

Deep Learning for Computer Vision

Ahmed Hosny Abdel-Gawad

Senior AI/CV Engineer



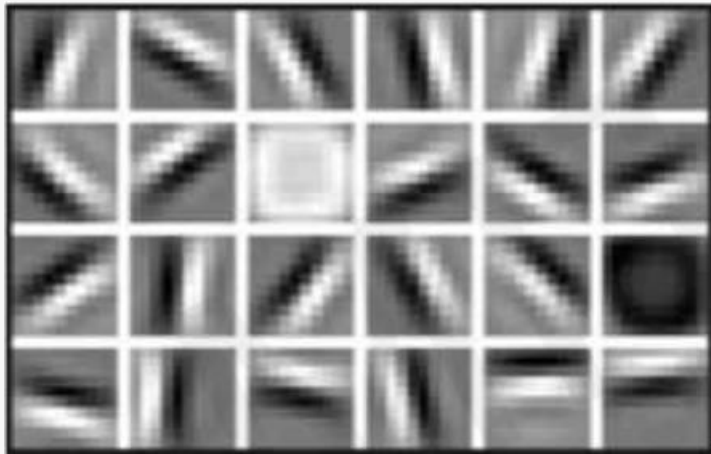
Deep Learning: ConvNets

Feature Representations

Hand engineered features are time consuming, brittle, and not scalable in practice

Can we learn the **underlying features** directly from data?

Low Level Features



Lines & Edges

Mid Level Features



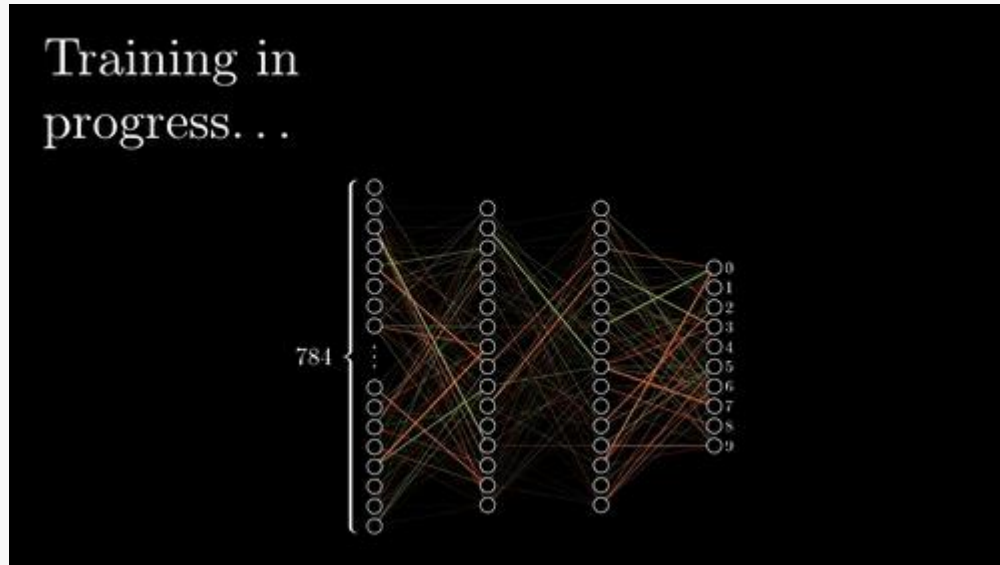
Eyes & Nose & Ears

High Level Features



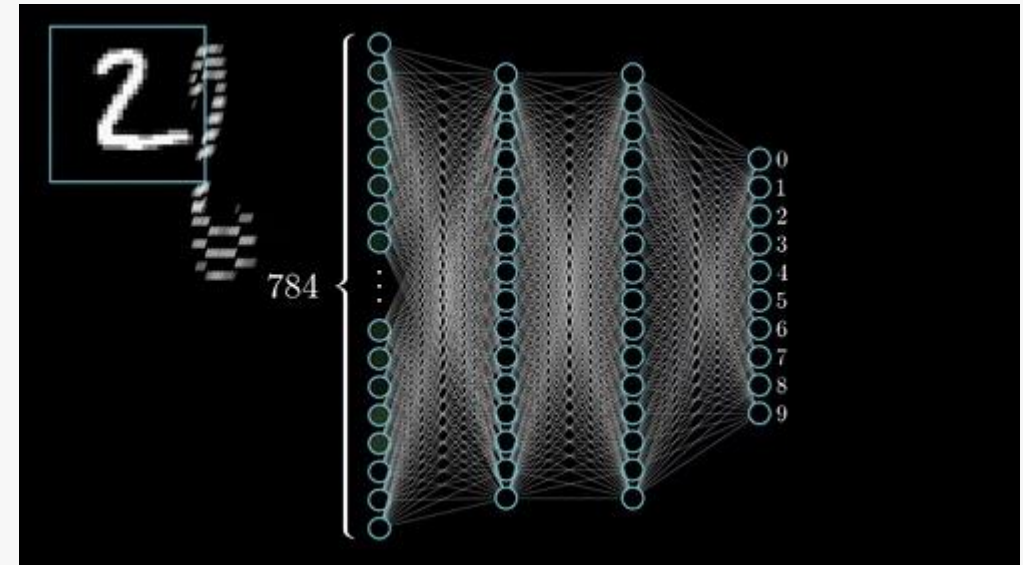
Facial Structure

Fully Connected Neural Networks



Input: 2D Image

- Vector of pixel values



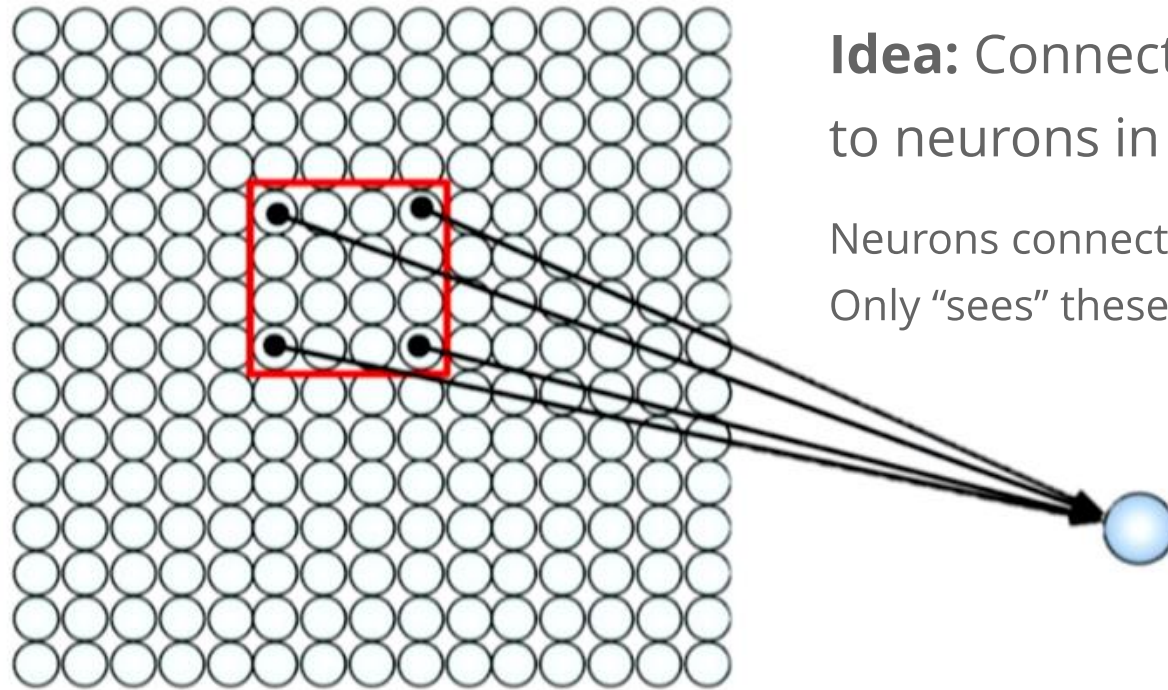
Fully Connected: Connect neurons in hidden layer to all neurons in input

- No spatial information!
- And many, many parameters.

Using **Spatial** Structure

Input: 2D Image

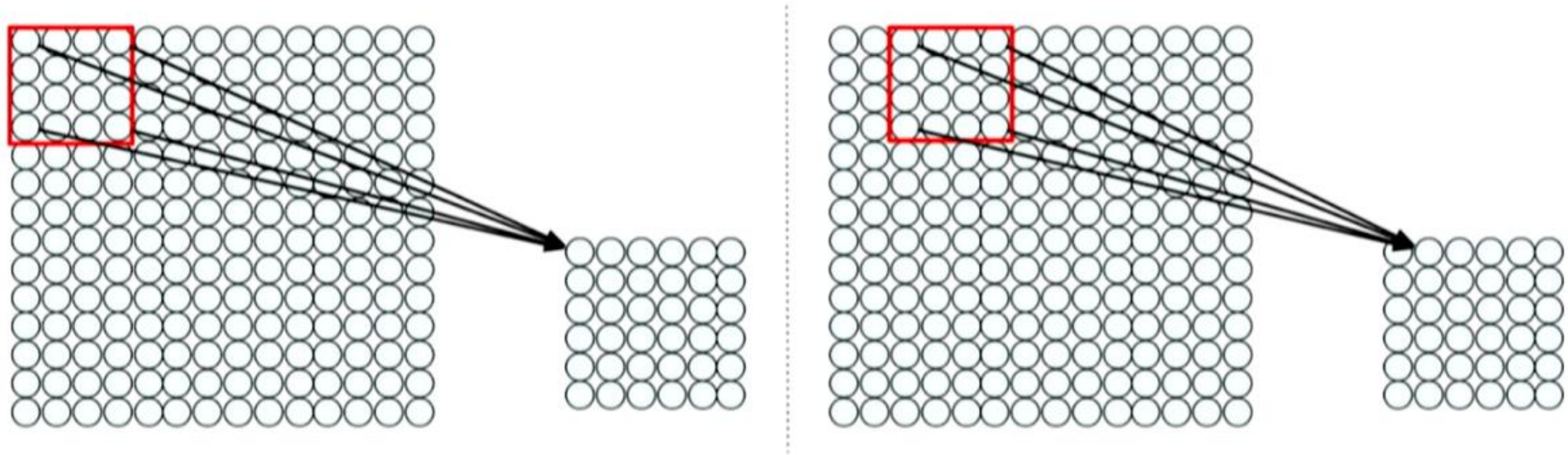
- Array of pixel values



Idea: Connect patches of input to neurons in hidden layer.

Neurons connected to region of input.
Only "sees" these values.

Using **Spatial** Structure

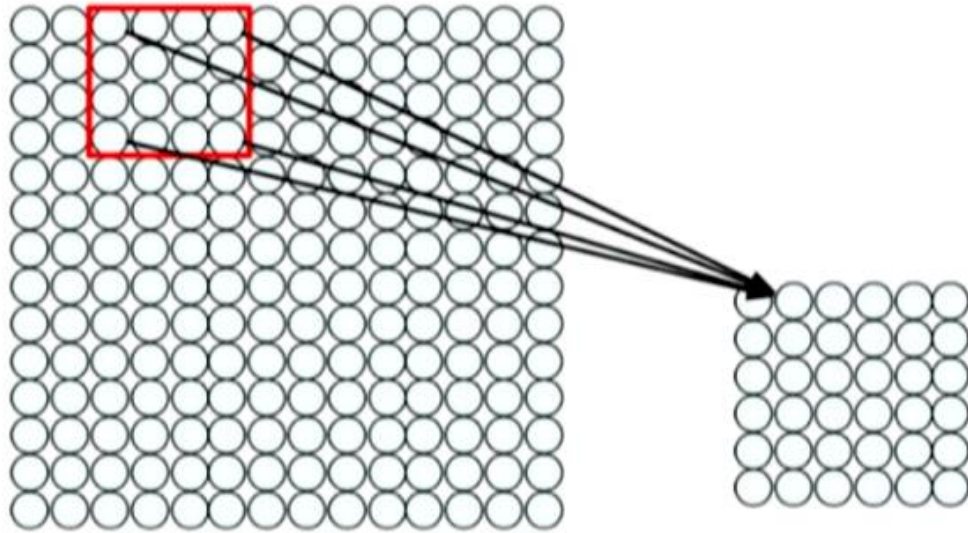


Connect patch in input layer to a single neuron in subsequent layer.

Use sliding window to define connections.

How can we **weight** the patch to detect particular features?

Using **Spatial** Structure



- Filter of size 4x4: 16 different weights
- Apply this same filter to 4x4 patches in input
- Shift by 2 pixels for next patch

This “patchy” operation is **convolution**

1. Apply a set of weights – a filter – to extract local features
2. Use **multiple filters** to extract different features
3. **Spatially share** parameters of each filter

So what is Convolution?

Let's try to understand the concept of convolution with a simple **1-dimensional** case first.

$$\begin{array}{|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ \hline \end{array} \otimes \begin{array}{|c|c|} \hline -1 & 1 \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 1 & 0 & 0 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|c|c|c|c|} \hline -1 & 1 & 0 & 0 & 1 & 1 & 1 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & -1 & 1 & 1 & 1 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|c|c|c|c|} \hline 0 & -1 & 1 & 0 & 1 & 1 & 1 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & -1 & 1 & 1 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|c|c|c|c|} \hline 0 & 0 & -1 & 1 & 1 & 1 & 1 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 1 & -1 & 1 \\ \hline \end{array}$$

So what is Convolution?

Then let's try a **2-dimensional** case.

0	0	1	0	1
2	1	0	1	0
0	0	1	0	1
0	1	2	1	0
2	1	0	1	0



1	0	1
0	0	0
1	0	1



2	0	4
4	4	2
3	2	2

0	1	0	1	0	1
2	0	1	0	1	0
0	1	0	1	0	1
0	1	2	1	0	
2	1	0	1	0	

0	0 ¹	1 ⁰	0 ¹	1
2	1 ⁰	0 ⁰	1 ⁰	0
0	0 ¹	1 ⁰	0 ¹	1
0	1	2	1	0
2	1	0	1	0

0	0		1	1	0	0	1
2	1		0	0	1	0	0
0	0		1	1	0	0	1
0	1	2	1	0			
2	1	0	1	0			

0	0	1	0	1			
2	1	0	0	1	1	0	
0	0	0	0	1	0	0	1
0	1	1	0	2	1	0	
2	1	0	1	0			

0	0	1	0	1			
2		1	1	0	0	1	0
0		0	0	1	0	0	1
0		1	1	2	0	1	0
2	1	0	1	0			

So what is Convolution?

Then let's try a **2-dimensional** case.

=

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

1	0	1
0	1	0
1	0	1

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

So what is **Convolution**?

=

Input

Channel 0

0	0	2	0	0
1	2	2	1	0
0	0	1	2	2
0	0	1	0	1
1	2	1	0	0

Kernel

Channel 0

1	1	1
1	0	0
0	-1	1

Bias

1

Output

Channel 0

5	5
1	7

Convolution Edge Pixels Problem

Input	Kernel	Output																																			
Channel 0	Channel 0	Channel 0																																			
<table><tr><td>0</td><td>0</td><td>2</td><td>0</td><td>0</td></tr><tr><td>1</td><td>2</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>2</td><td>2</td></tr><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>2</td><td>1</td><td>0</td><td>0</td></tr></table>	0	0	2	0	0	1	2	2	1	0	0	0	1	2	2	0	0	1	0	1	1	2	1	0	0	<table><tr><td>1</td><td>1</td></tr><tr><td>0</td><td>-1</td></tr></table> <table><tr><td>Bias</td></tr><tr><td>1</td></tr></table>	1	1	0	-1	Bias	1	<table><tr><td>-1</td><td>2</td></tr><tr><td>1</td><td>4</td></tr></table>	-1	2	1	4
0	0	2	0	0																																	
1	2	2	1	0																																	
0	0	1	2	2																																	
0	0	1	0	1																																	
1	2	1	0	0																																	
1	1																																				
0	-1																																				
Bias																																					
1																																					
-1	2																																				
1	4																																				

Convolution with Padding

Input (w/ padding)

Channel 0

0	0	2	0	0	0
1	2	2	1	0	0
0	0	1	2	2	0
0	0	1	0	1	0
1	2	1	0	0	0
0	0	0	0	0	0

Kernel

Channel 0

1	1
0	-1

Bias

1

Output

Channel 0

-1	2	1
1	4	3
4	2	1

Input (w/ padding)

Channel 0

0	0	0	0	0
0	1	1	0	0
0	1	1	0	0

0	1	0	0	0
0	0	0	0	0

Channel 1

0	0	0	0	0
0	2	2	0	0
0	0	2	1	0

0	1	1	1	0
0	0	0	0	0

Channel 2

0	0	0	0	0
0	2	0	1	0
0	2	2	1	0

0	2	0	2	0
0	0	0	0	0

Kernel 0

Channel 0

1	0	1
1	0	-1
-1	1	0

Channel 1

-1	-1	1
-1	0	-1
0	0	0

Channel 2

0	0	-1
0	1	1
0	0	1

Bias 0

1

Kernel 1

Channel 0

-1	0	-1
0	0	-1
-1	0	1

Channel 1

1	-1	0
0	-1	0
1	0	1

Channel 2

-1	1	-1
1	1	0
-1	1	1

Bias 1

0

Output

Channel 0

3	2	0
4	1	0
3	1	0

Channel 1

7	2	2
4	0	8
1	-2	1

Convolution layer

The **Convolution** Operation - Why?

