UiT

THE ARCTIC
UNIVERSITY
OF NORWAY

Deep learning by convolutional networks

Michael Kampffmeyer[1]

Geilo Winter School, 16. Jan 2017

[1]michael.c.kampffmeyer@uit.no

UNIVERSITY OF TROMSØ

What we will cover...

Introduction

Background

CNNs

Practical tips & software

Segmentation & object detection

Conclusion

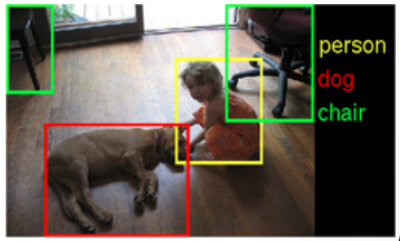
Slide inspirations

Hugo Larochelle: Neural networks course
Christopher Olah: http://colah.github.io
Fei-Fei Li, Andrej Karpathy and Justin Johnson: cs231n slides

Deep Learning is everywhere

Deep Learning is everywhere

- Image processing
  - Classification
  - Segmentation
  - Localization
  - Detection
- Speech and text processing
  - Translation
  - Caption generation
  - Word embeddings
  - Sequence prediction
- Reinforcement learning
  - Automatic game playing
- ... and much more

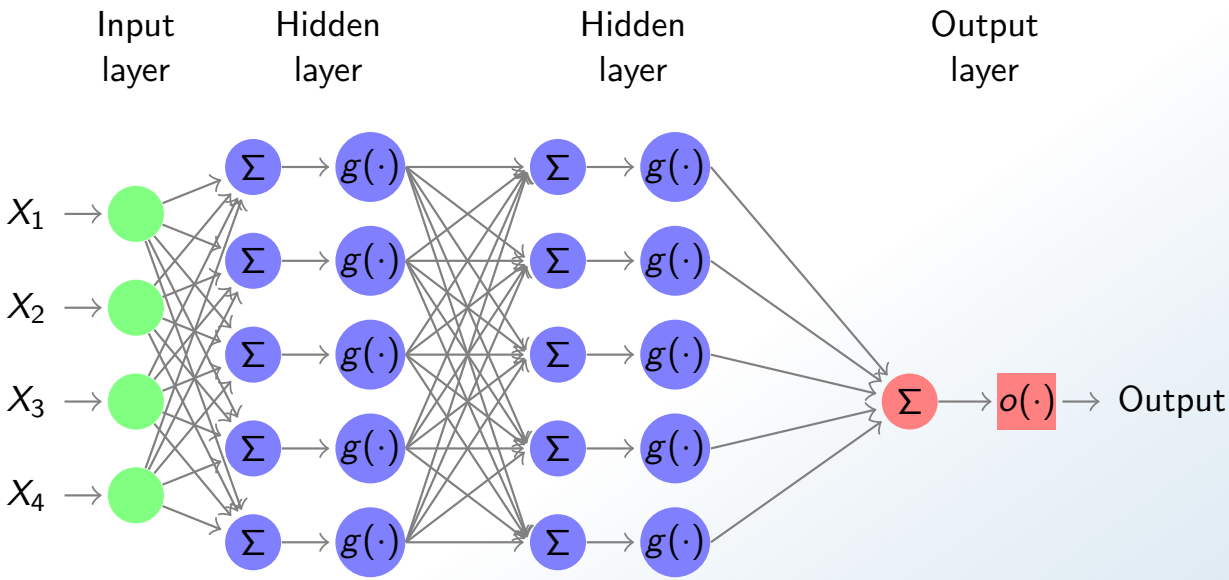


[http://image-net.org]

[Karpathy, 2015]

[https://deepmind.com]

Neural Networks



Neurons

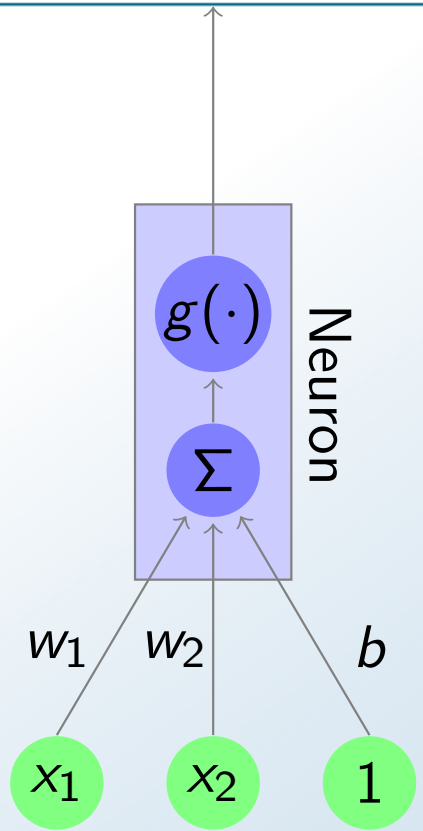
Pre-activation of single neuron

$$a(\boldsymbol{x}) = b + \sum_i w_i x_i = b + \boldsymbol{w}^T \boldsymbol{x}$$

Output of neuron

$$h(\boldsymbol{x}) = g(a(\boldsymbol{x}))$$

b is the bias
$\boldsymbol{w}$ are the weights
$g()$ is the activation function



Neurons

Theorem (Universal approximation theorem)

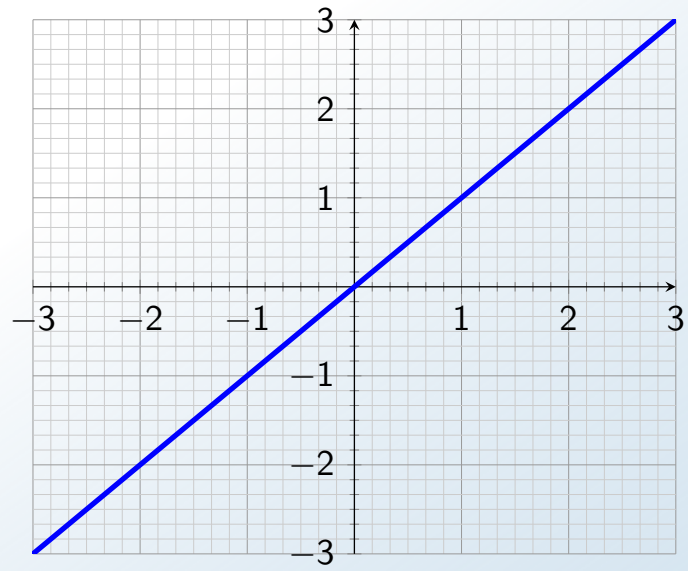
"A single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units" (Hornik, 1991)

- However, learning it is very difficult
- In practice use hierarchical representations

