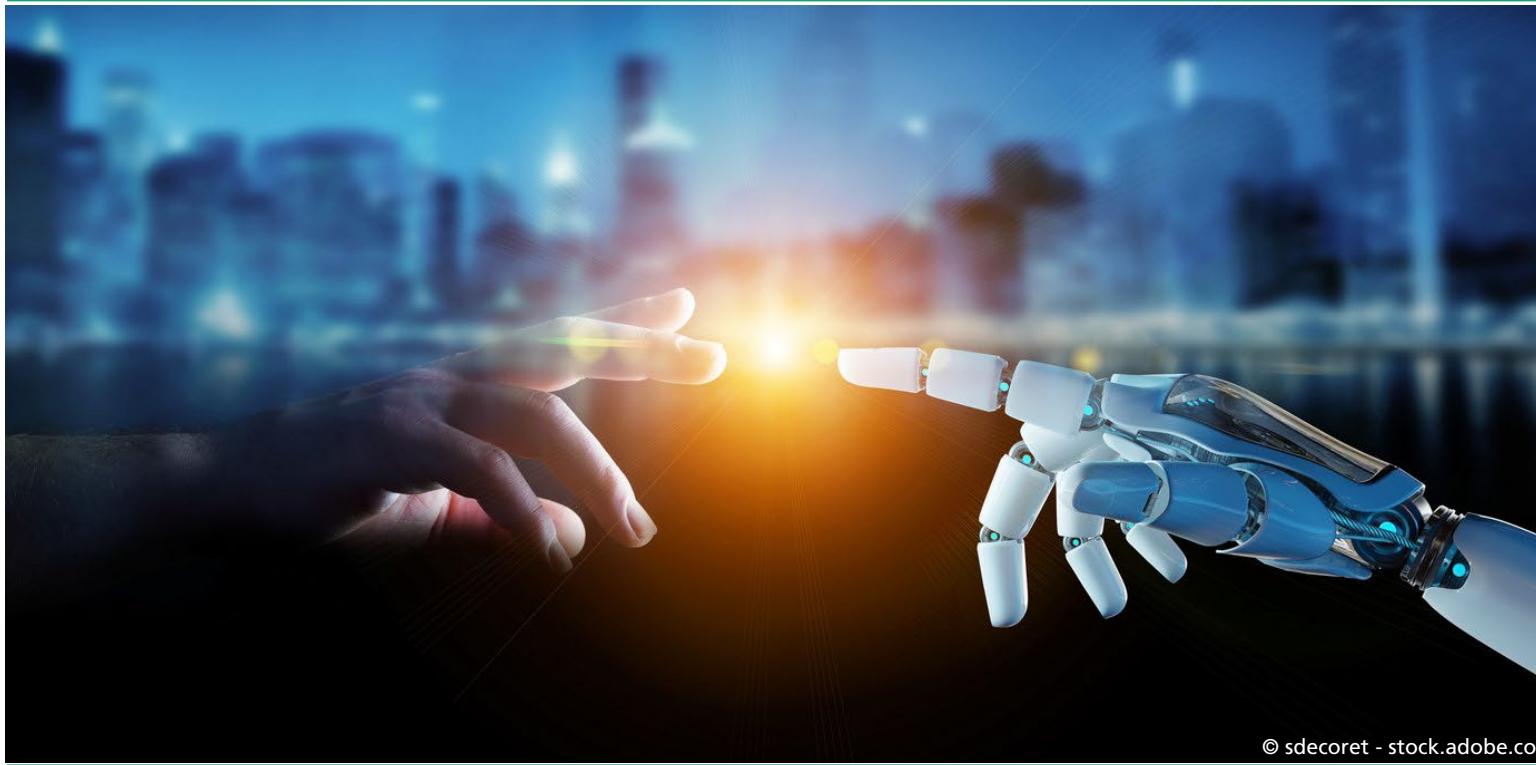


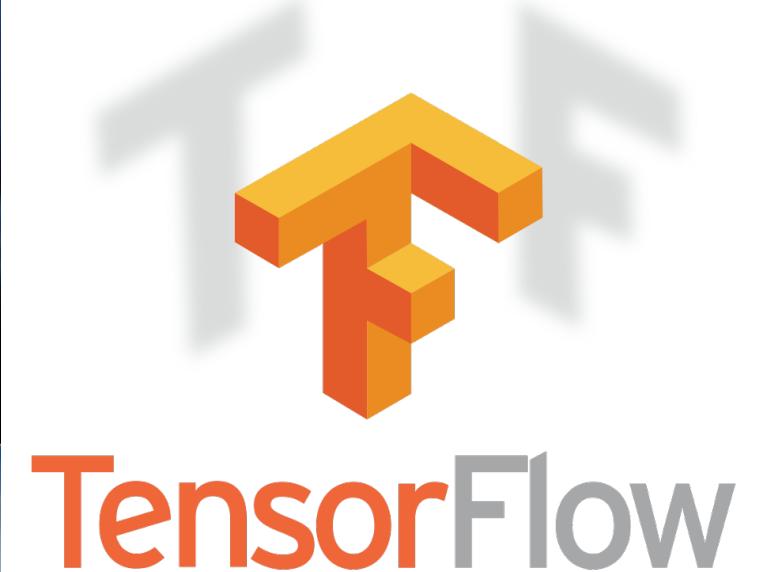
# Image Recognition

Dr. Gerhard Paaß

Fraunhofer Institute for Intelligent Analysis and Information Systems (IAIS)  
Sankt Augustin



© sdecoret - stock.adobe.com



TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc.  
Tensorflow Logo by TensorFlow - vectors combined, edited - Begoon / Apache 2.0

# Course Overview

- |   |   |
|---|---|
| 1. Intro to Deep Learning                 | Recent successes, Machine Learning, Deep Learning & types |
| 2. Intro to Tensorflow                    | Basics of Tensorflow, logistic regression                 |
| 3. Building Blocks of Deep Learning       | Steps in Deep Learning, basic components                  |
| 4. Unsupervised Learning                  | Embeddings for meaning representation, Word2Vec, BERT     |
| 5. Image Recognition                      | Analyze Images: CNN, Vision Transformer                   |
| 6. Generating Text Sequences              | Text Sequences: Predict new words, RNN, GPT               |
| 7. Sequence-to-Sequence and Dialog Models | Transformer Translator and Dialog models                  |
| 8. Reinforcement Learning for Control     | Games and Robots: Multistep control                       |
| 9. Generative Models                      | Generate new images: GAN and Large Language Models        |

 : link to background material,

 : link to images used in lecture,    G. : Terms that may be asked in the exam

# Image Recognition

## Agenda

1. Image Recognition Tasks
2. Convolutional Layers as local Feature Detectors
3. Simple Convolutional Neural Network
4. Advanced Convolutional Neural Networks
5. Vision Transformer
6. Semantic Segmentation
7. Summary

# Different Types of Image Recognition

## ■ Detect the objects in an image

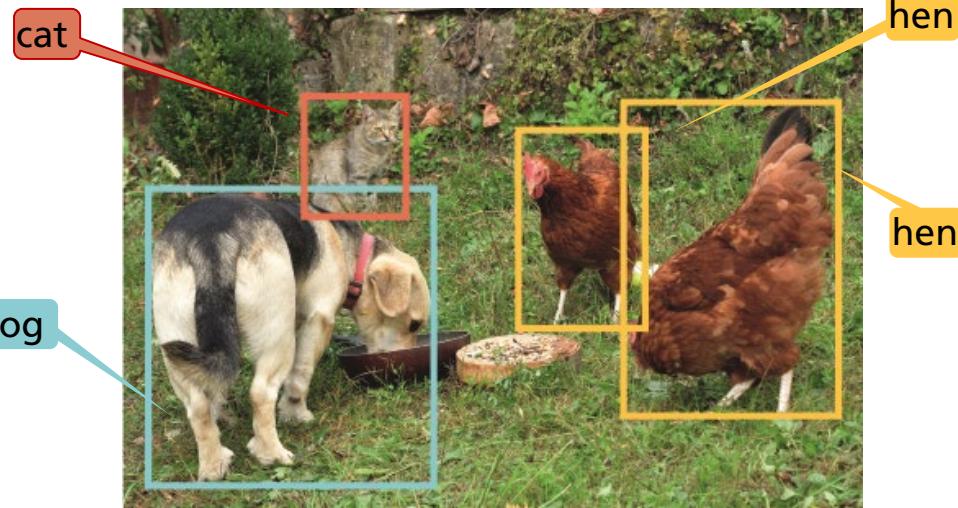
- Determine their type: classification
- Find the location of an object
  - Bounding box
  - Pixel of an object

## ■ Image Captioning: Describe the main contents of the image

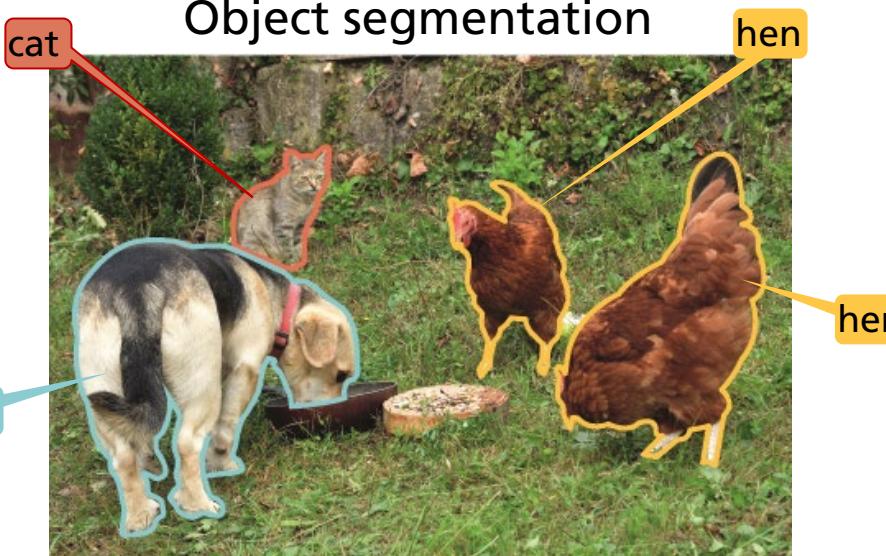
### Object classification



### Object localization



### Object segmentation



### Image Captioning

Two hens, a dog and a cat on a meadow

Deng et al. 2009: **ImageNet**: A Large-Scale Hierarchical Image Database [🔗](#)

22k categories and 15M images

Annotate the most prominent object



- Animals
  - Bird
  - Fish
  - Mammal
  - Invertebrate



"bird" by Mathias Appel / CC0 1.0

- Plants
  - Tree
  - Flower
  - Food
  - Materials



"Palm tree" by L. Braughler / CC BY 2.0

- Structures
- Artifact
  - Tools
  - Appliances
  - Structures



"S-396 Tango class submarine" by Peer.Gynt / CC BY-SA 2.0

- Person
- Scenes
  - Indoor
  - Geologic
- Sport



"Soccer Practice" by mill56 / CC BY 2.0

# ImageNet Classification Challenge

1000 object classes

1431167 images



**GT:** Stork

Criterion: Top-5 error

## Output

Nest

**Stork**

Chicks

Twigs

Eggshell



## Output

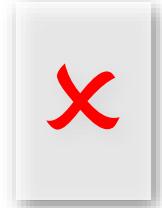
Nest

Bird

Chicks

Twigs

Eggshell

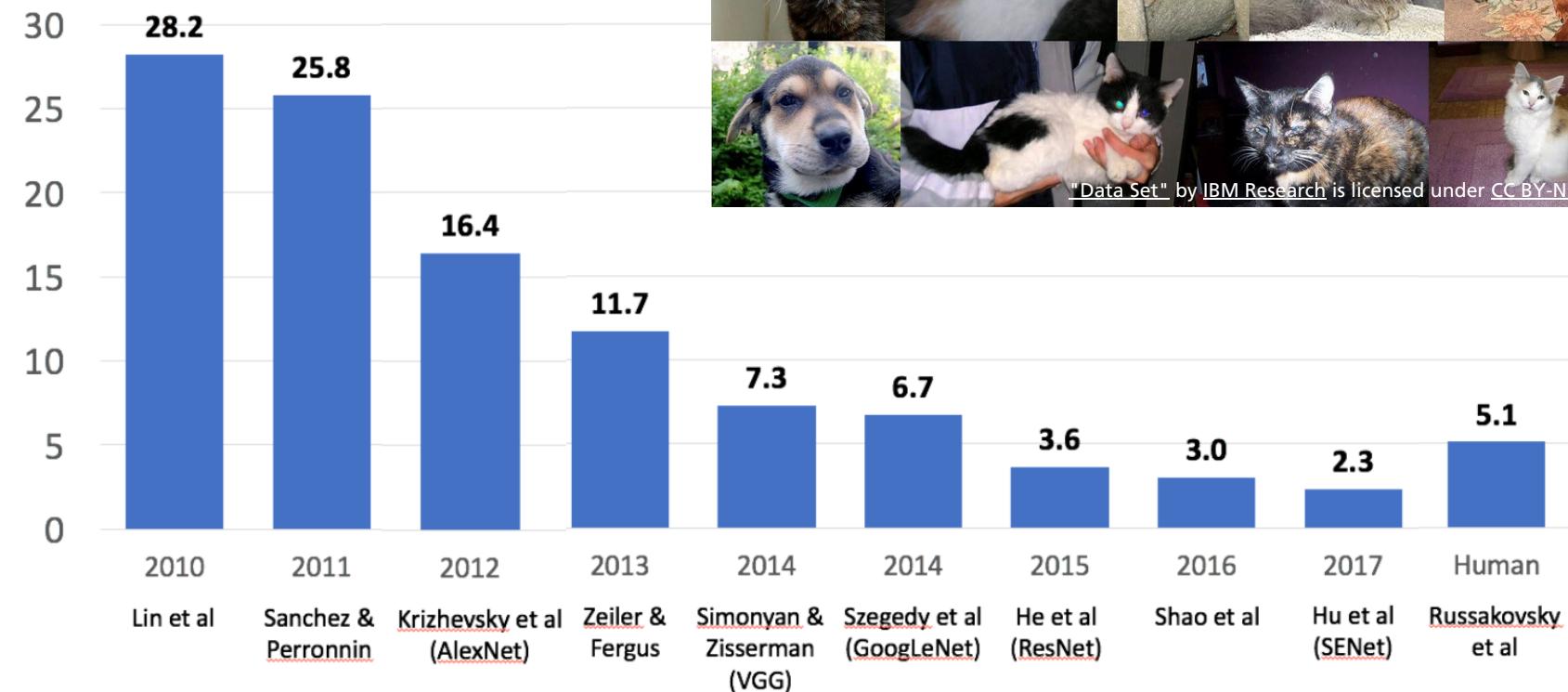


# ImageNet Classification Challenge: Results

1000 object classes

1431167 images

Top-5 error



"Data Set" by IBM Research is licensed under CC BY-ND 2.0

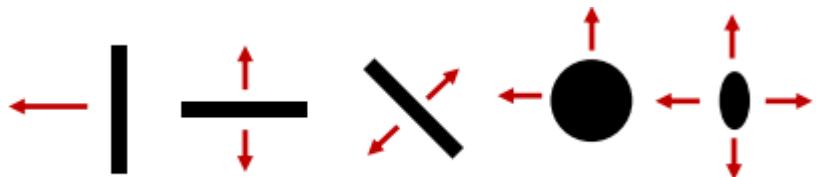
# Image Recognition

## Agenda

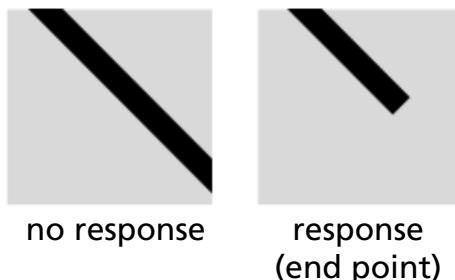
1. Image Recognition Tasks
2. Convolutional Layers as local Feature Detectors
3. Simple Convolutional Neural Network
4. Advanced Convolutional Neural Networks
5. Vision Transformer
6. Semantic Segmentation
7. Summary

