Deep Learning

Murtaza Taj/Usman Nazir
murtaza.taj@lums.edu.pk
usman.nazir@bnu.edu.pk

Week 6 : Convolutional Neural Network
Thursday 14th Nov 2024

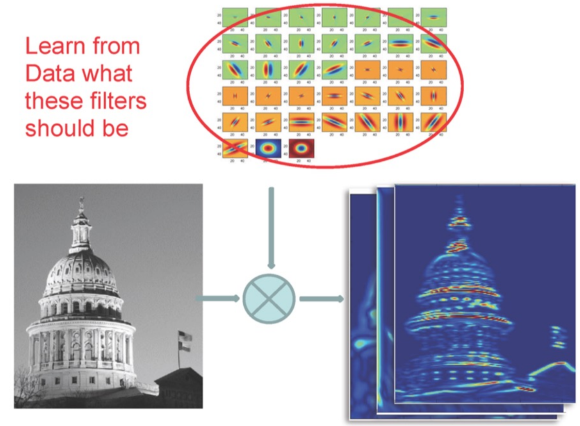
Vertical Edge Detection

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

Original Image: Opera

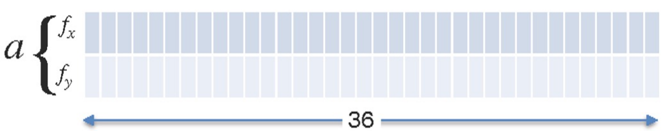
Inverted Image for Visualisation

CNN Key Idea 1: Filter are learned



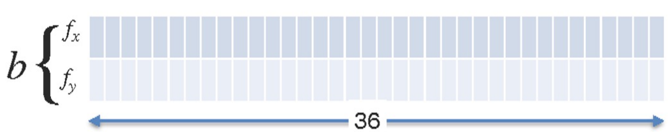
Texture representation: example

6x6

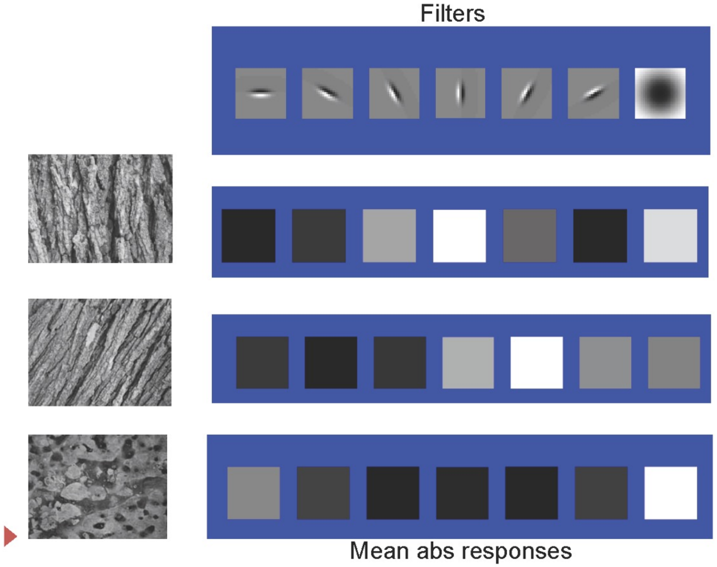


$$D(a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}$$
$$= \sqrt{\sum_{i=1}^{2} (a_i - b_i)^2}$$

6x6



Representing texture by mean abs response



Derek Hoiem

Neuron

$$\mathbf{w} = \{w_0, w_1, w_2, \cdots\}^T$$
$$\mathbf{x} = \{1, x_1, x_2, \cdots\}^T$$
$$y = \mathbf{w}^T\mathbf{x} = \sum w_j x_j$$

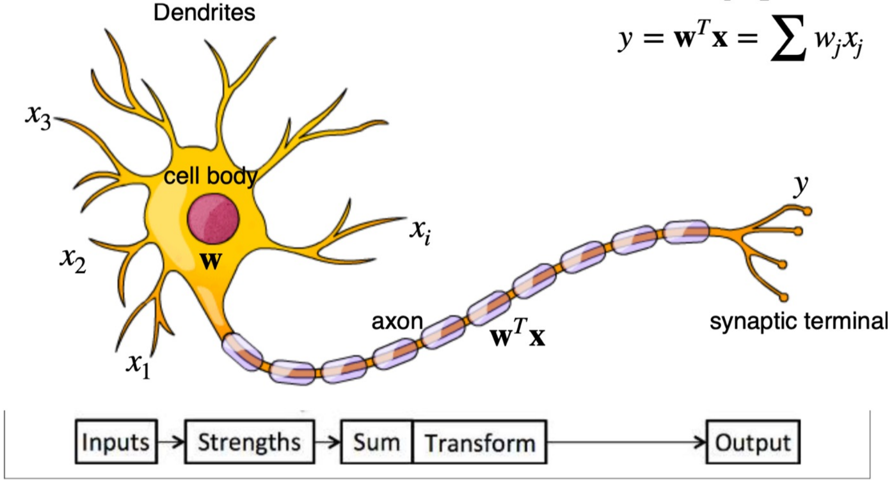
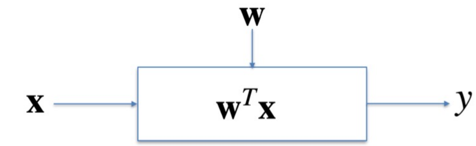
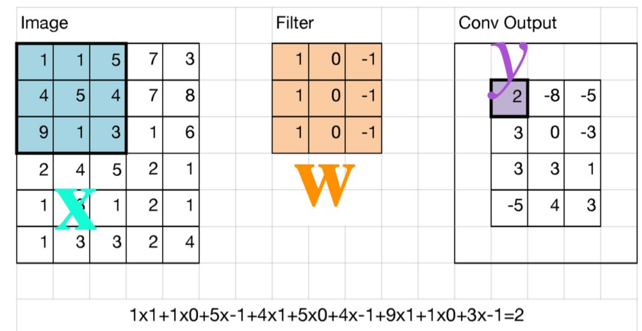


Figure 1-6. A functional description of a biological neuron's structure

Recap: Single Step of Convolution

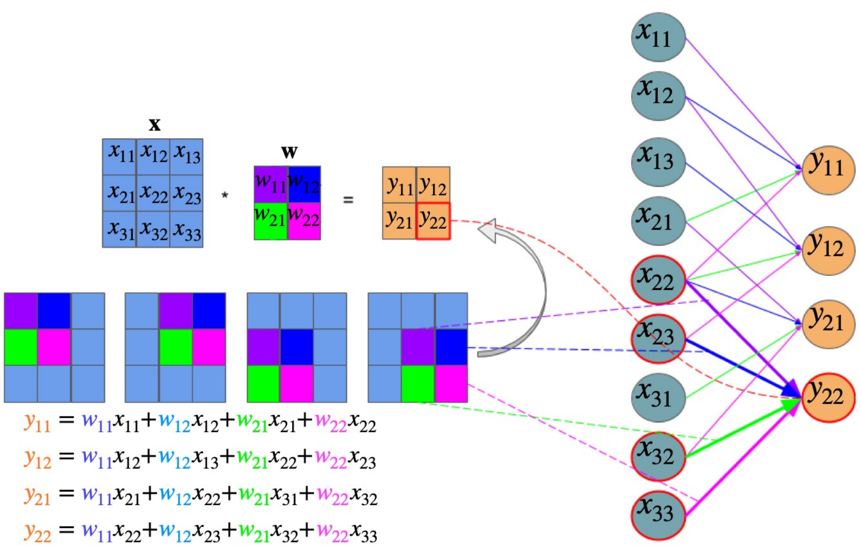
$$\mathbf{w} = \{w_0, w_1, w_2, \cdots\}^T$$
$$\mathbf{x} = \{1, x_1, x_2, \cdots\}^T$$
$$y = \mathbf{w}^T\mathbf{x} = \sum w_j x_j$$





$$y = \mathbf{w}^T\mathbf{x}$$

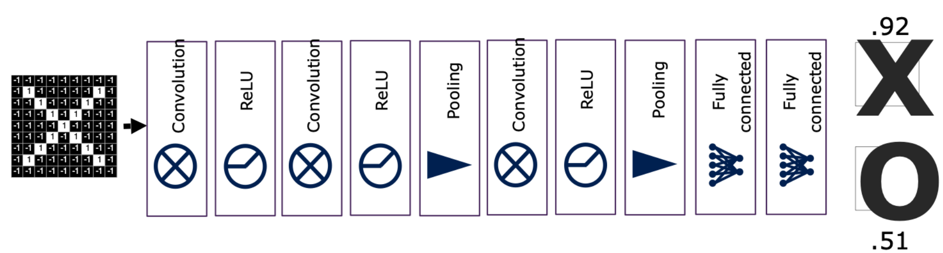
Feedforward in CNN is identical with convolution operation



https://grzegorzgwardys.wordpress.com/2016/04/22/

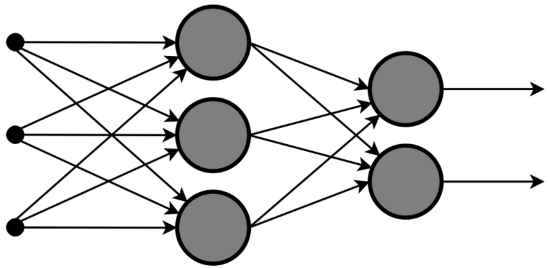
Putting it all together

A set of pixels becomes a set of votes.

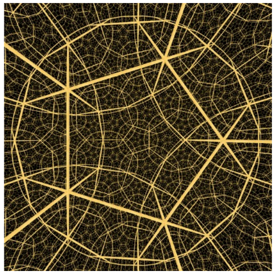


Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

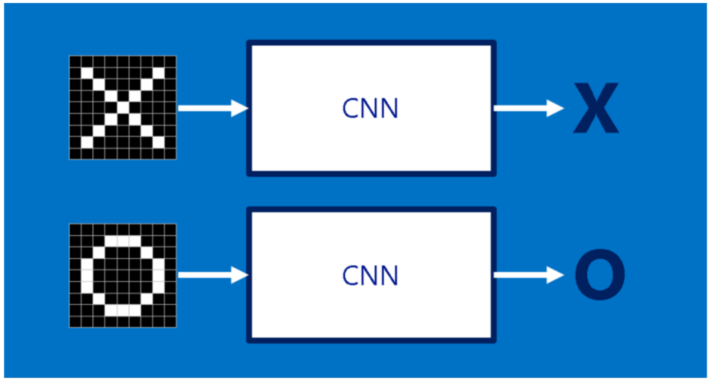
Overview



Recap Neural Networks

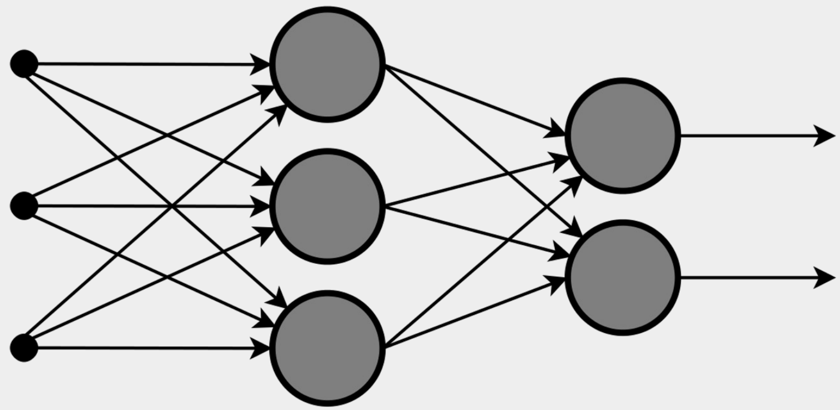


Erlangen Programme of Neural Networks

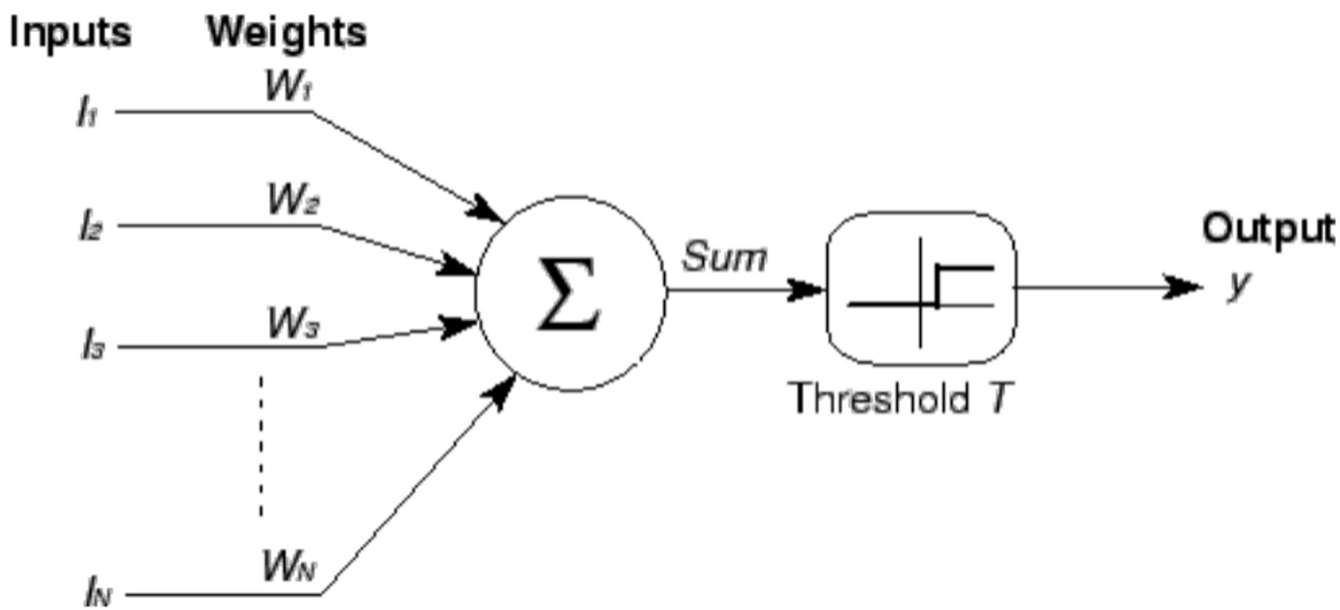


Toy ConvNet

History of Neural Networks

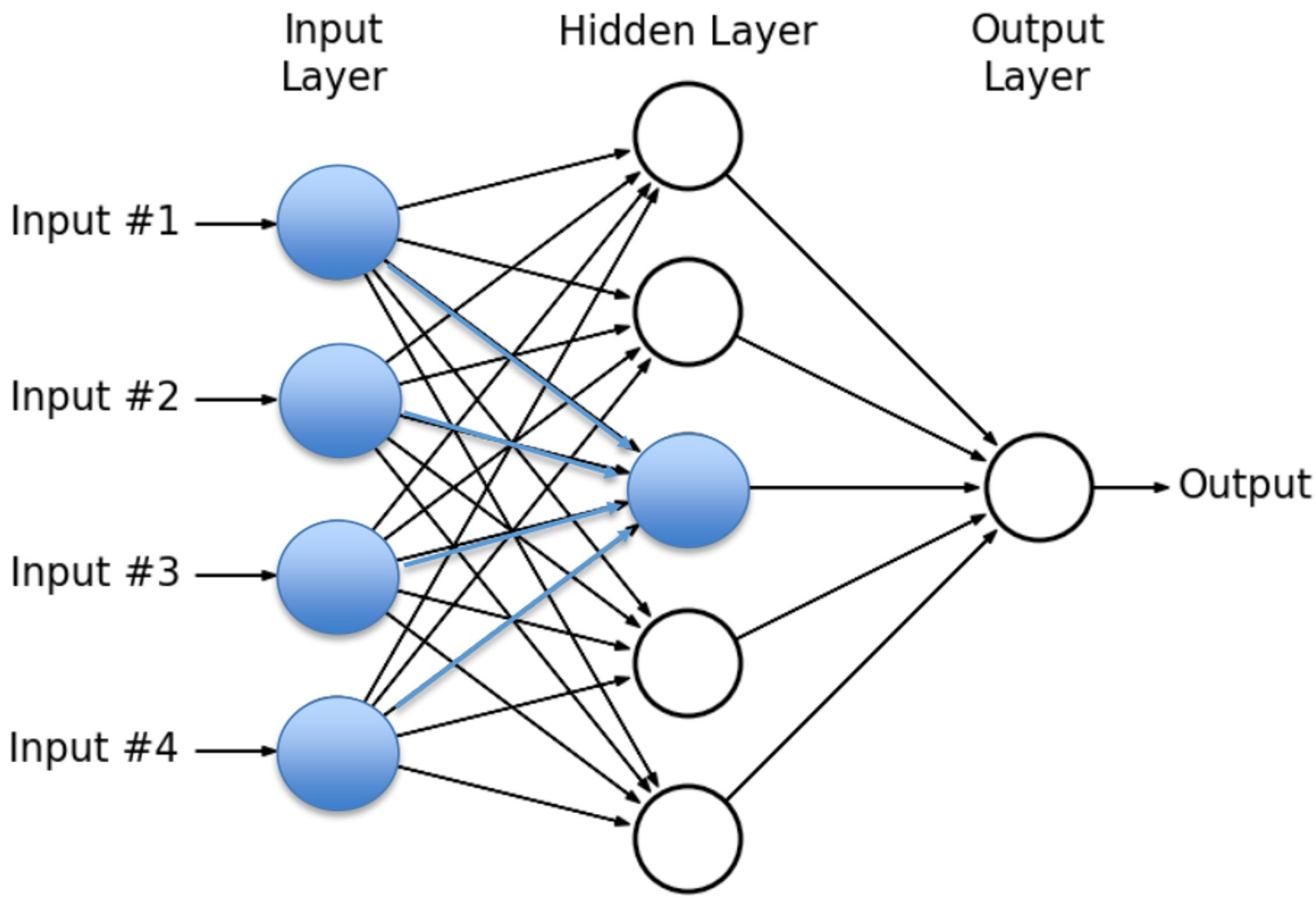


70 Years Ago



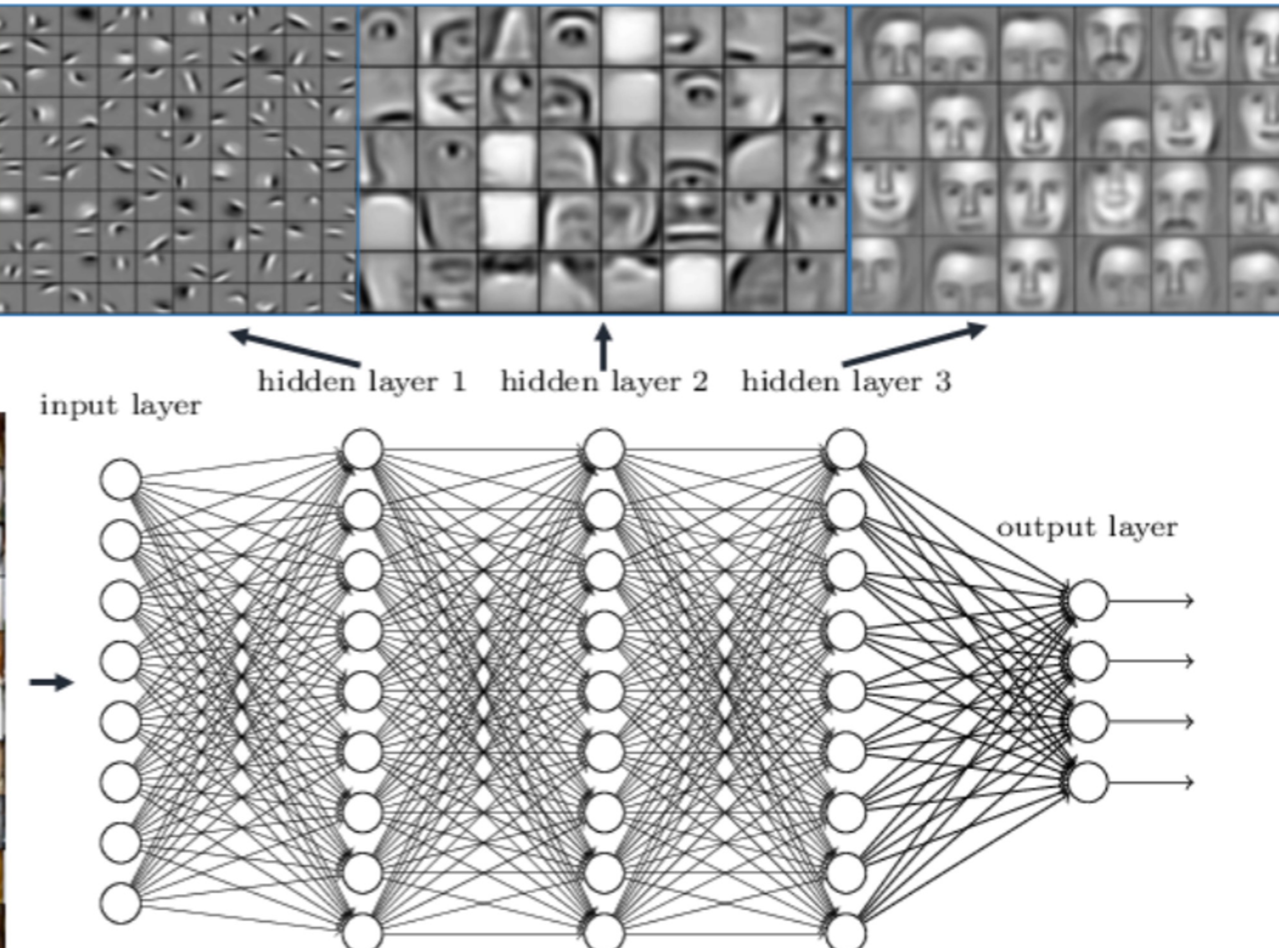
McCullogh & Pitts 1943
Rosenblatt 1957

50 Years Ago

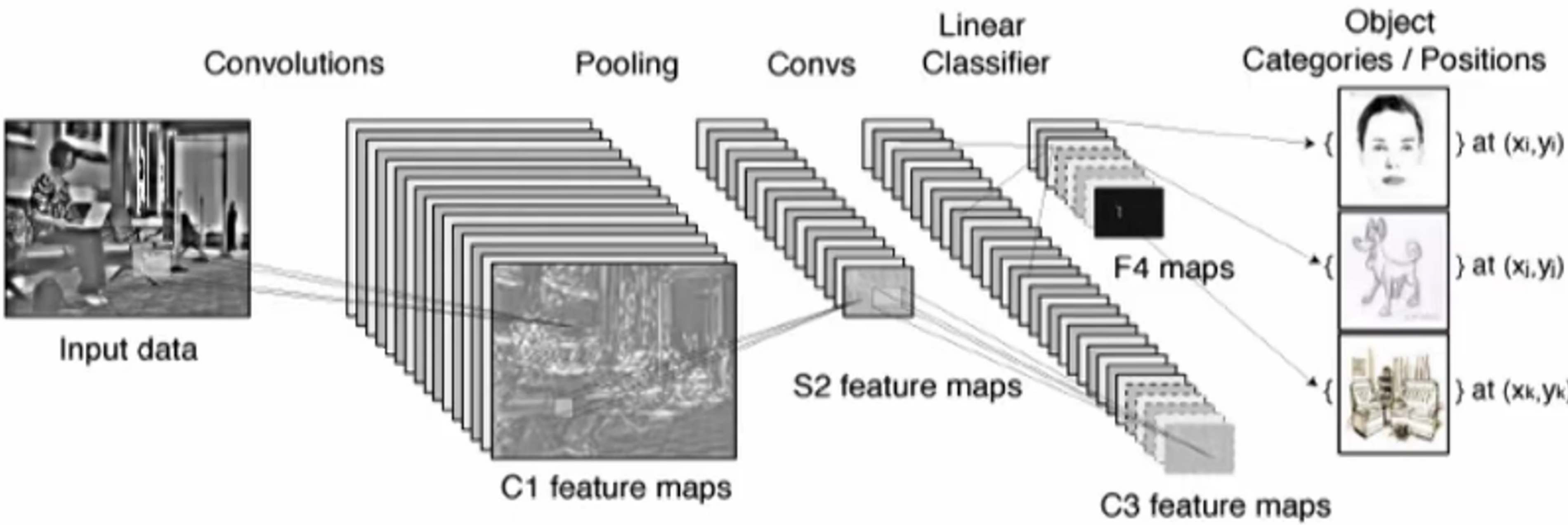

Input Layer
Hidden Layer
Output Layer
Input #1
Input #2
Input #3
Input #4
Output

10 Years Ago

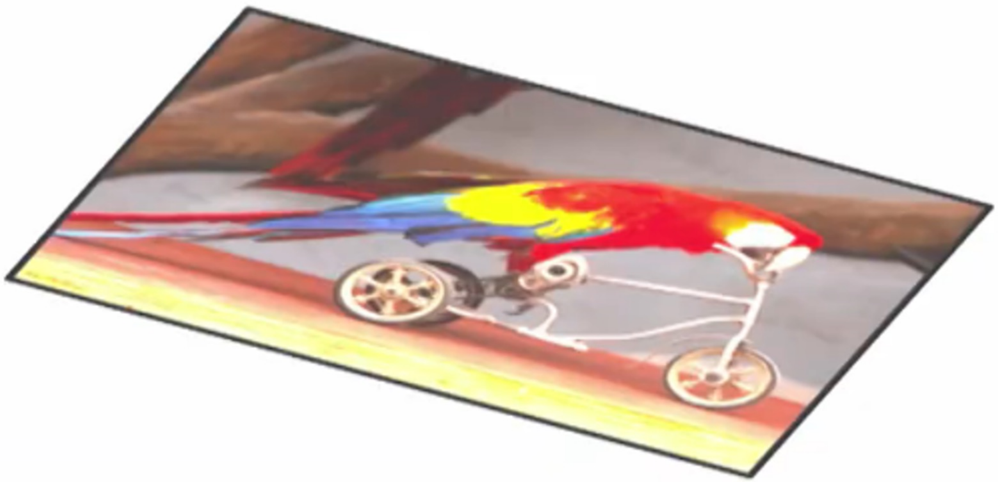
Deep neural networks learn hierarchical feature representations



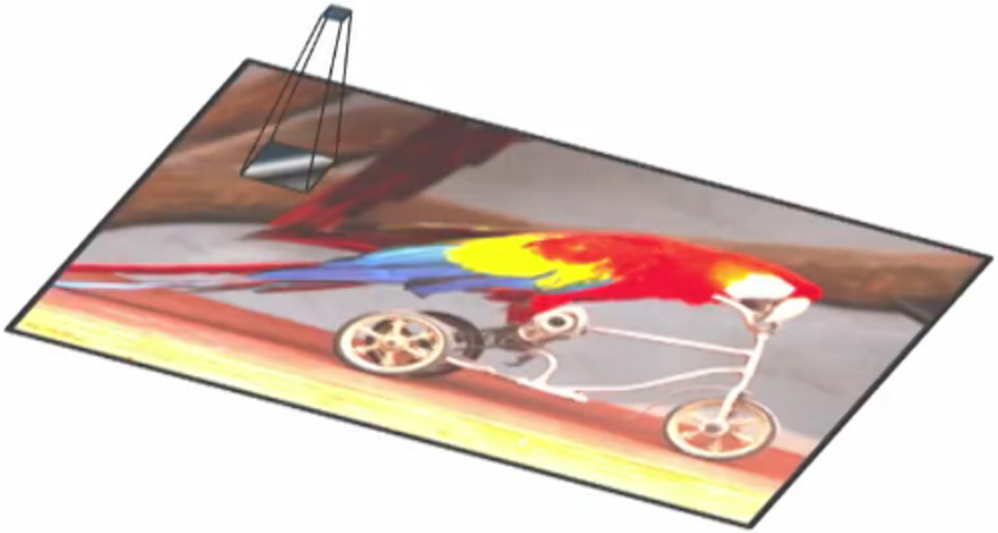
Forward: Filter, subsample, filter, nonlinearity, subsample, ..., classify



Backward: backpropagation (propagate error signal backward)

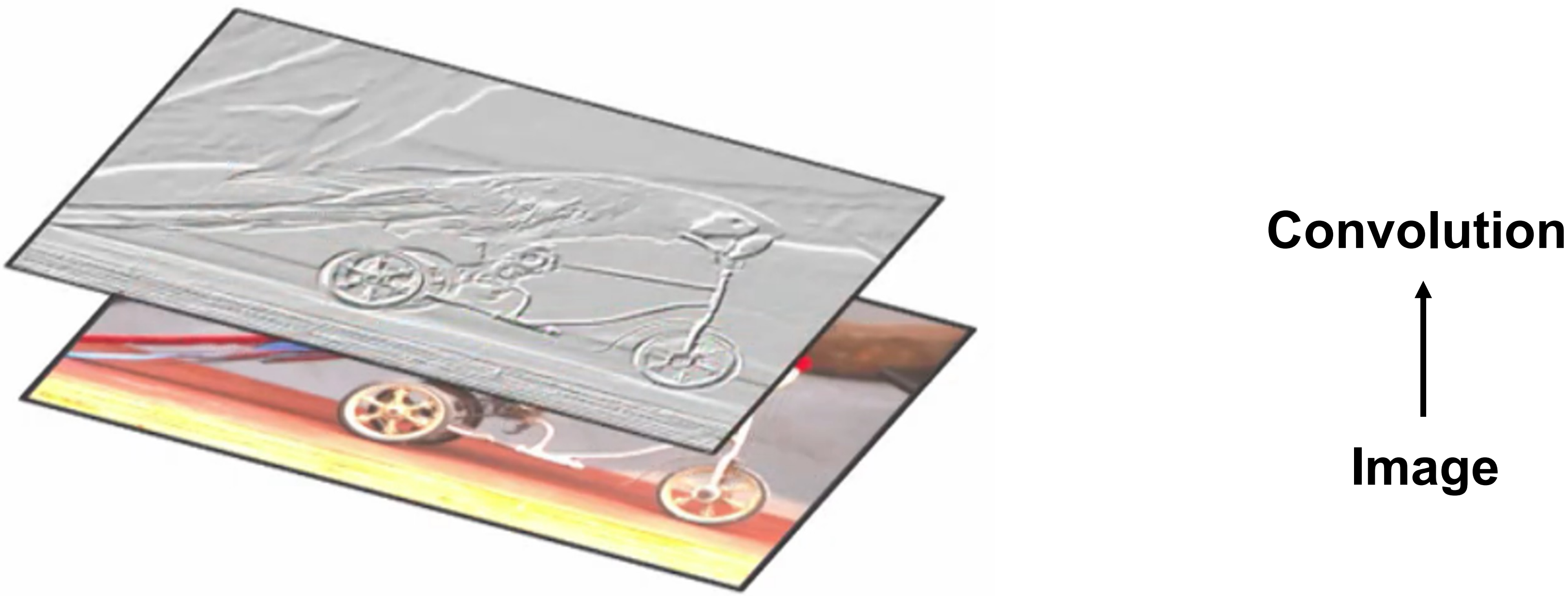


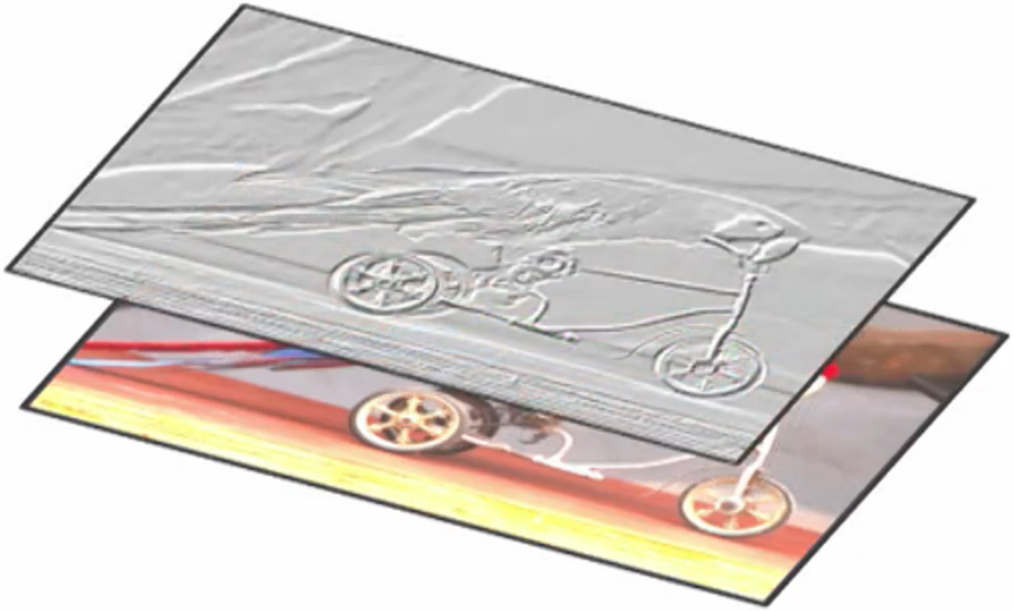
Image



Convolution

Image

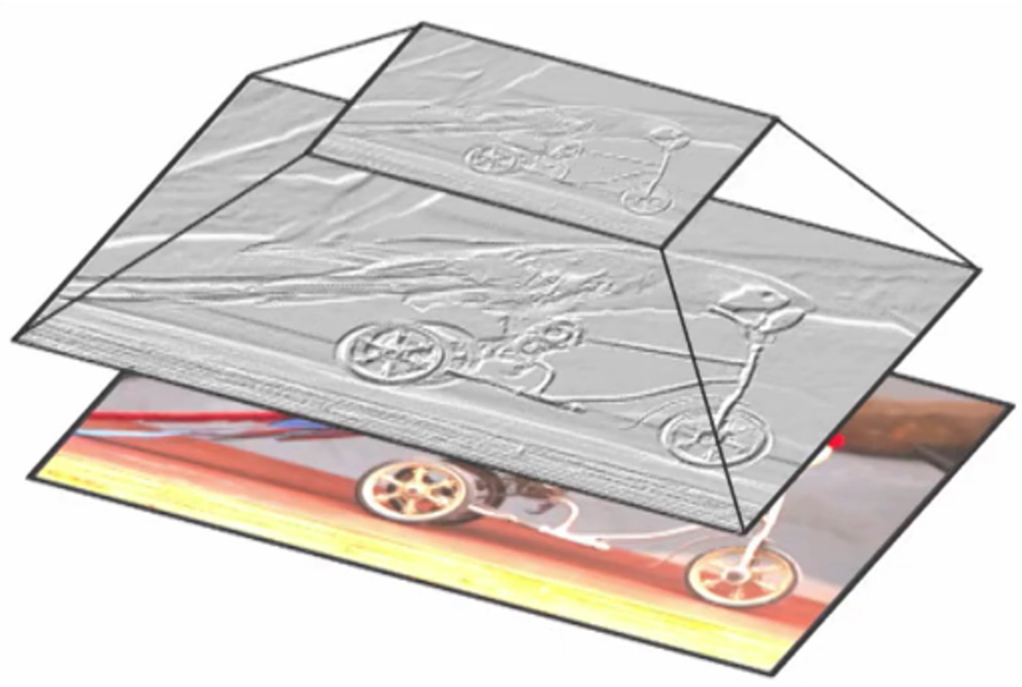

Convolution
Image



Pooling

Convolution

Image



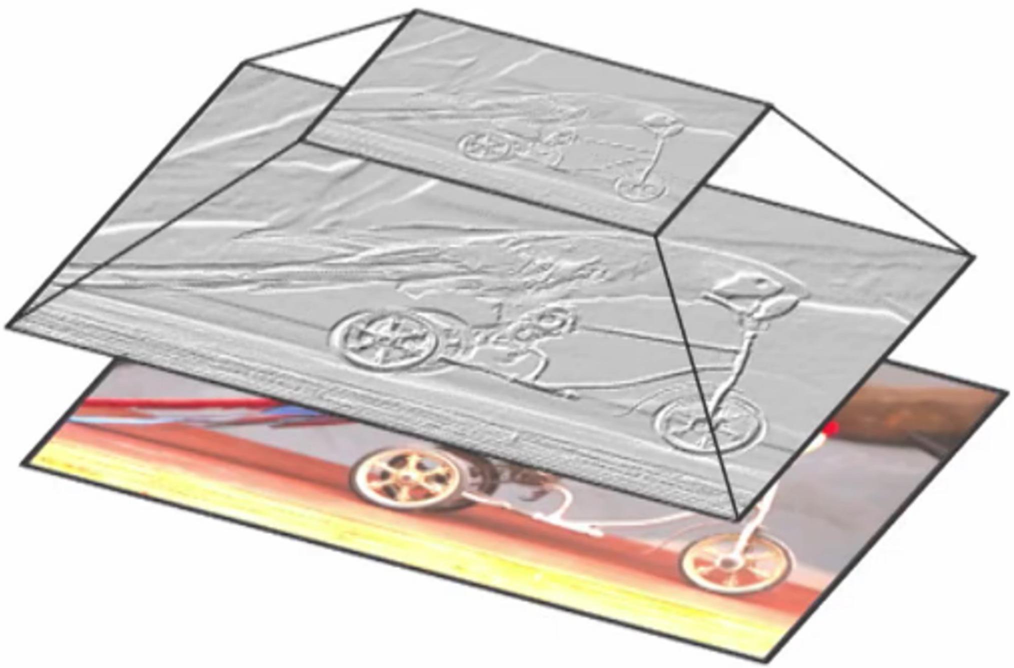
Pooling

↑

Convolution

↑

Image



Nonlinearity

↑

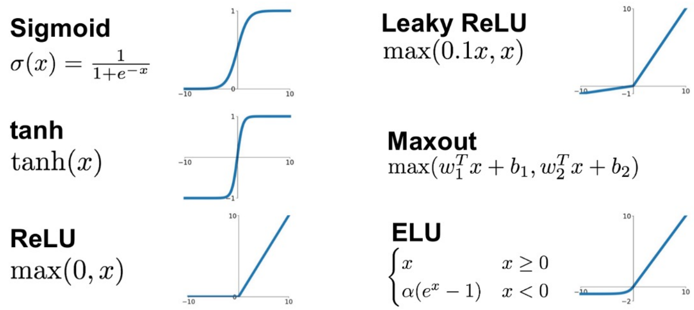
Pooling

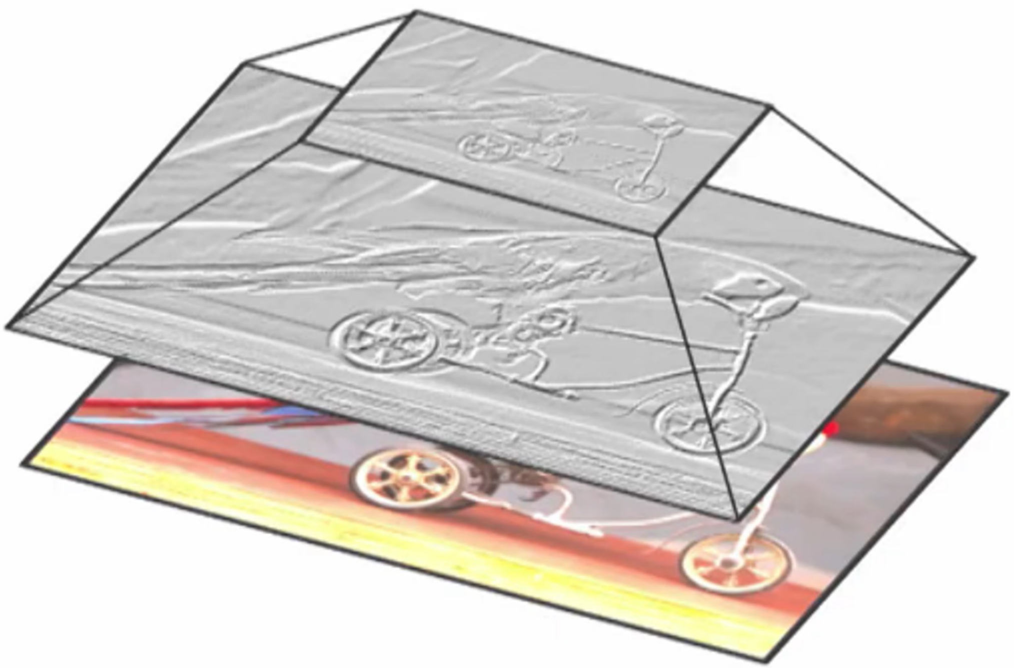
↑

Convolution

↑

Image





Nonlinearity

↑

Pooling

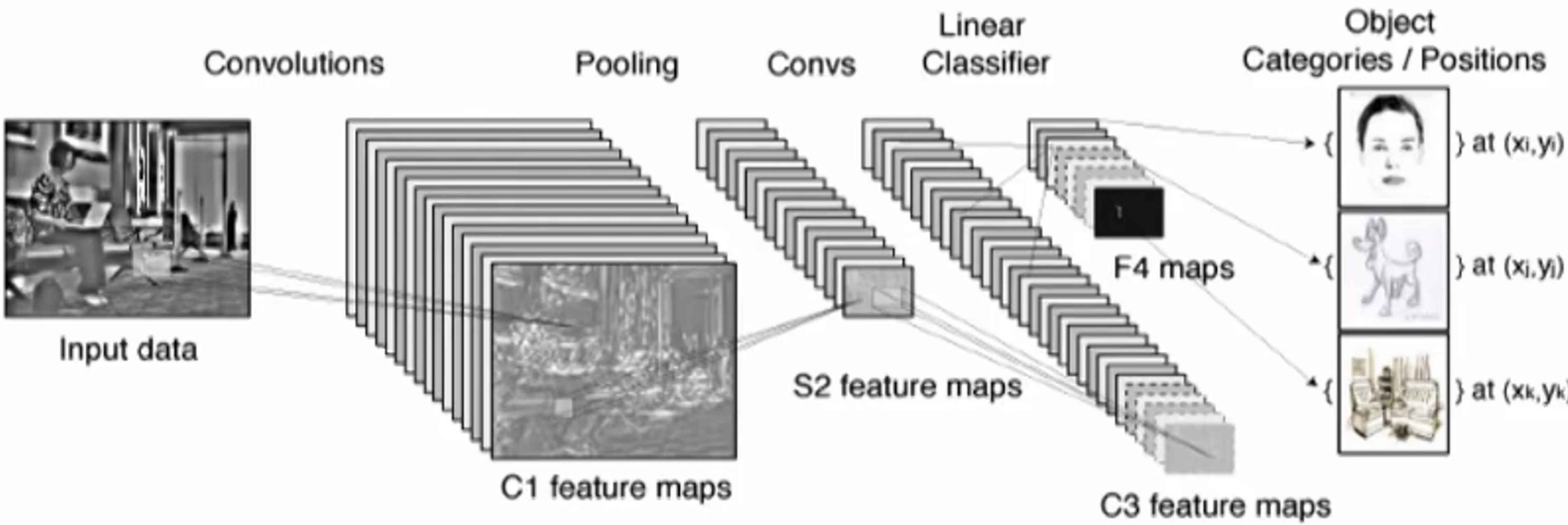
↑

Convolution

↑

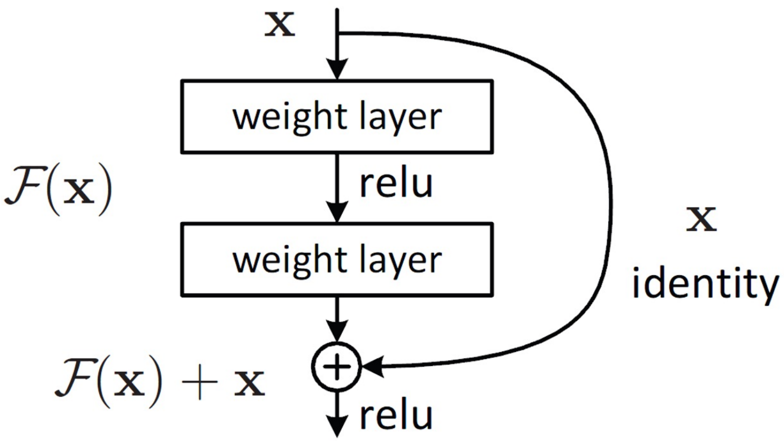
Image

Forward: Filter, subsample, filter, nonlinearity, subsample, …., classify

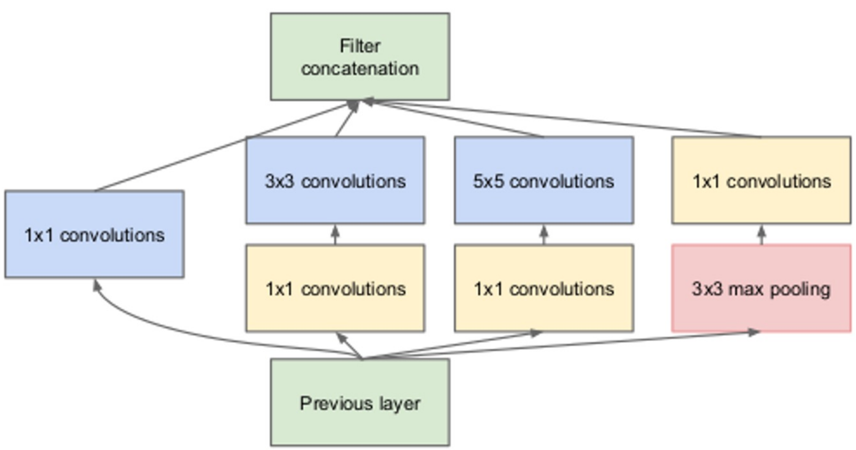


Backward: backpropagation (propagate error signal backward)

