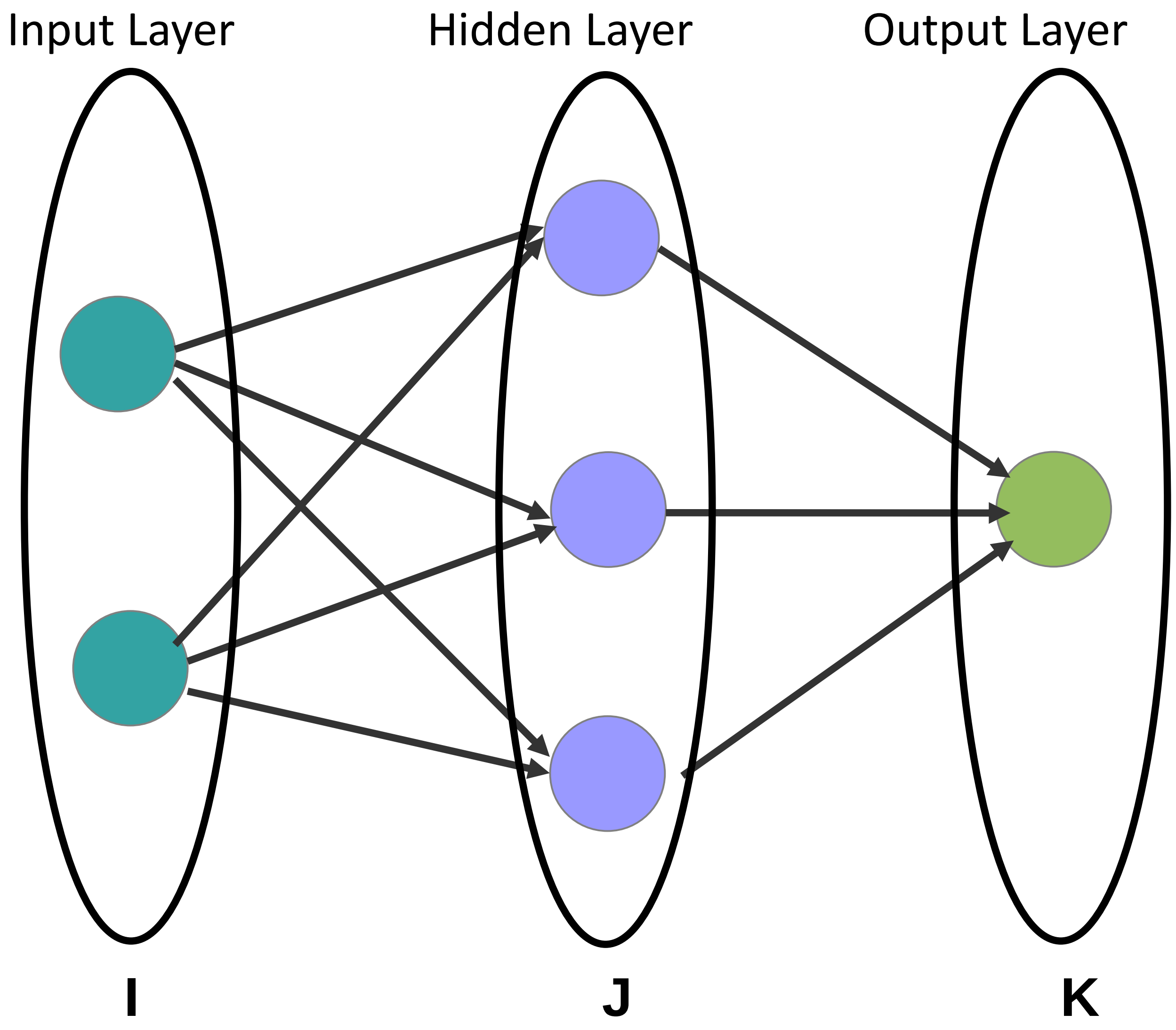
- All materials, slides, lecture notes, exercises are for personal use by course participants only.
- Copying and distribution of the material or of parts of it, by any means is strictly forbidden.

Error back-propagation algorithm

- Die Vorlesungs- und Übungsunterlagen sind ausschließlich für den Gebrauch in meinen Lehrveranstaltungen bestimmt! Es ist ausdrücklich nur die private Verwendung der Unterlagen für die Kursteilnehmer gestattet.
- Die Weitergabe der Unterlagen an Dritte, ihre Vervielfältigung oder Verwendung auch von Auszügen davon in anderen elektronischen oder gedruckten Publikationen ist nicht gestattet.

Artificial Neural Network



Backpropagation Algorithm

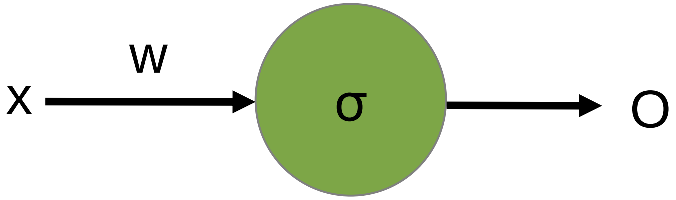
- Supervised learning algorithm.

- Provides a way to train a neural networks with any number of hidden units arranged in any number of layers.

- Trains multilayer feed-forward network by gradient descent using a weight adjustment based on the sigmoid function.

- For better understanding we divide the back-propagation learning algorithm into three phases:
    - Forward propagation
    - Error calculation
    - Weight updating by (backpropagation based on error calculation)

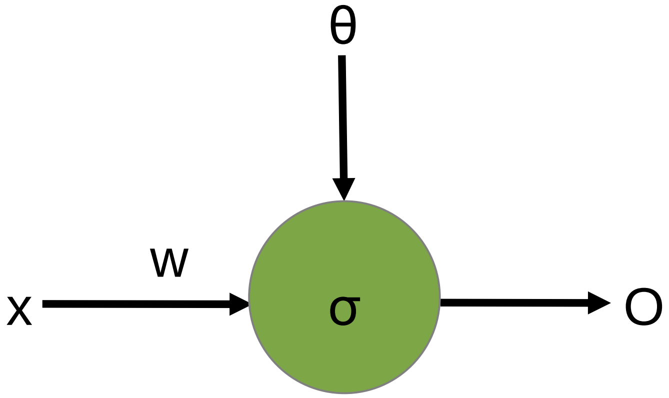
Relationship Among Input, Weight, Bias, Sigmoid Function and Output of a Neuron



$O = \sigma(xw)$

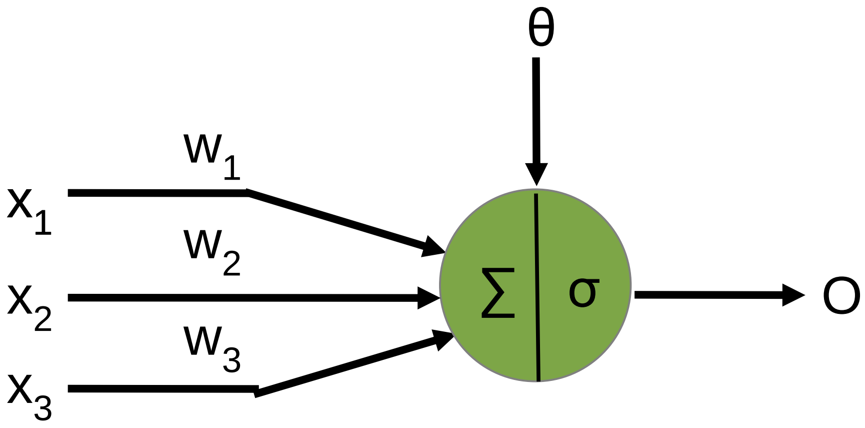
Single Input Neuron

x : input
w : weight
σ : sigmoid transfer function
θ : bias
O : output



$O = \sigma(xw + \theta)$

Single Input Neuron with bias



$O = \sigma(x_1w_1 + x_2w_2 + x_3w_3 + \theta)$

Multiple Input Neuron

More detailed notations

$\sigma(x) = 1/\left(1 + e^{-x}\right)$: *Sigmoid activation function*

X_j^l : *Input to node j of layer l*

W_{ij}^l : *Input weight to node j of layer l*

θ_j^l : *Bias for node j of layer l*

O_j^l : *Output of node j of layer l*

t_j: *Target value of node j of the output layer*

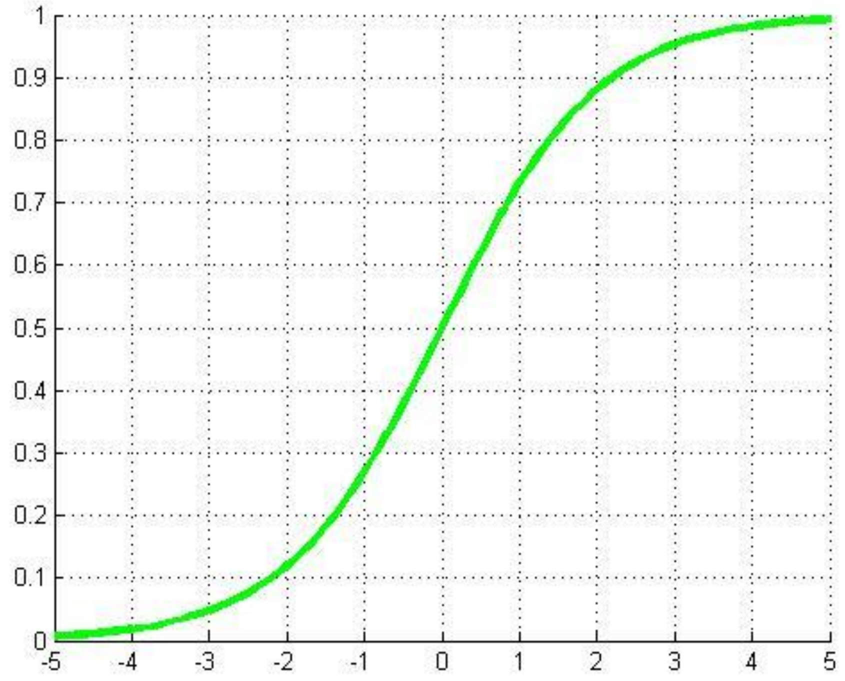
Sigmoid Activation Function

$$\sigma(x) = 1/\left(1 + e^{-x}\right)$$

- The sigmoid function is used as an activation function

- It is similar to the step function but it is linear, continuous, differentiable and increasing function.

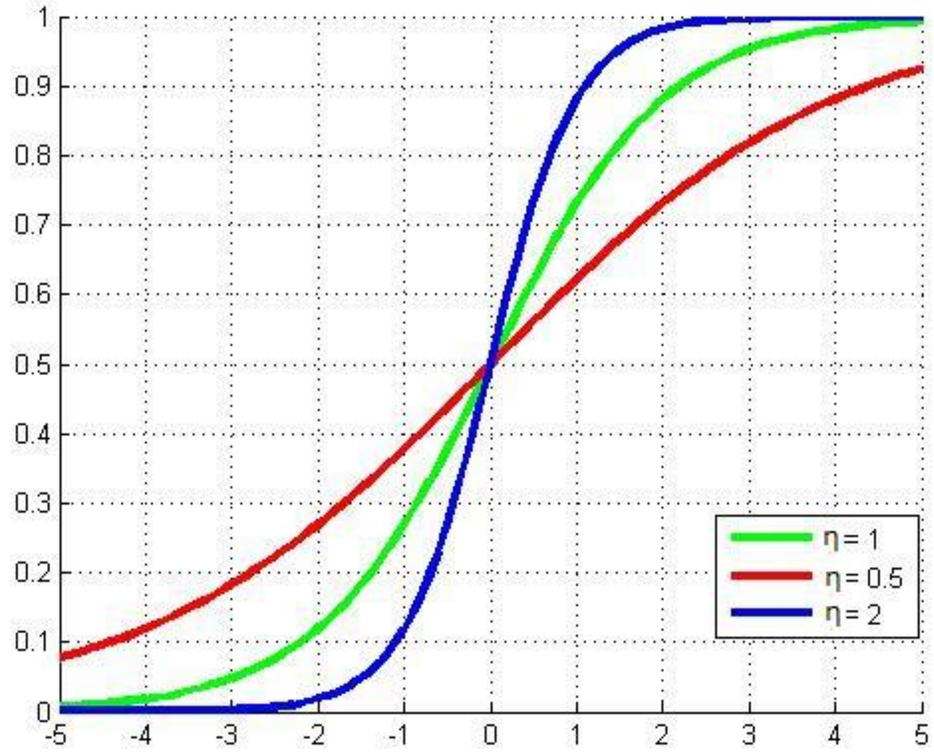
- Bounded range - but never reaches max or min



Sigmoid Activation Function

$$\sigma(x) = 1/\left(1 + e^{-\eta x}\right)$$

- Steepness η
- As the value of η increases the curve rises with faster rate.
- As the value of η decreases the curve rises with slower rate.
- A steeper function is equivalent to using a higher learning rate



Gradient Descent Method

- It calculates the first derivative of a function.

- The purpose is to find the local minima.

- This method is used to calculate the error of a neuron output based on the associated weight and for minimizing the error.

- The target is to minimize the output error as much as possible.

- Application of gradient descent method in backpropagation algorithm.

Derivative of Sigmoid Activation Function

$$
\begin{aligned}
\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) \\
&= \sigma(x) - \sigma(x)^2 \\
\sigma' &= \sigma(1-\sigma)
\end{aligned}
$$

Steps of a Backpropagation Algorithm

- Propagate forward to calculate the output
- Calculating the error at each node of each layer (output and hidden layer)
- Propagate backward and update the associated weights

Error Calculation

- Given a set of training data points t_k (t: target values or expected output) and output layer output O_k we can write the squared error as:

$$E = \frac{1}{2}\sum_{k \in K}(\mathcal{O}_k - t_k)^2$$

- We want to calculate $\boldsymbol{\partial E / \partial W_{jk}}$, the rate of change of the error with respect to the given connective weight, so that we can minimize it.

- Now we consider two cases:
    - Is the node an output node ?
    - Is it one of the hidden node ?

Error calculation at output layer

$$\frac{\partial E}{\partial W_{jk}} = \frac{\partial}{\partial W_{jk}} \frac{1}{2} \sum_{k \in K} (\mathcal{O}_k - t_k)^2$$

$$\frac{\partial E}{\partial W_{jk}} = (\mathcal{O}_k - t_k) \frac{\partial}{\partial W_{jk}} \sigma(x_k)$$

$$\frac{\partial E}{\partial W_{jk}} = (\mathcal{O}_k - t_k)\mathcal{O}_k(1 - \mathcal{O}_k)\mathcal{O}_j$$

We can simplify the equation:

$$\frac{\partial E}{\partial W_{jk}} = \mathcal{O}_j \delta_k$$

where, $\delta_k = \mathcal{O}_k(1 - \mathcal{O}_k)(\mathcal{O}_k - t_k)$

Error calculation at hidden layer

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}} \frac{1}{2} \sum_{k \in K} (\mathcal{O}_k - t_k)^2$$

$$\frac{\partial E}{\partial W_{ij}} = \sum_{k \in K} (\mathcal{O}_k - t_k) \sigma(x_k)(1 - \sigma(x_k)) \frac{\partial x_k}{\partial W_{ij}}$$

$$\frac{\partial E}{\partial W_{ij}} = \sum_{k \in K} (\mathcal{O}_k - t_k) \mathcal{O}_k (1 - \mathcal{O}_k) W_{jk} \frac{\partial \mathcal{O}_j}{\partial W_{ij}}$$

$$\frac{\partial E}{\partial W_{ij}} = \mathcal{O}_j (1 - \mathcal{O}_j) \mathcal{O}_i \sum_{k \in K} (\mathcal{O}_k - t_k) \mathcal{O}_k (1 - \mathcal{O}_k) W_{jk}$$

Now, we can simplify the equation as:

$$\frac{\partial E}{\partial W_{ij}} = \mathcal{O}_i \mathcal{O}_j (1 - \mathcal{O}_j) \sum_{k \in K} \delta_k W_{jk} \quad = \mathcal{O}_i \delta_j$$

Where, $\delta_j = \mathcal{O}_j (1 - \mathcal{O}_j) \sum_{k \in K} \delta_k W_{jk}$

Error calculation for bias term

Incorporating the bias term ϑ into the equation leads to:

$$\frac{\partial \mathcal{O}}{\partial \theta} = \mathcal{O}(1 - \mathcal{O})\frac{\partial \theta}{\partial \theta}$$

and because of ∂θ/∂θ = 1 the bias term as output from a node which is always one.