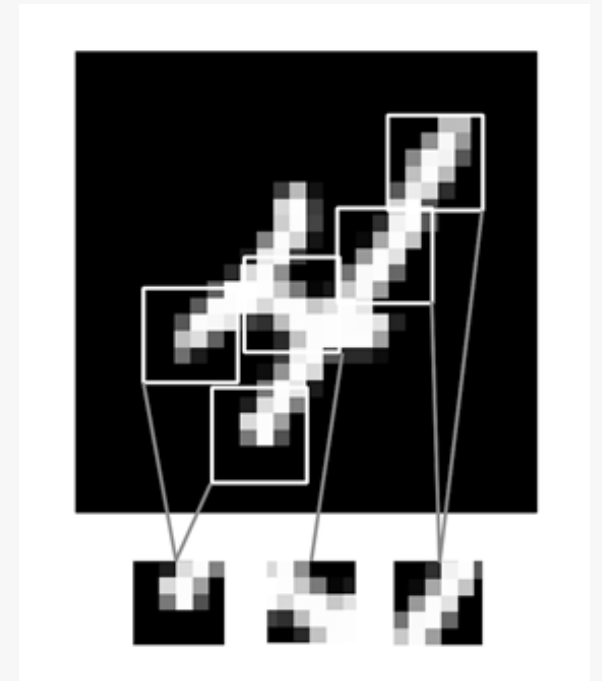
Densely connected layers learn **global** patterns in their input feature space (for example, for a MNIST digit, patterns involving all pixels), whereas **convolution layers** learn **local** patterns.

Key Convolution Properties:

1. The patterns they learn are **Translation Invariant**

For example, in the upper-left corner. A **densely** connected network would have to learn the pattern **anew** if it appeared at a new location.

After learning a certain pattern in the lower-right corner of a picture, a **convnet** can recognize it **anywhere**: this makes convnets data efficient, they need fewer training samples to learn representations that have generalization power.



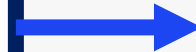
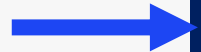
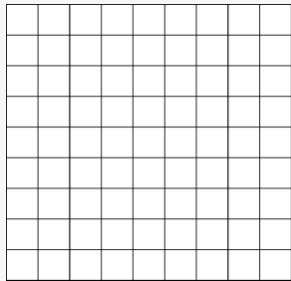
Feature Extraction & Convolution

A Case Study

A toy **ConvNet**: X's and O's

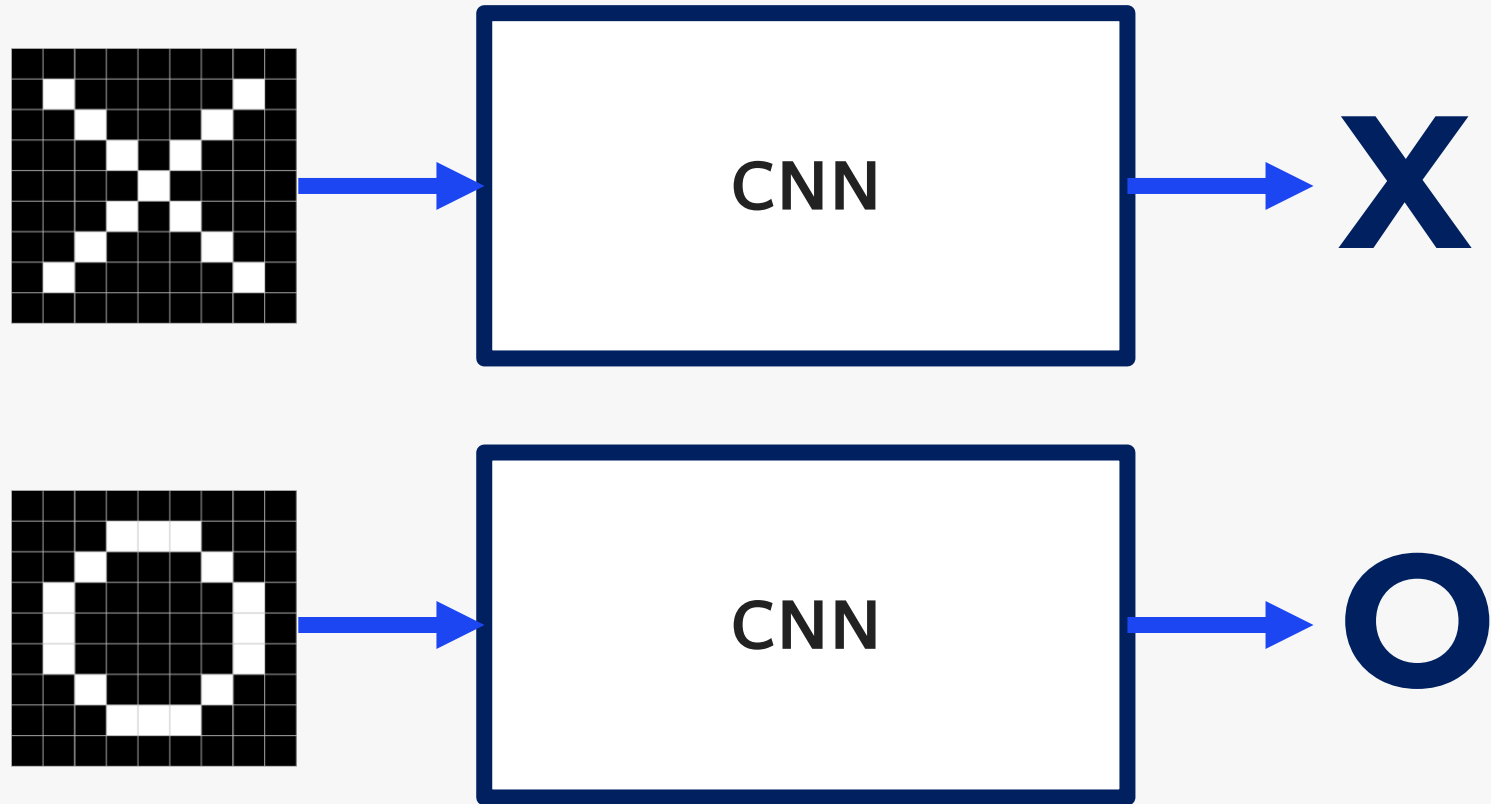
Says whether a picture is of an X or an O

A two-dimensional
array of pixels

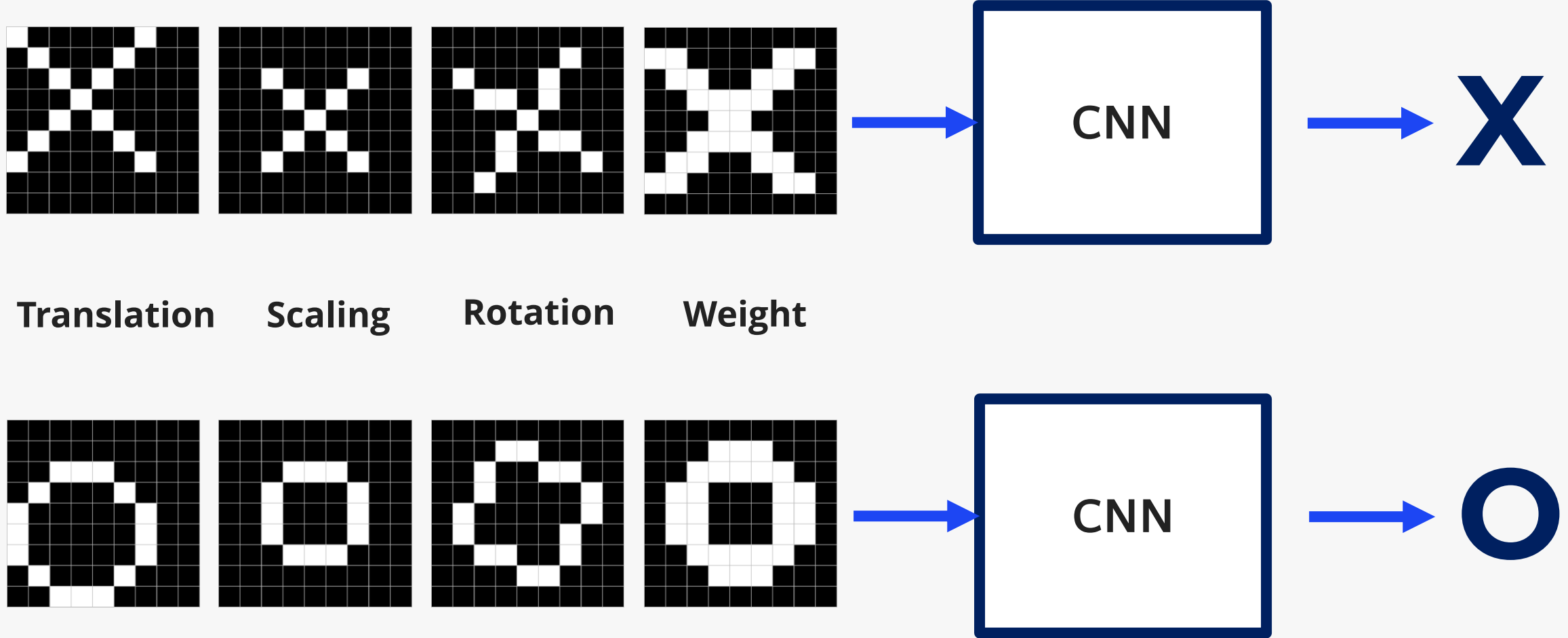


X or **O**

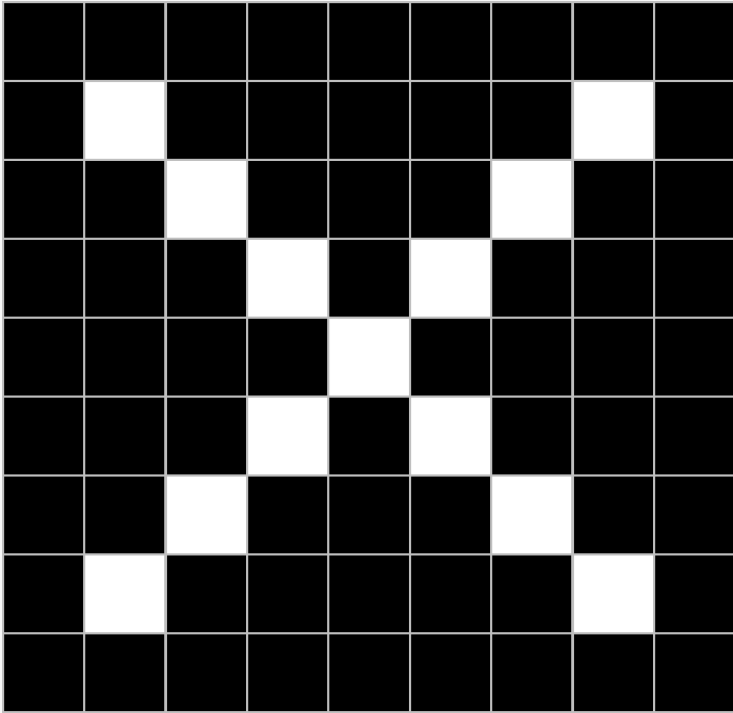
A toy **ConvNet**: X's and O's



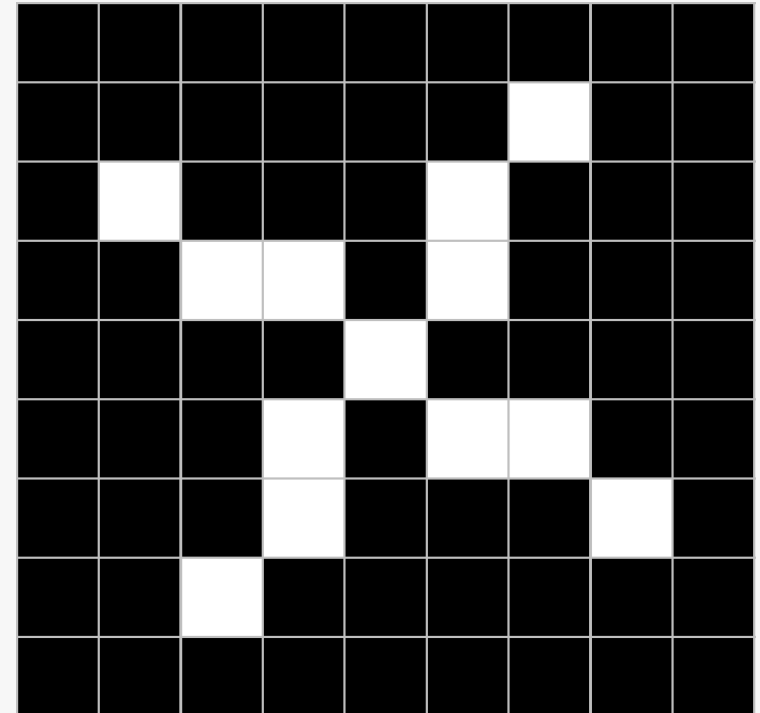
X's and O's: Trickier Cases



X or X?



?
=



What Computers See

$$=$$

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

?

=

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

What Computers See

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

?

