# INTRODUCTION TO DEEP LEARNING

## PART II – ADVANCED CONCEPT AND CNN

Auliya Fitri, Sai Vemuri, Sreerag Naveenachandran

Machine Learning Group
Institute of Data Science

DLR

# Schedule

| Date | Time | Activity |
|---|---|---|
| 13.11.2025<br>Day 1 | 09:00 - 10:00 | Introduction and basics |
| | 10:00 - 10:30 | Hands-on I |
| | 10:30 - 10:45 | Coffee break |
| | 10:45 - 11:45 | Advanced concept and Convolutional Neural Network |
| | 11:45 - 12:15 | Hands-on II |
| | 12:15 - 12:30 | Recap Day 1 |
| 14.11.2025<br>Day 2 | 09:00 - 10:00 | Deep Generative Models |
| | 10:00 - 10:30 | Hands-on III |
| | 10:30 - 10:45 | Coffee break |
| | 10:45 - 11:45 | Transformers, LLM, and other interesting architectures |
| | 11:45 - 12:15 | Hands-on IV |
| | 12:15 - 12:30 | Code and knowledge sources + closing |

We're here!

# Curriculum

**II. Advanced Concept and Convolutional Neural Network**

- Regularization
- Variants of Neural Networks
- Convolutional Neural Networks (CNN)

*Inspired by lectures from Paris Saclay and MIT; images taken from these, if not noted otherwise*

# Neural network concepts

**Building the network**

- Input layer
- Hidden layers
- Output layer

**Forward propagate input**

- Propagate input through neurons
- Apply non-linearities

**Evaluate performance**

- Compute the loss of the prediction

**Back propagate**

- Update weights according to the optimization algorithm

**Improving the network**

- Regularization,
- Advanced architectures

# Improve the Network: Regularization

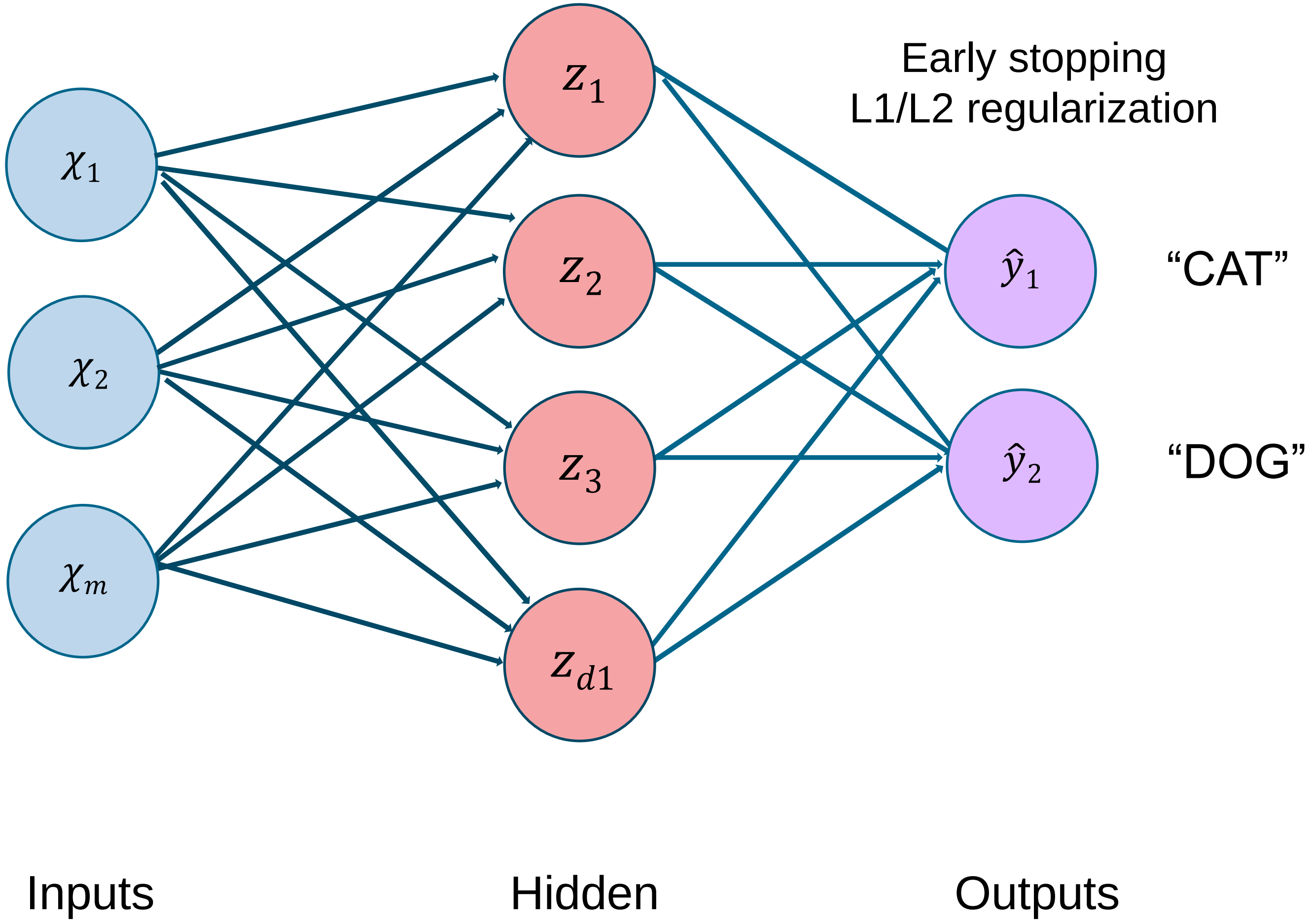
Data Augmentation

CAT

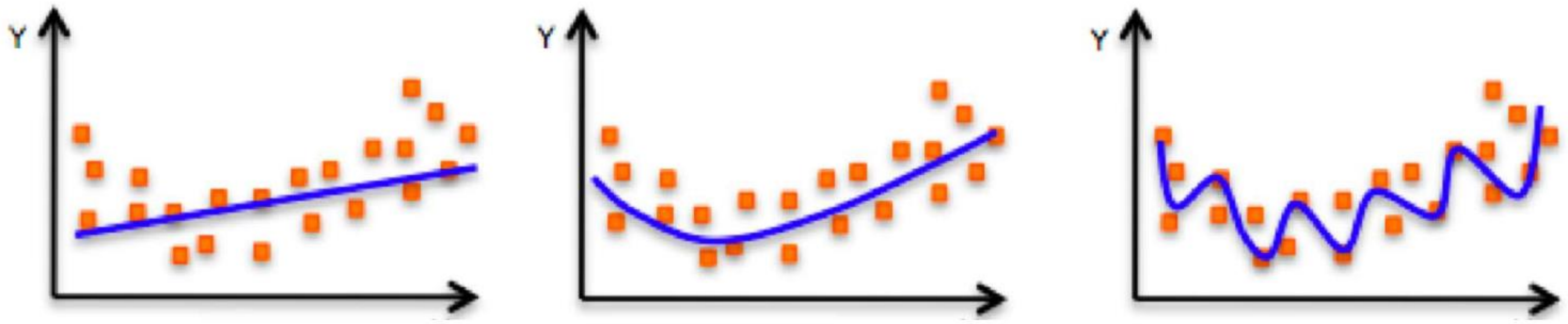
$$\begin{bmatrix} \chi_1 \\ \chi_2 \\ \chi_m \end{bmatrix}$$

Dropout

Early stopping
L1/L2 regularization



# Regularization
## The overfitting problem




**Underfitting**
Model does not have capacity to fully learn the data

**Ideal fit**

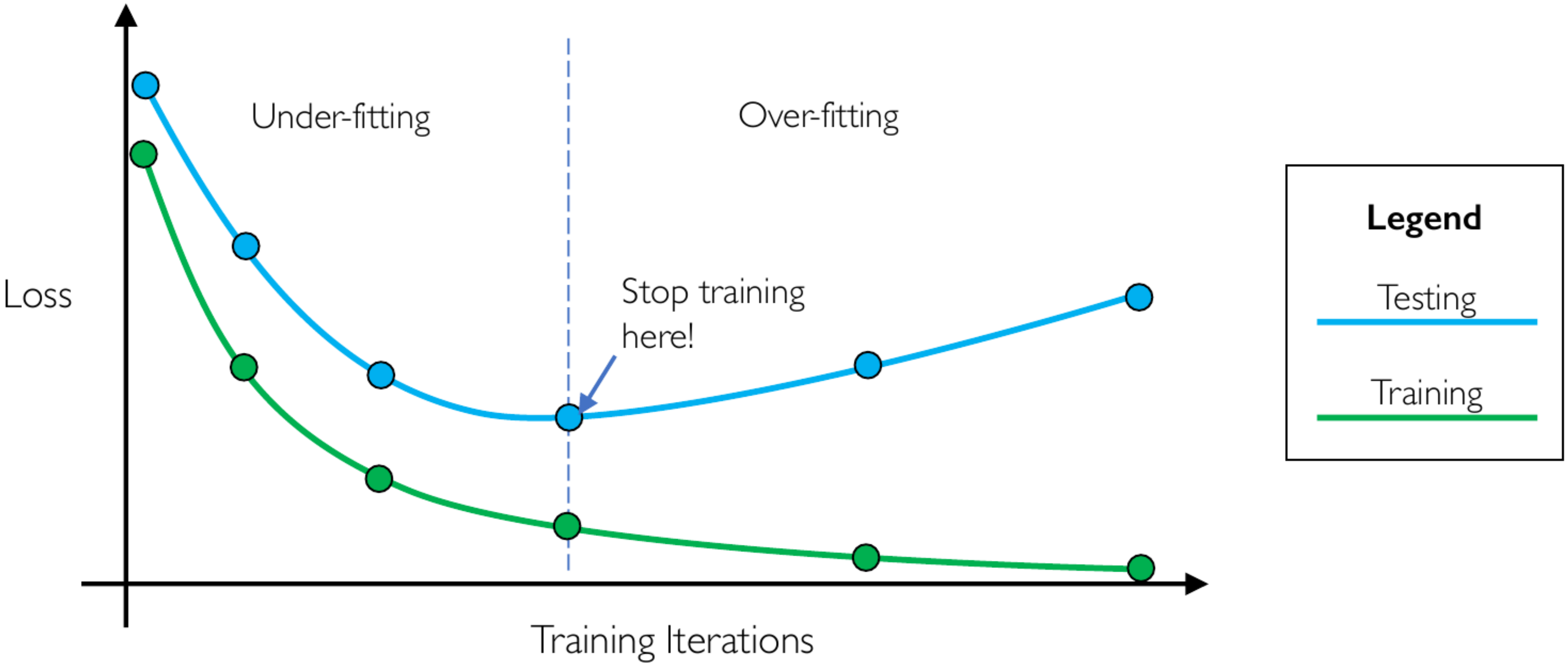
**Overfitting**
Too complex, extra parameters, does not generalize well

= High bias

**Tradeoff**

= High variance

# Regularization
## Preventing overfitting: Early stopping



# Regularization

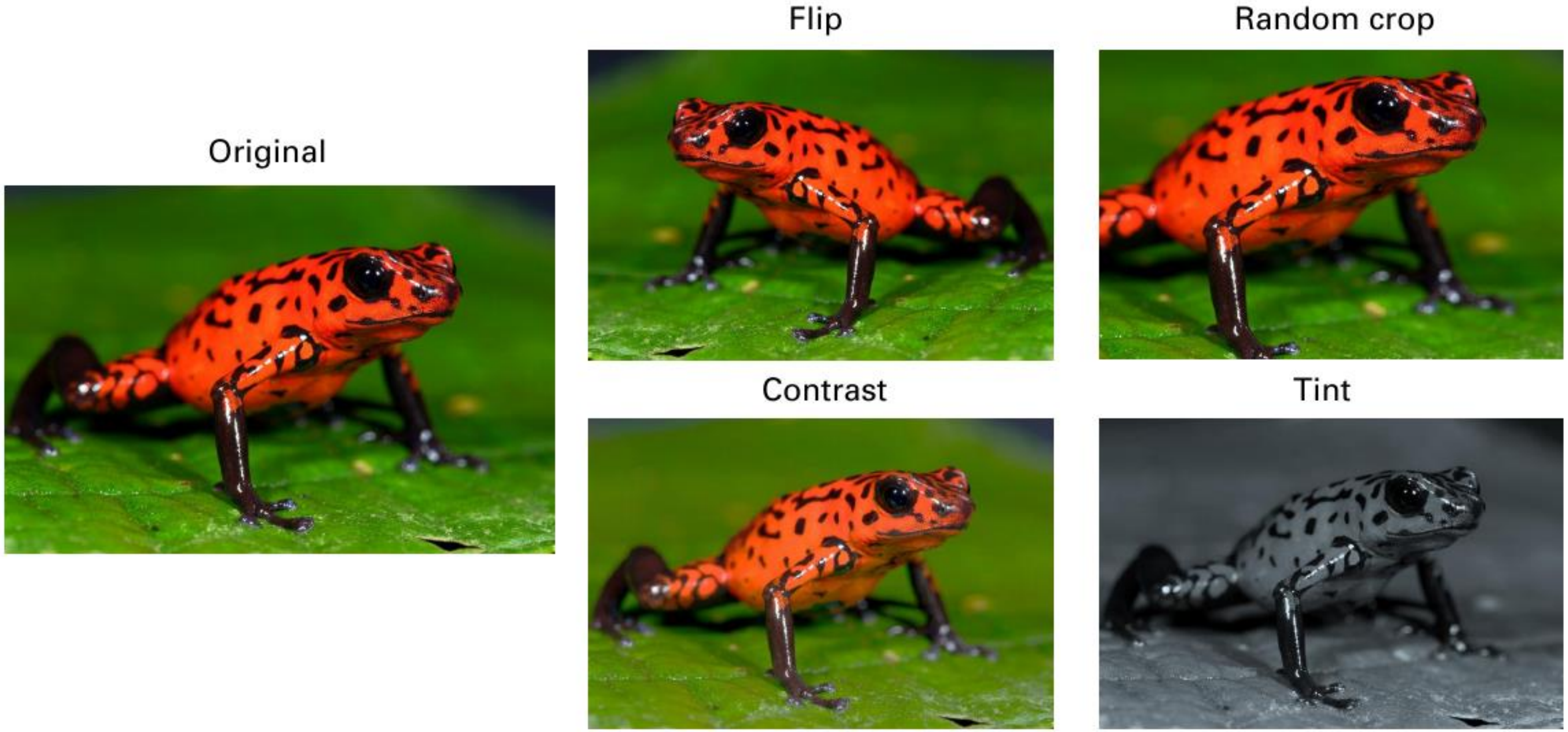
## Preventing overfitting: $l_1$ regularization

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

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

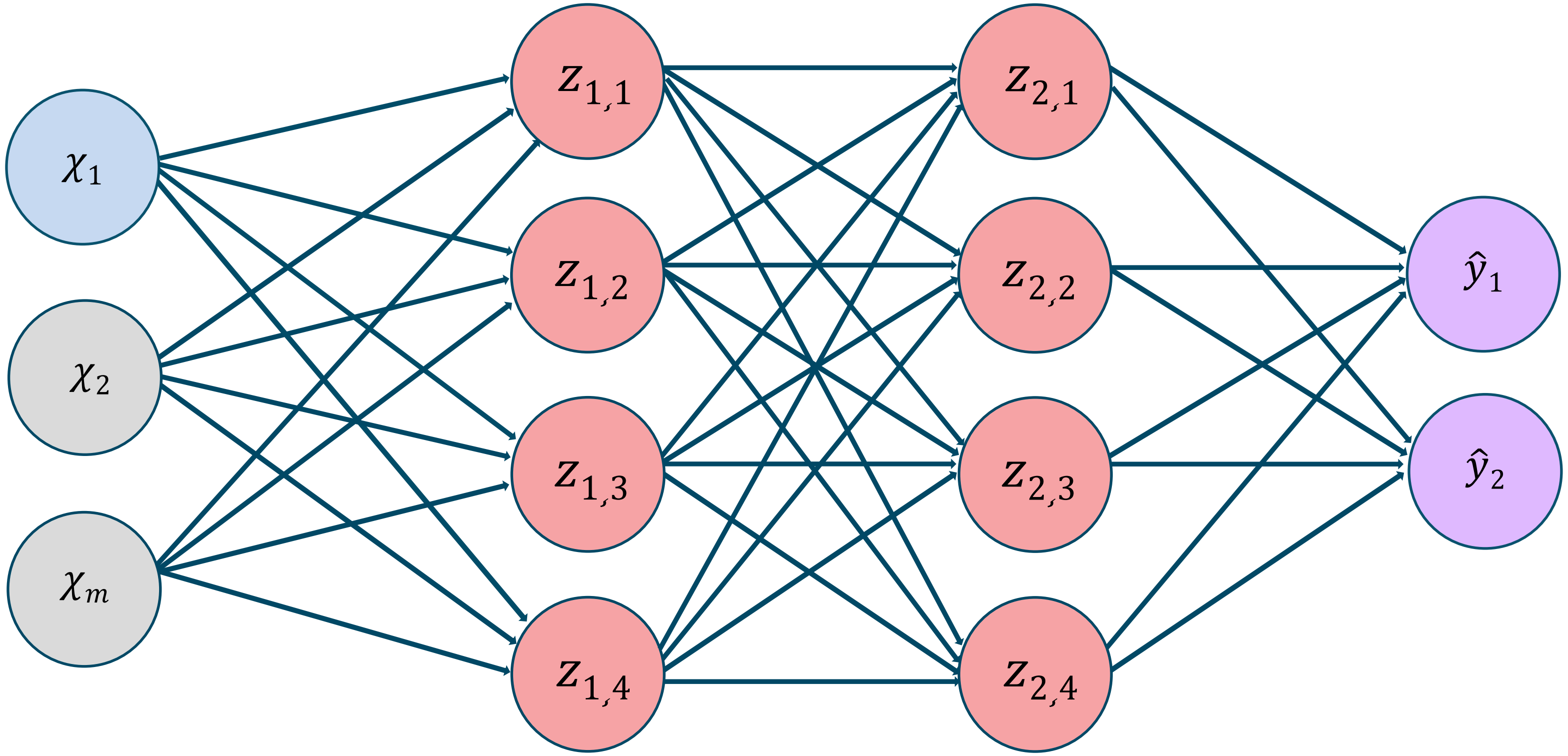
→ Leads to sparser weights (more zeroes in weights) that are not too adapted to the data at hand

# Regularization
## Preventing overfitting: Data augmentation



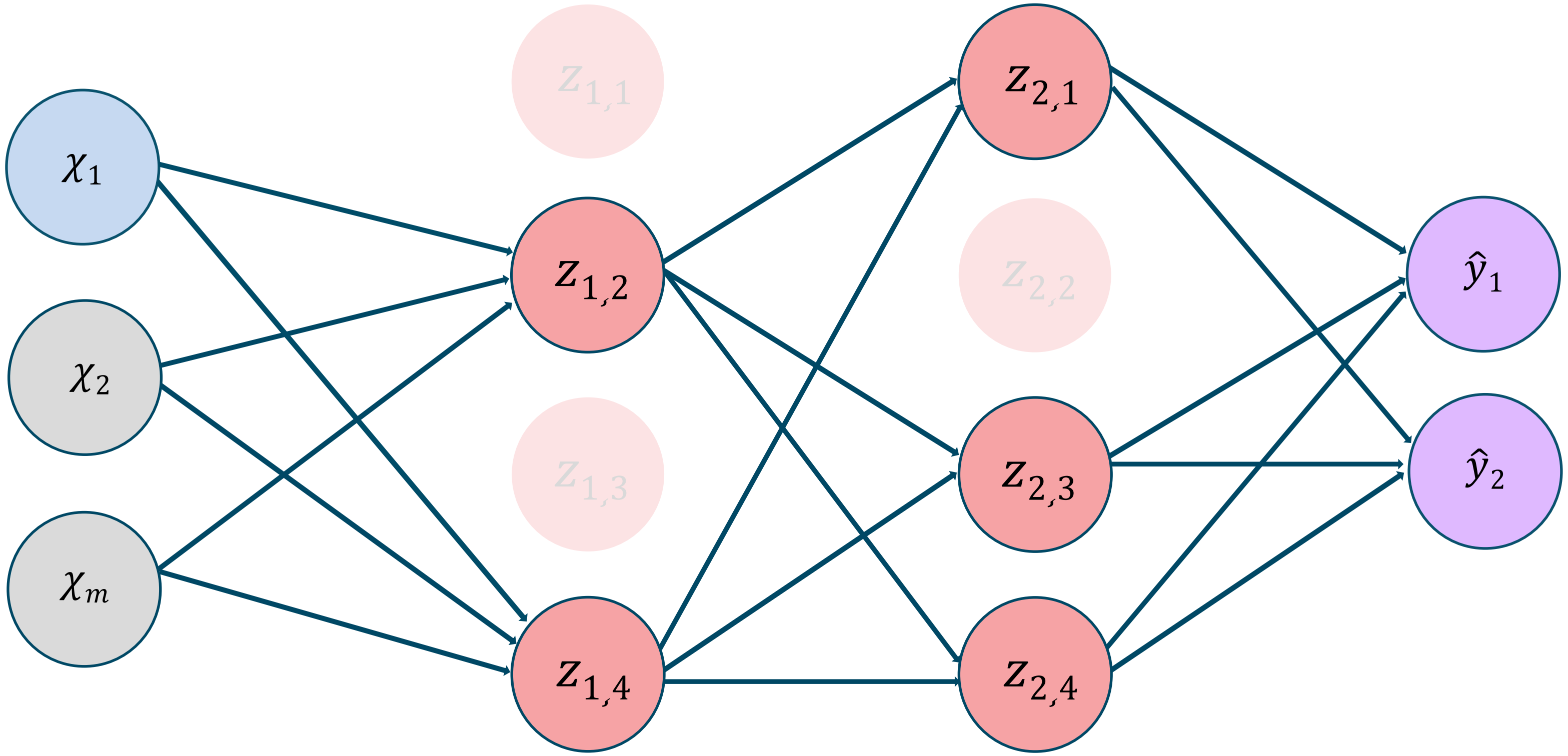
# Regularization
## Preventing overfitting: Dropout

During training, randomly set some activations to 0



# Regularization
## Preventing overfitting: Dropout

During training, randomly set some activations to 0



# Other ways to improve the network

**Data preprocessing**

*Data Cleaning (lack/noise)*

*Data Transformation (normalization, …)*

*Data Reduction (aggregation, …)*

**Network initialization**

*Random Initializations (e.g. Glorot)*

**Batch normalization**

*standardizes layer inputs to stabilize learning process & reducing training epochs*

