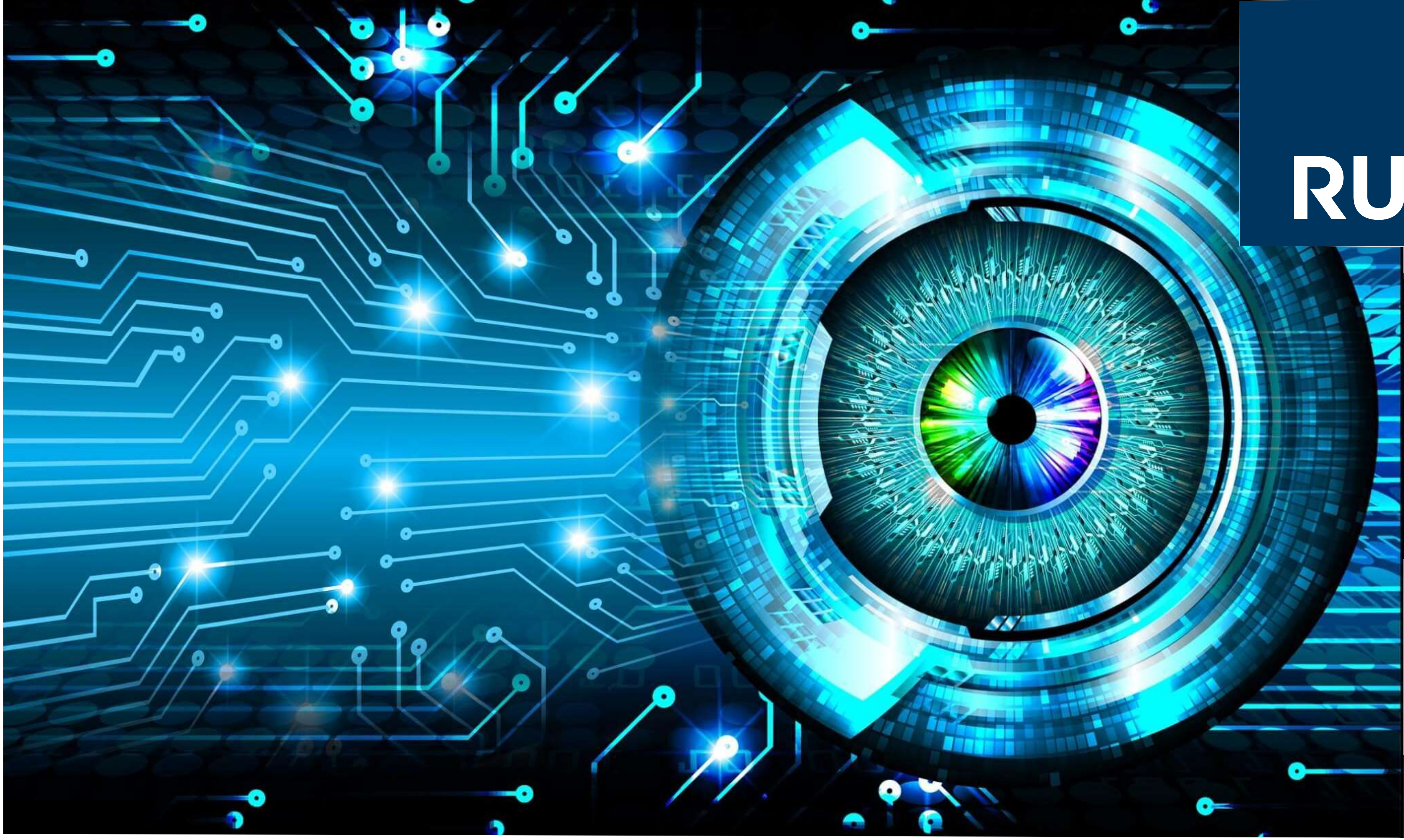


https://iaarbook.github.io/img/computer-vision.jpg

RUHR-UNIVERSITÄT BOCHUM

BASICS ON (CONVOLUTIONAL) NEURAL NETWORKS

How to teach a computer numbers

Computational Physics II - 25/07/2022 - Jurek Völp, jurek.voelp@ruhr-uni-bochum.de

