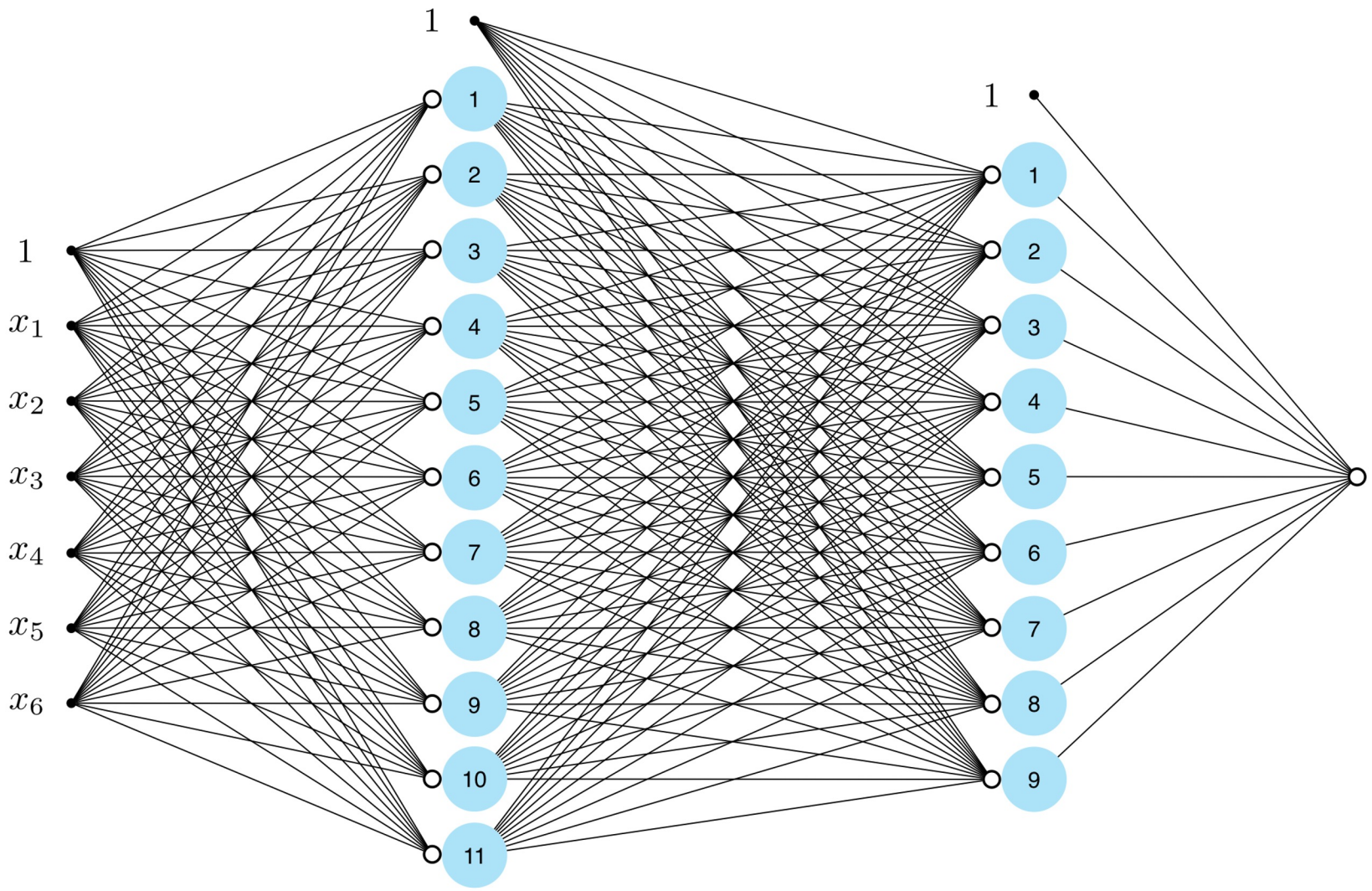
Convolutional Neural Networks

Tony T. Luo

July 2023

Multi-Layer Perceptron:
fully connected neural network

A two-layer MLP network



Python implementation (using Numpy)

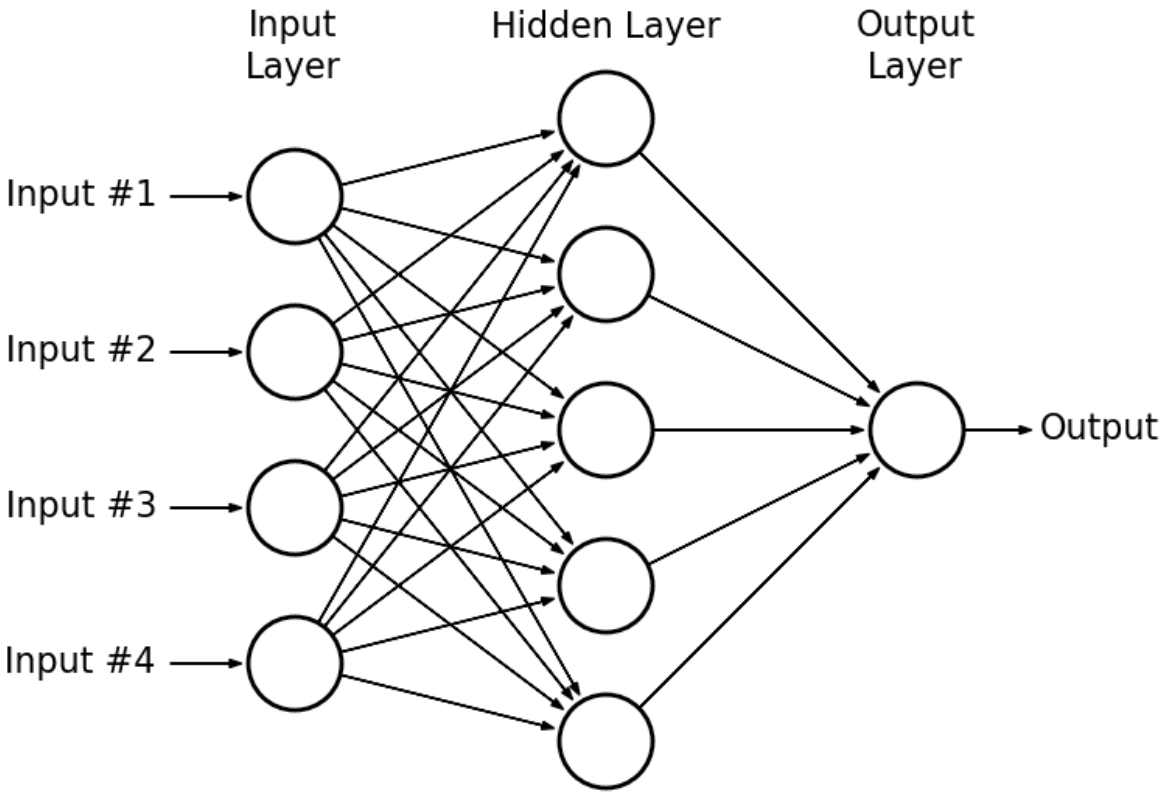
```
# neural network feature transformation
def feature_transforms(a, w):

    # loop through each layer
    for W in w:

        # compute inner-product with current layer weights
        a = W[0] + np.dot(a.T, W[1:])

        # pass through activation
        a = activation(a).T

    return a
```

- Notice bias and activation function

Convolutional Neural Networks

Why CNN (why not MLP)?