**Optimizers**

*(Stochastic methods like Adam)*

**Training Schedules**

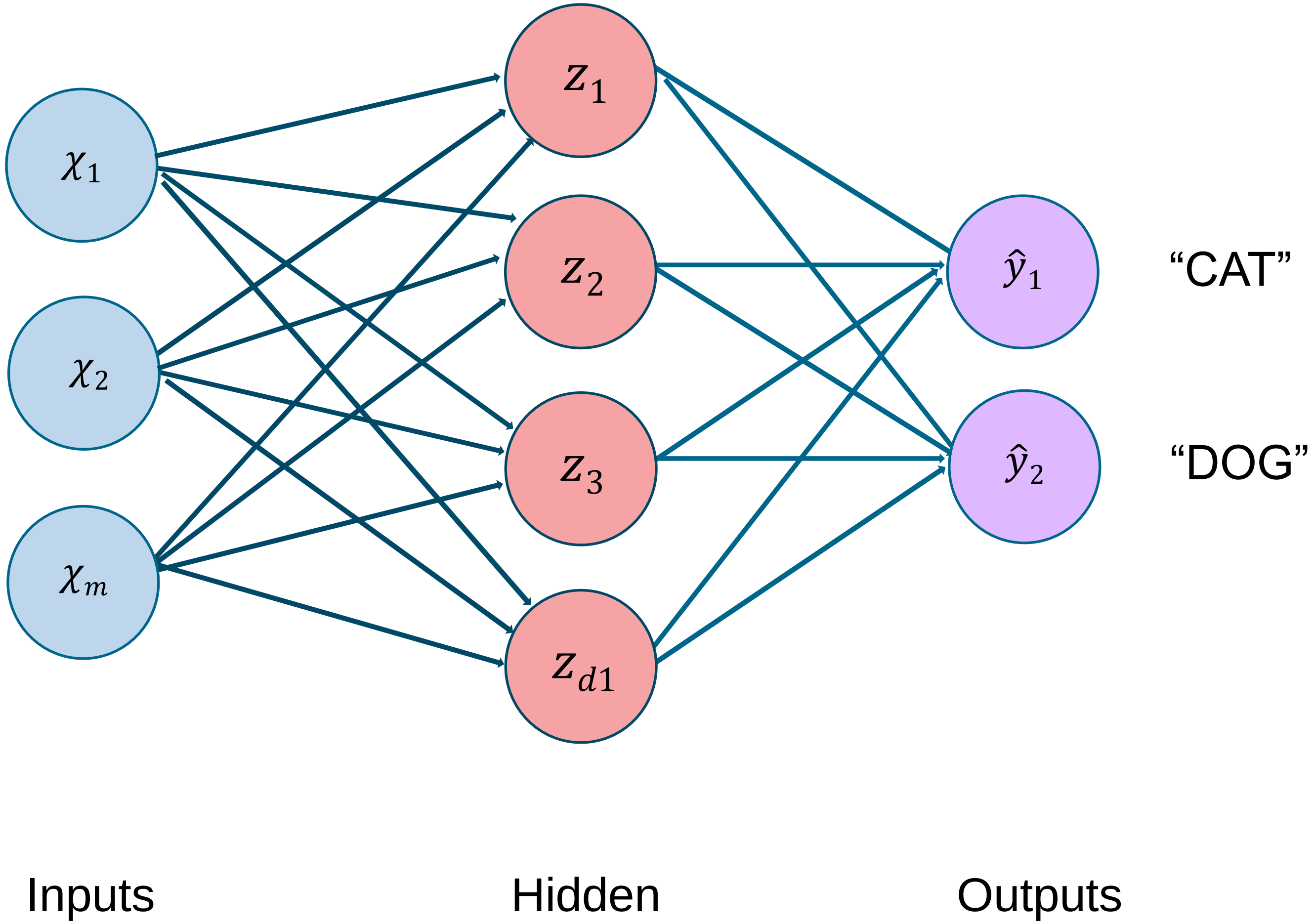
**Hyperparameters tuning**

**Minibatch composition**

*(shuffle data, take only a subset, … )*

# Variants of Neural Networks


$$\begin{bmatrix} \chi_1 \\ \chi_2 \\ \chi_m \end{bmatrix}$$

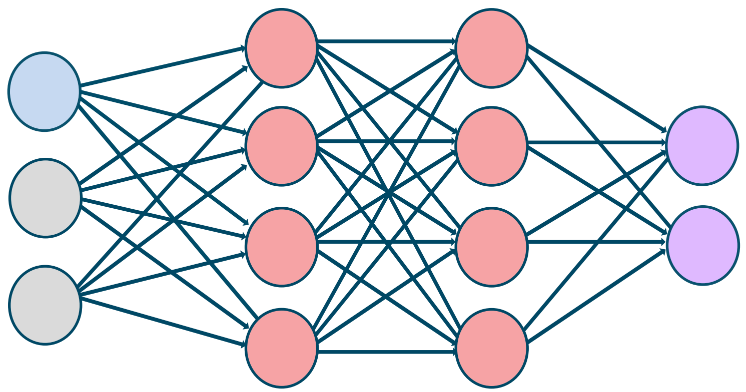


# Types of Neural Networks Layer

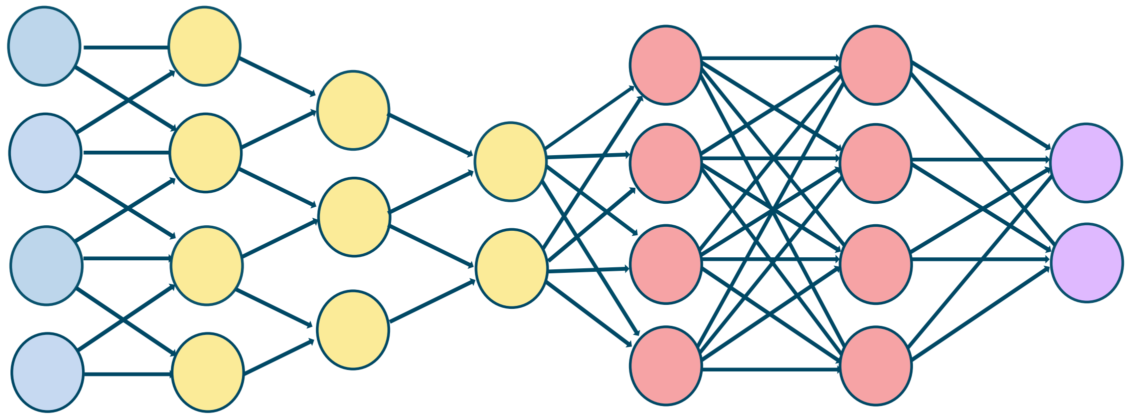
| Application | Layer Type | | | |
|---|---|---|---|---|
| | Fully Connected (dense) | Convolution | Deconvolution | Recurrent |
| Image Classification | ✓ | ✓ | | |
| Image segmentation | | ✓ | ✓ | |
| Text processing | ✓ | | | ✓ |
| Speech recognition | ✓ | | | ✓ |
| Time series | ✓ | | | ✓ |
| Image-to-image translation (GAN) | ✓ | ✓ | ✓ | |
| Autoencoders | ✓ | ✓ | | |
| Deepfake | ✓ | ✓ | | ✓ |

# Variants of Neural Network - Examples

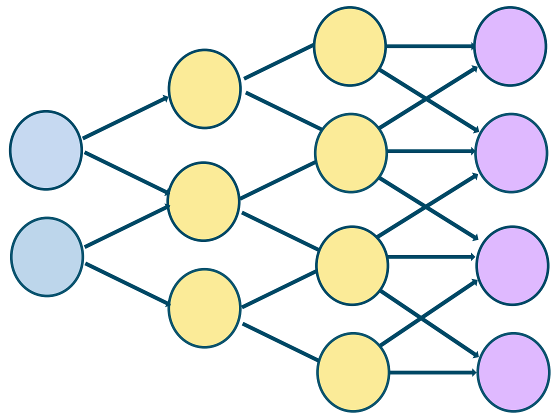
Deep Feed Forward (DFF) (fully connected)



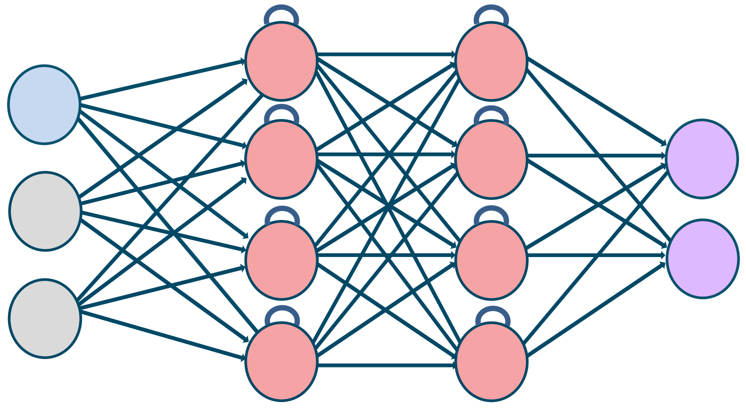
Deep convolution network (DCN)

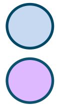


Deconvolution network (DN)

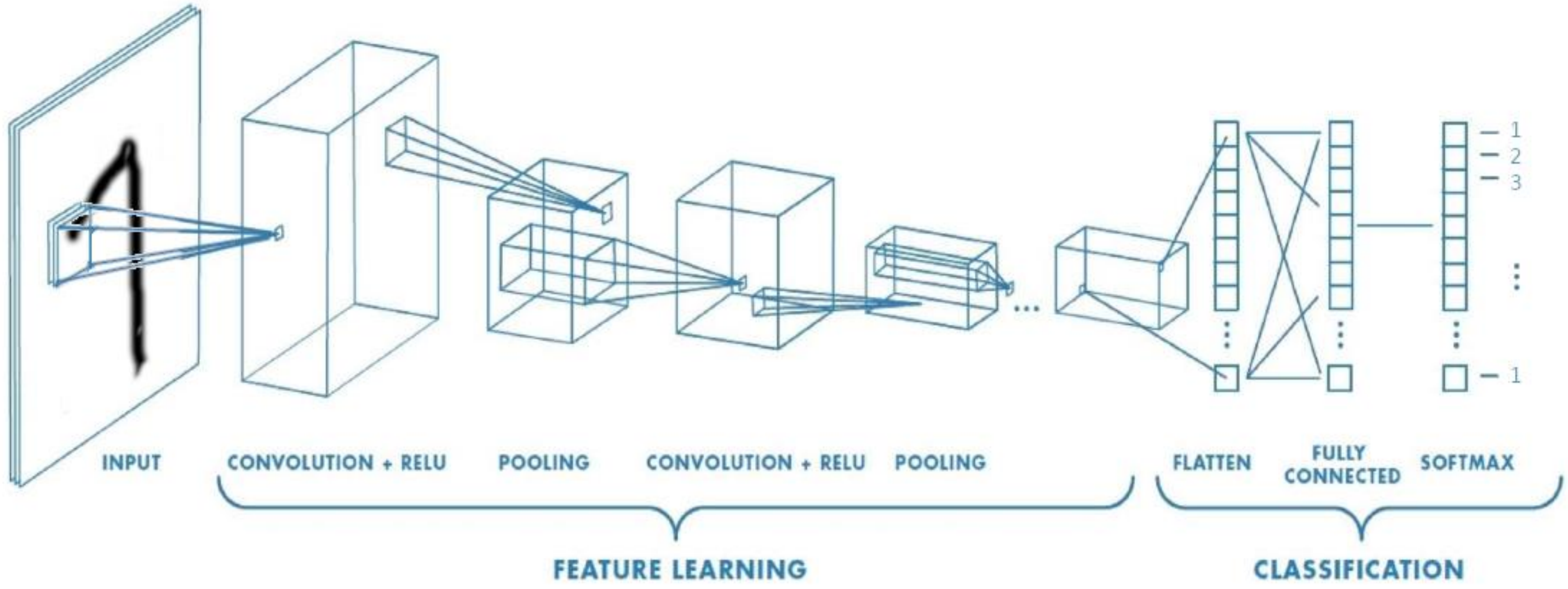


Recurrent neural network (RNN)

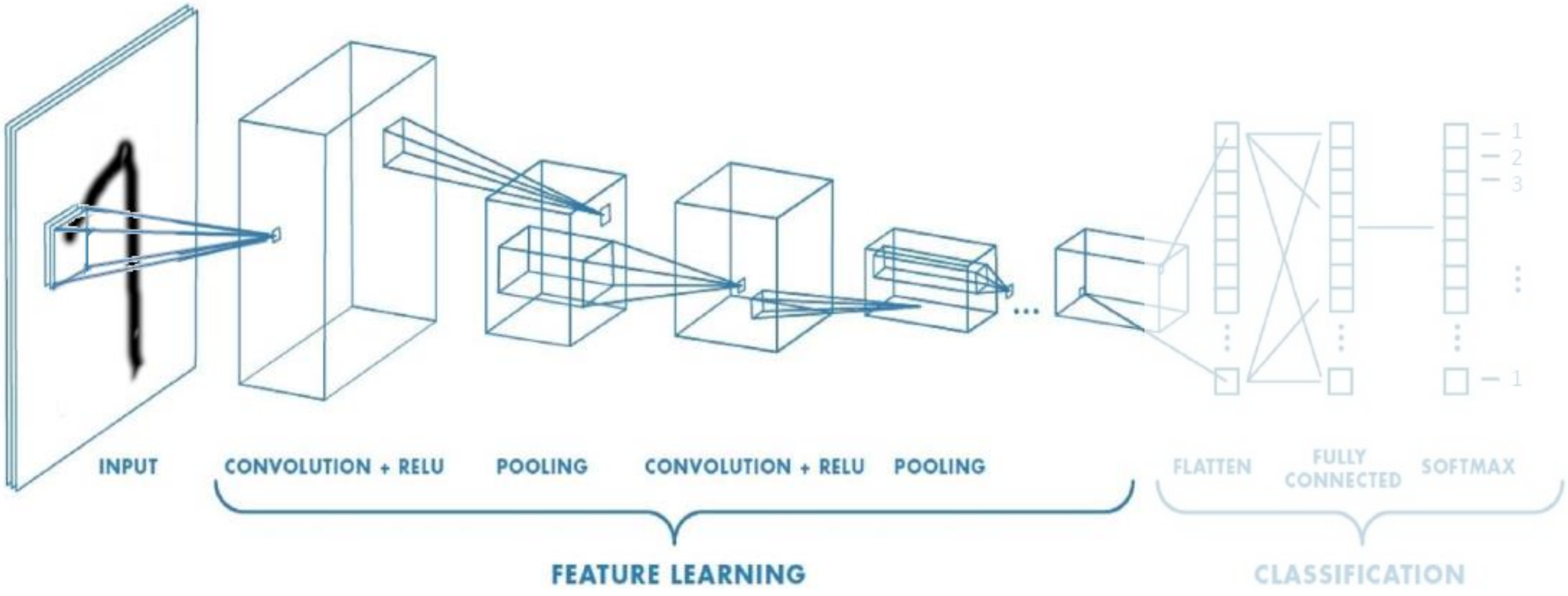




# Convolutional Neural Networks



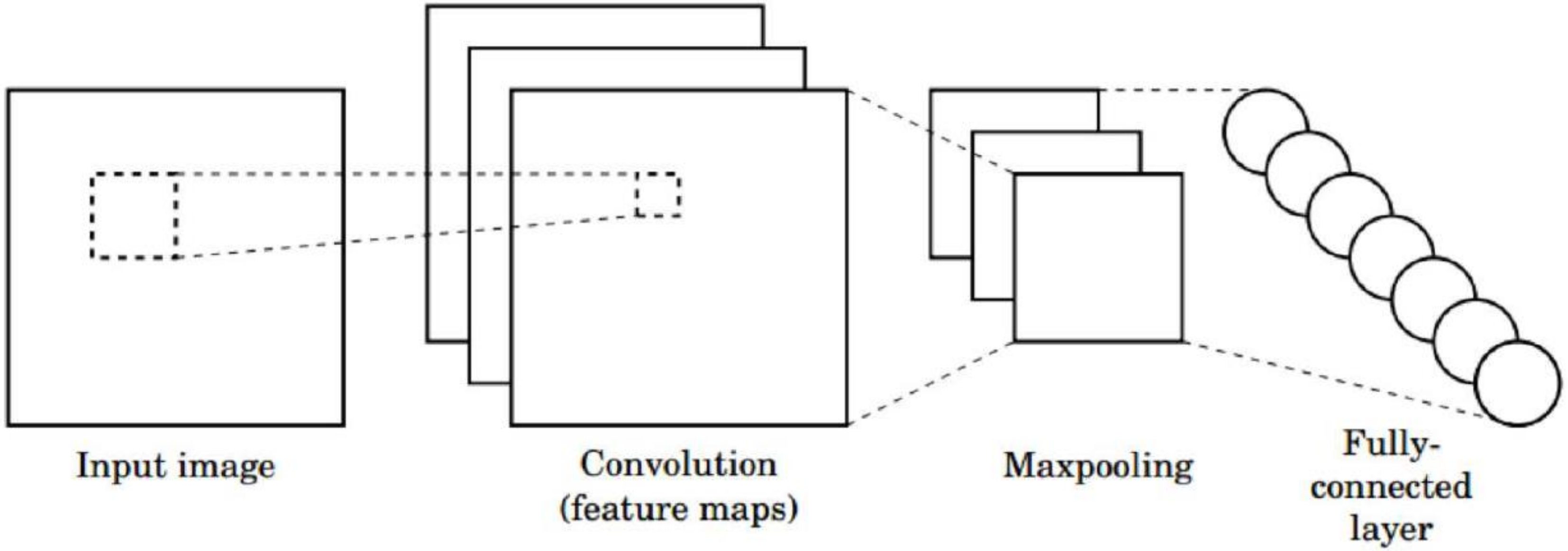
# Feature Learning




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
## Feature extraction with convolutions

# Convolutional Neural Networks
## Building a CNN



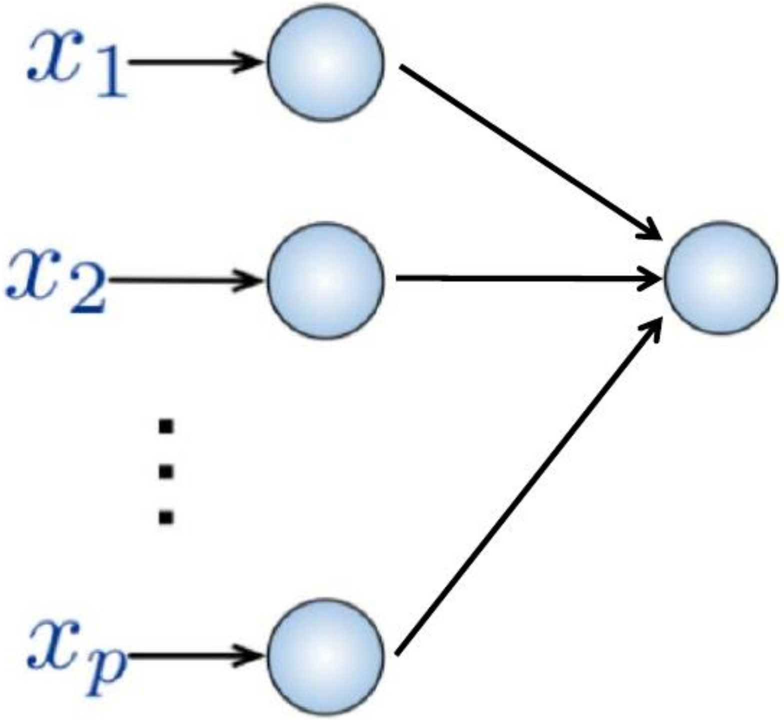

1. **Convolution**: Apply filters with learned weights to generate feature maps.
2. **Non-linearity**: Often ReLU.
3. **Pooling**: Downsampling operation on each feature map.

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

# Convolutional Neural Networks
## Learning on image data

**Input:**

- 2D image
- Vector of pixel values

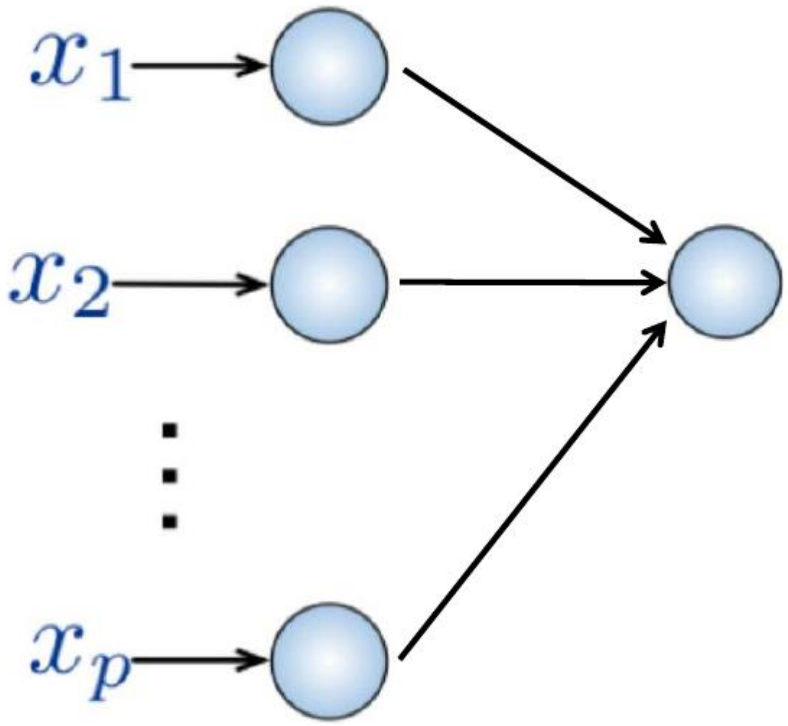



**Fully Connected:**

- 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

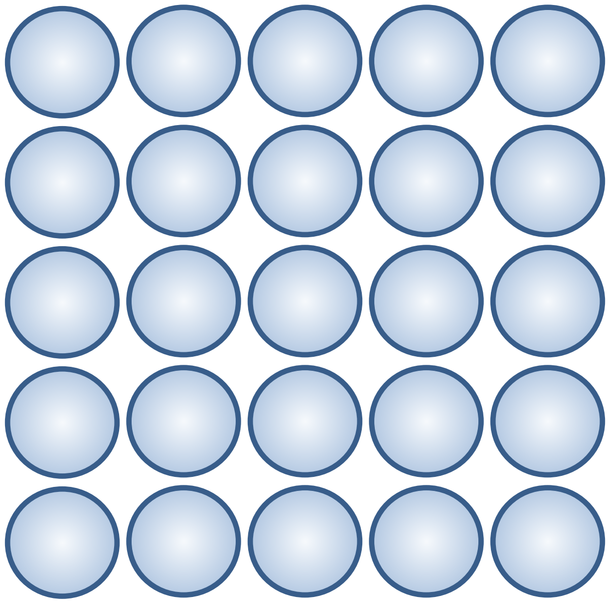
**Input:**
- 2D image
- Vector of pixel values




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

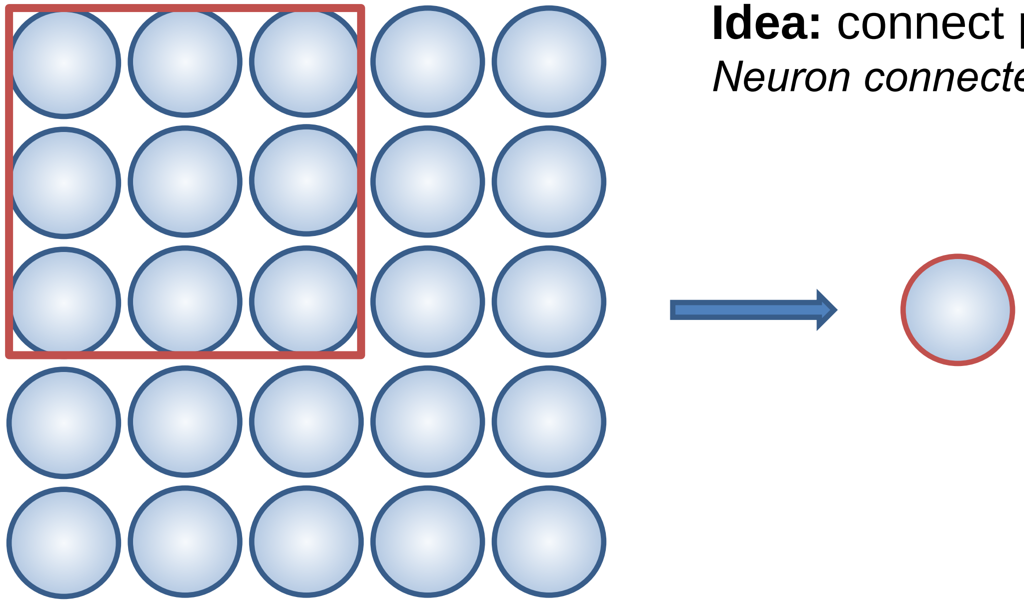
How can we use **spatial structure** in the input to inform the architecture of the network?

