VL Deep Learning for Natural Language Processing

14. Convolutional Neural Networks

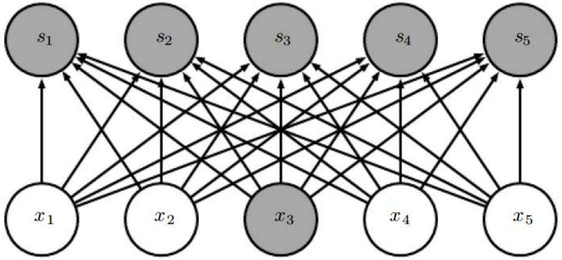
Prof. Dr. Ralf Krestel

AG Information Profiling and Retrieval

Convolutional Neural Networks

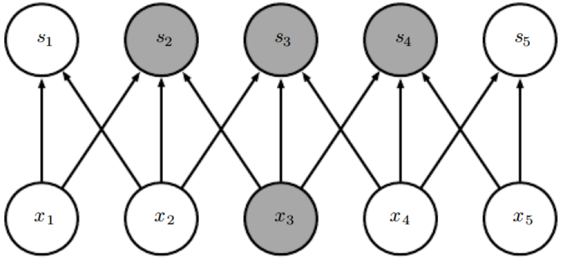
- Very sucessful in computer vision
  - Uses convolutions
- Convolutional layer is not densly connected
  - Densly connected layer:

Matrix multiplication



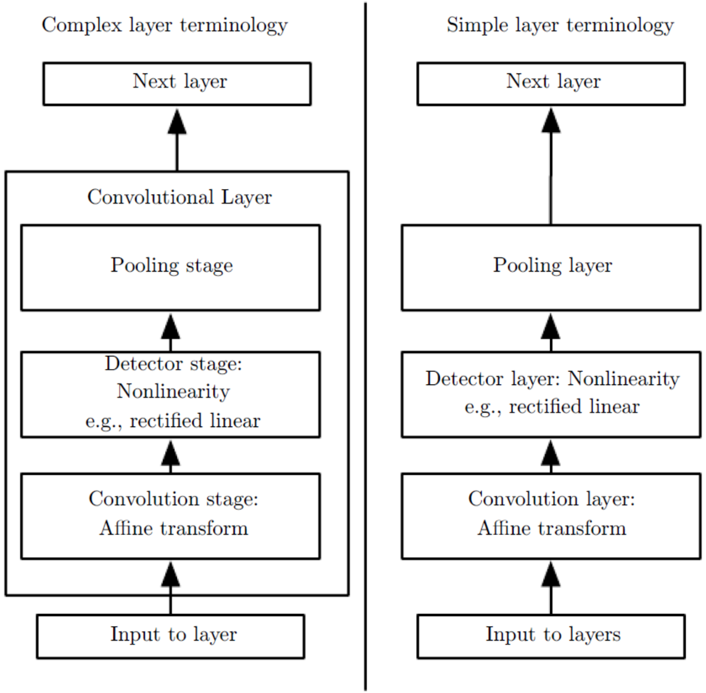
  - Convolutional layer

Convolution with kernel of size 3



Convnets

- Various names in the literature
  - Complex representation
    - Convolutional layer consists of multiple phases
  - Simple representation
    - Each phase is considered its own layer
  - Compromise in Keras
    - Convolutional layer
    - + Pooling layer



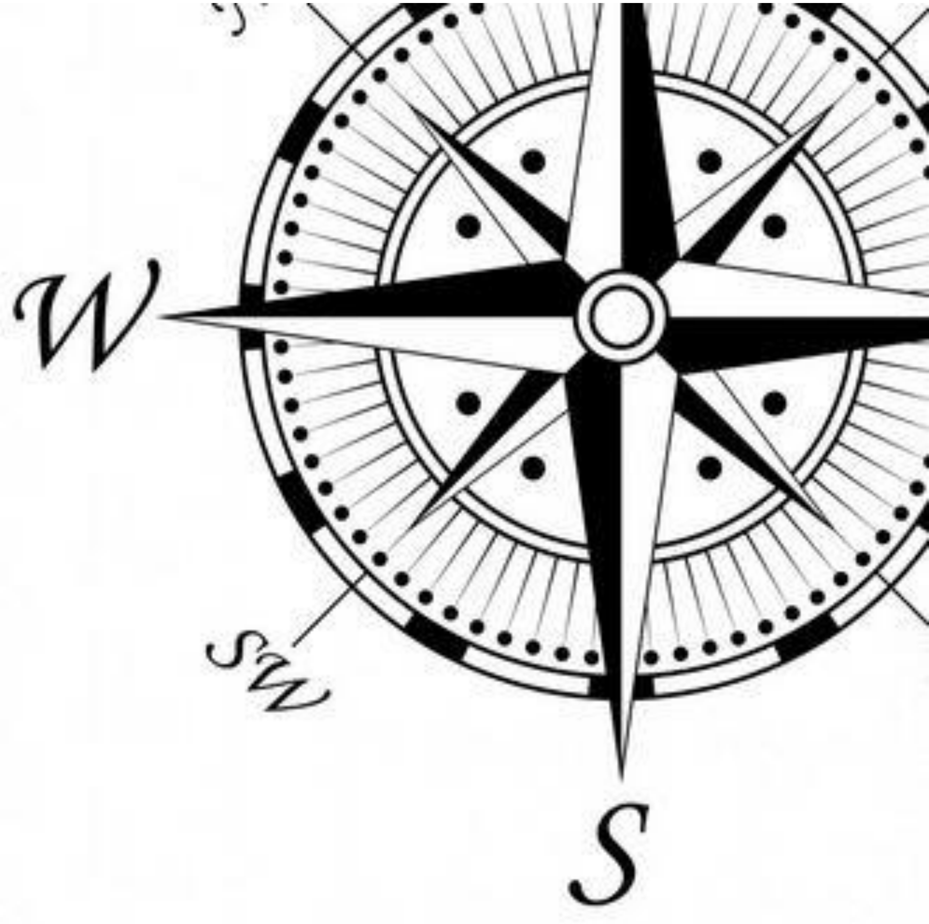
Learning Goals for this Chapter

- Understand the basics of convnets
  - feature maps
  - convolutions
  - max-pooling
- Augment training data
- Fight over- and underfitting
- Employ pretrained networks
- Fine-tune convnets

- Relevant chapters:
  - P5
  - S11 (2019) https://www.youtube.com/watch?v=EAJoRA0KX7I

Topics Today

1. **Convolutional Neural Networks**
2. Over- and Underfitting
3. Data Augmentation
4. Pretrained CNN
5. 1D-CNN



CNN Modell

Number of features

Size of patches

image_height, image_width, image_channels

```
from keras import layers
from keras import models
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.summary()
```

```
Layer (type)                    Output Shape          Param #
=================================================================
conv2d_1 (Conv2D)               (None, 26, 26, 32)    320
_________________________________________________________________
maxpooling2d_1 (MaxPooling2D)   (None, 13, 13, 32)    0
_________________________________________________________________
conv2d_2 (Conv2D)               (None, 11, 11, 64)    18496
_________________________________________________________________
maxpooling2d_2 (MaxPooling2D)   (None, 5, 5, 64)      0
_________________________________________________________________
conv2d_3 (Conv2D)               (None, 3, 3, 64)      36928
_________________________________________________________________
flatten_1 (Flatten)             (None, 576)           0
_________________________________________________________________
dense_1 (Dense)                 (None, 64)            36928
_________________________________________________________________
dense_2 (Dense)                 (None, 10)            650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
```