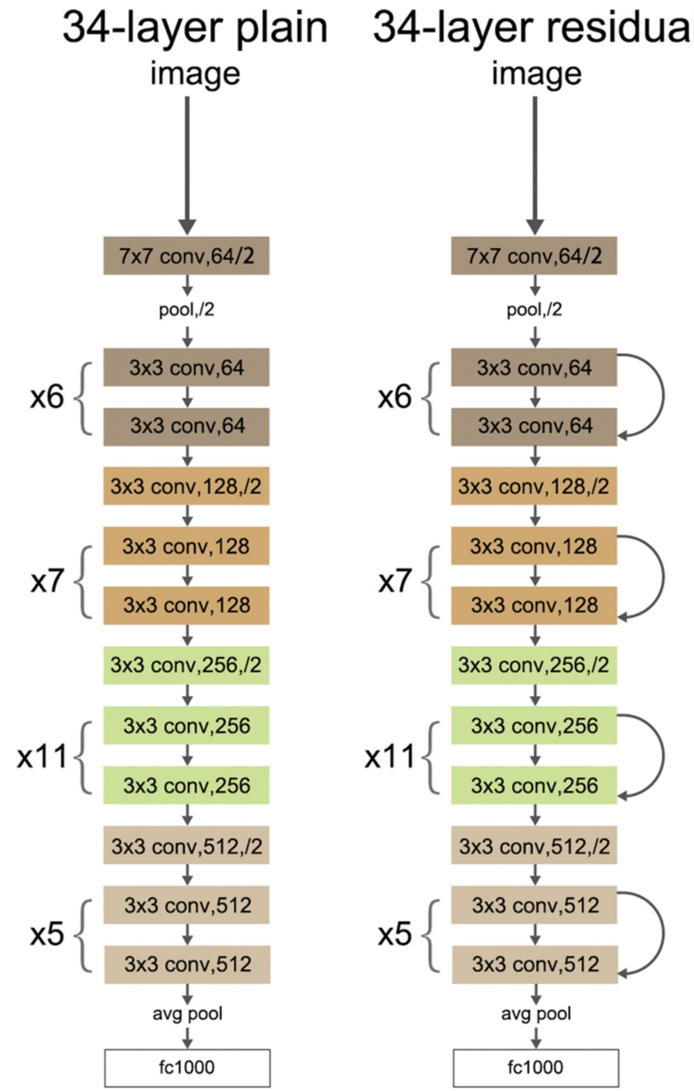
7 Year Ago



$\mathbf{x}$
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
$\mathbf{x}$
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

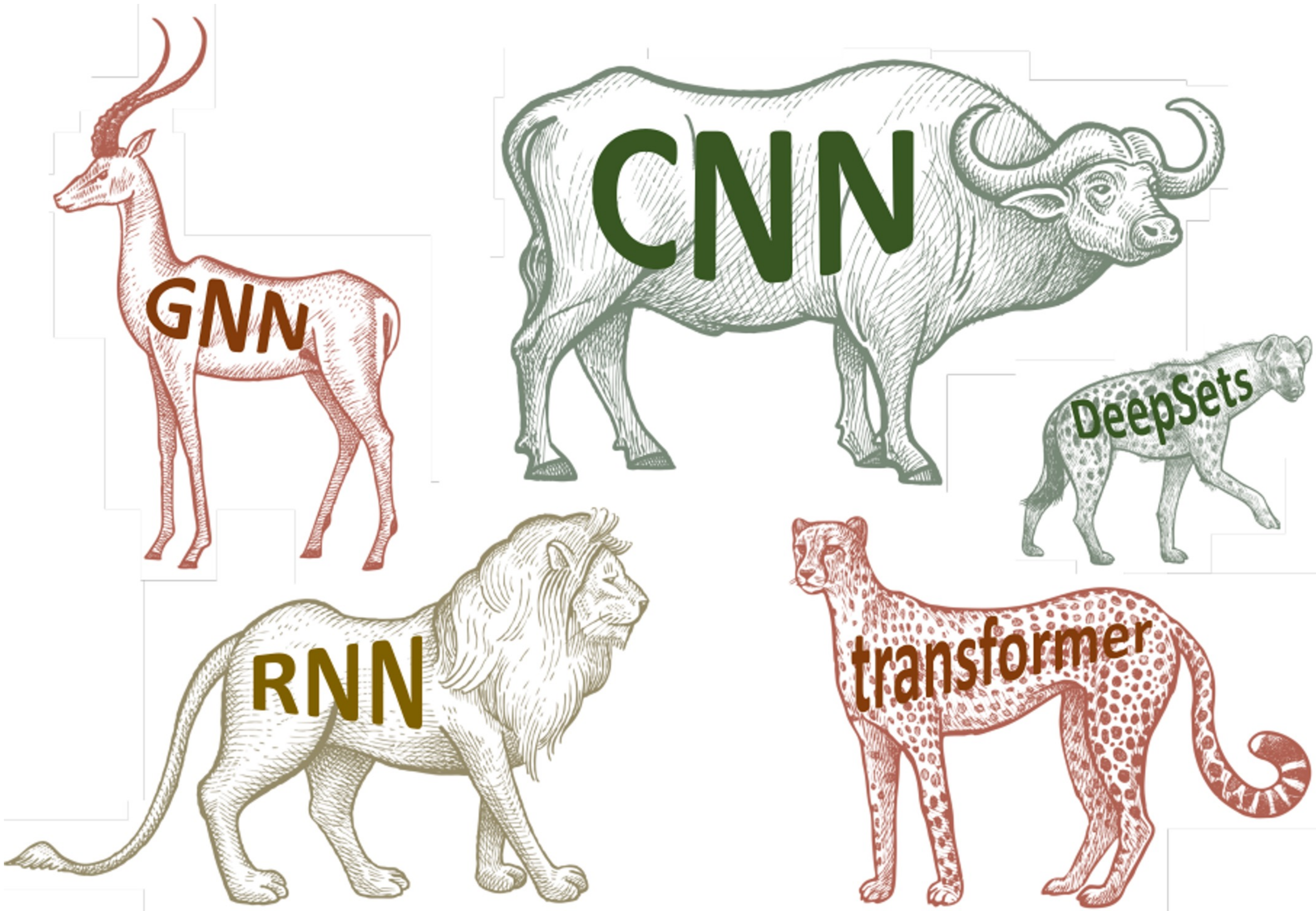


Filter concatenation
1x1 convolutions
3x3 convolutions
5x5 convolutions
1x1 convolutions
1x1 convolutions
1x1 convolutions
3x3 max pooling
Previous layer



34-layer plain
image
7x7 conv,64/2
pool,/2
x6
3x3 conv,64
3x3 conv,64
3x3 conv,128,/2
x7
3x3 conv,128
3x3 conv,128
3x3 conv,256,/2
x11
3x3 conv,256
3x3 conv,256
3x3 conv,512,/2
x5
3x3 conv,512
3x3 conv,512
avg pool
fc1000
34-layer residual
image
7x7 conv,64/2
pool,/2
x6
3x3 conv,64
3x3 conv,64
3x3 conv,128,/2
x7
3x3 conv,128
3x3 conv,128
3x3 conv,256,/2
x11
3x3 conv,256
3x3 conv,256
3x3 conv,512,/2
x5
3x3 conv,512
3x3 conv,512
avg pool
fc1000

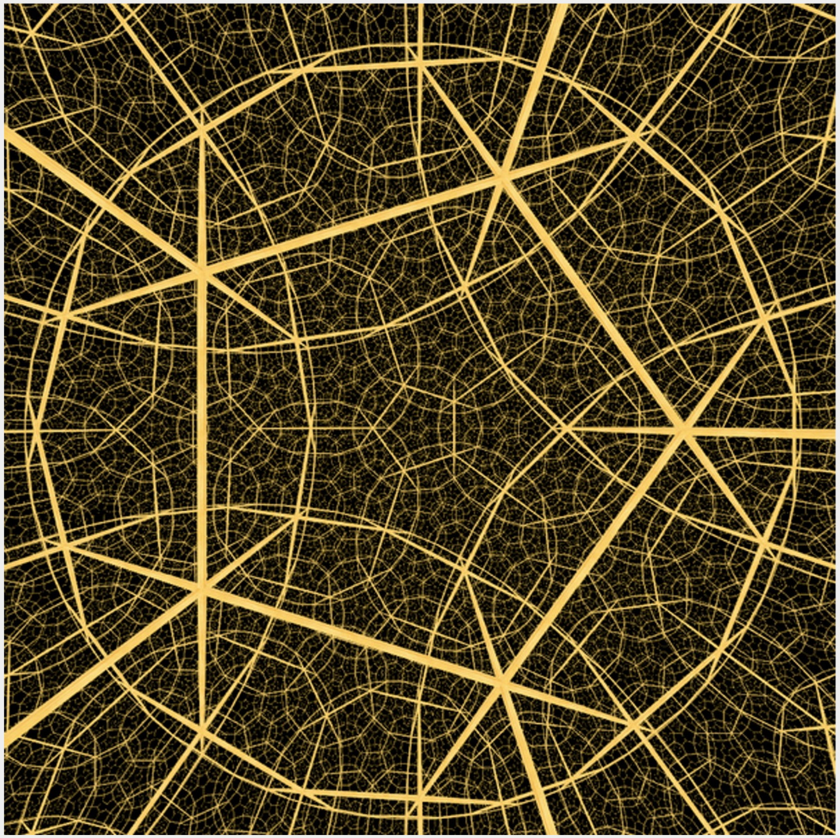
Many Architectures, Few Principles



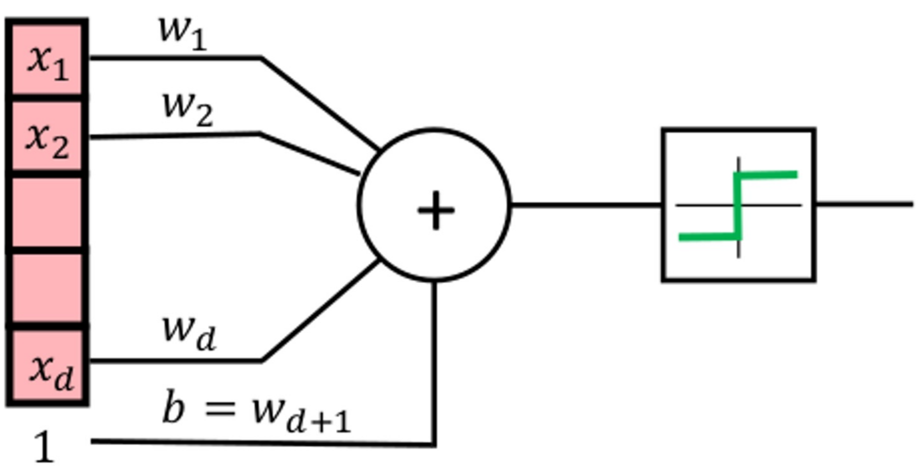
?

Unification of Computer Vision and Pattern Recognition from Symmetry and Equivariance

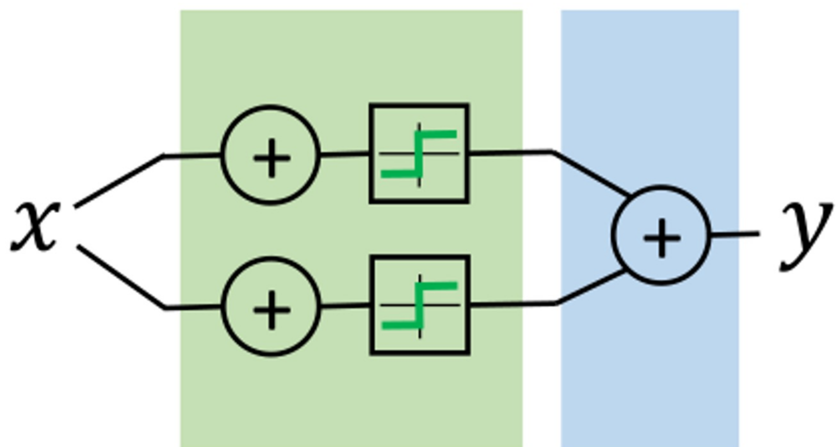
The Erlangen Programme of Neural Networks



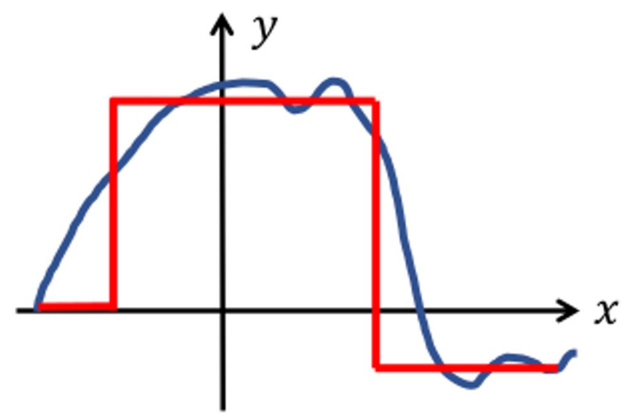
Simplest Neural Network



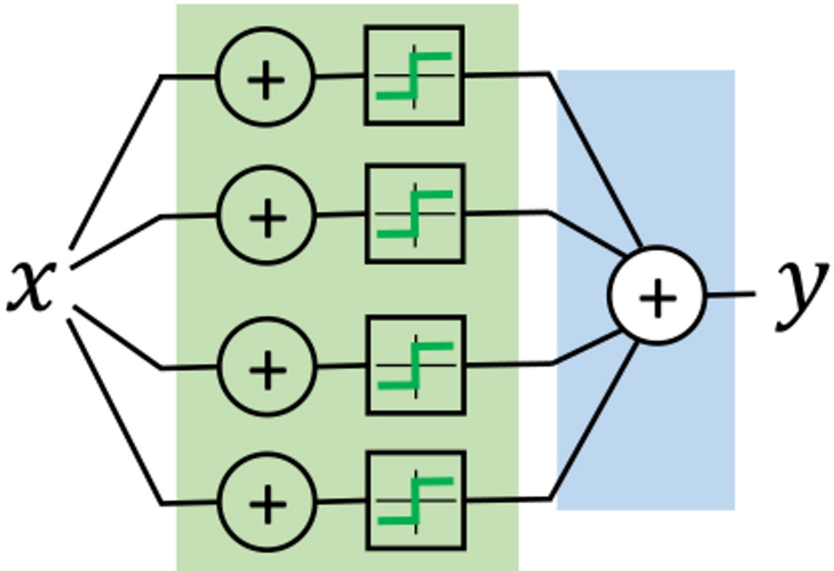
$$y = \mathrm{sign}\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}\right)$$

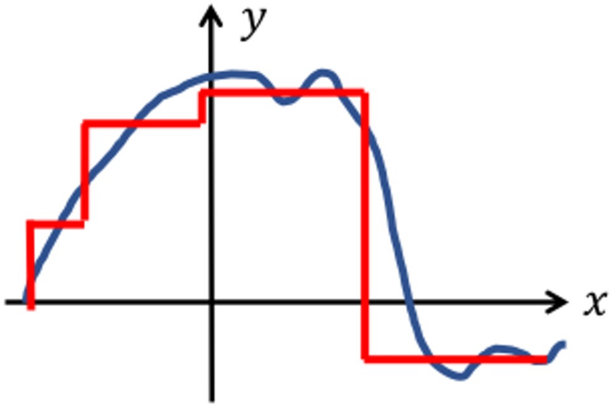


x
y

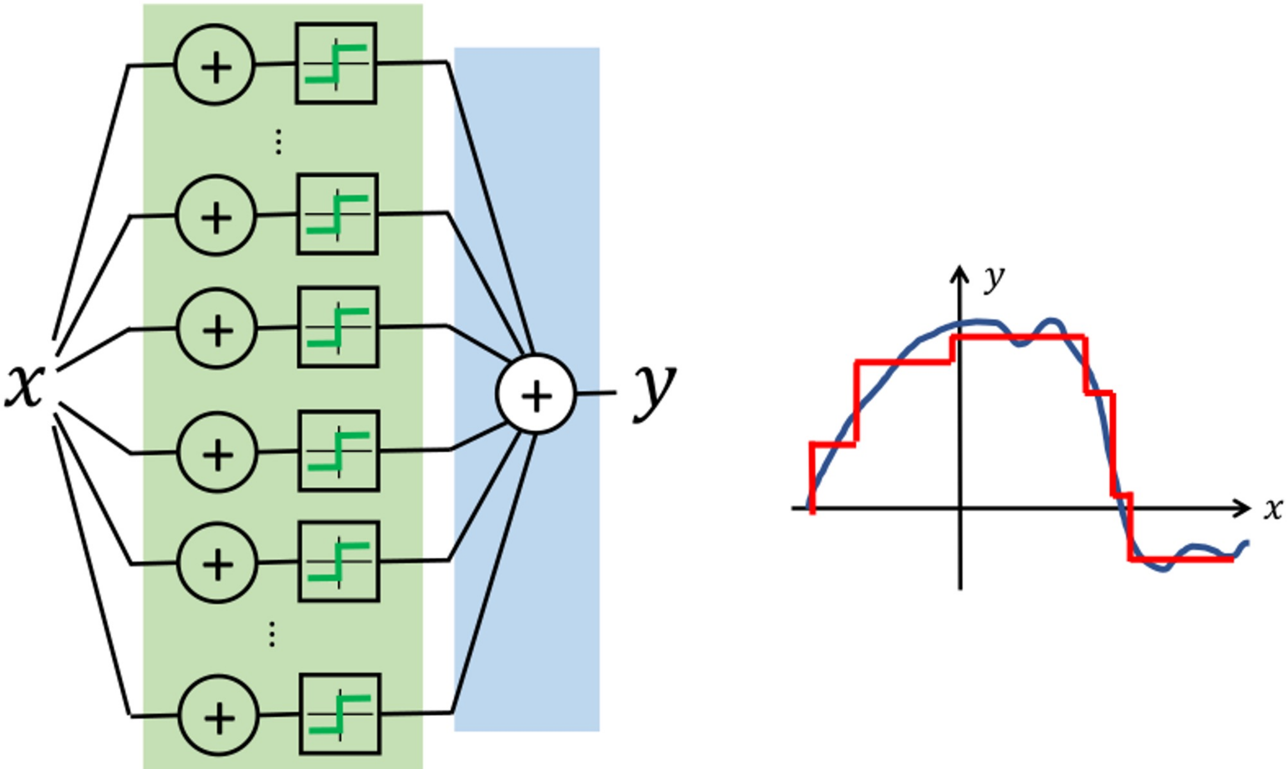


y
x


x
+
y


y
x

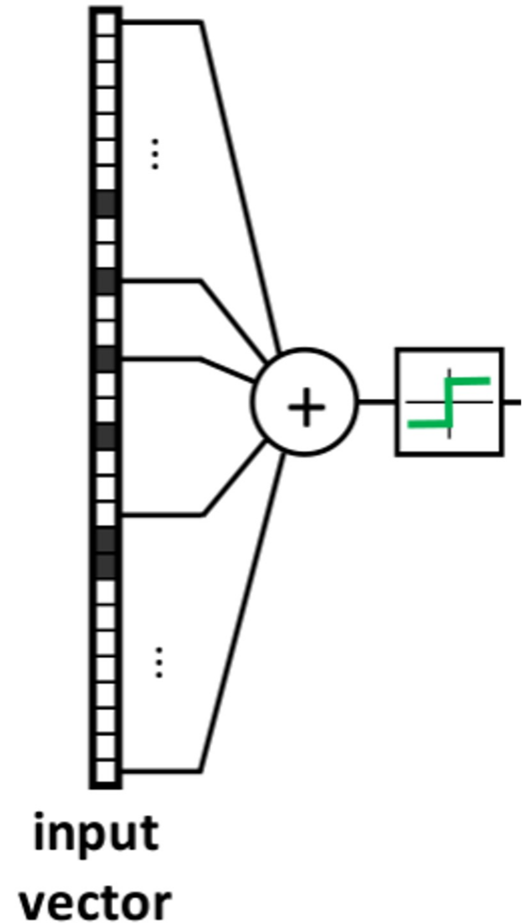
Universal Approximation



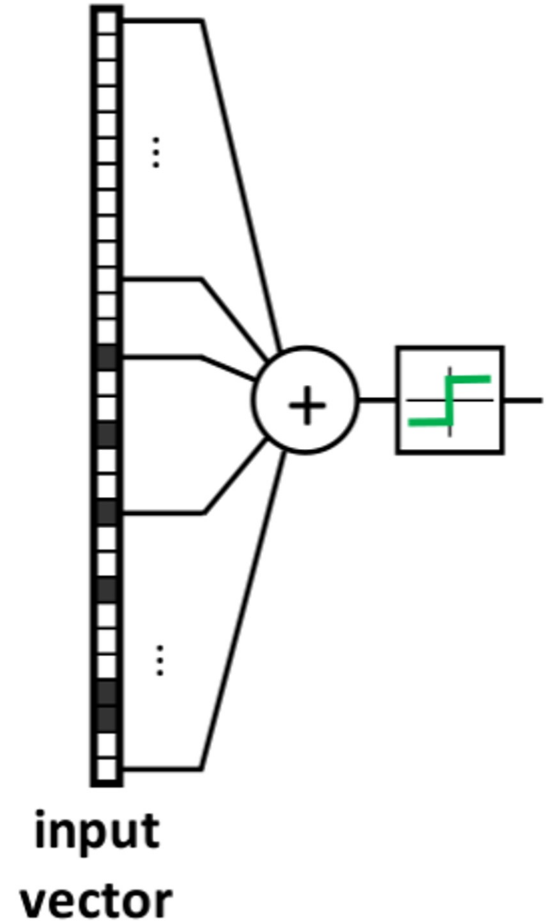
can approximate a continuous function to any desired accuracy

Cybenko 1989; Hornik 1991

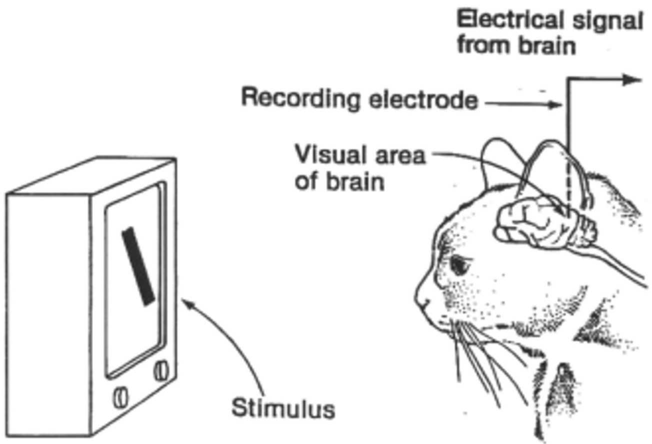
Curse of Dimensionality



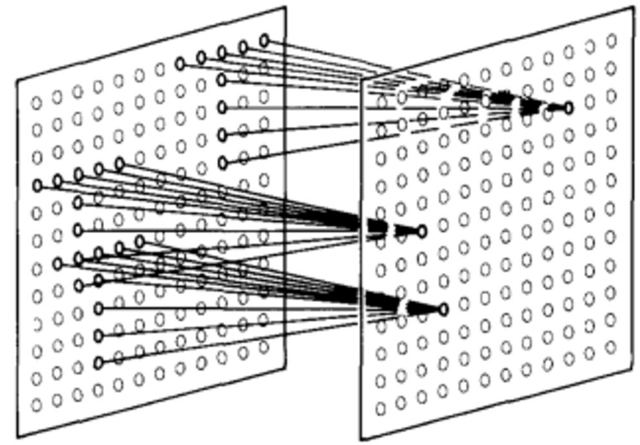
Curse of Dimensionality



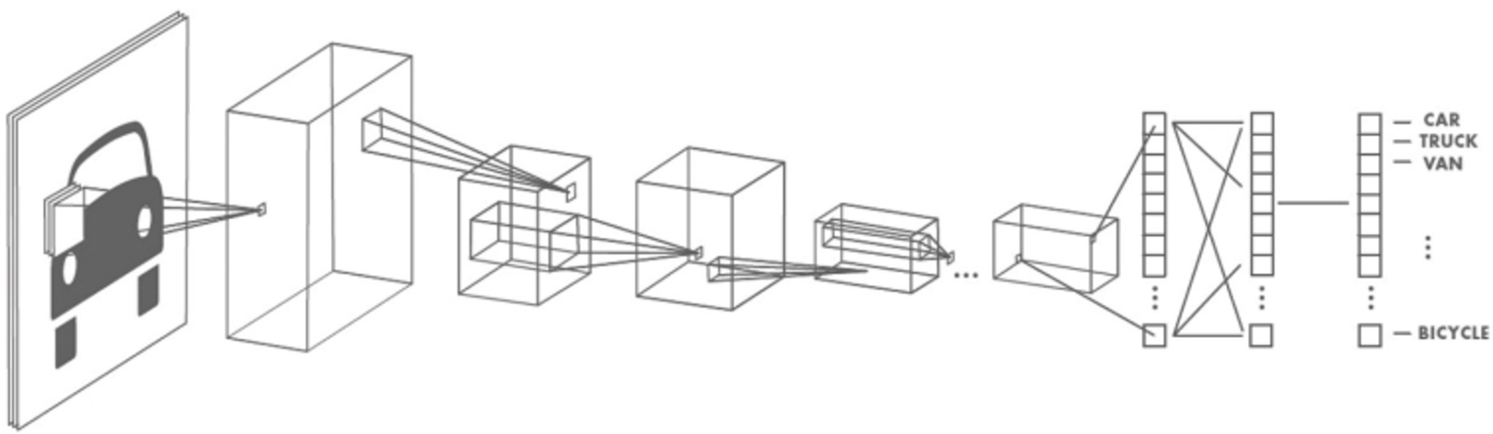
must learn shift invariance from data!



Hubel, Wiesel 1962; 1981



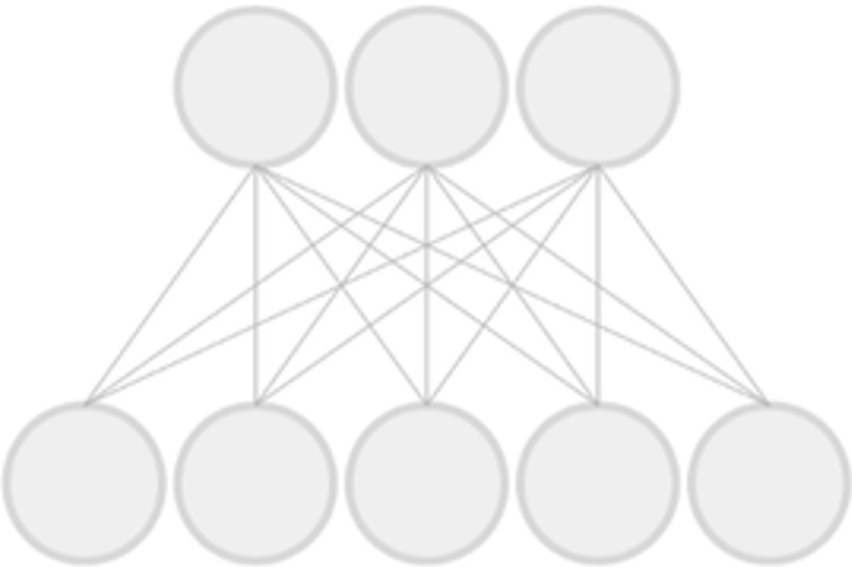
Fukushima 1980



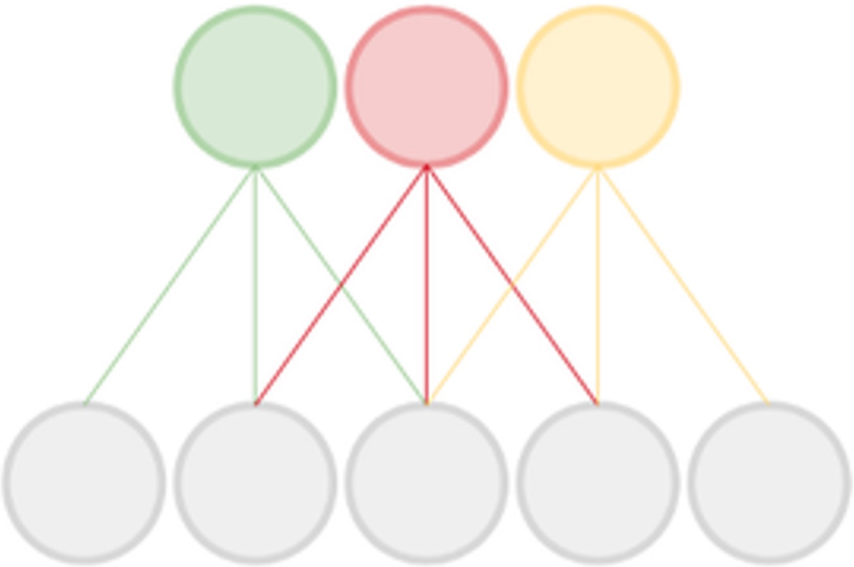
LeCun et al. 1989

Convolution

1. Local filters (local receptive field connectivity)



Fully connected layer

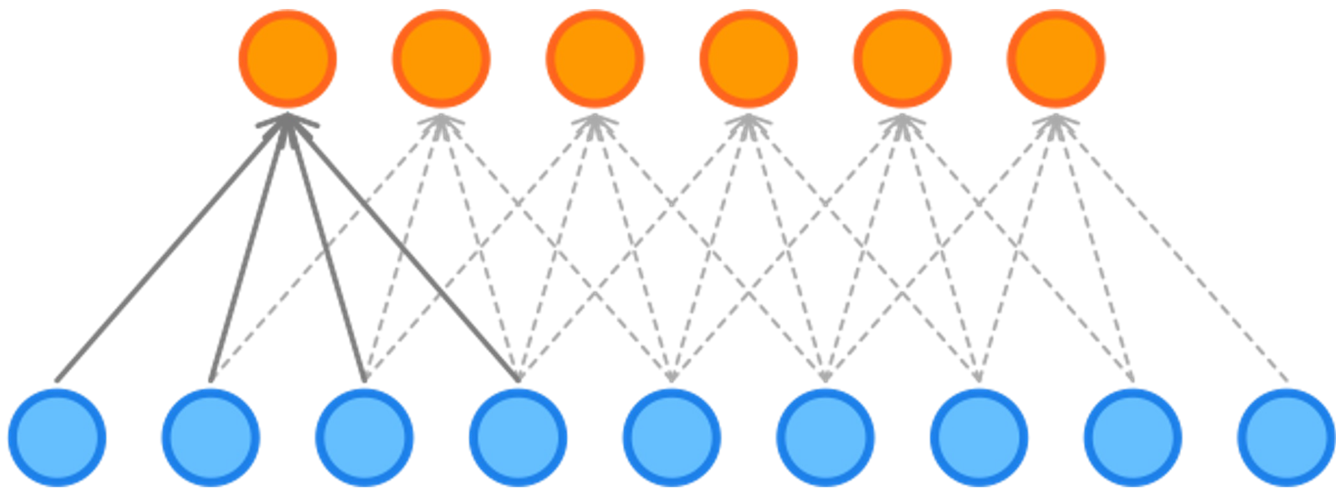


Convolutional layer

Convolution

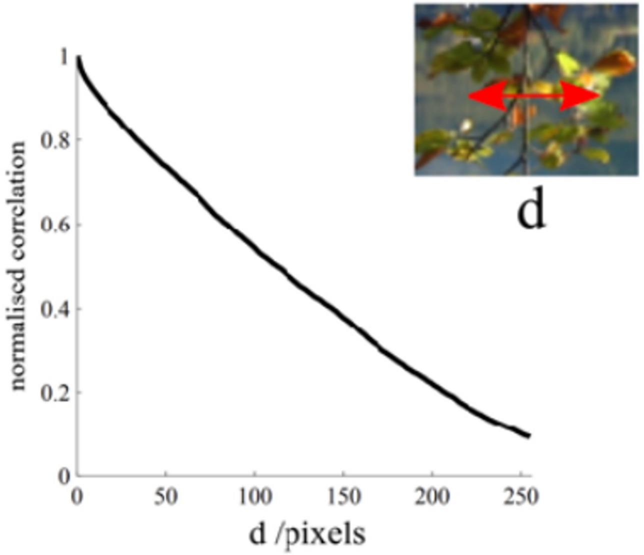
1. Local filters (local receptive field connectivity)

```
kernel_size = 4
strides = 1
padding = 'valid'
```



Convolution

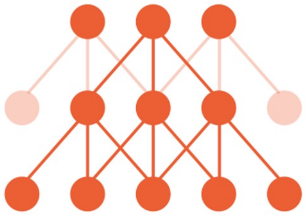
1. Local filters (local receptive field connectivity)



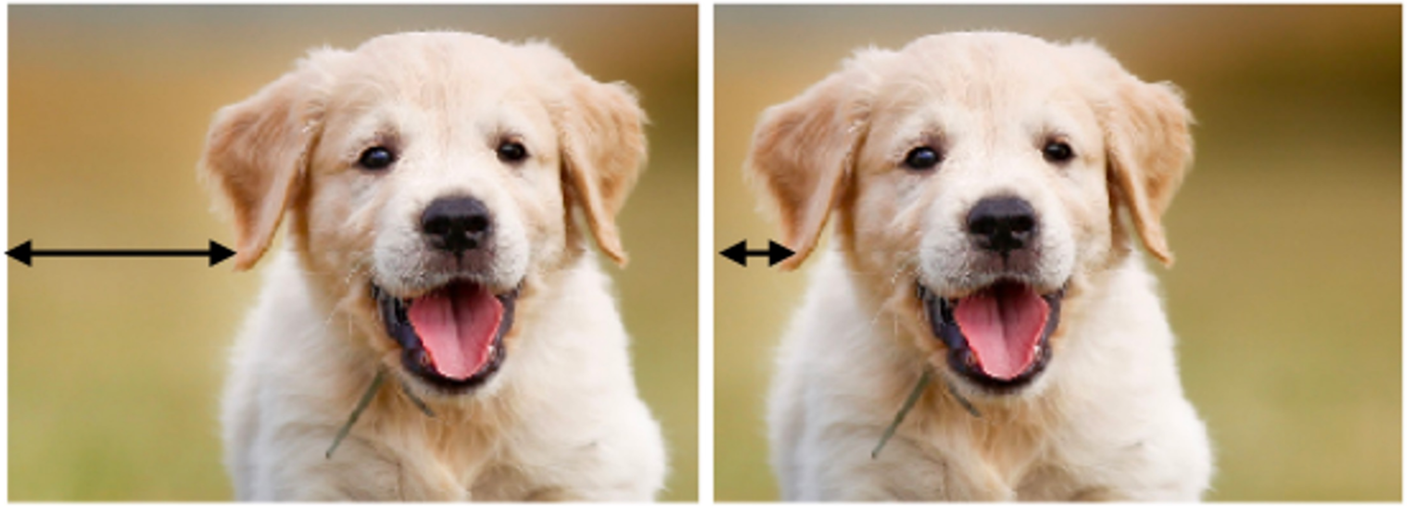
Locality of pixel statistics (Property of Image data)

http://www.gatsby.ucl.ac.uk/~turner/teaching/4g3/2013/stats-recep-field.pdf

Convolution

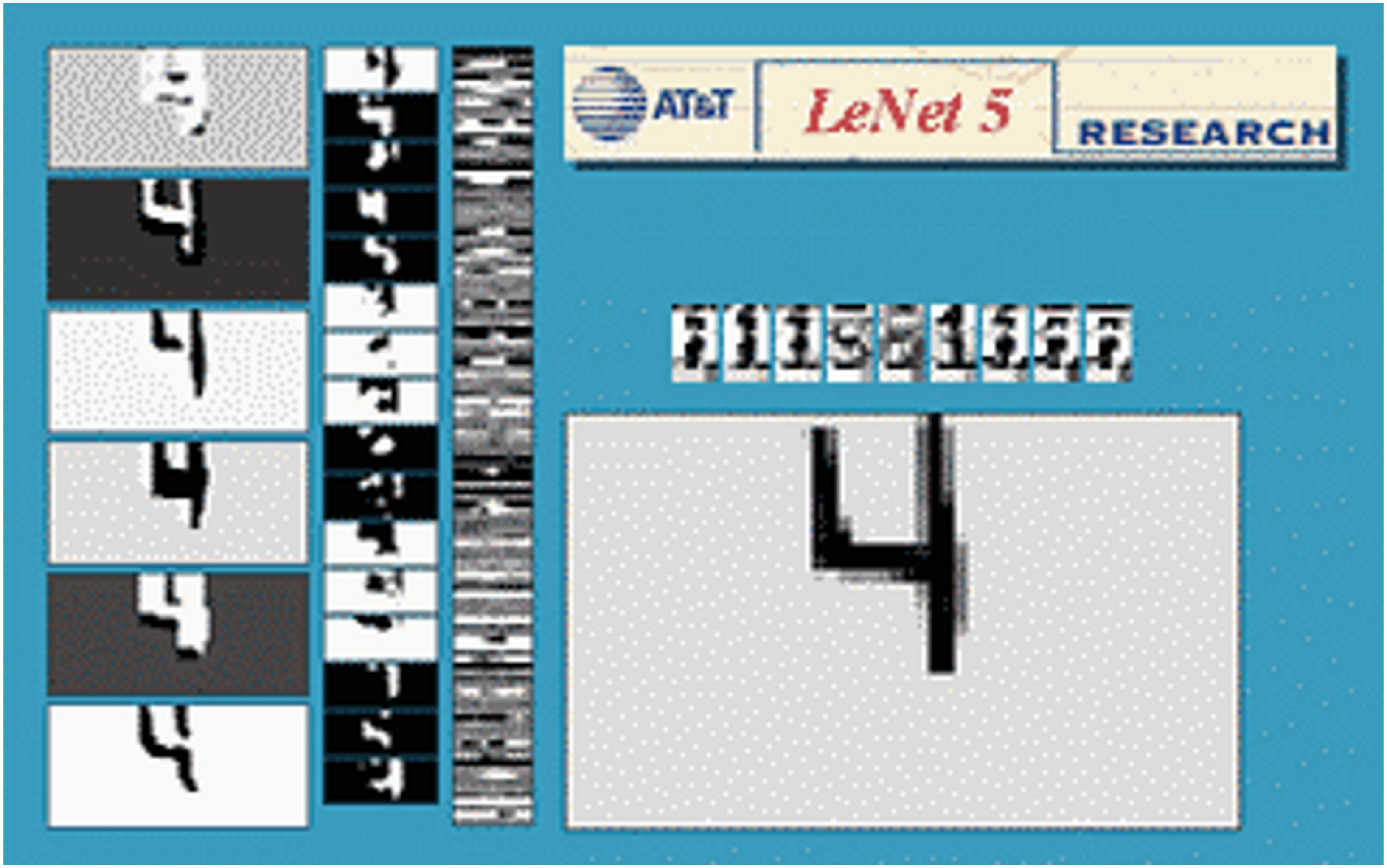


1. Local filters (local receptive field connectivity)
2. Translation weight tying (Weight sharing exploits translation symmetry)

