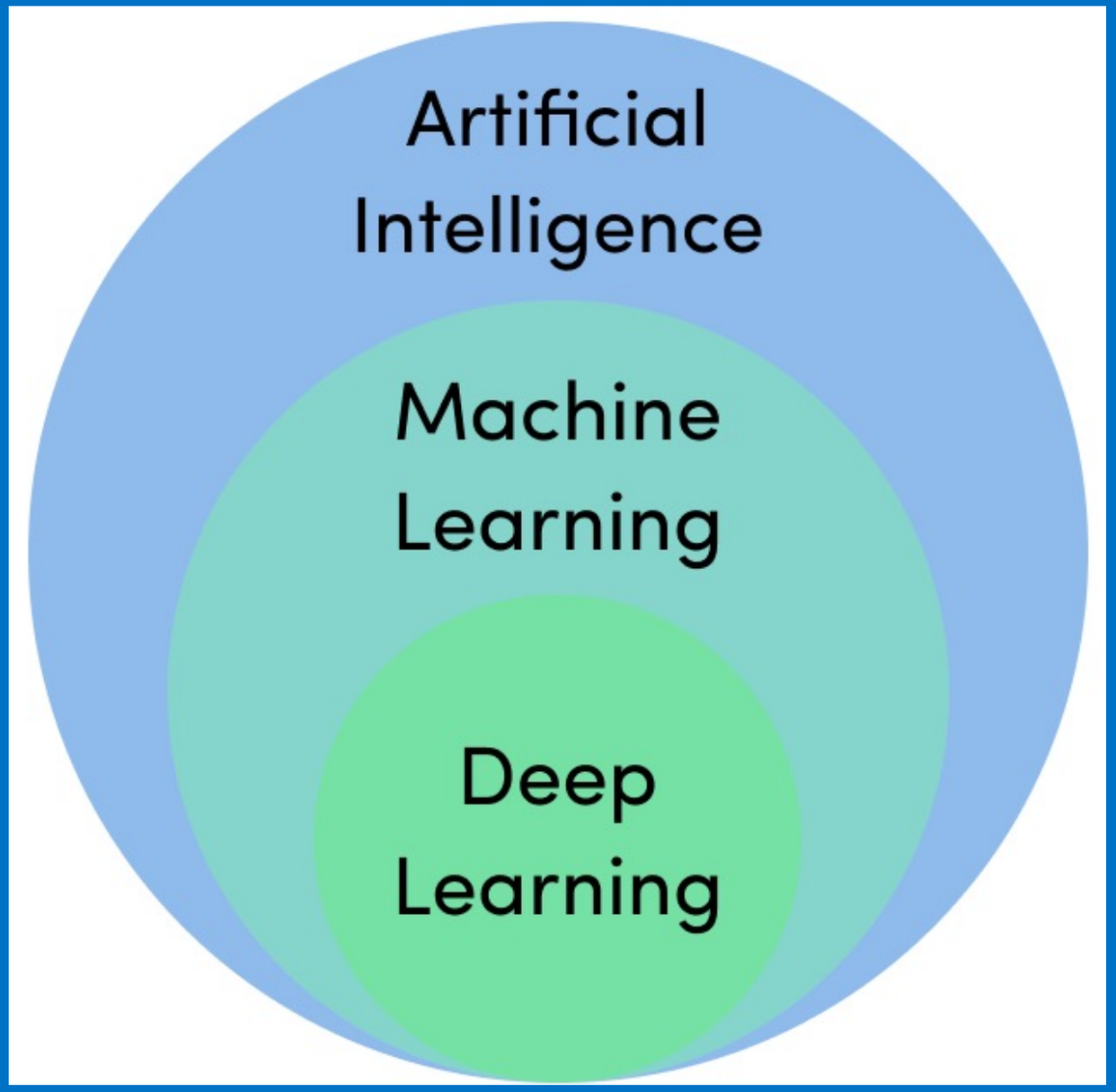
Deep Learning for Medical Imaging

Lecture 7-January

Waqas Sultani
MedAI Research Group
Information Technology
University

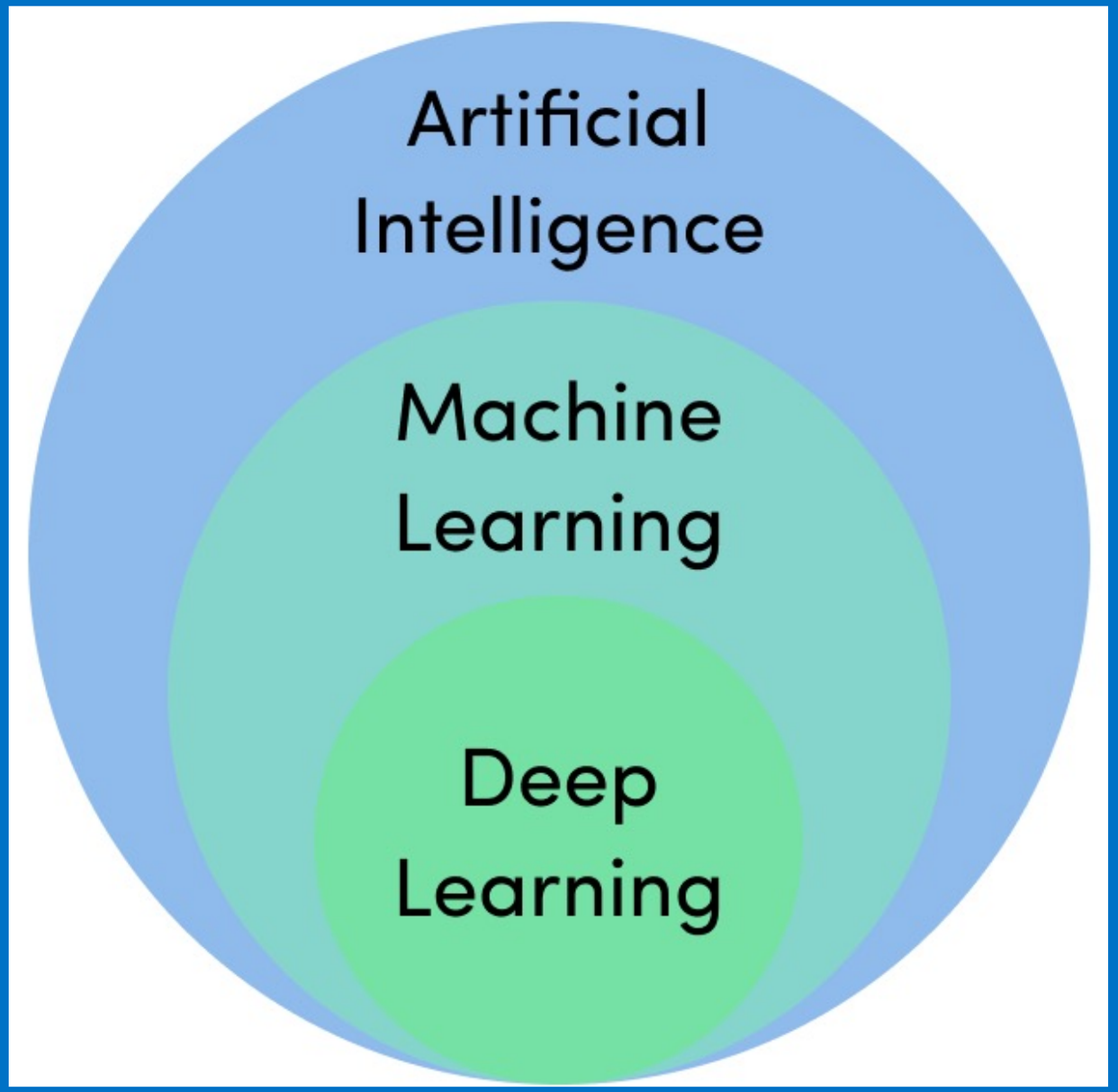
- Introduction to Deep Learning
- Few Shot Learning
- Weakly Supervised Learning
- Self-Supervised Learning

Deep Learning --- General Introduction



AI

- A branch of computer science dealing with the simulation of intelligent behaviour in computers.
- The capability of a machine to imitate intelligent human behaviour.
- A computer system able to perform tasks that normally require human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages.

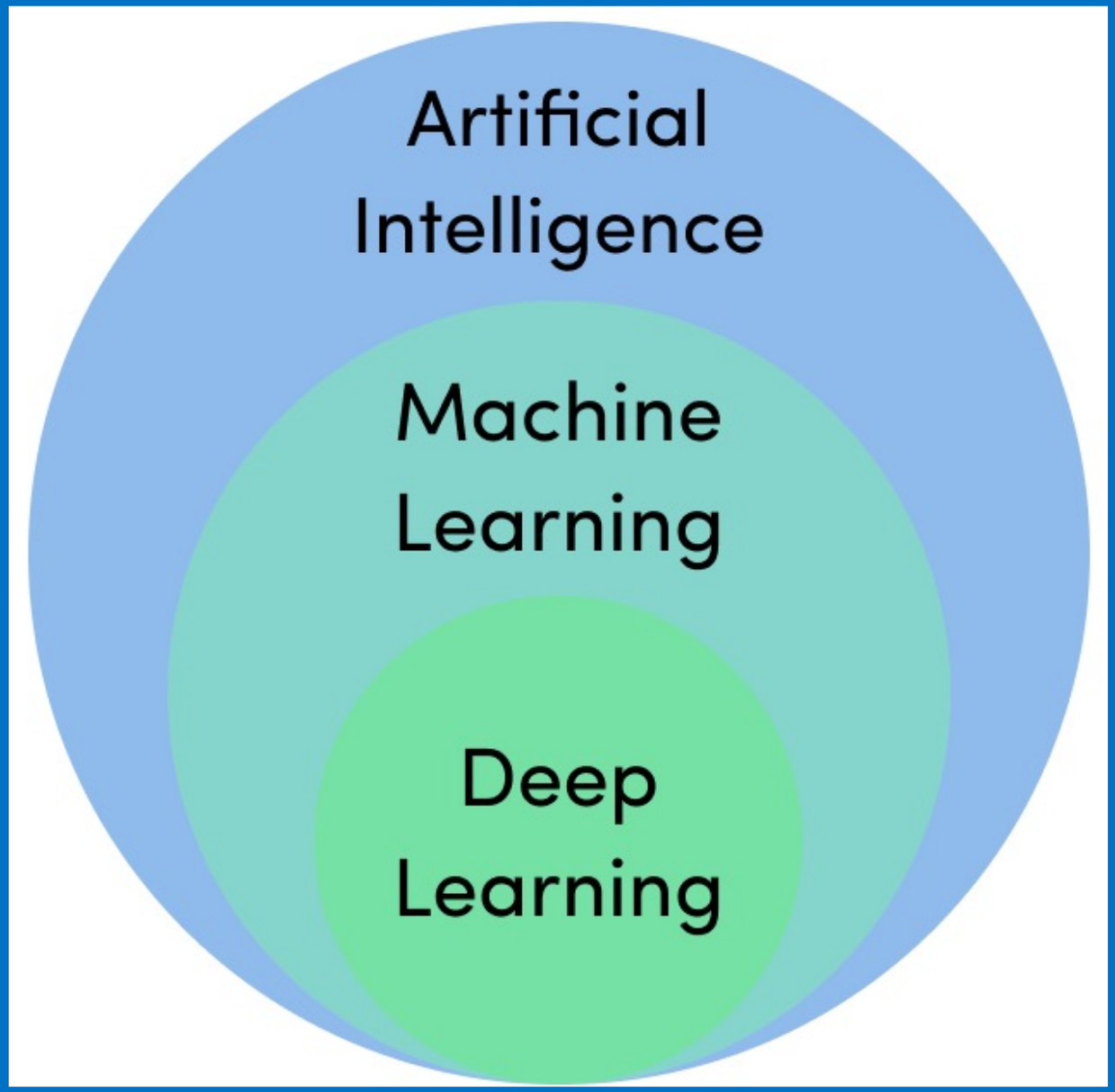


AI

- A branch of computer science dealing with the simulation of intelligent behaviour in computers.
- The capability of a machine to imitate intelligent human behaviour.
- A computer system able to perform tasks that normally require human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages.

ML

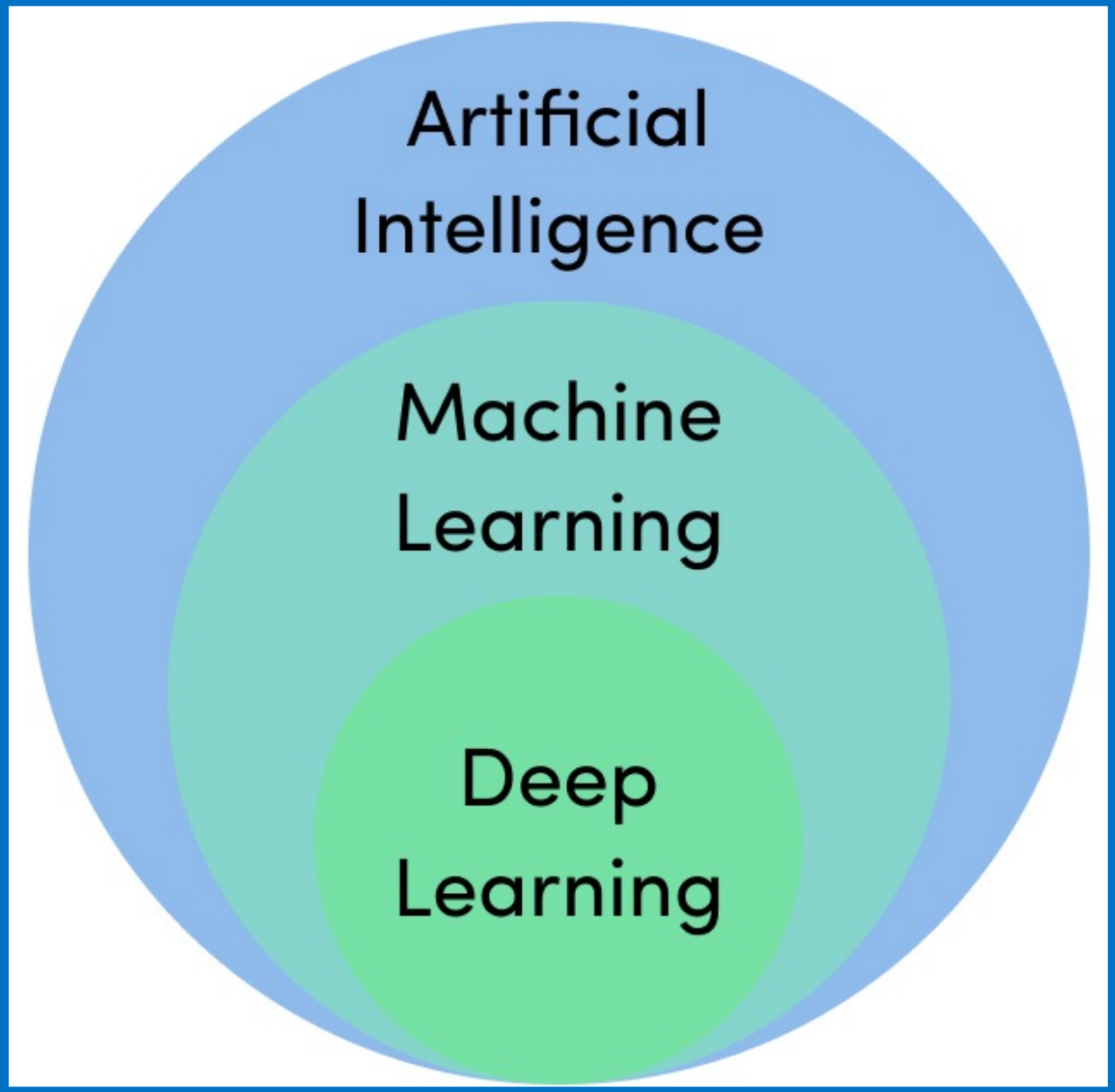
- A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E. –Tom Mitchell



AI (Not ML)

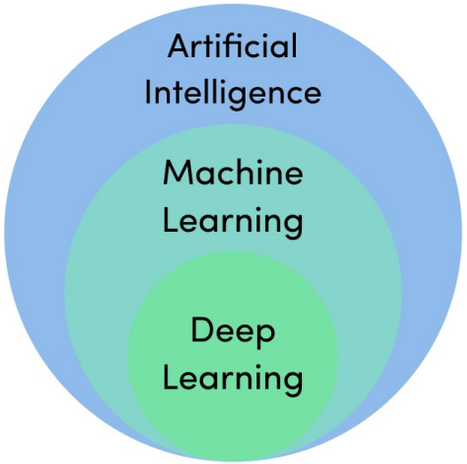
- Machine learning is a subset of AI. That is, all machine learning counts as AI, but not all AI counts as machine learning.
- For example, symbolic logic – rules engines, expert systems and knowledge graphs – could all be described as AI, and none of them are machine learning.

- One aspect that separates machine *learning* from the knowledge graphs and expert systems is its ability to modify itself when exposed to more data; i.e. machine learning is *dynamic* and does not require human intervention to make certain changes.



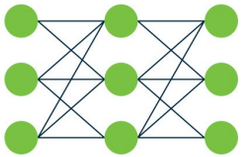
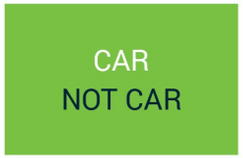
Deep Learning

•Deep learning is a subset of machine learning. Usually, when people use the term deep learning, they are referring to deep artificial neural networks.



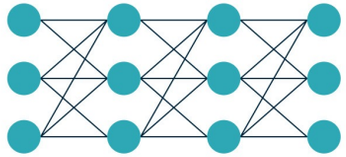
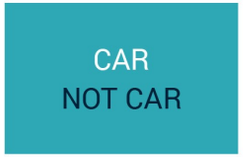
Machine Learning



Input → Feature extraction → Classification → Output

Deep Learning

Input → Feature extraction + Classification → Output

How it Works:
Convolutional Neural Networks

A toy ConvNet: X's and O's

Says whether a picture is of an X or an O

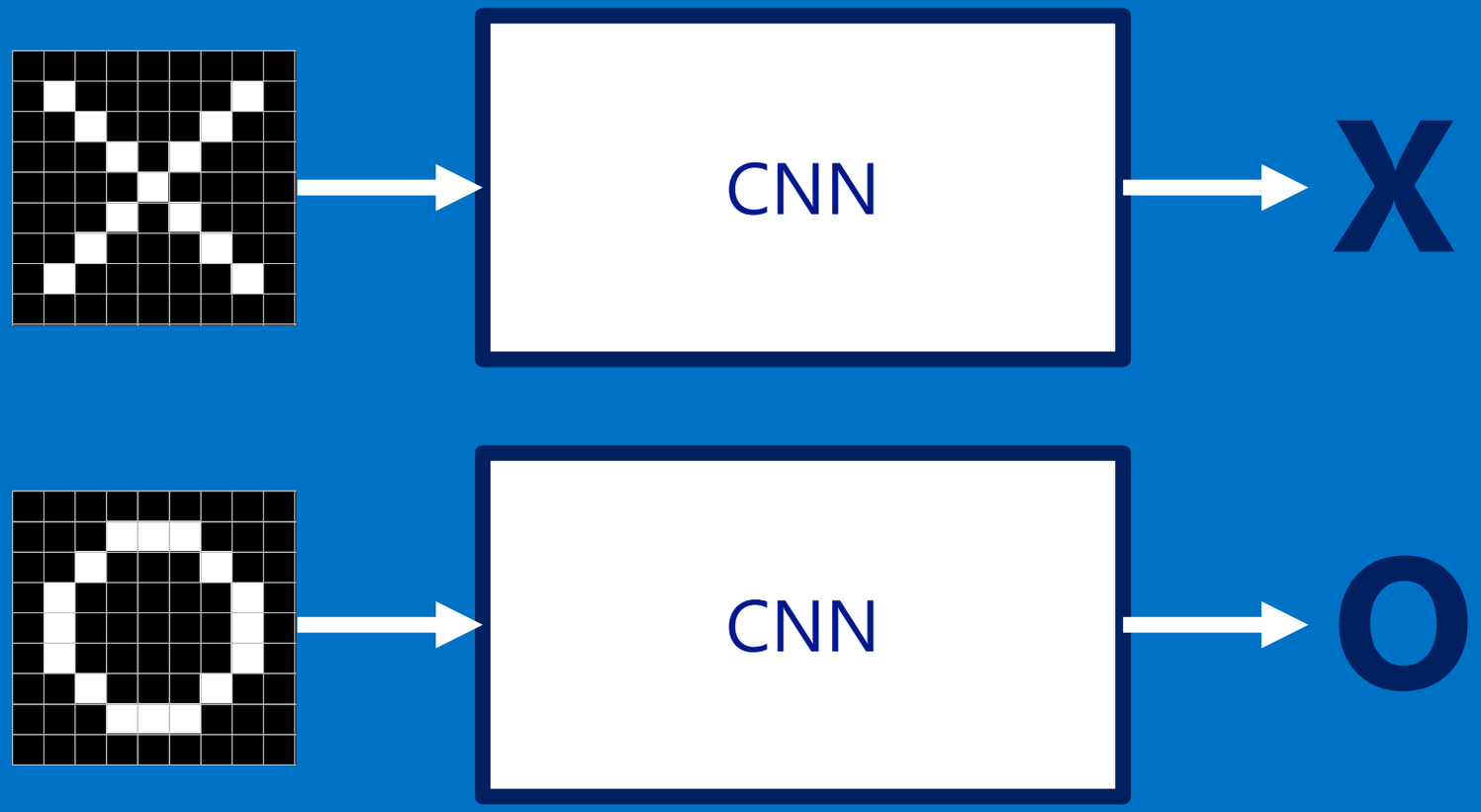
A two-dimensional
array of pixels

A toy ConvNet: X's and O's

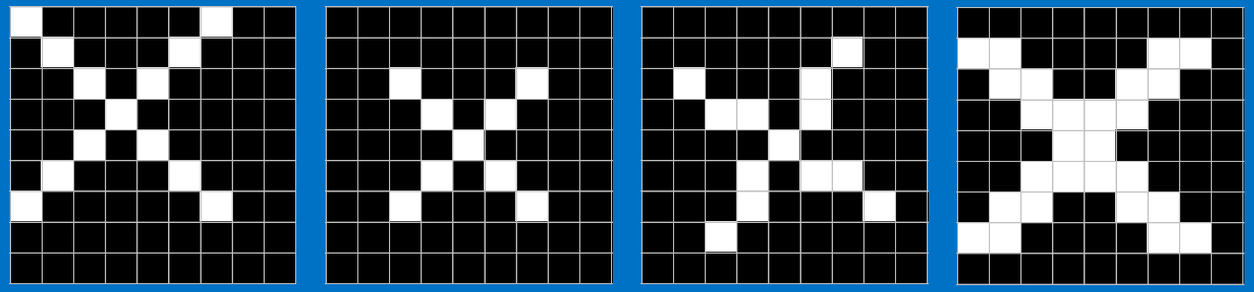
Says whether a picture is of an X or an O

A two-dimensional
array of pixels

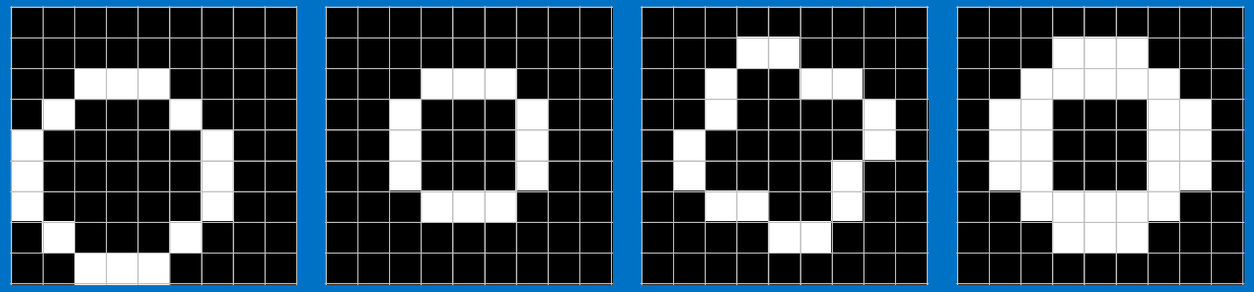
For example



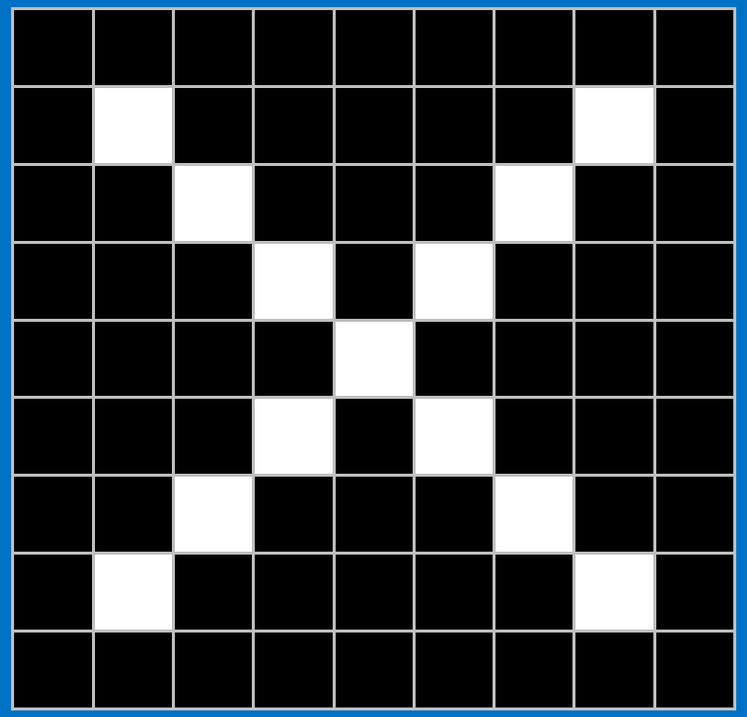
Trickier cases



translation scaling rotation weight

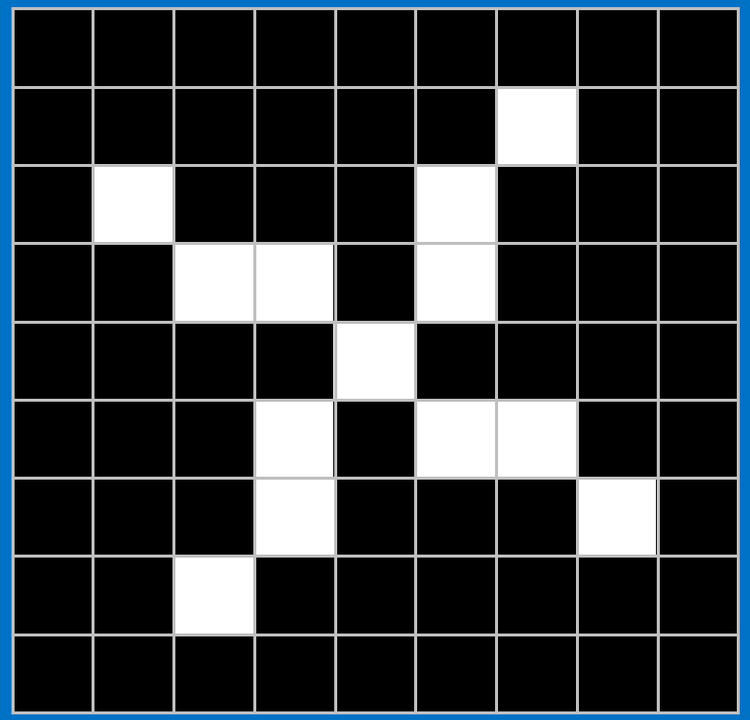


Deciding is hard



?

=



What computers see

-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	-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	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	-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	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	-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	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

What computers see

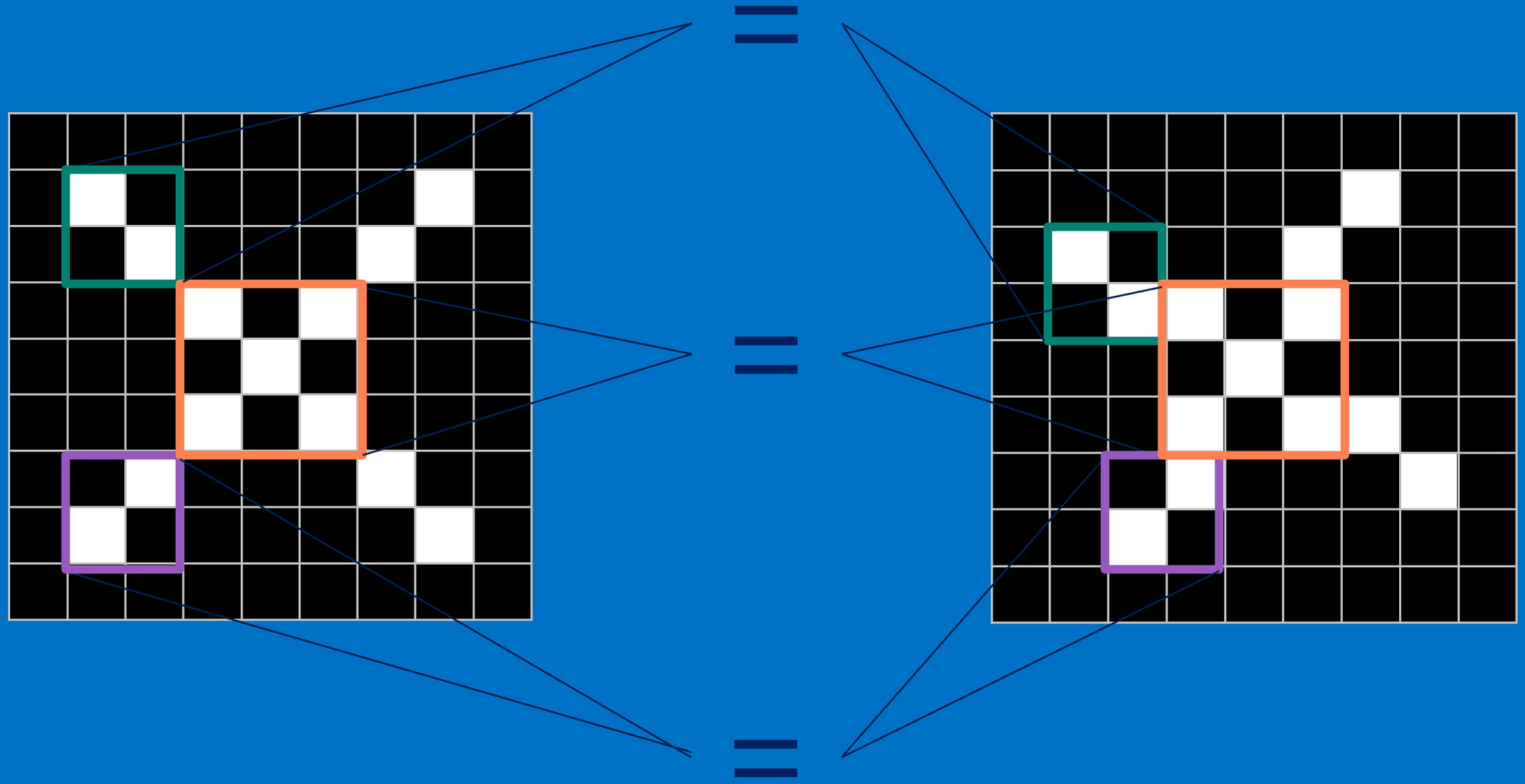
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	X	-1	-1	-1	-1	X	X	-1
-1	X	X	-1	-1	X	X	-1	-1
-1	-1	X	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	X	-1	-1
-1	-1	X	X	-1	-1	X	X	-1
-1	X	X	-1	-1	-1	-1	X	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

Computers are literal

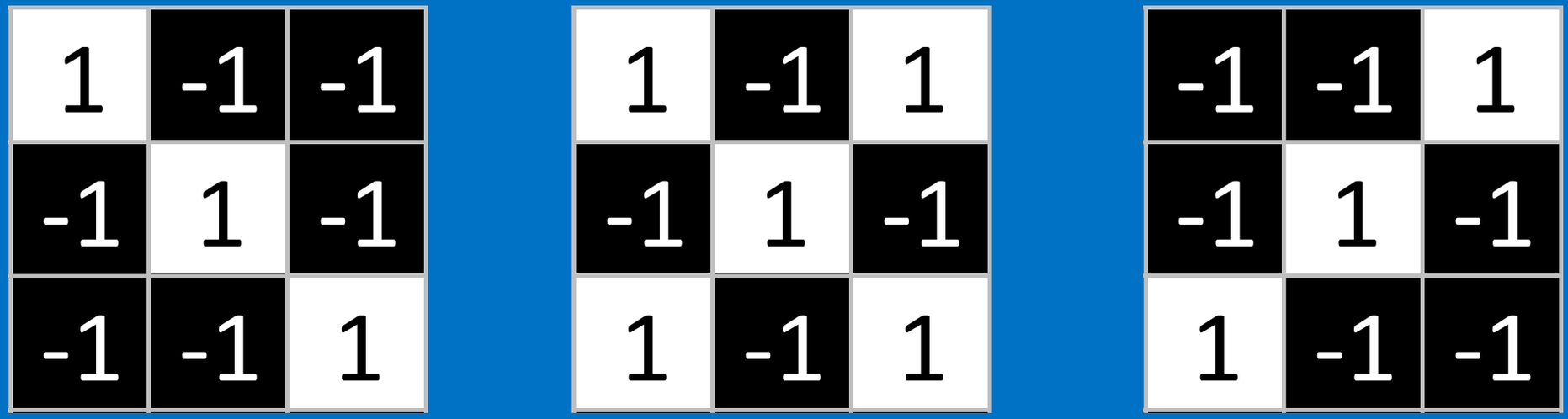
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1
-1	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	-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	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

ConvNets match pieces of the image



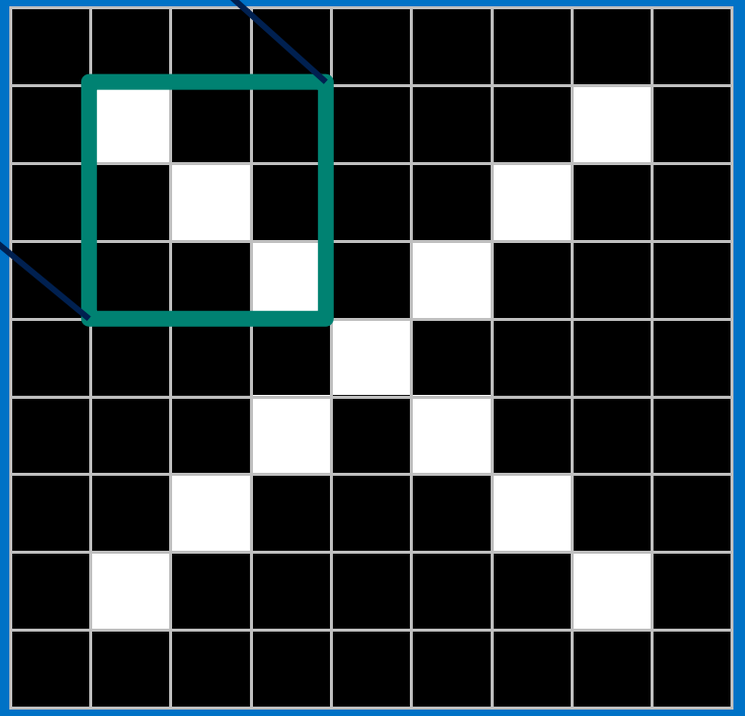
Features match pieces of the image



1	-1	-1
-1	1	-1
-1	-1	1

1	-1	1
-1	1	-1
1	-1	1

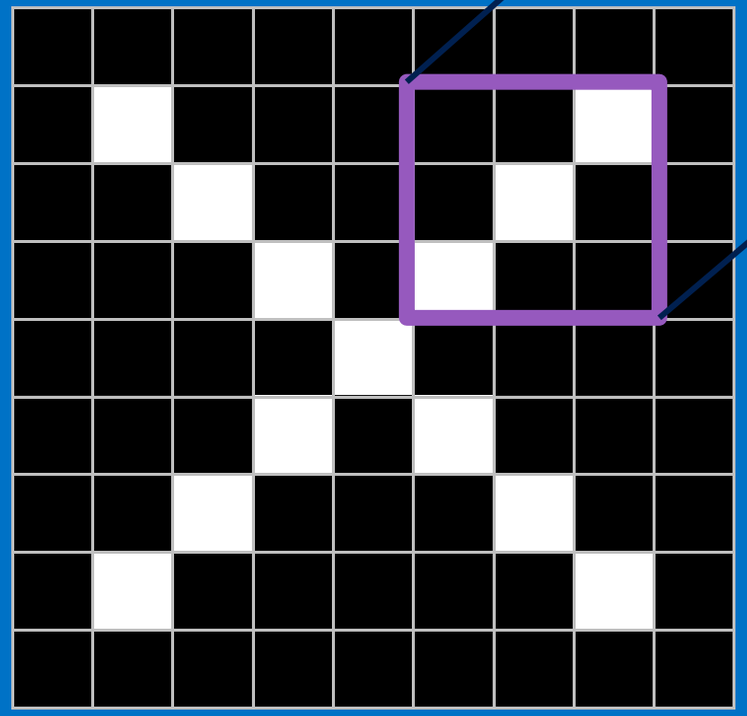
-1	-1	1
-1	1	-1
1	-1	-1



1	-1	-1
-1	1	-1
-1	-1	1

1	-1	1
-1	1	-1
1	-1	1

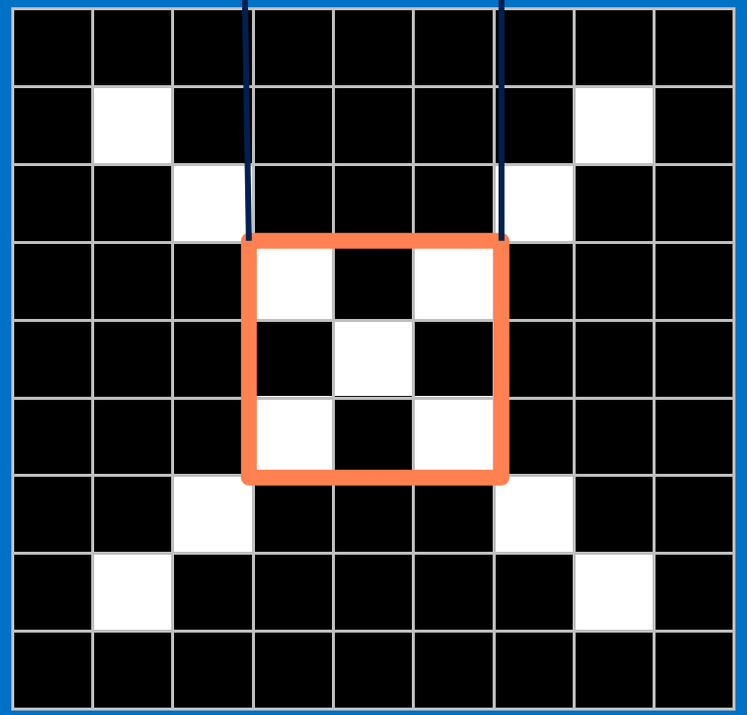
-1	-1	1
-1	1	-1
1	-1	-1

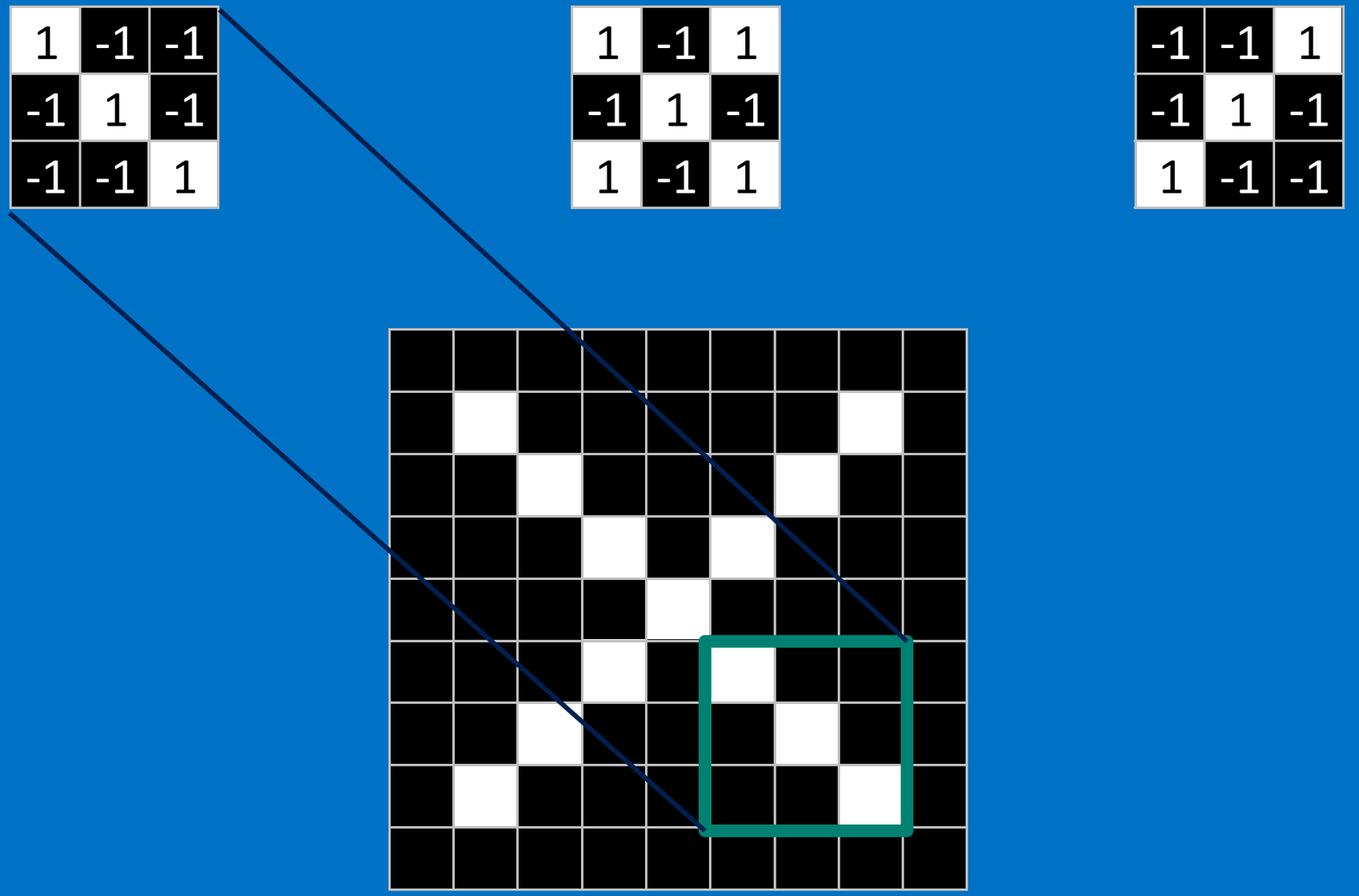


1	-1	-1
-1	1	-1
-1	-1	1

1	-1	1
-1	1	-1
1	-1	1

-1	-1	1
-1	1	-1
1	-1	-1

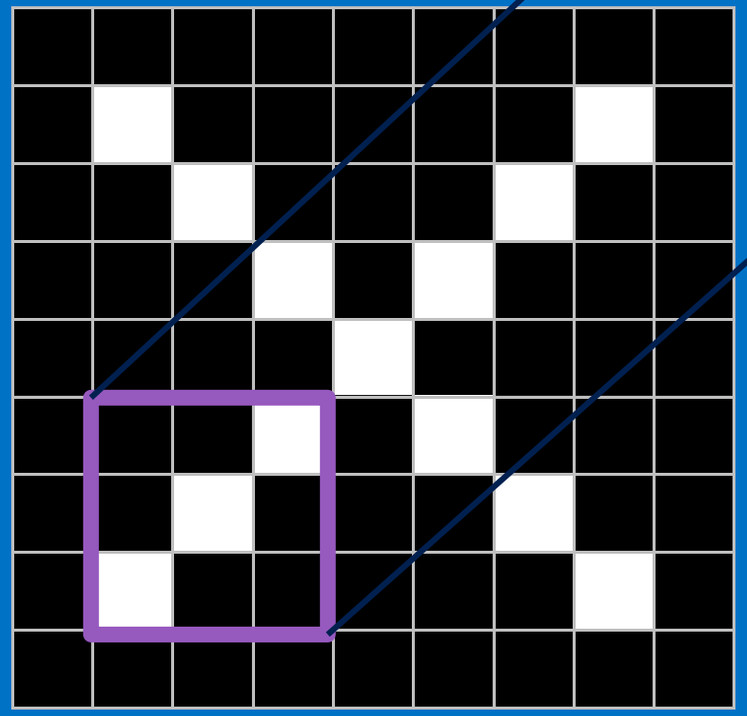



1 -1 -1
-1 1 -1
-1 -1 1
1 -1 1
-1 1 -1
1 -1 1
-1 -1 1
-1 1 -1
1 -1 -1

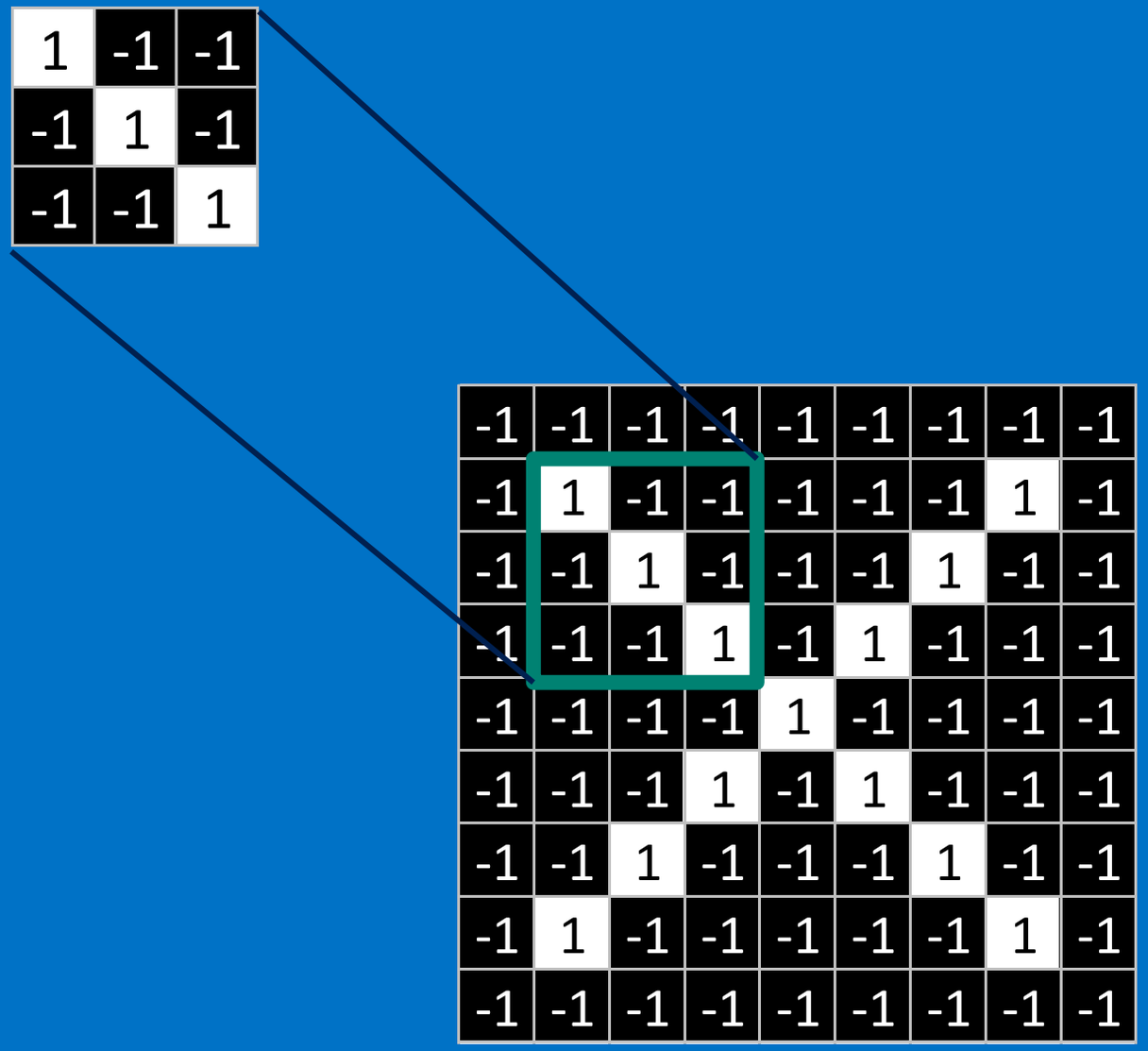
1	-1	-1
-1	1	-1
-1	-1	1

1	-1	1
-1	1	-1
1	-1	1

-1	-1	1
-1	1	-1
1	-1	-1



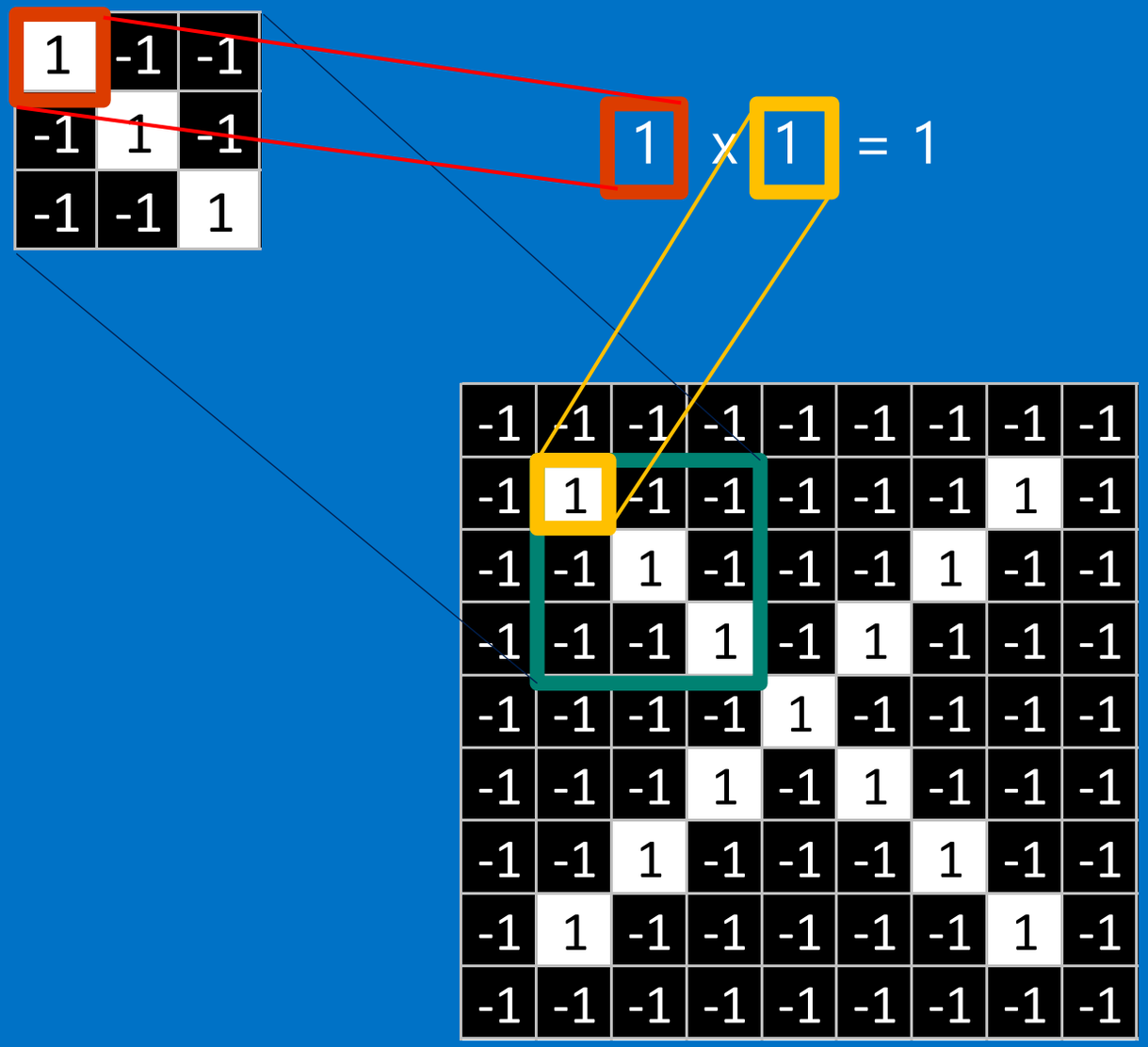
Filtering: The math behind the match



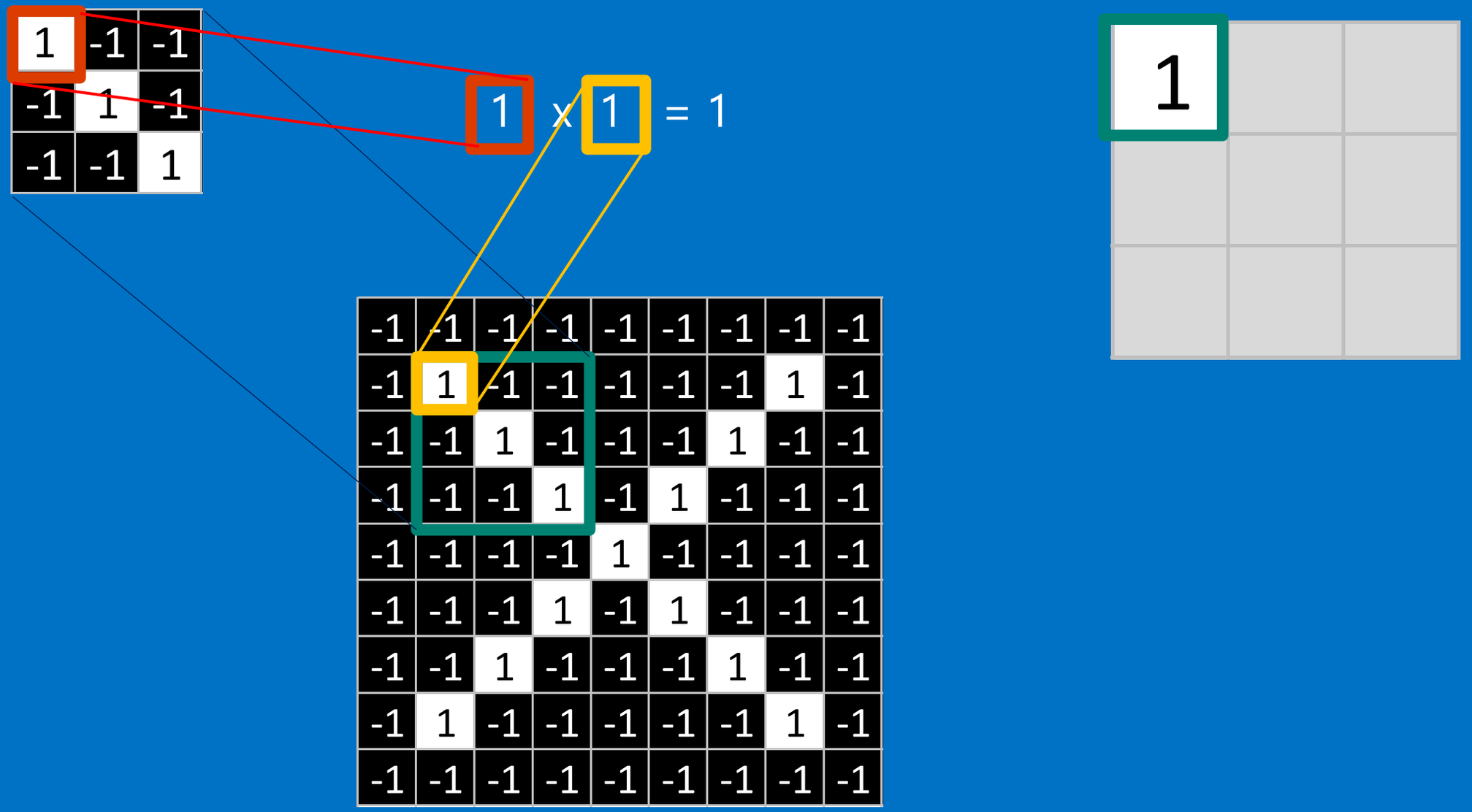
Filtering: The math behind the match

1. Line up the feature and the image patch.
2. Multiply each image pixel by the corresponding feature pixel.
3. Add them up.
4. Divide by the total number of pixels in the feature.