This holds for any layer *l* we are concerned with, a substitution into the previous equations gives us that

$$\frac{\partial E}{\partial \theta} = \delta_\ell$$

Weight and bias update

- According to the previous equations we can say that,

$$\Delta W = \delta_l * O_{l-1}, \text{ for any layer}$$

- If we incorporate variable learning rate (η),

$$\Delta W = \eta * \delta_l * O_{l-1}$$

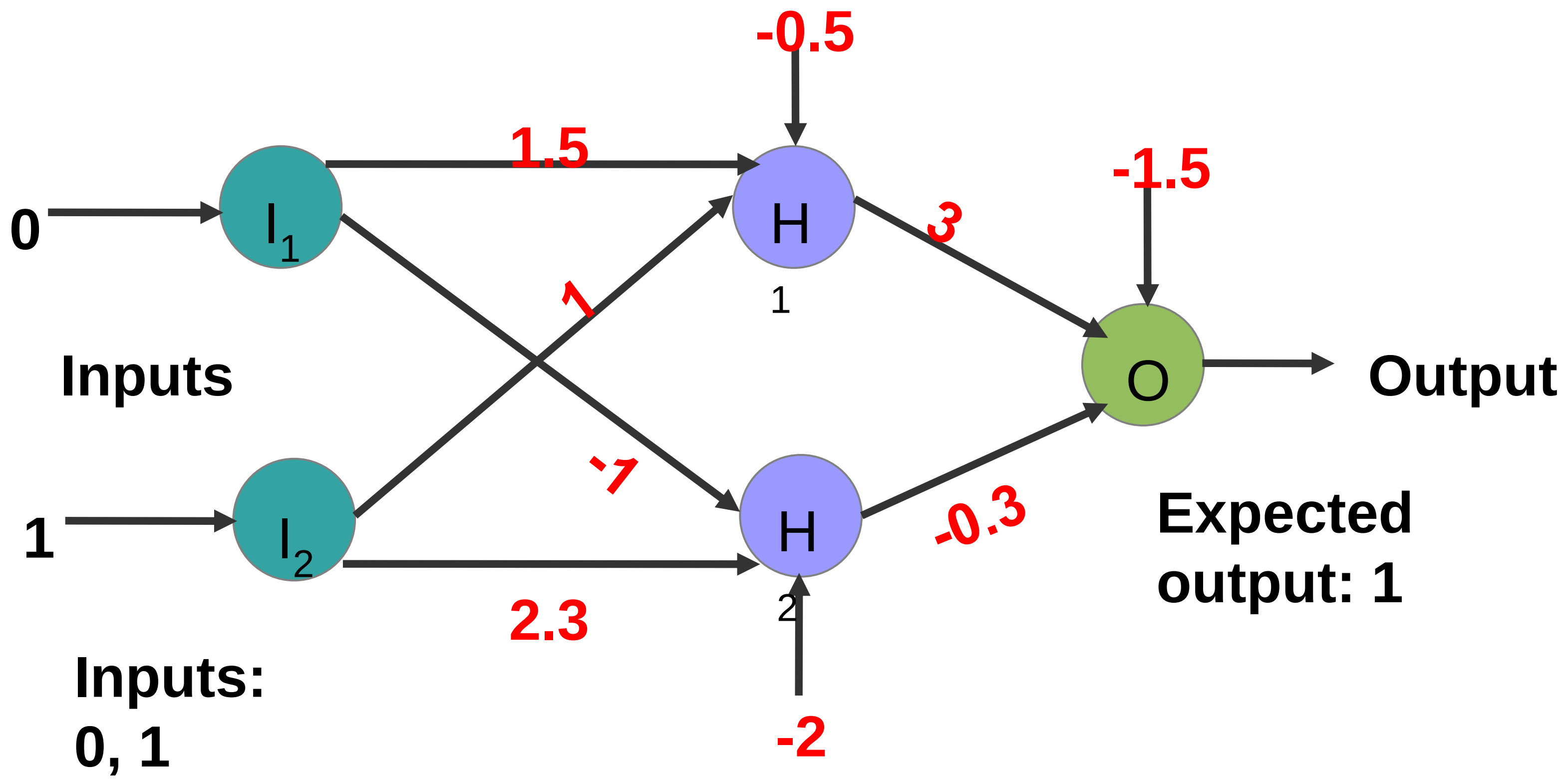
- And for bias factor,

$$\Delta\vartheta = -\eta * \delta_l$$

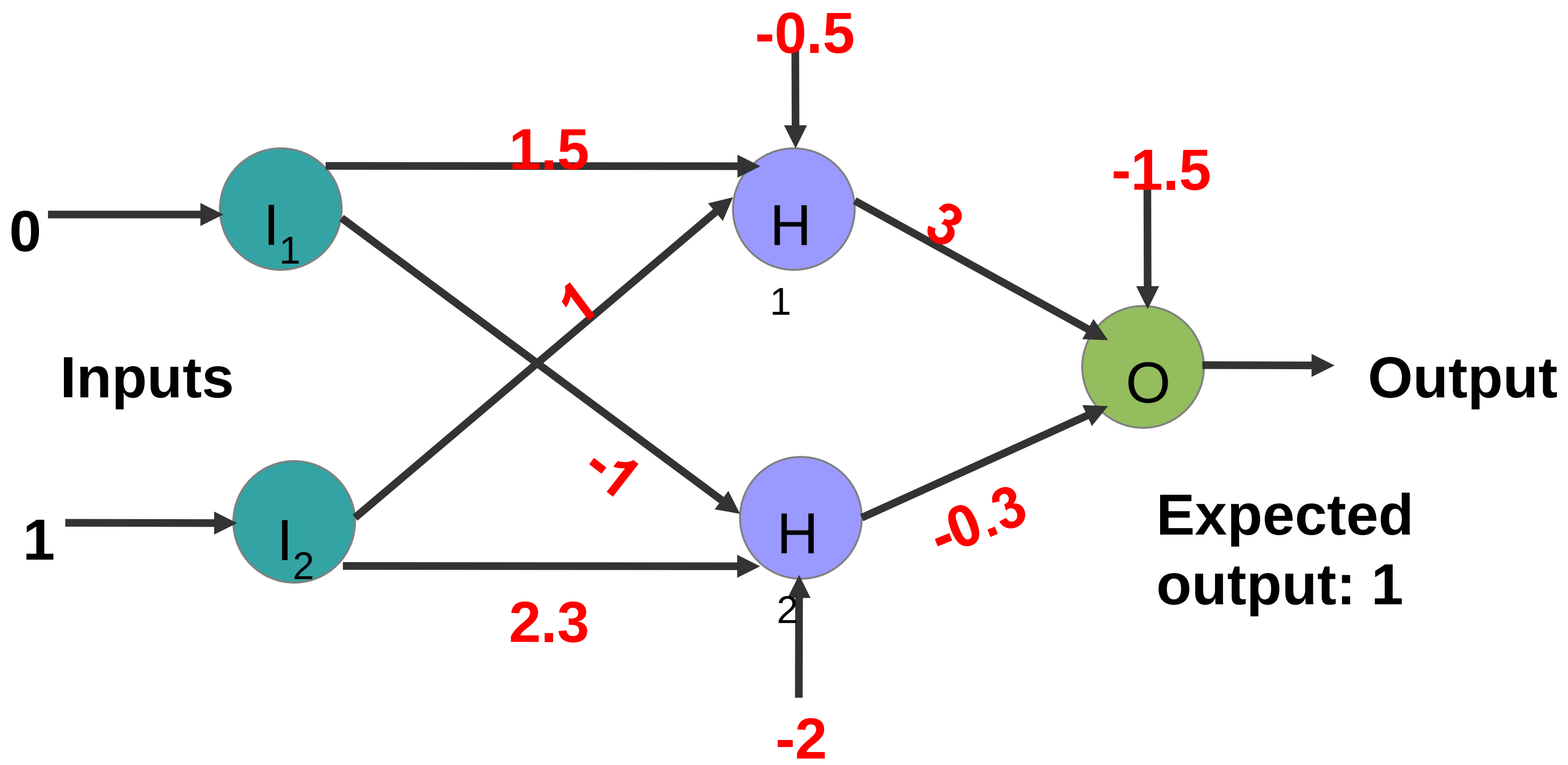
So, the updated weight equation is: $W = W + \Delta W$
And the updated bias equation is: $\theta = \theta + \Delta\theta$

Realizing Backpropagation Algorithm with XOR Problem

- For two inputs 0, 1 the expected output is 1
- At first weights and biases are randomly assigned in the network
- We will calculate the output and verify with the expected value
- If the output is not equal to the expected value the weights and biases have to be updated



Realizing Backpropagation Algorithm with XOR Problem



H_1: net = 0 * (1.5) + 1 * (1) – 0.5 = 0.5 **contributes to output = 1 / (1 + $e^{-0.5}$) = 0.622**

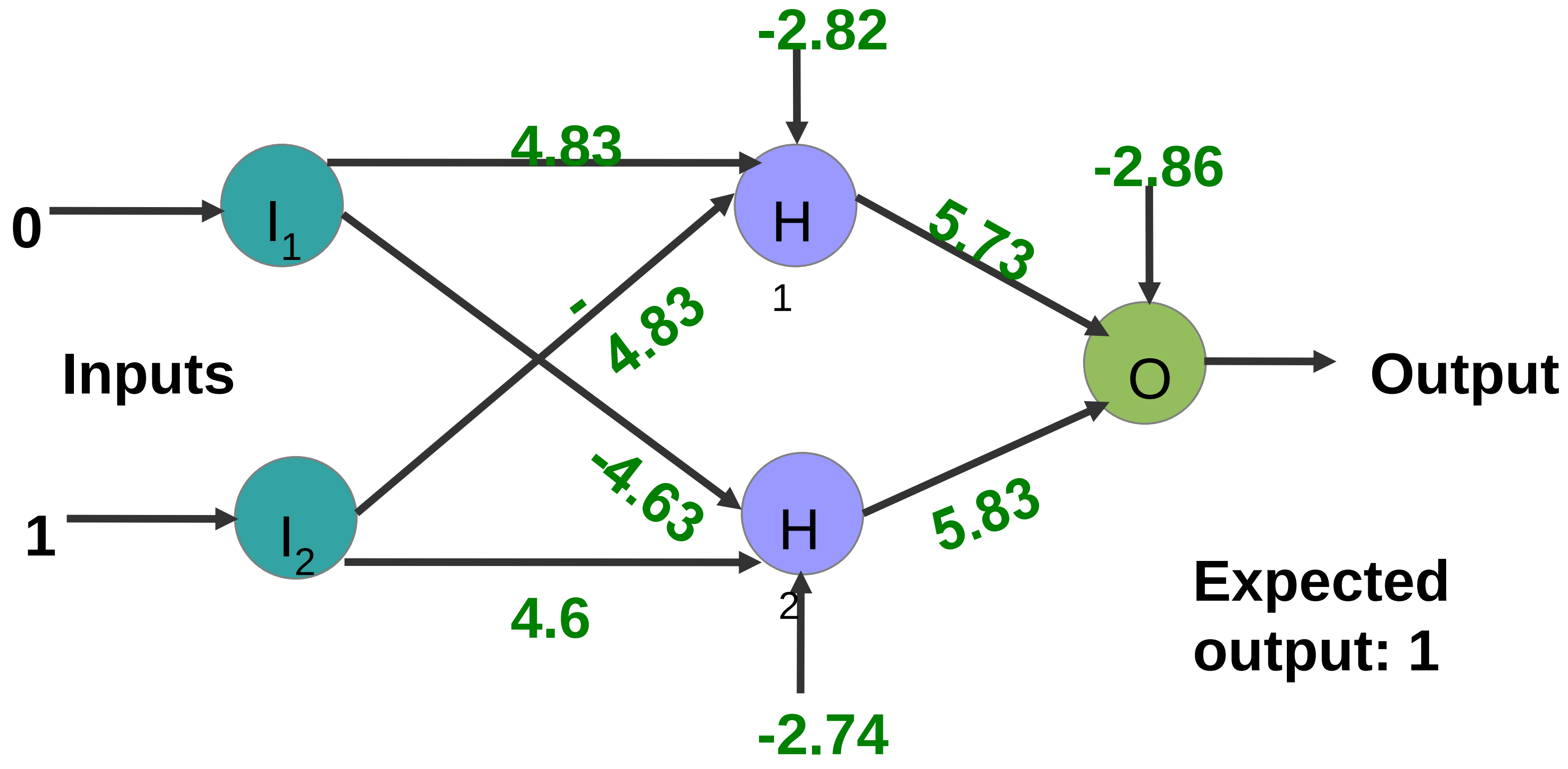
H_2: net = 0 * (-1) + 1 * (2.3) – 2 = 0.3 **contributes to output = 1 / (1 + $e^{-0.3}$) = 0.574**

output: net = 0.622 * (3) + 0.574 * (-0.3) – 1.5 = 0.194

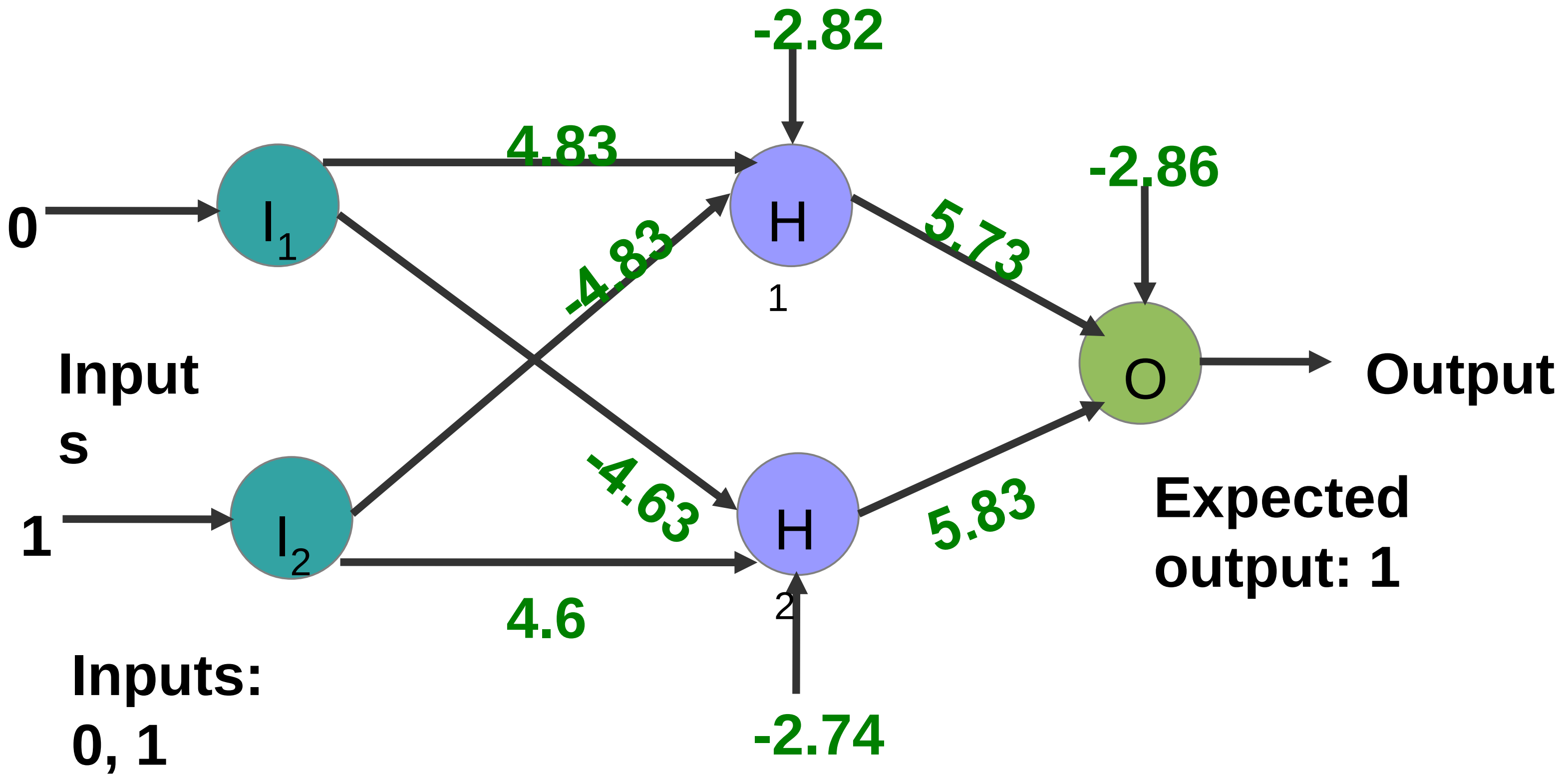
activated output = 1 / (1 + $e^{-0.194}$) = 0.548