# Convolutional Neural Networks
## Learning on image data



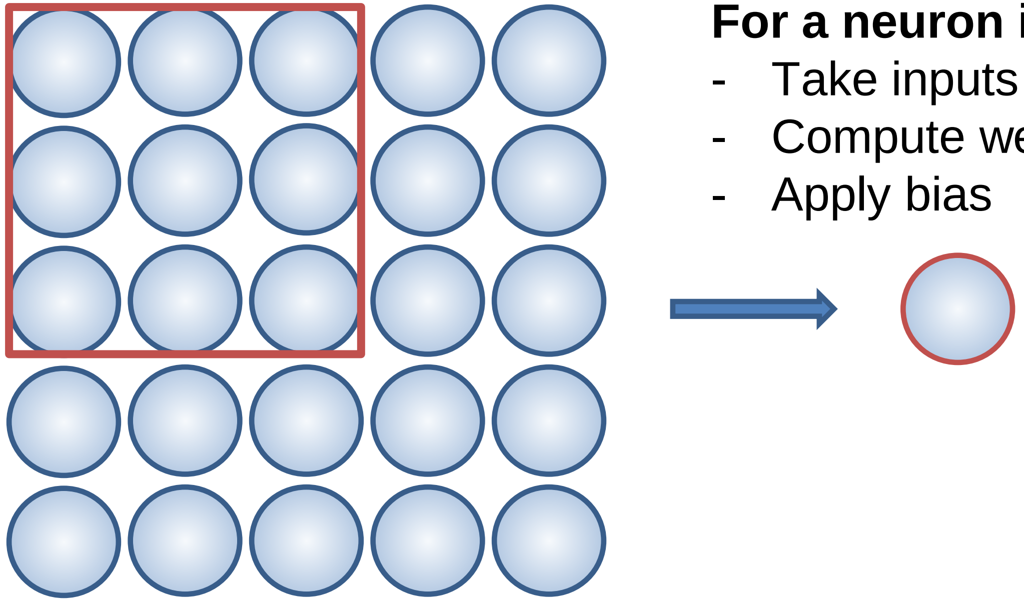
**Input:** 2D image (5x5 or 256x256 or …)
Array of pixel values

# Convolutional Neural Networks
## Learning on image data



**Idea:** connect patches of input to neurons in hidden layer
*Neuron connected to the region of input only „sees“ these values.*

# Convolutional Neural Networks
## Learning on image data



3x3 filter: matrix of weights $W_{ij}$

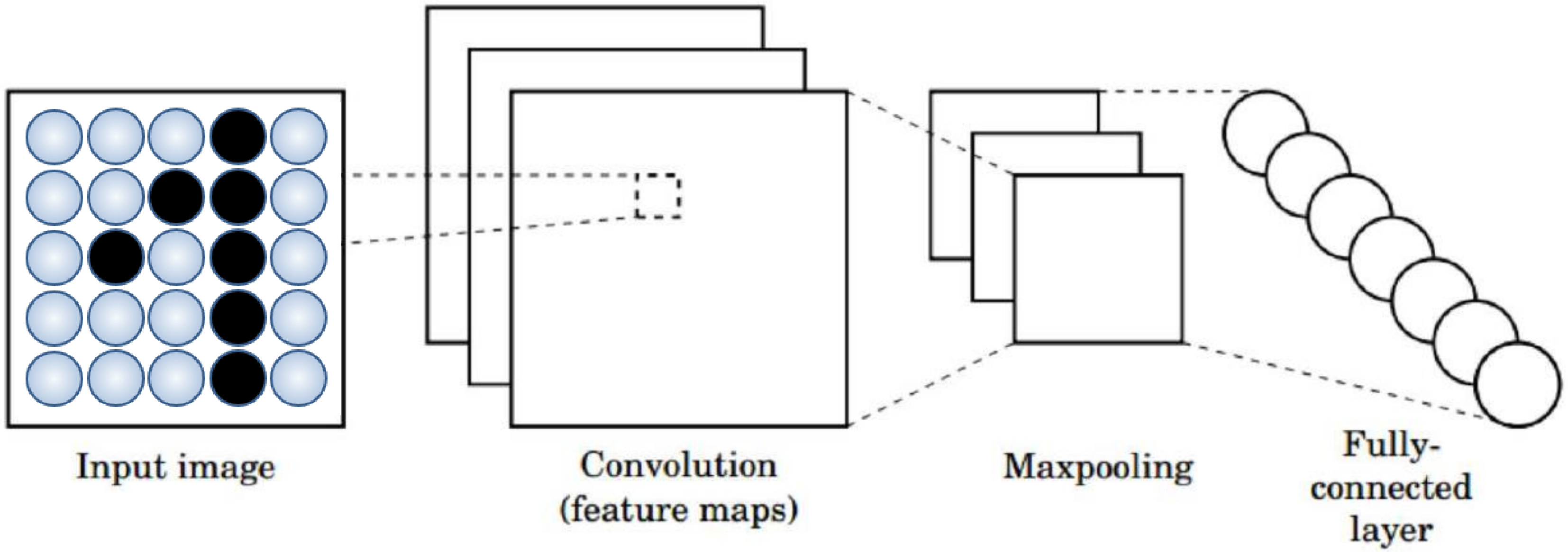
**For a neuron in hidden layer:**
- Take inputs from patch
- Compute weighted sum
- Apply bias

$$\sum_{i=1}^{3}\sum_{j=1}^{3} W_{ij}\, X_{i+p,j+q} + b$$

For neuron (p,q) in hidden layer

1) Applying a window of weights
2) Computing linear combinations
3) Activating with non-linear function

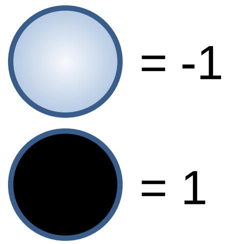
# Convolutional Neural Networks
## Building a CNN - Example




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

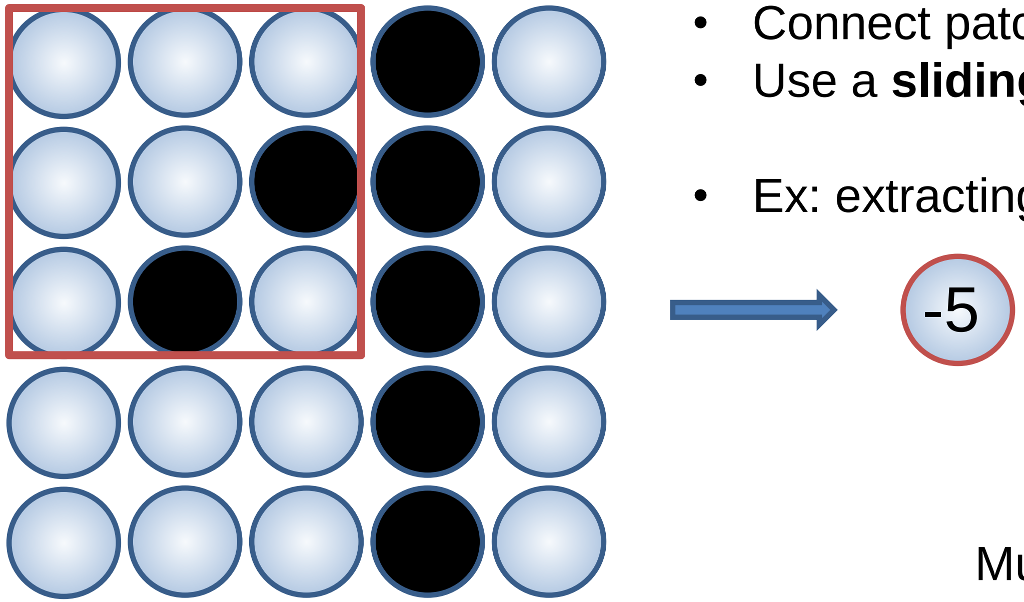
**Train model with image data.**
**Learn weights of filters in convolutional layers.**

# Convolutional Neural Networks
## Learning on image data - Example




- Connect patch in input layer to a single neuron in subsequent layer
- Use a **sliding window** to define connections

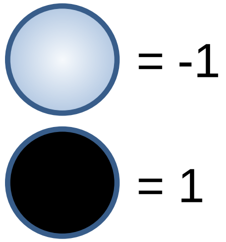
- Ex: extracting feature with filter $fi_0$



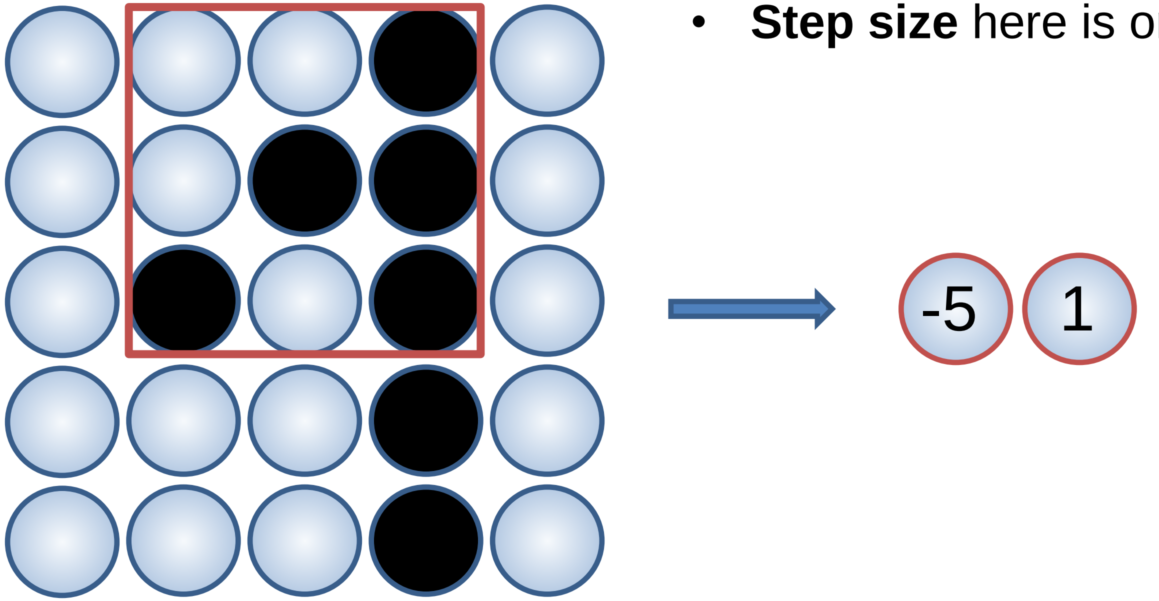

Multiply same indices of clipping and filter and sum it up.

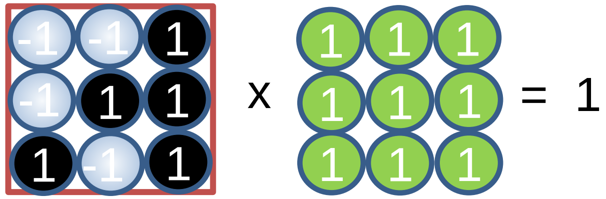
$$\begin{bmatrix}-1&-1&-1\\-1&-1&1\\-1&1&-1\end{bmatrix} \times \begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} = (-1)+(-1)+(-1)+(-1)+(-1)+1+(-1)+1+(-1) = -5$$

