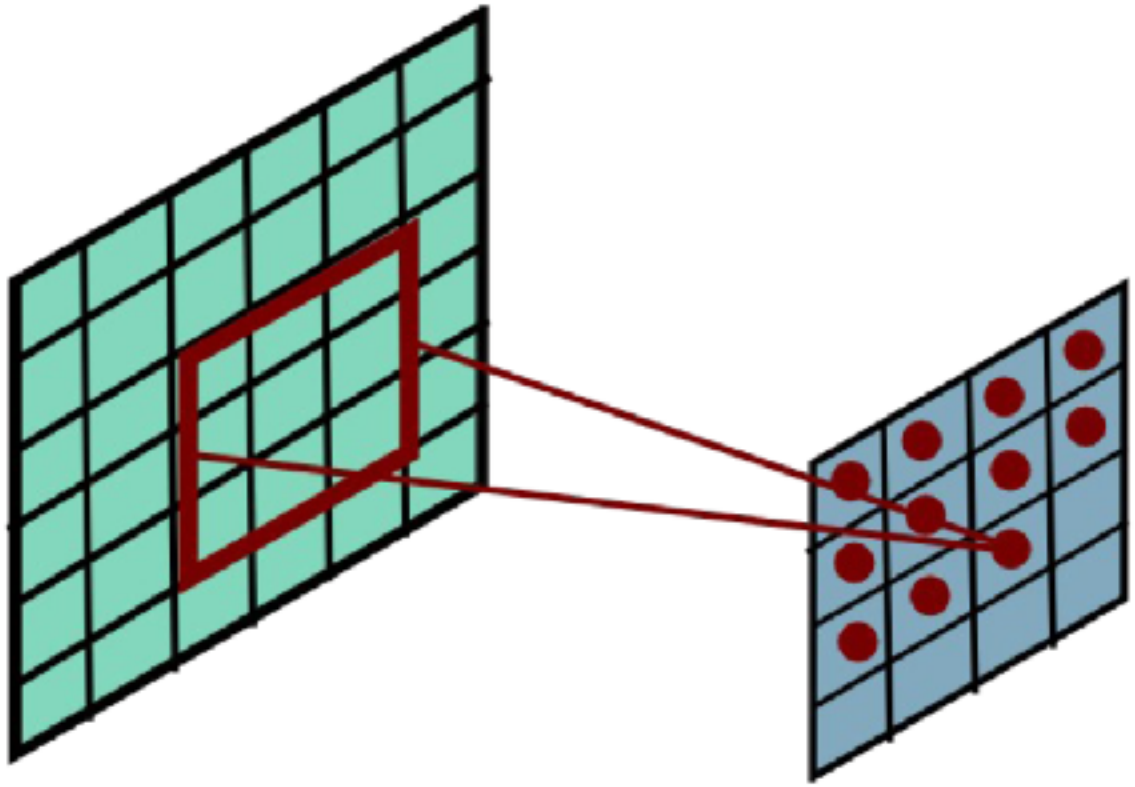
Convolution procedure



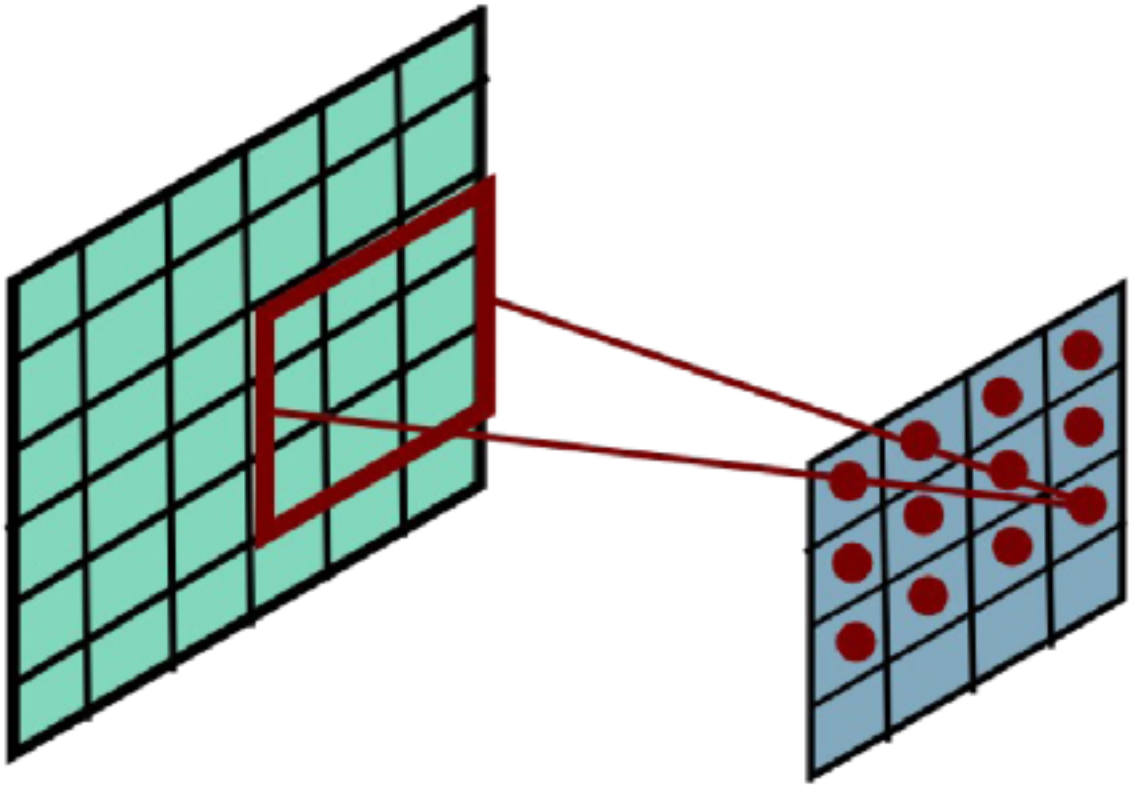
Convolution procedure



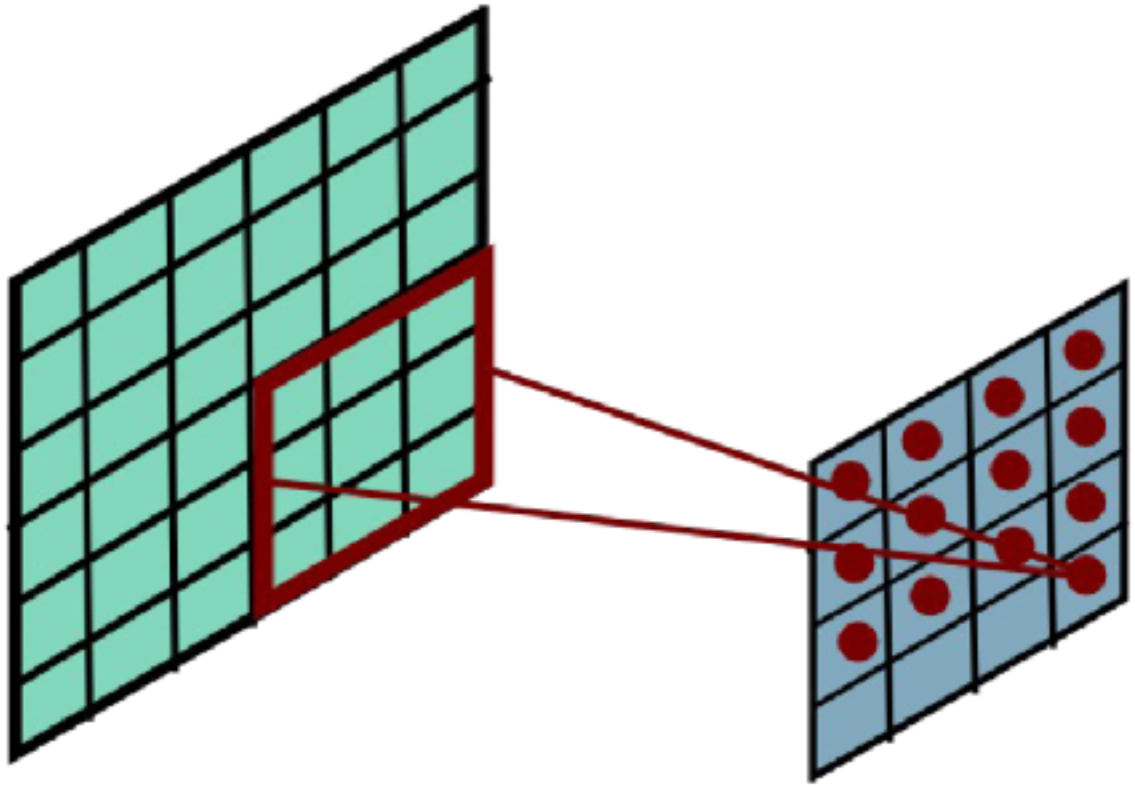
Convolution procedure



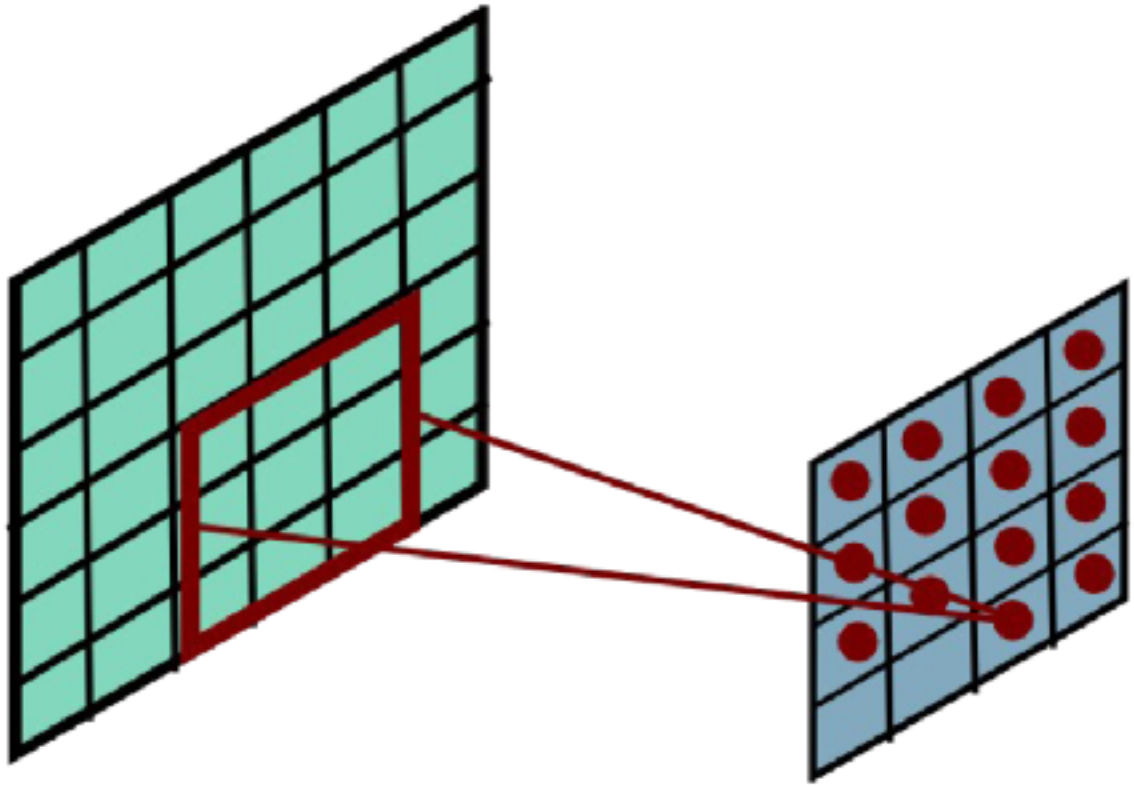
Convolution procedure



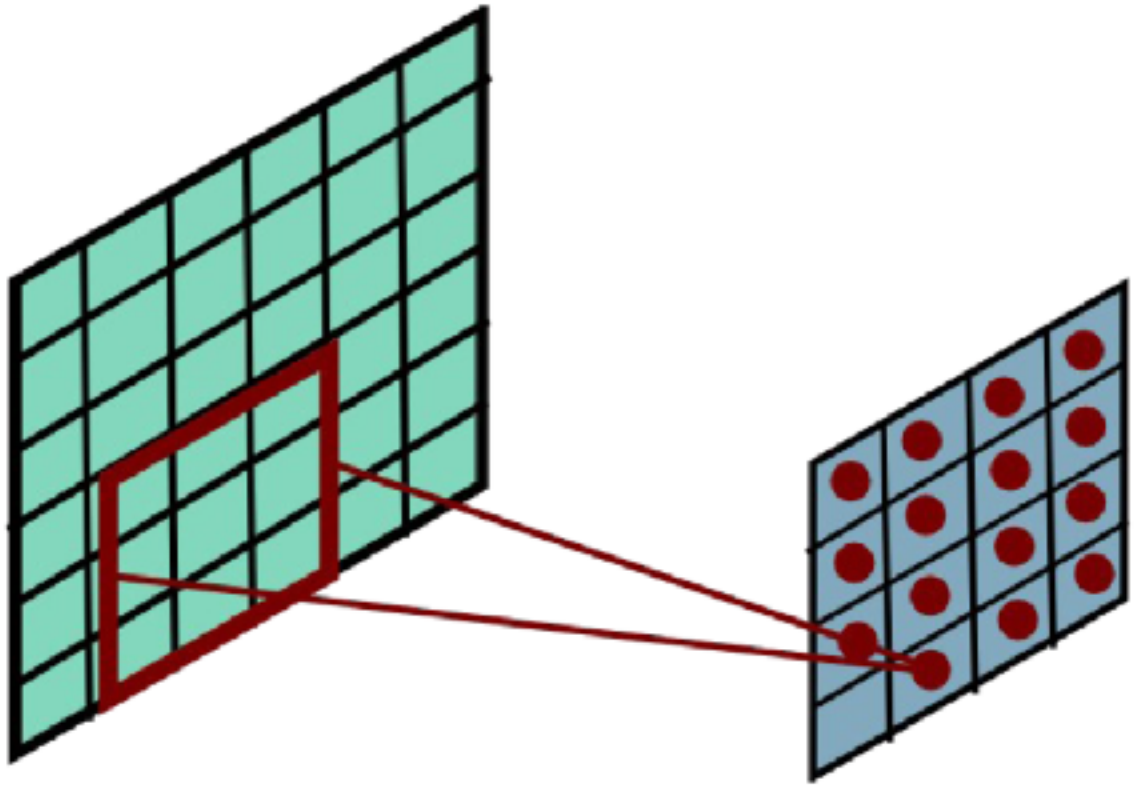
Convolution procedure



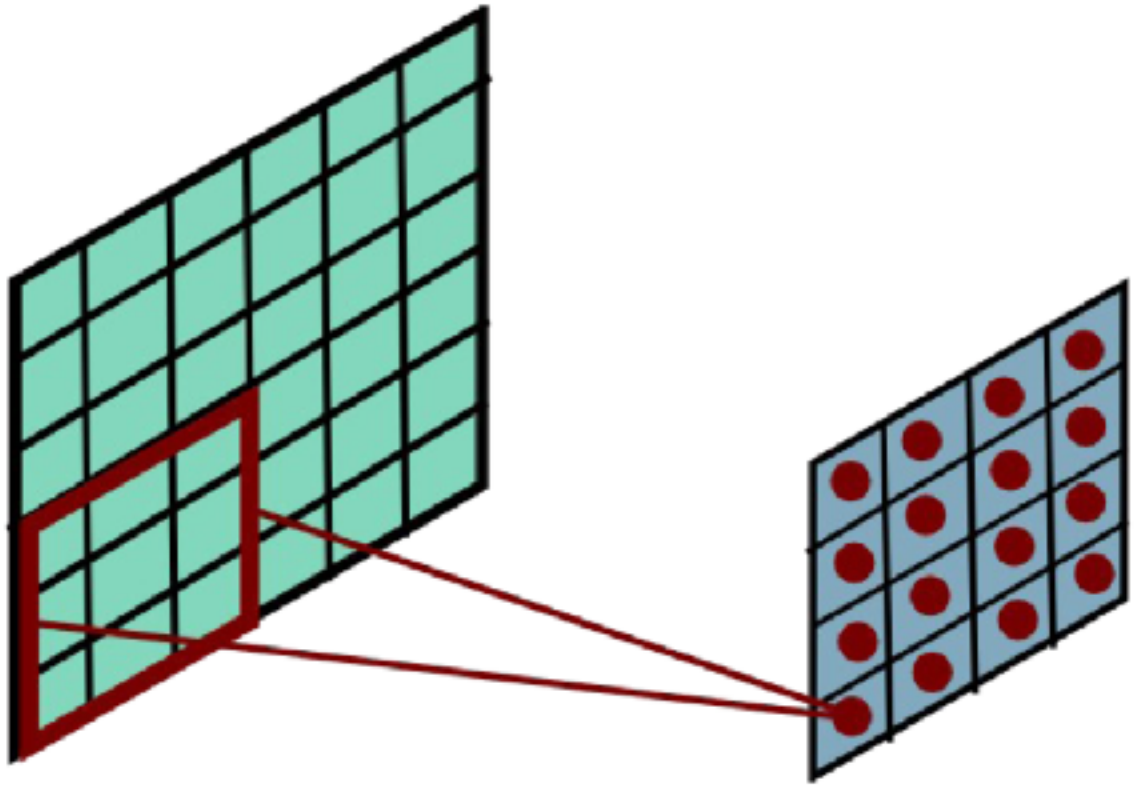
Convolution procedure



Convolution procedure



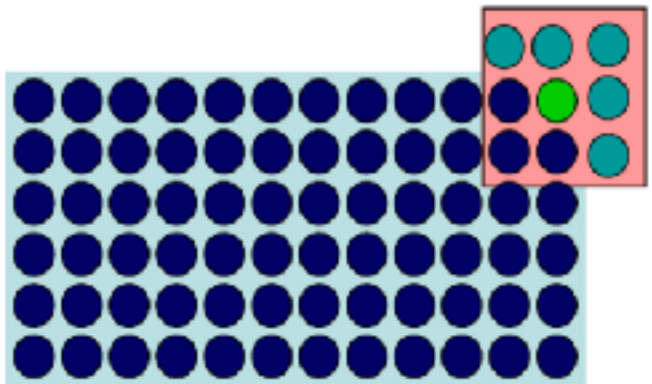
Convolution procedure



Mathieu et al. “Fast training of CNNs through FFTs” ICLR 2014

Convolution procedure

- Positioning
  - Place the centre of the window over the pixel position to be filtered
- Multiply
  - the original pixel values of the image by the filter values corresponding to their location
- Add
  - Add together the obtained multiplication results
  - The result of this addition is the convolution result
- Border
  - Mirror the border (Padding)



