Hochschule Konstanz
Fakultät Informatik

First Version might change slightly

Oliver Dürr

Short course on deep learning
Block 3

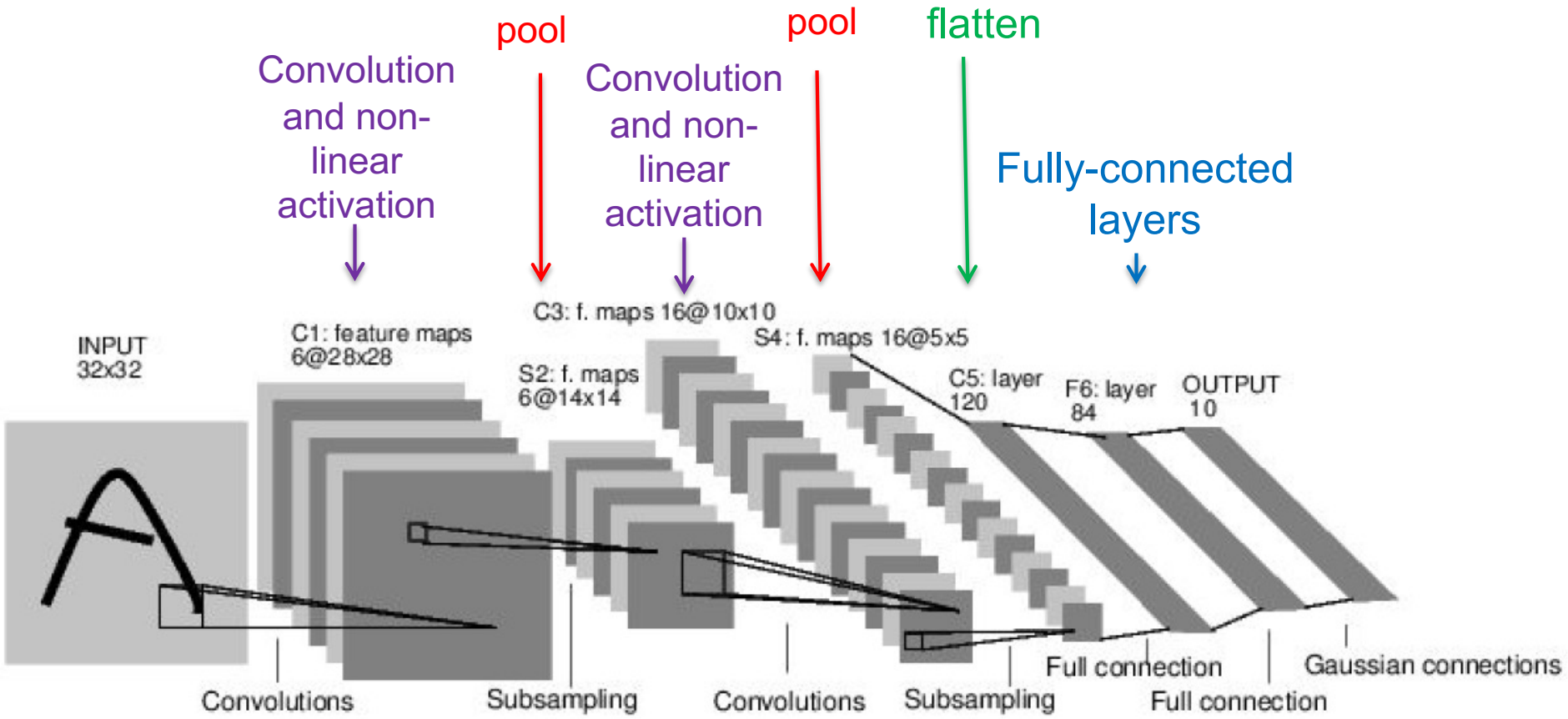
Southern Taiwan University of Science and Technology

This course is a short version of dl_course_2022 I created with Beate Sick.

Topics of today

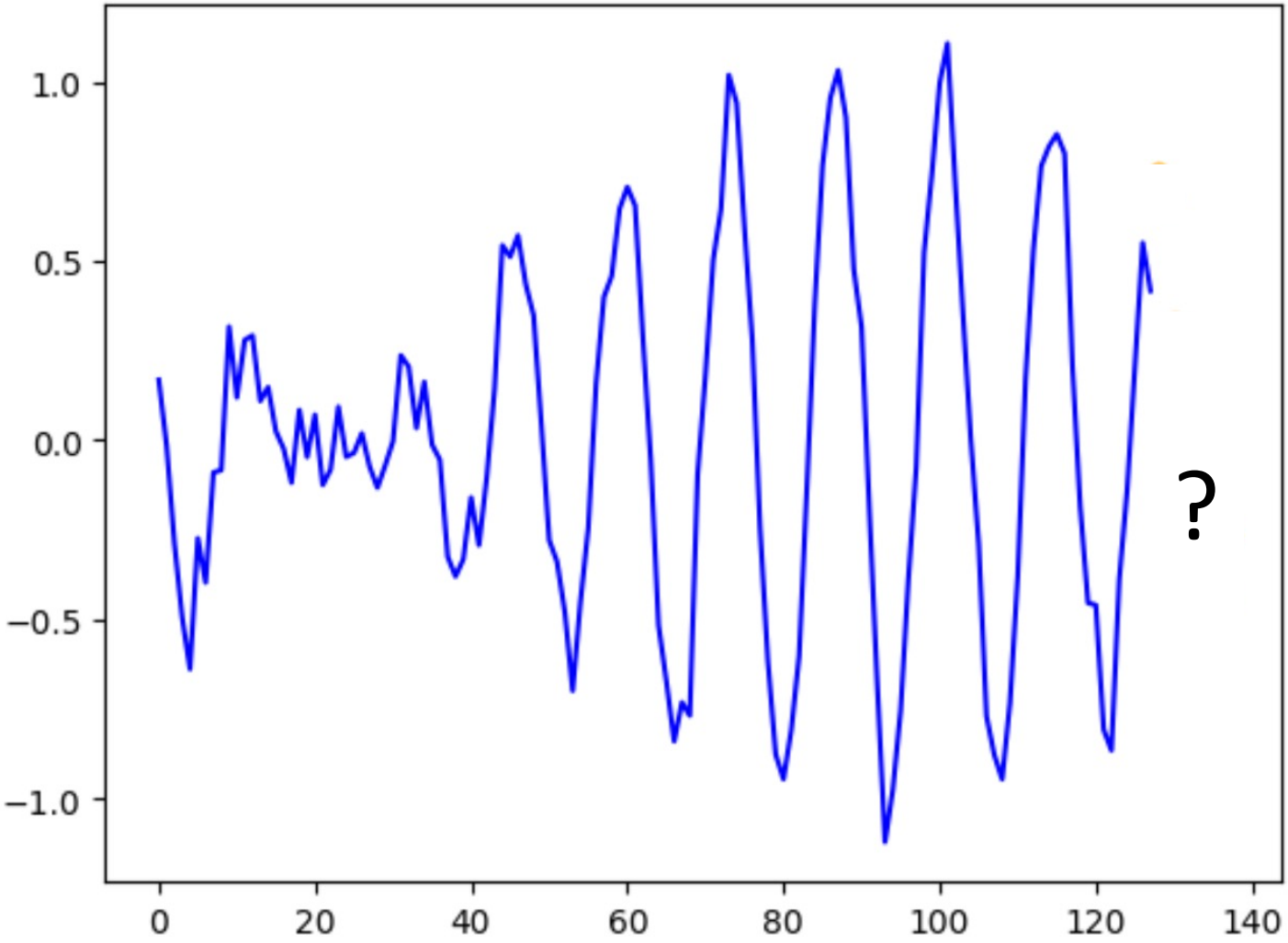
- 1D-CNN for sequence data
- Tricks of the trade in CNNs
  - Standardizing / Batch-Norm
  - Dropout
- Challenge winning CNN architectures
- DL with few data:
  - Transfer learning
  - Augmentation

Typical architecture of a simple CNN



1D CNNs for sequence data

How to make predictions based on a given time series?

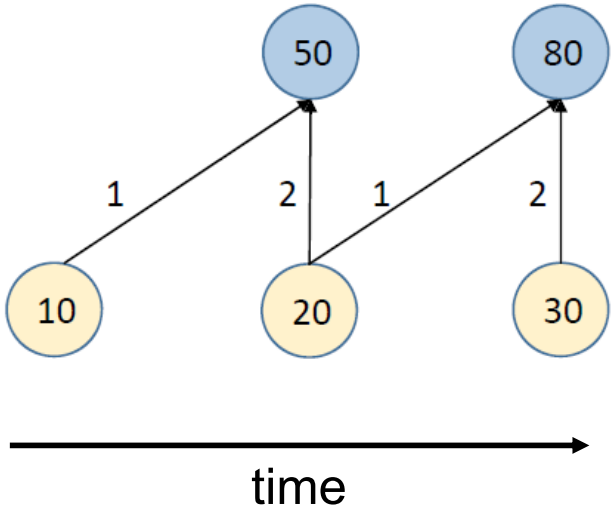


1D "causal" convolution for time-ordered data

Toy example:

Input X: 10,20,30

1D kernel of size 2: 1,2



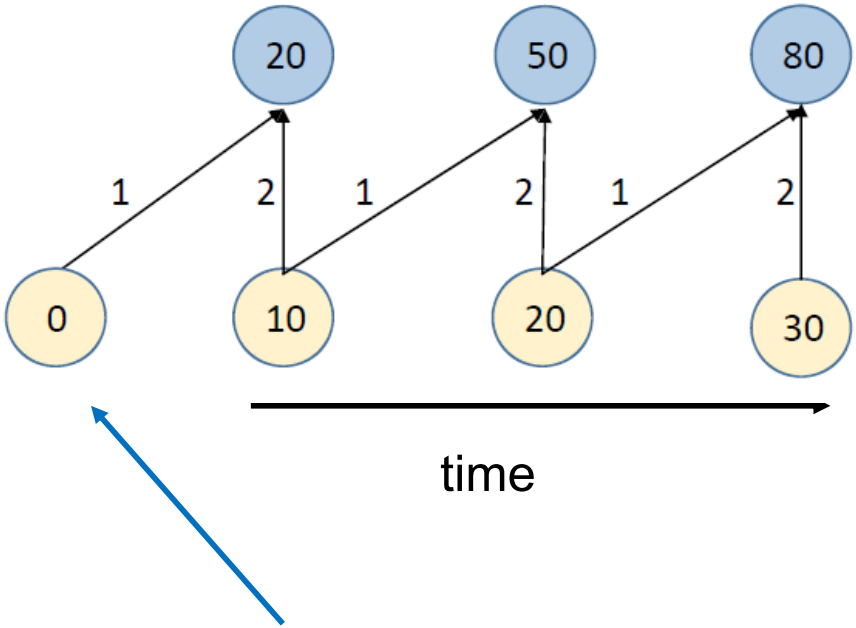
It's called "causal" networks, because the architecture ensured that only information from the past has an influence on the present and future.