Problems with MLP (all FC layers)

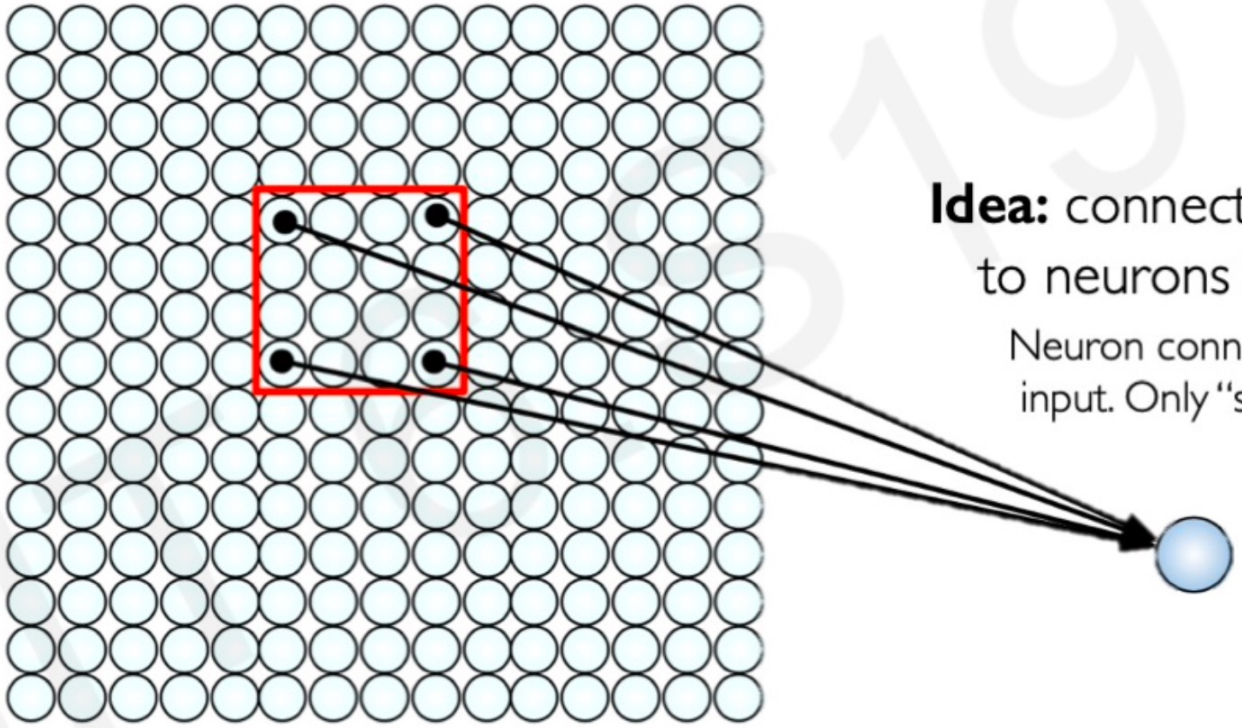


- Too many parameters
  - U1 * U2 * U3 …
- Does not preserve spatial information
  - Every neuron sees the same inputs!

Preserving spatial structure

Each neuron only sees a “patch”

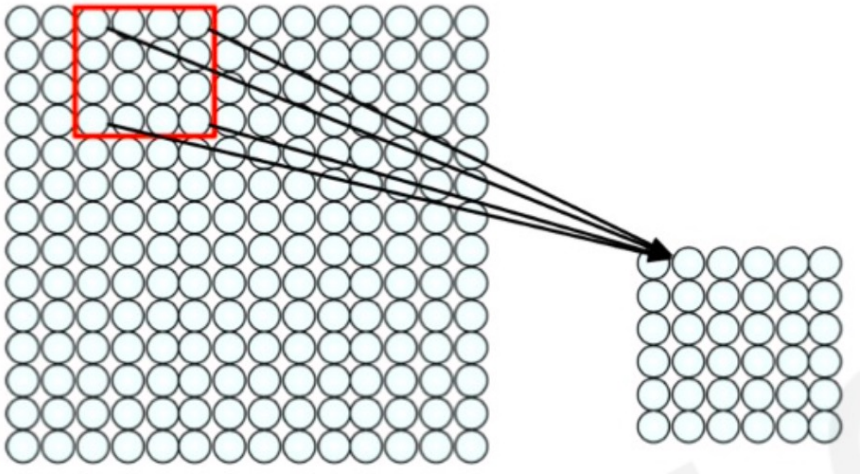
Input: 2D image.
Array of pixel values



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

Neuron connected to region of input. Only “sees” these values.

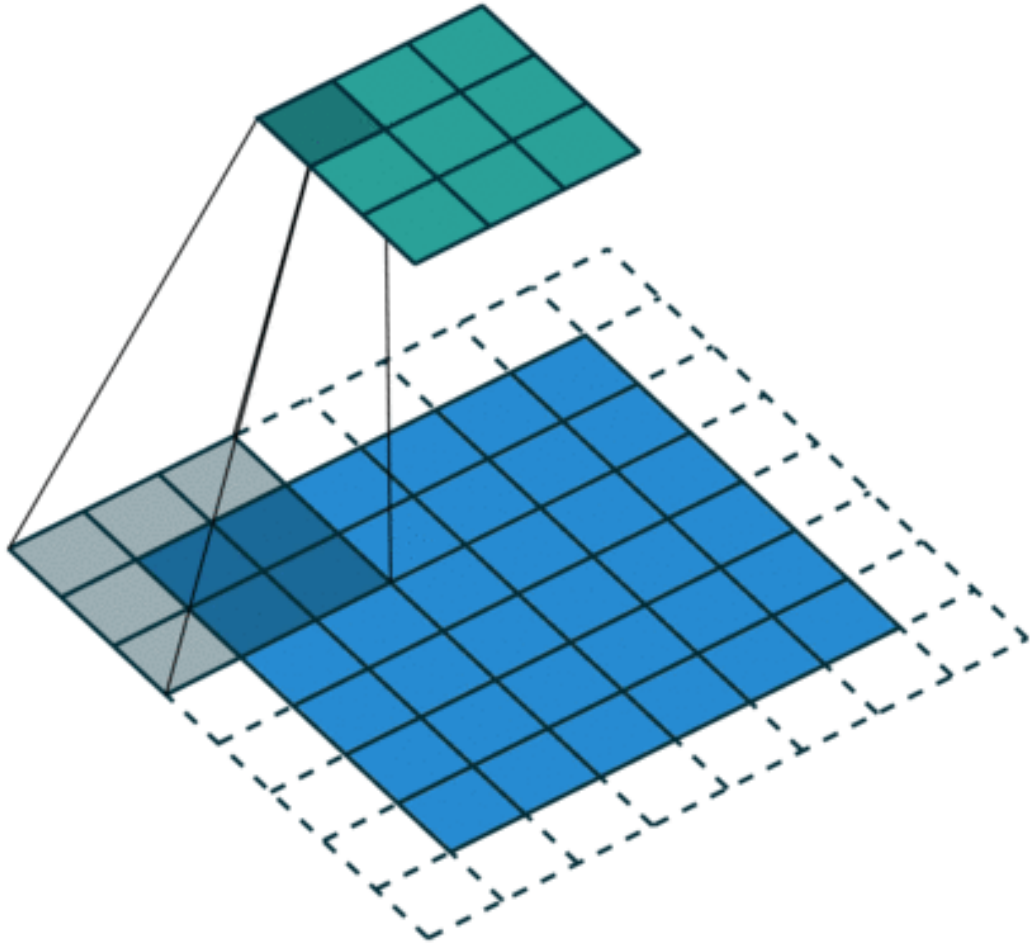
How to map a patch to a single neuron?

