Deep Learning: Session 07

Convolutional Nets

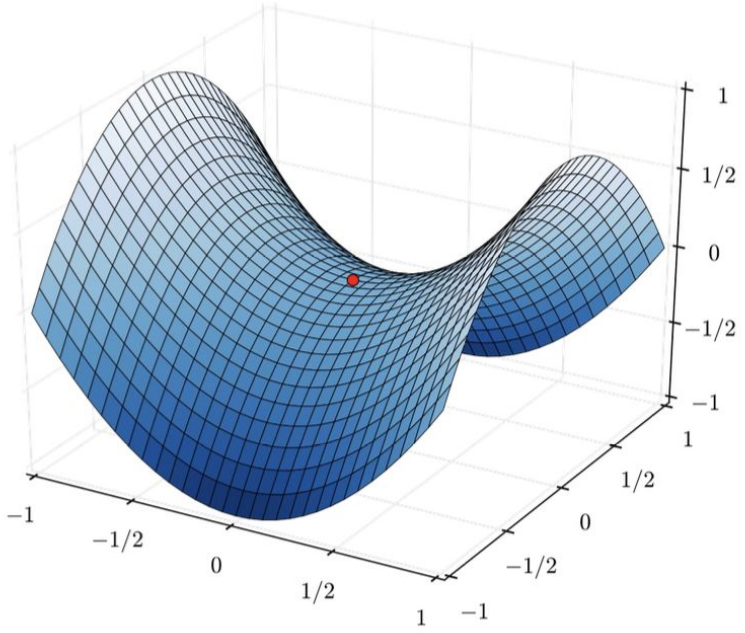
Outline

1. Recap
2. How do we see ?
3. The convolution operator
4. Architecture

Recap: Local Minima

- The intuitions in 2 or 3 dimensions are not valid for high dimensional spaces, where we don't know what is exactly happens…

- But where we know that local minima tends to be Saddle Points rather than actual local minima



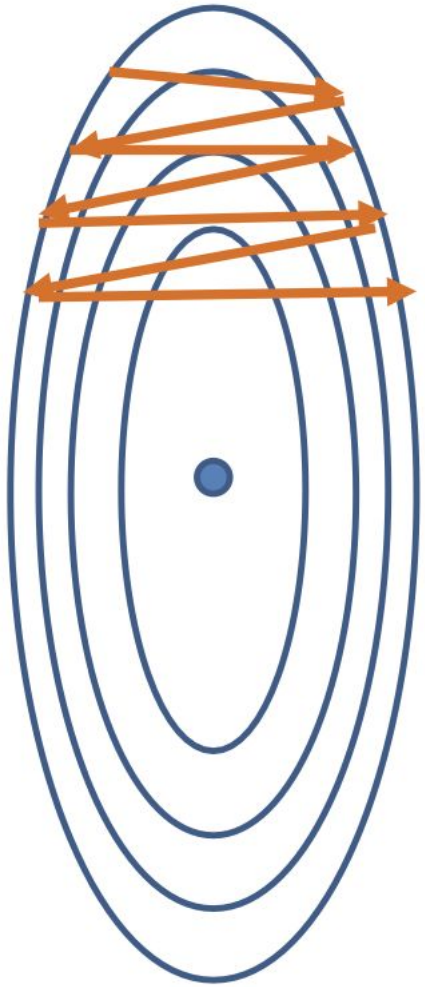
Recap: Optimization

- There are several ways to make faster Gradient Descent.
- Some of them are based on the amount of data used to compute the gradients (e.g., SGD using minibatches).
- Others are based on optimize the Gradient Descent algorithm itself
  - Learning Rate Decay
  - Momentum
  - RMSProp
  - Adam

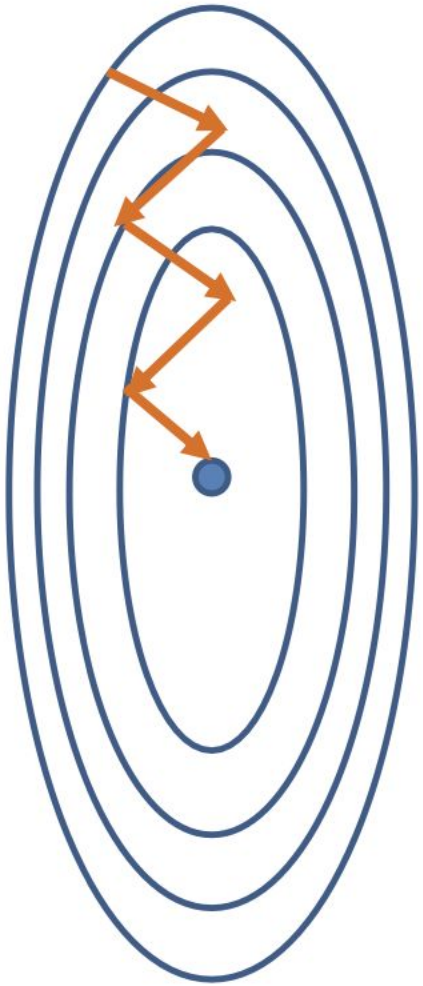
Recap: SGD with Mini-Batches

- Shuffle training data (M)
- Pick up a mini-batch (X << M)
- Compute gradient for those X training points (step)
- Upgrade gradient.
- Epoch is an iteration over all the training data. So #steps = M / X

Recap: Momentum RMSprop and Adam



Recap: Momentum RMSprop and Adam



Recap: Momentum RMSprop and Adam

- Momentum and RMSProp are techniques to drive gradient descent, avoiding large steps in some directions that are cancel out in the following iteration.

- Adam combines momentum and RMSProp.

Recap: Callbacks

- **Callbacks in Keras/Tensorflow**

https://keras.io/api/callbacks/

You can pass a list of callbacks (as the keyword argument `callbacks`) to the `.fit()` method of a model:

```
my_callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=2),
    tf.keras.callbacks.ModelCheckpoint(filepath='model.{epoch:02d}-{val_loss:.2f}.h5'),
    tf.keras.callbacks.TensorBoard(log_dir='./logs'),
]
model.fit(dataset, epochs=10, callbacks=my_callbacks)
```

The relevant methods of the callbacks will then be called at each stage of the training.

Recap: Hyperparams

- Deep nets have many hyperparams:
  - Learning rate
  - Number of hidden layers
  - Number of hidden units
  - Size of minibatches
  - Hyperparams related with the optimization algorithm
  - ...

Recap: Hyperparams

- Two different approaches:

  – One model where you carefully change hyperparams while training by looking at val_loss.

  – Different models where you just look at val_loss

- When running different models don't use exhaustive grids of every pair of hyperparams.

- There are packages that search for you the best configuration (e.g. Keras Tuner)

Recap: Hyperparams

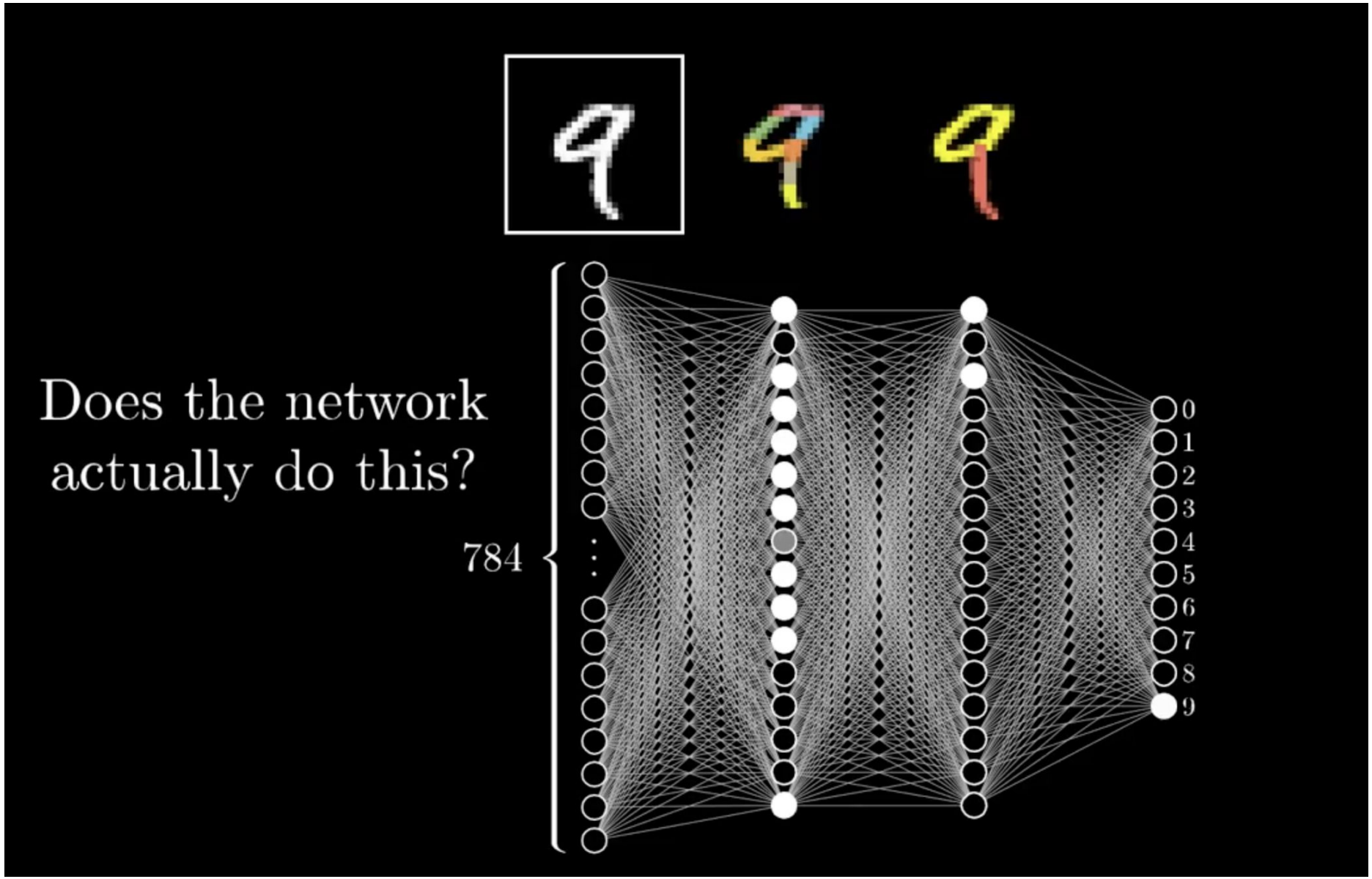
Keras Tuner

https://keras.io/keras_tuner/

KerasTuner

KerasTuner is an easy-to-use, scalable hyperparameter optimization framework that solves the pain points of hyperparameter search. Easily configure your search space with a define-by-run syntax, then leverage one of the available search algorithms to find the best hyperparameter values for your models. KerasTuner comes with Bayesian Optimization, Hyperband, and Random Search algorithms built-in, and is also designed to be easy for researchers to extend in order to experiment with new search algorithms.

Recap



Outline

1. Recap
2. How do we see ?
3. The convolution operator
4. Architecture

ANNs for images?

A single hidden layer with 1024 units:

- 200 x 200 image -> ~40M weights

ANNs for images?

A single hidden layer with 1024 units:

- 200 x 200 image -> ~ 40M weights
- 1000 x 1000 image -> ~ 130M weights

ANNs for images?

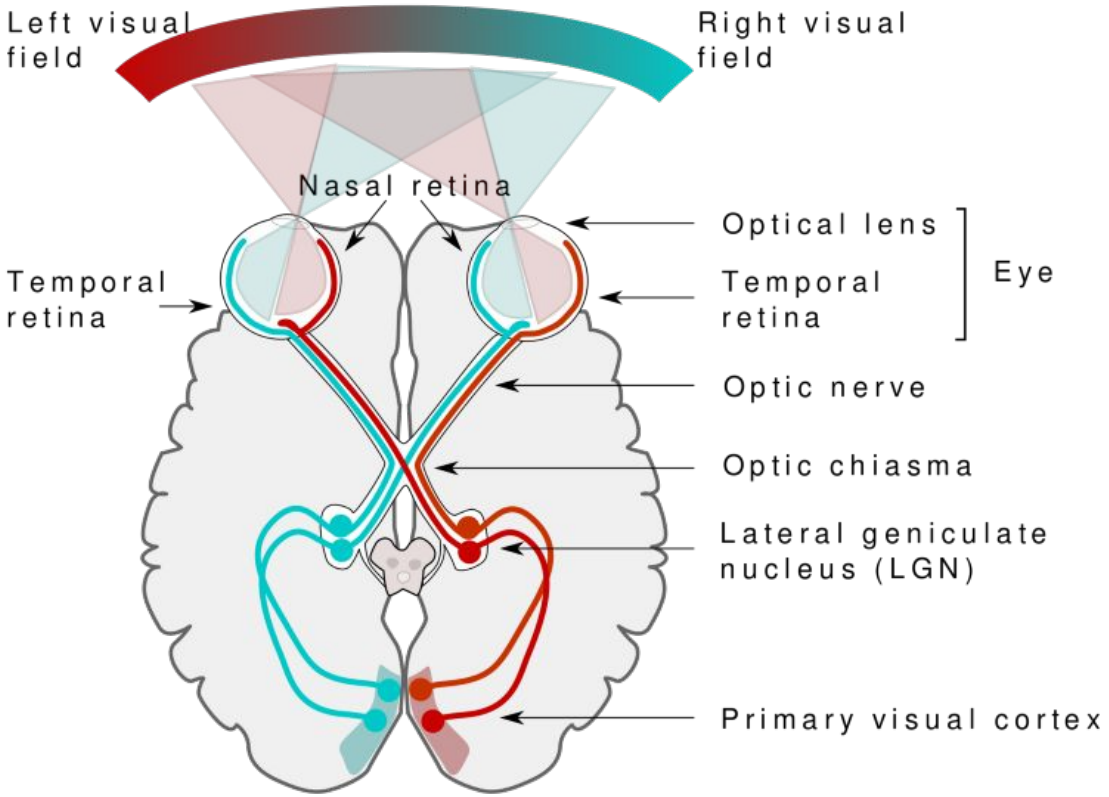
A single hidden layer with 1024 units:

- 200 x 200 image -> ~ 40M weights
- 1000 x 1000 image -> ~ 130M weights
- 4000 x 4000 image -> ~ freaking too many !

