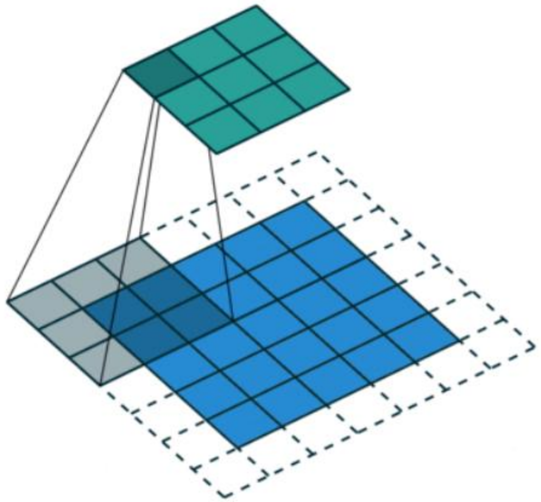
Image Classification with CNN



Outline

- What is CNN
- Why CNNs Matter
- How CNNs Work
- Classical CNNs

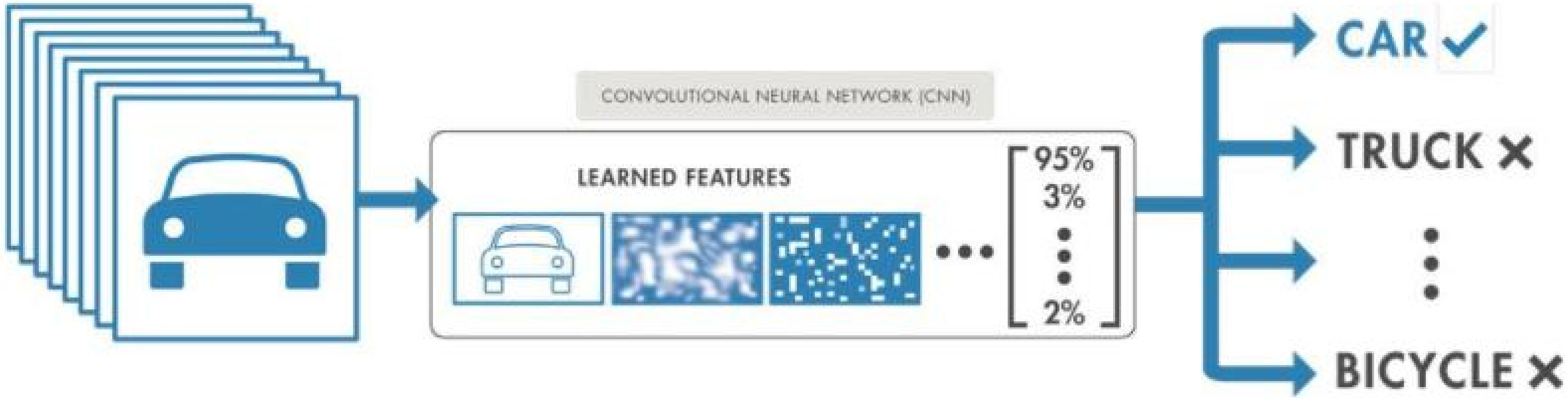
CNN

- A convolutional neural network (CNN or ConvNet), is a network architecture for deep learning which learns directly from data, eliminating the need for manual feature extraction.
- CNNs are particularly useful for finding patterns in images to recognize objects, faces, and scenes. They can also be quite effective for classifying non-image data such as audio, time series, and signal data.
- Applications that call for object recognition and computer vision — such as self-driving vehicles and face-recognition applications — rely heavily on CNNs.

Why CNNs Matter

Using CNNs for deep learning is popular due to three important factors:

- CNNs eliminate the need for manual feature extraction—the features are learned directly by the CNN.
- CNNs produce highly accurate recognition results.
- CNNs can be retrained for new recognition tasks, enabling you to build on pre-existing networks.



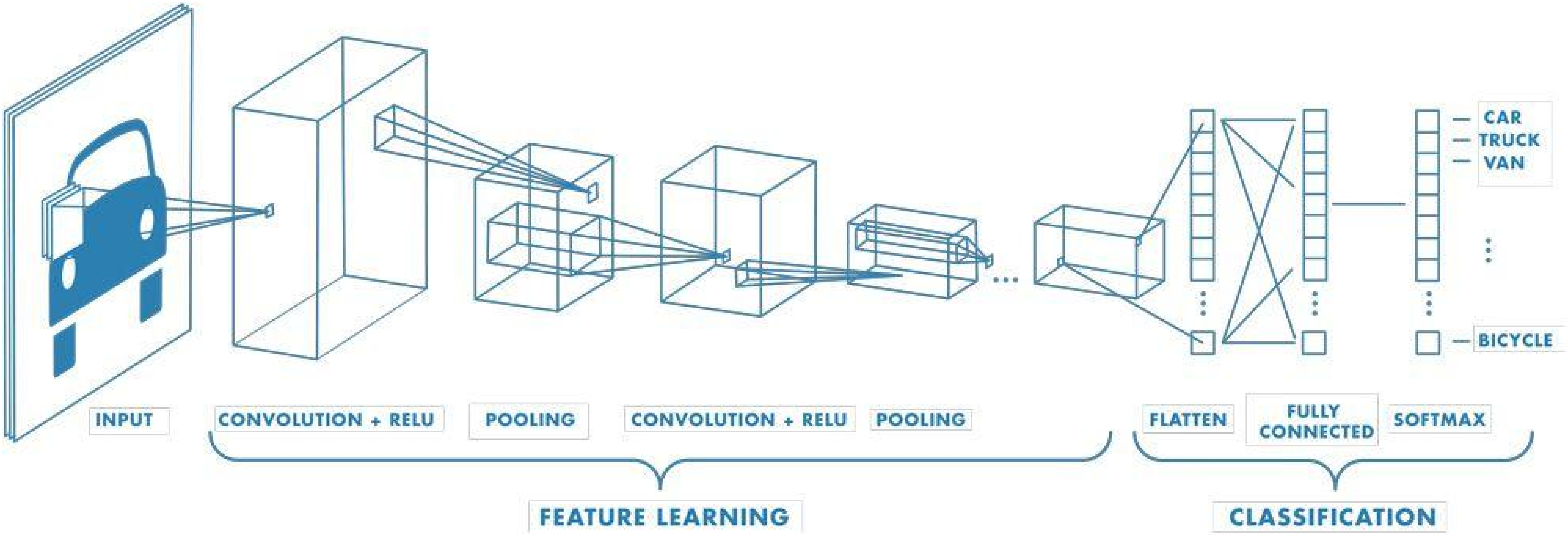
How CNNs Work

A convolutional neural network can have tens or hundreds of layers that each learn to detect different features of an image. Three of the most common layers are: convolution, activation or ReLU, and pooling.

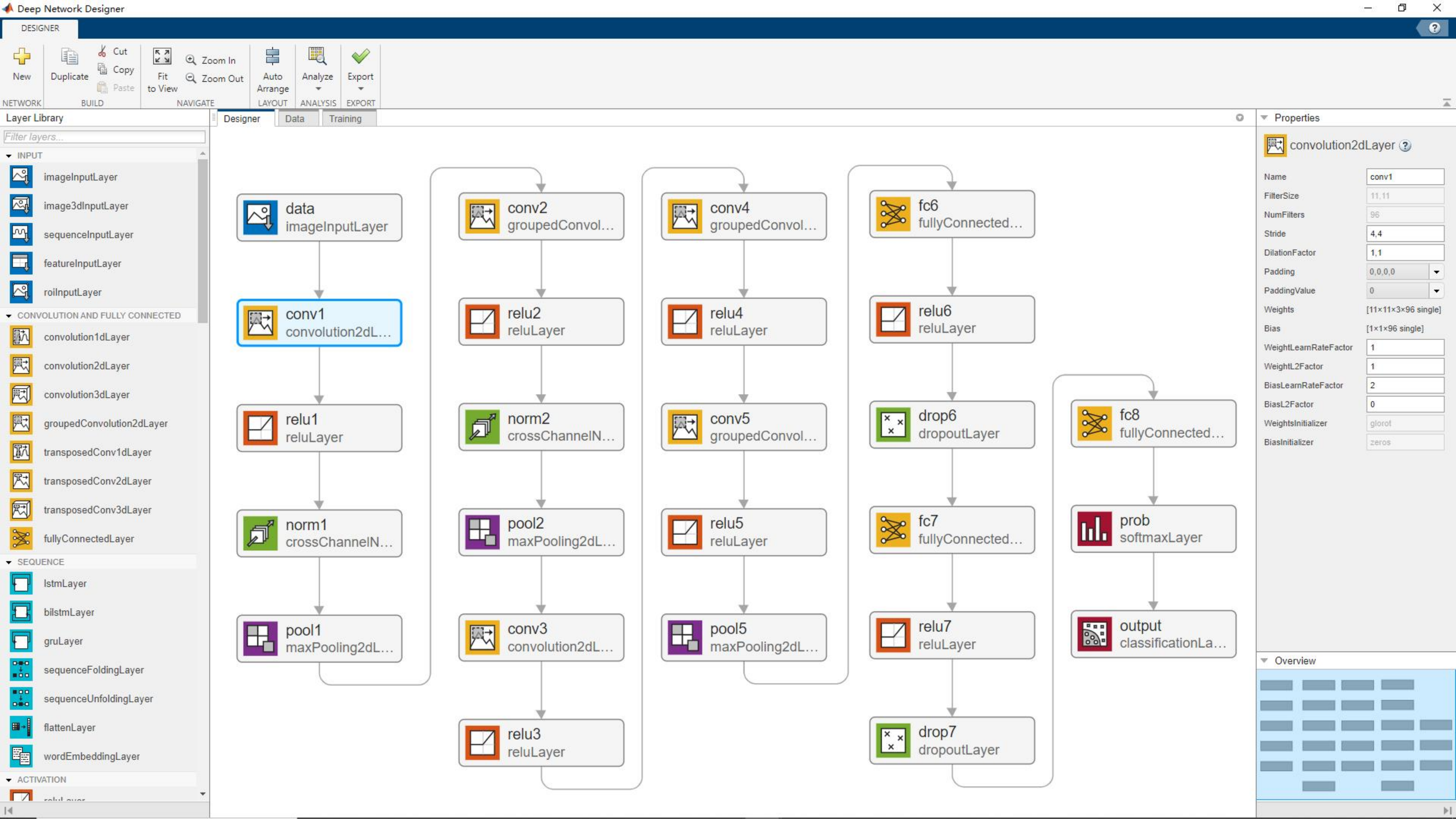
- **Convolution** puts the input images through a set of convolutional filters, each of which activates certain features from the images.
- **Rectified linear unit (ReLU)** allows for faster and more effective training by mapping negative values to zero and maintaining positive values. This is sometimes referred to as activation, because only the activated features are carried forward into the next layer.
- **Pooling** simplifies the output by performing nonlinear downsampling, reducing the number of parameters that the network needs to learn.

How CNNs Work

These operations are repeated over tens or hundreds of layers, with each layer learning to identify different features. Filters are applied to each training image at different resolutions, and the output of each convolved image is used as the input to the next layer.




Deep Network Designer Start Page
MATLAB Deep Network Designer
Getting Started | Compare Pretrained Networks | Transfer Learning
General
Blank Network
From Workspace
Image Networks (Pretrained)
SqueezeNet
GoogLeNet
ResNet-50
EfficientNet-b0
DarkNet-53
DarkNet-19
ShuffleNet
NasNet-Mobile
NasNet-Large
Xception
Places365-Goog...
Show more
Sequence Networks
Sequence-to-Label
Sequence-to-Sequ...
Audio Networks (Pretrained)

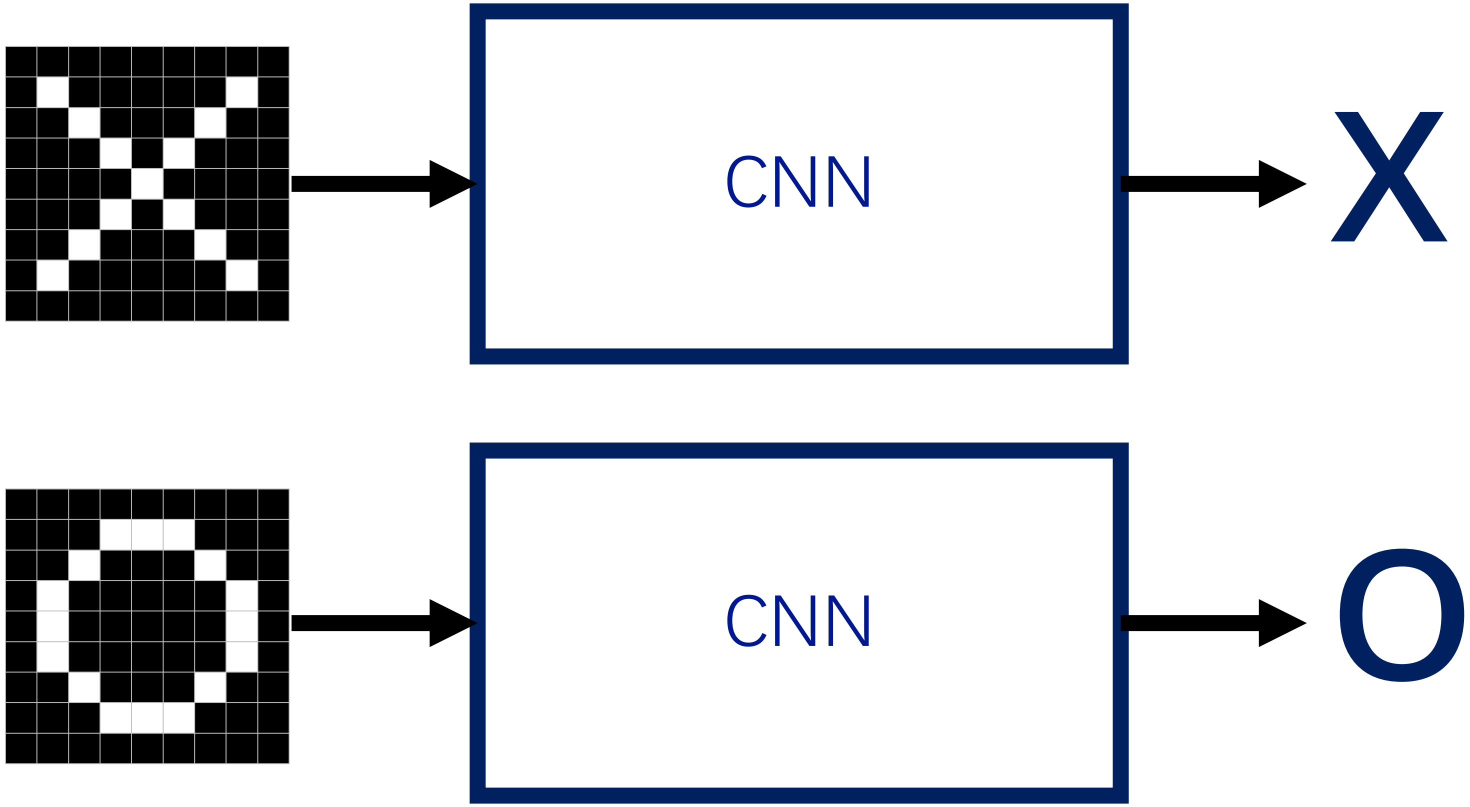

Deep Network Designer
DESIGNER
New
Duplicate
Cut
Copy
Paste
Fit to View
Zoom In
Zoom Out
Auto Arrange
Analyze
Export
NETWORK
BUILD
NAVIGATE
LAYOUT
ANALYSIS
EXPORT
Layer Library
Filter layers...
INPUT
imageInputLayer
image3dInputLayer
sequenceInputLayer
featureInputLayer
roiInputLayer
CONVOLUTION AND FULLY CONNECTED
convolution1dLayer
convolution2dLayer
convolution3dLayer
groupedConvolution2dLayer
transposedConv1dLayer
transposedConv2dLayer
transposedConv3dLayer
fullyConnectedLayer
SEQUENCE
lstmLayer
bilstmLayer
gruLayer
sequenceFoldingLayer
sequenceUnfoldingLayer
flattenLayer
wordEmbeddingLayer
ACTIVATION
Designer
Data
Training
data imageInputLayer
conv1 convolution2dL...
relu1 reluLayer
norm1 crossChannelN...
pool1 maxPooling2dL...
conv2 groupedConvol...
relu2 reluLayer
norm2 crossChannelN...
pool2 maxPooling2dL...
conv3 convolution2dL...
relu3 reluLayer
conv4 groupedConvol...
relu4 reluLayer
conv5 groupedConvol...
relu5 reluLayer
pool5 maxPooling2dL...
fc6 fullyConnected...
relu6 reluLayer
drop6 dropoutLayer
fc7 fullyConnected...
relu7 reluLayer
drop7 dropoutLayer
fc8 fullyConnected...
prob softmaxLayer
output classificationLa...
Properties
convolution2dLayer
Name conv1
FilterSize 11,11
NumFilters 96
Stride 4,4
DilationFactor 1,1
Padding 0,0,0,0
PaddingValue 0
Weights [11×11×3×96 single]
Bias [1×1×96 single]
WeightLearnRateFactor 1
WeightL2Factor 1
BiasLearnRateFactor 2
BiasL2Factor 0
WeightsInitializer glorot
BiasInitializer zeros
Overview

A toy ConvNet: X's and O's

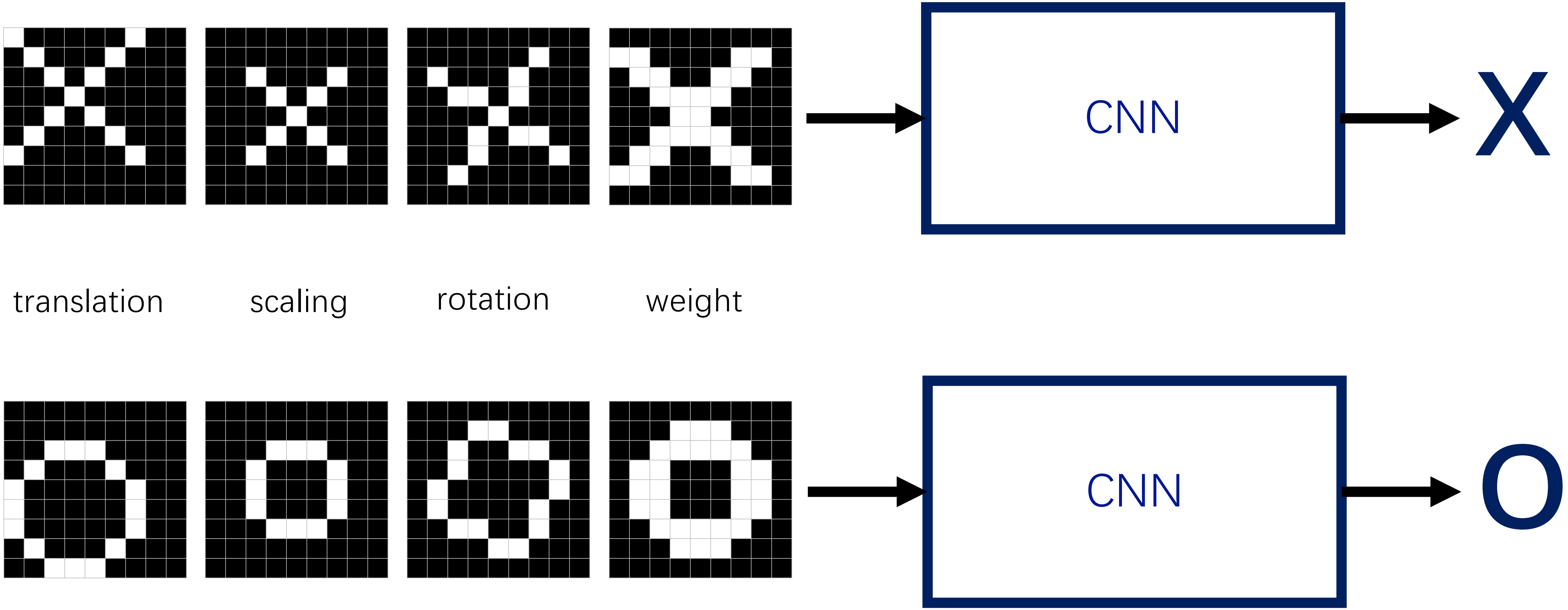
Says whether a picture is of an X or an O