Activations - Linear

- No input squashing
- Not used in practice

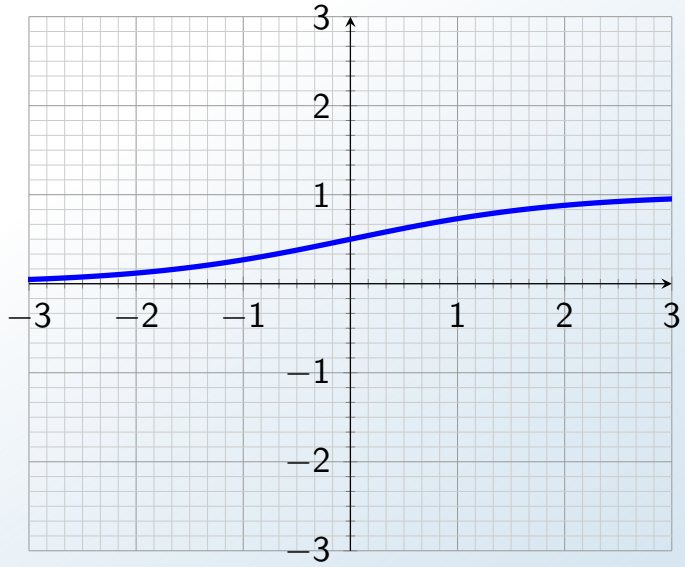
$$g(a) = a$$



Activations - Sigmoid

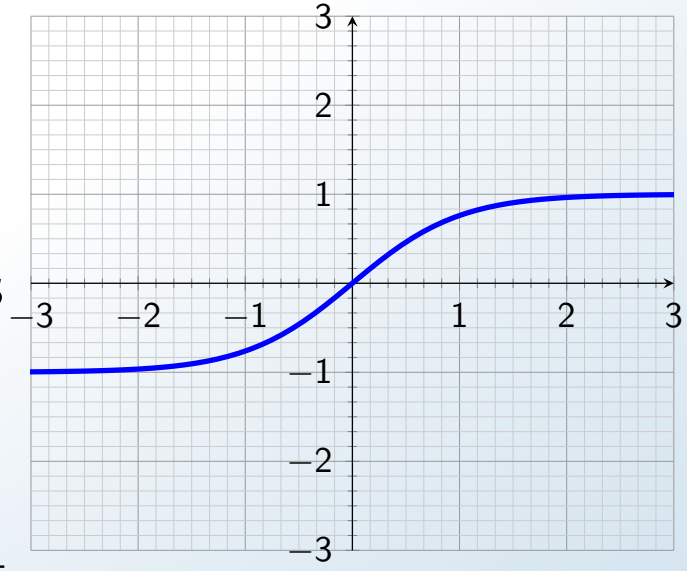
- Squashes between 0 and 1
- Strictly increasing
- Bounded
- Always positive
- Used in AE, RNN, shallow CNNs

$$g(a) = sigmoid(a) = \frac{1}{1 + e^{-a}}$$



Activations - Tanh

- Squashes between -1 and 1
- Strictly increasing
- Bounded
- Both positive and negative activations
- Used mainly in RNN

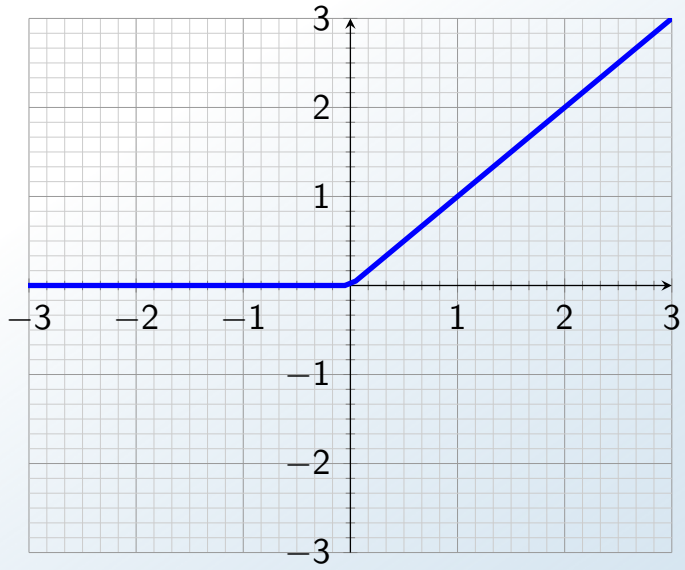


$$g(a) = tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} = \frac{e^{2a} - 1}{e^{2a} + 1}$$

Activations - ReLU

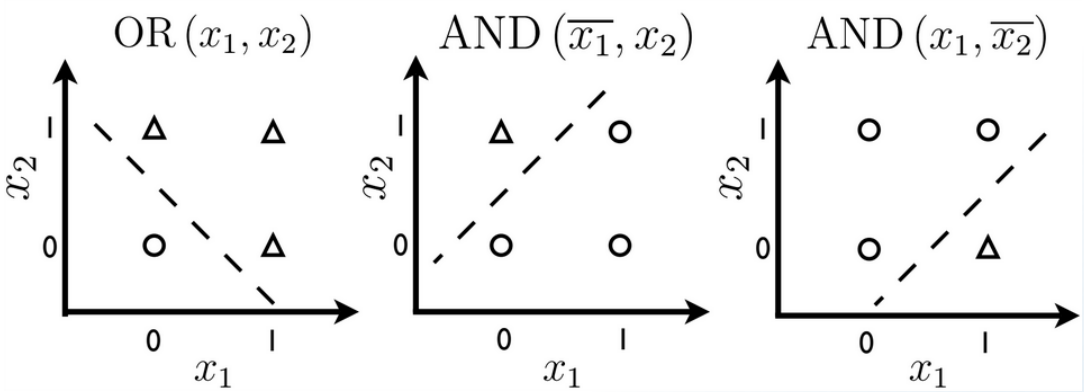
- Not decreasing
- Bounded lower end
- Sparse activations (faster training)
- More robust (vanishing gradients)
- Used in CNNs

$$g(a) = ReLU(a) = max(a, 0)$$



Stacking Neurons

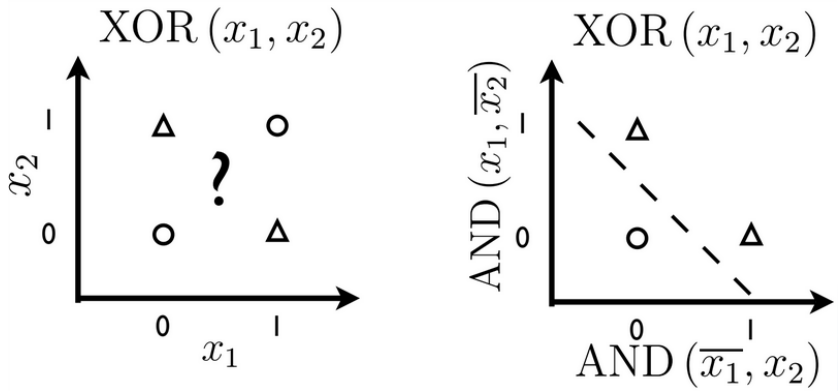
A single neuron can solve linear problems



[Source: Hugo Larochelle]

Stacking Neurons

But not nonlinear problems



[Source: Hugo Larochelle]

These require transformations

Power of hierarchical representations

Forward pass

Hidden layer (pre-activation)

$$\boldsymbol{a}^{(k)}(\boldsymbol{x}) = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)}\boldsymbol{h}^{(k-1)}(\boldsymbol{x})$$

Hidden layer activation

$$\boldsymbol{h}^{(k)}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{a}^{(k)}(\boldsymbol{x}))$$

Output layer

$$\boldsymbol{h}^{(L+1)}(\boldsymbol{x}) = \boldsymbol{o}(\boldsymbol{a}^{(L+1)}(\boldsymbol{x})) = \boldsymbol{f}(\boldsymbol{x})$$

Output activation

- Common multi-class classification loss function
- Want
  - Estimate $p(y = c|\boldsymbol{x})$
  - Strictly positive
  - Sums to 1
- Bounded lower end

$$o(\boldsymbol{a}) = softmax(\boldsymbol{a}) = \left[\frac{e^{(a_1)}}{\sum_c e^{(a_c)}} \cdots \frac{e^{(a_C)}}{\sum_c e^{(a_c)}} \right]$$

Classification loss function

Lossfunction: Measure of goodness of how well the model is performing

In classification want to estimate

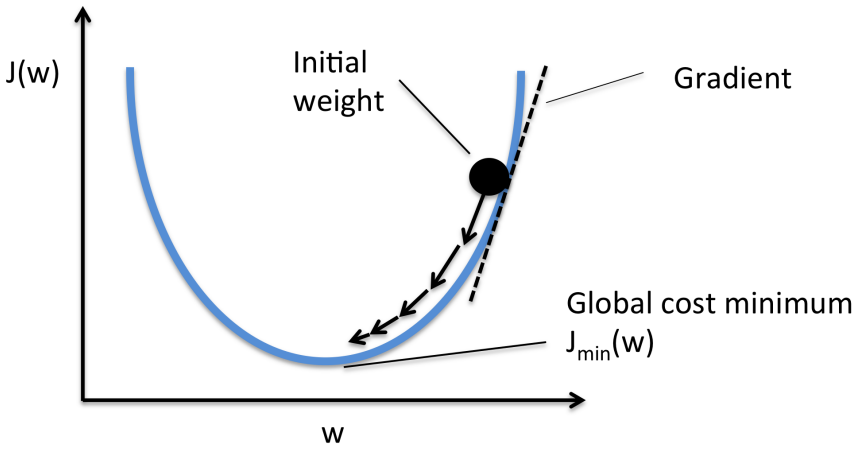
$$f(\boldsymbol{x})_c = p(y = c|\boldsymbol{x})$$

