Deep Learning Tutorial
Part II: Advanced concepts

June 16, 2021

Auliya Fitri, Jakob Gawlikowski

Machine Learning Group
Institute of Data Science (DW)
Jena

Curriculum

A: Theoretical introduction – Morning
I. Introduction and basics
II. Advanced concepts
III. Practical application

B. Hands-on seminar – Afternoon
Run prepared Jupyter Notebooks online on Binder, or locally on your own laptop.

Curriculum

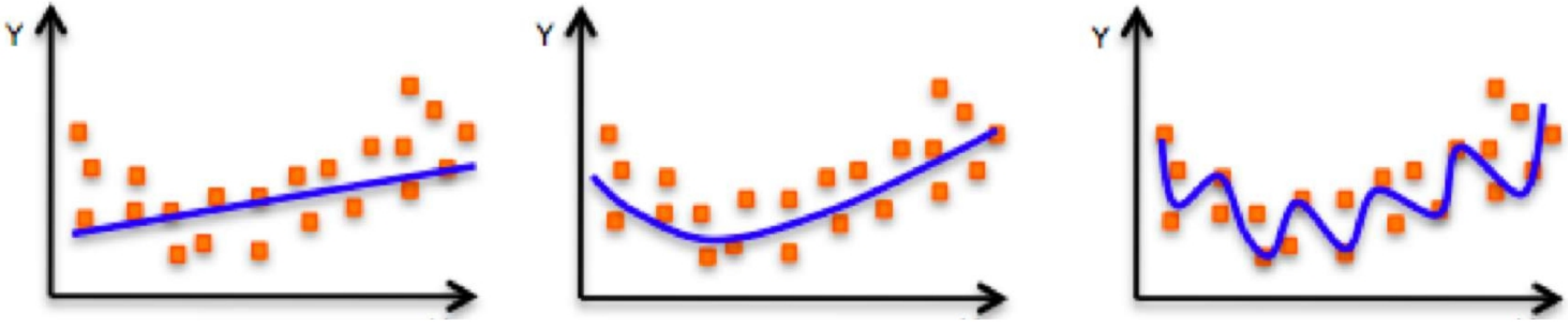
II. Advanced concepts

- Regularization
- Convolutional Neural Networks (CNNs)

Inspired by lectures from MIT and Hacettepe Üniversitesi; images taken from these, if not noted otherwise

Regularization

Regularization
The overfitting problem



Underfitting
Model does not have capacity to fully learn the data

Ideal fit

Overfitting
Too complex, extra parameters, does not generalize well

= High bias

Tradeoff

= High variance

Regularization

Preventing overfitting: Early stopping

Regularization

Preventing overfitting: l_1 regularization

We want to find the network weights that ***achieve the lowest loss***

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right) + \frac{\lambda}{2} \sum_{l} \| \boldsymbol{W}_l \|$$

→ Leads to sparser weights (more zeroes in weights) that are not too adapted to the data at hand

Regularization
Preventing overfitting: Data augmentation

Original

Flip

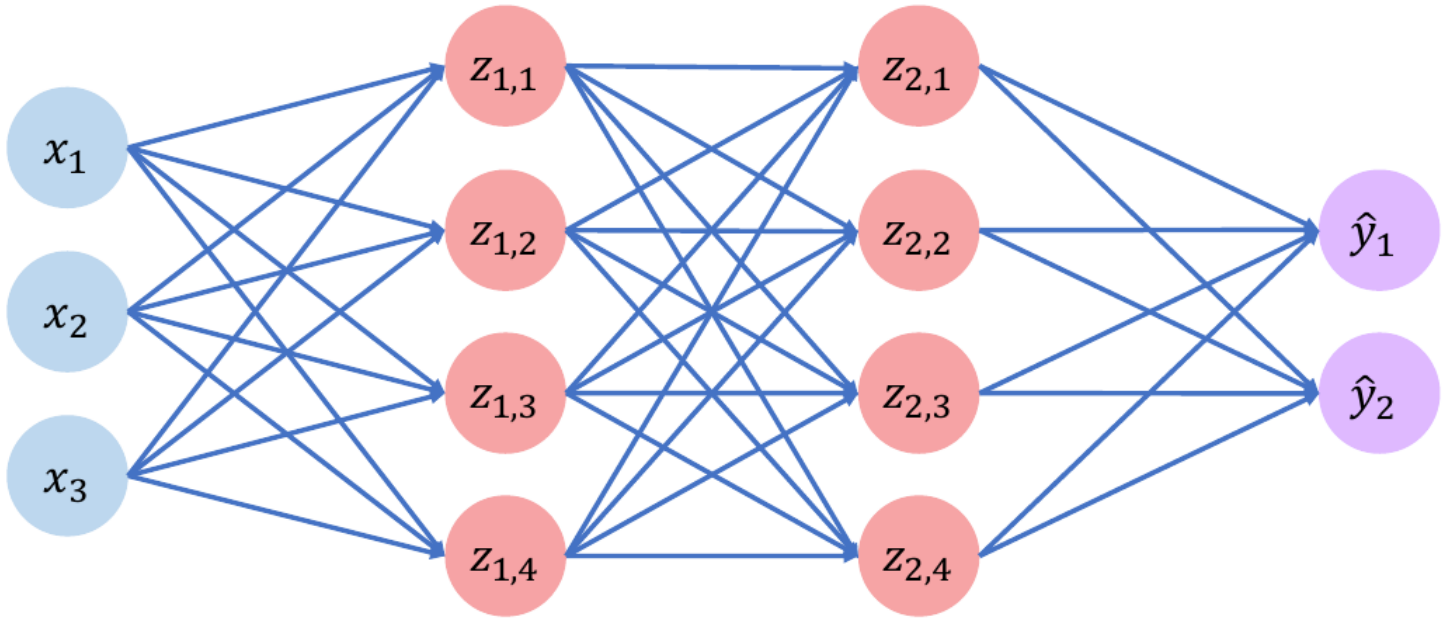
Random crop

Contrast

Tint

Regularization
Preventing overfitting: Dropout

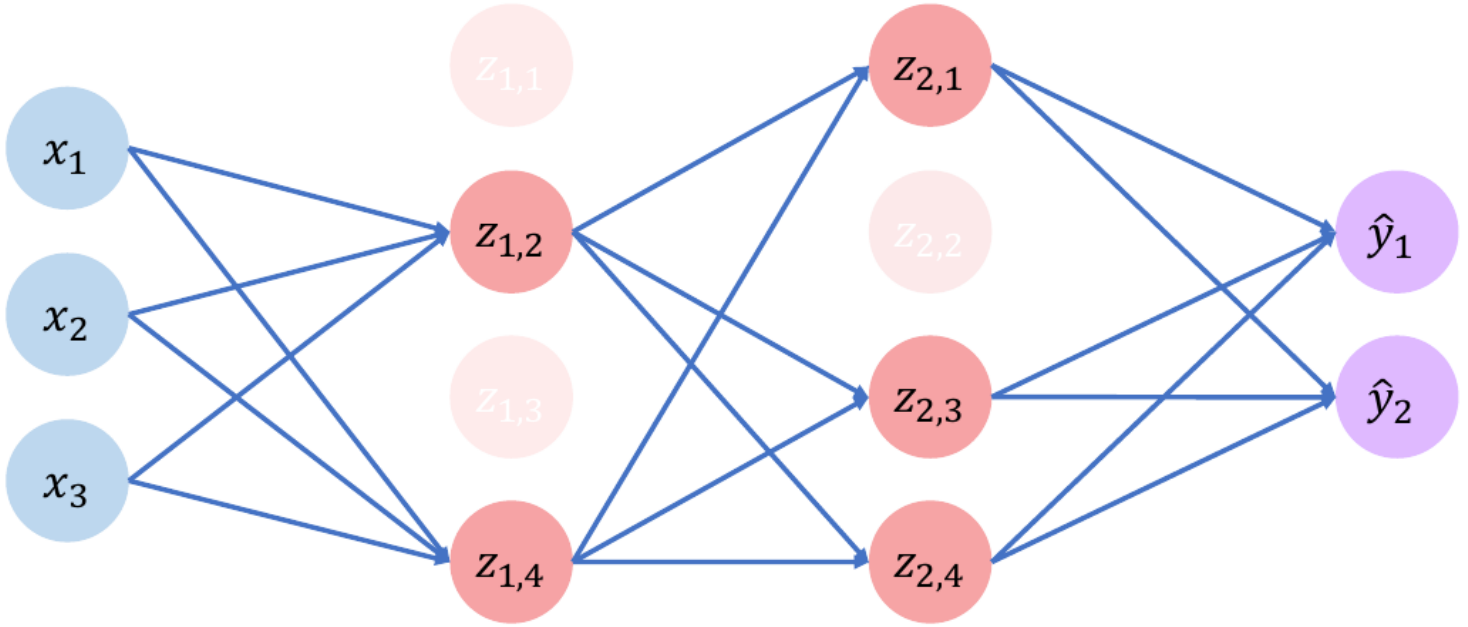
- During training, randomly set some activations to 0



Regularization
Preventing overfitting: Dropout

- During training, randomly set some activations to 0
  - Typically 'drop' 50% of activations in layer
  - Forces network to not rely on any 1 node