Convolutional Neural Networks

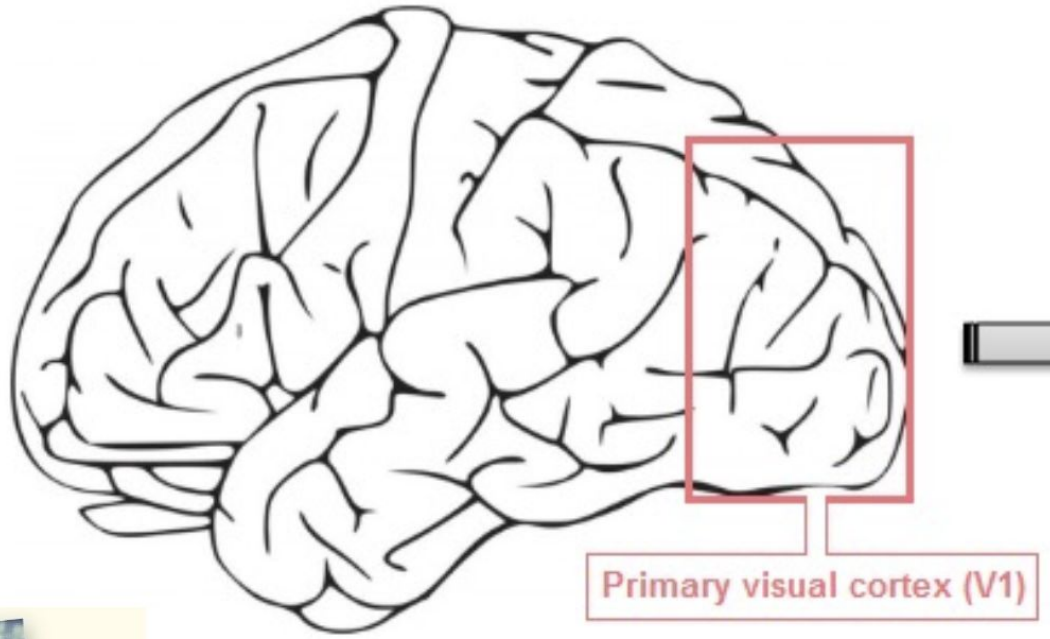
- Convolutional neural networks or convnets or CNNs are a type of neural networks specialized on images.
- Feed forward networks (multilayer perceptron) consider images as any other input, without taking advantage of the intrinsic nature of the images.
- CNNs appears in 1998 introduced in a paper by Le-cun and Bengio. They were inspired on the research in the 1950s and 1960s on the visually perception on the brain.
- The first Convolutional Neural Network was called Le-Net5 and was able to perform the MNIST task (recognize written digits)

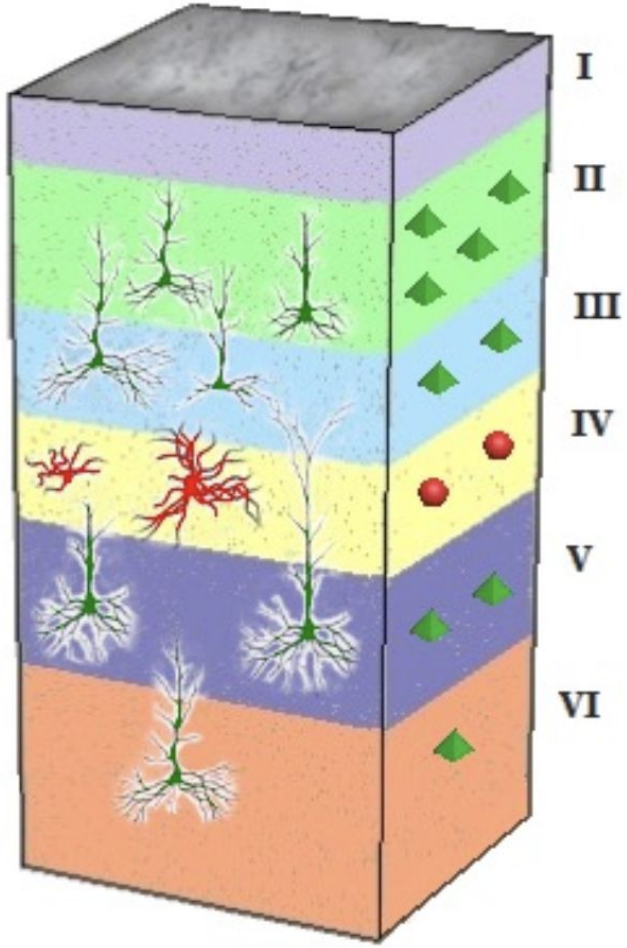
How our brain sees an image

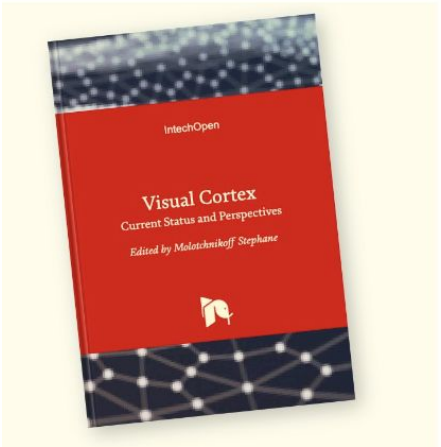


- Light receptors in your eyes send signals to the primary visual cortex through the optic nerve
- Primary visual cortex interprets what your eyes see
- There is a hierarchical structure of neurons (layers) that play an important role cause somehow extracts and propagate different patterns.

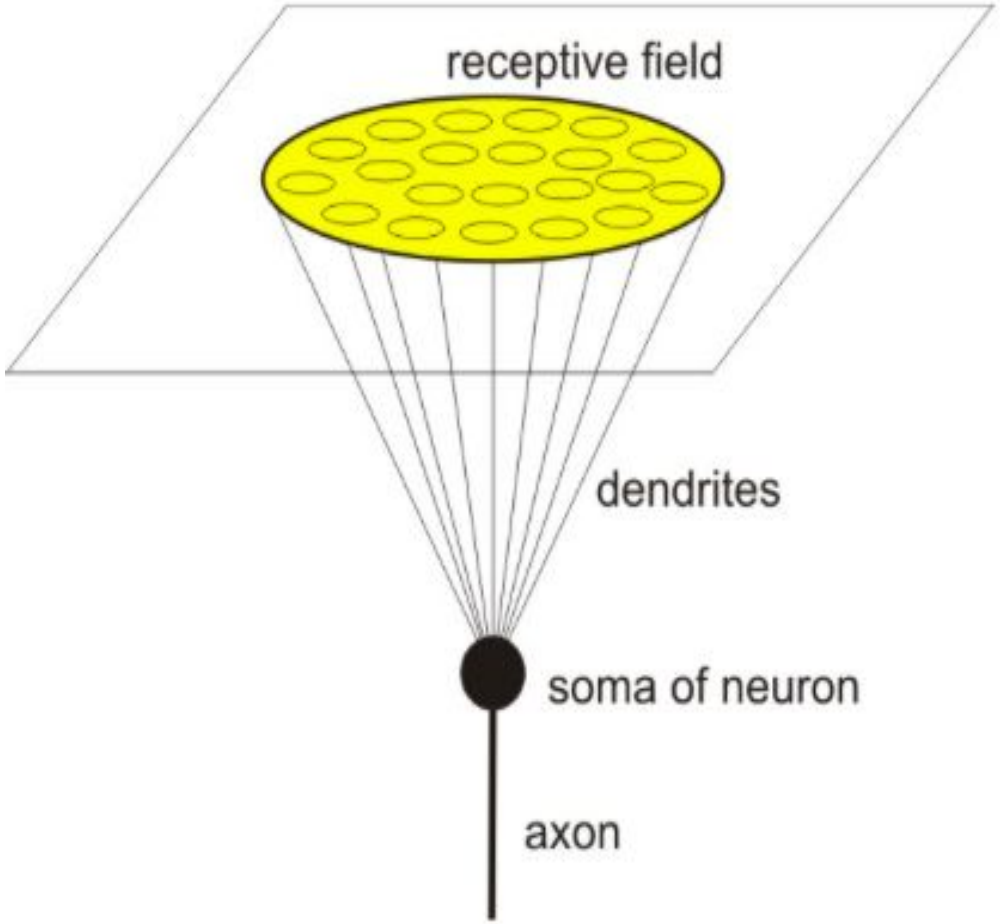
How our brain sees an image







How our brain sees an image



- Visual neurons have a **receptive field**
- The receptive field is the specific region of the retina in which something could fire the neuron.
- Some visual neurons have large receptive fields and this has an important effect: **location invariance**
- Location invariance: if some pattern is able to fire the neuron, the location of this pattern does not matter as long as it is within the receptor field
- … in other words, this visual neuron is able to find a pattern no matter how is located in the receptive field.

Outline

1. ANNs recap
2. DNNs – MNIST problem
3. DNNs - Key Concepts
4. CNNs – Motivation
5. **CNNs – Convolution**
6. CNNs
7. Transfer Learning

The Convolution Operator

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

=

The Convolution Operator

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

=

The Convolution Operator

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

=

4		

The Convolution Operator

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

=

4	1	

The Convolution Operator

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

=

4	1	6

The Convolution Operator

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

=

4	1	6
5		

The Convolution Operator

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

=

4	1	6
5	8	

The Convolution Operator

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

=

4	1	6
5	8	7

The Convolution Operator

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

=

4	1	6
5	8	7
5		

The Convolution Operator

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

=

4	1	6
5	8	7
5	3	

The Convolution Operator

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

=

4	1	6
5	8	7
5	3	3

The Convolution Operator

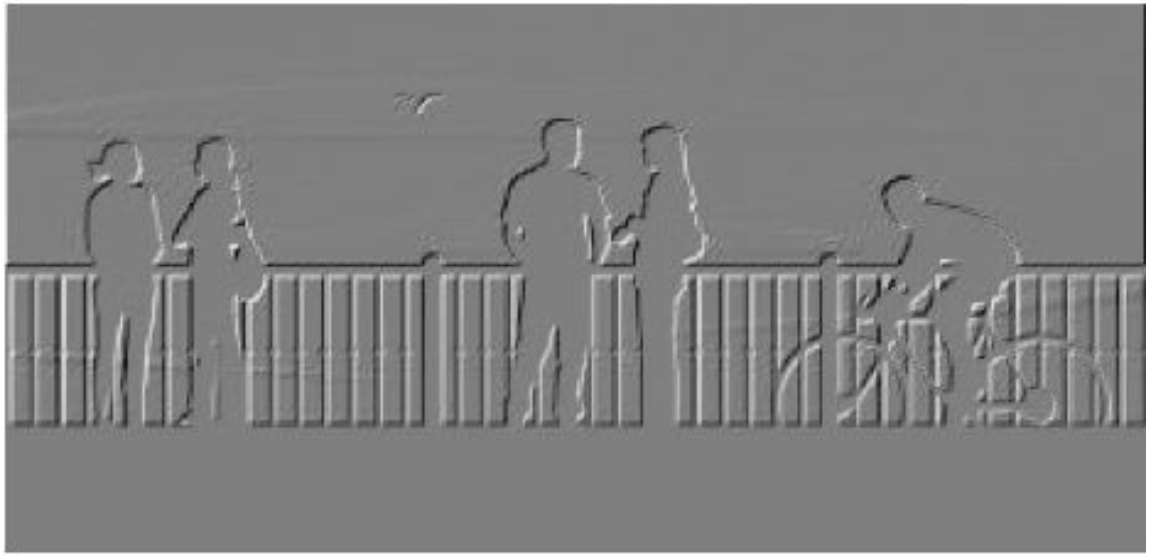
Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

The Convolution Operator: Edge Detector

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

*



The Convolution Operator: Edge Detector Exercise

Let’s play: Session 7: Convolutional Neural Networks

Convolutional Operator Code

ConvolutionalOperator_template.ipynb

boardwalk.jpg

Open ConvolutionalOperator_template in Google Colab

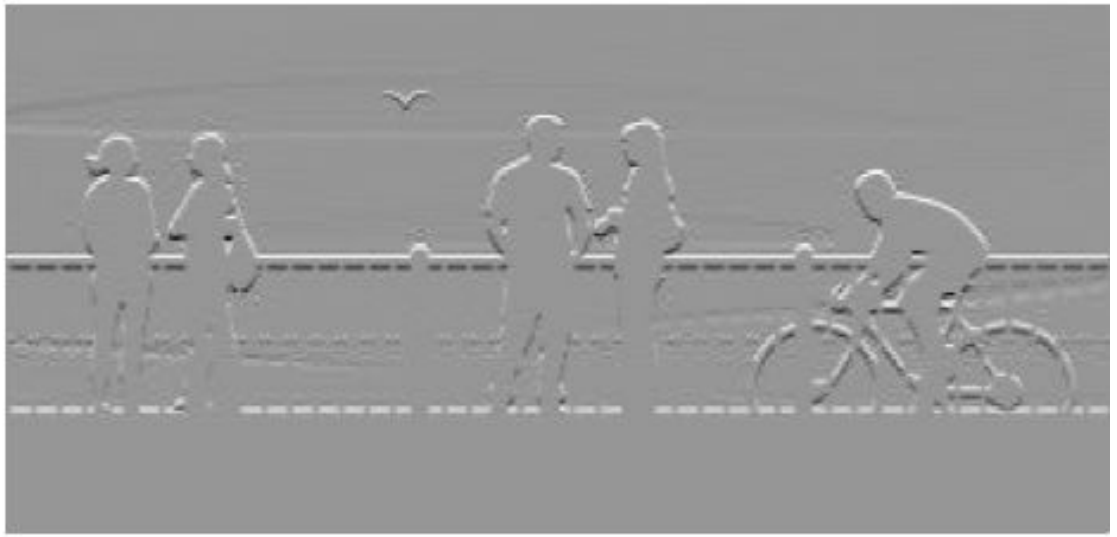
Upload boardwalk.jpg to Google Colab (Files/Upload)

The Convolution Operator: Edge Detector

Exercise: Create a horizontal edge detector

?	?	?
?	?	?
?	?	?

*



Outline

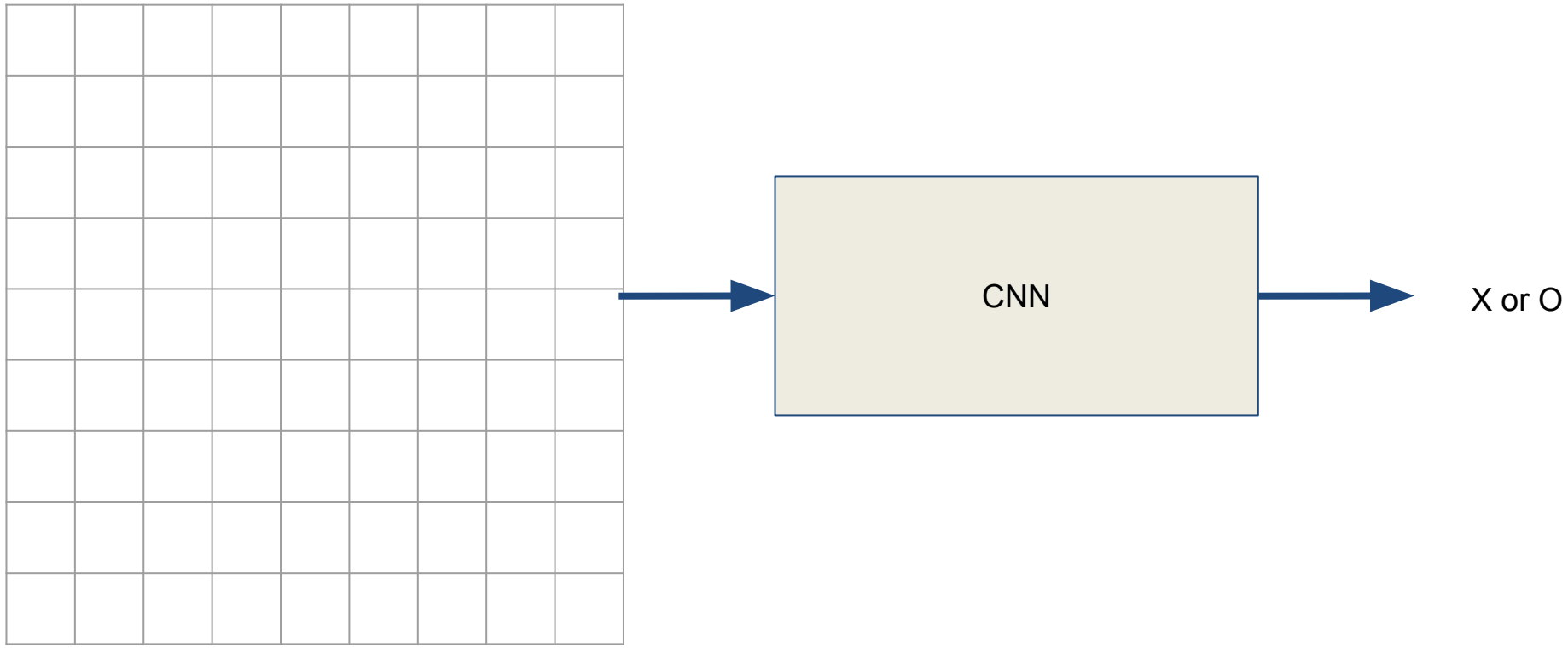
1. ANNs recap
2. DNNs – MNIST problem
3. DNNs - Key Concepts
4. CNNs – Motivation
5. CNNs – Convolution
6. **CNNs**
7. Transfer Learning

How do CNNs really work ?