Numerical Example

Image

1	1	5	7	3
4	5	4	7	8
9	1	3	1	6
2	4	5	2	1
1	3	1	2	1
1	3	3	2	4

Filter

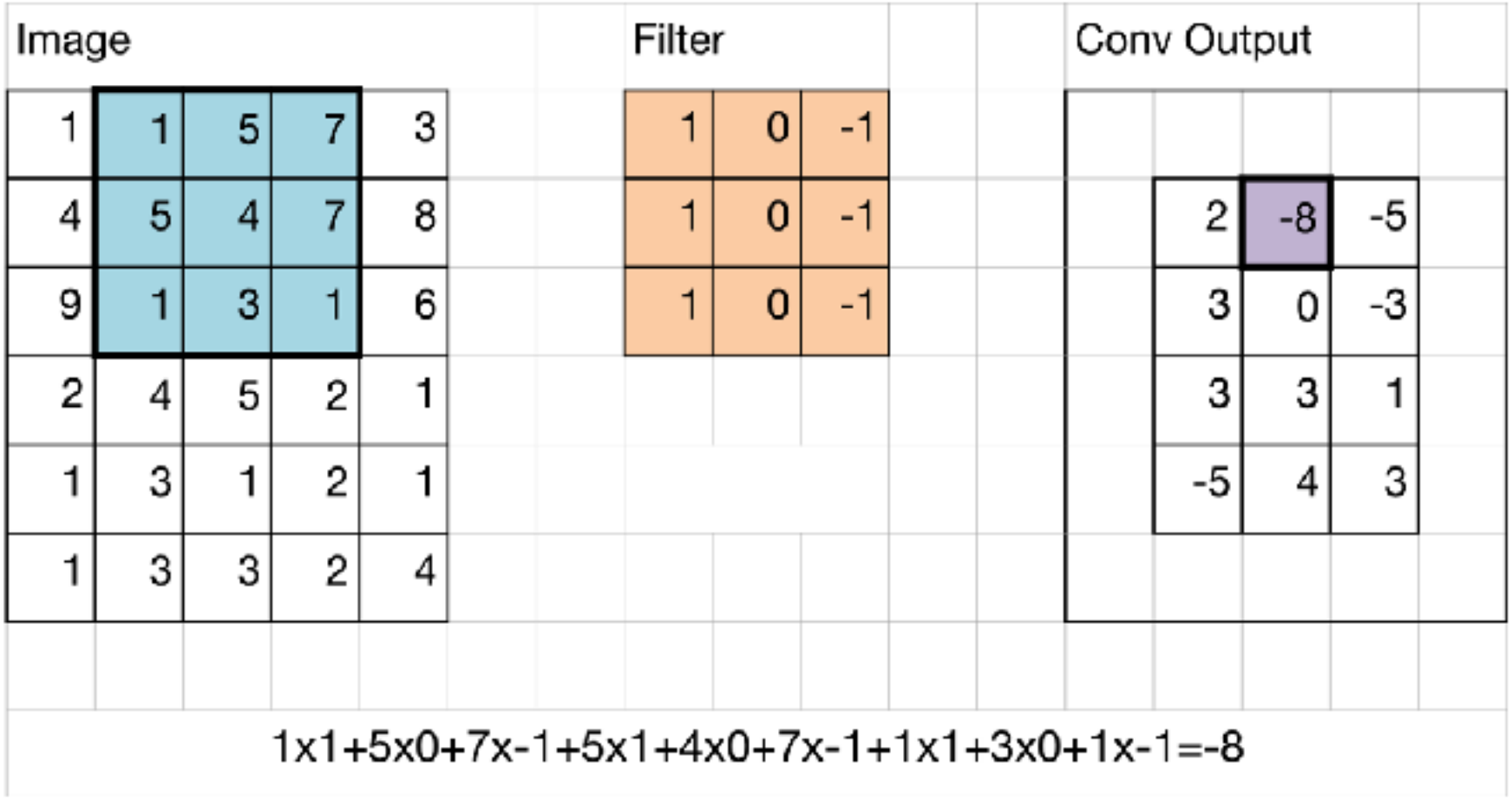
1	0	-1
1	0	-1
1	0	-1

Conv Output

2	-8	-5
3	0	-3
3	3	1
-5	4	3

1x1+1x0+5x-1+4x1+5x0+4x-1+9x1+1x0+3x-1=2

Numerical Example



Numerical Example

Image

1	1	5	7	3
4	5	4	7	8
9	1	3	1	6
2	4	5	2	1
1	3	1	2	1
1	3	3	2	4

Filter

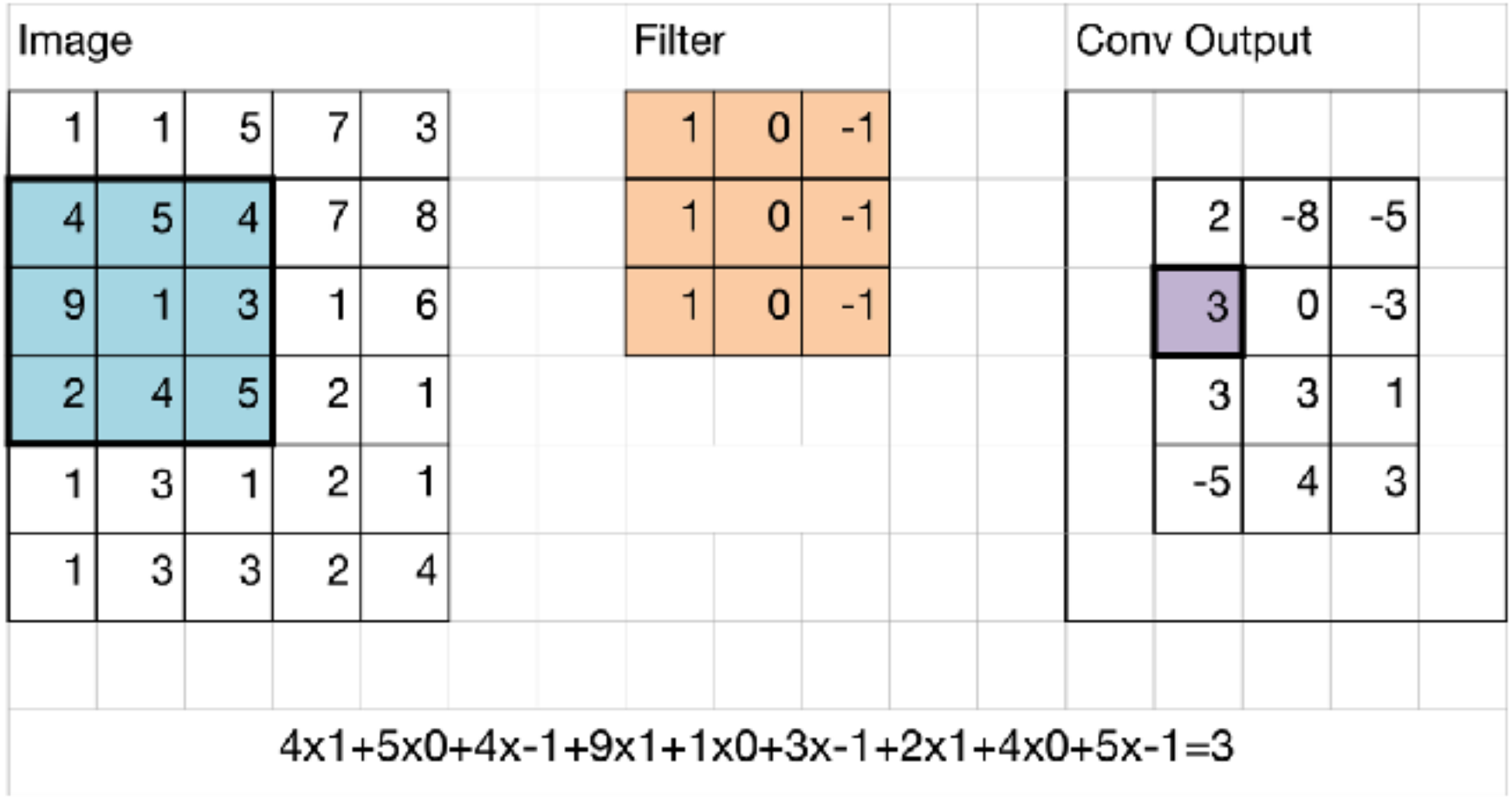
1	0	-1
1	0	-1
1	0	-1

Conv Output

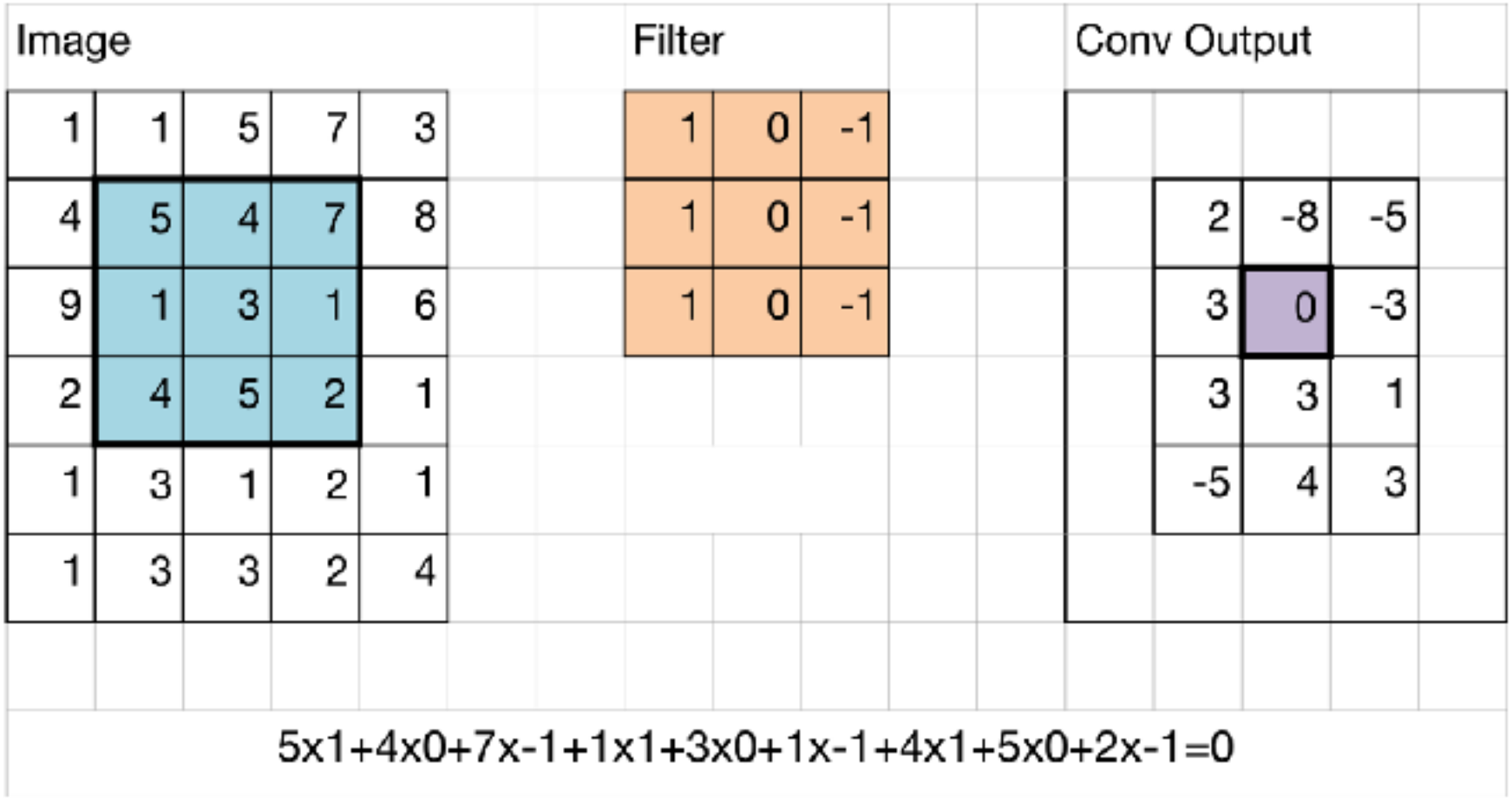
2	-8	-5
3	0	-3
3	3	1
-5	4	3

5x1+7x0+3x-1+4x1+7x0+8x-1+3x1+1x0+6x-1=-5

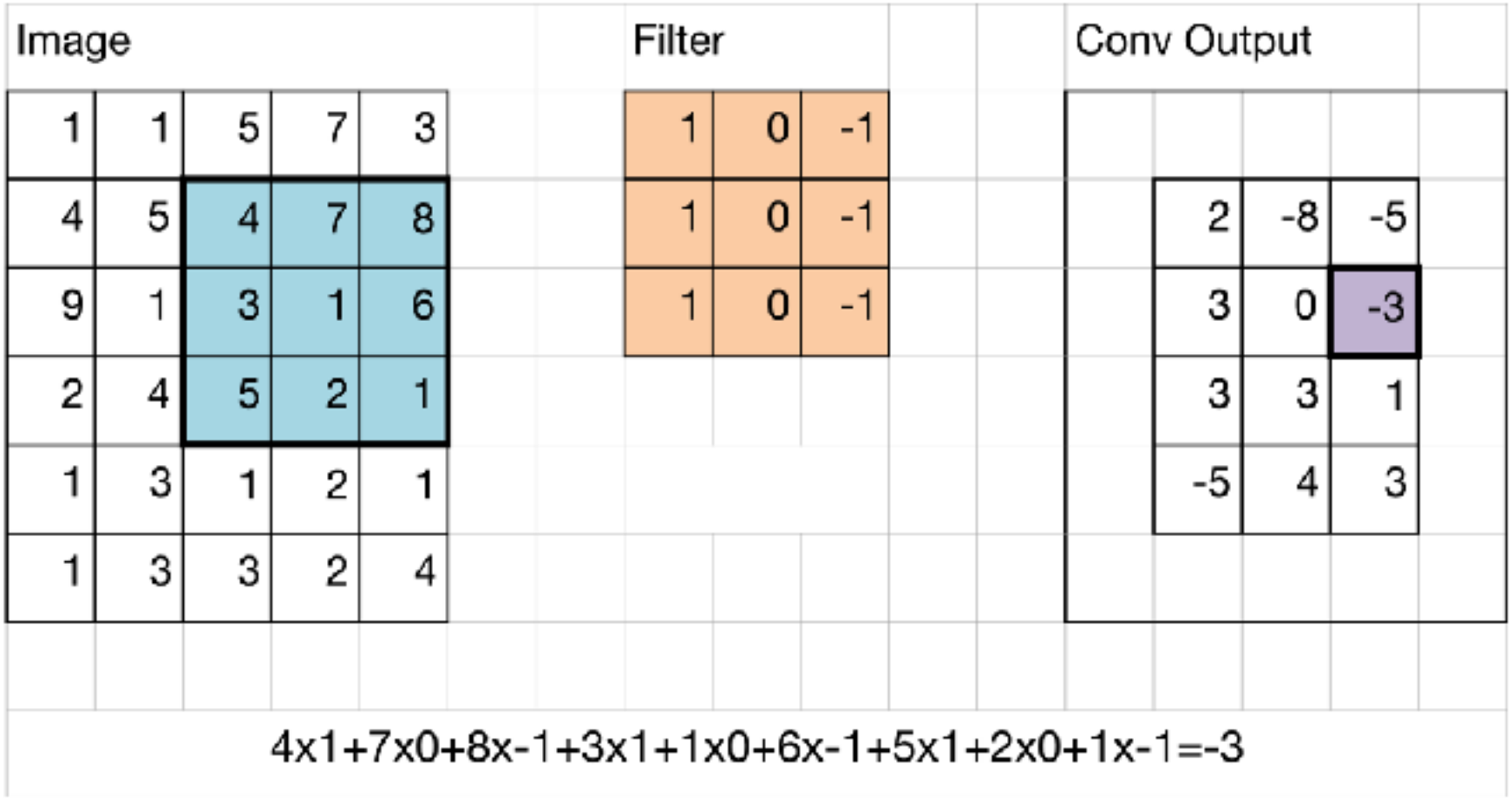
Numerical Example



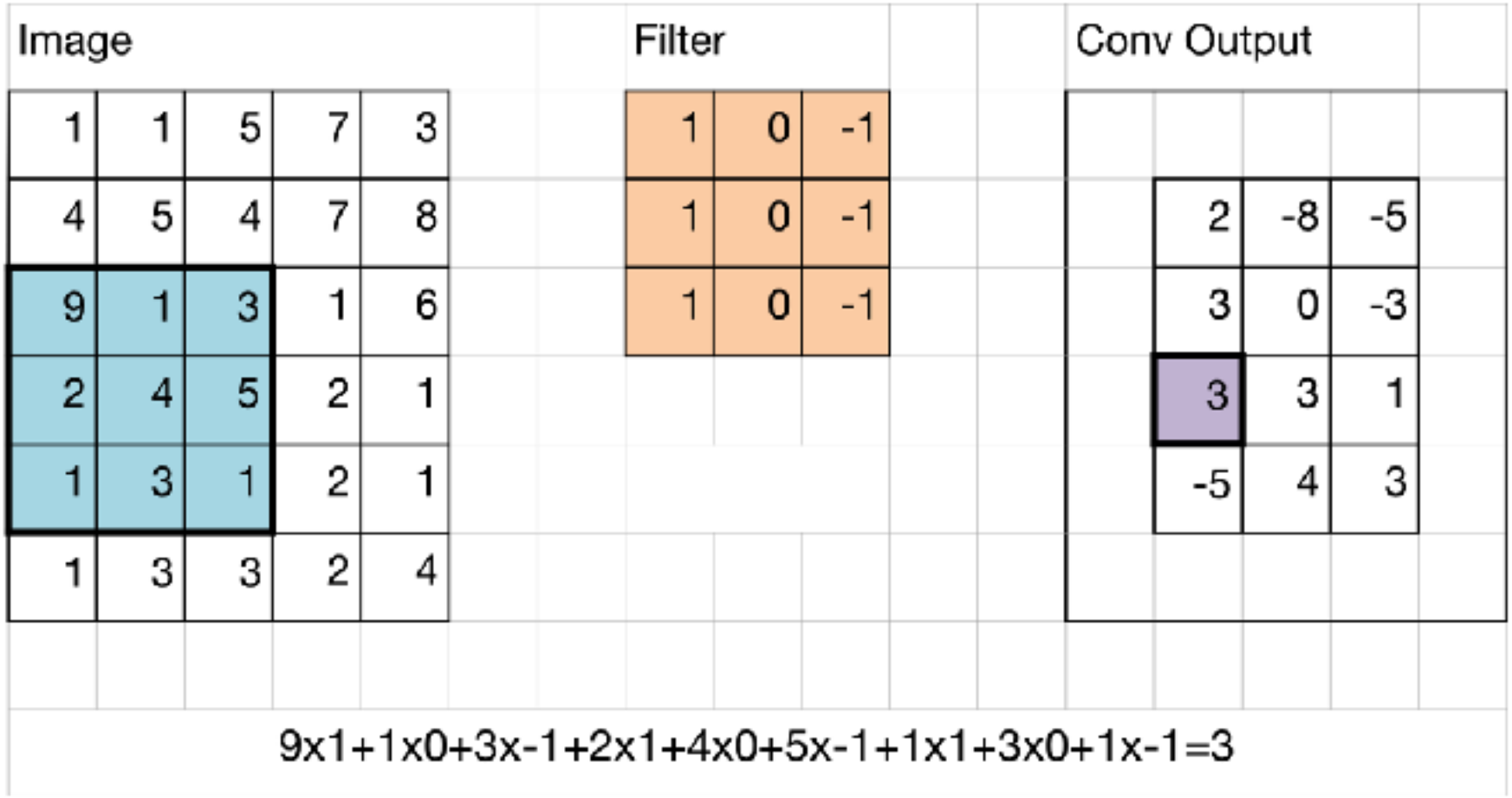
Numerical Example



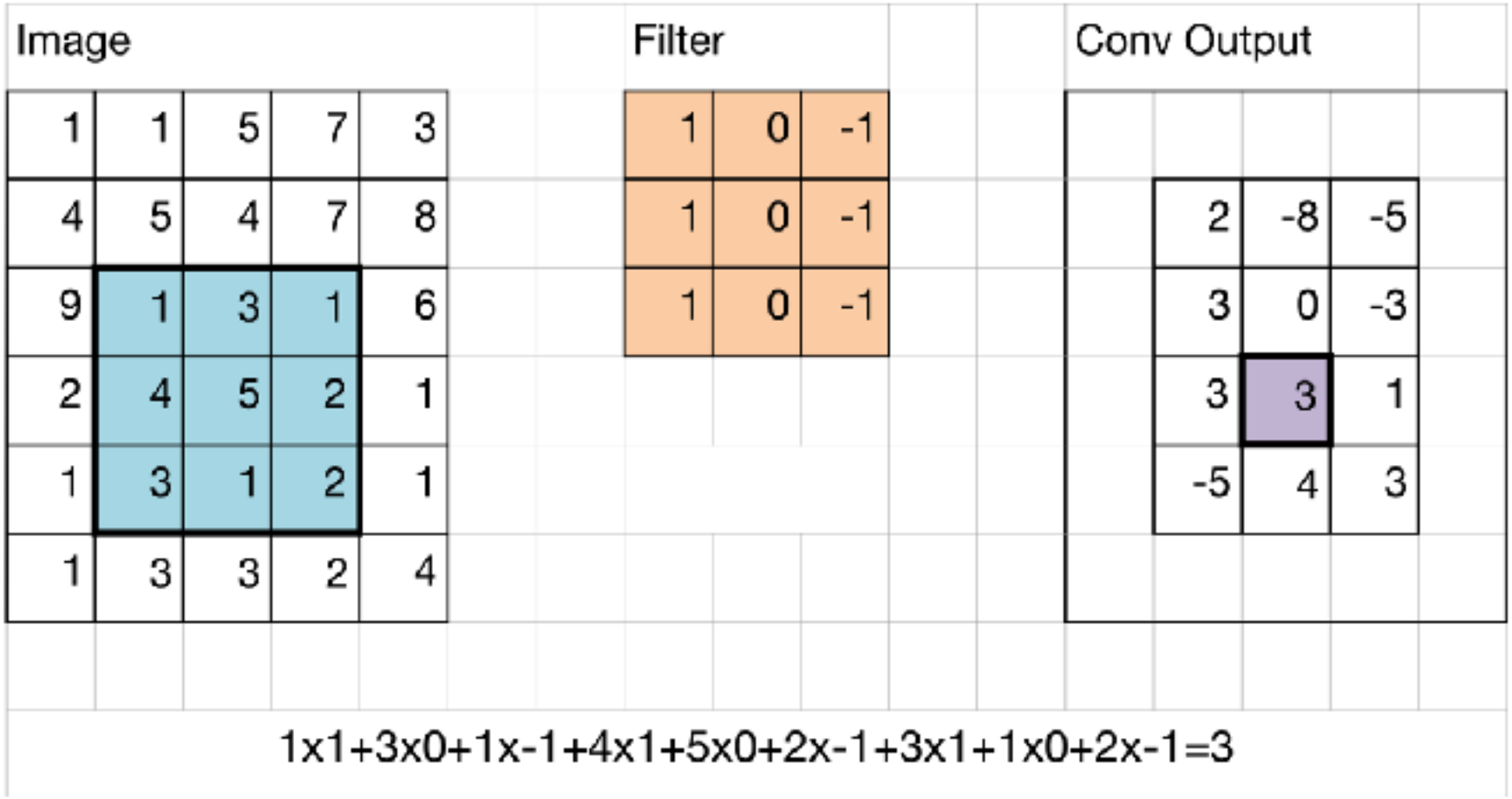
Numerical Example



Numerical Example



Numerical Example



Numerical Example

Image

1	1	5	7	3
4	5	4	7	8
9	1	3	1	6
2	4	5	2	1
1	3	1	2	1
1	3	3	2	4

Filter

1	0	-1
1	0	-1
1	0	-1

Conv Output

2	-8	-5
3	0	-3
3	3	1
-5	4	3

3x1+1x0+6x-1+5x1+2x0+1x-1+1x1+2x0+1x-1=1

Numerical Example

Image

1	1	5	7	3
4	5	4	7	8
9	1	3	1	6
2	4	5	2	1
1	3	1	2	1
1	3	3	2	4

Filter

1	0	-1
1	0	-1
1	0	-1

Conv Output

2	-8	-5
3	0	-3
3	3	1
-5	4	3

2x1+4x0+5x-1+1x1+3x0+1x-1+1x1+3x0+3x-1=-5

Numerical Example

Image

1	1	5	7	3
4	5	4	7	8
9	1	3	1	6
2	4	5	2	1
1	3	1	2	1
1	3	3	2	4

Filter

1	0	-1
1	0	-1
1	0	-1

Conv Output

2	-8	-5
3	0	-3
3	3	1
-5	4	3

4x1+5x0+2x-1+3x1+1x0+2x-1+3x1+3x0+2x-1=4

Numerical Example

Image

1	1	5	7	3
4	5	4	7	8
9	1	3	1	6
2	4	5	2	1
1	3	1	2	1
1	3	3	2	4

Filter

1	0	-1
1	0	-1
1	0	-1

Conv Output

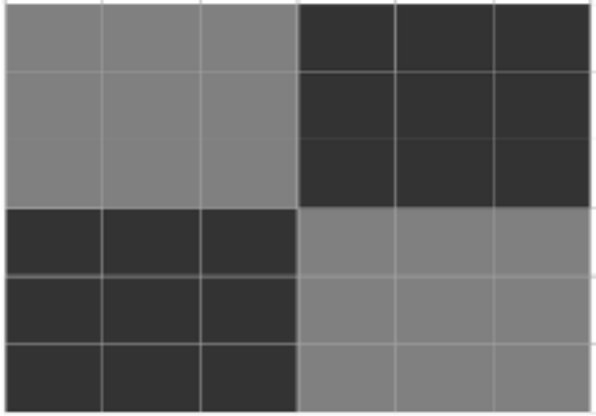
2	-8	-5
3	0	-3
3	3	1
-5	4	3

5x1+2x0+1x-1+1x1+2x0+1x-1+3x1+2x0+4x-1=3

Smoothing Filter

Image

128	128	128	52	52	52
128	128	128	52	52	52
128	128	128	52	52	52
52	52	52	128	128	128
52	52	52	128	128	128
52	52	52	128	128	128



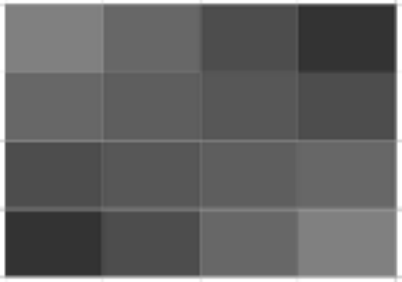
Filter

1	1	1
1	1	1
1	1	1

x1/9

Conv Output

128	103	77	52
103	94	86	77
77	86	94	103
52	77	103	128



Zero Padding - Edge Filter - Horizontal

Image

0	0	0	0	0	0	0	0
0	128	128	128	52	52	52	0
0	128	128	128	52	52	52	0
0	128	128	128	52	52	52	0
0	52	52	52	128	128	128	0
0	52	52	52	128	128	128	0
0	52	52	52	128	128	128	0
0	0	0	0	0	0	0	0

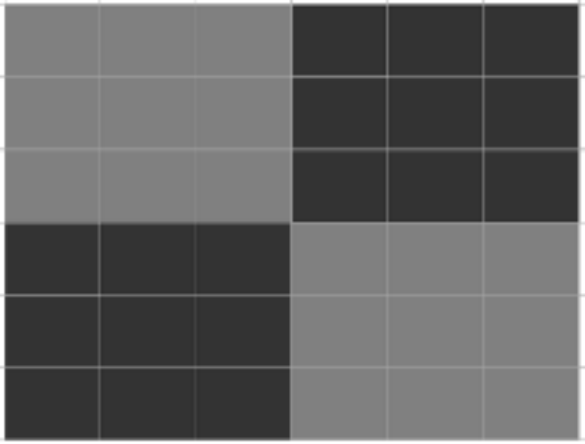
Filter

1	1	1
0	0	0
-1	-1	-1

Ignoring flip

Conv Output

-256	-384	-308	-232	-156
0	0	0	0	0
152	228	76	-76	-228
152	228	76	-76	-228
0	0	0	0	0
104	156	232	308	384



Mirror Padding 1 for Filter Size 3x3

Image

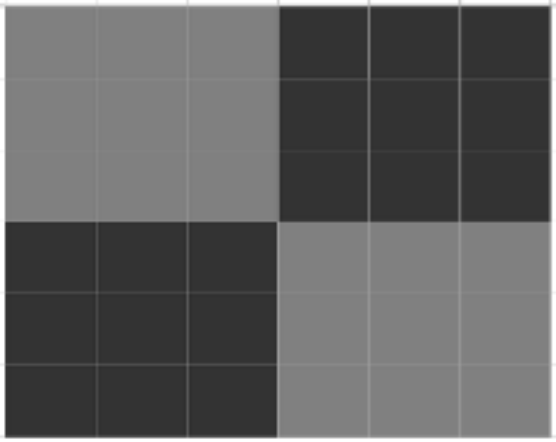
128	125	128	128	52	52	52	52
126	128	126	128	52	128	52	128
128	125	128	128	52	52	52	52
128	128	128	128	52	52	52	52
52	52	52	52	128	128	128	128
52	52	52	52	128	128	128	128
52	52	52	52	128	128	128	128
52	52	52	52	128	128	128	128

Filter

1	1	1
0	0	0
-1	-1	-1

Conv Output

0	0	0	0	0	0
-4	-2	-2	76	76	152
225	225	76	-76	-228	-228
228	228	76	-76	-228	-228
0	0	0	0	0	0
0	0	0	0	0	0



Mirror Padding 1 for Filter Size 3x3

Image

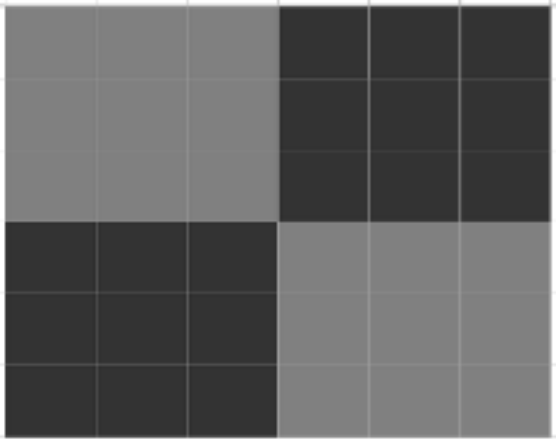
128	125	128	128	52	52	52	52
126	128	126	128	52	128	52	128
128	125	128	128	52	52	52	52
128	128	128	128	52	52	52	52
52	52	52	52	128	128	128	128
52	52	52	52	128	128	128	128
52	52	52	52	128	128	128	128
52	52	52	52	128	128	128	128

Filter

1	1	1
0	0	0
-1	-1	-1

Conv Output

0	0	0	0	0	0
-4	-2	-2	76	76	152
225	225	76	-76	-228	-228
228	228	76	-76	-228	-228
0	0	0	0	0	0
0	0	0	0	0	0



Mirror Padding 1 for Filter Size 3x3

Image

128	128	128	128	52	52	52	52
128	128	128	128	52	52	52	52
128	128	128	128	52	52	52	52
128	128	128	128	52	52	52	52
52	52	52	52	128	128	128	128
52	52	52	52	128	128	128	128
52	52	52	52	128	128	128	128
52	52	52	52	128	128	128	128