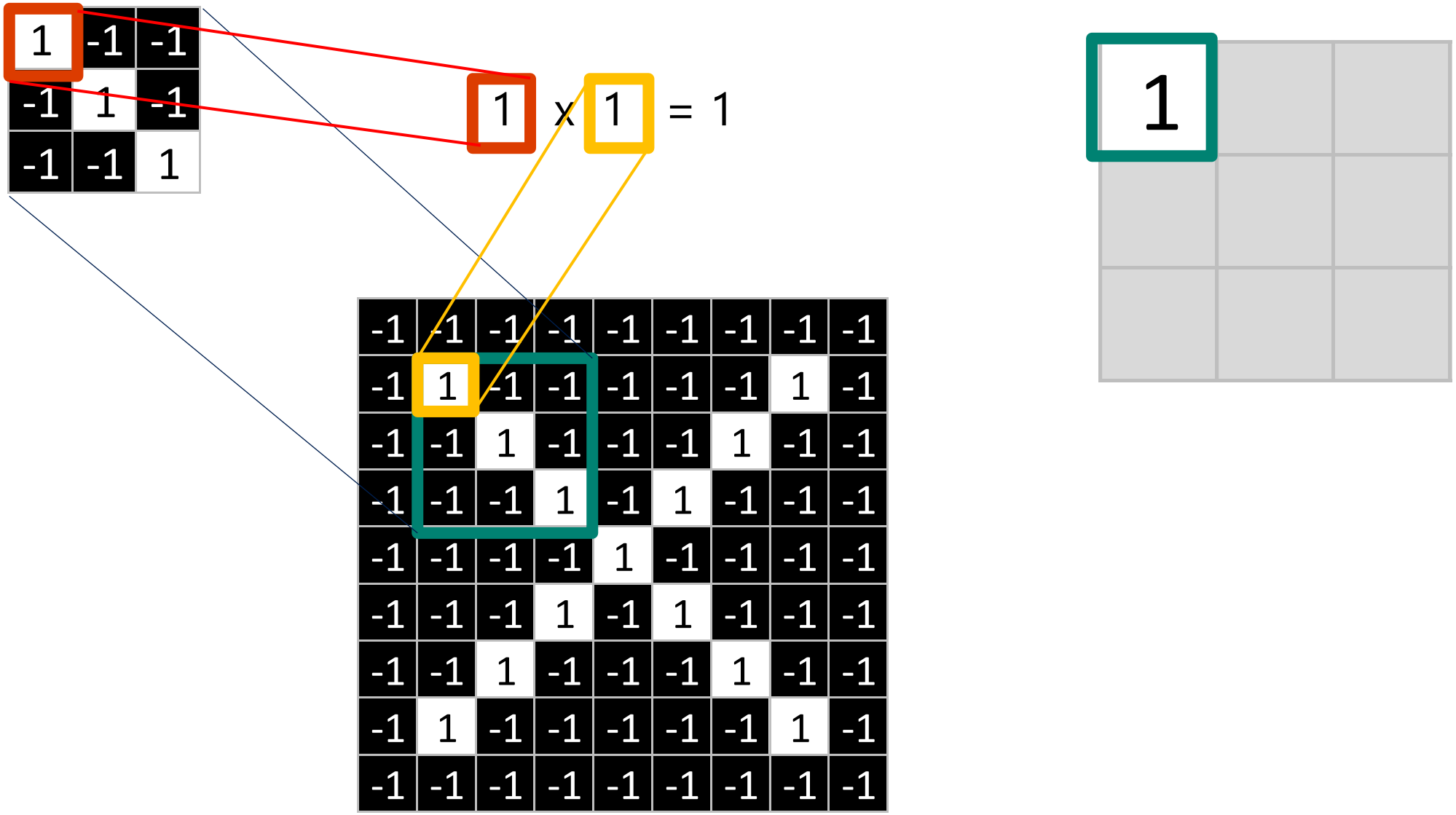


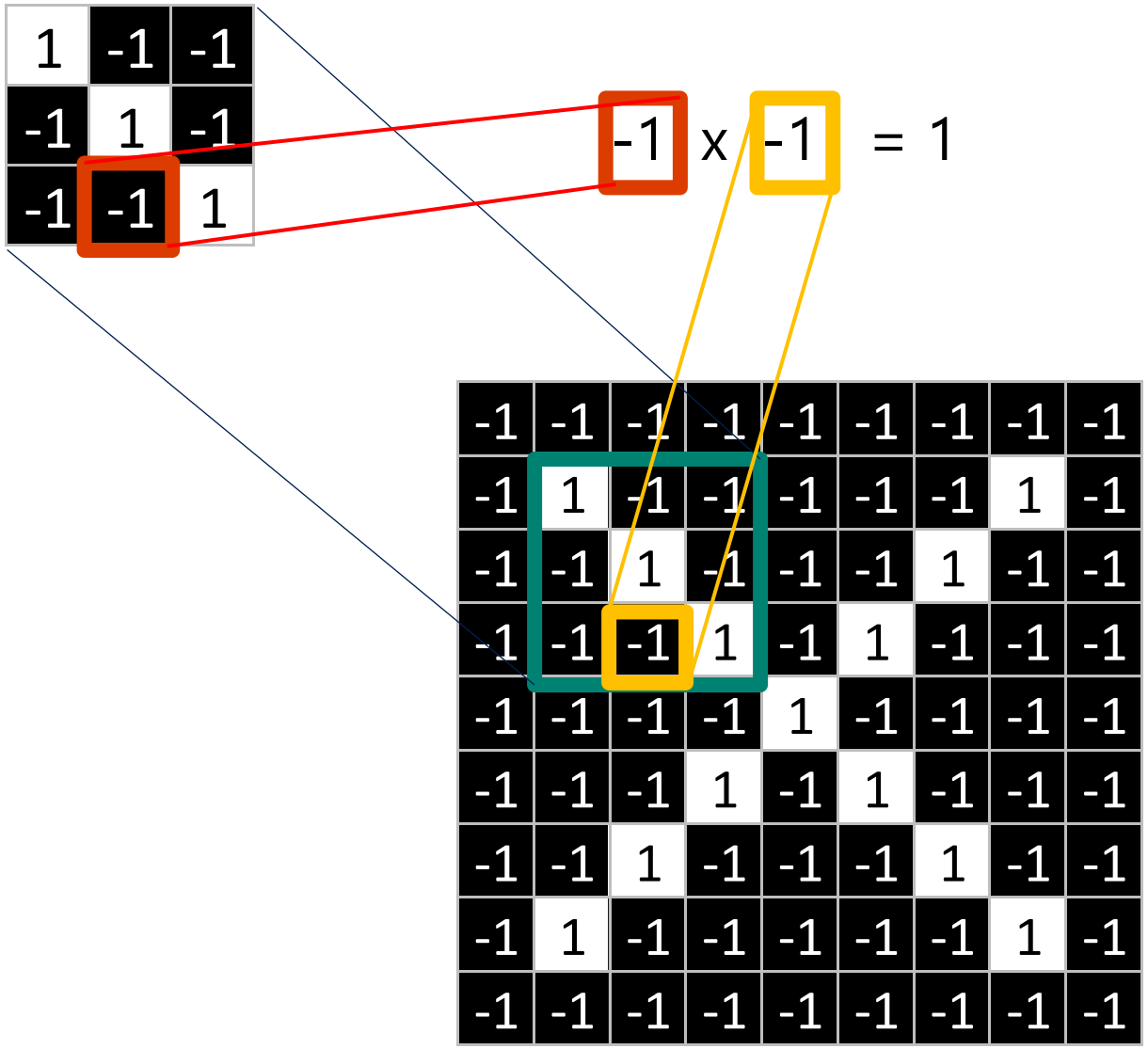
- Basic idea: weighted sum
- Weights are called a “filter” or “kernel”
- patch <dot> filter = output hidden neuron
  - Element-wise multiply
- This operation is called “convolution”

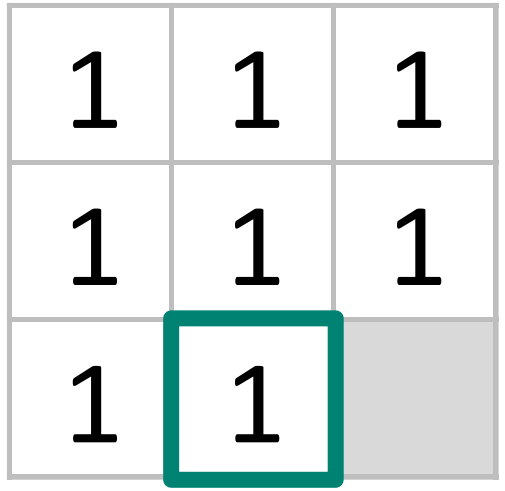
- Each filter corresponds to a certain “feature” of input
- Use multiple filters to extract multiple features

