Deep Learning

Dr. Santosh Kumar Vipparthi
Dept. of CSE

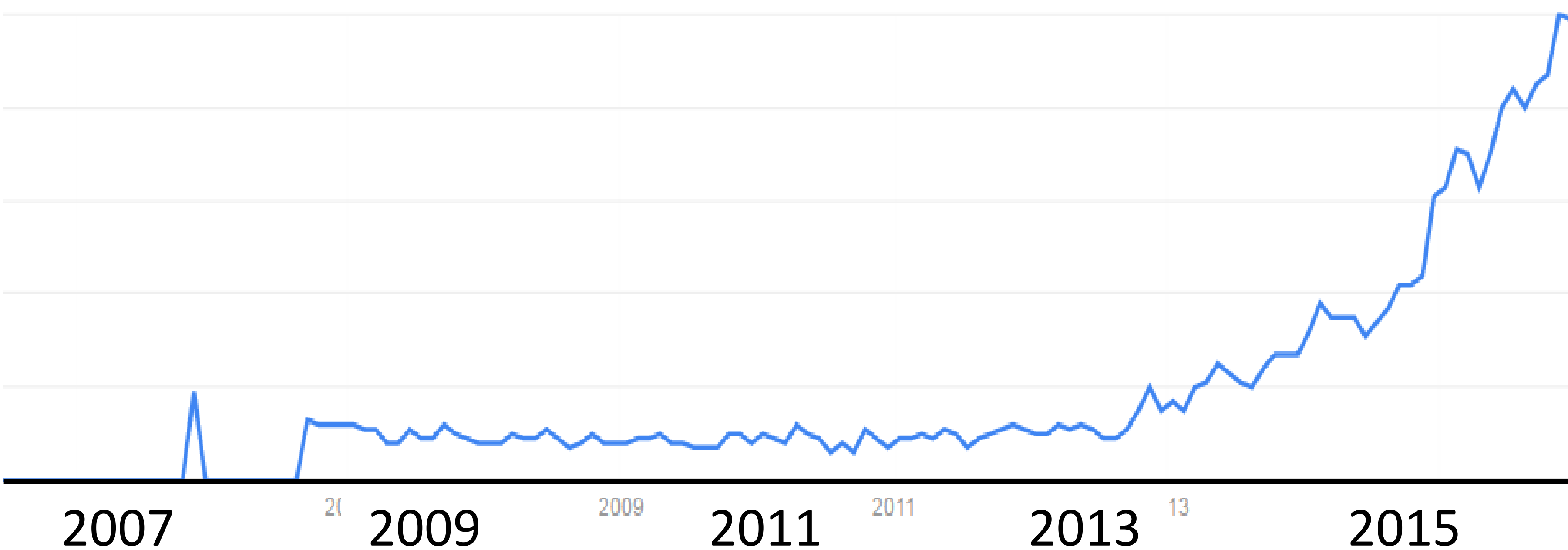
Based on notes from Andrej Karpathy, Fei-Fei Li, Justin Johnson, Hung-yi Lee

Introduction

DL attracts lots of attention.

- Google Trends

Deep learning obtains many exciting results.



Outline

Introduction of Deep Learning

Why Deep?

Tips for Training Deep Neural Network

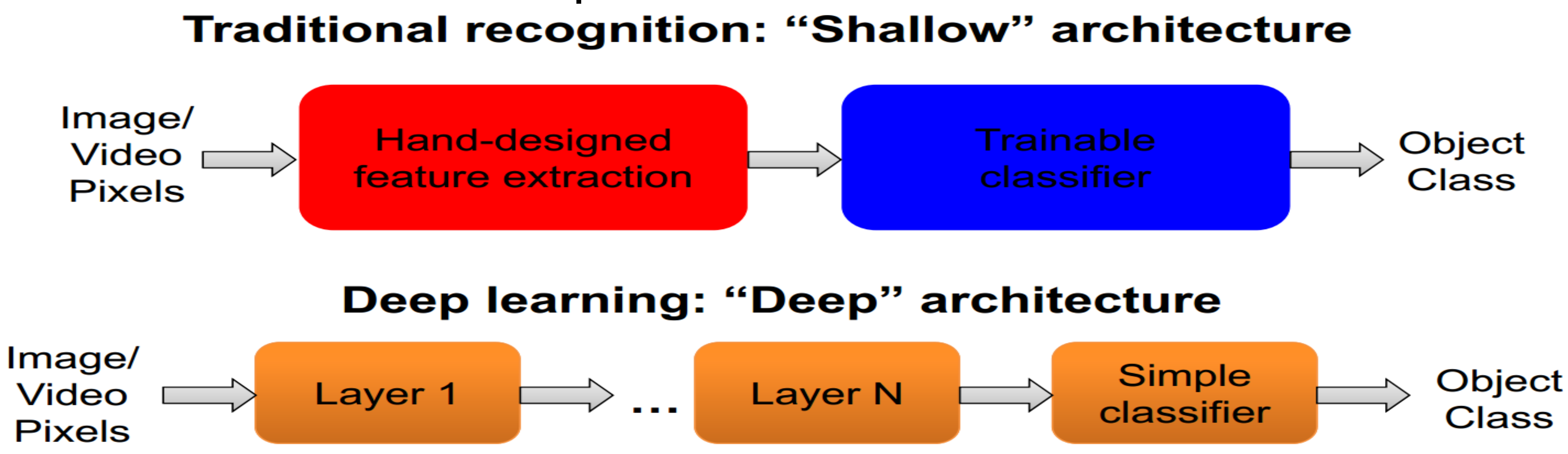
What is Deep Learning (DL) ?

A machine learning subfield of learning **representations** of data. Exceptional effective at **learning patterns**.

Deep learning algorithms attempt to learn (multiple levels of) representation by using a **hierarchy of multiple layers**

If you provide the system **tons of information**, it begins to understand it and respond in useful ways.

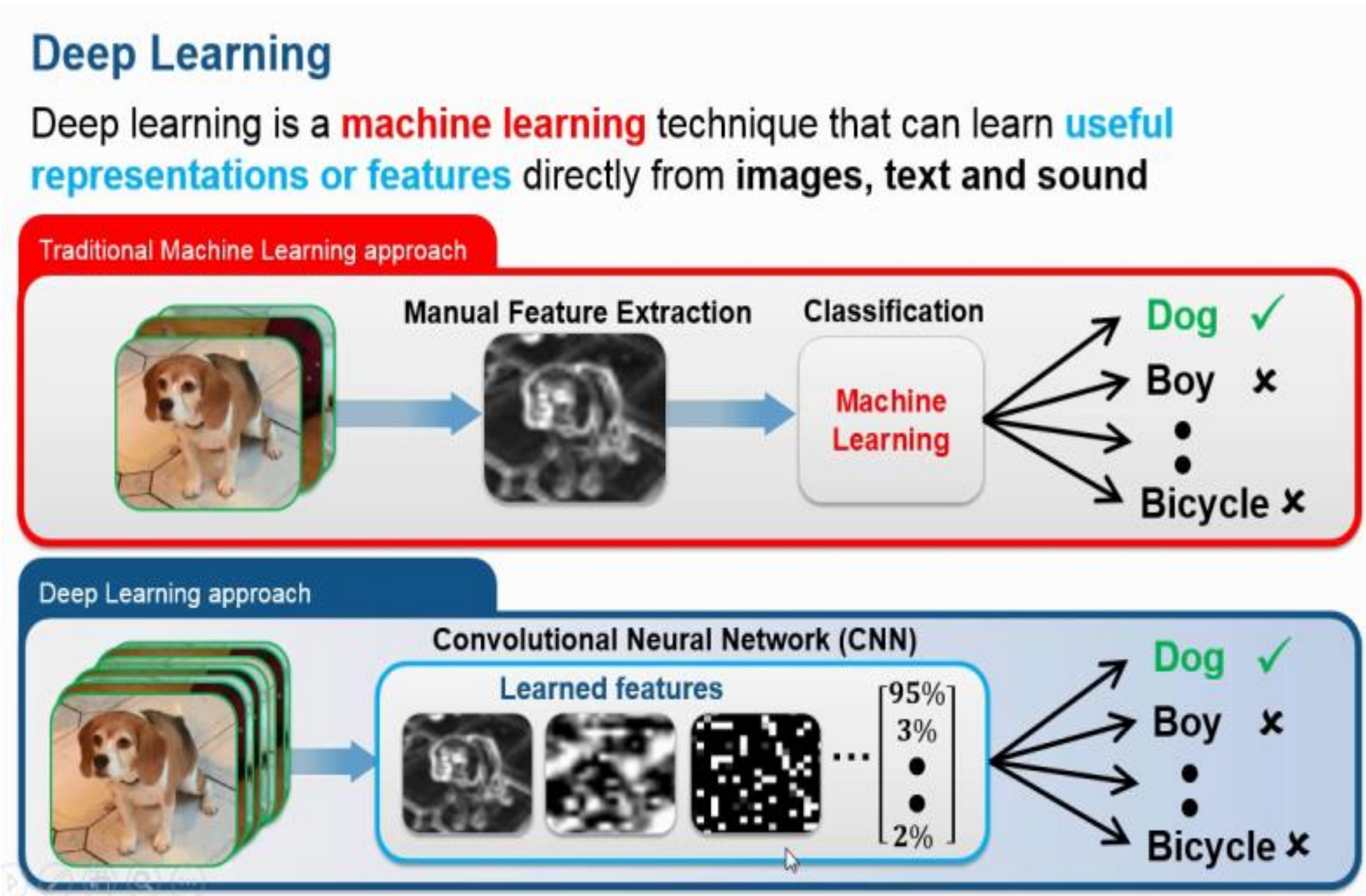
- "Shallow" vs. "deep" architectures



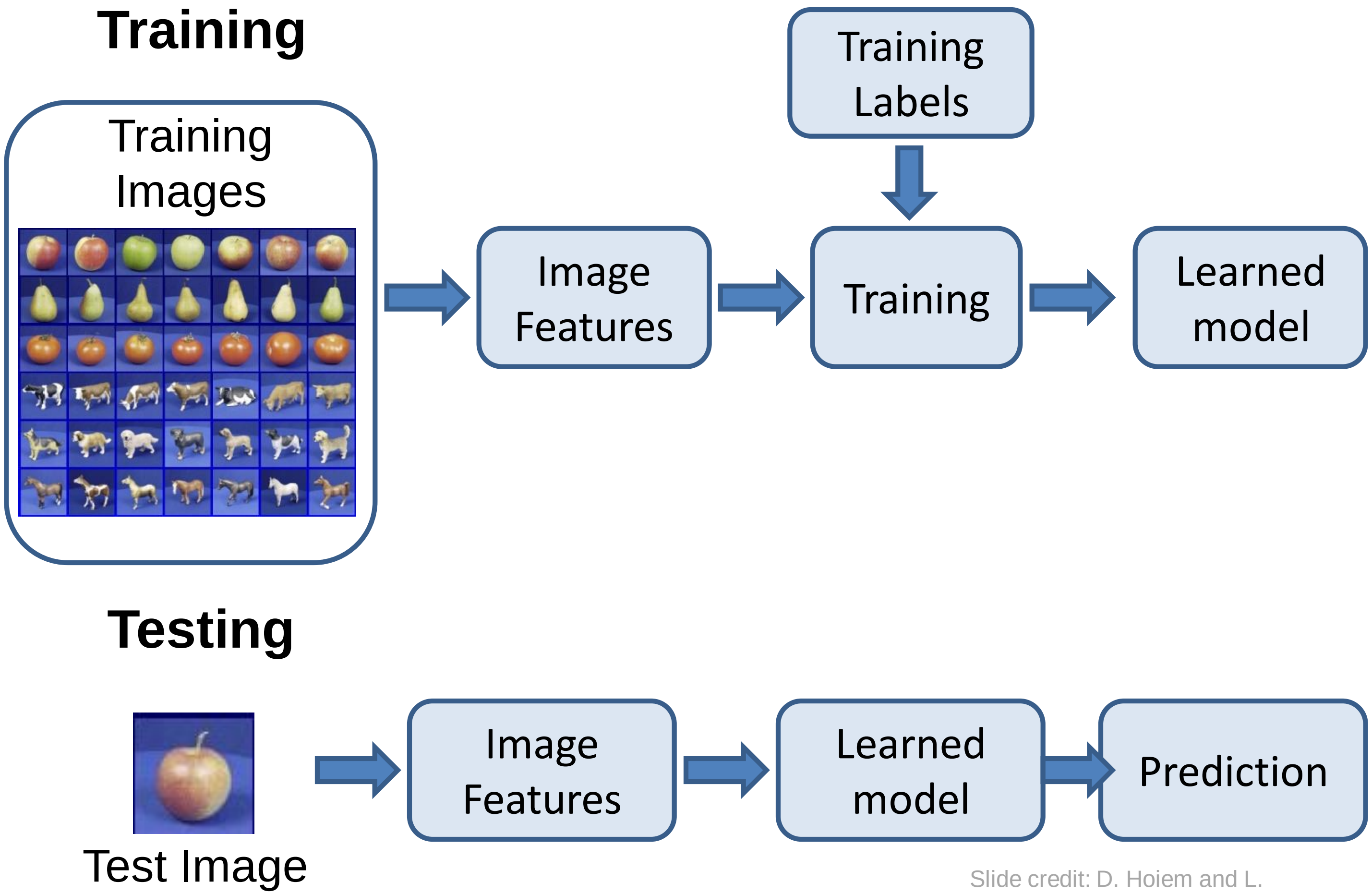
Learn a *feature hierarchy* all the way from pixels to classifier

reference : http://web.engr.illinois.edu/~slazebni/spring14/lec24_cnn.pdf

Example


Deep Learning
Deep learning is a machine learning technique that can learn useful representations or features directly from images, text and sound
Traditional Machine Learning approach
Manual Feature Extraction
Classification
Machine Learning
Dog ✓
Boy ×
Bicycle ×
Deep Learning approach
Convolutional Neural Network (CNN)
Learned features
95%
3%
2%
Dog ✓
Boy ×
Bicycle ×

Steps



Slide credit: D. Hoiem and L. Lazebnik

Quantitative Analysis

TABLE II
recognition accuracy comparison on MMI dataset

Methods	6-Class Exp.	7-Class Exp.
LBP [9]	76.5	81.7
Two-Phase [10]	75.4	82.0
LDP [11]	80.5	84.0
LDN [12]	80.5	83.0
LDTexP [13]	83.4	86.0
LDTerP [14]	80.6	80.0
Spatio-Temopral* [25]	81.2	-
QUEST	**83.05**	**84.0**

TABLE III
recognition accuracy comparison on GEMEP-FERA dataset

Methods	5-Class Exp.	6-Class Exp.
LBP [9]	92.2	87.8
Two-Phase [10]	88.6	85.0
LDP [11]	94.0	90.0
LDN [12]	93.4	91.0
LDTexP [13]	94.0	91.8
QUEST	**94.3**	**91.33**

Why is DL useful?

- Manually designed features are often **over-specified**, **incomplete** and take a **long time to design** and validate
- Learned Features are **easy to adapt**, **fast** to learn
- Deep learning provides a very **flexible**, (almost?) **universal**, learnable framework for representing world, visual and linguistic information.
- Can learn both unsupervised and supervised
- Effective **end-to-end** joint system learning
- Utilize large amounts of training data

Artificial Intelligence –
Deep Learning and its applications

- Trend Prediction
- Recognition
- New Knowledge
- Making Sense
- Replacing Human

Artificial Intelligence –
Deep Learning and its applications

- Information retrieval (search engines)
- Pattern recognition
- Audience targeting
- Sentiment analysis (based on written text)
- Personalization
- Automation
- Natural Language Processing
- Social media mining
- Organic search and content performance
- Brand and product differentiation

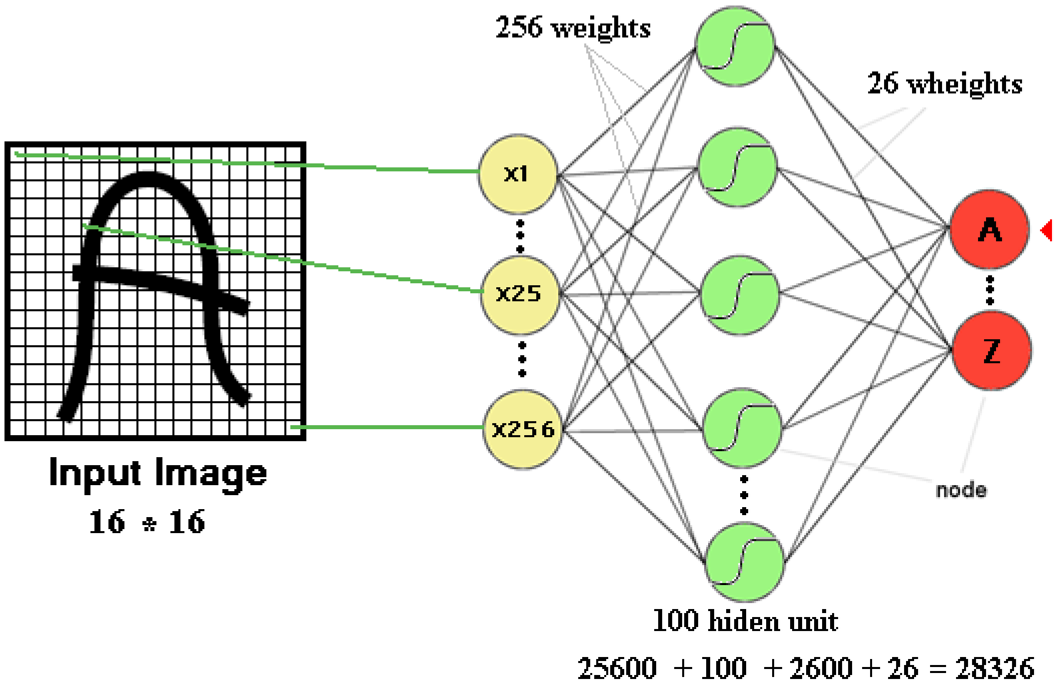
Artificial Intelligence –
Deep Learning and its applications

- Language Translation
- Speech Recognition
- Generating Handwriting
- Face Recognition
- Autonomous Driving
- Generating Arts
- Imitating Famous Painters
- Generating Music
- Generating Photos

Deep Learning: Convolutional Neural Networks

Drawbacks of Neural Networks

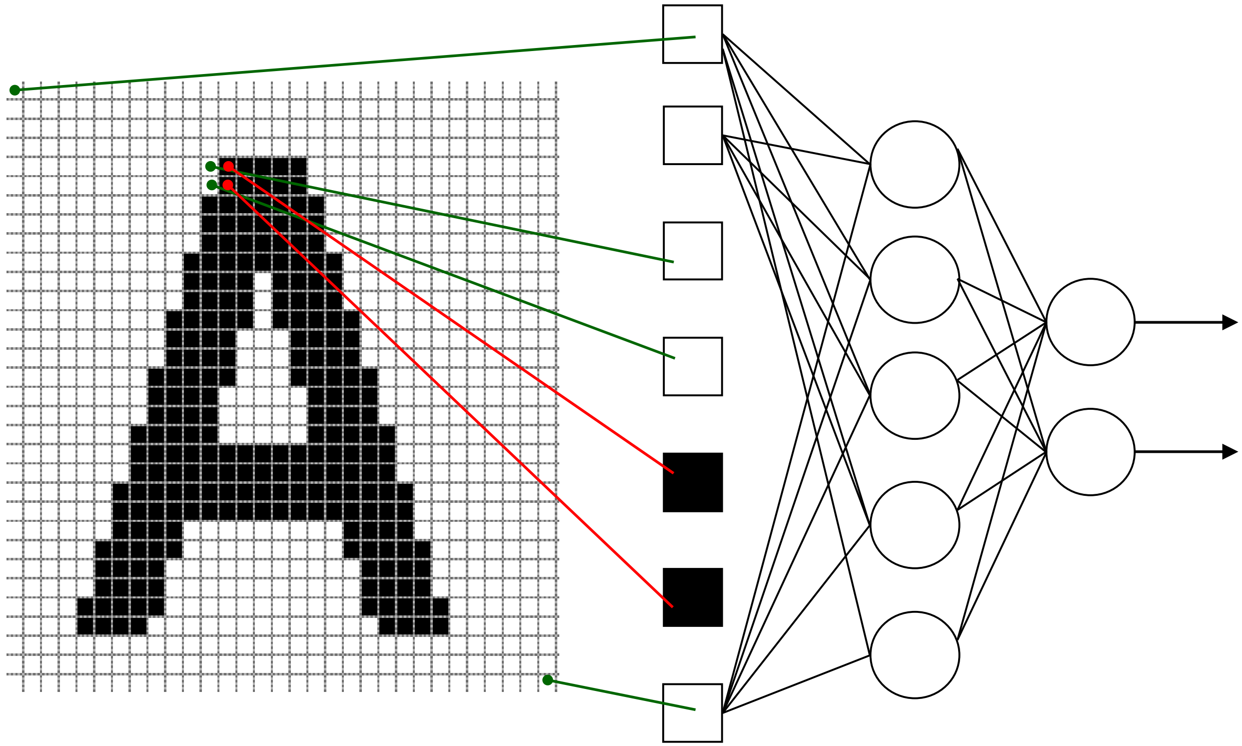
- ❑ The number of trainable parameters becomes extremely large.



