

Convolutional Neural Networks (CNNs)

Kristie Wirth

Jon Courtney

Frank Burkholder

Original content from:

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

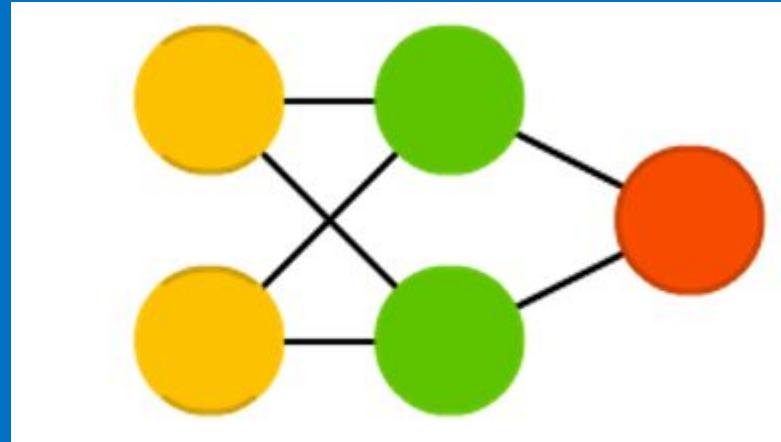
<http://cs231n.github.io/convolutional-networks/>

Learning Objectives

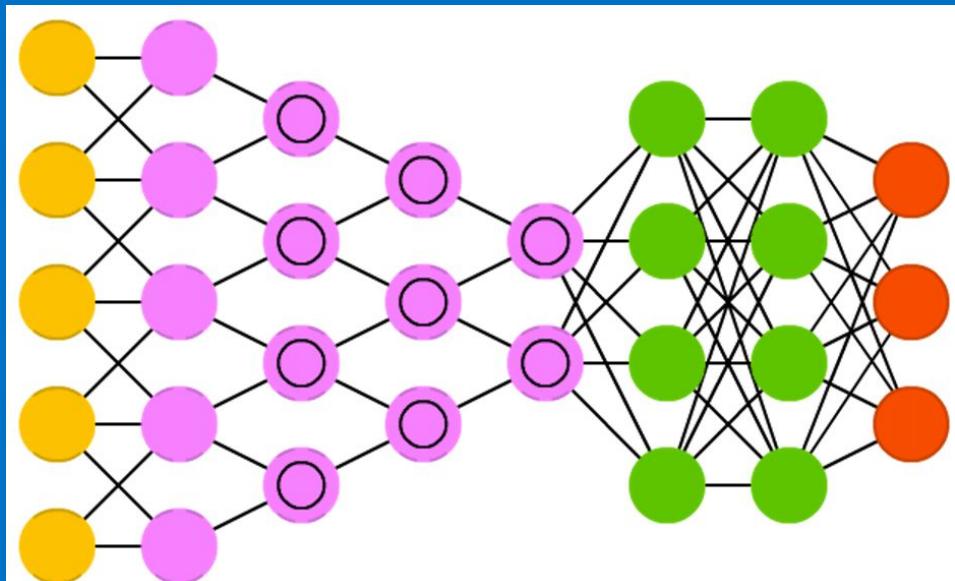
- Identify the layers in CNNs.
- Describe the process of convolution.
- Understand how the process of pooling works.

- Backfed Input Cell
- Input Cell
- △ Noisy Input Cell
- Hidden Cell
- Probabilistic Hidden Cell
- △ Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- △ Different Memory Cell
- Kernel
- Convolution or Pool

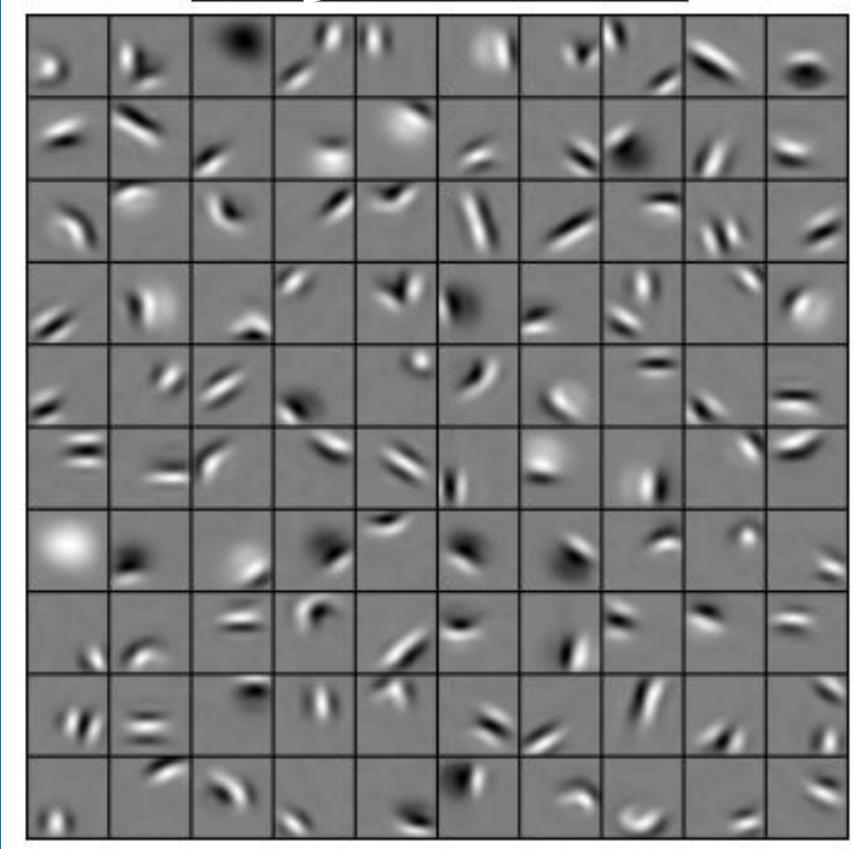
Previously: Multilayer Perceptrons (MLP)



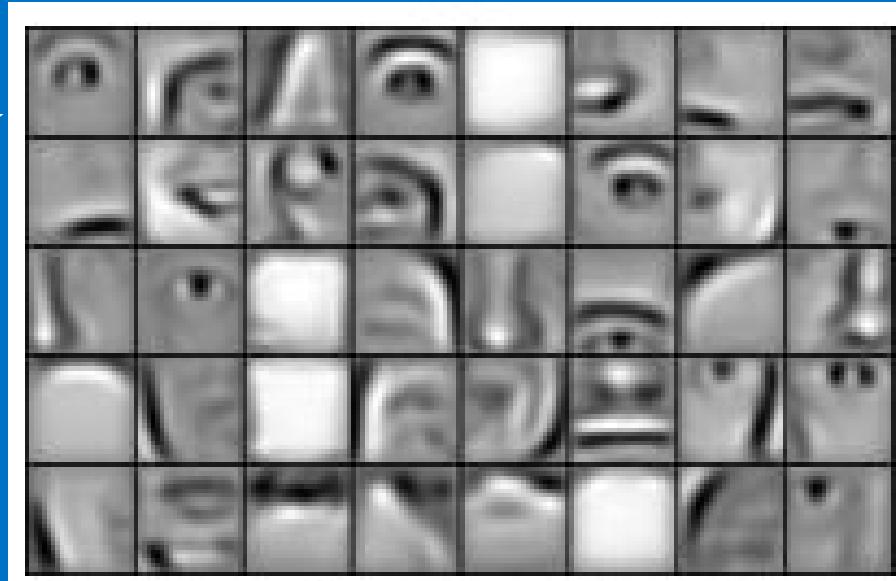
Now: Convolutional Neural Networks (CNN)



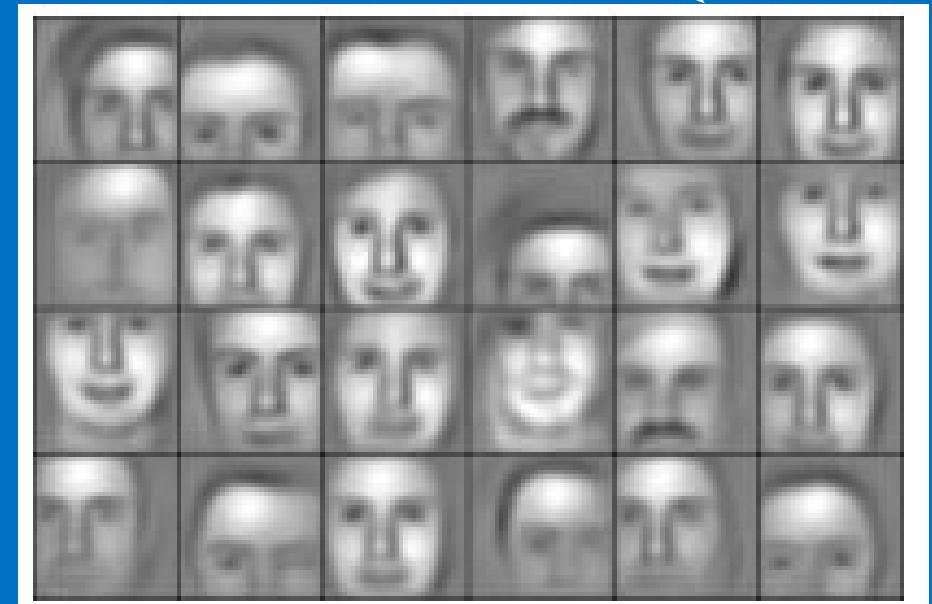
Original (larger) chart comparing structures of many more neural networks:
<http://www.asimovinstitute.org/neural-network-zoo/>