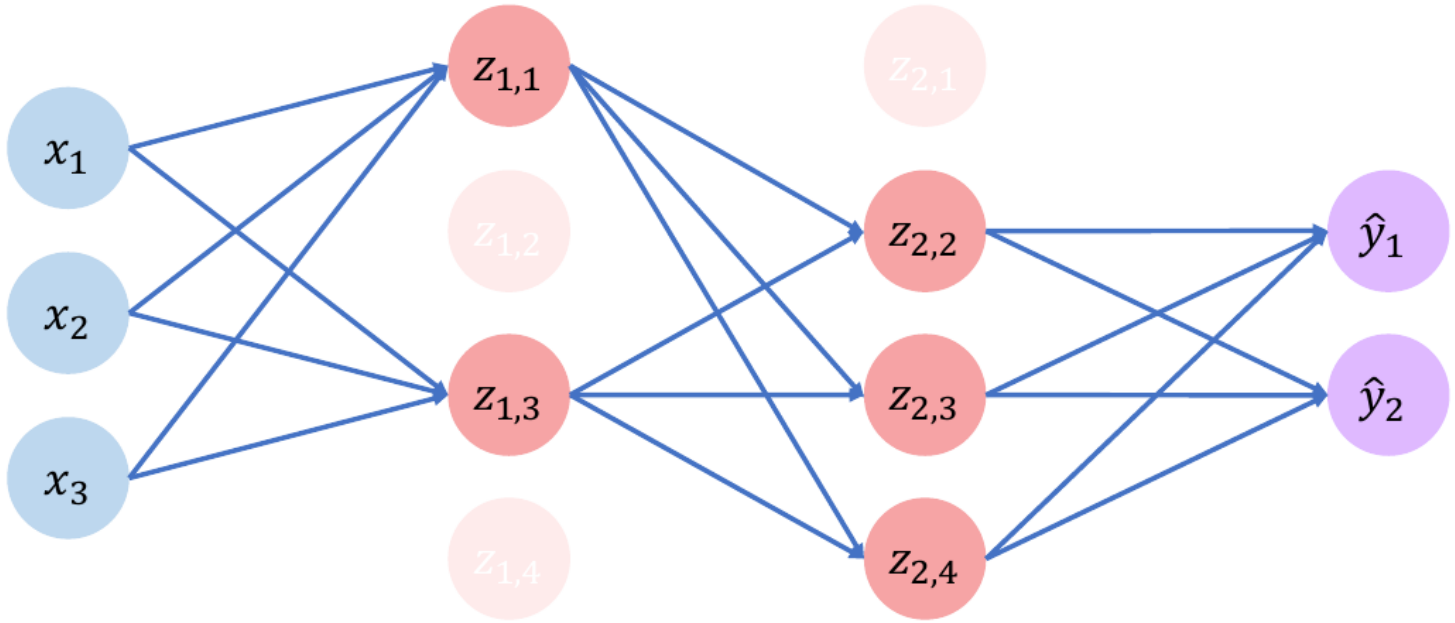
```
tf.keras.layers.Dropout(p=0.5)
```



Regularization
Preventing overfitting: Dropout

- During training, randomly set some activations to 0
  - Typically 'drop' 50% of activations in layer
  - Forces network to not rely on any 1 node

`tf.keras.layers.Dropout(p=0.5)`



Other training considerations

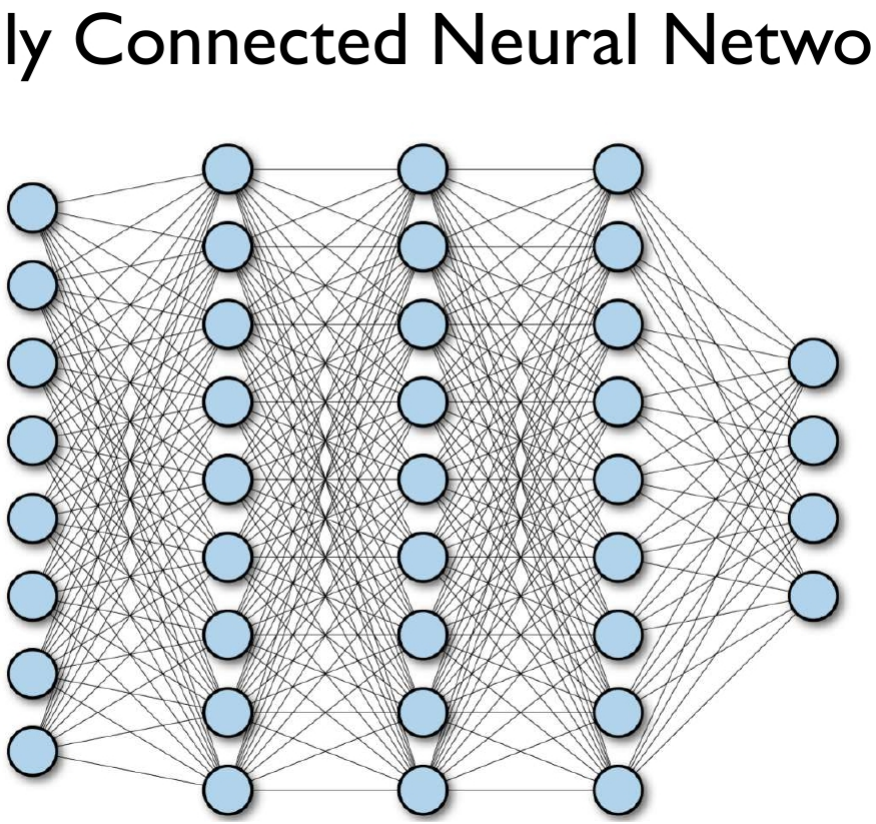
There are a few practical issues (translating to hyper-parameters) for which we do not have enough time, such as:

- Data preprocessing *(normalize your inputs)*
- Network initialization *(use random initializations, e.g. Glorot)*
- Batch normalization *(use)*
- Optimizers, training schedules, learning rates *(when in doubt, use Adam; try different learning rates)*
- Minibatch composition *(shuffle your data sensibly, use large minibatches)*

Fortunately, frameworks and pre-implemented code often already take care of this; nevertheless, you should keep these influences in mind.

Convolutional Neural Networks (CNNs)

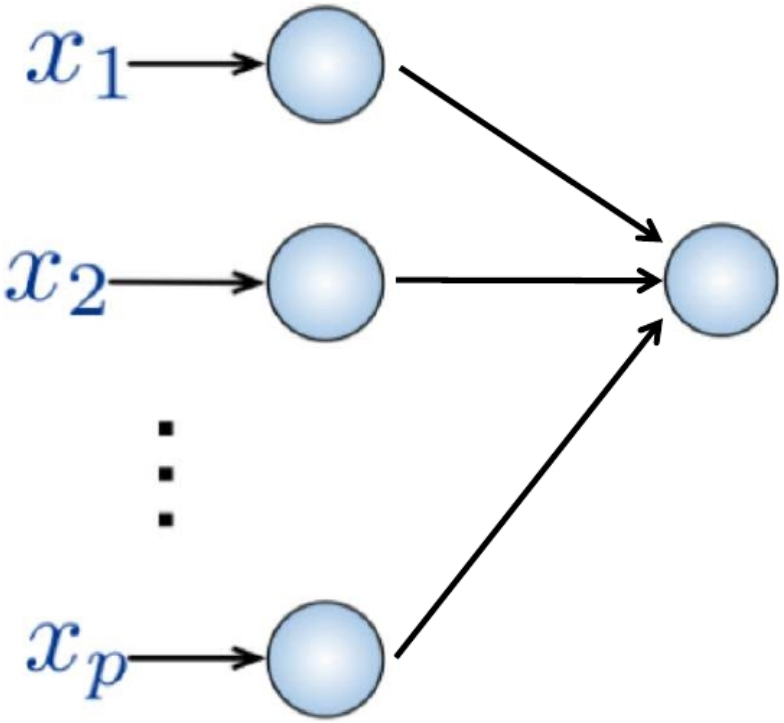
Convolutional Neural Networks
Learning on image data



Convolutional Neural Networks
Learning on image data

Input:

- 2D image
- Vector of pixel values



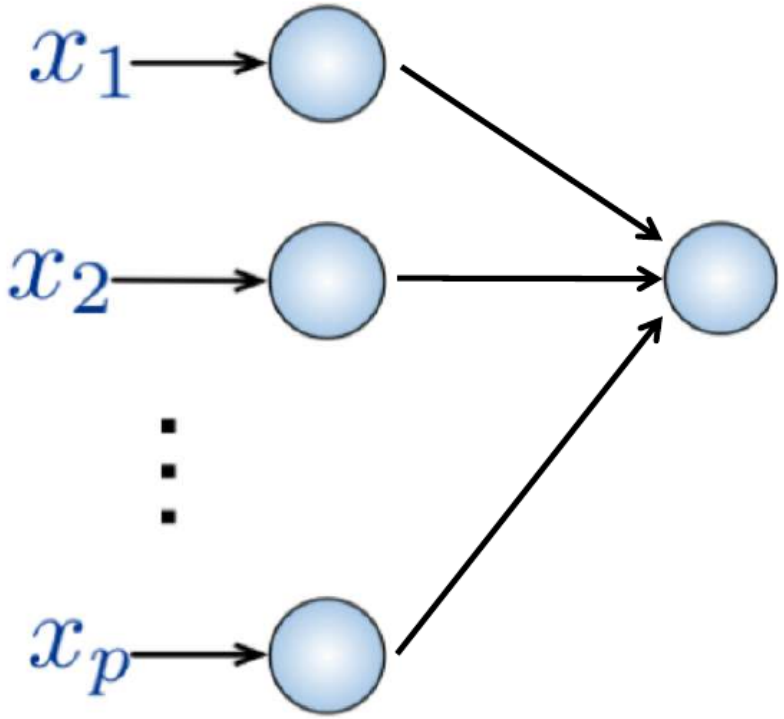
Fully Connected:

- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

Convolutional Neural Networks
Learning on image data

Input:

- 2D image
- Vector of pixel values



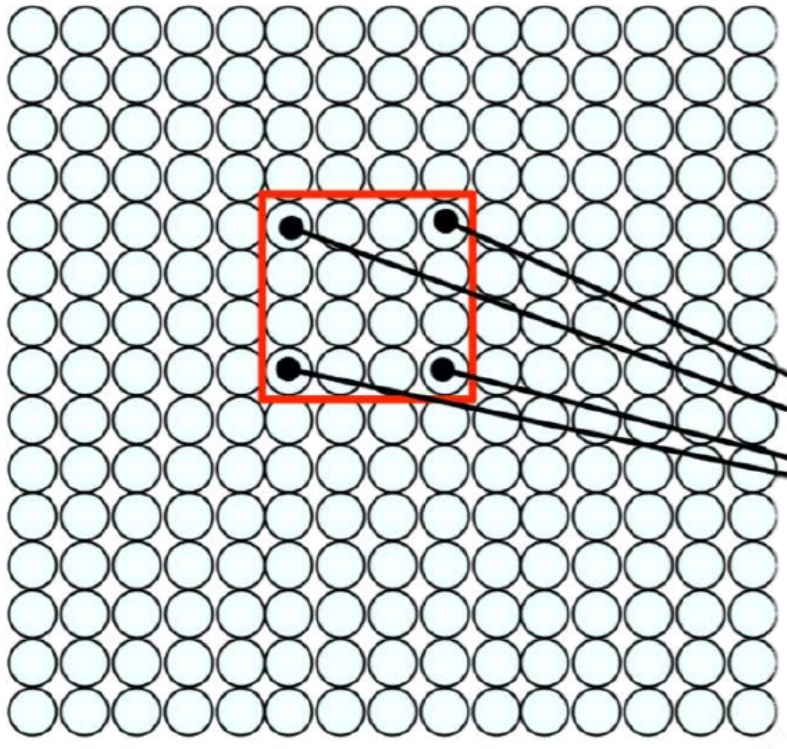
Fully Connected:

- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

How can we use **spatial structure** in the input to inform the architecture of the network?

Convolutional Neural Networks
Learning on image data

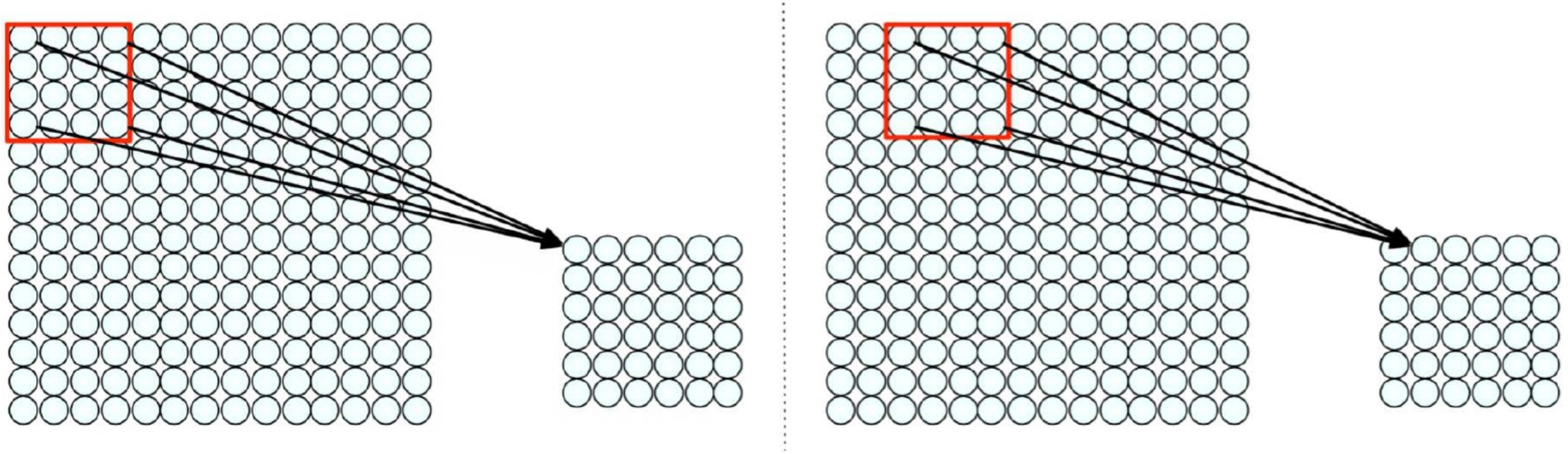
Input: 2D image.
Array of pixel values



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

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

Convolutional Neural Networks
Learning on image data



Connect patch in input layer to a single neuron in subsequent layer.
Use a sliding window to define connections.
How can we **weight** *the patch to detect particular features?*

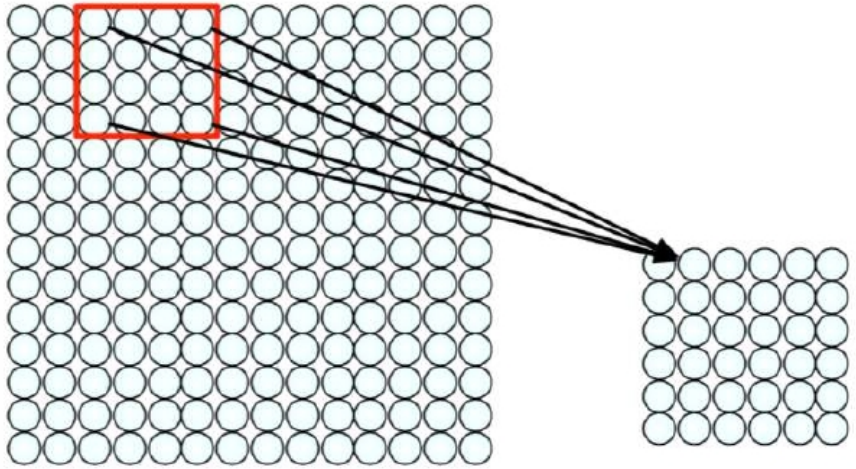
Convolutional Neural Networks
Learning on image data

1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) Spatially **share** parameters of each filter
(features that matter in one part of the input should matter elsewhere)

Convolutional Neural Networks
Learning on image data



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

This "patchy" operation is **convolution**

1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) **Spatially share** parameters of each filter