Zero-padding in 1D "causal" CNNs

Toy example:

Input X: 10,20,30

1D kernel of size 2: 1,2

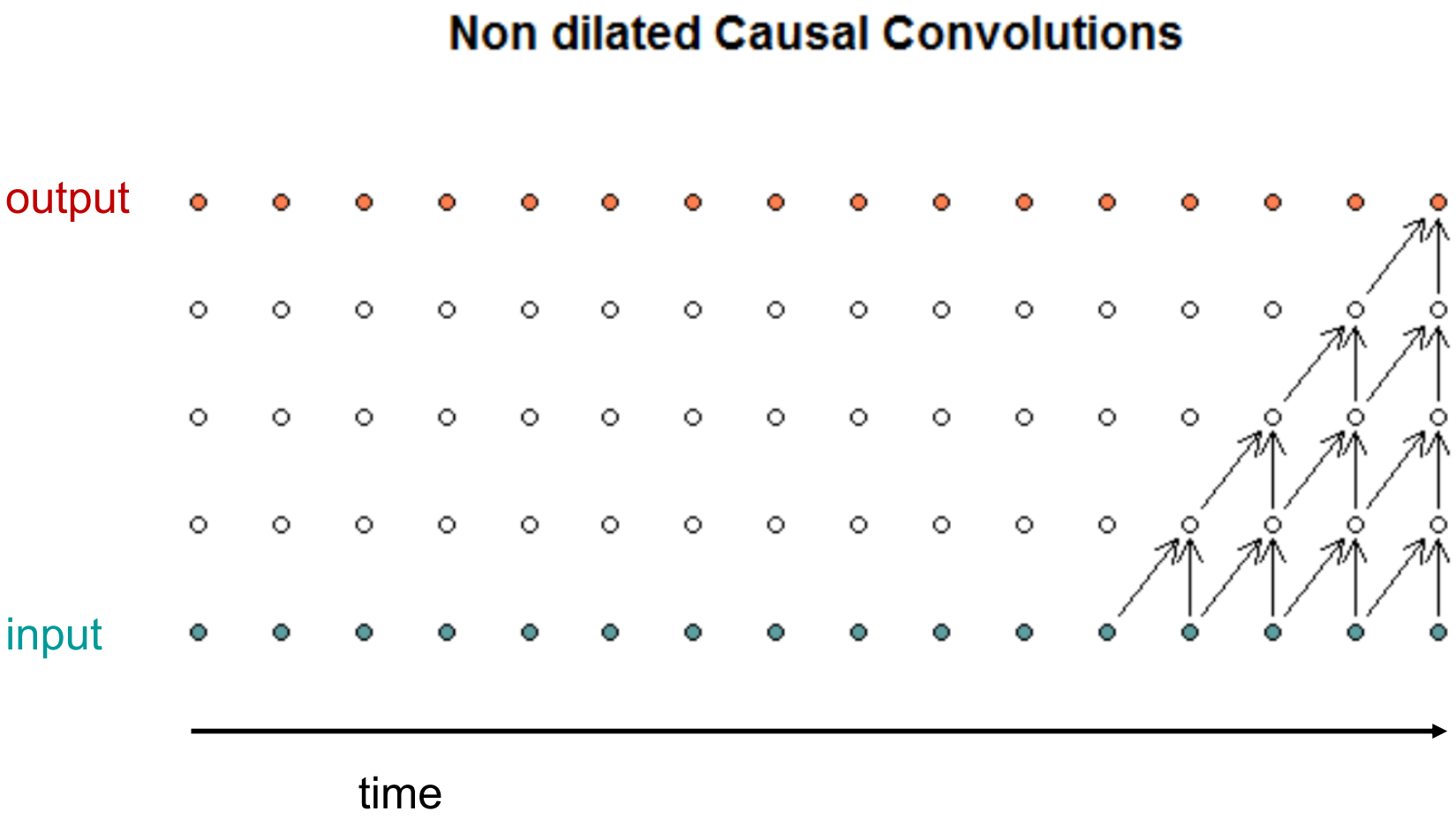


To make all layers the same size, a zero padding is added to the beginning of the input layers

1D "causal" convolution in Keras

```
model = Sequential()
model.add(Convolution1D(filters=1,
                        kernel_size=2,
                        padding='causal',
                        dilation_rate=1,
                        use_bias=False,
                        batch_input_shape=(None,3, 1)))
model.summary()
```

Stacking 1D "causal" convolutions without dilation



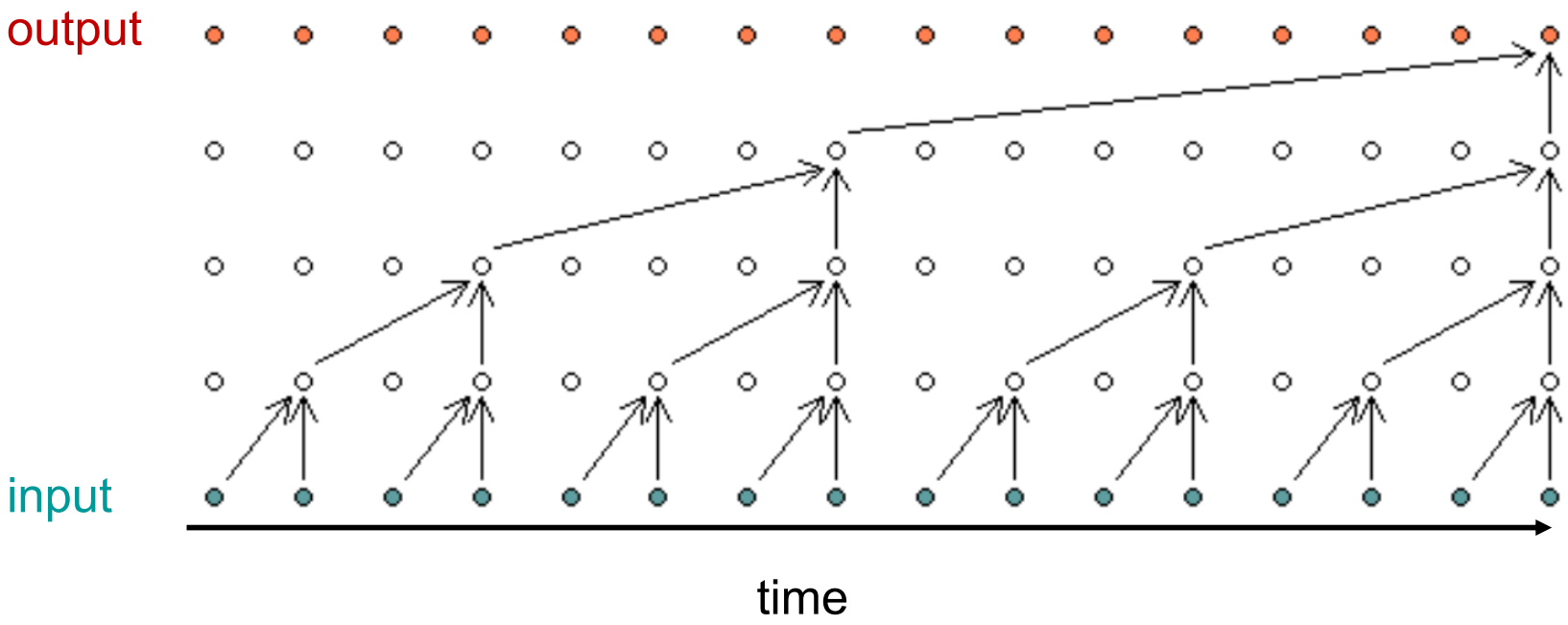
Stacking k causal 1D convolutions with kernel size 2 allows to look back k time-steps.

After 4 layers each neuron has a "memory" of 4 time-steps back in the past.

https://deepmind.com/blog/wavenet-generative-model-raw-audio/

Dilation allows to increase receptive field

To increase the memory of neurons in the output layer, you can use "dilated" convolutions:



After 4 layers each neuron has a "memory" of 15 time-steps back in the past.

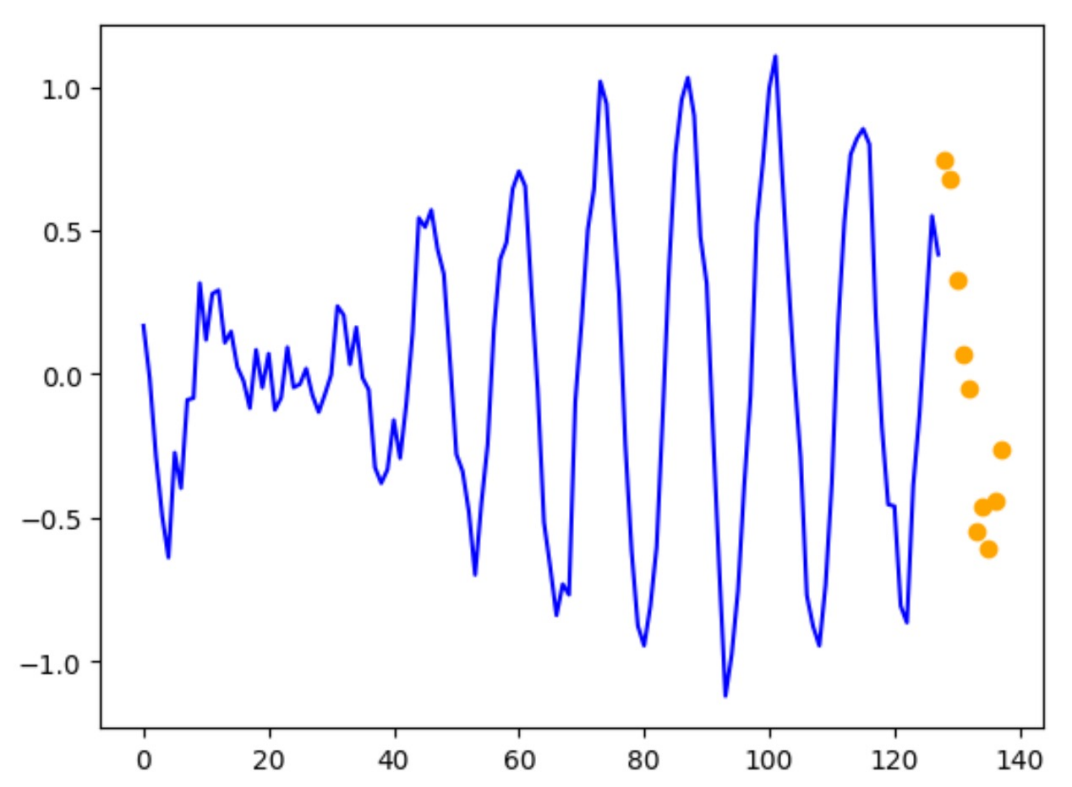
https://deepmind.com/blog/wavenet-generative-model-raw-audio/

Dilated 1D causal convolution in Keras

To use time-dilated convolutions, simply use the argument rate dilation_rate=... in the Convolution1D layer.

```
X,Y = gen_data(noise=0)

modeldil = Sequential()
#<------ Just replaced this block
modeldil.add(Convolution1D(filters=32, kernel_size=ks, padding='causal', dilation_rate=1,
                           batch_input_shape=(None, None, 1)))
modeldil.add(Convolution1D(filters=32, kernel_size=ks, padding='causal', dilation_rate=2))
modeldil.add(Convolution1D(filters=32, kernel_size=ks, padding='causal', dilation_rate=4))
modeldil.add(Convolution1D(filters=32, kernel_size=ks, padding='causal', dilation_rate=8))
#<------ Just replaced this block

modeldil.add(Dense(1))
modeldil.add(Lambda(slice, arguments={'slice_length':look_ahead}))

modeldil.summary()

modeldil.compile(optimizer='adam',loss='mean_squared_error')

histdil = modeldil.fit(X[0:800], Y[0:800],
                       epochs=200,
                       batch_size=128,
                       validation_data=(X[800:1000],Y[800:1000]), verbose=0)
```

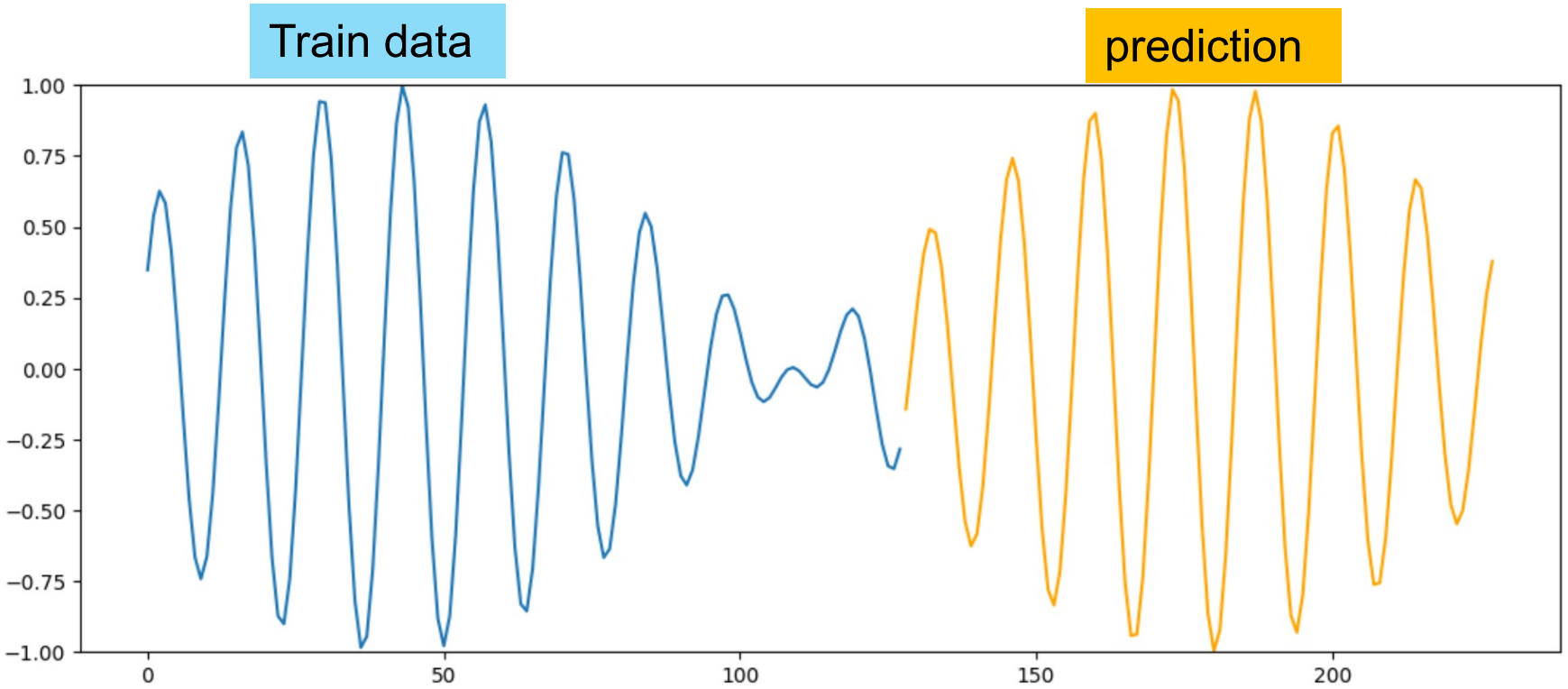
1D-Convolution



Work through the notebook (optional) 09_1DConv.ipynb

Dilated 1D causal CNNs help if long memory is needed

Dilated 1D CNNs can picked up the long-range time dependencies.



