Machine Intelligence:: Deep Learning

Beate Sick, Lilach Goren, Pascal Bühler

Institut für Datenanalyse und Prozessdesign
Zürcher Hochschule für Angewandte Wissenschaften

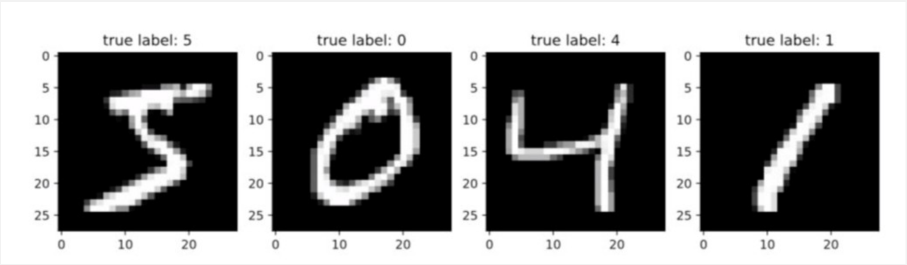
Most of the slides were developed by Oliver Dürr & Beate Sick

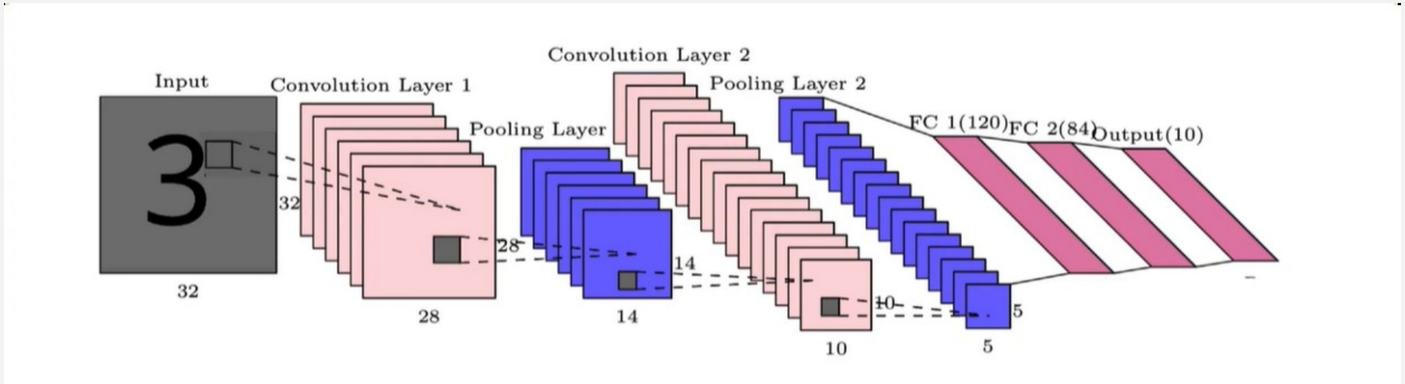
Outline of the DL Module (tentative)

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-archictecture
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:
MNIST with CNN





Use a CNN to classify MNIST data to 10 possible classes

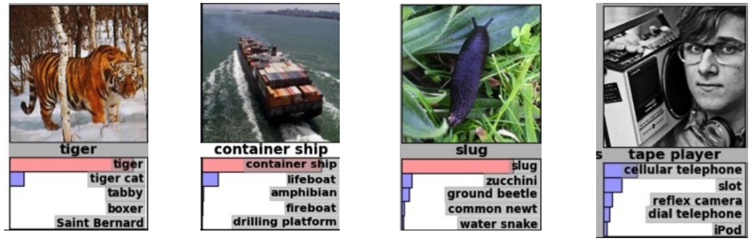
06_cnn_mnist_shuffled_keras_torch.ipynb <- first part with the unshuffled cnn

Model	Loss	Accuracy	# Params
FcNN	0.1	0.97	84060
CNN	0.05	0.98	35962

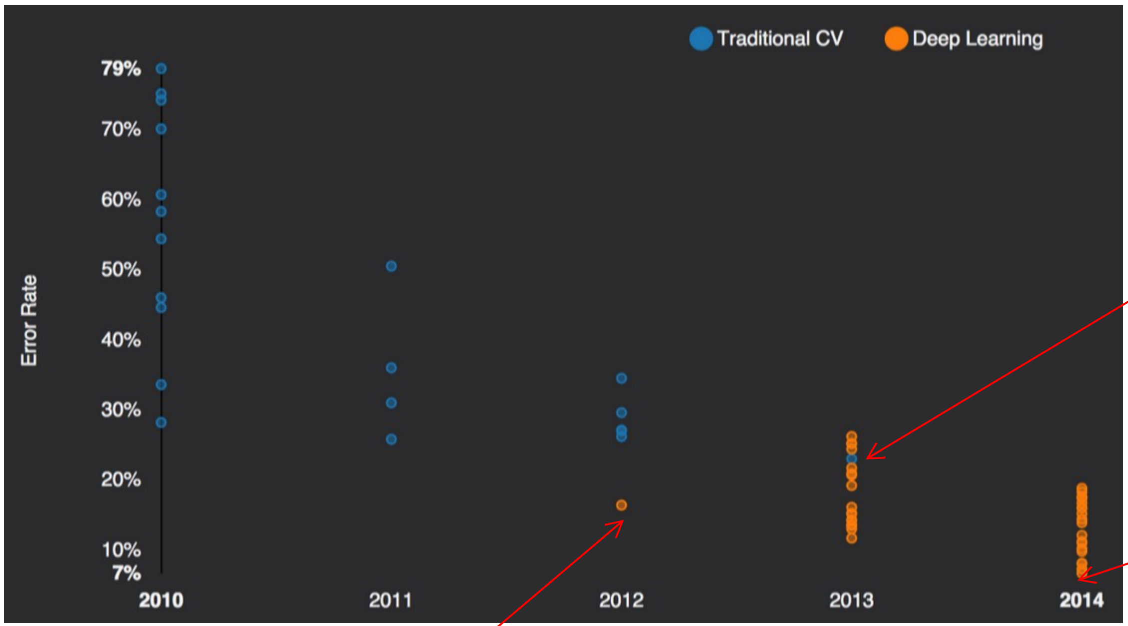
What is a good CNN architecture?