Reformulated as minimization problem (minimize negative log-likelihood)

$$\ell(f(\boldsymbol{x}), y) = -\sum_c 1_{y=c} \log f(\boldsymbol{x})_c = -\log f(\boldsymbol{x})_y$$

Reminder: Gradient descent

- Minimizing loss function by following gradient
- Mini-batch SGD



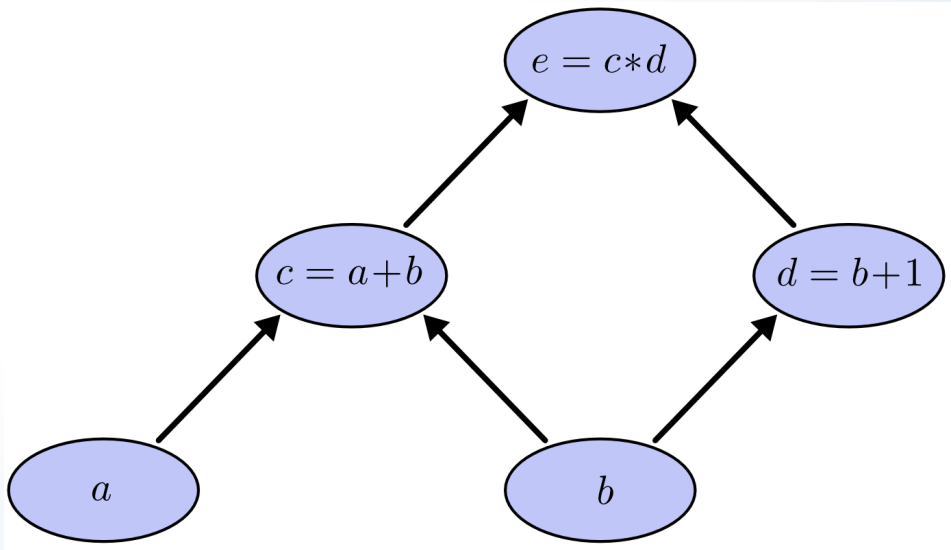
Data: Training samples
Result: Trained model
initialize parameters Θ ;
for *N epochs* do
  for *each training sample* $(\boldsymbol{x}_t, y_t)$ do
    $\Delta = -\nabla_\Theta \ell(f(\boldsymbol{x}_t; \Theta), y_t) - \lambda \nabla_\Theta \Omega(\Theta)$;
    $\Theta \leftarrow \Theta + \alpha \Delta$;
  end
end

[Source: sebastianraschka.com]

Backpropagation - Intuition

Common abstraction for neural networks: Computation graph

- $e = (a + b) * (b + 1)$

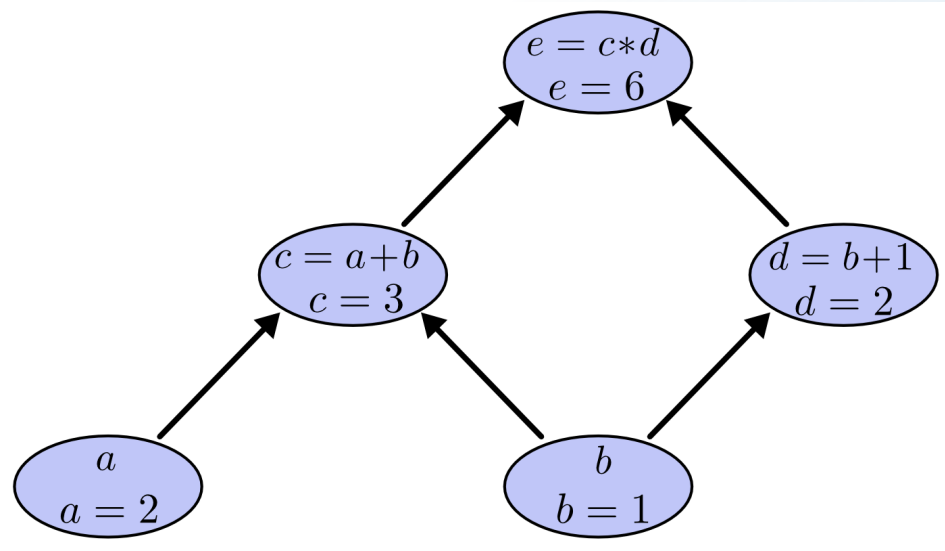


[Source: Christopher Olah (http://colah.github.io)]

Backpropagation - Intuition

Common abstraction for neural networks: Computation graph

- $e = (a + b) * (b + 1)$
- $a = 2$ and $b = 1$



[Source: Christopher Olah (http://colah.github.io)]

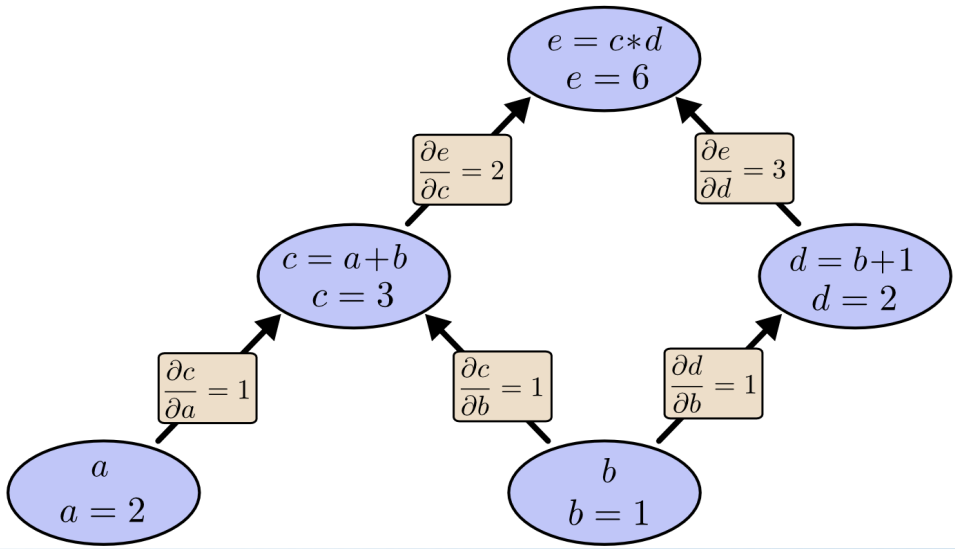
Backpropagation - Intuition

Common abstraction for neural networks: Computation graph

- $e = (a + b) * (b + 1)$
- $a = 2$ and $b = 1$

Compute partial derivatives

- $\frac{\partial}{\partial a}(a + b) = \frac{\partial a}{\partial a} + \frac{\partial b}{\partial a} = 1$
- $\frac{\partial}{\partial c}(c * d) = c\frac{\partial d}{\partial c} + d\frac{\partial c}{\partial c} = d$



[Source: Christopher Olah (http://colah.github.io)]

Backpropagation - Intuition

Common abstraction for neural networks: Computation graph

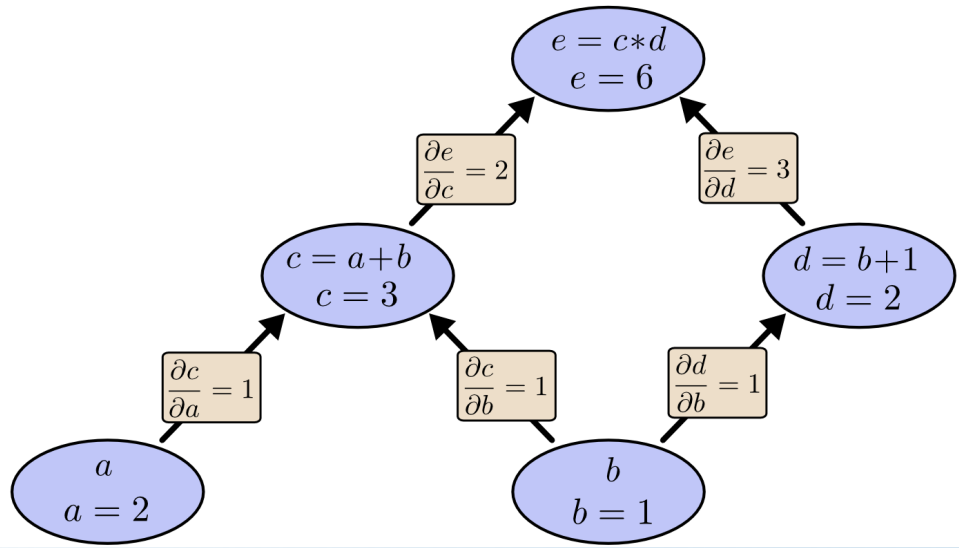
- $e = (a + b) * (b + 1)$
- $a = 2$ and $b = 1$