Deep Learning: Convolutional Neural Networks

Drawbacks of Neural Networks

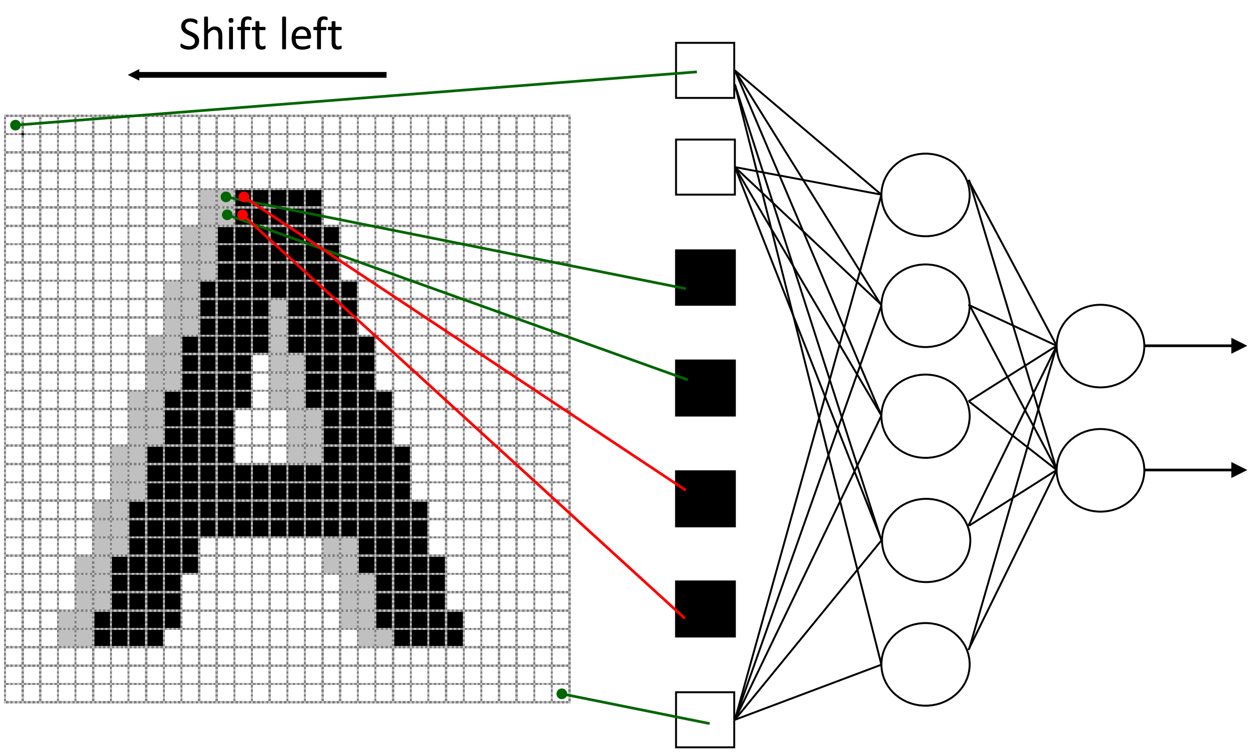
- Little or no invariance to shifting, scaling, and other forms of distortion



Deep Learning: Convolutional Neural Networks

Drawbacks of Neural Networks

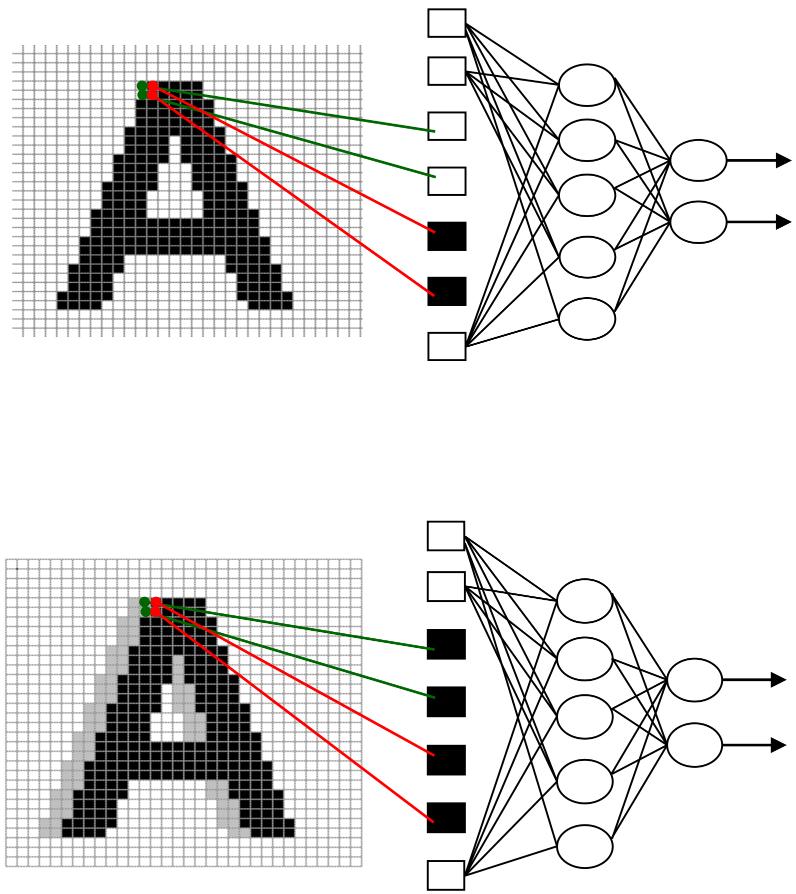
- Little or no invariance to shifting, scaling, and other forms of distortion

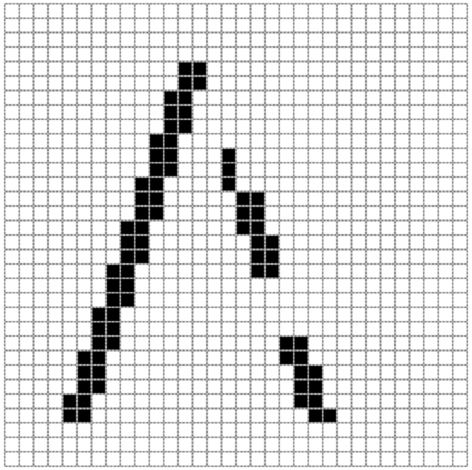


Deep Learning: Convolutional Neural Networks

Drawbacks of Neural Networks

- Little or no invariance to shifting, scaling, and other forms of distortion





Deep Learning: Convolutional Neural Networks

Drawbacks of Neural Networks

- ❑ Feature extraction and training

CNN

Next: Convolutional Neural Networks

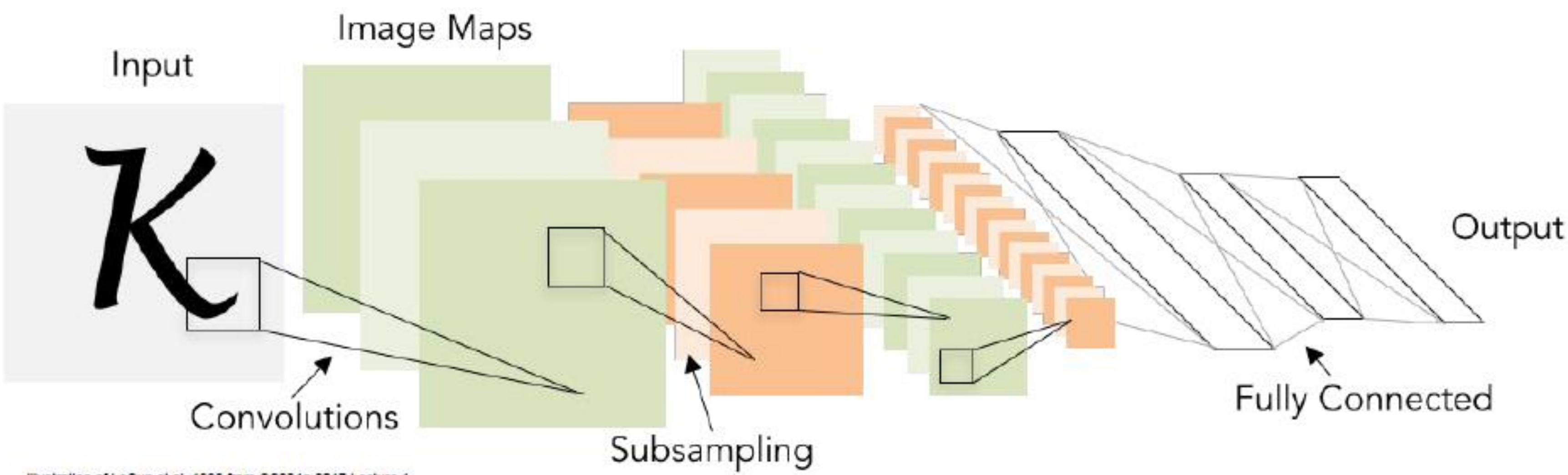
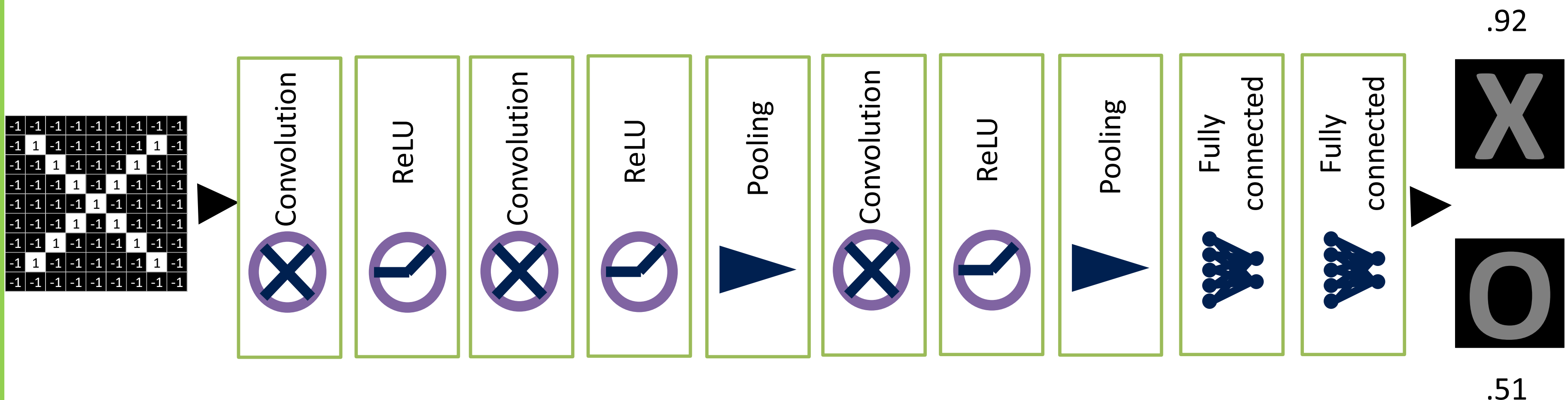


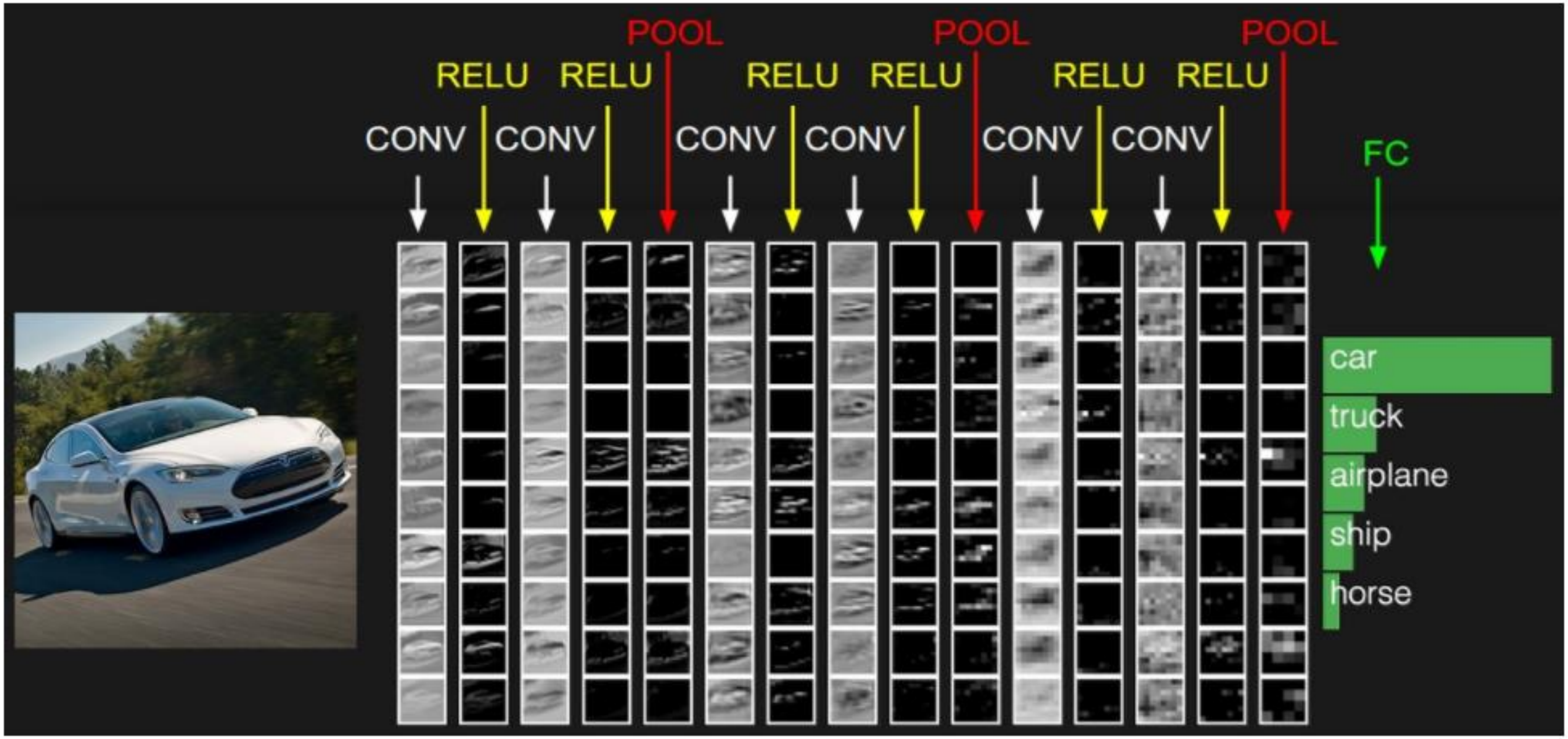
Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Putting it all together

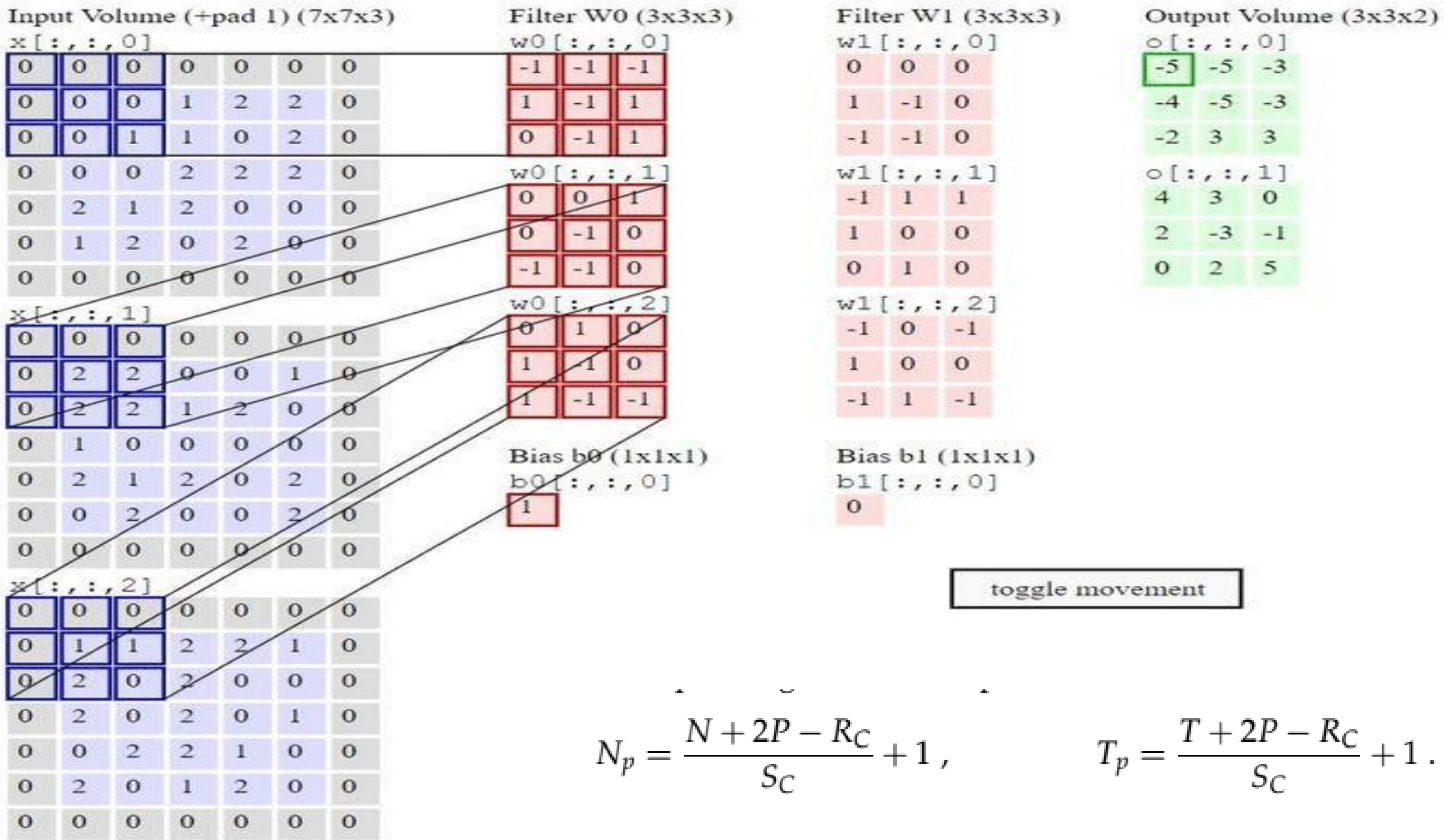
A set of pixels becomes a set of votes.



Deep Learning: CNN



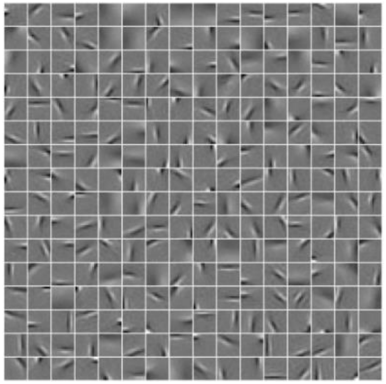
Deep Learning: CNN

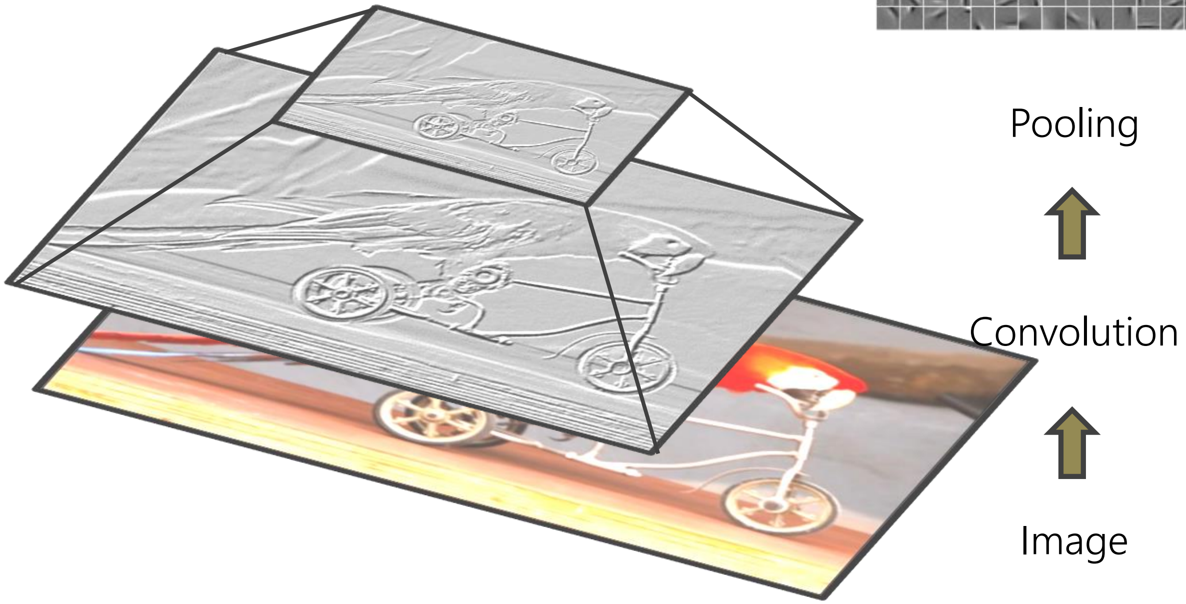


$$N_p = \frac{N + 2P - R_C}{S_C} + 1\,, \qquad T_p = \frac{T + 2P - R_C}{S_C} + 1\,.$$

Fig. Convolution Layer

2D input





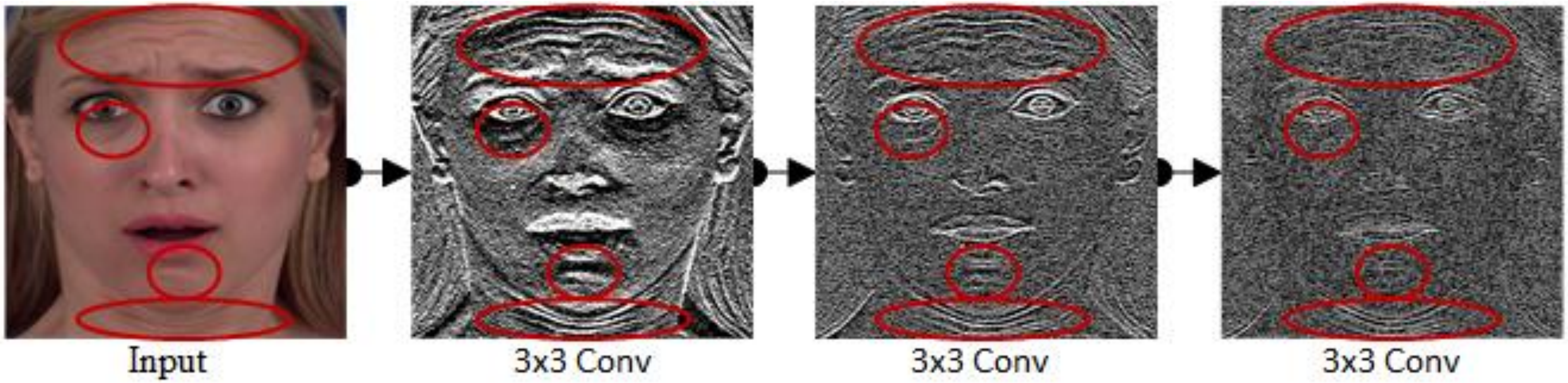
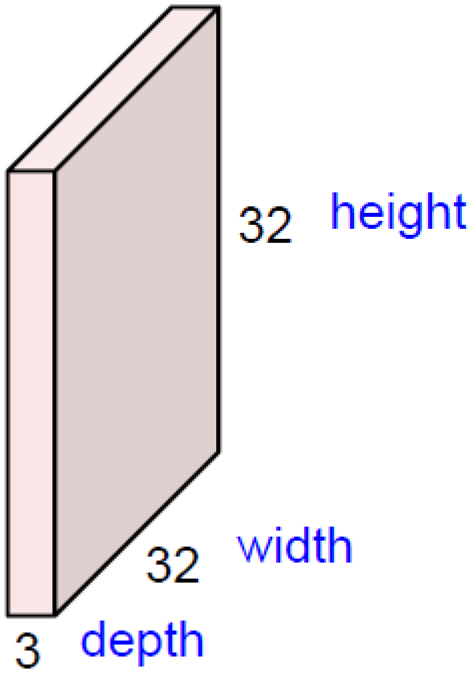


Fig. Visual representation of repetitive 3x3 convolution operation

Deep Learning: Convolutional Neural Networks

Convolution Layer

32x32x3 image -> preserve spatial structure

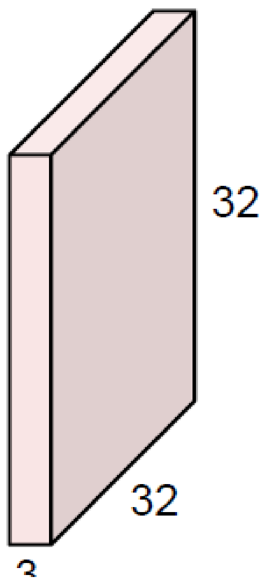


Deep Learning: Convolutional Neural Networks

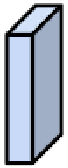
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



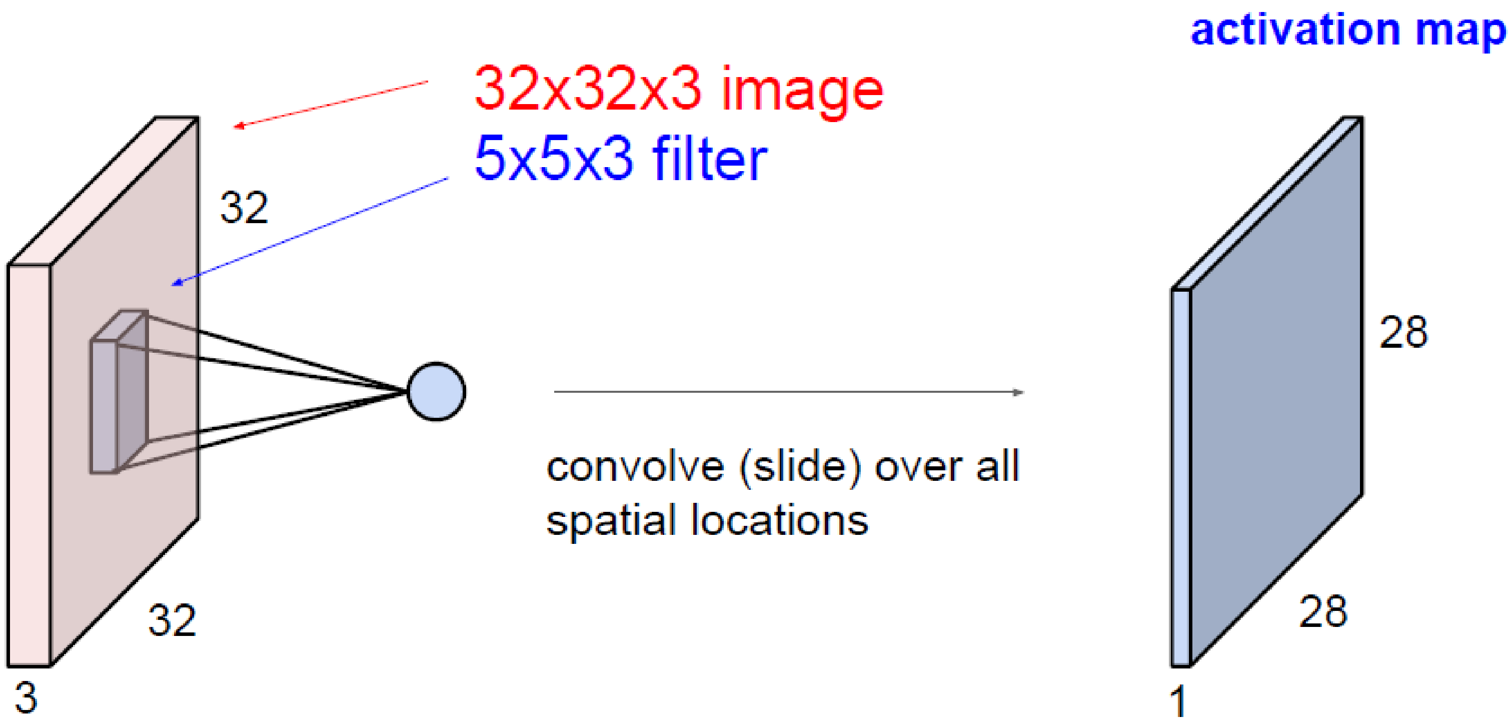
5x5x3 filter



Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

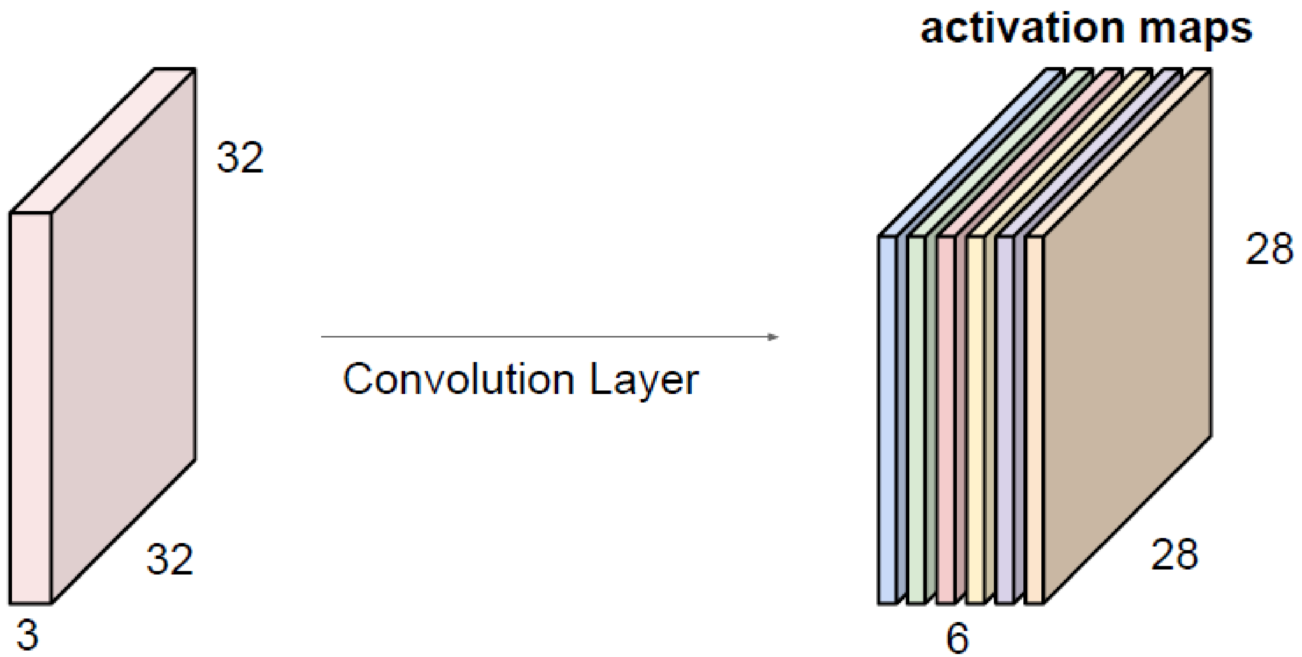
Deep Learning: Convolutional Neural Networks