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

Figure: https://medium.com/global-silicon-valley/machine-learning-yesterday-today-tomorrow-3d3023c7b519

Review of ImageNet winning CNN architectures

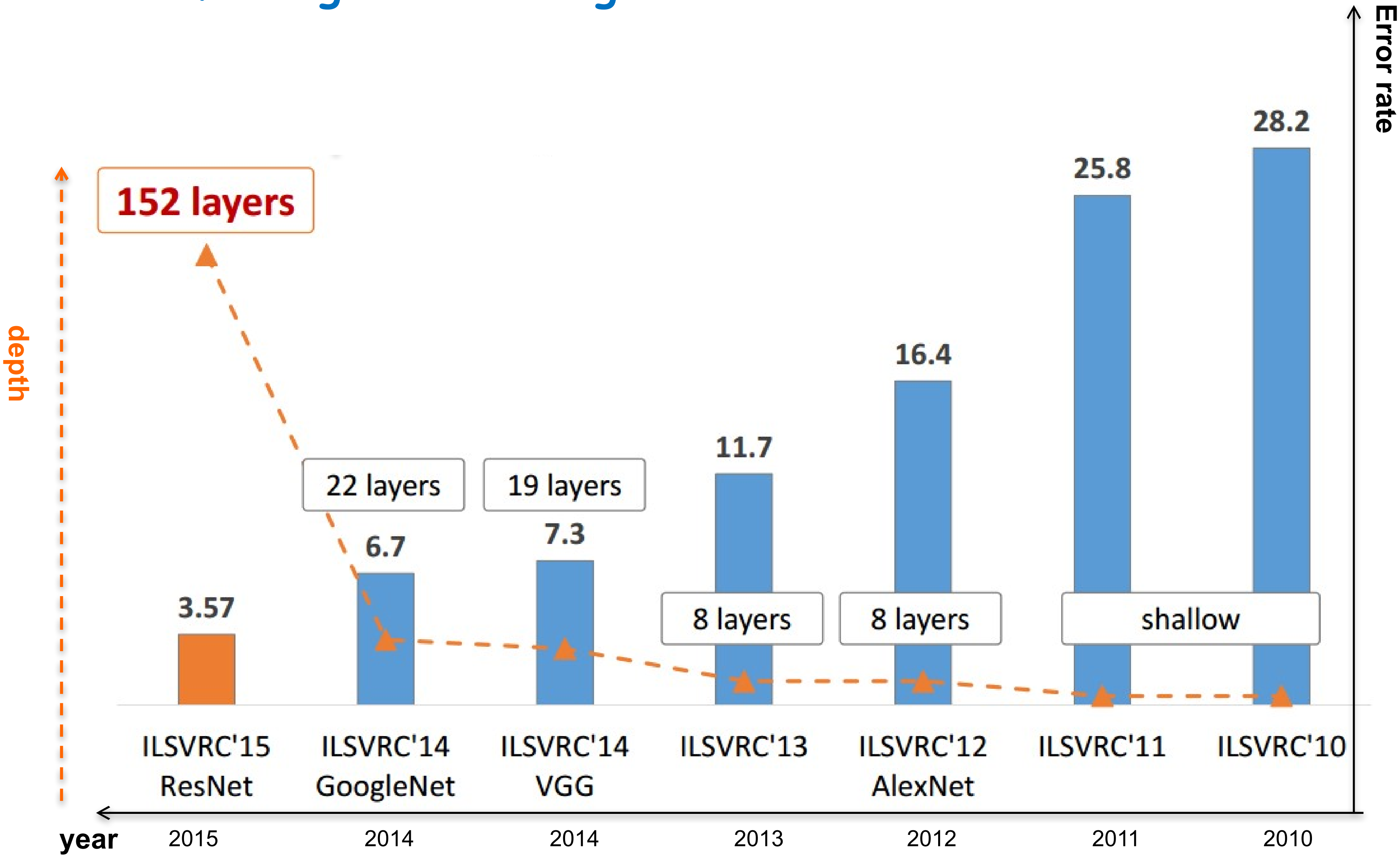


Image credits (modified): K.He, X.Zhang, S.Ren, J.Sun. “Deep Residual Learning for Image Recognition”. arXiv 2015

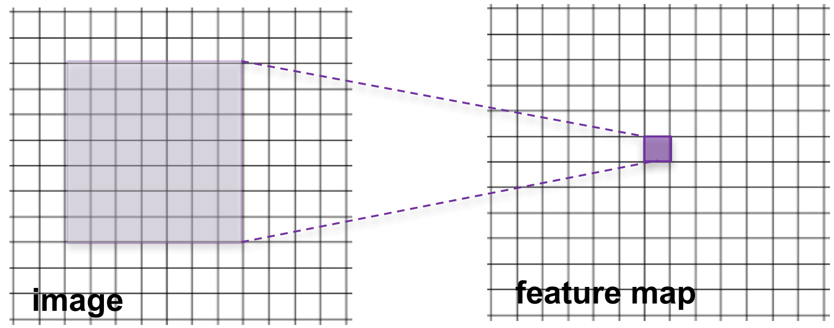
The trend in modern CNN architectures goes to small filters

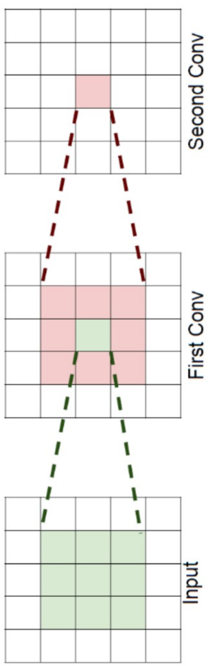
Why do modern architectures use very small filters?
Determine the receptive field in the following situation:

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

49 weights

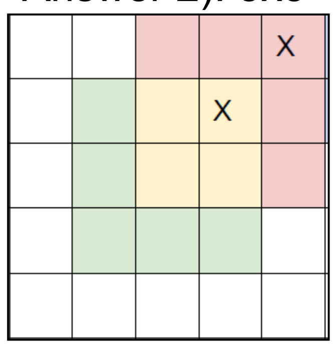
Answer1): 7x7





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

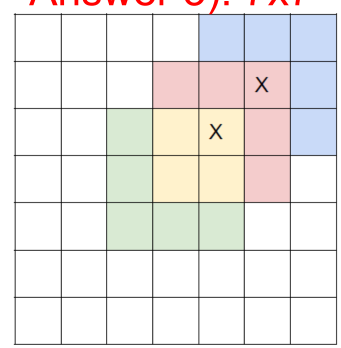
Answer 2): 5x5



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!

“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-trainined model is available (see excercise)

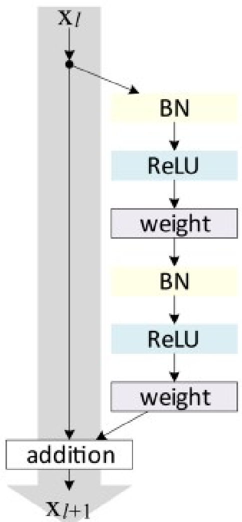
http://arxiv.org/abs/1409.1556

“ResNet” from Microsoft 2015 winner of imageNet

152
layers

ResNet basic design (VGG-style)

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



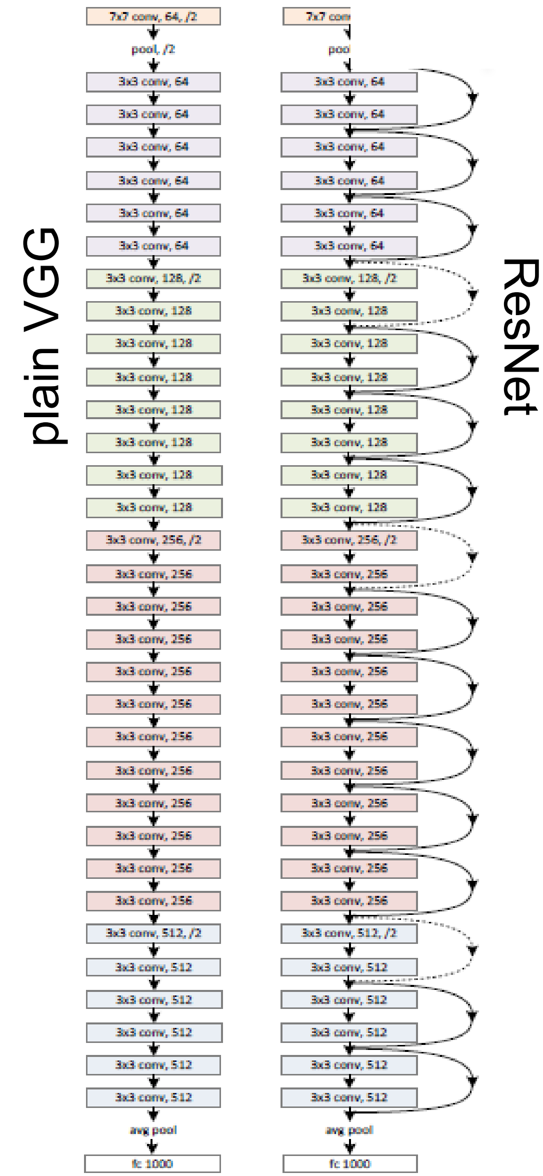
$F(x_l)$

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!