Compute partial derivatives

- $\frac{\partial}{\partial a}(a + b) = \frac{\partial a}{\partial a} + \frac{\partial b}{\partial a} = 1$
- $\frac{\partial}{\partial c}(c * d) = c\frac{\partial d}{\partial c} + d\frac{\partial c}{\partial c} = d$

Compute $\frac{\partial e}{\partial b}$

- Multivariate chain rule
  - $\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c}\frac{\partial c}{\partial b} + \frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$
- $\frac{\partial e}{\partial b} = 2 * 1 + 3 * 1$

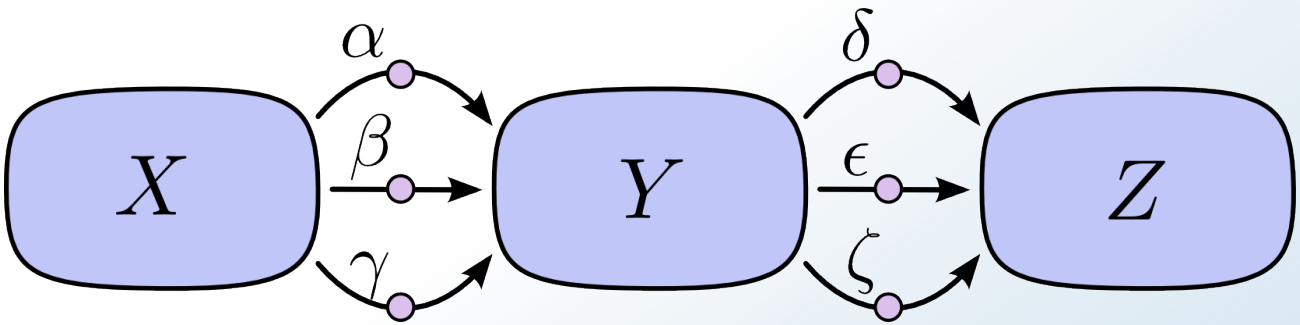


[Source: Christopher Olah (http://colah.github.io)]

Backpropagation - Intuition

Getting closer to a Neural Networks

- A more complex example with 9 paths
- Computing $\frac{\partial Z}{\partial X} = \alpha\delta + \alpha\epsilon + \alpha\zeta + \beta\delta + \beta\epsilon + \beta\zeta + \gamma\delta + \gamma\epsilon + \gamma\zeta$
- Does not scale to large networks

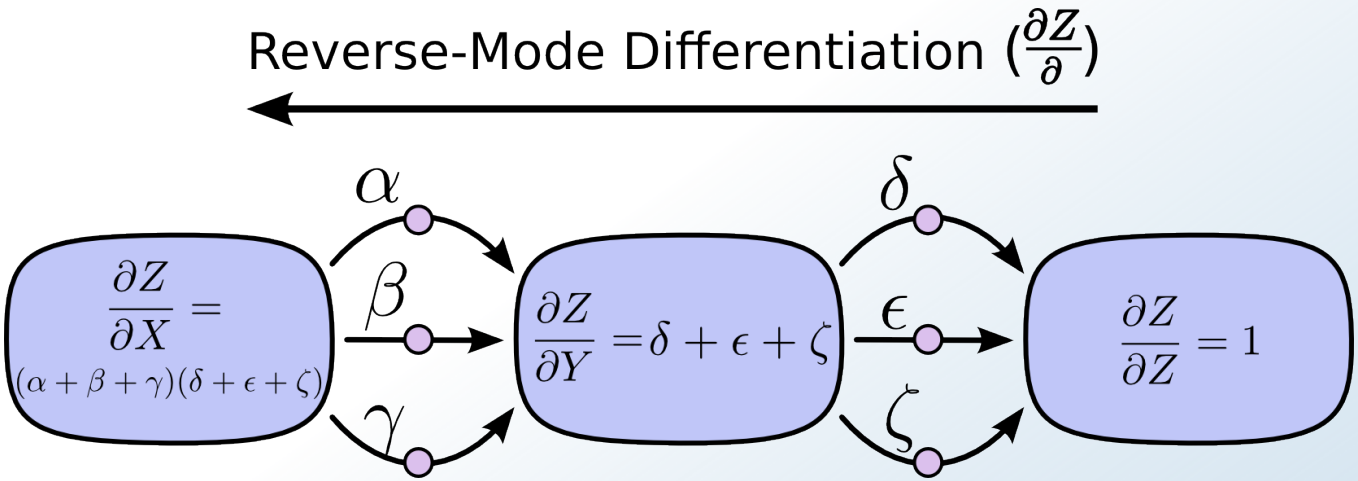


[Source: Christopher Olah (http://colah.github.io)]

Backpropagation - Intuition

Getting closer to a Neural Networks

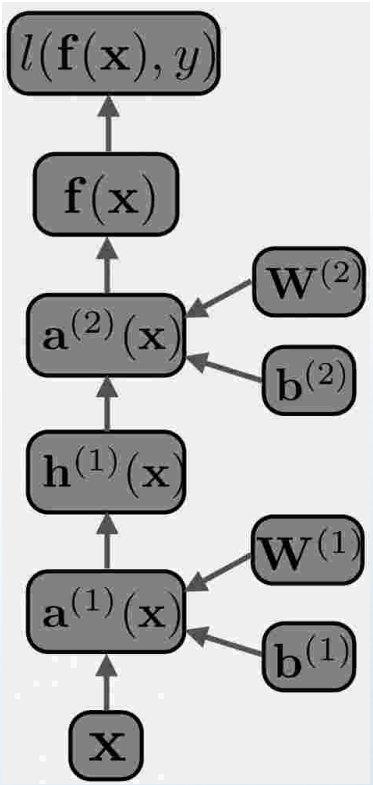
- Backpropagation more efficient
- Computing $\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma)(\delta + \epsilon + \zeta)$



[Source: Christopher Olah (http://colah.github.io)]

Backpropagation - Intuition

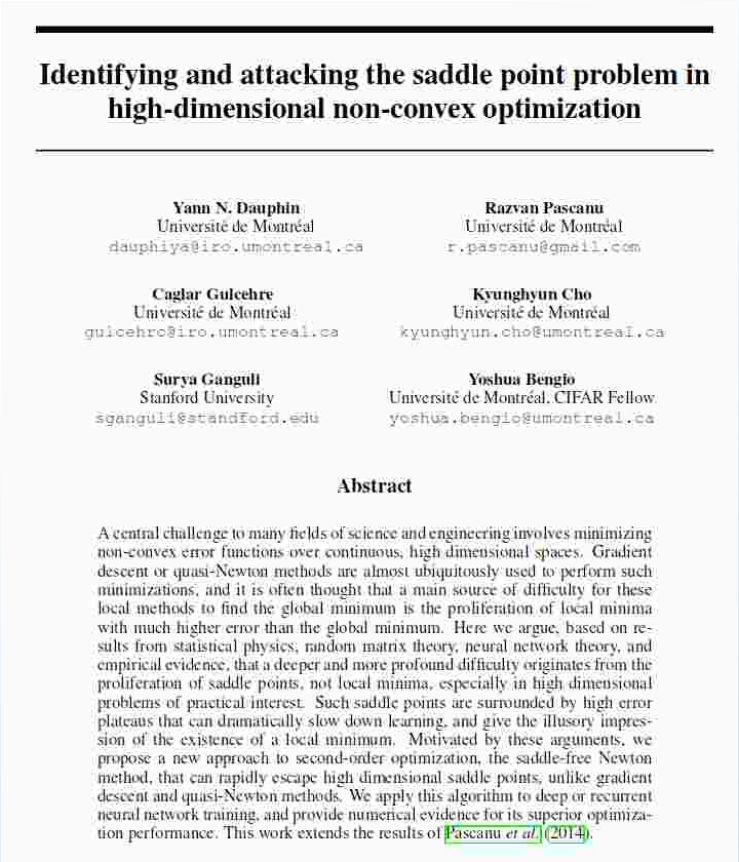
- Self containing modules
- Forward propagation compute output based on child layer(s)
- Backward propagation compute gradient wrt. children based on parent layer(s)



