Machine Intelligence:: Deep Learning
Week 3

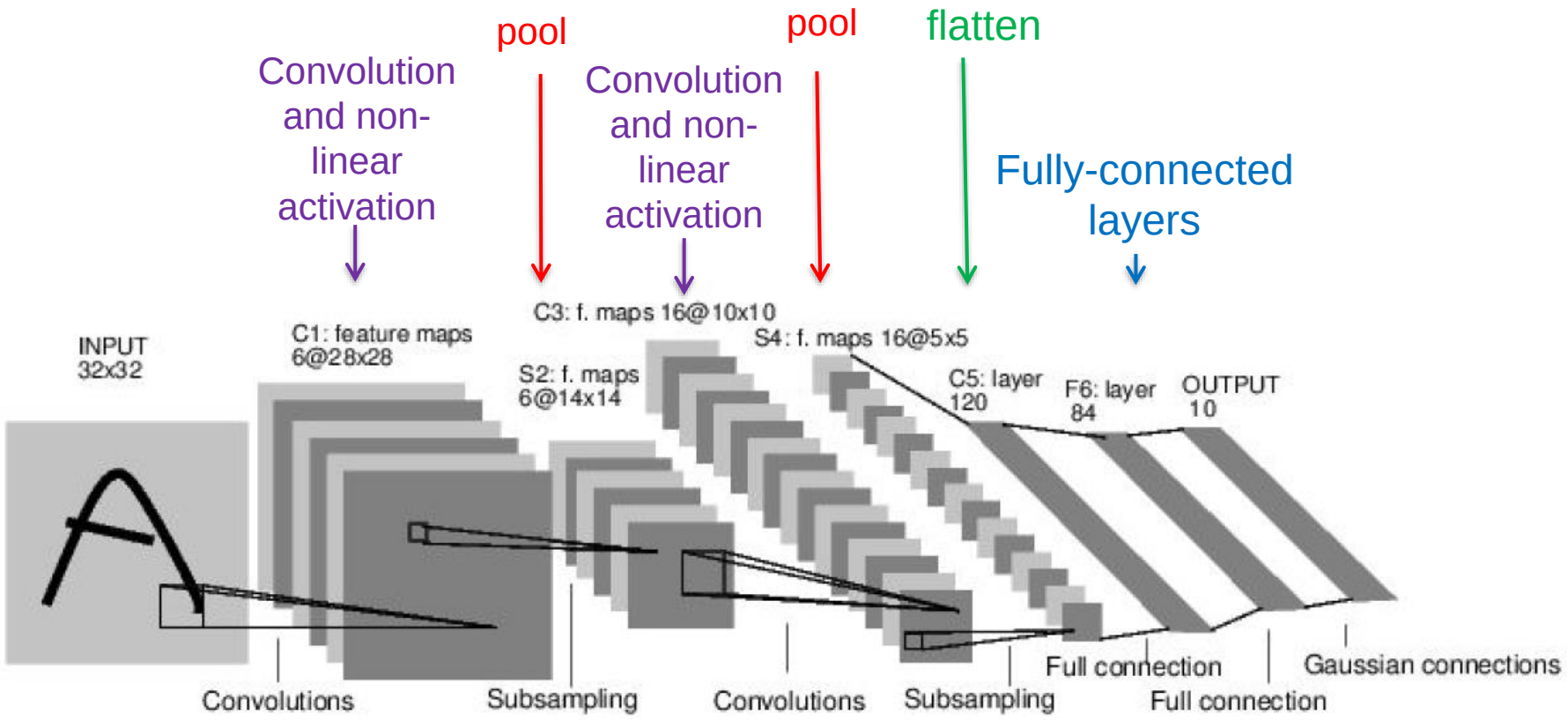
Beate Sick, Loran Avci, Oliver Dürr

Institut für Datenanalyse und Prozessdesign
Zürcher Hochschule für Angewandte Wissenschaften

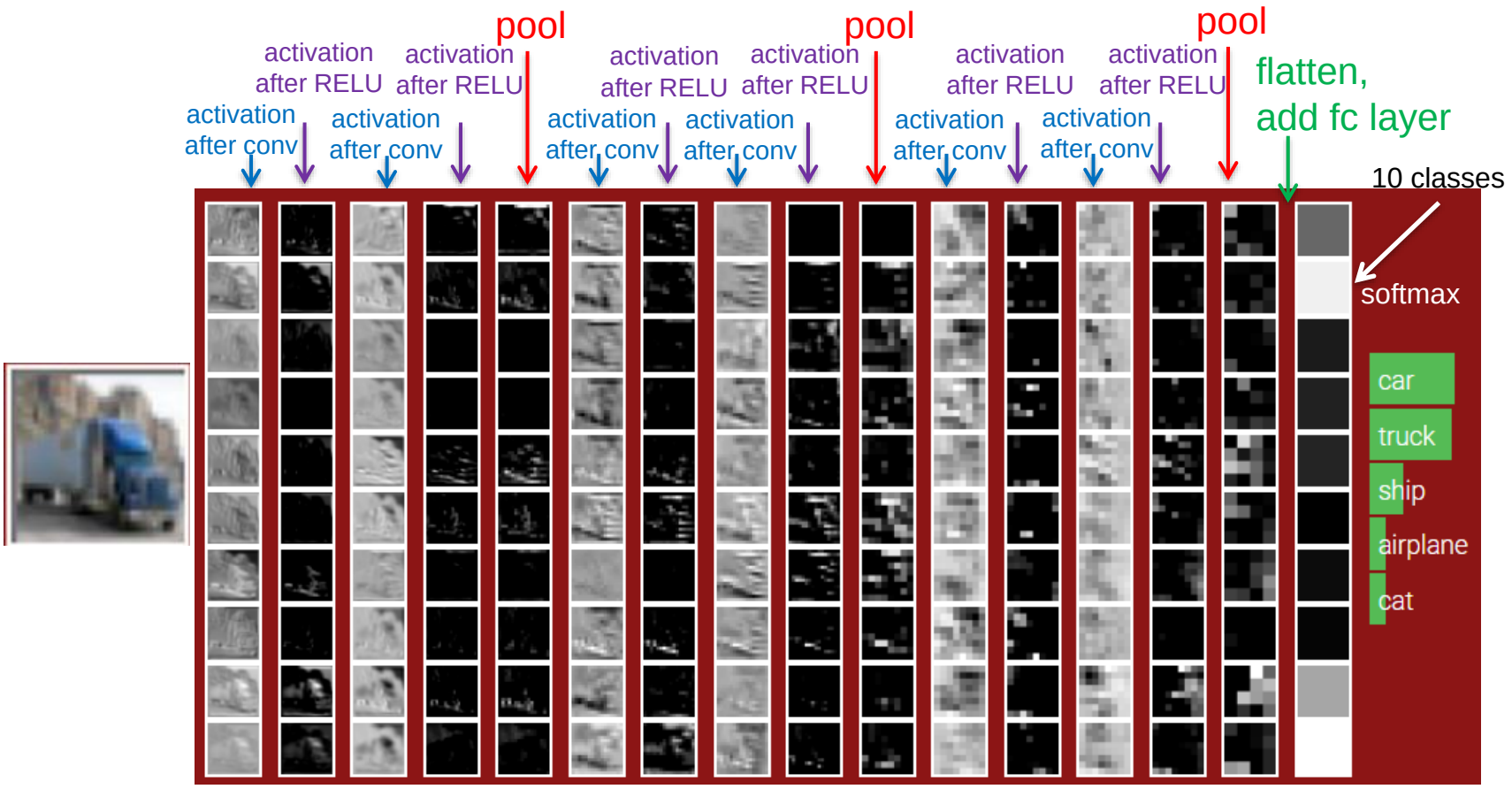
Topics of today

- Biological Inspiration of CNNs
- Tricks of the trade in CNNs
    - Standardizing / Batch-Norm
    - Dropout
- What does a CNN look at?
    - shining light into the black box
- Challenge winning CNN architectures
- DL with few data:
    - Transfer learning
    - Augmentation

Typical architecture of a simple CNN



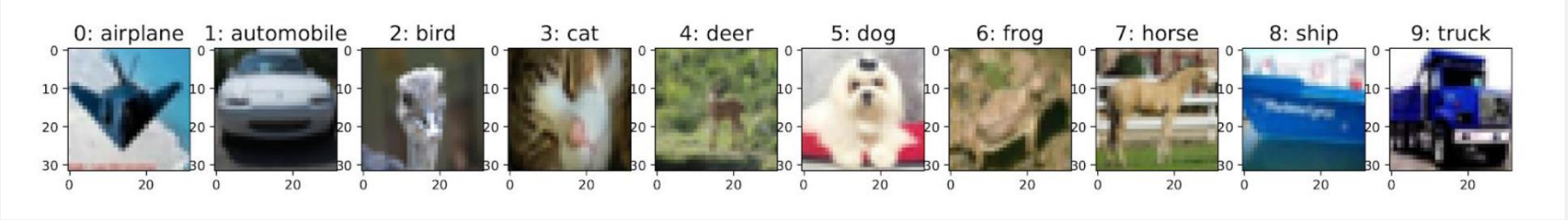
A CNN at work



Activation maps give insight on the spatial positions where the filter pattern was found in the input one layer below (in higher layers activation maps have no easy interpretation)
-> only the activation maps in the first hidden layer correspond directly to features of the input image.

http://cs231n.stanford.edu/

Looking back at Homework: Develop a CNN for cifar10 data



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

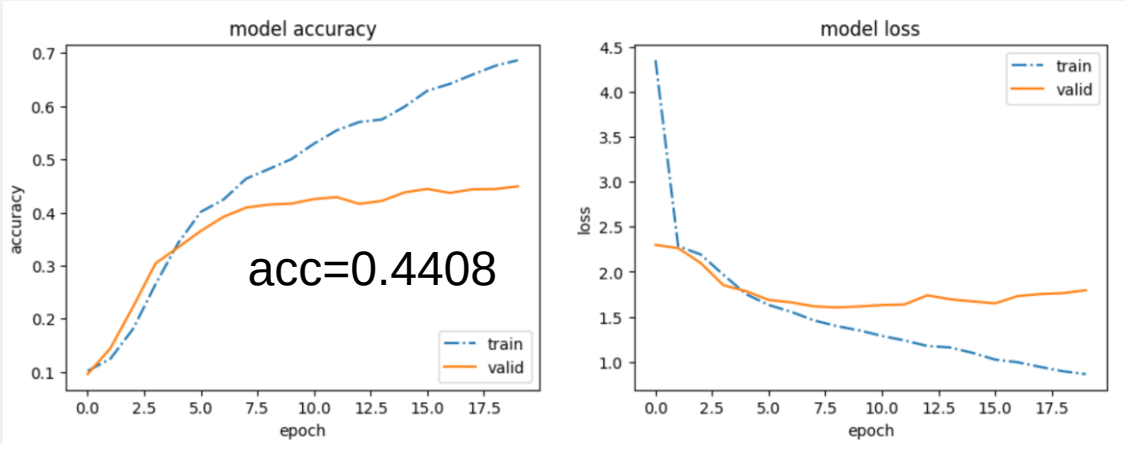
Investigate the impact of standardizing the data on the performance

Notebook:
https://github.com/tensorchiefs/dl_course_2021/blob/master/notebooks/07_cifar10_norm_sol.ipynb

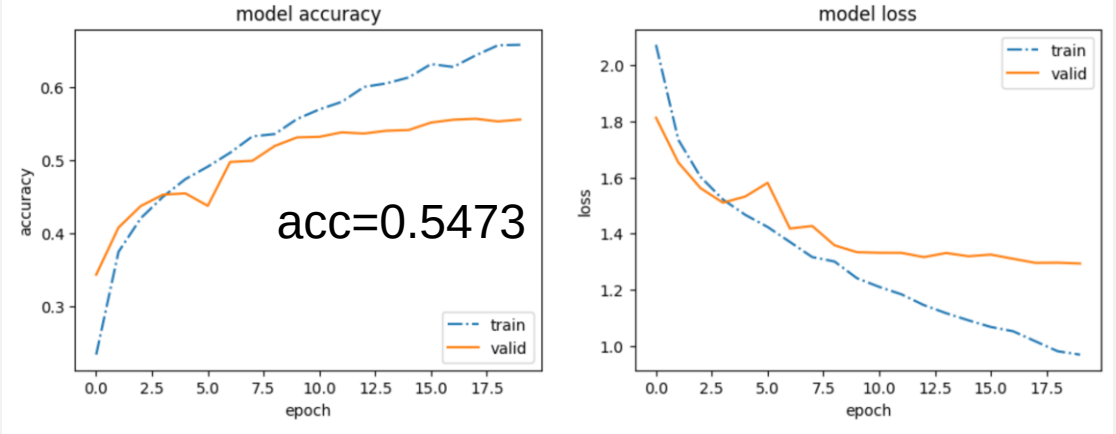
Take-home messages from the homework

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

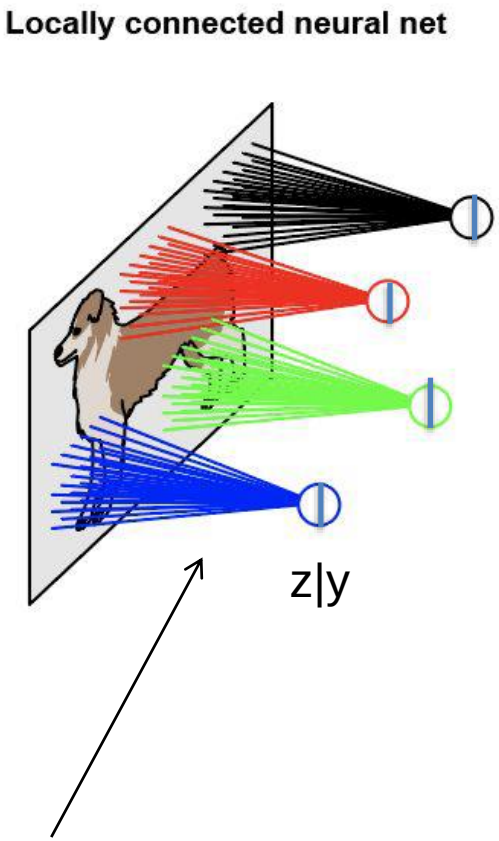
Without standardizing the input to the CNN



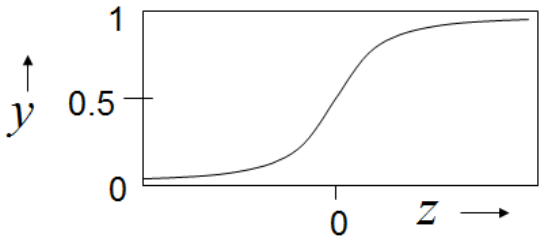
After standardizing the input to the CNN



Shared weight and its consequences: standardize input



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



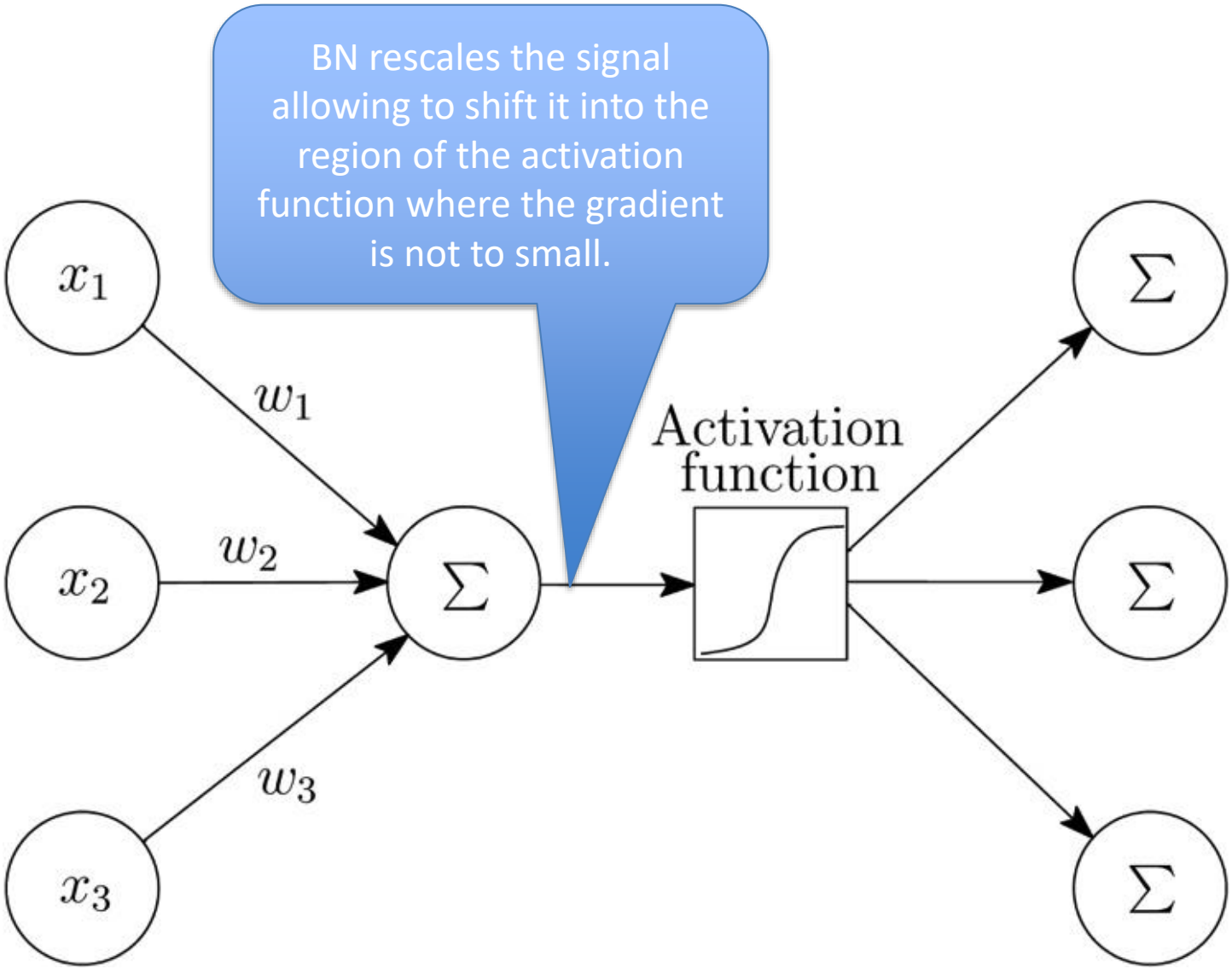
Shared weights: by using the same weights for each patch of the image

→ To get for many patches y-values close to the sweet-spot of the activation function, the x-values should not vary to much (it cannot be compensated by changed weights when weights are shared).