Convolution Layer



Deep Learning: Convolutional Neural Networks

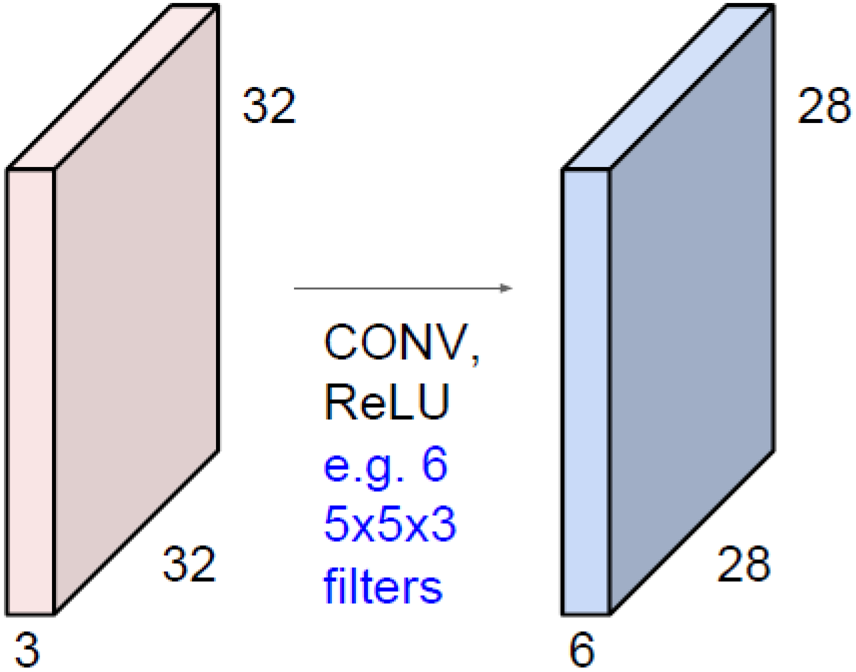
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

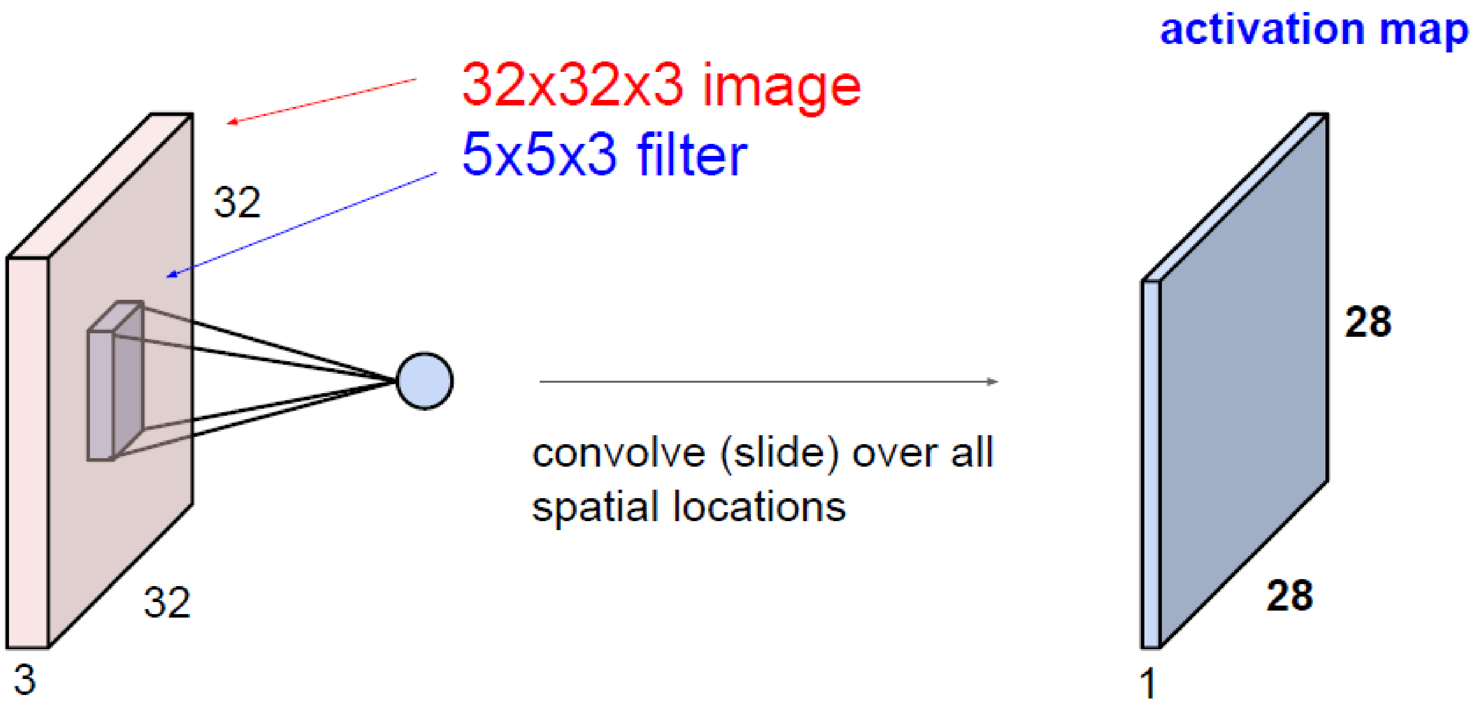
Deep Learning: Convolutional Neural Networks

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Deep Learning: Convolutional Neural Networks

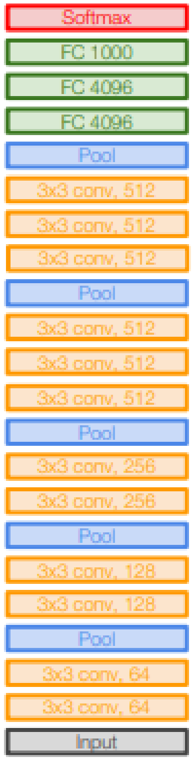
A closer look at spatial dimensions:



Deep Learning: Convolutional Neural Networks

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000



VGG16

```
INPUT:      [224x224x3]    memory:  224*224*3=150K   weights: 0
CONV3-64:   [224x224x64]   memory:  224*224*64=3.2M  weights: (3*3*3)*64 = 1,728
CONV3-64:   [224x224x64]   memory:  224*224*64=3.2M  weights: (3*3*64)*64 = 36,864
POOL2:      [112x112x64]   memory:  112*112*64=800K  weights: 0
CONV3-128:  [112x112x128]  memory:  112*112*128=1.6M weights: (3*3*64)*128 = 73,728
CONV3-128:  [112x112x128]  memory:  112*112*128=1.6M weights: (3*3*128)*128 = 147,456
POOL2:      [56x56x128]    memory:  56*56*128=400K   weights: 0
CONV3-256:  [56x56x256]    memory:  56*56*256=800K   weights: (3*3*128)*256 = 294,912
CONV3-256:  [56x56x256]    memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
CONV3-256:  [56x56x256]    memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
POOL2:      [28x28x256]    memory:  28*28*256=200K   weights: 0
CONV3-512:  [28x28x512]    memory:  28*28*512=400K   weights: (3*3*256)*512 = 1,179,648
CONV3-512:  [28x28x512]    memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
CONV3-512:  [28x28x512]    memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
POOL2:      [14x14x512]    memory:  14*14*512=100K   weights: 0
CONV3-512:  [14x14x512]    memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512:  [14x14x512]    memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512:  [14x14x512]    memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
POOL2:      [7x7x512]      memory:  7*7*512=25K      weights: 0
FC:         [1x1x4096]     memory:  4096             weights: 7*7*512*4096 = 102,760,448
FC:         [1x1x4096]     memory:  4096             weights: 4096*4096 = 16,777,216
FC:         [1x1x1000]     memory:  1000             weights: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters
```

Back Propagation in CNN

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

W_{11}	W_{12}
W_{21}	W_{22}

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

W_{11}	W_{12}
W_{21}	W_{22}

h_{11}	h_{12}
h_{21}	h_{22}

$$h_{11} = W_{11}X_{11} + W_{12}X_{12} + W_{21}X_{21} + W_{22}X_{22}$$

$$h_{12} = W_{11}X_{12} + W_{12}X_{13} + W_{21}X_{22} + W_{22}X_{23}$$

$$h_{21} = W_{11}X_{21} + W_{12}X_{22} + W_{21}X_{31} + W_{22}X_{32}$$

$$h_{22} = W_{11}X_{22} + W_{12}X_{23} + W_{21}X_{32} + W_{22}X_{33}$$

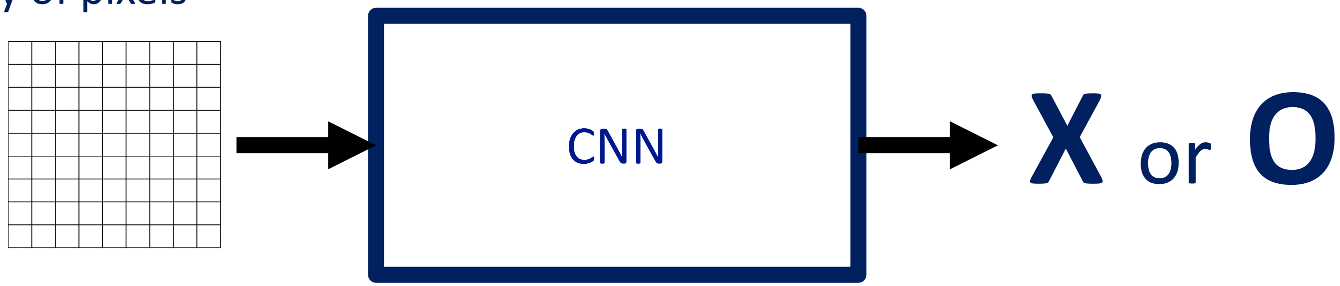
How it Works:

Convolutional Neural Networks

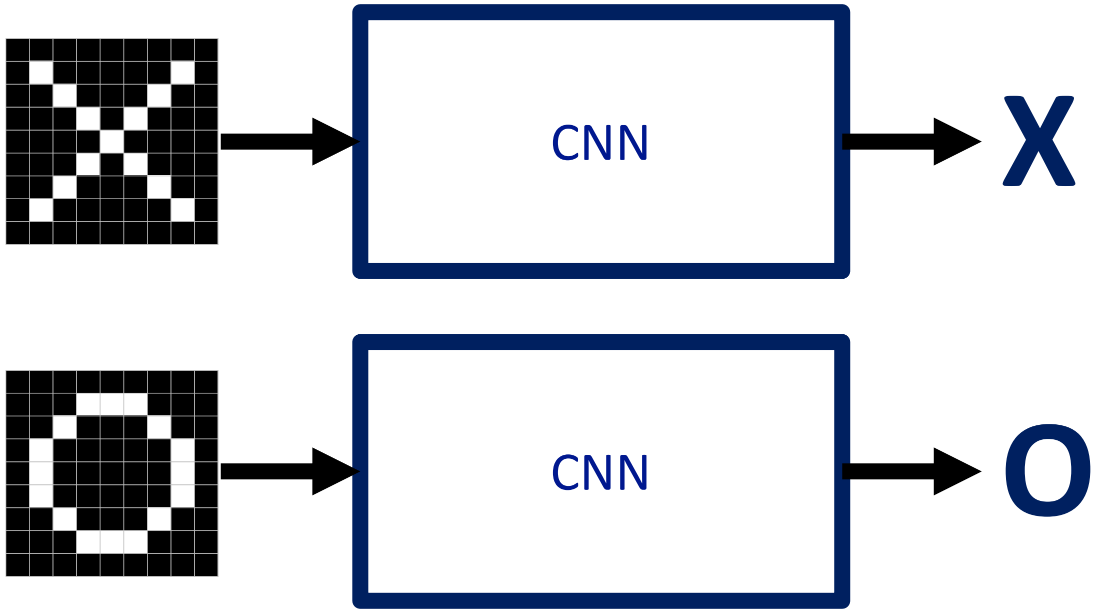
A toy ConvNet: X's and O's

Says whether a picture is of an X or an O

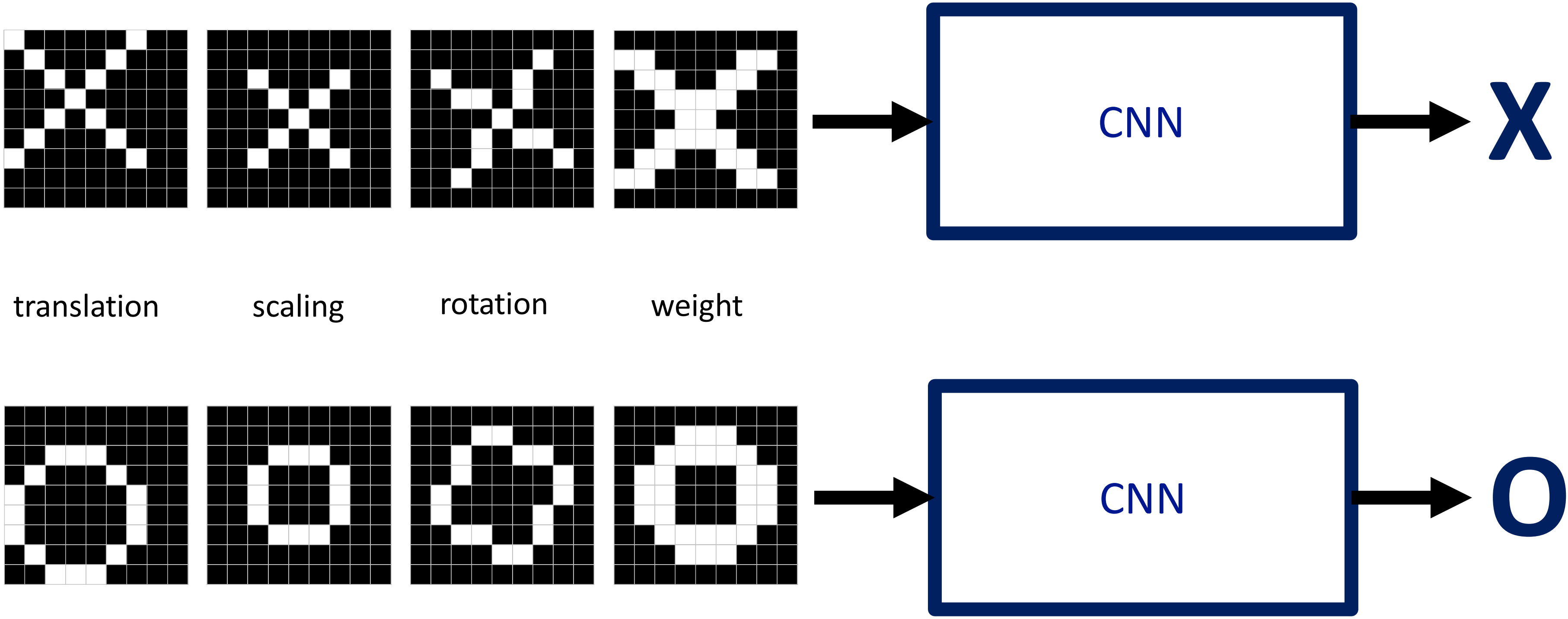
A two-dimensional array of pixels



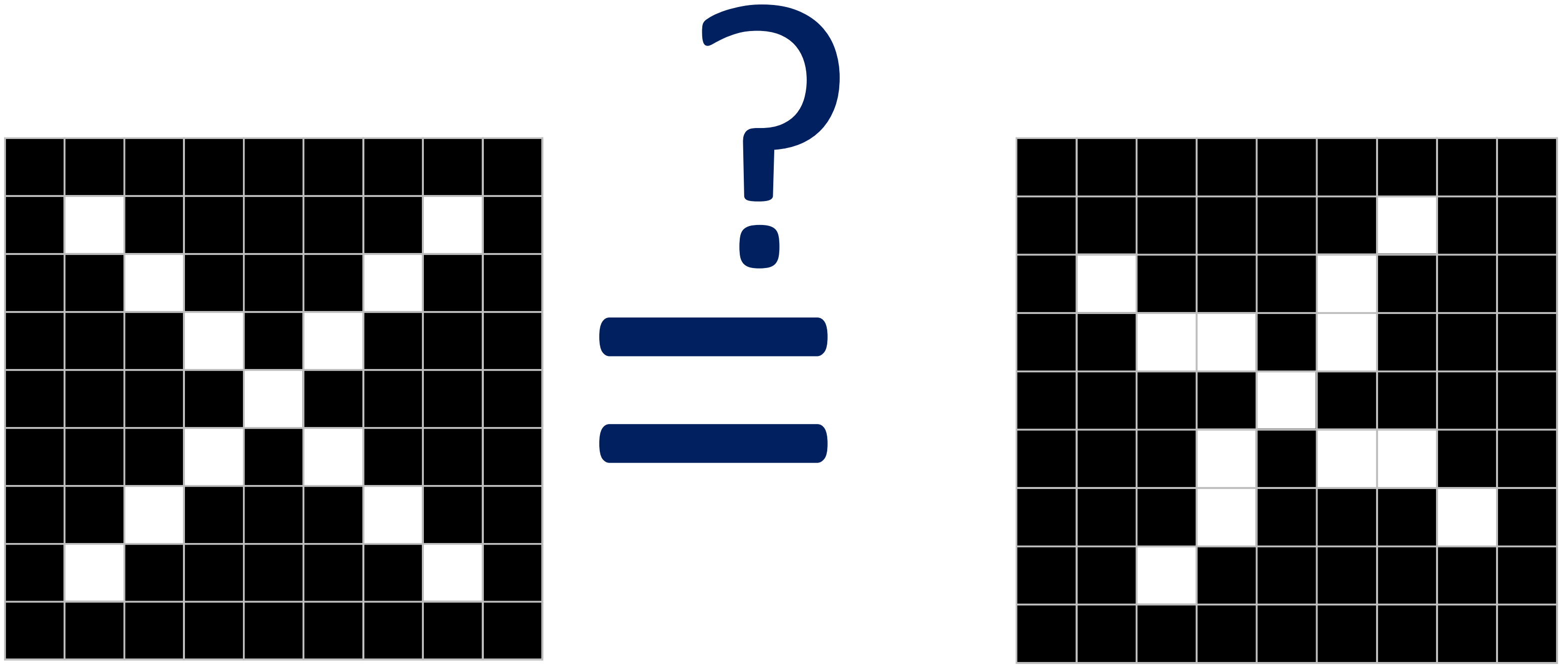
For example



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

$\overset{?}{=}$

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

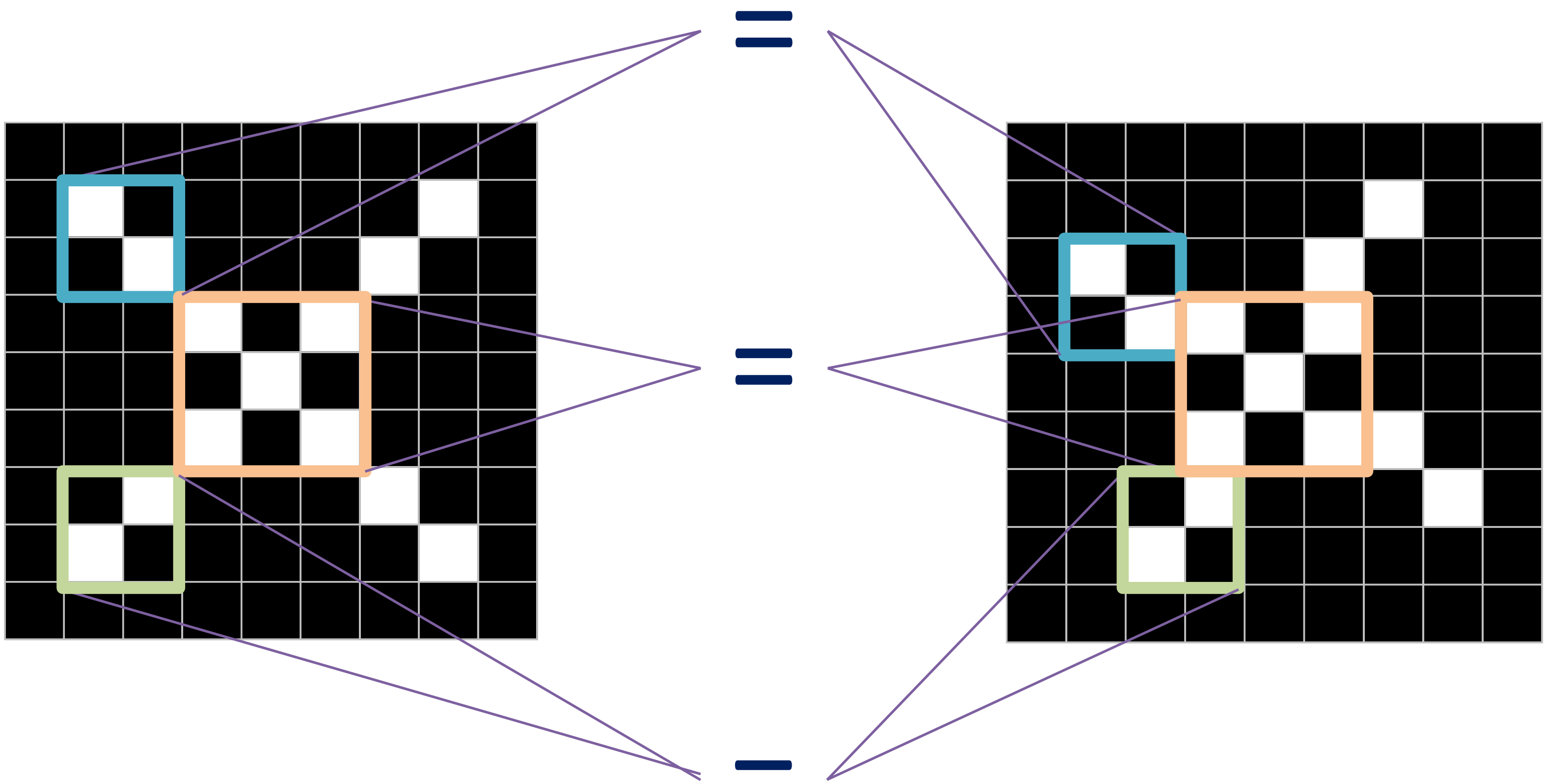
Computers are literal

-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

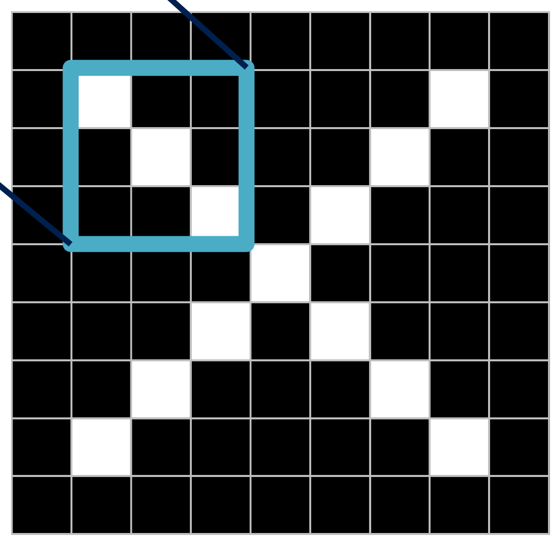
1	-1	1
-1	1	-1
1	-1	1

-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

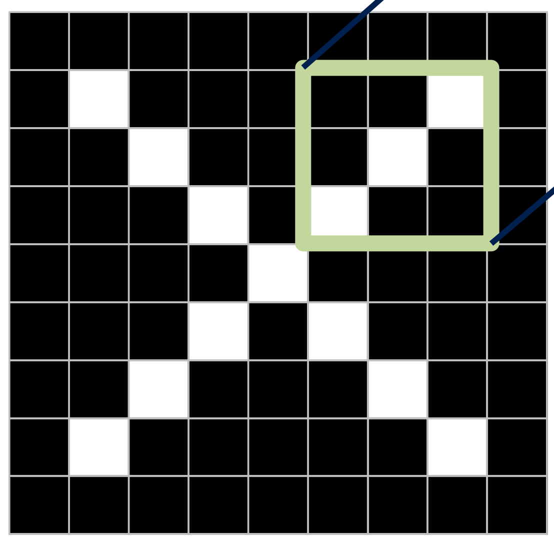
-1	-1	1
-1	1	-1
1	-1	-1



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

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

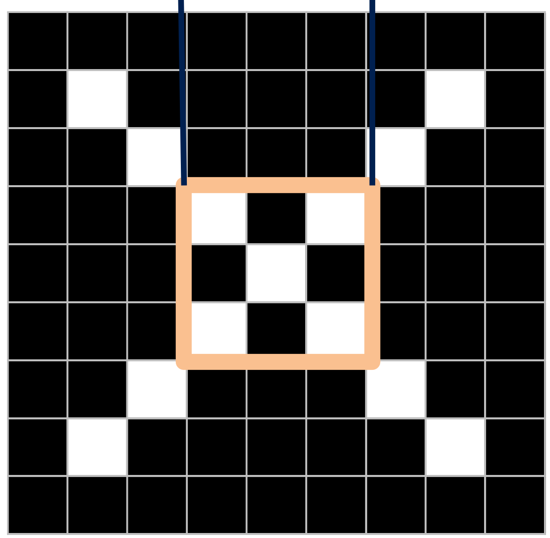
-1	-1	1
-1	1	-1
1	-1	-1



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

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

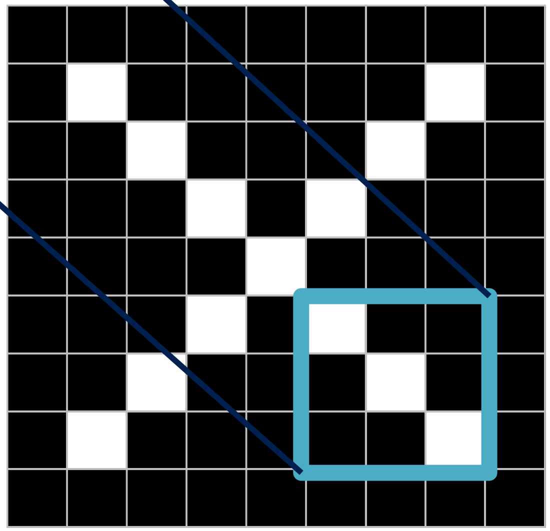
-1	-1	1
-1	1	-1
1	-1	-1



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

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

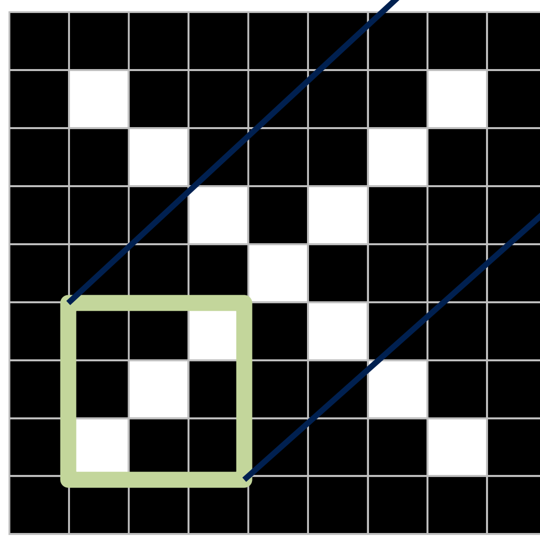
-1	-1	1
-1	1	-1
1	-1	-1



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.

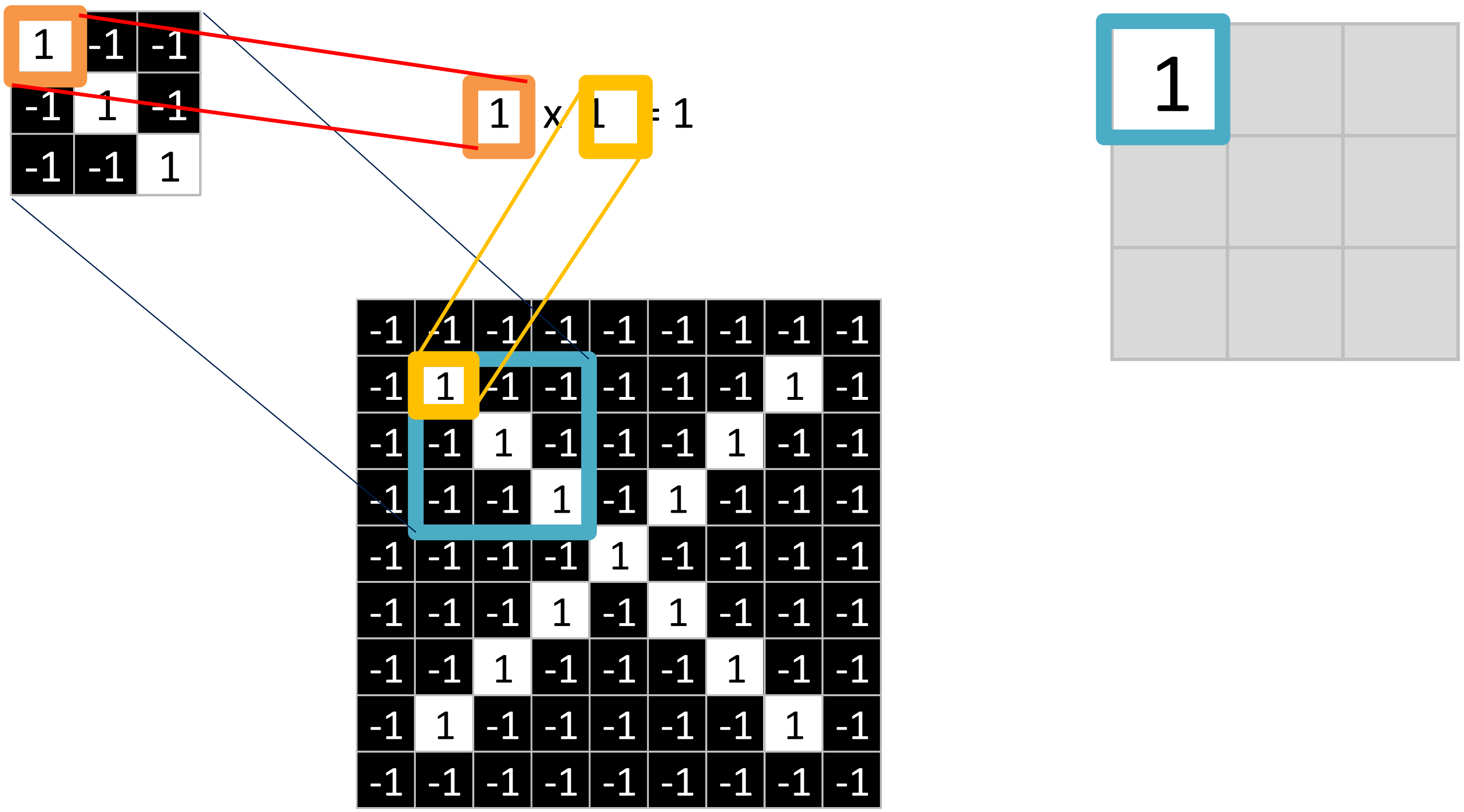
Filtering: The math behind the match

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

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

Filtering: The math behind the match



Filtering: The math behind the match

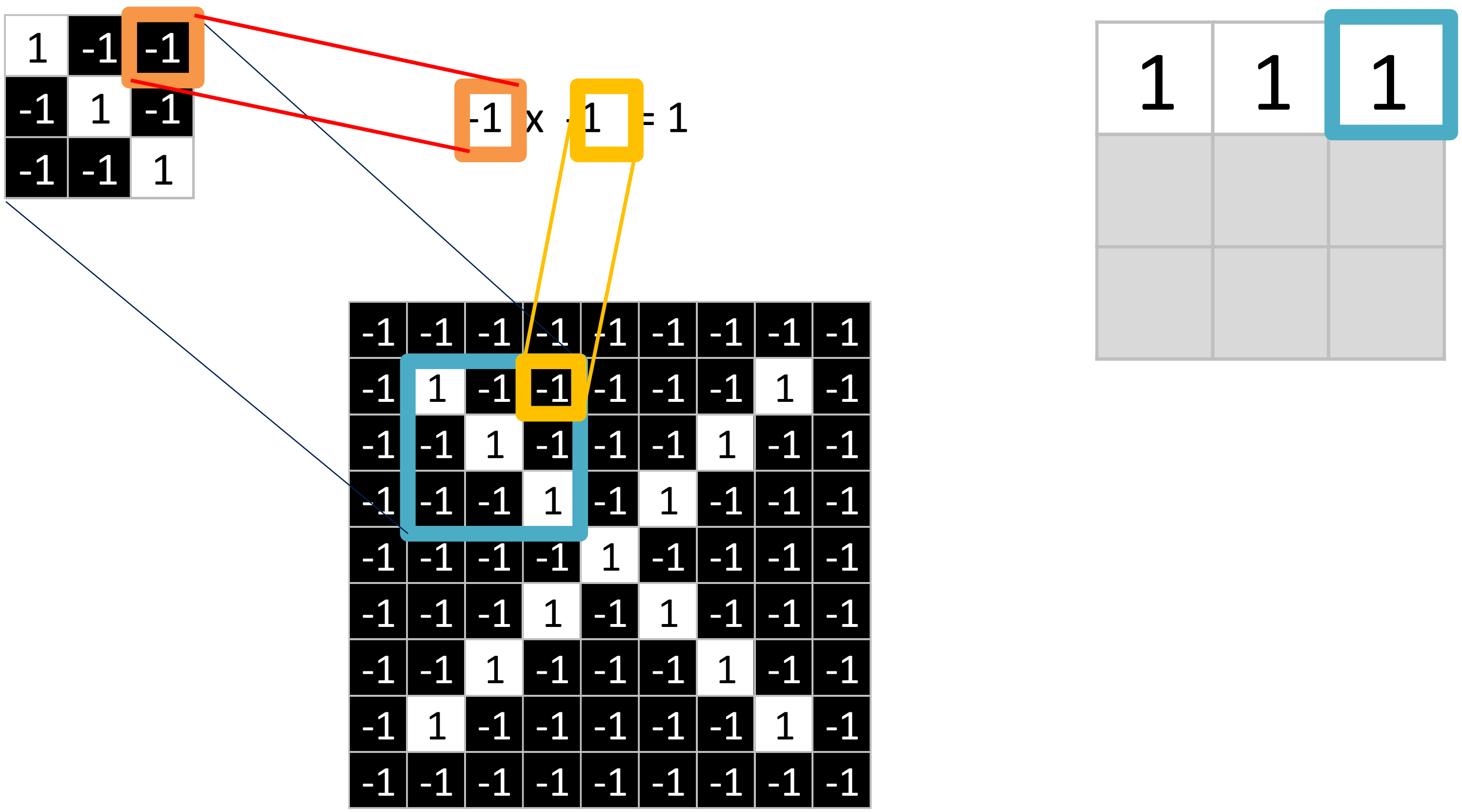
1	-1	-1
-1	1	-1
-1	-1	1

$-1 \times -1 = 1$

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

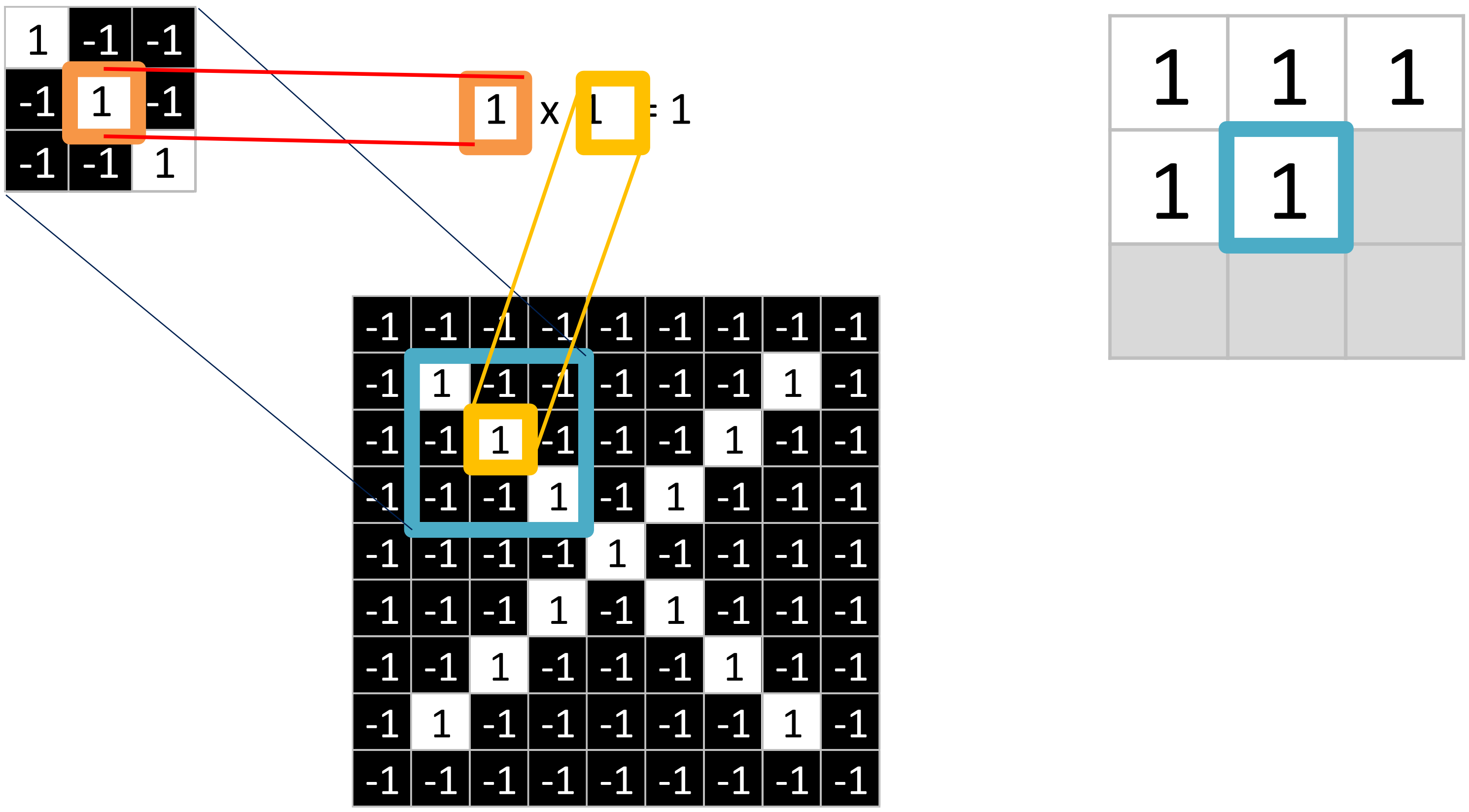
1	-1	-1
-1	1	-1
-1	-1	1

-1 x -1 = 1

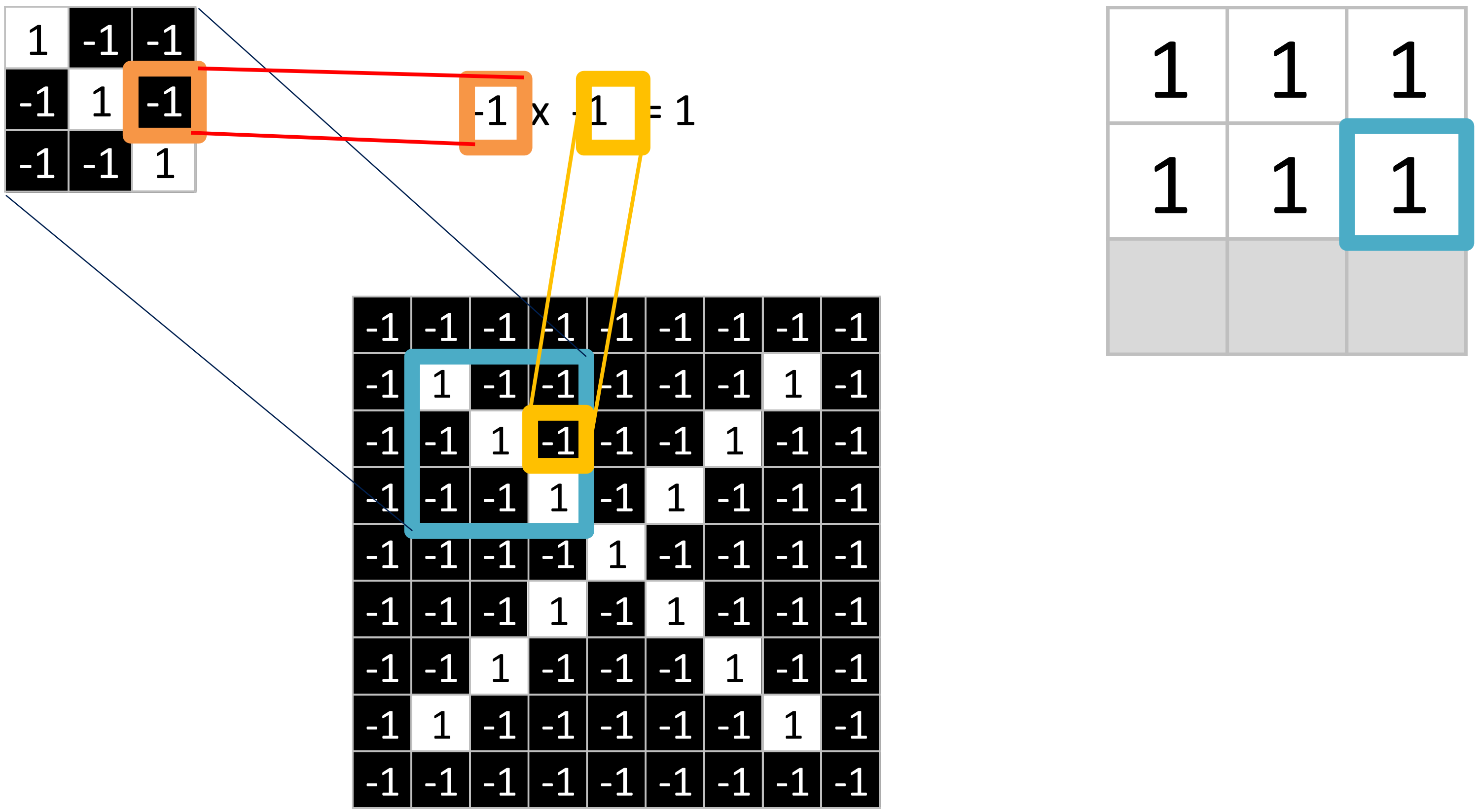
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