First Layer After Activation



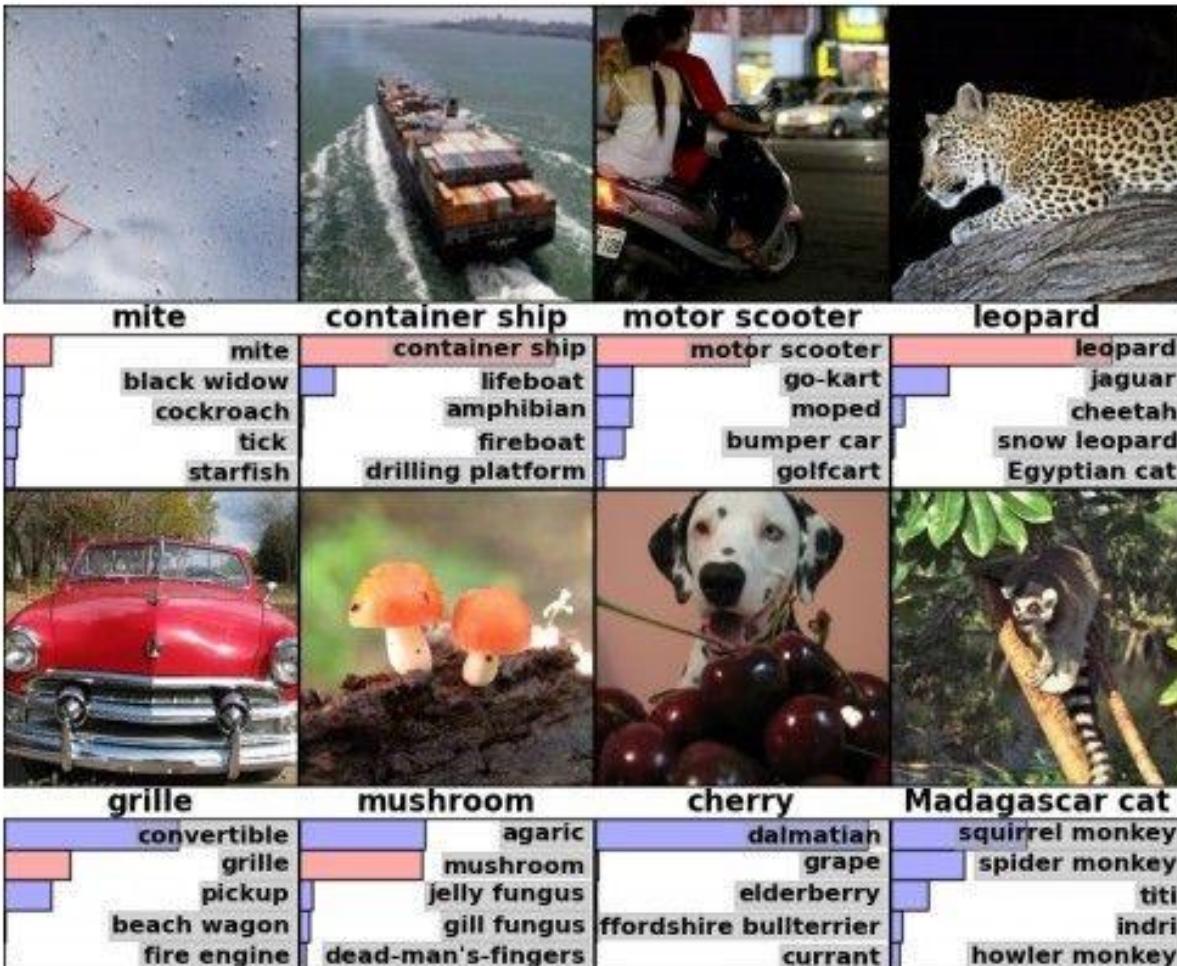
Second Layer After Activation



Third Layer After Activation

Functions of CNNs

Classification

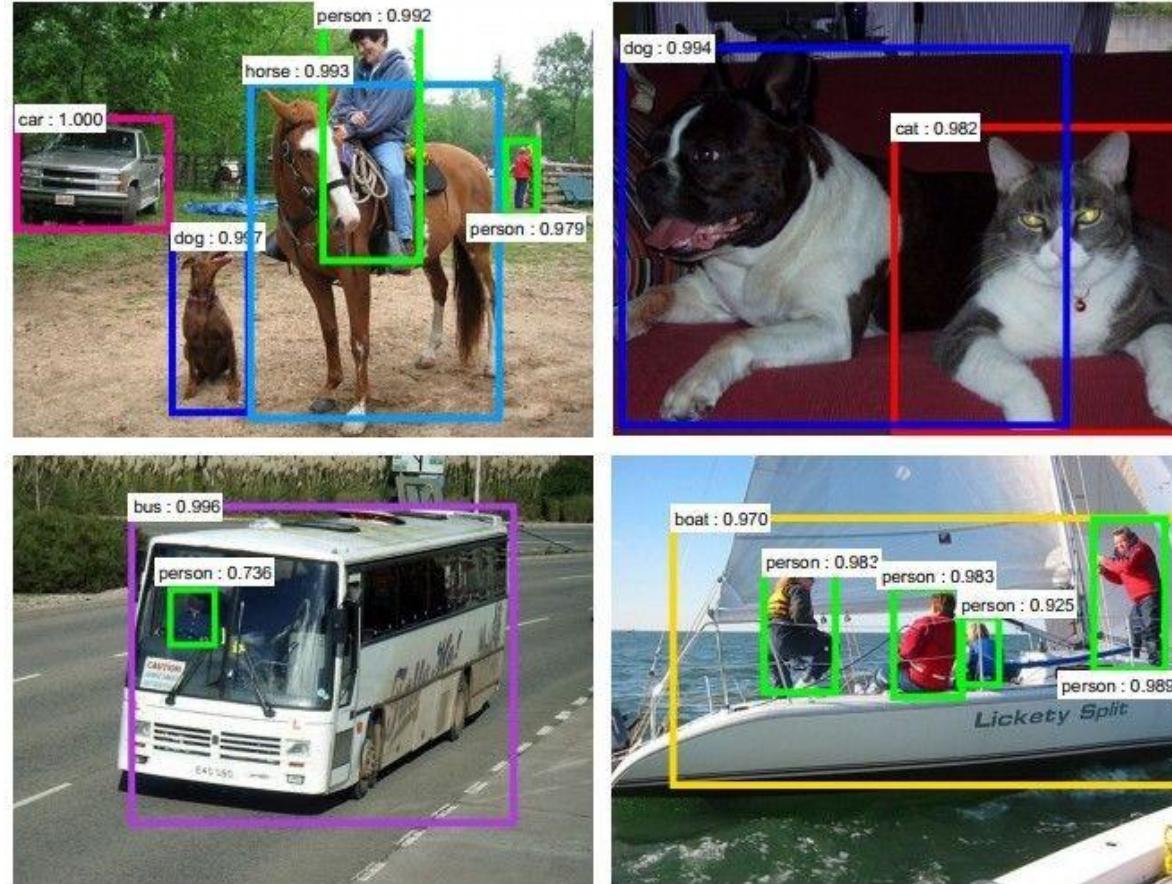


Retrieval



Functions of CNNs

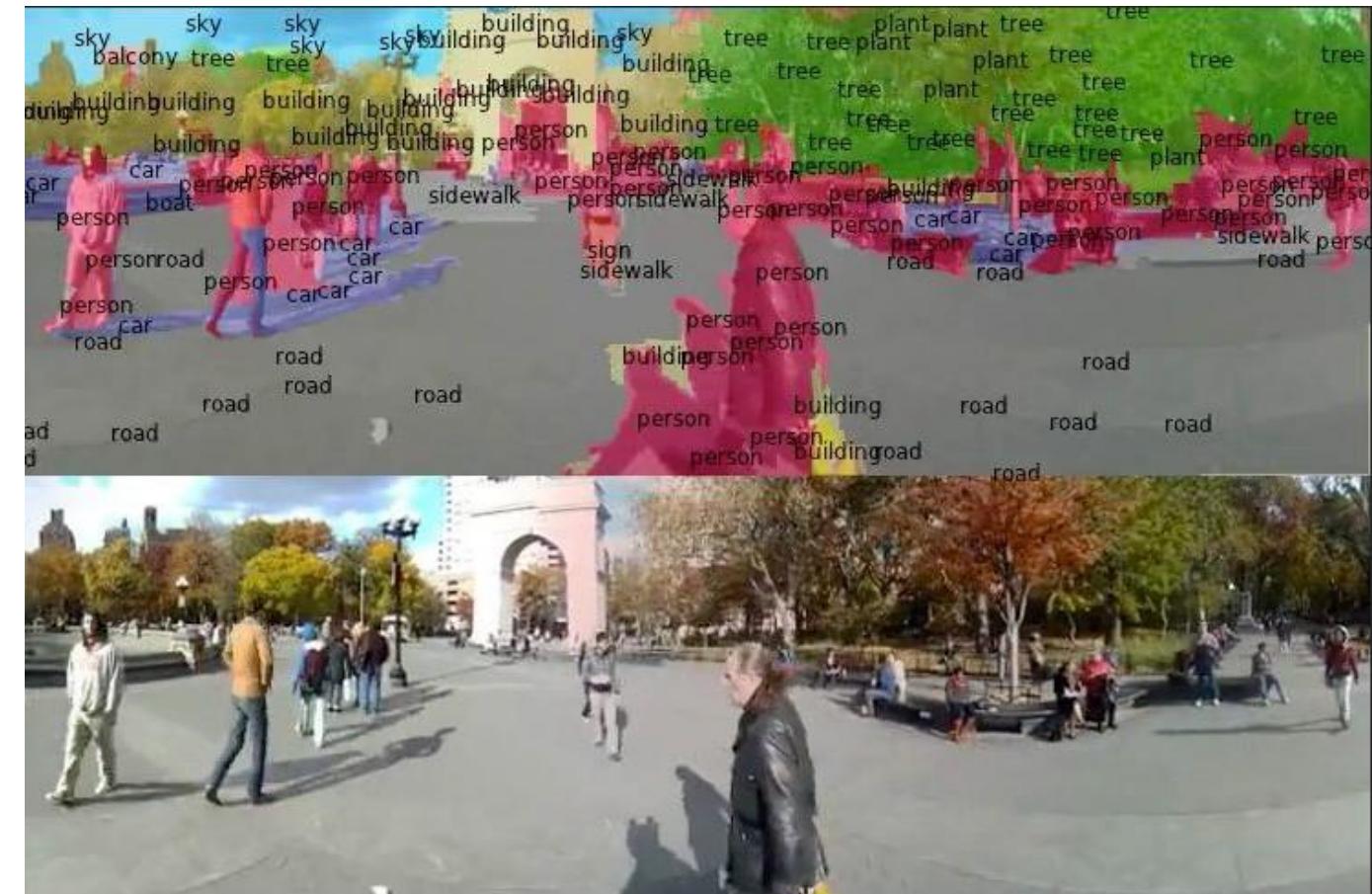
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



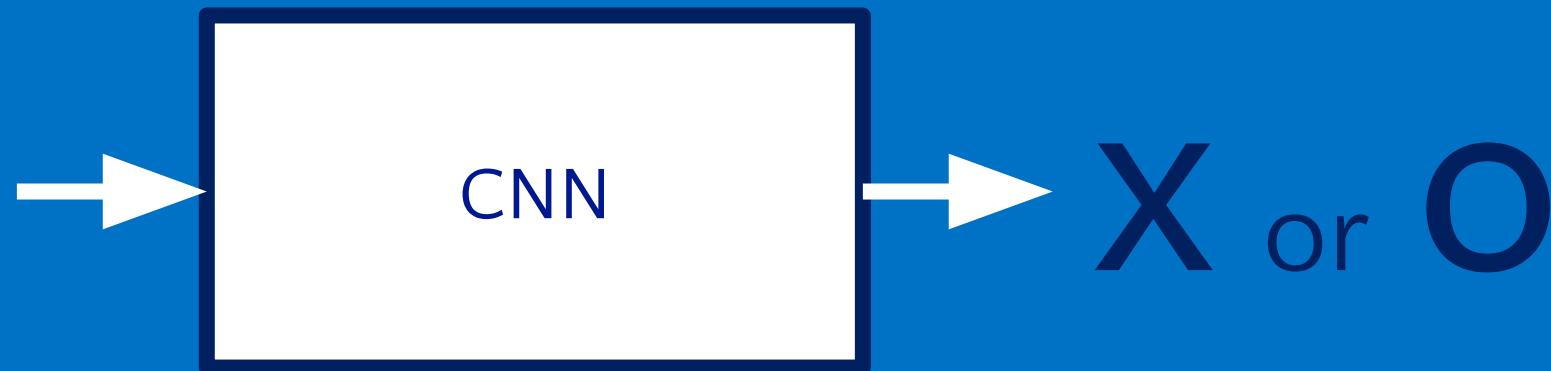
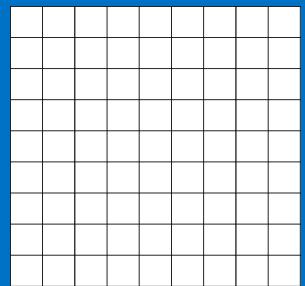
Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

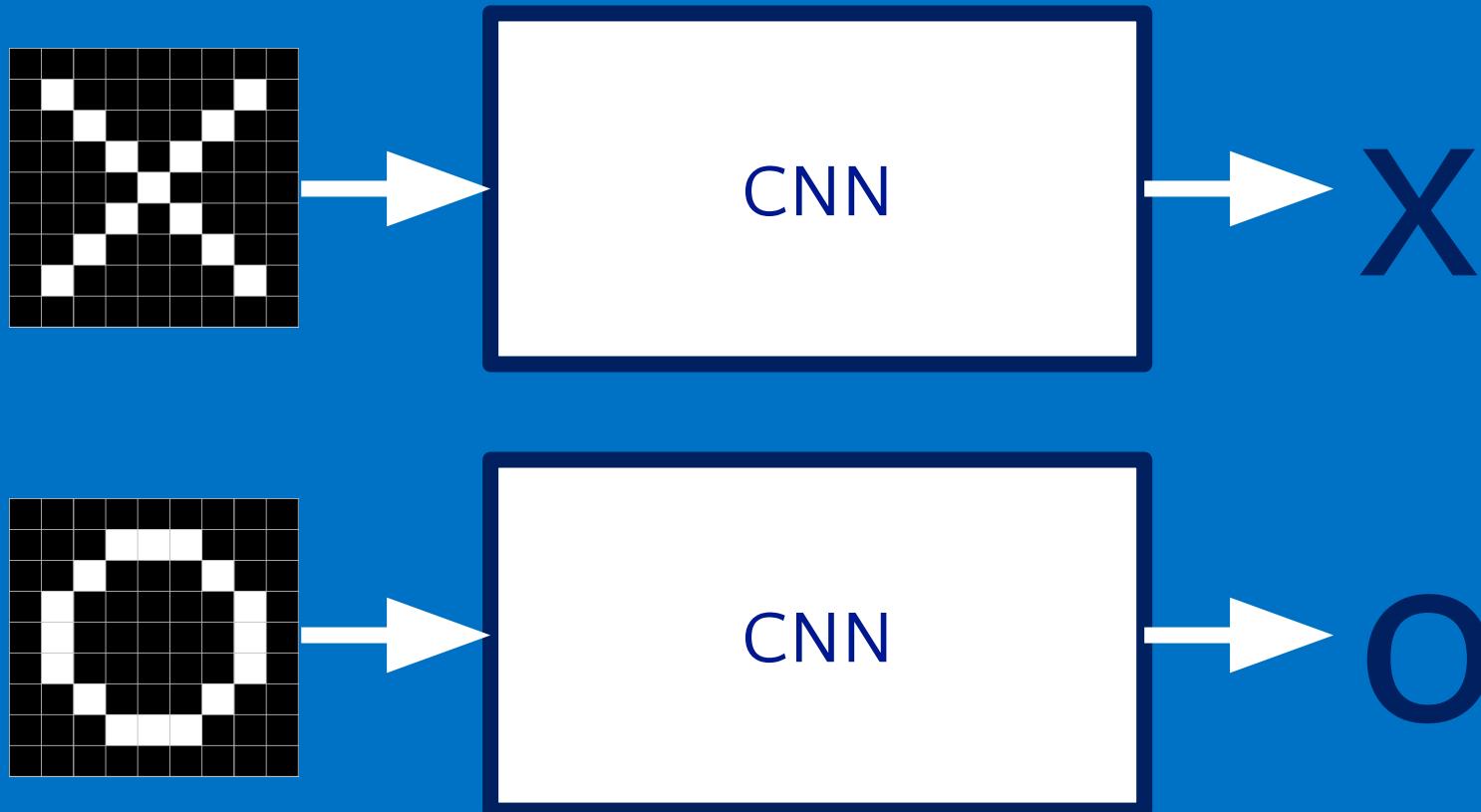
An example CNN: X's and O's

Says whether a picture is of an X or an O

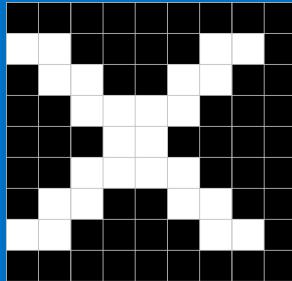
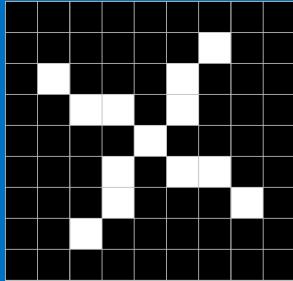
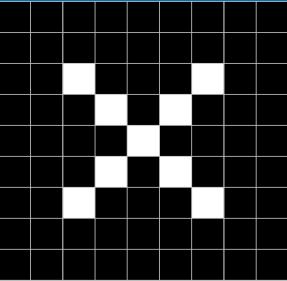
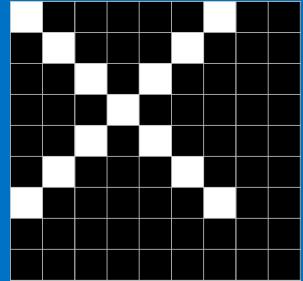
A two-dimensional
array of pixels