Convolution Operation



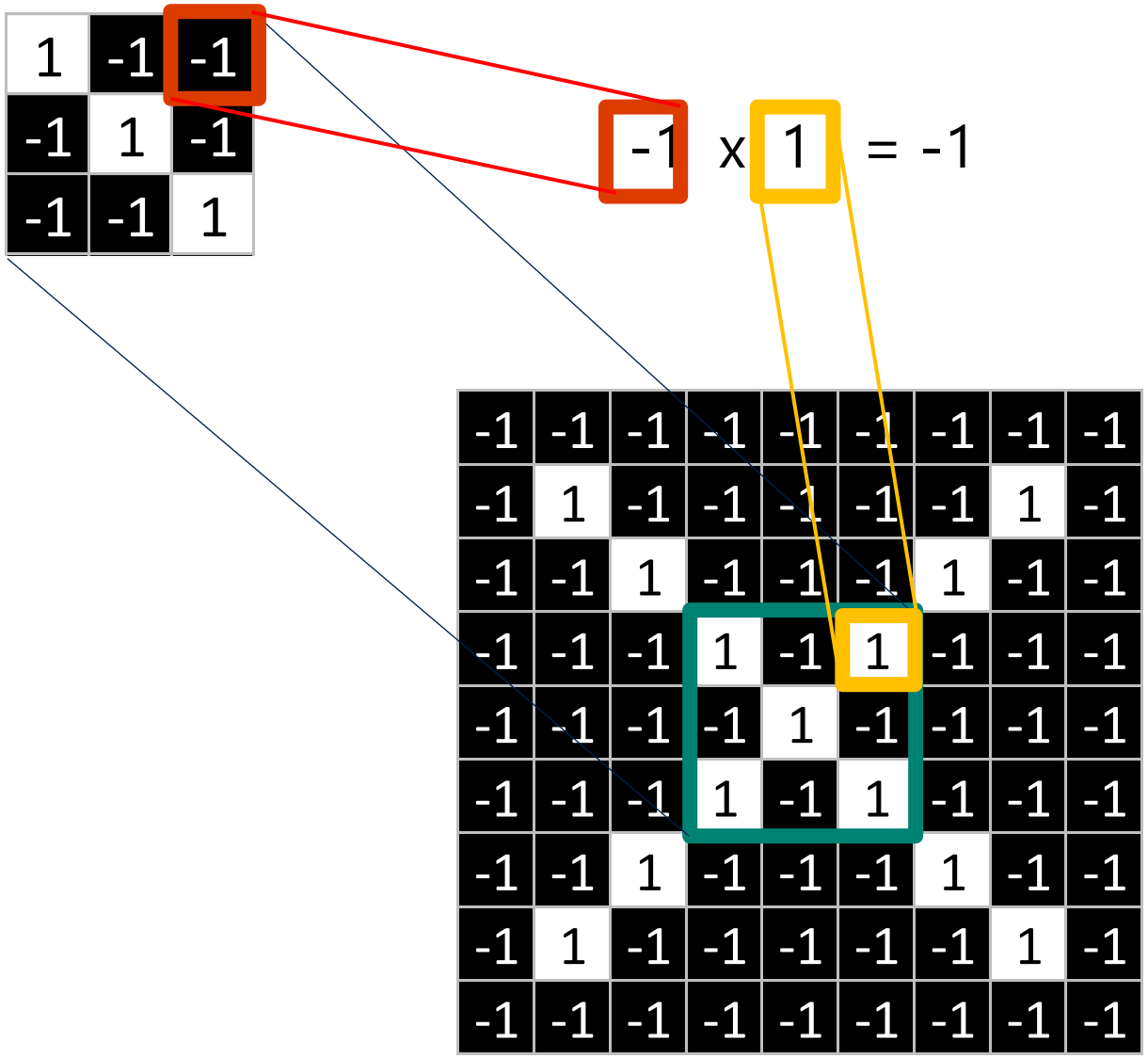

1 -1 -1
-1 1 -1
-1 -1 1
1 x 1 = 1
-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 -1 -1 1 -1 -1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
1

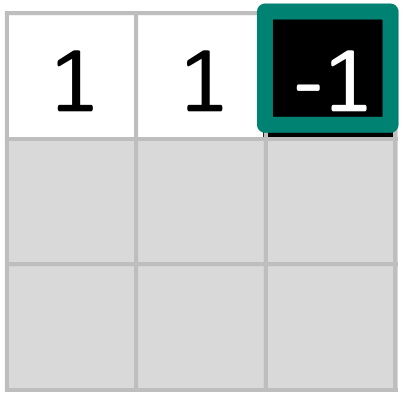


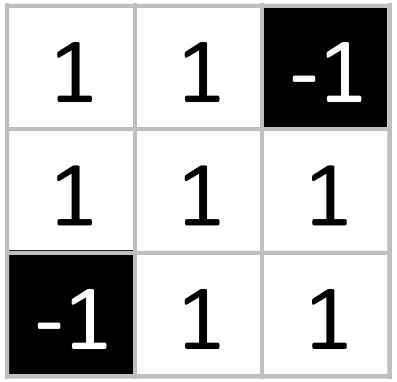


Sum: 9 → 9/9=1
(single neuron)