If you want to get a better understanding how 1D convolution work, you can go through the notebook at (optional in case you want to work with 1D CNN)
https://github.com/tensorchiefs/dl_course_2022/blob/master/notebooks/09_1DConv_sol.ipynb.

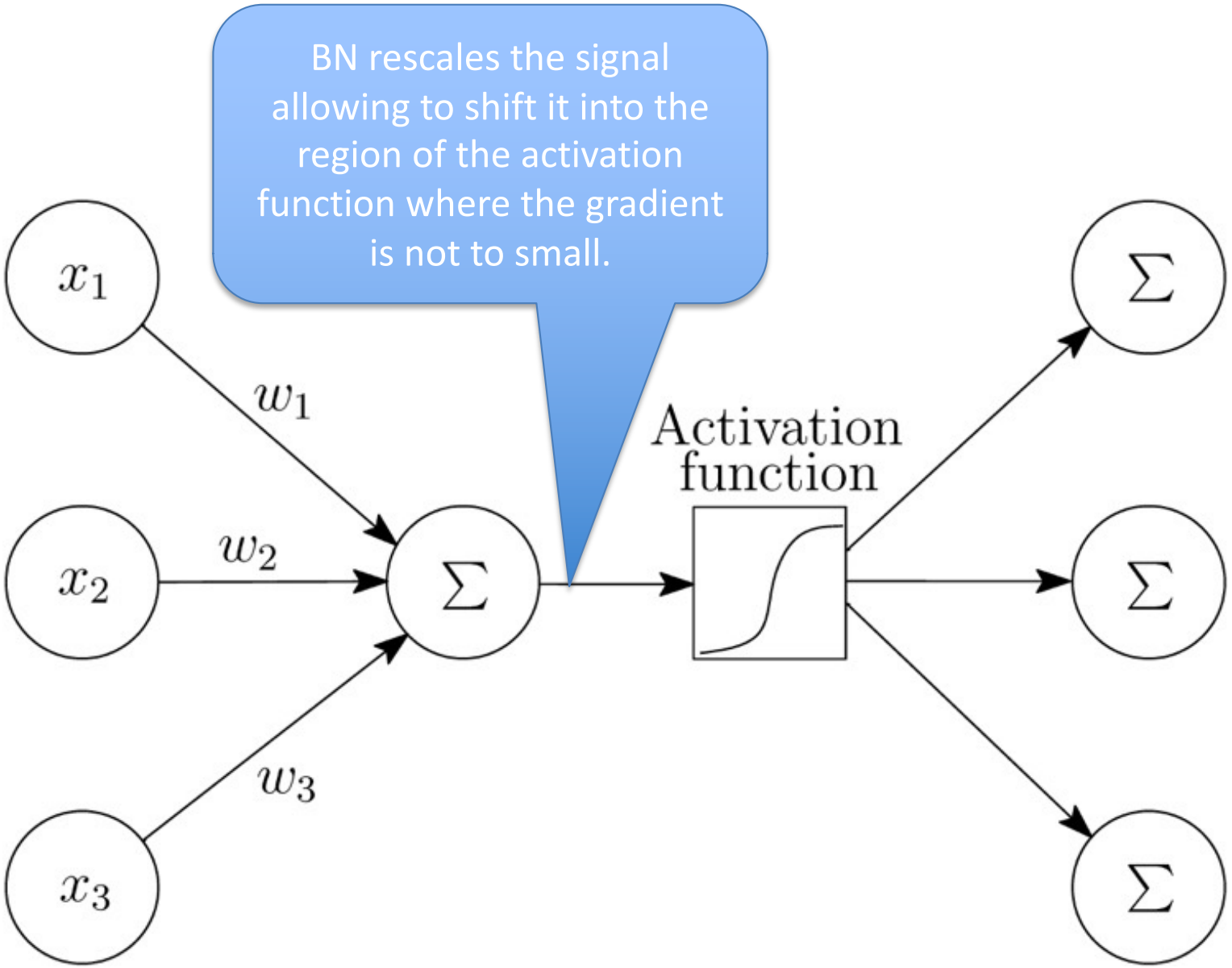
Data Standardization

Batch Norm Layer

Data Standardization

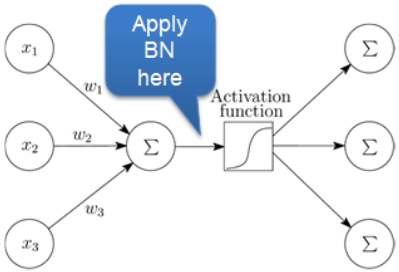
- In contrast to other classifiers like trees (or Random Forest) result strongly depends on range of input data
- If you “steal” an architecture, make sure to use correct standardization
- Also important in transfer learning (see below)
- Standardization
  - Divide by range
  - Z-Transformation
- Automatic Z-transformation
  - Batch-Normalization

What is the idea of Batch-Normalization (BN)



Batch Normalization

- Idea: Introduce batch-norm layers between convolutions.



A BN layer performs a 2-step procedure with **α and β as learnable** parameter:

Step 1: $$\hat{x} = \frac{x - avg_{batch}(x)}{stdev_{batch}(x) + \epsilon}$$

$avg(\hat{x}) = 0$

$\text{stdev}(\hat{x}) = 1$

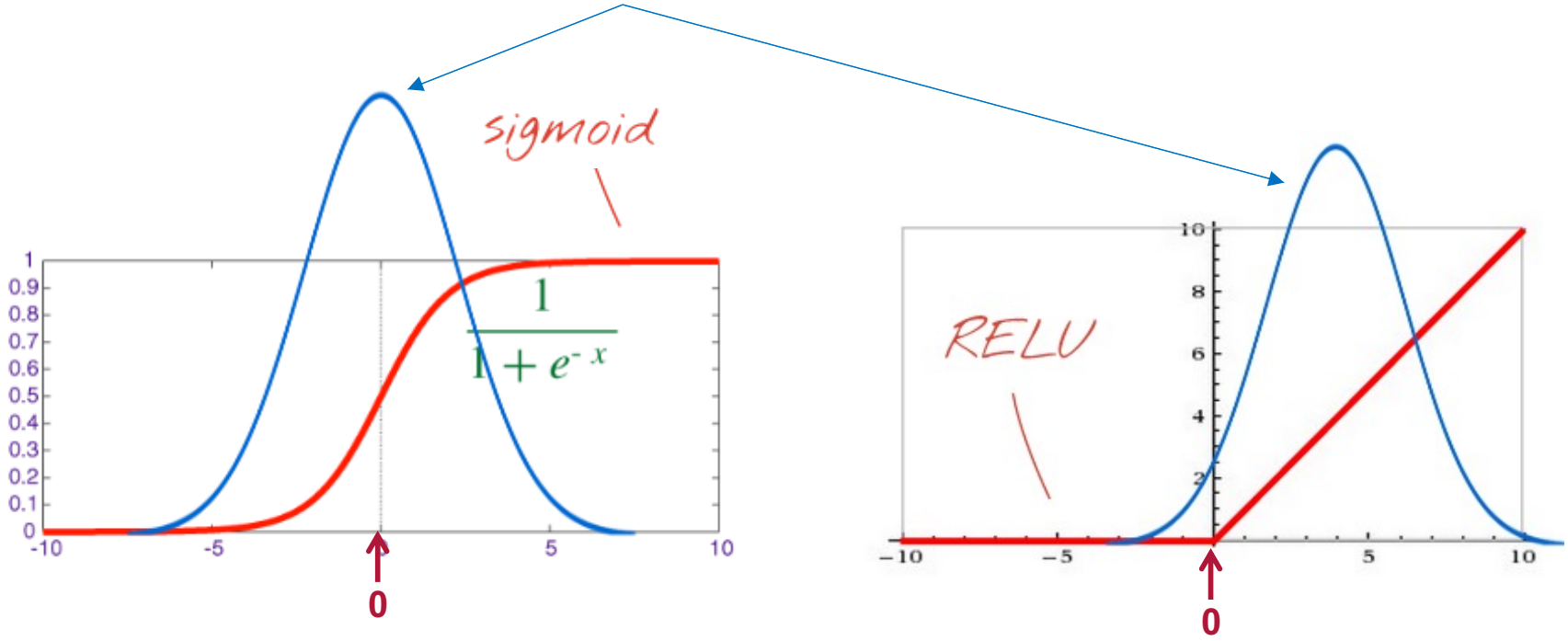
Step 2: $$BN(x) = \alpha\hat{x} + \beta$$

The learned parameters α and β determine how strictly the standardization is done. If the learned α=stdev(**x**) and the learned β=avg(**x**), then the standardization performed in step 1 is undone in step 2 and BN(**x**)=**x**.

Batch Normalization is beneficial in many NN

After BN the input to the activation function is in the sweet spot

Observed distributions of signal after BN before going into the activation layer.

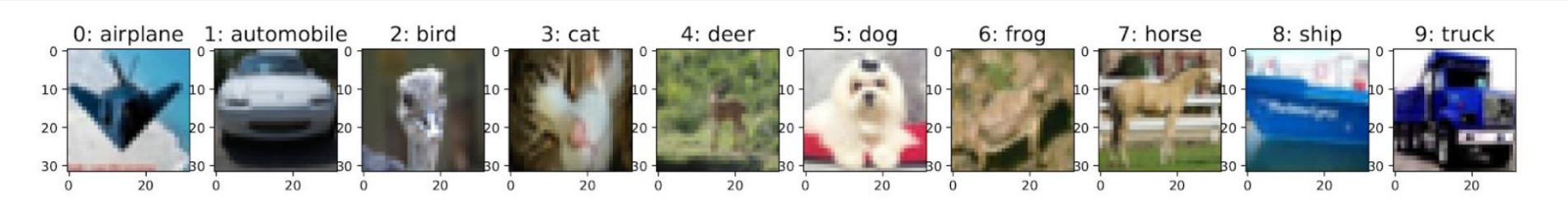


When using BN consider the following:

- Using a higher learning rate might work better
- Use less regularization, e.g. reduce dropout probability

Image credits: Martin Gorner:
https://docs.google.com/presentation/d/e/2PACX-1vRouwj_3cYsmLrNNI3Uq5gv5-hYp_QFdeoan2GlxKgIZRSejozruAbVV0IMXBoPsINB7Jw92vJo2EAM/pub?slide=id.g187d73109b_1_2921

Develop a CNN for cifar10 data [Just for Reference]