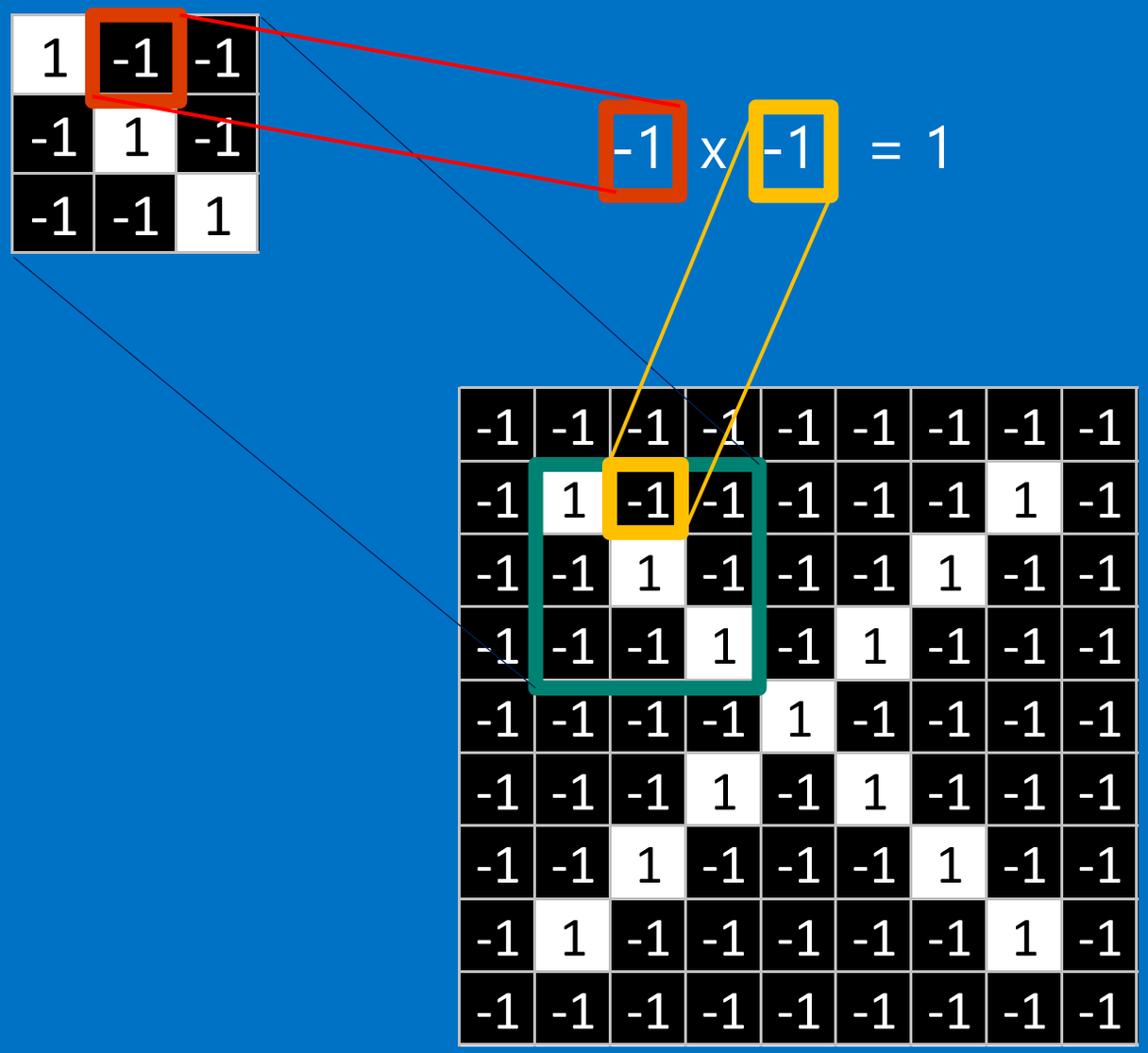
Filtering: The math behind the match

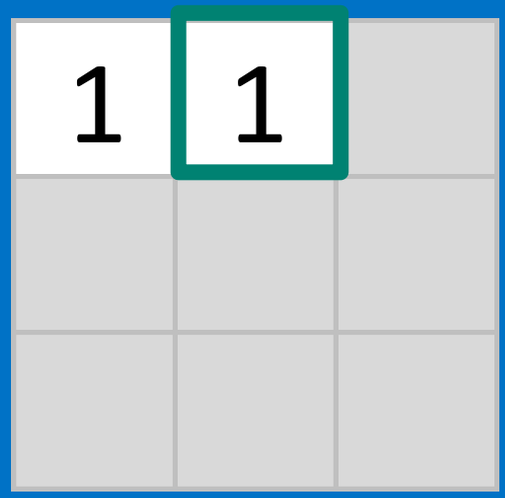


Filtering: The math behind the match

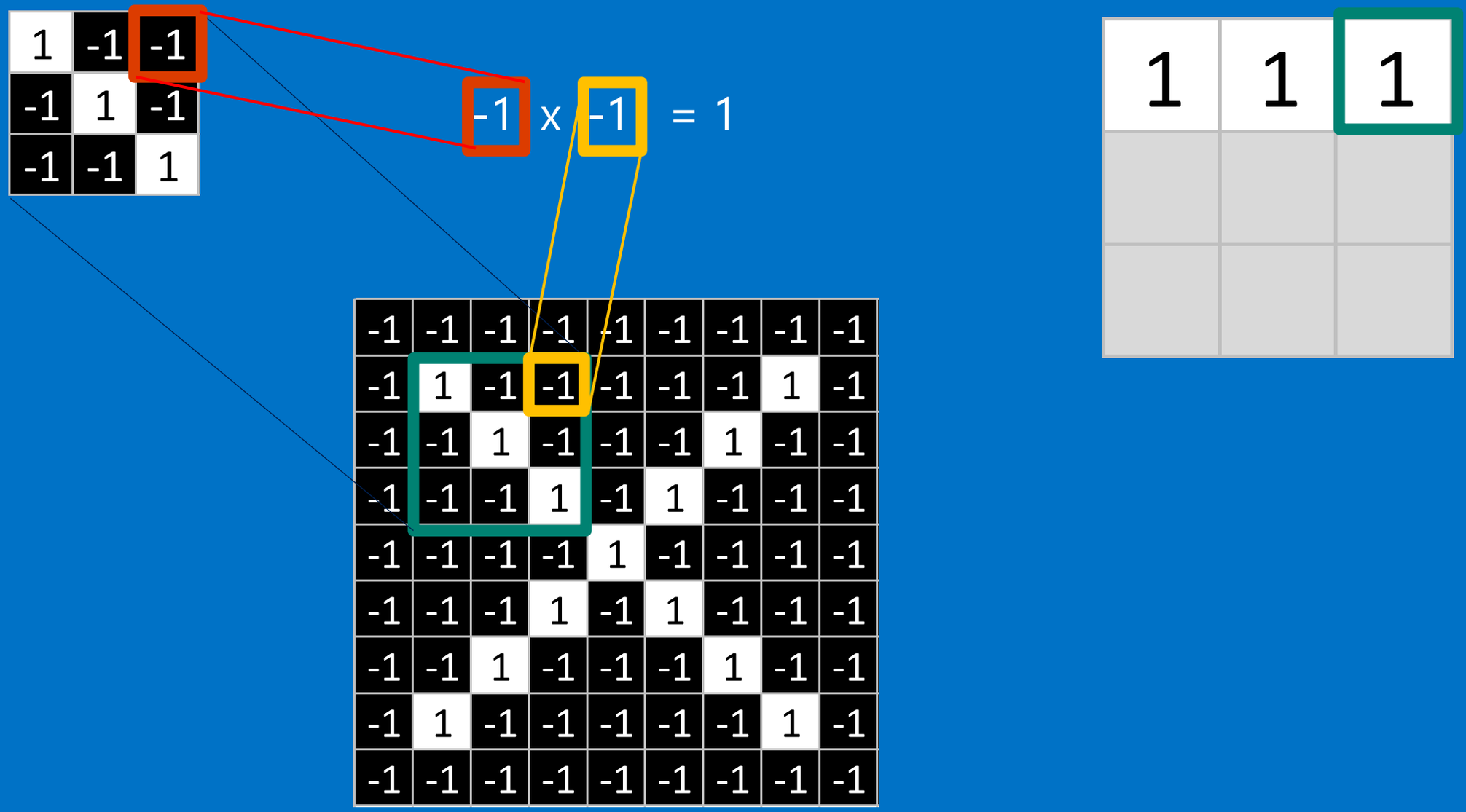


Filtering: The math behind the match

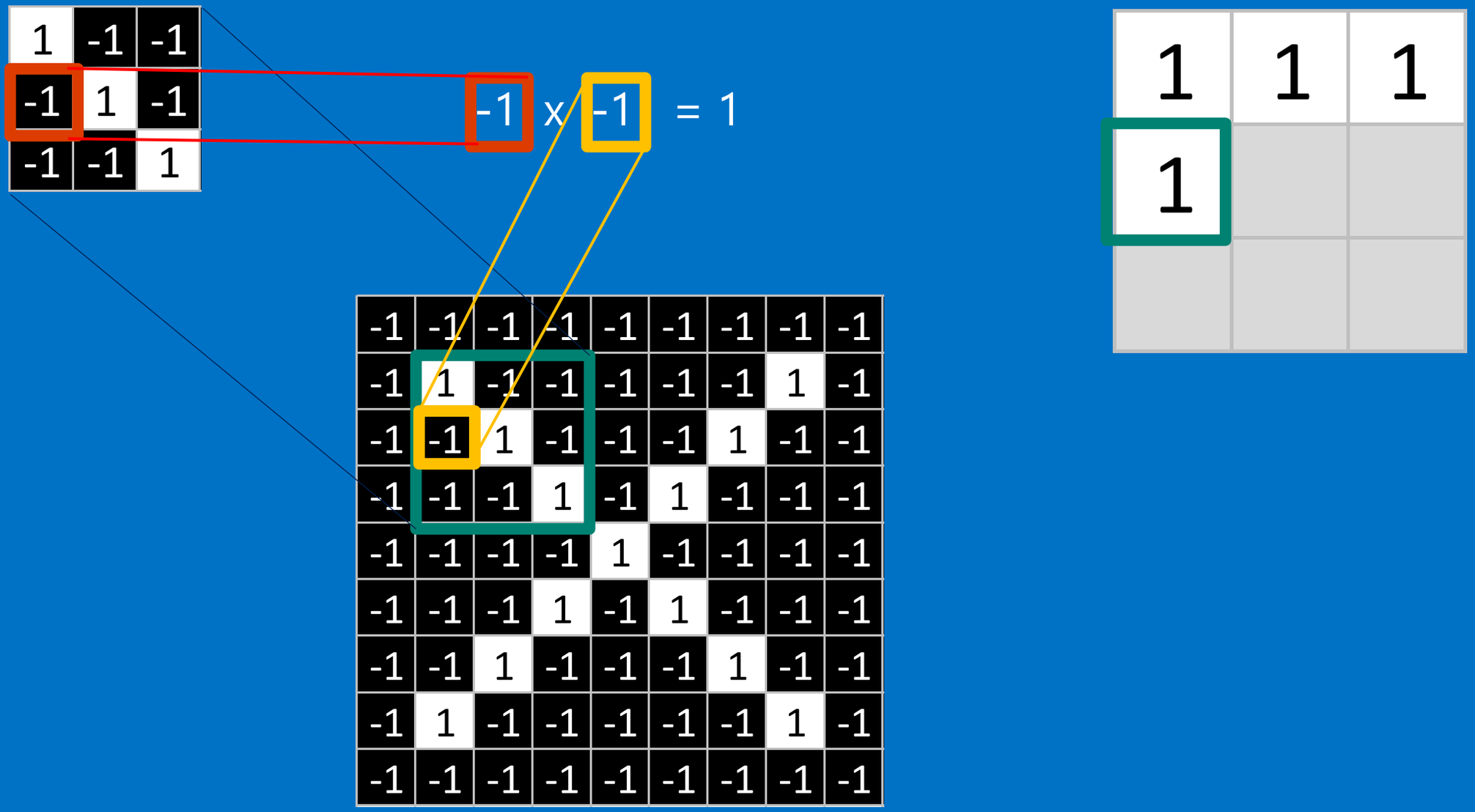




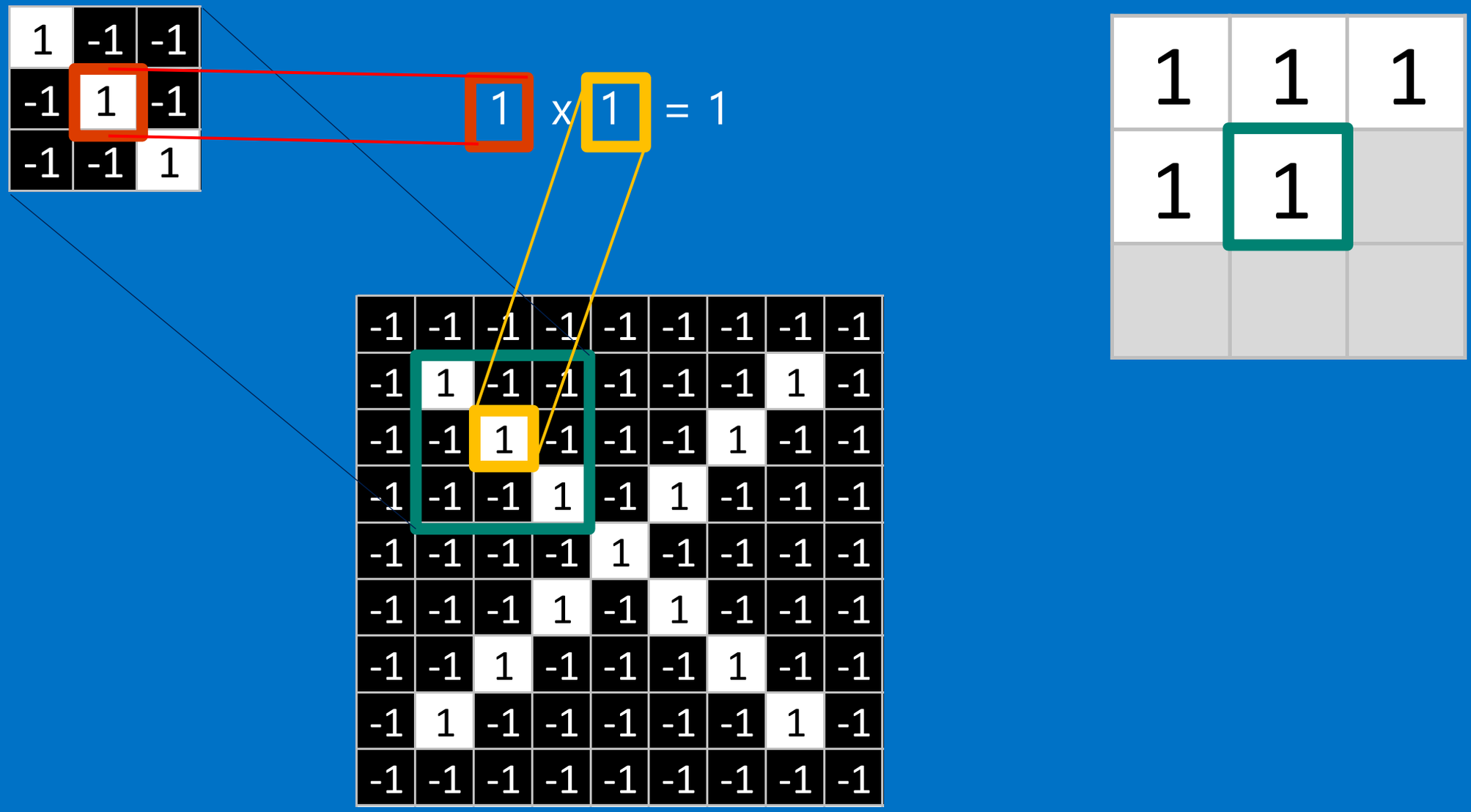
Filtering: The math behind the match



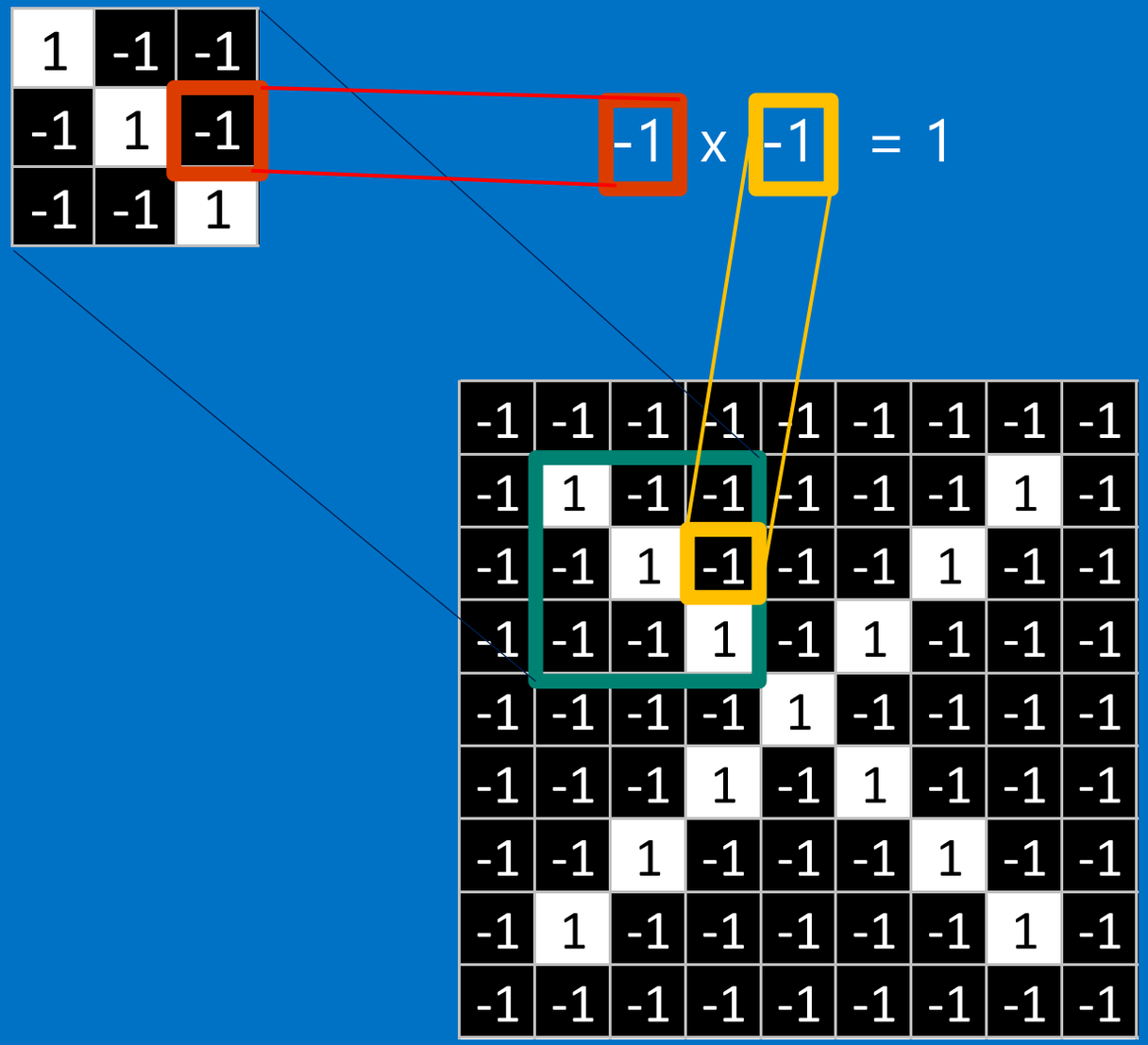
Filtering: The math behind the match

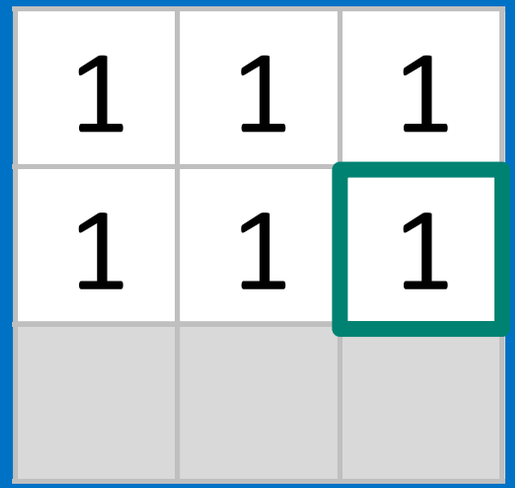


Filtering: The math behind the match

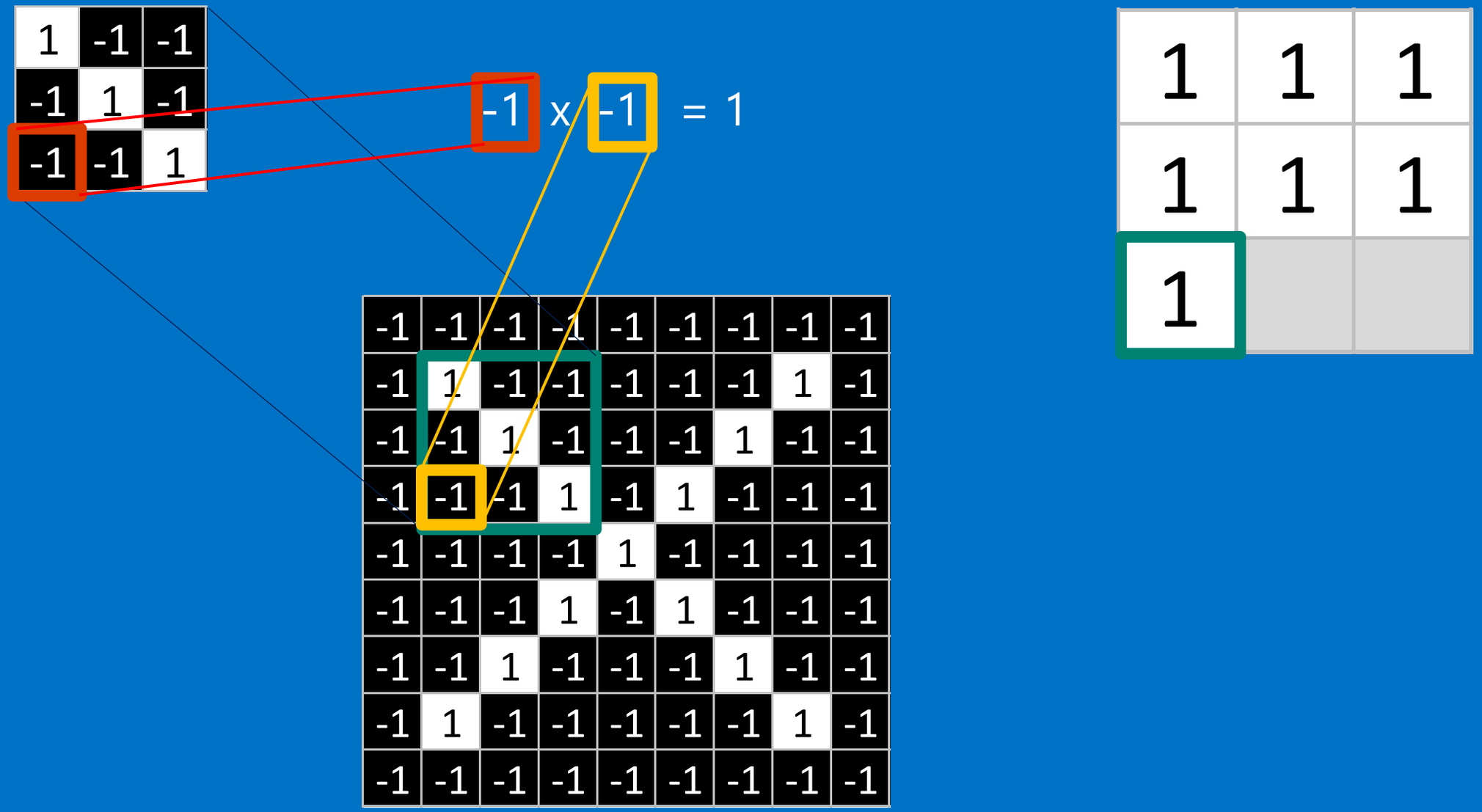


Filtering: The math behind the match

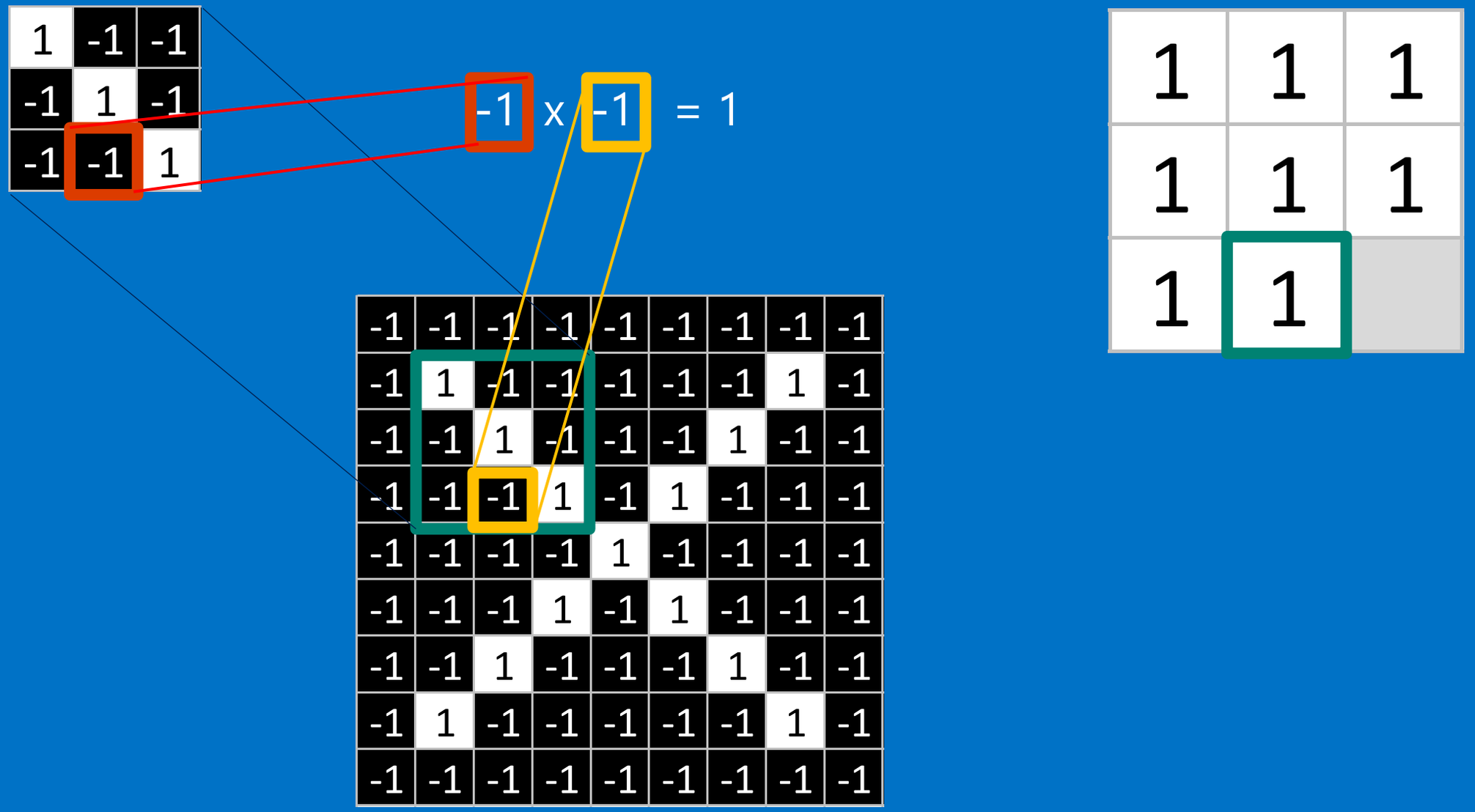




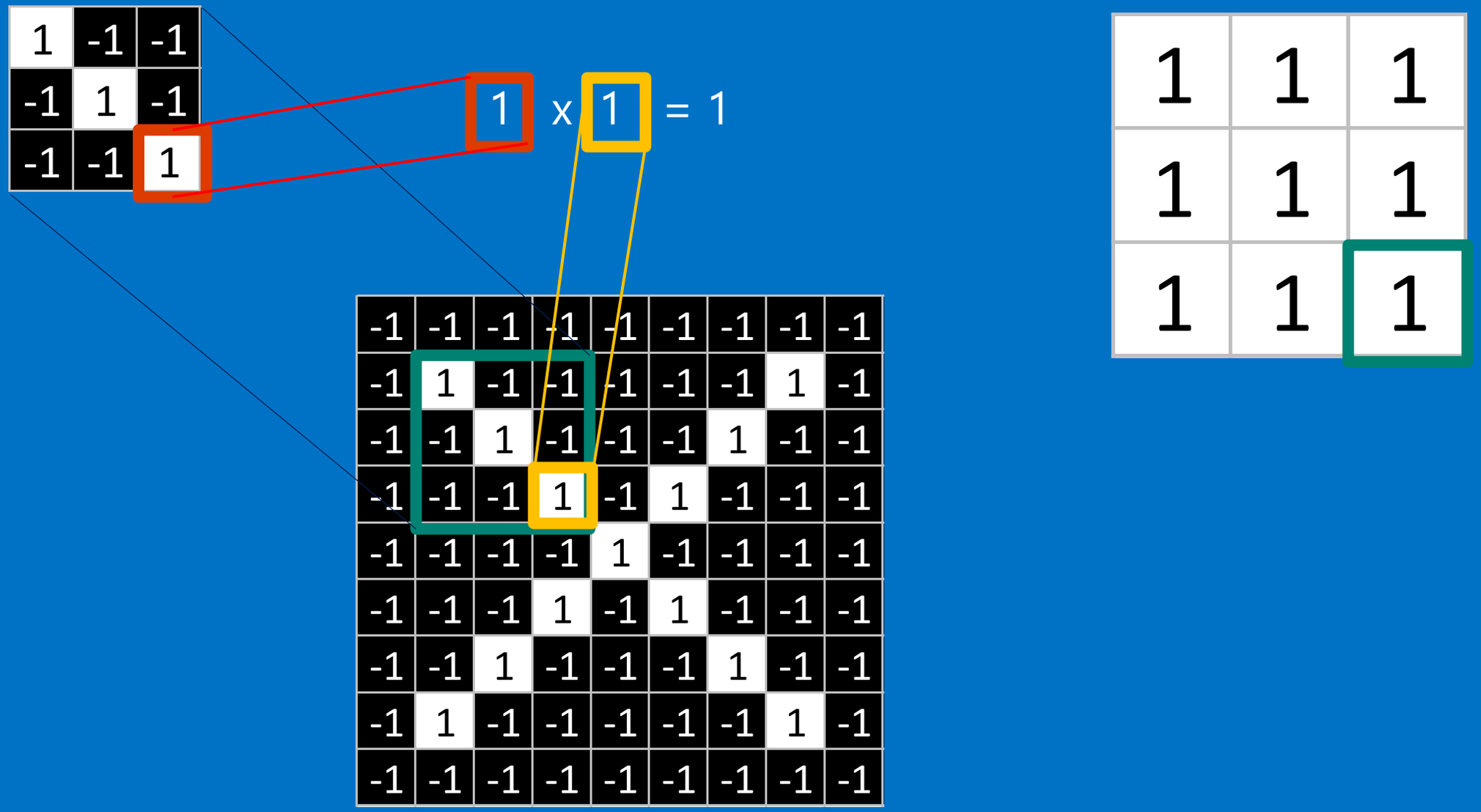
Filtering: The math behind the match



Filtering: The math behind the match



Filtering: The math behind the match



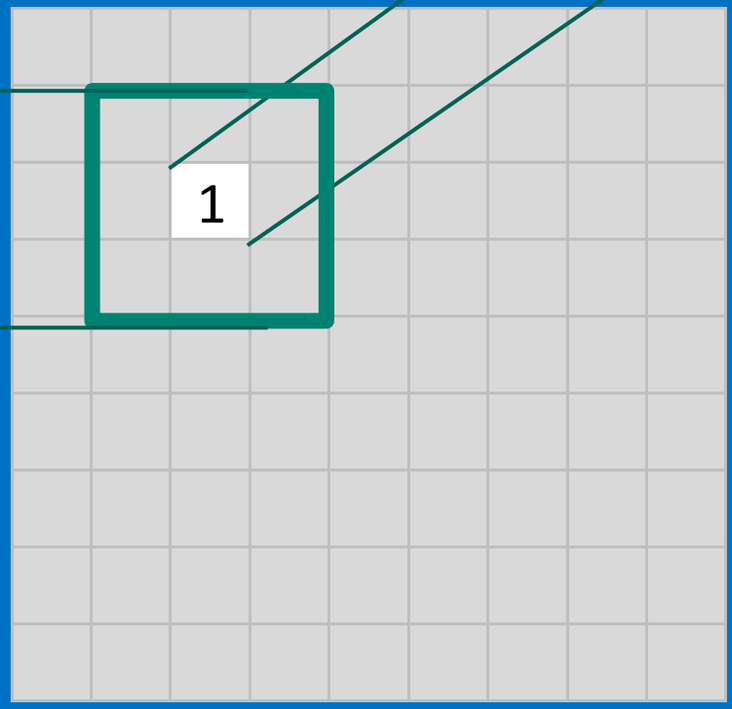
Filtering: The math behind the match

1	-1	-1
-1	1	-1
-1	-1	1

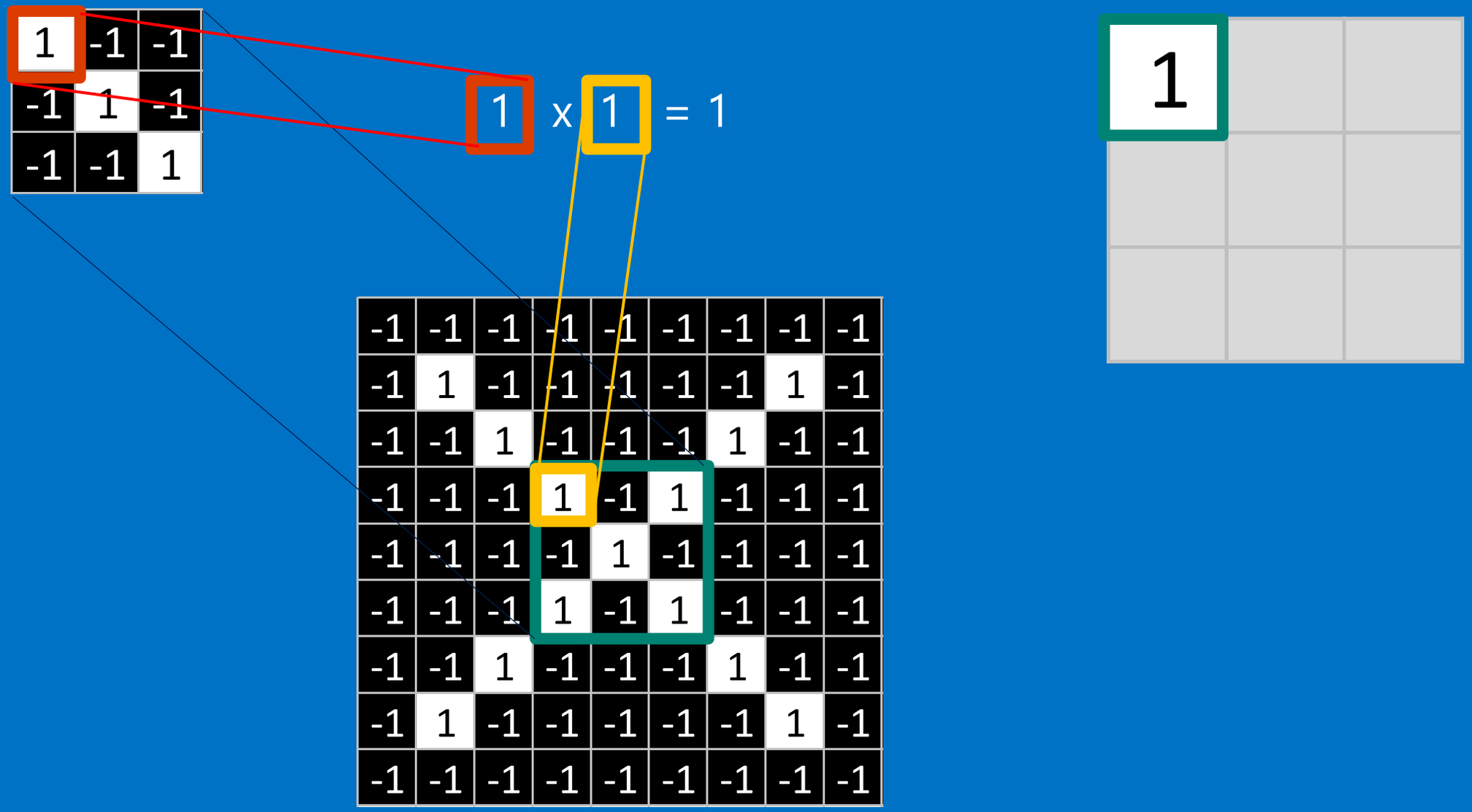
1	1	1
1	1	1
1	1	1

$$\frac{1+1+1+1+1+1+1+1+1}{9} = 1$$

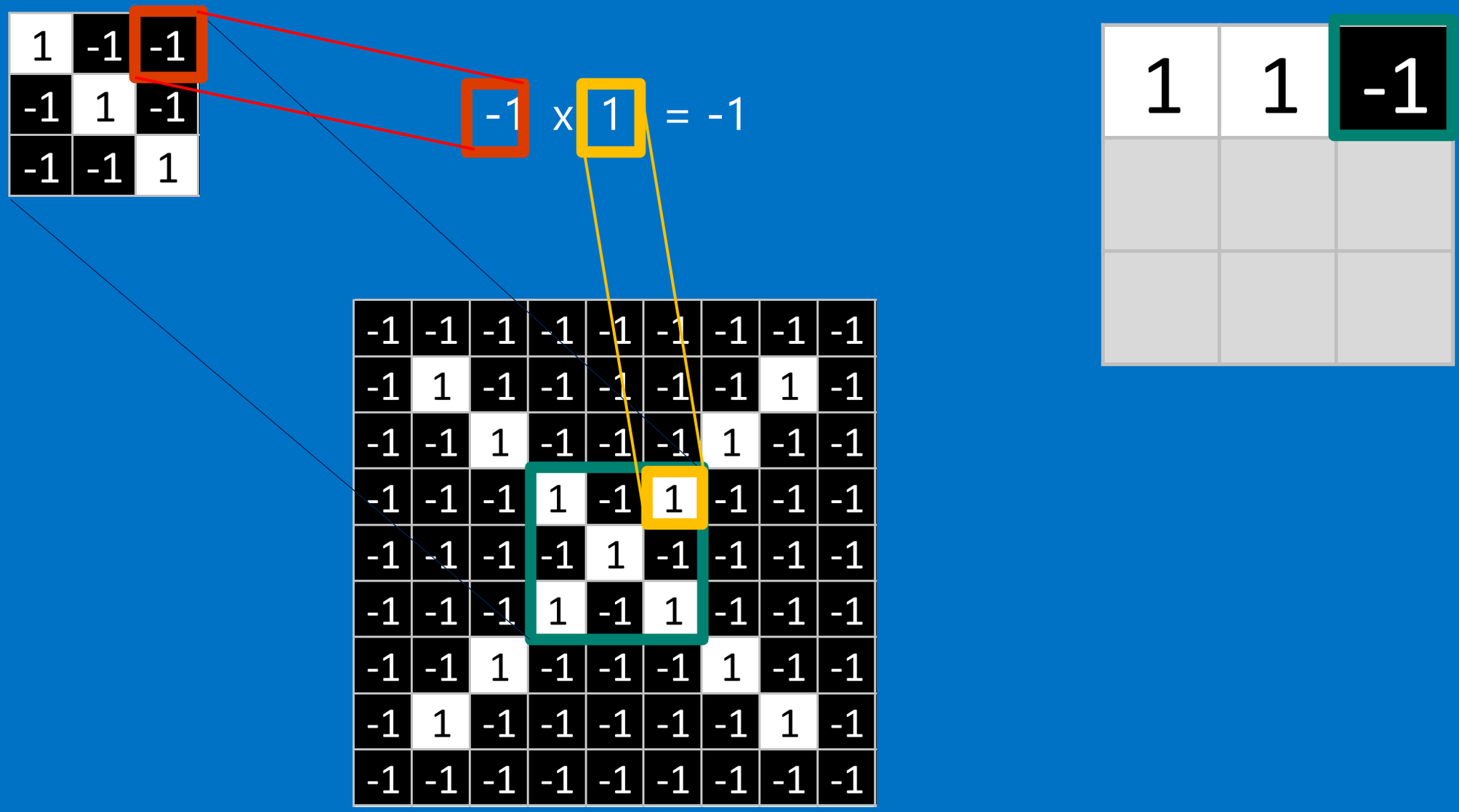
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



Filtering: The math behind the match



Filtering: The math behind the match



Filtering: The math behind the match

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	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	-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	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

1	1	-1
1	1	1
-1	1	1

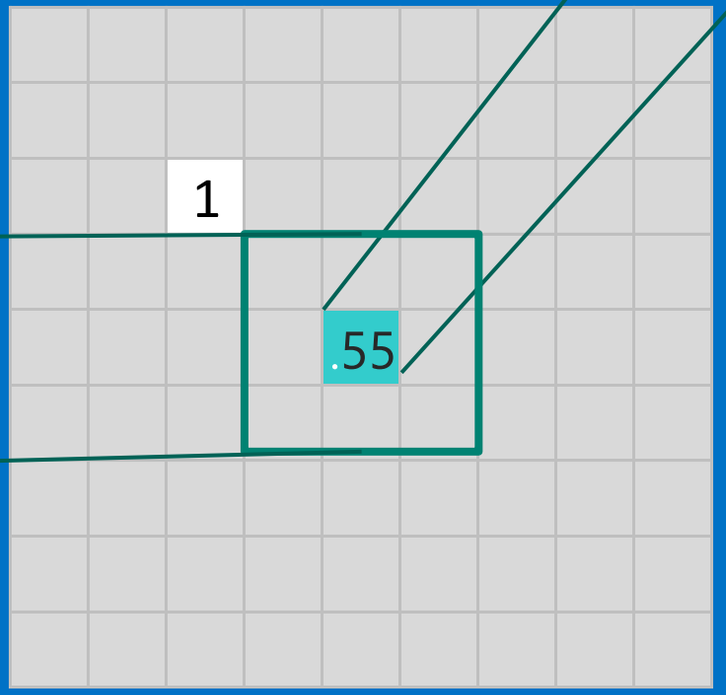
Filtering: The math behind the match

1	-1	-1
-1	1	-1
-1	-1	1

1	1	-1
1	1	1
-1	1	1

$$\frac{1+1-1+1+1+1-1+1+1}{9} = .55$$

-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	-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	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	-1	-1	-1	-1	-1	-1



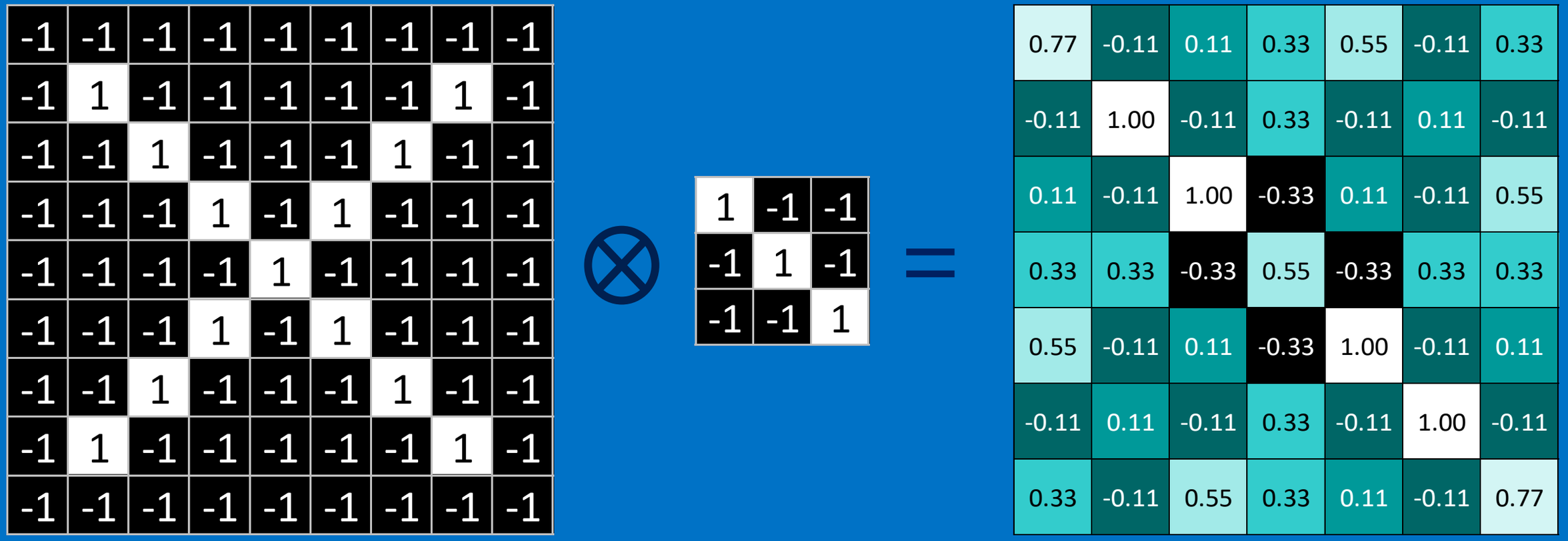
Convolution: Trying every possible match