Outline

- Basics of neural networks

  How a machine learns

- Convolutional Neural Network

  How a machine learns to read numbers

- Summary

Basics of neural networks

How a machine learns

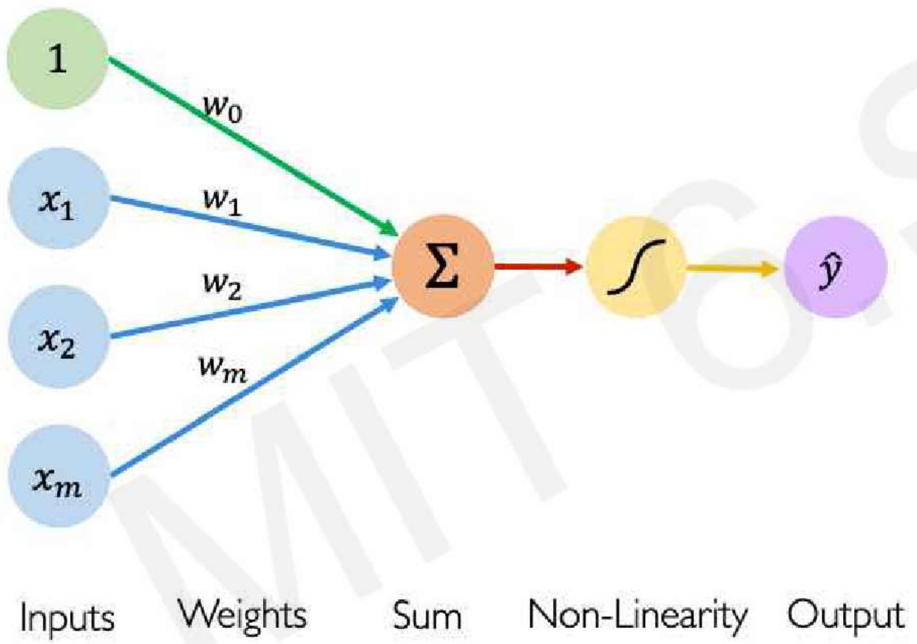
Neuron

- Basic part in a neural network

- Mathematical representation:

  $\hat{y} = g\left(\sum_i x_i w_i\right)$

  - $g(z) \mathrel{\widehat{=}}$ *Activation Function*
    - ➢ Sigmoid
    - ➢ Hyperbolic Tangent
    - ➢ Relu
  - $x_i \mathrel{\widehat{=}}$ *Inputs*
  - $w_i \mathrel{\widehat{=}}$ *Weights*
    - ➢ How important is a certain input?



http://introtodeeplearning.com, MIT 6.S191

Neuron – Implementation

```
import numpy as np
class Neuron:
    def __init__(self, numberOfInputs, actFunc = "sigmoid"):
        # generate random weigts
        self.weights = 2 * np.random.random(numberOfInputs) - 1
        self.bias = 2 * np.random.random() - 1
        # select the activation function
        if type(actFunc) == str:
            if actFunc == "sigmoid":
                self.activation_func = lambda z: 1 / (1 + np.exp(-z))
                self.gradActivation_func = lambda z: self.activation_func(z) * (1 - self.activation_func(z))
            elif actFunc == "hyperbolicTagent":
                self.activation_func = lambda z: (np.exp(z) - np.exp(-z)) / (np.exp(z) + np.exp(-z))
                self.gradActivation_func = lambda z: 1 - self.activation_func(z)**2
            elif actFunc == "relu":
                self.activation_func = lambda z: np.maximum(0, z)
                self.gradActivation_func = lambda z: np.heaviside(z,0)
        elif callable(actFunc):
            self.activation_func = actFunc
        else:
            raise Exception("No valid activation function given.")

    def __call__(self, input):
        if np.size(input, axis=1) == self.weights.size:
            return self.activation_func(self.bias + np.dot(self.weights, input.T))
        else:
            raise Exception("Input is not in the right size.")
```

Neuron – Implementation

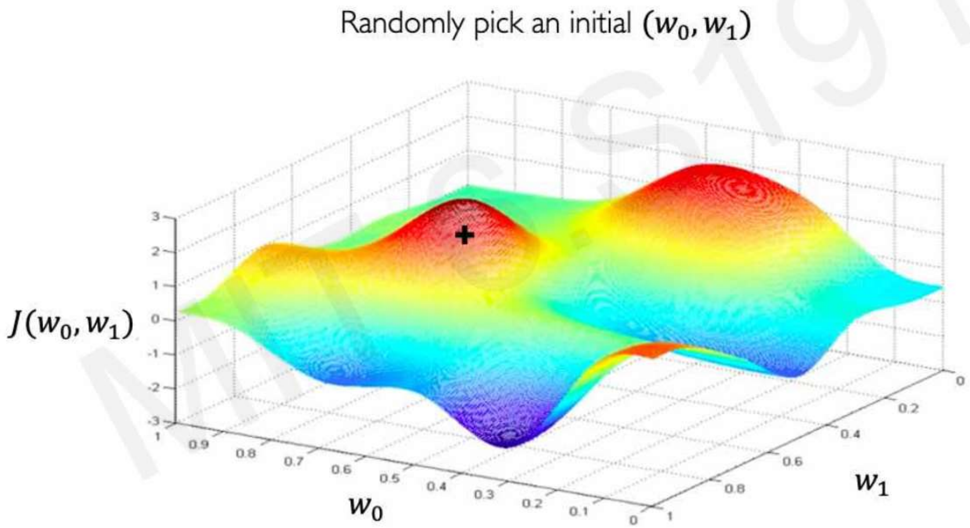
```python
import numpy as np
class Neuron:
    def __init__(self, numberOfInputs, actFunc = "sigmoid"):
        # generate random weigts
        self.weights = 2 * np.random.random(numberOfInputs) - 1
        self.bias = 2 * np.random.random() - 1
        # select the activation function
        if type(actFunc) == str:
            if actFunc == "sigmoid":
                self.activation_func = lambda z: 1 / (1 + np.exp(-z))
                self.gradActivation_func = lambda z: self.activation_func(z) * (1 - self.activation_func(z))
            elif actFunc == "hyperbolicTagent":
                self.activation_func = lambda z: (np.exp(z) - np.exp(-z)) / (np.exp(z) + np.exp(-z))
                self.gradActivation_func = lambda z: 1 - self.activation_func(z)**2
            elif actFunc == "relu":
                self.activation_func = lambda z: np.maximum(0, z)
                self.gradActivation_func = lambda z: np.heaviside(z,0)
        elif callable(actFunc):
            self.activation_func = actFunc
        else:
            raise Exception("No valid activation function given.")

    def __call__(self, input):
        if np.size(input, axis=1) == self.weights.size:
            return self.activation_func(self.bias + np.dot(self.weights, input.T))
        else:
            raise Exception("Input is not in the right size.")
```