=

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

What Computers See

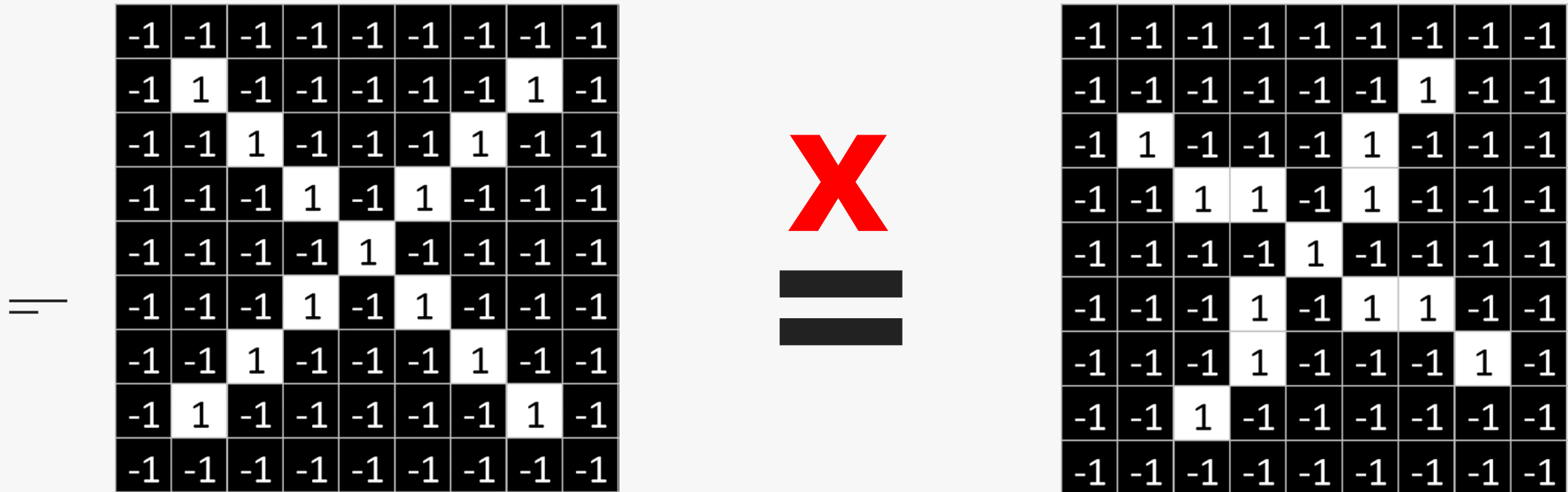
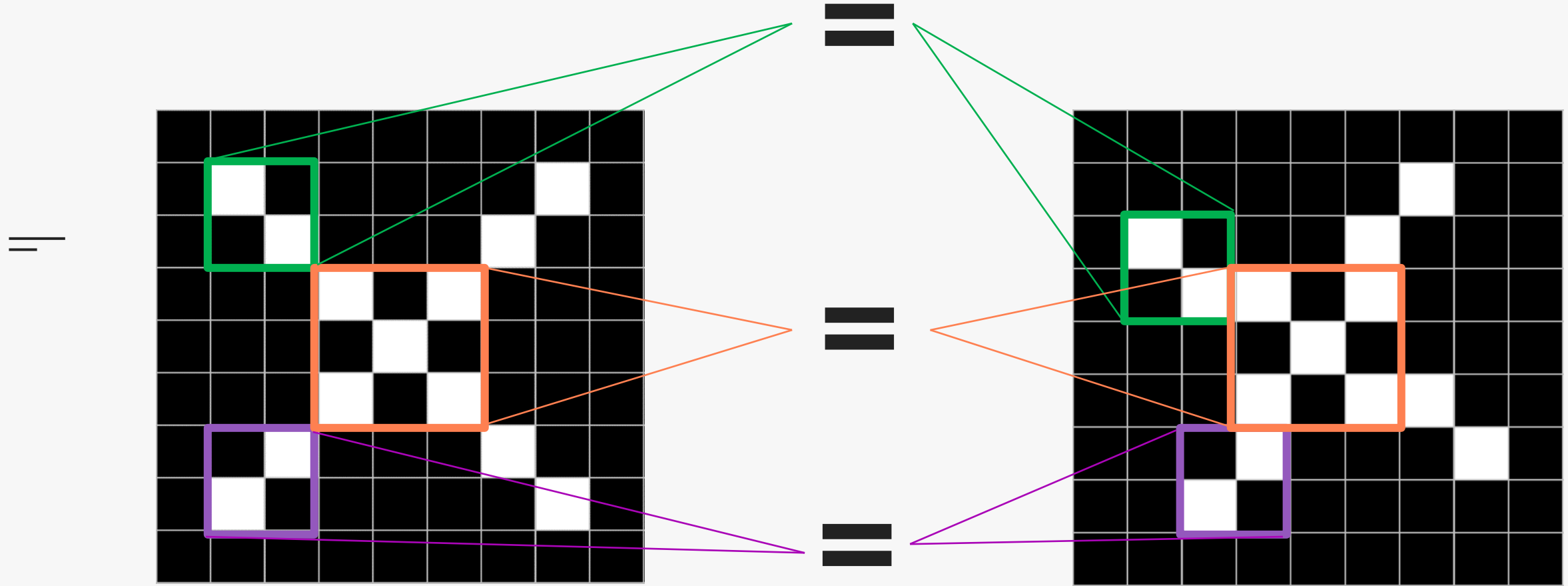


Image is represented as matrix of pixel values... and computers are **literal**!

We want to be able to classify an X as an X even if it's shifted, shrunk, rotated, deformed.

Features of X

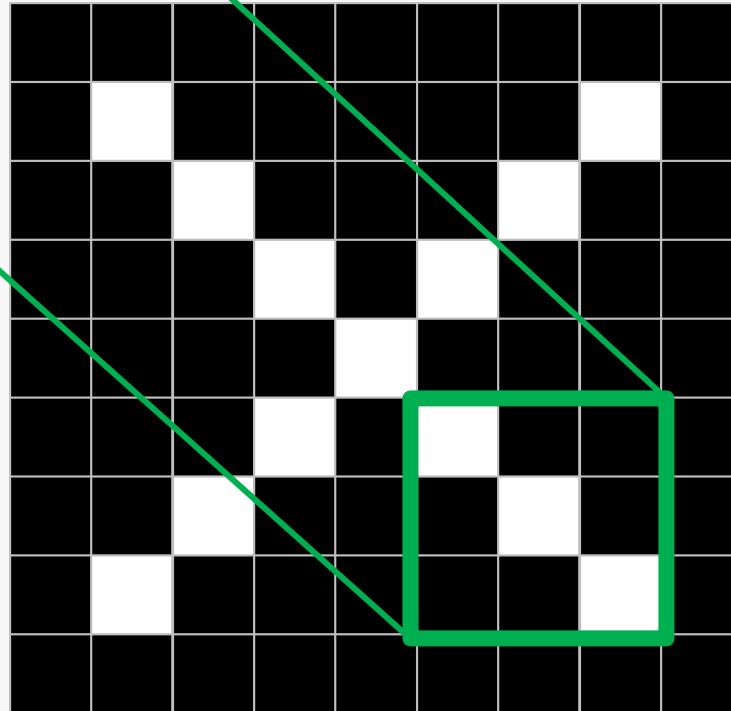


Filters to Detect X Features

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

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

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

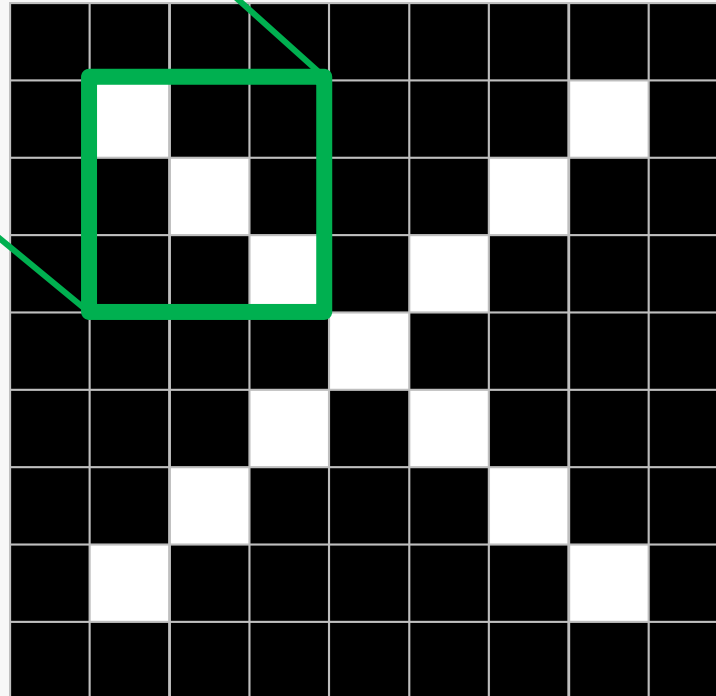


Filters to Detect X Features

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

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

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

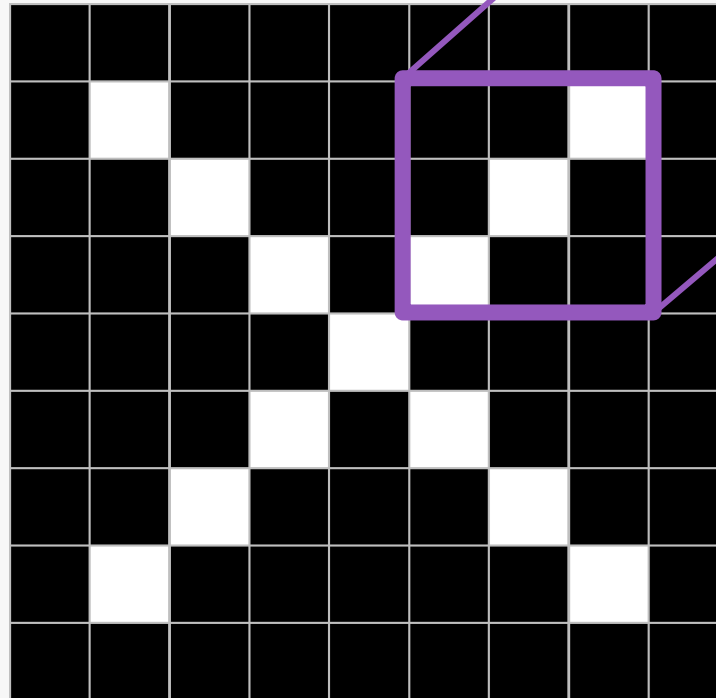


Filters to Detect X Features

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

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

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

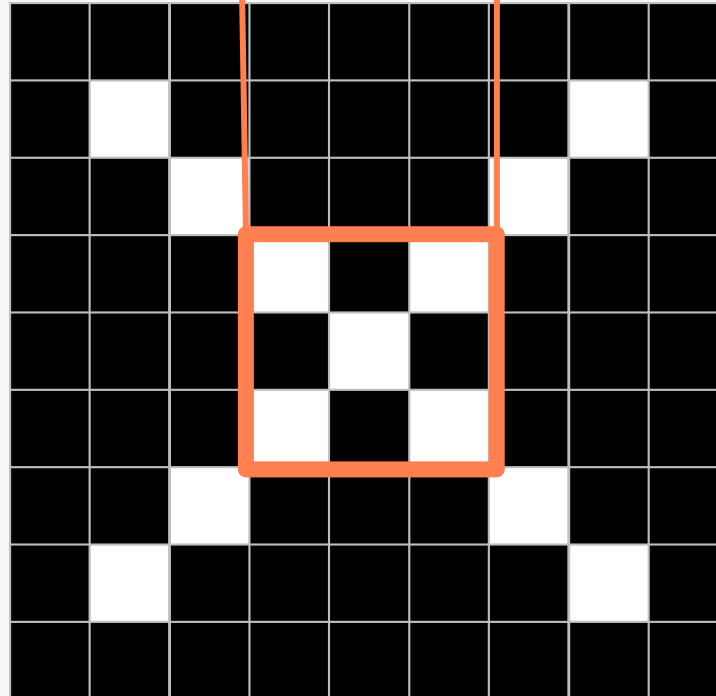


Filters to Detect X Features

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

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

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

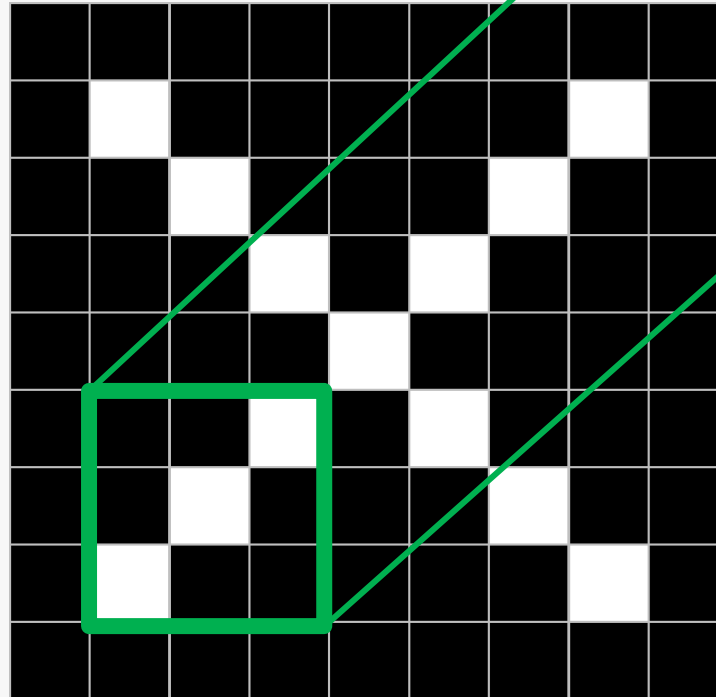


Filters to Detect X Features

1	-1	-1
-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

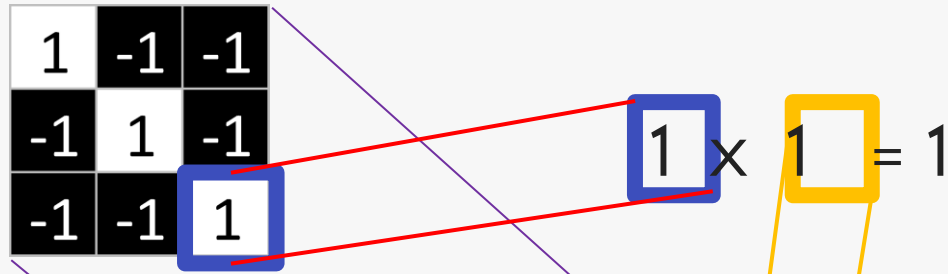
1	-1	-1
-1	1	-1
-1	-1	1

$$1 \times 1 = 1$$

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

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

1	1	1
1	1	1
1	1	1

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

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

Filtering: The math behind the match

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

1	1	-1
1	1	1
-1	1	1

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

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

			1					

Trying Every Possible Match

$$\begin{array}{c} = \\ \hline \end{array}
 \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ \hline -1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\ \hline -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\ \hline -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ \hline -1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 \\ \hline -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ \hline -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\ \hline -1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\ \hline -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ \hline \end{array}
 \otimes
 \begin{array}{|c|c|c|} \hline 1 & -1 & -1 \\ \hline -1 & 1 & -1 \\ \hline -1 & -1 & 1 \\ \hline \end{array}
 =
 \begin{array}{|c|c|c|c|c|c|c|} \hline 0.77 & -0.11 & 0.11 & 0.33 & 0.55 & -0.11 & 0.33 \\ \hline -0.11 & 1.00 & -0.11 & 0.33 & -0.11 & 0.11 & -0.11 \\ \hline 0.11 & -0.11 & 1.00 & -0.33 & 0.11 & -0.11 & 0.55 \\ \hline 0.33 & 0.33 & -0.33 & 0.55 & -0.33 & 0.33 & 0.33 \\ \hline 0.55 & -0.11 & 0.11 & -0.33 & 1.00 & -0.11 & 0.11 \\ \hline -0.11 & 0.11 & -0.11 & 0.33 & -0.11 & 1.00 & -0.11 \\ \hline 0.33 & -0.11 & 0.55 & 0.33 & 0.11 & -0.11 & 0.77 \\ \hline \end{array}$$

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



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

=

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

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



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

=

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

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



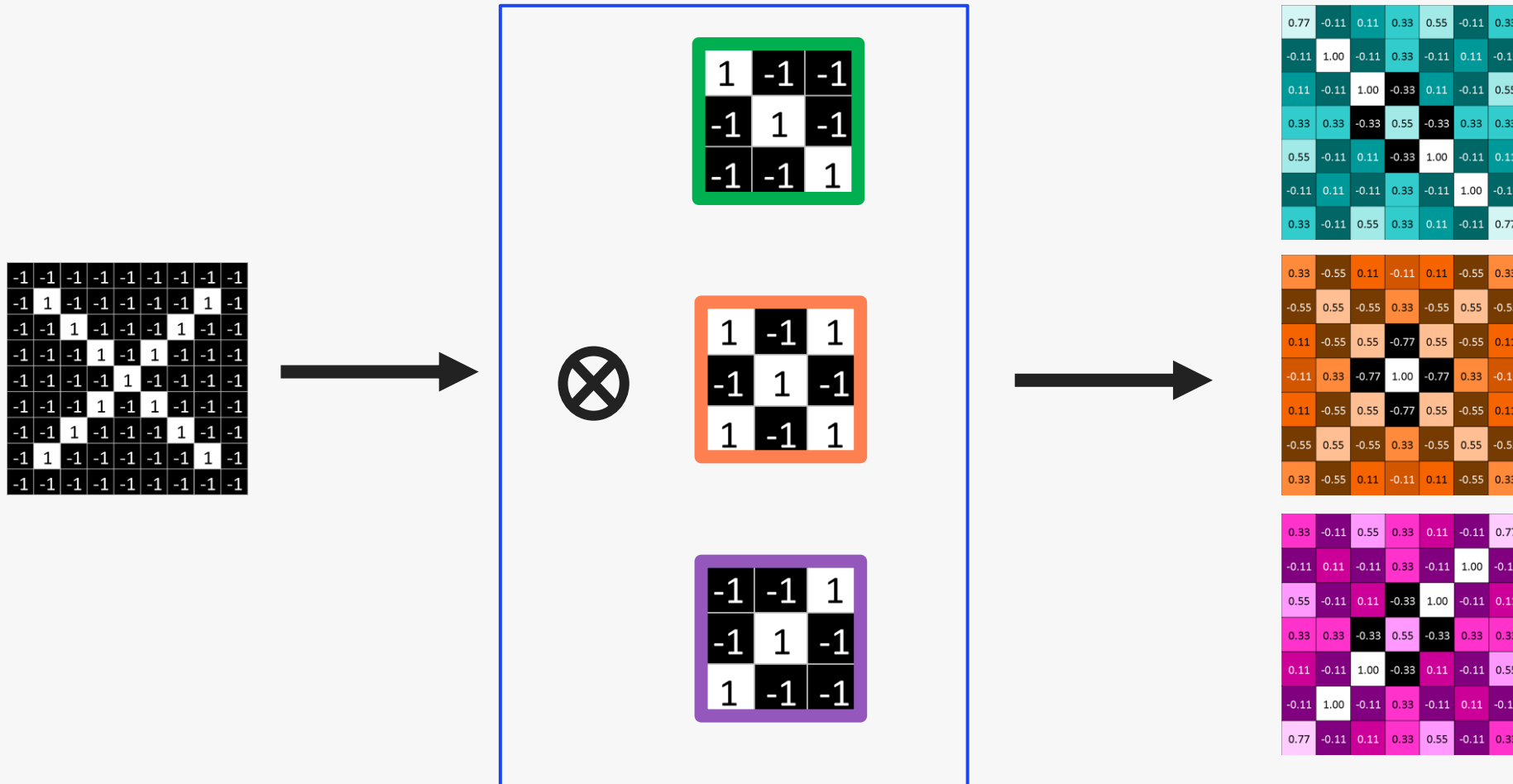
-1	-1	1
-1	1	-1
1	-1	-1

=

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

Convolution layer

One image becomes a stack of filtered images



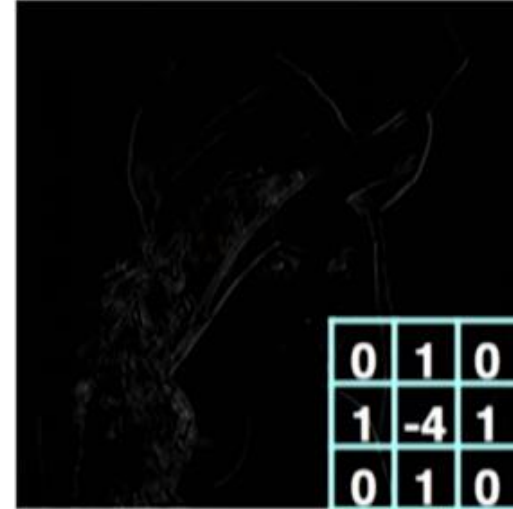
Producing Feature Maps



Original



Sharpen



Edge Detect



"Strong" Edge
Detect

The **Convolution** Operation - Why?