[Source: Hugo Larochelle]

Problems with deep networks

- Optimization more difficult
  - Vanishing gradients (Filippo)
- Overfitting is a problem
  - Better regularization
- However, many benefits



Identifying and attacking the saddle point problem in high-dimensional non-convex optimization

Yann N. Dauphin
Université de Montréal
dauphiya@iro.umontreal.ca

Razvan Pascanu
Université de Montréal
r.pascanu@gmail.com

Caglar Gulcehre
Université de Montréal
gulcehrc@iro.umontreal.ca

Kyunghyun Cho
Université de Montréal
kyunghyun.cho@umontreal.ca

Surya Ganguli
Stanford University
sganguli@standford.edu

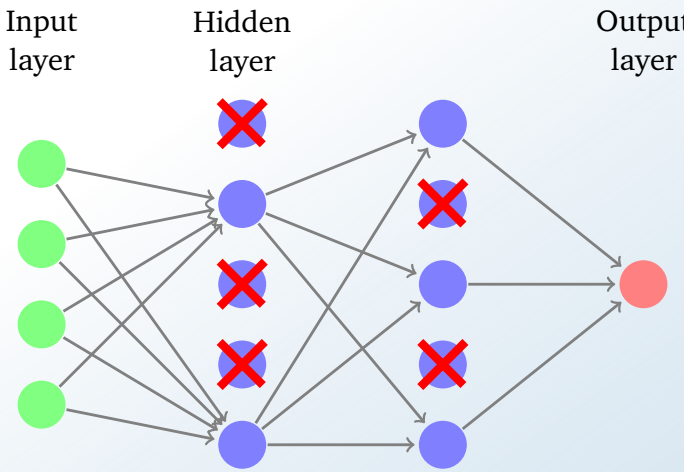
Yoshua Bengio
Université de Montréal, CIFAR Fellow
yoshua.bengio@umontreal.ca

Abstract

A central challenge to many fields of science and engineering involves minimizing non-convex error functions over continuous, high dimensional spaces. Gradient descent or quasi-Newton methods are almost ubiquitously used to perform such minimizations, and it is often thought that a main source of difficulty for these local methods to find the global minimum is the proliferation of local minima with much higher error than the global minimum. Here we argue, based on results from statistical physics, random matrix theory, neural network theory, and empirical evidence, that a deeper and more profound difficulty originates from the proliferation of saddle points, not local minima, especially in high dimensional problems of practical interest. Such saddle points are surrounded by high error plateaus that can dramatically slow down learning, and give the illusory impression of the existence of a local minimum. Motivated by these arguments, we propose a new approach to second-order optimization, the saddle-free Newton method, that can rapidly escape high dimensional saddle points, unlike gradient descent and quasi-Newton methods. We apply this algorithm to deep or recurrent neural network training, and provide numerical evidence for its superior optimization performance. This work extends the results of Pascanu *et al.* (2014).

Dropout regularization (Hinton et al. 2012)

- Training
  - Drop units with dropout probability p
  - Reduces co-adaption
- Test
  - Scale weights by dropout rate (1-p)



Batch normalization (Ioffe and Szegedy, 2015)

- Normalize pre-activation
- Training
  - Normalize batch by mean and std
- Test
  - Normalize by global mean and std

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

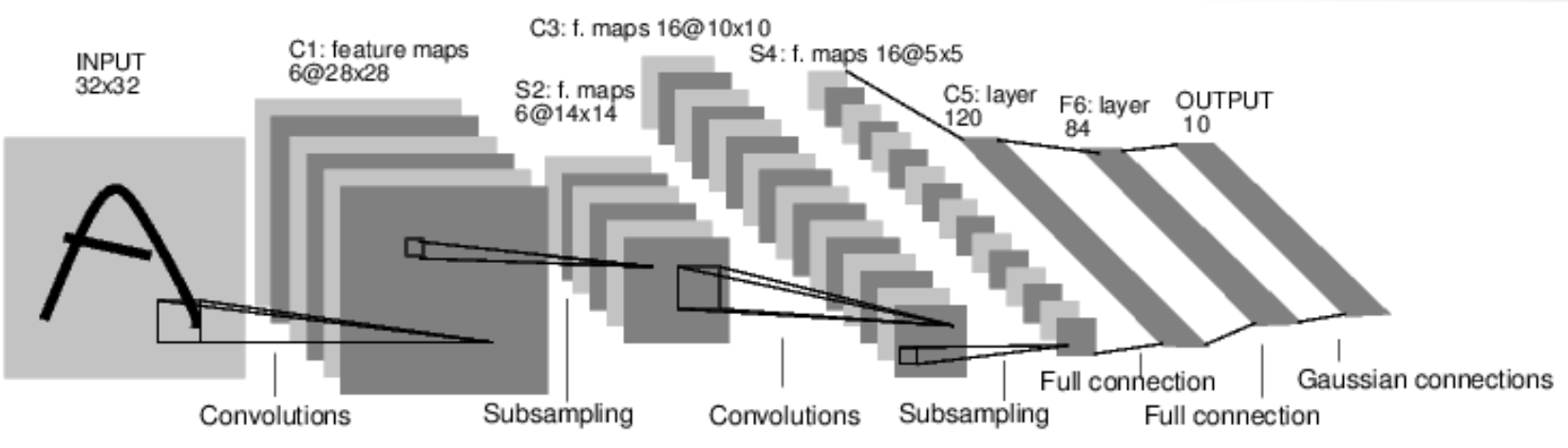
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

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

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

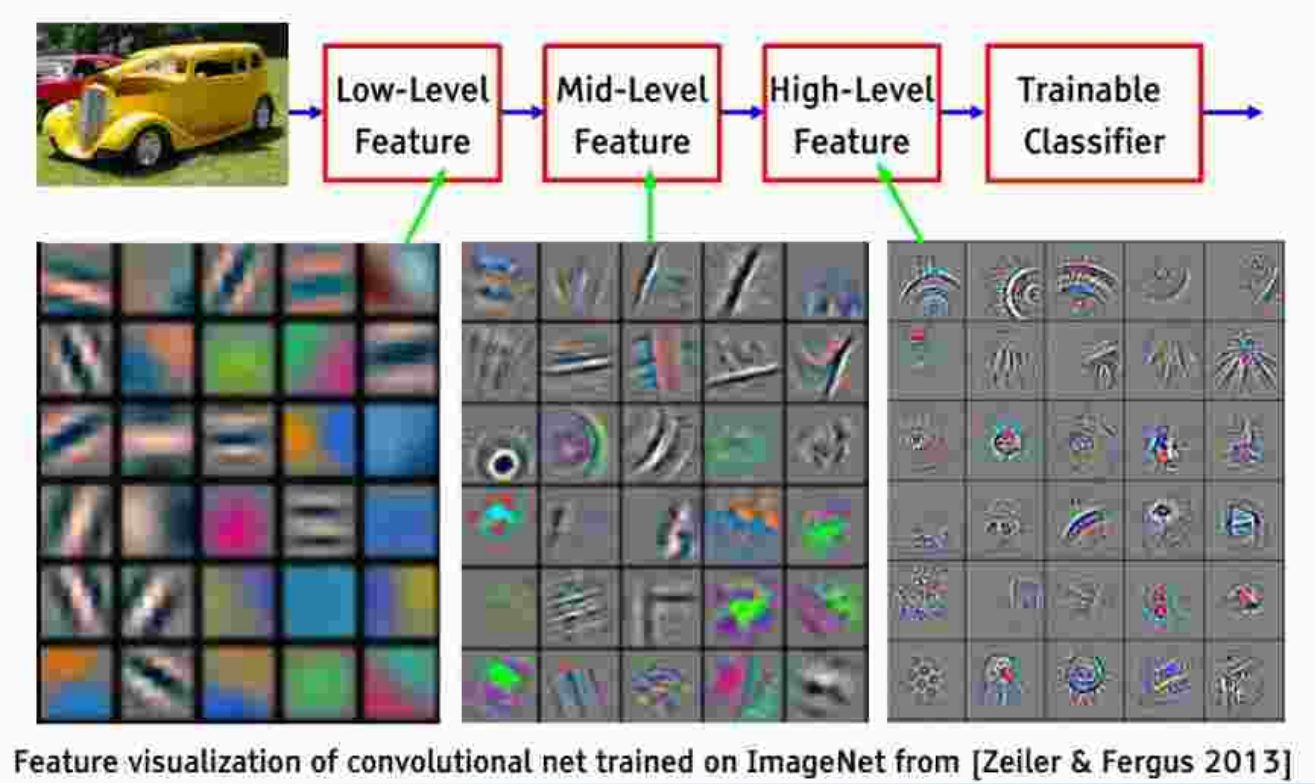
[Ioffe and Szegedy, 2015]

