How do Convolutional Neural Networks work?

Original source:

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

https://github.com/brohrer/public-hosting/blob/master/how_CNNs_work.pptx?raw=true

Minor changes, extensions and notes, also added examples (mainly in Polish)

Artur Gramacki (a.gramacki@issi.uz.zgora.pl)

Brandon Rohrer

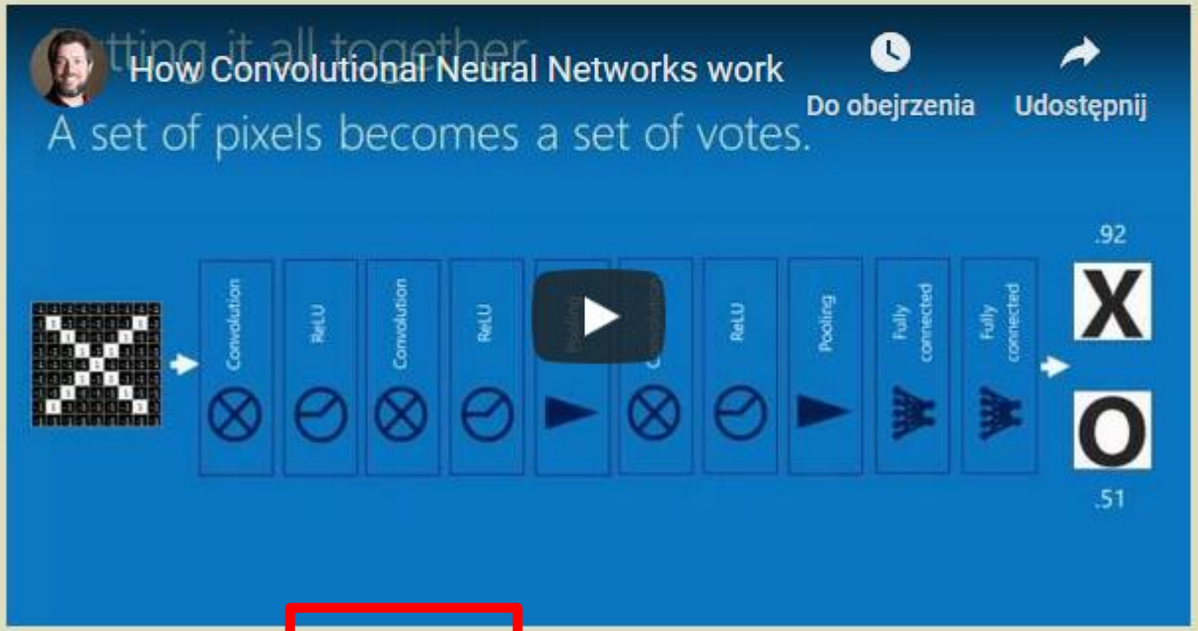
59,3 tys. subskrypcji

https://www.youtube.com/user/BrandonRohrer

How do Convolutional Neural Networks work?

part of the End-to-End Machine Learning library

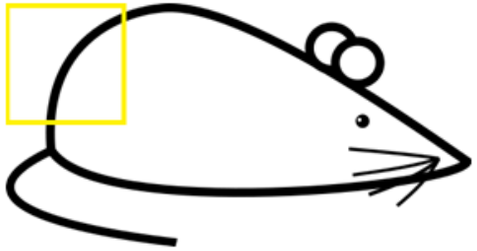
Find the rest of the How Neural Networks Work video series in this free online course.



slides pdf [2MB] ppt [6MB]

Intuicje

Original image



Visualization of the filter on the image

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

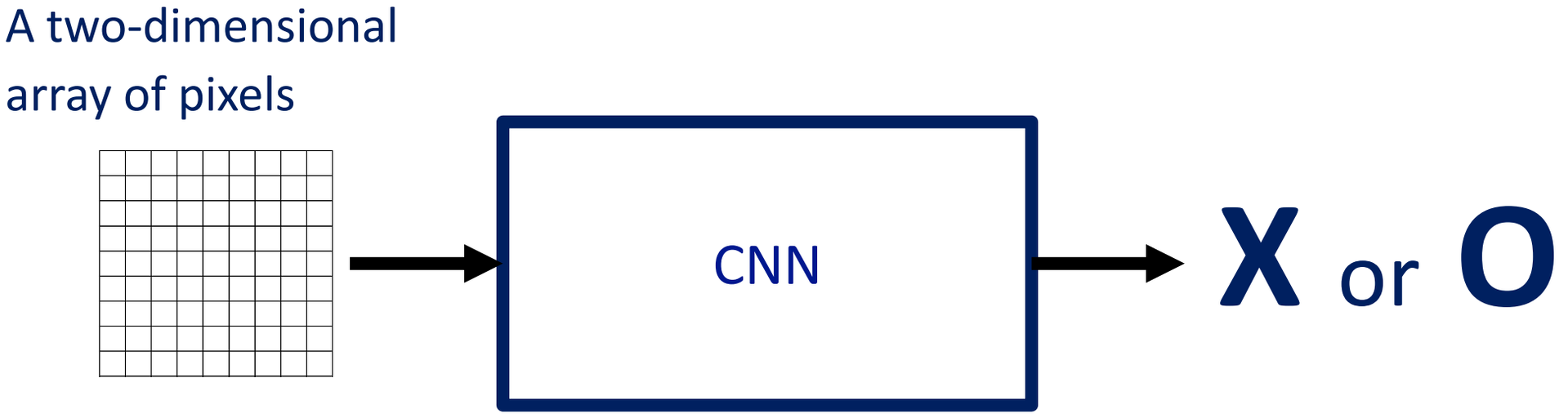
Pixel representation of filter

Multiplication and Summation = 0

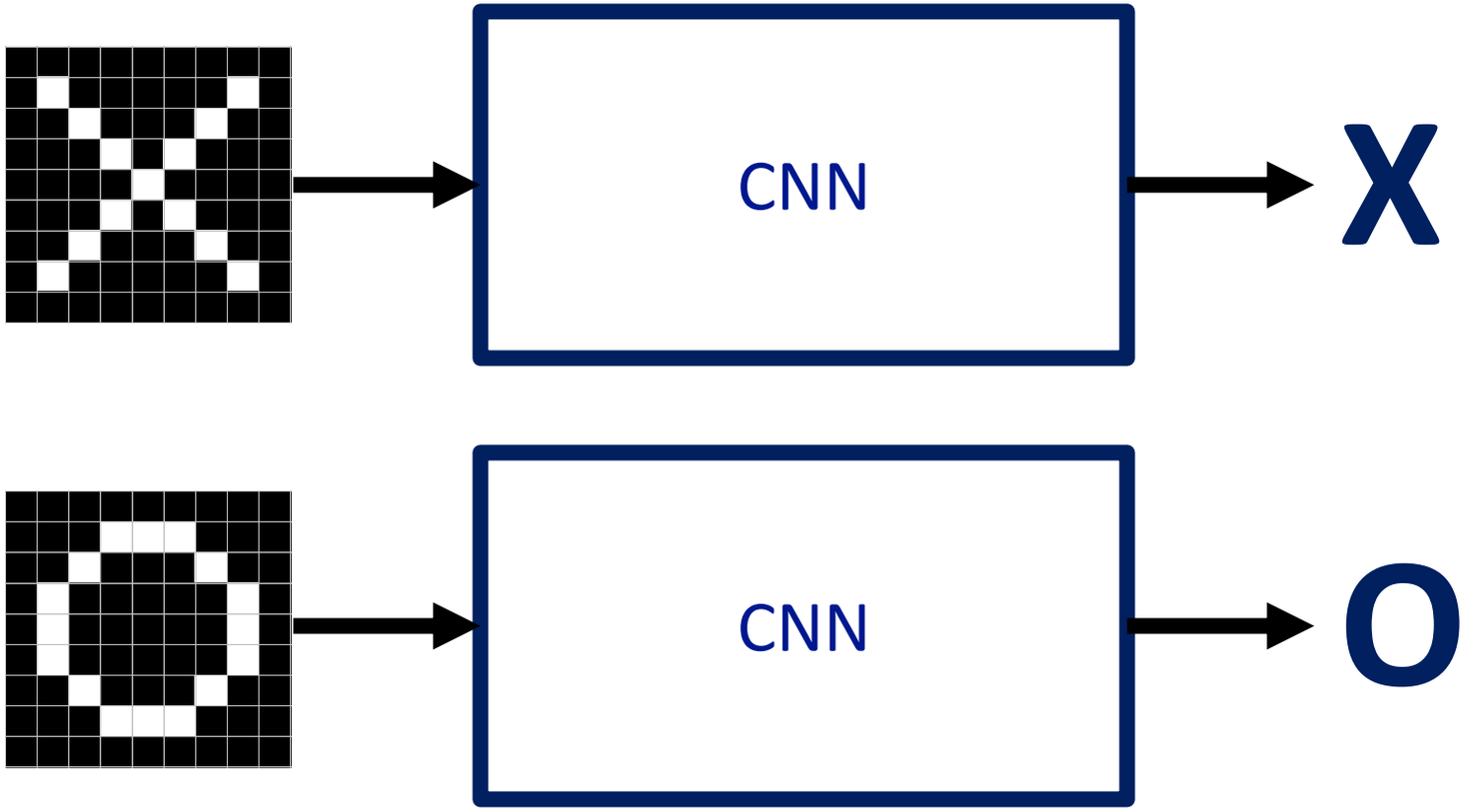
Źródło: https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

A toy ConvNet: X's and O's

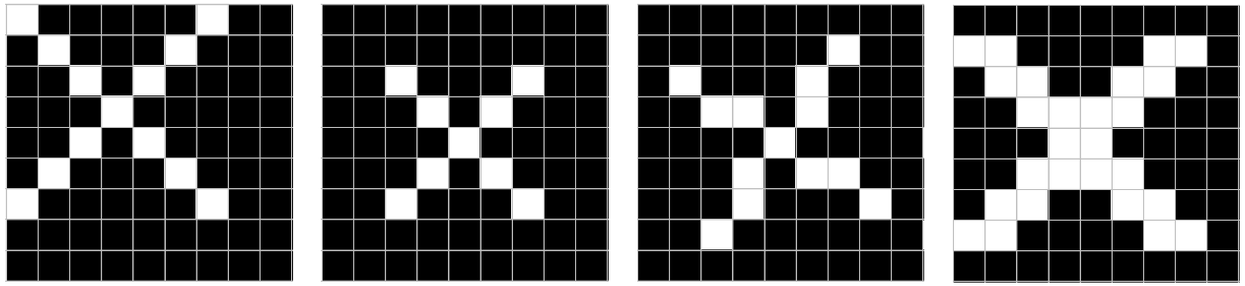
Says whether a picture is of an X or an O



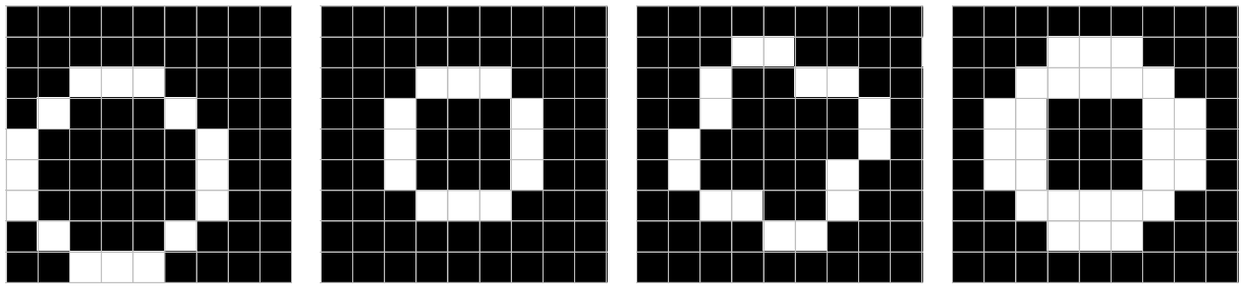
For example



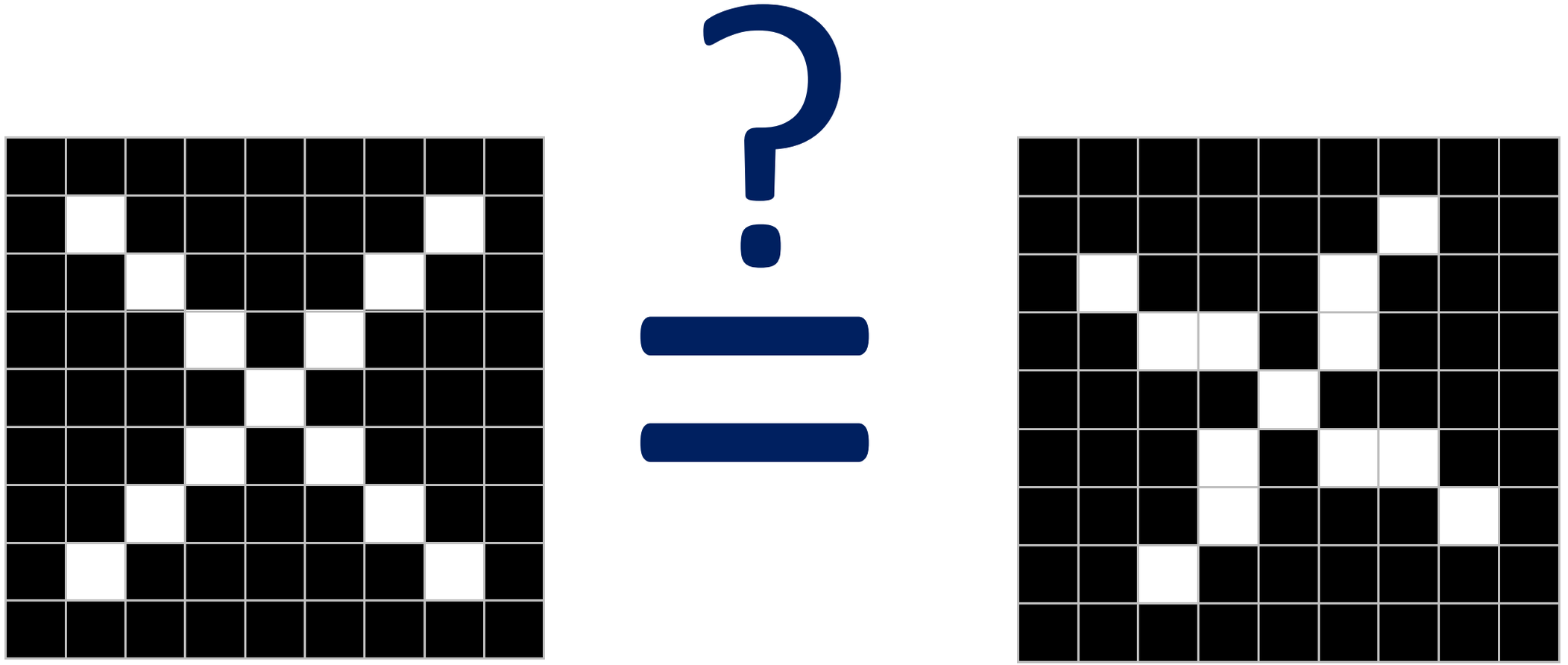
Trickier cases



translation scaling rotation weight

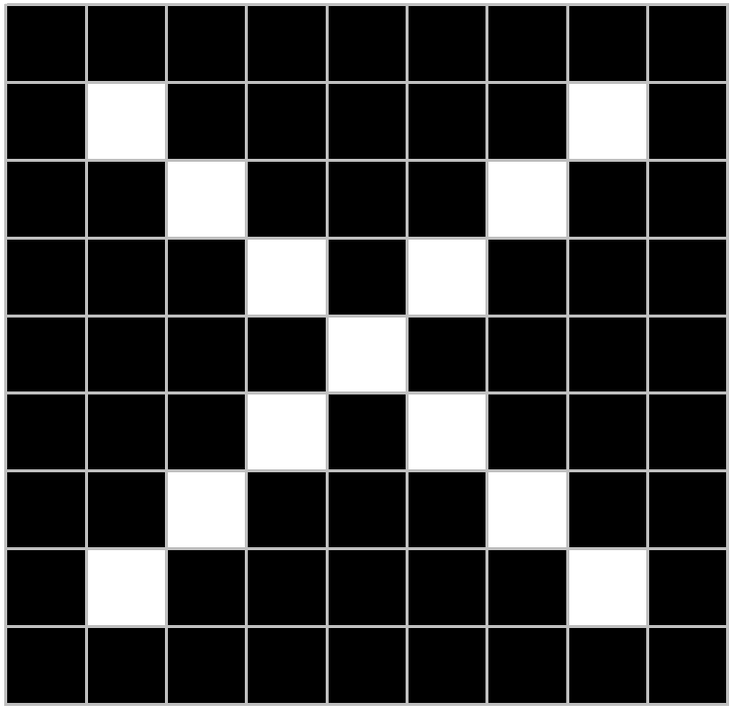


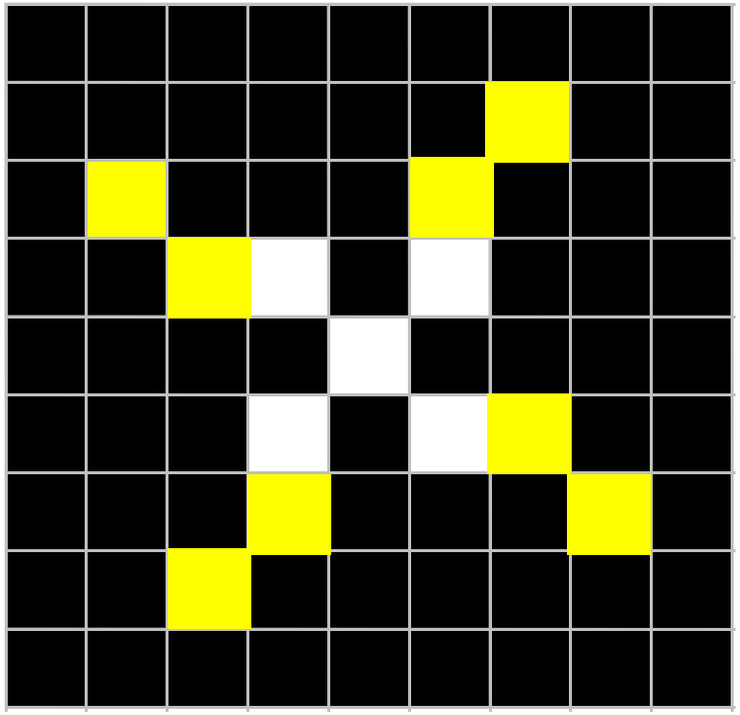
Deciding is hard



Komputery są EKSTREMALNIE dokładne. W rozpoznawaniu obrazów za dokładne. Obraz będzie zgodny ze wzorcem, gdy wszystkie (zdecydowana większość) piksele we wzorcu będą zgadały się z pikselami w porównywanym obrazie.

Porównajmy zgodność pikseli





Na żółto piksele niezgodne ze wzorcem, zgodnych jest mniejszość, mimo to ludzie bez problemu widzą w znaku po prawej stronie krzyżyk. Komputer nie jest taki mądry. CNN pomagają mu zmądrzeć ☺

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

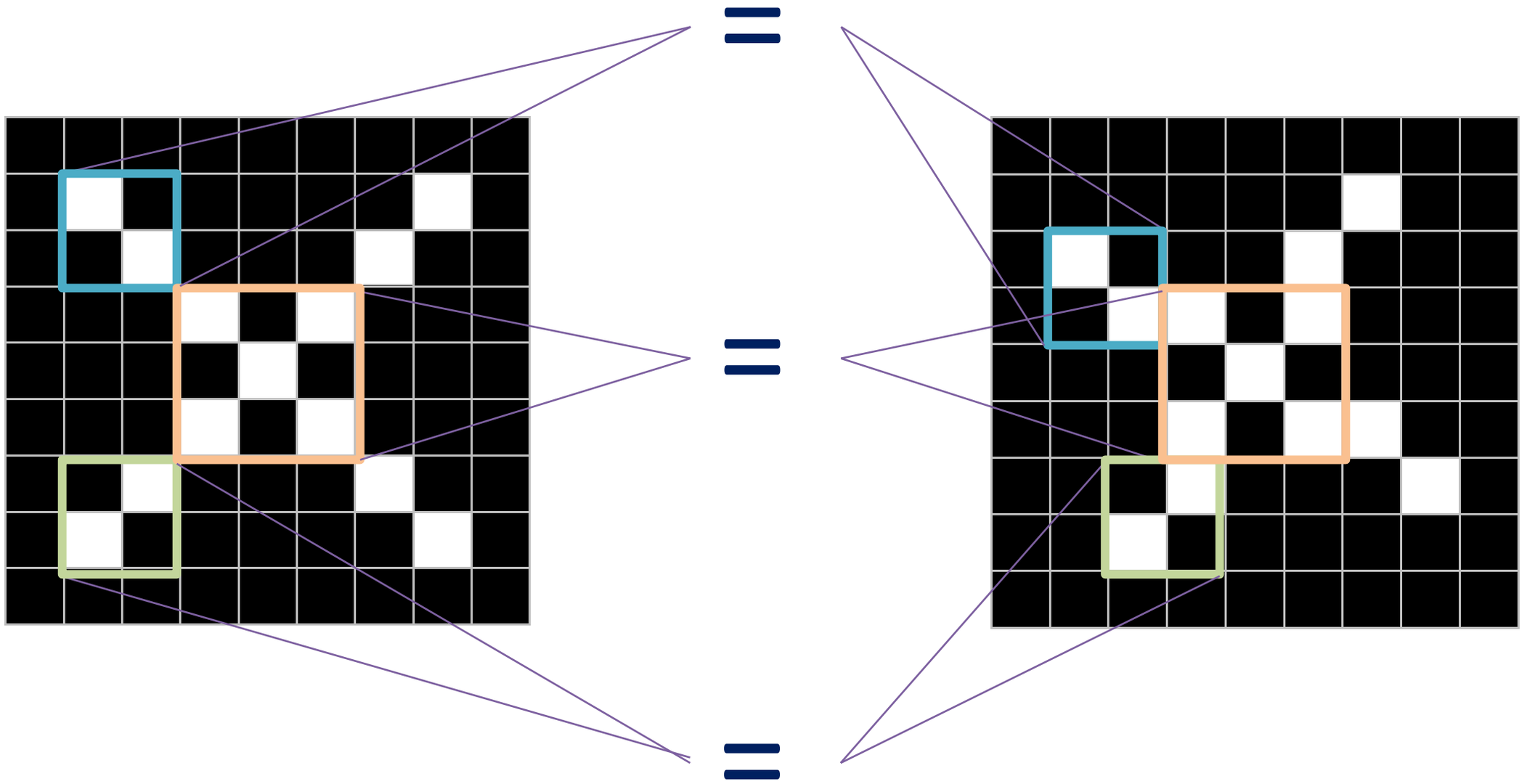
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	X	-1	-1	-1	-1	X	X	-1
-1	X	X	-1	-1	X	X	-1	-1
-1	-1	X	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	X	-1	-1
-1	-1	X	X	-1	-1	X	X	-1
-1	X	X	-1	-1	-1	-1	X	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

Computers are literal

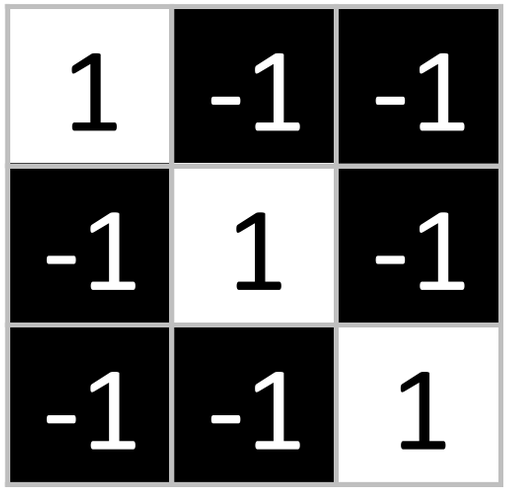
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

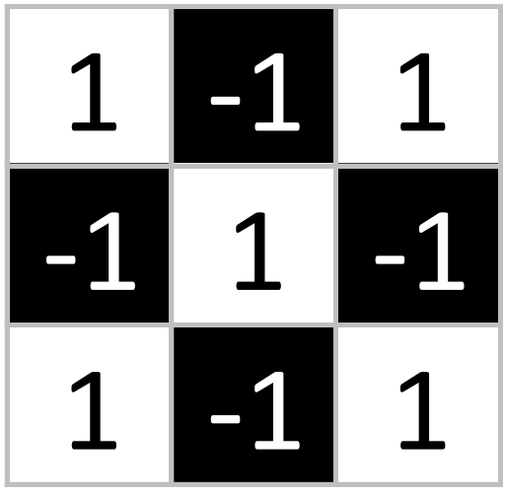
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	1	-1	-1
-1	-1	-1	1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

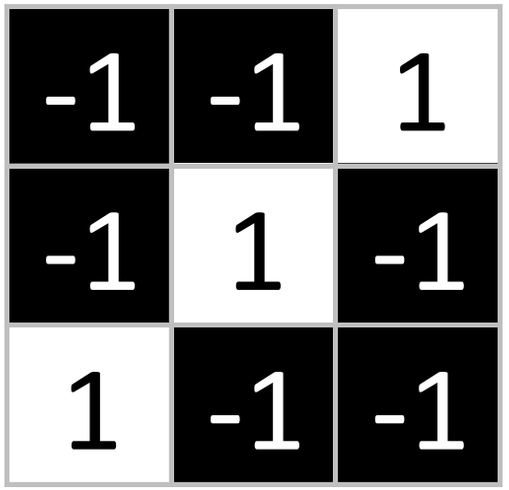
ConvNets match pieces of the image



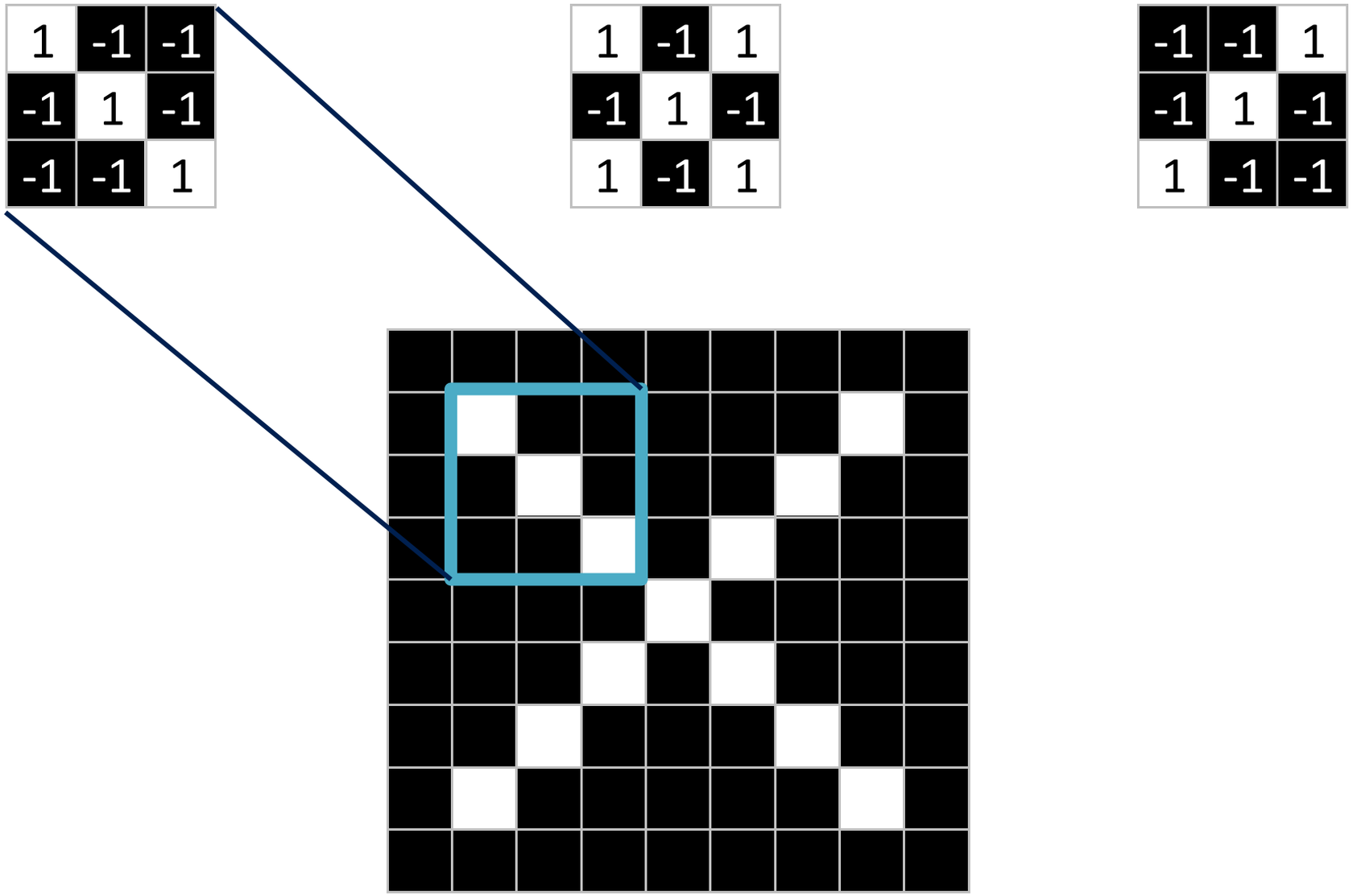
Features match pieces of the image







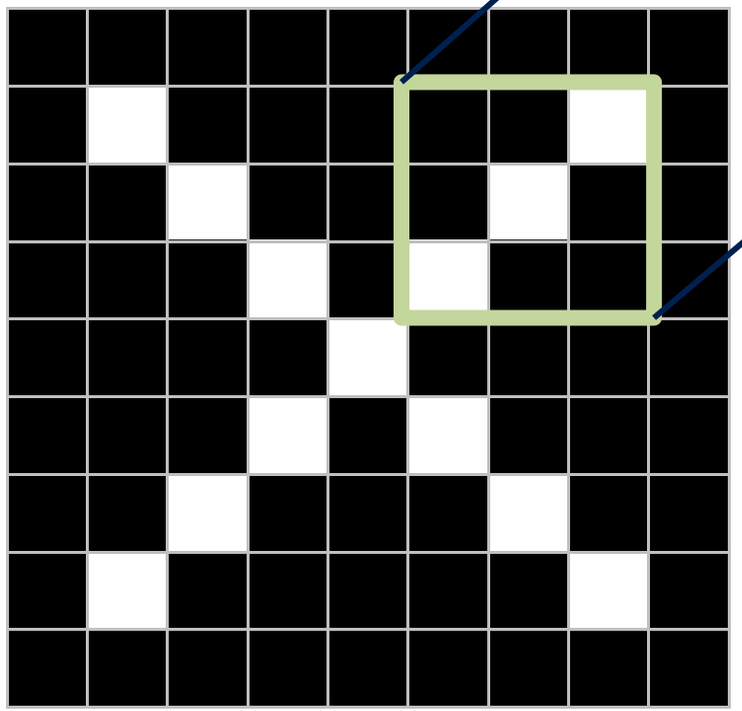
Będziemy w obrazach poszukiwali takich trzech wzorców (cech): **\ X /**
Będą to nasze filtry


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

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

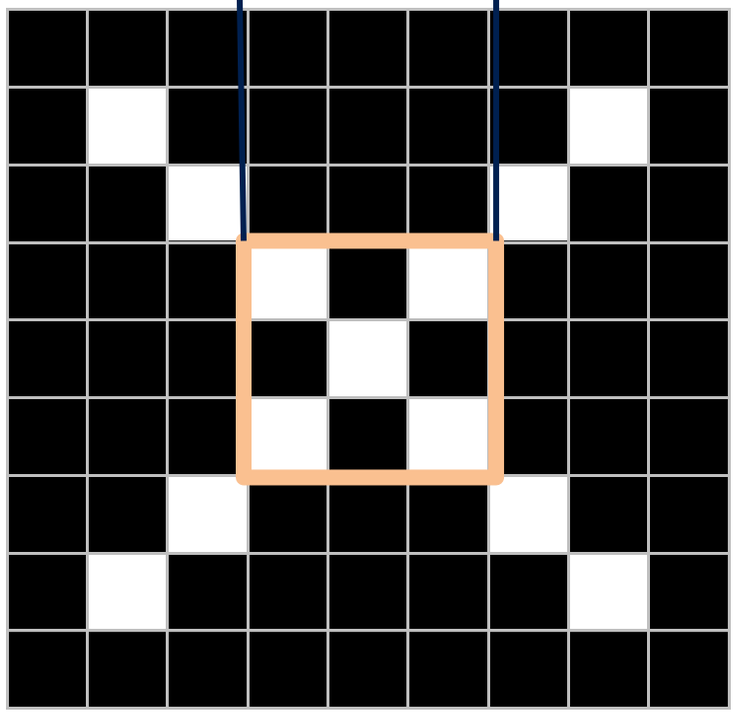
1	-1	1
-1	1	-1
1	-1	1

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

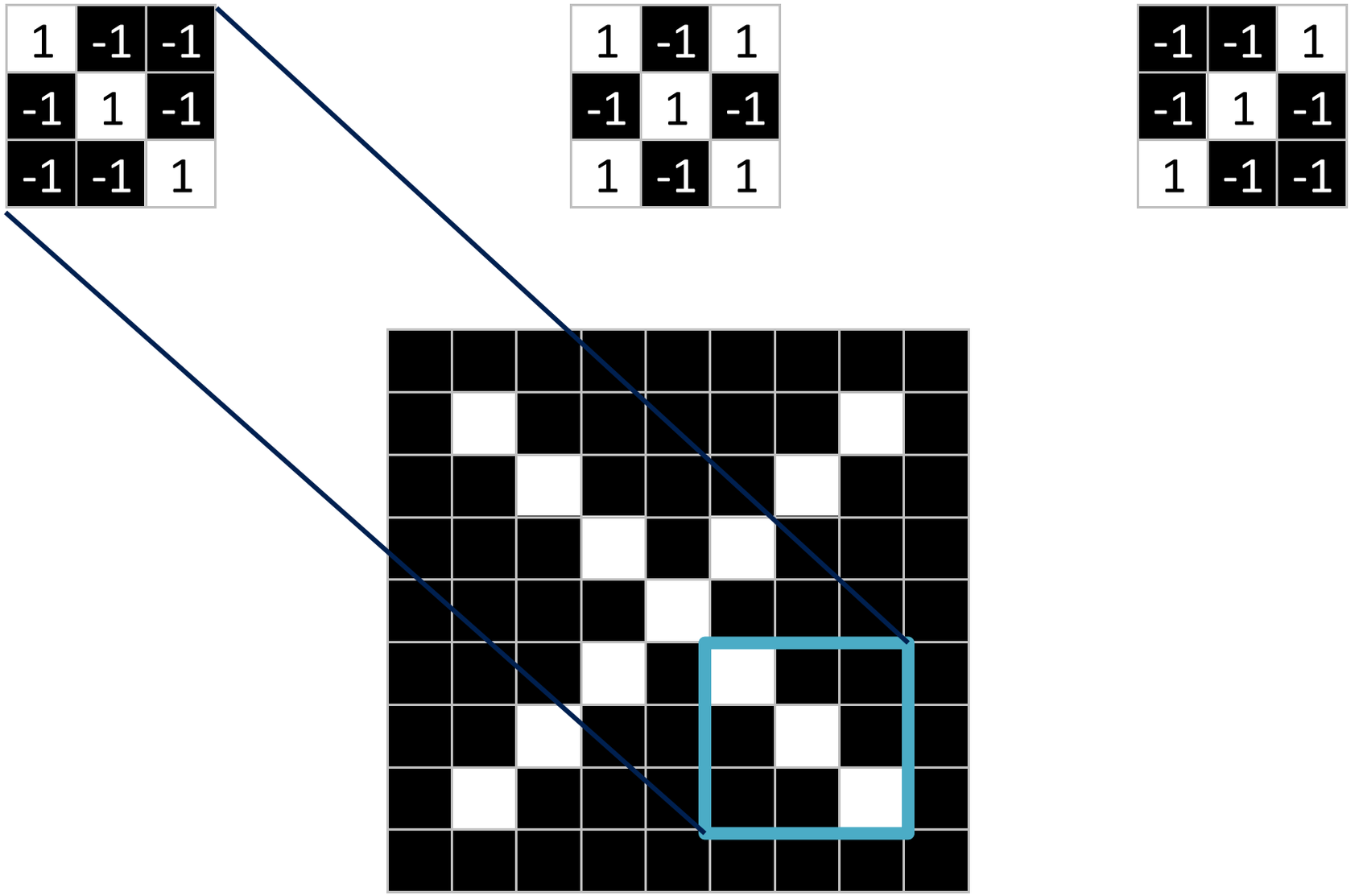


1	-1	-1
-1	1	-1
-1	-1	1

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



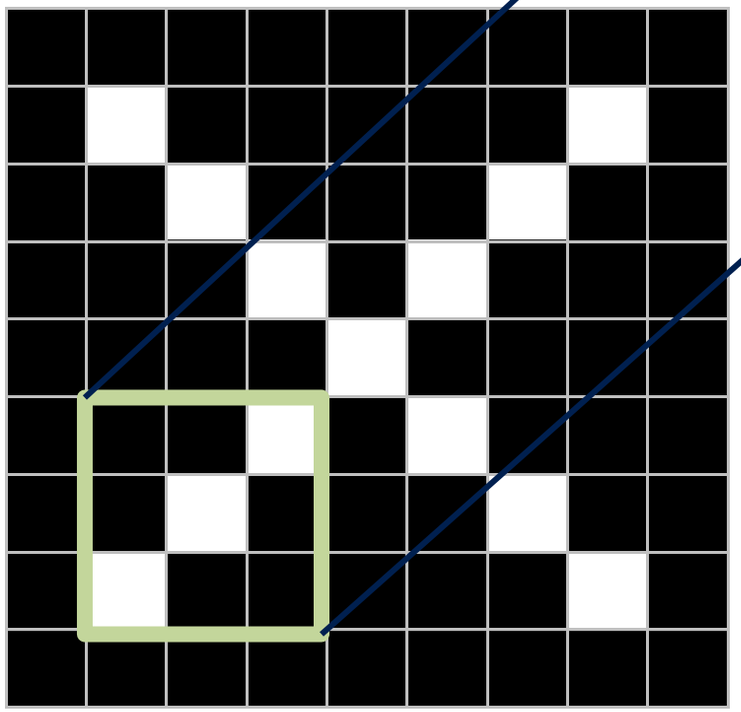
-1	-1	1
-1	1	-1
1	-1	-1


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

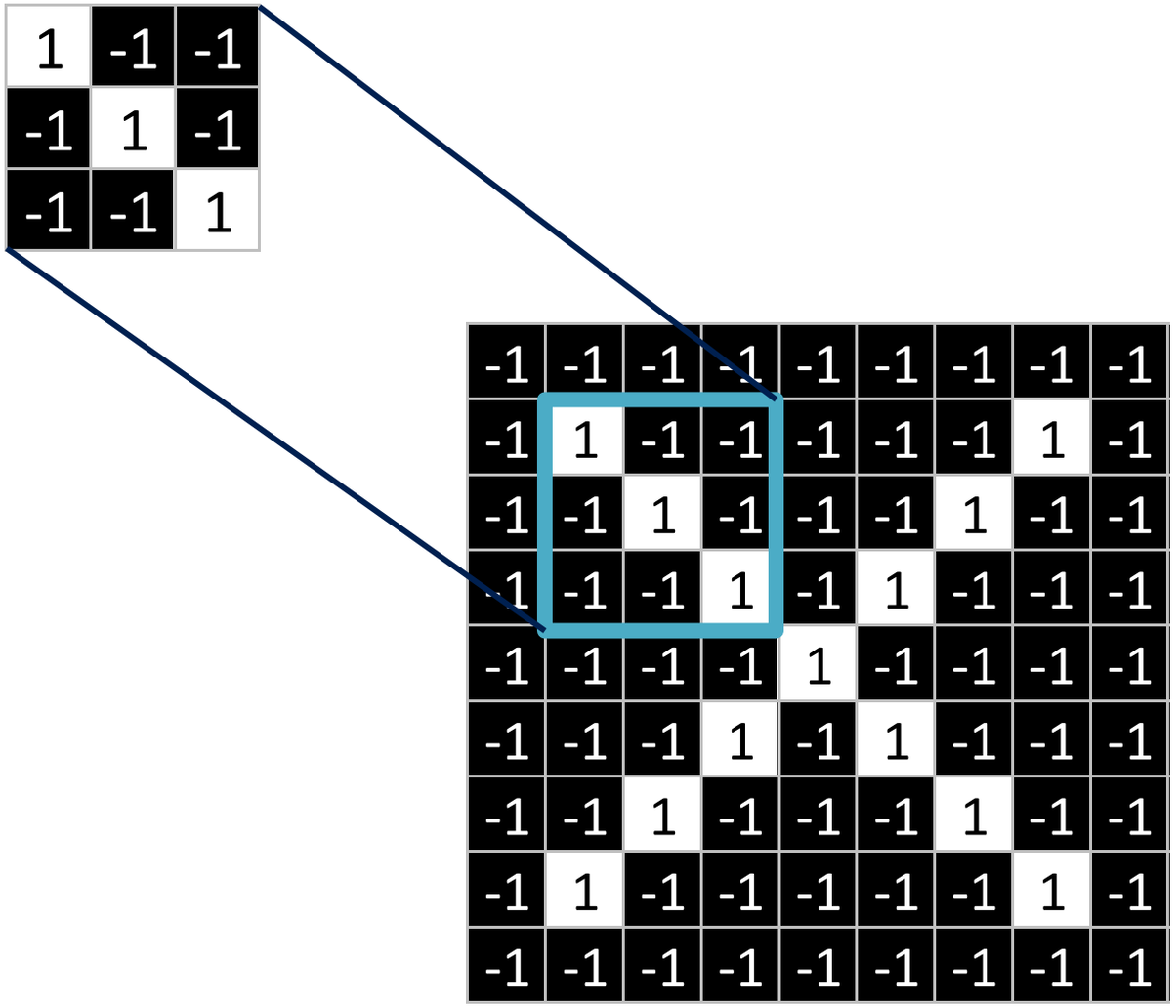
1	-1	-1
-1	1	-1
-1	-1	1

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

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



Filtering: The math behind the match

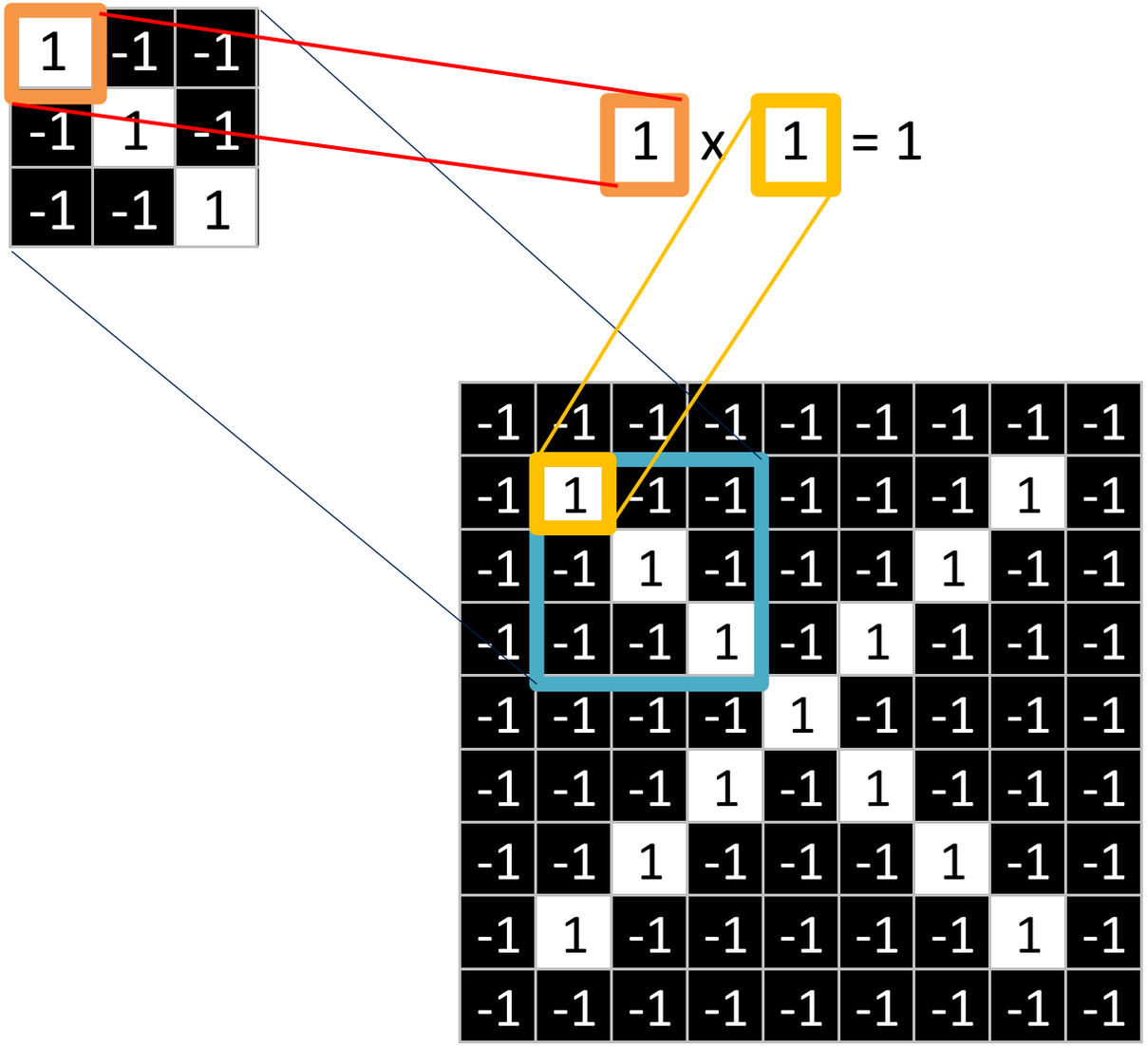


Filtering: The math behind the match

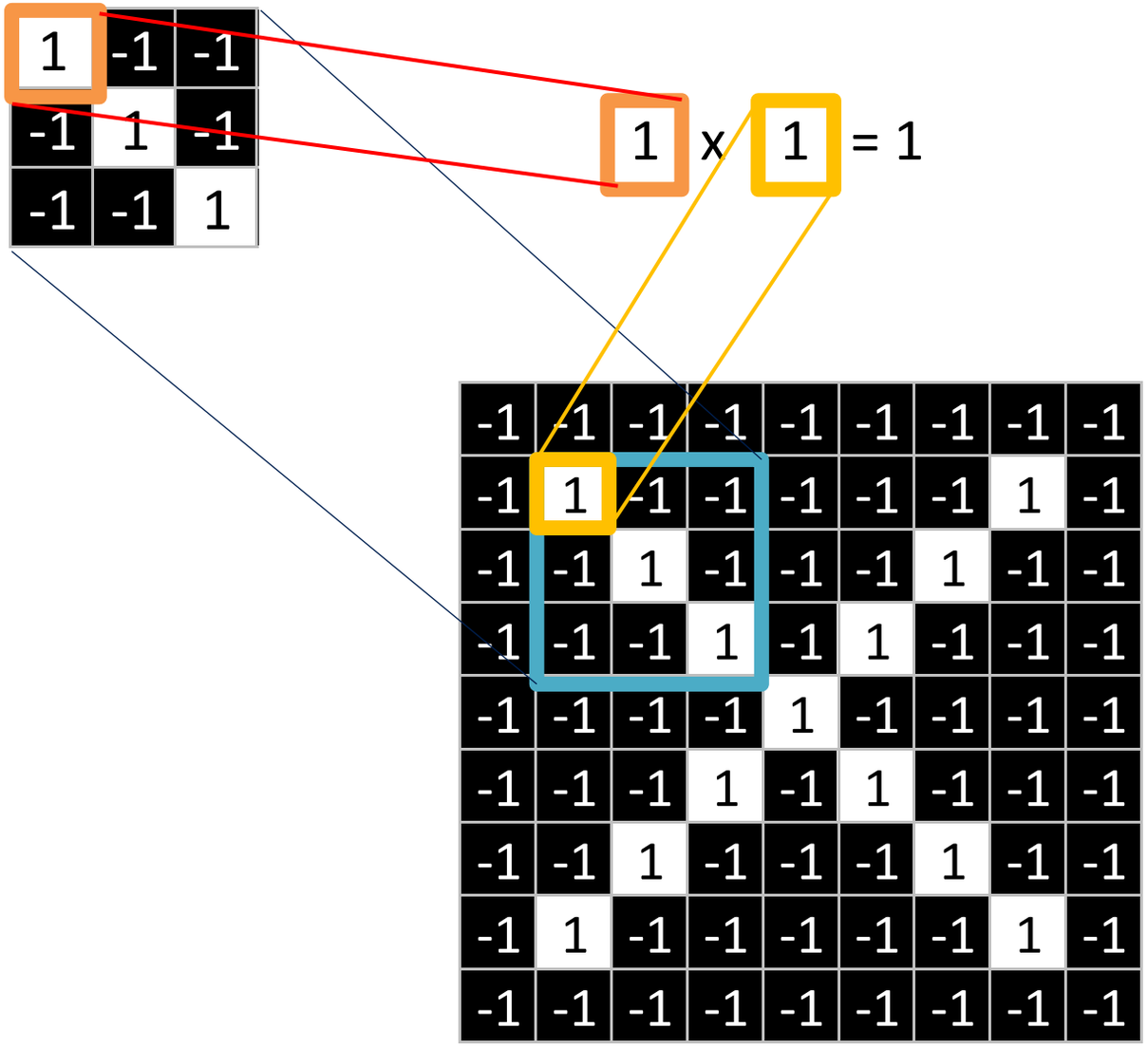
1. Line up the feature and the image patch.
2. Multiply each image pixel by the corresponding feature pixel.
3. Add them up.
4. Divide by the total number of pixels in the feature.