Data Standardization

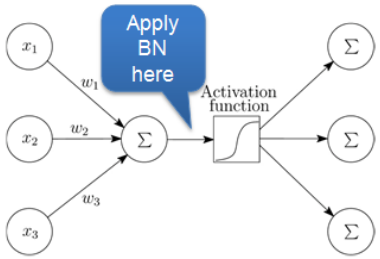
Batch Norm Layer

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

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

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

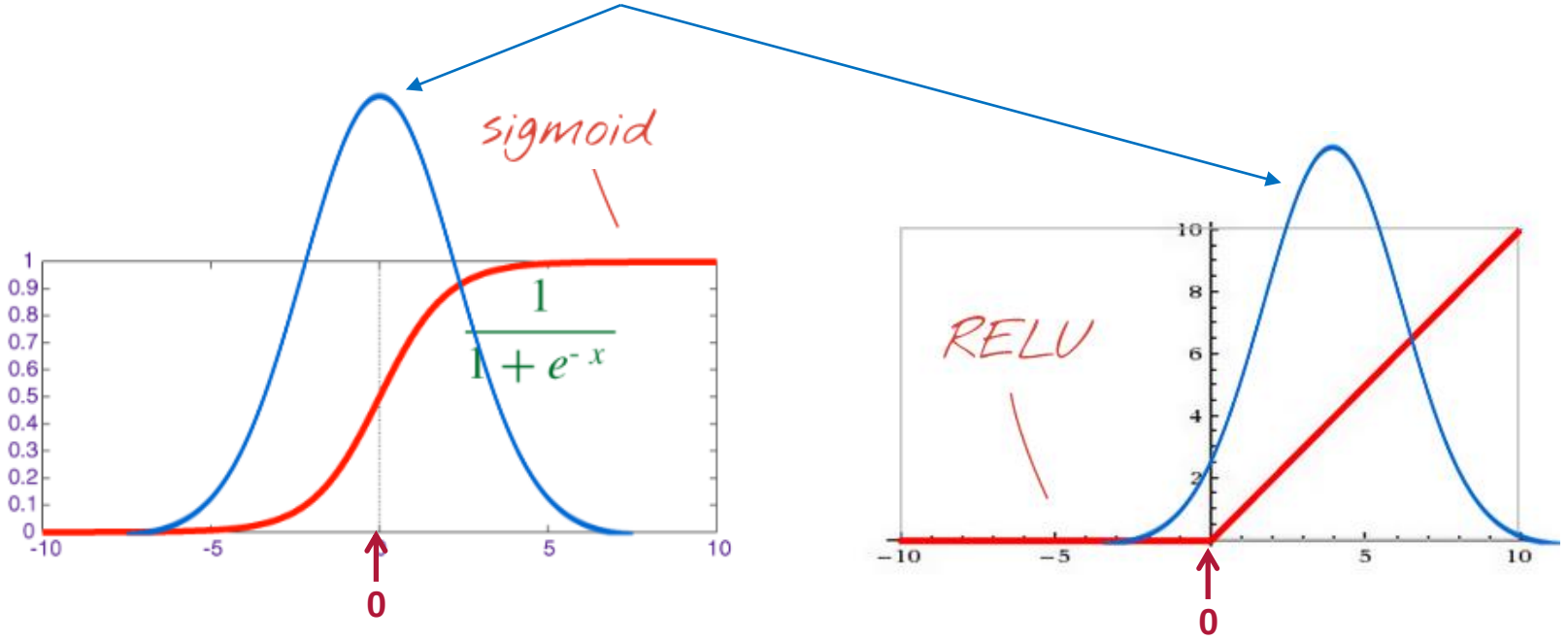
Step 2: $$BN(x) = \alpha\hat{x} + \beta$$

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.



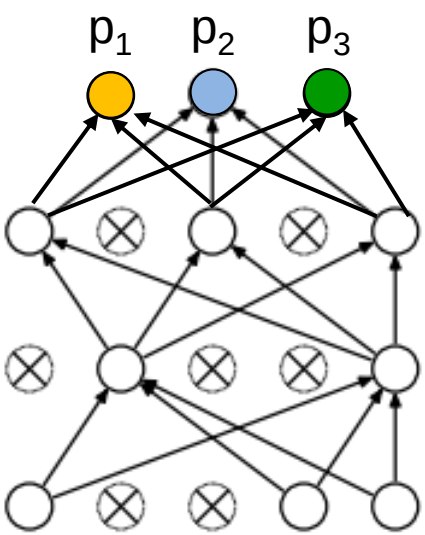
When using BN consider the following:

- Using a higher learning rate might work better
- Use less regularization, e.g. reduce dropout probability
- In the linear transformation the biases can be dropped (step 2 takes care of the shift)
- In case of ReLu only the shift β in steps 2 need to be learned (α can be dropped)

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

Dropout during training

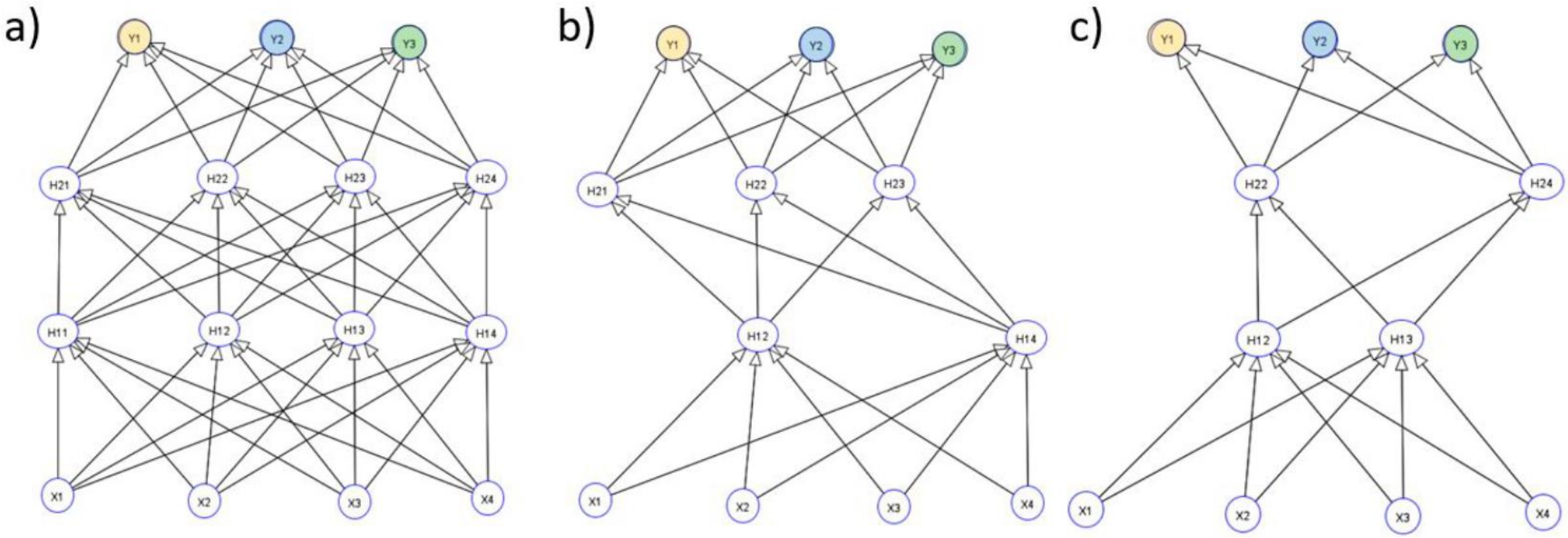
Dropout helps to fight overfitting



Using dropout during training implies:

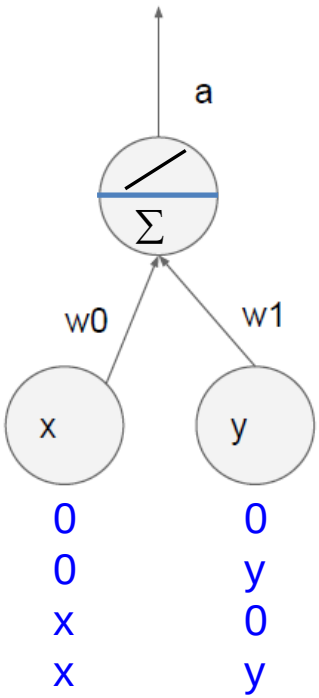
- In each training step only weights to not-dropped units are updated → we train a sparse sub-model NN
- For predictions with the trained NN we freeze the weights corresponding to averaging over the ensemble of trained models we should be able to “reduce noise”, “overfitting”

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



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**

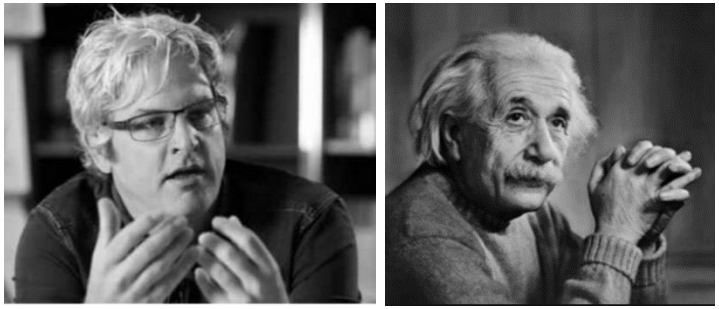
during training with dropout probability 0.5:

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)**

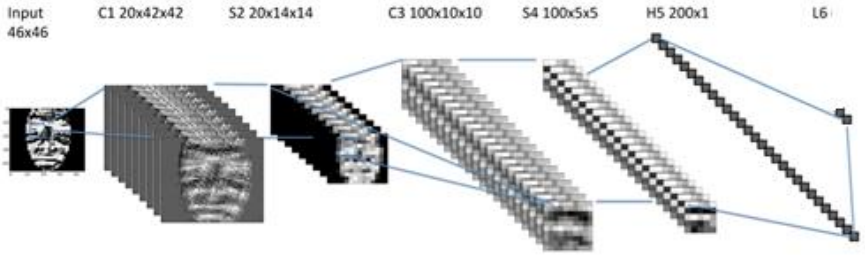
=> To get same expected output in training and test time, we reduce the weights during test time by multiplying them by the dropout probability p=0.5

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

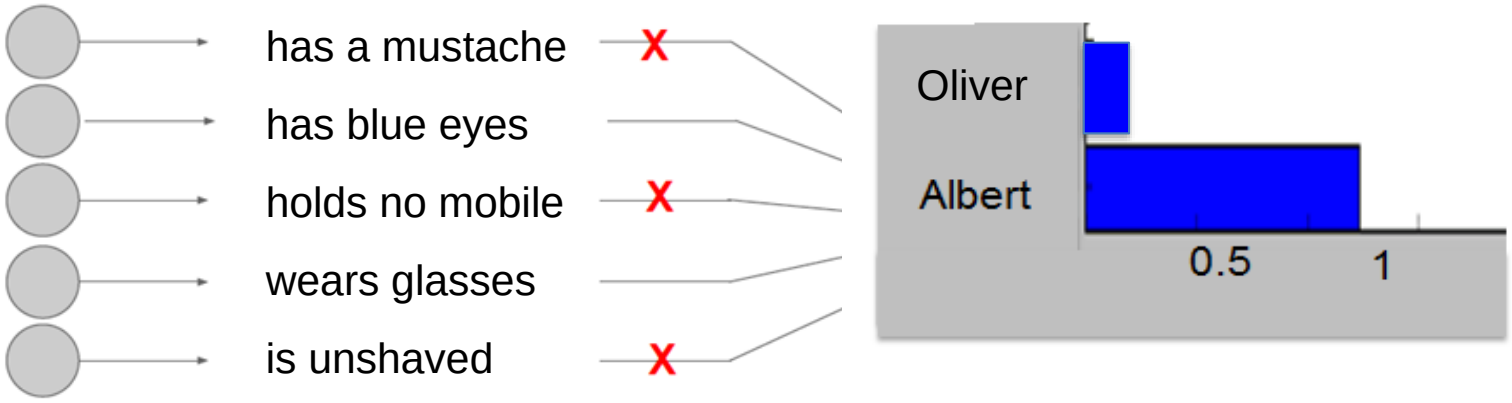
The training data consists of many different pictures of Oliver and Einstein



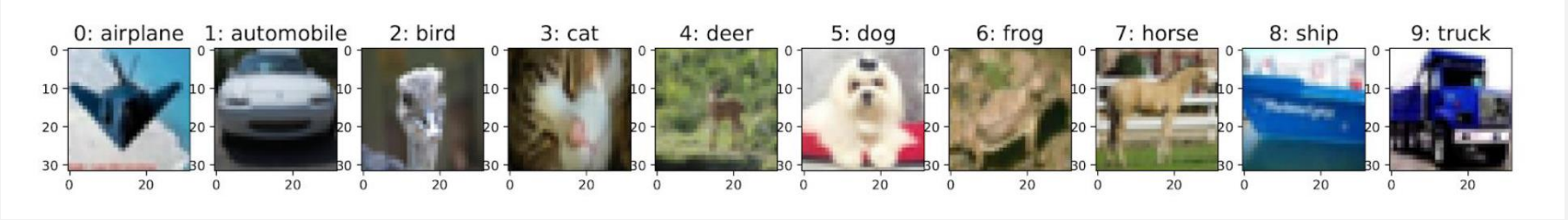
We need a huge number of neurons to extract good features which help to distinguish Oliver from Einstein



Dropout forces the network to learn redundant and independent features



Can batchnorm and dropout improve performance?

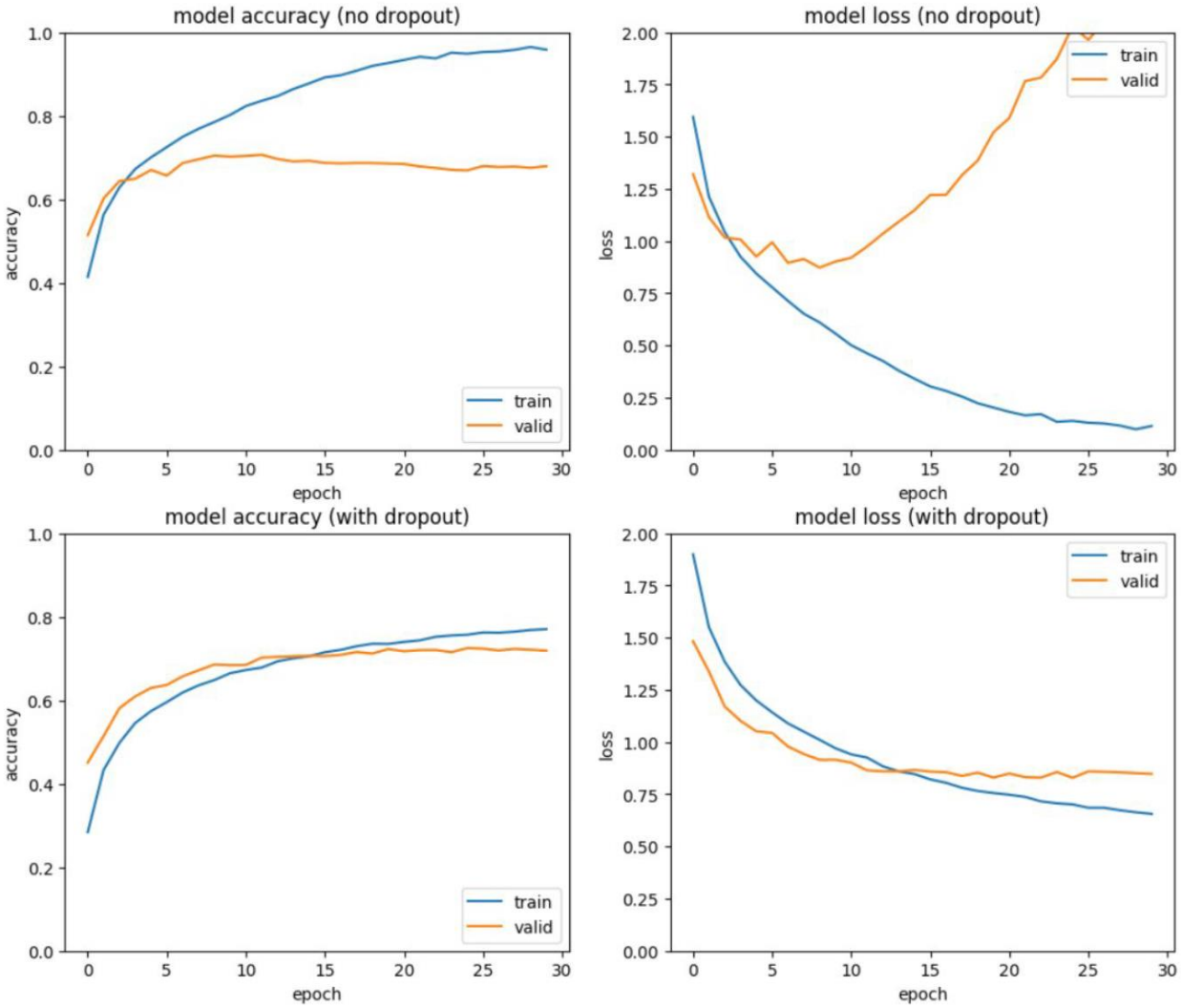


Notebook: https://github.com/tensorchiefs/dl_course_2021/blob/master/notebooks/08_cifar10_tricks.ipynb

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

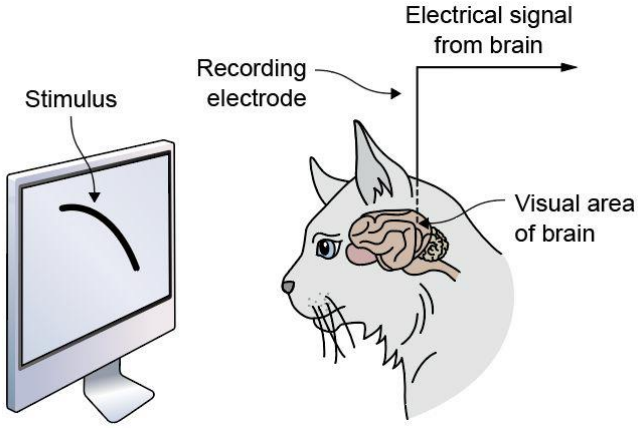
17

Dropout fights overfitting in a CIFAR10 CNN

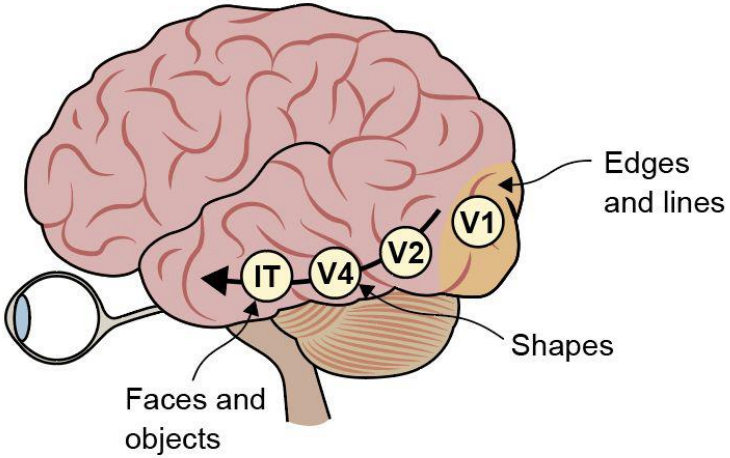


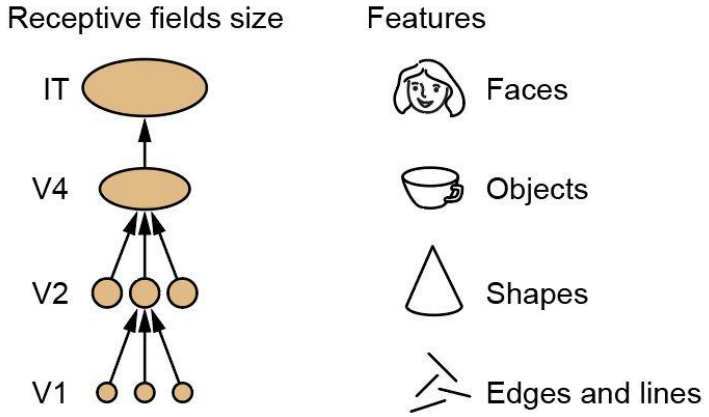
Biological Inspiration of CNNs

How does the brain respond to visually recieved stimuli?