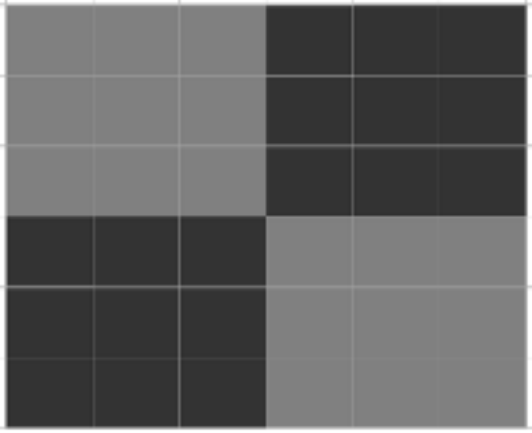
Filter

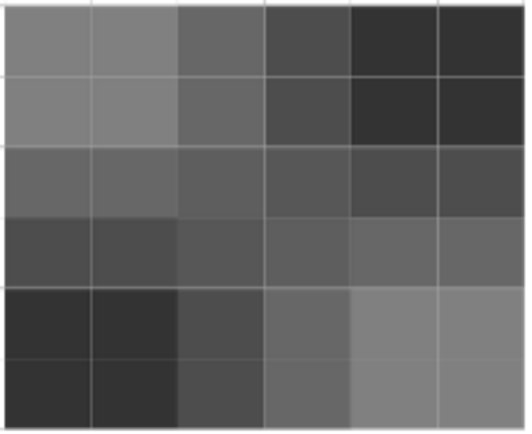
1	1	1
1	1	1
1	1	1

x1/9

Conv Output

128	128	103	77	52	52
128	128	103	77	52	52
103	103	94	86	77	77
77	77	86	94	103	103
52	52	77	103	128	128
52	52	77	103	128	128





Mirror Padding 2 for Filter Size 5x5

Image

128	128	128	128	128	52	52	52	52	52
128	128	128	128	128	52	52	52	52	52
128	128	128	128	128	52	52	52	52	52
128	128	128	128	128	52	52	52	52	52
128	128	128	128	128	52	52	52	52	52
52	52	52	52	52	128	128	128	128	128
52	52	52	52	52	128	128	128	128	128
52	52	52	52	52	128	128	128	128	128
52	52	52	52	52	128	128	128	128	128
52	52	52	52	52	128	128	128	128	128

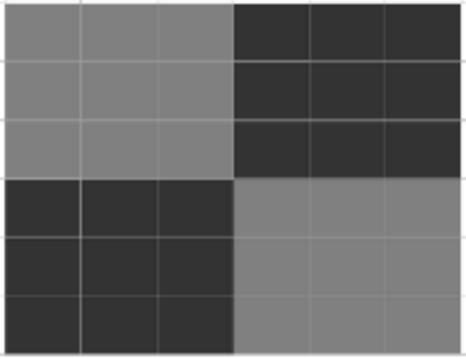
Filter

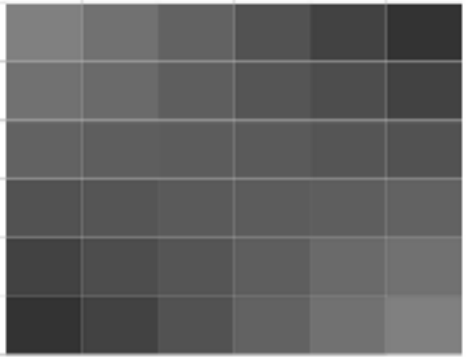
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

x1/25

Conv Output

128	128	109	90	71	52
128	128	109	90	71	52
109	109	100	90	81	71
90	90	90	90	90	90
71	71	81	90	100	109
52	52	71	90	109	128





Stride 2, Padding 2

Image

128	128	128	128	128	52	52	52	52	52
128	128	128	128	128	52	52	52	52	52
128	128	128	128	128	52	52	52	52	52
128	128	128	128	128	52	52	52	52	52
128	128	128	128	128	52	52	52	52	52
52	52	52	52	52	128	128	128	128	128
52	52	52	52	52	128	128	128	128	128
52	52	52	52	52	128	128	128	128	128
52	52	52	52	52	128	128	128	128	128
52	52	52	52	52	128	128	128	128	128

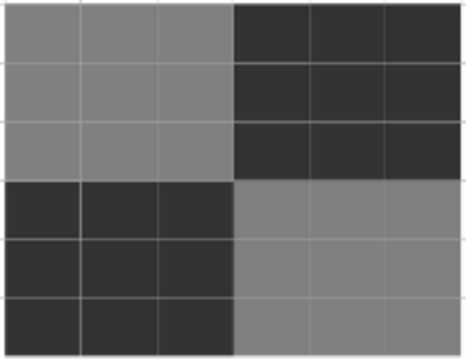
Filter

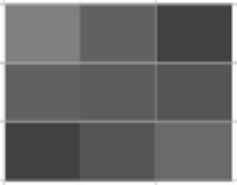
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

x1/25

Conv Output

128	98	67			
98	92	85			
67	85	104			

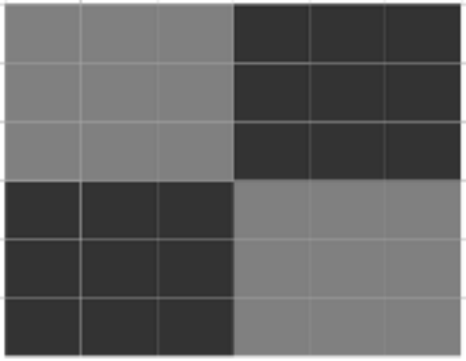




Stride 2, Padding 2

Image

128	128	128	128	128	52	52	52	52	52
128	128	128	128	128	52	52	52	52	52
128	128	128	128	128	52	52	52	52	52
128	128	128	128	128	52	52	52	52	52
128	128	128	128	128	52	52	52	52	52
52	52	52	52	52	128	128	128	128	128
52	52	52	52	52	128	128	128	128	128
52	52	52	52	52	128	128	128	128	128
52	52	52	52	52	128	128	128	128	128
52	52	52	52	52	128	128	128	128	128



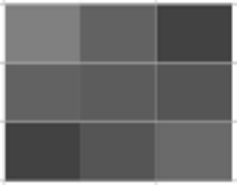
Filter

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

x1/25

Conv Output

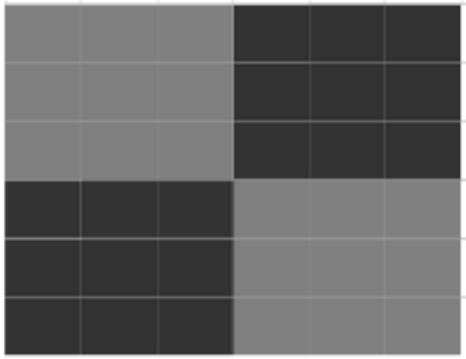
128	98	67			
98	92	85			
67	85	104			



Stride = 2, Padding 0

Image

128	128	128	52	52	52
128	128	128	52	52	52
128	128	128	52	52	52
52	52	52	128	128	128
52	52	52	128	128	128
52	52	52	128	128	128



Filter

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

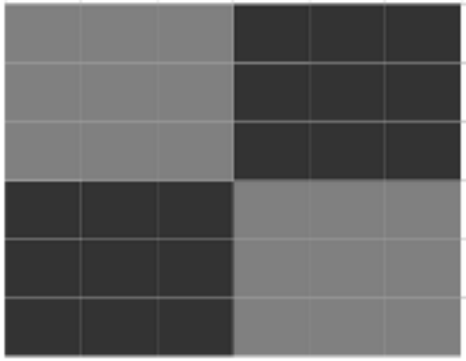
x1/25

Conv Output

Stride = 2, Padding 0

Image

128	128	128	52	52	52
128	128	128	52	52	52
128	128	128	52	52	52
52	52	52	128	128	128
52	52	52	128	128	128
52	52	52	128	128	128



Filter

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

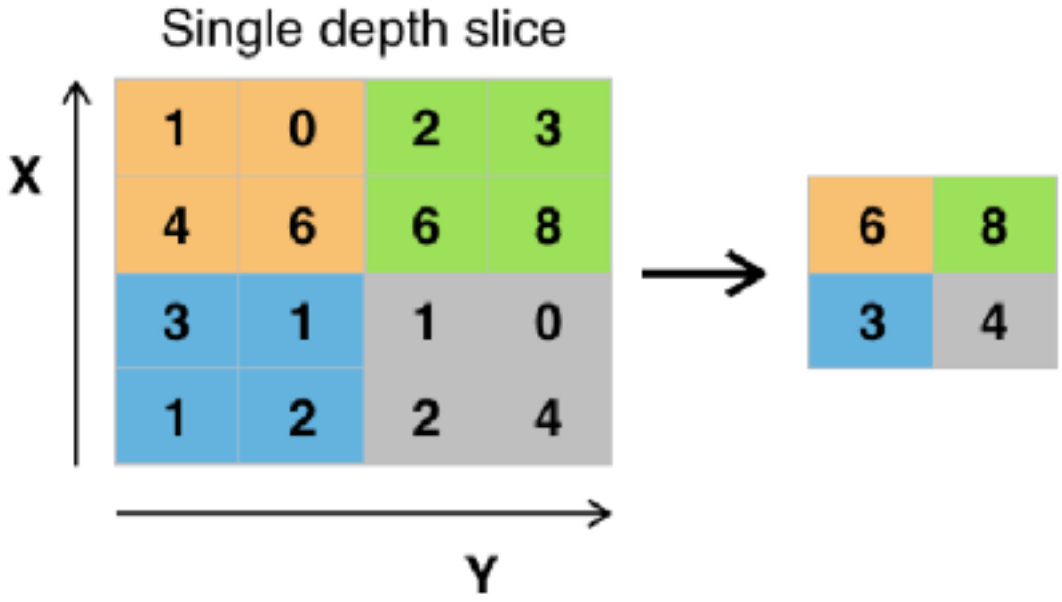
x1/25

Conv Output

92

Pooling

- Input: 4x4
- Pool size 2x2
- Stride 2
- In case pools do not overlap



Today's Agenda

- Why we do convolution?
- How to define a convolution kernel?
- Filter Banks?

Filtering Examples

Original
Cameraman

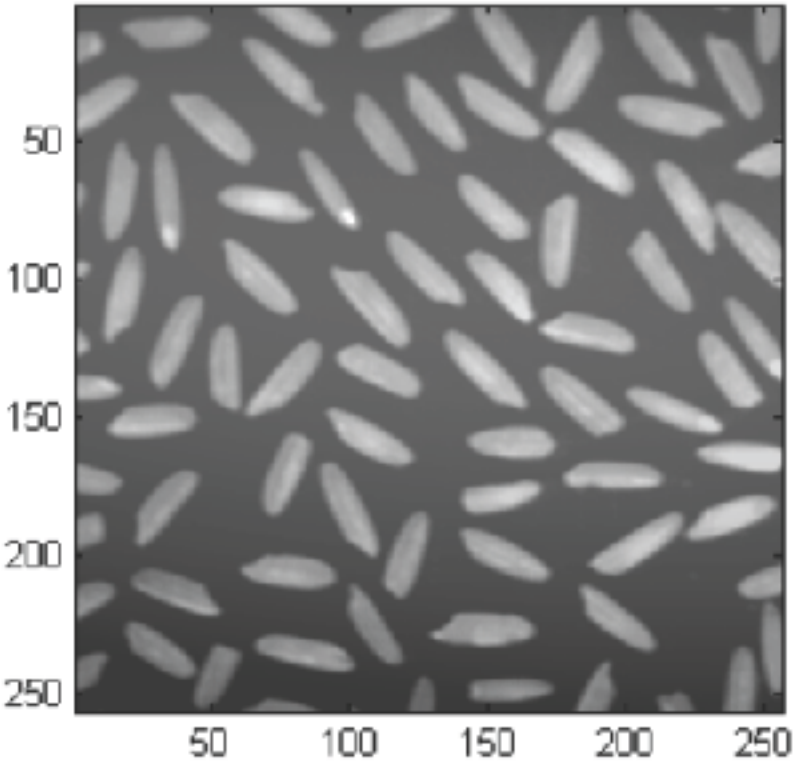
Cameraman blurred by convolution
Filter impulse response

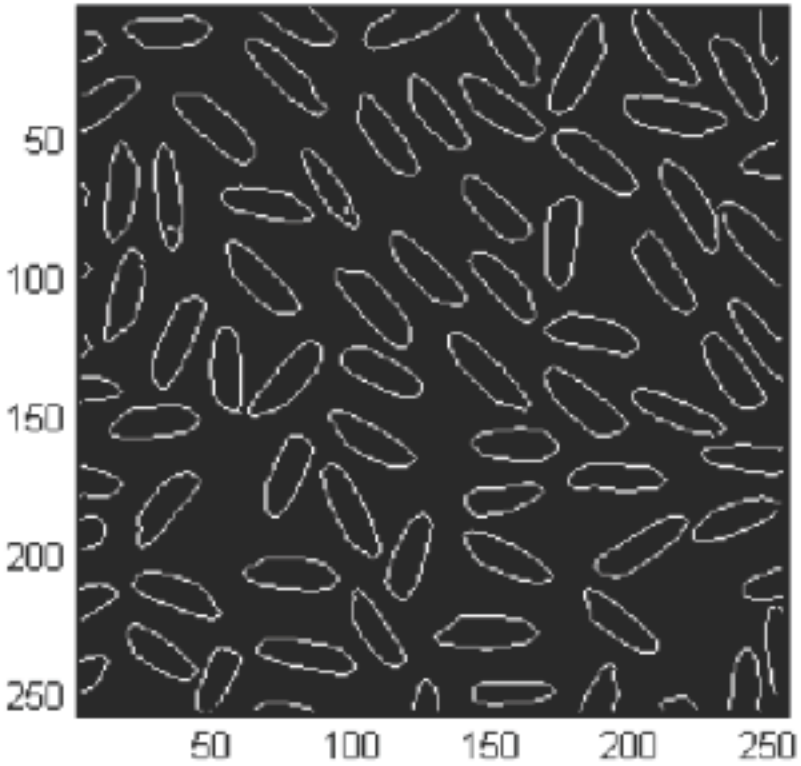
$$\frac{1}{25}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & [1] & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Noise Removal via fastNlMeansDenoising

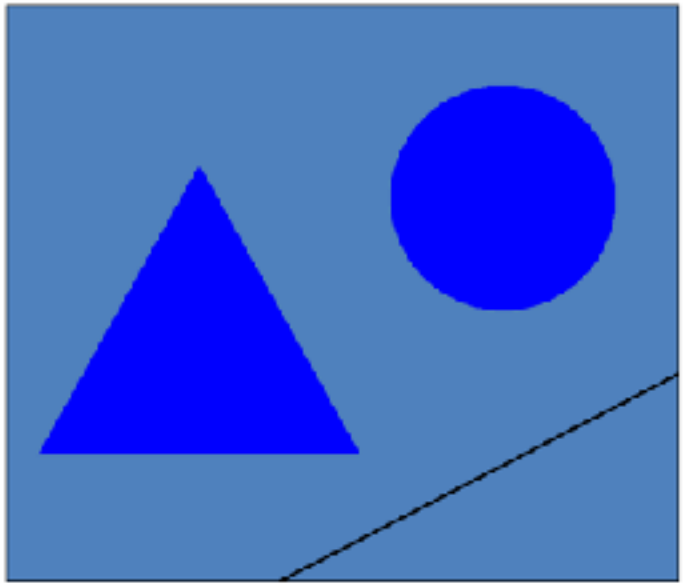
https://funvision.blogspot.com/2020/04/opencv-c-tutorial-smoothing-blur-noise.html

Finding Shapes from Edges





Finding Shapes from Edges



Finding Edges - Differentiation

- Continuous form

$$\frac{\partial f}{\partial x} = \lim_{\Delta h \to 0} \frac{f(x + \Delta h) - f(x)}{\Delta h}$$

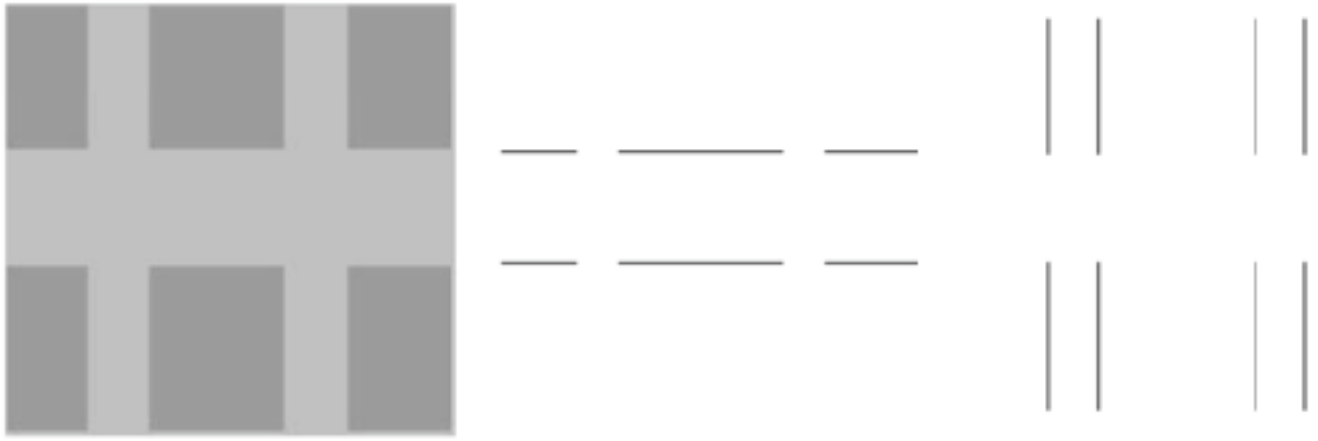
- Discrete form (put $\Delta h = 1$)

$$f_x = \frac{\partial f}{\partial x} = f(x+1) - f(x)$$

Discrete Derivatives in 2-D

$$f_x = \frac{\partial f}{\partial x} = f(x+1) - f(x) \qquad f_y = \frac{\partial f}{\partial y} = f(y+1) - f(y)$$

$$h_x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \qquad h_y = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$



synthetic image h_y output h_x output

Vertical Edge Detection

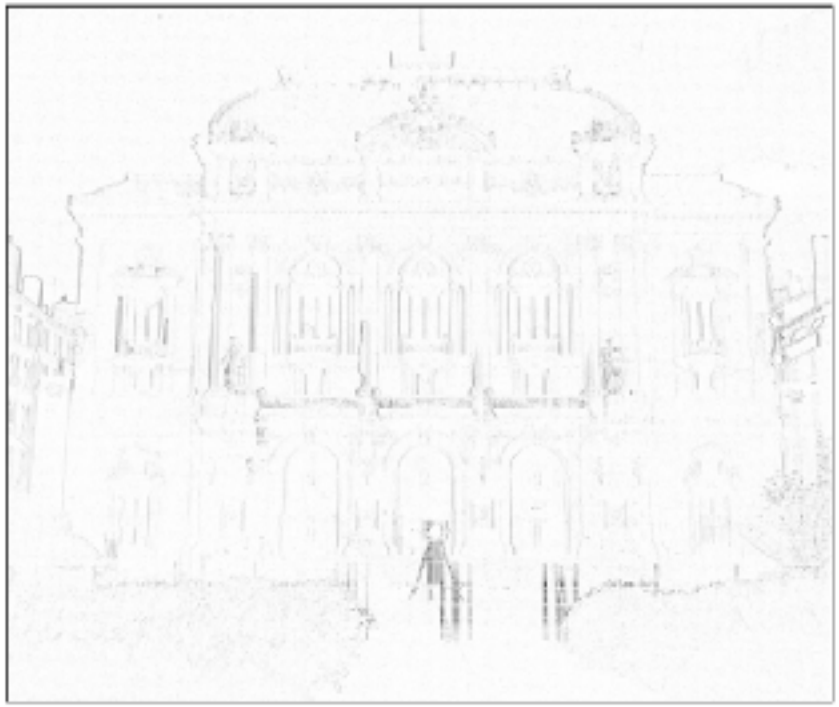
$$h_x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Original Image: Opera

Vertical Edge Detection

$$h_x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Original Image: Opera

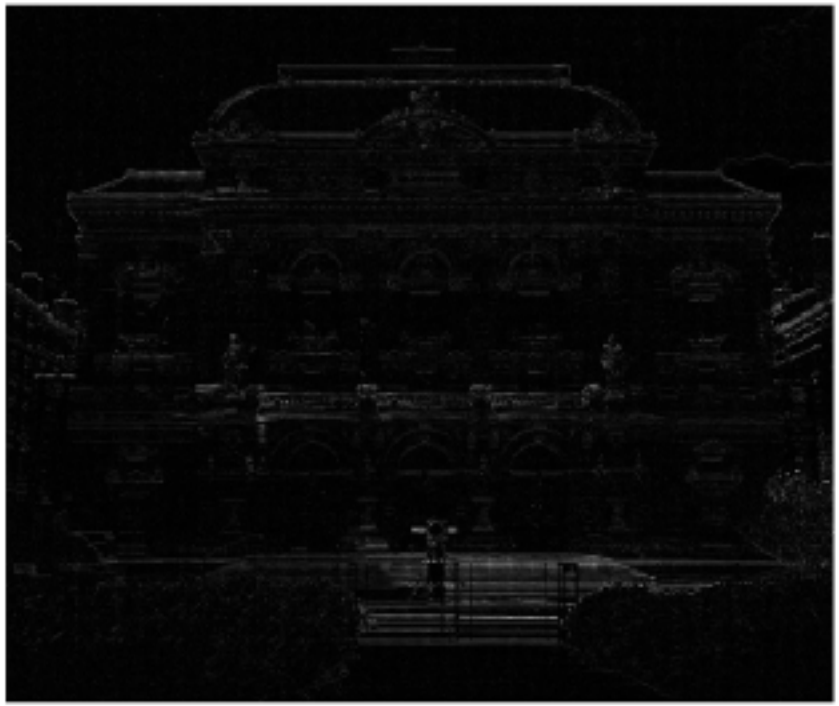


Inverted Image for Visualisation

Horizontal Edge Detection

$$h_y = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

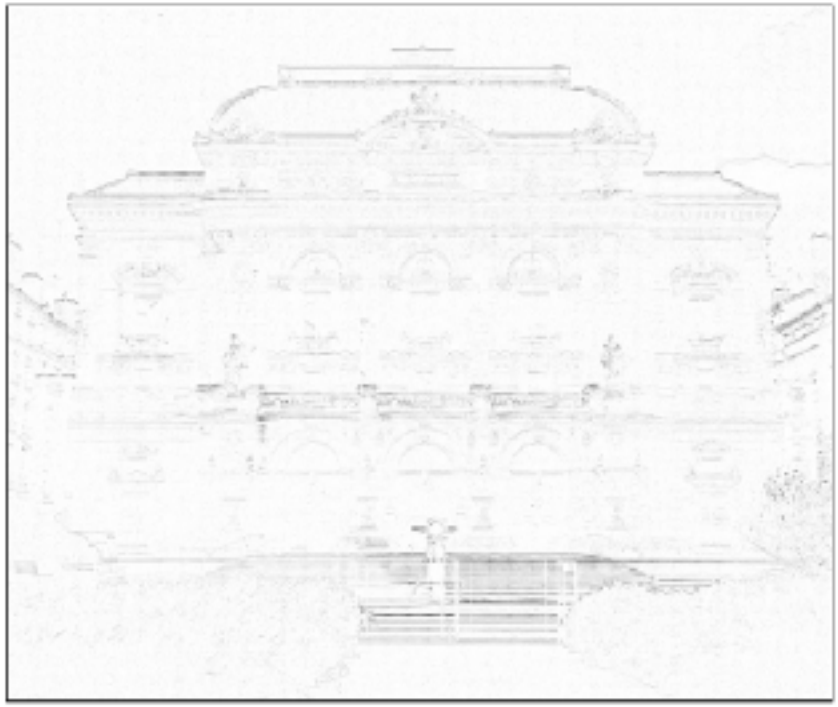
Original Image: Opera



Horizontal Edge Detection

$$h_y = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Original Image: Opera