Convolutional Neural Networks
Filters to detect features

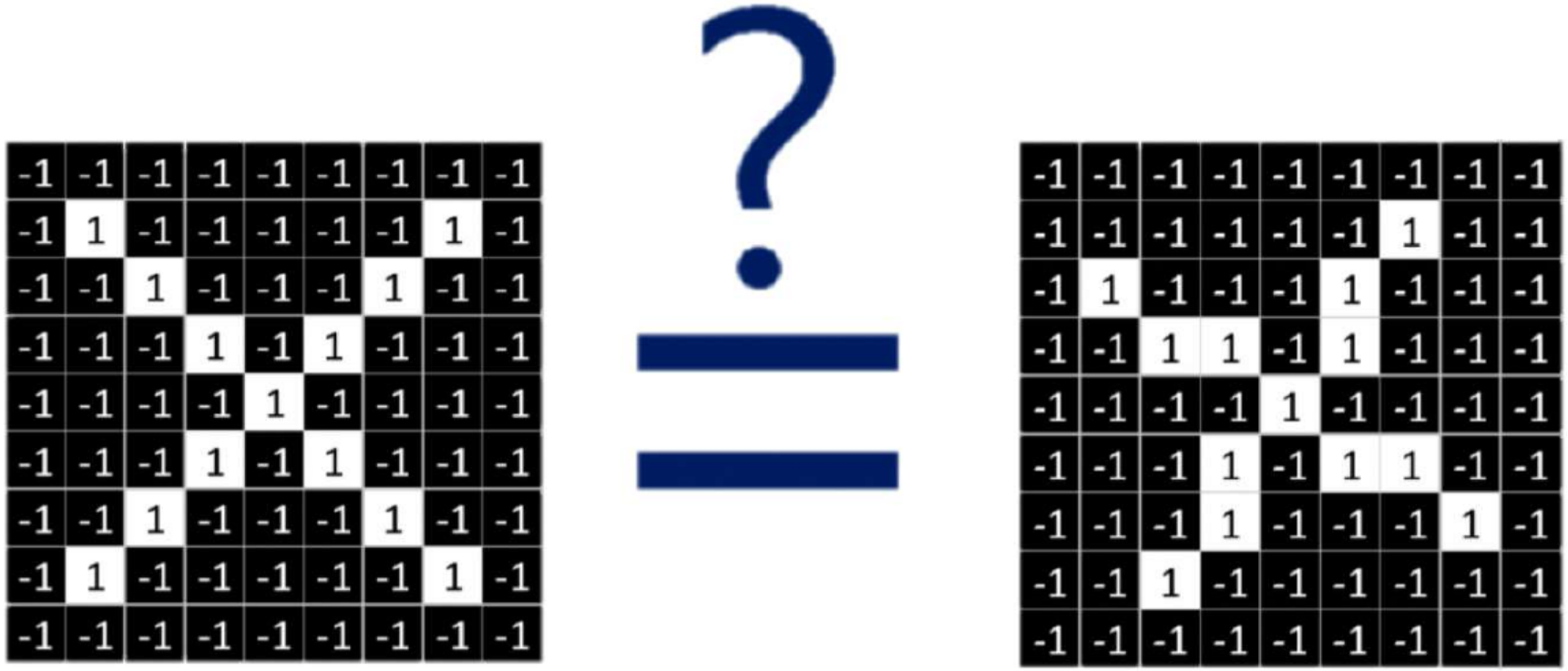
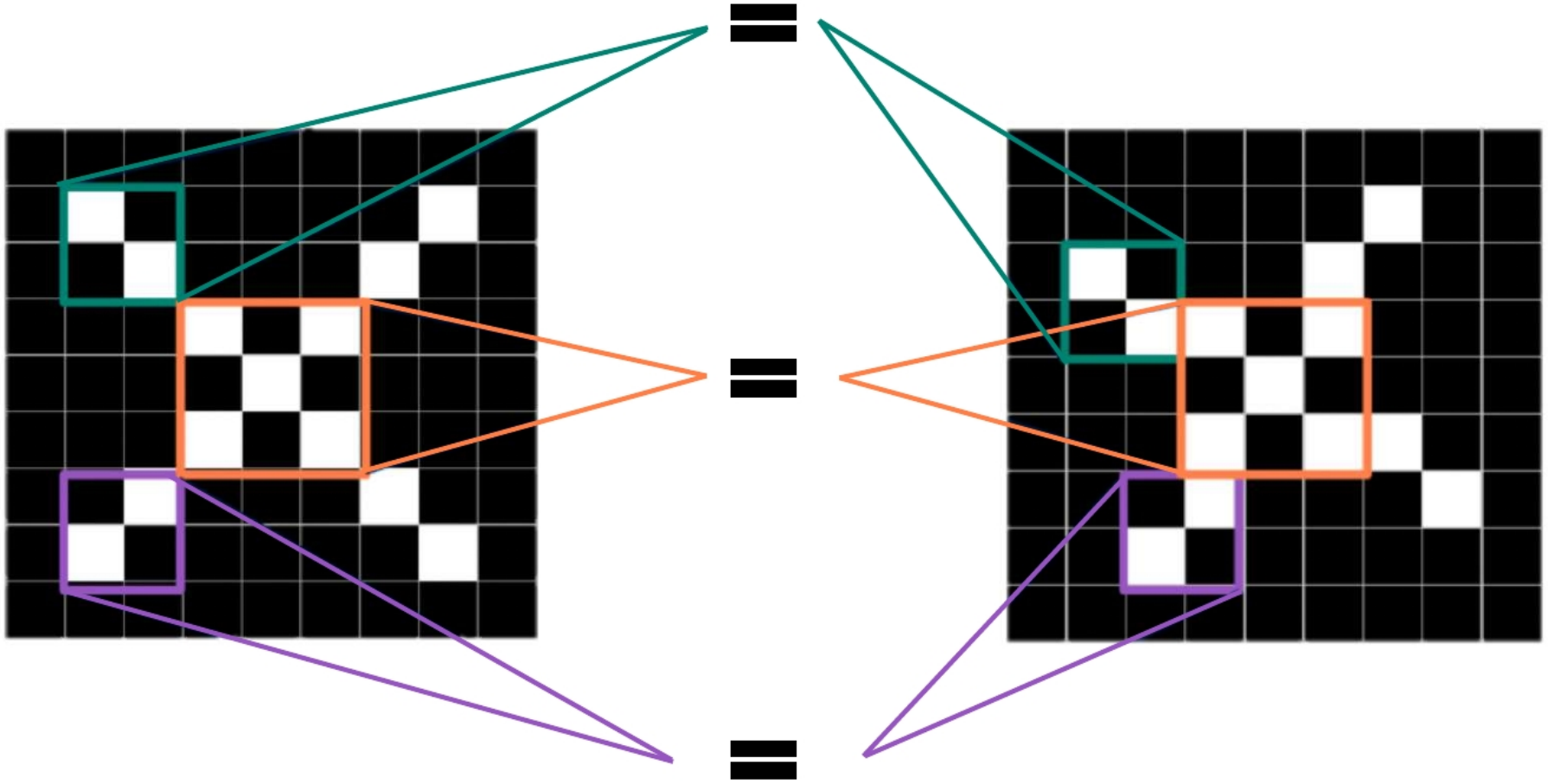
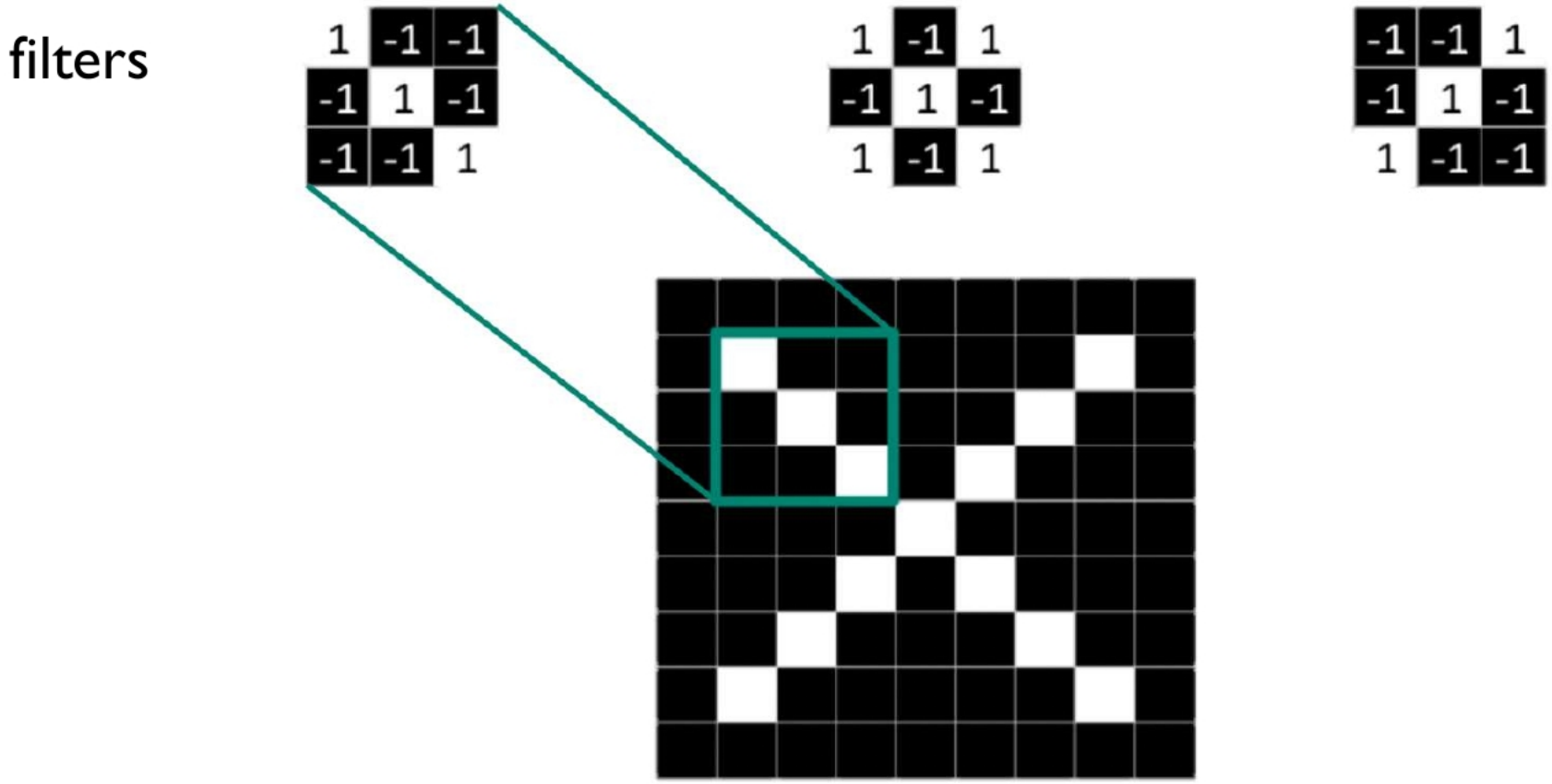


Image is represented as matrix of pixel values… and computers are literal!
We want to be able to classify an X as an X even if it's shifted, shrunk, rotated, deformed.