Realizing Backpropagation Algorithm with XOR Problem

After weight and bias updating using backpropagation algorithm



Realizing Backpropagation Algorithm with XOR Problem



H_1: net = 0 * (4.83) + 1 * (-4.83) – 2.82 = -7.65 **contribution to output = 1 / (1 + $e^{7.65}$) = 4.758x10^{-4}**

H_2: net = 0 * (-4.63) + 1 * (4.6) – 2.74 = 1.86 **contribution to output = 1 / (1 + $e^{-1.86}$) = 0.8652**

output: net = 4.758x10^{-4} * (5.73) + 0.8265 * (5.83) – 2.86 = 2.187

activated output = 1 / (1 + $e^{-2.187}$) = 0.8991 ≡ 1

Summarized backpropagation algorithm

- Run the network forward with your input data to get the network output
- For each output node calculate $\delta_k = \mathcal{O}_k(1 - \mathcal{O}_k)(\mathcal{O}_k - t_k)$
- For each hidden node calculate $\delta_j = \mathcal{O}_j(1 - \mathcal{O}_j)\sum_{k \in K} \delta_k W_{jk}$
- Update the weights and biases as follows:

$$\boldsymbol{W} = \boldsymbol{W} + \Delta \boldsymbol{W}$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$$

where, $\Delta W = \eta * \delta_l * O_{l-1}$ *and* $\Delta \vartheta = -\eta * \delta_l$

- Repeat the whole process until the satisfactory result is achieved, e.g. error below a limit set

Issue: number of hidden layers and neurons in an ANN

- For many problems, one hidden layer is sufficient

- Two hidden layers are required when the function is discontinuous

- The number of neurons is very important:
  - Too few: Under-fit the data – NN can not learn the details
  - Too many: Over-fit the data – NN learns the insignificant details

- Start with small and increase the number until satisfactory results are obtained

Issue: number of hidden layers and neurons in an ANN

There are many rules for determining the correct number of neurons to use in the hidden layers, such as the following:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer

- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer

- The number of hidden neurons should be less than twice the size of the input layer.

- These three rules provide a starting point of consideration

- Ultimately, the selection of an architecture for neural network will often come down to trial and error measurement

Drawbacks of Back-Propagation Algorithm

- The convergence obtained from error back-propagation learning is slow for more layers – vanishing gradient problem

- It is hard to know how many neurons and layers are necessary

- The convergence in backpropagation learning is not guaranteed

- The result may generally converge to any local minimum on the error surface
- Further reading and improvement: “Stochastic gradient descent algorithm” and “Levenberg-Marquardt algorithm”

- Die Vorlesungs- und Übungsunterlagen sind ausschließlich für den Gebrauch in meinen Lehrveranstaltungen bestimmt! Es ist ausdrücklich nur die private Verwendung der Unterlagen für die Kursteilnehmer gestattet.
- Die Weitergabe der Unterlagen an Dritte, ihre Vervielfältigung oder Verwendung auch von Auszügen davon in anderen elektronischen oder gedruckten Publikationen ist nicht gestattet.

Machine Learning Evaluation Metrics

Metrics for classification: confusion matrix

		True condition	
	Total population	Condition positive	Condition negative
Predicted condition	Predicted condition positive	**True positive**	**False positive**, Type I error
	Predicted condition negative	**False negative**, Type II error	**True negative**

From: https://en.wikipedia.org/wiki/Sensitivity_and_specificity

Example:
Covid-19 test

- **True Positive**: A person is tested as Covid-19 positive and actually has the Covid-19 disease.
- **True Negative**: A person is tested as Covid-19 negative and actually doesn't have the Covid-19 disease.
- **False Positives (FP):** A person is tested as Covid-19 positive and actually doesn't have the Covid-19 disease. (Also known as a "Type I error.")
- **False Negatives (FN):** A person is tested as Covid-19 negative, but actually has the Covid-19 disease. (Also known as a "Type II error.").

Metrics for classification: confusion matrix

		True condition				
	Total population	Condition positive	Condition negative	Prevalence $= \frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) $= \frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**	**False positive**, Type I error	Positive predictive value (PPV), Precision $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$	
Predicted condition	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection, Power $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR−}}$	F_1 score $= 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
		False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) $= \frac{\text{FNR}}{\text{TNR}}$		

From: https://en.wikipedia.org/wiki/Sensitivity_and_specificity

Metrics for classification: confusion matrix

		True condition				
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$	
Predicted condition	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection, Power $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR−}}$	F_1 score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
		False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$		

From: https://en.wikipedia.org/wiki/Sensitivity_and_specificity

- **Accuracy:** the proportion of the total number of predictions that were correct.
- **Positive Predictive Value or Precision:** the proportion of positive cases that were correctly identified.
- **Negative Predictive Value:** the proportion of negative cases that were correctly identified.
- **Sensitivity or Recall:** the proportion of actual positive cases that are correctly identified.
- **Specificity:** the proportion of actual negative cases that are correctly identified.

Metrics for classification: recall, specificity, precision

sensitivity, recall, hit rate, or true positive rate (TPR)

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{FNR}$$

specificity, selectivity or true negative rate (TNR)

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} = 1 - \mathrm{FPR}$$

precision or positive predictive value (PPV)

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} = 1 - \mathrm{FDR}$$

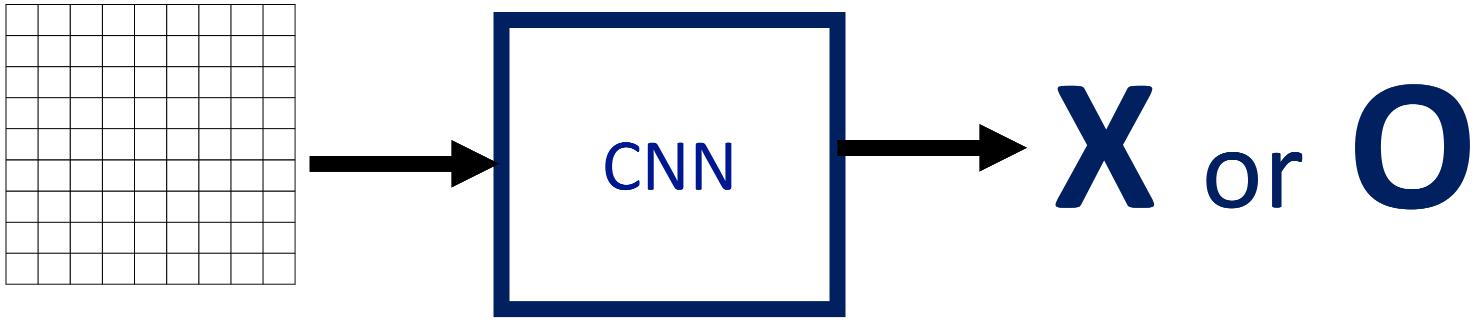
Recall (sensitivity): ratio of positive class correctly detected

Specificity: ratio of actual negatives

Precision: shows the accuracy of positive class

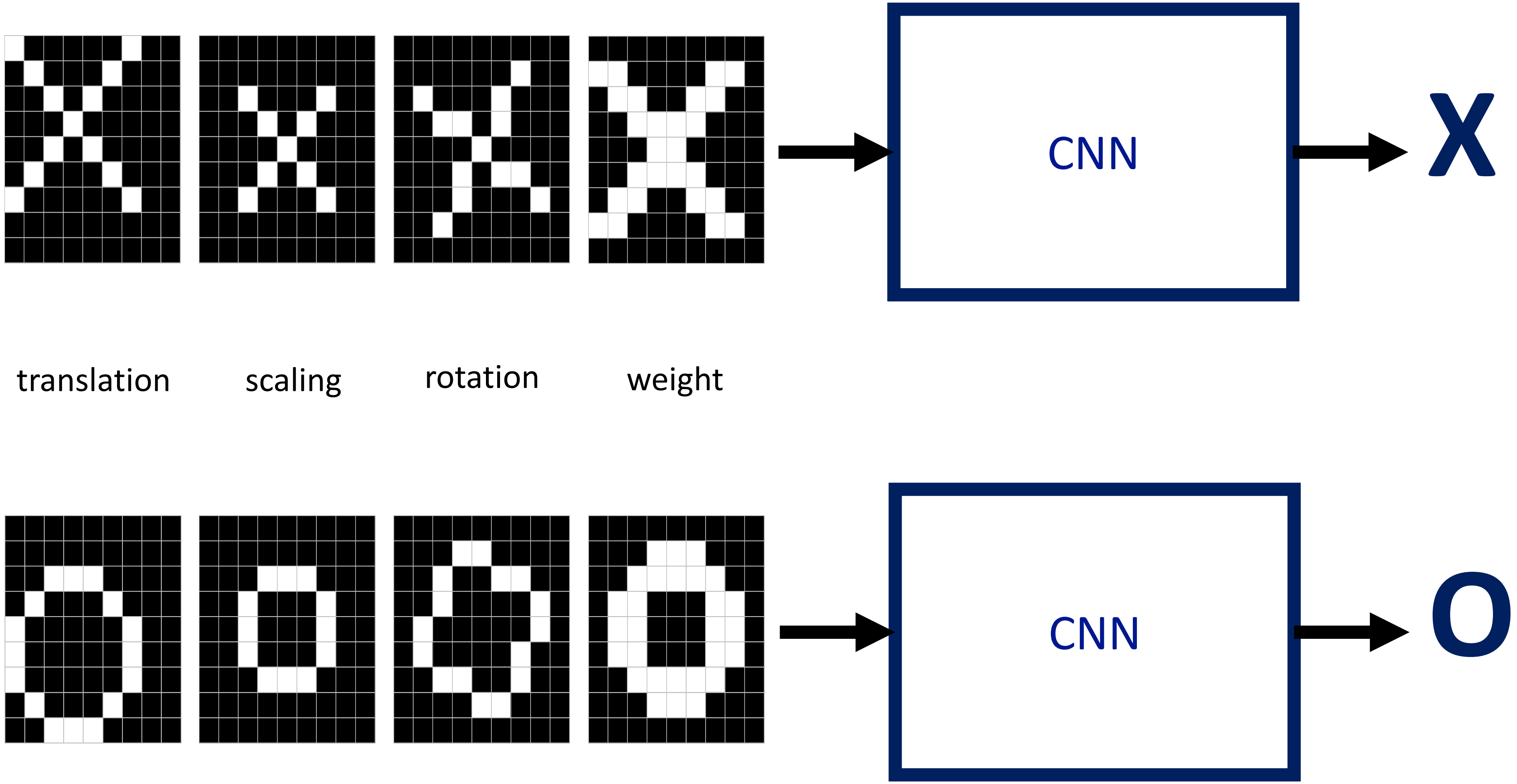
Convolutional Neural Network (CNN)

How does a Convolutional Neural Network work?