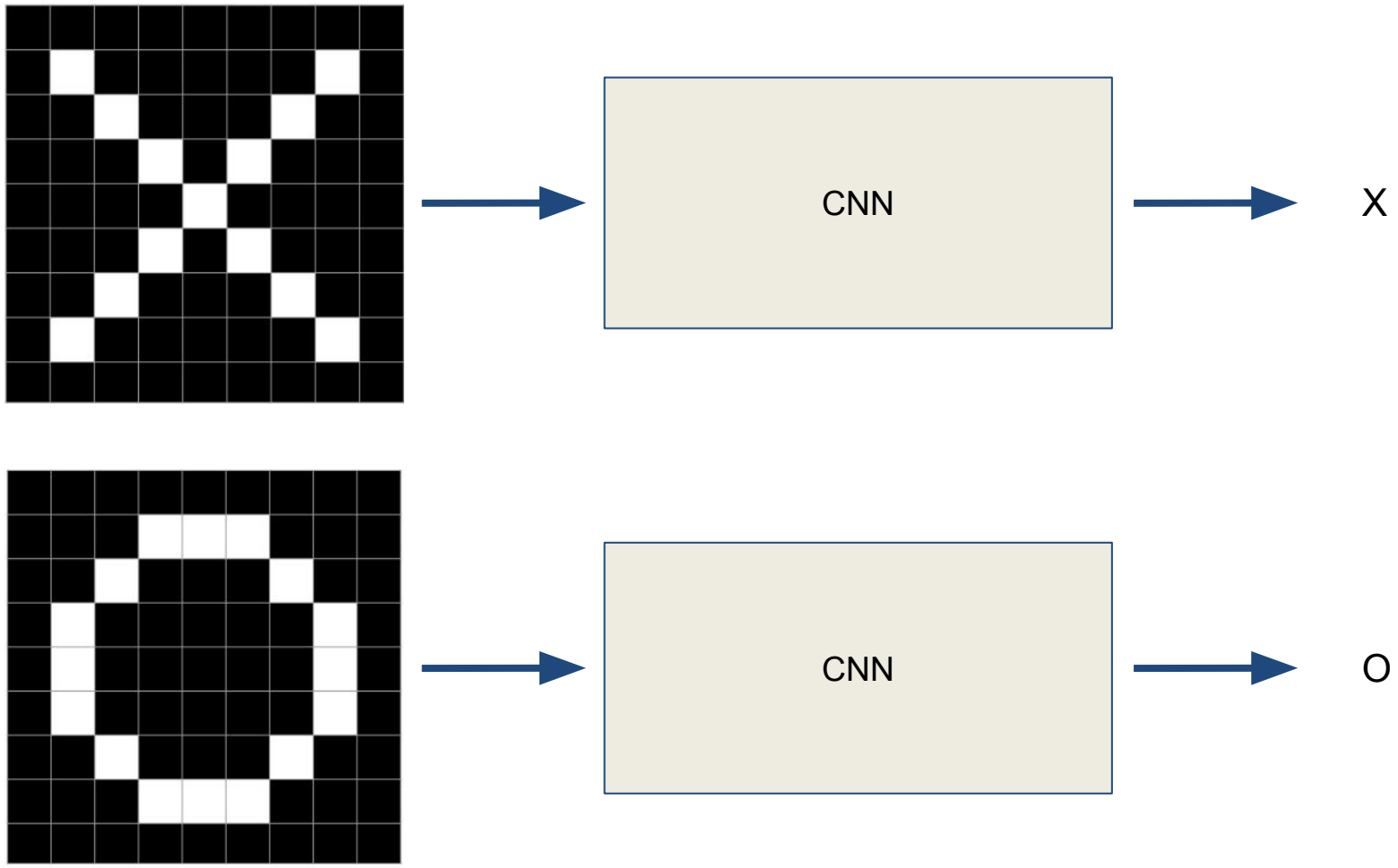
A toy ConvNet: Is it an X or an O?

Input: a two-dimensional array of pixels
- Black or White

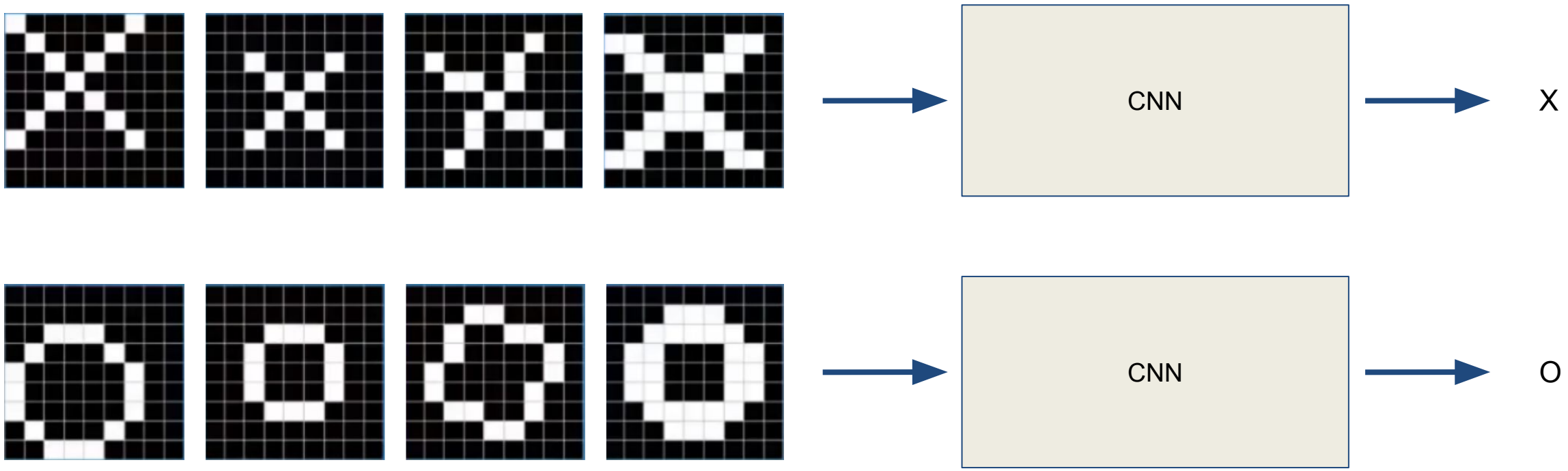
Output: two class classification task
- X or O



How do CNNs really work ?



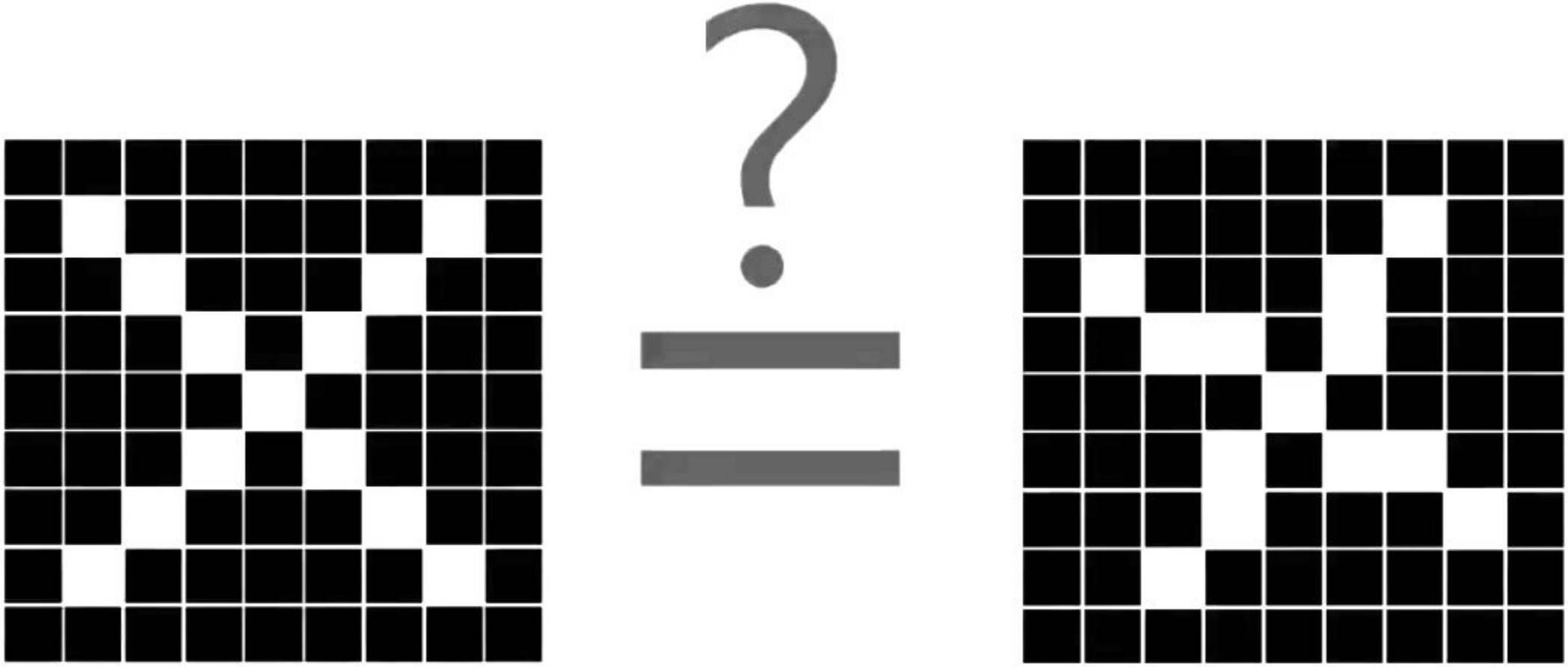
How do CNNs really work ?



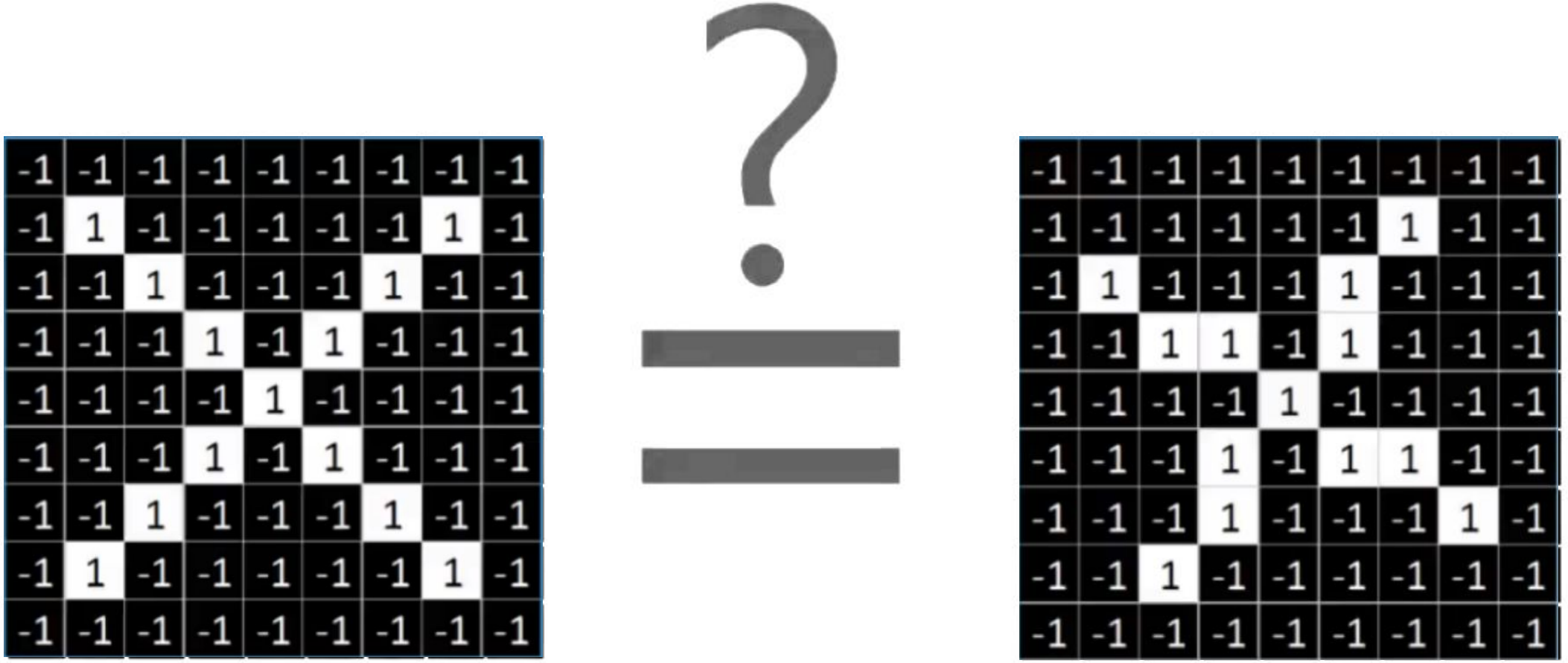
Trickier cases: solutions on images should be robust to:

- rotation, translation, scaling, weight, etc.