A two-dimensional
array of pixels

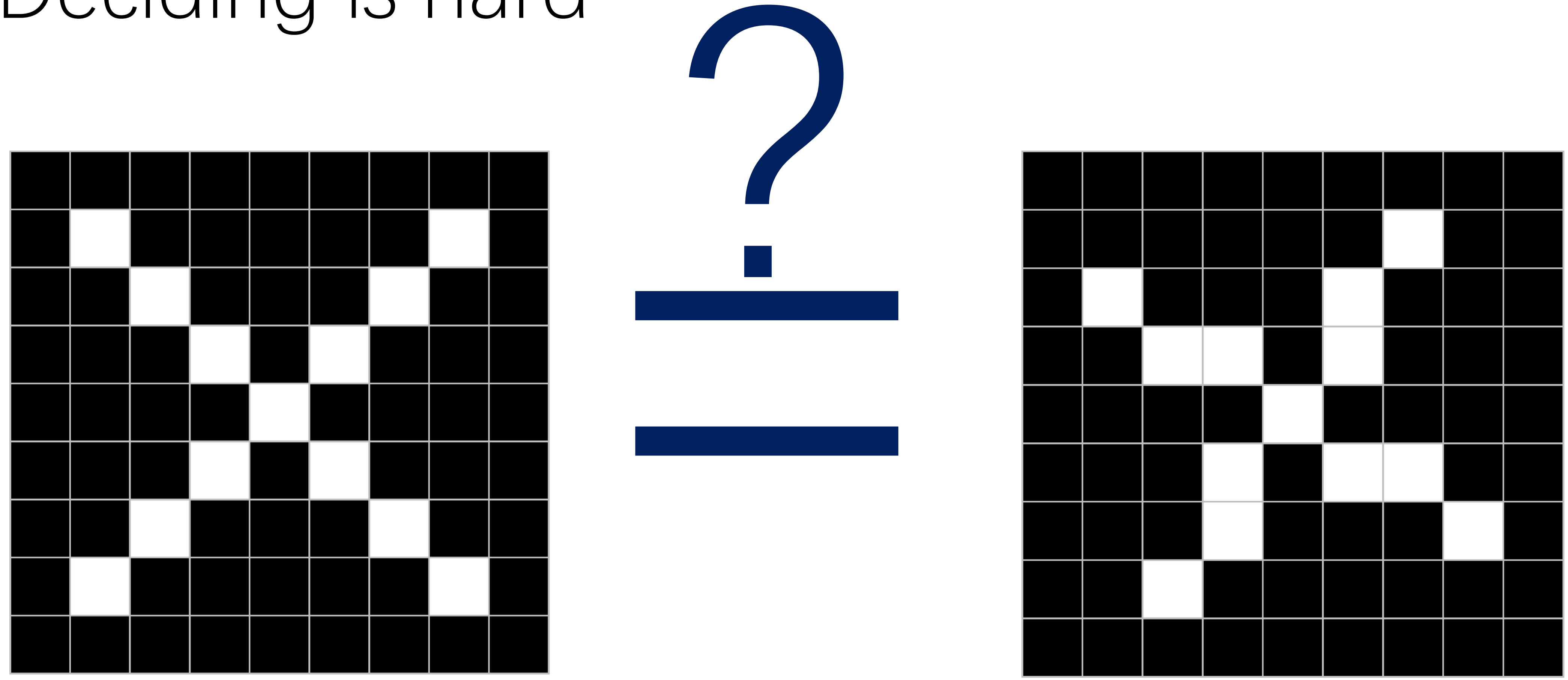
For example



Trickier cases



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

$$\stackrel{?}{=}$$

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

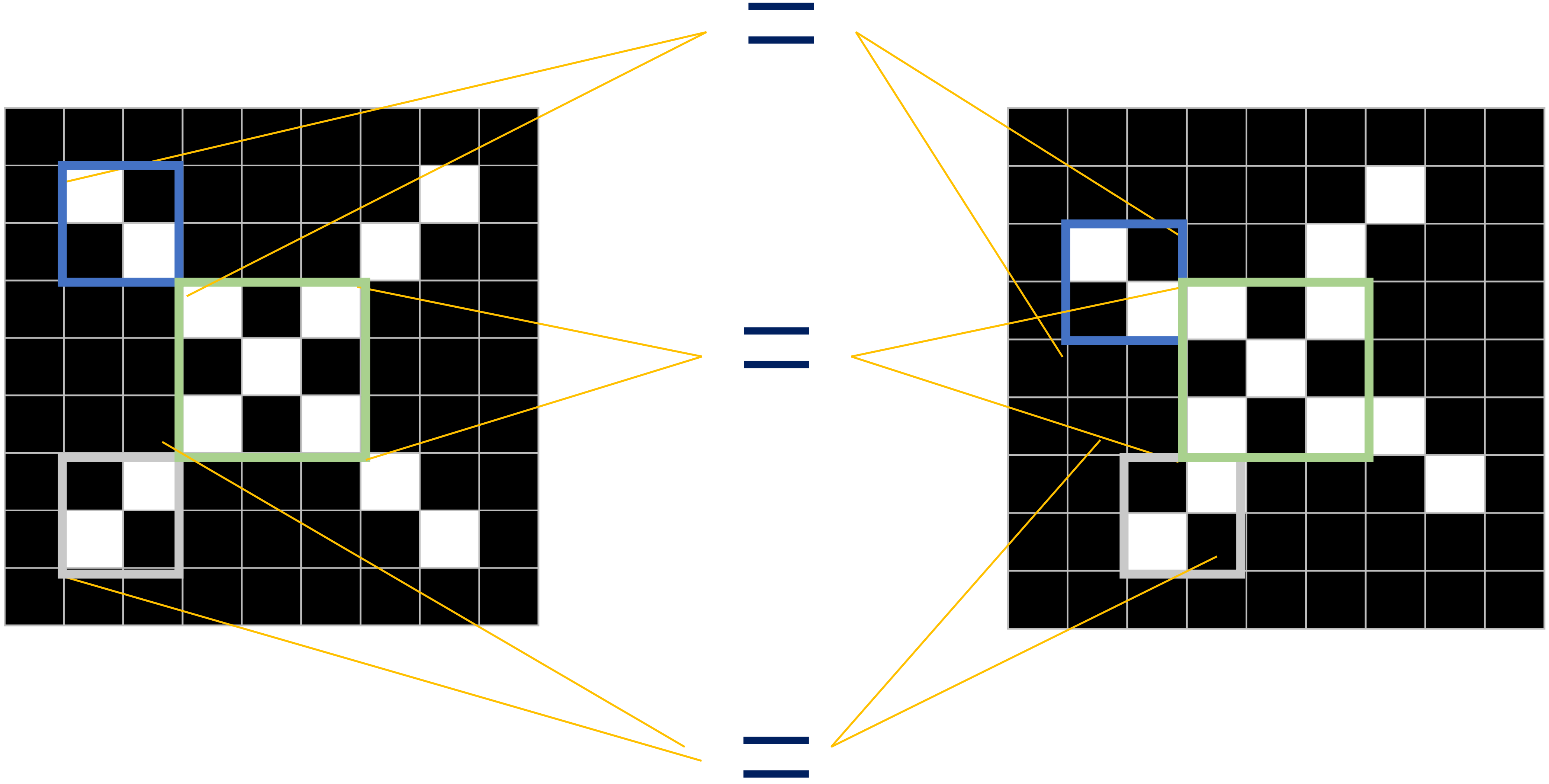
Computers are literal

-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

$\neq$

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

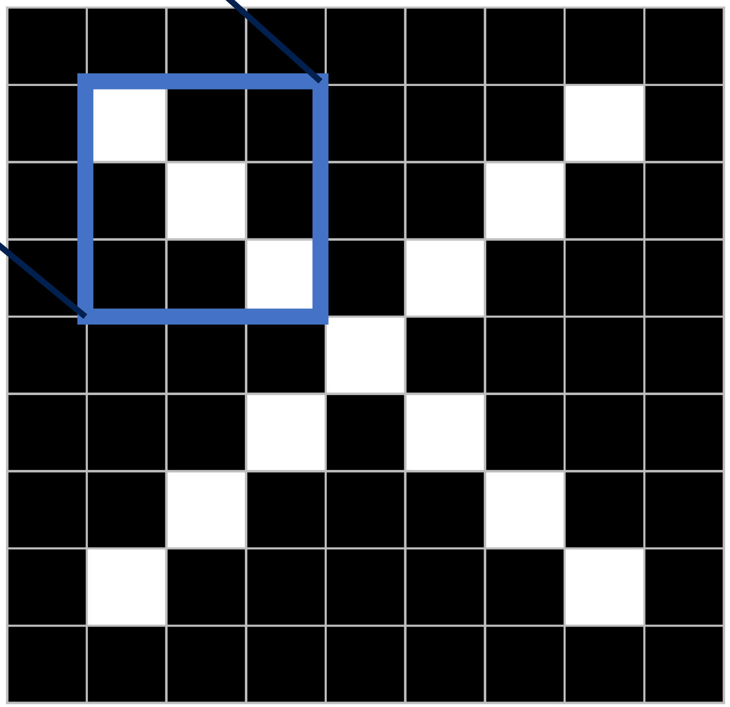
1	-1	1
-1	1	-1
1	-1	1

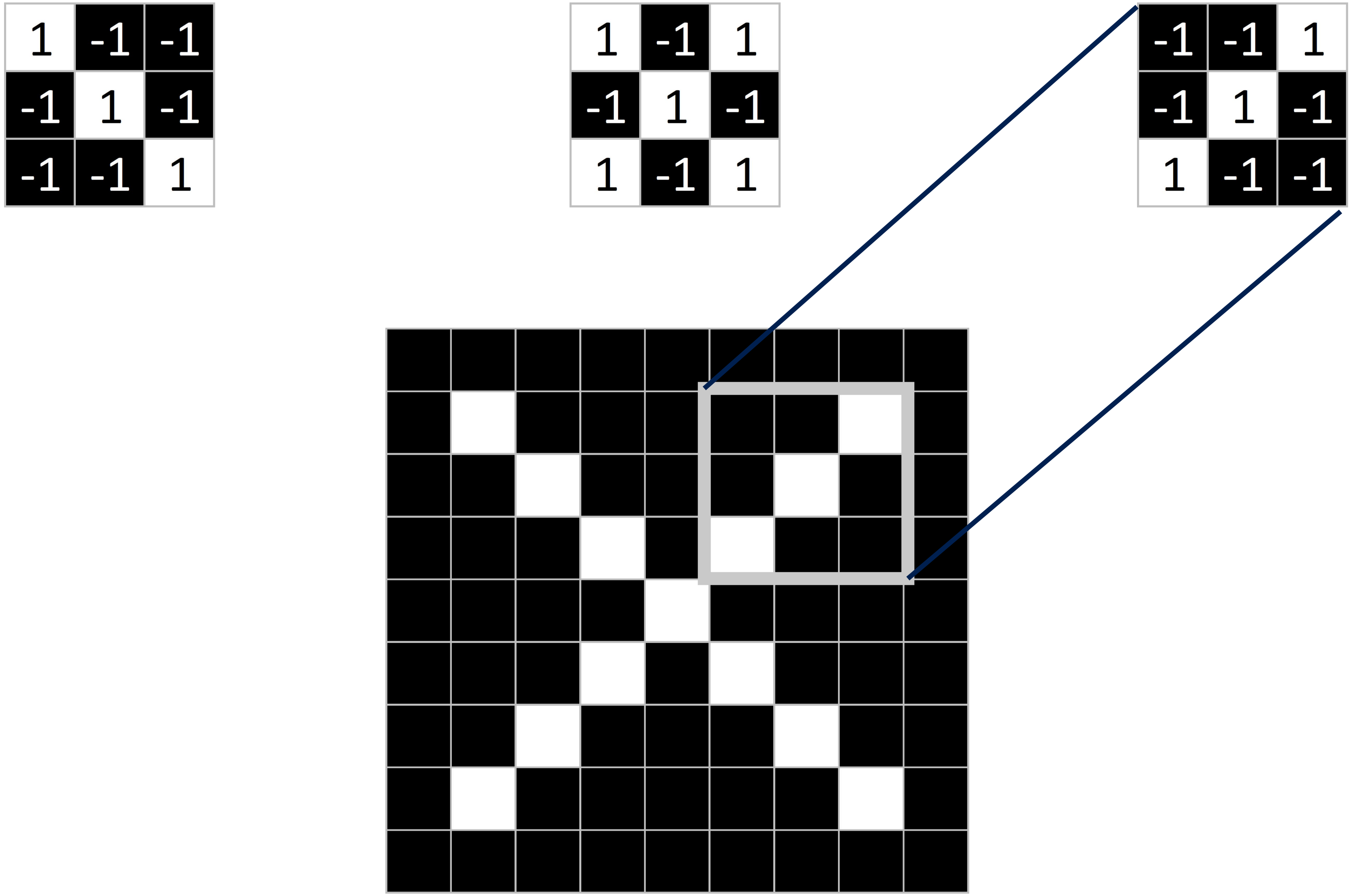
-1	-1	1
-1	1	-1
1	-1	-1

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

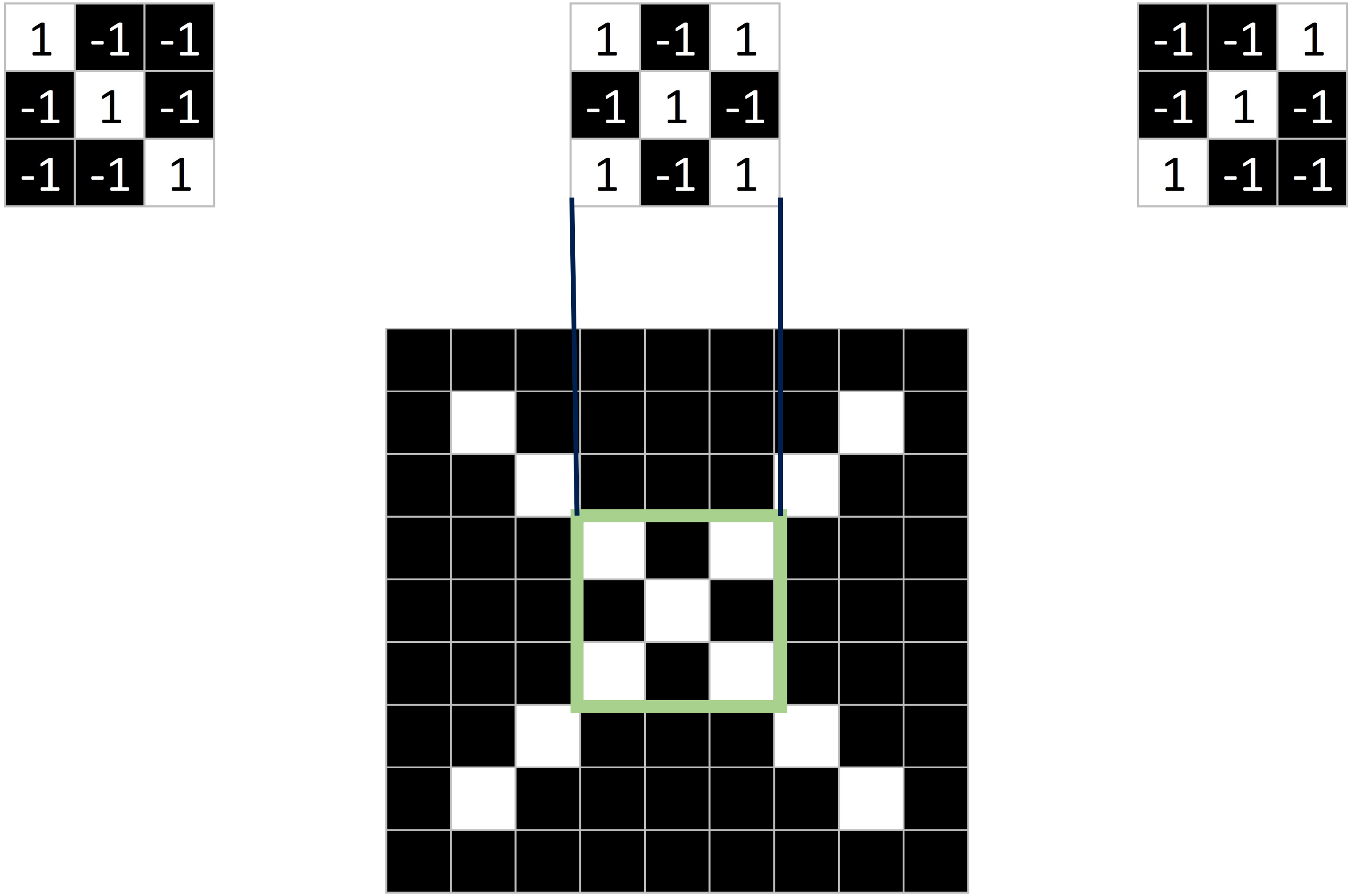
1	-1	1
-1	1	-1
1	-1	1

-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

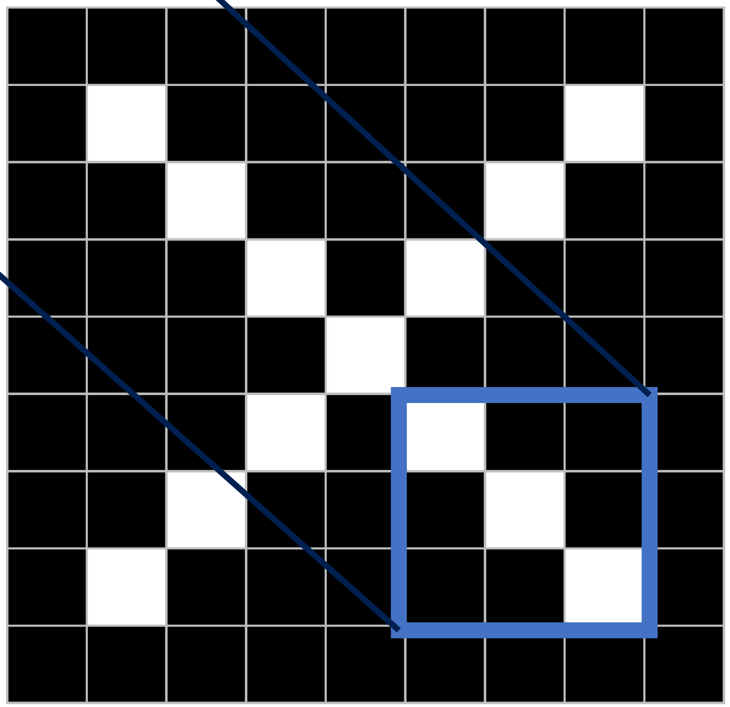


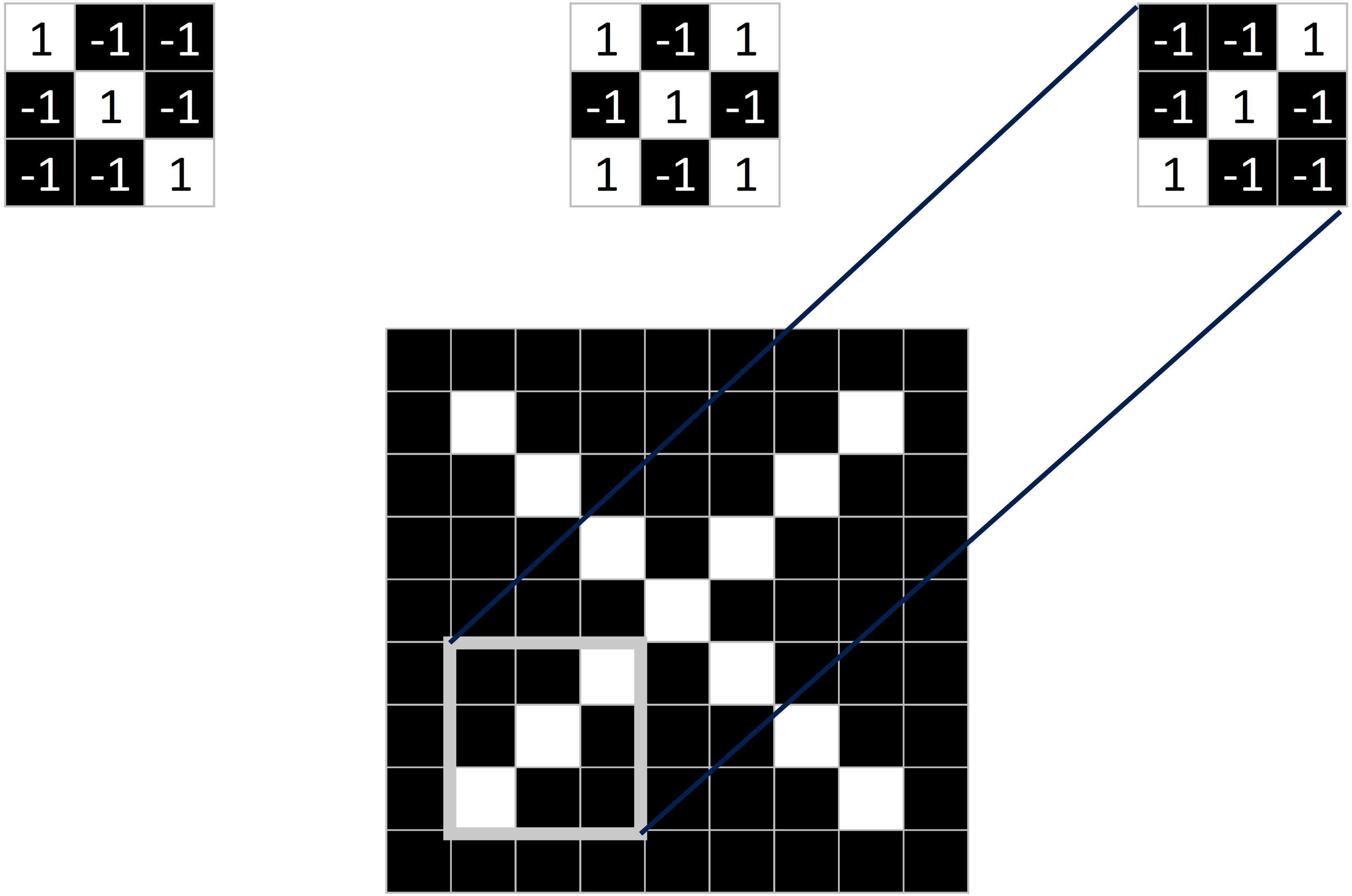
1 -1 -1
-1 1 -1
-1 -1 1
1 -1 1
-1 1 -1
1 -1 1
-1 -1 1
-1 1 -1
1 -1 -1

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




1 -1 -1
-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
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-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. 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.

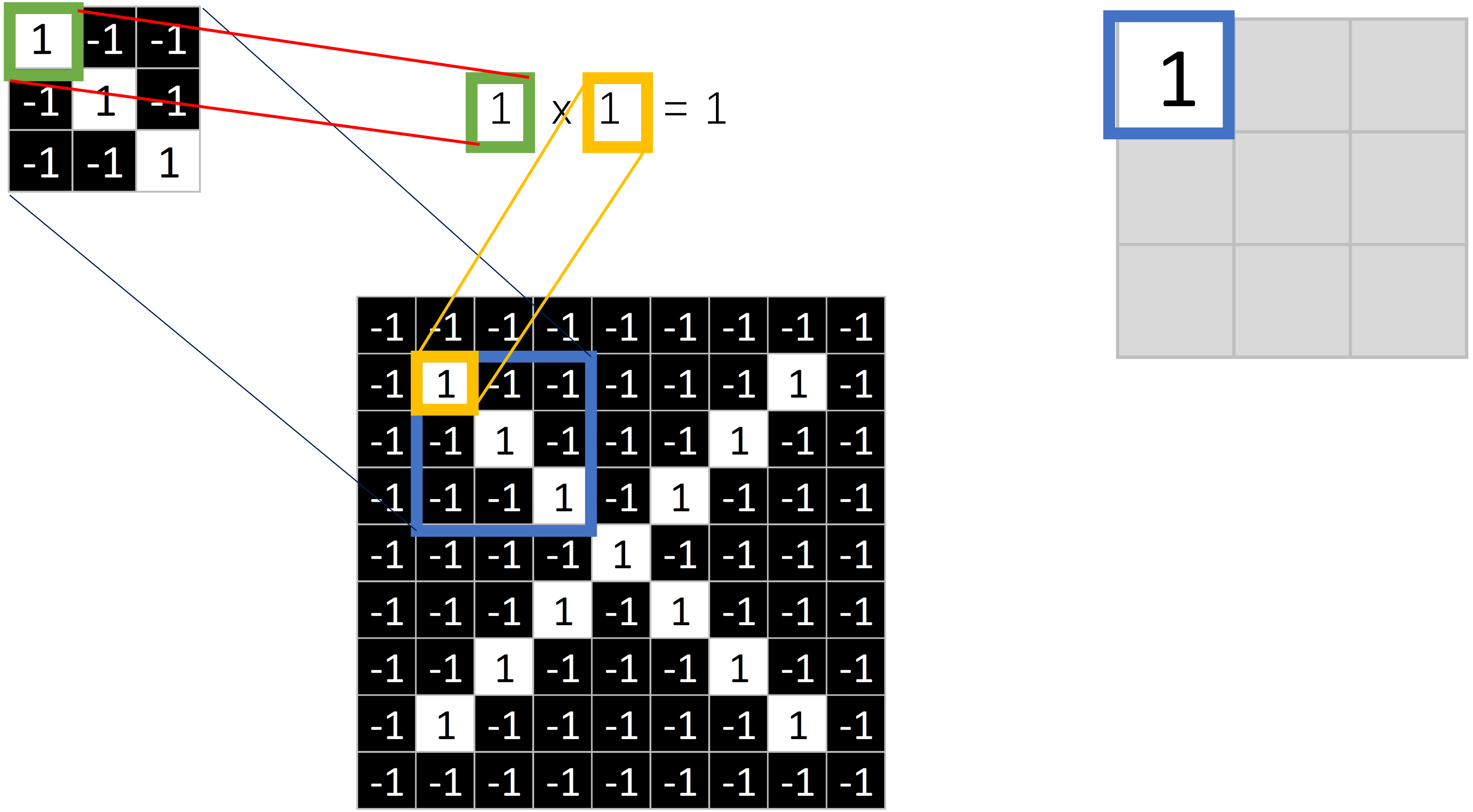
Filtering: The math behind the match

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

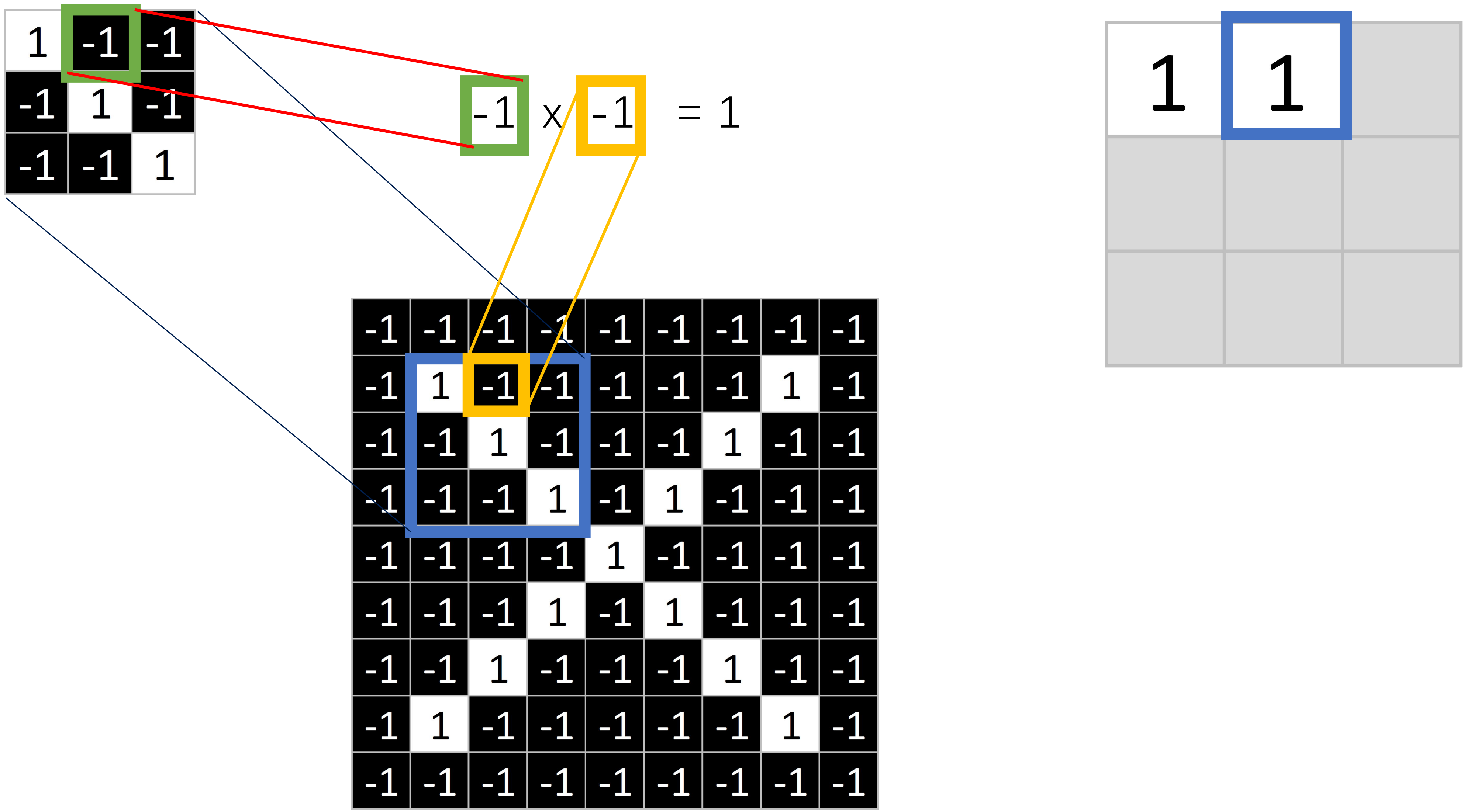
$\boxed{1} \times \boxed{1} = 1$

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

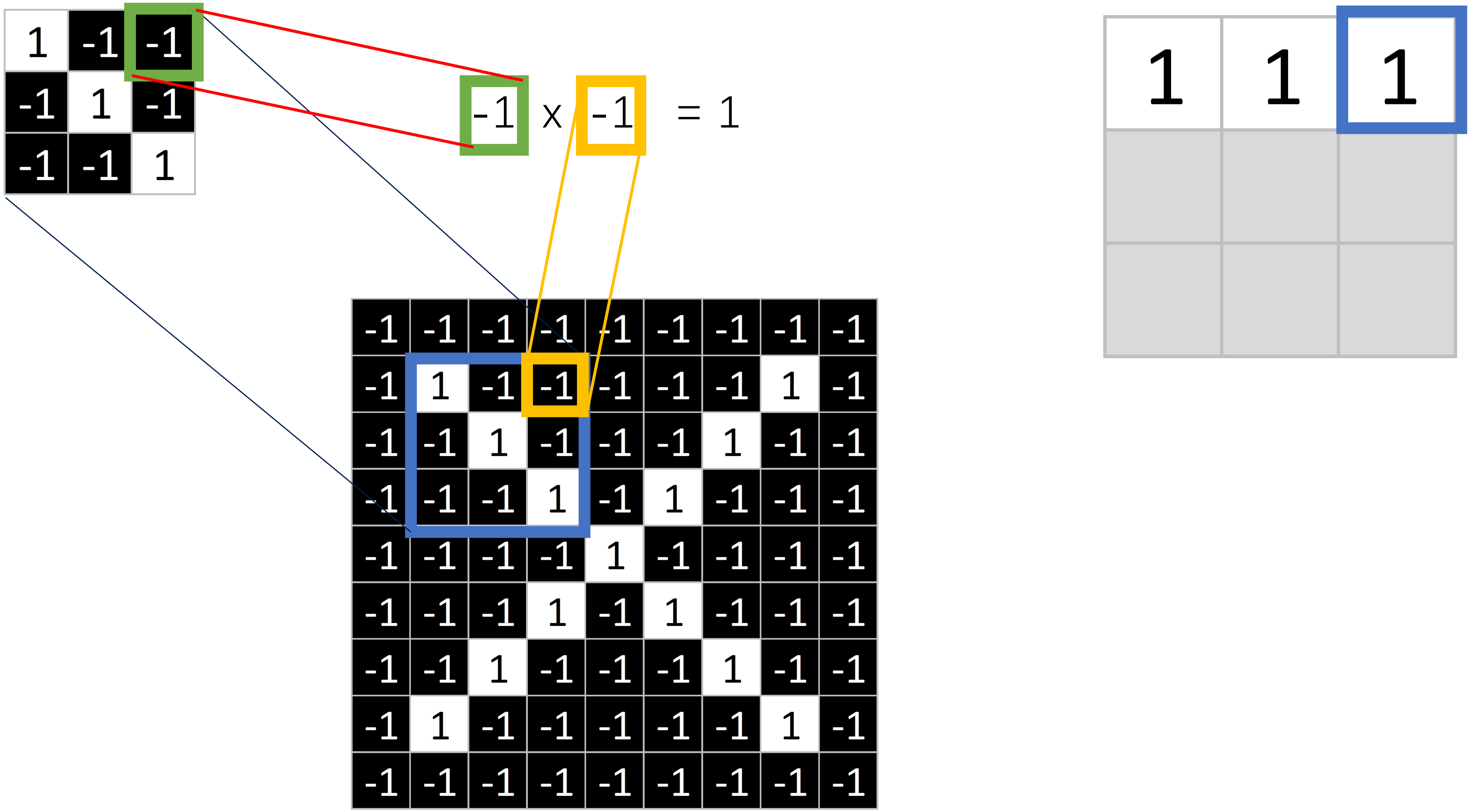
Filtering: The math behind the match



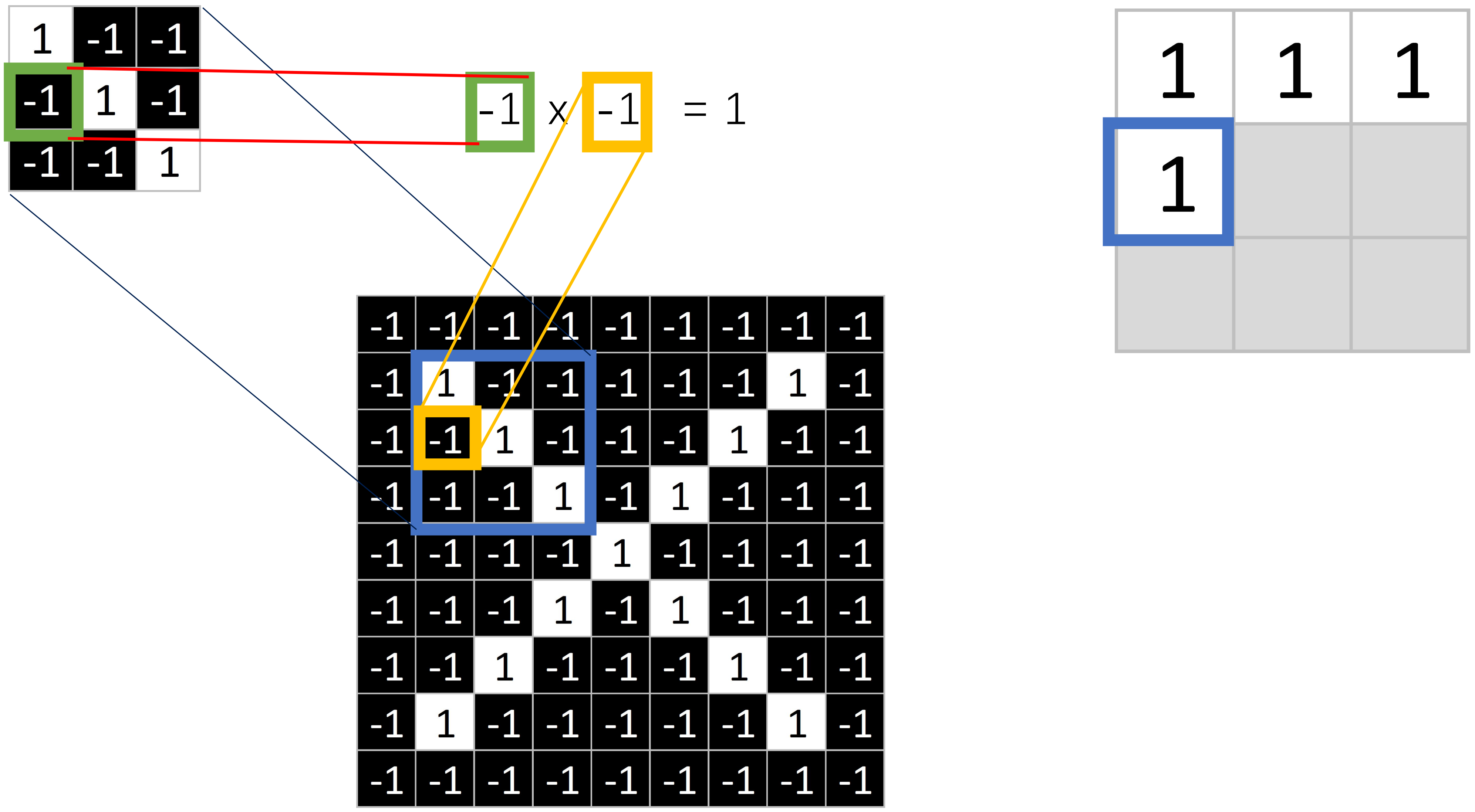
Filtering: The math behind the match



Filtering: The math behind the match



Filtering: The math behind the match



Filtering: The math behind the match

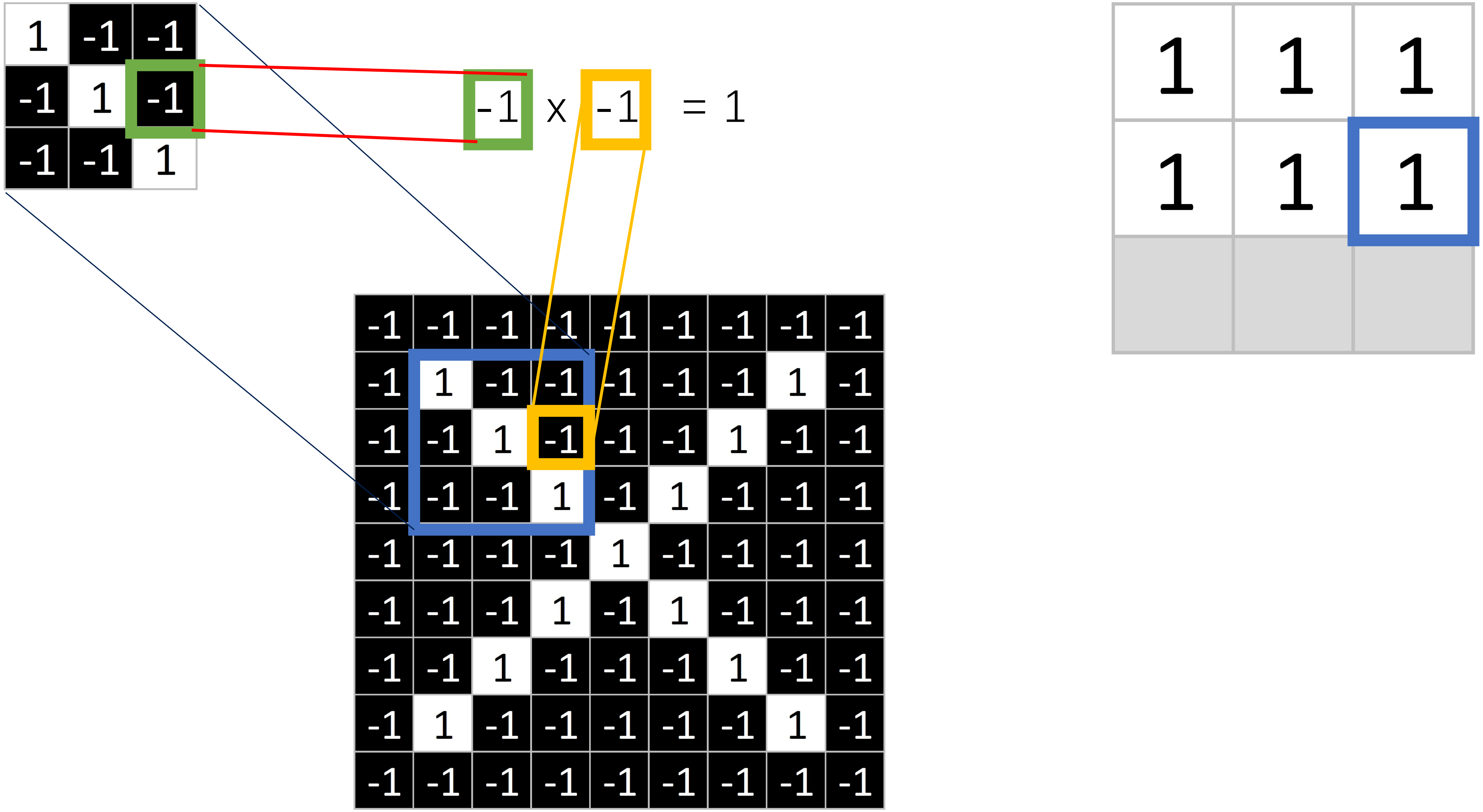
1	-1	-1
-1	1	-1
-1	-1	1

1 x 1 = 1

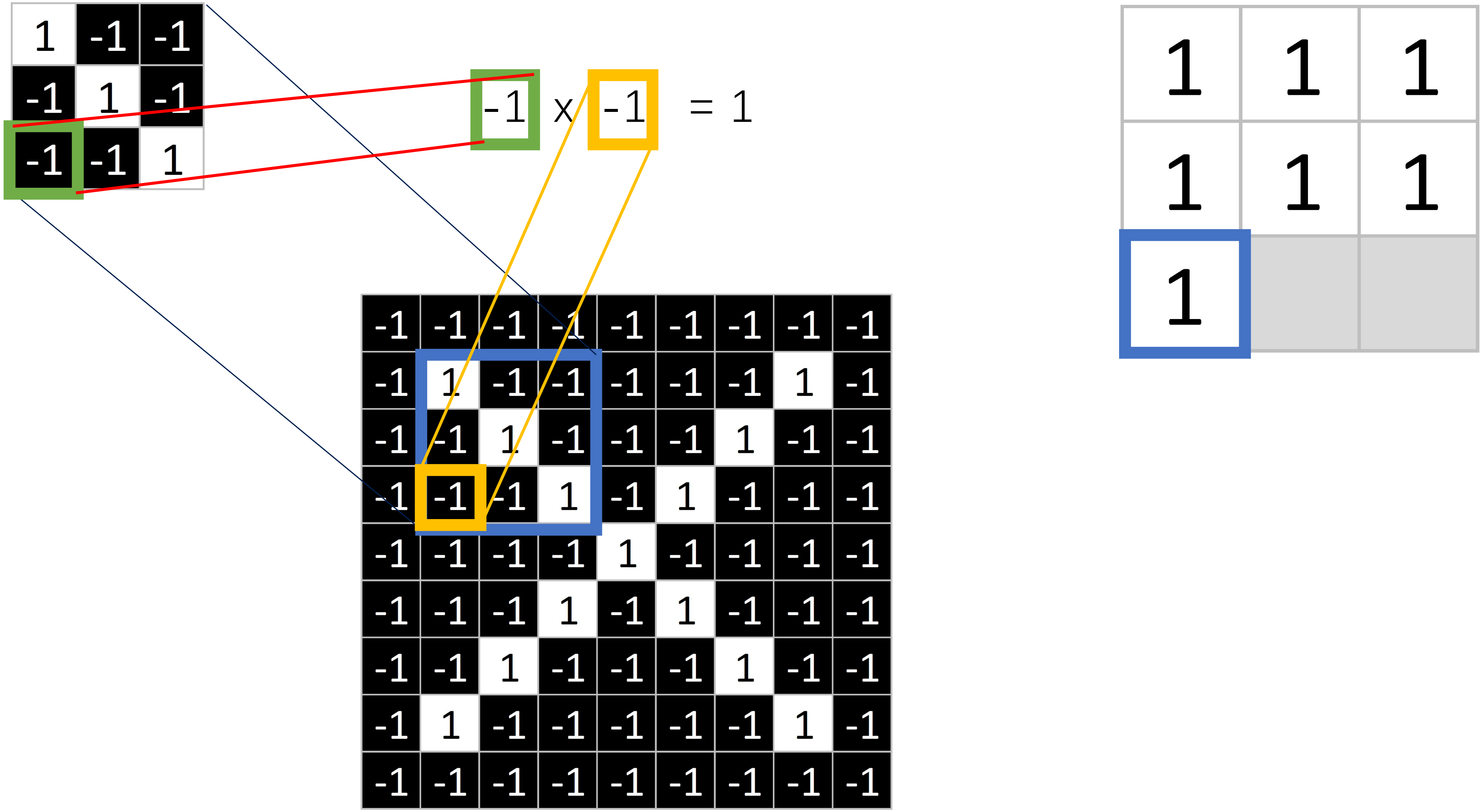
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