$H(x_l)=x_{l+1} = x_l + F(x_l)$

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

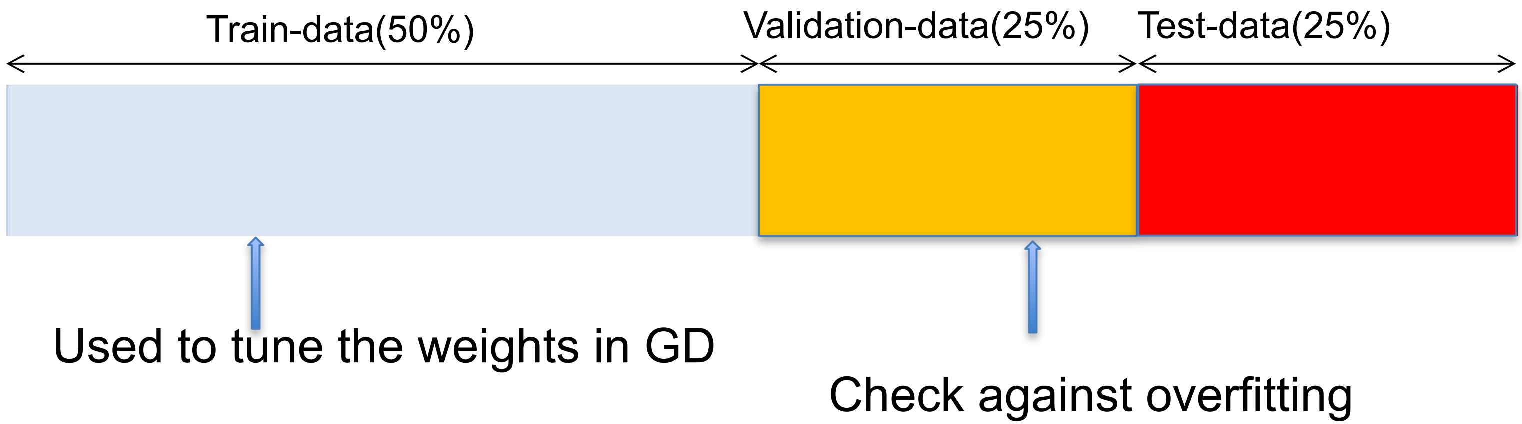


credits: https://chatbotslife.com/resnets-highwaynets-and-densenets-oh-my-9bb15918ee32

Tricks of the trade

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

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

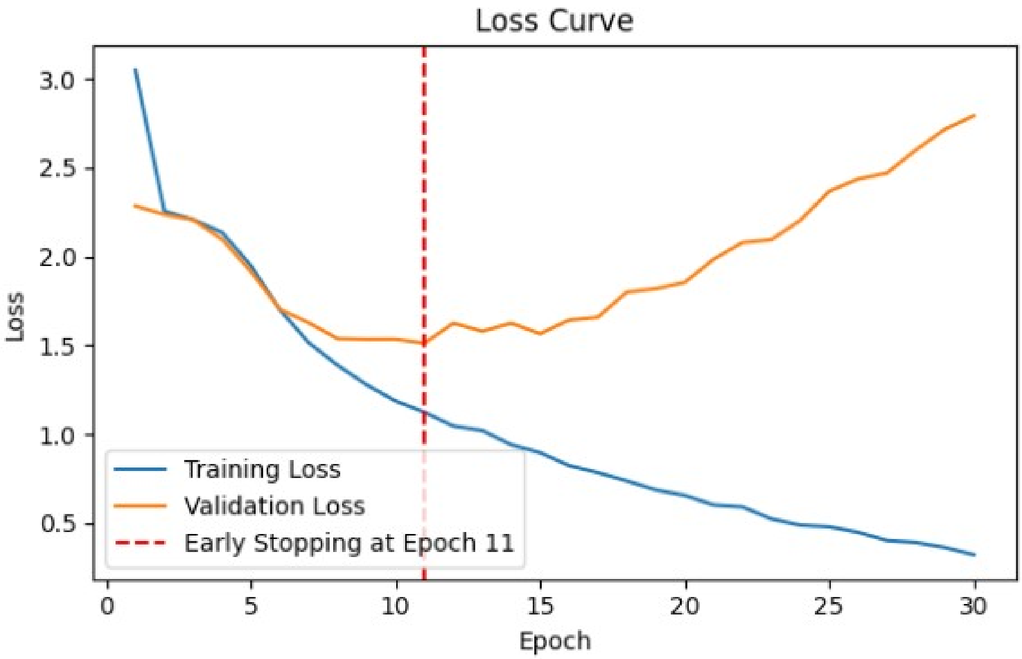


Best practice: Lock an extra test data set away, and use it only at the very end, to evaluate the chosen model, that performed best on your validation set.
Reason: **When trying many models, you probably overfit on the validation set**.