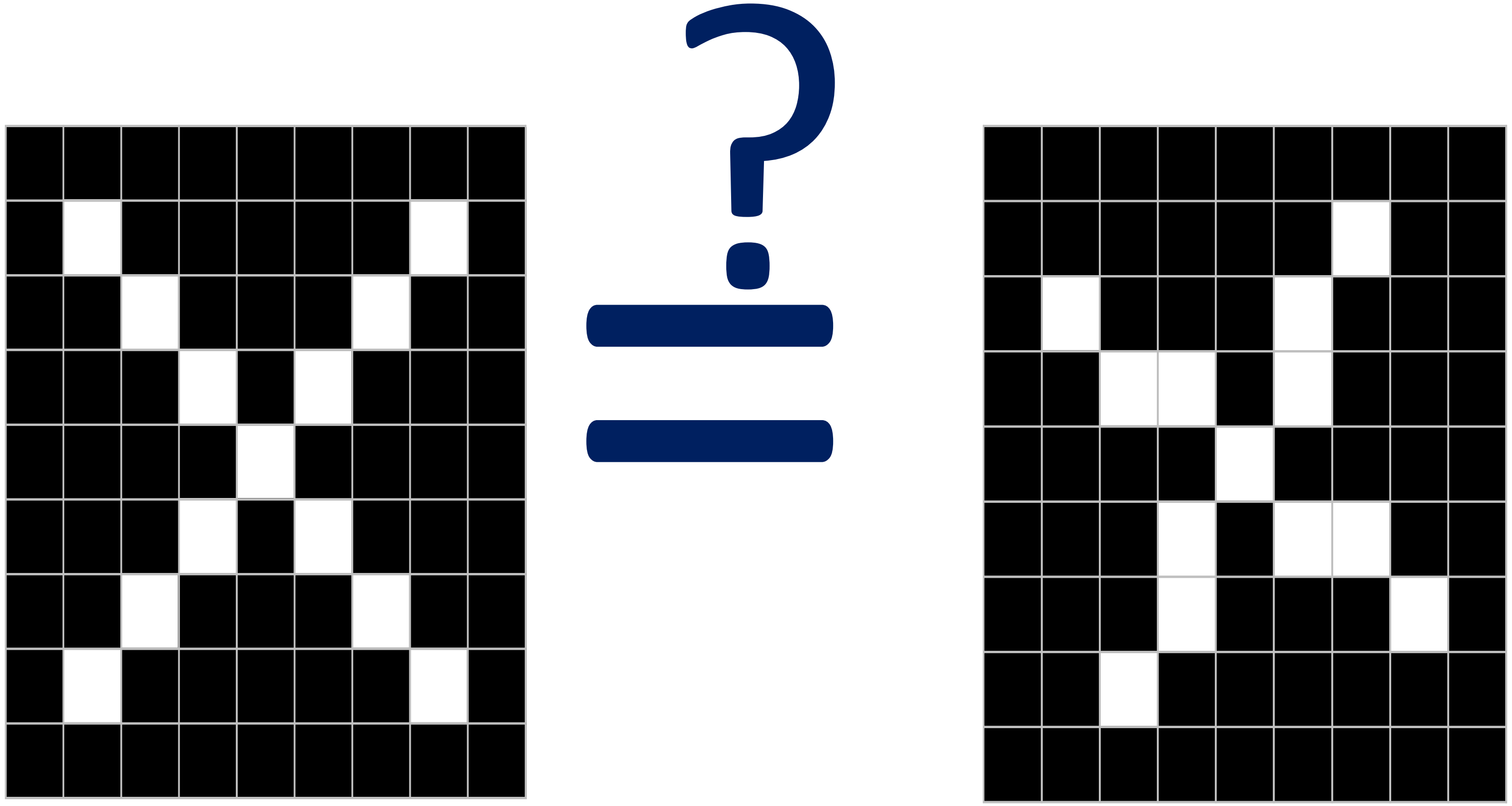


A two-dimensional array of pixels

Some cases



Deciding is hard



Data representation

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

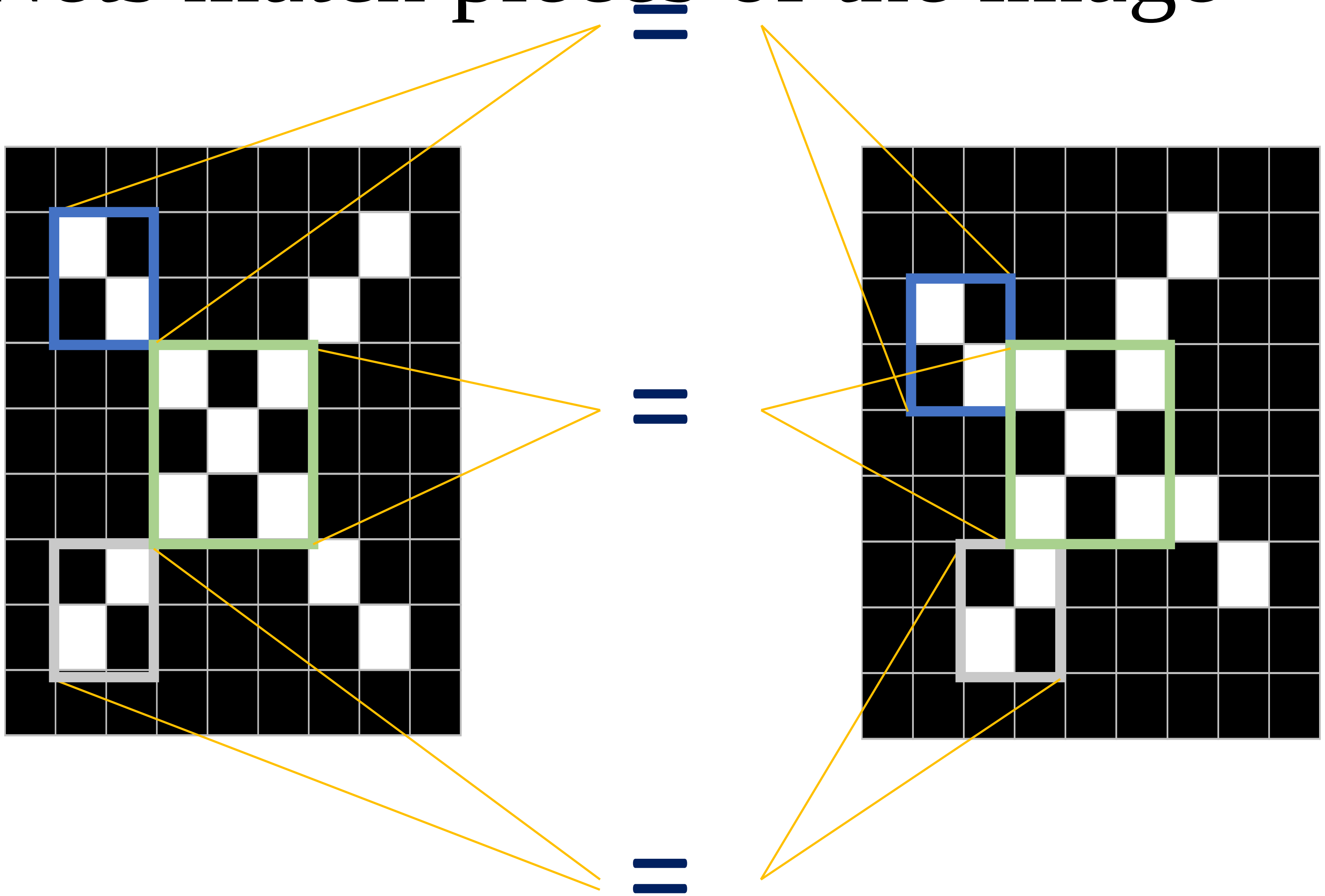
$$\stackrel{?}{=}$$

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

Matching totally?

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

ConvNets match pieces of the image

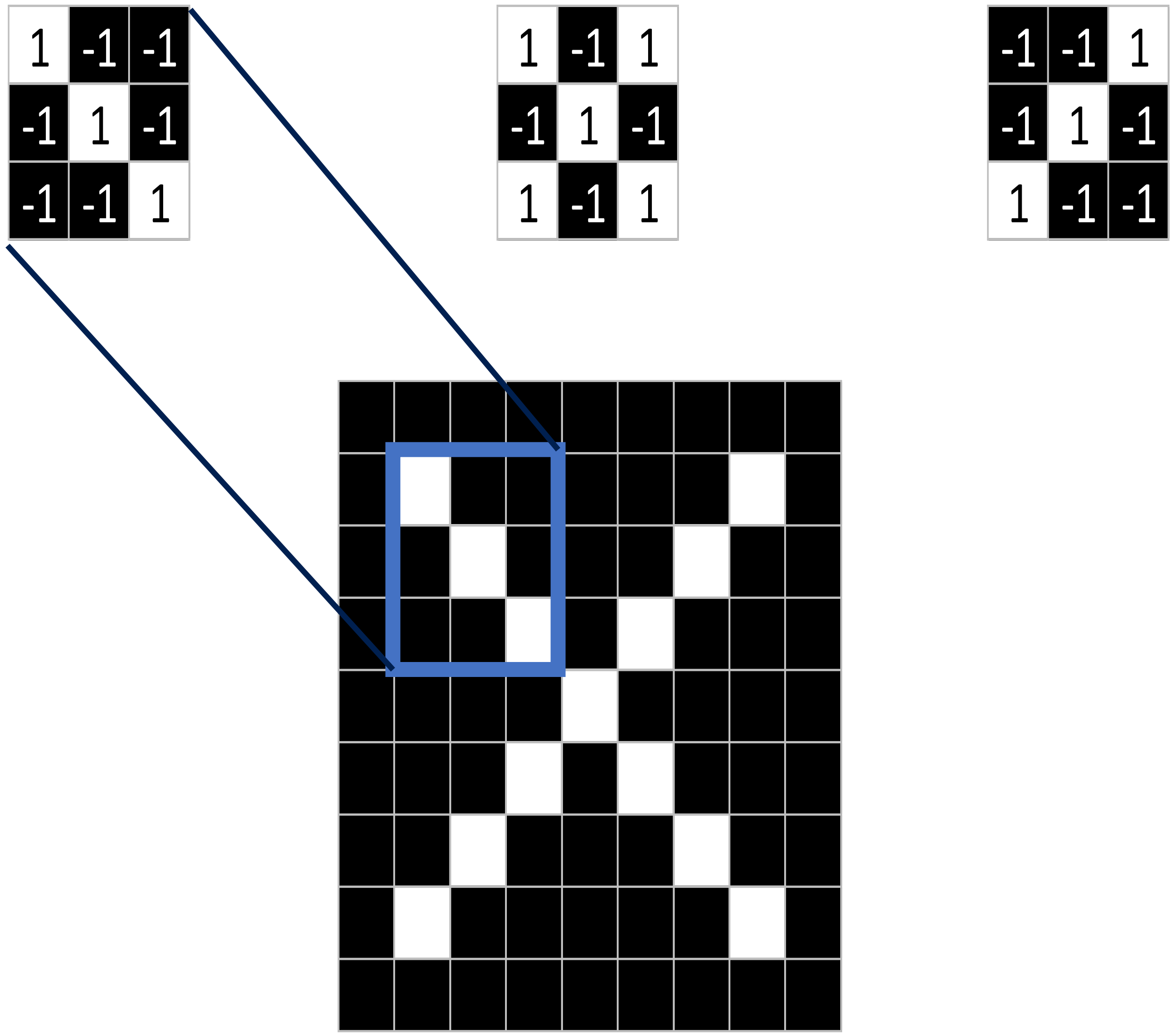


Features match pieces of the image

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

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

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


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

Filtering: The math behind the match

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

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

Filtering: The math behind the match

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	1	1
1	1	1
1	1	1

Filtering: The math behind the match

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

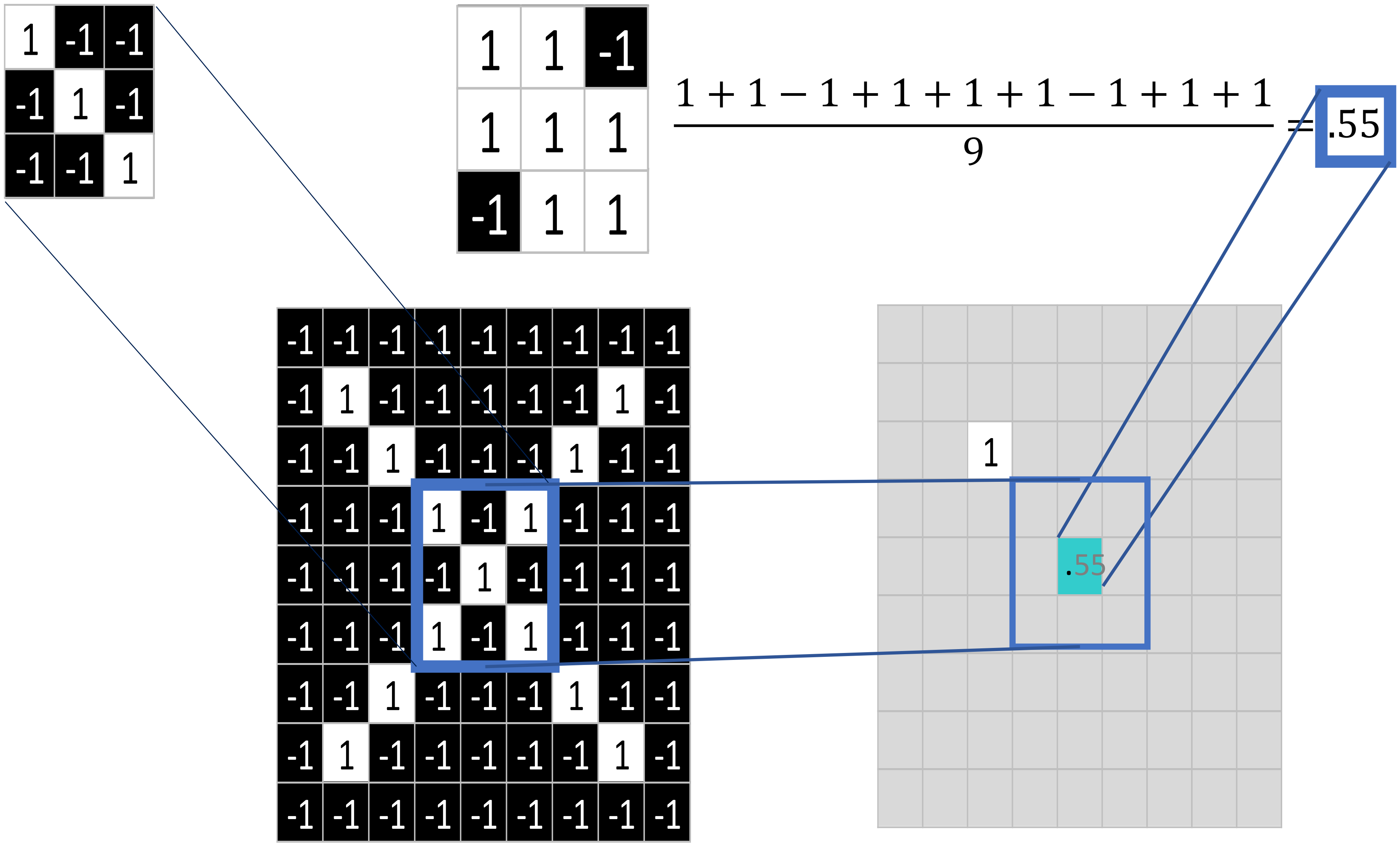
1	1	1
1	1	1
1	1	1

$$\frac{1+1+1+1+1+1+1+1+1}{9} = \boxed{1}$$

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

	1							

Filtering: The math behind the match



$$\frac{1+1-1+1+1+1-1+1+1}{9} = .55$$

Convolution: Trying every possible match

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

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

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

Convolution: Trying every possible match

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

$\otimes$

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

Pooling

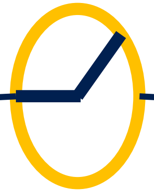
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

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

Rectified Linear Units (ReLUs)

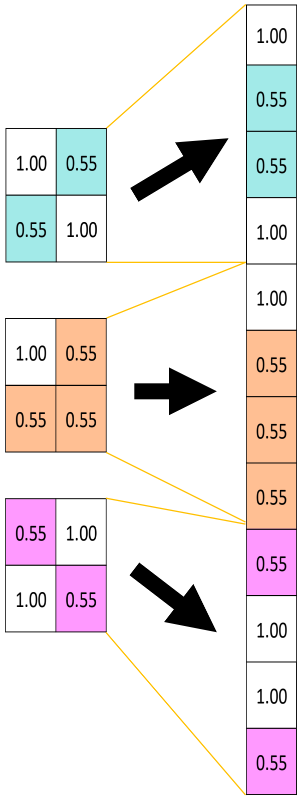
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77