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

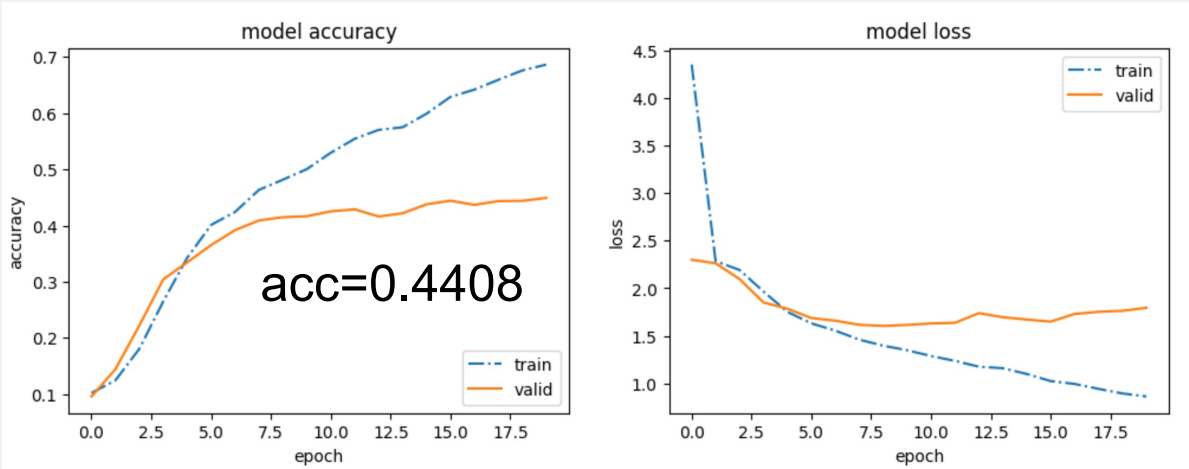
Investigate the impact of standardizing the data on the performance

Notebook:
https://github.com/tensorchiefs/dl_course_2022/blob/master/notebooks/07_cifar10_norm_sol.ipynb

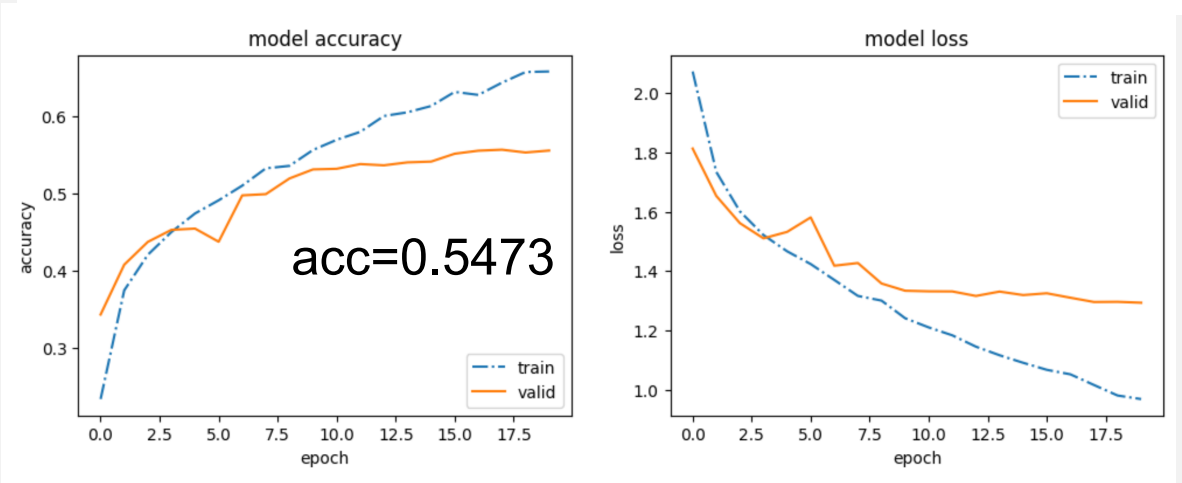
Take-home messages from the homework

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

Without standardizing the input to the CNN

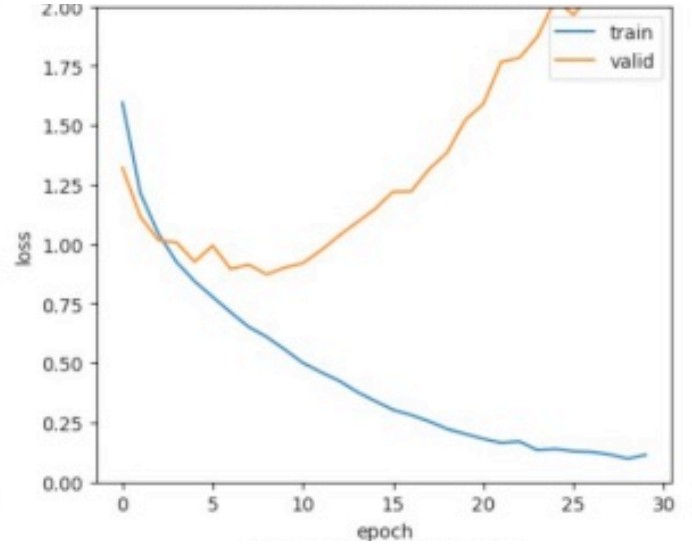


After standardizing the input to the CNN



Dropout during training

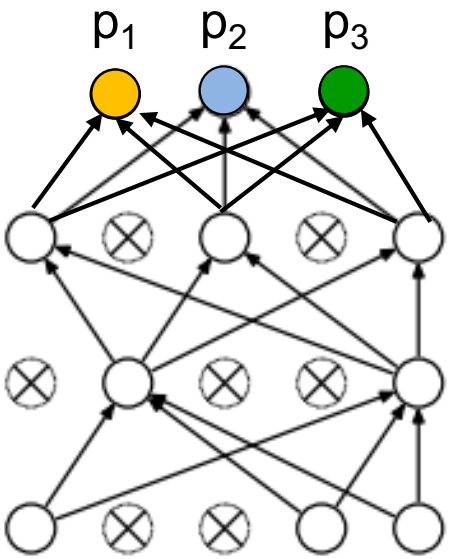
To avoid overfitting early stopping




```
>>> callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
>>> # This callback will stop the training when there is no improvement in
>>> # the loss for three consecutive epochs.
>>> model = tf.keras.models.Sequential([tf.keras.layers.Dense(10)])
>>> model.compile(tf.keras.optimizers.SGD(), loss='mse')
>>> history = model.fit(np.arange(100).reshape(5, 20), np.zeros(5),
...                     epochs=10, batch_size=1, callbacks=[callback],
...                     verbose=0)
>>> len(history.history['loss'])  # Only 4 epochs are run.
4
```

See https://keras.io/api/callbacks/early_stopping/

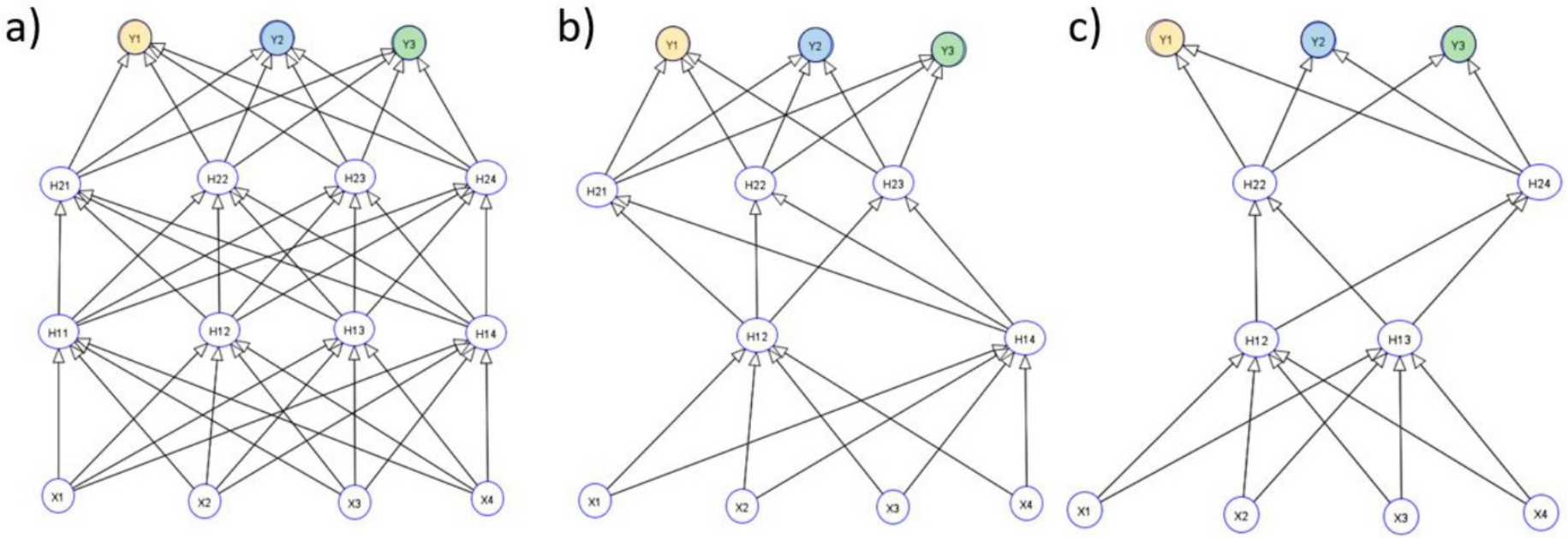
Dropout helps to fight overfitting



Using dropout during training implies:

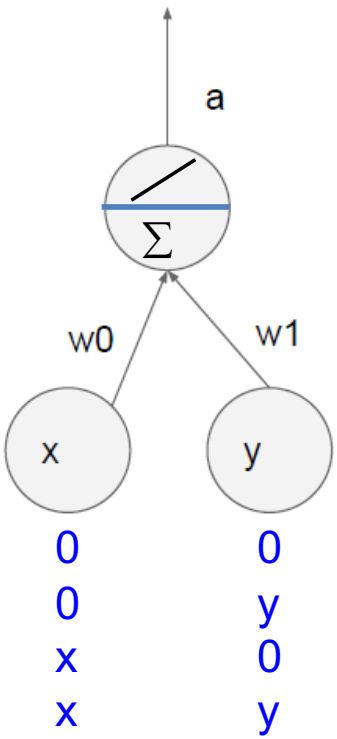
- In each training step only weights to not-dropped units are updated → we train a sparse sub-model NN
- For predictions with the trained NN we freeze the weights corresponding to averaging over the ensemble of trained models we should be able to “reduce noise”, “overfitting”

Dropout: kind of NN ensemble



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 require weight adaptation



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**

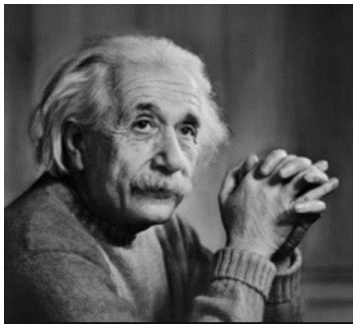
during training with dropout probability 0.5:

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)**

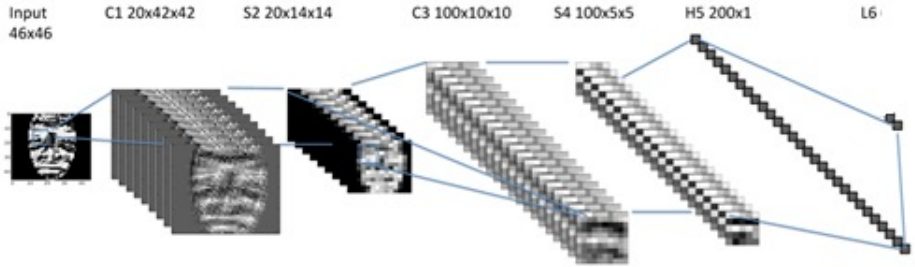
=> To get same expected output in training and test time, we reduce the weights during test time by multiplying them by the dropout probability p=0.5