RUHR UNIVERSITÄT BOCHUM RUB

How to learn/train

- The cost/loss function quantifies how incorrect the prediction is!
  - Function of the weights
  - Examples:
    - Binary Cross Entropy Loss – between 0 and 1
    - Mean Squared Error Loss – any real number

- **Aim of the training: Optimize/Reduce the cost function!**
- Gradient descent:
  - ➢ random initial values

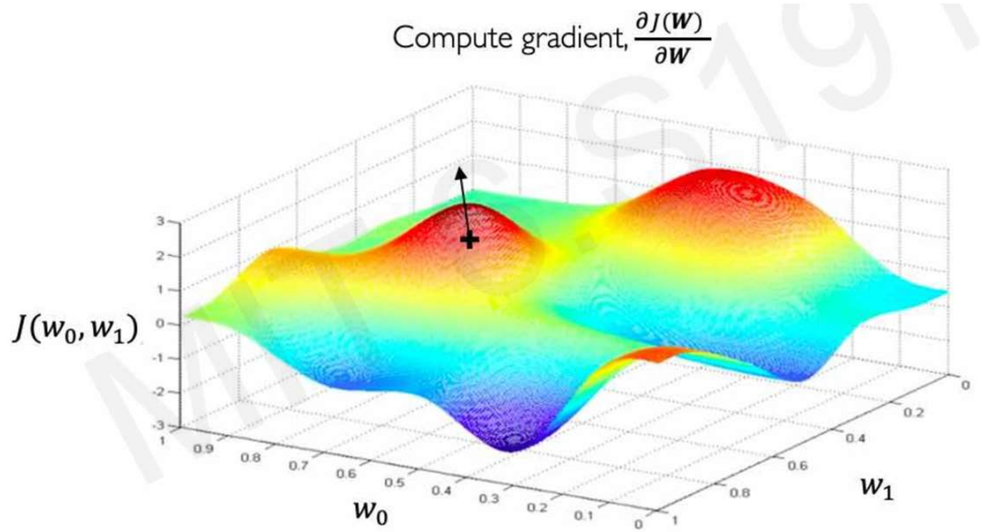


http://introtodeeplearning.com, MIT 6.S191

How to learn/train

- The cost/loss function quantifies how incorrect the prediction is!
  - Function of the weights
  - Examples:
    - Binary Cross Entropy Loss – between 0 and 1
    - Mean Squared Error Loss – any real number

- **Aim of the training: Optimize/Reduce the cost function!**
- Gradient descent:
  - random initial values
  - ➢ compute gradient

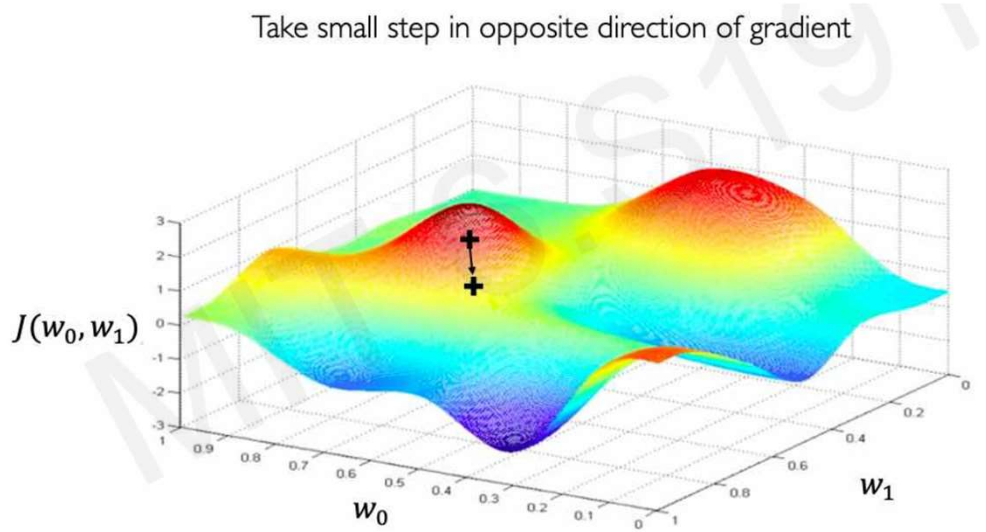


http://introtodeeplearning.com, MIT 6.S191

How to learn/train

- The cost/loss function quantifies how incorrect the prediction is!
  - Function of the weights
  - Examples:
    - Binary Cross Entropy Loss – between 0 and 1
    - Mean Squared Error Loss – any real number