0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

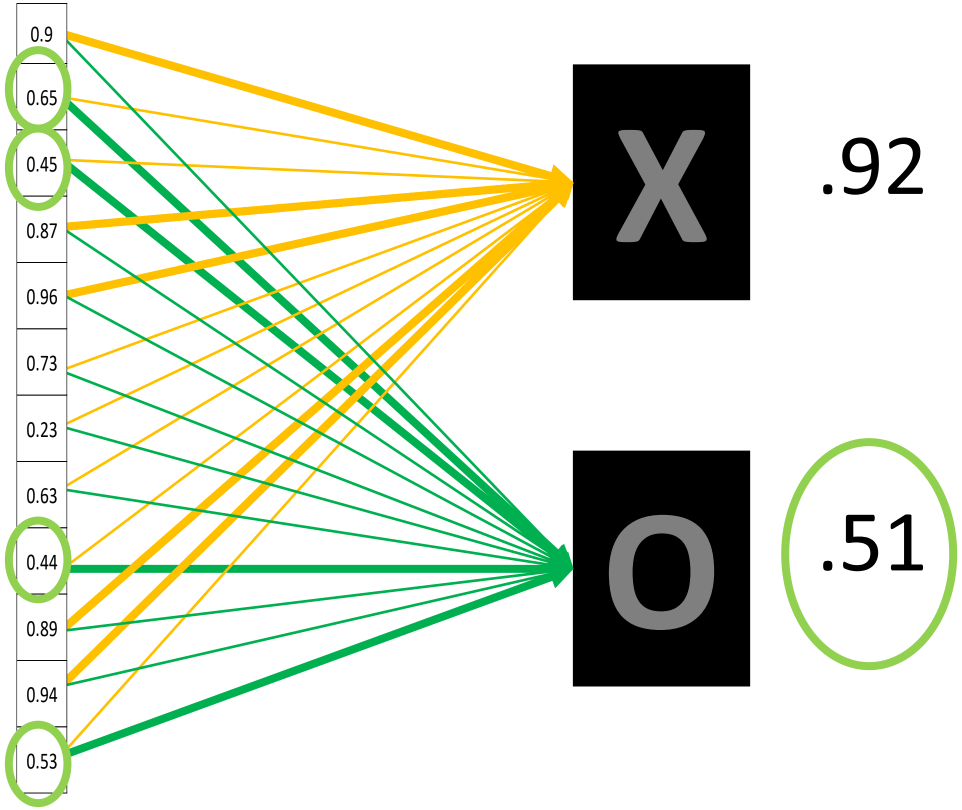
Fully connected layer

Every value gets a vote



Fully connected layer

Vote on X or O



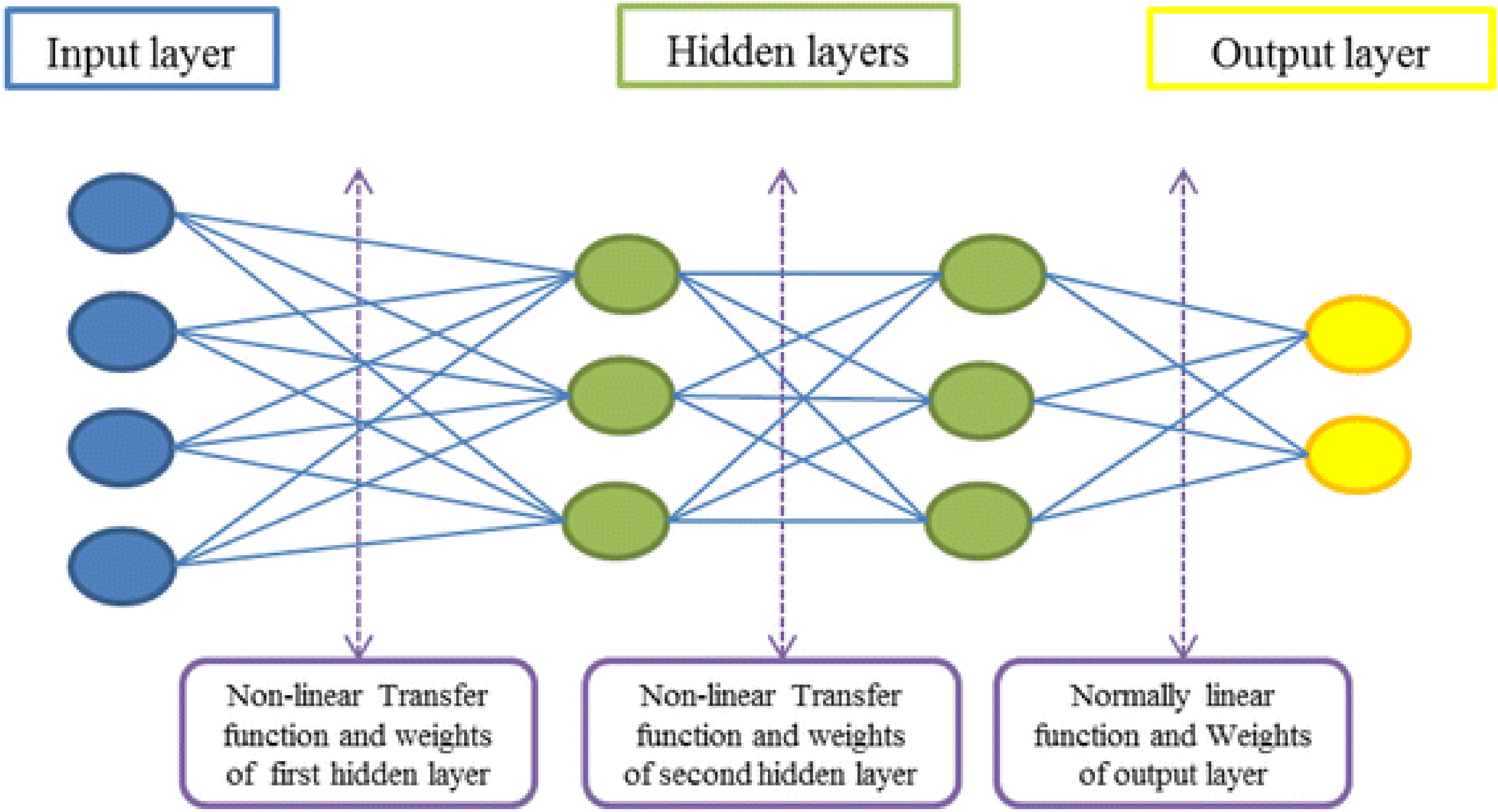
Putting it all together

A set of pixels becomes a set of votes.

Limitations

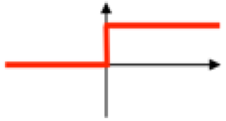
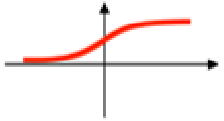
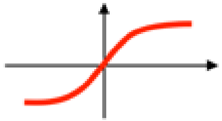
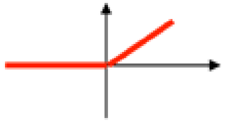
ConvNets only capture local “spatial” patterns in data.

Neurons in artificial intelligence
Multilayer perceptron



https://media.springernature.com/full/springer-static/image/art%3A10.1186%2Fs40201-015-0227-6/MediaObjects/40201_2015_227_Fig2_HTML.gif

Neurons in artificial intelligence
Multilayer perceptron

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	

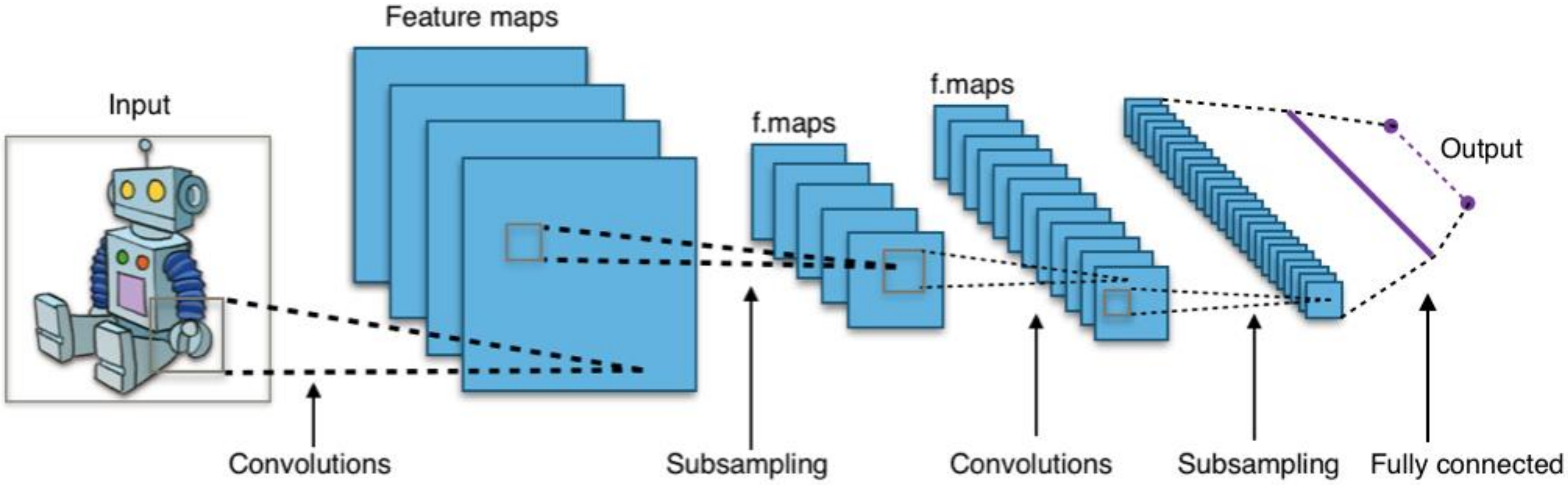
http://rasbt.github.io/mlxtend/user_guide/general_concepts/activation-functions

Convolutional Neural Network
Field of application

CNNs are often used for

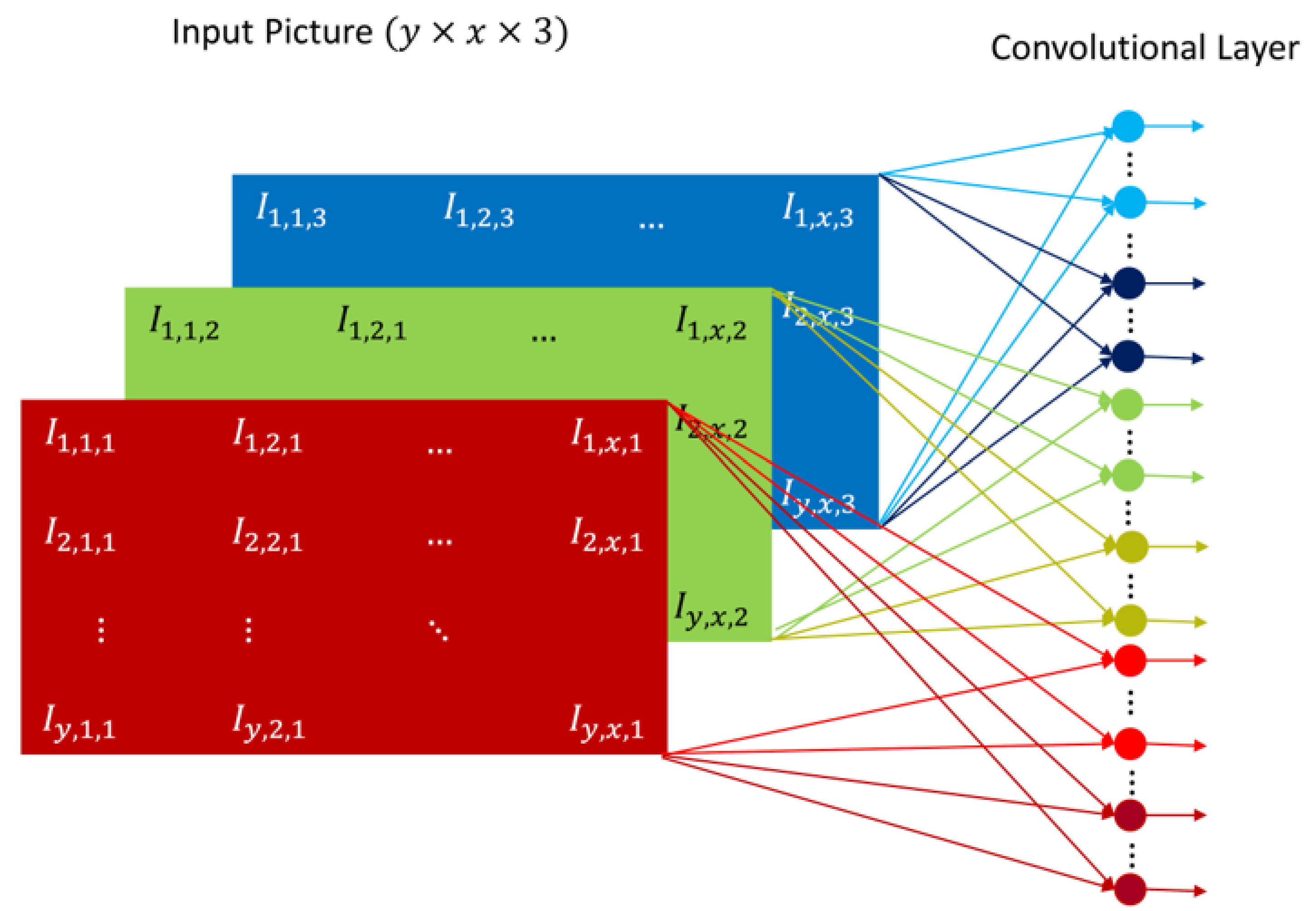
- Object recognition and classification in robots and autonomous systems
- Face recognition
- Object recognition in augmented reality
- handwriting recognition
- traffic sign recognition in cars
- speech recognition

Convolutional Neural Network
Design and function

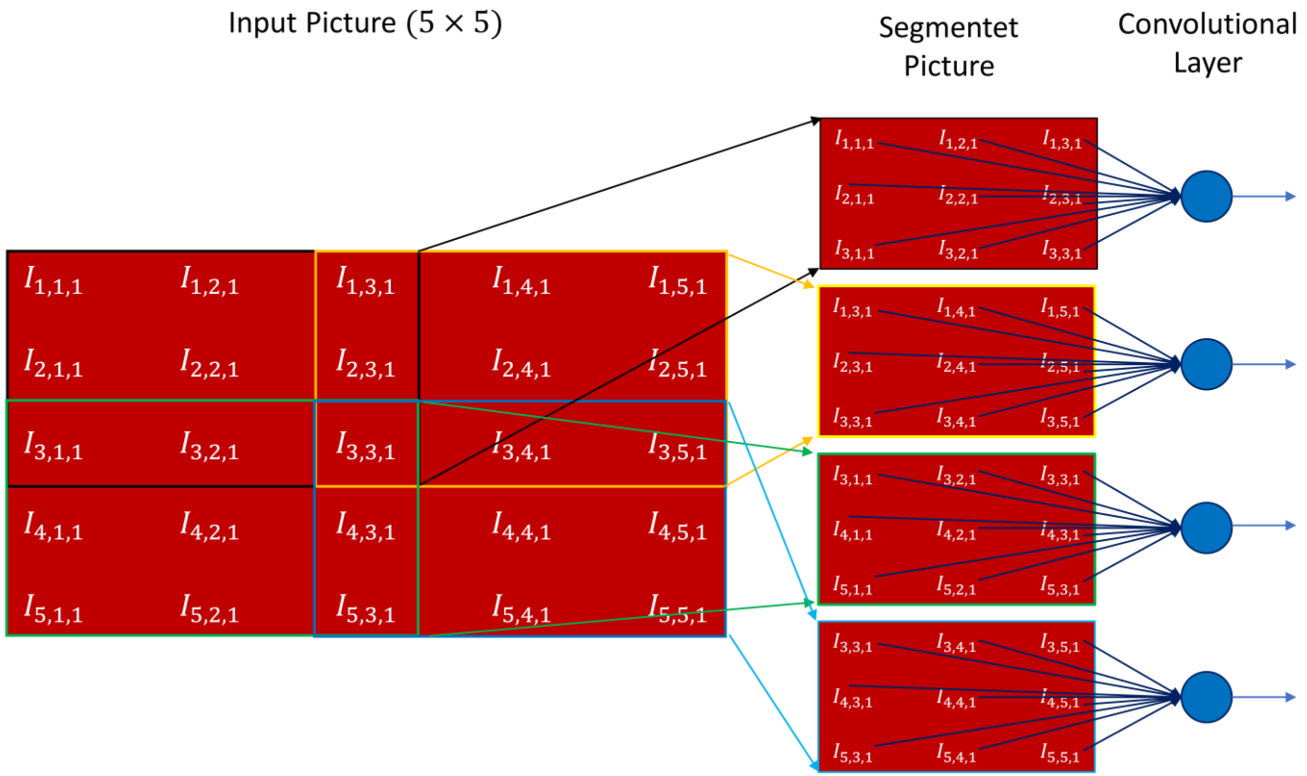


https://jaai.de/wp-content/uploads/2018/02/Typical_cnn.png

Convolutional Neural Network
Design and function - convolution



Convolutional Neural Network
Design and function - convolution



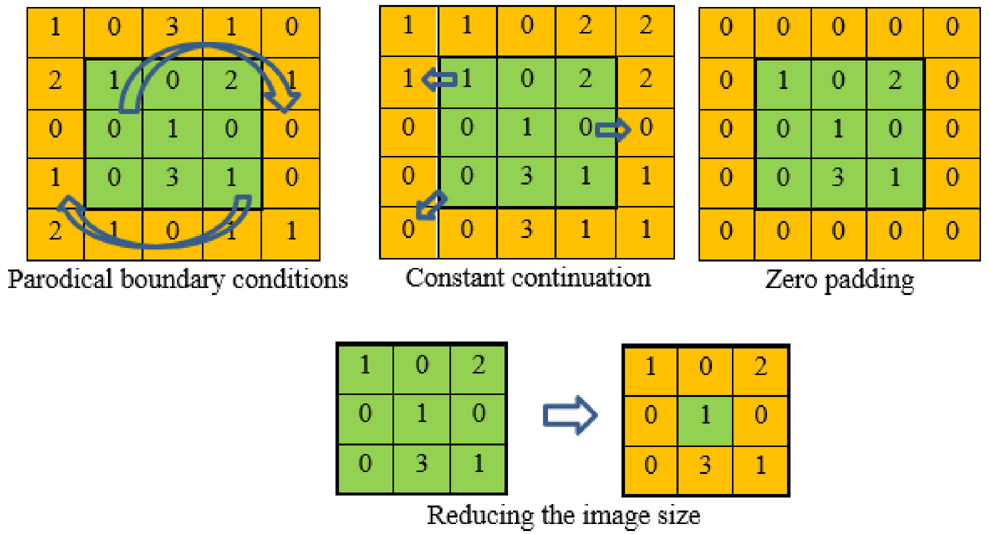
Convolutional Neural Network
Design and function - convolution