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	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	-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	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

Convolution: Trying every possible match



-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	-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	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	-1	-1	-1	-1	-1	-1

⊗

1	-1	-1
-1	1	-1
-1	-1	1

=

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

-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	-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	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	-1	-1	-1	-1	-1	-1

⊗

1	-1	1
-1	1	-1
1	-1	1

=

0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.11	0.33	-0.77	1.00	-0.77	0.33	-0.11
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33

-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	-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	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	-1	-1	-1	-1	-1	-1

⊗

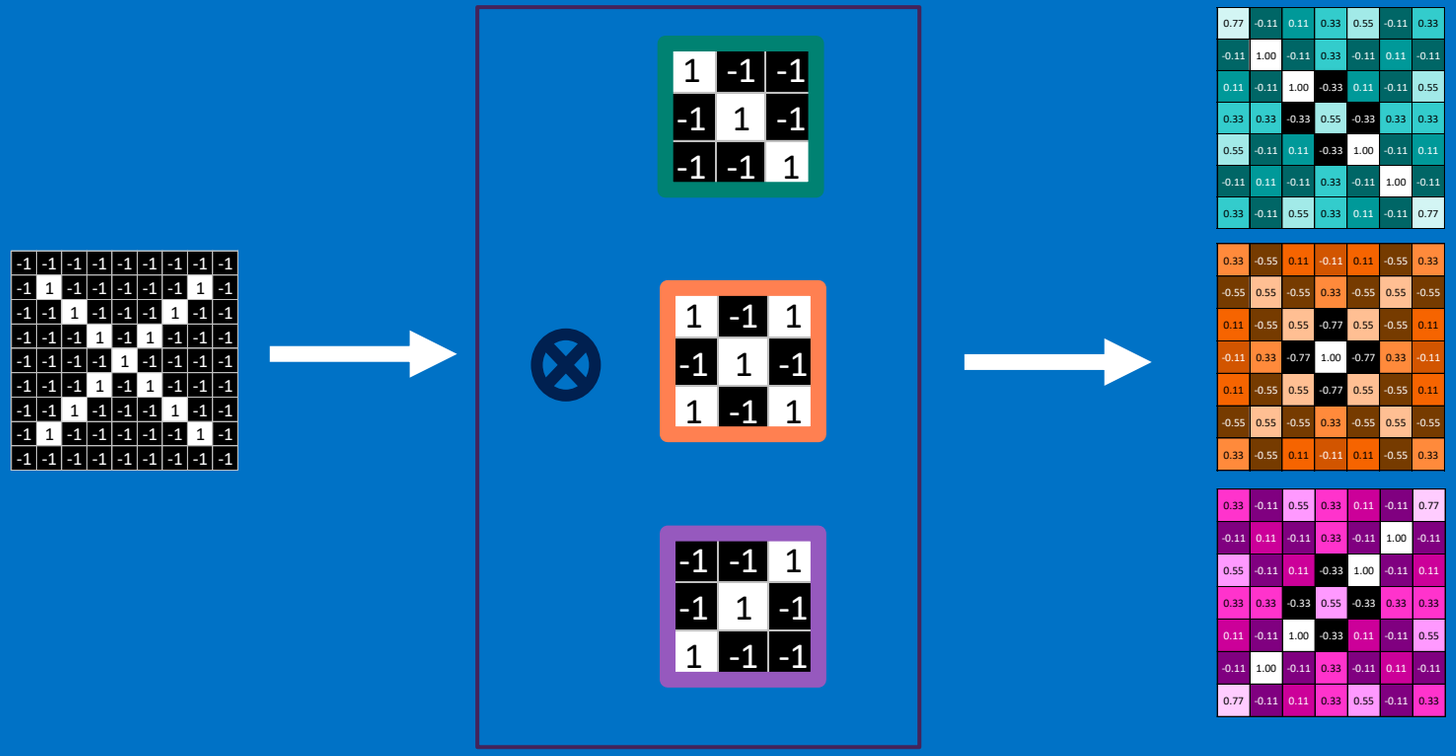
-1	-1	1
-1	1	-1
1	-1	-1

=

0.33	-0.11	0.55	0.33	0.11	-0.11	0.77
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33

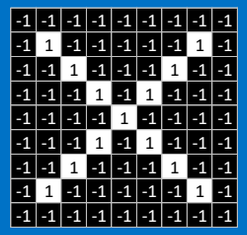
Convolution layer

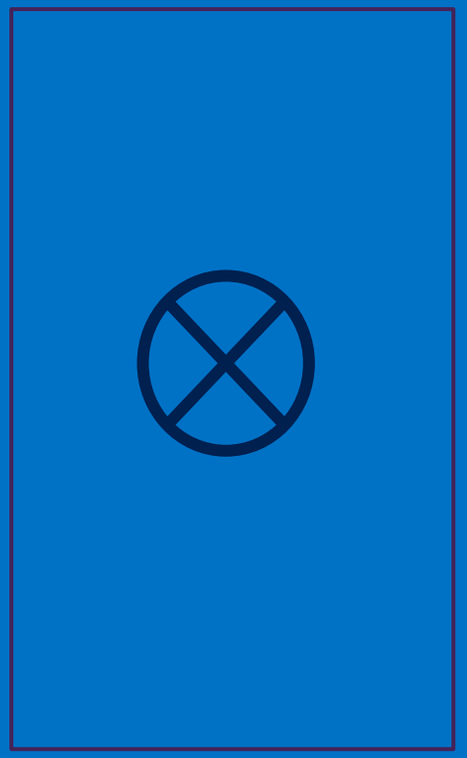
One image becomes a stack of filtered images



Convolution layer

One image becomes a stack of filtered images





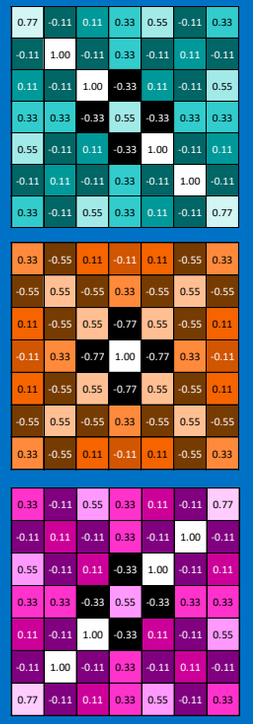
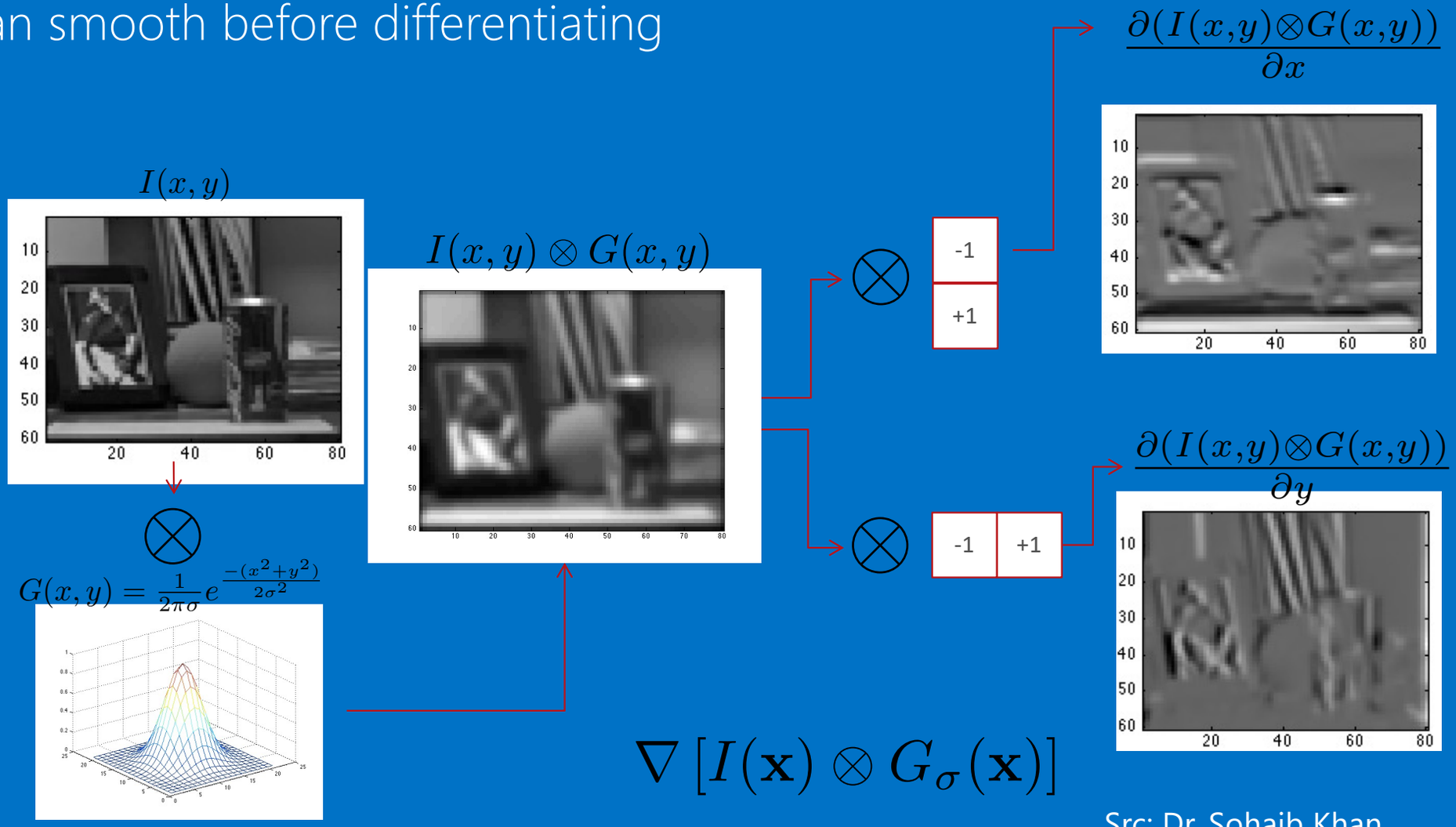


Image Gradient

- Derivatives are sensitive to noise
  so we can smooth before differentiating

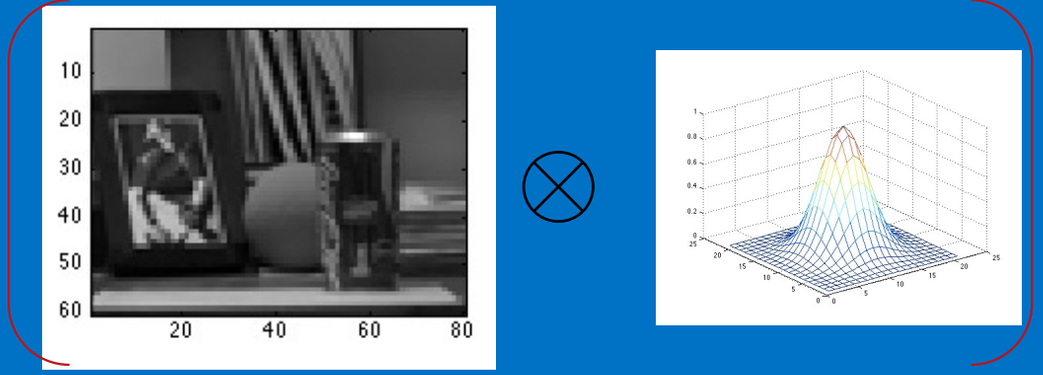


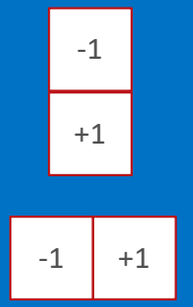
Src: Dr. Sohaib Khan

Image Gradient

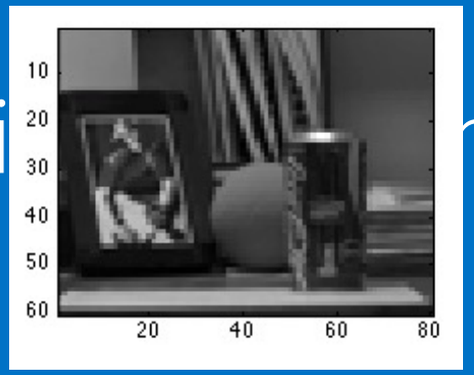
- But convolution is associative operation.

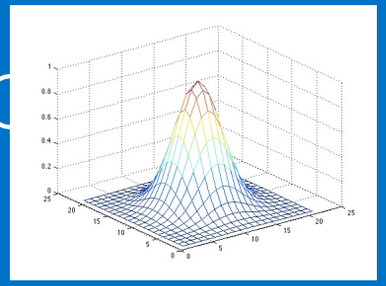
$$\nabla [I(x,y)) \otimes G_\sigma(x,y)]$$

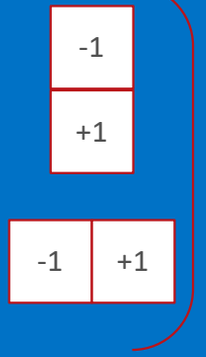




$$I(x,y) \otimes \nabla G_\sigma(x,y)$$



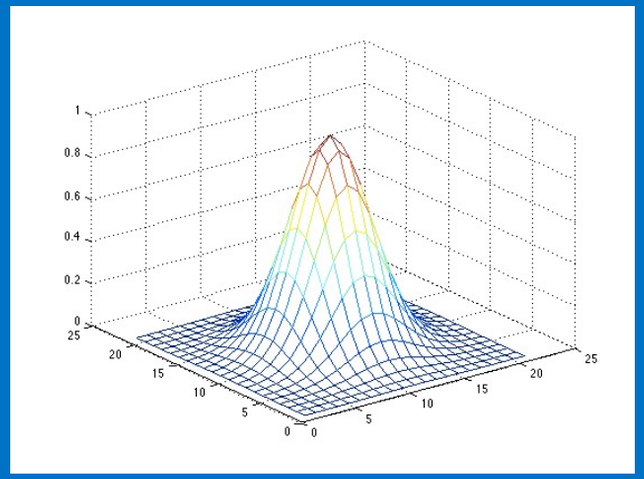




Src: Dr. Sohaib Khan

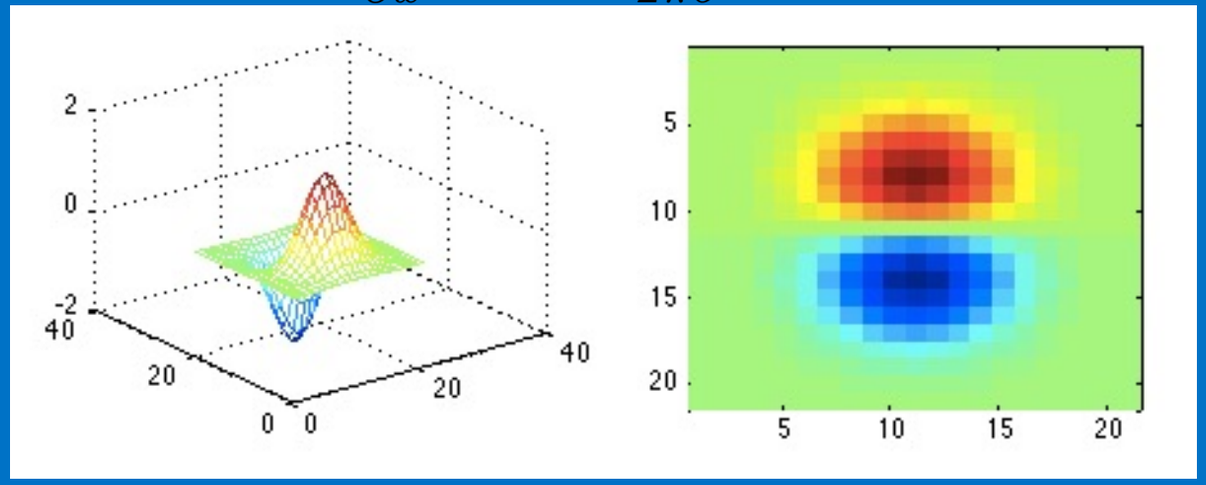
Image Gradient

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

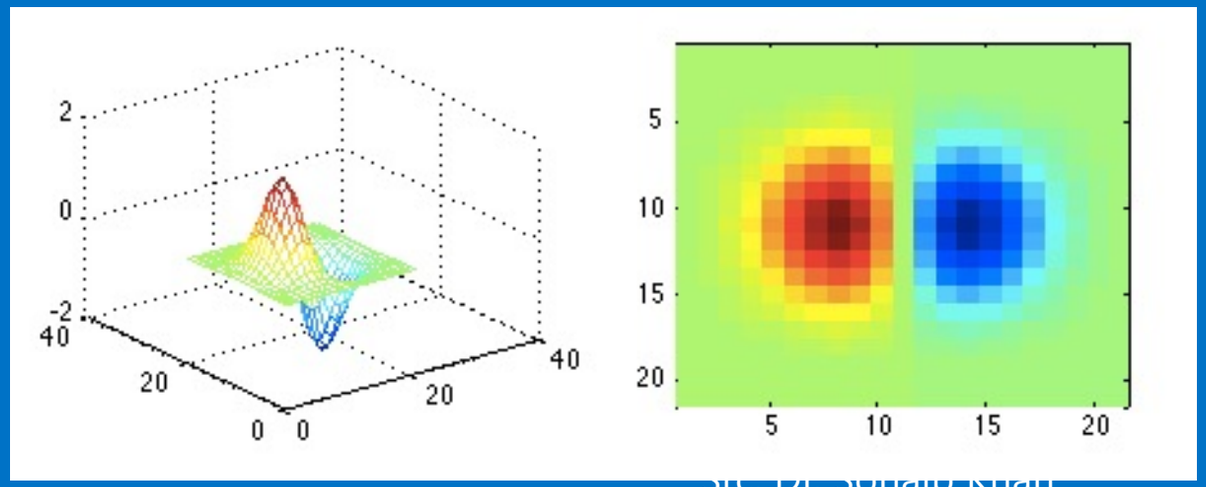


$$\nabla G(x,y) = \begin{bmatrix} \frac{\partial G(x,y)}{\partial x} \\ \frac{\partial G(x,y)}{\partial y} \end{bmatrix}$$

$$\frac{\partial G(x,y)}{\partial x} = -\frac{x}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



$$\frac{\partial G(x,y)}{\partial y} = -\frac{y}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

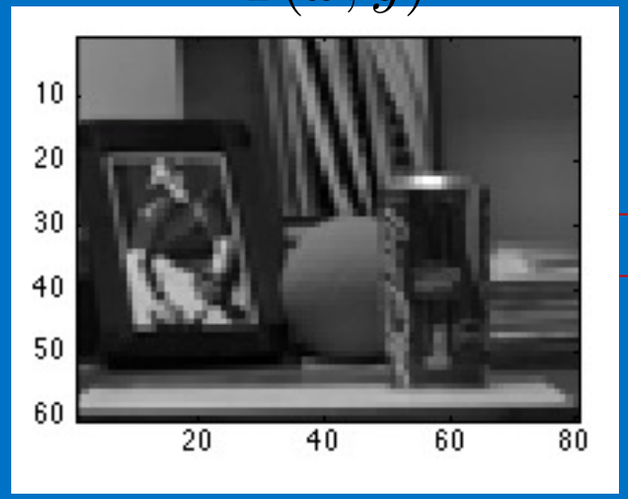


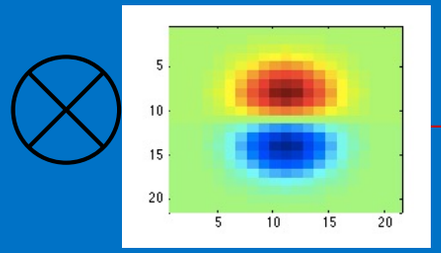
Src: Dr. Sohaib Khan

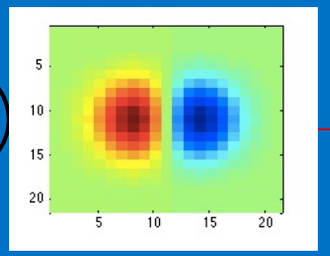
Image Gradient

- Computing Derivatives

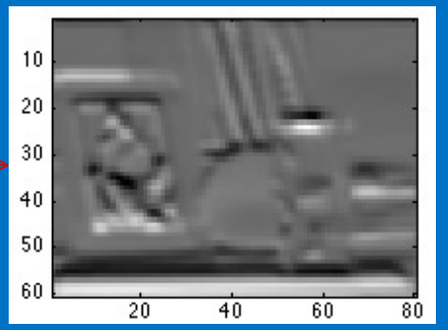
$I(x,y)$



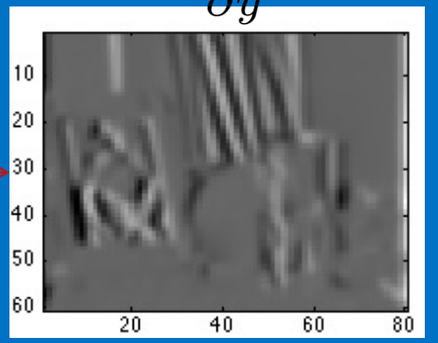




$\frac{\partial(I(x,y)\otimes G(x,y))}{\partial x}$



$\frac{\partial(I(x,y)\otimes G(x,y))}{\partial y}$



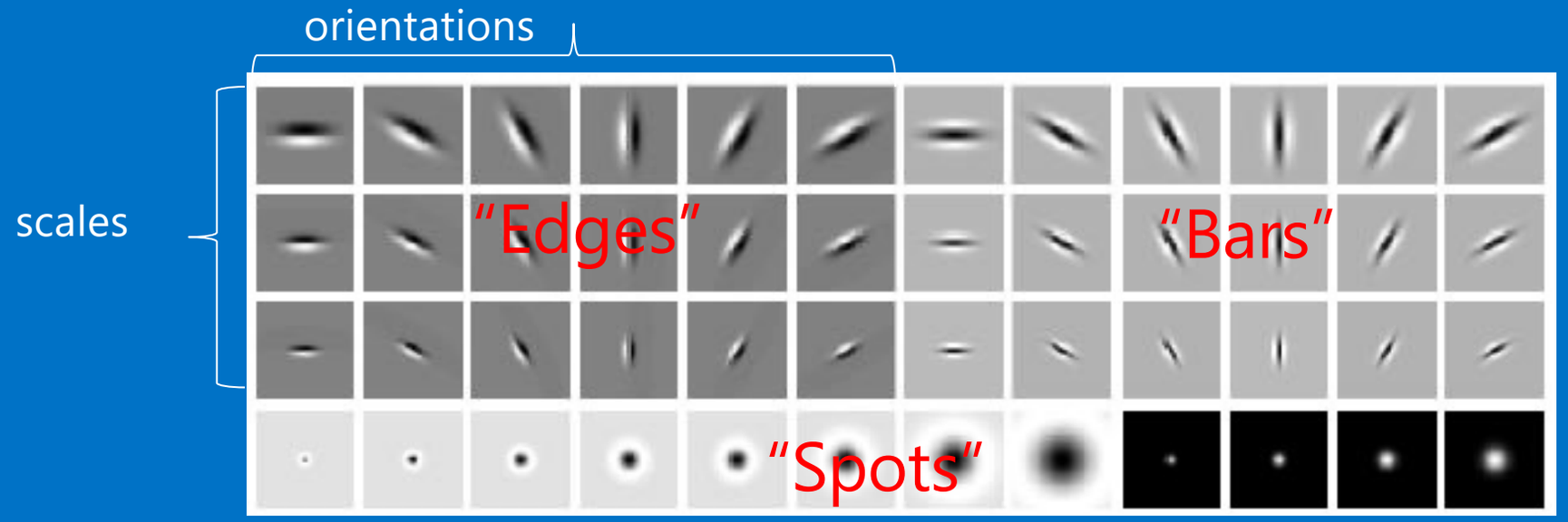
$$I(x,y) \otimes \nabla G_\sigma(x,y)$$

Src: Dr. Sohaib Khan

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.
  - **still can think of nearness, farness in feature space**

Filter banks

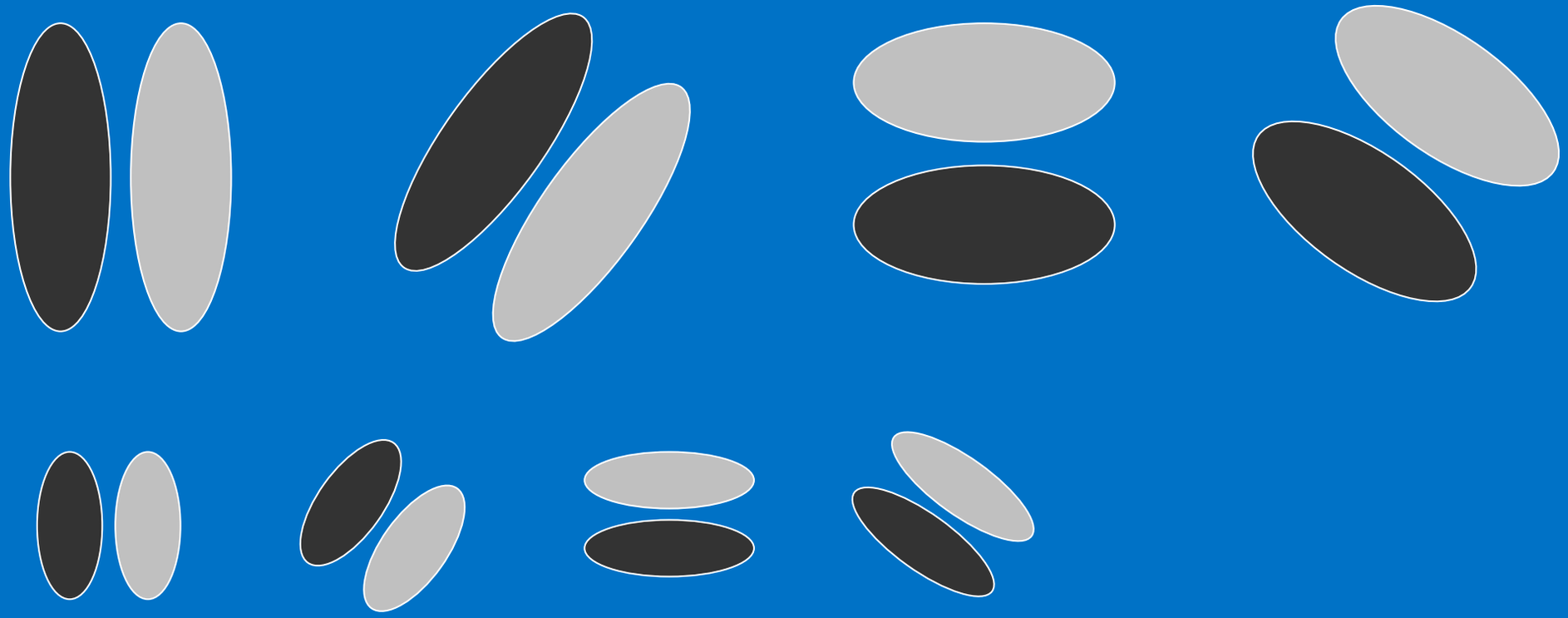


- What filters to put in the bank?
  - Typically we want a combination of scales and orientations, different types of patterns.

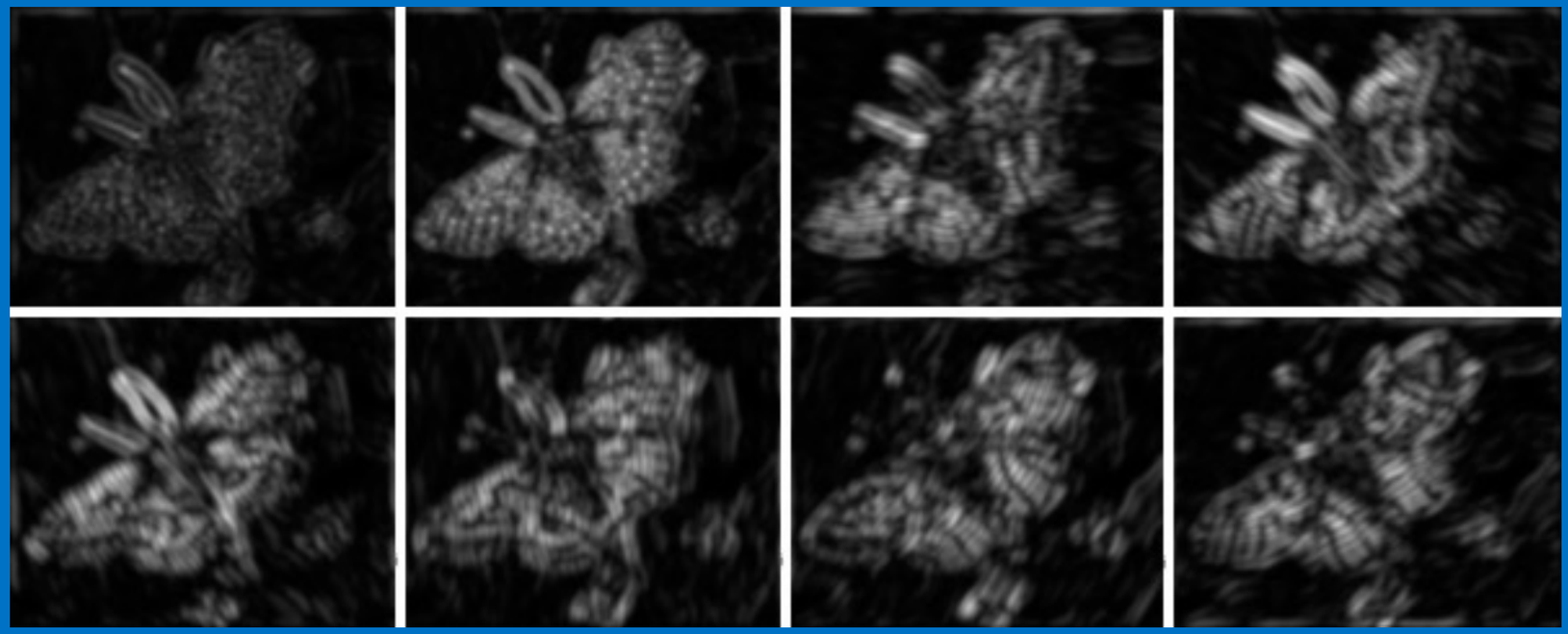
Matlab code available for these examples:
http://www.robots.ox.ac.uk/~vgg/research/texclass/filters.html

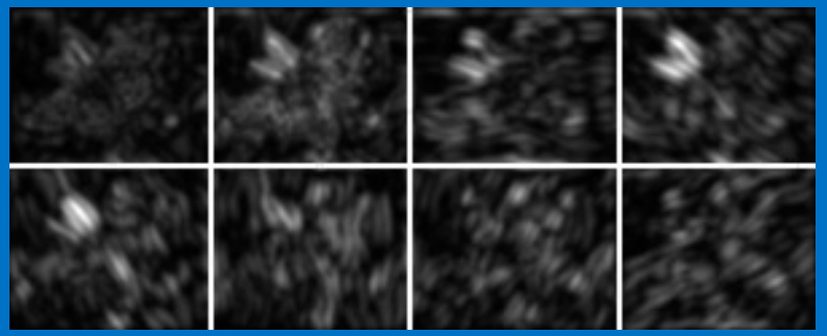
Learn: use lots of filters, multi-ori&scale.

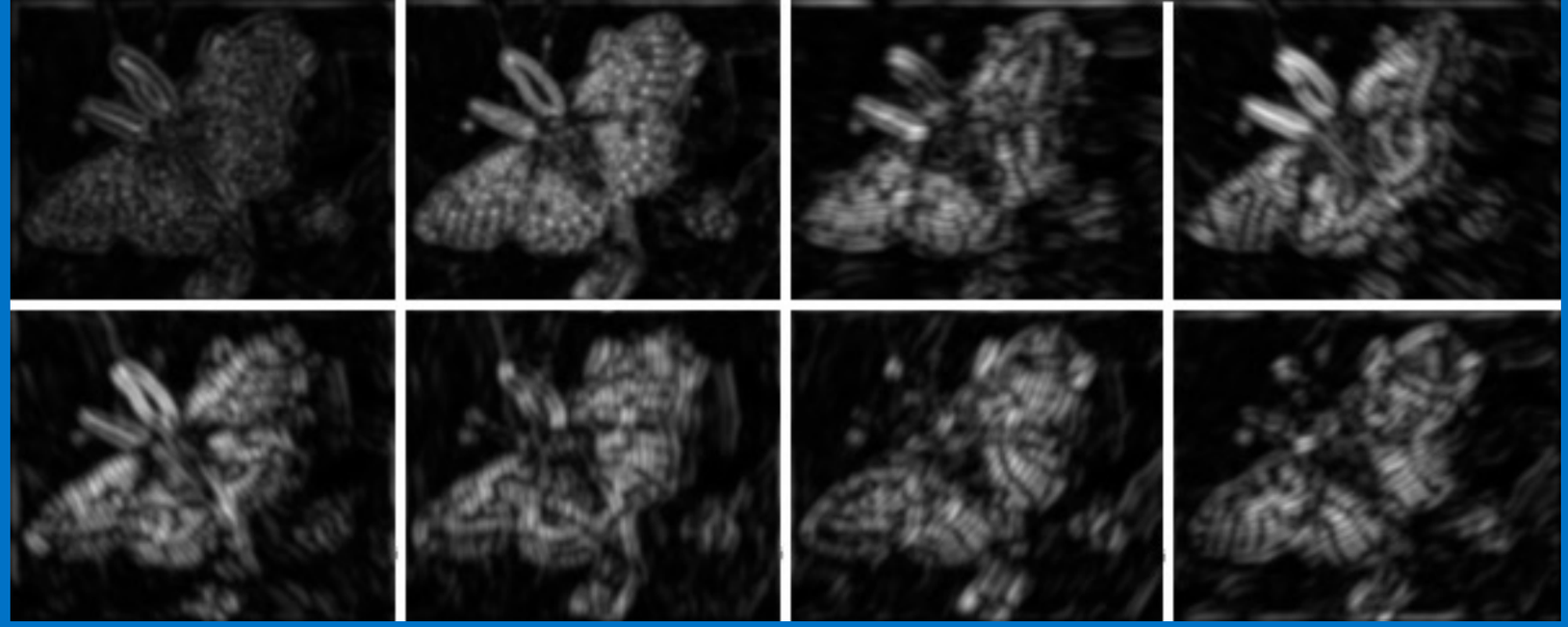
Malik and Perona

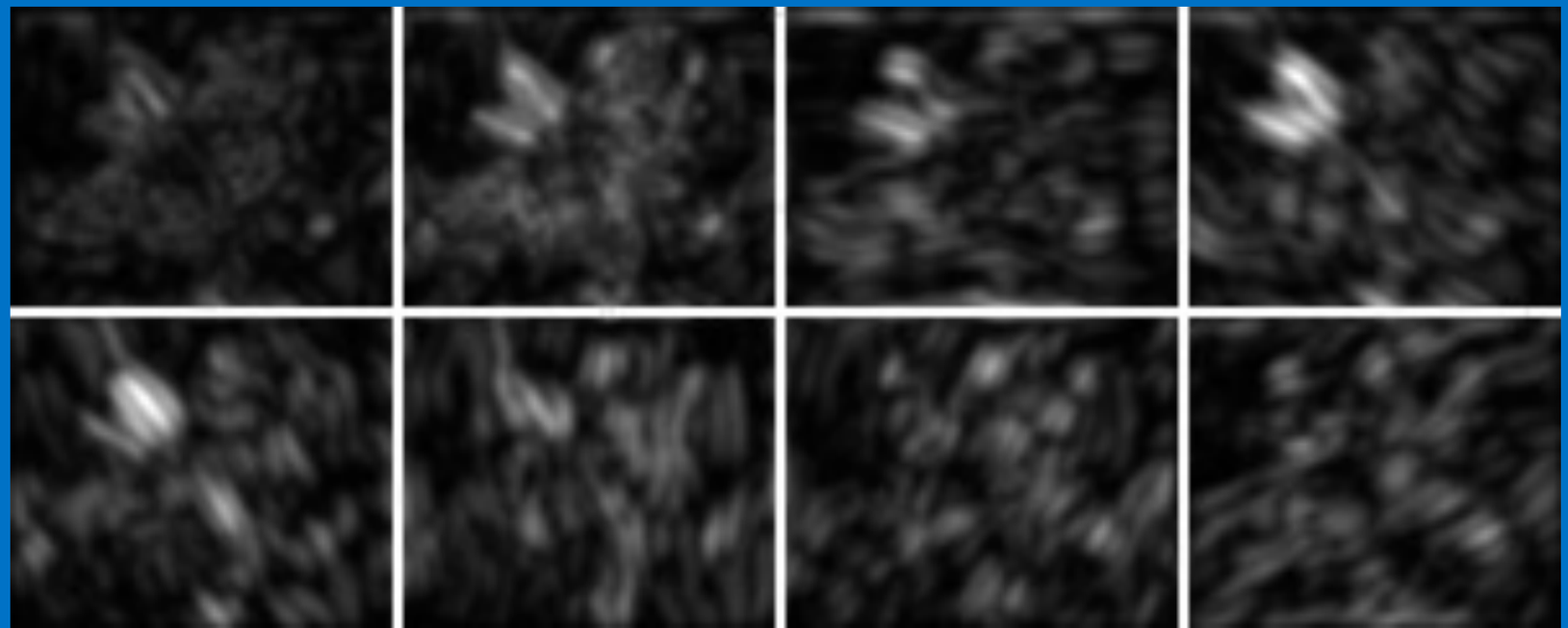


Malik J, Perona P. Preattentive texture discrimination with early vision mechanisms. J OPT SOC AM A 7: (5) 923-932 MAY 1990









Filter bank

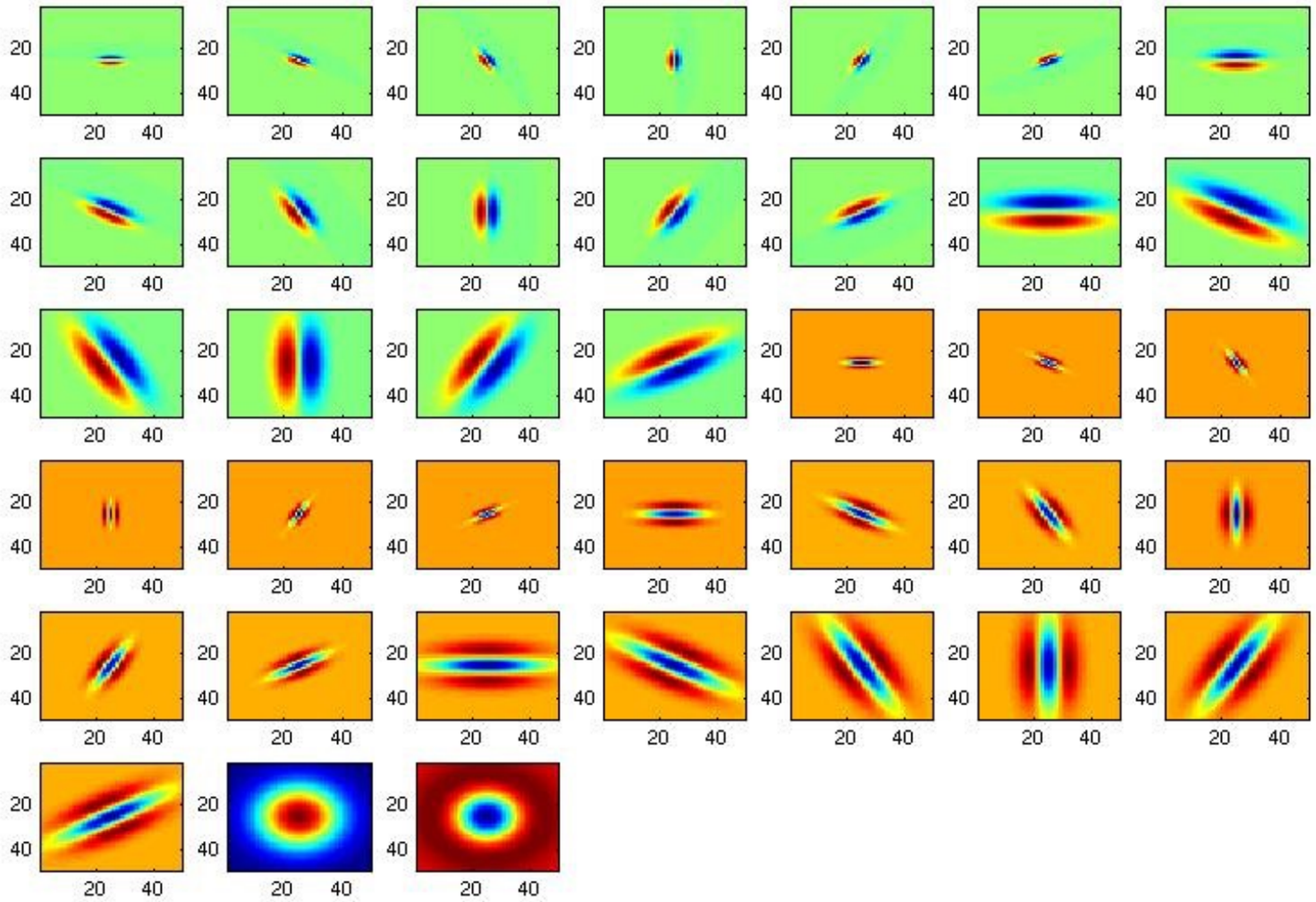
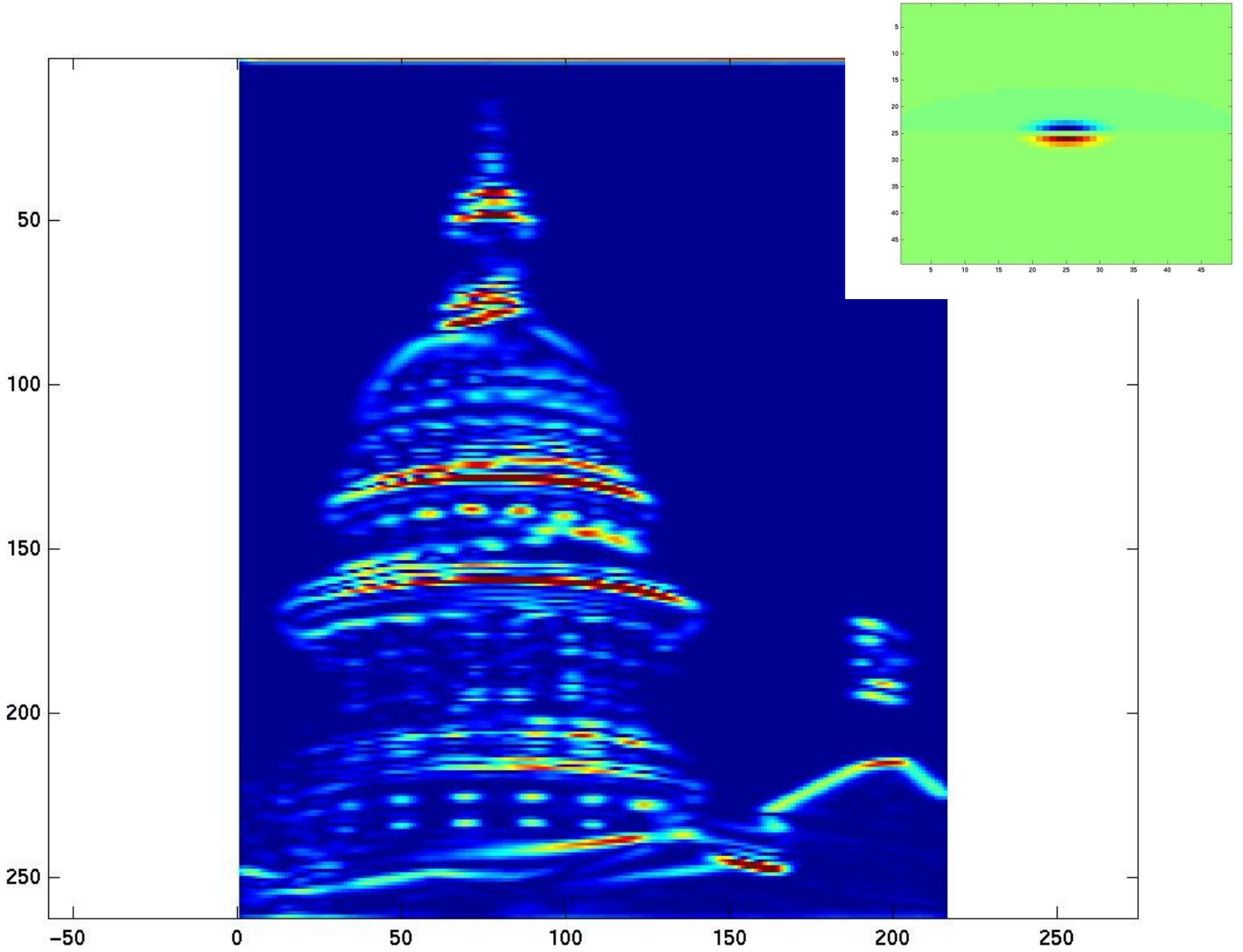
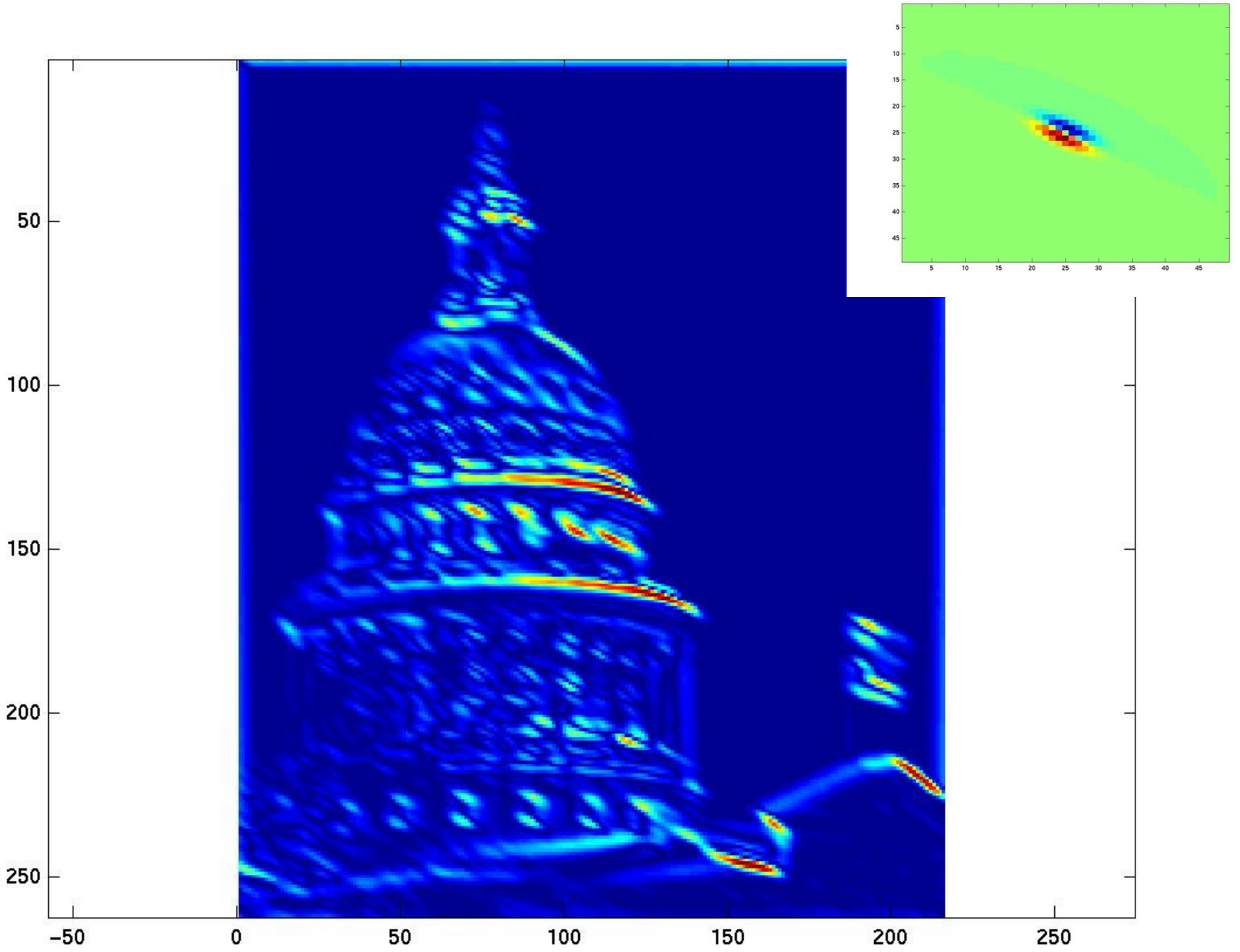
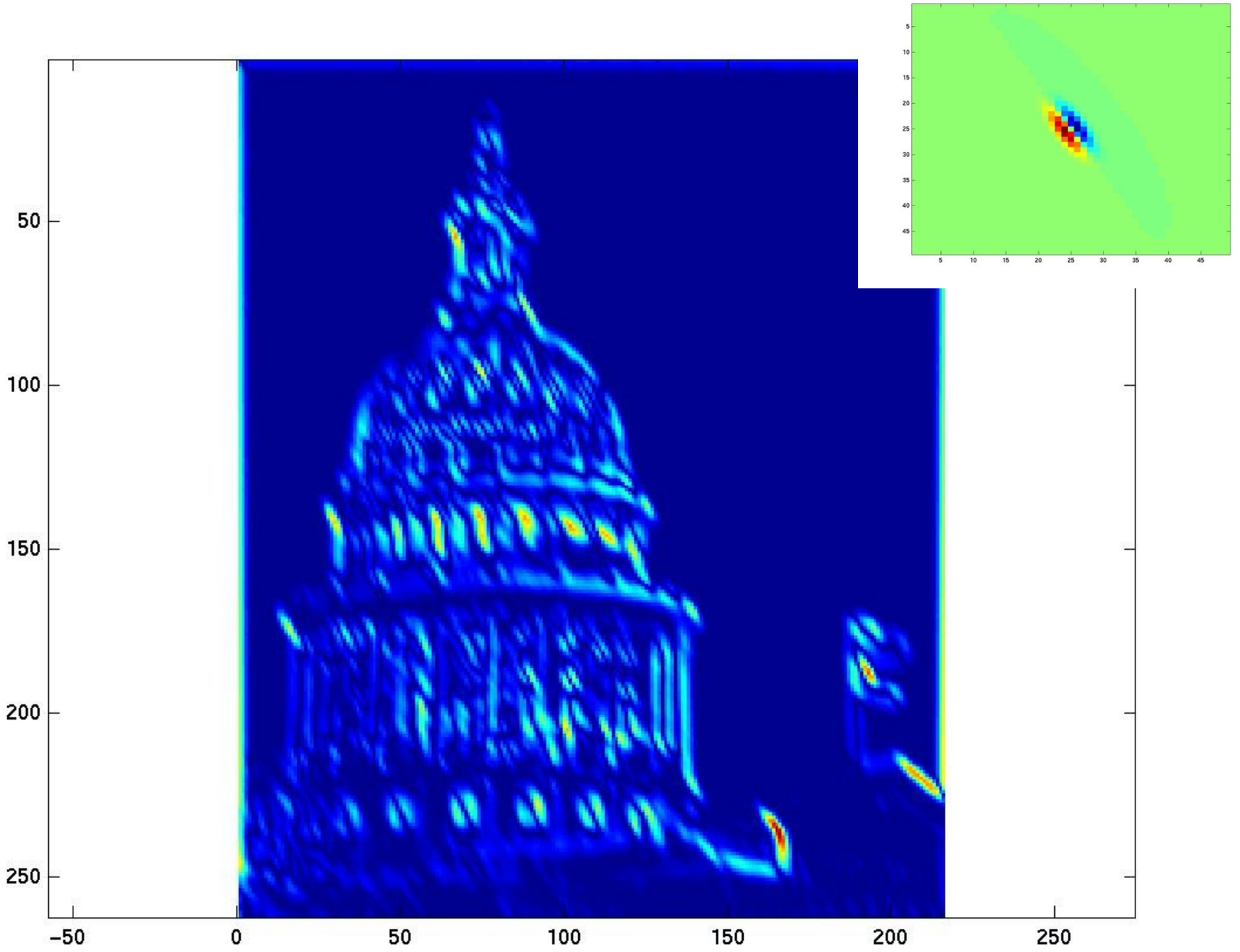