# Convolutional Neural Networks
## Learning on image data - Example

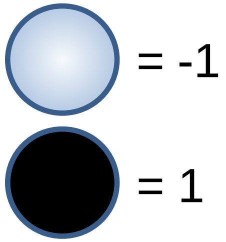


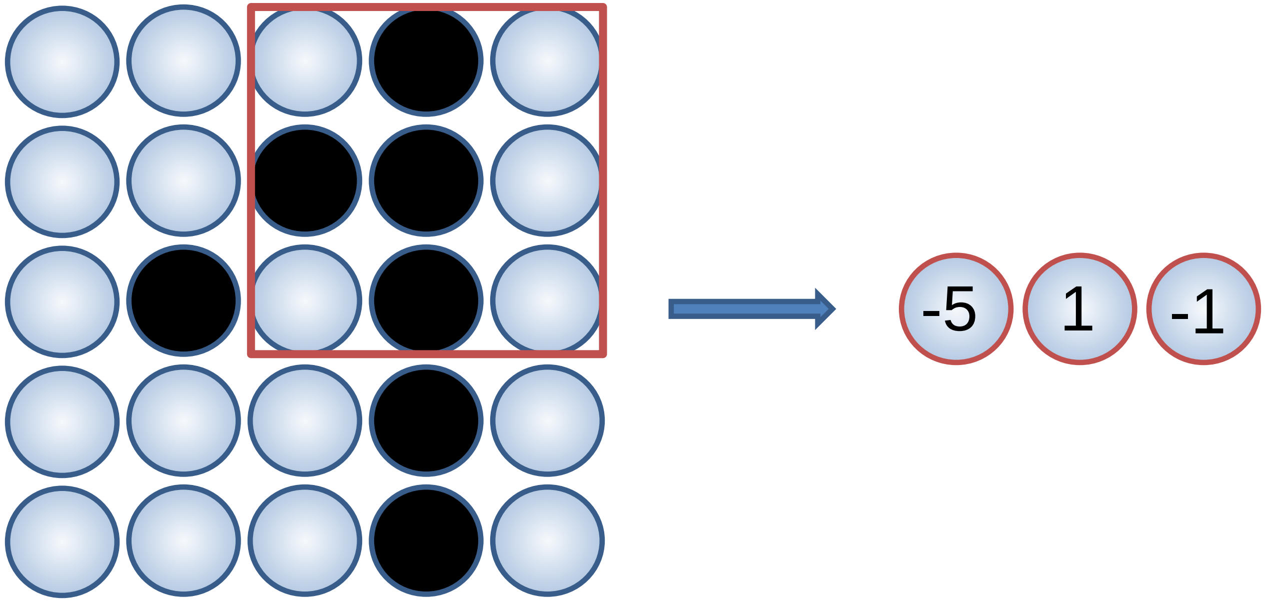

- **Step size** here is one and the clipping is 3x3

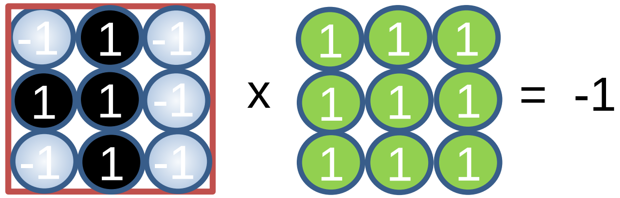


# Convolutional Neural Networks
## Learning on image data - Example

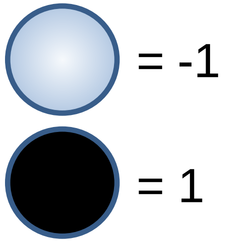


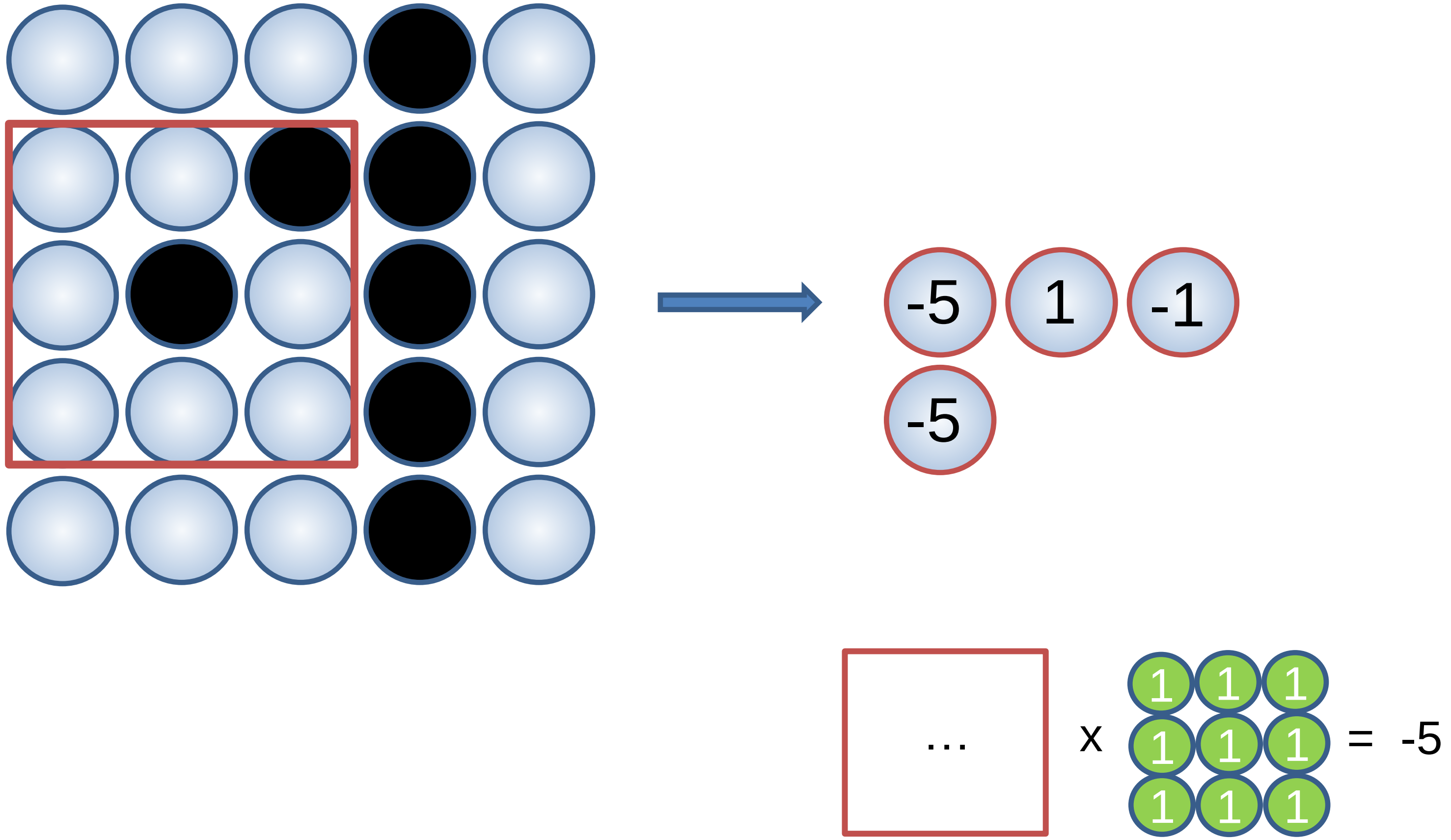



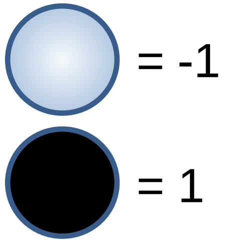
# Convolutional Neural Networks

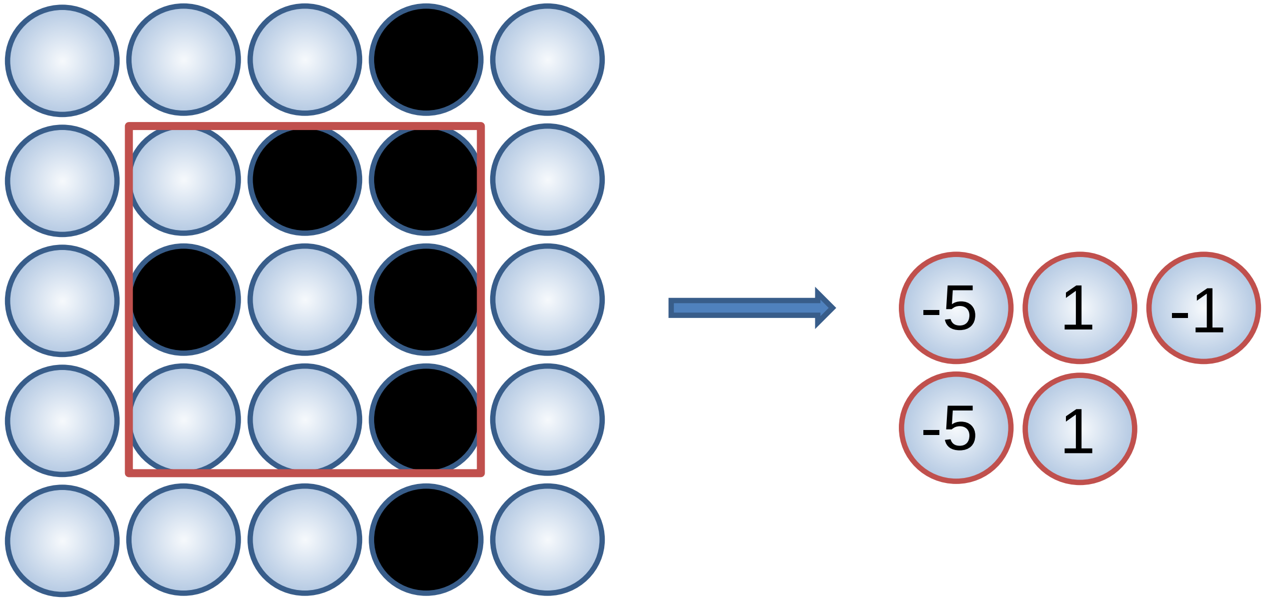
## Learning on image data - Example

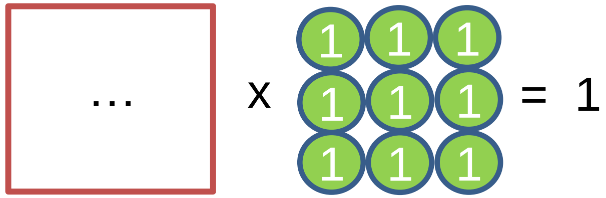


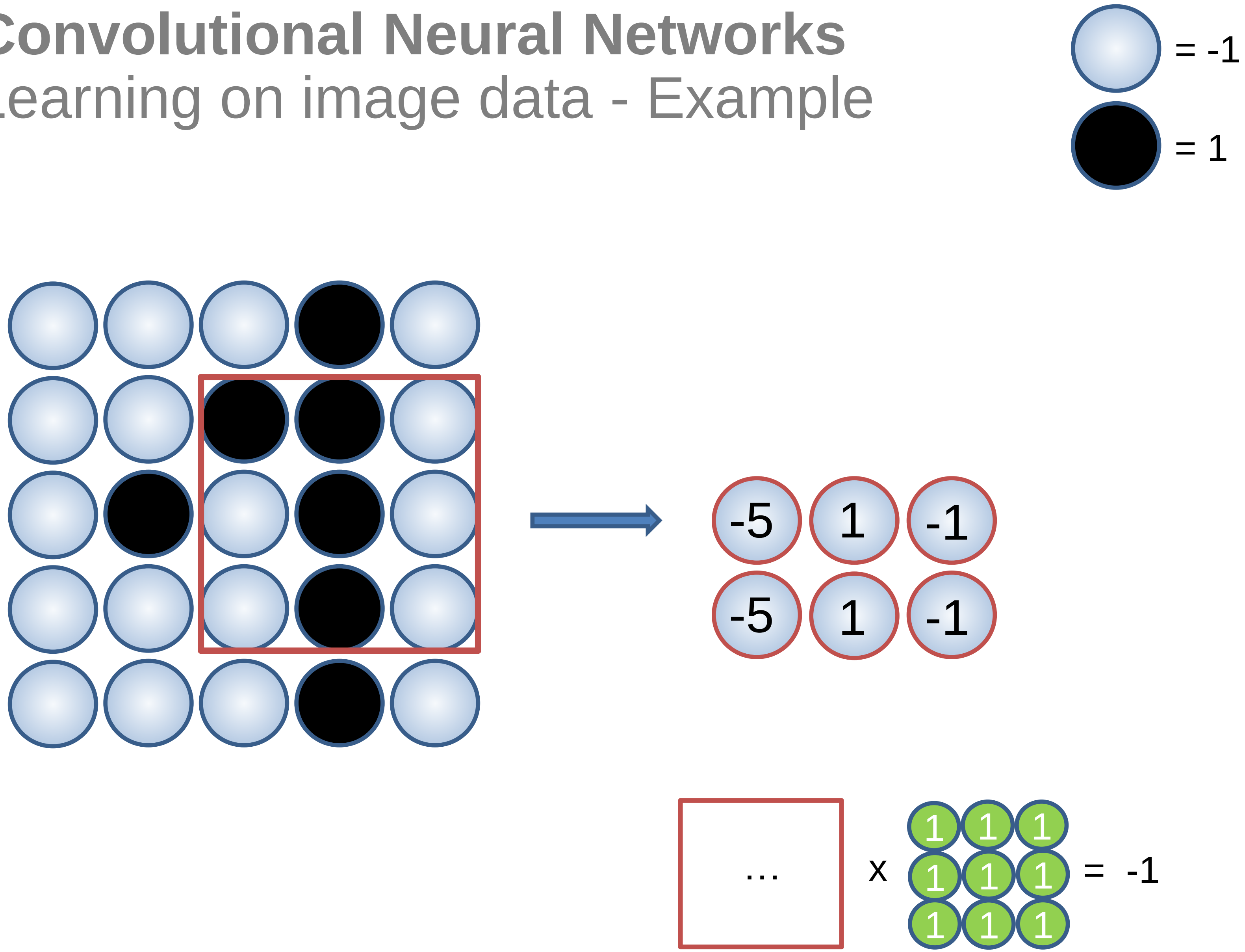
# Convolutional Neural Networks
## Learning on image data - Example



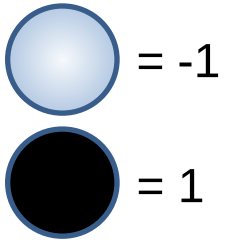



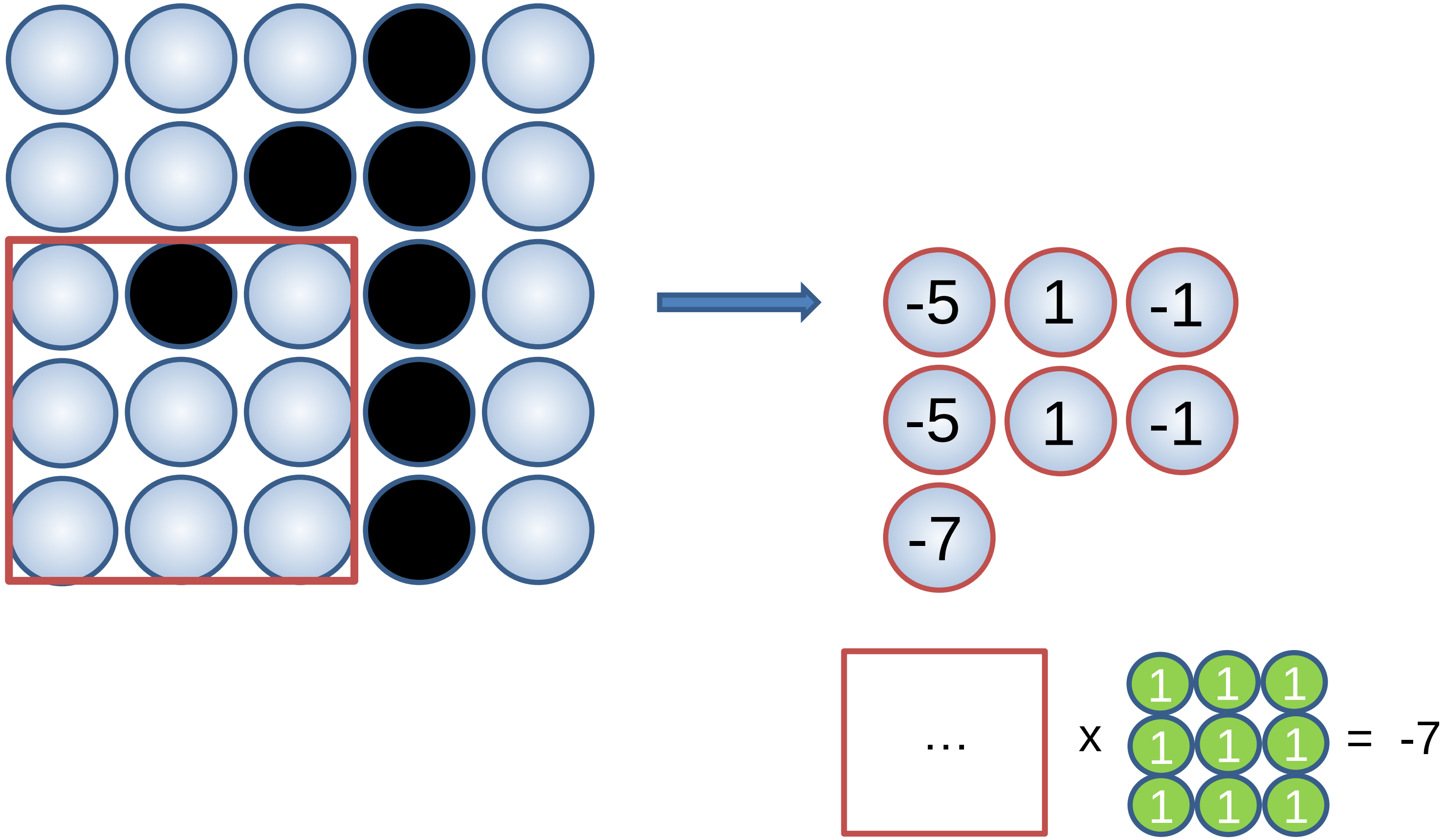



# Convolutional Neural Networks
## Learning on image data - Example

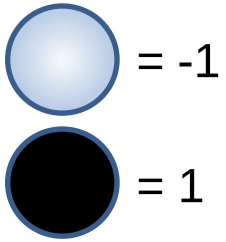


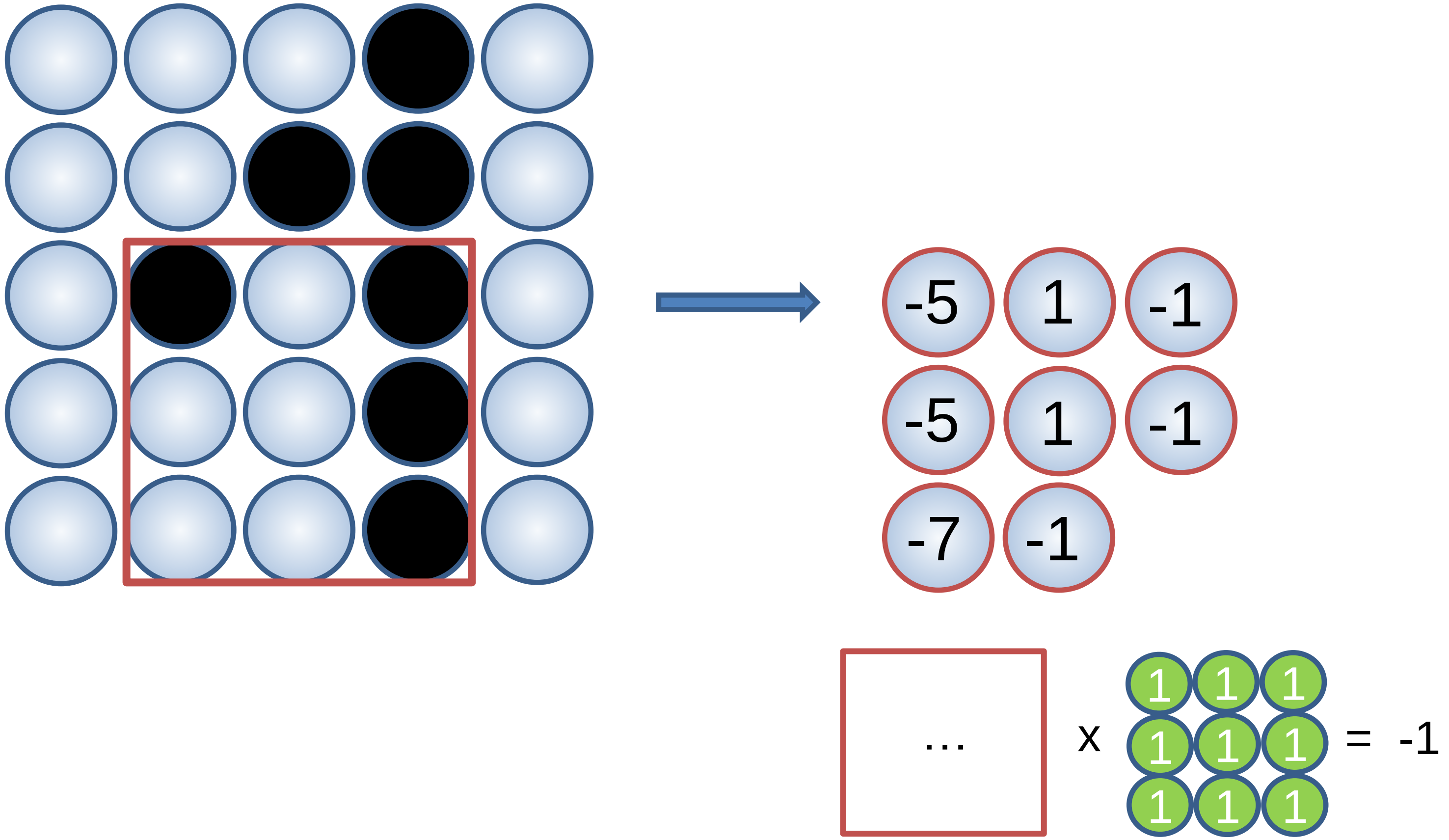
# Convolutional Neural Networks
## Learning on image data - Example



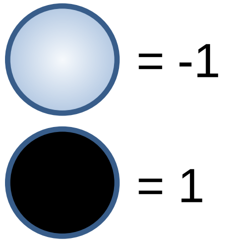
# Convolutional Neural Networks
## Learning on image data - Example



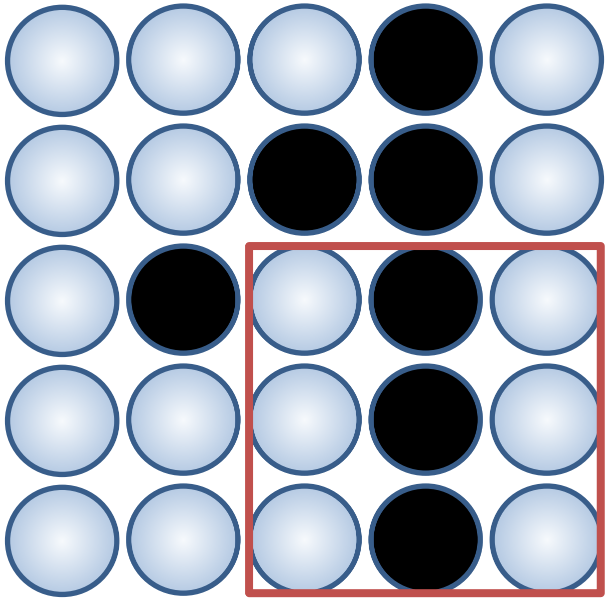



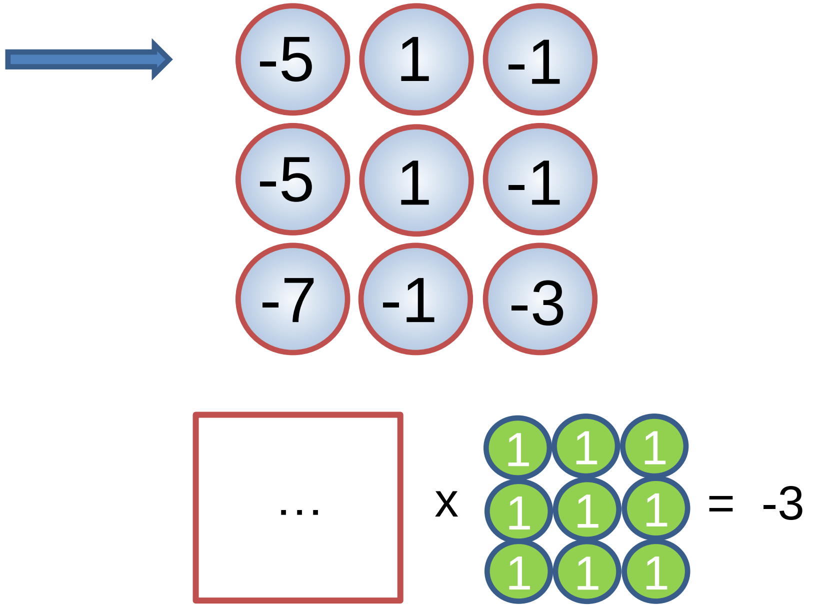
# Convolutional Neural Networks

## Learning on image data - Example



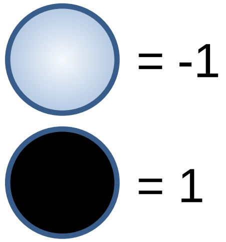

- This results in a “feature map” → here : 9 neurons for the 5x5 image

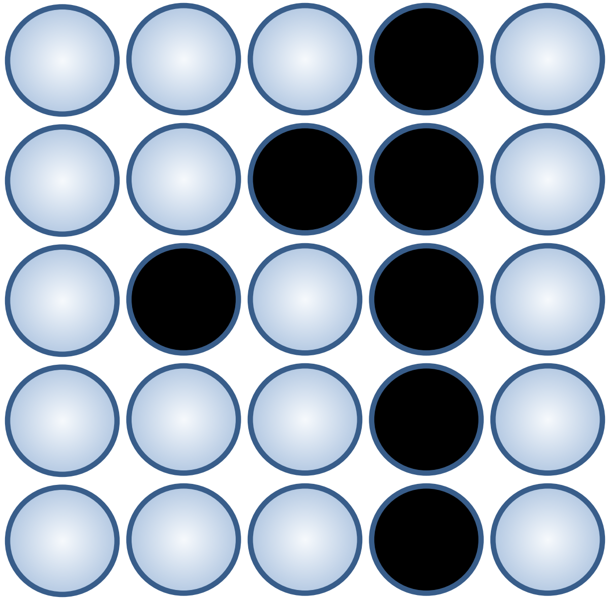





- Filter of size 3x3 → 9 separate weights
- Scan step wise the image with the filter
- That’s **Convolution**

# Convolutional Neural Networks
## Learning – Feature Extraction

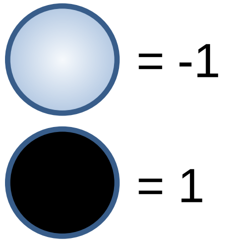


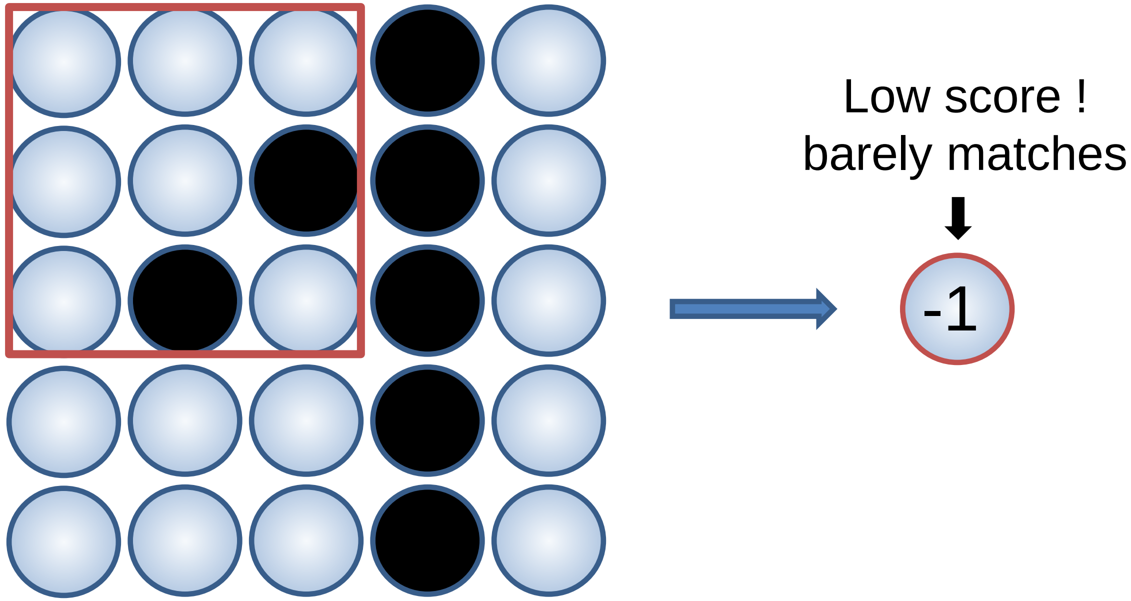



- Repeat the scan process with **multiple filters**

Next: extracting feature with filter $fi_1 = \begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & 1 \\ 1 & -1 & 1 \end{bmatrix}$

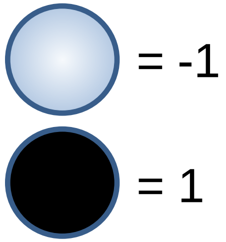
# Convolutional Neural Networks
## Learning – Feature Extraction

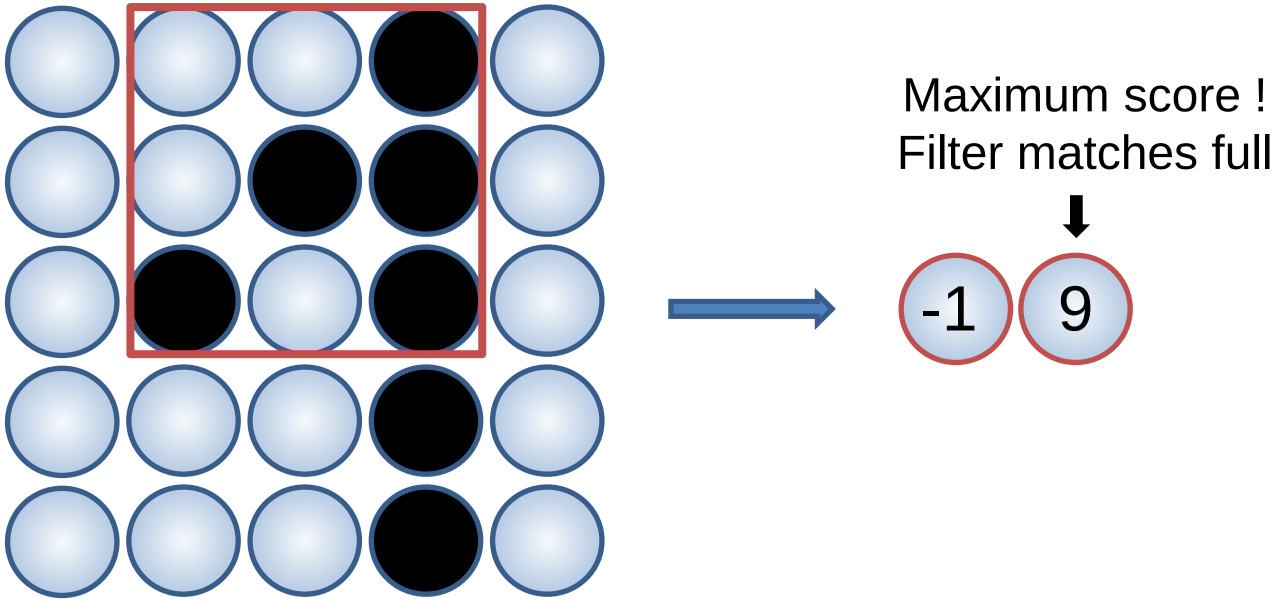


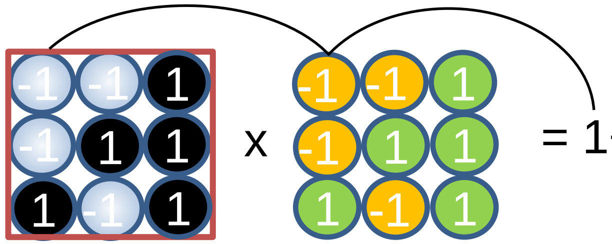




$$\begin{bmatrix}-1&-1&-1\\-1&-1&1\\-1&1&-1\end{bmatrix} \times \begin{bmatrix}-1&-1&1\\-1&1&1\\1&-1&1\end{bmatrix} = 1+1+(-1)+1+(-1)+1+(-1)+(-1)+(-1) = -1$$