CNN Training und Evaluation

```
from keras.datasets import mnist
from keras.utils import to_categorical
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
model.compile(optimizer='rmsprop',
        loss='categorical_crossentropy',
        metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
test_loss, test_acc = model.evaluate(test_images, test_labels)
>>> test_acc
0.99080
```

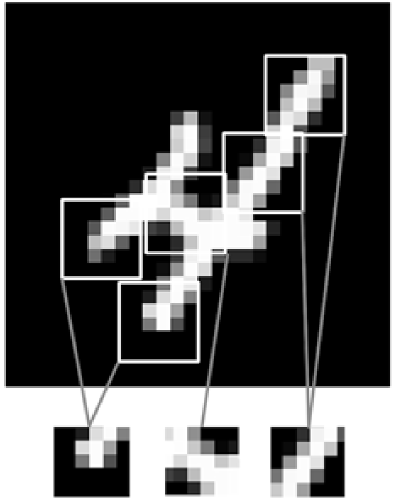
Test accuracy with densely connected neural net: 97.8%

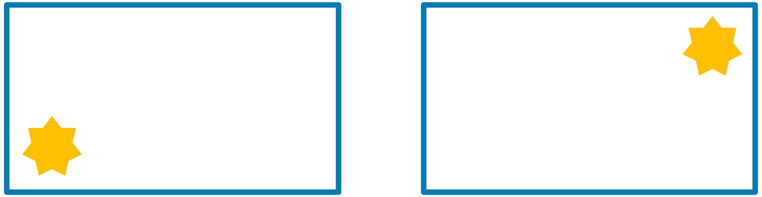
Convolutional Layer I

- **Densly connected layers** learn to detect **global** patterns.

- **Convolutional layers** learn **local** patterns.
  - In the previous example these patterns were of size (3,3)

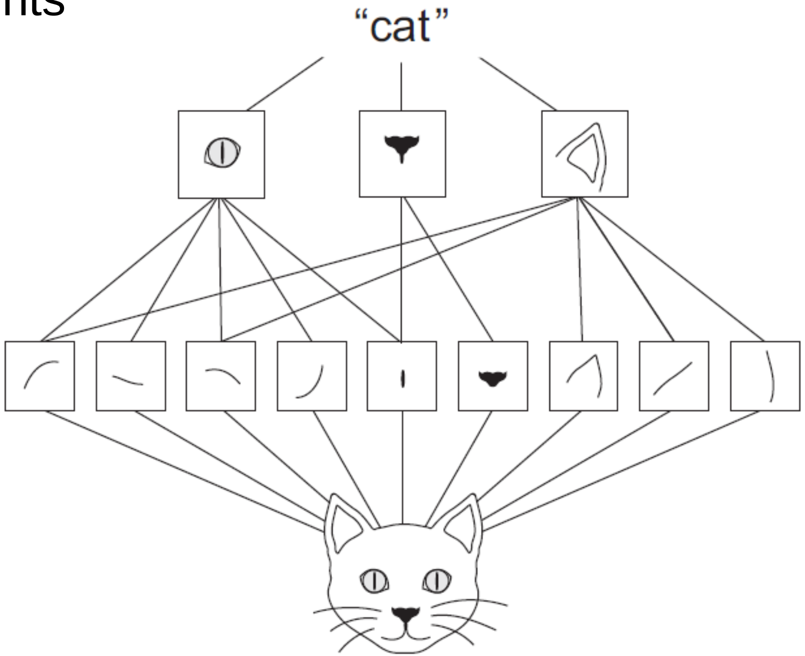
- The learnt patterns are **translation invariant**.
  - A learnt pattern from the bottom left can also be detected in the top right.





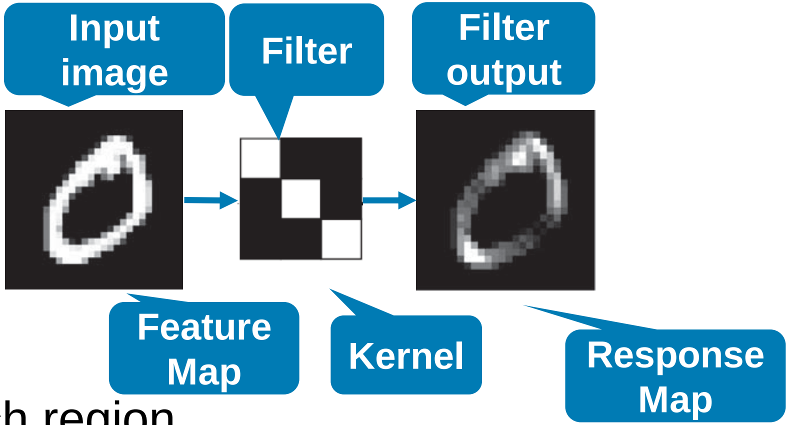
Convolutional Layer II

- Multiple convolutional layers can detect **hierarchies of patterns**.
  - Increase of complexity of components
  - Possible to learn concepts