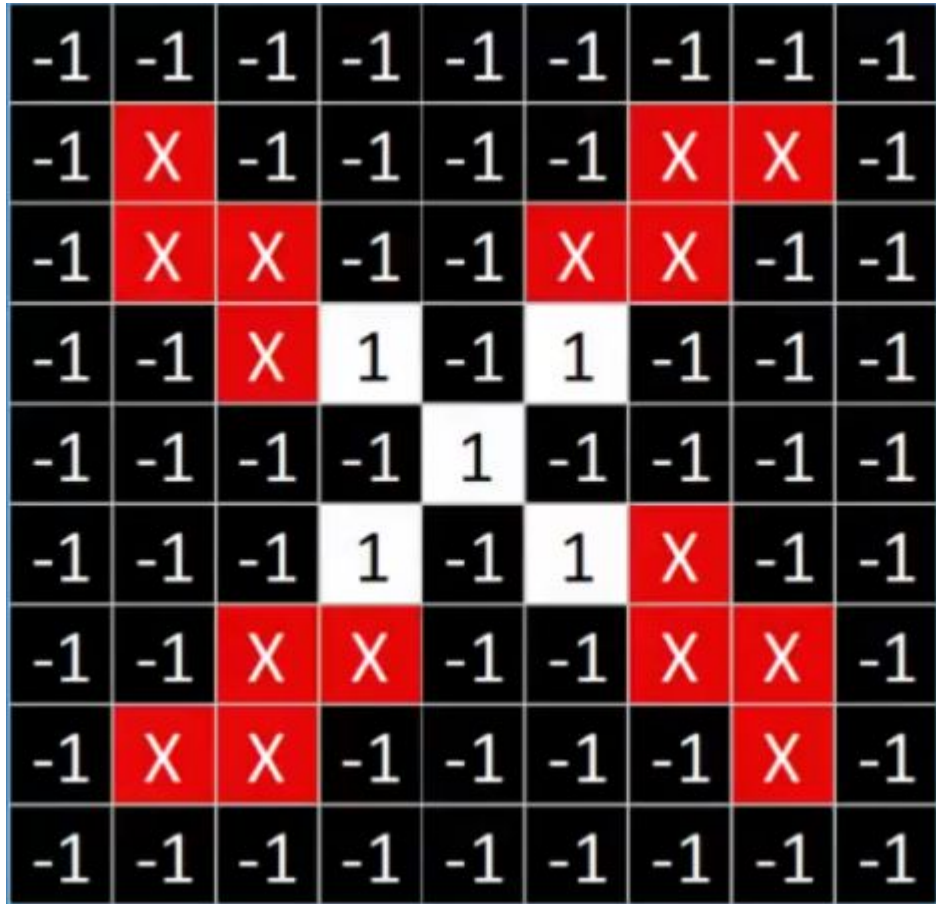
How do CNNs really work ?



How do CNNs really work ?



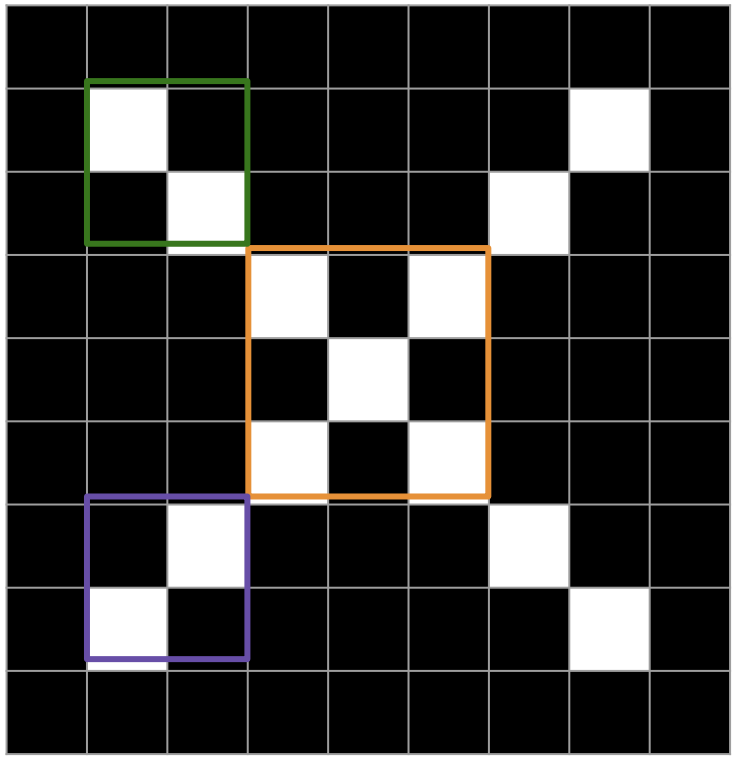
How do CNNs really work ?



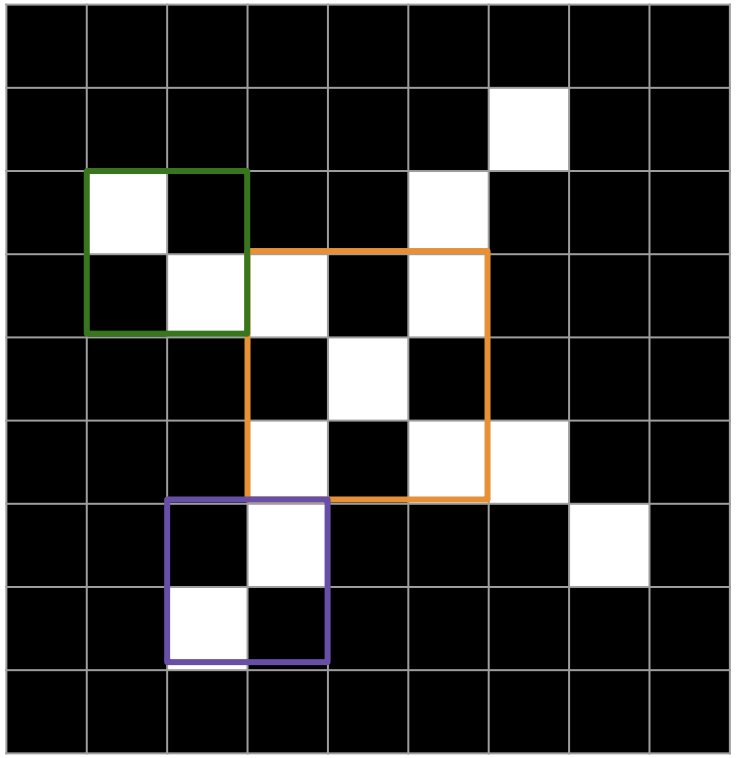
How do CNNs really work ?

Two main concepts:

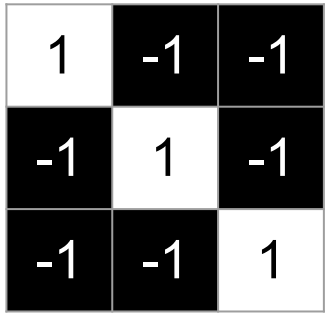
- Divide and conquer (again)
- From global to local solutions

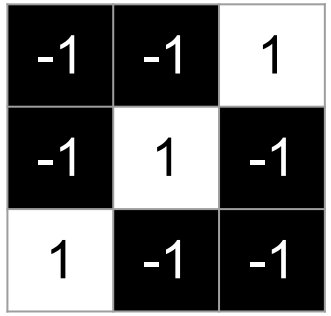


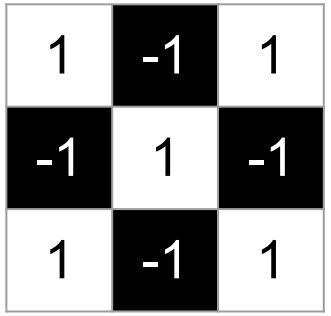
?



How do CNNs really work ?

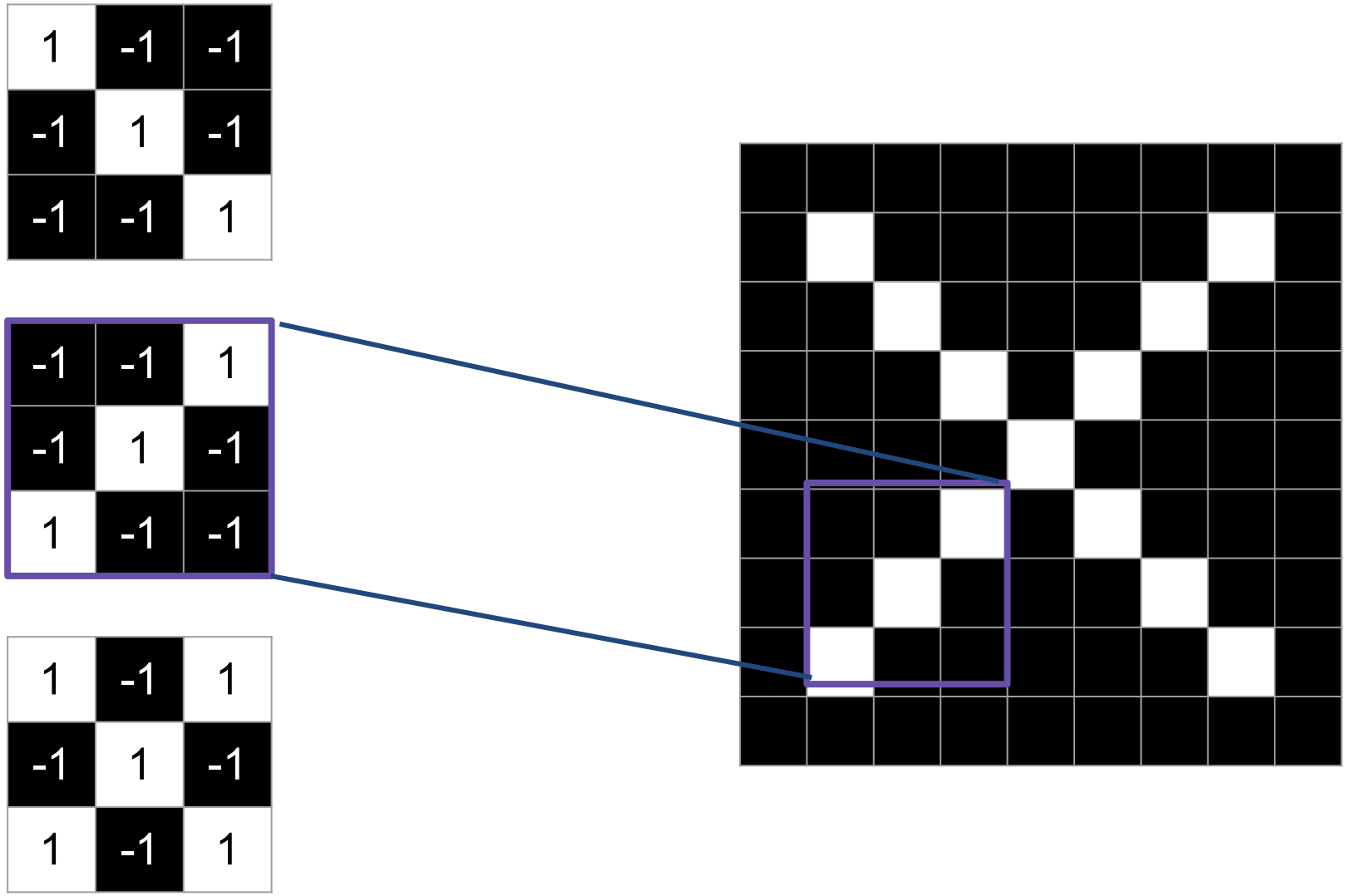






- Lets assume we have some characteristics or patterns that we can match with different parts of the image

How do CNNs really work ?



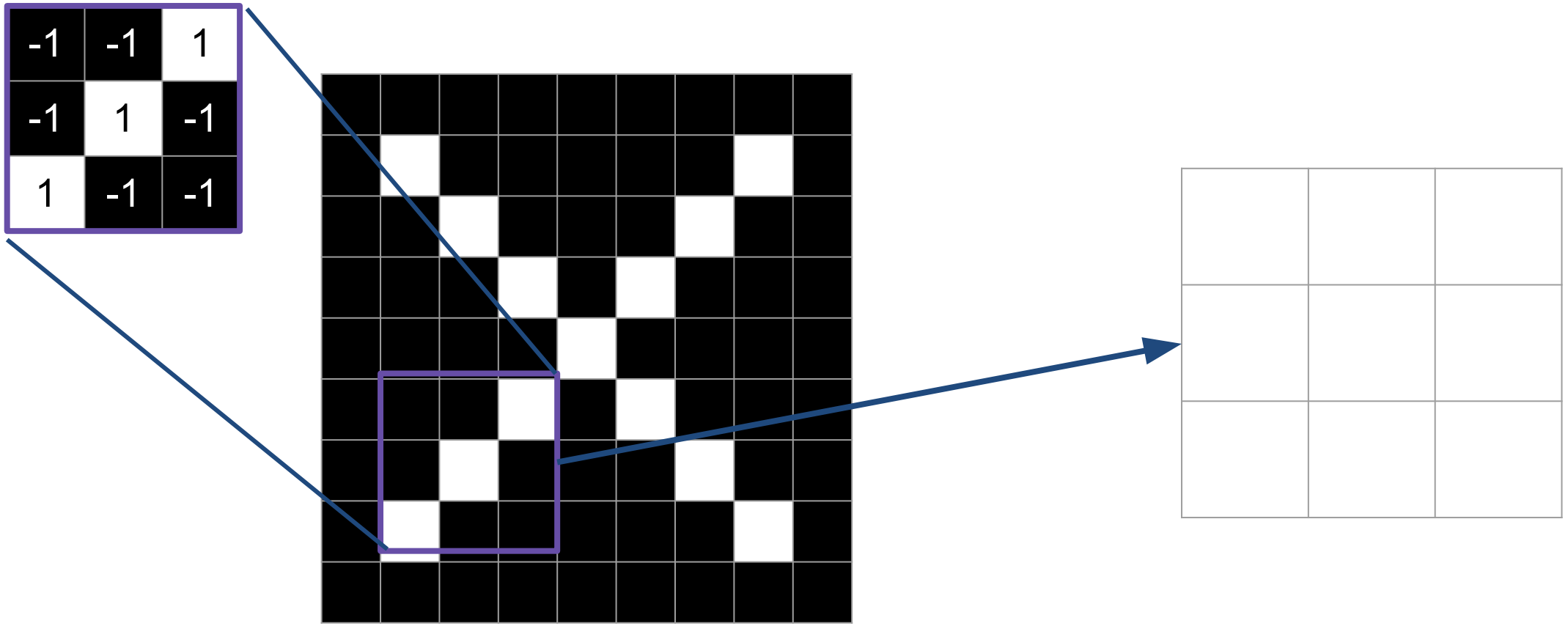
How do CNNs really work ?