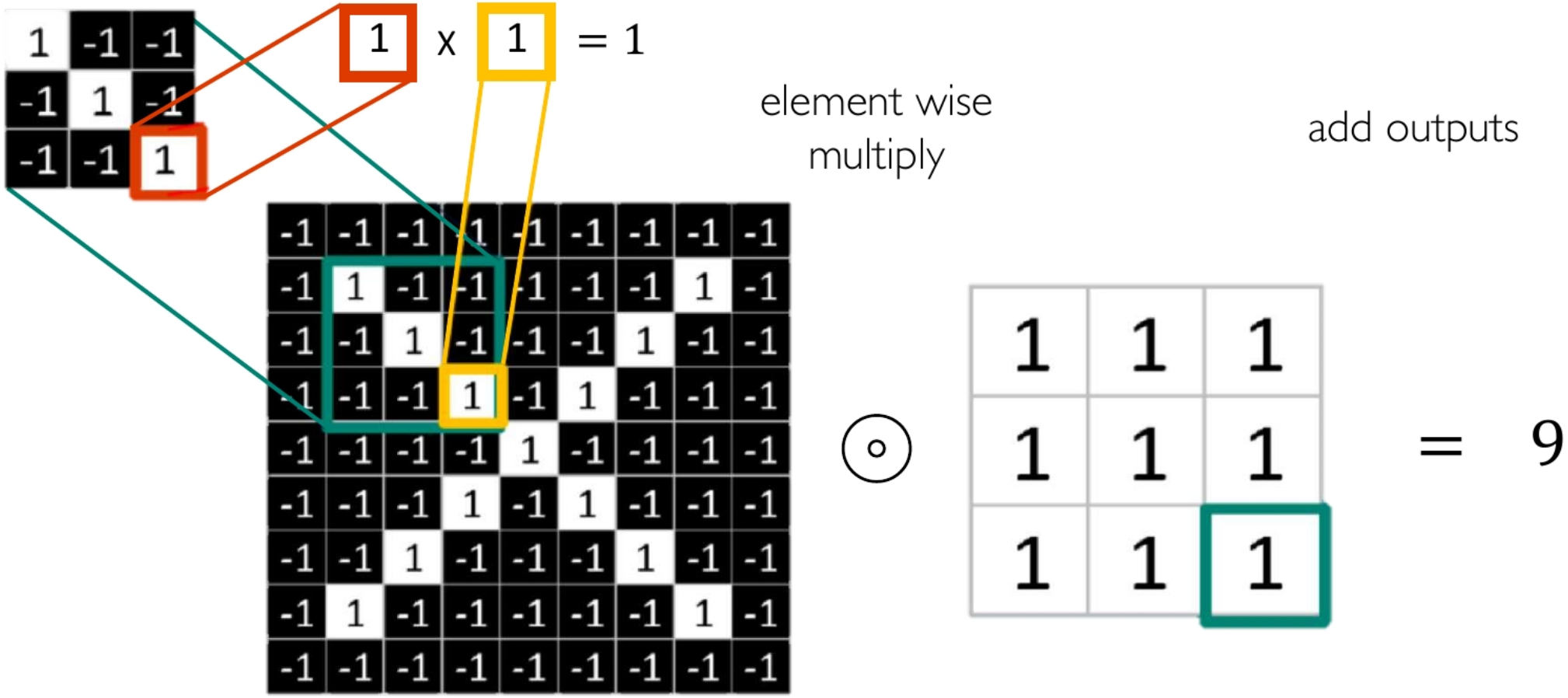
Convolutional Neural Networks
Filters to detect features



Convolutional Neural Networks
Filters to detect features

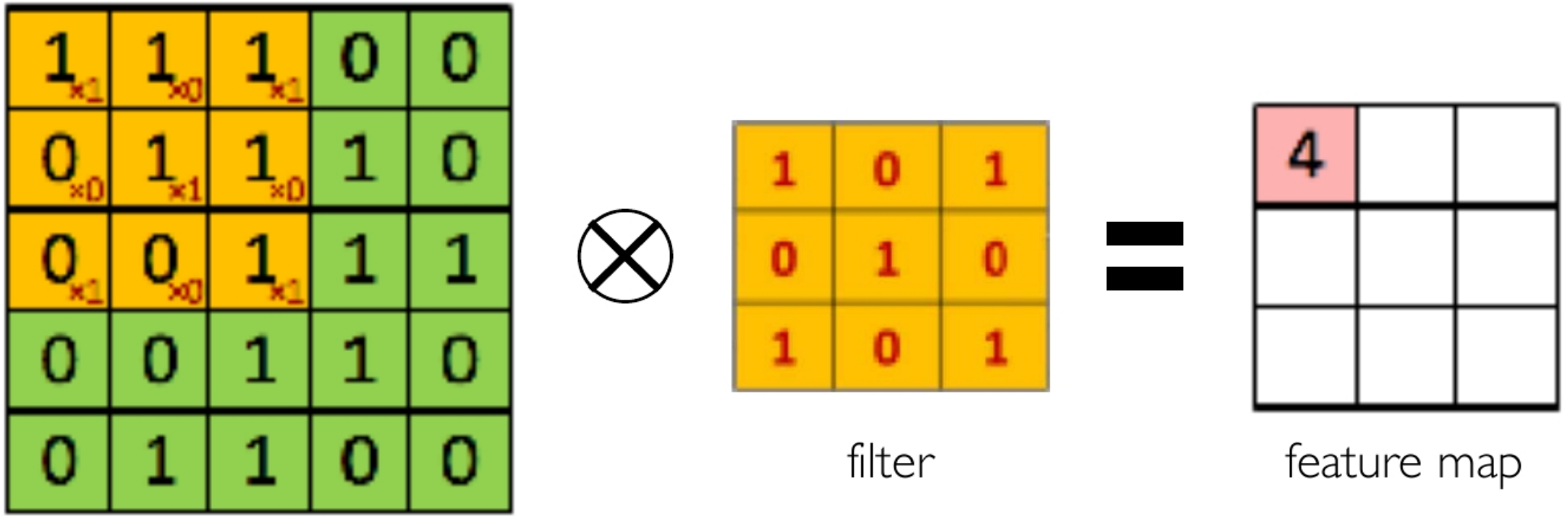


Convolutional Neural Networks
The convolution operation

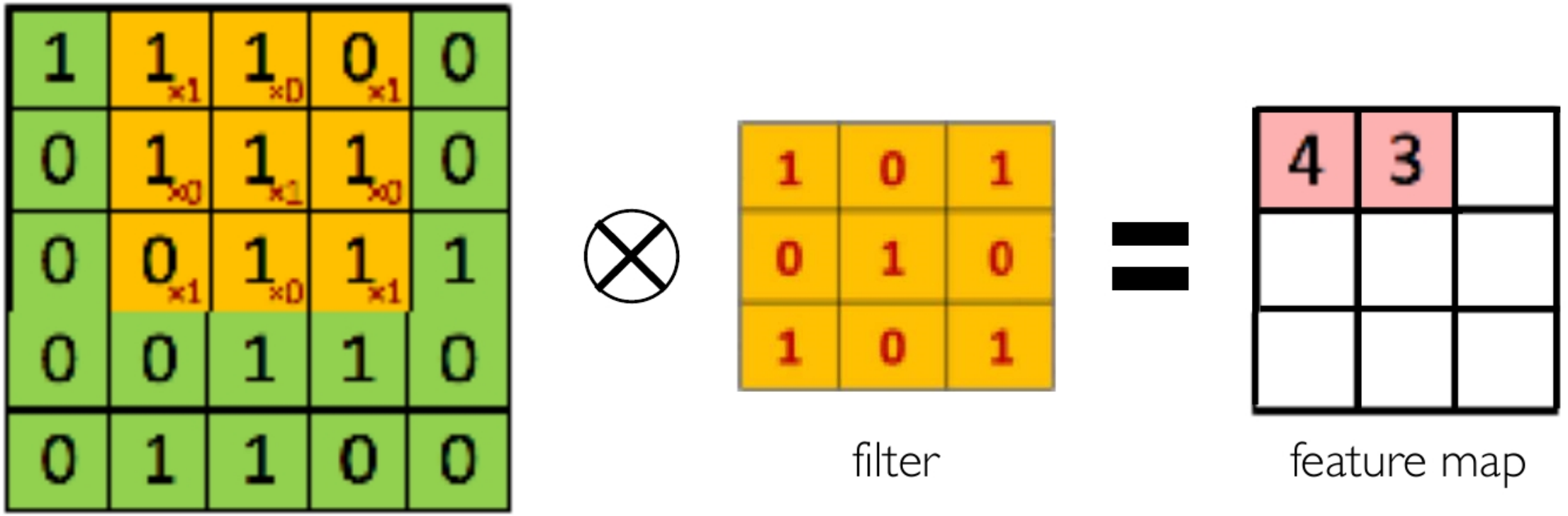


Convolutional Neural Networks
The convolution operation

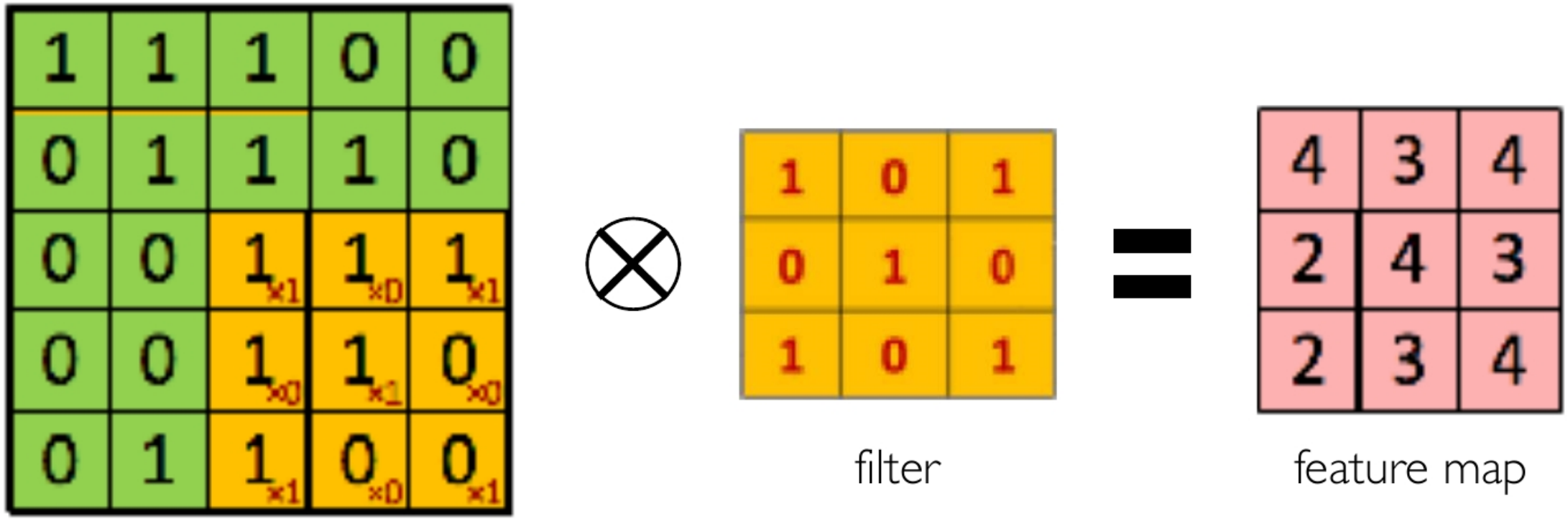
We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



