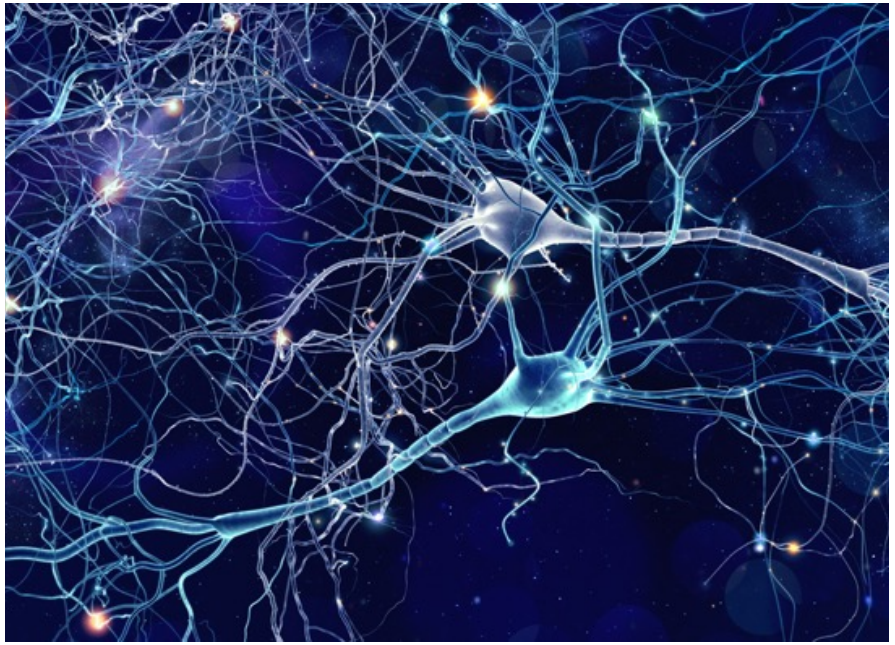
MSE | MASTER OF SCIENCE IN ENGINEERING

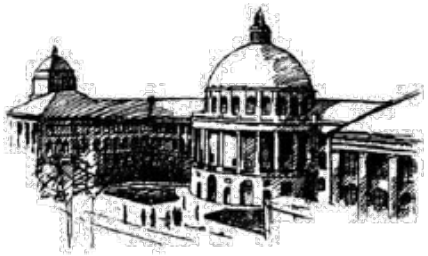
Image Classification and Network Architectures



Computer Science

Prof. Dr. Thomas Koller

Short CV: Thomas Koller



Dipl. Informatik Ing. ETH
- Computer Graphics at IPS (Supercomputing Centre ETH)

PhD ETH Zürich, Computer Vision Lab
- 3D Image Processing, Visualization

Bitplane AG
- Head of Development, 3D Image Processing, Visualization

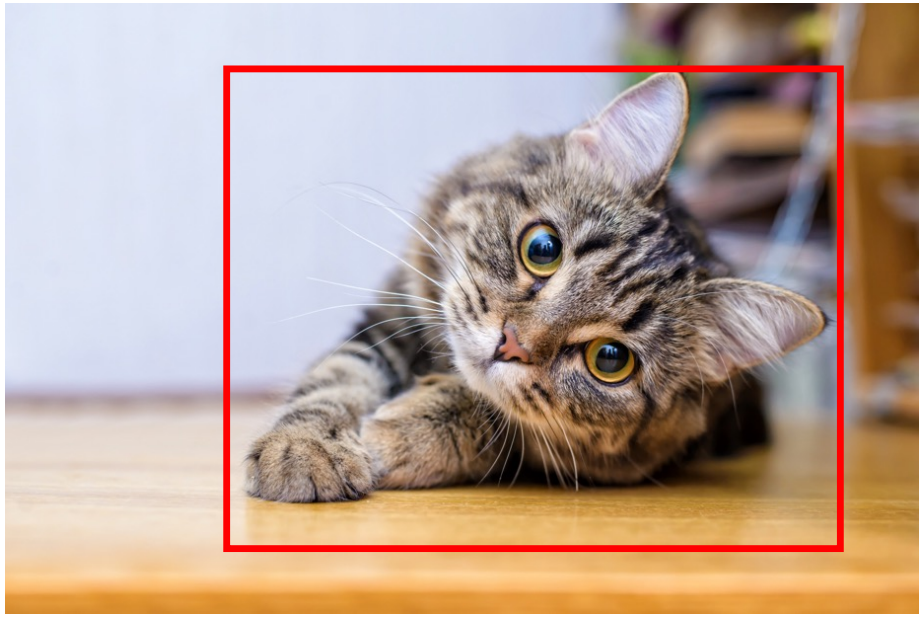
Zühlke Engineering AG:
- Software Engineering Consultant, Project Manager, Architect, Senior Trainer

HSLU:
- Computer Vision, Reinforcement Learning, Deep Learning
- MSE responsible for HSLU Informatik

Problems in Computer Vision

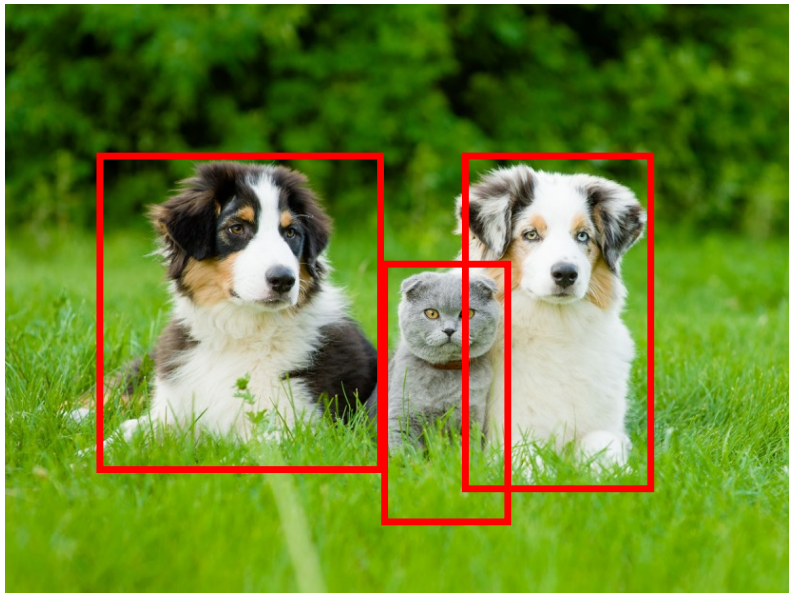
Image Classification

Classification and Localization



Problems in Computer Vision

Object Detection and Classification



Semantic Segmentation

Agenda

Image Classification:

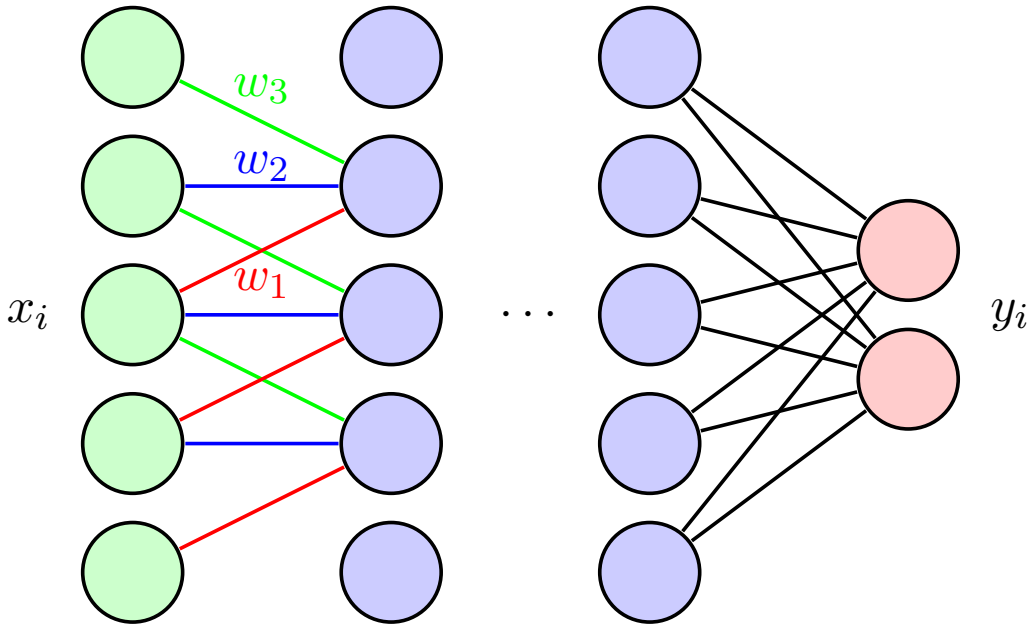
- Recap of convolutional neural networks
- Which concepts and architectures have been proven successful for accurate image classification and related tasks?
- How do the best networks look like?
- How to train larger CNNs?

Recap: Convolutional Neural Networks

Original idea from image processing, where convolution (filters) are applied for example for feature selection.

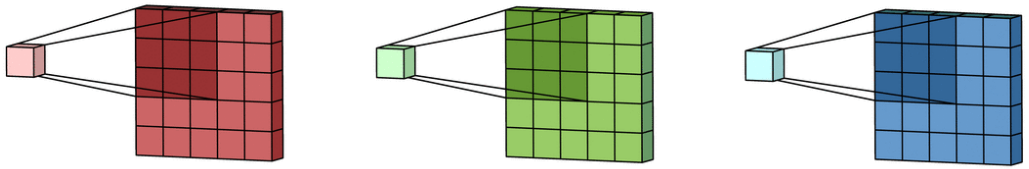
Properties:

- Weights (parameters) are **shared**
- Connections are **sparse**
- Each node in a layer has a receptive field

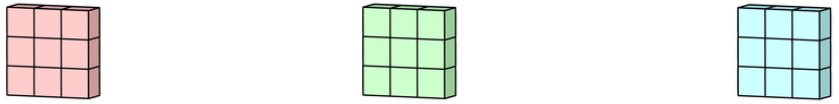


CNNs: Multiple channels into 1 output channel

- When the input to the layer has multiple channels, a (different) 2D Kernel is applied to each channel



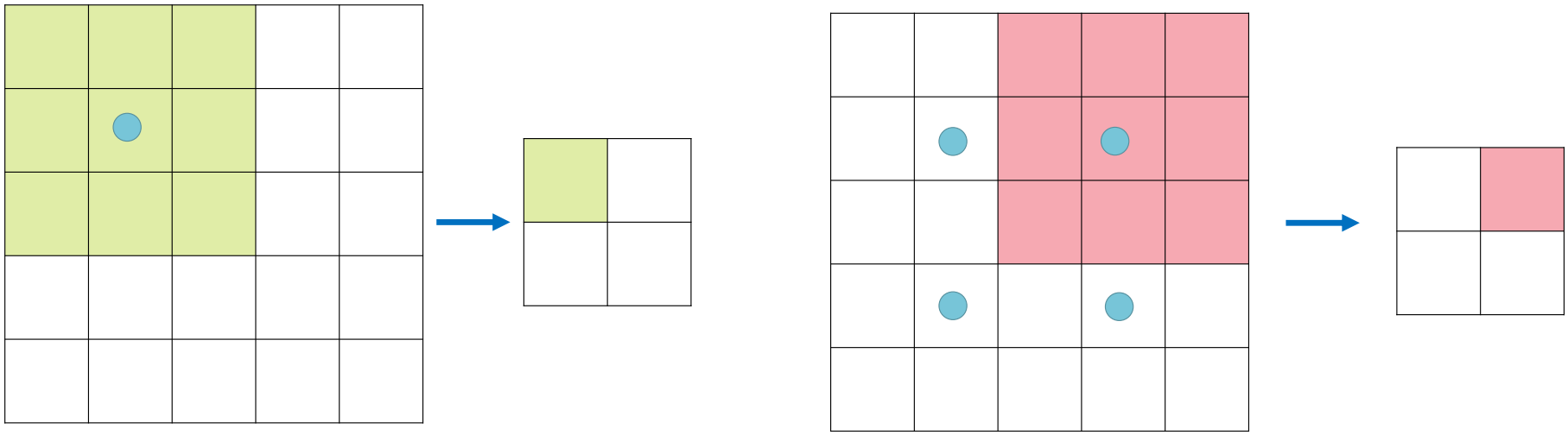
- The resulting channels are added (+bias)



https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

Decrease Image Size: Strided Convolutions

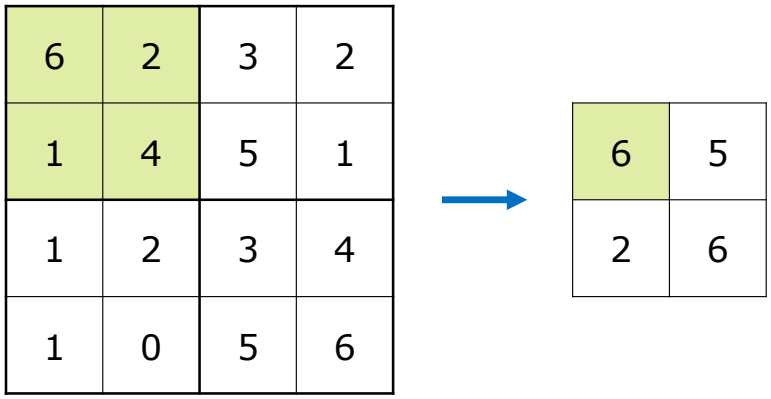
Strided 3x3 convolution with stride = 2



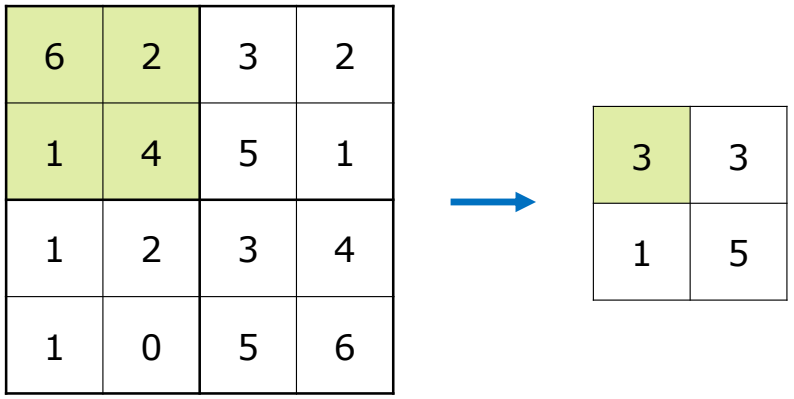
Filter moves 2 positions in **input** for every position in **output**