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

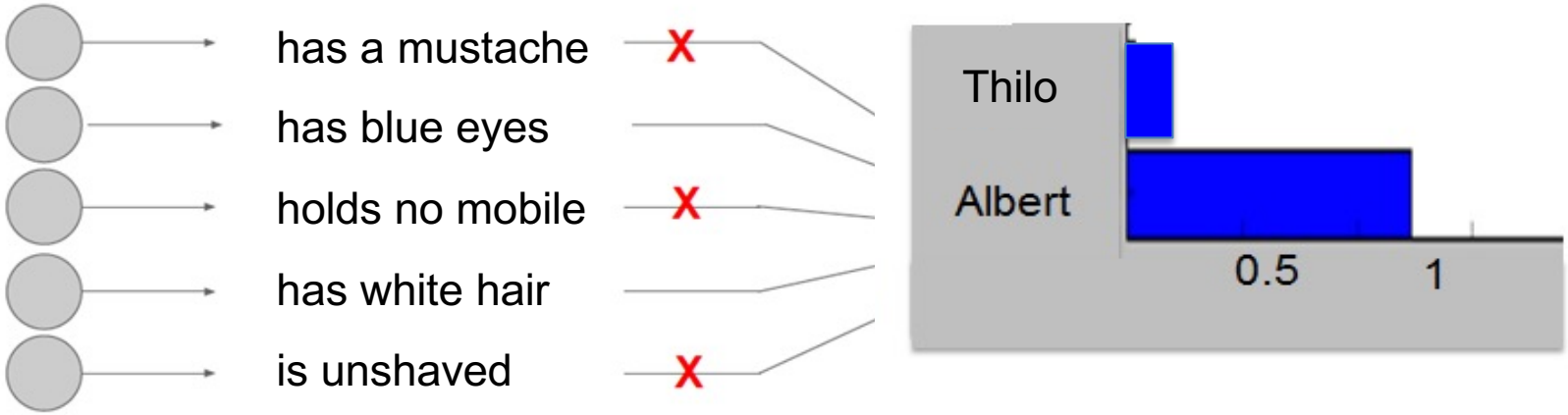
The training data consists of many different pictures of Thilo and Einstein



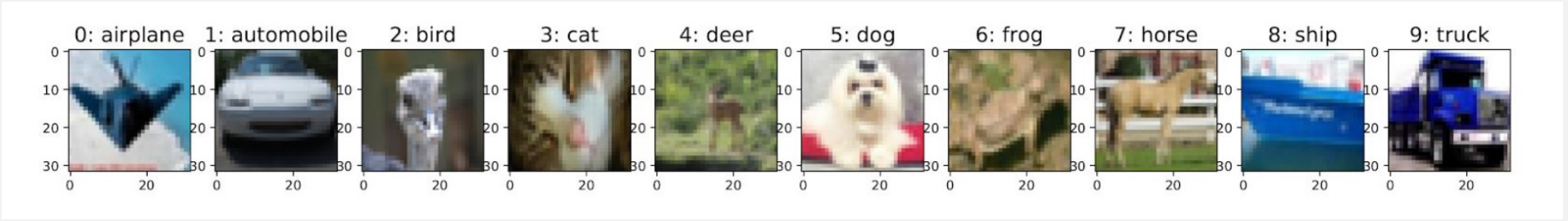
We need a huge number of neurons to extract good features which help to distinguish Thilo from Einstein



Dropout forces the network to learn redundant and independent features

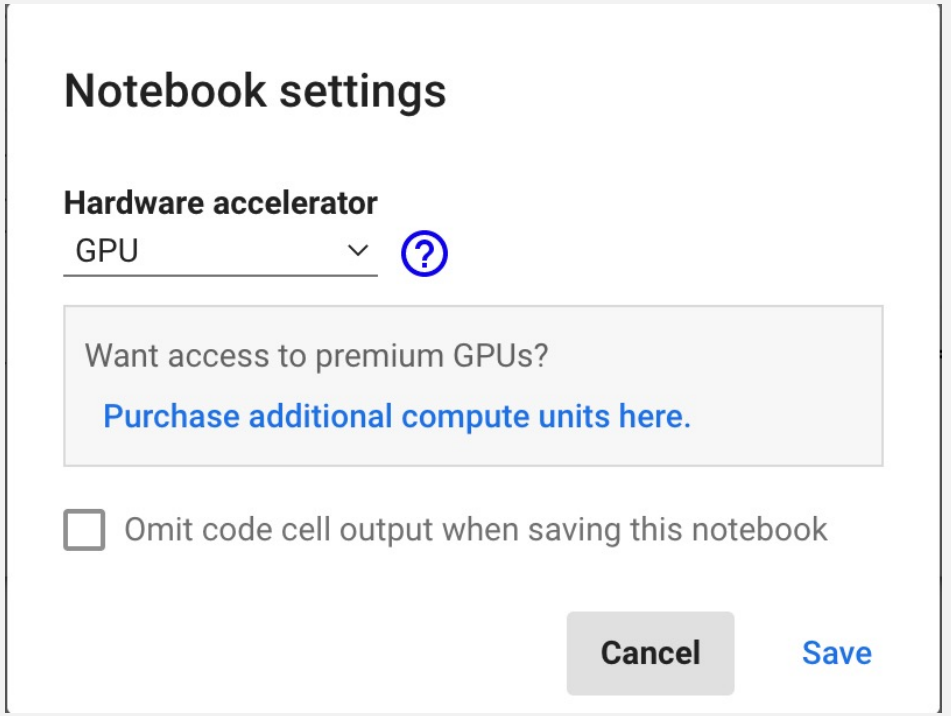


Can batchnorm and dropout improve performance?



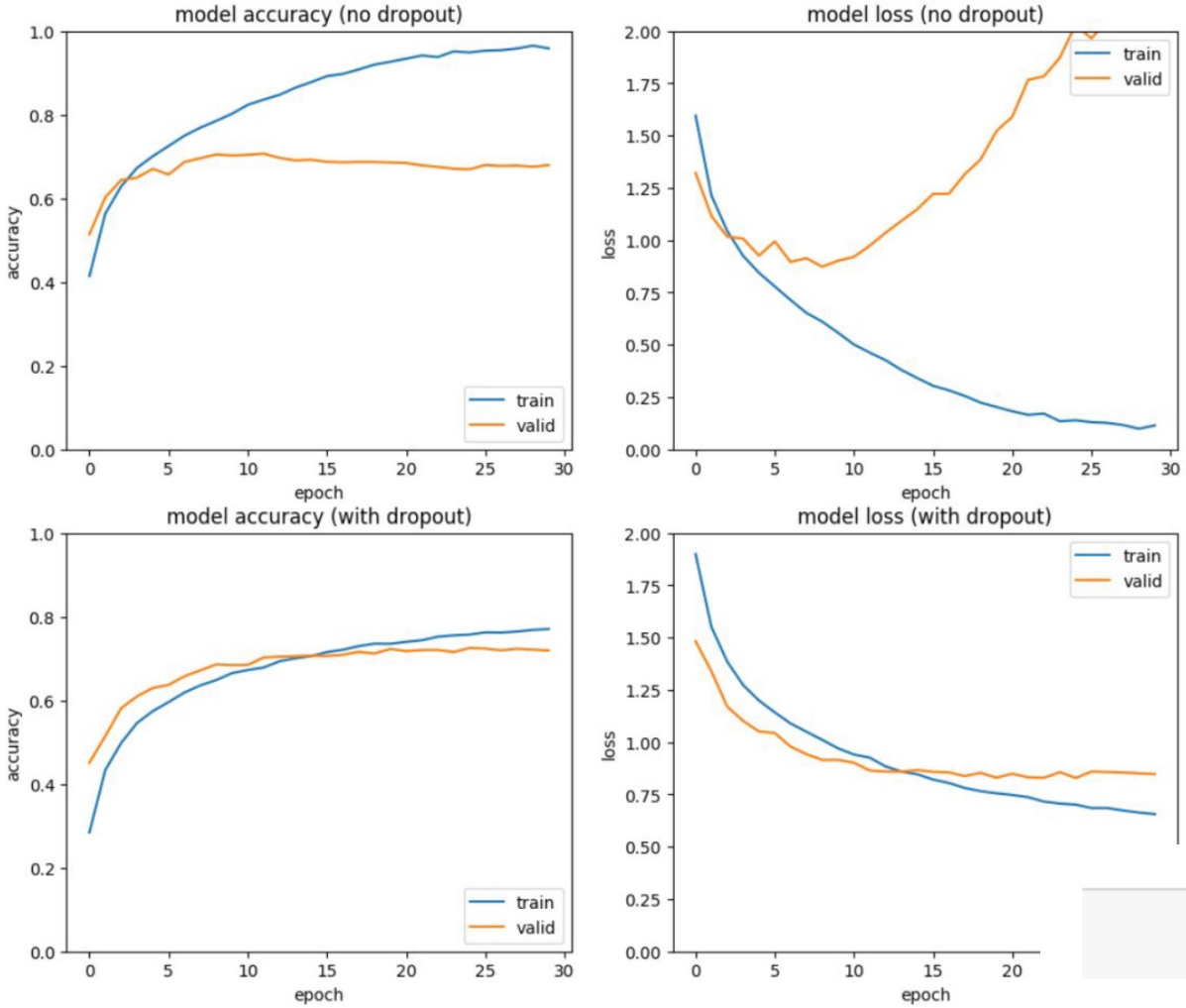
Notebook: 08_cifar10_tricks.ipynb

cnn from scratch
cnn from scratch with dropout
cnn from scratch with batchnorm



28

Results: Dropout fights overfitting in a CIFAR10 CNN



	Acc
rf on pixelvalues	0.3932
cnn from scratch	0.5893
cnn from scratch with dropout	0.6109
cnn from scratch with batchnorm	0.6059

Challenge winning CNN architectures

LeNet-5 1990s: first CNN for ZIP code recognition

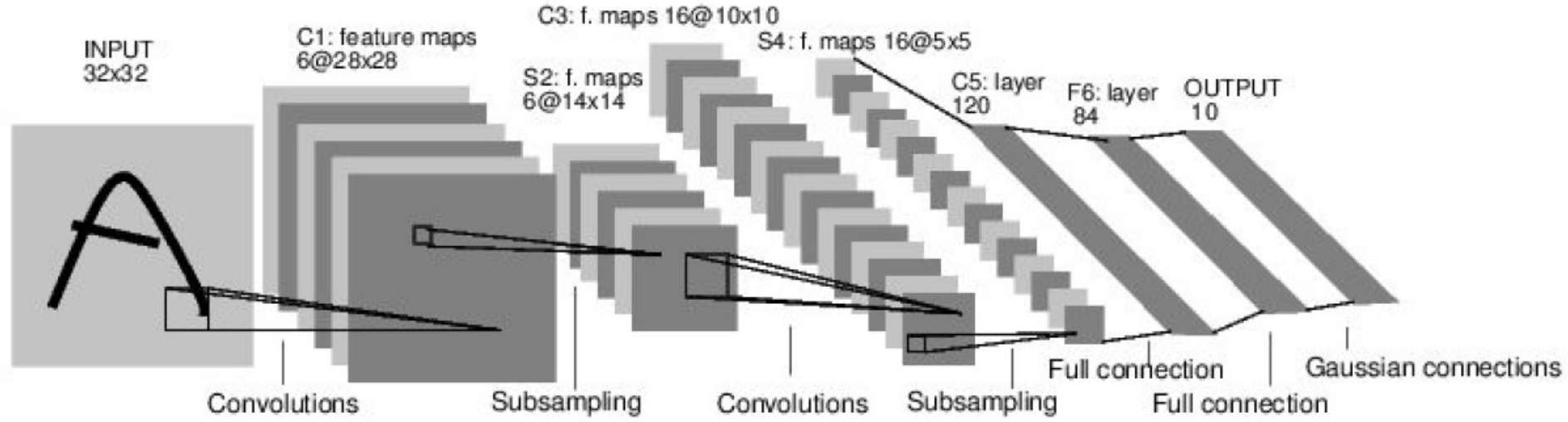


Image credits: http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

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]