Filtering: The idea that started everything

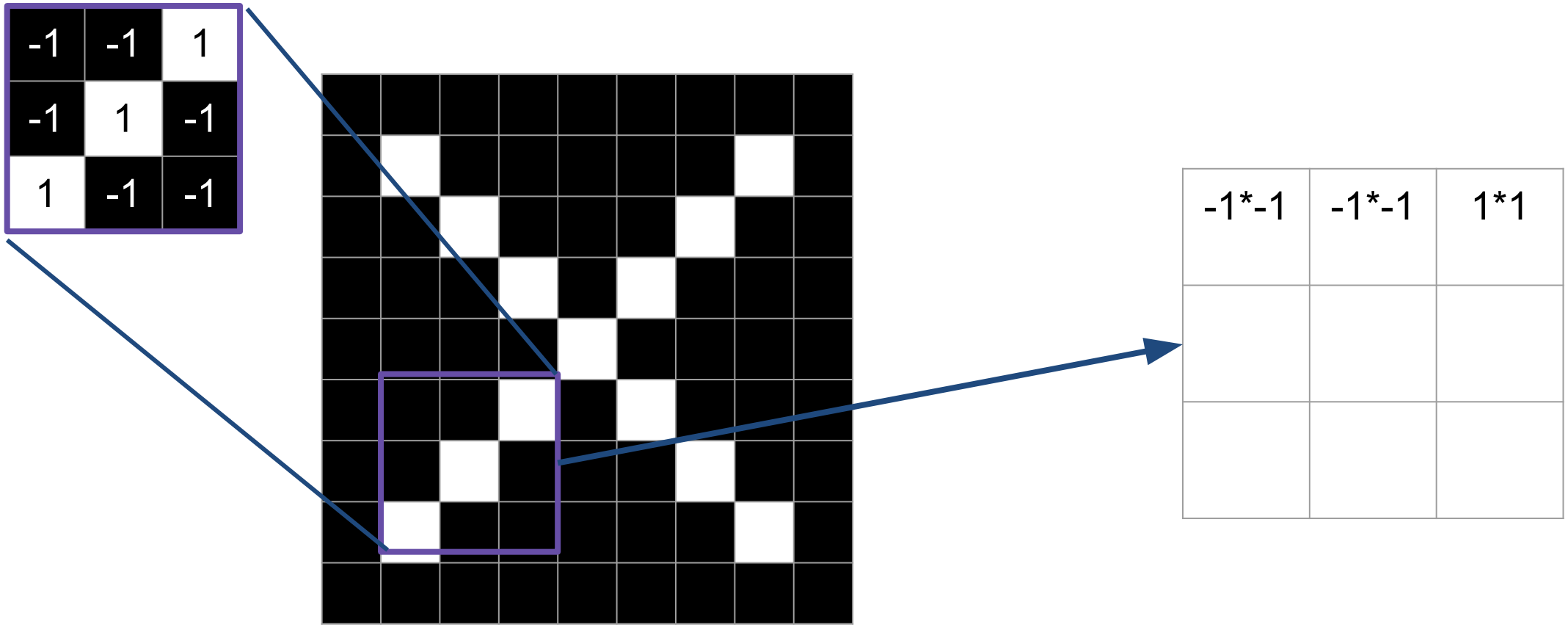
- The idea
  - We want a higher number when the pattern is closer to the part of the image we are comparing with (receptive field)

- The math
  - Line up the feature with some part of the image (receptive field)
  - Multiply each image pixel by the corresponding feature pixel
  - Add them up
  - Divide by the total number of pixels in the feature.
  - From global to local solutions

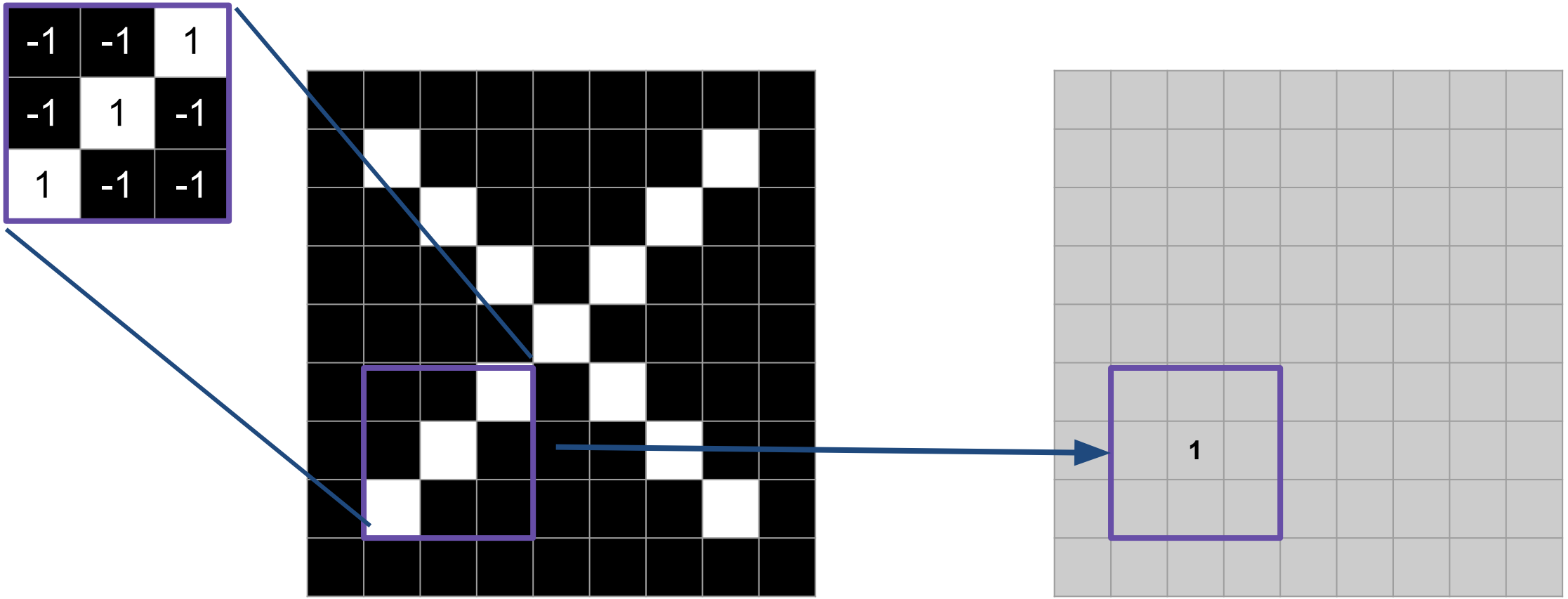
How do CNNs really work ?



How do CNNs really work ?

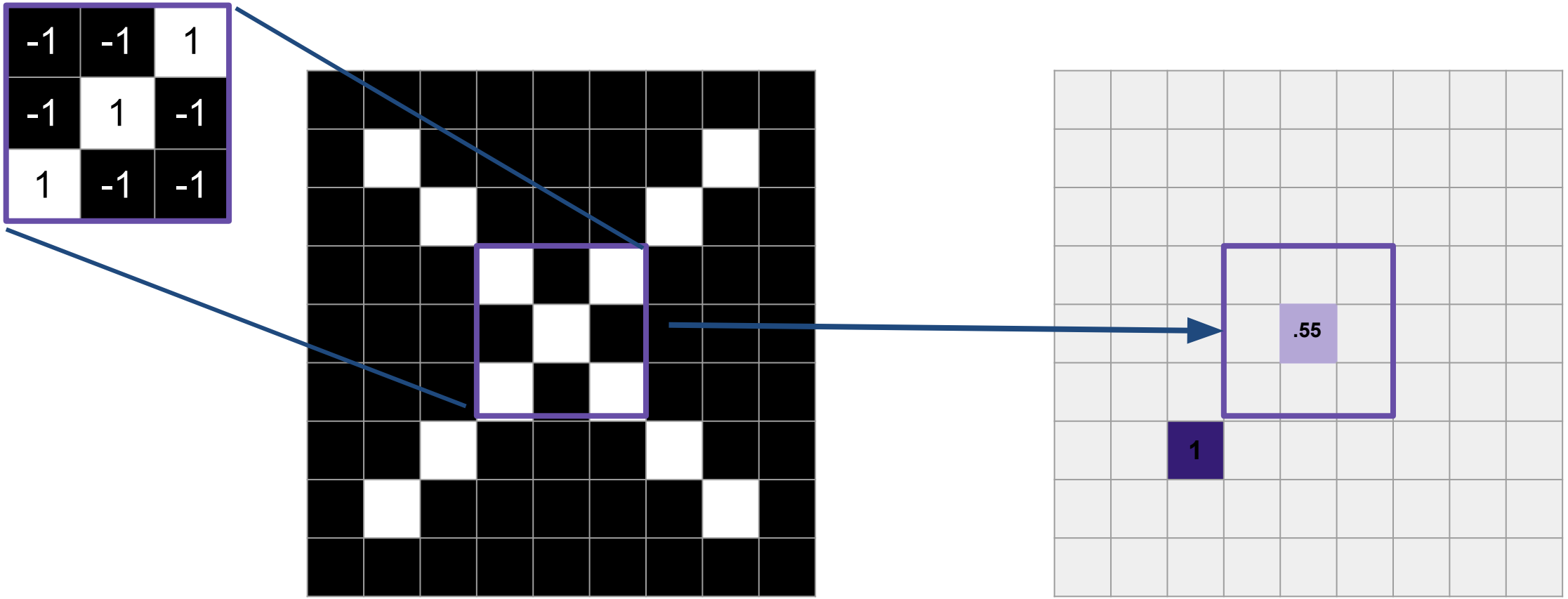


How do CNNs really work ?



(1+1+1+1+1+1+1+1+1) / 9 = 1

How do CNNs really work ?



(-1+1+1+1+1+1+1+1-1) / 9 = .55

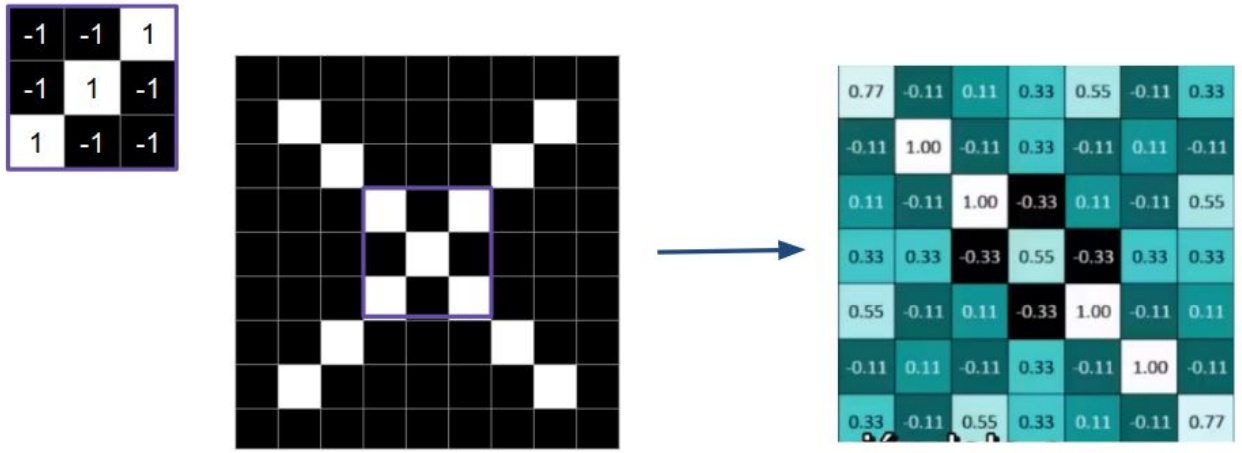
How do CNNs really work ?

Convolution:

- Traverse the image checking every possible receptive field

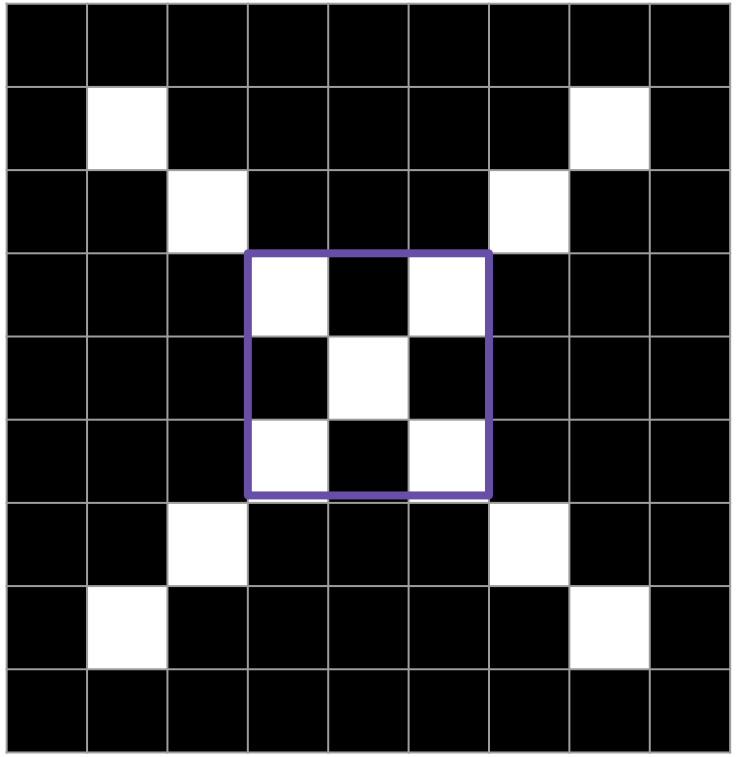
- Idea: Try all possible filtering

- Input: One image and one filter
- Output: A feature map
    - Another image that represents a map that tells us where in the image the pattern appears!



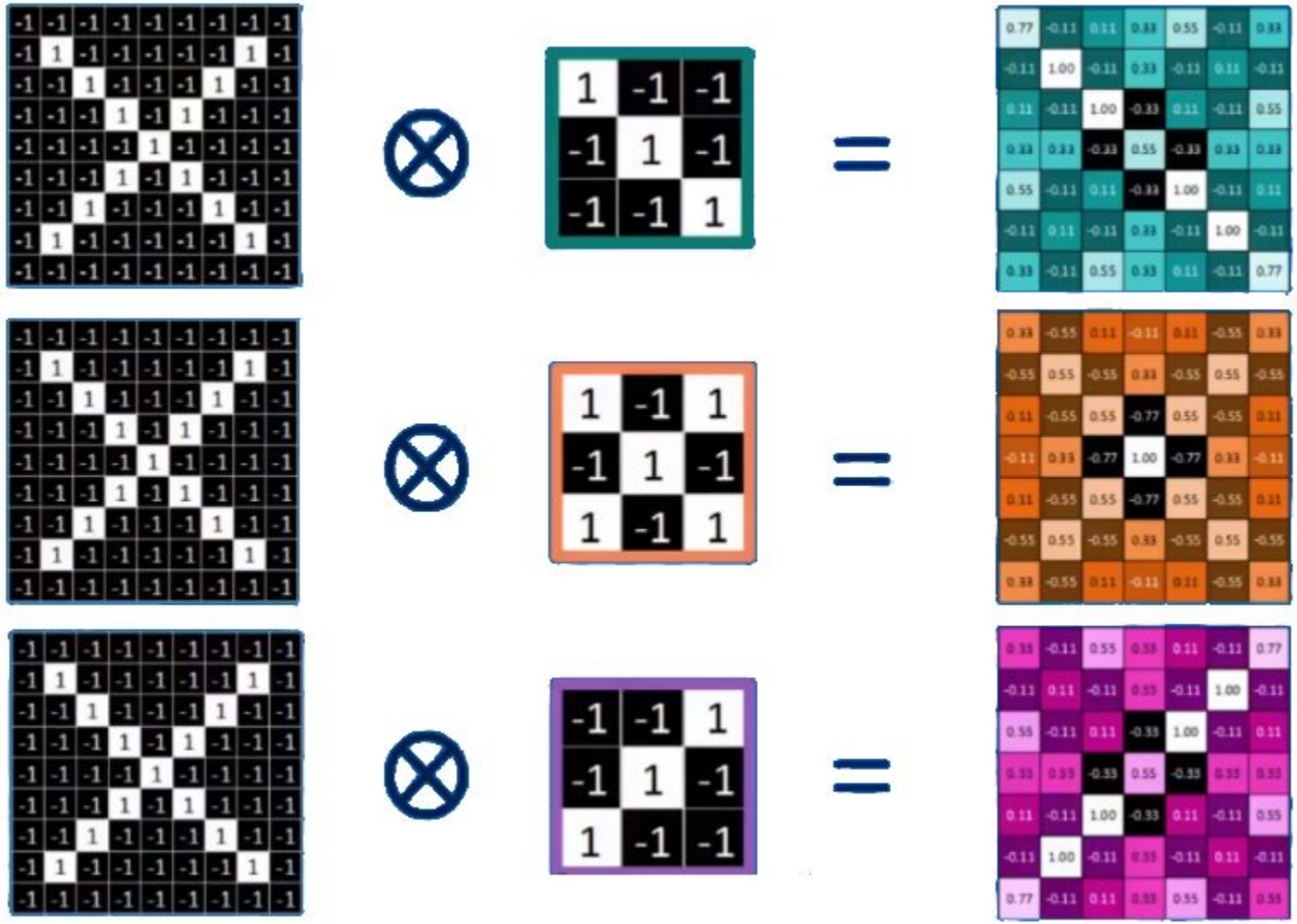
How do CNNs really work ?