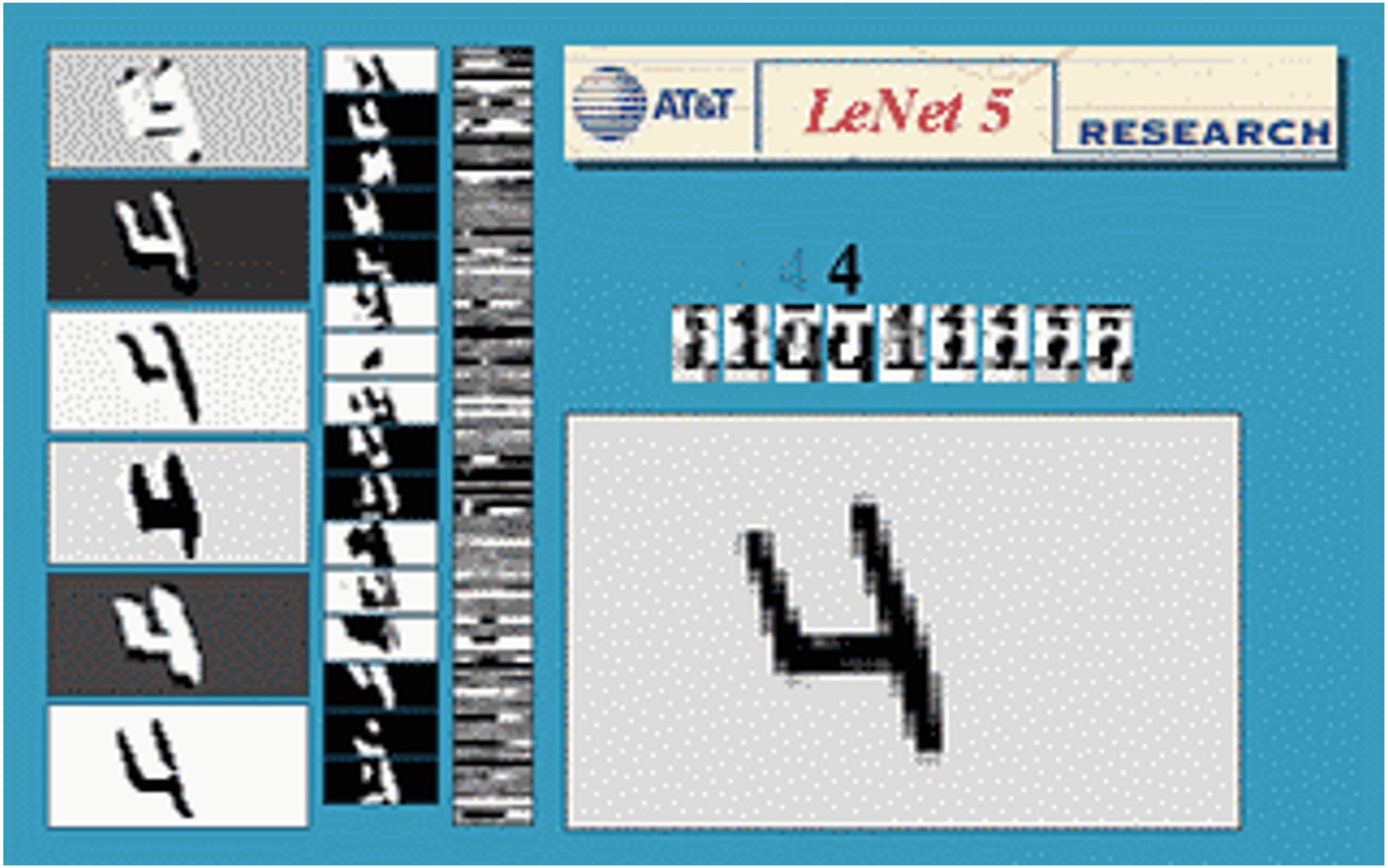


Translational invariance of classification (Property of task)

https://www.healthypawspetinsurance.com/Images/



http://yann.lecun.com/exdb/lenet/translation.html



http://yann.lecun.com/exdb/lenet/translation.html

$\mathfrak{G}$-invariance $\quad f(\rho(\mathfrak{g})x) = f(x)$

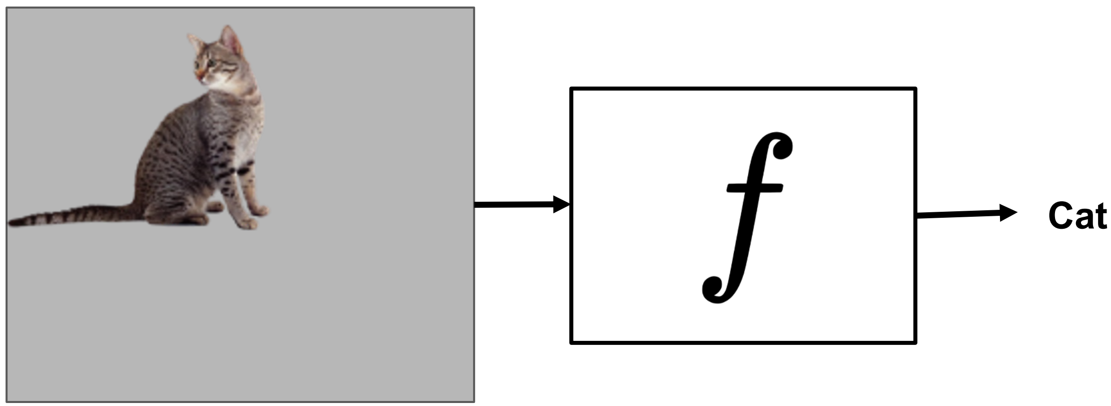


Image Classification

$\mathfrak{G}$-invariance $f(\rho(\mathfrak{g})x) = f(x)$

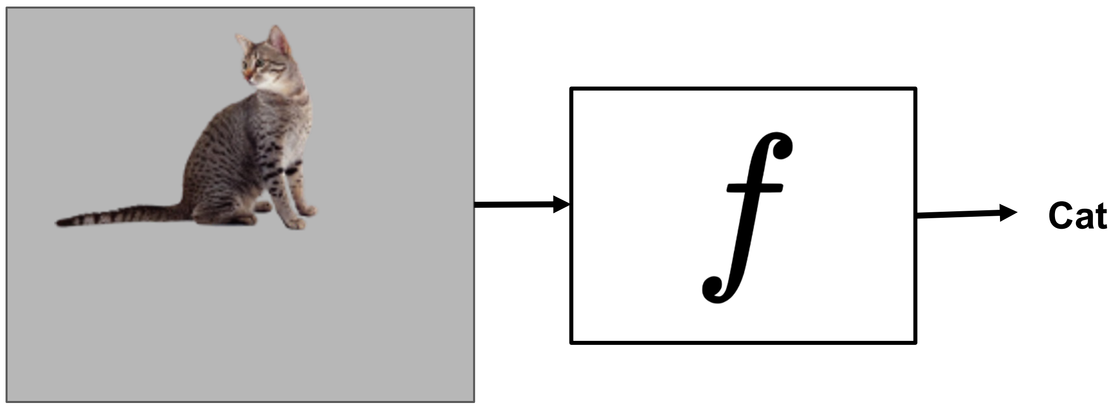


Image Classification

$\mathfrak{G}$-invariance $\quad f(\rho(\mathfrak{g})x) = f(x)$

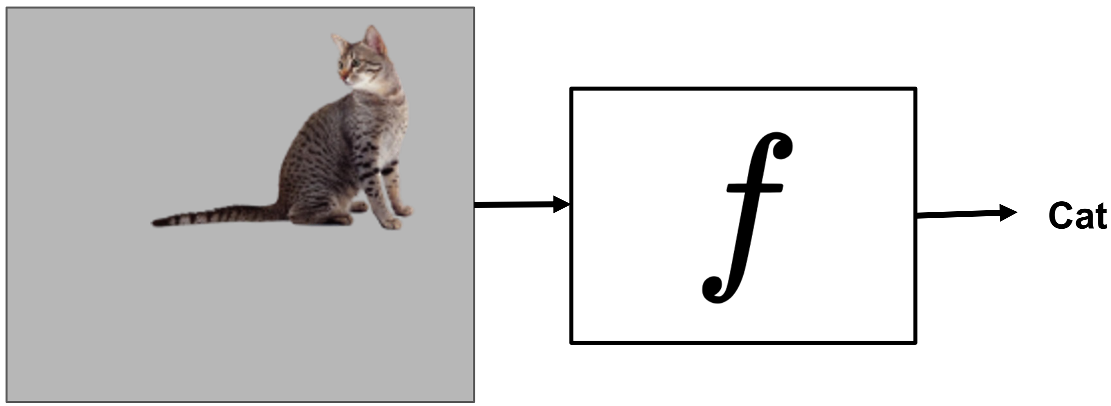


Image Classification

$\mathfrak{G}$-invariance $\quad f(\rho(\mathrm{g})x) = f(x)$

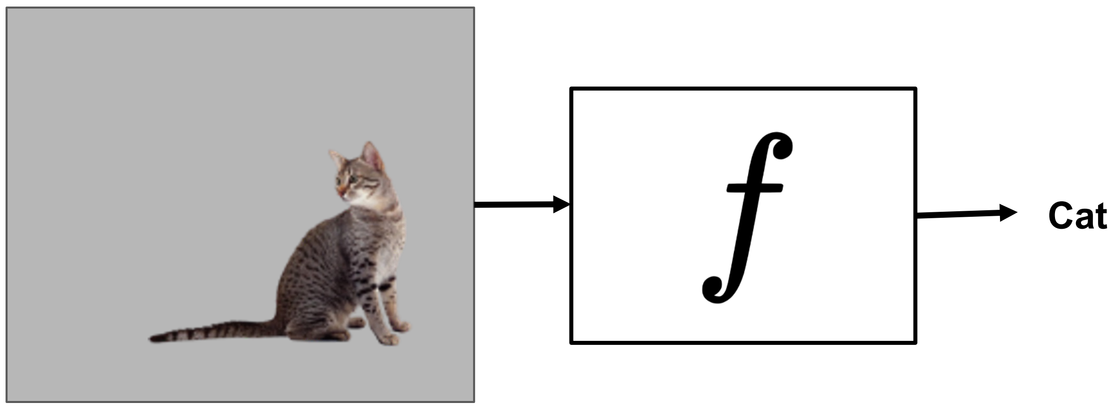


Image Classification

$\mathfrak{G}$-invariance $\quad f(\rho(\mathfrak{g})x) = f(x)$

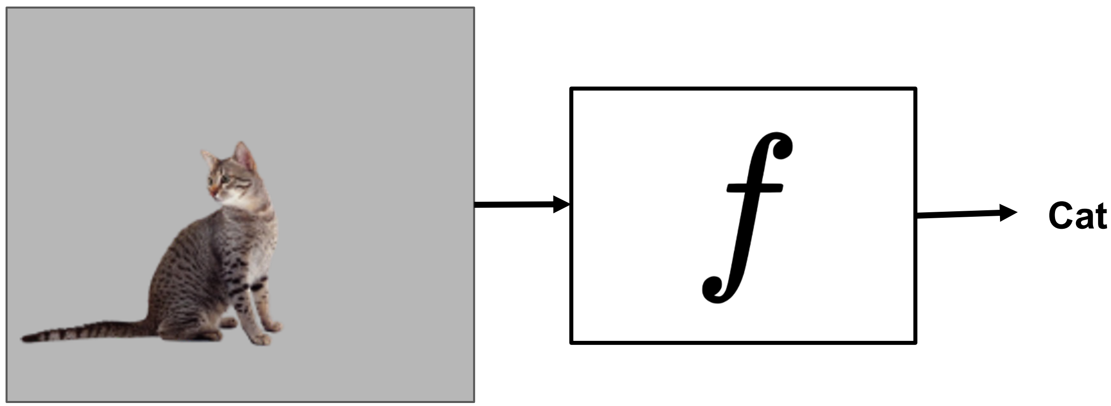


Image Classification

$\mathfrak{G}$-equivariance $f(\rho(\mathrm{g})x) = \rho(\mathrm{g})f(x)$

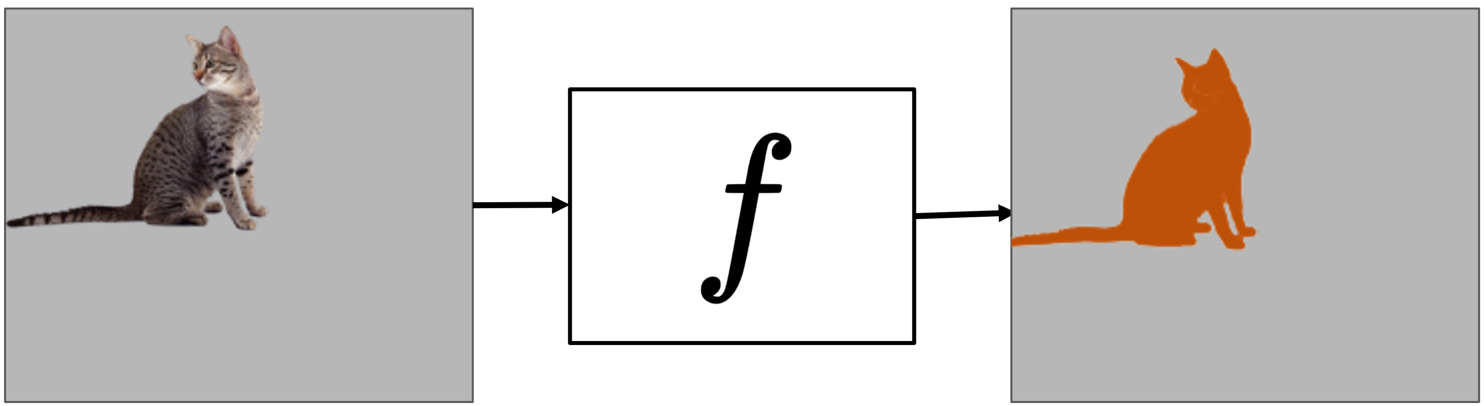


Image Segmentation

$\mathfrak{G}$-equivariance $f(\rho(g)x) = \rho(g)f(x)$

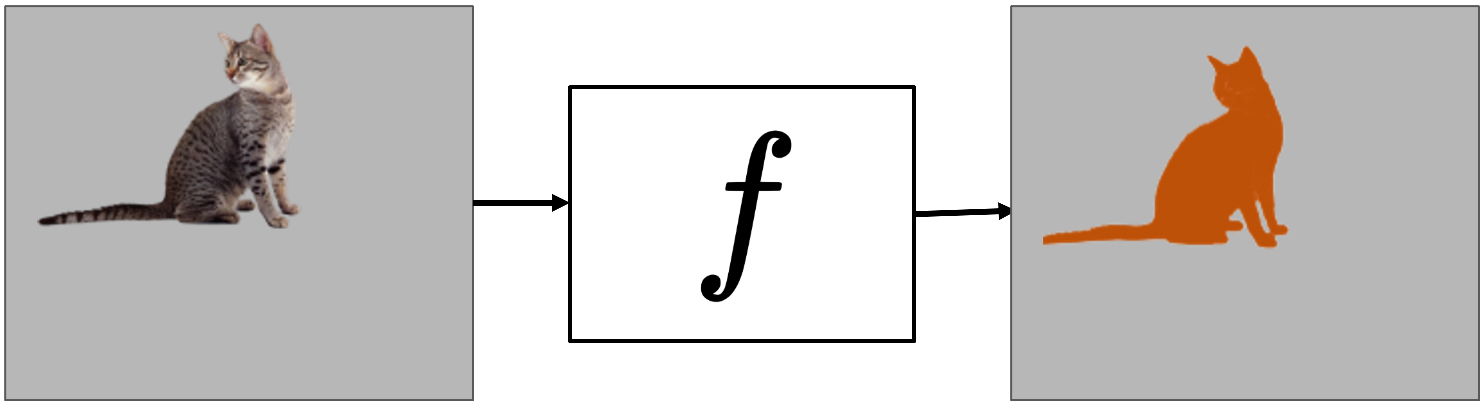


Image Segmentation

$\mathfrak{G}$-equivariance $f(\rho(\mathrm{g})x) = \rho(\mathrm{g})f(x)$

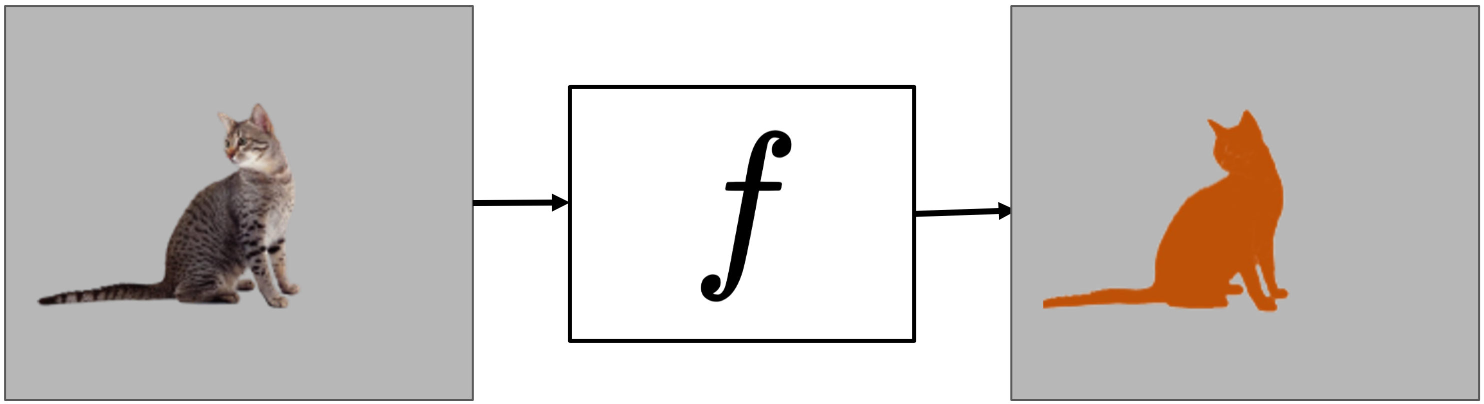


Image Segmentation

$\mathfrak{G}$-equivariance $f(\rho(\mathfrak{g})x) = \rho(\mathfrak{g})f(x)$

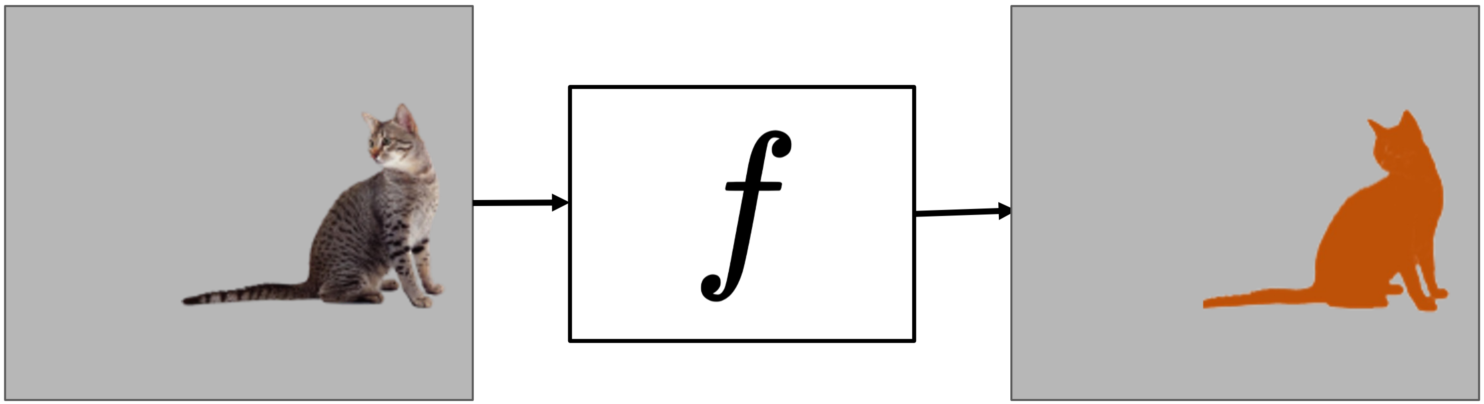


Image Segmentation

Symmetry: Invariance

Set of input transformations leaving f invariant

image

$$f(\mathbf{I}) = f(\mathcal{T}_\theta[\mathbf{I}])$$

function/
feature mapping

transformation

Notational aside:

e.g. Geometric translation

$$\mathcal{T}_\theta[\mathbf{I}](\mathbf{x}) = \mathbf{I}(\mathbf{x} - \boldsymbol{\theta})$$

e.g. Geometric rotation

$$\mathcal{T}_\theta[\mathbf{I}](\mathbf{x}) = \mathbf{I}(\mathbf{R}_{\boldsymbol{\theta}}^{-1}\mathbf{x})$$

e.g. Pixel normalisation

$$\mathcal{T}[\mathbf{I}] = (\mathbf{I} - \boldsymbol{\mu})/.\boldsymbol{\sigma}^{-1}$$

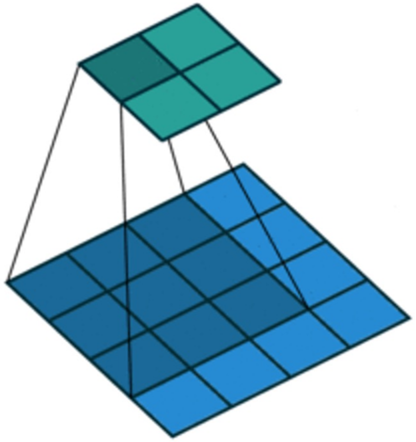
Symmetry: Equivariance

Different representations of same transformation

$$\mathcal{S}_\theta[f](\mathbf{I}) = f(\mathcal{T}_\theta[\mathbf{I}])$$

transformation in feature space

Mapping (equivariance transformations) preserves algebraic structure of transformation



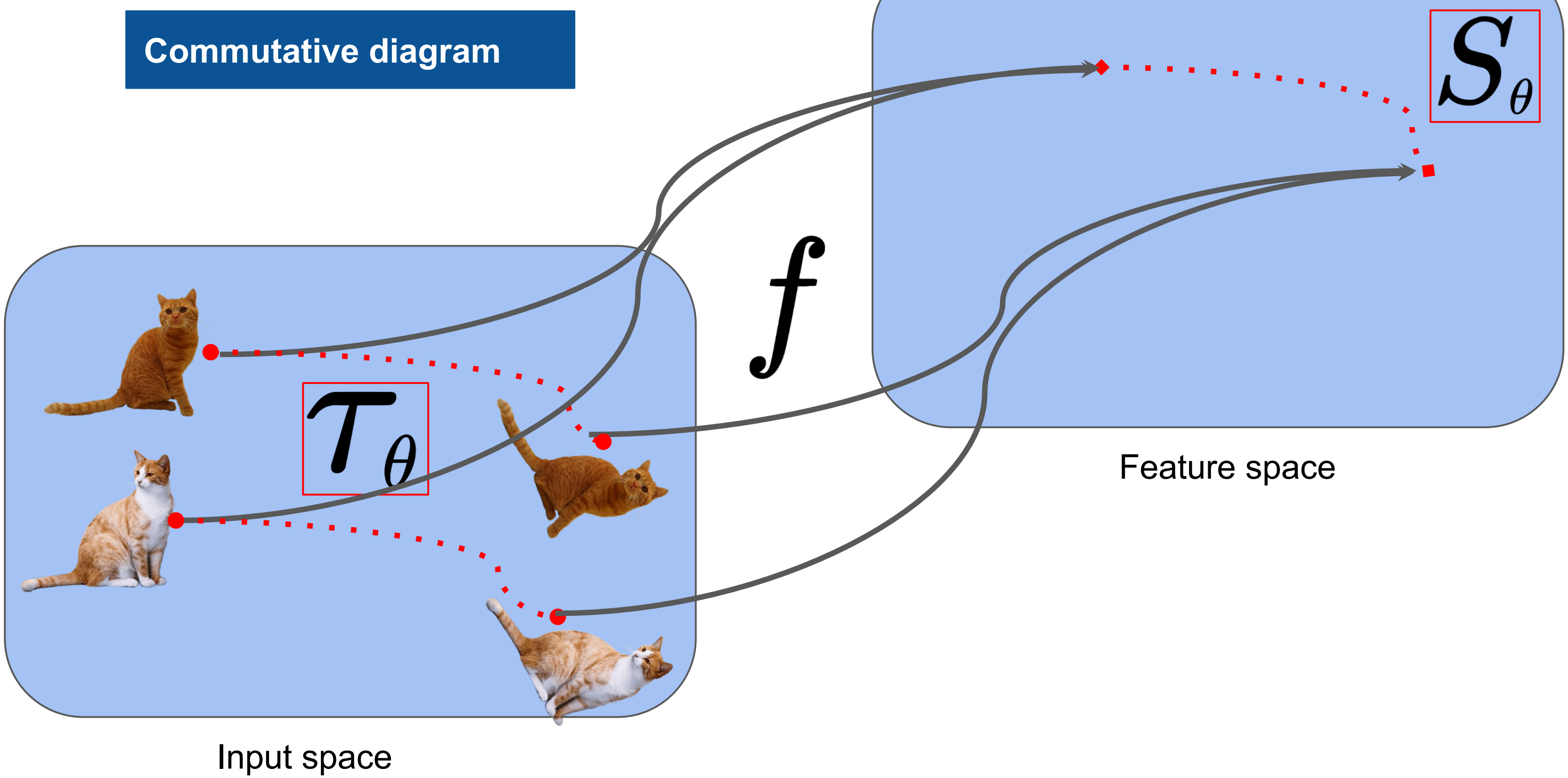
Invariance

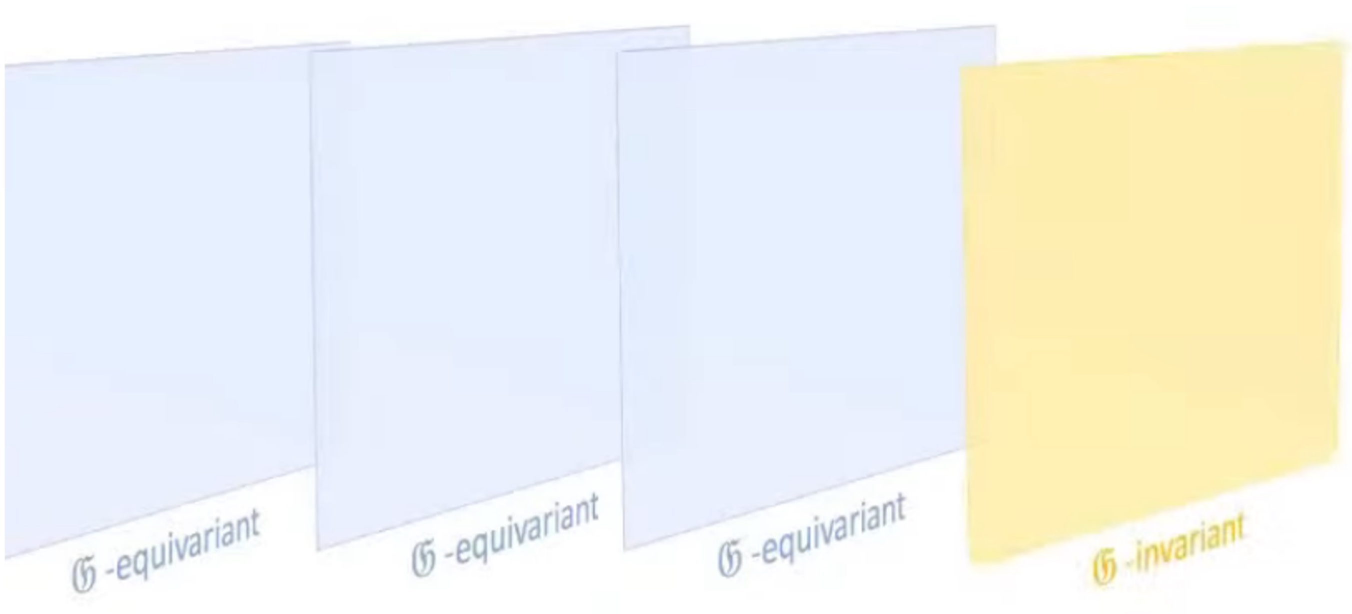
$$\mathcal{S}_\theta = \mathrm{Id}$$

Convolution (and correlation)

$$[\mathbf{I} * \mathbf{W}](\mathbf{x} - \theta) = \mathcal{T}_\theta[\mathbf{I}] * \mathbf{W}(\mathbf{x})$$

Equivariance





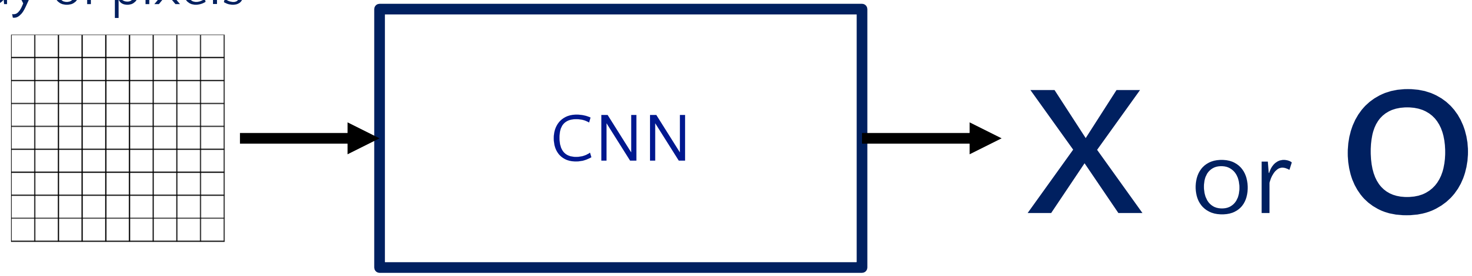
These two principles (Invariance and Equivariance) give us a very general blueprint of Neural Network Architectures.

A toy ConvNet: X's and O's

A toy ConvNet: X's and O's

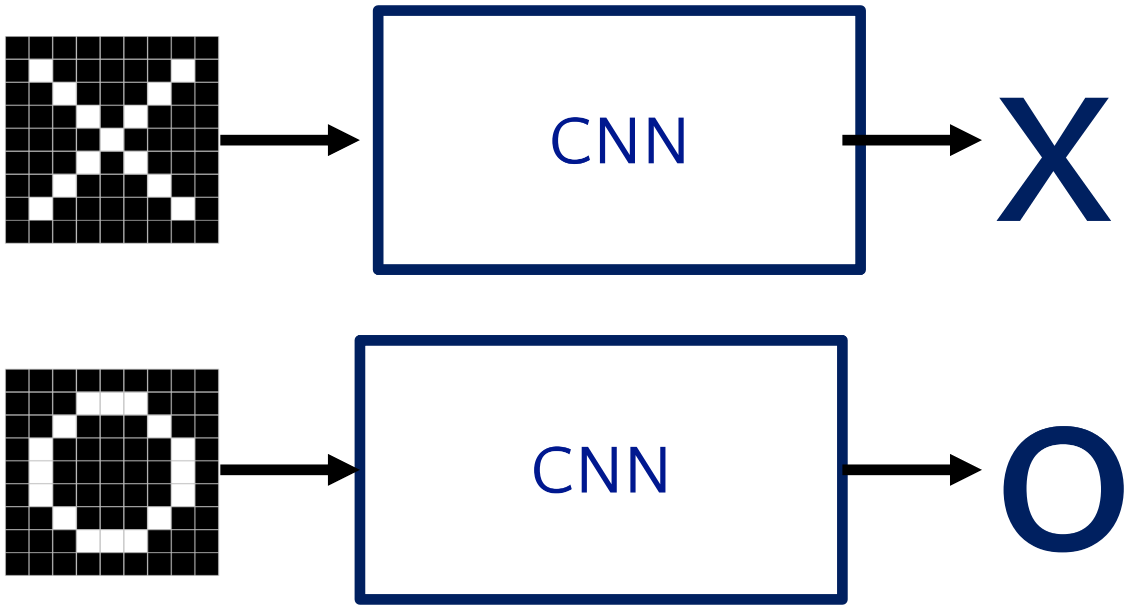
Says whether a picture is of an X or an O

A two-dimensional array of pixels



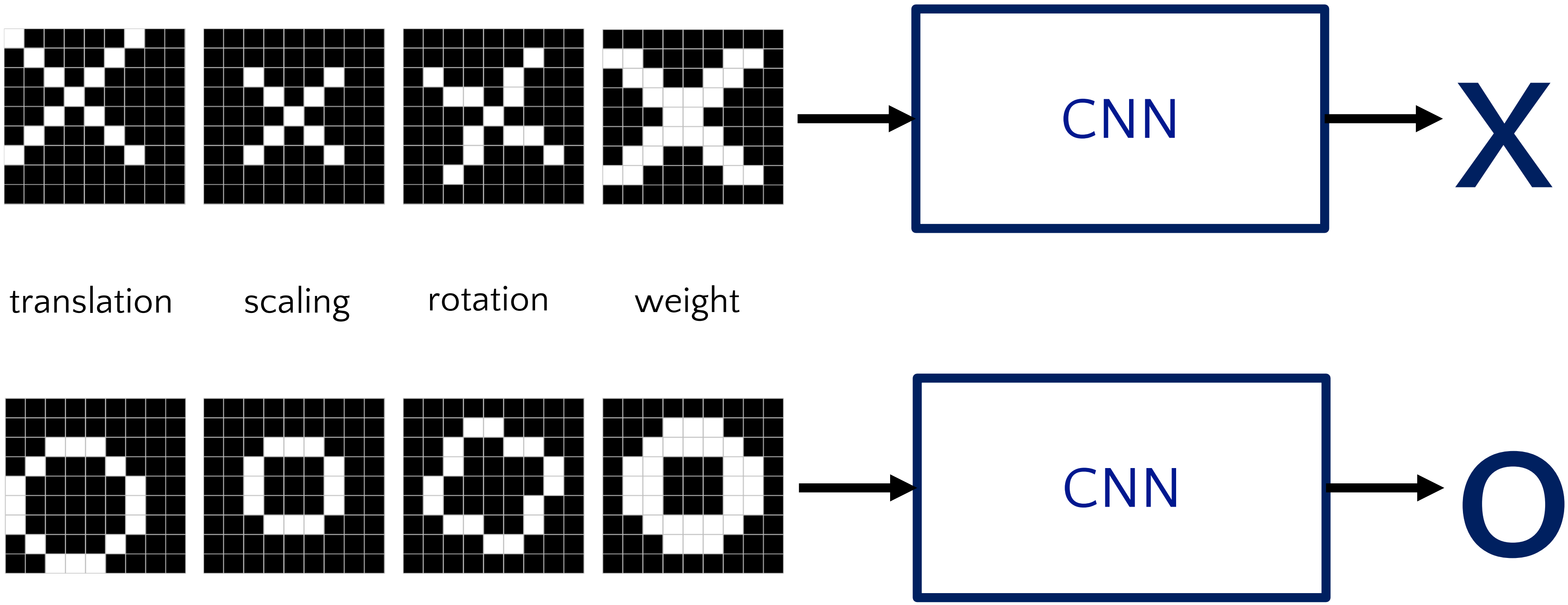
Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

For example



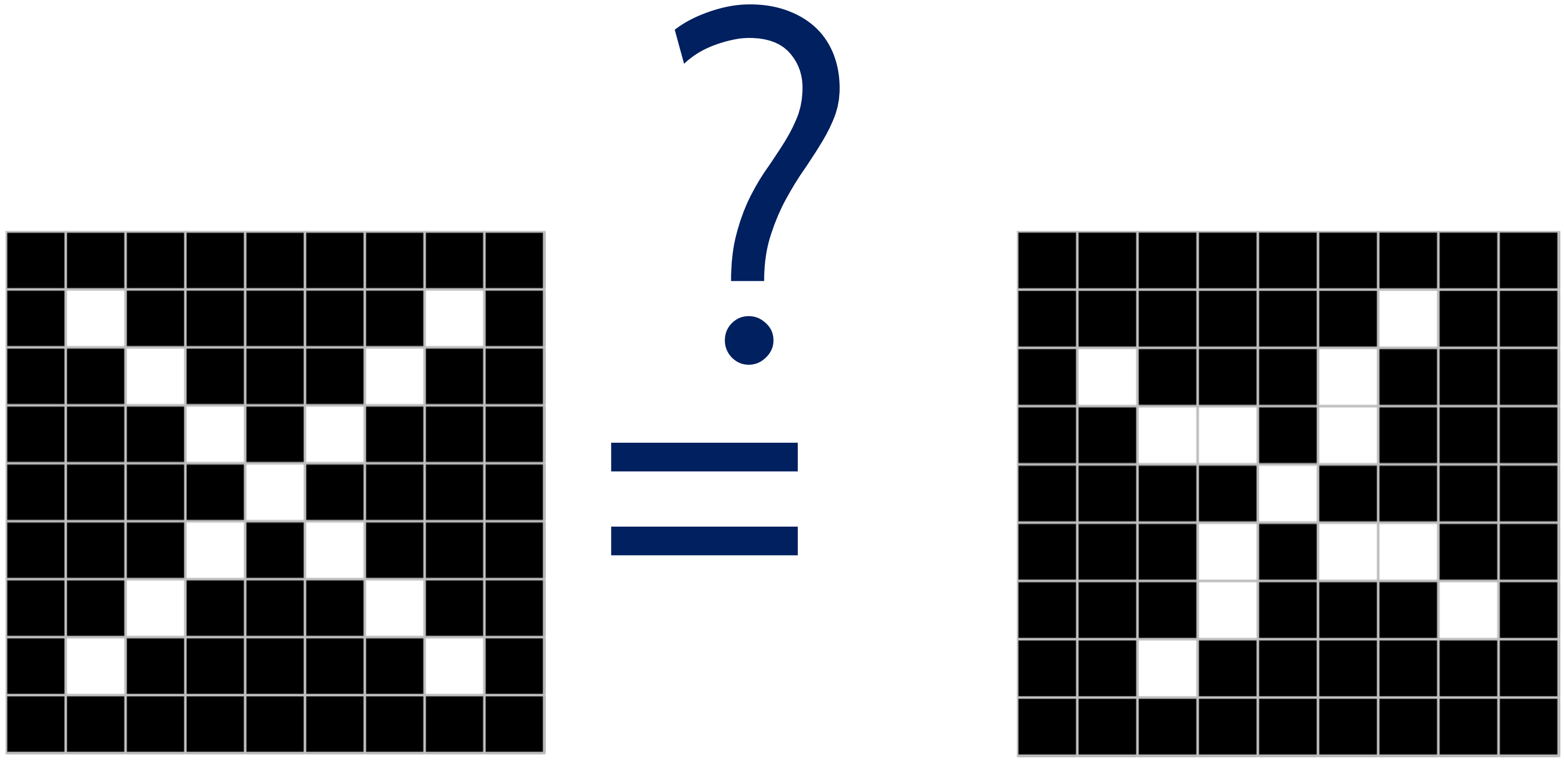
Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Trickier cases



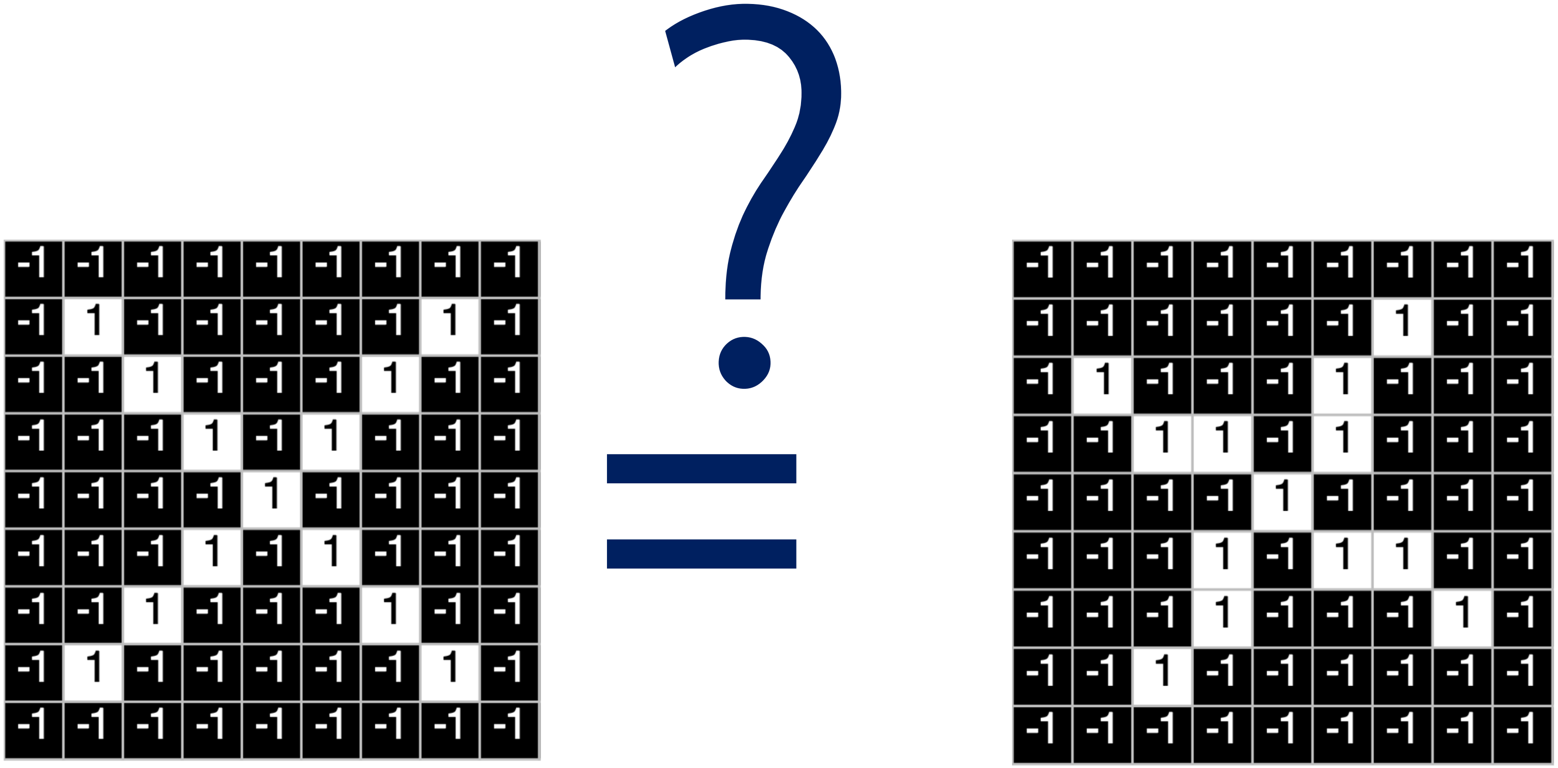
Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Deciding is hard