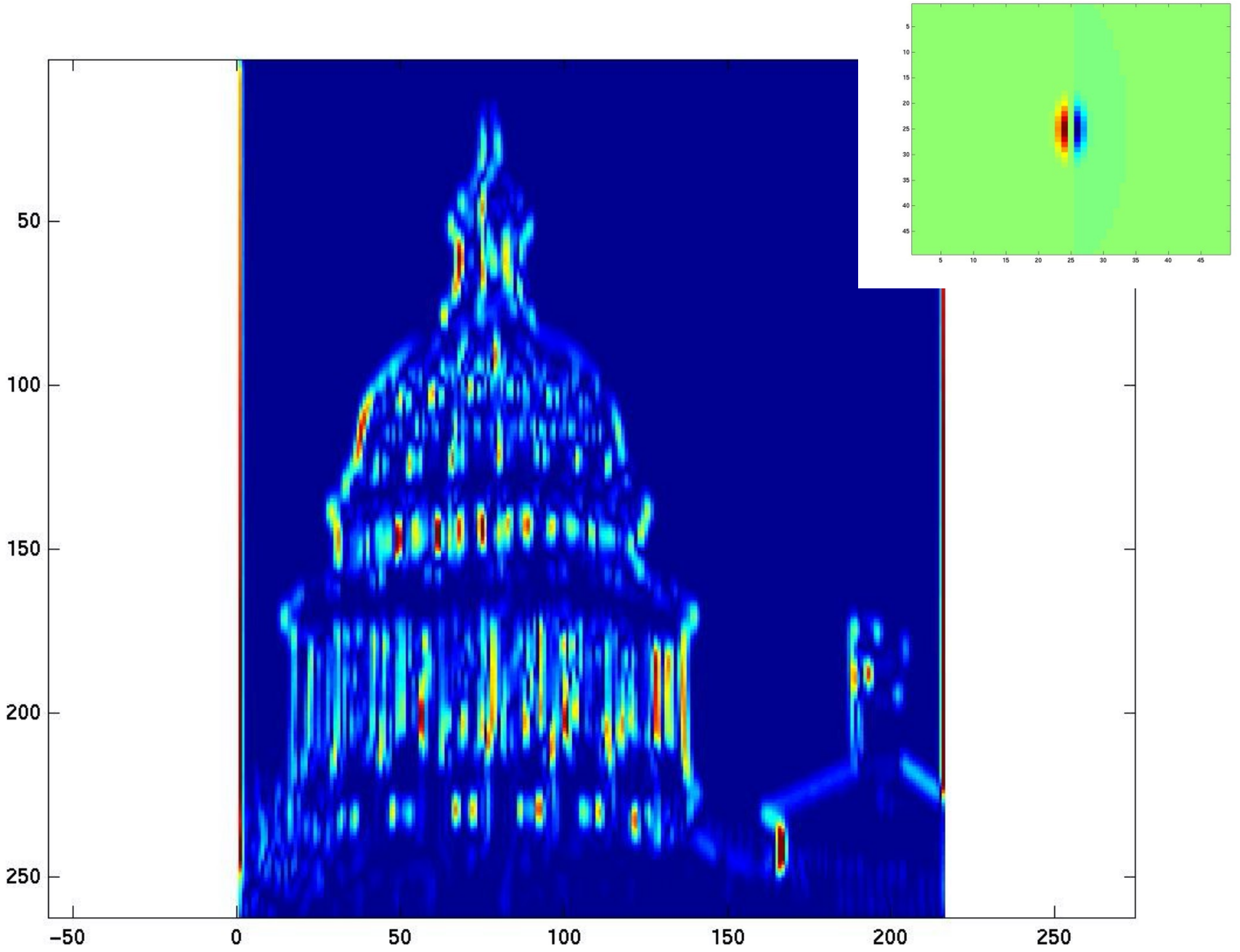
Image from http://www.texasexplorer.com/austincap2.jpg