- **Aim of the training: Optimize/Reduce the cost function!**
- Gradient descent:
  - random initial values
  - compute gradient
  - ➢ move in opposite direction

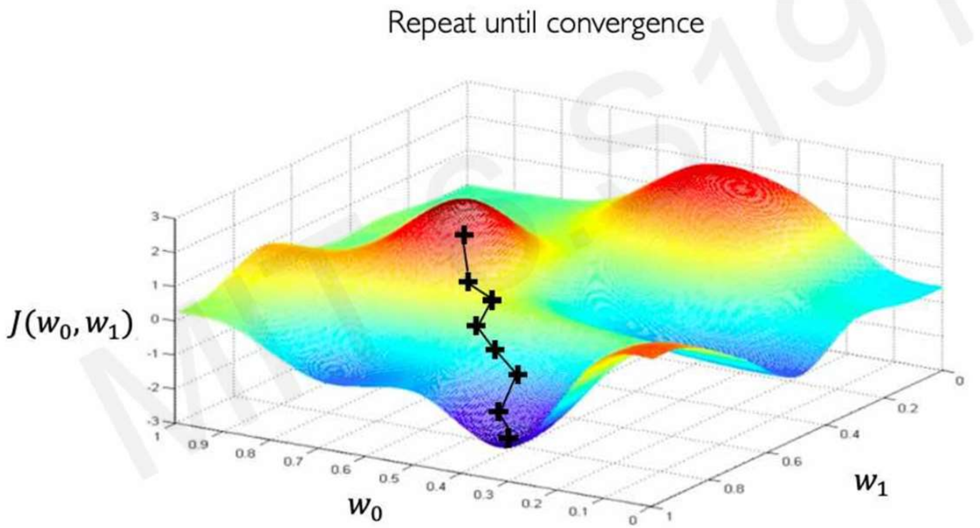


http://introtodeeplearning.com, MIT 6.S191

How to learn/train

- The cost/loss function quantifies how incorrect the prediction is!
  - Function of the weights
  - Examples:
    - Binary Cross Entropy Loss – between 0 and 1
    - Mean Squared Error Loss – any real number

- **Aim of the training: Optimize/Reduce the cost function!**
- Gradient descent:
  - random initial values
  - compute gradient
  - move in opposite direction
  - ➢ repeat until convergence



http://introtodeeplearning.com, MIT 6.S191

How to learn/train - Implementation

- mean squared error loss function
  - hand calculated!
  - special case: one neuron neural network

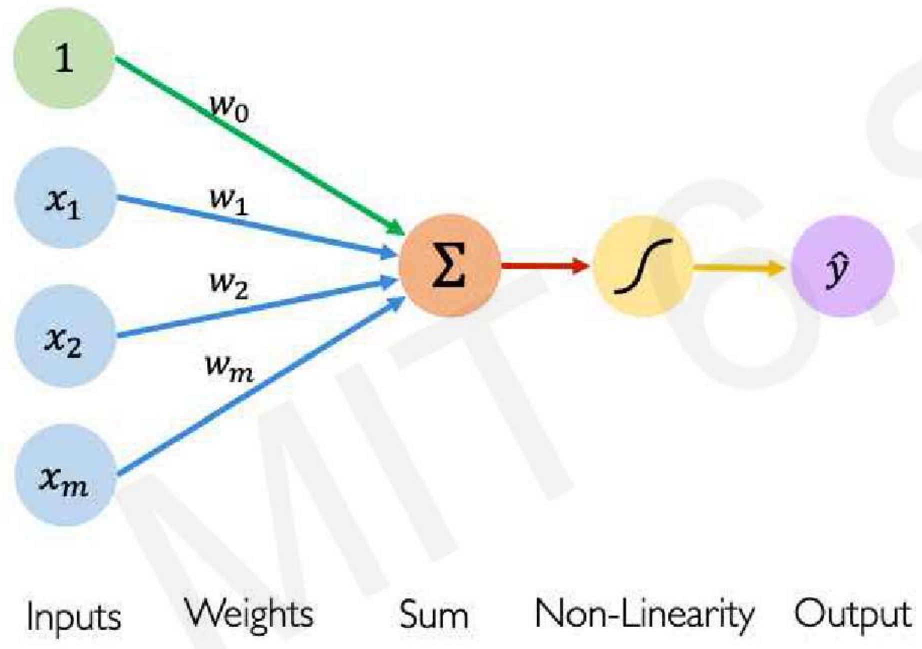
$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\Big(\underbrace{y^{(i)}}_{\text{Actual}} - \underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big)^2$$