1	1	1
1		

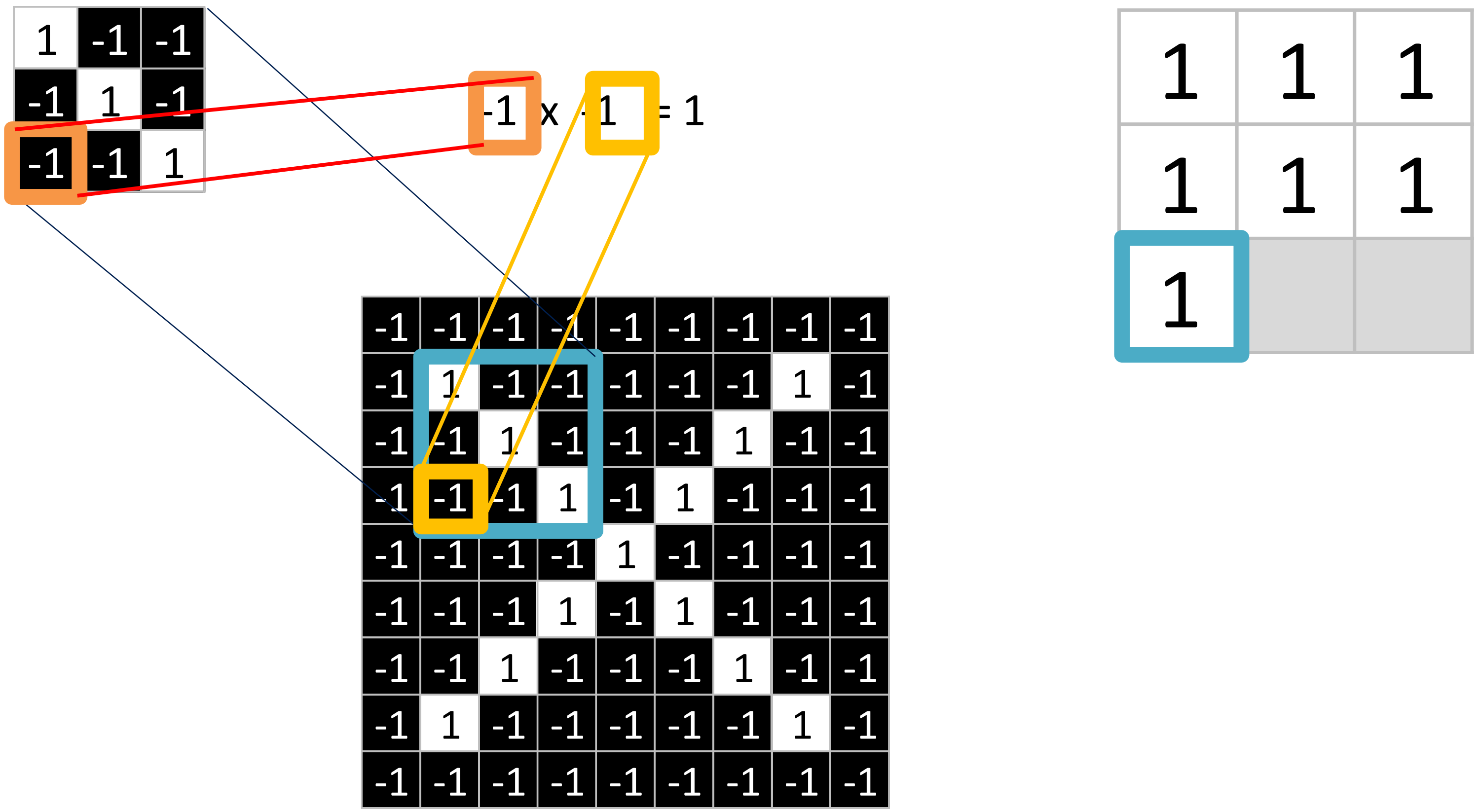
Filtering: The math behind the match



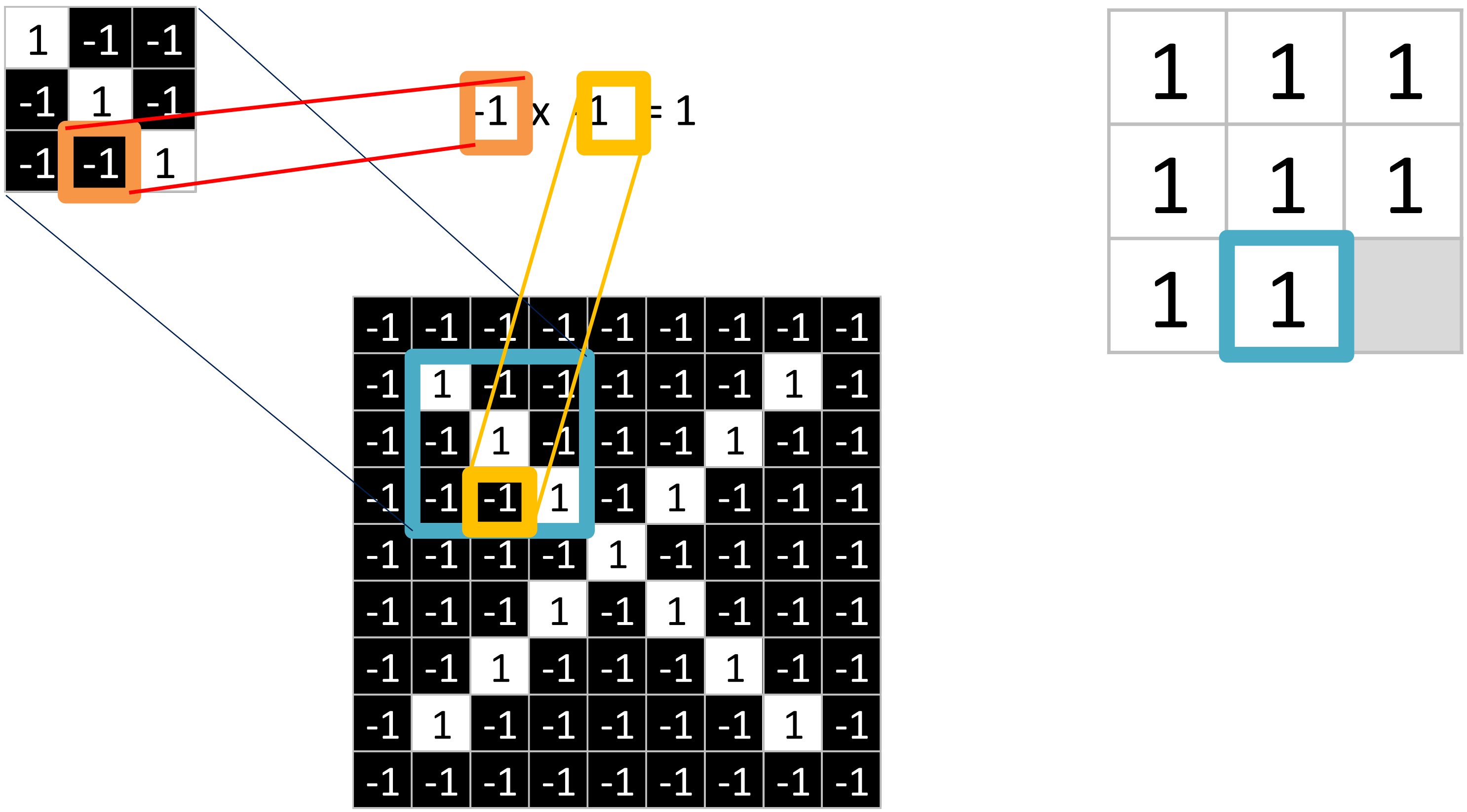
Filtering: The math behind the match



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	1
1	1	1

Filtering: The math behind the match

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

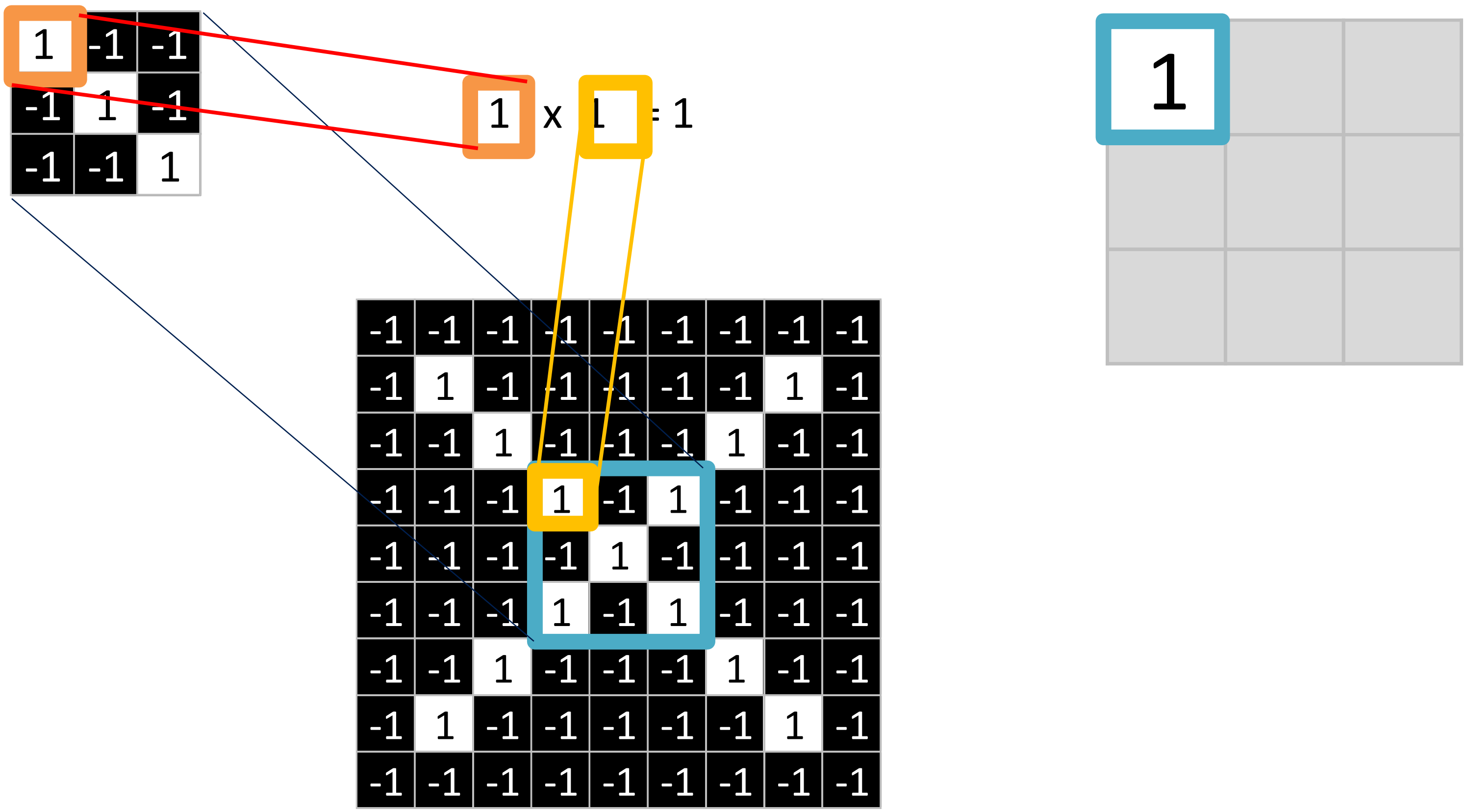
1	1	1
1	1	1
1	1	1

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

-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

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

$-1 \times 1 = -1$

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

1	1	-1

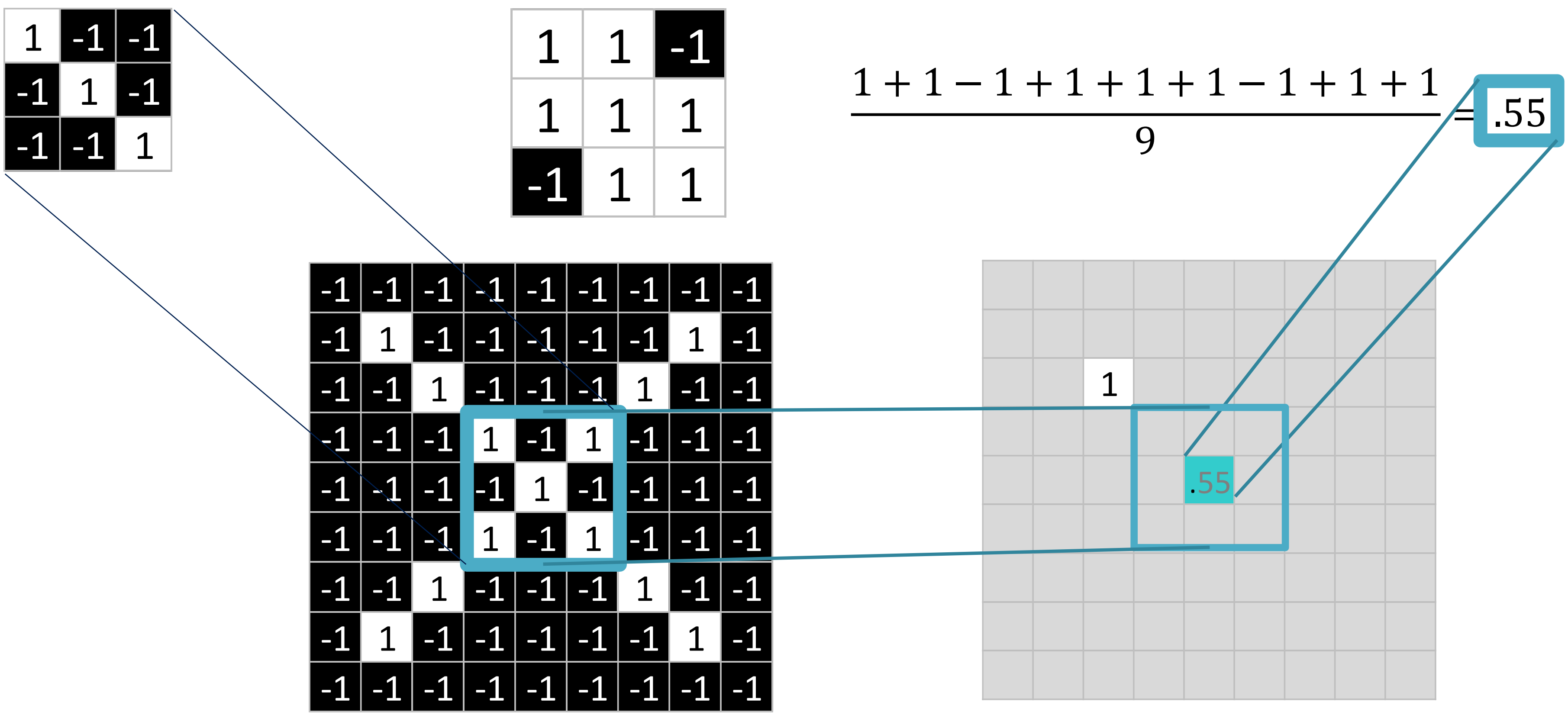
Filtering: The math behind the match

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

1	1	-1
1	1	1
-1	1	1

Filtering: The math behind the match



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

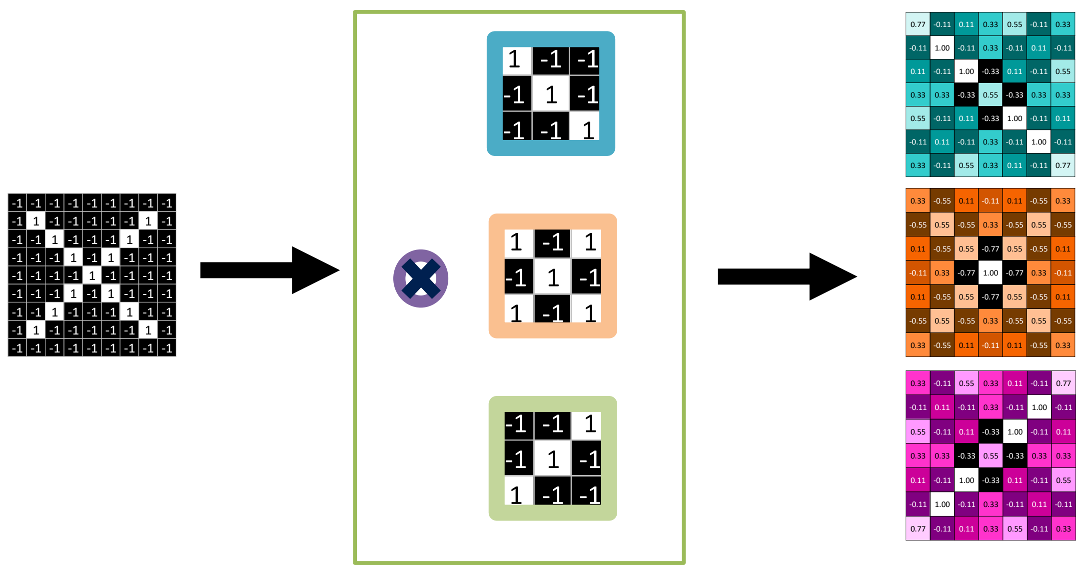
$$\begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & 1 & -1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

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

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

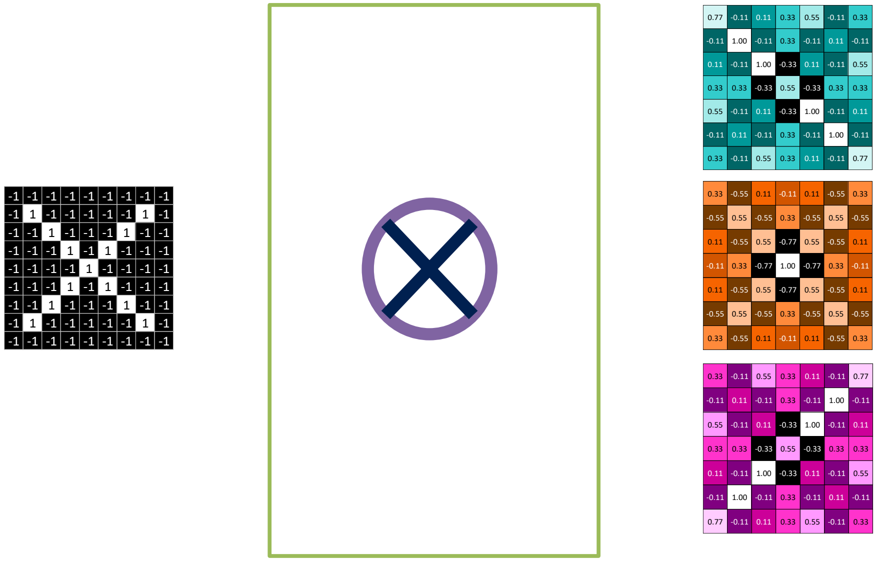
Convolution layer

One image becomes a stack of filtered images



Convolution layer

One image becomes a stack of filtered images

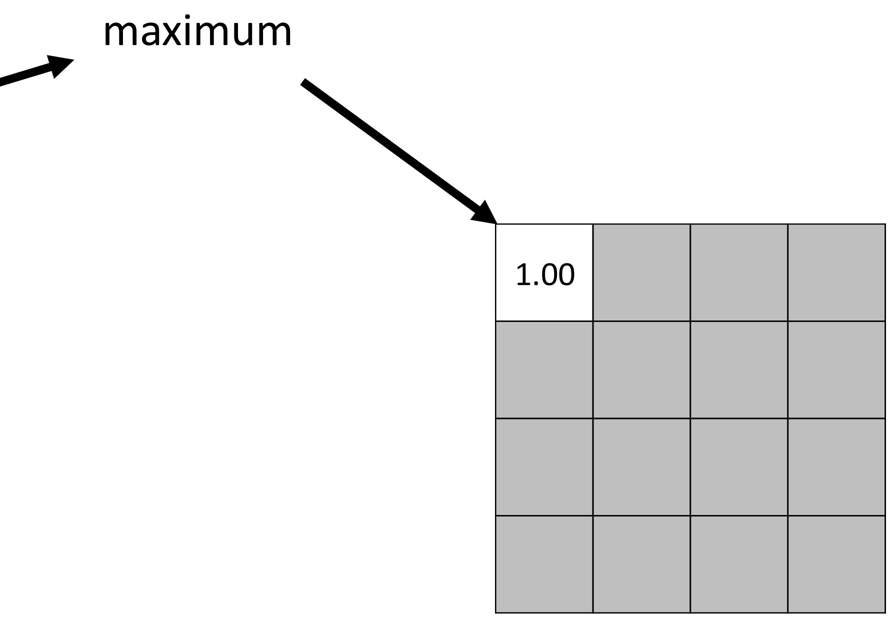


Pooling: Shrinking the image stack

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

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



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

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

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

Introduction to Deep Learning

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

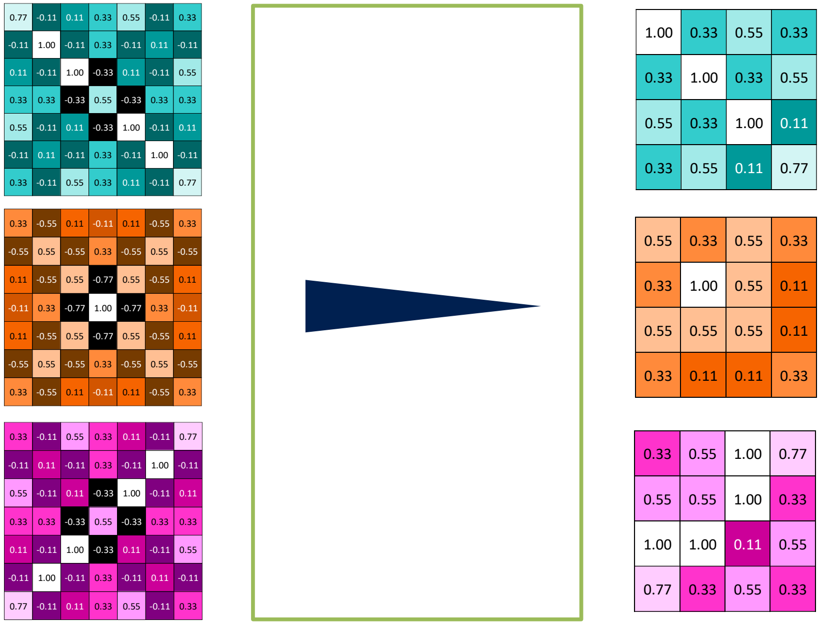
0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

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.

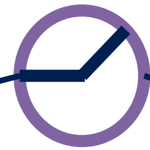


Normalization

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

Change everything negative to zero.

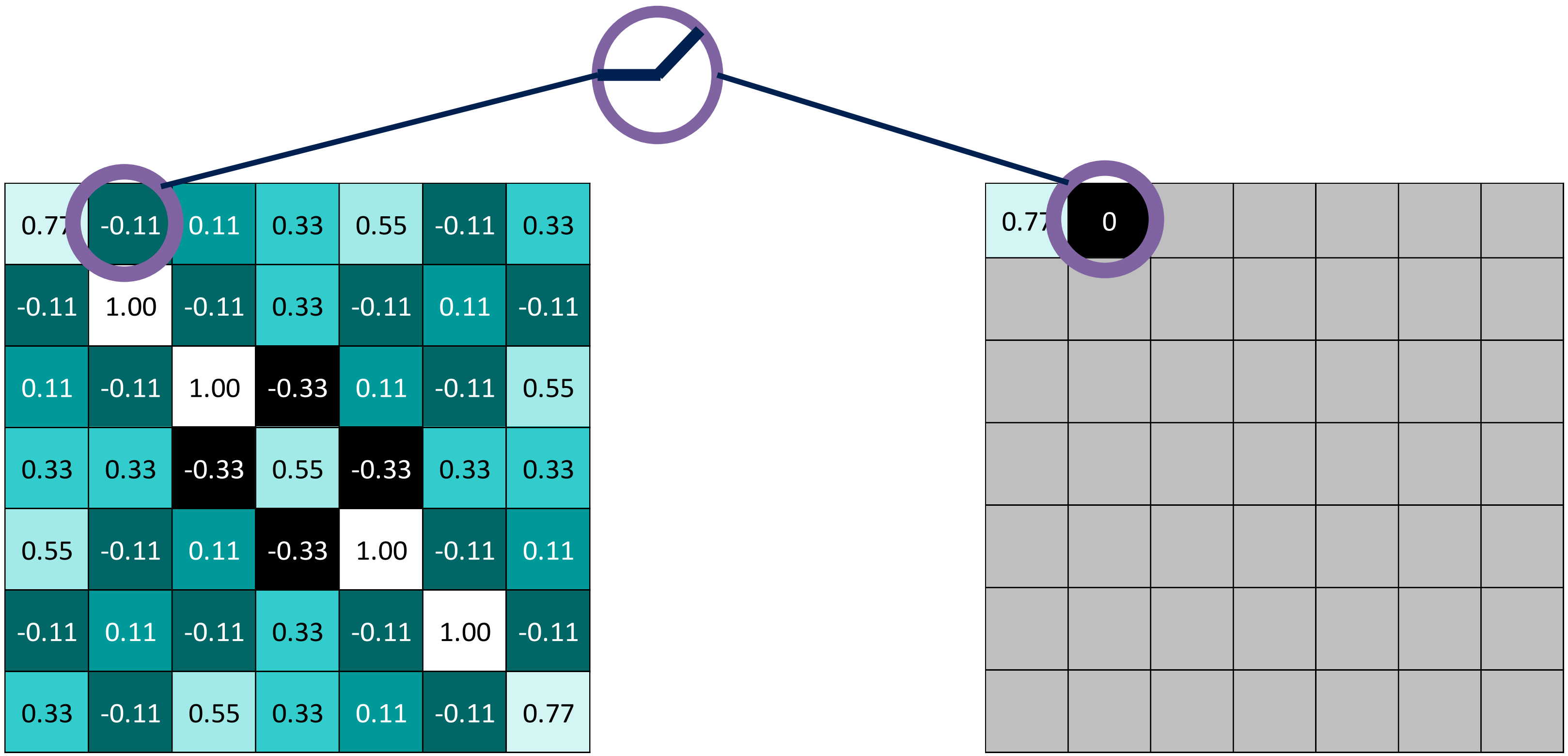
Rectified Linear Units (ReLUs)



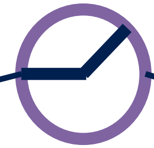
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

0.77						

Rectified Linear Units (ReLUs)



Rectified Linear Units (ReLUs)