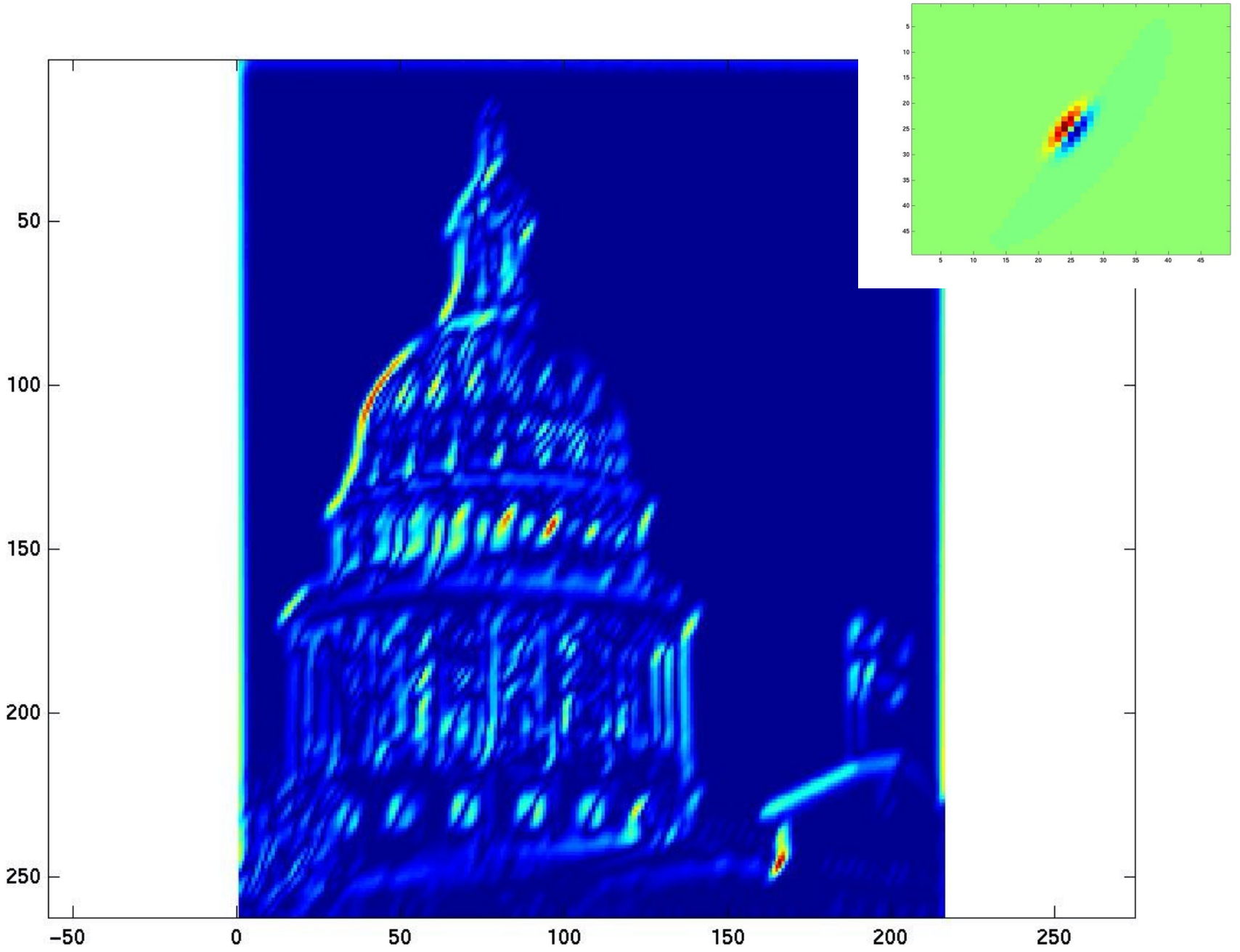
Kristen Grauman

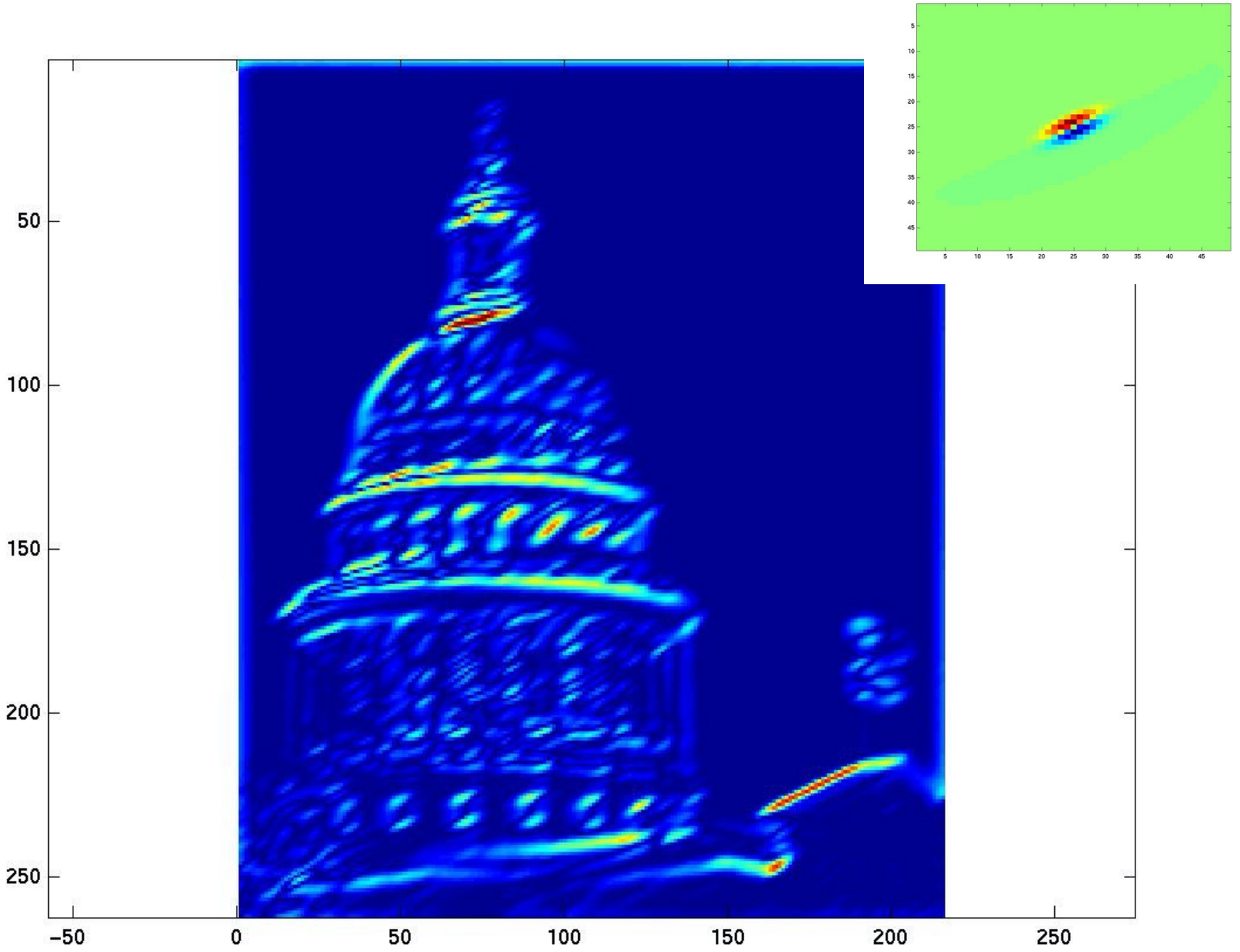

50
100
150
200
250
−50
0
50
100
150
200
250
5
10
15
20
25
30
35
40
45

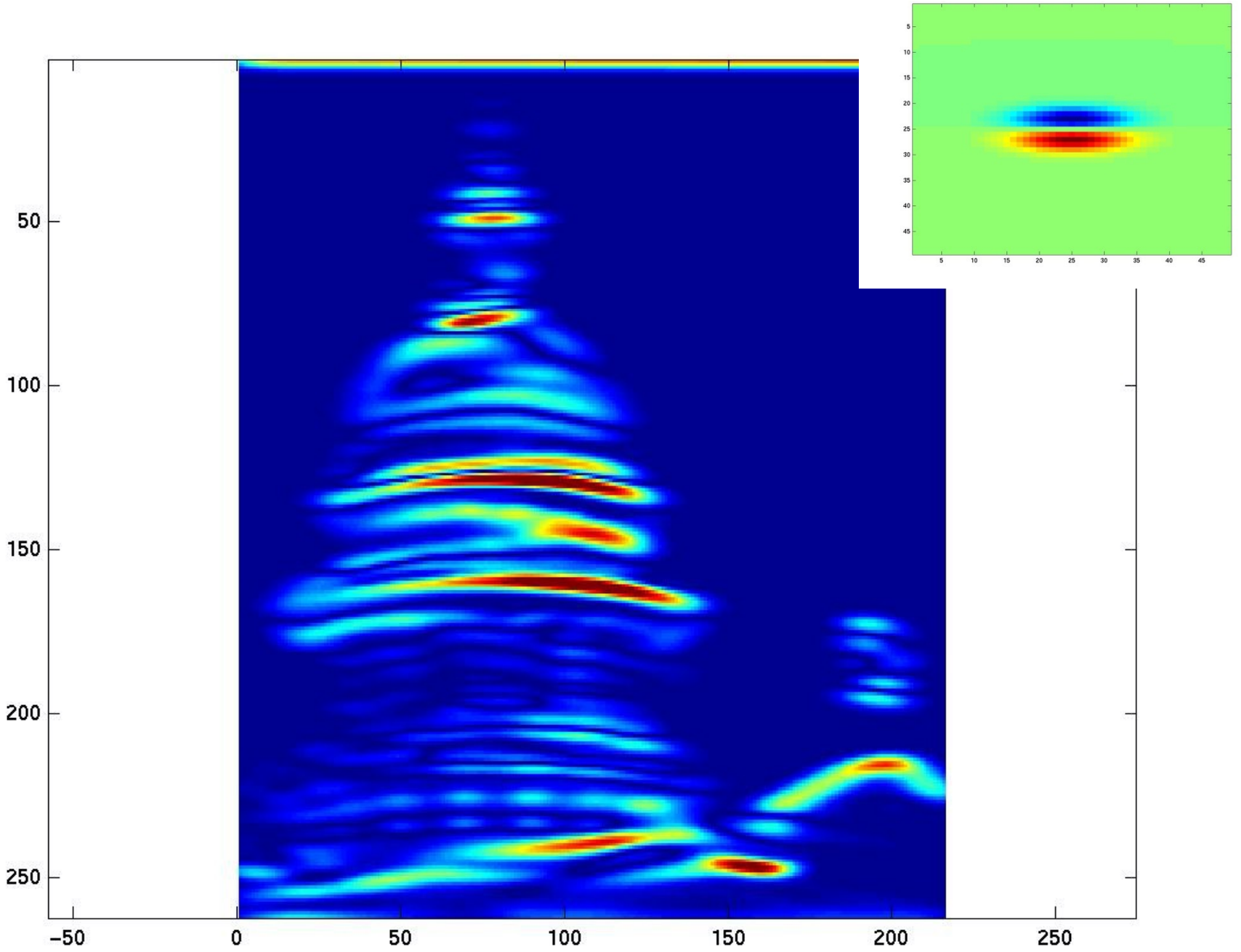

50
100
150
200
250
−50
0
50
100
150
200
250
5
10
15
20
25
30
35
40
45

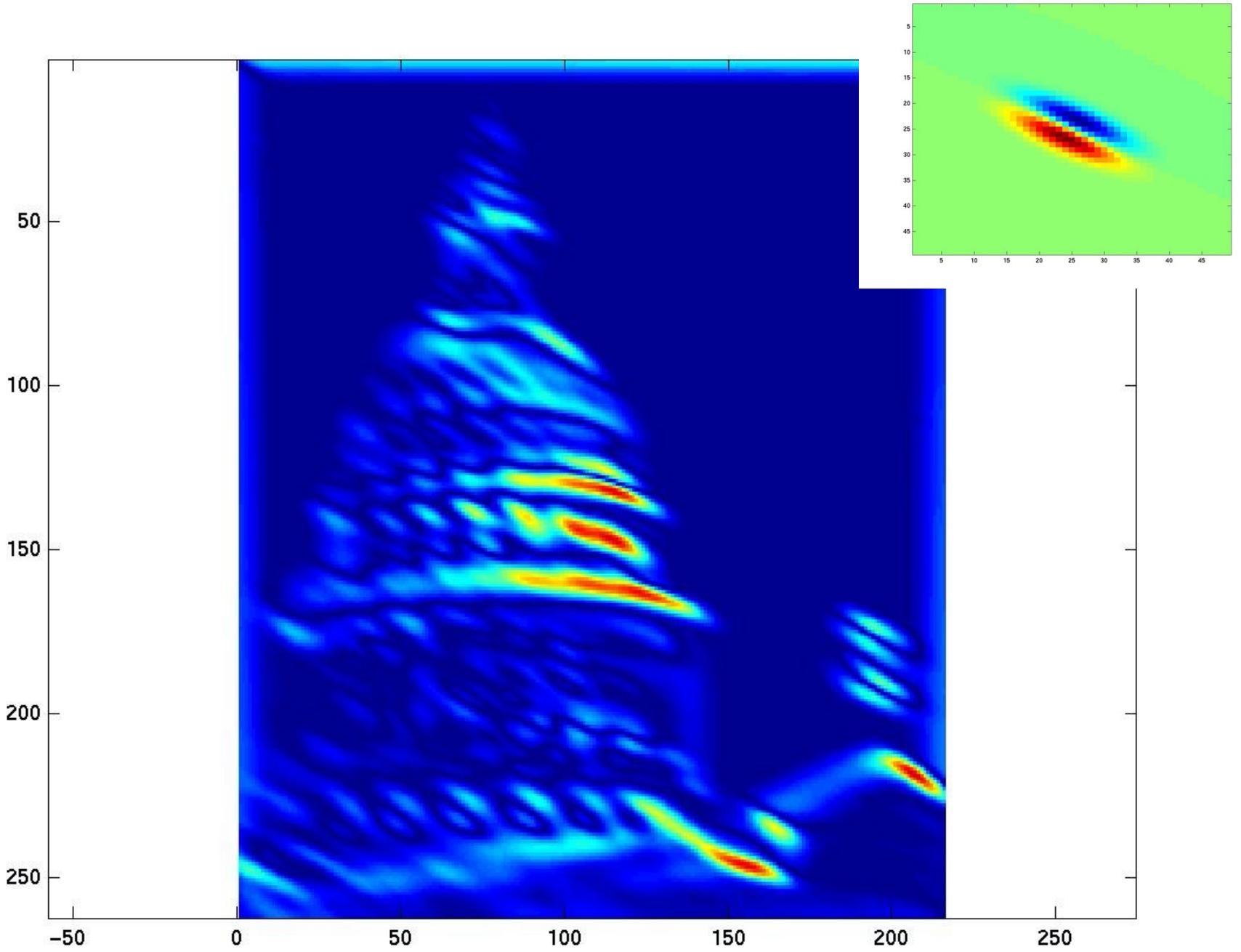

50
100
150
200
250
−50
0
50
100
150
200
250
5
10
15
20
25
30
35
40
45

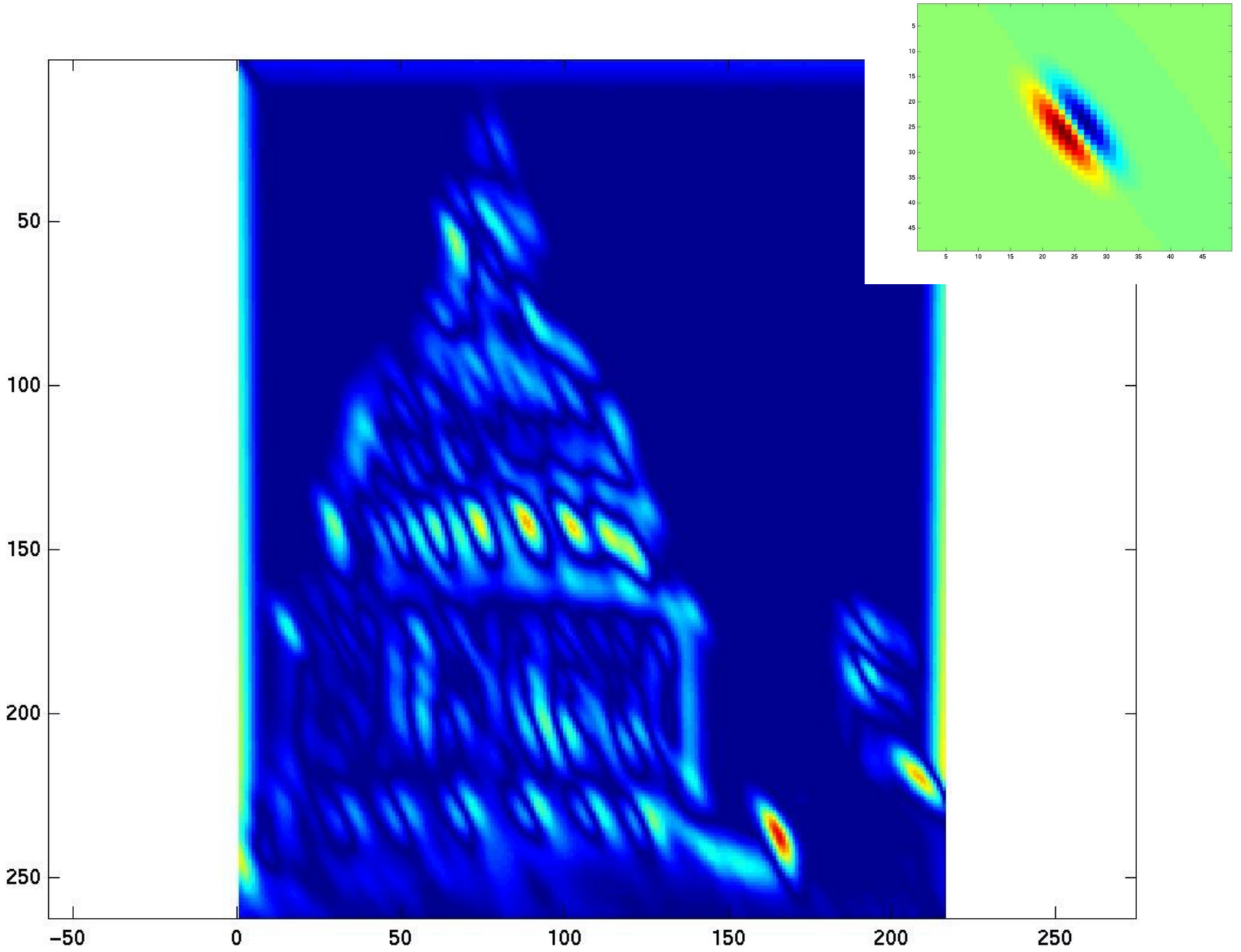

50
100
150
200
250
−50
0
50
100
150
200
250
5
10
15
20
25
30
35
40
45

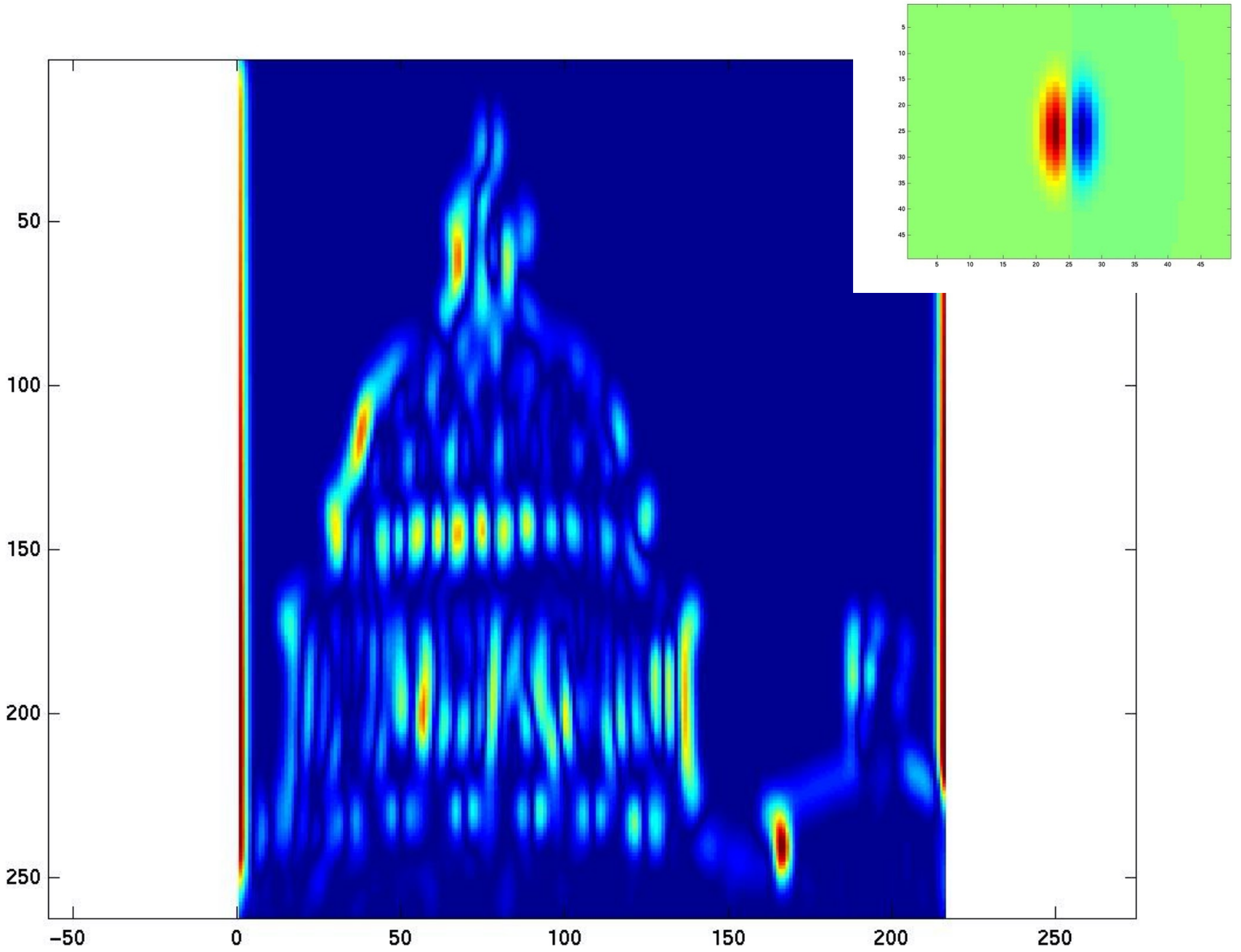

50
100
150
200
250
−50
0
50
100
150
200
250
5
10
15
20
25
30
35
40
45

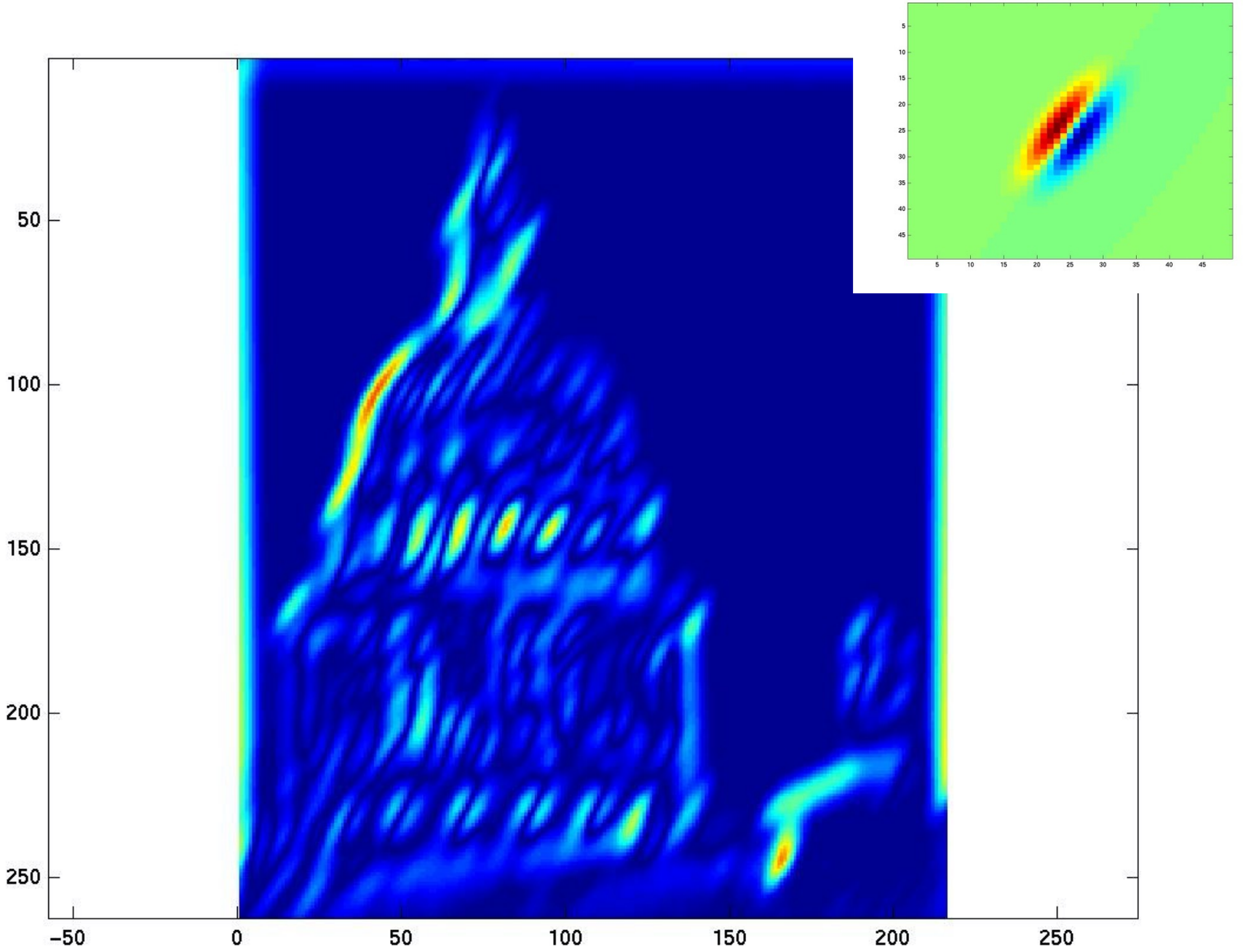

50
100
150
200
250
−50
0
50
100
150
200
250
5
10
15
20
25
30
35
40
45

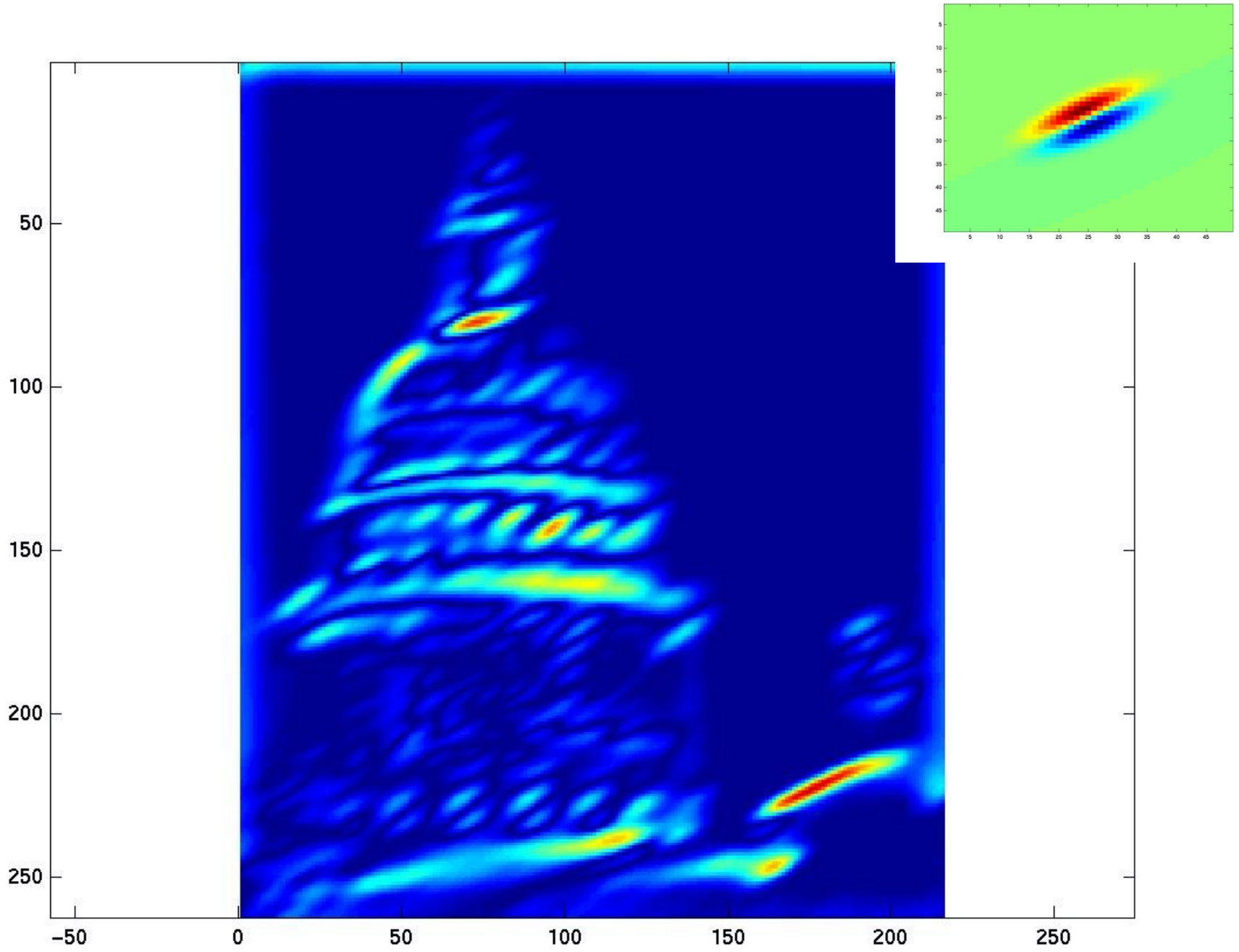

50
100
150
200
250
−50
0
50
100
150
200
250
5
10
15
20
25
30
35
40
45

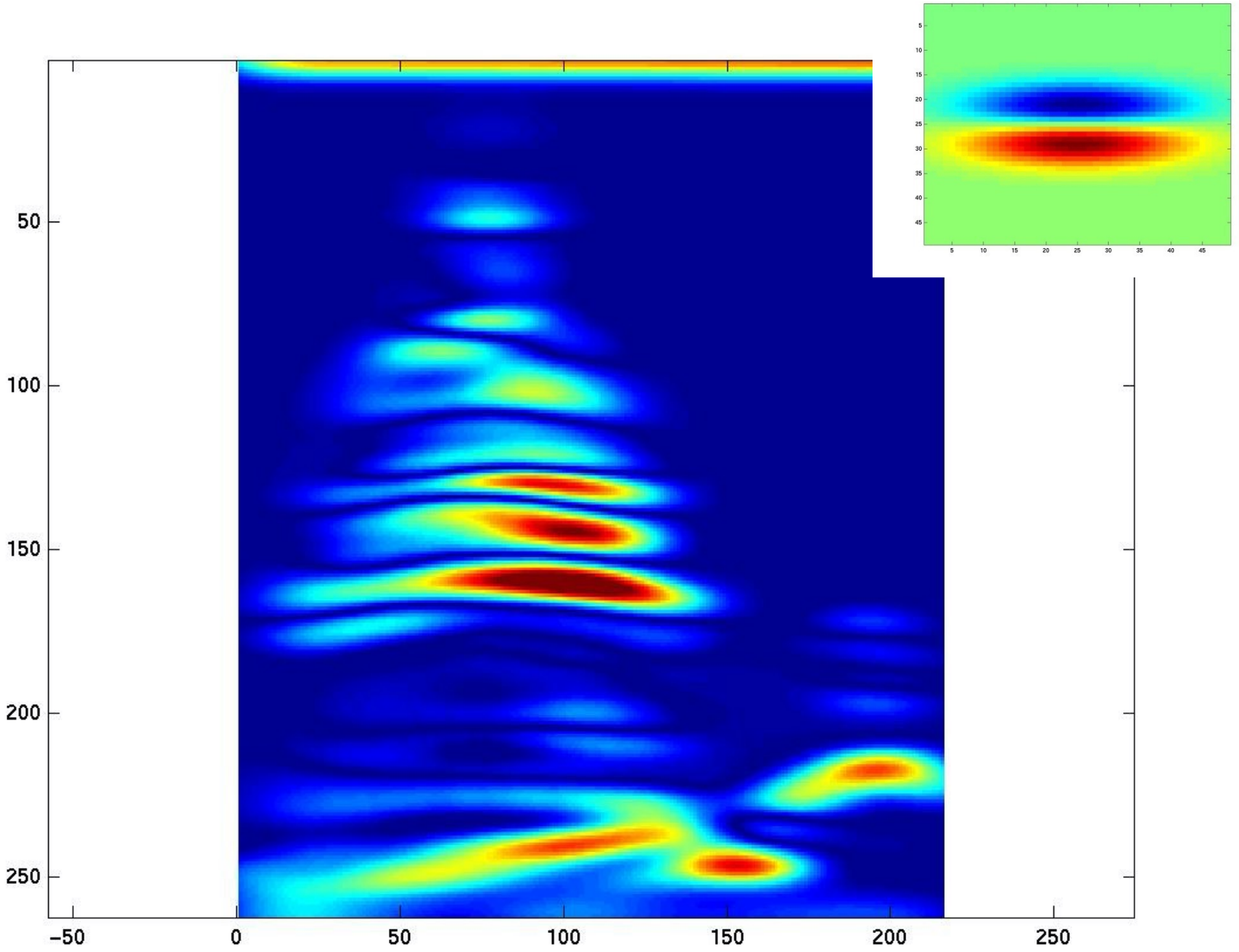

50
100
150
200
250
−50
0
50
100
150
200
250
5
10
15
20
25
30
35
40
45

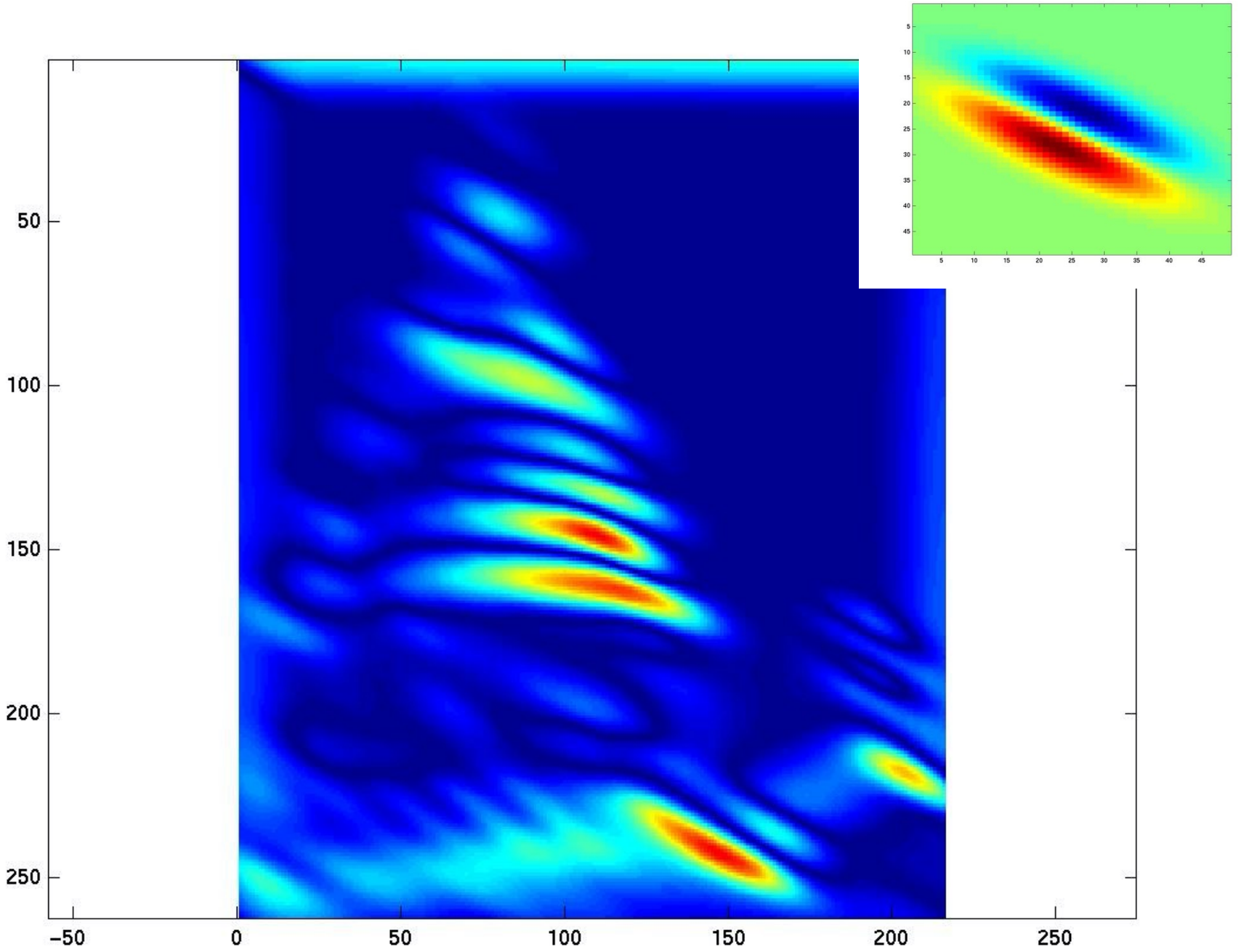

50
100
150
200
250
−50
0
50
100
150
200
250
5
10
15
20
25
30
35
40
45

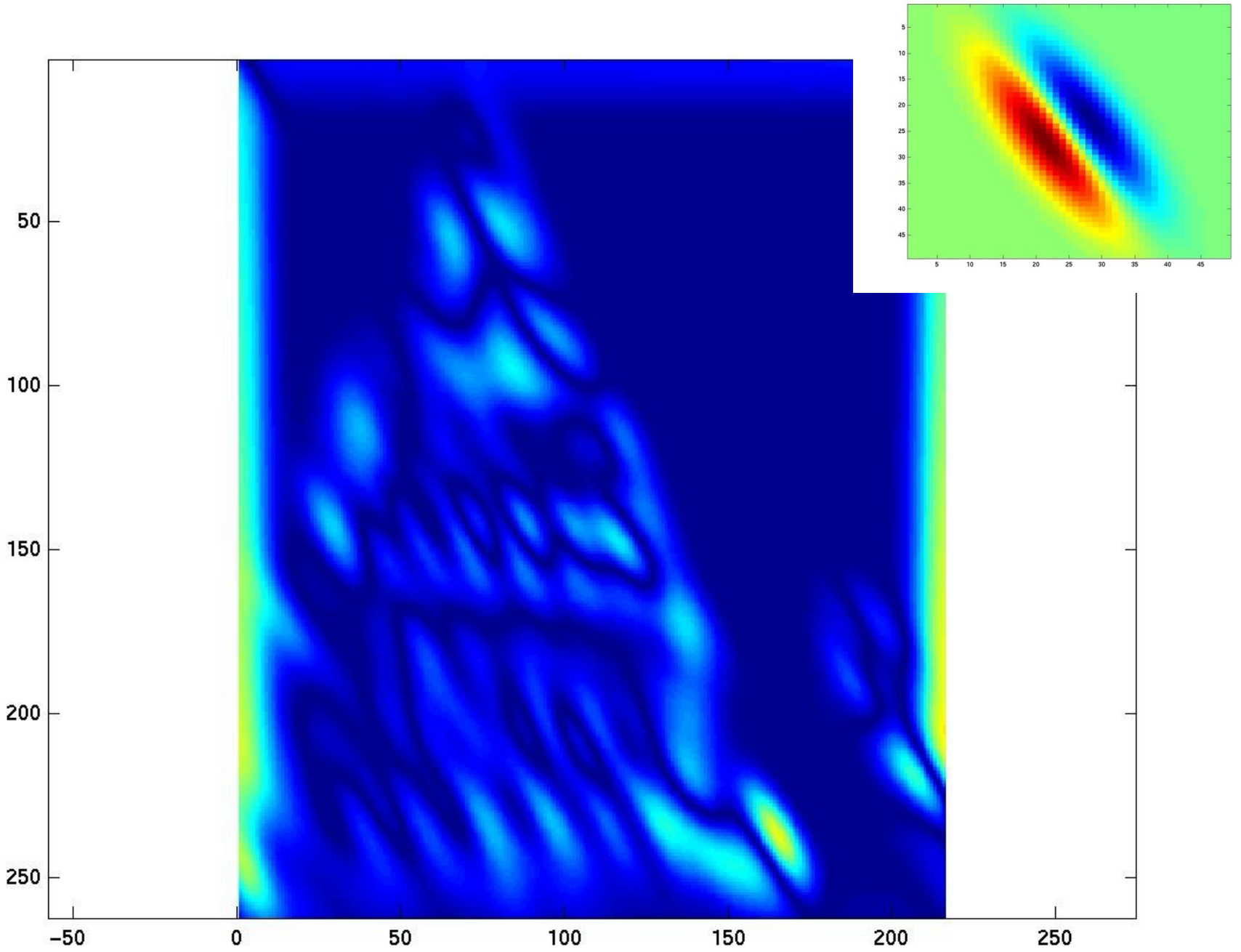

50
100
150
200
250
−50
0
50
100
150
200
250
5
10
15
20
25
30
35
40
45

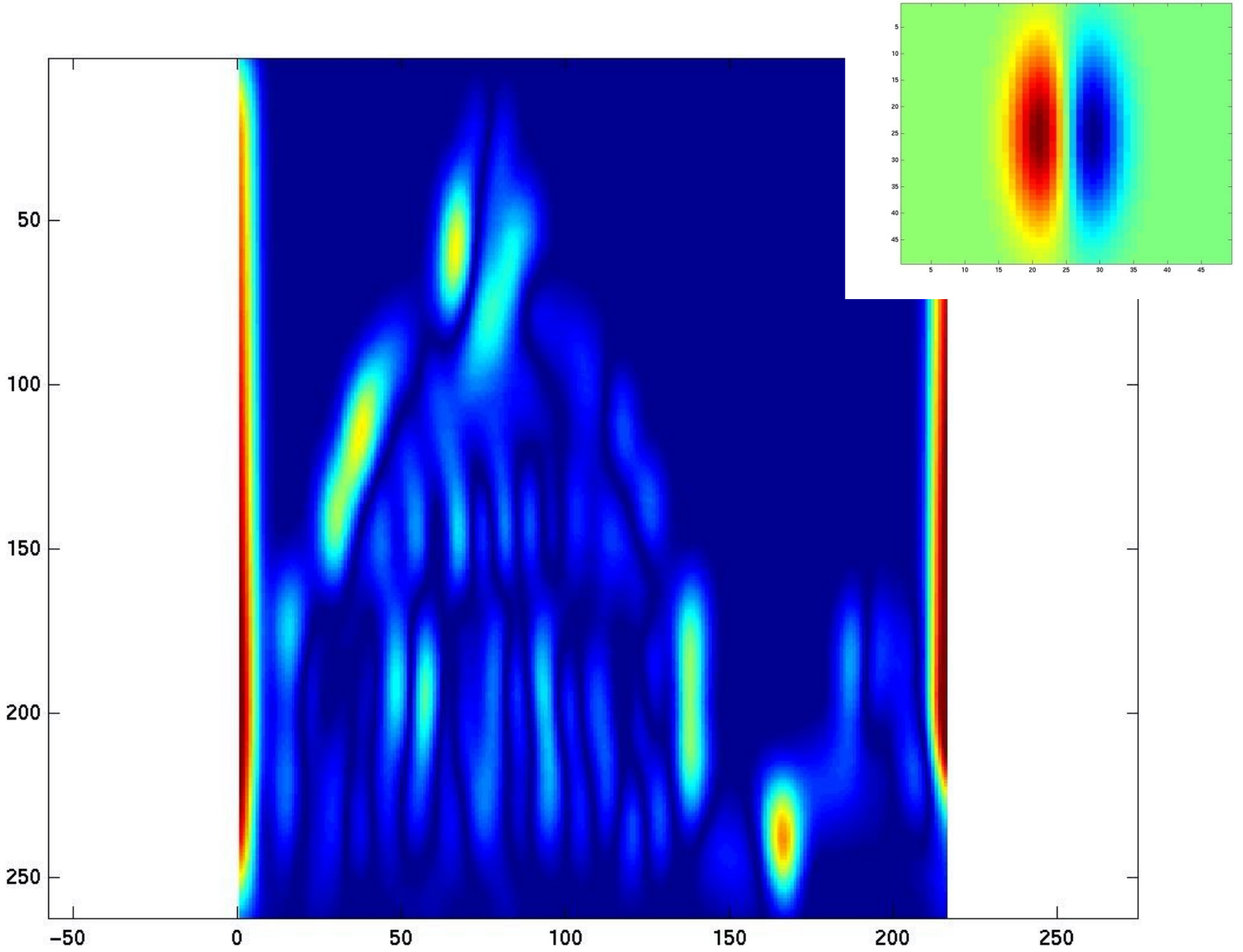

50
100
150
200
250
−50
0
50
100
150
200
250
5
10
15
20
25
30
35
40
45

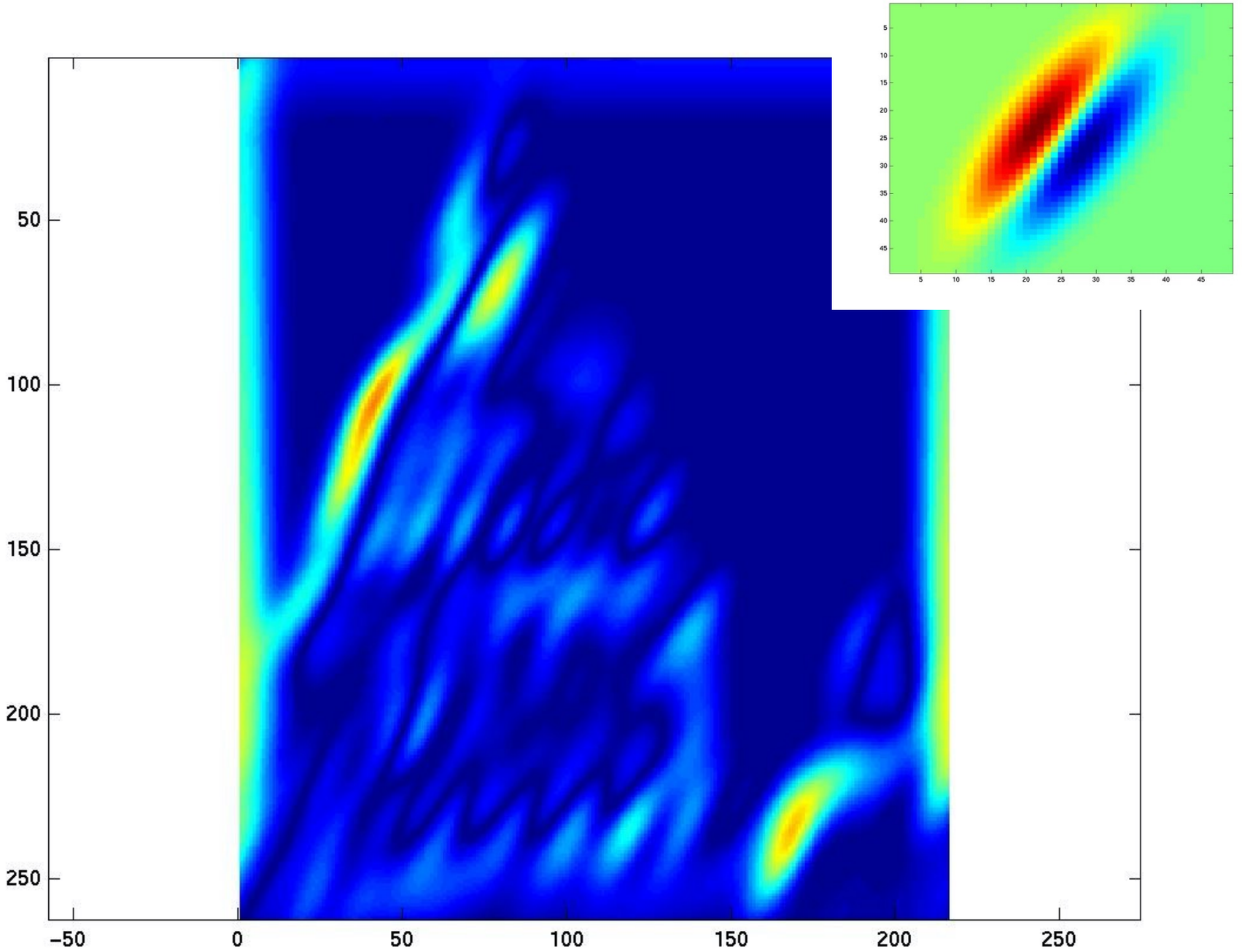

50
100
150
200
250
−50
0
50
100
150
200
250

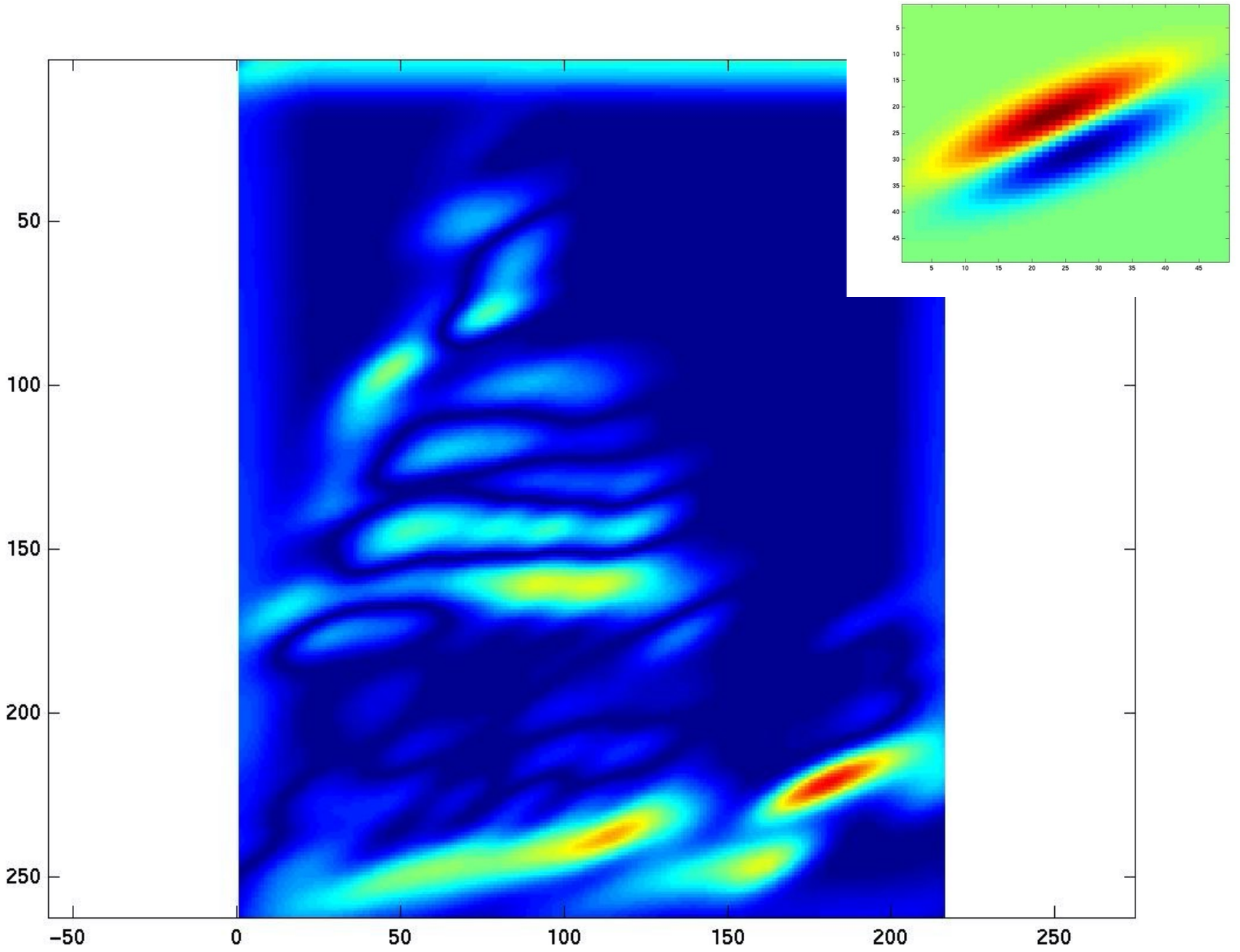

50
100
150
200
250
−50
0
50
100
150
200
250
5
10
15
20
25
30
35
40
45

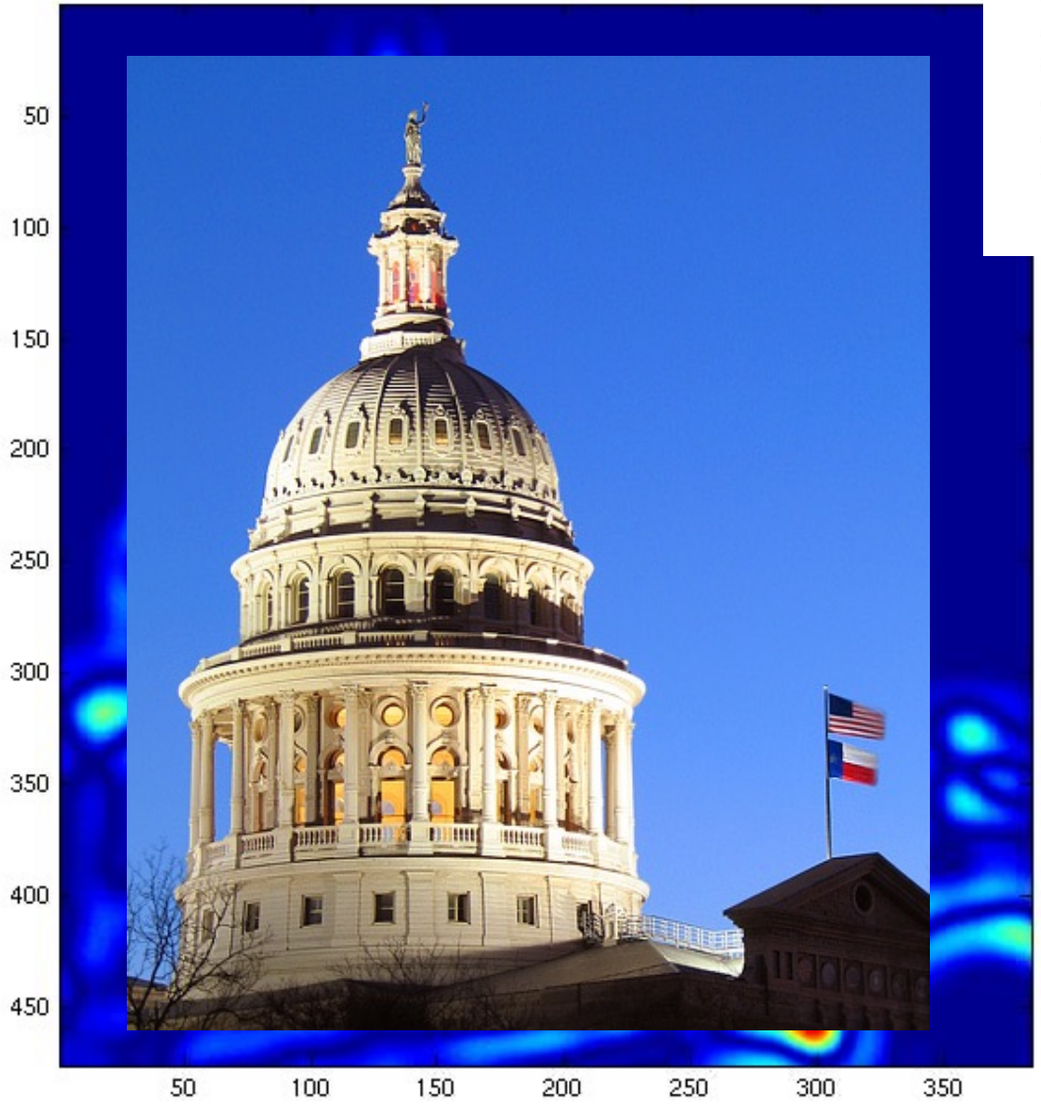

50
100
150
200
250
−50
0
50
100
150
200
250
5
10
15
20
25
30
35
40
45

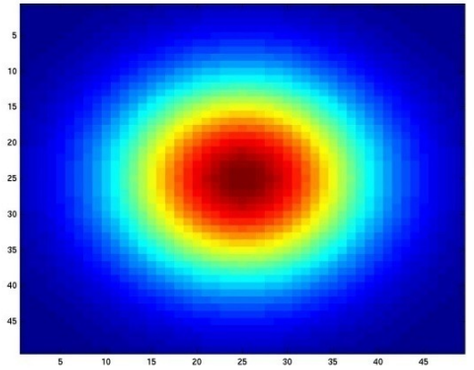

50
100
150
200
250
−50
0
50
100
150
200
250
5
10
15
20
25
30
35
40
45


50
100
150
200
250
−50
0
50
100
150
200
250
5
10
15
20
25
30
35
40
45

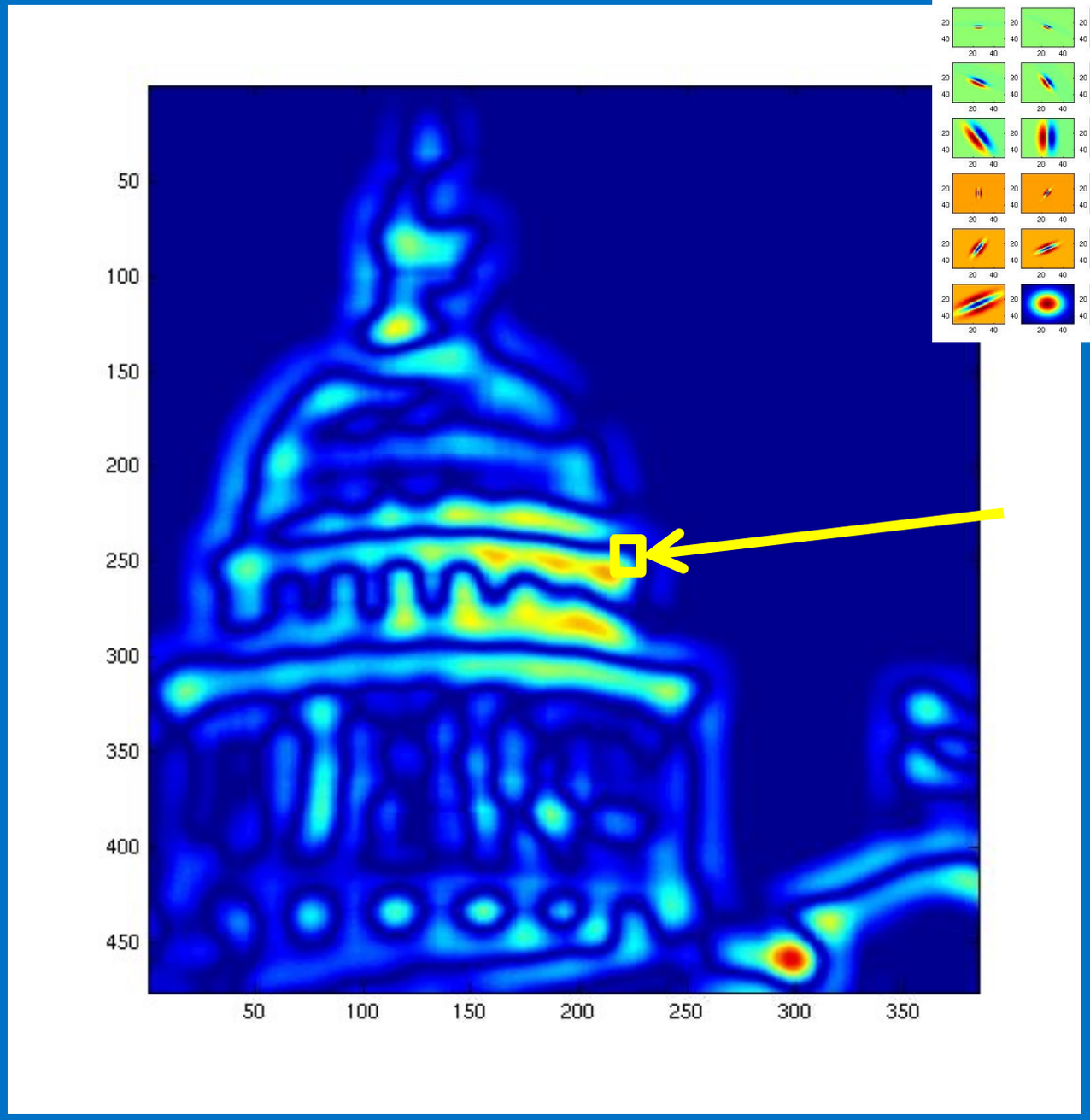

50
100
150
200
250
−50
0
50
100
150
200
250
5
10
15
20
25
30
35
40
45


50
100
150
200
250
−50
0
50
100
150
200
250
5
10
15
20
25
30
35
40
45





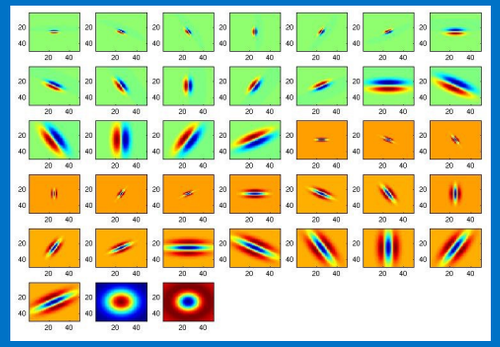
Kristen Grauman

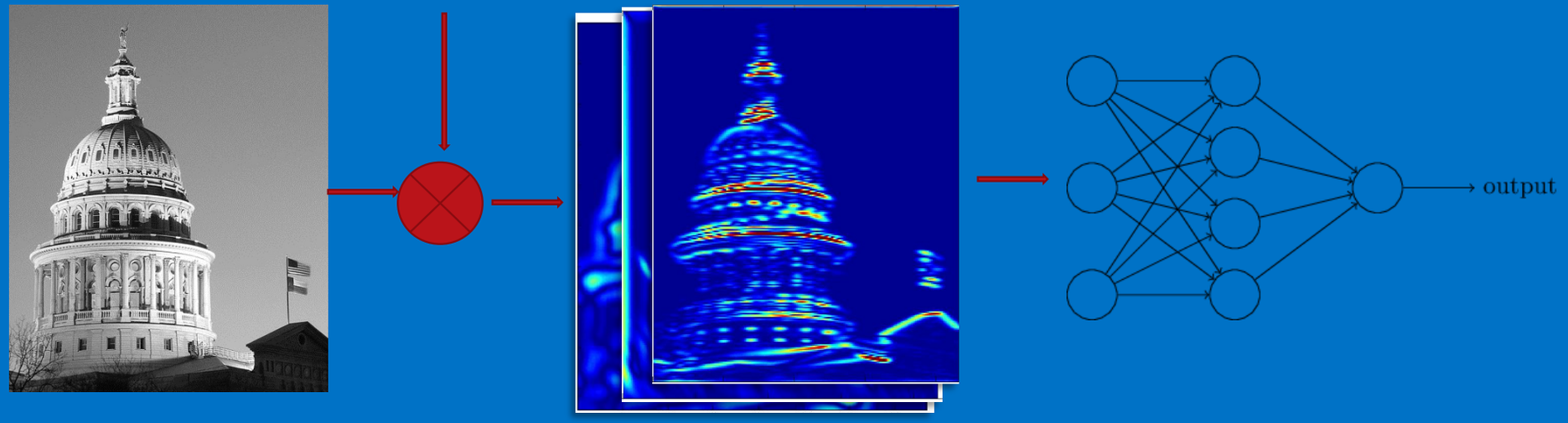


[r1, r2, …, r38]

We can form a feature vector from the list of responses at each pixel.

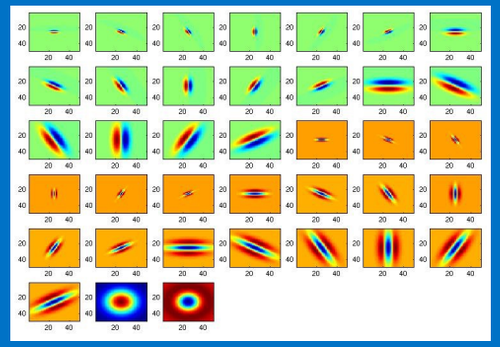
Feed to Neural Network



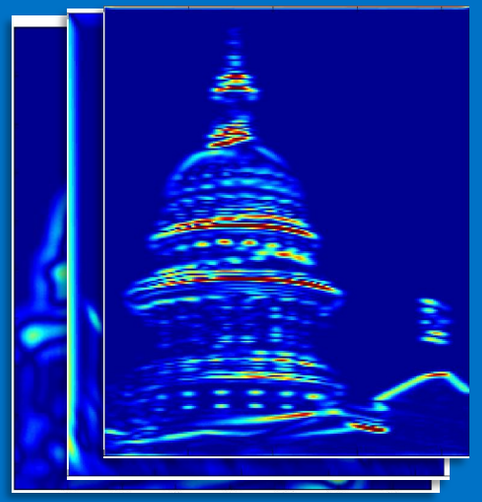


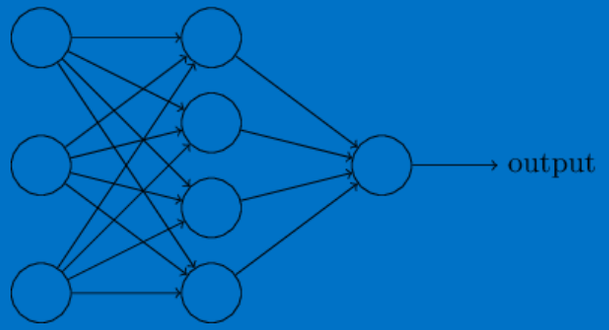
Feature volume consists of Channels, one channel for each filter

Feed to Neural Network



You can Flatten this volume to a vector

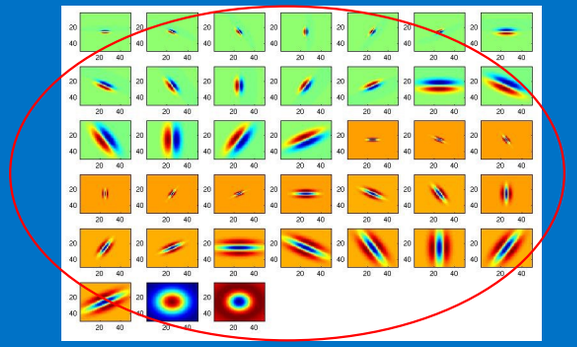




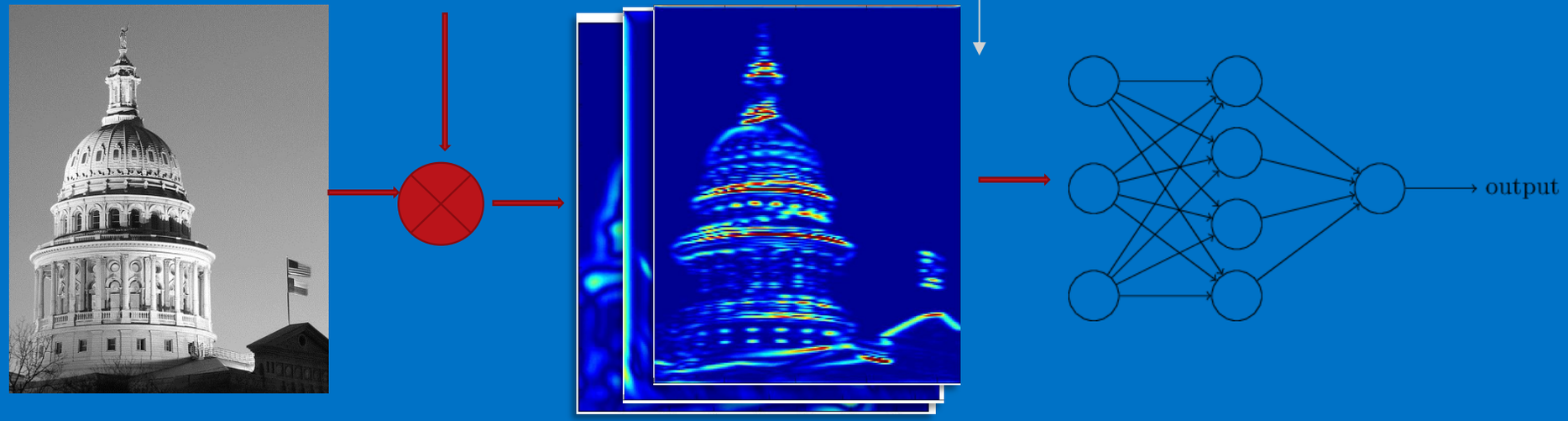
Train Neural Network

Convolutional Neural Network

Learn from Data what these filters should be



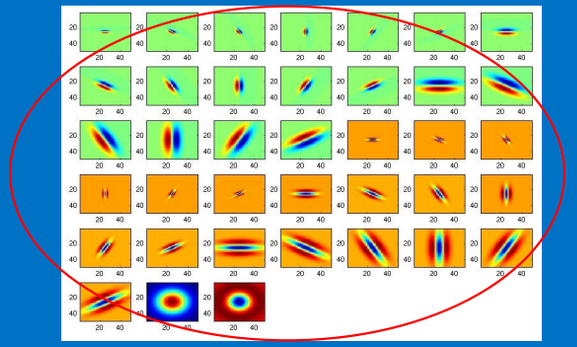
You can Flatten this volume to a vector



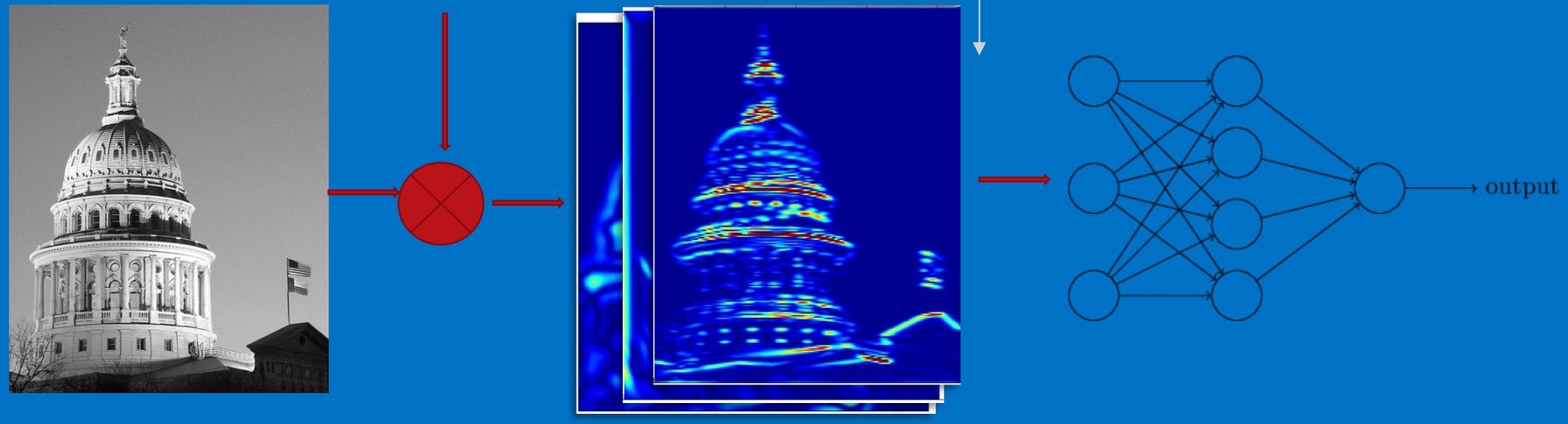
Train Neural Network

Convolutional Neural Network

Learn from Data what these filters should be



You can Flatten this volume to a vector



Move the gradient from Fully Connected Layer to CNN layer

Train Neural Network

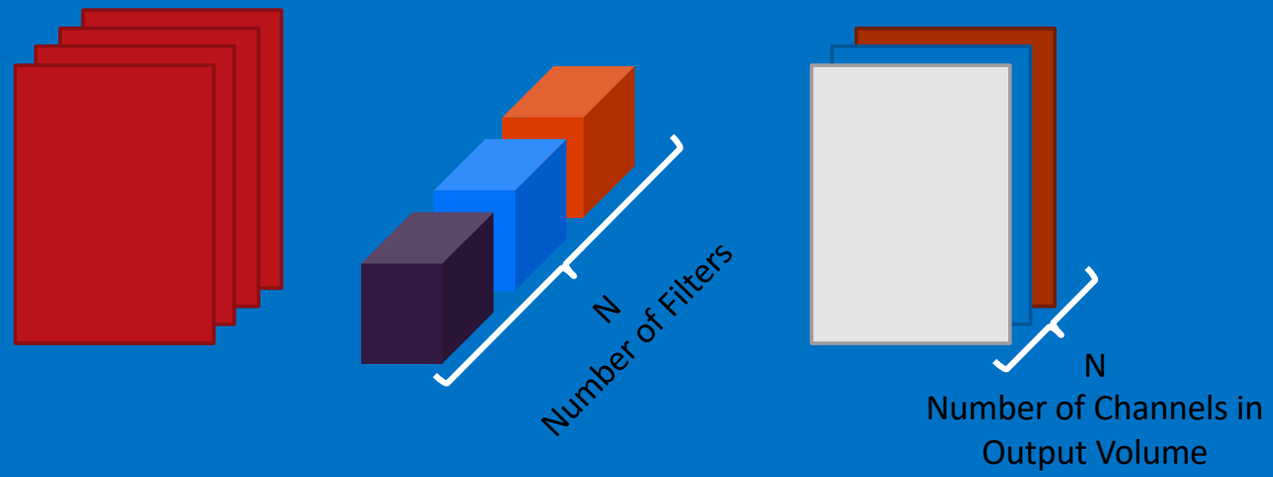
Convolutional Layer

Input Volume

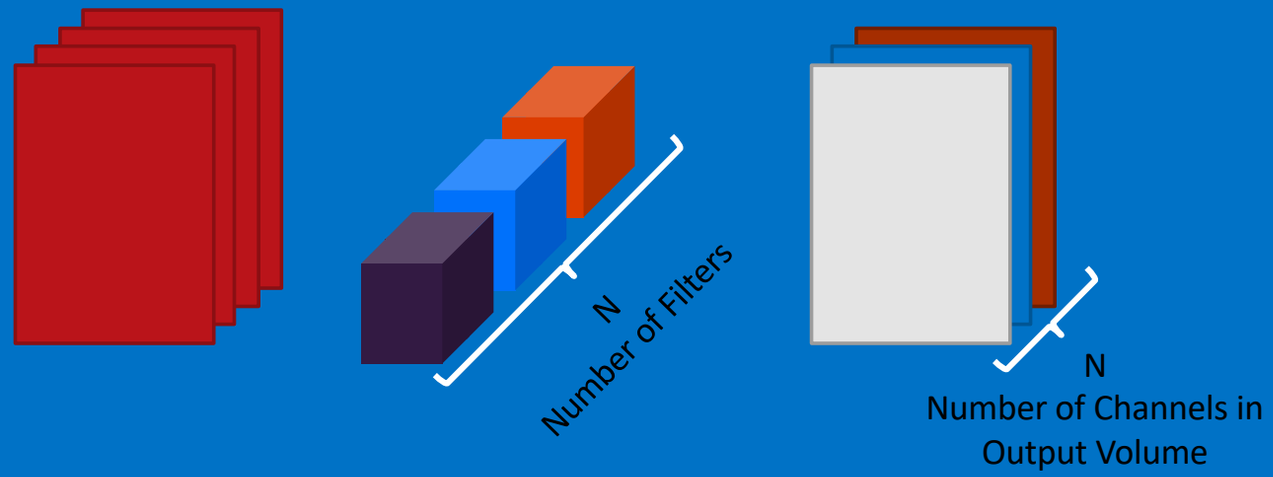
Convolutional Filters

Output Volume

Convolutional Layer

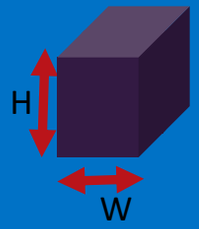


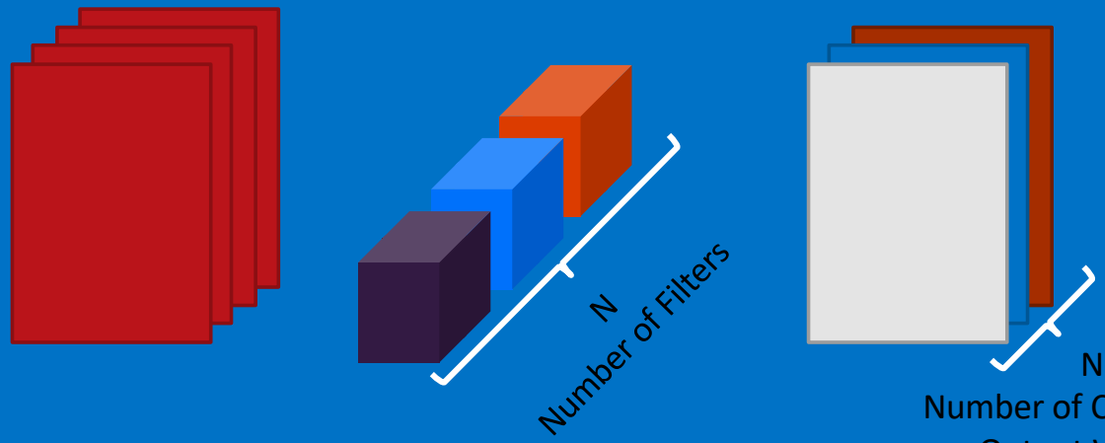
Convolutional Layer

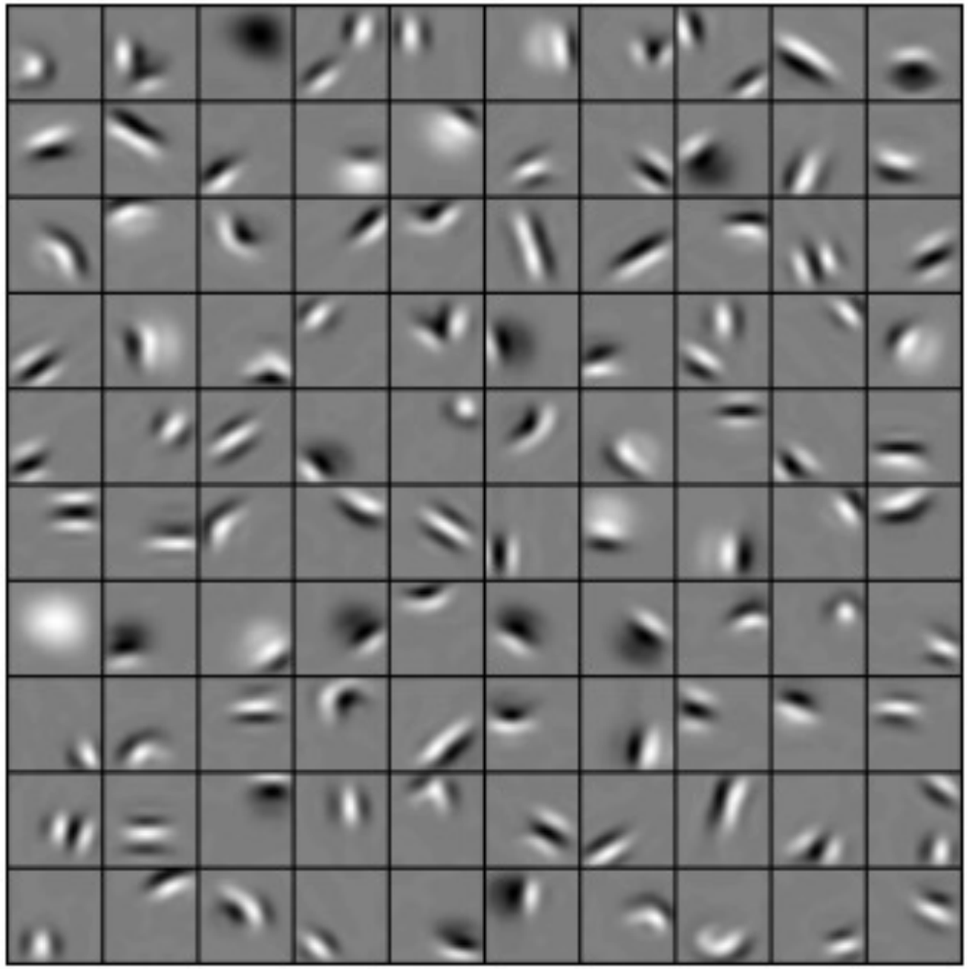


Convolutional Layer

Each filter.
We define height and width.
Mostly they are same.

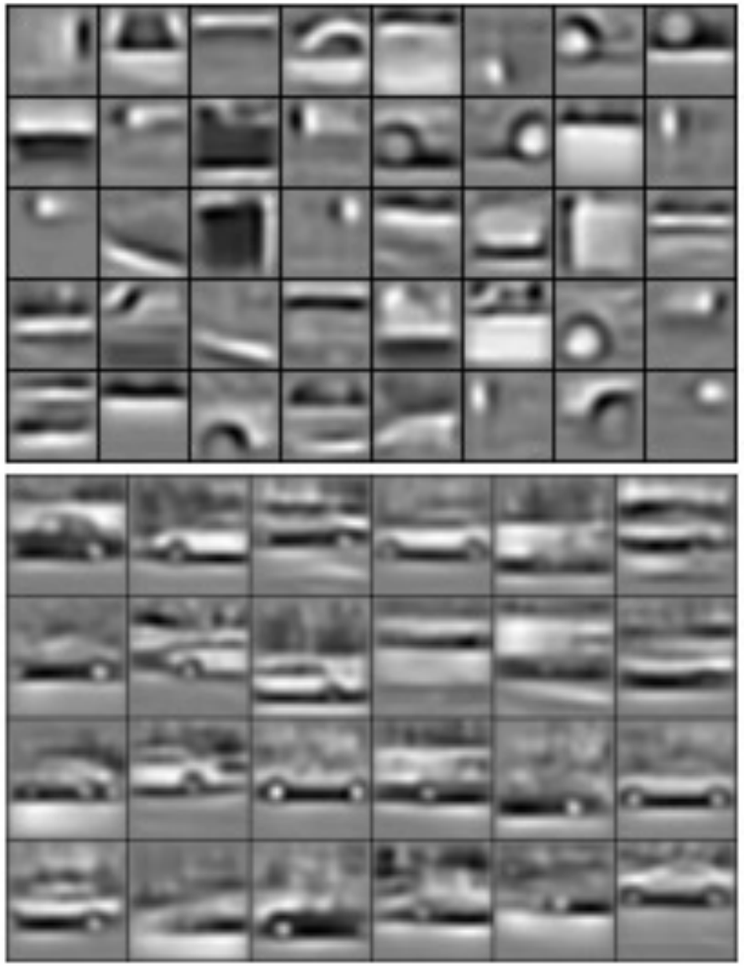






faces

cars

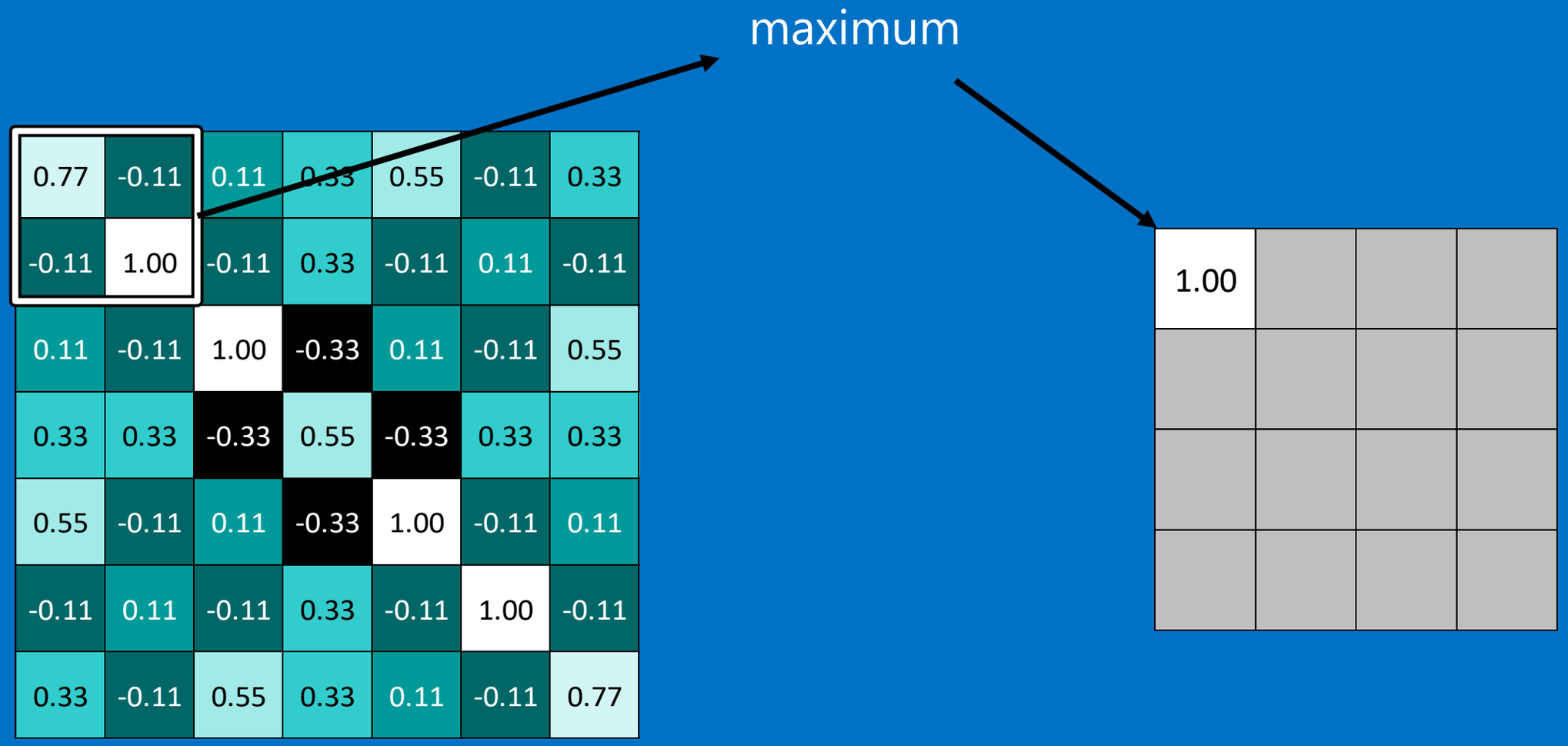


Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations
Honglak Lee, Roger Grosse, Rajesh Ranganath, Andrew Y. Ng

Pooling: Shrinking the image stack

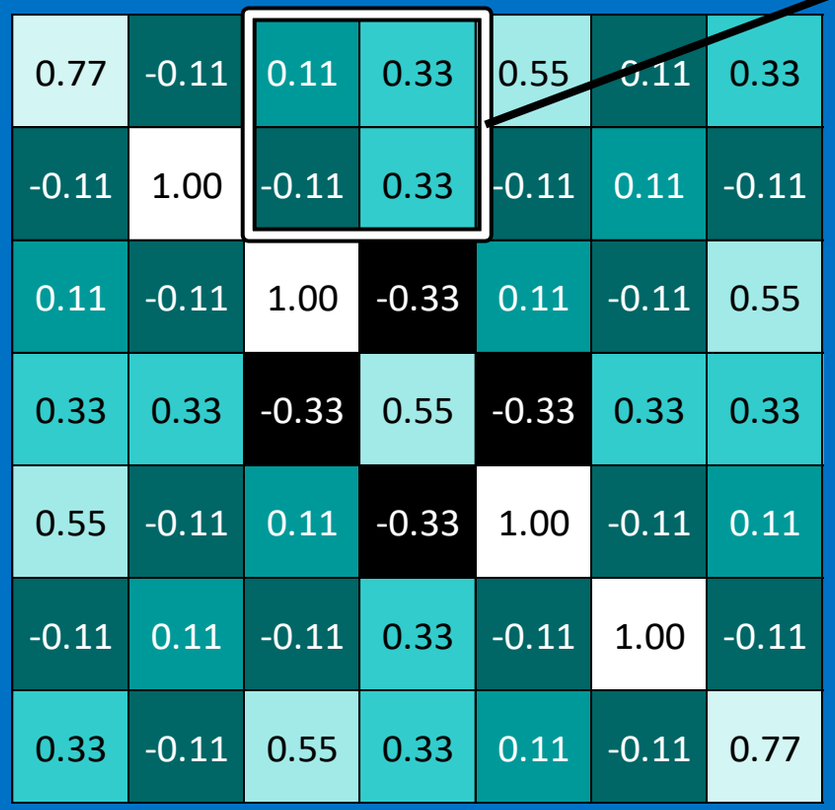
1. Pick a window size (usually 2 or 3).
2. Pick a stride (usually 2).
3. Walk your window across your filtered images.
4. From each window, take the maximum value.