# Hubel & Wiesel 1959

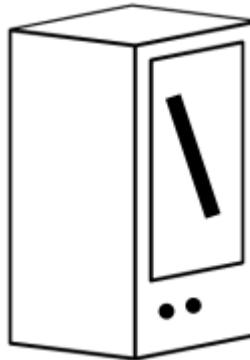
- Experiments with the visual cortex of cats
  - record nerve signals from single cells
- Simple cells:  
response to sticks of different orientation
- Complex cells:  
Response to light orientation & movement



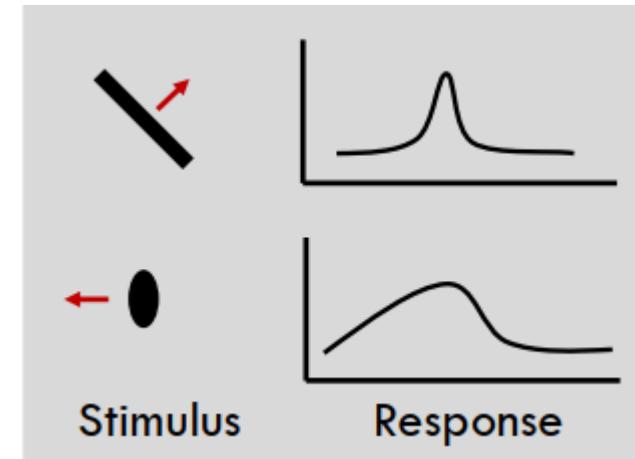
- Hypercomplex cells:  
response to movement with an end point



brain first detects local features  
which are processed further



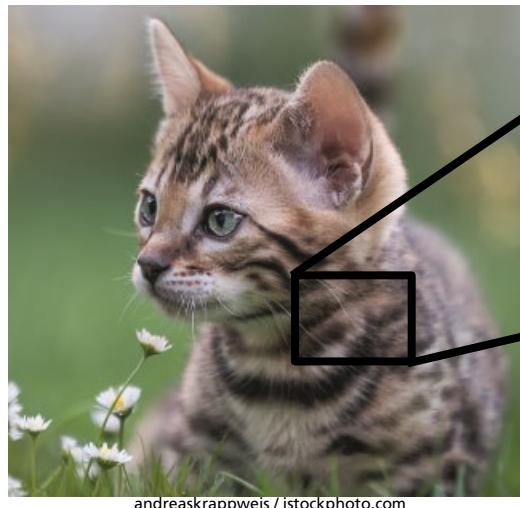
Stimulus



Stimulus

# Local Feature Detection: Convolution

- Define a small kernel: matrix, e.g. 3x3
- move kernel over input pixel matrix
- multiply kernel values with pixel values
- store sum in output field



105 112 108 133 104 99 106 99 96 103 132 119 104 97 93 971
1 76 89 98 105 128 105 87 86 93 99 125 112 105 103 99 951
1 93 98 103 106 104 79 98 103 99 105 123 116 105 94 95 951
1 99 98 95 93 123 131 127 108 85 98 102 99 95 93 101 941
1 106 95 82 84 88 95 88 85 105 107 109 98 79 84 86 951
1 114 109 65 55 55 69 64 64 54 64 87 102 129 90 74 84 911
1 133 103 102 108 105 85 85 85 93 93 94 84 84 102 93 85 821
1 133 137 144 149 149 149 149 149 149 149 149 149 149 149 149 1491
1 139 133 148 137 133 131 137 84 84 84 84 84 84 84 84 72 981
1 127 125 133 147 133 127 126 131 131 131 95 95 99 75 81 84 72 941
1 119 114 108 123 118 148 135 118 113 109 108 92 76 85 72 79 781
1 89 93 98 98 97 103 147 131 118 113 114 113 109 106 95 77 901
1 63 77 88 85 77 78 102 123 117 119 127 126 126 109 115 871
1 62 65 62 89 77 71 71 101 124 124 128 129 101 107 114 131 1101
1 64 65 62 89 77 71 71 101 124 124 128 129 101 107 114 131 1101
1 87 65 73 87 106 95 69 45 75 138 126 187 92 94 349 5121
1 118 97 92 86 117 123 116 66 41 51 95 93 99 95 349 5121
1 164 144 112 88 82 128 124 104 76 49 45 86 89 381 382 3891
1 157 178 157 128 93 86 114 132 132 87 69 55 79 82 82 99 941
1 133 128 134 165 133 148 109 118 121 134 134 87 85 53 89 961
1 128 128 134 157 151 144 128 115 104 107 107 107 81 72 911
1 123 147 147 147 147 147 147 147 147 147 147 147 147 147 147 1471
1 123 147 147 147 147 147 147 147 147 147 147 147 147 147 147 1471
1 123 147 147 147 147 147 147 147 147 147 147 147 147 147 147 1471
1 122 164 148 109 75 98 78 83 93 103 109 109 102 81 89 8471

represent image  
by a pixel matrix

Image

50	200	235	201		
15	135	88	100		
0	42	77	165		
6	108	250	144		

Convolution  
Kernel

-1	-1	-1
-1	8	-1
-1	-1	-1

Resulting  
Feature Matrix


0	0	0	0	0	0
0	50	200	235	201	0
0	15	135	88	100	0
0	0	42	77	165	0
0	6	108	250	144	0

\*

Kernel = 3x3
Stride = 1
Padding = 0

-1	4
-3	2

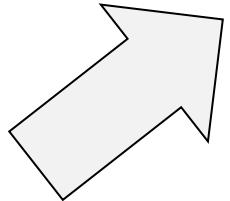

Kernel = 2x2
Stride = 2
Padding = 1

# Example: Edge detection

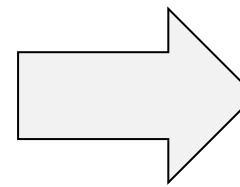
- Apply kernel with specific values



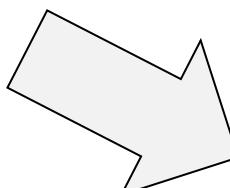
[Before any kernel convolution](#). By Michael Plotke CC BY-SA 3.0



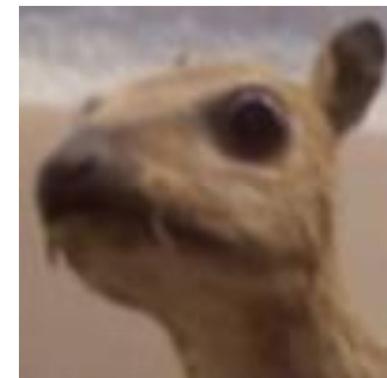
$$\odot \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$



$$\odot \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} =$$



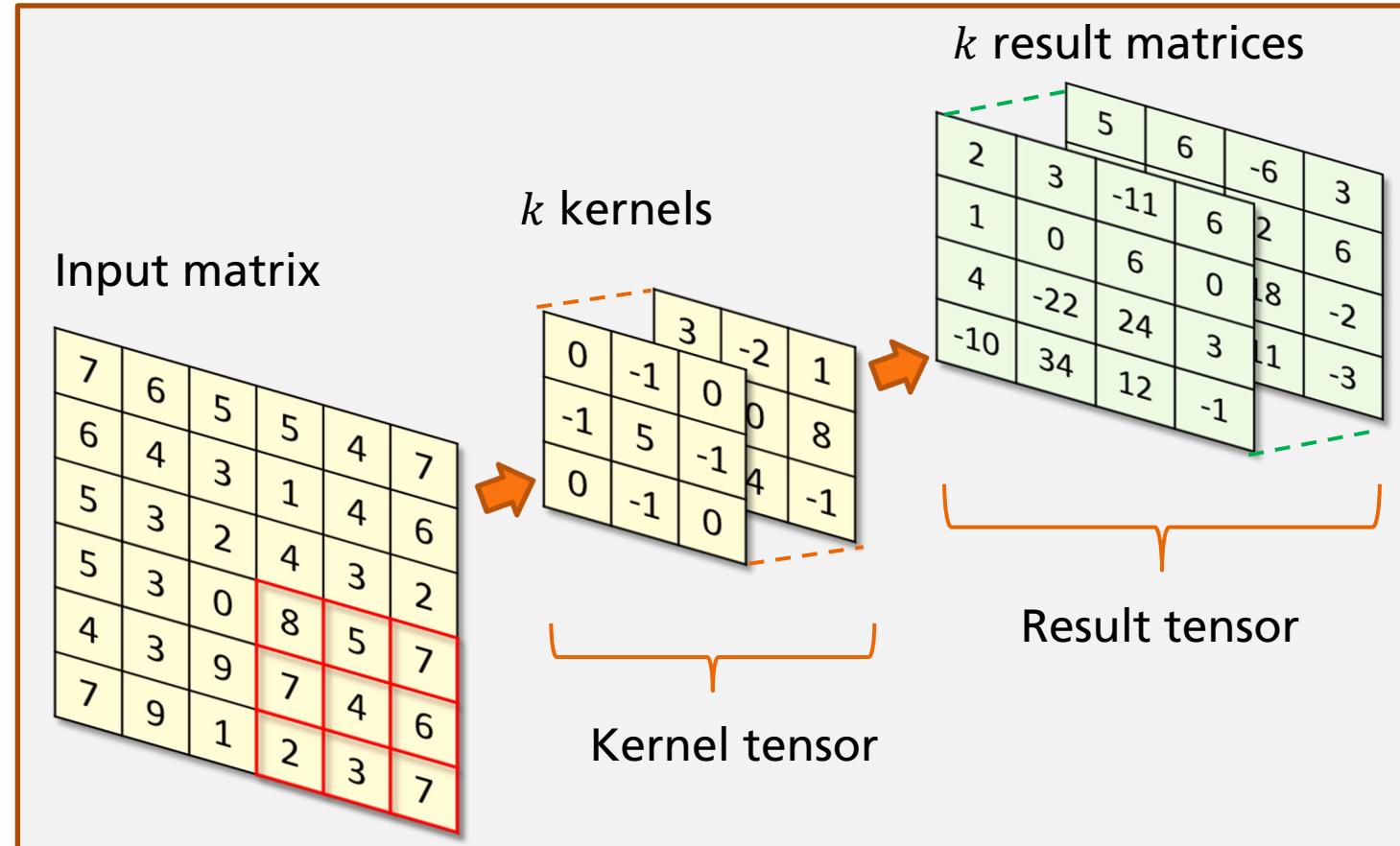
$$\odot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 8 & 1 \\ 1 & 1 & 1 \end{bmatrix} =$$



Kernel (image processing)

# Convolution: Filter bank

- **Filter bank:** Convolutional layer with  $k$  independent kernels
- number of kernels can be selected
- Values in kernels are parameters: Adapted to training data
- Convolutions with multiple filters change the size of the output ("hidden units")



# Convolution for RGB Images

- Apply  $3 \times 3 \times 3$  kernel to the three input pixel matrix

0	0	0	0	0	0	0	...
0	156	155	156	158	158	158	...
0	153	154	157	159	159	159	...
0	149	151	155	158	159	159	...
0	146	146	149	153	158	158	...
0	145	143	143	148	158	158	...
...	...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	0	...
0	167	166	167	169	169	169	...
0	164	165	168	170	170	170	...
0	160	162	166	169	170	170	...
0	156	156	159	163	168	168	...
0	155	153	153	158	168	168	...
...	...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	0	...
0	163	162	163	165	165	165	...
0	160	161	164	166	166	166	...
0	156	158	162	165	166	166	...
0	155	155	158	162	167	167	...
0	154	152	152	157	167	167	...
...	...	...	...	...	...	...	...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

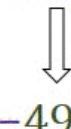
Kernel Channel #1



308

1	0	0
1	-1	-1
1	0	-1

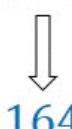
Kernel Channel #2



-498

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



164

+

+ 1 = -25

Bias = 1  
↑

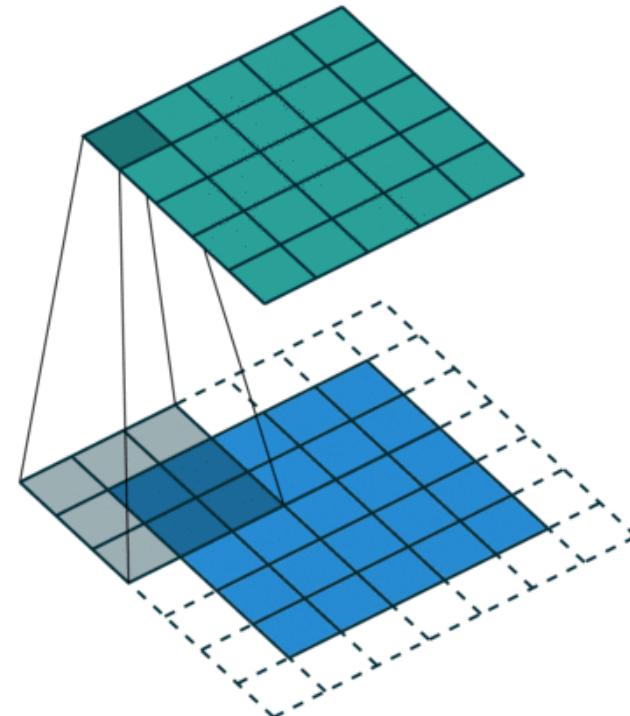
-25				...
				...
				...
				...
...	...	...	...	...