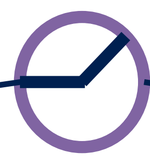


0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

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

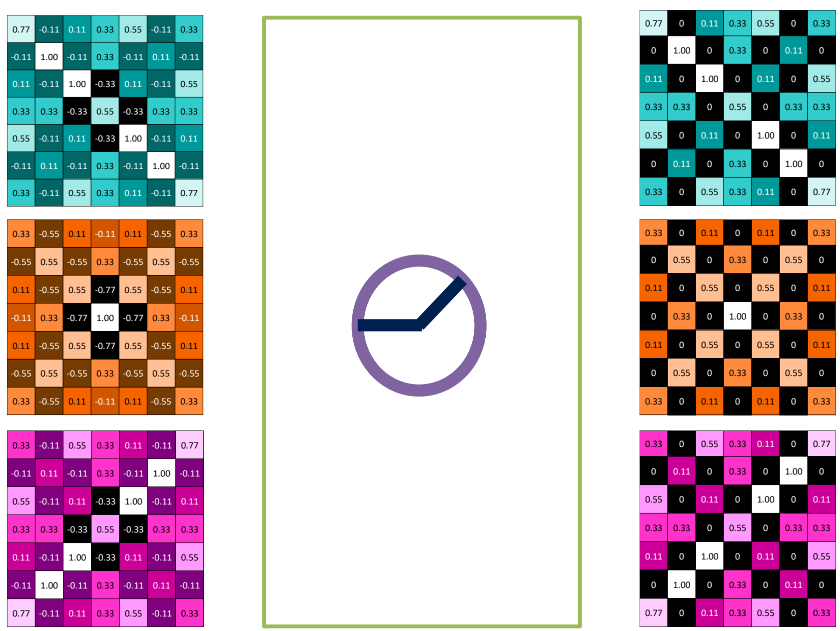


0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

Deep Learning

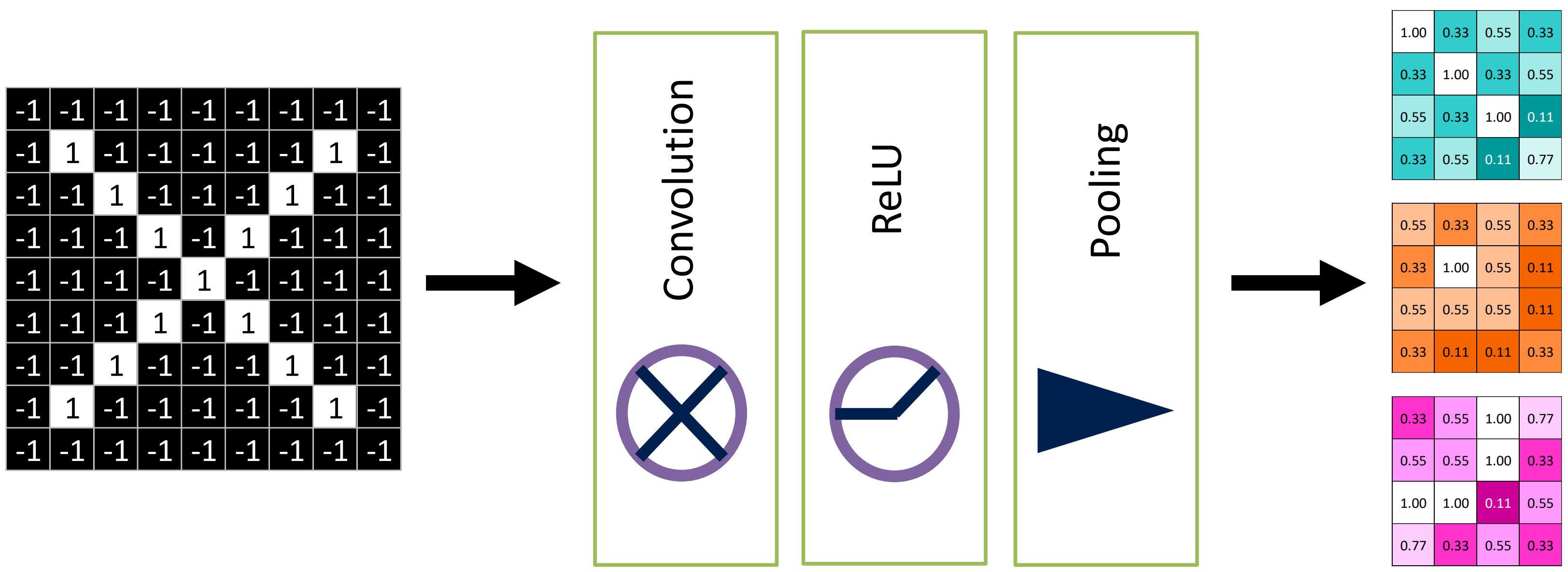
ReLU layer

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



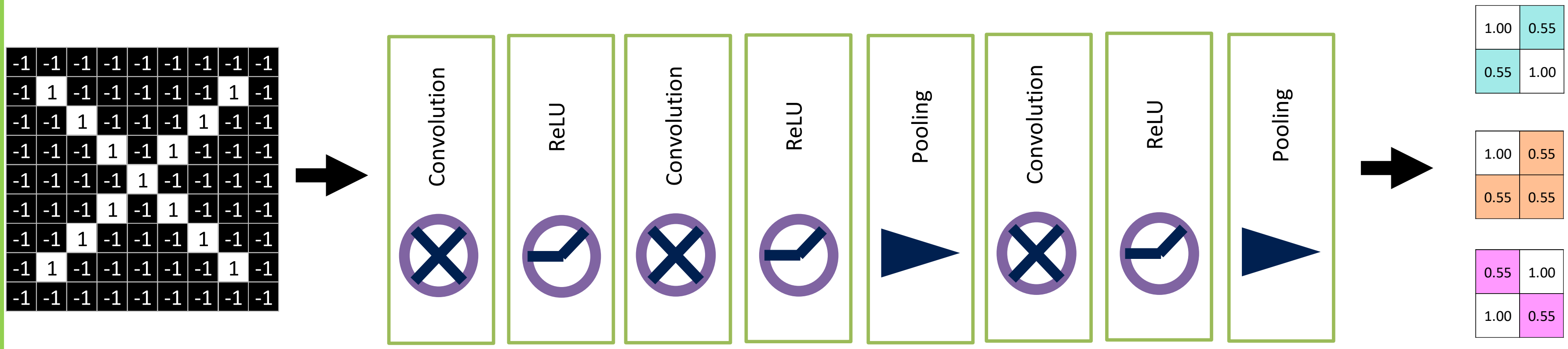
Layers get stacked

The output of one becomes the input of the next.



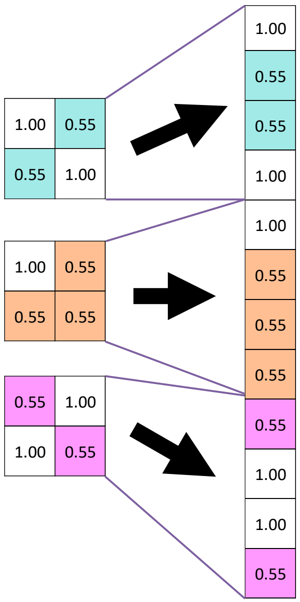
Deep stacking

Layers can be repeated several (or many) times.



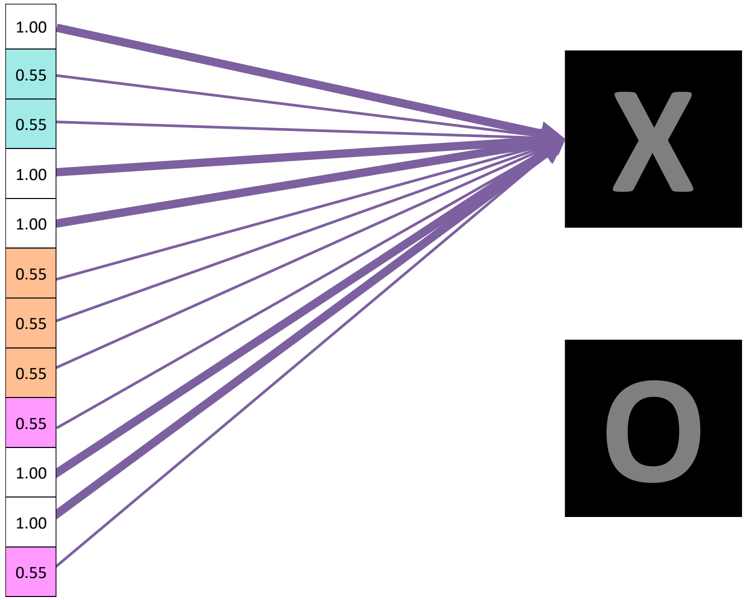
Fully connected layer

Every value gets a vote



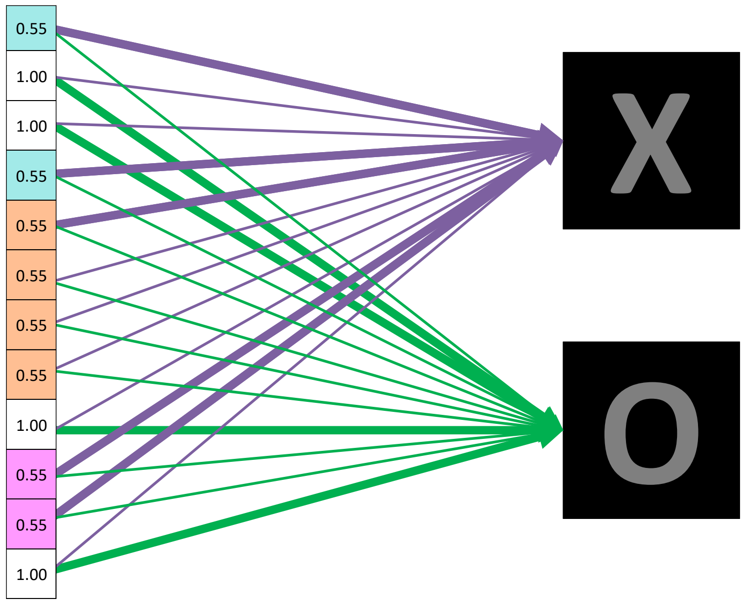
Fully connected layer

Vote depends on how strongly a value predicts X or O



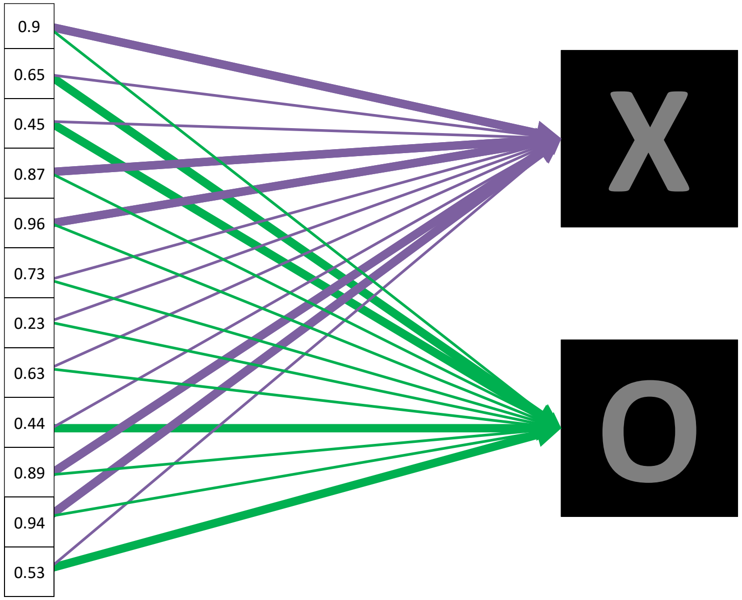
Fully connected layer

Vote depends on how strongly a value predicts X or O



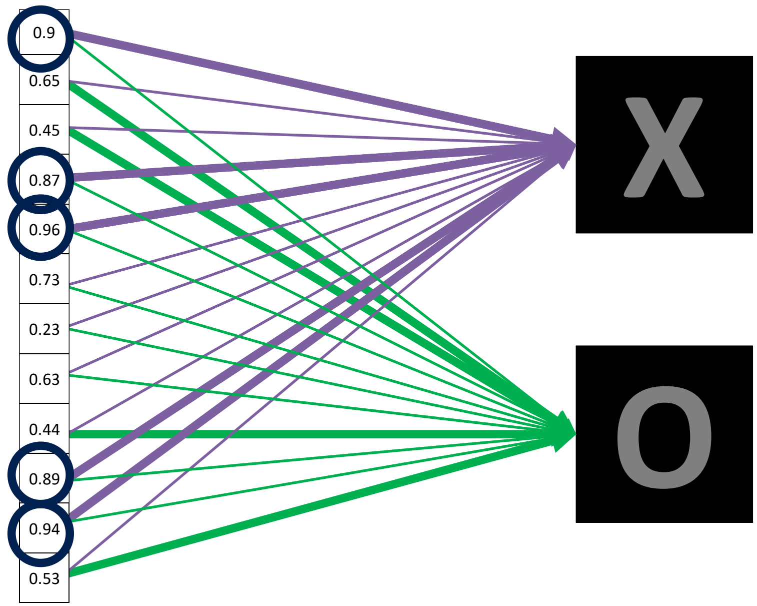
Fully connected layer

Future values vote on X or O



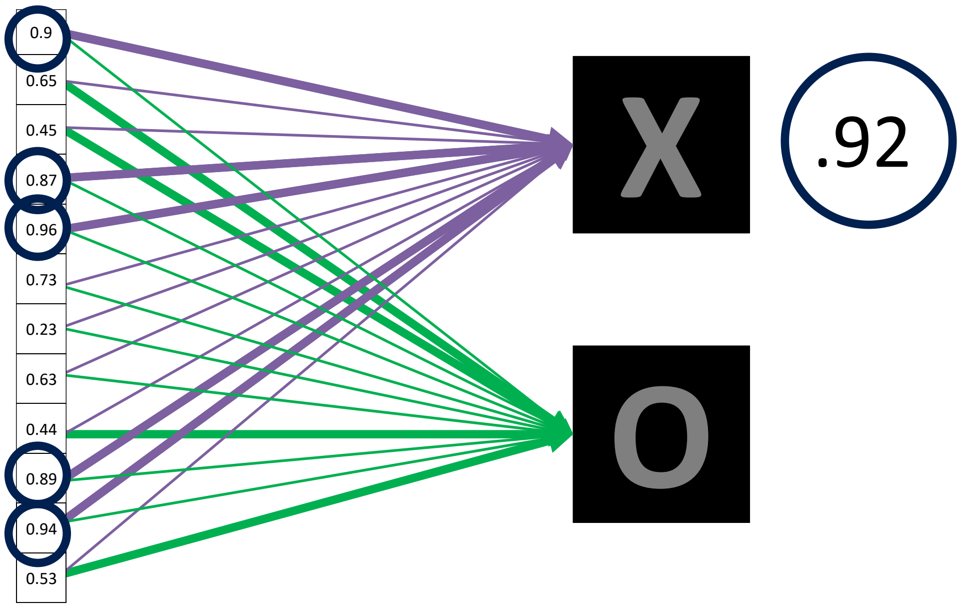
Fully connected layer

Future values vote on X or O



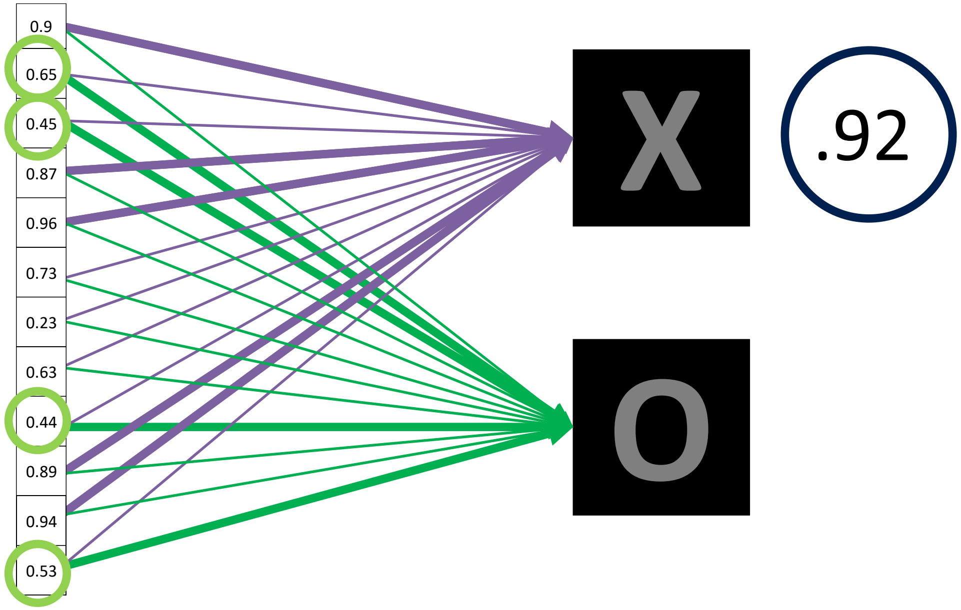
Fully connected layer

Future values vote on X or O



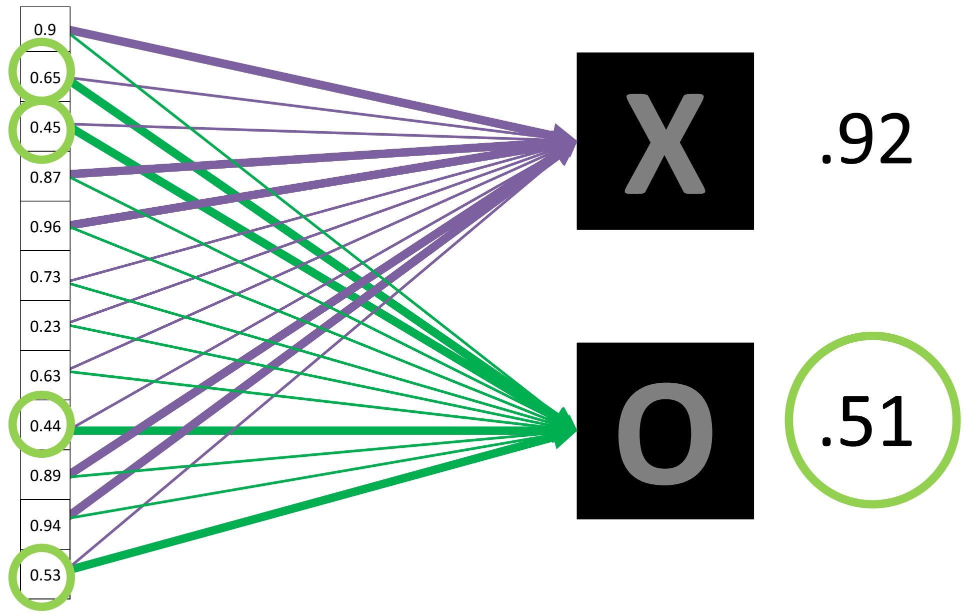
Fully connected layer

Future values vote on X or O



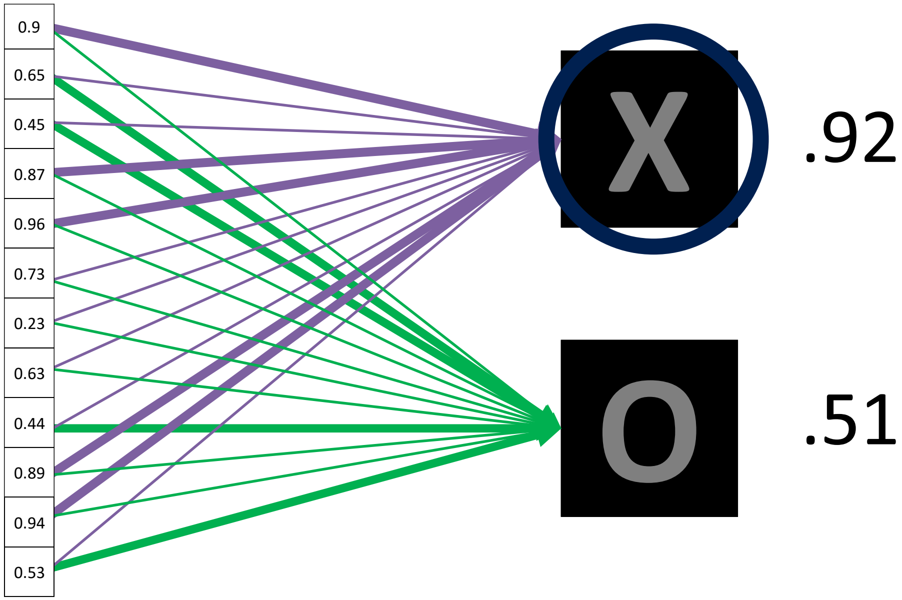
Fully connected layer

Future values vote on X or O



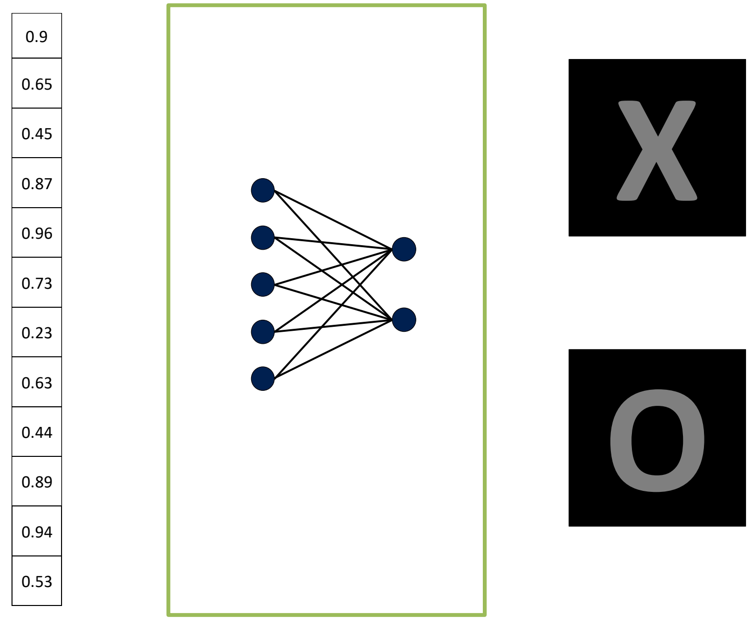
Fully connected layer

Future values vote on X or O



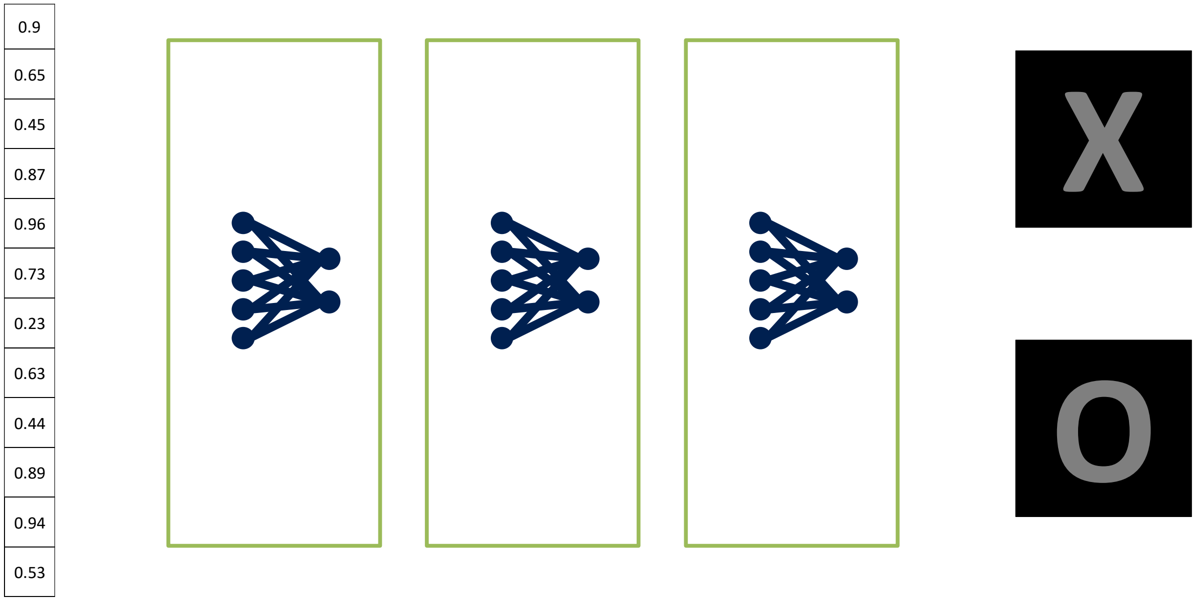
Fully connected layer

A list of feature values becomes a list of votes.



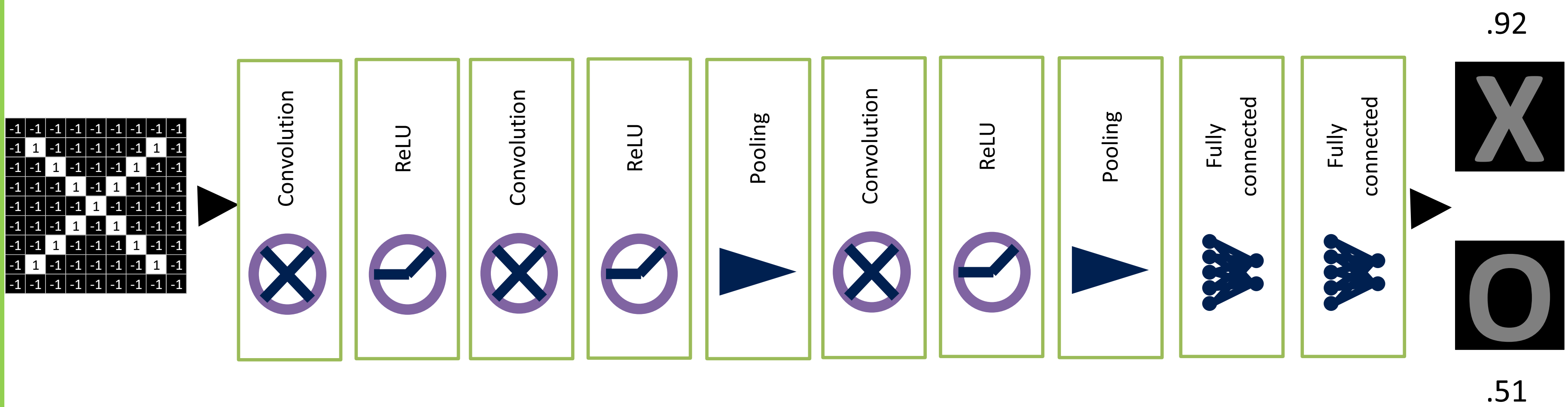
Fully connected layer

These can also be stacked.



Putting it all together

A set of pixels becomes a set of votes.