Decrease Image Size: Pooling

Max Pooling

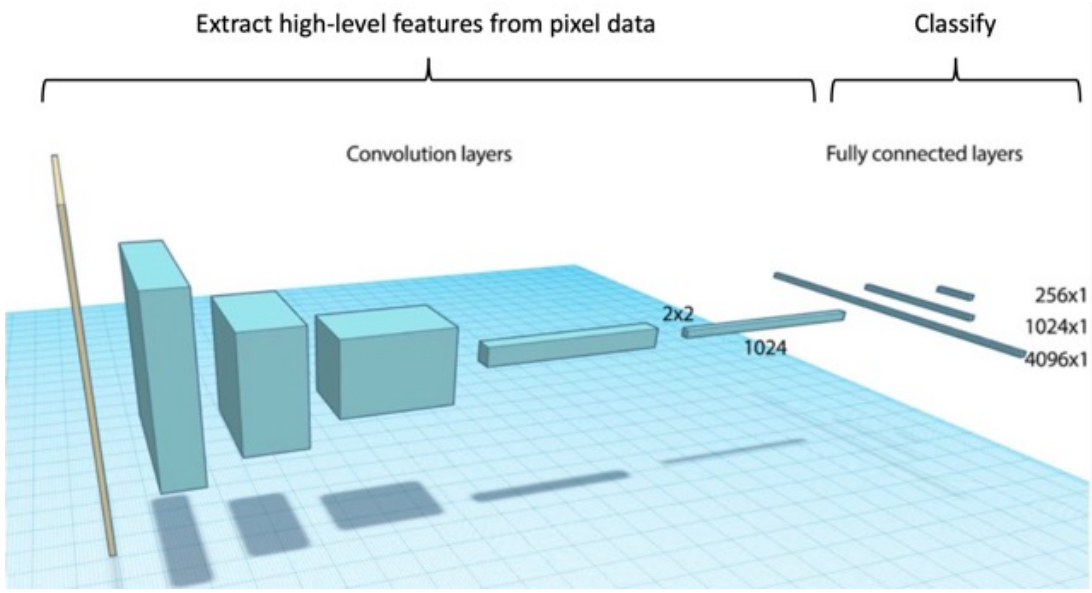


Average Pooling



Typical Architecture for CNNs for Image Classification

- Reduce spatial resolution (Images get smaller)
- Increase number of channels (maps)
- Fully connected layers at the end (classification head)



Convolutional Layers

```
keras.layers.Conv2D(
        filters, kernel_size,
        strides=(1, 1),
        padding="valid",
        data_format=None, dilation_rate=(1, 1), groups=1,
        activation=None, use_bias=True,
        kernel_initializer="glorot_uniform", bias_initializer="zeros",
        kernel_regularizer=None, bias_regularizer=None, ...)
```

- **filters**: int, the dimension of the output space (the number of filters in the convolution).
- **kernel_size**: int or tuple/list of 2 integer, specifying the size of the convolution window.
- **strides**: int or tuple/list of 2 integer, specifying the stride length of the convolution. strides > 1 is incompatible with dilation_rate > 1.
- **padding**: string, either "valid" or "same" (case-insensitive). "valid" means no padding. "same"results in padding evenly to the left/right or up/down of the input. When padding="same" andstrides=1, the output has the same size as the input.

Example

```
def build_model():
    inp = keras.Input(shape=(32, 32, 3))
    x = keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu")(inp)
    x = keras.layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu")(x)
    x = keras.layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu")(x)
    x = keras.layers.Flatten()(x)
    x = keras.layers.Dense(100, activation='softmax')(x)

    return keras.Model(inp, x)
```

Example

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 32, 32, 3)	0
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 32)	9,248
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 32)	0
conv2d_2 (Conv2D)	(None, 4, 4, 32)	9,248
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 100)	51,300

Total params: 70,692 (276.14 KB)

Trainable params: 70,692 (276.14 KB)

Non-trainable params: 0 (0.00 B)

Neural Networks for Image Classification evolved over time

- Do networks just need to have a specific **capacity** for a challenge/problem or does the **architecture** matter?
- Has the architecture changed with more experience on solving different image classification tasks?
- Which are the most important building blocks that enable training deep neural networks?
- How does the architecture change the result?

Image Net Challenge

Data Source:

- 14'197'122 Images organized in WordNet Hierarchy (nouns)
- 21'841 synsets (synonyms)

- For example: 3822 synsets in animals with 732 images per set in average

- Challenge on 1000 classes
- One label per image (original challenge)
- Scoring on Top-1 and Top-5 errors


Herb, herbaceous plant
A plant lacking a permanent woody stem; many are flowering garden plants or potherbs; some having medicinal properties; some are pests
1400 pictures
75.29% Popularity Percentile
Wordnet IDs
pot plant (0)
acrogen (0)
apomict (0)
aquatic (0)
cryptogam (1)
annual (0)
biennial (0)
perennial (1)
escape (0)
hygrophyte (0)
neophyte (0)
embryo (0)
monocarp, monocarpic plant, m
sporophyte (0)
gametophyte (2)
houseplant (12)
garden plant (1)
vascular plant, tracheophyte (4
pteridophyte, nonflowering p
spermatophyte, phanerogam
herb, herbaceous plant (104
barrenwort, bishop's hat,
mayapple, May apple, wilc
buttercup, butterflower, b
goldthread, golden thread
winter aconite, Eranthis h
hepatica, liverleaf (0)
goldenseal, golden seal, y
false rue anemone, false r
giant buttercup, Laccopet
false bugbane, Trautvette
globeflower, globe flower
legume, leguminous plant
Treemap Visualization
Images of the Synset
Downloads
ImageNet 2011 Fall Release
Vascular plant, tracheophyte
Herb, herbaceous plant
Salsify
False
Ayapana
Halogeton
Cottonweed
Egyptian
Prairie
Gromwell
Hedge
Shamrock
Mustang
Lamb
Bugleweed
Tassel
Maranta
Costmary
Columbo
False
Licorice
Rattlesnake
Gesneria
Bloodwort
Horse
Winter
Button
Fringepod
Giant
Yerba
Indian
Winged
Fenugreek
Drypis
Stickweed
Salad
Ginseng
Tansy
Tansy-leaved
Martynia
Sand
Sweet
Sweet
Chamois
Rattlesnake
Chaenactis
Galax
Scopolia
Bladderpod

ImageNet Challenge vs. previous challenges

Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. (* = equal contribution) **ImageNet Large Scale Visual Recognition Challenge**. *IJCV*, 2015.

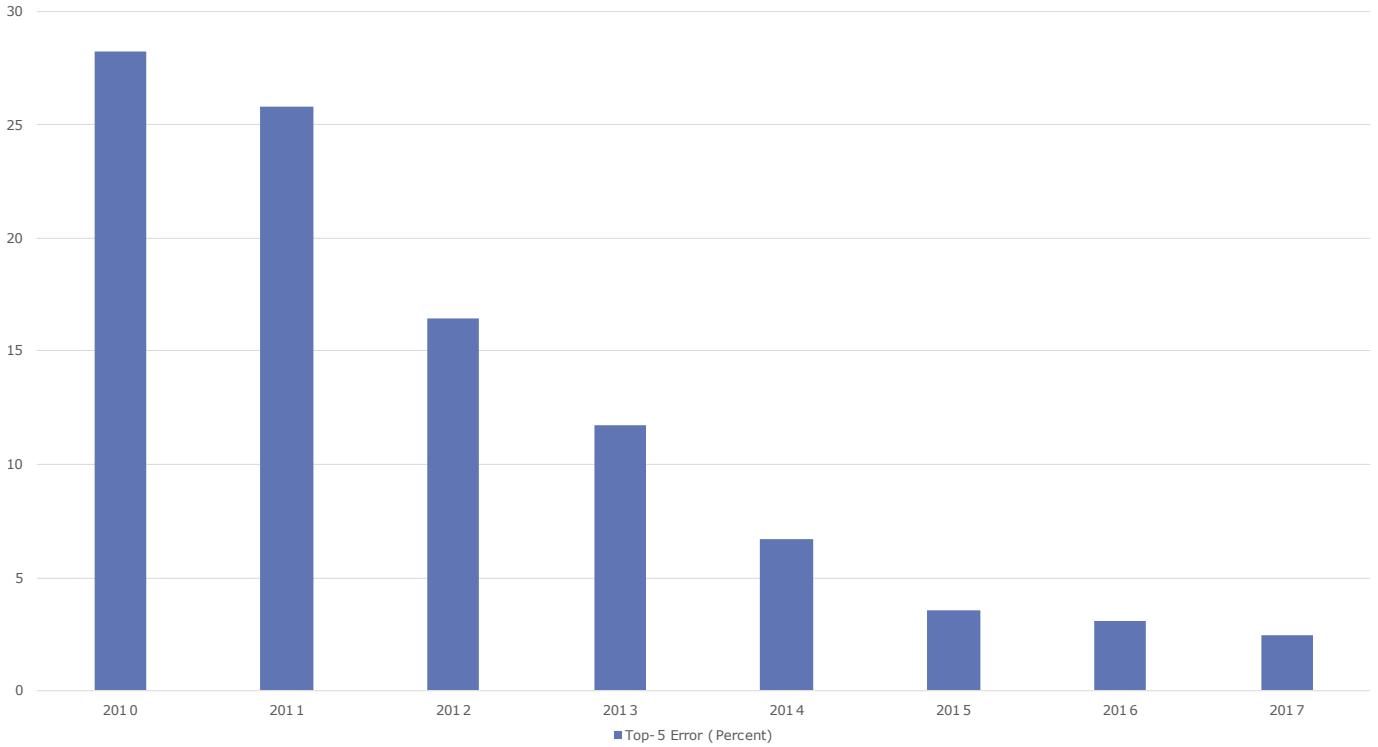
ImageNet Challenge

SVM

Deep Convolutional Neural Networks

2012

Image classification results during challenge



Before Neural Networks: Features & Classifier

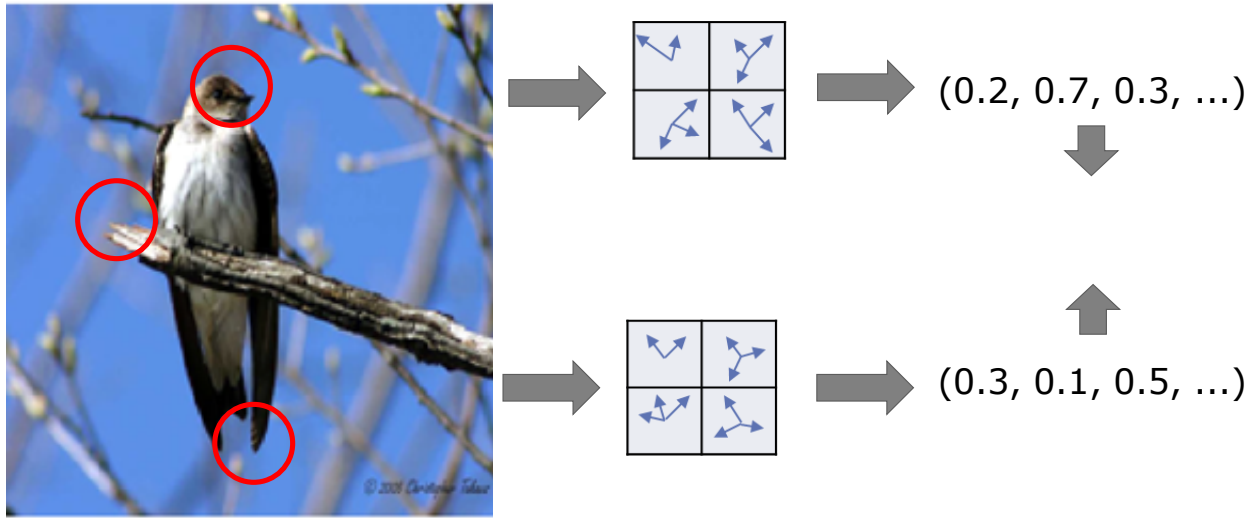
Before CNNs the most successful approaches involved calculating features in the images and then use a classifier

- Often different features were used for different tasks
- Features had to be *hand-crafted* for the task