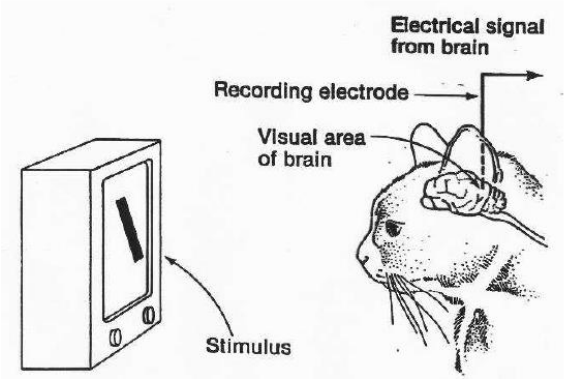
Setup of the experiment of Hubel and Wiesel in late 1950s in which they discovered neurons in the visual cortex that responded when moving **edges** were shown to the cat.

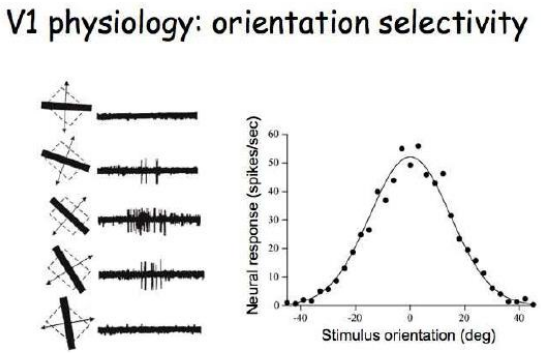


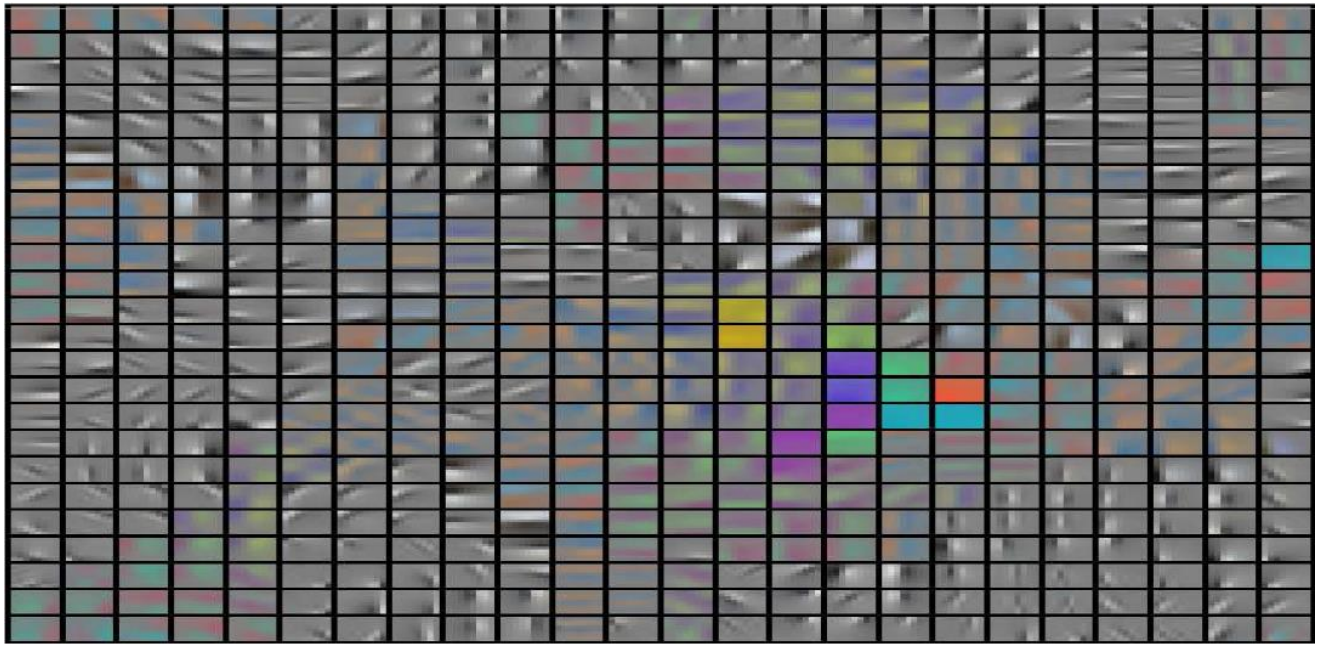


Organization of the visual cortex in a brain, where neurons in different regions respond to more and more complex stimuli

Compare neurons in brain region V1 in first layer of a CNN





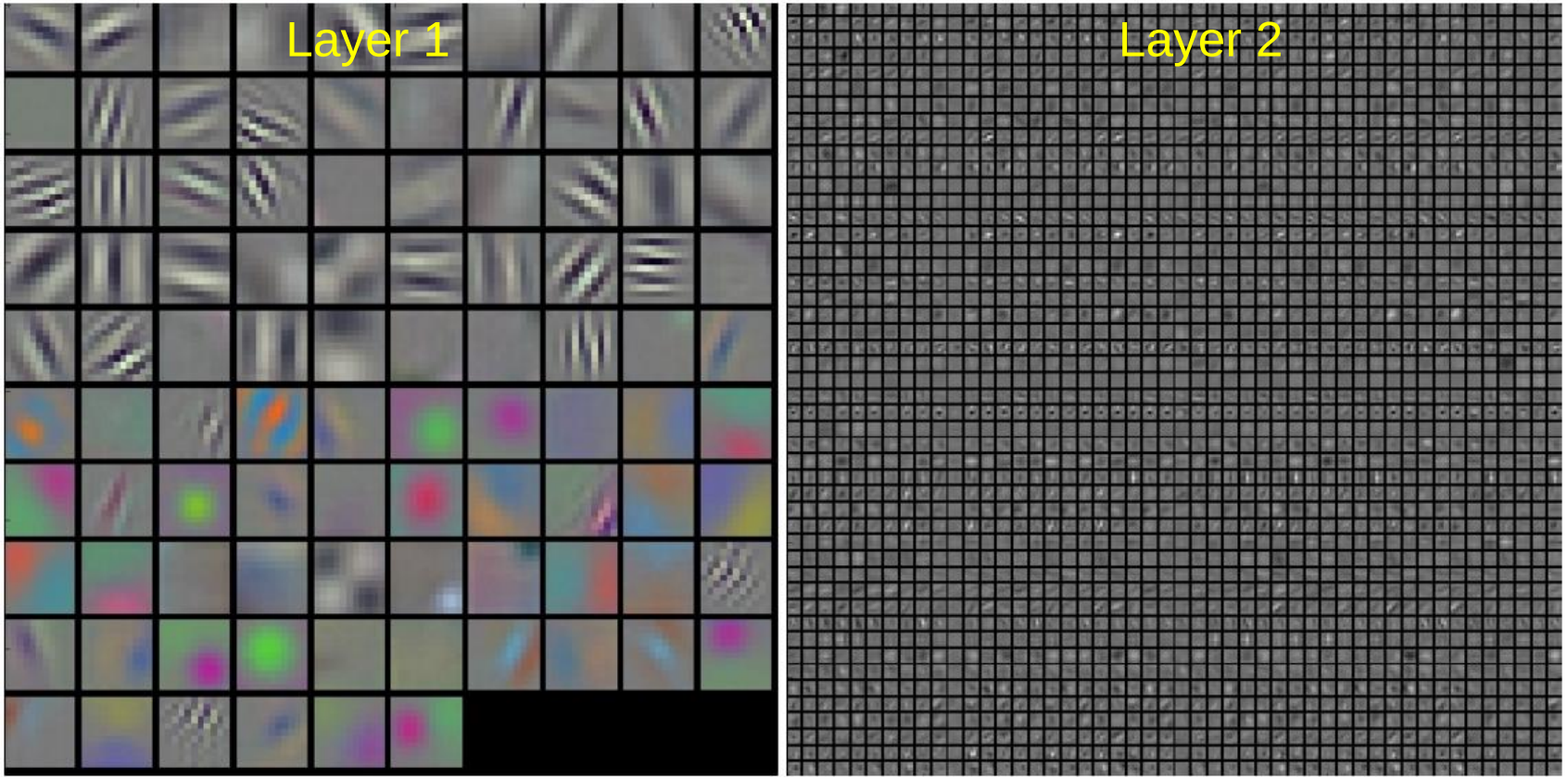


Neurons in brain region V1 and neurons in 1. layer of a CNN respond to similar patterns

Slide (modified) credits: Nando de Freitas

Visualize the weights used in filters

Filter weights from a trained Alex Net



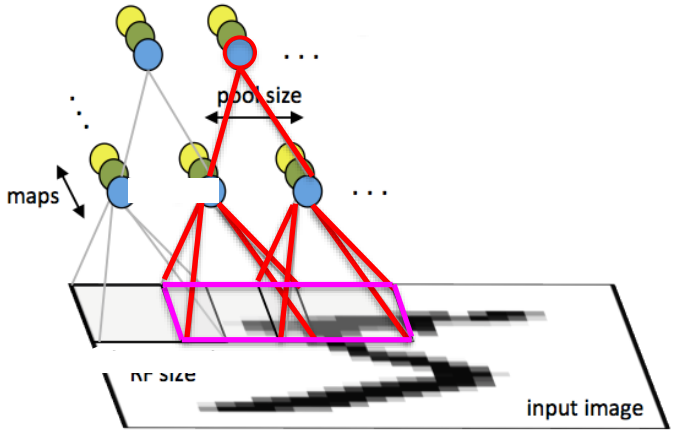
Only in layer 1 the filter pattern correspond to extracted patterns in the image.

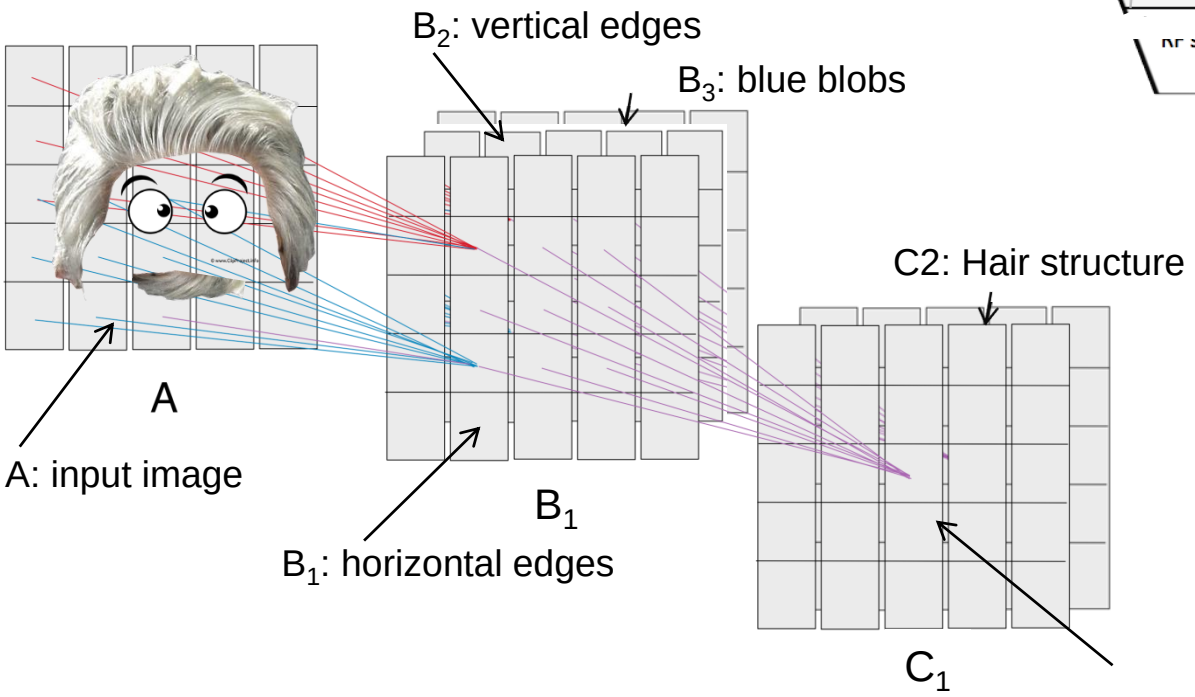
In higher layers we can only check if patterns look noisy, which would indicate that the network that hasn't been trained for long enough, or possibly with a too low regularization strength that may have led to overfitting.

Image credits: http://cs231n.github.io/understanding-cnn/

The receptive field

For each pixel of a feature map we can determine the connected area in the input image – this area in the input image is called receptive field.

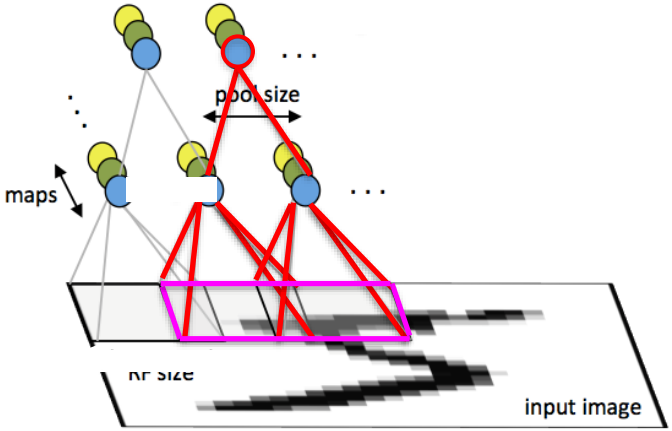


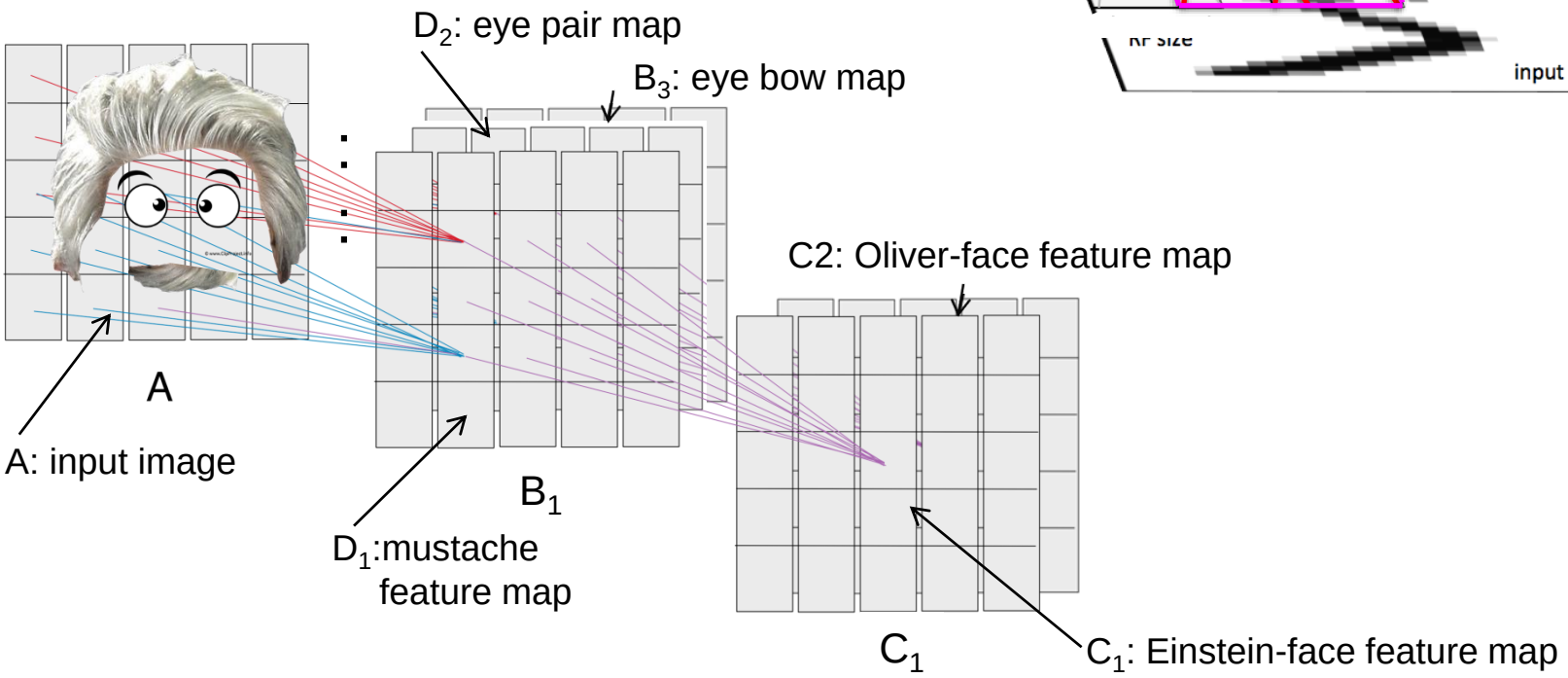


An activation map gets activated by a certain structure of the feature maps one layer below, which by itself depends on the input of a lower layer etc and finally on the input image. Activation maps close to the input image are activated by simple structures in the image, higher maps by more complex image structures.

The receptive field

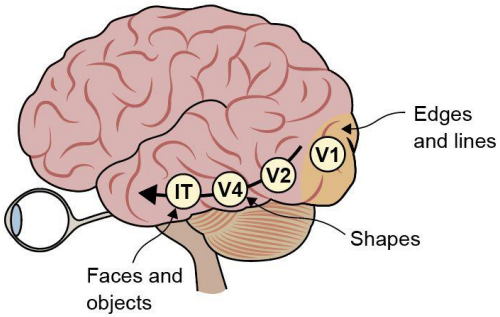
The receptive field gets larger and larger when going further away from the input

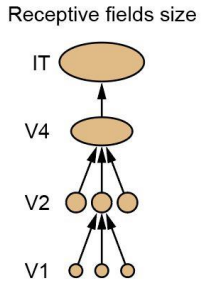


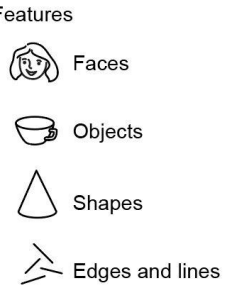


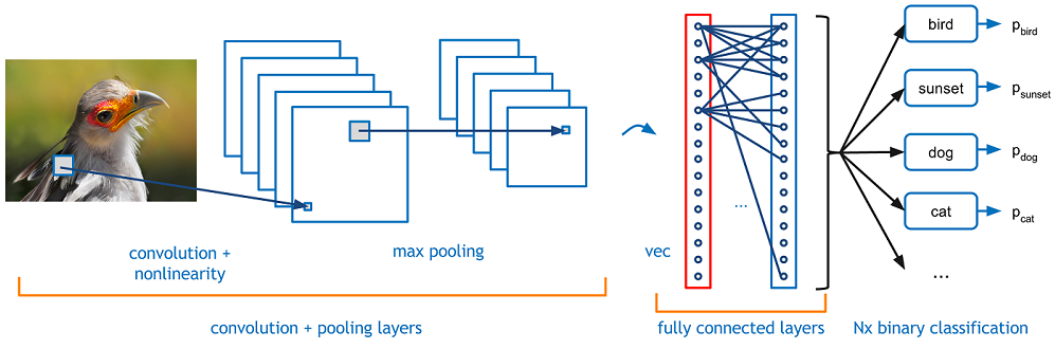
Filter cascade across different channels can capture relative position of different features in input image. Einstein-face-filter will have a high value at expected mustache position.