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

What computers see



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

What computers see

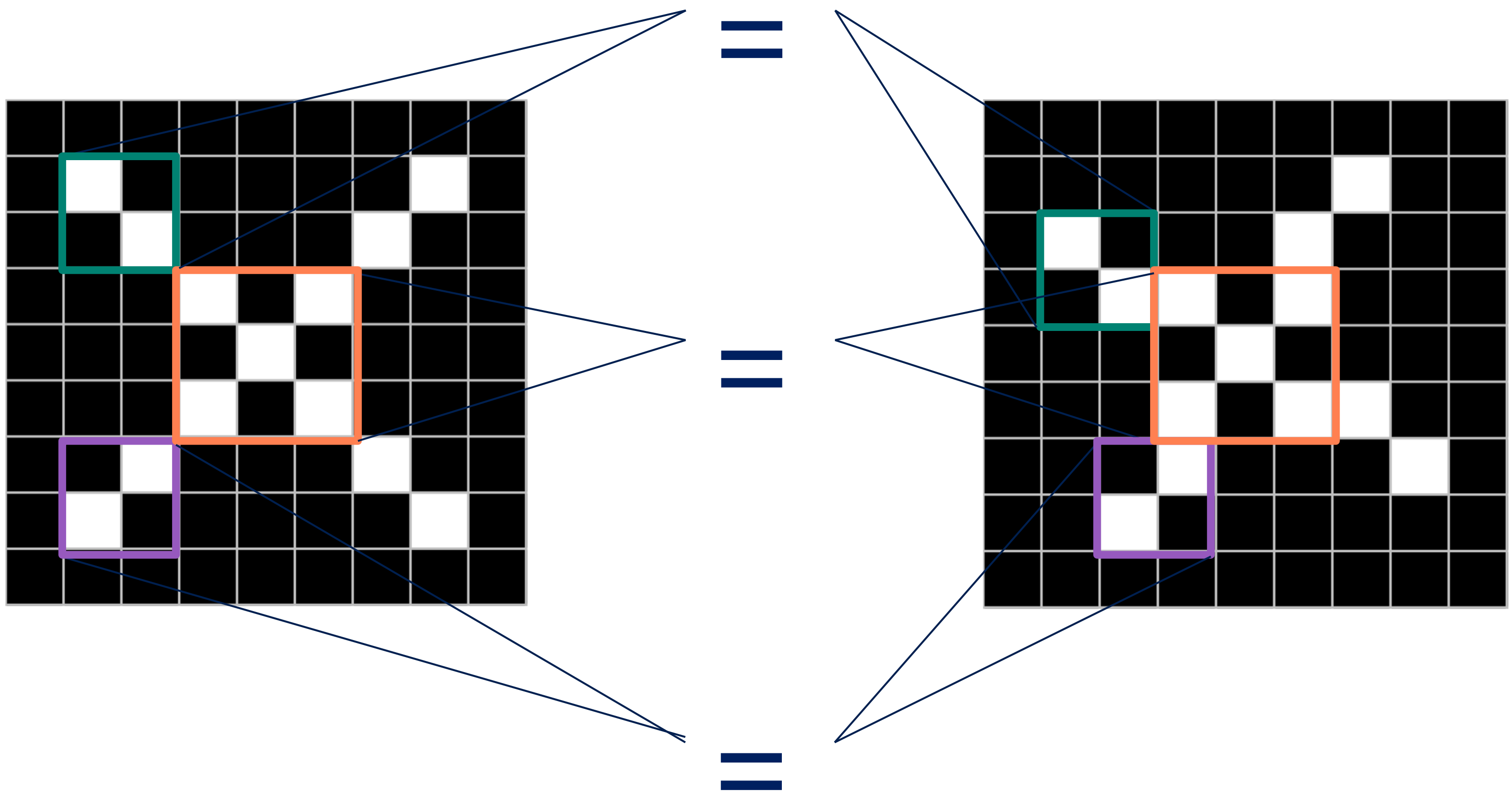
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	X	-1	-1	-1	-1	X	X	-1
-1	X	X	-1	-1	X	X	-1	-1
-1	-1	X	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	X	-1	-1
-1	-1	X	X	-1	-1	X	X	-1
-1	X	X	-1	-1	-1	-1	X	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Computers are literal

$$
\begin{bmatrix}
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\
-1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\
-1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\
-1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1
\end{bmatrix}
\neq
\begin{bmatrix}
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 & -1 \\
-1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 \\
-1 & -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 \\
-1 & -1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1
\end{bmatrix}
$$

ConvNets match pieces of the image



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

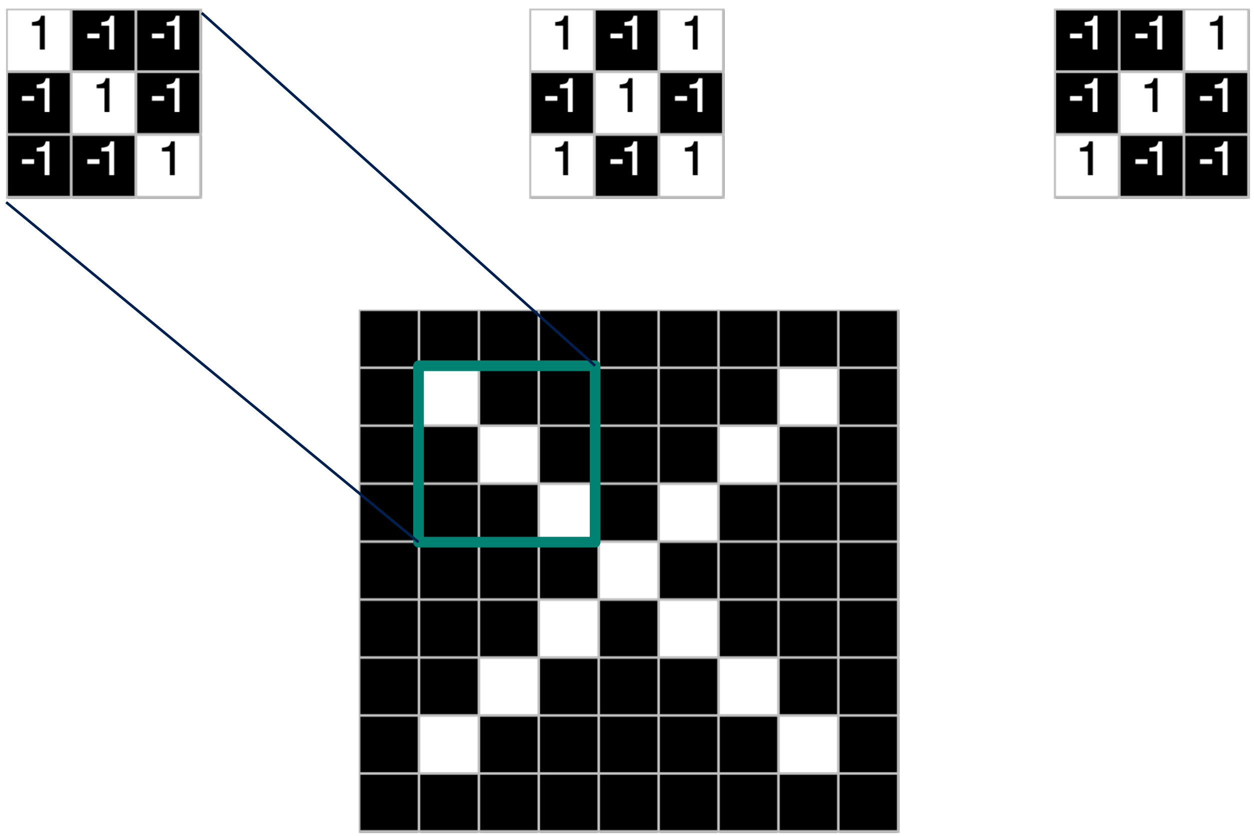
Features match pieces of the image

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

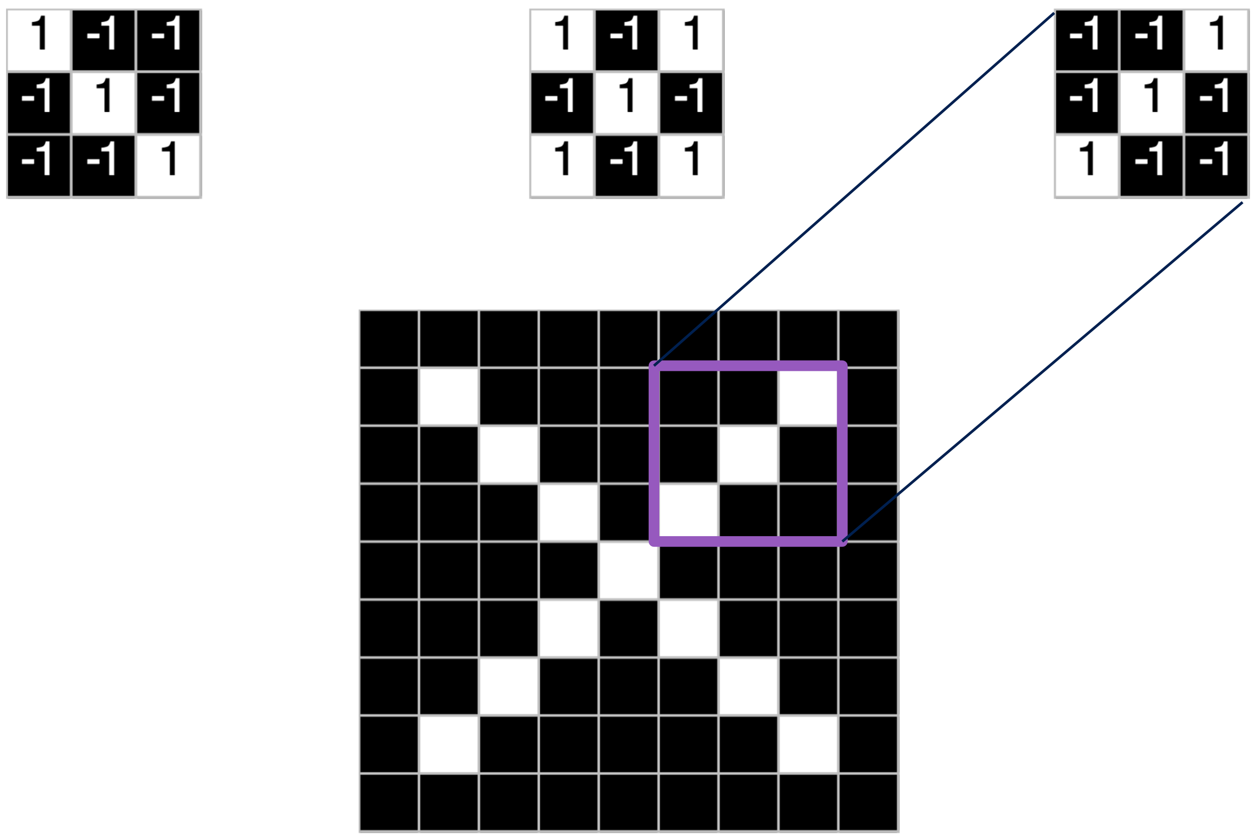
1	-1	1
-1	1	-1
1	-1	1

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

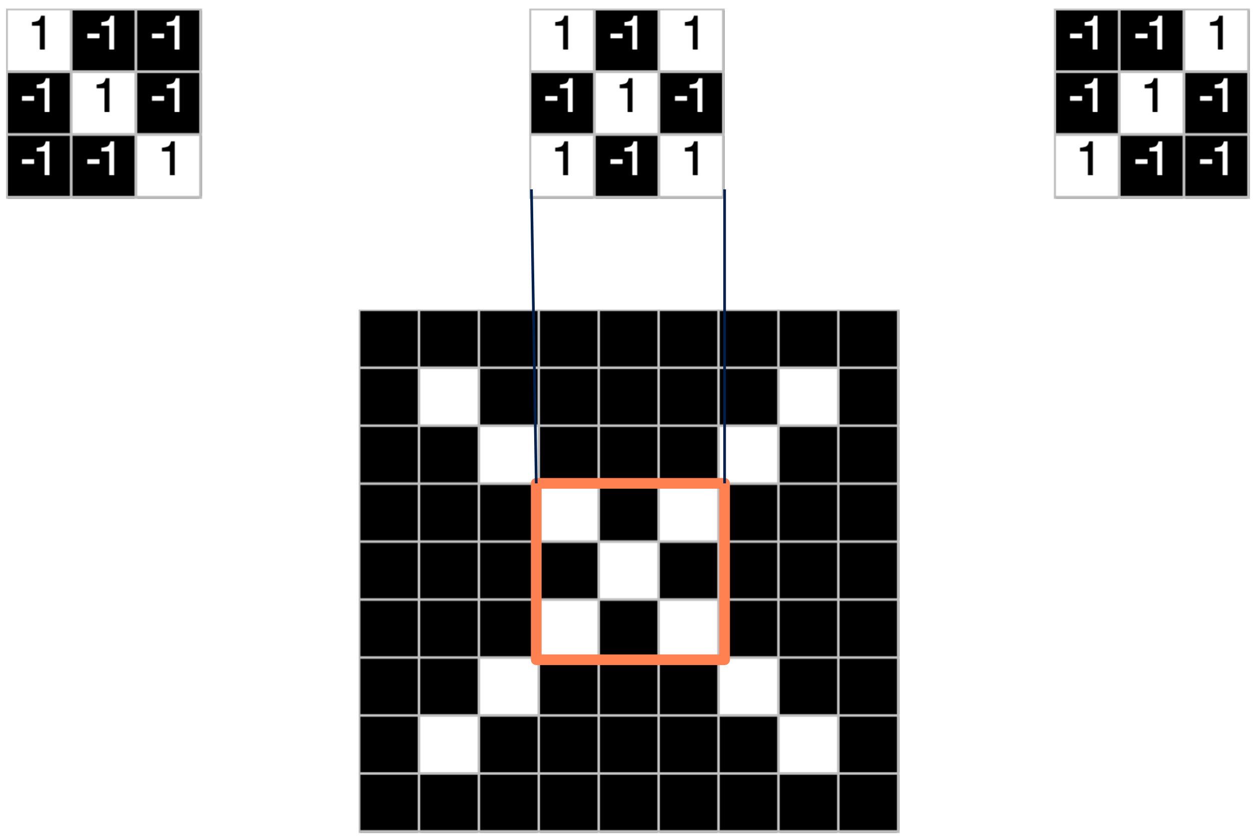
Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html



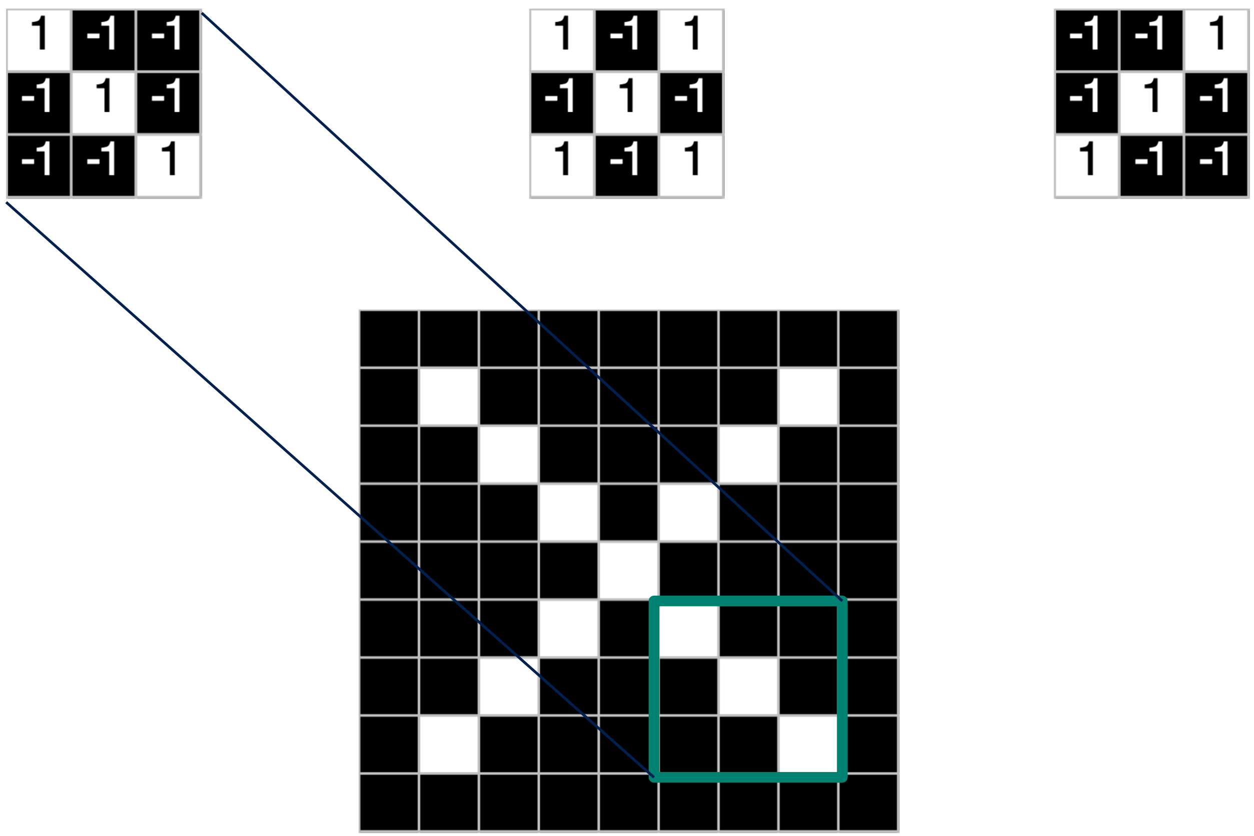
Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html



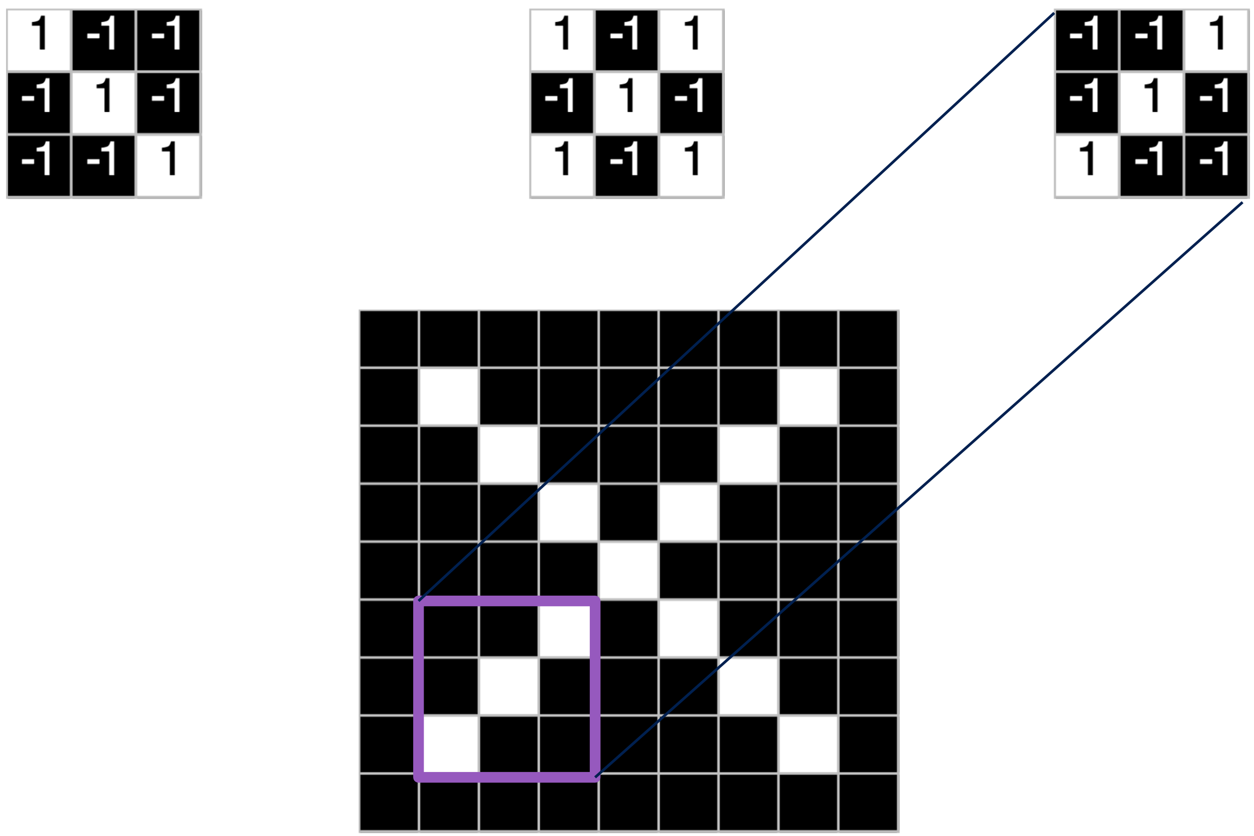
Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Filtering: The math behind the match

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

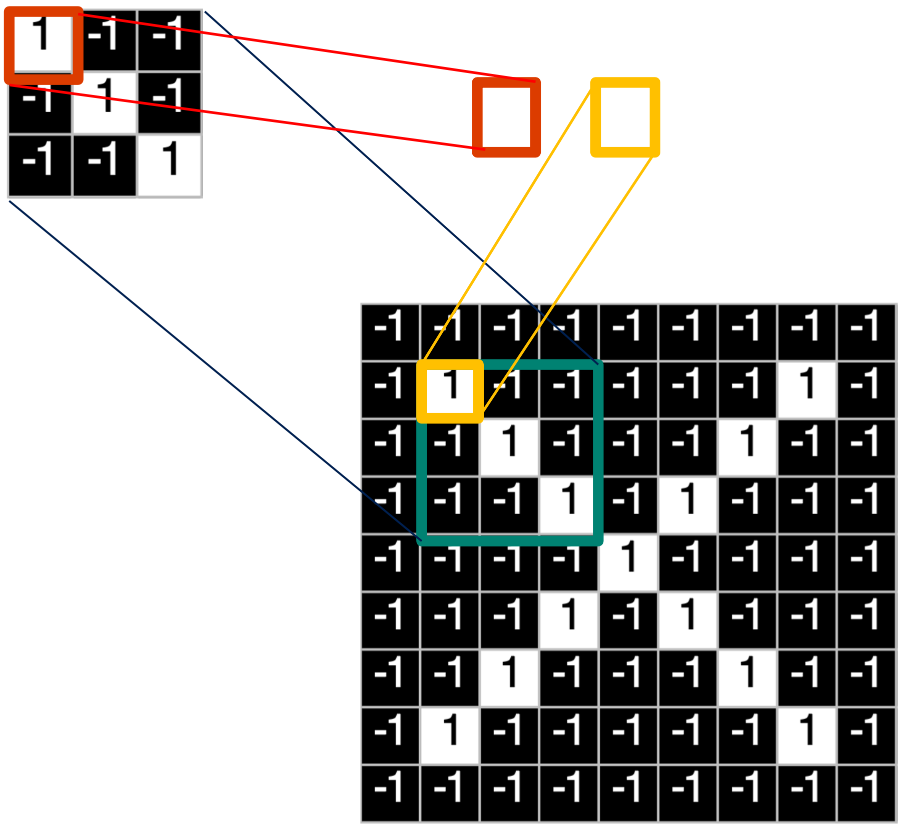
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Filtering: The math behind the match

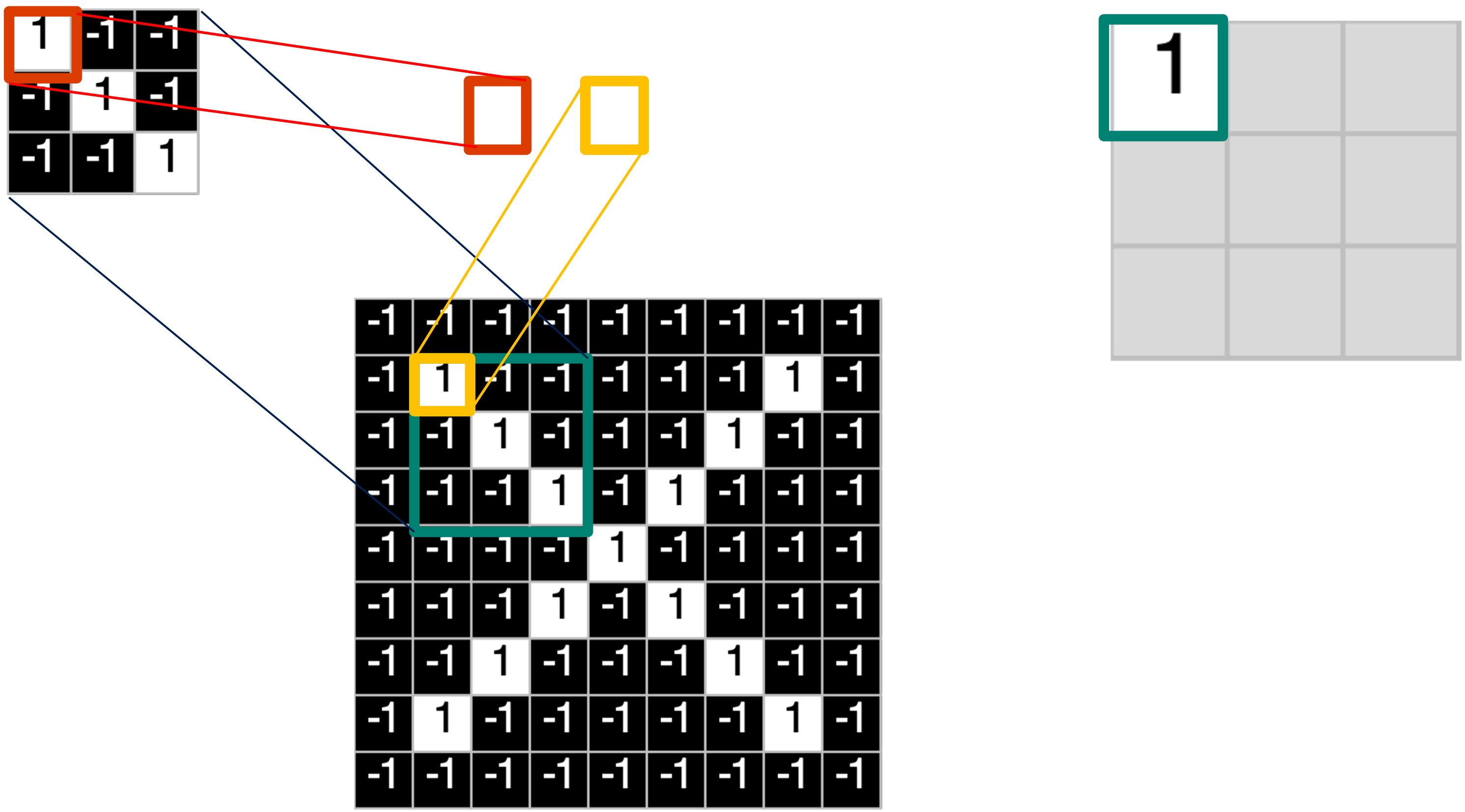
1. Line up the feature and the image patch.
2. Multiply each image pixel by the corresponding feature pixel.
3. Add them up.
4. Divide by the total number of pixels in the feature.

Filtering: The math behind the match



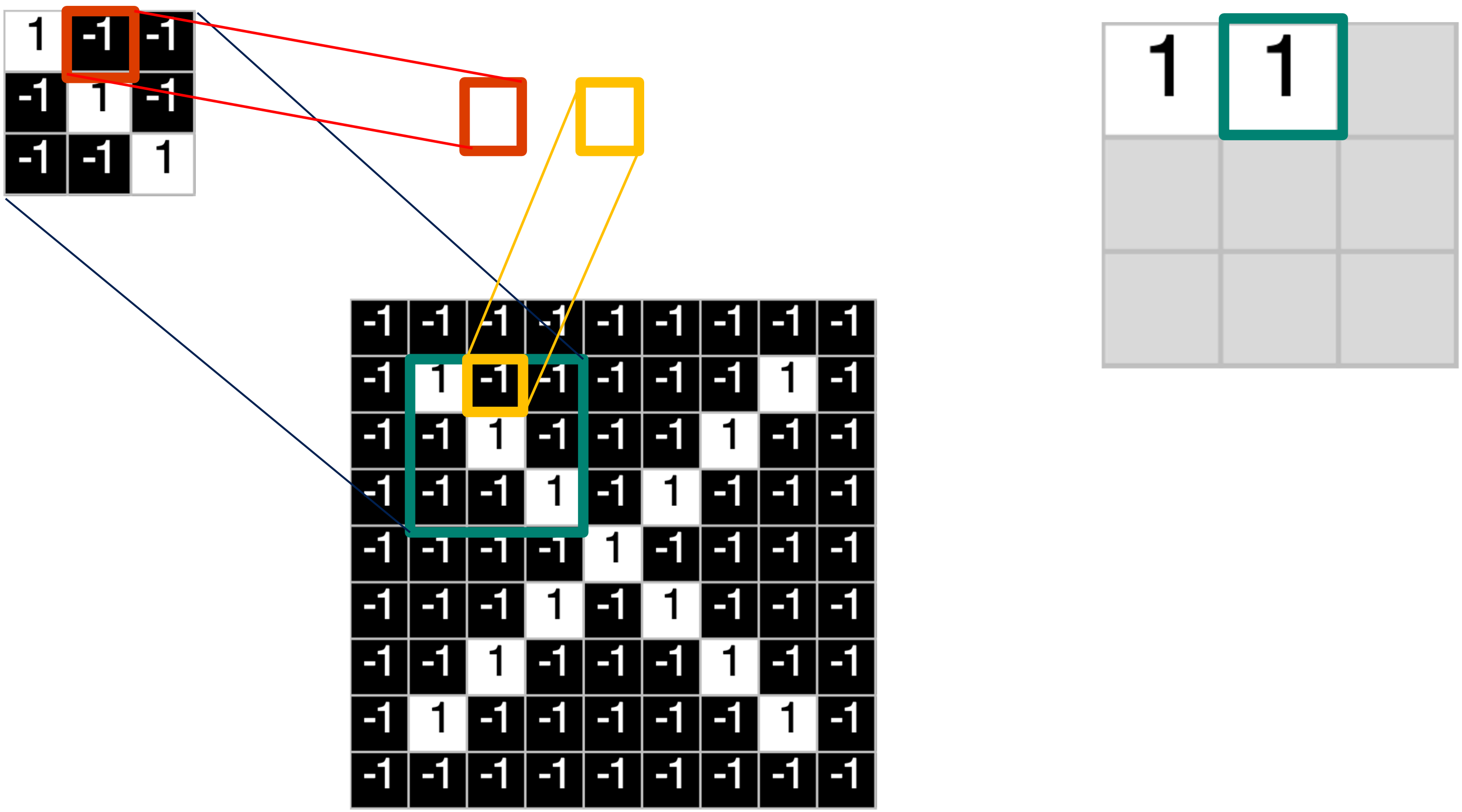
Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Filtering: The math behind the match



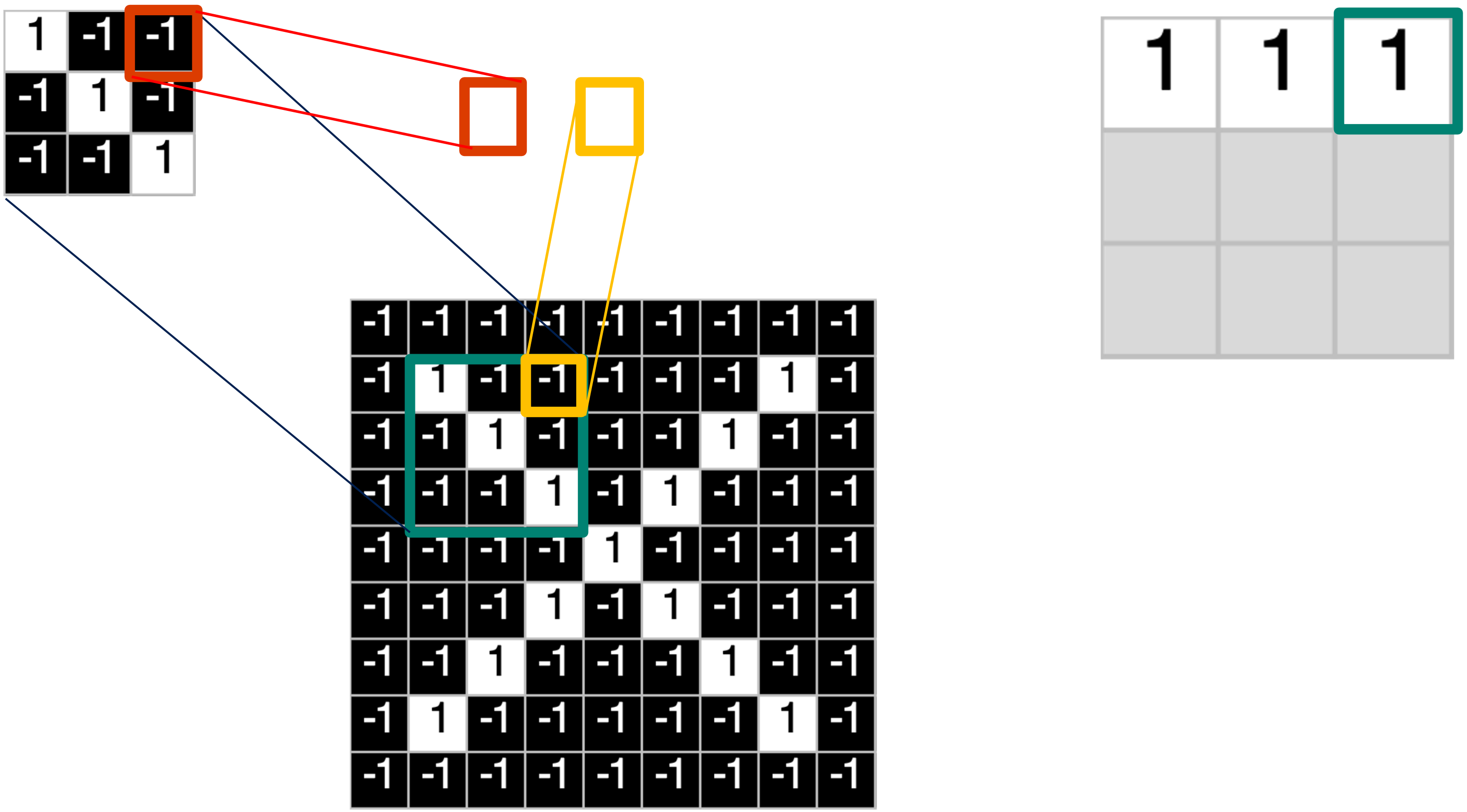
Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Filtering: The math behind the match



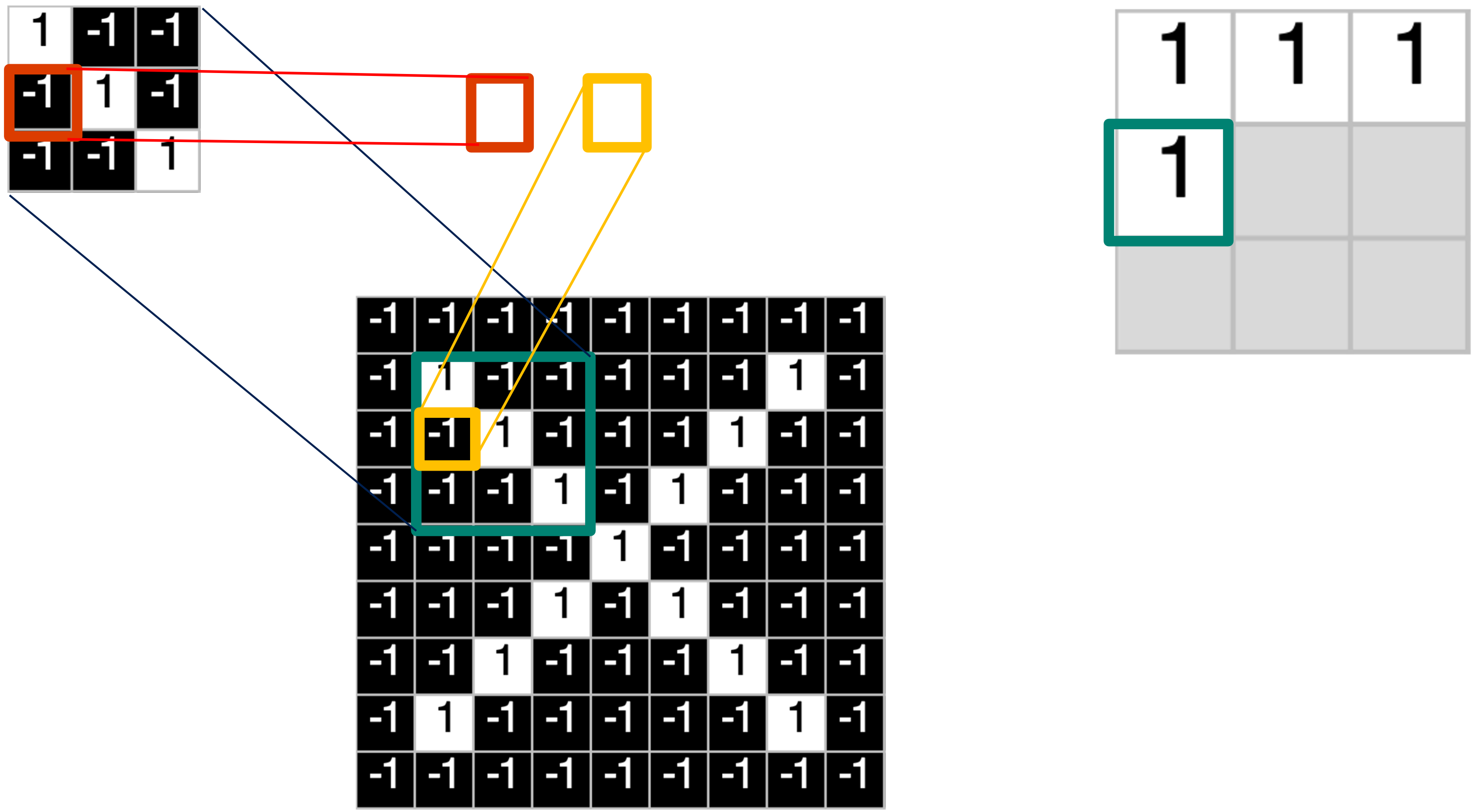
Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Filtering: The math behind the match



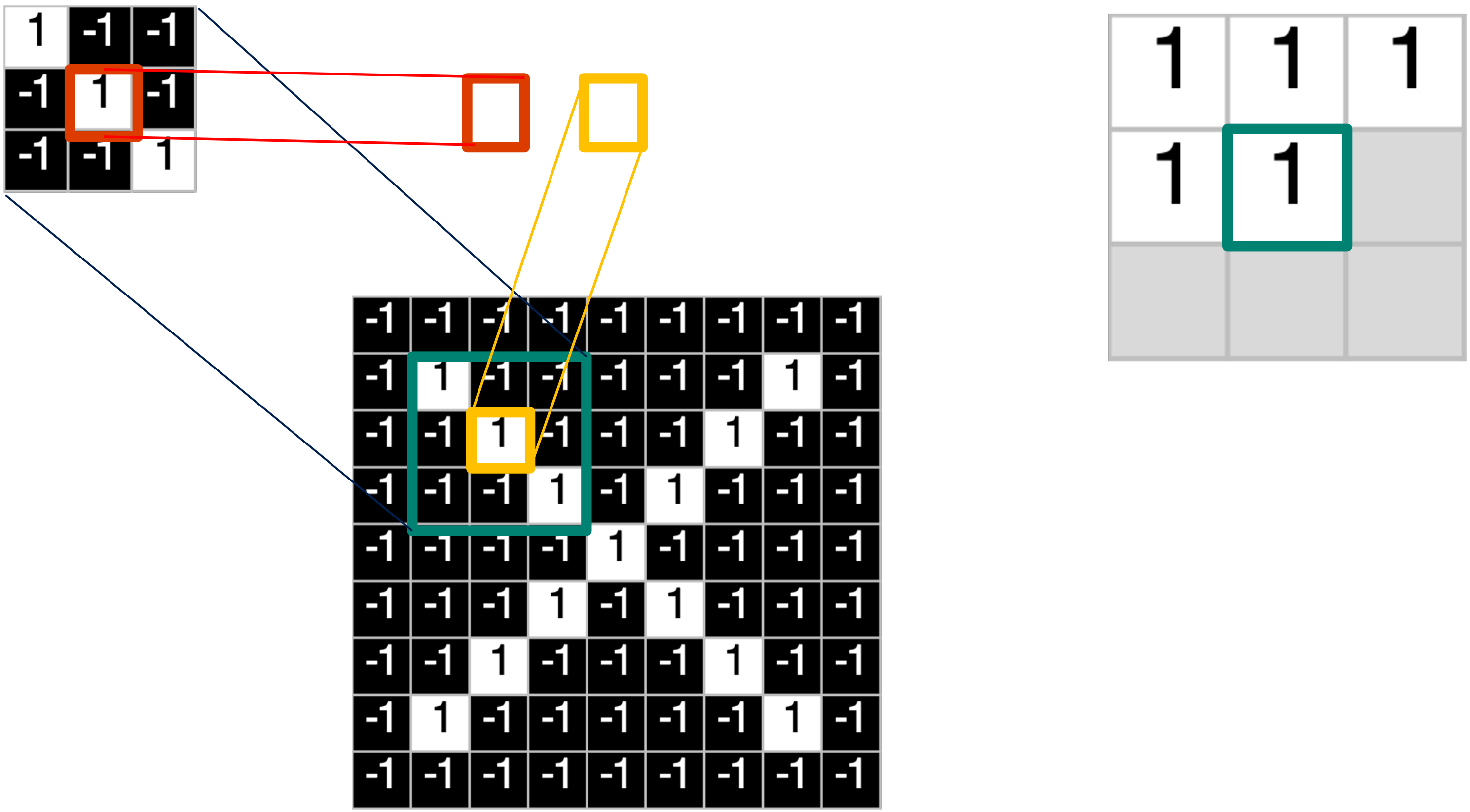
Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Filtering: The math behind the match



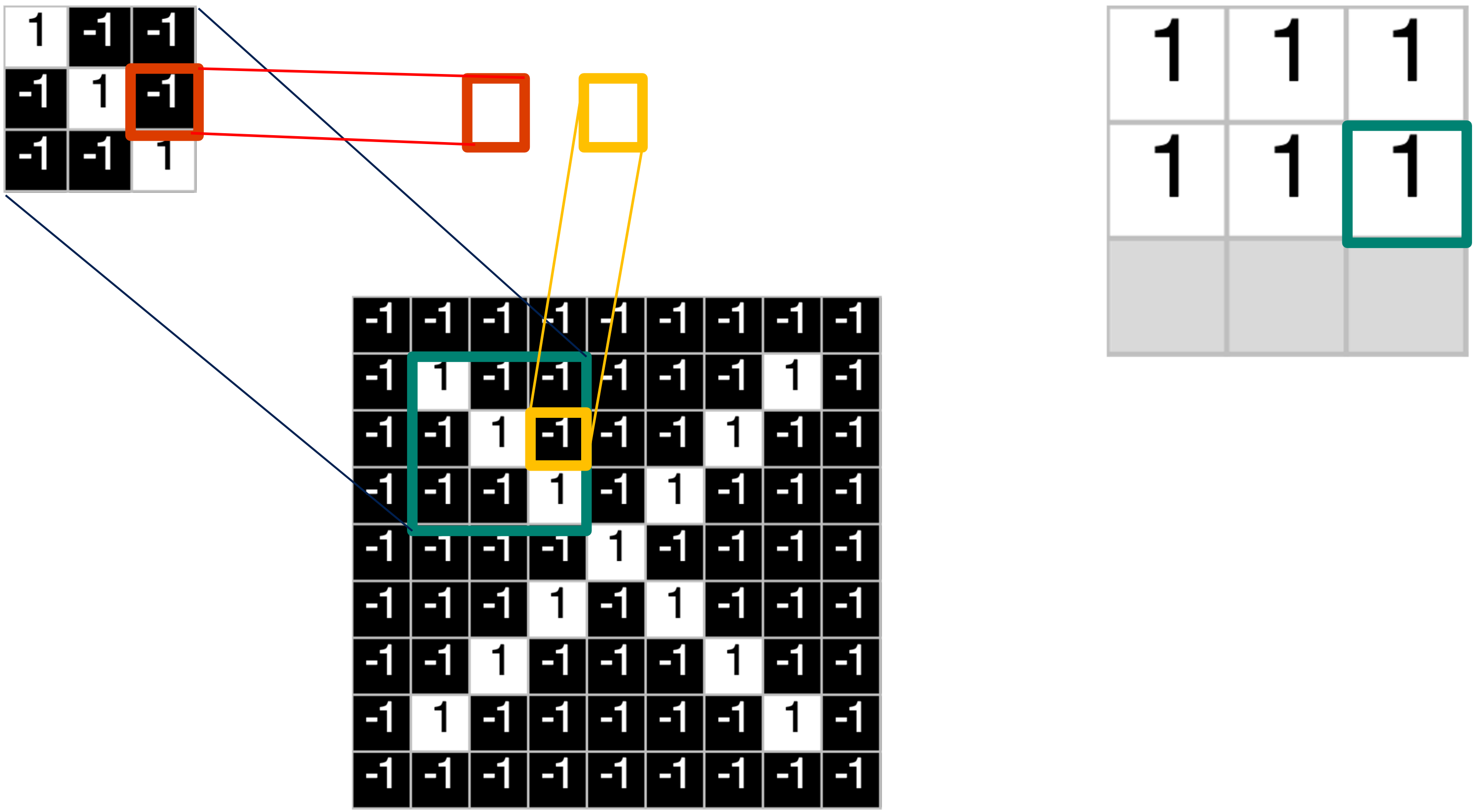
Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Filtering: The math behind the match