# Convolutional Neural Networks
## Learning – Feature Extraction

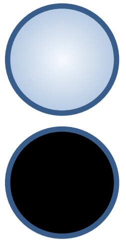


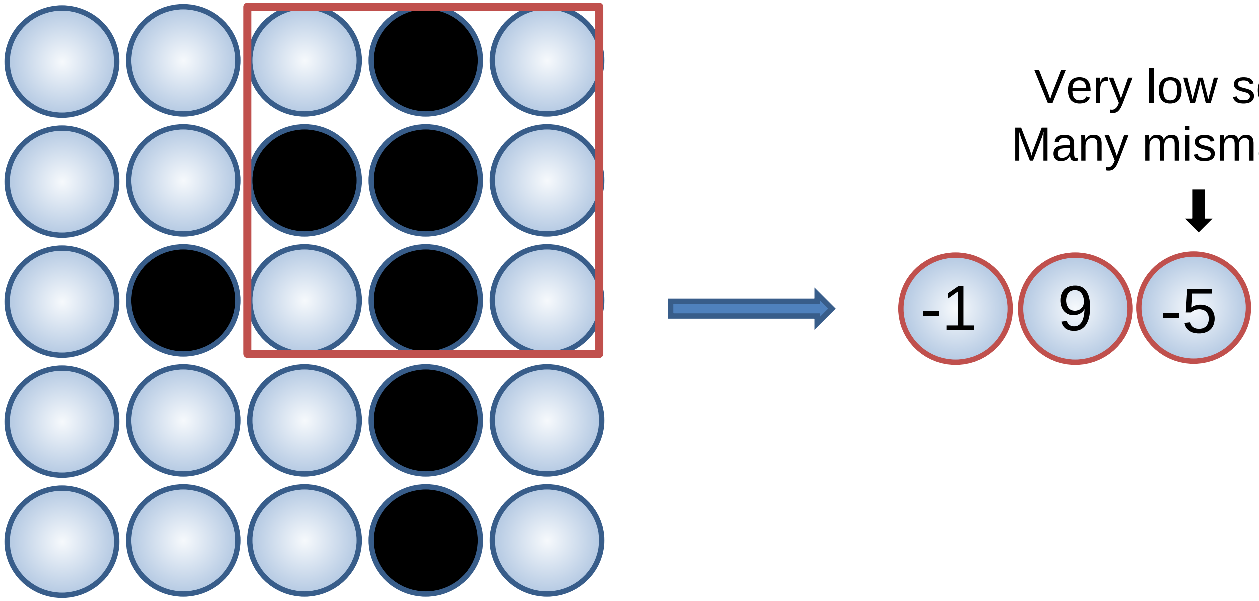



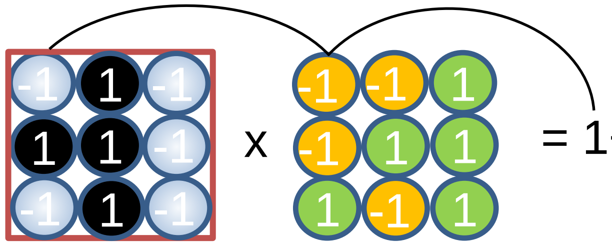

= 1+1+1+1+1+1+1+1+1 = 9

# Convolutional Neural Networks
## Learning – Feature Extraction



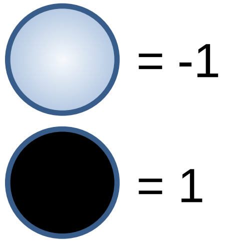

Very low score !
Many mismatches

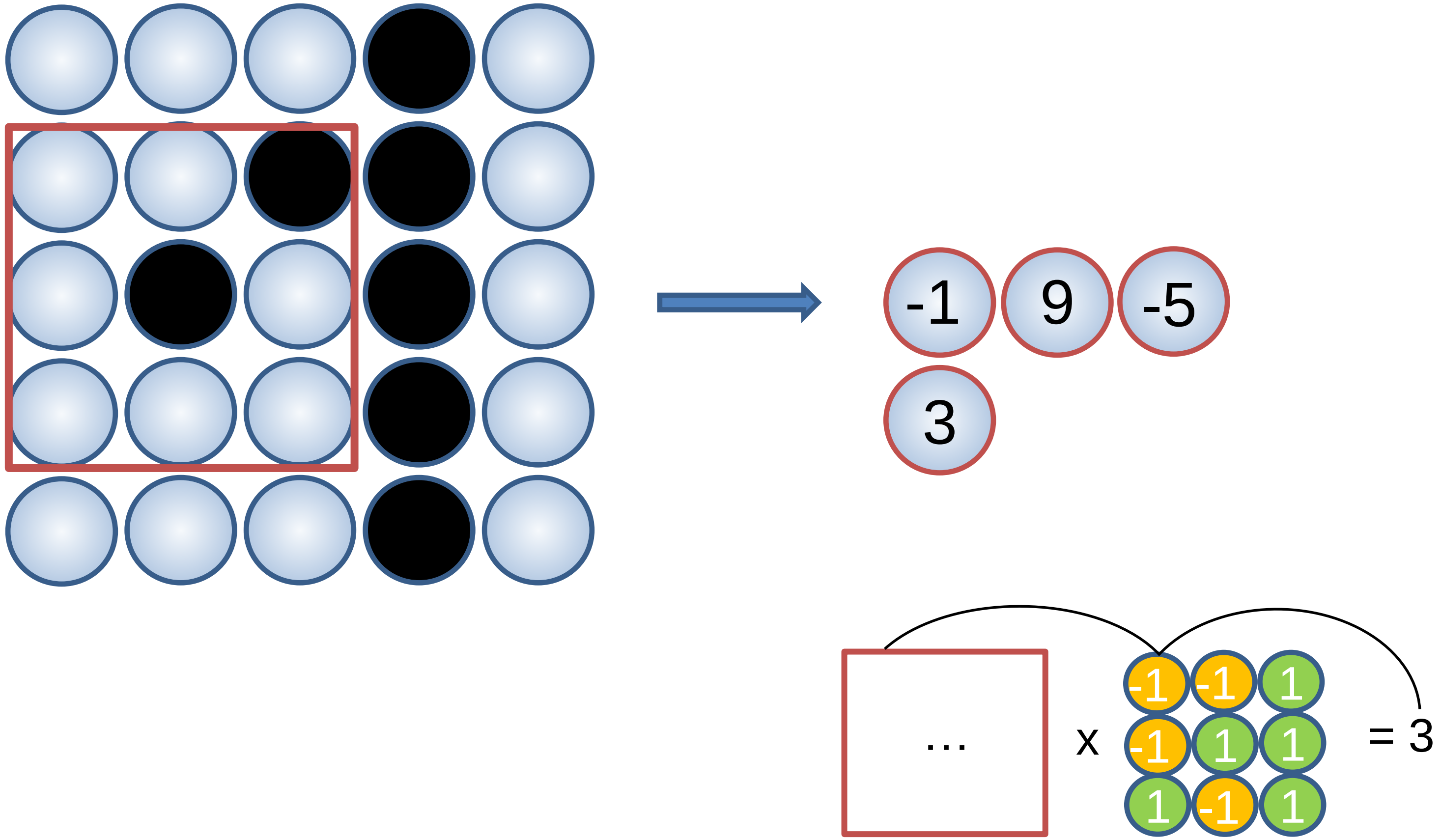


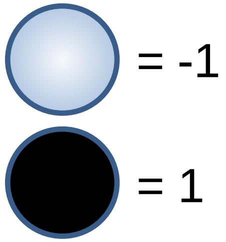

$= 1+(-1)+(-1)+(-1)+1+(-1)+(-1)+(-1)+(-1) = -5$

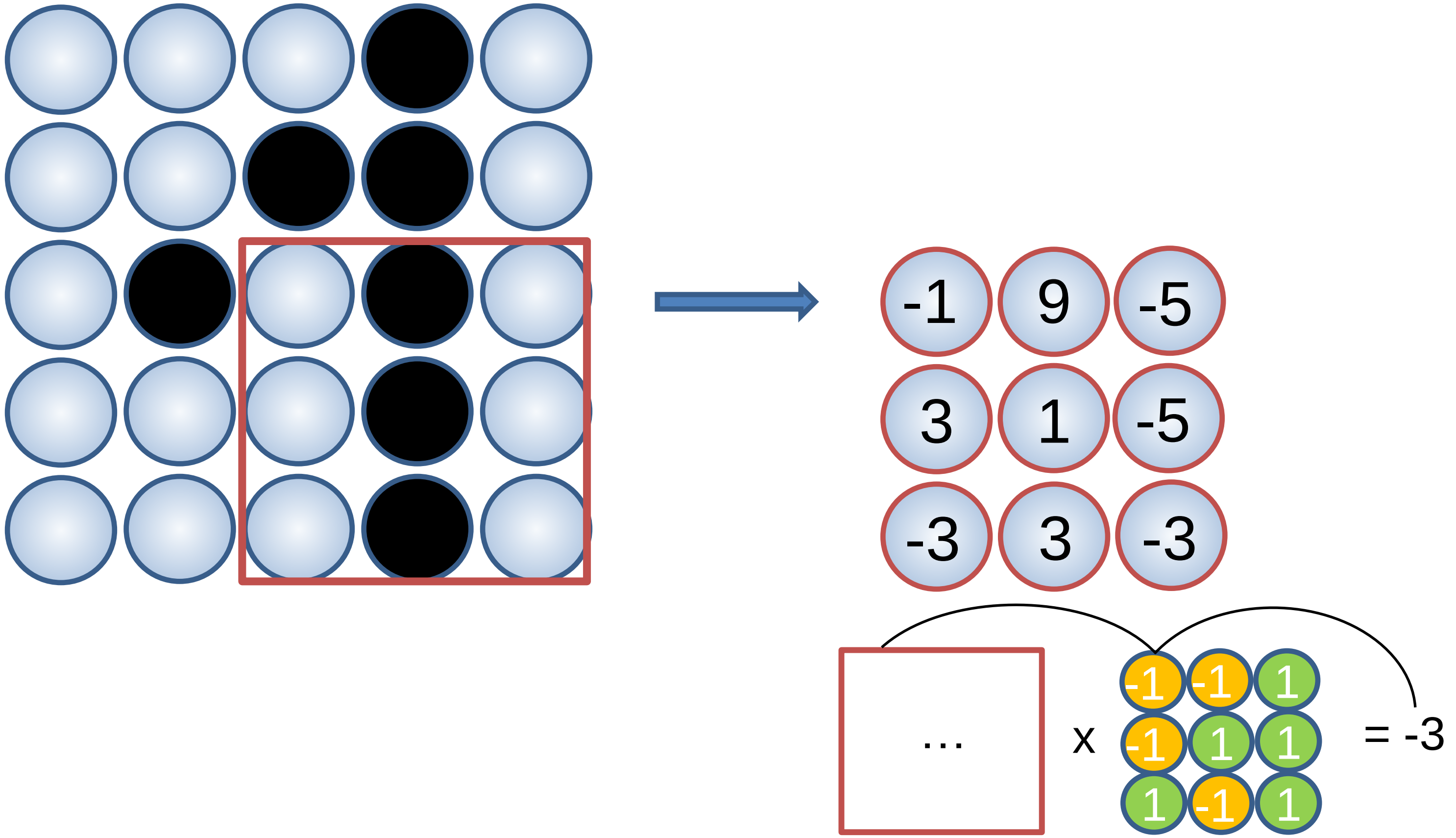
# Convolutional Neural Networks
## Learning – Feature Extraction



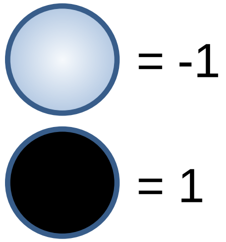



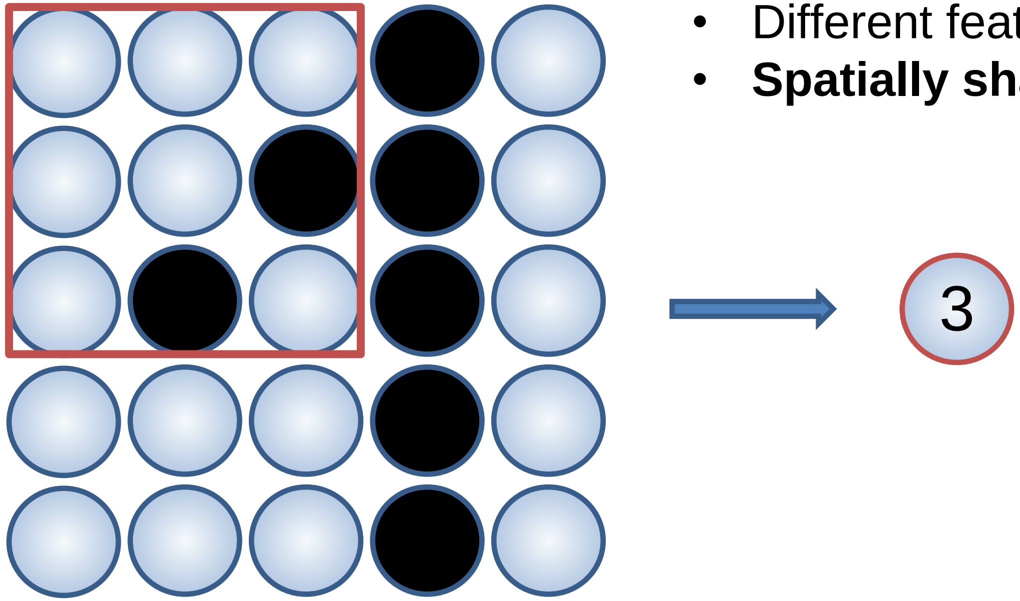
# Convolutional Neural Networks
## Learning – Feature Extraction



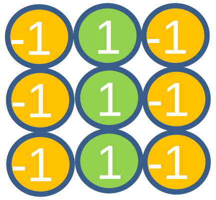
# Convolutional Neural Networks
## Learning – Feature Extraction

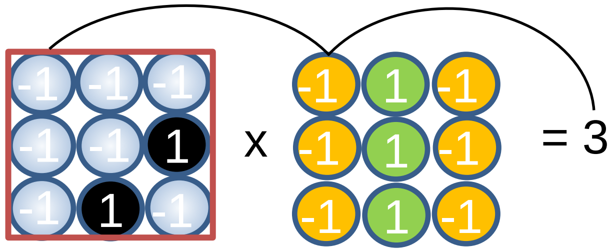



- Different feature will extract different features
- **Spatially share** parameters of each filter

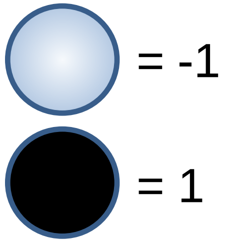
Next filter $\mathbf{fi_2}$ =



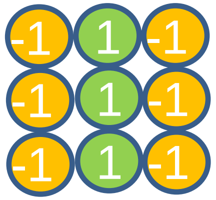
# Convolutional Neural Networks

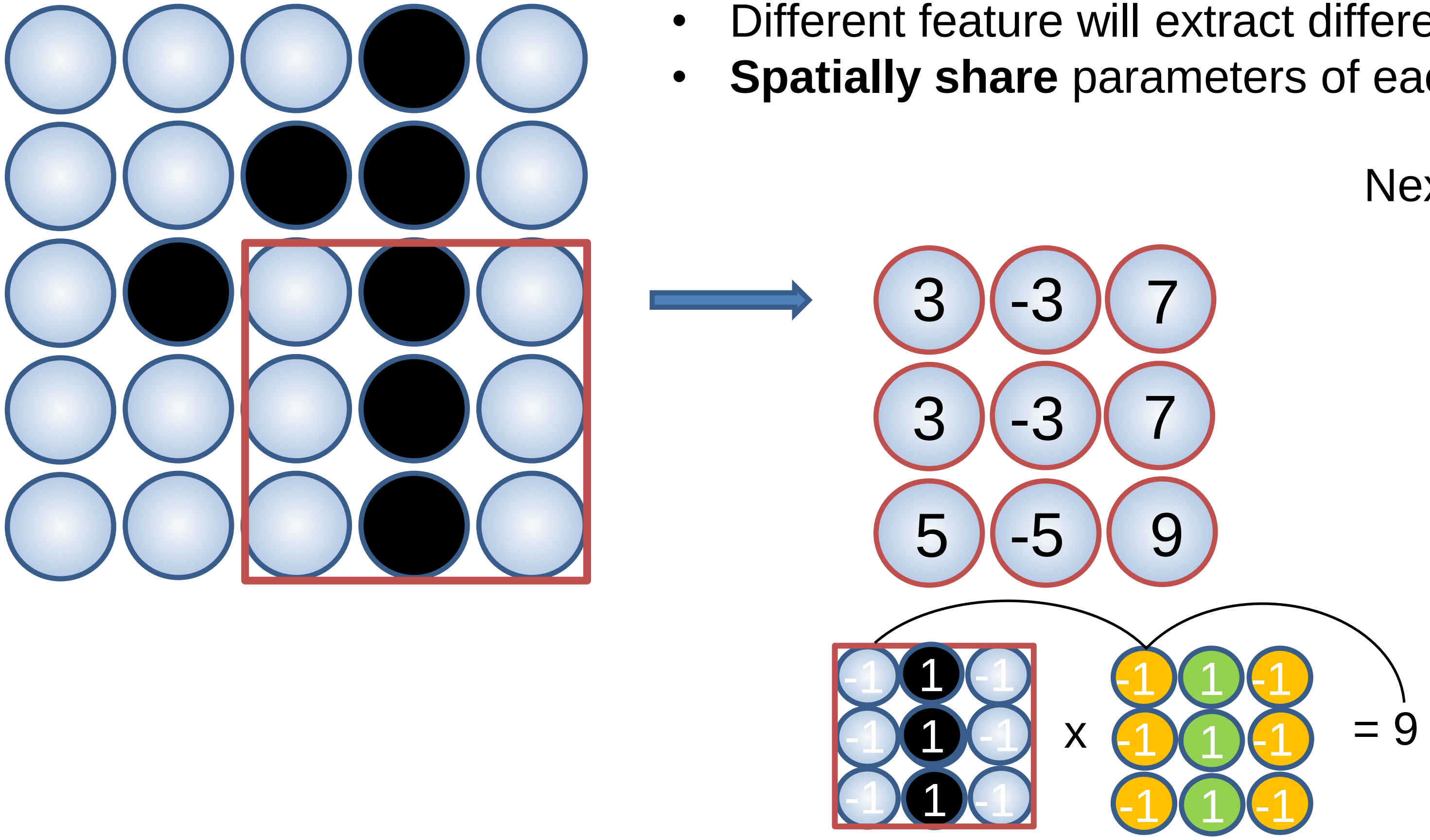
## Learning – Feature Extraction




- Different feature will extract different features
- **Spatially share** parameters of each filter