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

The first CNN (Yan LeCun)

https://www.youtube.com/watch?v=FwFduRA_L6Q

Review of ImageNet winning CNN architectures

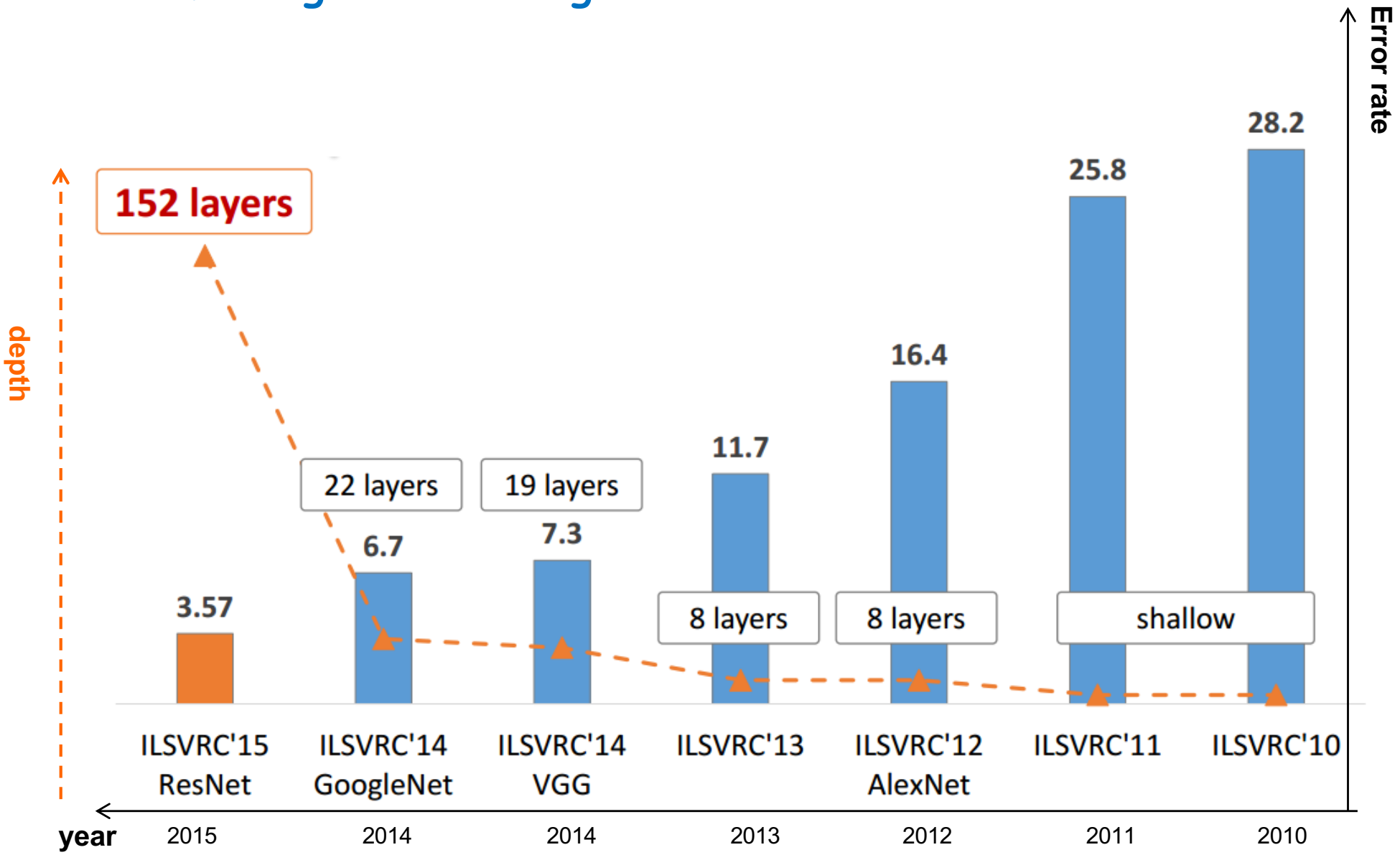


Image credits (modified): K.He, X.Zhang, S.Ren, J.Sun. “Deep Residual Learning for Image Recognition”. arXiv 2015

The trend in modern CNN architectures goes to small filters

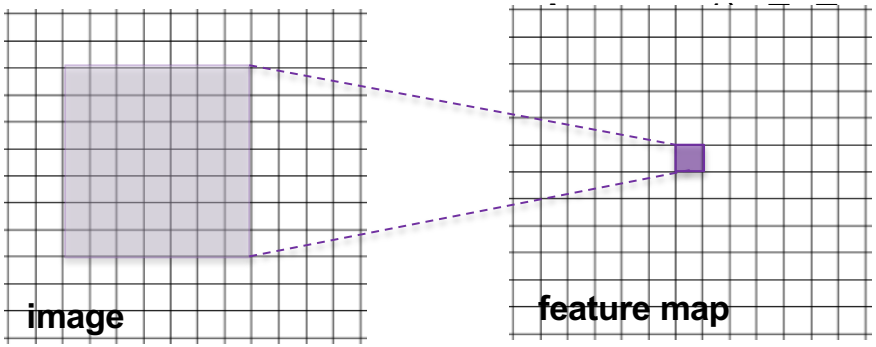
Why do modern architectures use very small filters?

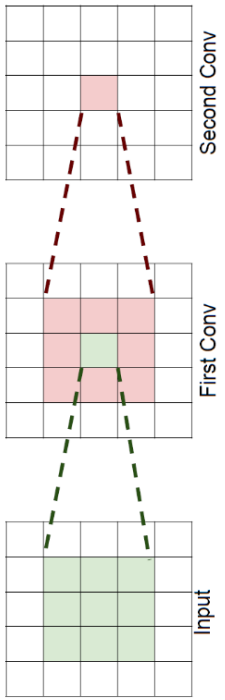
Determine the receptive field in the following situation:

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

49 weights

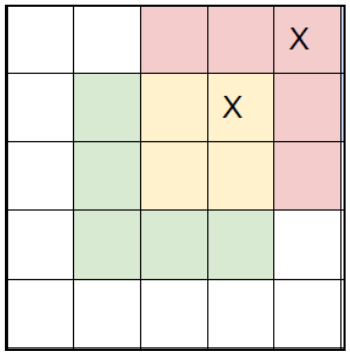
Answer1): 7x7





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

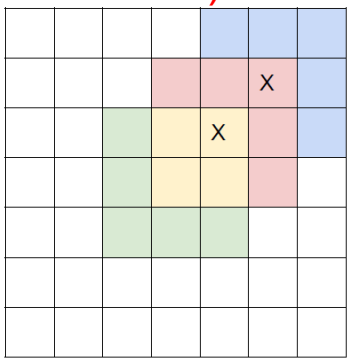
Answer 2): 5x5



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
- 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

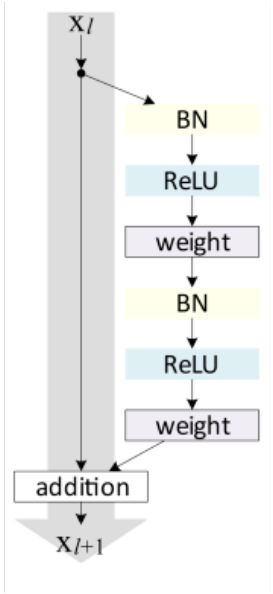
http://arxiv.org/abs/1409.1556

"ResNet" from Microsoft 2015 winner of imageNet

152 layers

ResNet basic design (VGG-style)

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



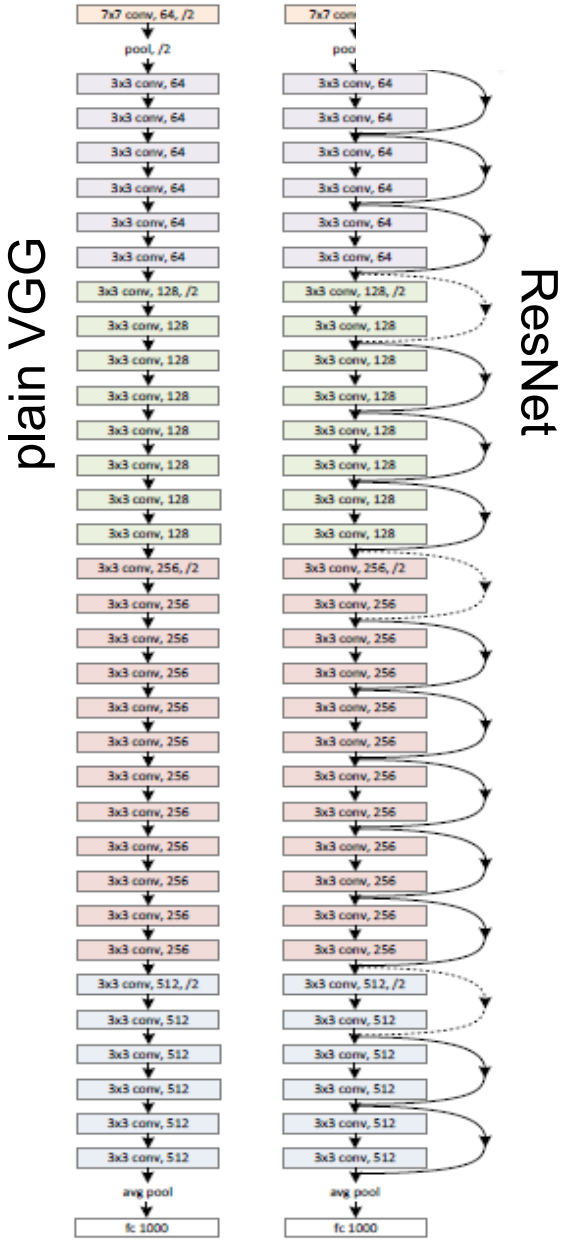
$F(x_l)$

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!

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

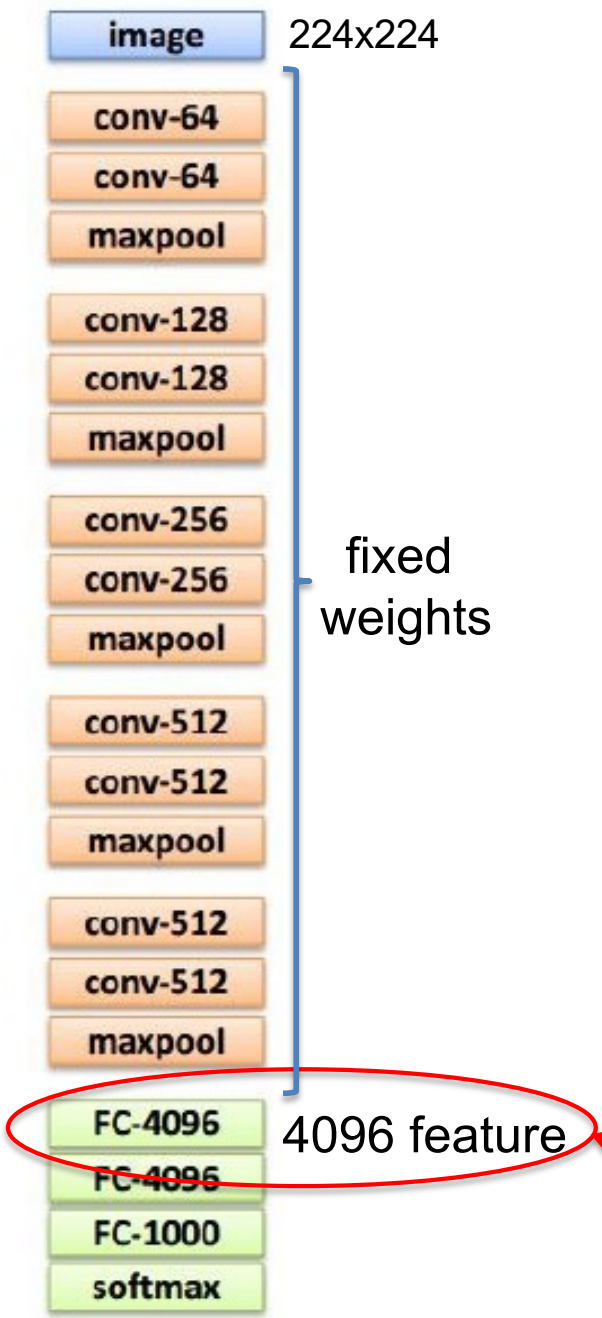
F(x) is called "residual" since it only learns the "delta" which is needed to add to x to get H(x)



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

What to do in case of limited data?

Use pre-trained CNNs for feature generation



- Load a pre-trained CNN – e.g.g VGG16
- Resize image to required size (224x224 for VGG16)
- 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

Are VGG features useful?