# Image Margins and Strides

- images are often padded with 0s
- otherwise:  
result matrix is  
smaller

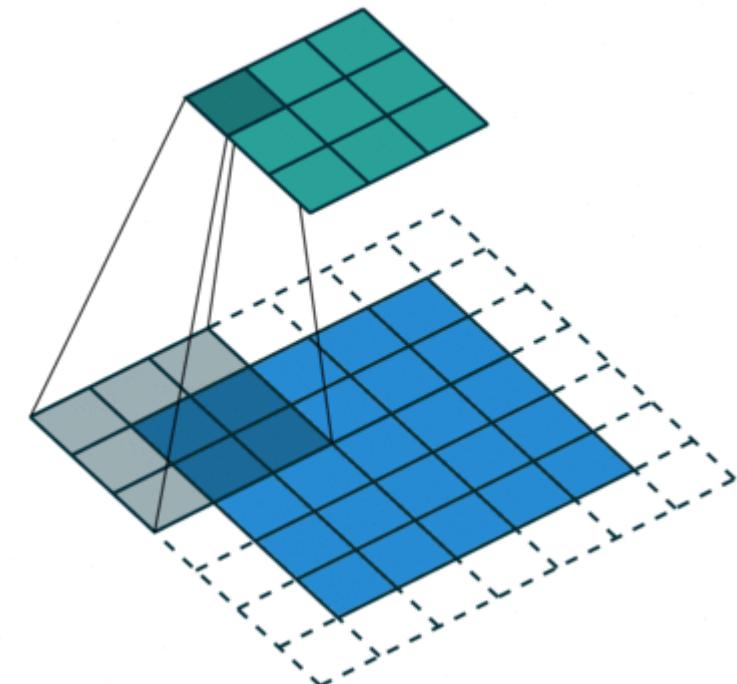
"same padding"

- Apply 3x3 kernel with stride 1



"valid padding"

- Apply 3x3 kernel with stride 2



# Convolutional Layers

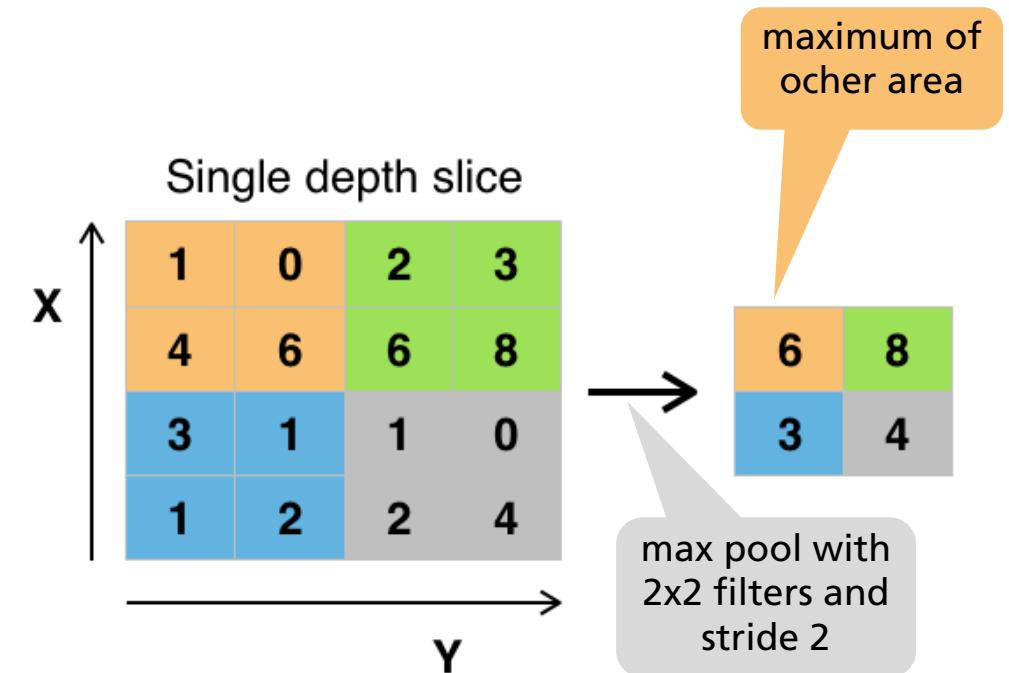
- Extract local features:
  - compute for receptive field of the kernel
  - specific bias for image recognition
- Same kernel is used for all positions in pixel matrix
  - parameter sharing
    - 1000x1000 pixel matrix and fully connected layer with 100 outputs  
→  $1000000 * 100 = 100M$  parameters
    - 1000x1000 pixel matrix and 100 kernels of size 3x3  
→ 900 parameters
- **Pro:** very flexible, few parameters
- **Con:** compute intensive (but there are special GPU-instructions to speed up)

# Pooling Layer

- By selecting many kernels for the convolutional layer we get very many features
  - $m \times m$  input matrix,  $k$  kernels  $\rightarrow \sim m * m * k$  output features  
 $\rightarrow$  need a way to reduce the number of features

## Pooling layer

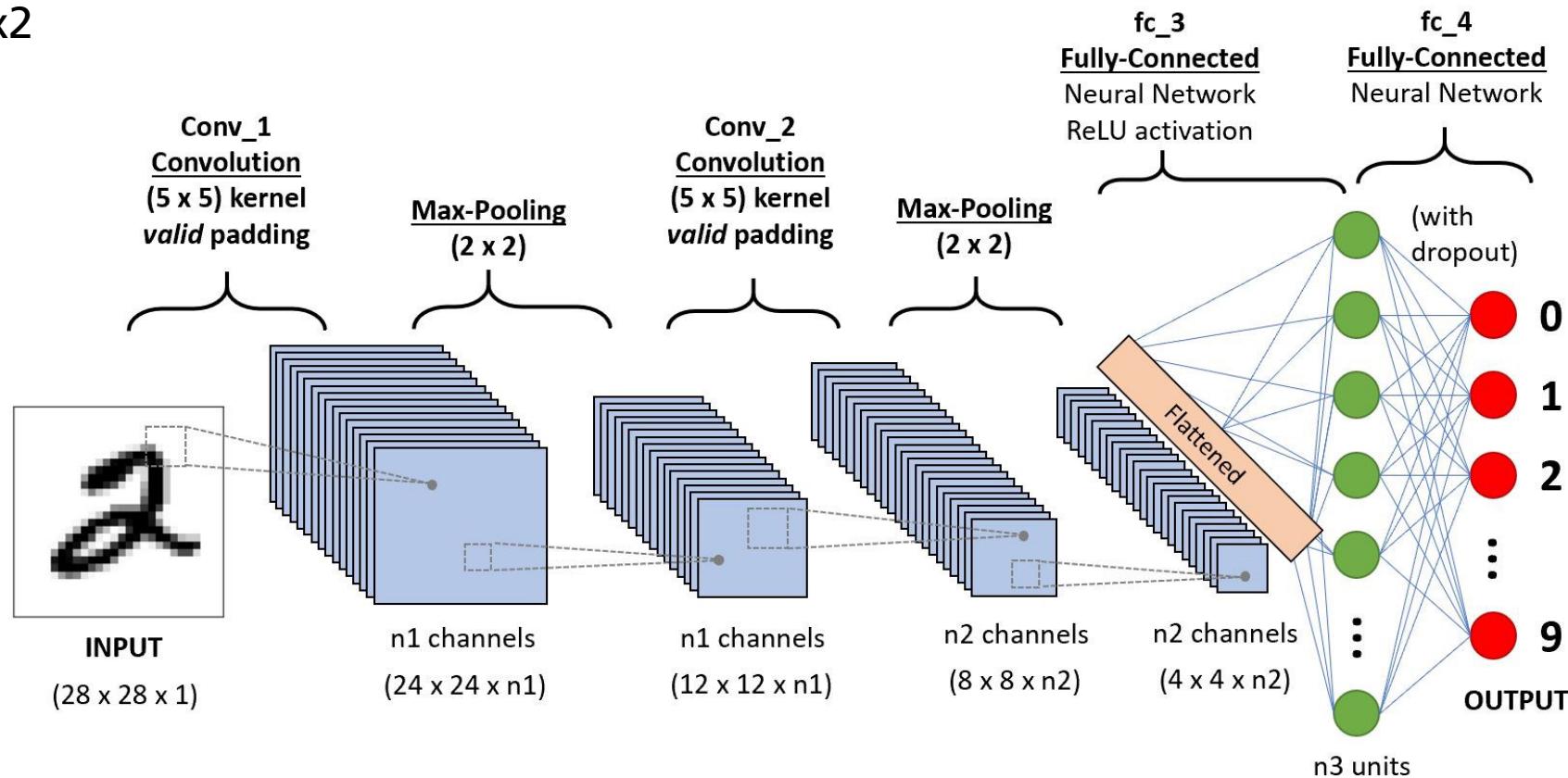
- Aggregate feature values in a feature matrix in a small field of size  $r \times r$
- usually separate aggregation for each feature matrix
- reduction of number of features by factor  $1/(r * r)$
- Aggregation function: mean, max, ...
- max-Averaging usually has the best performance



# Convolutional Neural Network

- Convolutional Neural networks (**CNN**) are a cascade of convolutional layers
  - convolution, e.g. with 5x5 kernels and ReLU activation
  - Max-Pooling, e.g. with 2x2
  - fully connect layer
  - final softmax-layer  
→ class probabilities

- Higher layers of CNN
  - have lower spatial resolution
  - more features per position



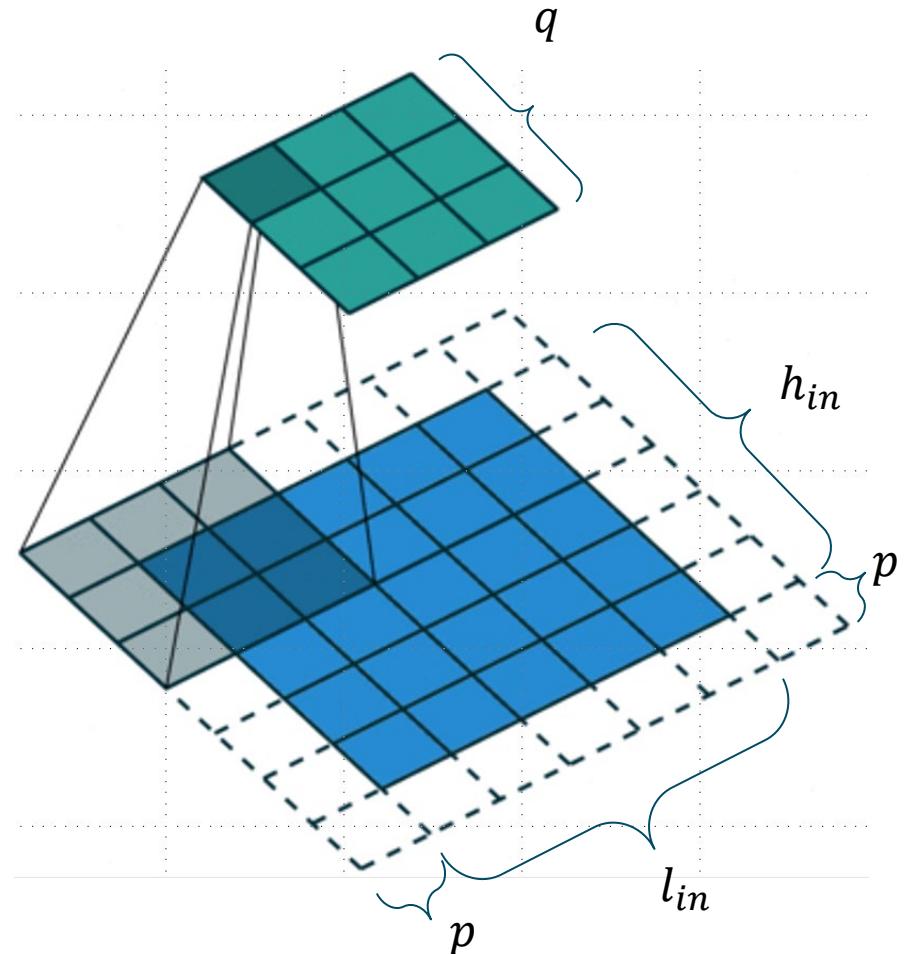
# Image Recognition

## Agenda

1. Image Recognition Tasks
2. Convolutional Layers as local Feature Detectors
3. Simple Convolutional Neural Network
4. Advanced Convolutional Neural Networks
5. Vision Transformer
6. Semantic Segmentation
7. Summary