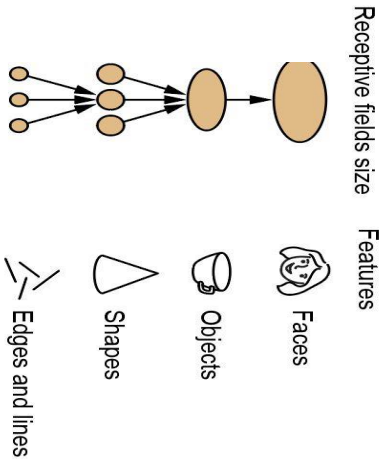
Weak analogies between brain and CNNs architecture



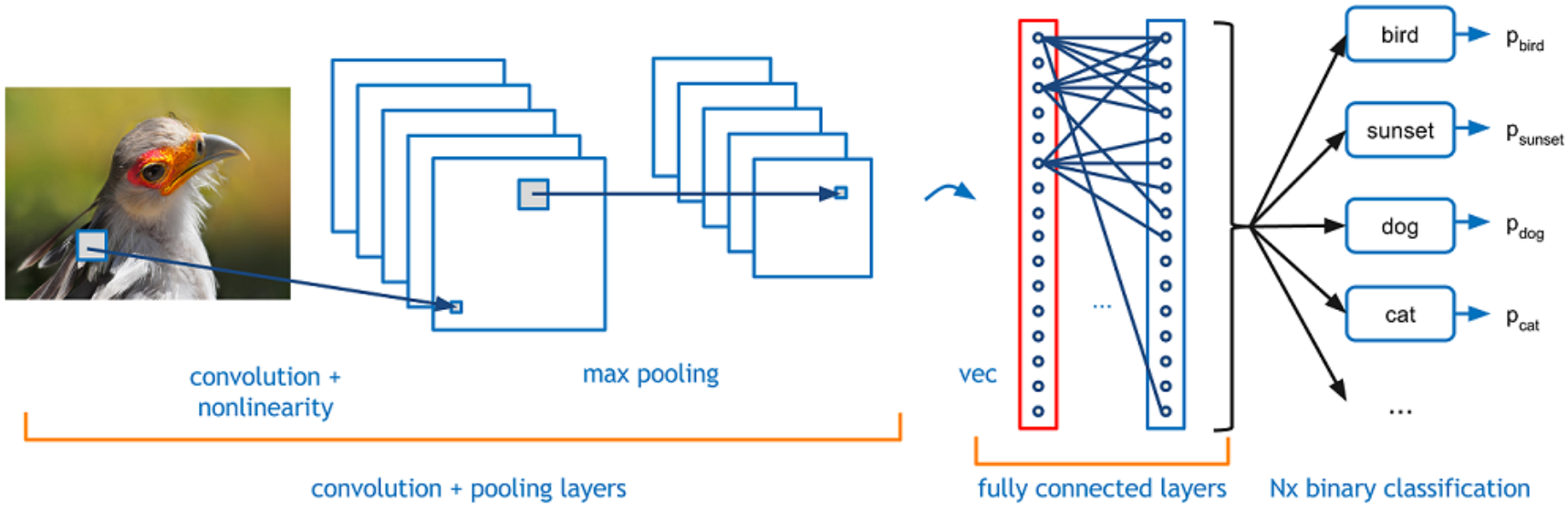








Convolutional part in CNN as representation learning



In a classical CNN we start with convolution layers and end with fc layers.

The task of the convolutional layers is to extract useful features from the image which might be appear at arbitrary positions in the image.

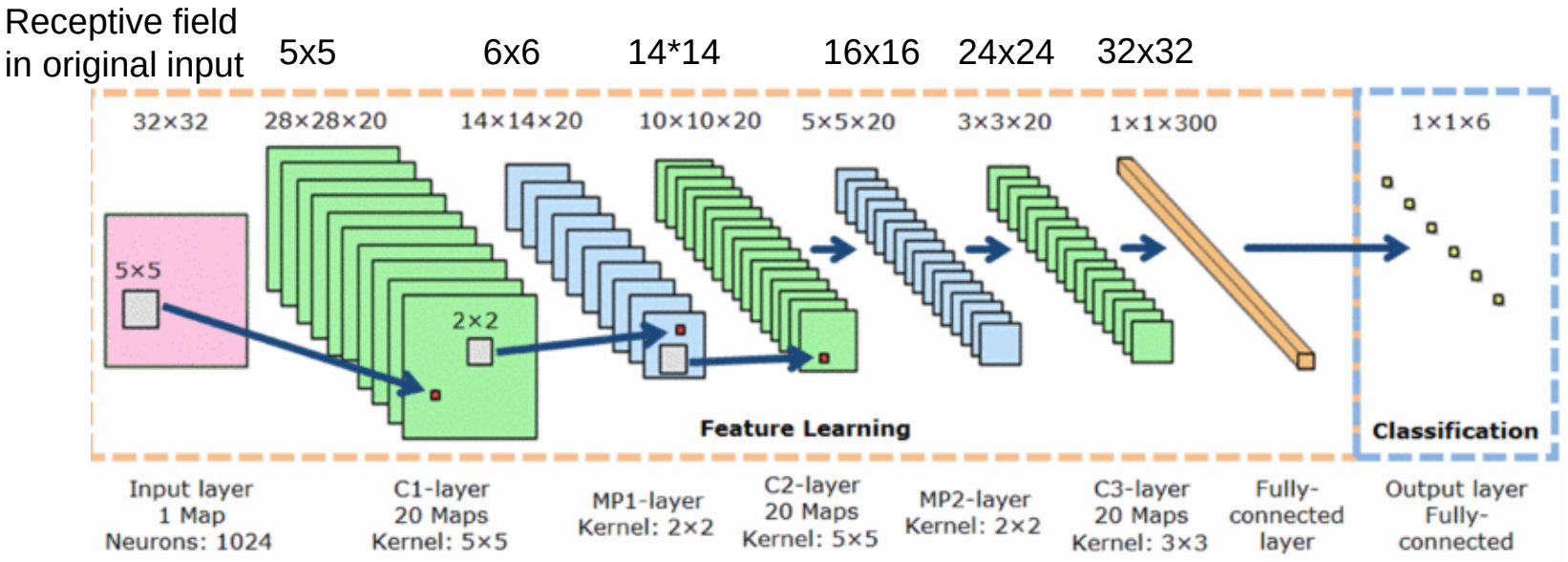
The task of the fc layer is to use these extracted features for classification.

Image credits:
https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

What input does activate a feature map in the CNN part or a neuron in the last layer?

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”.



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

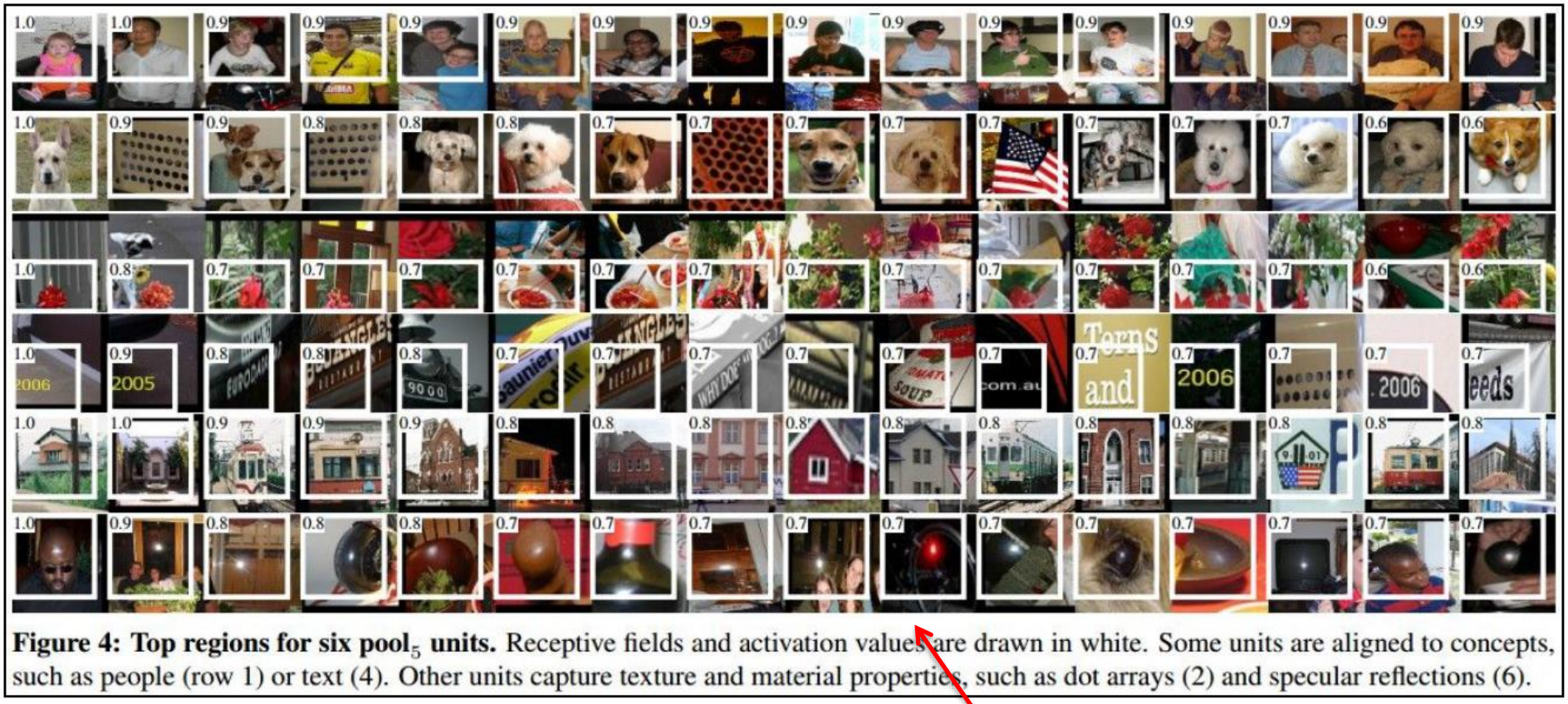
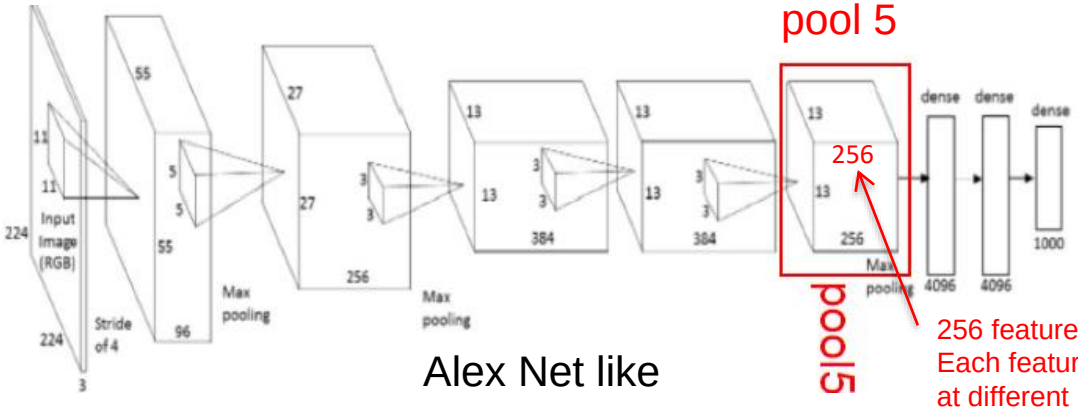


Figure 4: Top regions for six pool$_5$ units. Receptive fields and activation values are drawn in white. Some units are aligned to concepts, such as people (row 1) or text (4). Other units capture texture and material properties, such as dot arrays (2) and specular reflections (6).



Alex Net like

http://cs231n.github.io/understanding-cnn/

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**

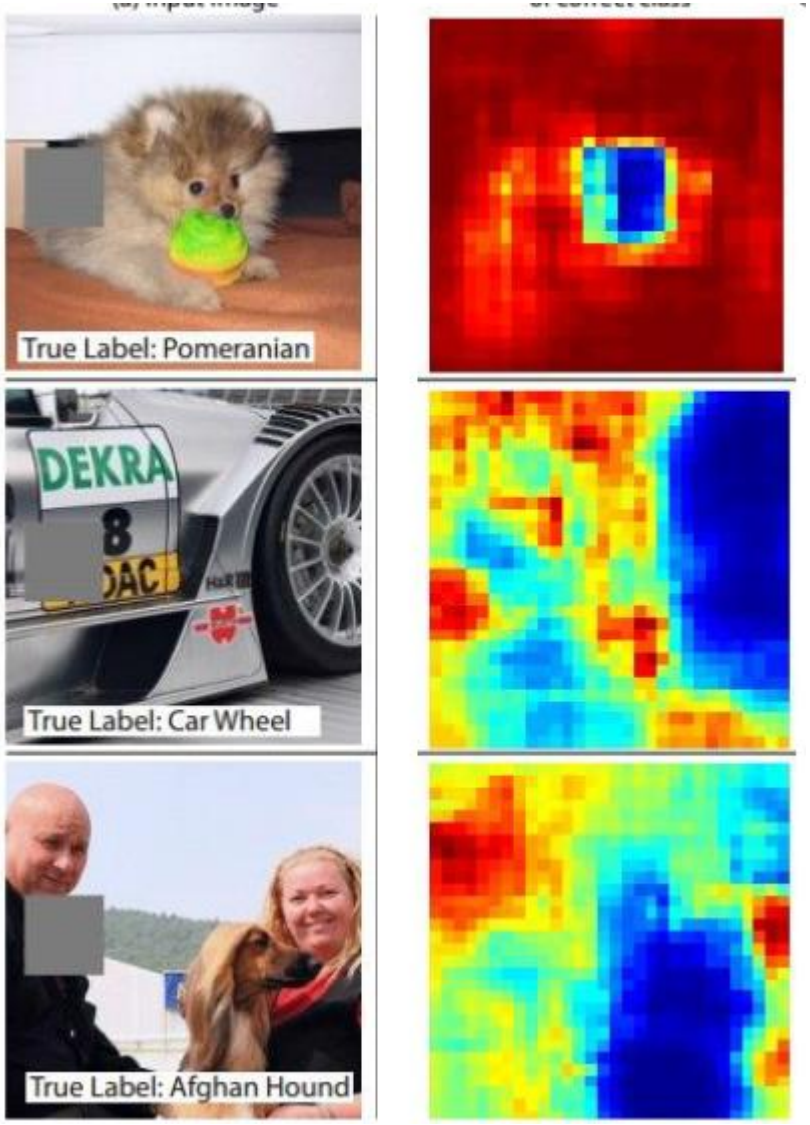
image credit: cs231n

What in the input image was important for the prediction?

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

Which pixels are important for the classification?
LIME: Local Interpretable Model-agnostic Explanations

Idea:

1) perturb interpretable features of the instance – e.g. randomly delete super-pixels in an image and track as perturbation vector such as (0,1,1,0,…,1)=x.
2) Classify perturbed instance by your model, here a CNN, and track the achieved classification-score=y
3) Identify for which features/super-pixels the presence in the perturbed input version are important to get a high classification score (use RF or lasso for y~x)

(a) Husky classified as wolf

(b) Explanation

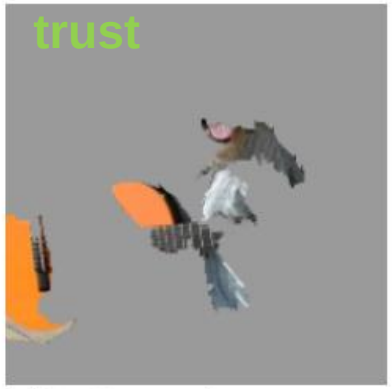
-> presence of snow was used to distinguish wolf and husky

-> Explain the CNN classification by showing instance-specific important features

visualize important feature allows to judge the individual classification

(a) Original Image

(b) Explaining *Electric guitar*



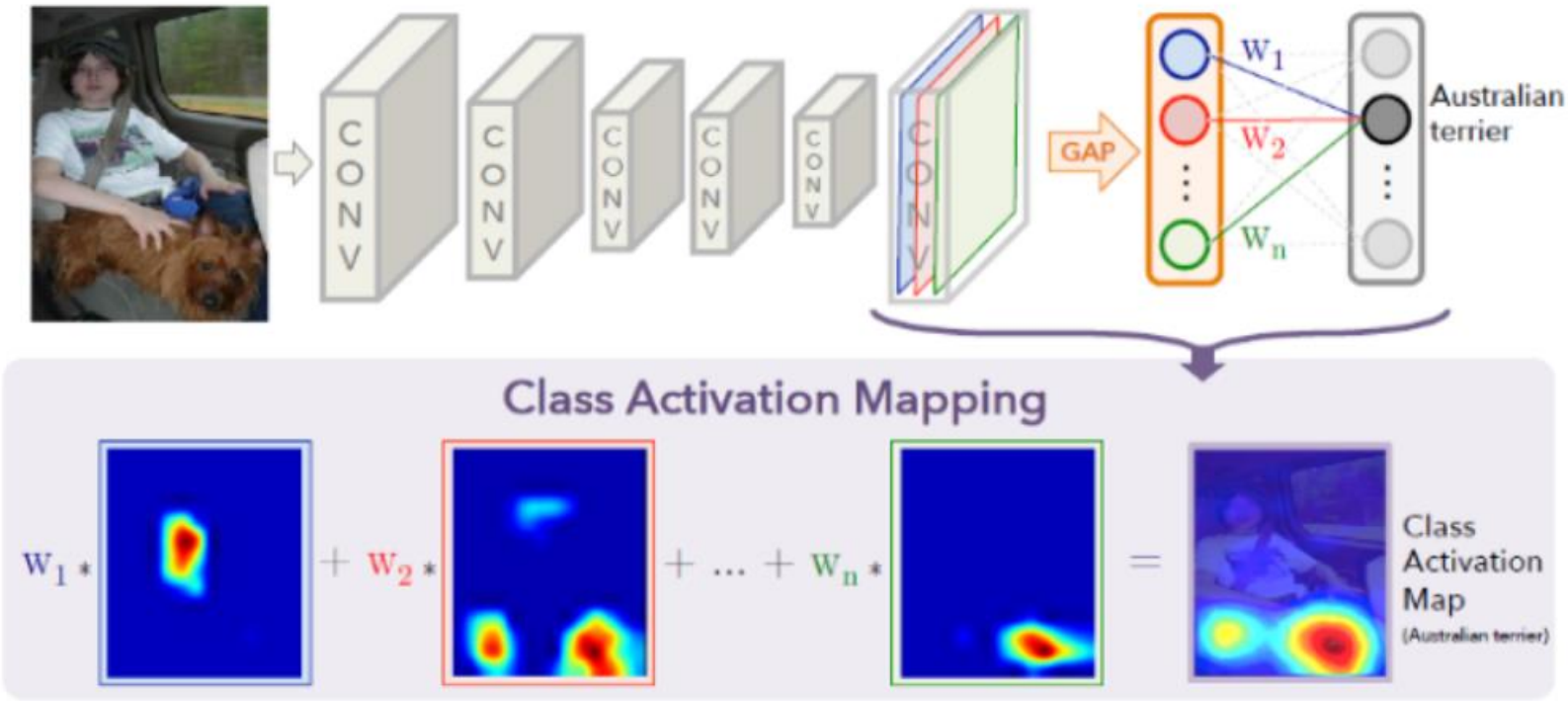
(c) Explaining *Acoustic guitar*

(d) Explaining *Labrador*

DOI: http://dx.doi.org/10.1145/2939672.2939778

Tf, python code: https://github.com/marcotcr/lime -> image tutorial

Class Activation Mapping (CAM)



We compute the CAM by multiplying each feature map of the final convolution layer with the corresponding weight connected to the neuron of the winning class.