CNNs



[LeCun et al. 1998]

Hierarchical features



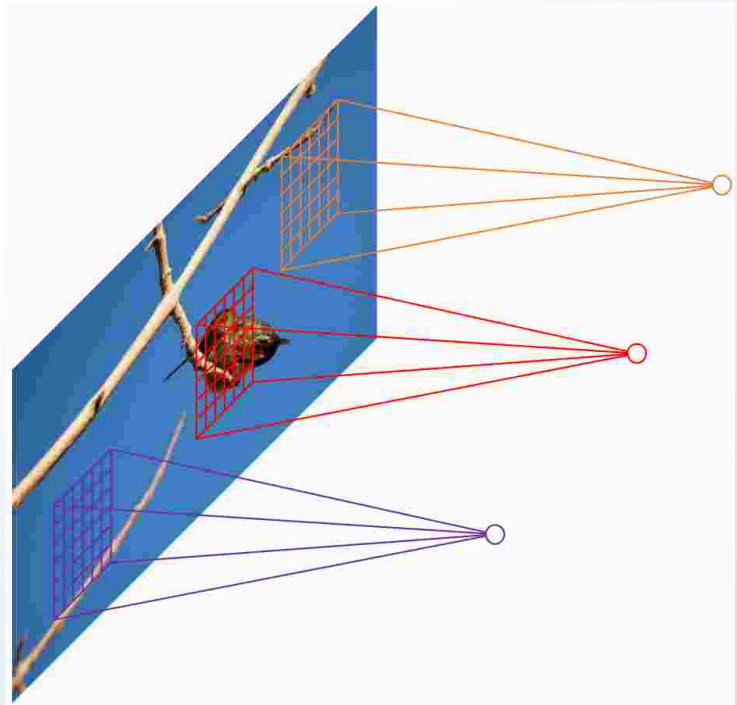
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

[Source: Y. LeCun]

Convolutions

Advantage of the convolution

- Locally connected
- Translation-invariant
- Position explicitly encoded
- Independent of input size



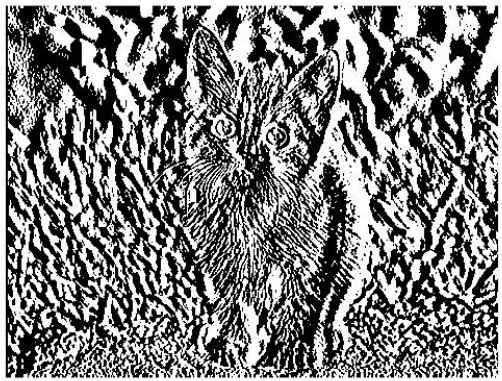
[He, ICCV15 tutorial]

Convolutions

Convolution is defined as

$$g(x,y) = w(x,y) * f(x,y) = g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

- Learn image filters w(x,y) to detect automatically relevant features



Convolutions

Convolution is defined as

$$g(x,y) = w(x,y) * f(x,y) = g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

f =

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

g =

24	40	52	45
64	96	112	92
112	160	176	140
108	152	164	129

w =

1	2	1
2	4	2
1	2	1

Convolutions

Convolution is defined as

$$g(x,y) = w(x,y) * f(x,y) = g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

f =

0	0	0	0	0	0
0	1	2	3	4	0
0	5	6	7	8	0
0	9	10	11	12	0
0	13	14	15	16	0
0	0	0	0	0	0

g =

24	40	52	45
64	96	112	92
112	160	176	140
108	152	164	129

w =

1	2	1
2	4	2
1	2	1

Convolutions

Convolution is defined as

$$g(x,y) = w(x,y) * f(x,y) = g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

f =

0	0	0	0	0	0
0	1	2	3	4	0
0	5	6	7	8	0
0	9	10	11	12	0
0	13	14	15	16	0
0	0	0	0	0	0

g =

24	40	52	45
64	96	112	92
112	160	176	140
108	152	164	129

w =

1	2	1
2	4	2
1	2	1

Convolutions

Convolution is defined as

$$g(x,y) = w(x,y) * f(x,y) = g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

f =

0	0	0	0	0	0
0	1	2	3	4	0
0	5	6	7	8	0
0	9	10	11	12	0
0	13	14	15	16	0
0	0	0	0	0	0

g =

24	40	52	45
64	96	112	92
112	160	176	140
108	152	164	129

w =

1	2	1
2	4	2
1	2	1

Convolutions

Convolution is defined as

$$g(x,y) = w(x,y) * f(x,y) = g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

f =

0	0	0	0	0	0
0	1	2	3	4	0
0	5	6	7	8	0
0	9	10	11	12	0
0	13	14	15	16	0
0	0	0	0	0	0

g =

24	40	52	45
64	96	112	92
112	160	176	140
108	152	164	129

w =

1	2	1
2	4	2
1	2	1

Convolutions

Convolution is defined as

$$g(x,y) = w(x,y) * f(x,y) = g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

f =

0	0	0	0	0	0
0	1	2	3	4	0
0	5	6	7	8	0
0	9	10	11	12	0
0	13	14	15	16	0
0	0	0	0	0	0

g =

24	40	52	45
64	96	112	92
112	160	176	140
108	152	164	129

w =

1	2	1
2	4	2
1	2	1

Convolutions

Convolution is defined as

$$g(x,y) = w(x,y) * f(x,y) = g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

f =

0	0	0	0	0	0
0	1	2	3	4	0
0	5	6	7	8	0
0	9	10	11	12	0
0	13	14	15	16	0
0	0	0	0	0	0

g =

24	40	52	45
64	96	112	92
112	160	176	140
108	152	164	129

w =

1	2	1
2	4	2
1	2	1

Pooling

- Mean and max pooling
- Larger receptive field
- Overlapping/Nonoverlapping
- Downsampeling feature representation

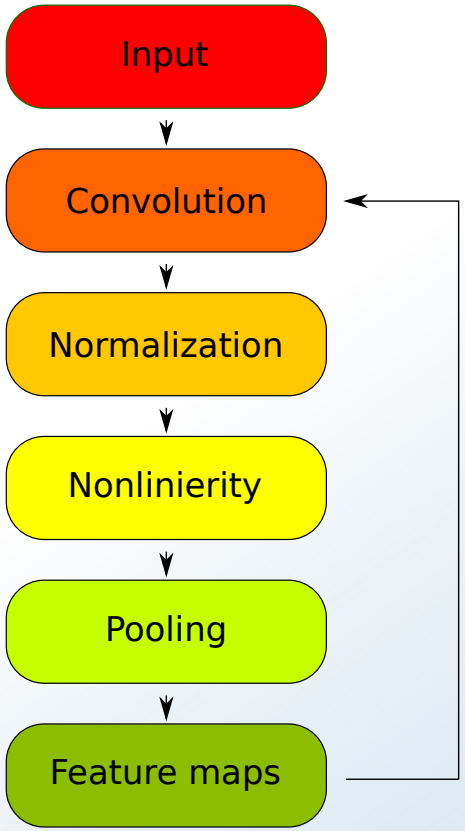
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