Move on to another patch







Sum: 5 → 5/9 = 0.55

After convolution over every patch…

9 x 9

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

$\otimes$

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

$=$

7 x 7

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

Feature map

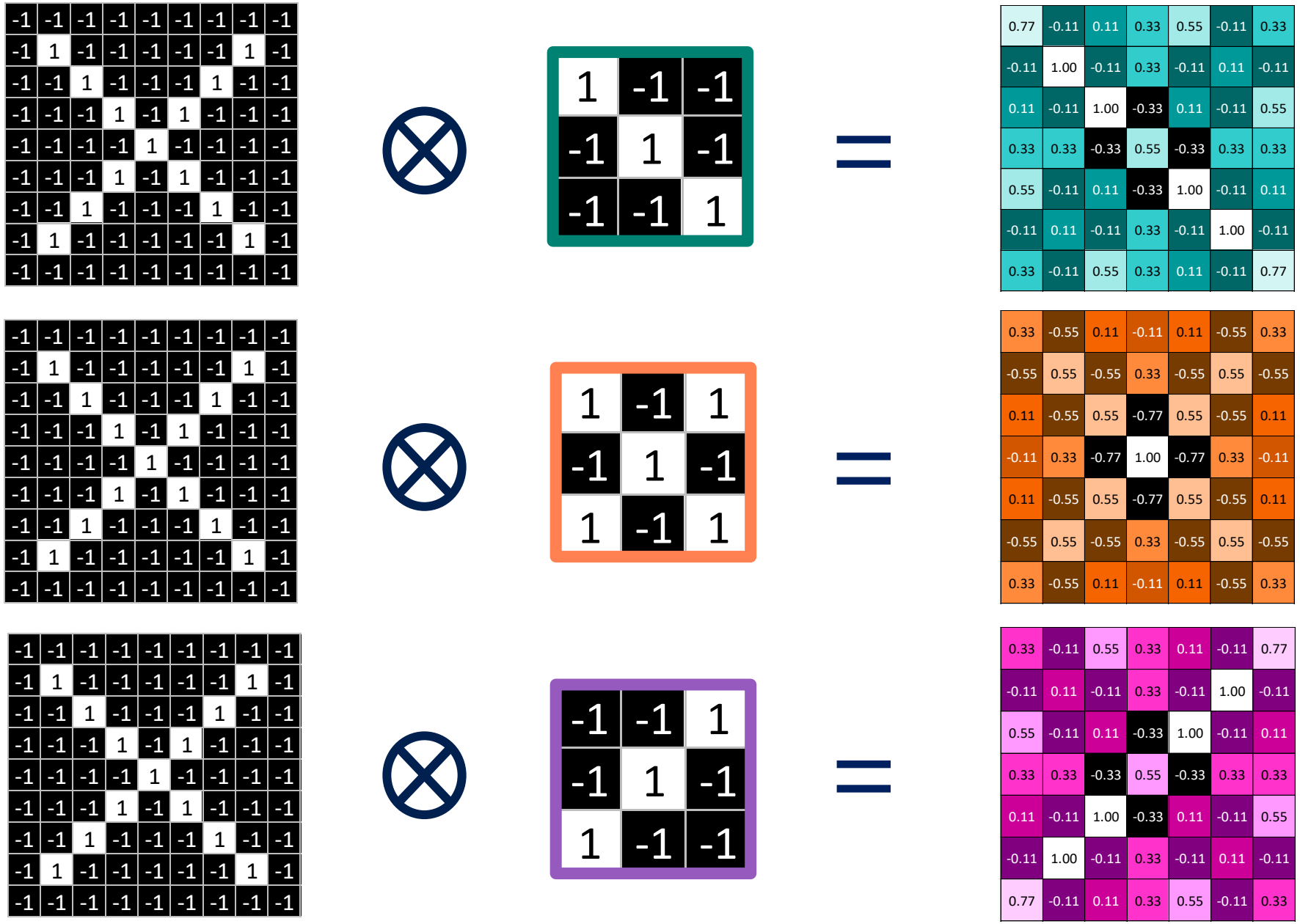
Assuming stride (step size) = 1

Each channel has a filter

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

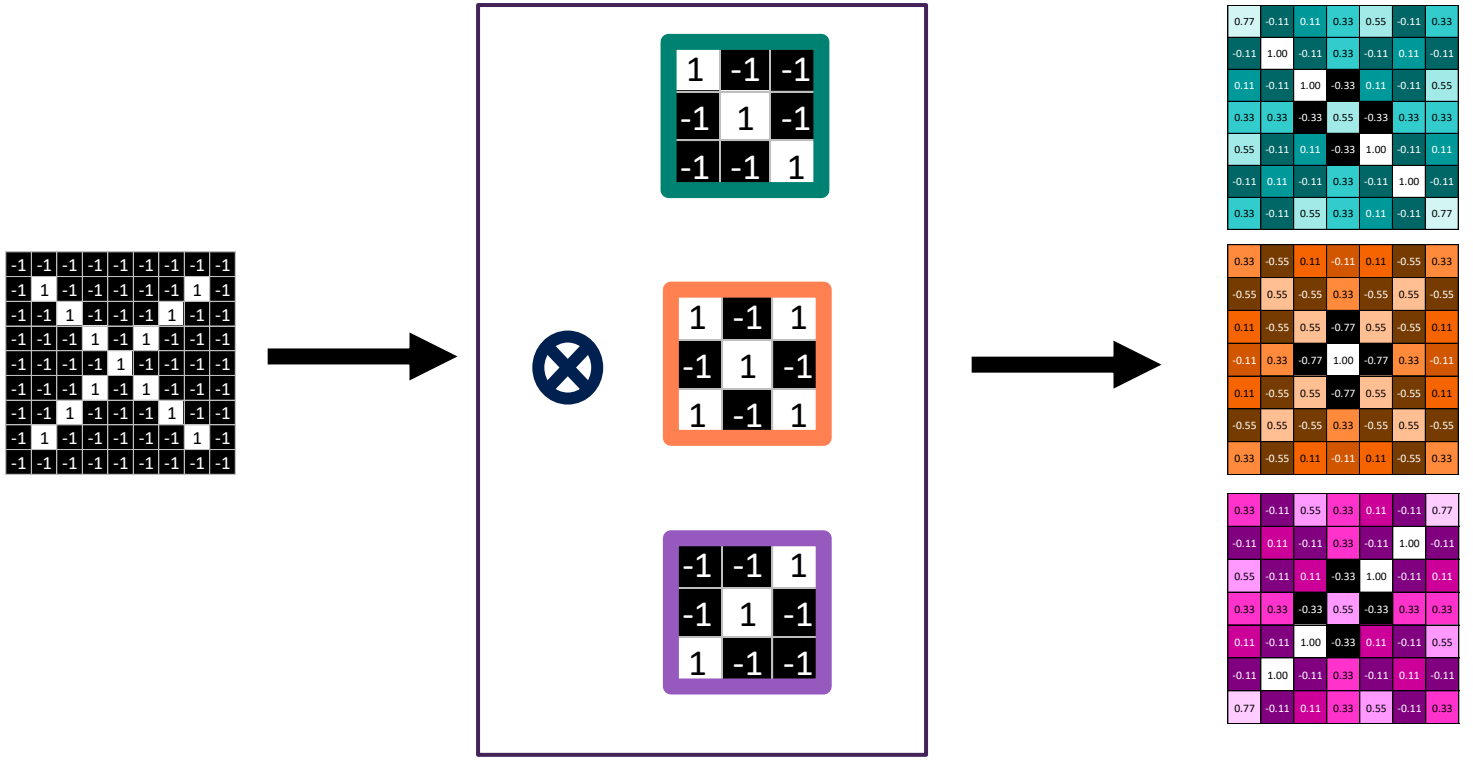
1	-1	1
-1	1	-1
1	-1	1

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

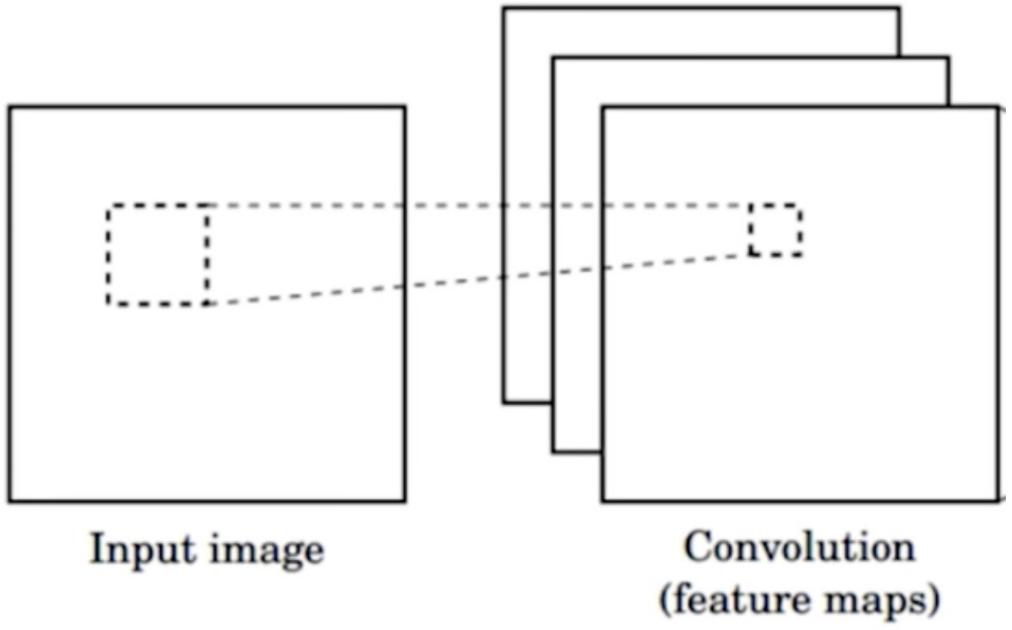

-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 -1 -1 1 -1 -1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
1 -1 -1
-1 1 -1
-1 -1 1
=
0.77 -0.11 0.11 0.33 0.55 -0.11 0.33
-0.11 1.00 -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

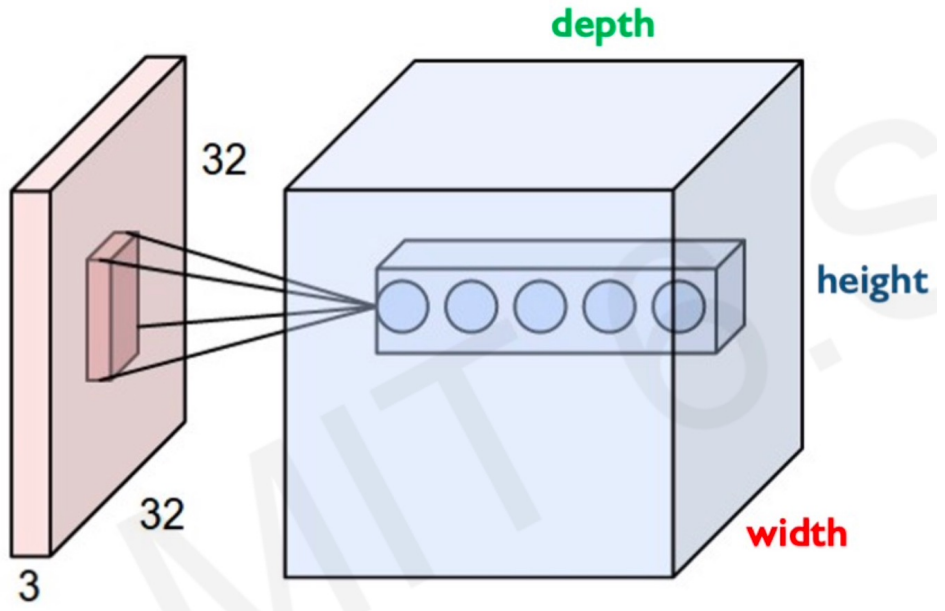
Output of a convolution layer

- A stack of feature maps



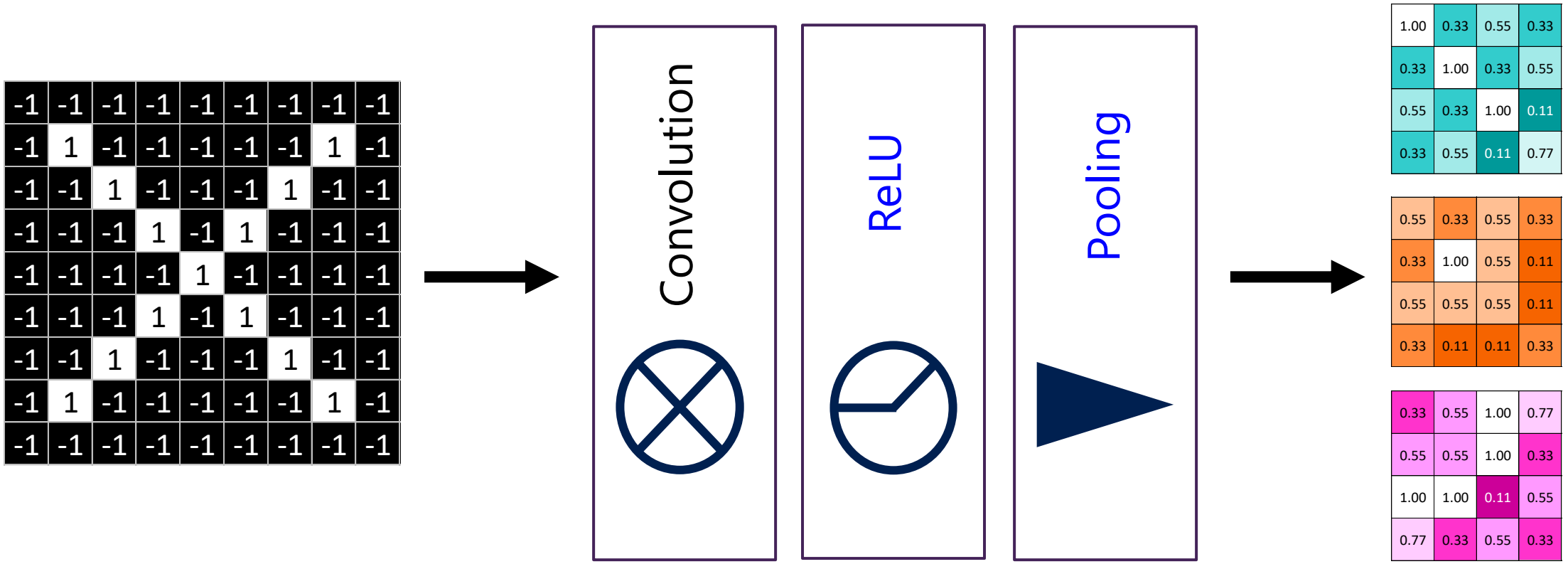
- Stacked together (depth increases)