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

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
```

```
## Early Stopping
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=10,
    verbose=1,
    restore_best_weights=False
)
```

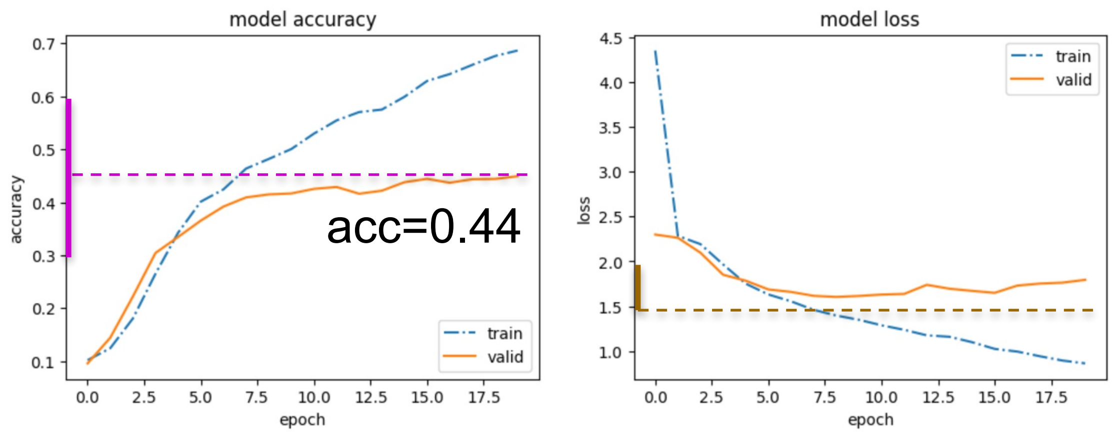
- If training loss does not go down to zero: model is not flexible enough
- Use weights @minimum of validation before overfitting
- Early stopping: stop to training if validation loss does not improve anymore

[07_early_stopping_and_modelweights]

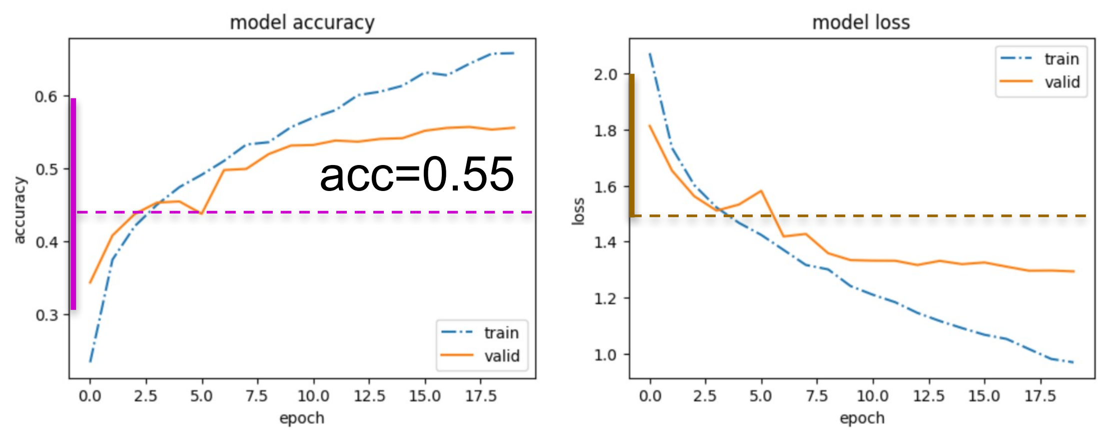
Take-home messages from CIFAR10 CNN study