http://introtodeeplearning.com, MIT 6.S191

```python
import numpy as np
def singleNeuronGradientMeanSquaredErrorLoss(numberOfInputs, input, real_output, predicted_output, neuron):
    """
    Hand calculated gradient of the mean squared error loss
    """
    n = real_output.size
    input = np.array(input)
    grad = np.zeros(numberOfInputs + 1)
    # weights gradient
    for j in range(numberOfInputs):
        sum = 0
        for i in range(n):
            sum += (real_output[i] - predicted_output[i]) * neuron.gradActivation_func(np.dot(neuron.weights, input[i,:])) * input[i,j]
        grad[j] = -2 / n * sum
    # bias gradient
    for i in range(n):
        sum += (real_output[i] - predicted_output[i]) * neuron.gradActivation_func(np.dot(neuron.weights, input[i,:]))
        grad[-1] = -2 / n * sum
    return grad
```

A simple neural network

- One neuron with three Inputs



http://introtodeeplearning.com, MIT 6.S191

- Data:

x_1	x_2	x_3	y_1	
0	0	1	0	Training data
0	1	1	0	
1	0	1	1	
1	1	1	1	
1	0	0	1	Testing data

A simple neural network – Implementation

```
numberOfTrainingIterations = 10000

train_input = np.array([[0, 0, 1],
                        [1, 1, 1],
                        [1, 0, 1],
                        [0, 1, 1]])
train_output = np.array([0, 1, 1, 0])

test_input = np.array([[1, 0, 0]])
test_output = np.array([1])

learing_rate = 1.

neuron = Neuron(3, actFunc="sigmoid")

for index in range(numberOfTrainingIterations):
    pred_output = neuron(train_input)
    grad = singleNeuronGradientMeanSquaredErrorLoss(3, train_input, train_output, pred_output, neuron)
    neuron.weights -= learing_rate * grad[0:-1]
    neuron.bias -= learing_rate * grad[-1]

print("Result: ", neuron(test_input))
print("Weights: ", neuron.weights)
print("Bias: ", neuron.bias)
```

A simple neural network – Implementation

```
numberOfTrainingIterations = 10000

train_input = np.array([[0, 0, 1],
                        [1, 1, 1],
                        [1, 0, 1],
                        [0, 1, 1]])
train_output = np.array([0, 1, 1, 0])

test_input = np.array([[1, 0, 0]])
test_output = np.array([1])

learing_rate = 1.

neuron = Neuron(3, actFunc="sigmoid")

for index in range(numberOfTrainingIterations):
    pred_output = neuron(train_input)
    grad = singleNeuronGradientMeanSquaredErrorLoss(3, train_input, train_output, pred_output, neuron)
    neuron.weights -= learing_rate * grad[0:-1]
    neuron.bias -= learing_rate * grad[-1]

print("Result: ", neuron(test_input))
print("Weights: ", neuron.weights)
print("Bias: ", neuron.bias)
```

```
Result:  [0.98828371]
Weights:  [ 8.88557619 -0.12700066 -2.06464961]
Bias:  -4.450586368204799
```

Compare with real solution: [1]

We can look at weights and bias → supervised learning

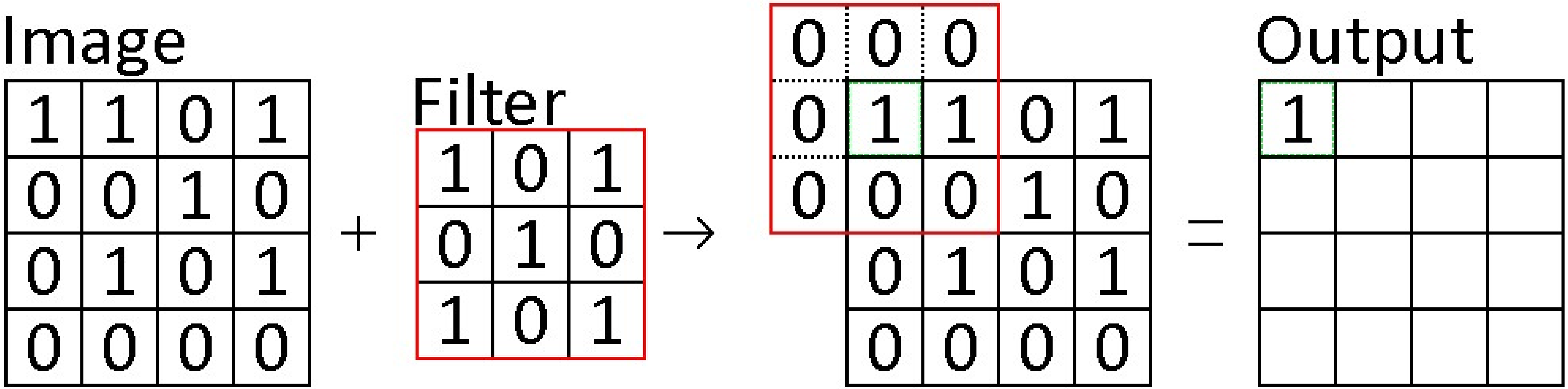
Convolutional Neural Network

How a machine learns to see numbers

Convolution

- Mathematical definition: $(f * g)(x) := \int_{\mathbb{R}^n} f(\tau)g(x-\tau)d\tau$
  - interpretation: *weighting a function* $g(x-\tau)$ *with* $f(\tau)$

- Application in image processing:
  - using 2d filters to detect specific structures
  - example:

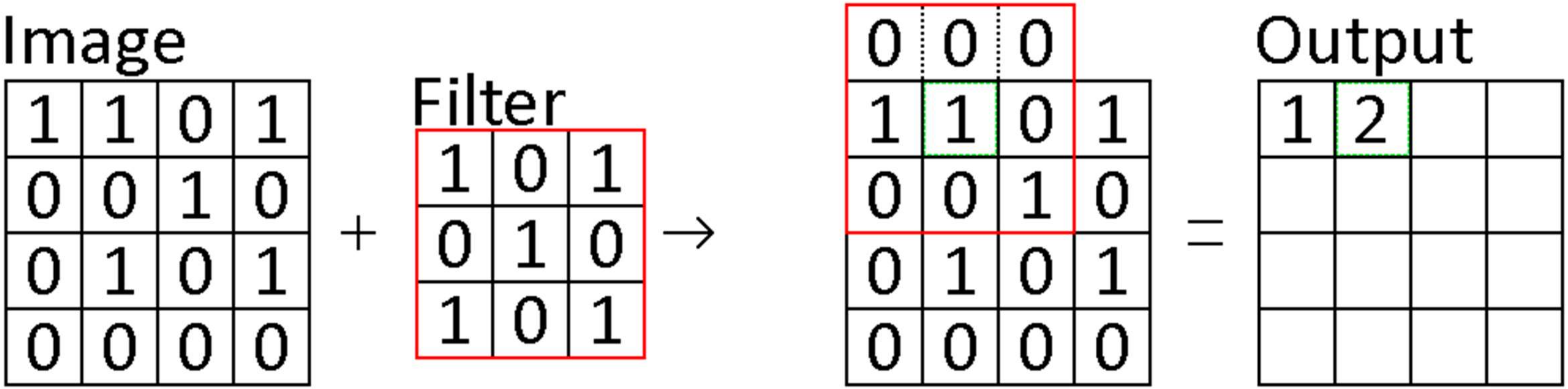


RUHR UNIVERSITÄT BOCHUM

Convolution

- Mathematical definition: $(f * g)(x) := \int_{\mathbb{R}^n} f(\tau) g(x-\tau) d\tau$
  - interpretation: *weighting a function $g(x-\tau)$ with $f(\tau)$*

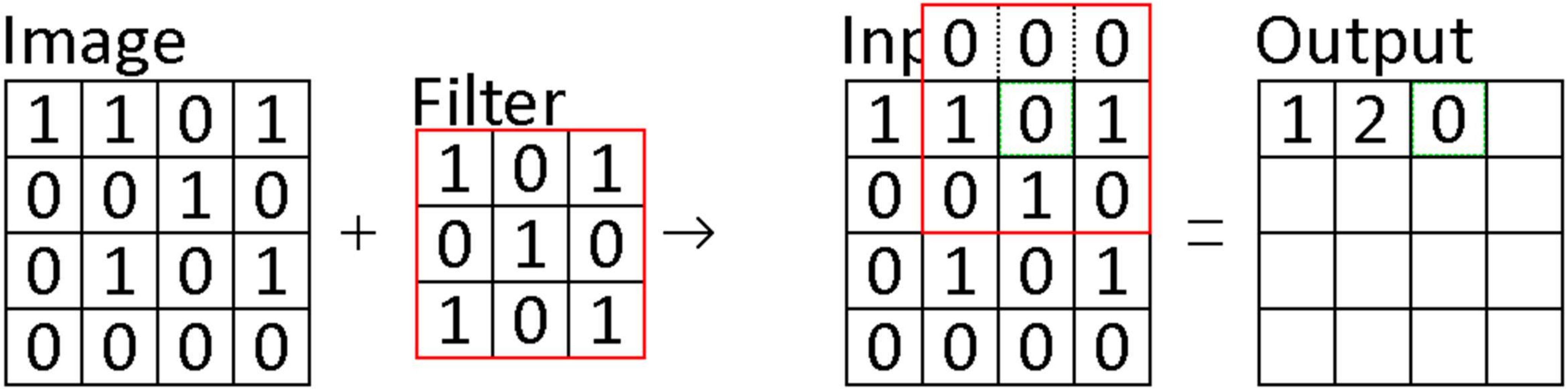
- Application in image processing:
  - using 2d filters to detect specific structures
  - example:



Convolution

- Mathematical definition: $(f * g)(x) := \int_{\mathbb{R}^n} f(\tau)g(x-\tau)d\tau$
  - interpretation: *weighting a function* $g(x-\tau)$ *with* $f(\tau)$

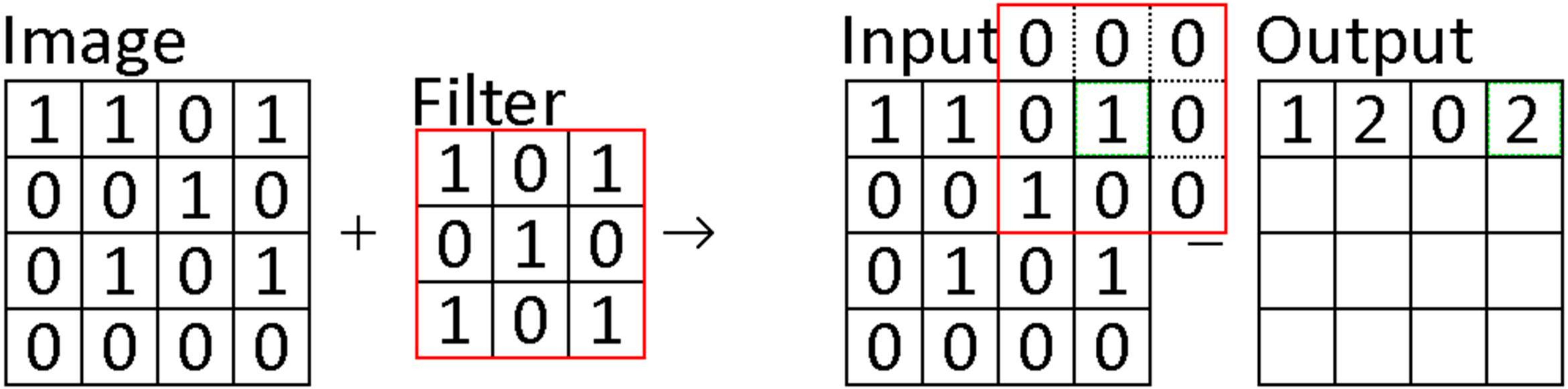
- Application in image processing:
  - using 2d filters to detect specific structures
  - example:



Convolution

- Mathematical definition: $(f * g)(x) := \int_{\mathbb{R}^n} f(\tau)g(x-\tau)d\tau$
  - interpretation: *weighting a function* $g(x-\tau)$ *with* $f(\tau)$

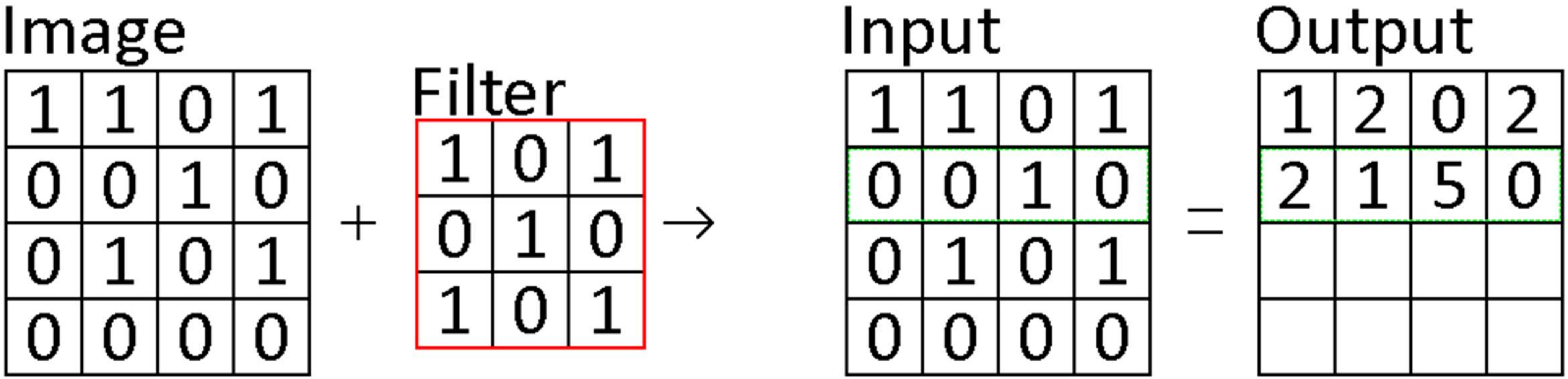
- Application in image processing:
  - using 2d filters to detect specific structures
  - example:



Convolution

- Mathematical definition: $(f * g)(x) \coloneqq \int_{\mathbb{R}^n} f(\tau)g(x-\tau)d\tau$
  - interpretation: *weighting a function* $g(x-\tau)$ *with* $f(\tau)$