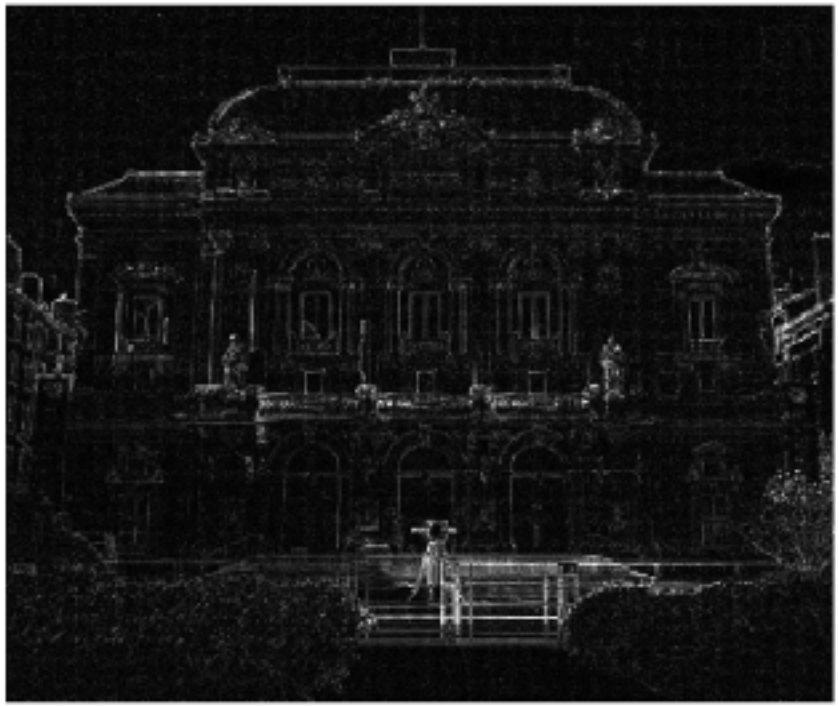


Inverted Image for Visualisation

Laplacian Filter

$$h = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

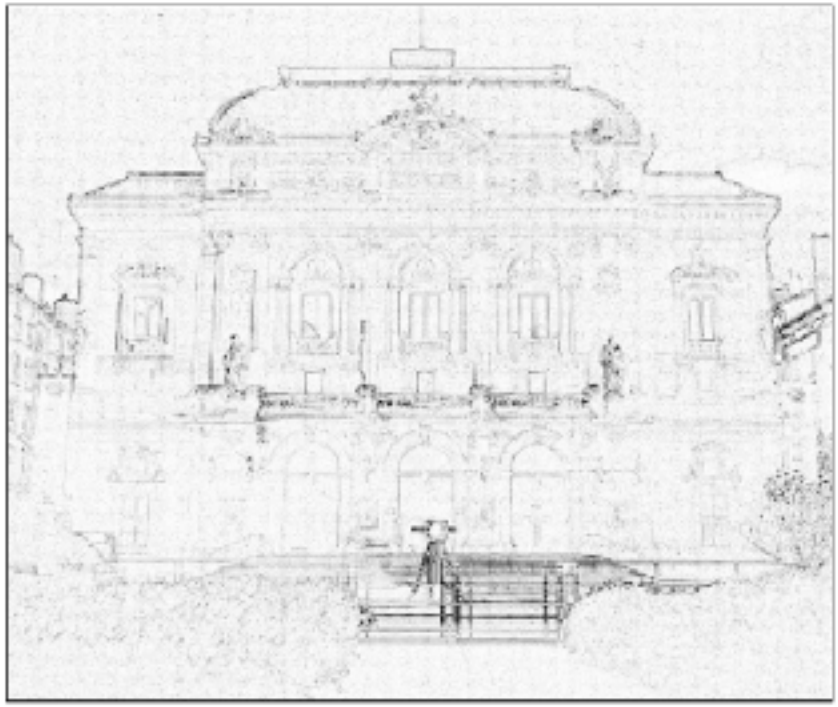
Original Image: Opera



Laplacian Filter

$$h = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Original Image: Opera



Inverted Image for Visualisation

Sobel, Prewit & Roberts Operators

Sobel

$$h_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$$h_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Prewit

$$h_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

$$h_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Roberts

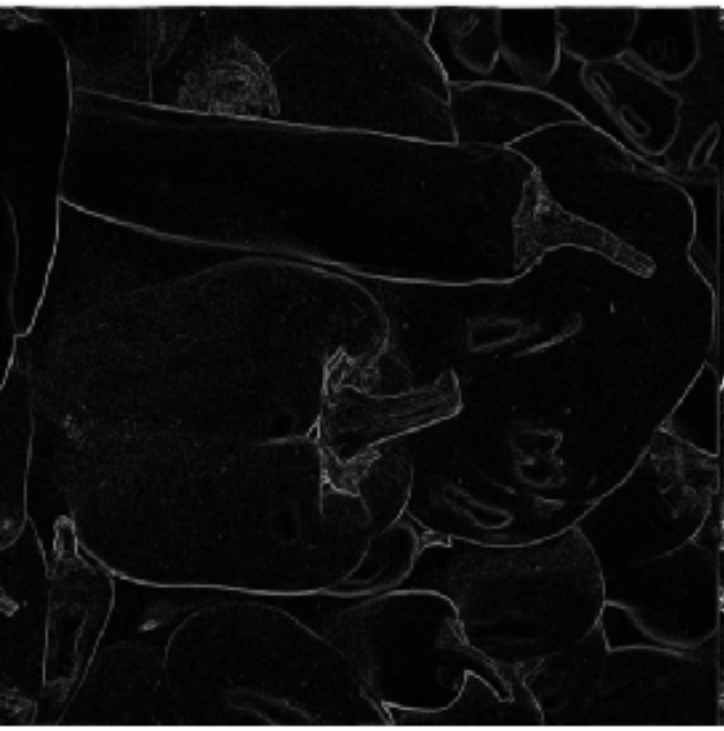
$$h_x = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

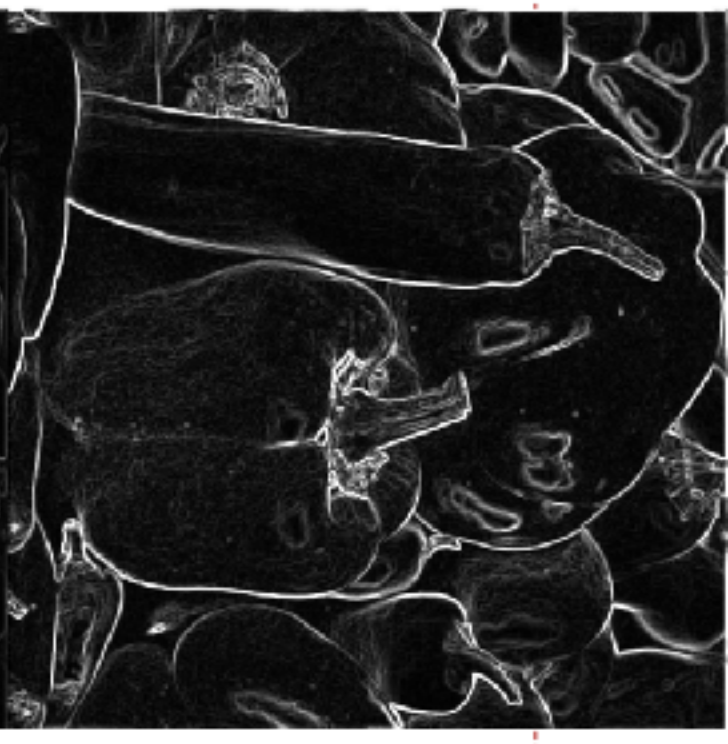
$$h_y = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Sobel

Sobel

Original

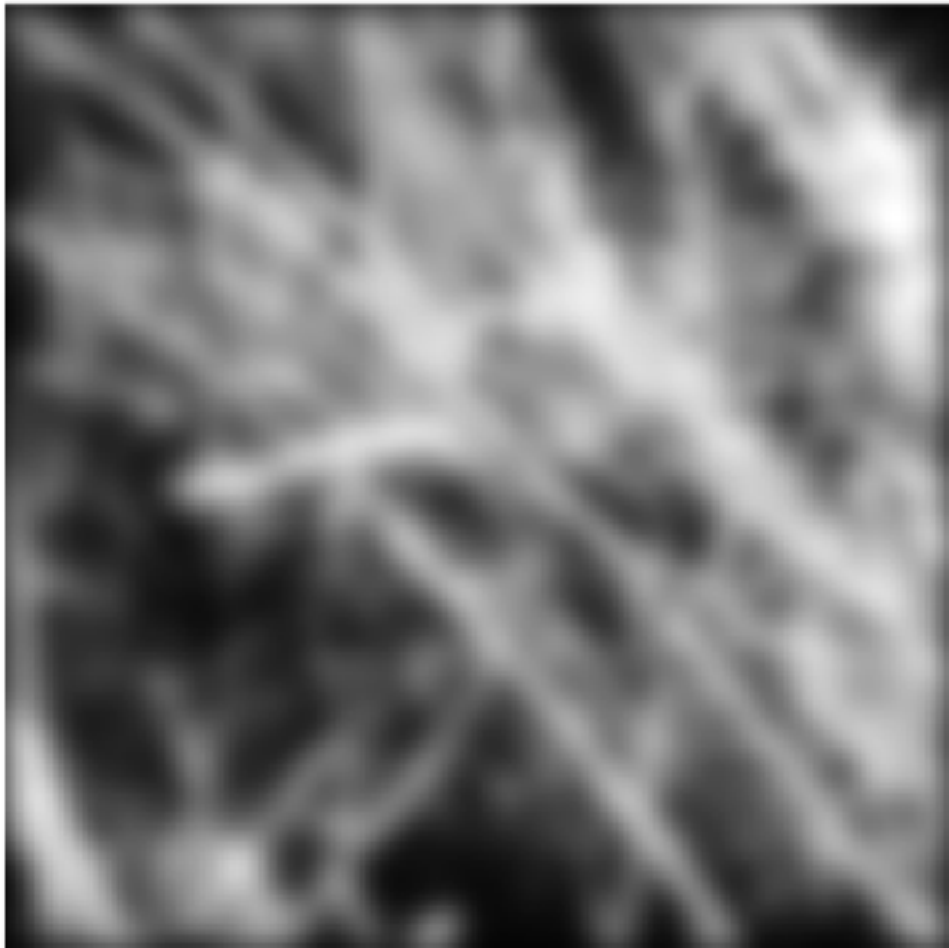




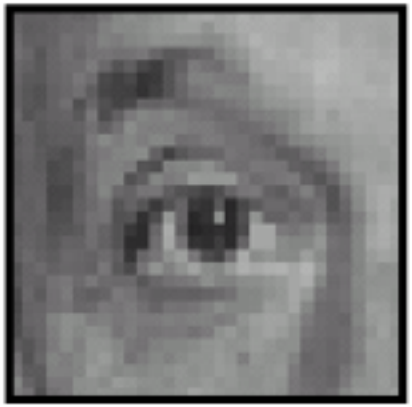
Roberts

Prewit

Smoothing with Gaussian filter



Practice with linear filters



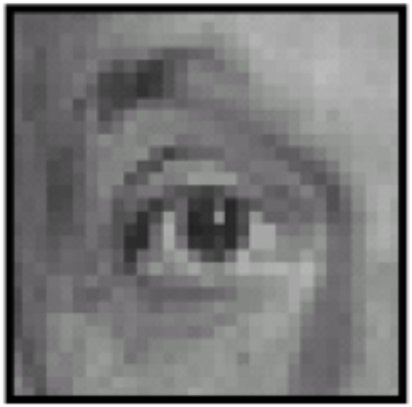
Original

0	0	0
0	1	0
0	0	0

?

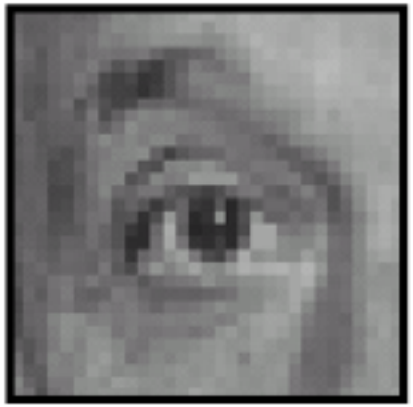
Source: D. Lowe

Practice with linear filters



Original

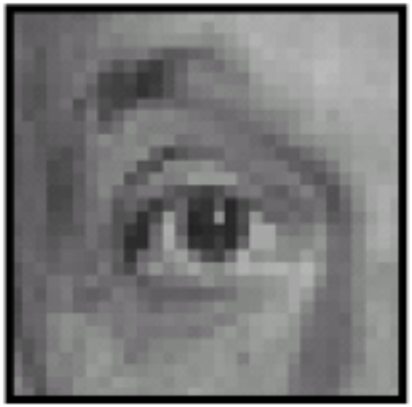
0	0	0
0	1	0
0	0	0



Filtered
(no change)

Source: D. Lowe

Practice with linear filters



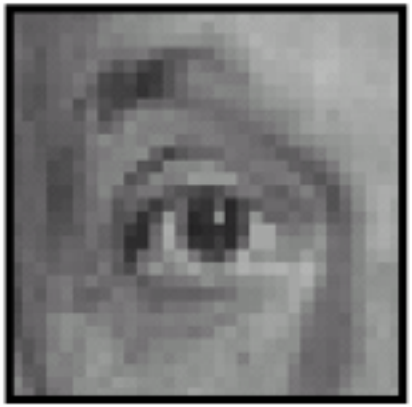
Original

0	0	0
0	0	1
0	0	0

?

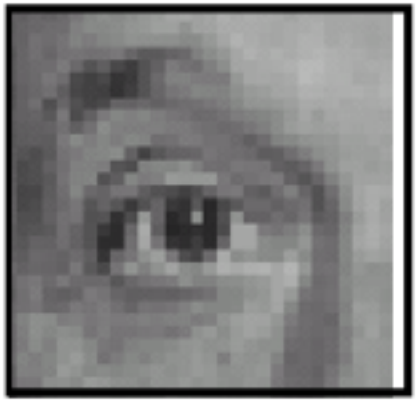
Source: D. Lowe

Practice with linear filters



Original

0	0	0
0	0	1
0	0	0



Shifted left
By 1 pixel

Source: D. Lowe

Practice with linear filters

Original

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

(Note that filter sums to 1)

Source: D. Lowe

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

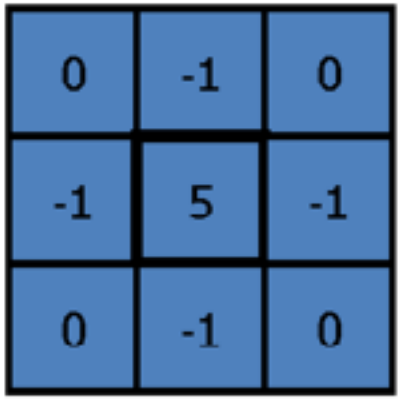
- Accentuates differences with local average

Source: D. Lowe

Sharpening

- Derivative operators can highlight gray-level discontinuities
- The result of the derivative operator can be superimposed on the original image

$$g(x, y) = f(x, y) + h(x, y)$$

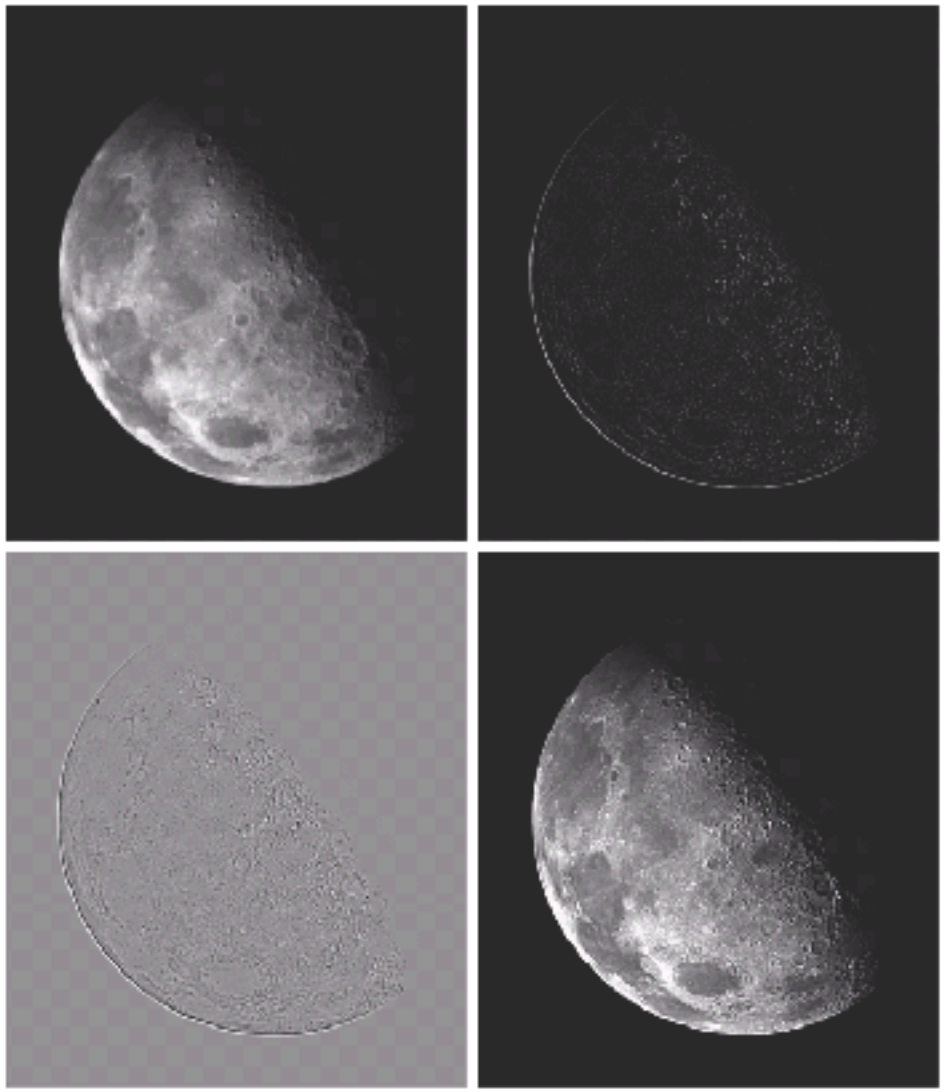


$$f = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad h = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

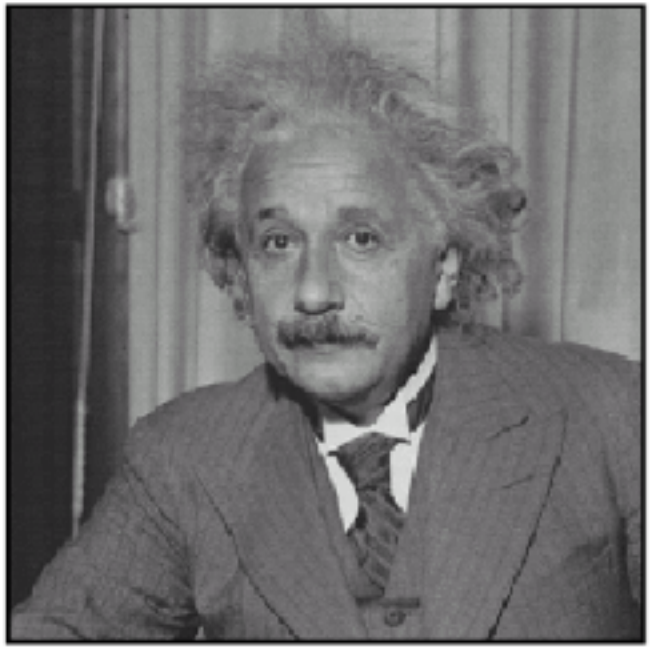
Sharpening Example

a b
c d

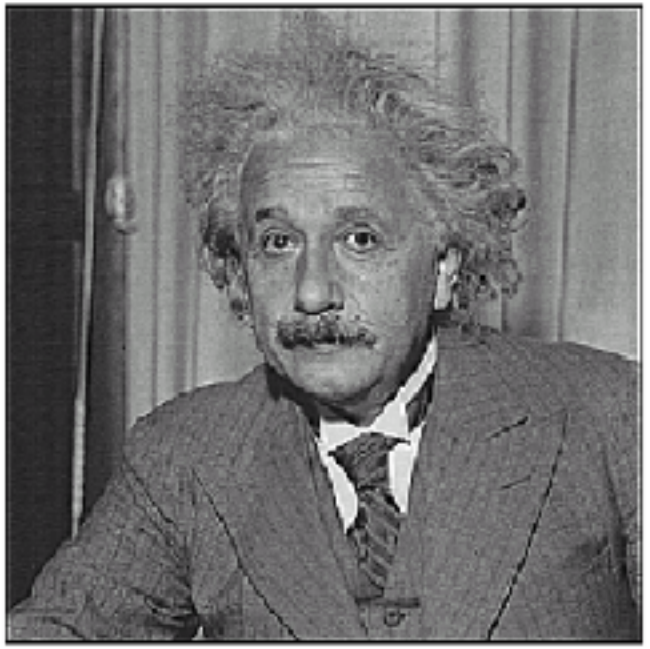
FIGURE 3.40
(a) Image of the North Pole of the moon.
(b) Laplacian-filtered image.
(c) Laplacian image scaled for display purposes.
(d) Image enhanced by using Eq. (3.7-5).
(Original image courtesy of NASA.)