CNN: Identifying X's and O's



Trickier cases

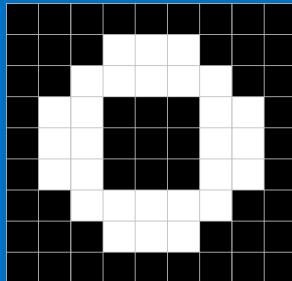
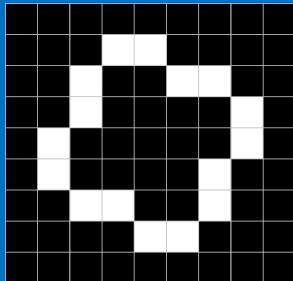
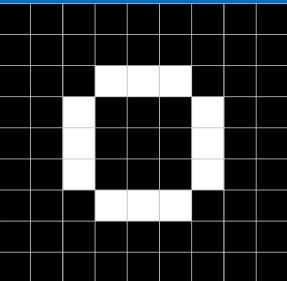
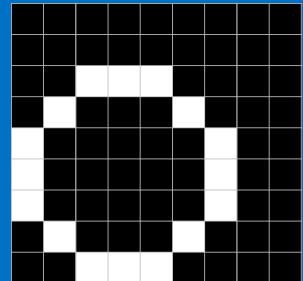


translation

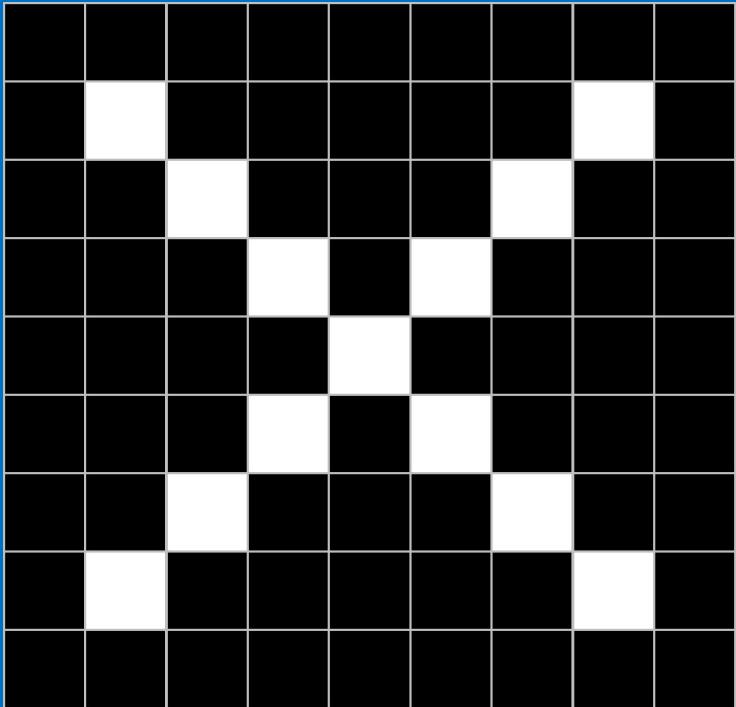
scaling

rotation

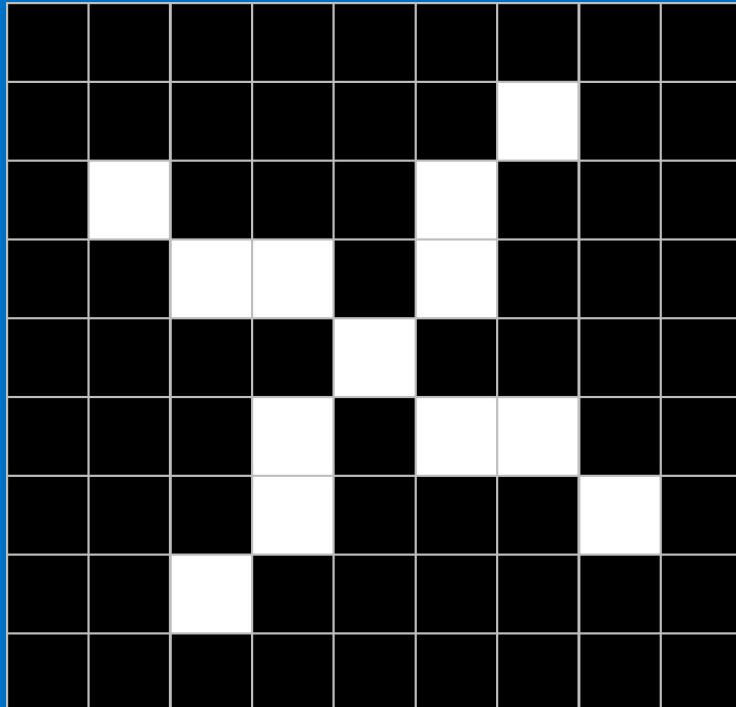
weight



Deciding is hard



?



What computers see

?