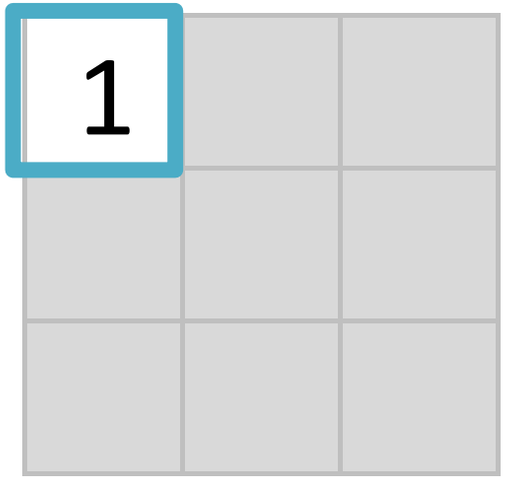
To co wyżej, to poznana już przez nas operacja splotu

Filtering: The math behind the match

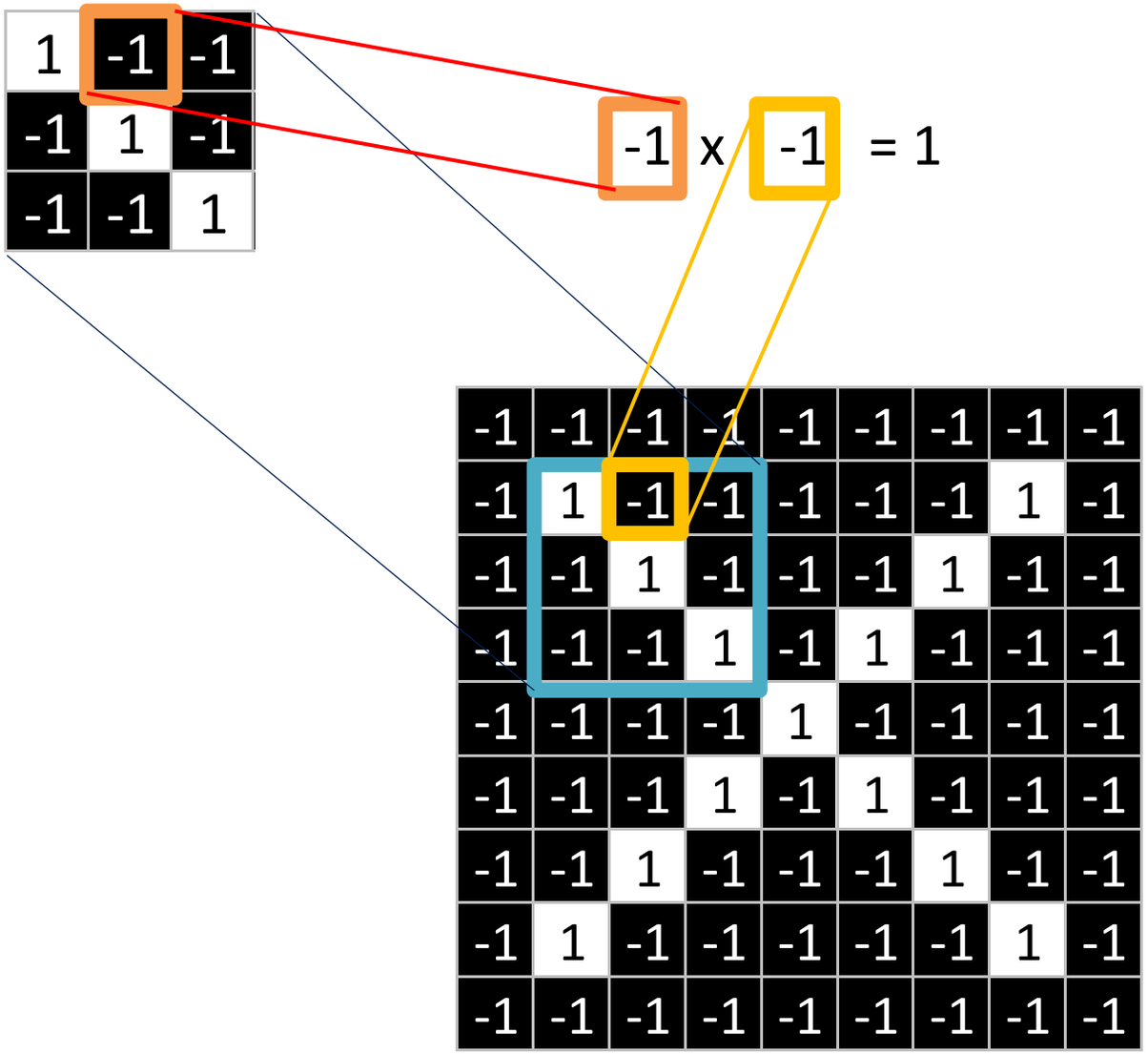


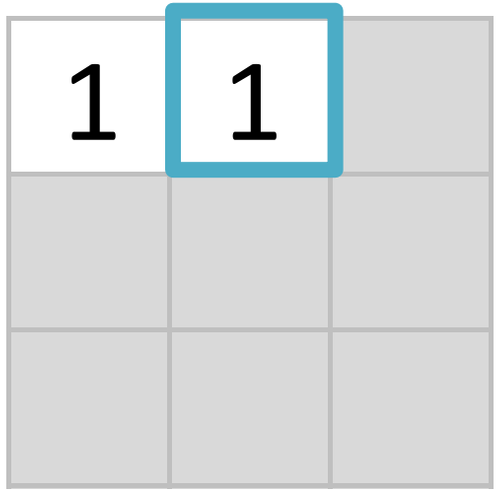
Filtering: The math behind the match



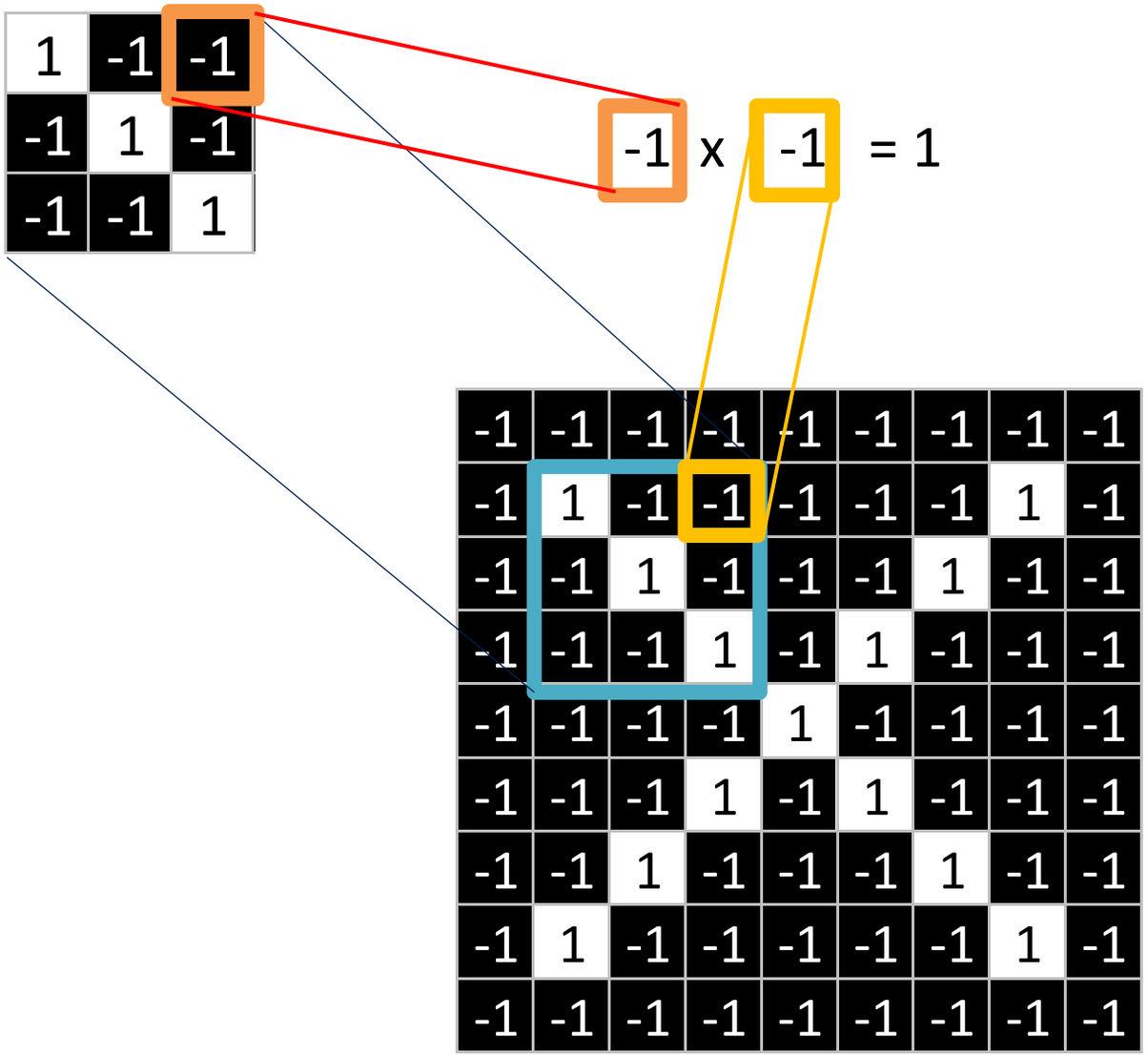


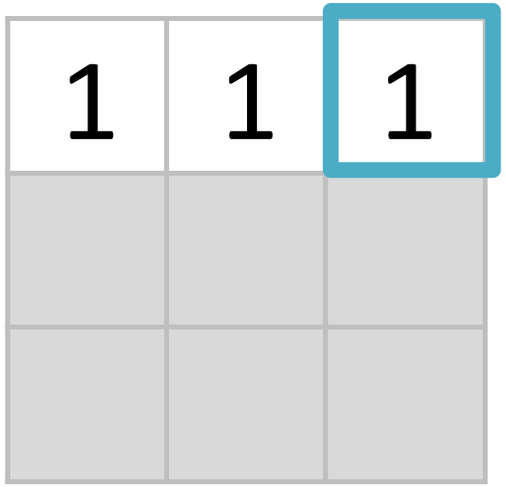
Filtering: The math behind the match



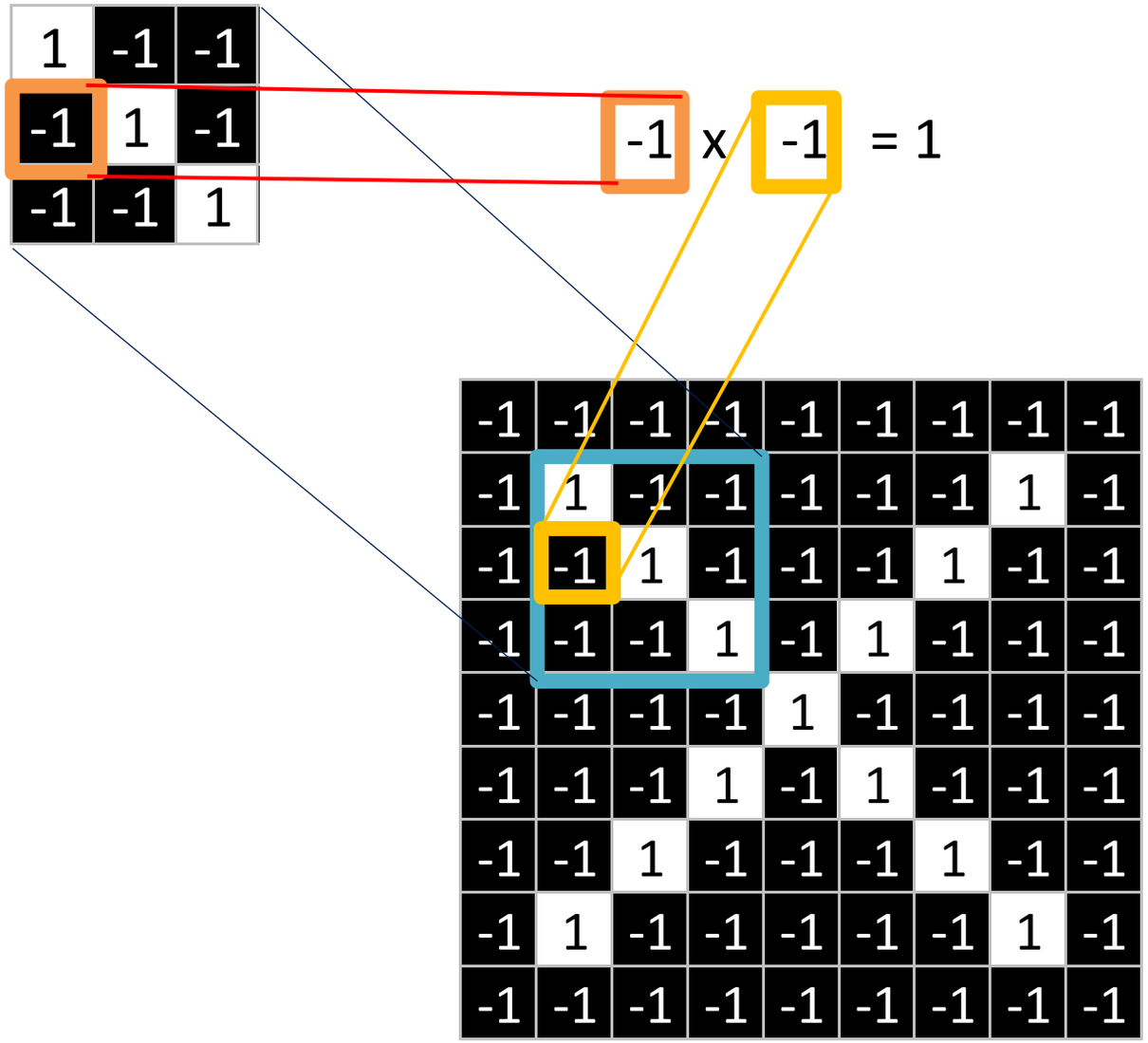


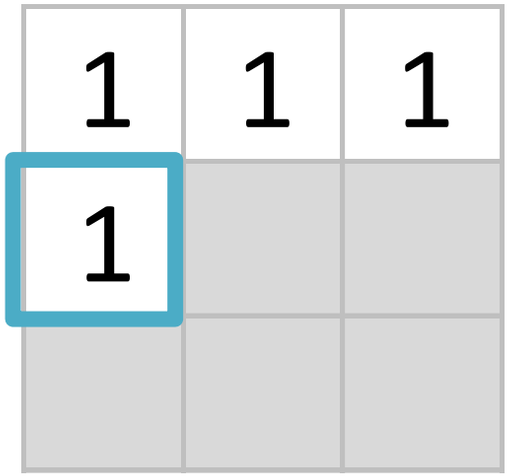
Filtering: The math behind the match



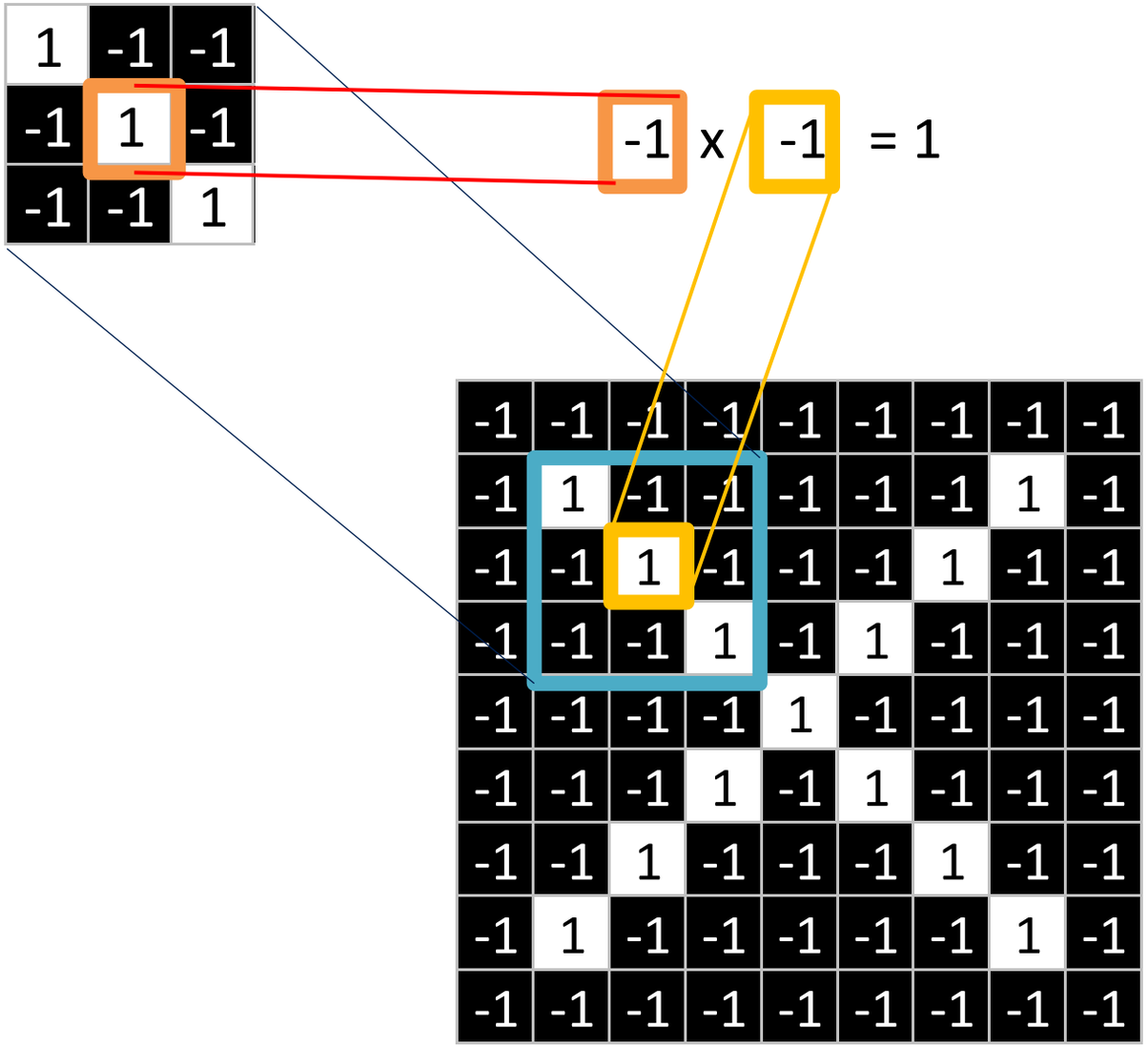


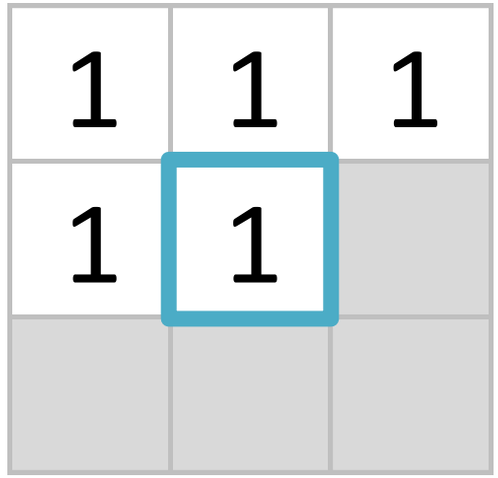
Filtering: The math behind the match



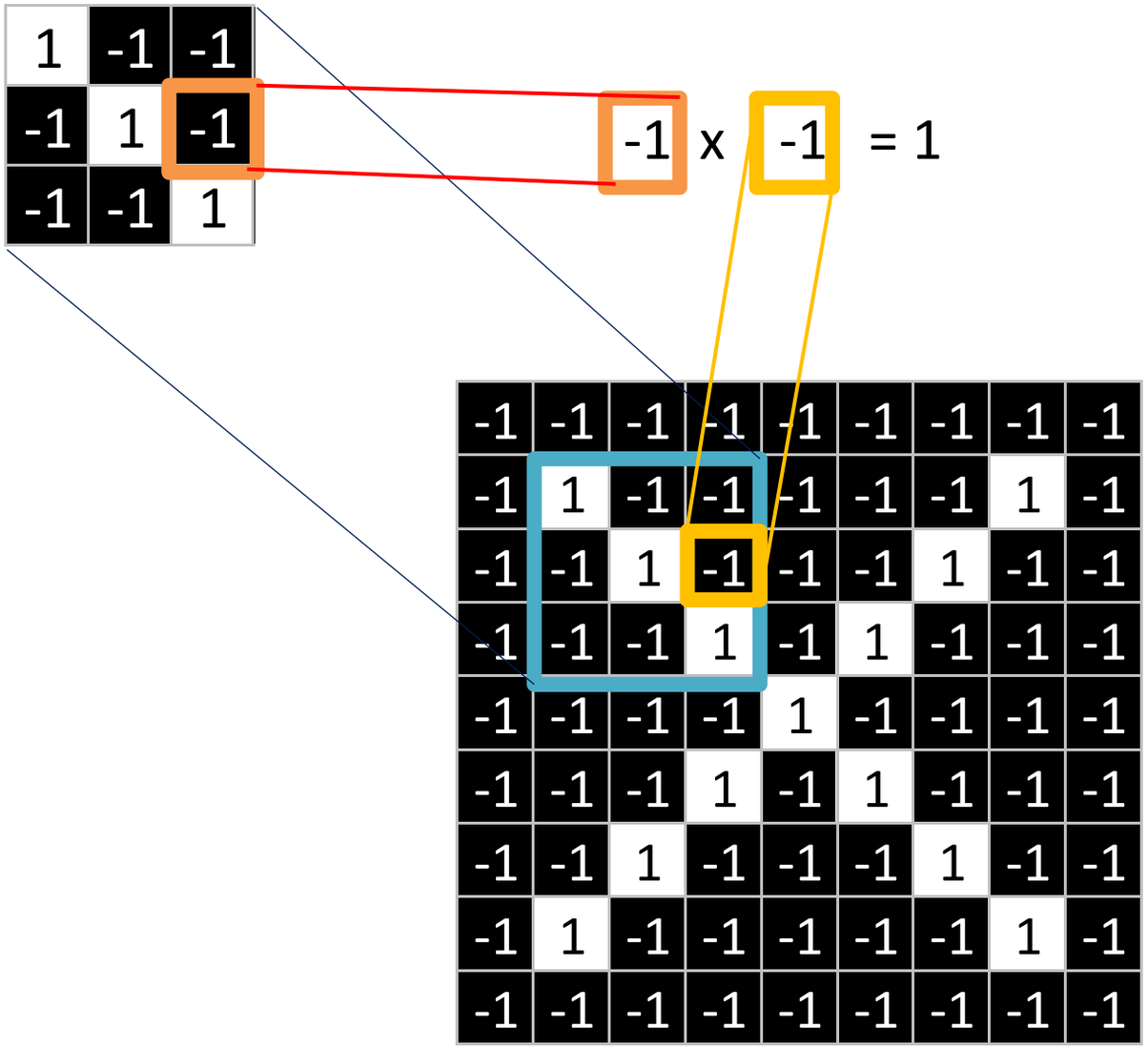


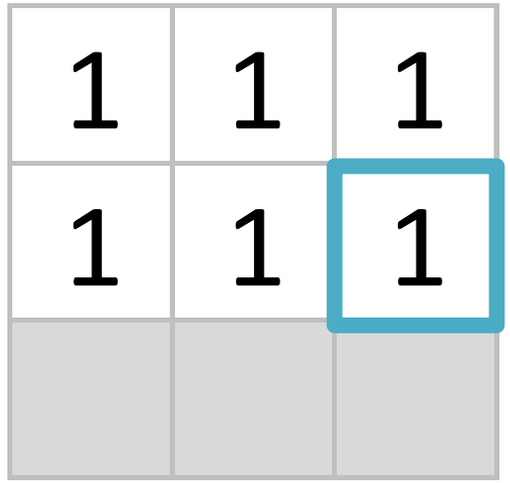
Filtering: The math behind the match



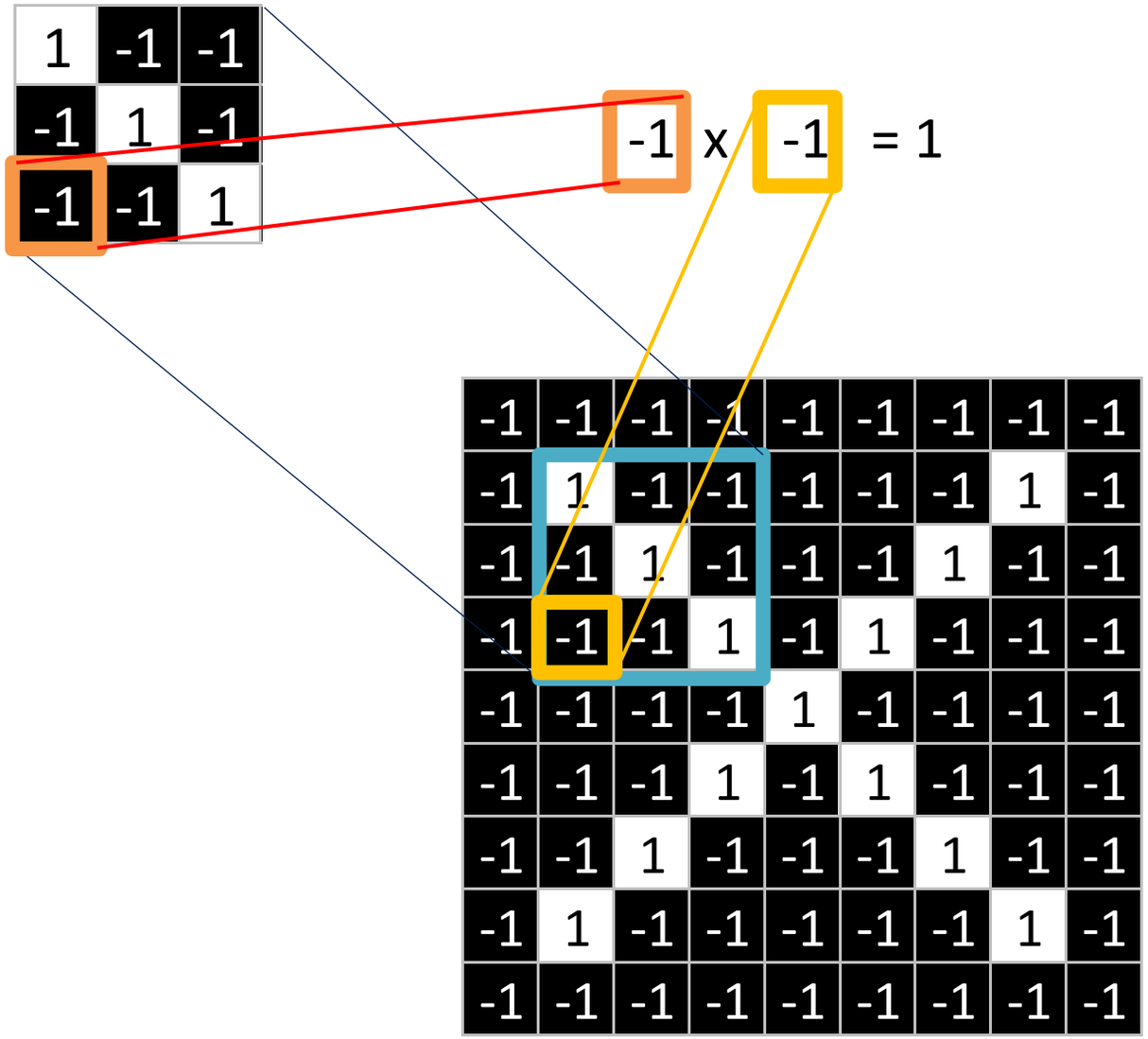


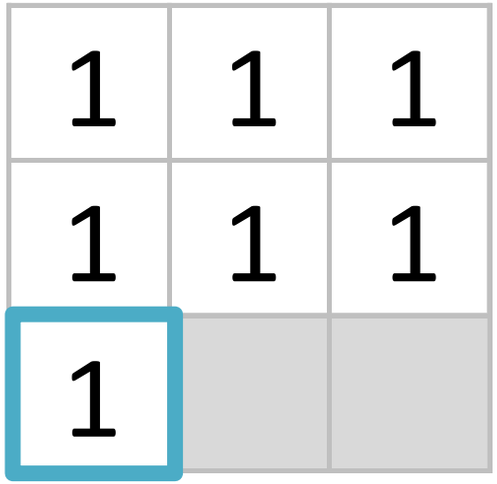
Filtering: The math behind the match



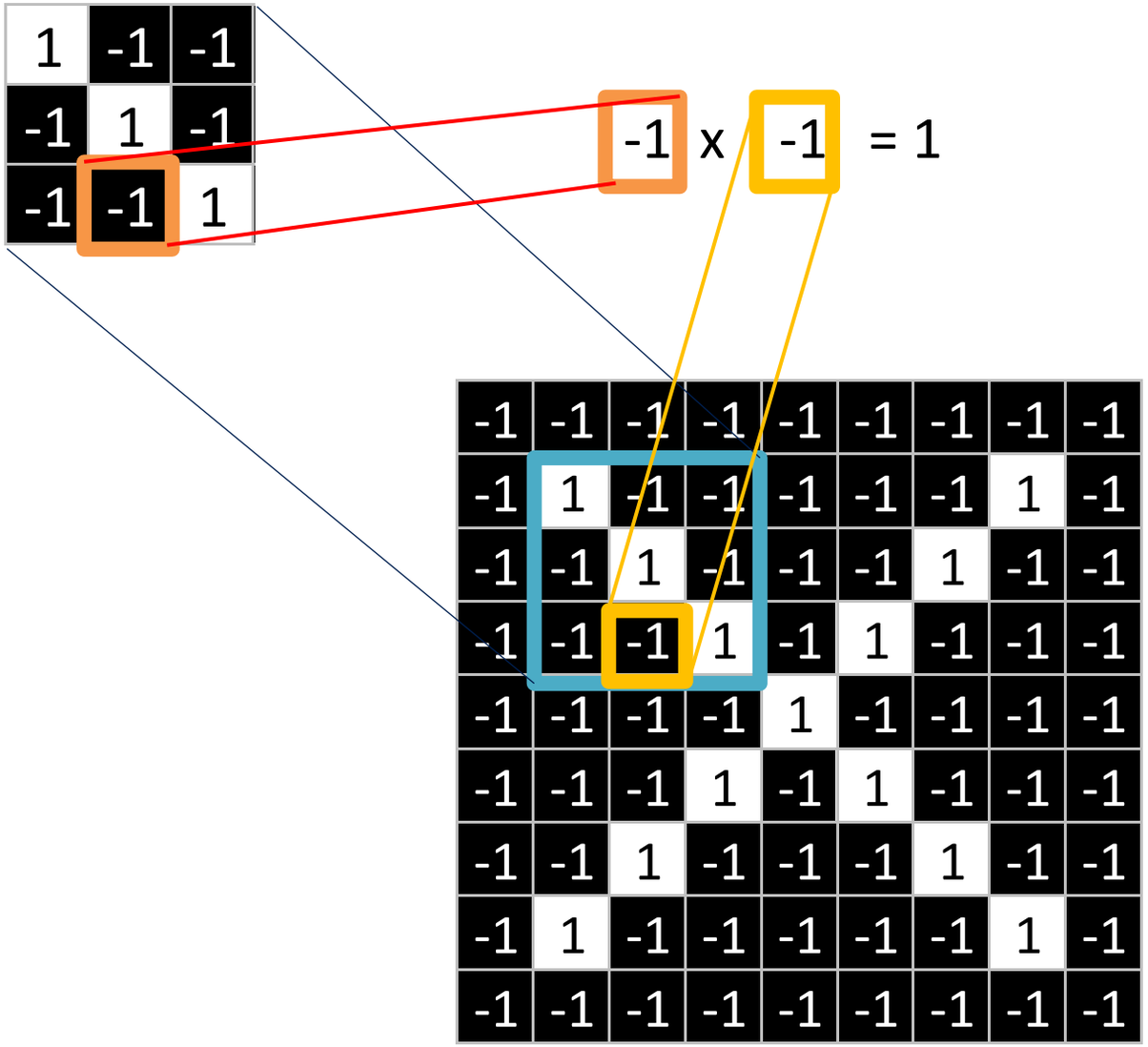


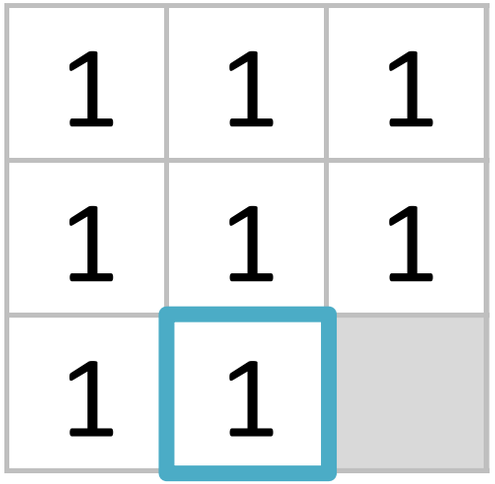
Filtering: The math behind the match



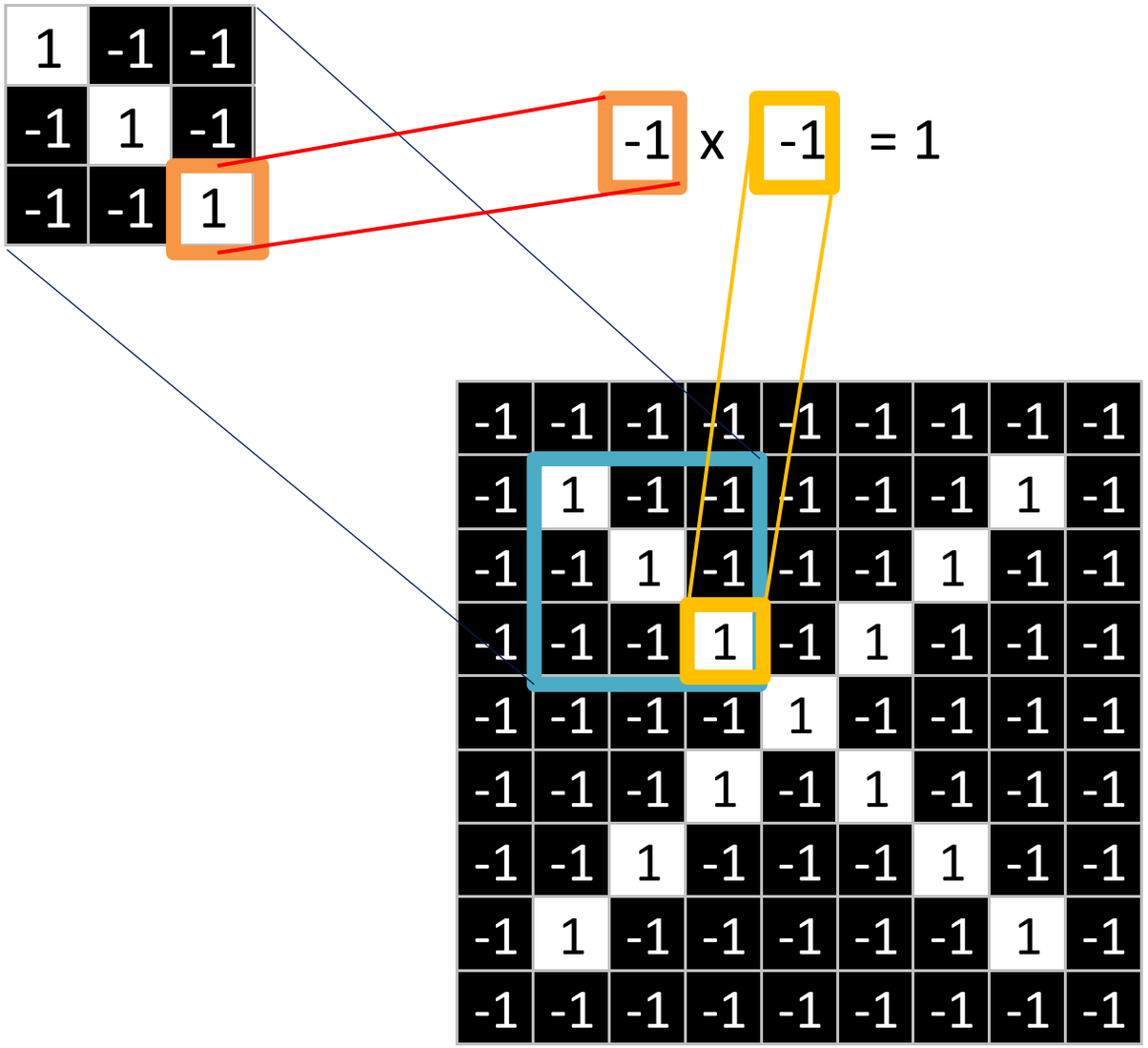


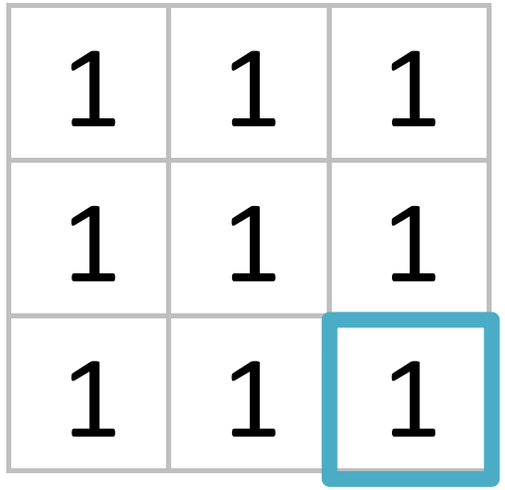
Filtering: The math behind the match



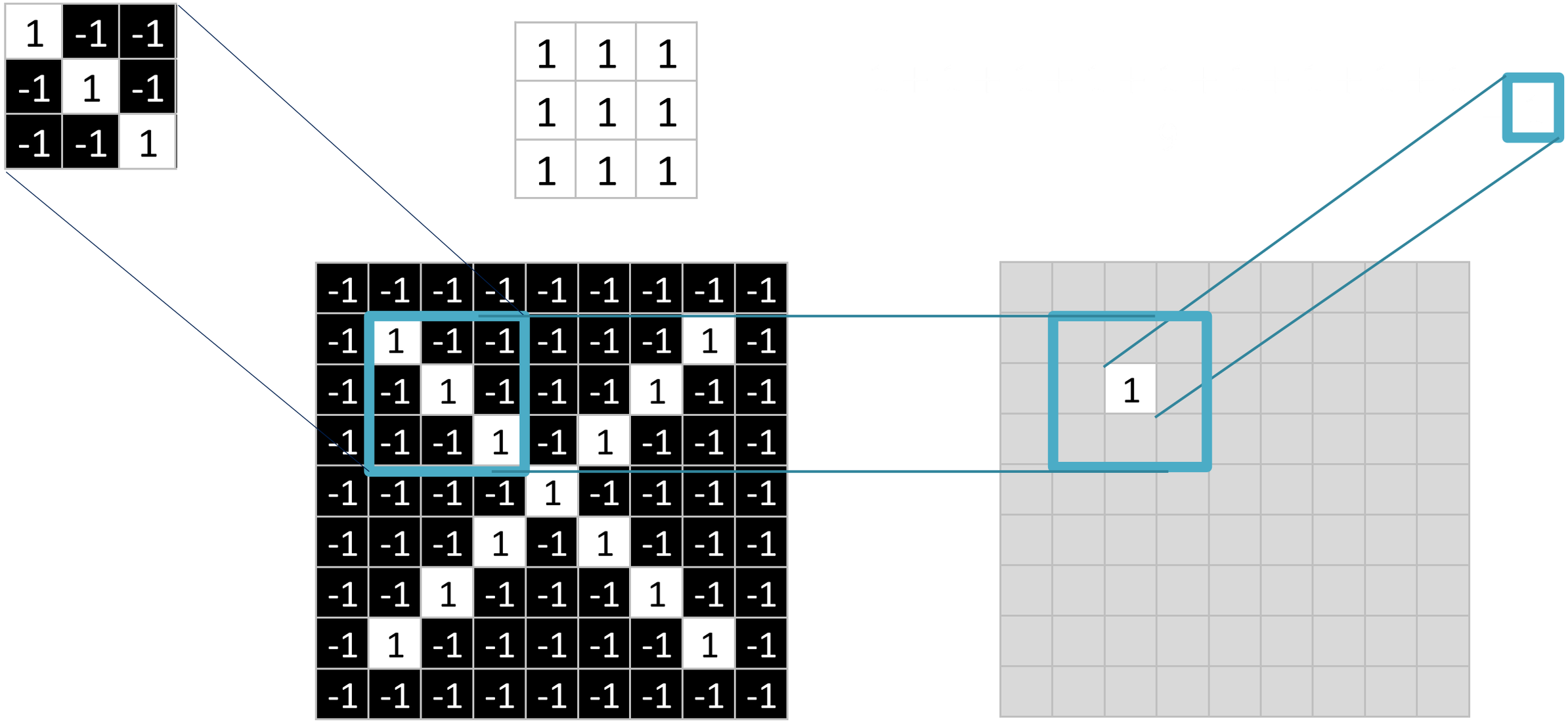


Filtering: The math behind the match

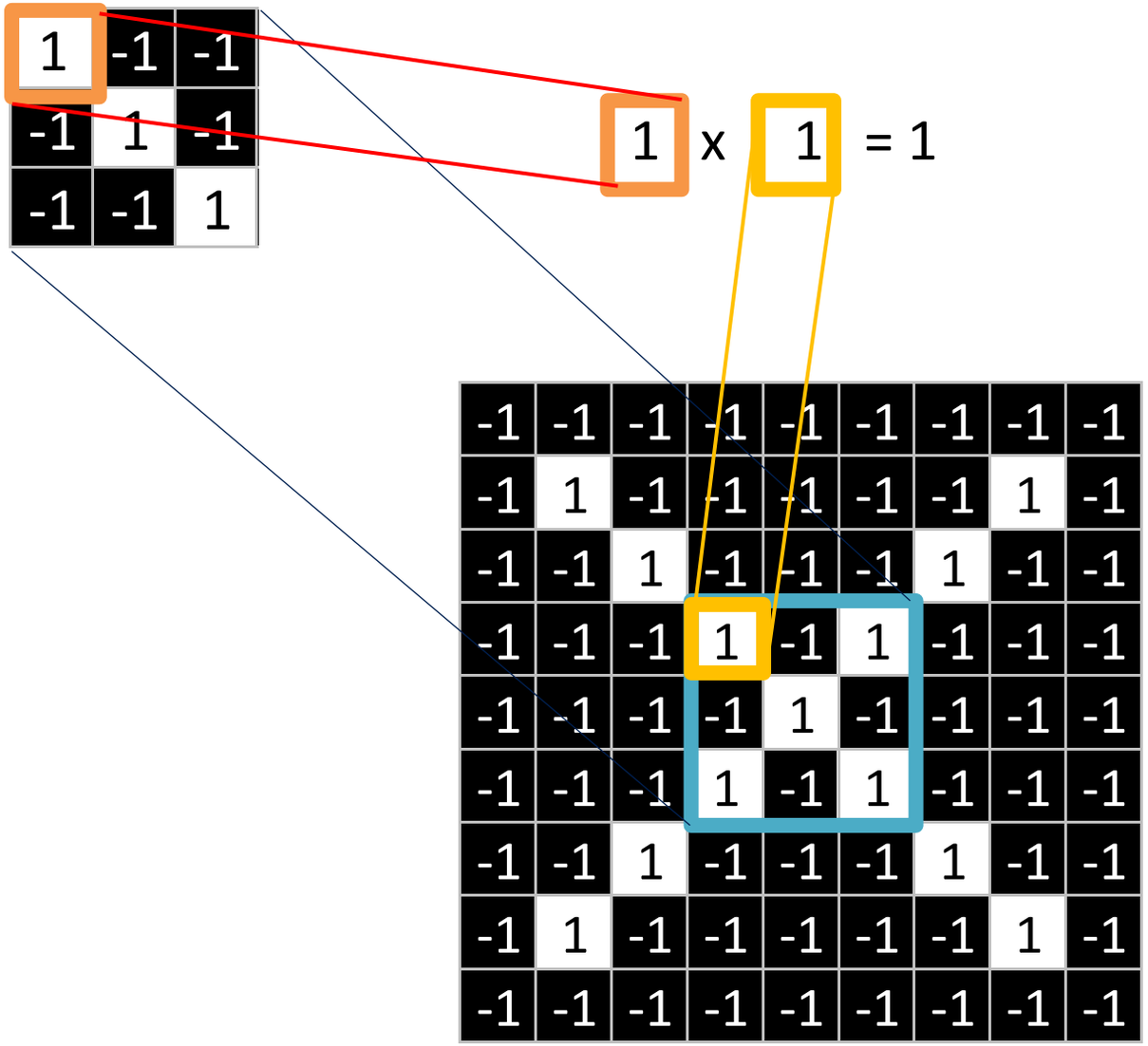


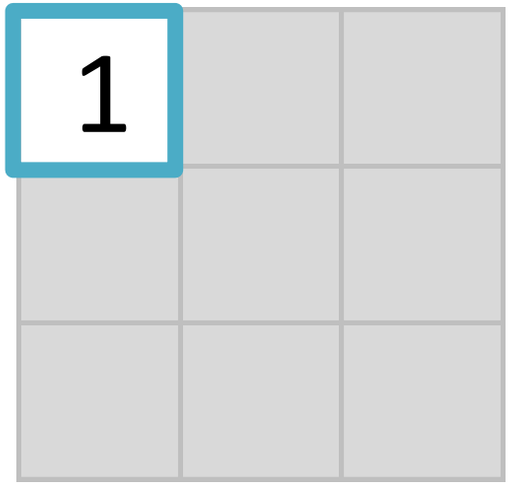


Filtering: The math behind the match

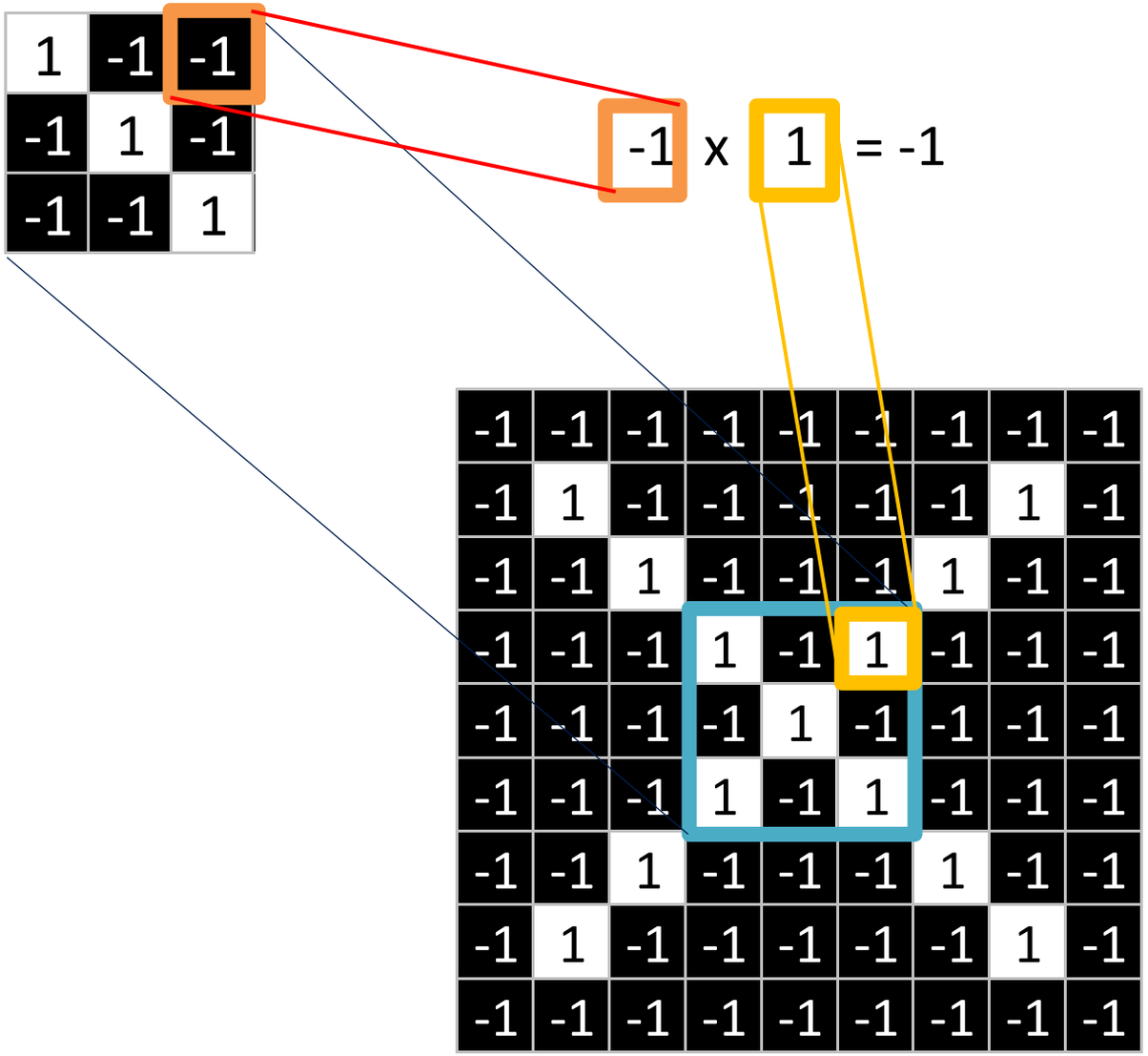


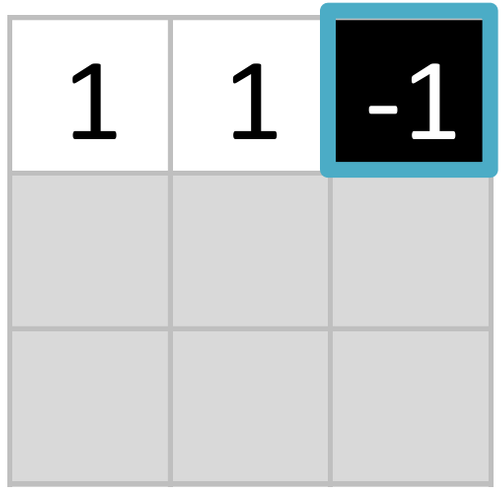
Filtering: The math behind the match



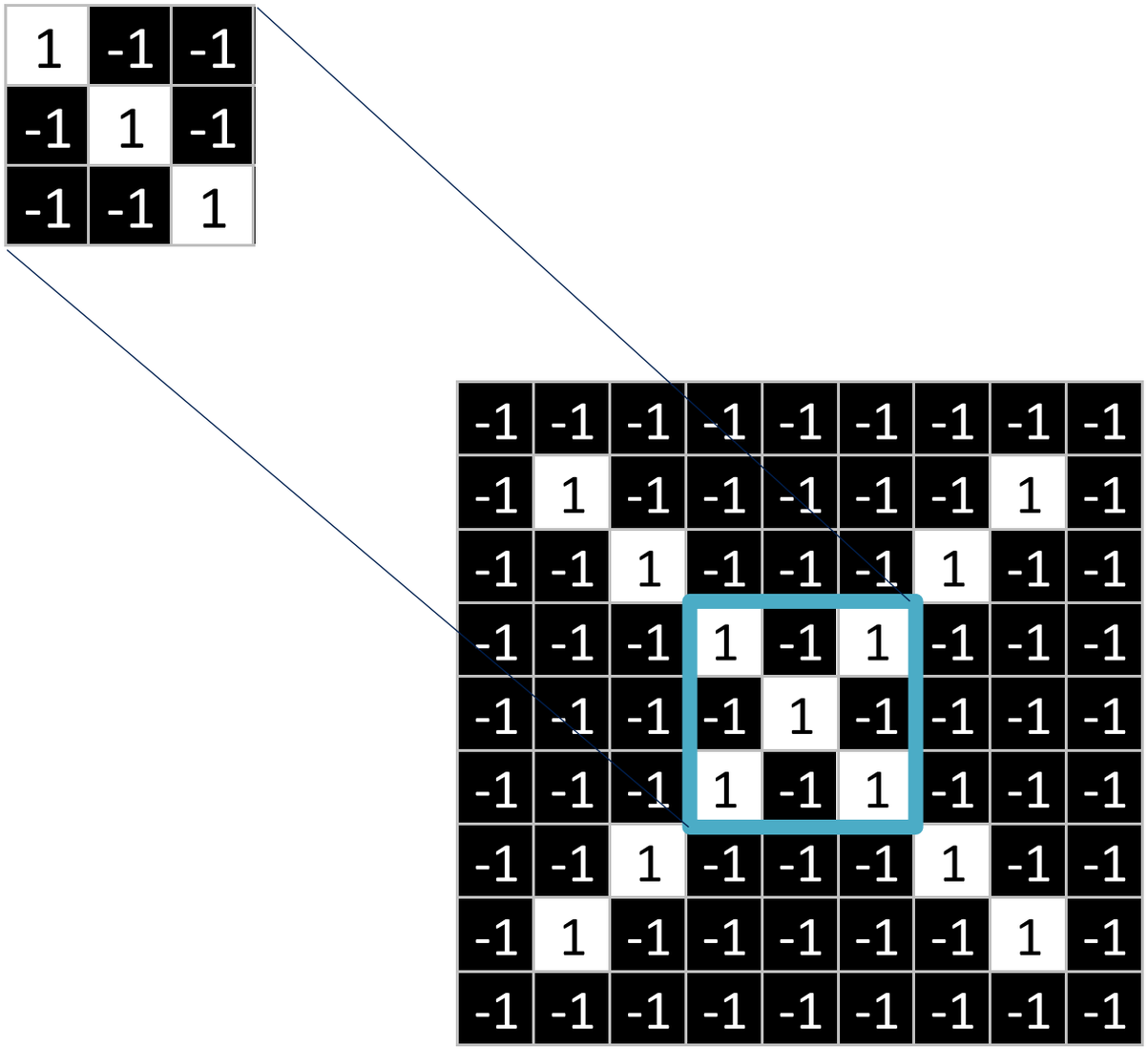


Filtering: The math behind the match



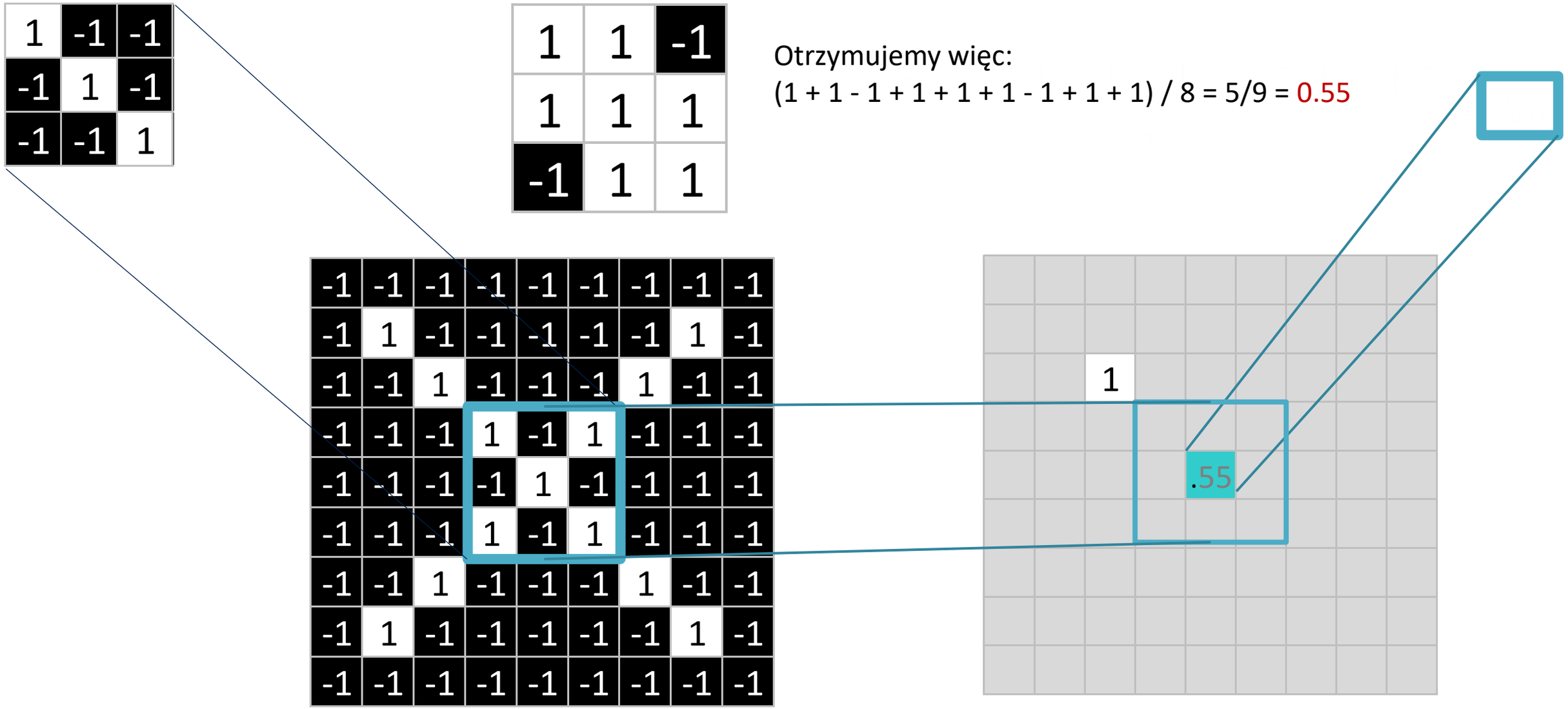


Filtering: The math behind the match