- 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

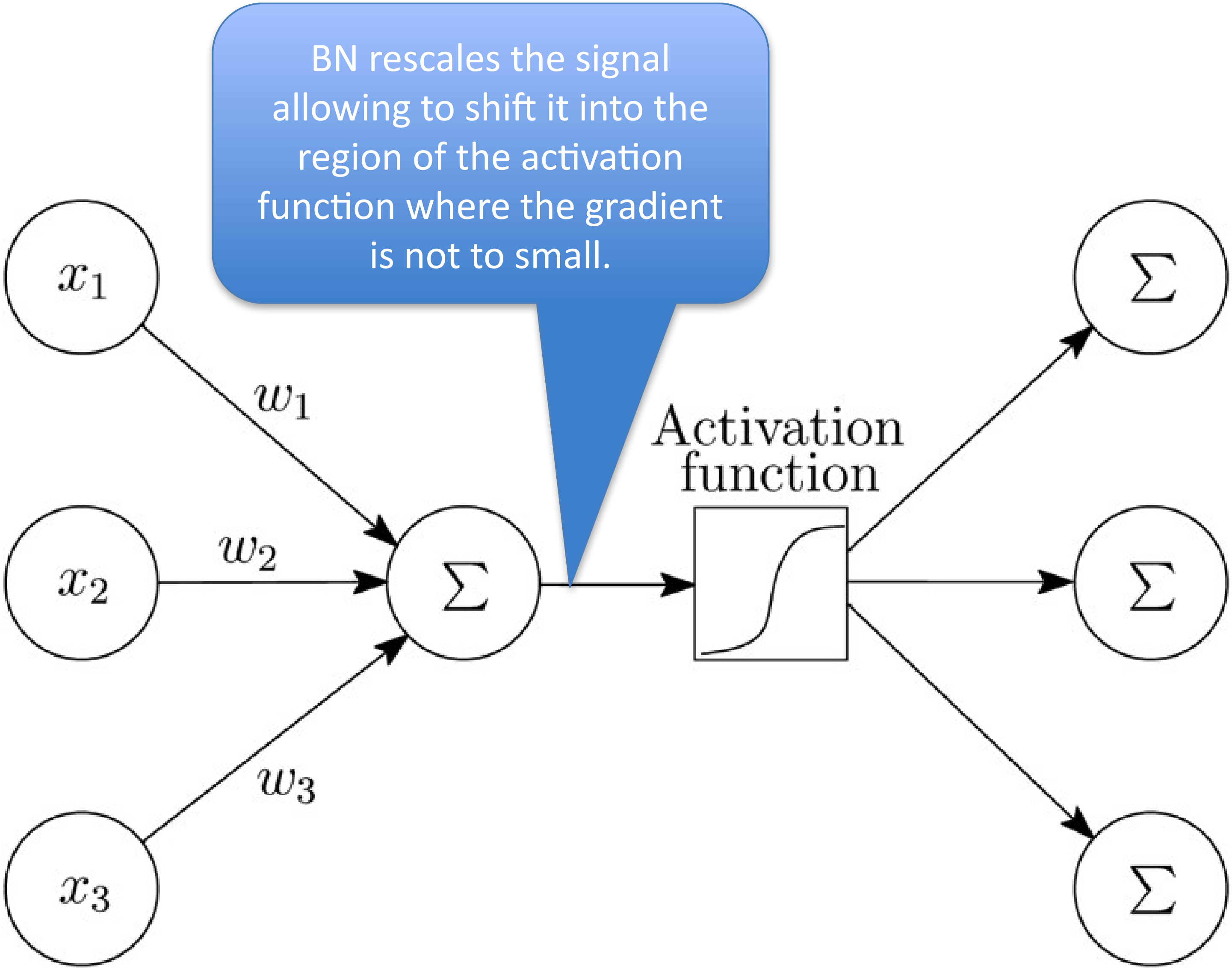


07_cifar10_tricks_keras_torch.ipynb <- CIFAR10 Study

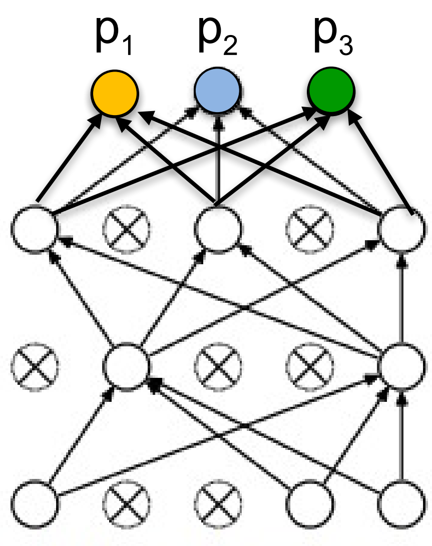
Regularization to avoid overfitting

- Batchnorm layer
- Dropout layer

What is the idea of Batch-Normalization (BN)



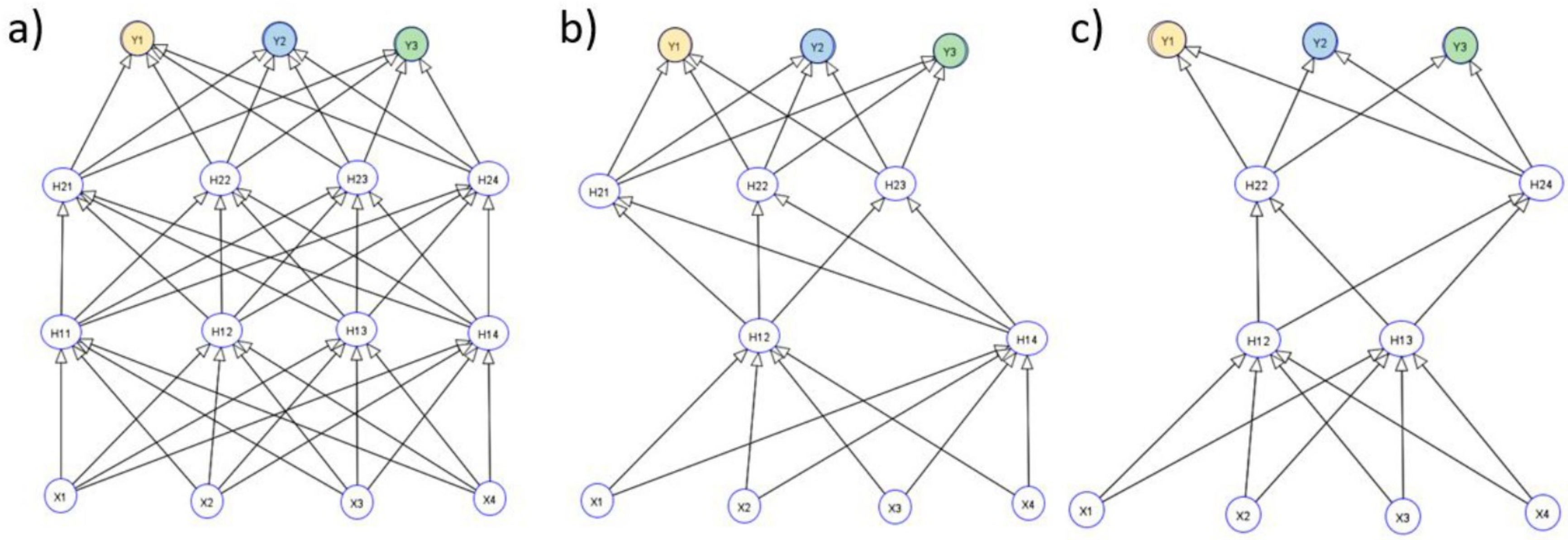
Dropout helps to fight overfitting



Using dropout during training implies:

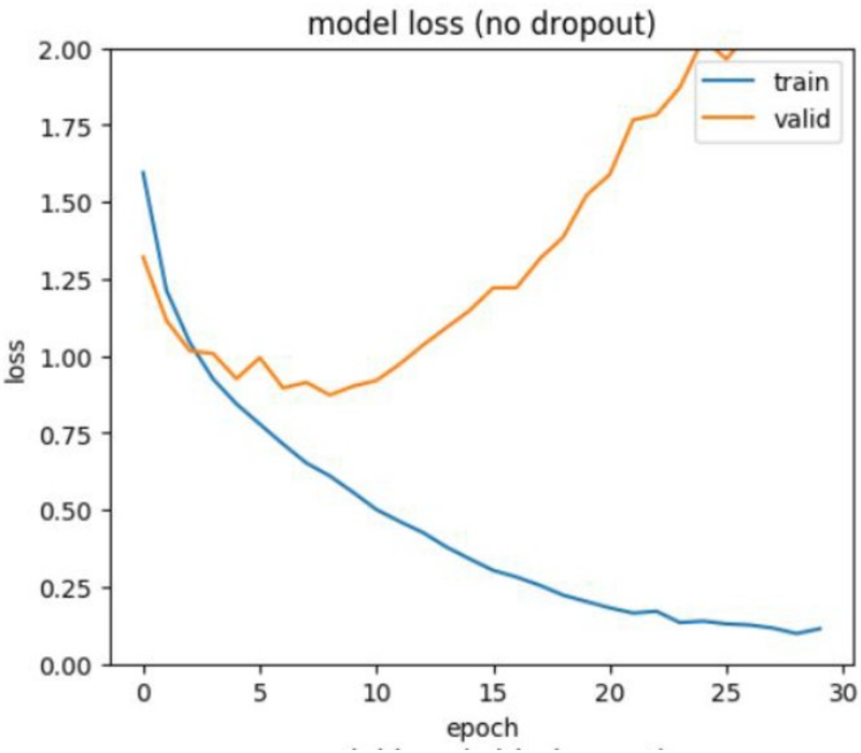
- In each training step only weights to not-dropped units are updated → we train a sparse sub-model NN