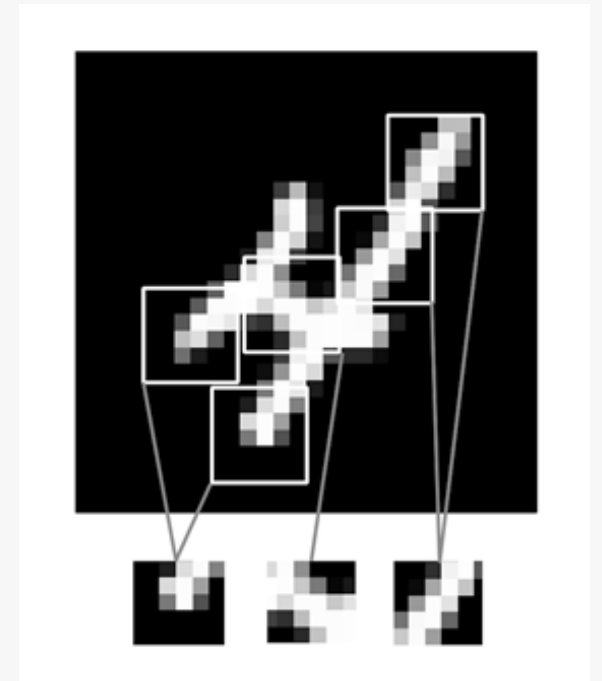
Densely connected layers learn **global** patterns in their input feature space (for example, for a MNIST digit, patterns involving all pixels), whereas **convolution layers** learn **local** patterns.

Key Convolution Properties:

1. The patterns they learn are **Translation Invariant**

For example, in the upper-left corner. A **densely** connected network would have to learn the pattern **anew** if it appeared at a new location.

After learning a certain pattern in the lower-right corner of a picture, a **convnet** can recognize it **anywhere**: this makes convnets data efficient, they need fewer training samples to learn representations that have generalization power.

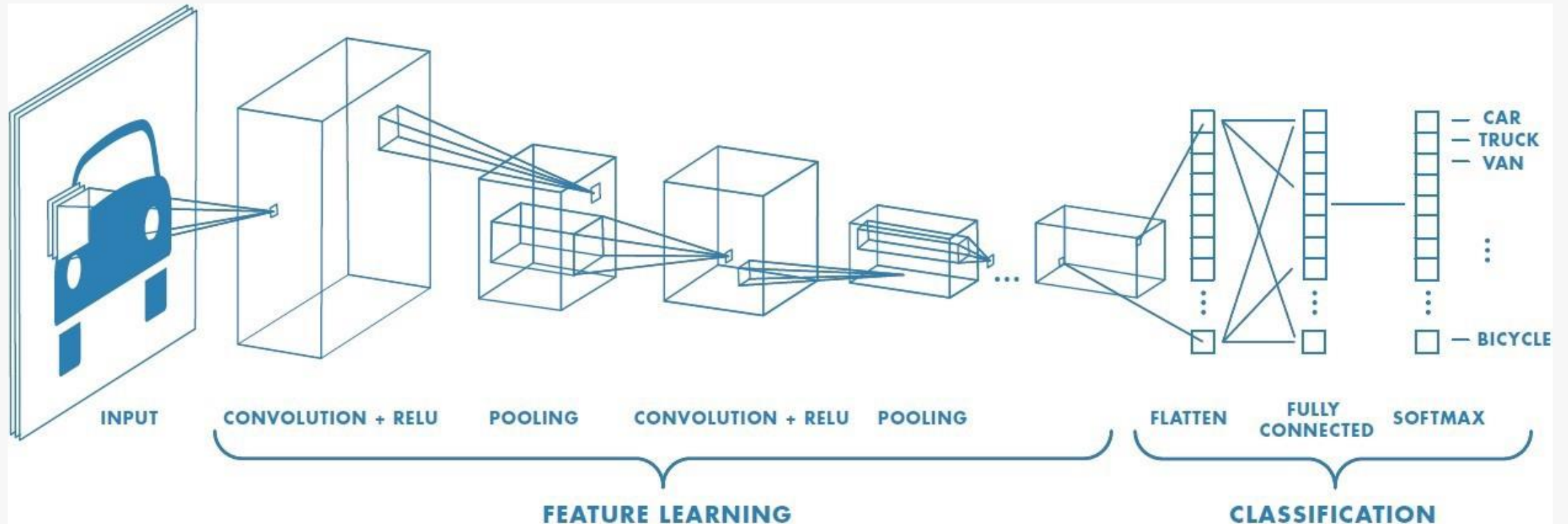




Convolutional Neural Networks

CNNs

CNNs for Classification

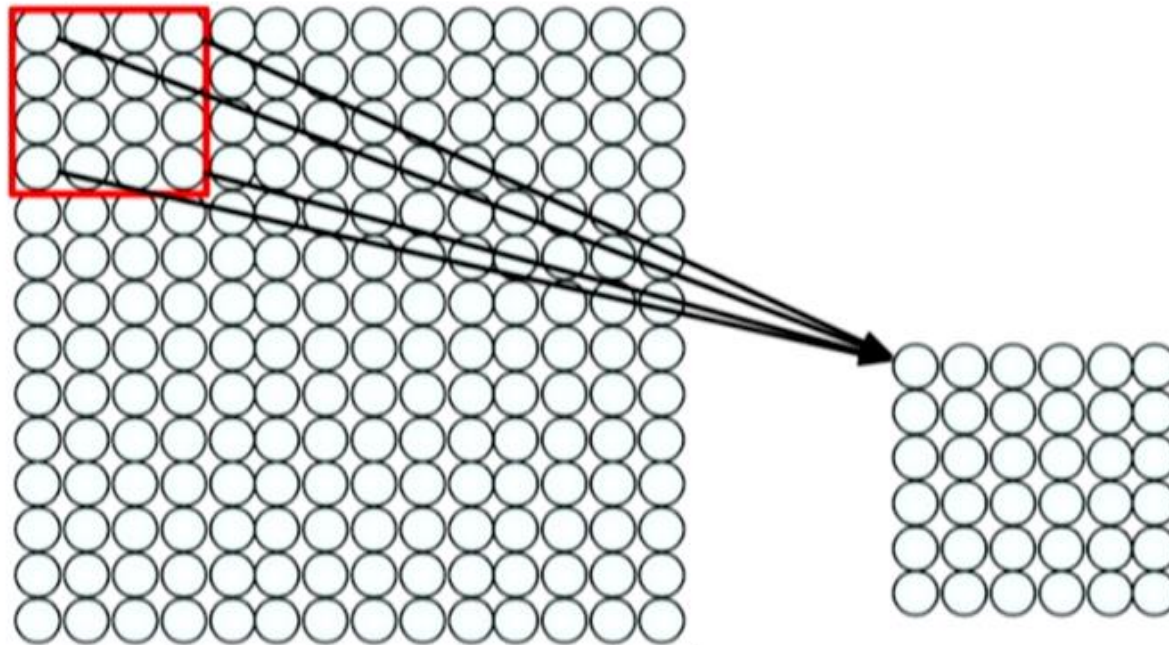


1. **Convolution:** Apply filters to generate feature maps.
2. **Non-linearity:** Often ReLU.
3. **Pooling:** Down-sampling operation on each feature map.

Train the model with image data.

Learn weights of filters in convolution layers.

Convolutional layer: Local Connectivity



 `tf.keras.layers.Conv2D`

For a neuron in hidden layer:

- Take inputs from patch
- Compute weighted sum
- Apply bias

4x4 filter: matrix
of weights w_{ij}

$$\sum_{i=1}^4 \sum_{j=1}^4 w_{ij} x_{i+p,j+q} + b$$

for neuron (p,q) in hidden layer

- 1) applying a window of weights
- 2) computing linear combinations
- 3) activating with non-linear function

Input (w/ padding)

Channel 0

0	0	0	0	0
0	1	1	0	0
0	1	1	0	0

0	1	0	0	0
0	0	0	0	0

Channel 1

0	0	0	0	0
0	2	2	0	0
0	0	2	1	0

0	1	1	1	0
0	0	0	0	0

Channel 2

0	0	0	0	0
0	2	0	1	0
0	2	2	1	0

0	2	0	2	0
0	0	0	0	0

Kernel 0

Channel 0

1	0	1
1	0	-1
-1	1	0

Channel 1

-1	-1	1
-1	0	-1
0	0	0

Channel 2

0	0	-1
0	1	1
0	0	1

Bias 0

1

Kernel 1

Channel 0

-1	0	-1
0	0	-1
-1	0	1

Channel 1

1	-1	0
0	-1	0
1	0	1

Channel 2

-1	1	-1
1	1	0
-1	1	1

Bias 1

0

Output

Channel 0

3	2	0
4	1	0
3	1	0

Channel 1

7	2	2
4	0	8
1	-2	1

Multiple Filters

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1



308

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2



-498

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



164

+

+

+ 1 = -25

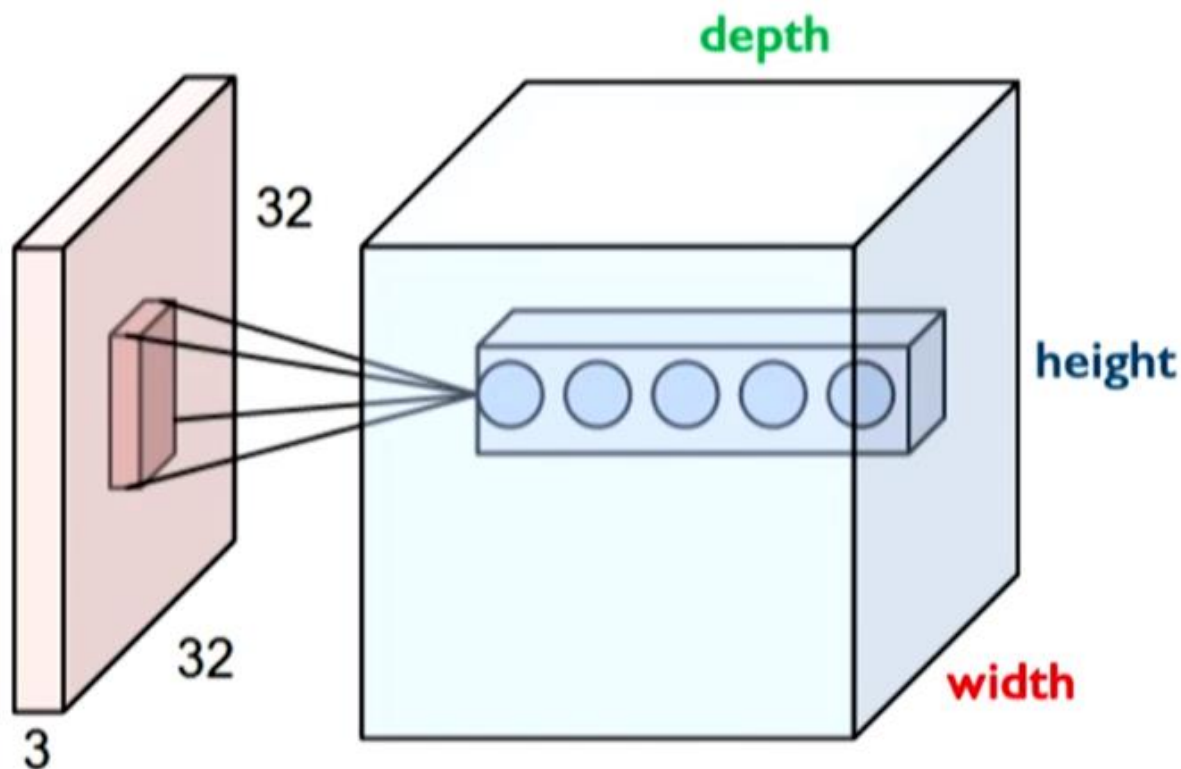


Bias = 1

Multiple Filters Colored Image

-25				...
				...
				...
				...
...

CNNs: Spatial Arrangement of Output Volume



Layer Dimensions:

$h \times w \times d$

where h and w are spatial dimensions
 d (depth) = number of filters

Stride:

Filter step size

Receptive Field:

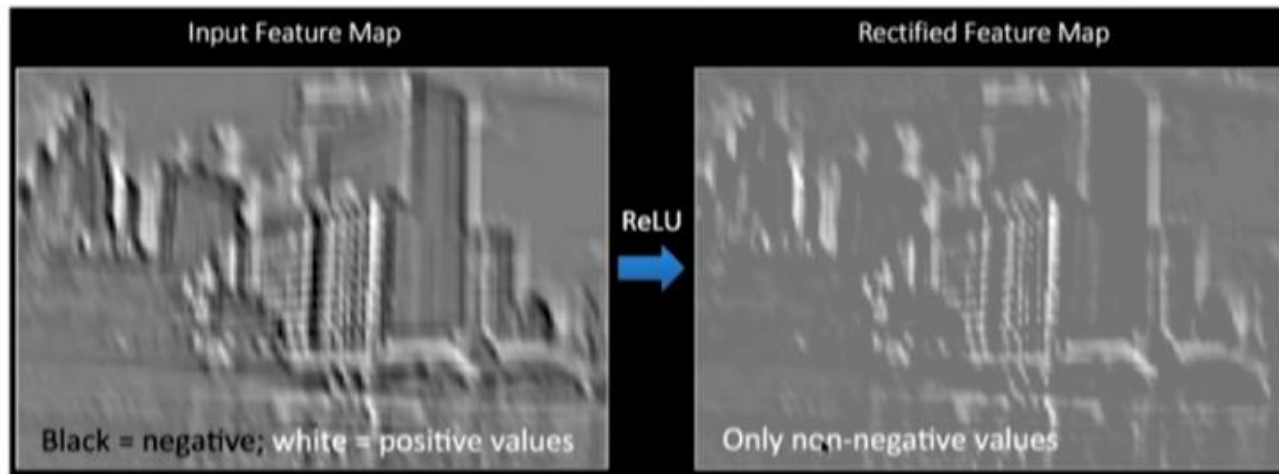
Locations in input image that
a node is path connected to



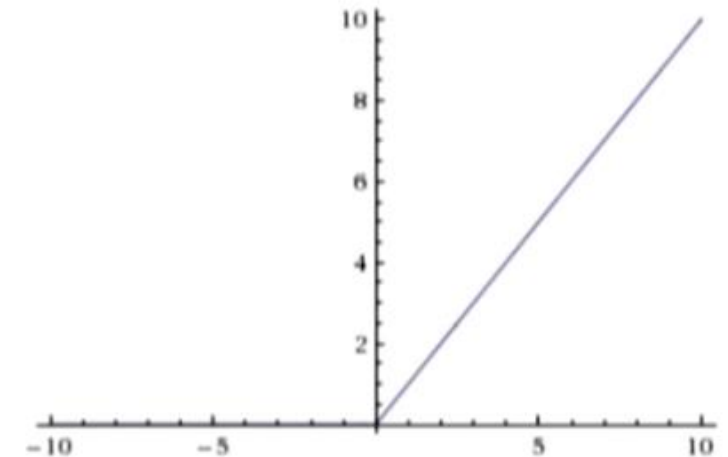
```
tf.keras.layers.Conv2D( filters=d, kernel_size=(h,w), strides=s )
```

