Mobile Sensing: Advanced Machine Learning

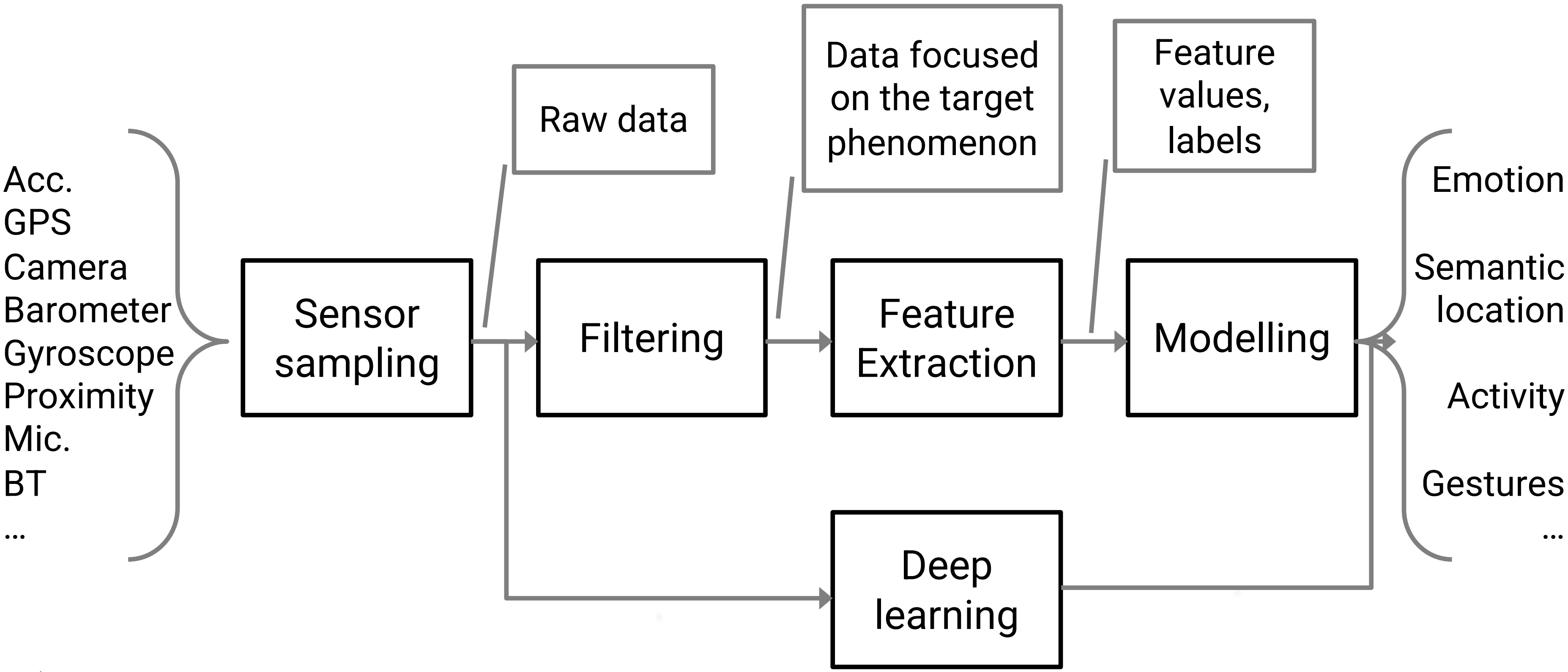
Partly based on: "CS231n: Convolutional Neural Networks for Visual Recognition" by FeiFei Li, Stanford University

Master studies, Winter 2021/2022

Dr Veljko Pejović
Veljko.Pejovic@fri.uni-lj.si

University *of Ljubljana*
Faculty *of Computer and Information Science*

Sensing and Learning Pipeline



University of Ljubljana
Faculty of Computer and Information Science

Automating Feature Engineering

- Feature engineering is challenging
  - An infinite number of potential features (e.g. mean, variance, mean crossing rate, peak frequency, power in a certain frequency domain, entropy, etc.)
  - Difficult to (visually) inspect feature and class values distribution in a dataset in order to get the idea about how informative each feature might end up being
  - Don’t know how the extracted features will interplay with the selected machine learning algorithms
- Deep learning: let the learning algorithm extract features by itself

University of Ljubljana
Faculty of Computer and Information Science

Automating Feature Engineering

- Example: recognize the animal in the photo
- You can extract a number of low-level features manually:
  - Contains blue; contains red; number of colours; colour diffusion; histogram; etc.

Penguin

Dolphin

Parrot

Human

Automating Feature Engineering

- Example: recognize the animal in the photo
- Idea: from low-level features we can train a classifier that infers *something* that will be useful for the final inference, e.g. has water, has ice, is a bird, etc.

Penguin

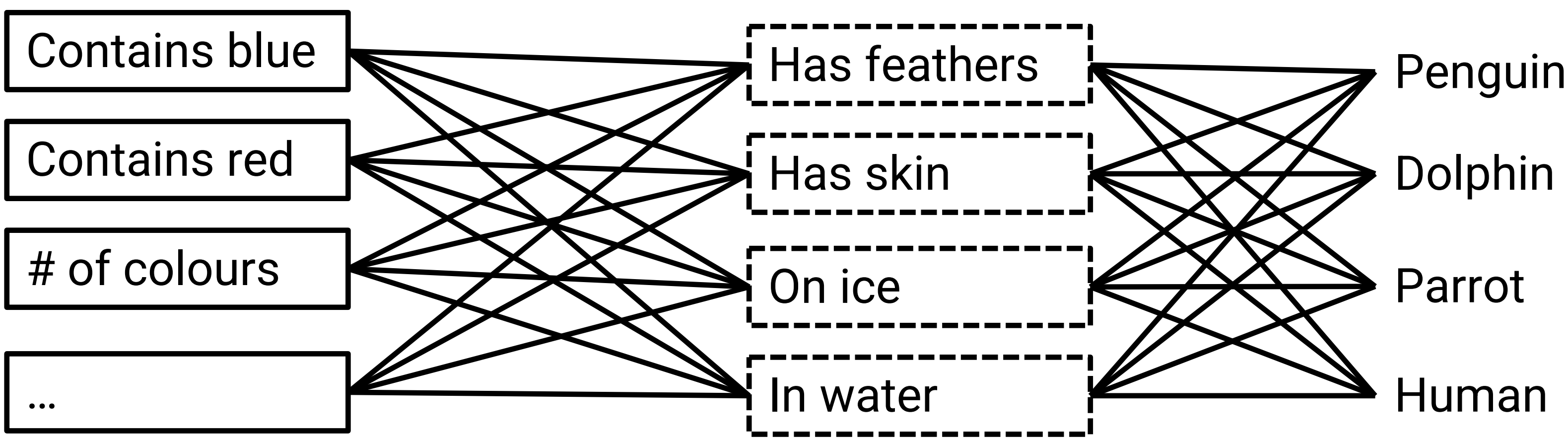
Dolphin

Parrot

Human

Automating Feature Engineering

- Example: recognize the animal in the photo
- The method should automatically identify that *something*

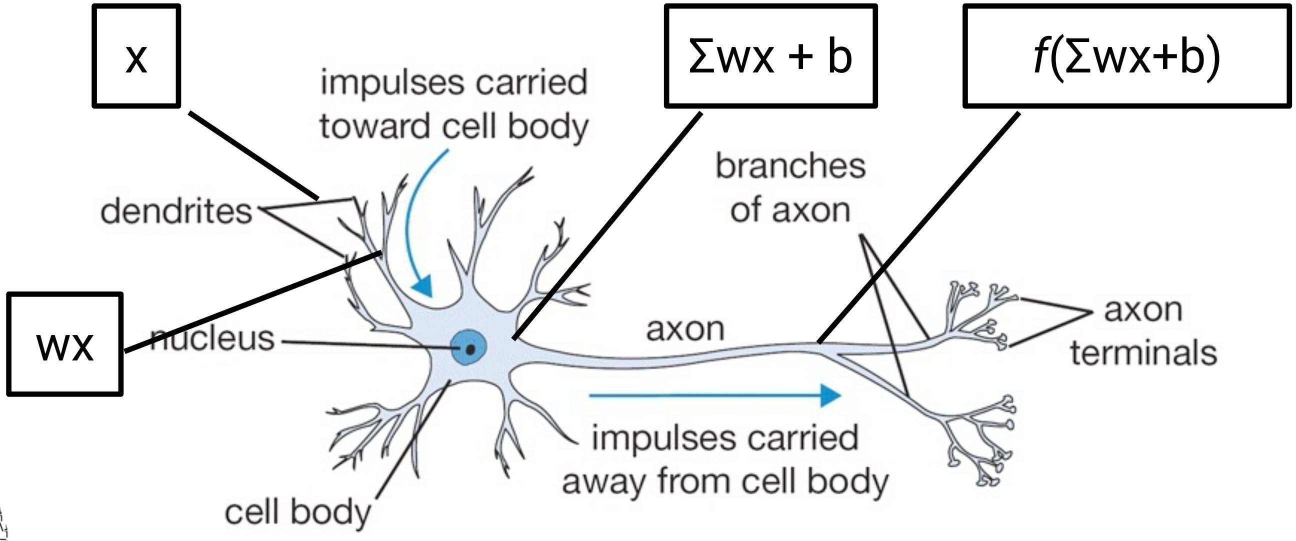


Basics – Perceptron

- Perceptron – a binary classifier:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$

- Inspired by the neuron:



Basics – Perceptron

- Perceptron's activation function is a simple Heaviside step function (0 or 1 output)
- Training is about adjusting weights:

$$w_i(t+1) = w_i(t) + r \cdot (d_j - y_j(t))x_{j,i}$$

- Perceptron is a linear classifier and cannot learn non-linear functions, e.g. XOR
- Stacking perceptrons without a non-linear activation function in a deep network makes no sense – why?

Activation Functions

- Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

- Rectifier (ReLU)

$$\max(0, x)$$

- Hyperbolic tangent

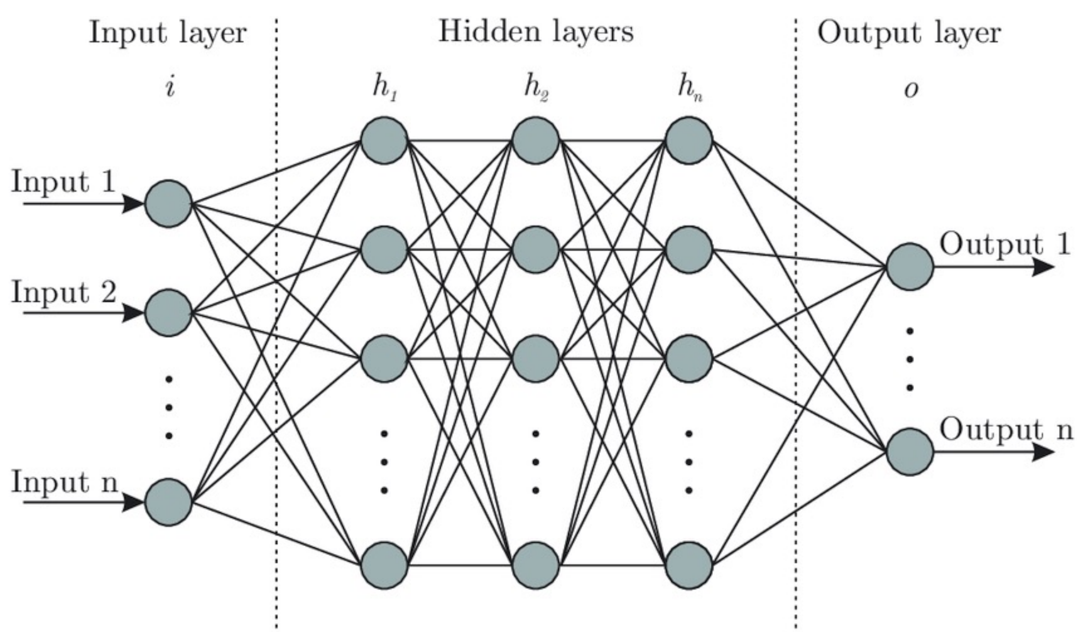
$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- Leaky ReLU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^{x} - 1) & x < 0 \end{cases}$$

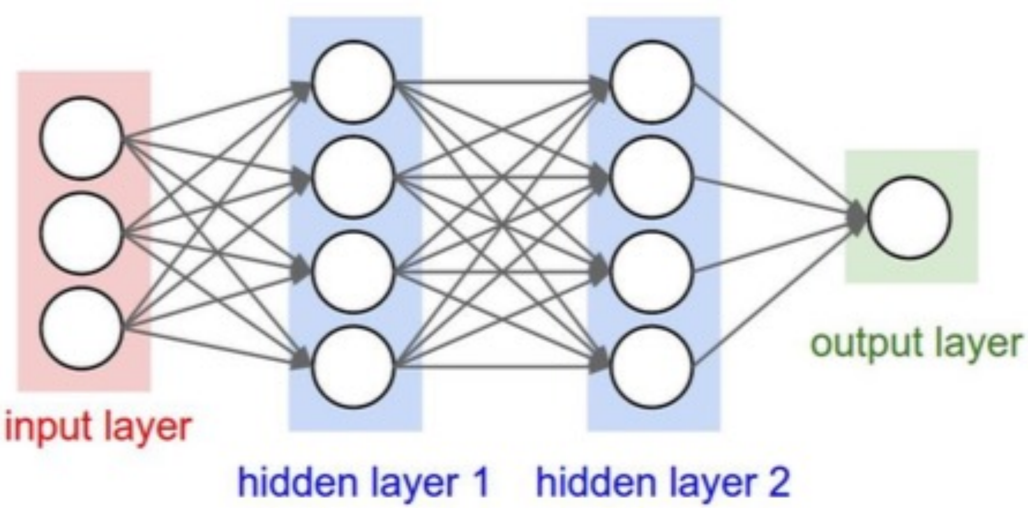
Multilayer Perceptron (aka Neural network)

- We can construct a network where intermediate (hidden) layers represent that *something* – key features that characterize classes
- Each hidden/output layer consists of perceptrons with nonlinear activation functions and biases



Multilayer Perceptron (aka Neural network)

- Example (a 3-layer neural network):




```
f = lambda x: np.maximum(0,x)
x = np.random.rand(3,1)
h1 = f(np.dot(W1,x) + b1)
h2 = f(np.dot(W2,h1) + b2)
s = np.dot(W3, h2) + b3
```

Neural Network Training

- The results of the output layer are scores (s)
- Loss function quantifies how well these scores match the ground truth label
  - Example loss function
    - for each training instance i: $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$
    - regularization (makes sure we use smaller weights): $\sum_k W_k^2$
    - total loss (we want to minimize this): $L = \frac{1}{N} \sum_{i=1..N} L_i + \sum_k W_k^2$

Neural Network Training

- Adjust the perceptrons' weights so that the loss is minimized
- Gradient descent: to find the weights that give the minimum loss, go in the opposite direction of the gradient
  - Calculate the gradient
  - Update the weights
  - Repeat the above steps until the stopping criterion is reached

$$W_{new} = W - r\frac{\partial L}{\partial W}$$

- Calculating the gradient is challenging!

Gradient Calculation

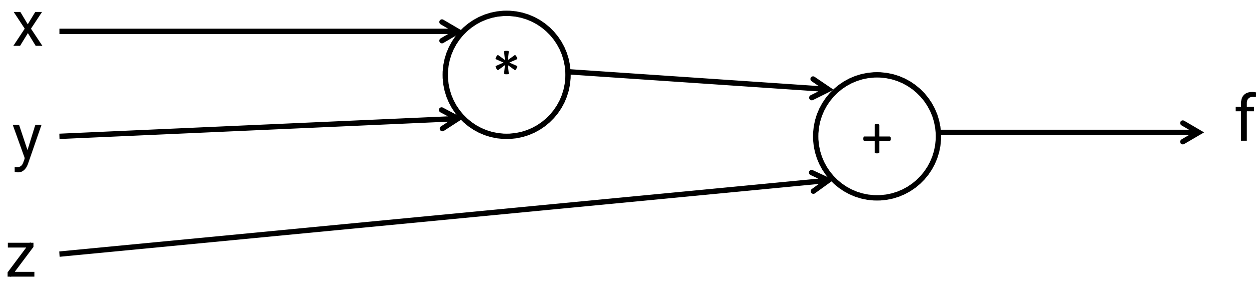
- Idea 1: calculate the gradient numerically
  - Not perfectly accurate
  - You need to do this for each variable (each element of each of the W matrices) and for each output dimension -> prohibitively slow

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- Idea 2: calculate the gradient analytically
  - Gets quite complicated for larger networks
  - Not modular – change one activation/loss function and you need to re-derive the expression

Gradient Calculation

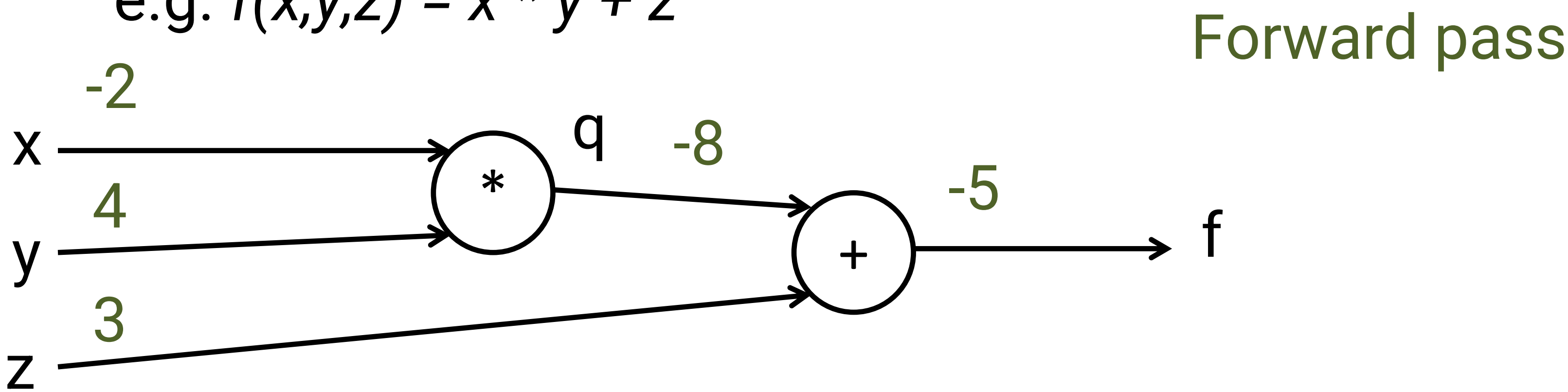
- Idea 3: computational graph & backpropagation
  - Represent calculation as a graph,
    e.g. $f(x,y,z) = x * y + z$



  - Calculate local gradients for each element, and then backpropagate to the previous element

Gradient Calculation

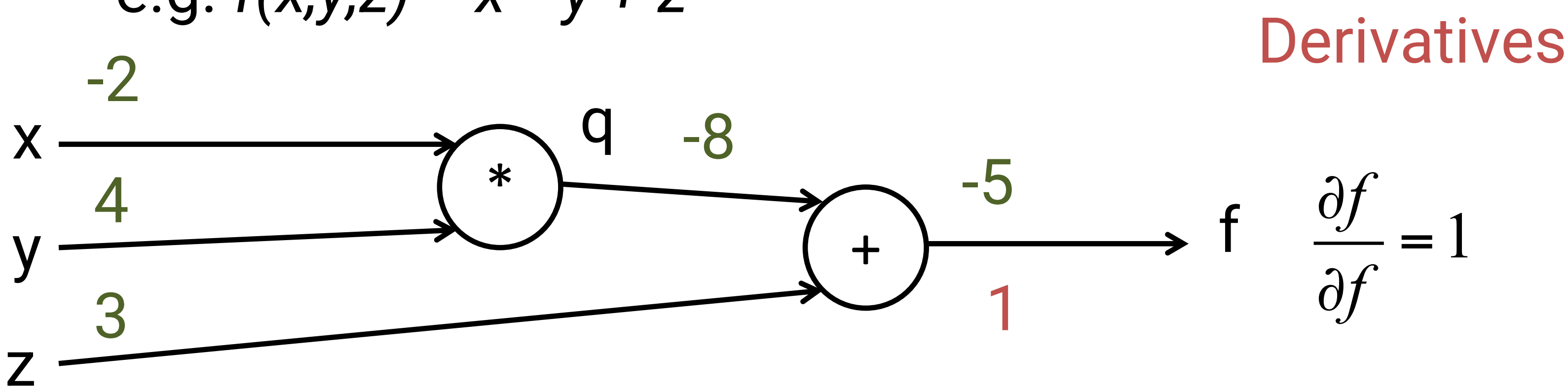
- Idea 3: computational graph & backpropagation
  - Represent calculation as a graph,
    e.g. $f(x,y,z) = x * y + z$



  - Calculate local gradients for each element, and then backpropagate to the previous element

Gradient Calculation

- Idea 3: computational graph & backpropagation
  - Represent calculation as a graph, e.g. $f(x,y,z) = x * y + z$

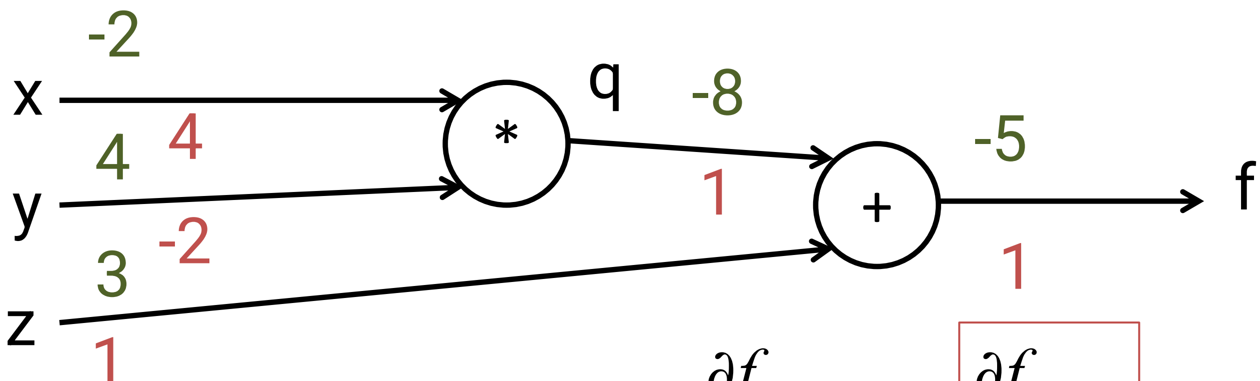


  - Calculate local gradients for each element, and then backpropagate to the previous element

University of Ljubljana
Faculty of Computer and Information Science

Gradient Calculation

- Idea 3: computational graph & backpropagation
  - Represent calculation as a graph, e.g. *f(x,y,z) = x * y + z*



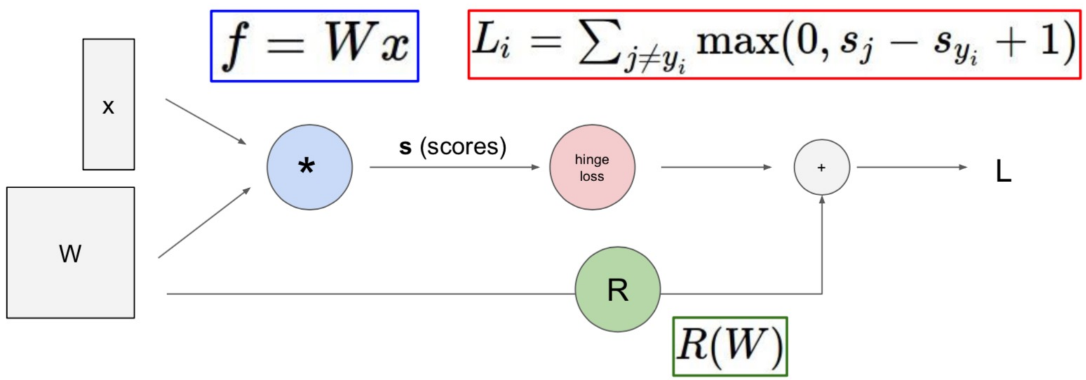
$$\frac{\partial f}{\partial f} = 1$$

$$\frac{\partial q}{\partial x} = y \qquad \frac{\partial q}{\partial y} = x \qquad \frac{\partial f}{\partial q} = 1 \qquad \boxed{\frac{\partial f}{\partial z} = 1}$$

Chain rule: $$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} \qquad \boxed{\frac{\partial f}{\partial x} = 4} \qquad \boxed{\frac{\partial f}{\partial y} = -2}$$

Neural Network Training

- But I have matrices (X, W_1, b_1, W_2, b_2, etc.)?



- The principle is the same – both forward and backward passes boil down to matrix algebra
  - However, calculating the gradient on all training samples is extremely time consuming
  - Stochastic gradient descent: a random subsample of the training data is used to calculate the gradient

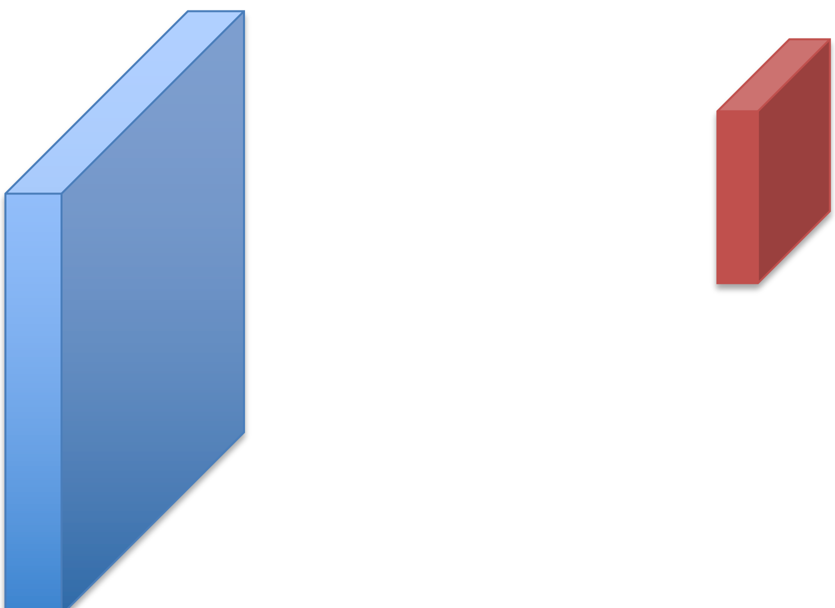
Convolutional Neural Networks

- Inspiration: how the brain works
  - Nearby cells in the cortex represent nearby regions in the visual field
  - Hierarchical cell organization:
    - Simple cells – detect light
    - Complex cells – detect light orientation and movement
    - Hypercomplex cells – response to movement with endpoint
- Applications of CNNs:
  - Image analysis (object recognition, segmentation, captioning)
  - Sound analysis (speech recognition, generation)
  - Many other purposes

Convolutional Layer

- Convolves (slides and calculates a dot product) the filter with the input (image)

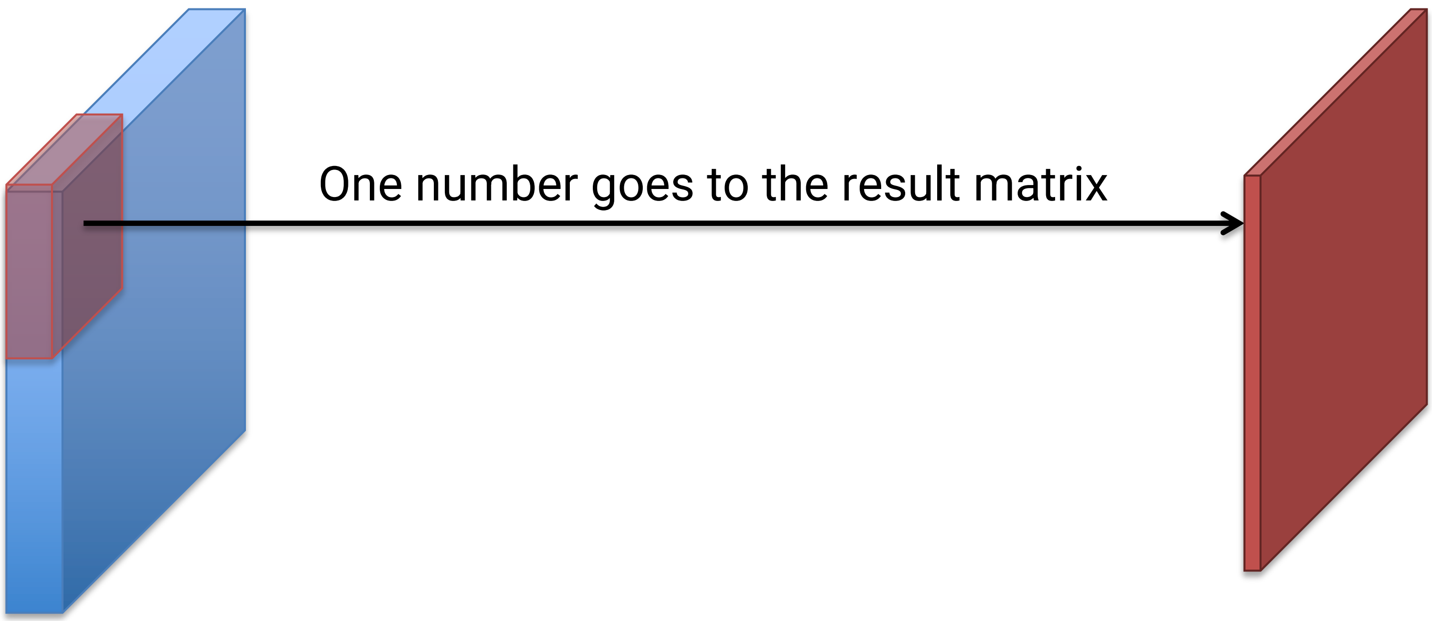
32x32x3 – image   5x5x3 – filter



Convolutional Layer

- Convolves (slides and calculates a dot product) the filter with the input (image)

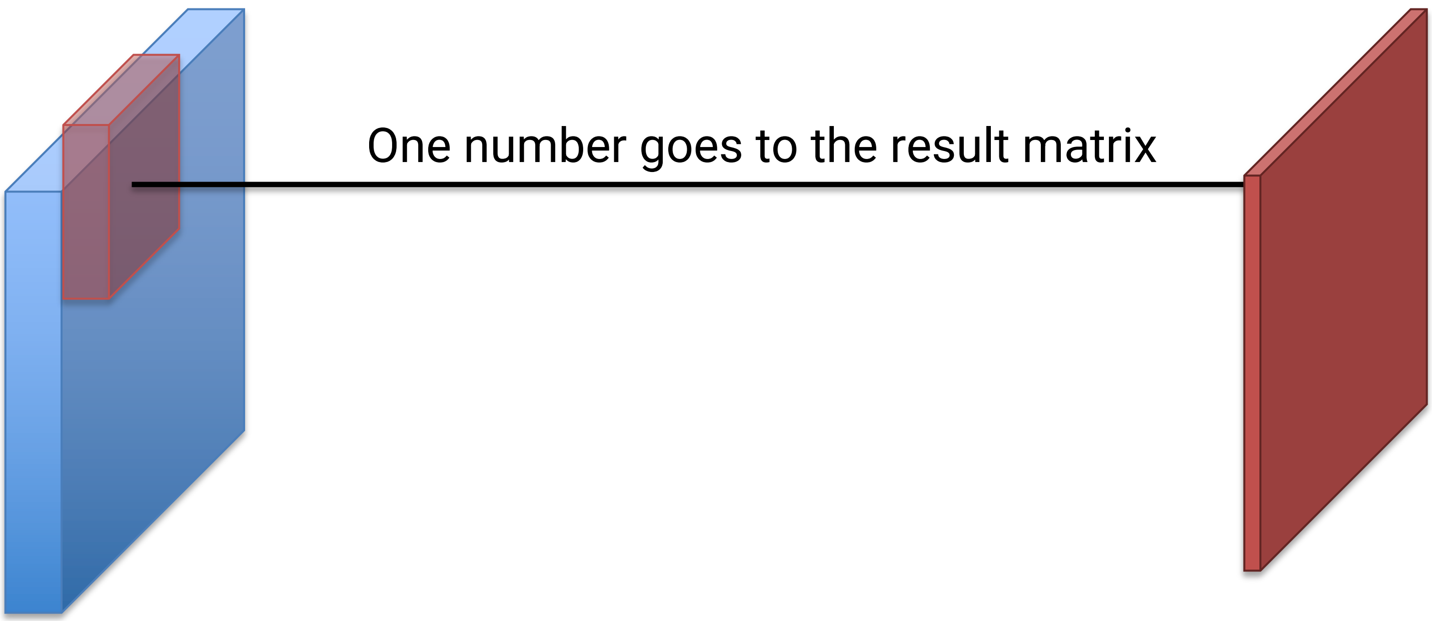
32x32x3 – image 5x5x3 – filter



Convolutional Layer

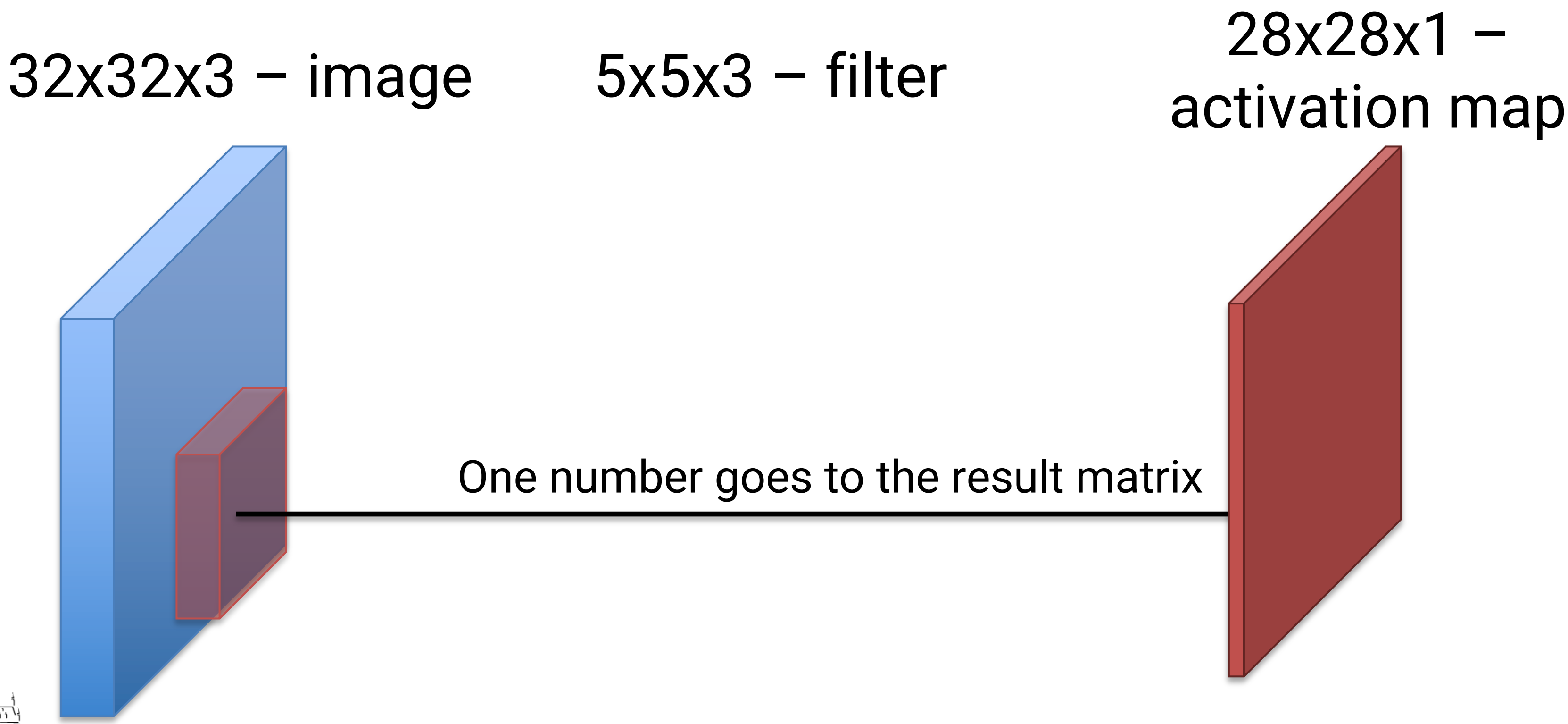
- Convolves (slides and calculates a dot product) the filter with the input (image)

32x32x3 – image 5x5x3 – filter



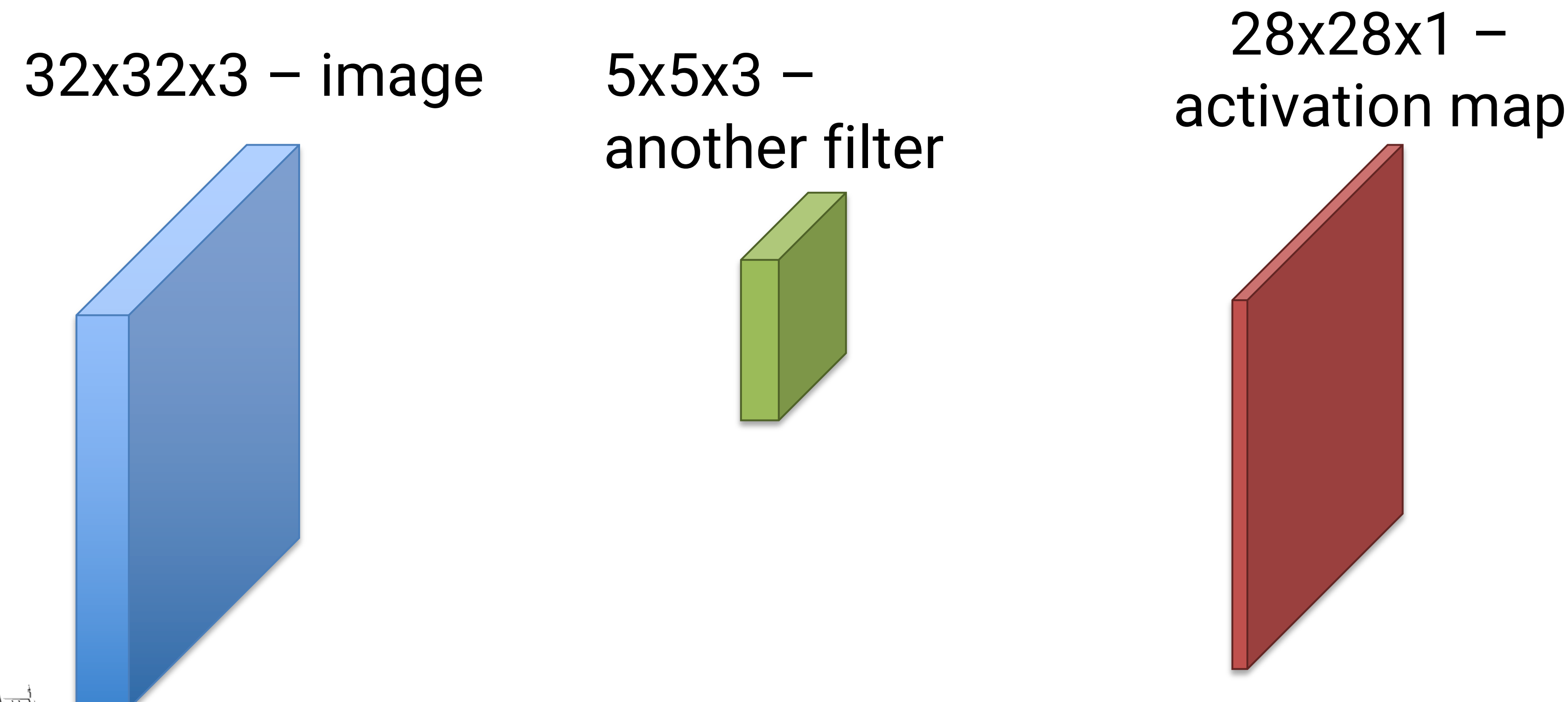
Convolutional Layer

- Convolves (slides and calculates a dot product) the filter with the input (image)



Convolutional Layer

- Convolves (slides and calculates a dot product) the filter with the input (image)



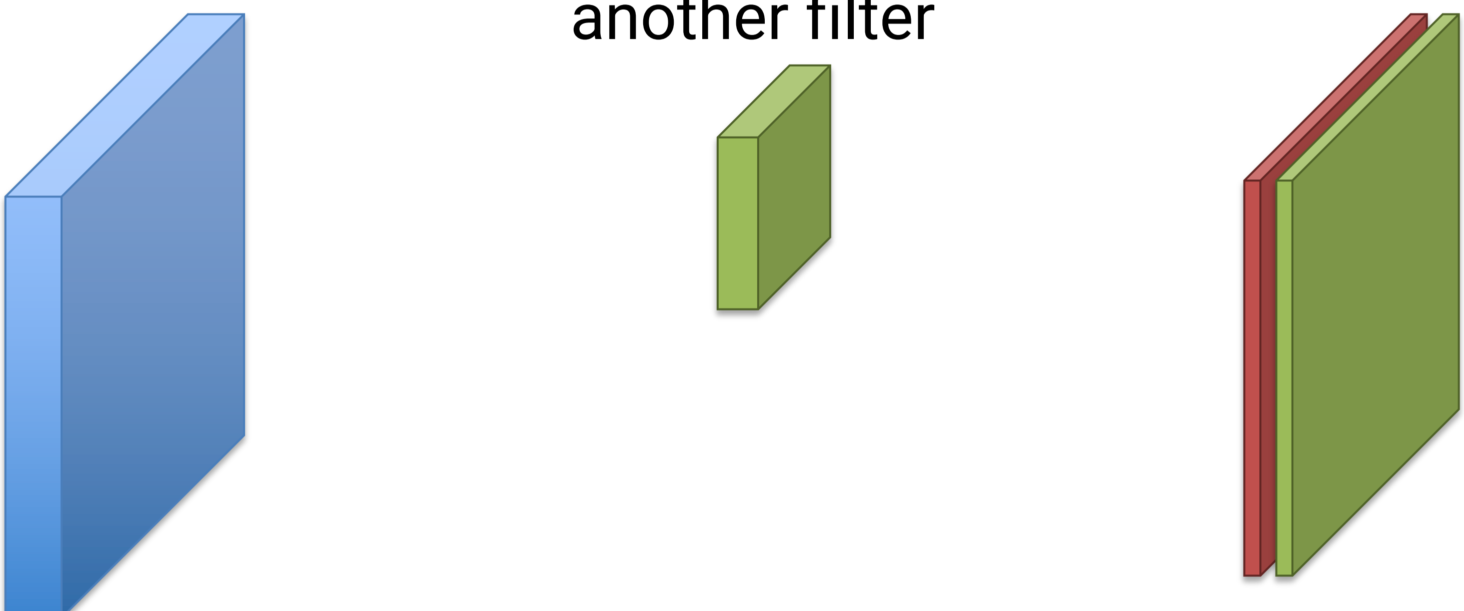
Convolutional Layer

- Convolves (slides and calculates a dot product) the filter with the input (image)

32x32x3 – image

5x5x3 – another filter

28x28x1 – one more activation map



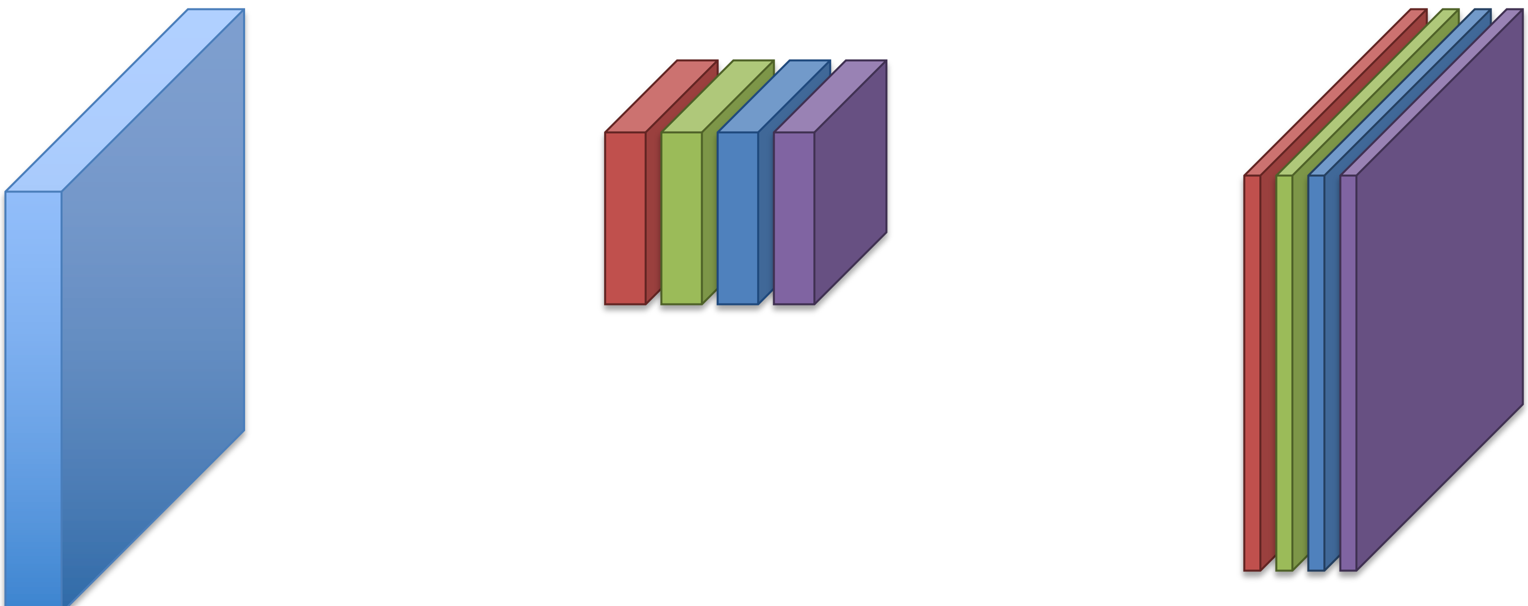
Convolutional Layer

- Convolves (slides and calculates a dot product) the filter with the input (image)

32x32x3 – image

four 5x5x3 filters

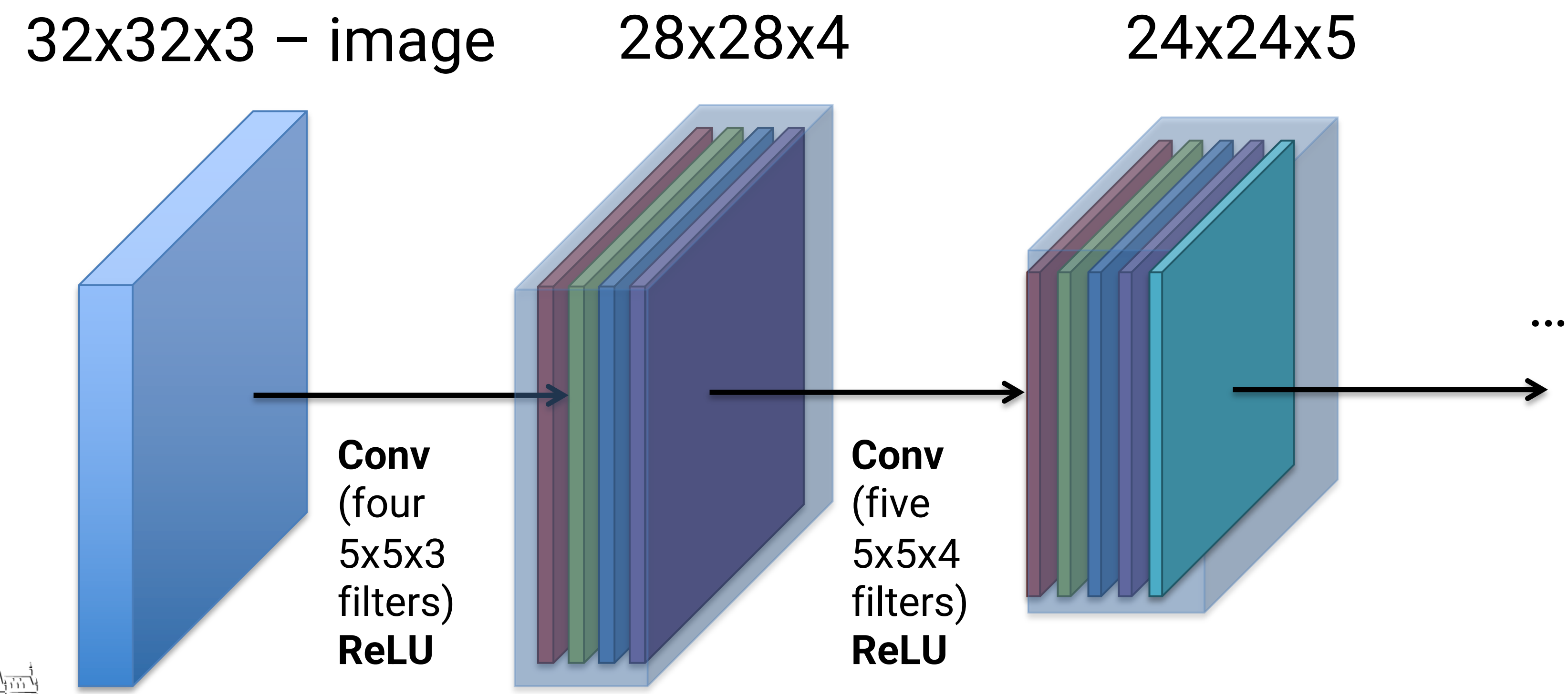
four 28x28x1 activation maps



University of Ljubljana
Faculty of Computer and Information Science

Convolutional Neural Network Structure

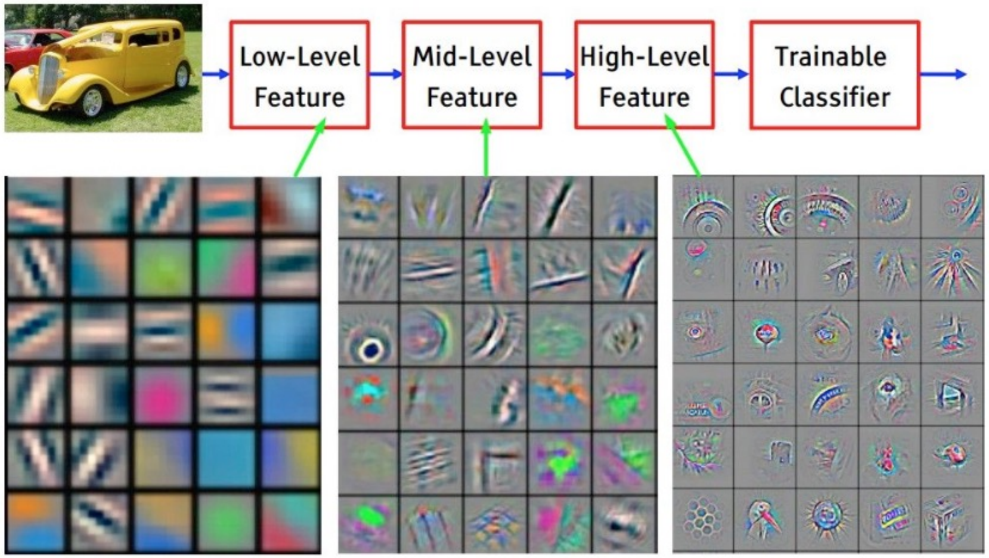
- A sequence of convolutional layers with activation layers (ReLU)



University of Ljubljana
Faculty of Computer and
Information Science

Feature Construction in CNN

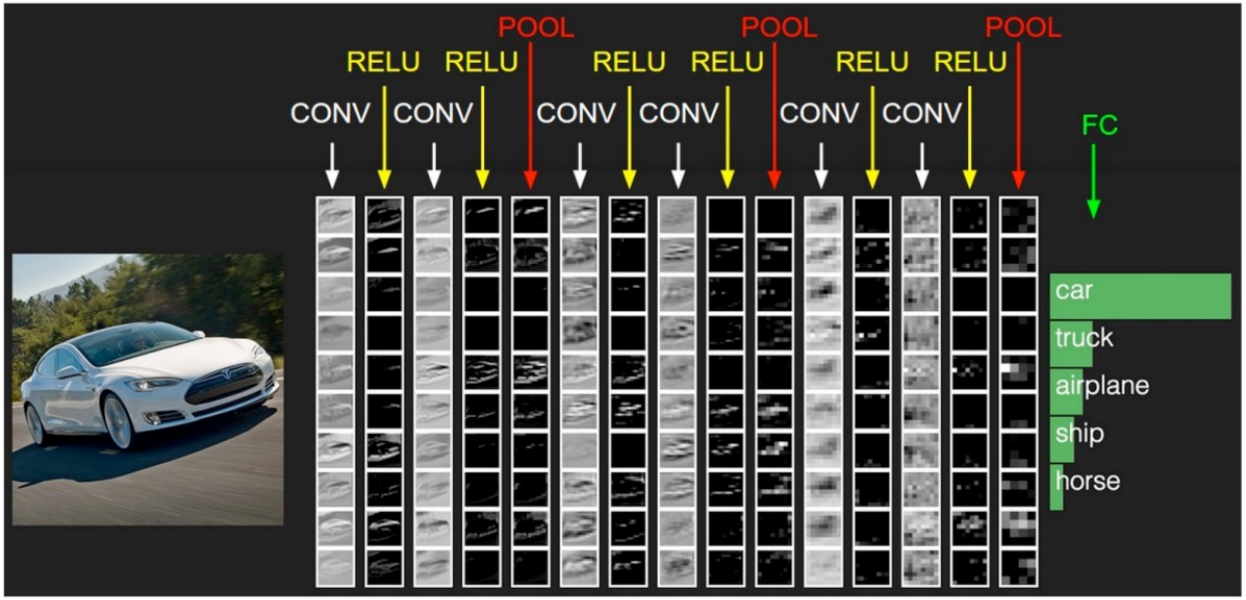
- Turns out that earlier layers detect simpler features (e.g. edges in an image), whereas further layers detect more complex features (e.g. specific blobs in an image)
  - Just like with human vision!



Classification in CNN

- The structure is (usually)
  - Input -> N x {Conv, ReLU}
  - Occasionally Pooling
    - MaxPooling most often
  - Finish with fully connected layer for classification

Reducing dimensionality, from e.g. 32x32 activation map to 16x16 activation map



Some Other Neural Network Types

- Recurrent Neural Networks (RNN)
  - Unlike a feed-forward network, can internally process (loop) the result of the calculation
  - Good for temporal data processing
- Long Short-Term Memory (LSTM)
  - RNN with “forgetting”
  - Great for speech processing
- Autoencoder
  - A simple network used for feature compression