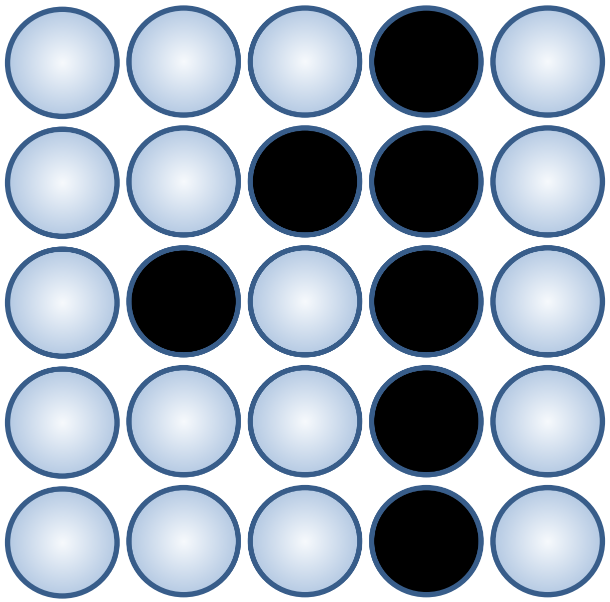
Next filter $\mathbf{fi}_2$ =



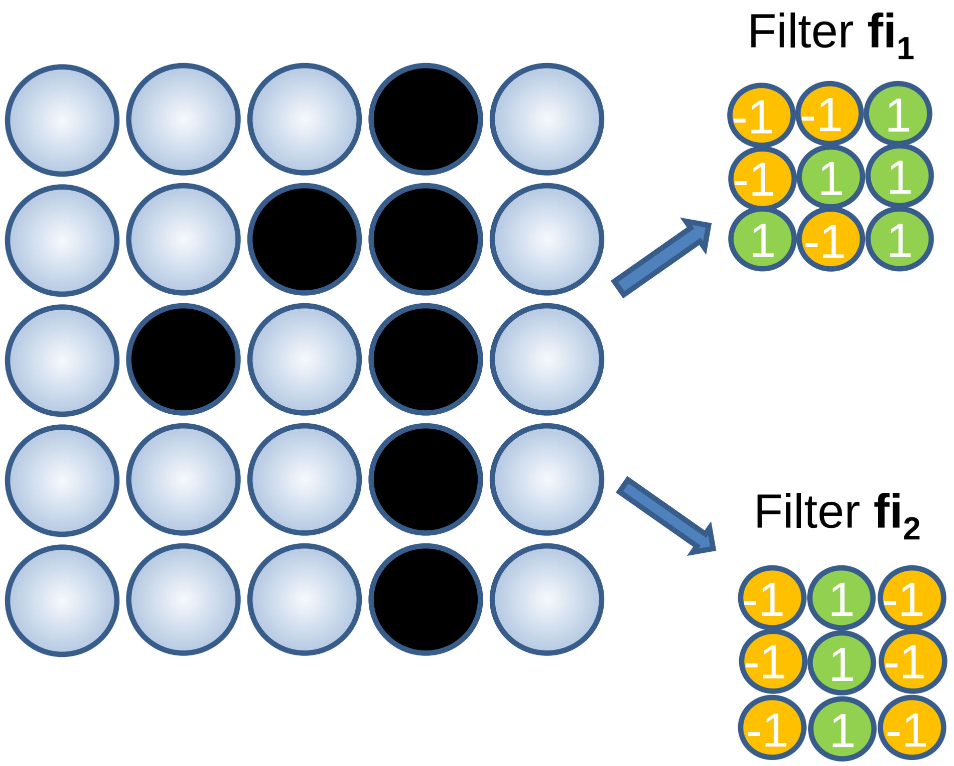
# Convolutional Neural Networks

## Learning – Feature Extraction: filter comparison



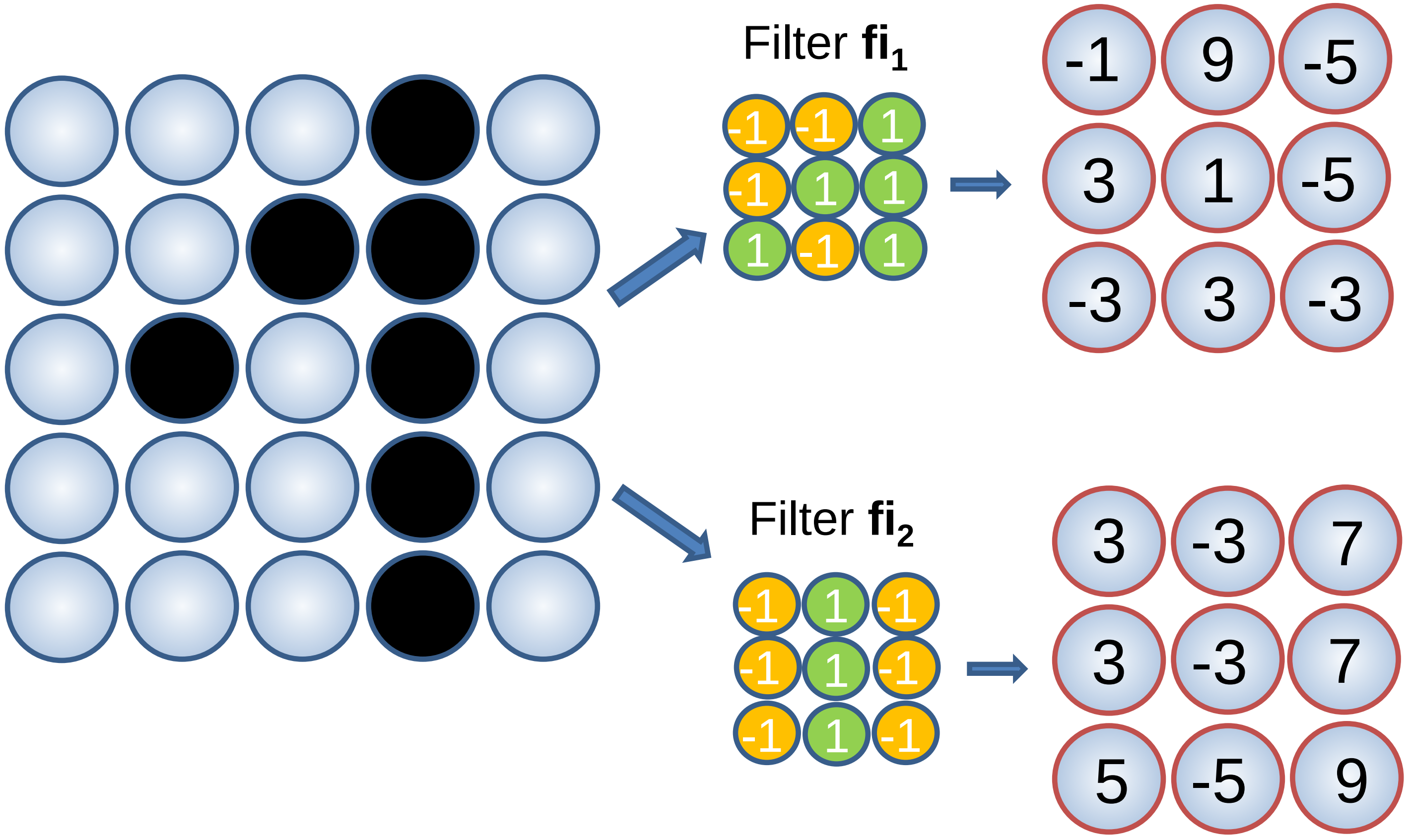
# Convolutional Neural Networks

## Learning – Feature Extraction: filter comparison

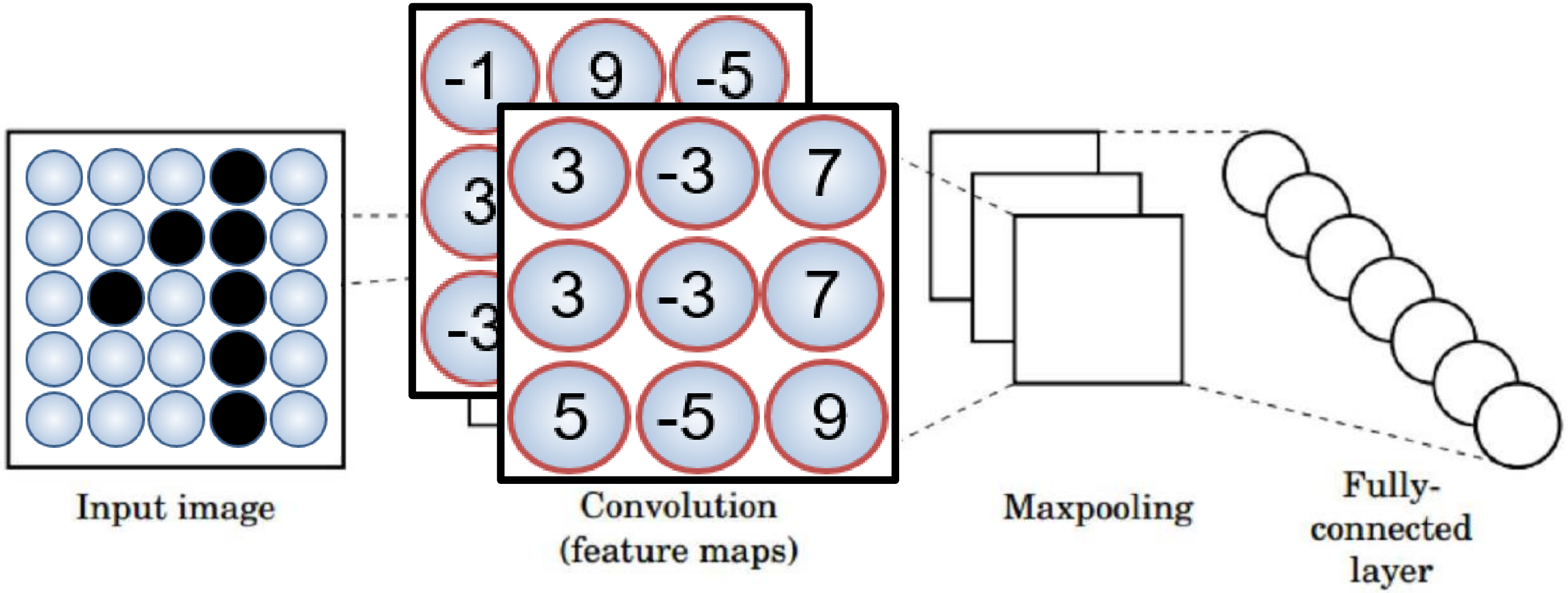


# Convolutional Neural Networks

## Learning – Feature Extraction: filter comparison



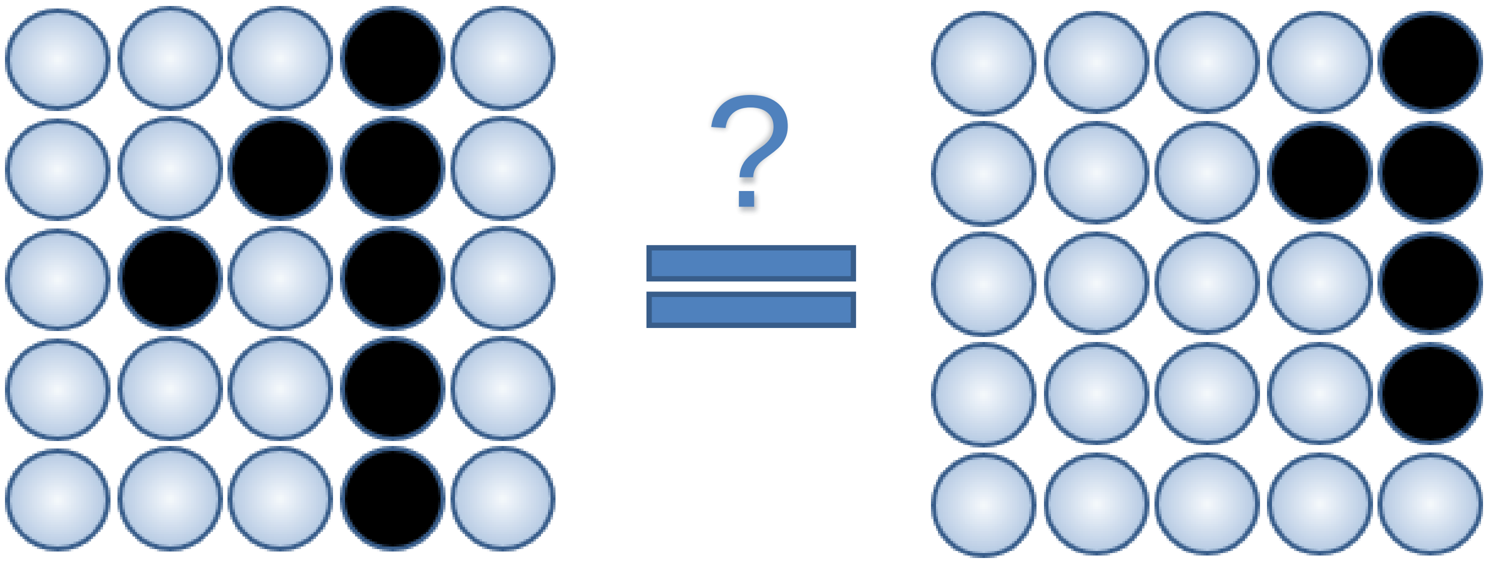
# Convolutional Neural Networks
## Building a CNN - Example



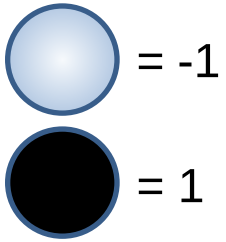

1. **Convolution**: Apply filters with learned weights to generate feature maps.
2. **Non-linearity**: Often ReLU.
3. **Pooling**: Downsampling operation on each feature map.

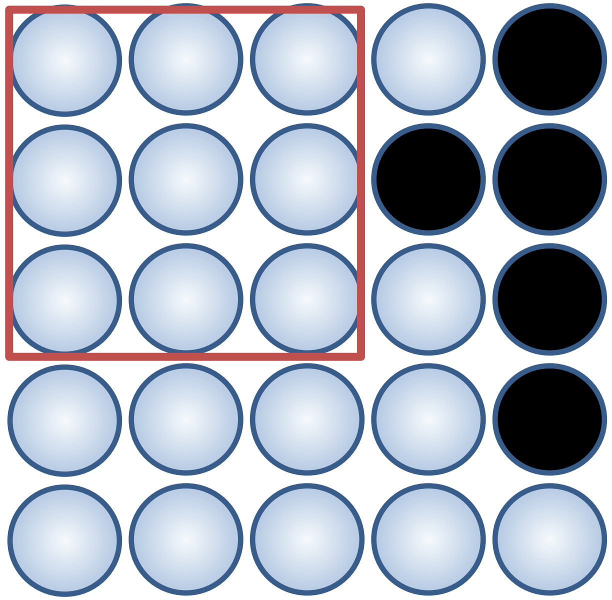
**Train model with image data.**
**Learn weights of filters in convolutional layers.**

# Convolutional Neural Networks
## Filters to detect features



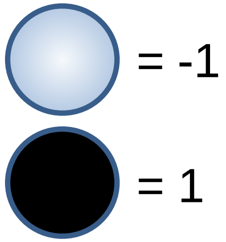
# Convolutional Neural Networks
## Filters to detect features

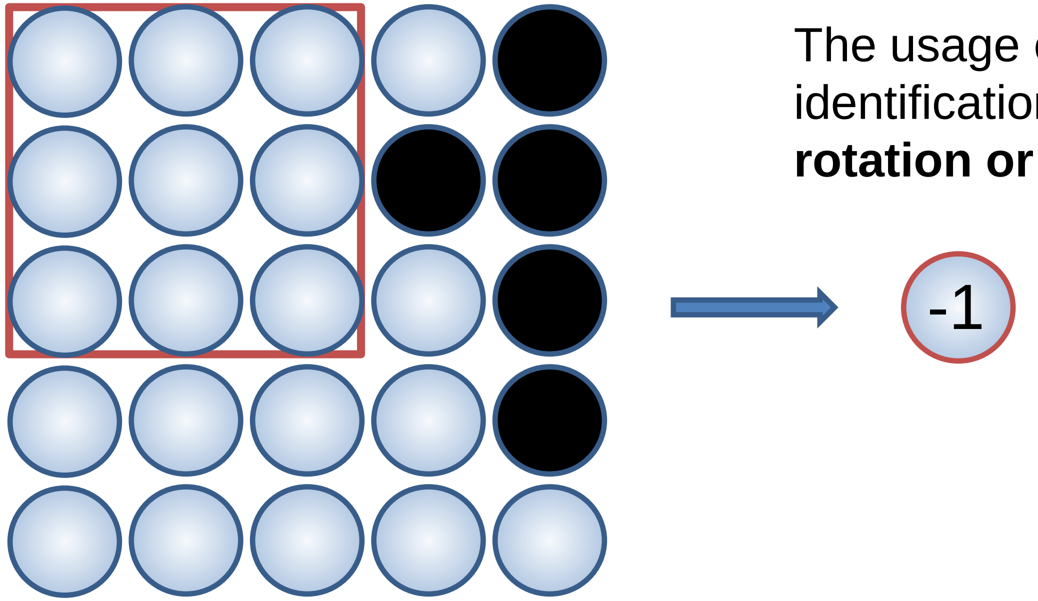


The usage of filters in detecting features allow good identification even in the event of **displacement, shrinkage, rotation or deformation**.

# Convolutional Neural Networks
## Filters to detect features



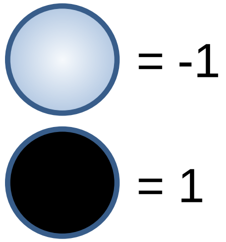



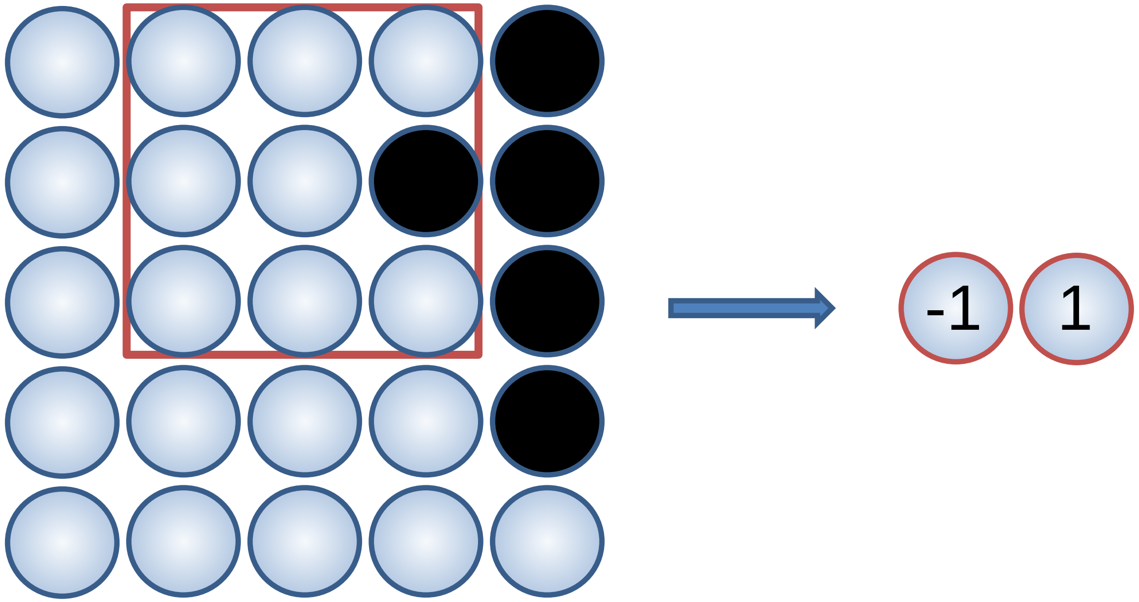

The usage of filters in detecting features allow good identification even in the event of **displacement, shrinkage, rotation or deformation**.

e.g. Filter $\mathbf{fi}_1 = \begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & 1 \\ 1 & -1 & 1 \end{bmatrix}$

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{bmatrix} \times \begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & 1 \\ 1 & -1 & 1 \end{bmatrix} = \ldots$$

# Convolutional Neural Networks
## Filters to detect features



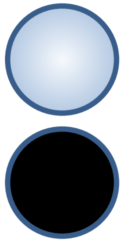

e.g. Filter $\mathbf{fi_1} = \begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & 1 \\ 1 & -1 & 1 \end{bmatrix}$

Feature/filter $fi_1$ still matches the deformed image well

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & 1 \\ -1 & -1 & -1 \end{bmatrix} \times \begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & 1 \\ 1 & -1 & 1 \end{bmatrix} = \ldots$$

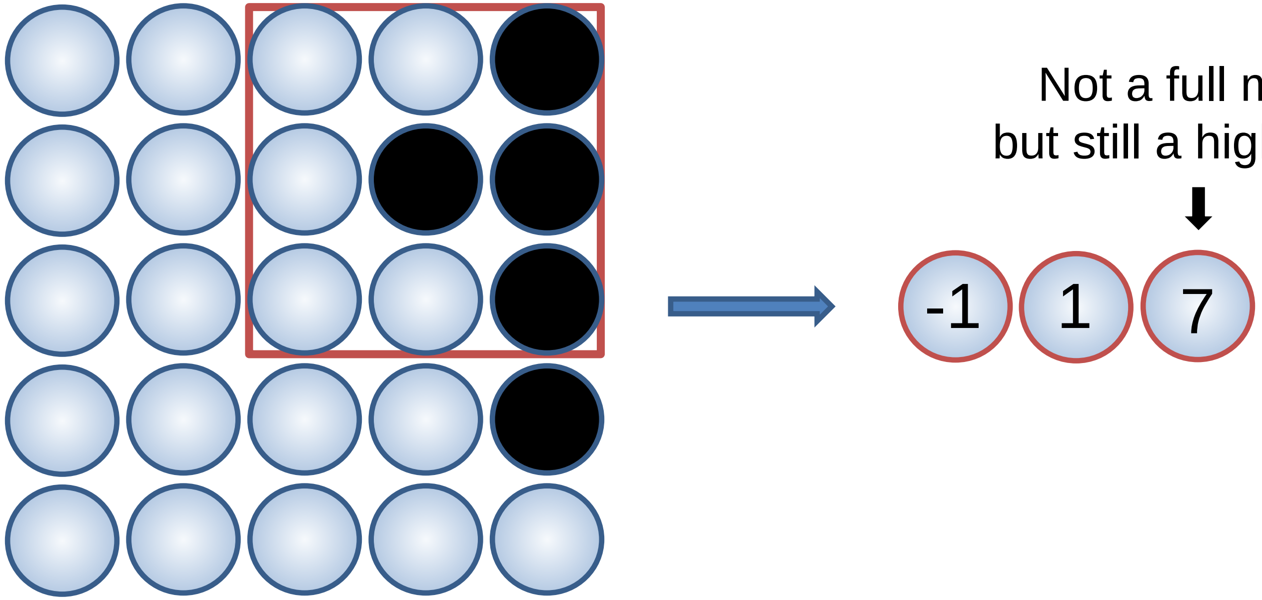
# Convolutional Neural Networks

## Filters to detect features



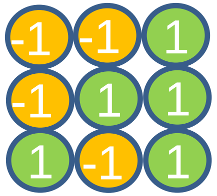
= -1

= 1

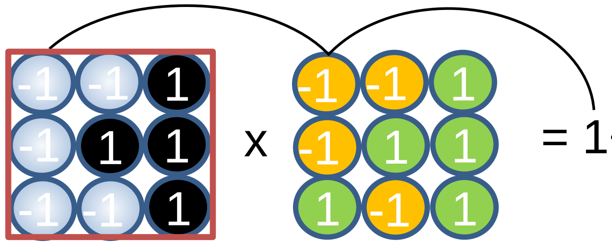



Not a full match
but still a high score

e.g. Filter **$fi_1$** =

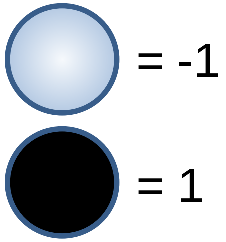


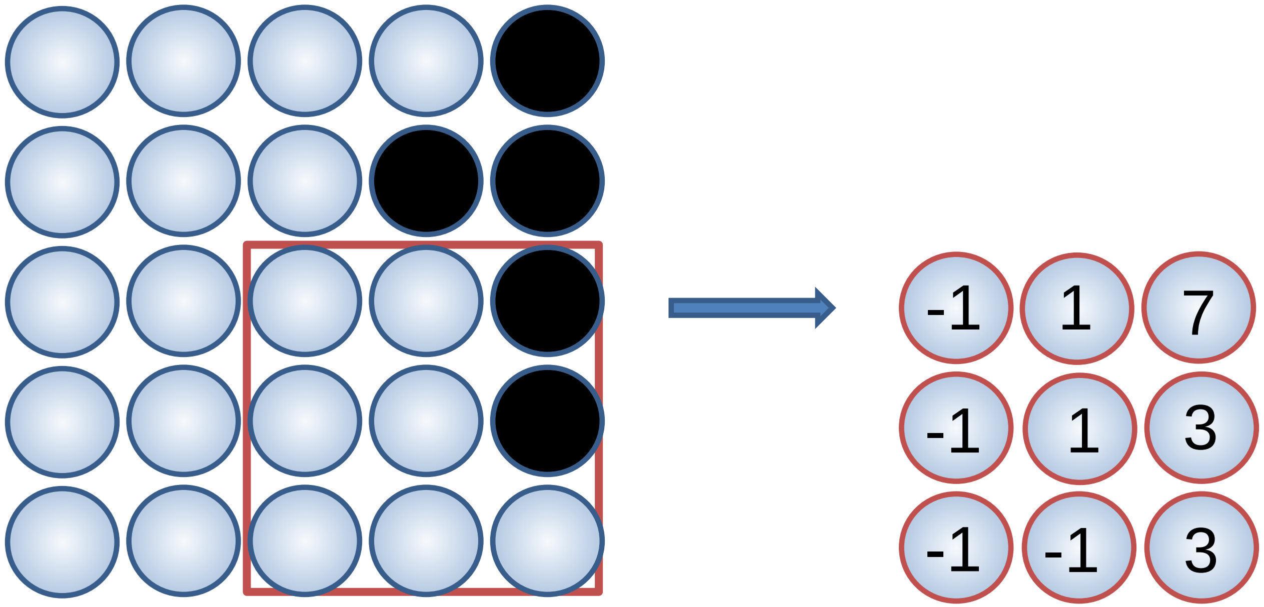

Feature/filter $fi_1$ still matches the deformed image well
Remember! **A full match would be 9**