1	1	-1
1	1	1
-1	1	1

Filtering: The math behind the match



Convolution: Trying every possible match

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

Otrzymujemy więc mapę dopasowia (map of matches) obrazu do wzorca „odwrotny ukośnik”.

1 to maksymalne dopasowanie

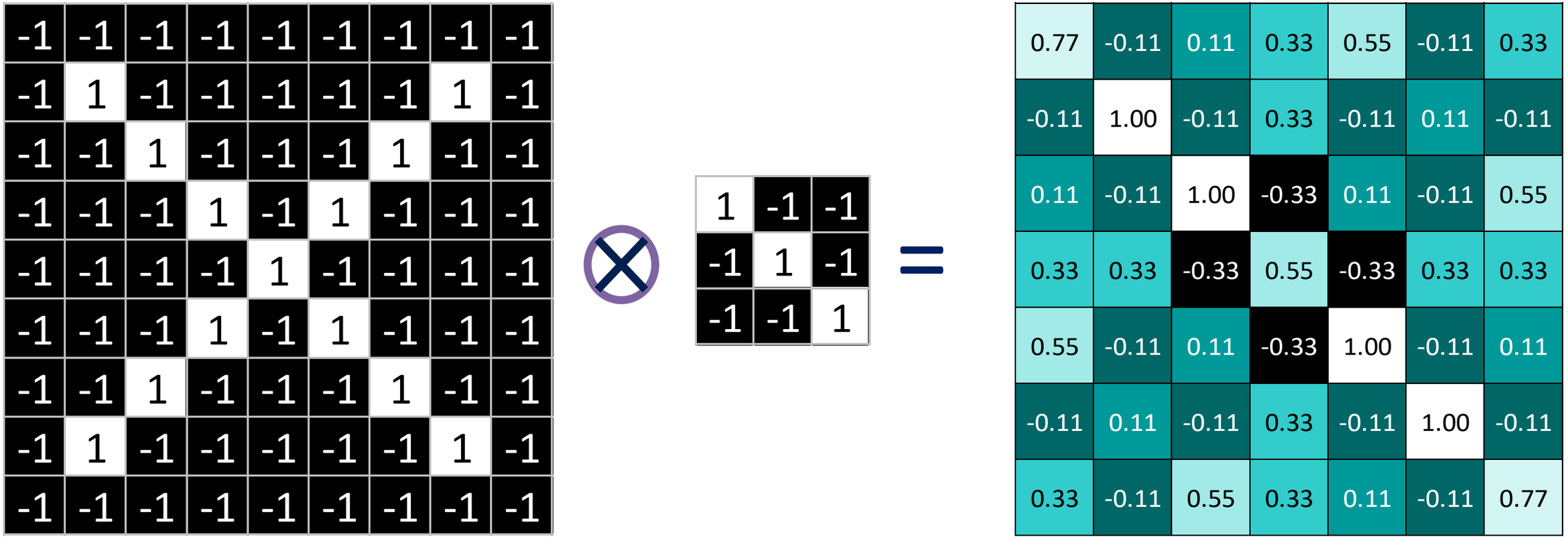
-1 to dopasowanie „w negatywie”

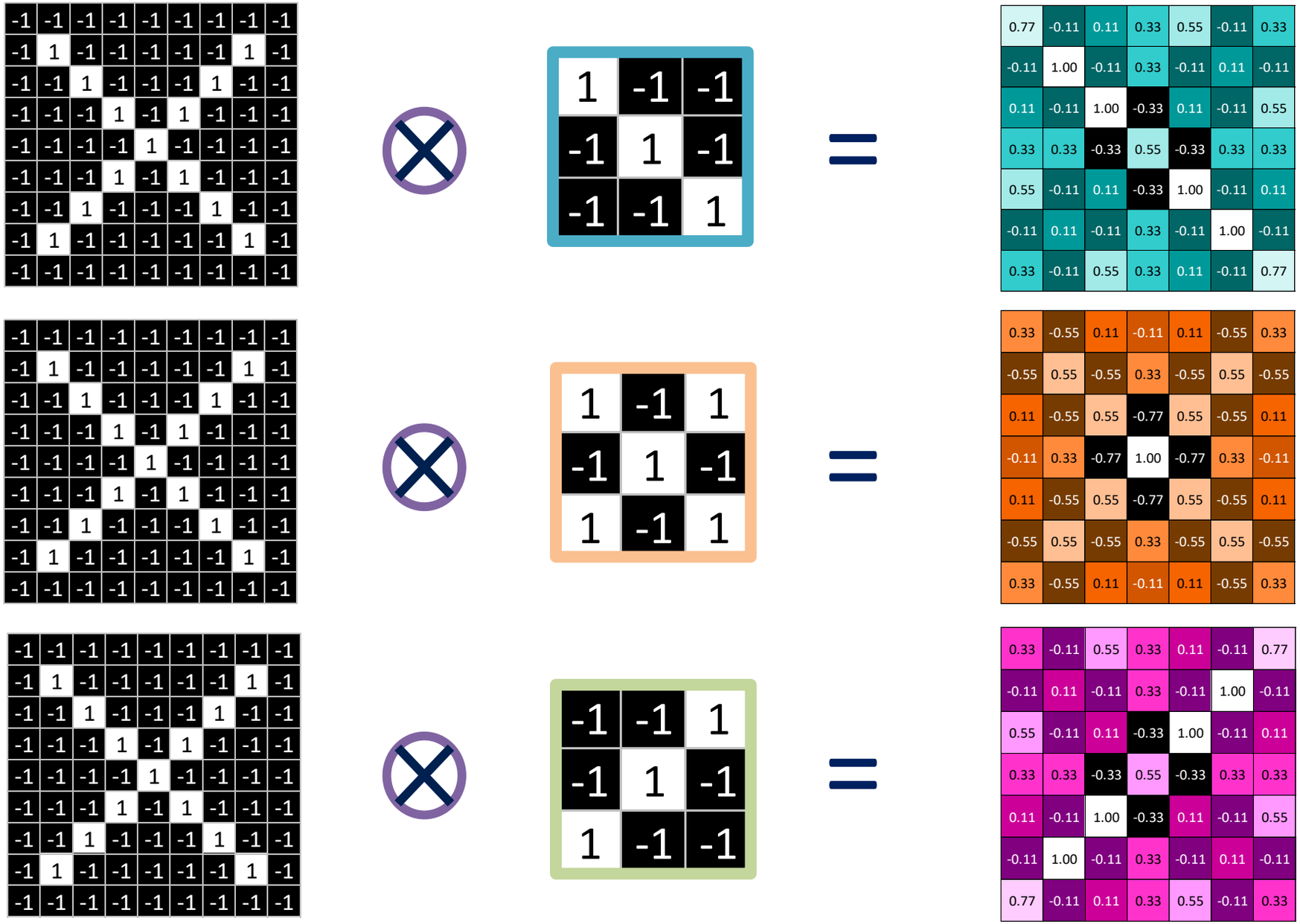
0 to brak dopasowania

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

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

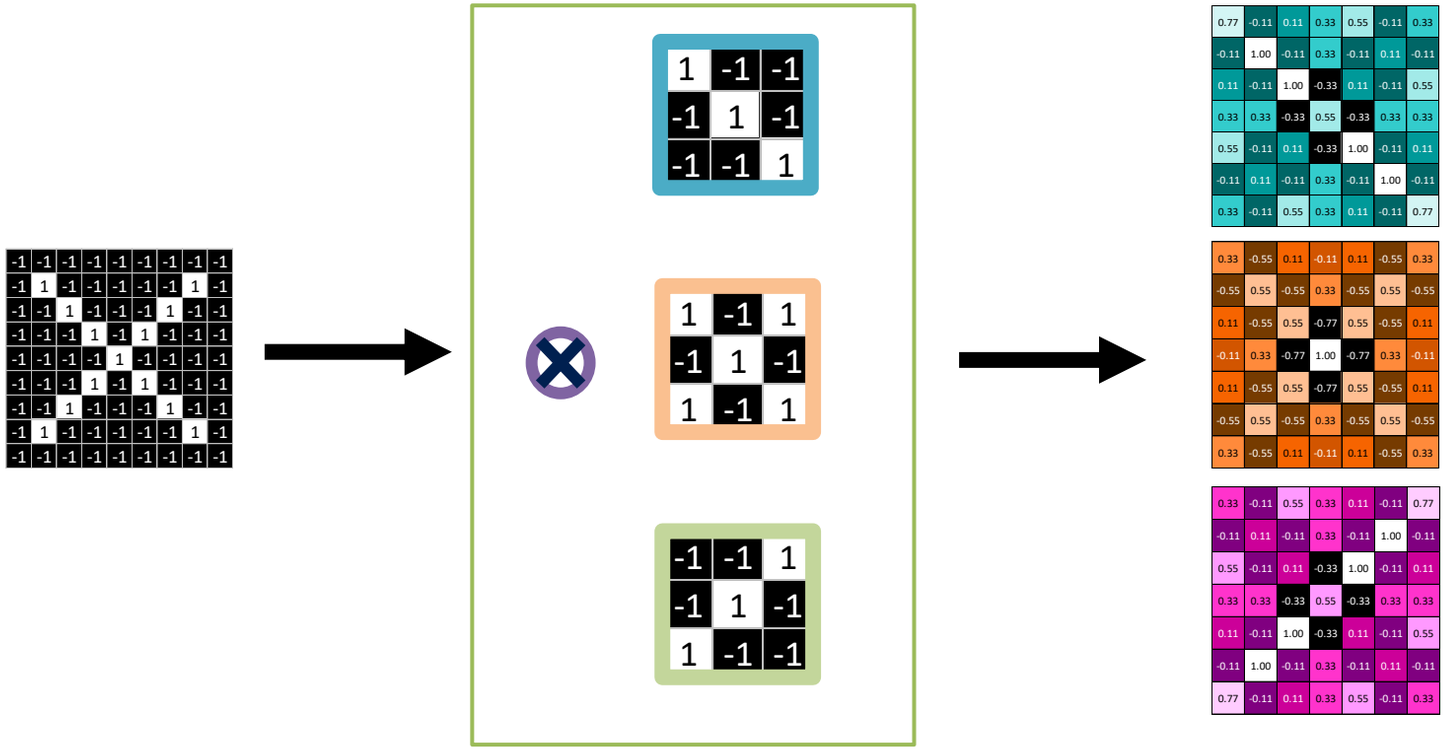
Convolution: Trying every possible match




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

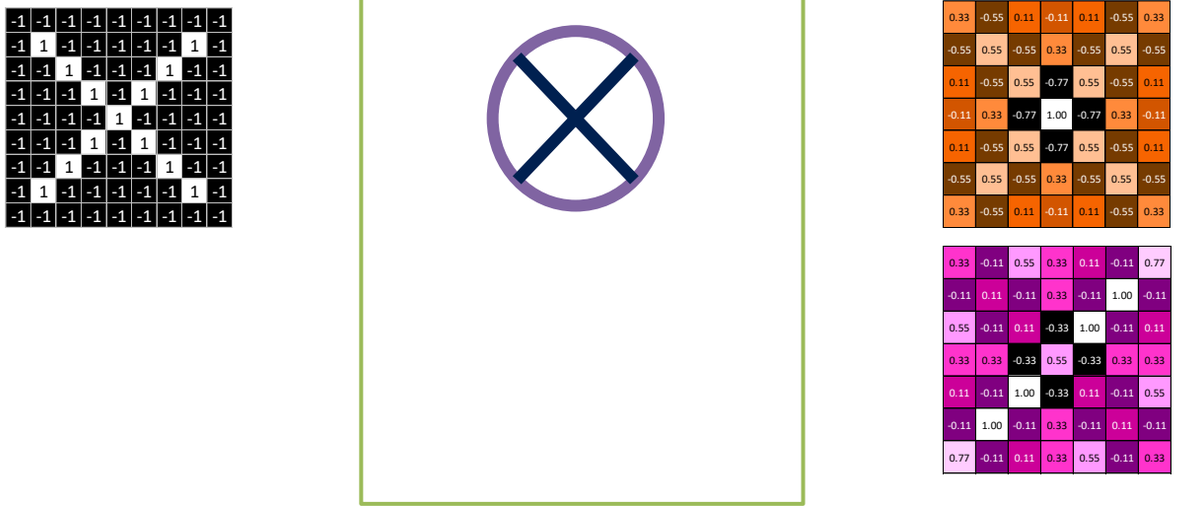
One image becomes a stack of filtered images



Zwróćmy już tutaj uwagę na to, że w przypadku dużej ilości wzorców (filtrów) będzie generowana duża liczba obrazów na wyjściu. Szybko zaczną pojawiać się problemy z wydajnością. CNN mają jednak na to sposoby.

Convolution layer

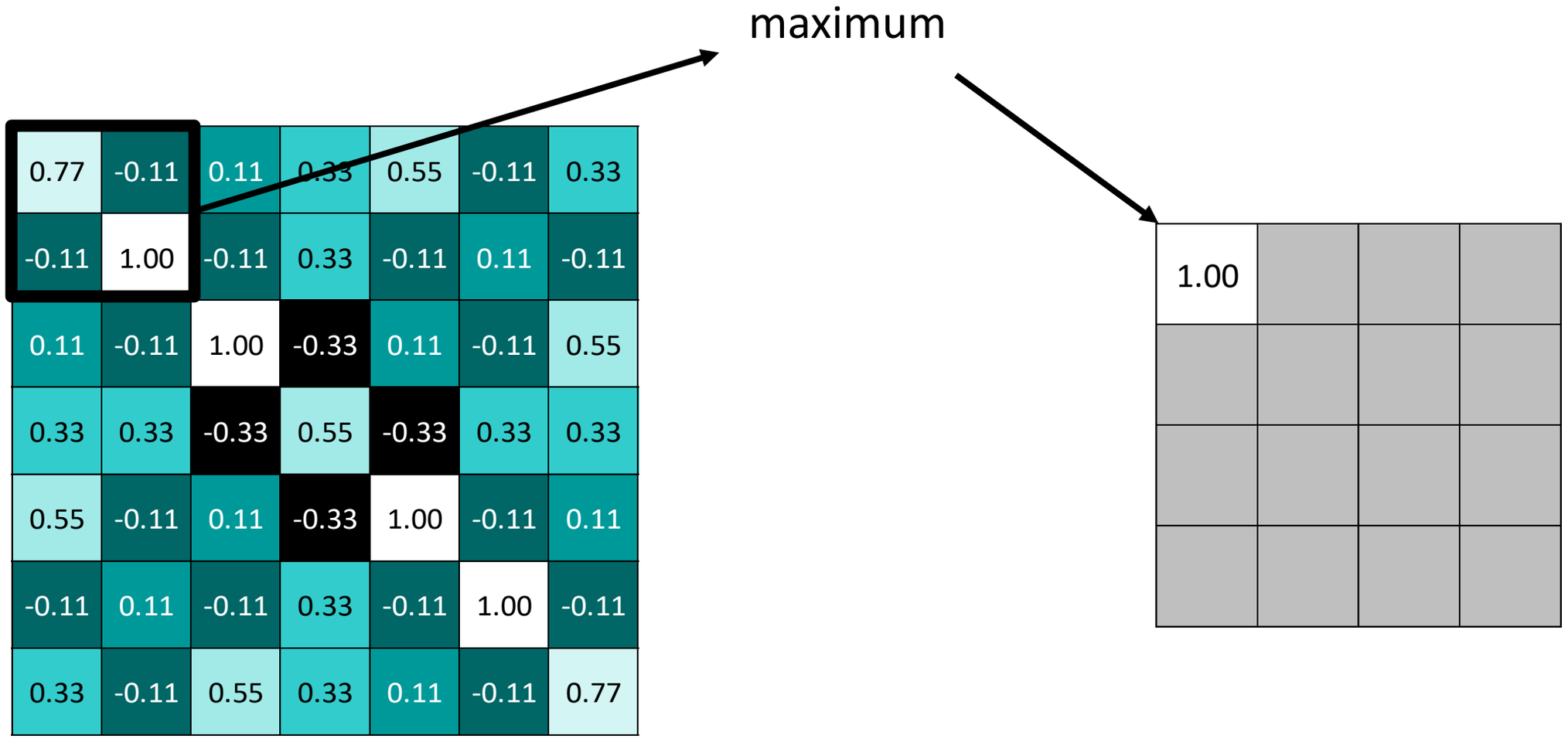
One image becomes a stack of filtered images



Pooling: Shrinking the image stack

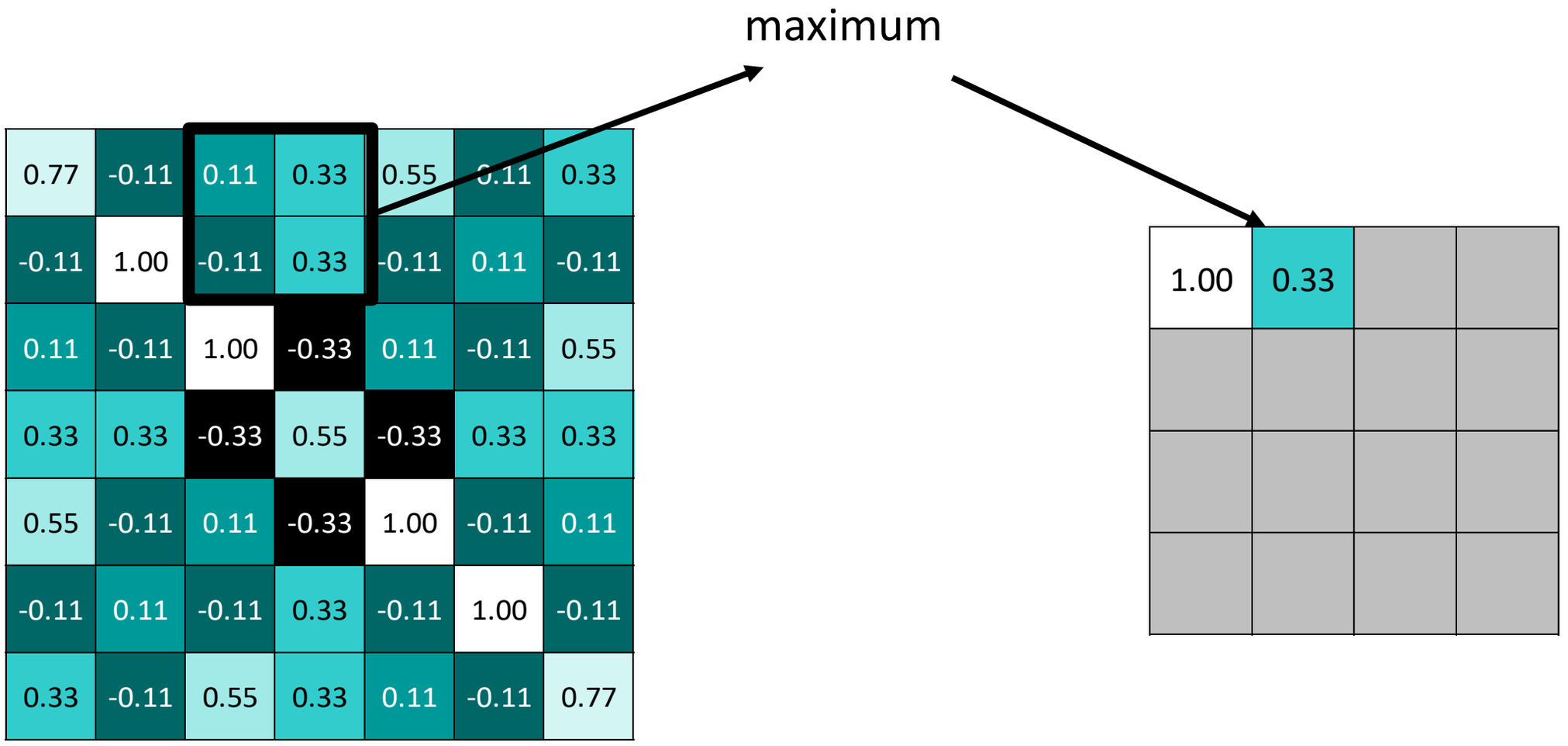
1. Pick a window size (usually 2 or 3).
2. Pick a stride (usually 2).
3. Walk your window across your filtered images.
4. From each window, take the maximum value.

Pooling

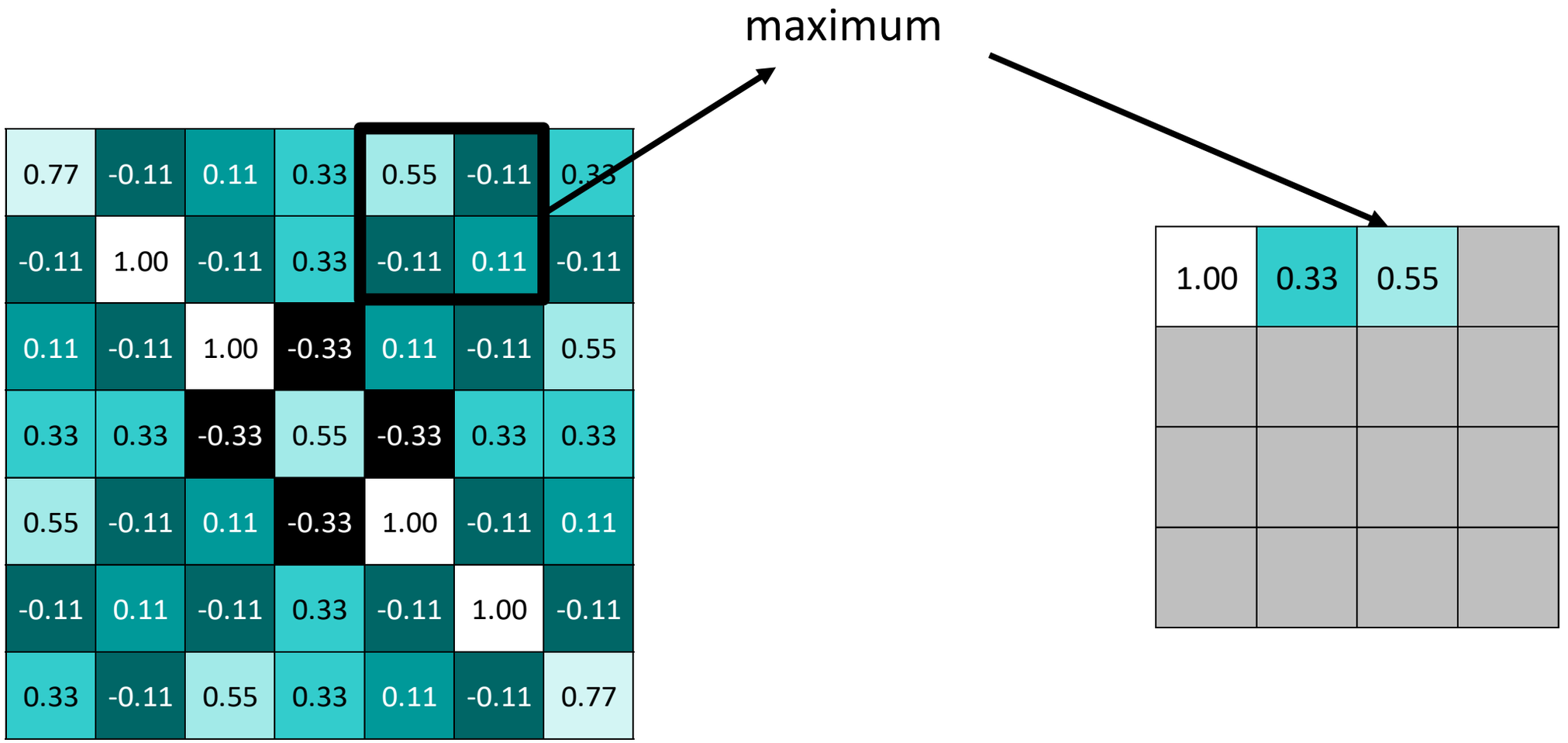


Pooling to bardzo prosta operacja ale jest kluczowa dla efektywnego działania CNN.
Obrazy są zmniejszane, tracimy oczywiście jakąś część informacji ale finalnie okazuje się,
że nie ma to znaczenia dla efektów końcowych (czyli dobrej klasyfikacji obrazów).