- For predictions with the trained NN we freeze the weights corresponding to averaging over the ensemble of trained models we should be able to “reduce noise”, “overfitting”
- JFI: To get same expected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

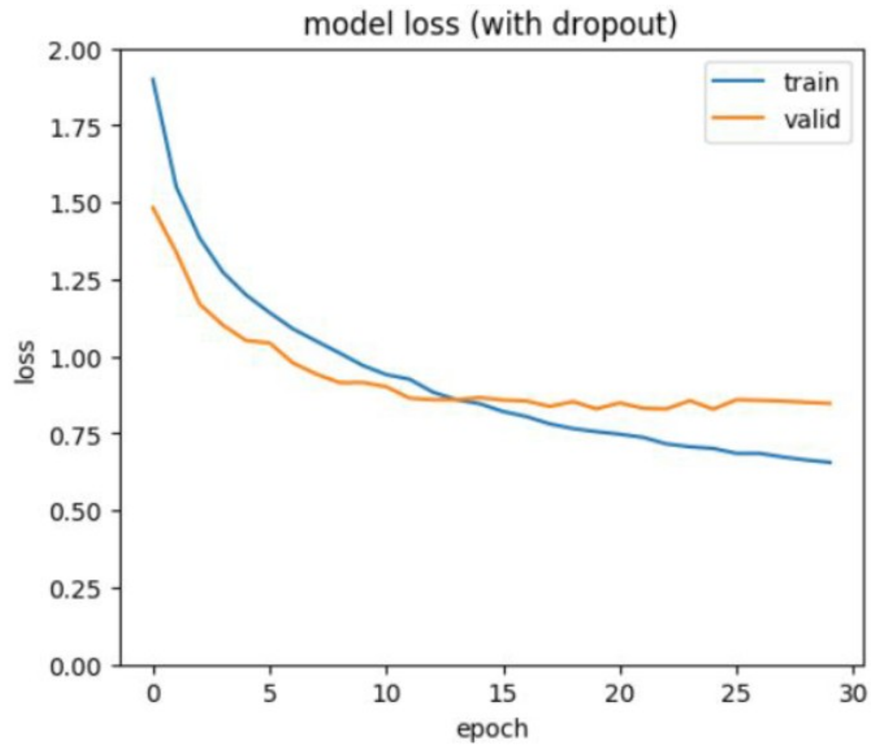
Dropout



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





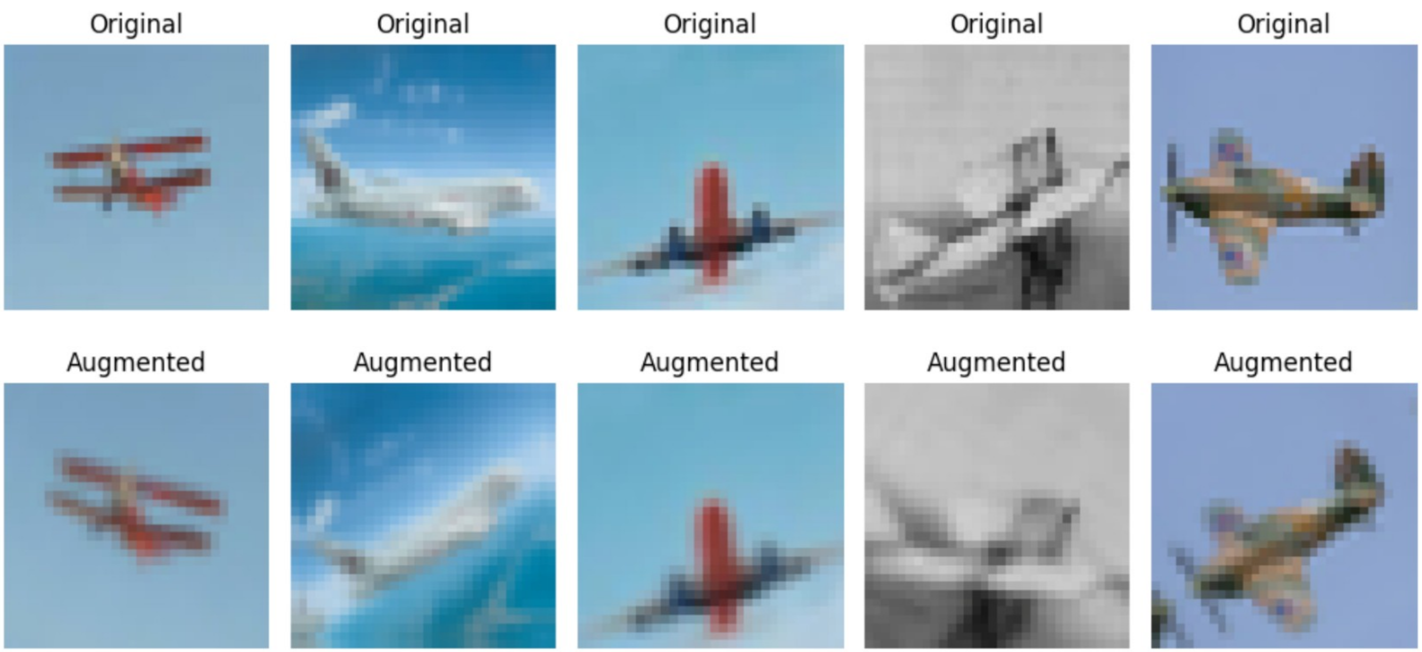
What to do in case of limited data?

- Data augmentation
- Pretrained (foundation) models

Fighting overfitting by Data augmentation:

During training random operations are done on the fly

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



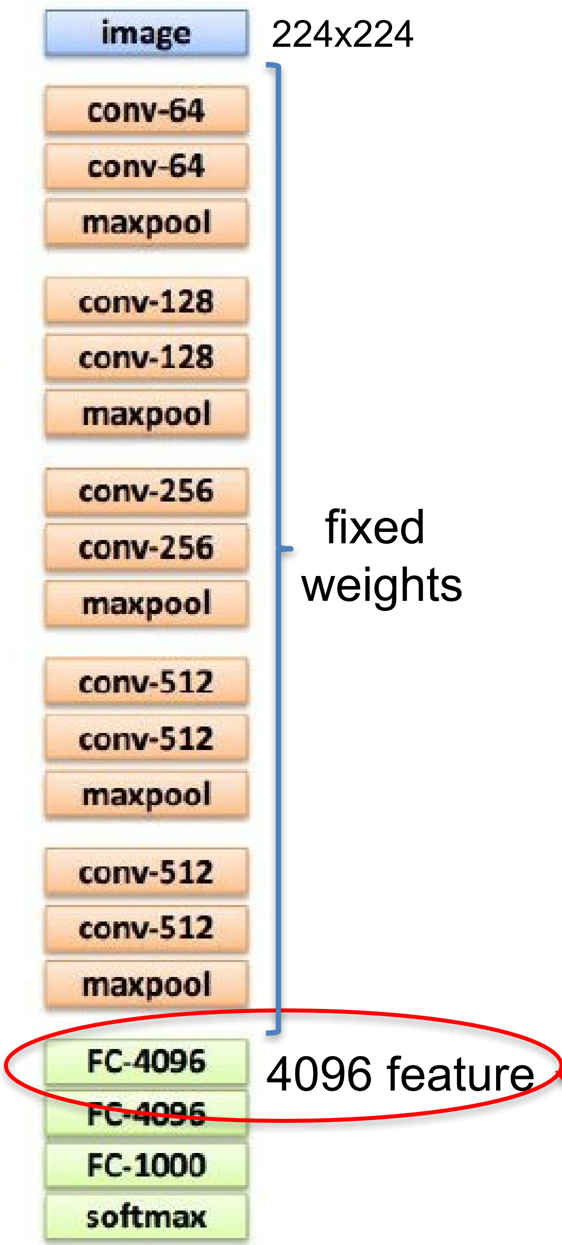
In Code: Augmentation