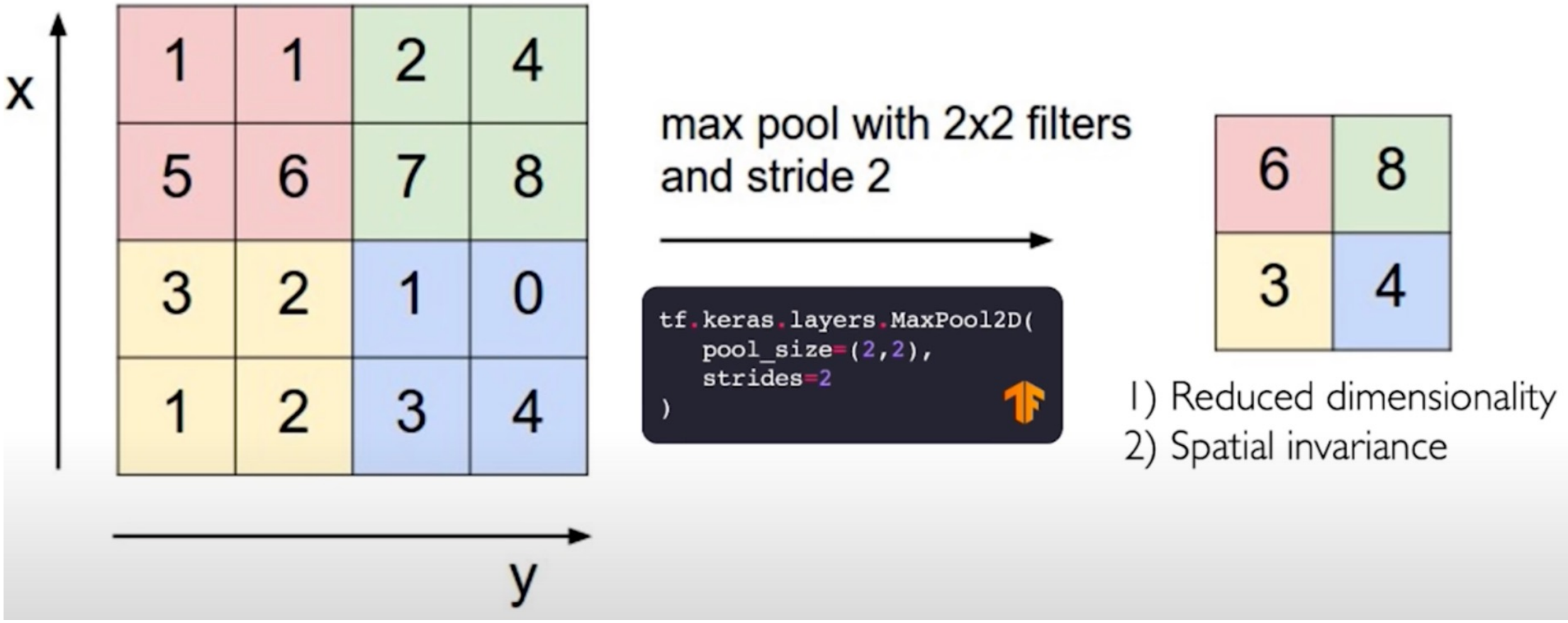


Pipeline

- Add *activation function* to each convolution
  - *Sigmoid, Relu, Tanh, etc.*
- Add pooling to downsample feature map

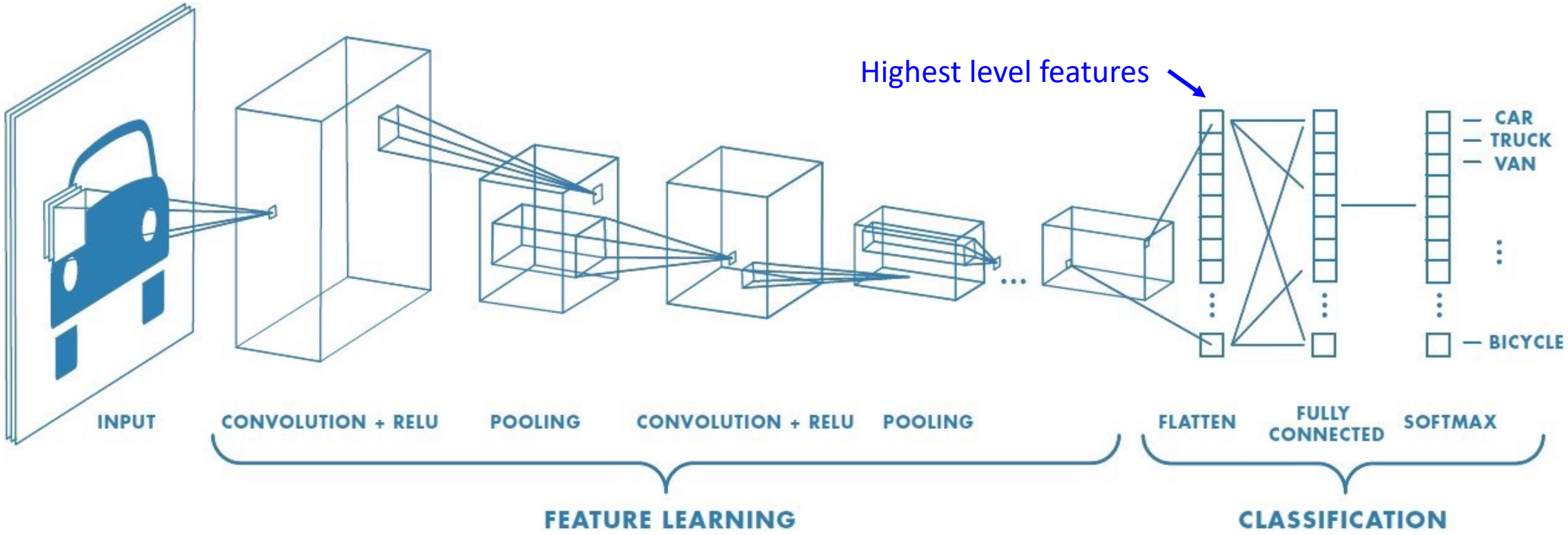


Pooling (max pooling, avg pooling)



Feature map

CNN: concatenates many layers together
(feature extraction) + FC (classification/regression)



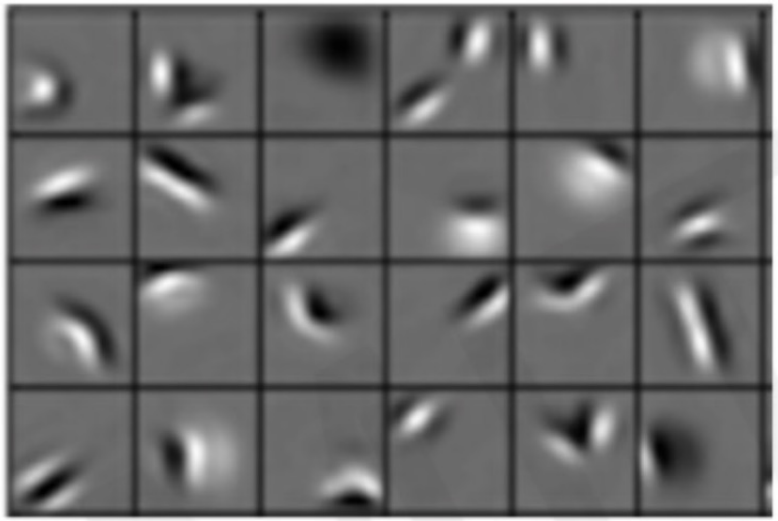
- Convolution + activation + pooling
- Convolution increases depth by <# of filters> folds
- Output of FC: scores of the input belonging to each class