Support Vector Machines (SVM) were the most popular classifier used

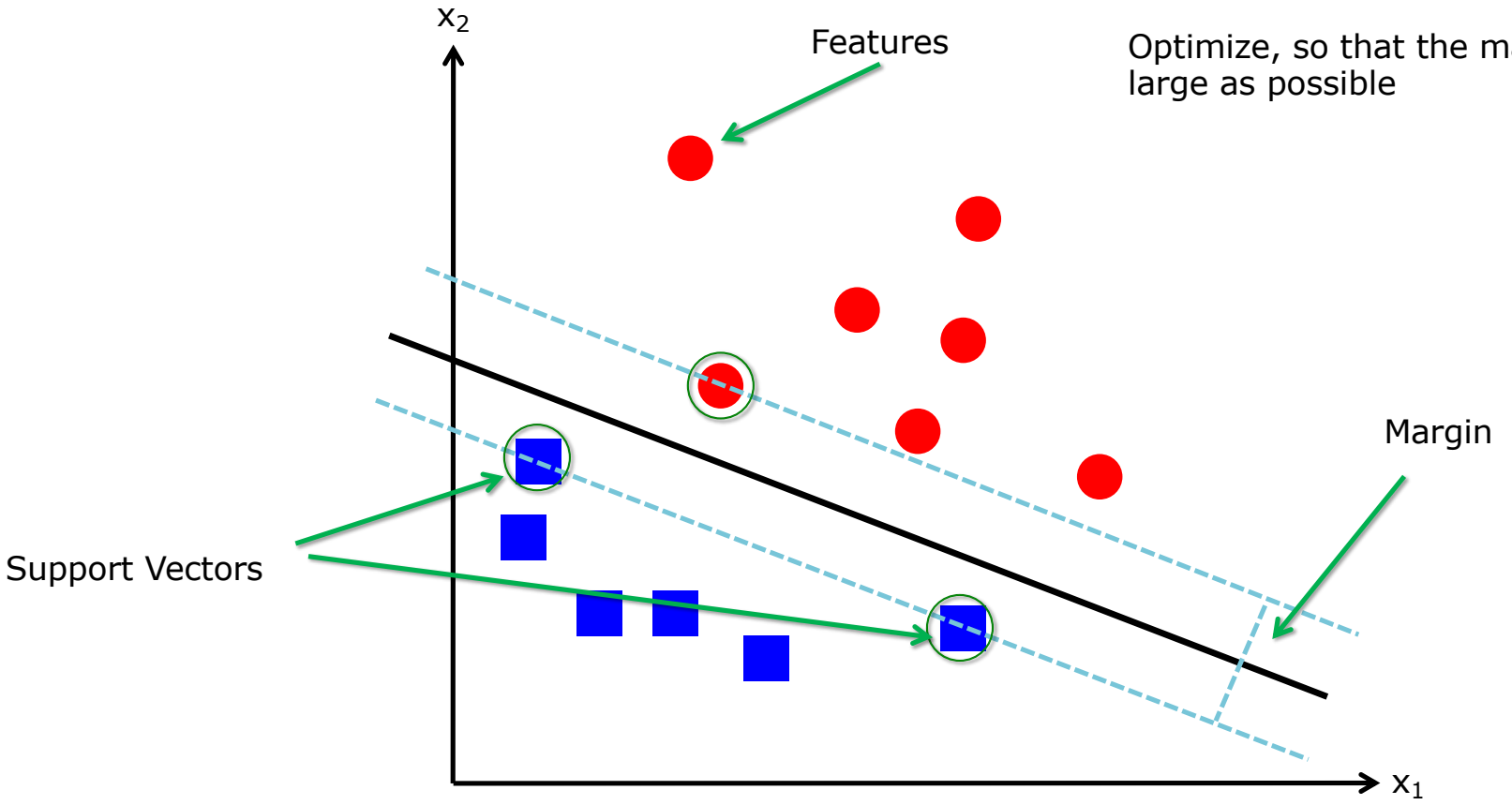
Features

- Features calculate properties of an image or an image region
- Besides classification, they are also used to find subregions or for image stitching etc.
- Feature vector as a robust (e.g., to deformations or illumination) low-dimensional description of an image



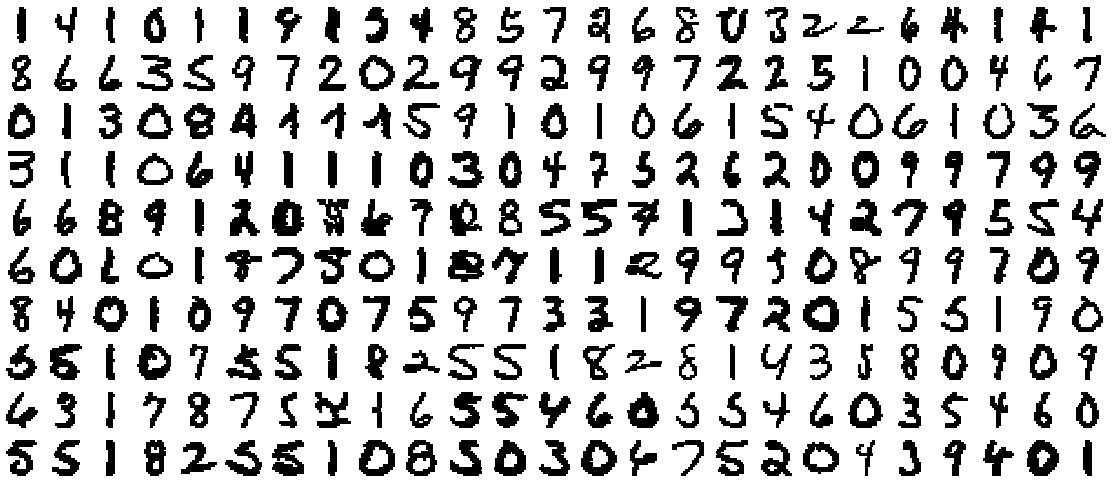
Classification: Support Vector Machines (SVM)

linear classifier



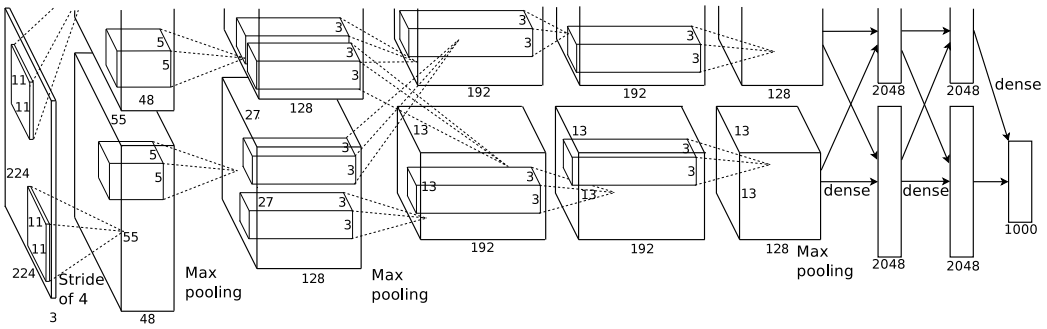
LeNet (1998)

- Handwritten Digit Recognition with a Back-Propagation Network, Le Cun et al.
- First convolutional network
- 2 conv layers, 2 pooling (averaging) layers



AlexNet (2012)

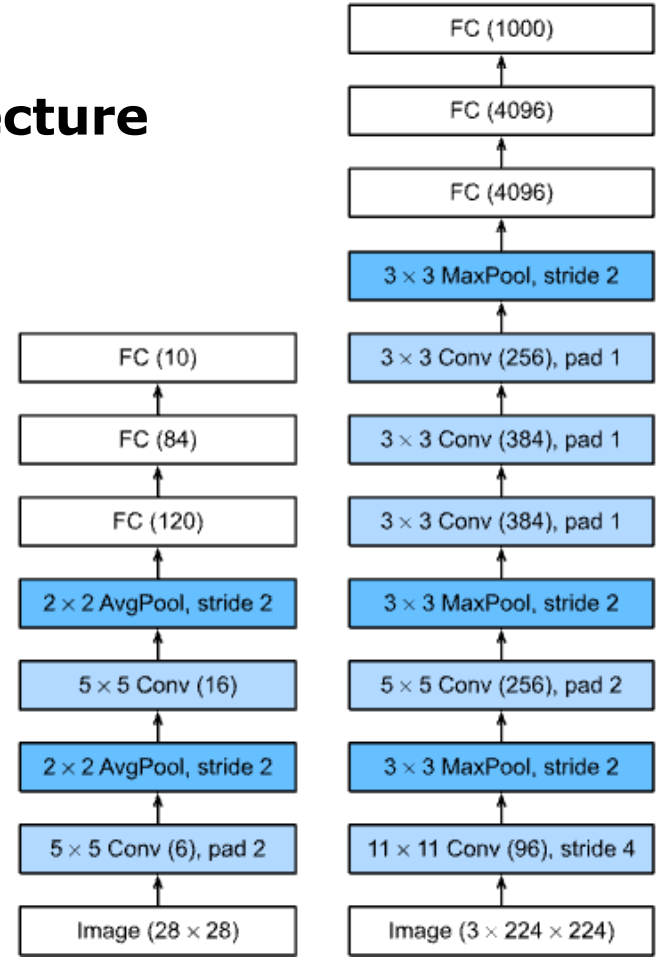
- 5 Convolution Layers,
- Max-Pool Layers
- 3 Fully connected layers at the end
- RELU activation
- 60 million parameters (!)



- Was trained on multiple GTX 580 GPUs with only 3GB Memory

LeNet vs. AlexNet Architecture

LeNet



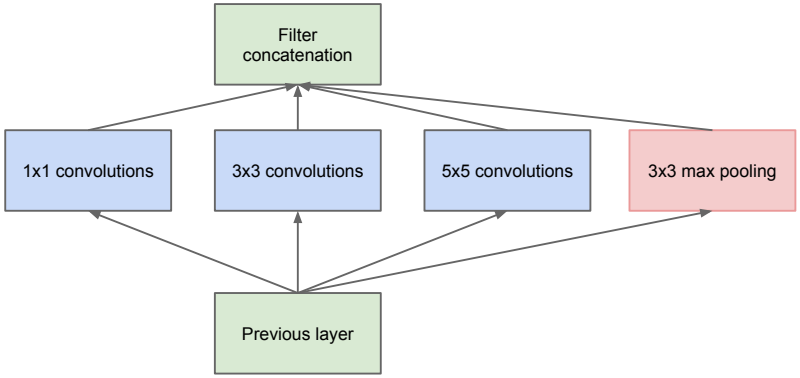
AlexNet

GoogLeNet: Inception Architecture

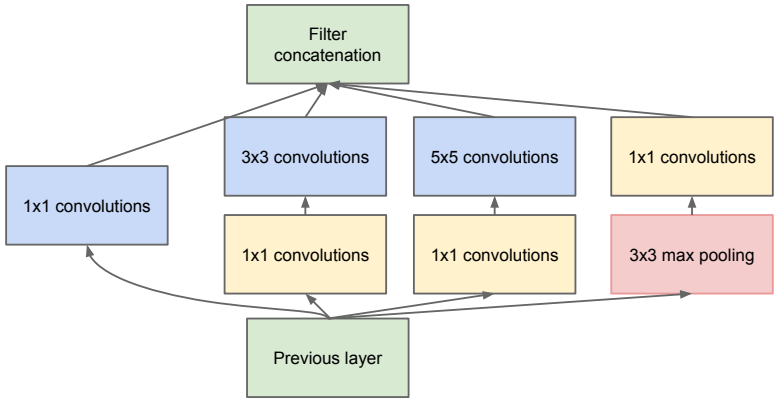
GoogLeNet: Motivation

Larger and deeper nets are necessary to achieve better results

New architecture with filters of various sizes in order to detect features at multiple resolutions



naive version

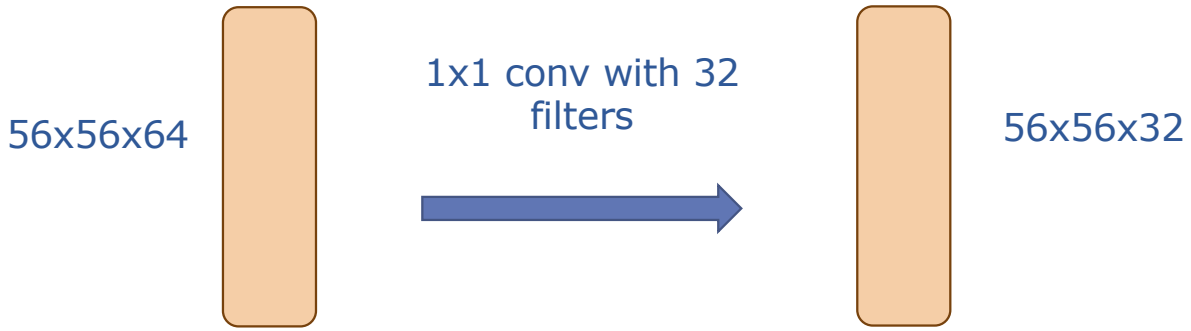


version with dimensionality reduction (bottleneck layers)

Bottleneck Layers: 1x1 Filters

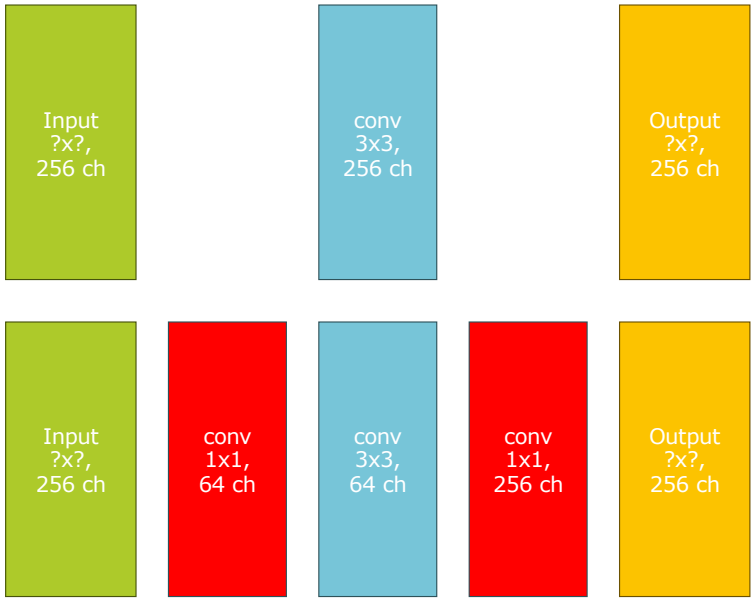
- Filter from 64 channels to 32 channels using only channels (no spatial correlations)
- Preserves spatial dimensions and reduces only depth

- Each filter is a 64-dim dot product



Bottleneck layers before convolutional layers

A bottleneck layer is a 1x1 convolutional layer from a higher number of channels to a lower number (or vice versa) and is often applied before a normal (spatial + depth) convolution.