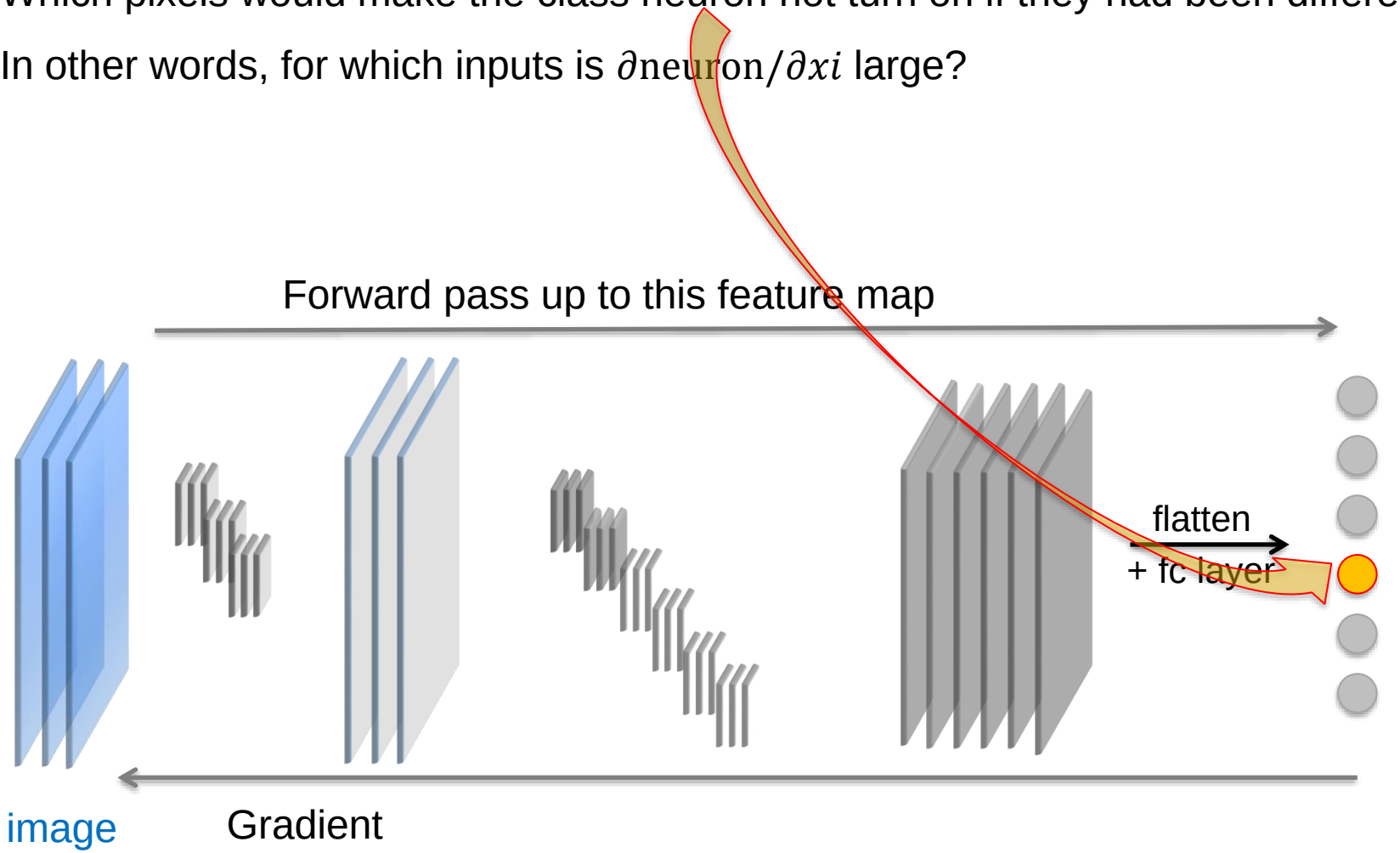
https://arxiv.org/abs/1512.04150

https://towardsdatascience.com/class-activation-mapping-using-transfer-learning-of-resnet50-e8ca7cfd657e

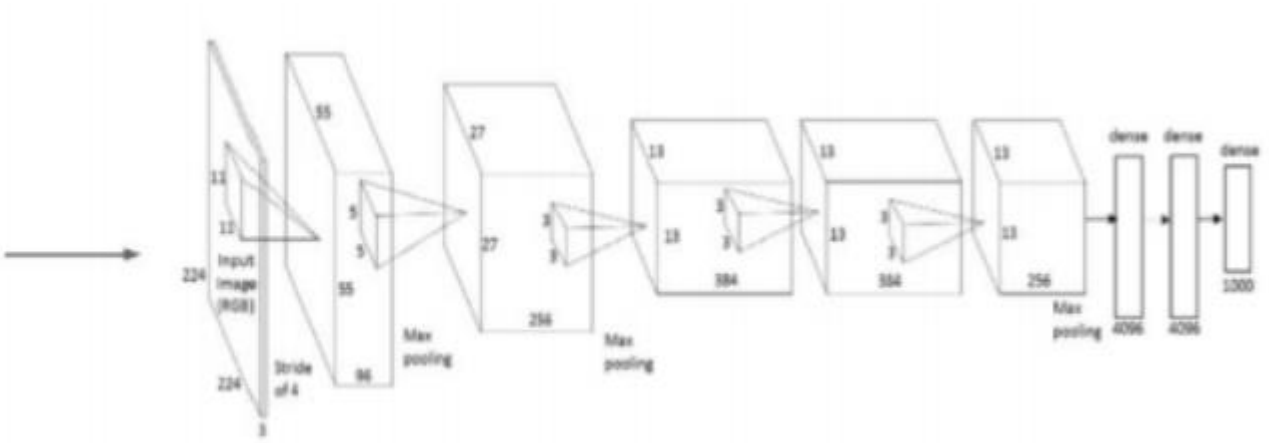
Gradient Backpropagation

Which pixels would make the class neuron not turn on if they had been different?

In other words, for which inputs is $\partial \text{neuron} / \partial xi$ large?

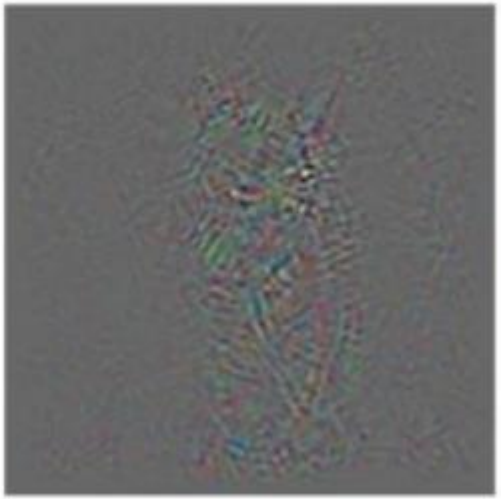


Gradient Backpropagation



1) Do a forward pass with the image

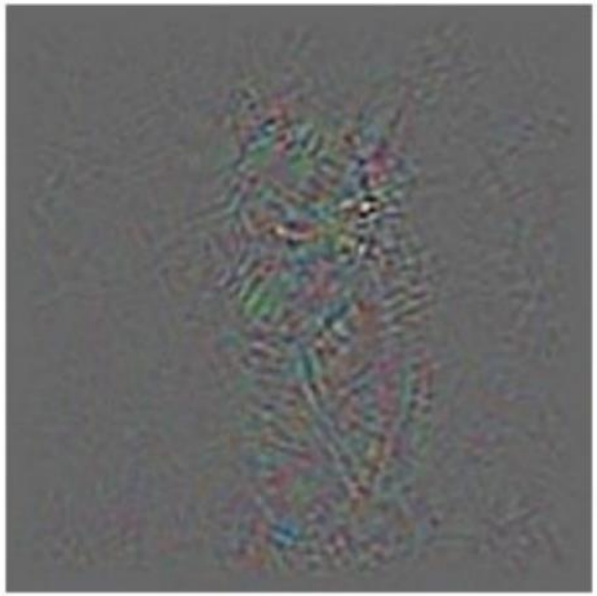
2) Compute the gradient of the winning class neuron using backprop



https://www.cs.toronto.edu/~guerzhoy/321/lec/W07/HowConvNetsSee.pdf

Guided Gradient Backpropagation

- We are only interested to see which image features the neuron detects, not in what kind of stuff it doesn't detect
- So only propagating positive gradients (set all the negative gradients to 0)

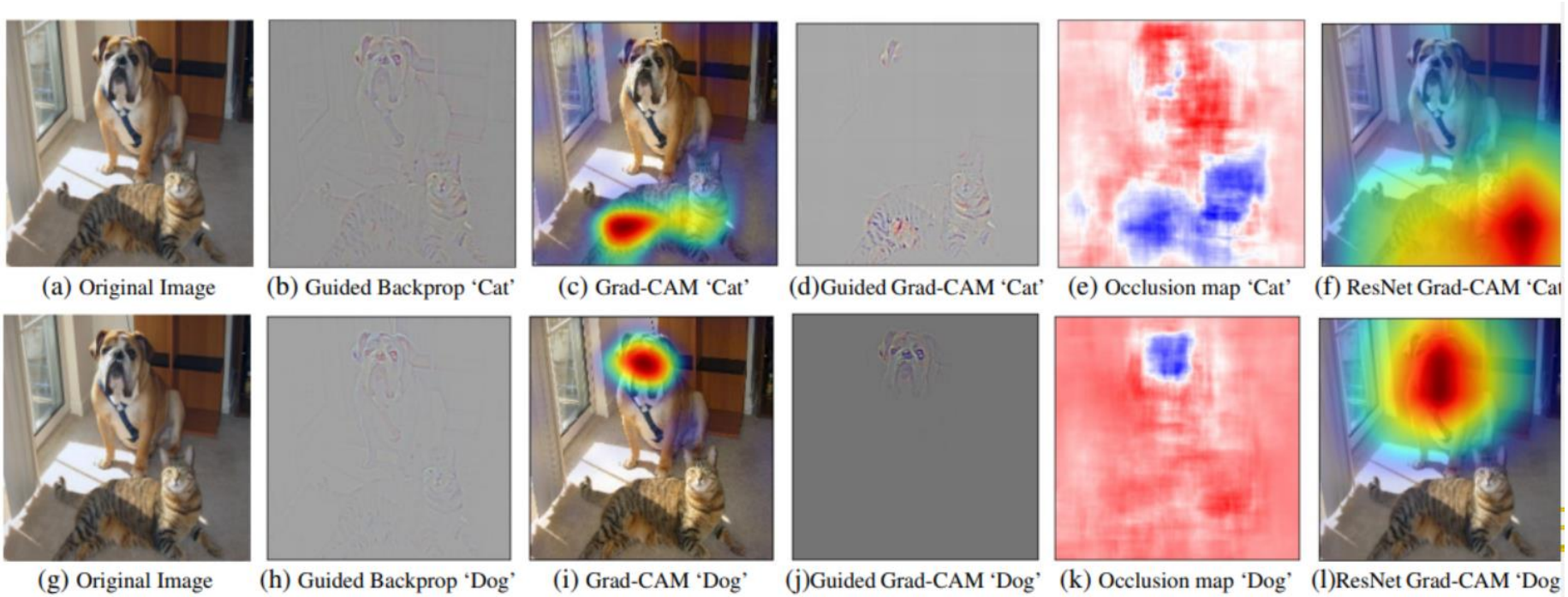


Backprop

Guided Backprop

https://www.cs.toronto.edu/~guerzhoy/321/lec/W07/HowConvNetsSee.pdf

What in the image was important for the prediction?



Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization https://arxiv.org/abs/1610.02391

Challenge winning CNN architectures

Review of ImageNet winning CNN architectures

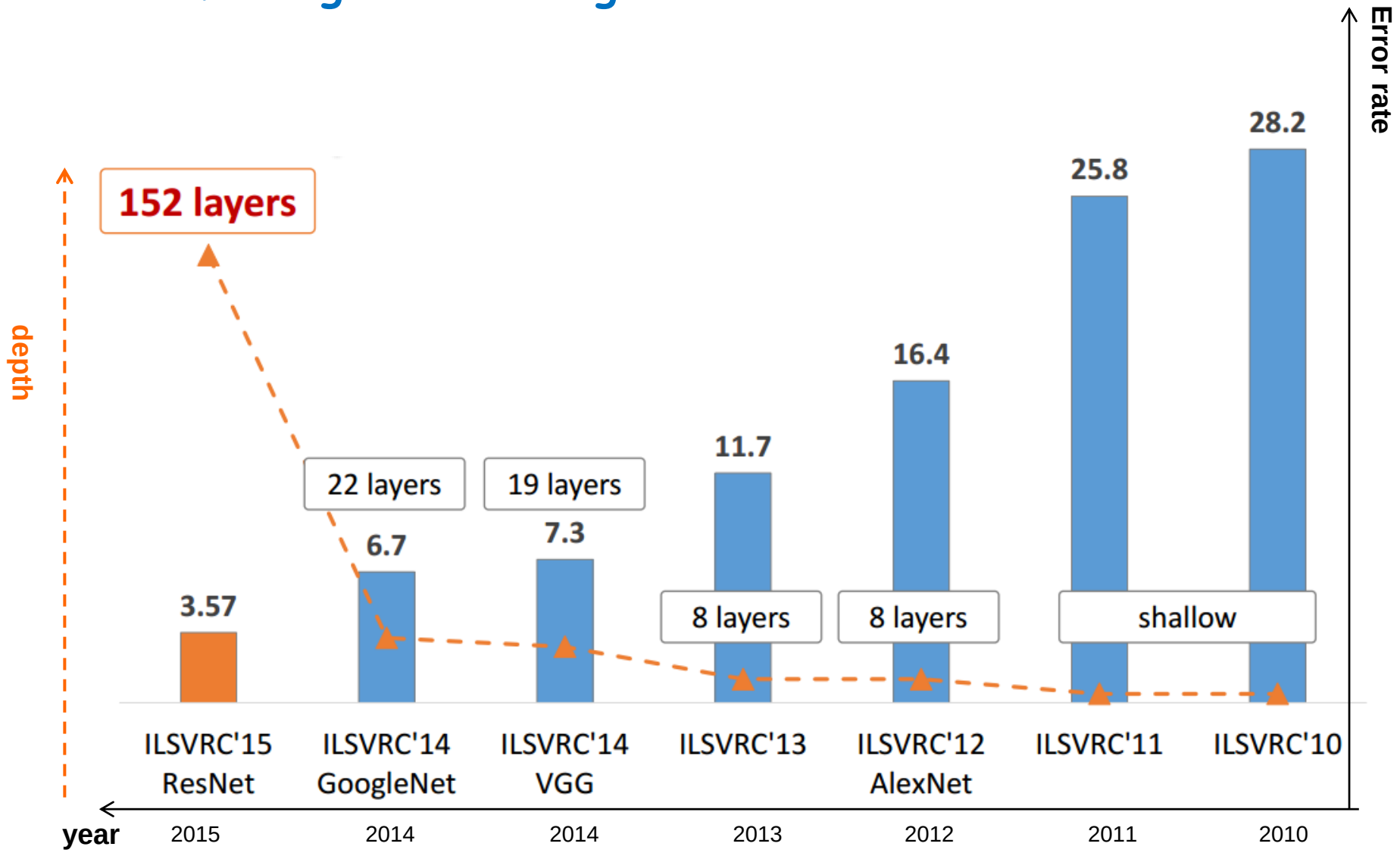


Image credits (modified): K.He, X.Zhang, S.Ren, J.Sun. “Deep Residual Learning for Image Recognition”. arXiv 2015

LeNet-5 1998: first CNN for ZIP code recognition

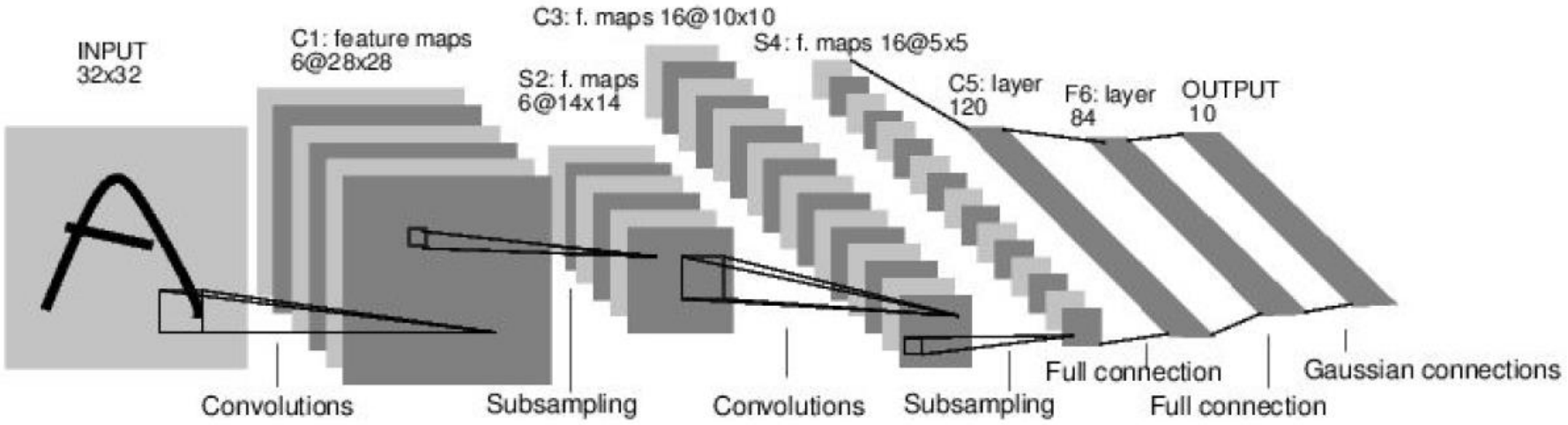


Image credits: http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

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]

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

The trend in modern CNN architectures goes to small filters

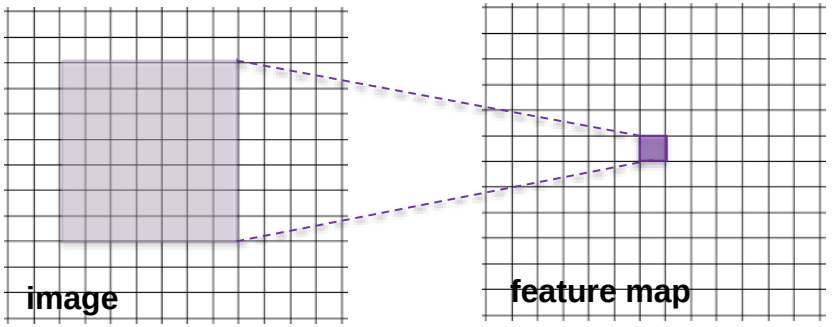
Why do modern architectures use very small filters?