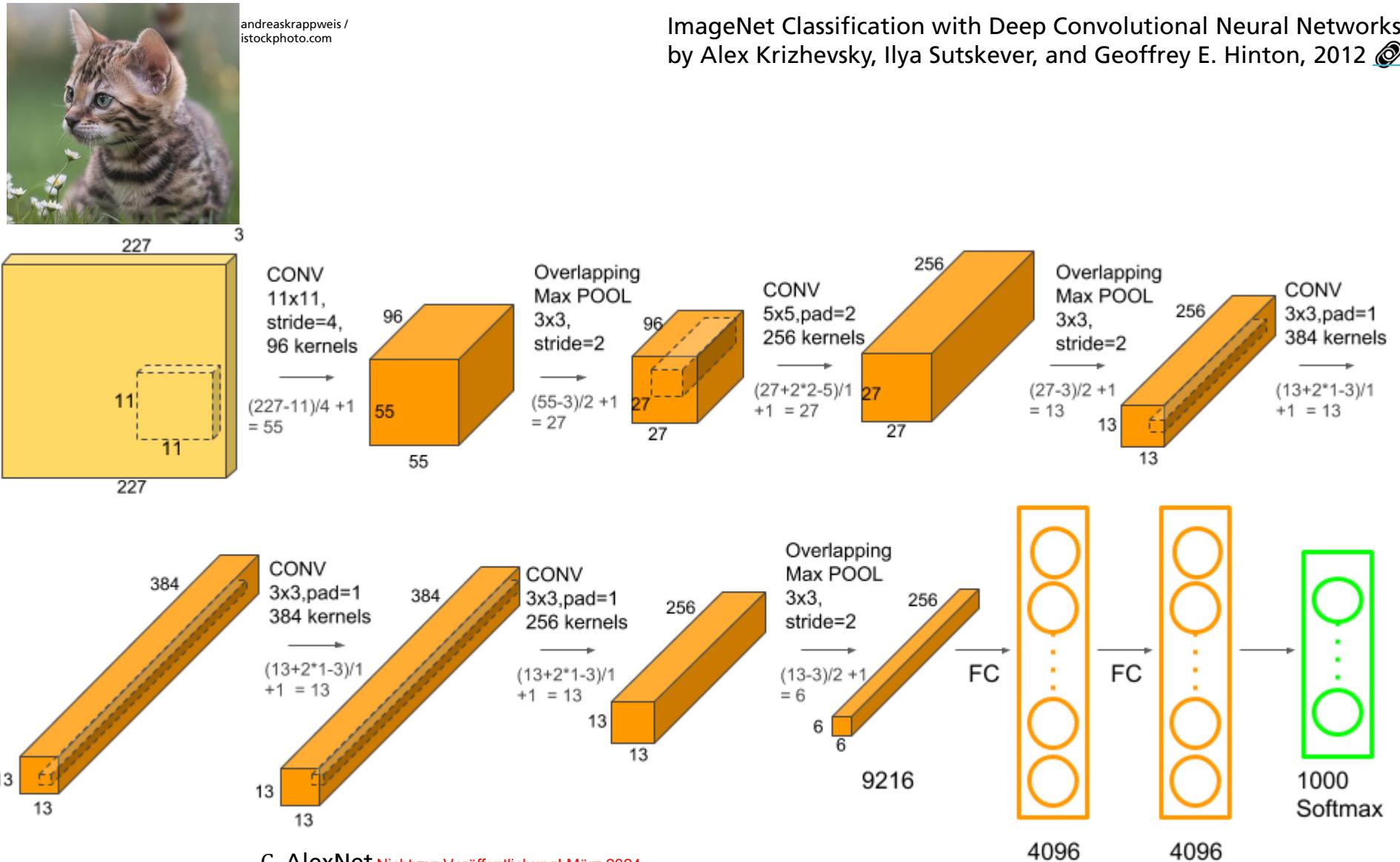
# Convolutional Layer Dimensions

- Accepts an input tensor of size  $l_{in} \times h_{in} \times m_{in}$  (width, height, num. features)
- Requires four hyperparameters:
  - number of filter / kernels  $k$
  - their spatial extent  $q \times q$
  - the stride (steplength)  $s$
  - the amount of zero-padding  $p$
- produces an output tensor of size  $l_{out} \times h_{out} \times m_{out}$ 
  - $l_{out} = \frac{l_{in}-q+2p}{s} + 1$
  - $h_{out} = \frac{h_{in}-q+2p}{s} + 1$
  - $m_{out} = k$
- with parameter sharing:  $q \times q \times m_{in}$  weights per kernel  
➔  $(q \times q \times m_{in}) * k$  weights and  $k$  biases



# Typical CNN Design: AlexNet

- convolutional layers (CONV) with Relu Activation
- max pooling (Max POOL)
- Dropout for regularization
- fully connected layers (FC) to aggregate
- Softmax to compute class probabilities



# CNN Training

- Training set  $(x_1, y_1), \dots, (x_n, y_n)$  with Image-label pairs Kernel (image processing)
- Initialize parameters  $w^{(0)}$  (kernel values, etc.) randomly.
- loop over batches of training data
  1. forward propagation with the current  $w^{(t)}$
  2. compute gradient for the next batch:  $grad$
  3. update parameter
$$w^{(t+1)} = w^{(t)} - \lambda * grad$$
  4. if  $\|grad\|^2 < \epsilon$  stop  
else continue with step 1
- High computational effort: 90 epochs  
took 5-6 days on two NVIDIA GTX 580 3GB GPUs

# Kernel Features

- Inputs with highest responds of different kernels
  - lines with different orientation and width
  - color blobs of different colors
  - textures

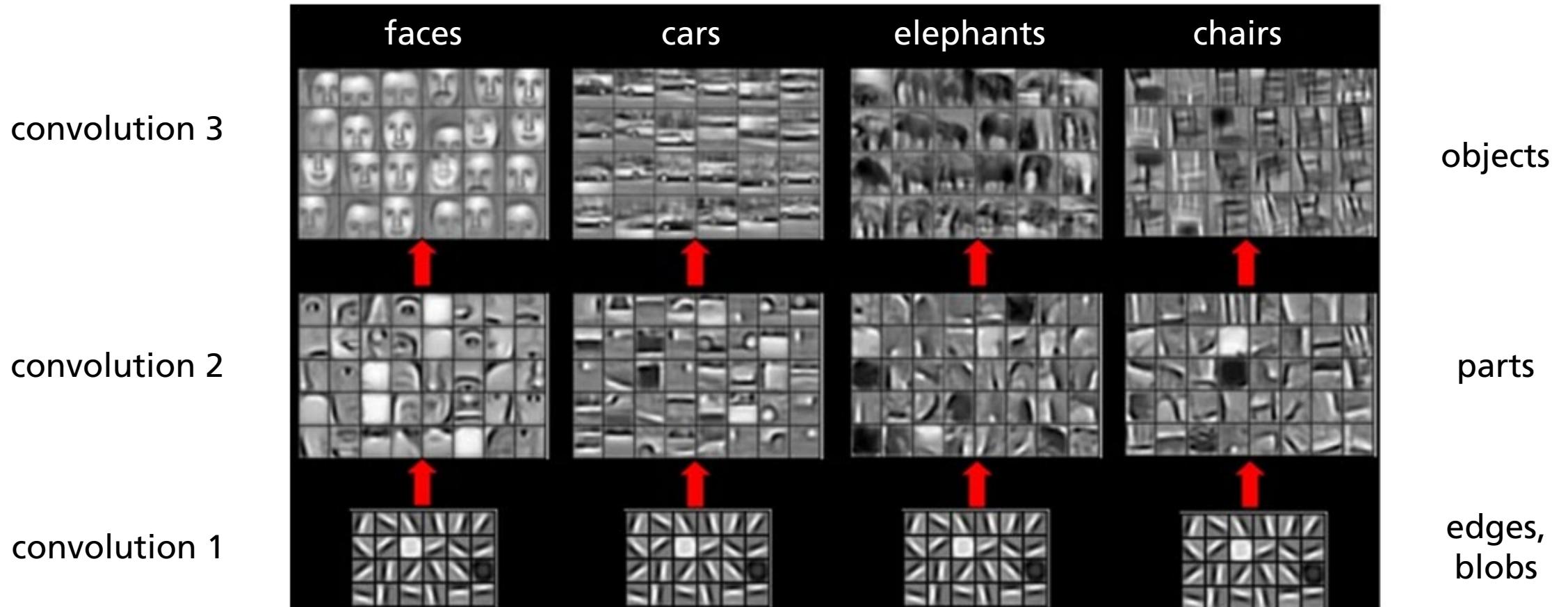


Krizhevsky et al. 2012 : ImageNet Classification with Deep Convolutional Neural Networks. p.6 

reduced top-5 error in ImageNet  
from 25.5% to 17.0%

33% reduction

# CNN: Intermediate features



# Hidden Vector Features

- compute feature vector  $f(x)$  in the last hidden layer for test image  $x$ .
- compute feature vectors  $f(x_i)$  in the last hidden layer for all training images  $x_i$ .
- determine the closest images

$$j = \operatorname{argmin}_i \|f(x) - f(x_i)\|^2$$

- good characterization of the image by hidden vector

may be used for image search

input  
image



"Rose" by Oberau-  
Online / CC BY 2.0



"Rose laser バラ  
レーザー" by T.Kiya  
/ CC BY-SA 2.0



"Rose Carina バラカ  
リーナ" by T.Kiya /  
CC BY-SA 2.0



"Rose, Princess Hanako, バ  
ラ, プリンセス ハナコ," by  
T.Kiya / CC BY-SA 2.0

Closest training images

[Krizhevsky et al. 2012] p.8 

# Image Recognition

1. Image Recognition Tasks
2. Convolutional Layers as local Feature Detectors
3. Simple Convolutional Neural Network
4. Advanced Convolutional Neural Networks
5. Vision Transformer
6. Semantic Segmentation
7. Summary

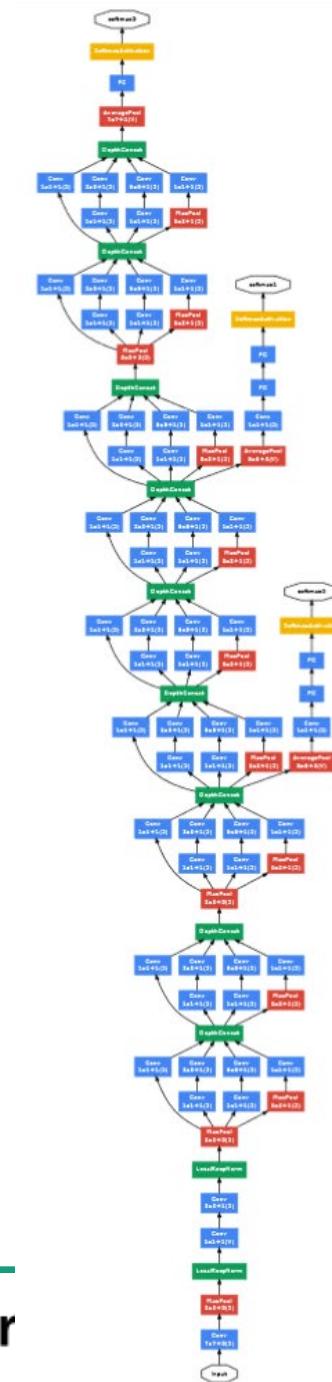
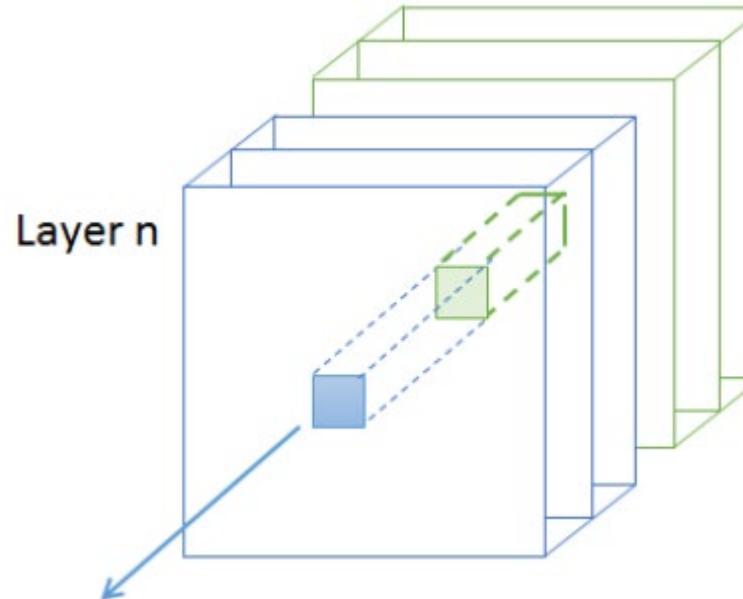
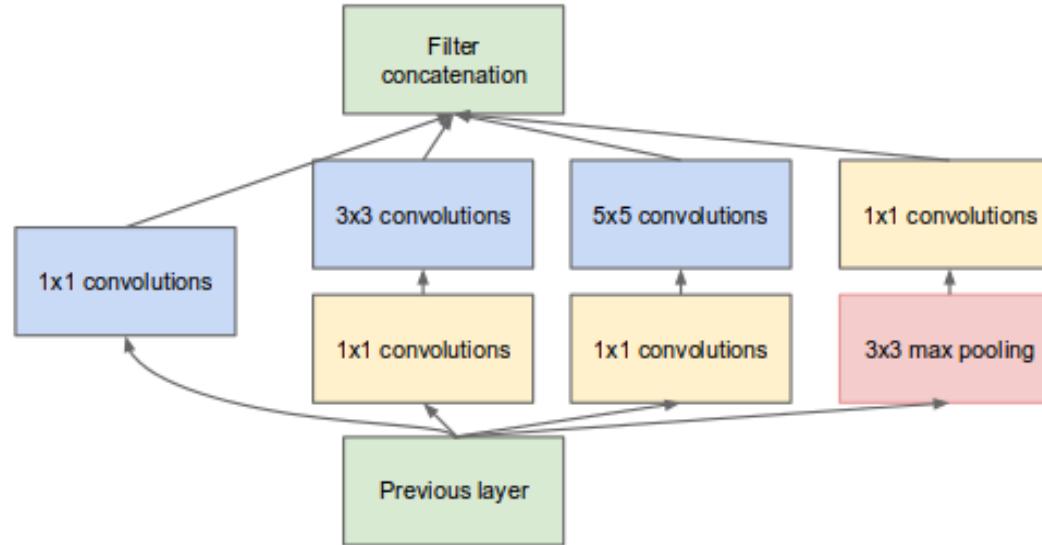
# Inception Net

- First really deep net
  - ~ 160 layers
  - ~ 60 convolutions

- Only one fully connected layer
- uses dropout regularization

## Inception Module

- process input with kernels of different size
- 1x1 convolutions:  
at each position compute linear mapping for values at same position  
→ downsample features



Szegedy et al. 2015: Going deeper with convolutions. [🔗](#)

# Inception Net

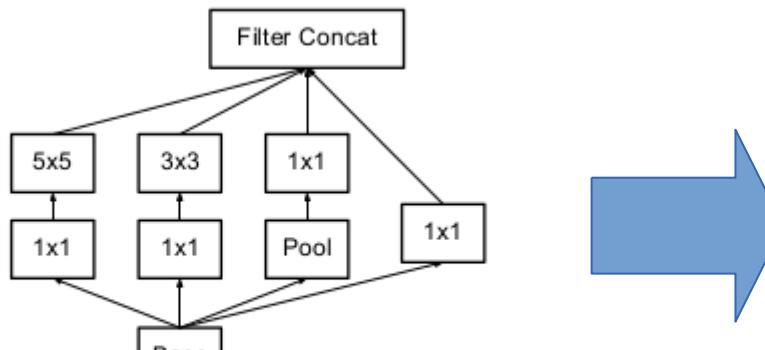
## ■ Regularization:

- Dropout layer: "drops" random portion of connections for one iteration

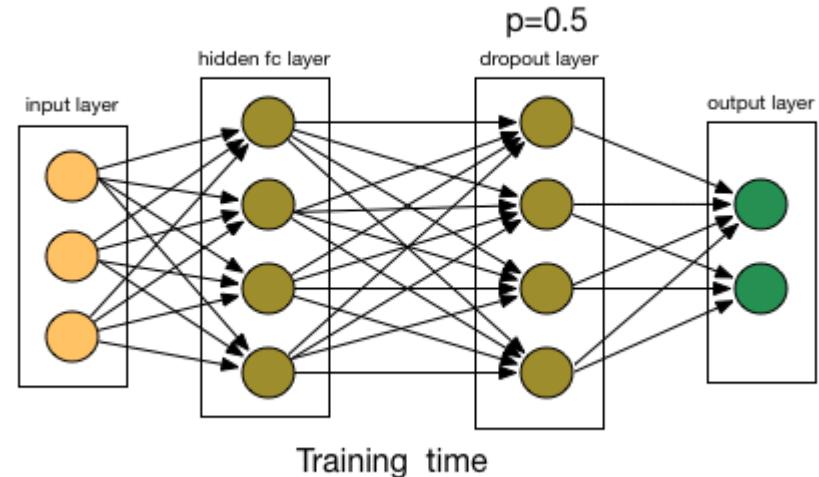
## ■ Separable convolution

- Replace larger kernels by 2 successive smaller kernels:  
e.g. 5x5 by 3x3 followed by 3x3

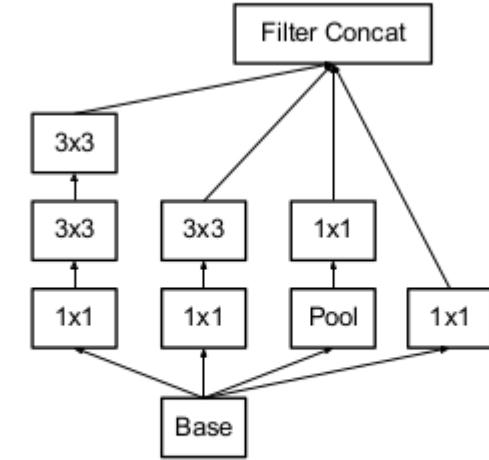
- less weights  
→ less memory & computation



Inception module version 1



Training time



Inception module version 3

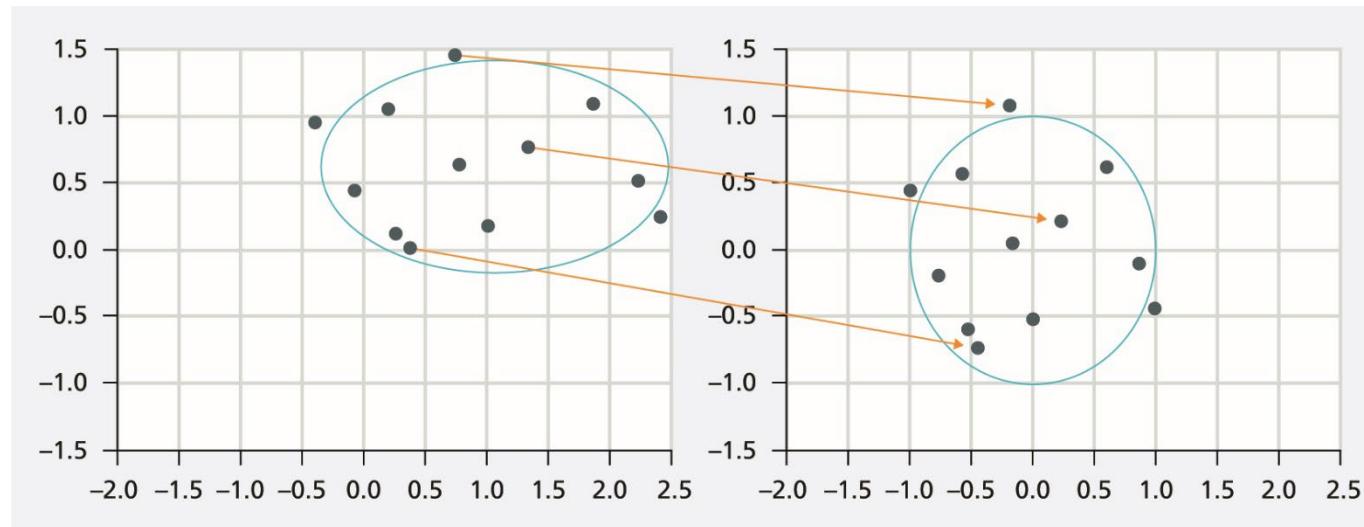
Szegedy et al. 2015: Going deeper with convolutions.

# Inception Net

## ■ Regularization:

- Batch Normalization
- Renormalization of layer outputs (batch by batch)
  - Fixes the covariance-shift between batches
  - Avoids the "*vanishing gradient*" problem
  - need to compute batch statistics

- Input: values of a feature for a minibatch  $x_1, \dots, x_m$
- compute mean  $\mu = \frac{1}{m} \sum_{i=1}^m x_i$
- compute variance  $\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu)^2$
- normalize  $\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$
- transform  $y_i = \gamma \hat{x}_i + \beta$  with trainable parameters  $\gamma$  and  $\beta$



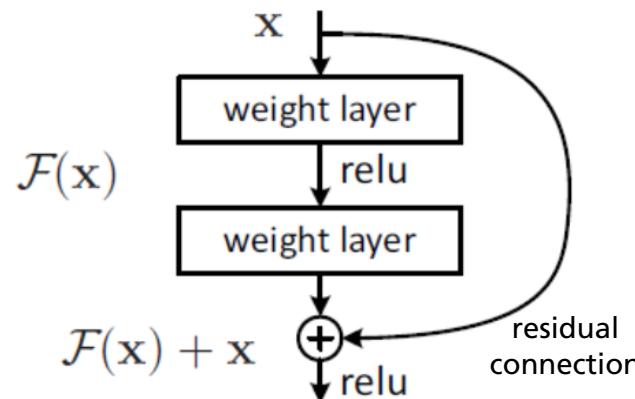
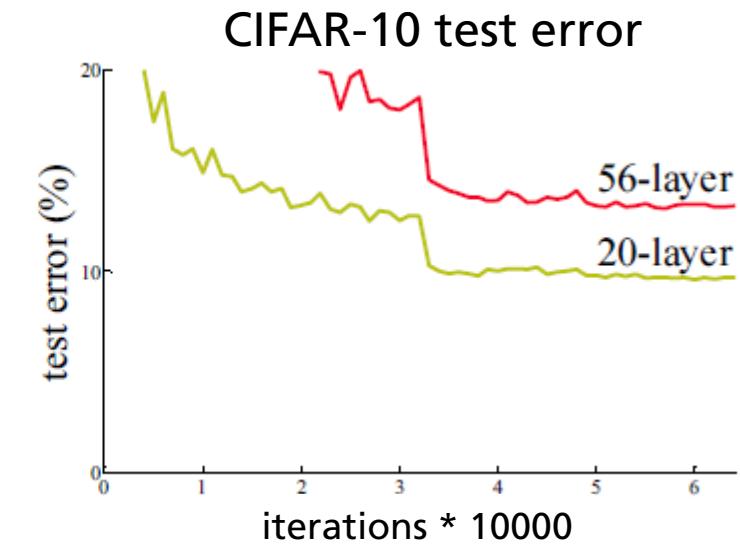
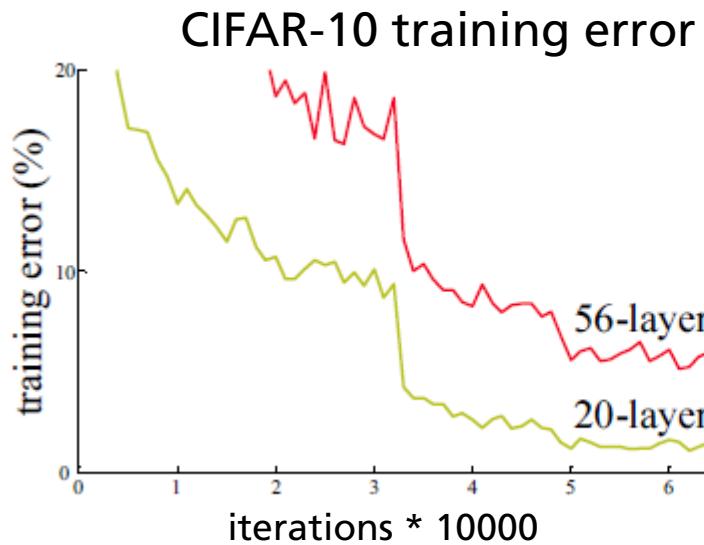
- Performance on the ImageNet benchmark
  - 3.58% top-5 error

Szegedy et al. (2016). Rethinking the inception architecture for computer vision. [🔗](#)

# ResNet: newer architecture for deep CNNs

[He et al. 2015] p.770 

- Training of deep networks is difficult
  - 20-layer network has lower error than a 56-layer network
  - deeper networks: **problem of vanishing/exploding gradients**
- Skip / residual connections
  - a skip / shortcut connection is added to add the input  $x$  to the output after few weight layers
  - The skip connection is added to output



at start with small weights  
→ identity connection

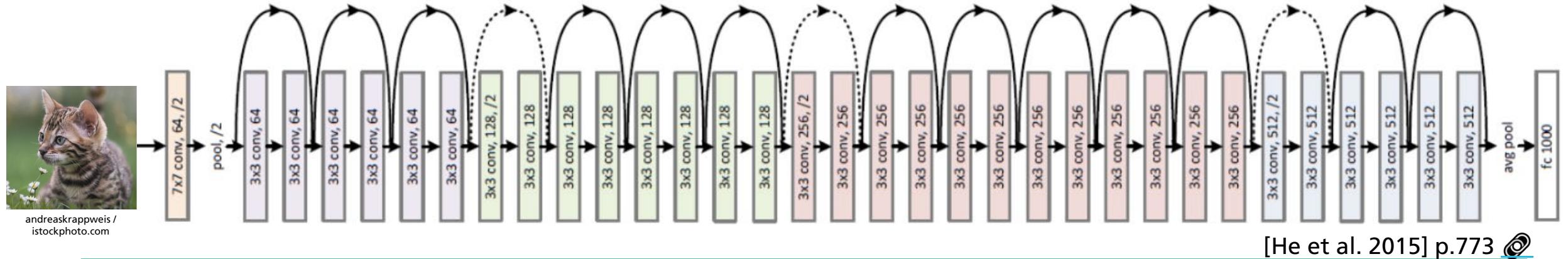
[He et al. 2015: Deep residual learning for image recognition.]

# ResNet

- Resnet architecture allows to use very many layers
- Batch normalization for regularization, no dropout
- SGD + Momentum
- minibatch size 256

## ■ Experimental Results

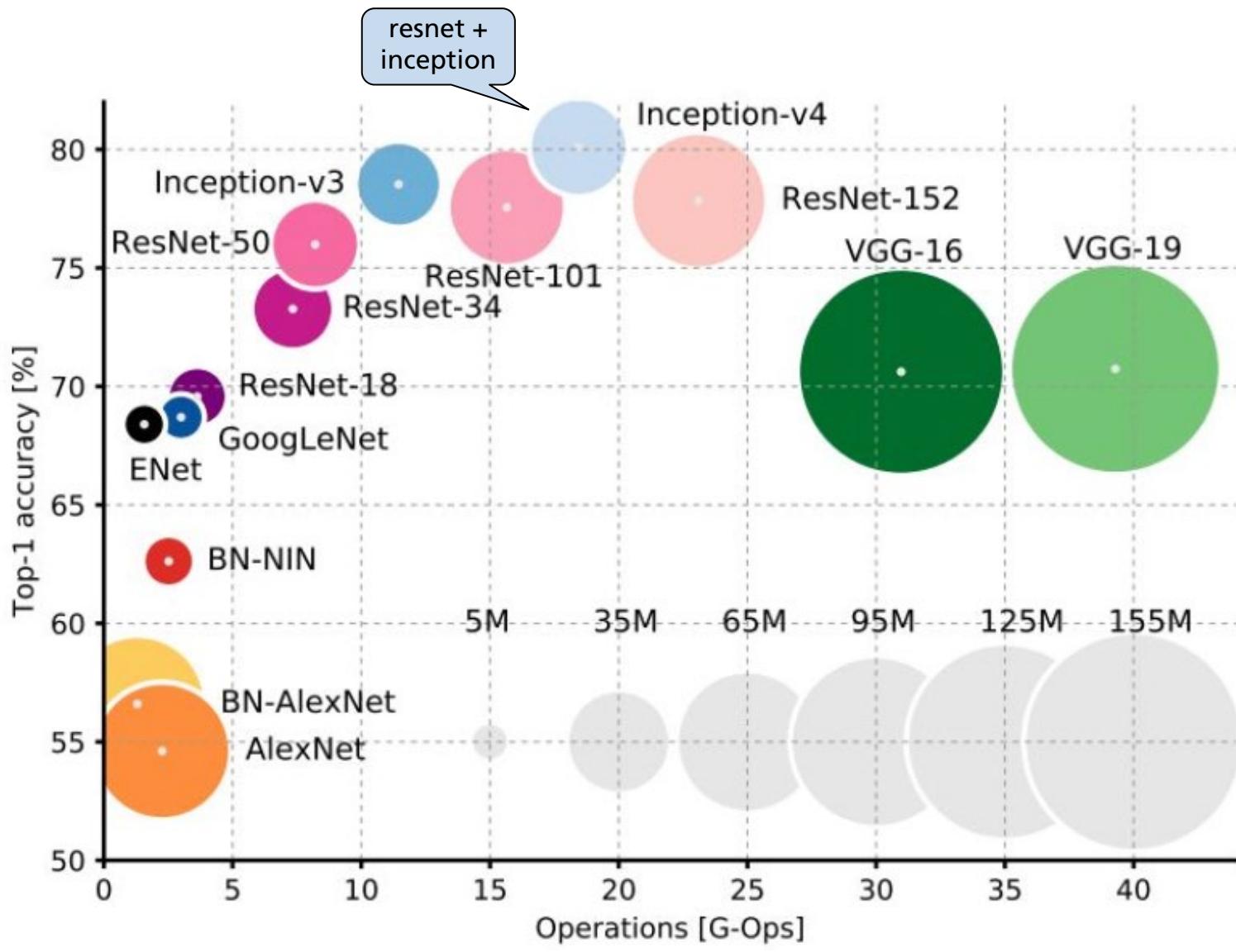
- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 layers on Cifar)
- Deeper networks now achieve lower training error as expected
- 3.6% top 5 error in ImageNet '2015 better than human performance