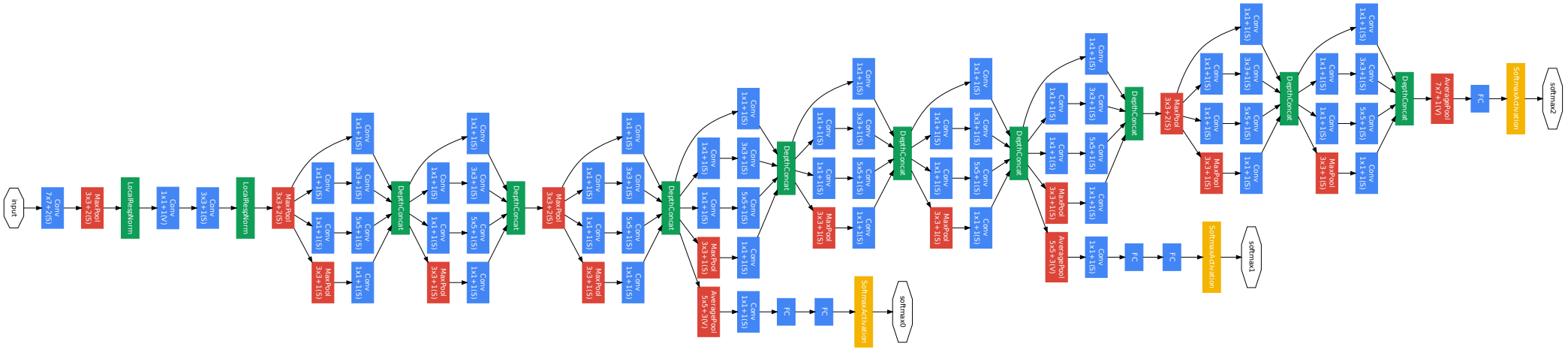
Parameters:

$(3*3*256 + 1) * 256 = 590'080$

$$
\begin{aligned}
(1*1*256 + 1) * 64 &= 16'448 \\
(3*3*64 + 1) * 64 &= 36'928 \\
(1*1*64 + 1) * 256 &= 16'640 \\
\text{Total:} &\quad 70'016
\end{aligned}
$$

GoogLeNet: The Inception Network

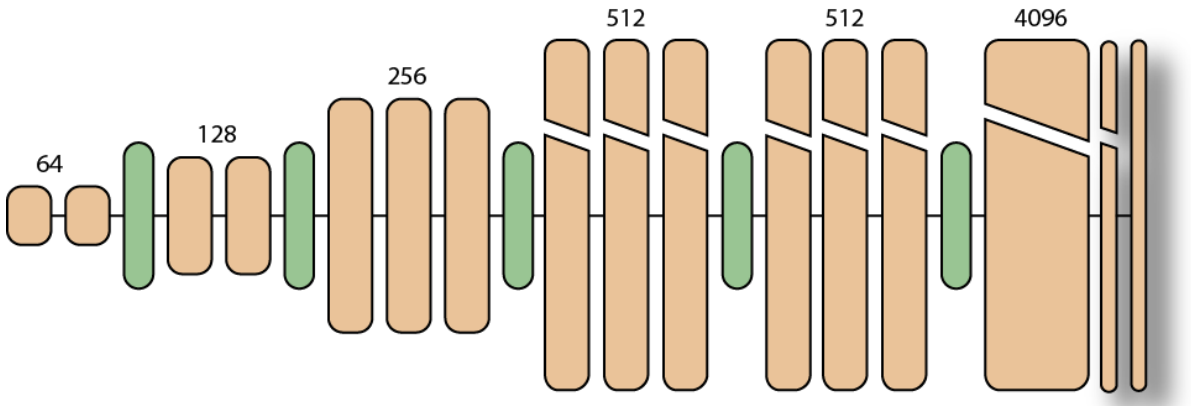
- 22 layers deep (layers with parameters)
- ca. 100 layer building blocks
- Needed to be trained at different depths



VGG Net

- Simonyan & Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*

- Examine effect of depth of networks
- 3x3 (!) convolutional layers only, some with pooling

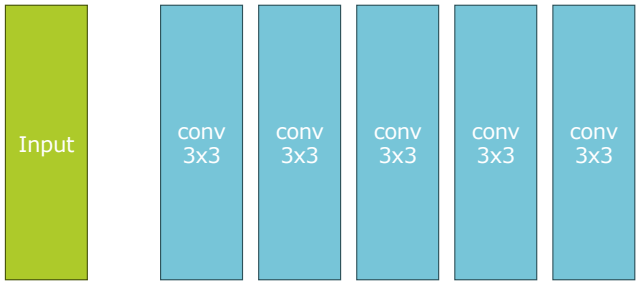


Replace large filter kernels with a sequence of smaller filters

Receptive field:

Parameters:

11 x 11



50

11 x 11

122

Example 3x3 vs 11x11 filter

```
inp = keras.Input(shape=(32, 32, 8))
x = keras.layers.Conv2D(8, kernel_size=(3, 3),...)(inp)
x = keras.layers.Conv2D(8, kernel_size=(3, 3),...)(x)
x = keras.layers.Conv2D(8, kernel_size=(3, 3),...)(x)
x = keras.layers.Conv2D(8, kernel_size=(3, 3),...)(x)
x = keras.layers.Conv2D(8, kernel_size=(3, 3),...)(x)
m3 = keras.Model(inp, x)
m3.summary()
```

Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 32, 32, 8)	0
conv2d_9 (Conv2D)	(None, 32, 32, 8)	584
conv2d_10 (Conv2D)	(None, 32, 32, 8)	584
conv2d_11 (Conv2D)	(None, 32, 32, 8)	584
conv2d_12 (Conv2D)	(None, 32, 32, 8)	584
conv2d_13 (Conv2D)	(None, 32, 32, 8)	584

Total params: 2,920 (11.41 KB)

```
inp = keras.Input(shape=(32, 32, 8))
x = keras.layers.Conv2D(8, kernel_size=(11, 11),...)(inp)
m11 = keras.Model(inp, x)
m11.summary()
```

Layer (type)	Output Shape	Param #
input_layer_4 (InputLayer)	(None, 32, 32, 8)	0
conv2d_14 (Conv2D)	(None, 32, 32, 8)	7,752

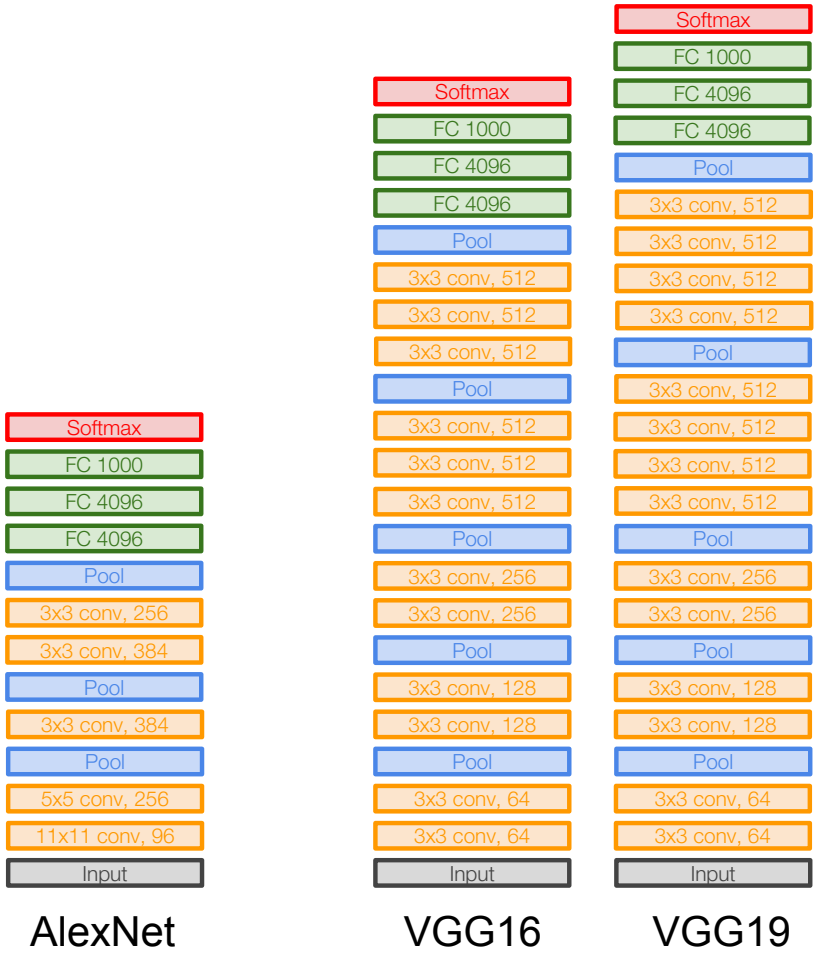
Total params: 7,752 (30.28 KB)

Trainable params: 7,752 (30.28 KB)

VGG Nets

Why use small filters?

- Stack of 3 3x3 filters has the same receptive e field as a 7x7 filters
- Less parameters
- More non-linearity
- 27 vs 49 weights (not counting biases) per channel

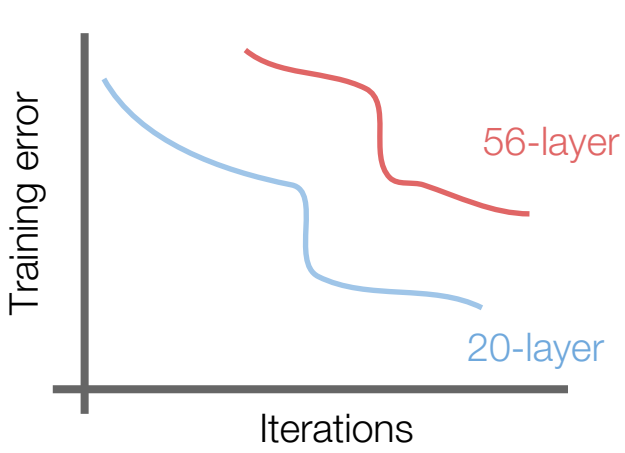


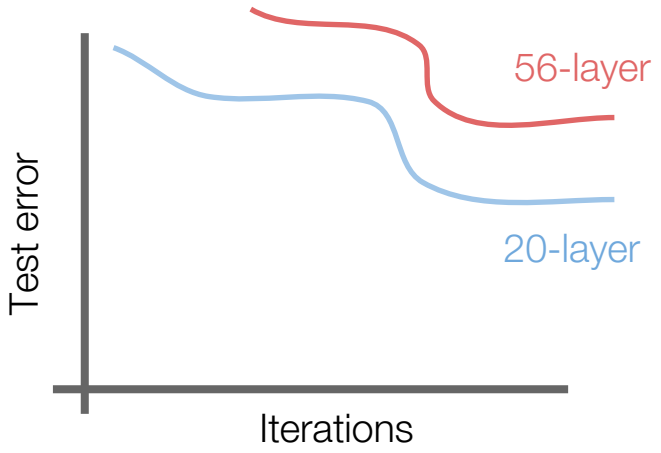
VGG Net 16

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=**3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=**3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=**1.6M** params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=**1.6M** params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

Residual Networks: ResNet

- What happens when we add layers to a convolutional network?
- Experiment: 56 layer model is worse than 20 layers, but not from overfitting
- How can this happen?

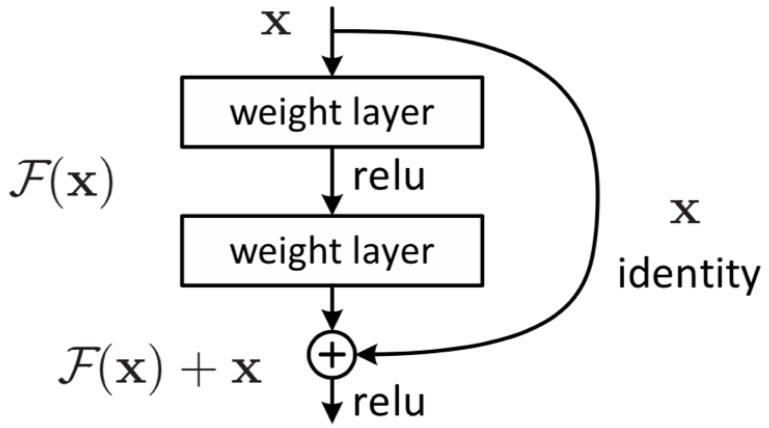




Residual Networks

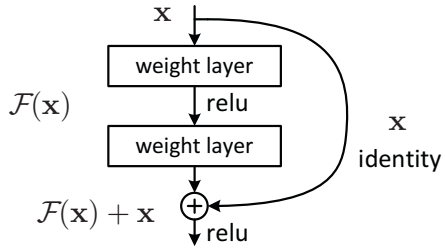
- Deeper Networks: If there are enough layers, adding more layers should not decrease performance
- The performance stays (at least) the same, if the new layers would do nothing...
- However, layers do not learn identity transform easily

Solution: Residual connections

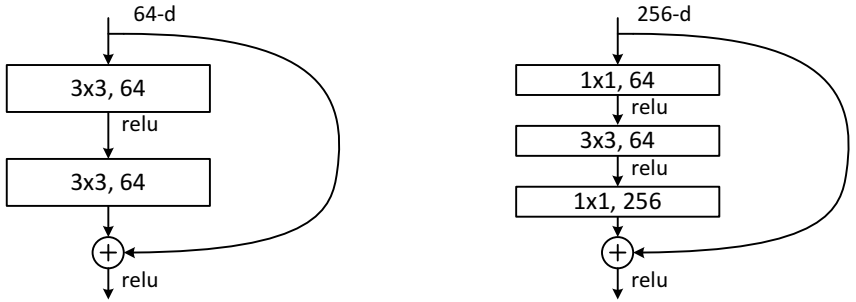


Residual connection

- Residual or skip connections **add** the **input** of a block of layers to its **output**
- I.e. the block only had to learn the **residual**: the difference to its input
- This helps to propagate the **gradient** backwards, as layers are initialized close to zero
- For deeper networks, the connections can be combined with bottleneck layers