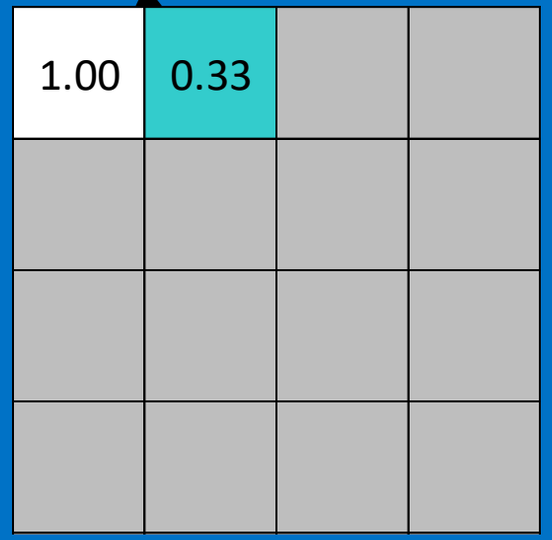
Pooling


maximum
0.77 -0.11 0.11 0.33 0.55 -0.11 0.33
-0.11 1.00 -0.11 0.33 -0.11 0.11 -0.11
0.11 -0.11 1.00 -0.33 0.11 -0.11 0.55
0.33 0.33 -0.33 0.55 -0.33 0.33 0.33
0.55 -0.11 0.11 -0.33 1.00 -0.11 0.11
-0.11 0.11 -0.11 0.33 -0.11 1.00 -0.11
0.33 -0.11 0.55 0.33 0.11 -0.11 0.77
1.00

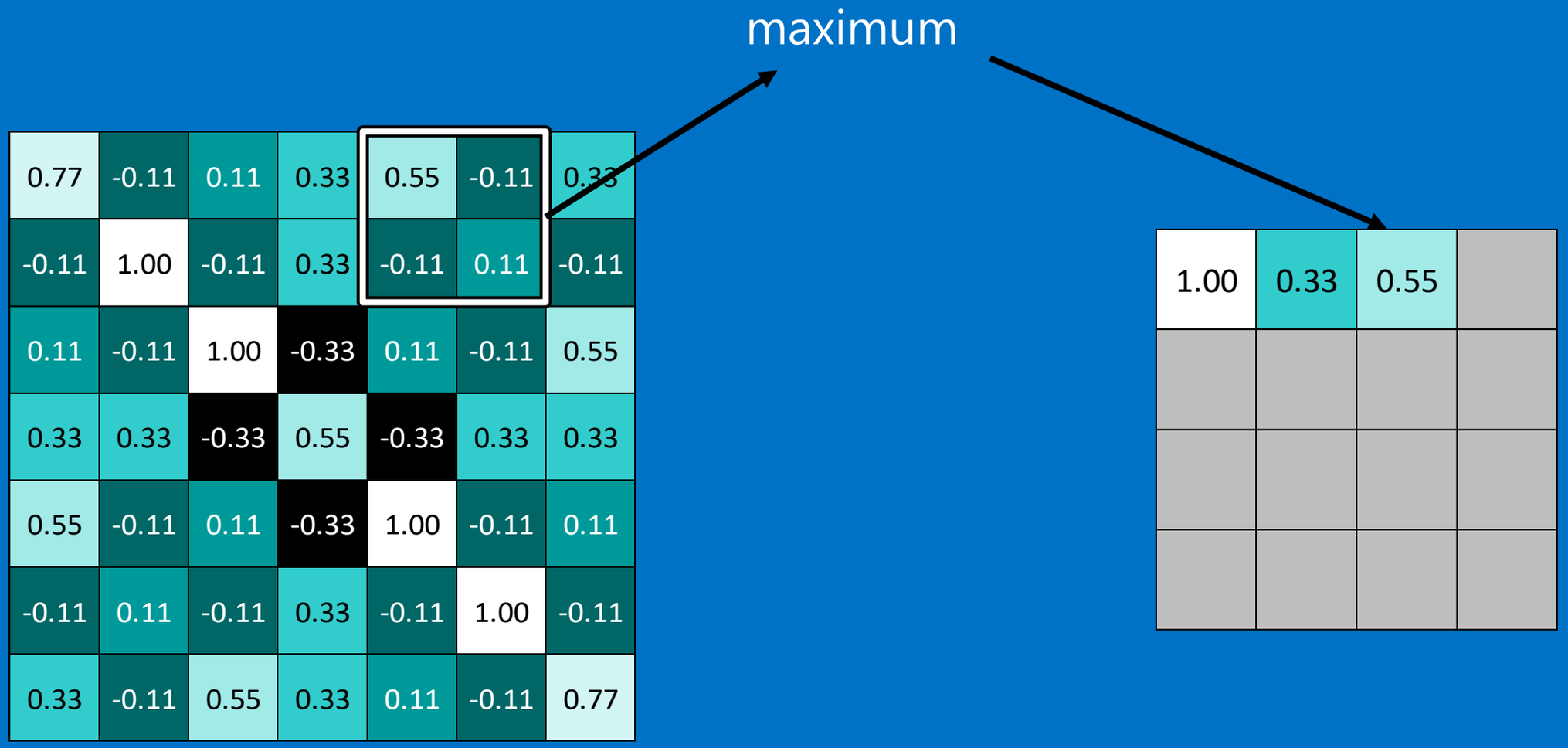
Pooling

maximum

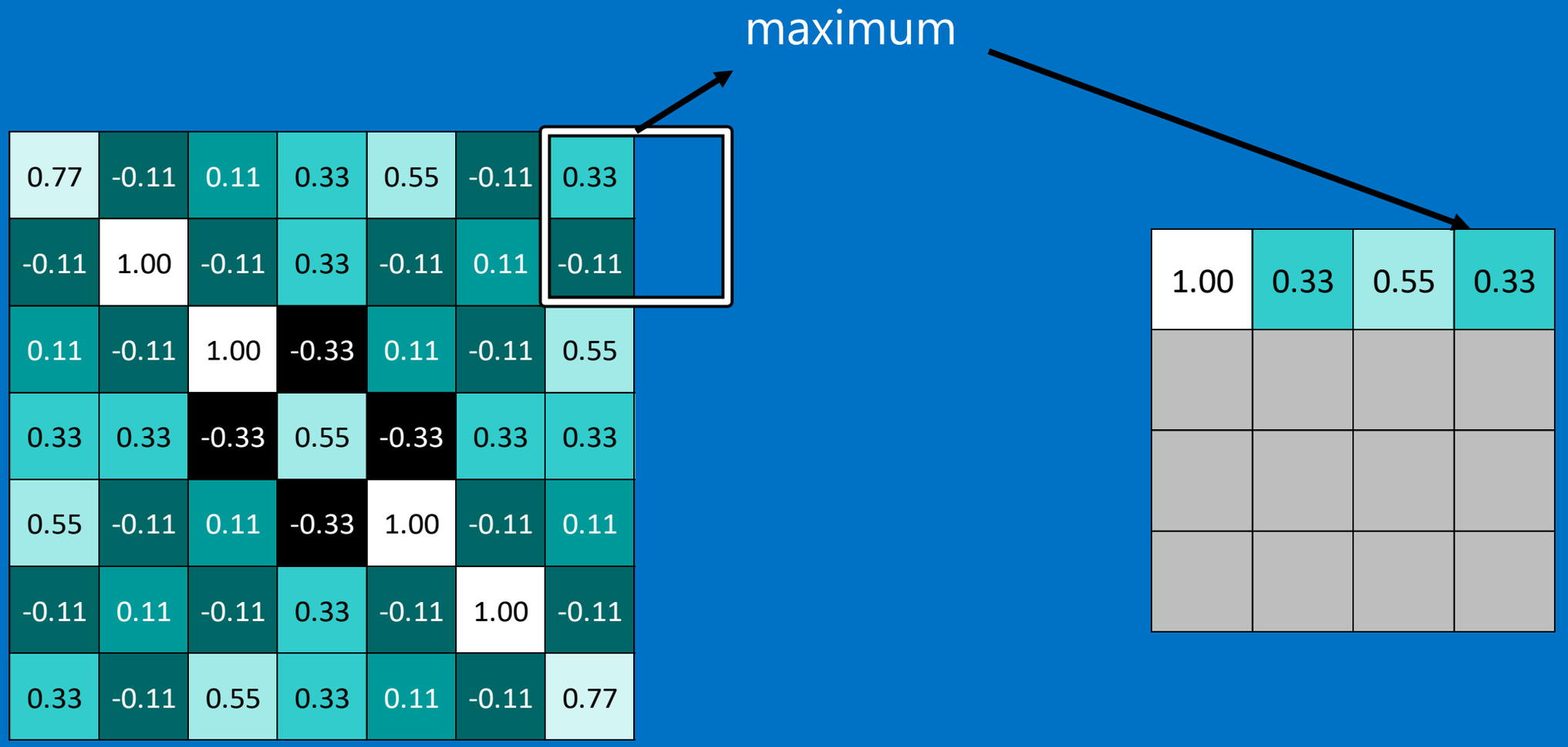




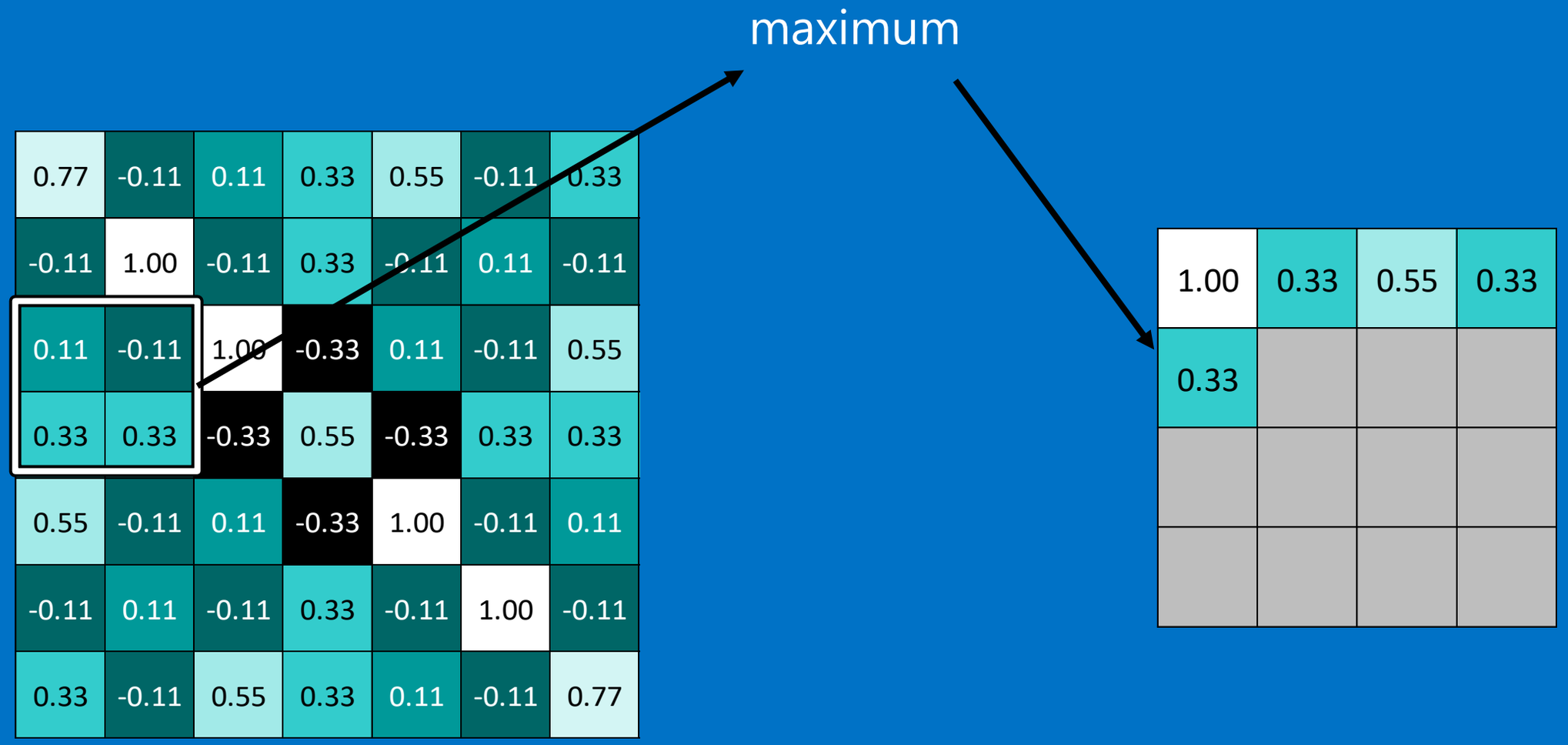
Pooling


maximum
0.77 -0.11 0.11 0.33 0.55 -0.11 0.33
-0.11 1.00 -0.11 0.33 -0.11 0.11 -0.11
0.11 -0.11 1.00 -0.33 0.11 -0.11 0.55
0.33 0.33 -0.33 0.55 -0.33 0.33 0.33
0.55 -0.11 0.11 -0.33 1.00 -0.11 0.11
-0.11 0.11 -0.11 0.33 -0.11 1.00 -0.11
0.33 -0.11 0.55 0.33 0.11 -0.11 0.77
1.00 0.33 0.55

Pooling


maximum
0.77 -0.11 0.11 0.33 0.55 -0.11 0.33
-0.11 1.00 -0.11 0.33 -0.11 0.11 -0.11
0.11 -0.11 1.00 -0.33 0.11 -0.11 0.55
0.33 0.33 -0.33 0.55 -0.33 0.33 0.33
0.55 -0.11 0.11 -0.33 1.00 -0.11 0.11
-0.11 0.11 -0.11 0.33 -0.11 1.00 -0.11
0.33 -0.11 0.55 0.33 0.11 -0.11 0.77
1.00 0.33 0.55 0.33

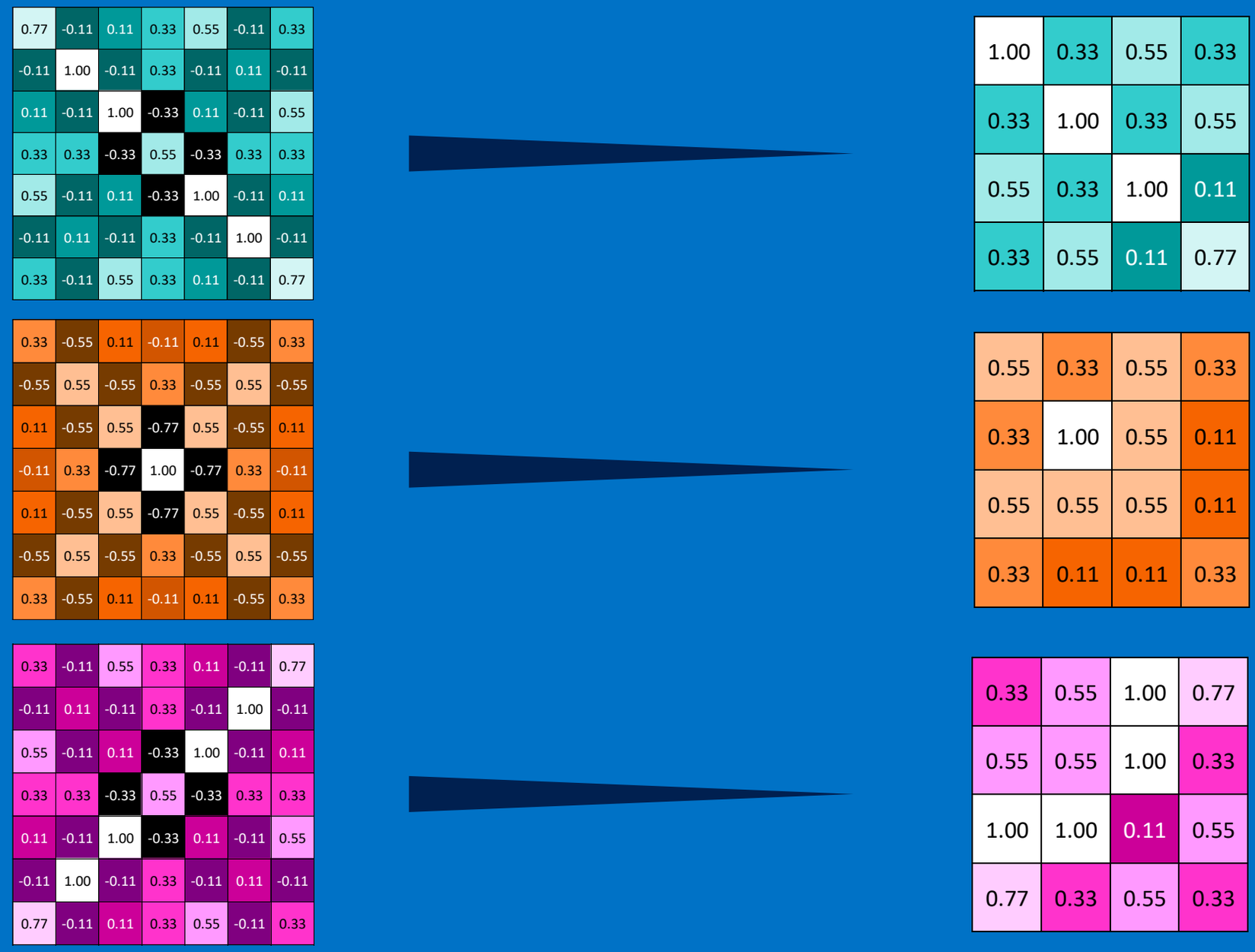
Pooling


maximum
0.77 -0.11 0.11 0.33 0.55 -0.11 0.33
-0.11 1.00 -0.11 0.33 -0.11 0.11 -0.11
0.11 -0.11 1.00 -0.33 0.11 -0.11 0.55
0.33 0.33 -0.33 0.55 -0.33 0.33 0.33
0.55 -0.11 0.11 -0.33 1.00 -0.11 0.11
-0.11 0.11 -0.11 0.33 -0.11 1.00 -0.11
0.33 -0.11 0.55 0.33 0.11 -0.11 0.77
1.00 0.33 0.55 0.33
0.33

Pooling

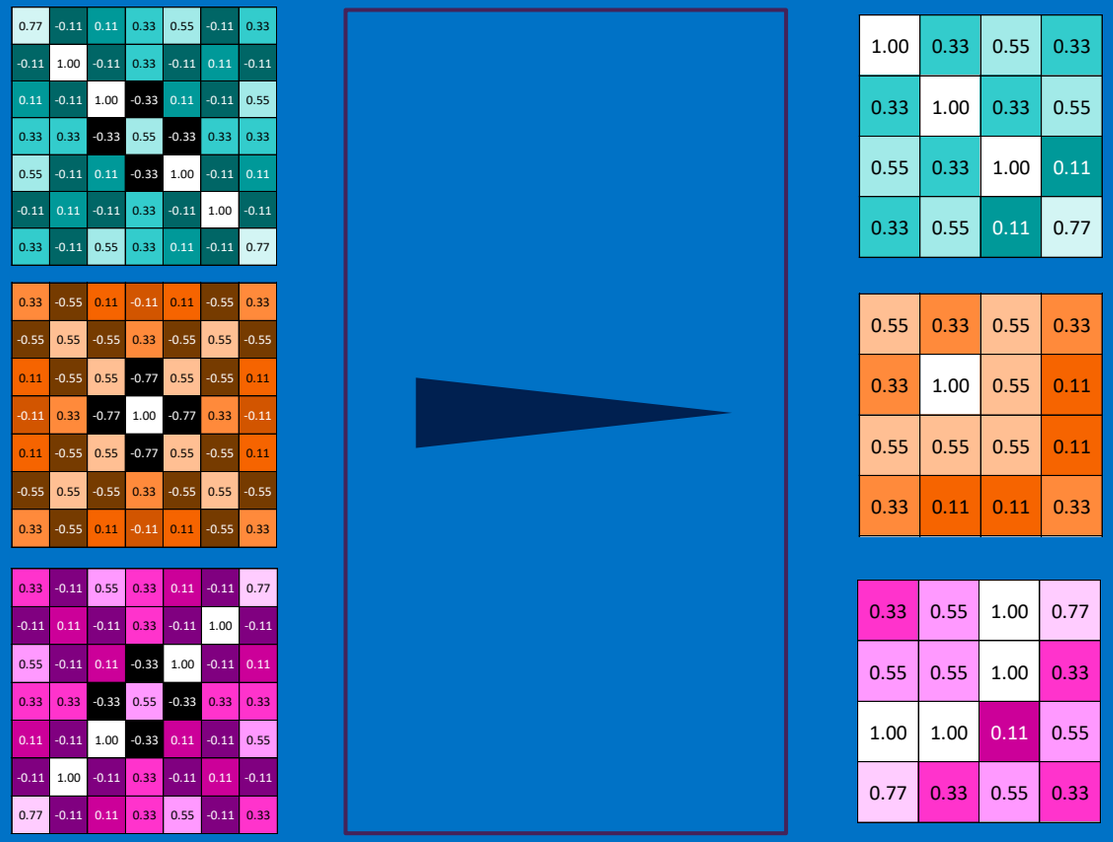
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77


0.77 -0.11 0.11 0.33 0.55 -0.11 0.33
-0.11 1.00 -0.11 0.33 -0.11 0.11 -0.11
0.11 -0.11 1.00 -0.33 0.11 -0.11 0.55
0.33 0.33 -0.33 0.55 -0.33 0.33 0.33
0.55 -0.11 0.11 -0.33 1.00 -0.11 0.11
-0.11 0.11 -0.11 0.33 -0.11 1.00 -0.11
0.33 -0.11 0.55 0.33 0.11 -0.11 0.77
1.00 0.33 0.55 0.33
0.33 1.00 0.33 0.55
0.55 0.33 1.00 0.11
0.33 0.55 0.11 0.77
0.33 -0.55 0.11 -0.11 0.11 -0.55 0.33
-0.55 0.55 -0.55 0.33 -0.55 0.55 -0.55
0.11 -0.55 0.55 -0.77 0.55 -0.55 0.11
-0.11 0.33 -0.77 1.00 -0.77 0.33 -0.11
0.11 -0.55 0.55 -0.77 0.55 -0.55 0.11
-0.55 0.55 -0.55 0.33 -0.55 0.55 -0.55
0.33 -0.55 0.11 -0.11 0.11 -0.55 0.33
0.55 0.33 0.55 0.33
0.33 1.00 0.55 0.11
0.55 0.55 0.55 0.11
0.33 0.11 0.11 0.33
0.33 -0.11 0.55 0.33 0.11 -0.11 0.77
-0.11 0.11 -0.11 0.33 -0.11 1.00 -0.11
0.55 -0.11 0.11 -0.33 1.00 -0.11 0.11
0.33 0.33 -0.33 0.55 -0.33 0.33 0.33
0.11 -0.11 1.00 -0.33 0.11 -0.11 0.55
-0.11 1.00 -0.11 0.33 -0.11 0.11 -0.11
0.77 -0.11 0.11 0.33 0.55 -0.11 0.33
0.33 0.55 1.00 0.77
0.55 0.55 1.00 0.33
1.00 1.00 0.11 0.55
0.77 0.33 0.55 0.33

Pooling layer

A stack of images becomes a stack of smaller images.

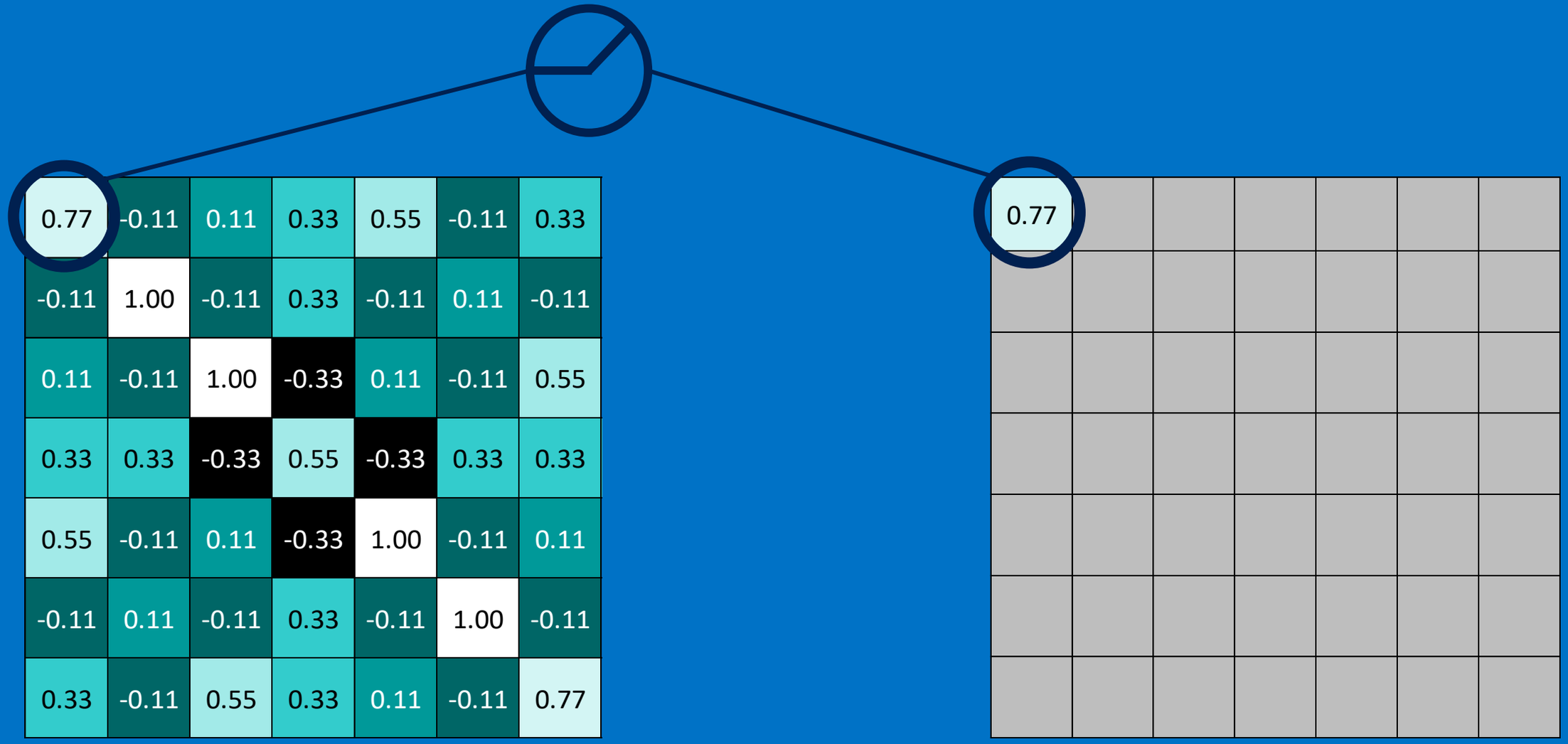


Normalization

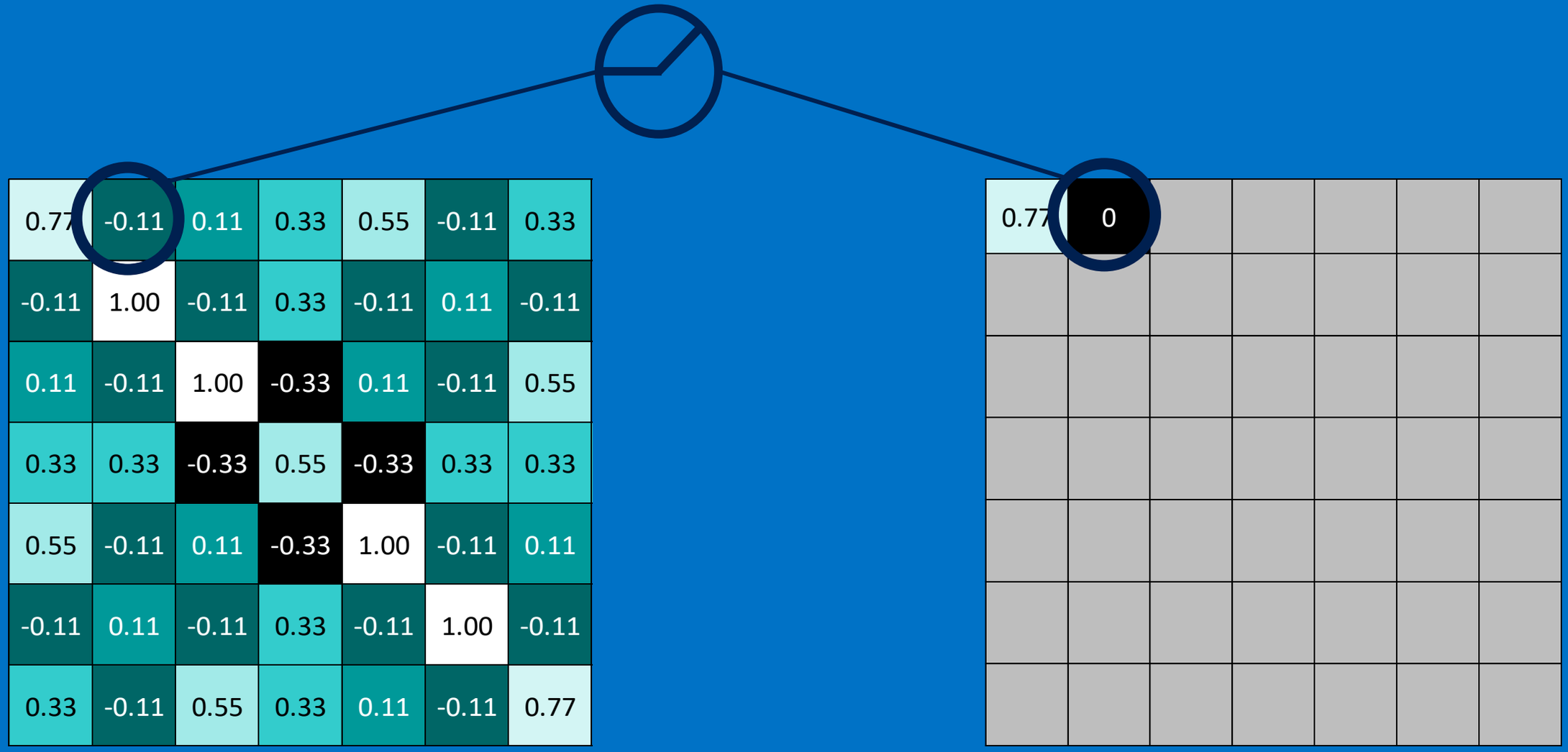
Keep the math from breaking by tweaking each of the values just a bit.

Change everything negative to zero.

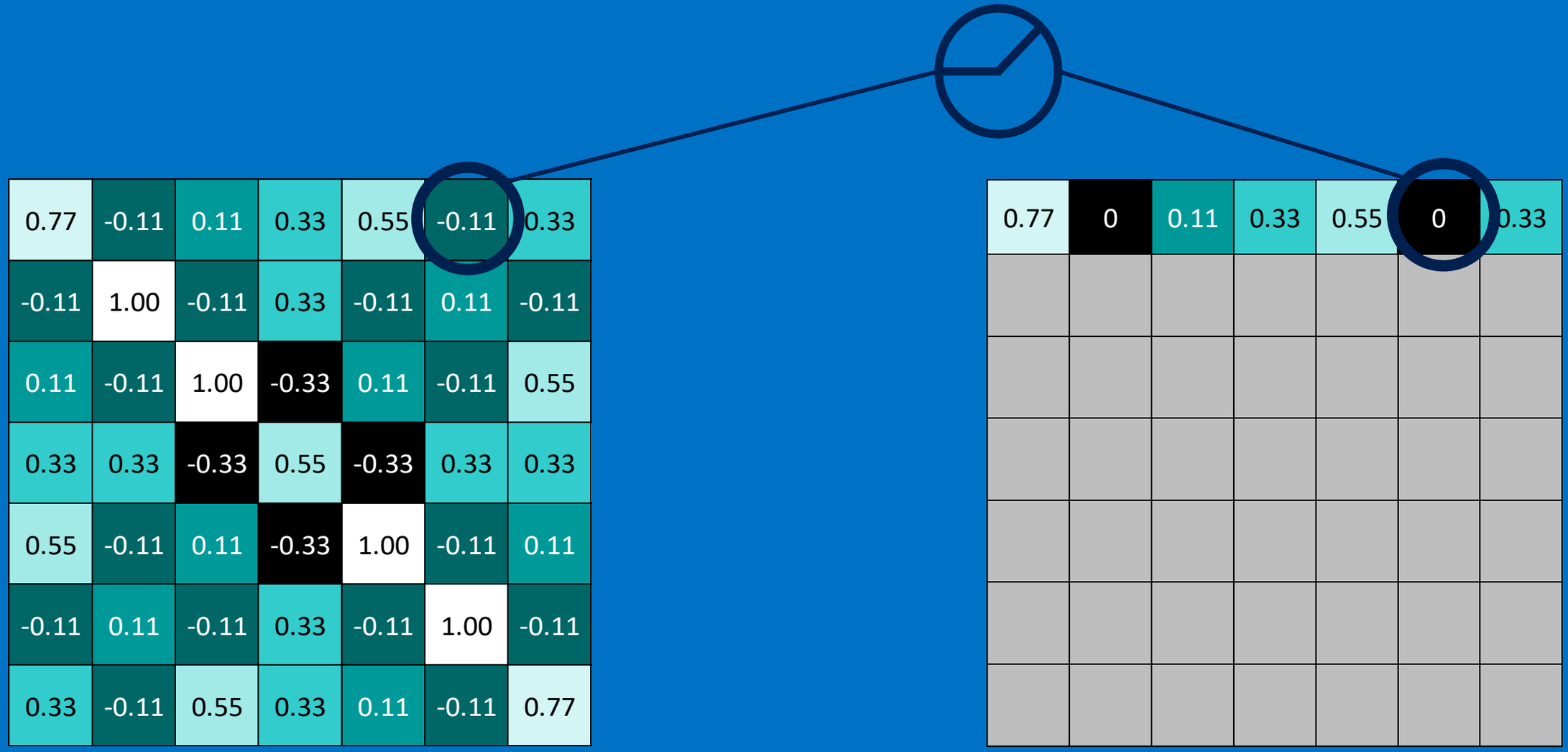
Rectified Linear Units (ReLUs)



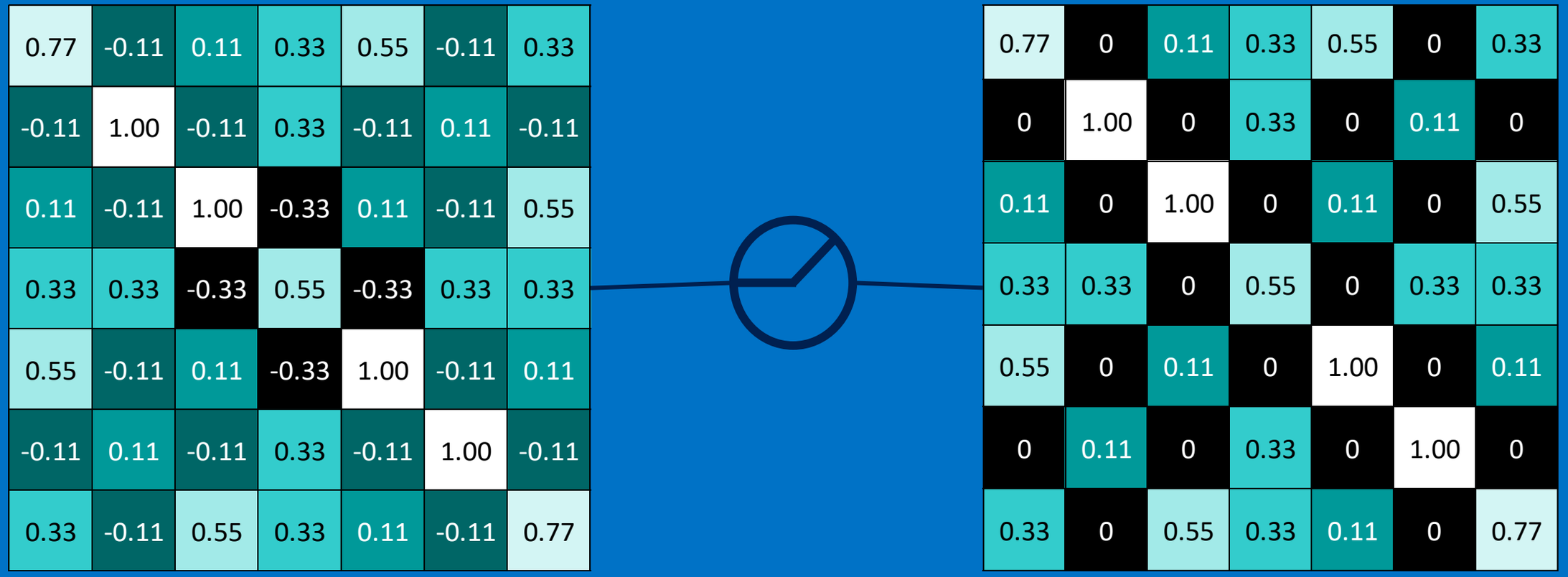
Rectified Linear Units (ReLUs)



Rectified Linear Units (ReLUs)

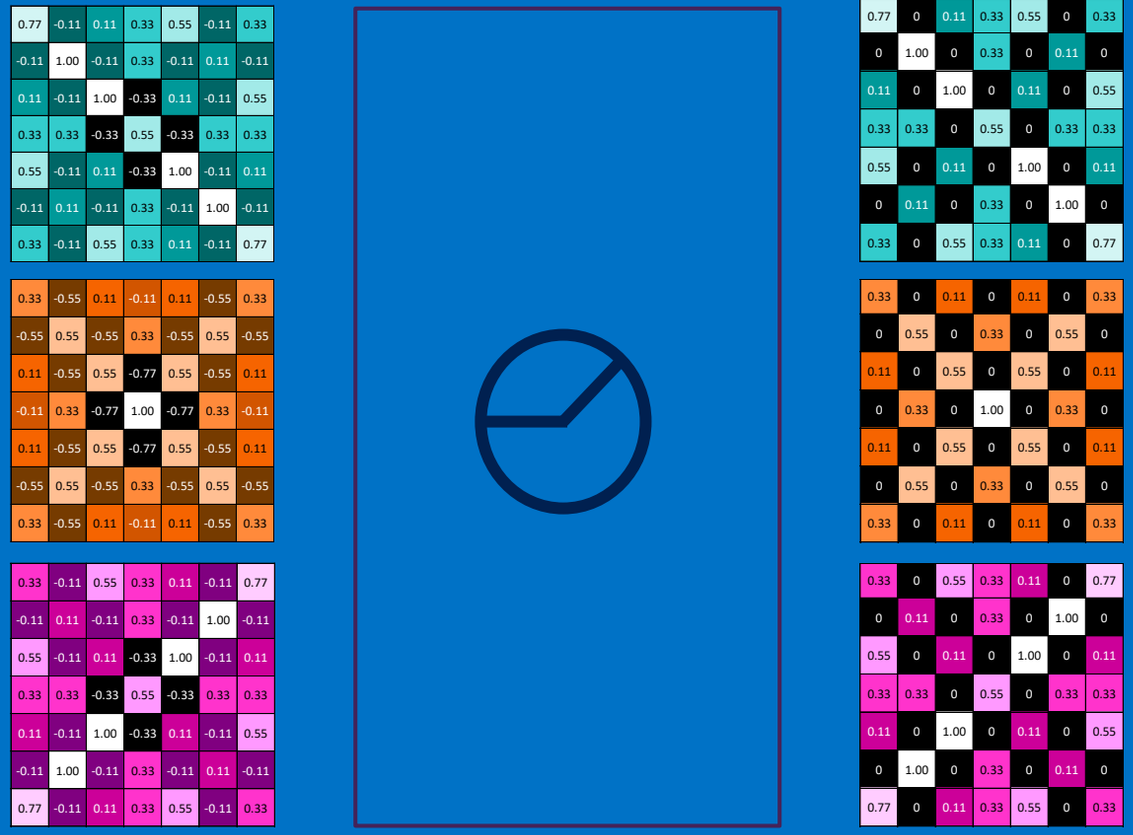


Rectified Linear Units (ReLUs)



ReLU layer

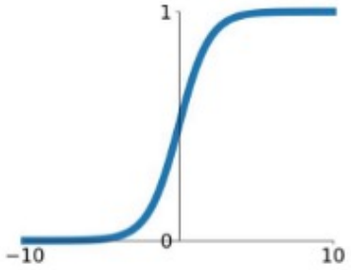
A stack of images becomes a stack of images with no negative values.



Activation Functions

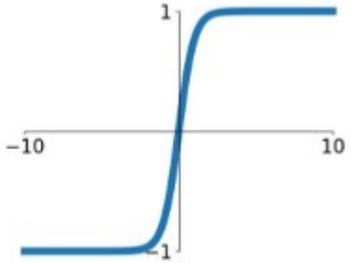
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



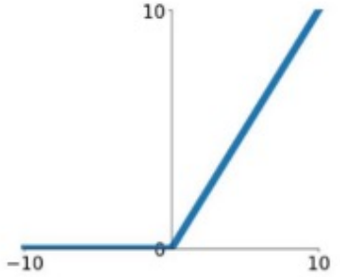
tanh

$$\tanh(x)$$



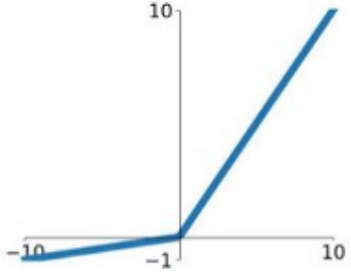
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

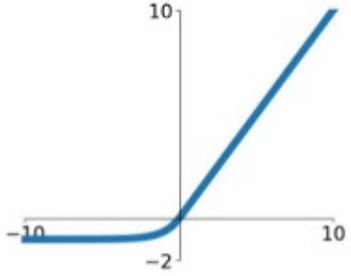


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

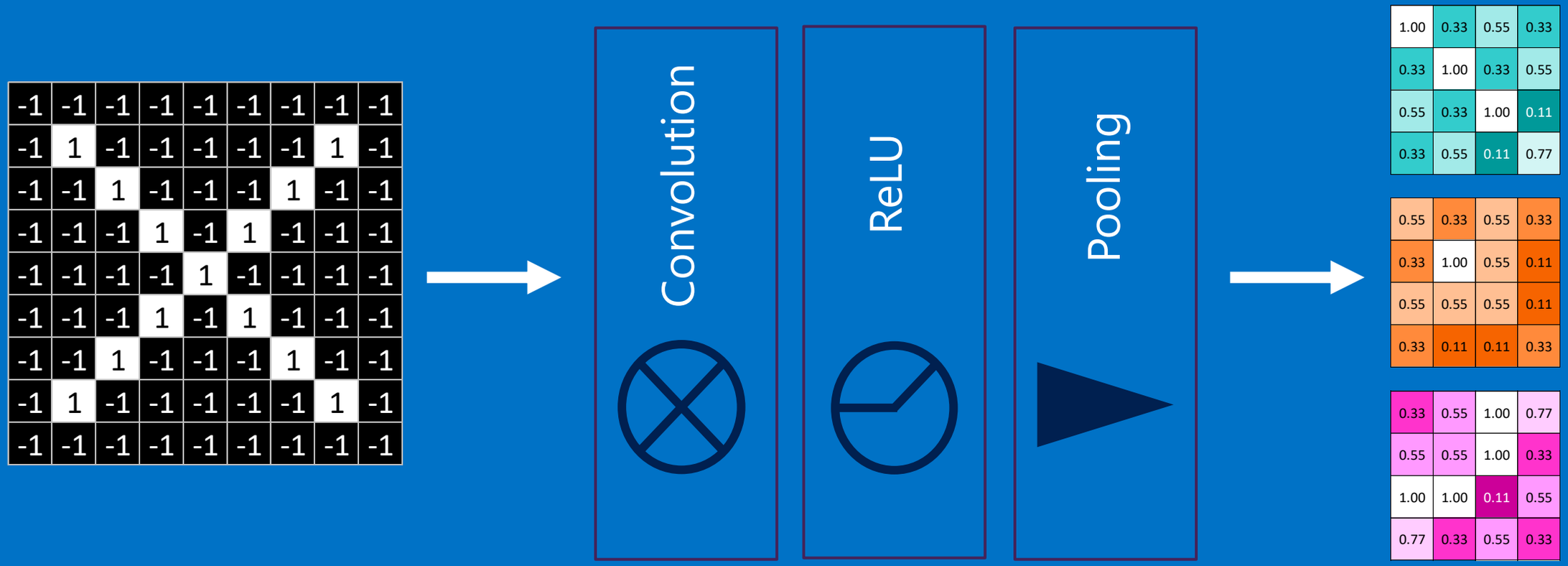
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



The purpose of the activation function is to **introduce non-linearity** into the output of a neuron.

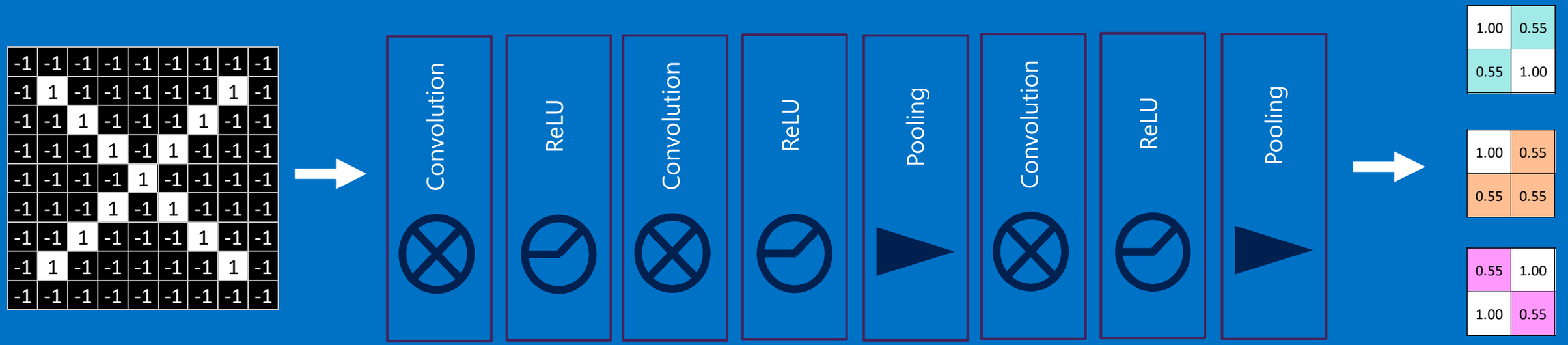
Layers get stacked

The output of one becomes the input of the next.



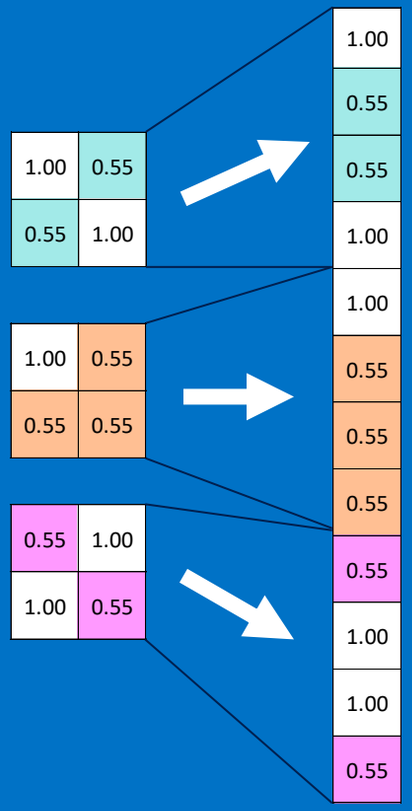
Deep stacking

Layers can be repeated several (or many) times.