1	1	1
1	1	

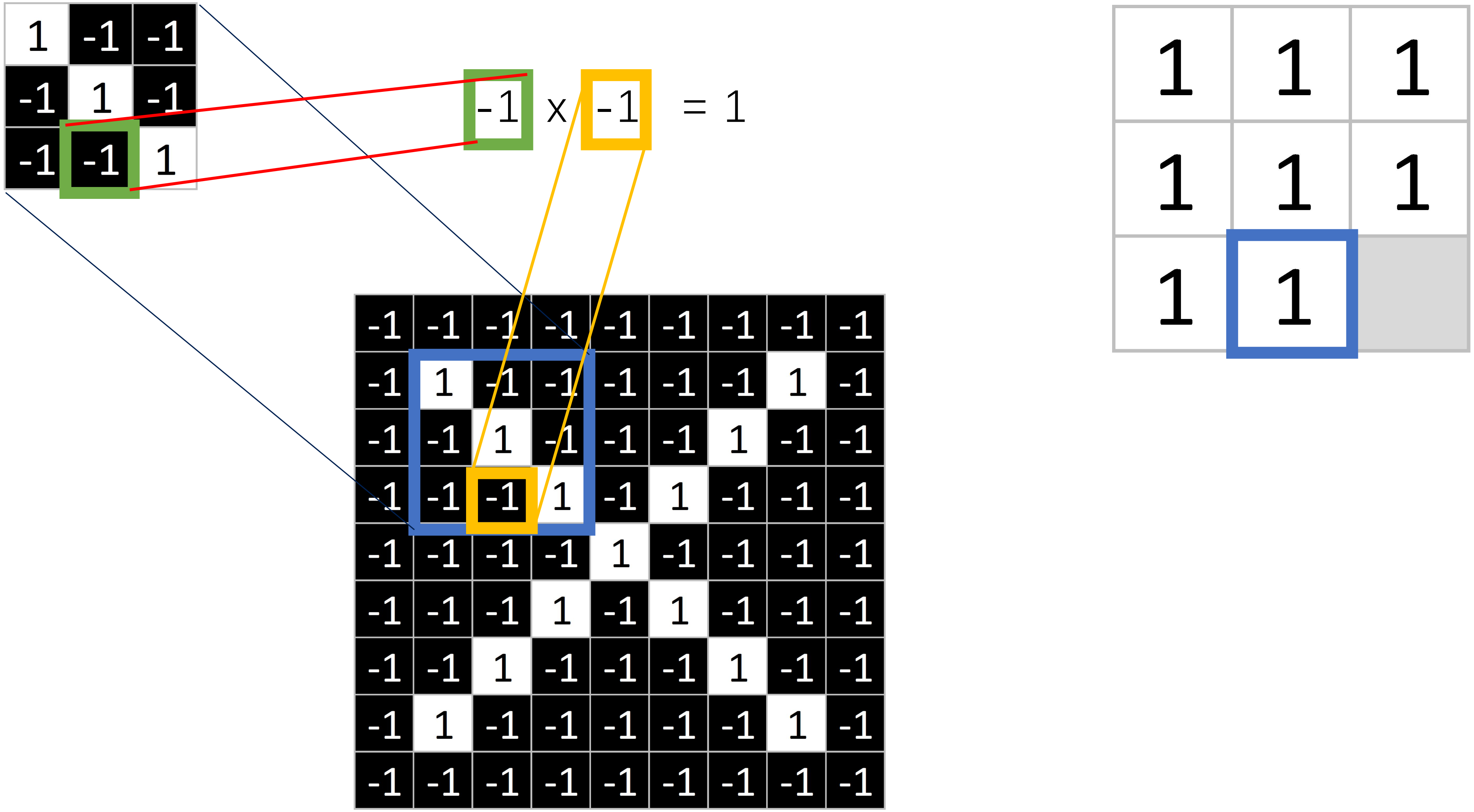
Filtering: The math behind the match



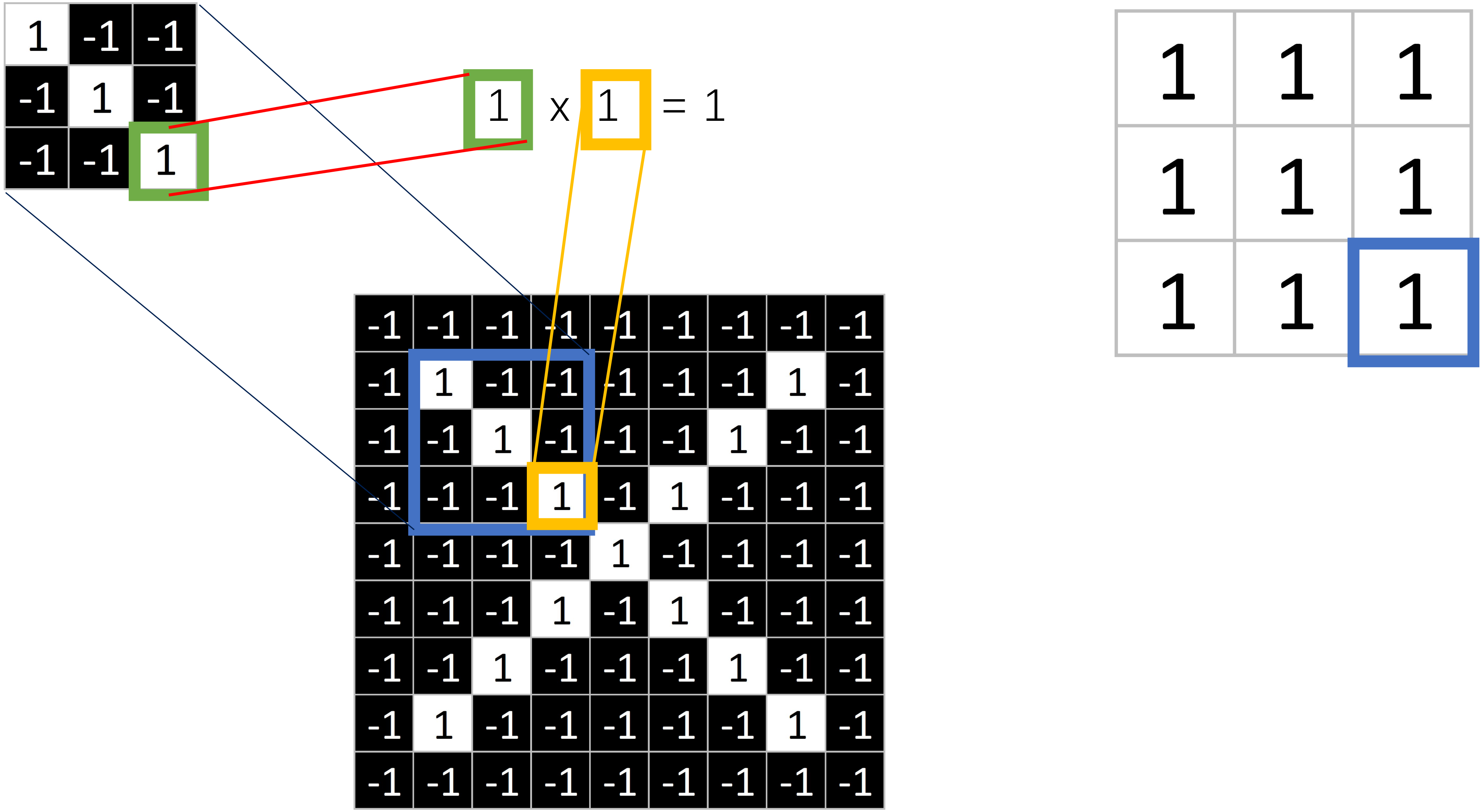
Filtering: The math behind the match



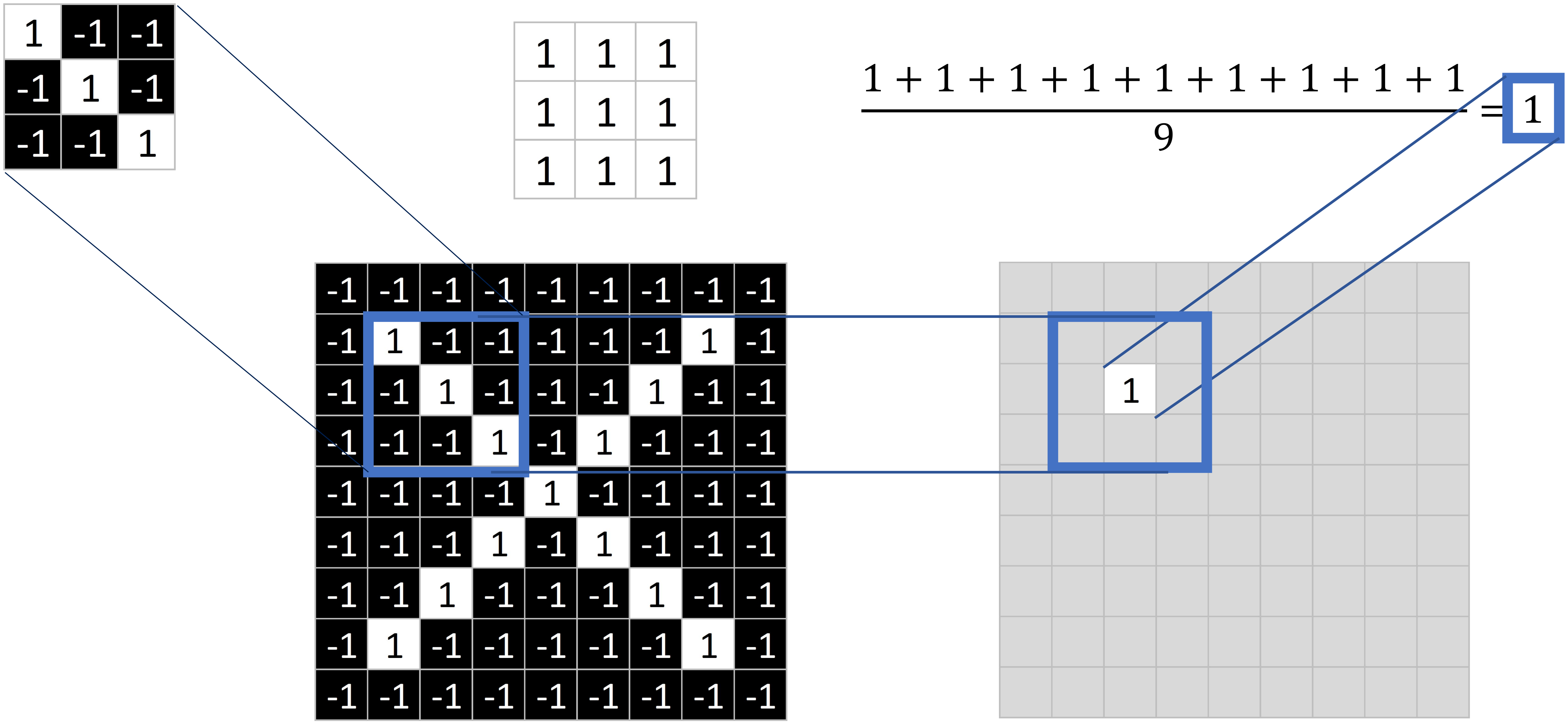
Filtering: The math behind the match



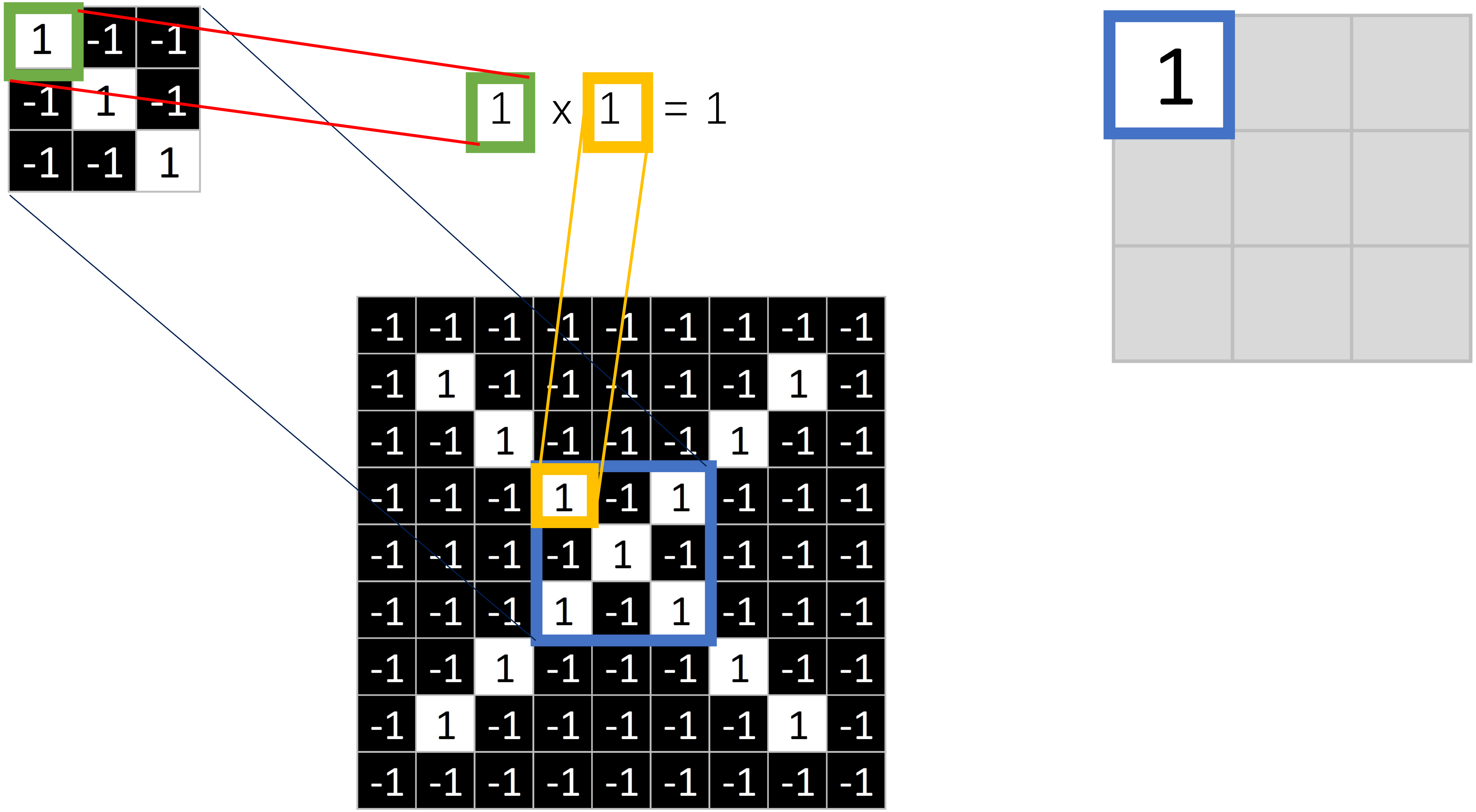
Filtering: The math behind the match



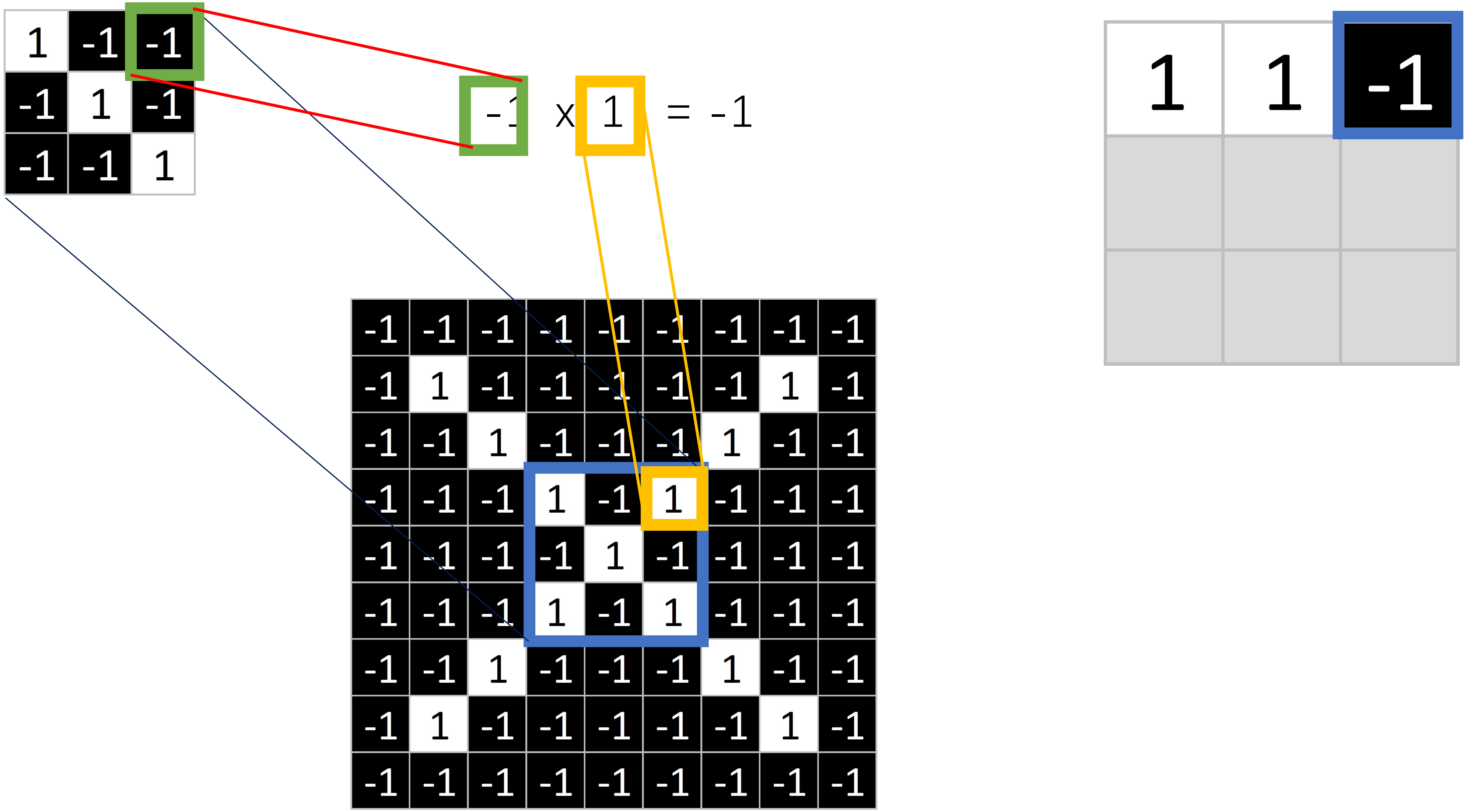
Filtering: The math behind the match



Filtering: The math behind the match



Filtering: The math behind the match



Filtering: The math behind the match

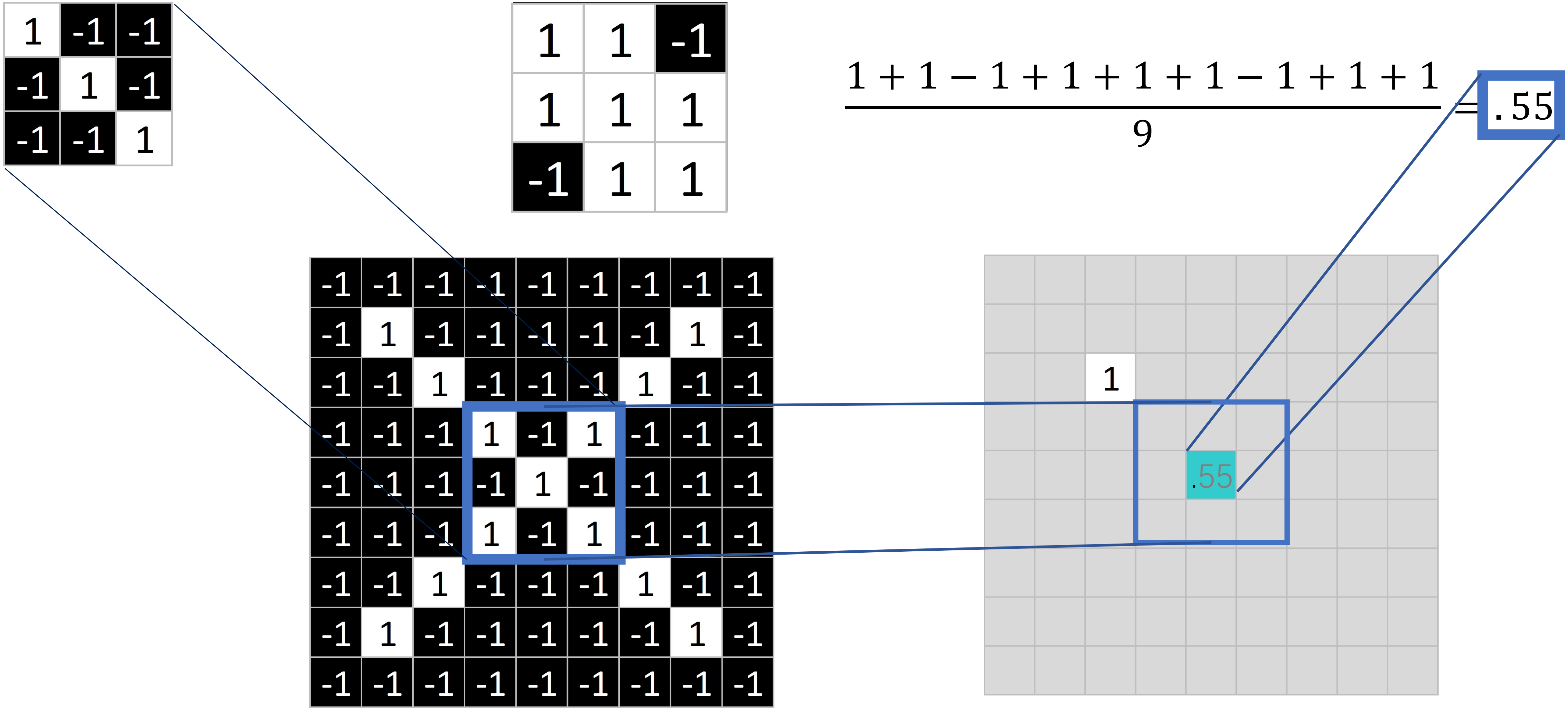
1	-1	-1
-1	1	-1
-1	-1	1

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

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



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

$\otimes$

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

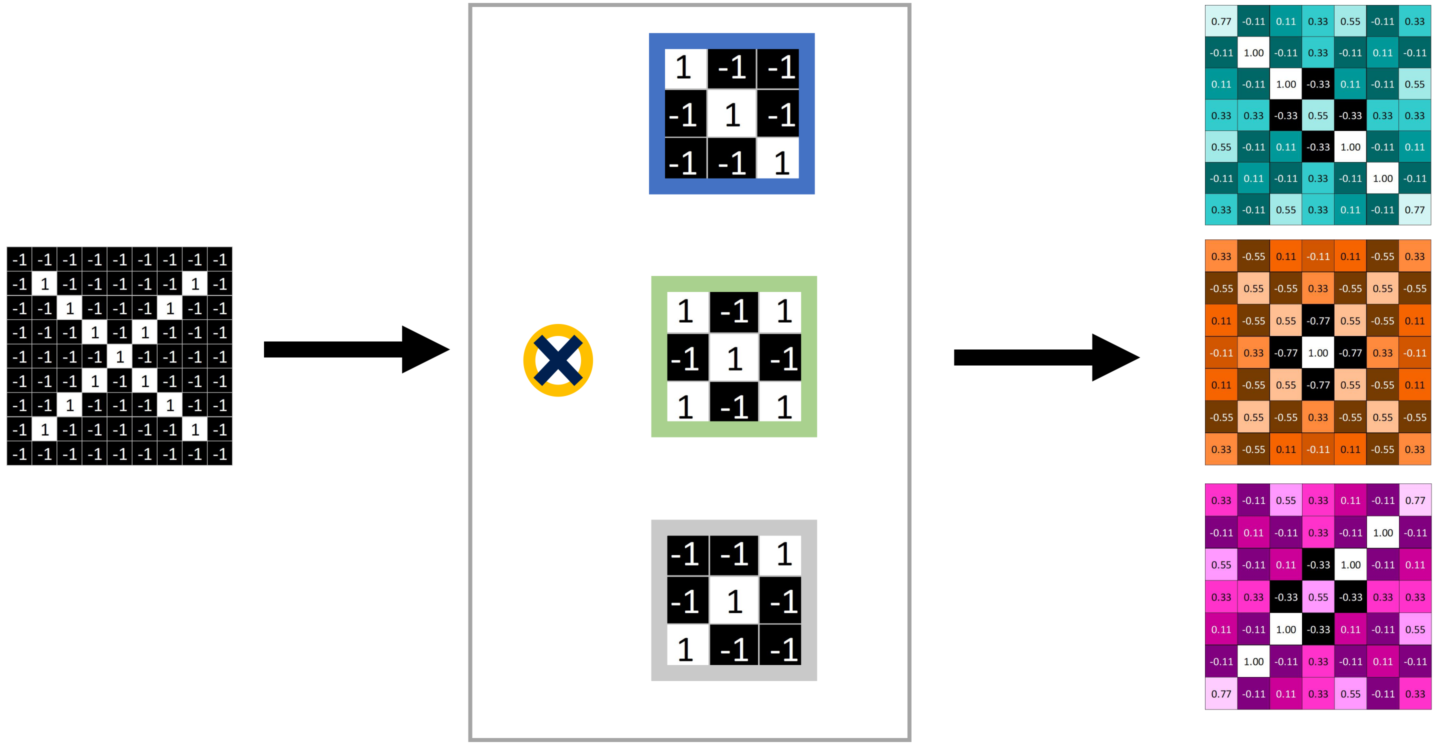
$$\begin{bmatrix} -1&-1&-1&-1&-1&-1&-1&-1&-1\\ -1&1&-1&-1&-1&-1&-1&1&-1\\ -1&-1&1&-1&-1&-1&1&-1&-1\\ -1&-1&-1&1&-1&1&-1&-1&-1\\ -1&-1&-1&-1&1&-1&-1&-1&-1\\ -1&-1&-1&1&-1&1&-1&-1&-1\\ -1&-1&1&-1&-1&-1&1&-1&-1\\ -1&1&-1&-1&-1&-1&-1&1&-1\\ -1&-1&-1&-1&-1&-1&-1&-1&-1 \end{bmatrix} \otimes \begin{bmatrix} 1&-1&-1\\ -1&1&-1\\ -1&-1&1 \end{bmatrix} = \begin{bmatrix} 0.77&-0.11&0.11&0.33&0.55&-0.11&0.33\\ -0.11&1.00&-0.11&0.33&-0.11&0.11&-0.11\\ 0.11&-0.11&1.00&-0.33&0.11&-0.11&0.55\\ 0.33&0.33&-0.33&0.55&-0.33&0.33&0.33\\ 0.55&-0.11&0.11&-0.33&1.00&-0.11&0.11\\ -0.11&0.11&-0.11&0.33&-0.11&1.00&-0.11\\ 0.33&-0.11&0.55&0.33&0.11&-0.11&0.77 \end{bmatrix}$$

$$\begin{bmatrix} -1&-1&-1&-1&-1&-1&-1&-1&-1\\ -1&1&-1&-1&-1&-1&-1&1&-1\\ -1&-1&1&-1&-1&-1&1&-1&-1\\ -1&-1&-1&1&-1&1&-1&-1&-1\\ -1&-1&-1&-1&1&-1&-1&-1&-1\\ -1&-1&-1&1&-1&1&-1&-1&-1\\ -1&-1&1&-1&-1&-1&1&-1&-1\\ -1&1&-1&-1&-1&-1&-1&1&-1\\ -1&-1&-1&-1&-1&-1&-1&-1&-1 \end{bmatrix} \otimes \begin{bmatrix} 1&-1&1\\ -1&1&-1\\ 1&-1&1 \end{bmatrix} = \begin{bmatrix} 0.33&-0.55&0.11&-0.11&0.11&-0.55&0.33\\ -0.55&0.55&-0.55&0.33&-0.55&0.55&-0.55\\ 0.11&-0.55&0.55&-0.77&0.55&-0.55&0.11\\ -0.11&0.33&-0.77&1.00&-0.77&0.33&-0.11\\ 0.11&-0.55&0.55&-0.77&0.55&-0.55&0.11\\ -0.55&0.55&-0.55&0.33&-0.55&0.55&-0.55\\ 0.33&-0.55&0.11&-0.11&0.11&-0.55&0.33 \end{bmatrix}$$

$$\begin{bmatrix} -1&-1&-1&-1&-1&-1&-1&-1&-1\\ -1&1&-1&-1&-1&-1&-1&1&-1\\ -1&-1&1&-1&-1&-1&1&-1&-1\\ -1&-1&-1&1&-1&1&-1&-1&-1\\ -1&-1&-1&-1&1&-1&-1&-1&-1\\ -1&-1&-1&1&-1&1&-1&-1&-1\\ -1&-1&1&-1&-1&-1&1&-1&-1\\ -1&1&-1&-1&-1&-1&-1&1&-1\\ -1&-1&-1&-1&-1&-1&-1&-1&-1 \end{bmatrix} \otimes \begin{bmatrix} -1&-1&1\\ -1&1&-1\\ 1&-1&-1 \end{bmatrix} = \begin{bmatrix} 0.33&-0.11&0.55&0.33&0.11&-0.11&0.77\\ -0.11&0.11&-0.11&0.33&-0.11&1.00&-0.11\\ 0.55&-0.11&0.11&-0.33&1.00&-0.11&0.11\\ 0.33&0.33&-0.33&0.55&-0.33&0.33&0.33\\ 0.11&-0.11&1.00&-0.33&0.11&-0.11&0.55\\ -0.11&1.00&-0.11&0.33&-0.11&0.11&-0.11\\ 0.77&-0.11&0.11&0.33&0.55&-0.11&0.33 \end{bmatrix}$$