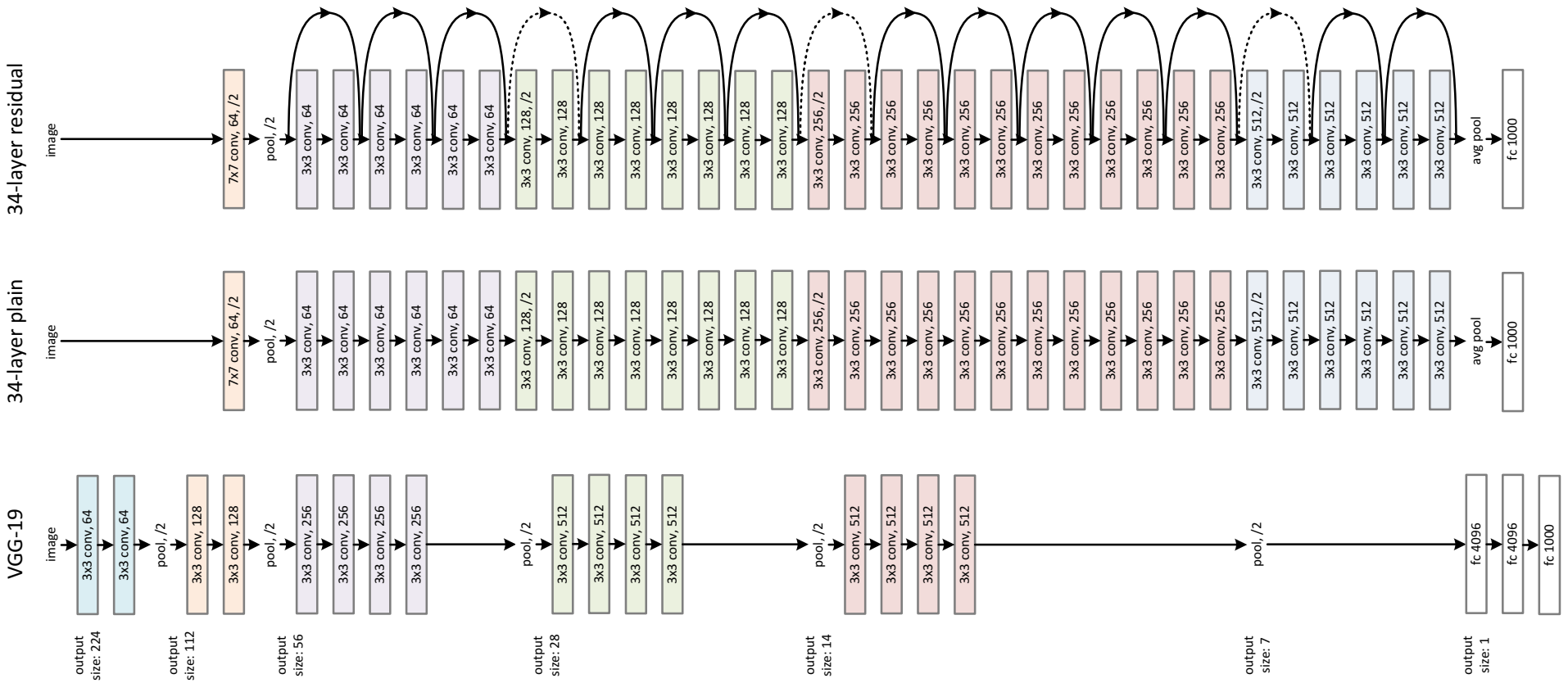


Original building block



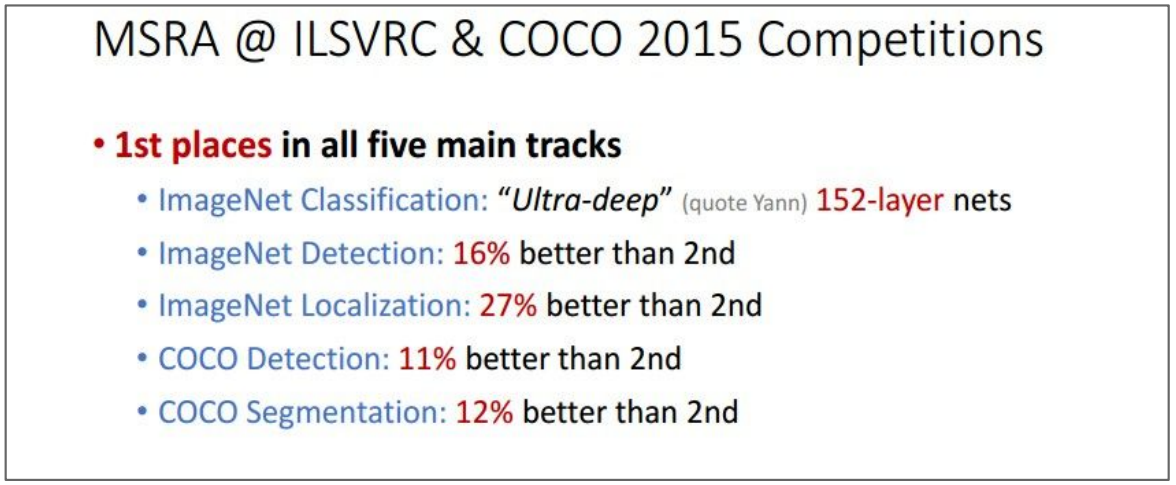
Building blocks for resnet-34 (left), and resnet-101 (right) with bottleneck layers

The ResNet 34



ResNet

- Able to train very deep models (152) layers
- Deeper layers now actually get lower training error



ImageNet Winners

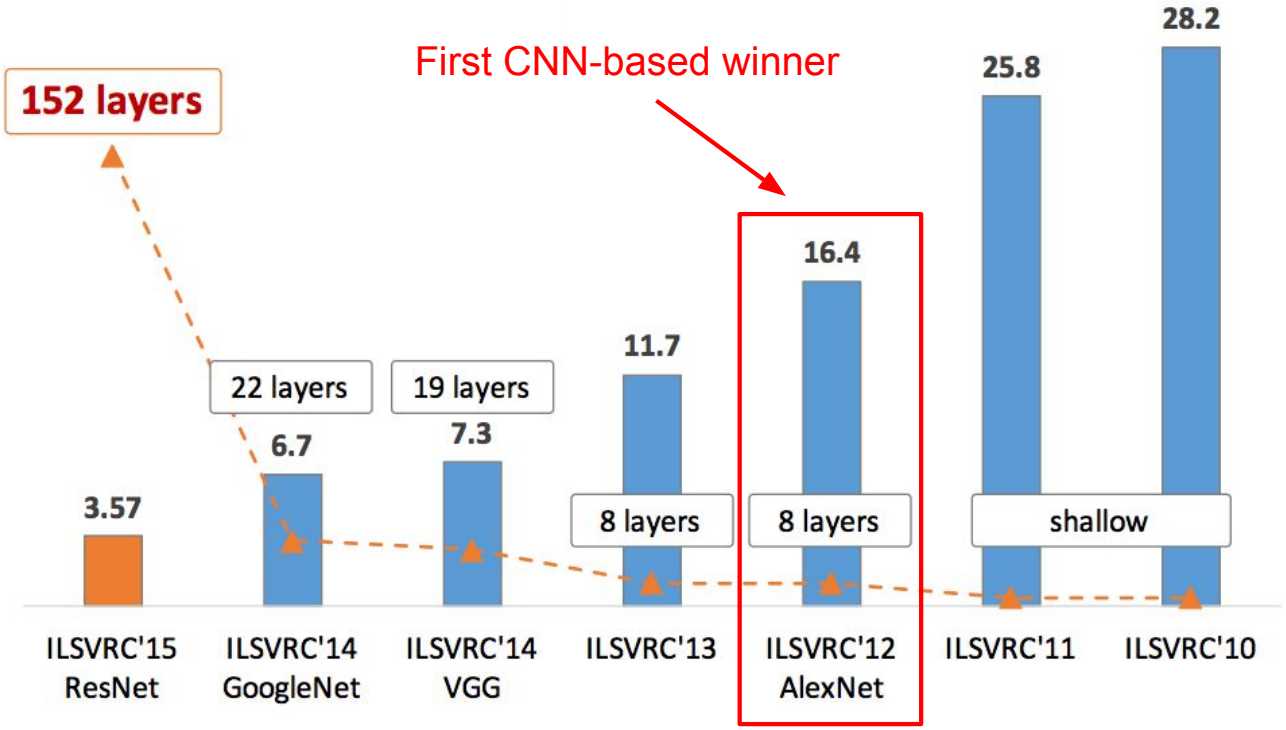


Figure copyright Kaiming He, 2016.

Batch Normalization

Problem:

- Distribution of input to layers change as previous layers are trained
- First input is normalized but input to other layers is not
- Slows down training as small learning rates are required or training does not converge

Solution:

- **Batch Normalization** to learn and adapt normalization of inputs

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe
Google Inc., *sioffe@google.com*

Christian Szegedy
Google Inc., *szegedy@google.com*

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch Normalization

- Improves Gradient flow
- Allows higher learning rates
- Reduces dependence on (correct) initialization
- Parameters are learned from mini-batches

- (Now) Usually inserted after convolutional and dense layers before the activation function

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Batch Normalization

```
keras.layers.BatchNormalization(
      axis=-1,
      momentum=0.99, epsilon=0.001,
      center=True, scale=True,
      beta_initializer="zeros", gamma_initializer="ones",
      moving_mean_initializer="zeros", moving_variance_initializer="ones",
      ...)
```

- **axis**: Integer, the axis that should be normalized (typically the features axis). For instance, after a Conv2D layer with data_format="channels_first", use axis=1.
- **momentum**: Momentum for the moving average.
- **epsilon**: Small float added to variance to avoid dividing by zero.
- **center**: If True, add offset of beta to normalized tensor. If False, beta is ignored.
- **scale**: If True, multiply by gamma. If False, gamma is not used. When the next layer is linear this can be disabled since the scaling will be done by the next layer.

Batch Normalization in Keras

Importantly, batch normalization works differently during training and during inference.

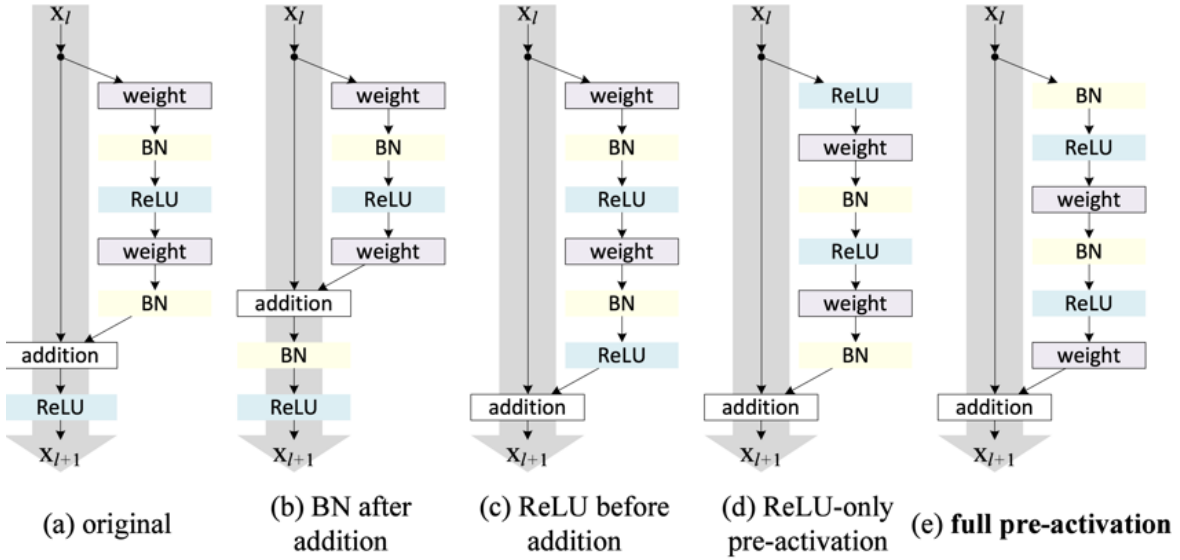
During training (i.e. when using fit() or when calling the layer/model with the argument training=True), the layer normalizes its output using the mean and standard deviation of the current batch of inputs. That is to say, for each channel being normalized, the layer returns gamma * (batch - mean(batch)) / sqrt(var(batch) + epsilon) + beta, where:

During inference (i.e. when using `evaluate()` or `predict()` or when calling the layer/model with the argument `training=False` (which is the default), the layer normalizes its output using a moving average of the mean and standard deviation of the batches it has seen during training. That is to say, it returns `gamma * (batch - self.moving_mean) / sqrt(self.moving_var+epsilon) + beta`.

https://keras.io/api/layers/normalization_layers/batch_normalization/

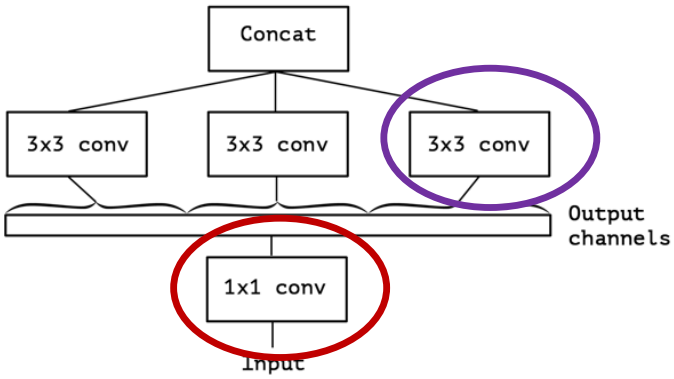
Residual connections and batch normalization

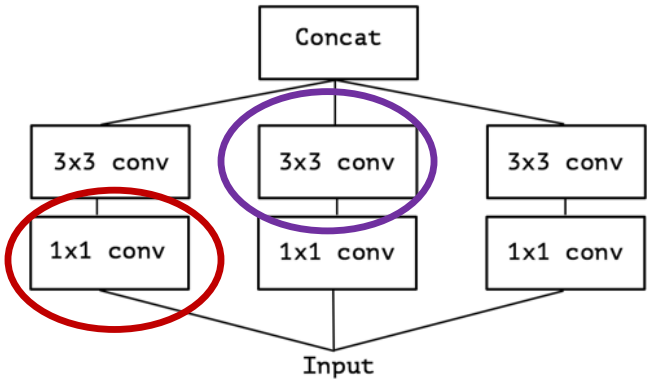
case	Fig.	ResNet-110	ResNet-164
original Residual Unit [1]	Fig. 4(a)	6.61	5.93
BN after addition	Fig. 4(b)	8.17	6.50
ReLU before addition	Fig. 4(c)	7.84	6.14
ReLU-only pre-activation	Fig. 4(d)	6.71	5.91
full pre-activation	Fig. 4(e)	**6.37**	**5.46**



Depthwise separable filters

- A convolutional filter has correlations across its depth (channel) and in its spatial dimension simultaneously
- Is this necessary or can the two be decoupled?
- Already in Inception blocks, there is some decoupling





Spatial correlations

Cross channel correlations

Depthwise separable convolutions

- Convolve **each channel** separately with a nxn filter to produce 1 channel output per input channel
- Use a 1x1 filter (called pointwise filter here) across the channels
- This is similar to the (extreme) inception module before, but in the reverse order

Separable Convolutional filterFilter

```
keras.layers.SeparableConv2D(
      filters,
      kernel_size, strides=(1, 1),
      padding="valid", ...
      depth_multiplier=1,...)
```

- **filters**: int, the dimensionality of the output space (i.e. the number of filters in the pointwise convolution).
- **kernel_size**: int or tuple/list of 2 integers, specifying the size of the depthwise convolution window
- **depth_multiplier**: The number of depthwise convolution output channels for each input channel. The total number of depthwise convolution output channels will be equal to input_channel * depth_multiplier.