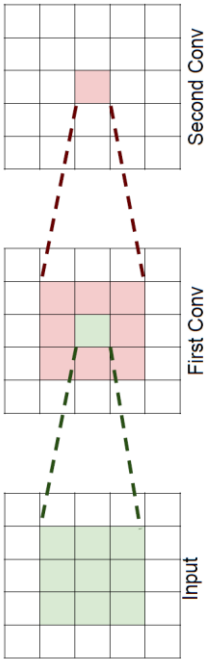
Determine the receptive field in the following situation:

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

49 weights

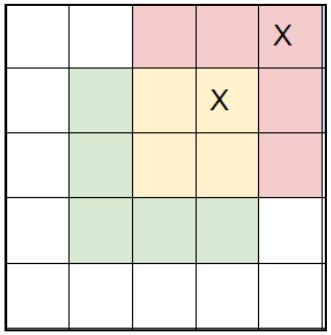
Answer1): 7x7





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

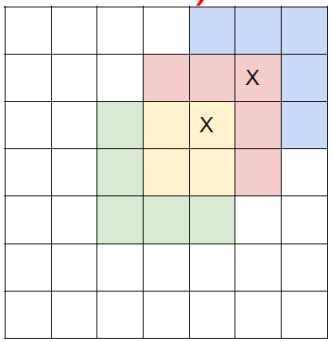
Answer 2): 5x5



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

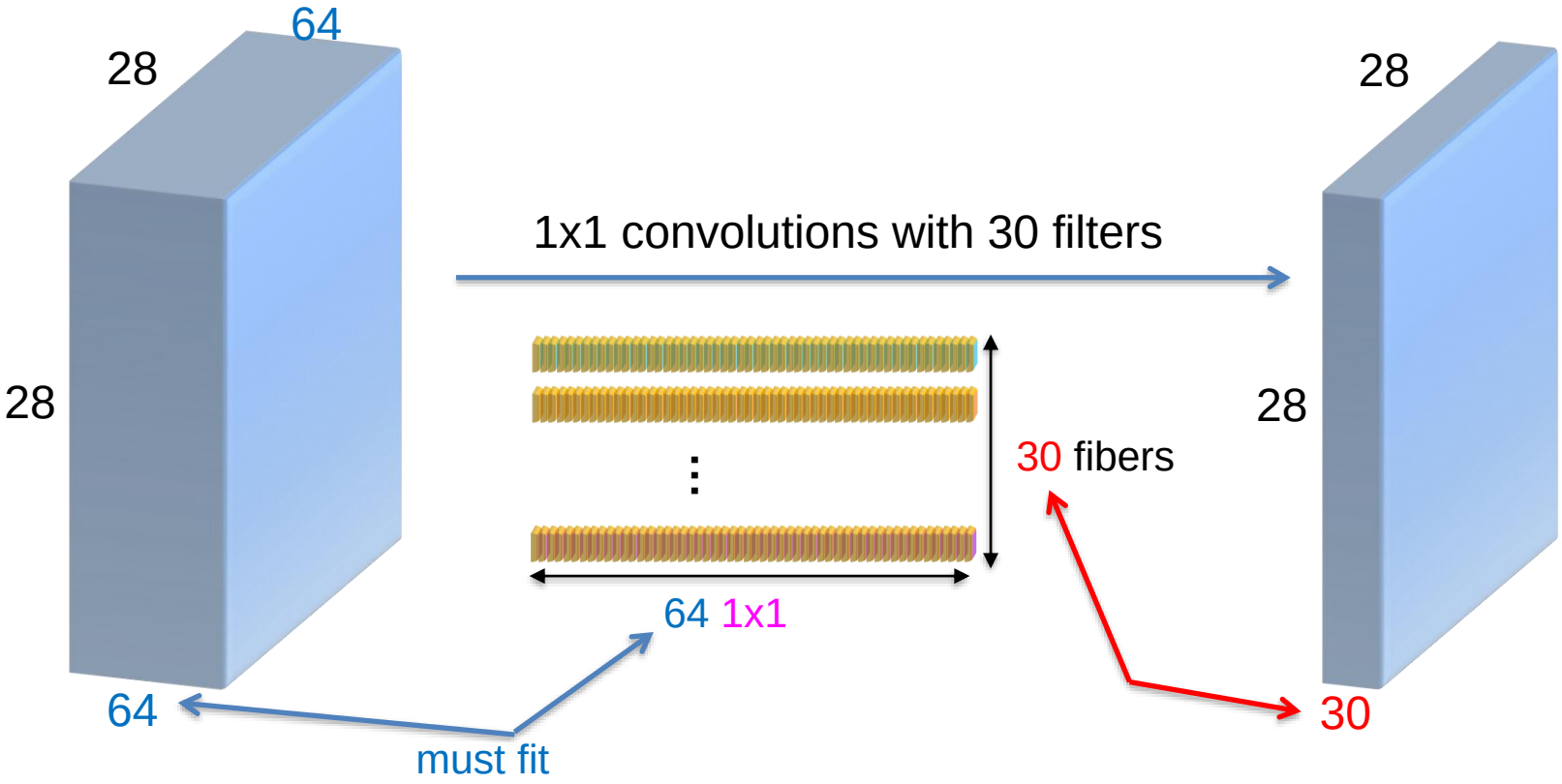
3*9=27 weights

Answer 3): 7x7



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

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

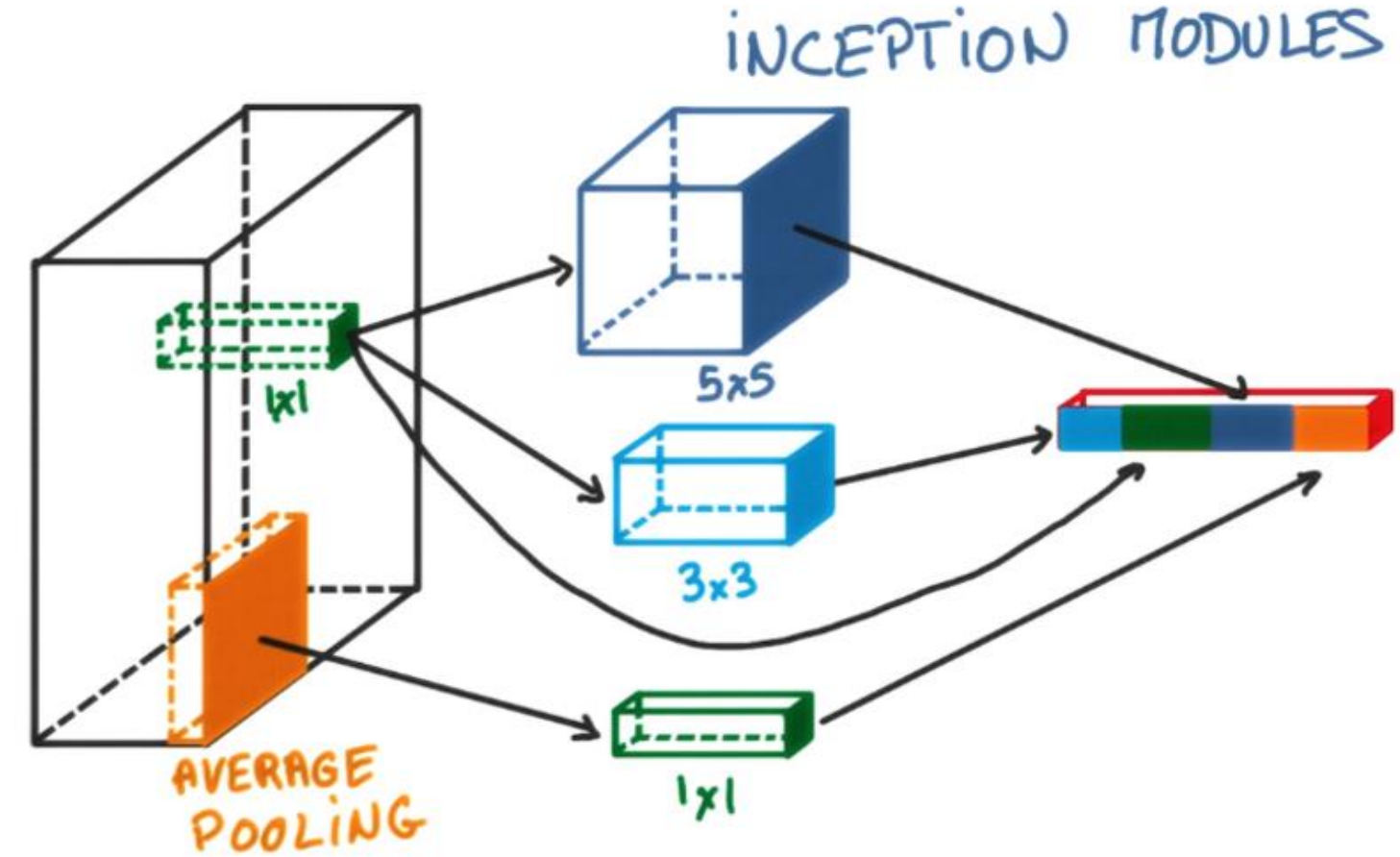


1x1 convolution act along a “fiber” in depth dimension across the channels.

→ efficient way to reduce/change the depth dimension

→ simultaneously introduce more non-linearity

The idea of inception modules



Between two layers just do several operations in parallel: pooling and 1x1 conv, and 3x3 and 5x5. “same”-conv and concatenate them together. Benefit: total number of parameters is small, yet performance better.

Slide (modified) credit: Google MOOC on tensorflow

How to concatenate tensors of different dimensions?

DepthConcat needs to make the tensor the same in all, but the depth dimension:

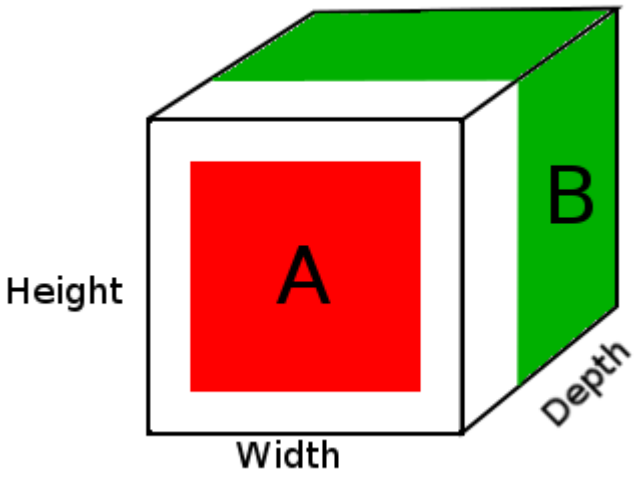
To deal with different output dimensions, the largest spatial dimension is select and zero-padding around the smaller dimensions is added.

Usually depth gets large!
Do 1x1 conv afterwards!

Pseudo code for an example:

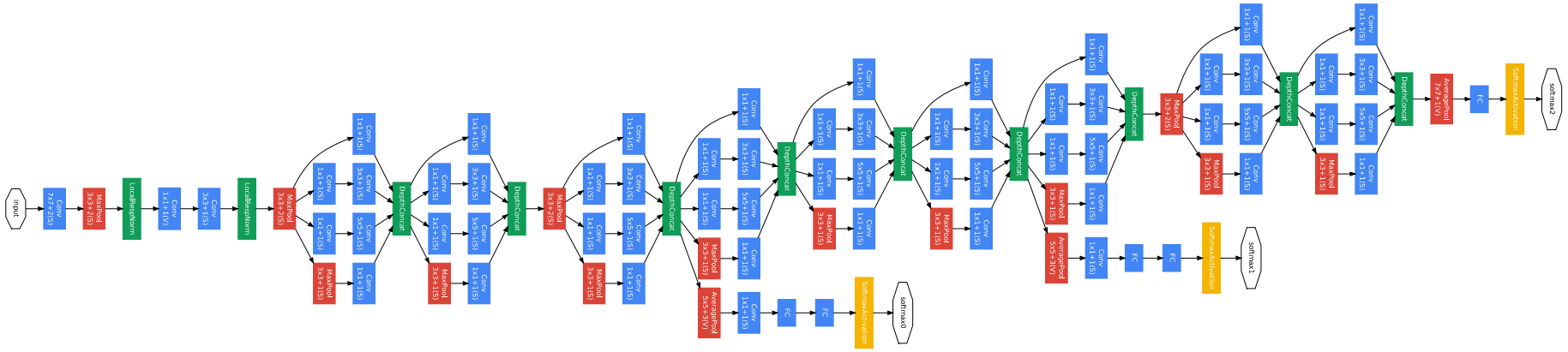
```
A = tensor of size (14, 14, 2)
B = tensor of size (16, 16, 3)
result = DepthConcat([A, B])
```

where result with have a
height of 16,
width of 16 and a
depth of 5 (2 + 3)

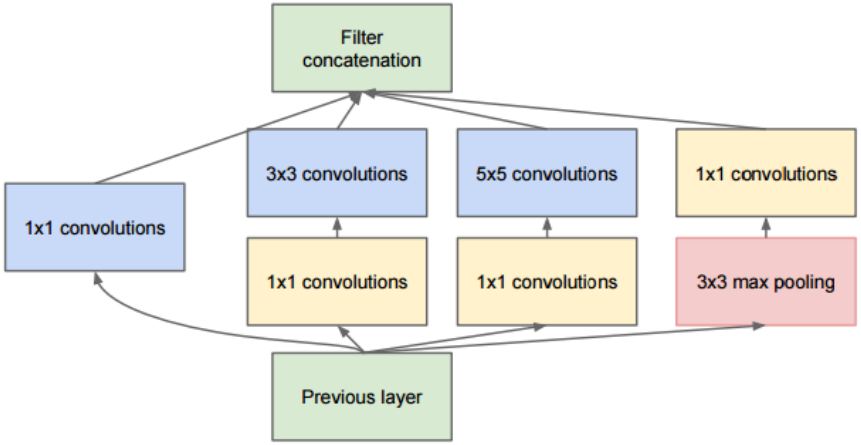


See: http://stats.stackexchange.com/questions/184823/how-does-the-depthconcat-operation-in-going-deeper-with-convolutions-work

Winning architecture (GoogLeNet, 2014)



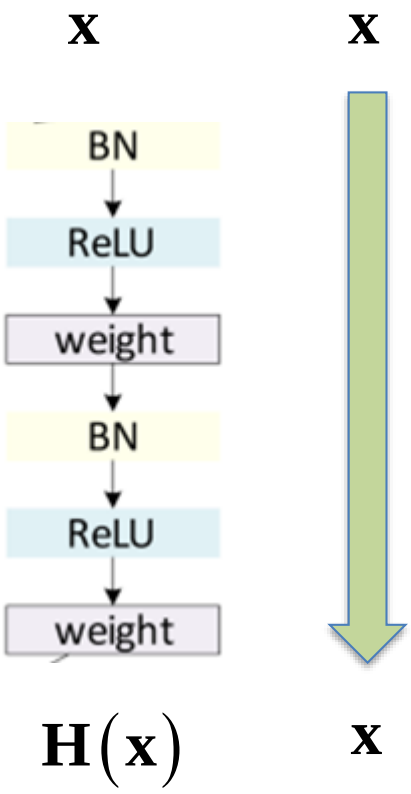
The inception module (convolutions and maxpooling)



Few parameters, hard to train.
Comments see here

Going deeper with convolution http://arxiv.org/abs/1409.4842

Highway Networks: providing a highway for the gradient



Idea: Use nonlinear transform T to determine how much of the output **y** is produced by H or the identity mapping. Technically we do that by:

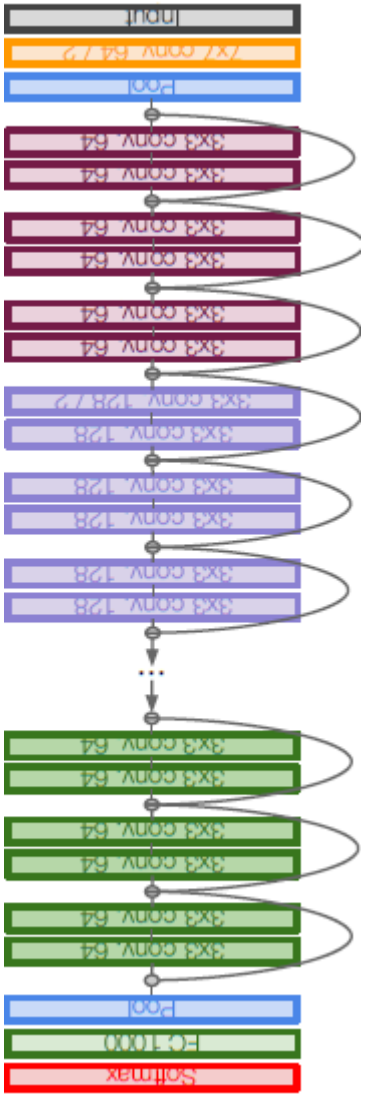
$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot (1 - T(\mathbf{x}, \mathbf{W_T}))$$

Special case:

$$\mathbf{y} = \begin{cases} \mathbf{x}, & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{0} \\ H(\mathbf{x}, \mathbf{W_H}), & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{1} \end{cases}$$

This opens a highway for the gradient:

$$\frac{d\mathbf{y}}{d\mathbf{x}} = \begin{cases} \mathbf{I}, & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{0}, \\ H'(\mathbf{x}, \mathbf{W_H}), & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{1}. \end{cases}$$



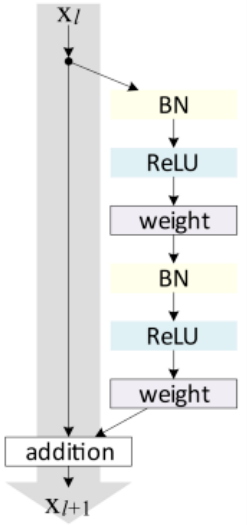
Srivastava, Greff, Schmidhuber 2014, “Highway Networks” https://arxiv.org/pdf/1505.00387.pdf

"ResNet" from Microsoft 2015 winner of imageNet

152 layers

ResNet basic design (VGG-style)