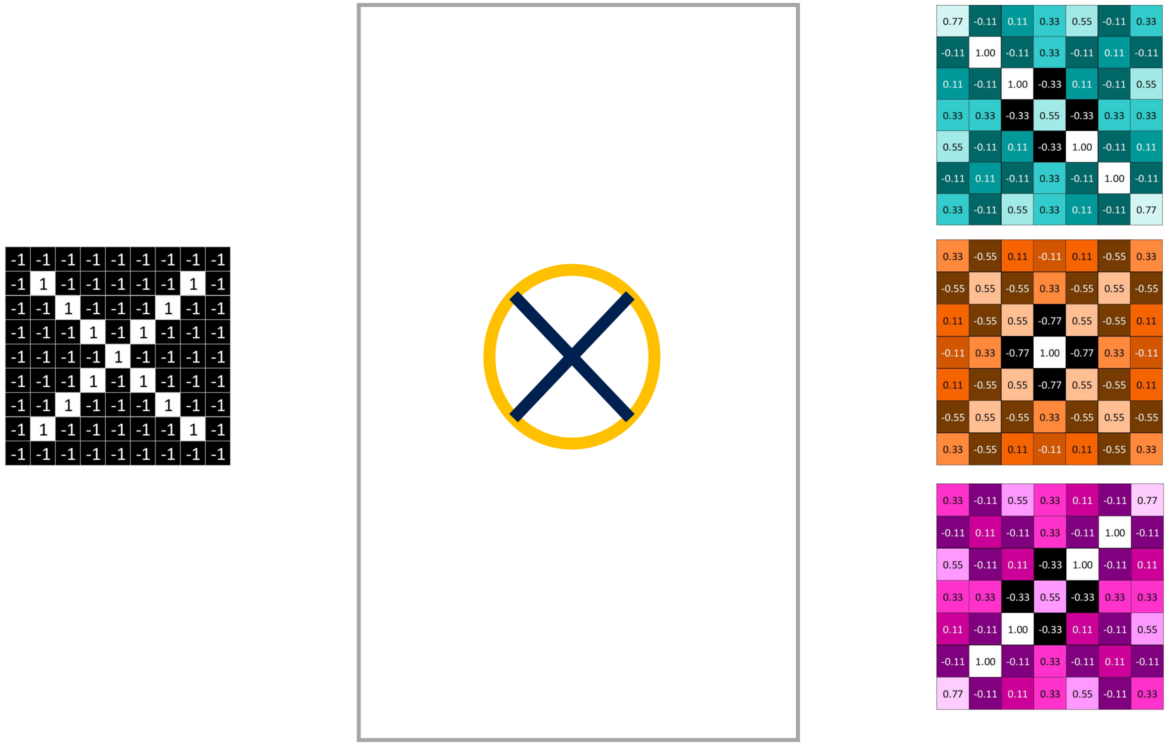
Convolution layer

One image becomes a stack of filtered images



Convolution layer

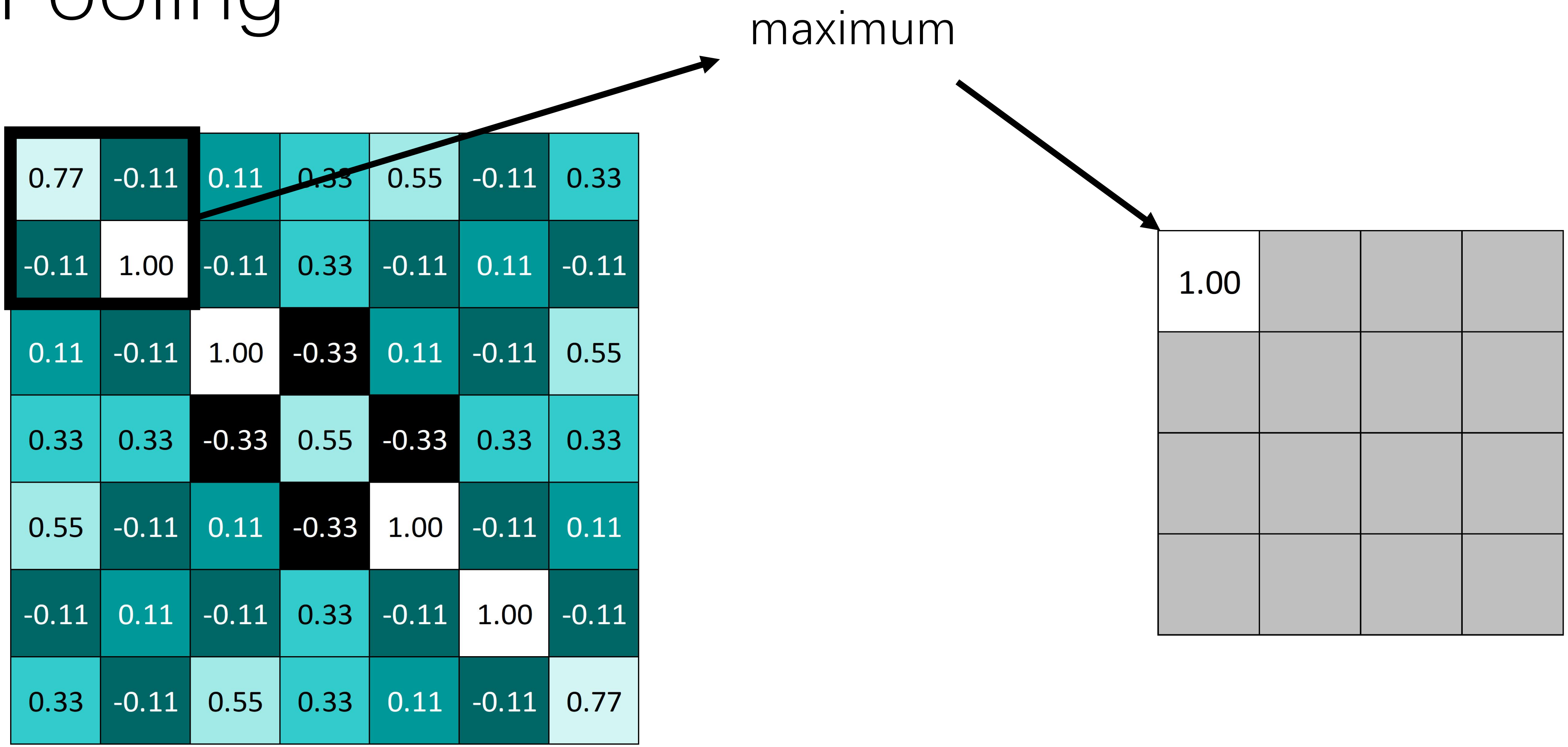
One image becomes a stack of filtered images



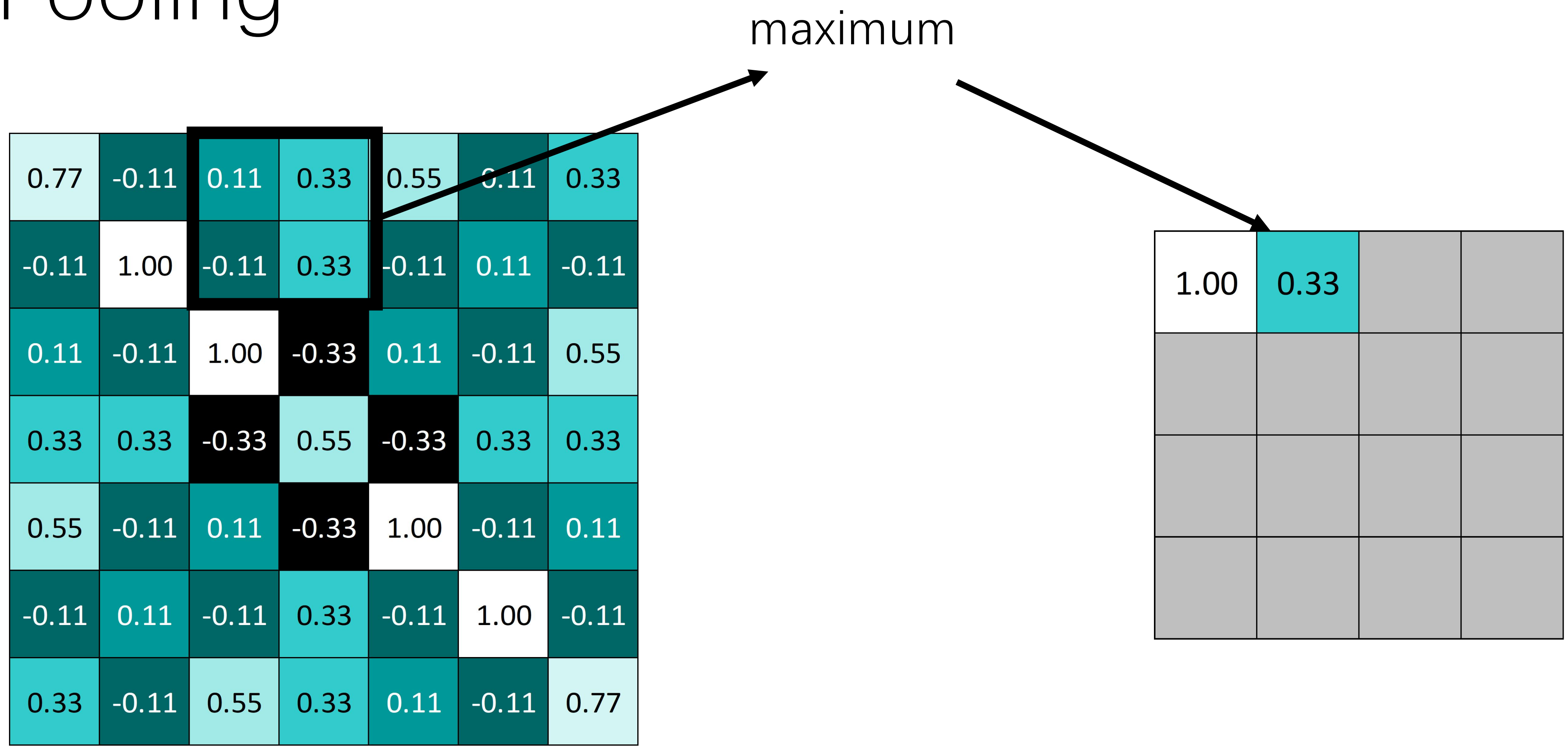
Pooling: Shrinking the image stack

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.

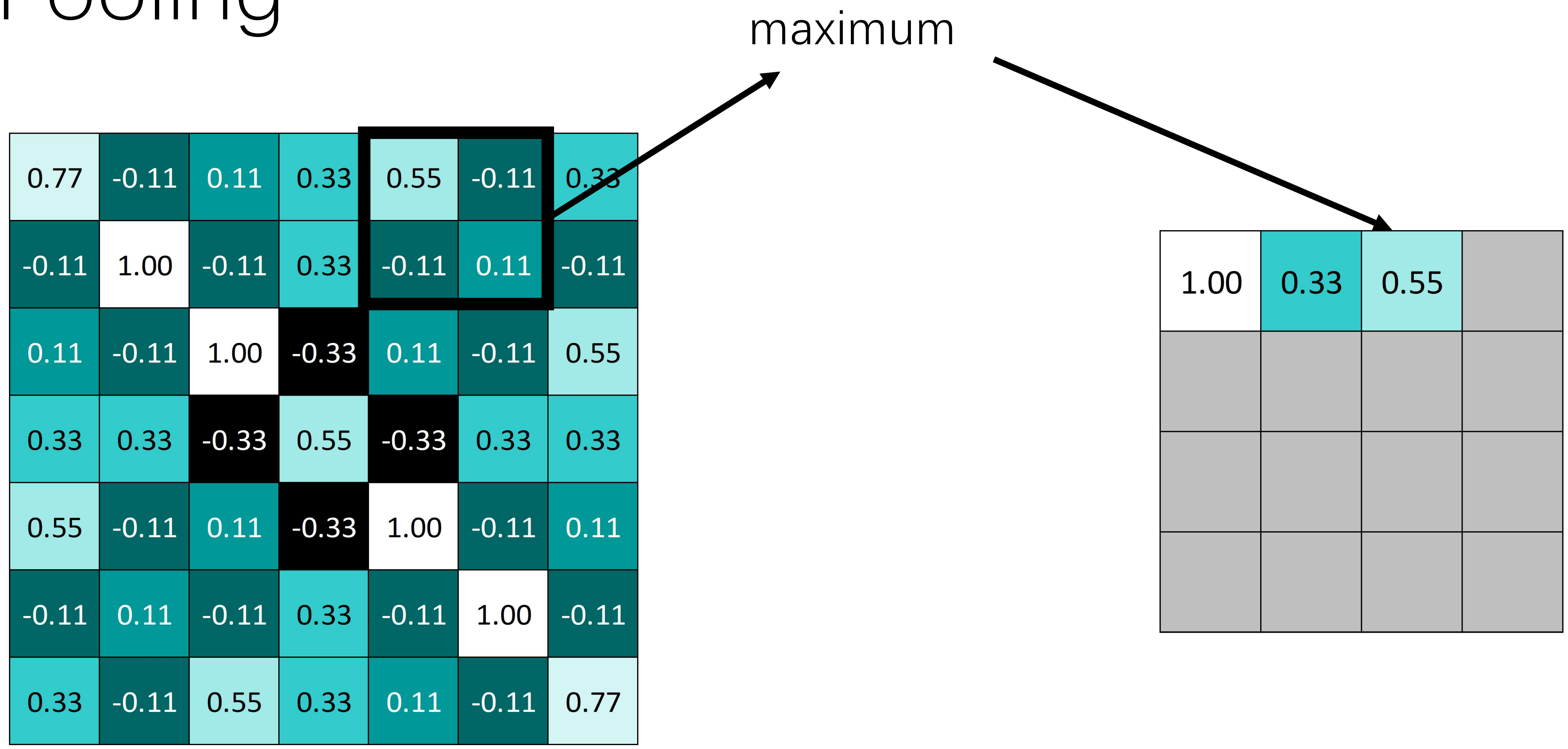
Pooling


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
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

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
maximum
1.00 0.33 0.55

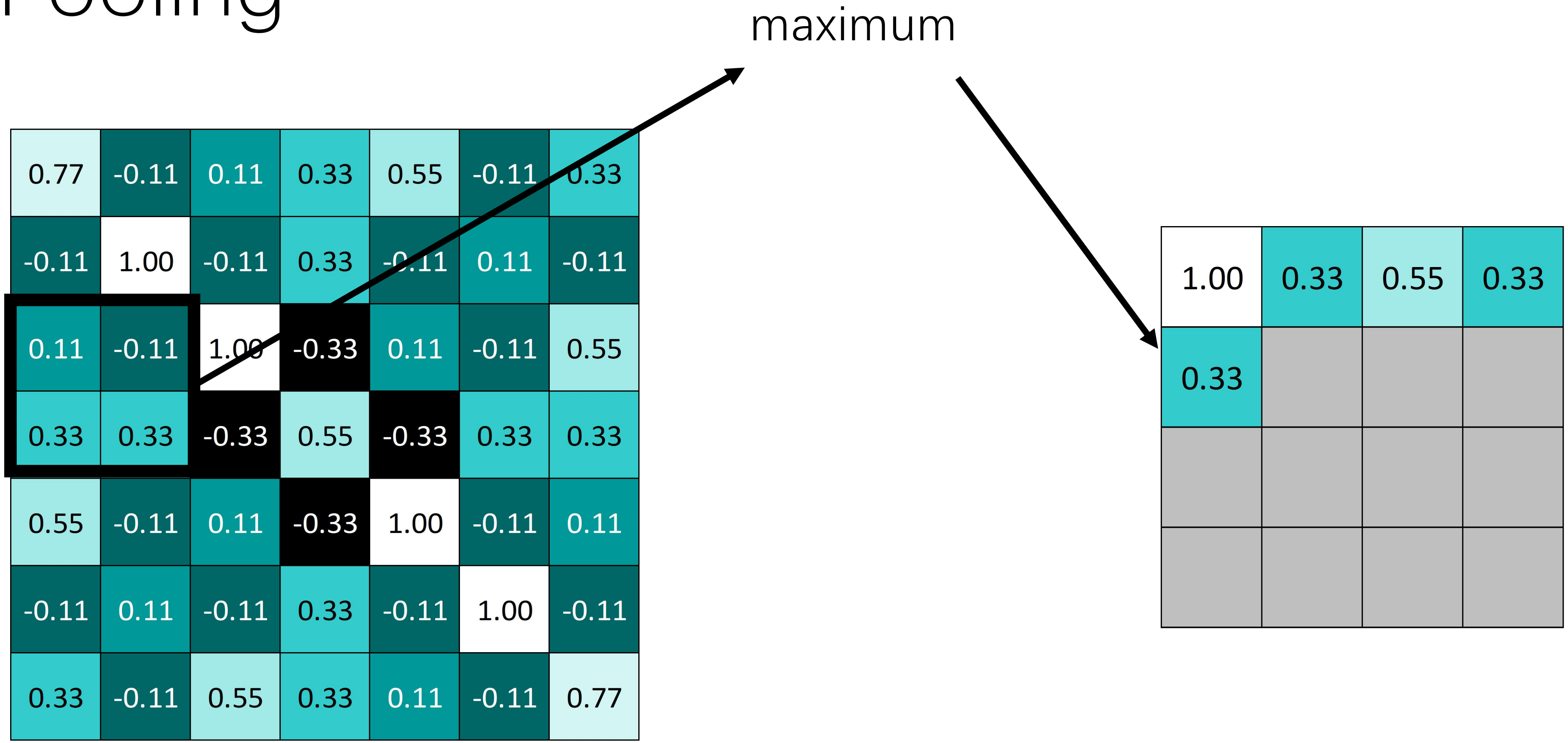
Pooling

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

maximum

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

max pooling

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

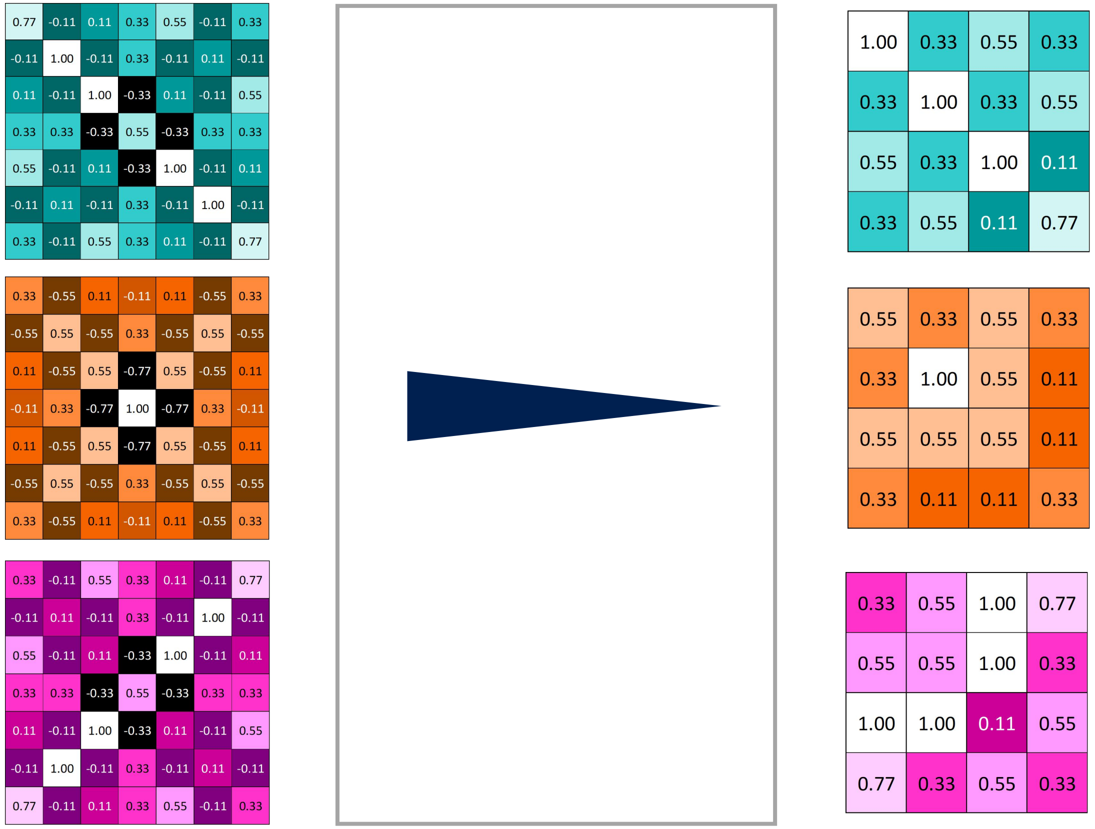
0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

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.

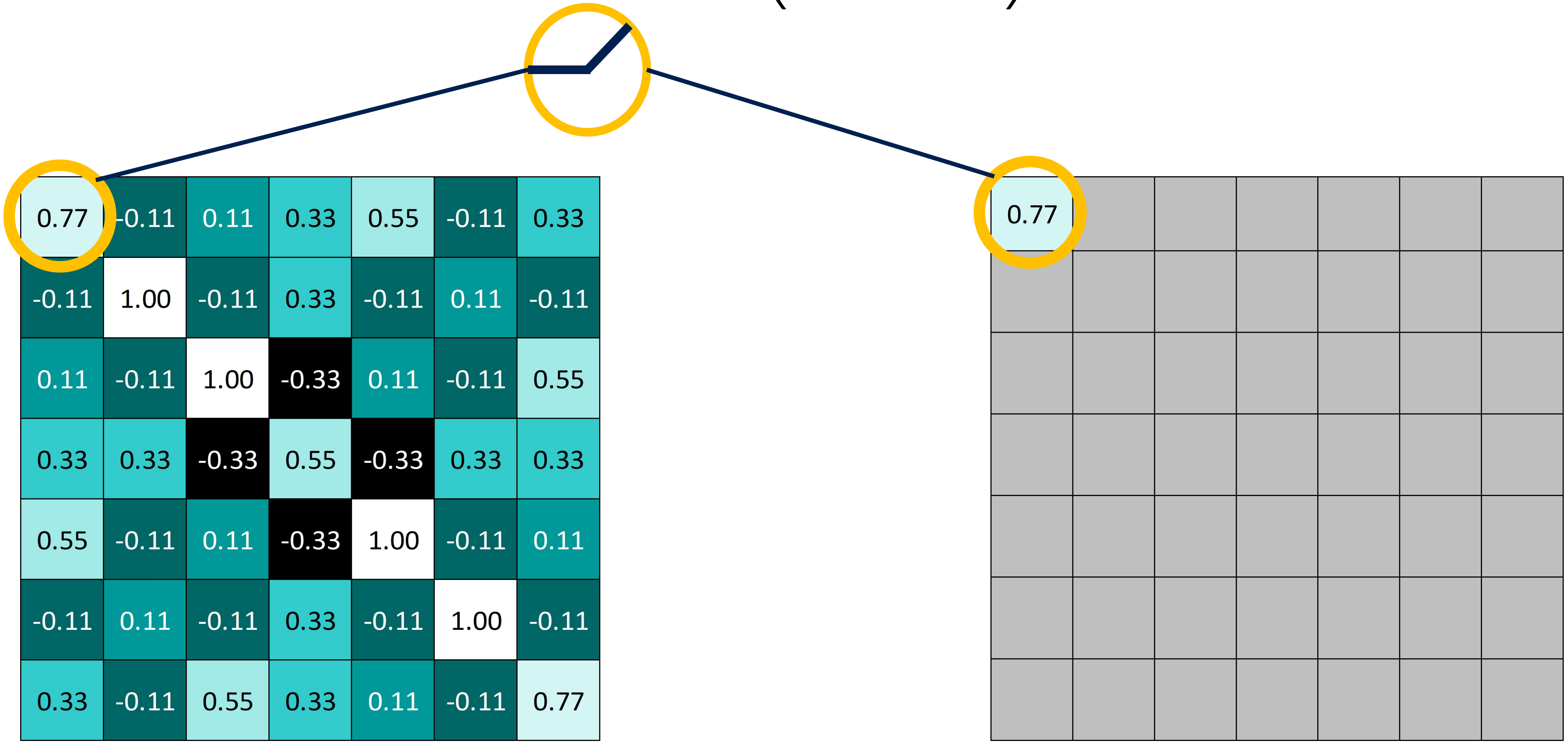


Normalization

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

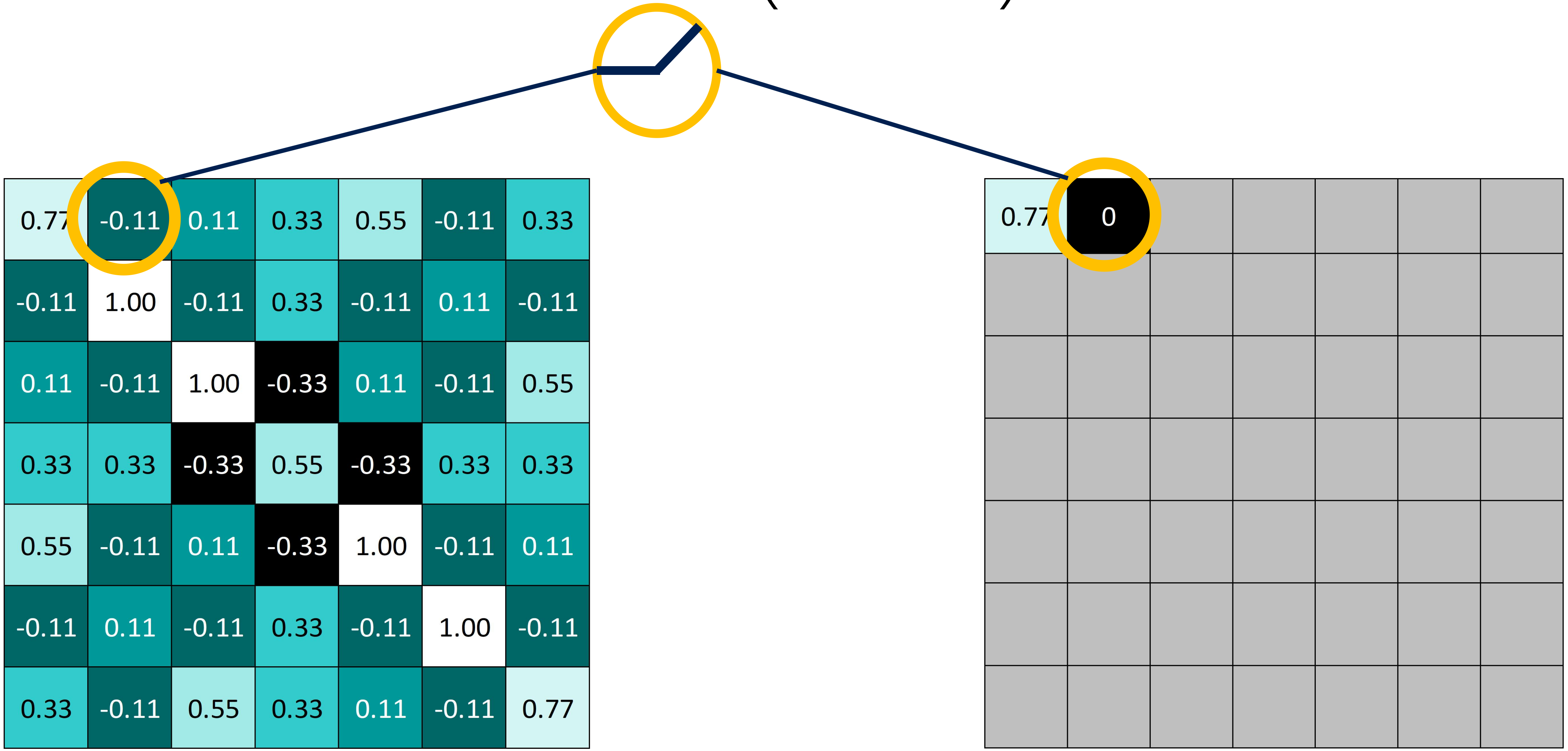
Change everything negative to zero.

Rectified Linear Units (ReLUs)



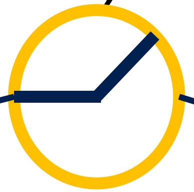
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

Rectified Linear Units (ReLUs)



Rectified Linear Units (ReLUs)

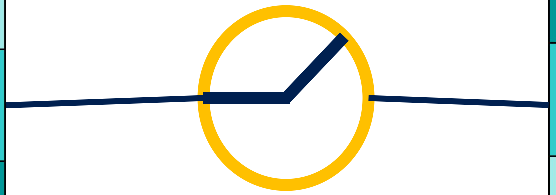
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77



0.77	0	0.11	0.33	0.55	0	0.33

Rectified Linear Units (ReLUs)

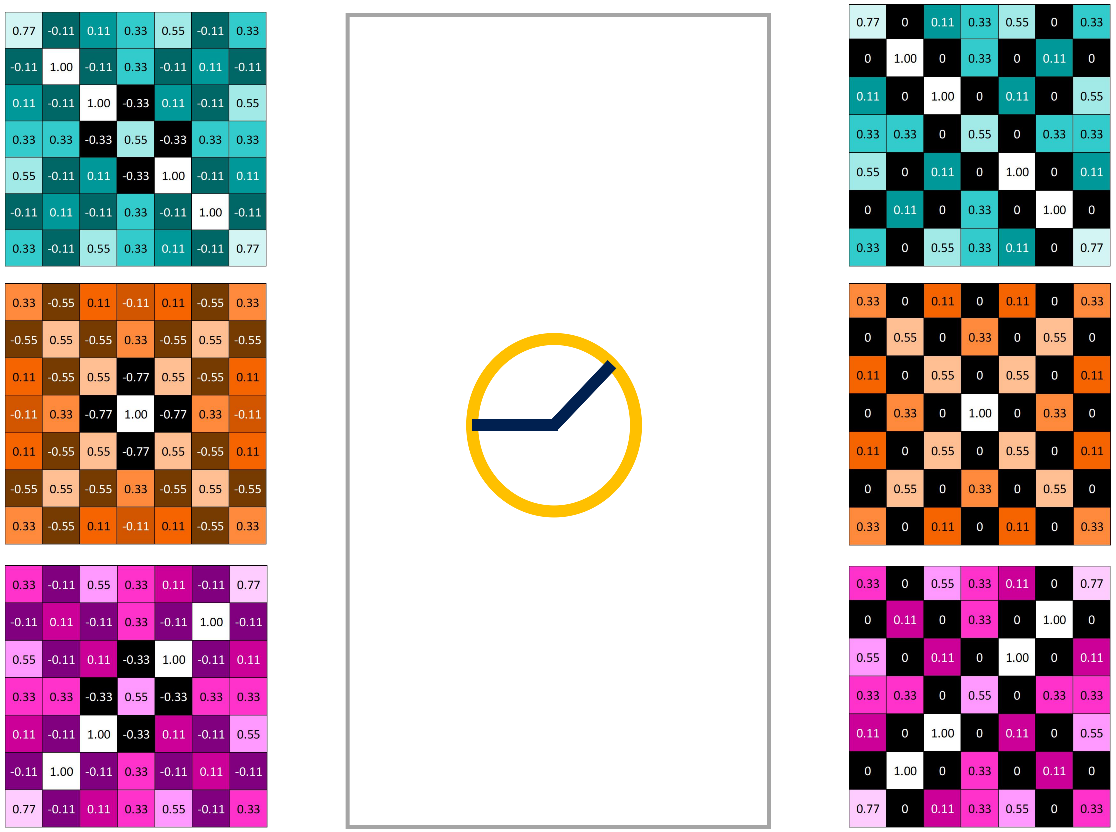
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77



0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

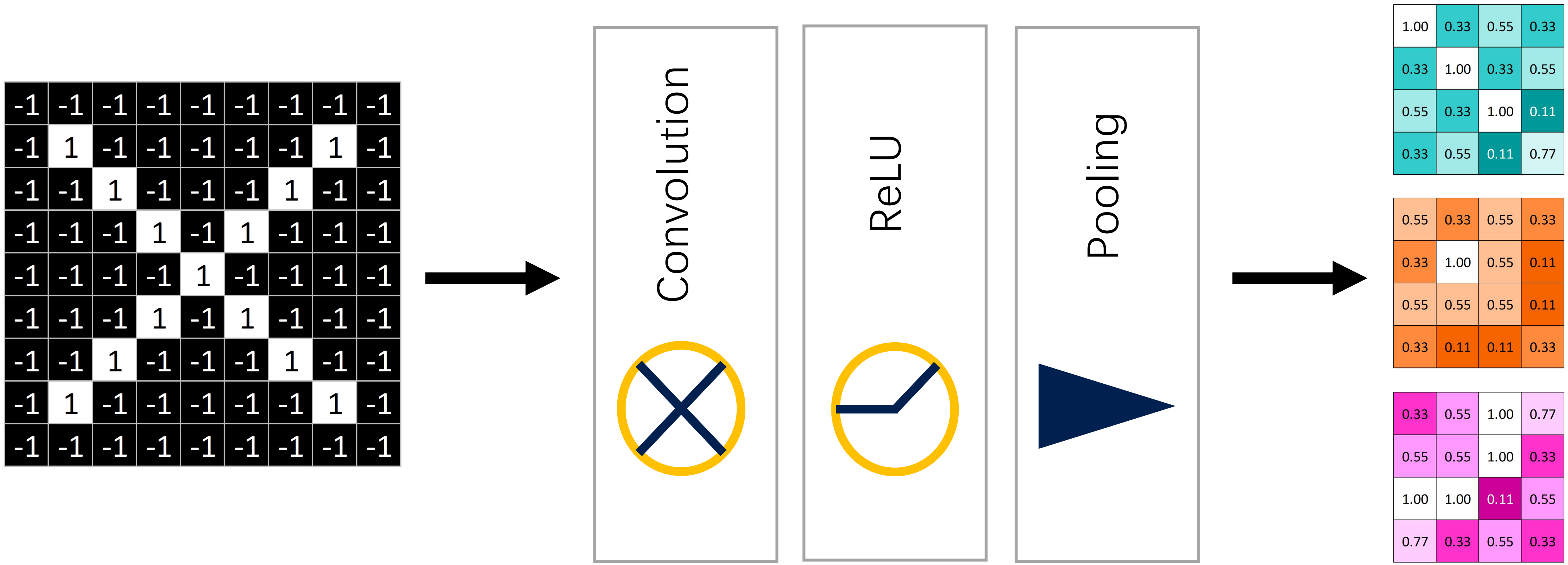
ReLU layer

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



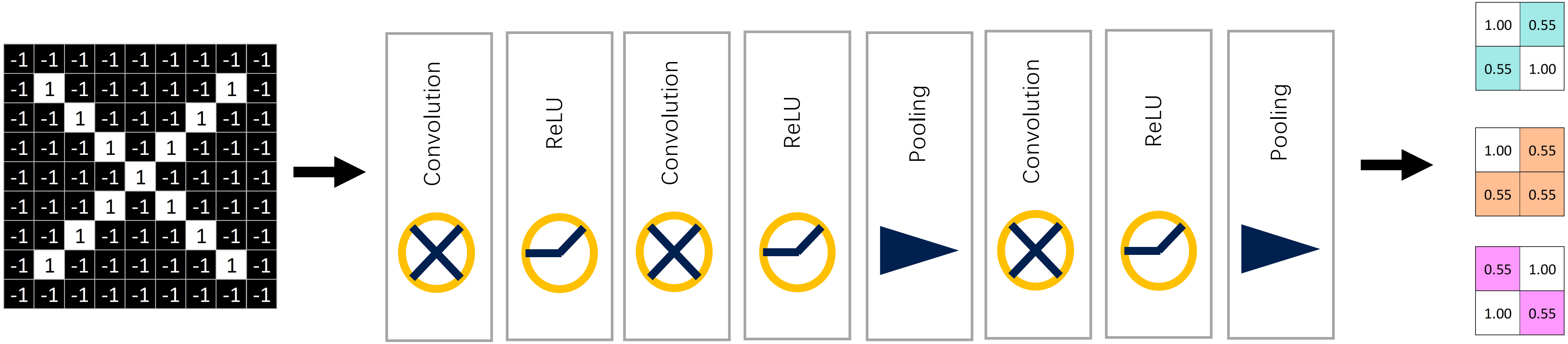
Layers get stacked

The output of one becomes the input of the next.