Separable filters

```
inp = keras.Input(shape=(32, 32, 8))
x = keras.layers.Conv2D(4, kernel_size=(3, 3),
padding="same")(inp)
m = keras.Model(inp, x)
m.summary()
```

Layer (type)	Output Shape	Param #
input_layer_8 (InputLayer)	(None, 32, 32, 8)	0
conv2d_16 (Conv2D)	(None, 32, 32, 4)	292

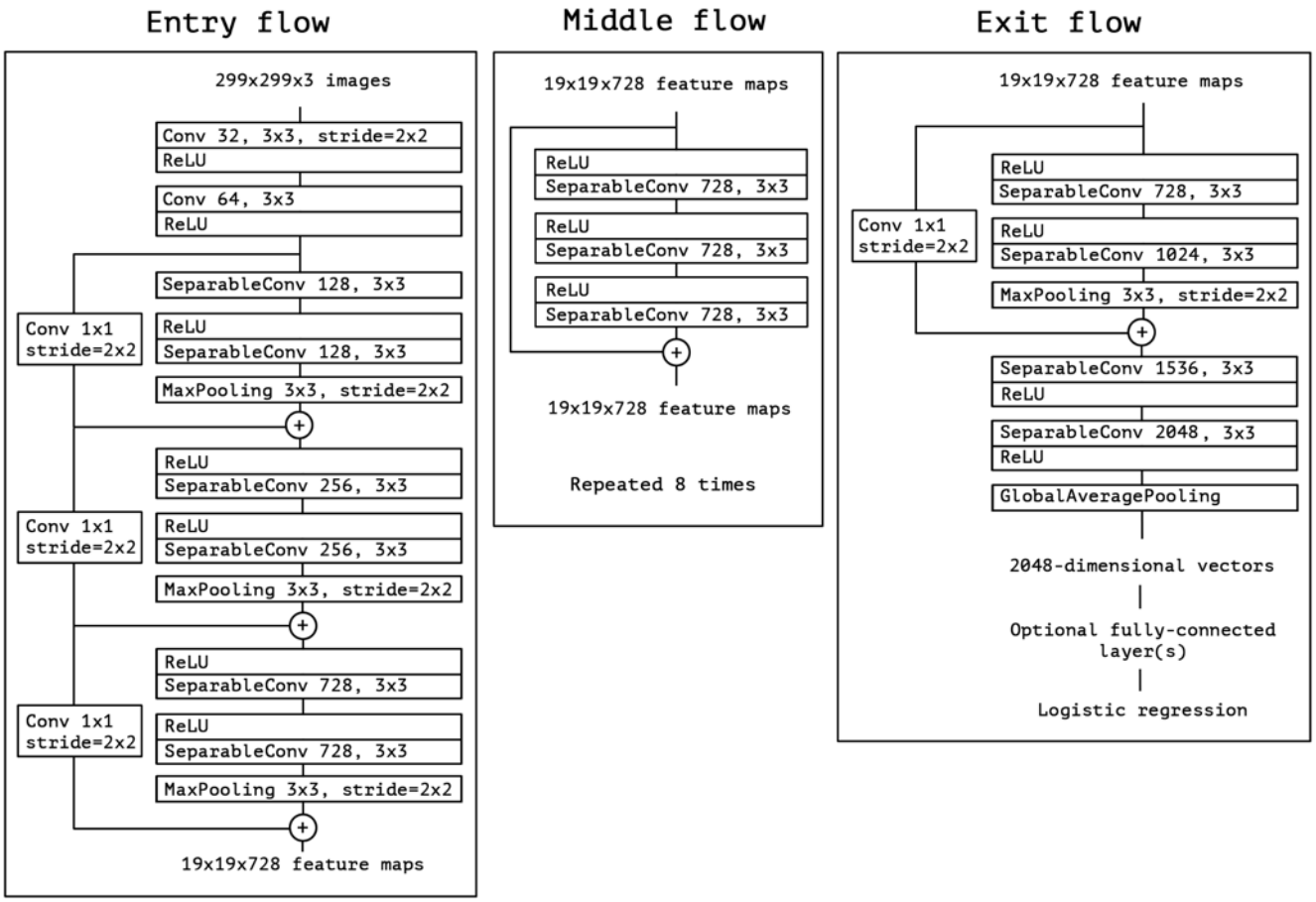
Total params: 292 (1.14 KB)

```
inp = keras.Input(shape=(32, 32, 8))
x = keras.layers.SeparableConv2D(4, kernel_size=(3, 3),
padding="same")(inp)
m = keras.Model(inp, x)
m.summary()
```

Layer (type)	Output Shape	Param #
input_layer_7 (InputLayer)	(None, 32, 32, 8)	0
separable_conv2d_1 (SeparableConv2D)	(None, 32, 32, 4)	108

Total params: 108 (432.00 B)

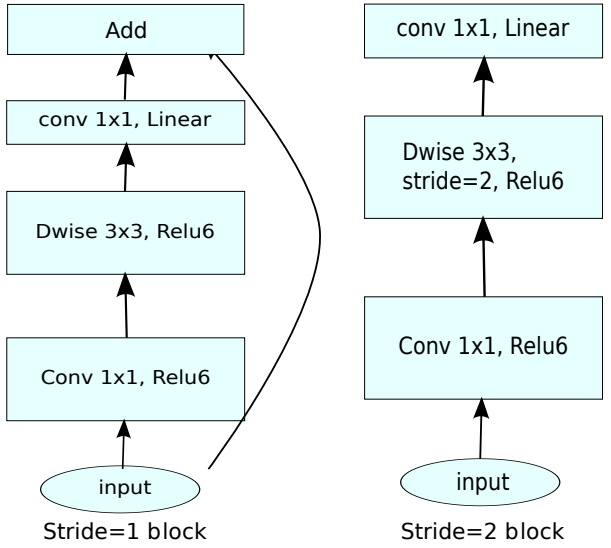
Xception Network: Depthwise Separable Convolutions

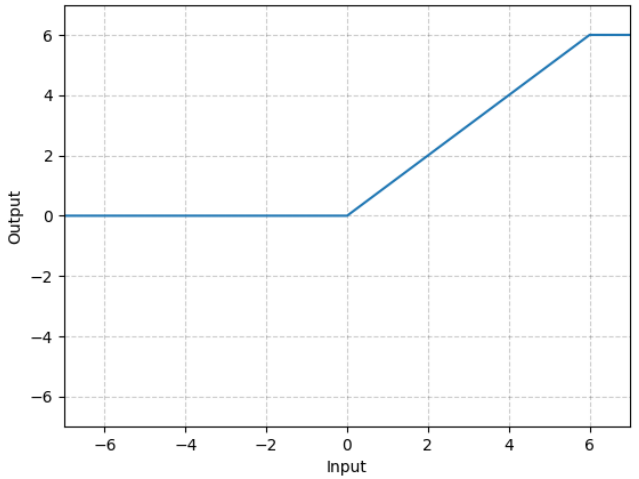


MobileNet V2

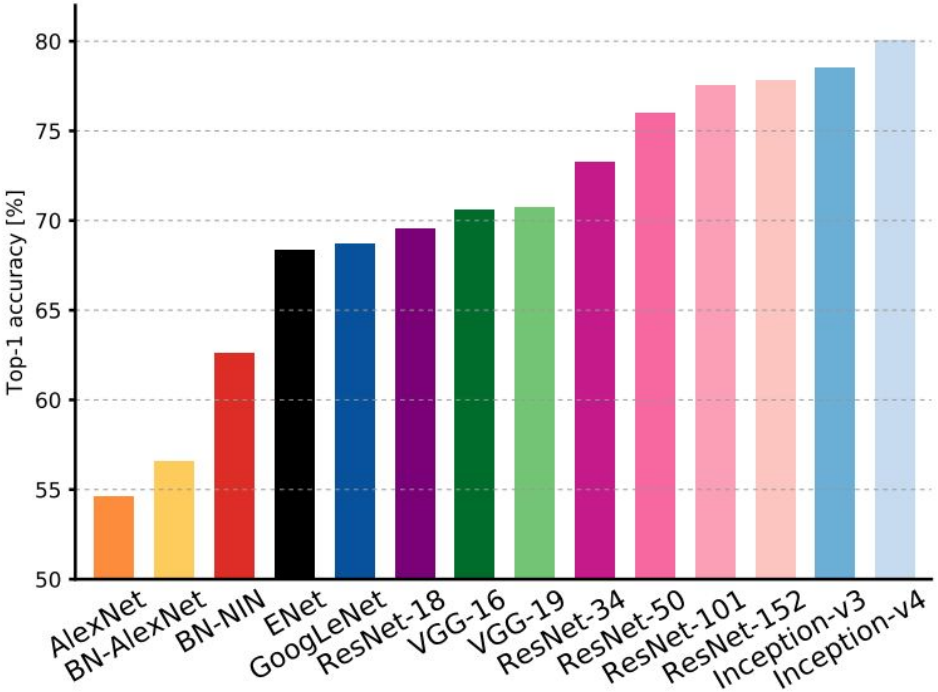
- Lightweight Design
- Bottleneck blocks with Depthwise seperable convolutions

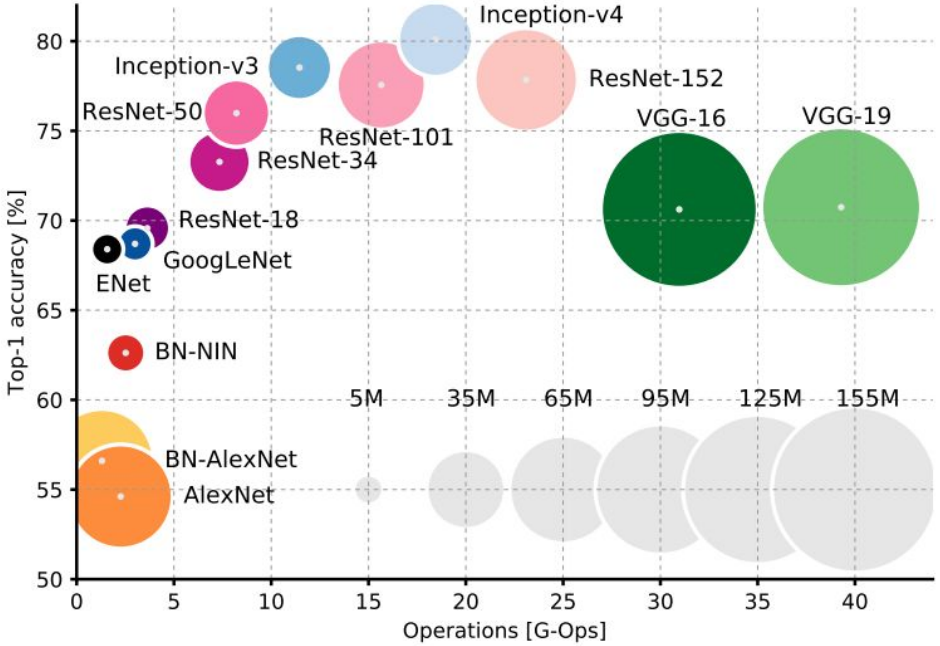
Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	