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



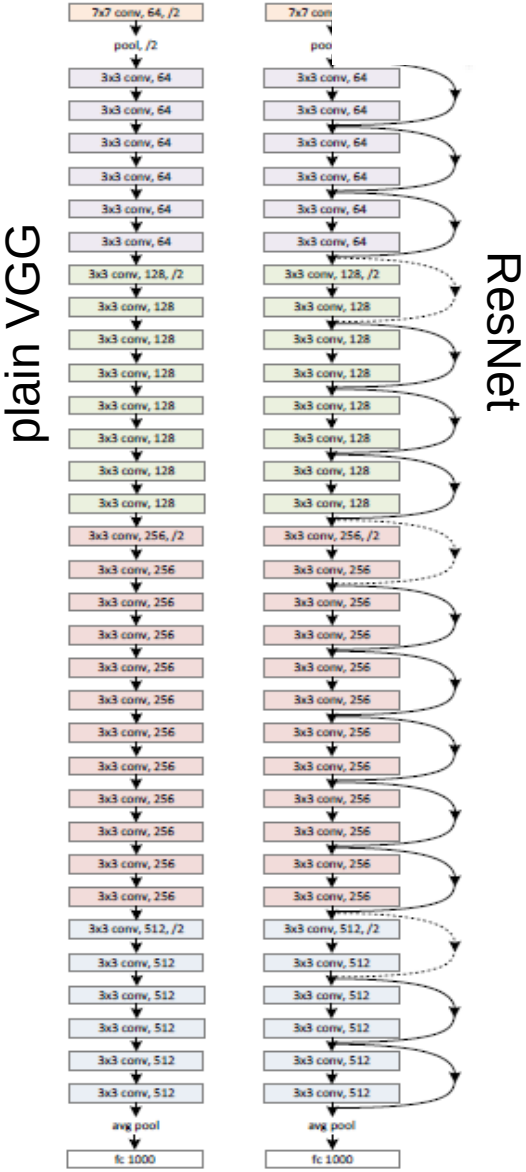
$F(x_l)$

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

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

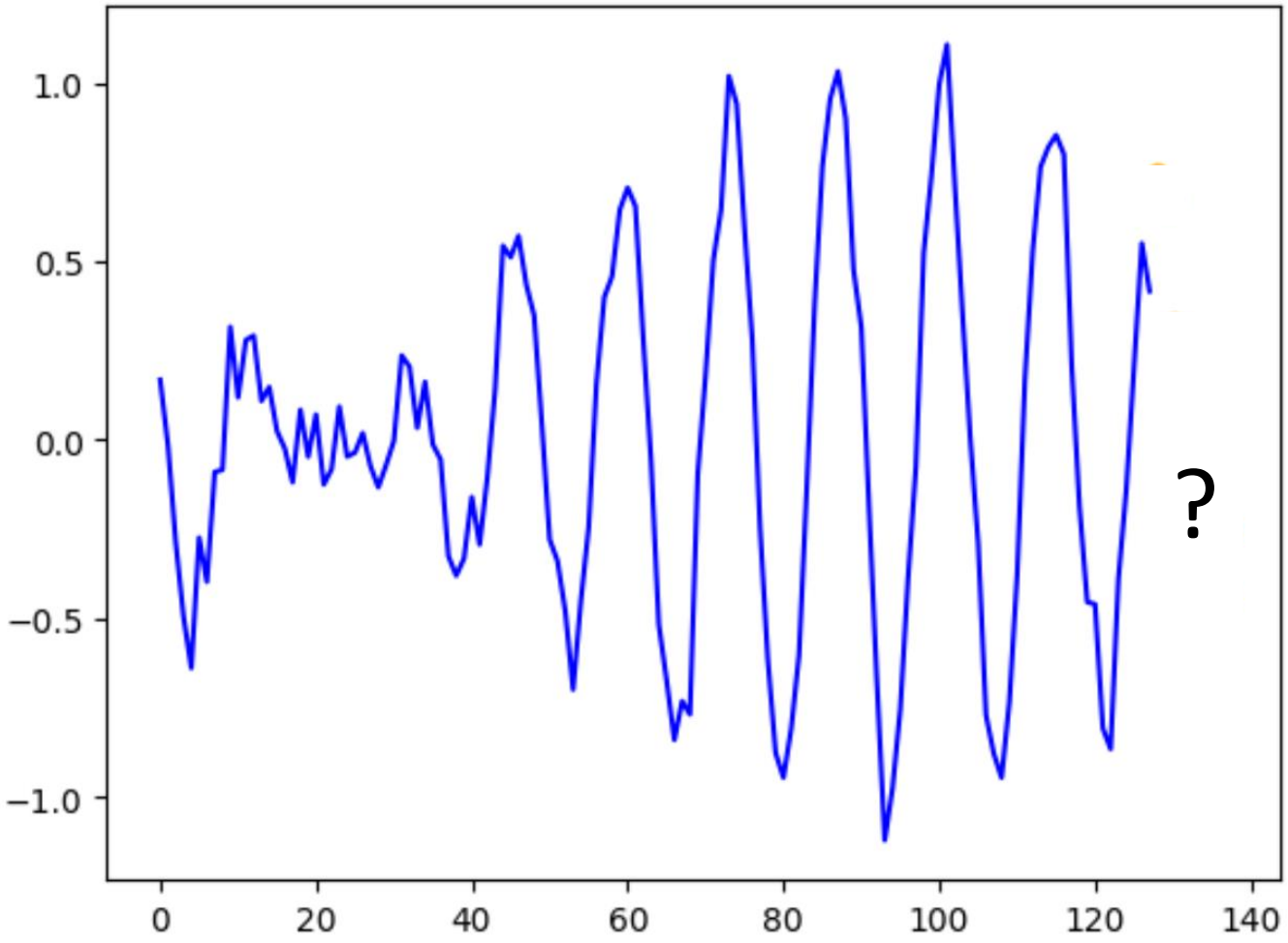
- 2nd place in the imageNet challenge
- More traditional, easier to train
- More weights than GoogLeNet
- Small pooling
- Stacked 3x3 convolutions before maxpooling

  -> 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

1D CNNs for sequence data

How to make predictions based on a given time series?

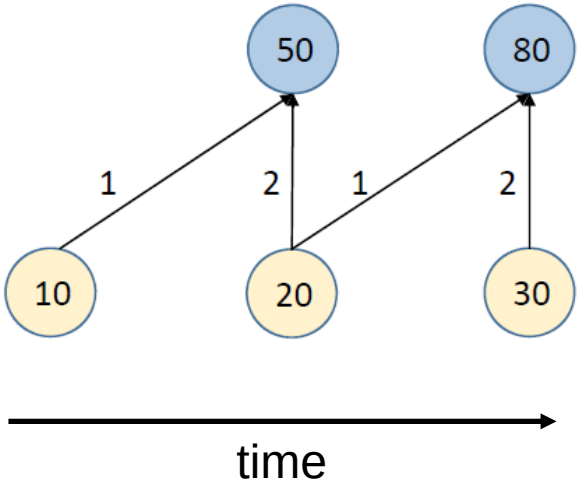


1D "causal" convolution for time-ordered data

Toy example:

Input X: 10,20,30

1D kernel of size 2: 1,2



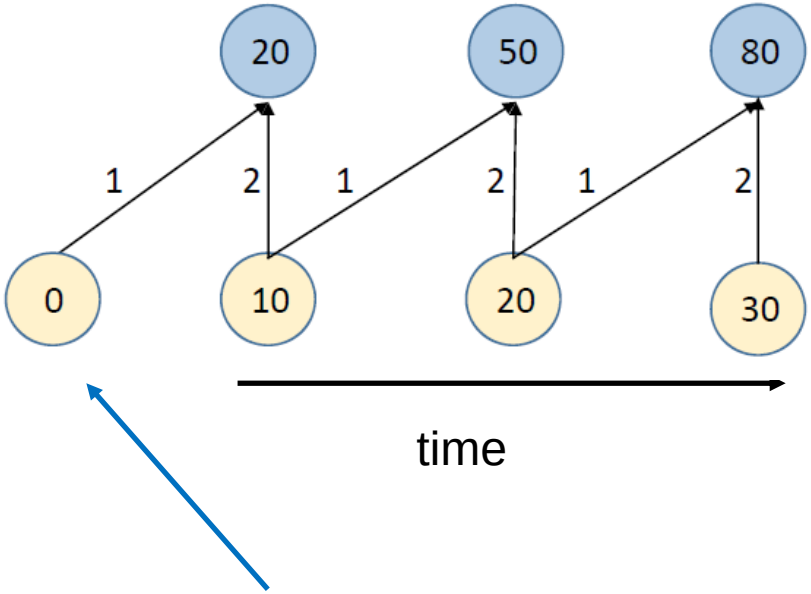
It's called "causal" networks, because the architecture ensured that only information from the past has an influence on the present and future.

Zero-padding in 1D "causal" CNNs

Toy example:

Input X: 10,20,30

1D kernel of size 2: 1,2

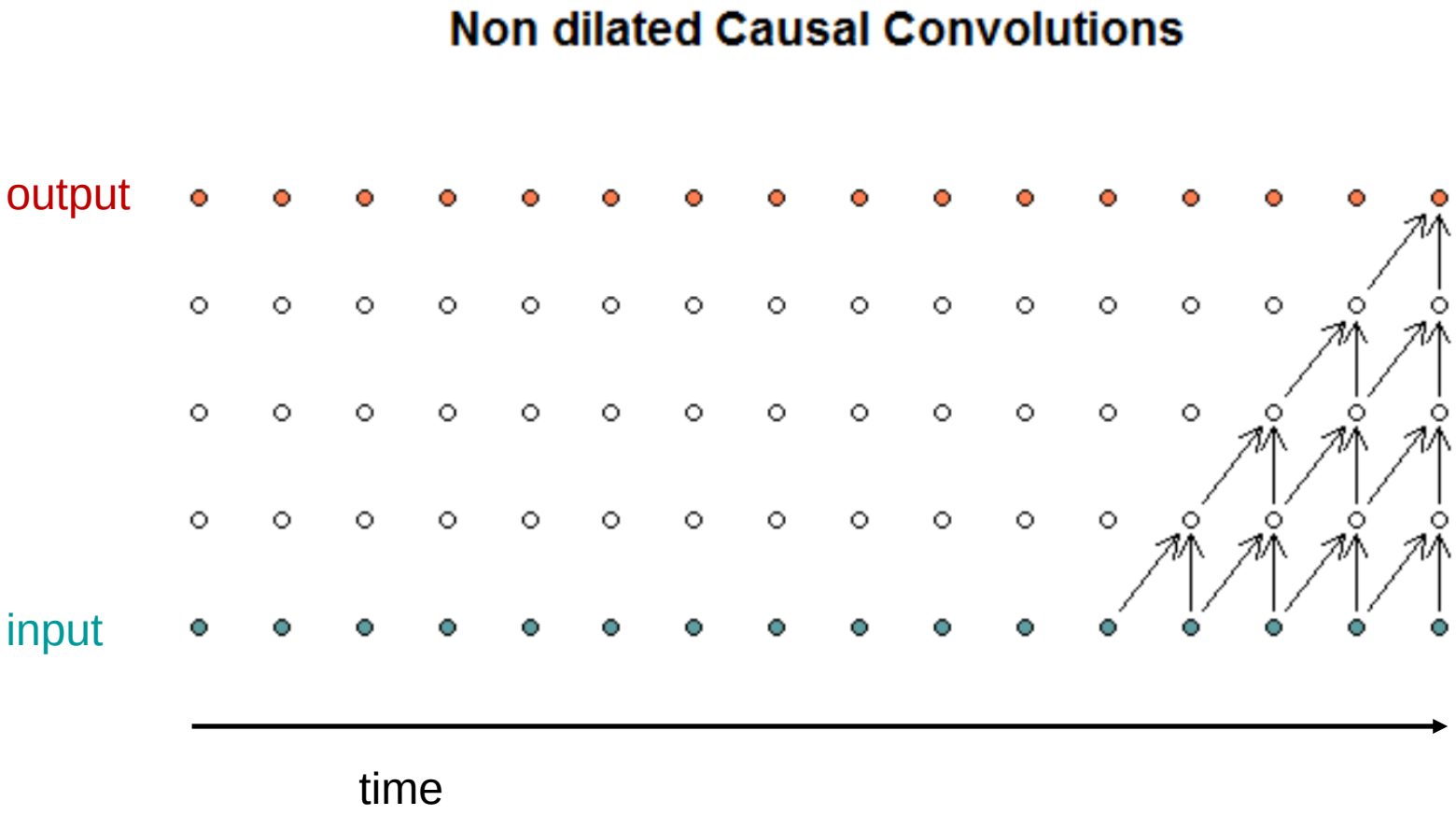


To make all layers the same size, a zero padding is added to the beginning of the input layers

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



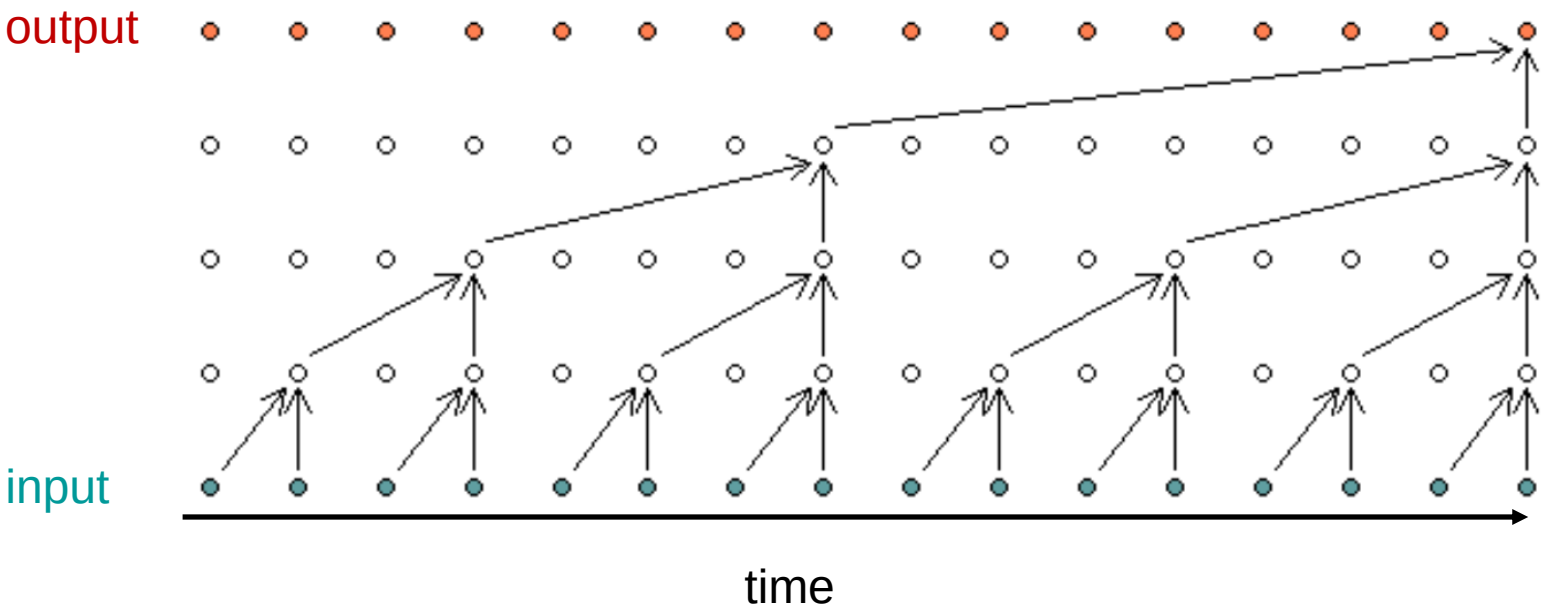
Stacking k causal 1D convolutions with kernel size 2 allows to look back k time-steps.

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.

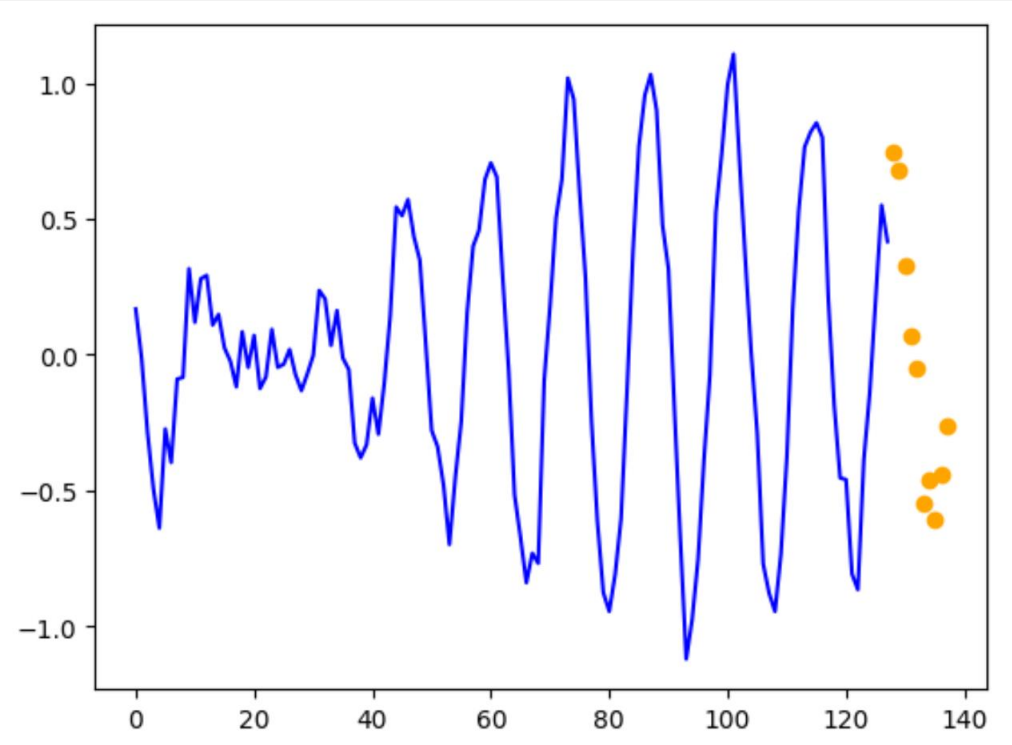
https://deepmind.com/blog/wavenet-generative-model-raw-audio/

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

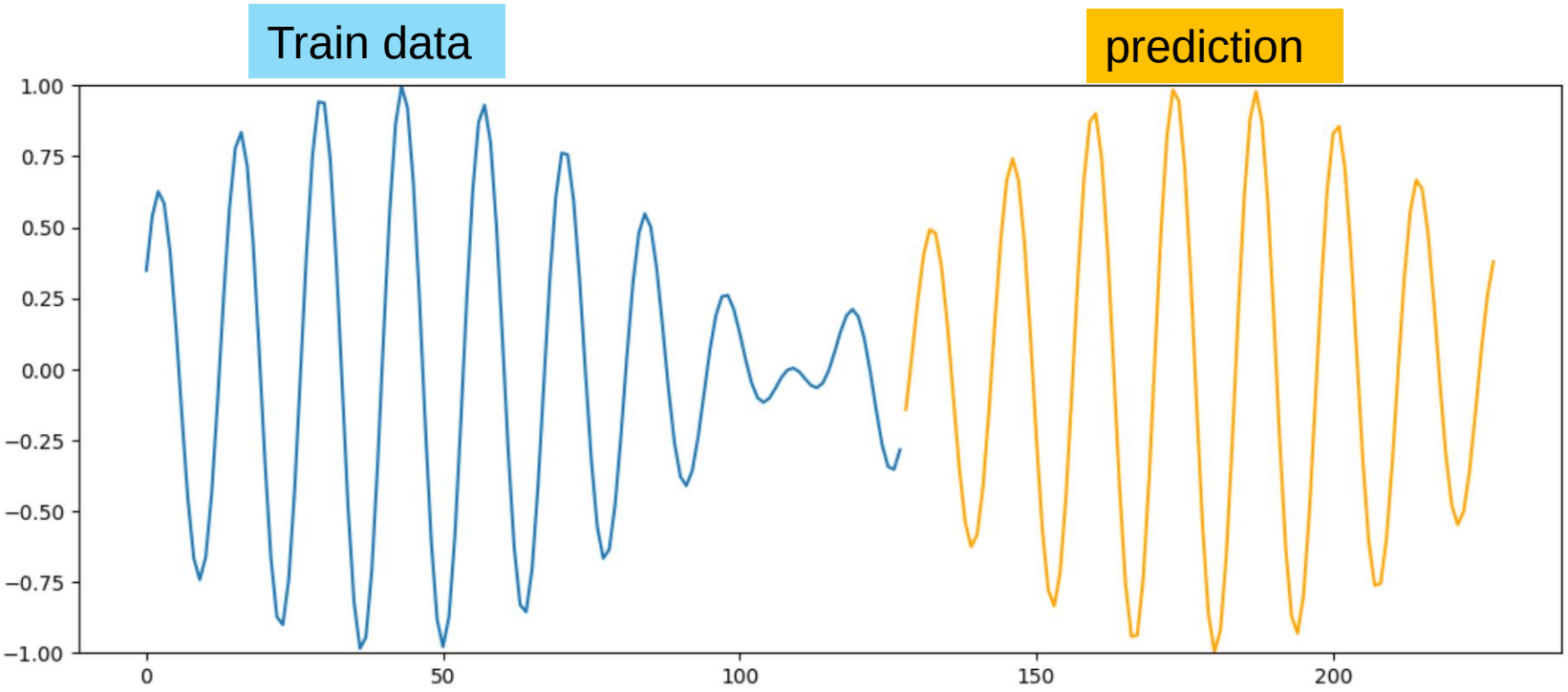


Work through the notebook (optional)
https://github.com/tensorchiefs/dl_course_2021/blob/master/notebooks/09_1DConv.ipynb.