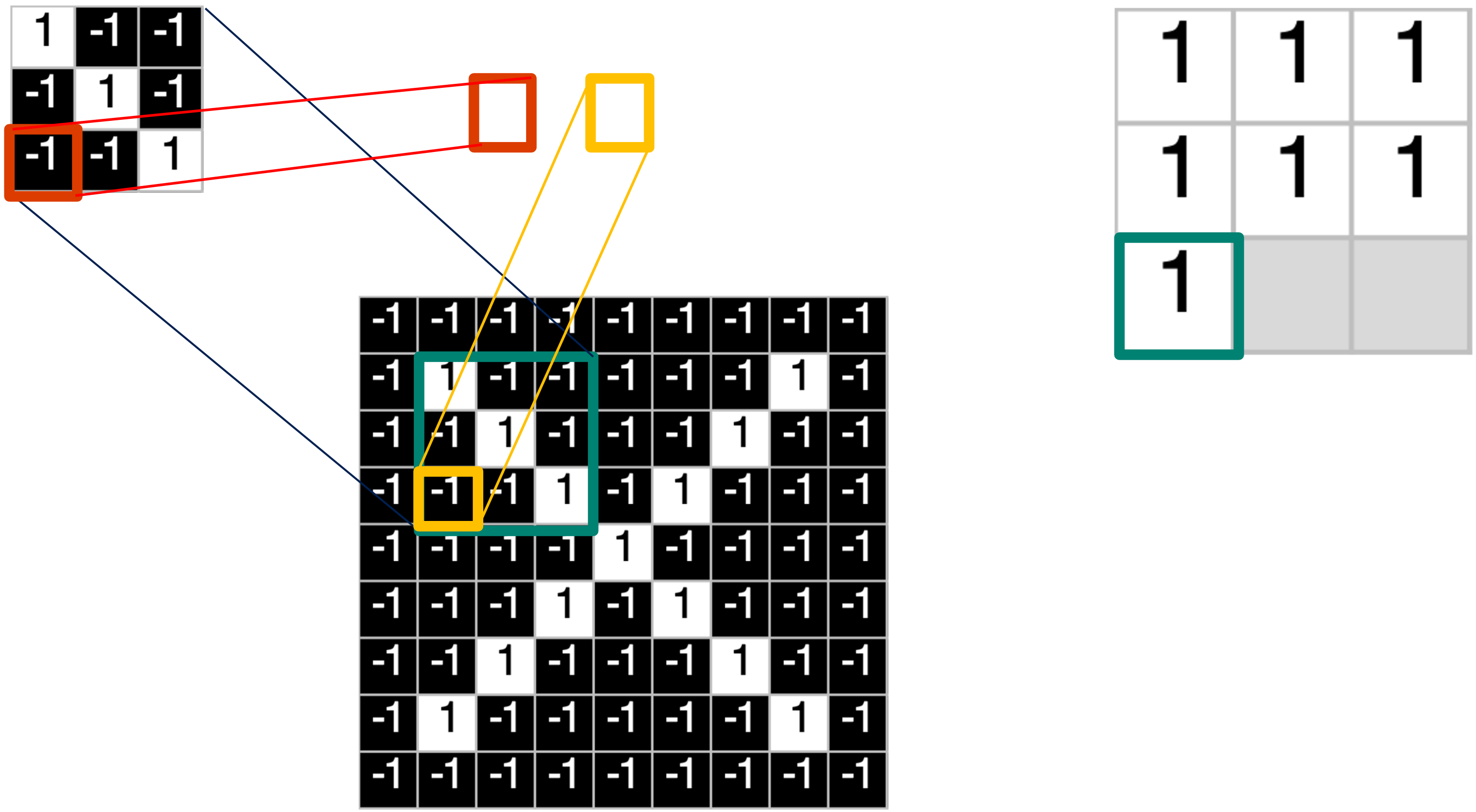
Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Filtering: The math behind the match



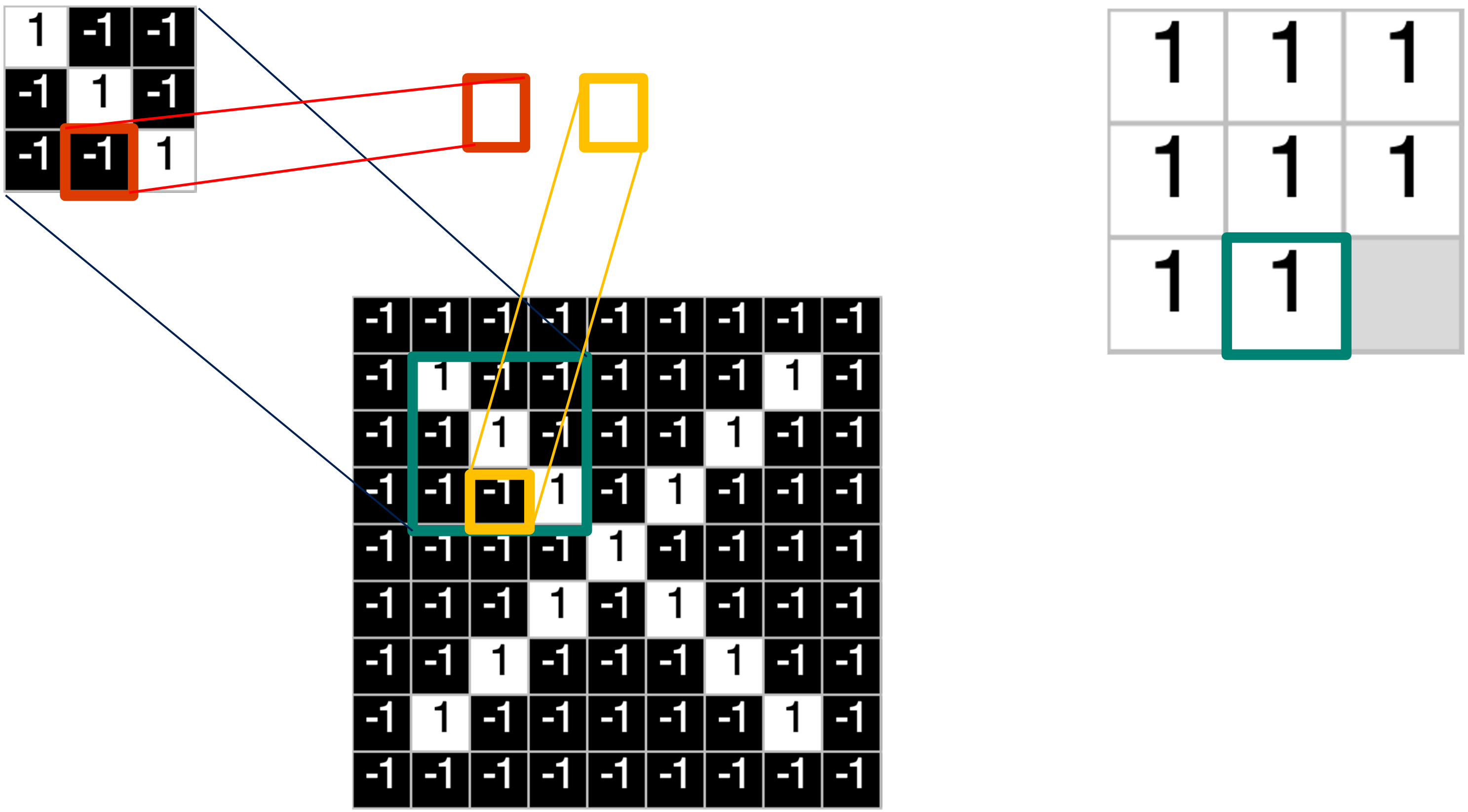
Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Filtering: The math behind the match

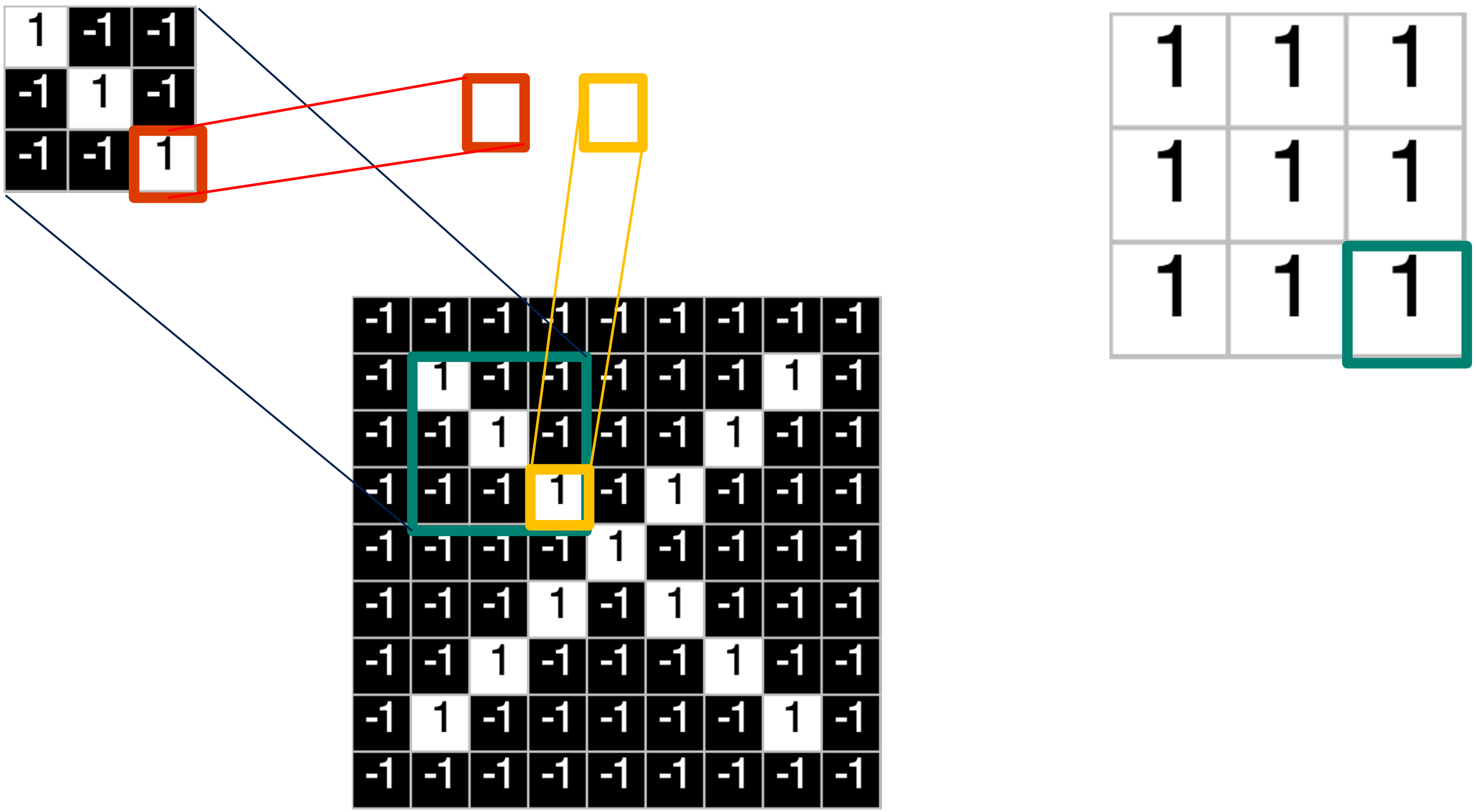


Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Filtering: The math behind the match

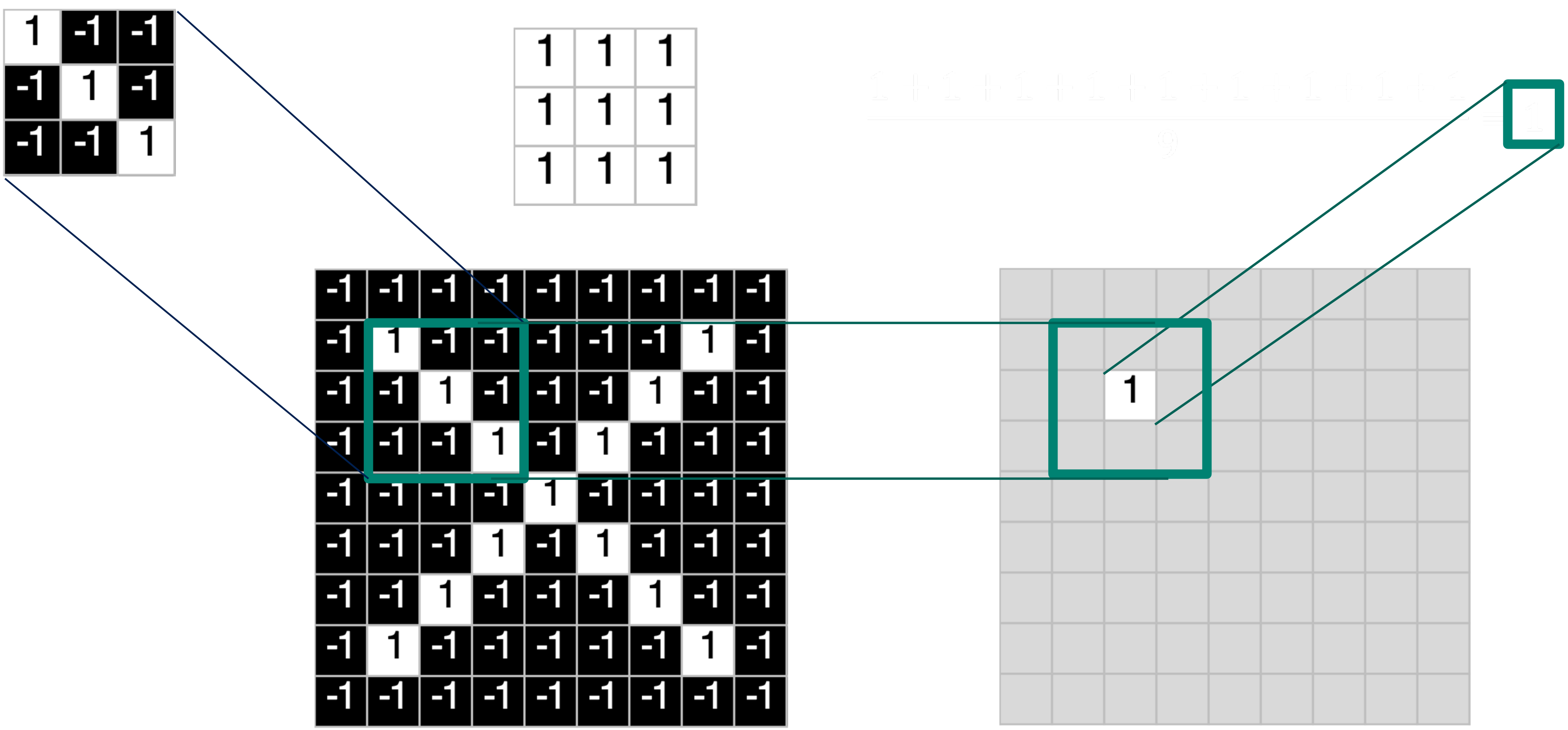


Filtering: The math behind the match

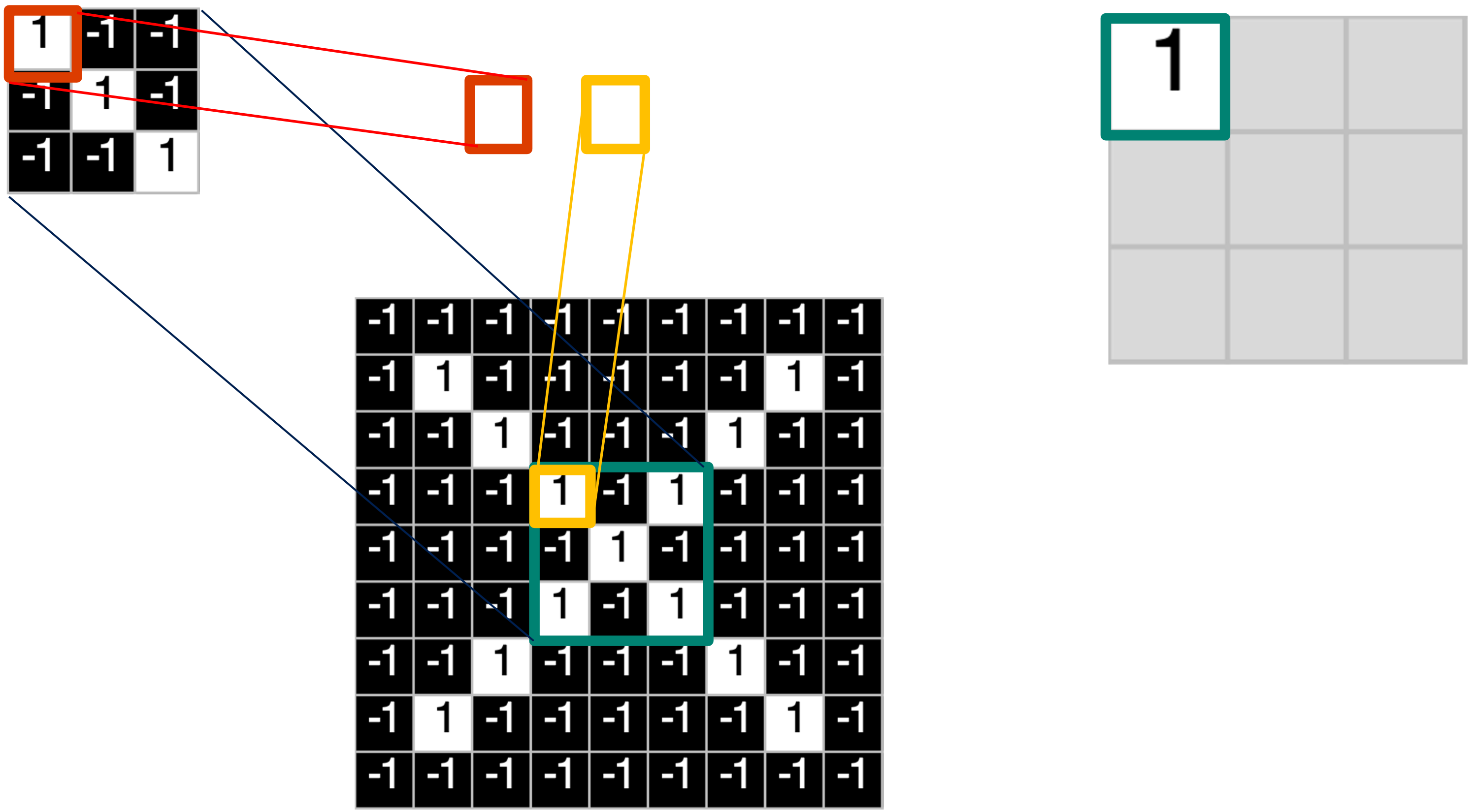


Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Filtering: The math behind the match

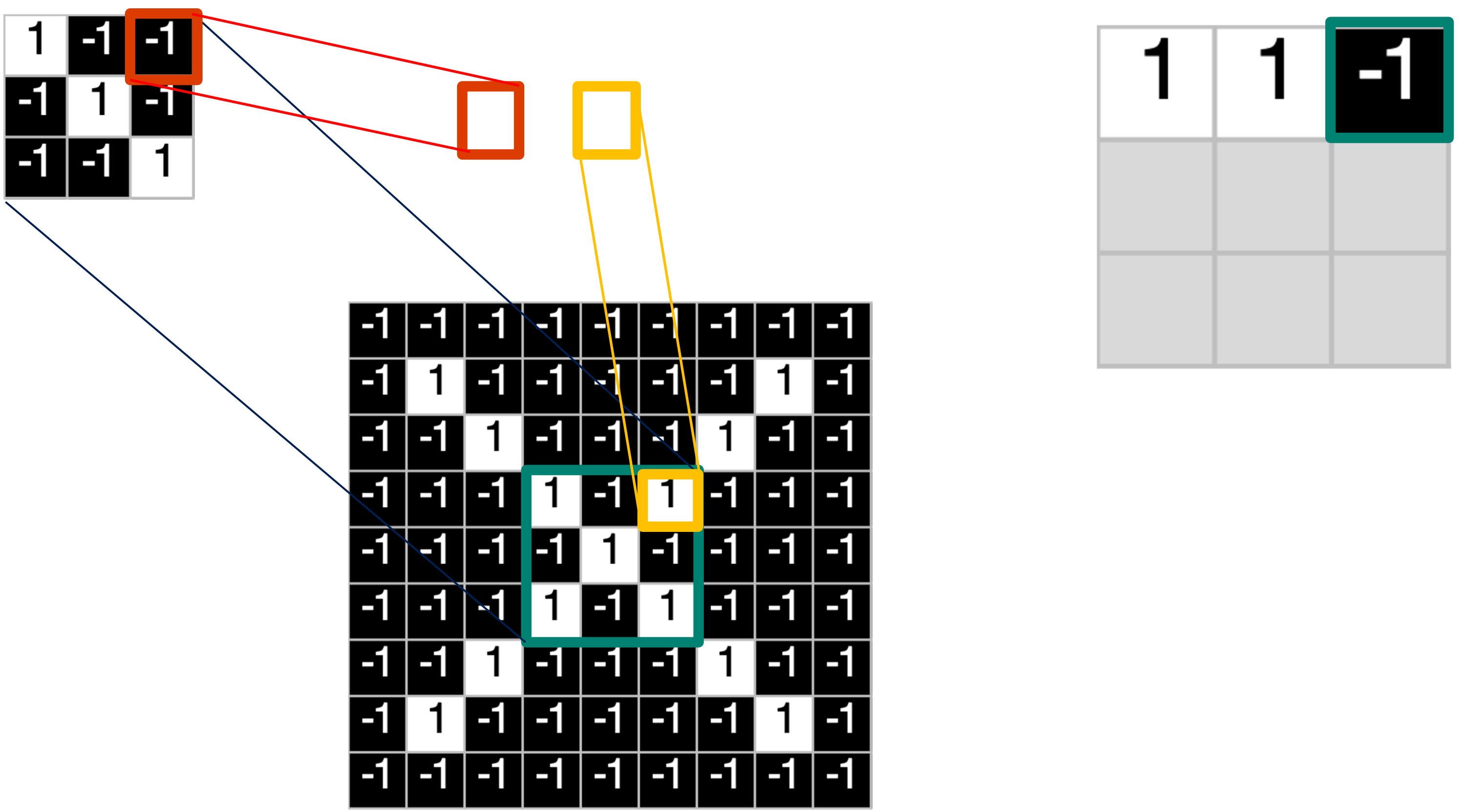


Filtering: The math behind the match



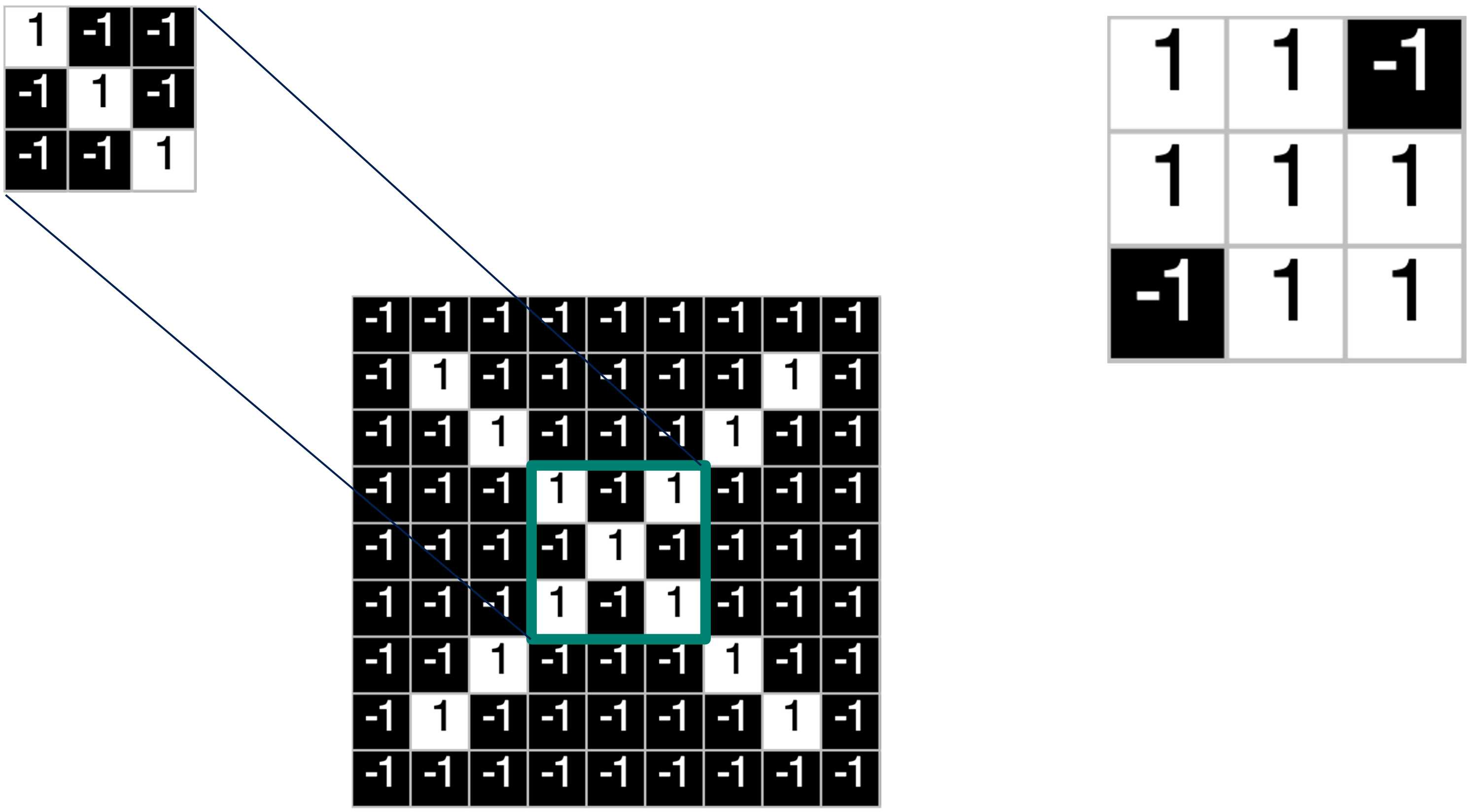
Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Filtering: The math behind the match

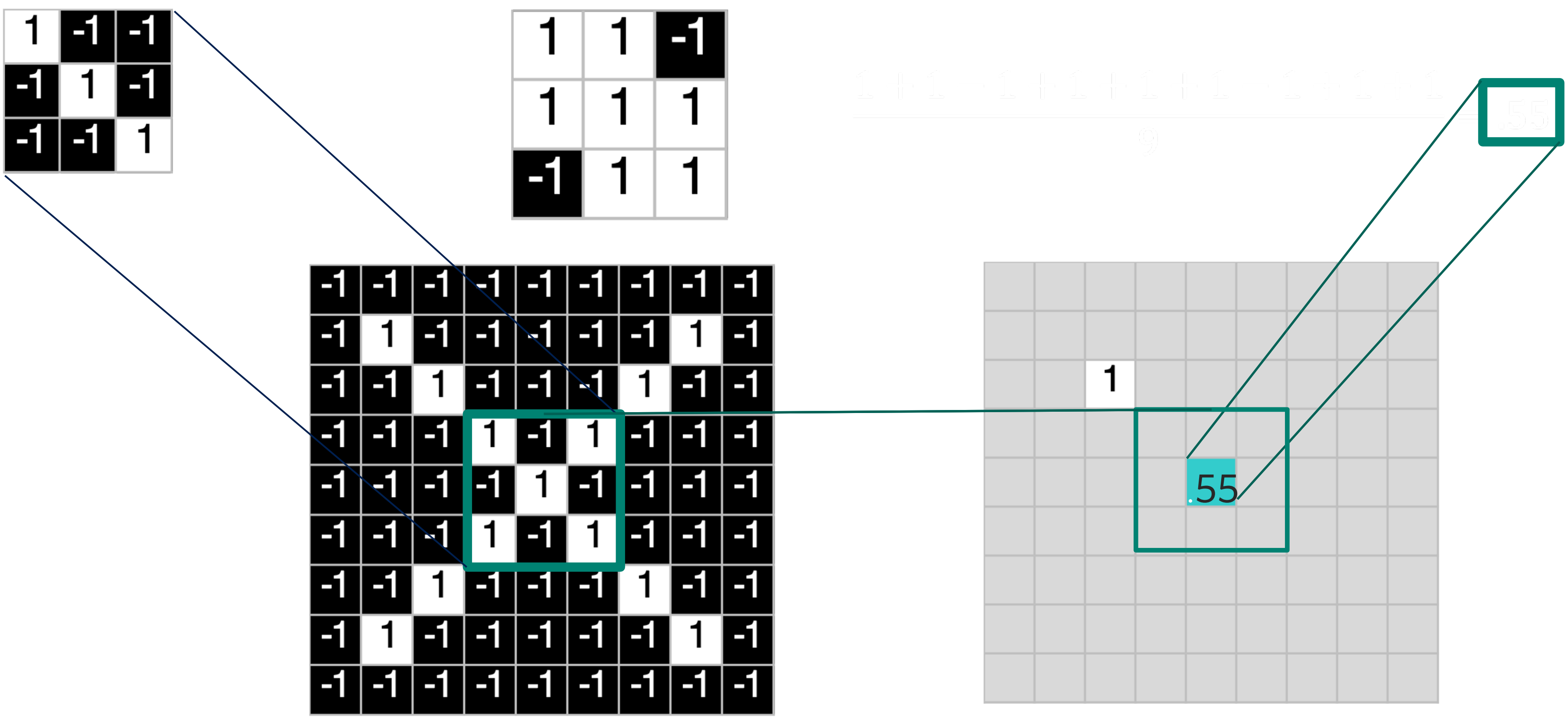


Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Filtering: The math behind the match



Filtering: The math behind the match



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

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

Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

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

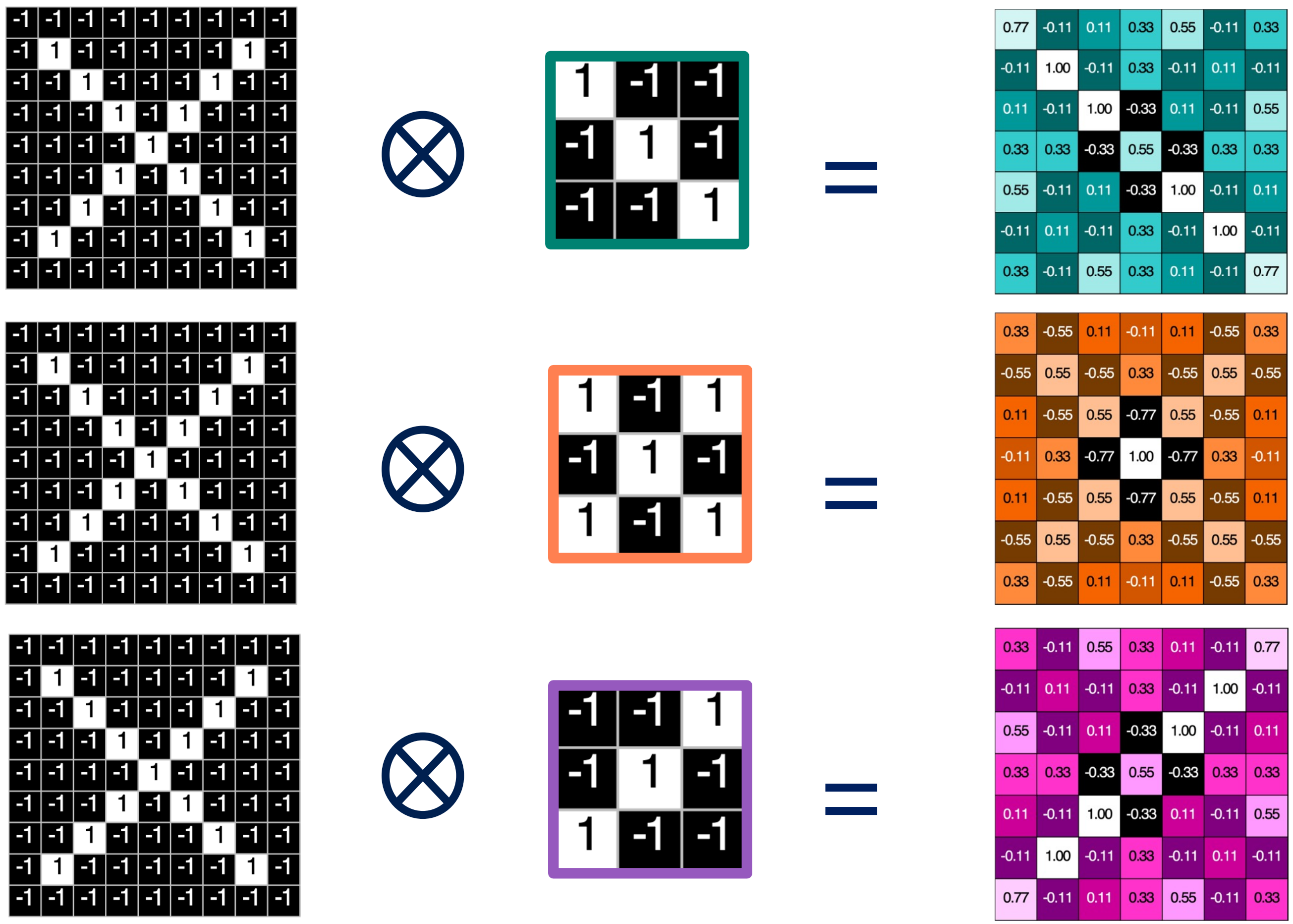
$\otimes$

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

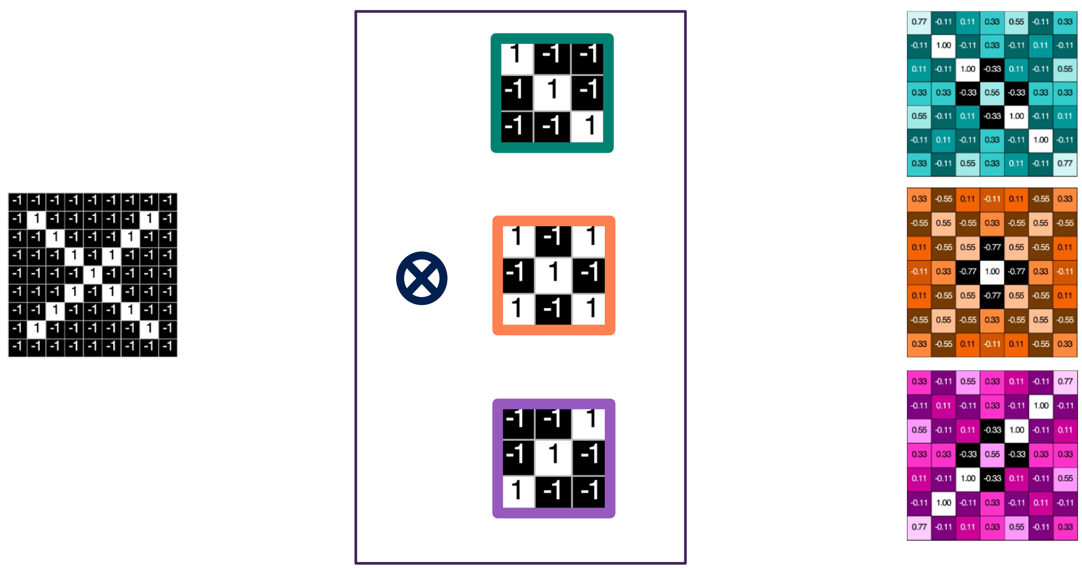
Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Convolution layer

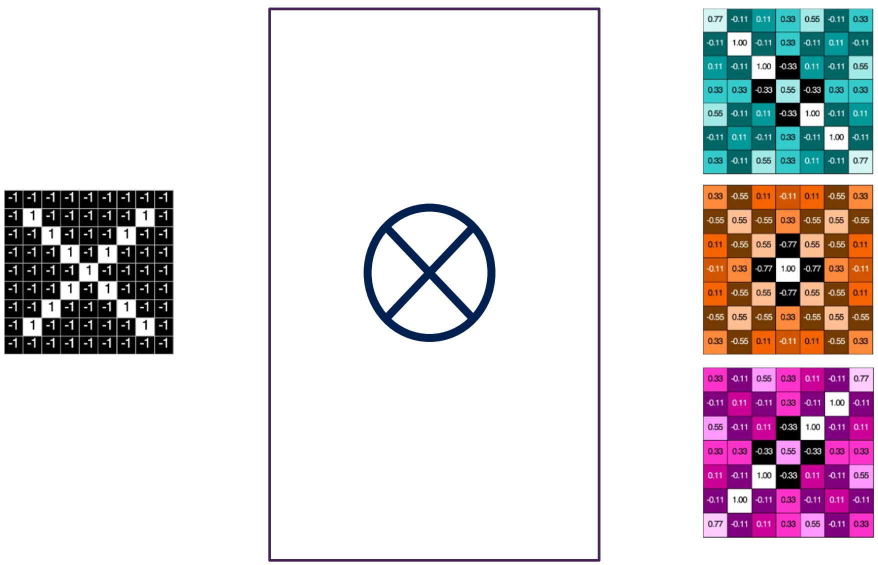
One image becomes a stack of filtered images



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Convolution layer

One image becomes a stack of filtered images



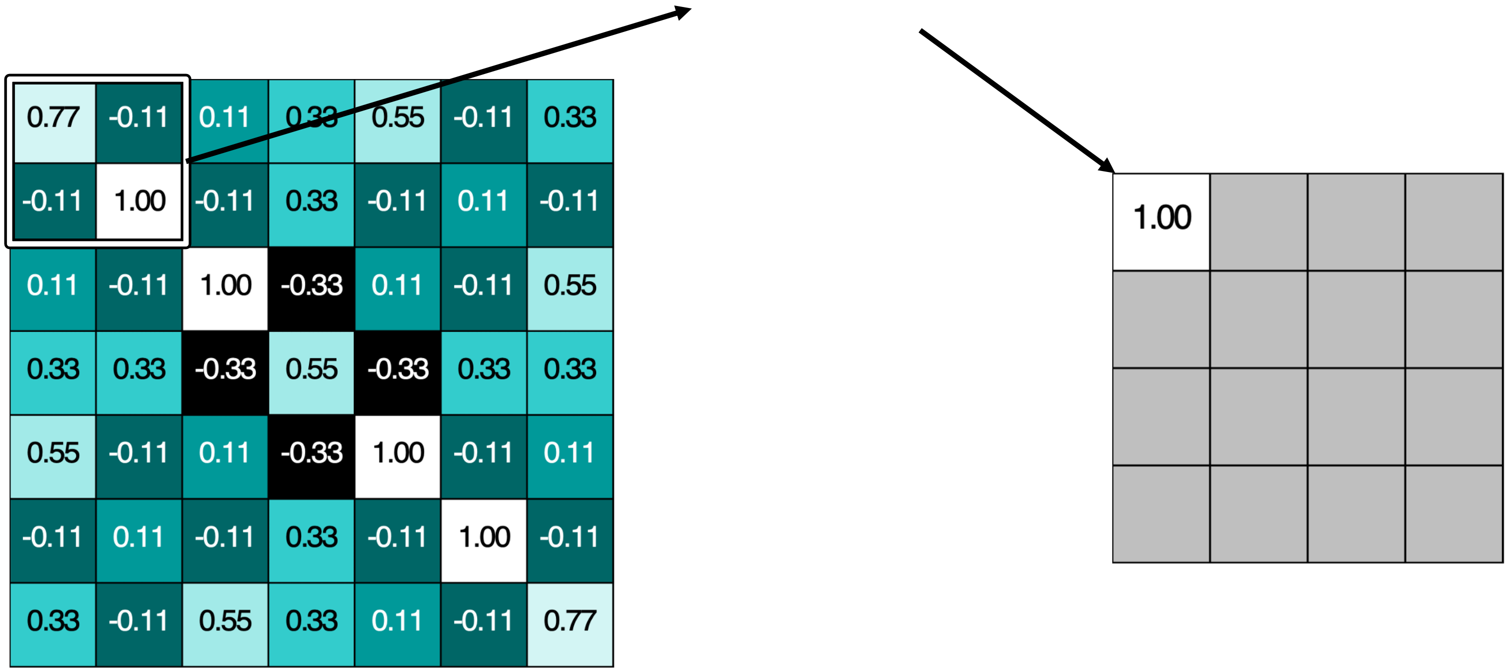
Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

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.