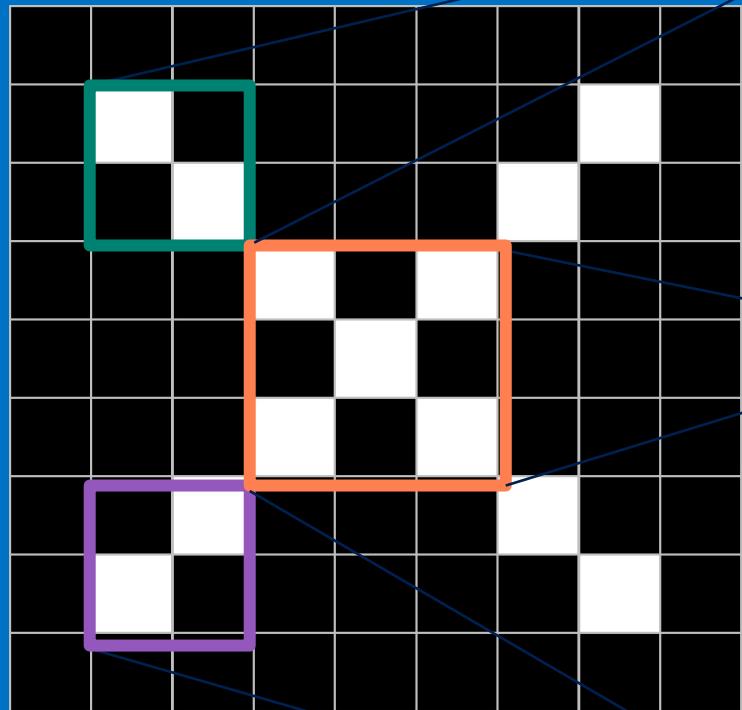
ANSWER

What computers see

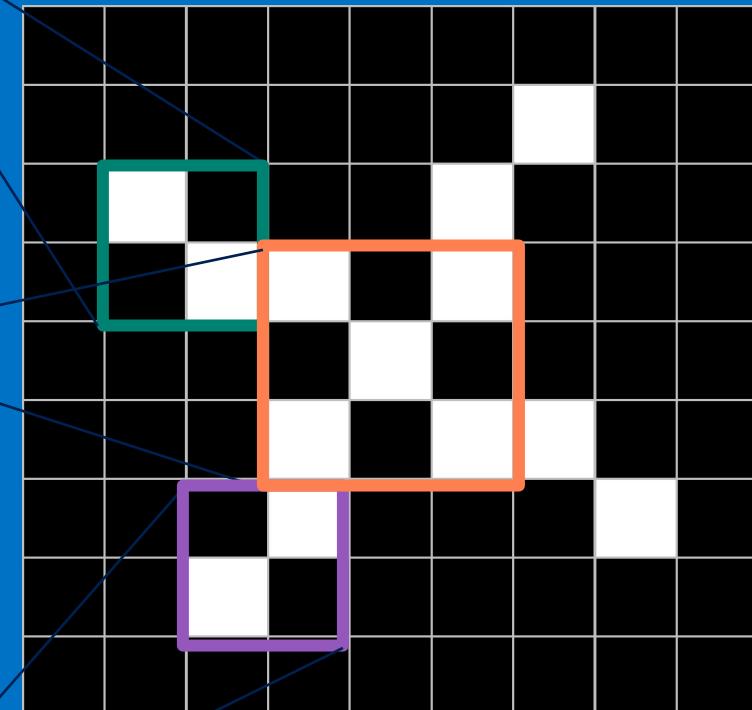
Computers are literal



CNNs match pieces of the image



=



=

=

Filters/kernels match pieces of the image

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

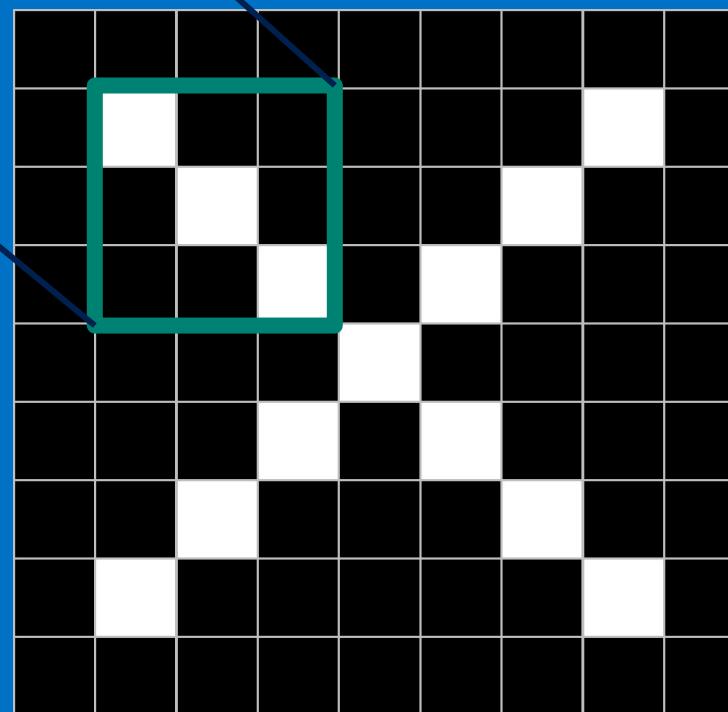
1	-1	1
-1	1	-1
1	-1	1

-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

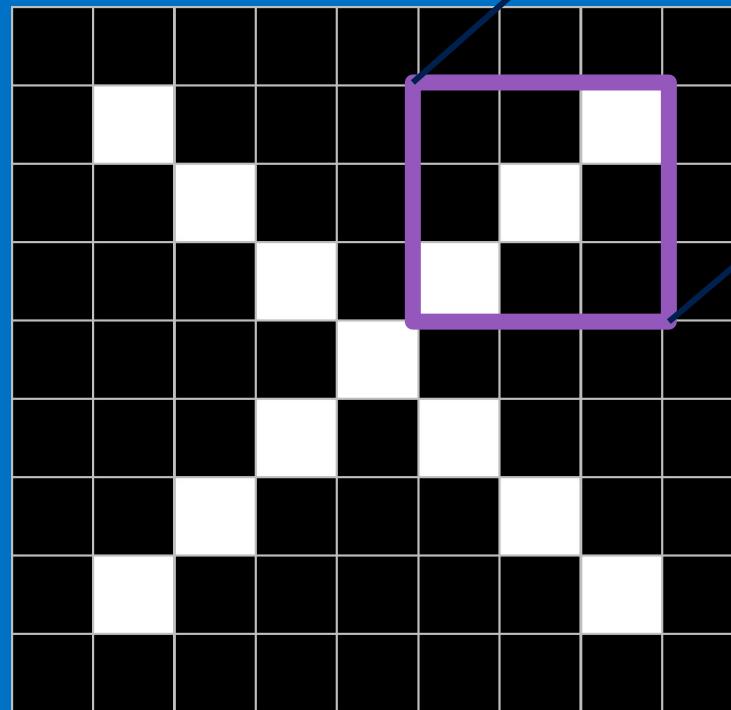
-1	-1	1
-1	1	-1
1	-1	-1



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

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

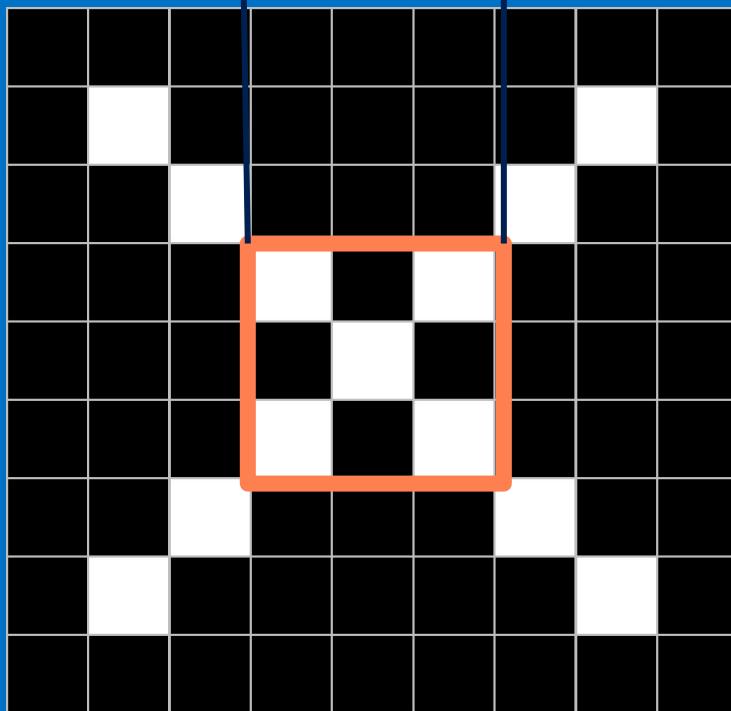
-1	-1	1
-1	1	-1
1	-1	-1



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

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

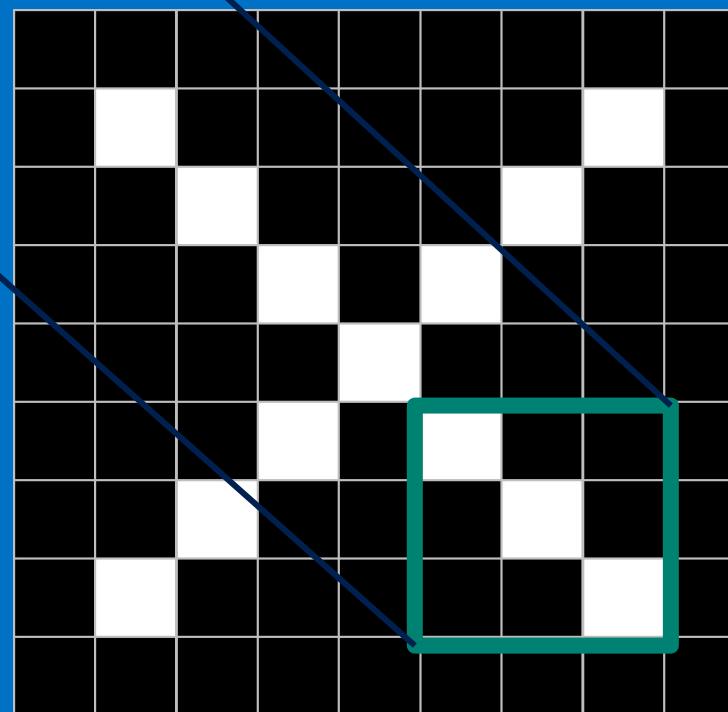
-1	-1	1
-1	1	-1
1	-1	-1



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



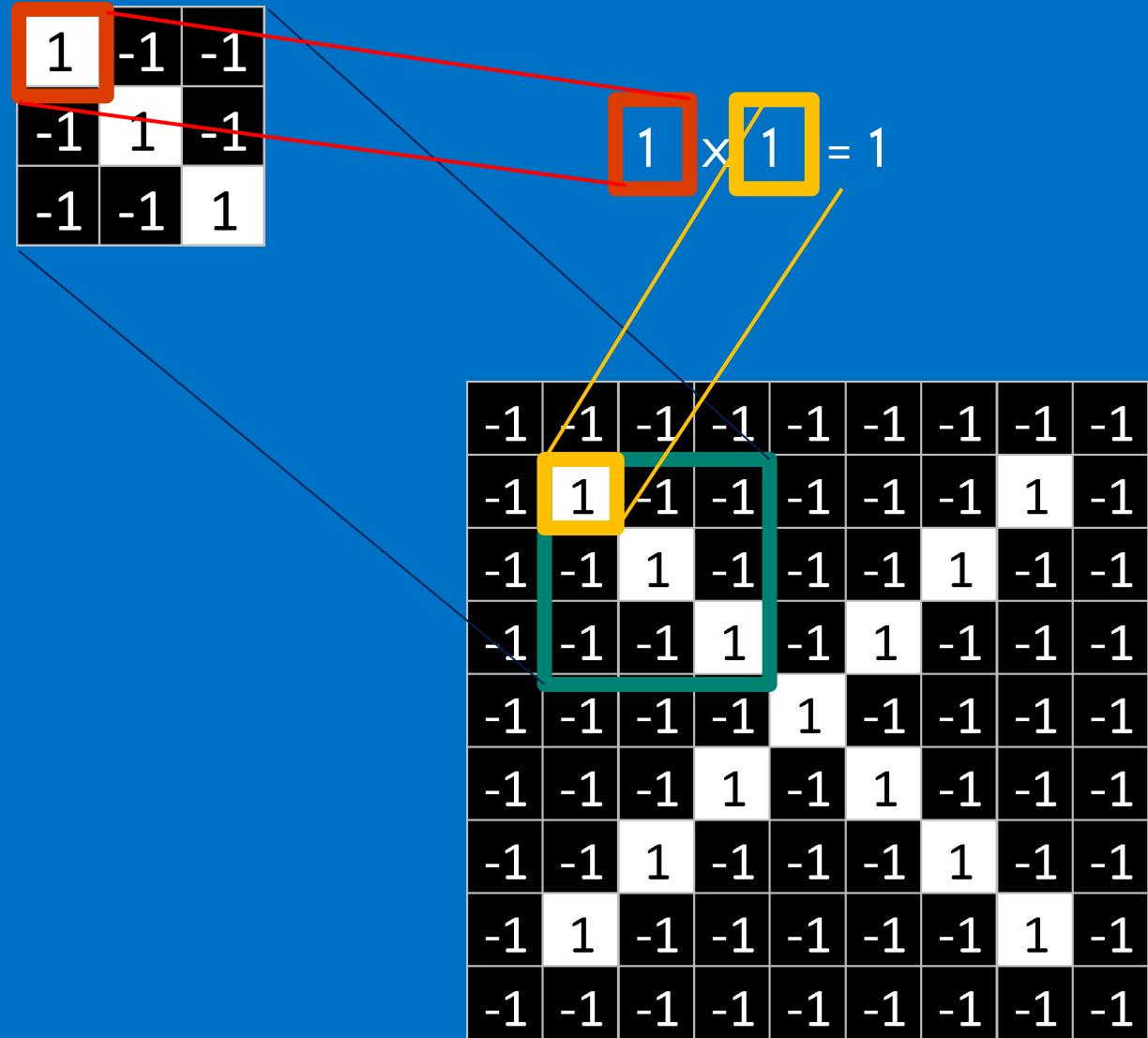
Convolution Layer - Filtering Our Image

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

Convolution Layer – Filtering Our Image

1. Line up the filter (i.e., mini piece of the image) and the original image.
2. Go through each pixel in the original image one by one and multiply it by the corresponding pixel in the filter
3. Find the total sum of all of these pixel multiplications
4. Divide this sum by the total number of pixels in the filter to determine its value for the activation map.

Convolution Layer – Filtering Our Image



Convolution Layer – Filtering Our Image

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

1	1	

Convolution Layer – Filtering Our Image

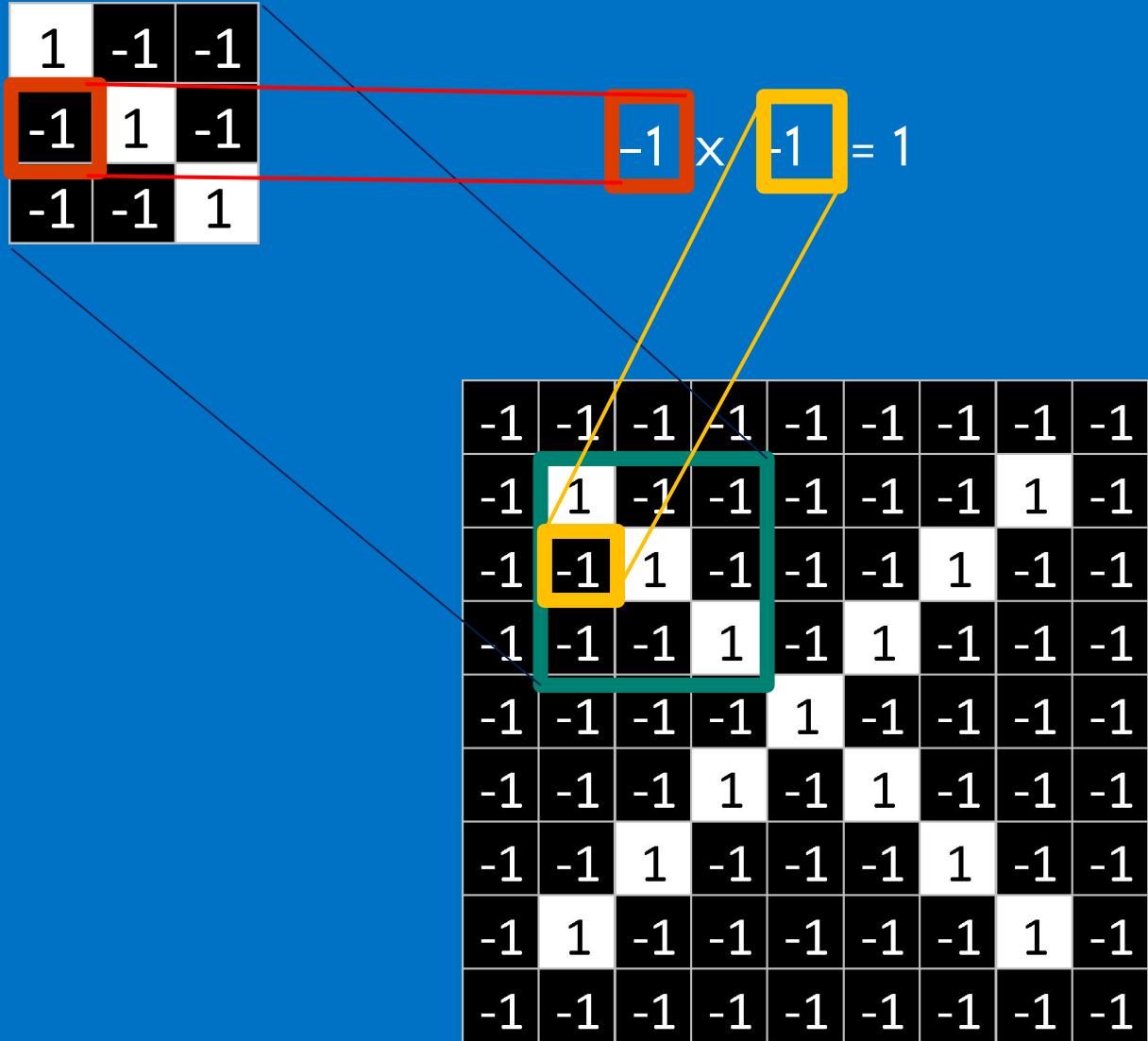
1	-1	-1
-1	1	-1
-1	-1	1

$$-1 \times -1 = 1$$

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

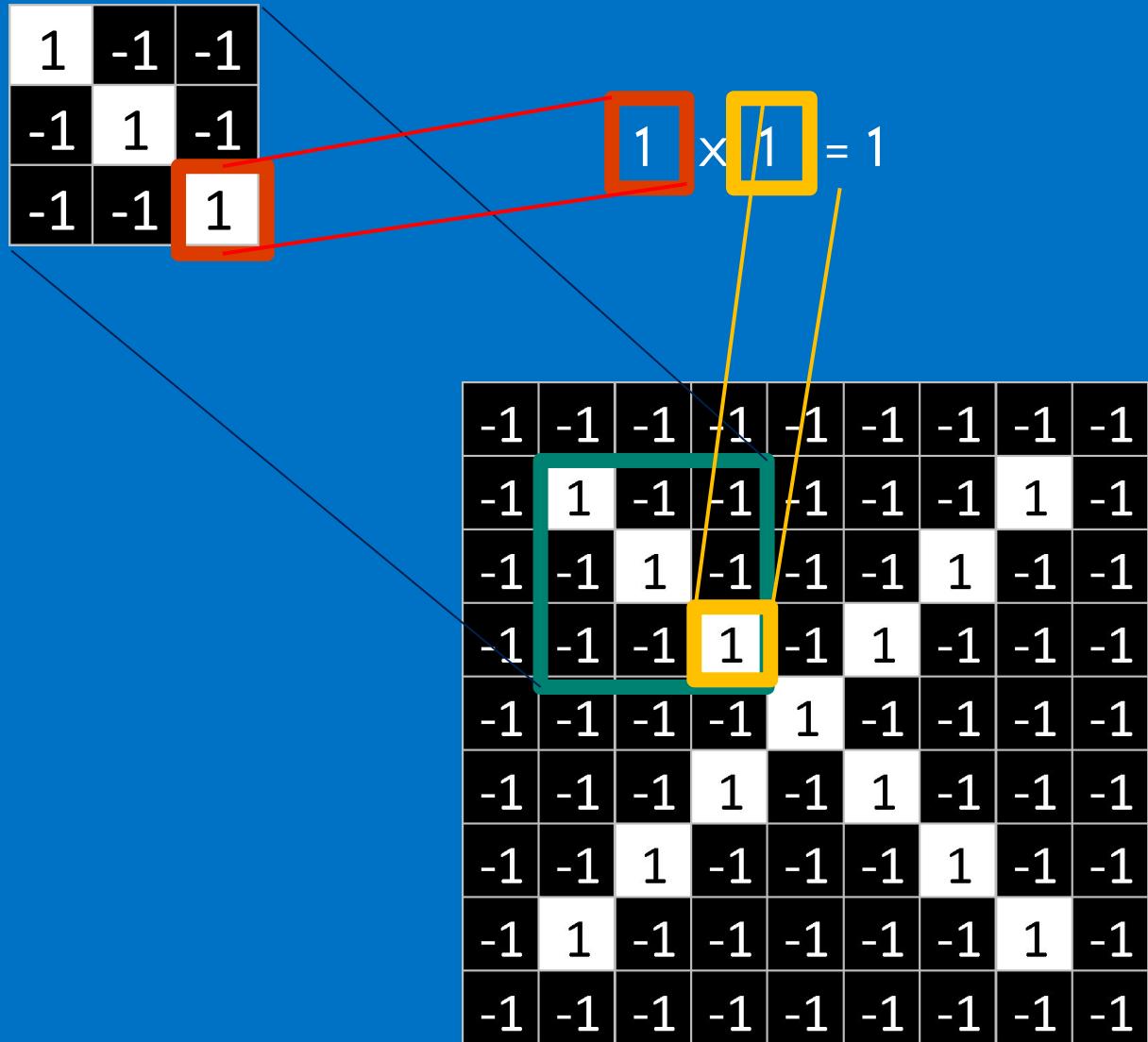
1	1	1

Convolution Layer – Filtering Our Image



1	1	1
1		

Convolution Layer – Filtering Our Image



1	1	1
1	1	1
1	1	1

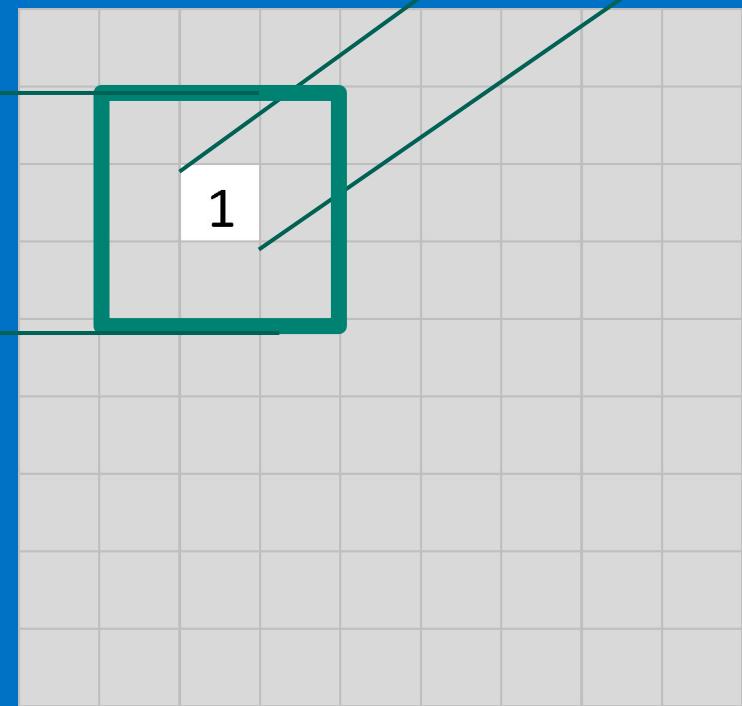
Convolution Layer – Filtering Our Image

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

1	1	1
1	1	1
1	1	1

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

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



Breakout: Apply Filter to Another Location

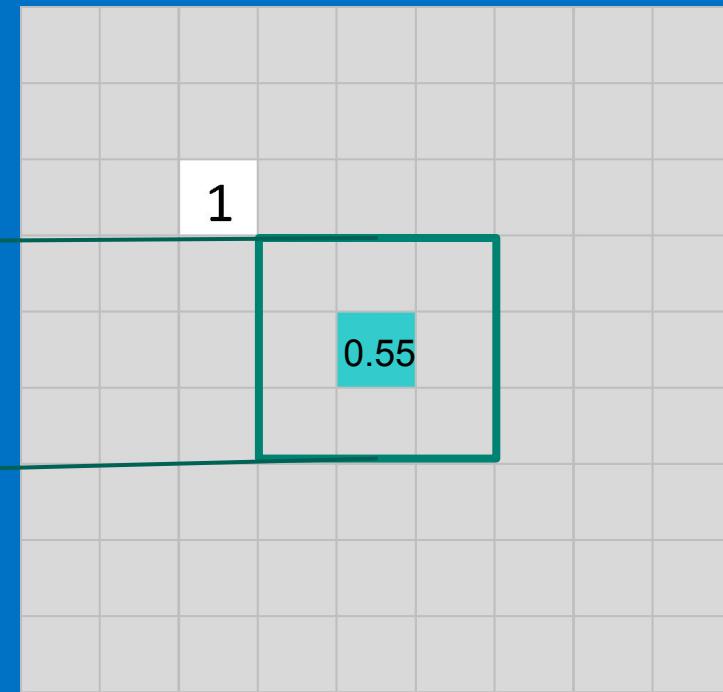
1	-1	-1
-1	1	-1
-1	-1	1

Breakout: Apply Filter to Another Location

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

A 9x9 input grid with values ranging from -1 to 1. A 3x3 kernel (highlighted in green) is applied at the center position (row 4, column 3). The result of this convolution step is highlighted in a green box.