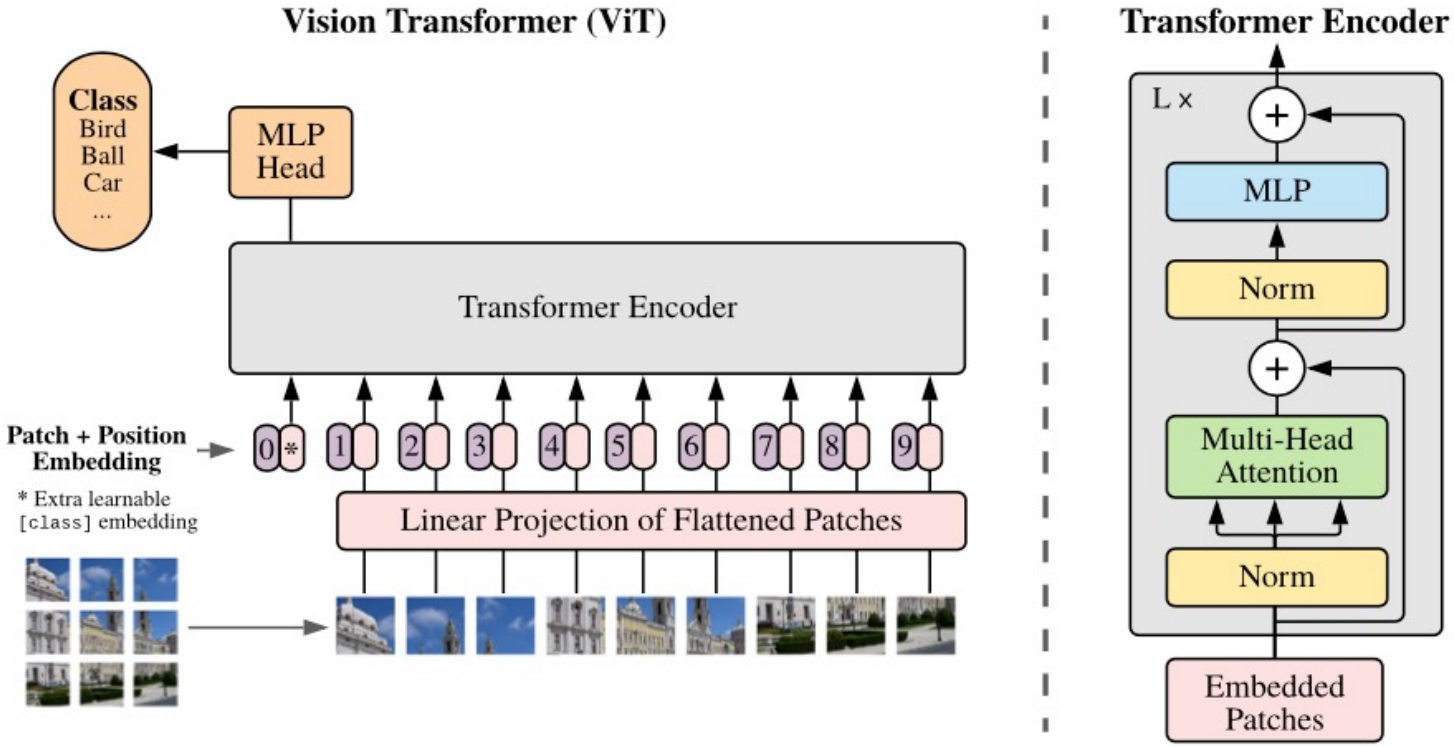


Comparison of Complexity

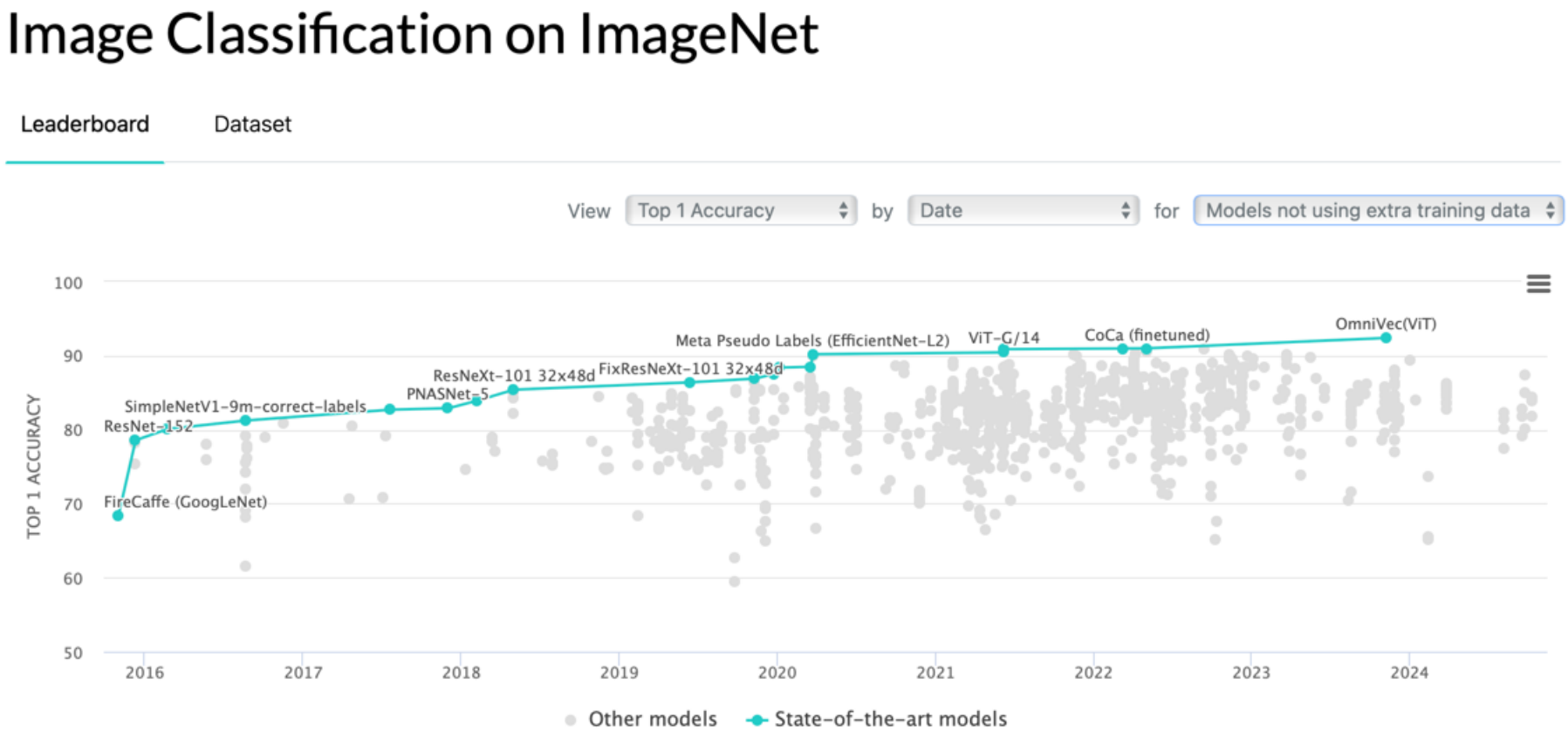




Newest Architecture: Vision Transformer → Next lecture



ImageNet: Newest results

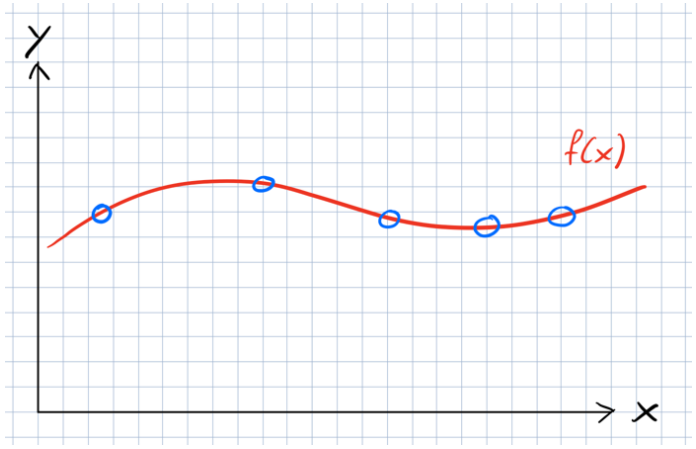


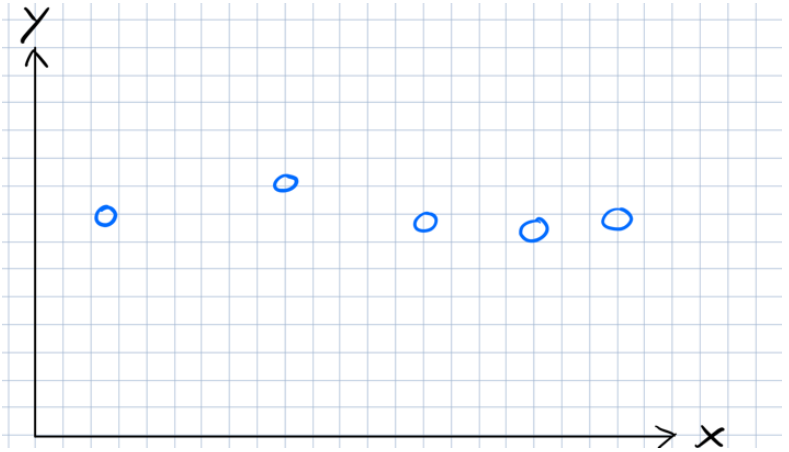
Training Deep Networks

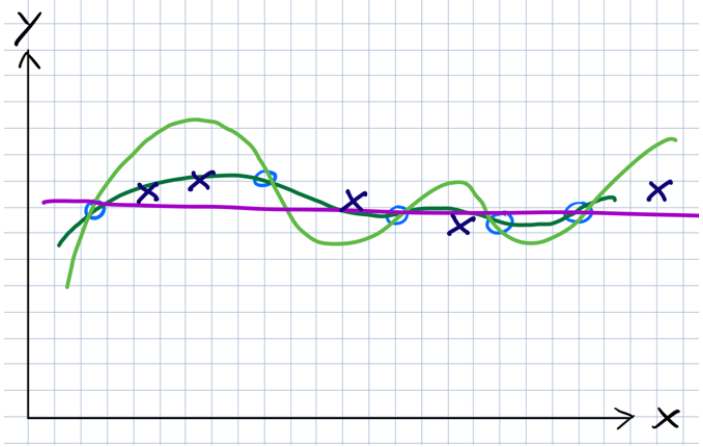
How deep and large should your network be?

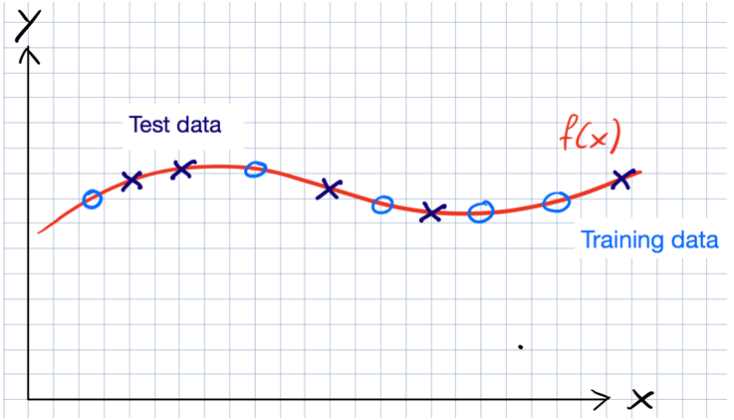
How can we train efficiently?

Example









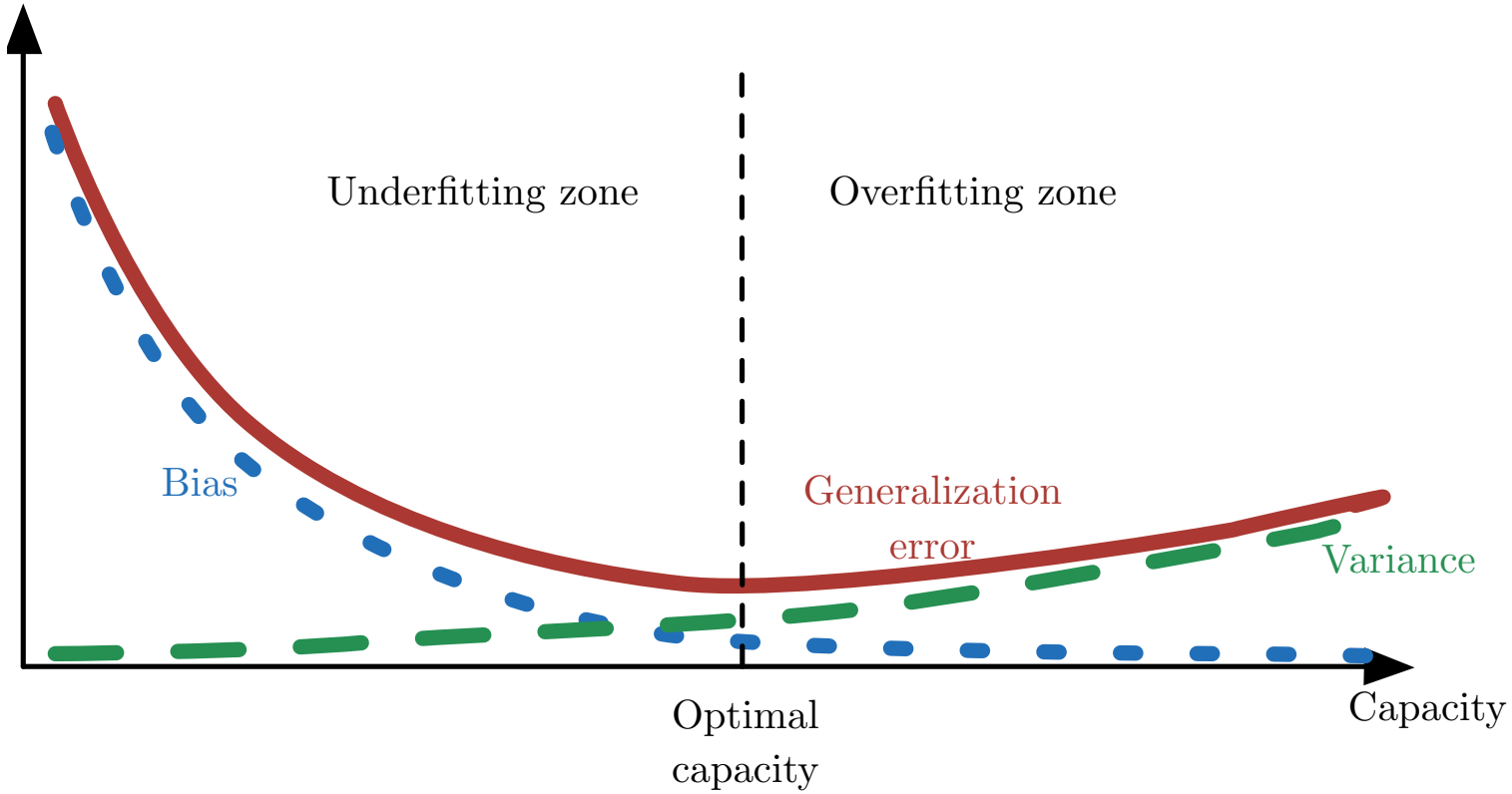
Errors

Training Error:	Error on training set
Validation Error:	Error on validation set
Generalization Error:	Gap between the error on the training and validation sets

Data Sets for training:

Training data set:	Data Set to run your training on
Validation data set:	Data Set to evaluate your trained model on and tune hyper parameters
Test data set:	Evaluation of final model

Optimal Capacity



Constrain model parameters

Problem:

- Add constraints to the model to favor less complex models, while keeping the depth

Effect:

- Complex models suffer from high variance error which leads to poor generalisation

Solution

- Regularisation, for example L2 or Dropout

L2 Regularization

Full loss function includes regularization over all parameters:

Classic view:

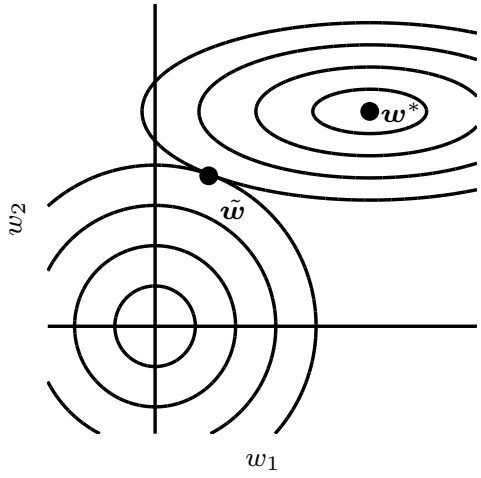
- Regularization works to prevent overfitting when we have a lot of features (or later a very powerful/deep model, etc.)