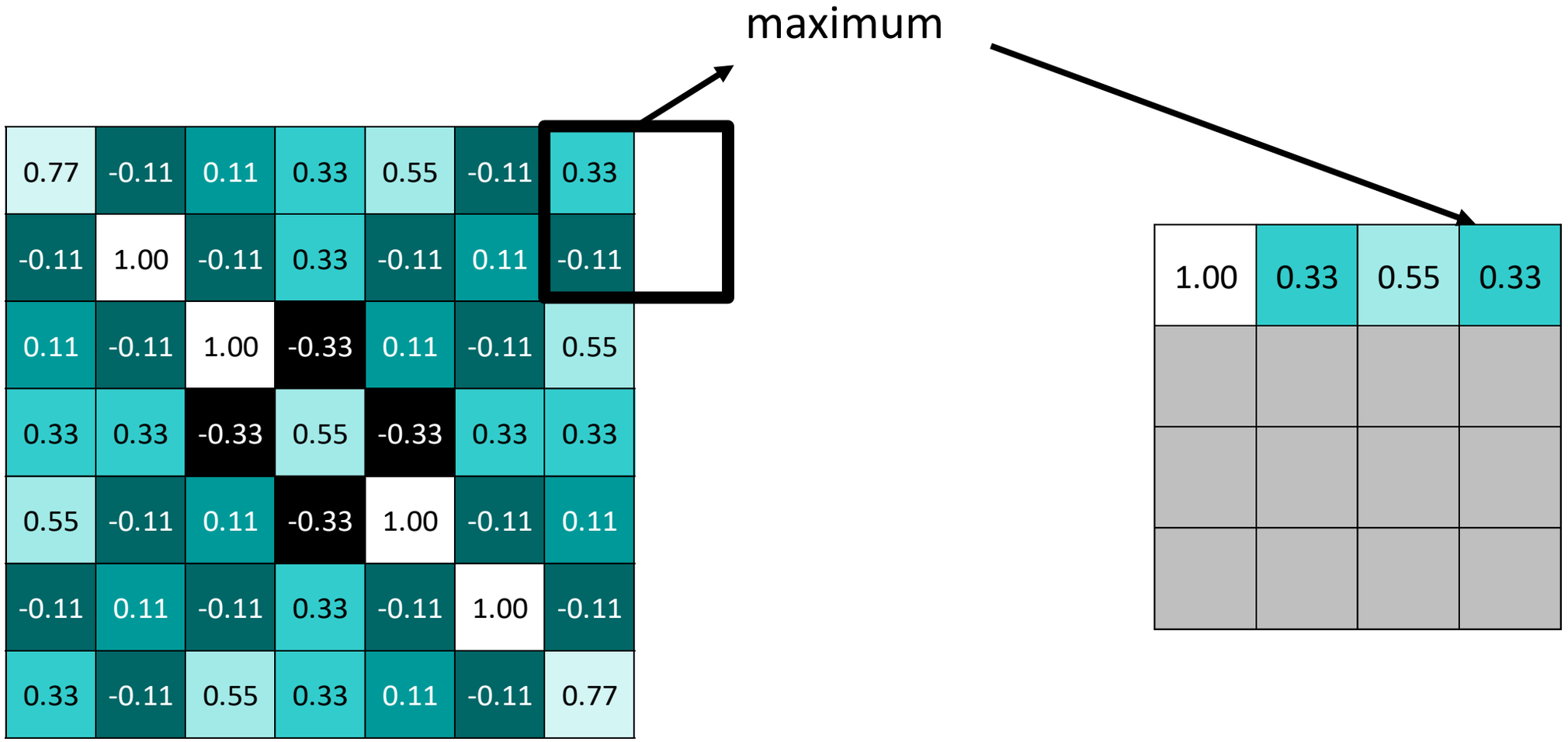
Pooling


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

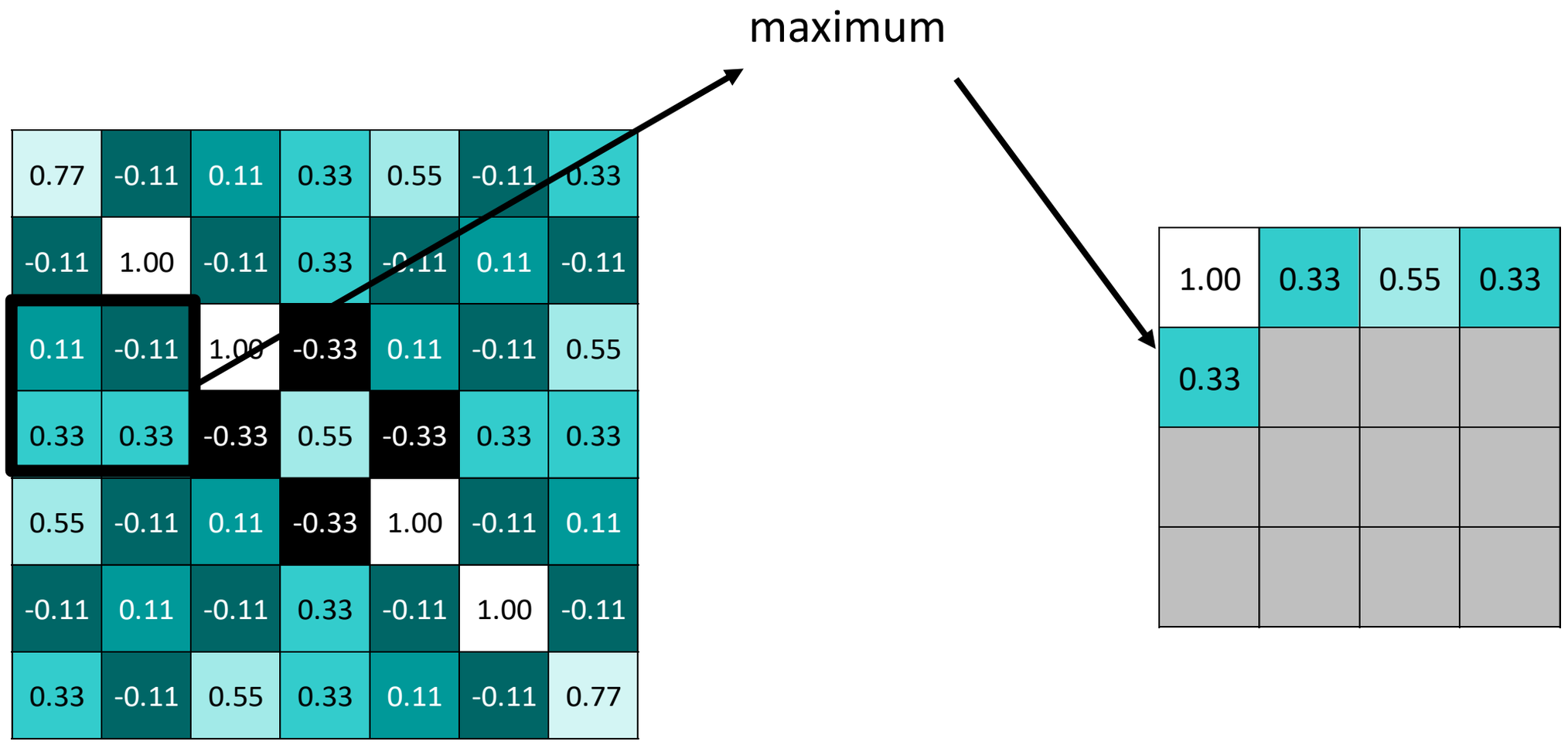
Pooling


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

Pooling


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

Pooling


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

Pooling

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

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

Porównując oba obrazy łatwo dostrzec, że generalny charakter się nie zmienił. Pooling potrafi więc sprawić, że obchodzimy w jakimś stopniu problem zbyt literalnego podchodzenia komputerów do porównywania obrazów.

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

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

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

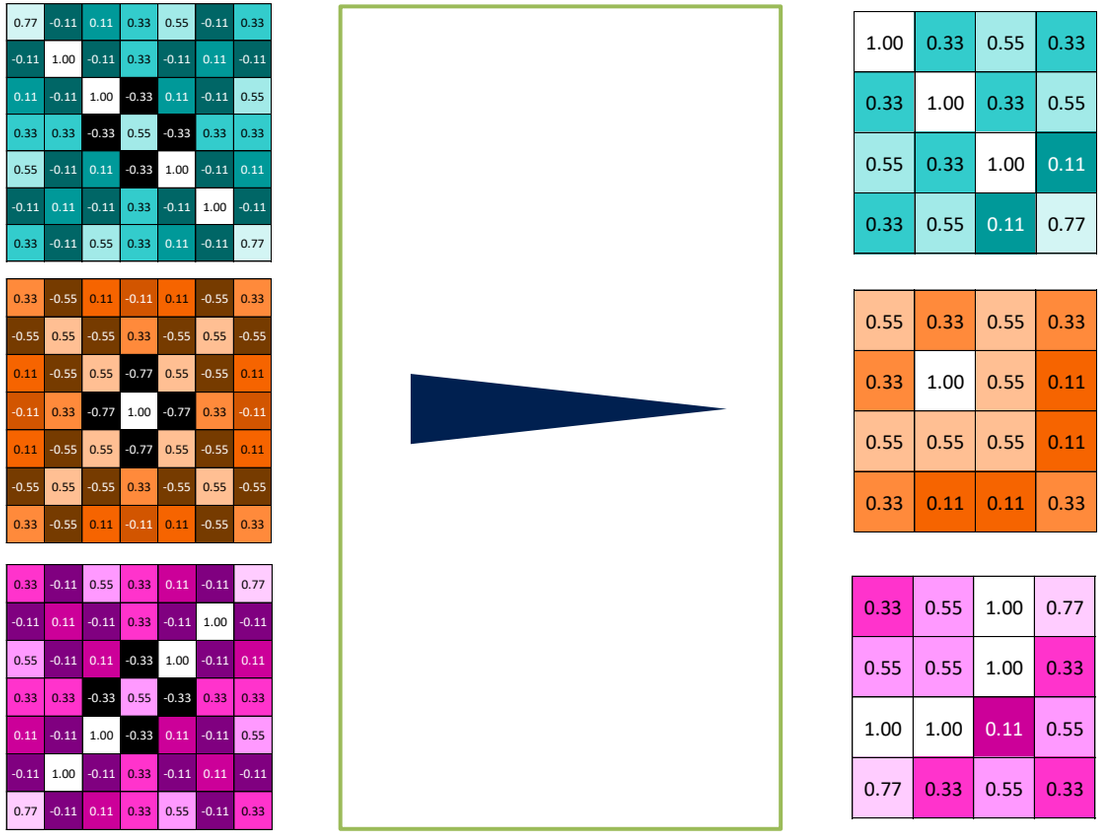
0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

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

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

Pooling layer

A stack of images becomes a stack of smaller images.

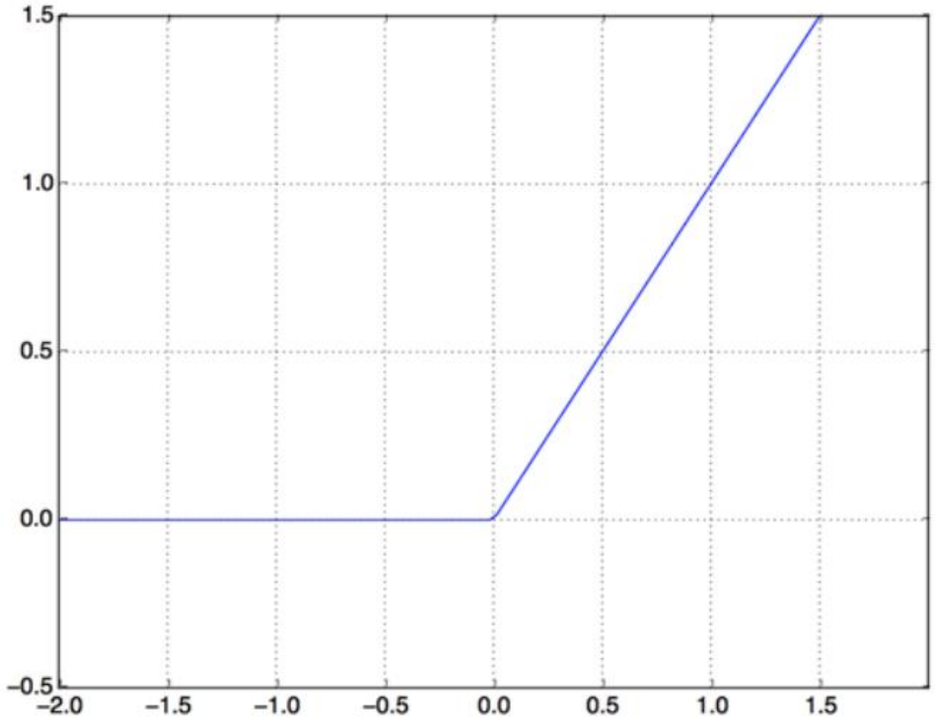


Normalization

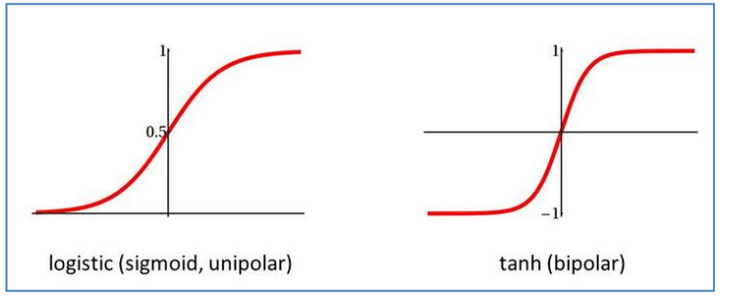
Keep the math from breaking by tweaking each of the values just a bit.

Change everything negative to zero.

Rectified Linear Unit (ReLU)

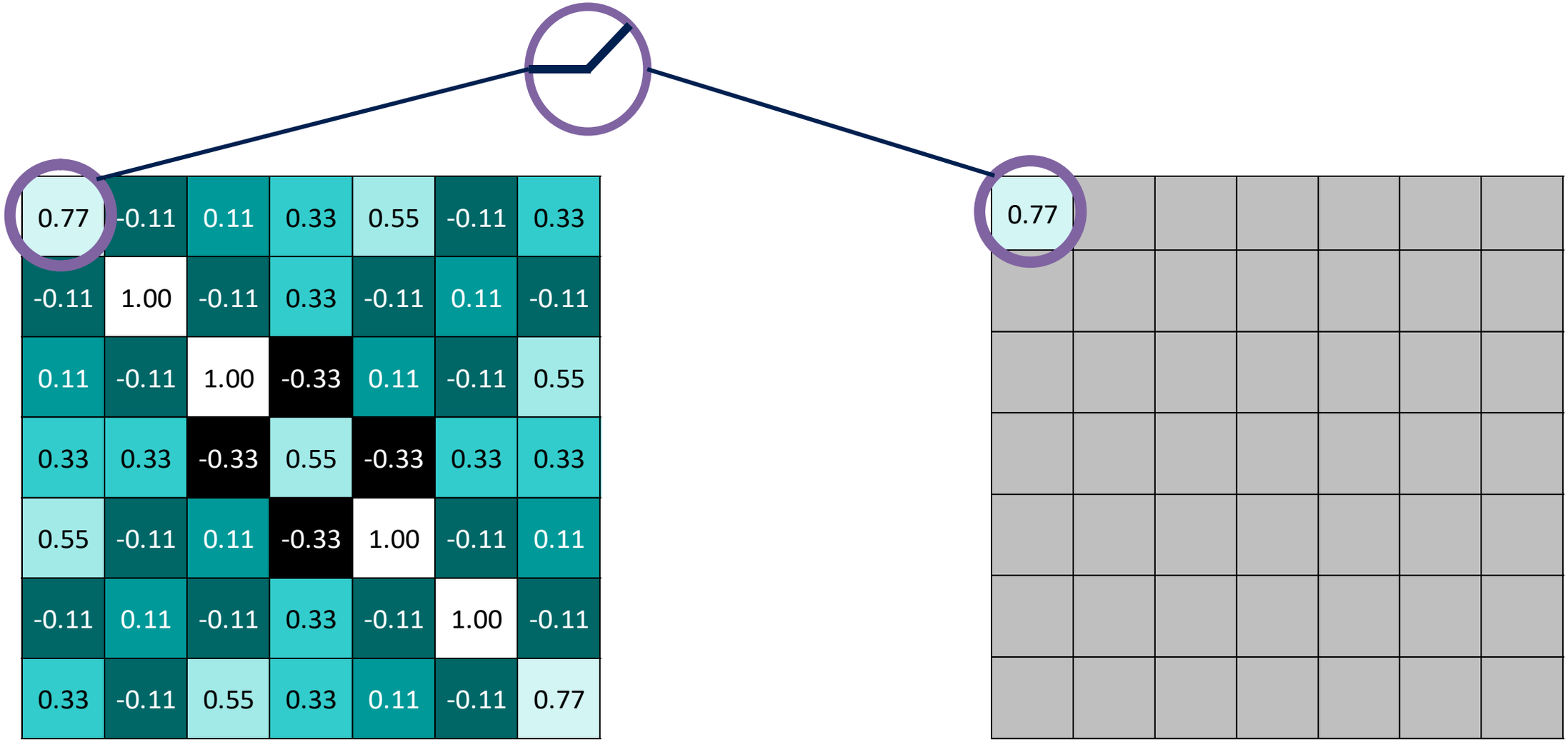


to rectify – sprostować, naprawić, korygować
naprawiać, reperować

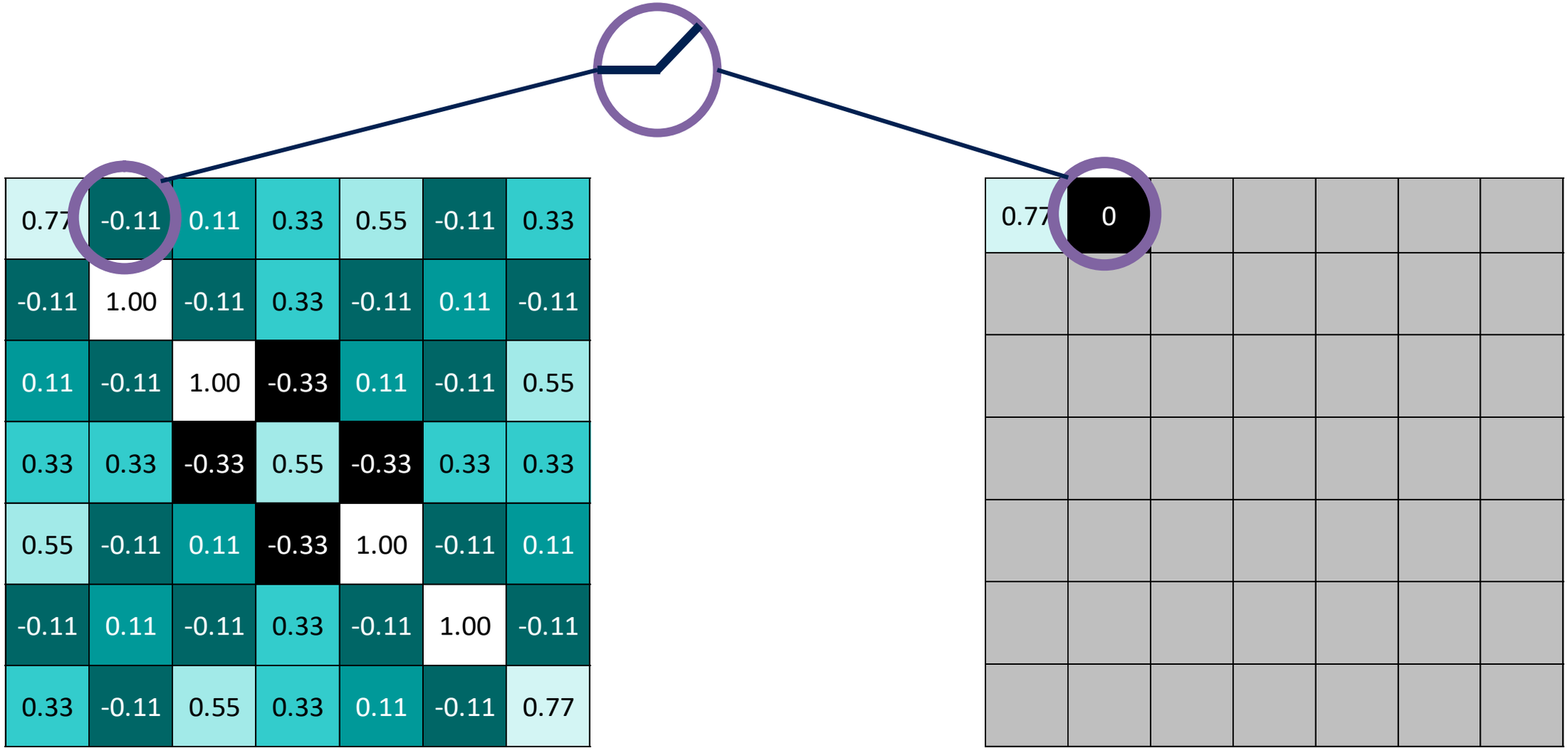


Funkcja ReLU jest funkcją nieliniową. Bez wprowadzenia do CNN nieliniowości mogłaby ona uczyć się tylko przekształceń liniowych i wówczas stosowanie wielu warstw w sieci CNN byłoby bardzo dyskusyjne. Wielowarstwowa sieć liniowa nadal potrafi tylko przekształcać dane liniowo.
Dużo większe możliwości dają sieci, które mają charakter nieliniowy.
Są też oczywiście inne funkcje nieliniowe (np. sigmoidalna, tangens hiperboliczny) ale w dziedzinie CNN nie są one powszechnie stosowane (bo nie dają zauważalnych korzyści).

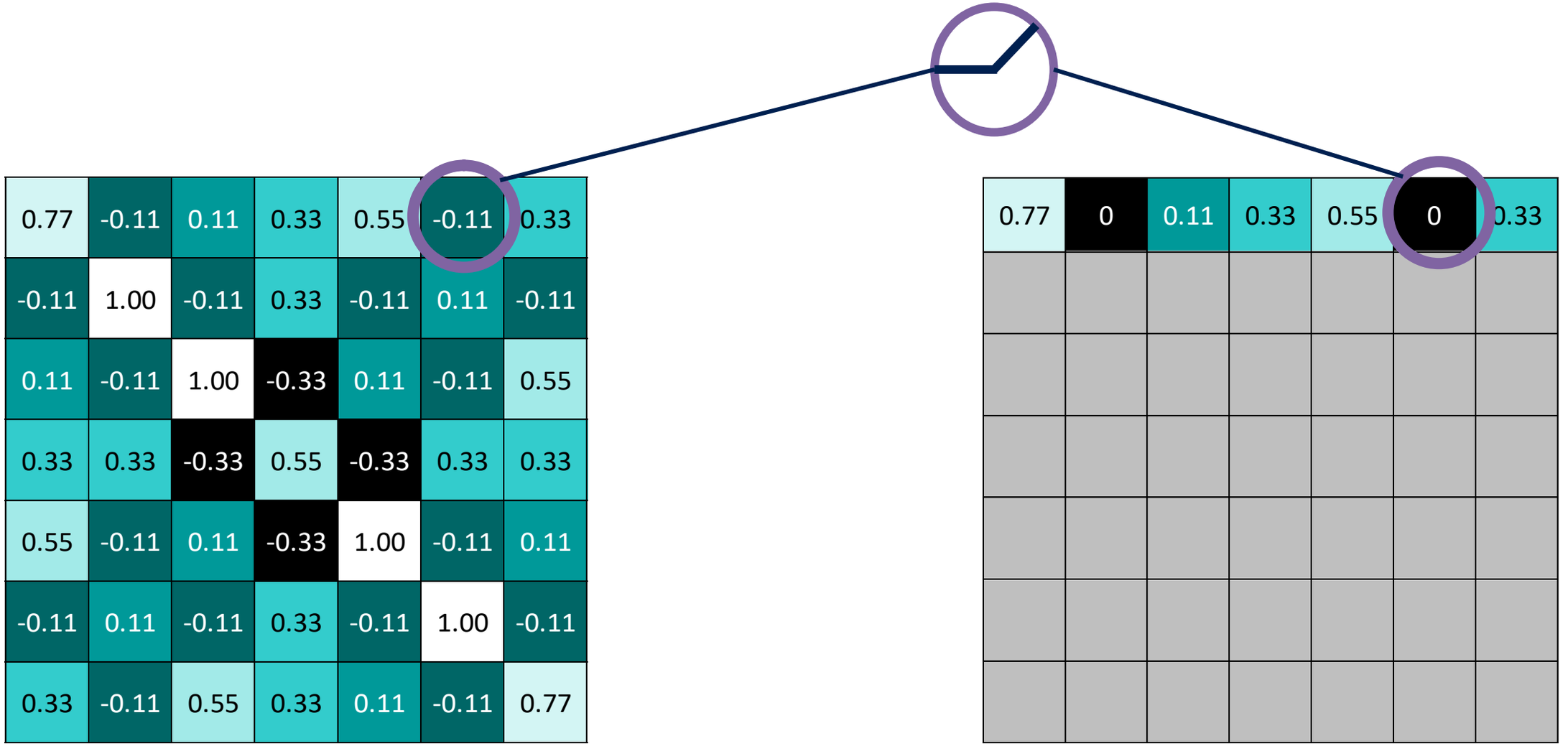
Rectified Linear Units (ReLUs)



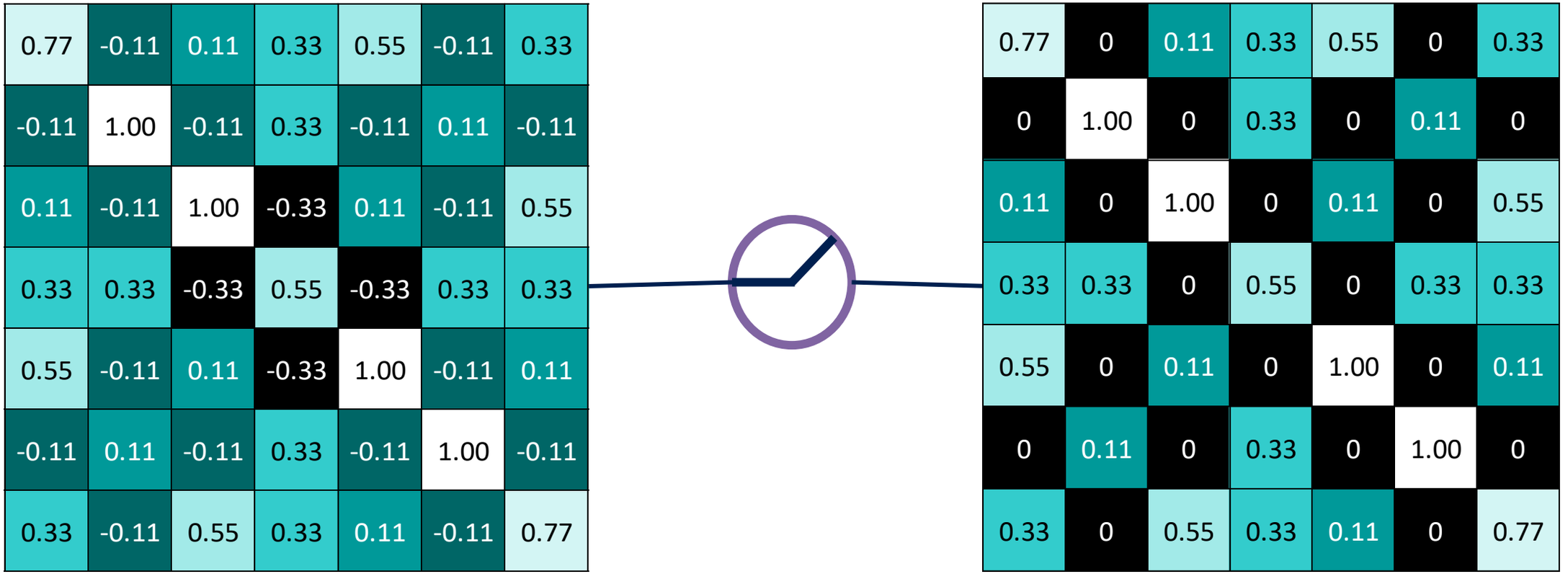
Rectified Linear Units (ReLUs)



Rectified Linear Units (ReLUs)



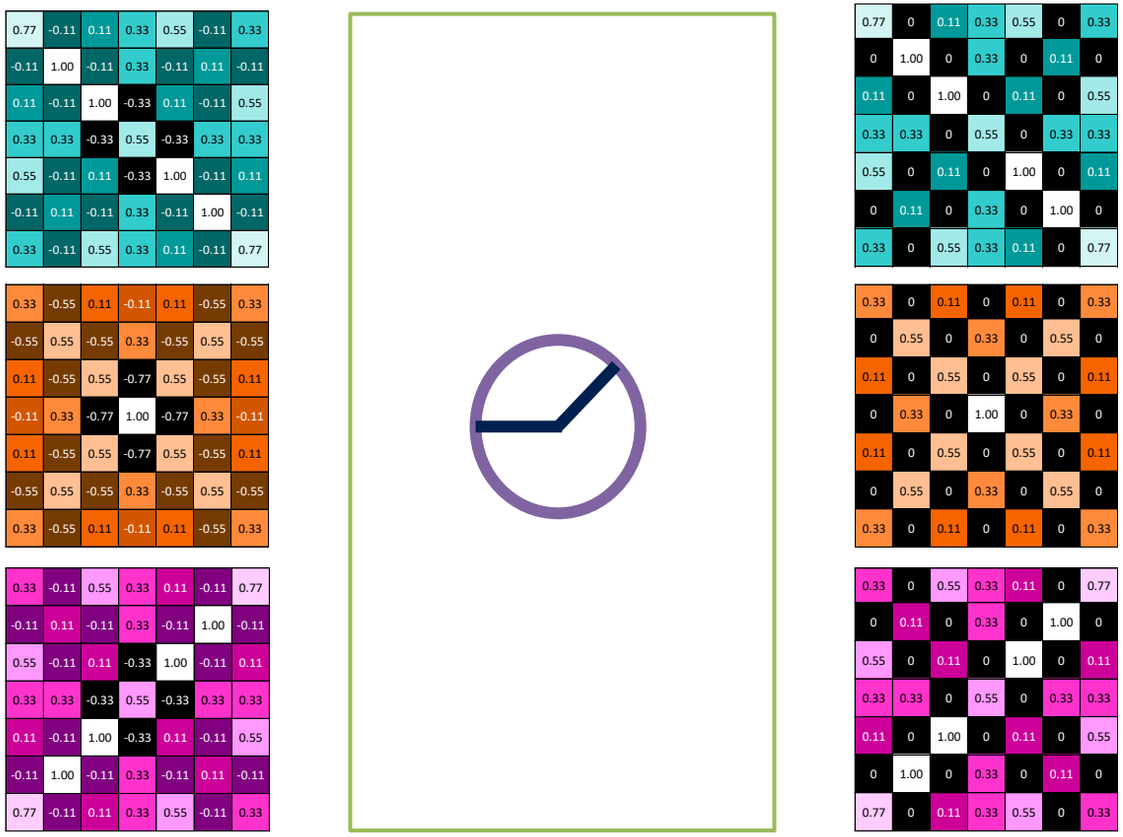
Rectified Linear Units (ReLUs)



Ujemne wartości zostały wyzerowane.

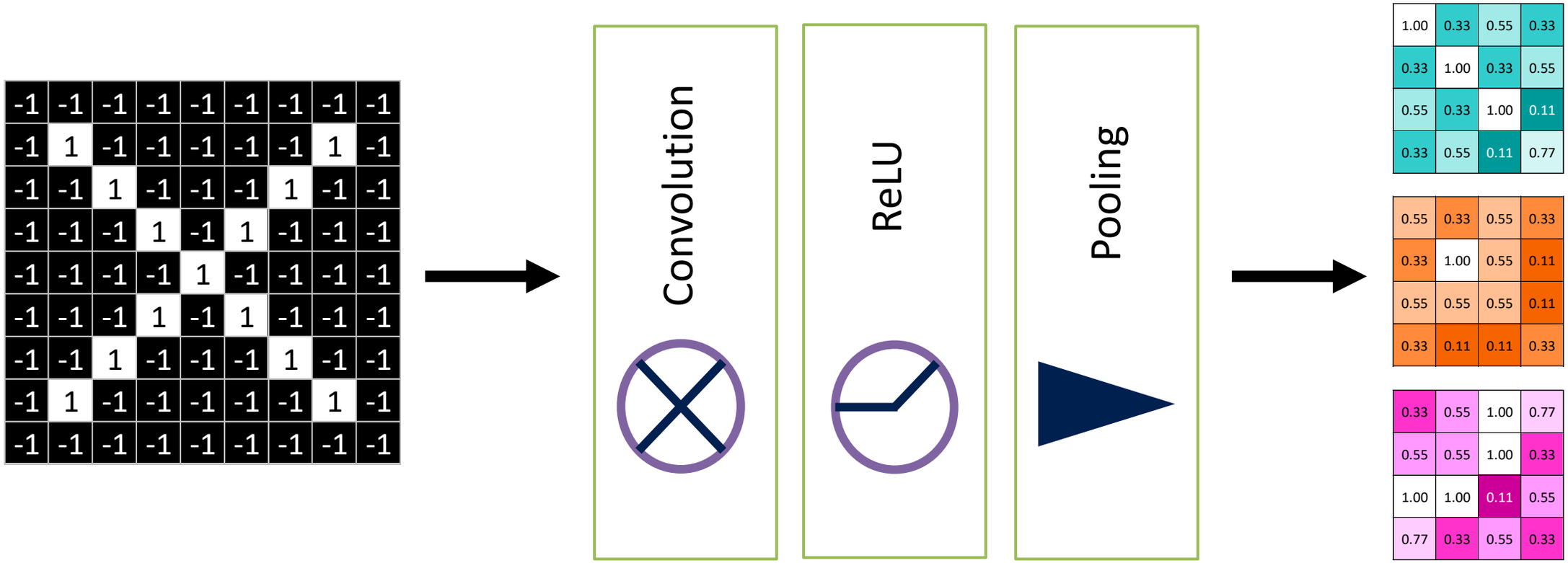
ReLU layer

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



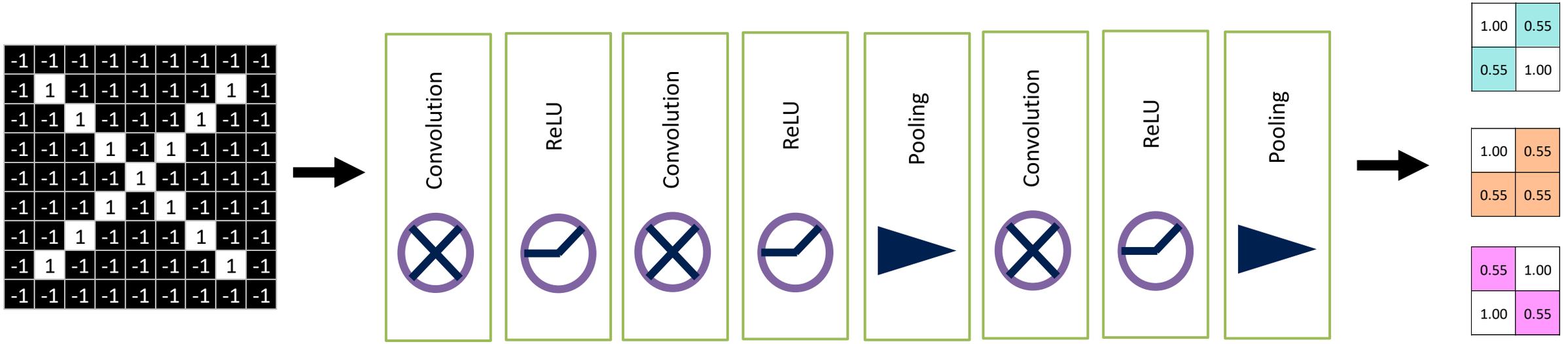
Layers get stacked

The output of one becomes the input of the next.



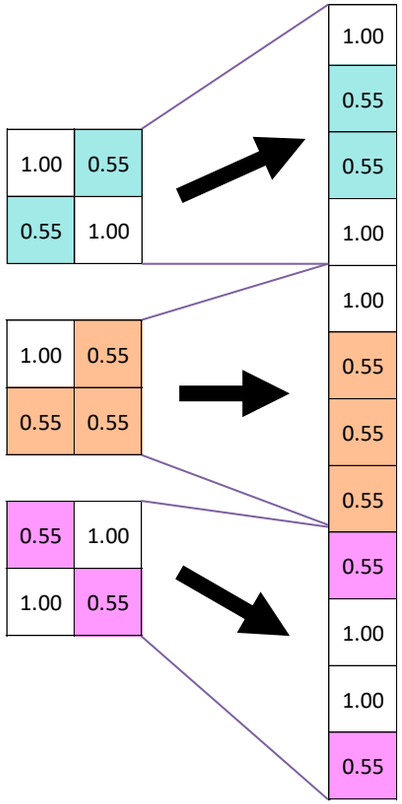
Deep stacking

Layers can be repeated several (or many) times.



Fully connected layer

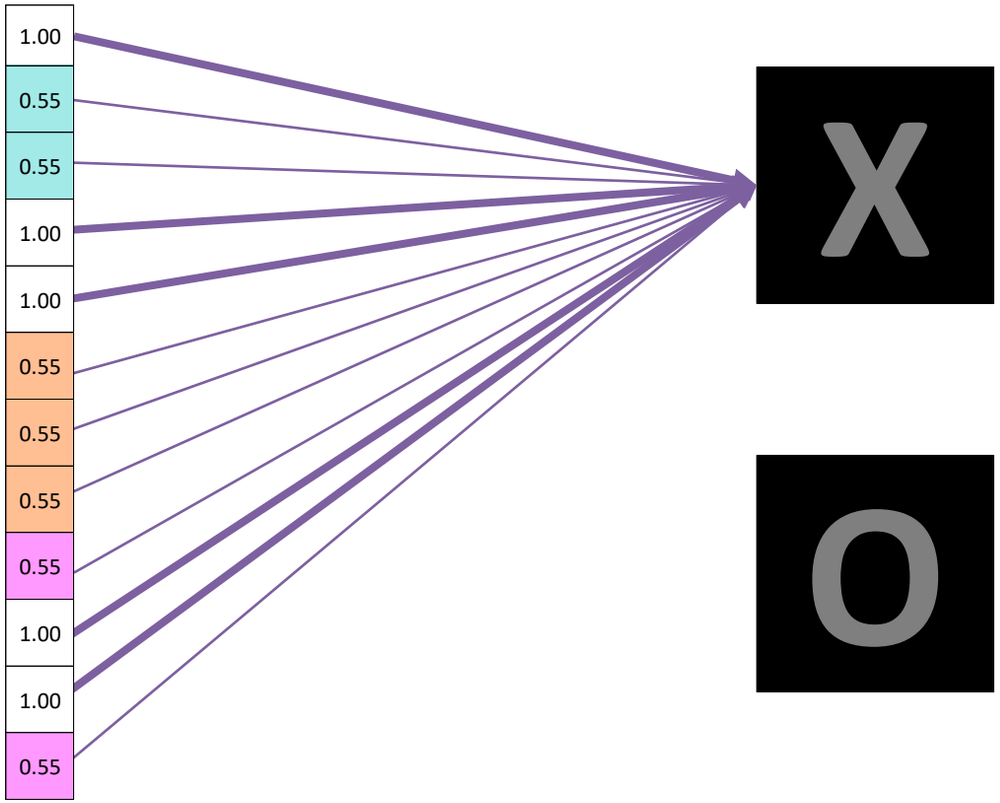
Every value gets a vote



Dwuwymiarowe obrazy wyjściowe przekształcamy do postaci wektora. Wektor ten stanie się następnie wejściem dla klasycznej sieci neuronowej (tzw. warstwa dense)