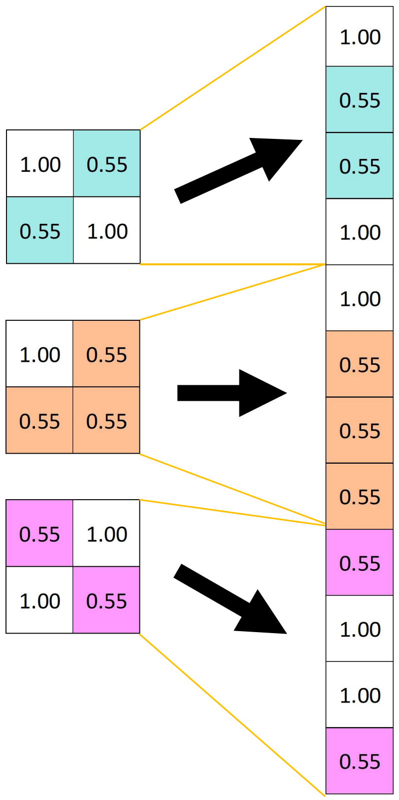
Deep stacking

Layers can be repeated several (or many) times.



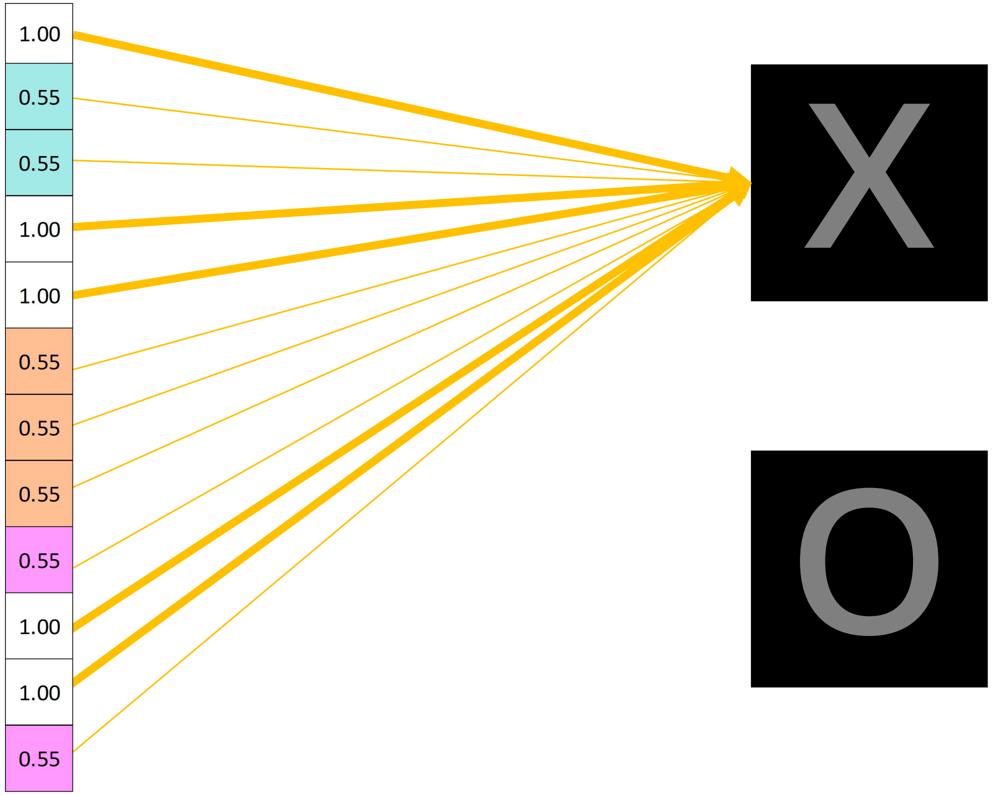
Fully connected layer

Every value gets a vote



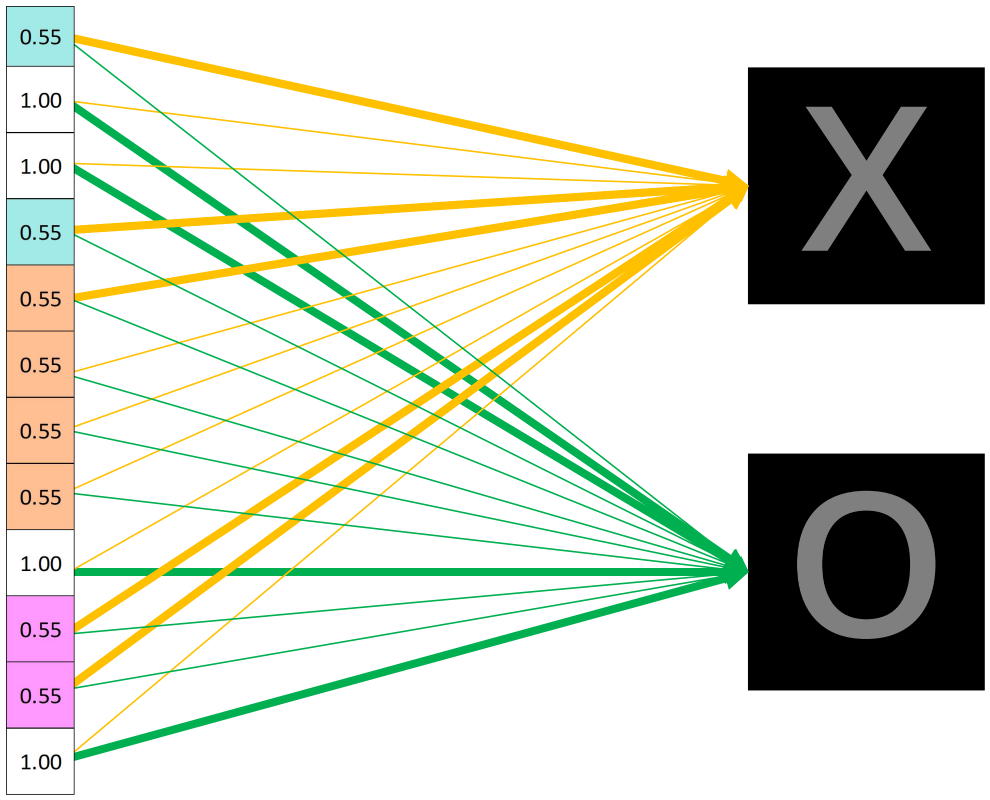
Fully connected layer

Vote depends on how strongly a value predicts X or O



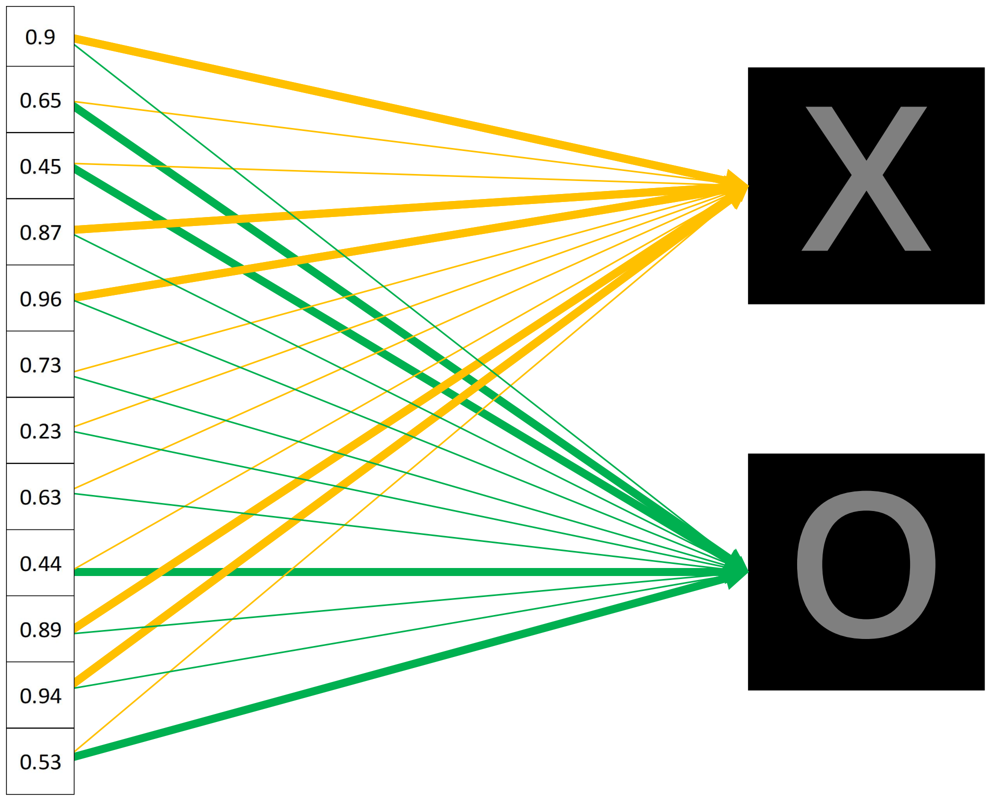
Fully connected layer

Vote depends on how strongly a value predicts X or O



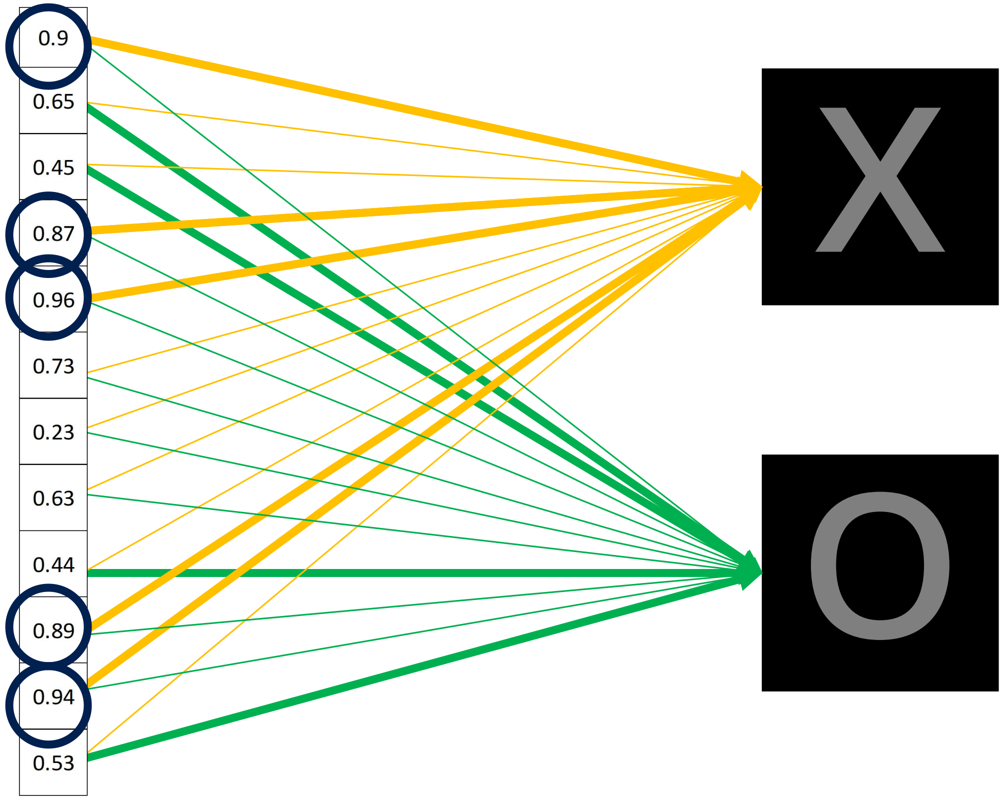
Fully connected layer

Future values vote on X or O



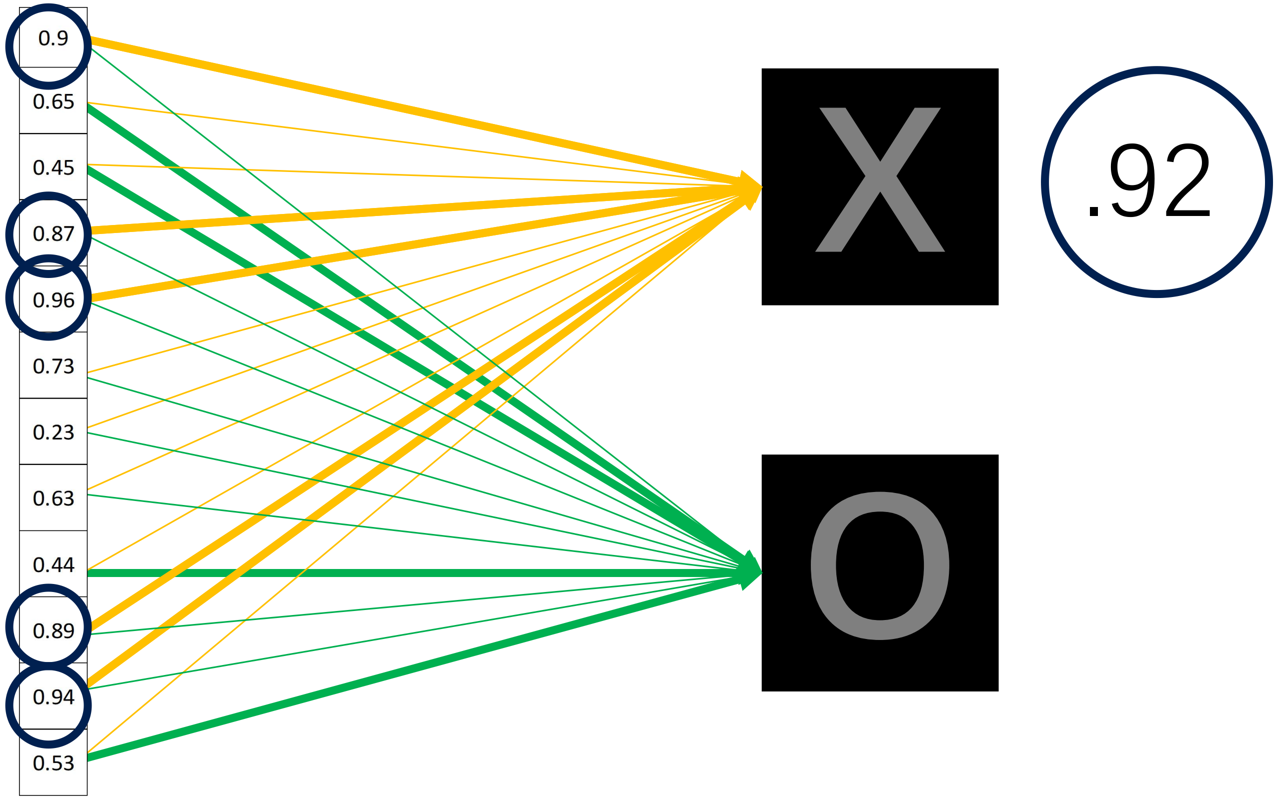
Fully connected layer

Future values vote on X or O



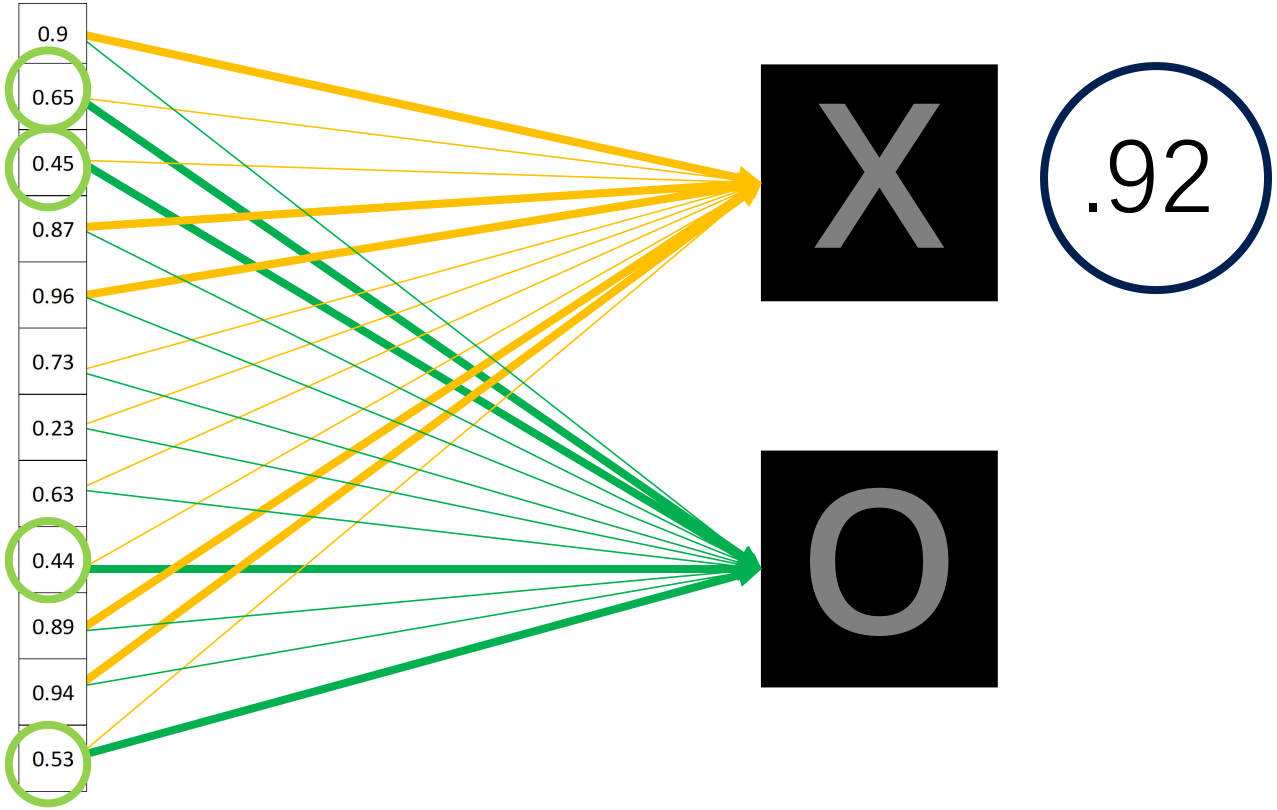
Fully connected layer

Future values vote on X or O



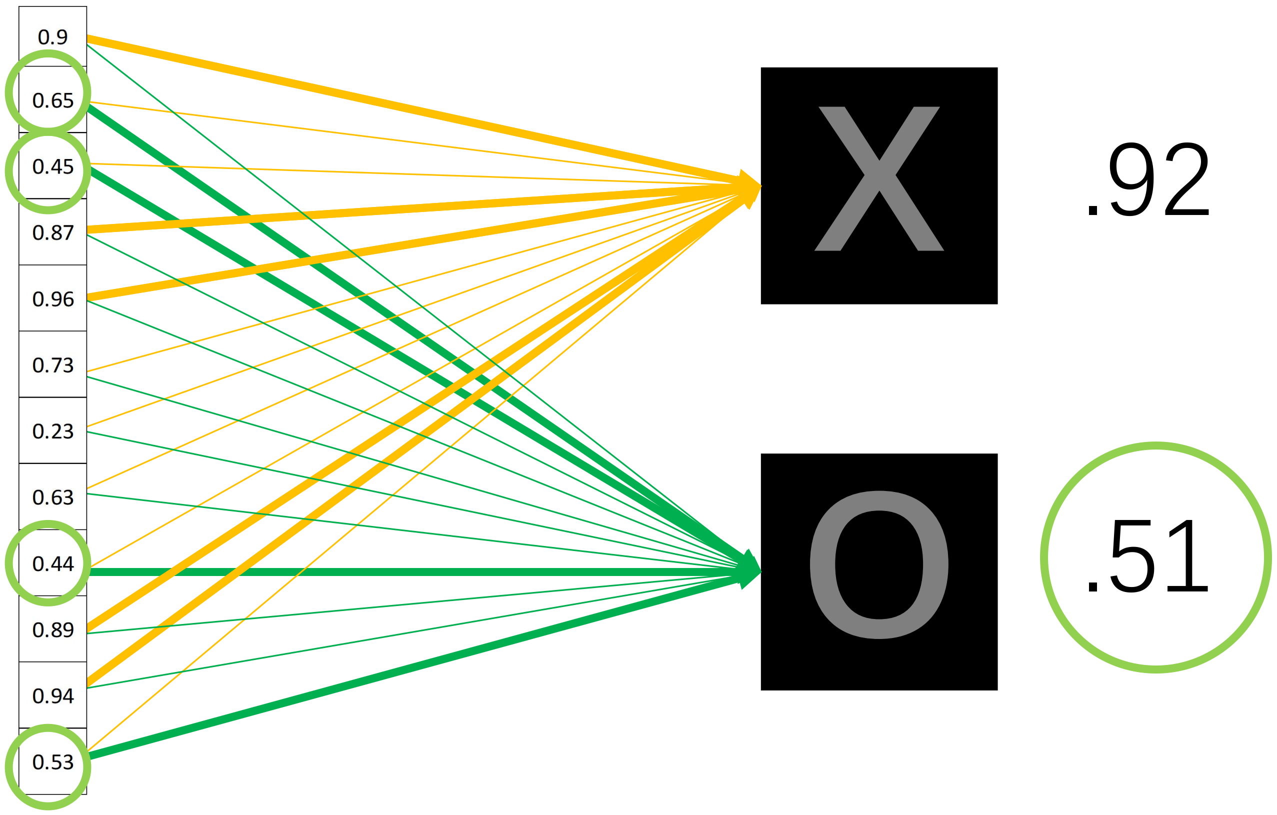
Fully connected layer

Future values vote on X or O



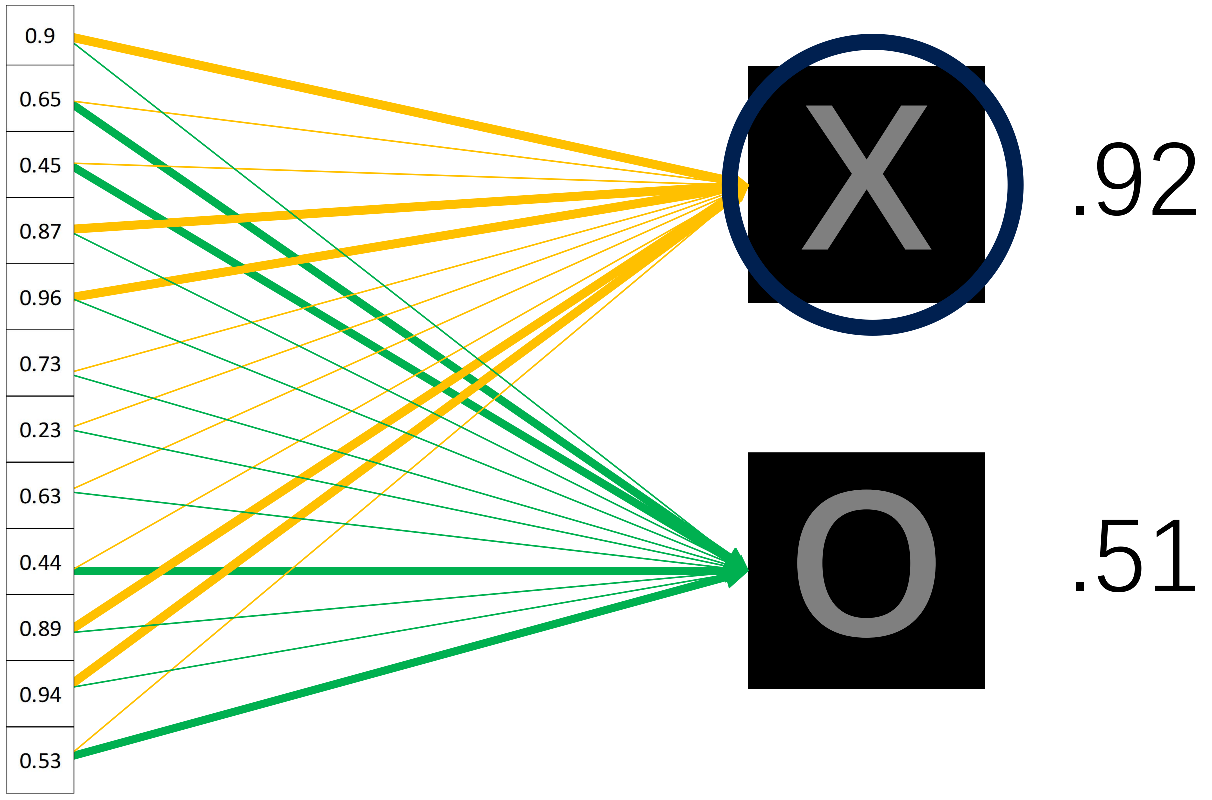
Fully connected layer

Future values vote on X or O



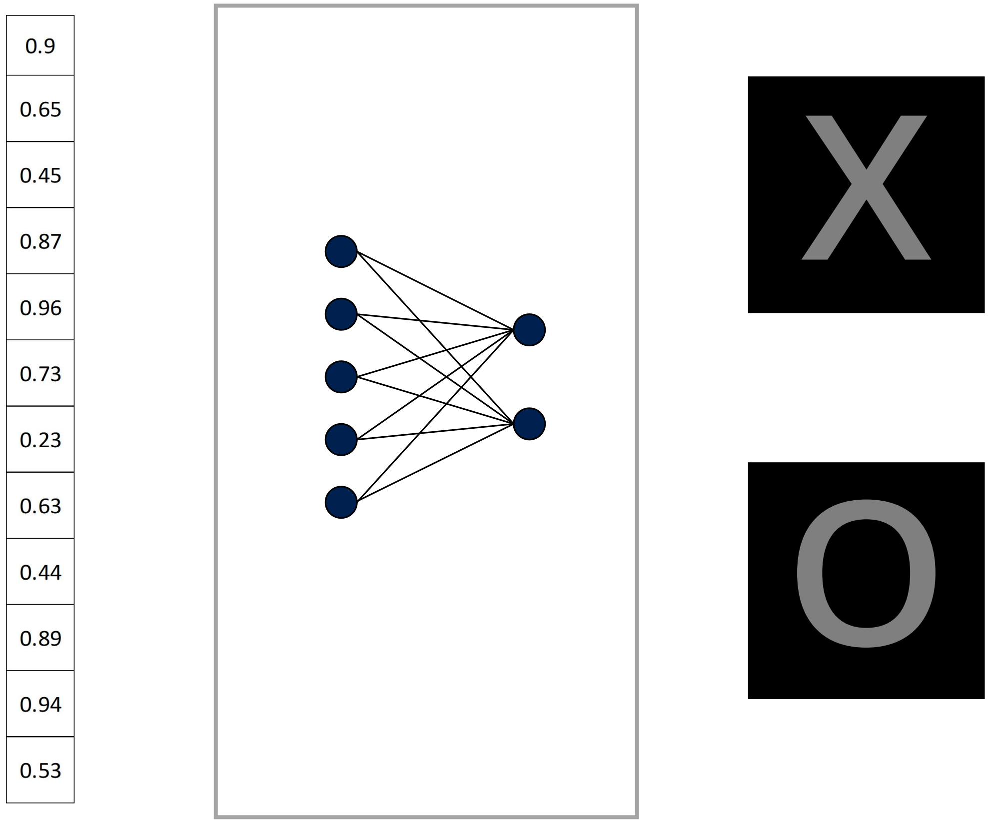
Fully connected layer

Future values vote on X or O



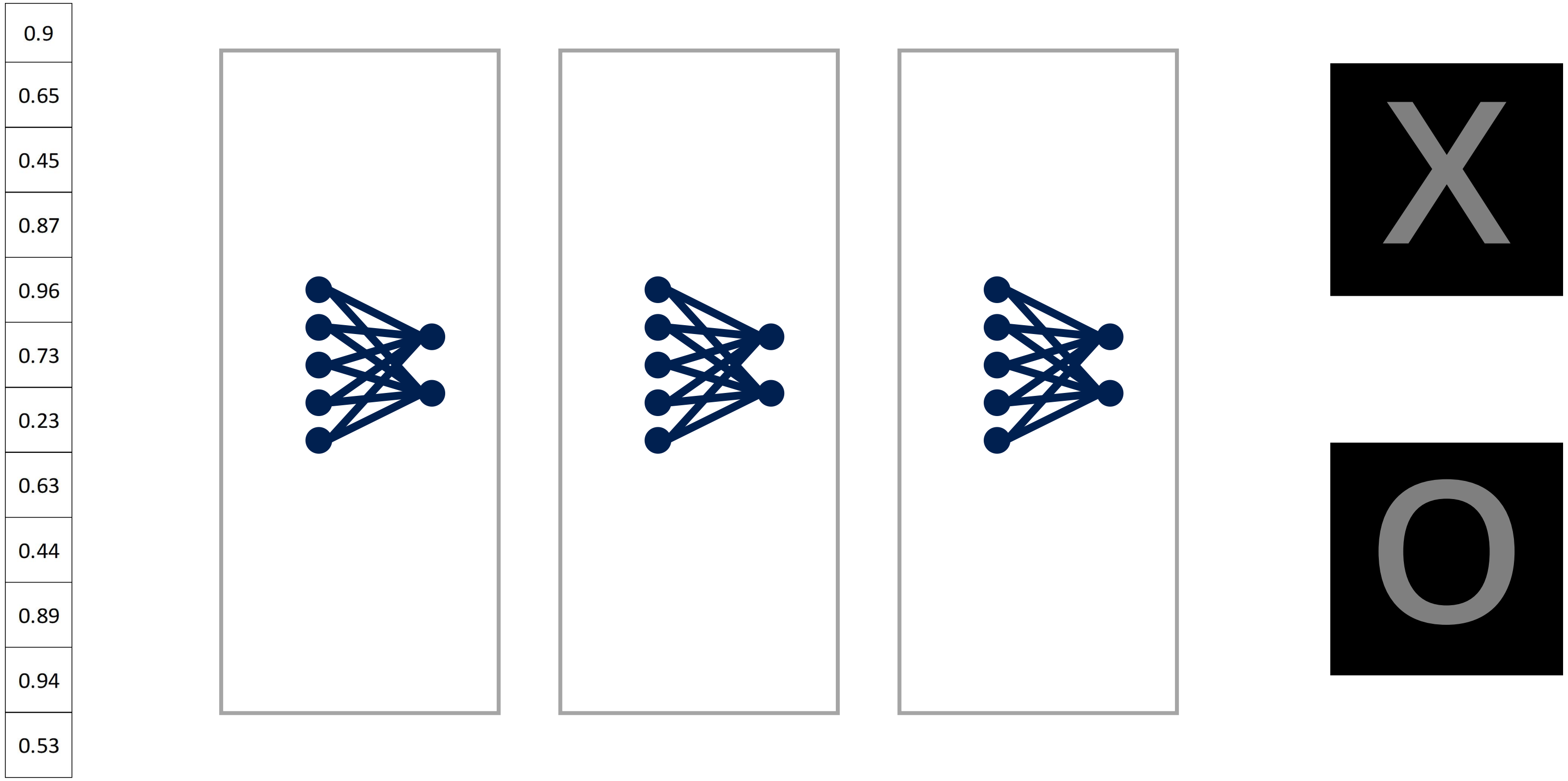
Fully connected layer

A list of feature values becomes a list of votes.