- Application in image processing:
  - using 2d filters to detect specific structures
  - example:



Convolution

- Mathematical definition: $(f * g)(x) \coloneqq \int_{\mathbb{R}^n} f(\tau) g(x - \tau) d\tau$
  - interpretation: *weighting a function* $g(x - \tau)$ *with* $f(\tau)$

- Application in image processing:
  - using 2d filters to detect specific structures
  - example:

Image

1	1	0	1
0	0	1	0
0	1	0	1
0	0	0	0

+

Filter

1	0	1
0	1	0
1	0	1

→

Input

1	1	0	1
0	0	1	0
0	1	0	1
0	0	0	0

=

Output

1	2	0	2
2	1	5	0
0	2	0	2

Convolution

- Mathematical definition: $(f * g)(x) \coloneqq \int_{\mathbb{R}^n} f(\tau)g(x-\tau)d\tau$
  - interpretation: *weighting a function* $g(x-\tau)$ *with* $f(\tau)$

- Application in image processing:
  - using 2d filters to detect specific structures
  - example:

$$\underset{\text{Image}}{\begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}} + \underset{\text{Filter}}{\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}} \rightarrow \underset{\text{Input}}{\begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}} = \underset{\text{Output}}{\begin{bmatrix} 1 & 2 & 0 & 2 \\ 2 & 1 & 5 & 0 \\ 0 & 2 & 0 & 2 \\ 1 & 0 & 2 & 0 \end{bmatrix}}$$

Convolutional Neural Network

- Engine: TensorFlow

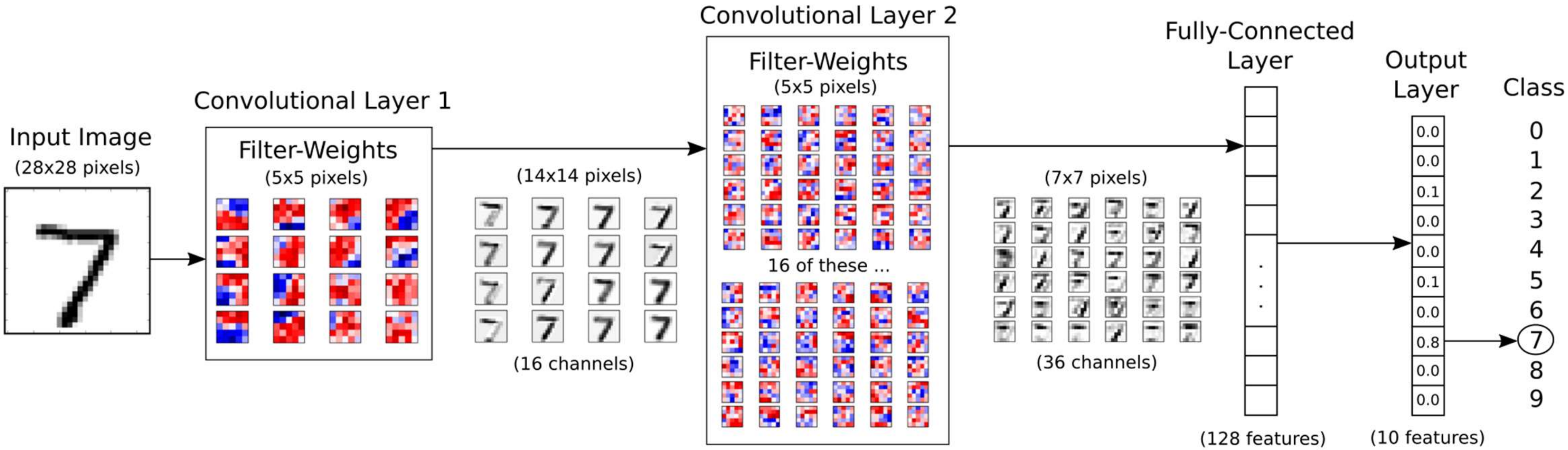
- Deep neural network = a lot of neurons, in many layers
  - 2 convolutional layer with 16 and 36 neurons and one 5x5 filter each
    - ➢ the optimal filters are generated here
  - 2 fully connected layer with 128 and 10 neurons
    - ➢ the classification takes place here

- Data: MNIST database, a lot of handwritten numbers

- Credits: Magnus Erik Hvass Pedersen, [1]

[1] https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/02_Convolutional_Neural_Network.ipynb

Convolutional Neural Network



https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/02_Convolutional_Neural_Network.ipynb

Now let's look at the results!

Summary

- Neural networks…
  - …are not that complicated
  - …can be written from scratch

- Convolutional neural networks are quite good in computer vision. (Better than “normal” neural networks, 91 % vs 99 %)

- Supplementary material can be found here:
  https://github.com/jurvo/cp2-nn-exam
  - Jupyter Notebooks on:
    - the simple neural network with dummy data
    - the simple neural network with titanic data (to play around)
    - the convolutional neural network