Convolutional Neural Networks
The convolution operation



Convolutional Neural Networks
The convolution operation



Convolutional Neural Networks
Feature extraction with convolutions

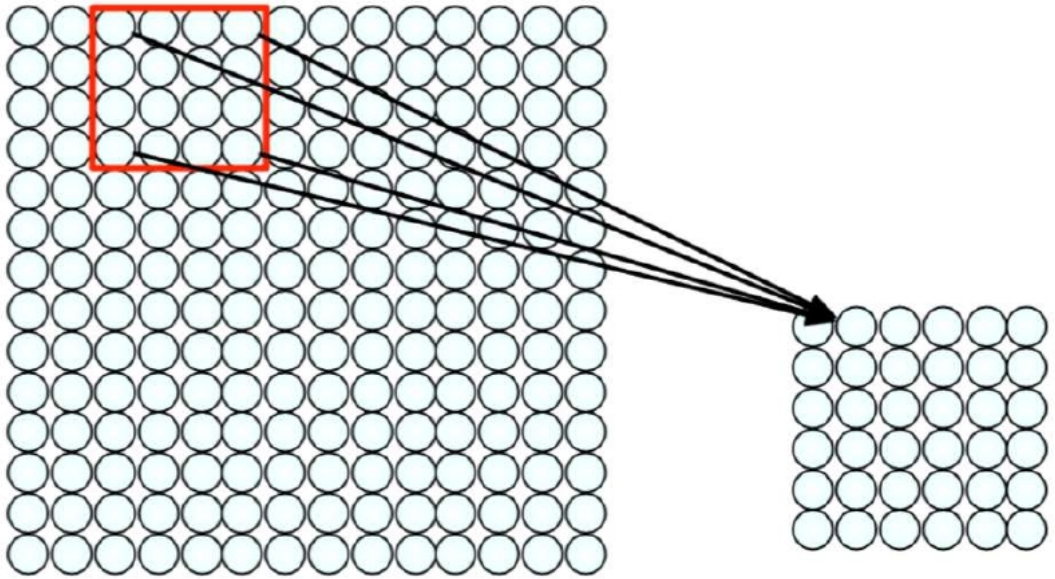
Original

Sharpen

Edge Detect

"Strong" Edge Detect

Convolutional Neural Networks
Feature extraction with convolutions

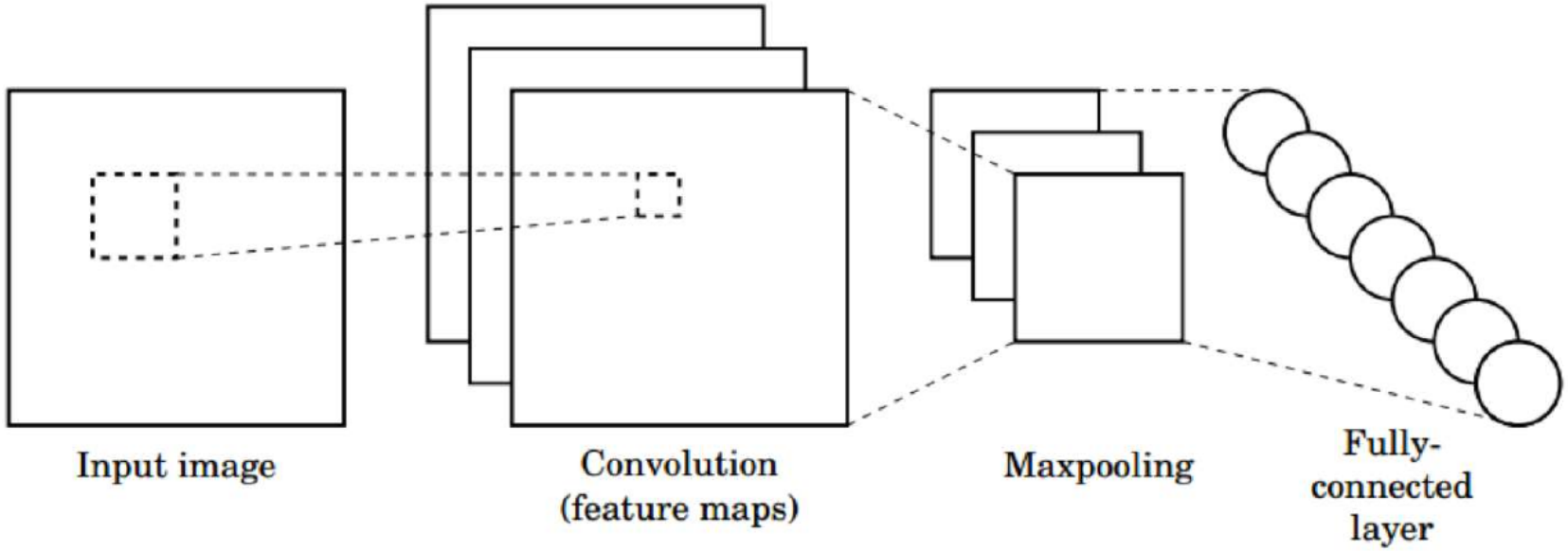


1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) **Spatially share** parameters of each filter

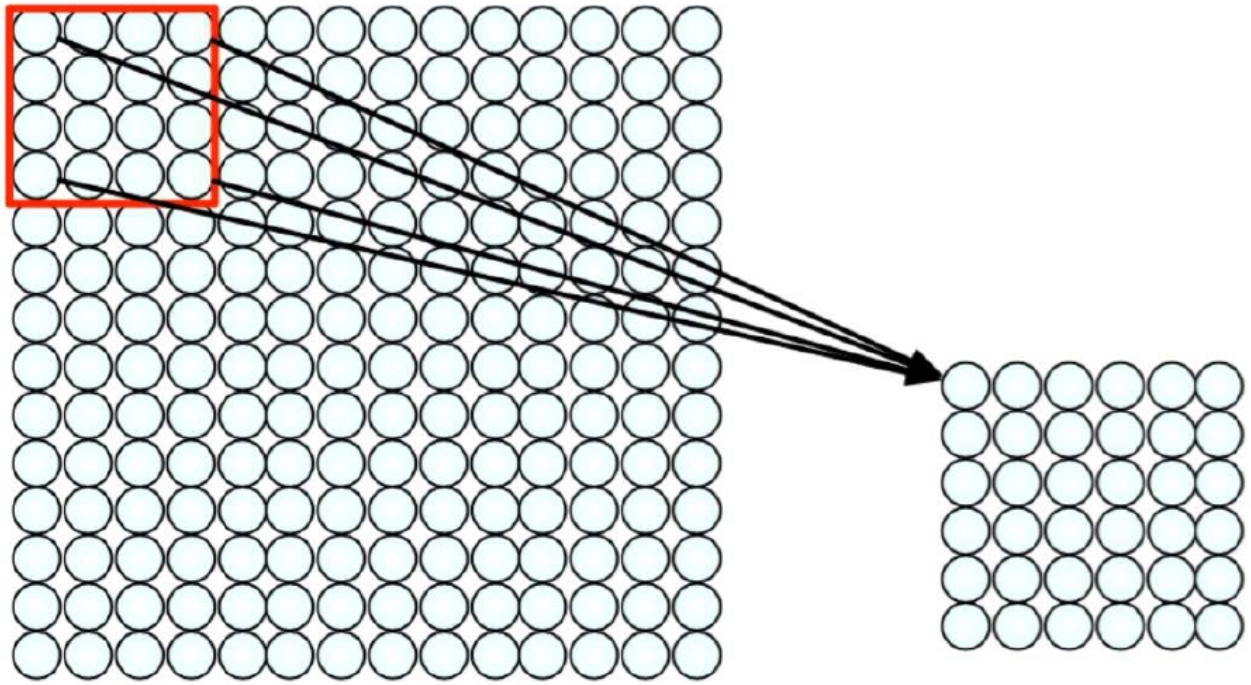
Convolutional Neural Networks
Building a CNN



1. **Convolution**: Apply filters with learned weights to generate feature maps.
2. **Non-linearity**: Often ReLU.
3. **Pooling**: Downsampling operation on each feature map.