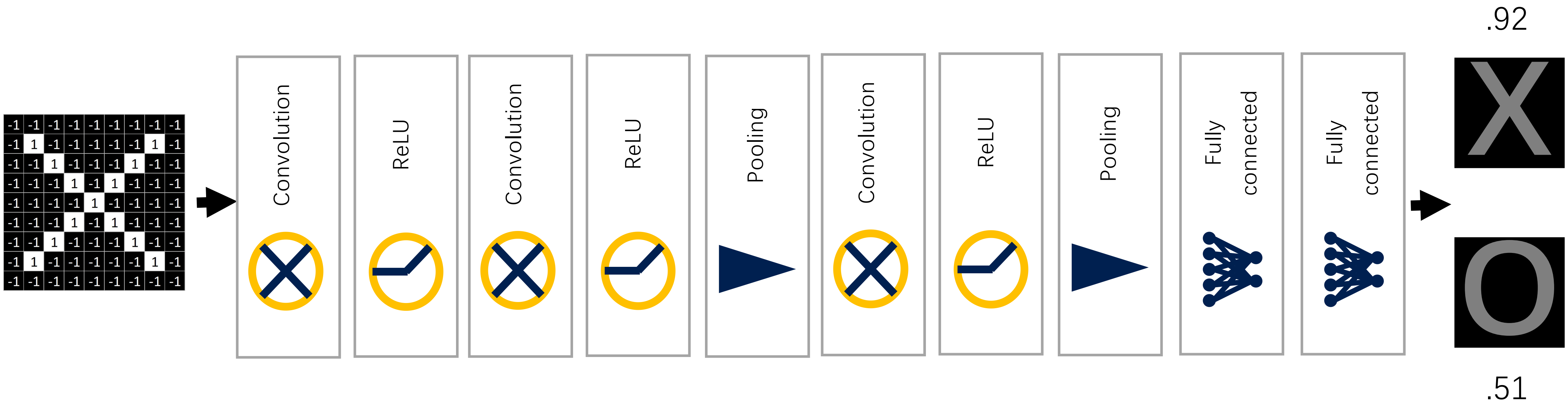
Fully connected layer

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?

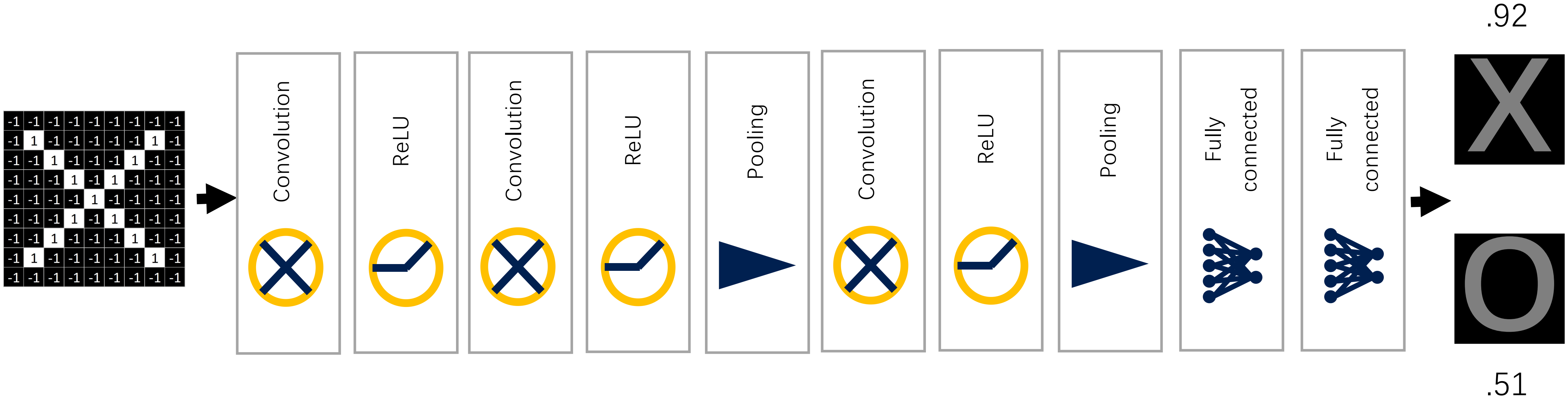
Features in convolutional layers

Voting weights in fully connected layers

A: Backpropagation

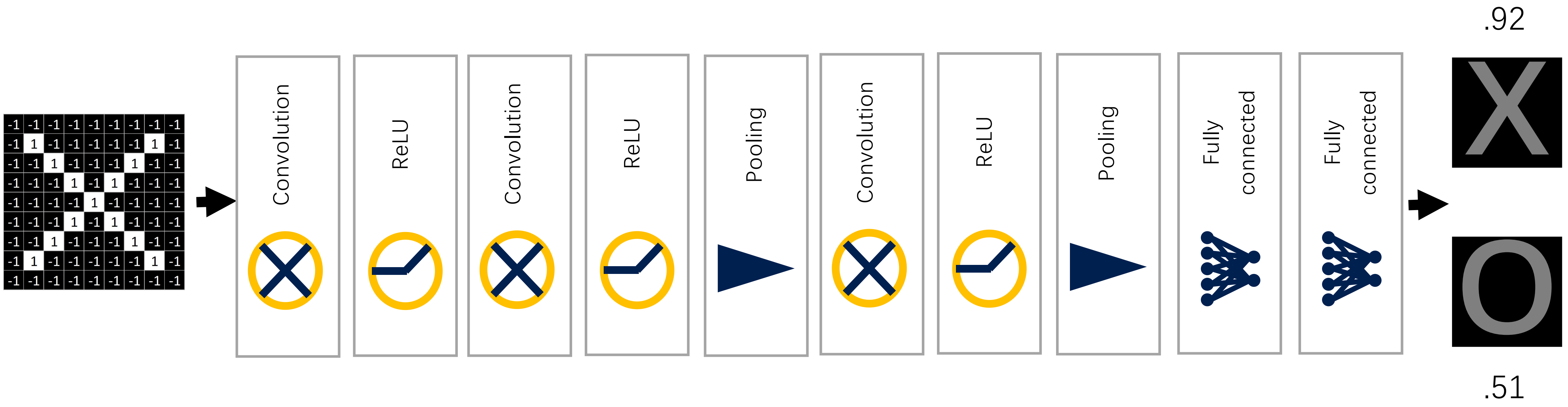
Backprop

Error = right answer – actual answer



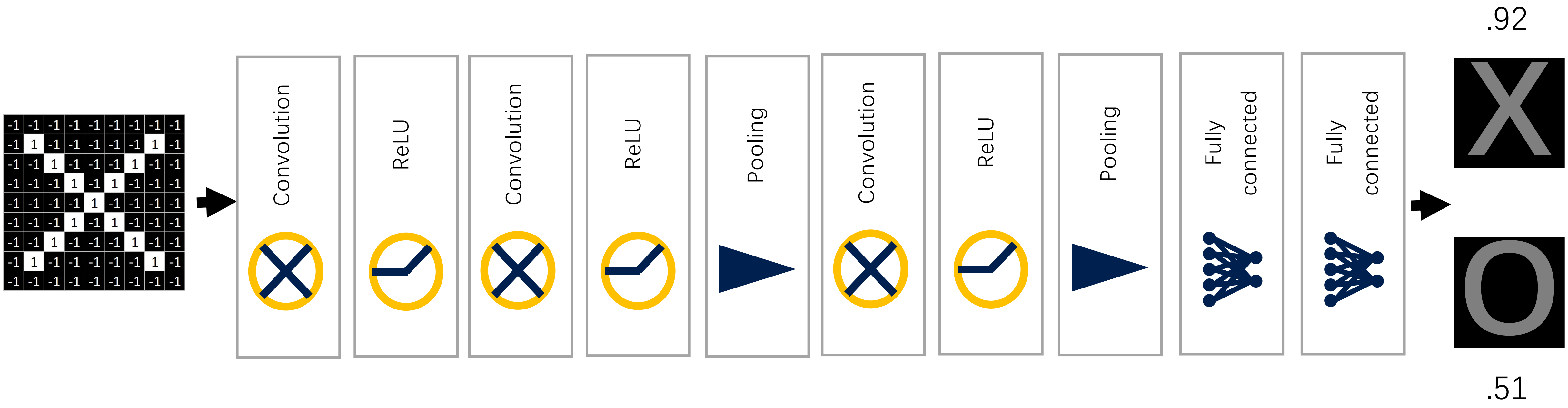
Backprop

	Right answer	Actual answer	Error
X	1		
O			



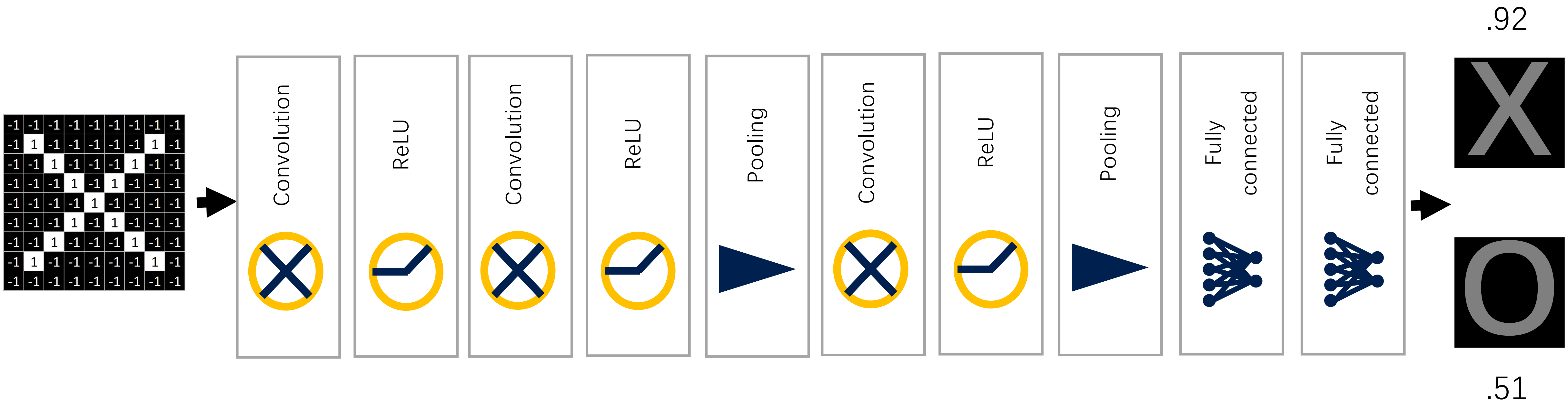
Backprop

	Right answer	Actual answer	Error
X	1	0.92	
O			



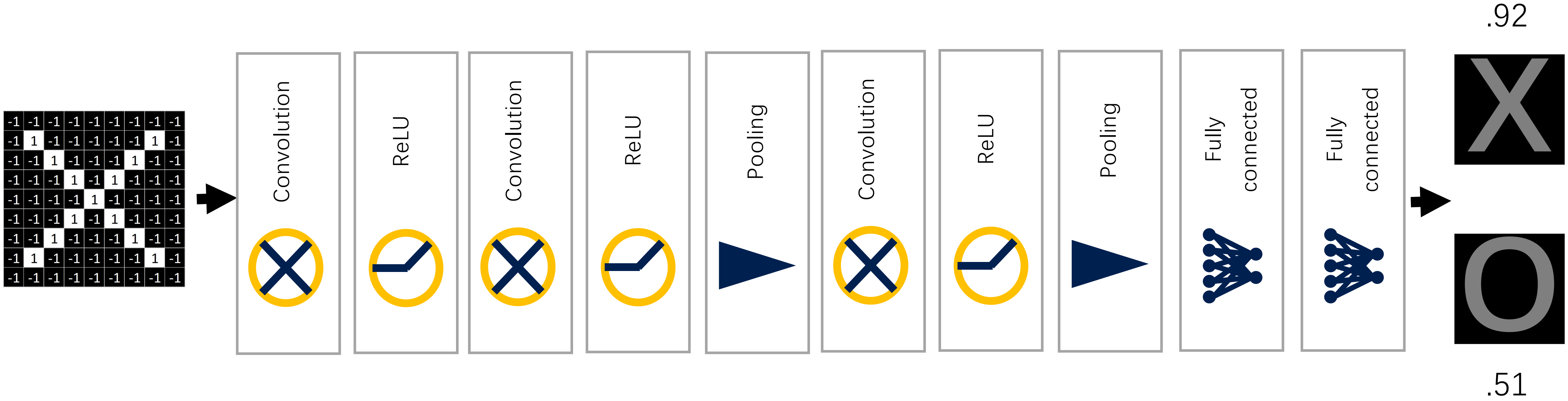
Backprop

	Right answer	Actual answer	Error
X	1	0.92	0.08
O			



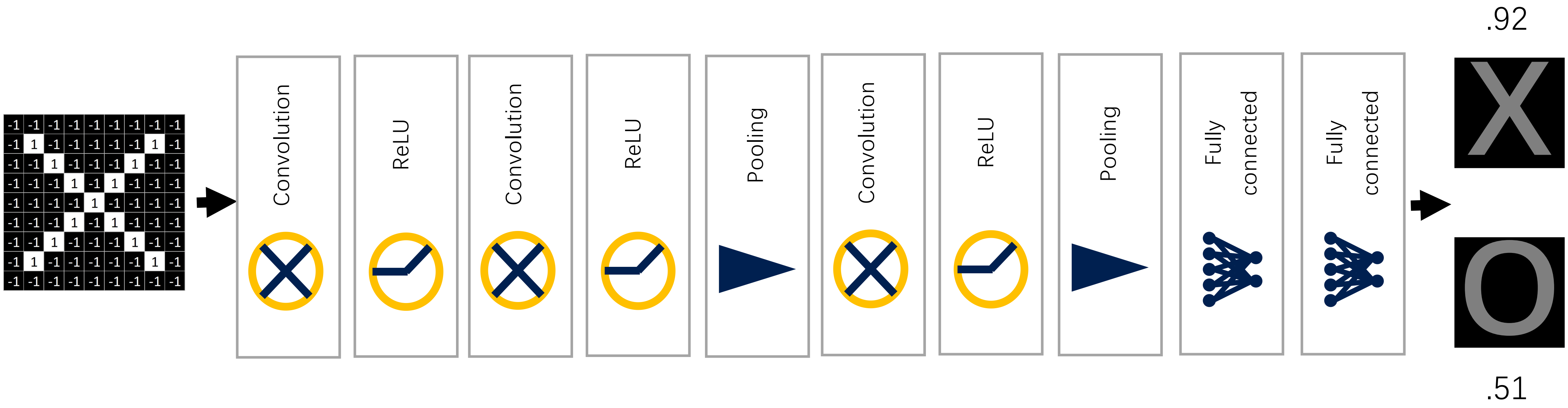
Backprop

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



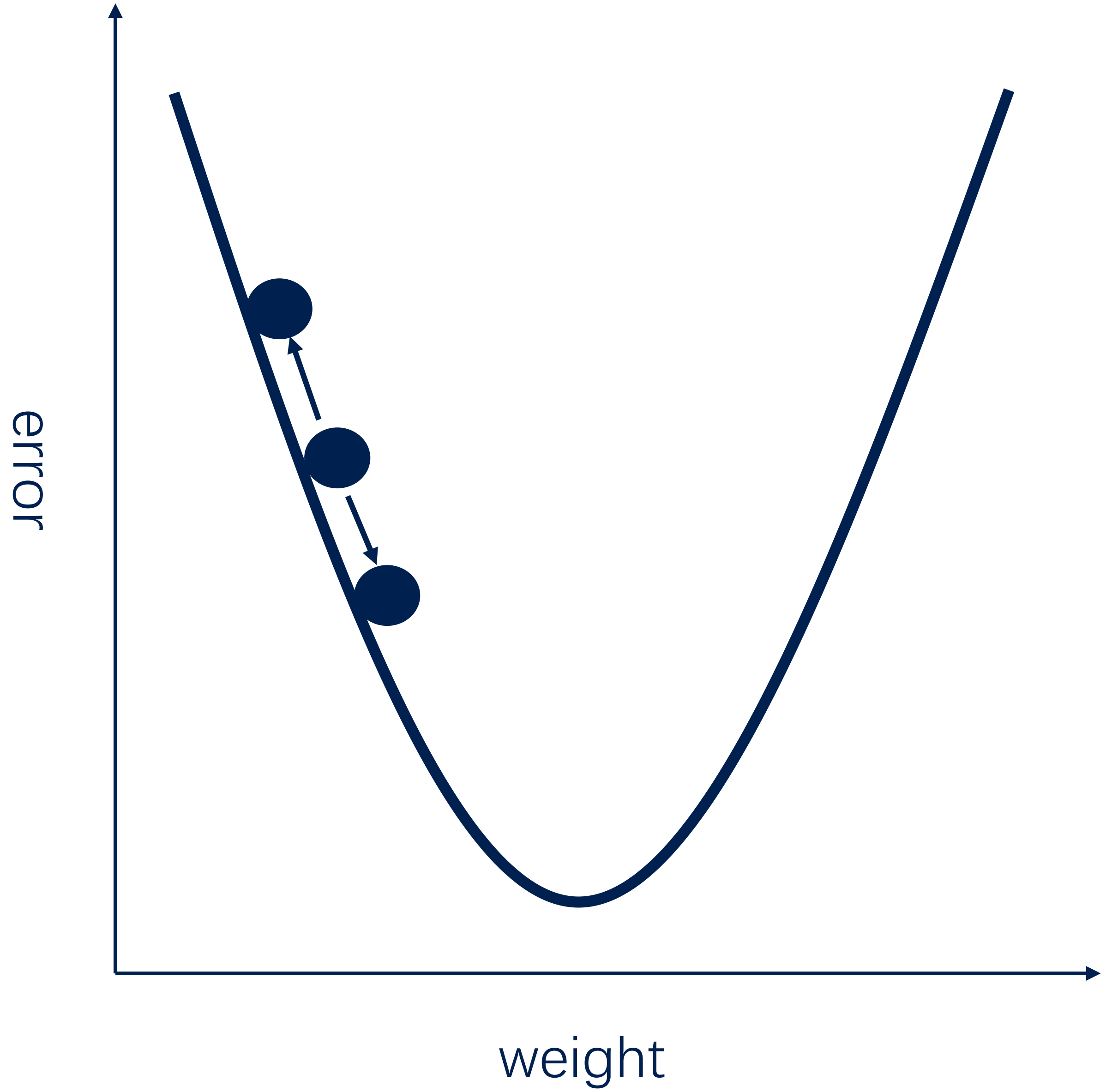
Backprop

	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.



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?

Classical CNNs

- LeNet-5
- AlexNet
- VGGNet
- GoogLeNet
- ResNet

LeNet-5

- *Gradient Based Learning Applied To Document Recognition - Y. Lecun, L. Bottou, Y. Bengio, P. Haffner; 1998*
- Helped establish how we use CNNs today
- Replaced manual feature extraction

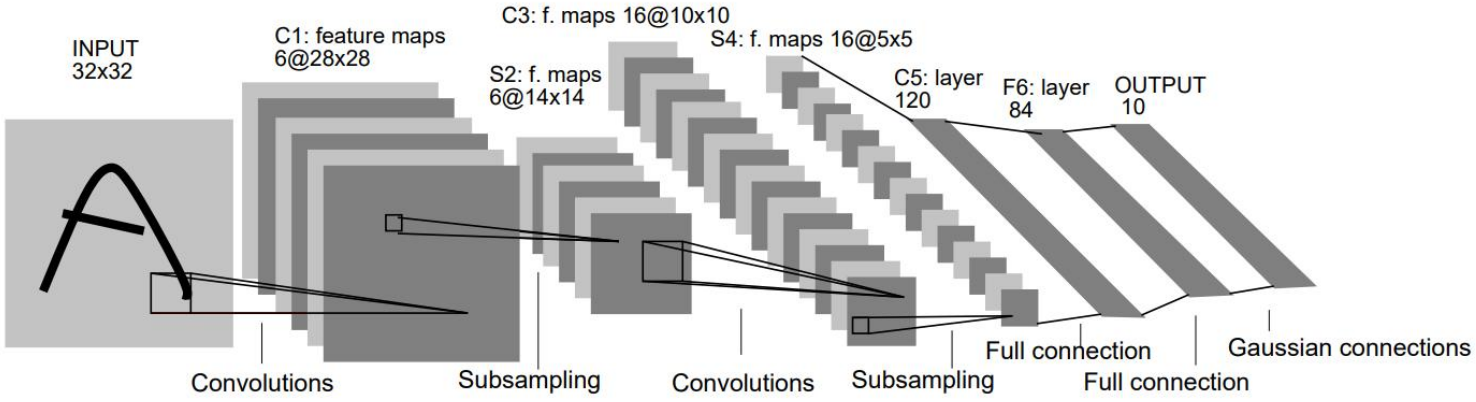
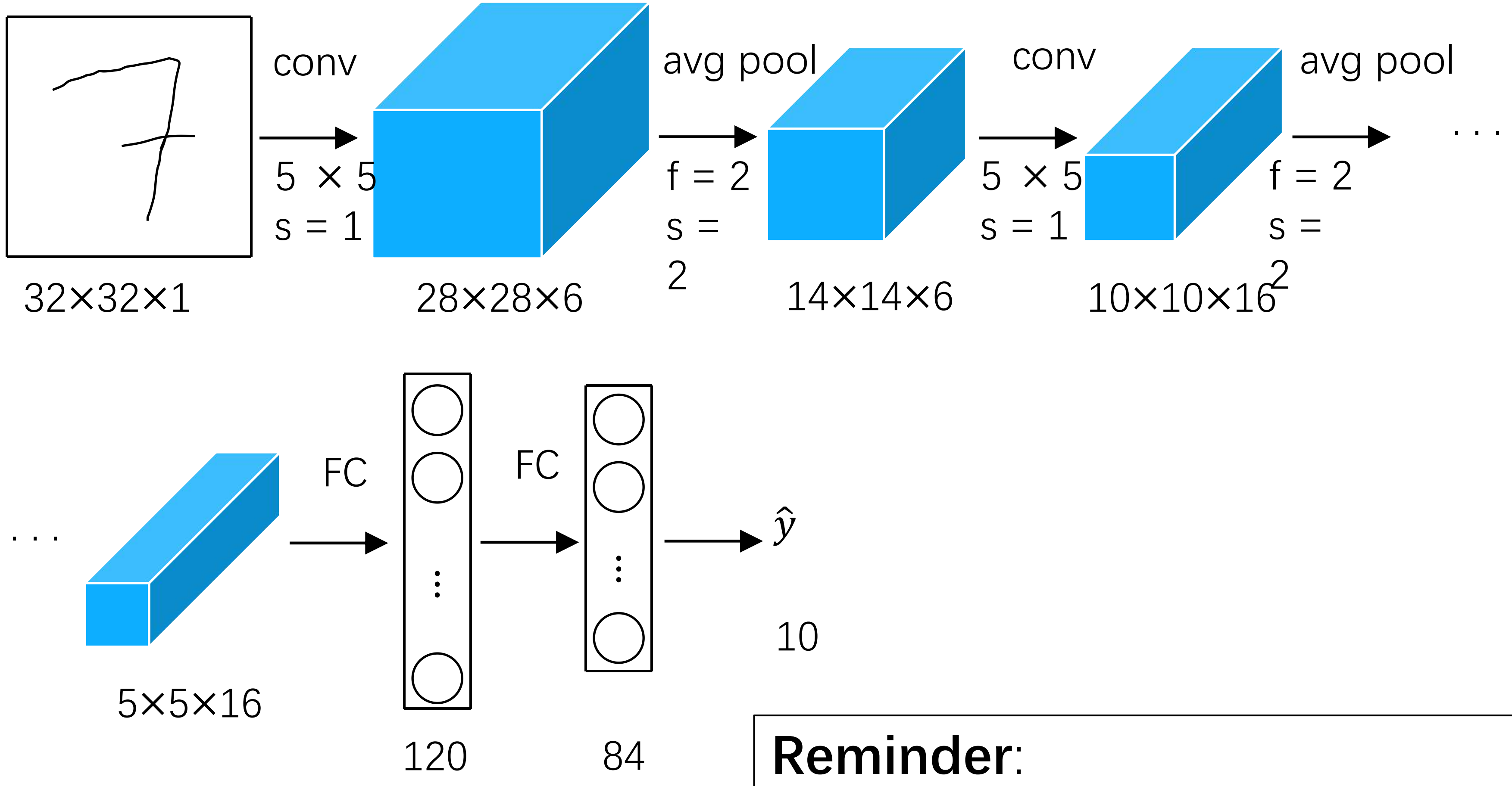


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

[**LeCun et al**., **1998**]

LeNet-5



Reminder:
Output size = (N+2P-F)/stride + 1

[**LeCun et al.**, **1998**]

LeNet-5

- Only 60K parameters
- As we go deeper in the network: $N_H\downarrow$, $N_W\downarrow$, $N_C\uparrow$
- General structure:
  conv->pool->conv->pool->FC->FC->output

- Different filters look at different channels
- Sigmoid and Tanh nonlinearity

[**LeCun et al**., **1998**]

AlexNet

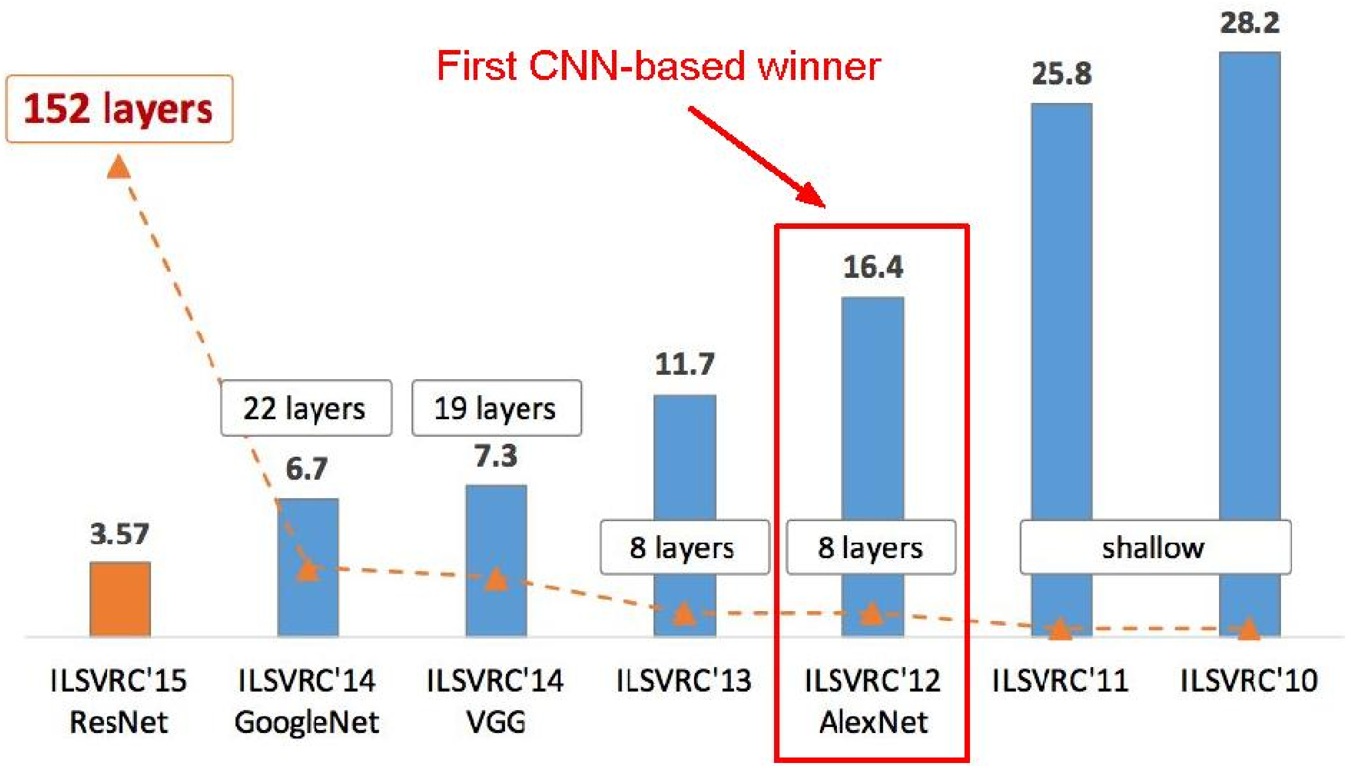
- *ImageNet Classification with Deep Convolutional Neural Networks - Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton; 2012*
- Facilitated by GPUs, highly optimized convolution implementation and large datasets (ImageNet)
- One of the largest CNNs to date
- Has 60 Million parameter compared to 60k parameter of LeNet-5

[**Krizhevsky et al**., **2012**]

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- The annual “Olympics” of computer vision.
- Teams from across the world compete to see who has the best computer vision model for tasks such as classification, localization, detection, and more.
- **2012** marked **the first year where a CNN was used** to achieve a top 5 test error rate of 15.3%.
- The next best entry achieved an error of 26.2%.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



AlexNet

Architecture
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

- Input: 227x227x3 images (224x224 before padding)
- First layer: 96 11x11 filters applied at stride 4

- **Output volume size?**

  (N-F)/s+1 = (227-11)/4+1 = 55 -> [55x55x96]

- **Number of parameters in this layer?**

  (11*11*3)*96 = 35K

[**Krizhevsky et al**., **2012**]

AlexNet

Architecture
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

- Input: 227x227x3 images (224x224 before padding)
- After CONV1: 55x55x96
- Second layer: 3x3 filters applied at stride 2

- **Output volume size?**

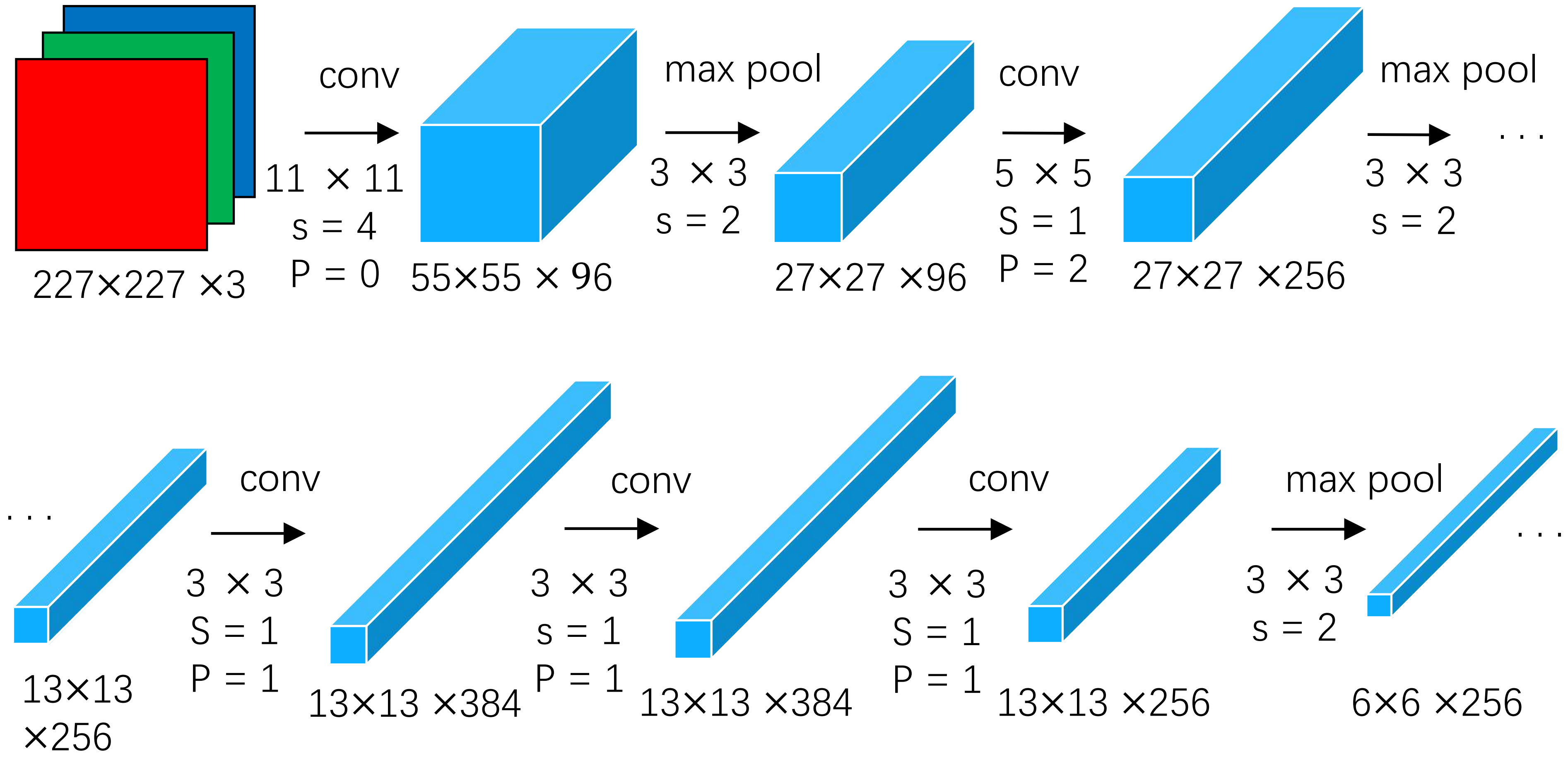
  (N-F)/s+1 = (55-3)/2+1 = 27 -> [27x27x96]

- **Number of parameters in this layer?**

  0!