Input Picture (5x5) Filter (3x3) Output (2x2)

$$\begin{pmatrix} 1 & 4 & 3 & 4 & 2 \\ 5 & \boxed{2} & 8 & \boxed{7} & 3 \\ 1 & 4 & 2 & 8 & 0 \\ 3 & \boxed{6} & 2 & \boxed{2} & 7 \\ 5 & 4 & 1 & 1 & 4 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 \\ 0 & \boxed{-1} & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & -4 \\ -4 & 4 \end{pmatrix}$$

$$\sum_{i=-r}^{r} \sum_{j=-r}^{r} g(i,j) * u_1(m_1 - i, m_2 - j) = u_2(m_1, m_2) \qquad \textbf{(3)}$$

Convolutional Neural Network
Design and function – boundary conditions



Parodical boundary conditions

Constant continuation

Zero padding

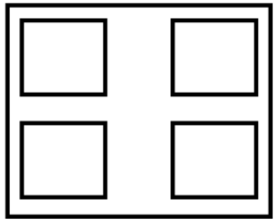
Reducing the image size

Convolutional Neural Network Design and function – pooling/subsampling

input matrix

$$\begin{pmatrix} 1 & 5 & 3 & 5 \\ 7 & 2 & 3 & 4 \\ 9 & 3 & 1 & 6 \\ 4 & 2 & 2 & 8 \end{pmatrix} \quad *$$

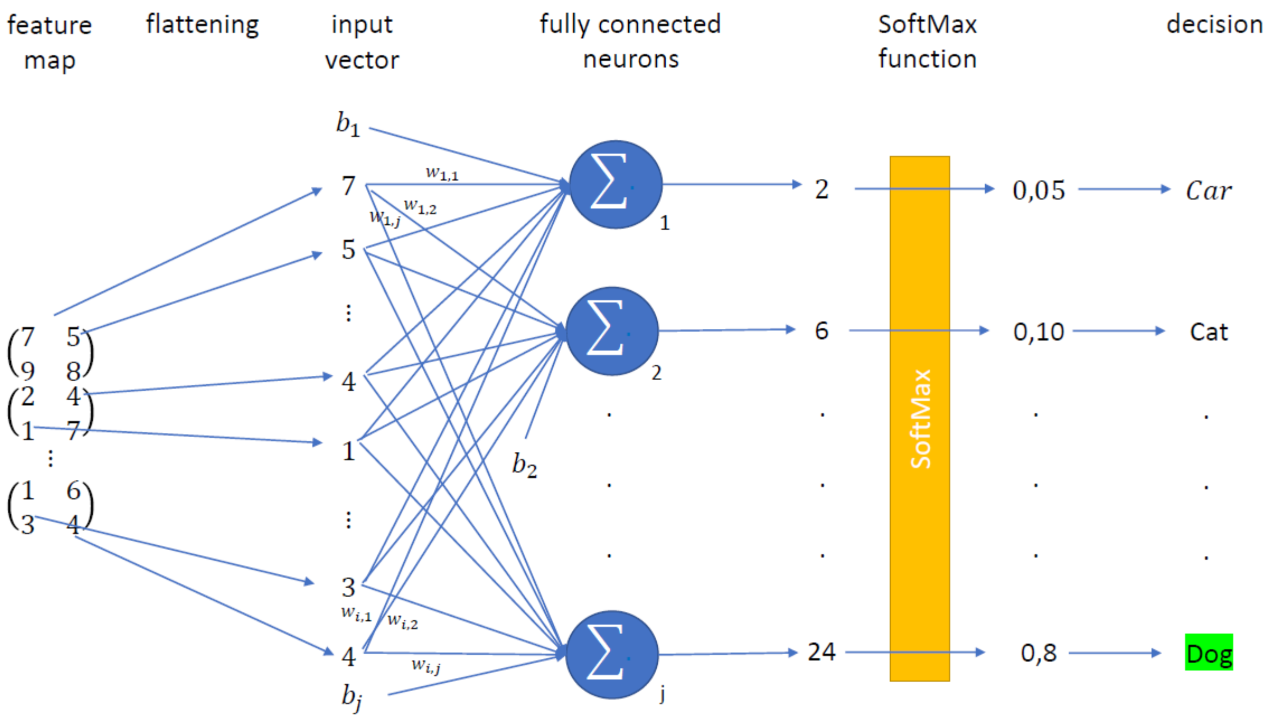
pooling mask



output

$$\Rightarrow \quad \begin{pmatrix} 7 & 5 \\ 9 & 8 \end{pmatrix}$$

Convolutional Neural Network Design and function – fully connected layer

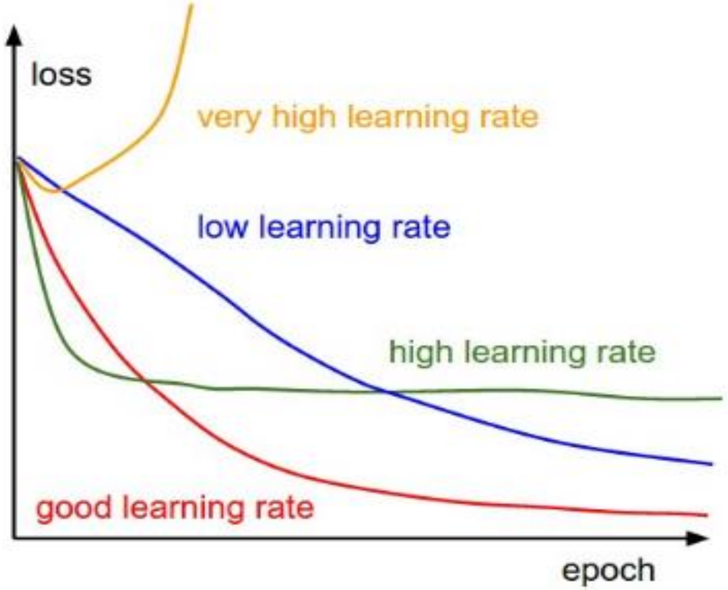


Convolutional Neural Network
Learning process

The learning process can be divided into three steps:

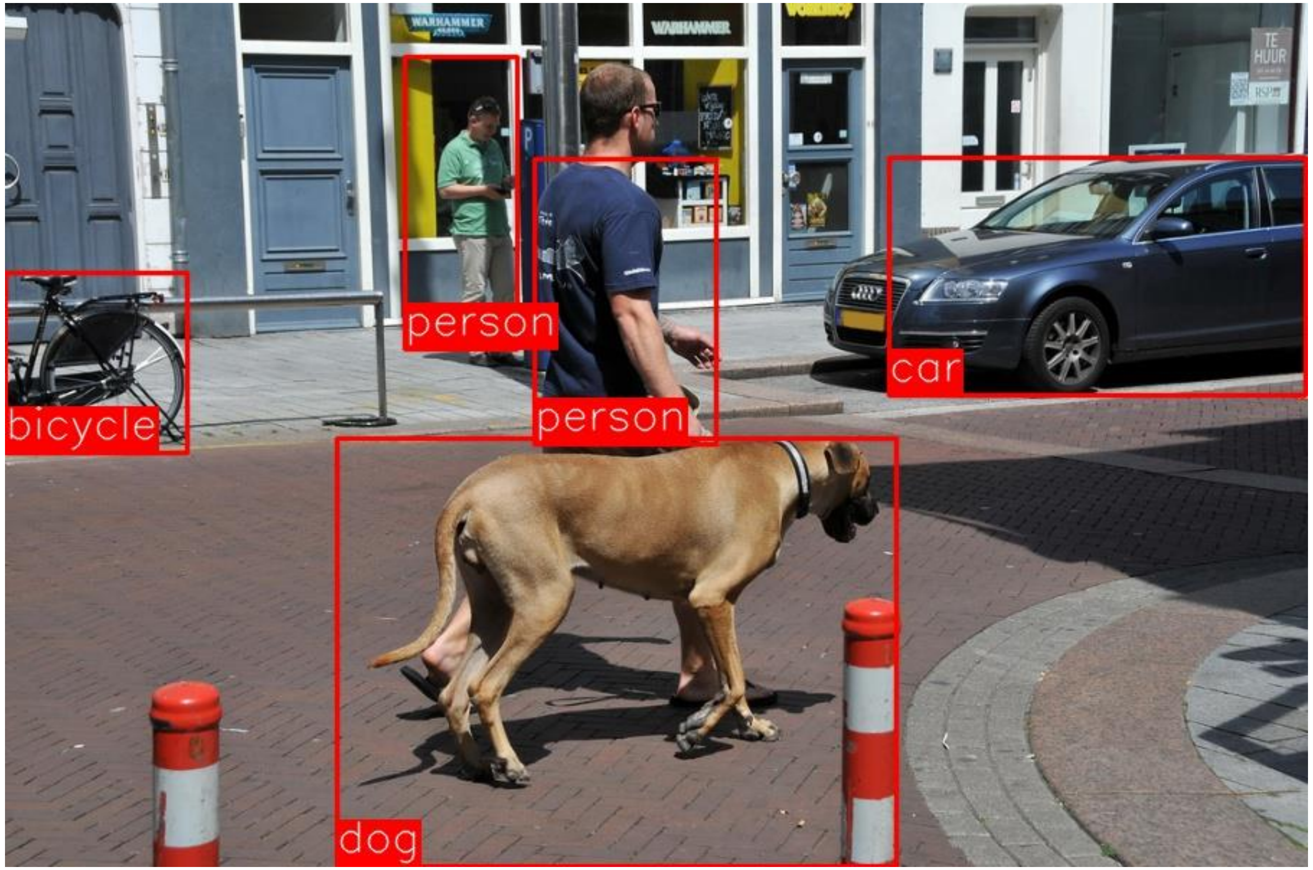
1. Perform classification
2. Calculate output error
3. Transfer output errors

$$w_{opt} = w - \eta \cdot \frac{\delta E(w)}{\delta w} \qquad \textbf{(4)}$$



Dirk von Grüningen; Martin Weilmenmann. Domänenübergreifende Sentiment-Analyse mit Deep Convolutional Neural Networks. [PDF] 2018 page 16

Convolutional Neural Network
Fast region-based CNN



https://mohitjain.me/2018/03/16/understanding-fast-r-cnn

Example

- Categorising of two different handwritten letters

Convolutional Neural Network
Categorising of two different handwritten letters

category 'N'

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

Category 'T'

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

Possible Filter Kernals
category 'N'

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

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

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

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

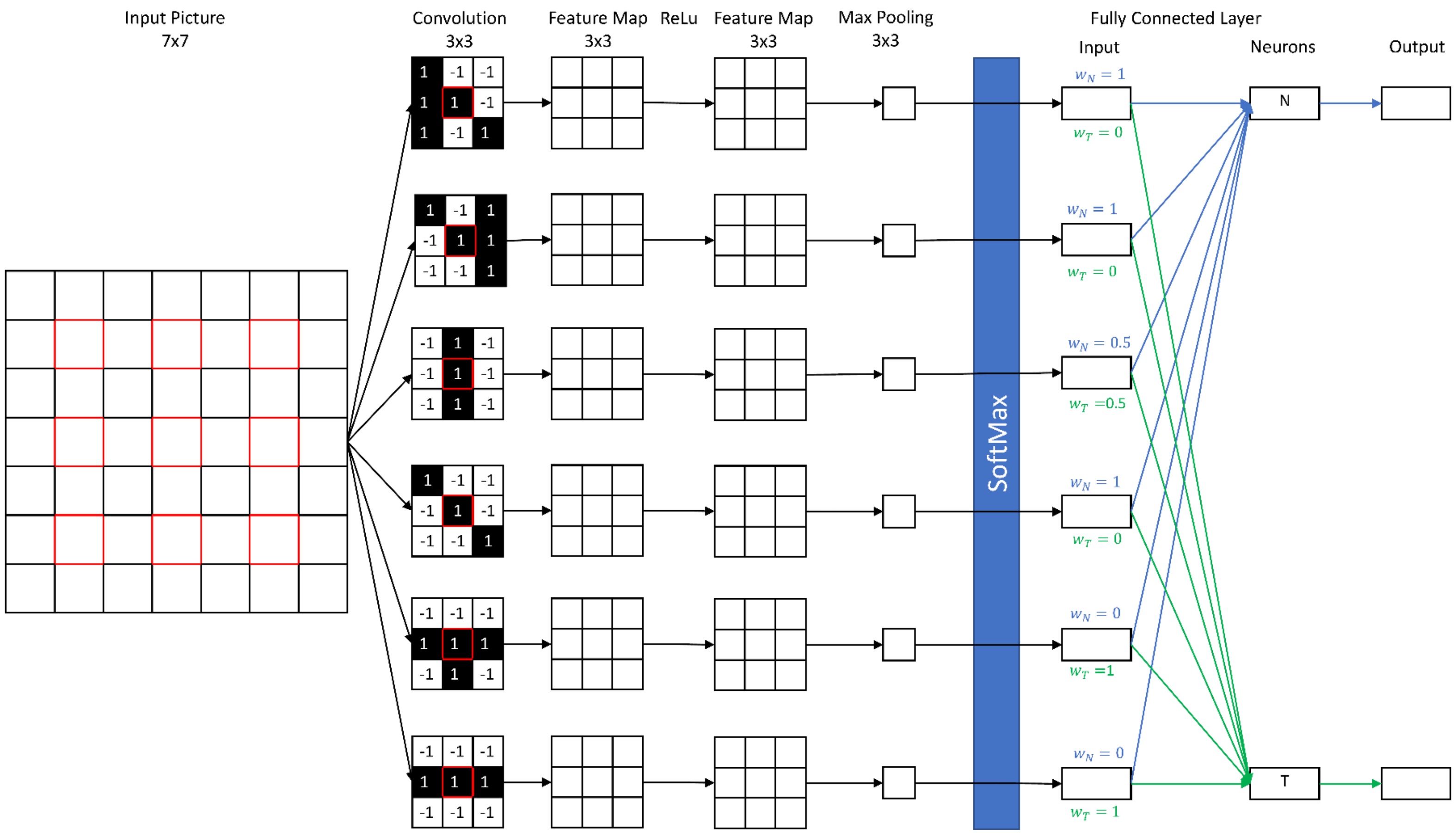
Possible Filter Kernals
category 'T'