Train model with image data.
Learn weights of filters in convolutional layers.

Convolutional Neural Networks
Building a CNN



4x4 filter: matrix of weights w_{ij}

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

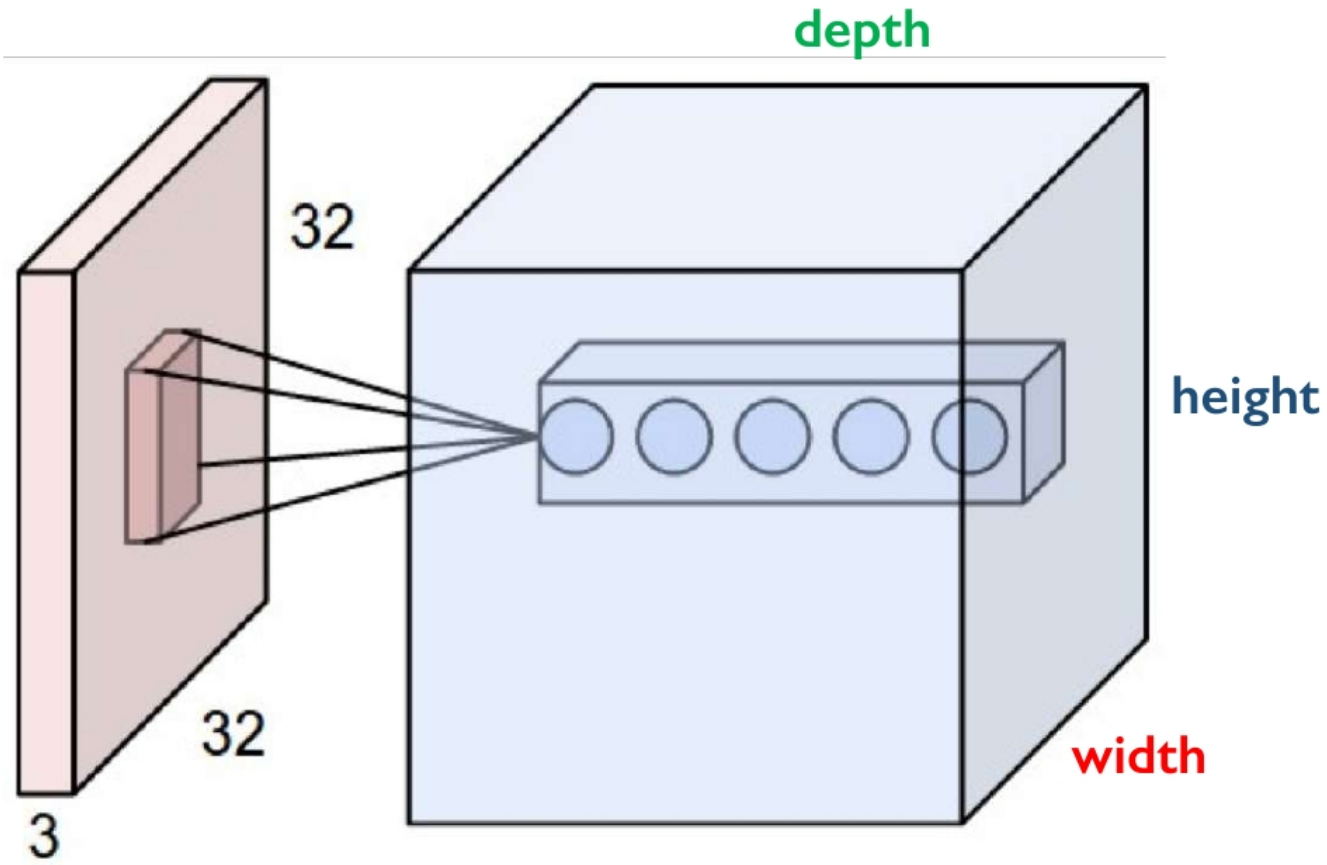
for neuron (p,q) in hidden layer

For a neuron in hidden layer:

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

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

Convolutional Neural Networks
Dimensionalities



Layer Dimensions:

$h \; x \; w \; x \; d$

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

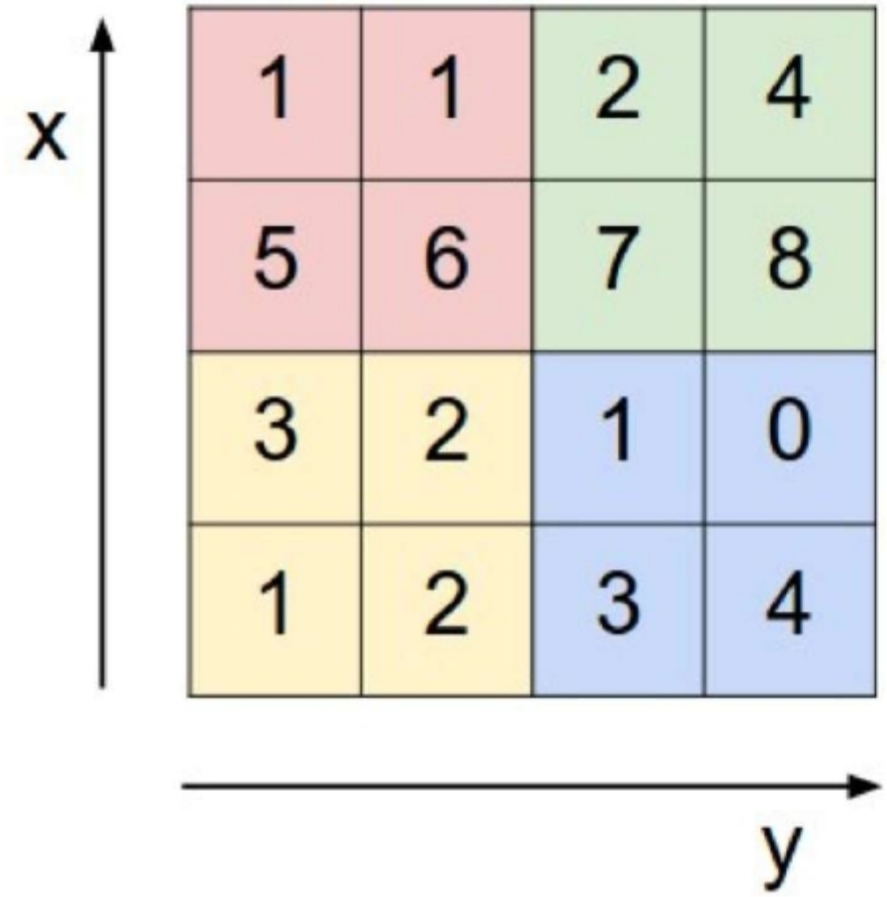
Stride:

Filter step size

Receptive Field:

Locations in input image that
a node is path connected to

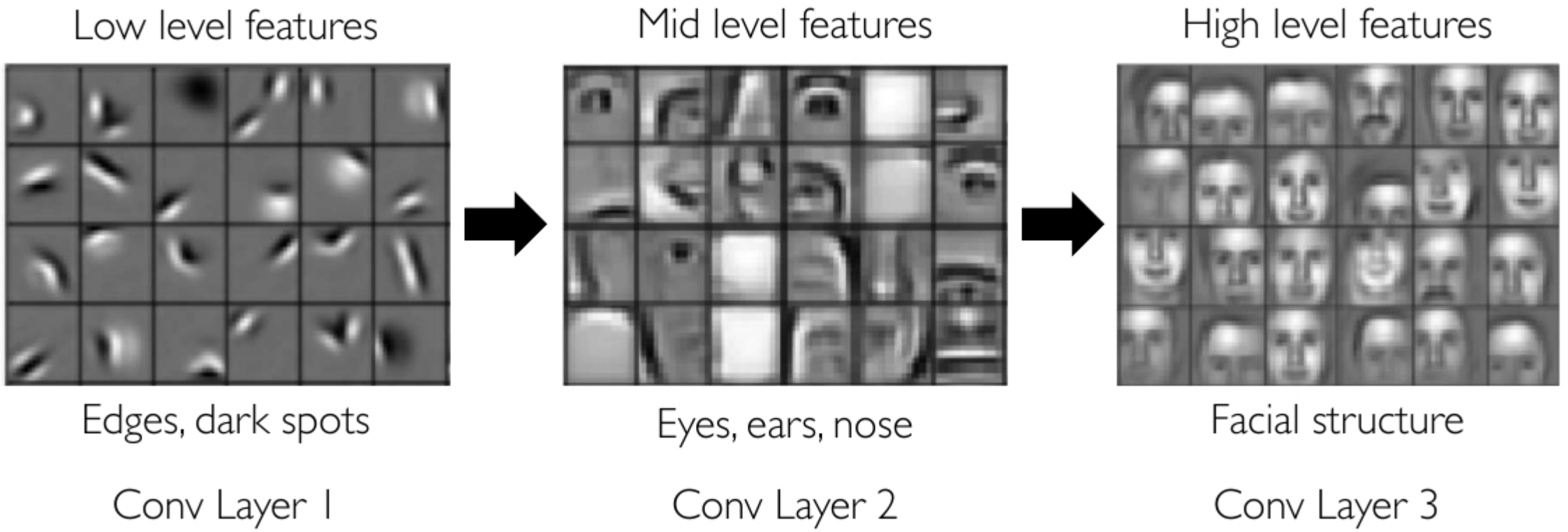
Convolutional Neural Networks
Pooling



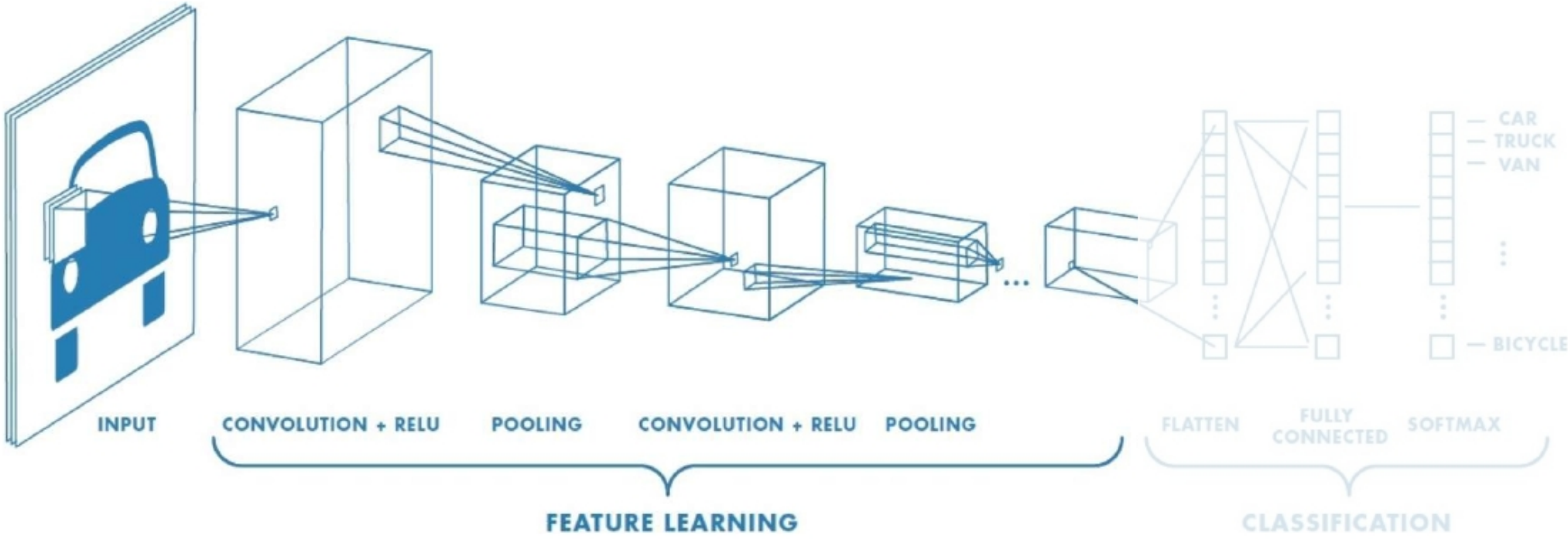
max pool with 2x2 filters
and stride 2

1) Reduced dimensionality
2) Spatial invariance

Convolutional Neural Networks
Representation learning

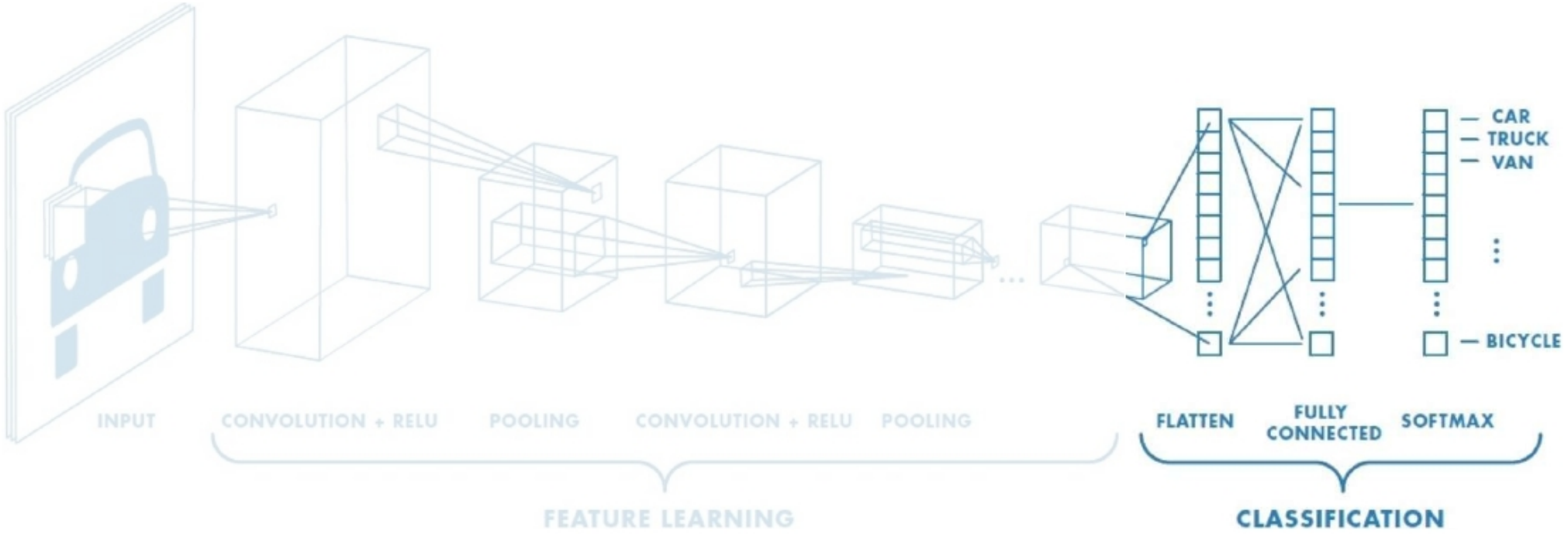


Convolutional Neural Networks
Feature learning



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

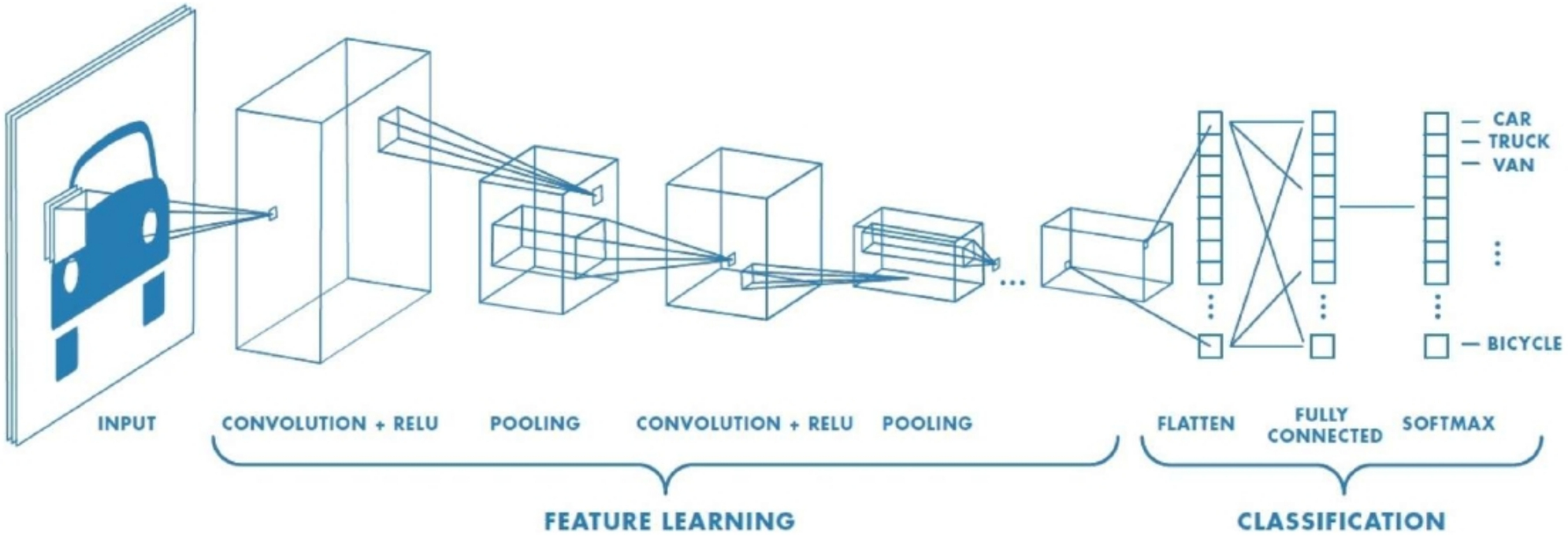
Convolutional Neural Networks
Using learned features for classification



- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

Convolutional Neural Networks
Training with backpropagation



Learn weights for convolutional filters and fully connected layers

Backpropagation: cross-entropy loss

$$J(\boldsymbol{\theta}) = \sum_i y^{(i)} \log\left(\hat{\boldsymbol{y}}^{(i)}\right)$$

Variants of Neural Networks

Convolution Neural Networks

Image Processing
Computer Vision

Recurrent Neural Networks

Time Series
Text
Audio

Autoencoder

Anomaly Detection
Data Compression
Image Generation

Curriculum

Next: III. Practical application

- Frameworks
- Pre-trained models
- Code & knowledge sources
- (Current research topics)

In B. Hands-on seminar – Try it out:

- Understand PyTorch’s Tensor library and neural networks at a high level
- Train a small neural network for regression analysis
- Train a small neural network to classify images