CNNs: Introducing Non-Linearity

- Apply after every convolution operation (i.e., after convolutional layers)
- ReLU: pixel-by-pixel operation that replaces all negative values by zero. **Non-linear operation!**

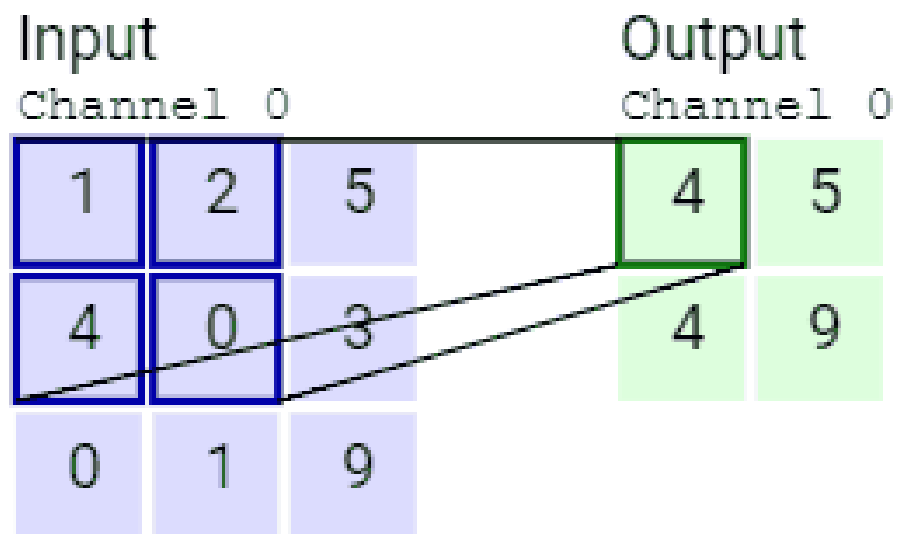


Rectified Linear Unit (ReLU)

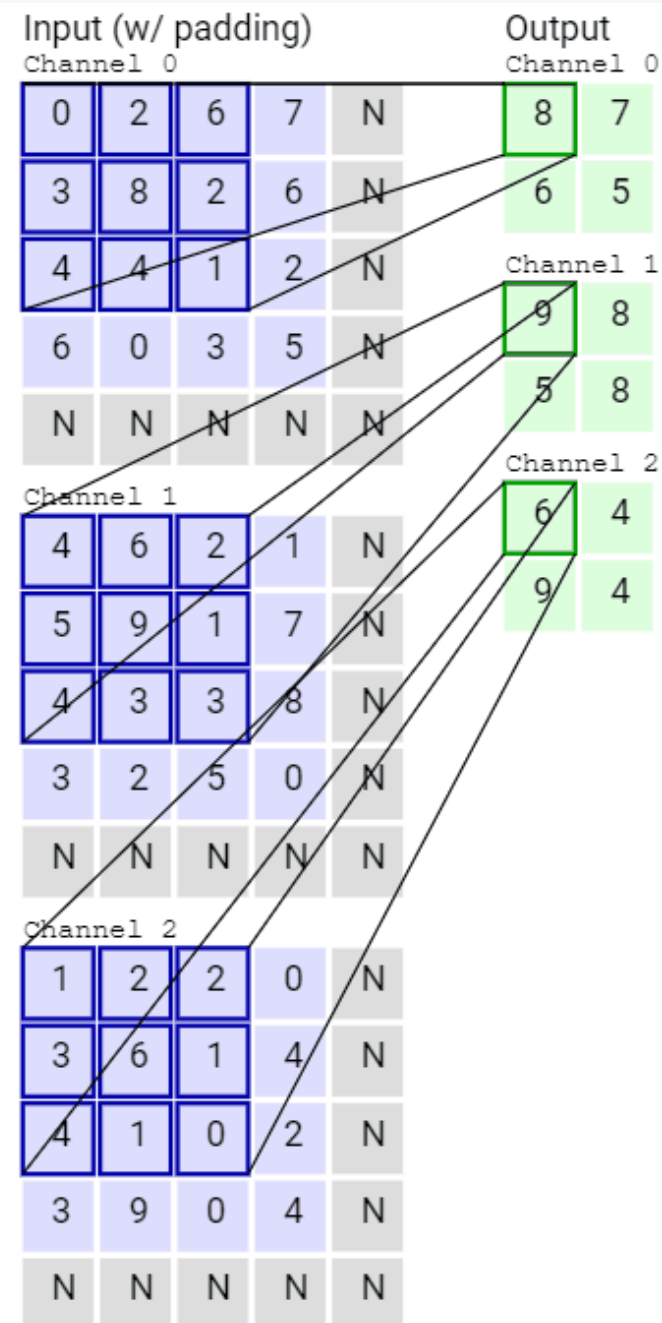


 `tf.keras.layers.ReLU`

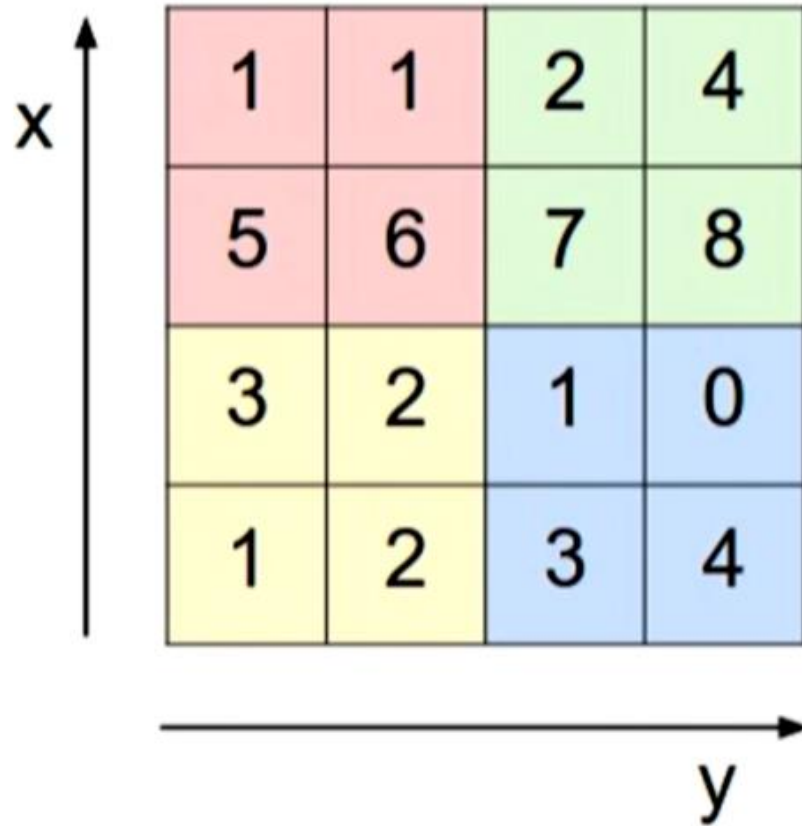
CNNs: Pooling



Max Pooling



CNNs: Pooling



max pool with 2x2 filters
and stride 2

```
tf.keras.layers.MaxPool2D(  
    pool_size=(2,2),  
    strides=2  
)
```



- 1) Reduced dimensionality
- 2) Spatial invariance

How else can we down-sample and preserve spatial invariance?

Back to The Case Study

Pooling

Maximum

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

1.00			

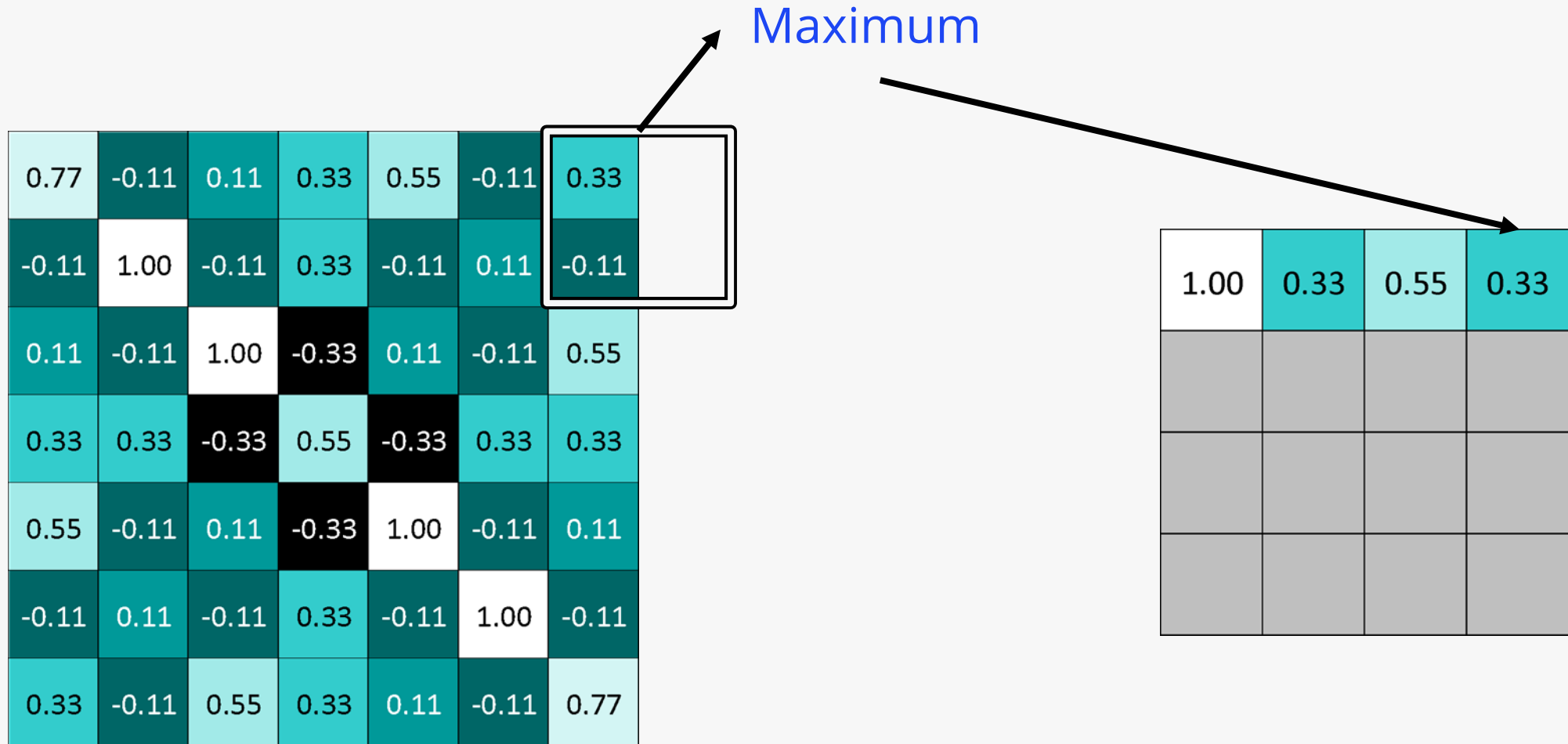
Pooling

Maximum

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

1.00	0.33		

Pooling



Pooling

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

Max Pooling

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

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



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

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



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

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



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

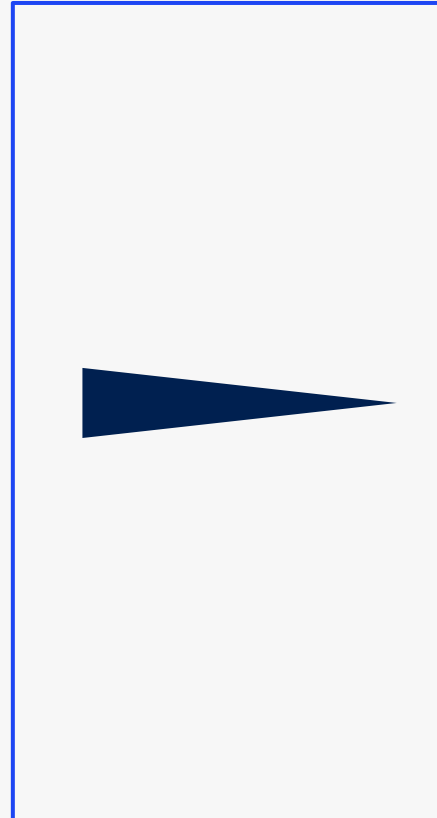
Pooling Layer

A stack of images becomes a stack of smaller images.