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



Convolution Layer – Filtering Our 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

=

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

Activation Map

After the convolution layer, we went from a 9x9 image to a 7x7 filtered image – how?

Activation Map

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

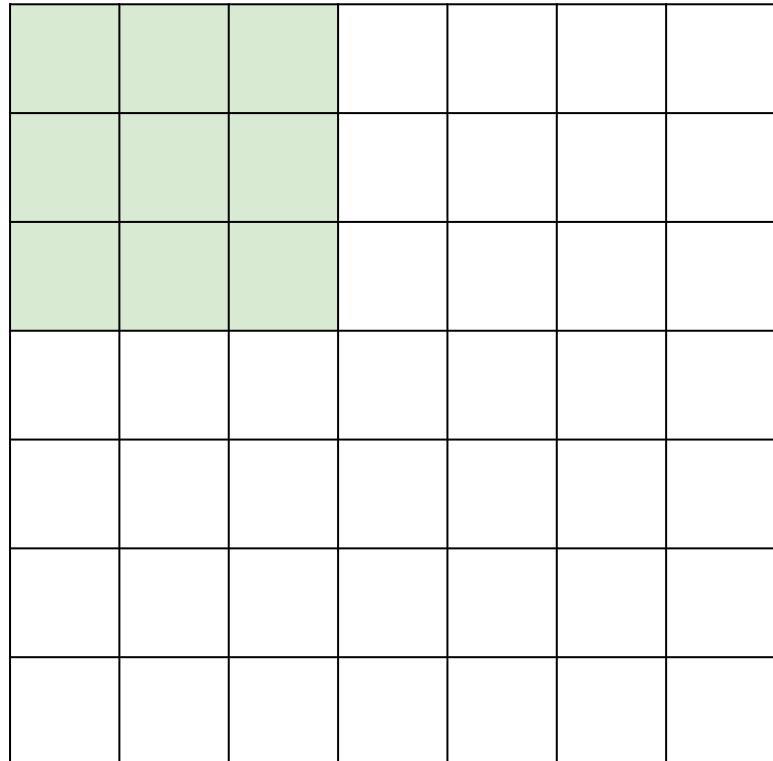


$$\begin{matrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{matrix} =$$

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

A closer look at spatial dimensions:

7

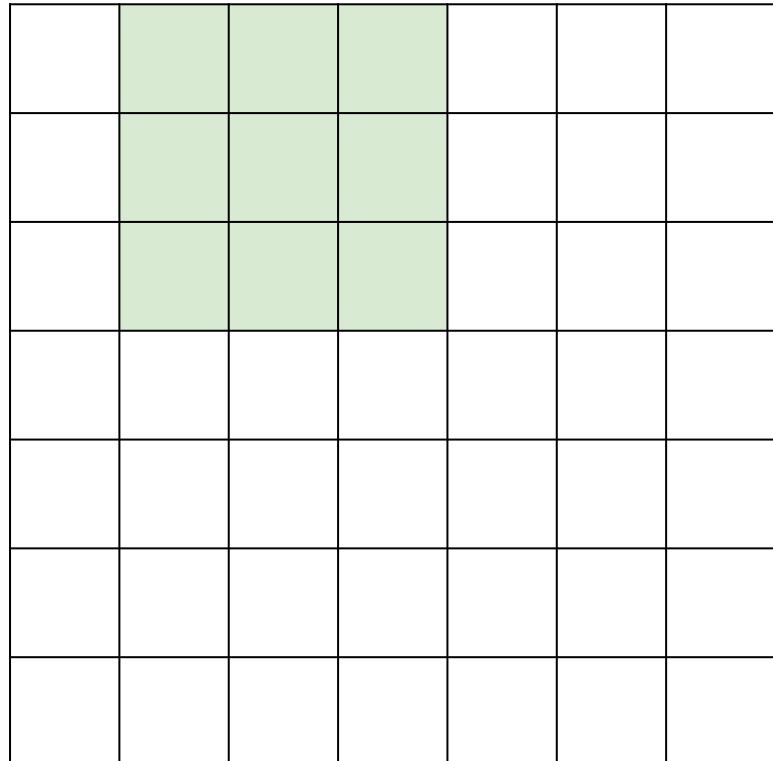


7x7 image input
Using a 3x3 filter

7

A closer look at spatial dimensions:

7

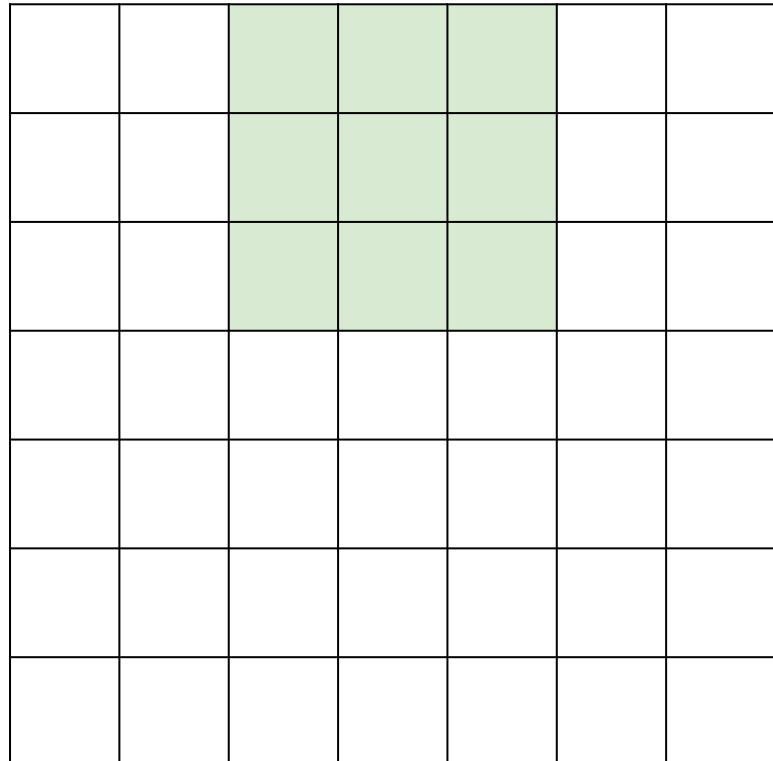


7x7 image input
Using a 3x3 filter

7

A closer look at spatial dimensions:

7

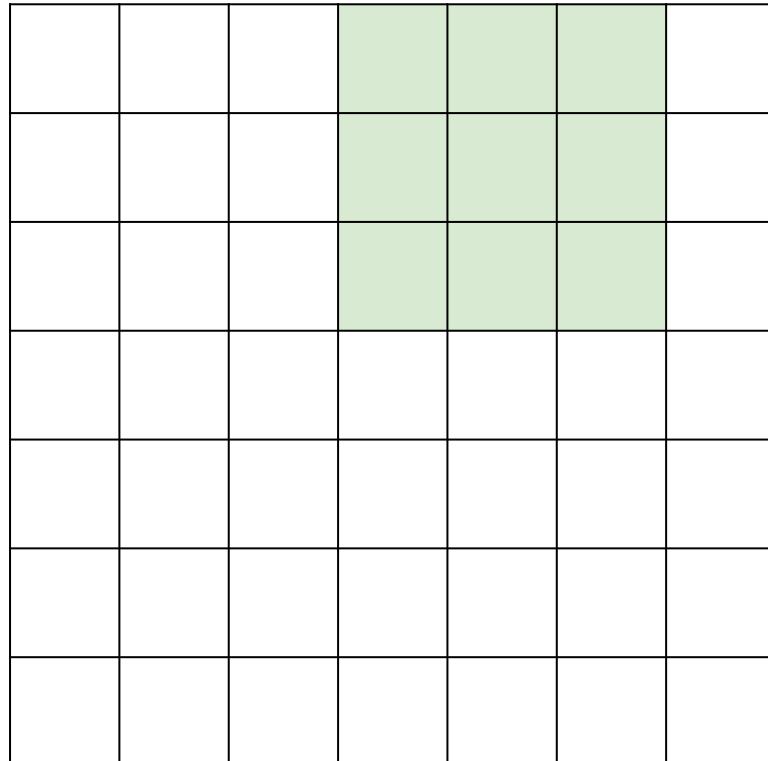


7x7 image input
Using a 3x3 filter

7

A closer look at spatial dimensions:

7

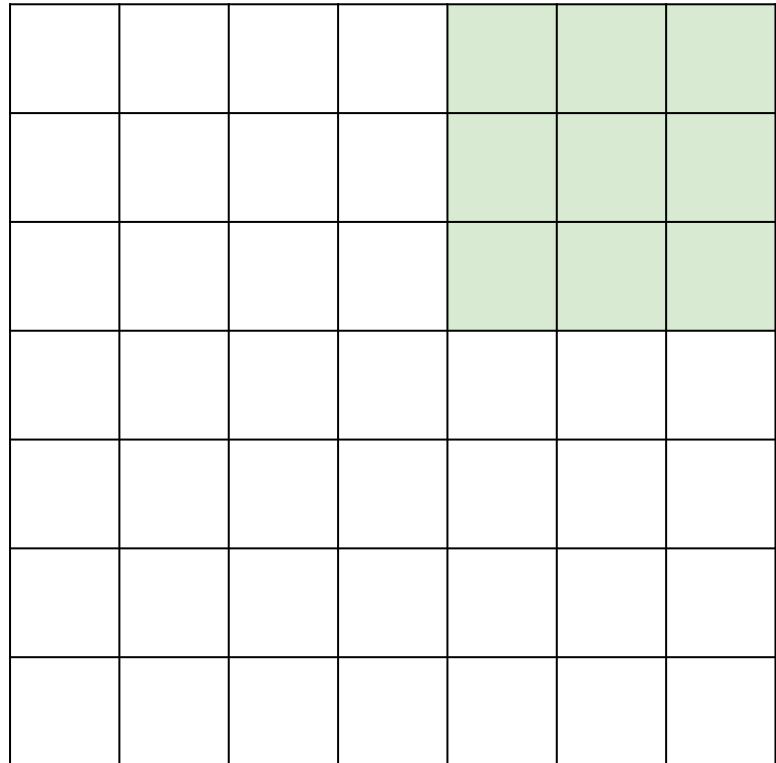


7x7 image input
Using a 3x3 filter

7

A closer look at spatial dimensions:

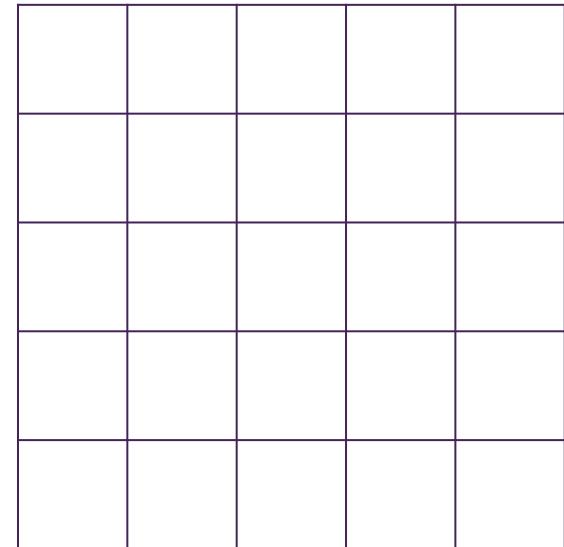
7



7

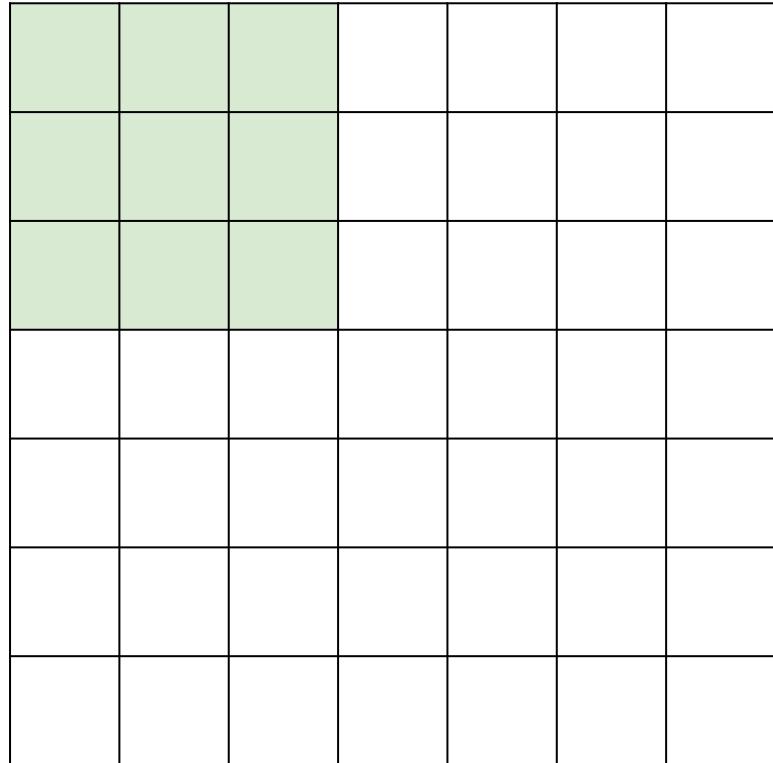
7x7 image input
Using a 3x3 filter

**=> 5x5
output**



A closer look at spatial dimensions:

7

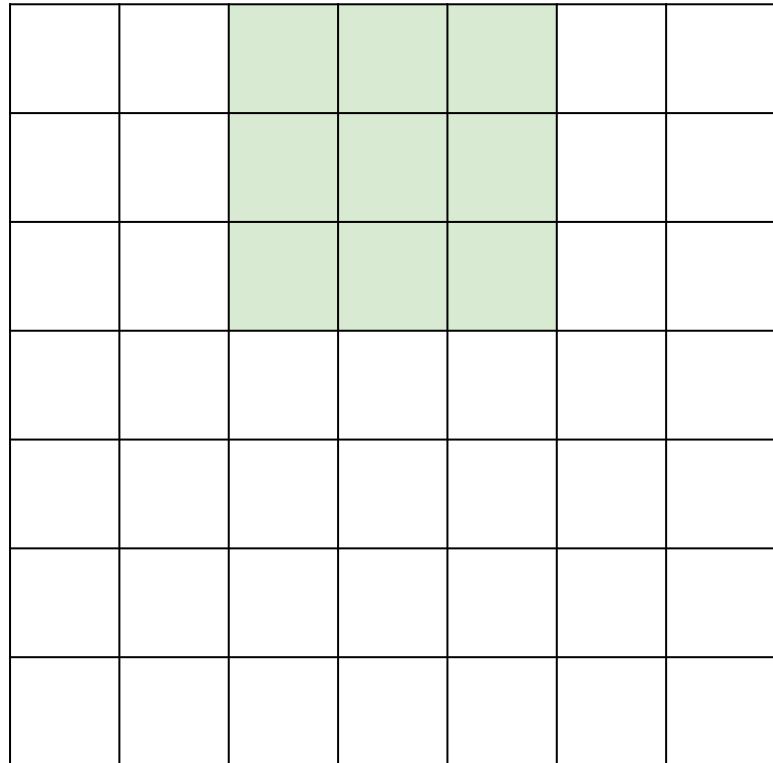


7

7x7 image input
Using a 3x3 filter
Applied **with stride 2**

A closer look at spatial dimensions:

7

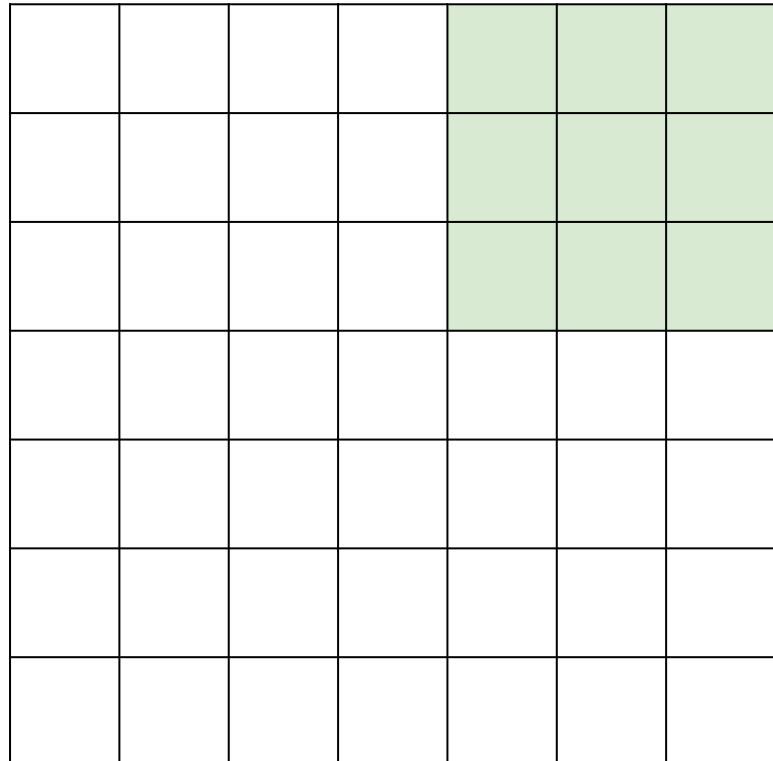


7

7x7 image input
Using a 3x3 filter
Applied **with stride 2**

A closer look at spatial dimensions:

7

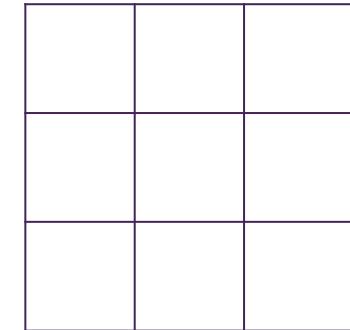


7

=> 3x3 output!

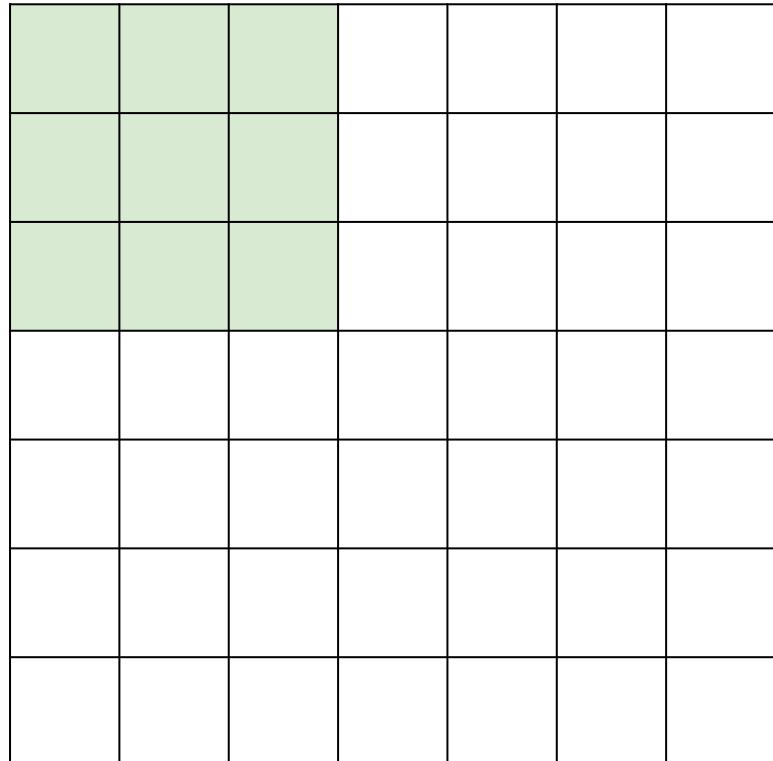
7x7 image input Using a
3x3 filter

Applied **with stride 2**



A closer look at spatial dimensions:

7

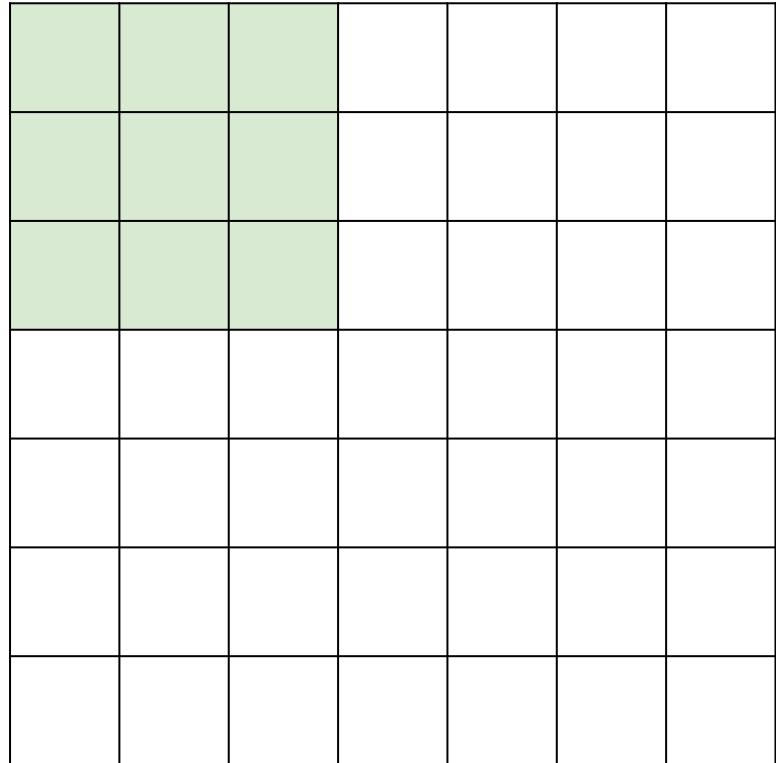


7

7x7 image input
What is the output of
3x3 filter applied
with stride 3?

A closer look at spatial dimensions:

7



7

7x7 image input

What is the output of
3x3 filter applied
with stride 3?

Doesn't fit!

For the convolution layer, you
cannot apply a filter with a stride
that does not fit cleanly.

In practice: Common to zero pad the border

0	0	0	0	0	0		
0							
0							
0							
0							

7x7 image input

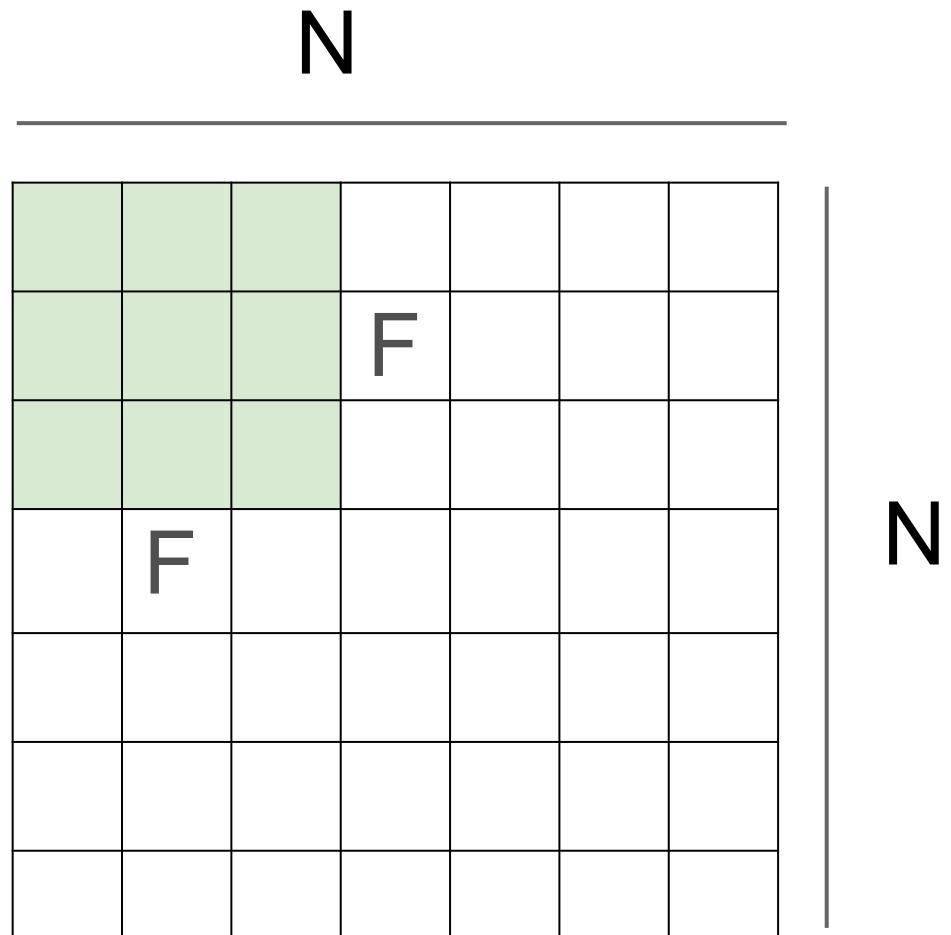
What is the output of 3x3 filter applied
with stride 1 and padded with a **1 pixel border**?

Answer: **7x7 output** (i.e., same as the original)

It's common to see convolutional layers with stride 1, filters of size $F \times F$, and zero-padding with $(F-1)/2$ (this will preserve the original size spatially).

e.g. $F = 3 \Rightarrow$ zero pad with 1 $F = 5 \Rightarrow$ zero pad with 2 $F = 7 \Rightarrow$ zero pad with 3

Calculating Output Size



N

N - original image dimension

F - filter dimension

S - stride

P - padding

Output size = $((N - F + 2P) / S) + 1$

e.g. $N = 7$, $F = 3$:

stride 1 => $(7 - 3)/1 + 1 = 5$

stride 2 => $(7 - 3)/2 + 1 = 3$

stride 3 => $(7 - 3)/3 + 1 = 2.33 \square$

Breakout: Compute Output Sizes

Input volume: 32x32

5x5 filter

Stride 1

Pad 2

Output volume size = ?

Input volume: 50x50

2x2 filter

Stride 2

Output volume size = ?

N - original image dimension

F - filter dimension

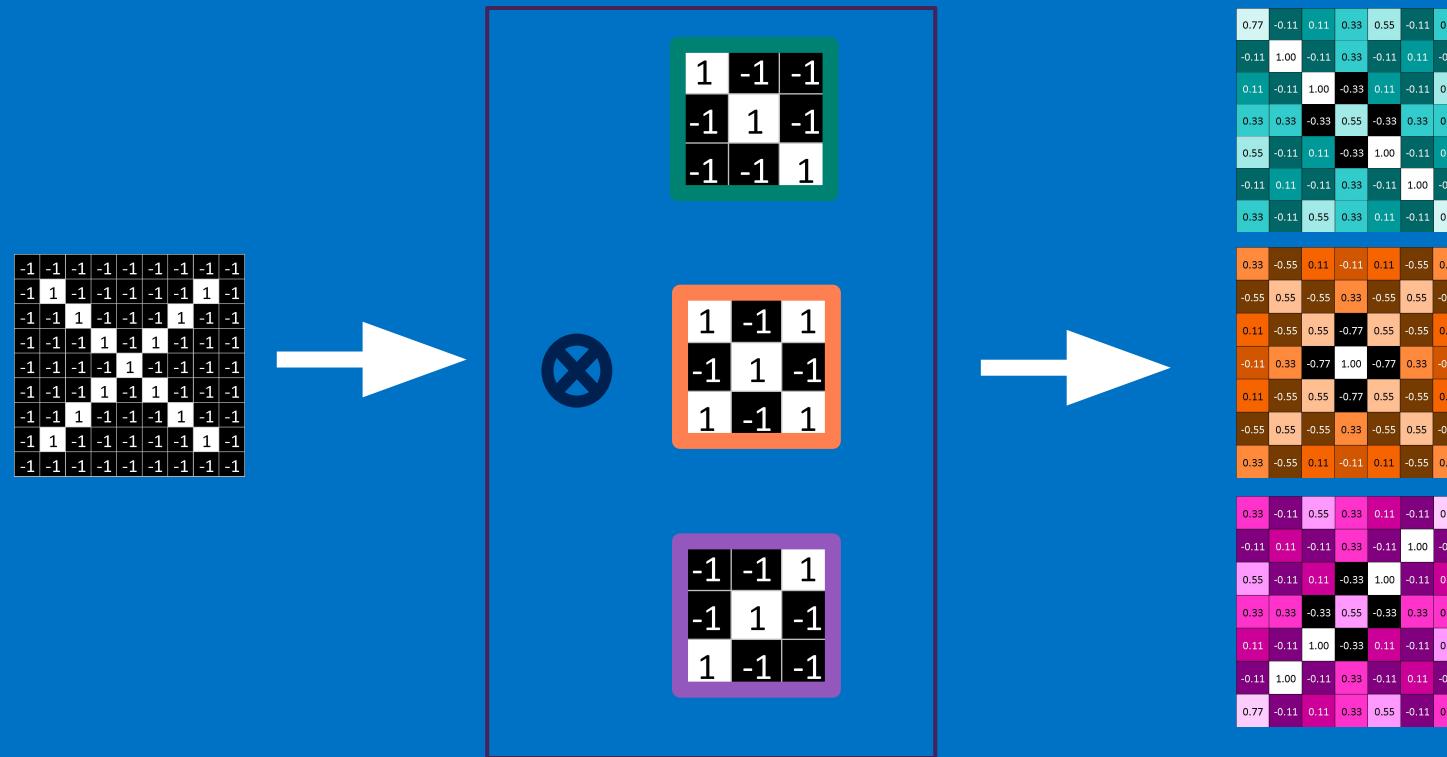
S - stride

P - padding

Output size = $((N - F + 2P) / S) + 1$

Convolution layer

One image becomes a stack of filtered images



Convolutional Layer: Summary

galvanize

- 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.