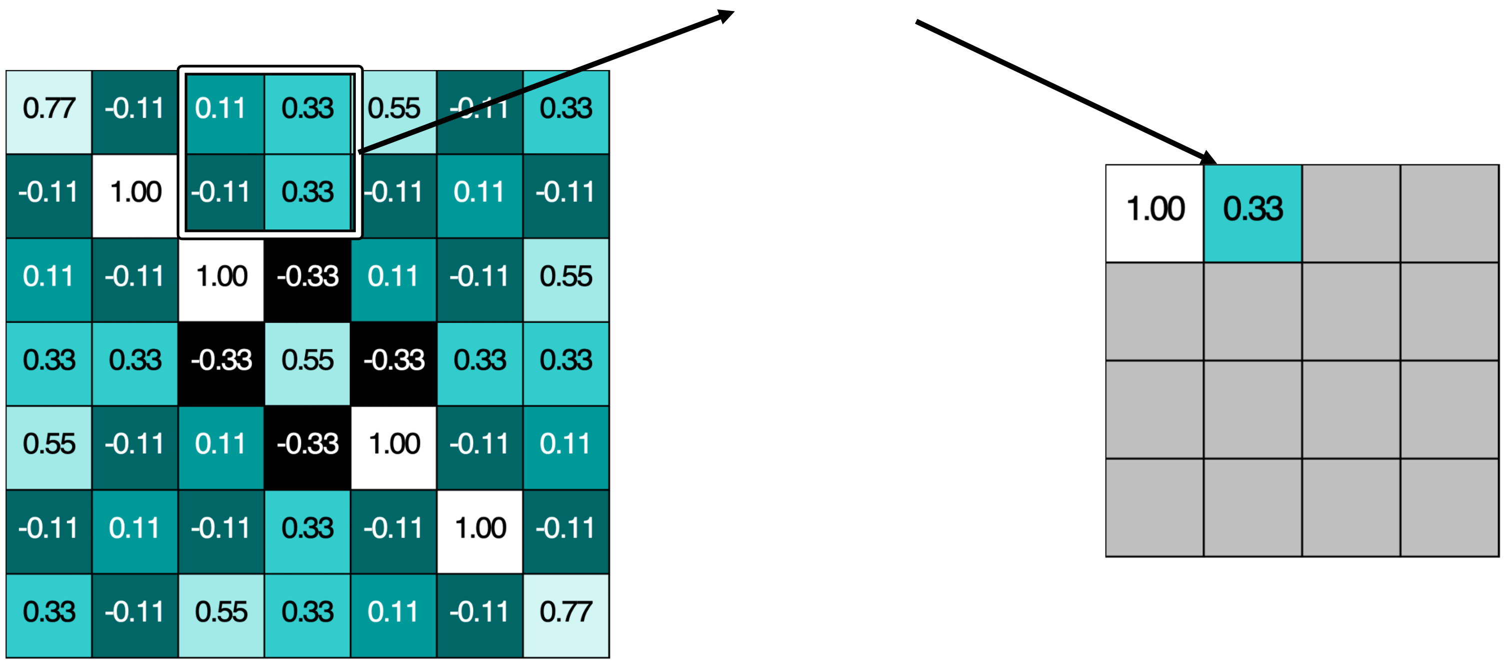
Pooling

- Max Pooling



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Pooling



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

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	

Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

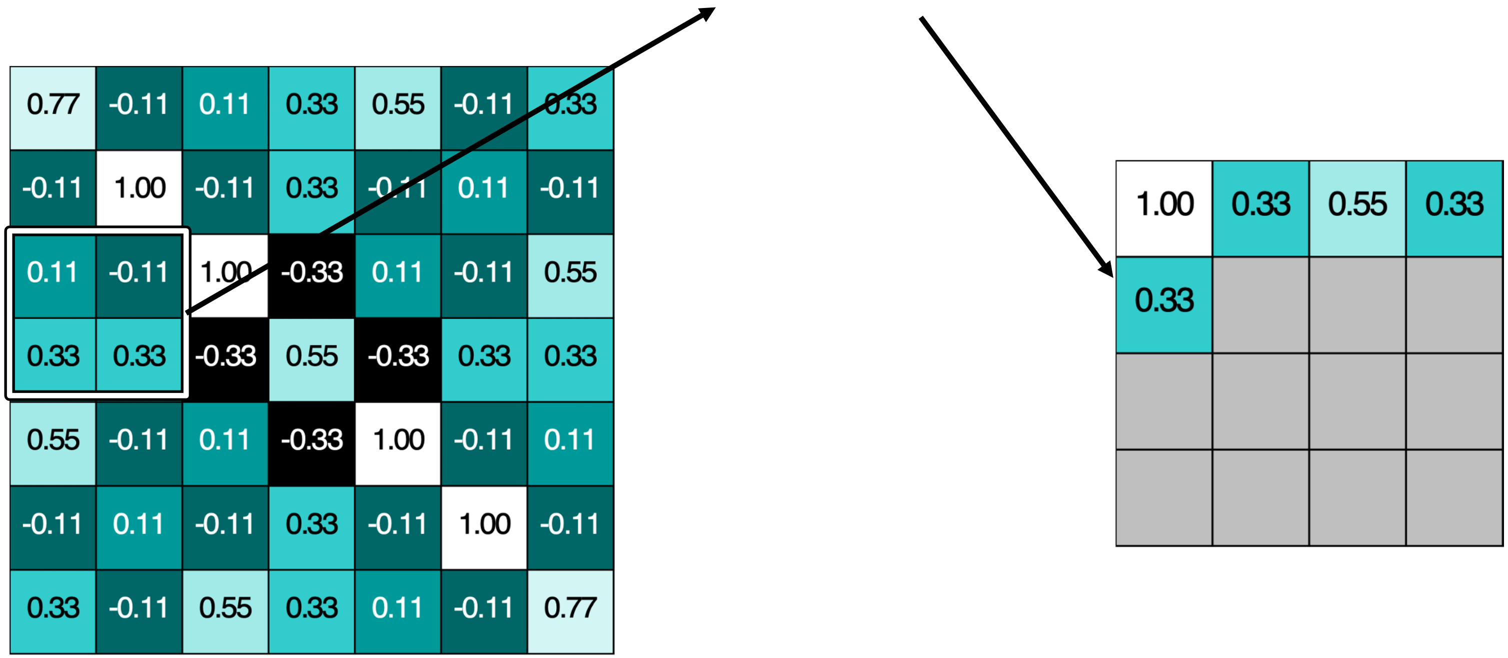
Pooling

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

Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

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

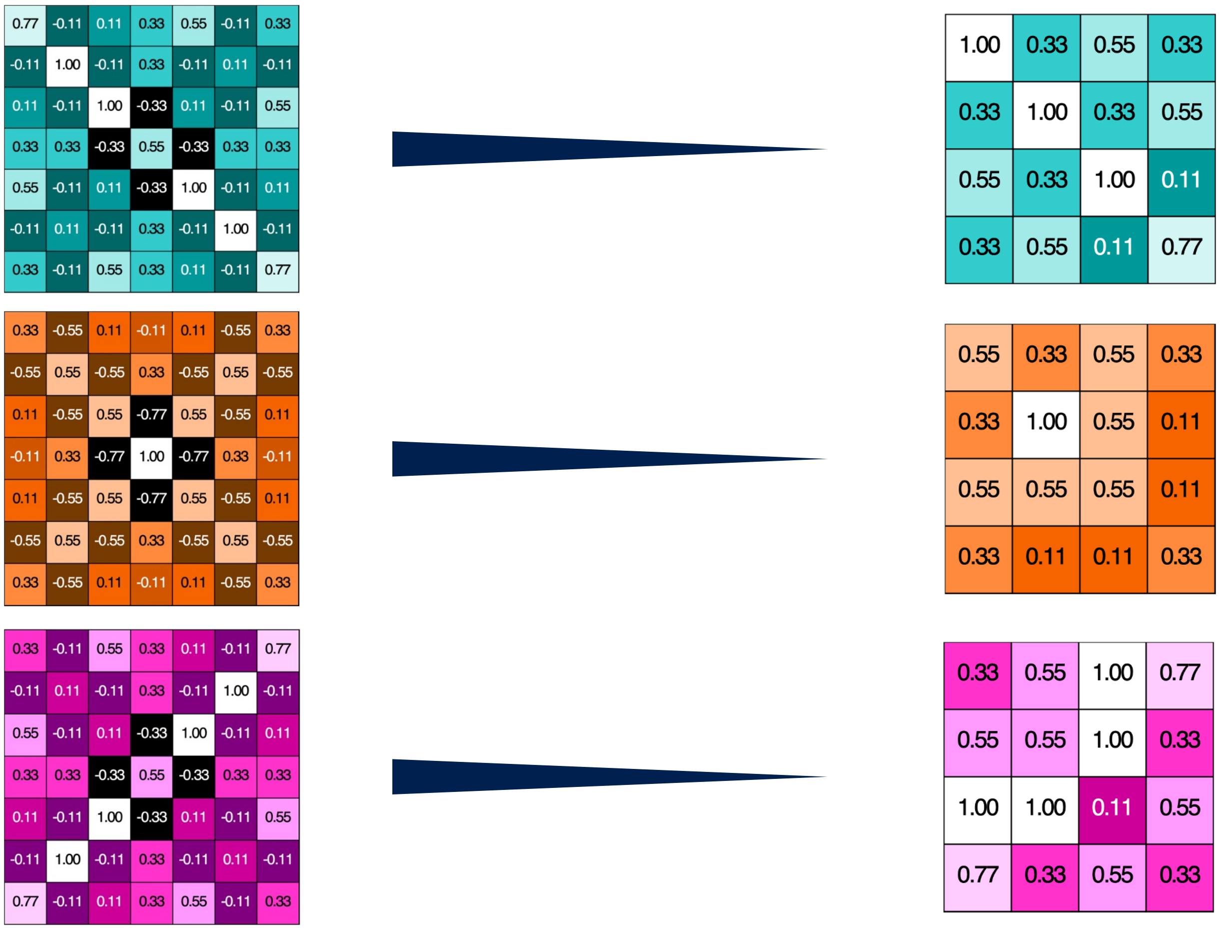
Pooling

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

Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

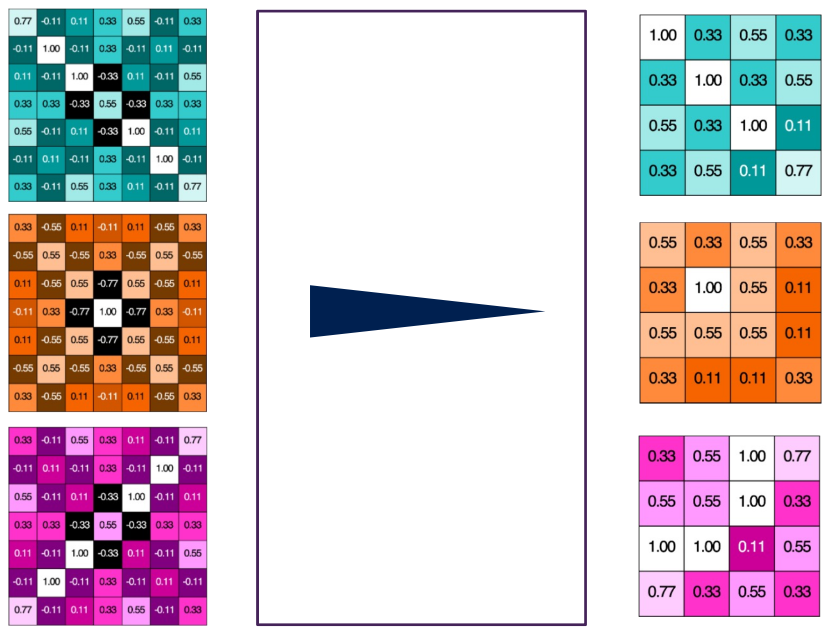
Pooling



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Pooling layer

A stack of images becomes a stack of smaller images.



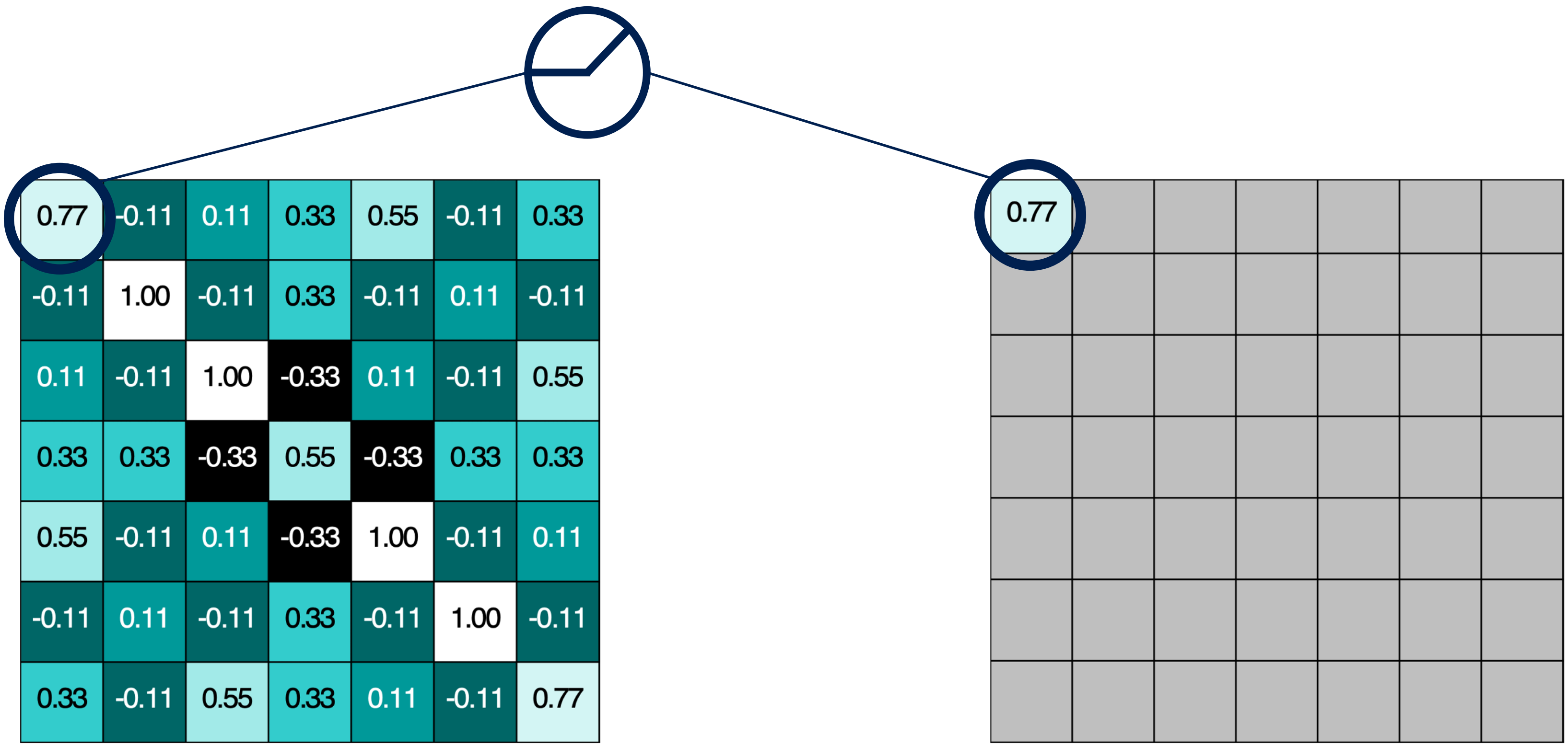
Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Normalization

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

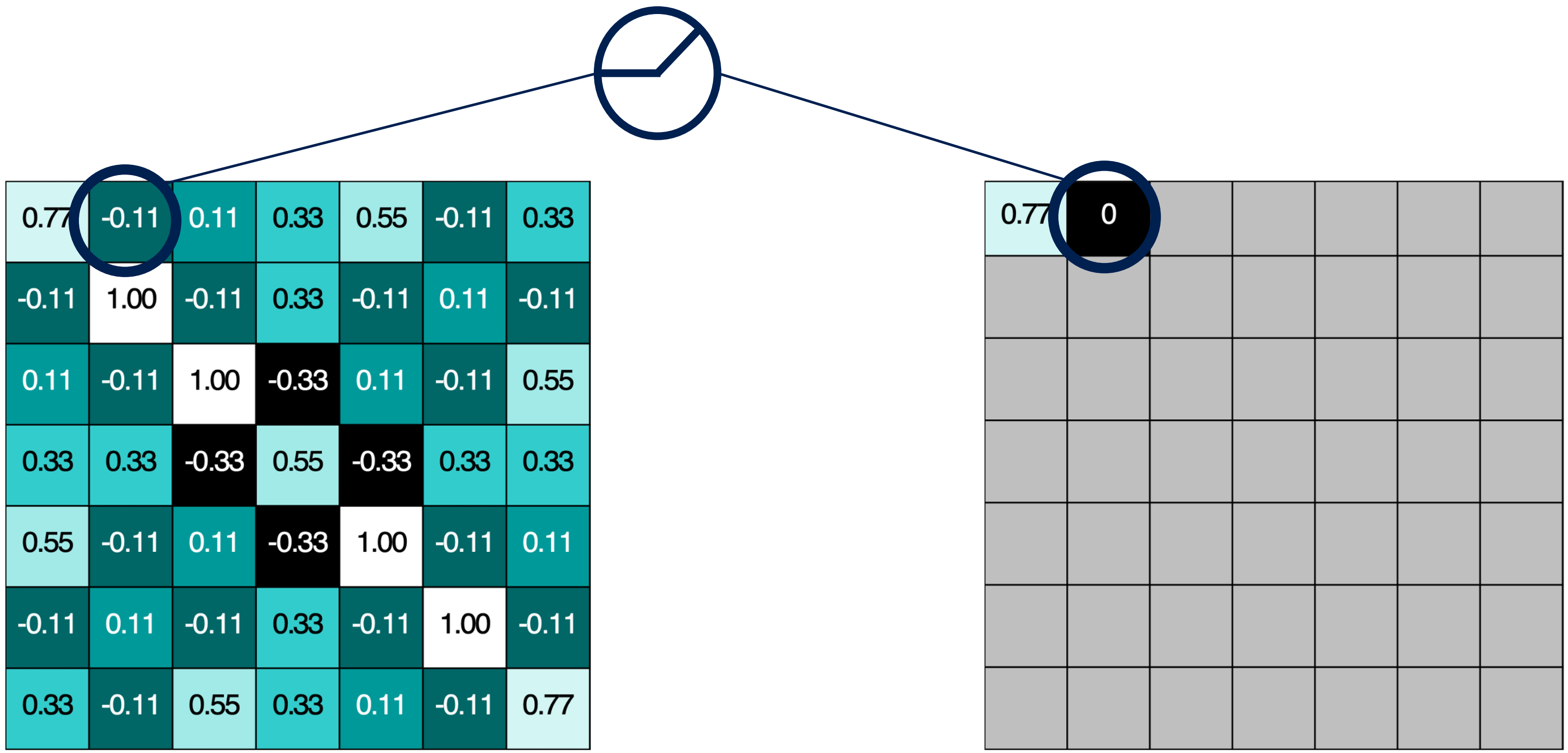
Change everything negative to zero.

Rectified Linear Units (ReLUs)

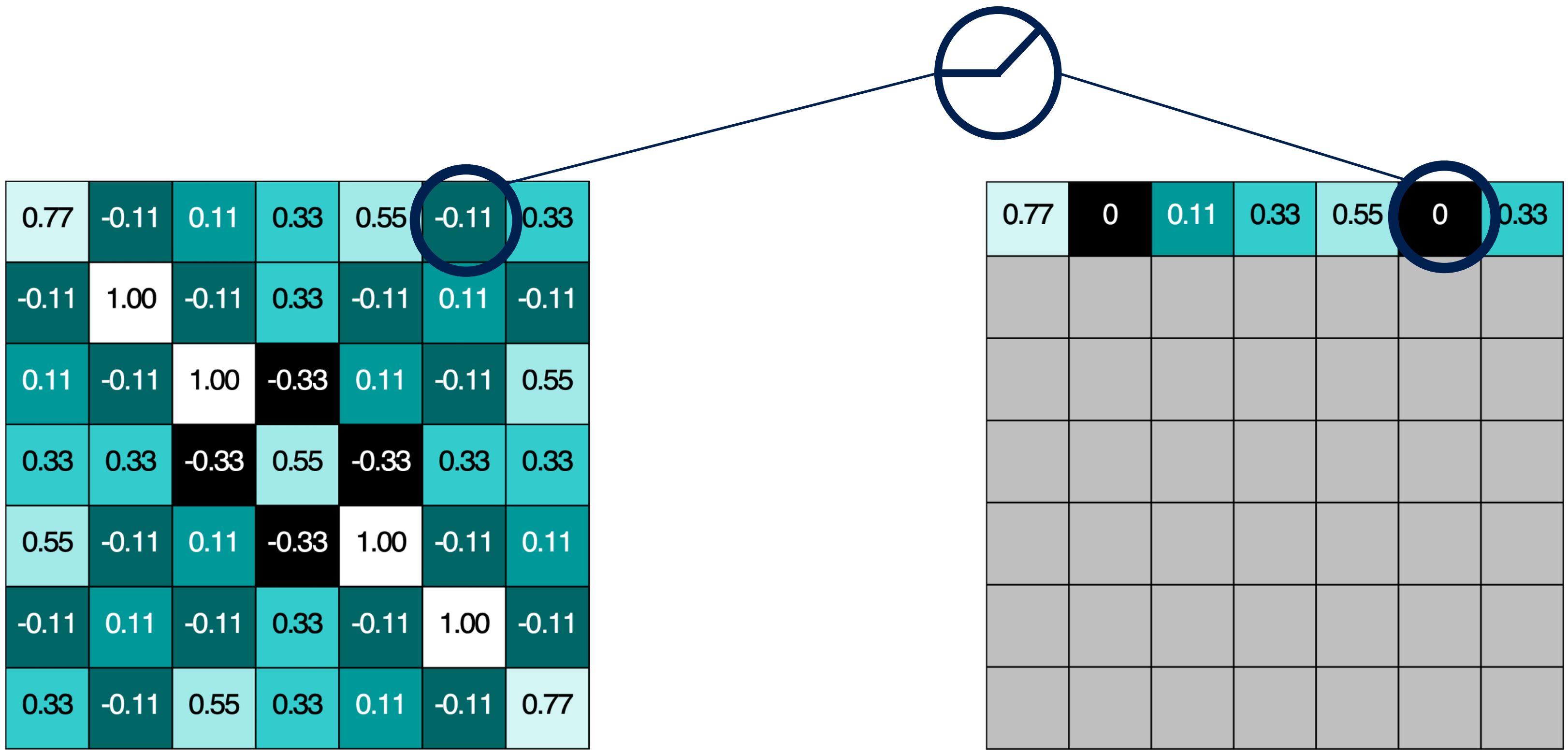


Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Rectified Linear Units (ReLUs)



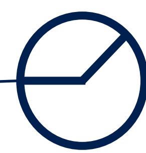
Rectified Linear Units (ReLUs)



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

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

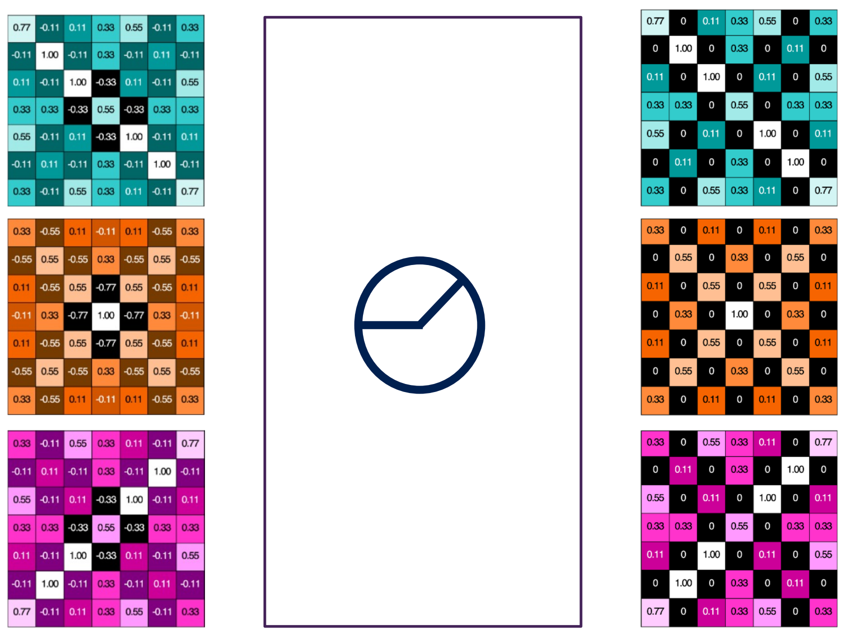


0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

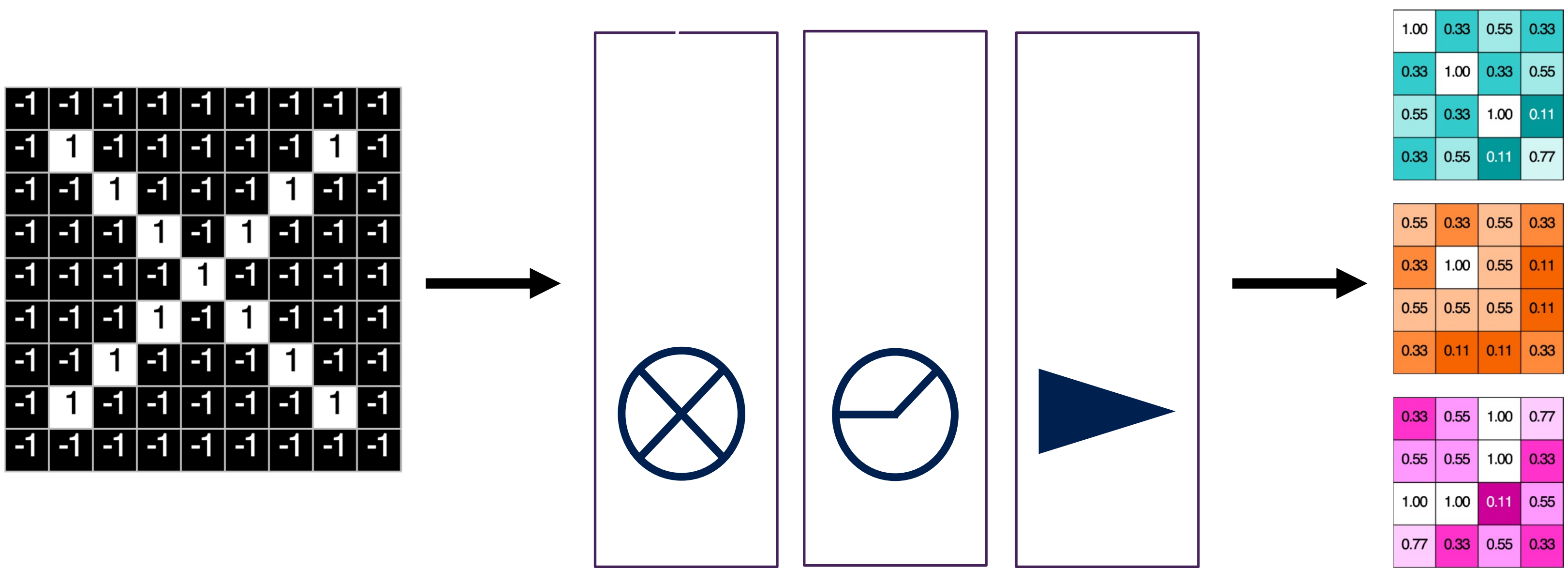
ReLU layer

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



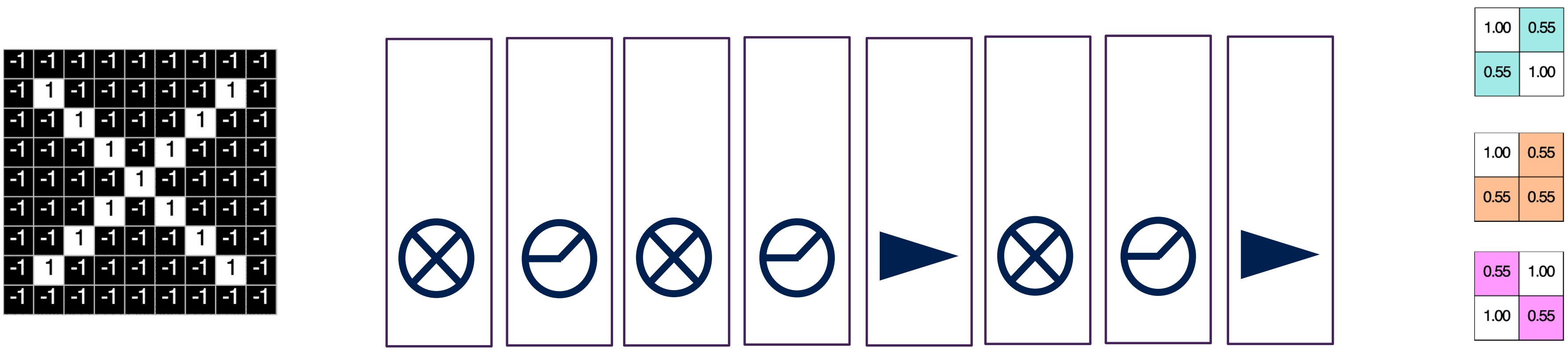
Layers get stacked

The output of one becomes the input of the next.



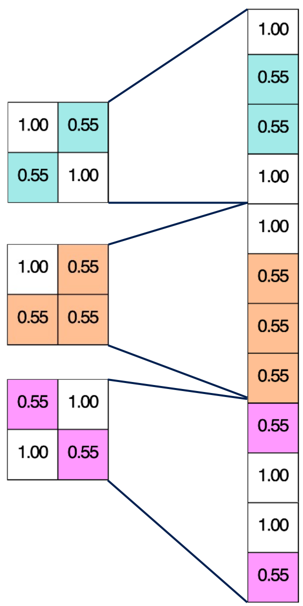
Deep stacking

Layers can be repeated several (or many) times.



Fully connected layer

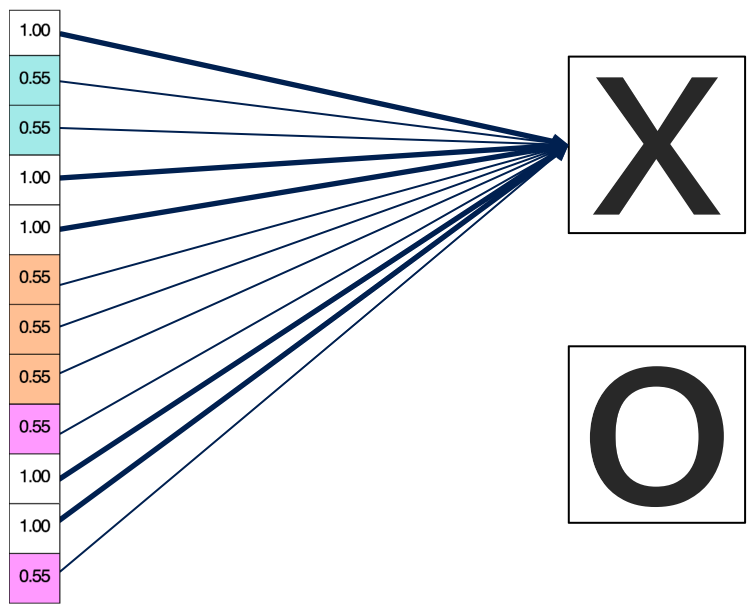
Every value gets a vote



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

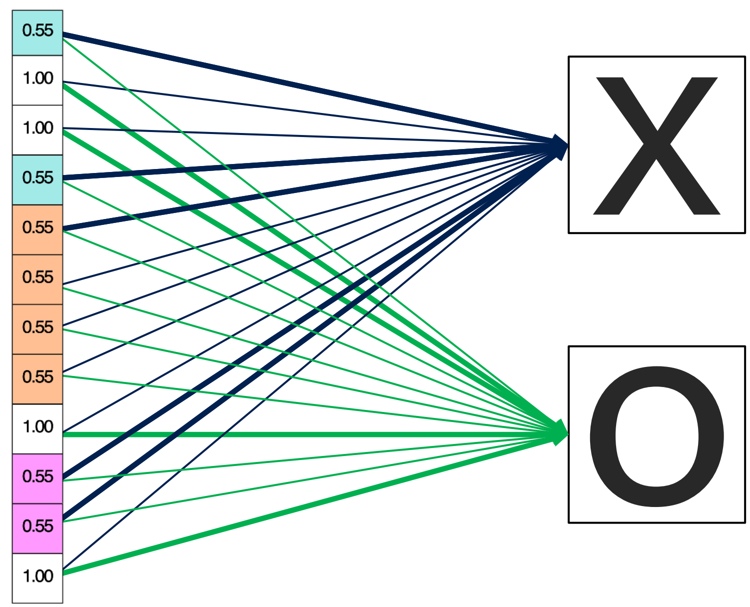
Fully connected layer

Vote depends on how strongly a value predicts X or O



Fully connected layer

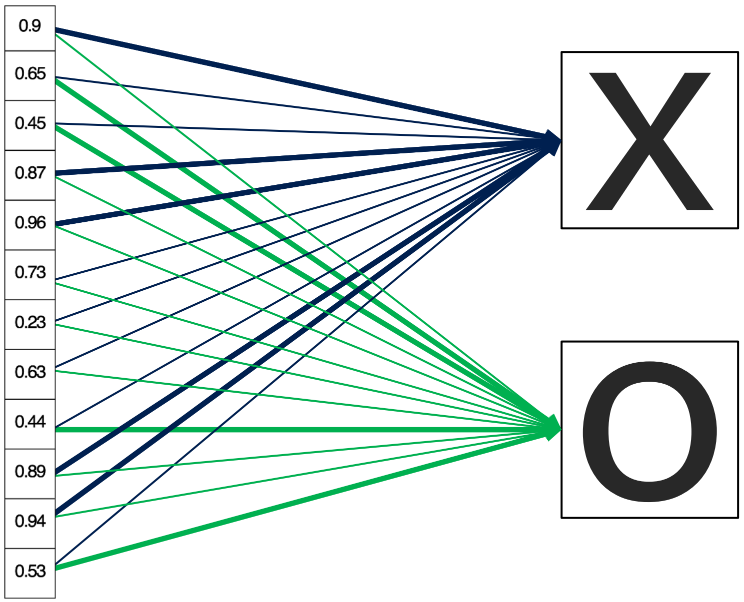
Vote depends on how strongly a value predicts X or O



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Fully connected layer

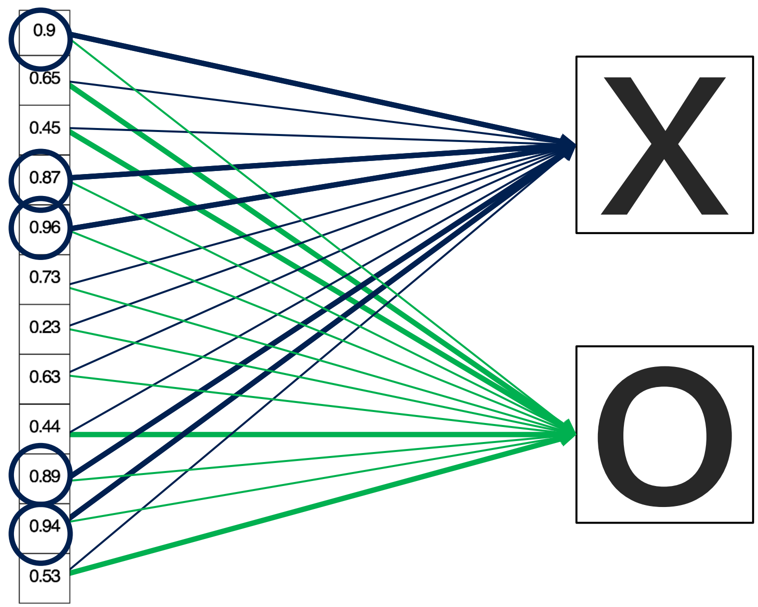
Future values vote on X or O



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Fully connected layer

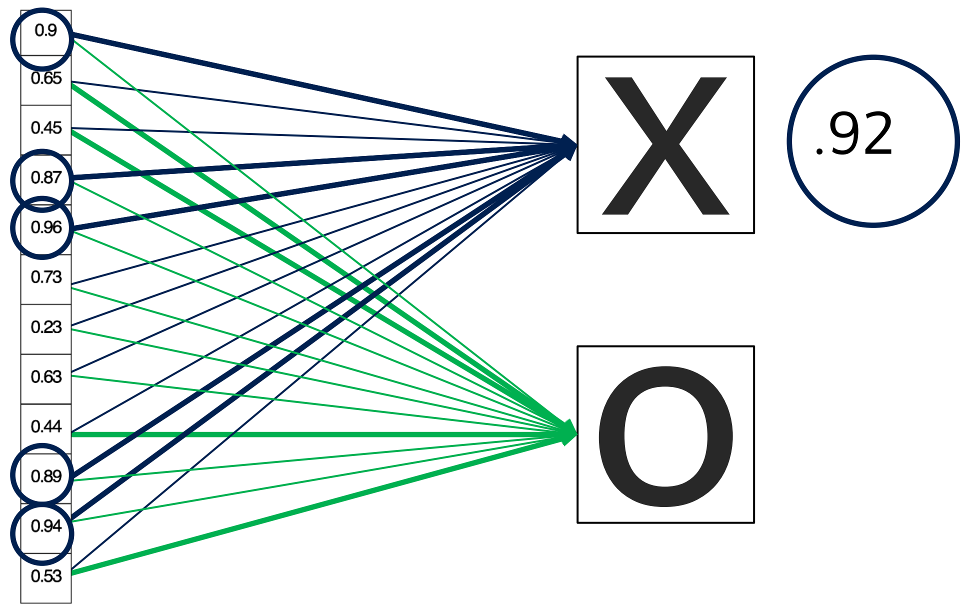
Future values vote on X or O



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Fully connected layer

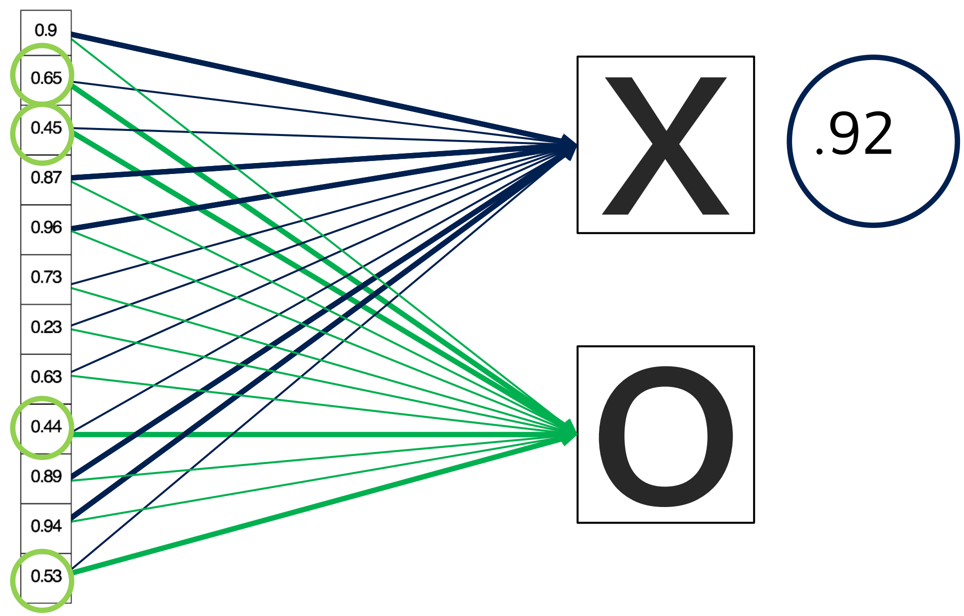
Future values vote on X or O



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

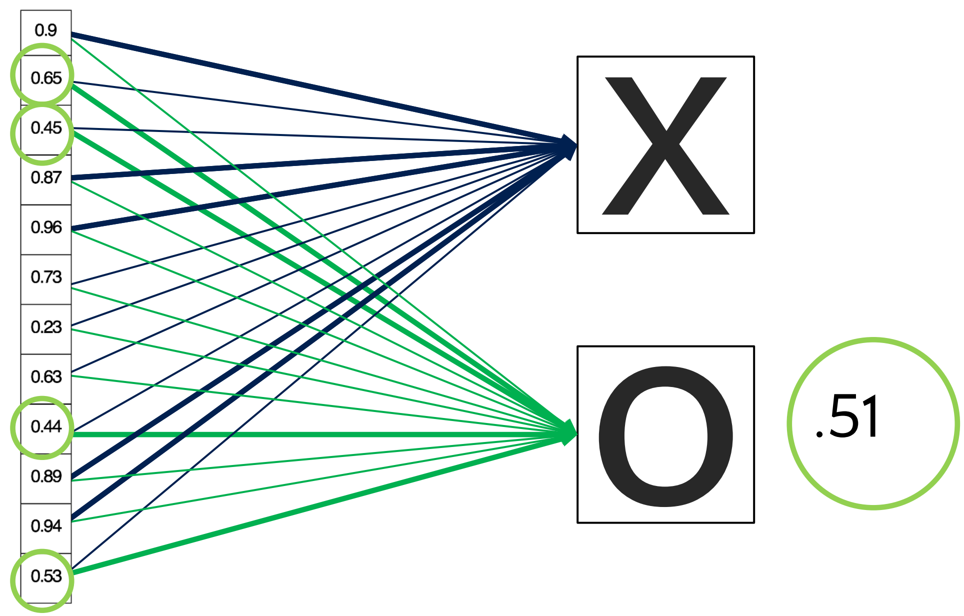
Fully connected layer

Future values vote on X or O



Fully connected layer

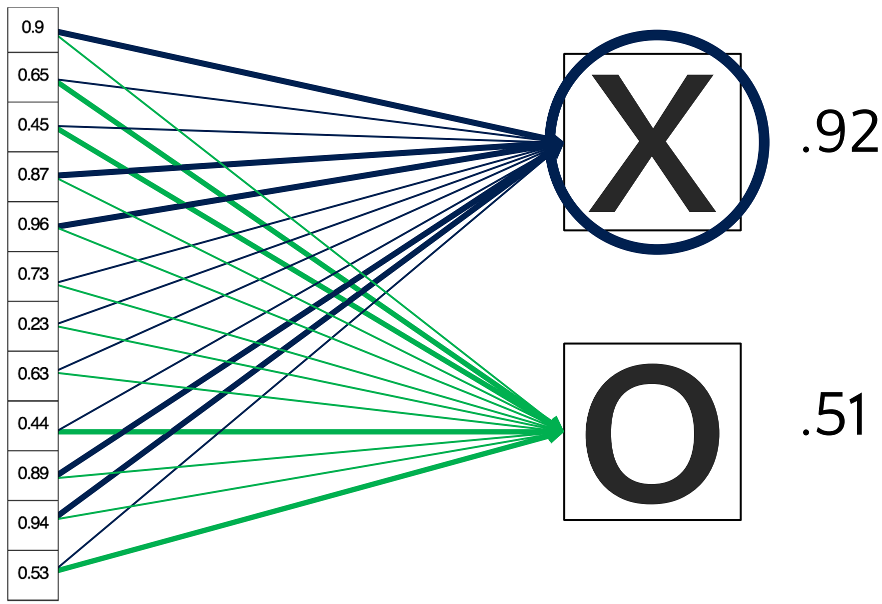
Future values vote on X or O



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Fully connected layer

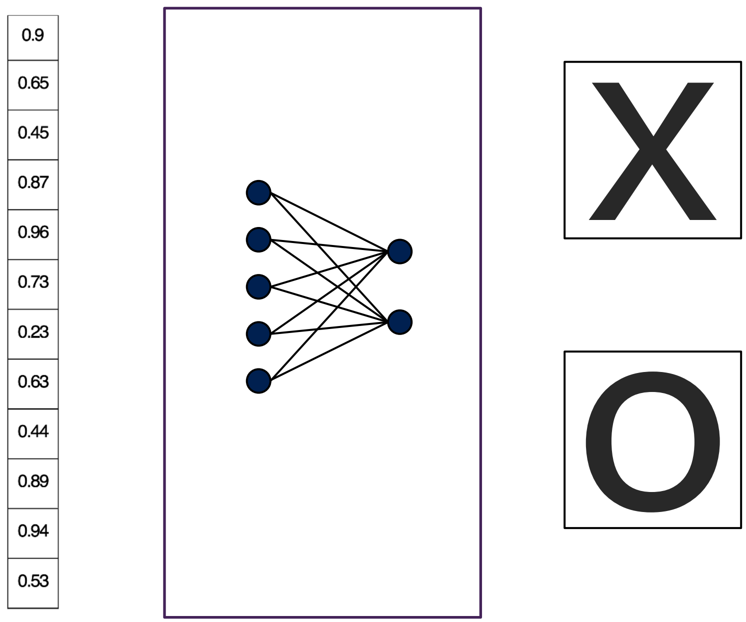
Future values vote on X or O



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

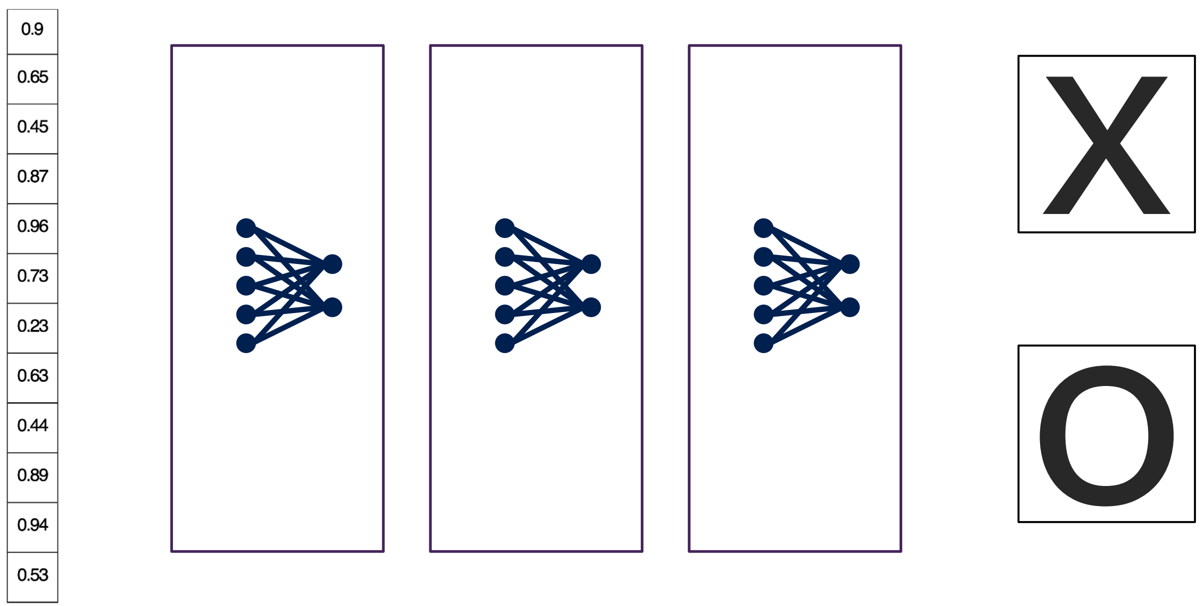
Fully connected layer

A list of feature values becomes a list of votes.