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

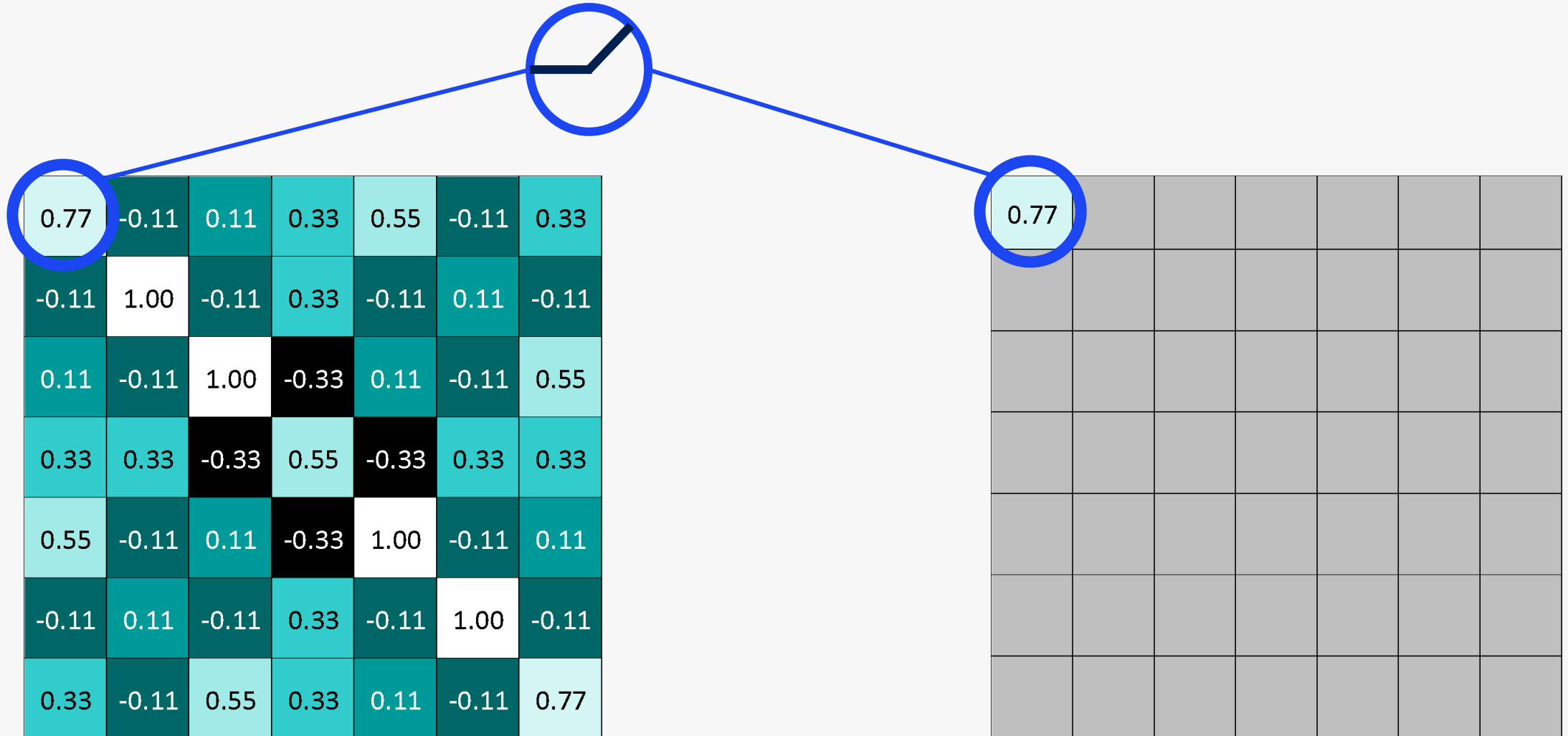


1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

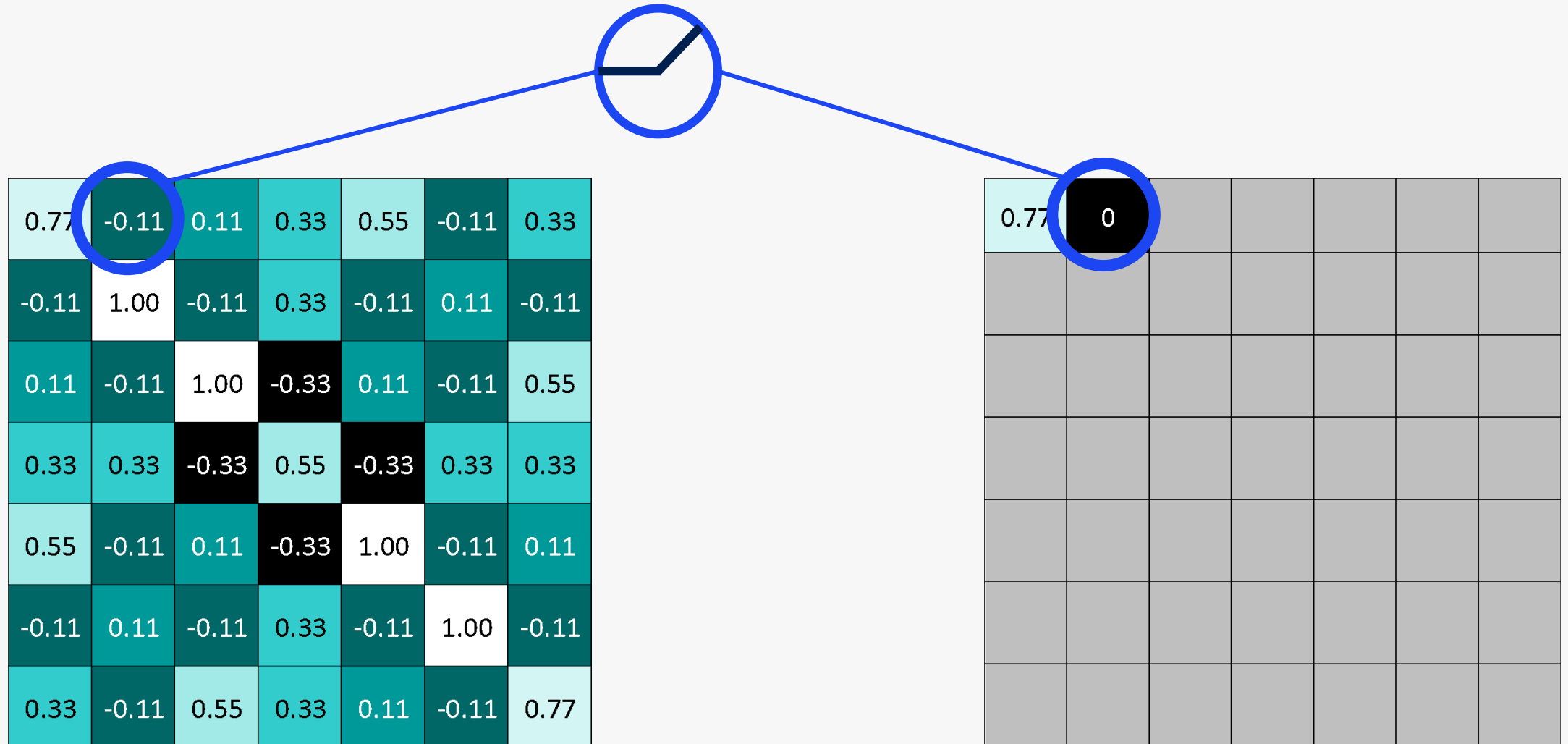
0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

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

Rectified Linear Units (ReLU)



Rectified Linear Units (ReLU)



Rectified Linear Units (ReLU)

=

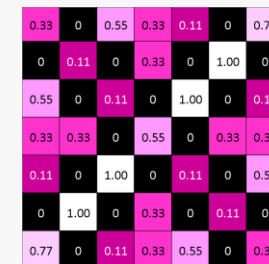
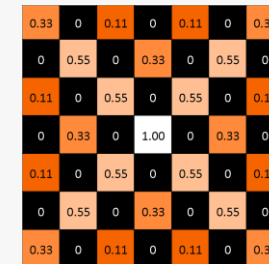
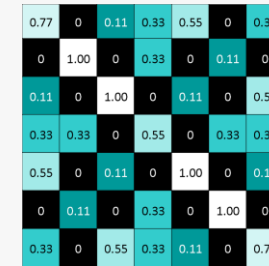
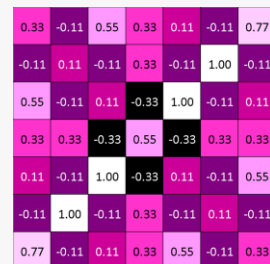
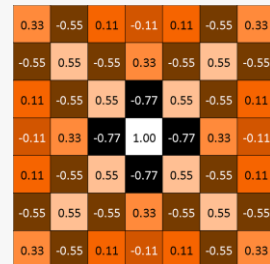
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77



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

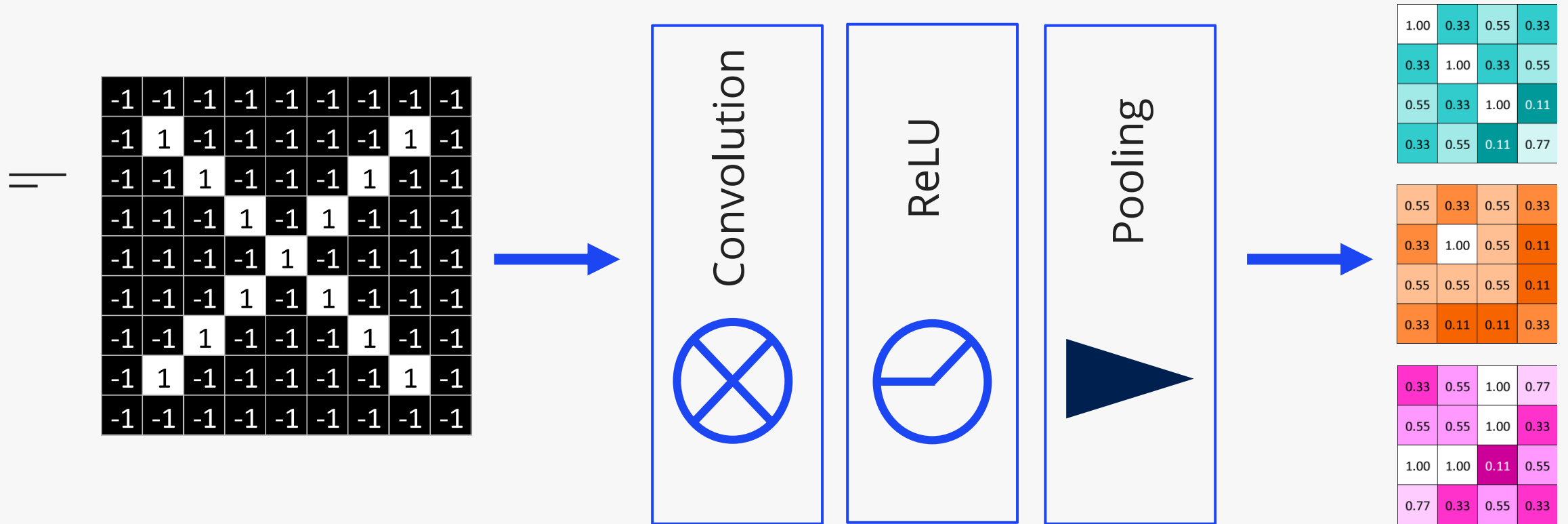
ReLU Layer

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



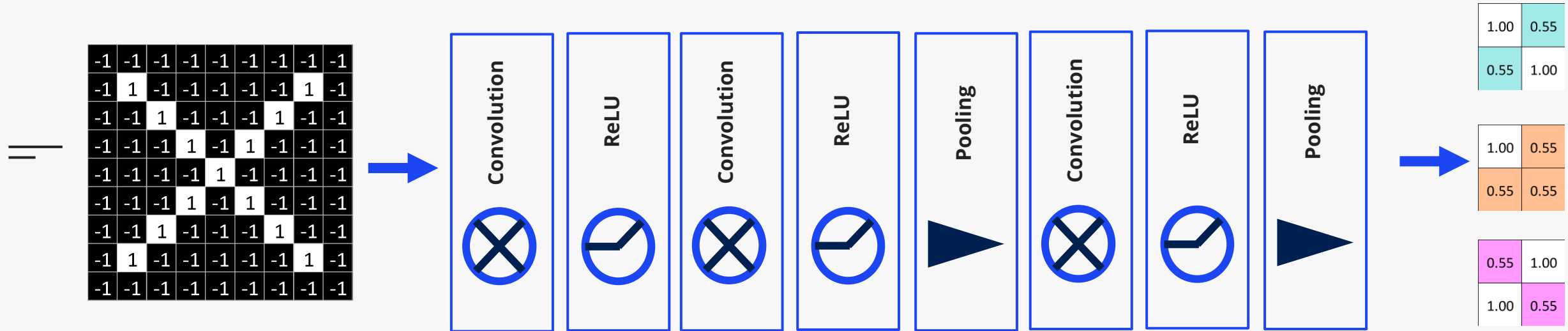
Layers Get **Stacked**

The output of one becomes the input of the next.

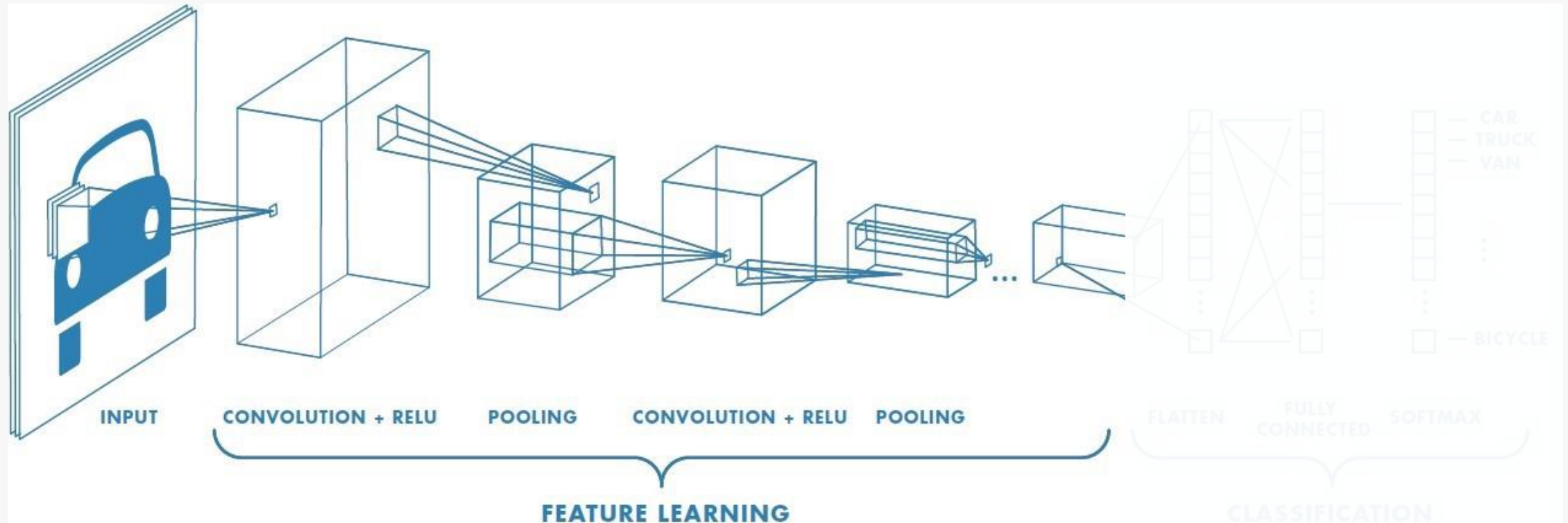


Deep Stacking

Layers can be repeated several (or many) times.

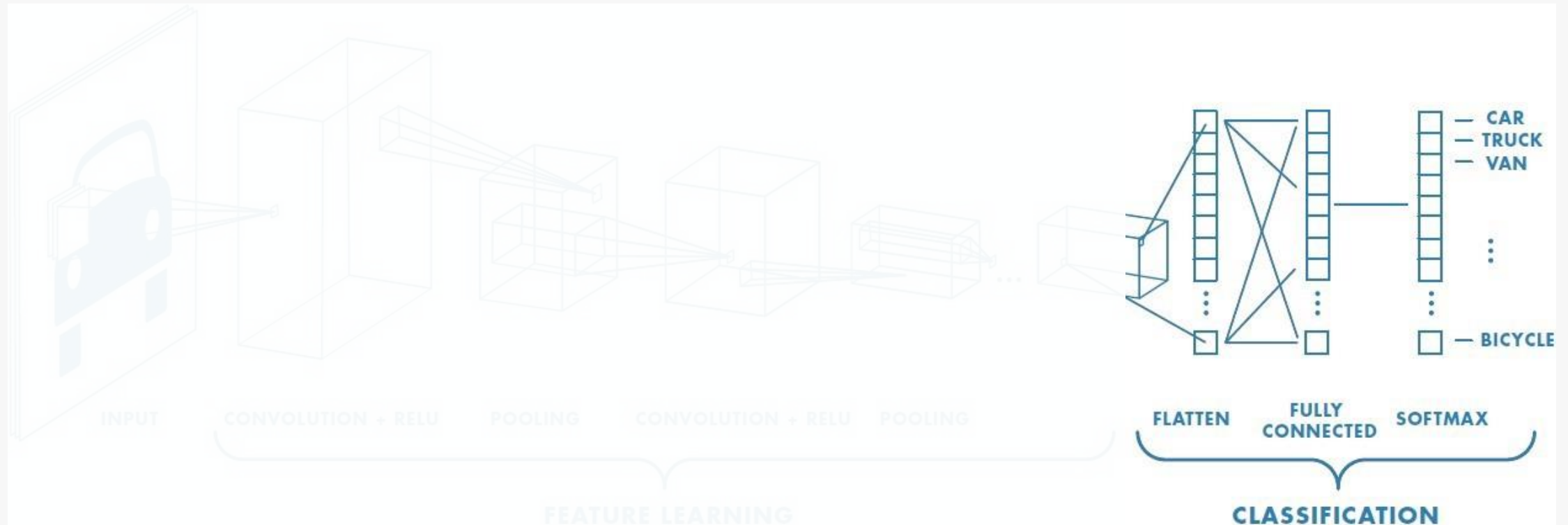


CNNs for Classification: Feature Learning



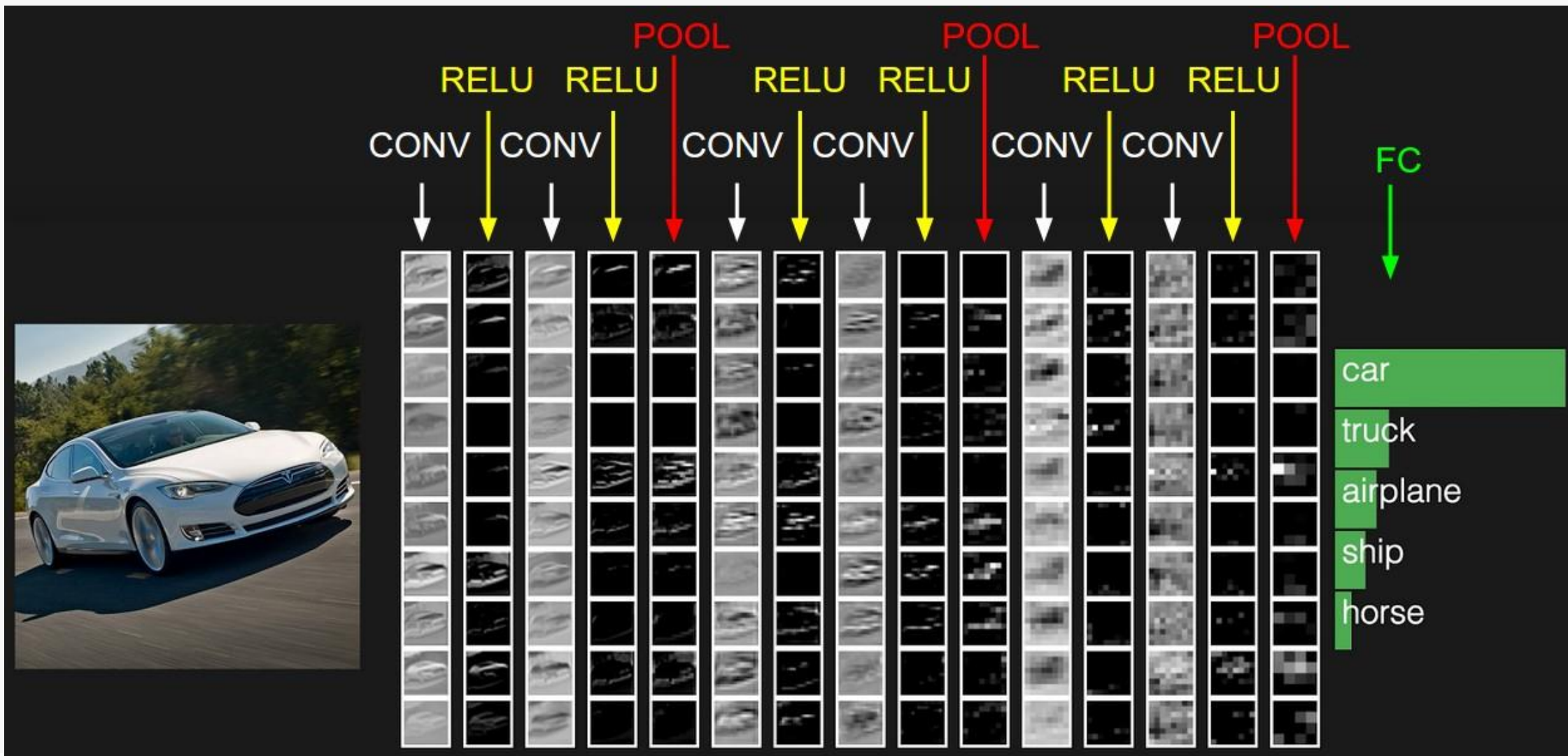
1. Learn features in input image through **convolution**.
2. Introduce **Non-linearity** through activation function (real-world data is non-linear).
3. Reduce dimensionality and preserve spatial invariance with **Pooling**.