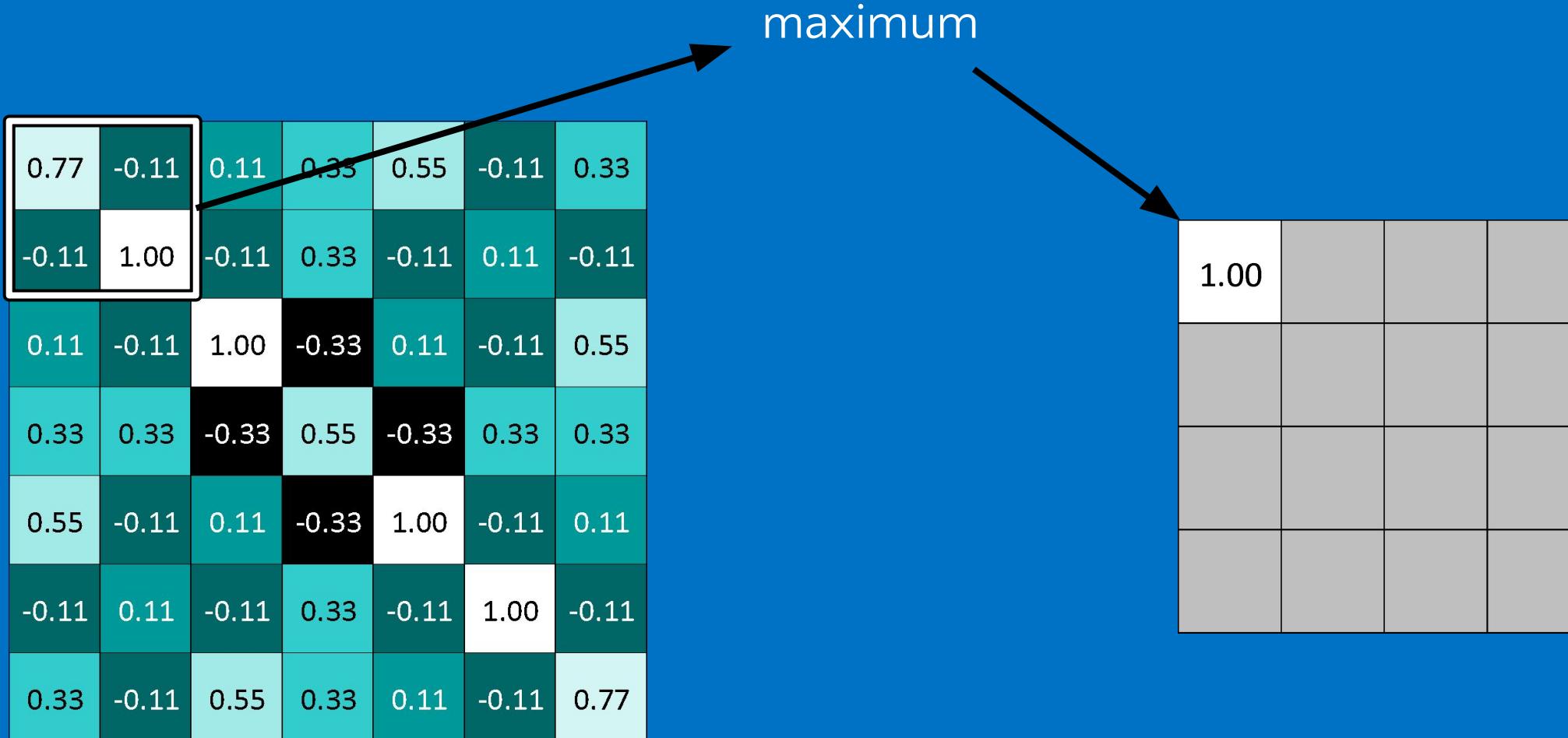
Common settings:

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

Pooling: Shrinking the image stack

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

Pooling (2x2 window / stride of 2)



Pooling (2x2 window / stride of 2)

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 (2x2 window / stride of 2)

Breakout: Calculate the Next 3 Values

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 (2x2 window / stride of 2)

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

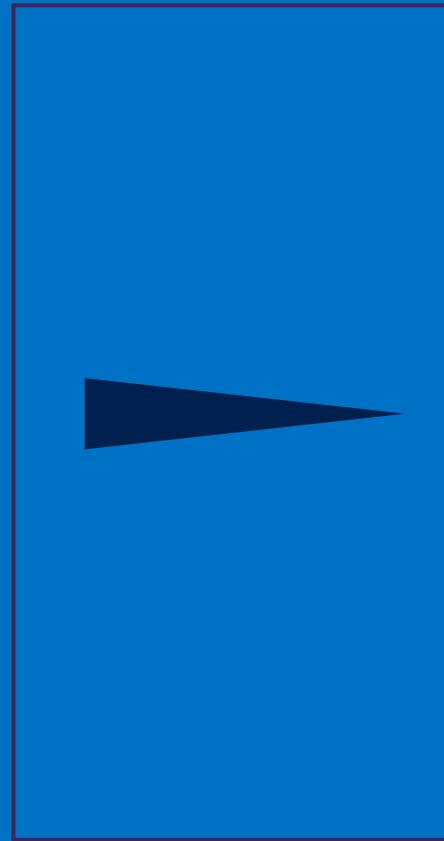
Pooling layer

A stack of images becomes a stack of smaller images.

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

0.33	-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

0.33	-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



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

0.55	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	0.55	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 Layers – Technical Summary

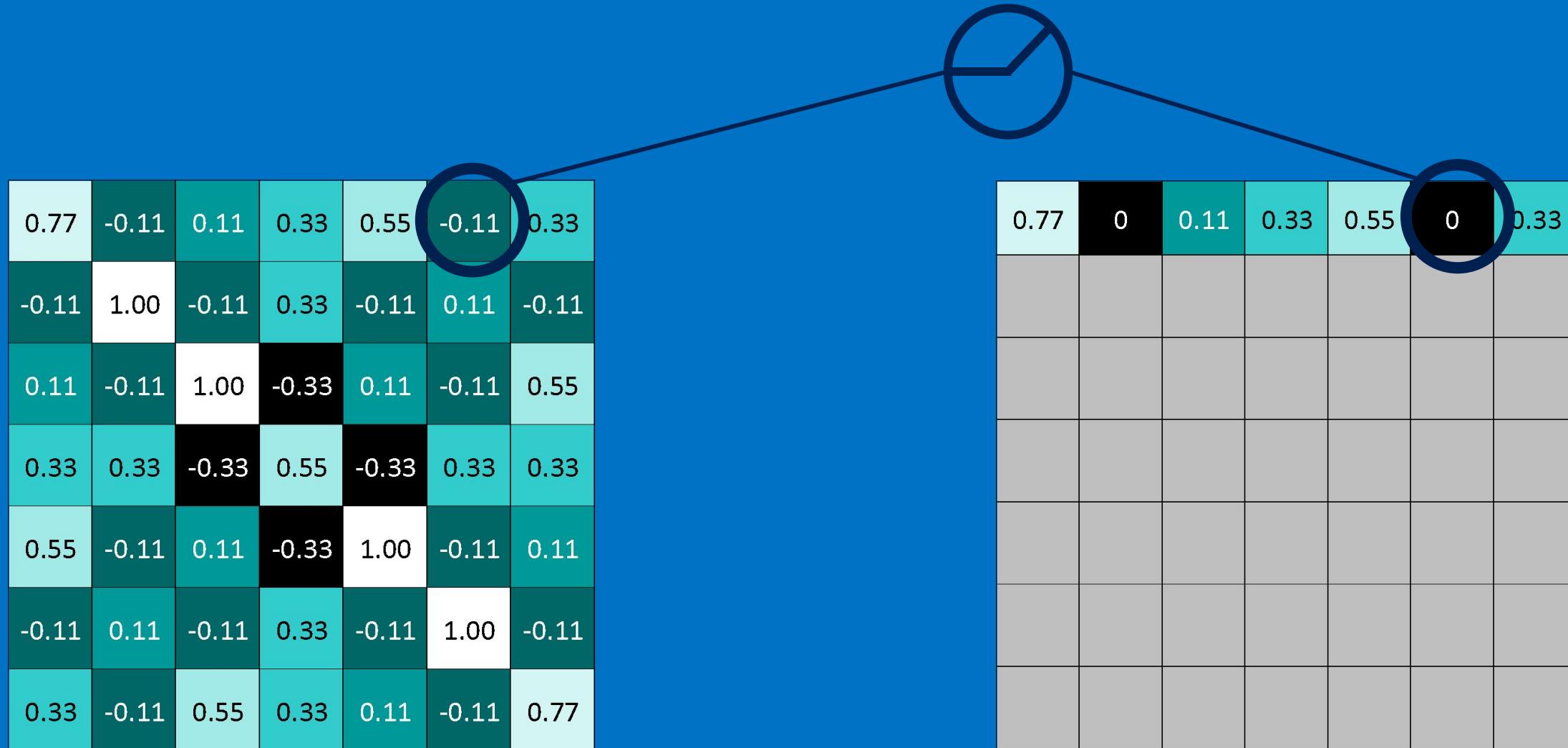
- Accepts a volume of size $W_1 \times H_1 \times D_1$
 - Requires three hyperparameters:
 - their spatial extent F ,
 - the stride S ,
 - Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F)/S + 1$
 - $H_2 = (H_1 - F)/S + 1$
 - $D_2 = D_1$
 - Introduces zero parameters since it computes a fixed function of the input
 - Note that it is not common to use zero-padding for Pooling layers
- Common settings:**
- $F = 2, S = 2$
- $F = 3, S = 2$

Concept Review: Activation Layer

- CNNs use activation function(s) just like MLPs
- Types of activation functions in this layer:
 - ReLU – maps all values between 0 and ∞
 - Leaky ReLU – instead of mapping all negative values to 0, these values have a small negative slope of about 0.01
 - Tanh – maps all values between -1 and 1
 - Sigmoid – maps all values between 0 and 1
- Keeps the math from breaking by tweaking each of the values just a bit.

Rectified Linear Units (ReLUs)

Computational units that perform the process of normalization



Rectified Linear Units (ReLUs)

Computational units that perform the process of normalization

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

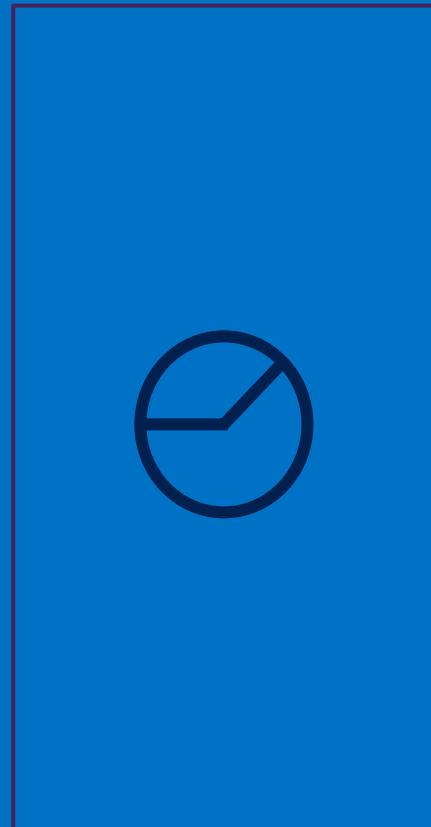
Activation Layer

A stack of images becomes a stack of images with adjusted values.

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

0.33	-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

0.33	-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



0.77	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

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

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

Layers get stacked

The output of one becomes the input of the next.