= 1+1+1+1+1+1+(-1)+1+1 = 7

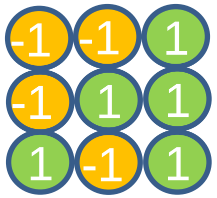
# Convolutional Neural Networks
## Filters to detect features



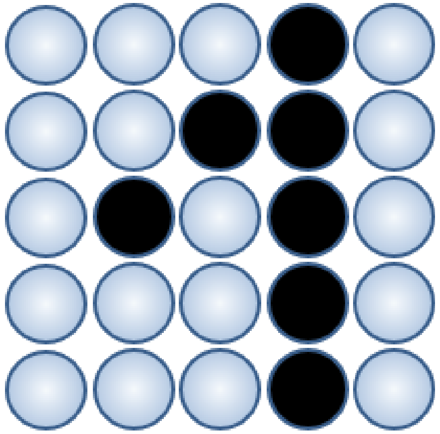



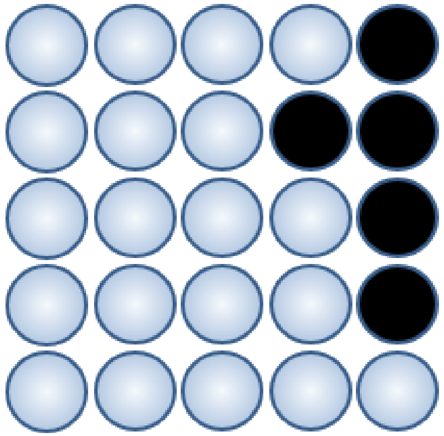

Finished feature map using filter **fi**$_1$

e.g. Filter **fi**$_1$ =

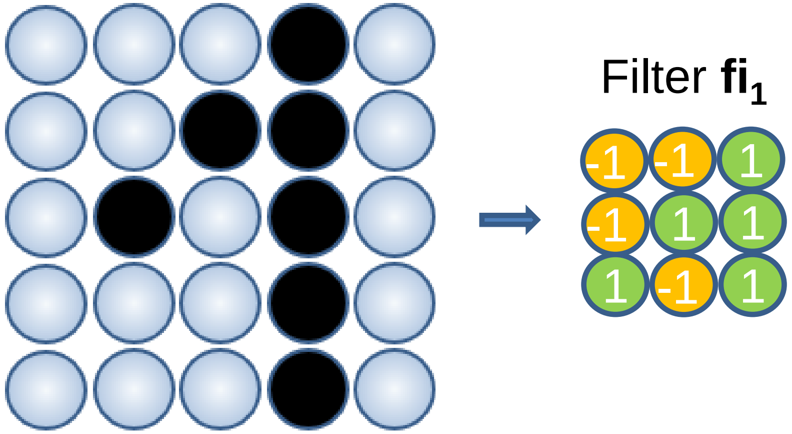


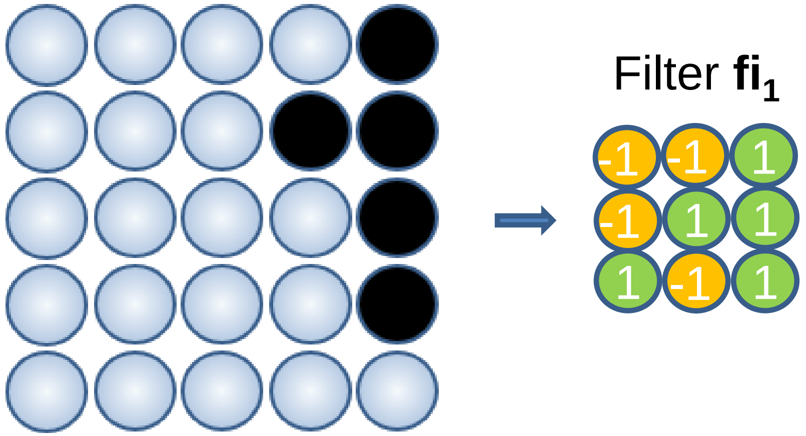
# Convolutional Neural Networks
## Filters to detect features: comparison



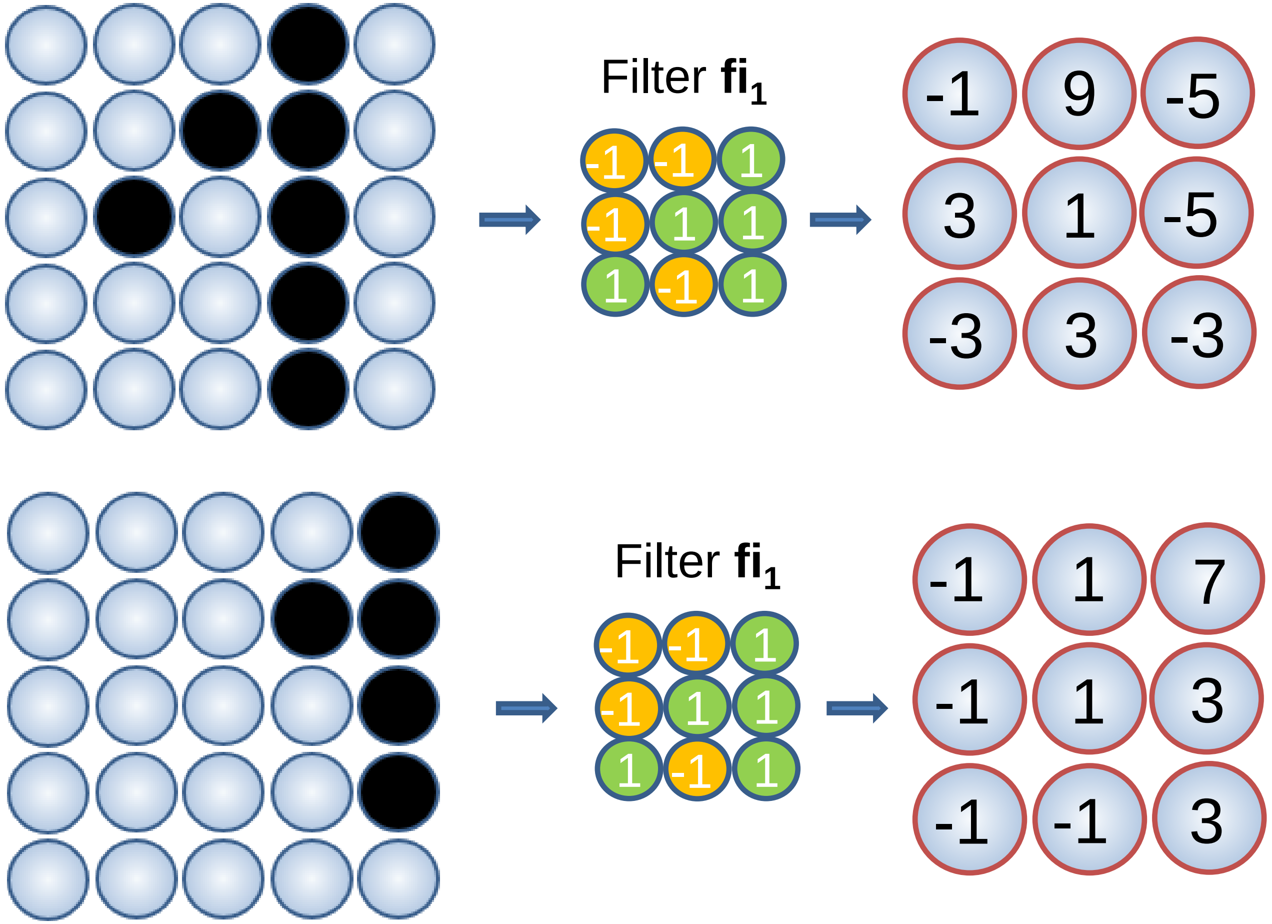


# Convolutional Neural Networks
## Filters to detect features: comparison

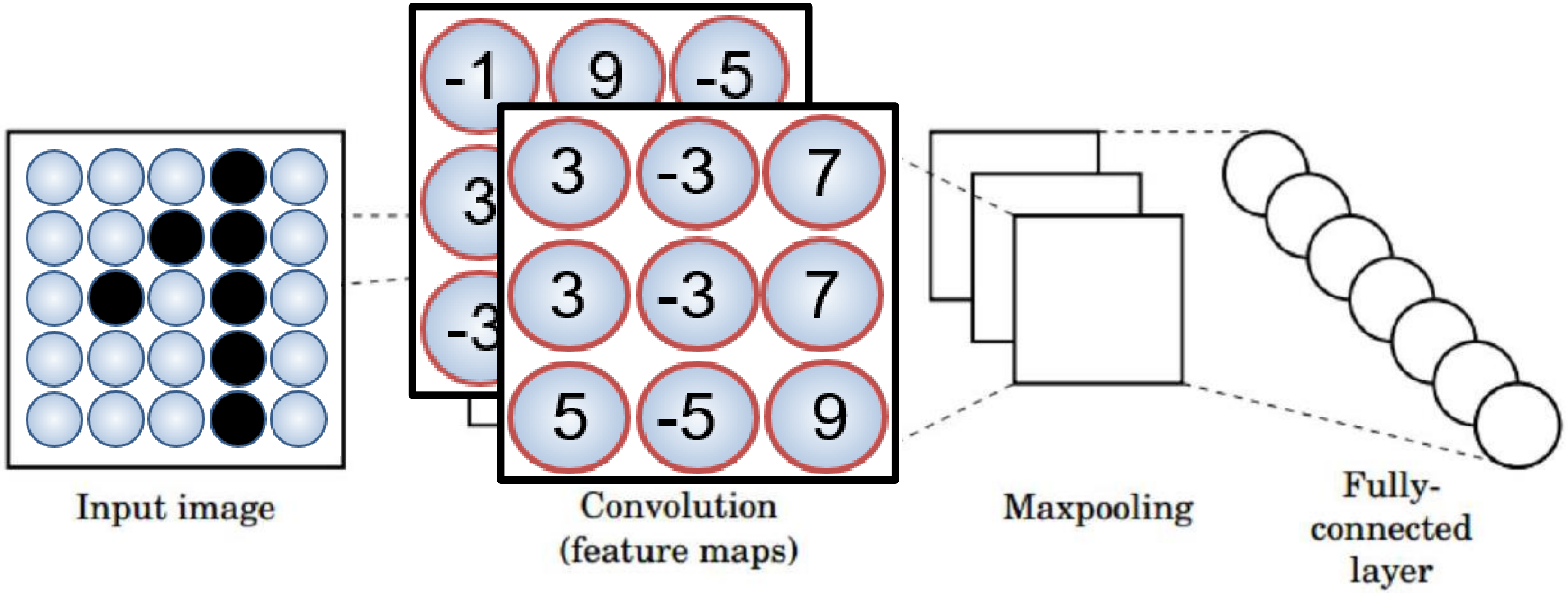


# Convolutional Neural Networks
## Filters to detect features: comparison



# Convolutional Neural Networks
## Pooling

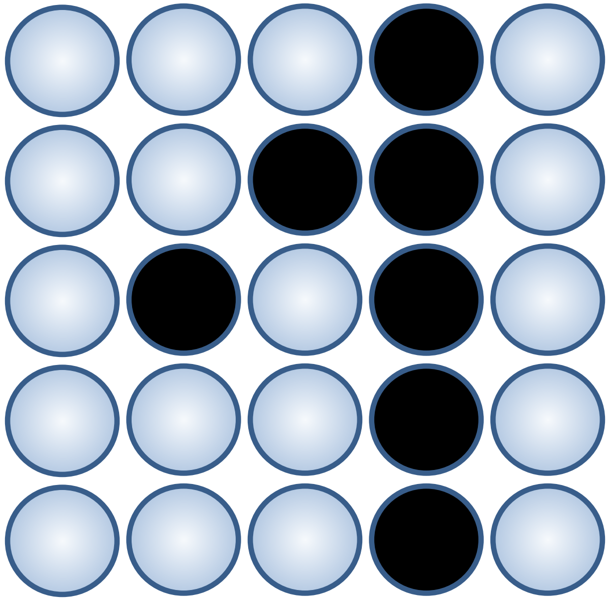


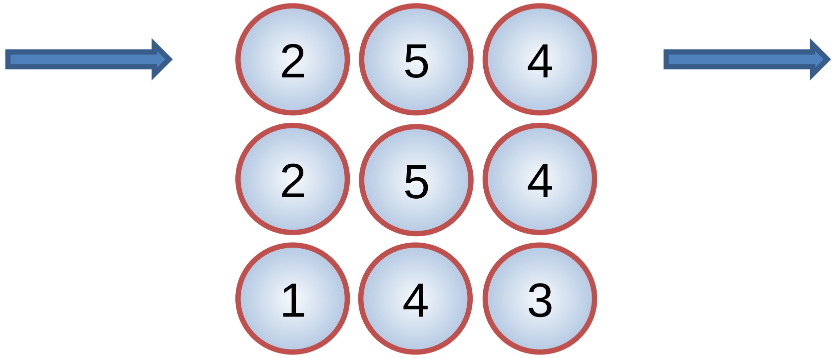

1. **Convolution**: Apply filters with learned weights to generate feature maps.
2. **Non-linearity**: Often ReLU.
3. **Pooling**: Downsampling operation on each feature map.

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

# Convolutional Neural Networks
## Max Pool – Example

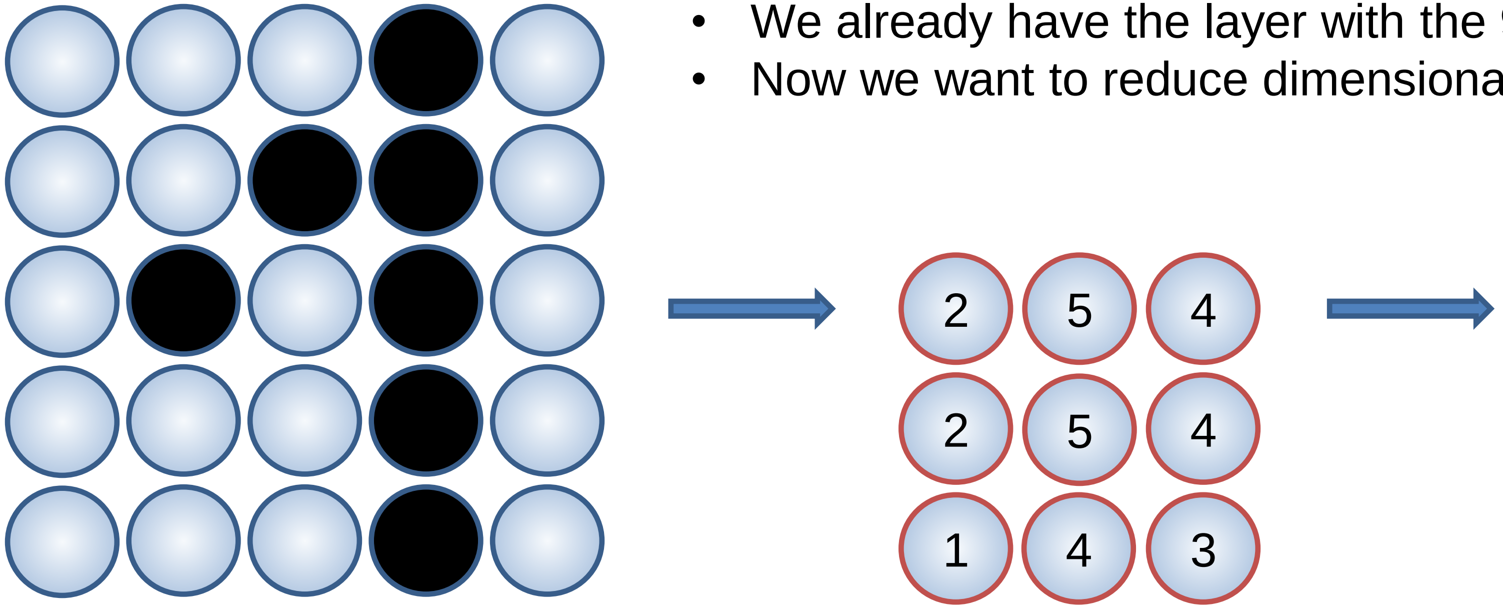
- We already have the layer with the 9 neurons for the 5x5 image





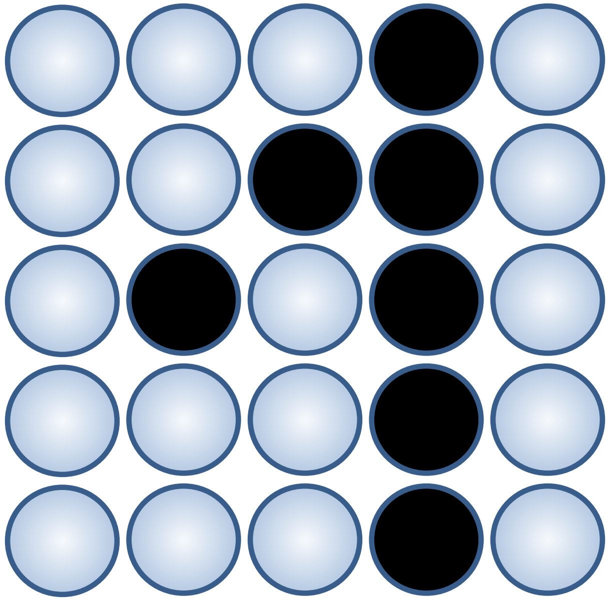
# Convolutional Neural Networks
## Max Pool – Example

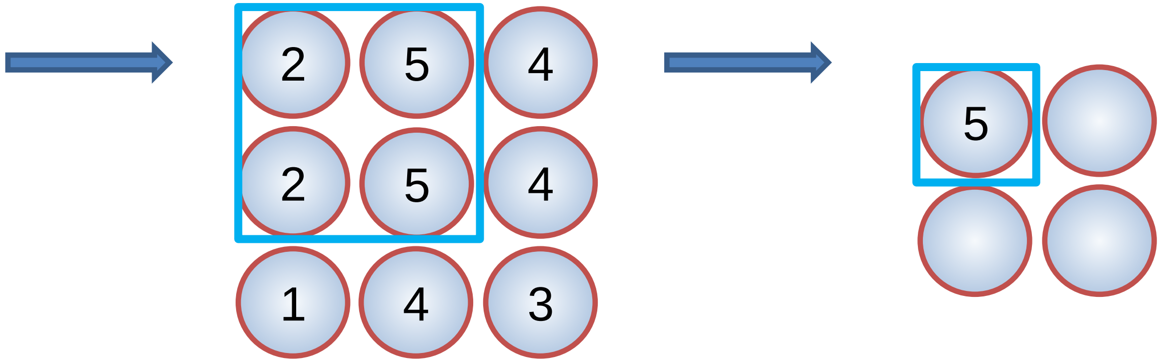
- We already have the layer with the 9 neurons for the 5x5 image
- Now we want to reduce dimensionality and spatial invariance



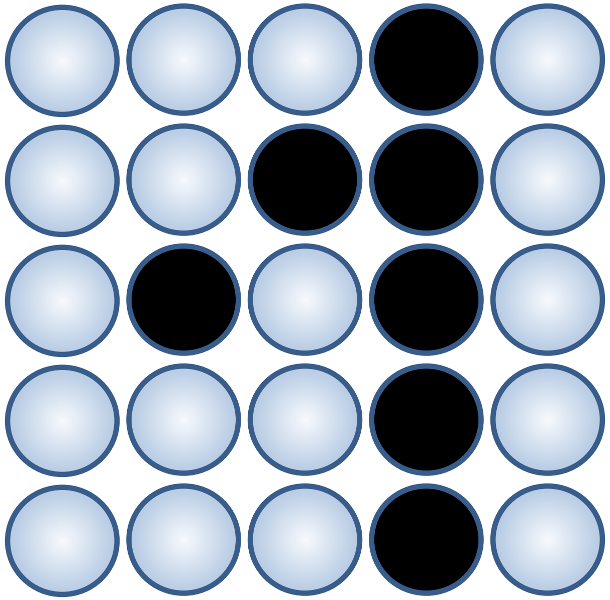
# Convolutional Neural Networks
## Max Pool – Example

- We already have the layer with the 9 neurons for the 5x5 image
- Now we want to reduce dimensionality and spatial invariance
- Using max pool (find max with 2x2 filters and stride 1

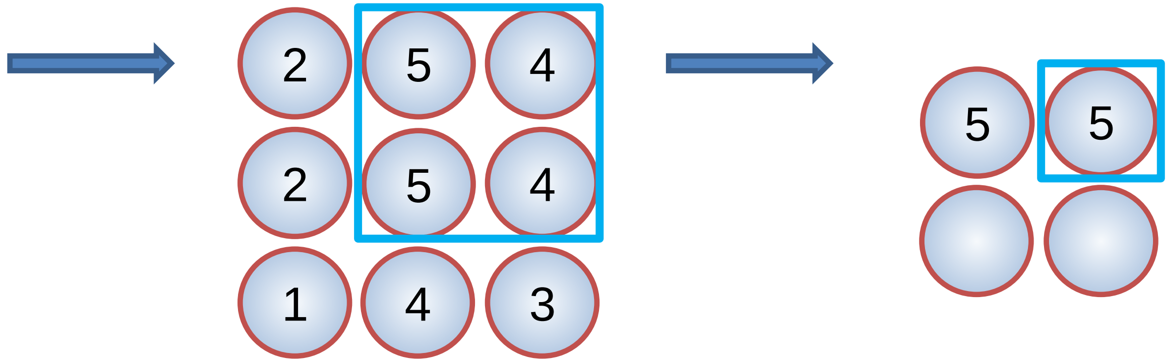




# Convolutional Neural Networks
## Max Pool – Example

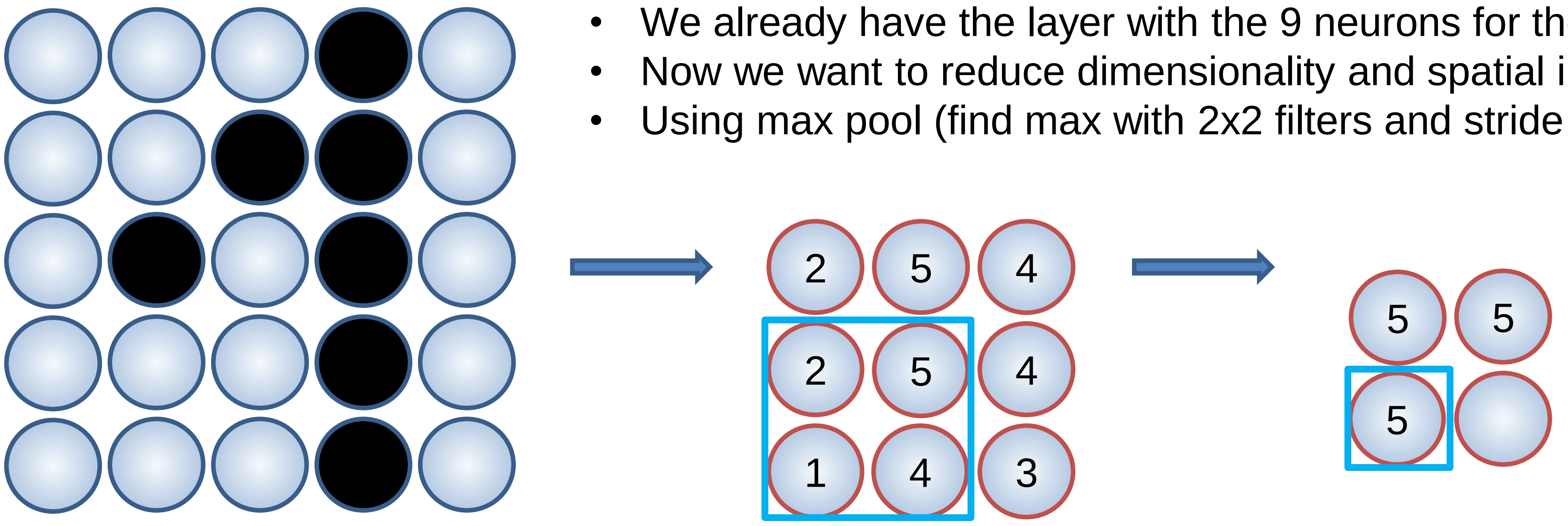


- We already have the layer with the 9 neurons for the 5x5 image
- Now we want to reduce dimensionality and spatial invariance
- Using max pool (find max with 2x2 filters and stride 1