CNNs for Classification: **Class Probabilities**



1. CONV and Pool layers output high-level feature of input
2. Fully connected layer uses these features for classifying input image.
3. Express output as **Probability** of image belonging to a particular class.

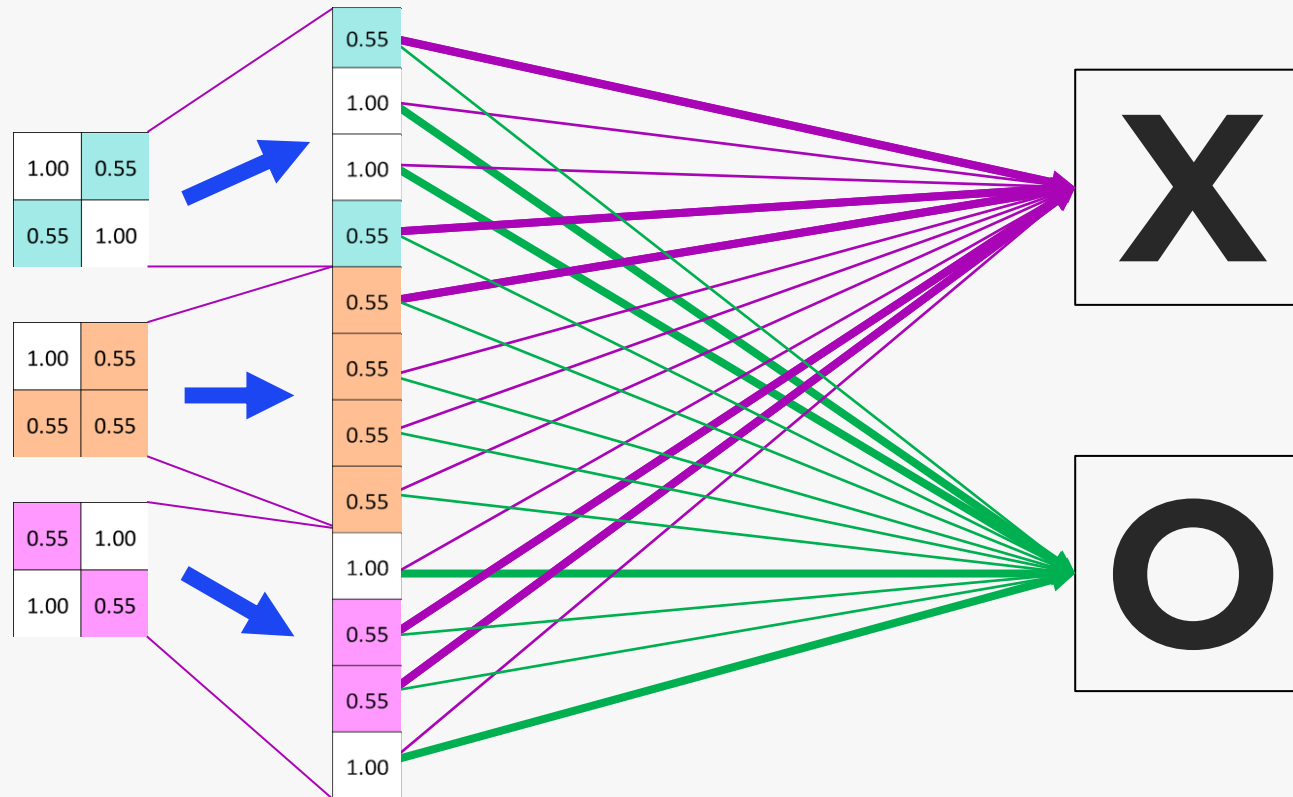
CNNs: Example



**Back to
The Case Study again!**

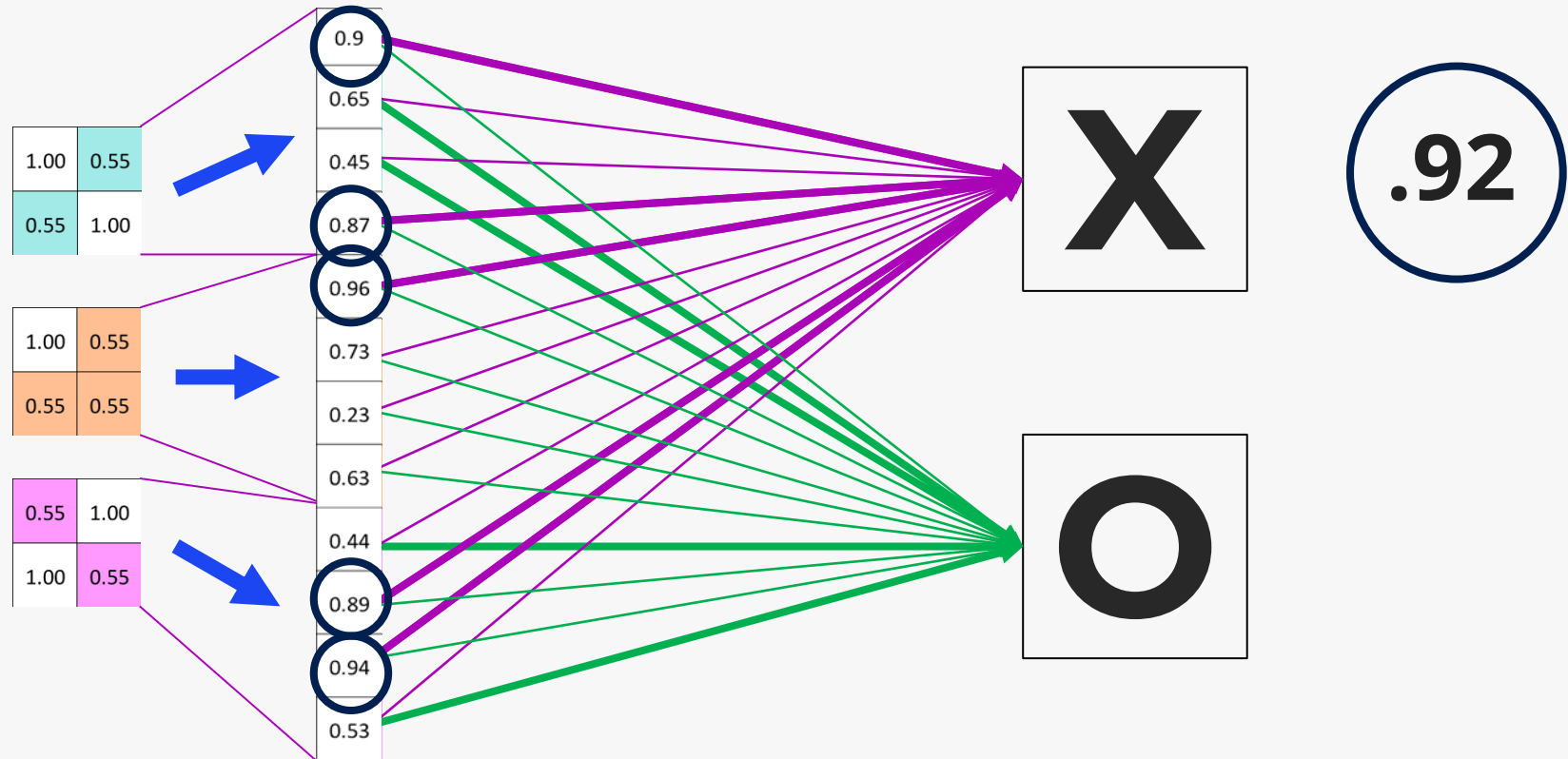
Fully Connected layer

Every value gets a vote depends on how strongly a value predicts X or O



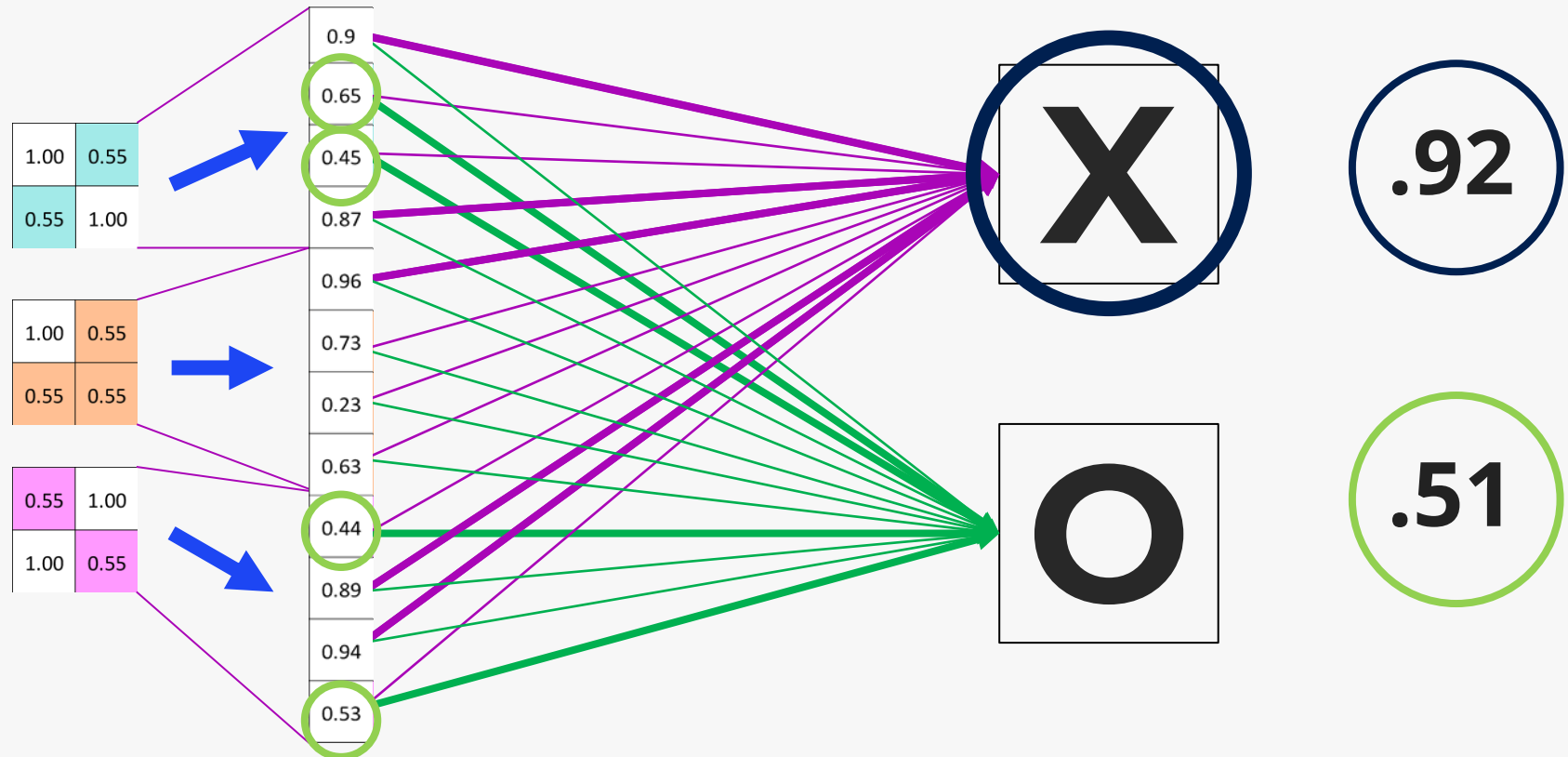
Fully Connected layer

Vote Values on X or O



Fully Connected layer

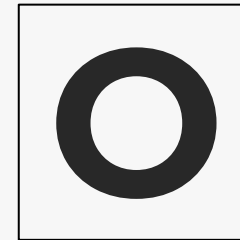
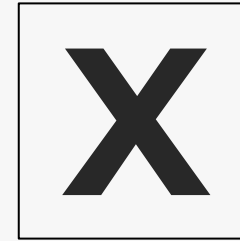
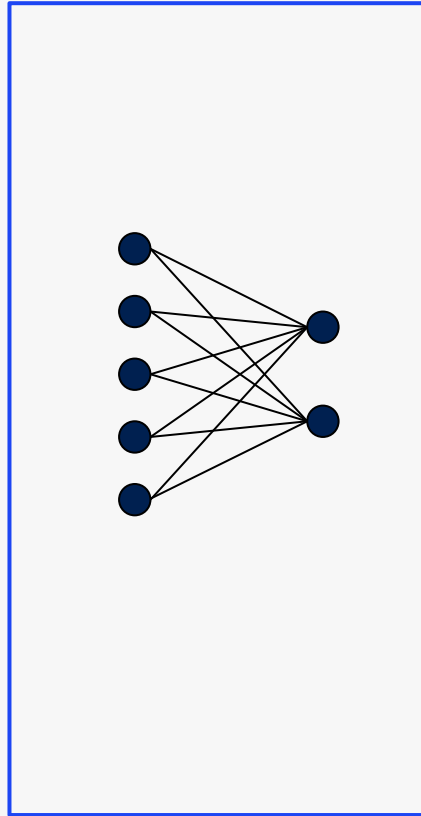
Vote Values on X or O



Fully Connected layer

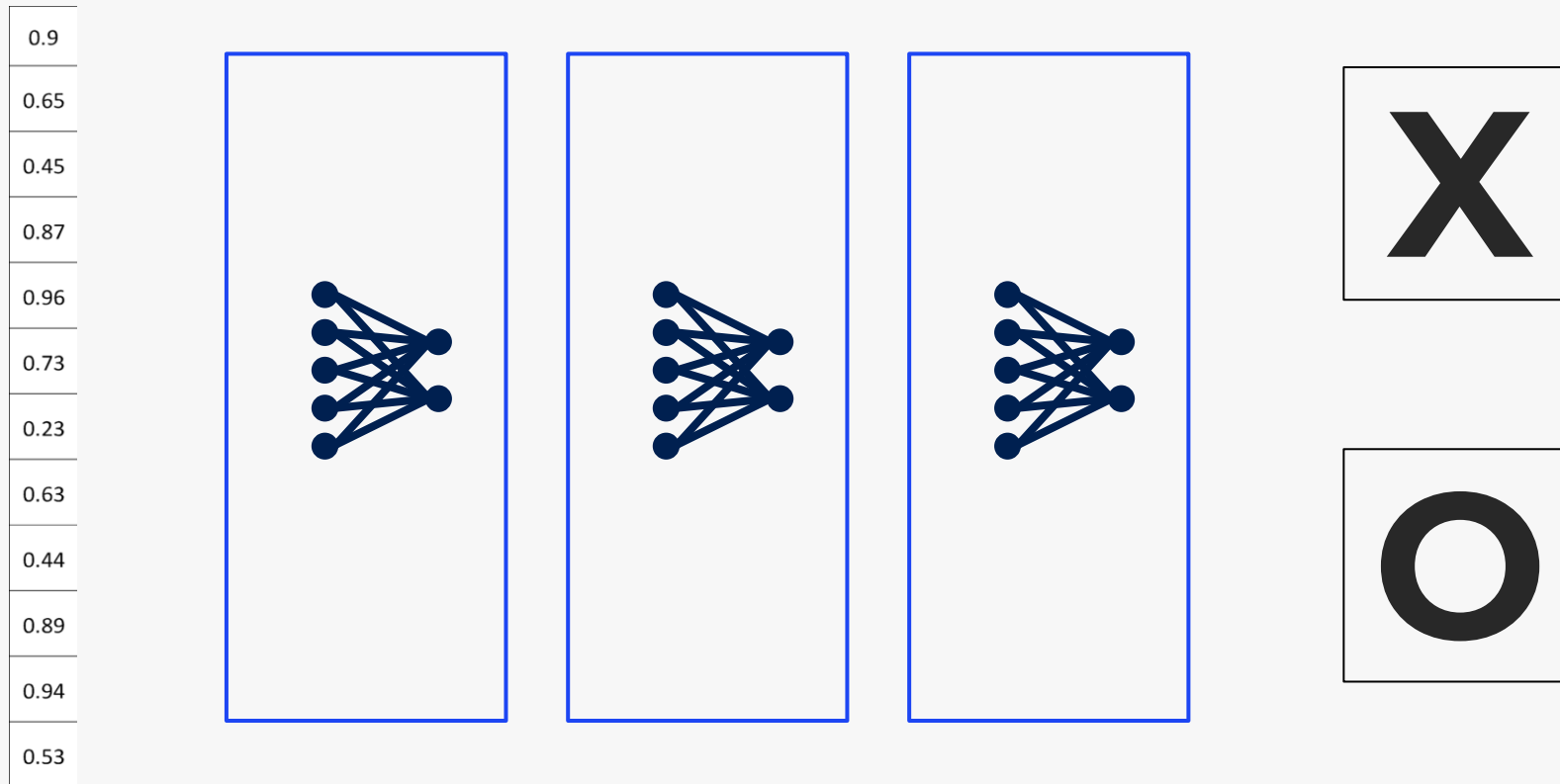
A list of feature values becomes a list of votes.