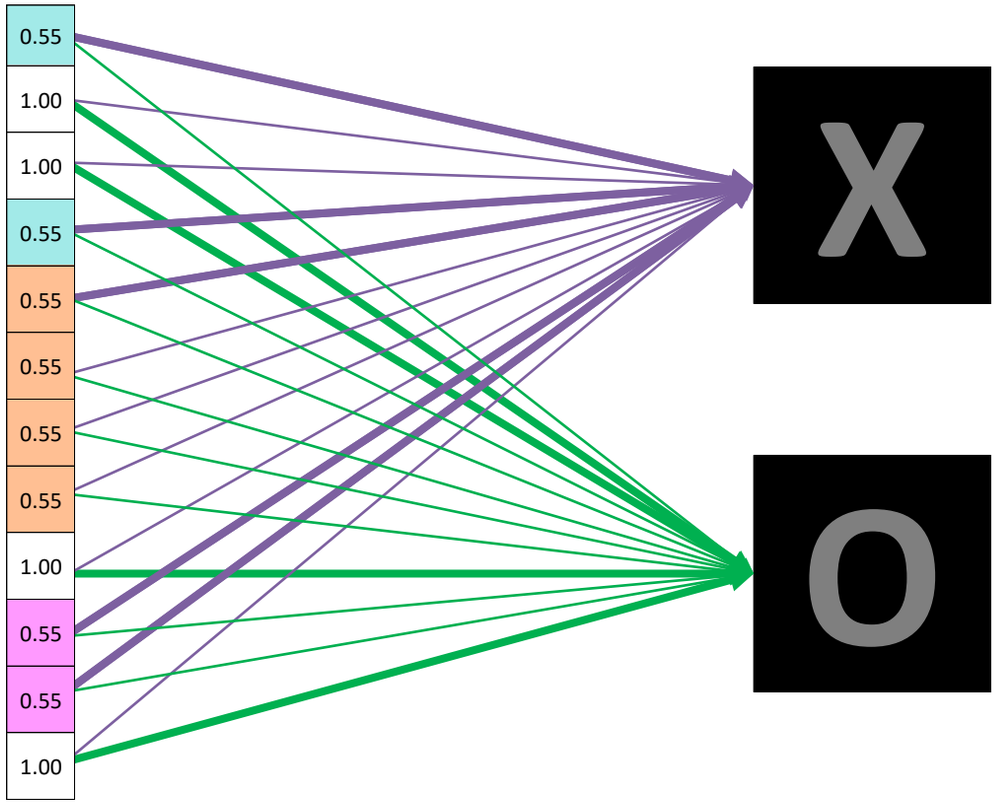
Fully connected layer

Vote depends on how strongly a value predicts X or O



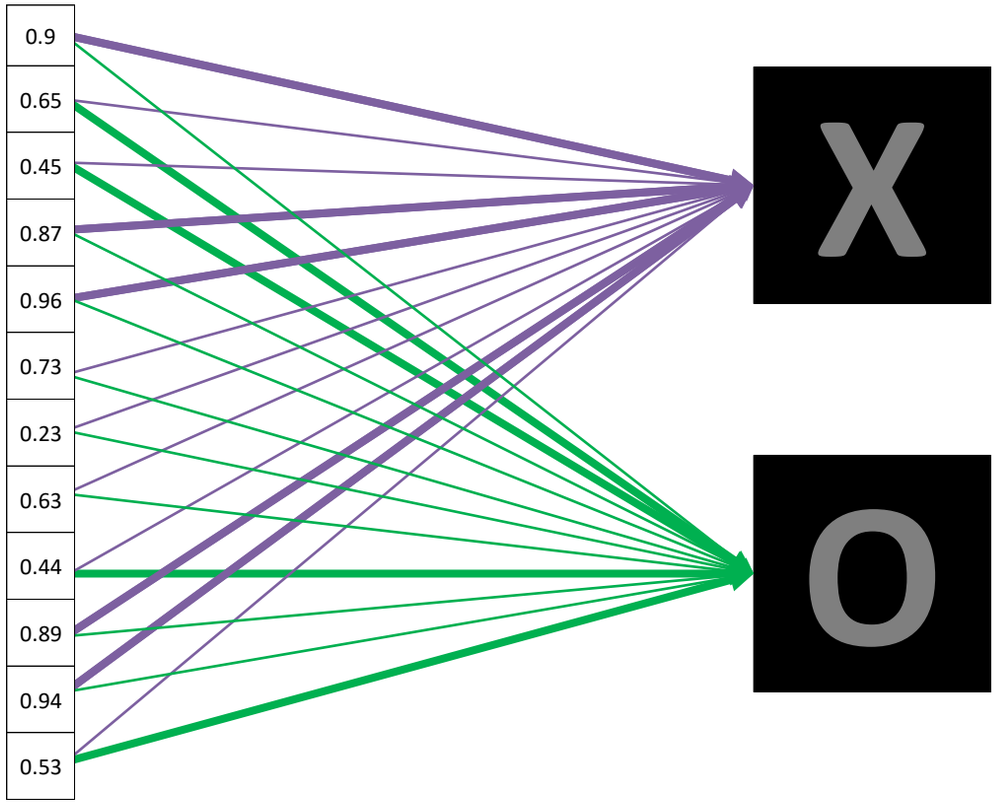
Fully connected layer

Vote depends on how strongly a value predicts X or O



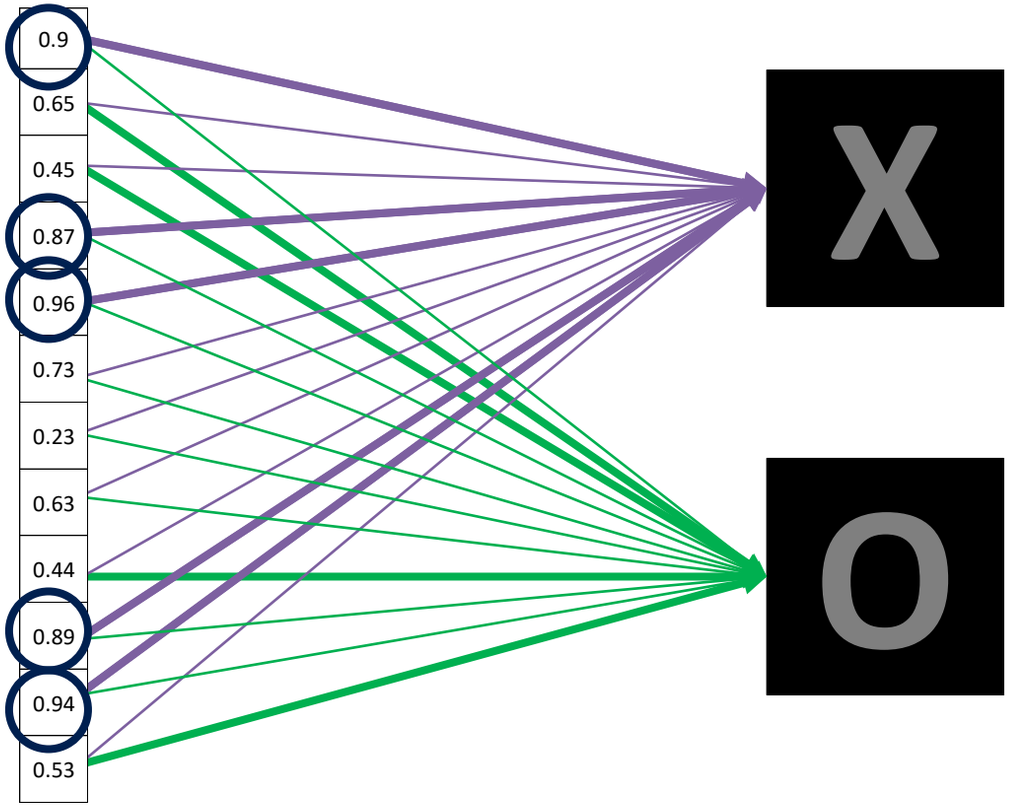
Fully connected layer

Future values vote on X or O



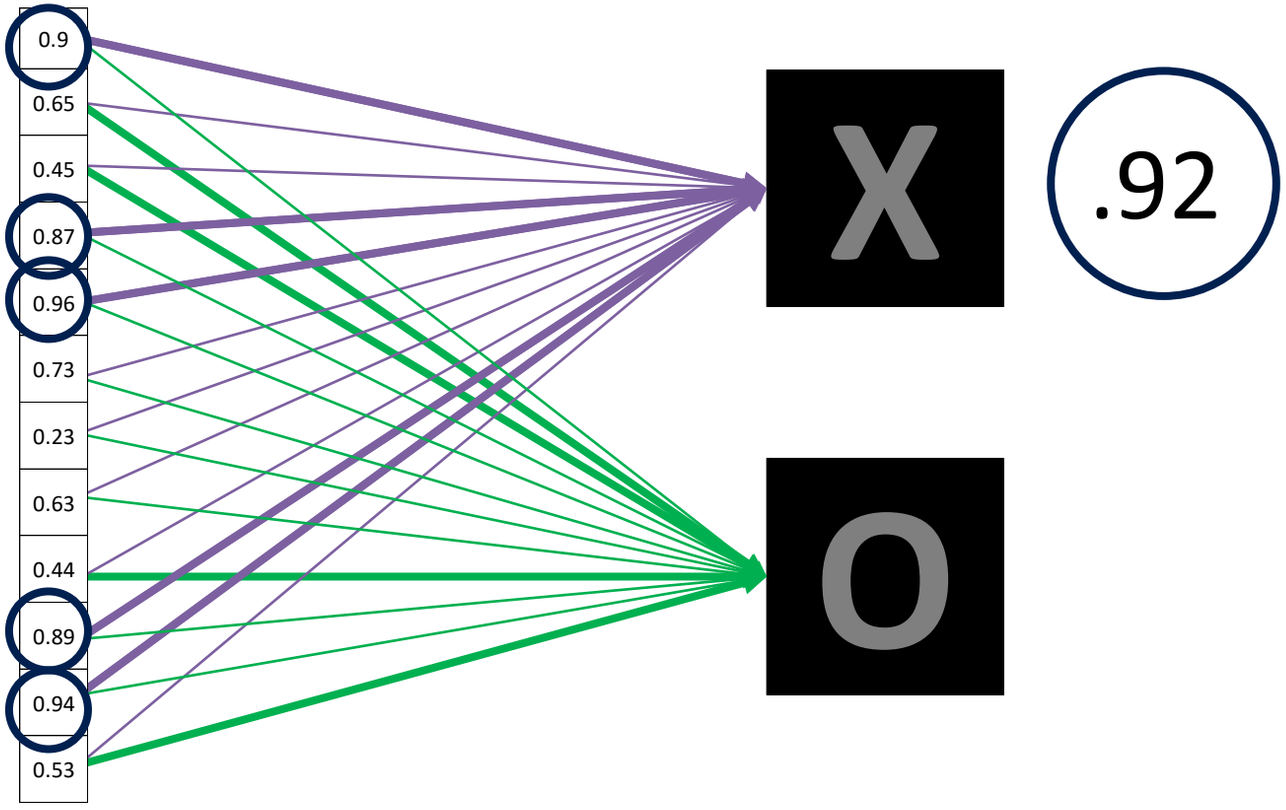
Fully connected layer

Future values vote on X or O



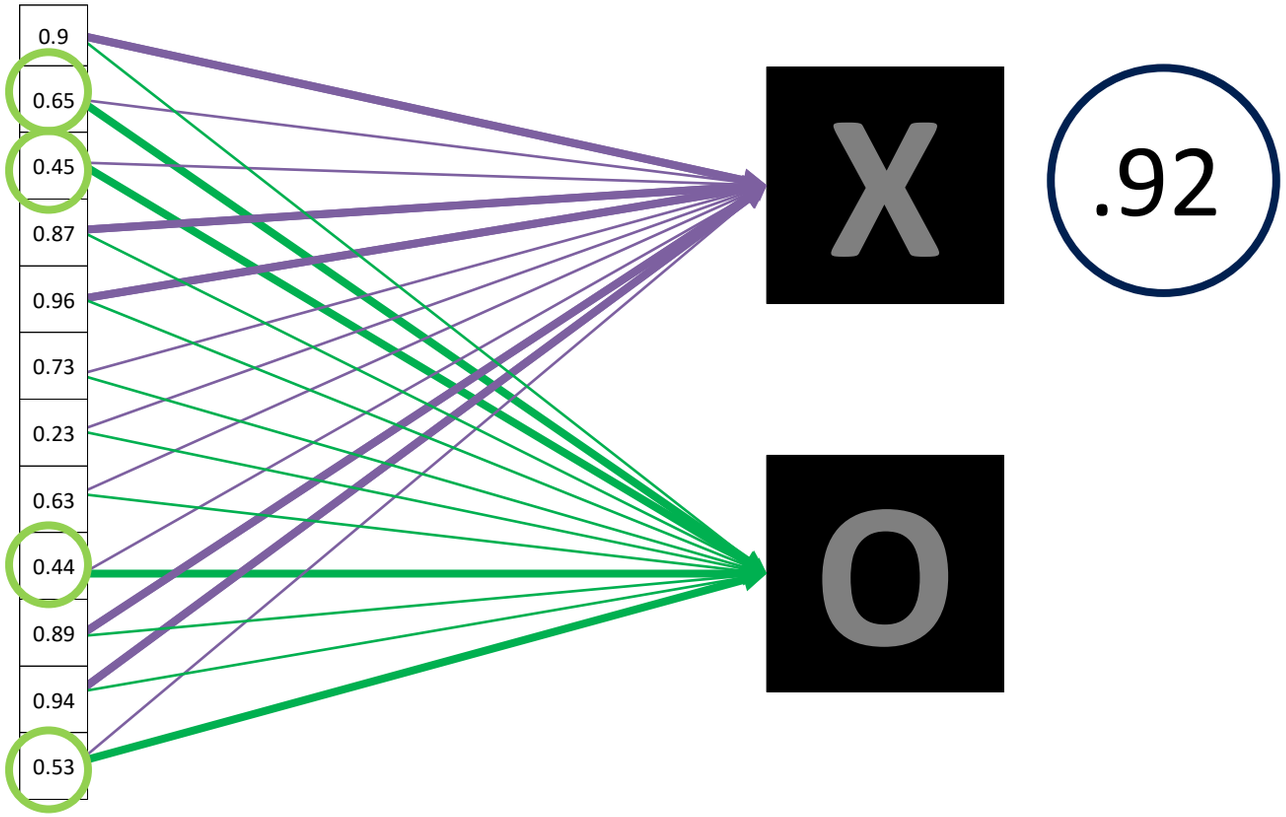
Fully connected layer

Future values vote on X or O



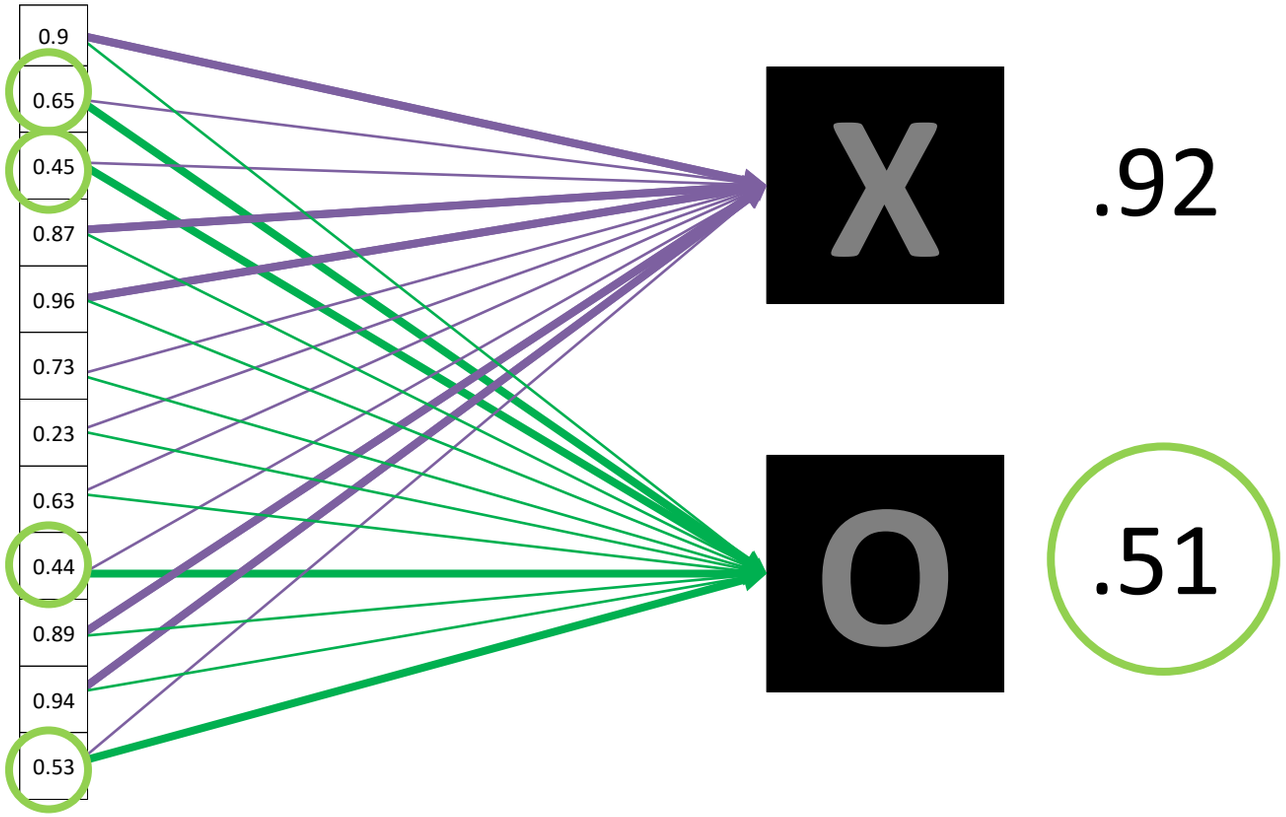
Fully connected layer

Future values vote on X or O



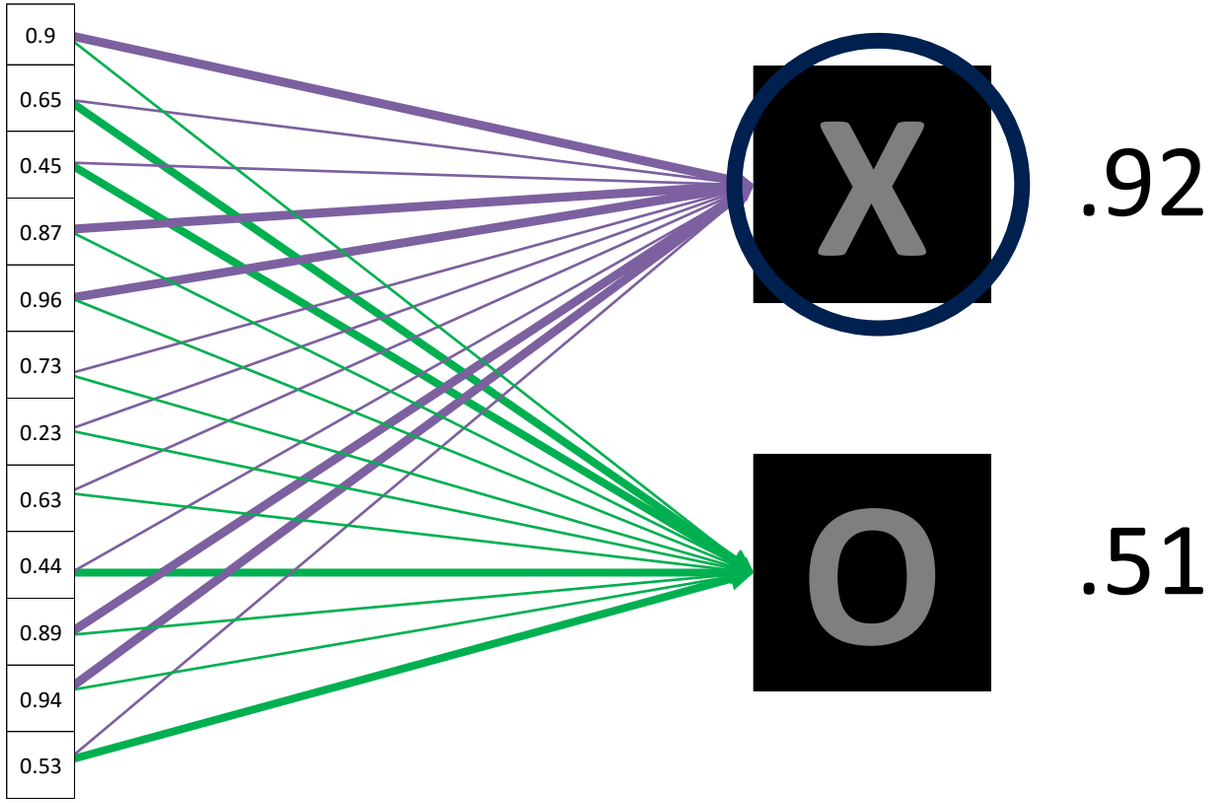
Fully connected layer

Future values vote on X or O



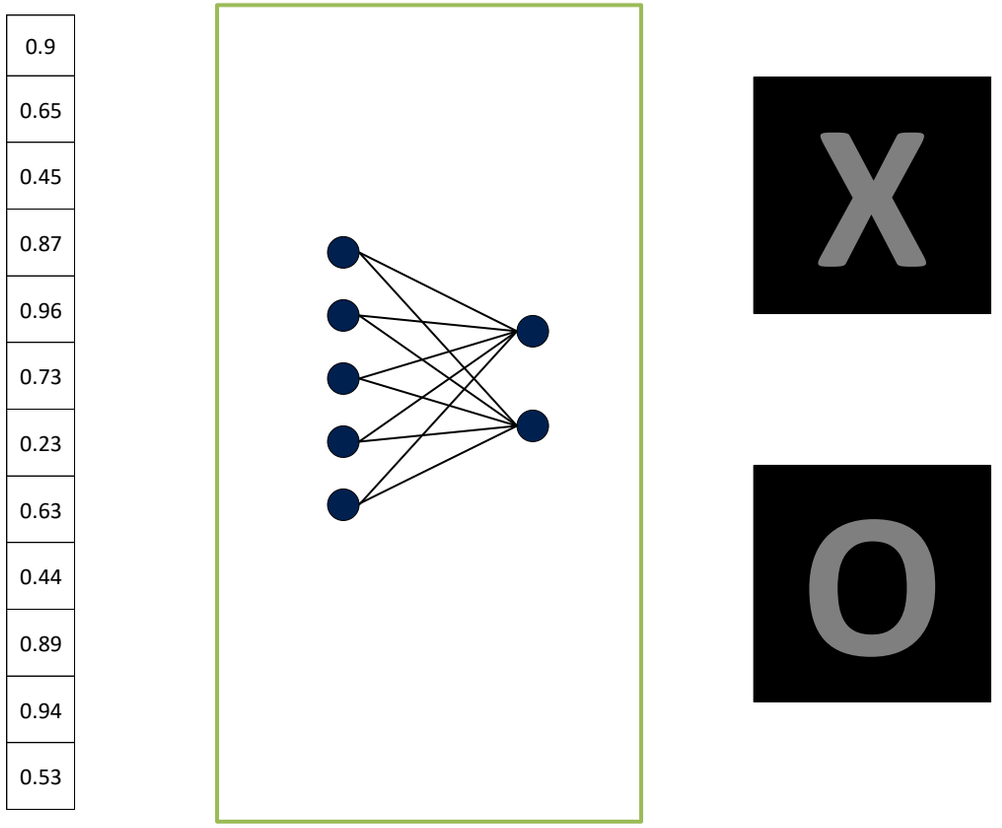
Fully connected layer

Future values vote on X or O



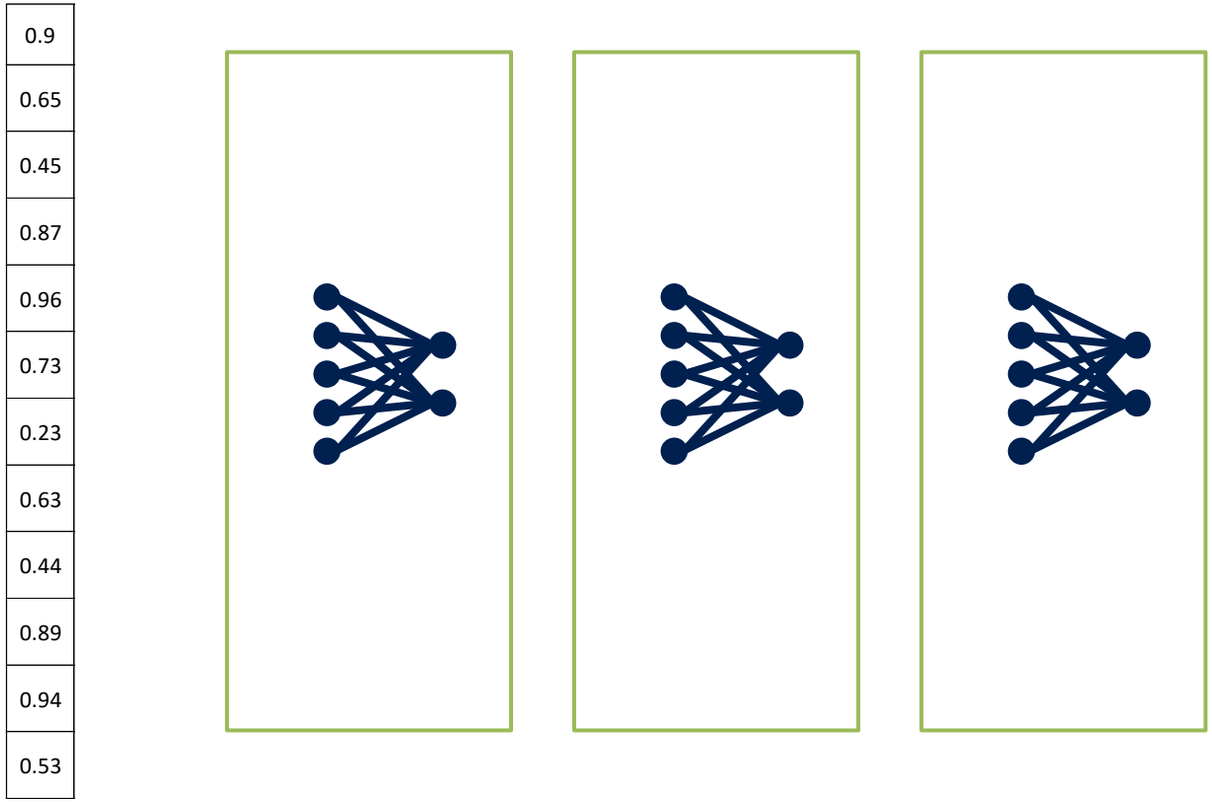
Fully connected layer

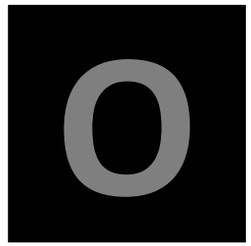
A list of feature values becomes a list of votes.



Fully connected layer

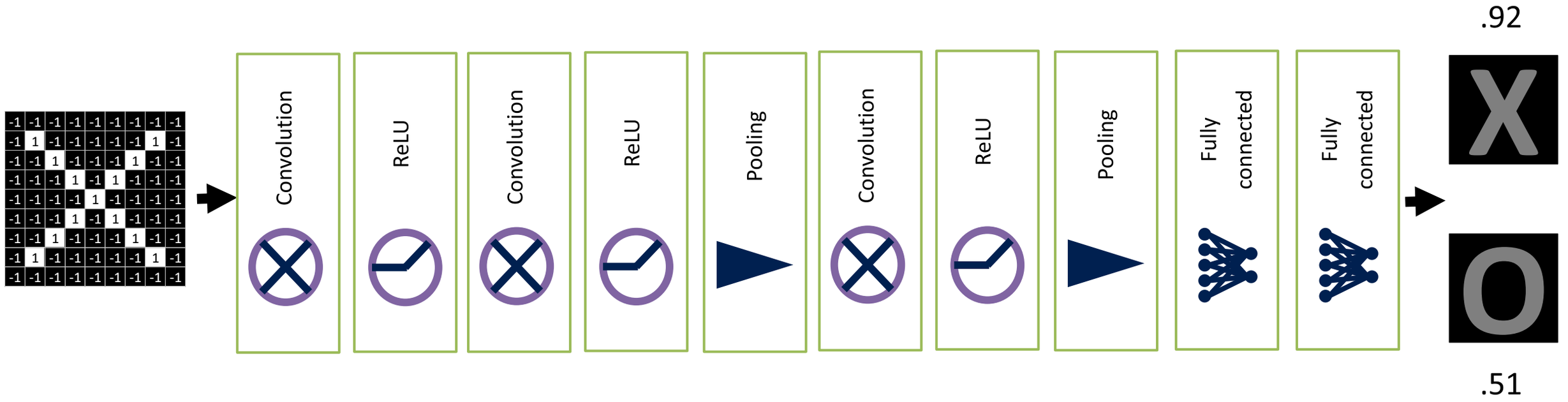
These can also be stacked.





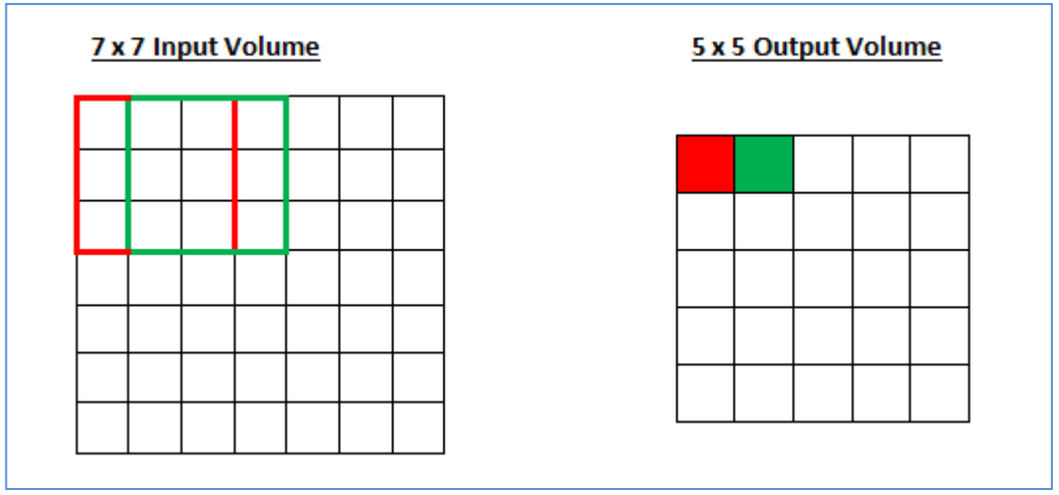
Putting it all together

A set of pixels becomes a set of votes.

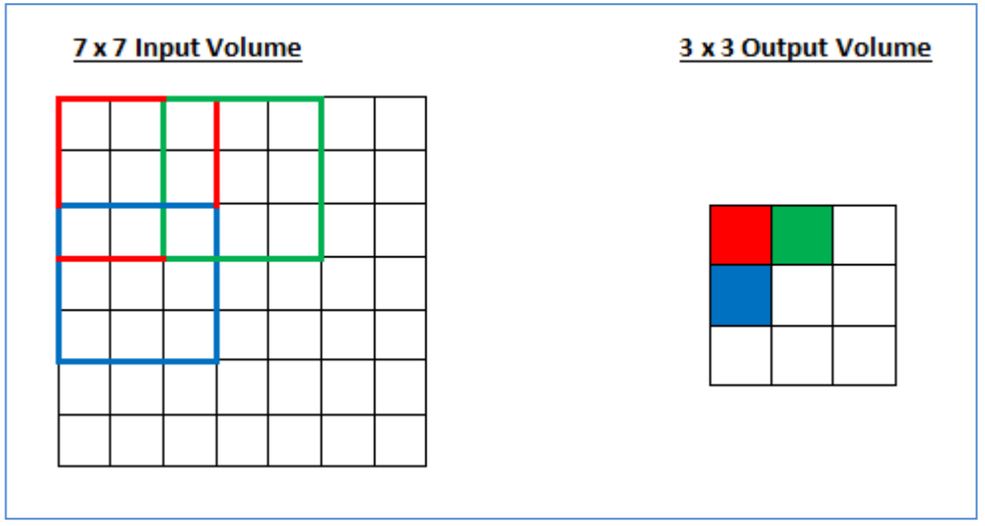


Stride (krok)

Zwykle ma wartość 1 ale można ustawić na więcej



stride = 1

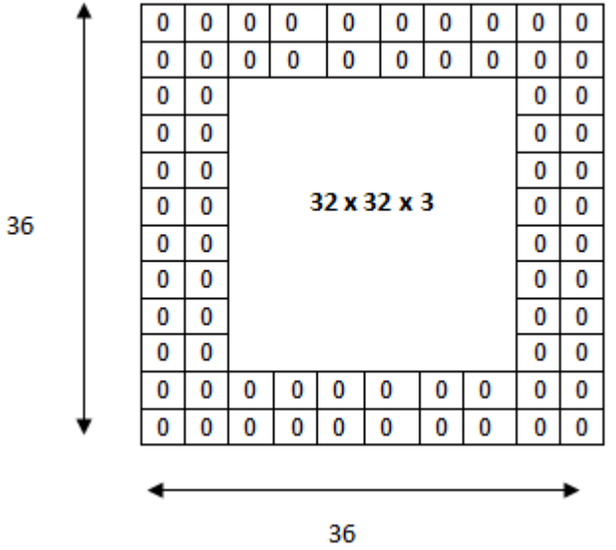


stride = 2

Źródło: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

Zero padding (dopełnienie zerami)

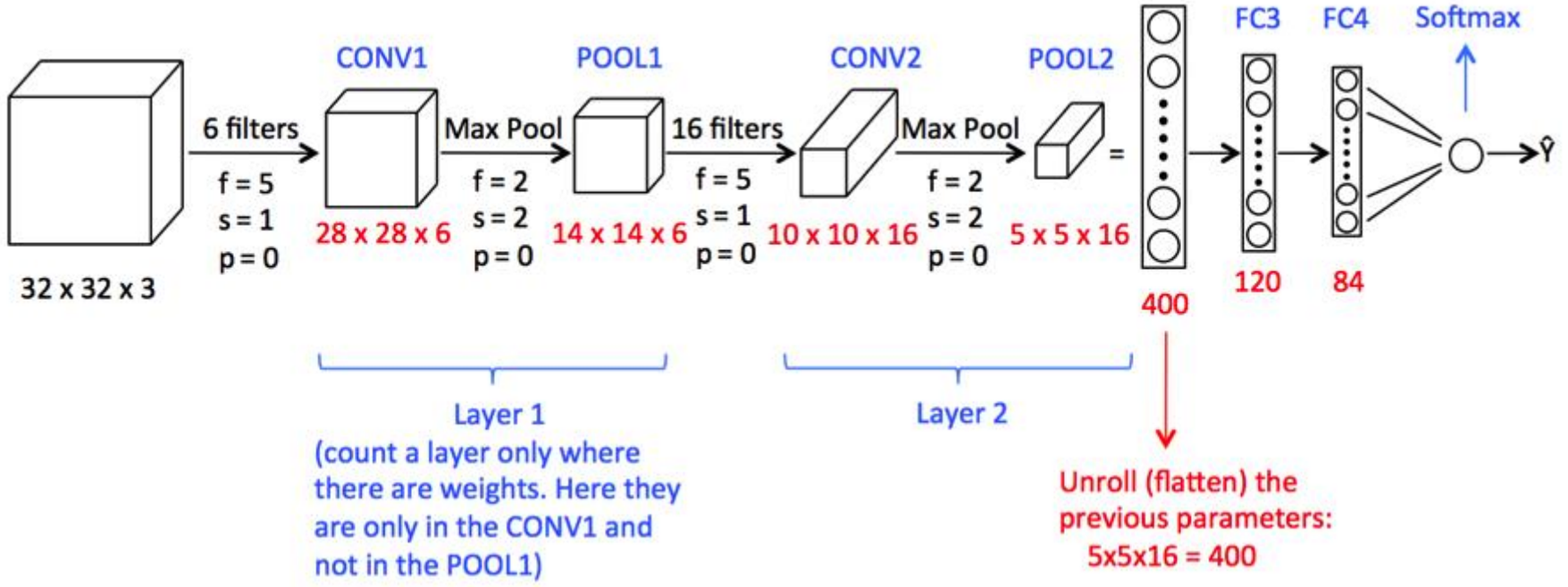
Gdy chcemy, aby mapa cech miała ten sam wymiar, co mapa wejściowa



The input volume is 32 x 32 x 3. If we imagine two borders of zeros around the volume, this gives us a 36 x 36 x 3 volume. Then, when we apply our conv layer with our three 5 x 5 x 3 filters and a stride of 1, then we will also get a 32 x 32 x3 output volume.

Źródło: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

Przykłady CNN 1

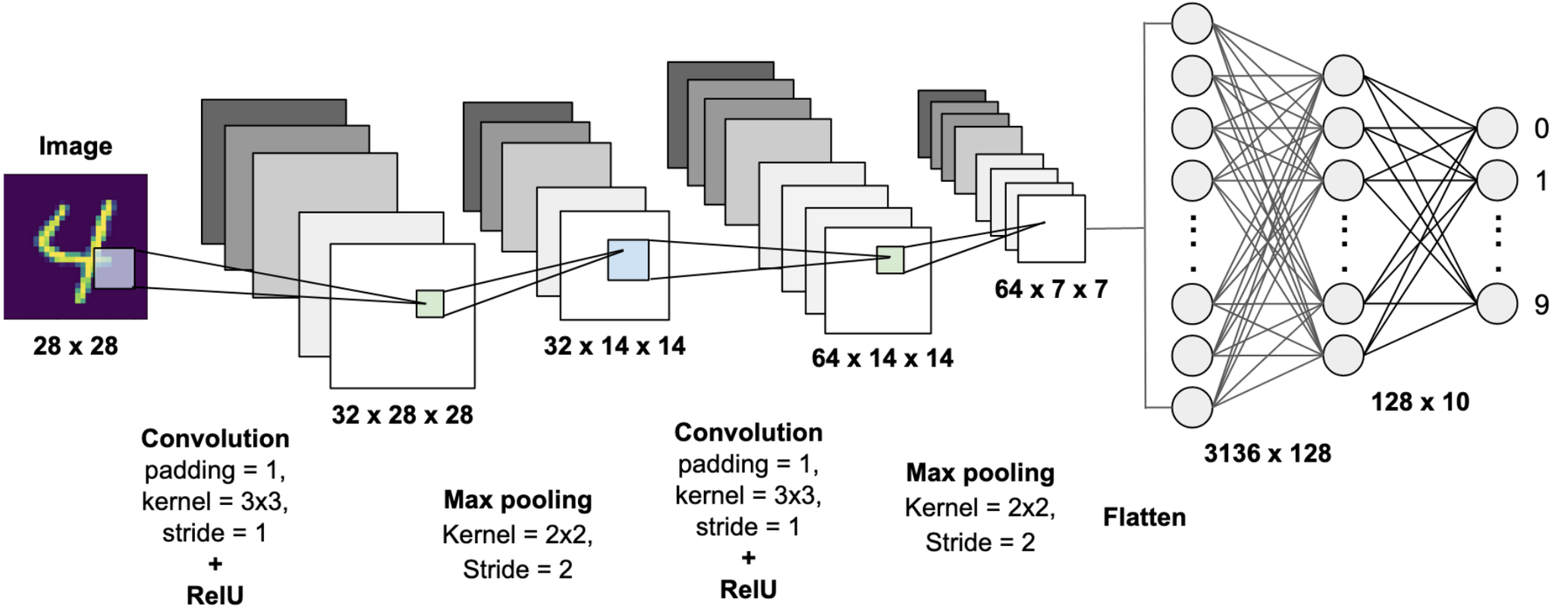


f – rozmiar filtra
s – wielkość kroku (stride)
p – zero padding
n – wielkość wejściowa obrazu

$$Output\ size = \left(\frac{n + 2p - f}{s} + 1\right) x \left(\frac{n + 2p - f}{s} + 1\right)$$

Źródło: https://medium.com/machine-learning-bites/deeplearning-series-convolutional-neural-networks-a9c2f2ee1524

Przykłady CNN 2



Źródło: https://mc.ai/mnist-handwritten-digits-classification-using-a-convolutional-neural-network-cnn/

Learning

Q: Where do all the magic numbers come from?