```python
# Define data augmentation pipeline
from keras.layers import RandomFlip, RandomRotation, RandomContrast, RandomZoom
data_augmentation = keras.Sequential([
    #RandomFlip("horizontal"),        # Randomly flip images horizontally
    RandomRotation(0.1),             # Randomly rotate images by 10%
    RandomZoom(0.1),                 # Randomly zoom in on images
    #RandomContrast(0.1),             # Adjust contrast randomly
], name='Augmentation')
```

```python
# Functional API Model Definition
x = Input(shape=input_shape)
x = data_augmentation(x)
# First Convolutional Block
x = Convolution2D(8, kernel_size, padding='same')(x)
...
```

07_cifar10_tricks_keras_torch.ipynb

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

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:

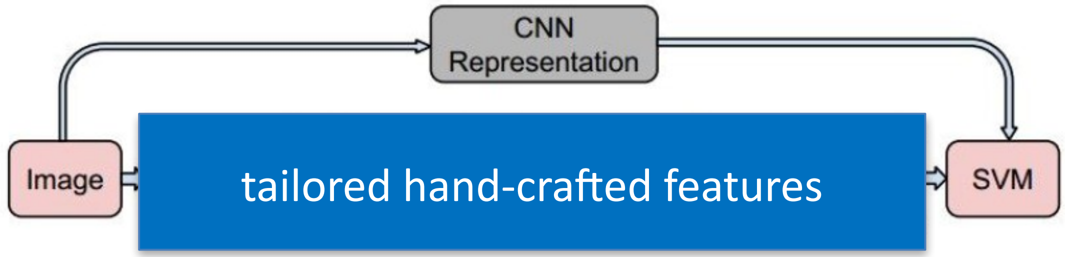
	Similar task (to imageNet challenge)	**Very different task** (to imageNet challenge)
little data	Extract CNN representation of one top fc layer and use these features to train an external classifier	You are in trouble - try to extract CNN representations from different stages and use them as input to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

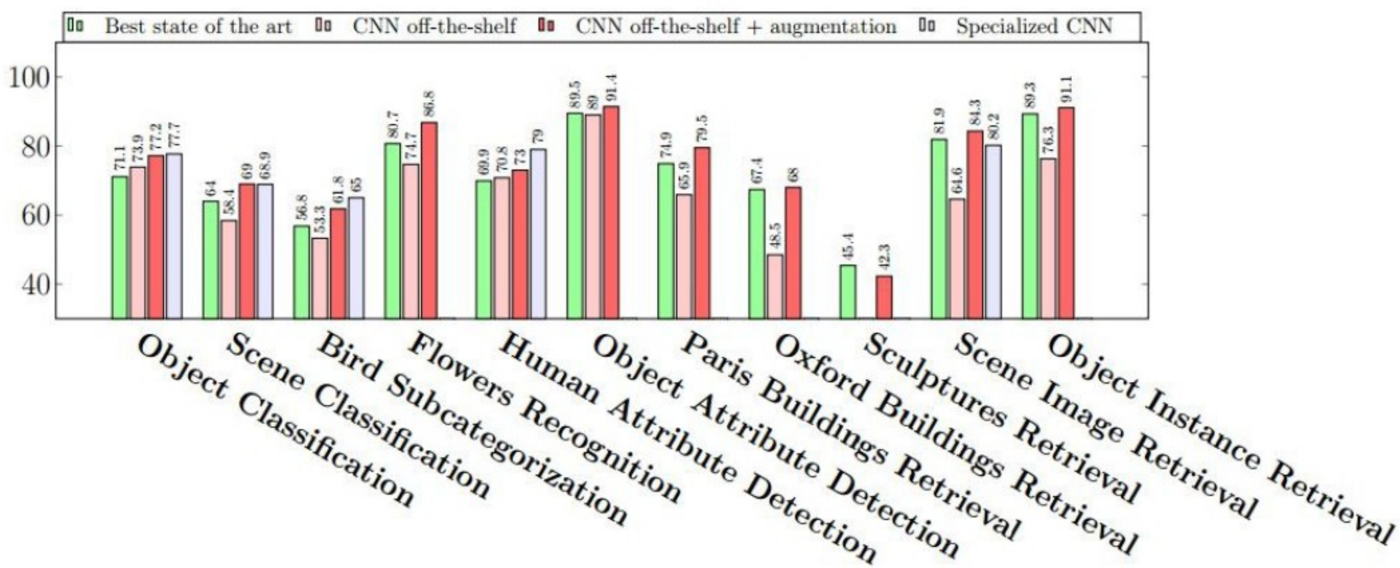
Where to get pretrained CNNs like VGG16 or ResNet: https://keras.io/api/applications/
Hint: first retrain only fully connected layer, only then add convolutional layers for fine-tuning.

Slide credits (modified): cs231n

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

CNN's distributed and compositional features generalize well to new tasks

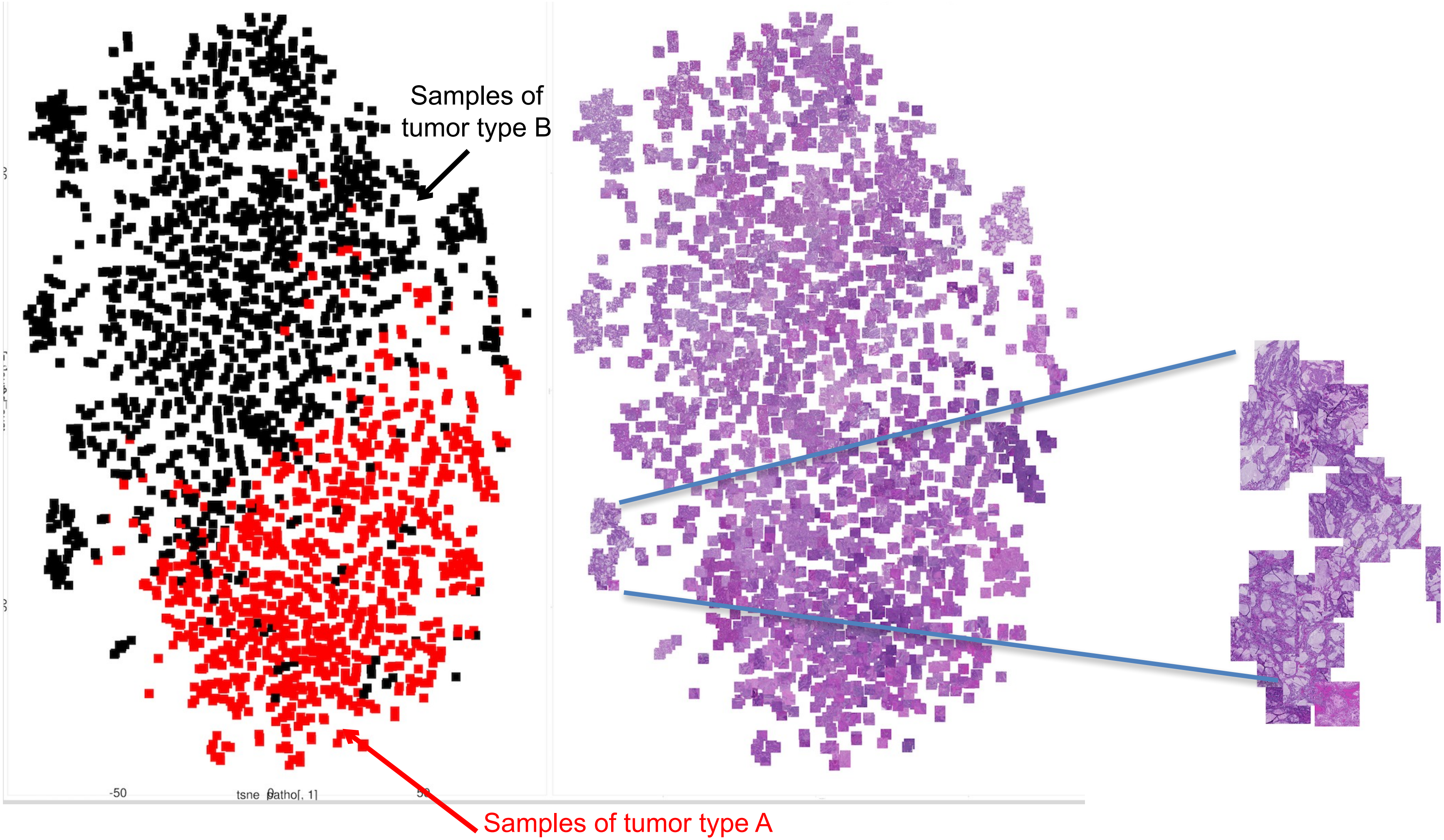




"Astonishingly, we report consistent superior results compared to the highly tuned state-of-the-art systems in all the visual classification tasks on various datasets."

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

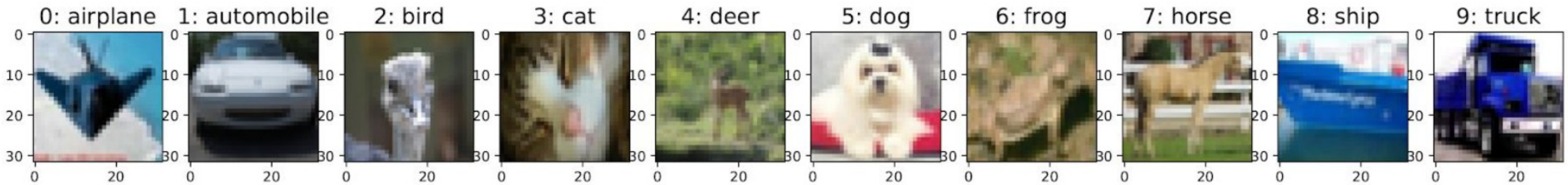
Use features from a pretrained VGG as input to t-SNE



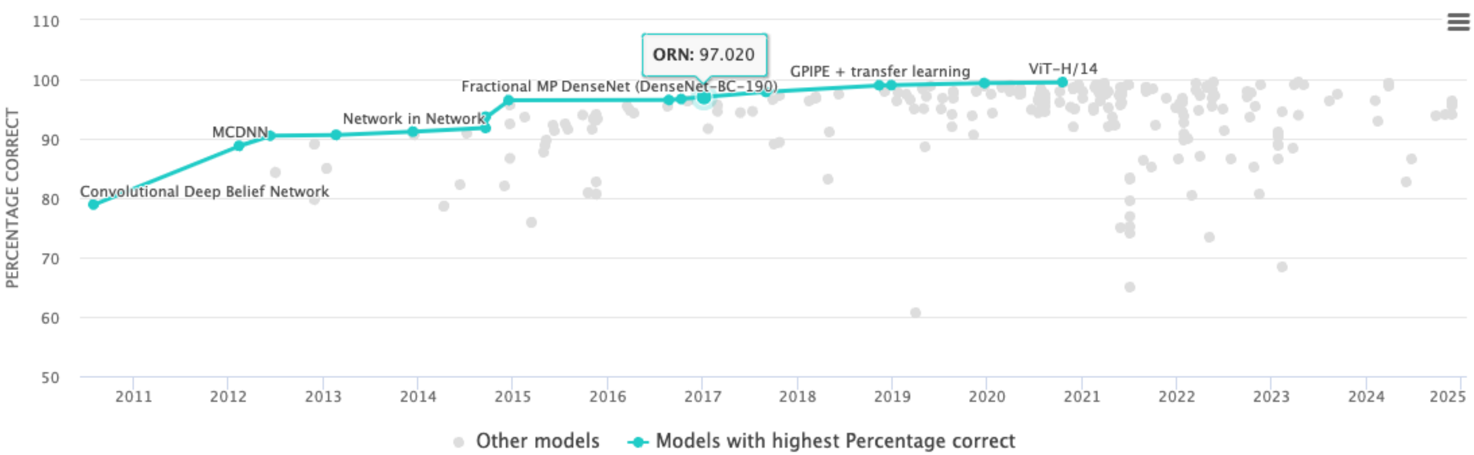
→ Different tissue types cluster together in t-SNE: we could use knn as classifier