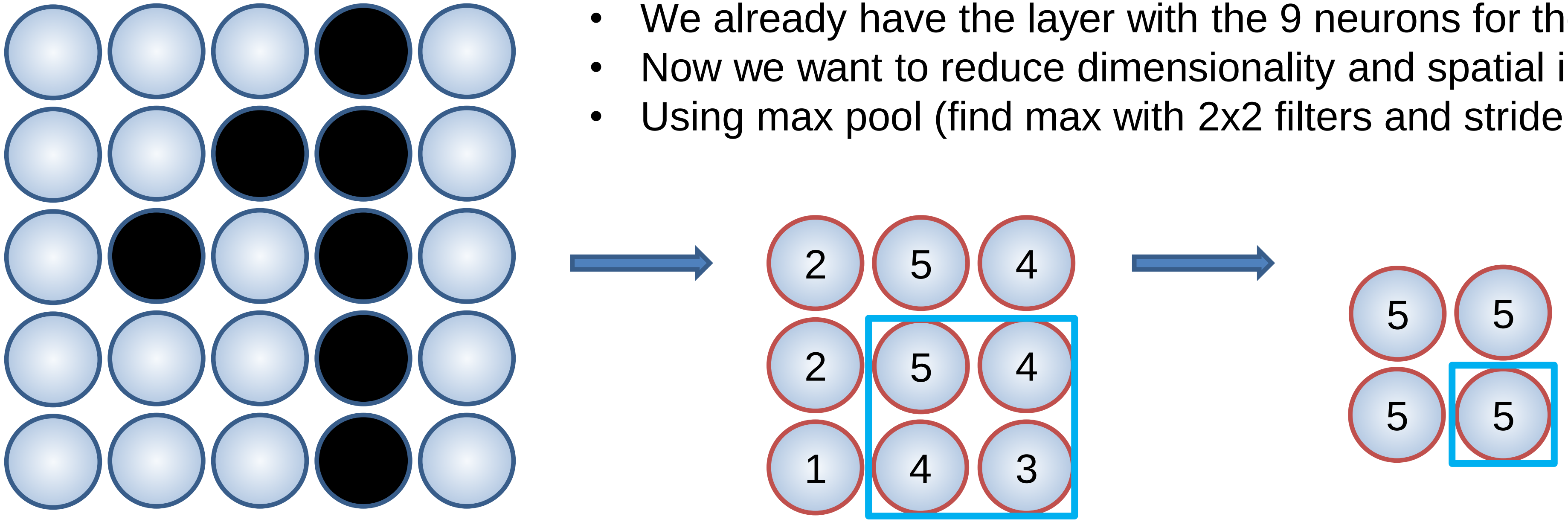
# Convolutional Neural Networks
## Max Pool – Example

- We already have the layer with the 9 neurons for the 5x5 image
- Now we want to reduce dimensionality and spatial invariance
- Using max pool (find max with 2x2 filters and stride 1



# Convolutional Neural Networks
## Max Pool – Example

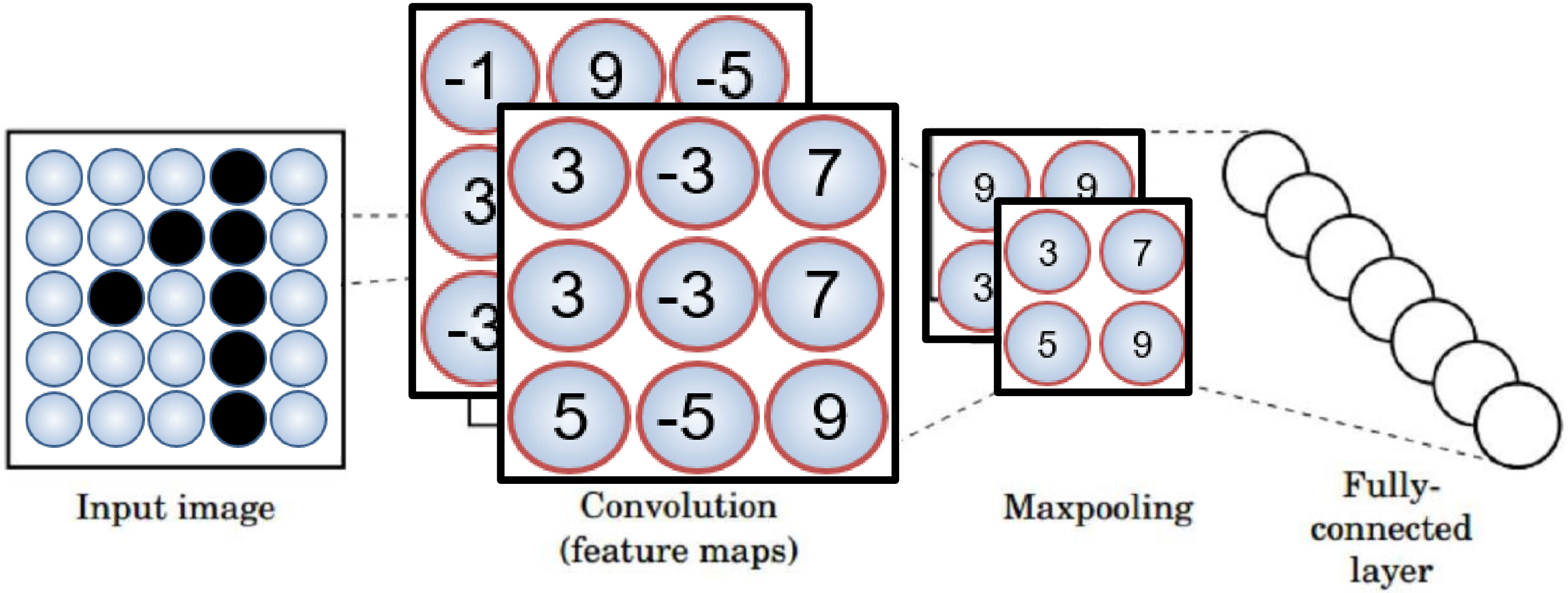
- We already have the layer with the 9 neurons for the 5x5 image
- Now we want to reduce dimensionality and spatial invariance
- Using max pool (find max with 2x2 filters and stride 1






A.Fitri, S.Vemuri, S.Naveenachandran, DLR-DW, 13.-14.11.2025

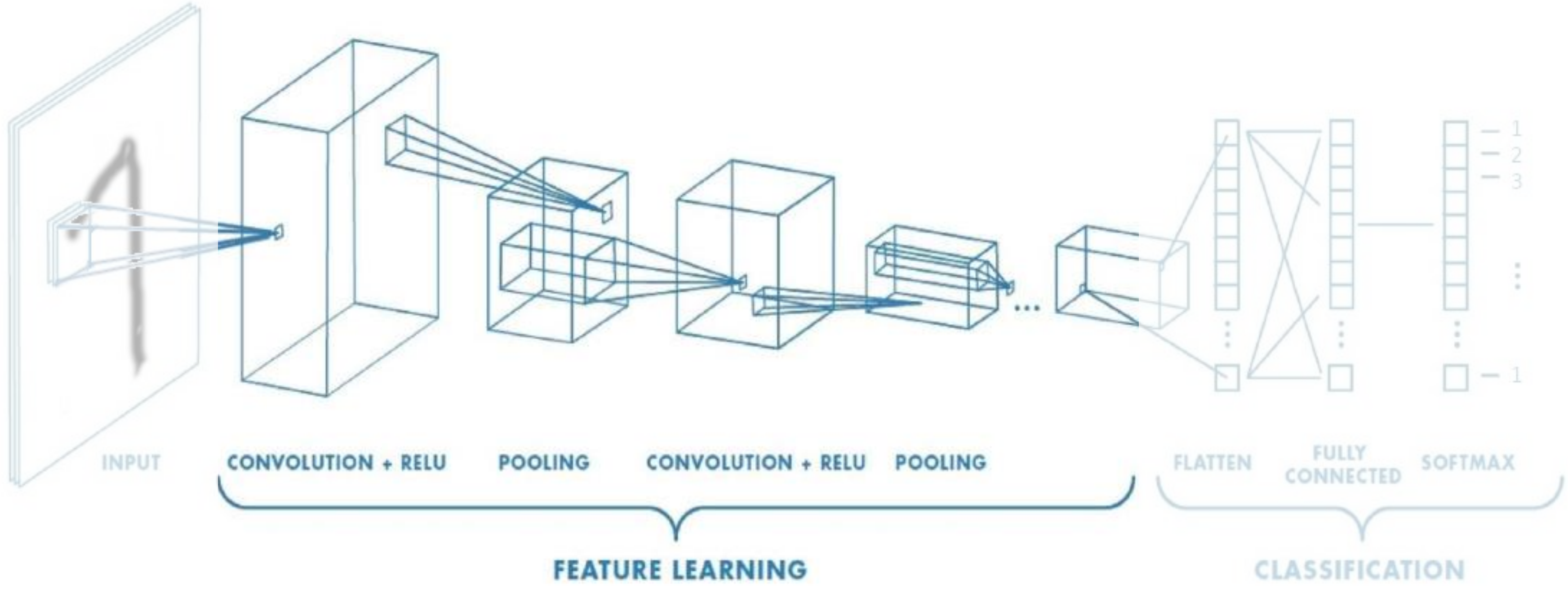
# Convolutional Neural Networks
## Building a CNN – Example



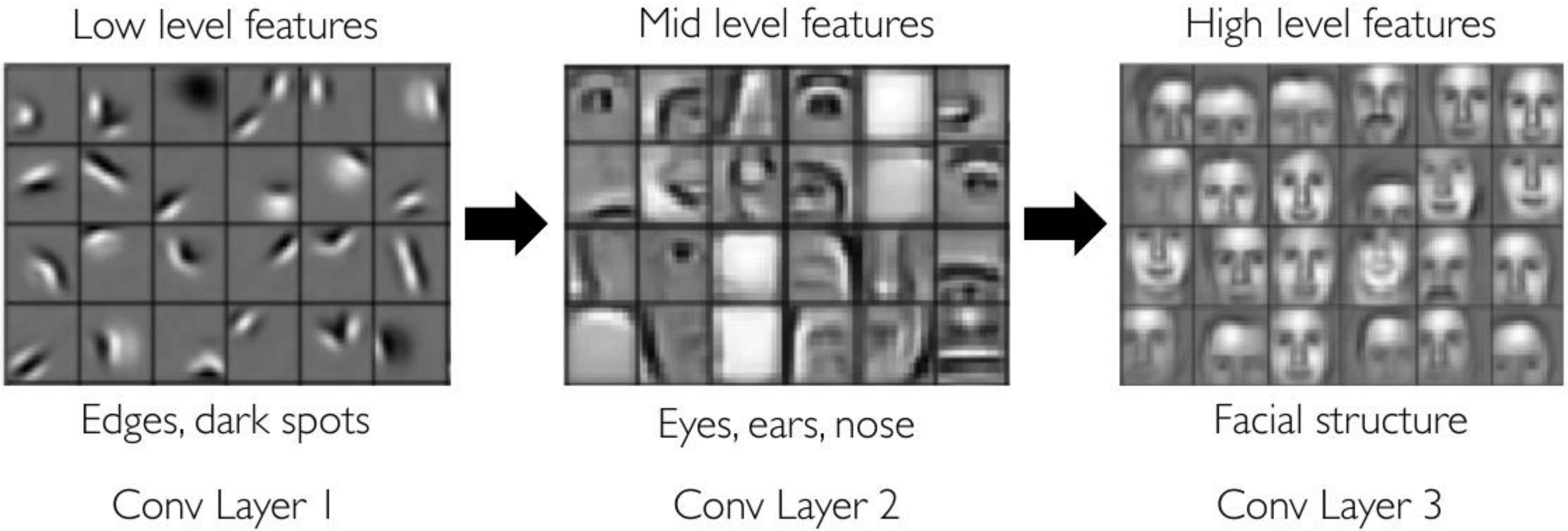

1. **Convolution**: Apply filters with learned weights to generate feature maps.
2. **Non-linearity**: Often ReLU.
3. **Pooling**: Downsampling operation on each feature map.

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

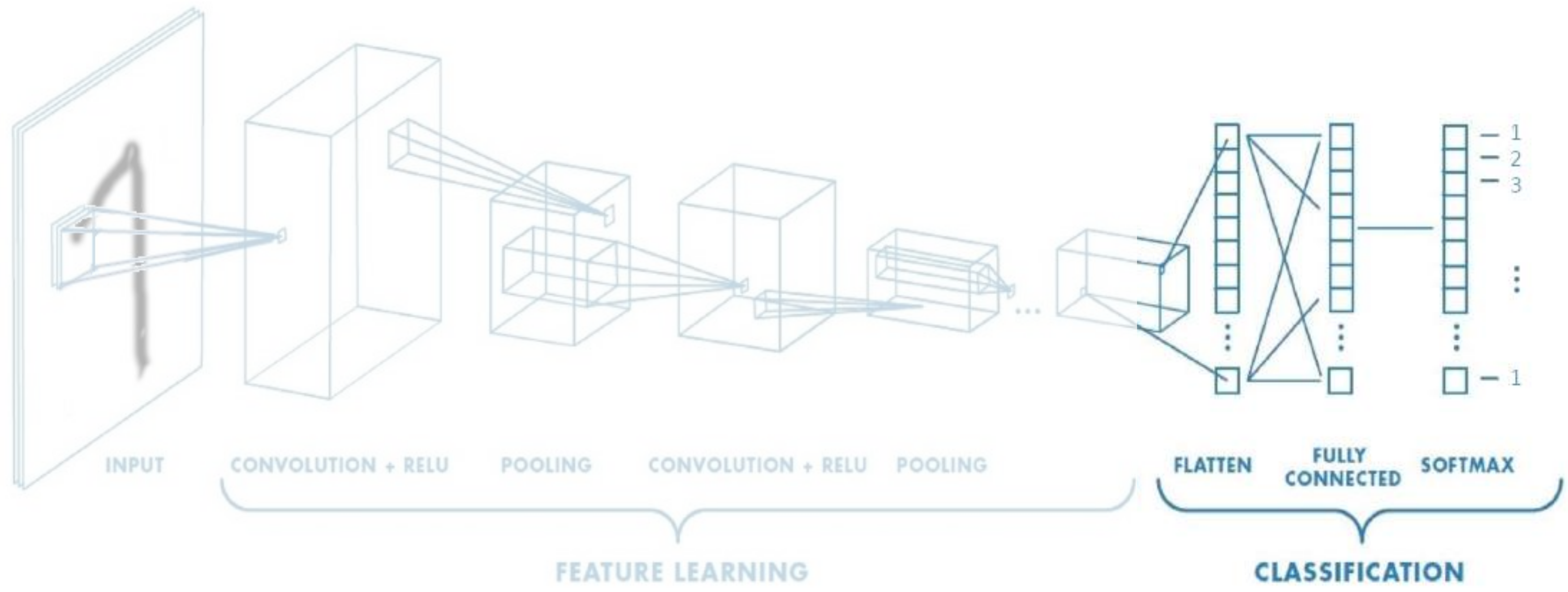
## Representation Learning



# Convolutional Neural Networks
## Representation Learning



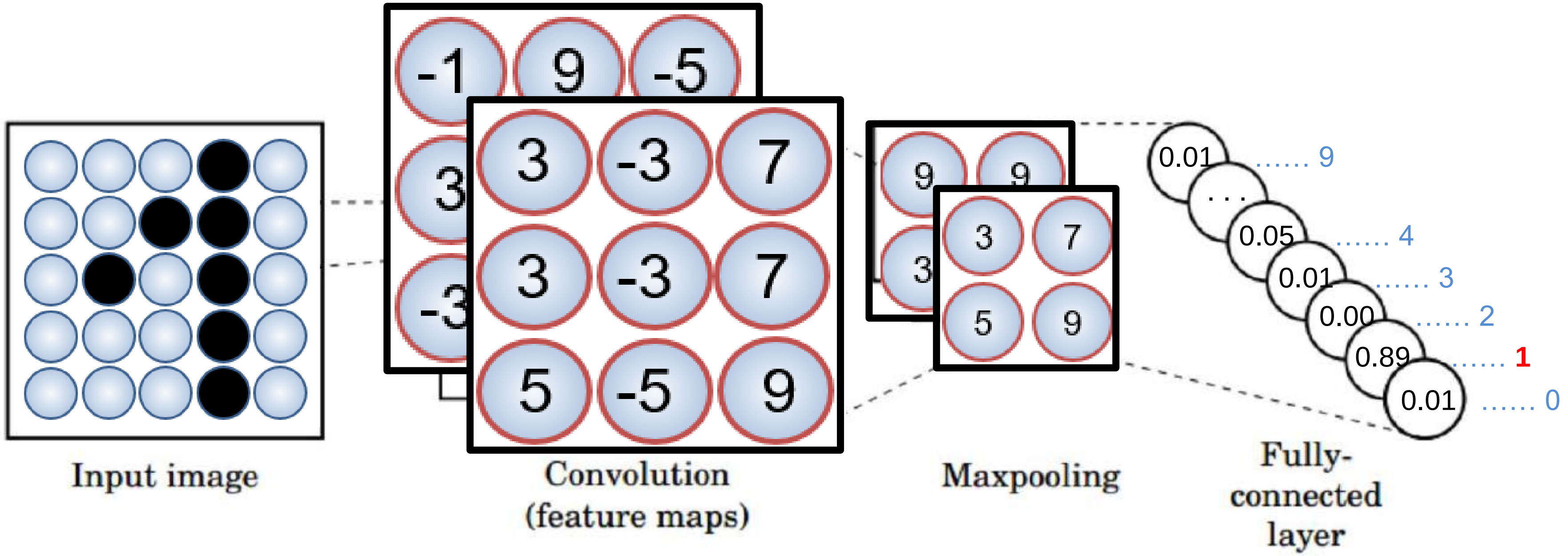
# Classification




- 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

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

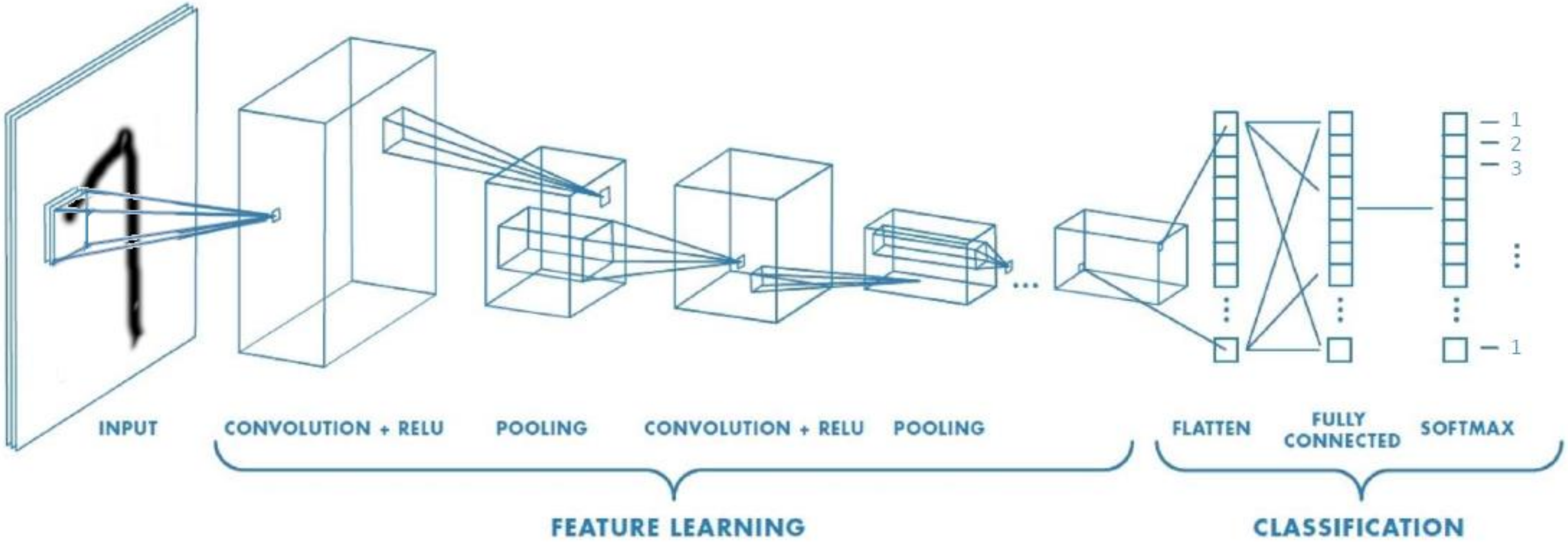
# Convolutional Neural Networks
## Building a CNN – Example




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
## Training with backpropagation




Learn weights for convolutional filters and fully connected layers

Backpropagation: cross-entropy loss

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

# Curriculum

**Next:**

- Hands-on II

# Imprint

| | |
|---|---|
| Topic: | **Introduction to Deep Learning**<br>Part II – Advanced Concept and Convolutional Neural Network |
| Date: | 2025-11-13 |
| Author: | Auliya Fitri, Sai Vemuri, Sreerag Naveenachandran |
| Institute: | Data Science |
| Image sources: | All images “DLR (CC BY-NC-ND 3.0)” unless otherwise stated |