$\xrightarrow[\text{stride 2}]{2 \times 2 \text{ pool}}$

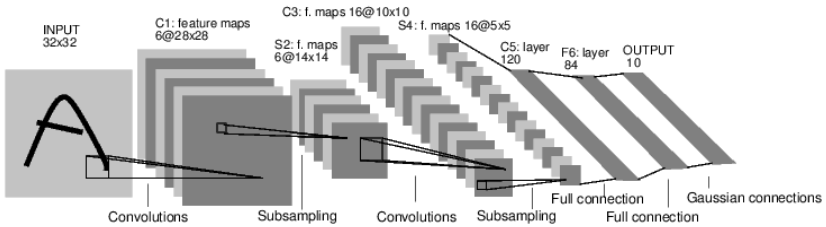
6	8
14	16

Architecture - Layer components

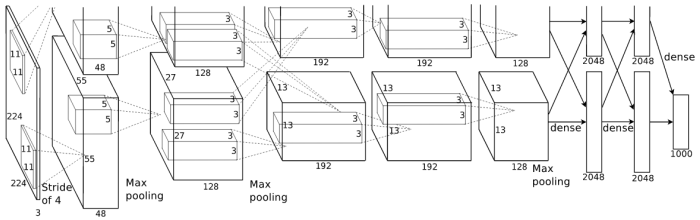
CNNs consist of several layers with these components



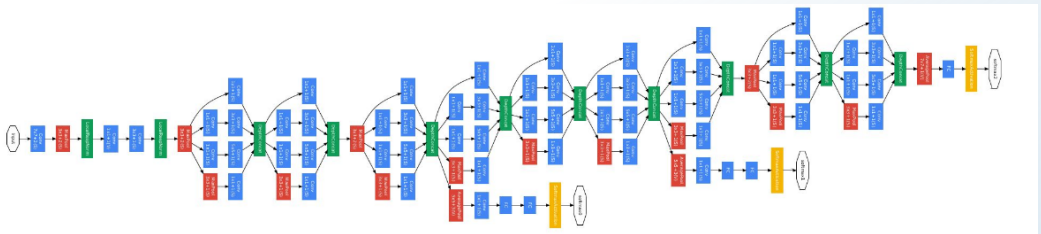
Increasing depth



[LeCun et al. 1998]



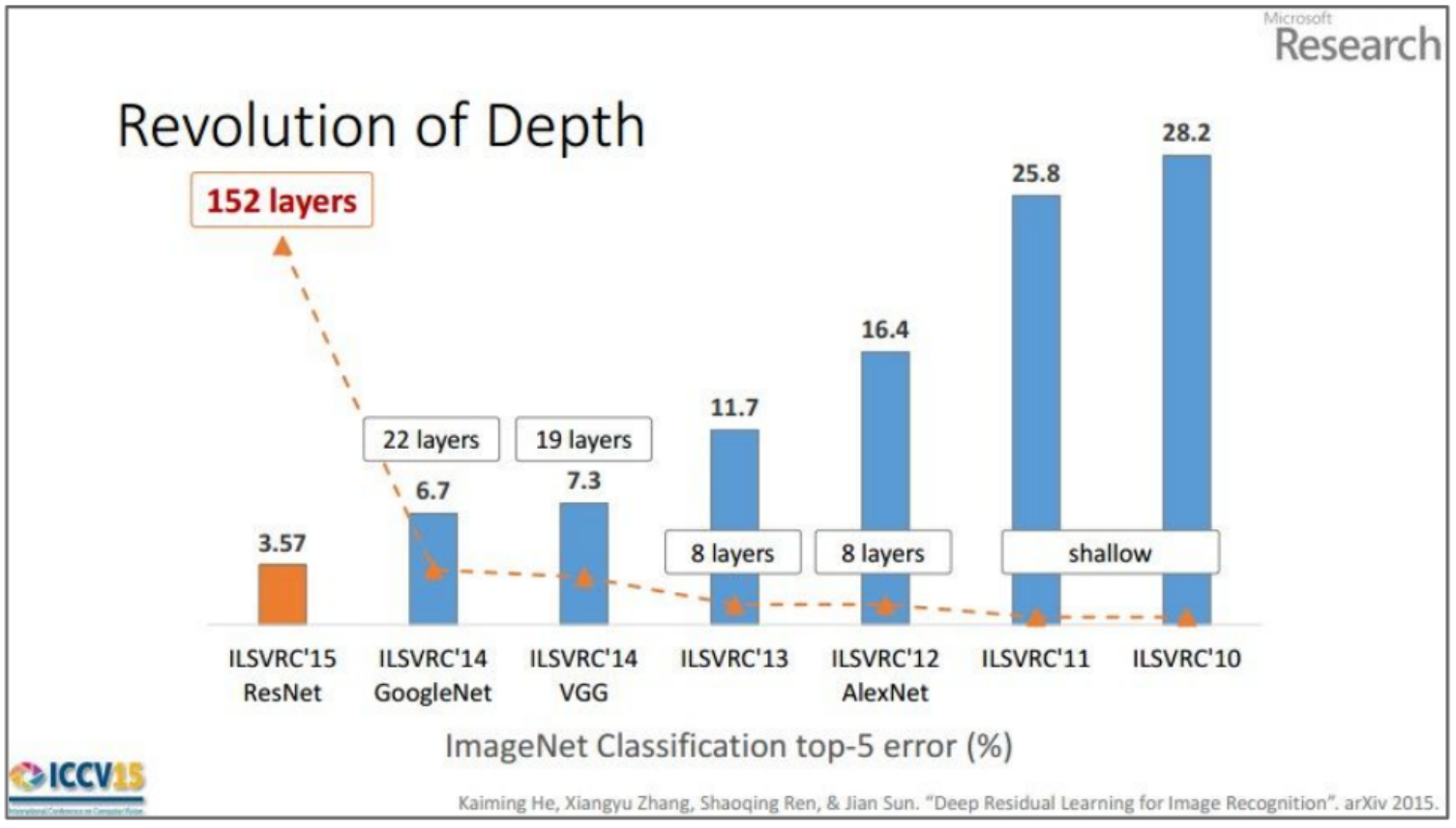
[Krizhevsky et al. 2012]



[Szegedy et al. 2015]

[He et al. 2015]

Increasing depth - ImageNet



[He, ICML 2016 Tutorial]

Practical tips

CNNs for small datasets:

- Data augmentation
- Transfer learning

Architecture:

- Small filters

Data augmentation

Common augmentations:

- Flip
- Rotate
- Random crops
- Jitter
  - Add noise
  - Change contrast
  - Move slightly along principle components of RGB colorspace