Fully connected layer

Every value gets a vote

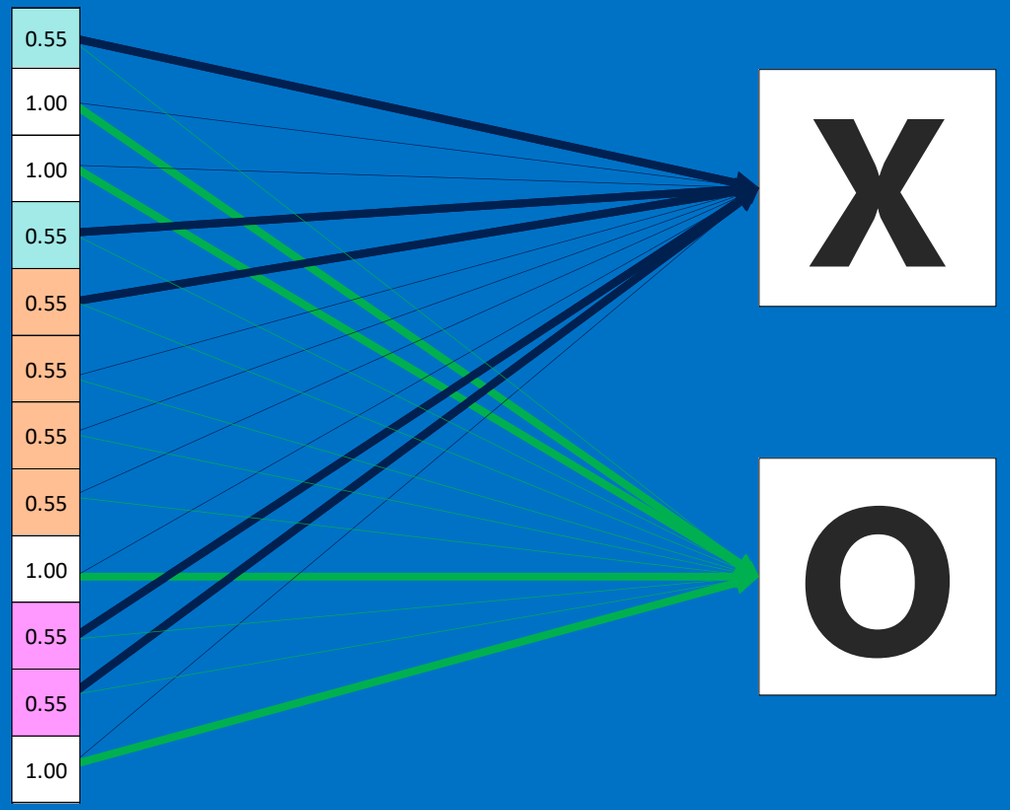


Fully connected layer

Vote depends on how strongly a value predicts X or O

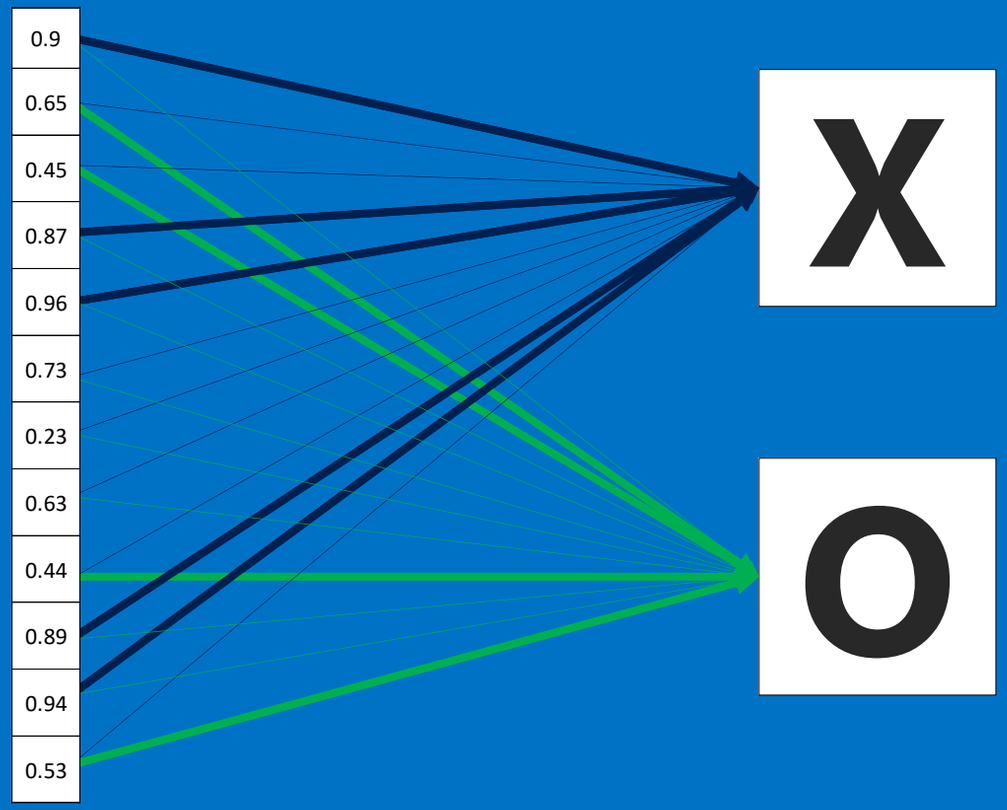
Fully connected layer

Vote depends on how strongly a value predicts X or O



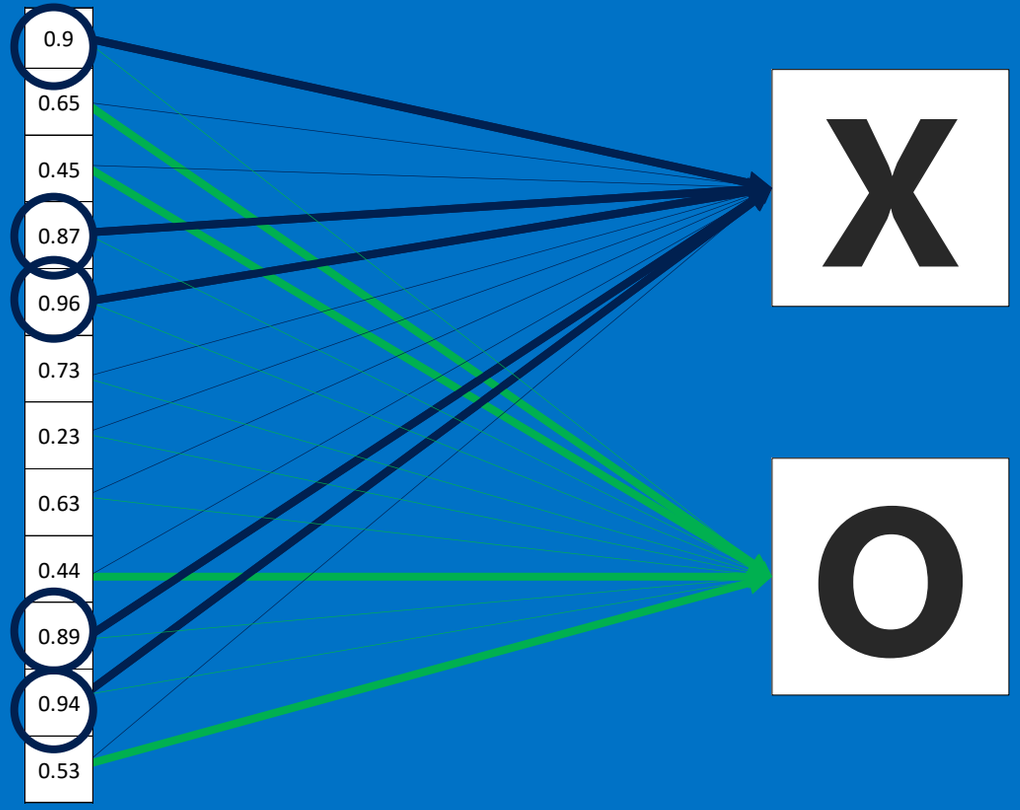
Fully connected layer

Future values vote on X or O



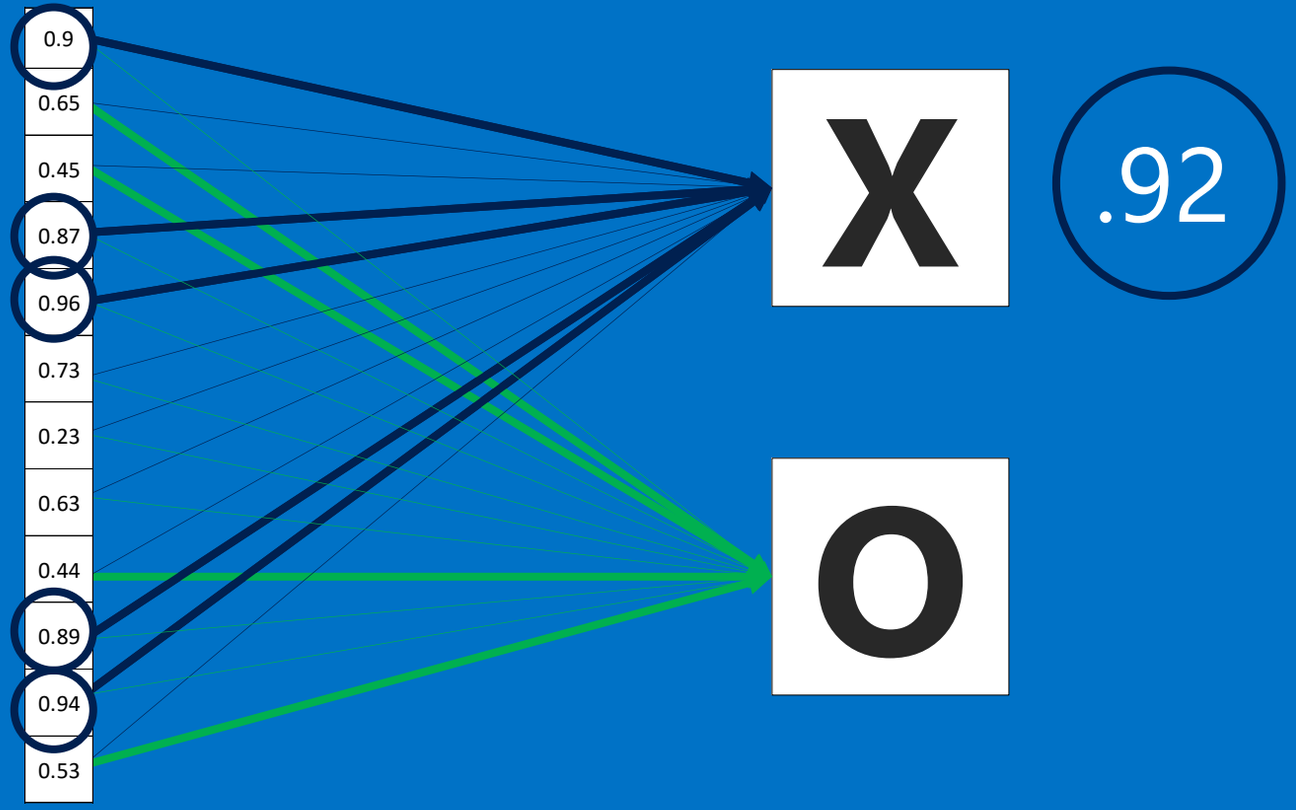
Fully connected layer

Future values vote on X or O



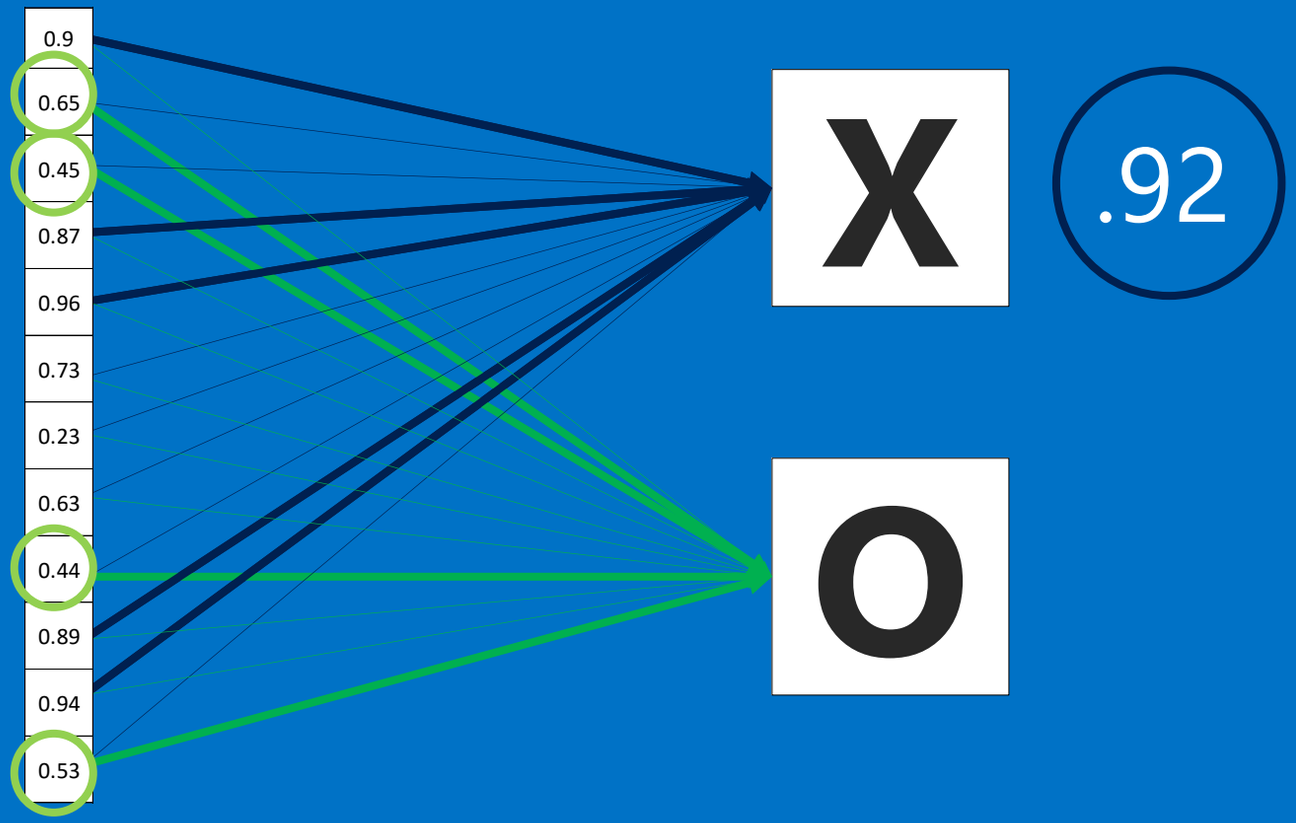
Fully connected layer

Future values vote on X or O



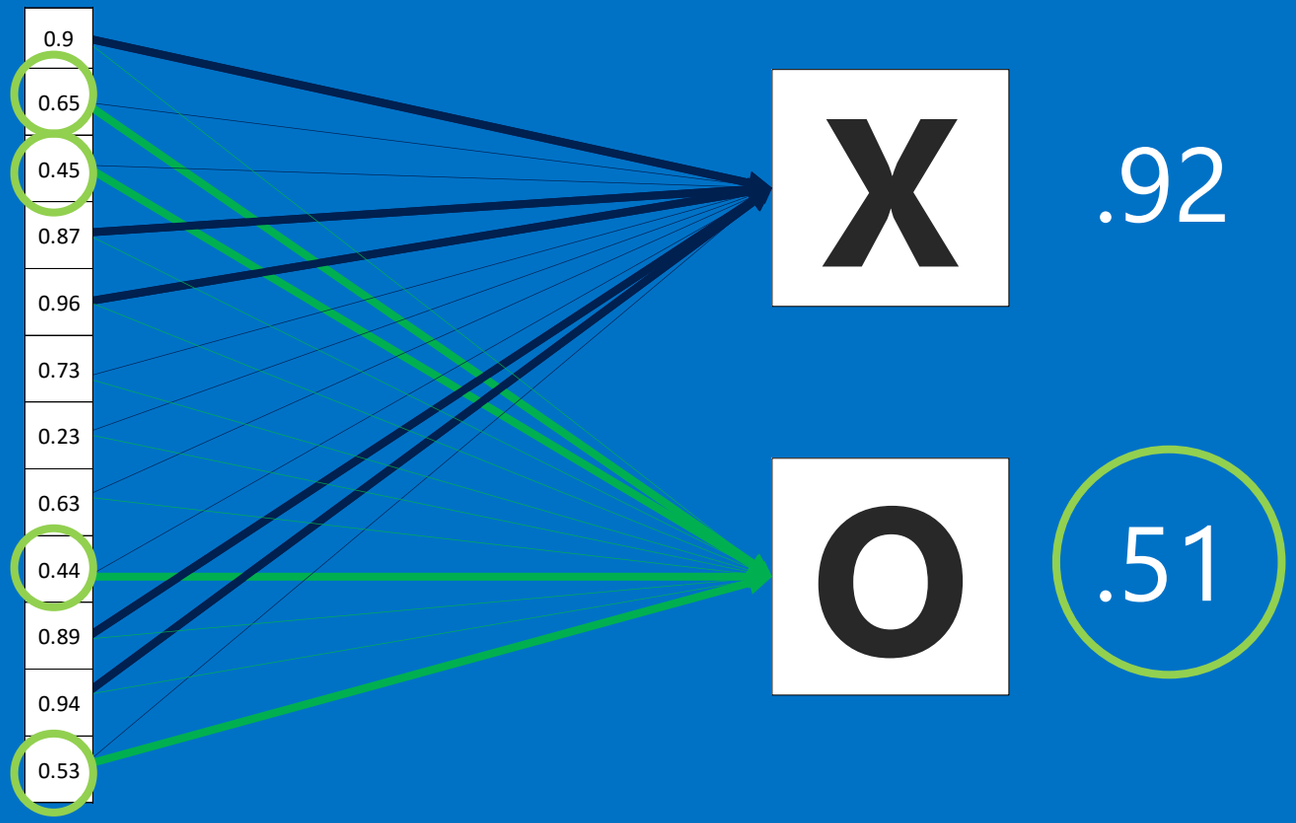
Fully connected layer

Future values vote on X or O



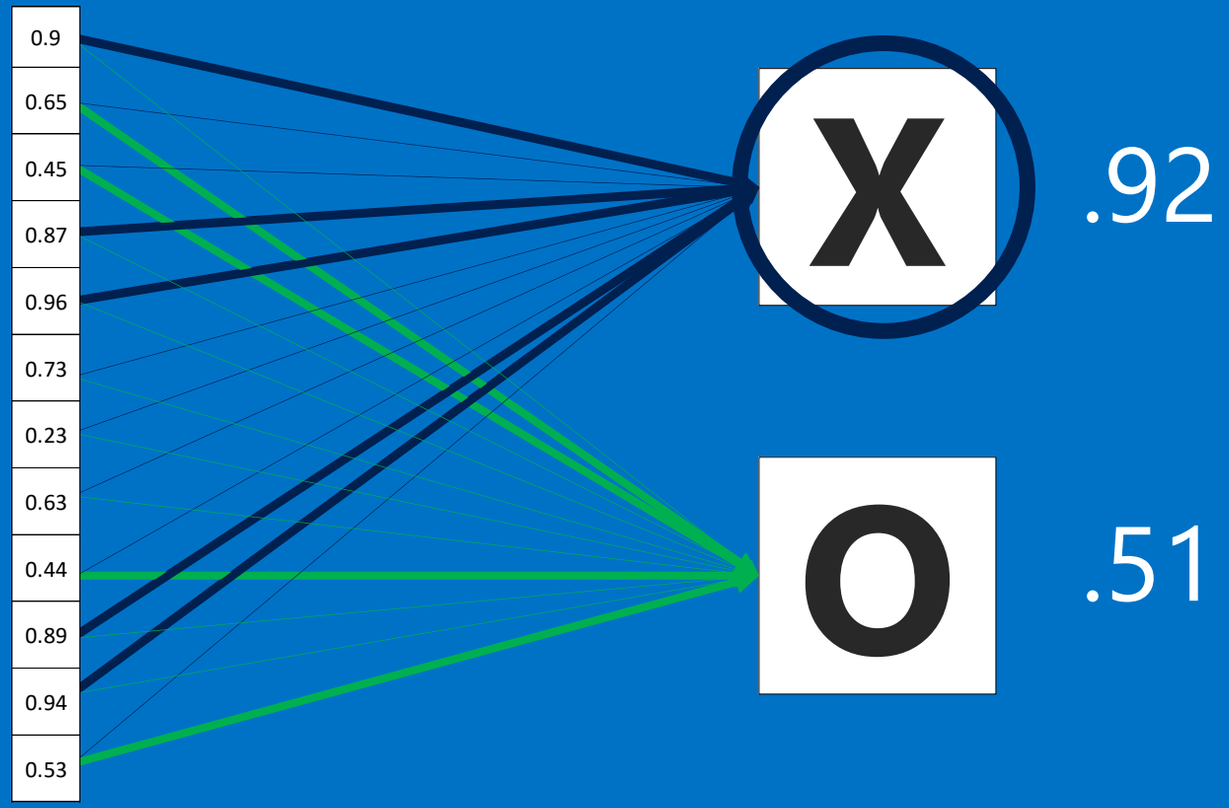
Fully connected layer

Future values vote on X or O



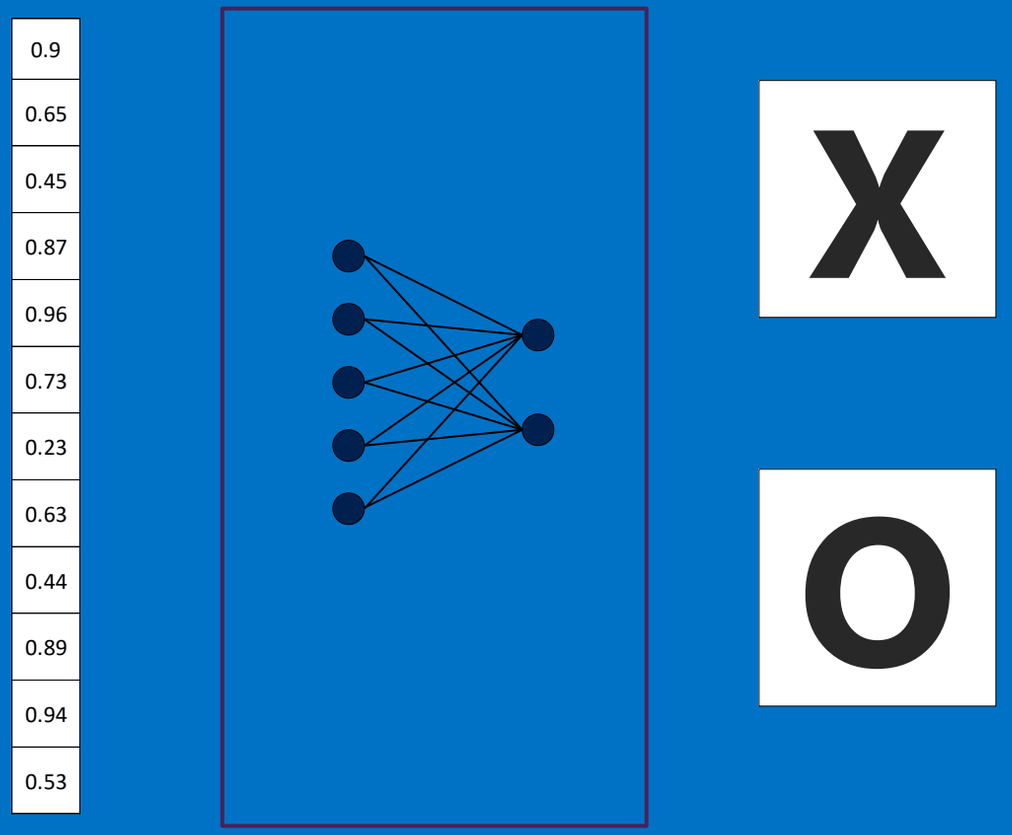
Fully connected layer

Future values vote on X or O



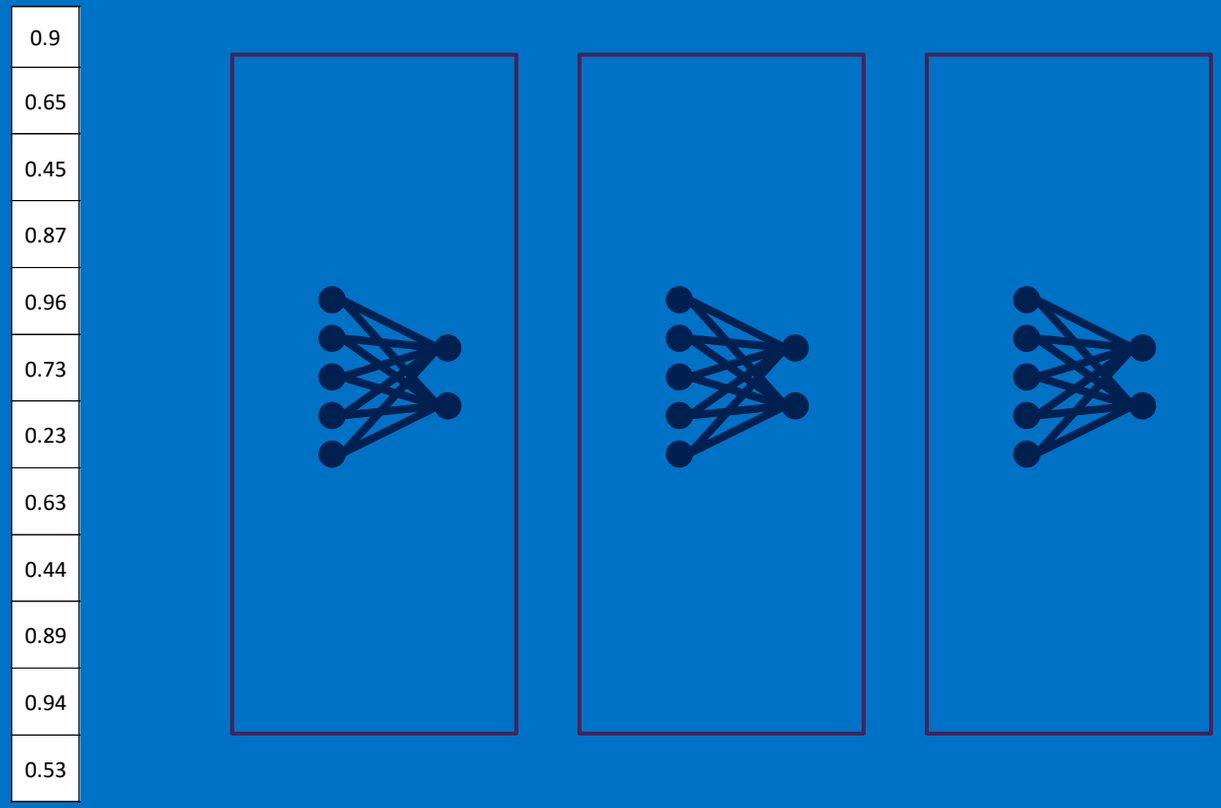
Fully connected layer

A list of feature values becomes a list of votes.



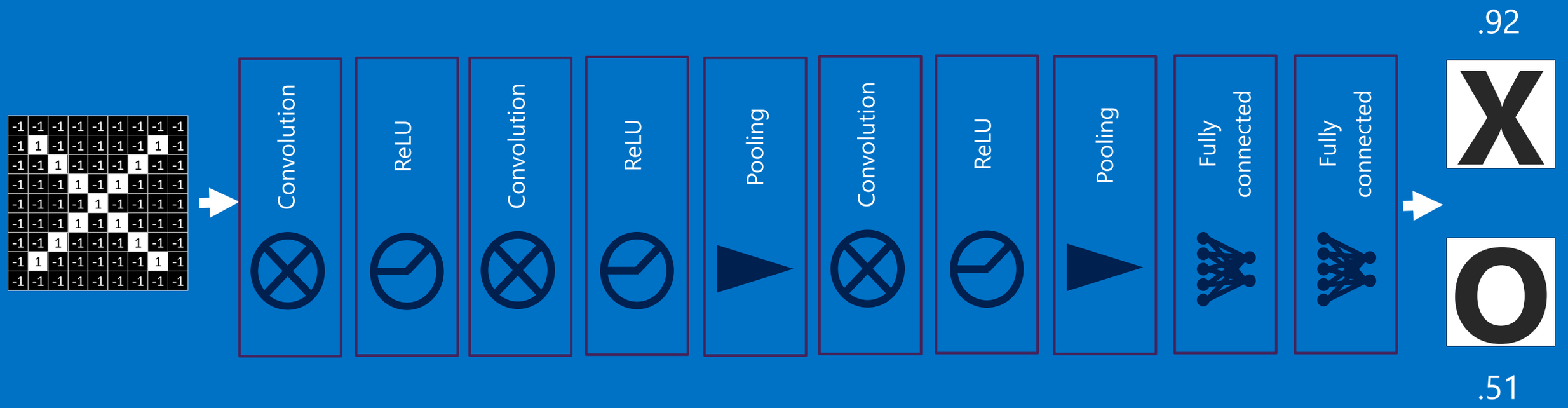
Fully connected layer

These can also be stacked.



Putting it all together

A set of pixels becomes a set of votes.



Learning

Q: Where do all the magic numbers come from?

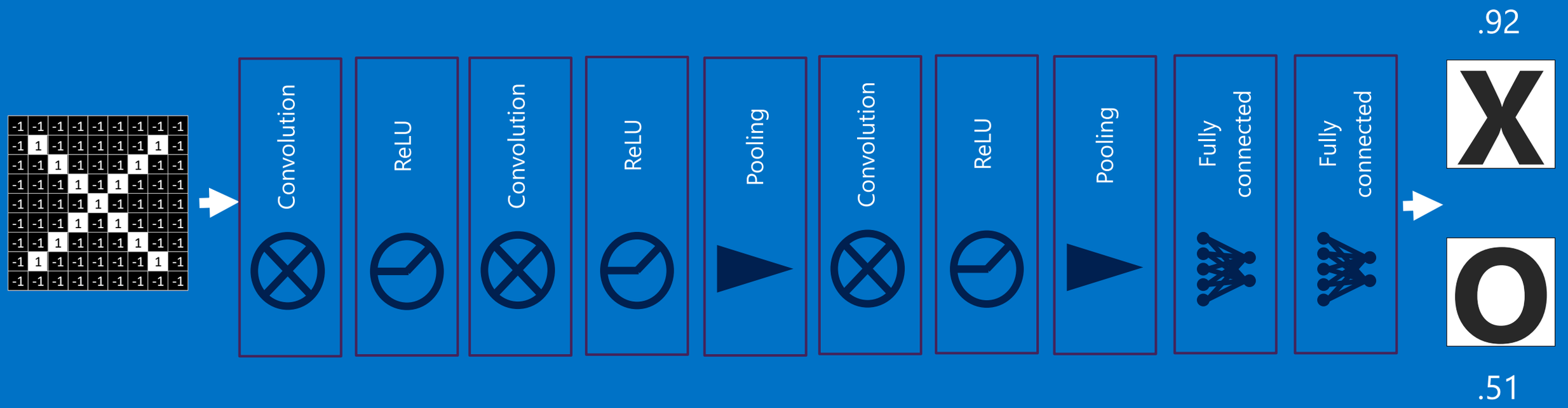
Features in convolutional layers

Voting weights in fully connected layers

A: Backpropagation

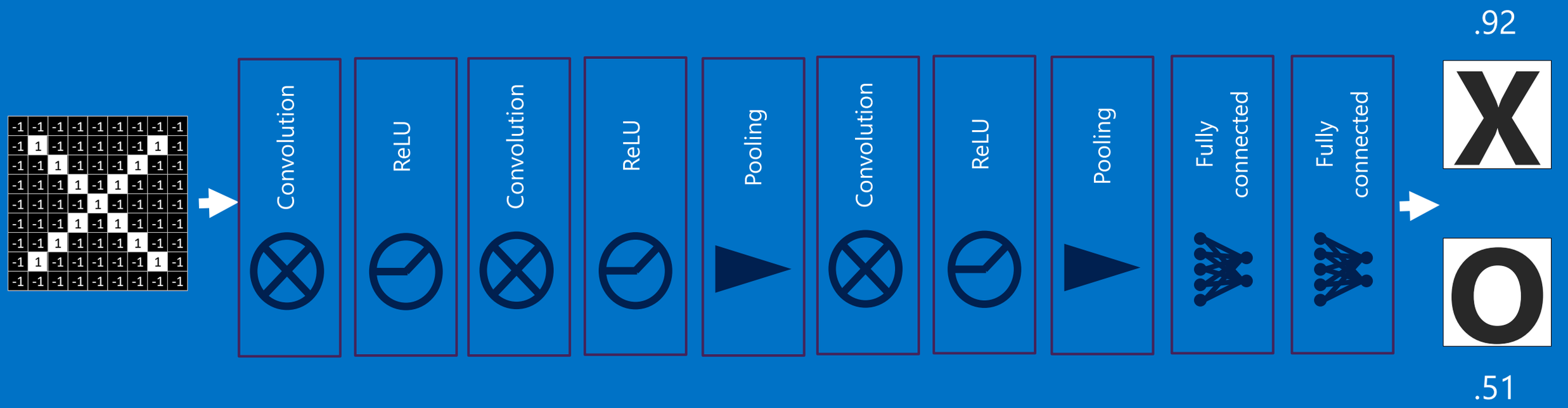
Backprop

Error = right answer – actual answer



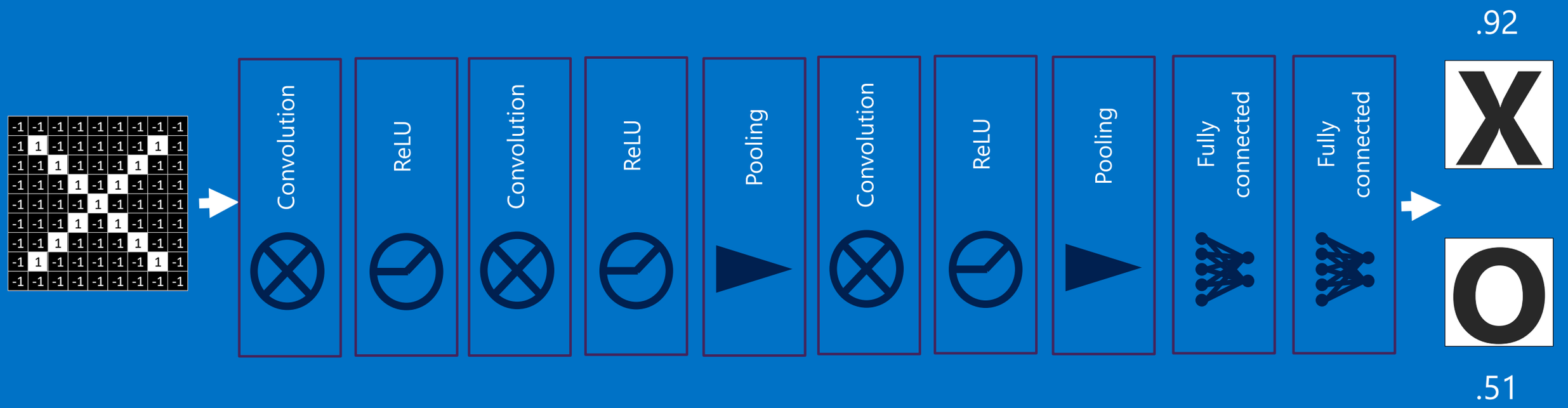
Backprop

	Right answer	Actual answer	Error
X	1		
O			



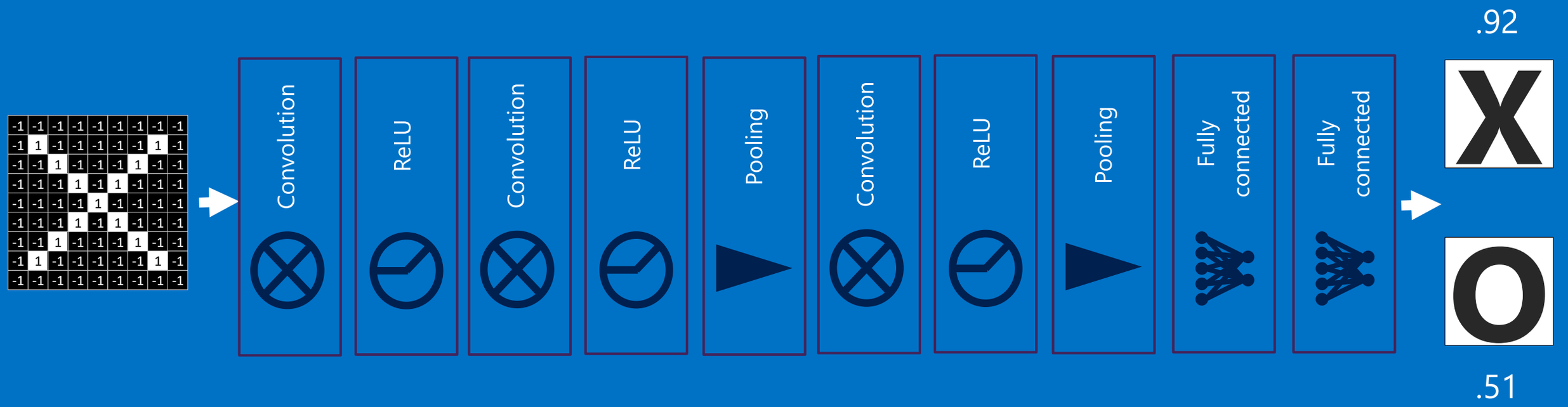
Backprop

	Right answer	Actual answer	Error
X	1	0.92	
O			



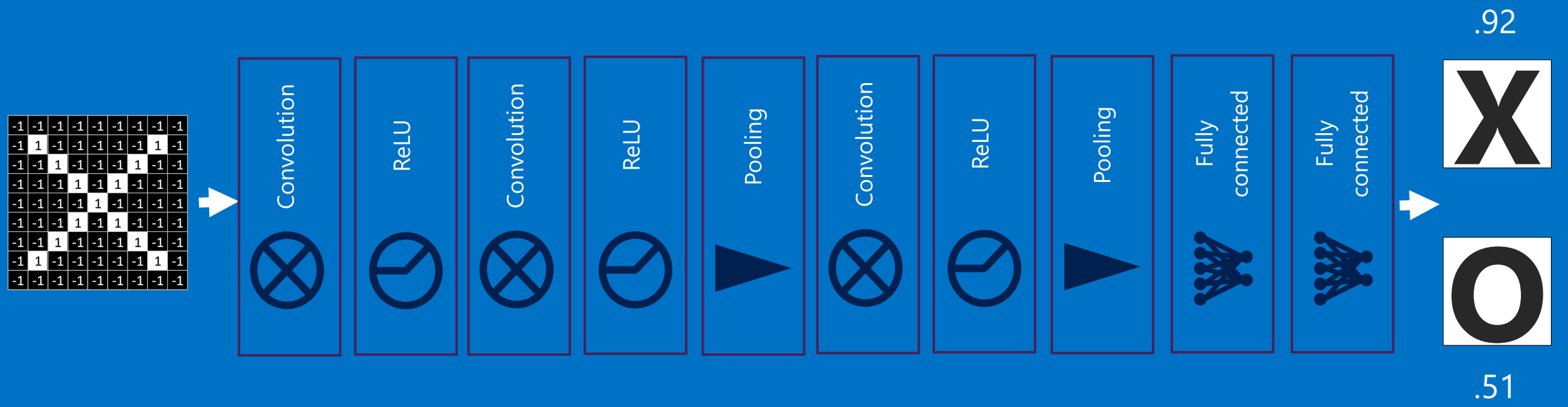
Backprop

	Right answer	Actual answer	Error
X	1	0.92	0.08
O			



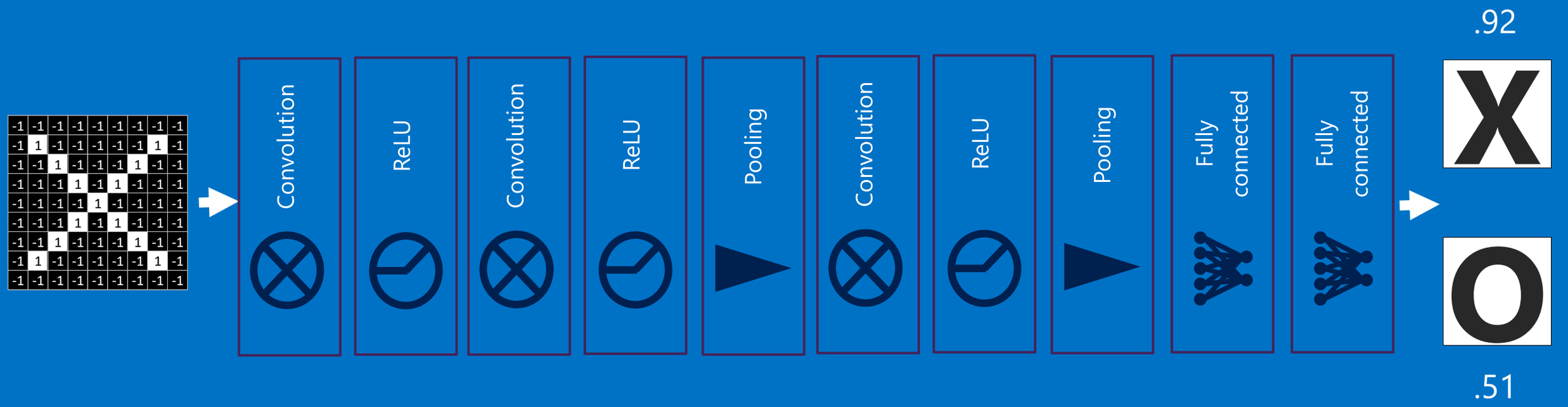
Backprop

	Right answer	Actual answer	Error
X	1	0.92	0.08
O	0	0.51	0.49



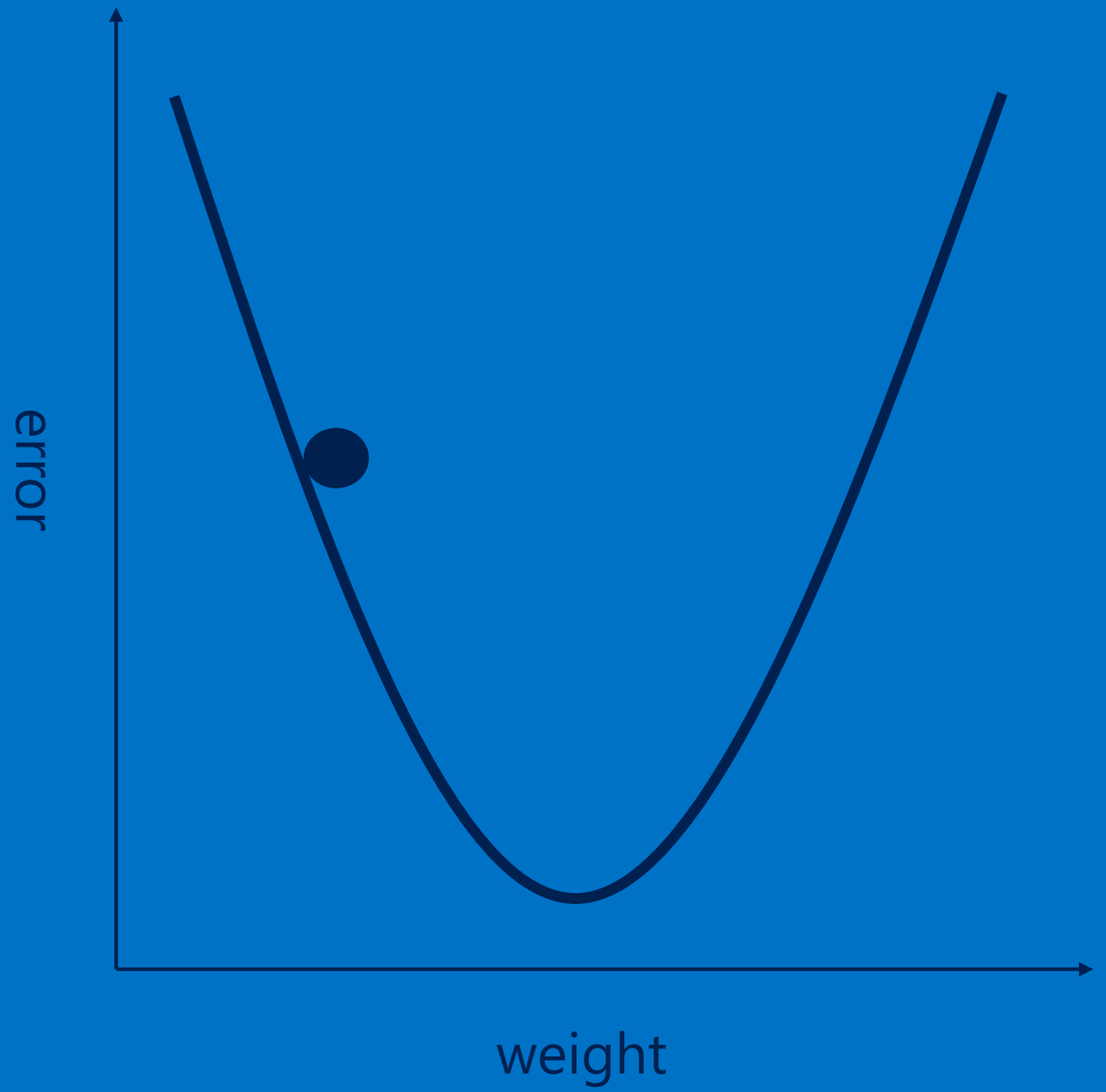
Backprop

	Right answer	Actual answer	Error
X	1	0.92	0.08
O	0	0.51	0.49
		Total	0.57



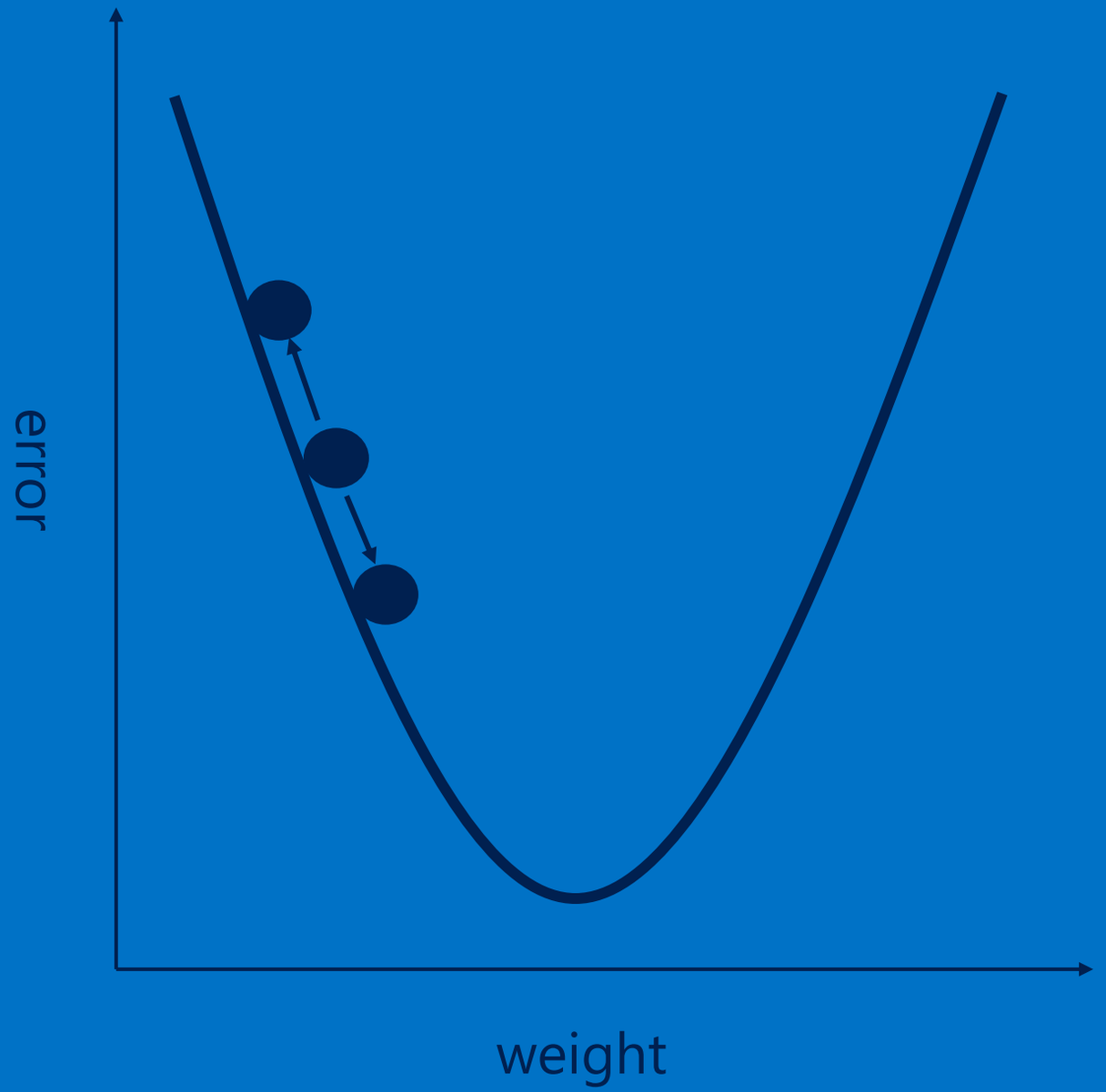
Gradient descent

For each feature pixel and voting weight, adjust it up and down a bit and see how the error changes.



Gradient descent

For each feature pixel and voting weight, adjust it up and down a bit and see how the error changes.



Hyperparameters (knobs)

- Convolution
  - Number of features
  - Size of features
- Pooling
  - Window size
  - Window stride
- Fully Connected
  - Number of neurons

Architecture

How many of each type of layer?

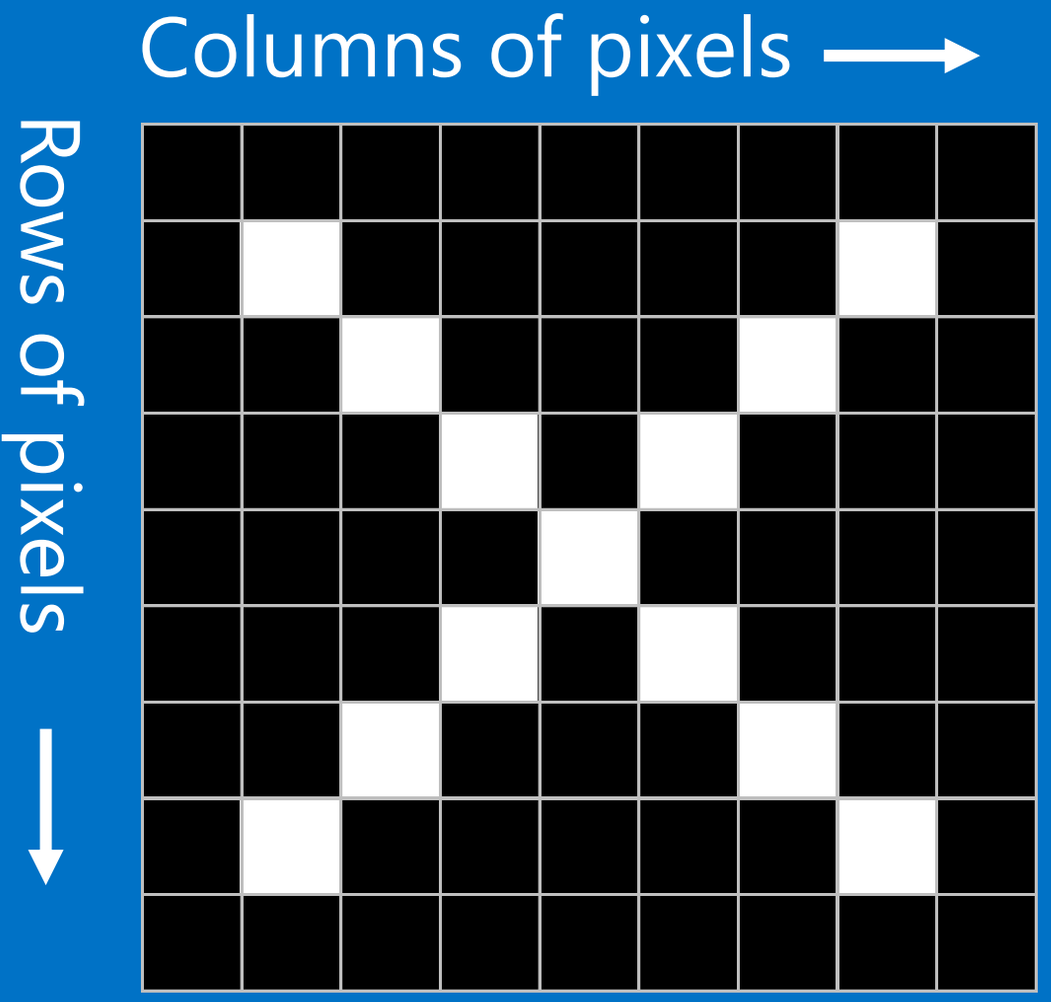
In what order?

Not just images

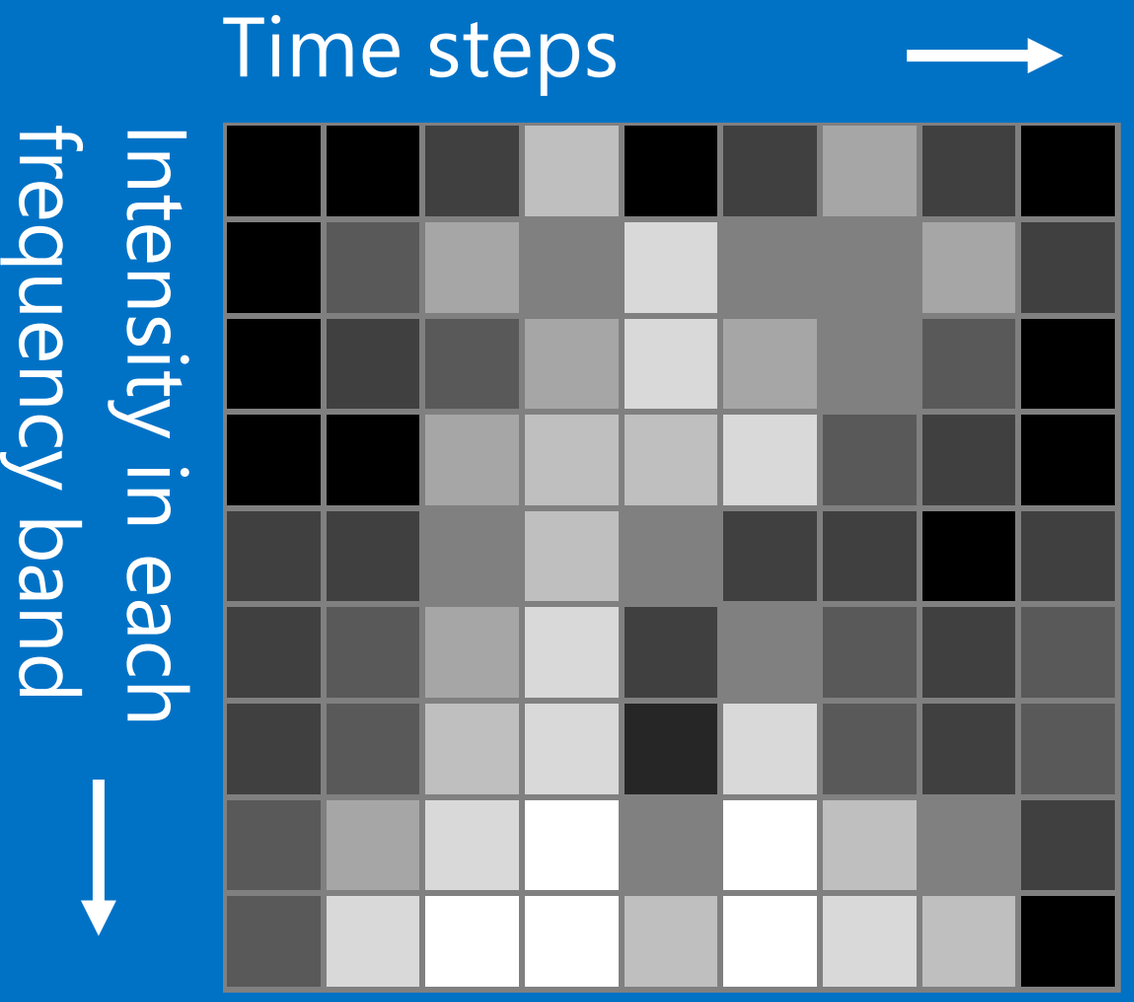
Any 2D (or 3D) data.

Things closer together are more closely related than things far away.