Sharpening



before



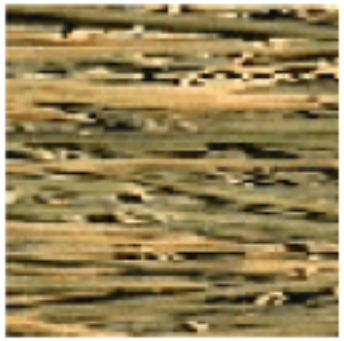
after

Source: D. Lowe

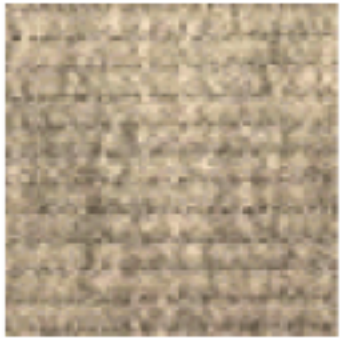
Image Classification using Filter Bank

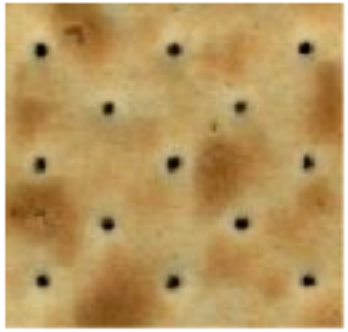
http://animals.nationalgeographic.com/





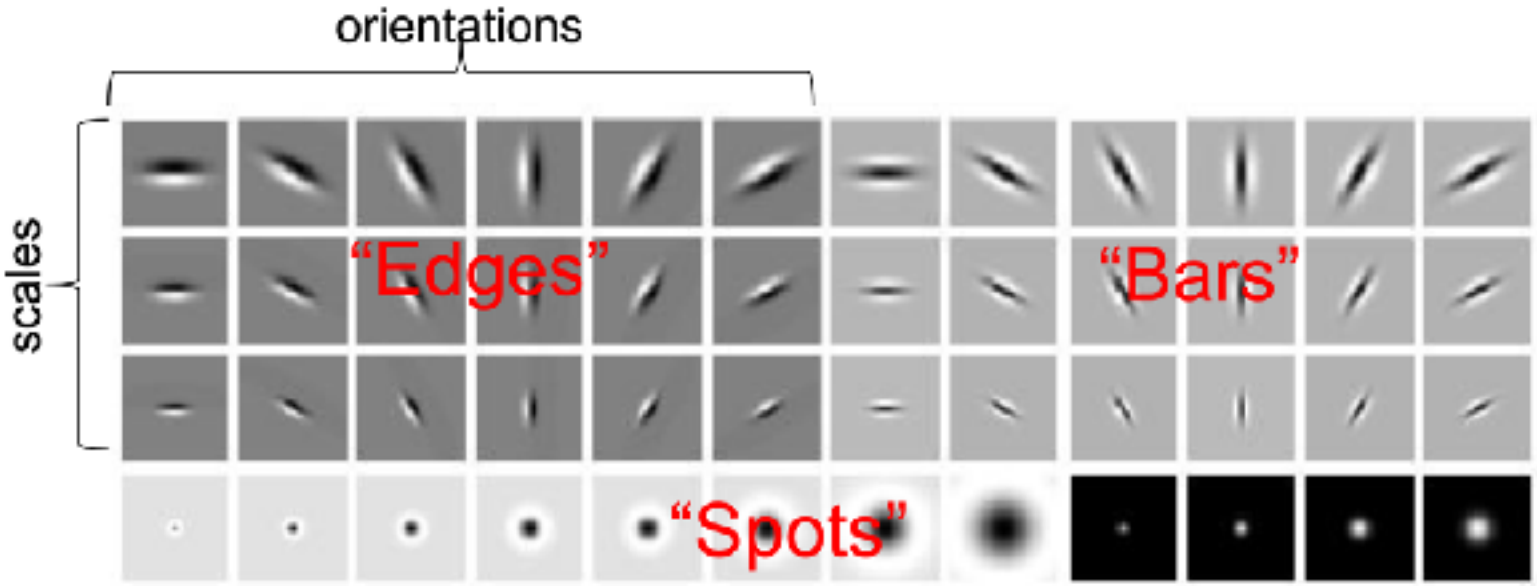




Filter banks

- 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

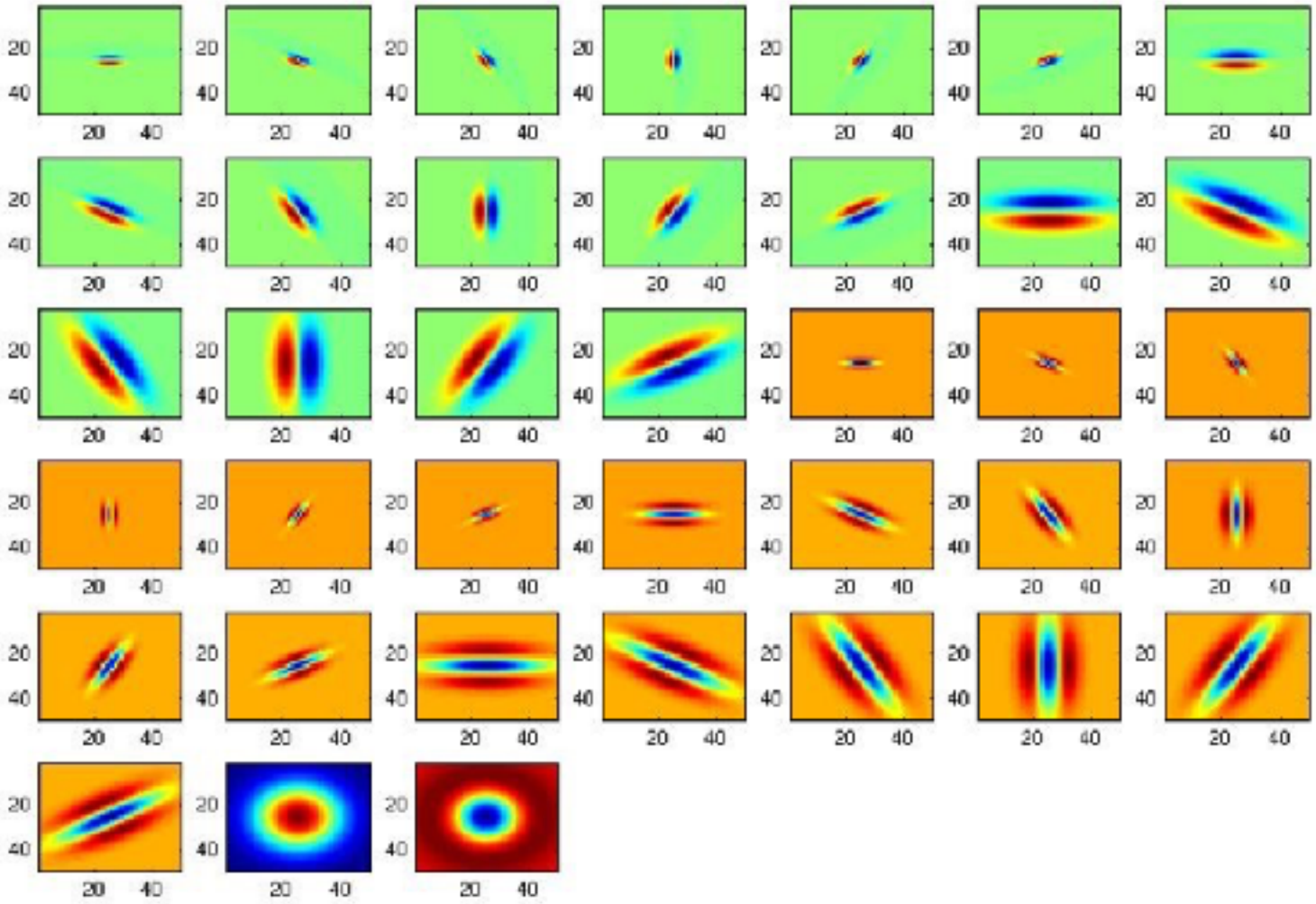
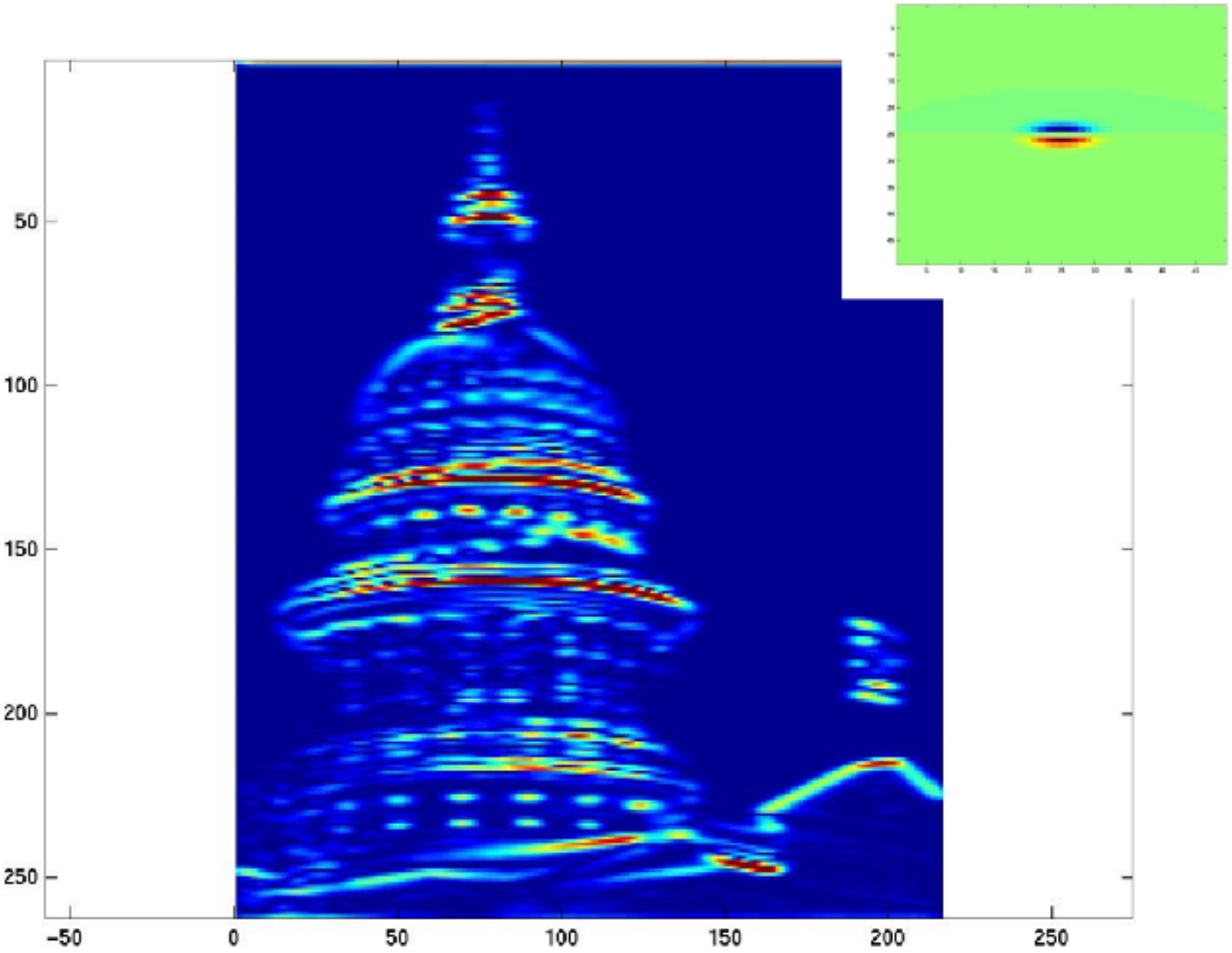
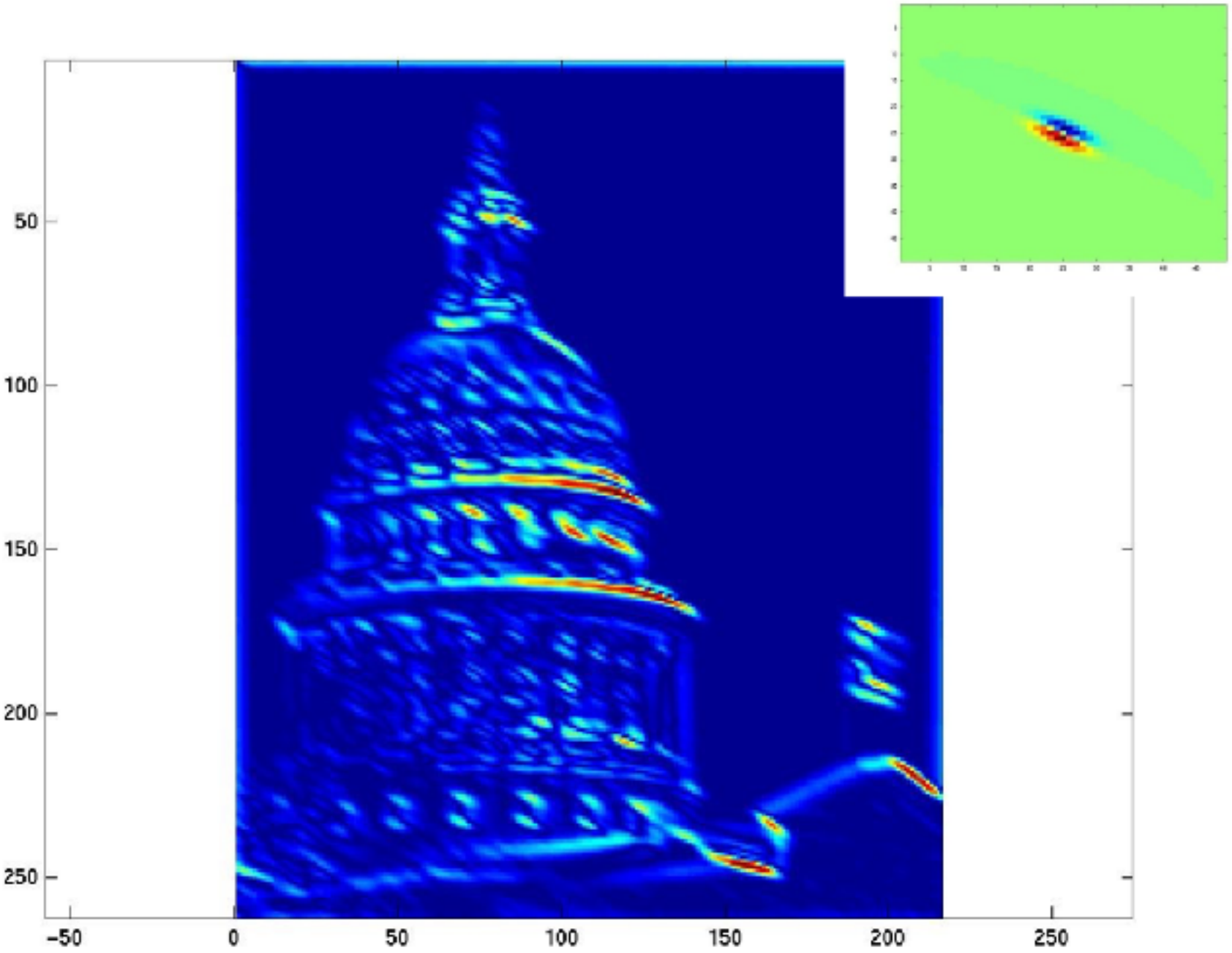
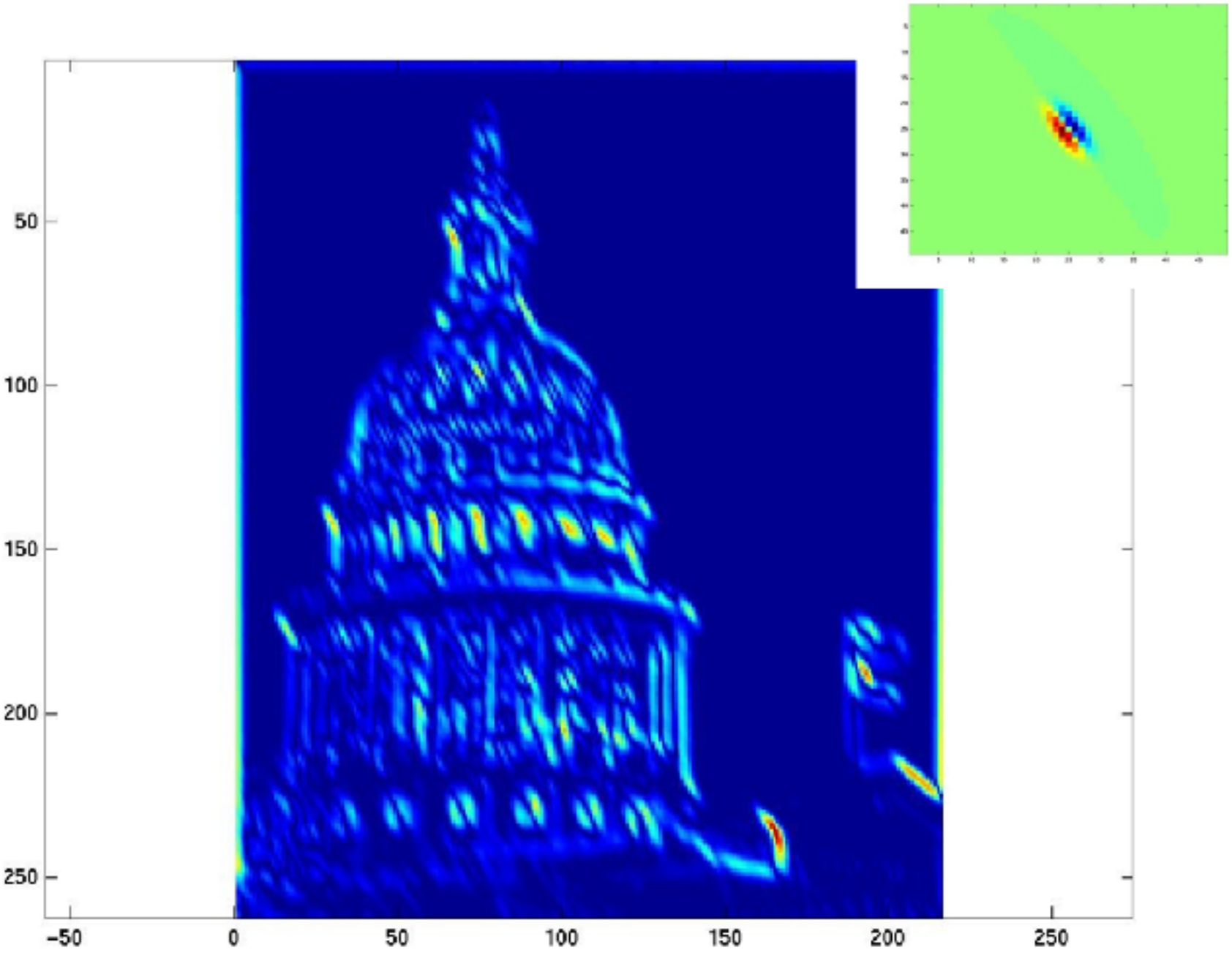


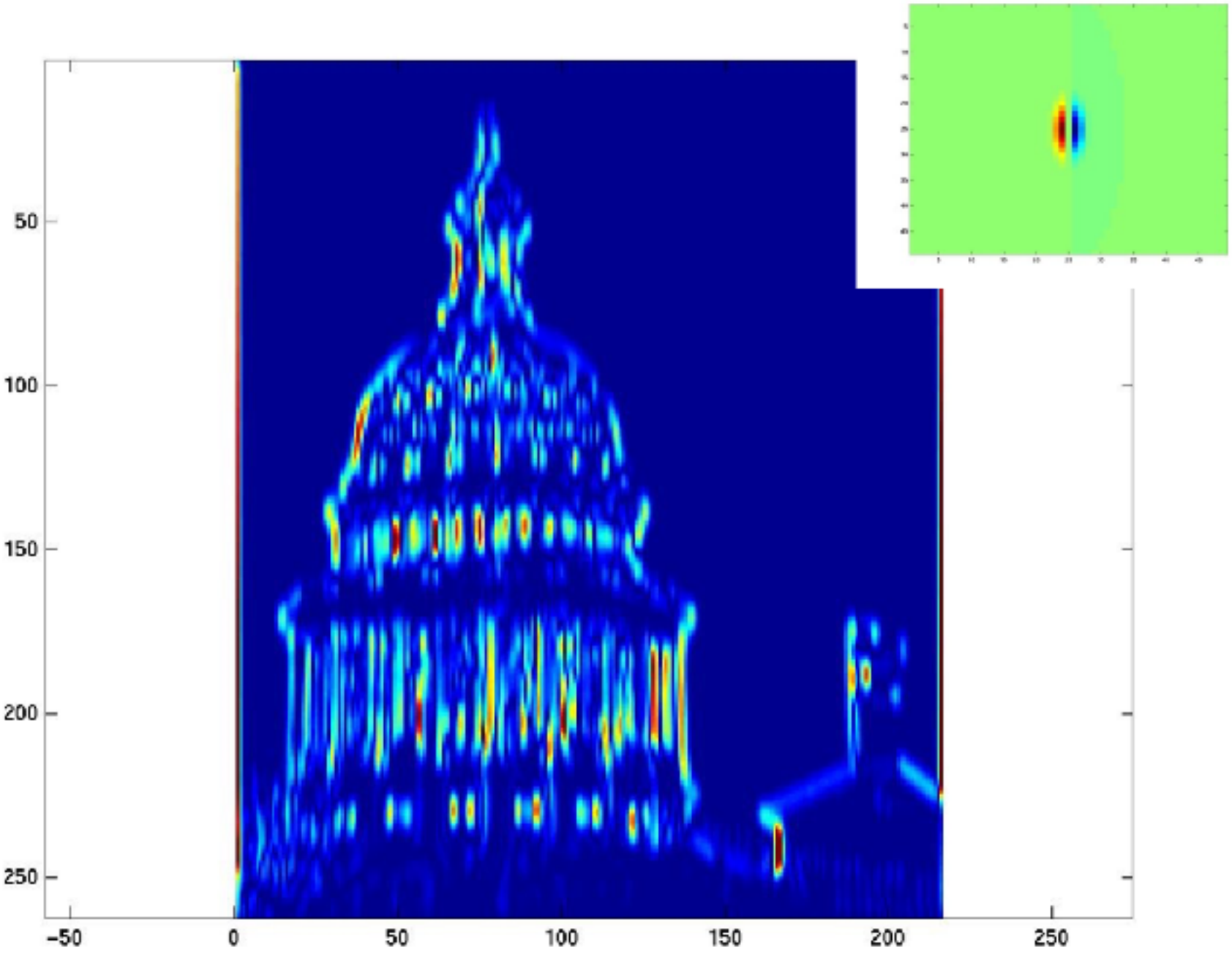
Image from http://www.texasexplorer.com/austincap2.jpg