SOFTMAX FUNCTION

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Intuition of “features”

Low level features



Edges, dark spots

Mid level features

Eyes, ears, nose

High level features

Facial structure

CNN is Versatile

- Image classification
- Object detection
- Segmentation
- … (probabilistic control / robotics)

Object detection

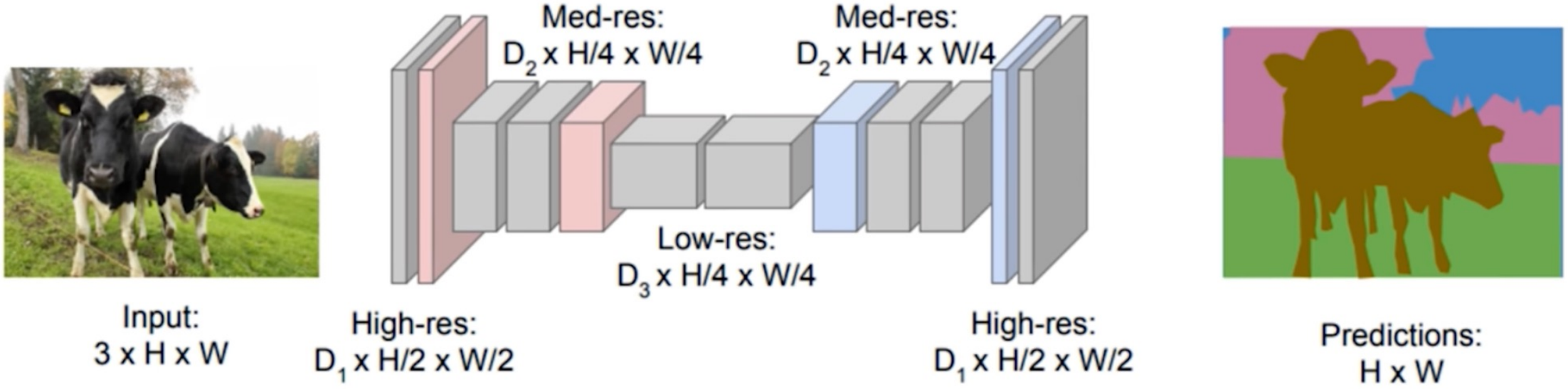
Output:
taxi: (x1, y1, w1, h1)

Multi-OD

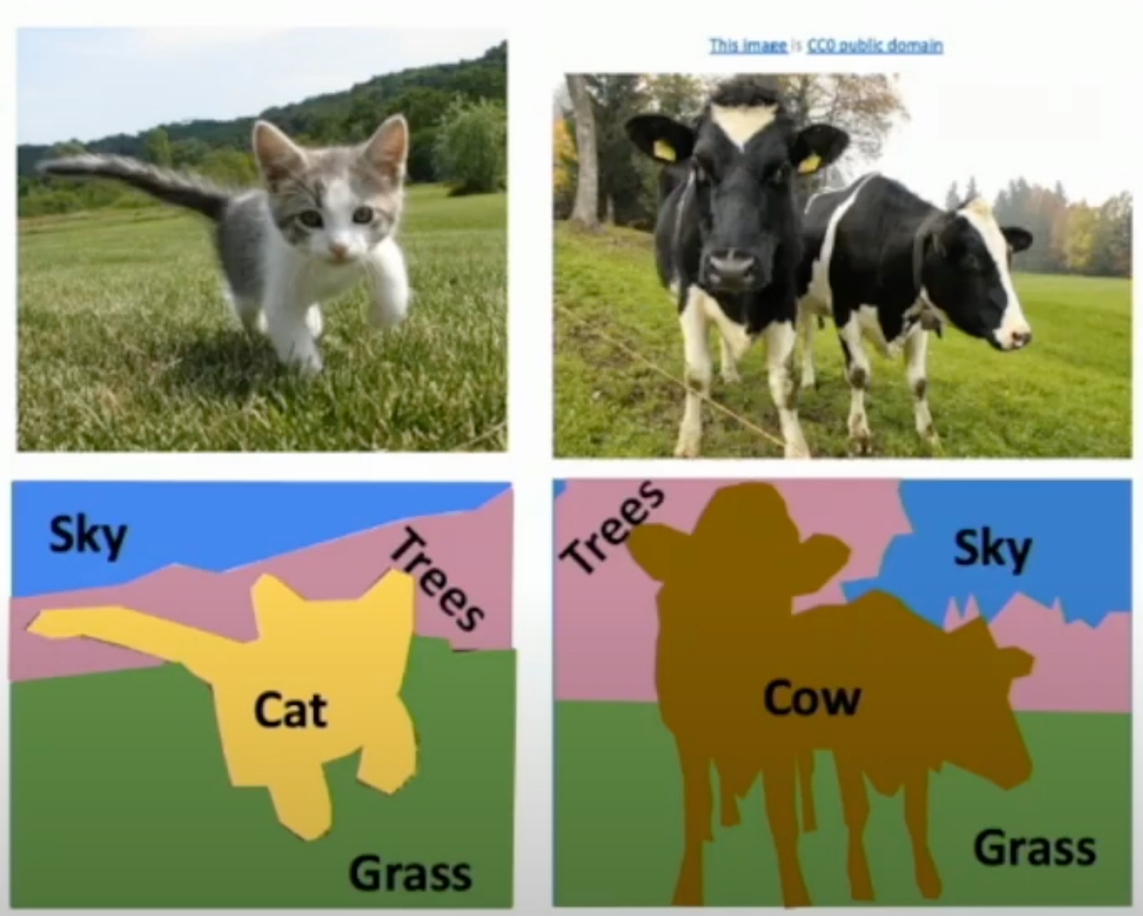
Output:
taxi: (x1, y1, w1, h1)
person: (x2, y2, w2, h2)
person: (x3, y3, w3, h3)
....

Semantic segmentation:

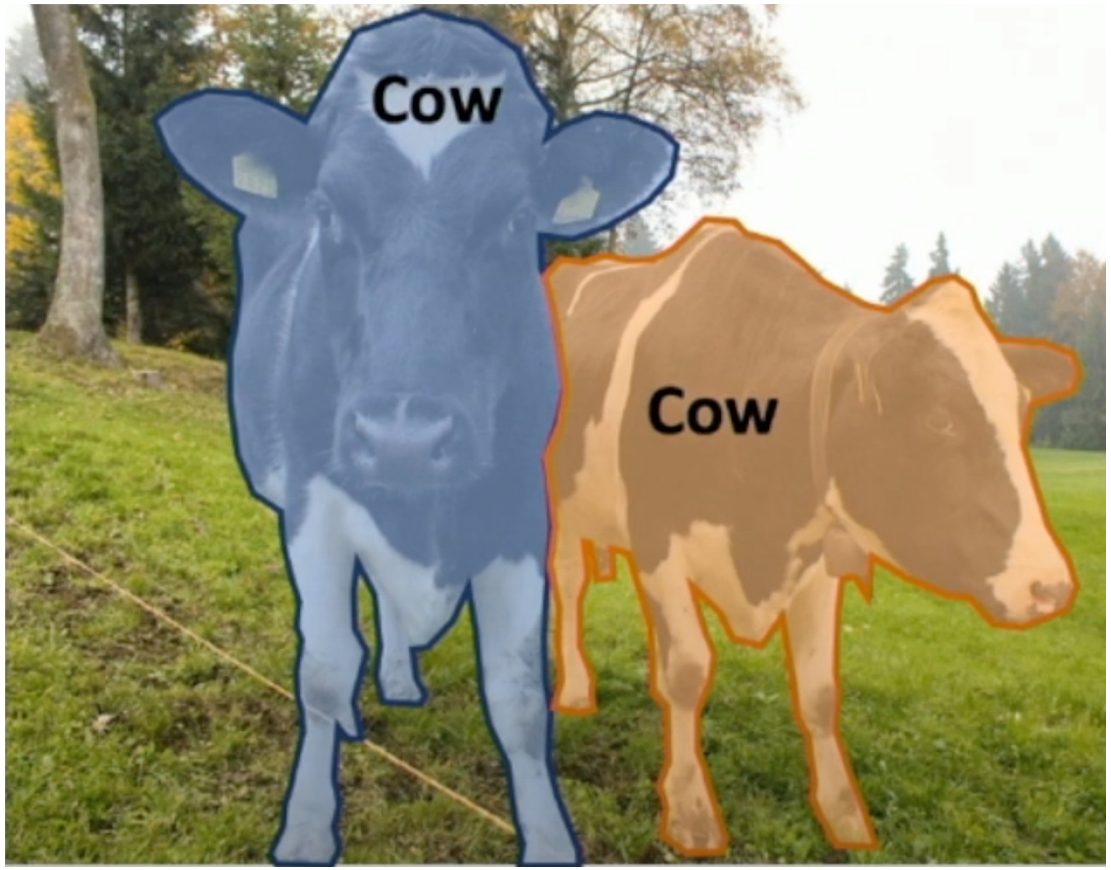
Assign a class label (membership) to every pixel