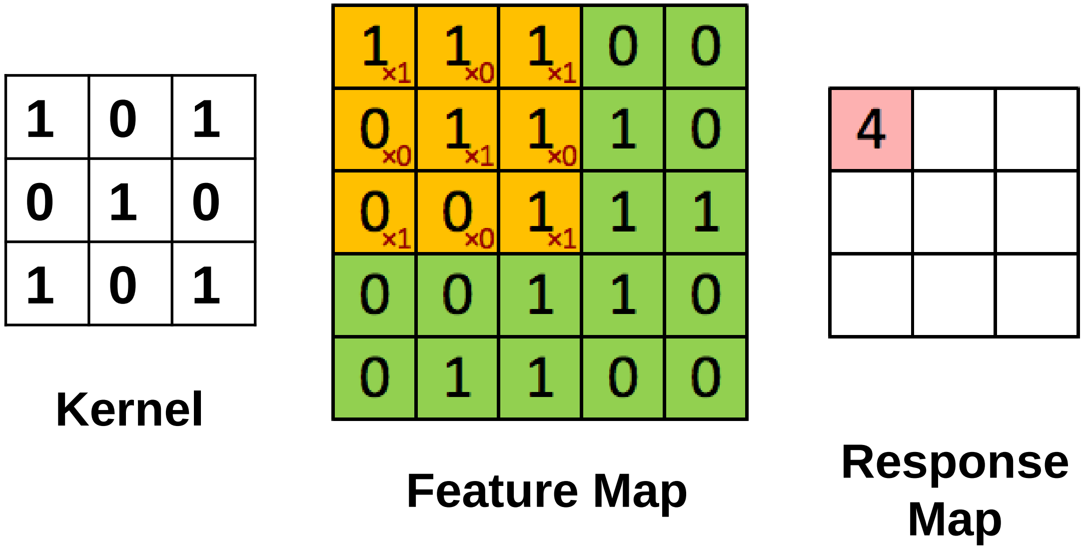
Convolutional Layer III

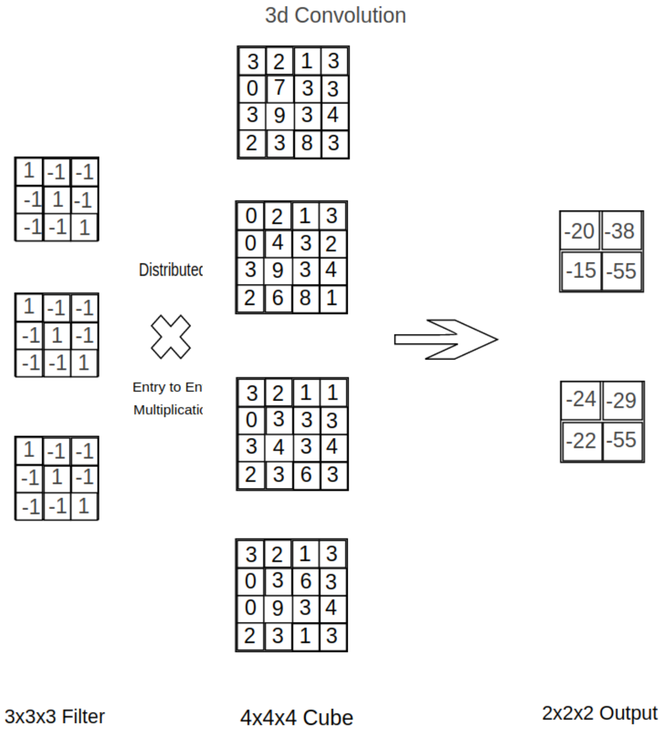
- Input for a convolutional layer are 3D tensors, so-called **feature maps**
  - Two spacial axes (height/width)
  - One depth axis (channel axis)
- An input image (feature map) is split up in many (overlapping) regions.
  - Depth axis represents
    - channels in the input
    - filters in the convolutional layer



- Different filters (kernels) are applied to each region.
  - Output: **response map** for each filter
  - indicates how strong a particular pattern (feature) is present at the given position
- To get the final output, the results of the different regions are assembled in an output feature map.

Convolutional Layer IV

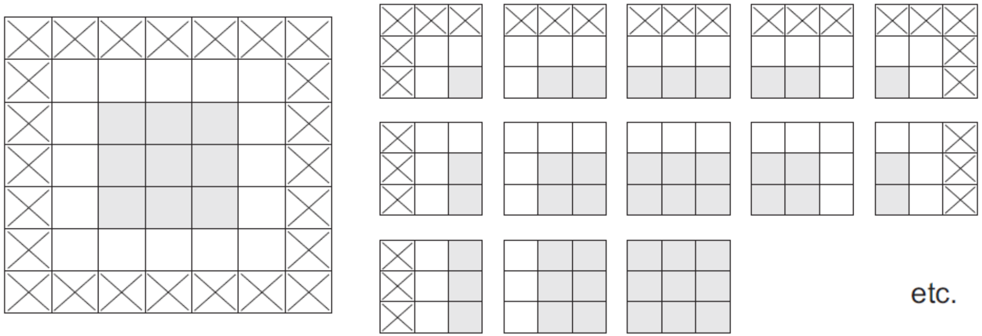


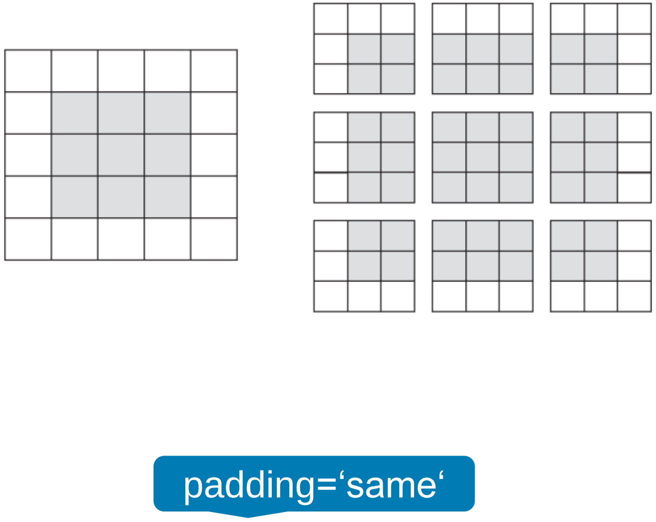


https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

https://towardsdatascience.com/step-by-step-implementation-3d-convolutional-neural-network-in-keras-12efbdd7b130

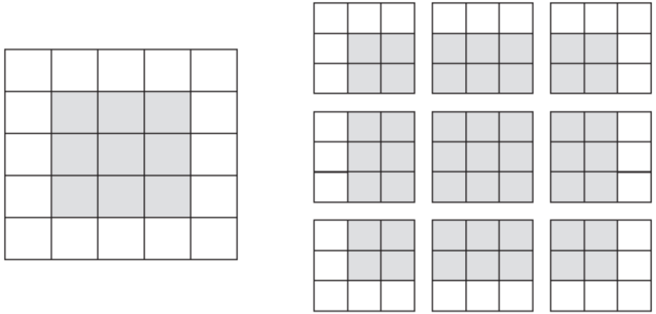
Padding

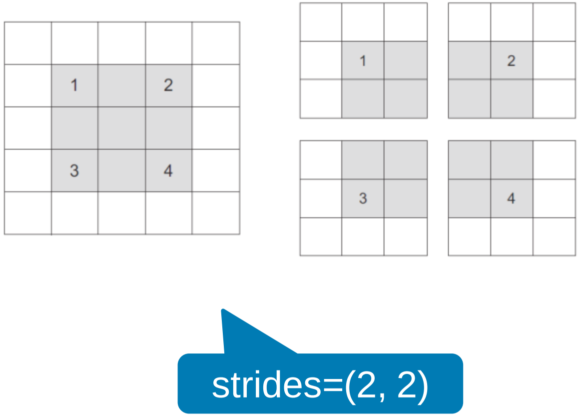





```
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,
    dilation_rate=(1, 1), activation=None, use_bias=True, kernel_initializer='glorot_uniform',
    bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None,
    activity_regularizer=None, kernel_constraint=None, bias_constraint=None)
```

Strides






```
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,
    dilation_rate=(1, 1), activation=None, use_bias=True, kernel_initializer='glorot_uniform',
    bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None,
    activity_regularizer=None, kernel_constraint=None, bias_constraint=None)
```

Pooling Layer

- Pooling layer compresses the feature maps (similar to convolutions).
  - Typically four pixels are aggregated (window of size (2, 2)).
  - Not overlapping

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
maxpooling2d_1 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_2 (Conv2D)	(None, 11, 11, 64)	18496
maxpooling2d_2 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_3 (Conv2D)	(None, 3, 3, 64)	36928
flatten_1 (Flatten)	(None, 576)	0
dense_1 (Dense)	(None, 64)	36928
dense_2 (Dense)	(None, 10)	650

Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0

- Max-Pooling takes the maximum of each channel for a (2, 2) region
  - Analog: average-pooling

Usually
- Convolution: window=3x3; stride=1
- Pooling: window=2x2; stride=2

- Pooling is necessary, since without pooling
  1. No hierarchy of features could be leant
  2. The number of parameters would explode → Overfitting
     - In our example: 15.8 million parameters

Why Pooling?

```
model_no_max_pool = models.Sequential()
model_no_max_pool.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28,28,1)))
model_no_max_pool.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_no_max_pool.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 22, 22, 64)        36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
```

- Two problems:
  1. Cannot learn spatial hierarchy of features
     - First layer $(3 \times 3) \rightarrow$ Third layer (7×7)
  2. Model too complex
     - Final layer has $22 \times 22 \times 64 = 30{,}976$ coefficients
       (+ dense layer with 512 units = 15,8 million parameters (weights) to learn)

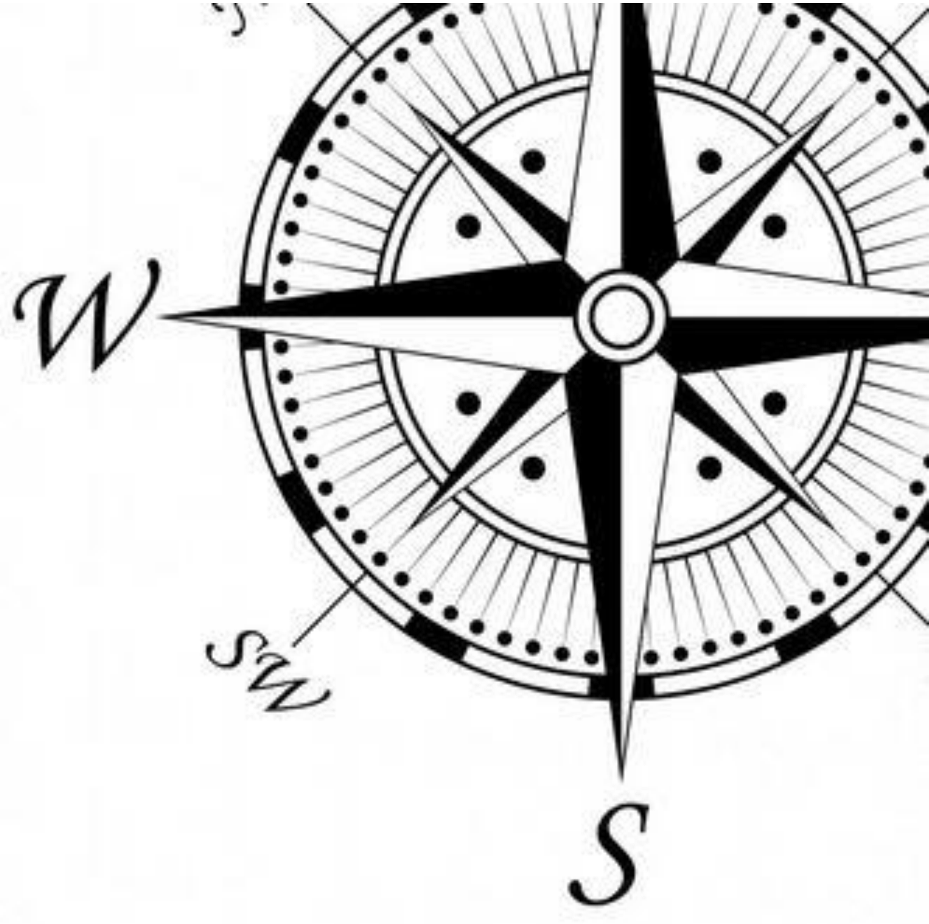
Overfitting!!!

Convolutional Layer

- Implement the MNIST example.
  - What happens if you change
    - The number of layers?
    - The window and/or stride sizes?
    - The number of channels/features?

Start 5 4 3 2 1 End

Topics Today



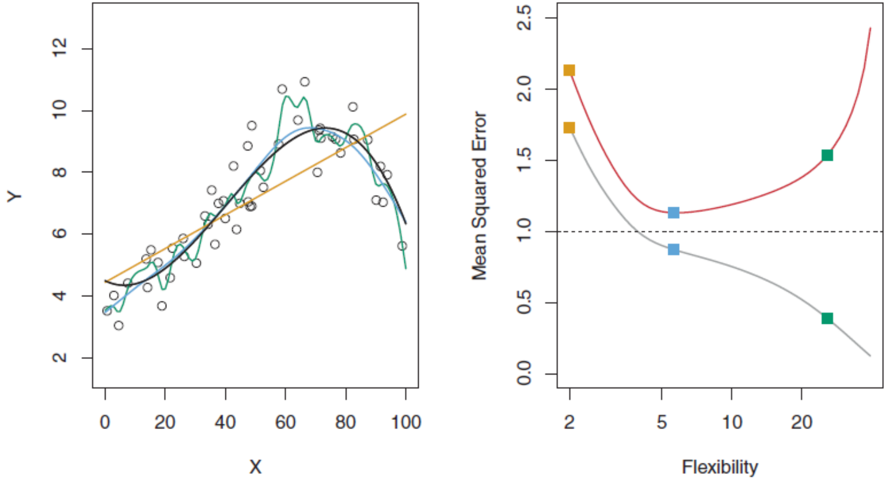
1. Convolutional Neural Networks
2. **Over- and Underfitting**
3. Data Augmentation
4. Pretrained CNN
5. 1D-CNN

Overfitting

- General problem
  - Optimization vs. Generalization
- At beginning of training…
  - NN learns general patterns
    - **Training loss and validation loss decrease**
    - The model is **underfitted**
- After long training
  - NN learns very specific patterns from prevalent in training data
    - **Training loss decreases** but **validation loss increases**
    - The model is **overfitted**
- Right amount of training is important and training process needs to be monitored
  - Using validation data

- Four remedies against overfitting:
  - More training data
  - Decreasing network capacity
  - Regularization of network weights
  - Regularization using dropout

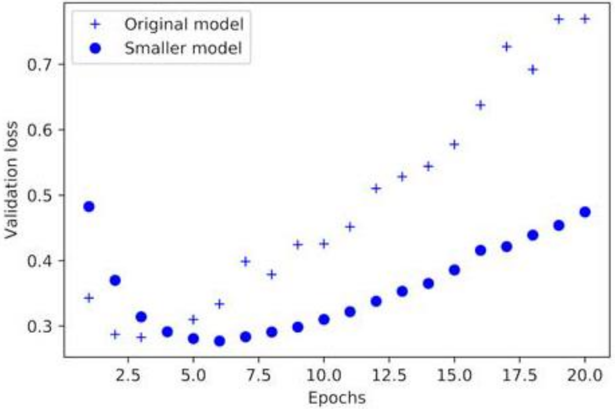
Decreasing Network Capacity I



- Easiest way to prevent overfitting: decreasing size/capacity/number of parameters of the model to to be trained

- More parameters ⟶ more possibilities for the model to learn the data **by heart**
  - Extreme case: More parameter than training samples
- Less parameters ⟶ model ist „forced“ to detect **general** patterns
- Too few parameters ⟶ underfitting

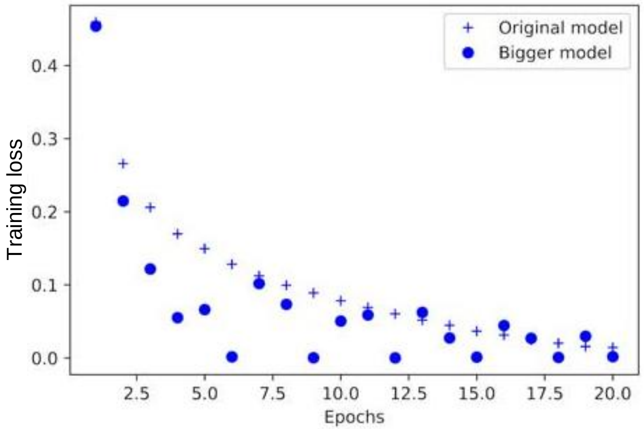
Decreasing Network Capacity II

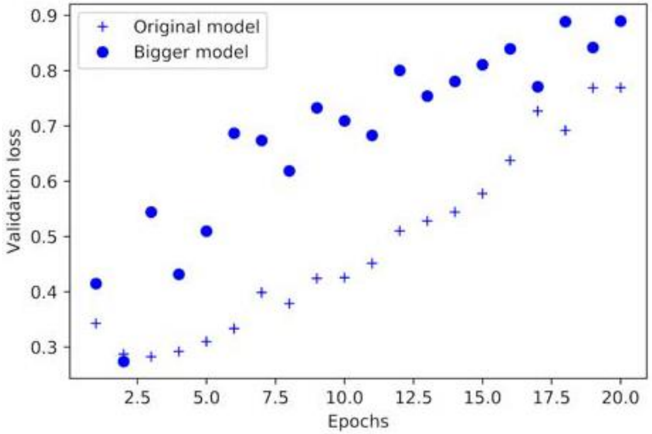



```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model = models.Sequential()
model.add(layers.Dense(4, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(4, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Decreasing Network Capacity III

```
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```





Weight Regularization I

- Idea: less complex models are less prone to overfitting.
- Simple model: parameter distribution has low entropy.
  - Goal: Weights rather small
- Penalizing of models with large weights
  - Inclusion in the loss function
    - E.g. using MSE: **objective function** to be minimized:

$$J(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=0}^{d} w_j^2$$

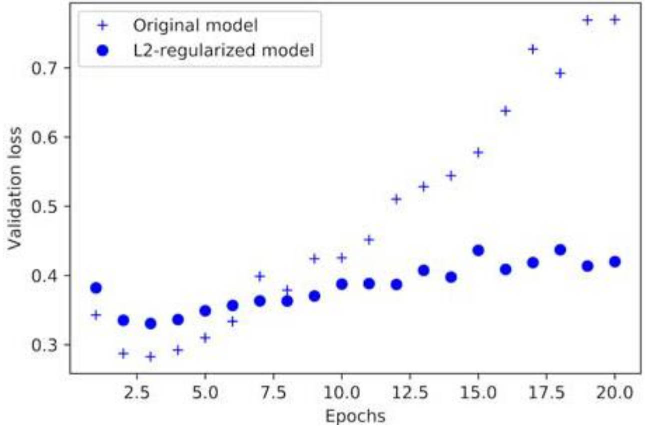
- **L1-Regularization**
  - Absolute values of weights ($|w|$)
  - Side effect: „Unimportant“ weights become $0 \longrightarrow$ smaller model
- **L2-Regularization**
  - Squared values of weights (w^2)

Weight Regularization II

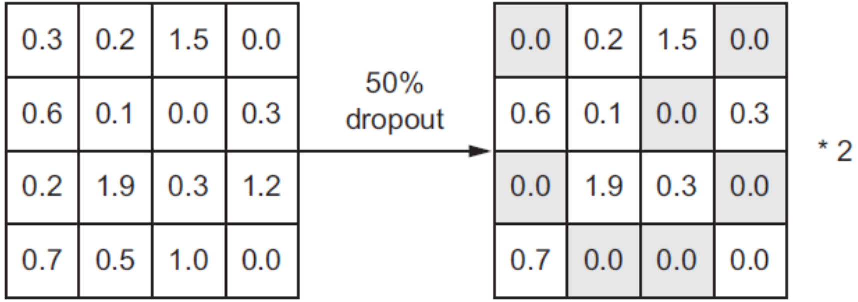
Weights are included in loss function with 0.001 times their squared value.

λ

```
from keras import regularizers
model = models.Sequential()
model.add(layers.Dense(16,kernel_regularizer=regularizers.l2(0.001),
activation='relu',input_shape=(10000,)))
model.add(layers.Dense(16,kernel_regularizer=regularizers.l2(0.001),
activation='relu'))
model.add(layers.Dense(1,activation='sigmoid'))
```



Dropout I

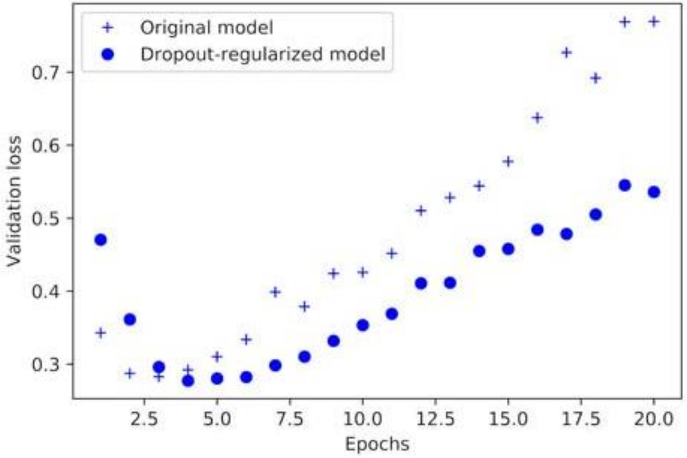


- Especially developed for deep learning
- Individual output values are set to 0 randomly during training (dropout).
  - Between 20% and 50%
  - The remaining values are scaled accordingly

- Idea: Including noise prevents the network to learn insignificant, random patterns from training data.

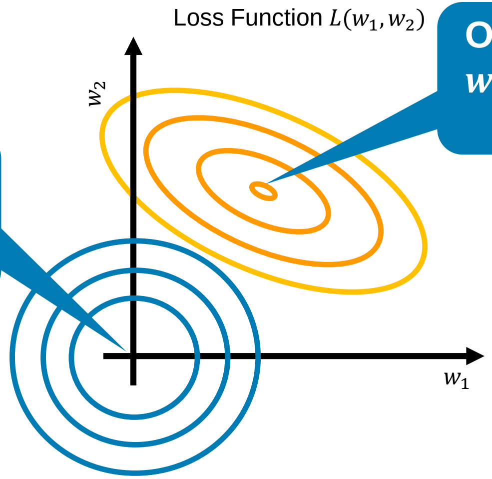
Dropout II

```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))
```



Regularization

- How do w_1 and w_2 change when an l_1- or l_2-regularizer is added?
  - Think of an geometric interpretation!



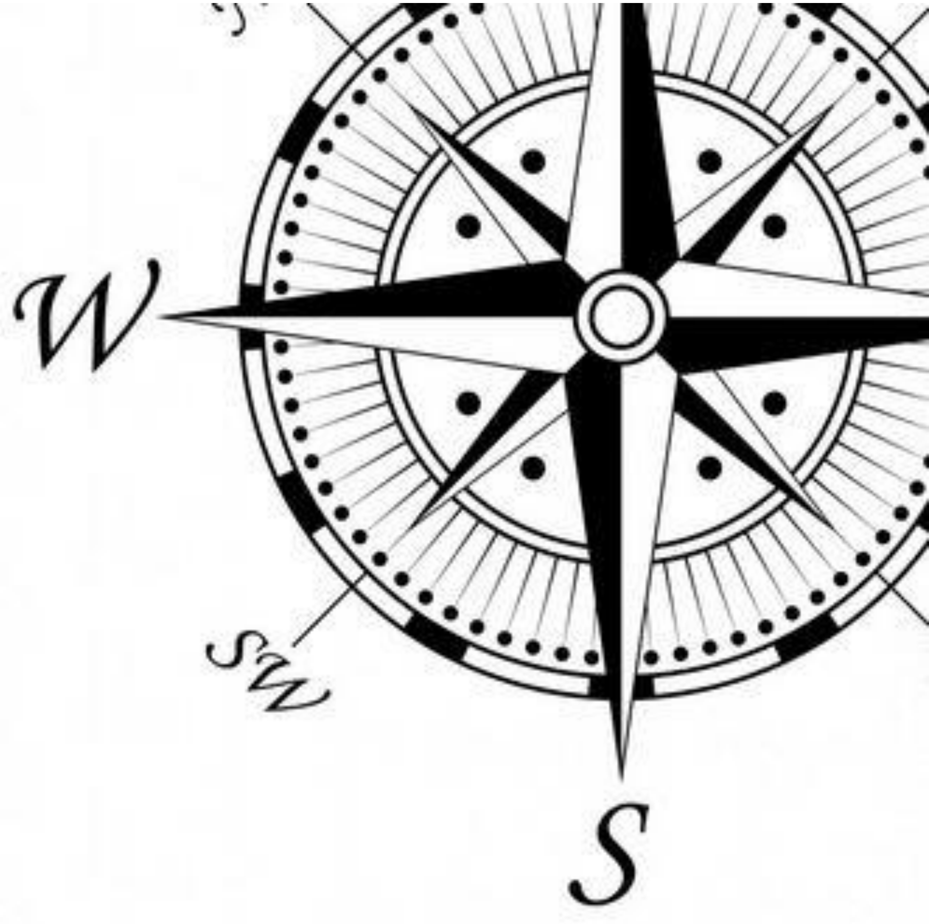
Optimal values for w_1 and w_2 for L2-regularizer

Optimal values for w_1 and w_2 without regularization

How does this look for an L1-regularizer?

Start 5 4 3 2 1 End

Topics Today



1. Convolutional Neural Networks
2. Over- and Underfitting
3. **Data Augmentation**
4. Pretrained CNN
5. 1D-CNN

Generating Training Data

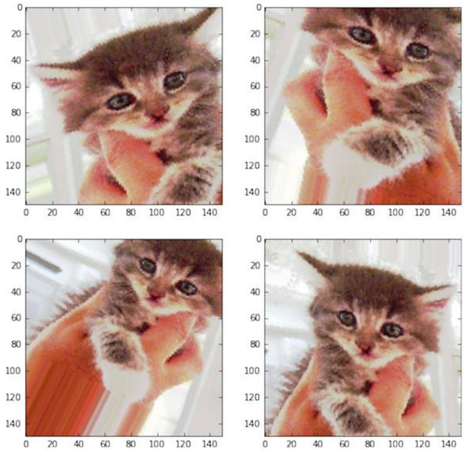
- More training data = better model
- Overfitting if not enough training data
  - Model generalizes badly
- With infinitely many, different training samples there would be no overfitting.

- How to increase the training samples for a given training data set?

- **Data Augmentation**
  - Transform input data slightly
  - Goal: During training each training sample is looked at only once

Data Augmentation to Fight Overfitting

- **No new information is generated**
- But information is **mashed-up in new ways**
- Overfitting cannot be prevented completely
  - Dropout before classification layer necessary

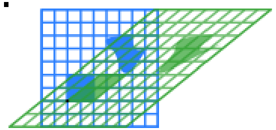


```
datagen = ImageDataGenerator(
          rotation_range=40,
          width_shift_range=0.2,
          height_shift_range=0.2,
          shear_range=0.2,
          zoom_range=0.2,
          horizontal_flip=True,
          fill_mode='nearest')
```

Image Manipulation Options

```
datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```

- `rotation_range` is a value in degrees (0–180), a range within which to randomly rotate pictures.
- `width_shift` and `height_shift` are ranges (as a fraction of total width or height) within which to randomly translate pictures vertically or horizontally.
- `shear_range` is for randomly applying shearing transformations.
- `zoom_range` is for randomly zooming inside pictures.
- `horizontal_flip` is for randomly flipping half the images horizontally;
- `fill_mode` is the strategy used for filling in newlycreated pixels, which can appear after a rotation or a width/height shift.

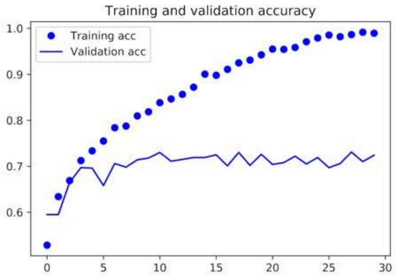


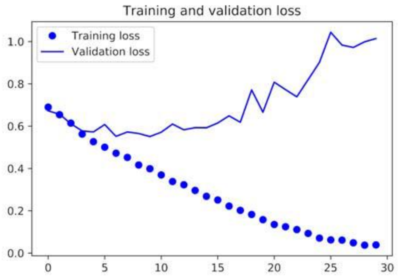
Data Augmentation: Example

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
        optimizer=optimizers.RMSprop(lr=1e-4),
        metrics=['acc'])
```

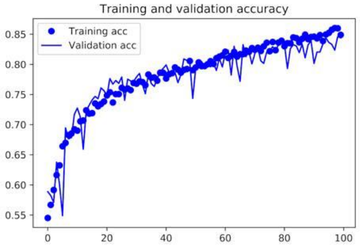
Data Augmentation: Example

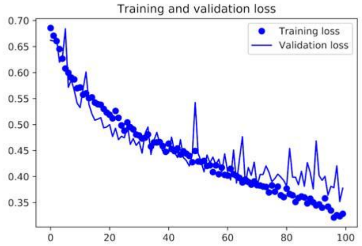
```
train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True))
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
        train_dir, target_size=(150, 150),
        batch_size=32, class_mode='binary')
validation_generator = test_datagen.flow_from_directory(
        validation_dir, target_size=(150, 150),
        batch_size=32, class_mode='binary')
history = model.fit_generator(
        train_generator, steps_per_epoch=100,
        epochs=100,
        validation_data=validation_generator,
        validation_steps=50)
```





Note that the validation and test data shouldn't be augmented!





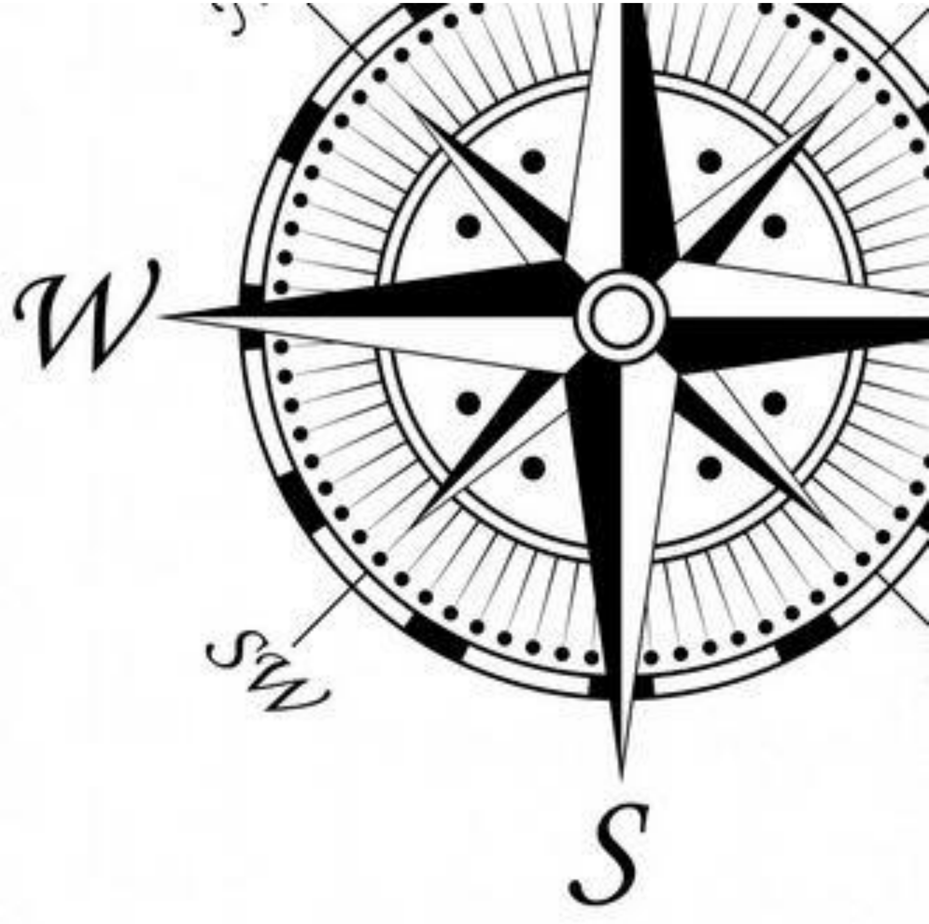
Data Augmentation

- How could data augmentation look like for sequential data?
  - How in particular for textual data?

Start 5 4 3 2 1 End

Topics Today

1. Convolutional Neural Networks
2. Over- and Underfitting
3. Data Augmentation
4. **Pretrained CNN**
5. 1D-CNN



Pretrained CNN

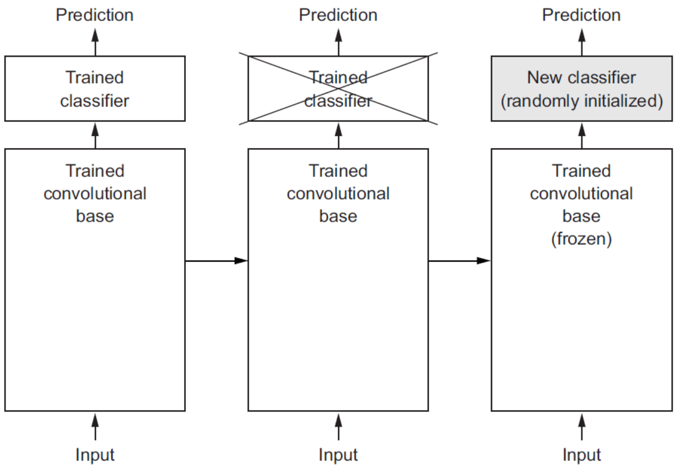
- If there is a **lack of training data**, you can use pretrained networks.
  - E.g. word embeddings trained on a large corpus
  - Or image representations trained for image classification on huge data sets
- Training data needs to be as **general** as possible
  - Better generalizable
  - Abstract representatoins of shapes, patterns (text: meaning)
    - Coverage of the complete visual/semantic space
- Pretrained nets provide representations
  - Visual and semantical
- **Serve as features/input** for a classifier
  - Domain can be different
    - Learnt: animal vs. plant
    - Deployed to recognize cats

  → **Transfer learning**

Pretrained Nets for Feature Generation

- Use of learned representations for new input data
- Only the **classification part** is newly trained
  - Not the feature extraction/convolutional basis
- You can use the **convolutional base** of another network
  - Generic features
- Classification part is very specific
  - Custom to classification problem
  - Not transferable to other classes
- The deeper in the net, the more complex/specific the features
  - Lines, circles
  - Cat eyes, dog eyes

If data differs significantly: Only reuse the lower layers

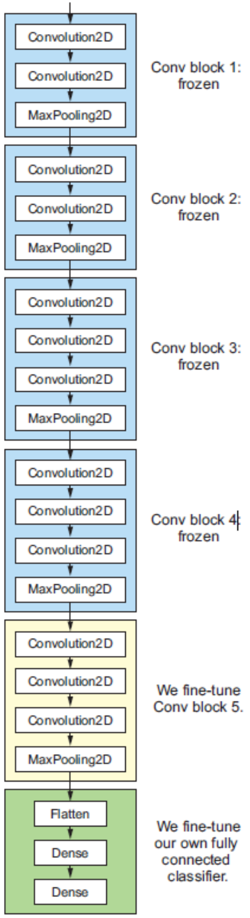


Fine-Tuning Pretrained Networks

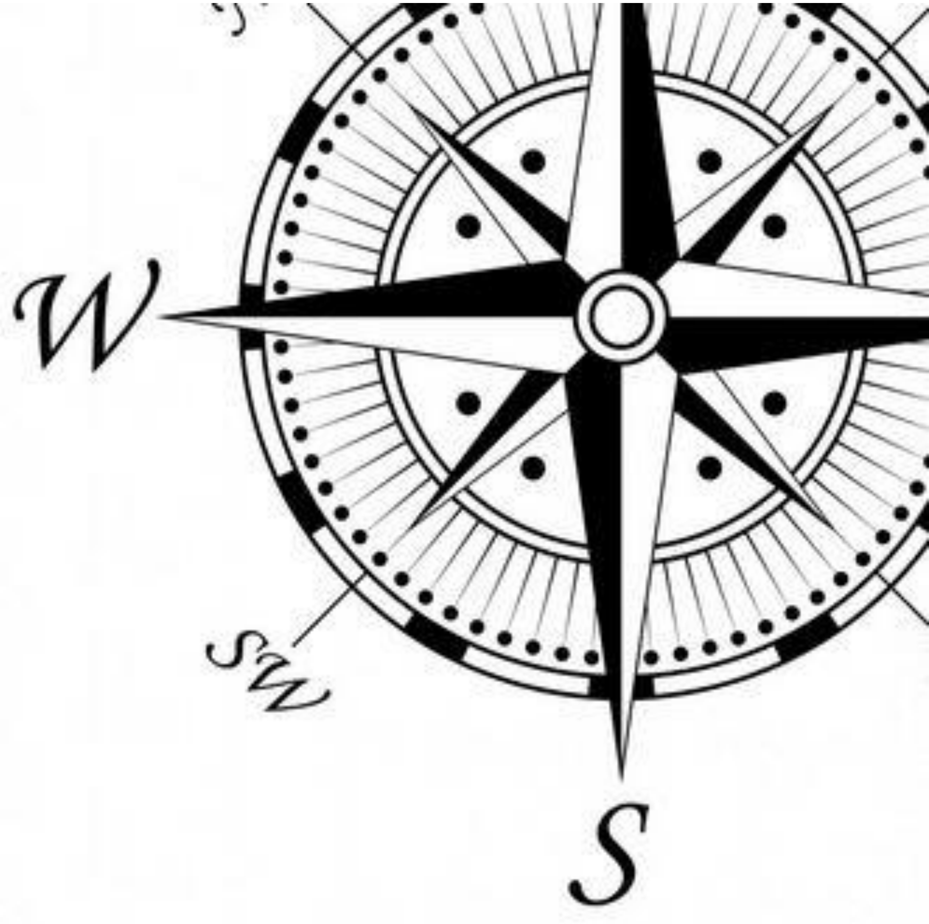
- Instead of training a whole new network (e.g. 15 million parameters) only train parts of it → **fine-tuning**
- First steps just like feature extraction
  1. Put own classification network on top of pretrained base network
  2. Freeze base network parameters
     - Will not be updated or further learnt
  3. Train the classifier network
  4. Unlock parameters of some layers for training
  5. Jointly train the unlocked base layers and the added classification layer

VGG16 Network

Important! Error signal otherwise too large!



Topics Today

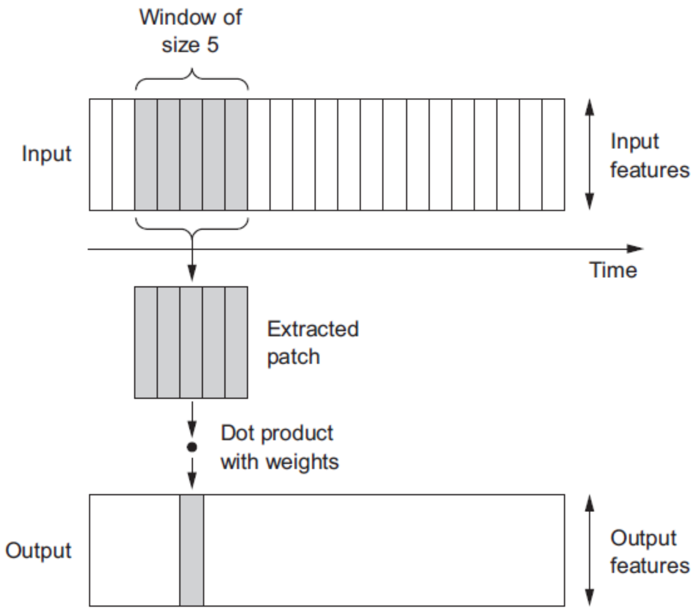


1. Convolutional Neural Networks
2. Over- and Underfitting
3. Data Augmentation
4. Pretrained CNN
5. **1D-CNN**

1D-Convolutions

- Traditionally:
  - Image data
  - Convolutions in 2D

- Works similar in 1D
  - E.g. text data
    - Depth not, e.g., 3 color channels
    - But word embeddings dimenstions
- 1D-Pooling
  - Analogous to 2D-pooling
  - A subsequence is replaced by its maximum (max-pooling)



Data Preprocessing

```
from keras.datasets import imdb
from keras.preprocessing import sequence
max_features = 10000
max_len = 500
print('Loading data...')
(x_train, y_train), (x_test, y_test) =
imdb.load_data(num_words=max_features)
print(len(x_train), 'train sequences')
print(len(x_test), 'test sequences')
print('Pad sequences (samples x time)')
x_train = sequence.pad_sequences(x_train, maxlen=max_len)
x_test = sequence.pad_sequences(x_test, maxlen=max_len)
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)
```

Tensors of shape: (samples, time, features)

Training and Evaluation

```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop
model = Sequential()
model.add(layers.Embedding(max_features, 128, input_length=max_len))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))
model.summary()
model.compile(optimizer=RMSprop(lr=1e-4),
        loss='binary_crossentropy',
        metrics=['acc'])
history = model.fit(x_train, y_train,
        epochs=10,
        batch_size=128,
        validation_split=0.2)
```

Depth of output

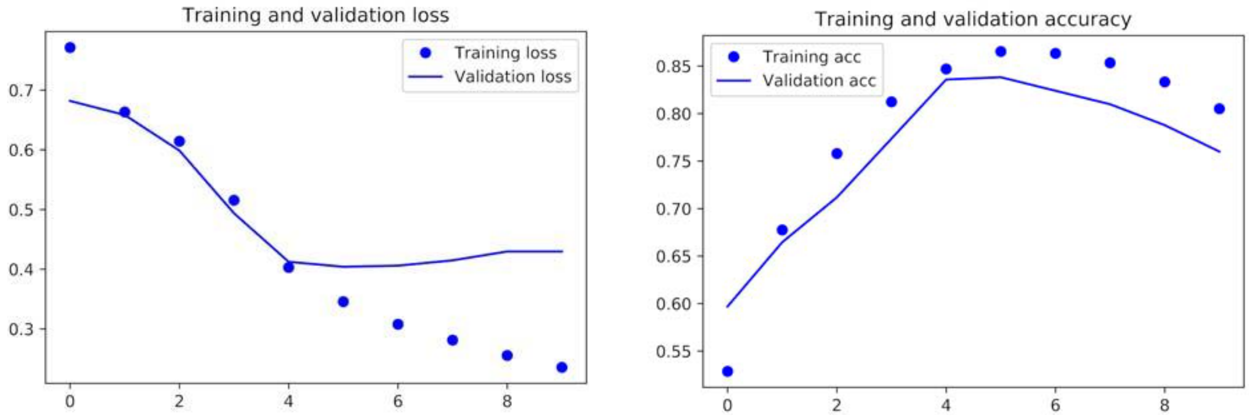
Large window sizes possible

Alternative: `layers.Flatten()`

Output layer: classification or regression

Evaluation

- A little worse than LSTM
  - But significantly faster (on CPU and GPU)



CNNs for Sequencial Data

- What‘s the advantage of large window sizes for 1D-CNNs?
  - How large is too large?

- How does the concept of spacial hierarchies translate to text data?

Start 5 4 3 2 1 End

Learning Goals for this Chapter

- Understand the basics of convnets
  - feature maps
  - convolutions
  - max-pooling
- Augment training data
- Fight over- and underfitting
- Employ pretrained networks
- Fine-tune convnets

- Relevant chapters:
  - P5
  - S11 (2019) https://www.youtube.com/watch?v=EAJoRA0KX7I