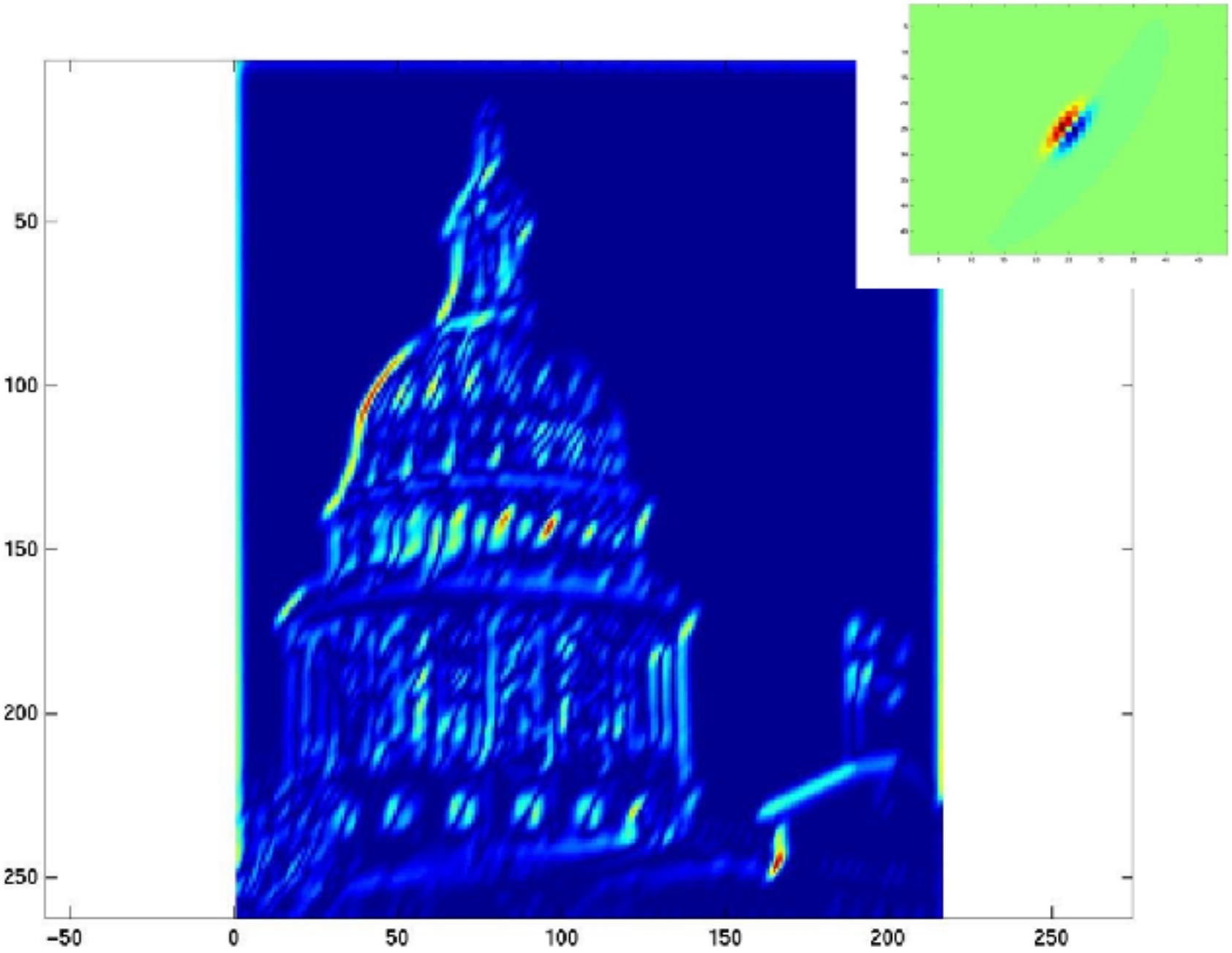
Kristen Grauman

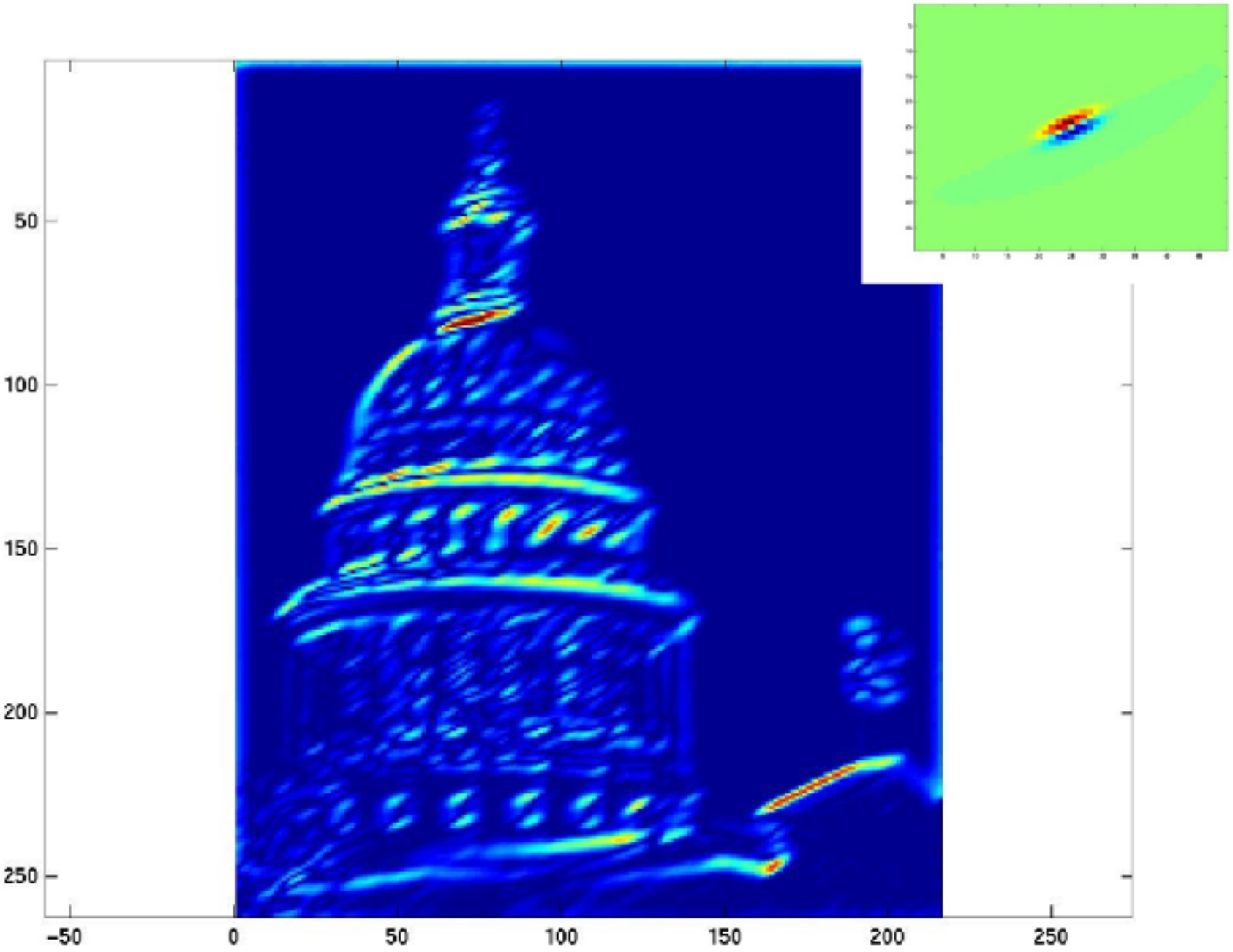

50
100
150
200
250
−50
0
50
100
150
200
250

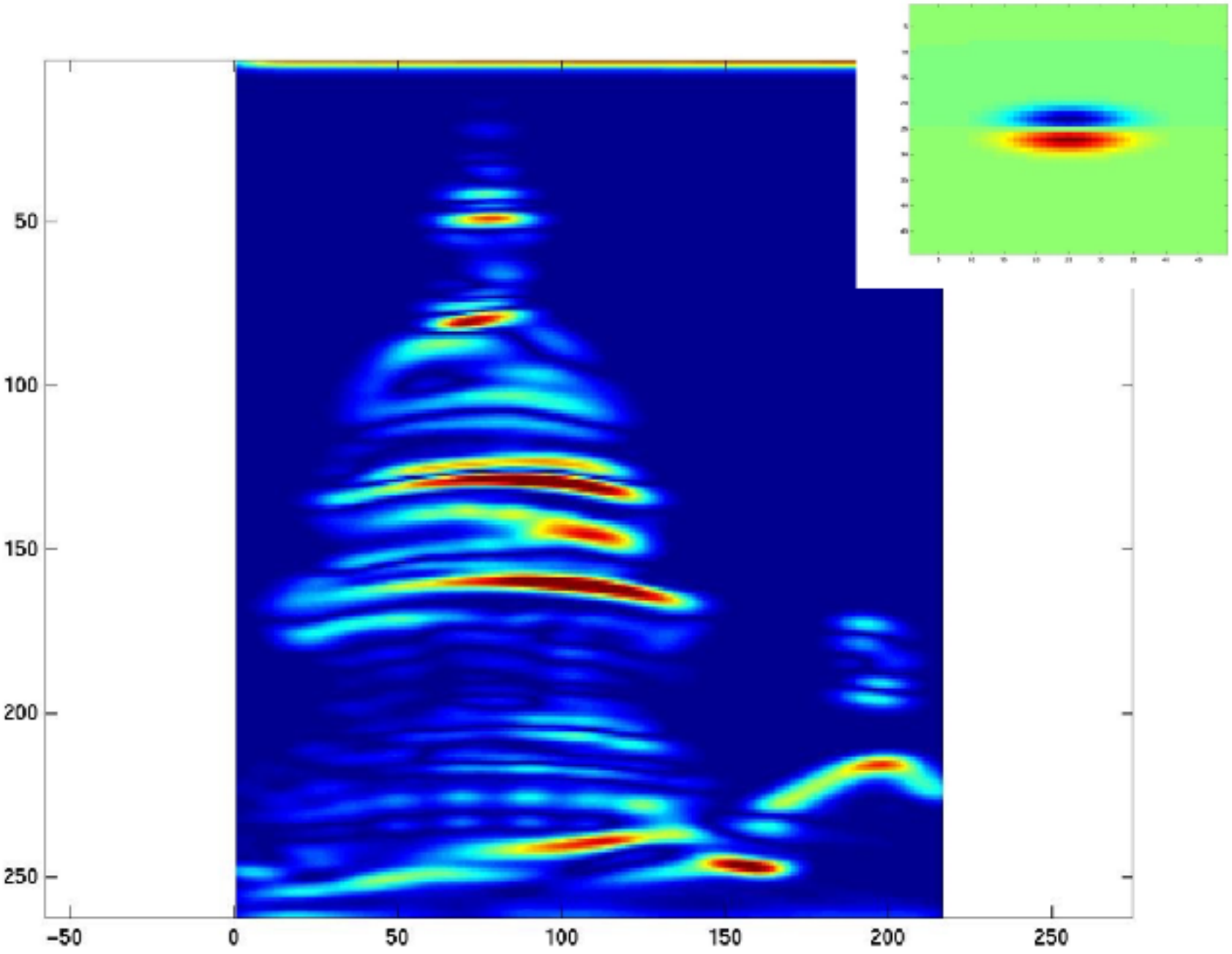

50
100
150
200
250
−50
0
50
100
150
200
250

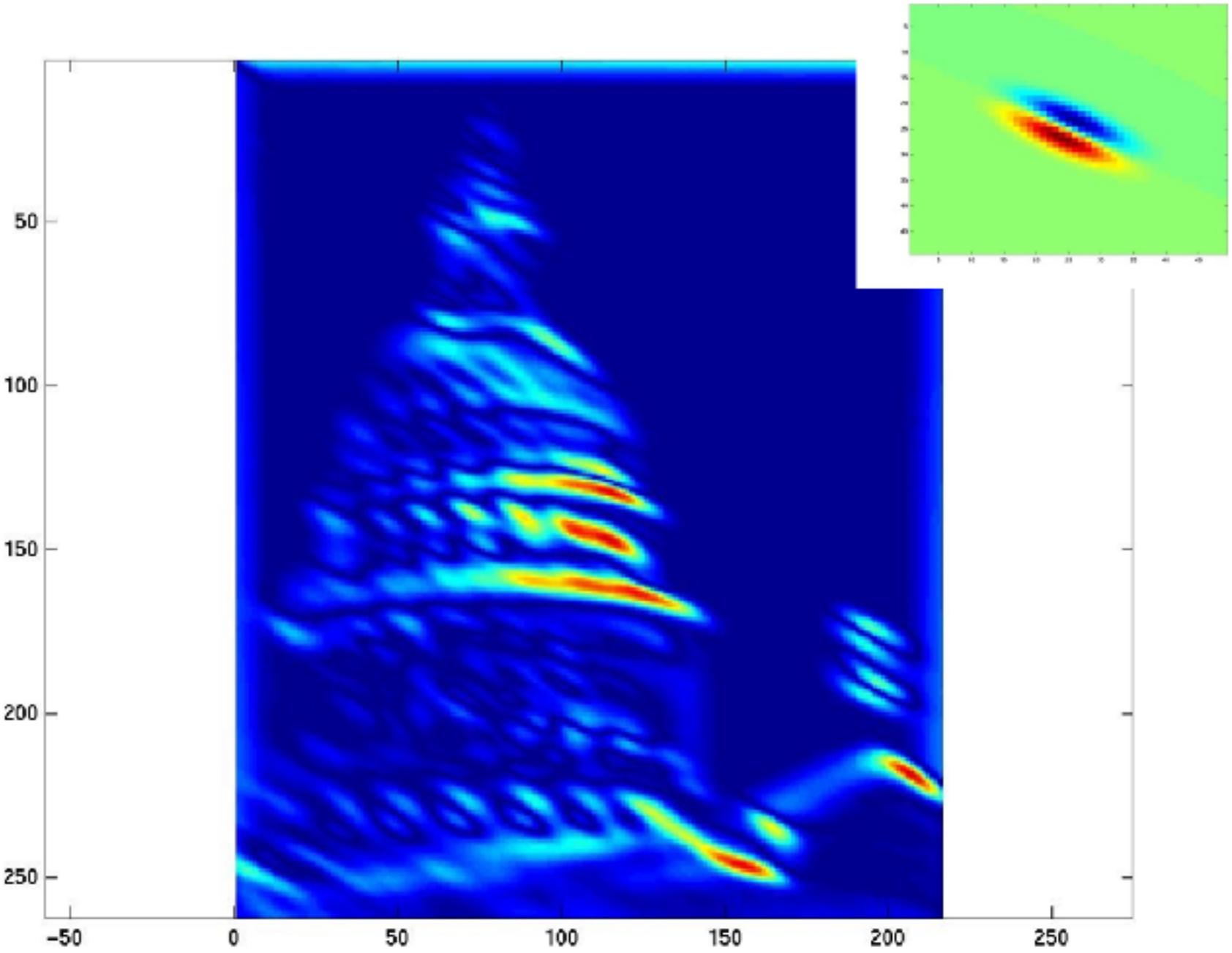

50
100
150
200
250
−50
0
50
100
150
200
250

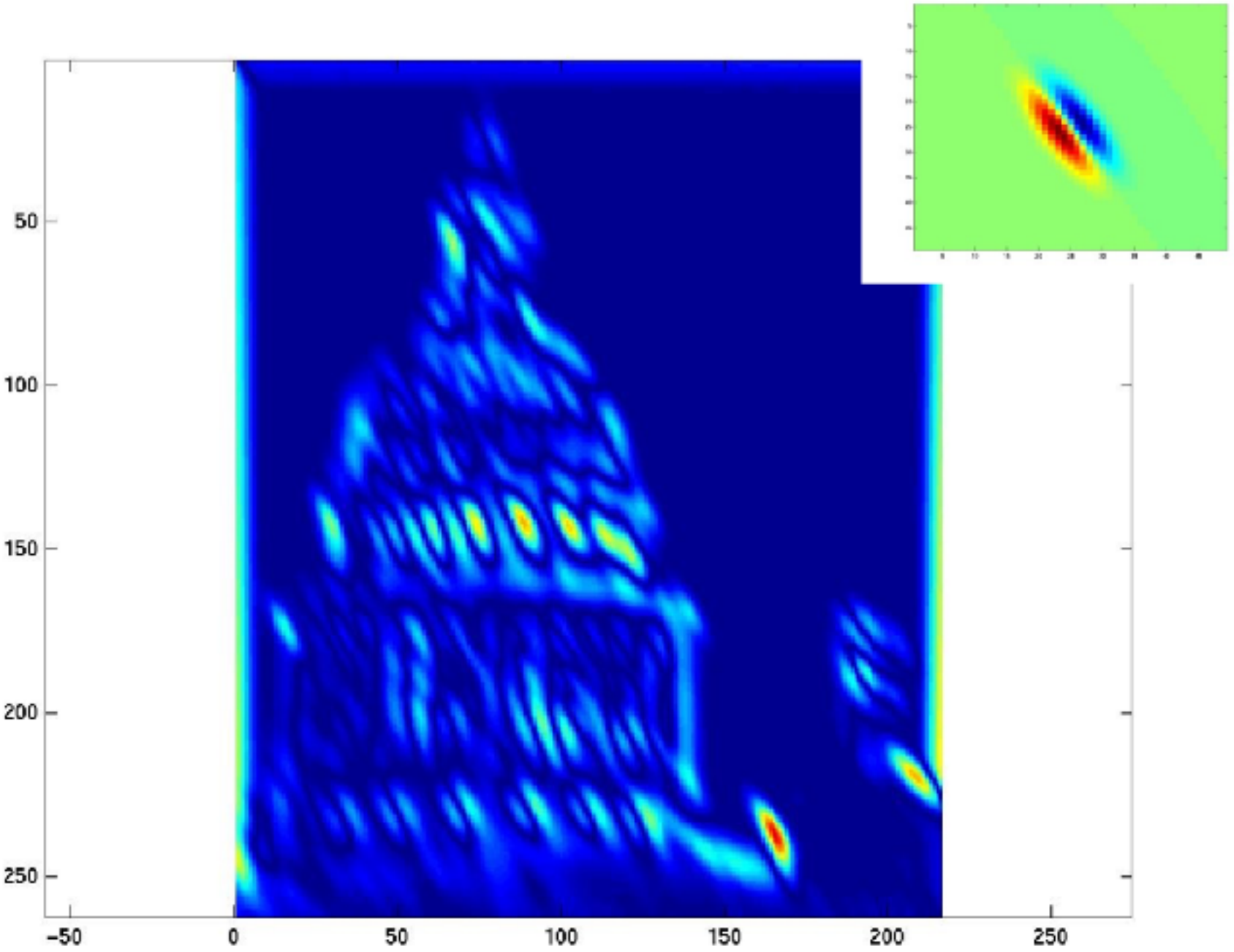

50
100
150
200
250
−50
0
50
100
150
200
250

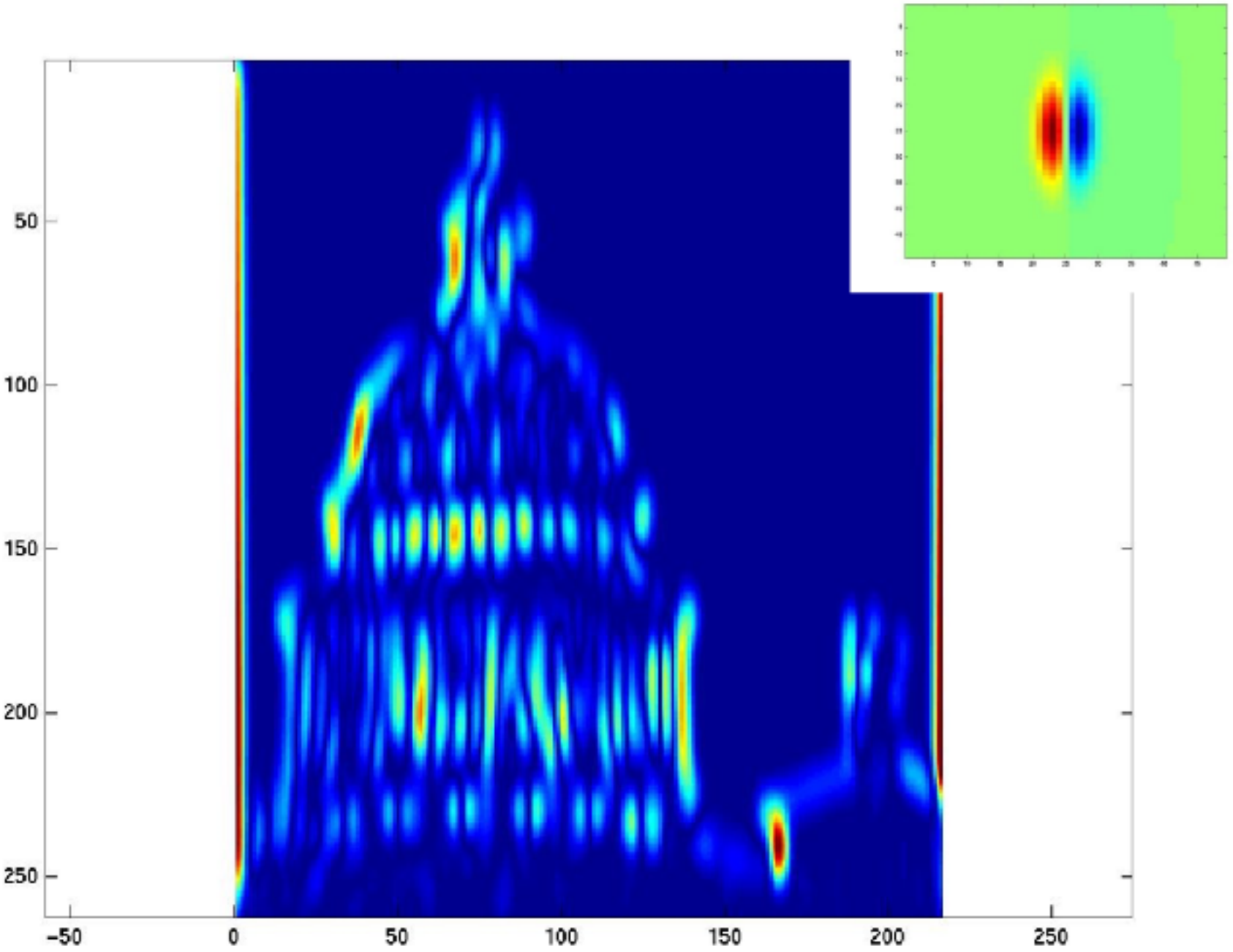

50
100
150
200
250
−50
0
50
100
150
200
250

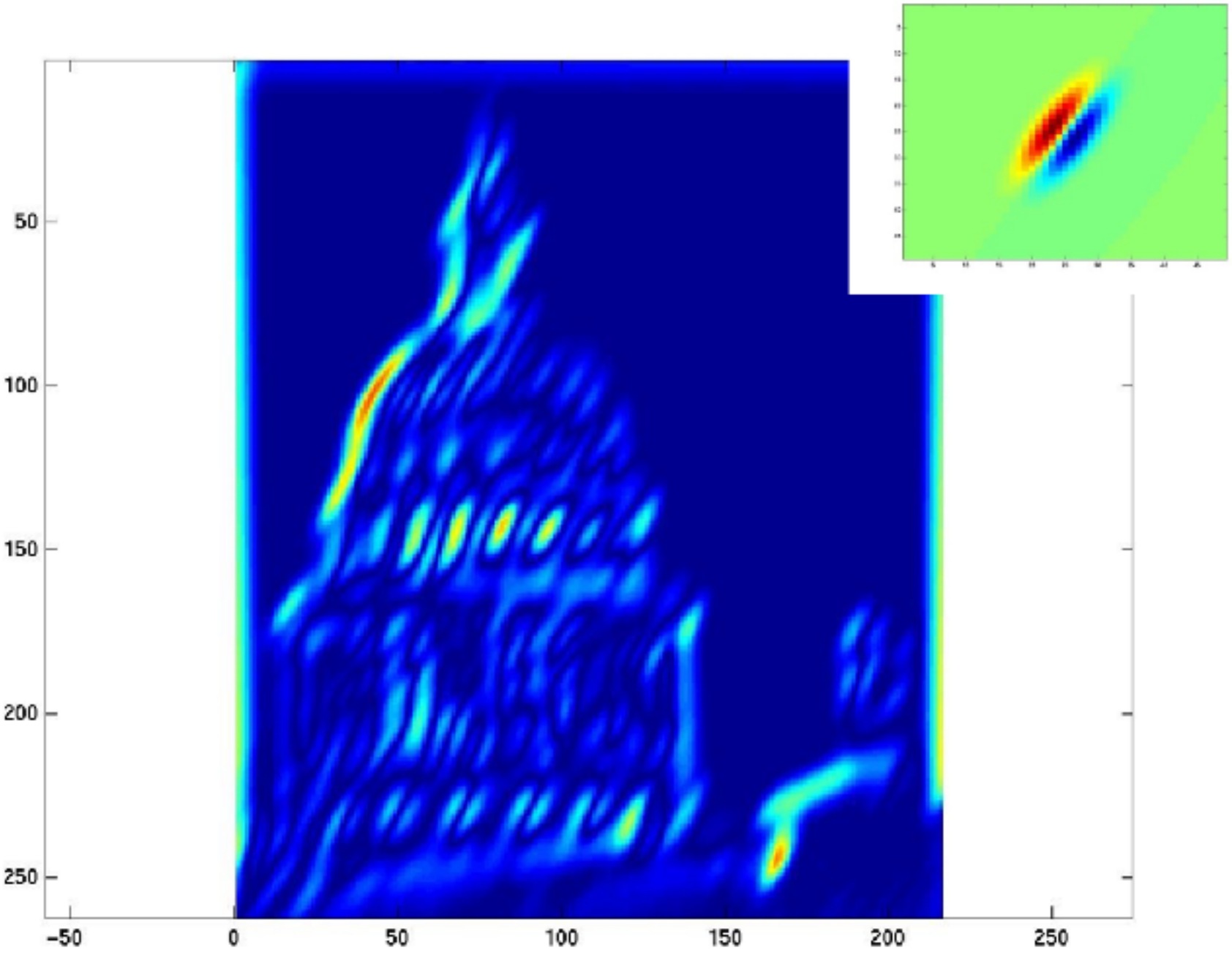

50
100
150
200
250
−50
0
50
100
150
200
250

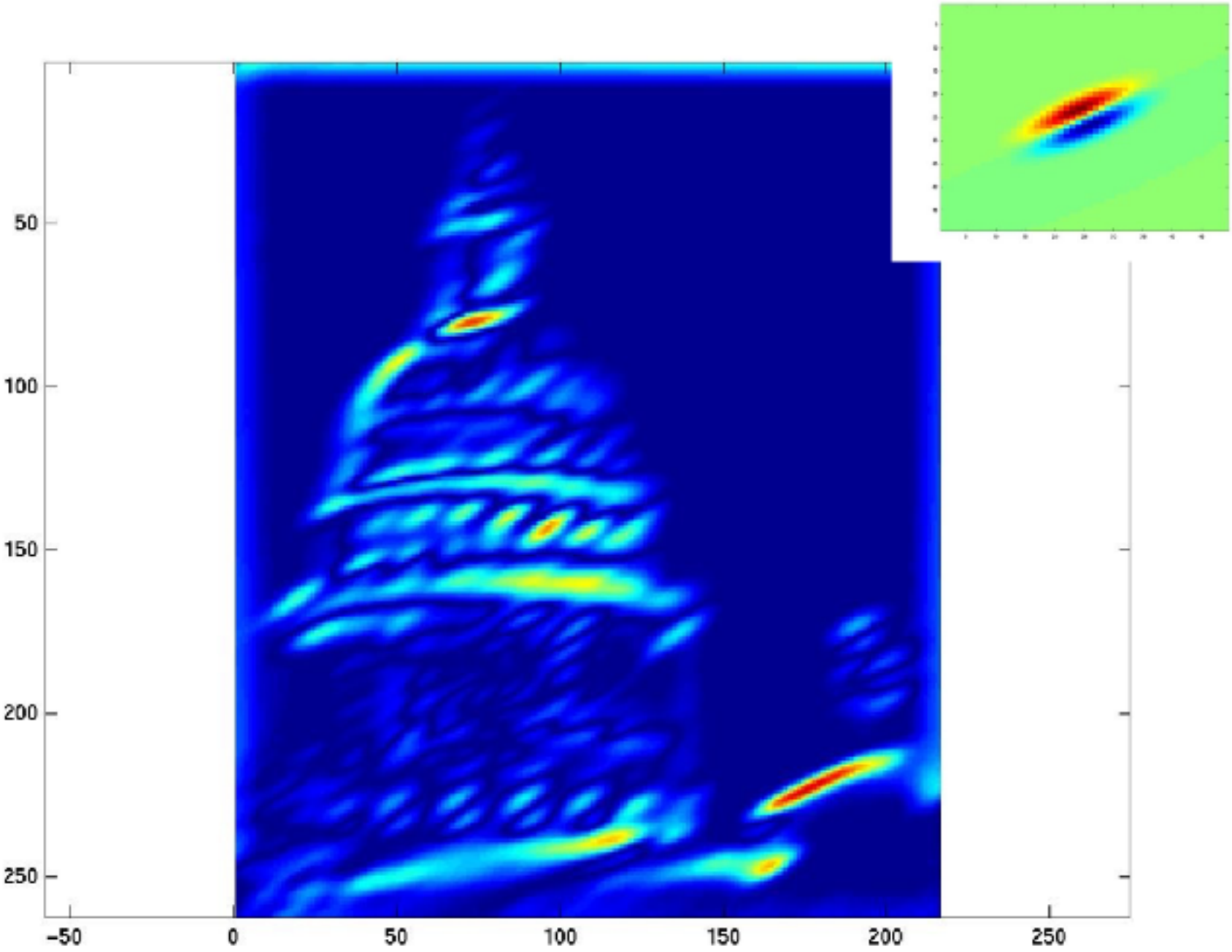

50
100
150
200
250
−50
0
50
100
150
200
250

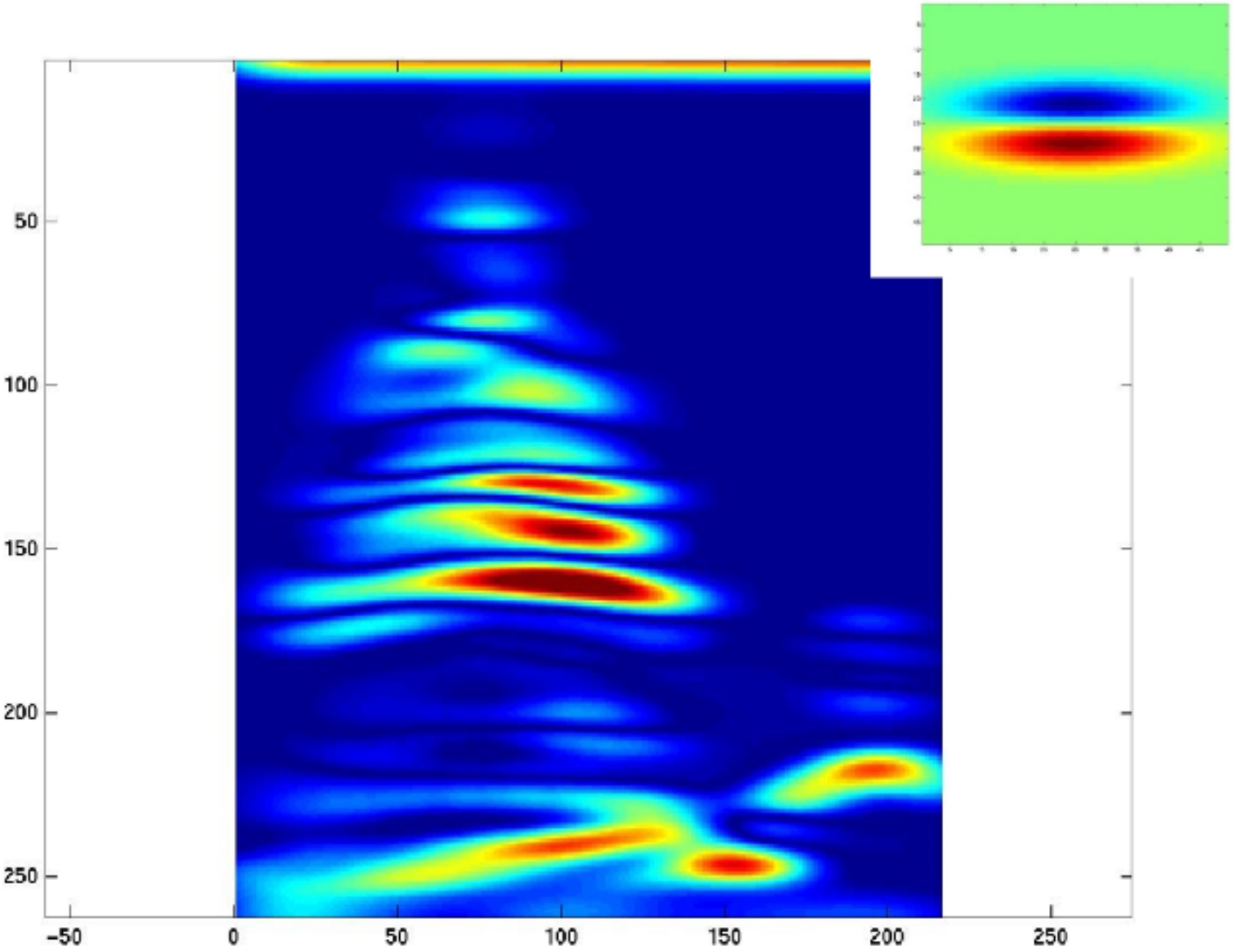

50
100
150
200
250
−50
0
50
100
150
200
250

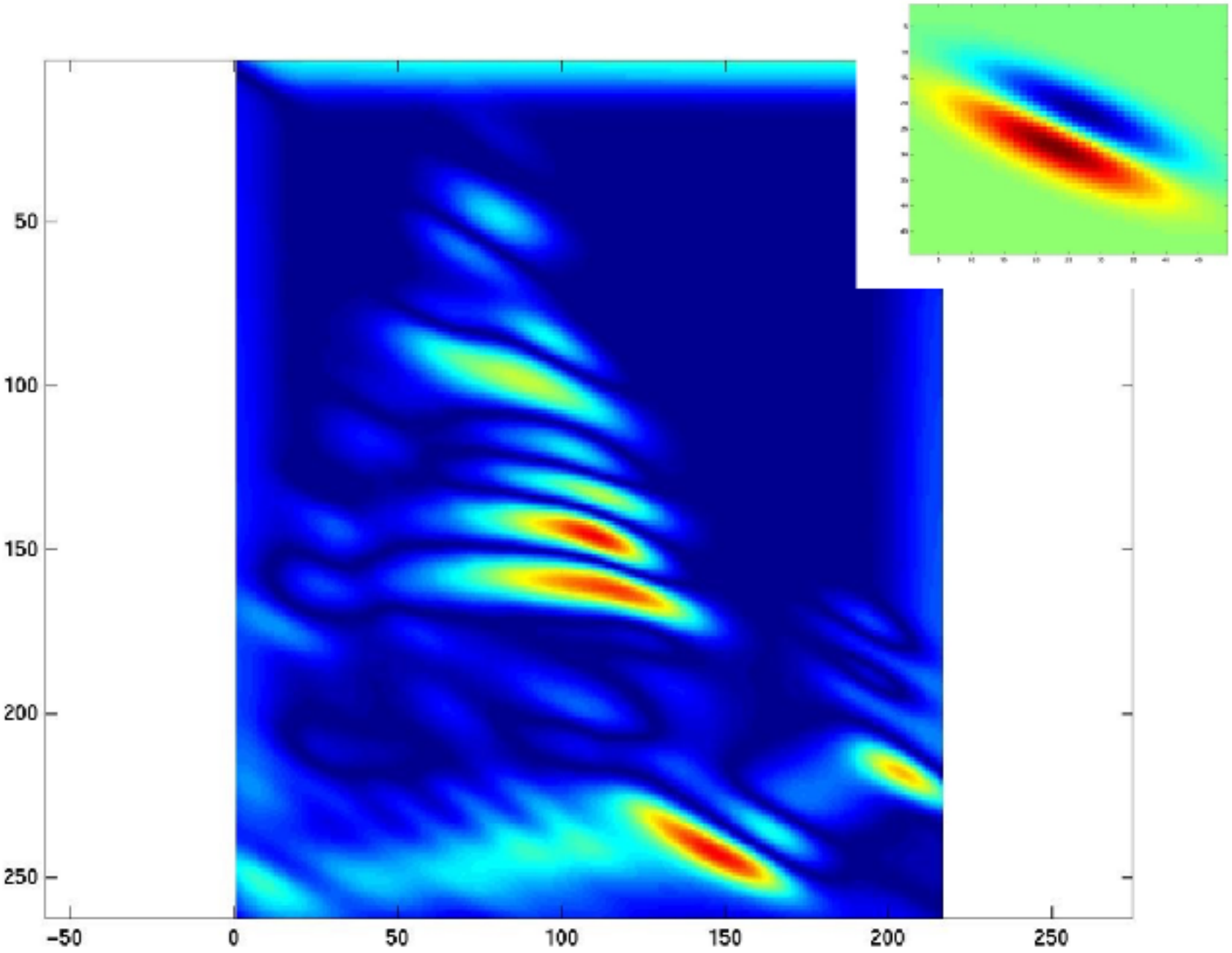

50
100
150
200
250
−50
0
50
100
150
200
250

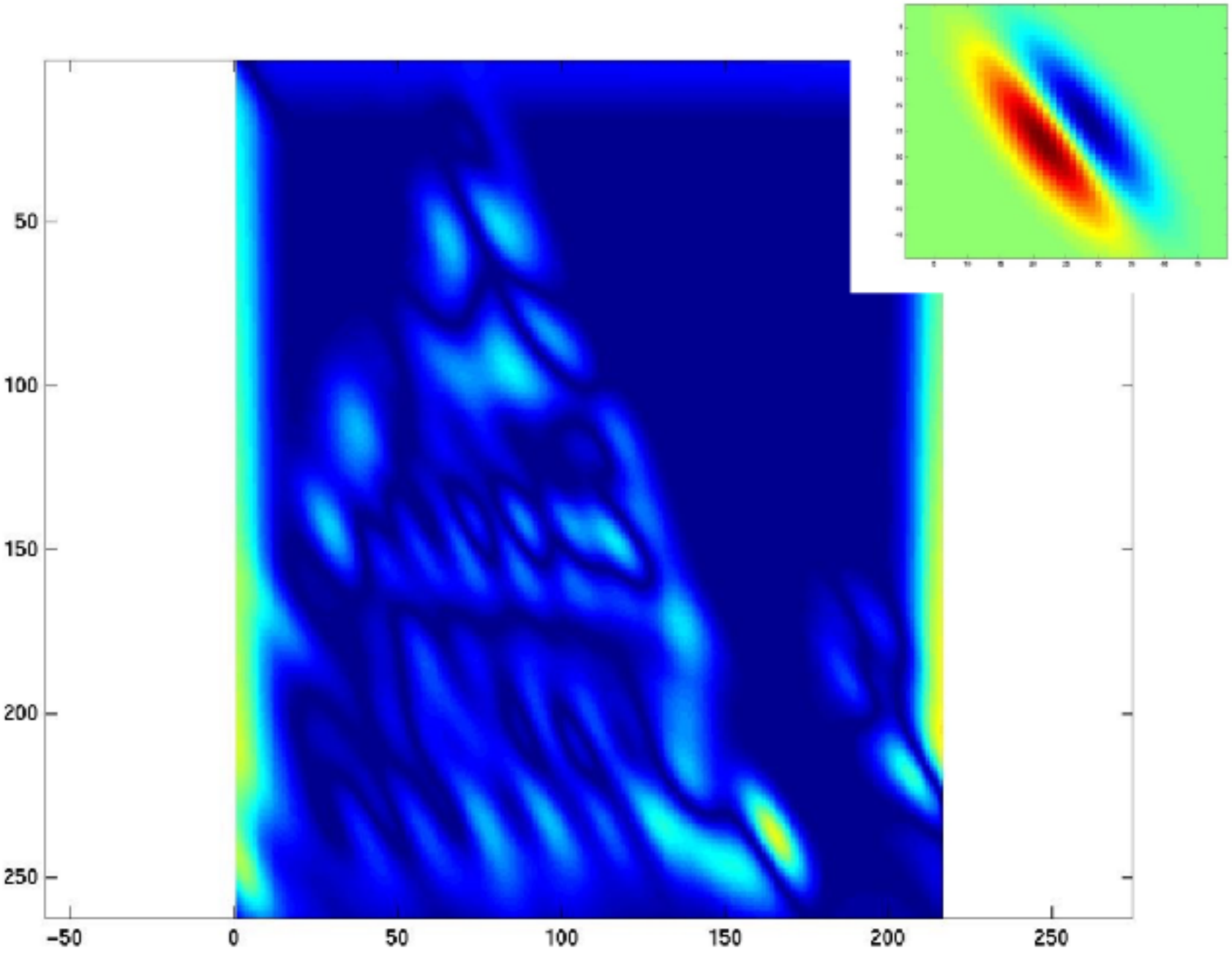

50
100
150
200
250
−50
0
50
100
150
200
250