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

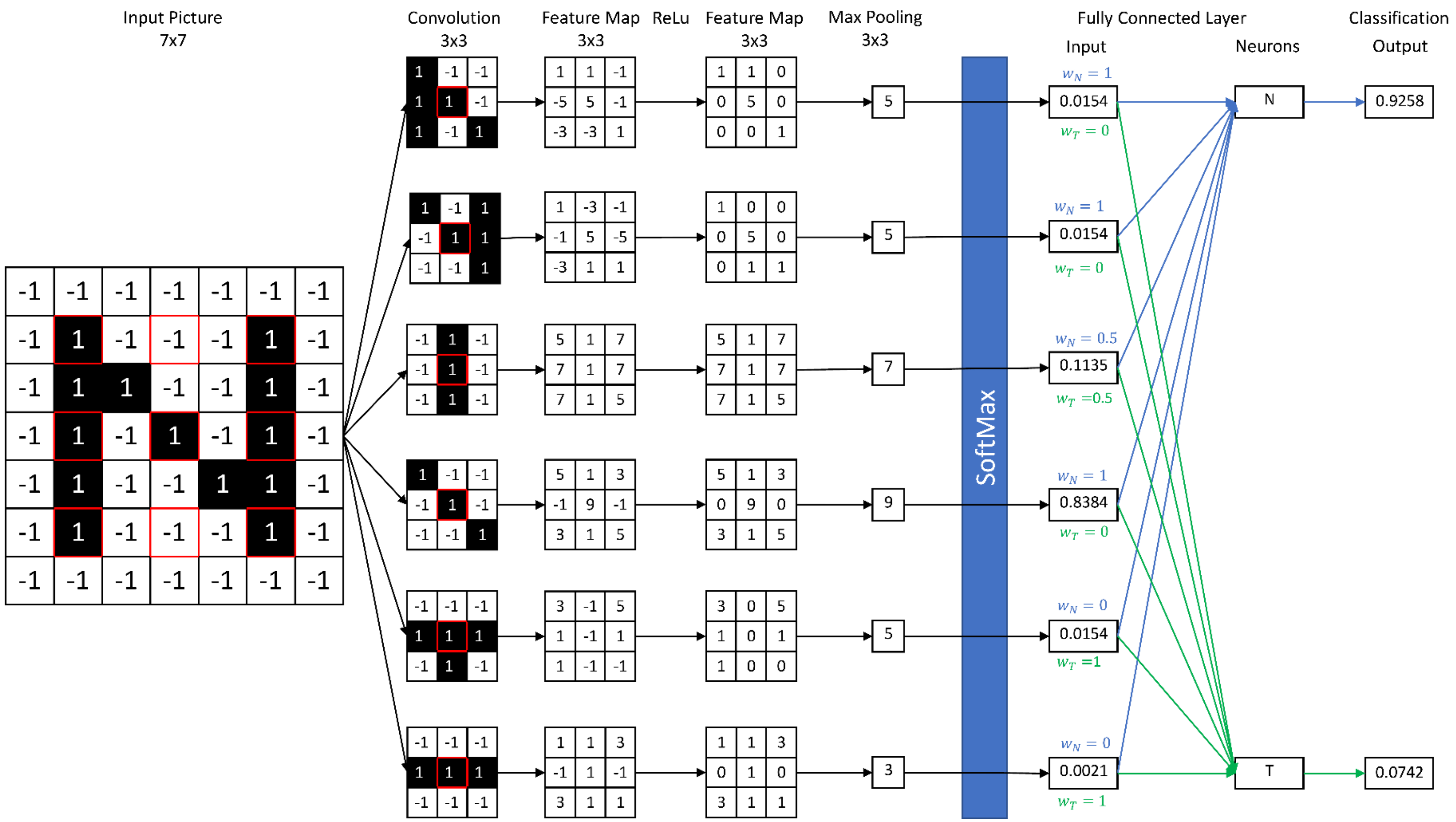
-1	-1	-1
1	1	1
-1	1	-1

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

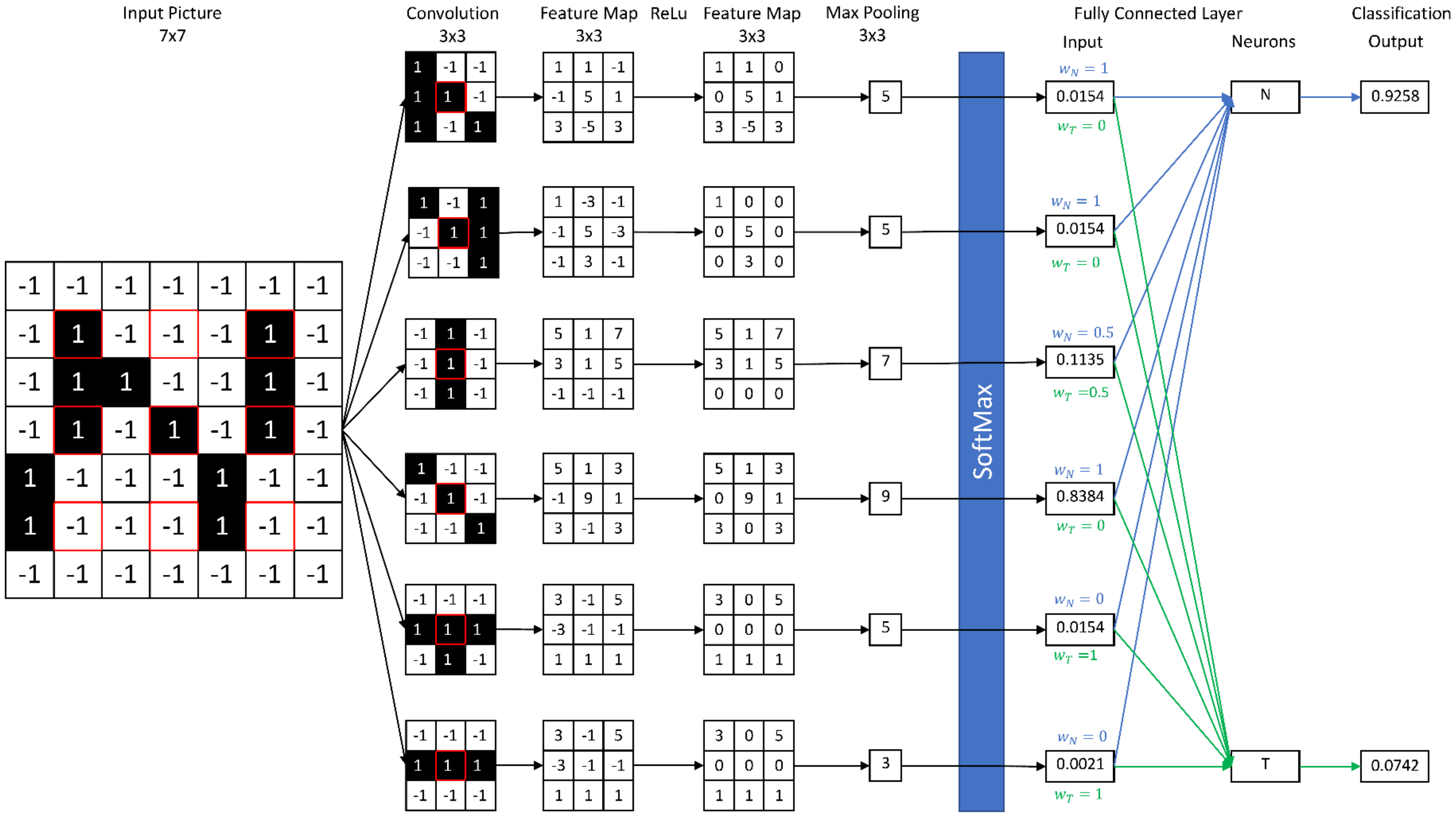
Convolutional Neural Network
Categorising of two different handwritten letters



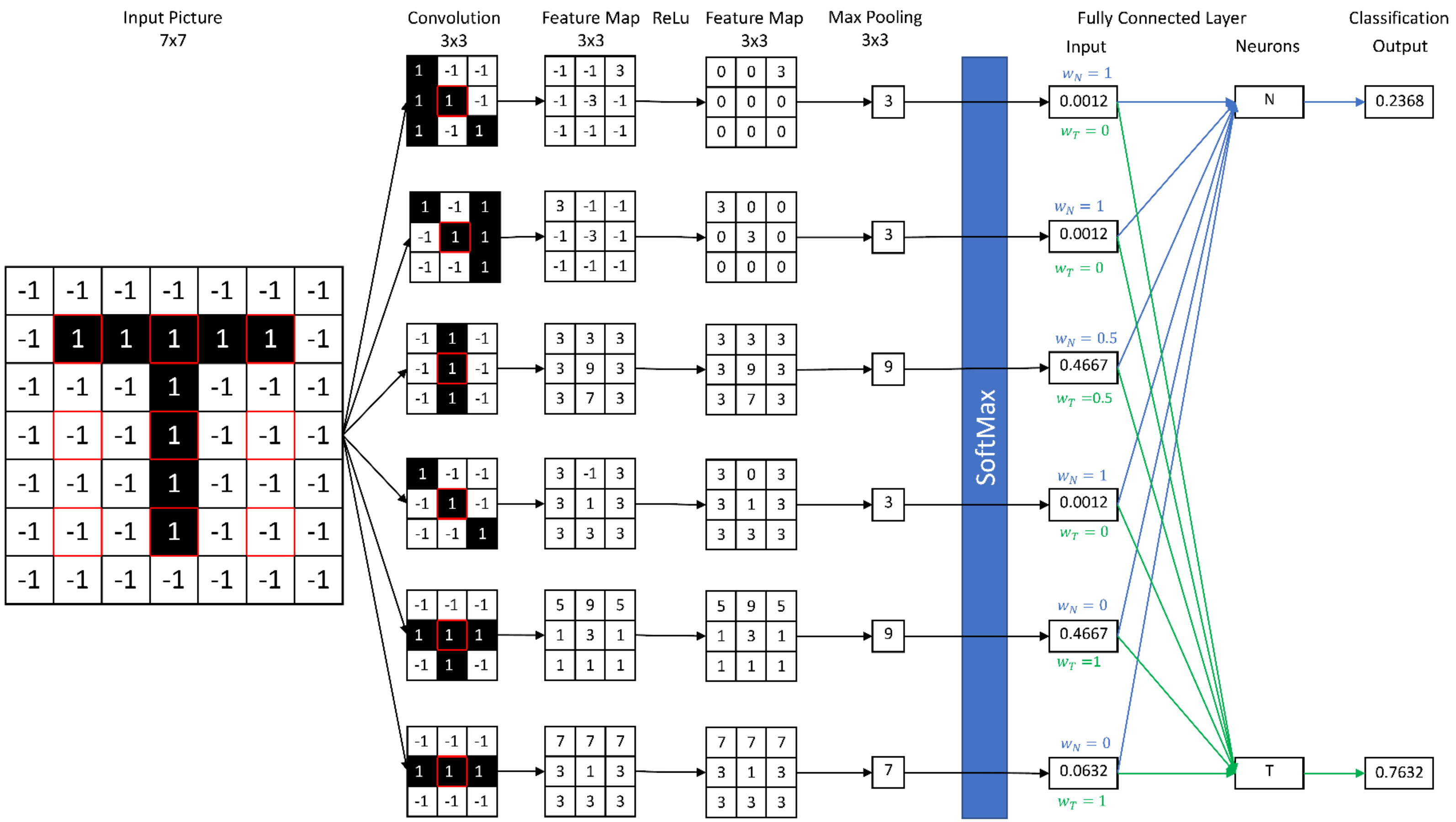
Convolutional Neural Network
Categorising of two different handwritten letters



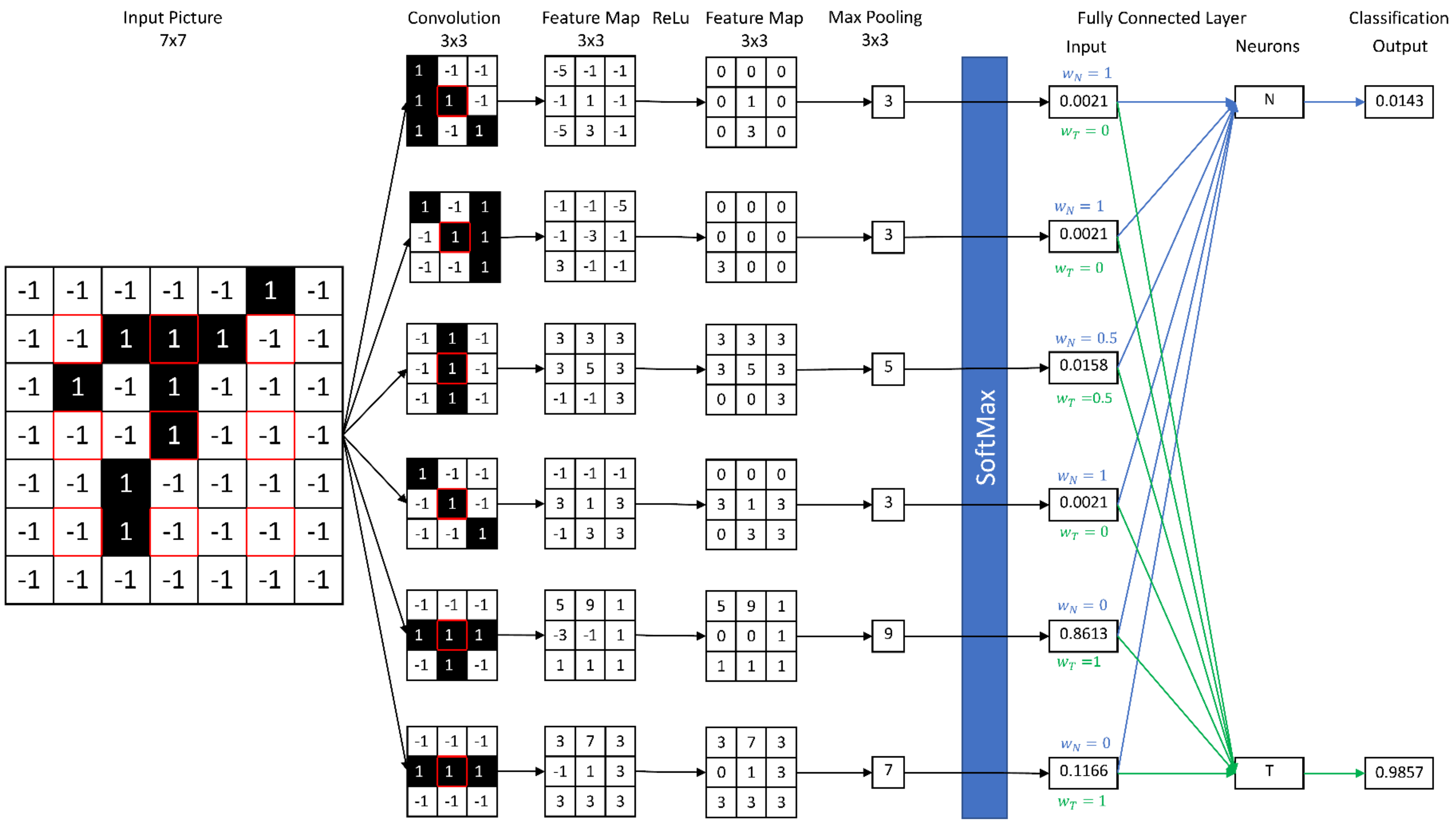
Convolutional Neural Network
Categorising of two different handwritten letters