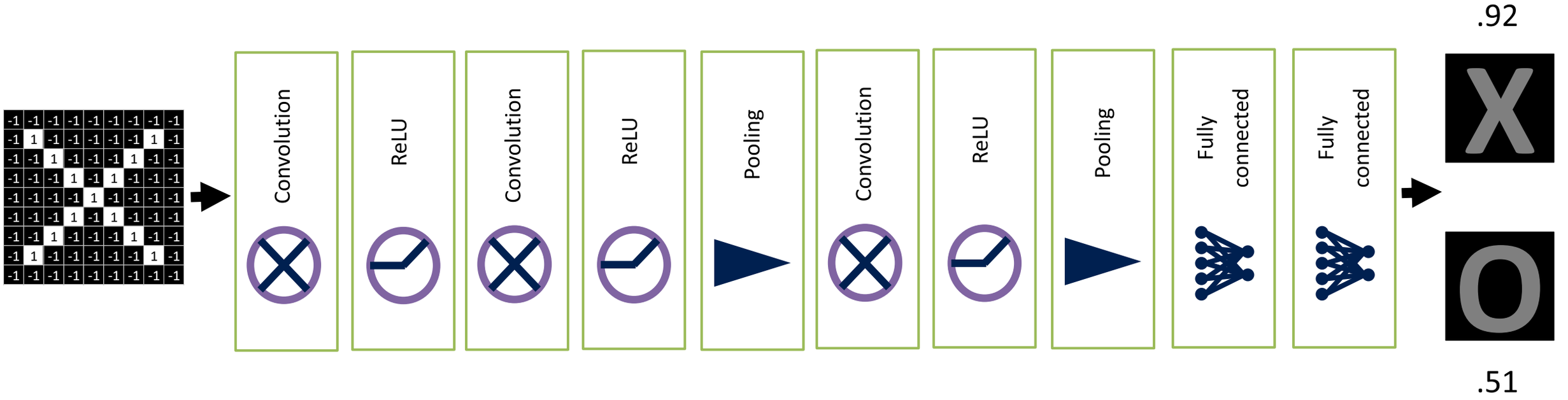
Features in convolutional layers

Voting weights in fully connected layers

A: Backpropagation

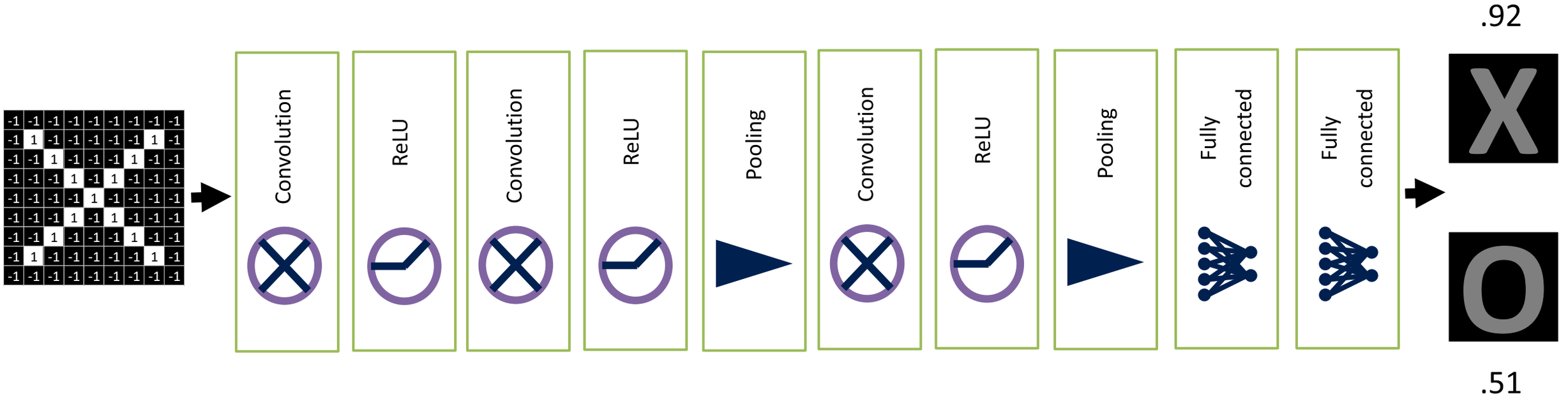
Backpropagation

Error = right answer – actual answer



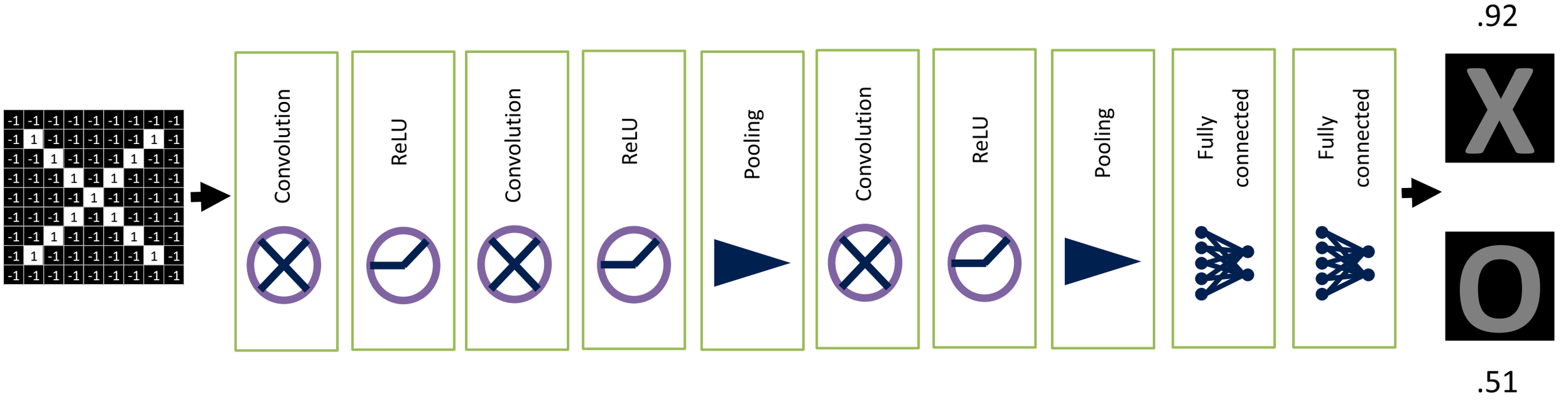
Backpropagation

	Right answer	Actual answer	Error
X	1		
O			



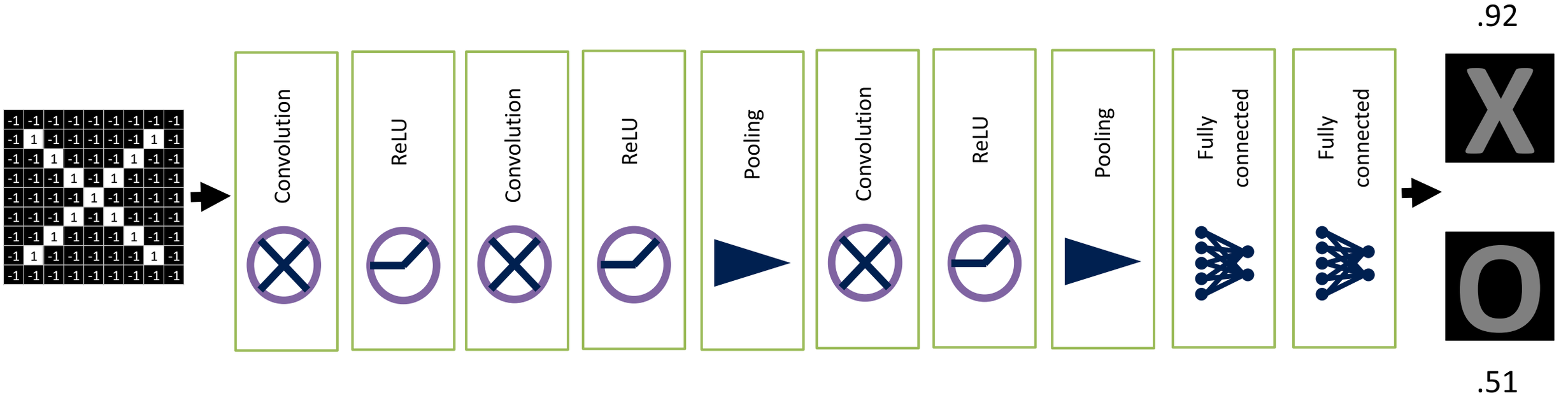
Backpropagation

	Right answer	Actual answer	Error
X	1	0.92	
O			



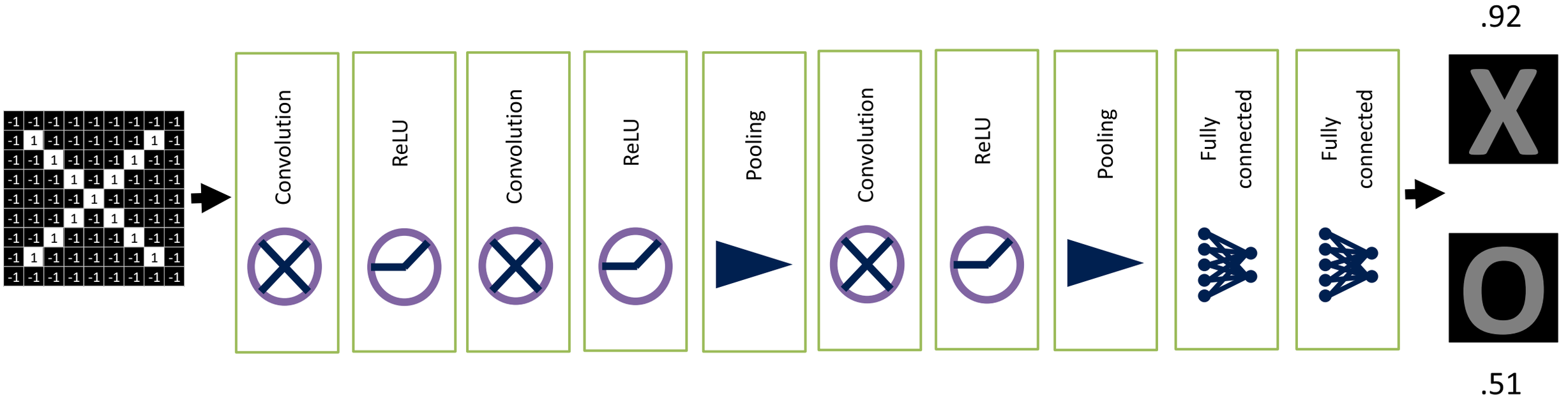
Backpropagation

	Right answer	Actual answer	Error
X	1	0.92	0.08
O			



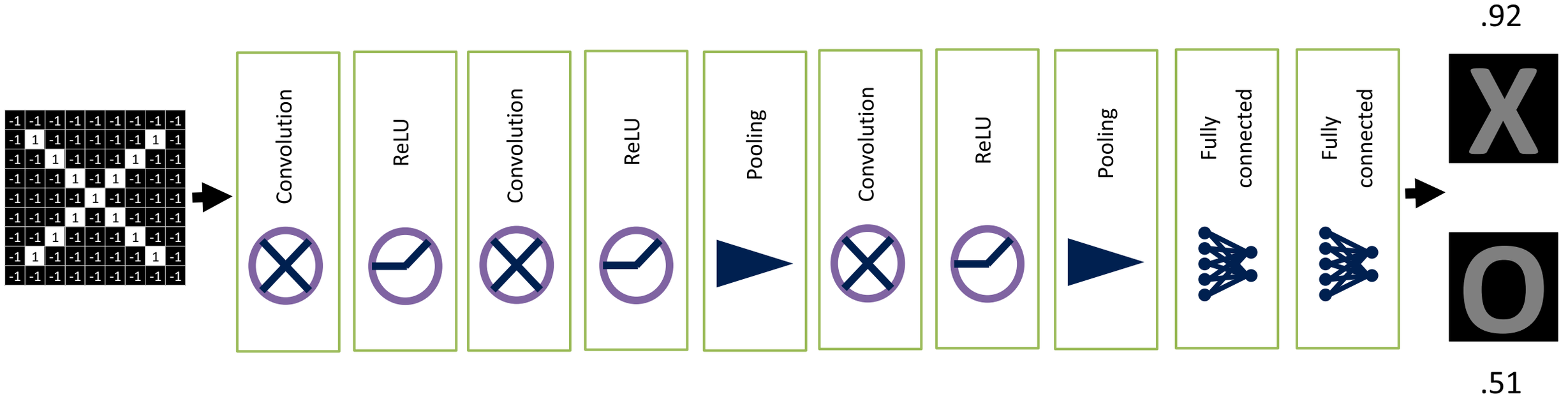
Backpropagation

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



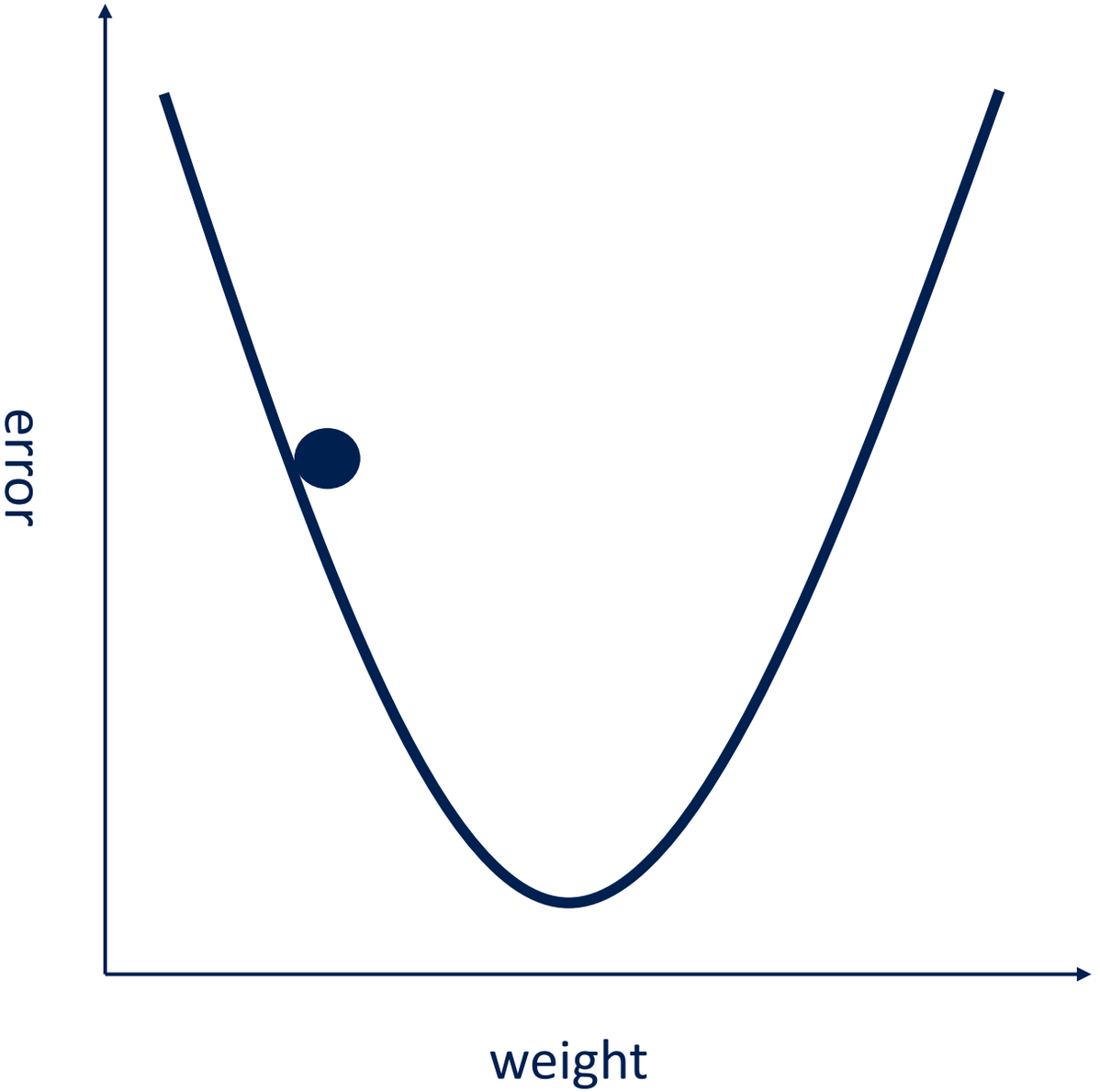
Backpropagation

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



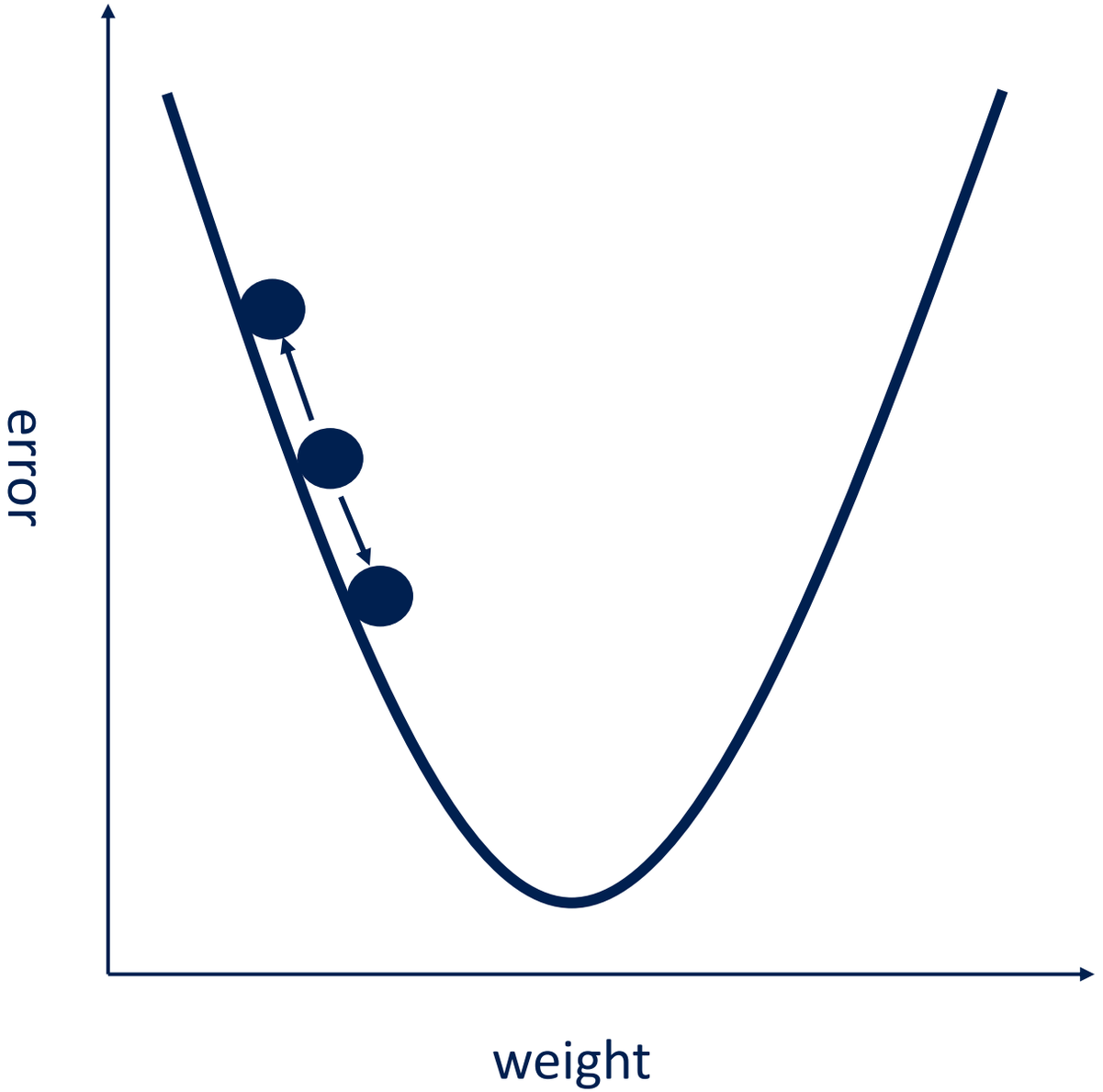
Gradient descent

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



Gradient descent

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



Hyperparameters (knobs)

Convolution

- Number of features
- Size of features

Pooling

- Window size
- Window stride

Fully Connected

- Number of neurons

Architecture

How many of each type of layer?

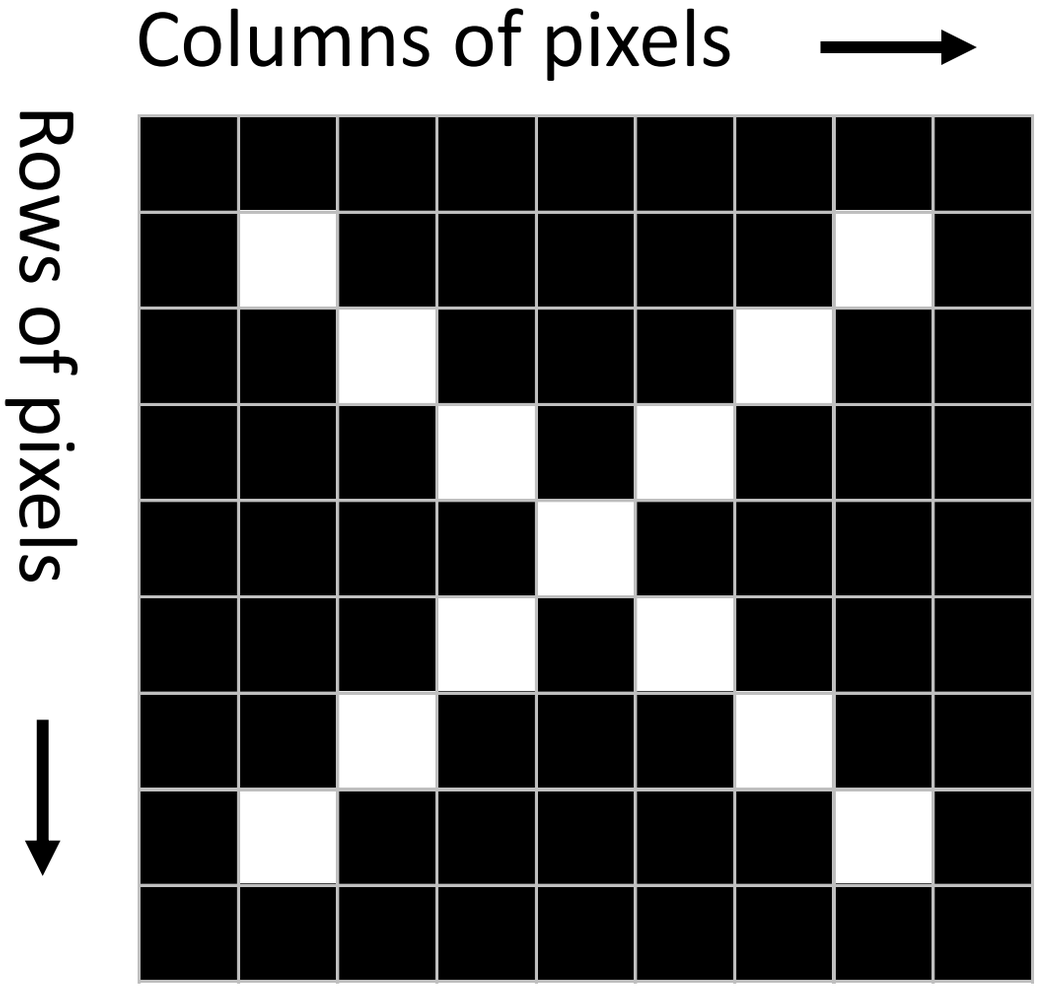
In what order?

Not just images

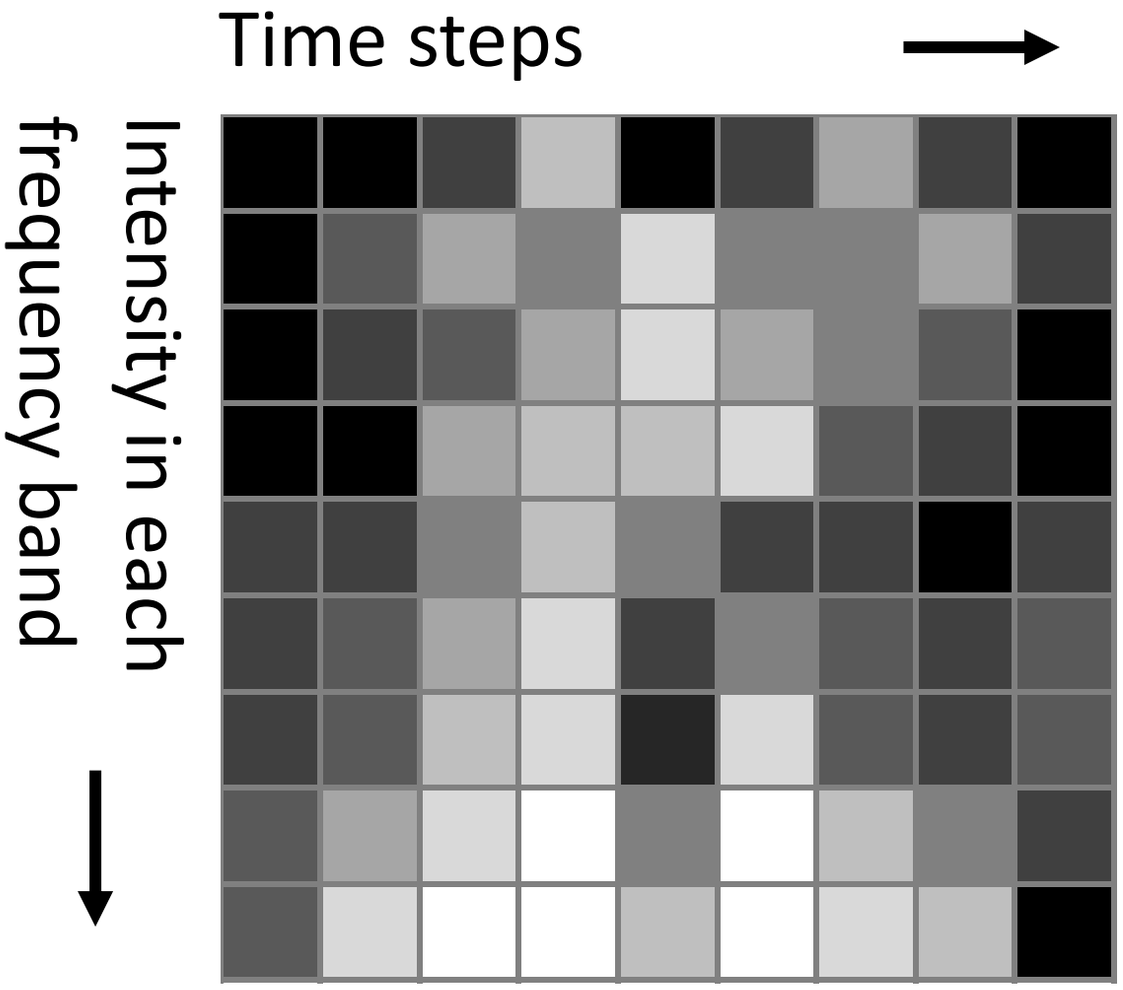
Any 2D (or 3D) data.

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

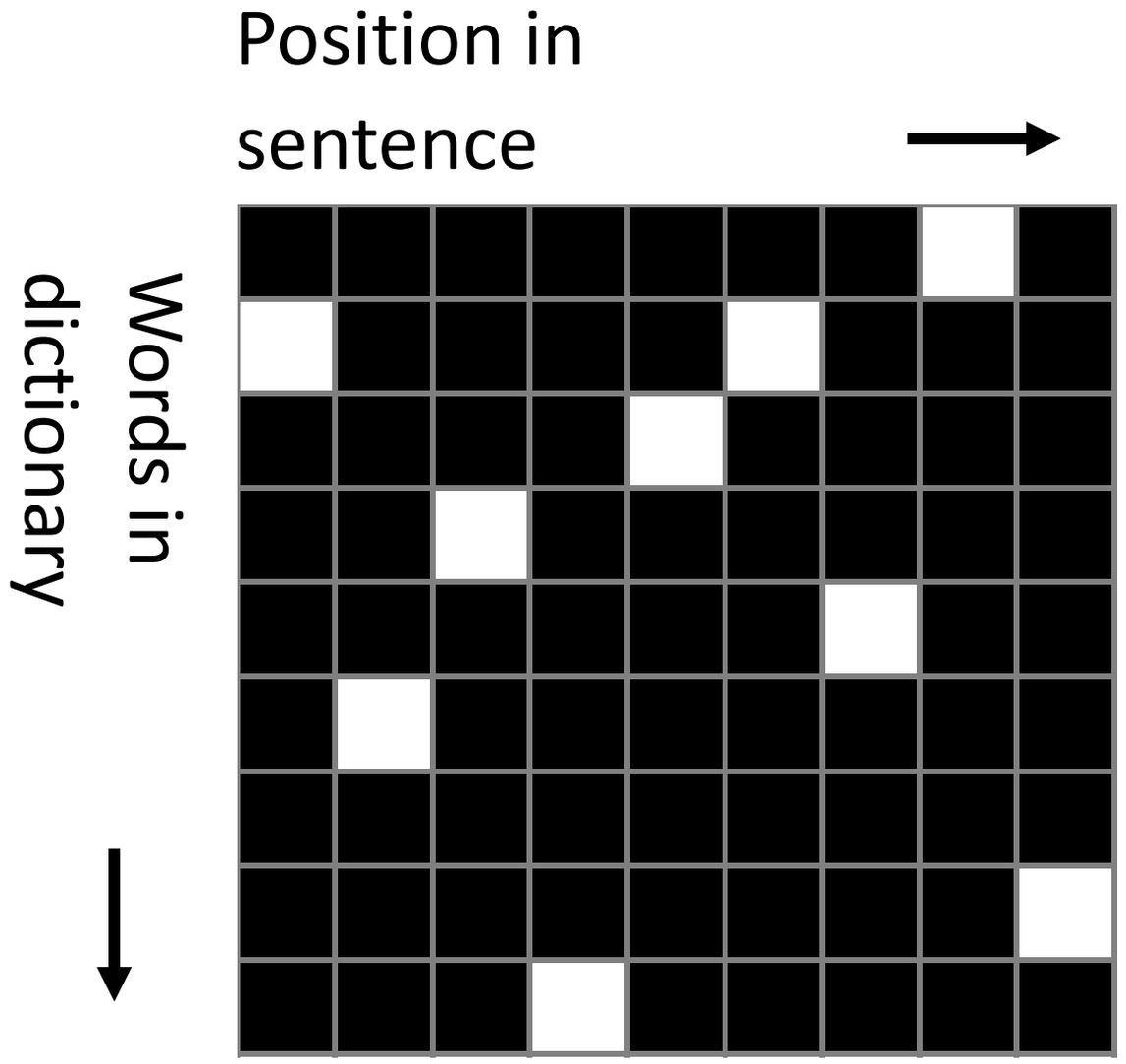
Images



Sound



Text



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.

Customer data

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

Rule of thumb

If your data is just as useful after swapping any of your columns with each other, then you can’t use Convolutional Neural Networks.

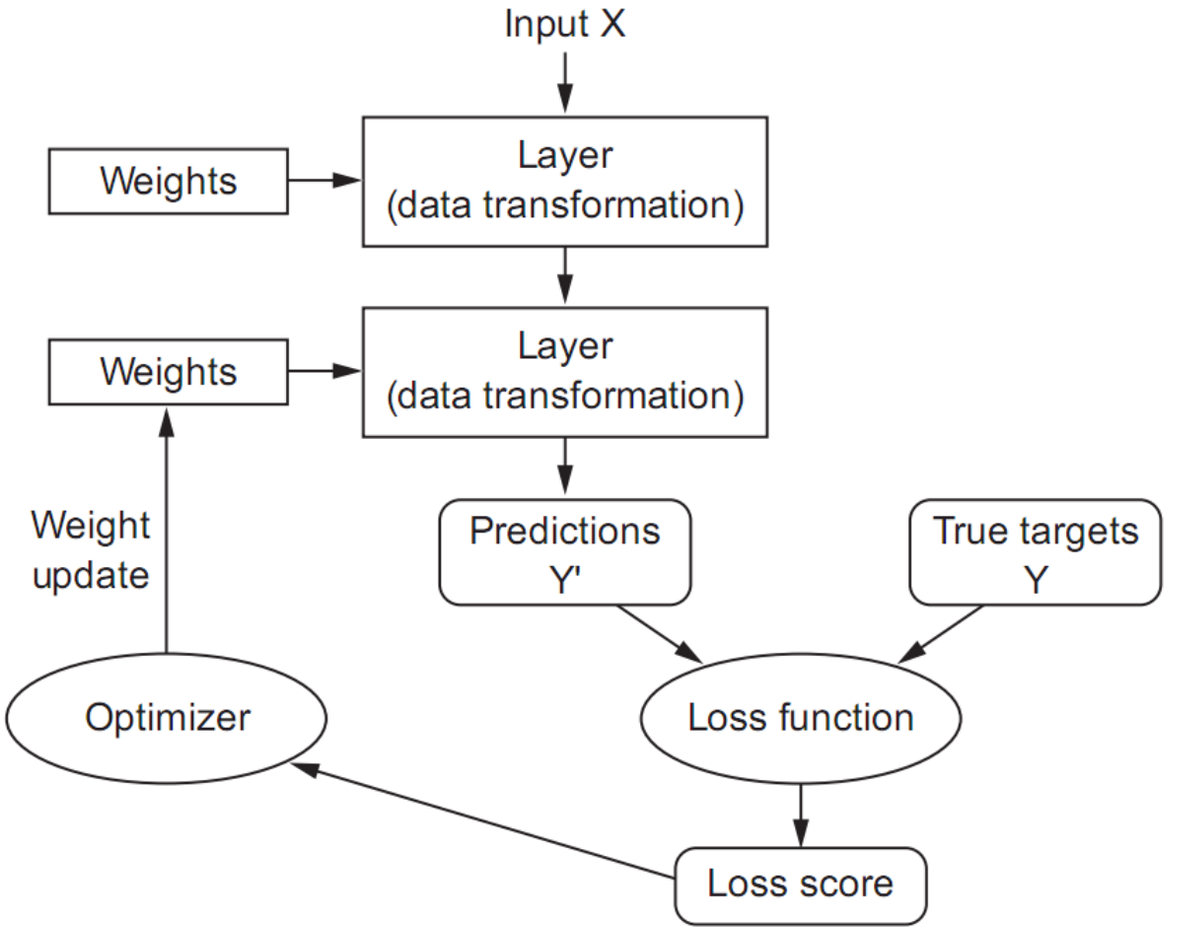
Rule of thumb (wolne tłumaczenie: zasada praktyczna). Powszechnie stosowana nazwa w sytuacjach, gdy mówimy o pewnych zasadach wywodzonych z doświadczenia. Formalne udowodnienie ich poprawności jest albo niemożliwe albo trudne.

In a nutshell

ConvNets are great at finding patterns and using them to classify images.

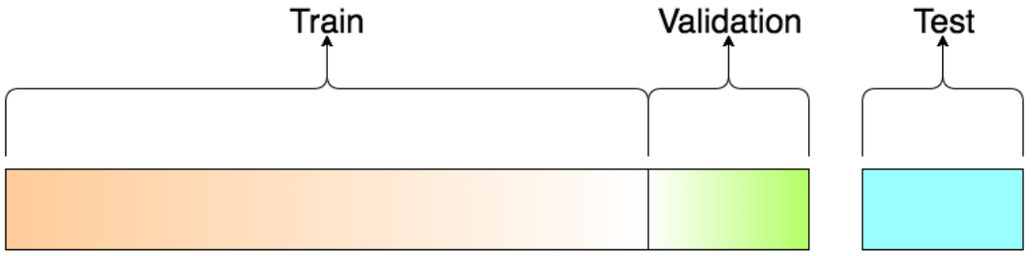
Anatomia uczenia

Warstwy, dane wejściowe, funkcja straty, optymalizator

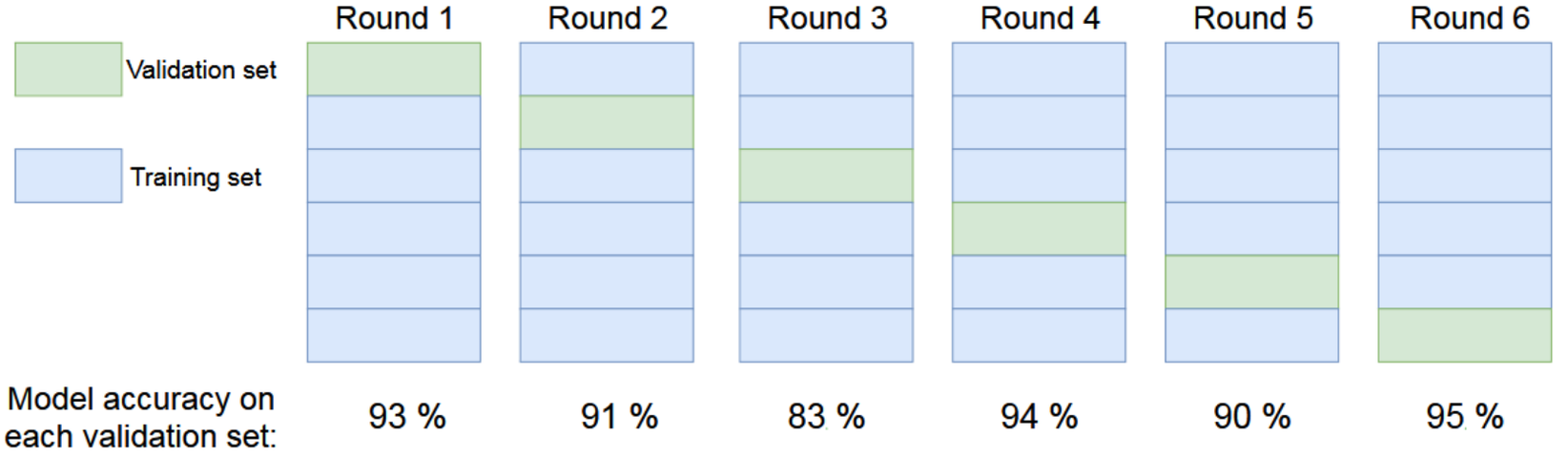


Anatomia uczenia, dane

Zbiory treningowy, walidacyjnym, testowy



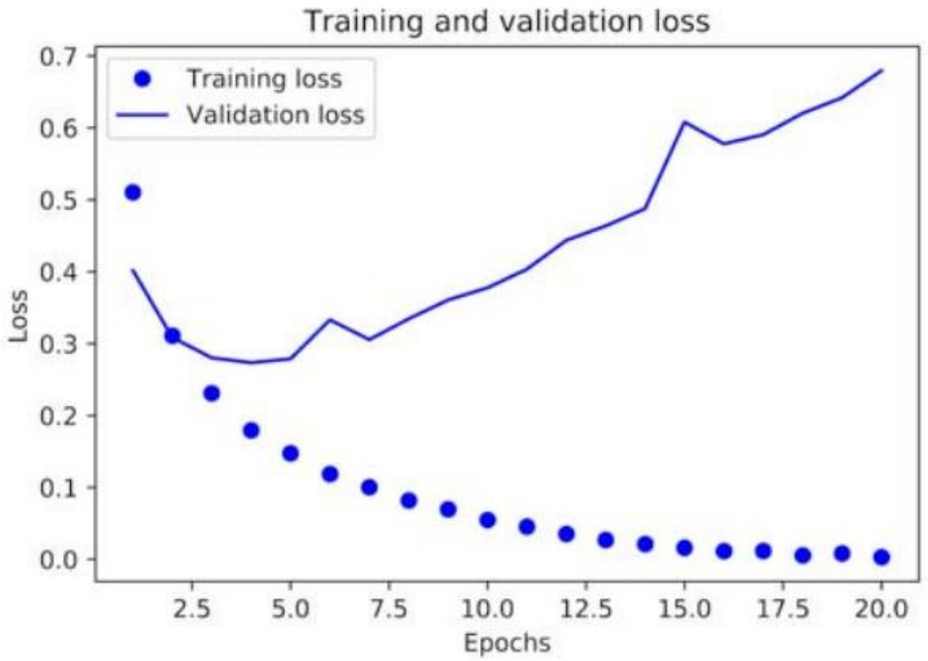
Walidacja krzyżowa

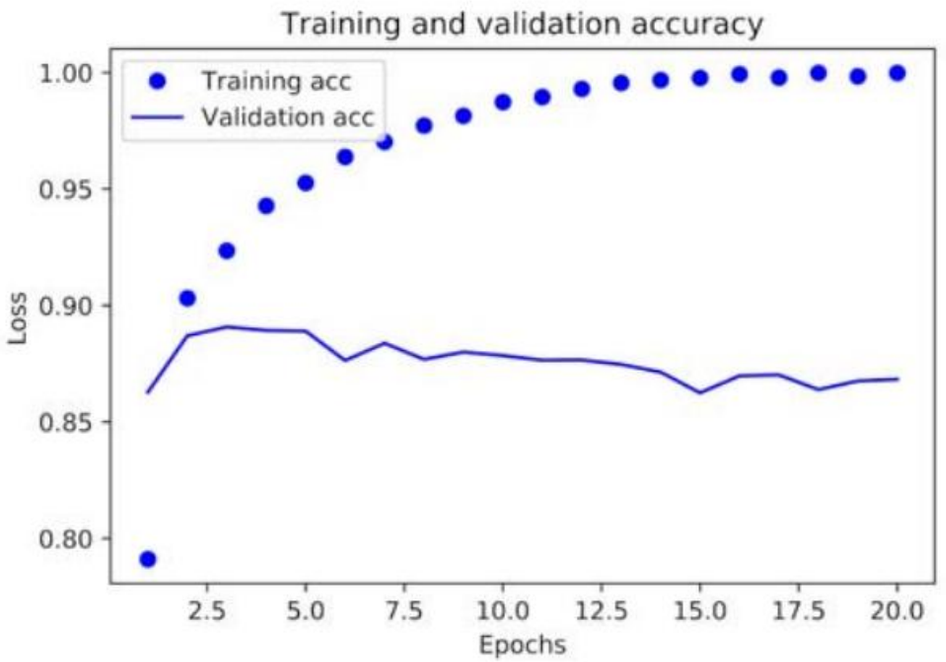


Accuracy = mean(93, 91, 83, 94, 90, 95) = 91 (%)

Anatomia uczenia, ocena wyników

Wartość straty i wartość dokładności

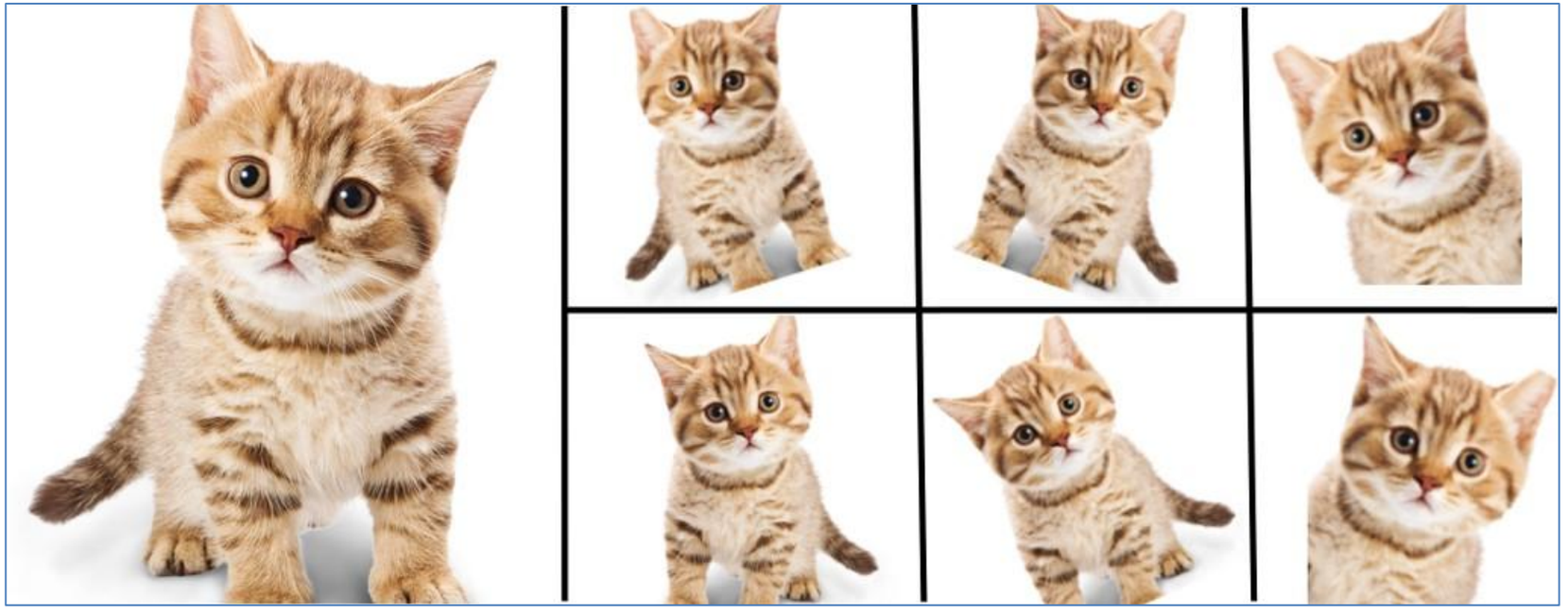




Strata trenowania spada z każdą kolejna epoką. Dokładność trenowania wzrasta.
Problem: od czwartej epoki strata walidacji rośnie a dokładność walidacji spada.
Mamy nadmierne dopasowanie modelu do danych treningowych. Model działa dobrze na zbiorze treningowym ale już gorzej na zbiorze nowych danych (walidacyjnych).

Anatomia uczenia, za mało danych

Augmentacja. Pozwala zmniejszyć skutki zbytniego dopasowania modelu



Zbiór MNIST

Modified **N**ational **I**nstitute of **S**tandards and **T**echnology database

Rodzaj „Hello World” w dziedzinie uczenia maszynowego

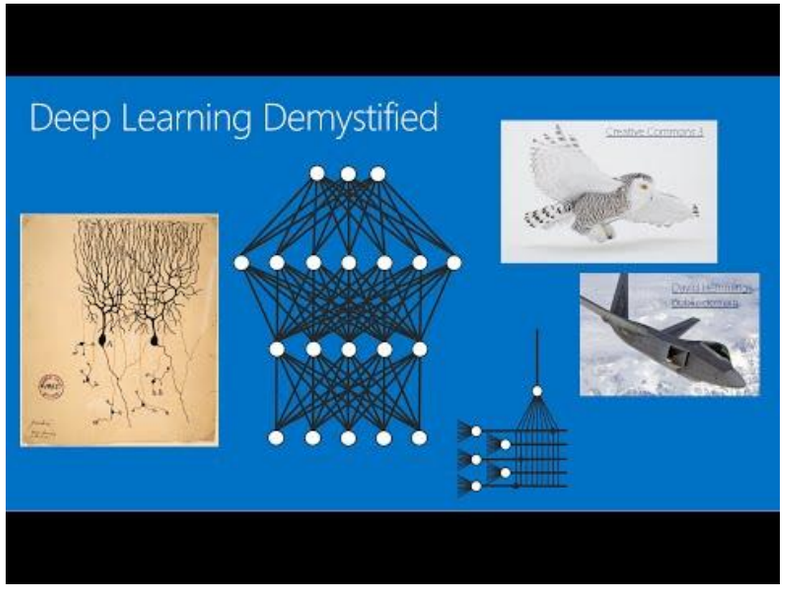
60000 próbek treningowych, 10000 testowych

28 x 28 pikseli

https://en.wikipedia.org/wiki/MNIST_database

Also check out

[Deep Learning Demystified]



[Notes from Stanford CS 231 course]

(Justin Johnson and Andrej Karpathy)

[The writings of Christopher Olah]

Some ConvNet/DNN toolkits

Caffe (Berkeley Vision and Learning Center)

CNTK (Microsoft)

Deeplearning4j (Skymind)

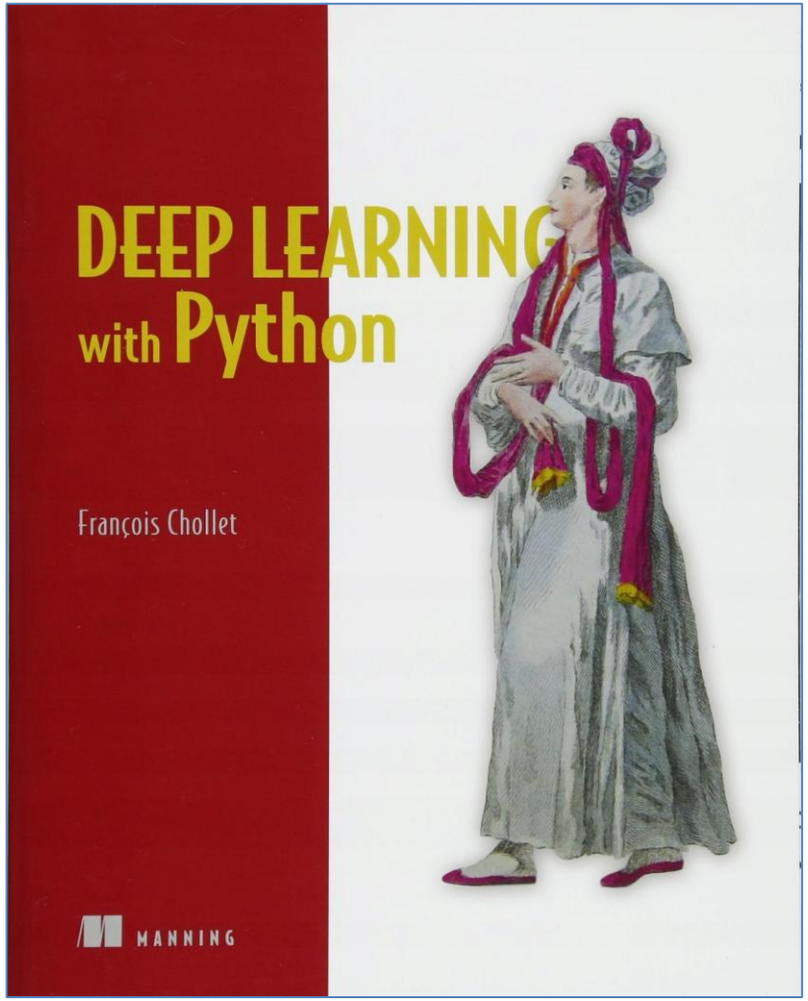
TensorFlow (Google)

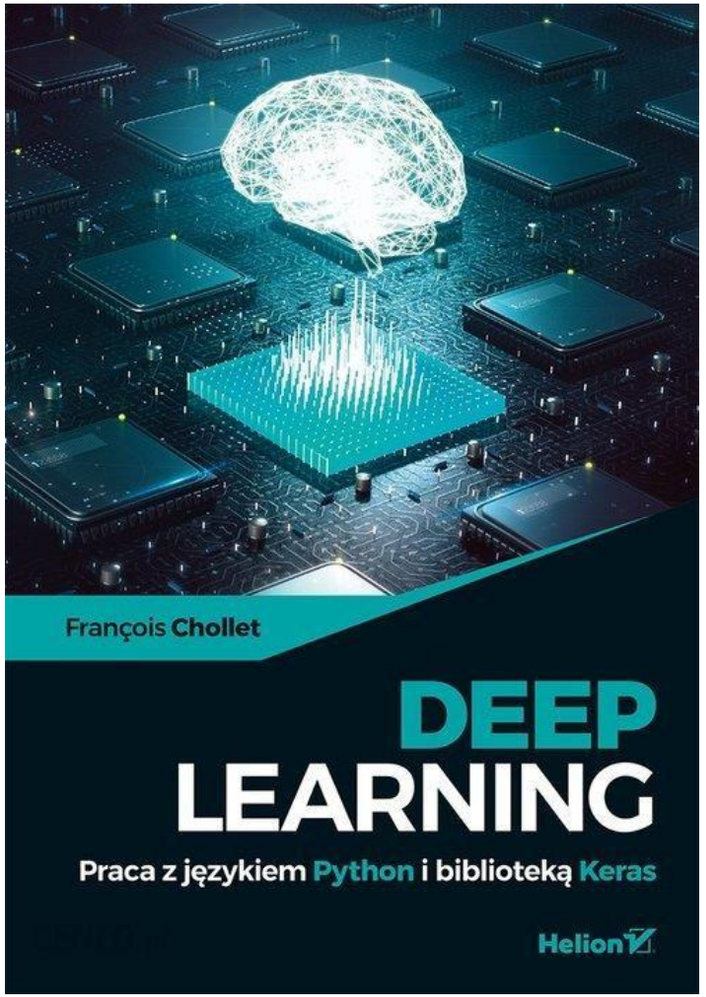
Theano (University of Montreal + broad community)

Torch (Ronan Collobert)

Many others

Polecana książka (ENG, PL)





Przykłady

Na początek sieci neuronowe tradycyjne (nie konwolucyjne lub inaczej głębokie)

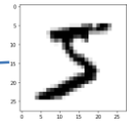
Przykład 1 „Hello world”

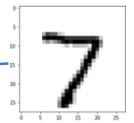
Z książki:

Deep Learning. Praca z językiem Python i biblioteką Keras

```
# Ładujemy klasyczny zbiór danych MNIST, jest dołączony już do Keras-a
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
# Podgląd wybranego rysunku ze zbioru MINST
import matplotlib.pyplot as plt
digit = train_images[0]
plt.imshow(digit, cmap=plt.cm.binary)
plt.show()

digit = test_images[0]
plt.imshow(digit, cmap=plt.cm.binary)
plt.show()
```





Przykład 1 „Hello world”

```
# Początkowo nasze obrazy były zapisane w postaci tensorów o wymiarach (N, 28, 28).
# Załóżmy, że chcemy je przekształcić w macierze o wymiarach (N, 28 * 28).
# Wykonujemy więc operację reshape.
# UWAGA: w zasadzie nic nie stoi na przeszkodzie, aby pracować z obrazami nieprzekształconymi.
# My robimy przekształcenie w celach edukacyjnych.
# Początkowe dane z zakresu 0-255 unit8 przekształcamy do zakresu 0-1 float32

train_images_new = train_images.reshape((60000, 28 * 28))
train_images_new = train_images_new.astype('float32') / 255

test_images_new = test_images.reshape((10000, 28 * 28))
test_images_new = test_images_new.astype('float32') / 255
```

```
# Dodatkowo musimy zakodować etykiety za pomoca kategorii.
# Zamiast wektora z liczbami (dla zbioru treningowego [5 0 4 ... 5 6 8])
# chcemy mieć macierz o rozmiarze 60000 x 10, gdzie w każdym wierszu
# będzie jedynka na tej pozycji, jaką liczbę przedstawia dany obrazek
from keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

Przykład 1 „Hello world”

```
print ("Tak było:\n",train_labels[0:9])
print("\nTak jest to przekodowaniu:\n", train_labels_new[0:9])

# Porównaj train_labels i test_labels z tymi, które były na początku.
print("\nZbiór treningowy:")
print(train_images_new.shape)
print(train_labels_new.shape)

print("\nZbiór testowy:")
print(test_images_new.shape)
print(test_labels_new.shape)
```

```
Tak było:
 [5 0 4 1 9 2 1 3 1]

Tak jest to przekodowaniu:
 [[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]

Zbiór treningowy:
(60000, 784)
(60000, 10)

Zbiór testowy:
(10000, 784)
(10000, 10)
```

Przykład 1 „Hello world”

```
from keras import models
from keras import layers

# Sieć zbudowana z wykorzystaniem modelu SEKWENCYJNEGO.
# Model ten zakłada, że sieć ma dokładnie:
# a) JEDNO wejście,
# b) JEDNO wyjście,
# c) składa się z liniowego stosu warstw.
network = models.Sequential()
network.add(layers.Dense(32, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))

# Odpowiednik, jak wyżej, ale z wykorzystaniem interfejsu API.
# Z jego pomocą można tworzyć sieci o praktycznie dowolnej architekturze.
input_tensor = layers.Input(shape = (784,))
x = layers.Dense(32, activation = 'relu')(input_tensor)
output_tensor = layers.Dense(10, activation = 'softmax')(x)
network = models.Model(inputs = input_tensor, outputs = output_tensor)
```

```
# Kompilowanie sieci
network.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

Przykład 1 „Hello world”

```
# Wizualizacja sieci
print(network.summary())
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 784)]             0
_________________________________________________________________
dense_7 (Dense)              (None, 32)                25120
_________________________________________________________________
dense_8 (Dense)              (None, 10)                330
=================================================================
Total params: 25,450
Trainable params: 25,450
Non-trainable params: 0
```

Przykład 1 „Hello world”

```
# Jesteśmy więc gotowi uruchomić trenowanie (uczenie) sieci
# Podczas trenowania wyświetlane są wartości straty i dokładności
network.fit(
  x = train_images,
  y = train_labels,
  epochs = 10,
  batch_size = 128)
```

```
Epoch 1/10
469/469 [==============================] - 1s 2ms/step - loss: 0.7507 - accuracy: 0.7998
Epoch 2/10
469/469 [==============================] - 1s 3ms/step - loss: 0.2584 - accuracy: 0.9273
Epoch 3/10
469/469 [==============================] - 1s 2ms/step - loss: 0.2071 - accuracy: 0.9412
Epoch 4/10
469/469 [==============================] - 1s 2ms/step - loss: 0.1670 - accuracy: 0.9536
Epoch 5/10
469/469 [==============================] - 1s 3ms/step - loss: 0.1412 - accuracy: 0.9594
Epoch 6/10
469/469 [==============================] - 1s 3ms/step - loss: 0.1332 - accuracy: 0.9628
Epoch 7/10
469/469 [==============================] - 1s 3ms/step - loss: 0.1207 - accuracy: 0.9654
Epoch 8/10
469/469 [==============================] - 1s 3ms/step - loss: 0.1101 - accuracy: 0.9688
Epoch 9/10
469/469 [==============================] - 1s 3ms/step - loss: 0.1033 - accuracy: 0.9702
Epoch 10/10
469/469 [==============================] - 1s 3ms/step - loss: 0.0979 - accuracy: 0.9719
```

Przykład 1 „Hello world”

```
# Na wyuczoną sieć podajemy dane testowe (dane te nie brały udziały w uczeniu)
test_loss, test_acc = network.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
print('test_loss:', test_loss)
```

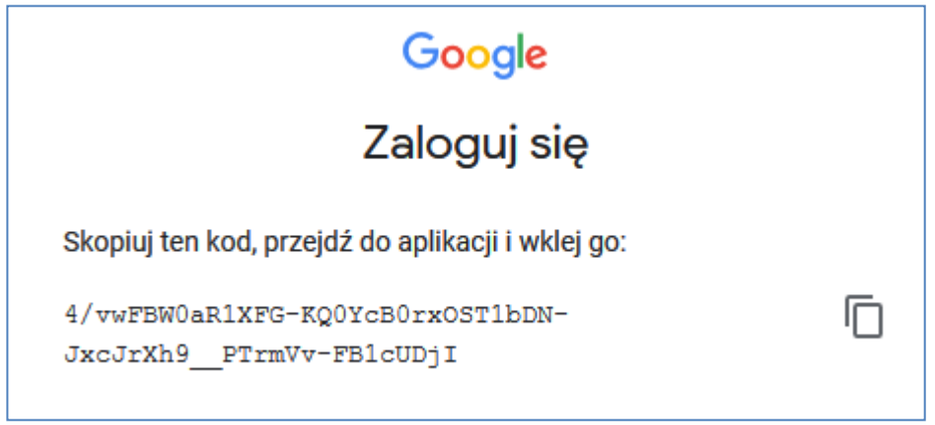
```
313/313 [==============================] - 1s 2ms/step - loss: 0.1232 - accuracy: 0.9639
test_accuracy: 0.9639000296592712
test_loss: 0.12322644144296646
```

Przykład 2, nieco bardziej rozbudowany

Montowanie zasobów dyskowych

```
# Na poczatek musimy sobie podmontowac dysk
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth

Enter your authorization code:
```



Przykład 2 i dalsze bazują na książce
Deep Learning. Praca z językiem Python i biblioteką Keras

UWAGA: książka bazuje na TensorFlow wersja 1.x
Tam gdzie to było konieczne przykłady zostały poprawione, tak aby działały w TensorFlow 2.x

Przykład 2

Pracujemy z Tensorflow w wersji 2

https://colab.research.google.com/notebooks/tensorflow_version.ipynb

```
# Na temat 'magics' czytaj tutaj:
# https://ipython.readthedocs.io/en/stable/interactive/magics.html

# https://colab.research.google.com/notebooks/tensorflow_version.ipynb
%tensorflow_version 2.x

import tensorflow as tf
import datetime, os
import matplotlib.pyplot as plt

from keras.datasets import mnist
from keras.utils import to_categorical
from keras import layers

print(tf.__version__)
```

```
TensorFlow 2.x selected.
2.1.0-rc1
Using TensorFlow backend.
```

Note

To Jupyter users: Magics are specific to and provided by the IPython kernel. Whether Magics are available on a kernel is a decision that is made by the kernel developer on a per-kernel basis. To work properly, Magics must use a syntax element which is not valid in the underlying language. For example, the IPython kernel uses the `%` syntax element for Magics as `%` is not a valid unary operator in Python. However, `%` might have meaning in other languages.