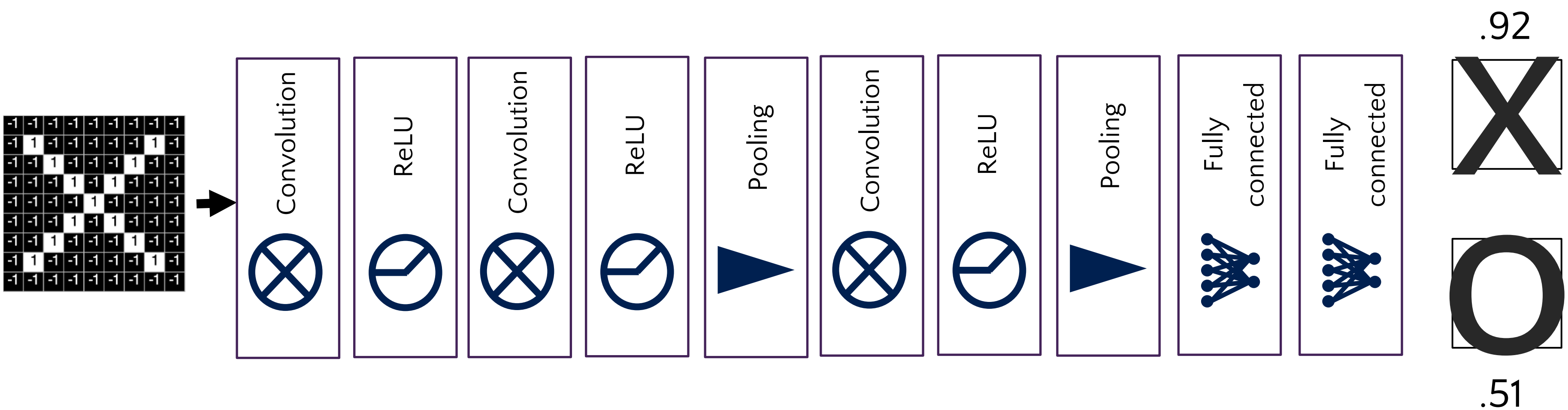
Fully connected layer

These can also be stacked.



Putting it all together

A set of pixels becomes a set of votes.



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Learning

Q: Where do all the magic numbers come from?

Features in convolutional layers

Voting weights in fully connected layers

A: Backpropagation

Hyperparameters (knobs)

Convolution

- Number of features
- Size of features

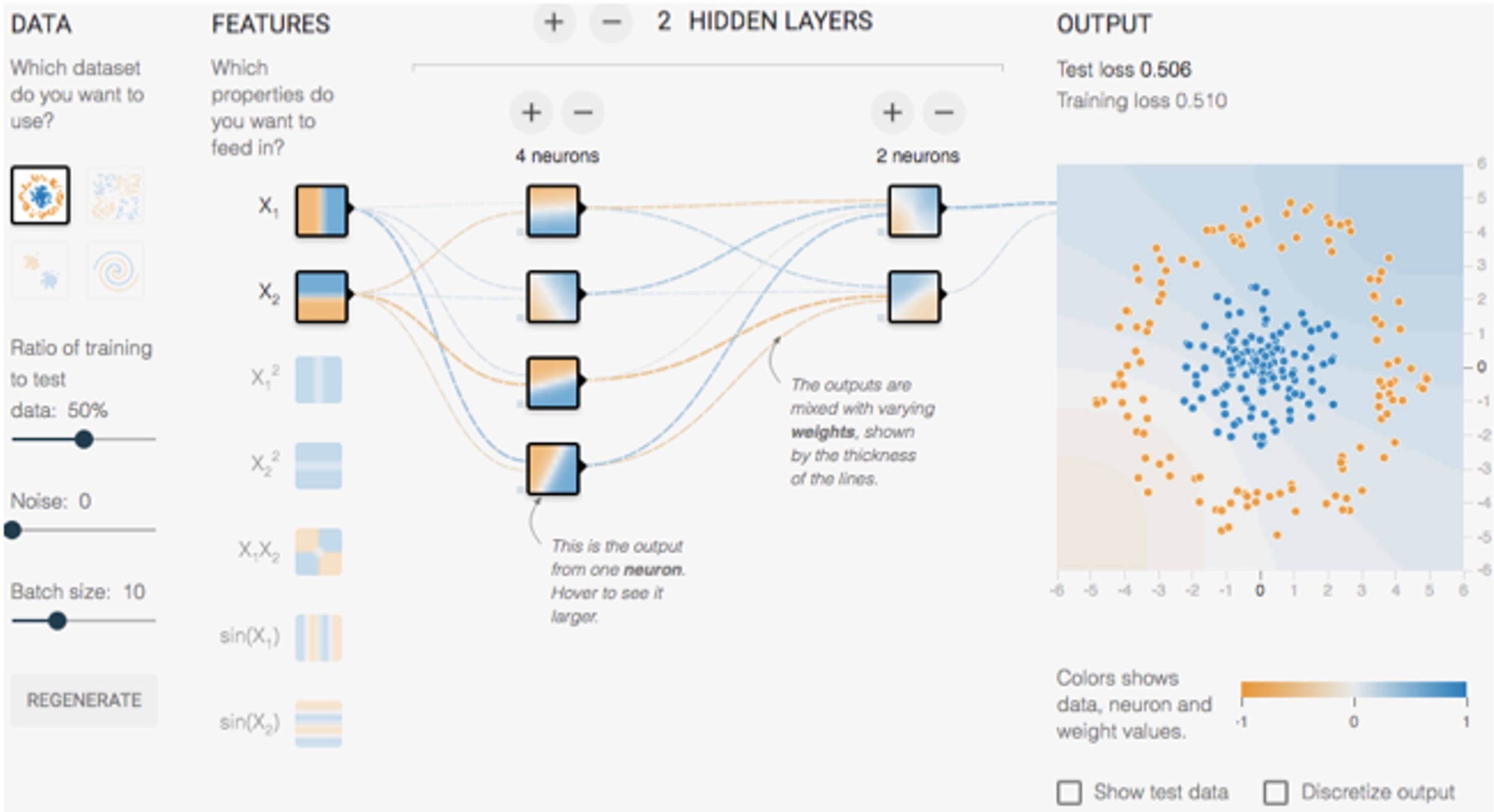
Pooling

- Window size
- Window stride

Fully Connected

- Number of neurons

Demo: http://playground.tensorflow.org/



Architecture

How many of each type of layer?

In what order?

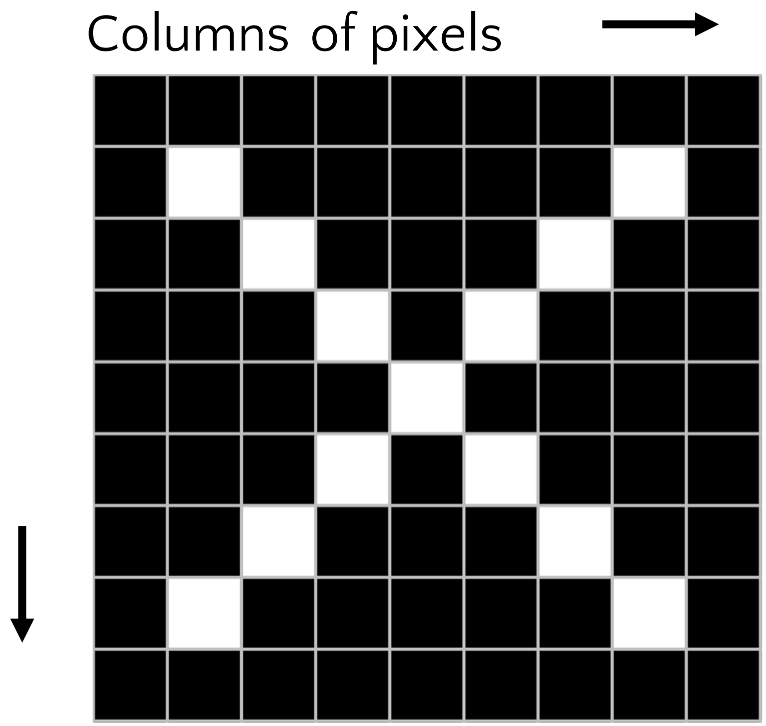
Not just images

Any 2D (or 3D) data.

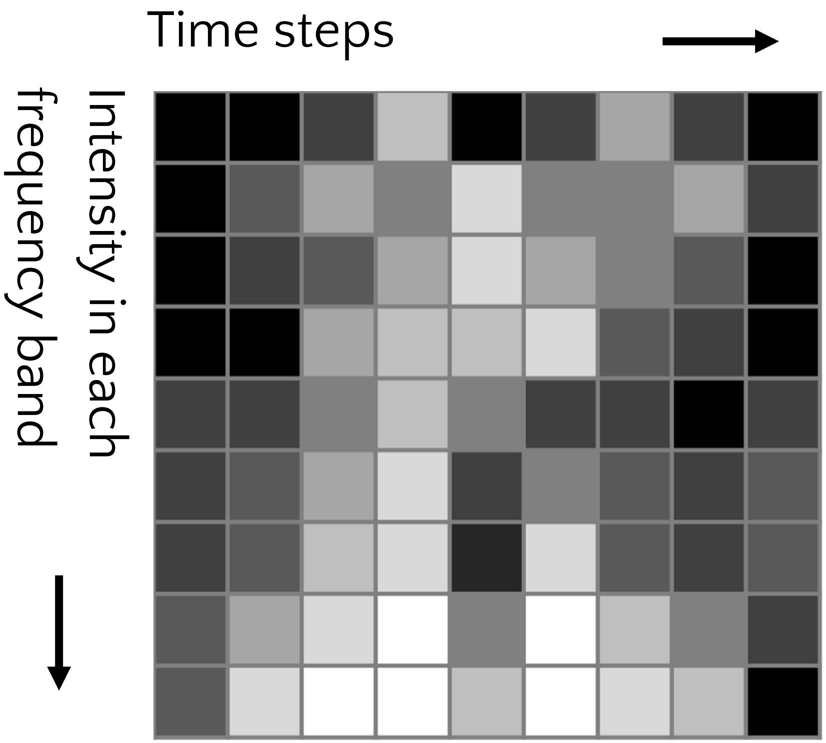
Things closer together are more closely related than things far away.

Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Images

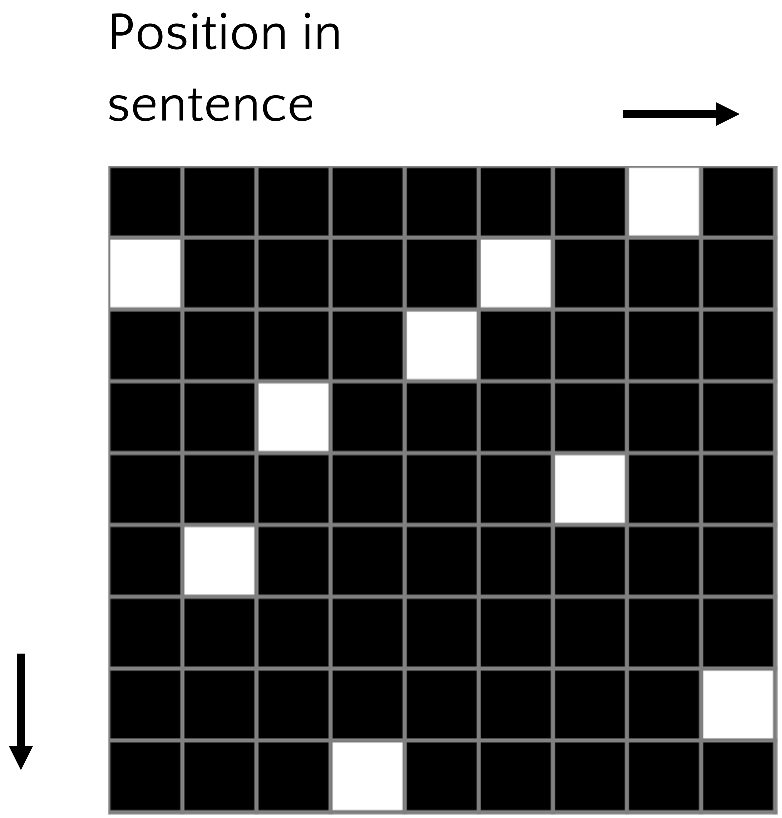


Sound



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Text



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

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.

Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

Customer data

Name, age,
address, email,
purchases,
browsing activity,...

A	22	1A	a@a	1	aa	a1.a	123	aa1
B	33	2B	b@b	2	bb	b2.b	234	bb2
C	44	3C	c@c	3	cc	c3.c	345	cc3
D	55	4D	d@d	4	dd	d4.d	456	dd4
E	66	5E	e@e	5	ee	e5.e	567	ee5
F	77	6F	f@f	6	ff	f6.f	678	ff6
G	88	7G	g@g	7	gg	g7.g	789	gg7
H	99	8H	h@h	8	hh	h8.h	890	hh8
I	111	9I	i@i	9	ii	i9.i	901	ii9

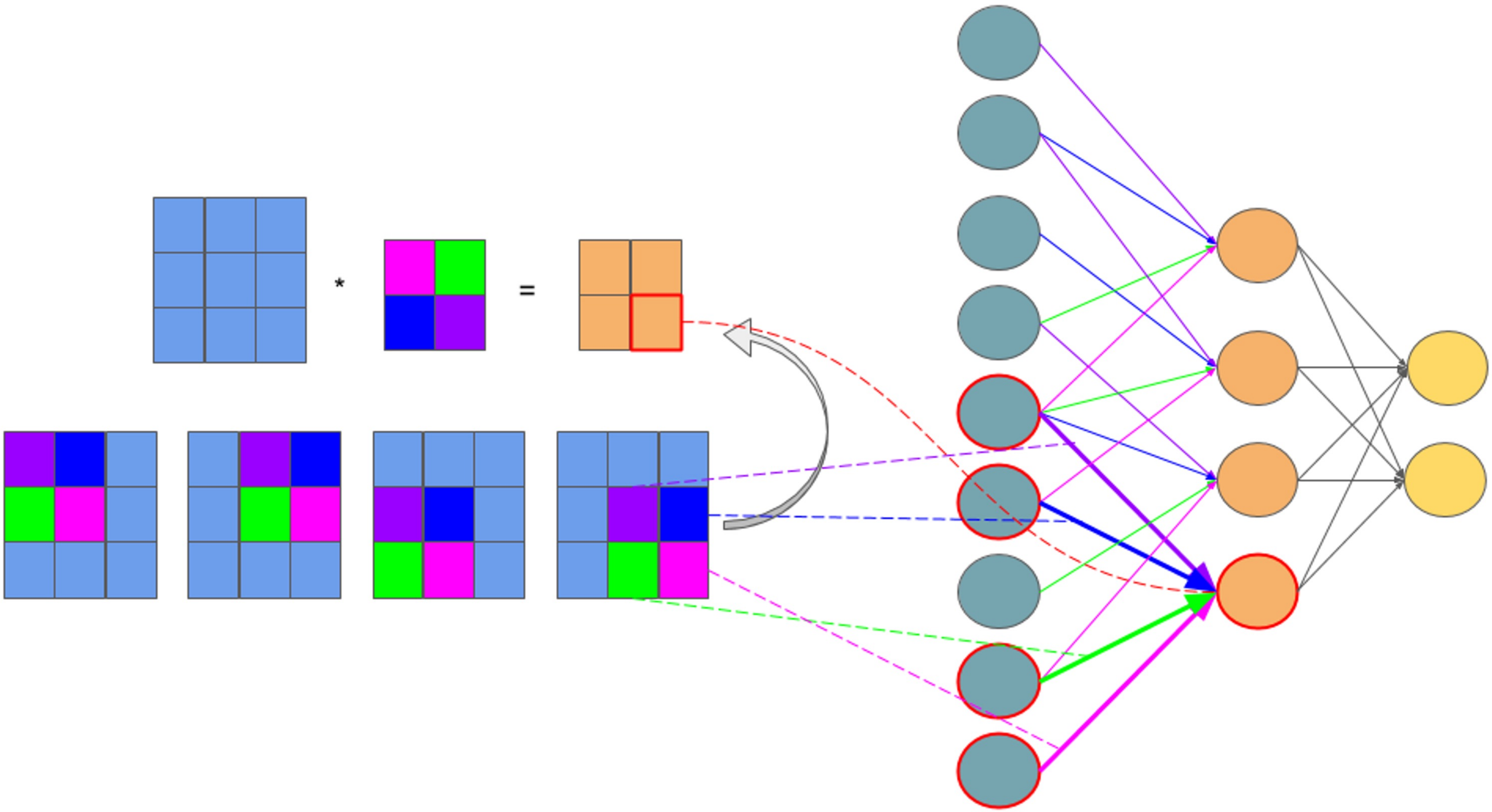
More Details

- Linear Regression
- https://youtu.be/EI6GES6q3wo

- Gradient Descent & Neural Network
- https://youtu.be/nE3FWsBU3Gs

- Multi-layer Neural Network Backpropagation
- https://youtu.be/827Pg9fn_1o

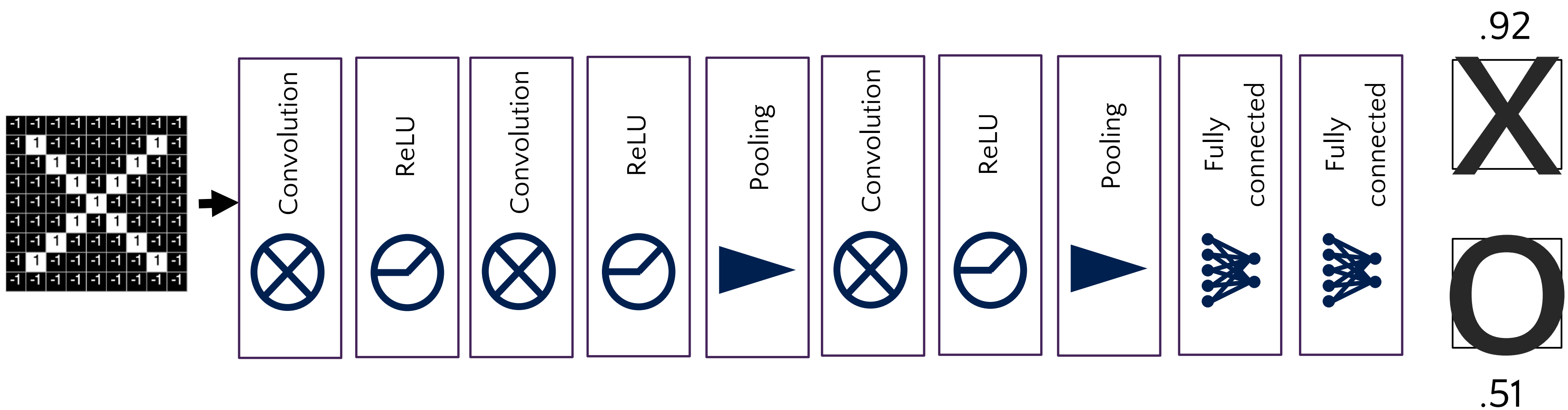
Feedforward in CNN is identical with convolution operation



https://grzegorzgwardys.wordpress.com/2016/04/22/8/

Recap: Putting it all together

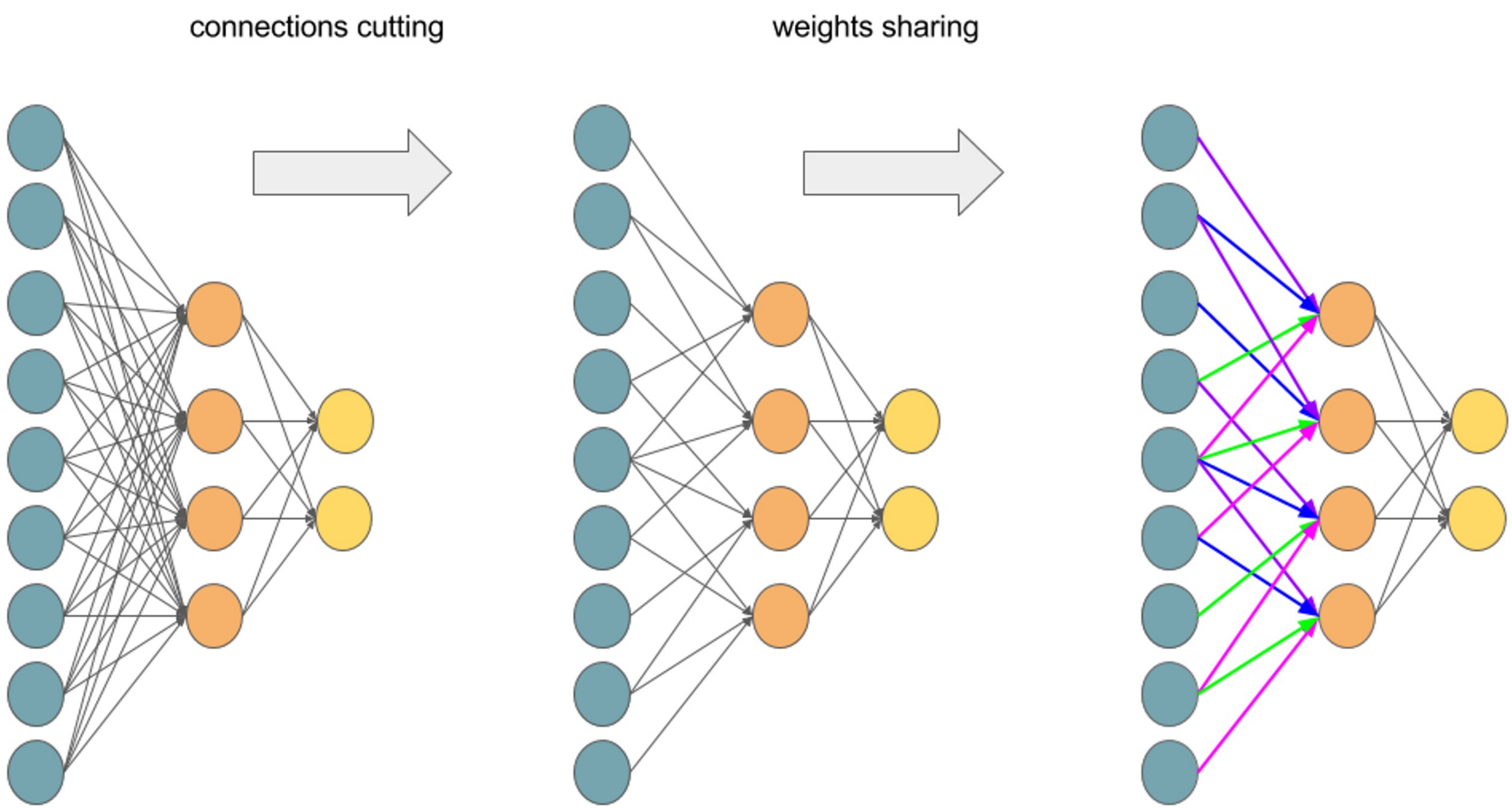
A set of pixels becomes a set of votes.



Brandon Rohrer https://brohrer.github.io/how_convolutional_neural_networks_work.html

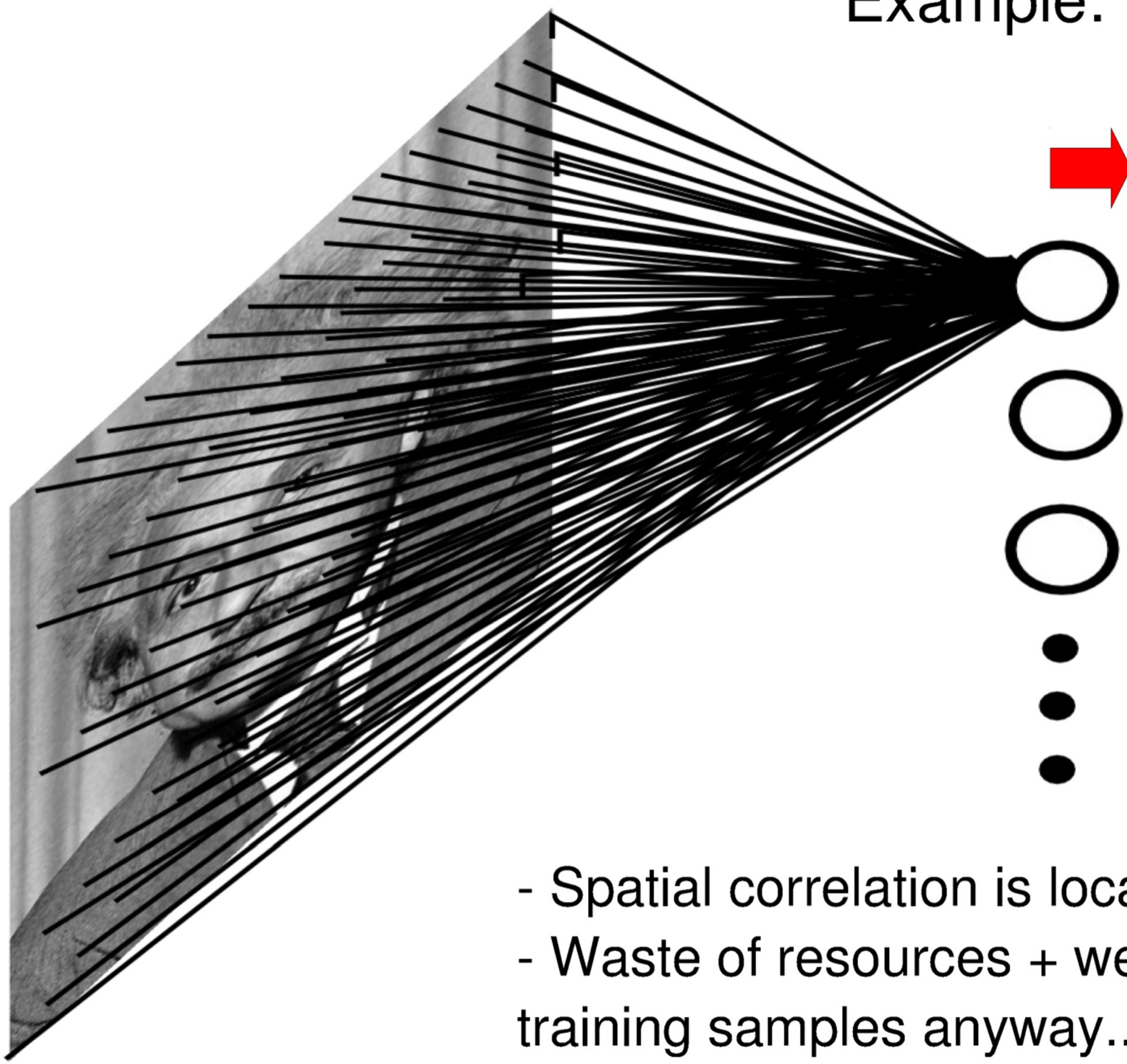
Any Questions?

Transforming Multilayer Perceptron to CNN



https://grzegorzgwardys.wordpress.com/2016/04/22/8/

Fully Connected Layer

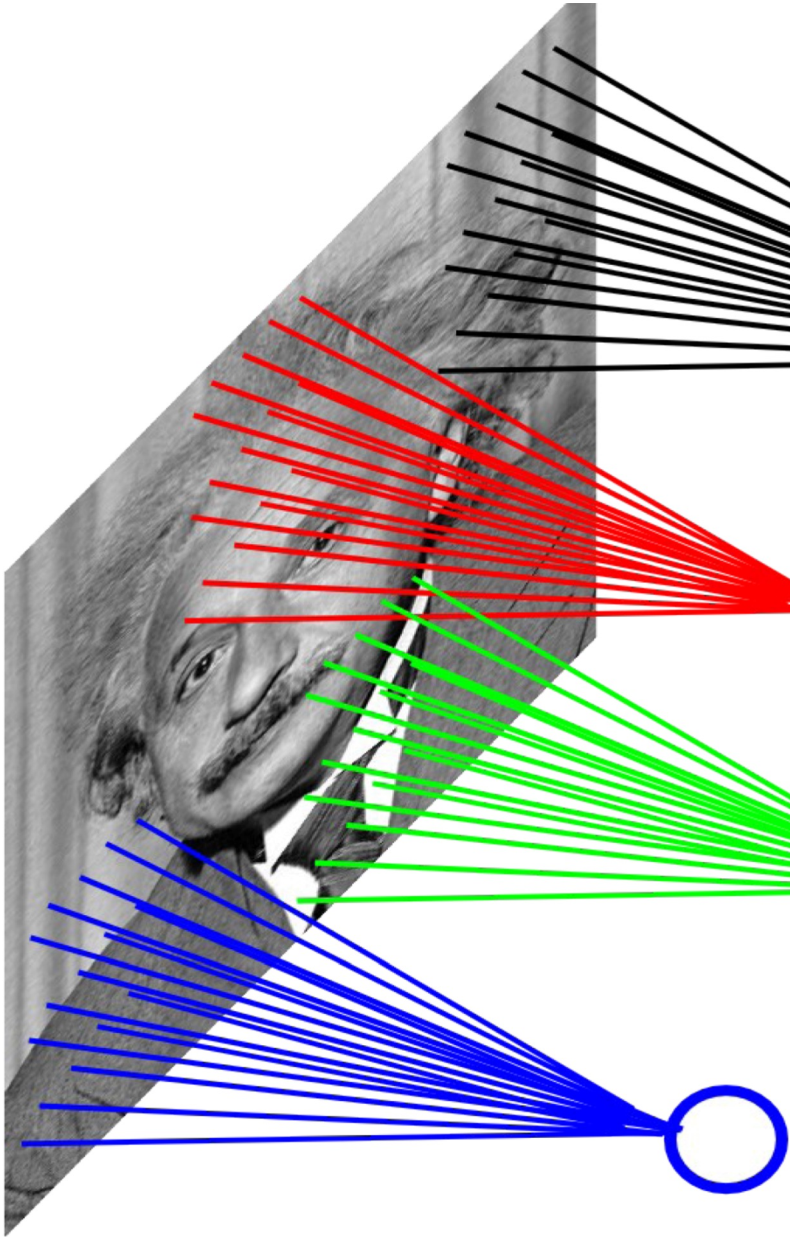


Example: 200x200 image
40K hidden units
➡ **~2B parameters**!!!

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

Ranzato

Locally Connected Layer



Example: 200x200 image
40K hidden units
Filter size: 10x10

Note: This parameterization is good when input image is registered (e.g., face recognition).

Ranzato

Next…More on CNNs

- Volumetric Convolution Filter

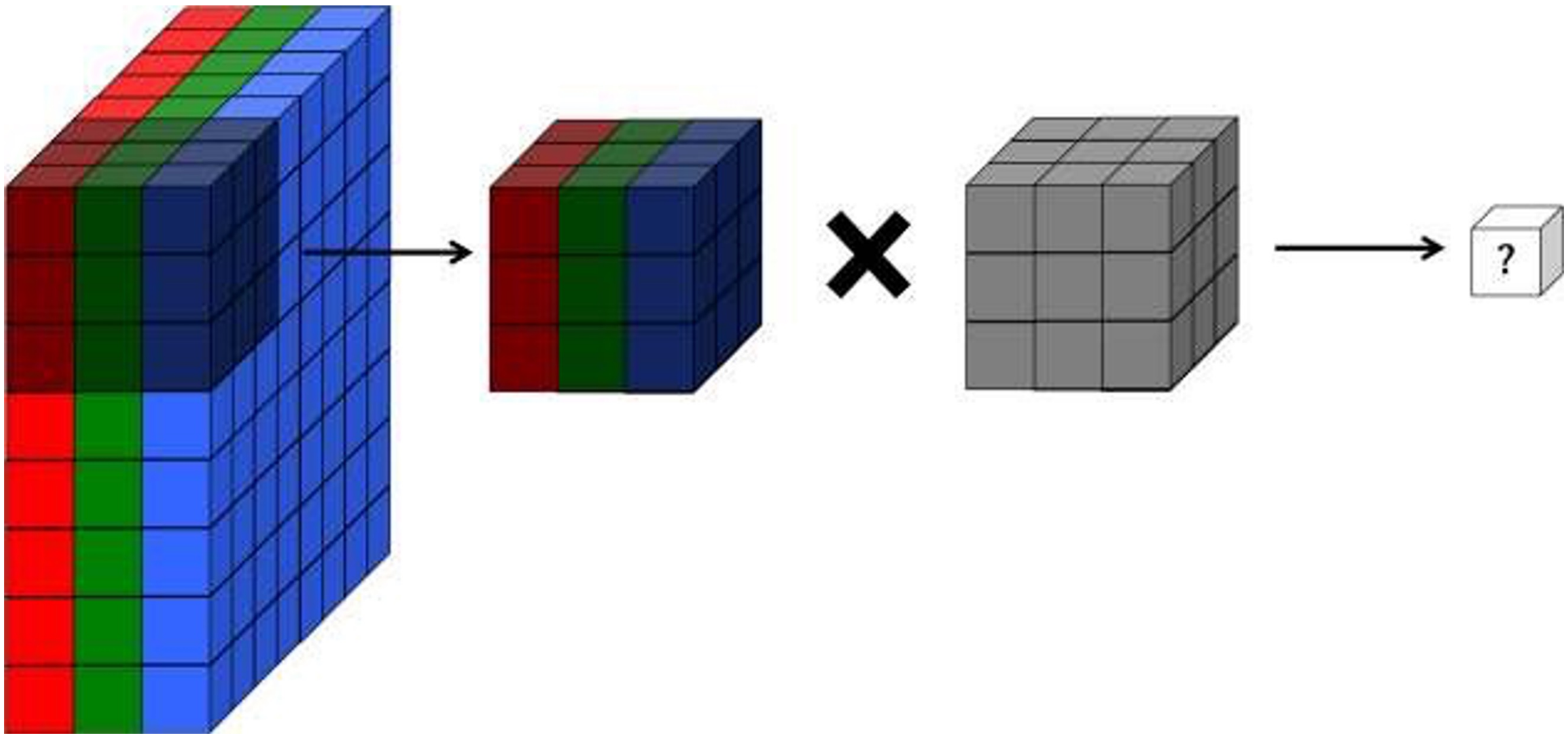


Figure 5-8. Representing a full-color RGB image as a volume and applying a volumetric convolutional filter

Volumetric Convolution Filter

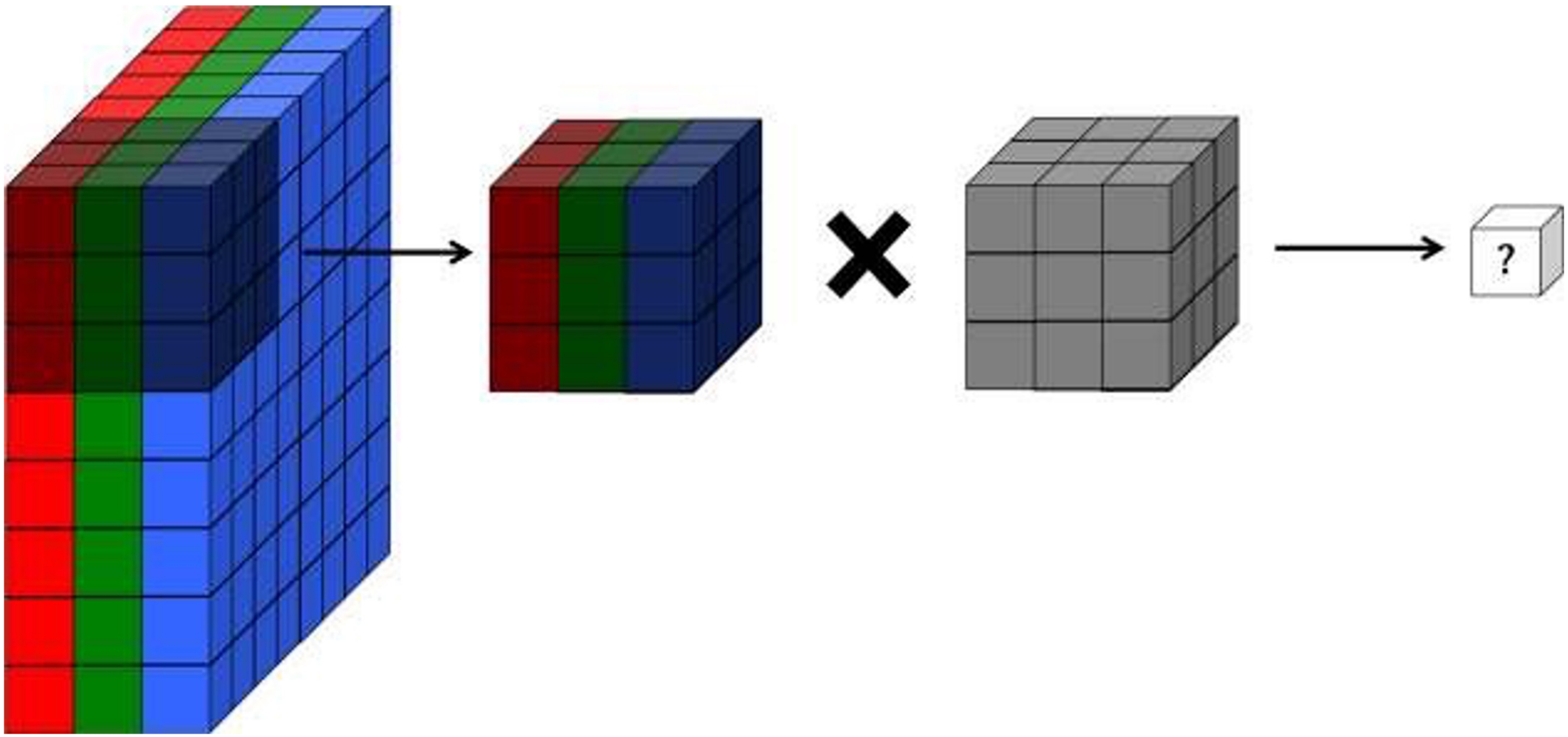


Figure 5-8. Representing a full-color RGB image as a volume and applying a volumetric convolutional filter