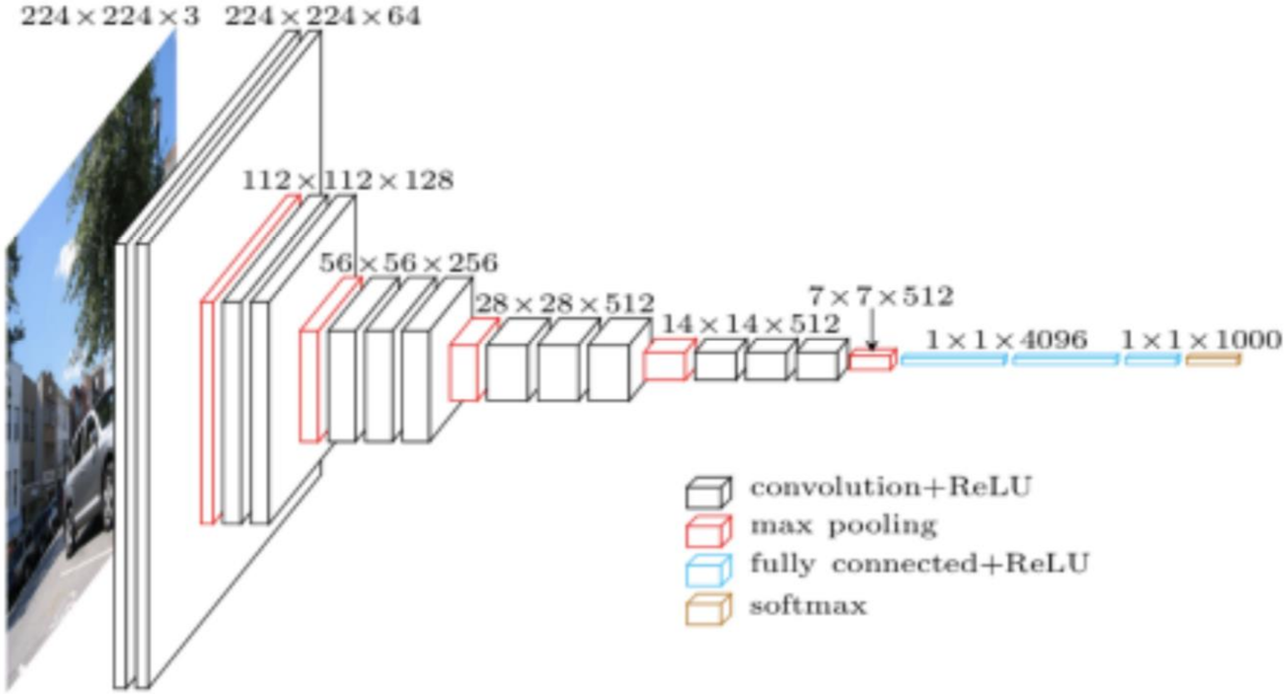
Convolutional Neural Network
Categorising of two different handwritten letters



Convolutional Neural Network
Categorising of two different handwritten letters

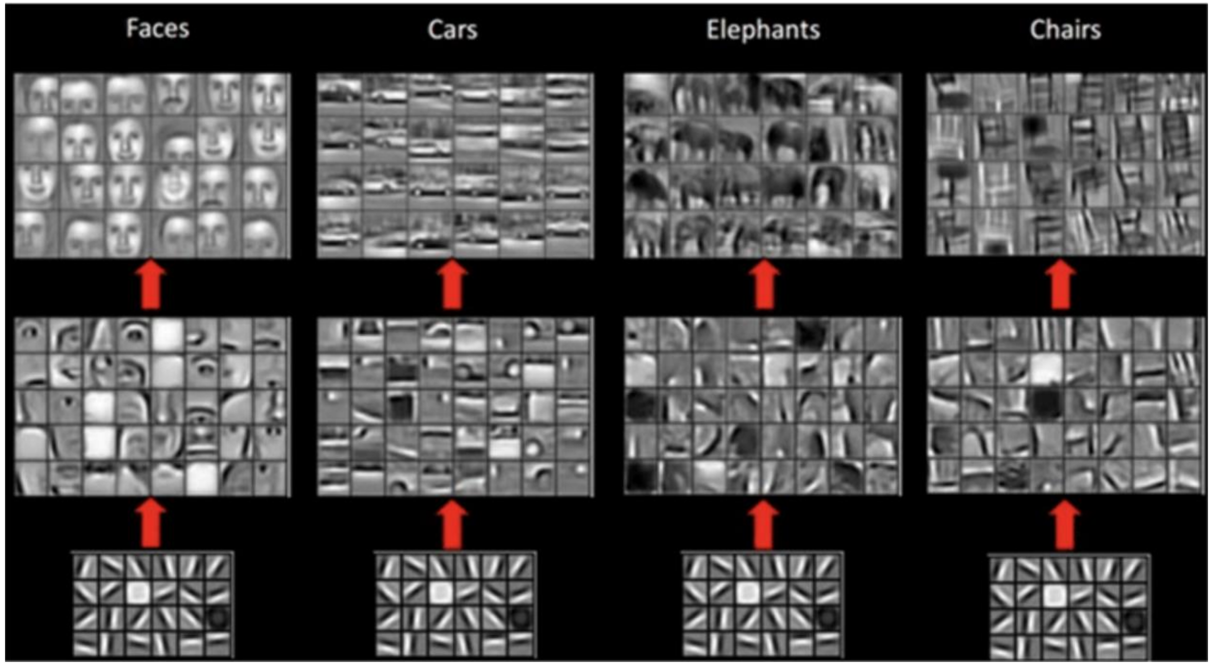


The complete Architecture of a CNN :



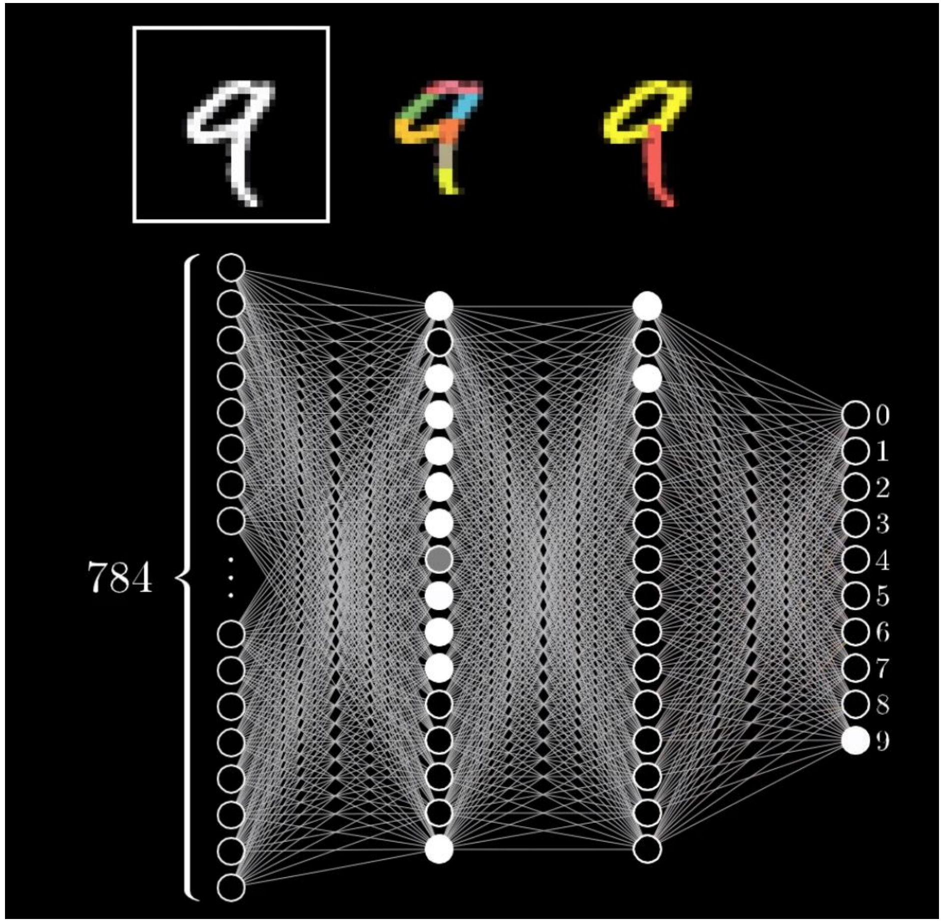
The architecture of a standard CNN.

What do CNN layers learn?



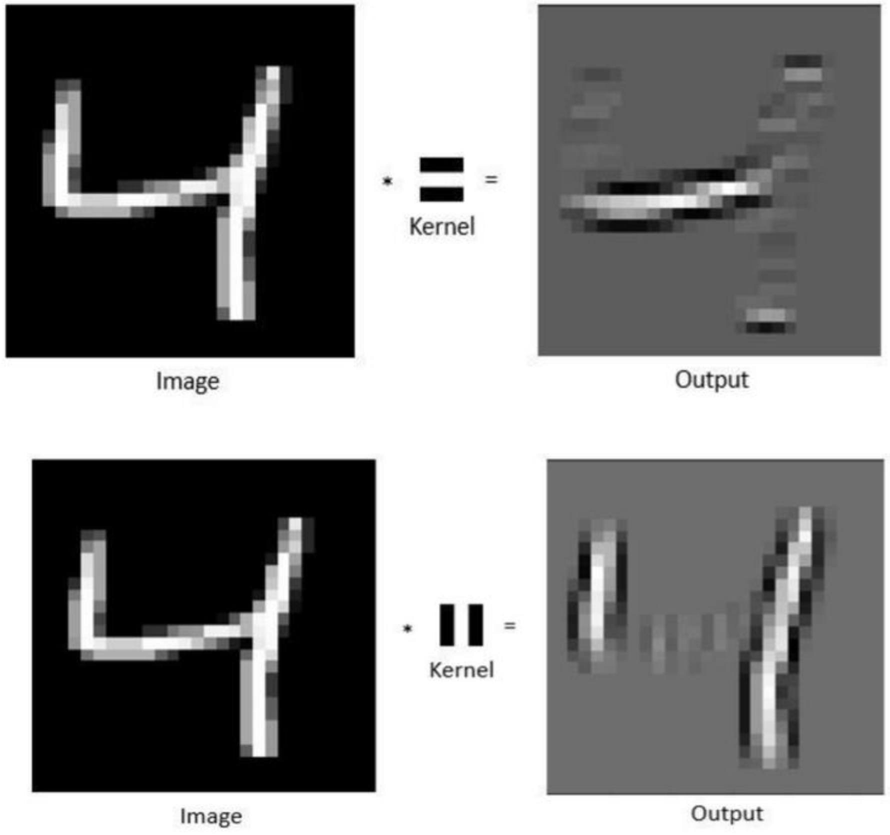
- Each CNN layer learns filters of increasing complexity.
- The first layers learn basic feature detection filters: edges, corners, etc.,
- The middle layers learn filters that detect parts of objects. For faces, they might learn to respond to eyes, noses, etc.,
- The last layers have higher representations: they learn to recognize full objects, in different shapes and positions.

Working Principle with Example:

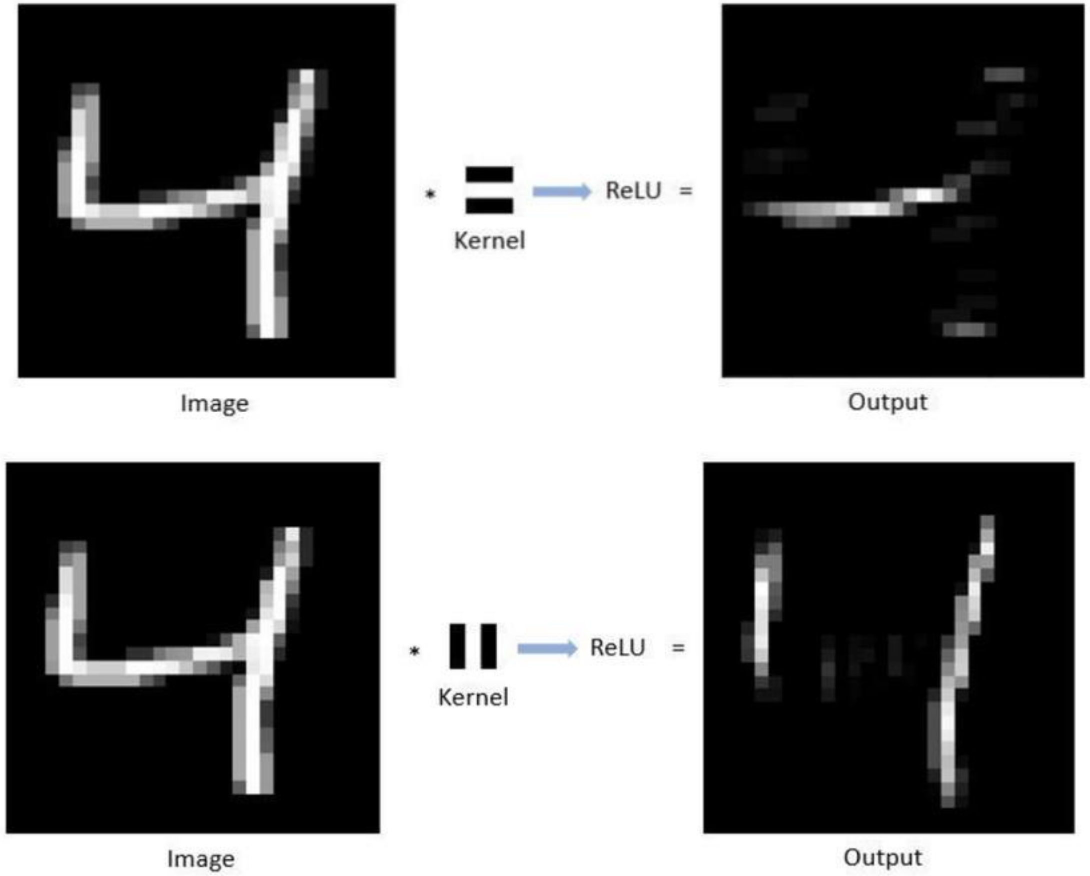


- Handwritten number
- Motto – an image is detected as edges…
- from edges to patterns…
- patterns to predictions…
- predictions to output

Convolution :



Activation :



- Die Vorlesungs- und Übungsunterlagen sind ausschließlich für den Gebrauch in meinen Lehrveranstaltungen bestimmt! Es ist ausdrücklich nur die private Verwendung der Unterlagen für die Kursteilnehmer gestattet.
- Die Weitergabe der Unterlagen an Dritte, ihre Vervielfältigung oder Verwendung auch von Auszügen davon in anderen elektronischen oder gedruckten Publikationen ist nicht gestattet.

- All materials, slides, lecture notes, exercises are for personal use by course participants only.
- Copying and distribution of the material or of parts of it, by any means is strictly forbidden.

References :

- [1] Sharma, N., Jain, V., & Mishra, A. (2018). An analysis of convolutional neural networks for image classification. *Procedia computer science*, *132*, 377-384.
- [2] Khan, A., Sohail, A., Zahoora, U., & Qureshi, A. S. (2019). A survey of the recent architectures of deep convolutional neural networks. *arXiv preprint arXiv:1901.06032*.
- [3] https://www.jeremyjordan.me/convnet-architectures/ by Jeremy Jordan
- [4] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
- [5] https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac
- [6] https://pathmind.com/wiki/convolutional-network by Chris Nicholson , the CEO of Path mind
- [7] Nielsen, M. A. (2015). *Neural networks and deep learning* (Vol. 25). San Francisco, CA, USA: Determination press.