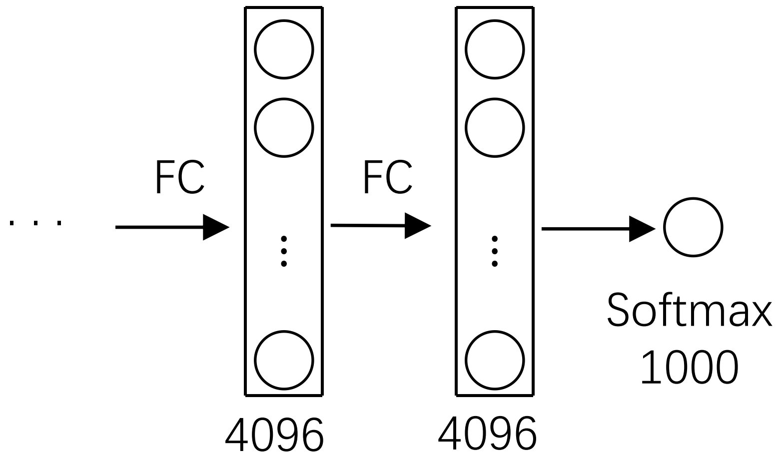
[**Krizhevsky et al**., **2012**]

AlexNet



[**Krizhevsky et al.**, **2012**]

AlexNet



[**Krizhevsky et al.**, **2012**]

AlexNet

Details/Retrospectives:

- first use of ReLU
- heavy data augmentation
- dropout 0.5
- batch size 128

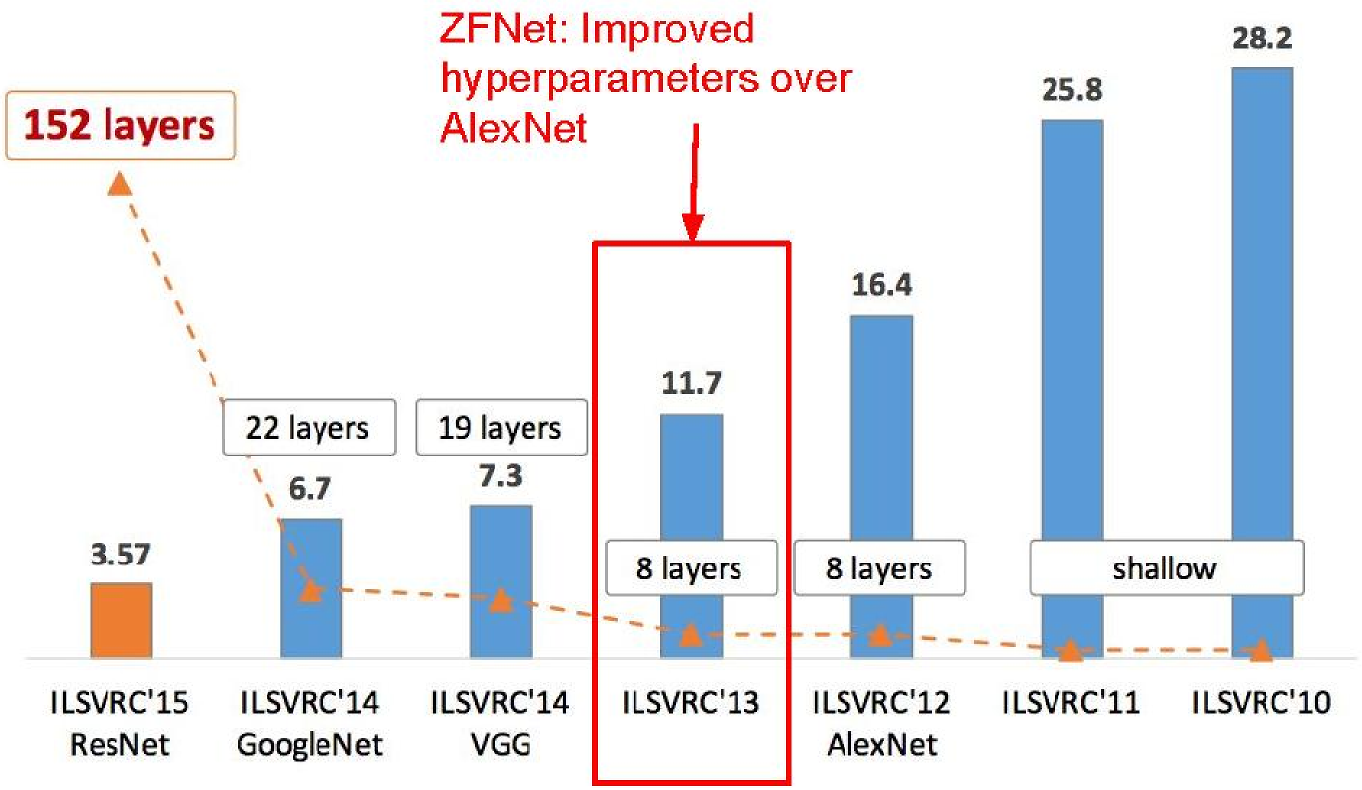
[**Krizhevsky et al.**, **2012**]

AlexNet

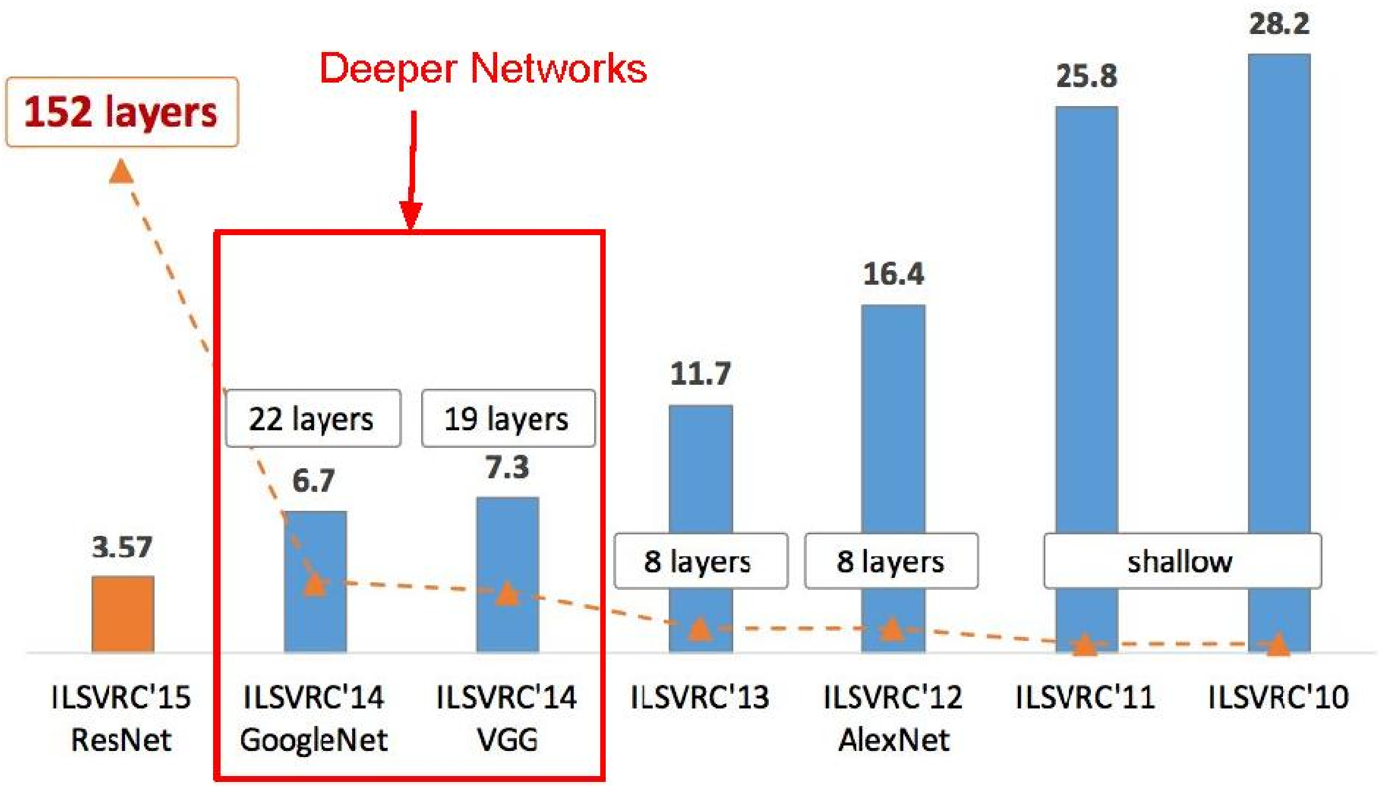
AlexNet was the coming out party for CNNs in the computer vision community. This was **the first time a model performed so well on a historically difficult ImageNet dataset**. This paper illustrated the benefits of CNNs and backed them up with record breaking performance in the competition.

[**Krizhevsky et al.**, **2012**]

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



VGGNet

- *Very Deep Convolutional Networks For Large Scale Image Recognition - Karen Simonyan and Andrew Zisserman; 2015*
- The runner-up at the ILSVRC 2014 competition
- Significantly deeper than AlexNet
- 140 million parameters

[Simonyan and Zisserman, 2014]

VGGNet

Input
3x3 conv, 64
3x3 conv, 64
Pool 1/2
3x3 conv, 128
3x3 conv, 128
Pool 1/2
3x3 conv, 256
3x3 conv, 256
Pool 1/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool 1/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool 1/2
FC 4096
FC 4096
FC 1000
Softmax

- **Smaller filters**
  Only 3x3 CONV filters, stride 1, pad 1
  and 2x2 MAX POOL , stride 2
- **Deeper network**
  AlexNet: 8 layers
  VGGNet: 16 - 19 layers
- ZFNet: 11.7% top 5 error in ILSVRC'13
- VGGNet: 7.3% top 5 error in ILSVRC'14

[Simonyan and Zisserman, 2014]

VGGNet

- **Why use smaller filters? (3x3 conv)**

Stack of three 3x3 conv (stride 1) layers has the same effective receptive field as one 7x7 conv layer.

- **What is the effective receptive field of three 3x3 conv (stride 1) layers?**

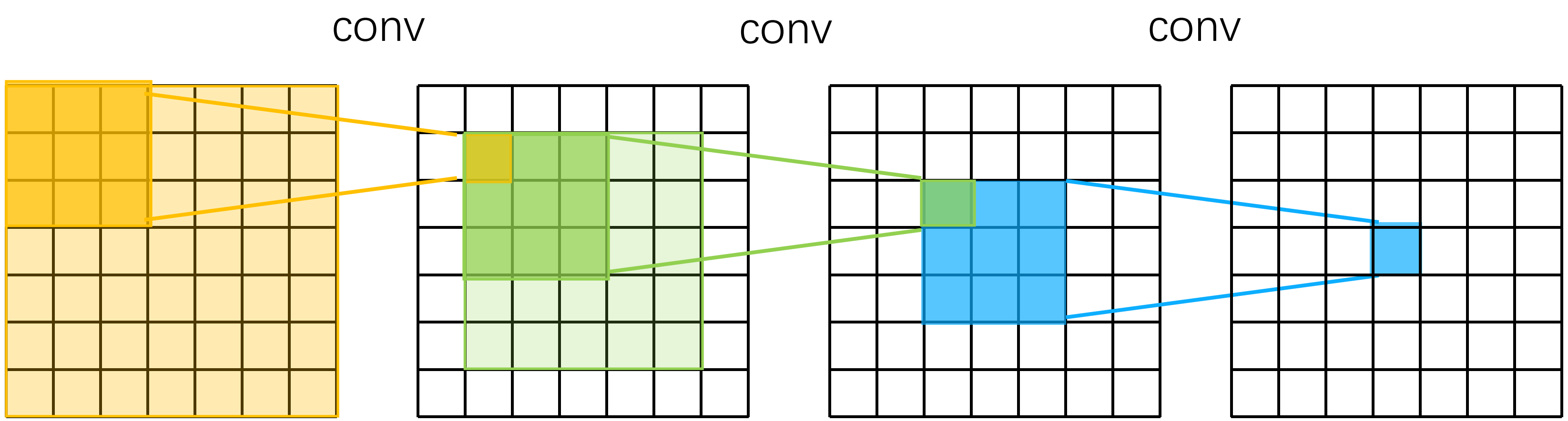
7x7

But deeper, more non-linearities

And fewer parameters: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer

[Simonyan and Zisserman, 2014]

Reminder: Receptive Field



Input	memory: 224*224*3=150K	params: 0
3x3 conv, 64	memory: 224*224*64=3.2M	params: (3*3*3)*64 = 1,728
3x3 conv, 64	memory: 224*224*64=3.2M	params: (3*3*64)*64 = 36,864
Pool	memory: 112*112*64=800K	params: 0
3x3 conv, 128	memory: 112*112*128=1.6M	params: (3*3*64)*128 = 73,728
3x3 conv, 128	memory: 112*112*128=1.6M	params: (3*3*128)*128 = 147,456
Pool	memory: 56*56*128=400K	params: 0
3x3 conv, 256	memory: 56*56*256=800K	params: (3*3*128)*256 = 294,912
3x3 conv, 256	memory: 56*56*256=800K	params: (3*3*256)*256 = 589,824
3x3 conv, 256	memory: 56*56*256=800K	params: (3*3*256)*256 = 589,824
Pool	memory: 28*28*256=200K	params: 0
3x3 conv, 512	memory: 28*28*512=400K	params: (3*3*256)*512 = 1,179,648
3x3 conv, 512	memory: 28*28*512=400K	params: (3*3*512)*512 = 2,359,296
3x3 conv, 512	memory: 28*28*512=400K	params: (3*3*512)*512 = 2,359,296
Pool	memory: 14*14*512=100K	params: 0
3x3 conv, 512	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
3x3 conv, 512	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
3x3 conv, 512	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
Pool	memory: 7*7*512=25K	params: 0
FC 4096	memory: 4096	params: 7*7*512*4096 = 102,760,448
FC 4096	memory: 4096	params: 4096*4096 = 16,777,216
FC 1000	memory: 1000	params: 4096*1000 = 4,096,000

[Simonyan and Zisserman, 2014]

VGGNet

Input
3x3 conv, 64
3x3 conv, 64
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
FC 4096
FC 4096
FC 1000
Softmax

VGG16:

TOTAL memory: 24M * 4 bytes ~= 96MB / image

TOTAL params: 138M parameters

[Simonyan and Zisserman, 2014]

VGGNet

Details/Retrospectives :

- ILSVRC'14 2nd in classification, 1st in localization
- Similar training procedure as AlexNet
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- FC7 features generalize well to other tasks

[Simonyan and Zisserman, 2014]

VGGNet

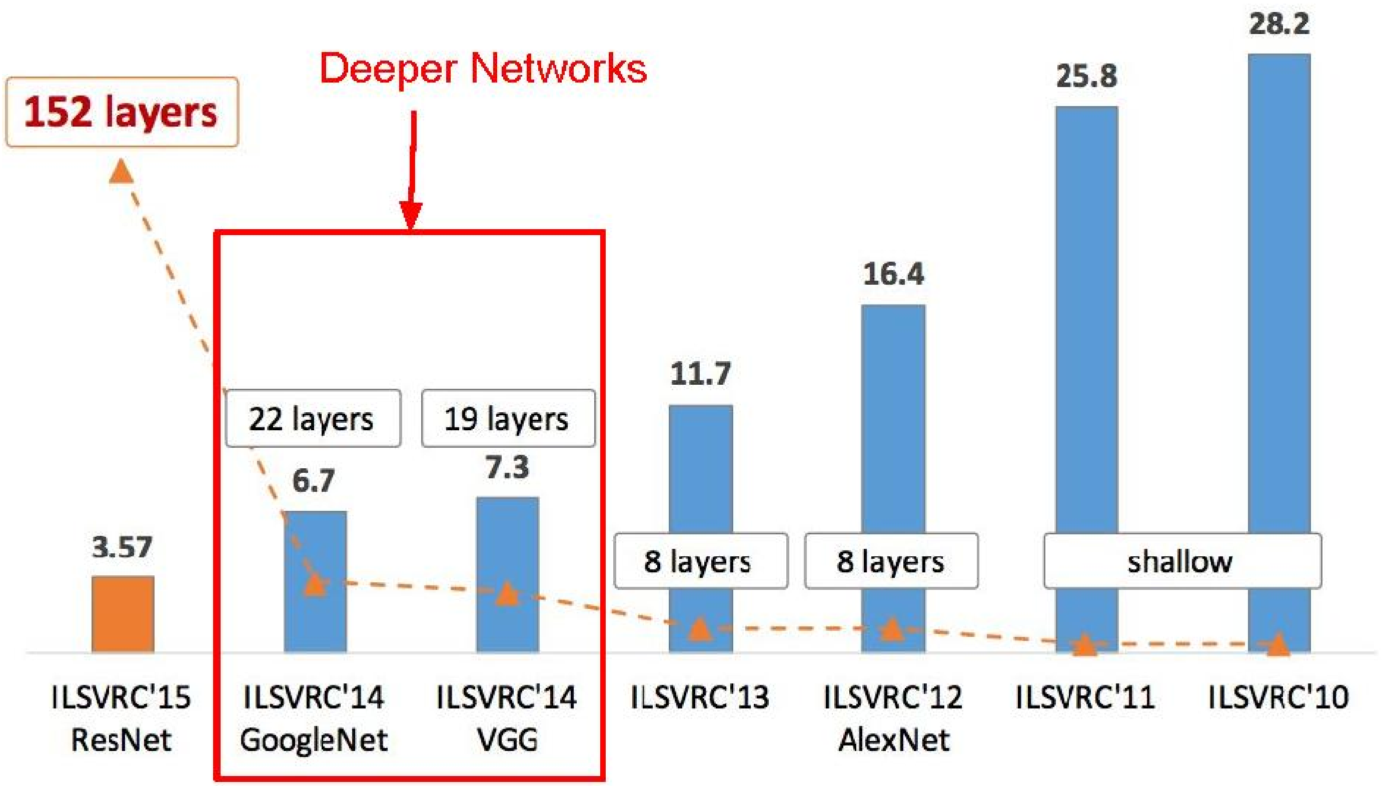
VGG Net reinforced the notion that **convolutional neural networks have to have a deep network of layers in order for this hierarchical representation of visual data to work**.

Keep it deep.

Keep it simple.

[Simonyan and Zisserman, 2014]

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

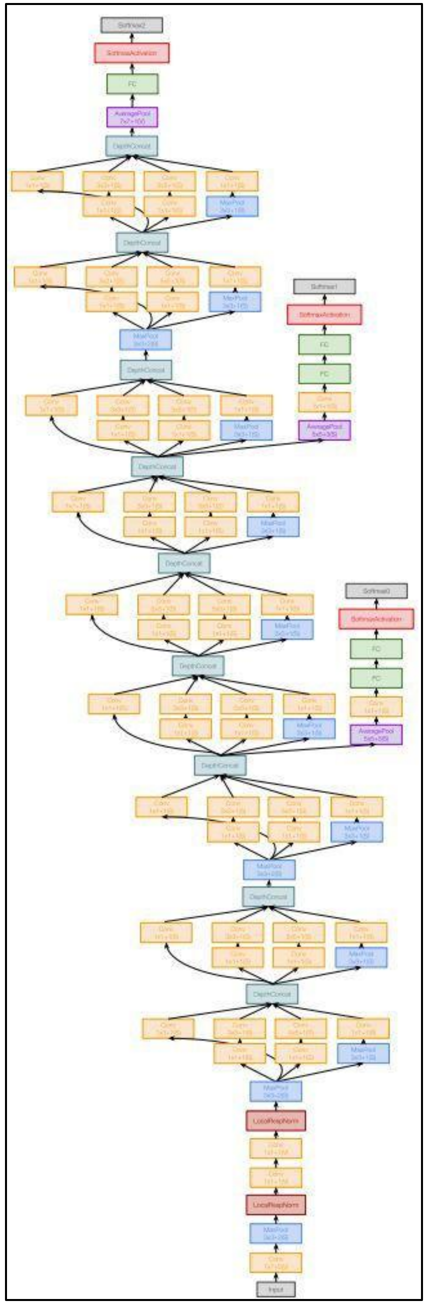


GoogleNet

- *Going Deeper with Convolutions - Christian Szegedy et al.; 2015*
- ILSVRC 2014 competition winner
- Also significantly deeper than AlexNet
- x12 less parameters than AlexNet
- Focused on computational efficiency

[Szegedy et al., 2014]

GoogleNet

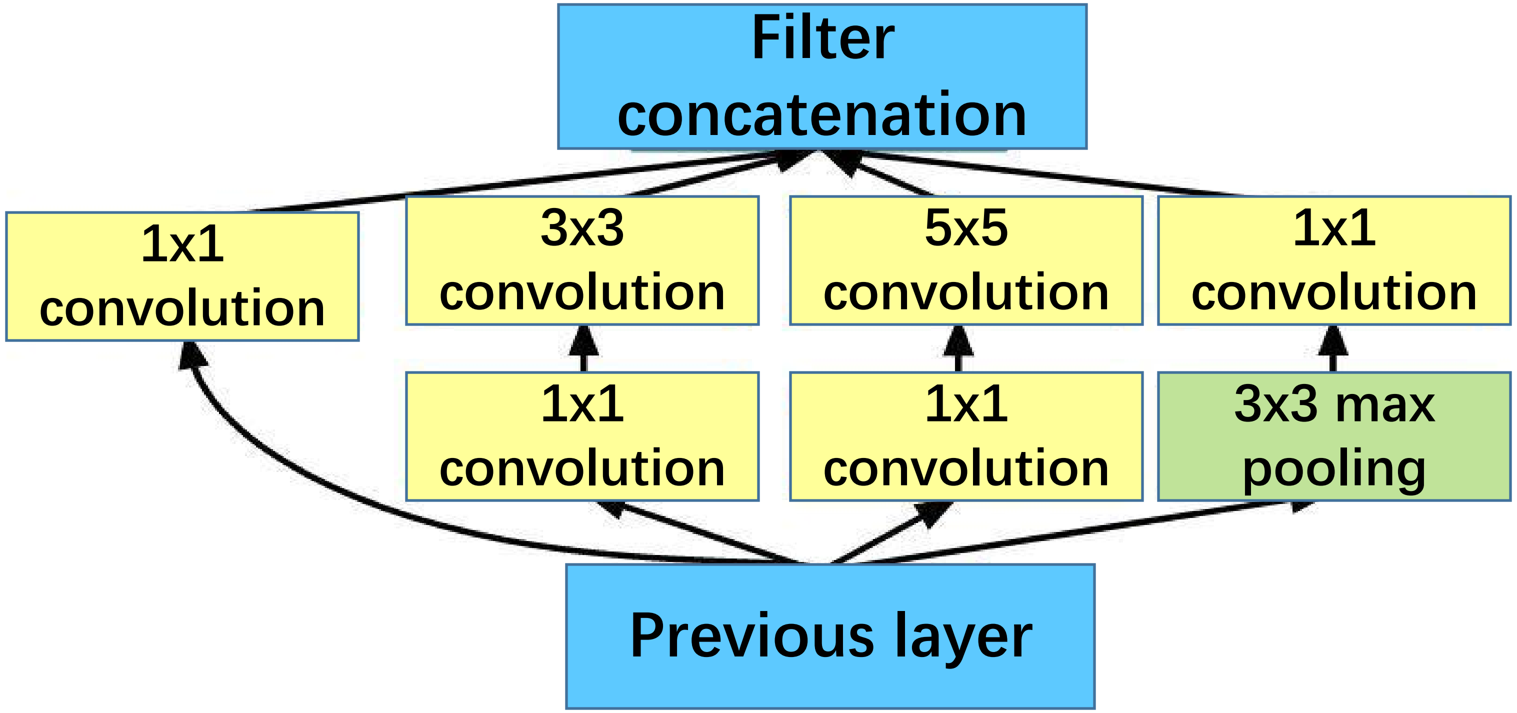


- 22 layers
- Efficient **"Inception" module** - strayed from the general approach of simply stacking conv and pooling layers on top of each other in a sequential structure
- No FC layers
- Only 5 million parameters!
- ILSVRC'14 classification winner (6.7% top 5 error)

[Szegedy et al., 2014]

GoogleNet

“**Inception module”:** design a good local network topology (network within a network) and then stack these modules on top of each other

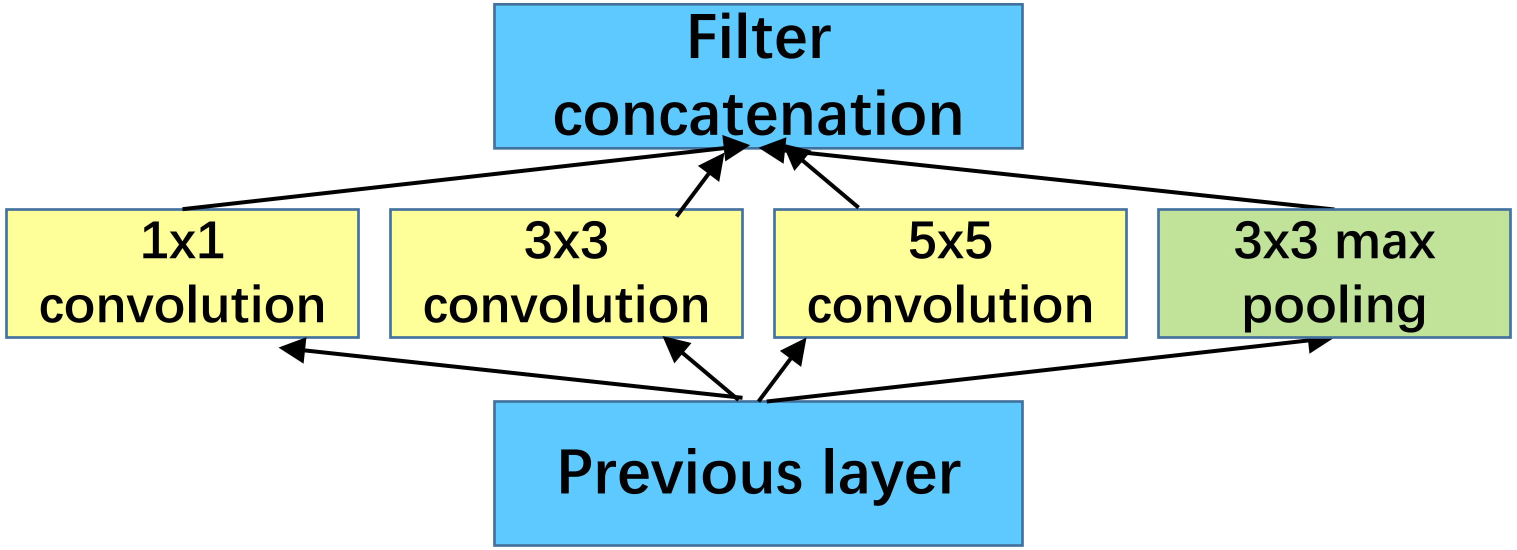


[Szegedy et al., 2014]

GoogleNet

Naïve Inception Model

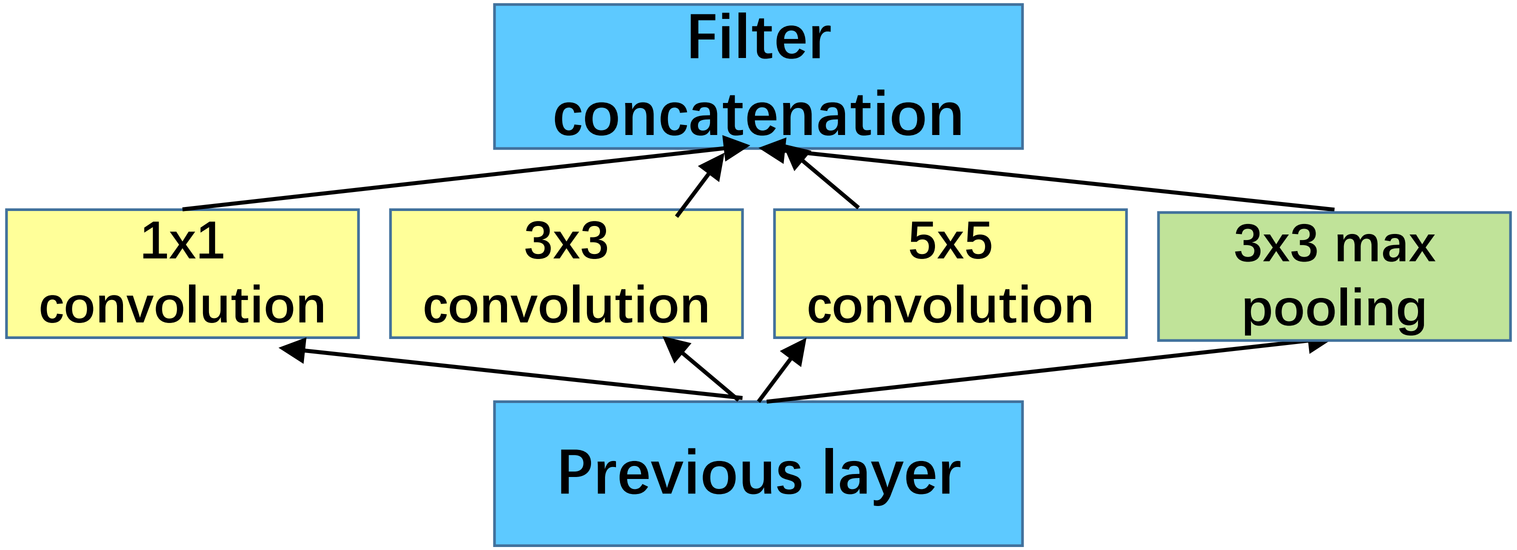
- Apply parallel filter operations on the input :
  - Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
  - Pooling operation (3x3)
- Concatenate all filter outputs together depth-wise