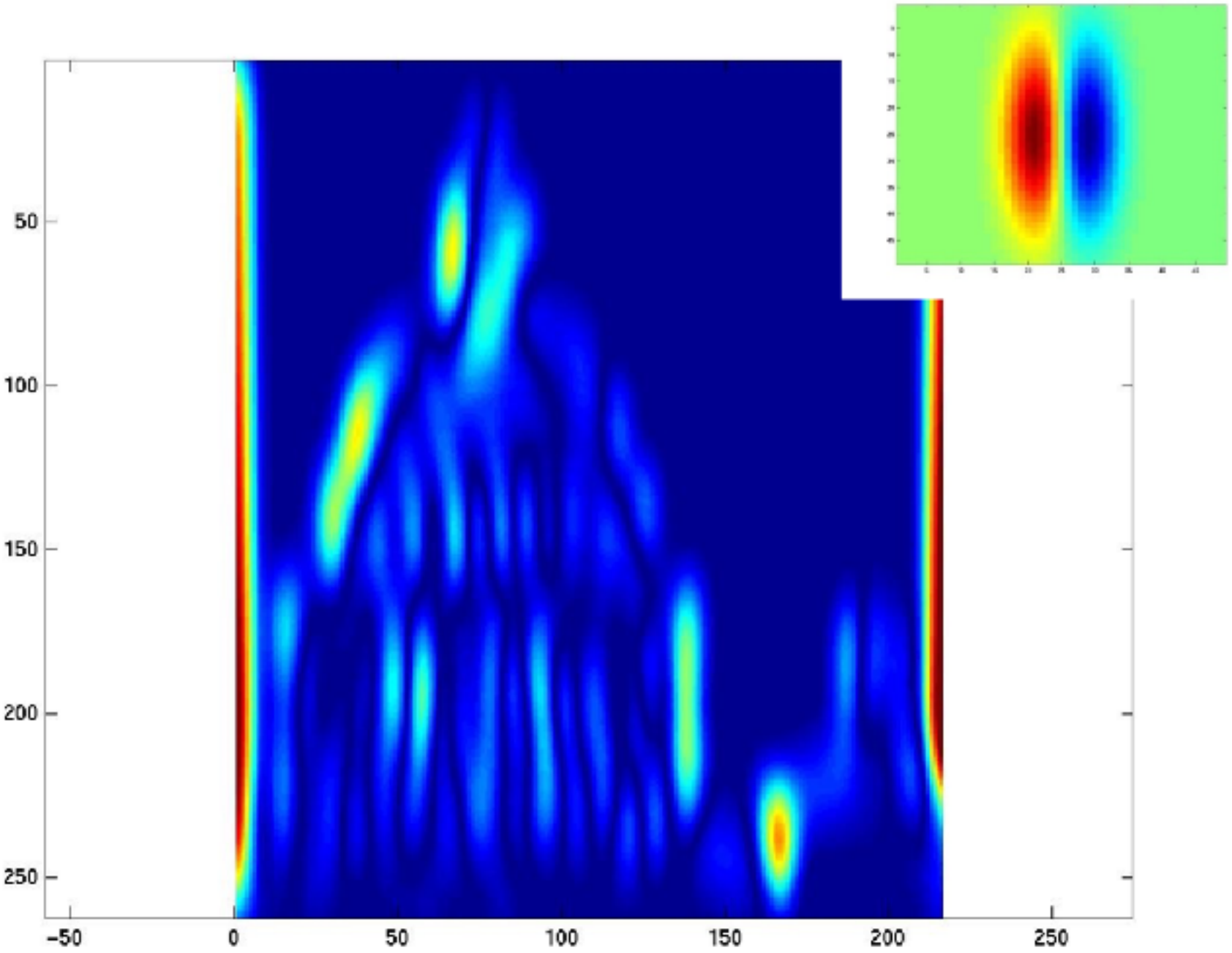

50
100
150
200
250
−50
0
50
100
150
200
250

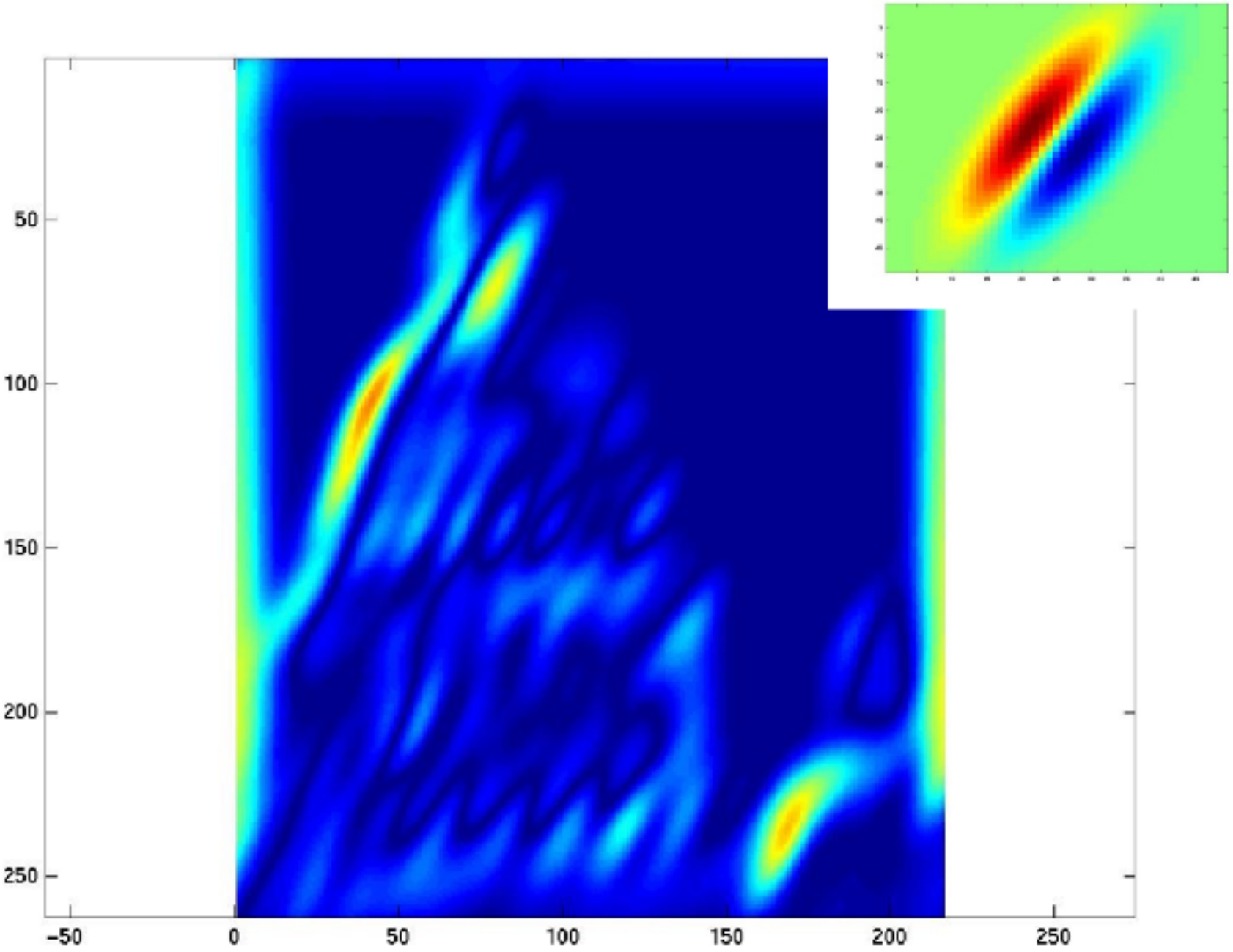

50
100
150
200
250
−50
0
50
100
150
200
250

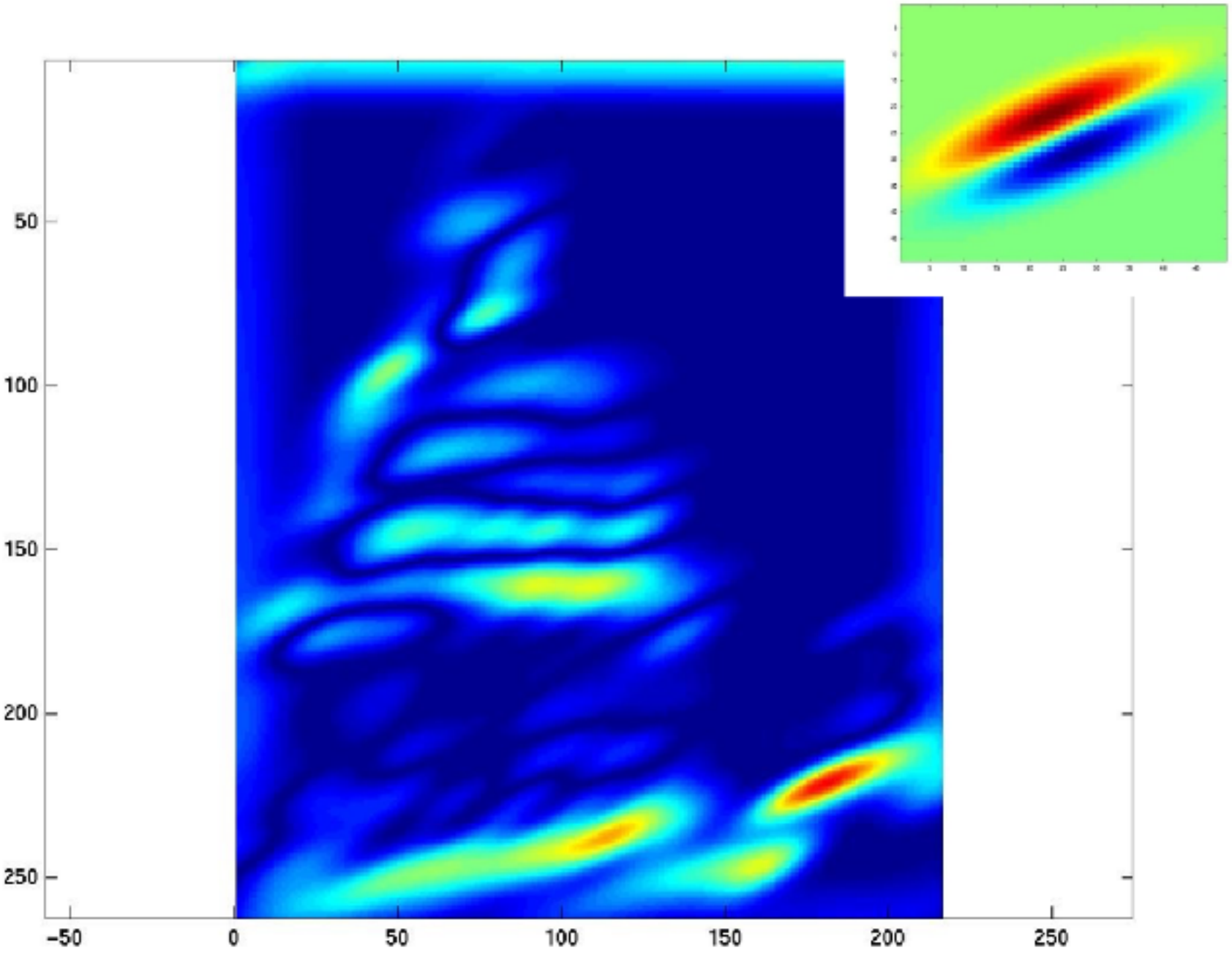

50
100
150
200
250
−50
0
50
100
150
200
250

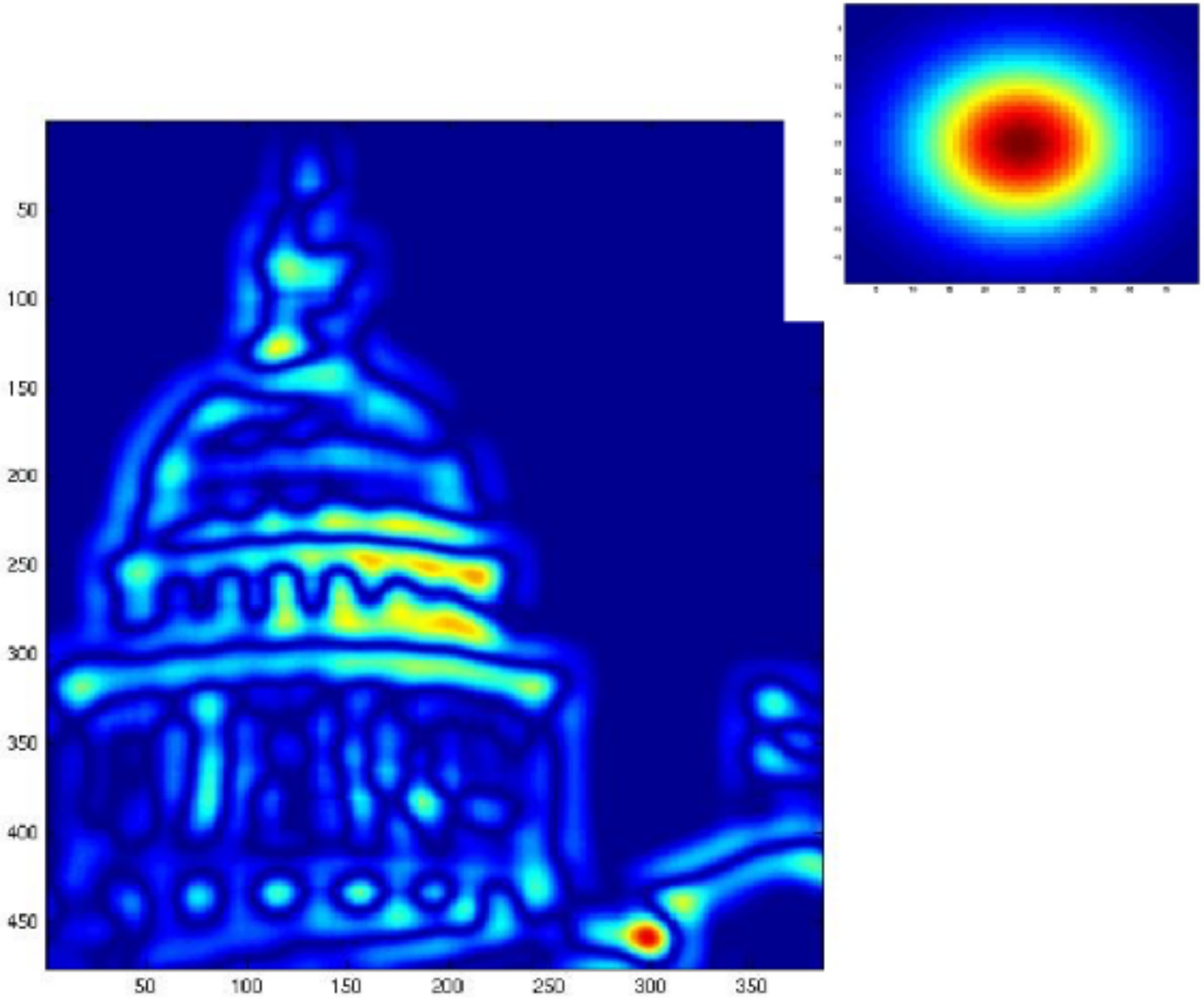

50
100
150
200
250
−50
0
50
100
150
200
250

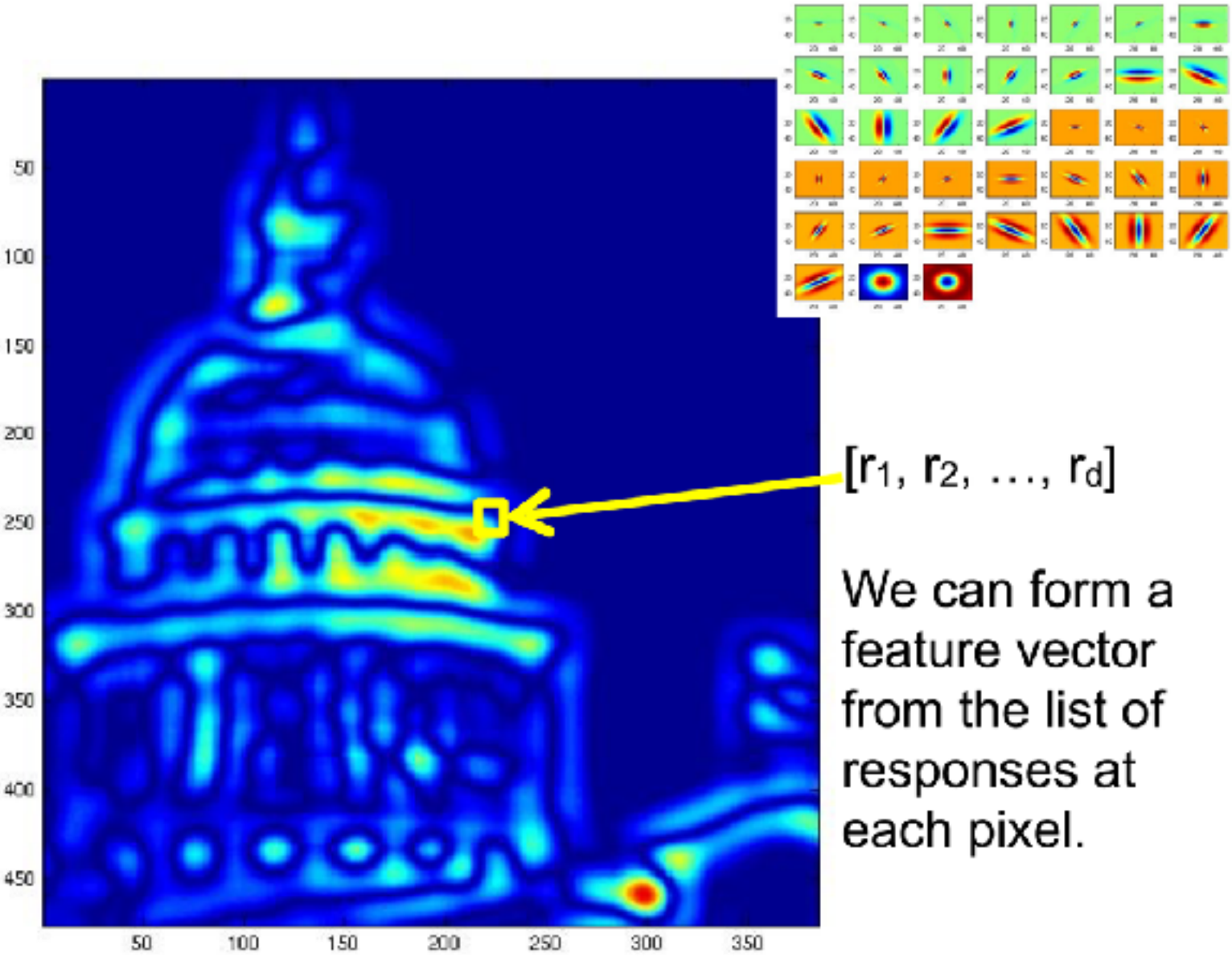

50
100
150
200
250
−50
0
50
100
150
200
250


50
100
150
200
250
−50
0
50
100
150
200
250


50
100
150
200
250
−50
0
50
100
150
200
250

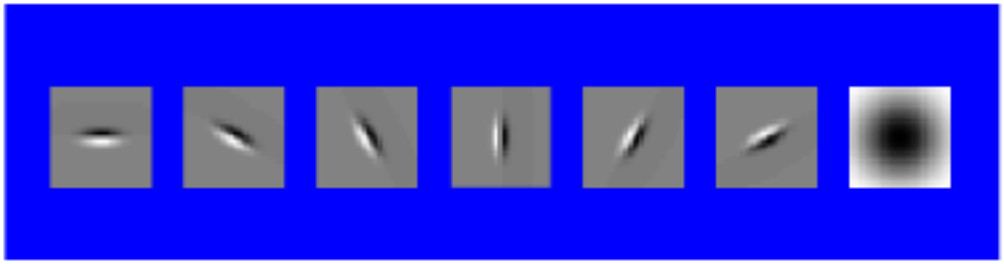

50
100
150
200
250
−50
0
50
100
150
200
250


50
100
150
200
250
300
350
400
450
50
100
150
200
250
300
350

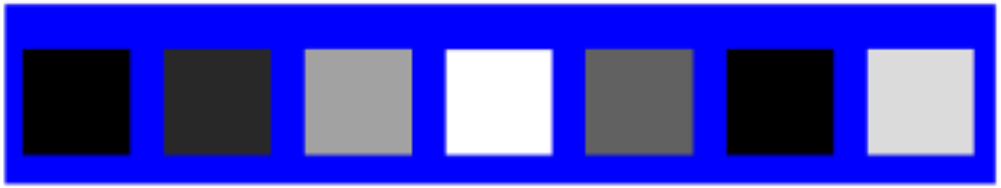

50
100
150
200
250
300
350
400
450
50
100
150
200
250
300
350
$[r_1, r_2, \ldots, r_d]$
We can form a feature vector from the list of responses at each pixel.

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

Filters