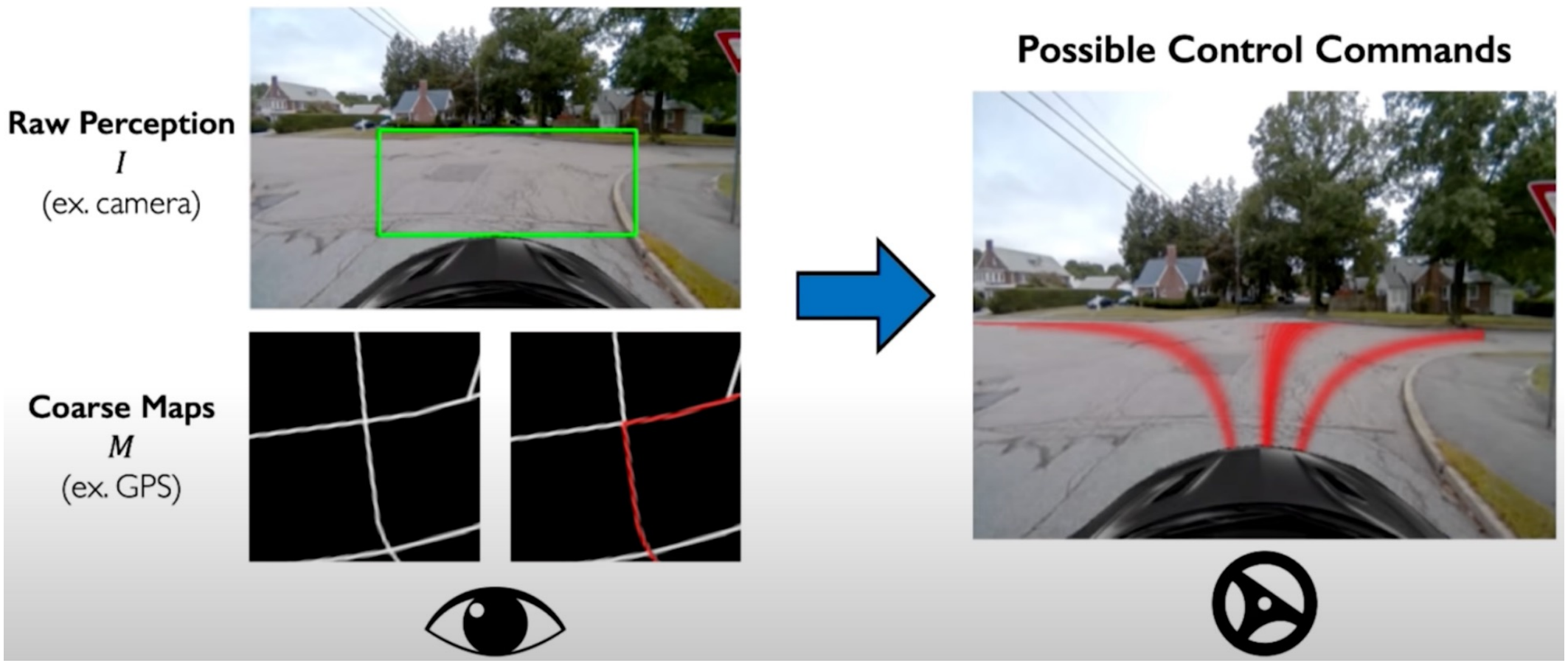
Semantic segmentation (pixels)



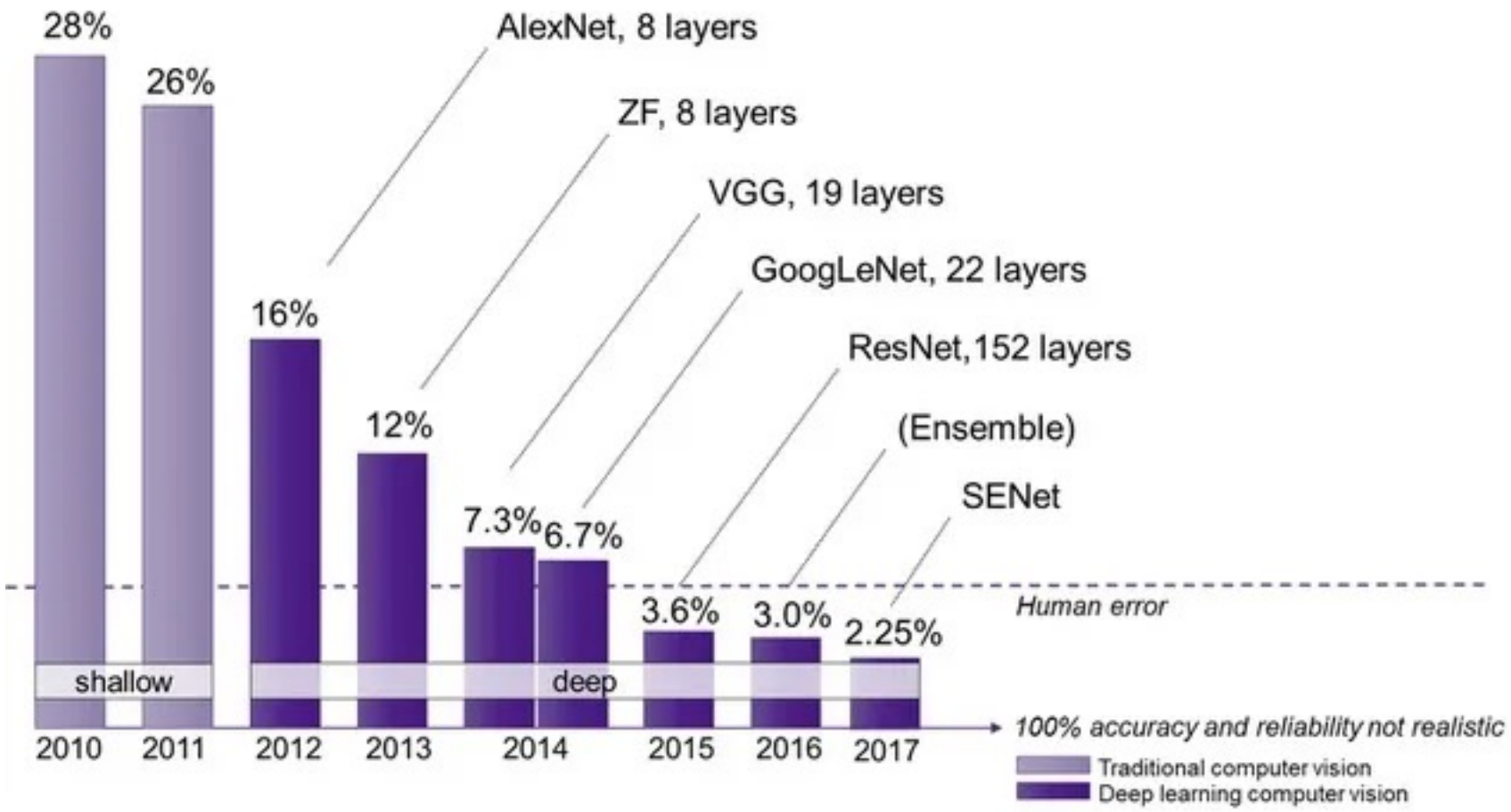
Instance segmentation (objects)



Robotic control



Anecdote: ImageNet competition (ILSVRC)



luman error is about 5.1%

Now let’s build our own CNN in Pytorch

Drawer slide

- Output dimension = (W – F + 2P) / S + 1

- W: width & height
- F: filter size
- P: padding
- S: stride

Coding Demo:

1_mlp_Day1.ipynb
2_cnn_Day1.ipynb
3_transfer_Day1.ipynb
4_save_load_Day1.ipynb
5_PINNs_Day1.ipynb