58

Dilated 1D causal CNNs help if long memory is needed

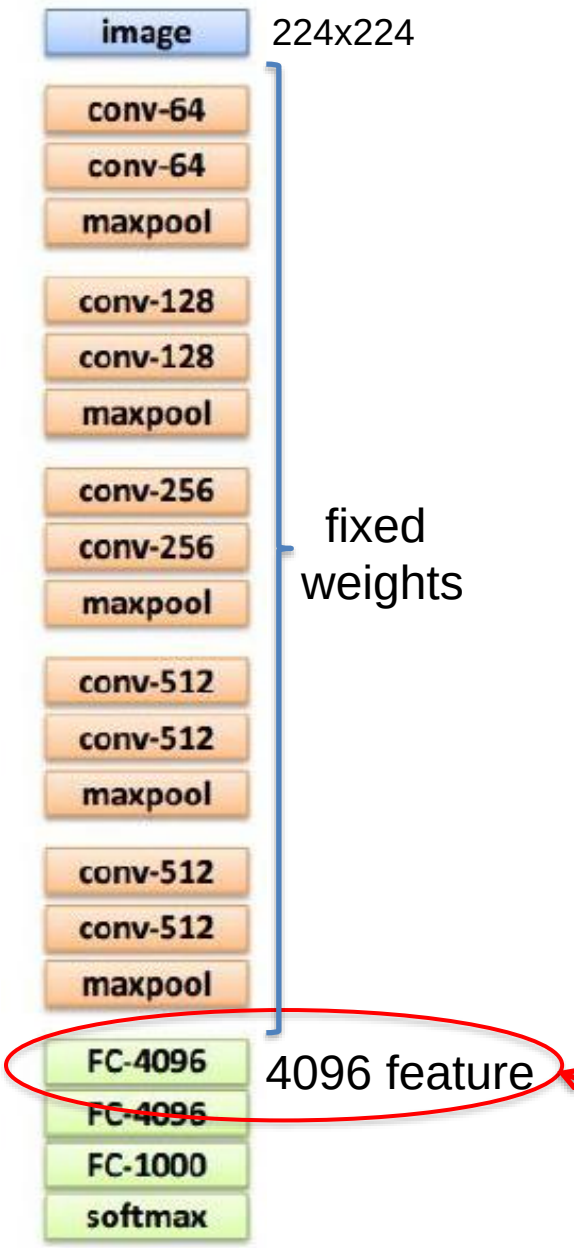
Dilated 1D CNNs can picked up the long-range time dependencies.



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_2021/blob/master/notebooks/09_1DConv_sol.ipynb.

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)

Are VGG features useful?

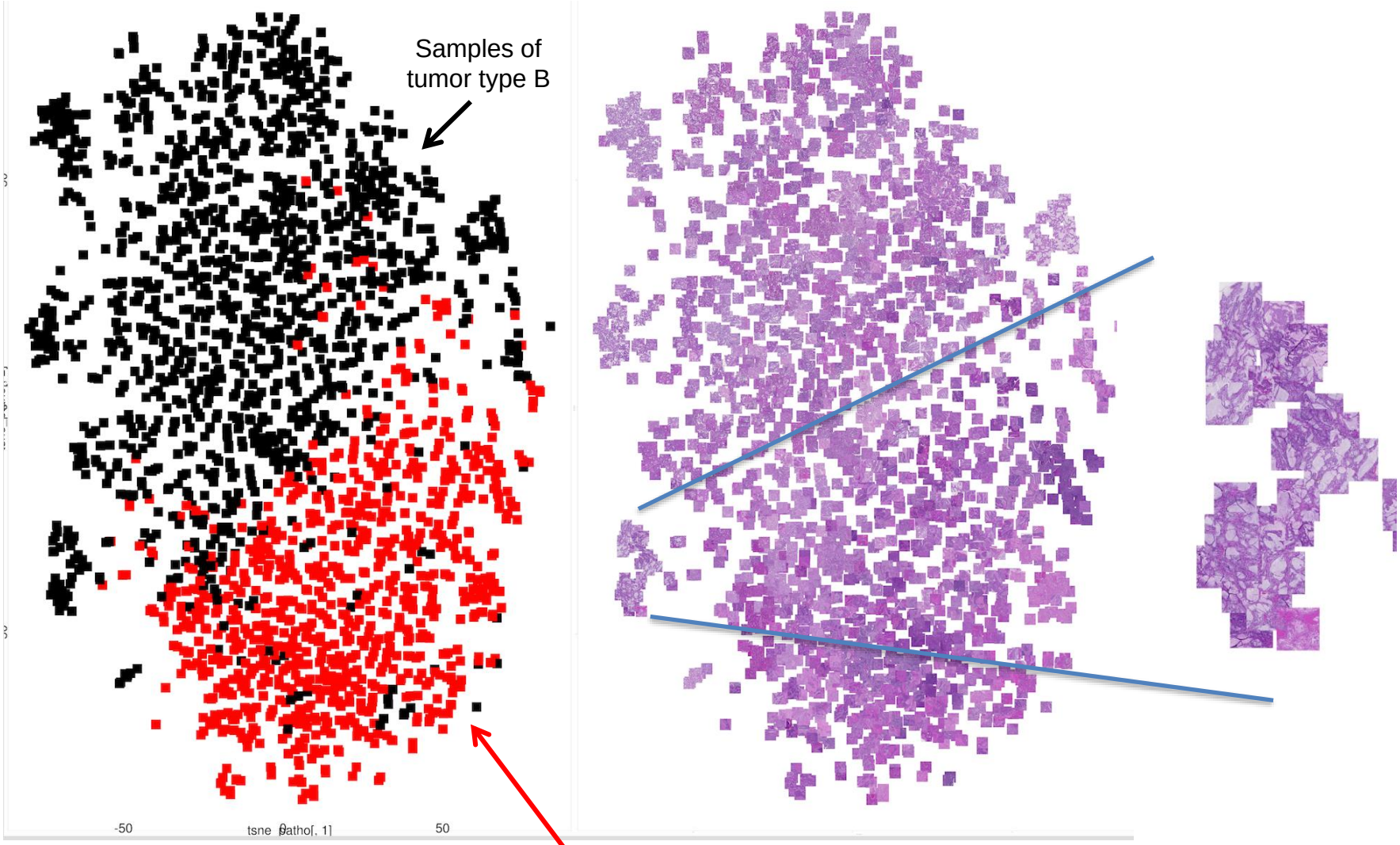
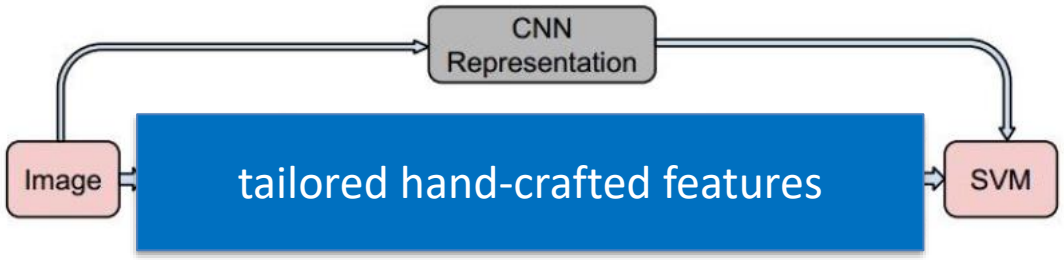
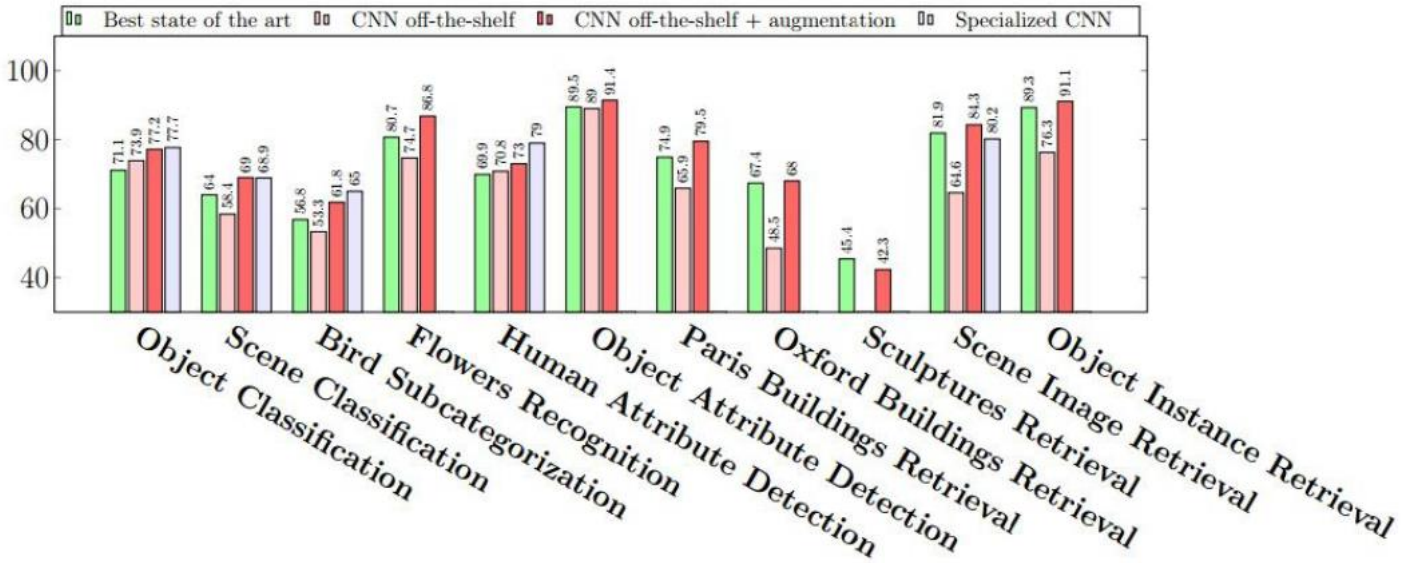


image credit: Elvis Murina

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



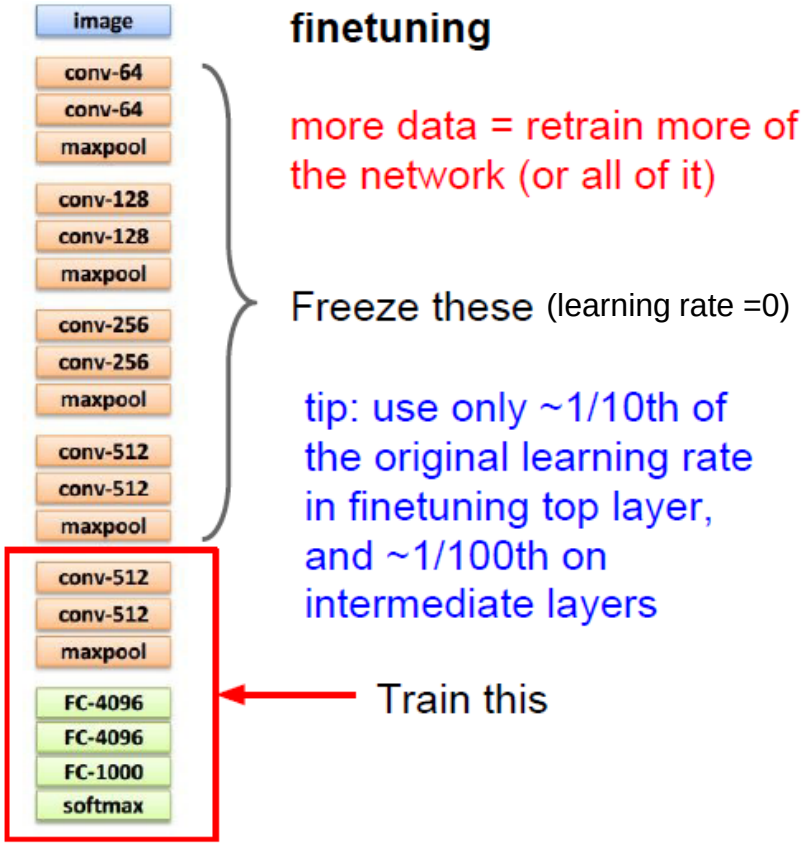


"Astonishingly, we report consistent superior results compared to the highly tuned state-of-the-art systems in all the visual classification tasks on various datasets."

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:

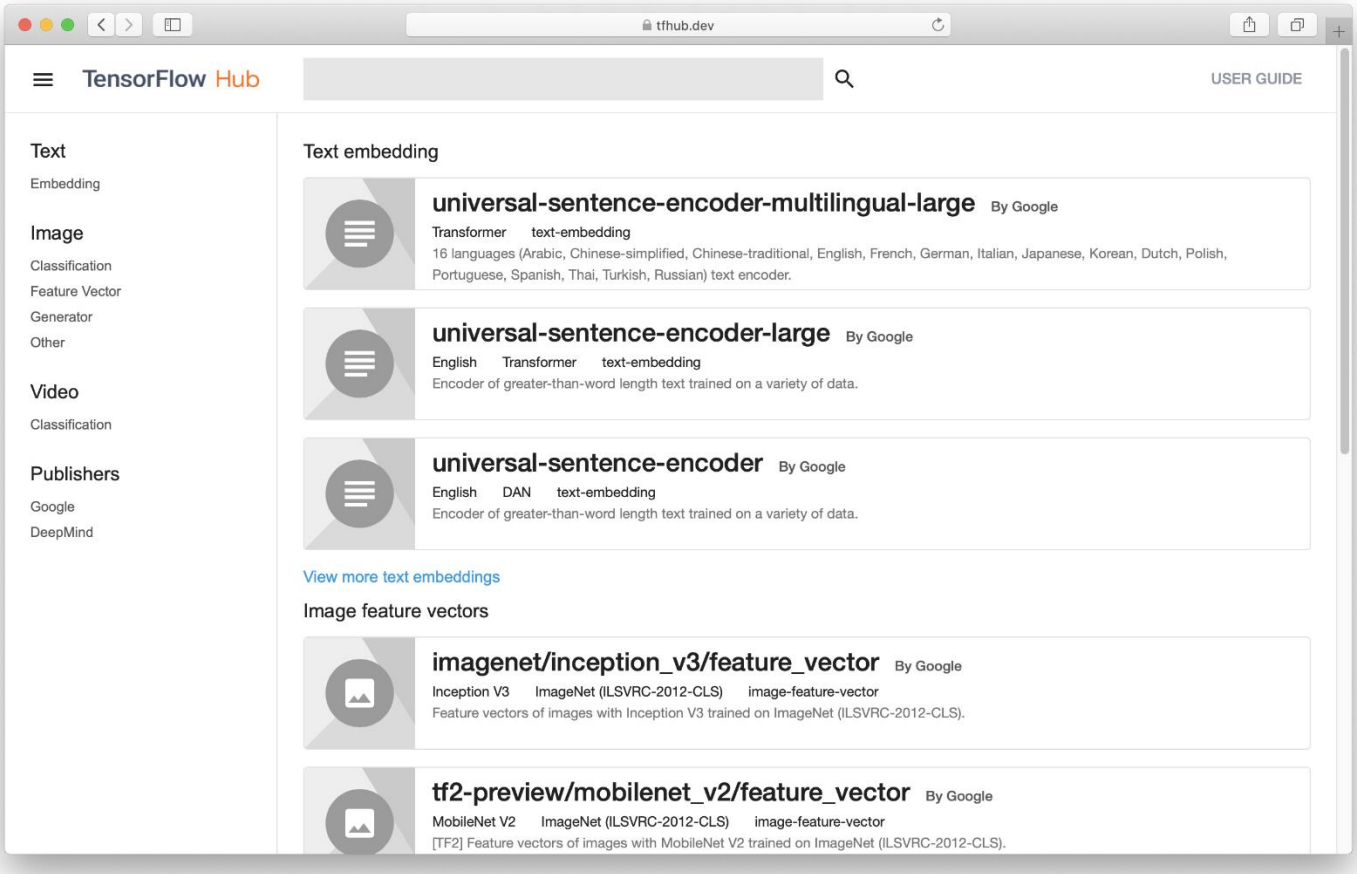
	Similar task (to imageNet challenge)	**Very different task** (to imageNet challenge)
little data	Extract CNN representation of one top fc layer and use these features to train an external classifier	You are in trouble - try to extract CNN representations from different stages and use them as input to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

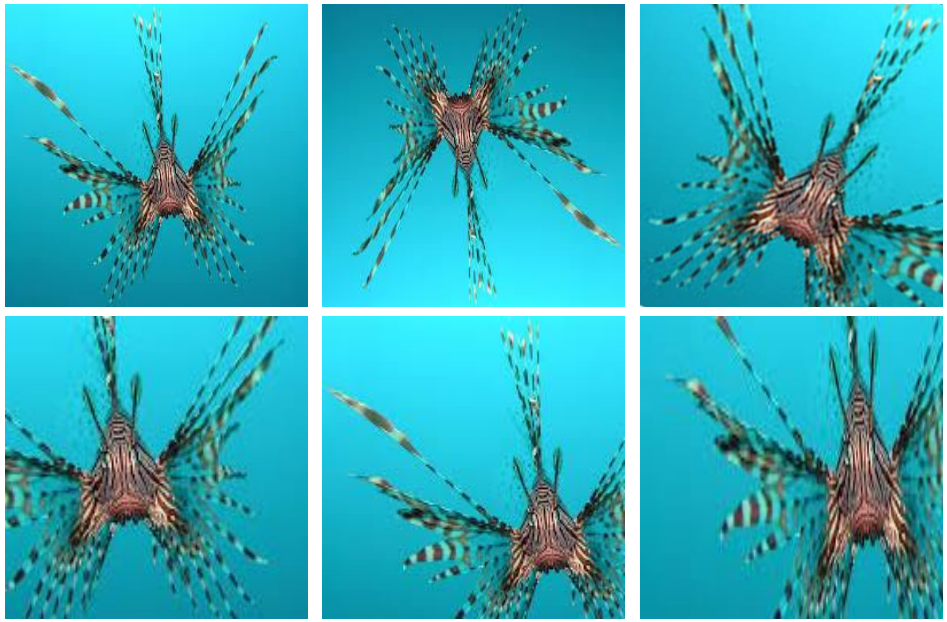
Where to find pretrained networks

- 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:

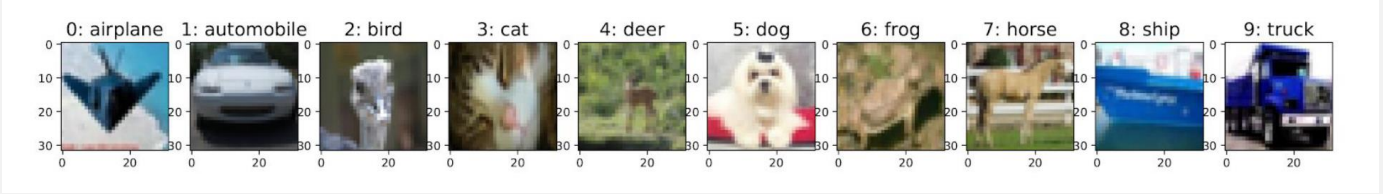
```
from keras.preprocessing.image import ImageDataGenerator

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]
)
```

Homework:
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

67

Summary

- NNs are loosely inspired by the structure of the brain.
  - When going deep the receptive field increases (~layer 5 sees whole input)
  - Deeper layer respond to more complex feature in the input

- 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