Volumetric Convolution Filter

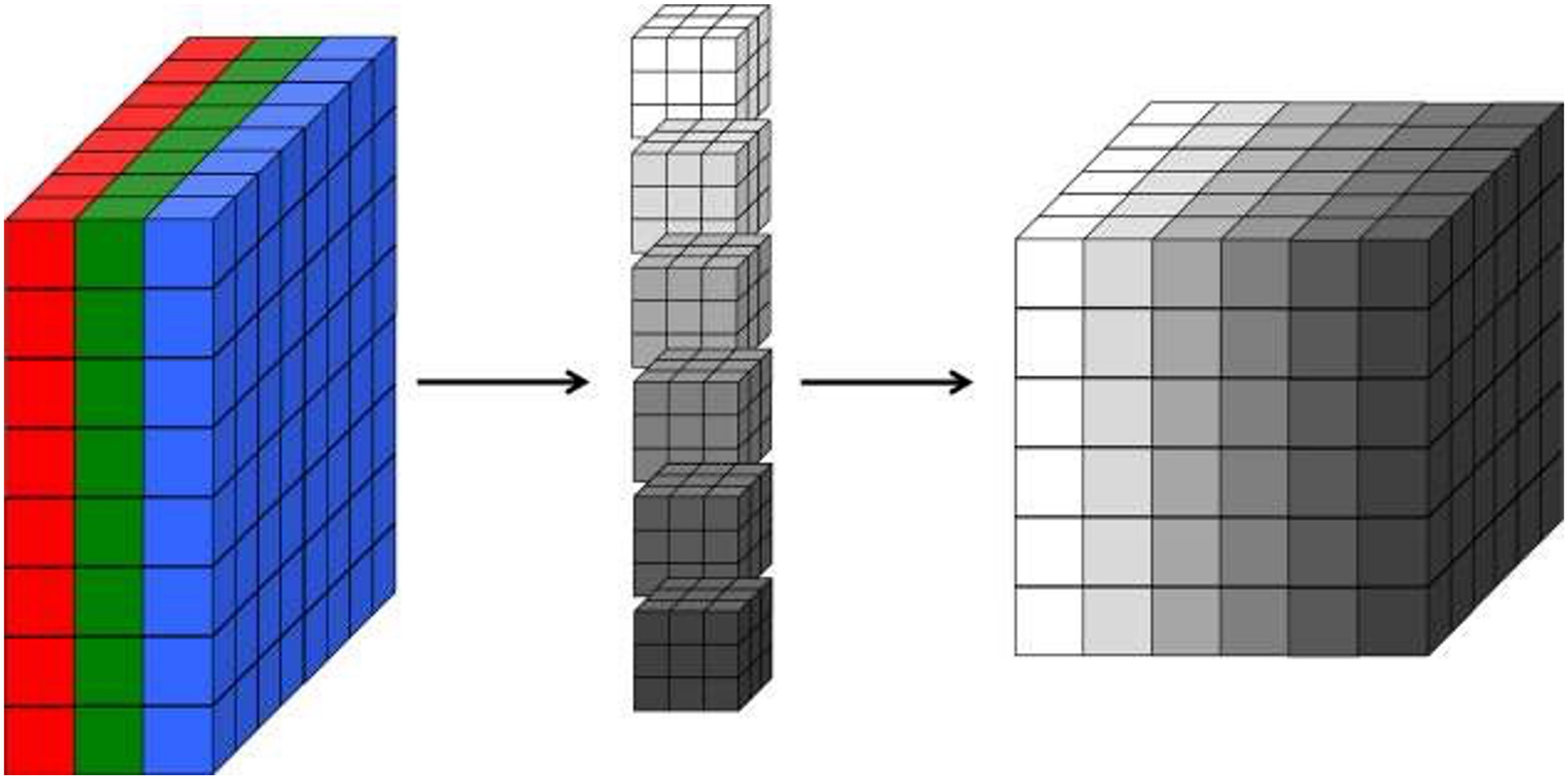
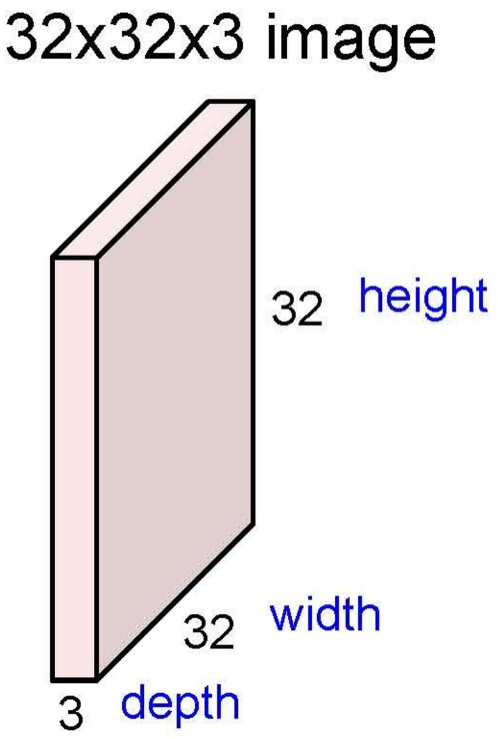
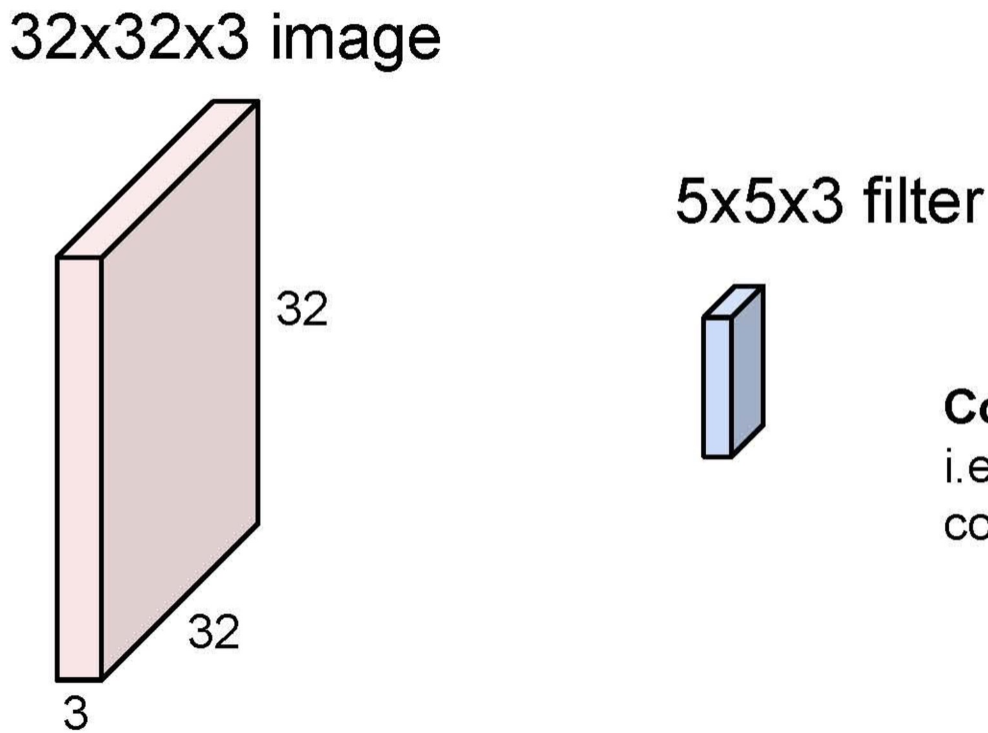


Figure 5-9. A three-dimensional visualization of a convolutional layer, where each filter corresponds to a slice in the resulting output volume

Convolution Layer

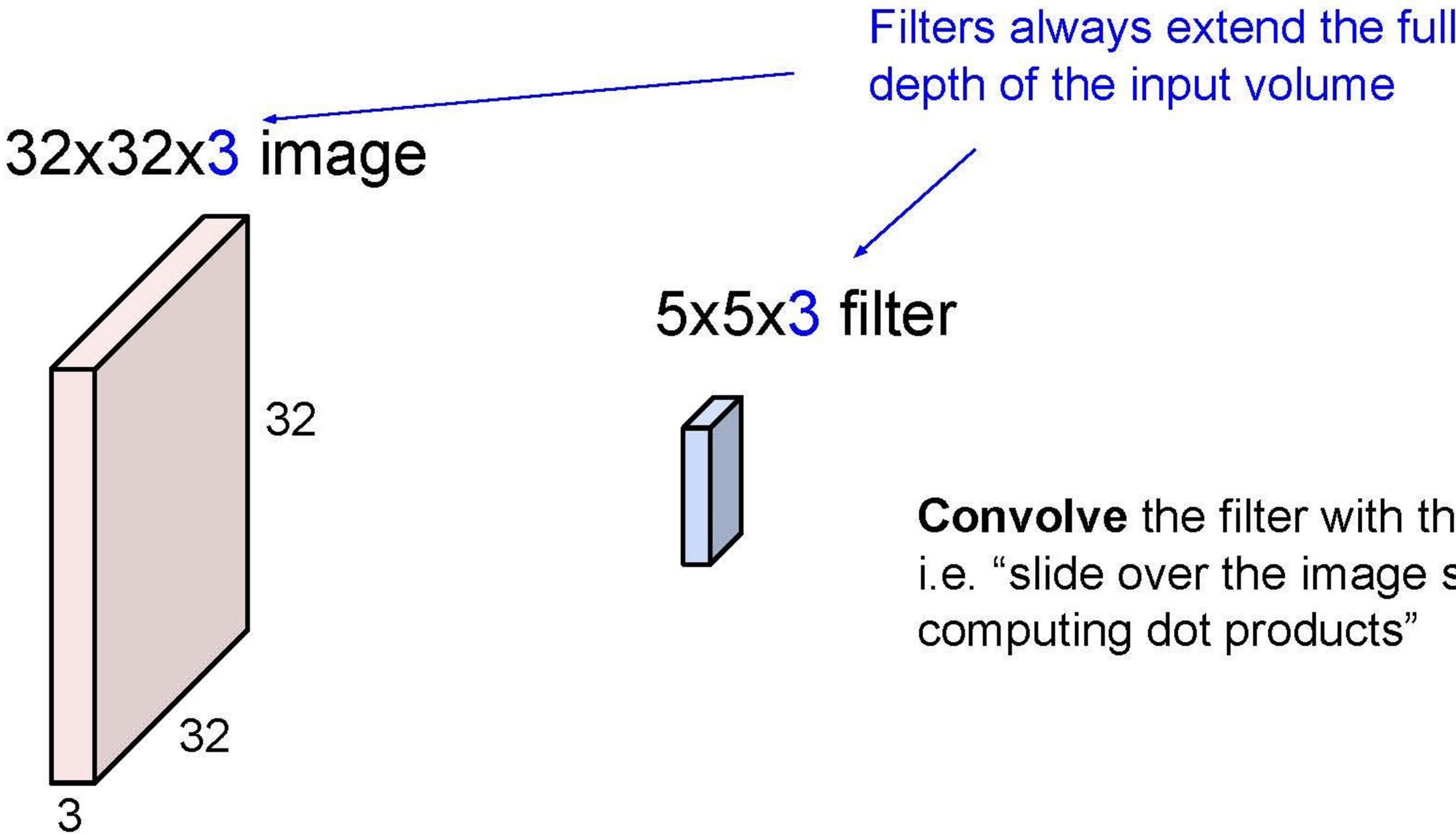


Convolution Layer



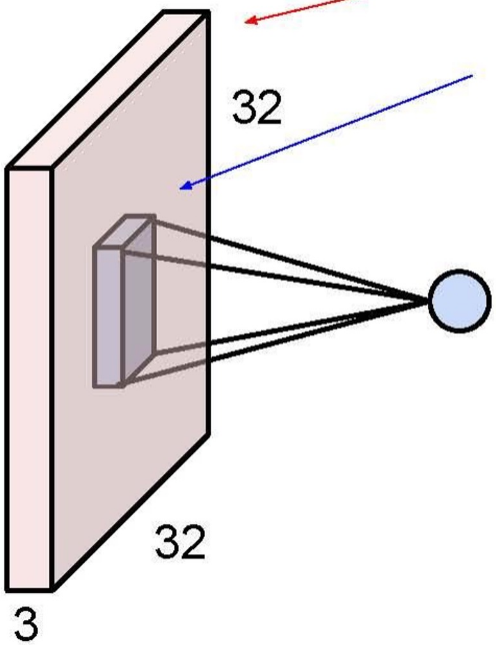
Convolve the filter with the image
i.e. "slide over the image spatially,
computing dot products"

Convolution Layer



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

Convolution Layer



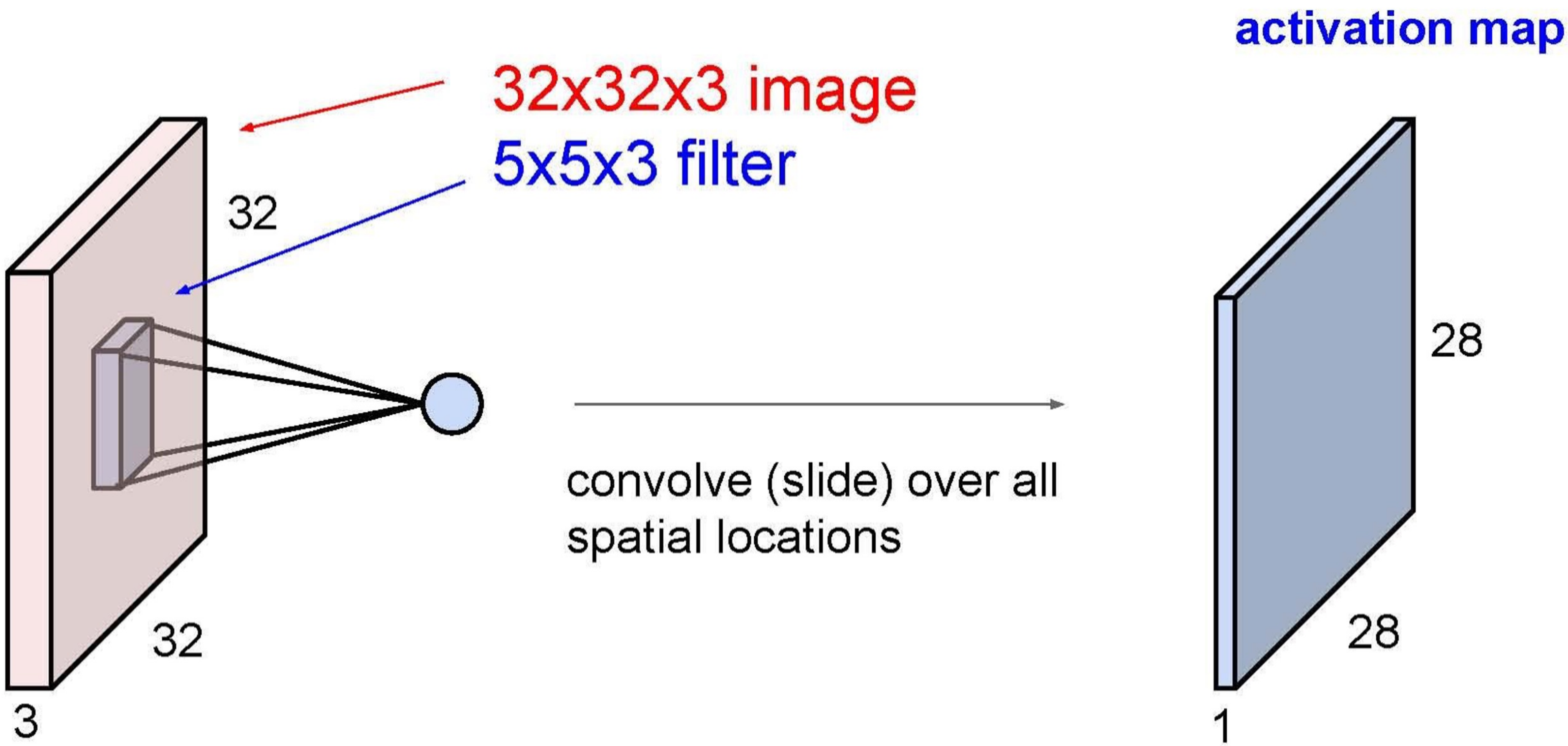
32x32x3 image

5x5x3 filter w

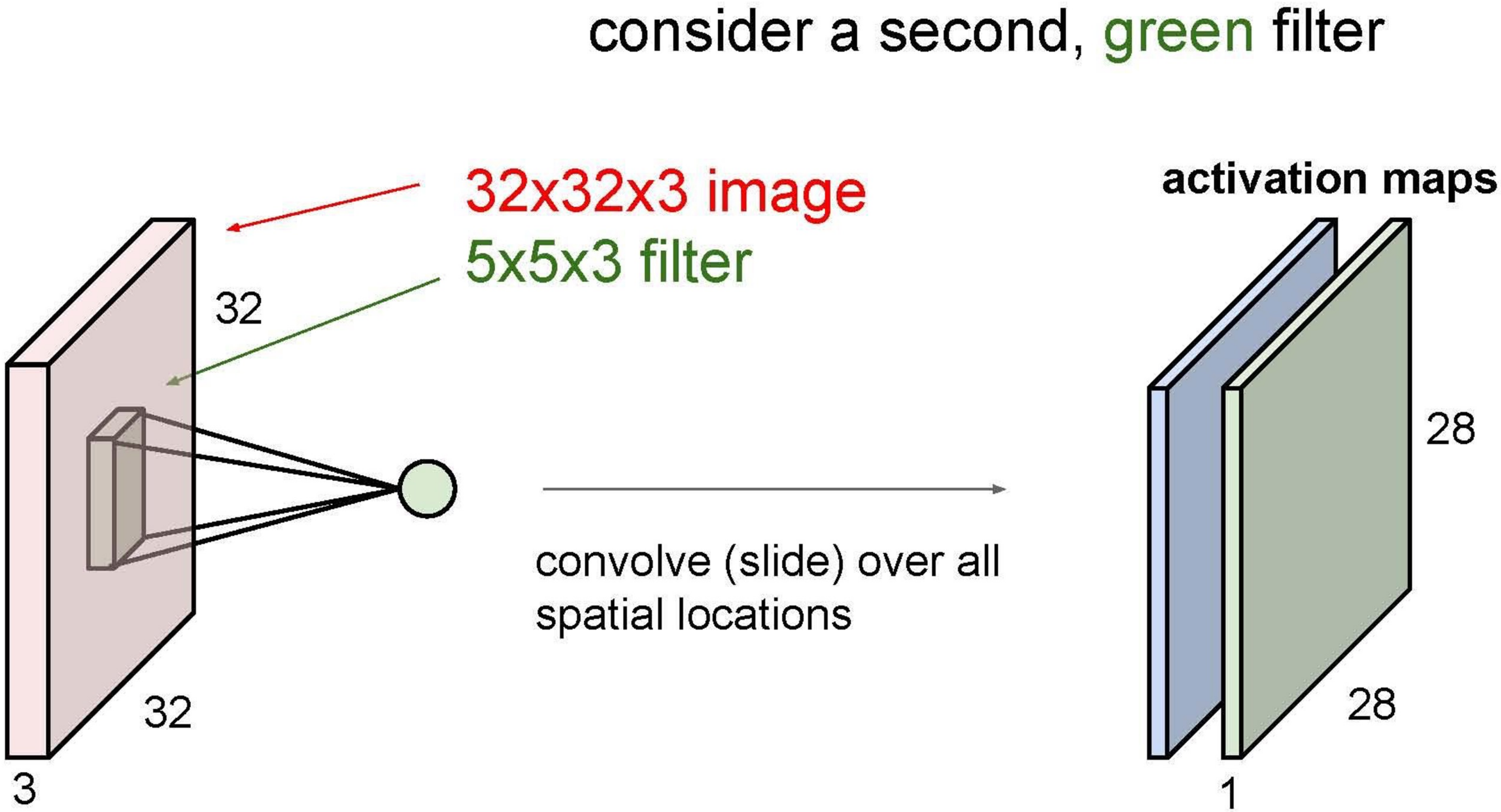
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolution Layer

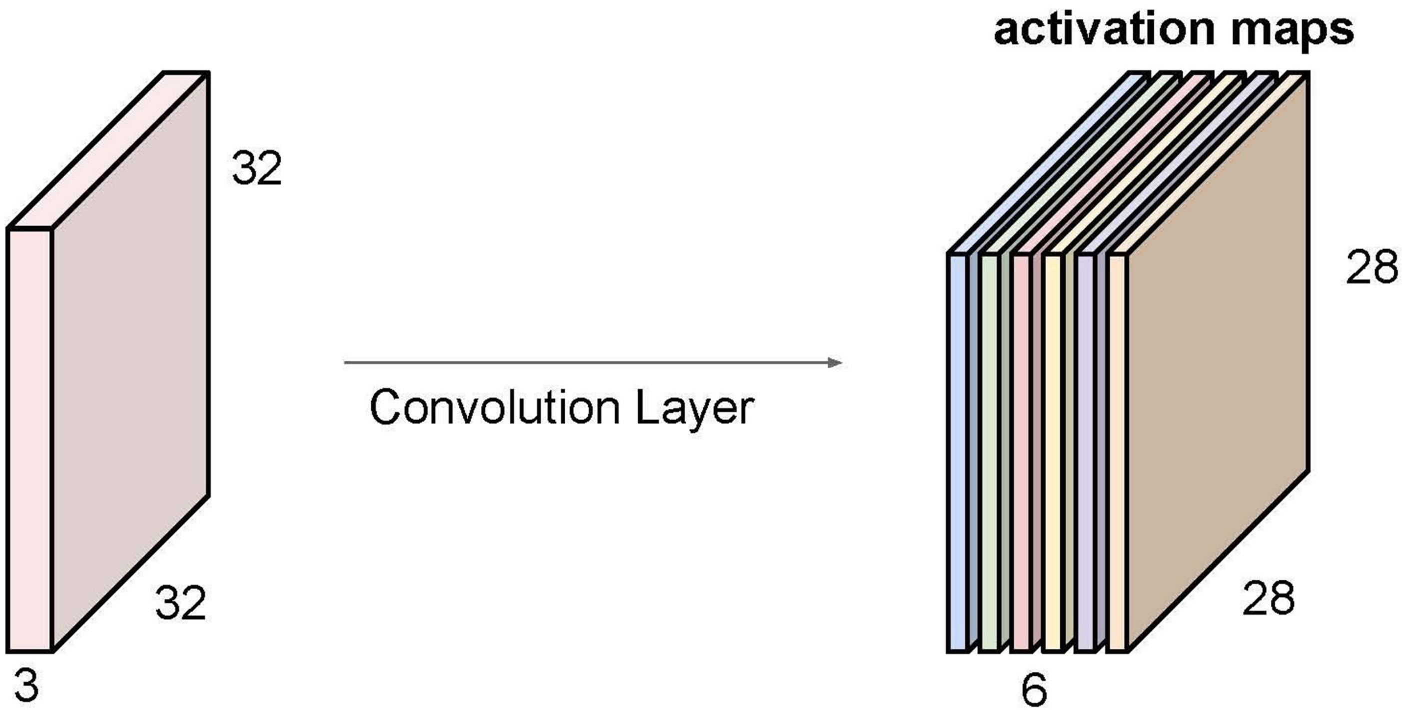


Convolution Layer



Convolution Layer

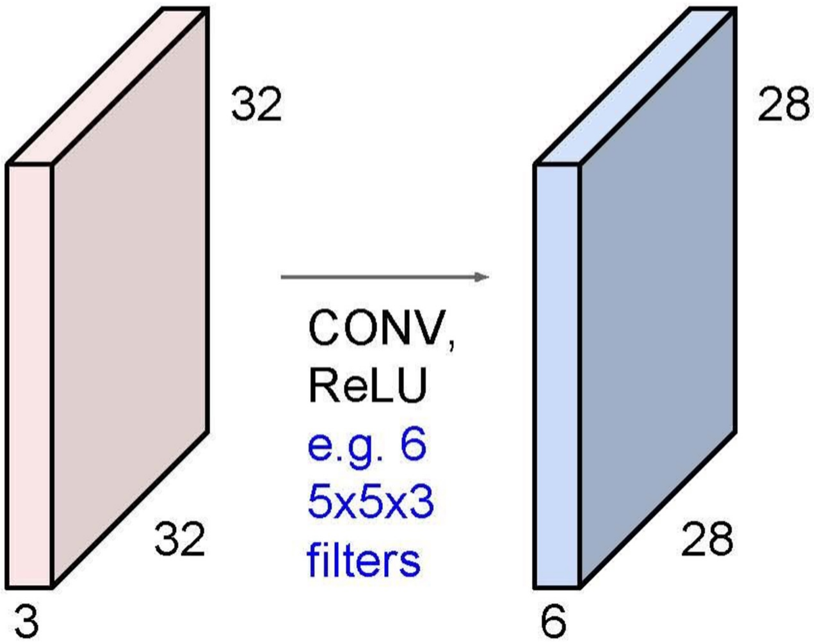
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

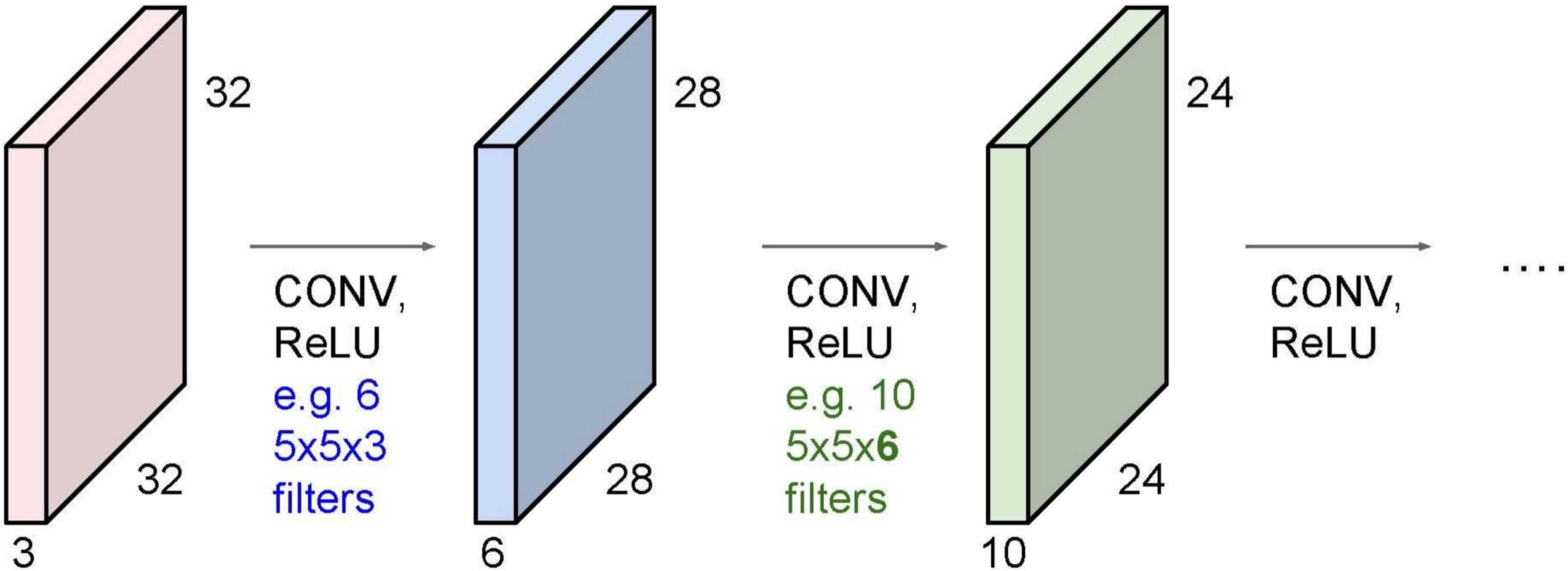
ConvNet

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



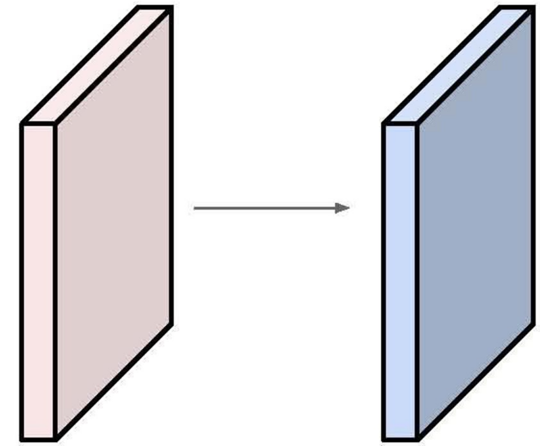
ConvNet

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



ConvNet

Examples time:



Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Output volume size: ?

???

32x32x10

Mirror Padding 2 for Filter Size 5x5

Image

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

Filter

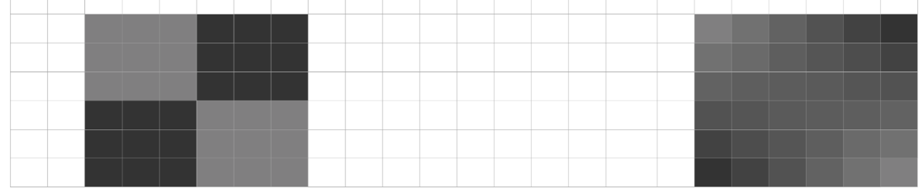
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

x1/25

Ignoring flip

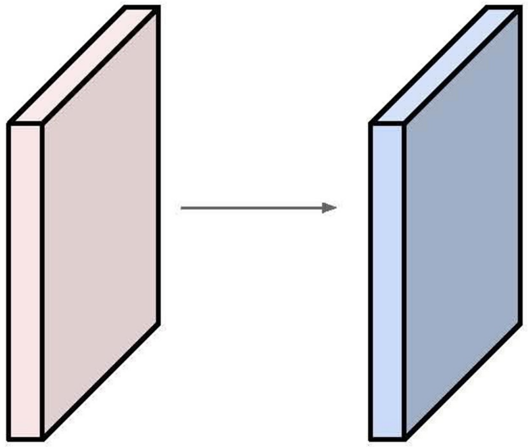
Conv Output

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



ConvNet

Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride **2**, pad 2

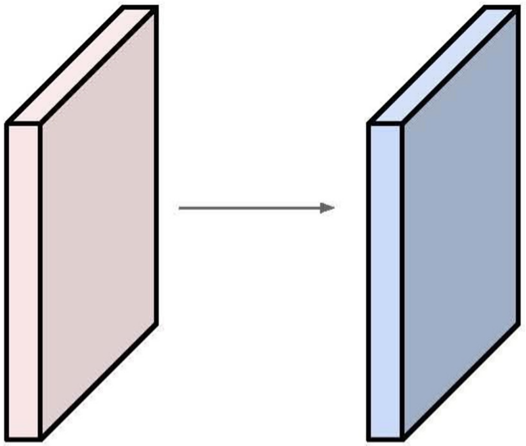
Output volume size: ?

???

16x16x10

ConvNet

Examples time:



Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Output volume size:

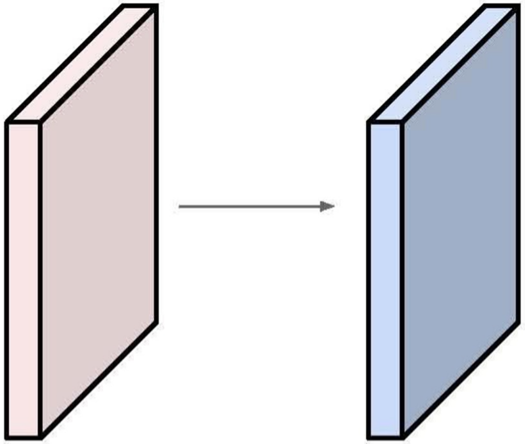
(32+2*2-5)/1+1 = 32 spatially, so

32x32x10

$$(W+2xP-F)/S + 1$$

ConvNet

Examples time:



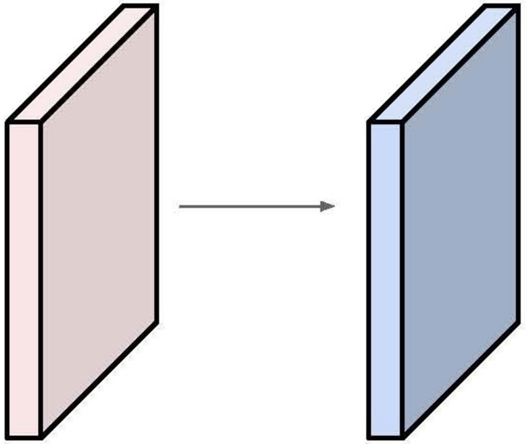
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

???

ConvNet

Examples time:



Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

each filter has 5*5*3 + 1 = 76 params (+1 for bias)

=> 76*10 = **760**

$$(F \times F \times 3 + 1) \times K$$

Summary of Conv Layer

- Input: $W_1 \times H_1 \times D_1$
- Output: $W_2 \times H_2 \times D_2$
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$
  - $D_2 = K$
- Parameters
  - No. of Filters: K
  - Their spatial extent: F
  - Stride: S
  - Padding: P
- Parameters (with sharing)
  - Weights per filter $F \times F \times D_1$
  - For K filters and K biases $(F \times F \times D_1 + 1)K$

- Common Settings:
  - K = power of 2
  - e.g. 32, 64, 128, 512
  - $F = 3, S = 1, P = 1$
  - $F = 5, S = 1, P = 2$
  - $F = 5, S = 2, P =$? *(whatever fits)*
  - $F = 1, S = 1, P = 0$

ConvNet

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

ConvNet

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Common settings:

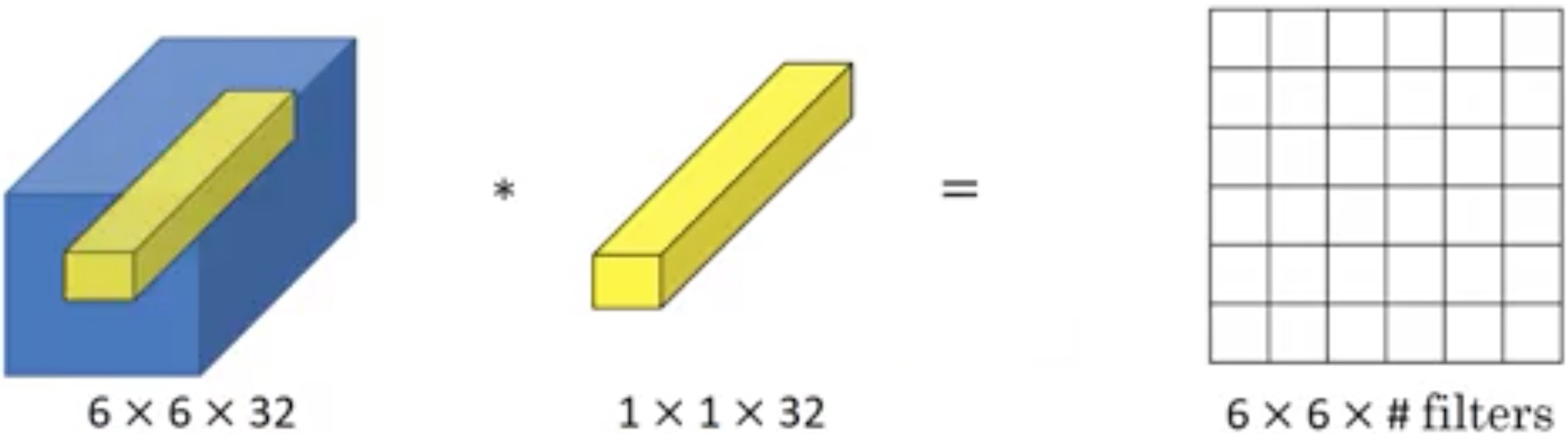
K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

1x1 Convolution

Table 1

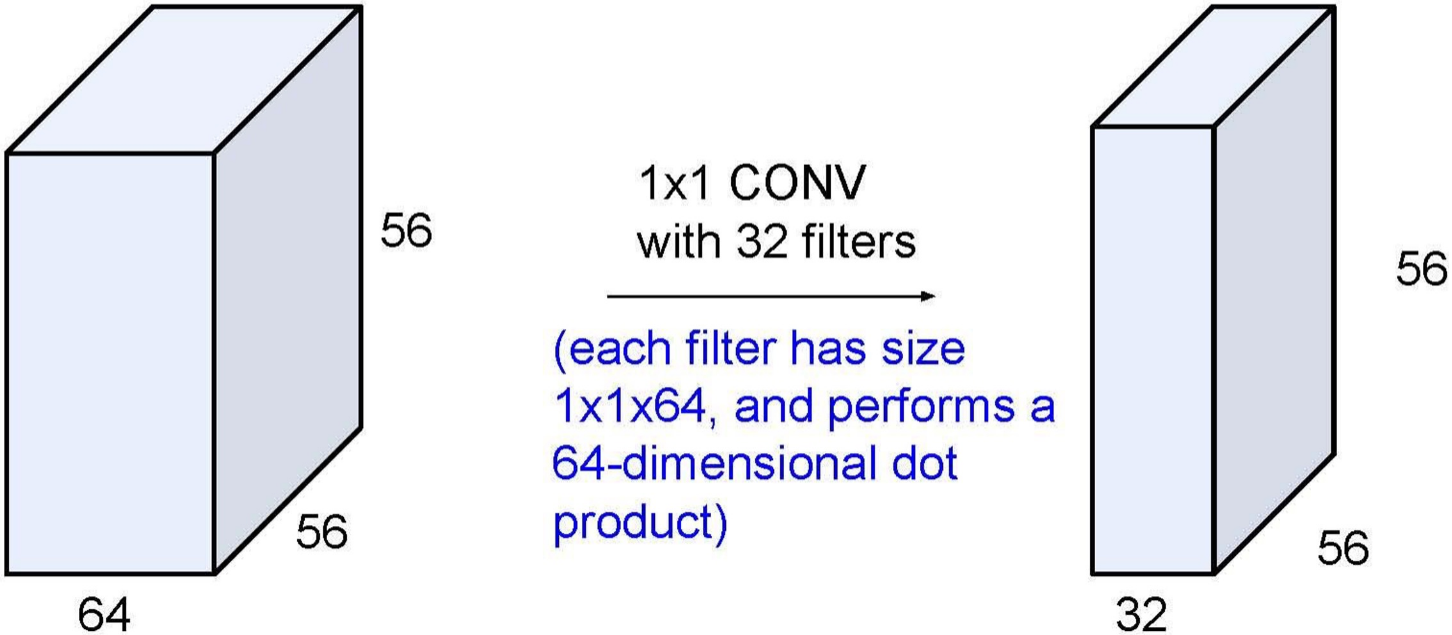
	Image							Filter				Conv Output						
	9	1	6	2	3	0			2			18	2	12	4	6	0	
	9	6	4	2	2	2						18	12	8	4	4	4	
	0	2	7	3	4	9						0	4	14	6	8	18	
	1	5	1	3	6	1						2	10	2	6	12	2	
	8	4	5	9	5	9						16	8	10	18	10	18	
	2	2	2	7	9	5						4	4	4	14	18	10	

1x1 Convolution

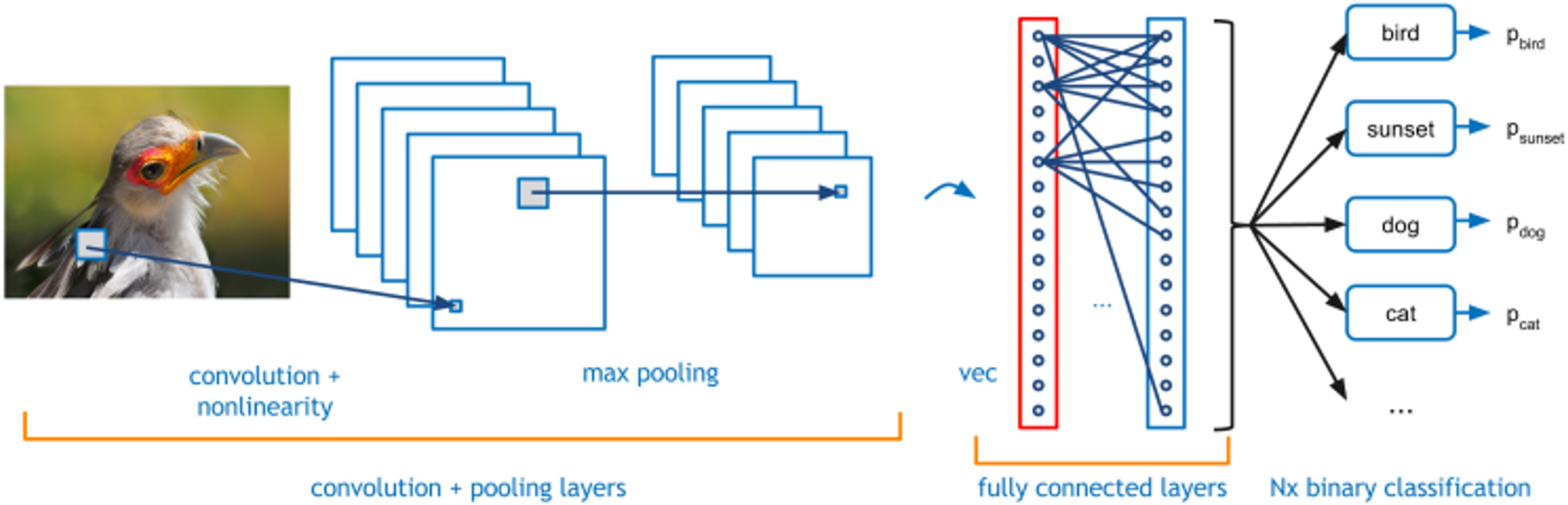


ConvNet

(btw, 1x1 convolution layers make perfect sense)

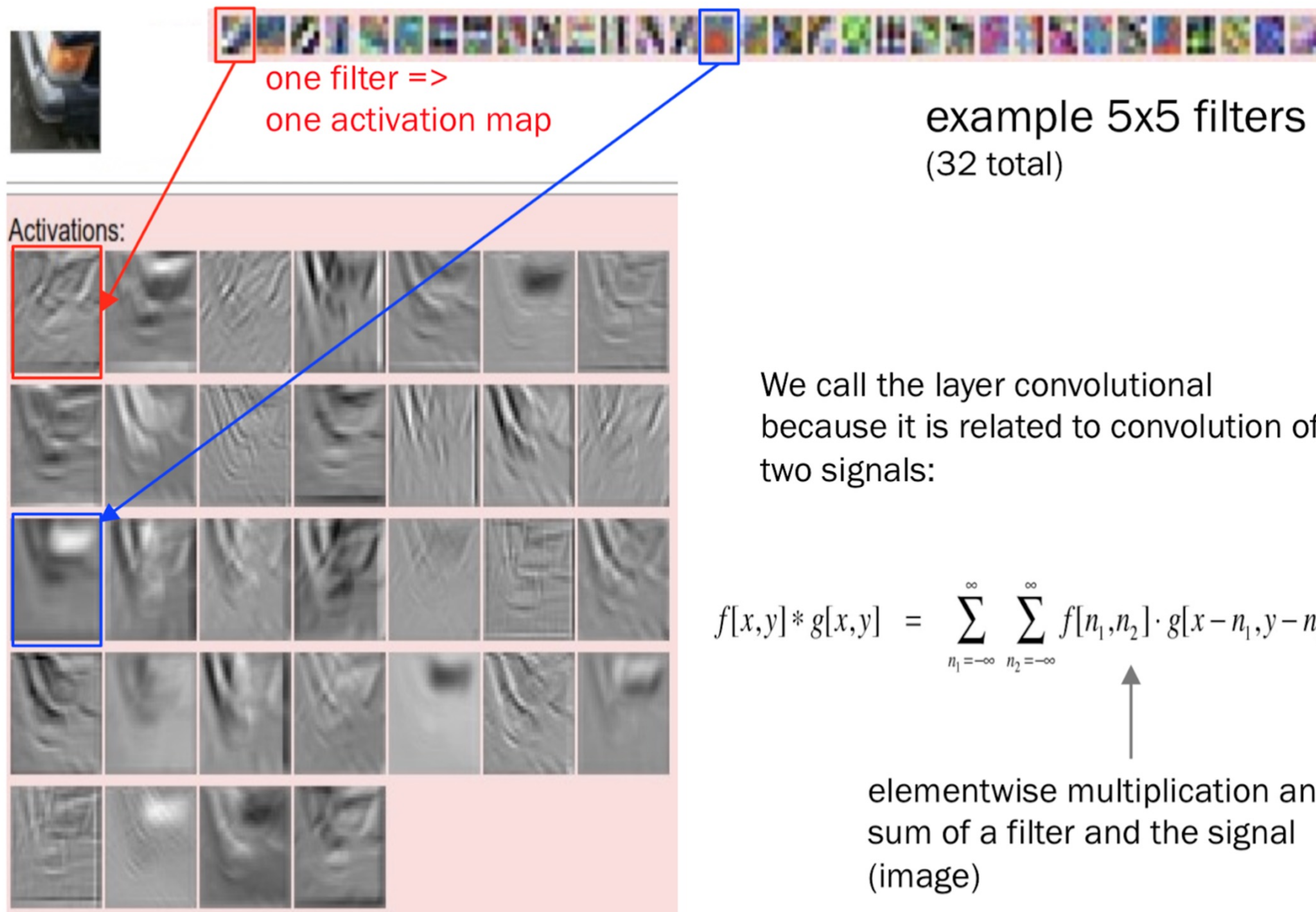


Method current used



http://code.flickr.net/2014/10/20/introducing-flickr-park-or-bird/

https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html



We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

Fei-Fei Li & Andrej Karpathy & Justin Johnson

Any Questions?