Przykład 2

Załadowanie danych treningowych, walidacyjnych oraz testowych

```
### https://www.tensorflow.org/tutorials/

# Ładujemy klasyczny zbiór danych MNIST
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Podgląd wybranego rysunku
digit = train_images[4,:,:]
plt.imshow(digit, cmap=plt.cm.binary)
# Można też sobie zobaczyć  obrazek jak jest zapisany fizycznie w tablicy
print(train_images[4,:,:])
print(train_labels[4])

# W celu monitorowania dokładności modelu W CZASIE trenowania utworzymy zbiór
# danych WALIDACYJNYCH. Zrobimy to odłączając 10000 próbek od treningowego zbioru
train_images = train_images[10000:]
train_labels = train_labels[10000:]

val_images = train_images[:10000]
val_labels = train_labels[:10000]
```

Przykład 2

Załadowanie danych treningowych, walidacyjnych oraz testowych

```
train_images = train_images.astype('float32') / 255
val_images = val_images.astype('float32') / 255
test_images = test_images.astype('float32') / 255

# Etykiety zapisane są w tablicy jako liczby całkowite.
# Do każdego obrazu przypisana jest jedna etykieta.
print(train_labels[0:9])

# Wyświetlamy rozmiary poszczególnych tablic
print("train_images.shape:", train_images.shape)
print("train_labels.shape:", train_labels.shape)
print("val_images.shape:", val_images.shape)
print("val_labels.shape:", val_labels.shape)
print("test_images.shape:", test_images.shape)
print("test_labels.shape:", test_labels.shape)
```

```
train_images.shape: (50000, 28, 28)
train_labels.shape: (50000,)
val_images.shape: (10000, 28, 28)
val_labels.shape: (10000,)
test_images.shape: (10000, 28, 28)
test_labels.shape: (10000,)
```

Przykład 2

Konwersja danych z (N, 28, 28) na (N, 28 * 28)

```
# Początkowo nasze obrazy były zapisane w postaci tensorów o wymiarach (N, 28, 28).
# Załóżmy, że chcemy je przekształcić w macierze o wymiarach (N, 28 * 28).
# Wykonujemy więc operację reshape.
# UWAGA: w zasadzie nic nie stoi na przeszkodzie, aby pracować z obrazami niepszekształconymi.
# My robimy przekształcenie w celach edykacyjnych.
train_images = train_images.reshape((50000, 28 * 28))
val_images = val_images.reshape((10000, 28 * 28))
test_images = test_images.reshape((10000, 28 * 28))

# Wyświetlany rozmiary obrazów po reshape
print("train_images.shape:", train_images.shape)
print("val_images.shape:", val_images.shape)
print("test_images.shape:", test_images.shape)
```

```
train_images.shape: (50000, 784)
val_images.shape: (10000, 784)
test_images.shape: (10000, 784)
```

Przykład 2

Struktura sieci

```
# Tworzymy najprostrzą sieć typu fully-connected.
network = tf.keras.models.Sequential([
  # 28 * 28, bo wcześniej zrobiliśmy 'reshape'
  tf.keras.layers.Flatten(input_shape=(28 * 28,)),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  # Ostatnia warstwa zwraca tablicę 10 wartości prawdopodobieństa
  # (suma wszystkich tych wartości jest równa 1)
  tf.keras.layers.Dense(10, activation='softmax')
])

# Prosta semigraficzna wizualizacja sieć
network.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 512)               401920
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
```

Dropout - consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting.

Przykład 2

Kompilacja sieci i jej trenowanie

```
# Sieć musi być najpierw skomilowana.
network.compile(optimizer='rmsprop',
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy'])

# Uczenie sieci.
# Jednocześnie monitrujemy funkcje straty i dokładności modelu przy przetwarzaniu 10000
# próbek, które przed chwila odłożyliśmy na bok (dane walidacyjne).
history = network.fit(train_images,
                      train_labels,
                      epochs=10,
                      batch_size=512,
                      validation_data=(val_images, val_labels),
                      verbose=1) # gdy 0 trenowanie w trybie "cichym"
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 2s 49us/sample - loss: 0.4432 - accuracy: 0.8728 - val_loss: 0.2244 - val_accuracy: 0.9365
Epoch 2/10
50000/50000 [==============================] - 0s 8us/sample - loss: 0.2033 - accuracy: 0.9411 - val_loss: 0.1494 - val_accuracy: 0.9570
Epoch 3/10
50000/50000 [==============================] - 0s 7us/sample - loss: 0.1432 - accuracy: 0.9594 - val_loss: 0.1022 - val_accuracy: 0.9720
Epoch 4/10
```

Przykład 2

Zapis wyników trenowania (na potrzeby wykresów)

```
# Wyniki uczenia zapisujemy sobie do zmiennej
history_dict = history.history
print(history_dict.keys())

# Oglądamy zapisany stan uczenia sieci
print('train_acc:', history_dict['accuracy'])
print('train_loss:', history_dict['loss'])
print('train_val_acc:', history_dict['val_accuracy'])
print('train_val_loss:', history_dict['val_loss'])

# Do zmiennych. Dalej będziemy to uzywać przy robieniu wykresów
accuracy = history_dict['accuracy']
val_accuracy  = history_dict['val_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']
```

```
train_acc: [0.87276, 0.94112, 0.95936, 0.96802, 0.97334, 0.9788, 0.98156, 0.98516, 0.98676,
train_loss: [0.4431690324401856, 0.2033136339378357, 0.14320741282463073, 0.110313406960964
train_val_acc: [0.9365, 0.957, 0.972, 0.9776, 0.9839, 0.9864, 0.9893, 0.9929, 0.9932, 0.995
train_val_loss: [0.22435370657444, 0.14935041229724885, 0.10219935108423234, 0.079190180170
```

Przykład 2

Na wyuczoną siec podajemy dane testowe (które nie brały udziału w uczeniu sieci)

– obserwujemy jak sieć radzi sobie z nowymi danymi

```
# Na wyuczoną sieć podajemy dane TESTOWE
test_loss, test_acc = network.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
print('test_loss:', test_loss)
```

```
10000/10000 [==============================] - 1s 55us/sample - loss: 0.0676 - accuracy: 0.9785
test_acc: 0.9785
test_loss: 0.06760850644751917
```

Przykład 2

Zapis wyników uczenia do pliku, format HDF5

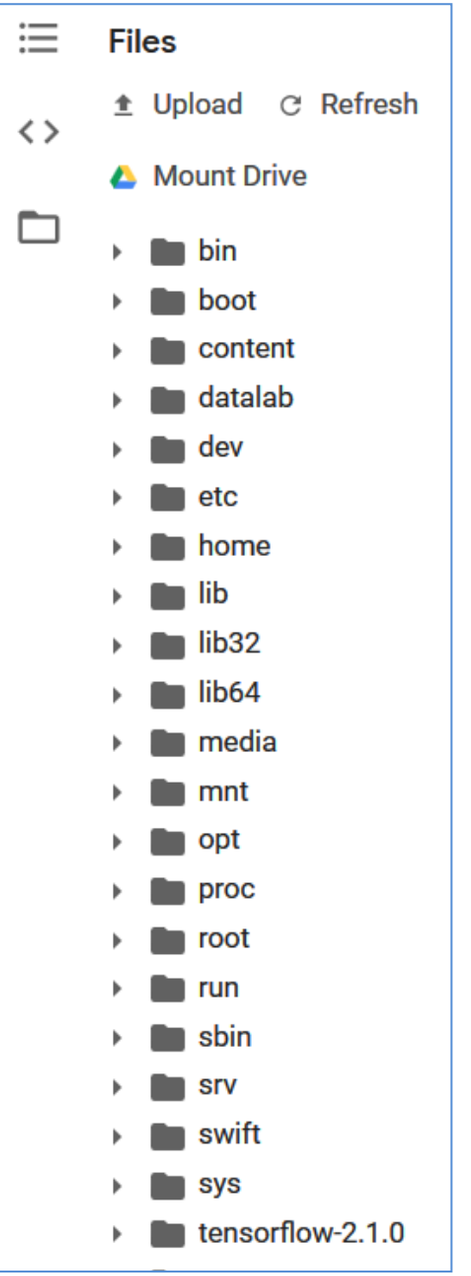
```
# Zapisanie aktualnego stanu sieci. Pliku szukaj w /content/drive'
# Zapis w formacie HDF5, https://en.wikipedia.org/wiki/Hierarchical_Data_Format
#
# Zapisane dane o sieci (struktura, wagi itp) możemy odczytać z pliku w formacie HDF5,
# nie trzeba więc sieci ponownie trenować np. po wyłączeniu kompa.
#
# This single HDF5 file will contain:
#     - the architecture of the model (allowing the recreation of the model)
#     - the weights of the model
#     - the training configuration (e.g. loss, optimizer)
#     - the state of the optimizer (allows you to resume the training from exactly where you left off)
network.save('my_model.h5')
```

Przykład 2

Gdzie jest plik? Polecenia systemu operacyjnego w COLAB

```
# Komendy shell-a wydajemy z wykrzyknikiem na poczatku
!pwd
!ls -la
# jesteśmy root-em :-)
!id
!uname -a
```

```
/content
total 3228
drwxr-xr-x 1 root root    4096 Jan 27 17:16 .
drwxr-xr-x 1 root root    4096 Jan 27 16:24 ..
drwxr-xr-x 1 root root    4096 Jan 13 16:38 .config
drwx------ 4 root root    4096 Jan 27 16:32 drive
-rw-r--r-- 1 root root 3282528 Jan 27 17:18 my_model.h5
drwxr-xr-x 1 root root    4096 Jan 13 16:38 sample_data
uid=0(root) gid=0(root) groups=0(root)
Linux 689e92733ff3 4.14.137+ #1 SMP Thu Aug 8 02:47:02 PDT 2019 x86_64 x86_64 x86_64 GNU/
```



Przykład 2

Odtworzenie zapisanego modelu

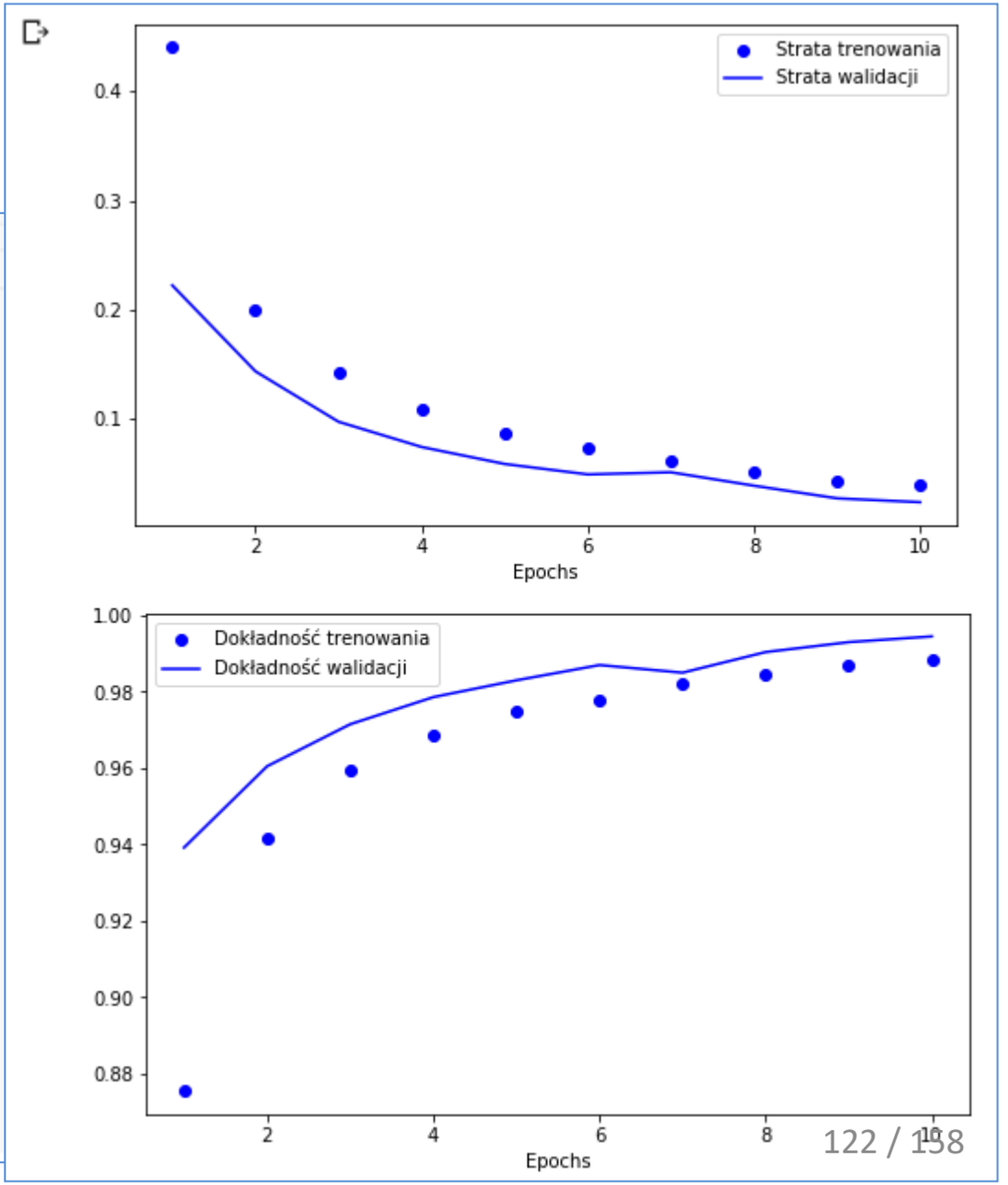
```
# Na próbę załadowanie zapisanego wcześniej modelu
network_restored = tf.keras.models.load_model('my_model.h5')
network_restored.summary()

# Na zapisaną sieć podajemy dane testowe. Otrzymujemy identyczne wyniki, jak wyżej.
# tym samym potwierdzamy, że dane wyuczonej sieci poprawnie się zapisały do pliku
# i potem poprawnie odczytały.
test_loss, test_acc = network_restored.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
print('test_loss:', test_loss)

# Komendy shell-a wydajemy z wykrzyknikiem na poczatku
!pwd
!ls -la
# jesteśmy root-em :-)
!id
!uname -a
```

```
10000/10000 [==============================] - 1s 63us/sample - loss: 0.0676 - accuracy: 0.9785
test_acc: 0.9785
test_loss: 0.06760850644751917
```

Przykład 2

Przykładowe wykresy

```
#################################################################
# Strata trenowania i walidacji
#################################################################
epochs = range(1, len(accuracy) + 1)
plt.rcParams['figure.figsize'] = [8, 5]
plt.plot(epochs, loss, 'bo', label='Strata trenowania')
plt.plot(epochs, val_loss, 'b', label='Strata walidacji')
plt.xlabel('Epochs')
plt.legend()
plt.show()

#################################################################
# Dokładność trenowania i walidacji
#################################################################
epochs = range(1, len(accuracy) + 1)
plt.rcParams['figure.figsize'] = [8, 5]
plt.plot(epochs, accuracy, 'bo', label='Dokładność trenowania')
plt.plot(epochs, val_accuracy, 'b', label='Dokładność walidacji')
plt.xlabel('Epochs')
plt.legend()
plt.show()
```



Przykład 3, callbacks (wywołania zwrotne)

- Collback to specjalny mechanizm, który pozwala sterować modelem podczas jego trenowania i ewaluacji
- Może czytać stan modelu i w razie potrzeby „w locie” zmieniać jego parametry
- Sposoby użycia:
  - tworzenie punktów kontrolnych – zapisywanie wag modelu w różnych momentach procesu trenowania
  - wczesne zatrzymanie – gdy strata walidacji przestaje ulegać poprawie
  - dynamiczne dostrajanie niektórych parametrów modelu w czasie jego trenowania
  - zapisywanie metryk trenowania i walidacji i ich wizualizacja online (TensorBoard)

Przykład 3, callbacks (wywołania zwrotne)

- Keras zawiera wiele standardowych wywołań zwrotnych (można też tworzyć własne)

`class BaseLogger` : Callback that accumulates epoch averages of metrics.

`class CSVLogger` : Callback that streams epoch results to a csv file.

`class Callback` : Abstract base class used to build new callbacks.

`class EarlyStopping` : Stop training when a monitored quantity has stopped improving.

`class History` : Callback that records events into a `History` object.

`class LambdaCallback` : Callback for creating simple, custom callbacks on-the-fly.

`class LearningRateScheduler` : Learning rate scheduler.

`class ModelCheckpoint` : Save the model after every epoch.

`class ProgbarLogger` : Callback that prints metrics to stdout.

`class ReduceLROnPlateau` : Reduce learning rate when a metric has stopped improving.

`class RemoteMonitor` : Callback used to stream events to a server.

`class TensorBoard` : Enable visualizations for TensorBoard.

`class TerminateOnNaN` : Callback that terminates training when a NaN loss is encountered.

Przykład 3, TensorBoard

- Umożliwia monitorowanie wszystkich procesów zachodzących w modelu podczas jego trenowania
- Interfejs WWW
    - obserwowanie grafik reprezentujących parametry trenowanego modelu
    - graficzne przedstawienie architektury modelu
    - generowanie histogramów aktywacji i gradientów
    - rójwymiarowe i dwuwymiarowe oglądanie osadzeń

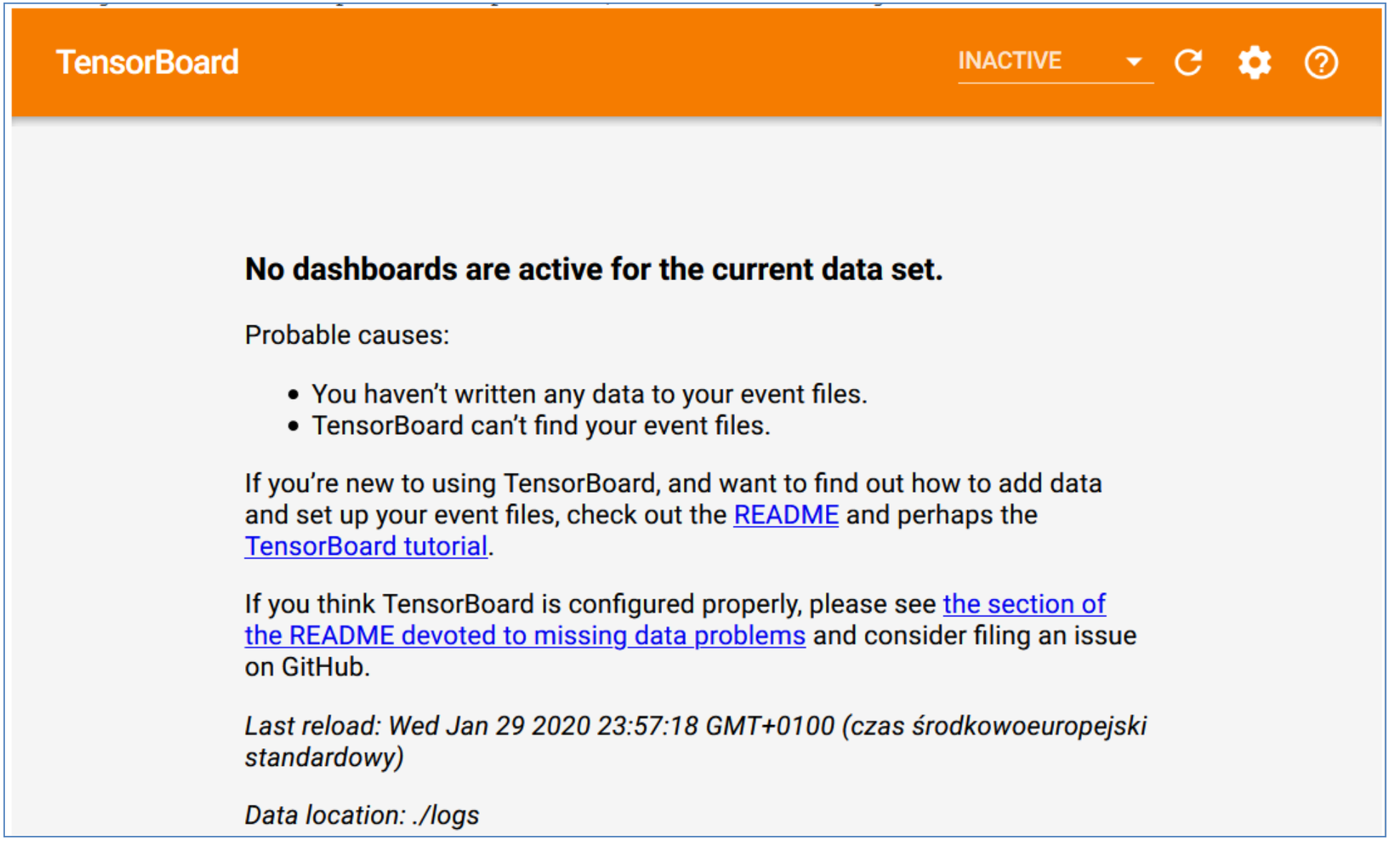
Przykład 3

```
%tensorflow_version 2.x

import tensorflow as tf
from keras import layers
from keras import callbacks

print(tf.__version__)

#Start by installing TF 2.0 and loading the TensorBoard notebook extension:
#!pip install -q tf-nightly-2.0-preview
# Load the TensorBoard notebook extension
%load_ext tensorboard
```

Przykład 3

```
# Wygląda na to, że za każdym razem jak chcę uruchomić TB, to trzeba "wyczyścić" katalogi.
# W wersji nie COLAB-owej jakoś nie pamiętam, żeby takie coś trzeba było robić.
# Też trzeba z-kill-ować TB i uruchomić od nowa.
# Gdy tego nie zrobimy, przebiegi acc oraz loss albo nie pojawiają się wcale albo
# pojawiają się jakieś bezsensowne.
# Wygląda też, że trzeba uruchomić nie tylko model.fit ale też
# model.compile i polecenie tworzące model

# Fizyczne kasowanie katalogu z systemu operacyjnego
!rm -rf logs

logs_base_dir = "./logs"
os.makedirs(logs_base_dir, exist_ok=True)
logdir = os.path.join(logs_base_dir, datetime.datetime.now().strftime("%Y-%m-%d-%H:%M:%S"))

# The %tensorboard magic has exactly the same format as the TensorBoard
#v command line invocation, but with a %-sign in front of it.
!kill 7713
%tensorboard --logdir {logs_base_dir}
```

Przykład 3


TensorBoard
INACTIVE
No dashboards are active for the current data set.
Probable causes:
You haven't written any data to your event files.
TensorBoard can't find your event files.
If you're new to using TensorBoard, and want to find out how to add data and set up your event files, check out the README and perhaps the TensorBoard tutorial.
If you think TensorBoard is configured properly, please see the section of the README devoted to missing data problems and consider filing an issue on GitHub.
Last reload: Wed Jan 29 2020 23:57:18 GMT+0100 (czas środkowoeuropejski standardowy)
Data location: ./logs

Przykład 3

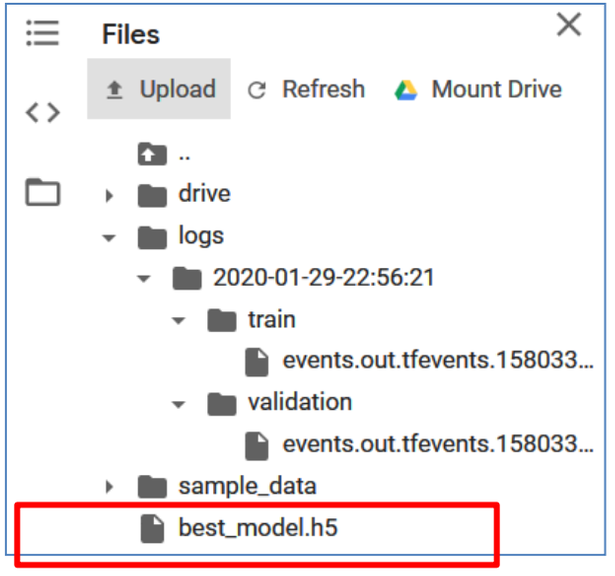
```
callbacks_list =[
                callbacks.ModelCheckpoint(
                    # ścieżka pliku modelu
                    filepath='best_model.h5',
                    # dzieki tym dwóm argumentom plik modelu nie
                    # zostanie nadpisany, jeżeli wartość val_loss
                    # nie ulegnie poprawie. Rozwiązanie to pozwala
                    # zachować wersję modelu, która została
                    # dotychczas najlepiej wytrenowana
                    monitor="val_loss",
                    save_best_only=True
                ),
                tf.keras.callbacks.TensorBoard(
                    log_dir=logdir,
                    profile_batch=0
                )
]
```

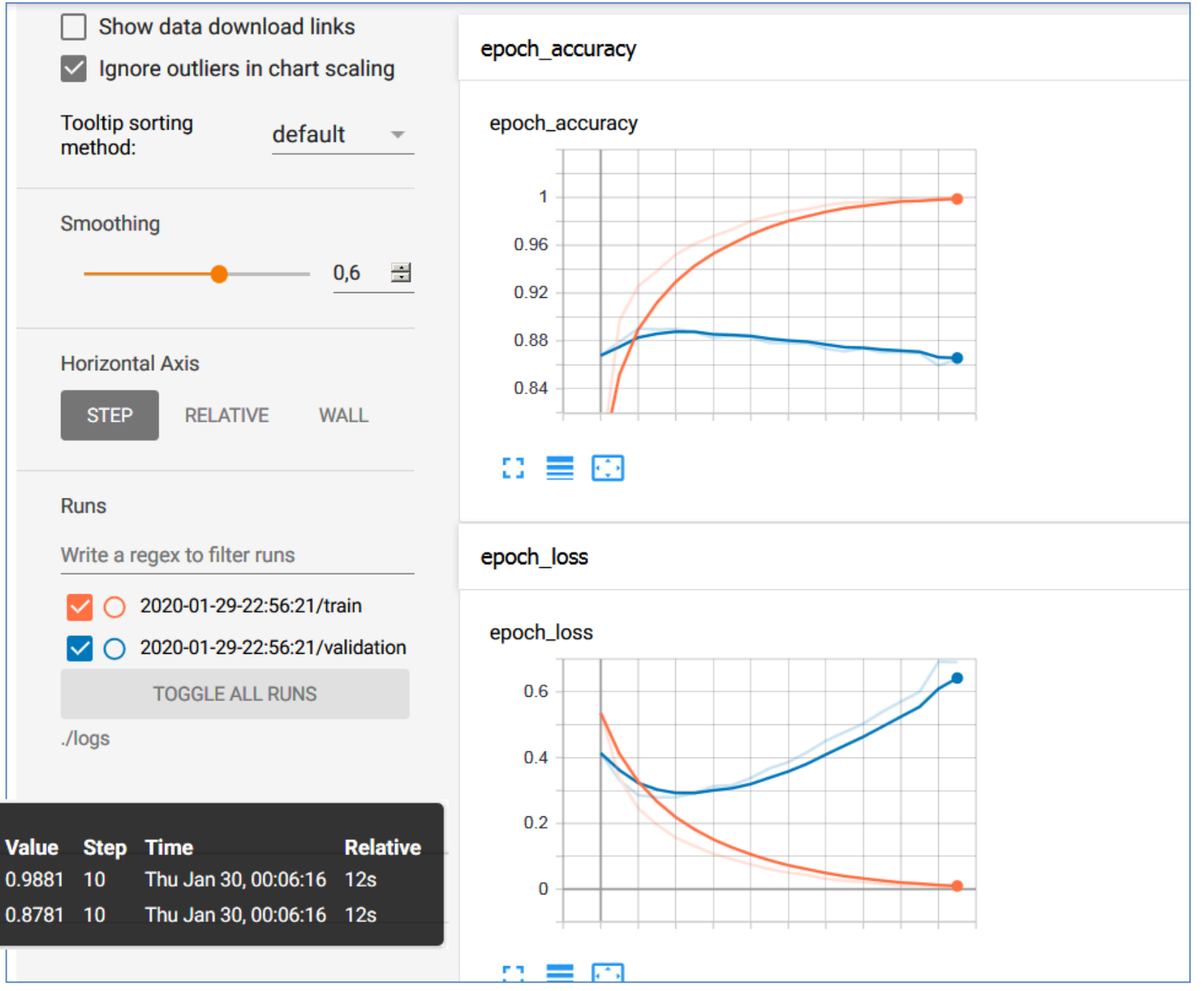
```
# UWAGA
# https://github.com/tensorflow/tensorboard/issues/2084
# musi być "profile_batch=0", bo inaczej nie wyswietla się poprawnie wykres accuracy
```

Przykład 3

```
# Budowa sieci neuronowej, kompilowanie, trenowanie
model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(16, activation='relu', input_shape=(10000,)),
  tf.keras.layers.Dense(16, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(x=partial_x_train,
                    y=partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val),
                    callbacks=callbacks_list)
```

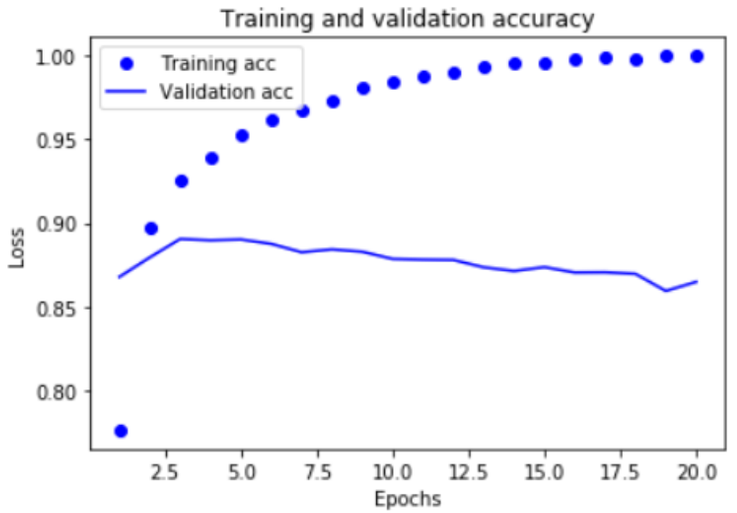
Przykład 3

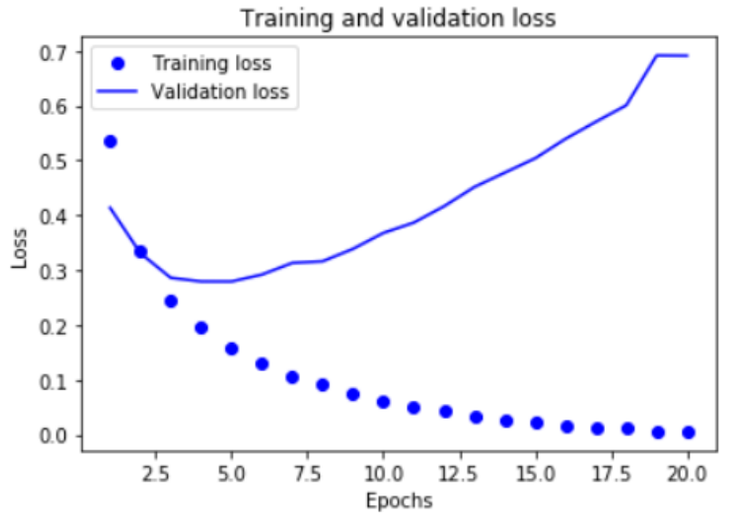


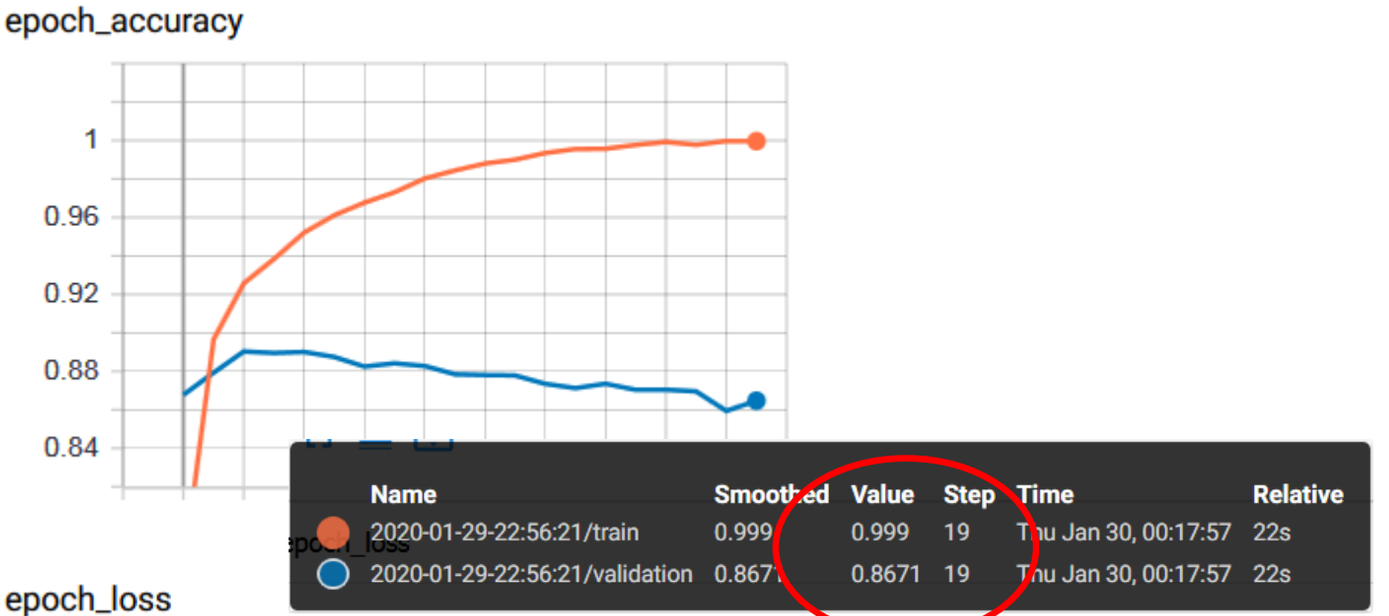


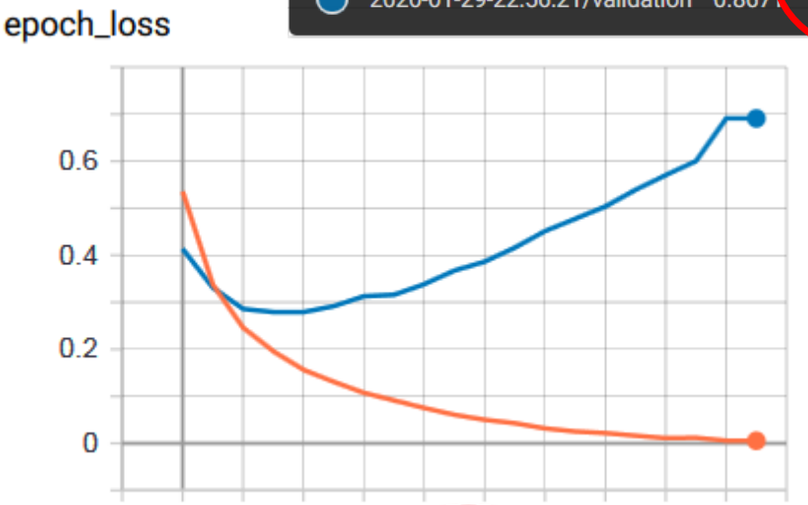
Name	Smoothed	Value	Step	Time	Relative
2020-01-29-22:56:21/train	0.9804	0.9881	10	Thu Jan 30, 00:06:16	12s
2020-01-29-22:56:21/validation	0.8803	0.8781	10	Thu Jan 30, 00:06:16	12s

Przykład 3










```
Epoch 20/20
15000/15000 [==============================] - 1s 80us/sample - loss: 0.0078 - accuracy: 0.9990 - val_loss: 0.6742 - val_accuracy: 0.8671
```

Podstawowe problemy SN

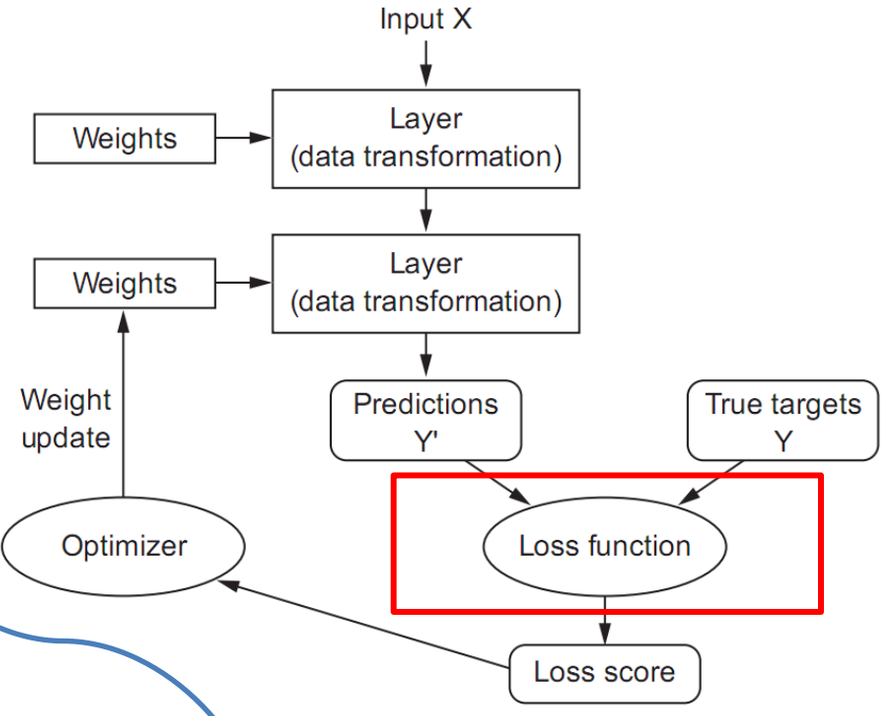
1. Nadmierne dopasowanie do danych uczących
2. Zbyt słabe dopasowanie do danych uczących
3. Za mało danych uczących
   technika augmentacji
4. Kompromis pomiędzy optymalizacją sieci a możliwościami uogólniania

Regularyzacja – sposób radzenia sobie z nadmiernym dopasowaniem

Regularyzacja

Ograniczamy złożoność sieci poprzez przyjmowanie tylko małych wartości wag

- wówczas model staje się bardziej regularny
- implementuje się go poprzez dodanie do funkcji straty kosztu związanego z dużymi wartościami wag
- regularyzacja **L1** - koszt jest dodawany proporcjonalnie do bezwzględnej wartości współczynników wag (normy L1 wag)
- regularyzacja **L2** - koszt jest dodawany proporcjonalnie do kwadratu wartości współczynników wag (normy L2 wag)



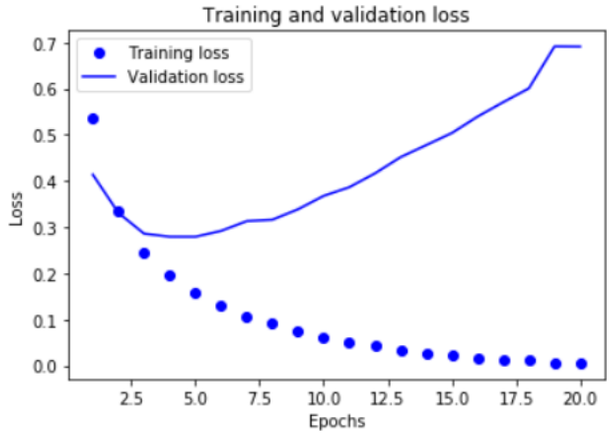
$$L(x,y) \equiv \sum_{i=1}^{n}(y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{n} \theta_i^2$$

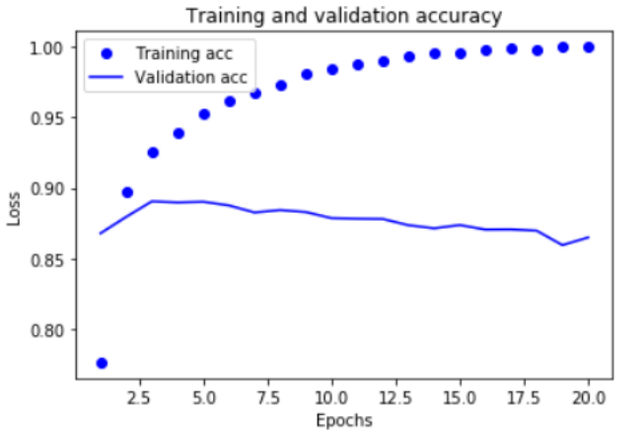
$$L(x,y) \equiv \sum_{i=1}^{n}(y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{n} |\theta_i|$$

Zapobieganie nadmiernemu dopasowaniu

1. Redukcja rozmiaru sieci

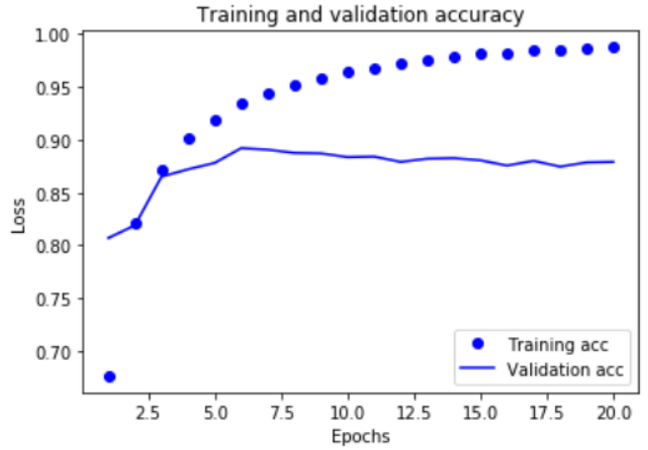
BYŁO






```
model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(4, activation='relu', input_shape=(NUMWORDS,)),
  tf.keras.layers.Dense(4, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

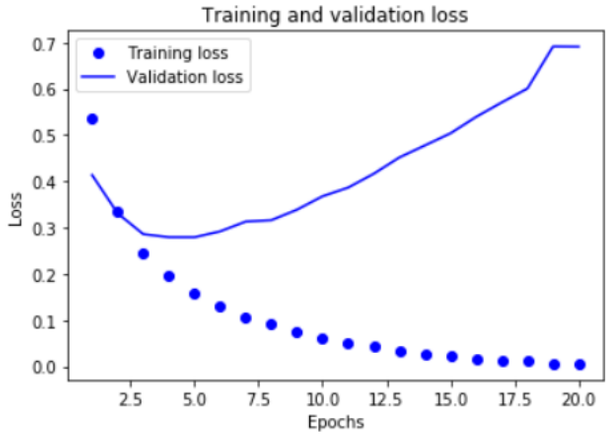
JEST

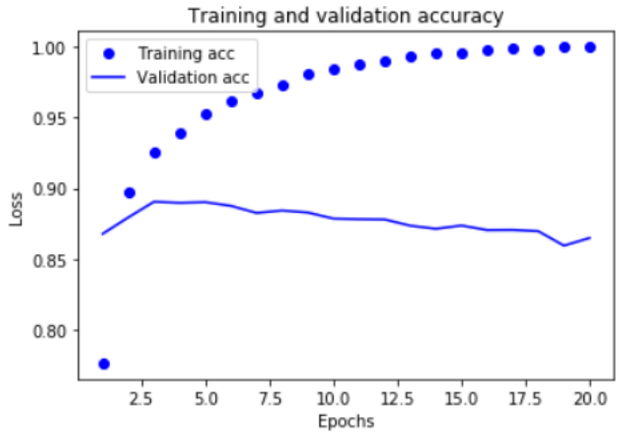


Zapobieganie nadmiernemu dopasowaniu

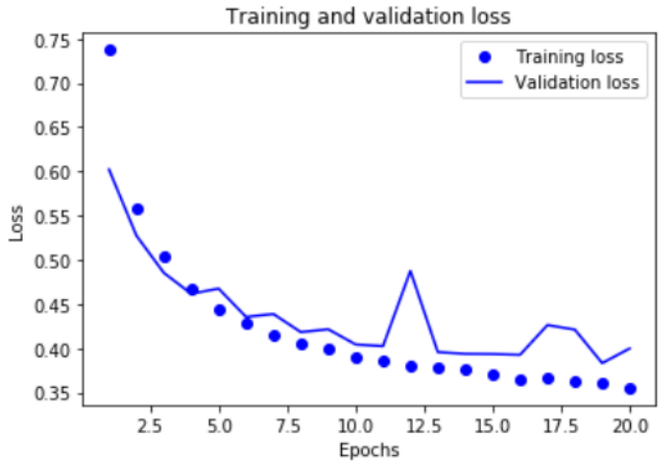
2. Dodanie regularyzacji L2

BYŁO

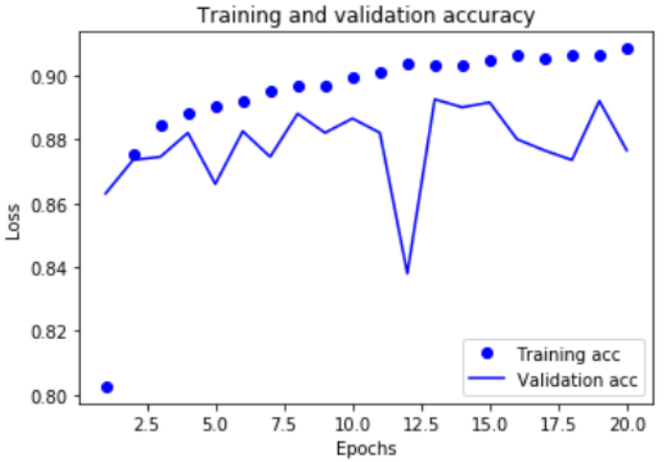




JEST



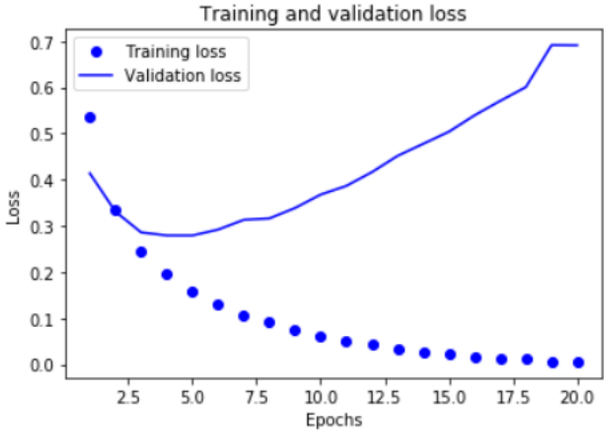

```
from keras import regularizers

model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(16,
                        kernel_regularizer=regularizers.l2(0.01),
                        activation='relu',
                        input_shape=(NUMWORDS,)),
  tf.keras.layers.Dense(16,
                        kernel_regularizer=regularizers.l2(0.01),
                        activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

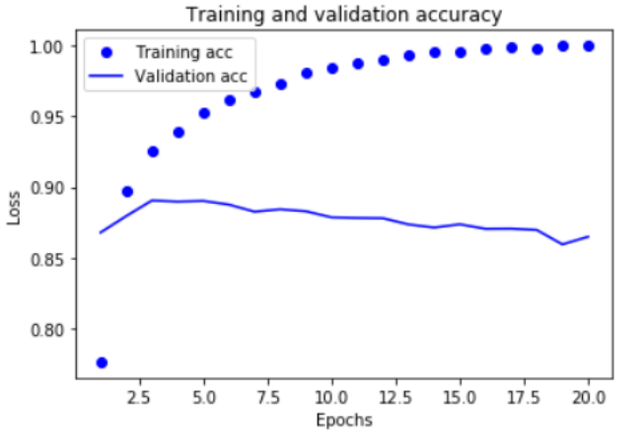


Zapobieganie nadmiernemu dopasowaniu

3. Technika dropout

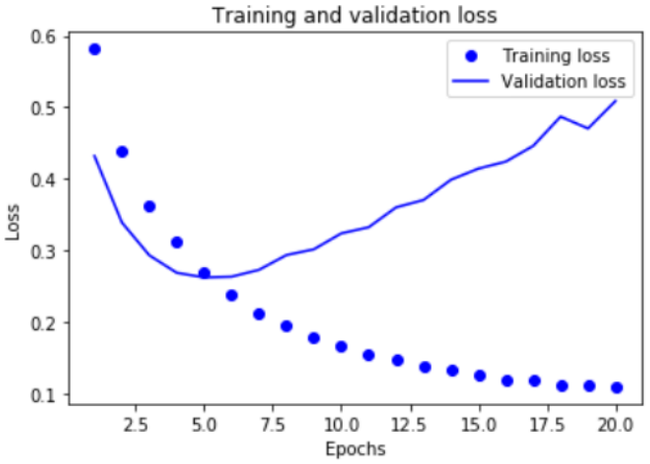
BYŁO

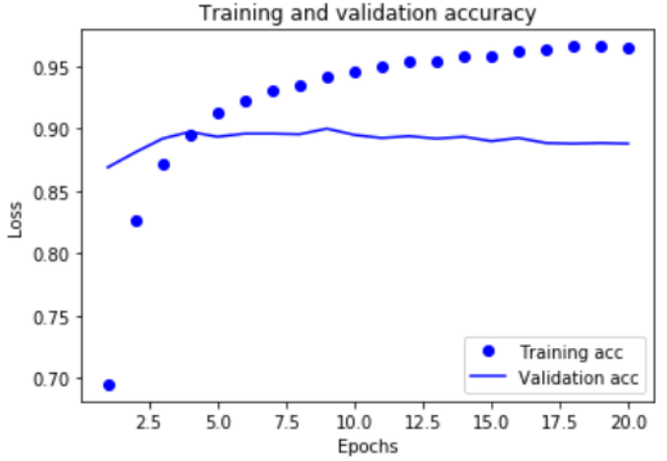





```
model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(16, activation='relu', input_shape=(NUMWORDS,)),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Dense(16, activation='relu'),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

JEST





Przykład 4, Uczenie głębokie

Wyniki klasyfikacji zbioru MNIST

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model.compile(tf.keras.optimizers.RMSprop(),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(test_loss)
print(test_acc)
```

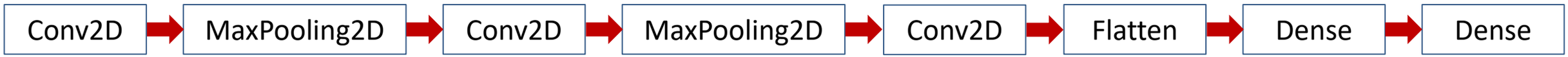
```
10000/10000 [==============================] - 1s 77us/sample - loss: 0.0330 - accuracy: 0.9931
0.03297394720344993
0.9931
```

Przykład 4, Uczenie głębokie

```
%tensorflow_version 2.x

import tensorflow as tf

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
                                 input_shape=(28, 28, 1)))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                36928
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
```

Obraz w odcieniach szarości, więc trzeci parametr = 1
w input_shape=(28, 28, 1)



Przykład 4, Uczenie głębokie

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                36928
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
```

Na wyjściu każdej warstwy Conv2D i MaxPooling2D pojawia się trójwymiarowy tensor o kształcie (wysokość, szerokość, kanały)

Pierwsza warstwa konwolucyjna zwraca mapę cech o wymiarze (26, 26, 32). 32 to ilość kanałów (filtrów). Zwracane są więc 32 przefiltrowane obrazy o wymiarach 26 x 26. Filtry mają postać łat o wymiarze 3 x 3 (to typowa wartość, może być też 4 x4 , 5 x5 ale zwykle nie więcej).

Wraz z zagłębianiem się w sieć wysokość i szerokość zmniejszają się a liczba kanałów zwiększa się (u nas 32, potem 64).

Ostatni tensor wyjściowy o kształcie (3, 3, 64) zostaje przekazany do klasyfikatora sieci gęstej (Dense). Klasyfikatory te przetwarzają jednowymiarowe wektory a nasze dane maja postać tensorów 3D. Trzeba je więc spłaszczyć, u nas do wektora o kształcie 3 x 3 x 64 = 576.

Klasyfikujemy obrazy do jednej z 10 klas, dlatego ostatnia warstwa generuje 10 wyjść. Funkcja aktywacji softmax sprawi, że na wyj. pojawią się prawdopodobieństwa przynależności do klasy.

Przykład 5

Sieć konwolucyjna z techniką augmentacji

- technika pozwalająca sztucznie powiększyć (wielokrotnie, zależy to od nas) liczność zbioru danych, u nas będą to obrazy ze zbioru „Cats vs. Dogs”
- 25000 zdjęć kotów i 25000 zdjęć psów w zbiorze uczącym. Zbiór testowy to 12500 zdjęć

Dogs vs. Cats

Create an algorithm to distinguish dogs from cats

215 teams · 6 years ago

kaggle

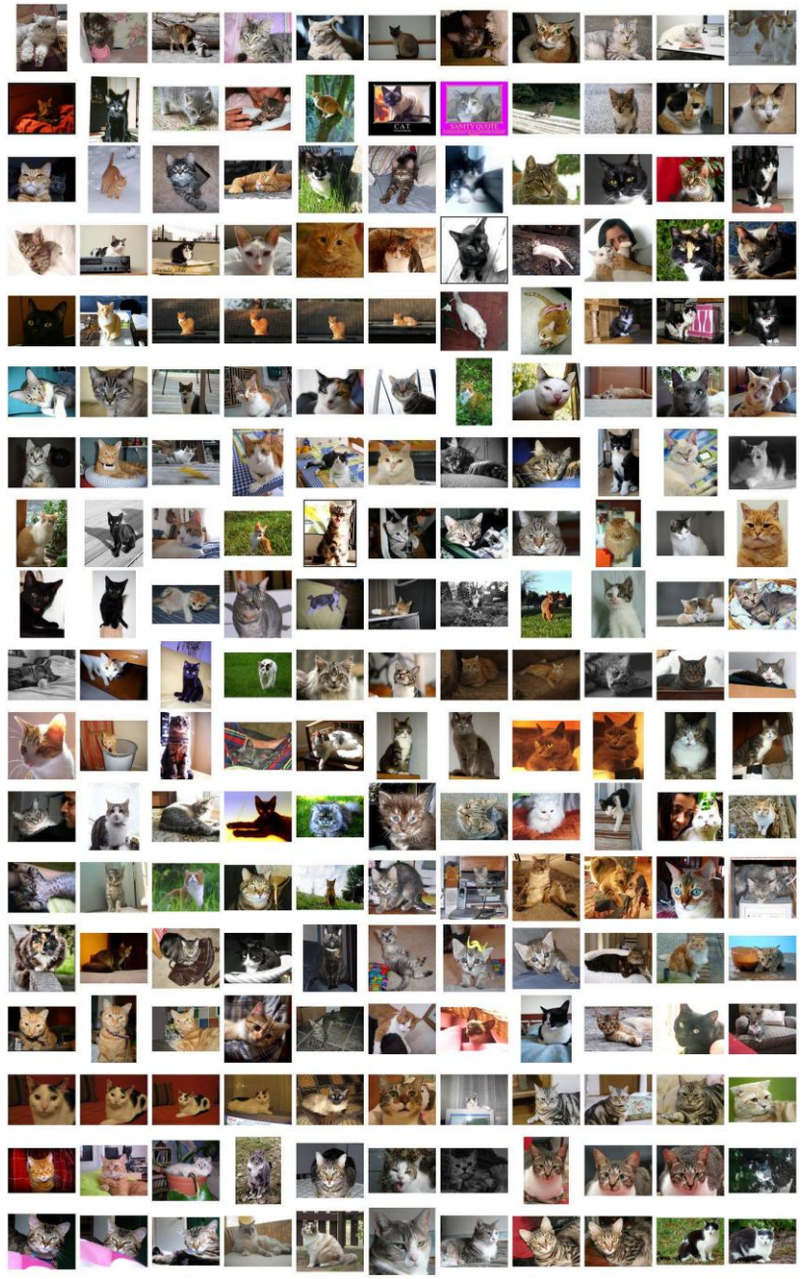
Przykład 5

Cats vs. Dogs





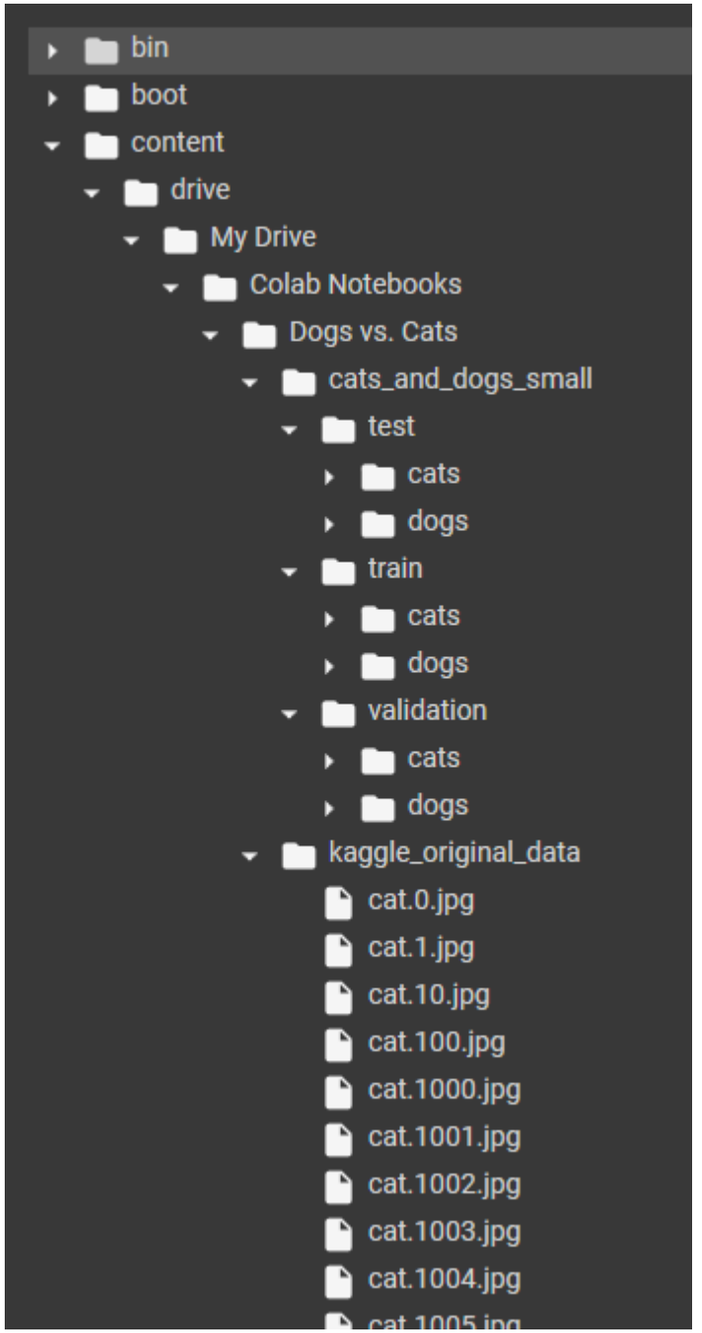


Przykład 5

Pracujemy na małym zbiorze danych

- total training cat images: 1000
- total training dog images: 1000
- total validation cat images: 500
- total validation dog images: 500
- total test cat images: 500
- total test dog images: 500



Przykład 5

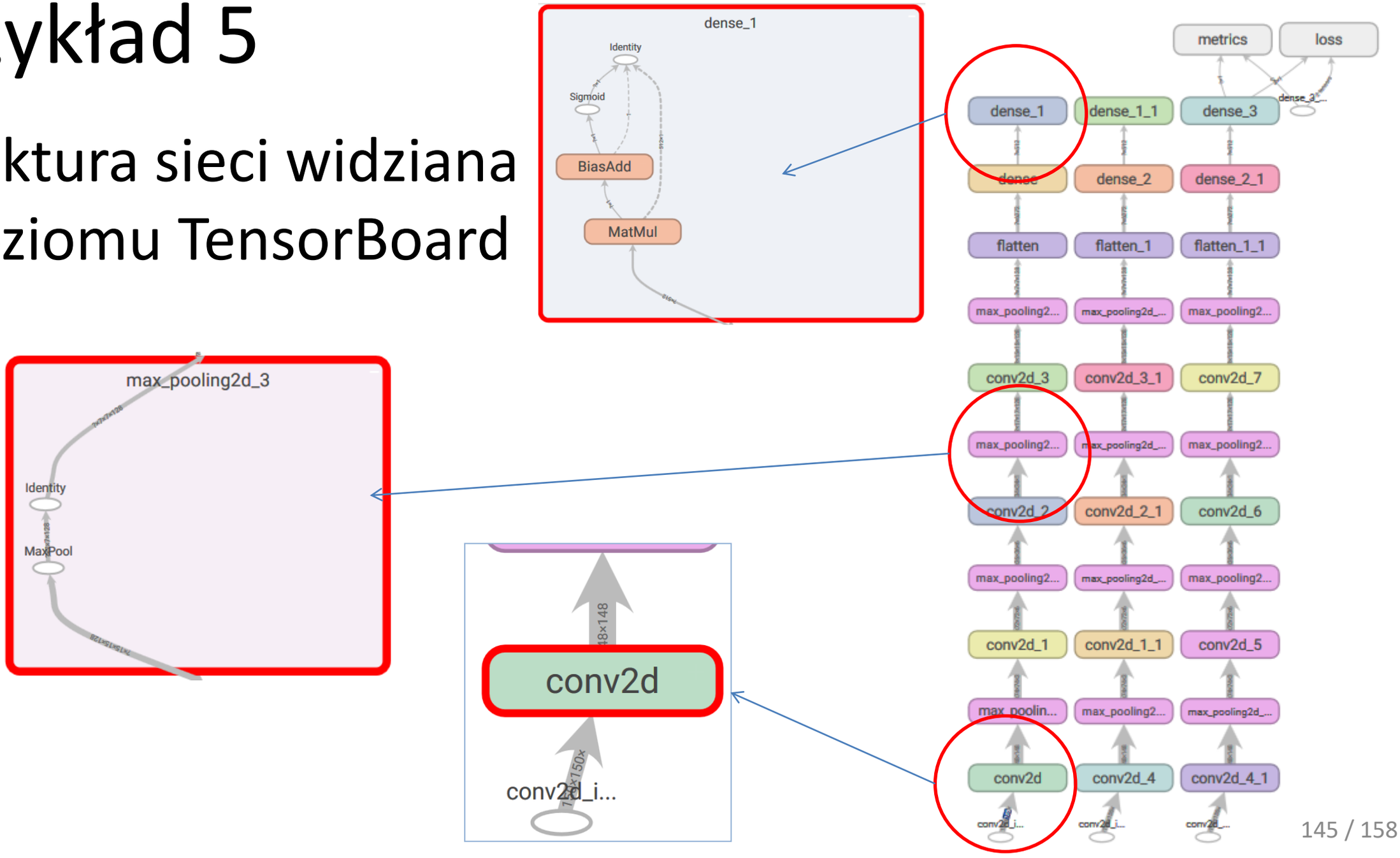
```
from keras import layers
from keras import models

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
                                 input_shape=(150, 150, 3)))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(128, (3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(128, (3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(512, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_2 (MaxPooling2	(None, 17, 17, 128)	0
conv2d_3 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_3 (MaxPooling2	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dense (Dense)	(None, 512)	3211776
dense_1 (Dense)	(None, 1)	513

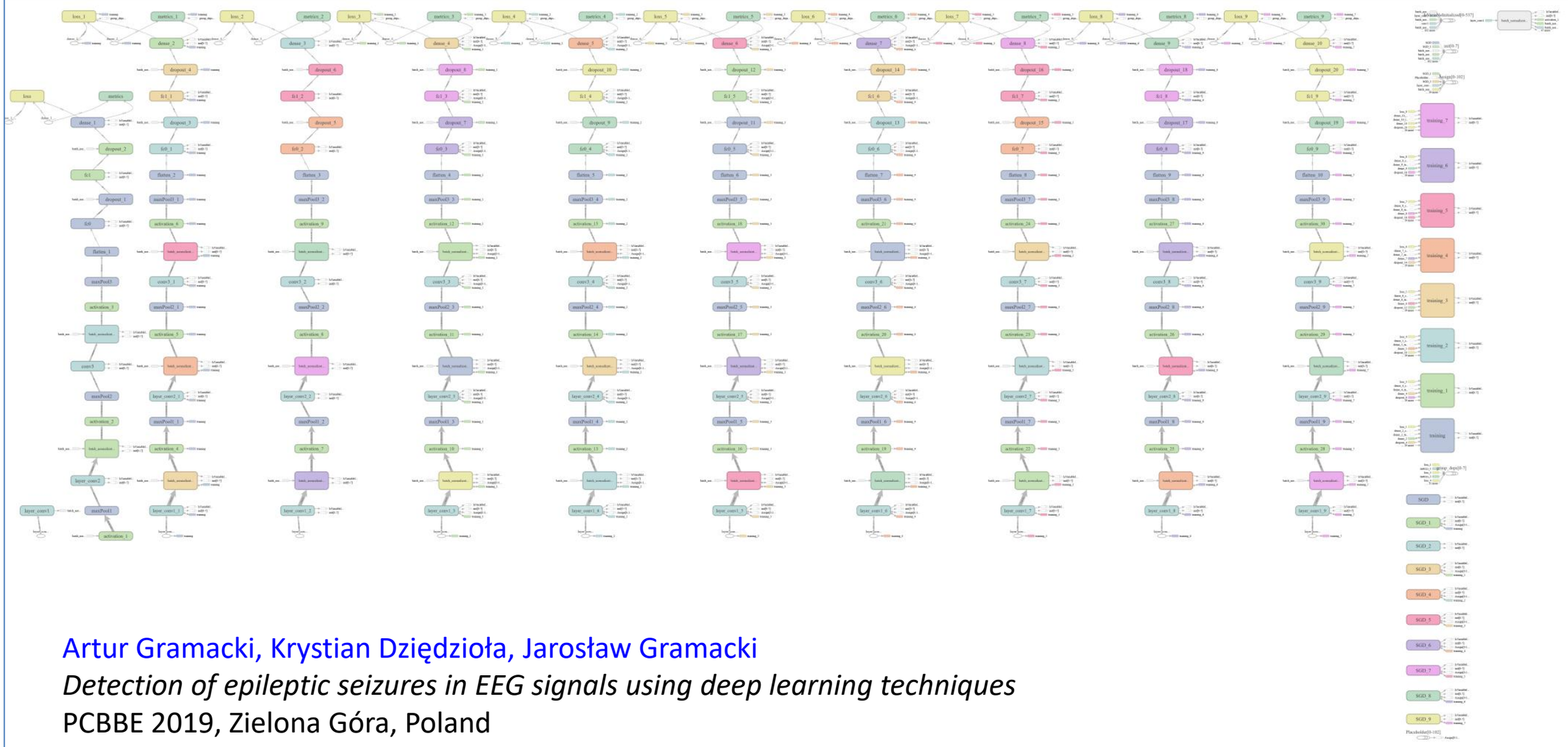
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0

Przykład 5

Struktura sieci widziana z poziomu TensorBoard



Inny przykład wewn. struktury CNN



Artur Gramacki, Krystian Dziędzioła, Jarosław Gramacki

Detection of epileptic seizures in EEG signals using deep learning techniques

PCBBE 2019, Zielona Góra, Poland

Inny przykład wewn. struktury CNN

Tak model wyglądał w Keras

Layer (type)	Output Shape	Param #
conv2D_1 (Conv2D)	(None, 100, 16, 64)	640
batch_norm_1 (BatchNormaliza	(None, 100, 16, 64)	256
activ_relu_1 (Activation)	(None, 100, 16, 64)	0
maxPool_1 (MaxPooling2D)	(None, 50, 8, 64)	0
conv2D_2 (Conv2D)	(None, 50, 8, 64)	36928
batch_norm_2 (BatchNormaliza	(None, 50, 8, 64)	256
activ_relu_2 (Activation)	(None, 50, 8, 64)	0
maxPool_2 (MaxPooling2D)	(None, 25, 4, 64)	0
conv2D_3 (Conv2D)	(None, 25, 4, 32)	18464
batch_norm_3 (BatchNormaliza	(None, 25, 4, 32)	128
activ_relu_3 (Activation)	(None, 25, 4, 32)	0
maxPool_3 (MaxPooling2D)	(None, 12, 2, 32)	0
flatten (Flatten)	(None, 768)	0
fully_conn_1 (Dense)	(None, 64)	49216
dropout_1 (Dropout)	(None, 64)	0
fully_conn_2 (Dense)	(None, 32)	2080
dropout_2 (Dropout)	(None, 32)	0
fully_conn_3 (Dense)	(None, 1)	33

Total params: 108,001
Trainable params: 107,681
Non-trainable params: 320

Przykład 5

Data preprocessing

- zdjęcia kotów i psów są bardzo różnej jakości i mają różne rozmiary. Na sieć muszą być podane jednak w postaci jednolitej i uporządkowanej (wybrano rozmiar: 150 x 150 pikseli)
- użyjemy tutaj ImageDataGenerator, który automatycznie przygotuje nam wymagane wsady tensorów

Przykład 5

```
from keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 150x150
        target_size=(150, 150),
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

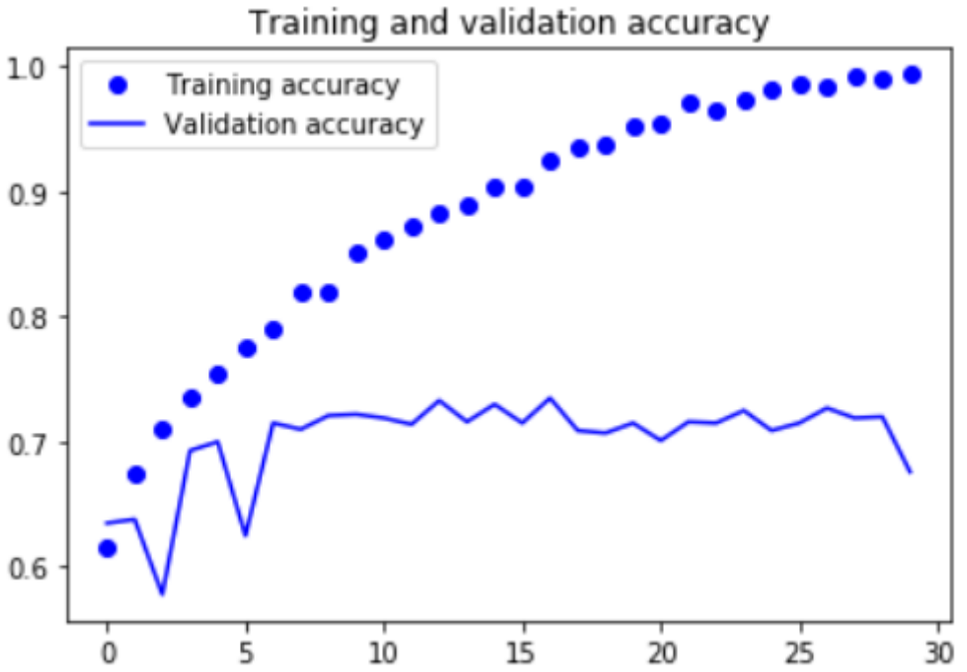
```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

```
for data_batch, labels_batch in train_generator:
    print('data batch shape:', data_batch.shape)
    print('labels batch shape:', labels_batch.shape)
    break
```

```
data batch shape: (20, 150, 150, 3)
labels batch shape: (20,)
```

Przykład 5

Wyniki trenowania (na razie bez augmentingu)

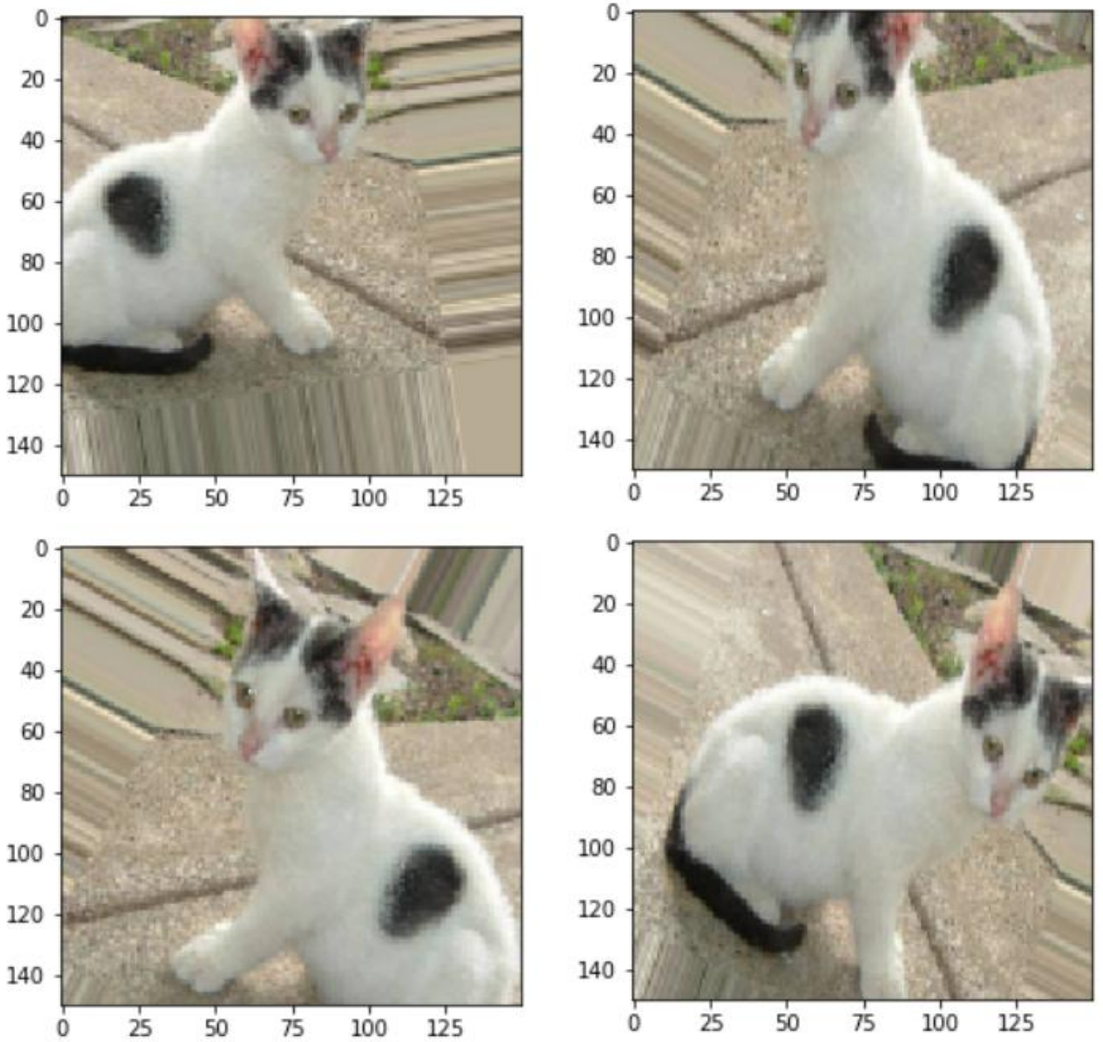


Widoczne wyraźne przeuczenie. Krzywe walidacyjne są „fatalne”

Przykład 5

Augmentacja

```
datagen = ImageDataGenerator(
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```



Przykład 5

Używając augmentacji warto też w sieć wbudować warstwę Dropout

– aby uniknąć przeuczenia, choć rozpłodzone obrazki nie są identyczne, niemniej są niewątpliwie bardzo podobne

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
                                 input_shape=(150, 150, 3)))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(128, (3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(128, (3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(512, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

Przykład 5

Trenujemy sieć na większej ilości danych

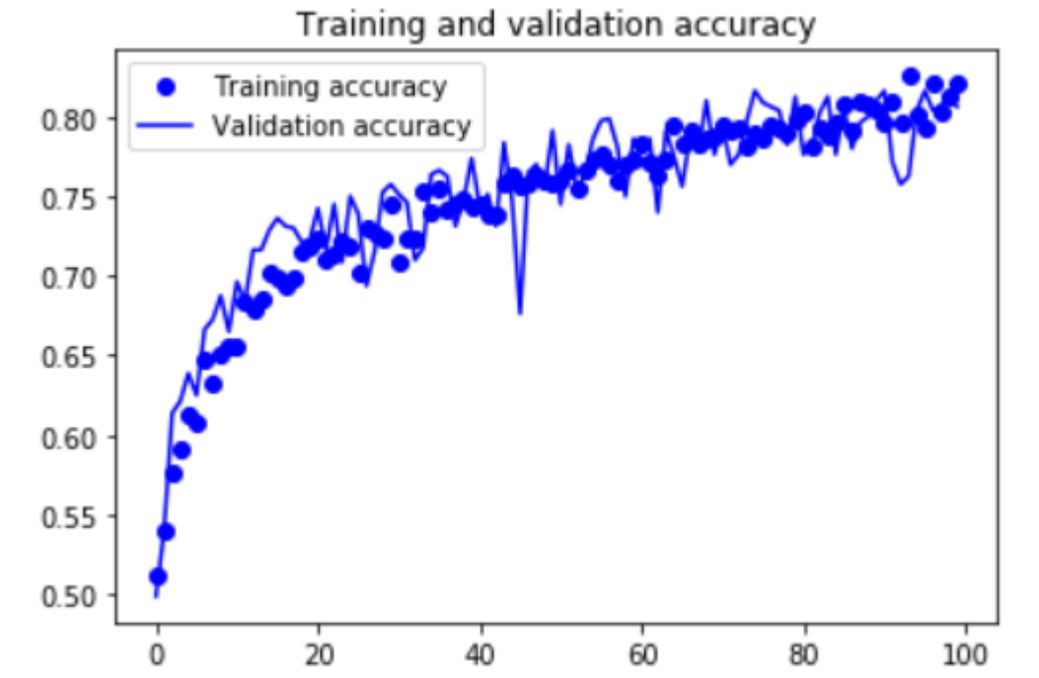
```
#history_2 = model.fit_generator(
history_2 = model.fit(
        train_generator,
        steps_per_epoch=100,
        epochs=100,
        validation_data=validation_generator,
        validation_steps=50)
```

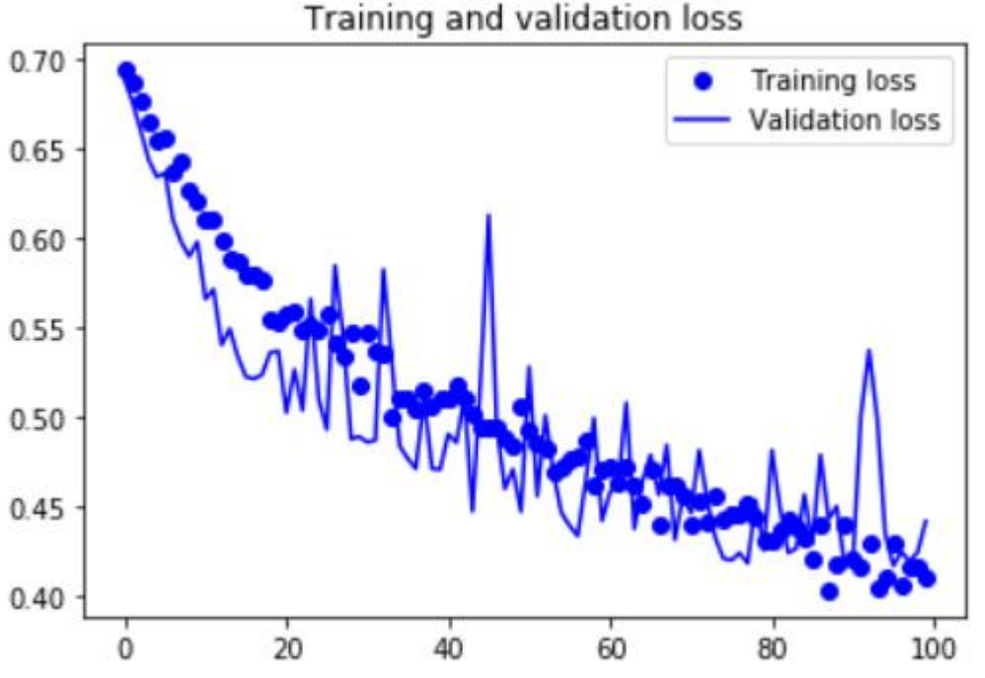
Gdy trenowanie trwa długo, warto zapisać jego wynik

```
import pickle

with open(WRK_DIR + 'history_2.pckl', 'wb') as f:
    pickle.dump(history_2.history, f)

with open(WRK_DIR + 'history_2.pckl', 'rb') as f:
    history_2_restored = pickle.load(f)
```

Przykład 5

Wyniki końcowe





Zniknął praktyczne efekt overfitting-u !!!
Ale tym razem musieliśmy użyć aż 100 epok (a być może trzeba by użyć ich jeszcze więcej, aby wyniki się „nasyciły”)

Czego nie omówiono

- Wizualizacja efektów uczenia (wyniki z warstw pośrednich, co się dzieje w środku sieci CNN)
- Przetwarzanie danych tekstowych
- Przetwarzanie danych o charakterze szeregów czasowych
- Inne rodzaje sieci CNN, nie tylko sekwencyjne (Sequential)
    - wymagane jest używanie funkcjonalnego interfejsu API Keras-a
- … i wiele, wiele innych zagadnień

Python i tablice (array)

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

https://docs.scipy.org/doc/numpy/user/quickstart.html

Python i tablice (array)

```
>>> x = np.array([2,3,4])
# Tak będzie błąd (muszą być nawiasy kwadratowe):
>>> x = np.array(1,2,3,4)

>>> x.dtype
dtype('int64')

>>> print(x)
[1.2 3.5 5.1]

>>> x = np.array([1.2, 3.5, 5.1])
>>> x.dtype
dtype('float64')
```

```
# Wiele wymiarów
>>> x = np.array([(1, 2, 3), (4, 5, 6)])

# Można też tak
>>> x = np.array([[1, 2, 3],
                  [4, 5, 6]])
>>> print(x)
[[1 2 3]
 [4 5 6]]

>>> print(x.shape)
(2, 3)

>>> print(x.ndim)
2
```

Python i tablice (array)

```
# Analogicznie więcej wymiarów.
# Cztery obrazy 3 x 3 (x, o, +, ),
# czyli tensor 4 x 3 x 3
>>> x3d = np.array([[[1, 0, 1],
                     [0, 1, 0],
                     [1, 0, 1]],

                    [[2, 2, 2],
                     [2, 0, 2],
                     [2, 2, 2]],

                    [[0, 3, 0],
                     [3, 3, 3],
                     [0, 3, 0]],

                    [[0, 0, 0],
                     [4, 4, 4],
                     [0, 0, 0]]])
```

UWAGA – konsekwentne stosowanie właściwych wcięć w kodach pozwala (jako tako) zapanować nad „nawiasologią”

```
>>> print(x3d)
# lub
>>> x3d[:,:,:]
[[[1 0 1]
  [0 1 0]
  [1 0 1]]

 [[2 2 2]
  [2 0 2]
  [2 2 2]]

 [[0 3 0]
  [3 3 3]
  [0 3 0]]

 [[0 0 0]
  [4 4 4]
  [0 0 0]]]

>>> print(x3d.shape)
(4, 3, 3)

>>> print(x3d.ndim)
3
```

Python i tablice (array)

```
>>> x = np.arange(10)**2

>>> print(x)
[ 0 1 4 9 16 25 36 49 64 81]

# UWAGA: indeksowanie zaczyna się od ZERA,
# nie od JEDEN!
# Więc teraz wyświetli się TRZECI element
# listy, czyli "4"
>>> print(x[2])
4

# Teraz wyświetlą się elementy
# TRZECI, CZWARTY i PIĄTY
# (czyli elementy o indeksach 2, 3, 4)
# Trzeba się do tego przyzwyczaić :-)
>>> x[2:5]
array([ 4, 9, 16])
```

```
>>> x3d[0]
array([[1, 0, 1],
       [0, 1, 0],
       [1, 0, 1]])

>>> x3d[3]
array([[0, 0, 0],
       [4, 4, 4],
       [0, 0, 0]])

# lub (chyba lepiej, bo od razu wiemy z jakim
# obiektem mamy do czynienia)
>>> x3d[0,:,:]
array([[1, 0, 1],
       [0, 1, 0],
       [1, 0, 1]])

# To samo, co wyżej
x3d[0:1,:,:]
array([[1, 0, 1],
       [0, 1, 0],
       [1, 0, 1]])
```

Python i tablice (array)

```
# Element DRUGI, czyli ten o indeksie PIERWSZYM
>>> x3d[1:2,:,:]
array([[[2, 2, 2],
        [2, 0, 2],
        [2, 2, 2]]])

# Elementy TRZECI i CZWARTY, czyli
# te o indeksach DWA i TRZY
>>> x3d[2:4,:,:]
array([[[0, 3, 0],
        [3, 3, 3],
        [0, 3, 0]],

       [[0, 0, 0],
        [4, 4, 4],
        [0, 0, 0]]])
```

```
array([[[1, 0, 1],
        [0, 1, 0],
        [1, 0, 1]],

       [[2, 2, 2],
        [2, 0, 2],
        [2, 2, 2]],

       [[0, 3, 0],
        [3, 3, 3],
        [0, 3, 0]],

       [[0, 0, 0],
        [4, 4, 4],
        [0, 0, 0]]])
```

```
# Środkowy element każdego "obrazka"
>>> x3d[:,1,1]
array([1, 0, 3, 4])

# Środkowa kolumna każdego "obrazka"
>>> x3d[:,:,1]
array([[0, 1, 0],
       [2, 0, 2],
       [3, 3, 3],
       [0, 4, 0]])

# Lewe górne rogi każdego "obrazków"
>>> x3d[:,0:2,0:2]
array([[[1, 0],
        [0, 1]],

       [[2, 2],
        [2, 0]],

       [[0, 3],
        [3, 3]],

       [[0, 0],
        [4, 4]]])
```

Dziękuję za uwagę