

Machine Intelligence:: Deep Learning

Beate Sick, Lilach Goren, Pascal Bühler

Institut für Datenanalyse und Prozessdesign
Zürcher Hochschule für Angewandte Wissenschaften

Outline of the DL Module (tentative)

The course is split in 8 sessions, each 4 lectures long. Topics might be adapted during the course

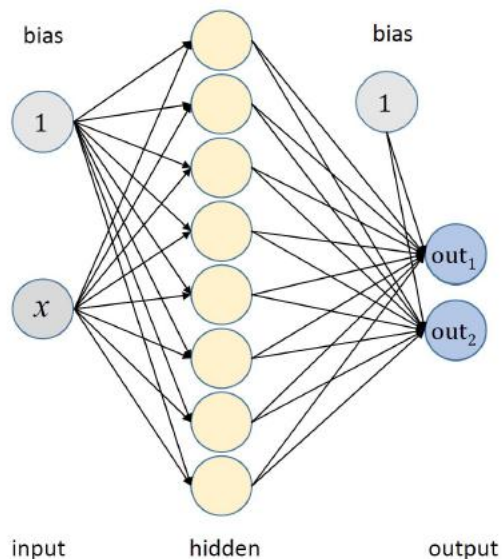
Day	Date	Time	Topic
1	15.04.2025	09:00-12:30	Introduction to Deep Learning & Keras, first NNs
-	21.04.2025	-	FRÜHLINGS-FERIEN
-	28.04.2025	-	FRÜHLINGS-FERIEN
2	06.05.2025	09:00-12:30	Loss, Optimization, Regression, Classification
3	13.05.2025	09:00-12:30	Computer vision, CNN-architecture
4	20.05.2025	09:00-12:30	DL in practice, pretrained (foundation) models
5	27.05.2025	09:00-12:30	Model evaluation, baselines, xAI, troubleshooting
6	03.06.2025	09:00-12:30	Generative Models, Transformer-architecture
7	10.06.2025	09:00-12:30	Vision Transformer
8	17.06.2025	09:00-12:30	Projects, deep Ensembling

Look back on homework: define and use a custom loss function



Notebook: [02_custom_loss.ipynb](#)

https://github.com/tensorchiefs/dl_course_2025/blob/main/notebooks/02_custom_loss.ipynb



$$(Y|x) \sim N(\mu(x), \sigma(x))$$

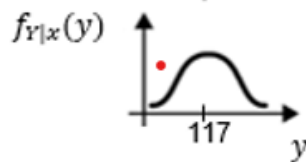
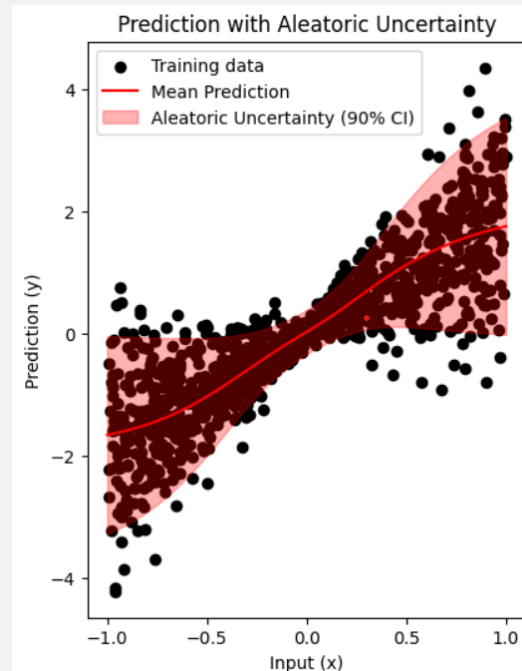
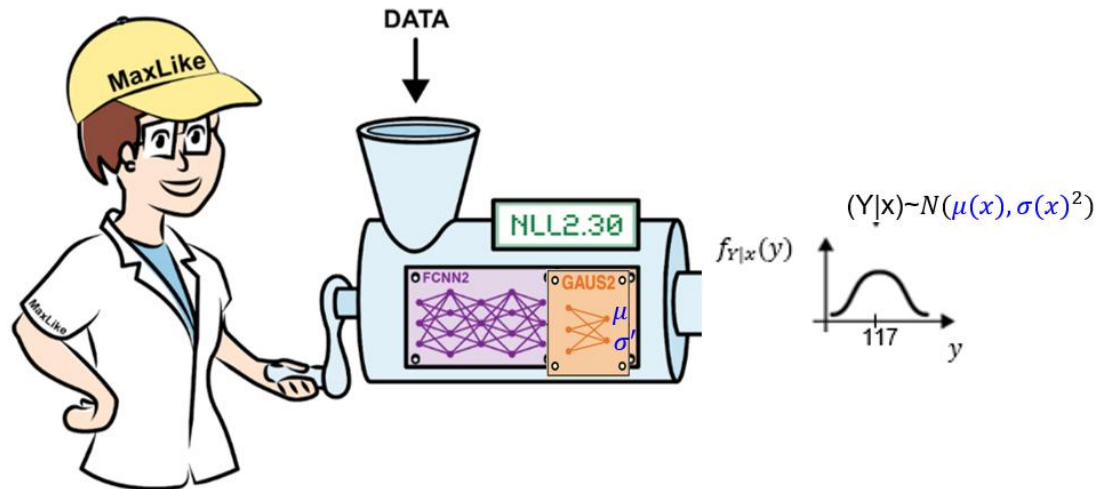
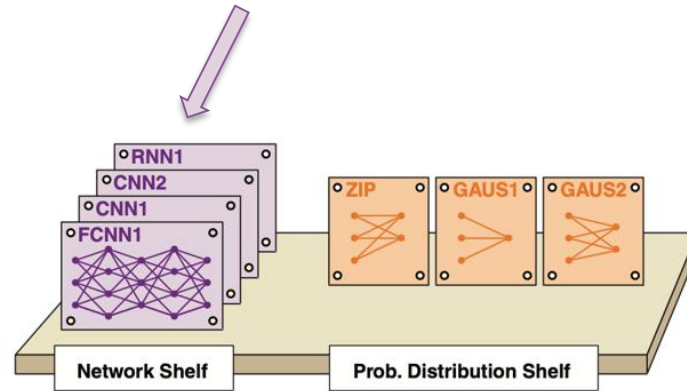


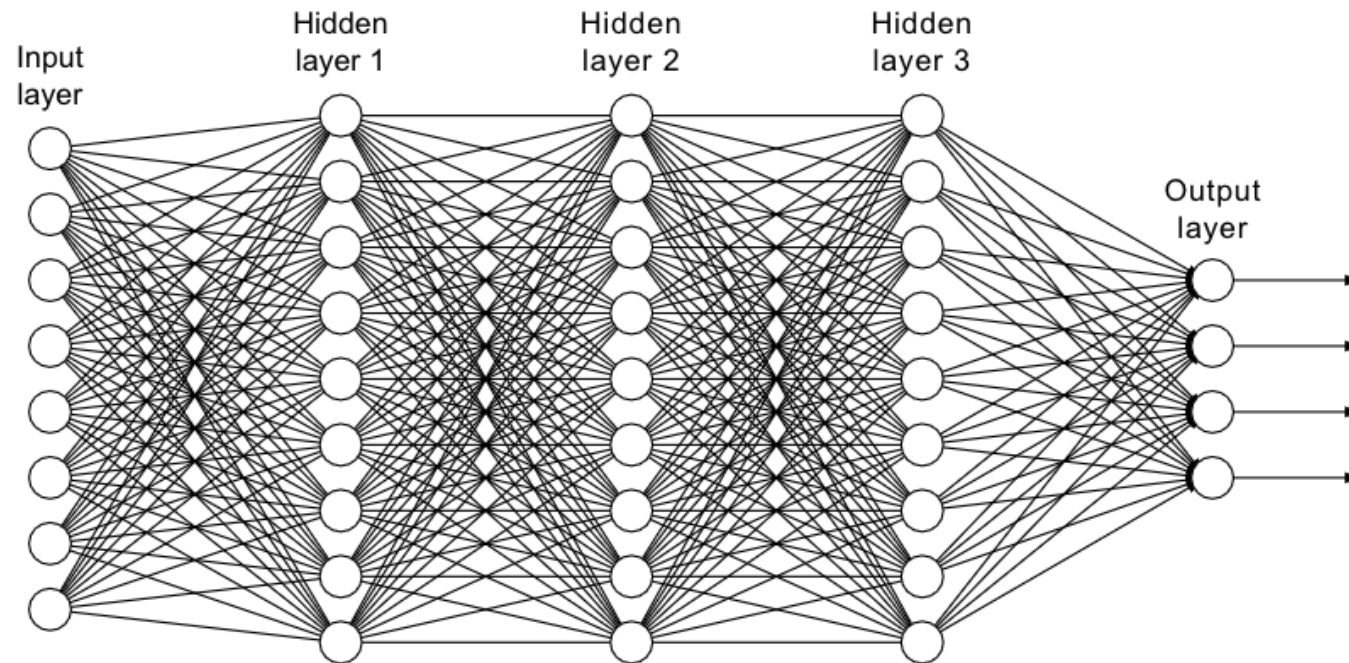
Figure 4.19 A NN with two output nodes can be used to control the parameter μ_x and σ_x of the conditional outcome distribution $N(\mu_x, \sigma_x)$ for regression tasks with non-constant variance



NN architectures

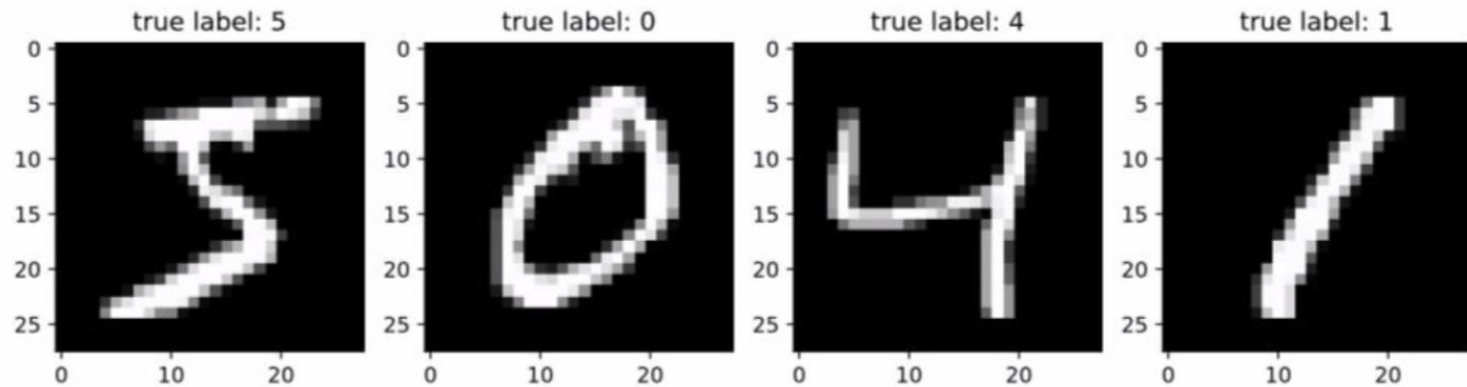


A fully connected NN

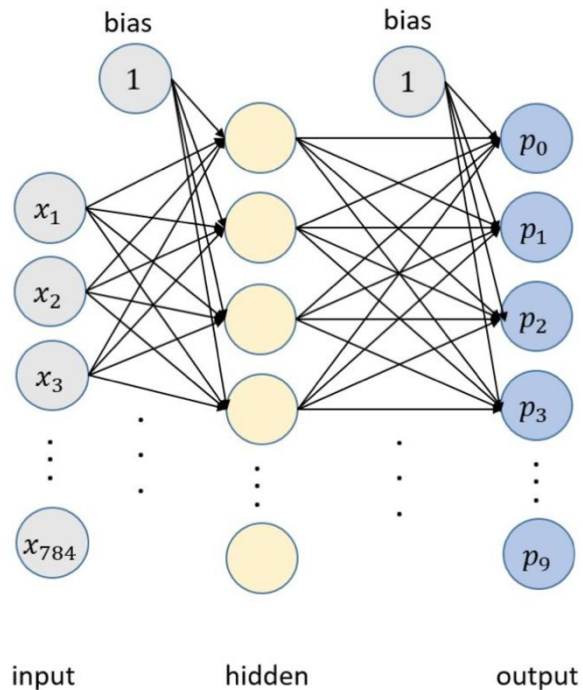


Probabilistic classification

- Usually in DL the model predicts a probability for each possible class
- Example:
 - Banknote from exercise – classes: “real” or “fake”
 - Typical example Number from hand-written digit – classes: 0, 1, ..., 9



Classification: Softmax Activation



$p_0, p_1 \dots p_9$ are probabilities for the classes 0 to 9.

Activation of last layer z_i incoming

$$p_i = \frac{e^{z_i}}{\sum_{j=0}^9 e^{z_j}}$$

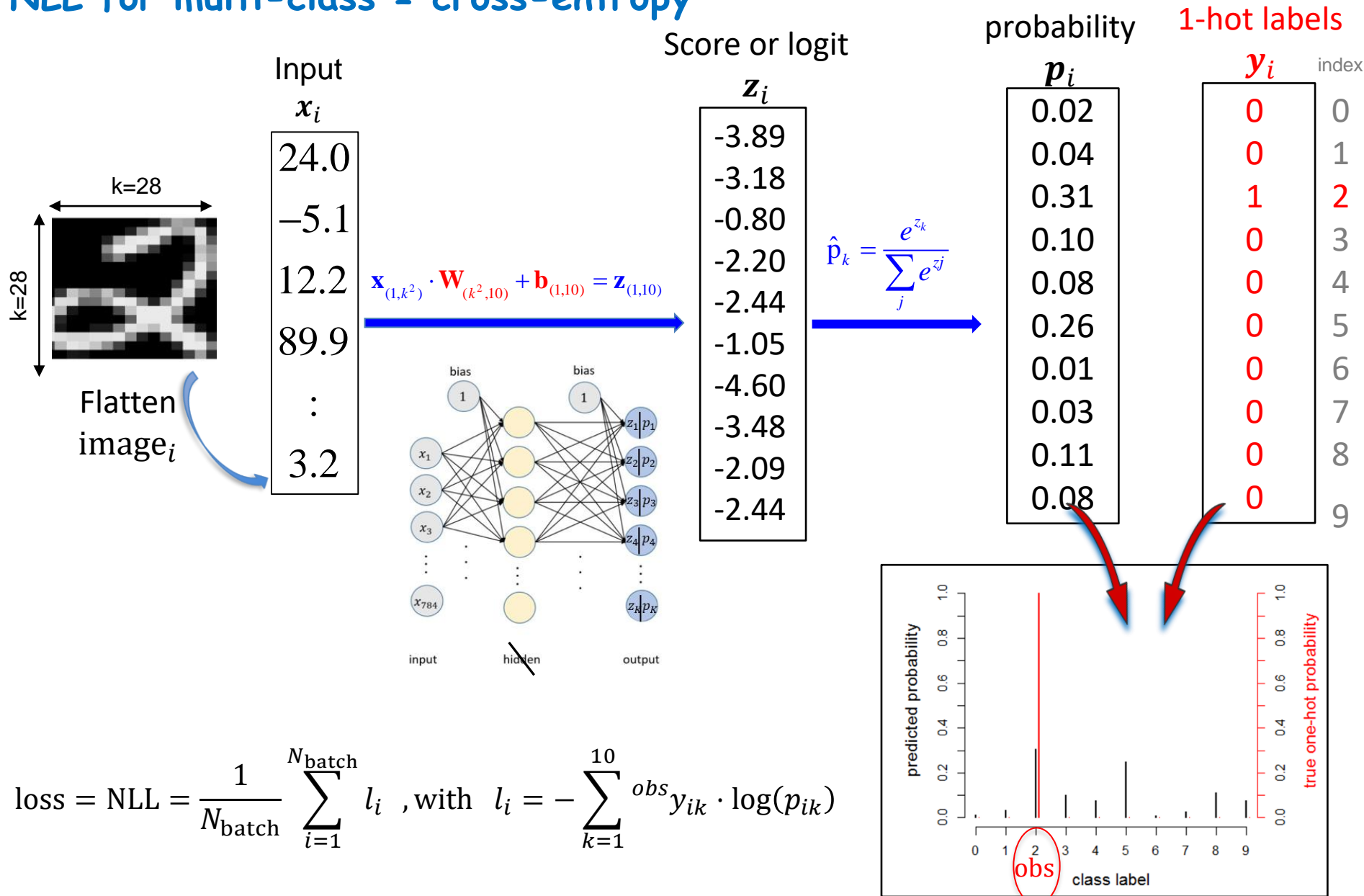
Makes
outcome
positive

Ensures that p_i 's sum up
to one

This activation is called softmax

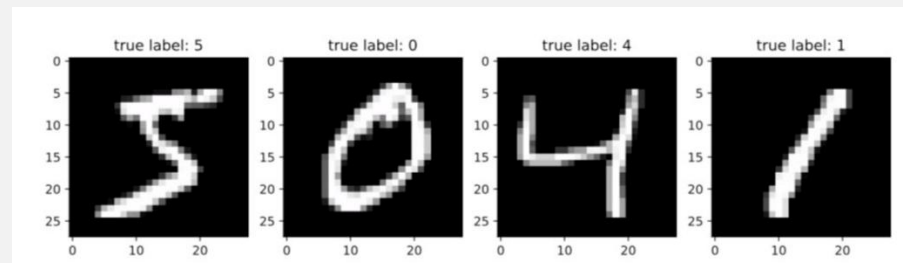
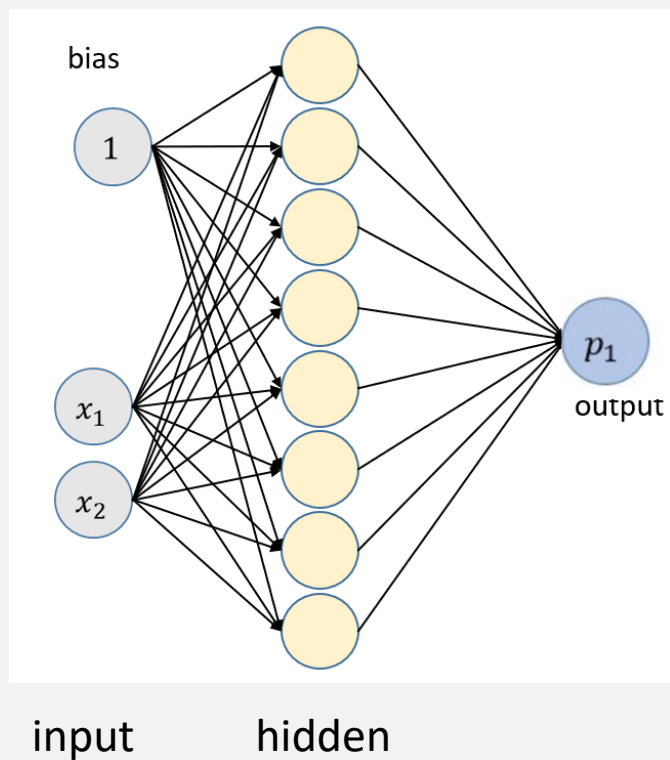
Figure 2.12: A fully connected NN with 2 hidden layers. For the MNIST example, the input layer has 784 values for the 28 x 28 pixels and the output layer out of 10 nodes for the 10 classes.

NLL for multi-class = cross-entropy



Loss = NLL= cross-entropy $(-\sum p_i^{\text{obs}} \log p_i^{\text{pred}})$ averaged over all images in mini-batch

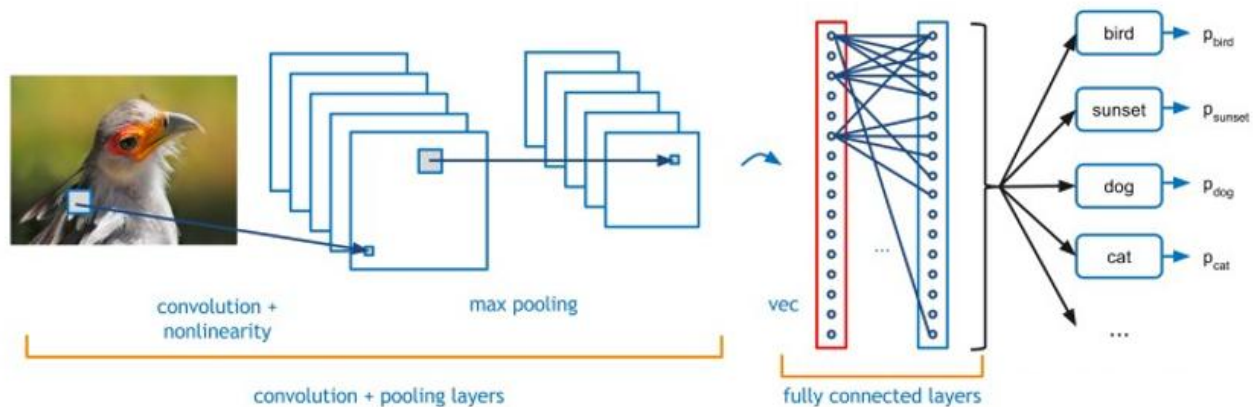
Exercise:



Use a fully connected NN to classify MNIST data to 10 possible classes

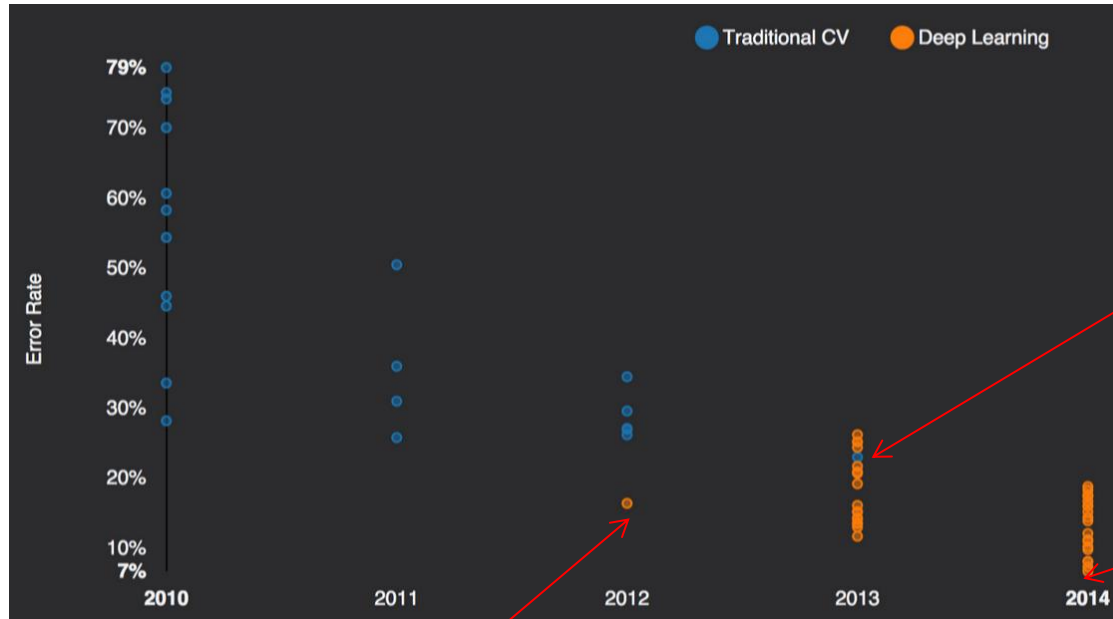
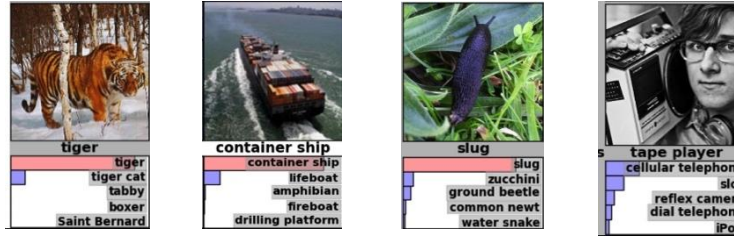
[03_fcnn_mnist_keras_torch.ipynb](#)

Convolutional Neural Networks for image data



The first DL breackthrough: Imagenet challenge

1000 classes
1 Mio samples



Human: 5% misclassification

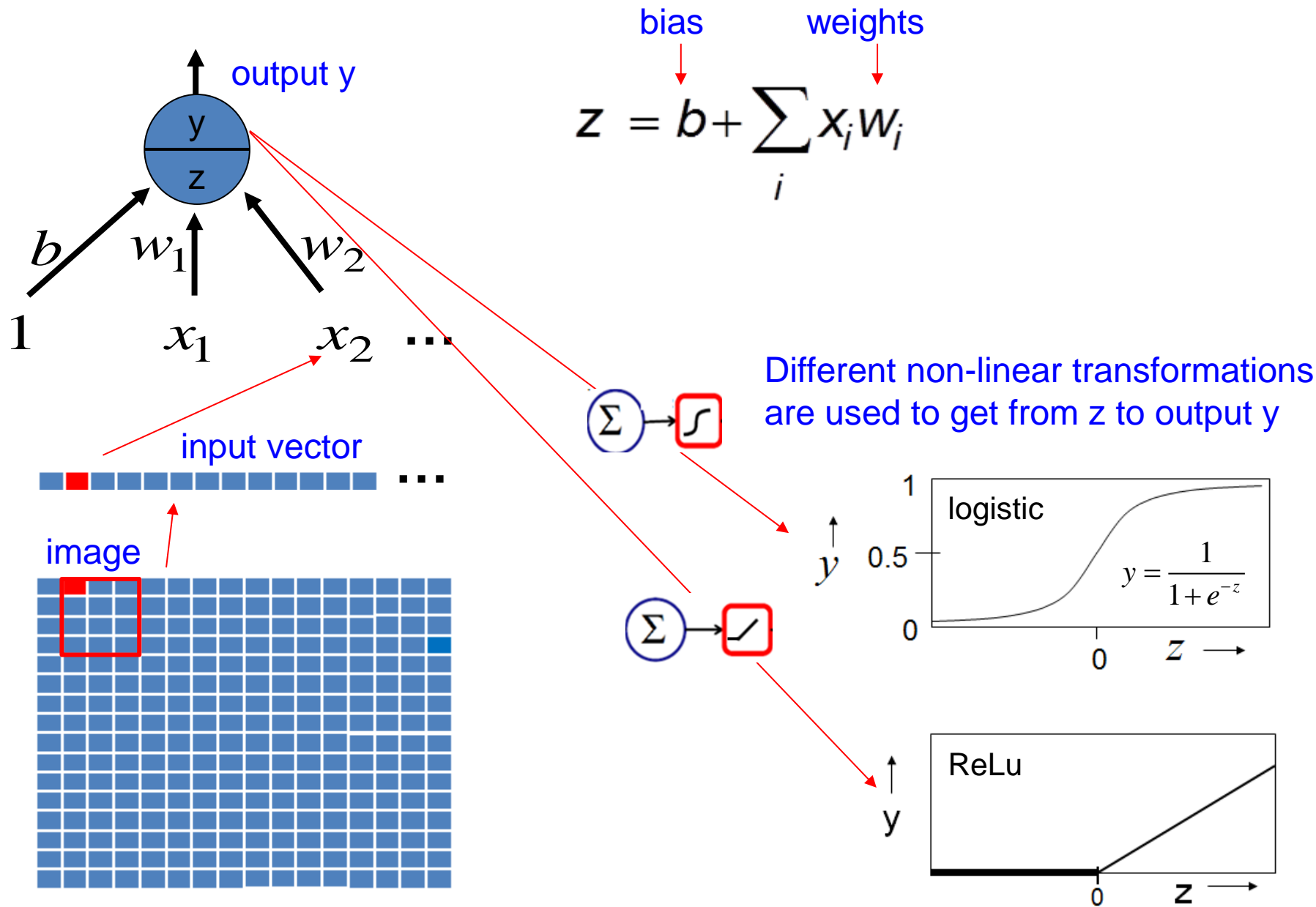
Only one non-CNN
approach in 2013

GoogLeNet 6.7%

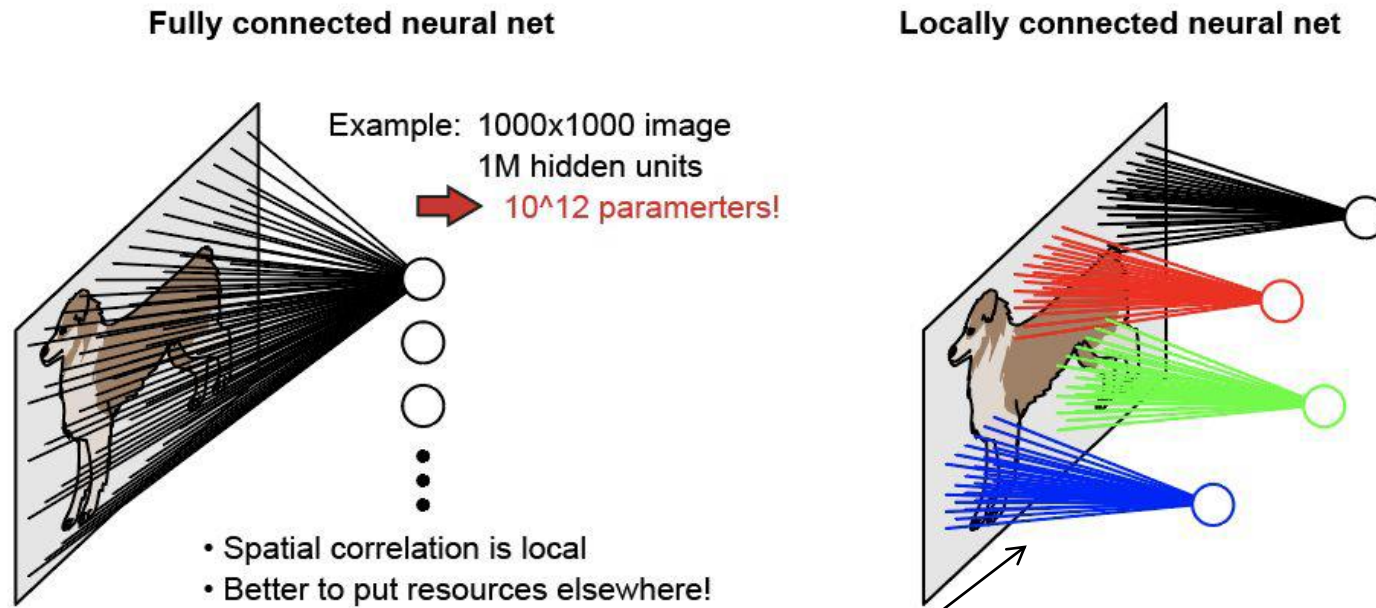
A. Krizhevsky
first CNN in 2012

Und es hat zoom gemacht

An artificial neuron



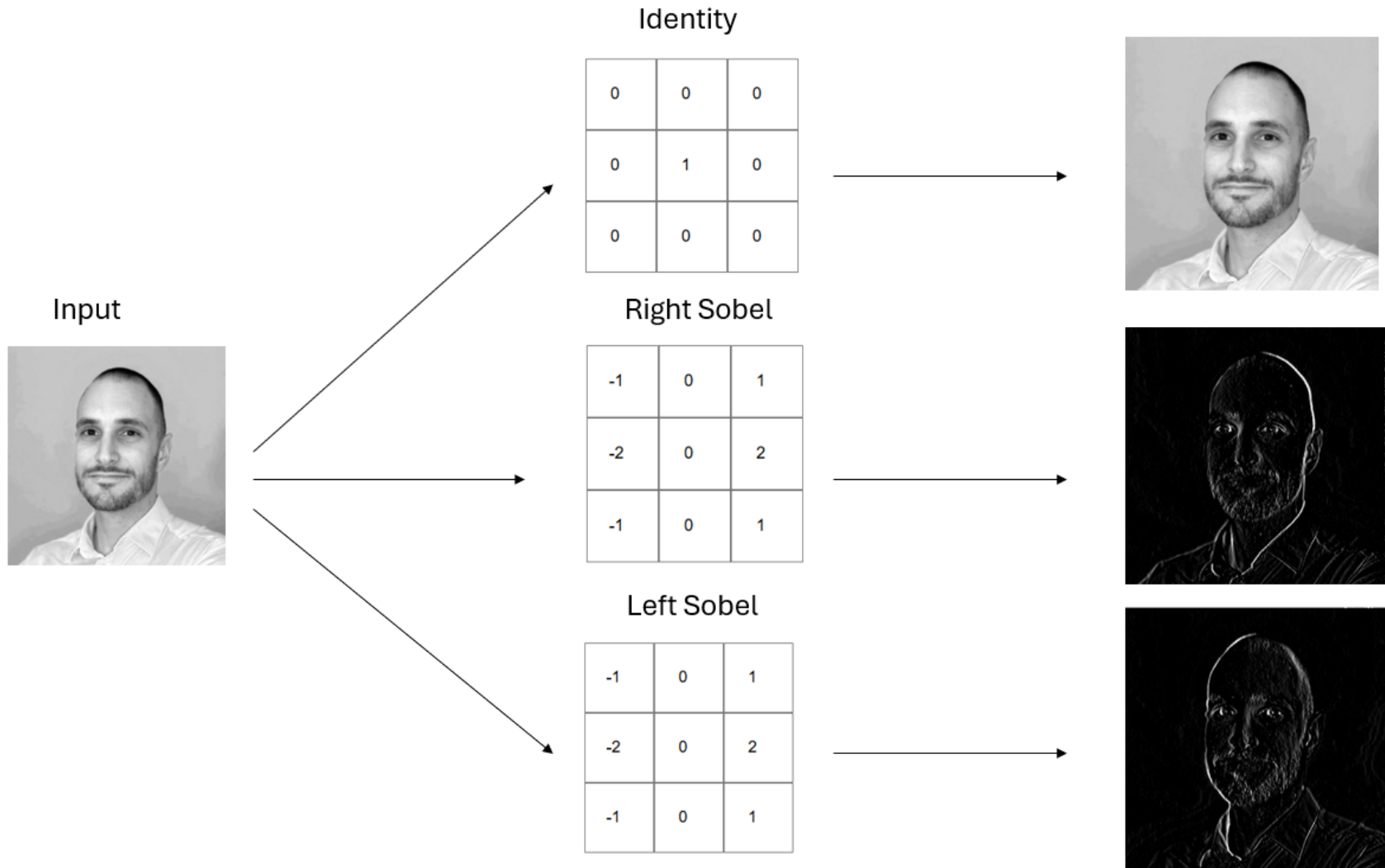
Convolution extracts local information using few weights



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



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

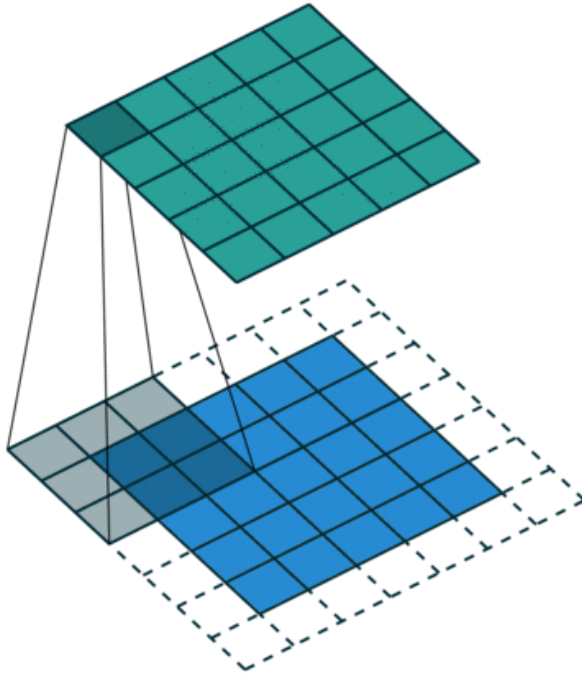
Convolution (let's ignore bias b):

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

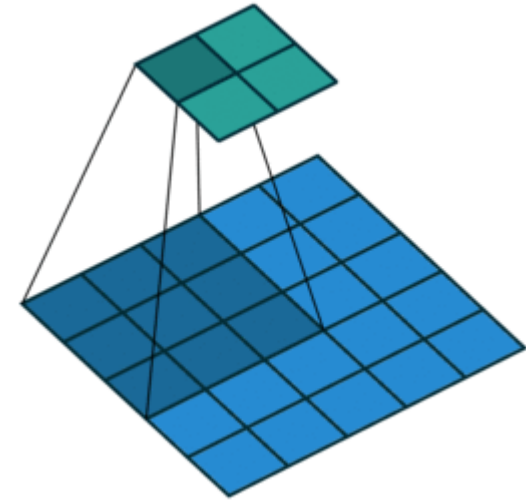
1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

4		

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.

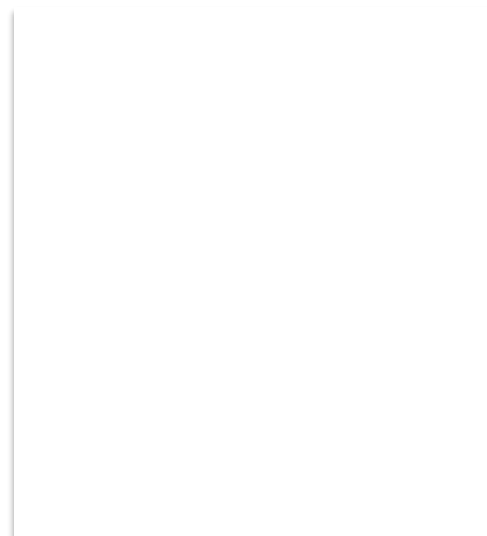
Exercise: Do one convolution step by hand



The kernel is 3x3 and is applied at each valid position
– 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

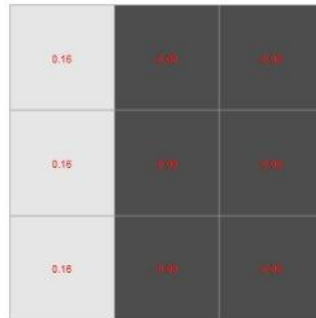
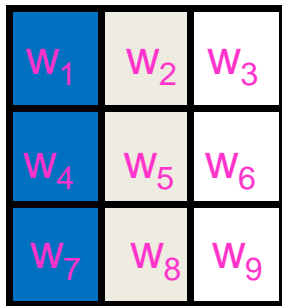


Convolutional networks use neighborhood information and replicated local feature extraction

In a locally connected network the calculation rule

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

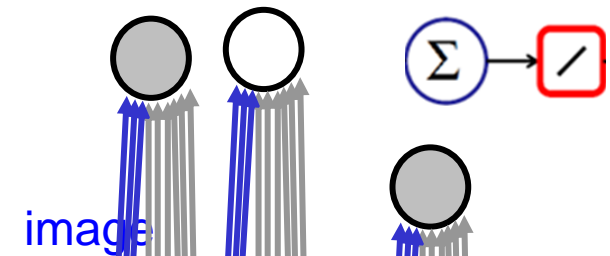
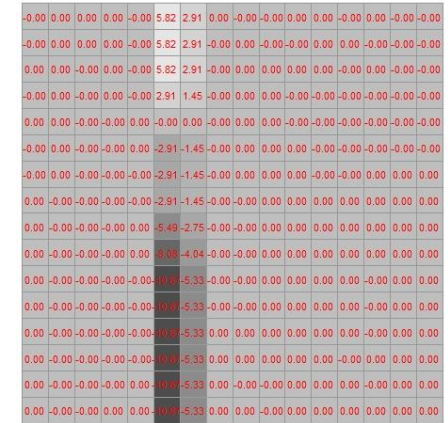
Pixel values in a small image patch are element-wise multiplied with weights of a small filter/kernel:



The filter is applied at each position of the image 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.

feature/activation map

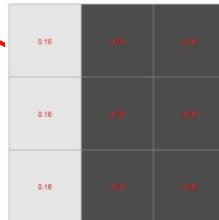
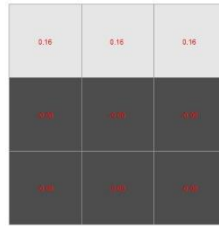
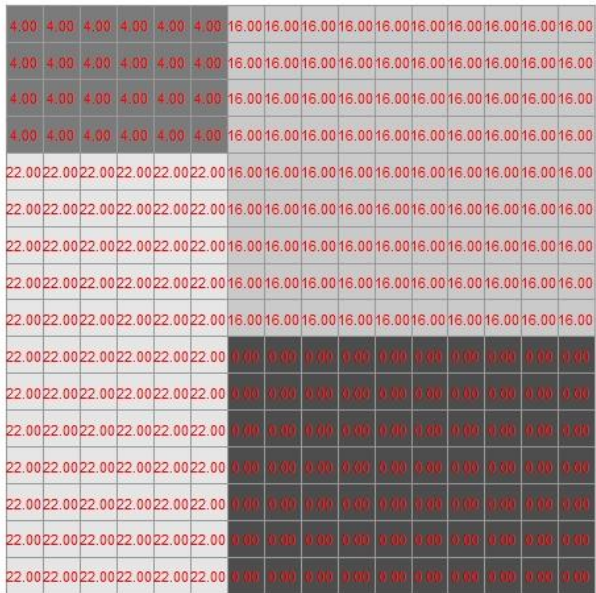


Convolutional networks use neighborhood information and replicated local feature extraction

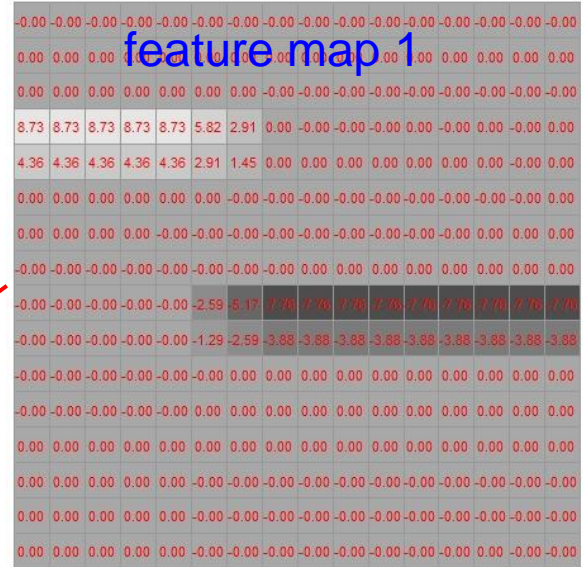
filtering =
convolution

kernel 1

image



feature map 1

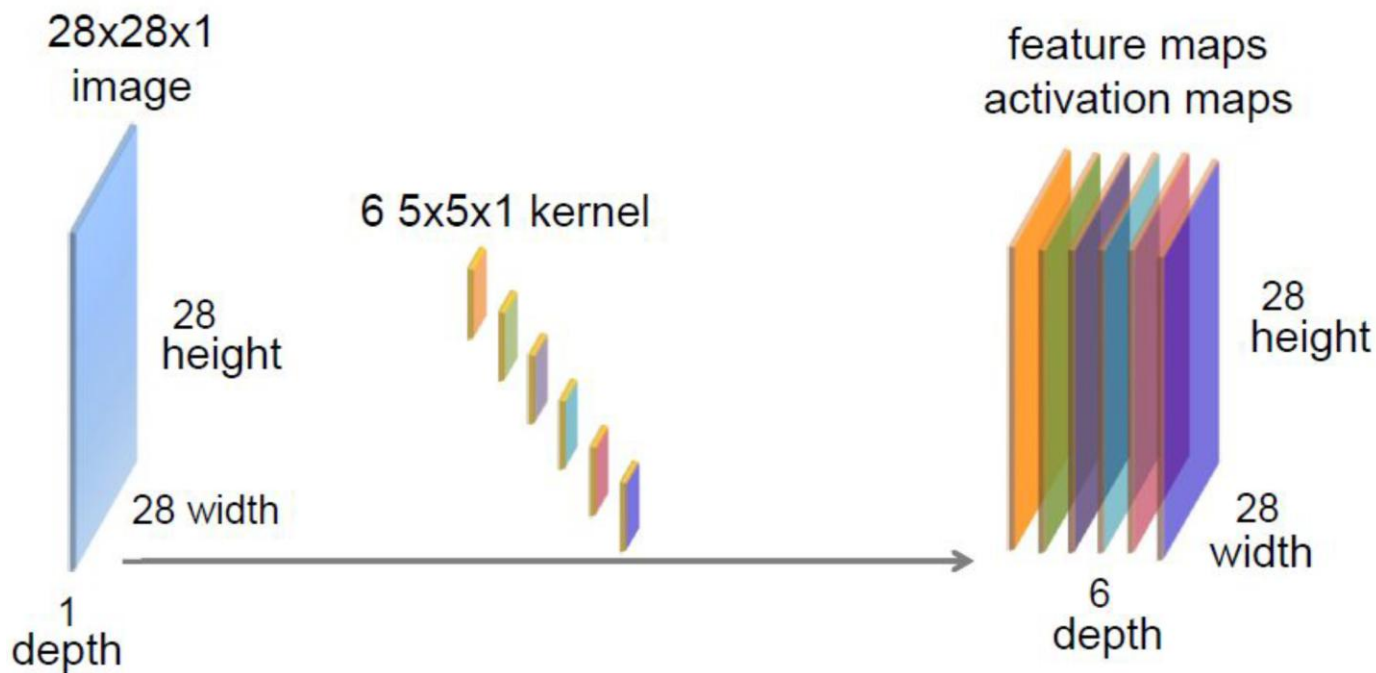


feature map 2



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

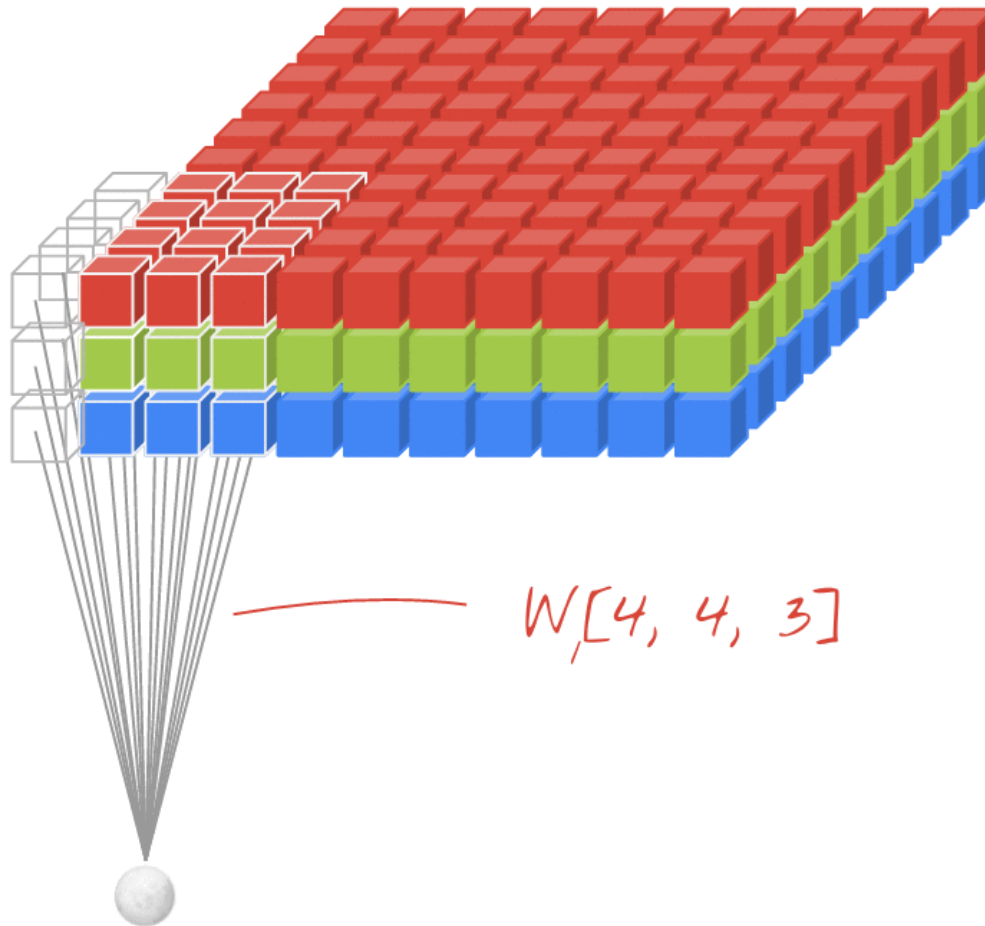
Convolution layer with a 1-channel input and 6 kernels



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

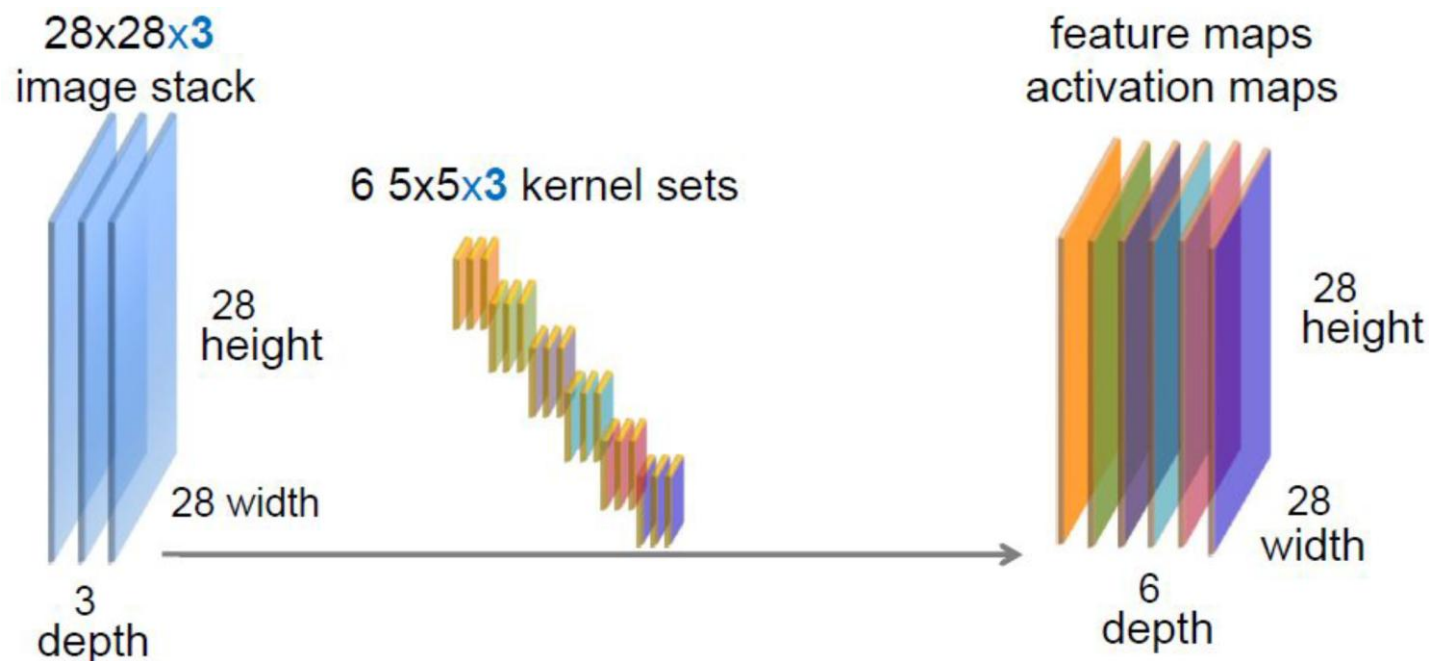
3 color channel input image



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

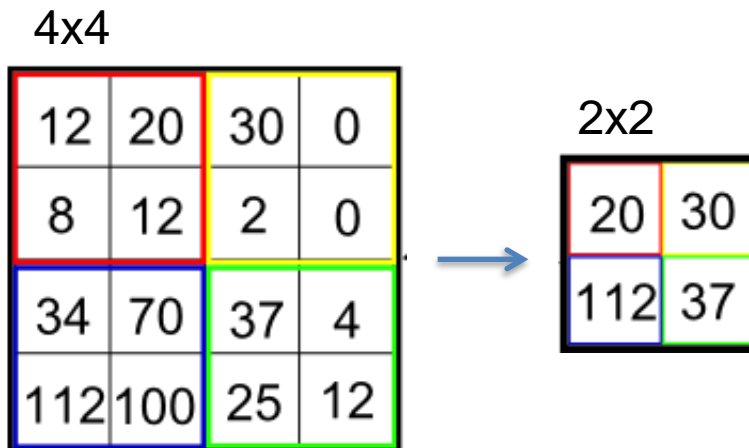
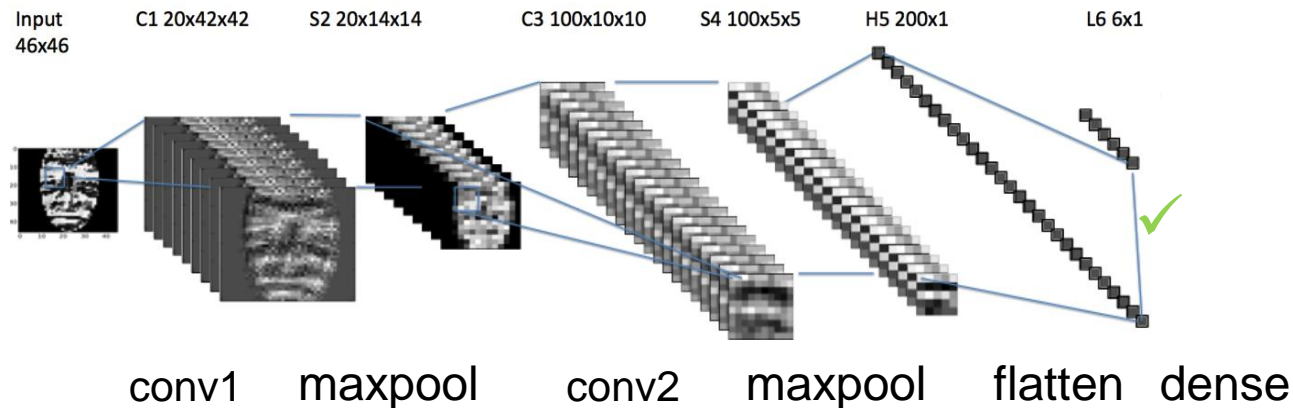
Animation credits: M.Gorner, <https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#10>
For an example with number see convolution demo in: <https://cs231n.github.io/convolutional-networks/>

Convolution layer with a 3-channel 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.

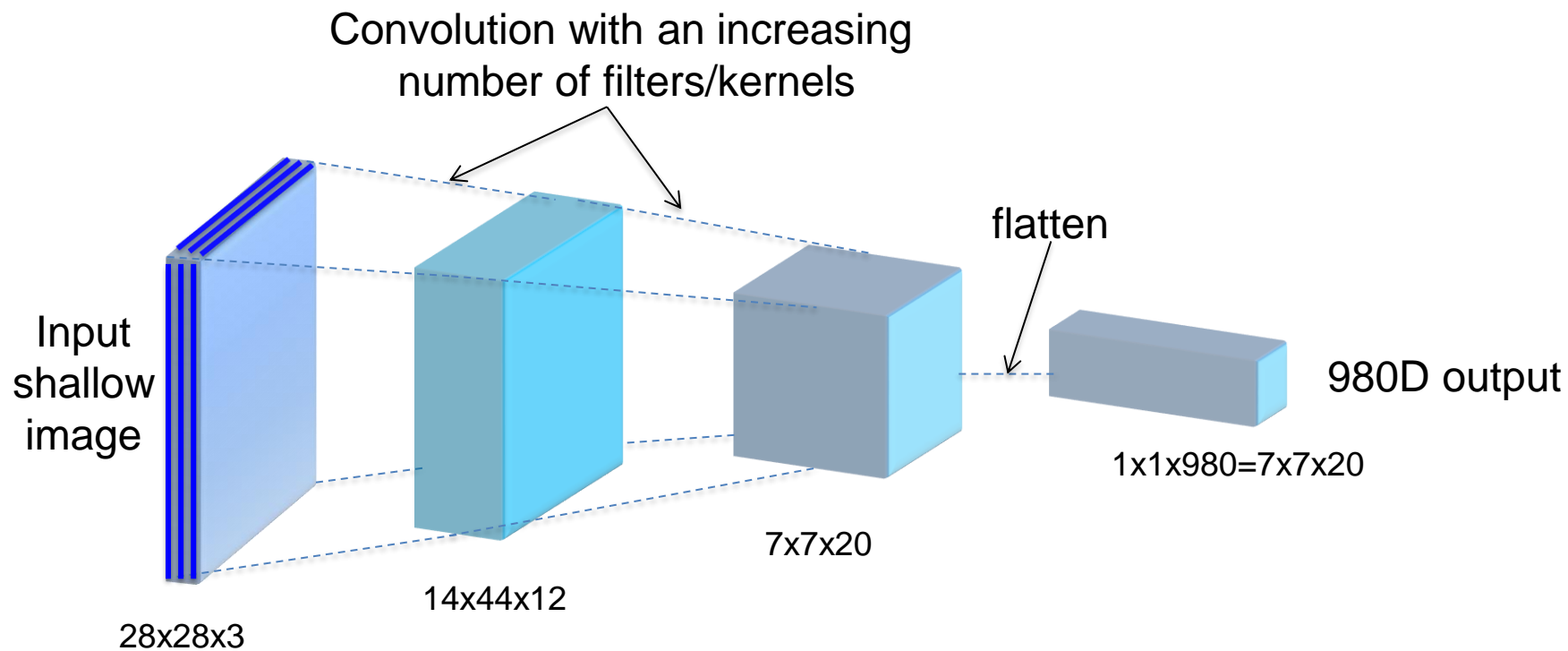
CNN ingredient II: Maxpooling Building Blocks reduce size



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

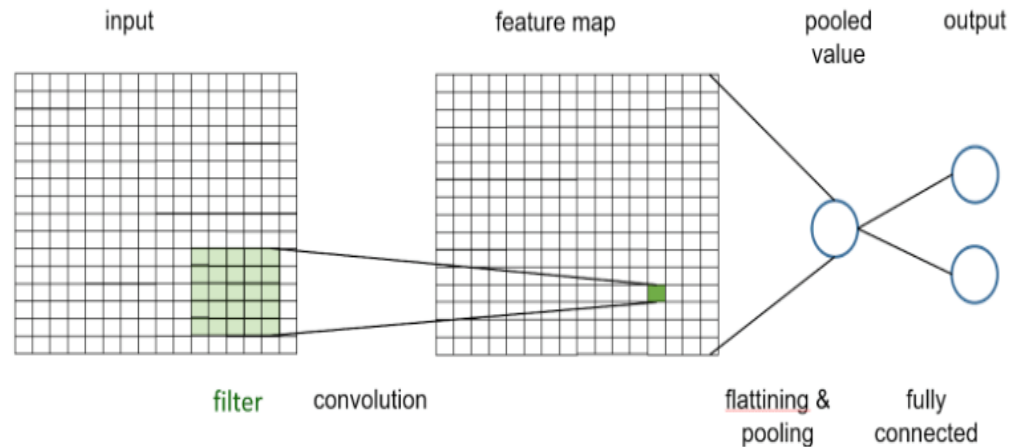
Hinton: „The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster“

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.

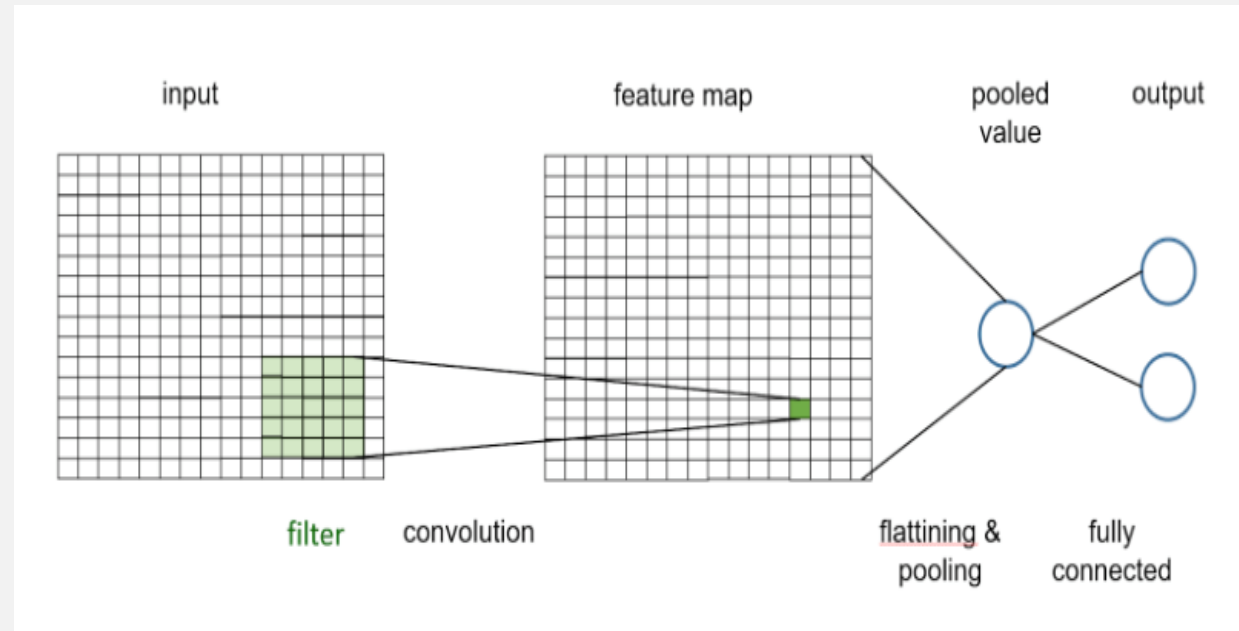
Building a very simple CNN with keras



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

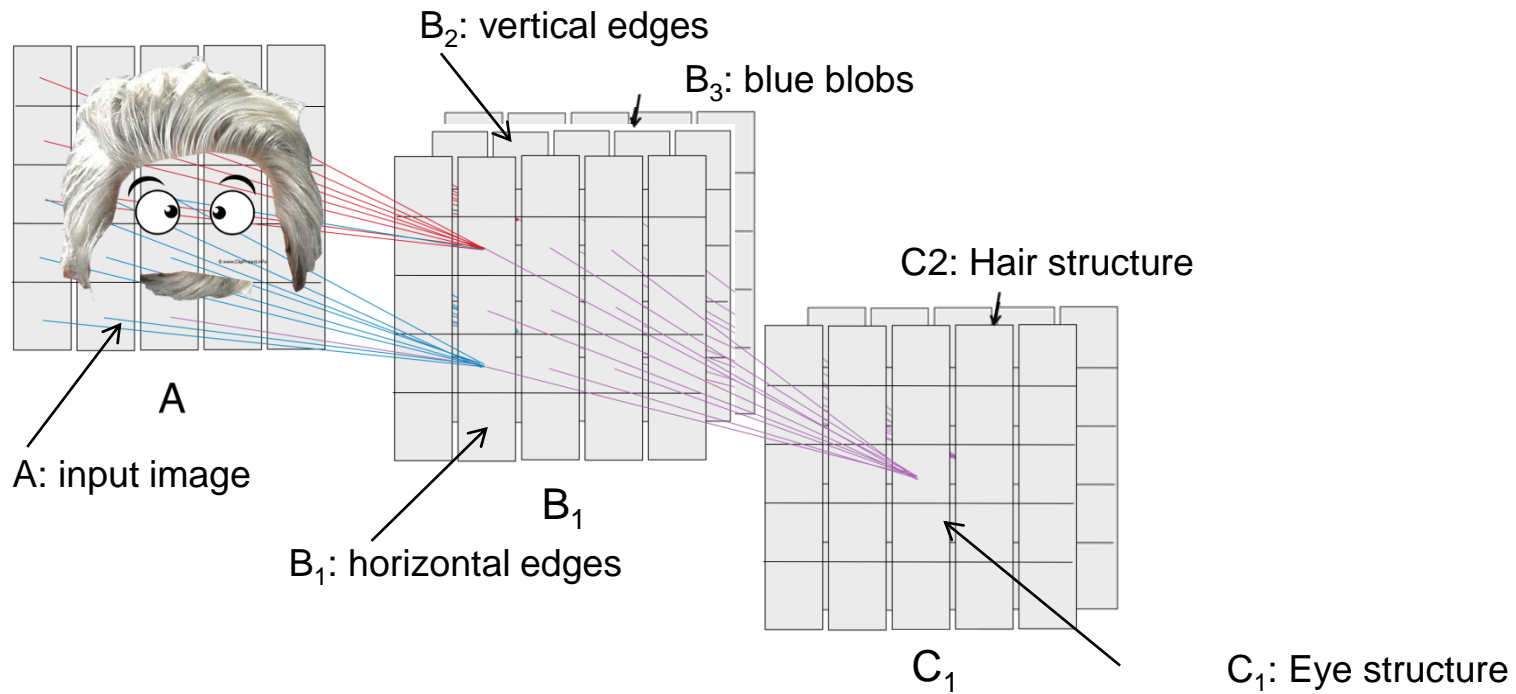
Exercise: Artstyle Lover

Train a CNN with only one filter in one conv-layer, that can predict, if the presented image shows vertical or horizontal lines.

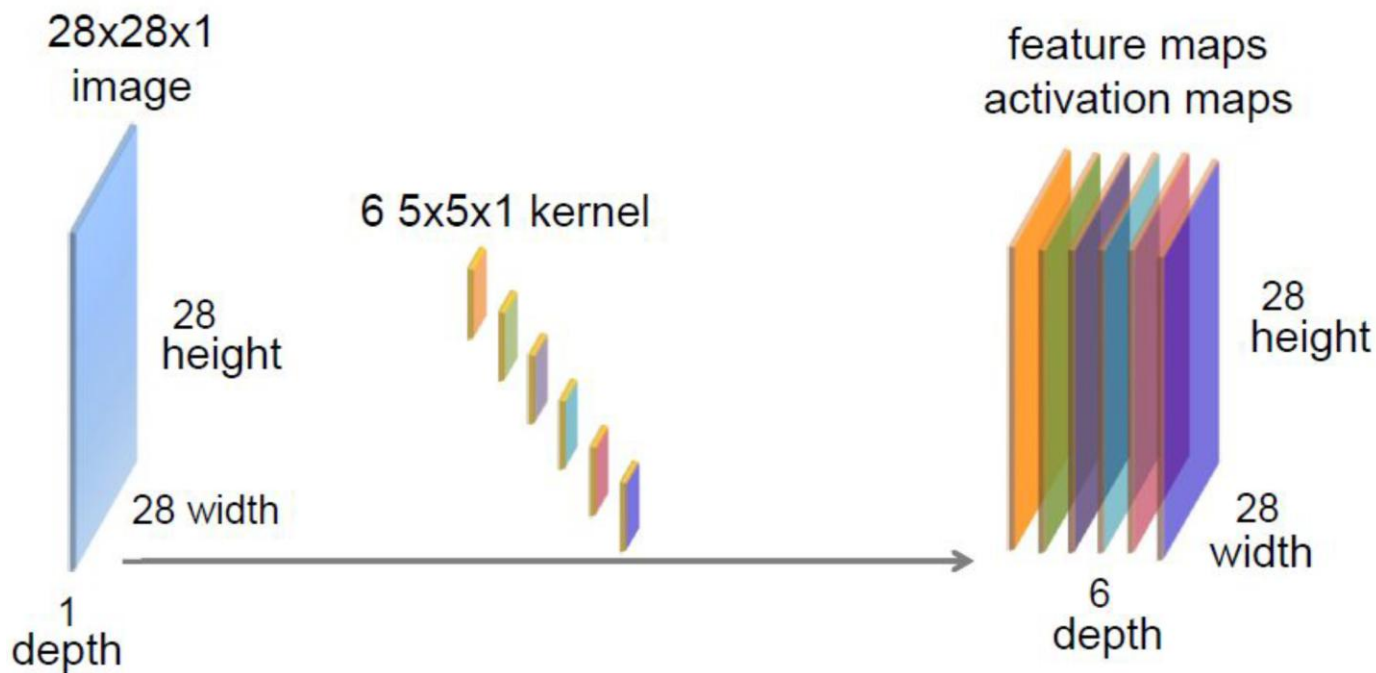


Open NB in: https://github.com/tensorchiefs/dl_course_2025/blob/master/notebooks/05_cnn_edge_lover_keras_torch.

More than one kernel (Motivatoion)

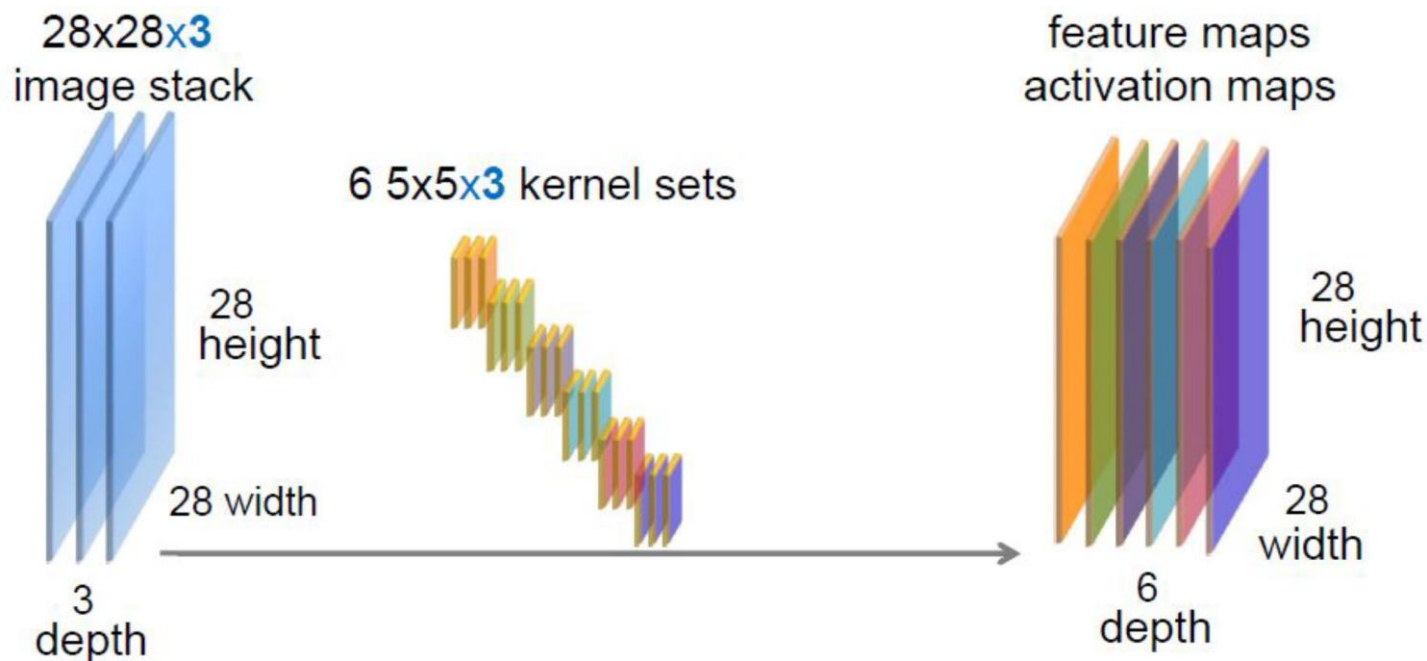


Convolution layer with a 1-channel input and 6 kernels



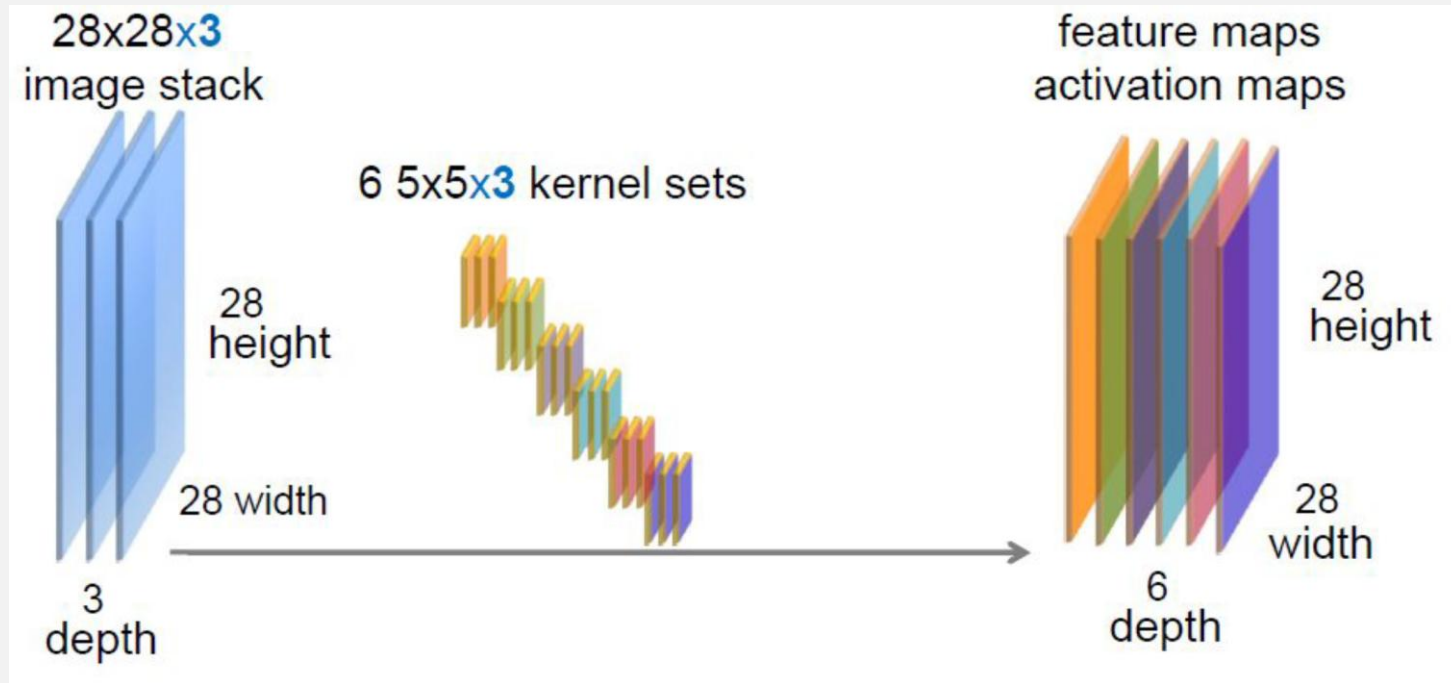
Convolution of the input image with 6 different kernels results in 6 activation maps.
If the input image has only one channel, then each kernel has also only one channel.

Convolution layer with a 3-channel input and 6 kernels



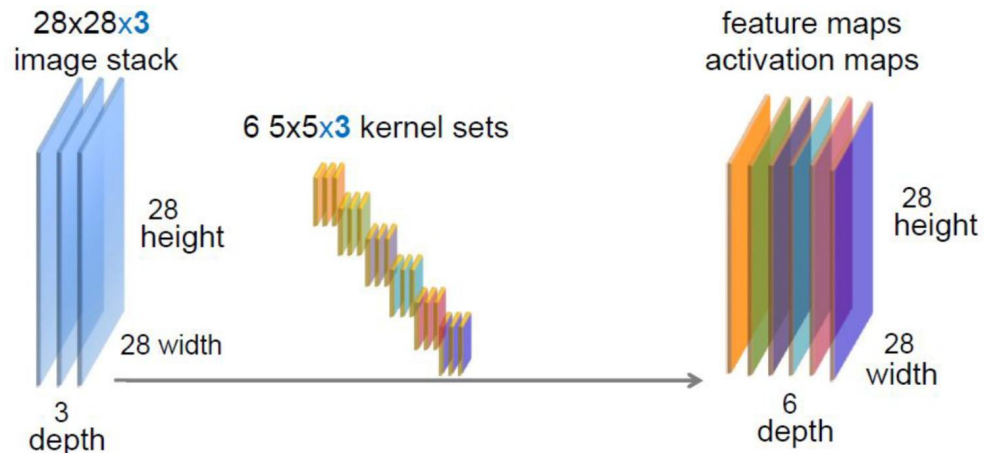
Convolution of the input image with 6 different kernels results in 6 activation maps.
If the input image has 3 channels, then each filter has also 3 channels.

Convolution layer with a 3-channel input and 6 kernels



How many weights?

Solution



$$6 * 5 * 5 * 3 + 6 = 456$$

Number of weights in Convolution

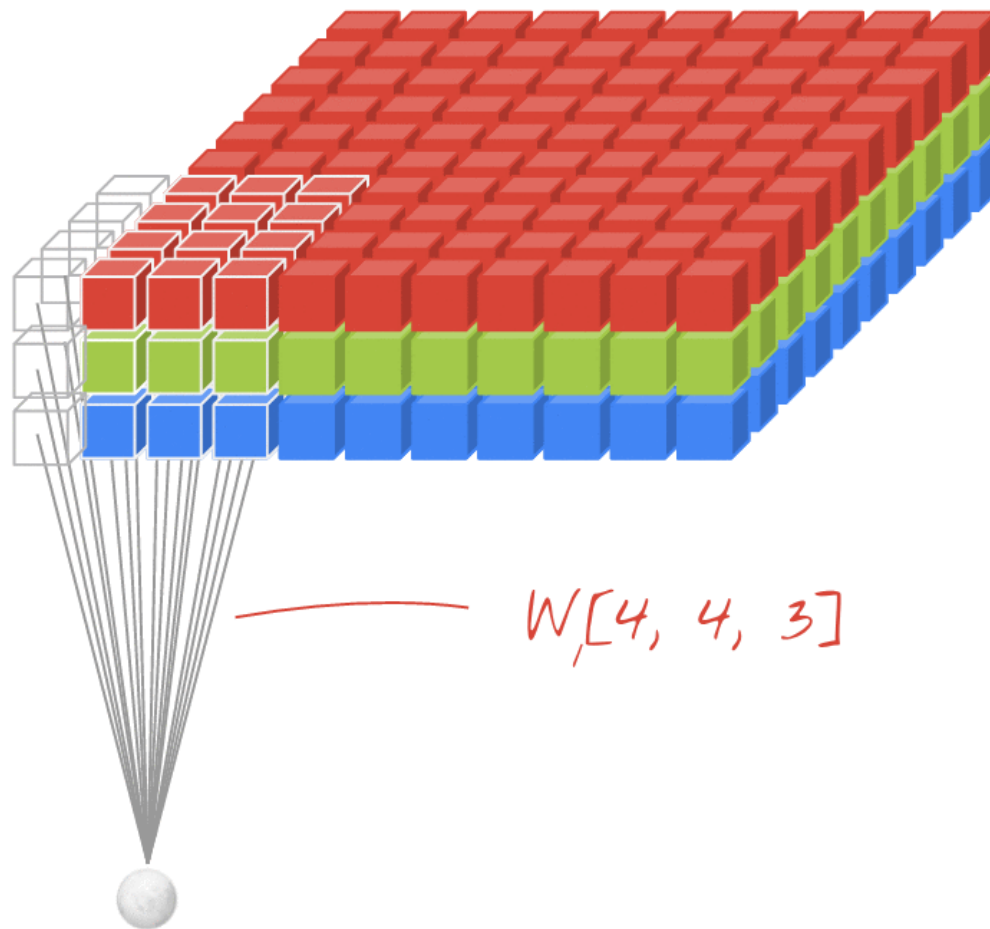
```
model = Sequential()  
model.add(Convolution2D(6, kernel_size=(5, 5), padding='same', input_shape=(28, 28, 3)))  
model.summary()
```

Model: "sequential_11"

Layer (type)	Output Shape	Param #
conv2d_44 (Conv2D)	(None, 28, 28, 6)	456

Total params: 456 (1.78 KB)
Trainable params: 456 (1.78 KB)
Non-trainable params: 0 (0.00 B)

Animated convolution with 3 input channels



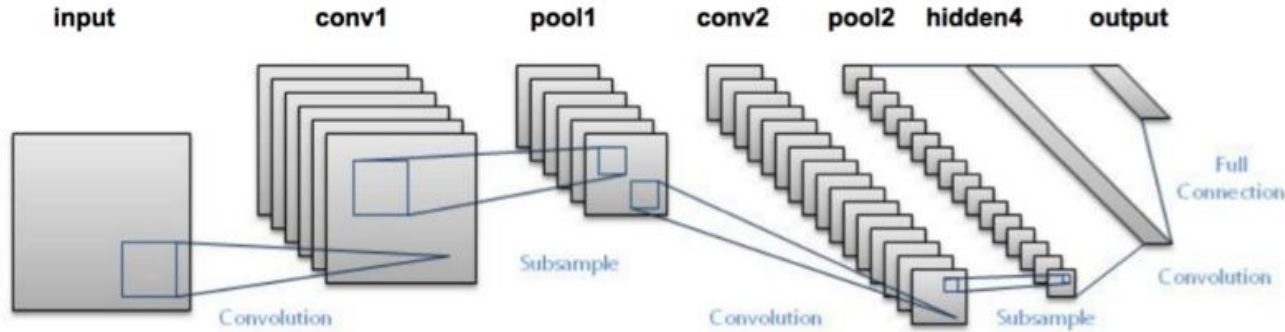
3 color channel input image

$W[4, 4, 3]$

The value of neuron j in the k -th featuremap are computed from the weights in the k -th filter w_{ki} and the input values x_{ji} at the position j :

$$y_{jF_k} = f(z_{jF_k}) = f(b_k + \sum x_{ji} \cdot w_{ki})$$

CNN for MNIST



```
model = Sequential()

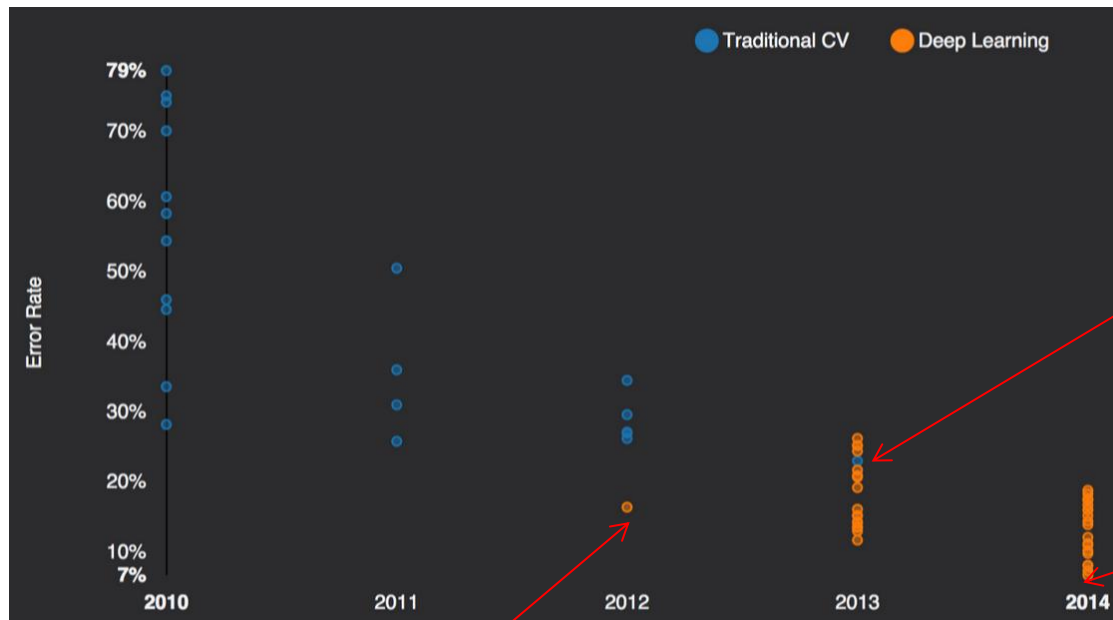
model.add(Convolution2D(filters=8, kernel_size=(3, 3),
                        padding='same', input_shape=(28, 28, 1)))
model.add(Activation('relu'))
model.add(Convolution2D(16, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(40))
model.add(Activation('relu'))
model.add(Dense(10))
model.add(Activation('softmax'))

# compile model and initialize weights
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

What is a good
CNN architecture?

CNN breakthrough in 2012: Imagenet challenge

1000 classes
1 Mio samples



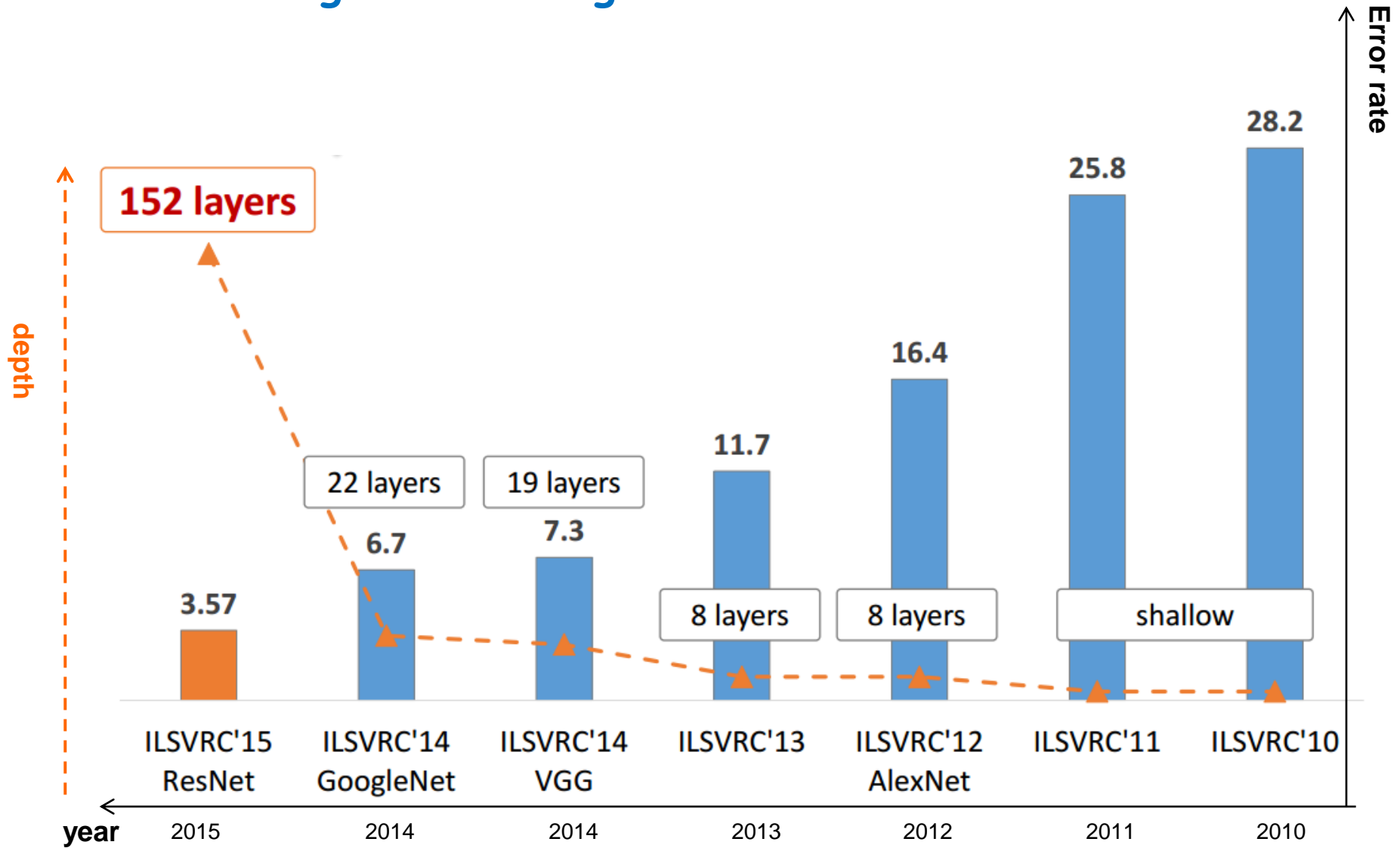
Human: 5% misclassification

Only one non-CNN approach in 2013

GoogLeNet 6.7%

A. Krizhevsky
first CNN in 2012
Und es hat zoom gemacht

Review of ImageNet winning CNN architectures



LeNet-5 1998: first CNN for ZIP code recognition

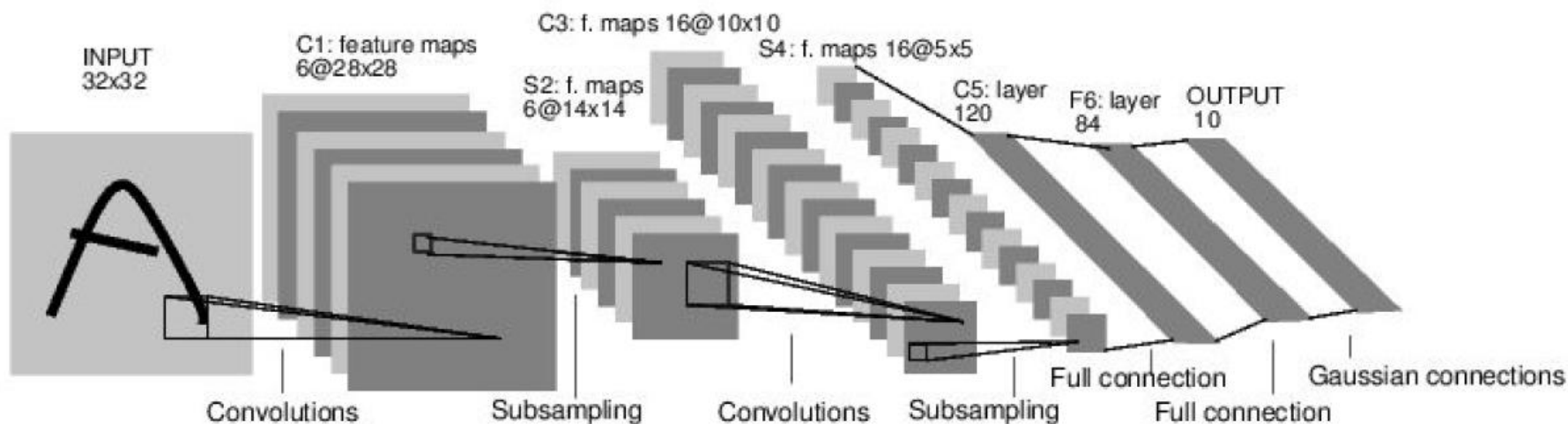


Image credits: <http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf>

Conv filters were 5×5 , applied at stride 1
Subsampling (Pooling) layers were 2×2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

<http://yann.lecun.com/exdb/lenet/index.html>

[Demo von 1993](#) Yann LeCun

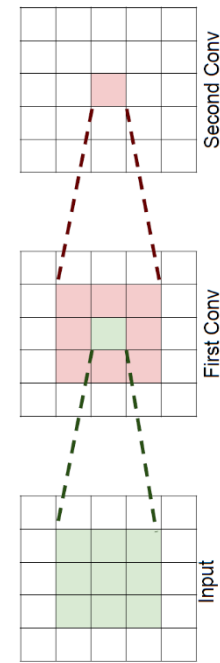
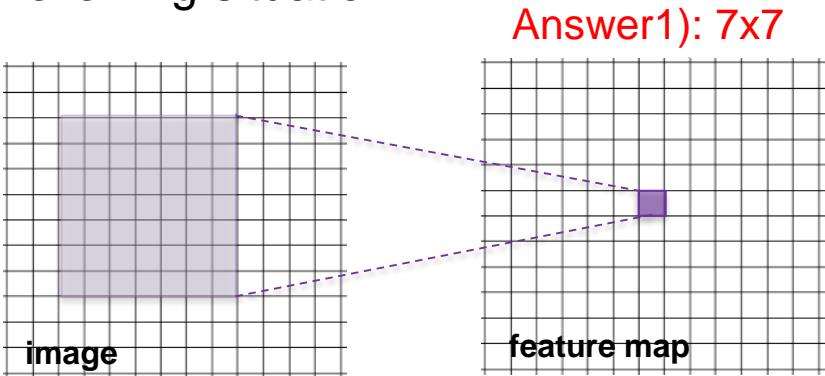


The trend in modern CNN architectures goes to small filters

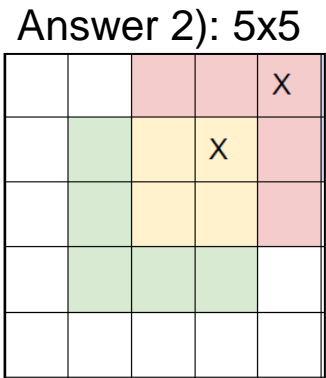
Why do modern architectures use very small filters?

Determine the receptive field in the following situation:

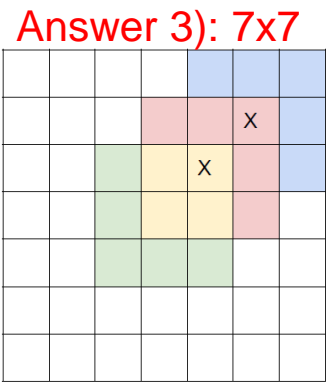
1) Suppose we have **one**
7x7 conv layers (stride 1)
49 weights



2) Suppose we **stack two**
3x3 conv layers (stride 1)



3) Suppose we **stack three**
3x3 conv layers (stride 1)
3*9=27 weights



We need less weights for the same receptive field when stacking small filters!

“Oxford Net” or “VGG Net” 2014 2nd place

- 2nd place in the imageNet challenge
- More **traditional**, easier to train
- Small pooling
- **Stacked 3x3 convolutions before maxpooling**
-> **large receptive field**
- no strides (stride 1)
- ReLU after conv. and FC (batchnorm was not introduced)
- Pre-trained model is available (see exercise)

<http://arxiv.org/abs/1409.1556>



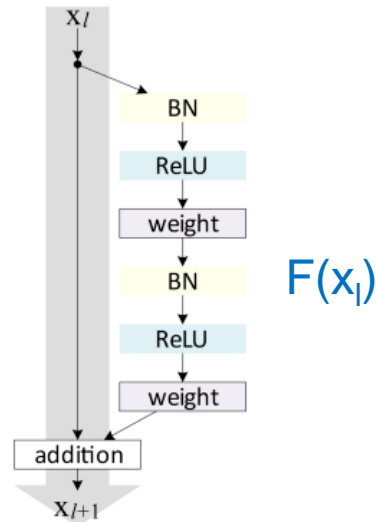
"ResNet" from Microsoft 2015 winner of imageNet

152
layers



ResNet basic design (VGG-style)

- add shortcut connections every two
- all 3x3 conv (almost)

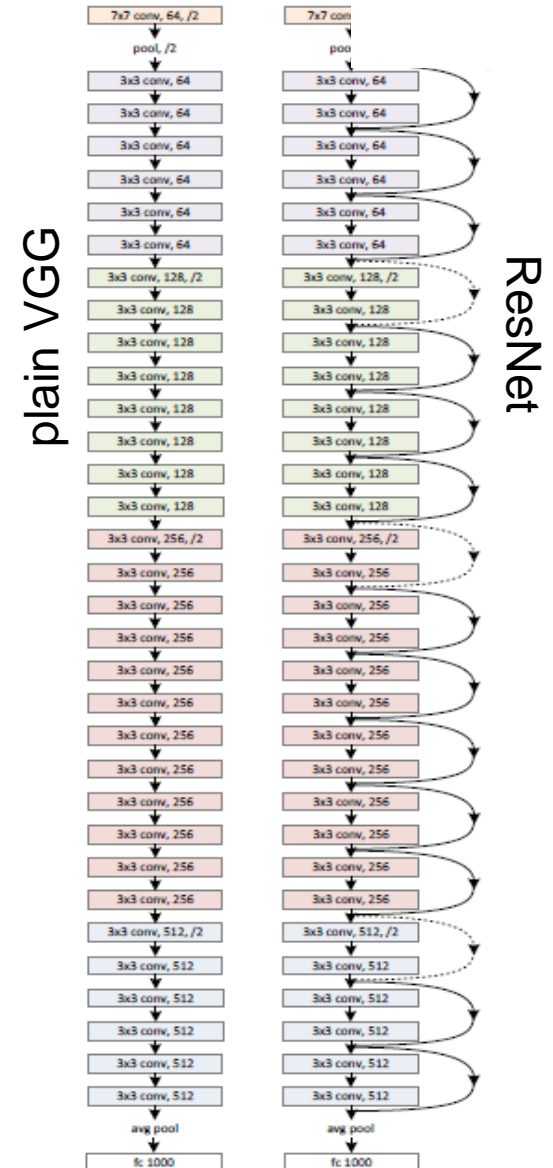


$$H(x_i) = x_{i+1} = x_i + F(x_i)$$

$F(x)$ is called “residual” since it only learns the “delta” which is needed to add to x to get $H(x)$

152 layers:
Why does this train at all?

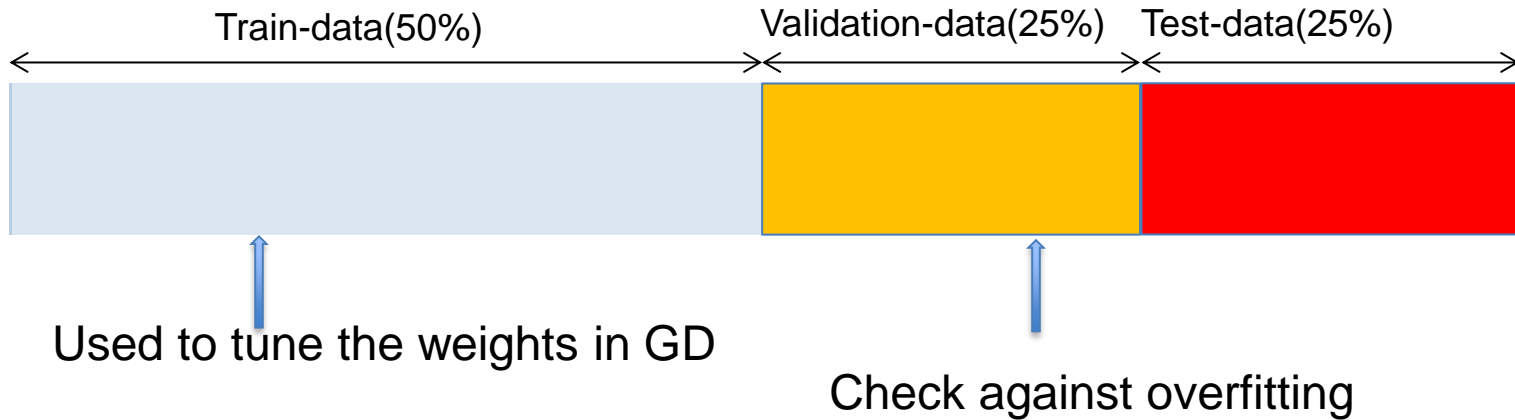
This deep architecture could still be trained, since the gradients can skip layers which diminish the gradient!



Tricks of the trade

- Early stopping
- Input Standardization
- Batch Norm Layer
- Dropout
- Data augmentation

Best practice: Split in Train, Validation, and Test Set

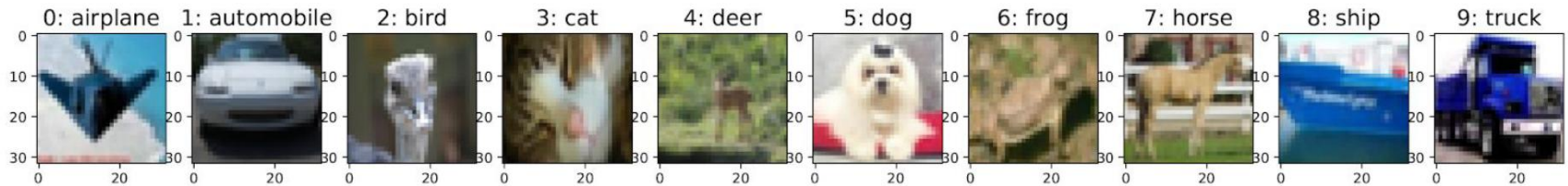


Best practice: Lock an extra **test data set** away, and use it only at the very end, to evaluate the chosen model, that performed best on your validation set.

Reason: **When trying many models, you probably overfit on the validation set.**

Determine performance metrics, such as MSE, to evaluate the predictions **on new validation or test data**

CIFAR10 study with tensorflow notebook



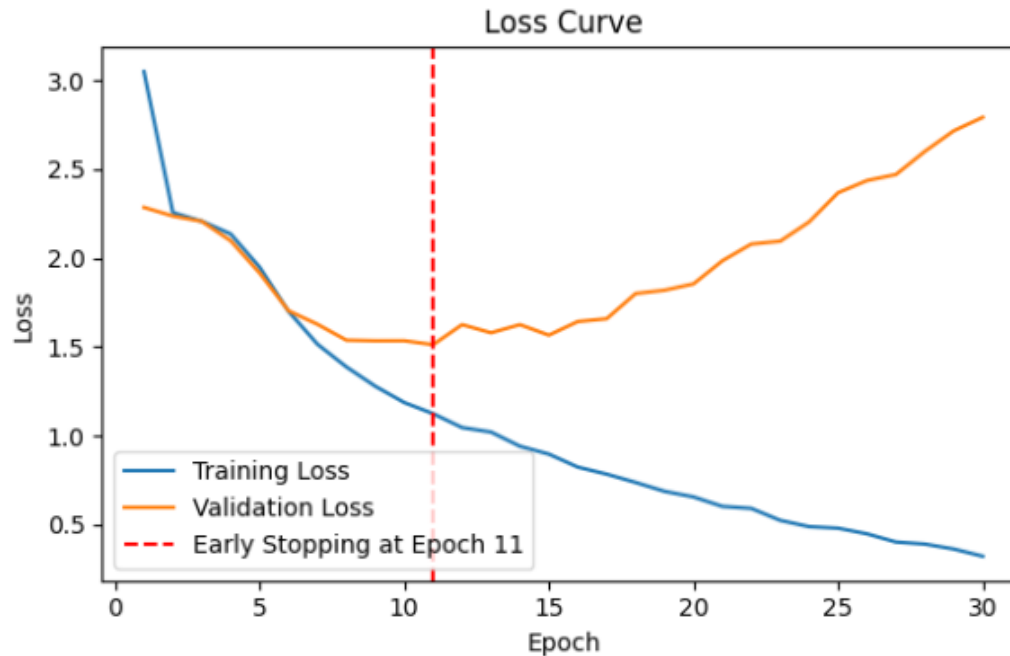
We define a CNN to classify cifar10 images (we have 10 classes)

Notebook:

https://github.com/tensorchiefs/dl_course_2025/blob/master/notebooks/07_cifar10_tricks_keras_torch.ipynb

Loss curve and early stopping

Very common check: Plot loss in train and validation data vs epoch of training.



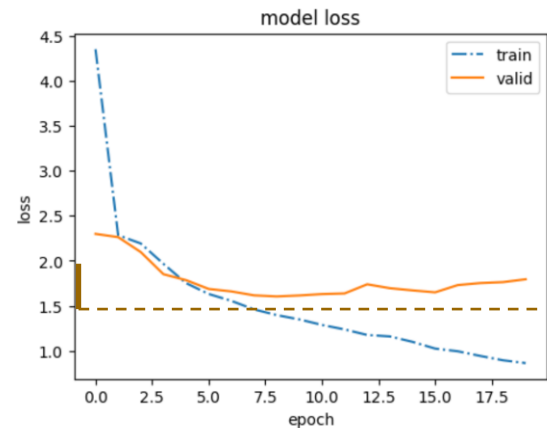
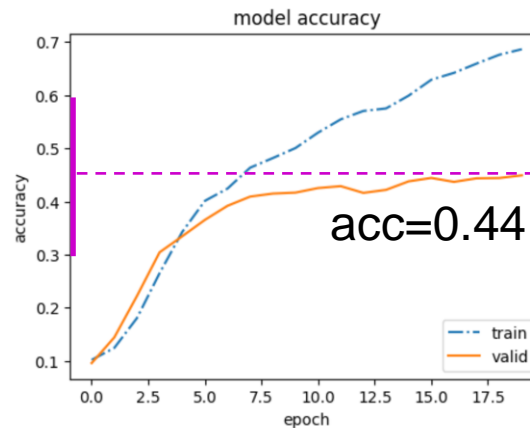
Training completed all epochs.
Best model weights were restored from epoch 11

- If **training loss** does not go down to zero: model is not flexible enough
- Use weights @minimum of validation before overfitting
- Early stopping: stop to training if validation loss does not improve anymore

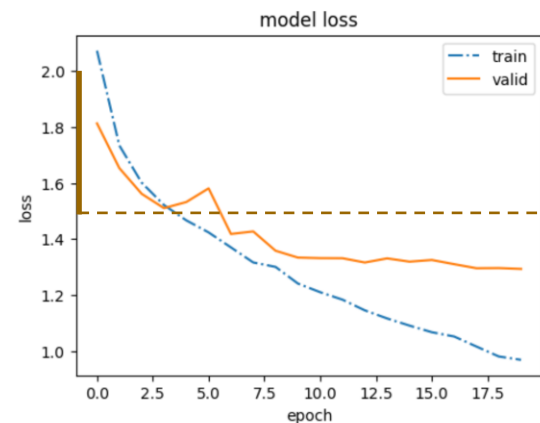
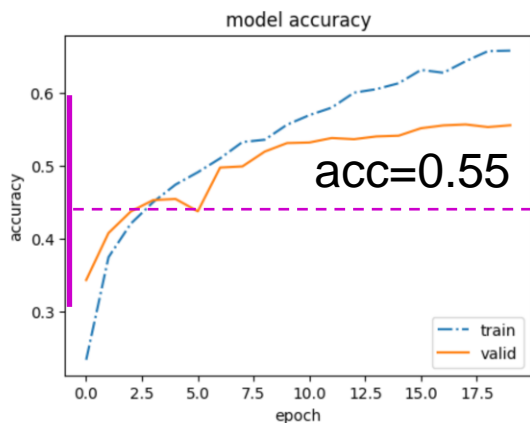
Take-home messages from CIFAR10 CNN study

- 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



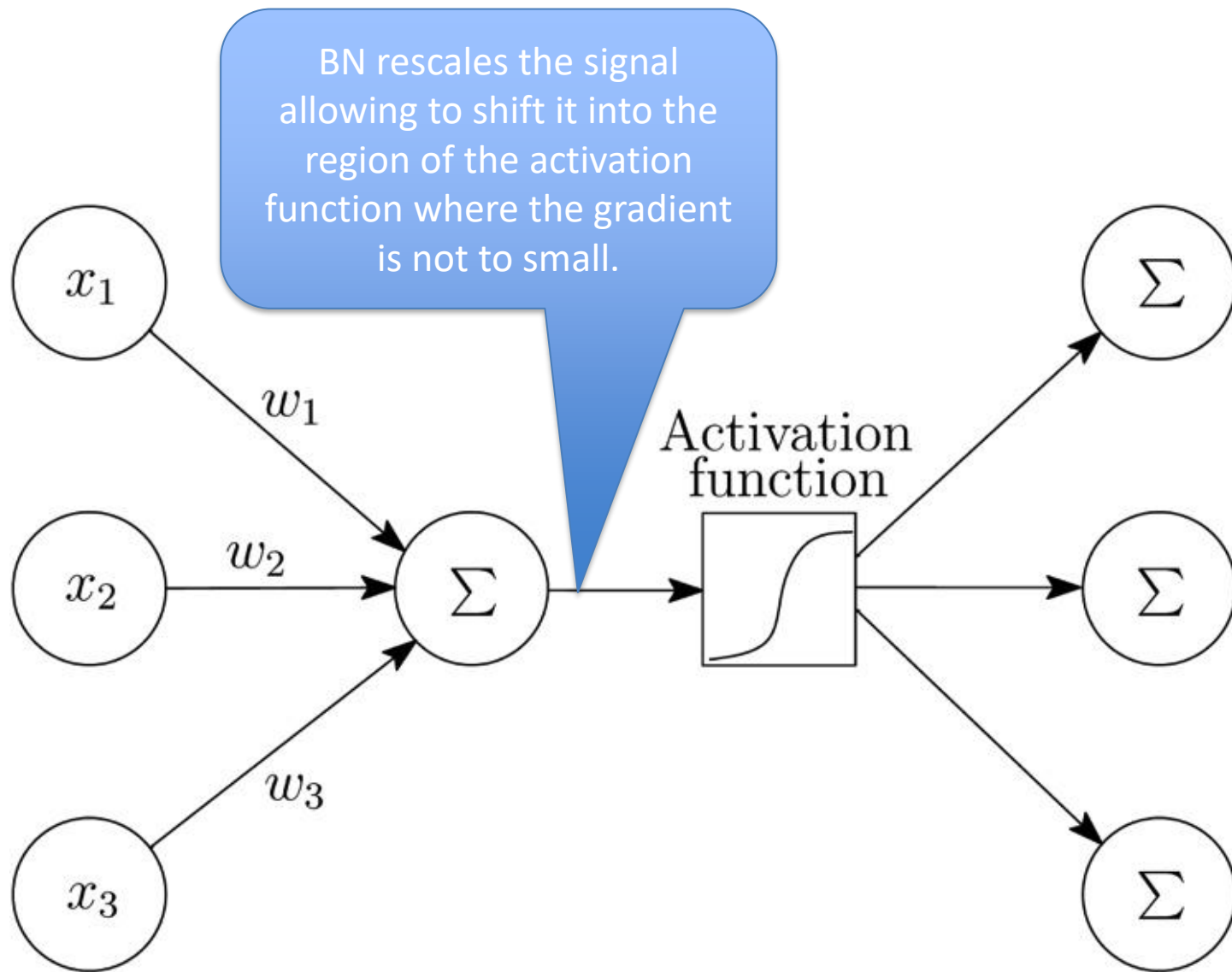
With normalizing
(pixel-value/255)
the input to the CNN



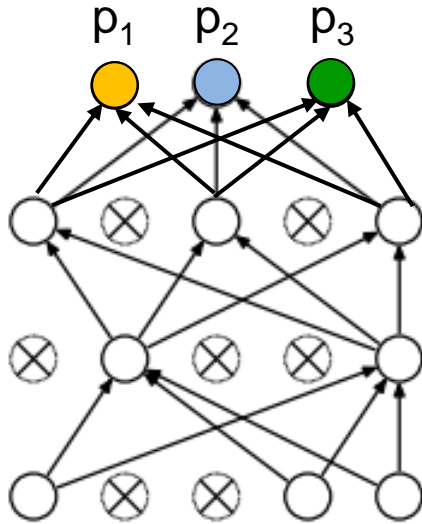
Regularization to avoid overfitting

- Batchnorm layer
- Dropout layer

What is the idea of Batch-Normalization (BN)



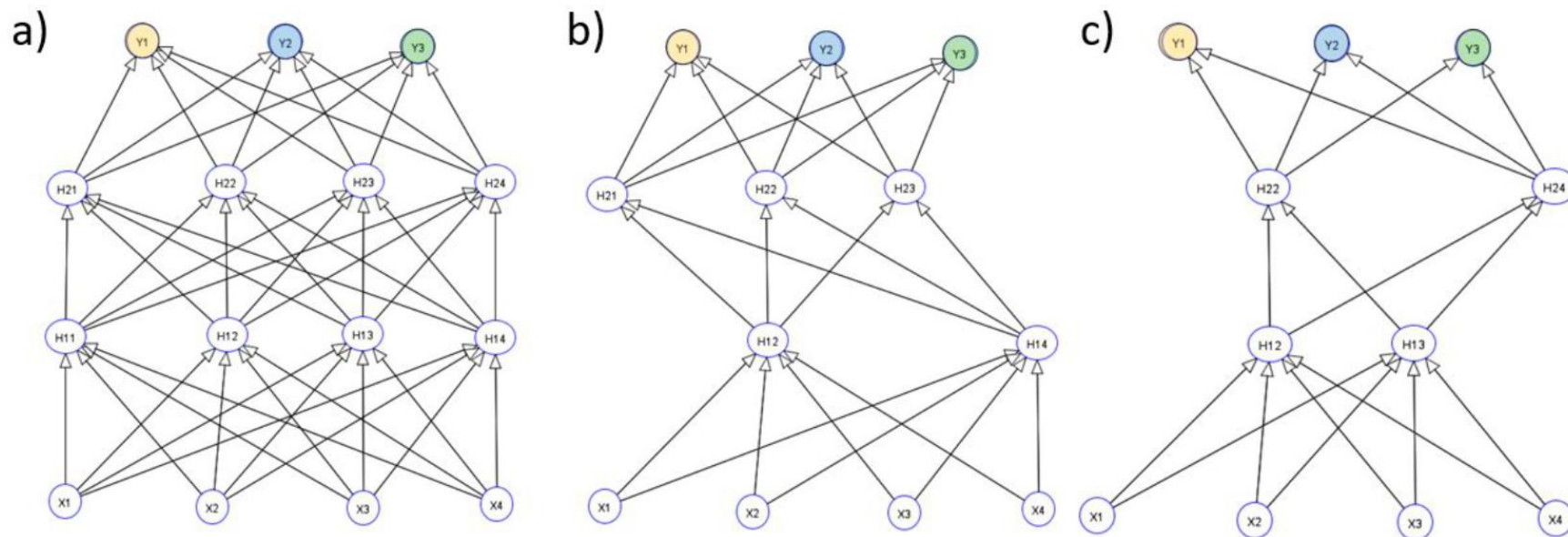
Dropout helps to fight overfitting



Using dropout during training implies:

- In each training step only weights to not-dropped units are updated \rightarrow 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”
- JFI: To get same expected output in training (with dropout) and after training (test time - without dropout), the weights are multiplied after training by the dropout probability $p=0.5$.

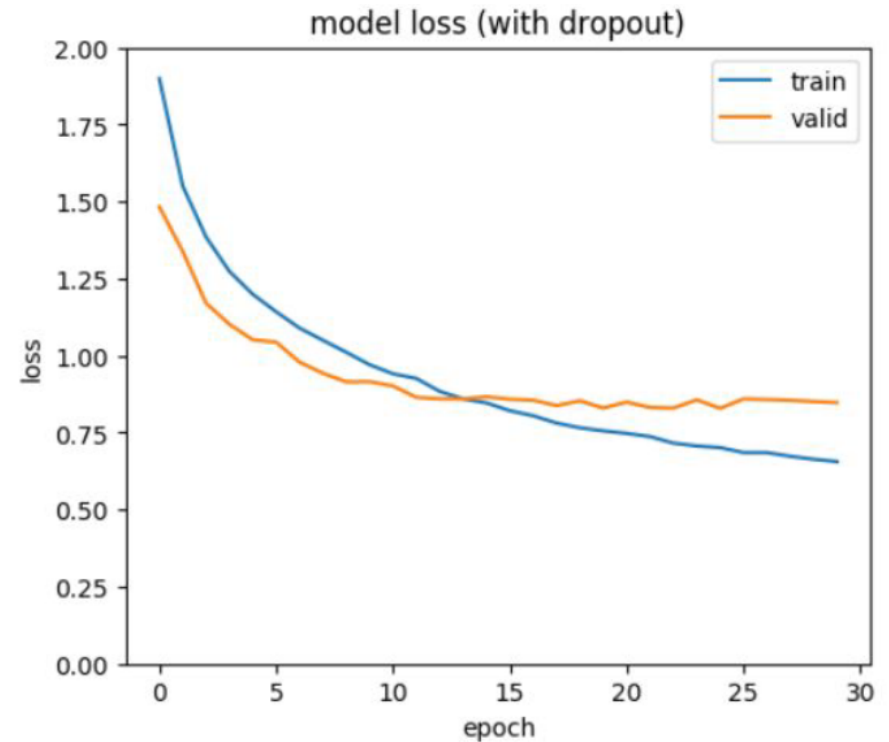
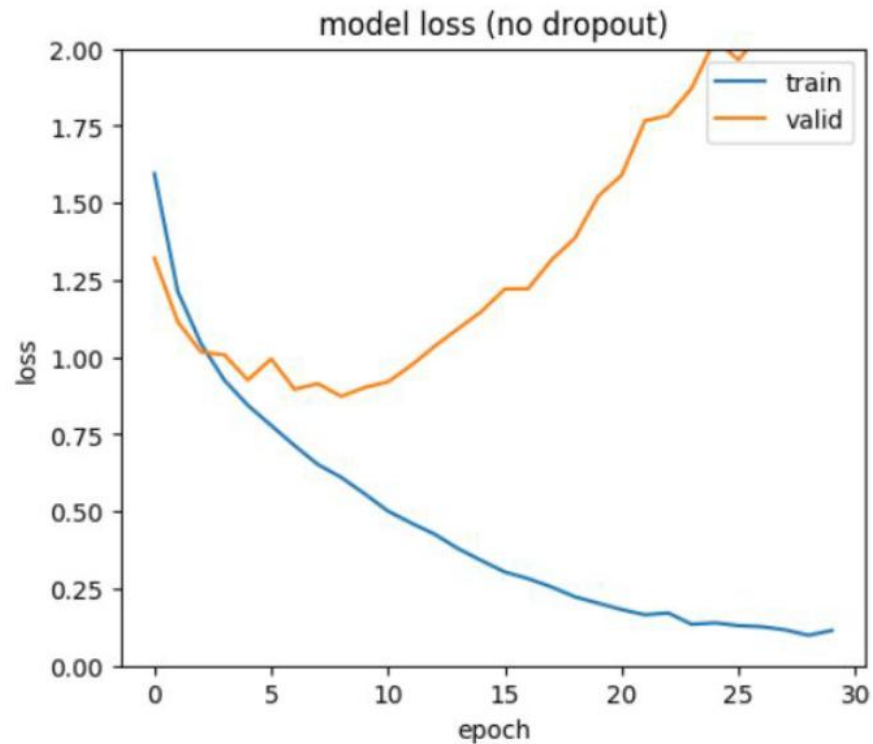
Dropout



Three NNs:

a) shows the full NN with all neurons (as used when NN is trained),
b) and c) show two versions of a thinned NN where some neurons are dropped (as done during training with dropout). Dropping neurons is the same as setting all connections that start from these neurons to zero.

Dropout fights overfitting in a CIFAR10 CNN



Summary

- The NN architecture choice depends on the structure of the data.
 - Fully connected NNs work best for tabular data
 - CNNs work best for images and other data with local order
- CNNs exploit the local structure of images by local connections and shared weight (same kernel is applied at each position of the image).
- Use tricks of the trade when building a CNN
 - Normalization of input data
 - Batchnorm
 - Dropout
 - Augmentation
 - Use challenge winning architectures