[He et al. 2015] p.773

# Comparing Complexity

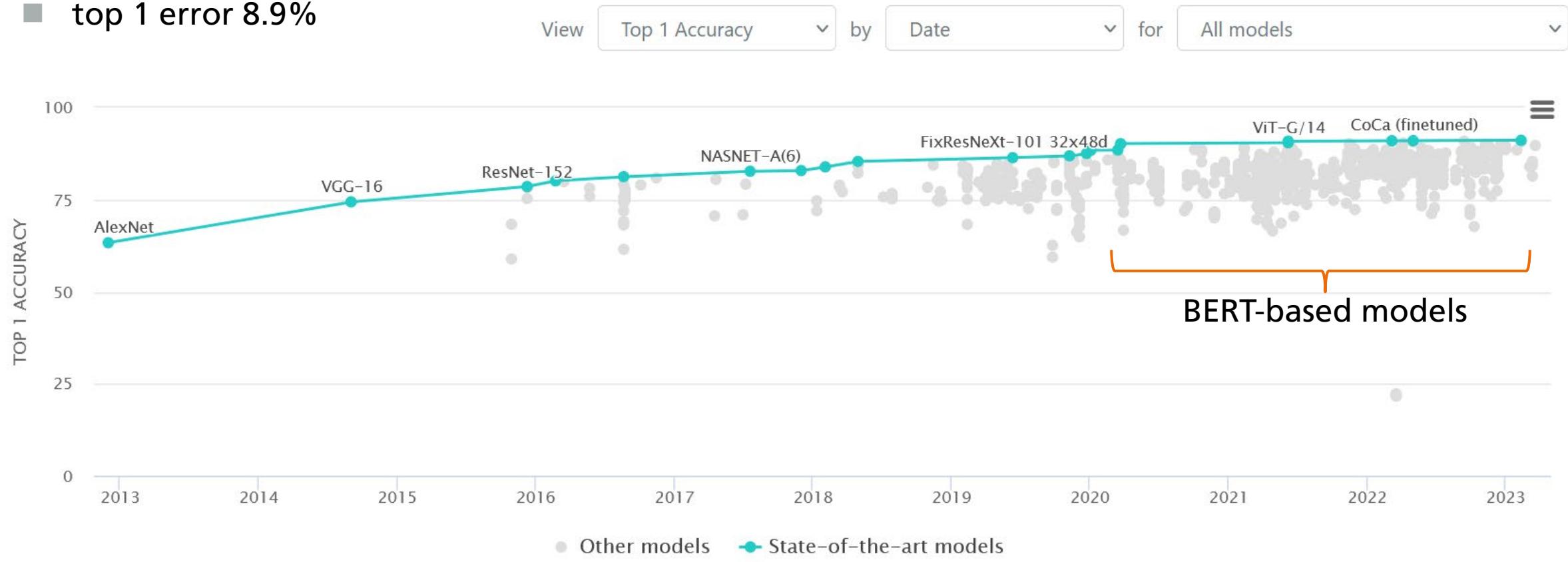
- ResNet has not very many parameters compared to its performance
- The computational effort is relatively low



[Canziani, Paszke, Culurciello 2017] p.2

# Improving ResNet

- Current state-of-the art:
  - top-5 error 0.98%
  - top 1 error 8.9%



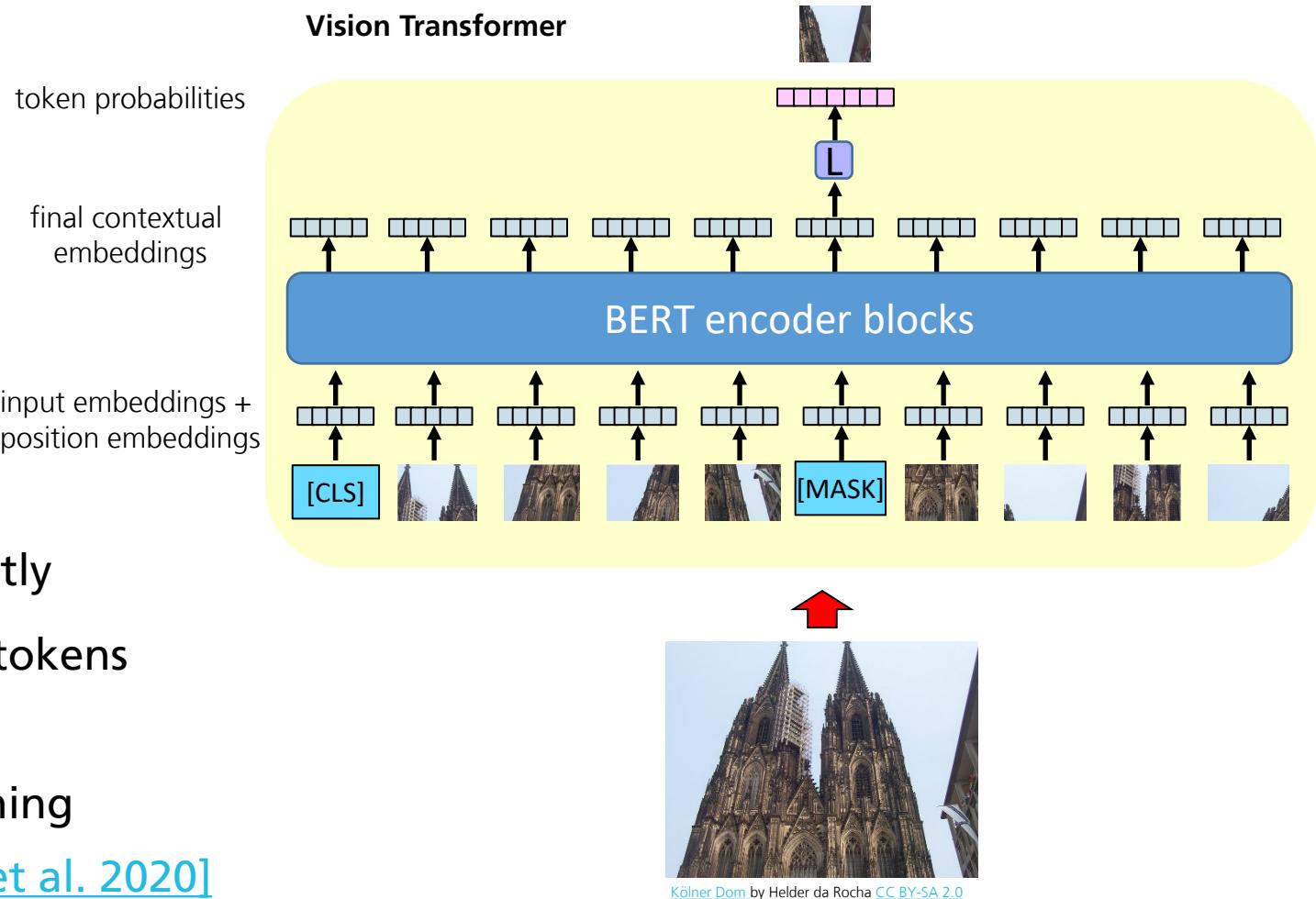
<https://paperswithcode.com/sota/image-classification-on-imagenet>

# Image Recognition

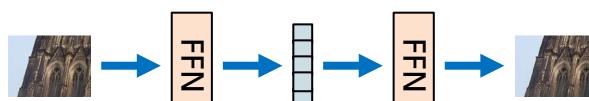
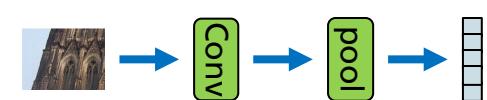
1. Image Recognition Tasks
2. Convolutional Layers as local Feature Detectors
3. Simple Convolutional Neural Network
4. Advanced Convolutional Neural Networks
5. Vision Transformer
6. Semantic Segmentation
7. Summary

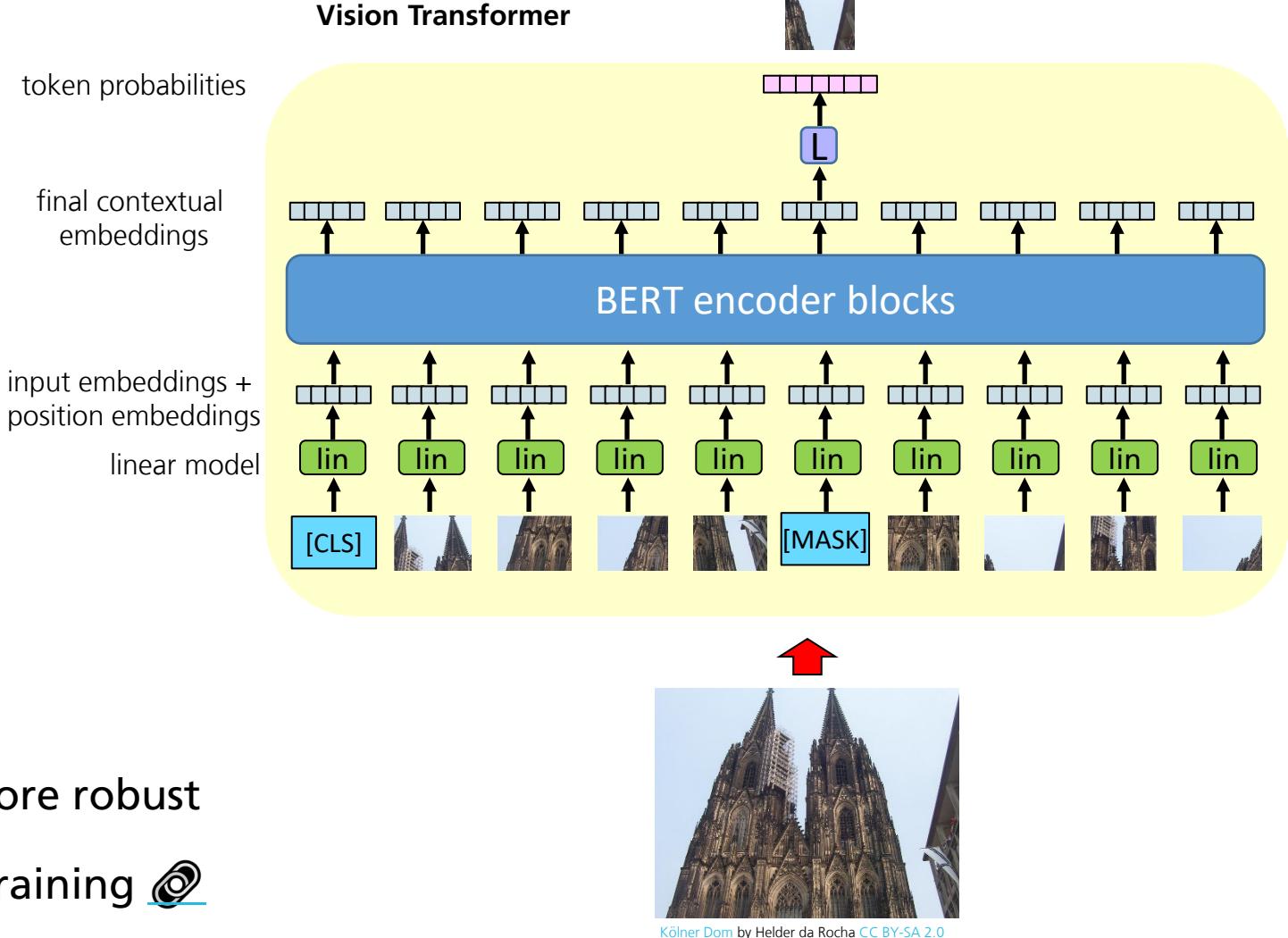
# Vision Transformer

- An image can be partitioned into image tokens  
→ e.g. 16x16 pixel **image patches**
- Assign each image patch to an embedding vector, e.g. by linear map
- Apply BERT encoder blocks create contextual embeddings
- Pre-training:
  - Predict masked tokens
  - or predict ImageNet classes directly
- Model the relation between image tokens
  - can fill in missing tokens
  - solve additional tasks by fine-tuning
- Vision Transformer** [\[Dosovitskiy et al. 2020\]](#)



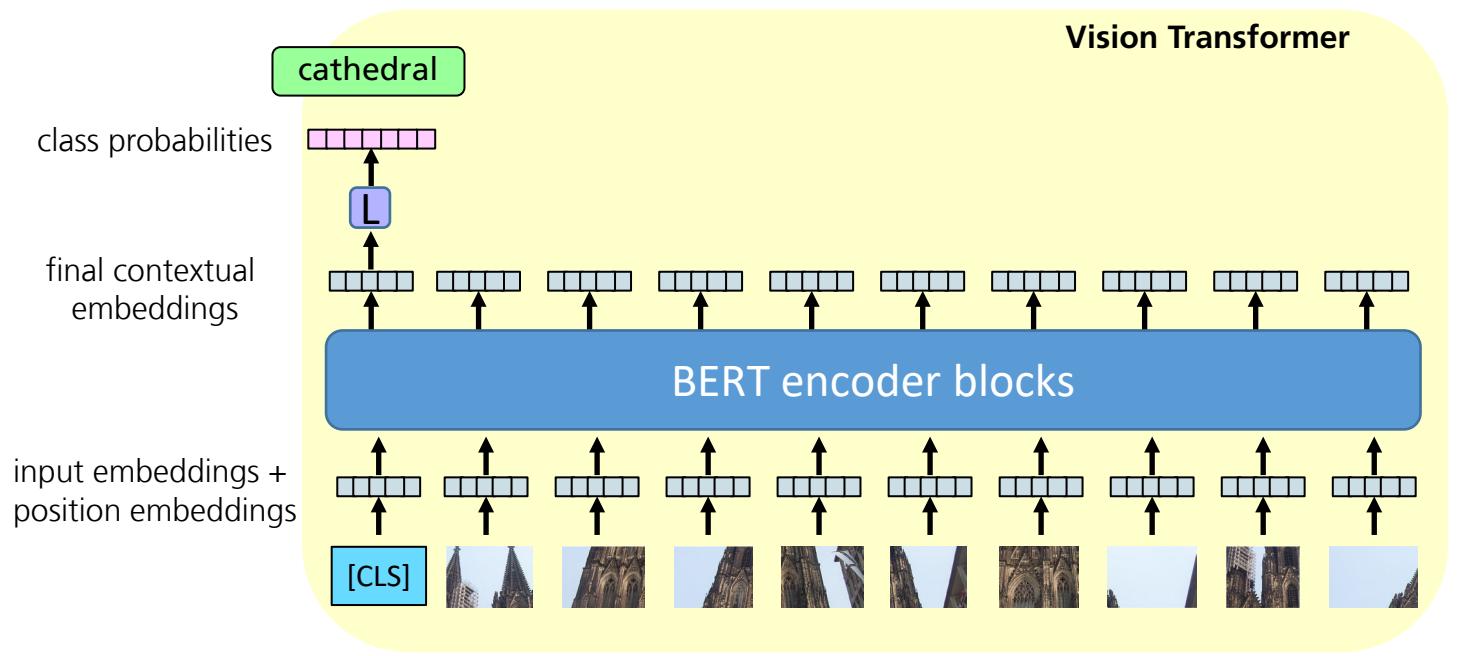
# Vision Transformer Input Encoding

- Generate input embeddings
  - Flatten image patches to vectors and transform linearly  
 $Ax + b$
- Alternative input embeddings
  - autoencoder for image patches
  - first local layers of CNN, e.g. ResNet 152