0.9
0.65
0.45
0.87
0.96
0.73
0.23
0.63
0.44
0.89
0.94
0.53



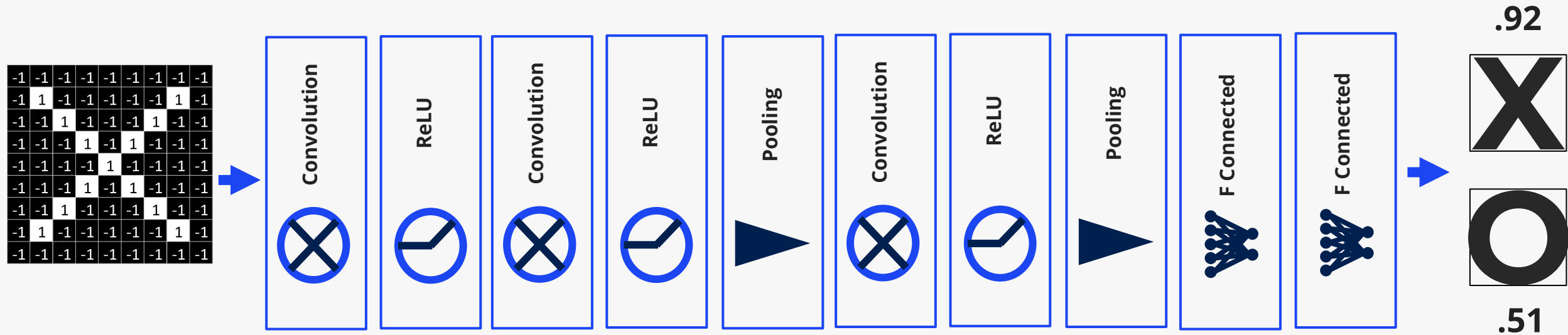
Fully Connected layer

These can also be stacked.



Backprop

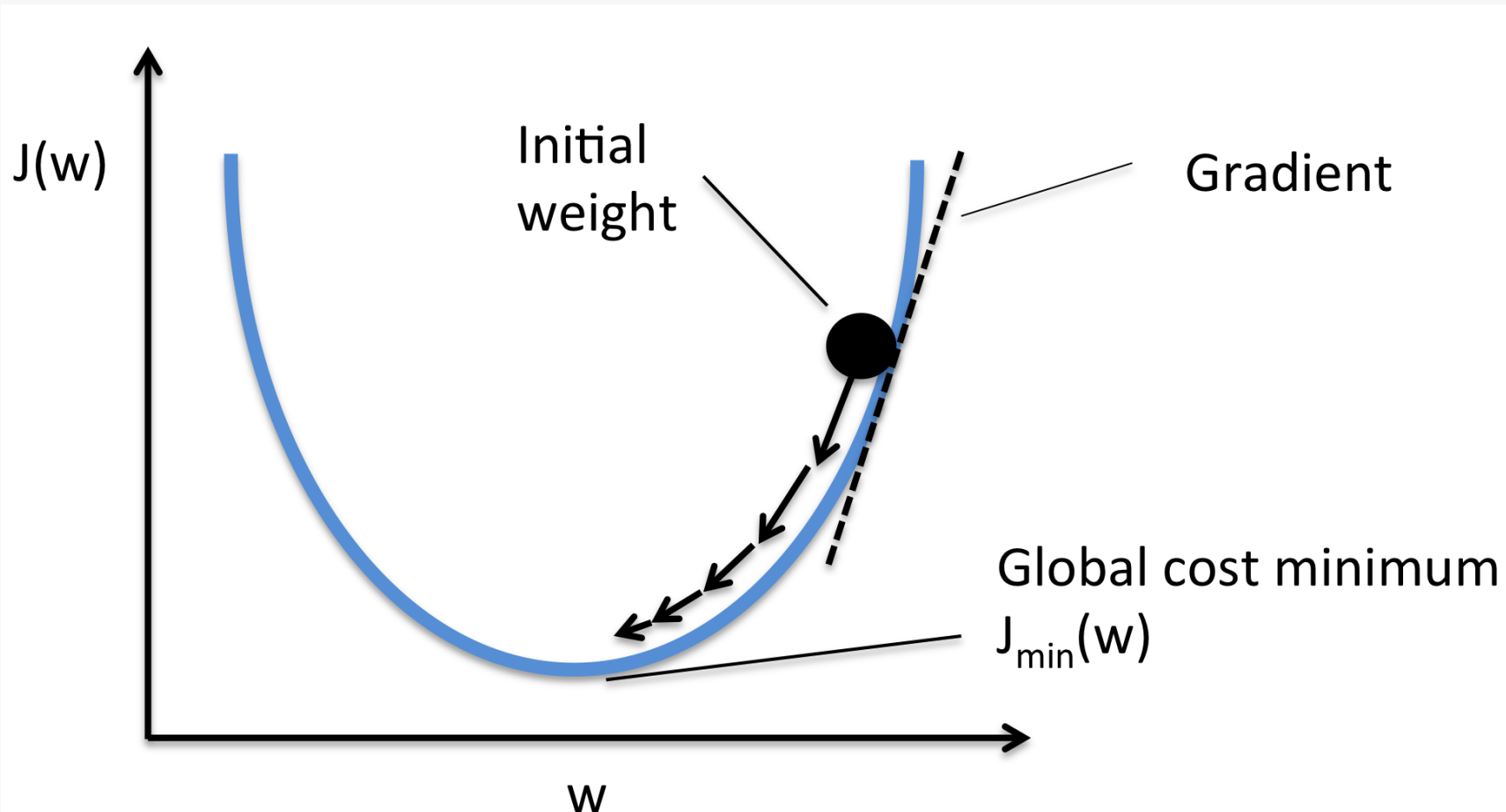
Error = right answer – actual answer



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

Gradient Descent

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



Putting it all together

```
import tensorflow as tf

def generate_model():
    model = tf.keras.Sequential([
        # first convolutional layer
        tf.keras.layers.Conv2D(32, filter_size=3, activation='relu'),
        tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

        # second convolutional layer
        tf.keras.layers.Conv2D(64, filter_size=3, activation='relu'),
        tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

        # fully connected classifier
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(1024, activation='relu'),
        tf.keras.layers.Dense(10, activation='softmax') # 10 outputs
    ])
    return model
```

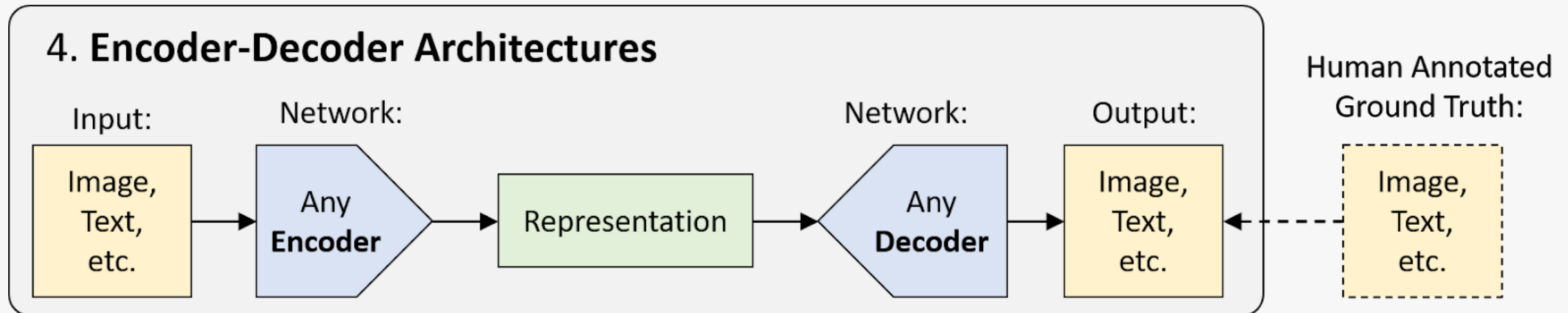


Encoder-Decoder Architectures (Pattern)

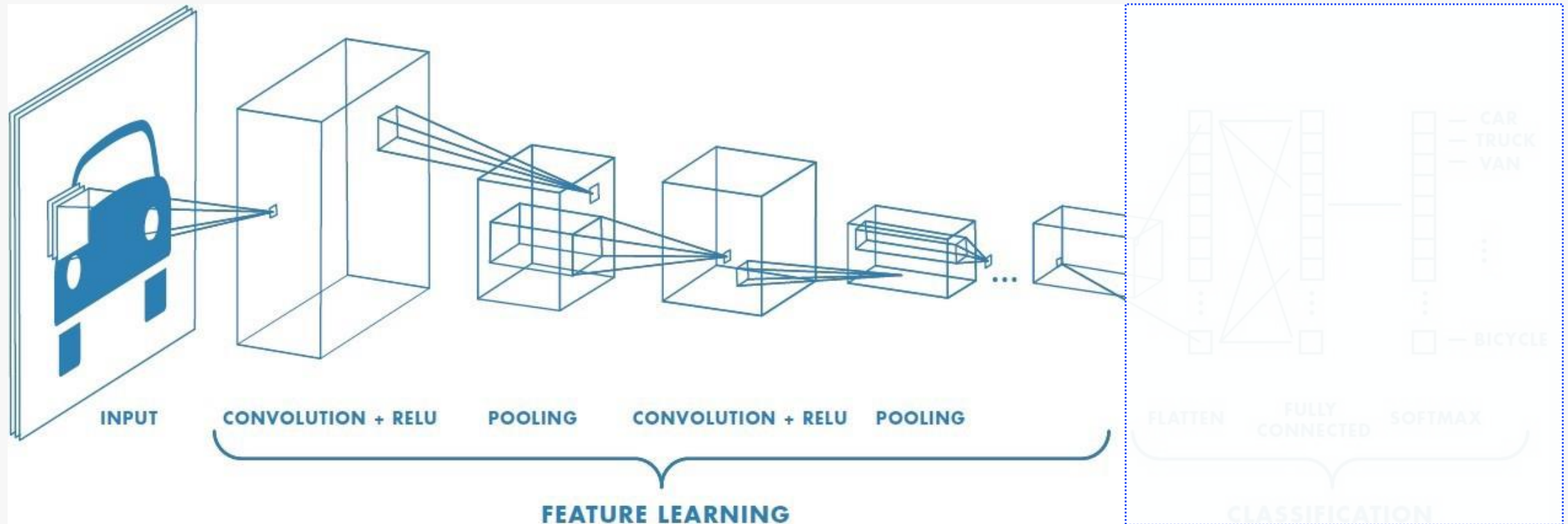
At a high-level, neural networks are either **encoders**, **decoders**, or a combination of **both**:

- **Encoders** find patterns in raw data to form compact, useful representations.
- **Decoders** generate high-resolution data from those representations. The generated data is either new examples or descriptive knowledge.

==

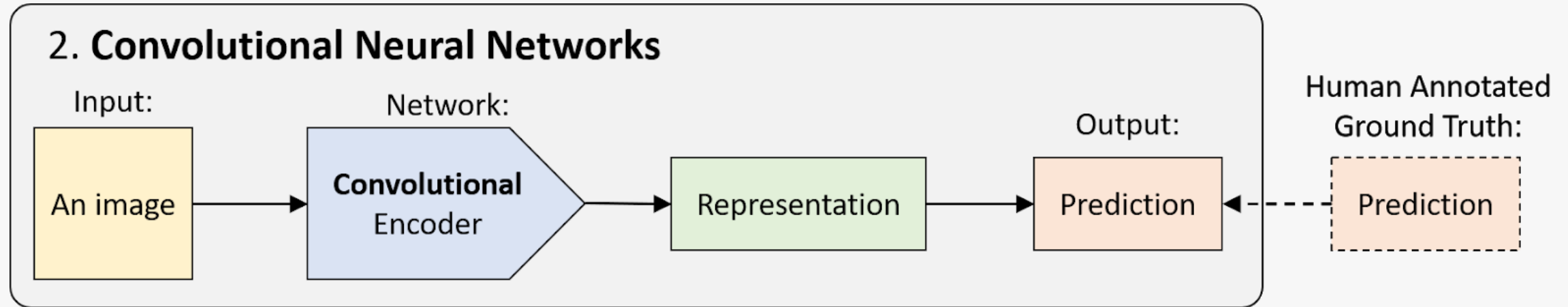


An Architecture for **Many Applications**



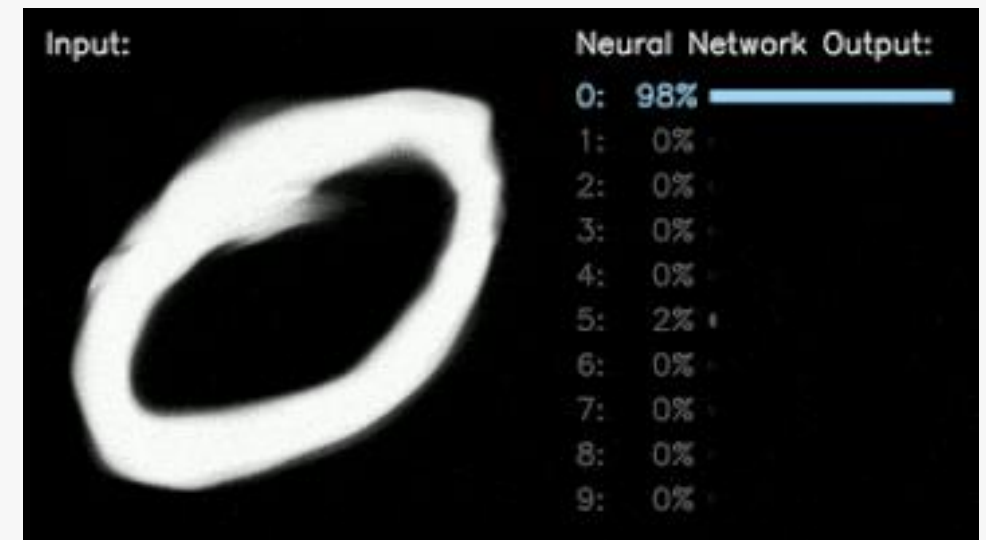
Classification
Object Detection
Segmentation
...

Convolutional Neural Networks (CNNs)



Instead of using only densely-connected layers, they use convolutional layers (**convolutional encoder**).

These networks are used for image classification, object detection, video action recognition, and any data that has some spatial invariance in its structure (e.g., speech audio).



Thank You!