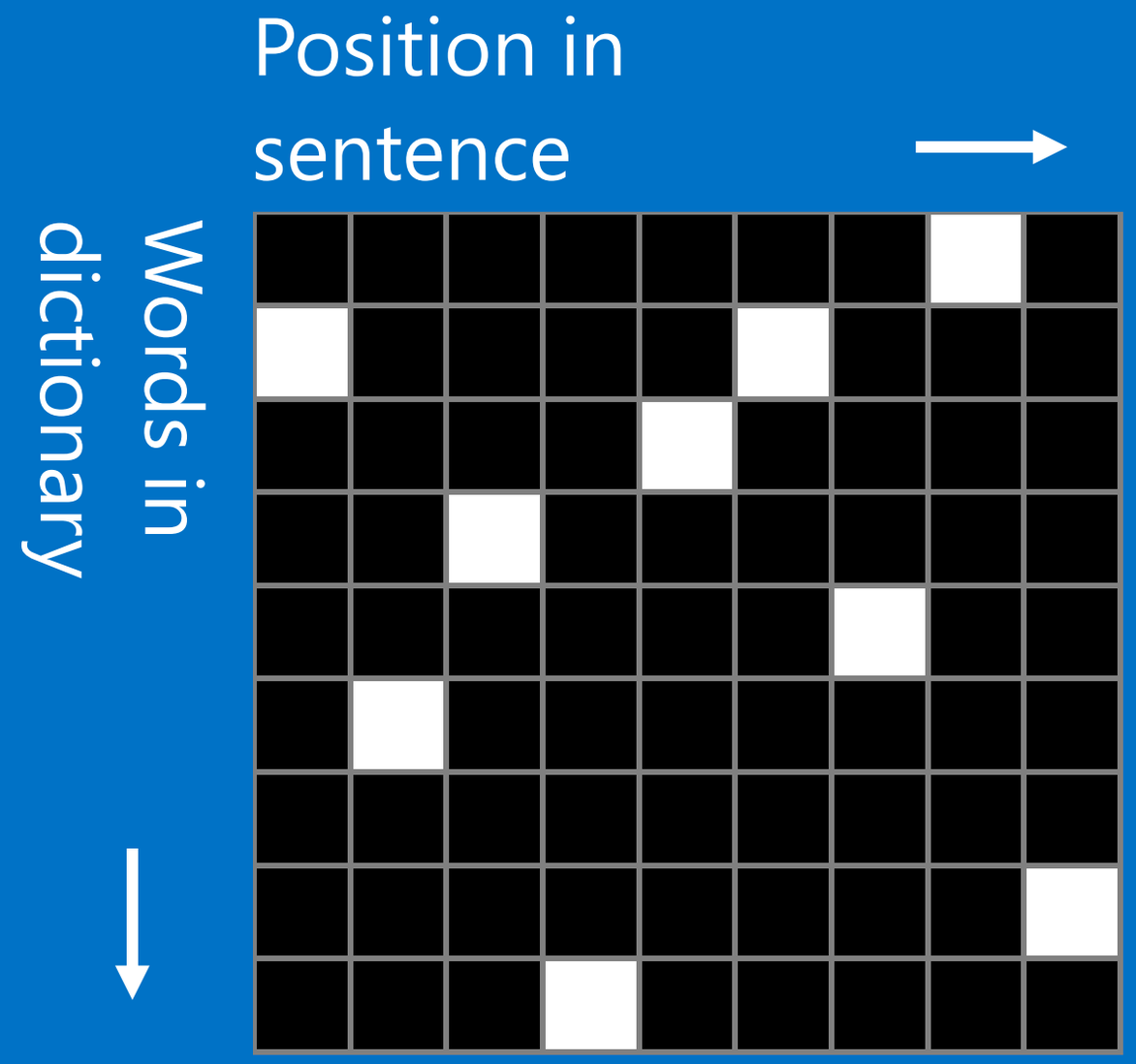
Images



Sound



Text



Limitations

ConvNets only capture local “spatial” patterns in data.

If the data can’t be made to look like an image, ConvNets are less useful.

In a nutshell

ConvNets are great at finding patterns and using them to classify images.

Difference between Batch and Epoch

- The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset.

- One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters. An epoch is comprised of one or more batches. For example, as above, an epoch that has one batch is called the batch gradient descent learning algorithm.

- You can think of a for-loop over the number of epochs where each loop proceeds over the training dataset. Within this for-loop is another nested for-loop that iterates over each batch of samples, where one batch has the specified “batch size” number of samples.

- The number of epochs is traditionally large, often hundreds or thousands, allowing the learning algorithm to run until the error from the model has been sufficiently minimized. You may see examples of the number of epochs in the literature and in tutorials set to 10, 100, 500, 1000, and larger.

- It is common to create line plots that show epochs along the x-axis as time and the error or skill of the model on the y-axis. These plots are sometimes called learning curves. These plots can help to diagnose whether the model has over learned, under learned, or is suitably fit to the training dataset.

Data Pre-processing

In the process of training our network, we're going to be multiplying (weights) and adding to (biases) these initial inputs in order to cause activations that we then backpropagate with the gradients to train the model.

There are some variations on how to normalize the images but most seem to use these two methods:

1.Subtract the mean per channel calculated over all images

2.Subtract by pixel/channel calculated over all images

Deep Backbone networks

- "backbone" refers to the feature extracting network which is used within the Deep architecture. This feature extractor is used to encode the network's input into a certain feature representation.

- VGG
- Resnet-18, Resnet-50, Resnet-101
- Desnet-121, Desnet-201

Fine-Tunning

Segmentation Network

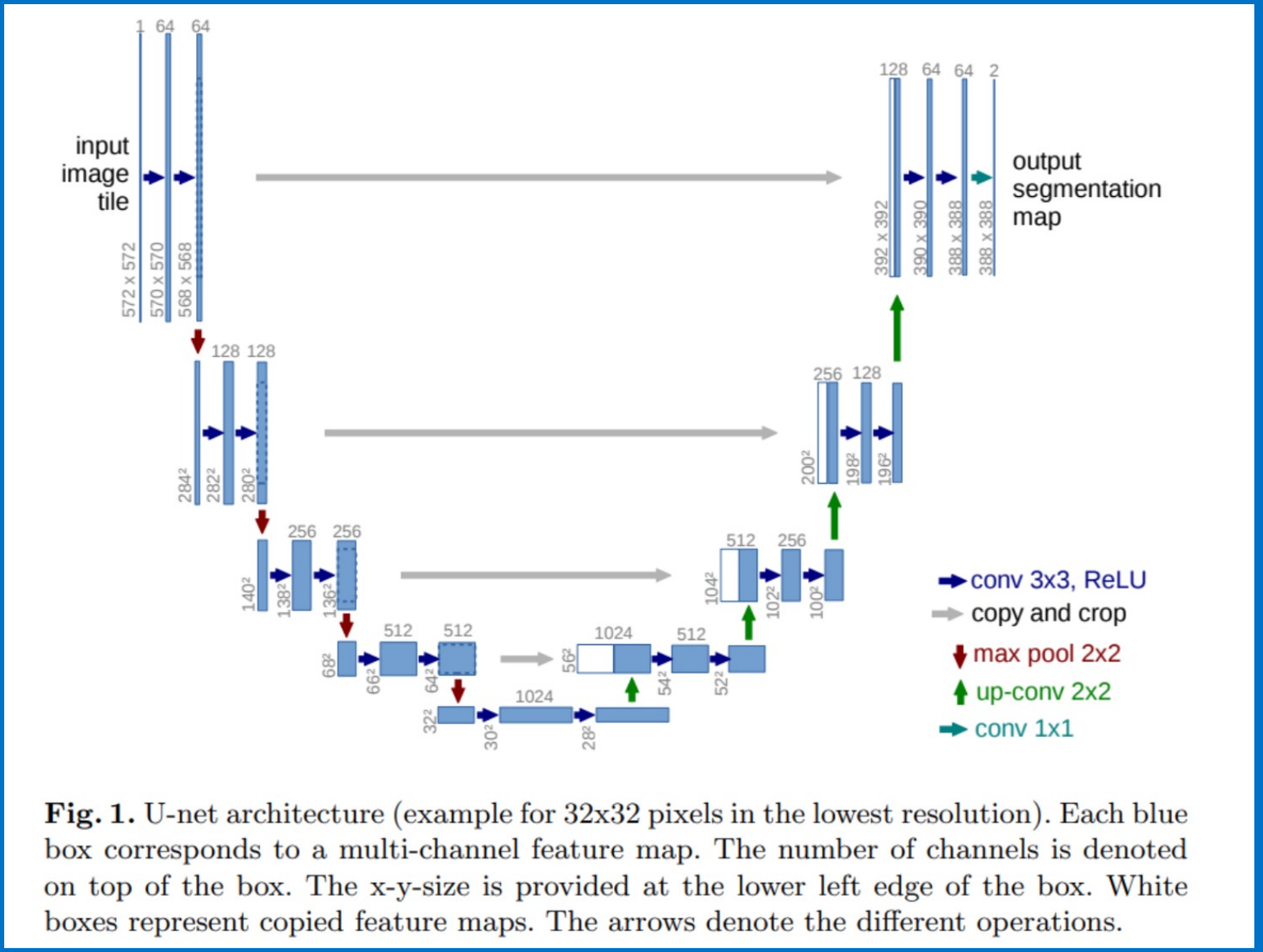
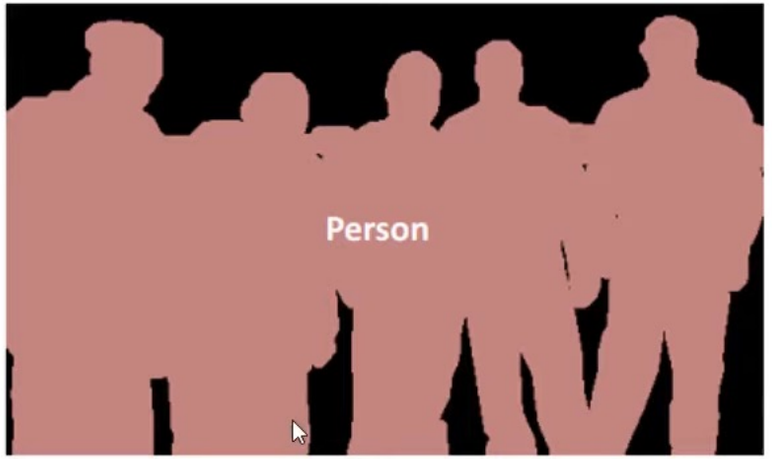
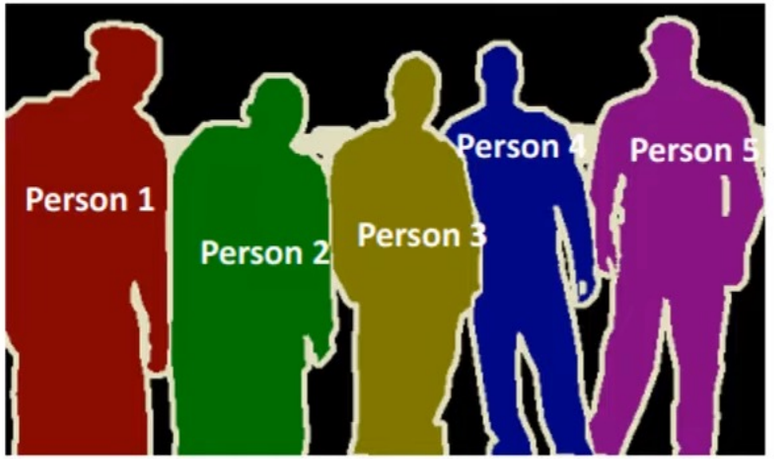


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

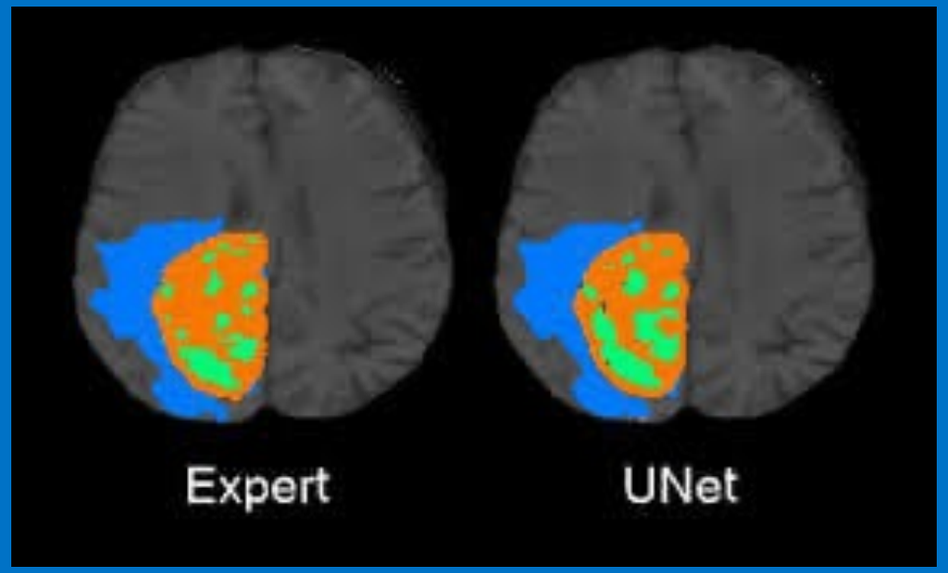
Object Detection

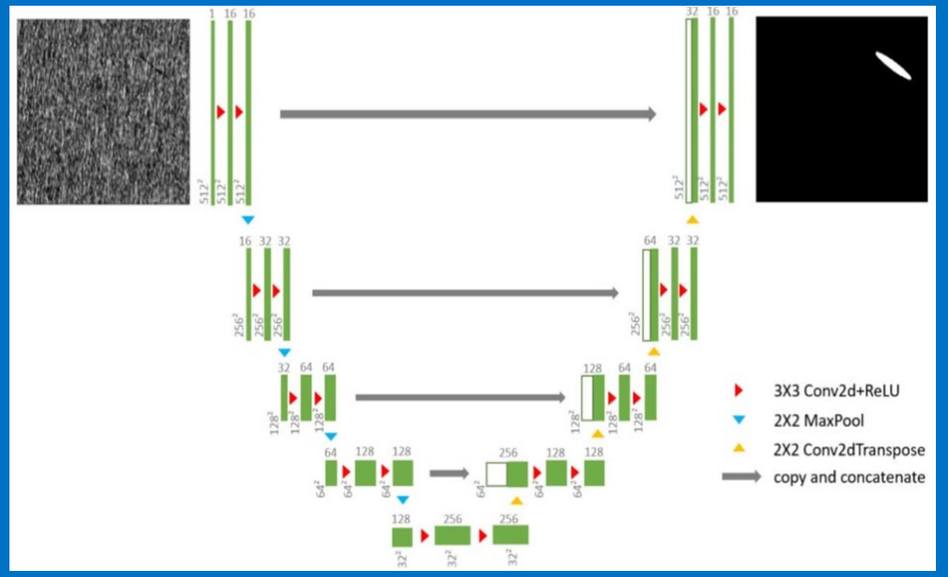


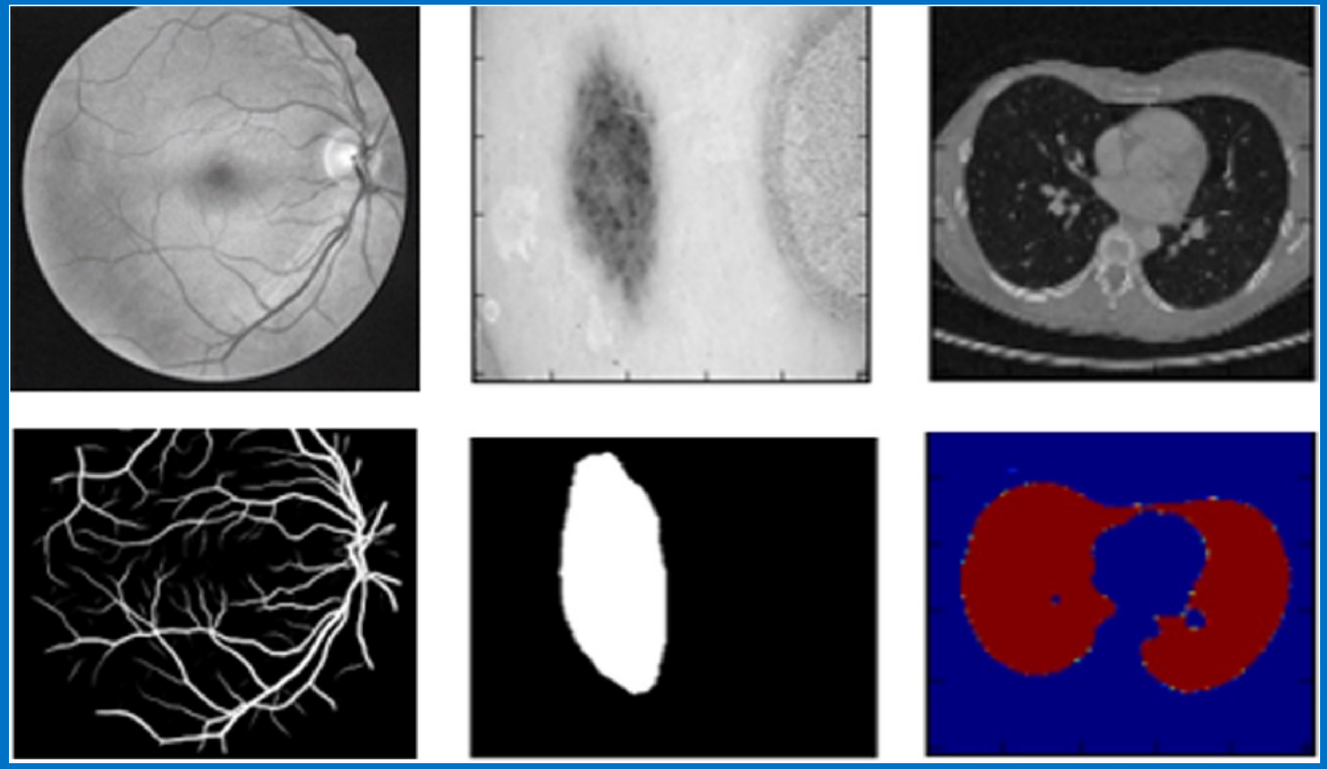
Semantic Segmentation

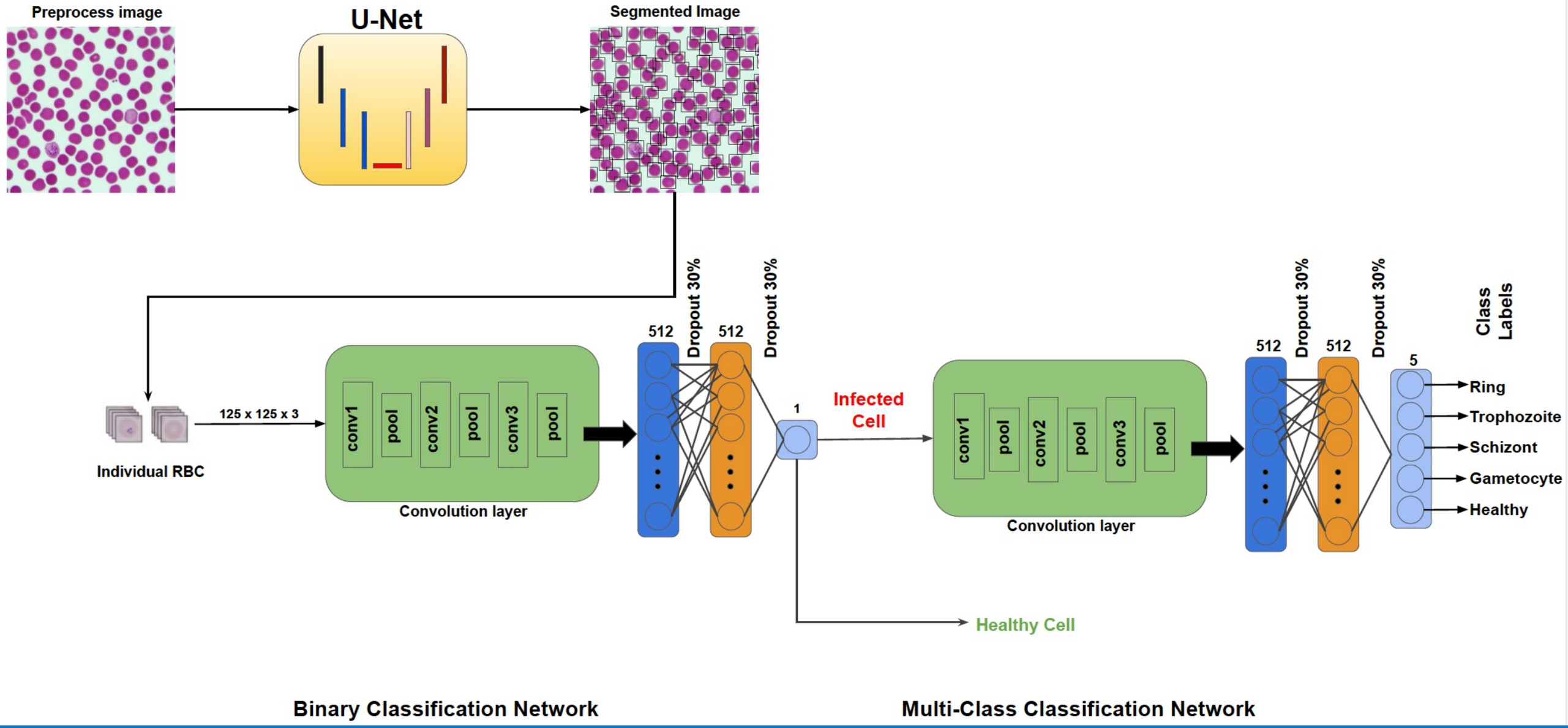


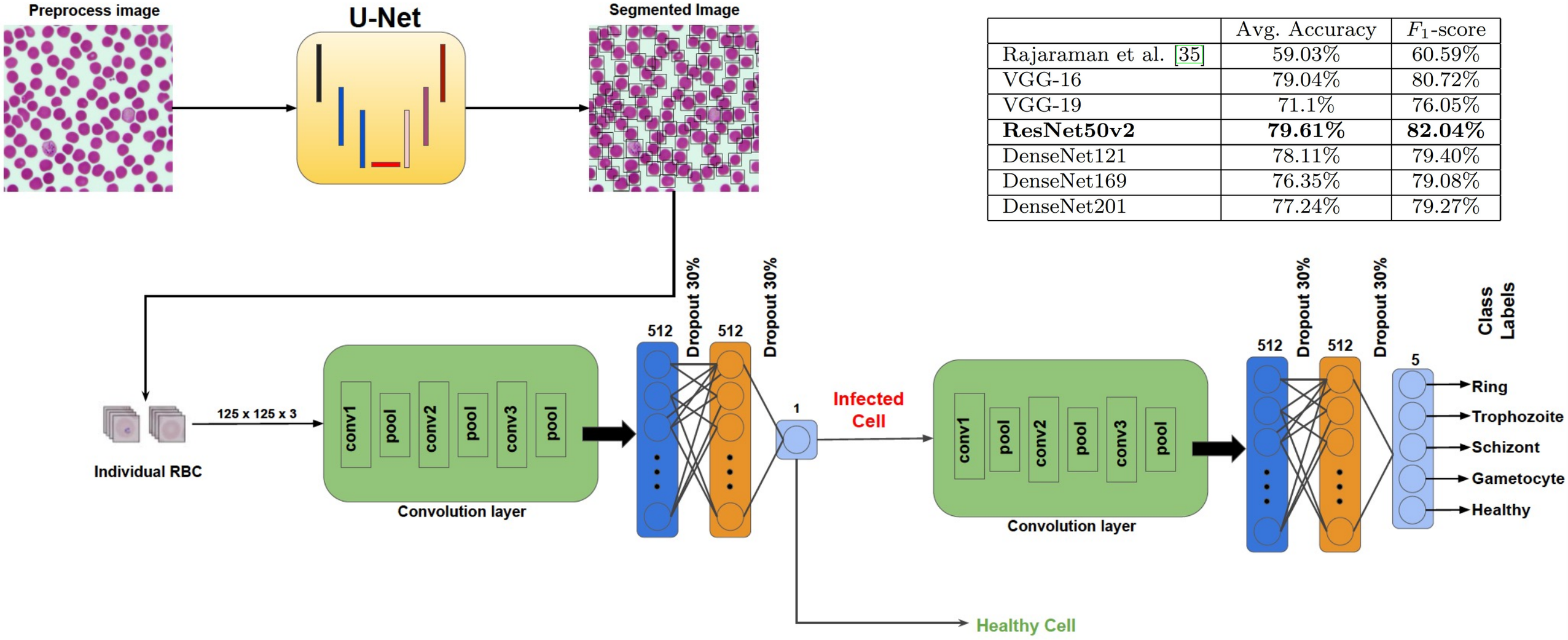
Instance Segmentation


Expert
UNet


3X3 Conv2d+ReLU
2X2 MaxPool
2X2 Conv2dTranspose
copy and concatenate




Preprocess image
U-Net
Segmented Image
125 x 125 x 3
Individual RBC
conv1
pool
conv2
pool
conv3
pool
Convolution layer
512
Dropout 30%
512
Dropout 30%
1
Infected Cell
Healthy Cell
conv1
pool
conv2
pool
conv3
pool
Convolution layer
512
Dropout 30%
512
Dropout 30%
5
Class Labels
Ring
Trophozoite
Schizont
Gametocyte
Healthy
Binary Classification Network
Multi-Class Classification Network



	Avg. Accuracy	F_1-score
Rajaraman et al. [35]	59.03%	60.59%
VGG-16	79.04%	80.72%
VGG-19	71.1%	76.05%
ResNet50v2	**79.61%**	**82.04%**
DenseNet121	78.11%	79.40%
DenseNet169	76.35%	79.08%
DenseNet201	77.24%	79.27%

PSPNet

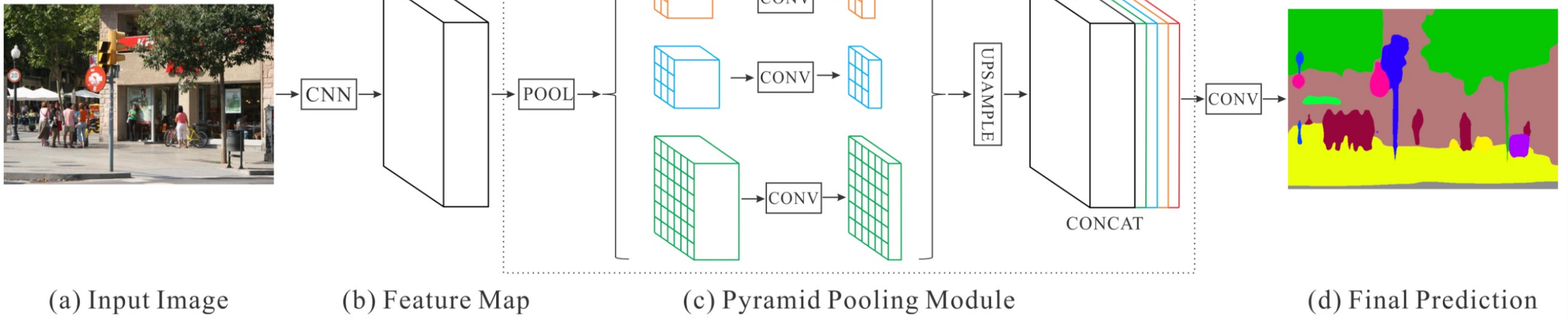
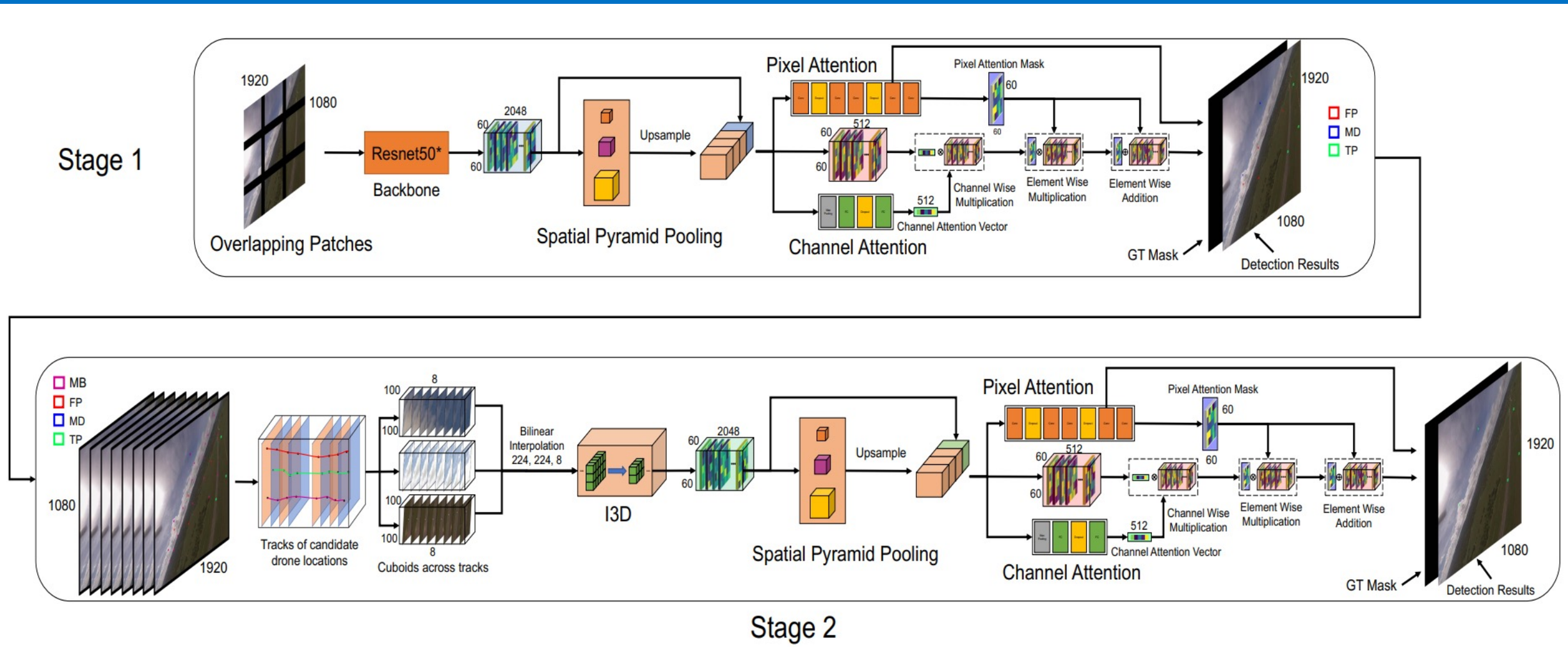


Figure 1. Overview of our proposed PSPNet. Given an input image (a), we first use CNN to get the feature map of the last convolutional layer (b), then a pyramid parsing module is applied to harvest different sub-region representations, followed by upsampling and concatenation layers to form the final feature representation, which carries both local and global context information in (c). Finally, the representation is fed into a convolution layer to get the final per-pixel prediction (d).


Stage 1
1920
1080
Resnet50*
Backbone
2048
60
60
Upsample
Overlapping Patches
Spatial Pyramid Pooling
Pixel Attention
Pixel Attention Mask
60
60
512
60
Channel Wise Multiplication
Element Wise Multiplication
Element Wise Addition
512
Channel Attention Vector
Channel Attention
1920
1080
GT Mask
Detection Results
FP
MD
TP
MB
FP
MD
TP
1080
1920
Tracks of candidate drone locations
8
100
100
100
100
8
Cuboids across tracks
Bilinear Interpolation 224, 224, 8
I3D
60
2048
60
Spatial Pyramid Pooling
Upsample
Pixel Attention
Pixel Attention Mask
60
60
512
60
Channel Wise Multiplication
Element Wise Multiplication
Element Wise Addition
512
Channel Attention Vector
Channel Attention
1920
1080
GT Mask
Detection Results
Stage 2

Recurrent Neural Network

- Recurrent neural networks are networks that are designed to interpret temporal or sequential information.

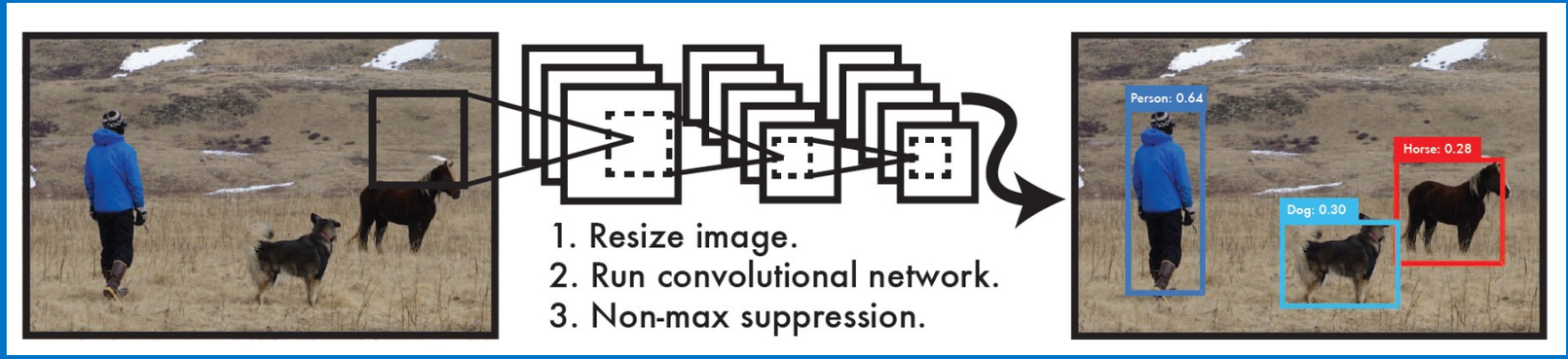
- RNNs use other data points in a sequence to make better predictions. They do this by taking in input and reusing the activations of previous nodes or later nodes in the sequence to influence the output.

- A recurrent network whose inputs are not fixed but rather constitute an input sequence can be used to transform an input sequence into an output sequence while taking into account contextual information in a flexible way.

- Long Short-Term Memory Networks

Object Detector

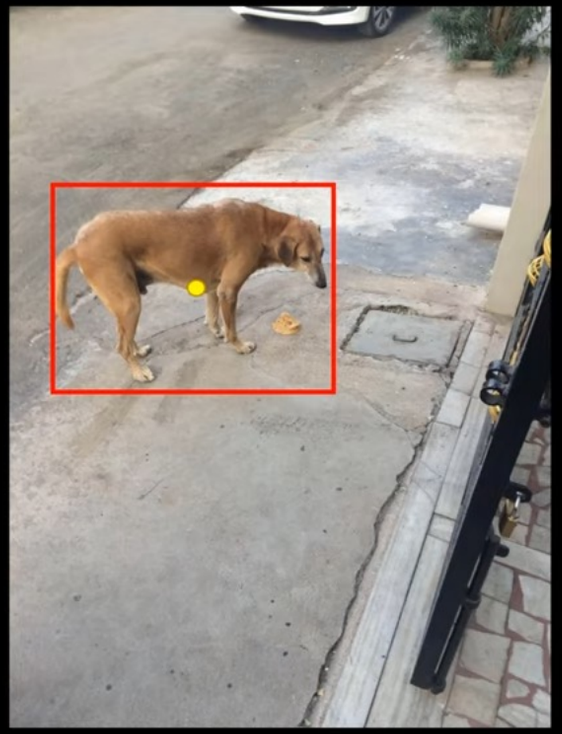
- Faster RCNN
- YOLO



https://www.youtube.com/watch?v=ag3DLKsl2vk

YOLO

Object Localization

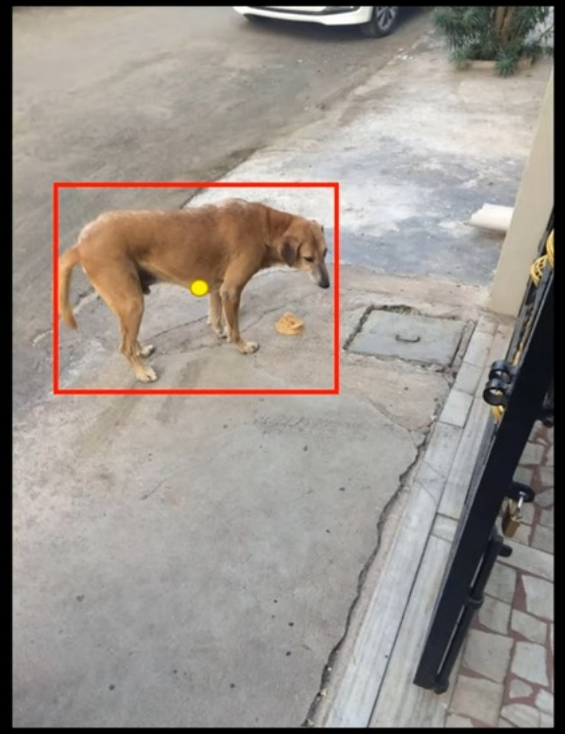


$$\begin{bmatrix} P_c \\ B_x \\ B_y \\ B_w \\ B_h \\ C_1 \\ C_2 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 50 \\ 70 \\ 60 \\ 70 \\ 1 \\ 0 \end{bmatrix}$$

C_1 = *Dog class*
C2 = Person Class

YOLO

Object Localization

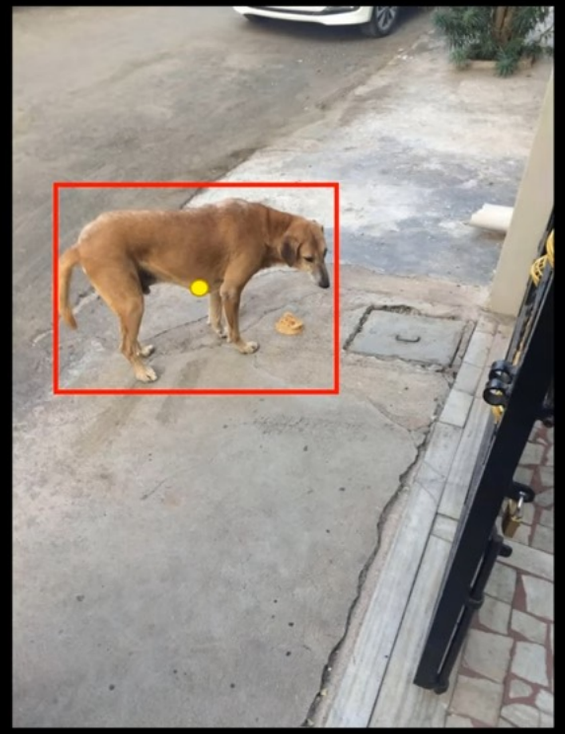


$$\begin{bmatrix} P_c \\ B_x \\ B_y \\ B_w \\ B_h \\ C_1 \\ C_2 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 50 \\ 70 \\ 60 \\ 70 \\ 1 \\ 0 \end{bmatrix}$$

C_1 = *Dog class*
C2 = Person Class

$$\begin{bmatrix} 1 \\ 30 \\ 28 \\ 28 \\ 82 \\ 0 \\ 1 \end{bmatrix}$$

Object Localization

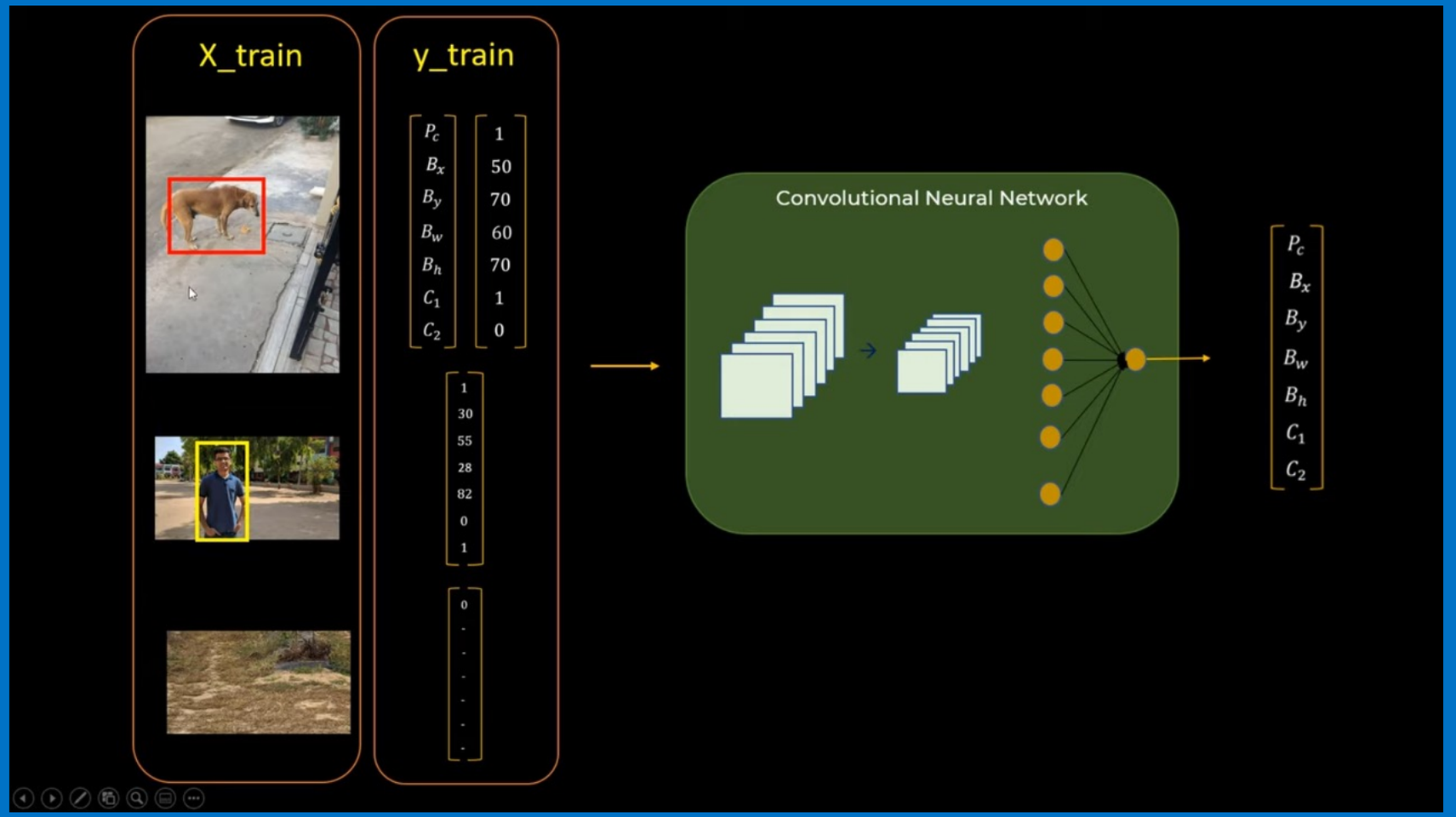


$$\begin{bmatrix} P_c \\ B_x \\ B_y \\ B_w \\ B_h \\ C_1 \\ C_2 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 50 \\ 70 \\ 60 \\ 70 \\ 1 \\ 0 \end{bmatrix}$$

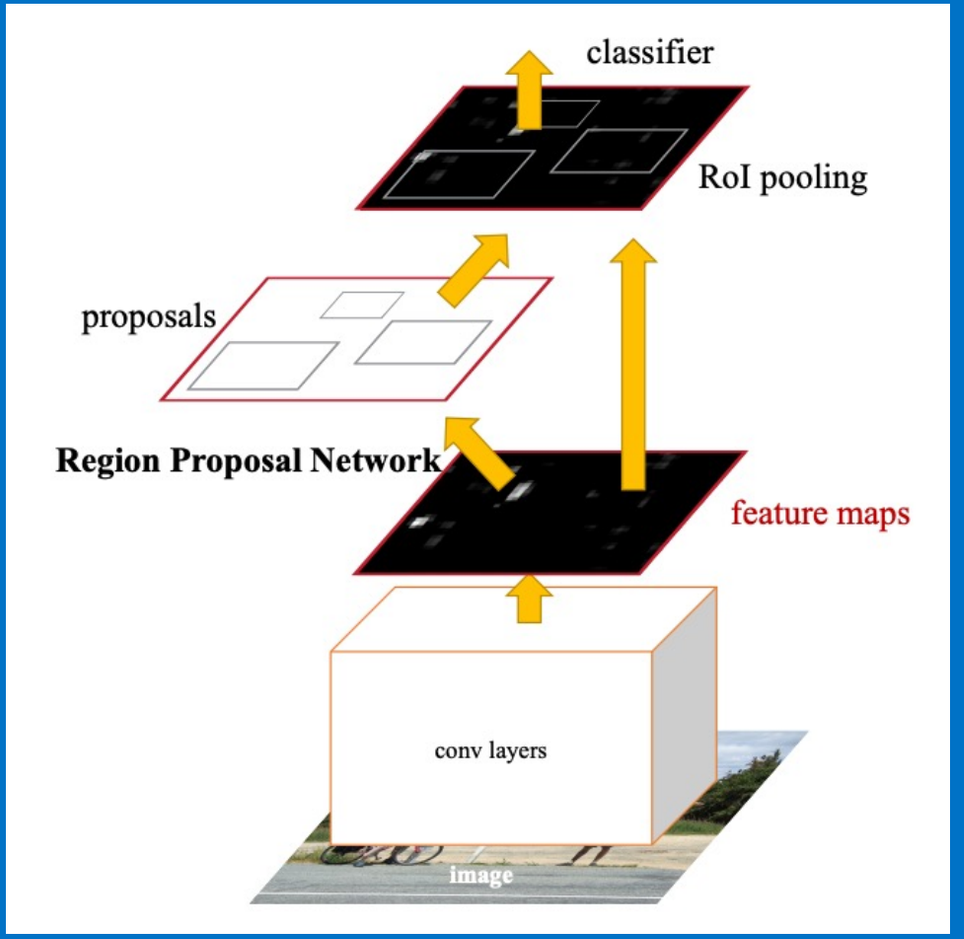
C_1 = *Dog class*
C2 = Person Class

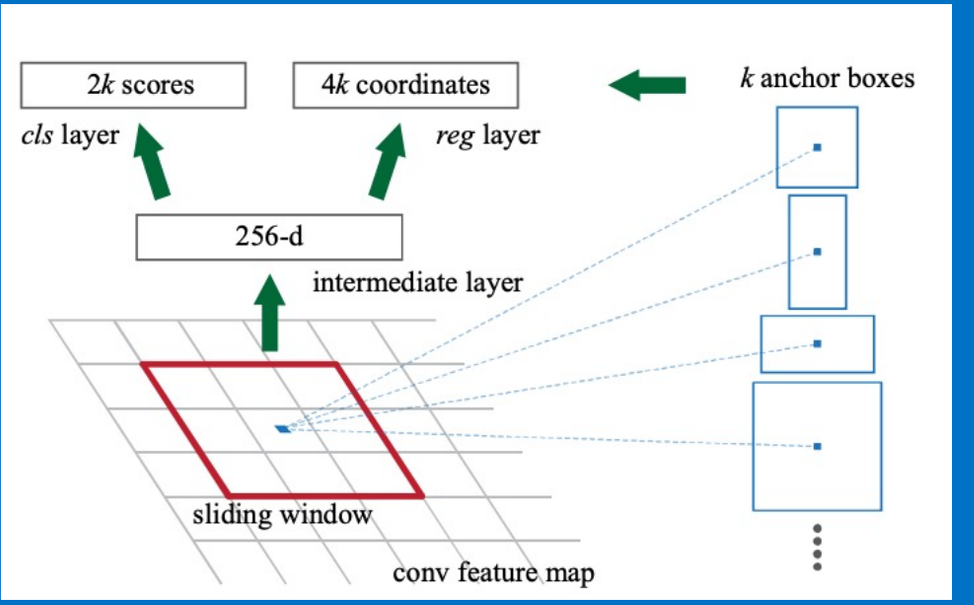
$$\begin{bmatrix} 1 \\ 30 \\ 28 \\ 28 \\ 82 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ - \\ - \\ - \\ - \\ - \\ - \end{bmatrix}$$


X_train
y_train
P_c
B_x
B_y
B_w
B_h
C_1
C_2
1
50
70
60
70
1
0
1
30
55
28
82
0
1
0
Convolutional Neural Network
P_c
B_x
B_y
B_w
B_h
C_1
C_2

Faster RCNN





https://www.youtube.com/watch?v=iHf2xHQ2VYo

Deep Learning Frameworks

- 1. TensorFlow
- 2. PyTorch
- 3. Keras
- 4. Sonnet
- 5. MXNet
- 6. Swift for TensorFlow
- 7. Gluon
- 8. DL4J
- 9. ONNX
- 10. Chainer

Deep Learning Hardware

- When it comes to hardware, we try to be as agnostic as possible.
- What that means is that our systems should be able to run on server grade machines with a number of GPU as well as small embedded devices without a GPU on board.
- For example, some of our customers prefer deployment on Amazon servers with V-100 GPU while others prefer having an edge solution like the Nvidia Tx2 or Nx boards. Some are even limited and want to run on say a NXP board which might not have any GPU on board

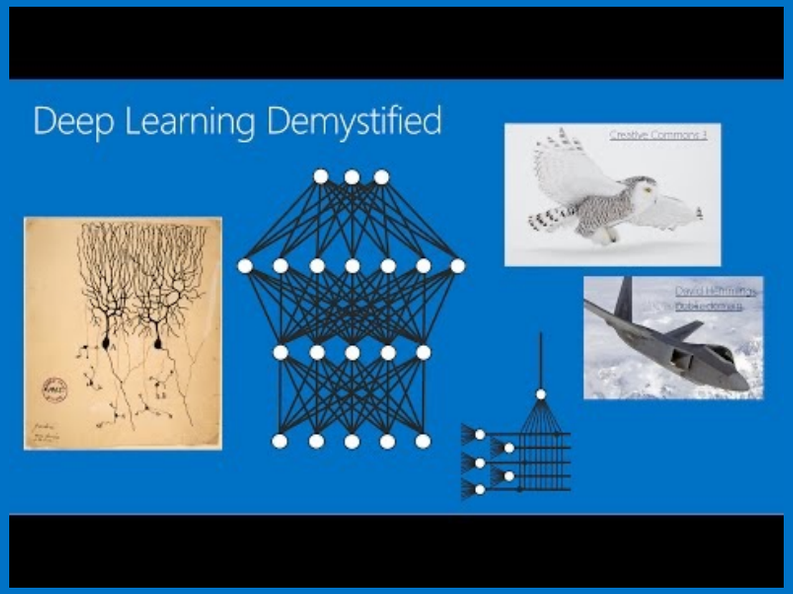
- To cater to all these different environments, we have to look at optimizations that are both hardware specific as well as deep learning model related

Deep Learning Hardware

- For example, when it comes to Nvidia hardware, Nvidia have developed TensorRT libs that provide acceleration when using deep learning technology on Nvidia devices.
- Another example is the OpenVino libs developed by intel that help accelerate deep learning inference on Intel CPU.
- Apple has Metal/Swift/CoreML acceleration that help improve performance of deep learning in Apple devices.
- Qualcomm has similar hardware accelerations available (AIMET) .
- On the model side, we try to make concerted effort in developing different class of models, from shallow to super deep. The shallow models obviously for smaller edge hardware while deeper ones for server grade machines. There is obviously always a trade off between accuracy and latency that you need to manage and be aware of.

Also check out

Deep Learning Demystified



Notes from Stanford CS 231 course
(Justin Johnson and Andrej Karpathy)
The writings of Christopher Olah

Slides Credit

Connect with me online:

Brandon Rohrer on LinkedIn

@_brohrer_ on Twitter

#HowItWorks #ConvNet

brohrer@microsoft.com

Dr. Mohsen Ali, ITU's Deep Learning Course