→ VGG features even work on images that are far away from the 1000 imageNet classes

Case Study: CIFAR10 Dataset

Cifar10 Data



10 Classes, with 5’000 images for training



Some examples:

VGG-19 with GradInit	2021	94.71
ResNet-18	2022	95.55

https://paperswithcode.com/sota/image-classification-on-cifar-10

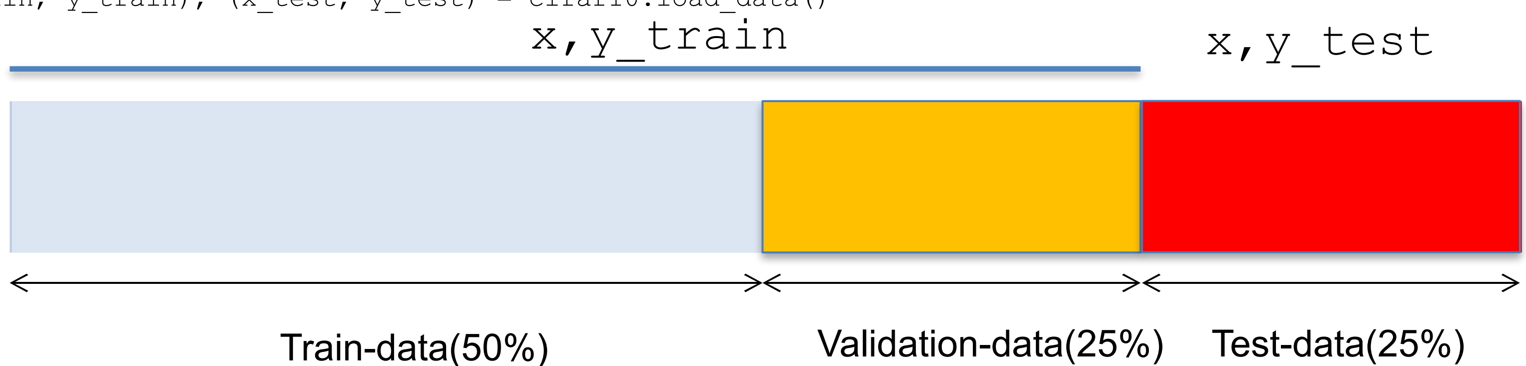
Dataset Splitting in Machine Learning Competitions

- Training Set
  - The set you get for training, you can do what you want with it.
- Test Set:
  - Final evaluation set, never seen by the model during training (e.g., 10,000 images in CIFAR-10).

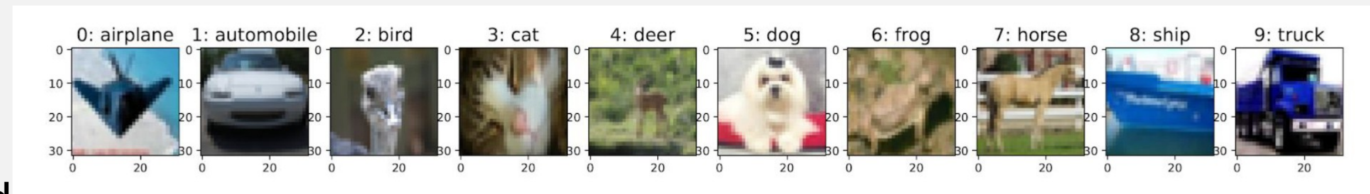
Note: CIFAR-10 and many other datasets do NOT provide a separate validation set.

- Typical Workflow
  - Hyperparameter Tuning
    - **Create validation set** from training set
      - Helps in tuning hyperparameters and preventing overfitting.
      - Common practice: split 80% training / 20% validation (e.g., 40,000 train / 10,000 validation).
  - Final Training: After hyperparameter tuning, models are often retrained on the entire training set
  - Evaluation on test set
    - Sometimes test set is secret and you to upload you predictions (private test set)

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```



Exercise: Transfer learning with CIFAR10 data



rf on vgg features

transfer learning on vgg features

cnn from scratch with data augmentation

08_classification_transfer_learning_few_labels_keras_torch.ipynb

Take home: DL with images in practice

- Use NN architectures inspired by challenge winning NNs

- Use tricks of the trade when building a CNN
    - Normalization of input data
    - Batchnorm
    - Dropout
    - Data Augmentation
    - Early stopping (patience)

- In case of few data
    - Do augmentation during training of a CNN
    - Use shallow learner (e.g. RF) based on image features extracted via pretrained CNNs
    - Fine-tune a pretrained CNN on few data (transfer learning)

- Use pretrained (foundation) models