Now:

- Regularization produces models that generalize well when we have a “big” model
- We do not care that our models overfit on the training data, even though they are hugely overfit

L2 Regularization

Limit the capacity of networks by adding a parameter norm penalty to the loss



Goodfellow 2016

$$\tilde{J}(\mathbf{w}; X, y) = J(\mathbf{w}; X, y) + \alpha\Omega(\mathbf{w})$$

$$\Omega(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2$$

$$\tilde{J}(\mathbf{w}; X, y) = \frac{\alpha}{2}\mathbf{w}^T\mathbf{w} + J(\mathbf{w}; X, y)$$

$$\nabla_w \tilde{J}(\mathbf{w}; X, y) = \alpha\mathbf{w} + \nabla_w J(\mathbf{w}; X, y)$$

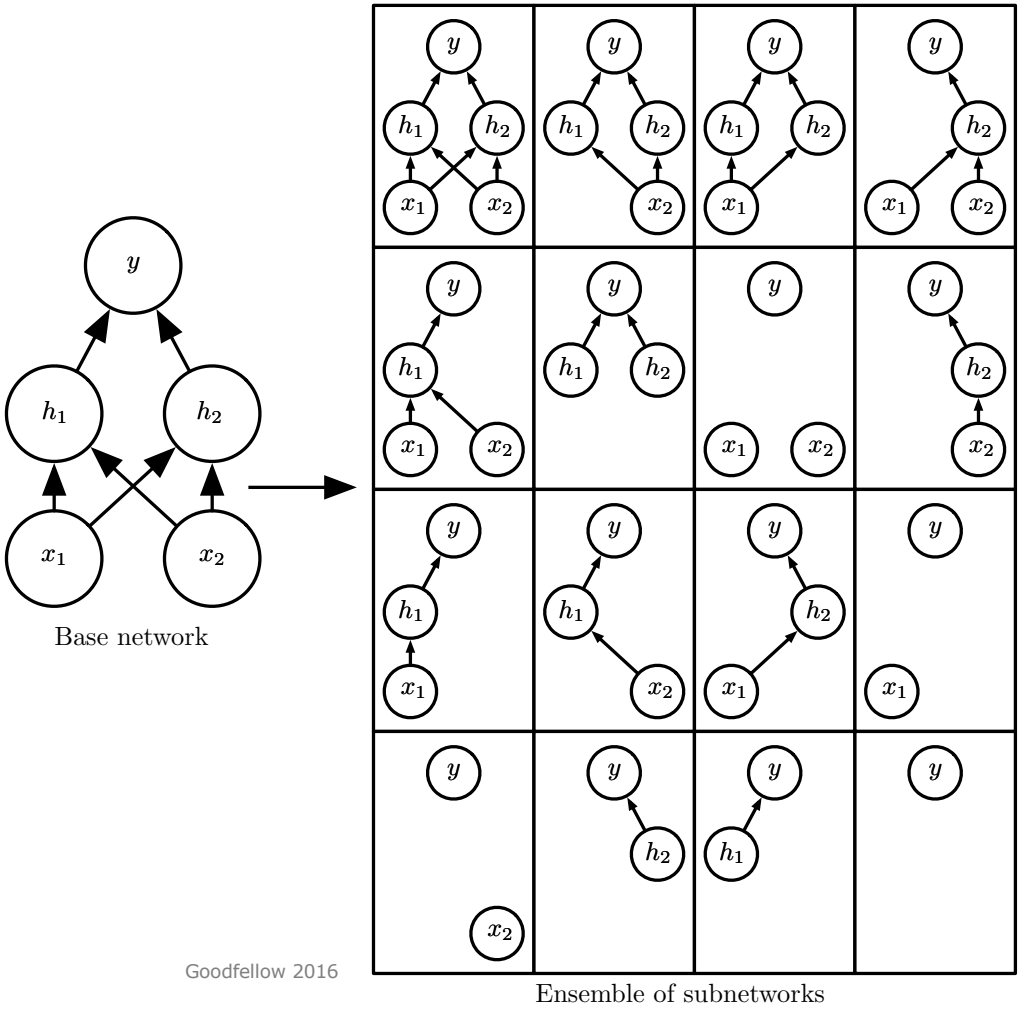
Other regularizations

Ensemble methods:

- Use several models that are trained separately and then vote on the result

Dropout:

- Randomly drop a node on a hidden layer or an input value
- (Effectively multiply the result of that node by 0)
- Training learns all sub networks of the original network



Base network

Ensemble of subnetworks

Goodfellow 2016

Dropout

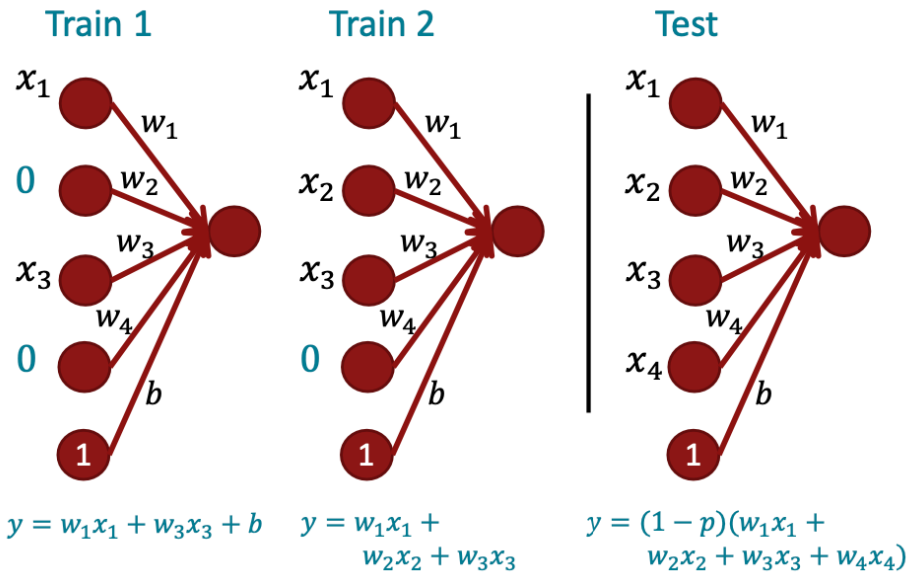
During training:

- Randomly set input to 0 with probability p (dropout ratio)

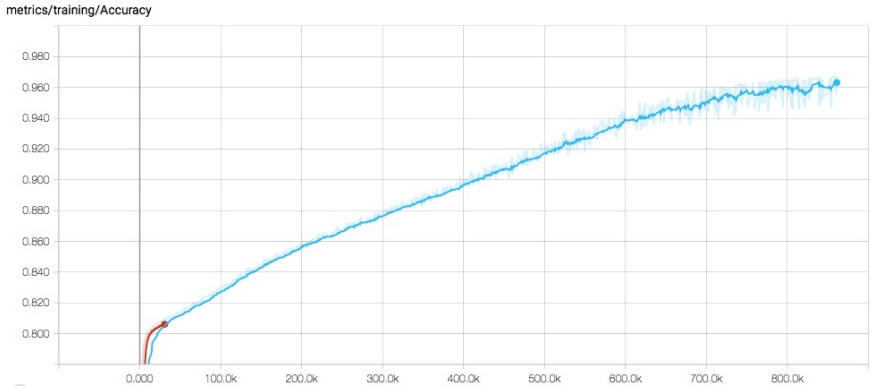
During testing / evaluation

- Multiply all weights by 1-p

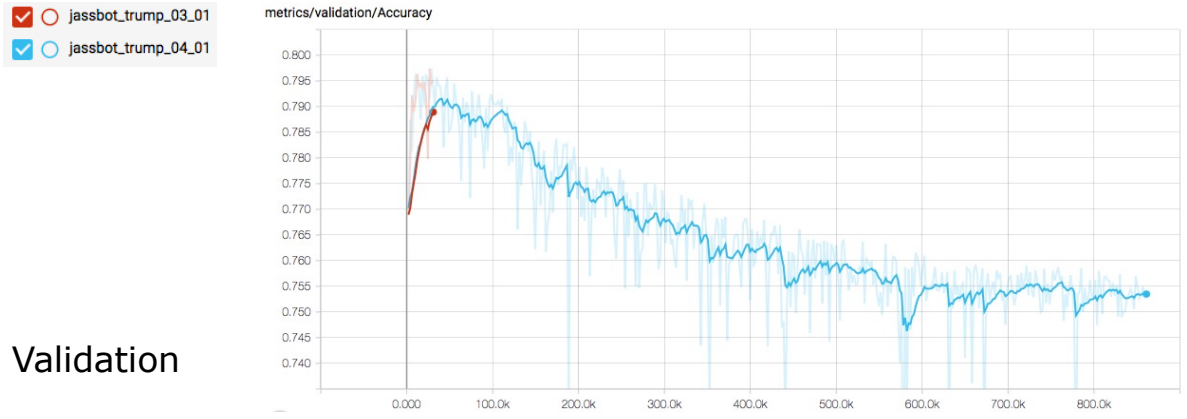
- (Alternatively multiply weights at training with 1 / (1-p))



Overfitting Example



Training



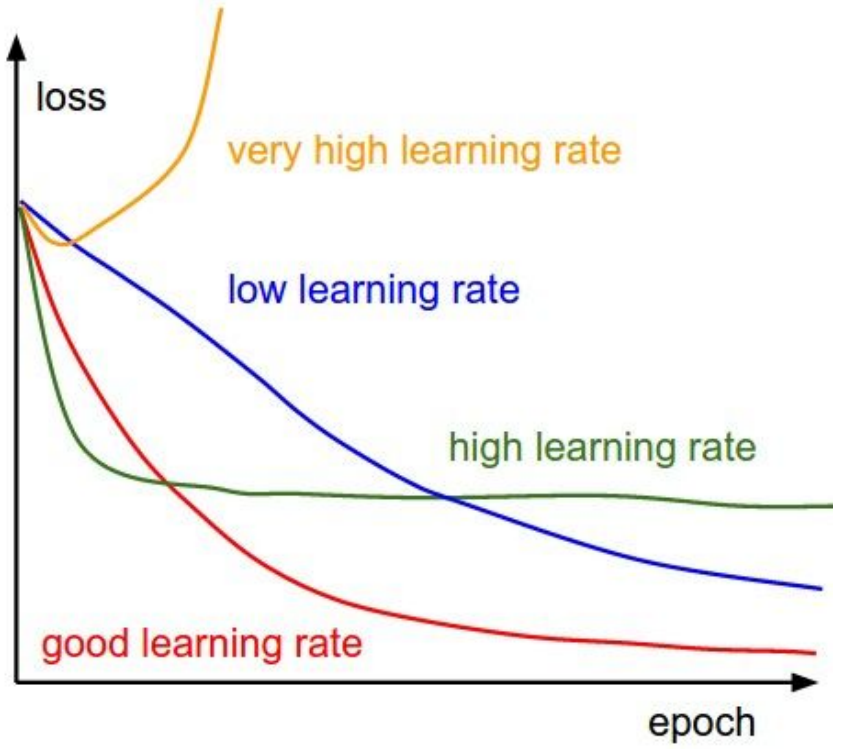
Validation

Hyperparameter Optimization

First **coarse** search of parameters, then finer search

Most important hyperparameters to try out:

- Network capacity/architecture → **Overfit first**
- **Learning rate**, decay rate if used
- Regularization weights
- Dropout percentage



Babysit the Learning Process

Tipps:

- Start with simple/little data and see that your model converges (will overfit)
- Check Loss before and after regularization, Check percentage of loss due to regularization
- Experiment with learning rate:
  - Does the loss go down fast enough?
  - Can you make the learning rate larger?

Tensorboard

- Display scalars (loss, metrics)
- Display distributions (weights, gradients)
- Display graph
- Visualize weights as images (?)
- Visualize embeddings

Use separate log directory for each run / experiment

Point tensorboard to the parent directory

```
tensorboard --logdir ./logs
```

```
tensorboard = \
    keras.callbacks.TensorBoard(
        log_dir=log_dir,
        histogram_freq=1,
        batch_size=batch_size,
        write_graph=True,
        write_grads=True)

model.fit(x, y, validation_split=0.25,
          epochs=nr_epochs,
batch_size=batch_size,
          shuffle=True,
          callbacks=[tensorboard])
```

Alternative: wandb

https://wandb.ai/

pip install wandb

```
import wandb

…

wandb.tensorboard.patch(root_logdir=log_dir)
wandb.init(sync_tensorboard=True, project='cifar-100')
```

Exercise:
Classification on the CIFAR-100 Dataset

Superclass	Classes
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor