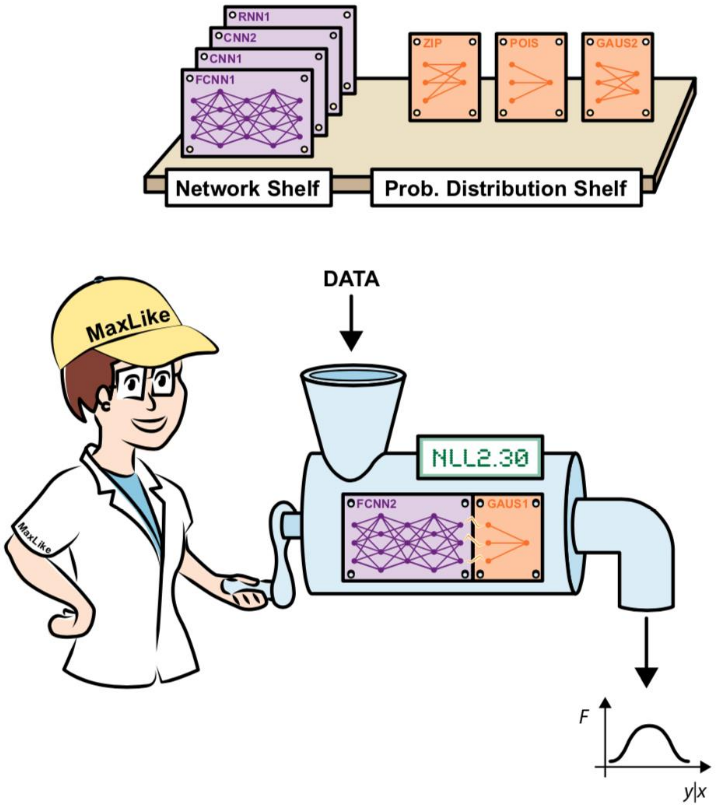
WBL Deep Learning:: Lecture 3

Beate Sick, Oliver Dürr

Convolutional Neural Networks cntd.

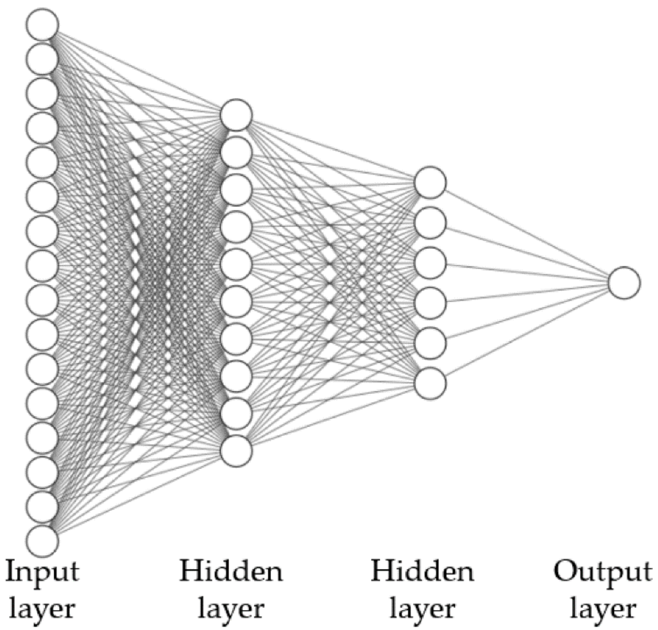
Topics of today

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

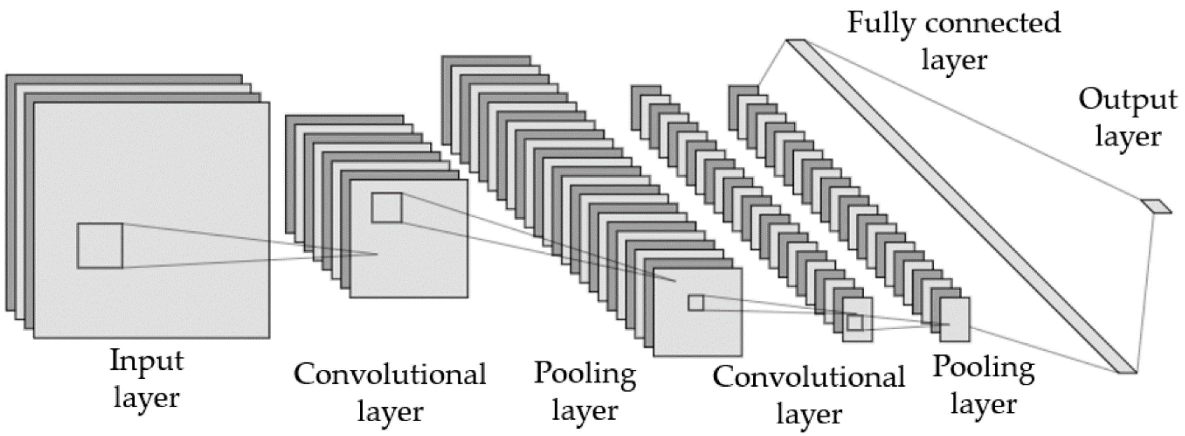


Recall: fully connected NN and convolutional NN

fcNN



CNN

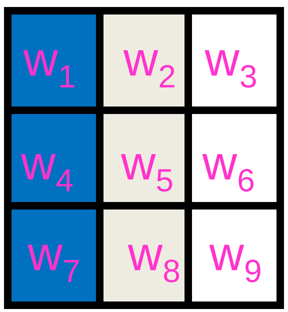


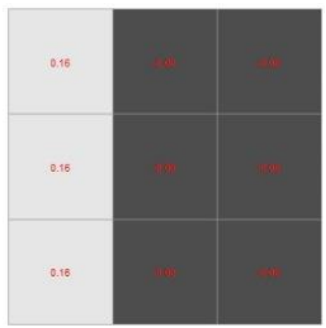
CNN ingredient I: convolution

In a locally connected network the calculation rule

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

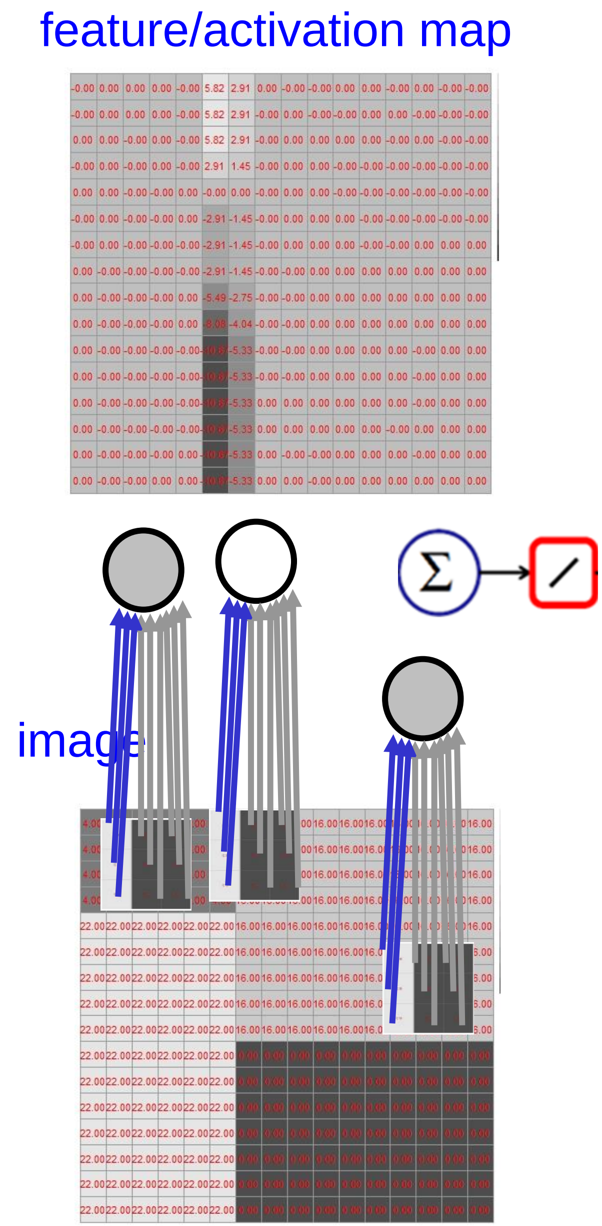
Pixel values in a small image patch are element-wise multilied with weights of a small filter/kernel:



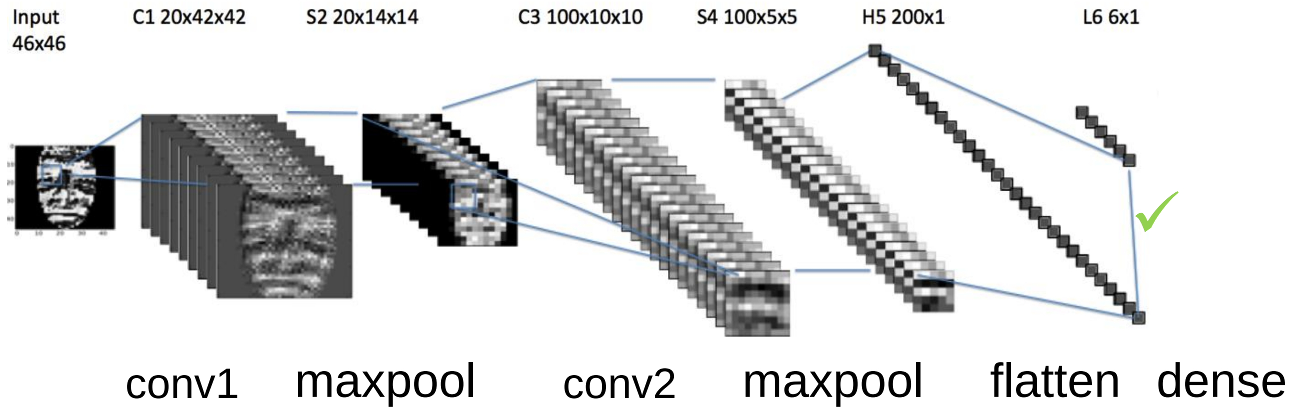


The filter is applied at each position of the image and it can be shown that the result is maximal if the image pattern corresponds to the weight pattern.

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



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

→

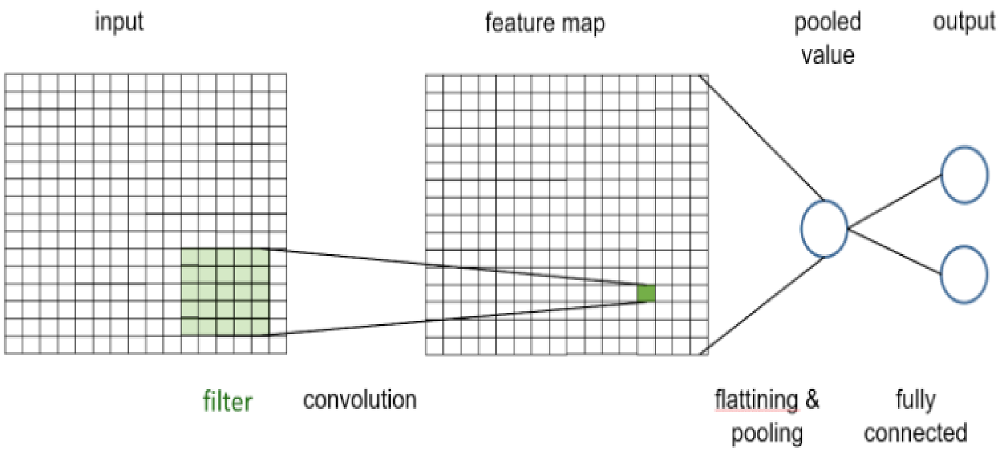
2x2

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"

Building a very simple CNN with keras

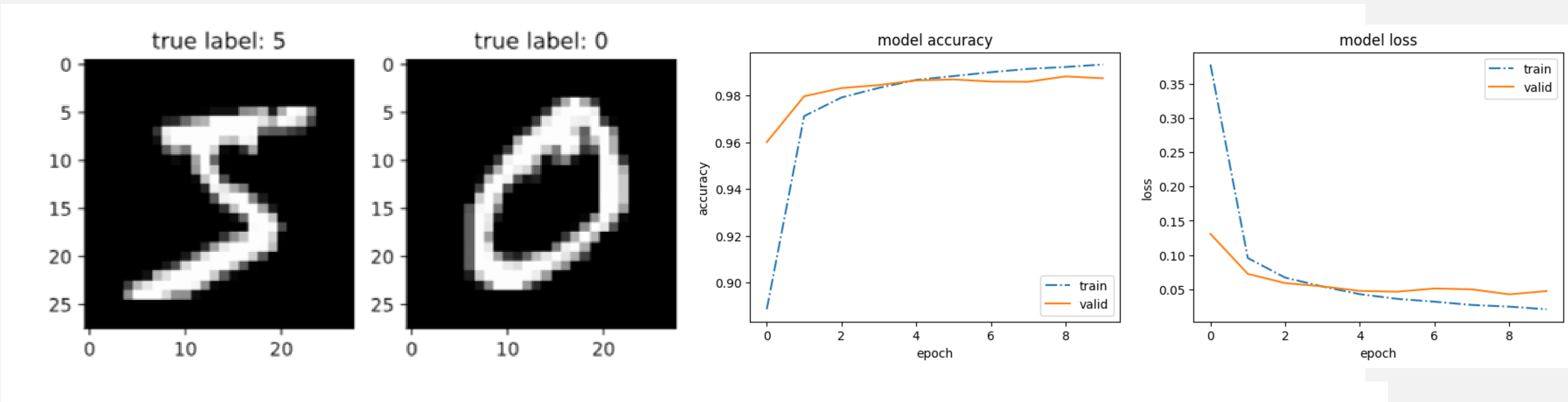


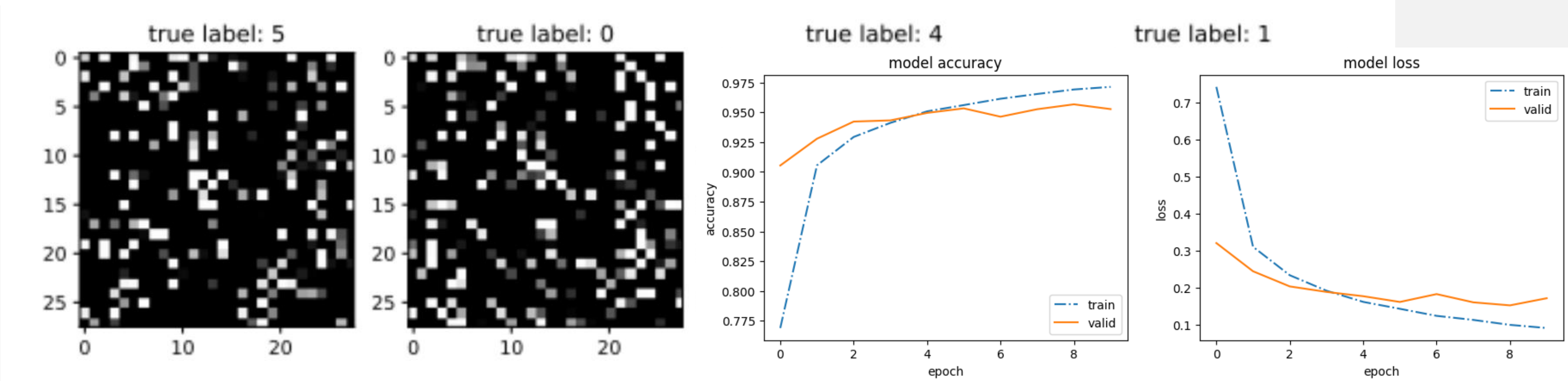

```
model <- keras_model_sequential()
model %>%
  layer_conv_2d(filters=,
                kernel_size = c(5,5),
                padding = 'same',
...              input_shape = ...,
                activation = 'linear') %>%
  # take the max over all values in the activation map
  layer_max_pooling_2d(pool_size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

Fill the gaps!

Open NB in: https://github.com/tensorchiefs/dl_course_2020/blob/master/notebooks/05_cnn_edge_lover.ipynb

Shuffling reduces performance of CNNs





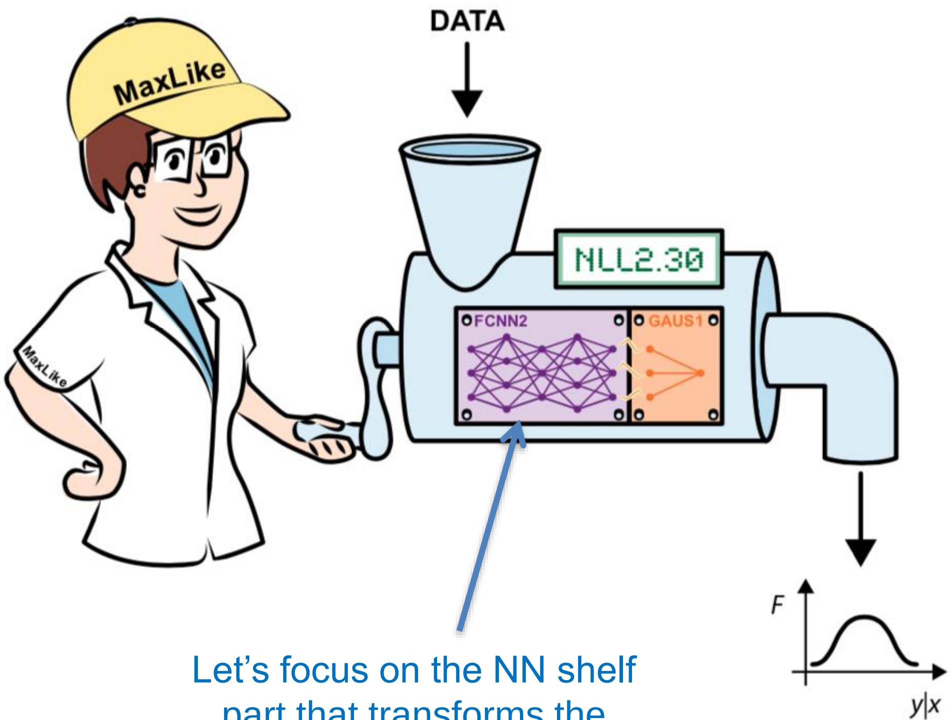
→ The performance of a CNN is better on original than on shuffled images

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

What does the CNN look at?

Looking in the feature extracting part

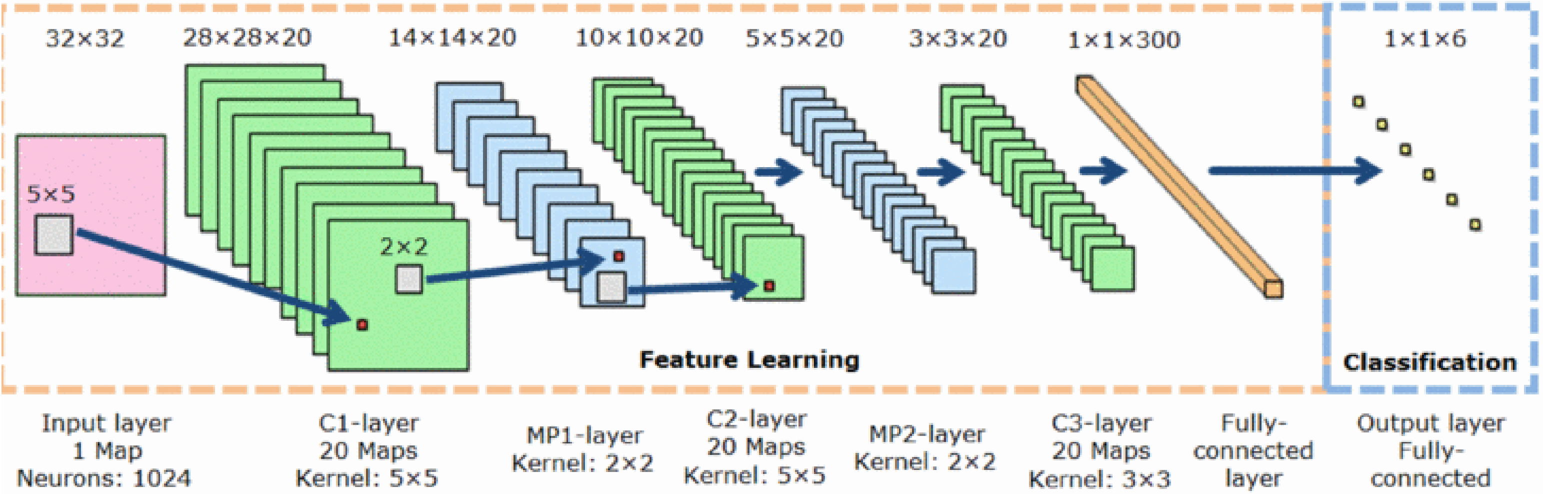


Let's focus on the NN shelf part that transforms the raw data into features
(which are used in the NN head to predict the parameters of the CPD)

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”.

Receptive field in original input 5x5 6x6 14*14 16x16 24x24 32x32



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

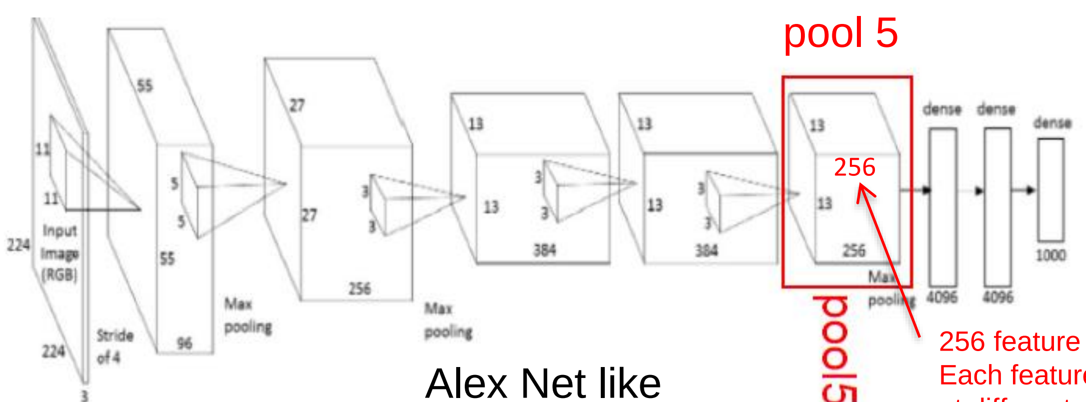
Code to determine size of receptive field: http://stackoverflow.com/questions/35582521/how-to-calculate-receptive-field-size

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.

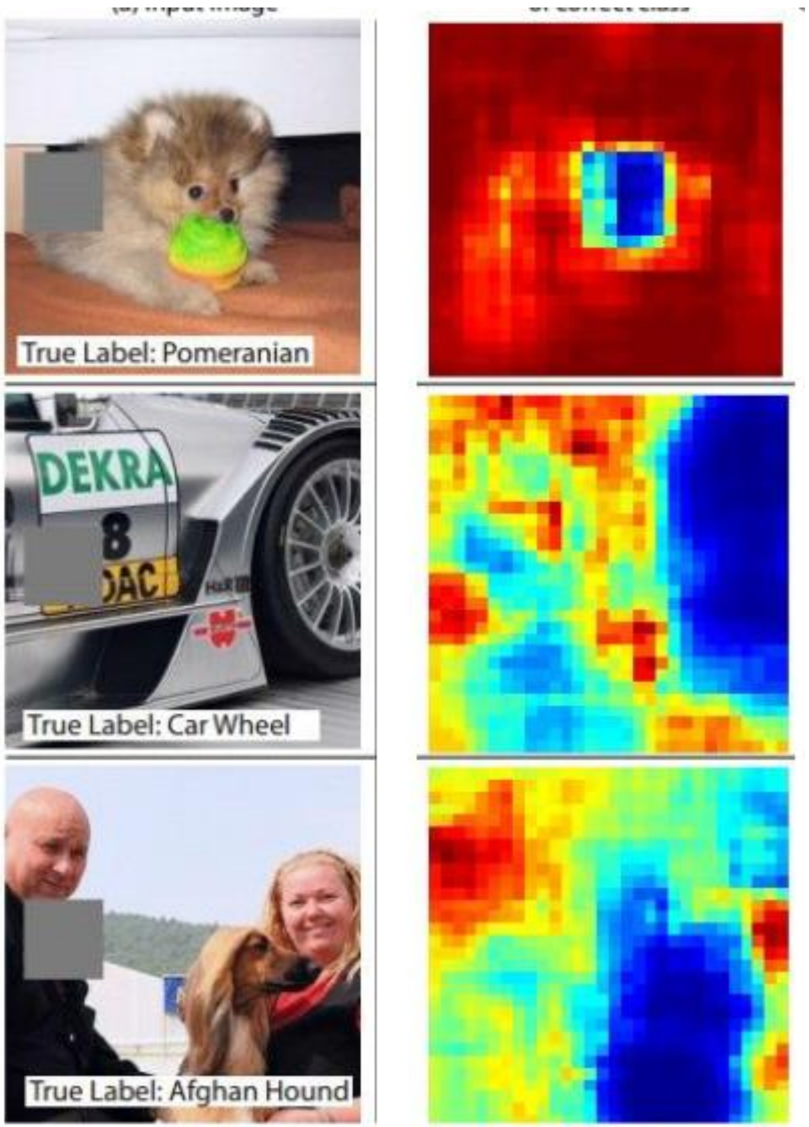
What kind of image (patches) excites a certain neuron corresponding to a large activation in a feature map?

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…

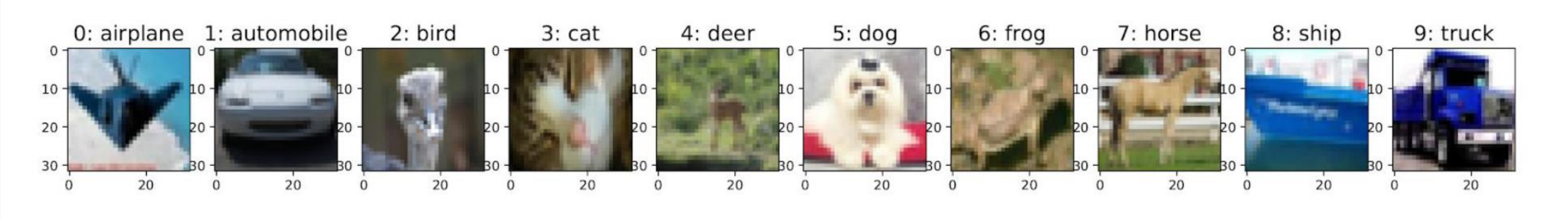
Occlusion experiments [Zeiler & Fergus 2013]

image credit: cs231n

Tricks of the Trade

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

Exercise: Develop a CNN for cifar10 data



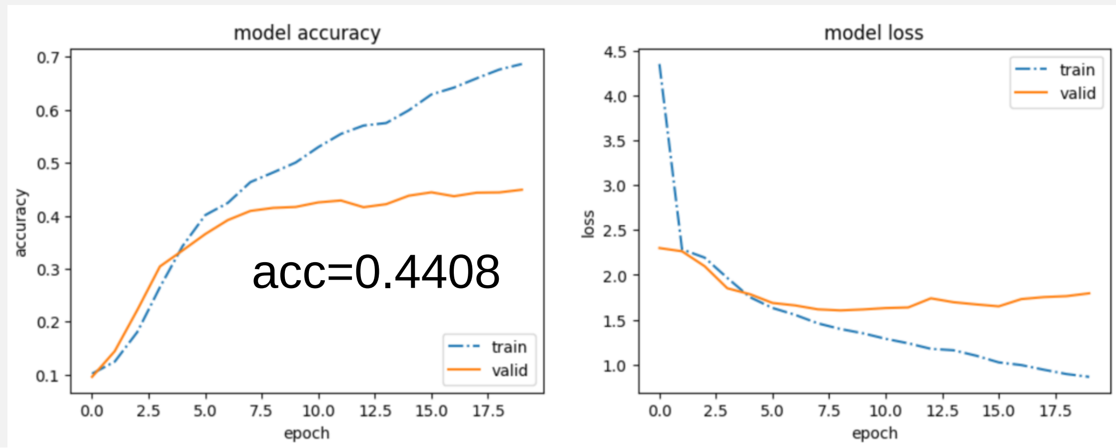
Develop a CNN to classify cifar10 images (we have 10 classes)

Investigate the impact of standardizing the data on the performance

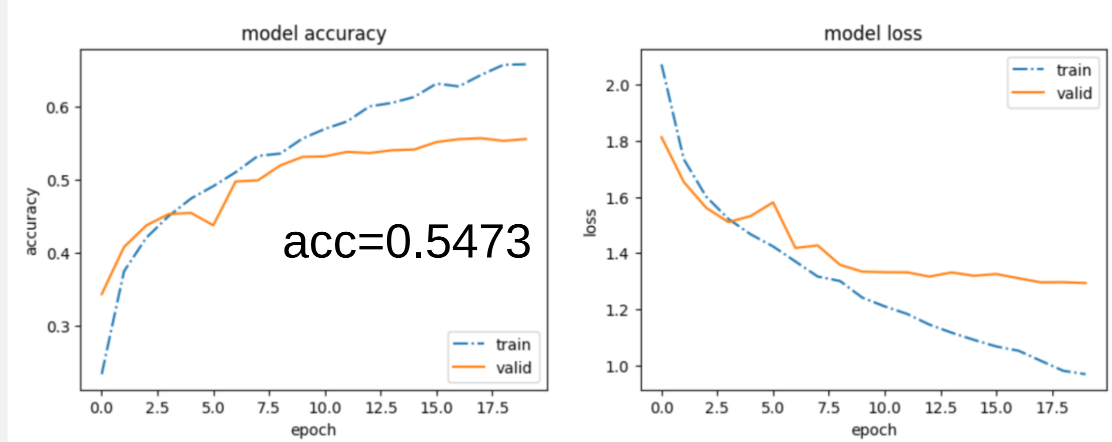
Take-home messages from the homework

- 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

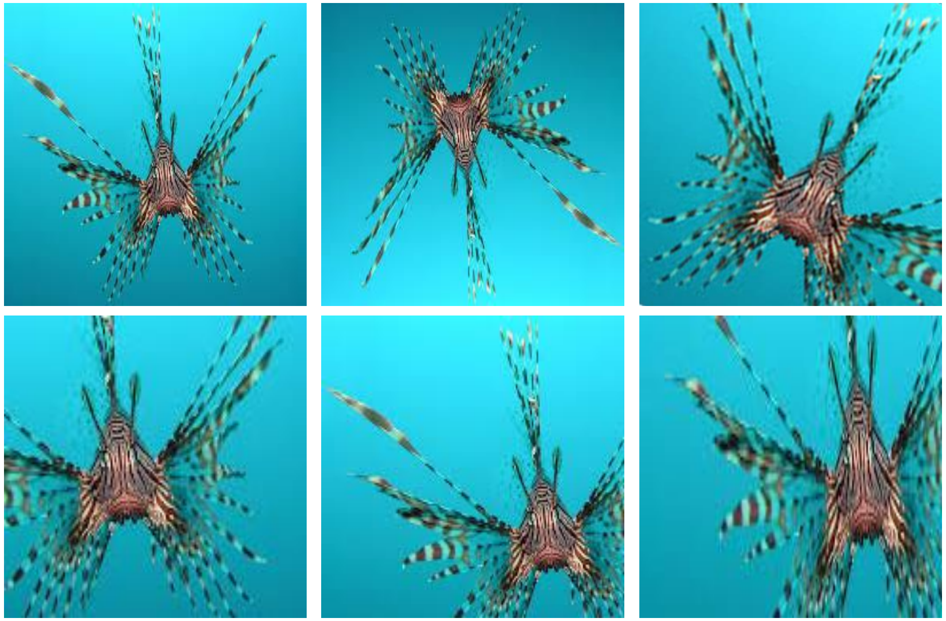


Data augmentation

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
- ….

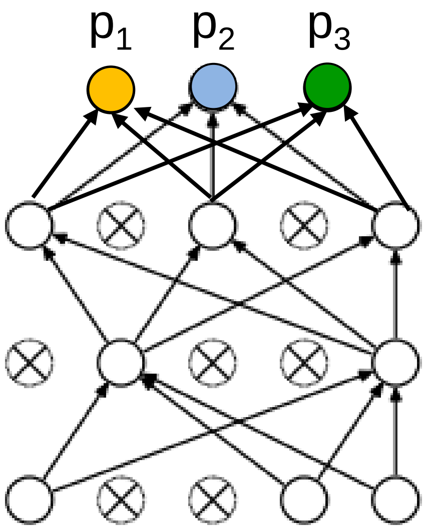


Data augmentation in Keras:

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

Dropout during training

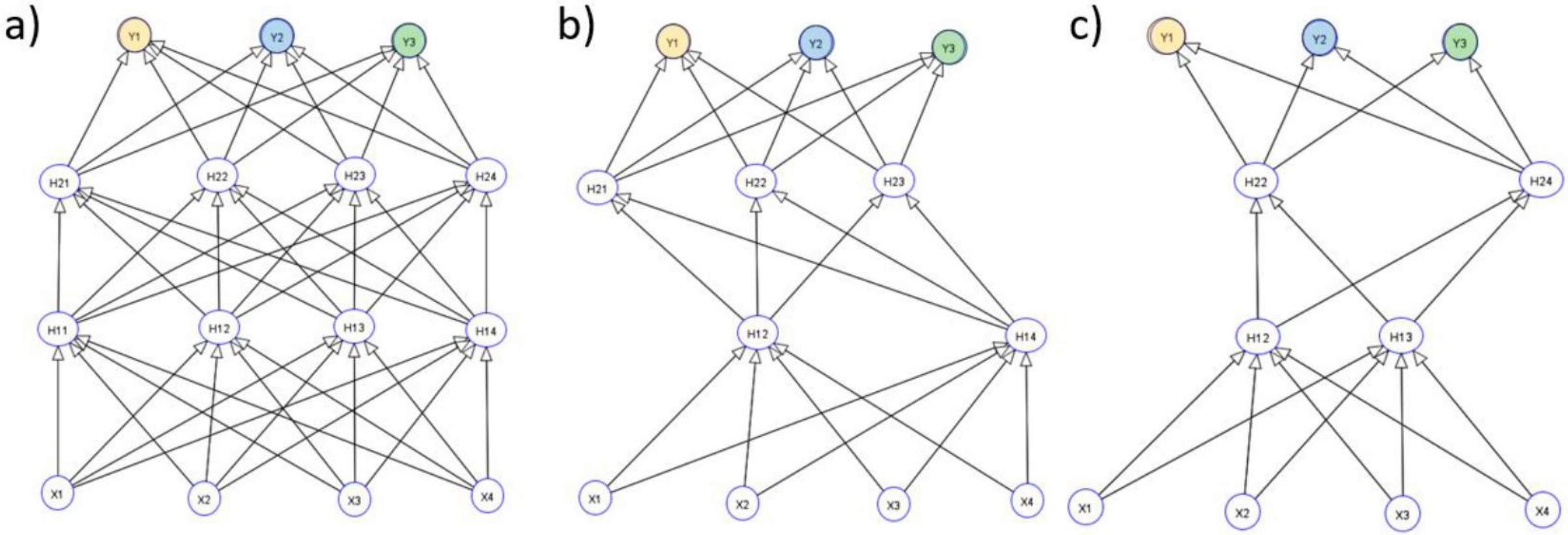
Dropout helps to fight overfitting



Using dropout during training implies:

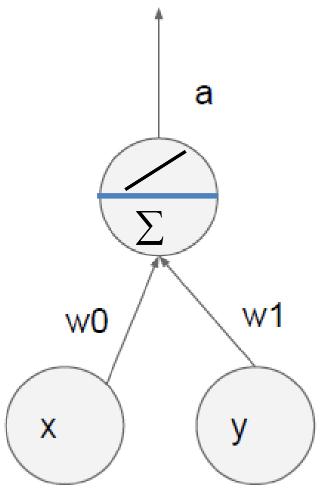
- In each training step only weights to not-dropped units are updated → we train a sparse sub-model NN
- For predictions with the trained NN we freeze the weights corresponding to averaging over the ensemble of trained models we should be able to "reduce noise", "overfitting"

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



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: **a = w0*x + w1*y**

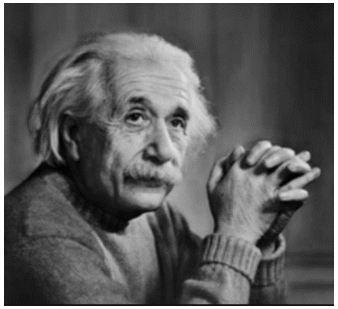
during training with dropout probability 0.5:

E[a] = ¼ * (w0*0 + w1*0 +
w0*0 + w1*y +
w0*x + w1*0 +
w0*x + w1*y)
= ¼ * (2 w0*x + 2 w1*y)
= ½ * (w0*x + w1*y)

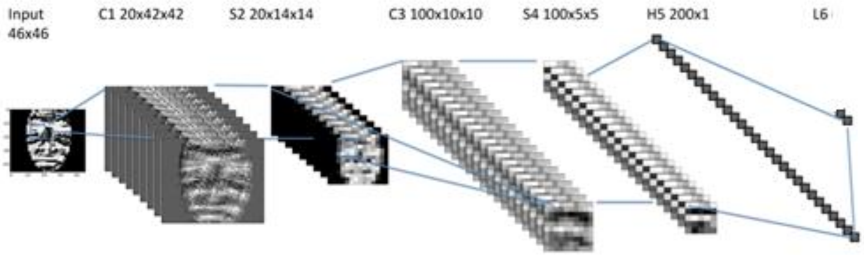
=> To get same expected output in training and test time, we reduce the weights during test time by multiplying them by the dropout probability p=0.5

Another intution: Why "dropout" can be a good idea

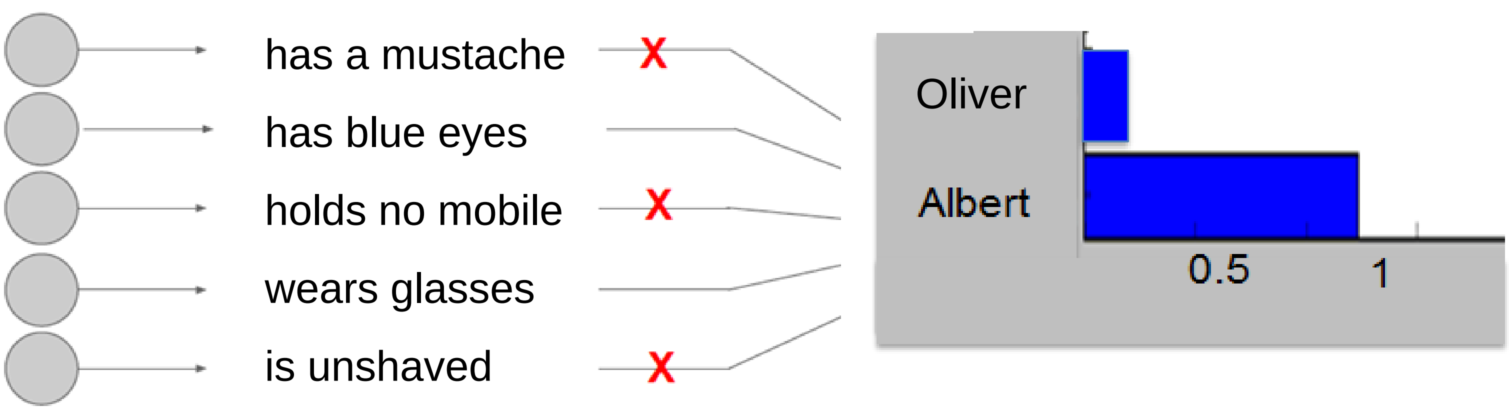
The training data consists of many different pictures of Oliver and Einstein



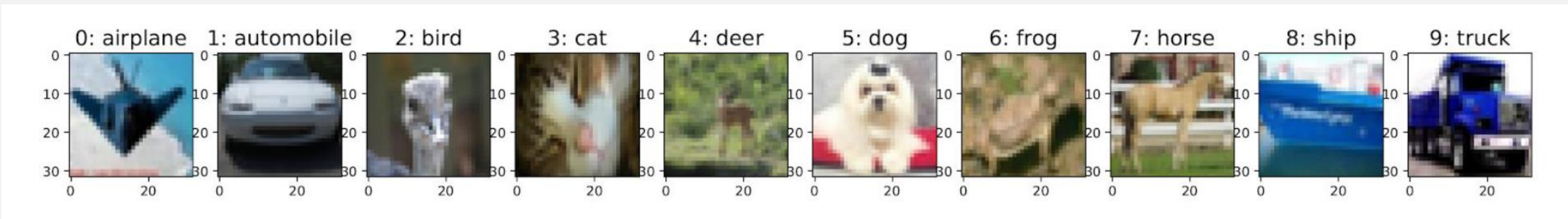
We need a huge number of neurons to extract good features which help to distinguish Oliver from Einstein



Dropout forces the network to learn redundant and independent features

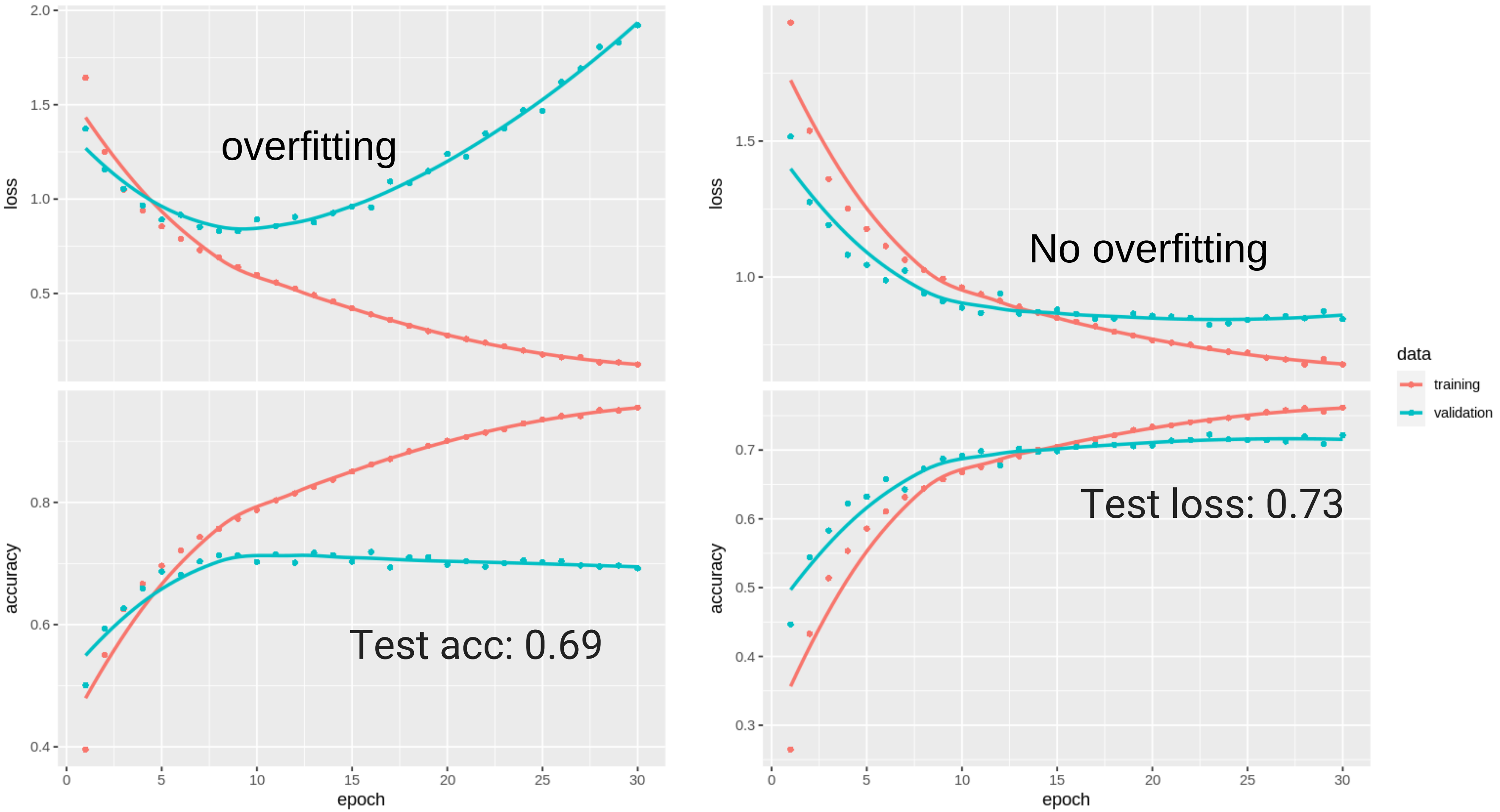


Experiment: Can dropout improve performance?



cnn from scratch
cnn from scratch with dropout

Dropout fights overfitting in a CIFAR10 CNN



Challenge winning CNN architectures

Review of ImageNet winning CNN architectures

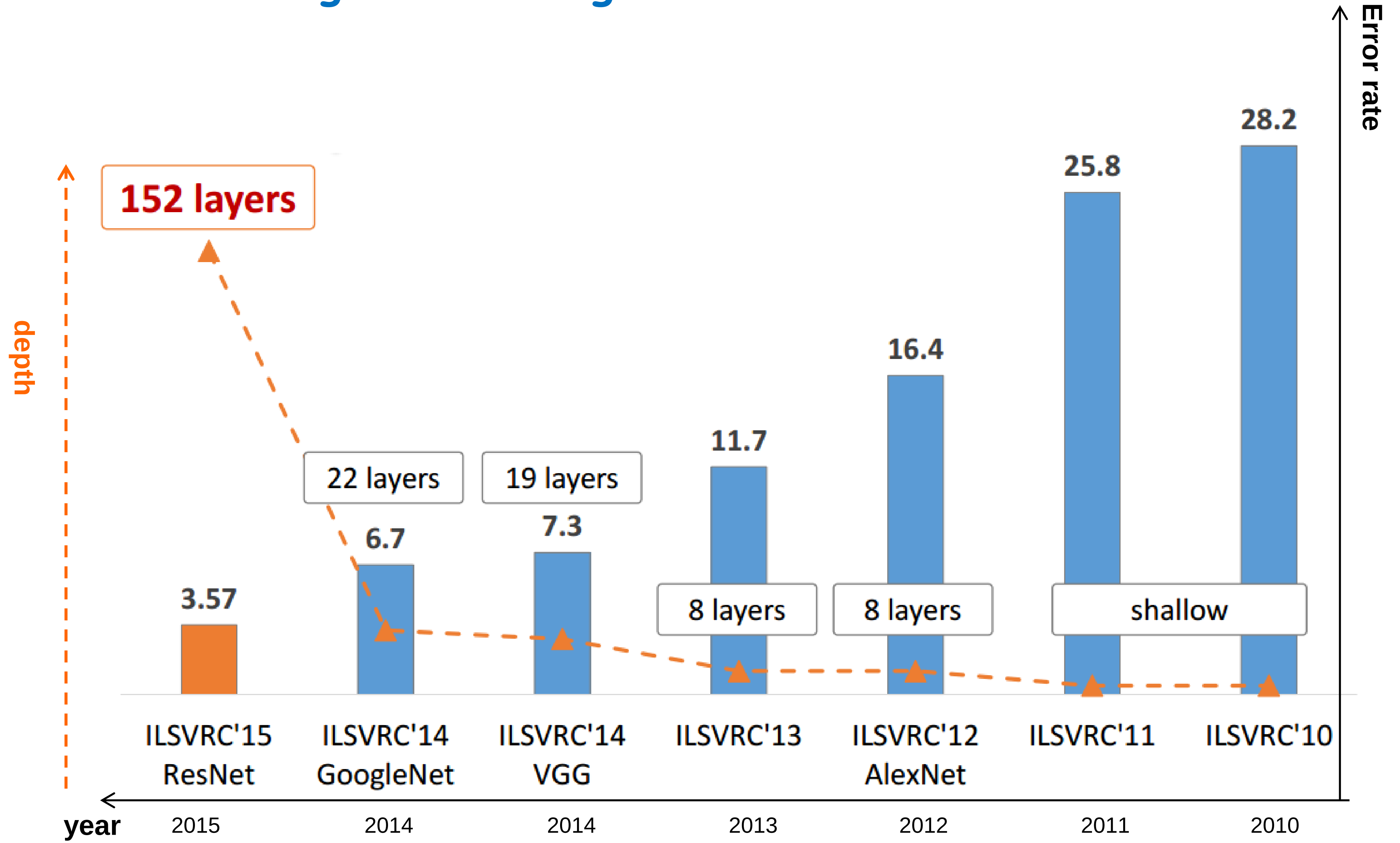


Image credits (modified): K.He, X.Zhang, S.Ren, J.Sun. “Deep Residual Learning for Image Recognition”. arXiv 2015

LeNet-5 1998: first CNN for ZIP code recognition

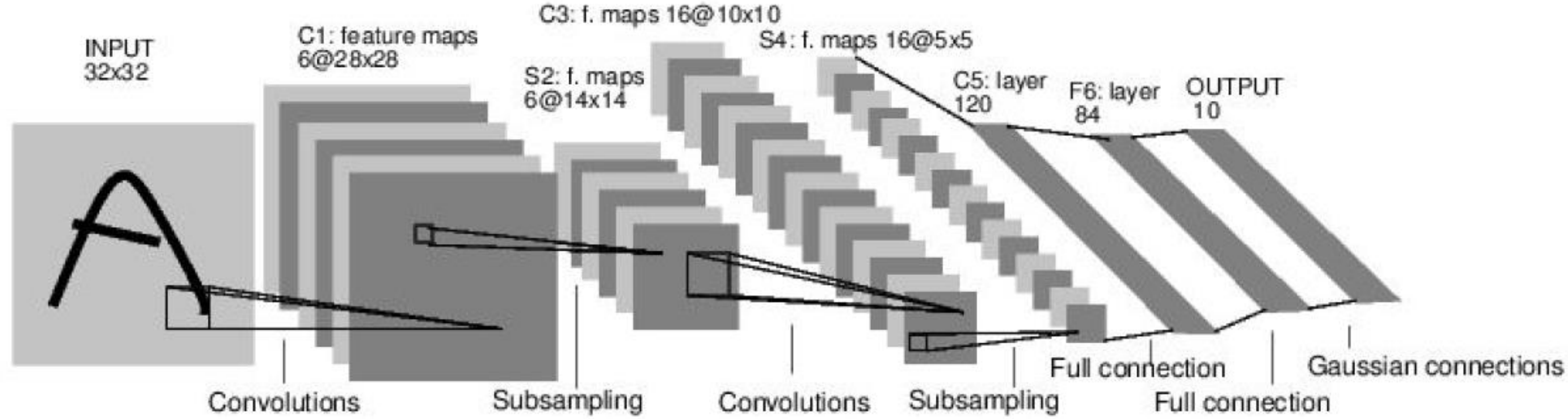


Image credits: http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

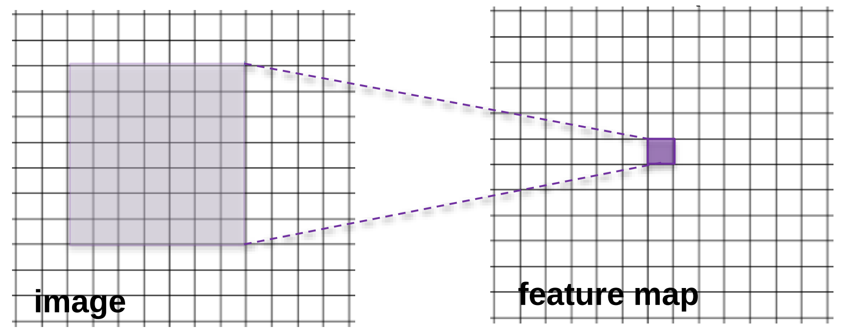
[Demo von 1993](#)

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

The trend in modern CNN architectures goes to small filters

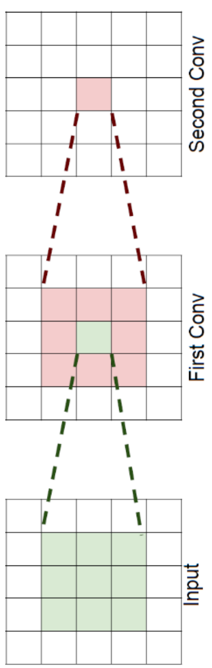
Why do modern architectures use very small filters?
Determine the receptive field in the following situation:

Answer1): 7x7



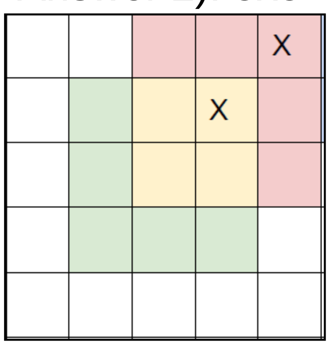
1) Suppose we have **one** 7x7 conv layers (stride 1)

49 weights



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

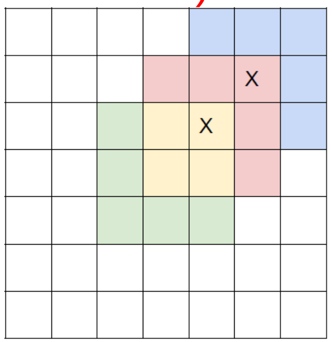
Answer 2): 5x5



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

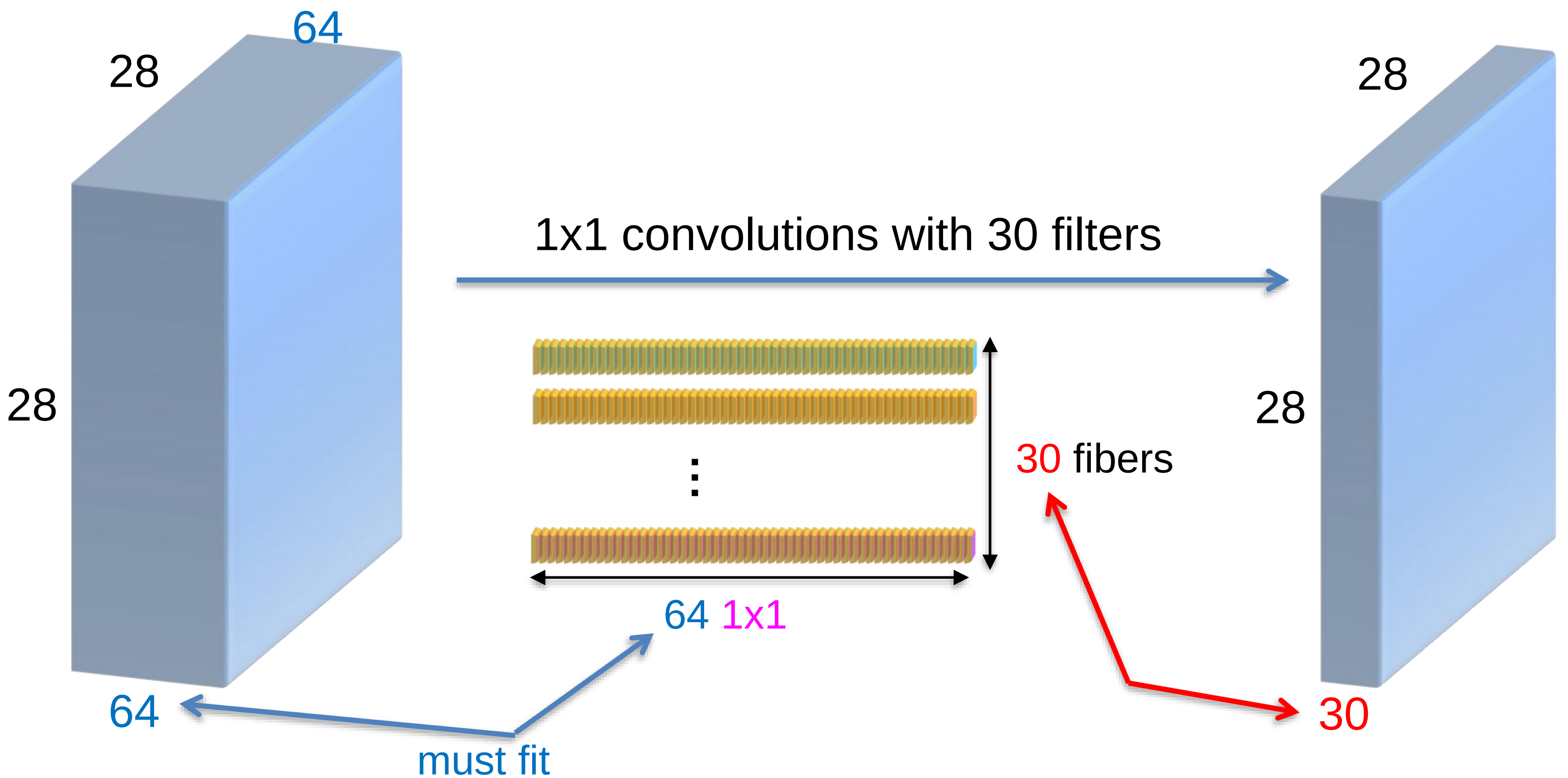
3*9=27 weights

Answer 3): 7x7



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

Go to the extreme: What is about filter size 1?
1x1 convolutions act only in depth dimension



1x1 convolution act along a “fiber” in depth dimension across the channels.

→ efficient way to reduce/change the depth dimension

→ simultaneously introduce more non-linearity

"Oxford Net" or "VGG Net" 2014 2nd place

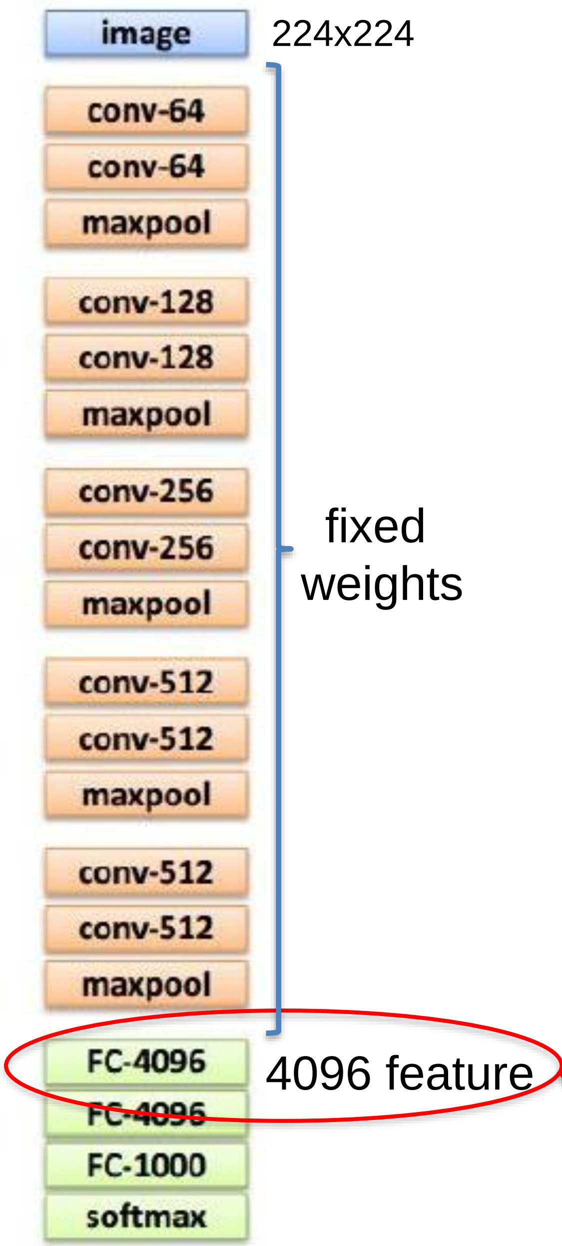
- 2nd place in the imageNet challenge
- More traditional, easier to train
- More weights than GoogLeNet
- Small pooling
- Stacked 3x3 convolutions before maxpooling

  -> large receptive field
- no strides (stride 1)
- ReLU after conv. and FC (batchnorm was not used)
- Pre-trainined model is available

http://arxiv.org/abs/1409.1556

What to do in case of limited data?

Use pre-trained CNNs for feature generation



- Load a pre-trained CNN – e.g.g VGG16
  developed by Karen Simonyan and Andrew Zisserman in 2014
- Images are resized to required size
- Rescaling of the pixel values to “VGG range”
- Do a forward pass and fetched features that are used as CNN representations, dump these features into a file on disk
- Use these CNN features as input to a simple classifier – e.g. fc NN, RF, SVM …
  (here it is easily possible to adapt to the new number of class labels)

Number features depends on input shape

Fetch this CNN feature vector for each image

Use pre-trained CNNs for transfer learning

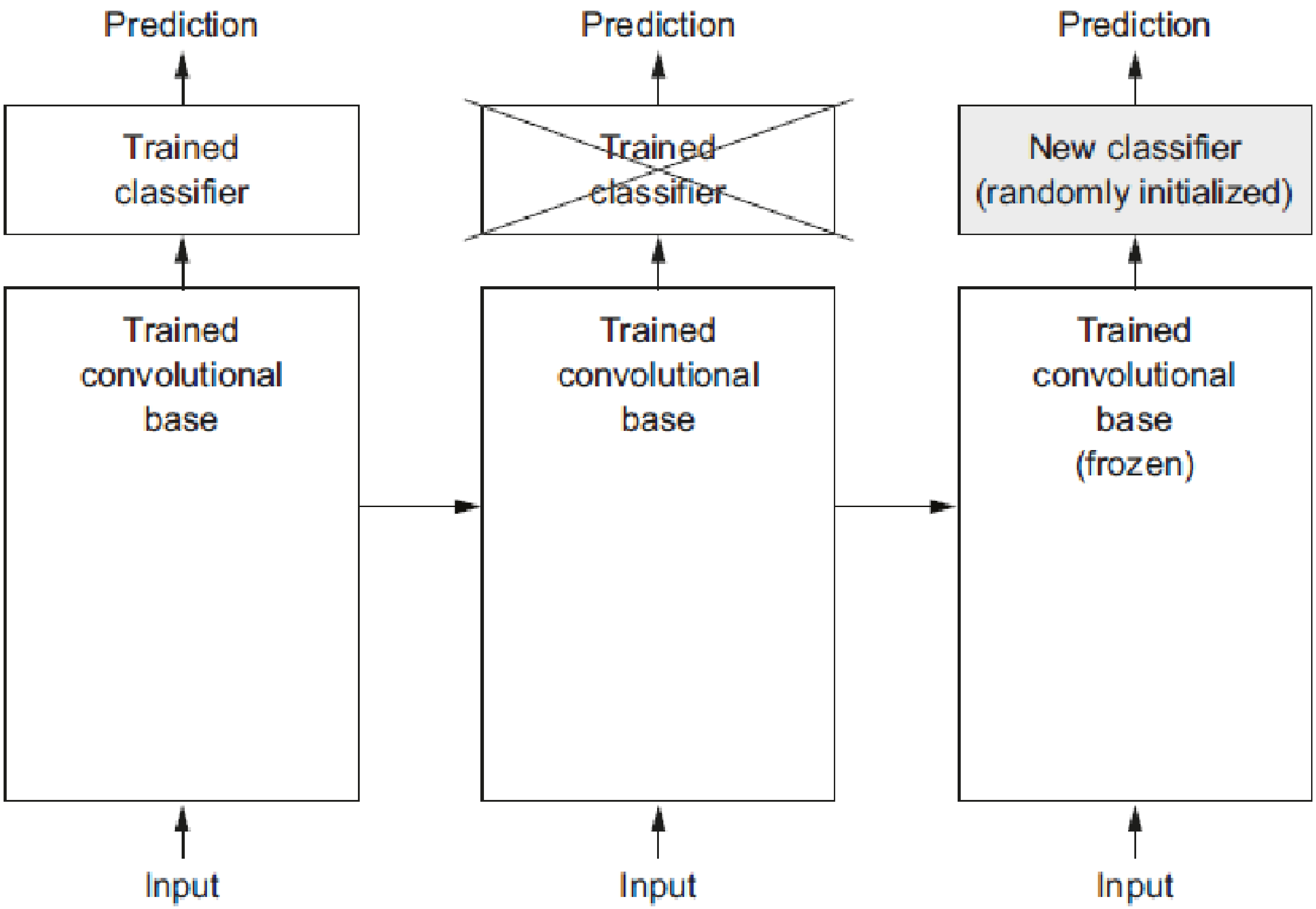
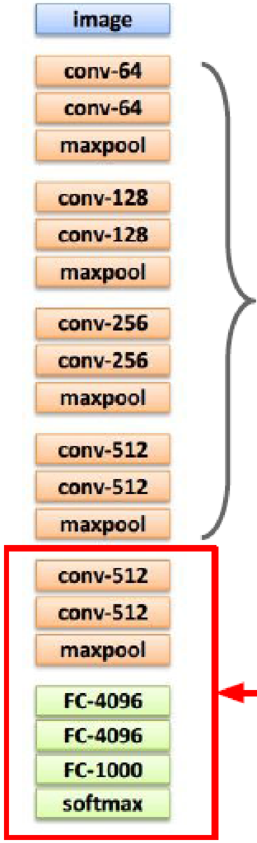


Figure 5.12 Swapping classifiers while keeping the same convolutional base

See DL with R book, chapter 5.4

Transfer learning beyond using off-shelf CNN feature

e.g. medium data set (<1M images)



finetuning

more data = retrain more of the network (or all of it)

Freeze these (learning rate =0)

tip: use only ~1/10th of the original learning rate in finetuning top layer, and ~1/100th on intermediate layers

Train this

The strategy for fine-tuning depends on the size of the data set and the type of images:

	Similar task (to imageNet challenge)	**Very different task** (to imageNet challenge)
little data	Extract CNN representation of one top fc layer and use these features to train an external classifier	You are in trouble - try to extract CNN representations from different stages and use them as input to new classifier
lots of data	Fine-tune a few layers including few convolutional layers	Fine-tune a large number of layers

Hint: first retrain only fully connected layer, only then add convolutional layers for fine-tuning.

Slide credits (modified): cs231n

Use pre-trained CNNs for transfer learning

Let's instantiate the VGG16 model.

Listing 5.16 Instantiating the VGG16 convolutional base

```
library(keras)

conv_base <- application_vgg16(
  weights = "imagenet",
  include_top = FALSE,
  input_shape = c(150, 150, 3)
)
```

Listing 5.20 Adding a densely connected classifier on top of the convolutional

```
model <- keras_model_sequential() %>%
  conv_base %>%
  layer_flatten() %>%
  layer_dense(units = 256, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
```

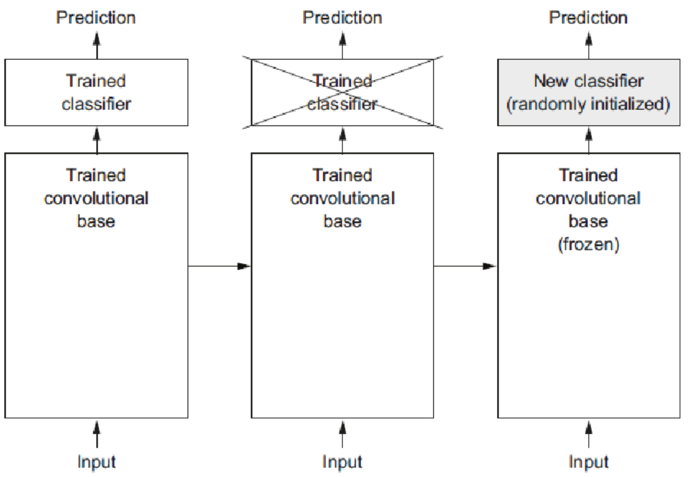
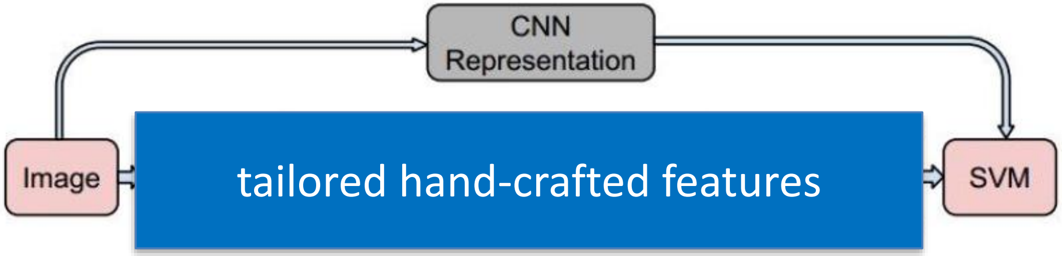
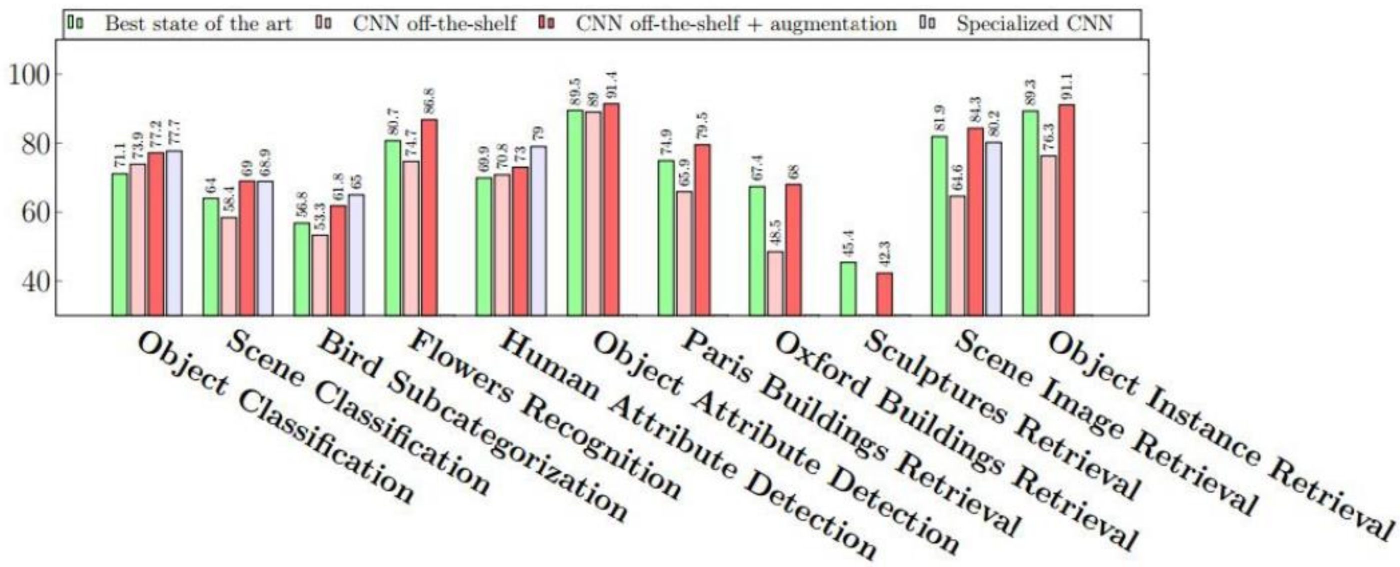


Figure 5.12 Swapping classifiers while keeping the same convolutional base

For an example in python, see also
https://github.com/tensorchiefs/dl_course_2018/blob/master/notebooks/11_8_faces_fine_tuning_solution.ipynb

Performance of off-the-shelf CNN features when compared to tailored hand-crafted features

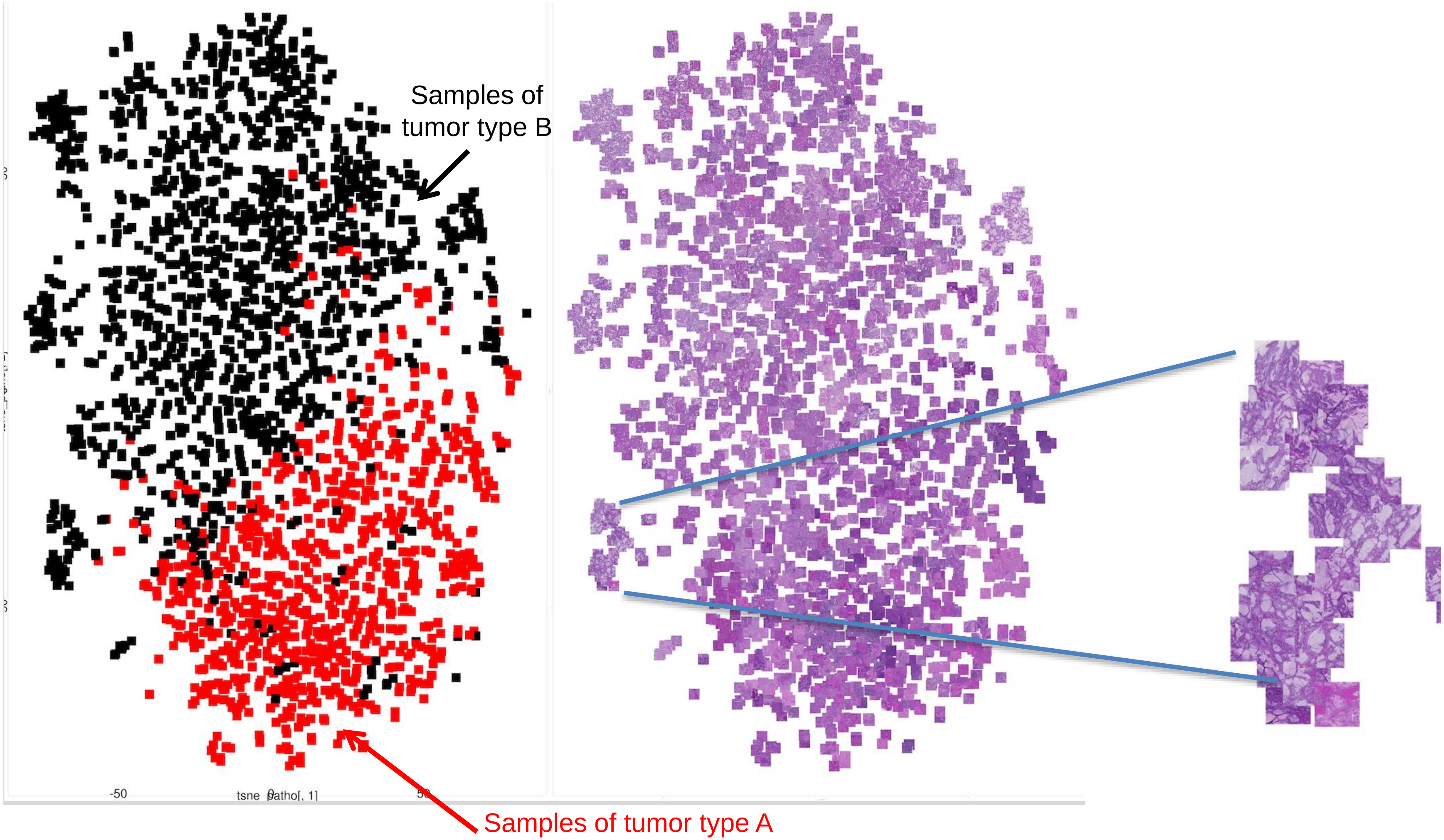




"Astonishingly, we report consistent superior results compared to the highly tuned state-of-the-art systems in all the visual classification tasks on various datasets."

CNN Features off-the-shelf: an Astounding Baseline for Recognition [Razavian et al, 2014]

Use features from a pretrained VGG as input to t-SNE



→ Different tissue types cluster together in t-SNE: we could use knn as classifier
→ VGG features even work on images that are far away from the 1000 imageNet classes

Exercise: Transfer learning with CIFAR10 data

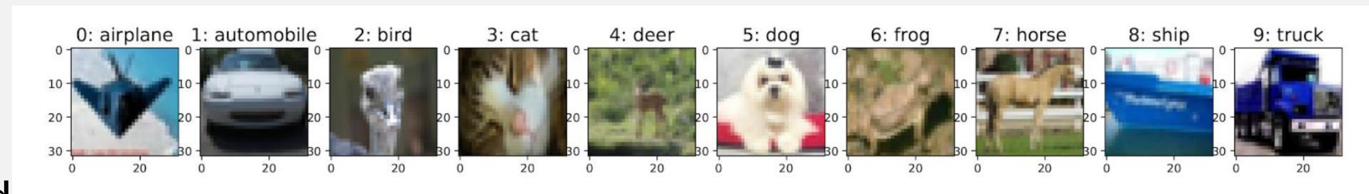
224x224

fixed weights

Depends on input shape

4096 feature

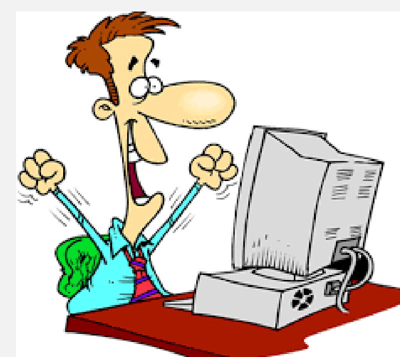
https://github/tensorchiefs/dl_course_2020/blob/master/notebooks/08_cifar10_tricks.ipynb



rf on vgg features

transfer learning on vgg features

cnn from scratch with data augmentation



Links to colab notebooks for hands-on exercises

- **Simple CNN (03_nb): Art Lover**
  https://github.com/tensorchiefs/dl_rcourse_2022/blob/main/notebooks/03_nb_ch02_03.ipynb

- **CNN (04_nb): Dropout CIFAR 10**
  https://github.com/tensorchiefs/dl_rcourse_2022/blob/main/notebooks/04_nb_ch02_03.ipynb

- **Transfer learning (05_nb): Cifar 10 with pretrained VGG CNN**
  https://github.com/tensorchiefs/dl_rcourse_2022/blob/main/notebooks/05_nb_ch02_03.ipynb

Summary on CNN specialities

- **Trick of the trade to get deep CNN trained:**
  - Stack enough layers and use small kernels (3x3)
  - Use dropout during training to avoid overfitting
  - Standardize your input data

- **DL with few labeled data:**
  - use data augmentation to enlarge the training data
  - transfer learning
    - use pretrained CNN with frozen convolutional part for finetuning
  - use pretrained CNN (e.g. VGG) as feature extractor
  - use VGG features as input for any classifier, e.g. RF
  - use VGG features as input for a t-SNE map
  - use a t-SNE map for generating labels to enlarge the training data