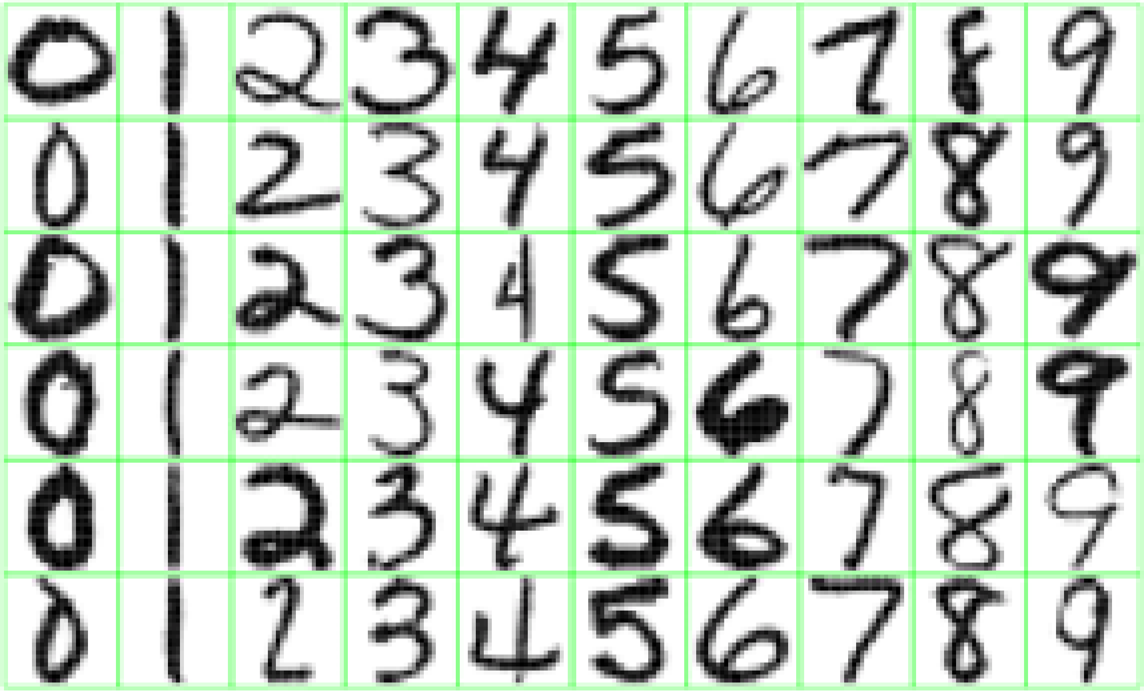


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

What features might you expect a good NN to learn, when trained with data like this?

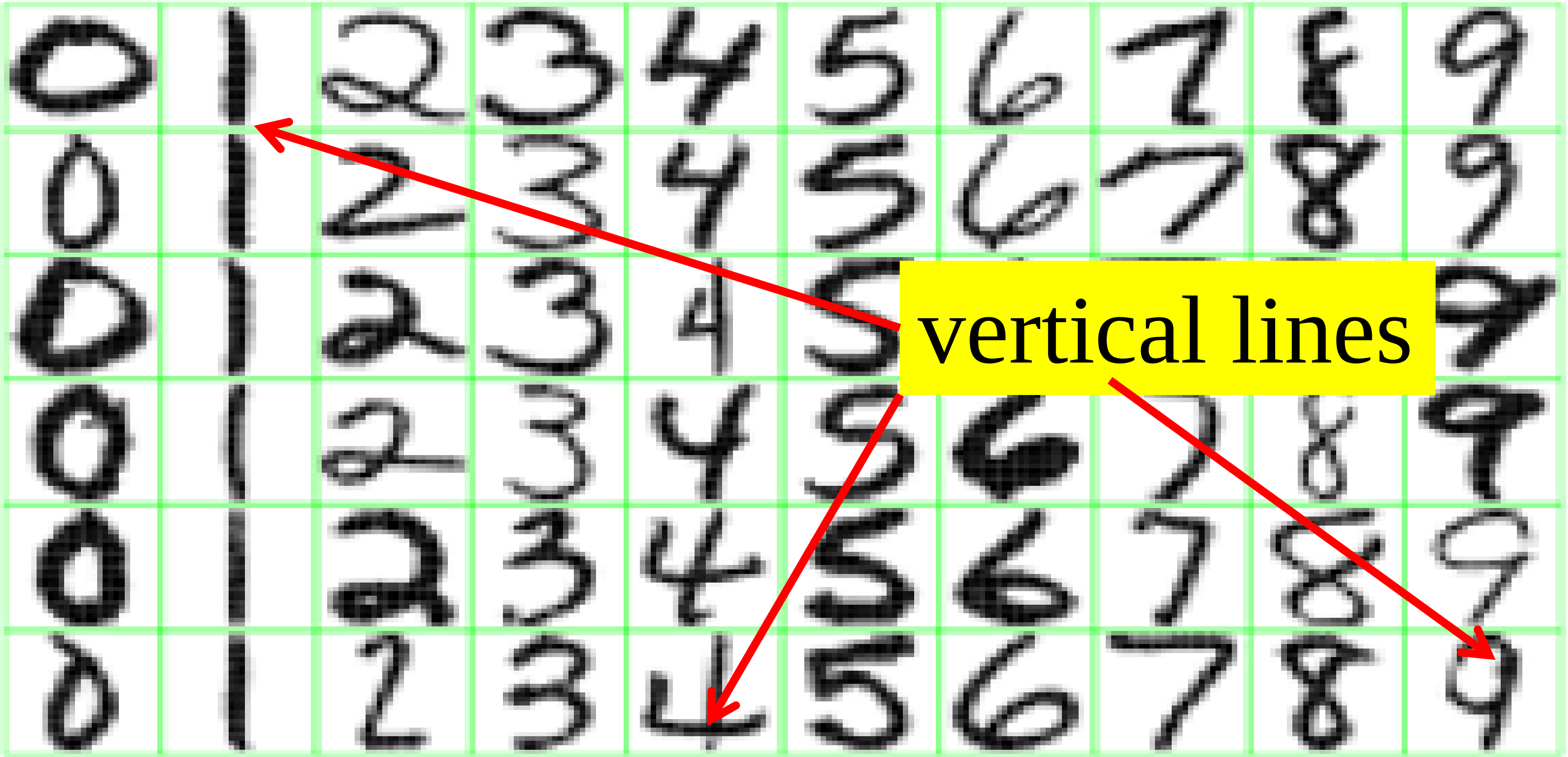


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

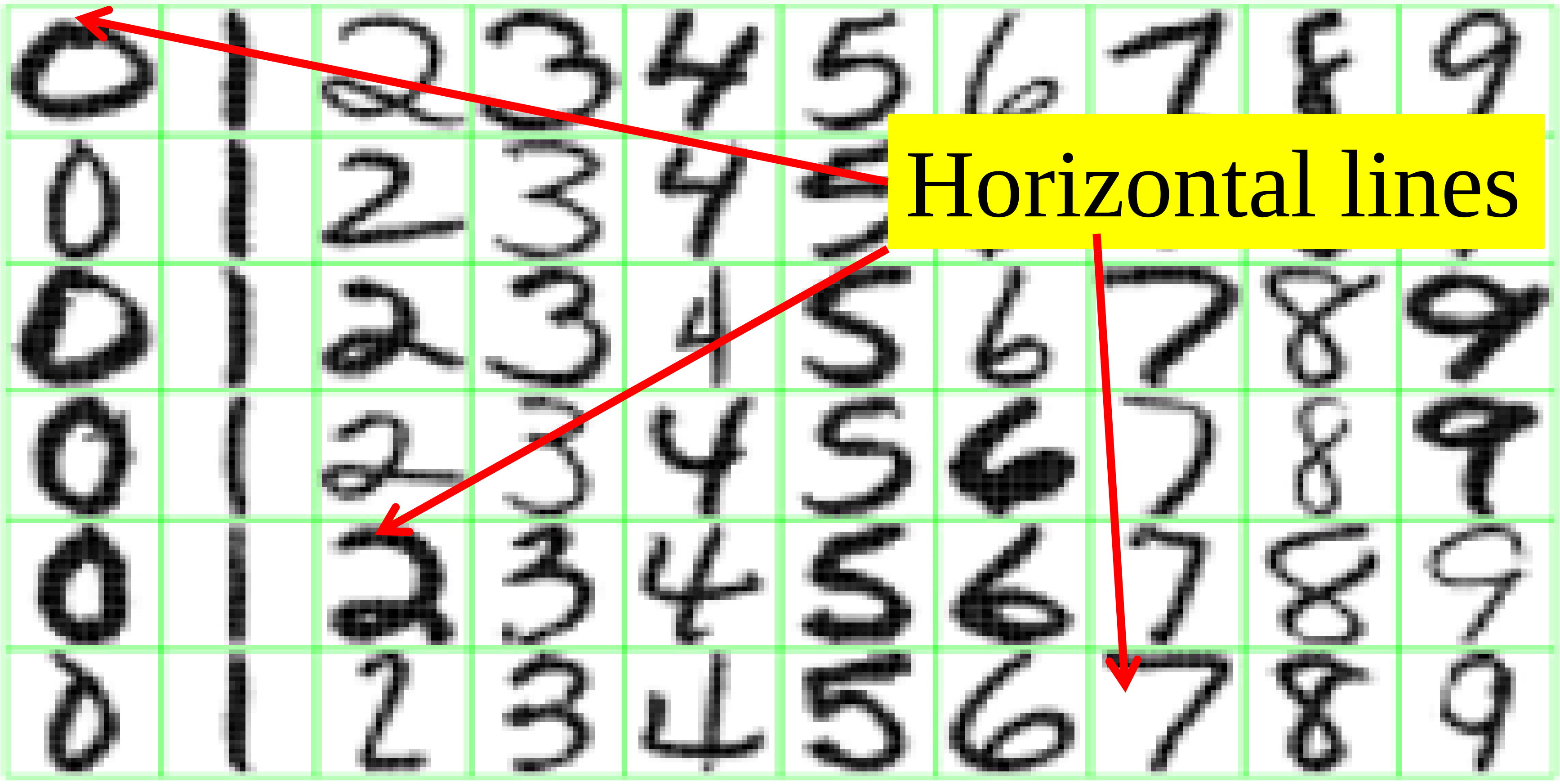


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

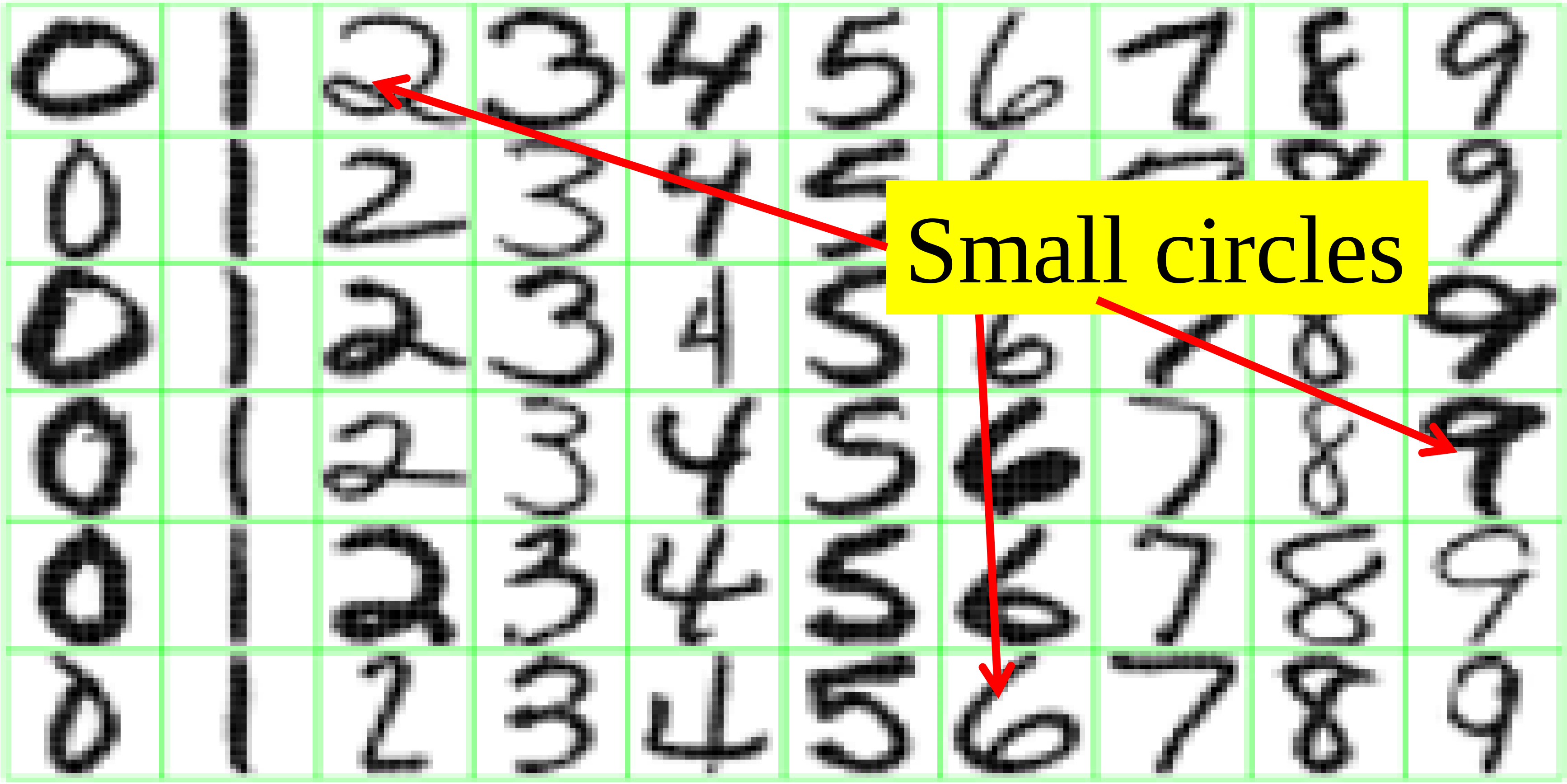
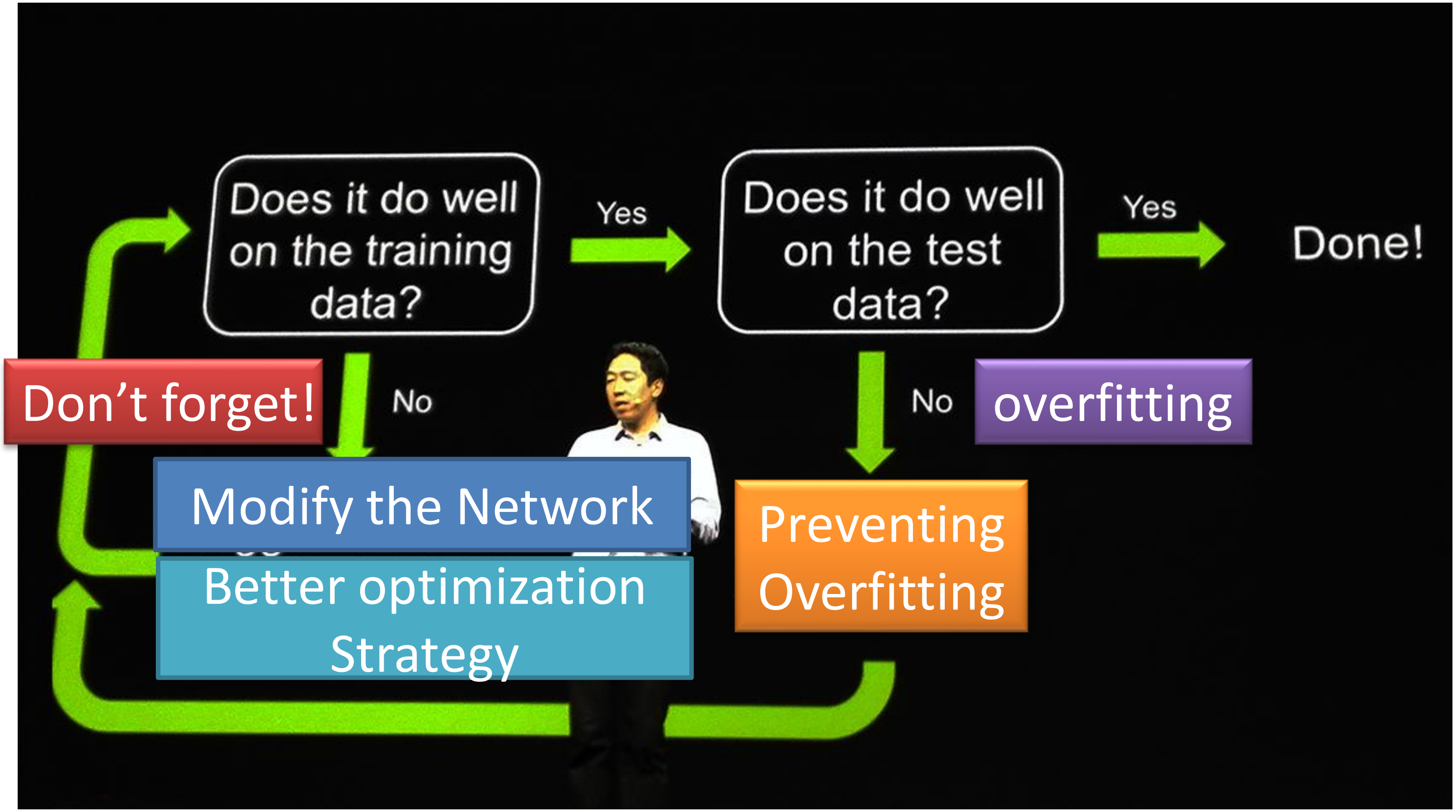


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

Suggestions while Training DNN

Recipe for Learning



http://www.gizmodo.com.au/2015/04/the-basic-recipe-for-machine-learning-explained-in-a-single-powerpoint-slide/

Recipe for Learning

Modify the Network

- New activation functions, for example, ReLU or Maxout

Better optimization Strategy

- Adaptive learning rates

Prevent Overfitting

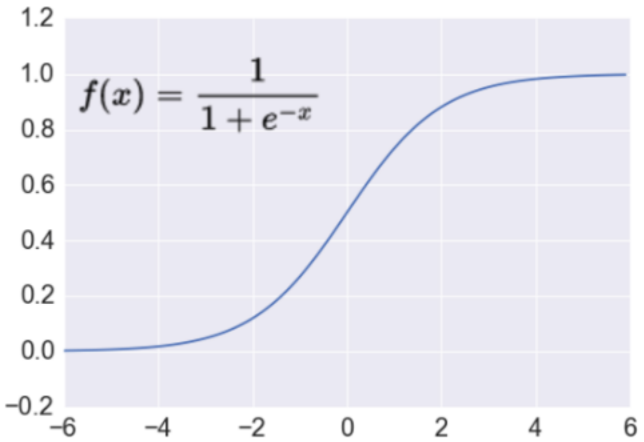
- Dropout

Only use this approach when you already obtained good results on the training data.

Suggestions while Training DNN

New Activation Function

Activation: Sigmoid



http://adilmoujahid.com/images/activation.png

Takes a real-valued number and "squashes" it into range between 0 and 1.

$$R^n \rightarrow [0,1]$$

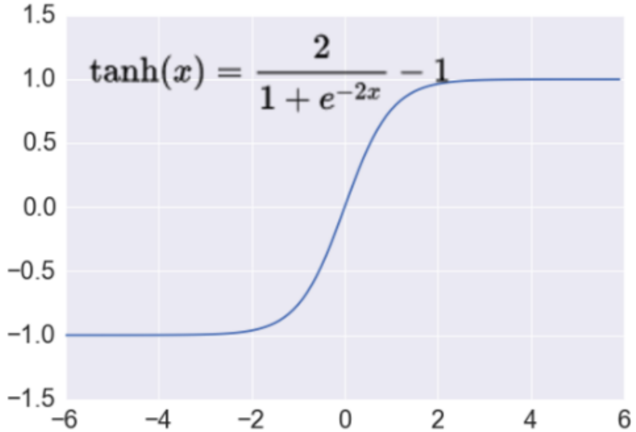
\+ Nice interpretation as the **firing rate** of a neuron

- 0 = not firing at all
- 1 = fully firing

\- Sigmoid neurons **saturate** and **kill gradients**, thus NN will barely learn

- when the neuron's activation are 0 or 1 (saturate)
  - ☹ gradient at these regions almost zero
  - ☹ almost no signal will flow to its weights
  - ☹ if initial weights are too large then most neurons would saturate

Activation: Tanh



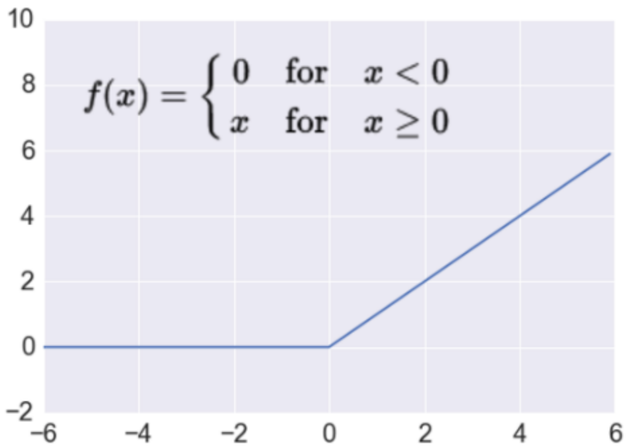
http://adilmoujahid.com/images/activation.png

Takes a real-valued number and “squashes” it into range between -1 and 1.

$$R^n \rightarrow [-1,1]$$

- Like sigmoid, tanh neurons **saturate**
- Unlike sigmoid, output is **zero-centered**
- Tanh is a **scaled sigmoid**: $\tanh(x) = 2sigm(2x) - 1$

Activation: ReLU



http://adilmoujahid.com/images/activation.png

Takes a real-valued number and thresholds it at zero $\quad f(x) = \max(0, x)$

$$R^n \to R^n_+$$

Most Deep Networks use ReLU nowadays

☺ Trains much **faster**

- accelerates the convergence of SGD
- due to linear, non-saturating form

☺ Less expensive operations

- compared to sigmoid/tanh (exponentials etc.)
- implemented by simply thresholding a matrix at zero

☺ More **expressive**

☺ Prevents the **gradient vanishing problem**

Tips for Training DNN

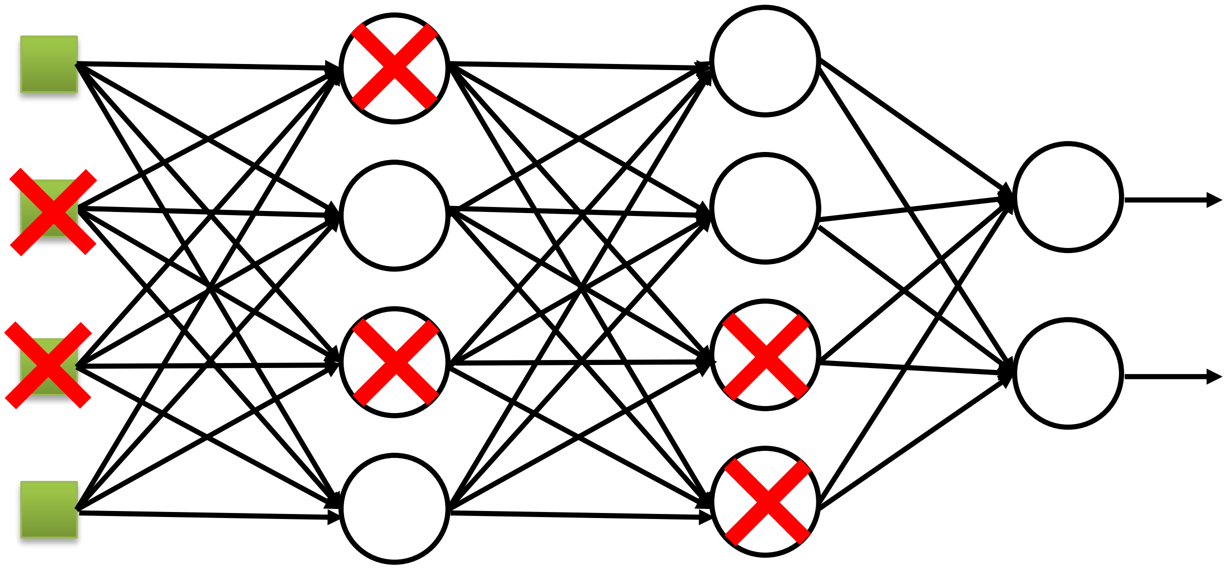
Dropout

Dropout

Pick a mini-batch

$$\theta^t \leftarrow \theta^{t-1} - \eta \nabla C(\theta^{t-1})$$

Training:



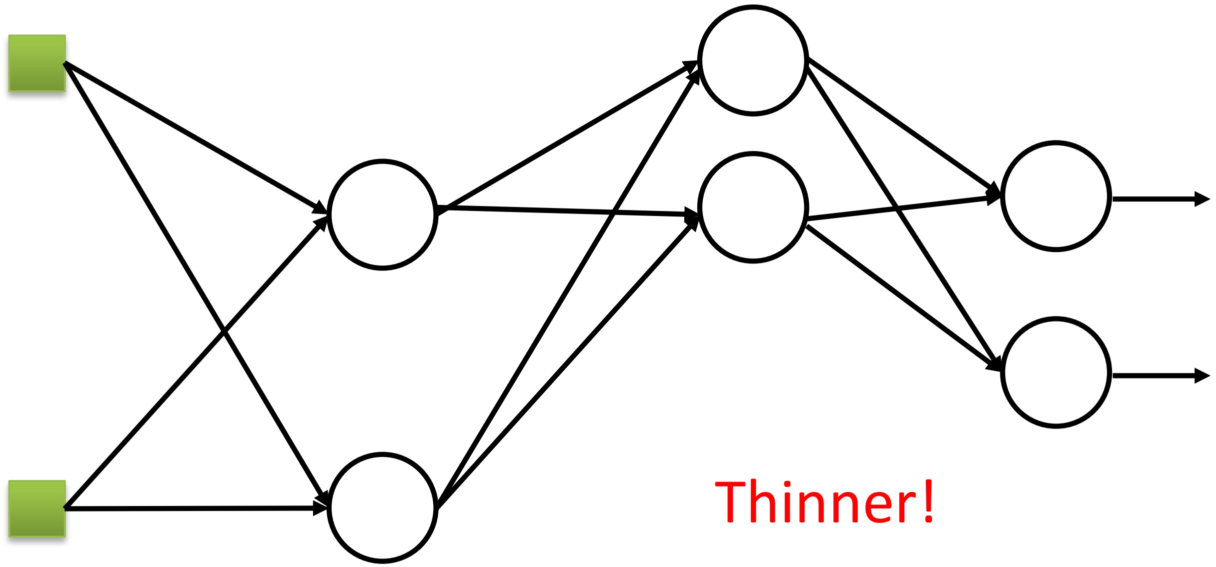
- **Each time before computing the gradients**
  - Each neuron has p% to dropout

Dropout

Pick a mini-batch

$$\theta^t \leftarrow \theta^{t-1} - \eta \nabla C(\theta^{t-1})$$

Training:

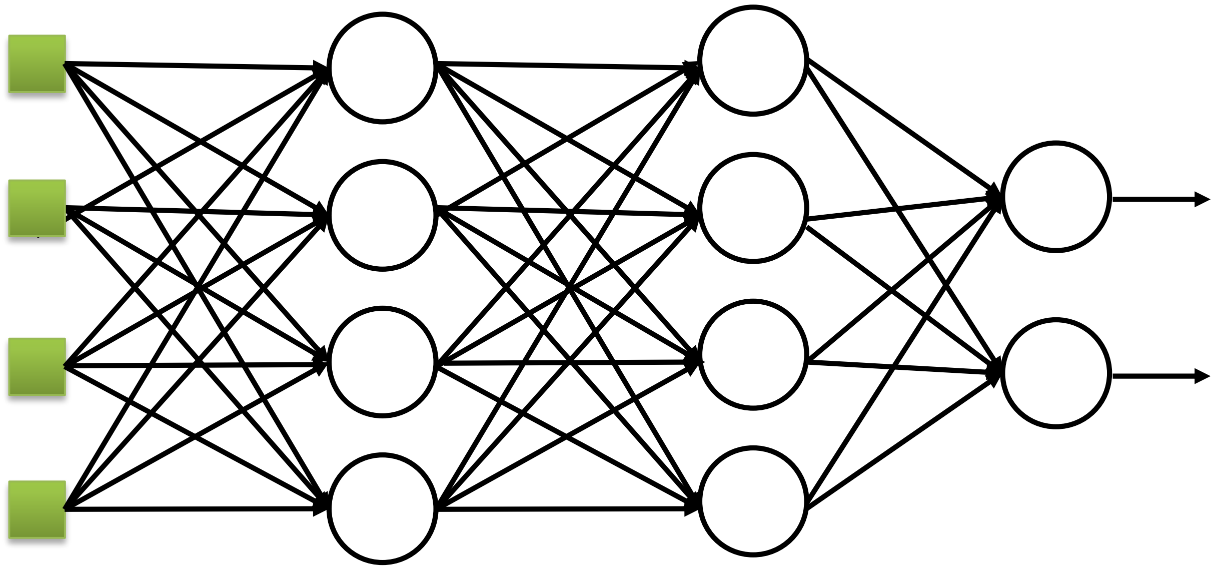


- **Each time before computing the gradients**
  - Each neuron has p% to dropout
    - **The structure of the network is changed.**
  - Using the new network for training

For each mini-batch, we resample the dropout neurons

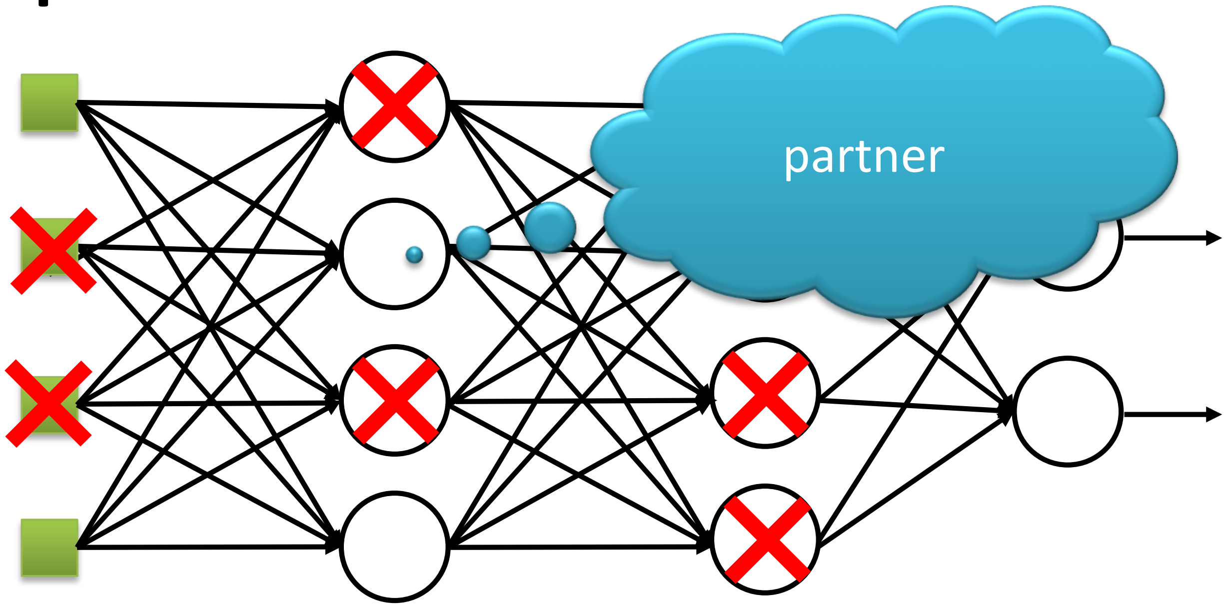
Dropout

Testing:



- **No dropout**
  - If the dropout rate at training is p%, all the weights times (1-p)%
  - Assume that the dropout rate is 50%. If a weight $w = 1$ by training, set $w = 0.5$ for testing.

Dropout - Intuitive Reason



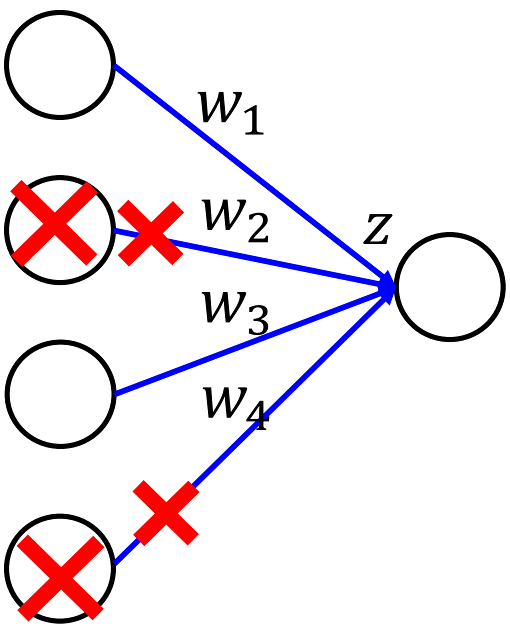
- When teams up, if everyone expect the partner will do the work, nothing will be done finally.
- However, if you know your partner will dropout, you will do better.
- When testing, no one dropout actually, so obtaining good results eventually.

Dropout - Intuitive Reason

- Why the weights should multiply (1-p)% (dropout rate) when testing?

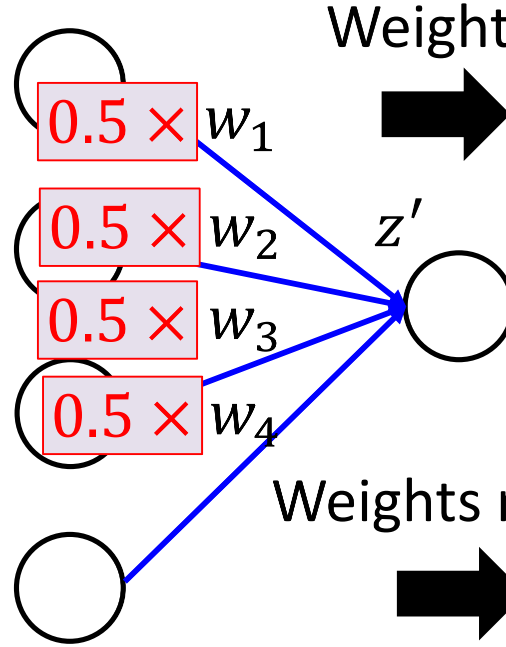
Training of Dropout

Assume dropout rate is 50%



Testing of Dropout

No dropout

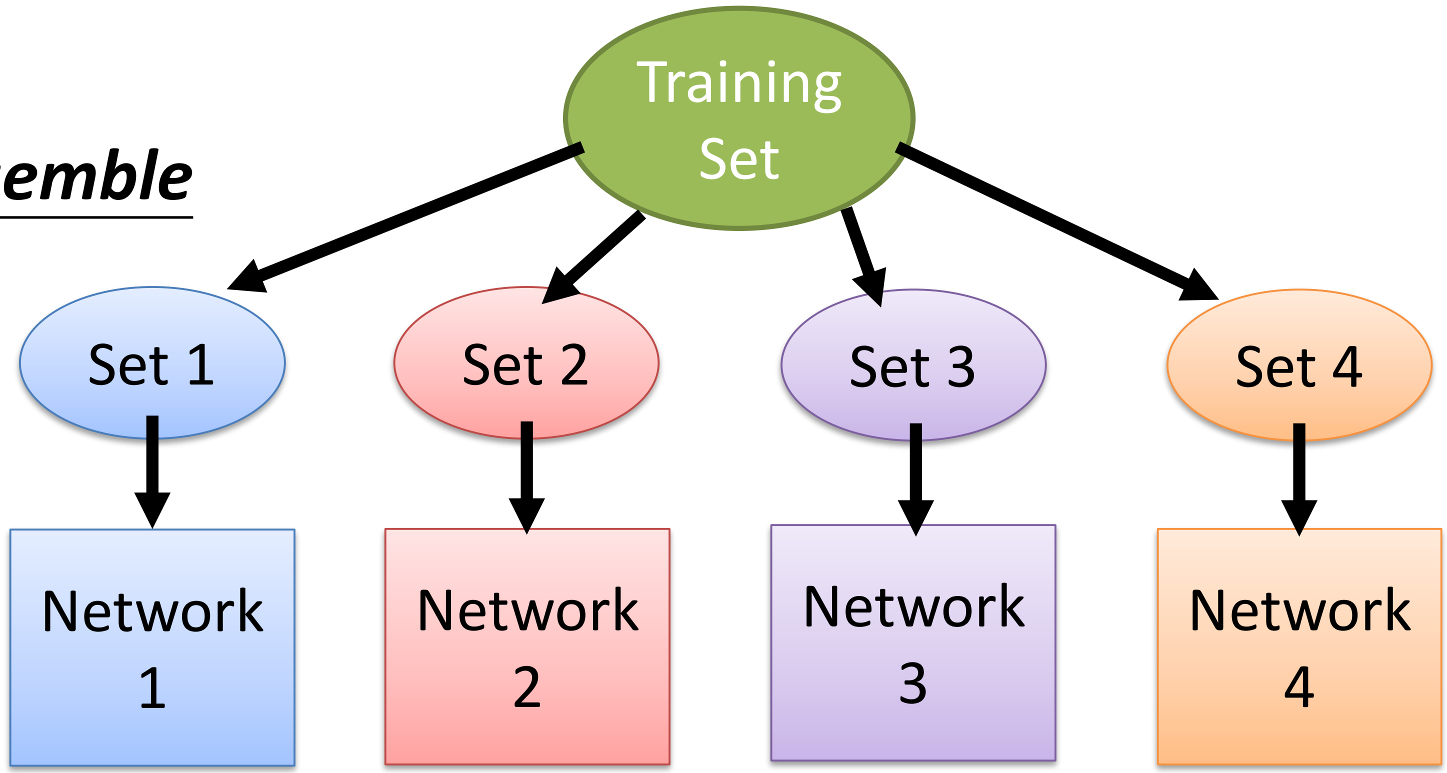


Weights from training $\Rightarrow$ $z' \approx 2z$

Weights multiply (1-p)% $\Rightarrow$ $z' \approx z$

Dropout is a kind of ensemble.

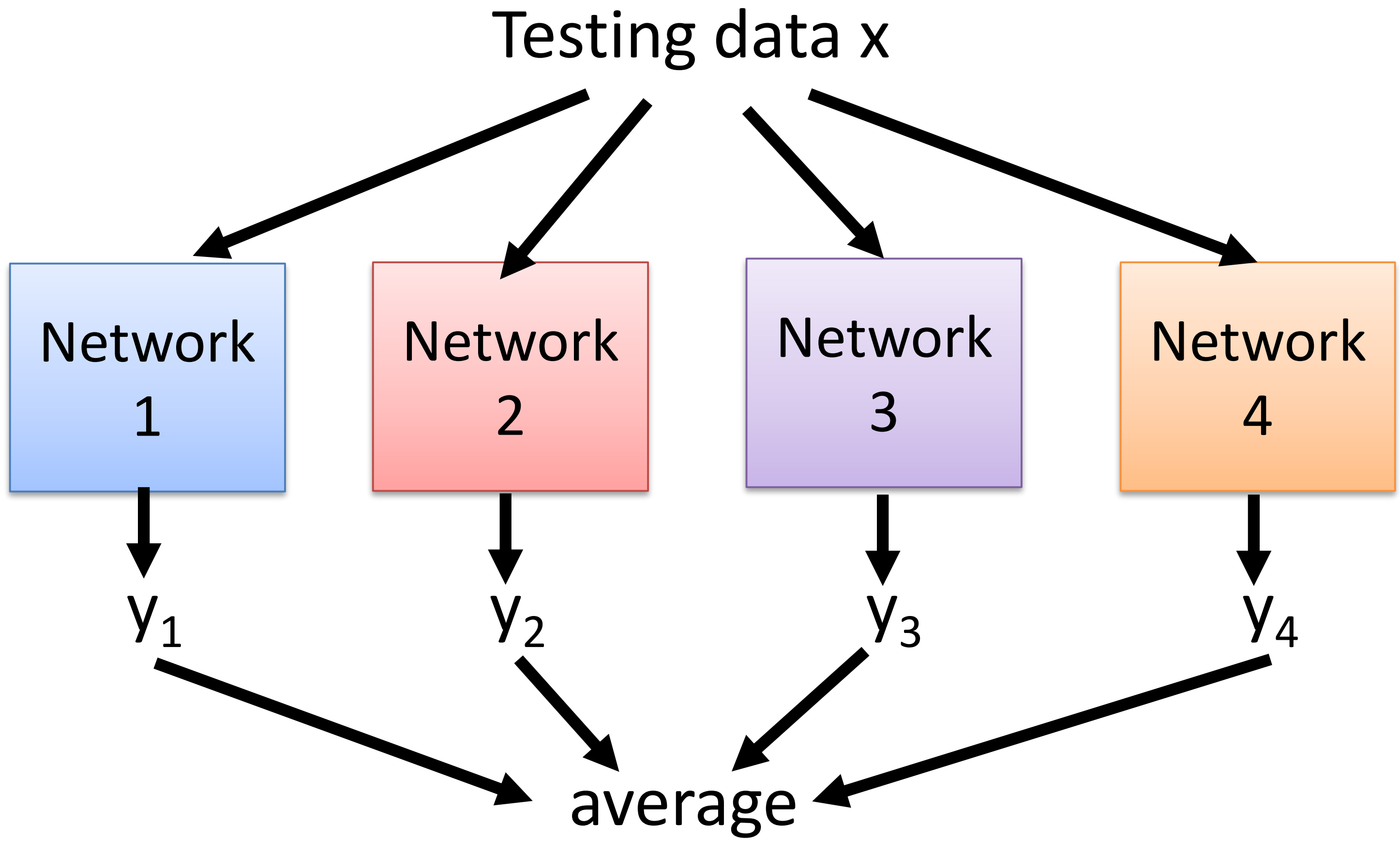
Ensemble



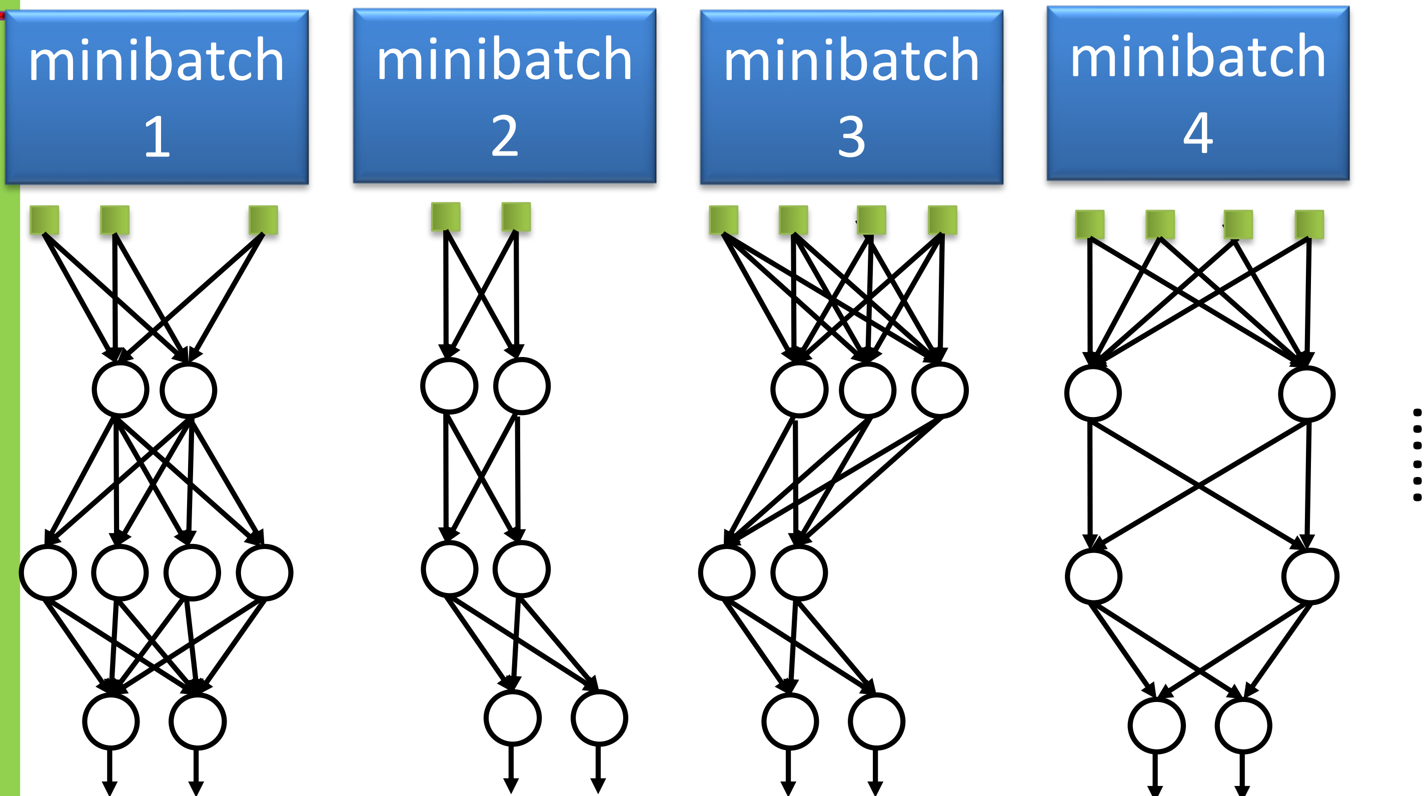
Train a bunch of networks with different structures

Dropout is a kind of ensemble.

Ensemble



Dropout is a kind of ensemble.



Training of Dropout

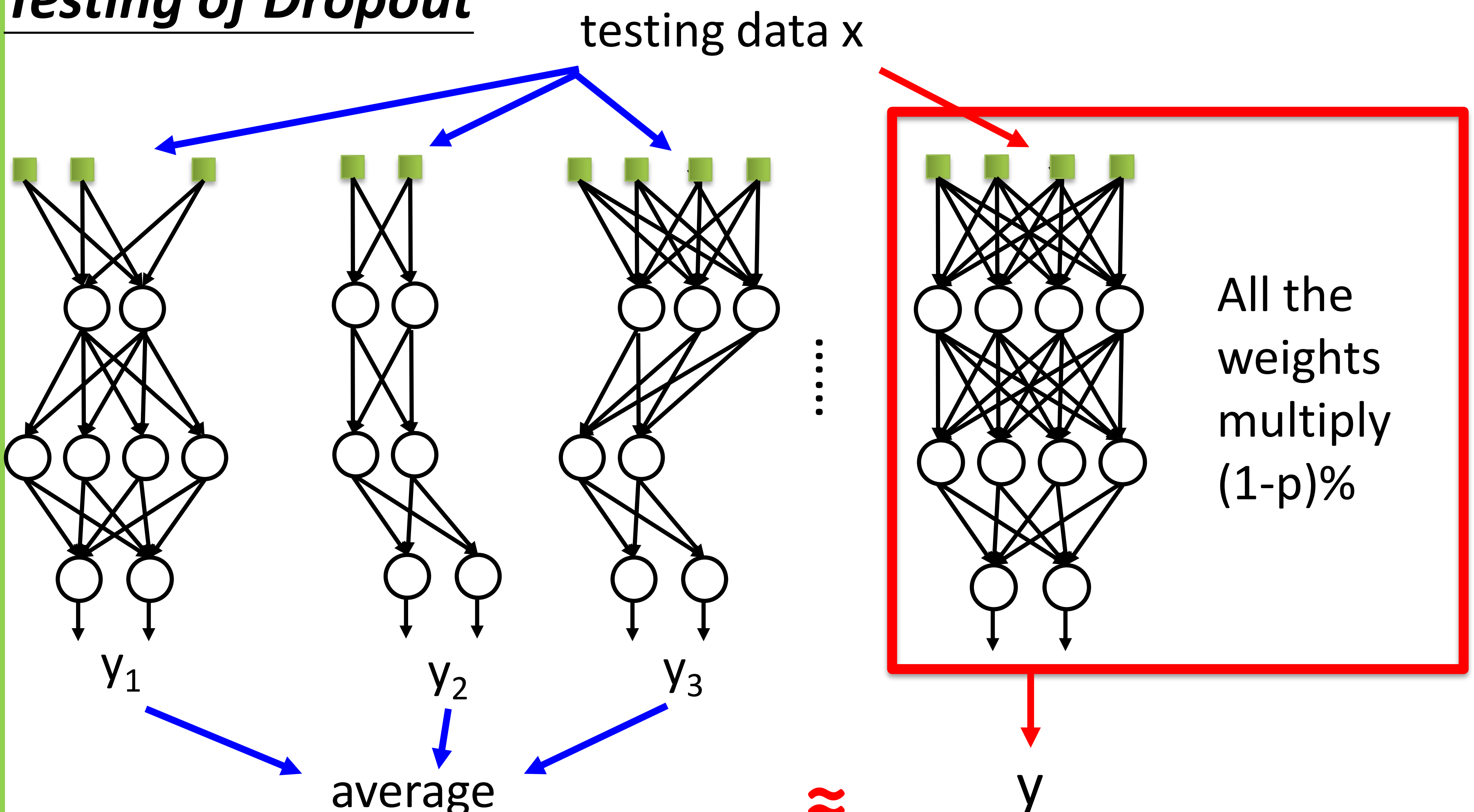
M neurons

2^M possible networks

- ➢Using one mini-batch to train one network
- ➢Some parameters in the network are shared

Dropout is a kind of ensemble.

Testing of Dropout



More about dropout

- More reference for dropout [Nitish Srivastava, JMLR'14] [Pierre Baldi, NIPS'13][Geoffrey E. Hinton, arXiv'12]
- Dropout works better with Maxout [Ian J. Goodfellow, ICML'13]
- Dropconnect [Li Wan, *ICML'13*]
  - Dropout delete neurons
  - Dropconnect deletes the connection between neurons
- Annealed dropout [S.J. Rennie, SLT'14]
  - Dropout rate decreases by epochs
- Standout [J. Ba, NISP'13]
  - Each neural has different dropout rate