-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

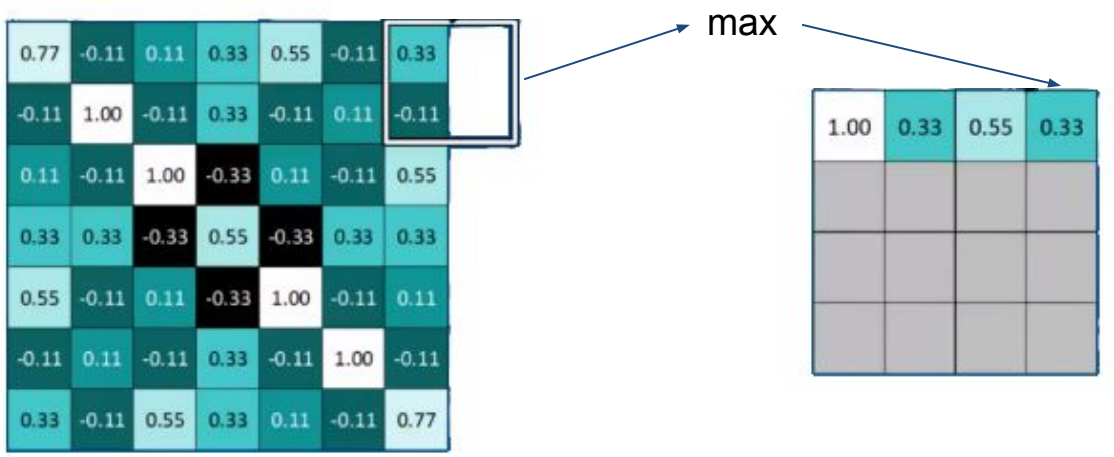
How do CNNs really work ?



How do CNNs really work ?

Pooling: Shrinking the image stack - Dimensionality Reduction

- Pick a window size (2 or 3)
- Walk your window across your feature maps
- From each window, take the maximum (min, mean) value.



How do CNNs really work ?

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

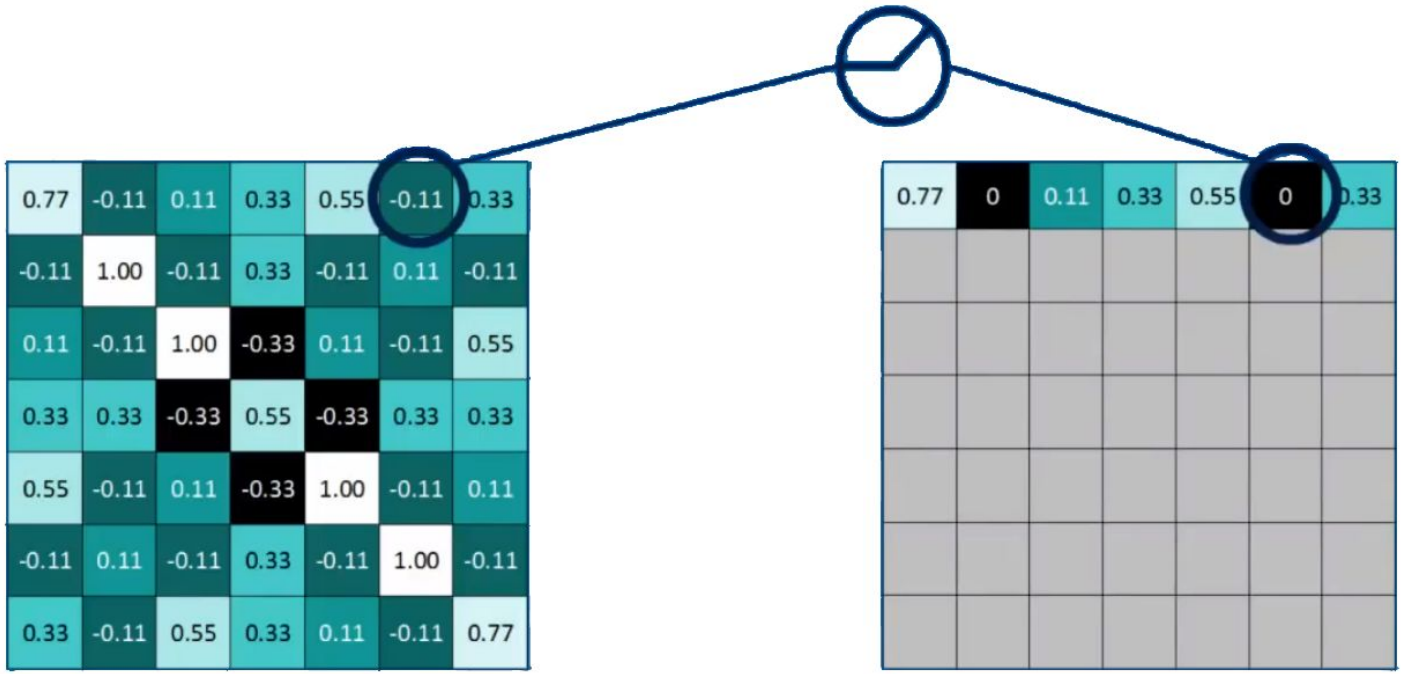
How do CNNs really work ?

- The Pooling layer **does NOT have trainable params**.
- Advantages:
  - **Pooling layers** reduce the size of the feature maps, which can **decrease the number of parameters in the model** and reduce the amount of computation required to apply the filters. This can make the model more efficient and faster to train.
  - Pooling layers can **make the model more tolerant to small variations** in the input data by downsampling the feature maps. This can make the model more robust and less prone to overfitting.
  - Pooling layers can **preserve the most important features in the data and discard the less important features**. This can help the model to focus on the most relevant patterns in the data and improve its performance.

How do CNNs really work ?

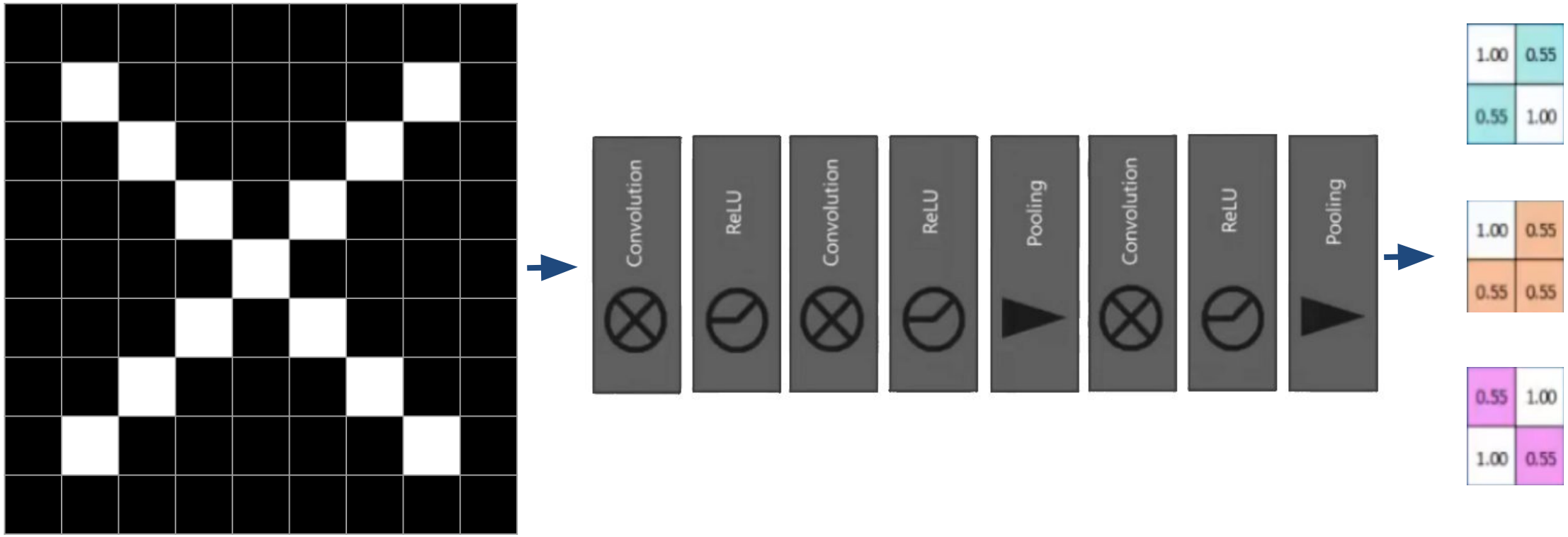
Normalization:

- We use an activation function to squeeze the output
- Same activation functions that are used in DNNs.



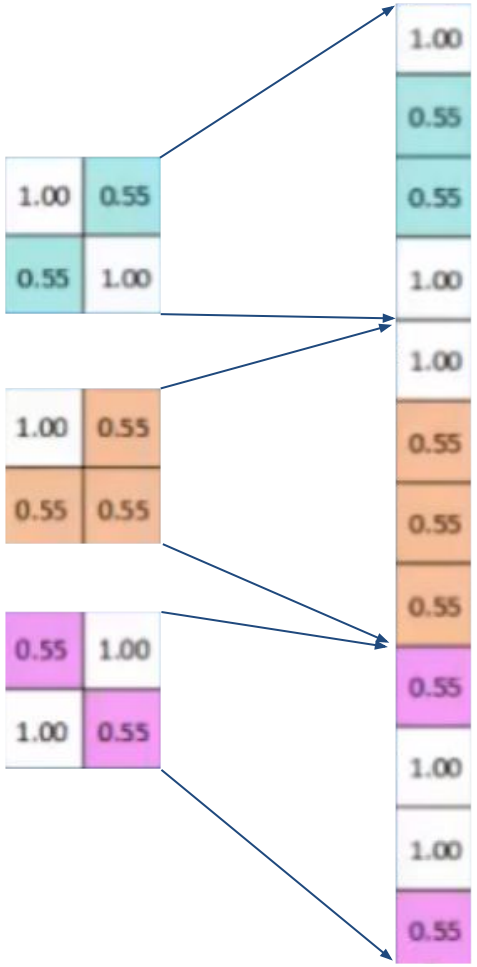
How do CNNs really work ?

Layers can (of course) be stacked!



How do CNNs really work ?

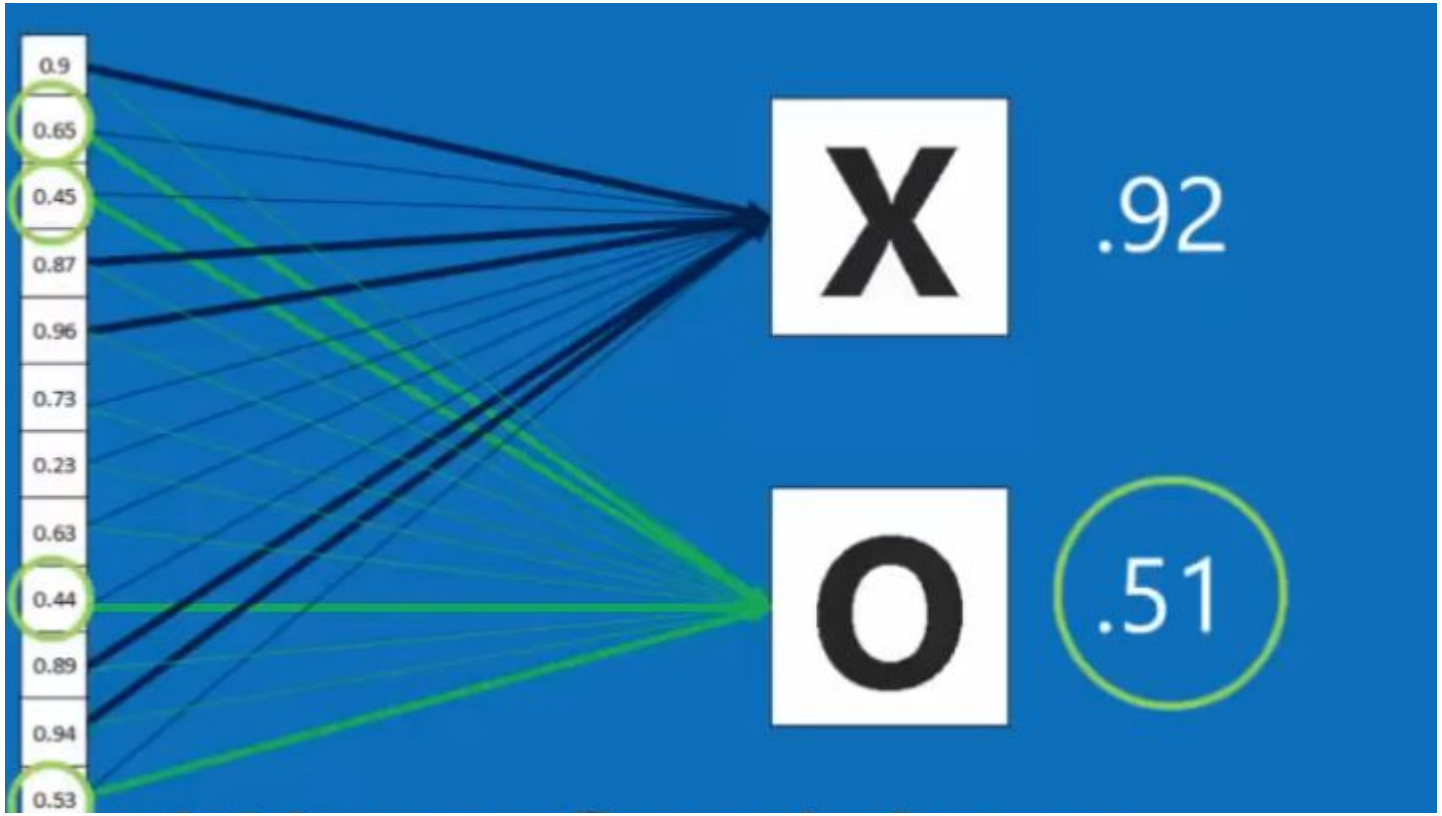
Flatten:



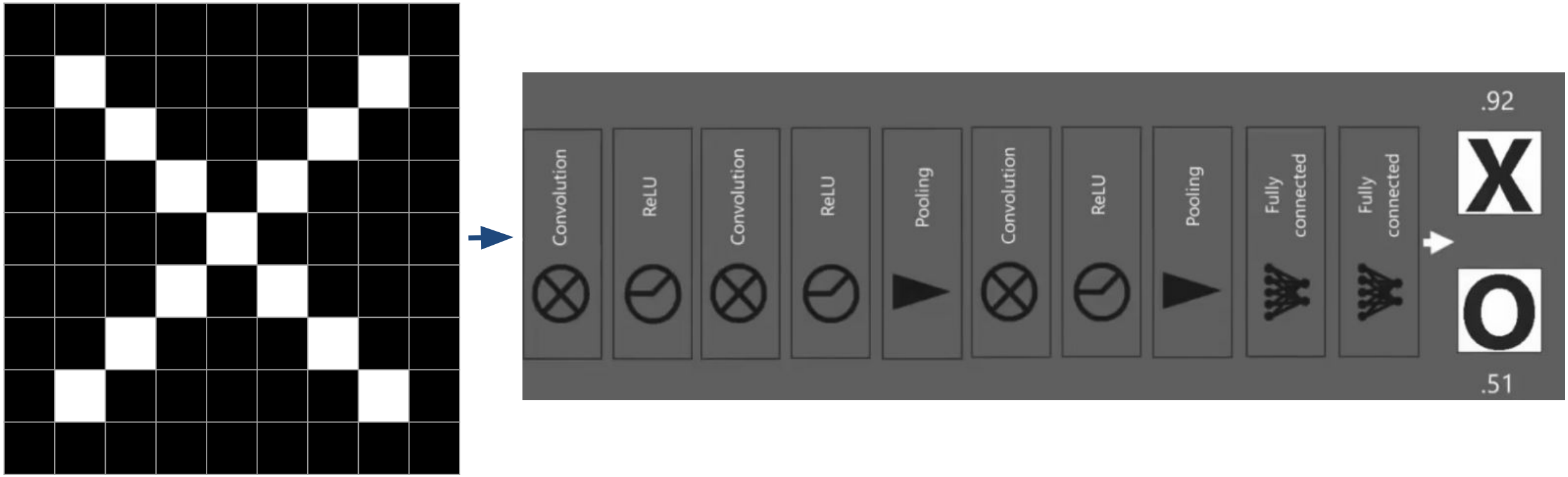
- Finally, visual characteristics are vectorized in order to take final decisions using a DNN layer(s)

- “Every value gets a vote”

How do CNNs really work ?



How do CNNs really work ?



How do CNNs really work ?

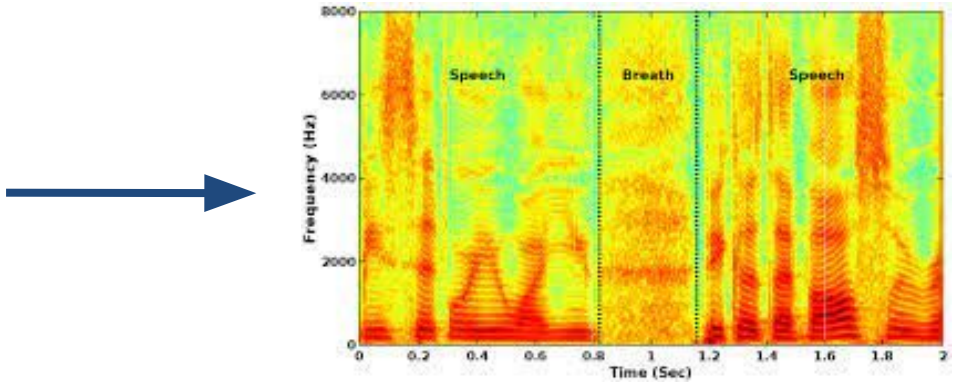
Training:

- Wait, that’s cool, but how do we decide which patterns ?
  - Once again, gradient descent (backpropagation)
- And what do we have to do then? Hyperparameters:
  - Convolution
    - Number of filters
    - Size of filters
  - Pooling
    - Window size
  - Fully Connected
    - Number of layers
    - Number of neurons

How do CNNs really work ?

And… can we use this somewhere else?

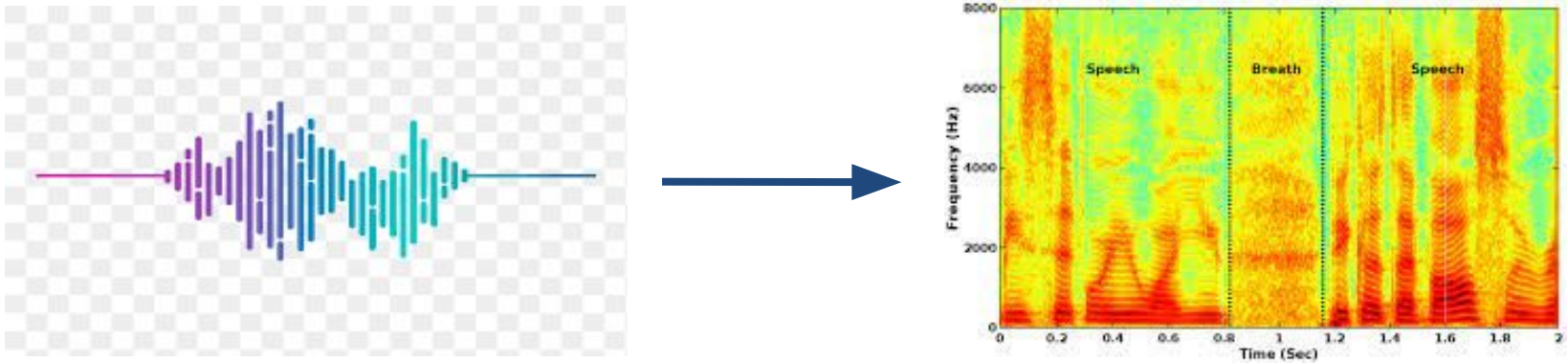
Yes! Any time the data has spatial information



How do CNNs really work ?

And… can we use this somewhere else?

Yes! Any time the data has spatial information



Rule of thumb: If you can swap around your input vectors… do NOT use a convolutional neural network :)

Stride & Padding

The stride (S) is the number of pixels that the convolutional filter moves right and down when it is applied to an input image.

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

Stride & Padding

The stride (S) is the number of pixels that the convolutional filter moves right and down when it is applied to an input image.

S = 1

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

Stride & Padding

The stride (S) is the number of pixels that the convolutional filter moves right and down when it is applied to an input image.

S = 1

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

Stride & Padding

The stride (S) is the number of pixels that the convolutional filter moves right and down when it is applied to an input image.

S = 2

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

Stride & Padding

The stride (S) is the number of pixels that the convolutional filter moves right and down when it is applied to an input image.

S = 2

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

*

1	0
1	-1

Stride & Padding

- The stride is a hyperparameter.
- A larger stride can reduce the size of the output feature maps, which can decrease the number of parameters in the model and reduce the amount of computation required to apply the filter.
- But a larger stride can also reduce the spatial resolution of the output feature maps, which can make it more difficult for the model to detect fine-grained patterns in the input data.

Stride & Padding

Padding is the number of rows and columns of zeros that are added to the input image before the convolutional filter is applied

5	2	3	4
6	7	8	5
4	5	4	5
1	0	2	3

Stride & Padding

Padding is the number of rows and columns of zeros that are added to the input image before the convolutional filter is applied

P = 1

0	**0**	**0**	**0**	**0**	**0**
0	5	2	3	4	**0**
0	6	7	8	5	**0**
0	4	5	4	5	**0**
0	1	0	2	3	**0**
0	**0**	**0**	**0**	**0**	**0**

Stride & Padding

Padding is the number of rows and columns of zeros that are added to the input image before the convolutional filter is applied

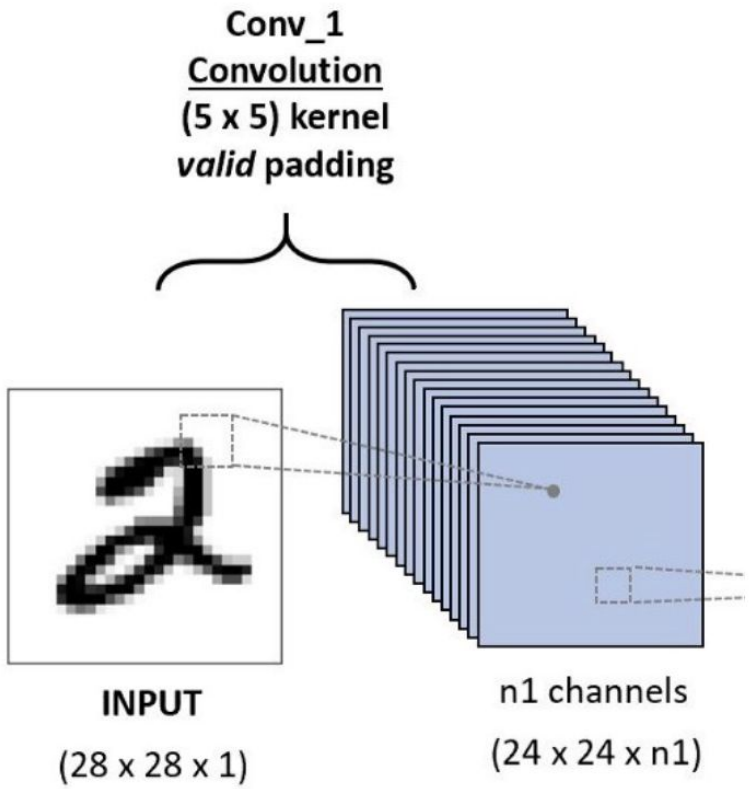
P = 2

0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	5	2	3	4	**0**	**0**
0	**0**	6	7	8	5	**0**	**0**
0	**0**	4	5	4	5	**0**	**0**
0	**0**	1	0	2	3	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Stride & Padding

- The padding is a hyperparameter.
- Padding can be used to:
  - Control the size of the output feature maps
  - Preserve the spatial dimensions of the input data
  - Prevent the loss of information at the borders of the input image
- Typical padding strategies in common frameworks (keras, torch) are:
  - Valid: Do not use padding
  - Same: Adds enough padding so that the output feature maps keep the same size as the input image.

The architecture: Convolutional Layer



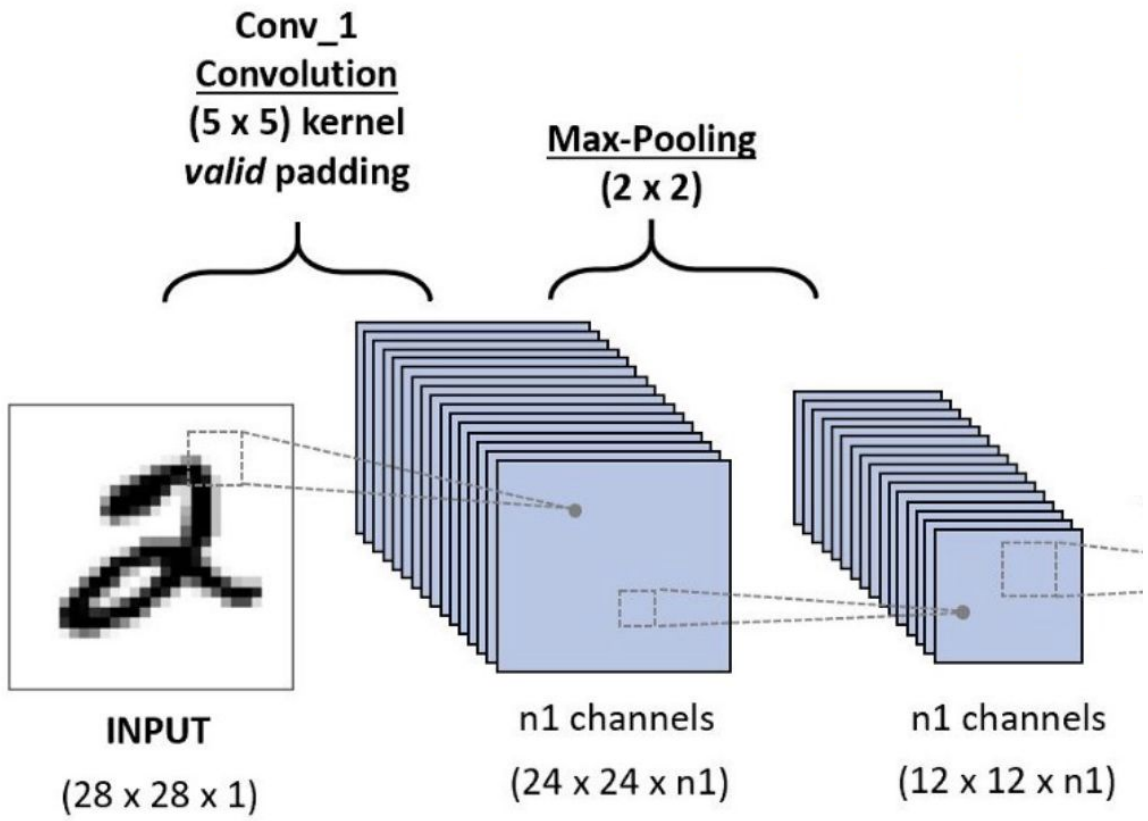
The main building block of a CNN is the convolutional layer.

- It is formed by a bunch (N) of filters of FxF size.
- The filters are applied (we convolve the image with each one) to the input, producing a set of outputs: N feature maps or channels.
- The size of the features maps is like the input size, according to the stride and padding.

$$N_h = (N_h + 2P - F)/S + 1$$
$$N_w = (N_w + 2P - F)/S + 1$$

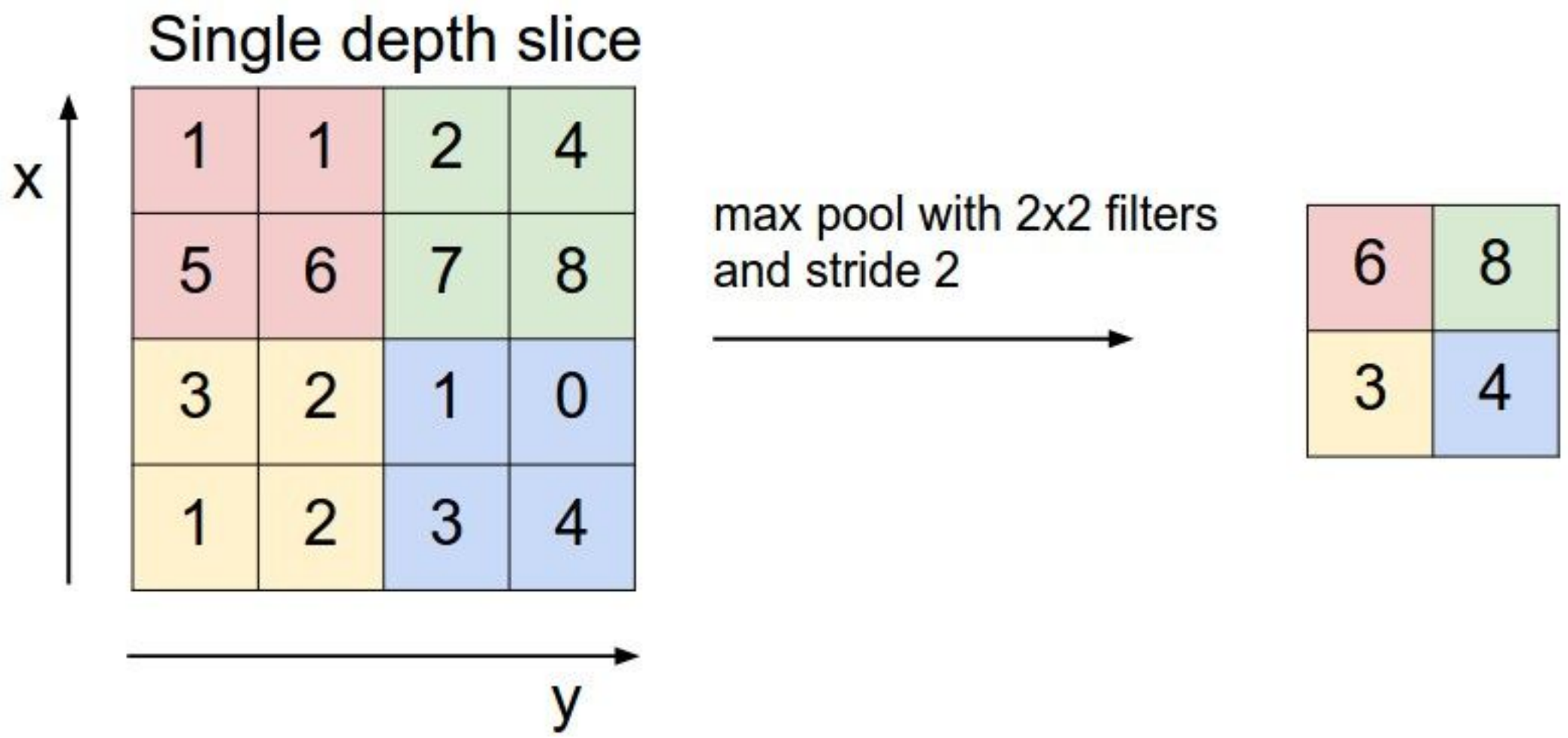
- After the convolution, we apply a non-linear function (e.g., ReLU) in an element-wise manner to the outputs and we add the bias (there is a bias per filter)