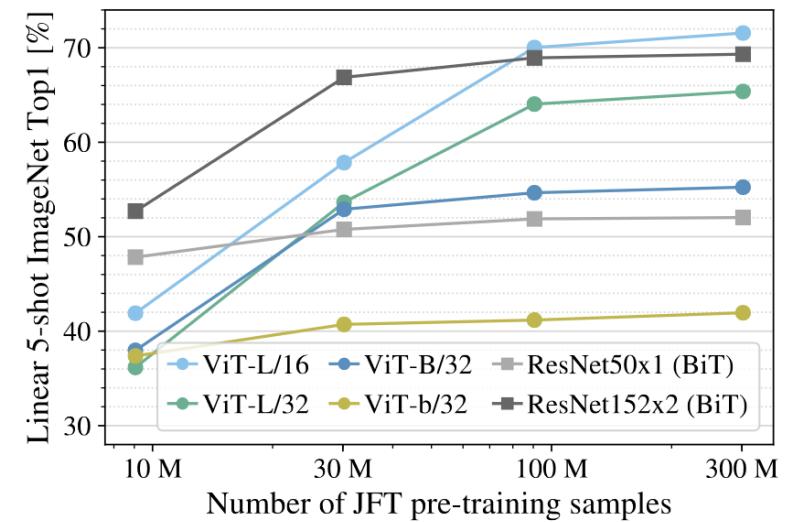


# ImageNet Classification

- Pre-train vision transformer on a large unlabeled image dataset
- weight decay dropout
- Fine-tune on ImageNet classification task with 1000 object classes
- Size of pre-training data matters
  - 10 M images: 42% top-1 acc.
  - 300M images: 73% top 1 acc. **SOTA**
  - model needs more data to acquire knowledge about images



[Dosovitskiy et al. 2020]

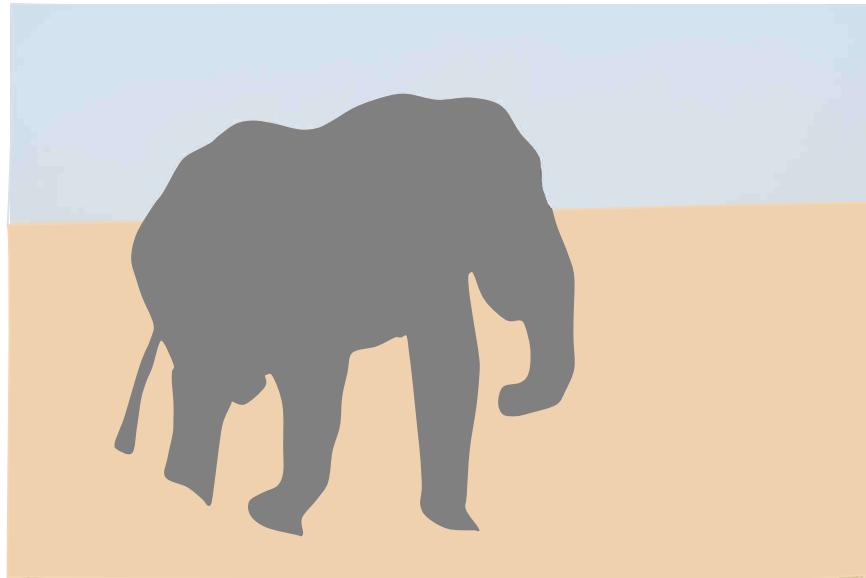
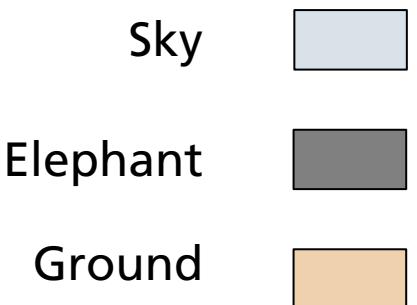


# Image Recognition

1. Image Recognition Tasks
2. Convolutional Layers as local Feature Detectors
3. Simple Convolutional Neural Network
4. Advanced Convolutional Neural Networks
5. Vision Transformer
6. Semantic Segmentation
7. Summary

# Semantic Segmentation

- Identify the pixels belonging to an object.
- Need to classify the objects of the image.
  - Integrate information in the image to a high level
  - propagate the classification information to each pixel.



# Downsampling and Upsampling

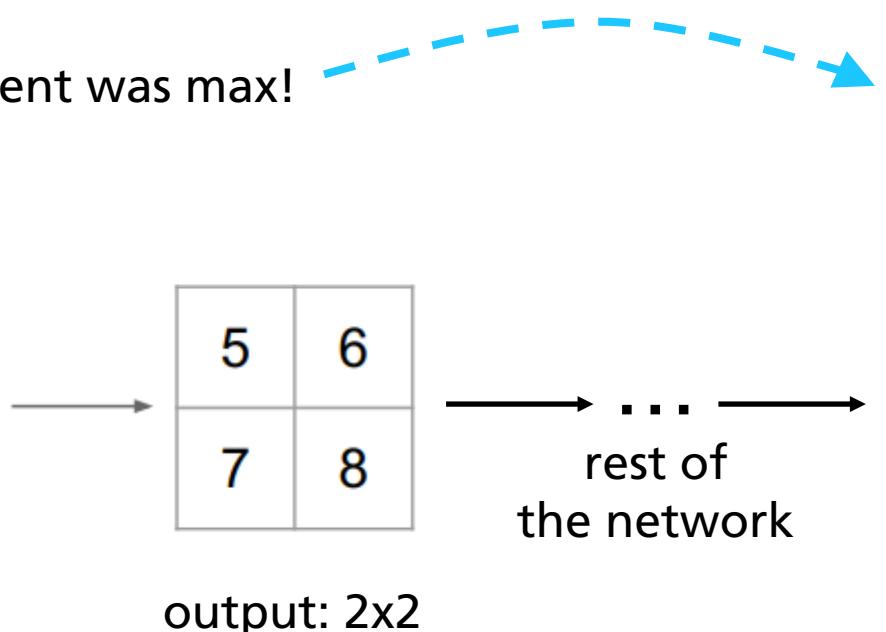
- Max-Pooling performs downsampling  
→ reduce spatial resolution
- Create an inverse upsampling  
→ increase spatial resolution

## Max Pooling

remember which element was max!

1	2	6	3
3	5	2	1
1	2	2	1
7	3	4	8

input: 4x4



## Max Unpooling

use max-position from pooling layer

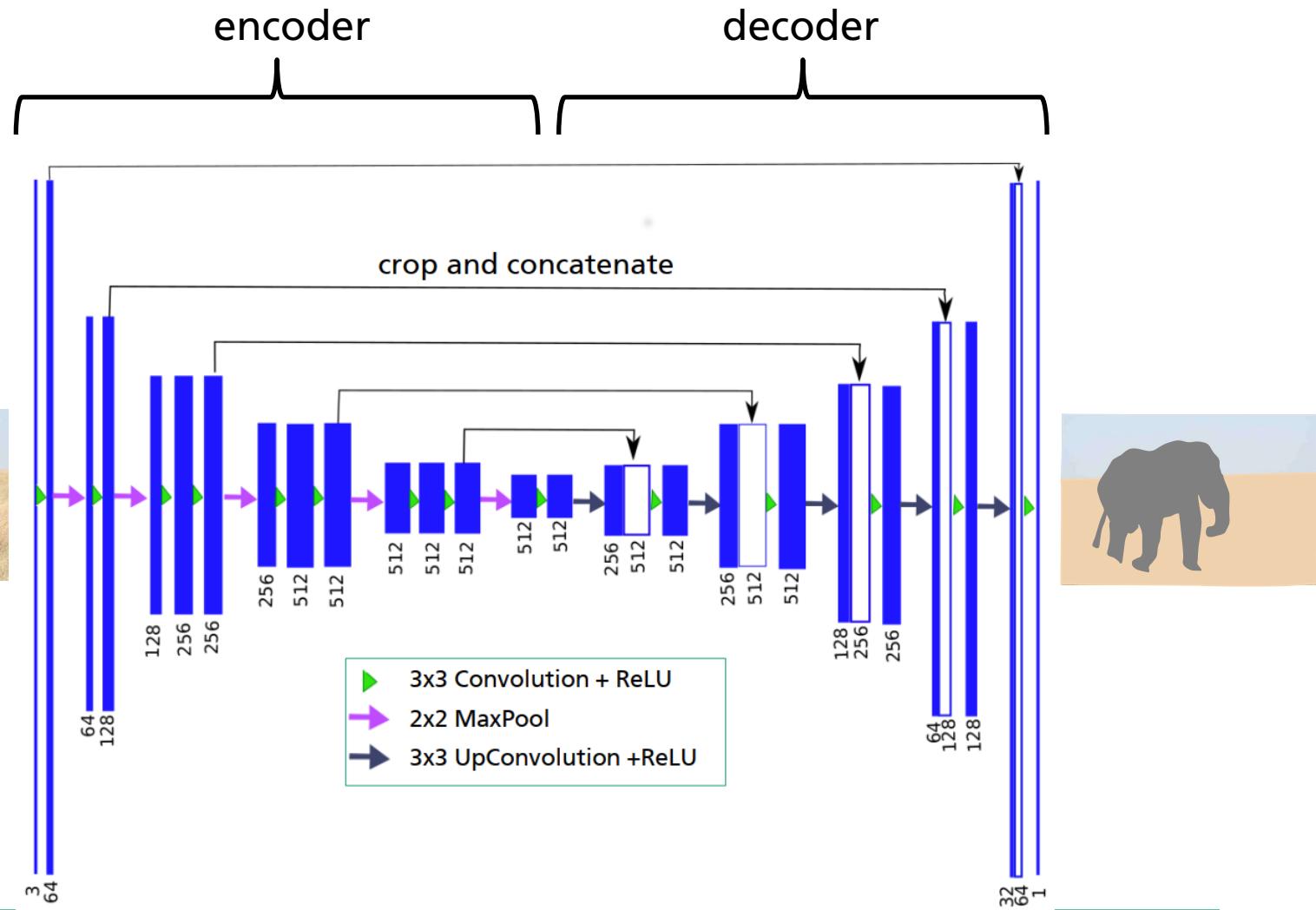
0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

# U-Net

- Reduces spatial resolution with maxpool
  - remember max-positions
- Distribute class information to positions by max-unpooling
- Training:  
get original image and segmented image



"Elephant" by gudi&cris / CC BY 2.0

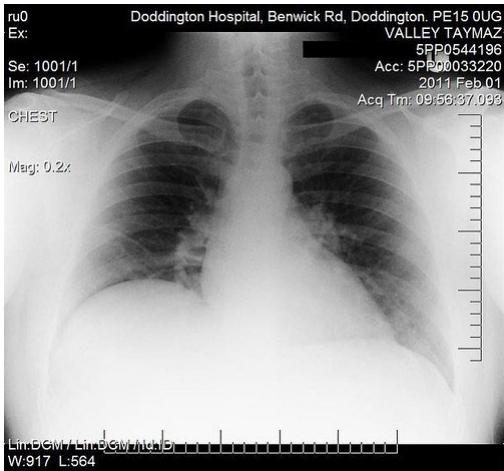


Ronneberger et al. 2015: U-Net: Convolutional Networks  
for Biomedical Image Segmentation 

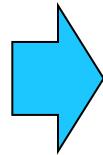
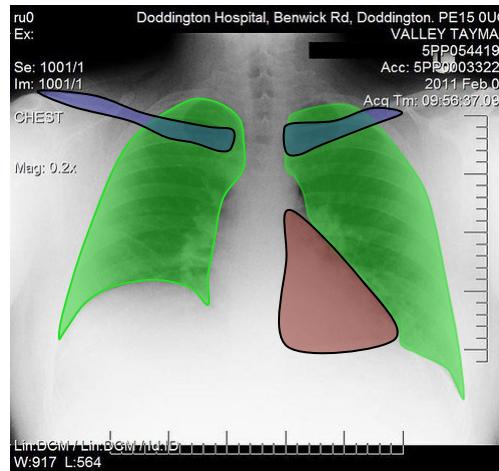
# Semantic Segmentation: Application

- medical image processing

Input image



Segmented image

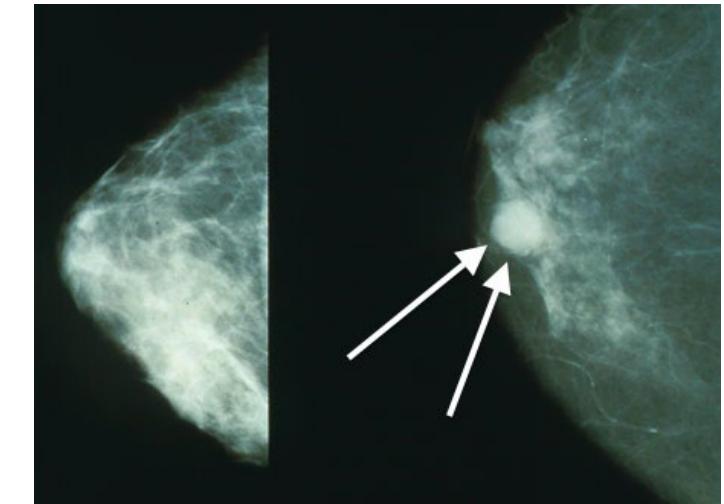


# Application: Cancer Detection

- Skin cancer detection
  - Esteva et al. 2017 [🔗](#)
  - Deep neural network was pretrained on ImageNet data
  - Deep neural network is as good as trained dermatologists
- Brain tumor detection (with U-Net) [🔗](#) p.9

- Breast cancer detection
  - McKinney et al. 2020 [🔗](#)
  - Reduction in false negatives and false positives
  - the AI system outperformed all six human experts: the area under the receiver operating characteristic curve (AUC-ROC) for the AI system was greater than the AUC-ROC for the average radiologist by an absolute margin of 11.5%.