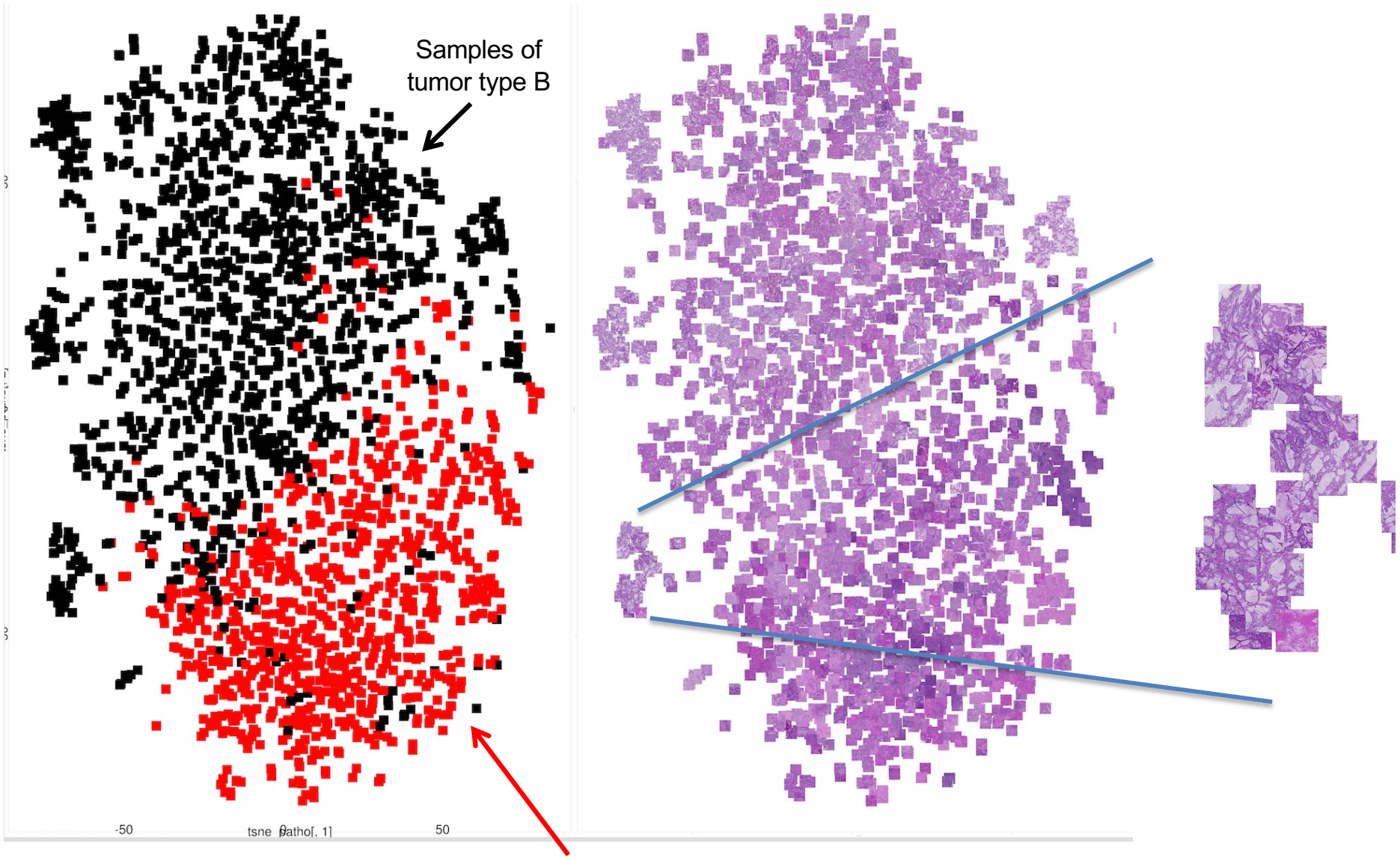
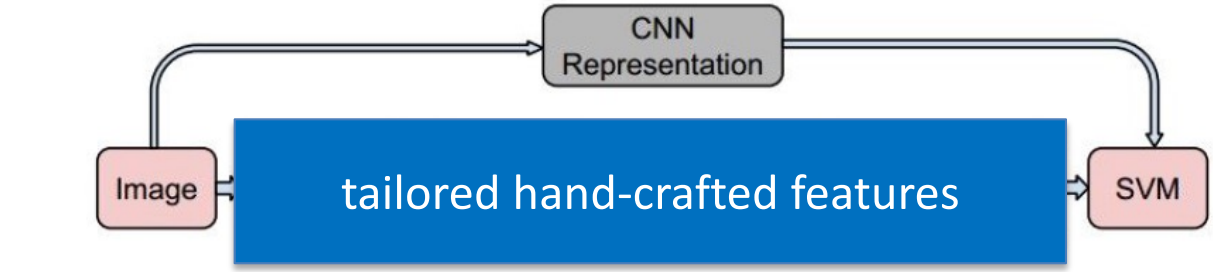
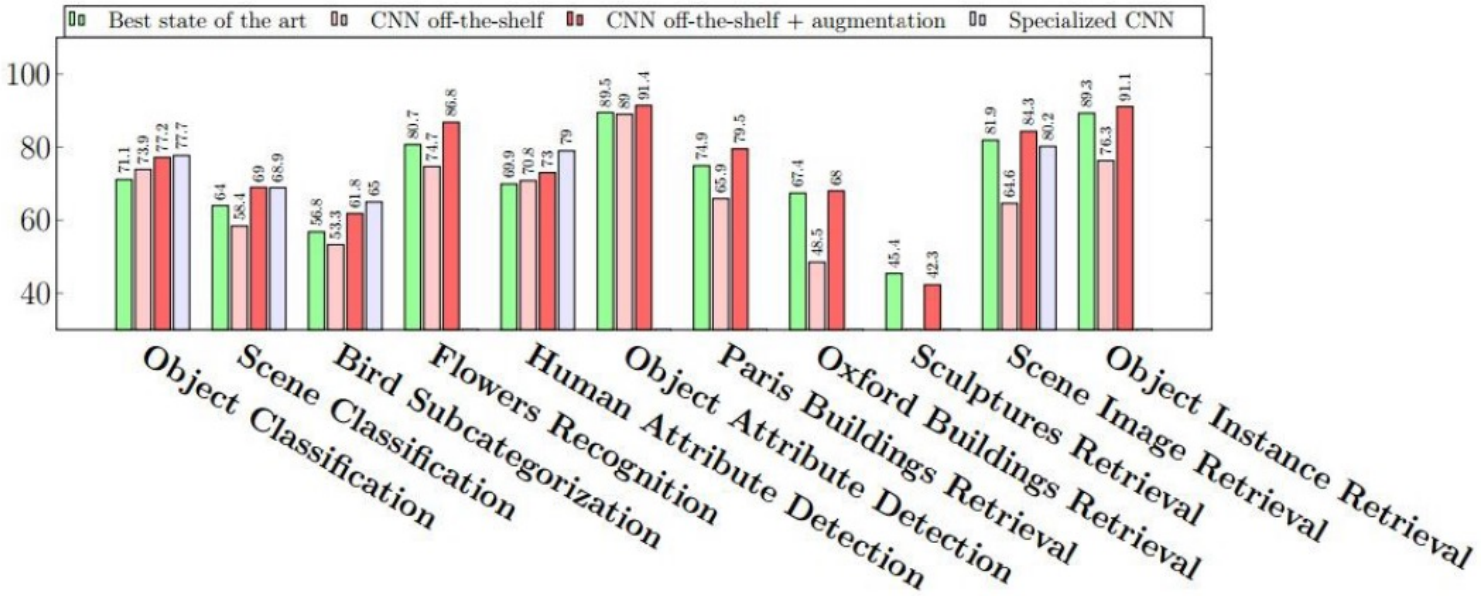


image credit: Elvis Murina

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



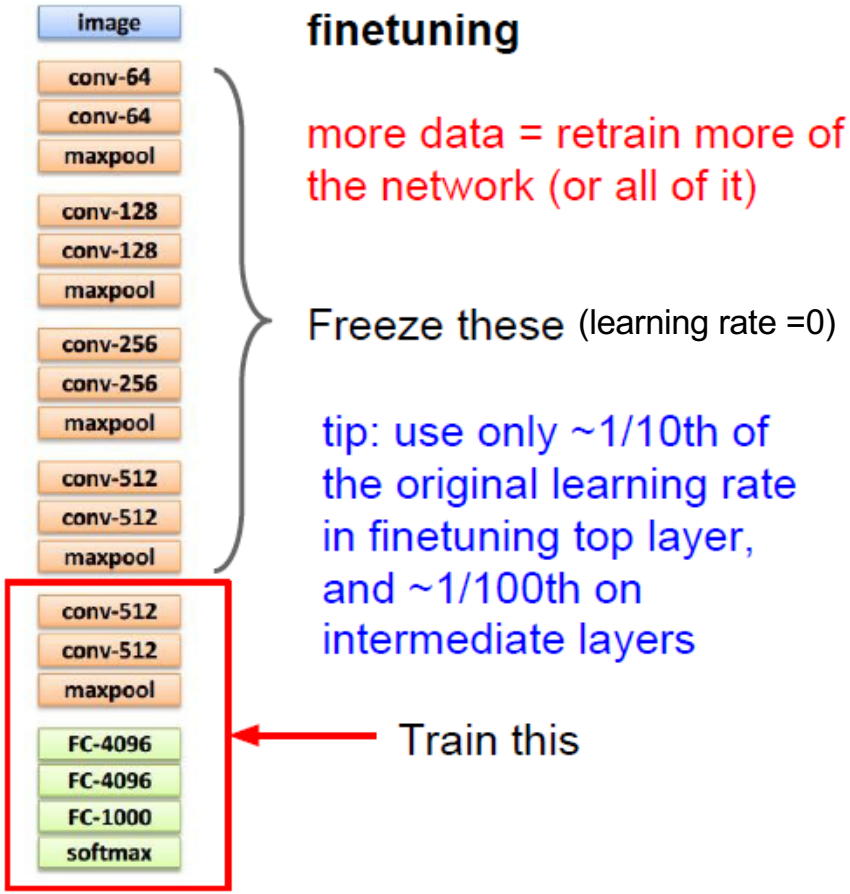


"Astonishingly, we report consistent superior results compared to the highly tuned state-of-the-art systems in all the visual classification tasks on various datasets."

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

Transfer learning beyond using off-shelf CNN feature

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



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

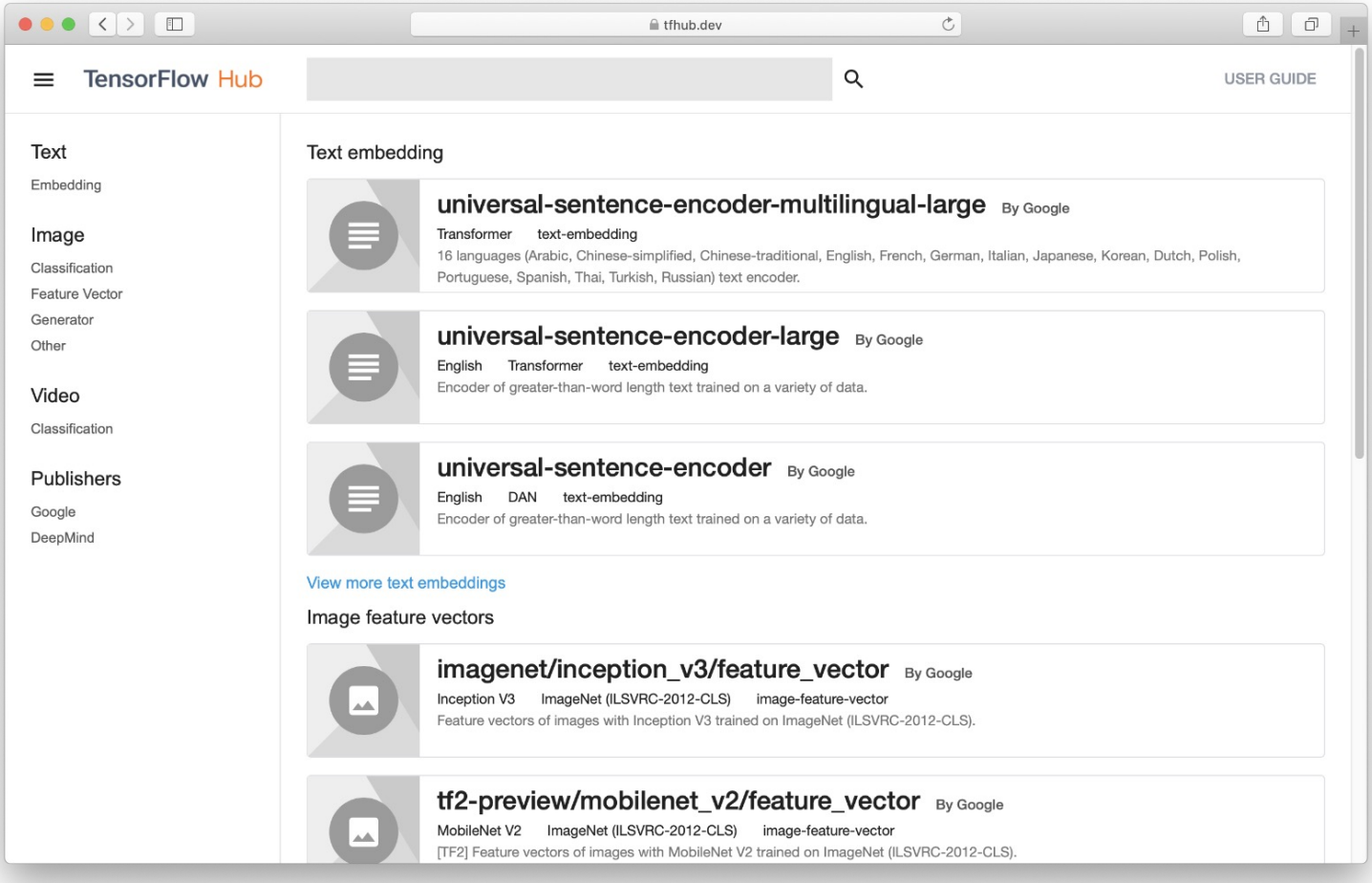
	Similar task (to imageNet challenge)	**Very different task** (to imageNet challenge)
little data	Extract CNN representation of one top fc layer and use these features to train an external classifier	You are in trouble - try to extract CNN representations from different stages and use them as input to 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

