WBL Deep Learning:: Lecture 2

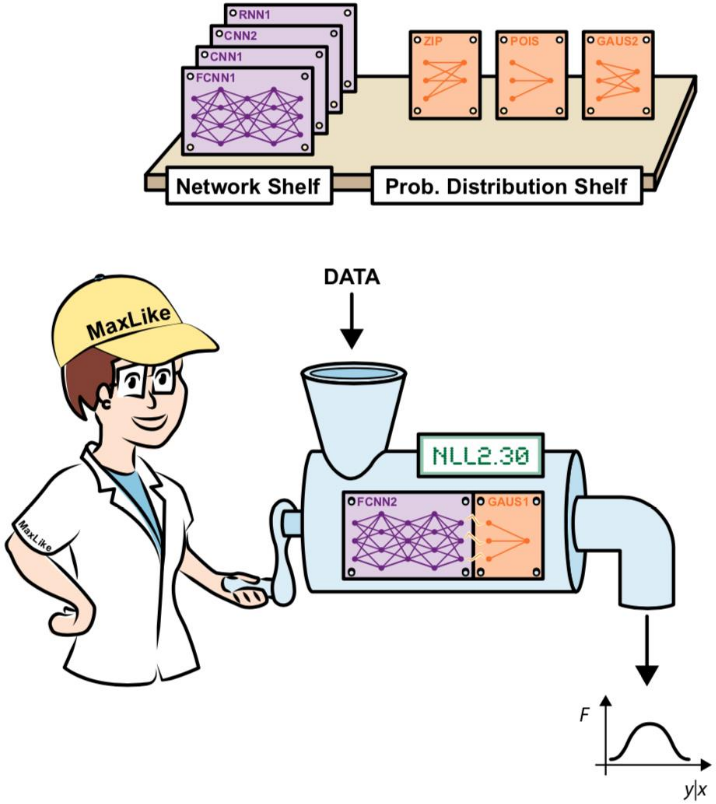
Beate Sick, Oliver Dürr

Convolutional Neural Networks

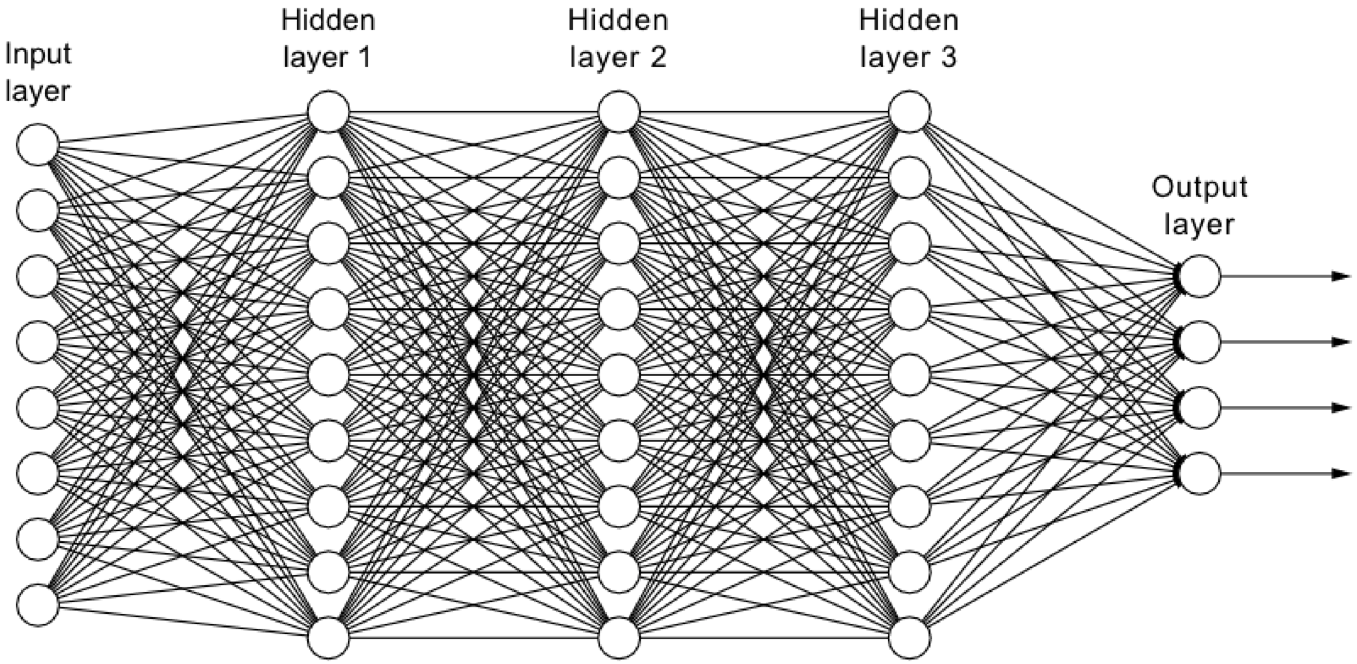
Zürich, 9/12/2022

Topics of today

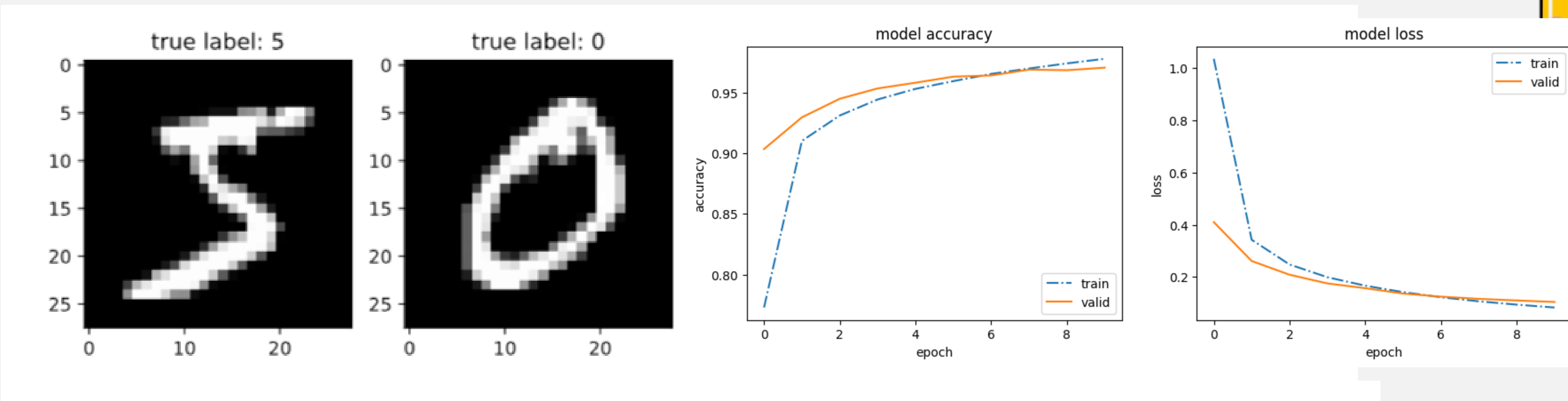
- Convolutional Neural Networks (CNN) for images
  - Introduction of convolution
  - What does a CNN look at
  - Tricks of the trade
    - Data Augmentation
    - Dropout during training
    - Batch Norm
    - Skip connection
  - Image challenge winning architectures
  - Few data and DL
  - 1D convolution for sequence data

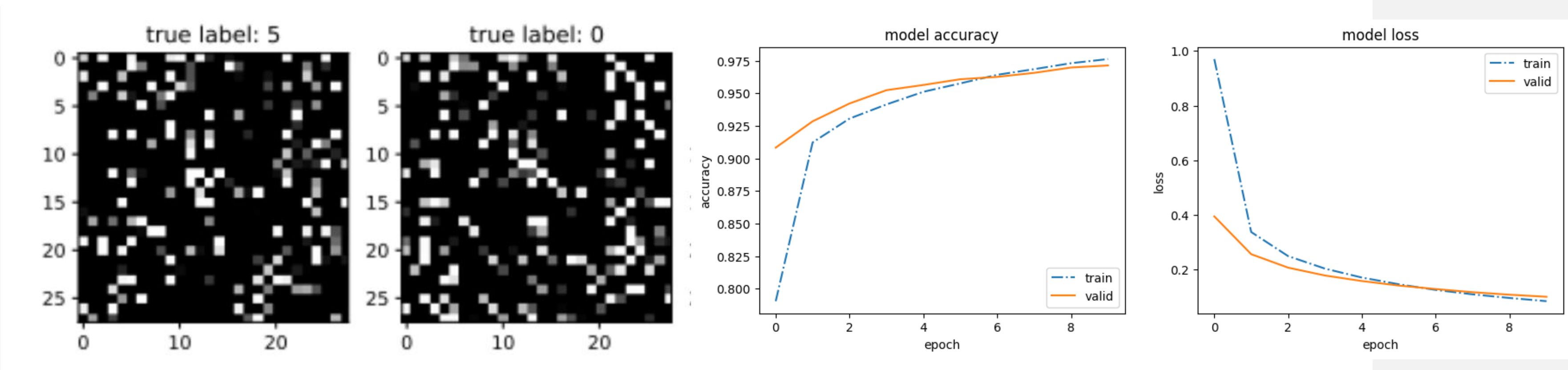


Recall: Architecture of a fully connected NN



MNIST exercise: Does shuffling disturb a fcNN?

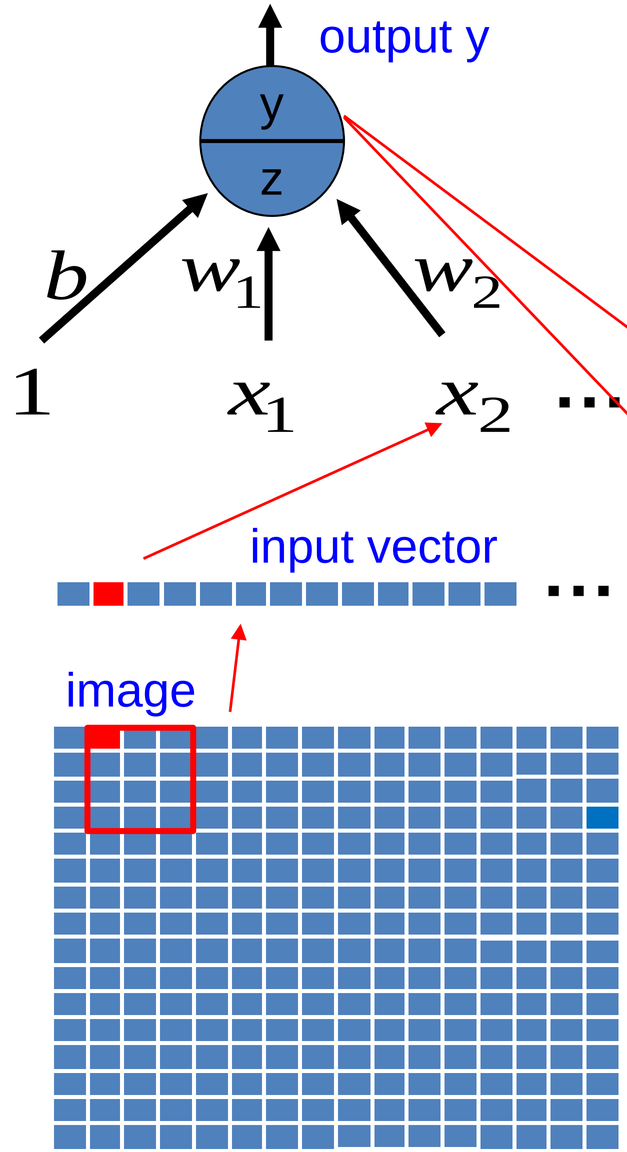




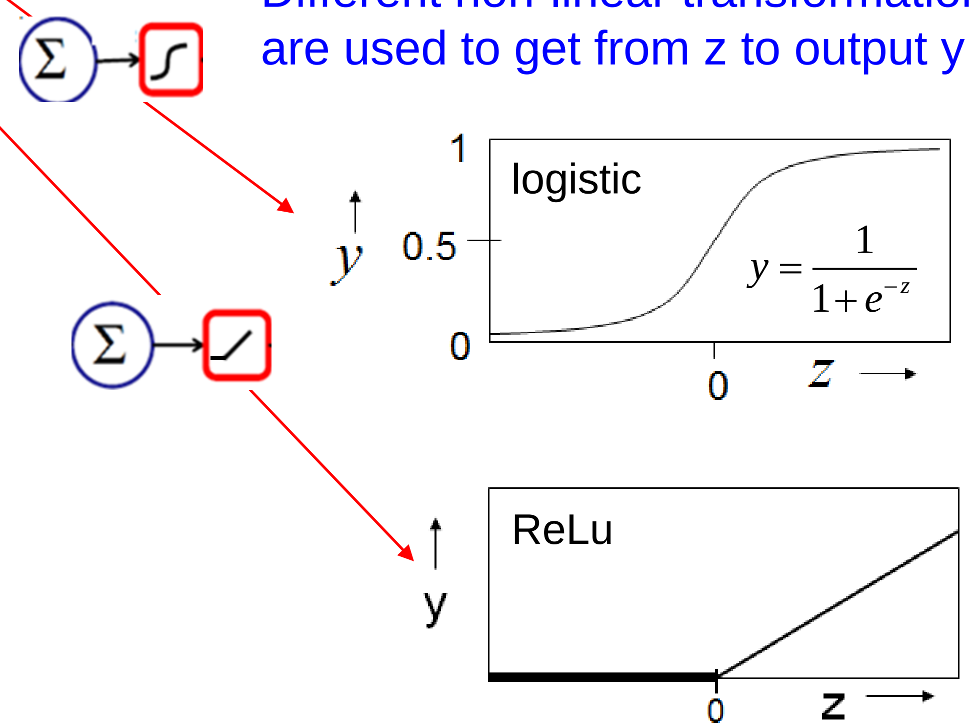
→ The performance of a fcNN is the same on original and shuffled images