[Szegedy et al., 2014]

GoogleNet

- What's the problem with this?
  High computational complexity



[Szegedy et al., 2014]

GoogleNet

- **Output volume sizes:**

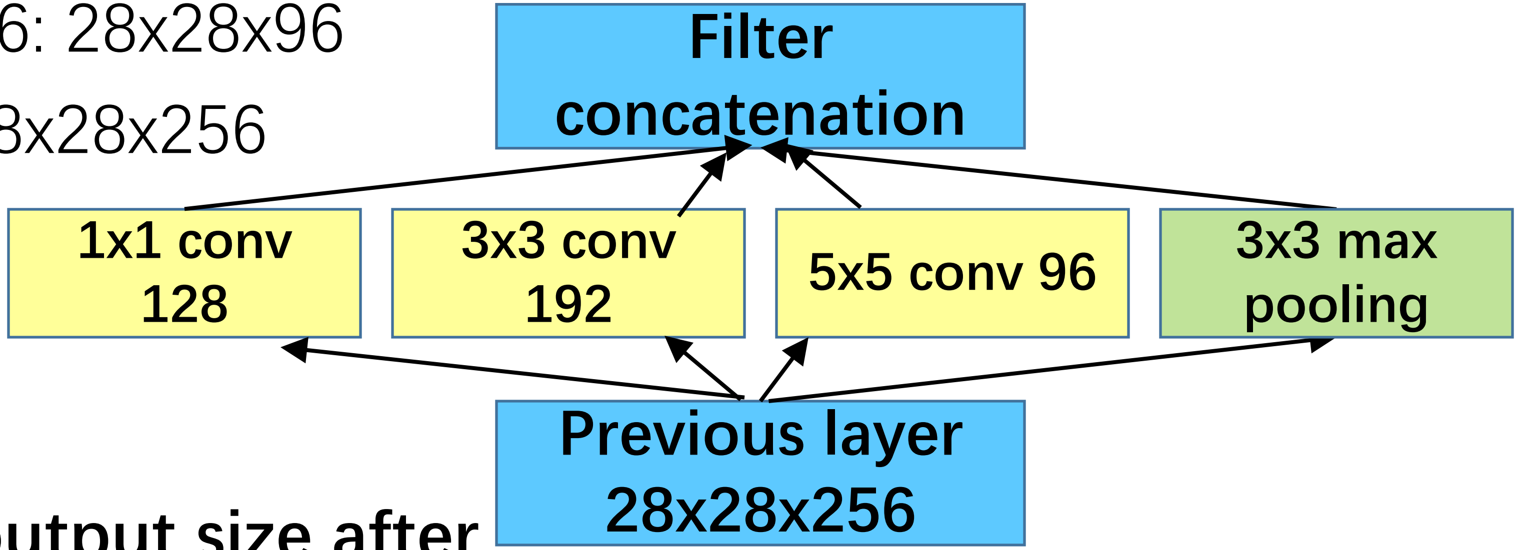
1x1 conv, 128: 28x28x128

3x3 conv, 192: 28x28x192

5x5 conv, 96: 28x28x96

3x3 pool: 28x28x256

Example:



- **What is output size after filter concatenation?**

28x28x(128+192+96+256) = 28x28x672

[Szegedy et al., 2014]

GoogleNet

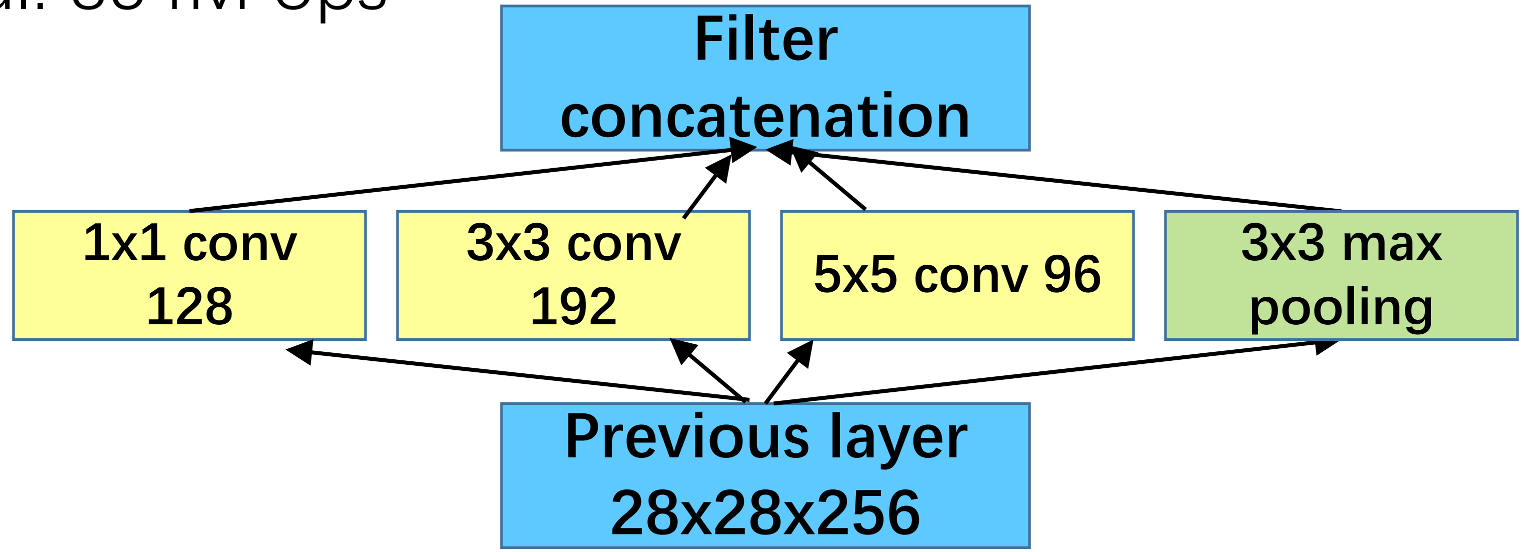
- **Number of convolution operations:**

1x1 conv, 128: 28x28x128x1x1x256

3x3 conv, 192: 28x28x192x3x3x256

5x5 conv, 96: 28x28x96x5x5x256

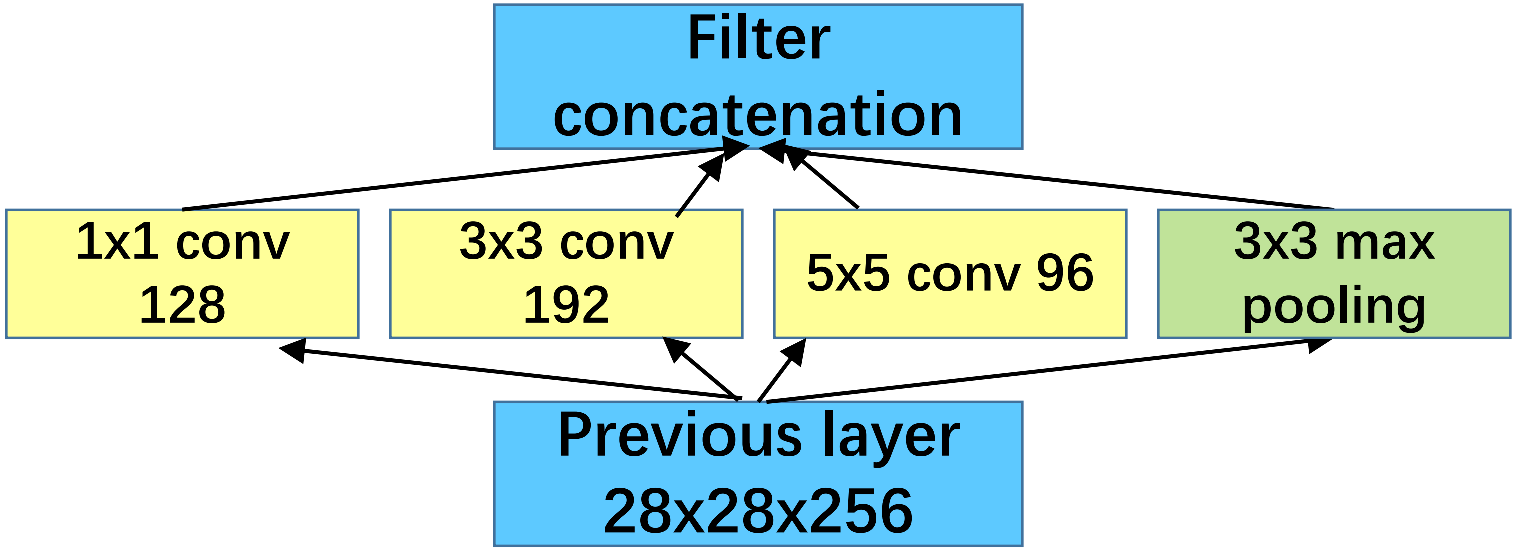
Total: 854M ops



[Szegedy et al., 2014]

GoogleNet

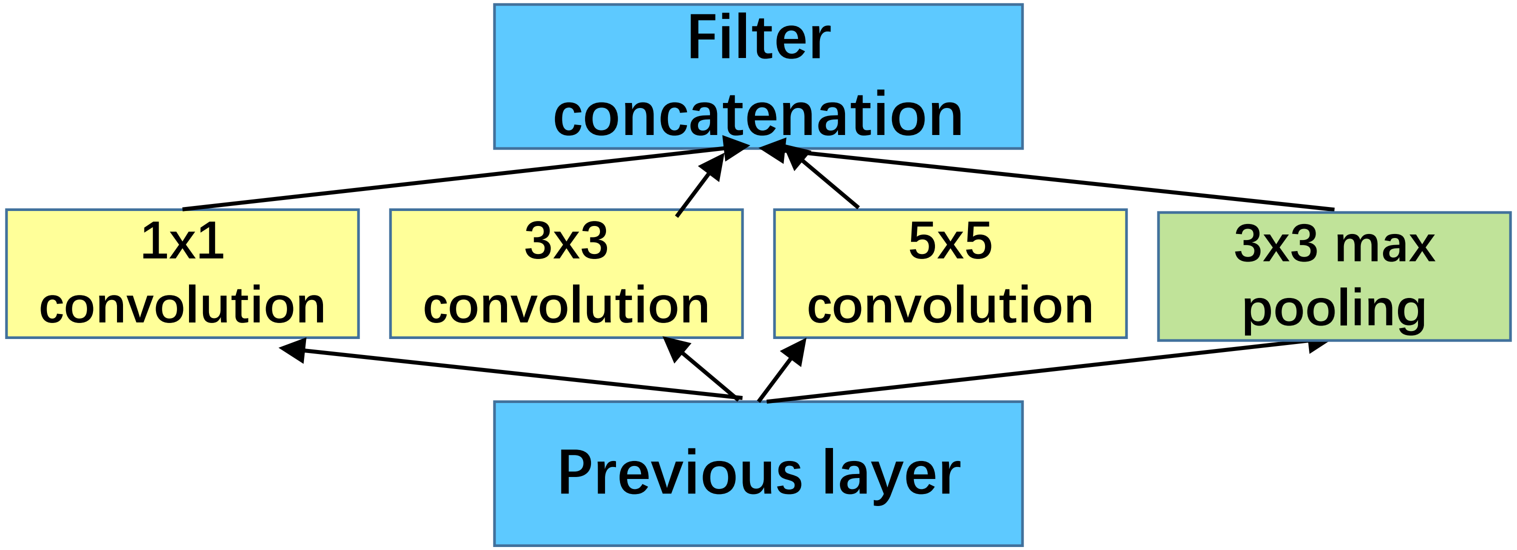
- **Very expensive compute!**
- Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer.



[Szegedy et al., 2014]

GoogleNet

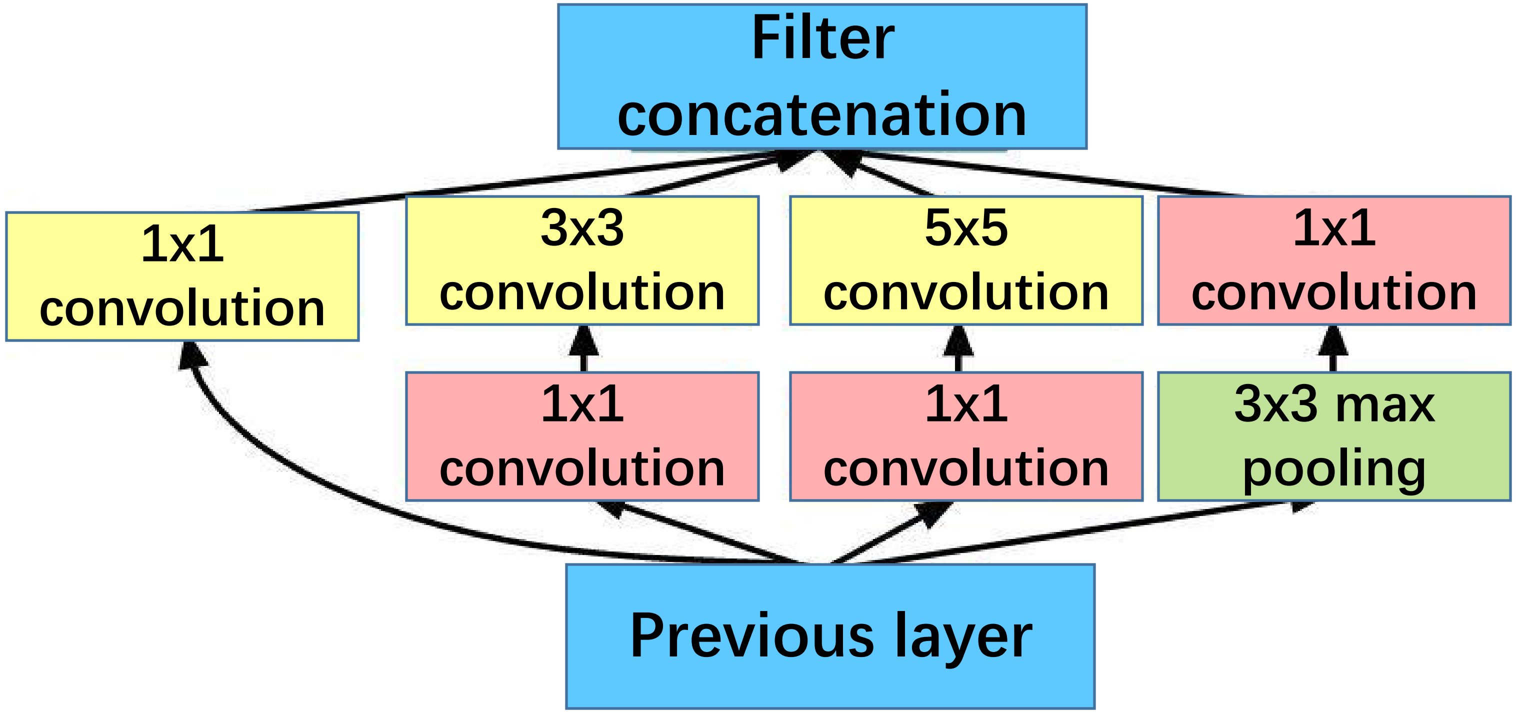
- **Solution:** “bottleneck” layers that use 1x1 convolutions to reduce feature depth (from previous hour).



[Szegedy et al., 2014]

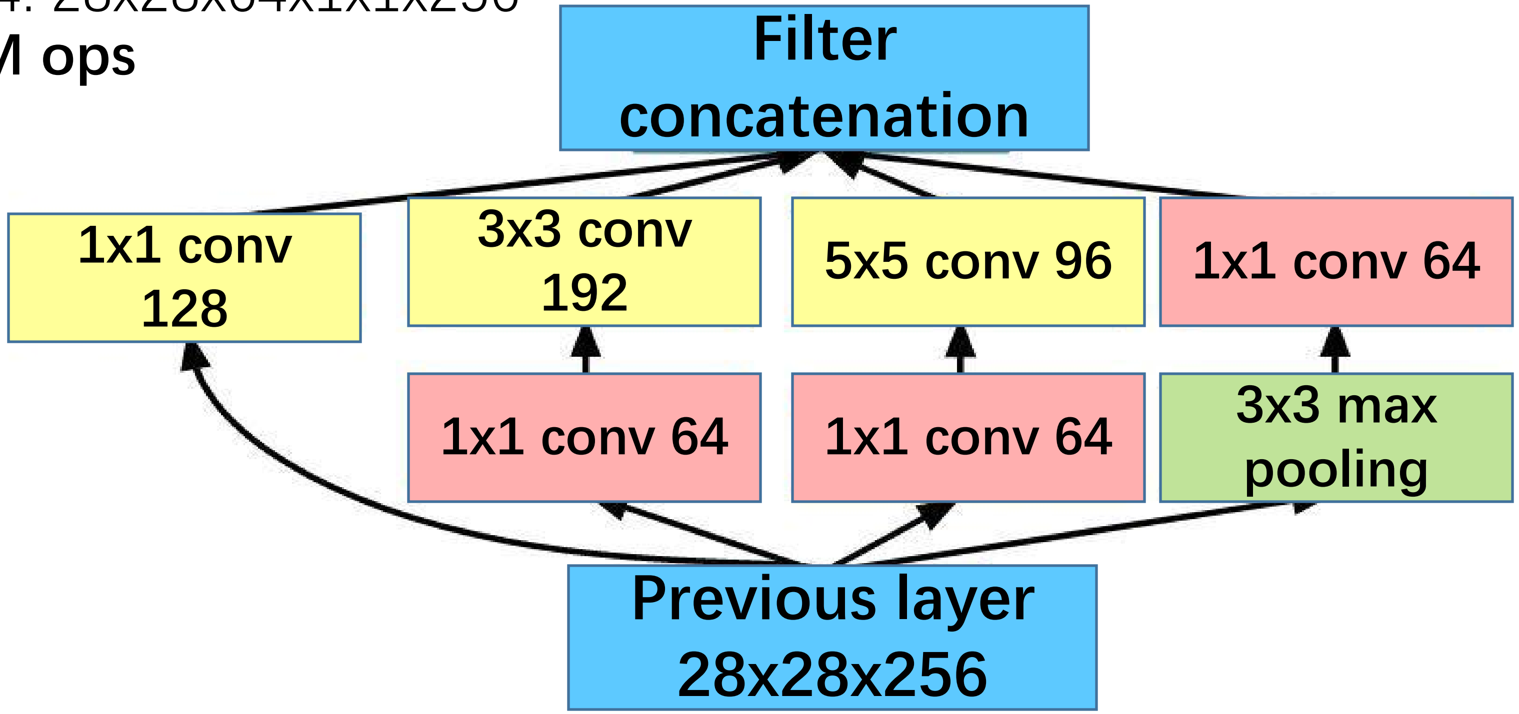
GoogleNet

- **Solution:** "bottleneck" layers that use 1x1 convolutions to reduce feature depth (from previous hour).



[Szegedy et al., 2014]

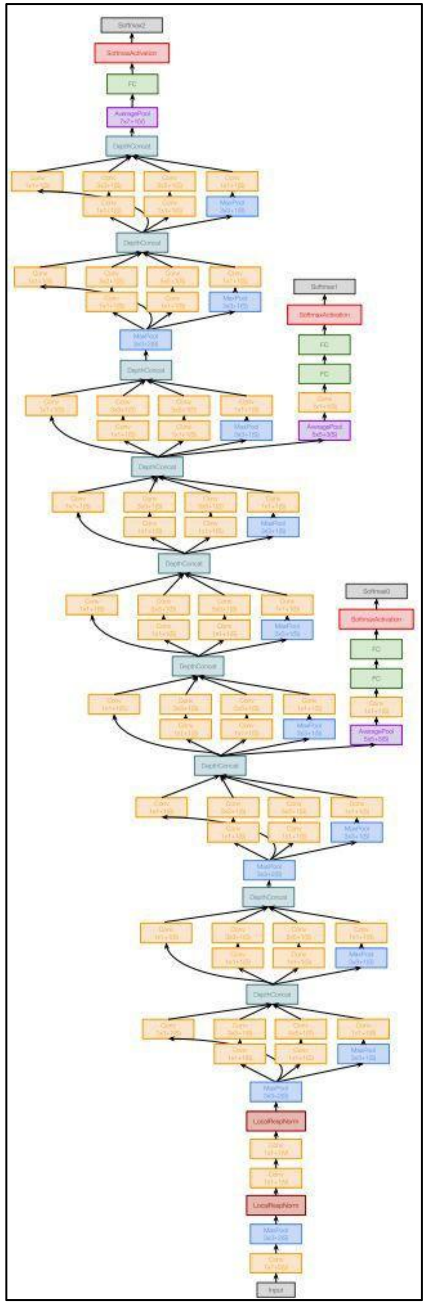
- **Number of convolution operations:**
  1x1 conv, 64: 28x28x64x1x1x256
  1x1 conv, 64: 28x28x64x1x1x256
  1x1 conv, 128: 28x28x128x1x1x256
  3x3 conv, 192: 28x28x192x3x3x64
  5x5 conv, 96: 28x28x96x5x5x264
  1x1 conv, 64: 28x28x64x1x1x256
  Total: 353M ops



- Compared to 854M ops for naive version

[Szegedy et al., 2014]

GoogleNet



Details/Retrospectives :

- Deeper networks, with computational efficiency
- 22 layers
- Efficient “Inception” module
- No FC layers
- 12x less params than AlexNet
- ILSVRC’14 classification winner (6.7% top 5 error)

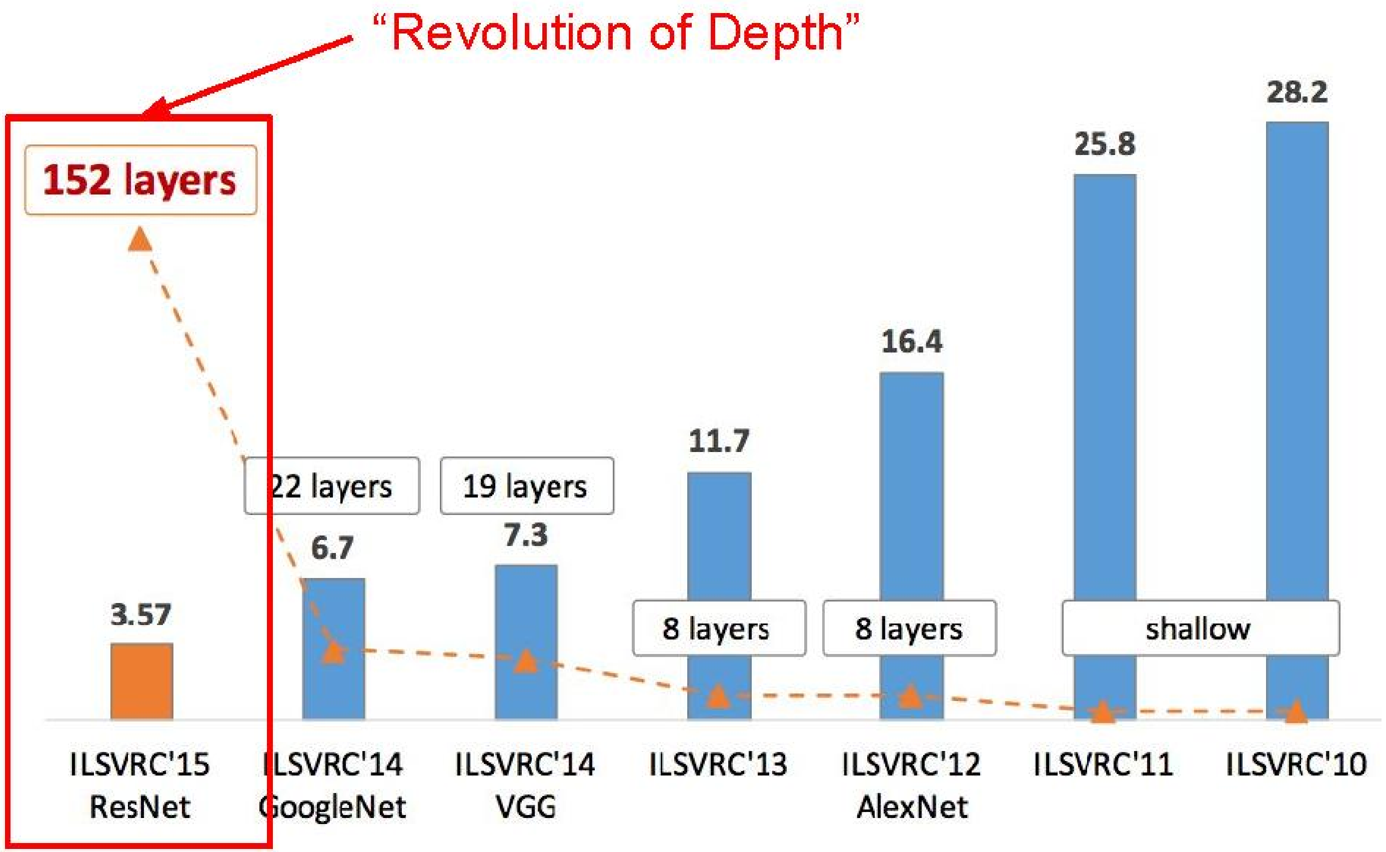
[Szegedy et al., 2014]

GoogleNet

Introduced the idea that CNN layers **didn’t always have to be stacked up sequentially**. Coming up with the Inception module, the authors showed that a creative structuring of layers can lead to improved performance and **computationally efficiency**.

[Szegedy et al., 2014]

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

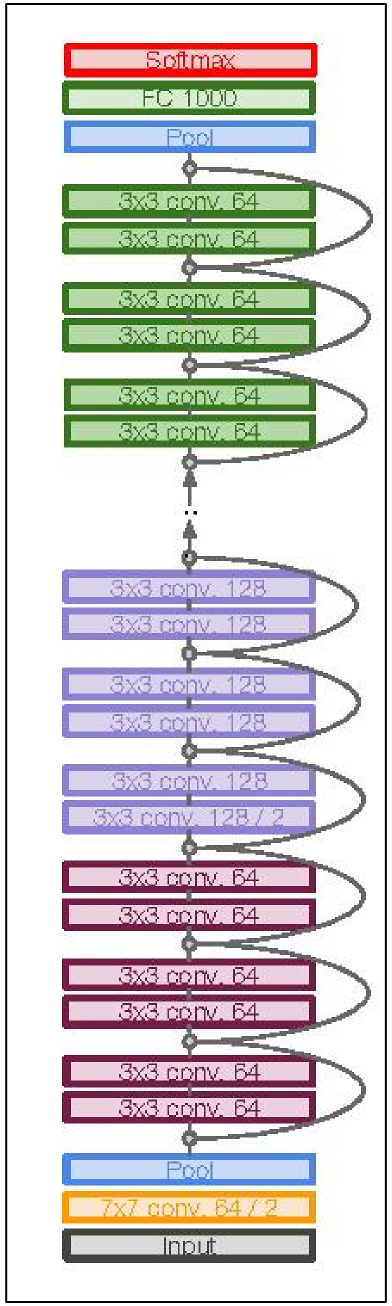


ResNet

- *Deep Residual Learning for Image Recognition - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun; 2015*
- Extremely deep network – 152 layers
- Deeper neural networks are more difficult to train.
- Deep networks suffer from vanishing and exploding gradients.
- Present a residual learning framework to ease the training of networks that are substantially deeper than those used previously.

[He et al., 2015]

ResNet



- ILSVRC'15 classification winner (3.57% top 5 error, humans generally hover around a 5-10% error rate)
  Swept all classification and detection competitions in ILSVRC'15 and COCO'15!

[He et al., 2015]

ResNet

- What happens when we continue stacking deeper layers on a convolutional neural network?

- 56-layer model performs worse on both training and test error

-> The deeper model performs worse (not caused by overfitting)!

[He et al., 2015]

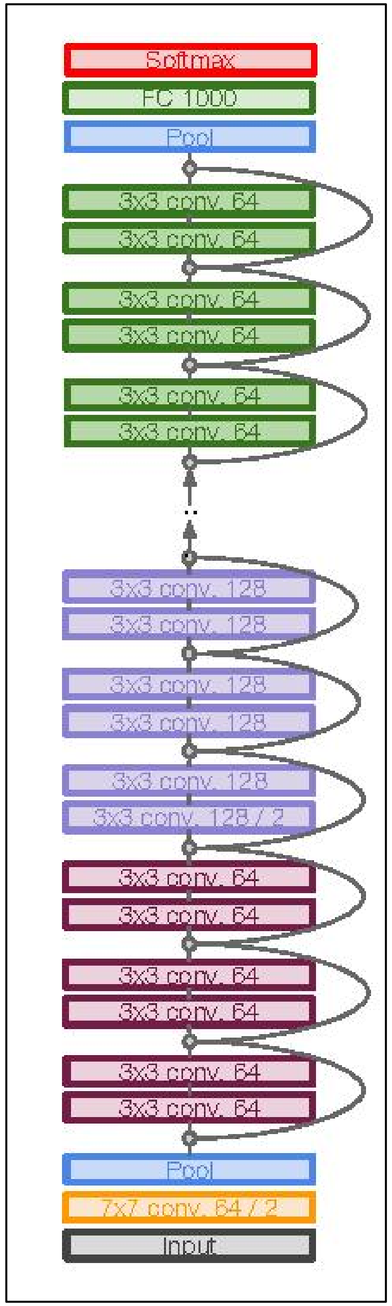
ResNet

- **Hypothesis**: The problem is an optimization problem. Very deep networks are harder to optimize.
- **Solution**: Use network layers to fit residual mapping instead of directly trying to fit a desired underlying mapping.

- We will use **skip connections** allowing us to take the activation from one layer and feed it into another layer, much deeper into the network.
- Use layers to fit residual F(x) = H(x) – x
instead of H(x) directly

[He et al., 2015]

ResNet

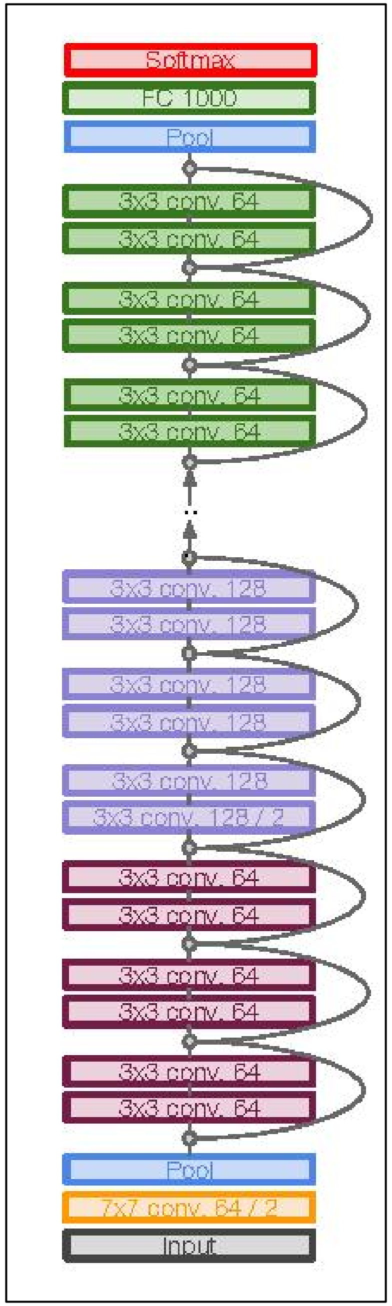


Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)

[He et al., 2015]

ResNet



- Total depths of 34, 50, 101, or 152 layers for ImageNet
- For deeper networks (ResNet-50+), use "bottleneck" layer to improve efficiency (similar to GoogLeNet)

[He et al., 2015]

ResNet

Experimental Results:

- Able to train very deep networks without degrading
- Deeper networks now achieve lower training errors as expected

[He et al., 2015]

ResNet

ResNet is among the **best** CNN architecture that we currently have and is a great innovation for the idea of residual learning. Even better than human performance!

[He et al., 2015]

Code Examples

```
%% 创建简单的CNN网络以用于图像分类
openExample('nnet/TrainABasicConvolutionalNeuralNetworkForClassificationExample')

%% 使用GoogLeNet对网络摄像头图像进行分类
openExample('nnet/ClassifyImagesFromWebcamUsingDeepLearningExample')

%% 使用预训练GoogLeNet对新图像进行分类（迁移学习）
openExample('nnet/TransferLearningUsingGoogLeNetExample')
```