The architecture: Pooling Layer



- After the conv layer, a pooling layer is often used
- The pooling layer just downsamples the outputs by taking the max (max pooling layer) or the average of groups of ‘pixels’ following the same way than in a convolution

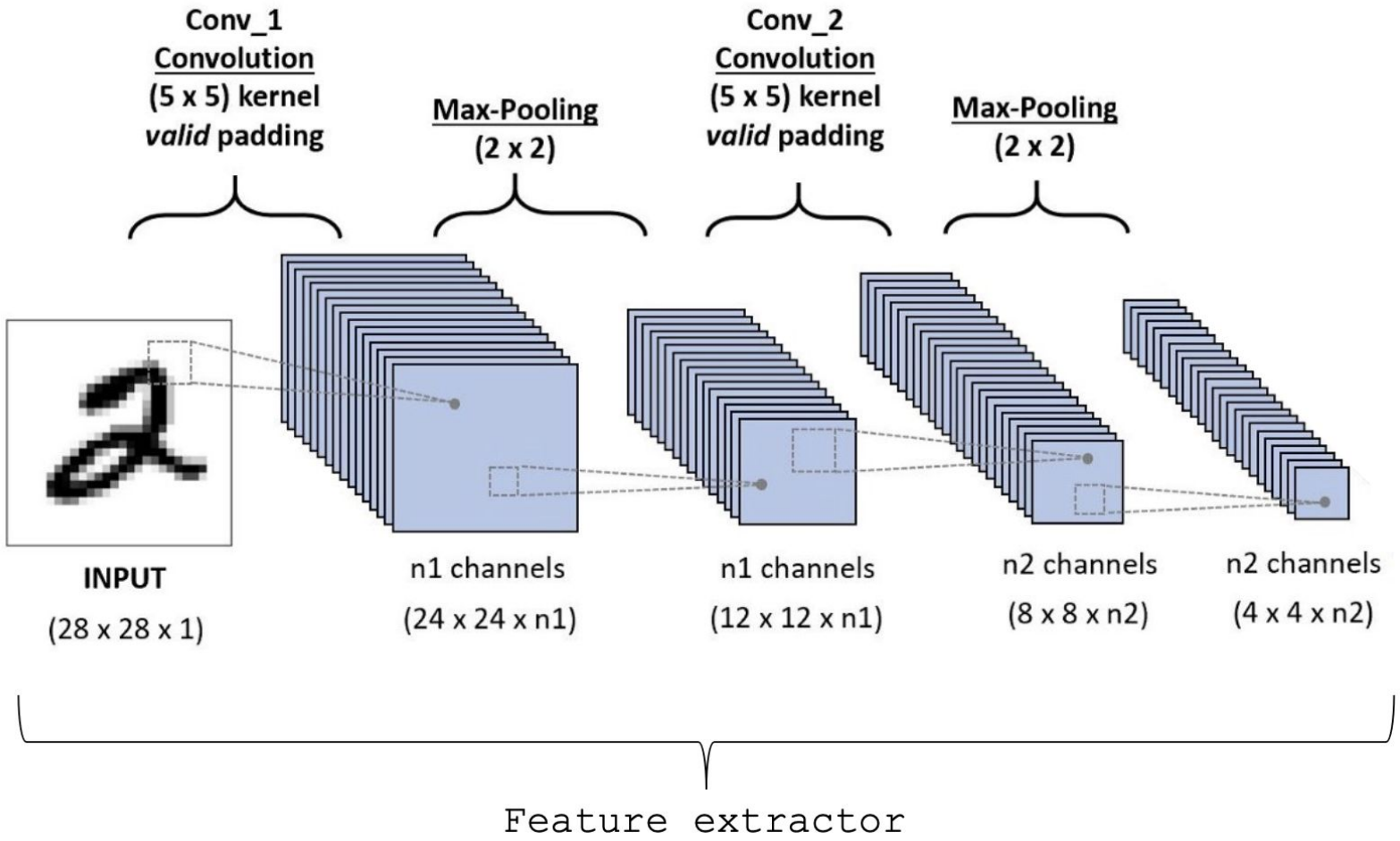
The architecture: Pooling Layer



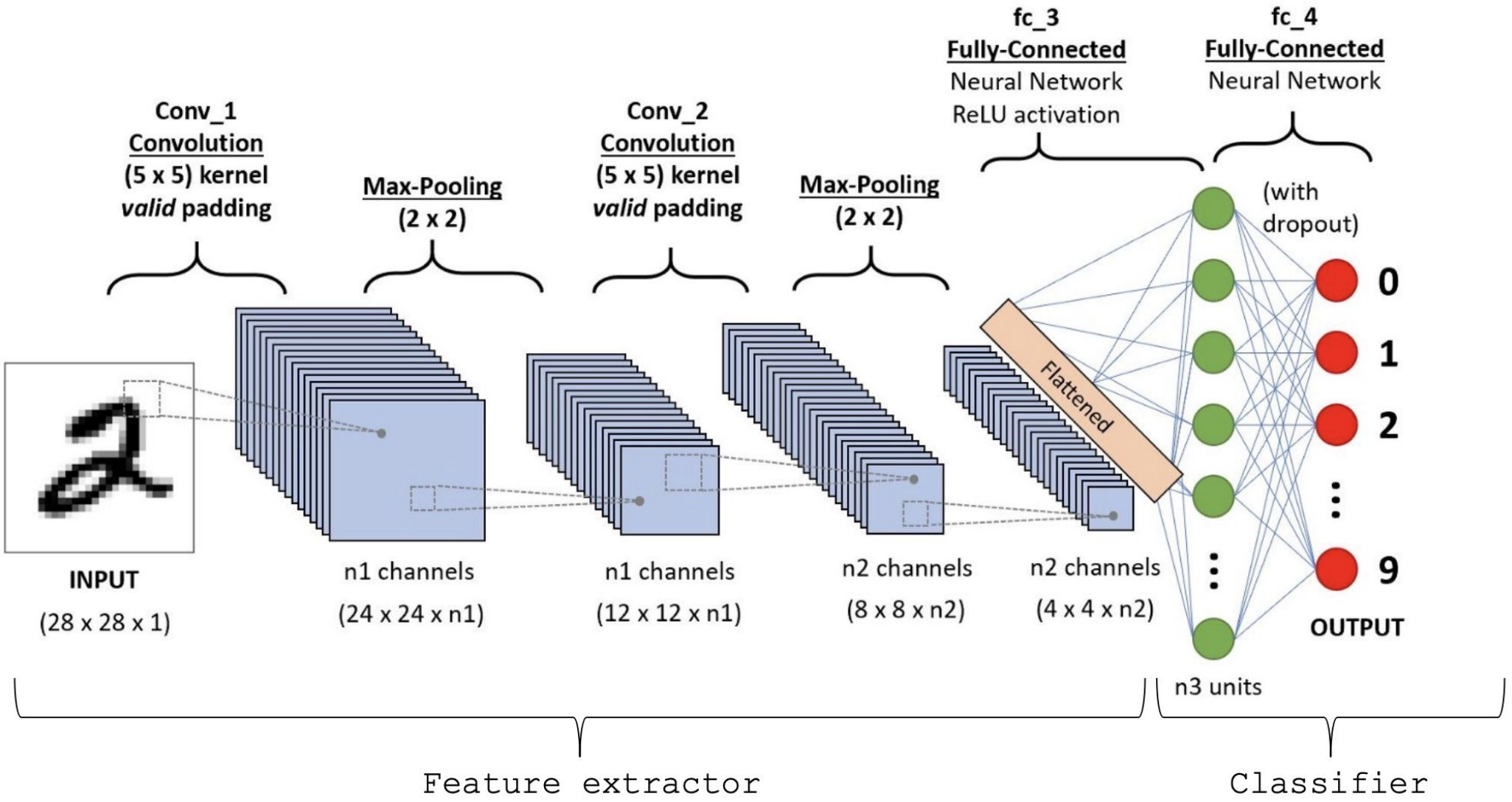
The architecture: Pooling Layer

- The Pooling layer **does NOT have trainable params**.
- Advantages:
  - **Pooling layers** reduce the size of the feature maps, which can **decrease the number of parameters in the model** and reduce the amount of computation required to apply the filters. This can make the model more efficient and faster to train.
  - Pooling layers can **make the model more tolerant to small variations** in the input data by downsampling the feature maps. This can make the model more robust and less prone to overfitting.
  - Pooling layers can **preserve the most important features in the data and discard the less important features**. This can help the model to focus on the most relevant patterns in the data and improve its performance.

The architecture: Feature extractor



The architecture: Classifier



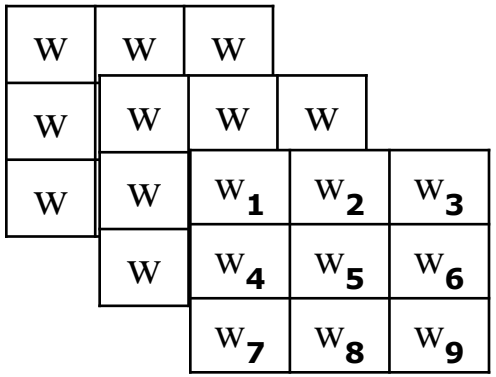
Key features of CNNs

- **Parameter sharing.** Filters are applied to the input image no matter its size, using the same set of trainable params -- those which form the filter itself.
- **Space Invariance.** CNNs can learn translation-invariant features, which means that they can recognize patterns in an image regardless of their position in the image – the filters are moved across the entire image to detect patterns.
- As DNNS, **CNNs can learn hierarchical features**, which means that they can learn increasingly complex patterns by learning simple patterns first and then building on them.
- CNNs can learn features that are **robust to noise**, occlusion, and other image variations, which makes them resistant to overfitting and enables them to generalize well to unseen data. This is thanks to the pooling layers which downsample the input data making the model more tolerant to small variations.

CNNs in Keras

How to build a convolutional layer.

- Decide the number of neurons (filters) of the layer.
- Decide the shape of every filter.
- Every component of a filter is now a weight/free parameter that we will learn.



CNNs in Keras

- **Convolutional layers in Keras/Tensorflow**

https://keras.io/api/layers/convolution_layers/convolution2d/

Conv2D layer

Conv2D class [source]

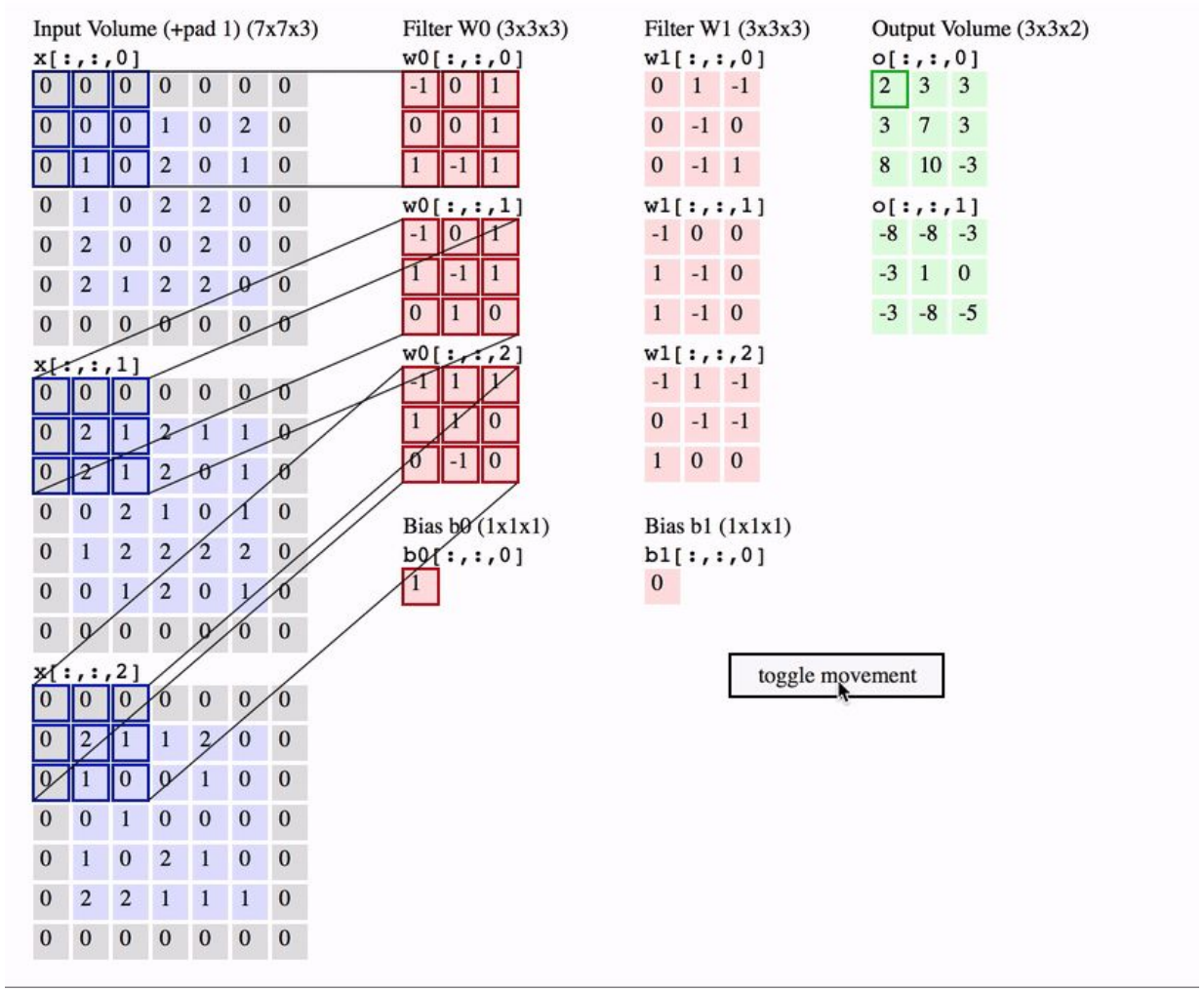
```
tf.keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding="valid",
    data_format=None,
    dilation_rate=(1, 1),
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

CNNs: More complex! Color images

R

G

B



Output dimensions:

$N_h = (N_h + 2P - F)/S + 1$
$N_w = (N_w + 2P - F)/S + 1$
$N_c = \#filters$

S = Stride □ 2
P = Padding □ 1
F = Size of the filter □ 3