An artificial neuron

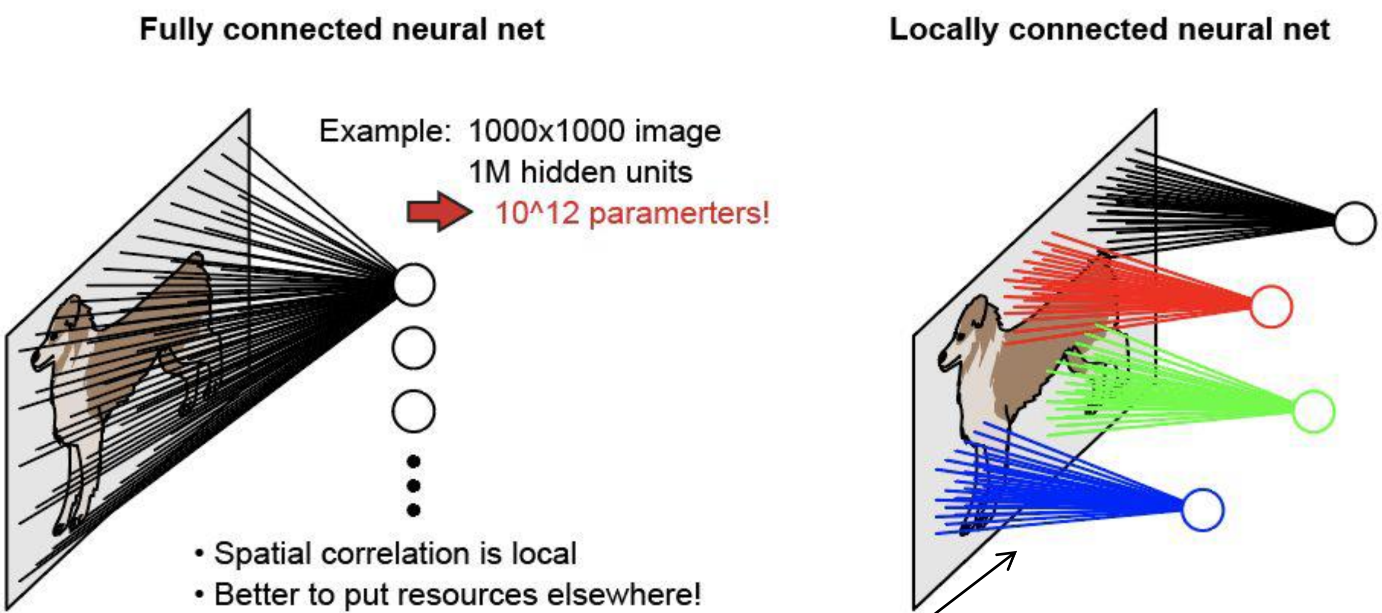
$$z = \overset{\text{bias}}{\downarrow}{b} + \sum_i x_i \overset{\text{weights}}{\downarrow}{w_i}$$



Different non-linear transformations are used to get from z to output y



Convolution extracts local information using few weights



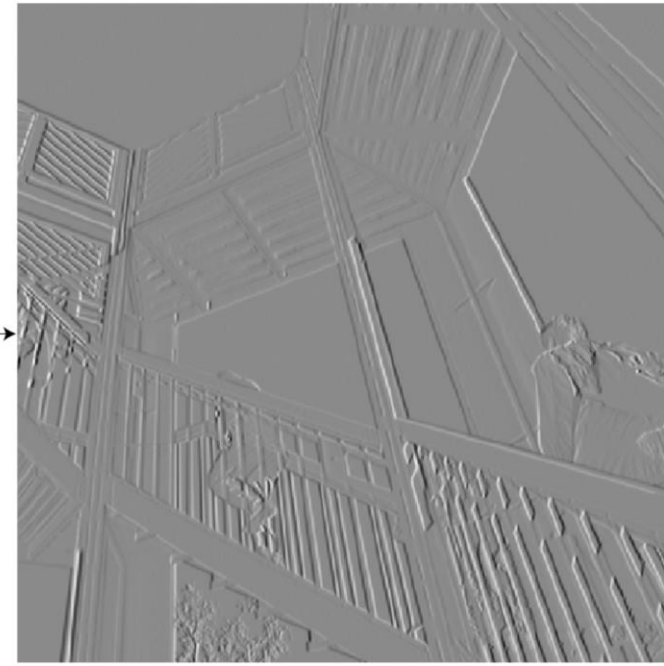
Shared weights:

by using the same weights for each patch of the image we need much less parameters than in the fully connected NN and get from each patch the same kind of local feature information such as the presence of an edge.

Example of designed Kernel / Filter

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

Horizontal Sobel kernel



Applying a vertical edge detector kernel

Taken from the great website on convolution: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

Convolution

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

Input X

1	0	1
0	1	0
1	0	1

Kernel W

4	3	4
2	4	3
2	3	4

Result Z

Convolution (let's ignore bias b):

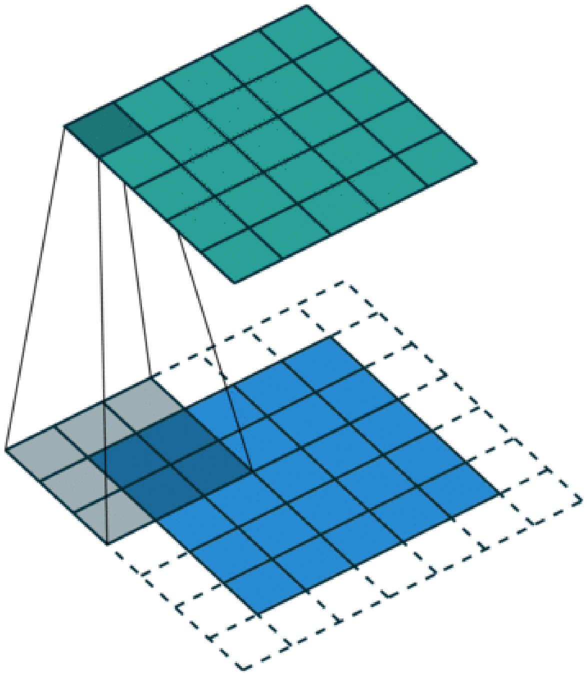
$$z = b + \sum_i x_i w_i$$

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

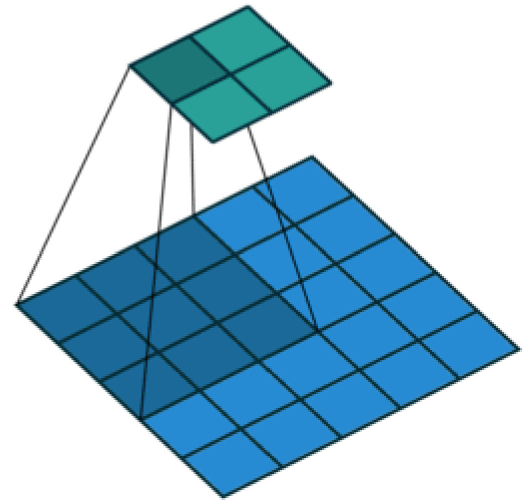
4		

https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

CNN Ingredient I: Convolution



Zero-padding to achieve
same size of feature and input



no padding to only use
valid input information

The *same* weights are used at each position of the input image.

Taken from the great website on convolution: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

Exercise: Do one convolution step by hand

The kernel is 3x3 and is applied at each valid positon
– how large is the resulting activation map?

The small numbers in the shaded region are the kernel weights.
Determine the position and the value within the resulting activation map.

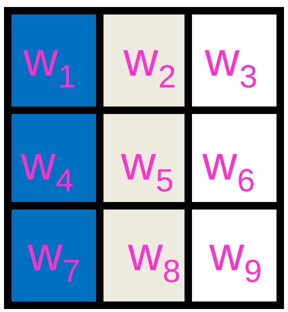
3	3	2	1	0
0_0	0_1	1_2	3	1
3_2	1_2	2_0	2	3
2_0	0_1	0_2	2	2
2	0	0	0	1

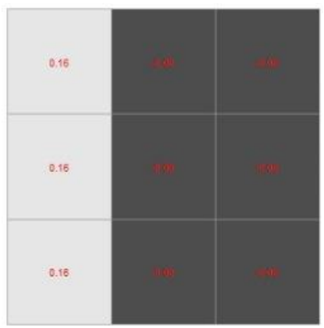
Convolutional networks use neighborhood information and replicated local feature extraction

In a locally connected network, the calculation rule

$$z = b + \sum_i x_i w_i$$

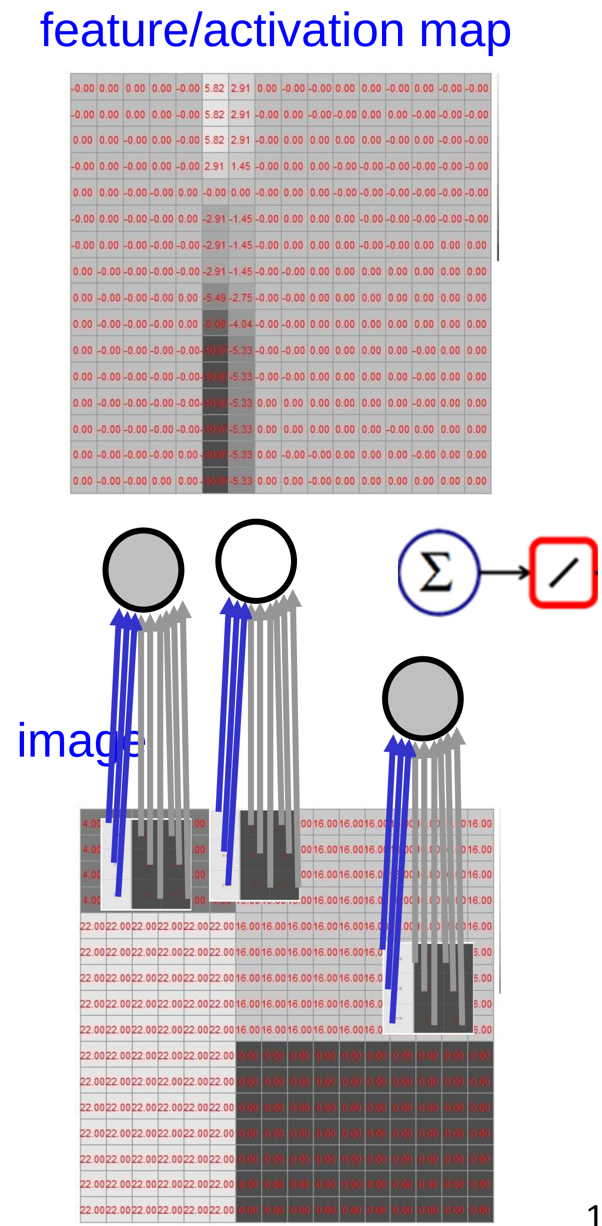
Pixel values in a small image patch are element-wise multilied with weights of a small filter/kernel:



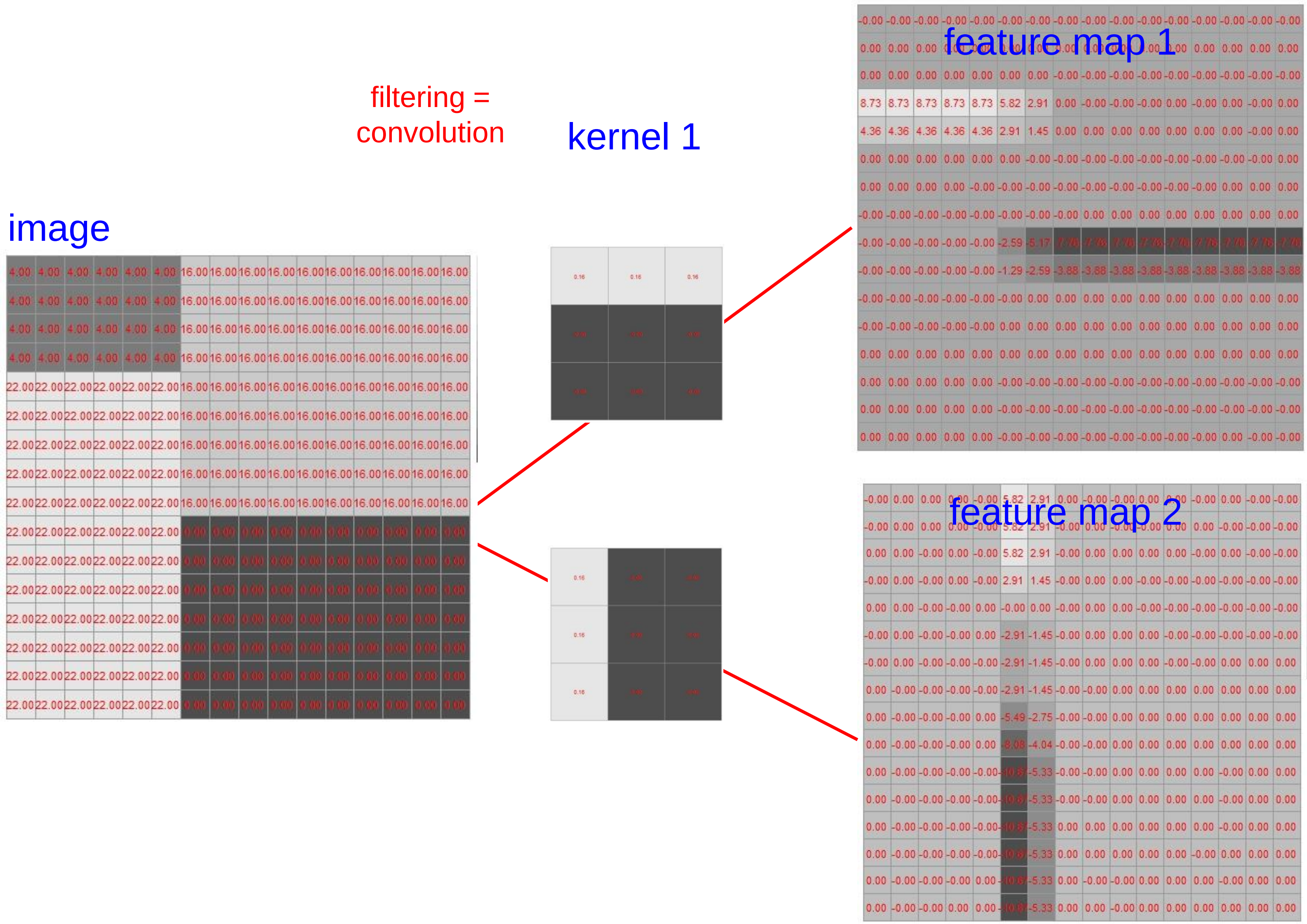


The filter is applied at each position of the images, and it can be shown that the result is maximal if the image pattern corresponds to the weight pattern.

The results form again an image called feature map (=activation map) which shows at which position the feature is present.

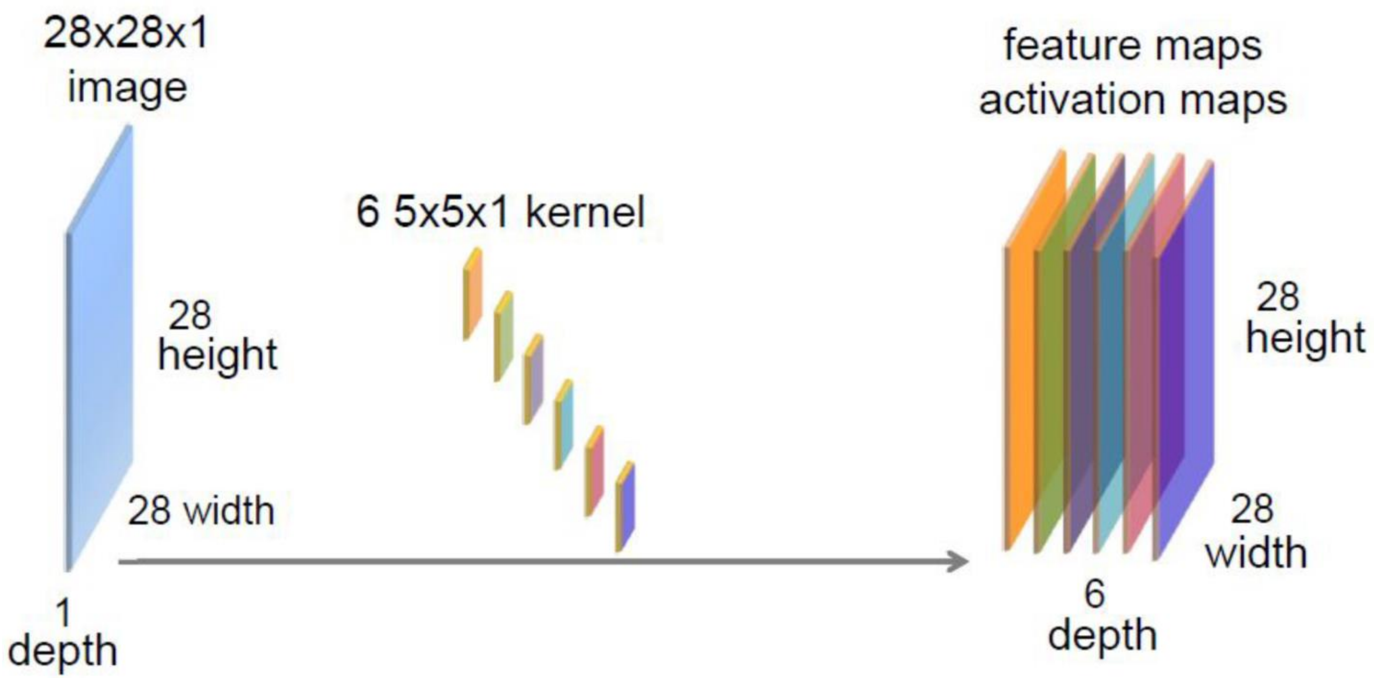


Convolutional networks use neighborhood information and replicated local feature extraction



The weights of each filter are randomly initiated and then adapted during the training.

Convolution layer with a 1-chanel input and 6 kernels

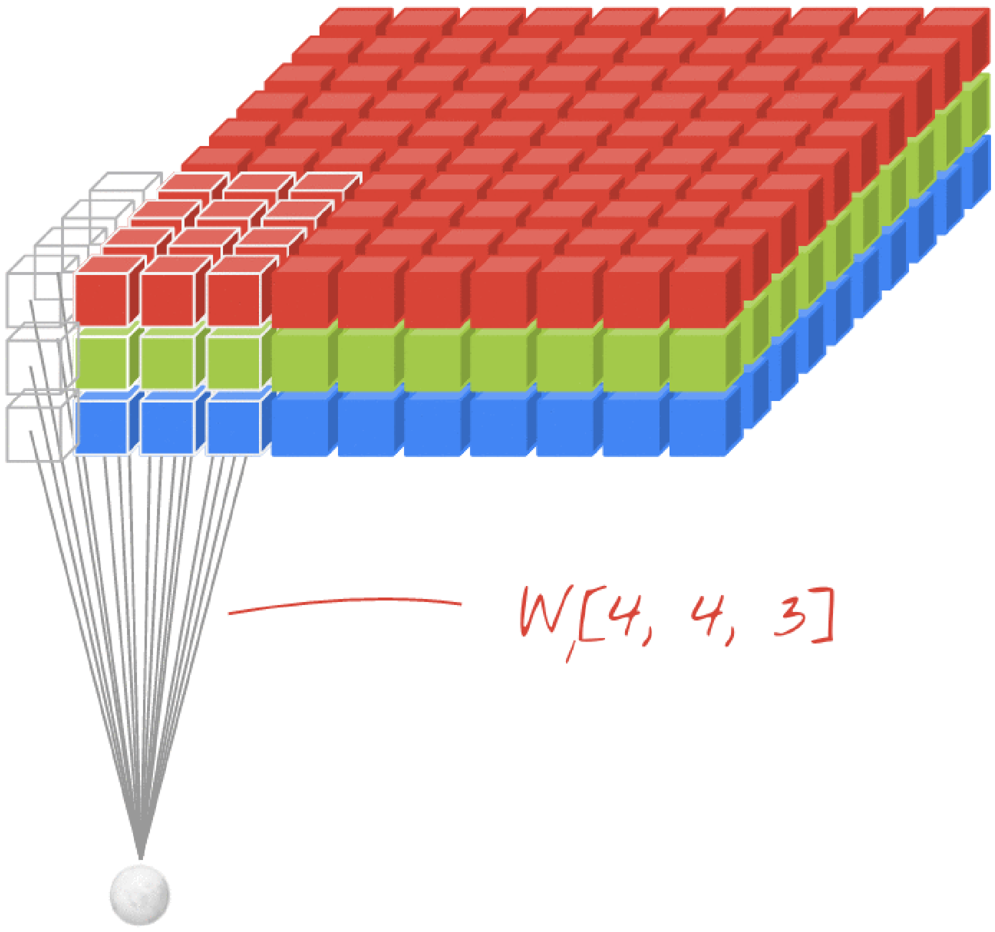


Convolution of the input image with 6 different kernels results in 6 activation maps.

If the input image has only one channel, then each kernel has also only one channel.

Animated convolution with 3 input channels

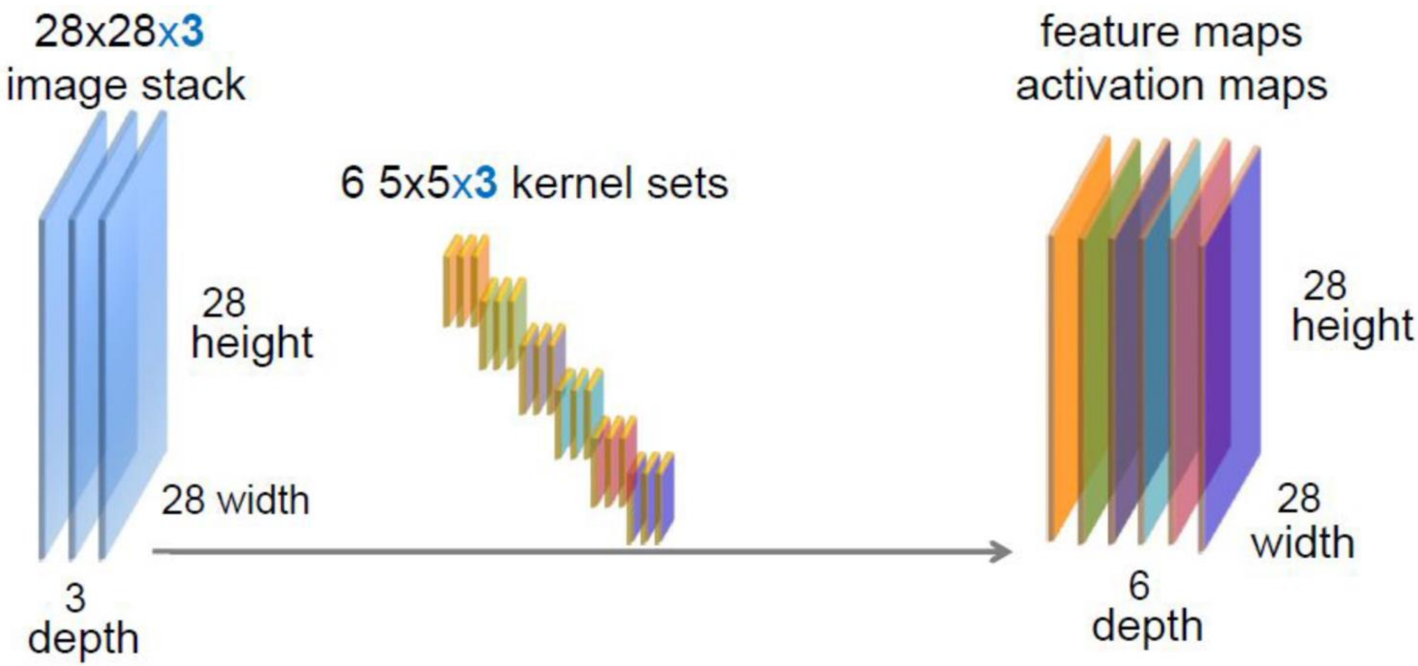
3 color channel input image



$$z = b + \sum_i x_i w_i$$

Animation credits: M.Gorner, https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#10
For an example with number see convolution demo in: https://cs231n.github.io/convolutional-networks/

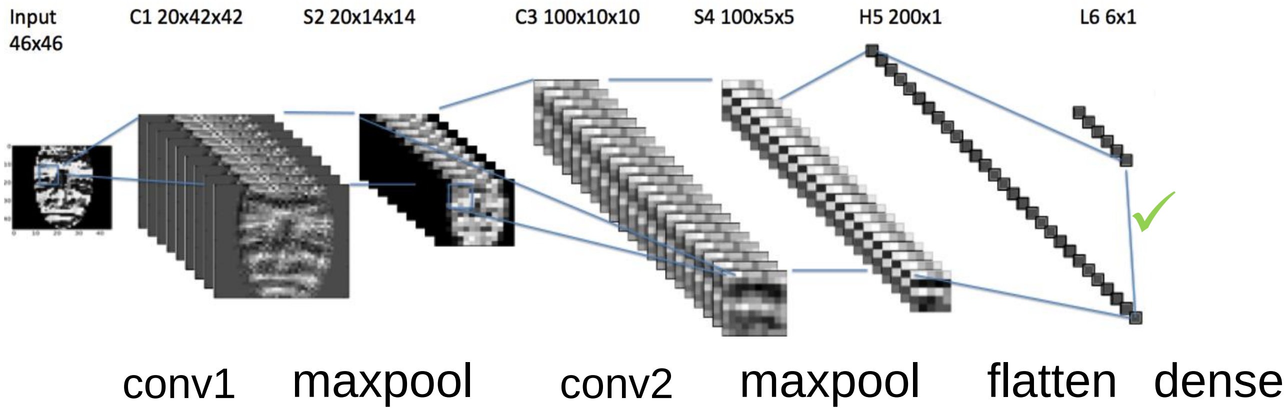
Convolution layer with a 3-chanel input and 6 kernels



Convolution of the input image with 6 different kernels results in 6 activation maps.

If the input image has 3 channels, then each filter has also 3 channels.

CNN ingredient II: Maxpooling Building Blocks reduce size



4x4

12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

→

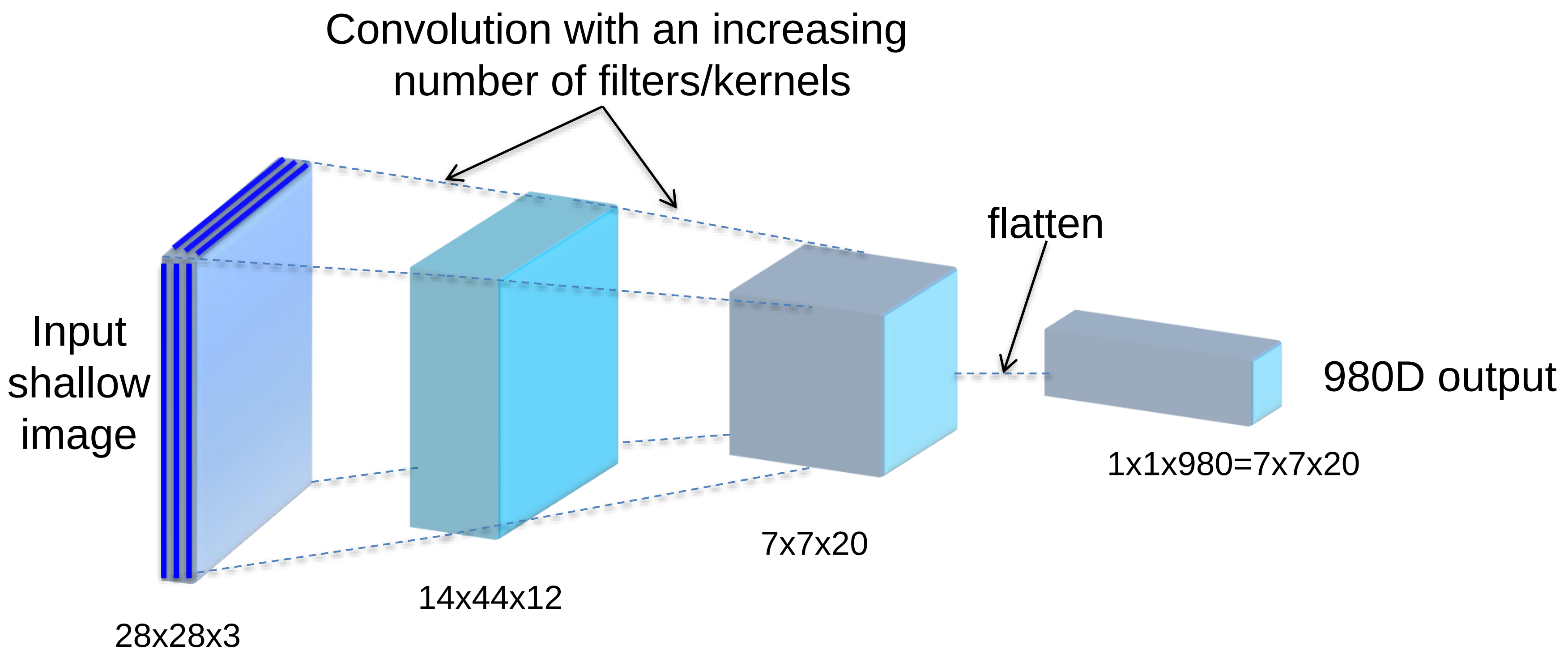
2x2

20	30
112	37

Simply join e.g. 2x2 adjacent pixels in one by taking the max.
→ less weights in model
→ Less train data needed
→ increased performance

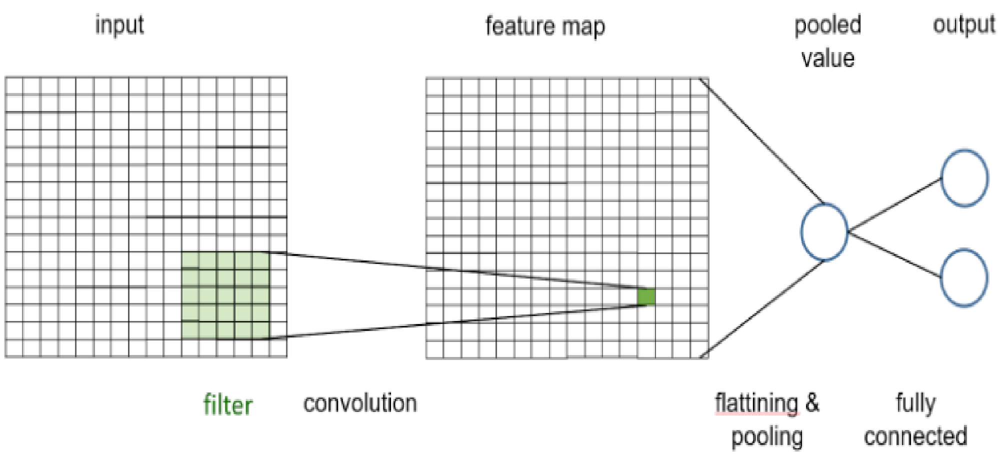
Hinton: „The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster"

Typical shape of a classical CNN



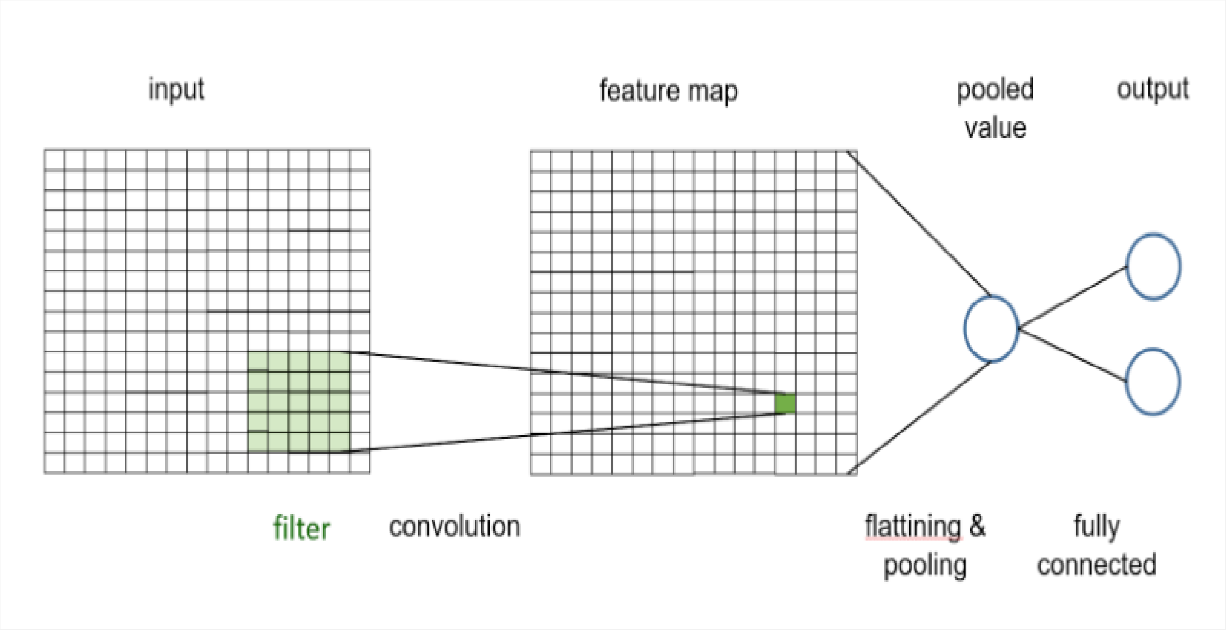
Spatial resolution is decreased e.g. via max-pooling while more abstract image features are detected in deeper layers.

Building a very simple CNN with keras




```
model <- keras_model_sequential()
model %>%
  layer_conv_2d(filters=1,
                kernel_size = c(5,5),
                padding = 'same',
                input_shape = c(pixel,pixel,1),
                activation = 'linear') %>%
  # take the max over all values in the activation map
  layer_max_pooling_2d(pool_size = c(pixel,pixel)) %>%
  layer_flatten() %>%
  layer_dense(units = 2,activation = 'softmax')
```

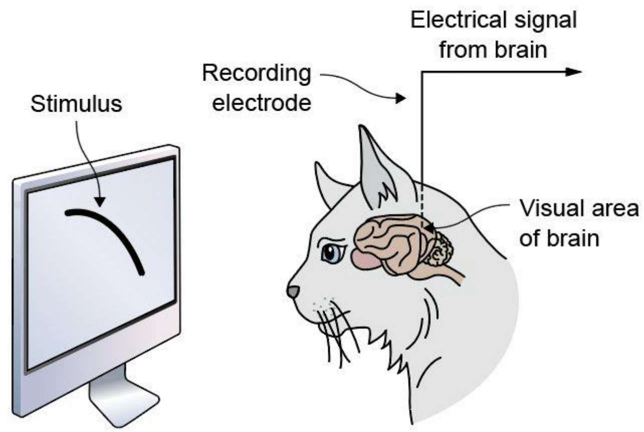
Exercise: Artstyle Lover



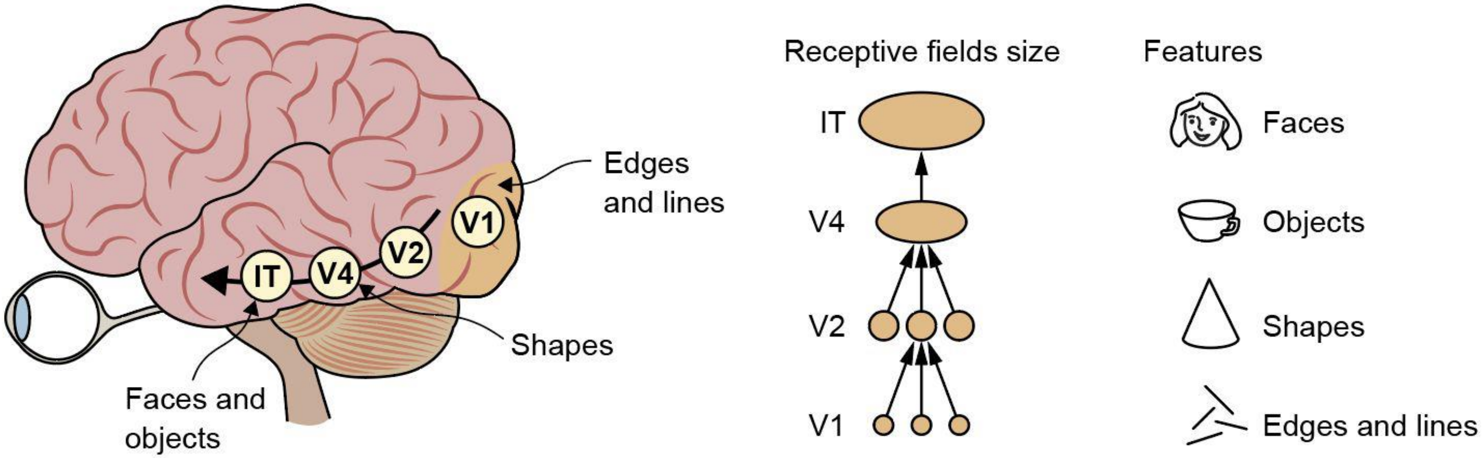
Open NB in: https://github.com/tensorchiefs/dl_rcourse_2022/blob/main/notebooks/03_nb_ch02_03.ipynb

Biological Inspiration of CNNs

How does the brain respond to visually recieved stimuli?

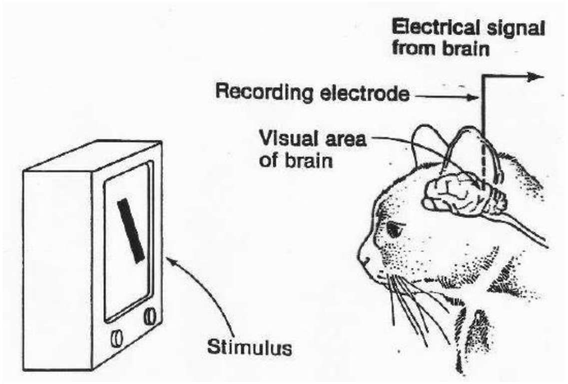


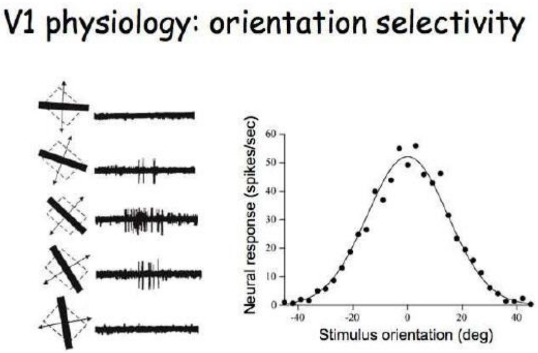
Setup of the experiment of Hubel and Wiesel in late 1950s in which they discovered neurons in the visual cortex that responded when moving **edges** were shown to the cat.

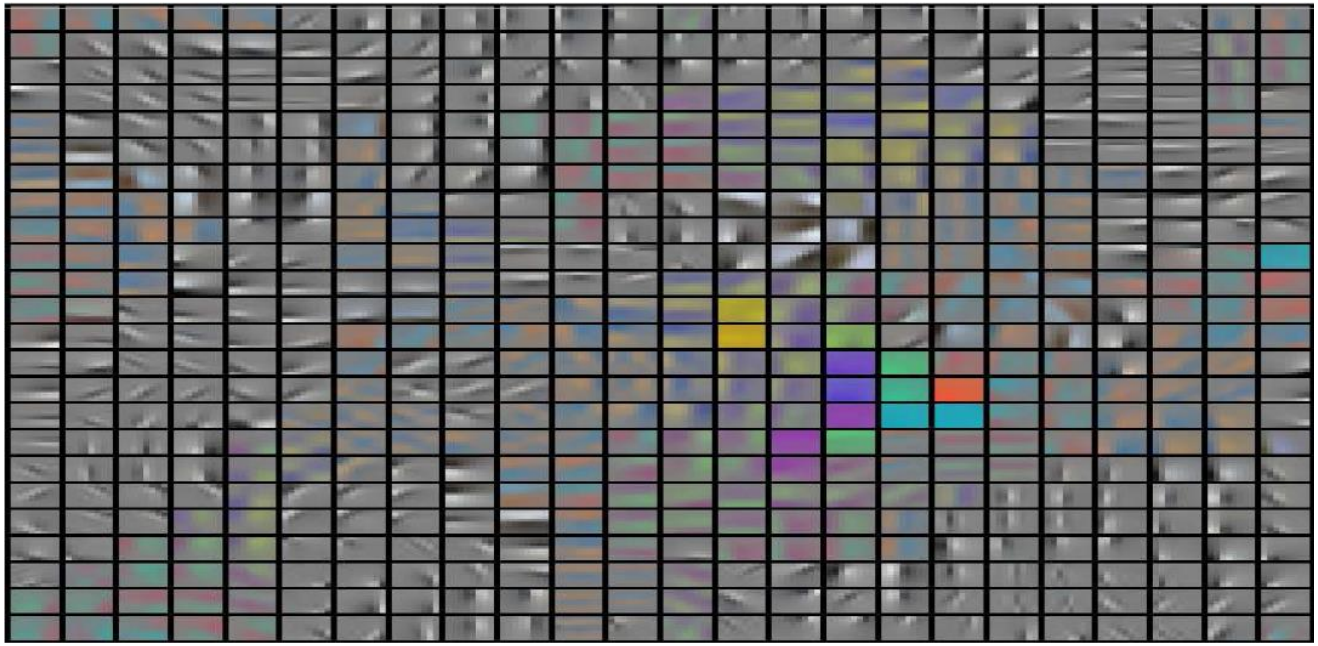


Organization of the visual cortex in a brain, where neurons in different regions respond to more and more complex stimuli

Compare neurons in brain region V1 in first layer of a CNN





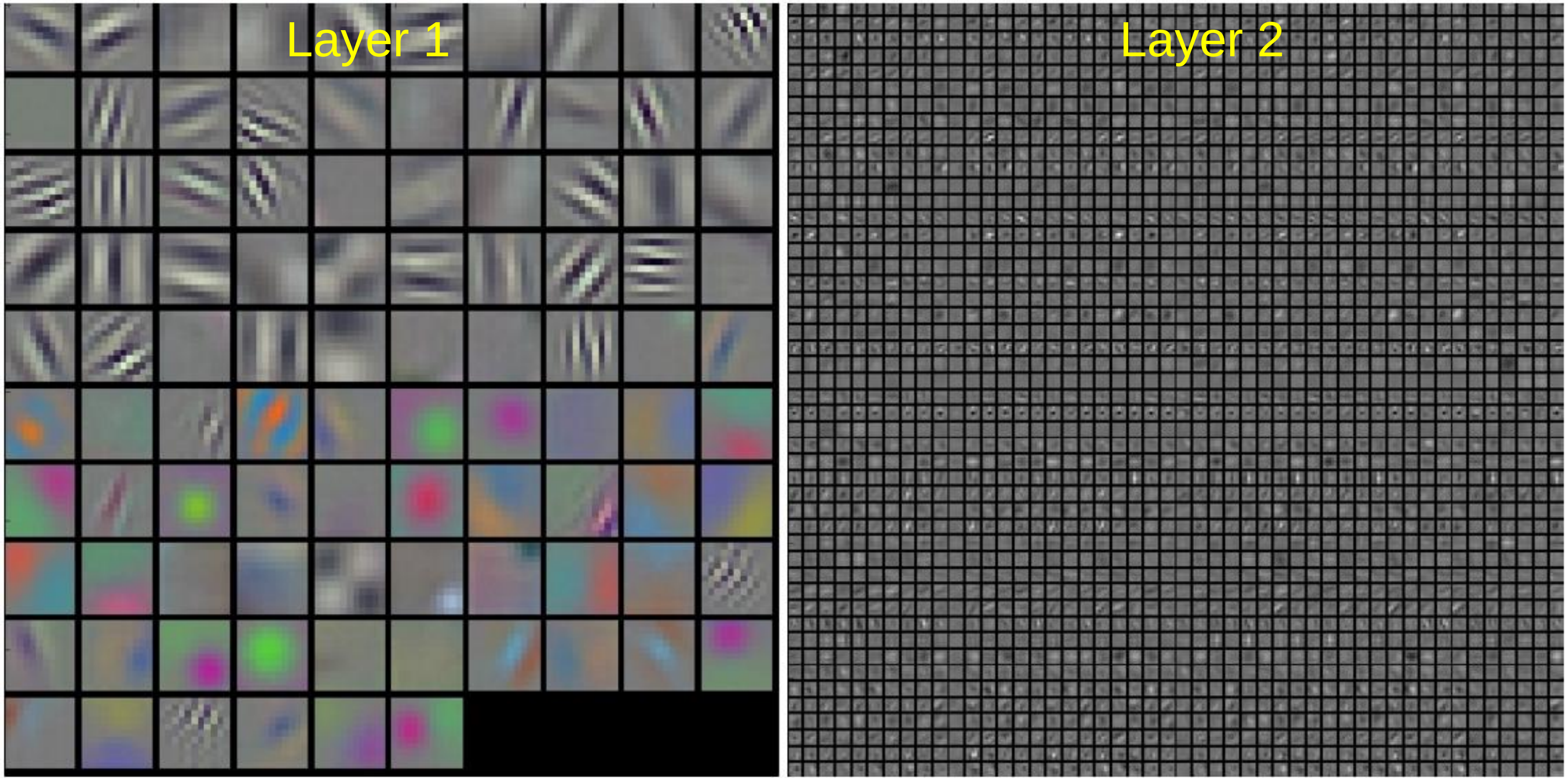


Neurons in brain region V1 and neurons in 1. layer of a CNN respond to similar patterns

Slide (modified) credits: Nando de Freitas

Visualize the weights used in filters

Filter weights from a trained Alex Net



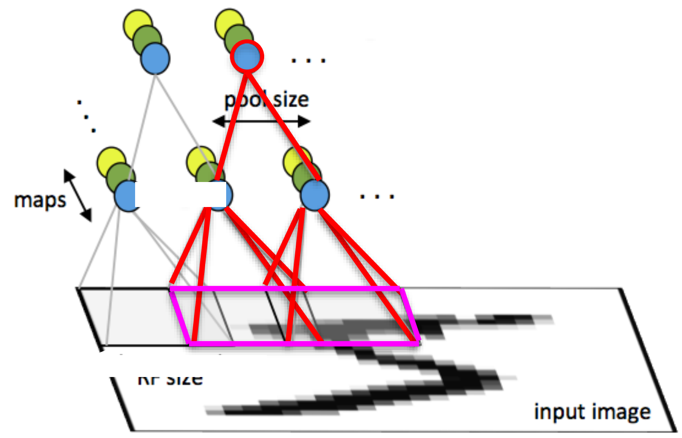
Only in layer 1 the filter pattern correspond to extracted patterns in the image.

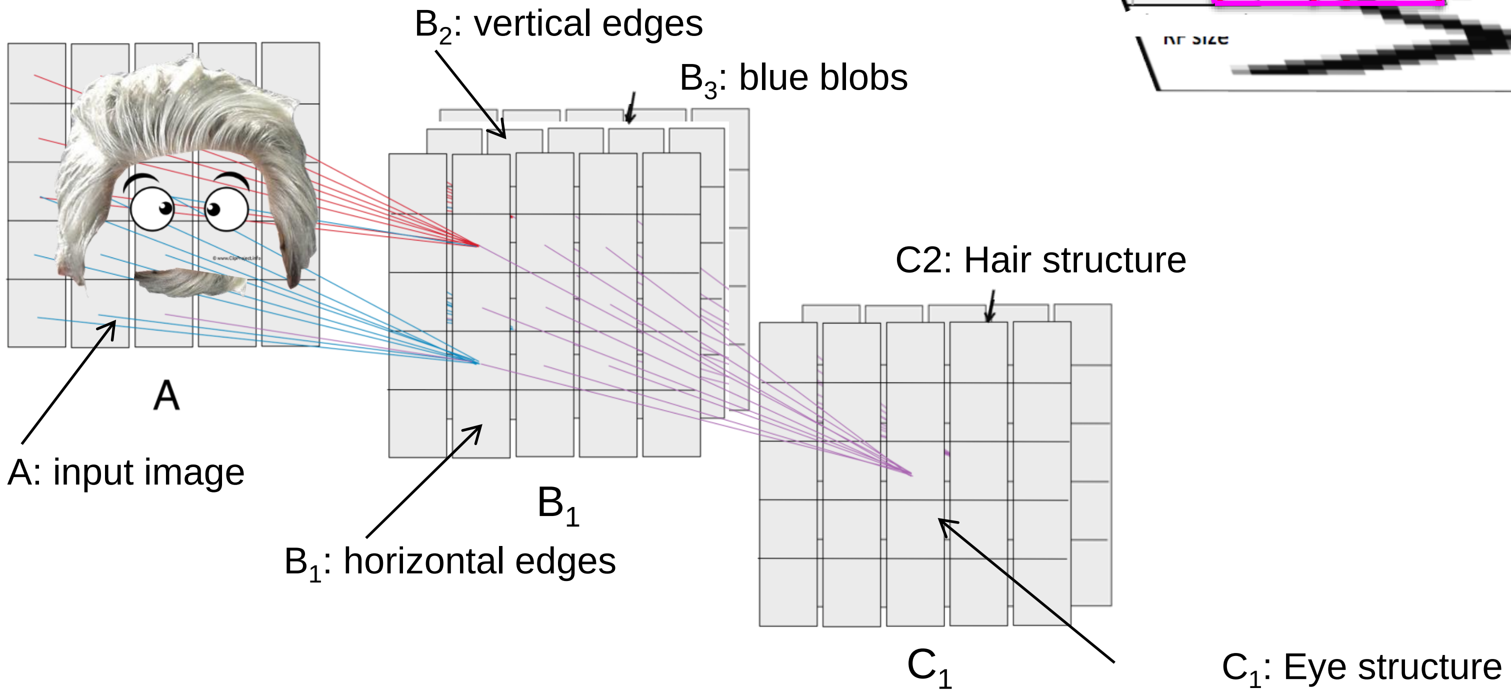
In higher layers we can only check if patterns look noisy, which would indicate that the network that hasn't been trained for long enough, or possibly with a too low regularization strength that may have led to overfitting.

Image credits: http://cs231n.github.io/understanding-cnn/

The receptive field

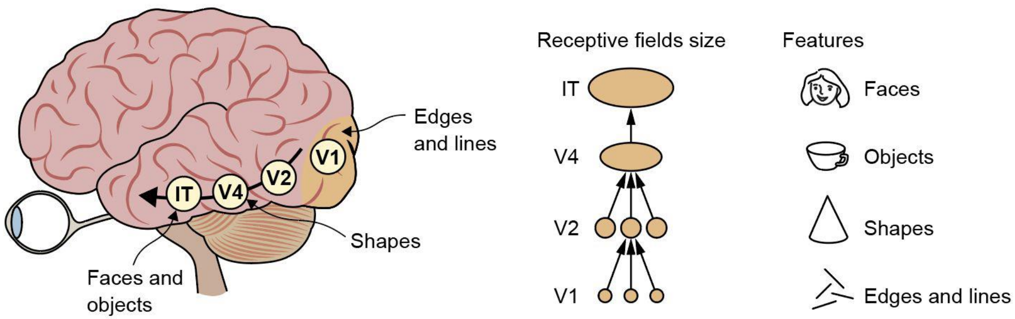
For each pixel of a feature map we can determine the connected area in the input image – this area in the input image is called receptive field.

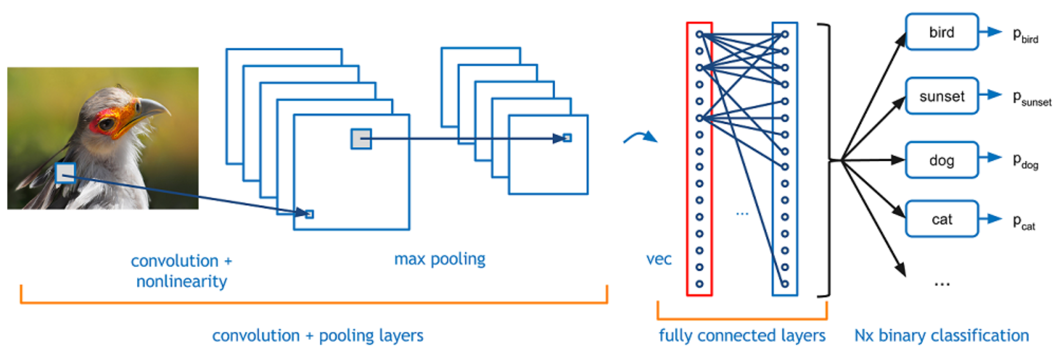


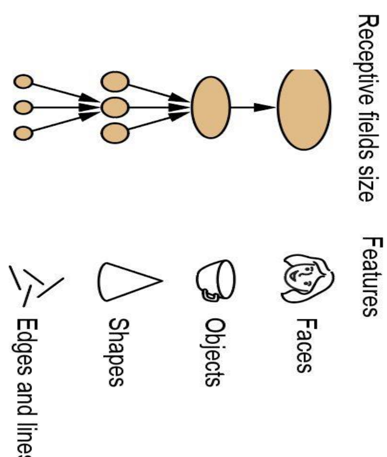


A feature map gets activated by a certain structure of the feature maps one layer below, which by itself depends on the input of a preceding layer etc and finally on the input image. Activation maps close to the input image are activated by simple structures in the image, higher maps by more complex image structures.

Weak analogies between brain and CNNs architecture

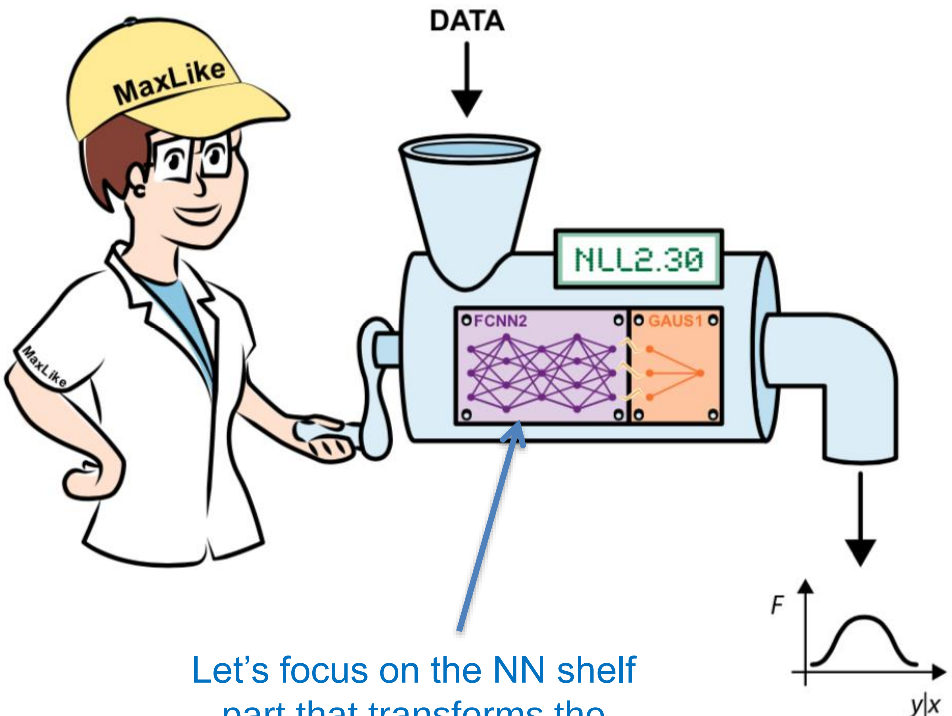






What does the CNN look at?

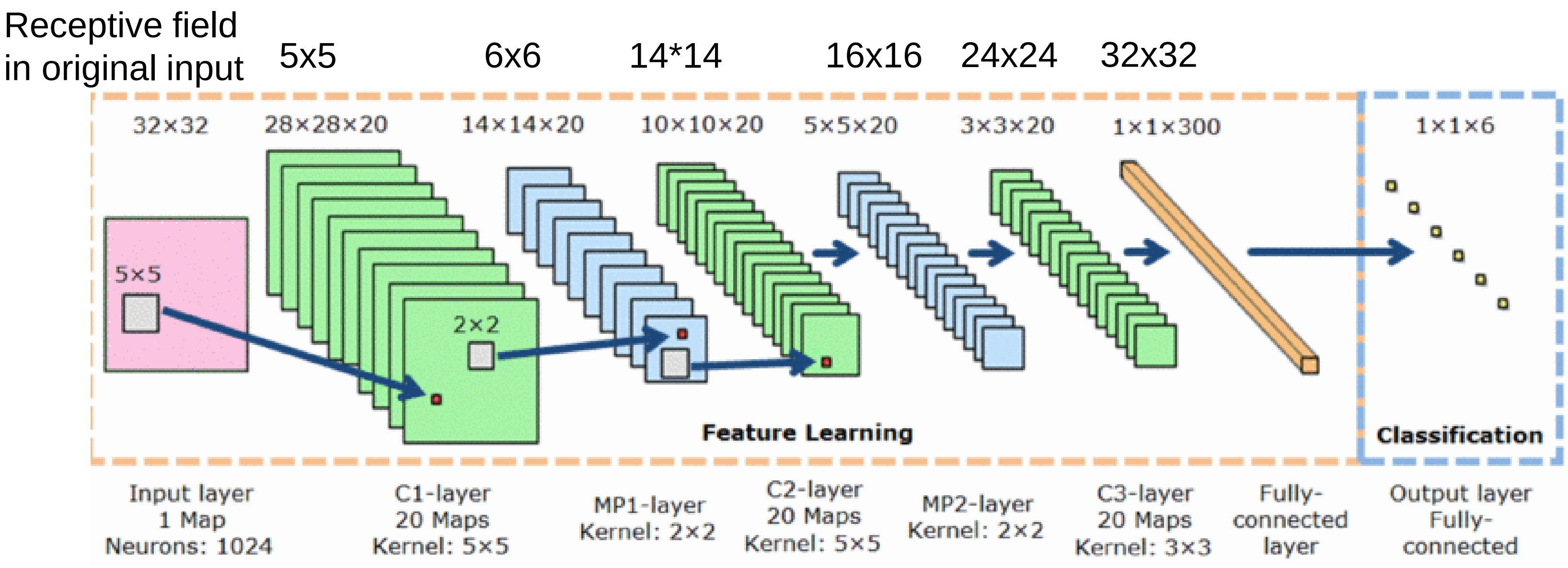
Looking in the feature extracting part



Let's focus on the NN shelf part that transforms the raw data into features
(which are used in the NN head to predict the parameters of the CPD)

The receptive field is growing from layer to layer

The receptive field of a neuron is the area in the original input image that impact the value of this neuron – “that can be seen by this neuron”.



Neurons from feature maps in higher layers have a larger receptive field than neurons sitting in feature maps closer to the input.

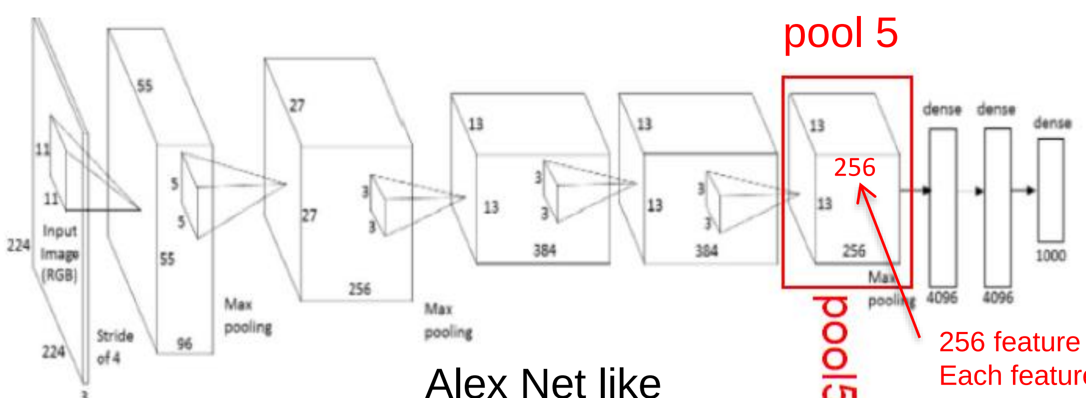
Code to determine size of receptive field: http://stackoverflow.com/questions/35582521/how-to-calculate-receptive-field-size

Figure credits: http://pubs.sciepub.com/ajme/2/7/9/figs

Visualize patches yielding high values in activation maps

Figure 4: Top regions for six pool$_5$ units. Receptive fields and activation values are drawn in white. Some units are aligned to concepts, such as people (row 1) or text (4). Other units capture texture and material properties, such as dot arrays (2) and specular reflections (6).

http://cs231n.github.io/understanding-cnn/



Alex Net like

Here we show image patches that activate maps in layer 5 most.

256 feature maps generated by 256 different 3x3 filters.
Each feature map consists of equivalent neurons looking at different positions of the input volume.

What kind of image (patches) excites a certain neuron corresponding to a large activation in a feature map?

10 images from data set leading to high signals 6 feature maps of **conv6**

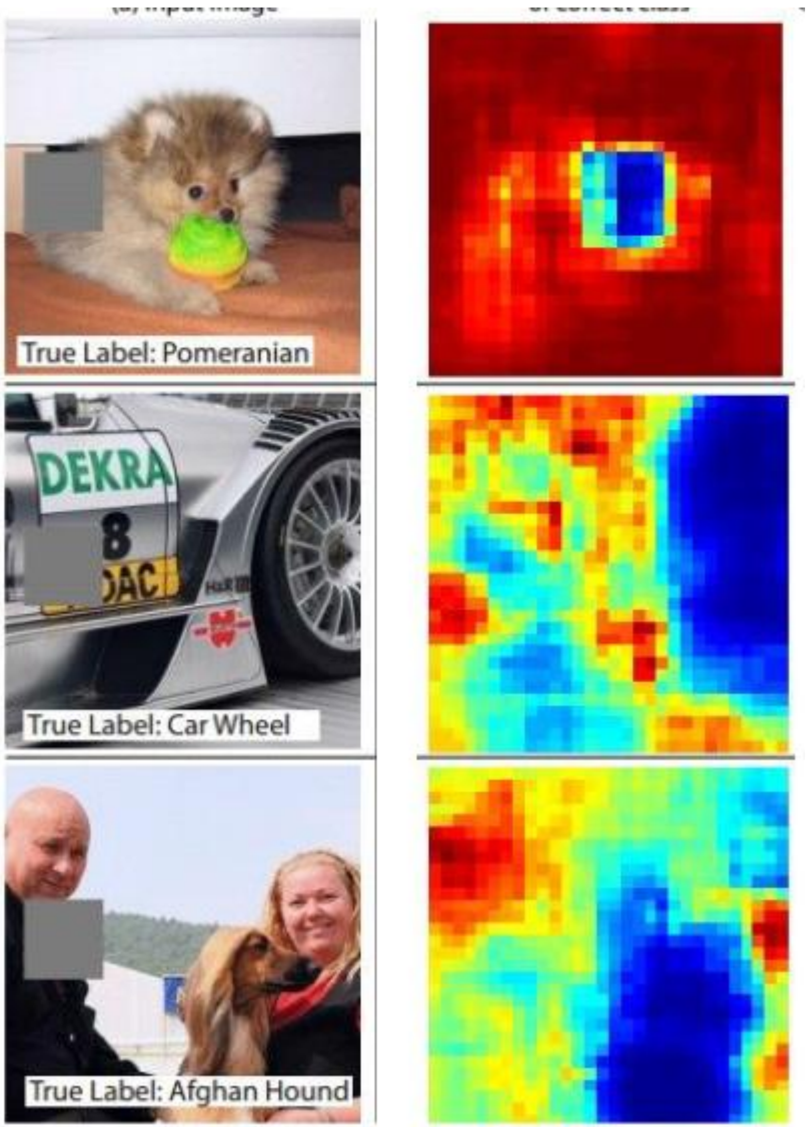
10 images from data set leading to high signals 6 feature maps of **conv9**

image credit: cs231n

Which pixels are important for the classification?
Occlusion experiments

Occlude part of the image with a mask and check for each position of the mask how strongly the score for the correct class is changing.

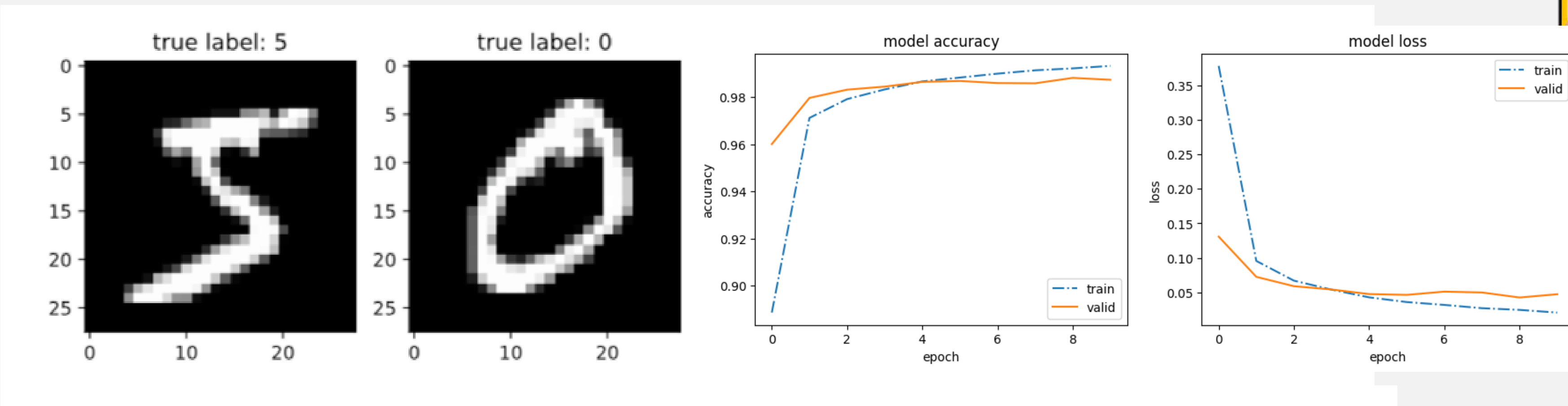
Warning:
Usefulness depends on application…

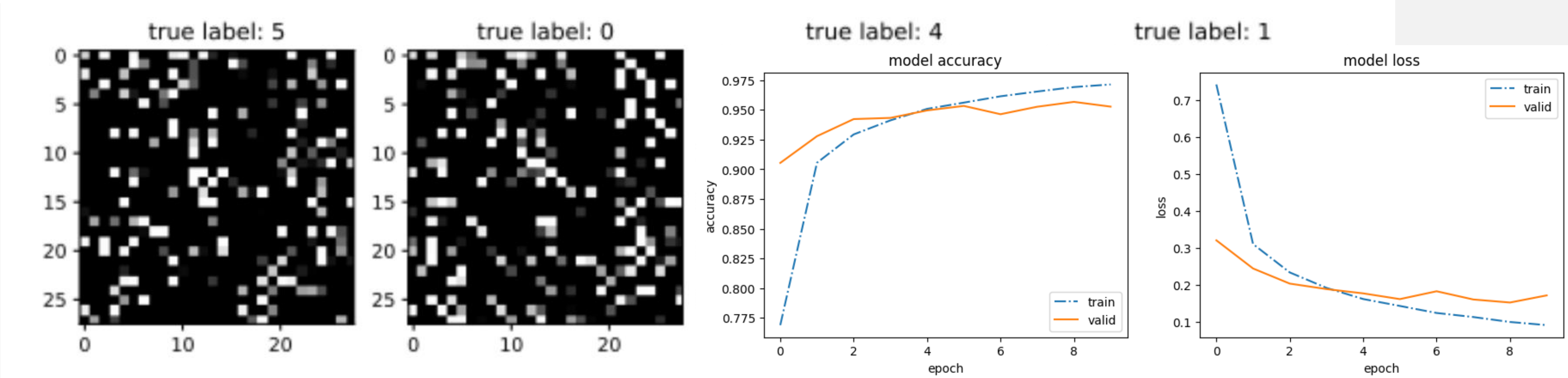


Occlusion experiments [Zeiler & Fergus 2013]

image credit: cs231n

Experiment: Does shuffling disturb a CNN?





→ The performance of a CNN is better on original than on shuffled images

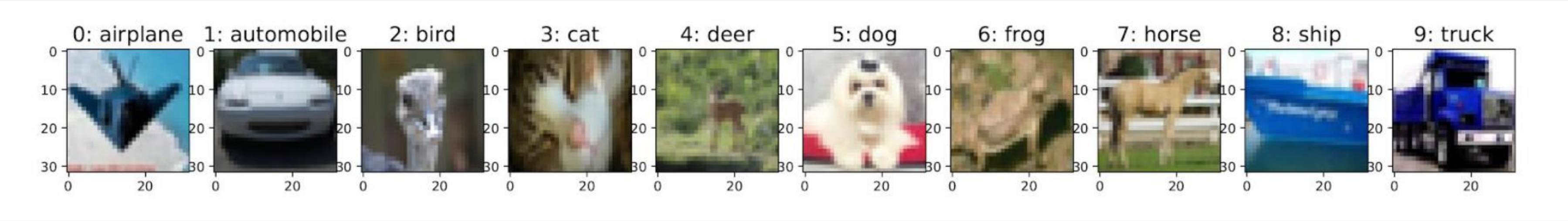
fcNN versus CNNs – some aspects

- A fcNN is good for tabular data, CNNs are good for ordered data (eg images)
- In a fcNN the order of the input does not matter, in CNN shuffling matters
- The CNN architecture imposes an inductive bias that neighborhood matters
- A node in one layer of a fcNN corresponds to one feature map in a convolution layer
- In each layer of a fcNN connecting p to q nodes, we learn q linear combinations of the incoming p signals, in each layer of a CNN connecting p channels with q channels we learn q filters (each having p channels) yielding q feature maps

Tricks of the Trade

- Data normalization
- Data augmentation
- Dropout
- Batch Norm (not covered)
- Skip connections (not covered)

Experiment: CNN for cifar10 data



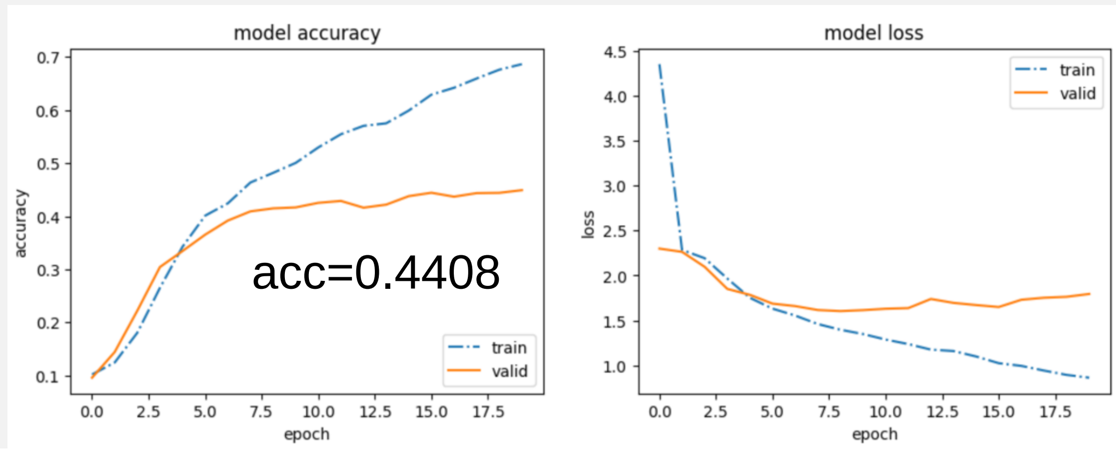
Develop a CNN to classify cifar10 images (we have 10 classes)

Investigate the impact of standardizing the data on the performance

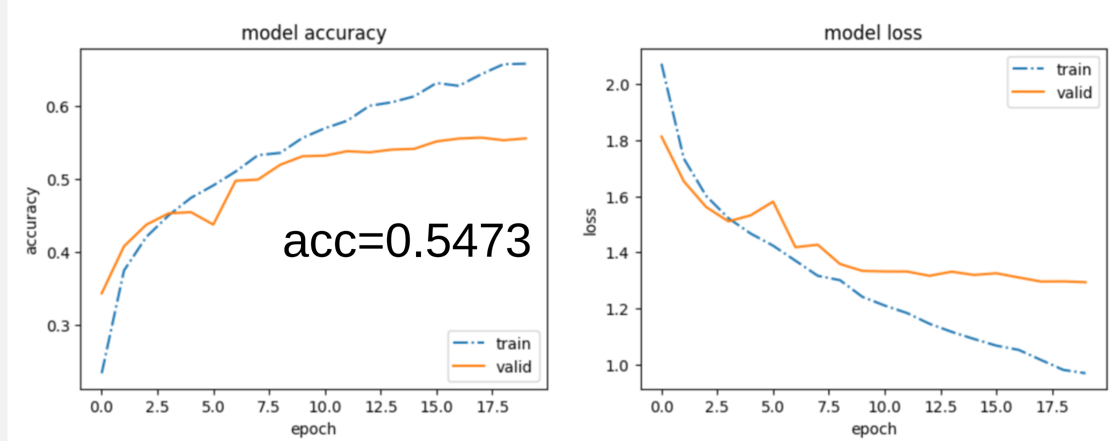
Take-home messages from the homework

- DL does not need a lot of preprocessing, but working with standardized (small-valued) input data often helps.

Without normalizing
the input to the CNN



After normalizing
(pixel-value/255)
the input to the CNN

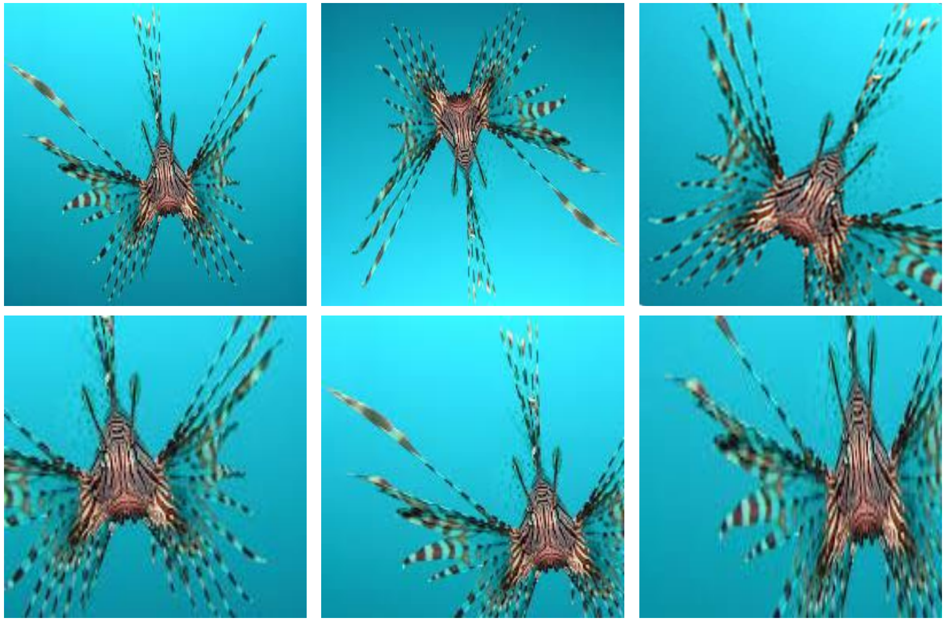


Data augmentation

We never have enough data!

Fighting overfitting by Data augmentation ("always" done): "generate more data" on the flight during fitting the model

- Rotate image within an angle range
- Flip image: left/right, up, down
- resize
- Take patches from images
- ….

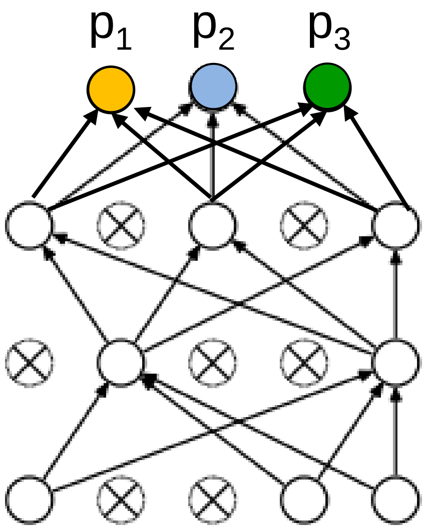


Data augmentation in Keras:

```
datagen <- image_data_generator(
  rotation_range = 20,
  width_shift_range = 0.2,
  height_shift_range = 0.2,
  horizontal_flip = TRUE
 )
```

Dropout during training

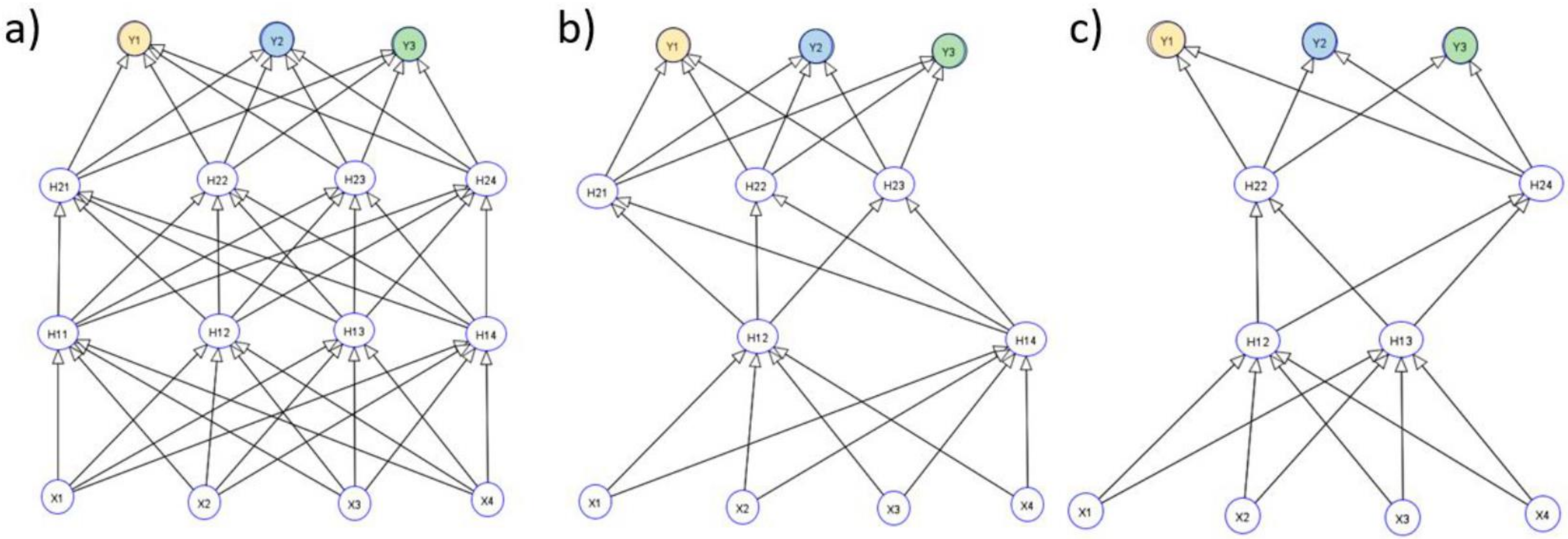
Dropout helps to fight overfitting



Using dropout during training implies:

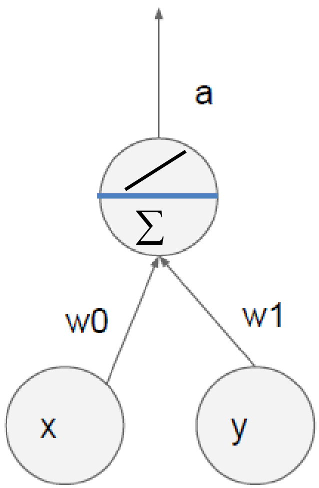
- In each training step only weights to not-dropped units are updated → we train a sparse sub-model NN
- For predictions with the trained NN we freeze the weights corresponding to averaging over the ensemble of trained models we should be able to "reduce noise", "overfitting"

Dropout



Three NNs: a) shows the full NN with all neurons, b) and c) show two versions of a thinned NN where some neurons are dropped. Dropping neurons is the same as setting all connections that start from these neurons to zero

Dropout-trained NN are kind of NN ensemble averages



x	y
0	0
0	y
x	0
x	y

Use the trained net without dropout during test time

Q: Suppose no dropout during test time (x, y are never dropped to zero), but a dropout probability p=0.5 during training

What is the expected value for the output **a** of this neuron?

during test w/o dropout: $\mathbf{a = w0*x + w1*y}$

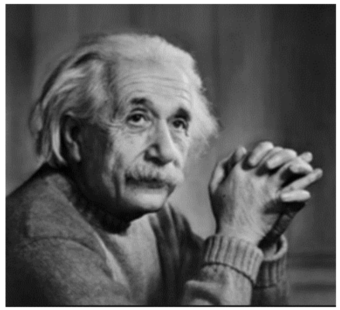
during training with dropout probability 0.5:

$$
\begin{aligned}
E[a] = \tfrac{1}{4} * (&w0*0 + w1*0 \;+ \\
&w0*0 + w1*y \;+ \\
&w0*x + w1*0 \;+ \\
&w0*x + w1*y) \\
= \tfrac{1}{4} * (&2\, w0*x + 2\, w1*y) \\
= \tfrac{1}{2} * &\mathbf{(w0*x + w1*y)}
\end{aligned}
$$

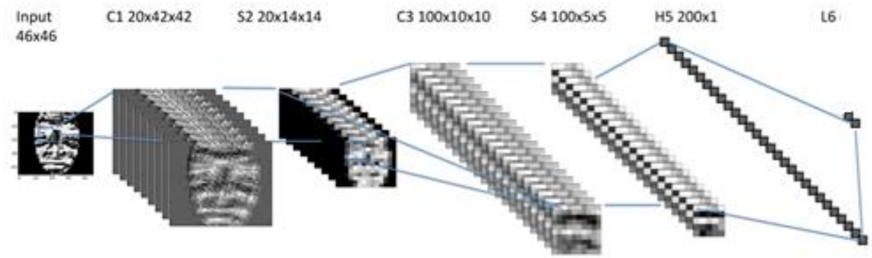
=> To get same expected output in training and test time, we reduce the weights during test time by multiplying them by the dropout probability p=0.5

Another intution: Why “dropout” can be a good idea

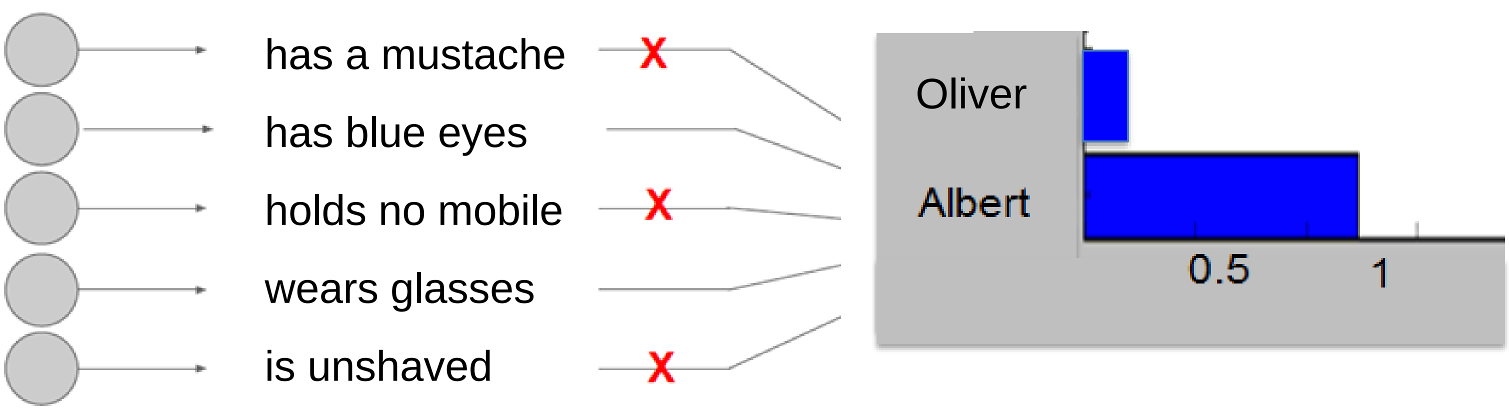
The training data consists of many different pictures of Oliver and Einstein



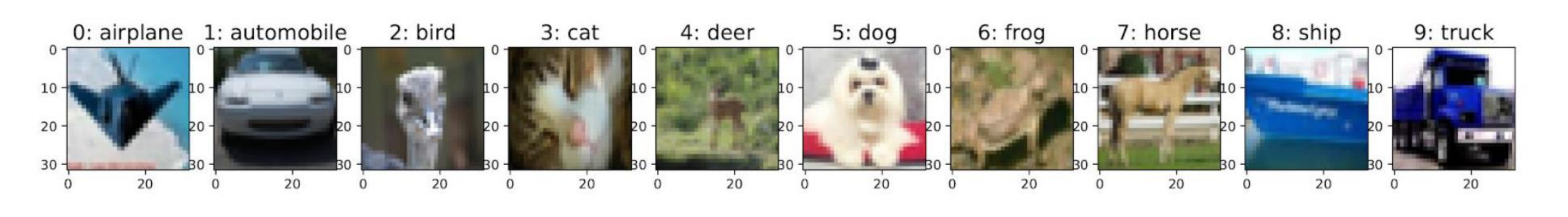
We need a huge number of neurons to extract good features which help to distinguish Oliver from Einstein



Dropout forces the network to learn redundant and independent features



Exercise: Develop a CNN for the CIFAR10 data set

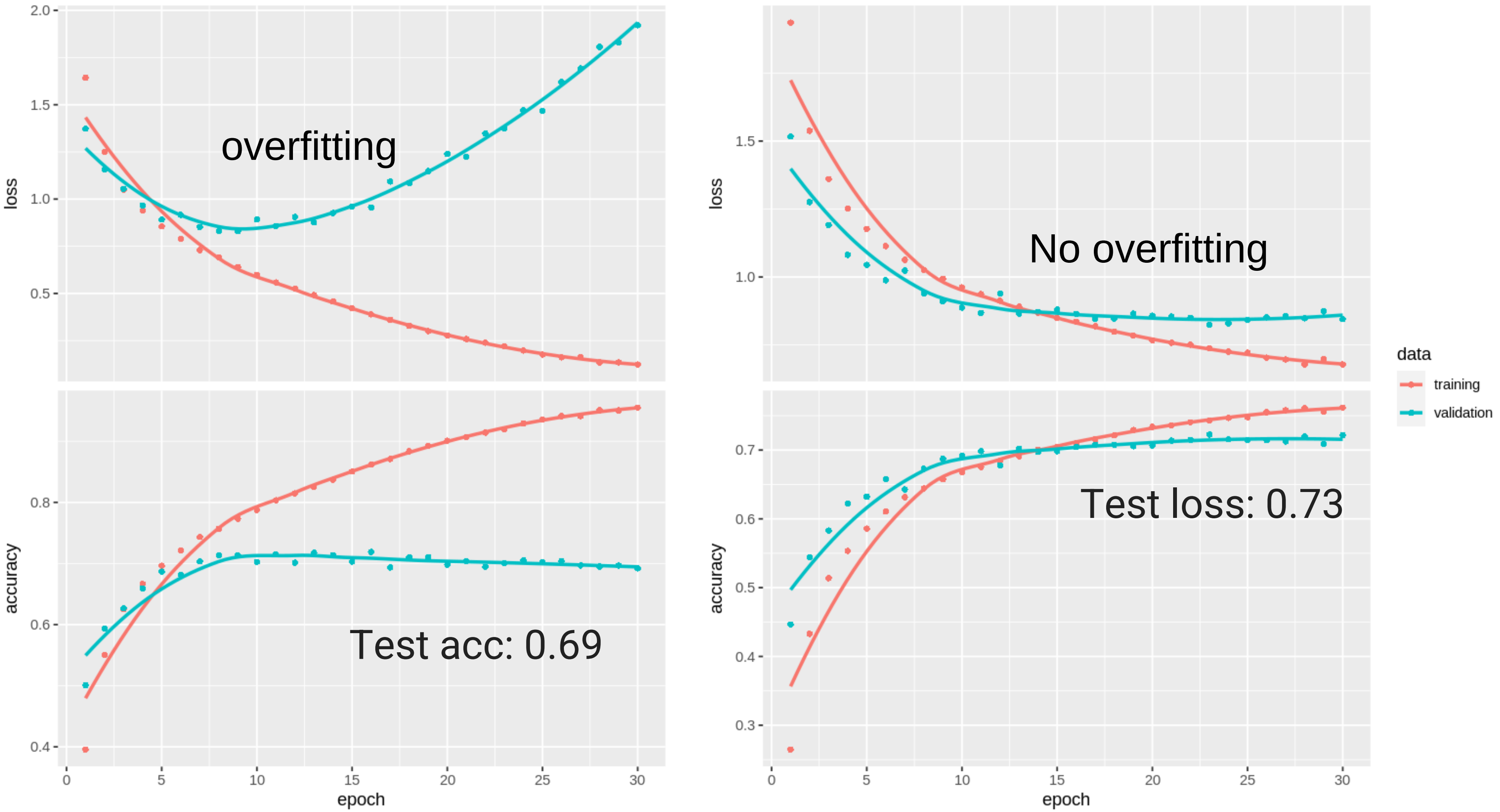


Train a CNN for CIFAR 10 data from scratch with and w/o dropout

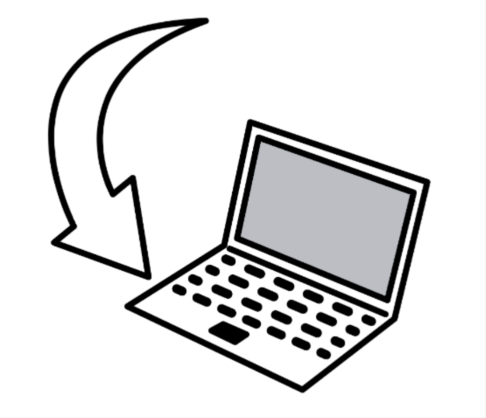
Does dropout during training help to prevent overfitting?

https://colab.research.google.com/github/tensorchiefs/dl_rcourse_2022/blob/main/notebooks/04_nb_ch02_03.ipynb

Dropout fights overfitting in a CIFAR10 CNN



Excercise



For the CNN lecture:

Play around with code, answer questions ask questions if you have any. See also https://tensorchiefs.github.io/dl_rcourse_2022/

1. Art lover example (03_nb_ch02_03.ipynb)
   https://colab.research.google.com/github/tensorchiefs/dl_rcourse_2022/blob/main/notebooks/03_nb_ch02_03.ipynb

2. Cifar10 example (04_nb_ch02_03.ipynb):
   https://colab.research.google.com/github/tensorchiefs/dl_rcourse_2022/blob/main/notebooks/04_nb_ch02_03.ipynb

Summary

- NNs work best when respecting the underlying structure of the data.
  - Use fully connected NN for tabular data
  - Use convolutional NN for data with local order such as images
- CNNs exploit the local structure of images by local connections and shared weight (same kernel is applied at each position of the image).