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



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

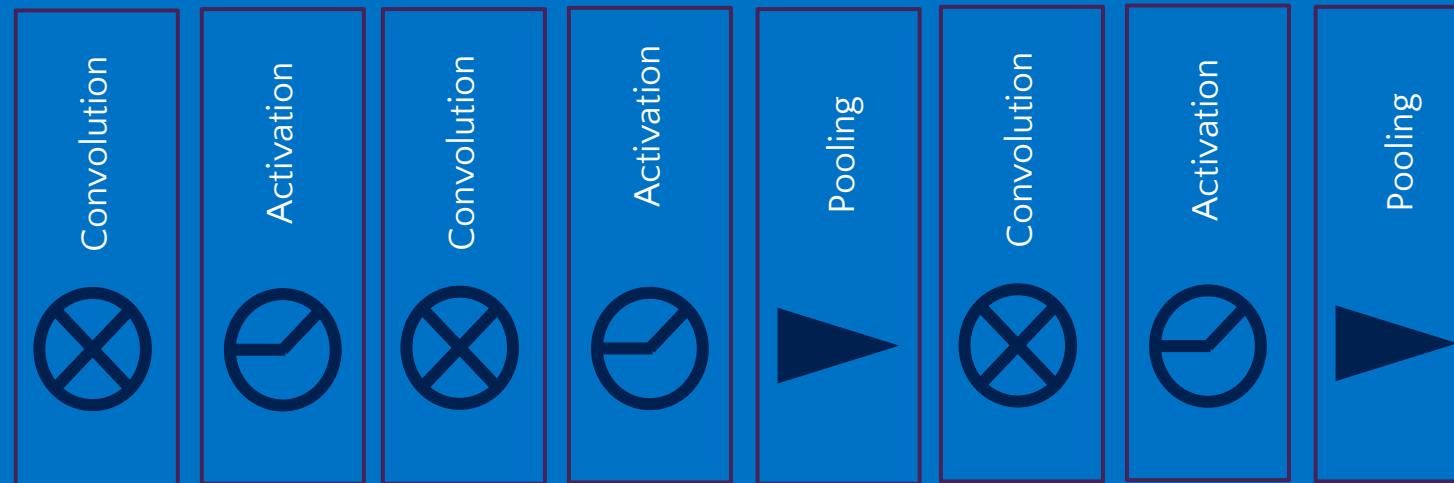
0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

0.33	0.55	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

Deep stacking

Layers can be repeated several (or many) 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	-1	-1	-1	-1



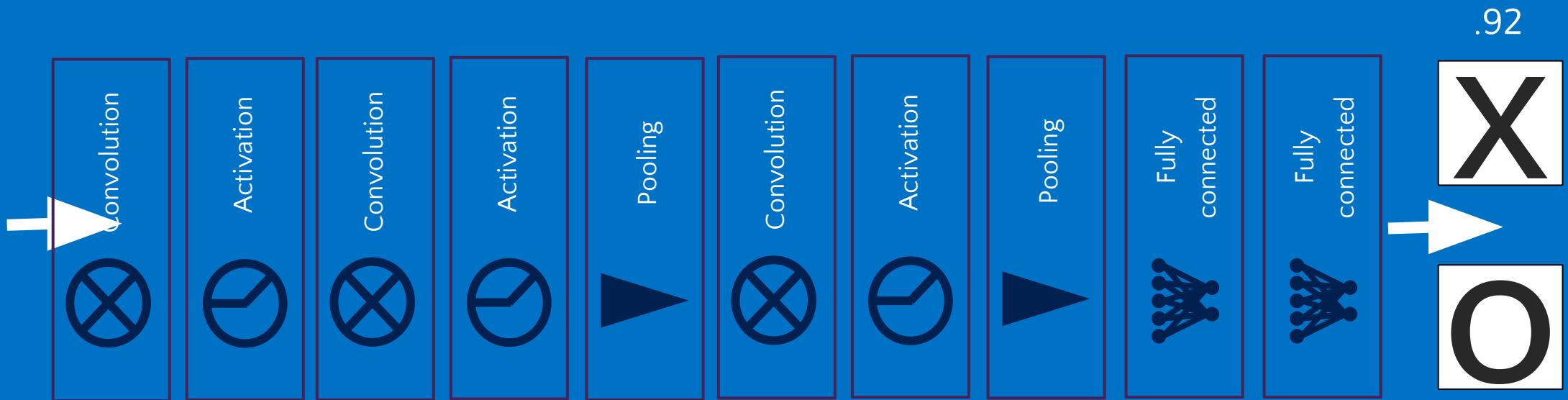
1.00	0.55
0.55	1.00
1.00	0.55
0.55	0.55
0.55	1.00
1.00	0.55

Concept Review: Fully Connected Layer

- Computing our votes for X & O through feed forward calculations
- The votes are the summation of the dot products between the activations & the weights
- We classify the image as an X or O depending on which has the higher value for a vote
- Both the filters used (e.g., the 3x3 diagonal line images) and voting weights are updated through backpropagation
- You can think of this as an MLP just stuck onto the end of the CNN

Putting it all together

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



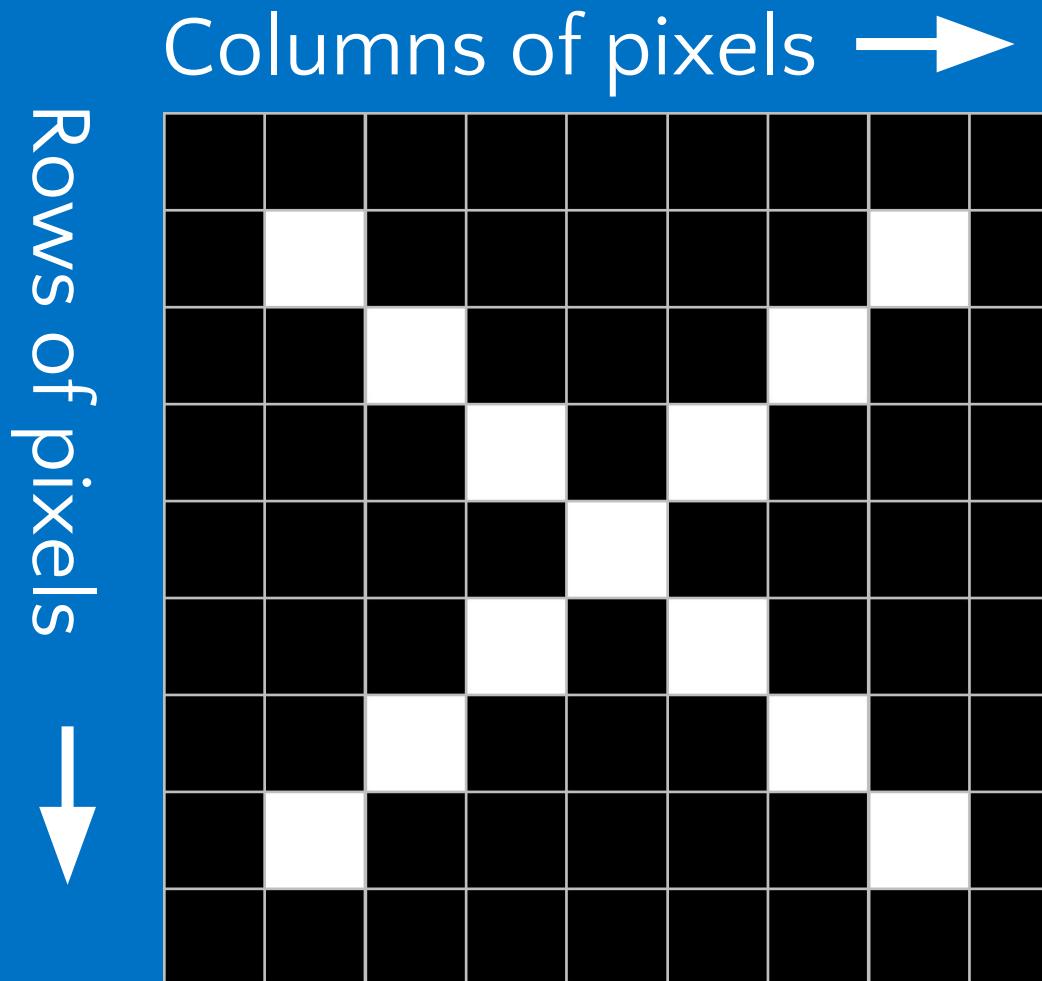
Available Hyperparameters

- Convolution
 - Number of filters
 - Size of filters (dimensions in pixels)
- Pooling
 - Window size
 - Window stride
- Fully Connected
 - Number of intermediate/hidden neurons

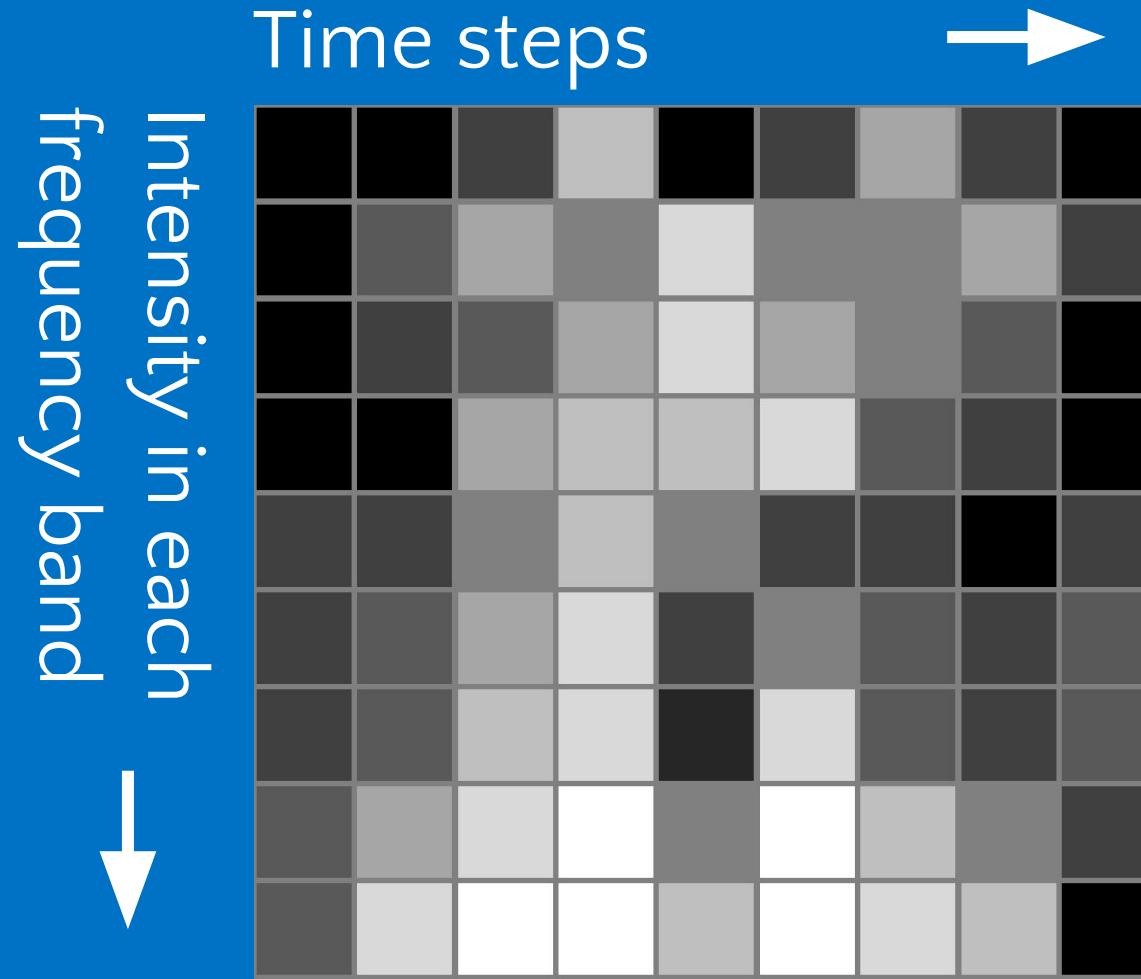
Available Architecture Settings

- How many of each type of layer do you want?
- What order do you want your layers to be in?
- You can even design your own custom layers

Order in Images

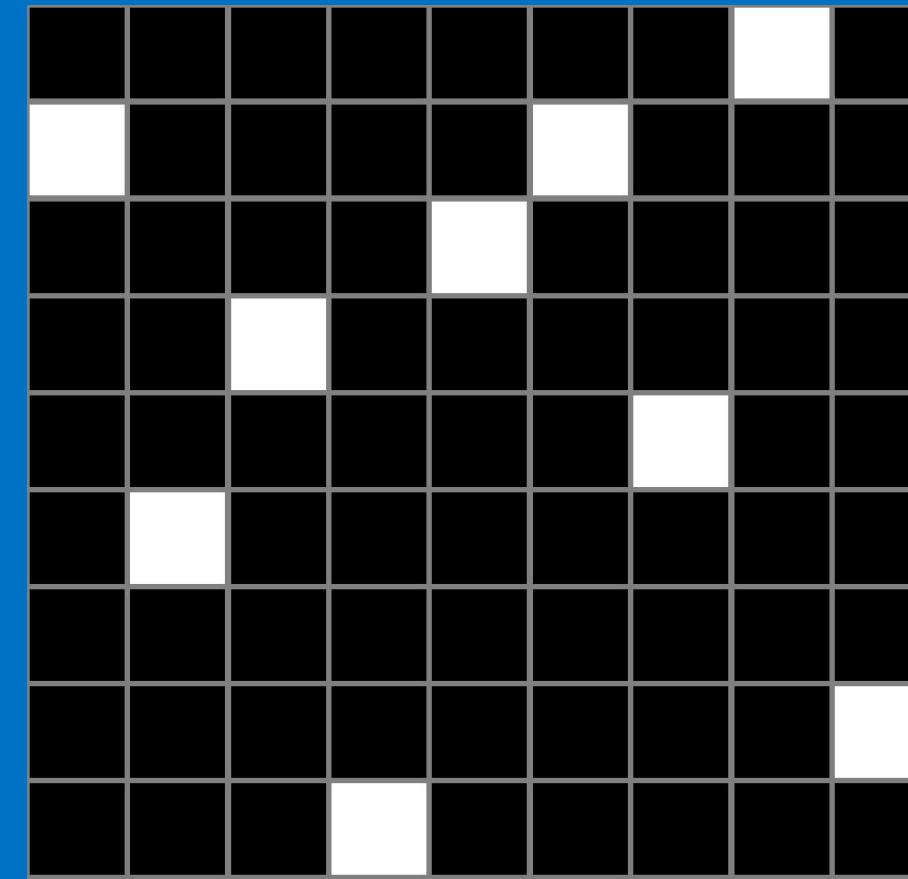


Order in Sound Data



Order in Text Data

Position in sentence



Words in dictionary

Customer Data (Not good for CNNs)

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



Customers



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

Advantages & Disadvantages of CNNs

Advantages:

- CNNs are great at finding patterns and using these patterns to classify images.
- If you can make your data look like images, then they are super useful!
- You can use any 2D, 3D, or even higher dimensional data in a CNN.

Disadvantages:

- CNNs only capture local “spatial” patterns in data.
- If your data is just as useful after swapping any of your columns with each other, then you can’t use CNNs.

Check in Questions

- What are the four types of layers in CNNs?
- How does convolution work?
- How does pooling work?

Additional Resources

- Visual Explanation of Image Kernels/Filters –
<http://setosa.io/ev/image-kernels/>
- Convolution Demo –
<http://cs231n.github.io/convolutional-networks/> (note the ‘Case Studies’ and ‘Computational Considerations’ sections)
- Another explanation of CNNs –
<https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>
- Working with images in Python
 - Skimage – <http://scikit-image.org/docs/dev/api/skimage.html>
 - PIL – <http://www.pythonware.com/products/pil/>