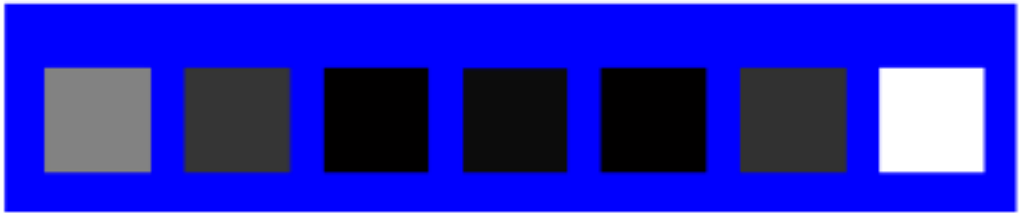


1



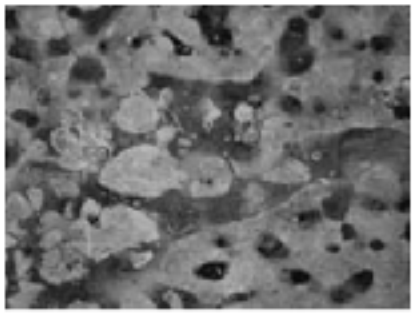
2

3



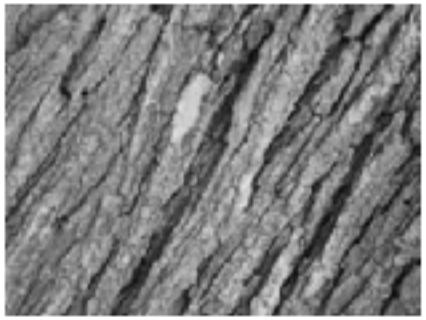
Mean abs responses

A



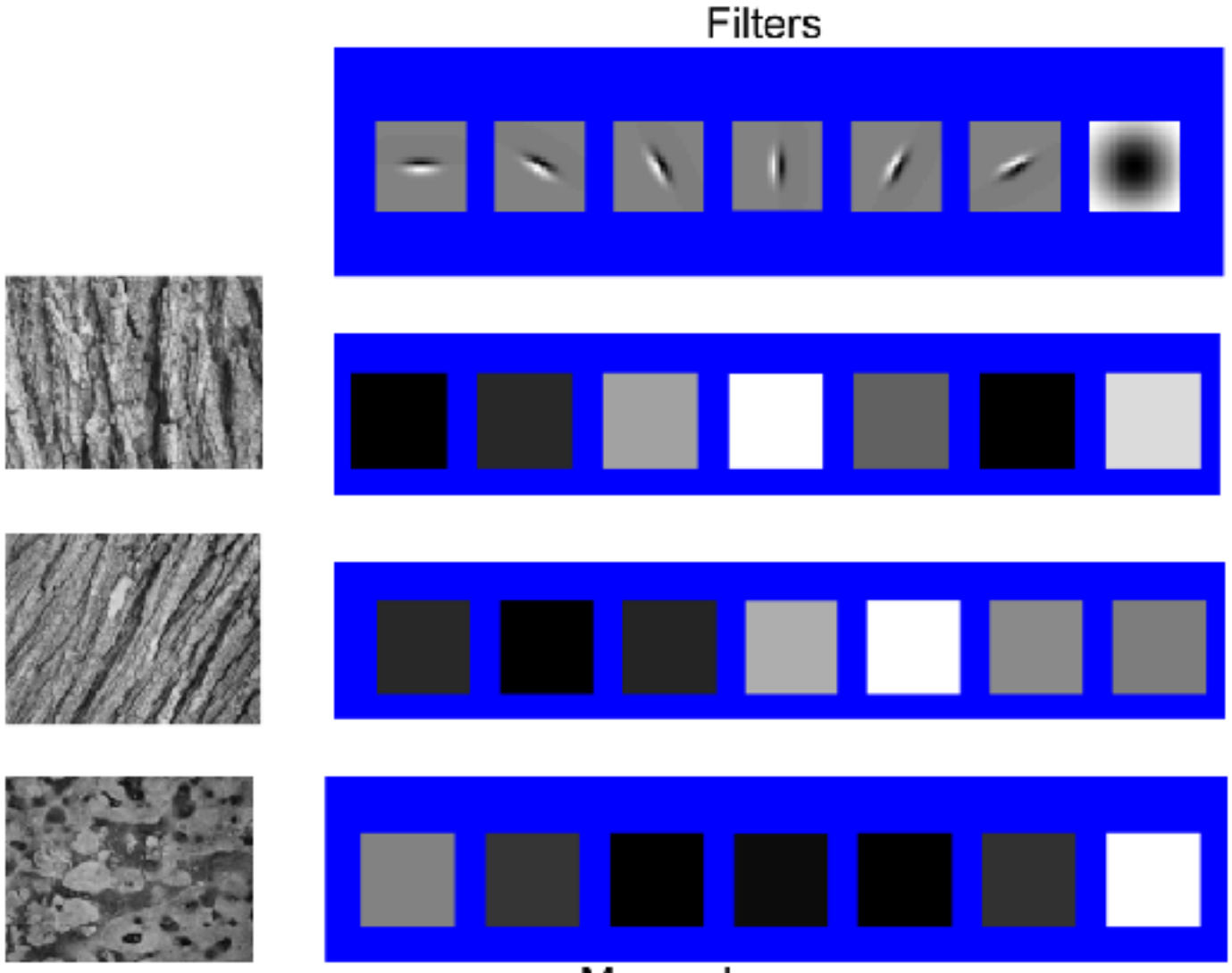
B

C



Derek Hoiem

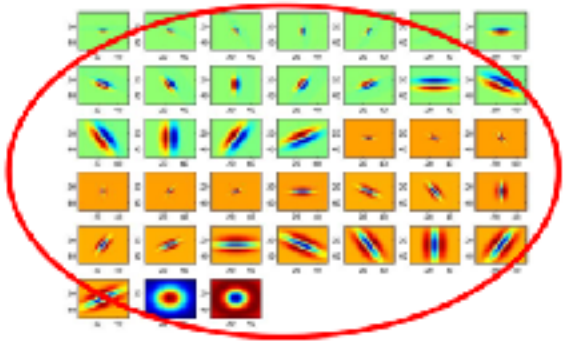
Representing texture by mean abs response

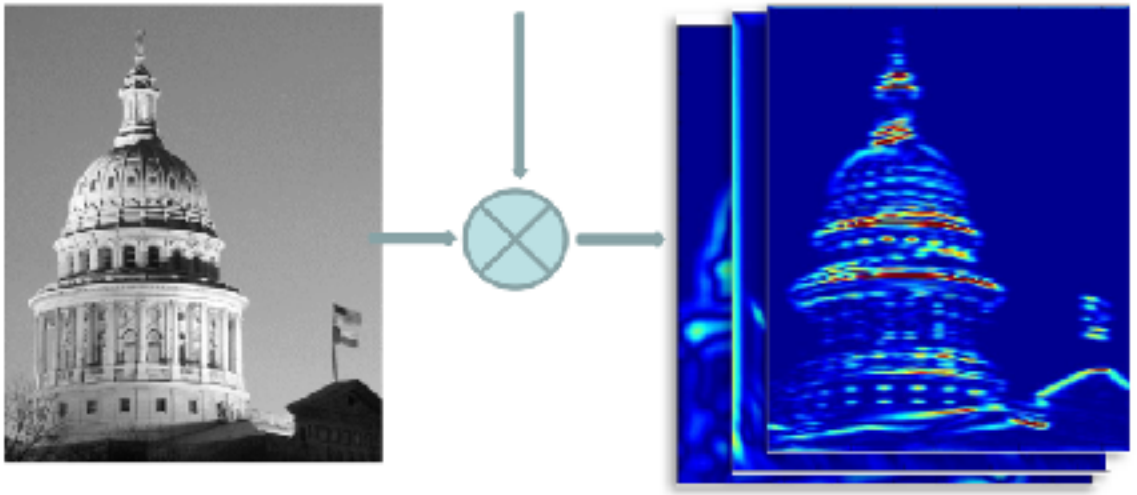


Derek Hoiem

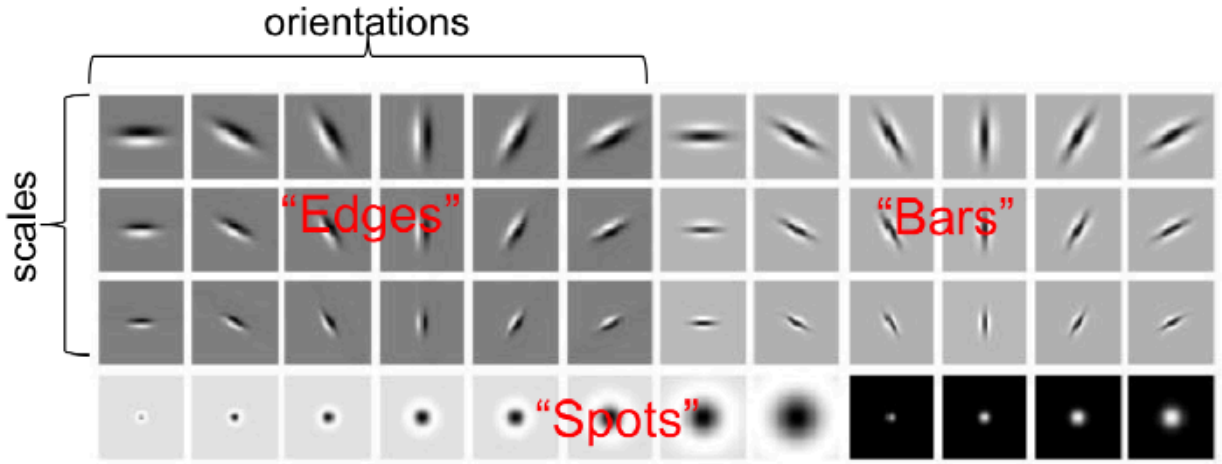
Convolutional Neural Network

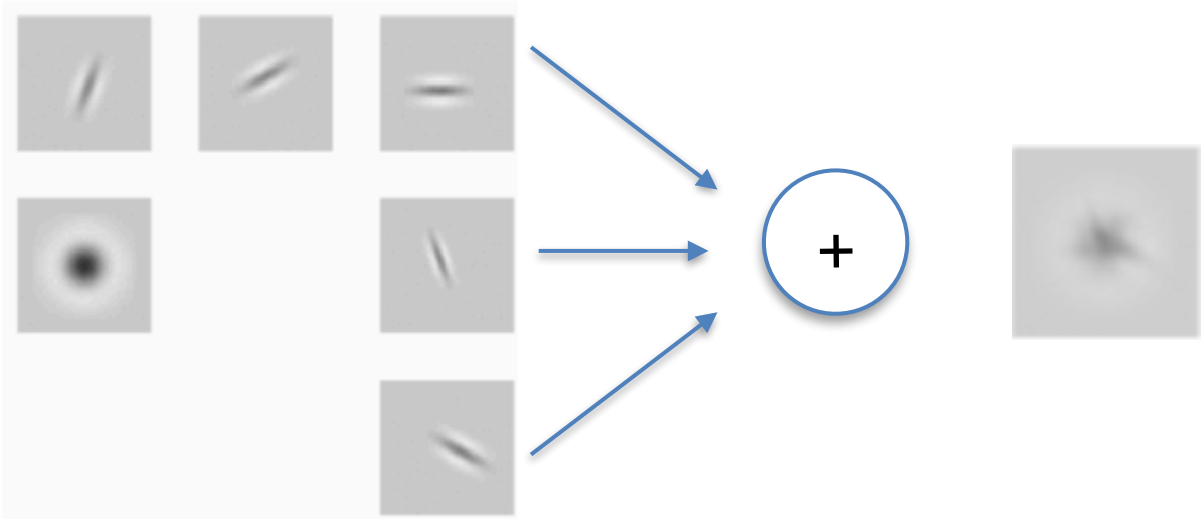
Learn from Data what these filters should be



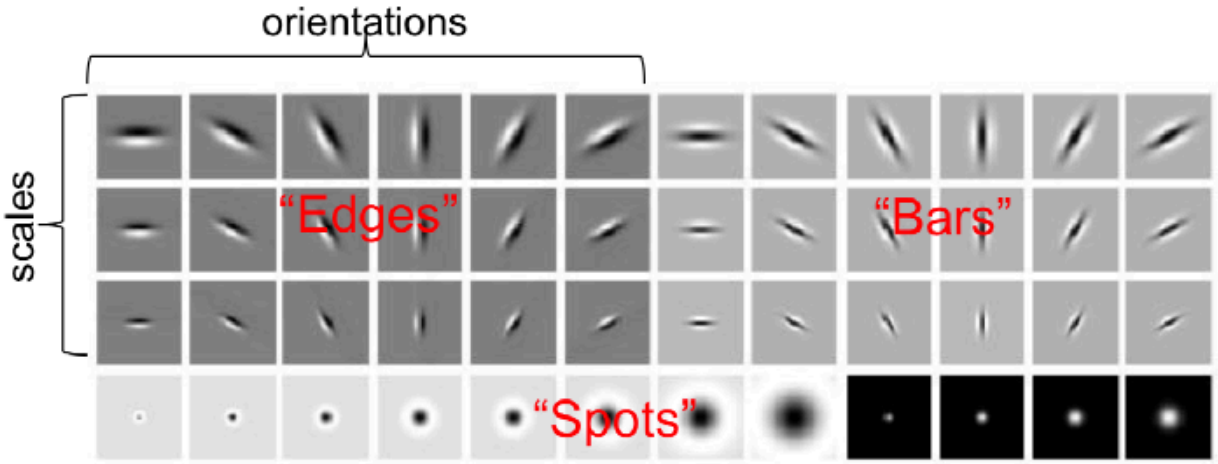


More on Edge Detection

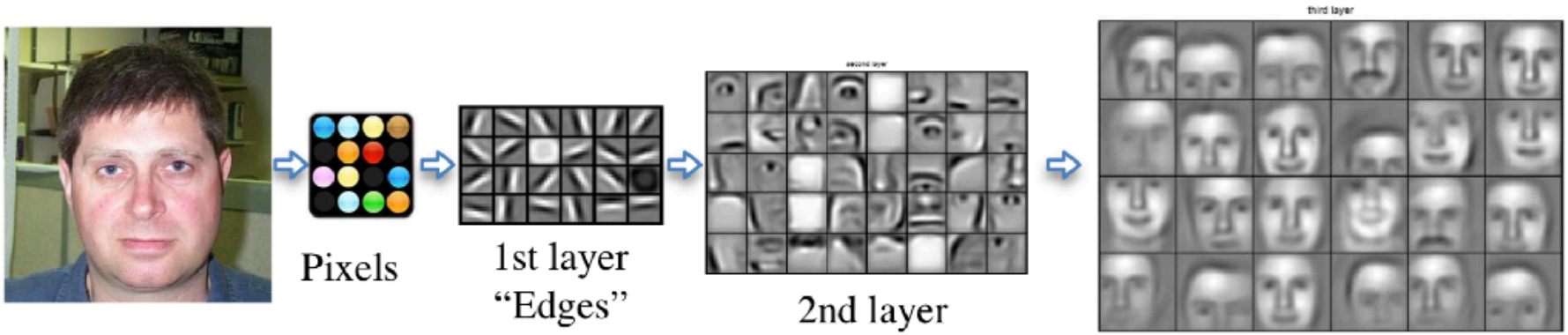




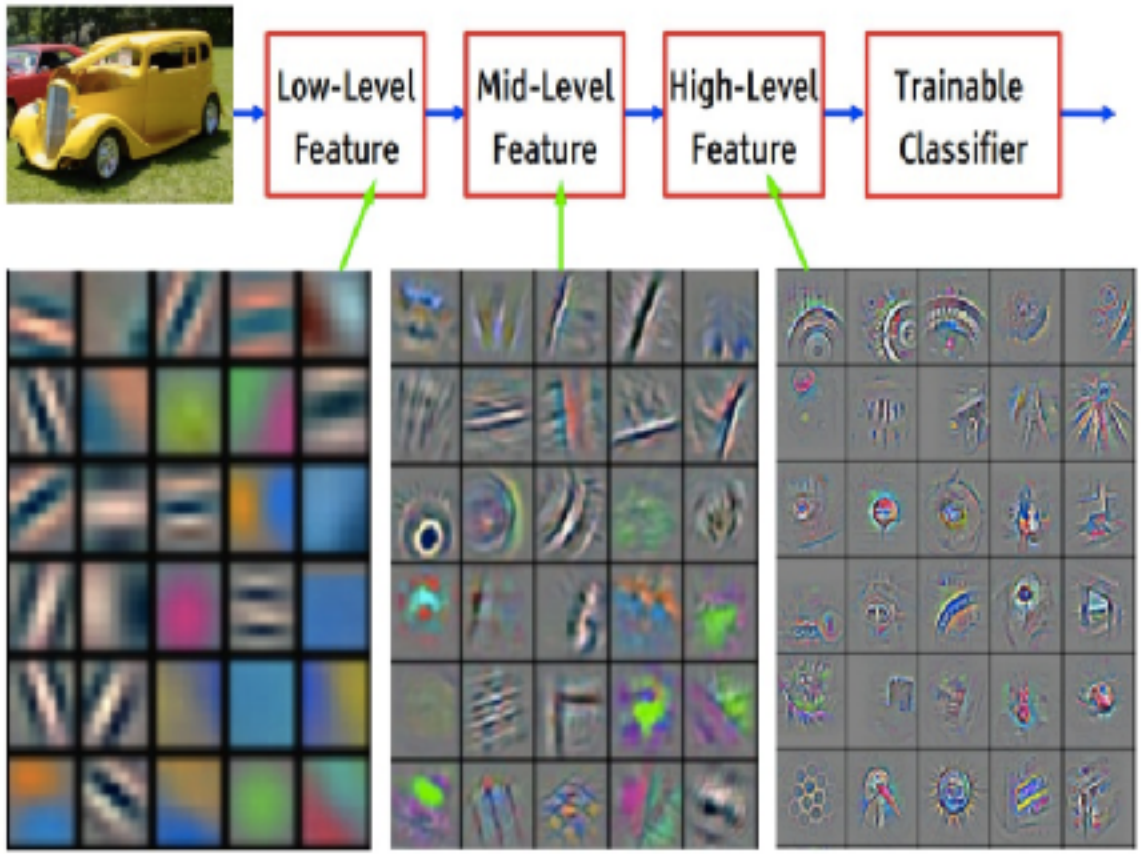
More on Edge Detection



Learned hierarchical feature representation: Sparse DBN



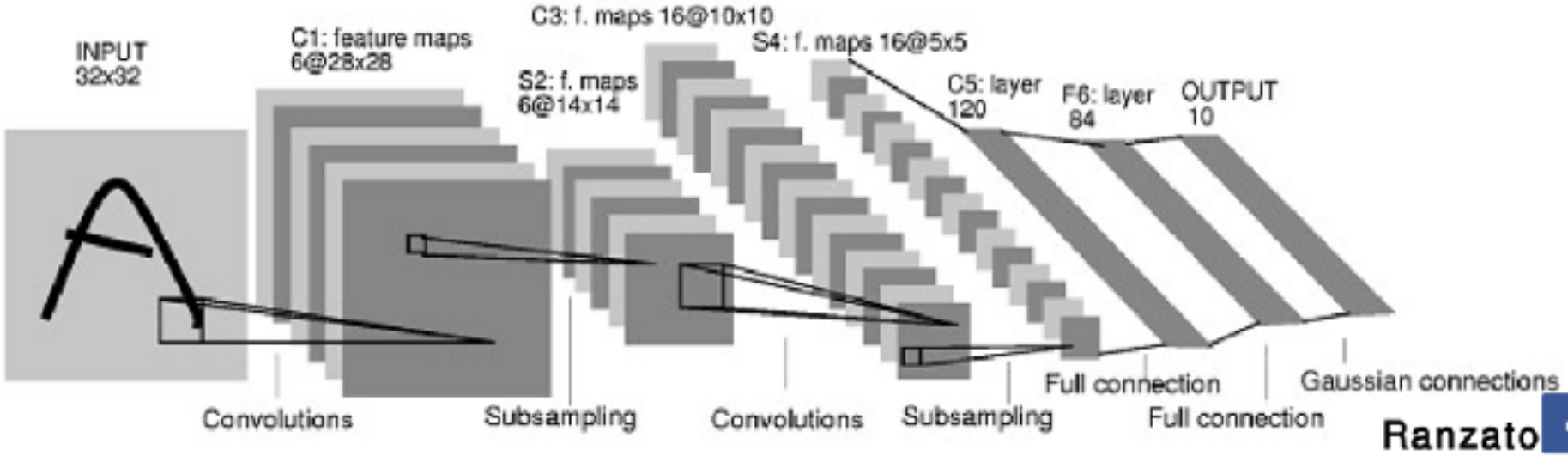
Convolutional Neural Network



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Convolutional Neural Network

- Le Cun 1998



Ranzato