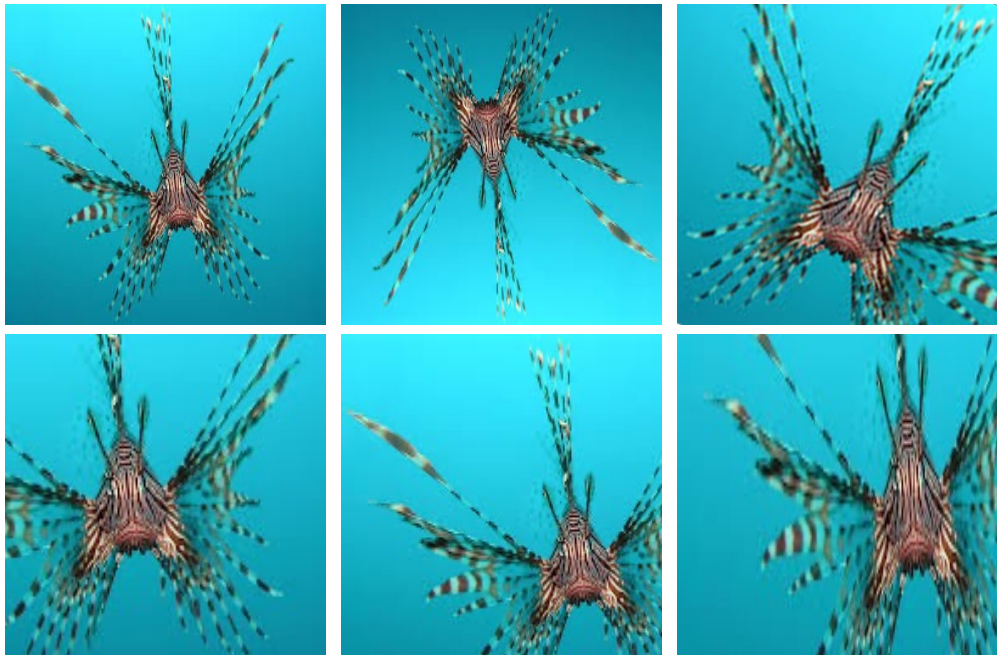
Where to find pretrained networks

- https://tfhub.dev/



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
- ….



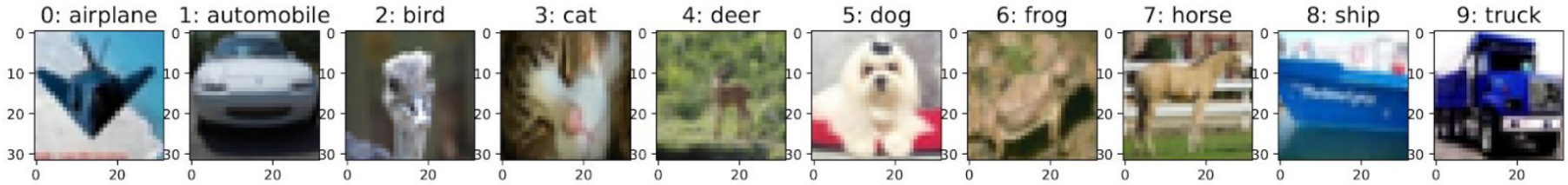
Data augmentation in Keras:

```
from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    vertical_flip=True,
    #zoom_range=[0.1,0.1]
)
```

```
train_generator = datagen.flow(
          x = X_train_new,
          y = Y_train,
          batch_size = 128,
          shuffle = True)
```

```
history = model.fit_generator(
                      train_generator,
                      samples_per_epoch = X_train_new.shape[0],
                      epochs = 400,
                      validation_data = (X_valid_new, Y_valid),
                      verbose = 2,callbacks=[checkpointer]
)
```

Finish 08_Cifar10_Tricks.ipynb



Notebook: https://github.com/tensorchiefs/dl_course_2022/blob/master/notebooks/08_cifar10_tricks.ipynb

Learning with few data

In case of few data you can work with features that you extract from a | inceases the training data and usually help to improve the performace.

Learning with few data

- Baseline model: RF on VGG features
- Transfer learning
- CNN from scratch with Data Augmentation
  - Exercise

44

Results

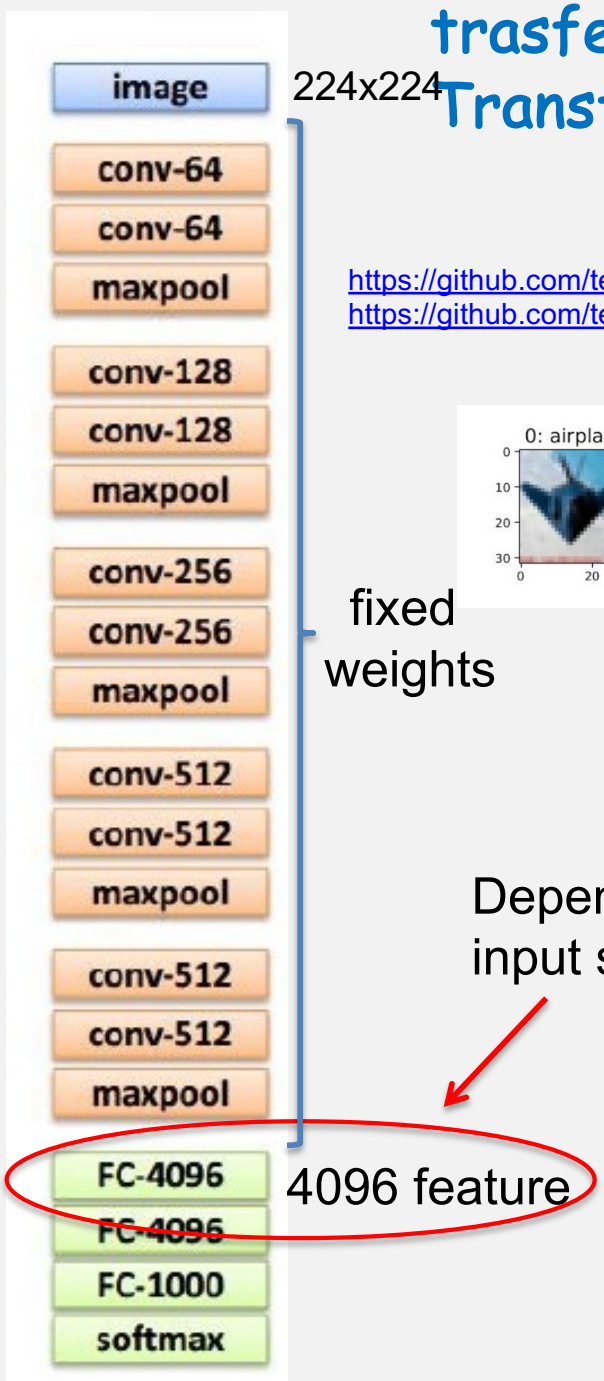
	Acc
rf on pixelvalues	0.3932
cnn from scratch	0.5893
cnn from scratch with dropout	0.6109
cnn from scratch with batchnorm	0.6059
rf on vgg features	0.4632
transfer learning on vgg features	0.5599
cnn from scratch with data augmentation	0.6577

```
1 X_train.shape
```

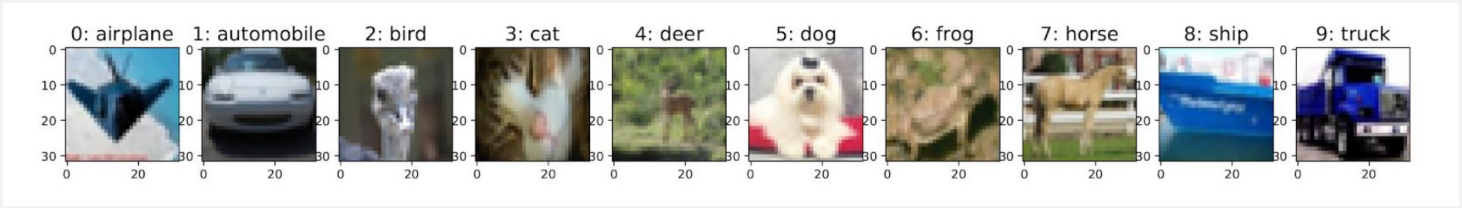
```
(10000, 32, 32, 3)
```

We have enough training data → transfer learning not needed

If you are interested in a situtation in which trasfer learning helps
Transfer learning for DL with few data



https://github.com/tensorchiefs/dl_course_2021/blob/master/notebooks/08b_classification_few_labels.ipynb
https://github.com/tensorchiefs/dl_course_2021/blob/master/notebooks/08b_classification_few_labels_solution.ipynb



- Only 100 labeled cifar10 images
- Use pixel values as features
- Use a pretrained VGG to get better features

46

Summary

- 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
  - Use batch-norm to get into the sweet-spot of the activation function (ReLu)
  - Use skip connections to avoid gradient vanishing

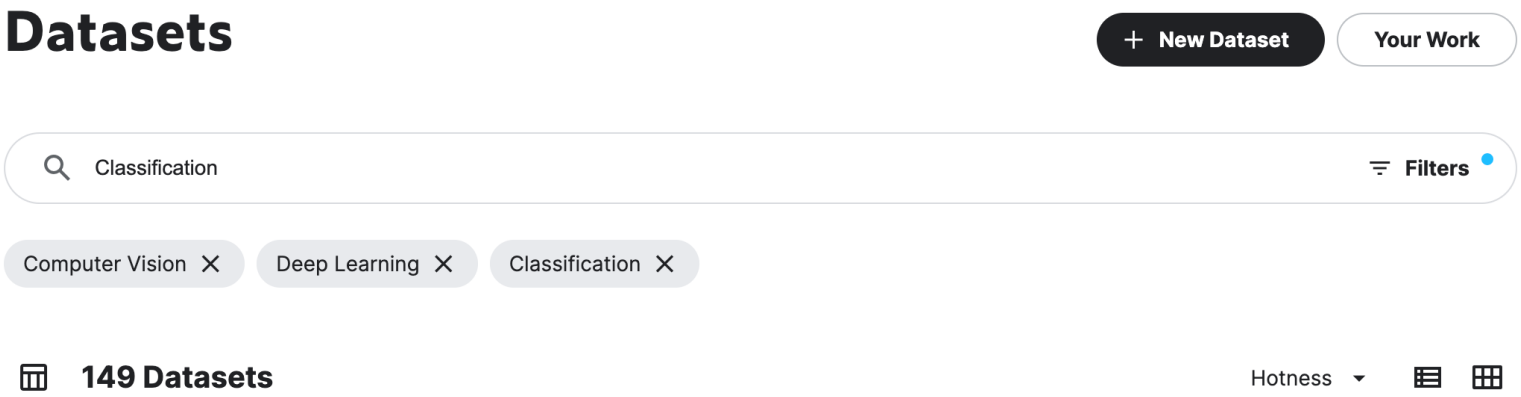
- Always use a simple baseline model (eg RF) as benchmark

- In case of few data
  - use data augmentation
  - use a pretrained CNN (eg imageNet VGG) as feature extractor or for finetuning

- 1D CNNs can be used in case of sequence data
  - for forecasting application use “causal” variants which only look back
  - dilation allows to increase the receptive field without needing more weights

Projects

- In principle you have the knowledge
- Team up
- Look for interesting data
  - You have a data set you like to analyze (talk to me)
  - Find a competion at Kaggle or a data set
  - https://www.kaggle.com/datasets?search=Classification&tags=13310-Deep+Learning%2C13302-Classification



  - If you find a dataset without classification thats also fine