Upper row and lower left: public domain [https://commons.wikimedia.org/wiki/File:Firm\\_red\\_skin\\_lump.jpg](https://commons.wikimedia.org/wiki/File:Firm_red_skin_lump.jpg), [https://commons.wikimedia.org/w/index.php?title=Special:Search&ilimit=50&offset=0&profile=default&search=skin+cancer&advancedSearchCurrent={}&ns0=1&ns6=1&ns12=1&ns14=1&ns100=1&ns106=1#media/File:Melanoma\\_with\\_color\\_differences.jpg](https://commons.wikimedia.org/w/index.php?title=Special:Search&ilimit=50&offset=0&profile=default&search=skin+cancer&advancedSearchCurrent={}&ns0=1&ns6=1&ns12=1&ns14=1&ns100=1&ns106=1#media/File:Melanoma_with_color_differences.jpg), [https://en.wikipedia.org/wiki/Melanoma#/media/File:Spitz\\_nevus.jpg](https://en.wikipedia.org/wiki/Melanoma#/media/File:Spitz_nevus.jpg)  
Lower row middle: Jmarchn [https://en.wikipedia.org/wiki/Nevus#/media/File:Becker's\\_nevus\\_of\\_shoulder.JPG](https://en.wikipedia.org/wiki/Nevus#/media/File:Becker's_nevus_of_shoulder.JPG) by [Jmarchn / CC BY-SA 3.0](#).  
lower row right: Mathew Bellemare [https://commons.wikimedia.org/wiki/File:51\\_105\\_Leg\\_Nevus\\_\(149029661\).jpeg](https://commons.wikimedia.org/wiki/File:51_105_Leg_Nevus_(149029661).jpeg) / [CC BY-SA 3.0](#)



Modified version of File:Mammo\_breast\_cancer.jpg with arrows pointing to breast Ca by Bakerstmd / CC BY-SA 4.0

# Semantic Segmentation: Application

## ■ self-driving cars



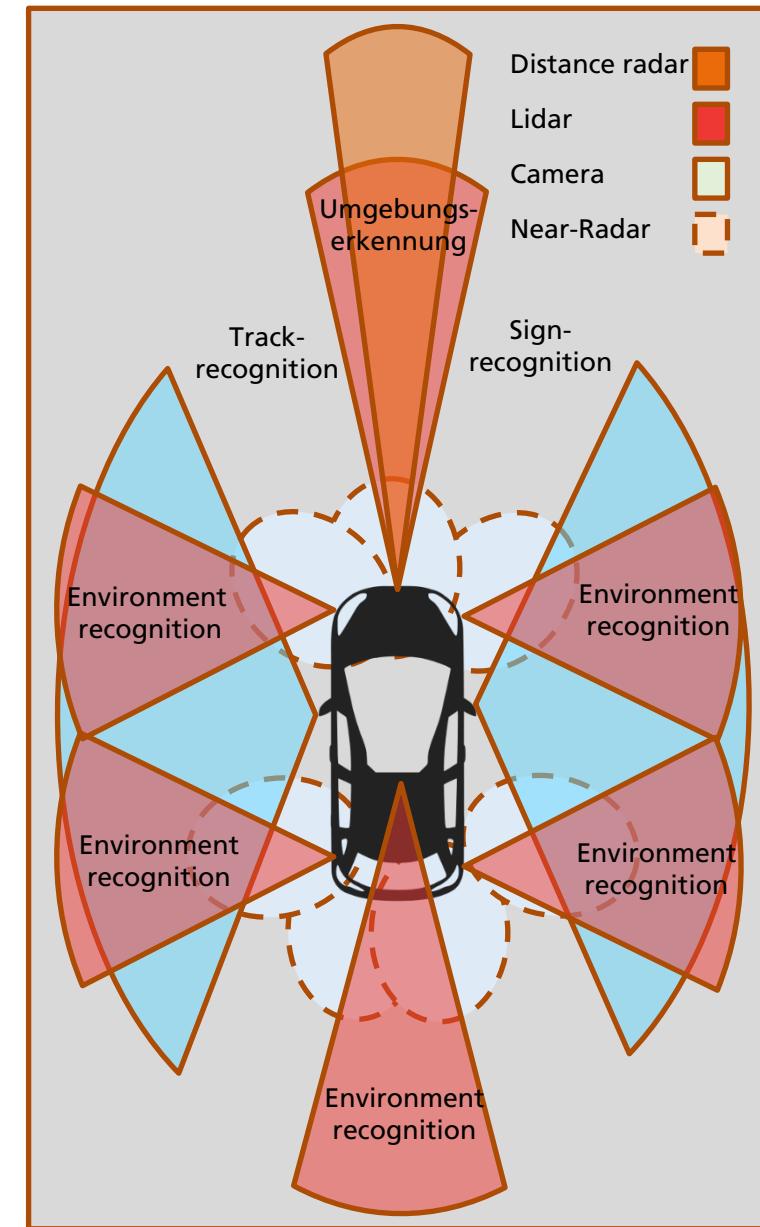
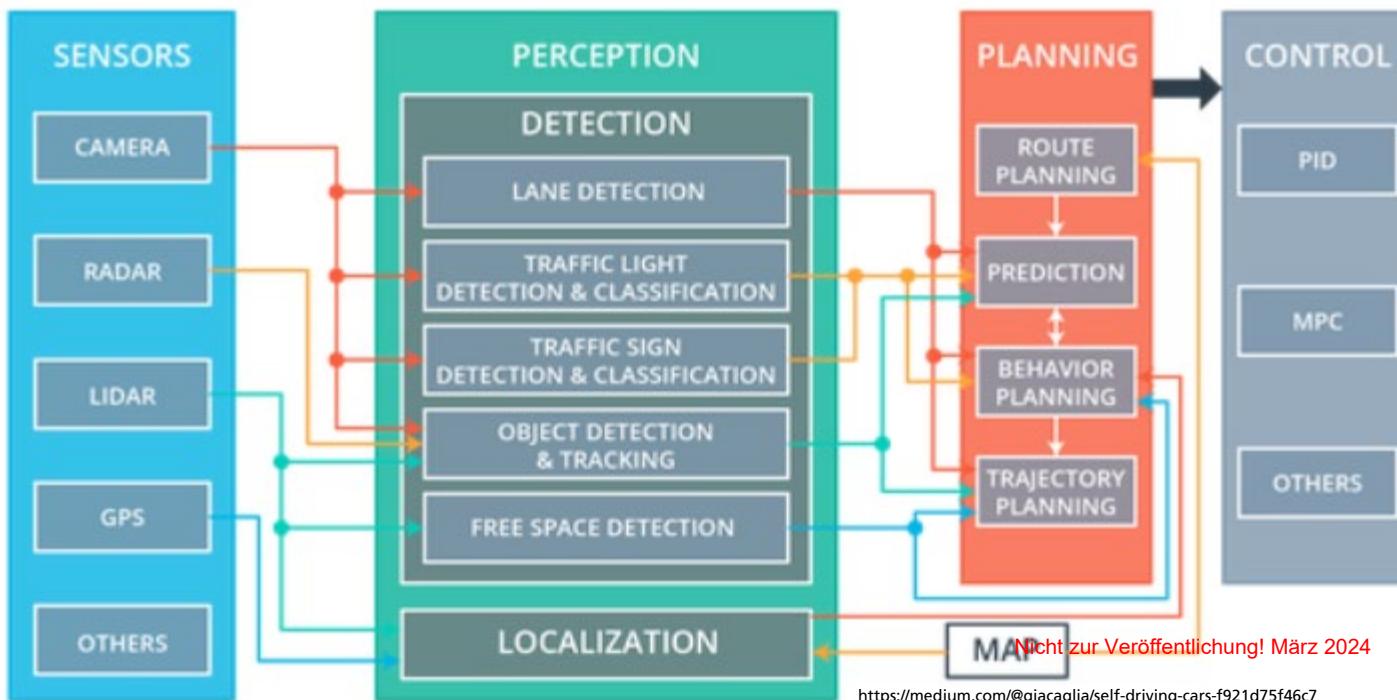
© scharfsinn86 - stock.adobe.com

Video of segmented traffic

benchmark: Cityscapes test (<https://paperswithcode.com/sota/semantic-segmentation-on-cityscapes>)

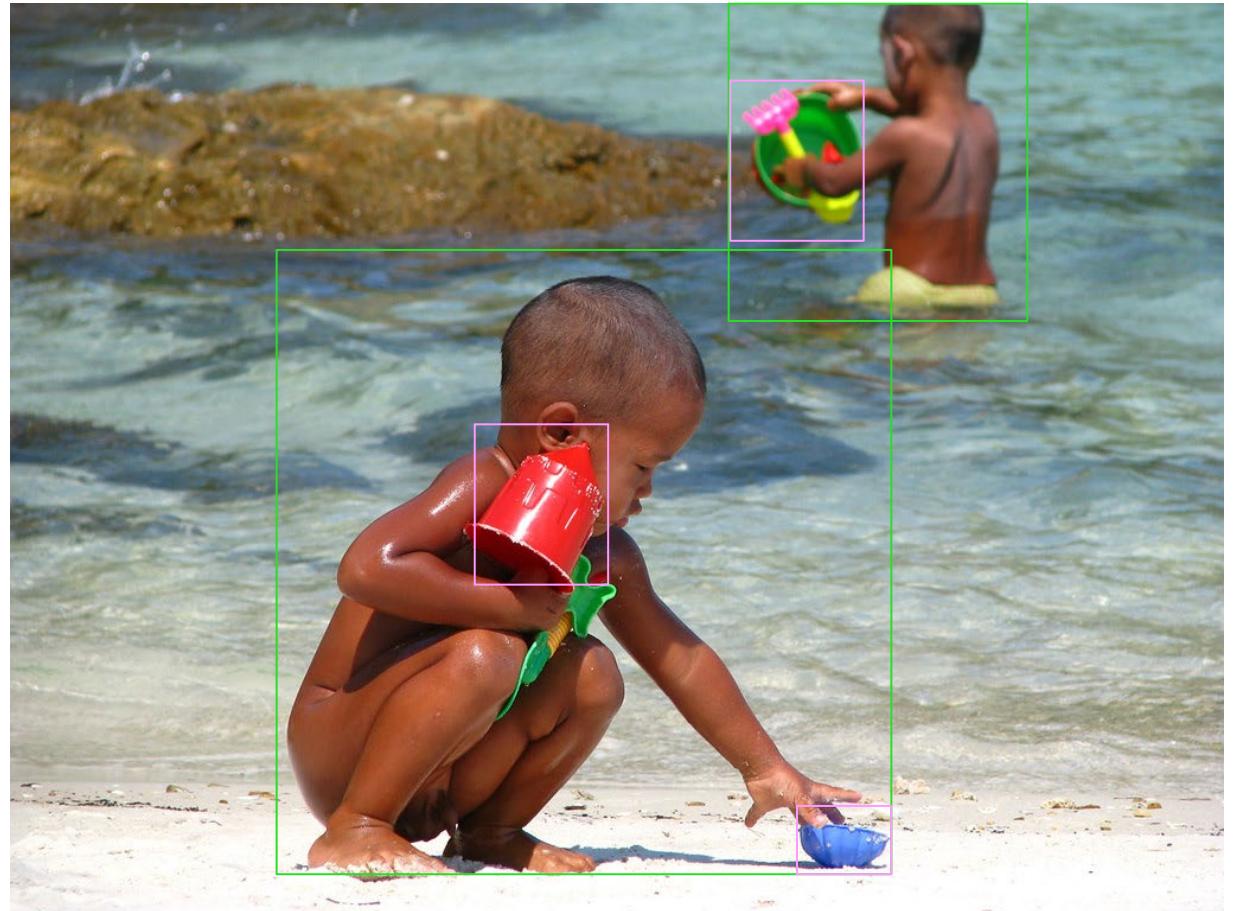
# Application: Self Driving Cars

- Integration of many sensors
  - Camera: important for lane, sign & object detection
  - Radar
  - Lidar: precision distance measuring
  - GPS for location detection
  - Movement & speed, acceleration



# Object detection and Captioning

- detect objects in the image
- assign properties
- find relations
- Training with image-caption pairs



Caption: Child playing on the beach and child playing in the water

Johnson et al. 2016: DenseCap: Fully Convolutional Localization Networks for Dense Captioning. 5th page

# Image Recognition

1. Image Recognition Tasks
2. Convolutional Layers as local Feature Detectors
3. Simple Convolutional Neural Network
4. Advanced Convolutional Neural Networks
5. Vision Transformer
6. Semantic Segmentation
7. Summary

# Summary

## ■ Convolutional Networks

- Consist from Convolutions: kernels which are shifted over the input
- A Convolutional layer contains many kernels → different features
- Maxpooling aggregates and reduces spatial resolution
- Regularization: Dropout / Batch normalization
- Modern architectures
  - inception module: use parallel branches with different kernel sizes
  - ResNet: use residual connections as shortcut. Allows many layers. Good standard architecture
  - **Vision Transformer**: Use BERT-like architecture to classify images  
→ give better results but requires more unlabeled training data
- Semantic segmentation
  - reduce spatial resolution and subsequently increase spatial resolution
  - Max Pooling and Max Unpooling

# Disclaimer

Copyright © by  
Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.  
Hansastraße 27 c, 80686 Munich, Germany

All rights reserved.

**Responsible contact: Dr. Gerhard Paaß, Fraunhofer IAIS, Sankt Augustin**  
E-mail: [gerhard.paass@iais.fraunhofer.de](mailto:gerhard.paass@iais.fraunhofer.de)

All copyrights for this presentation and their content are owned in full by the Fraunhofer-Gesellschaft, unless expressly indicated otherwise.

Each presentation may be used for personal editorial purposes only. Modifications of images and text are not permitted. Any download or printed copy of this presentation material shall not be distributed or used for commercial purposes without prior consent of the Fraunhofer-Gesellschaft.

Notwithstanding the above mentioned, the presentation may only be used for reporting on Fraunhofer-Gesellschaft and its institutes free of charge provided source references to Fraunhofer's copyright shall be included correctly and provided that two free copies of the publication shall be sent to the above mentioned address.

The Fraunhofer-Gesellschaft undertakes reasonable efforts to ensure that the contents of its presentations are accurate, complete and kept up to date. Nevertheless, the possibility of errors cannot be entirely ruled out. The Fraunhofer-Gesellschaft does not take any warranty in respect of the timeliness, accuracy or completeness of material published in its presentations, and disclaims all liability for (material or non-material) loss or damage arising from the use of content obtained from the presentations. The afore mentioned disclaimer includes damages of third parties.

Registered trademarks, names, and copyrighted text and images are not generally indicated as such in the presentations of the Fraunhofer-Gesellschaft. However, the absence of such indications in no way implies that these names, images or text belong to the public domain and may be used unrestrictedly with regard to trademark or copyright law.