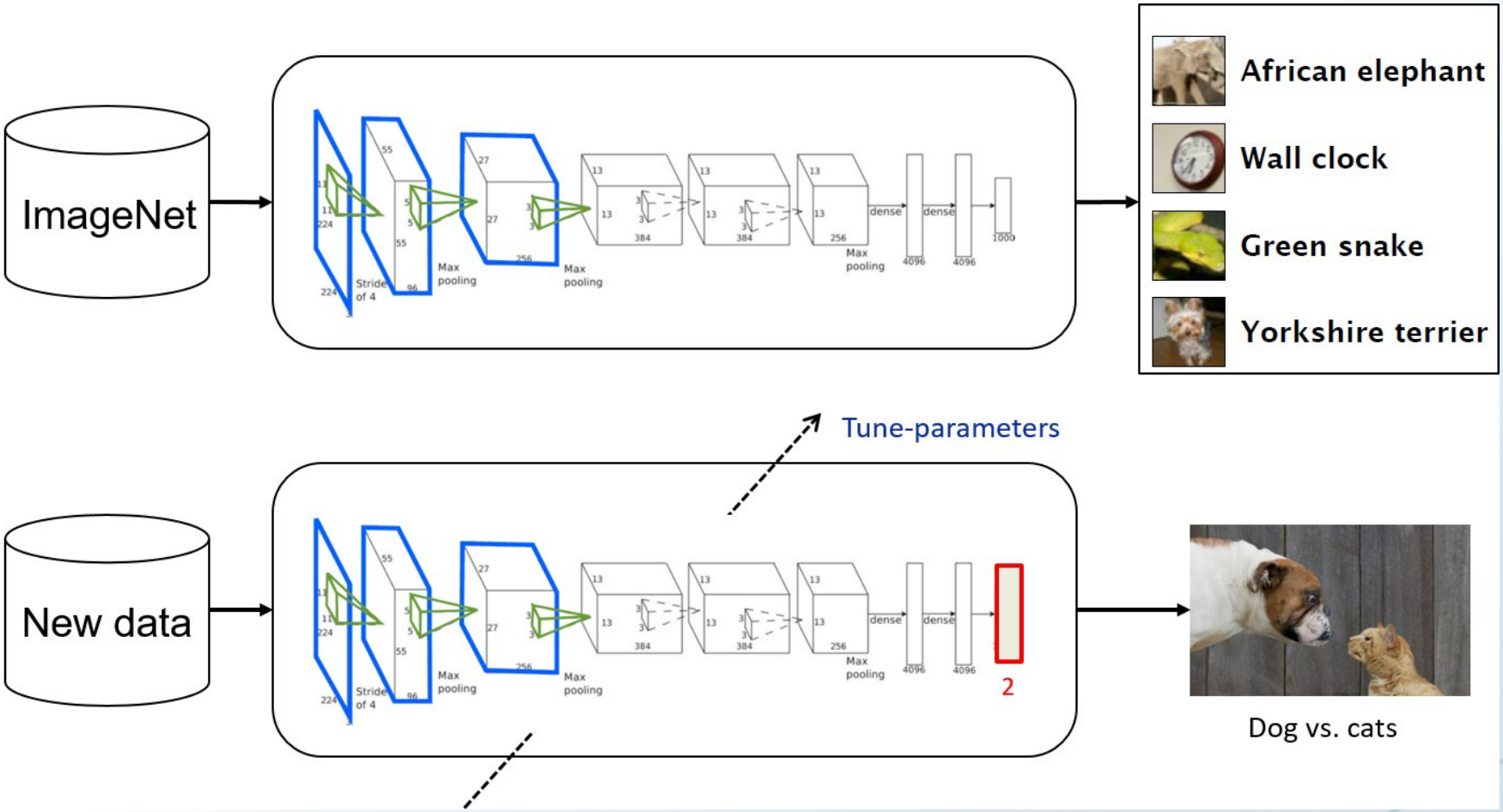
Transfer learning

Training CNNs on tasks without massive datasets is common practice
Several approaches:

- Perform unsupervised training on a large unlabeled dataset, and fine-tune with labelled data
- Pre-train on a large dataset, and fine-tune to the new data
- Pre-train on a large dataset, extract the features and classify the new data using your favorite classifier

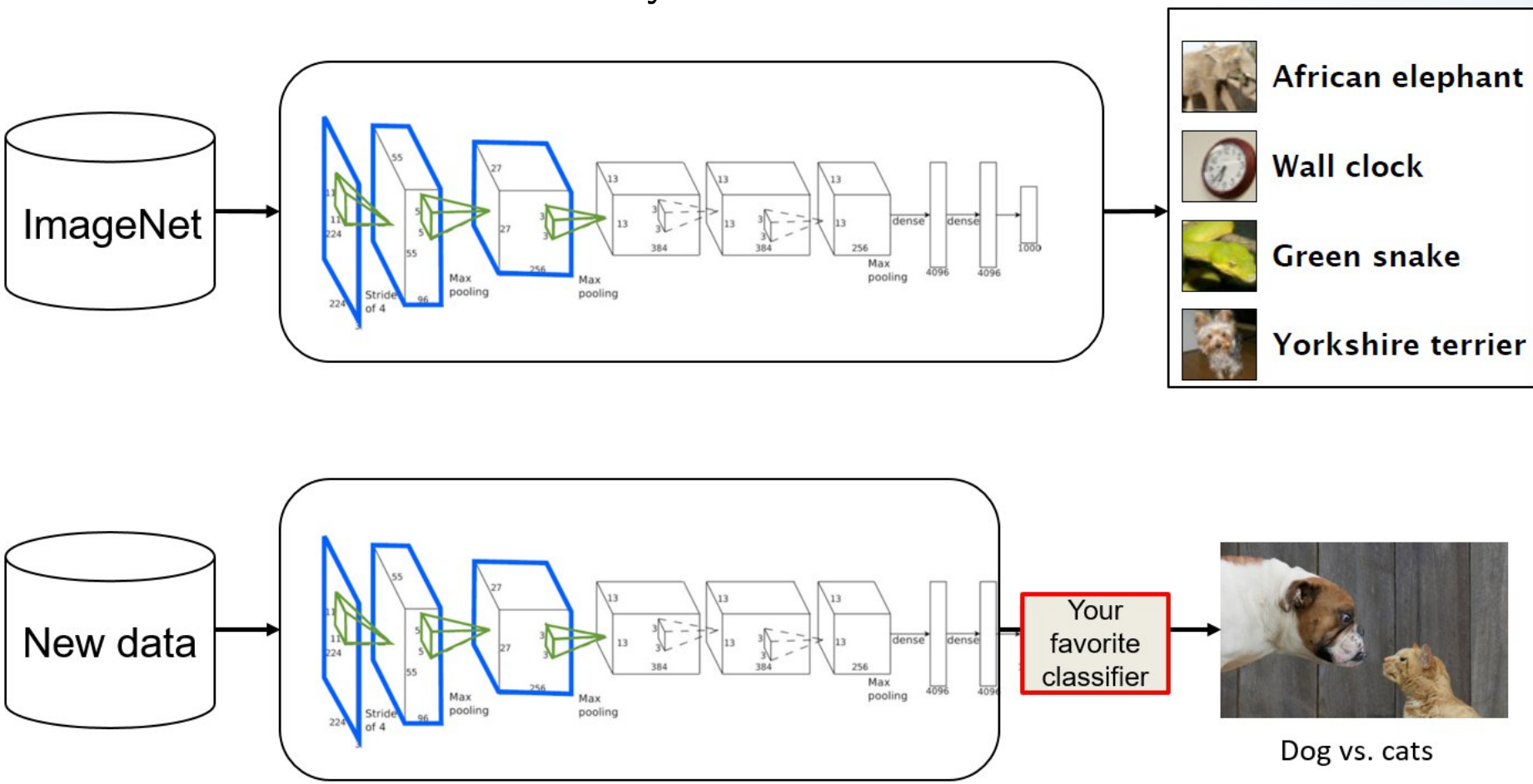
Transfer learning - Medium dataset

Take a pre-trained model and fine-tune to new tasks



Transfer learning - Small dataset

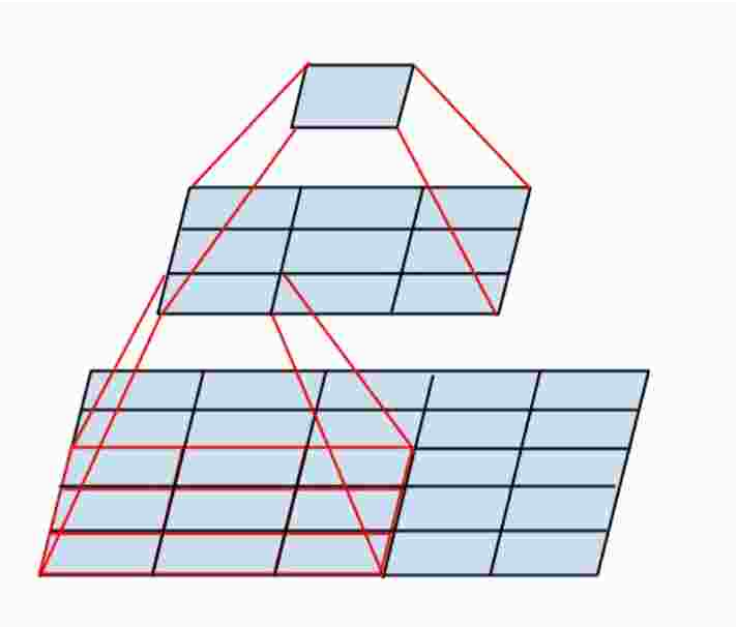
Extract the features and classify with favorite classifier



Small filters

What size filter to choose:

- Very common 3×3
- Larger receptive fields can be represented by small filters



[Szegedy et al., 2015]

Small filters

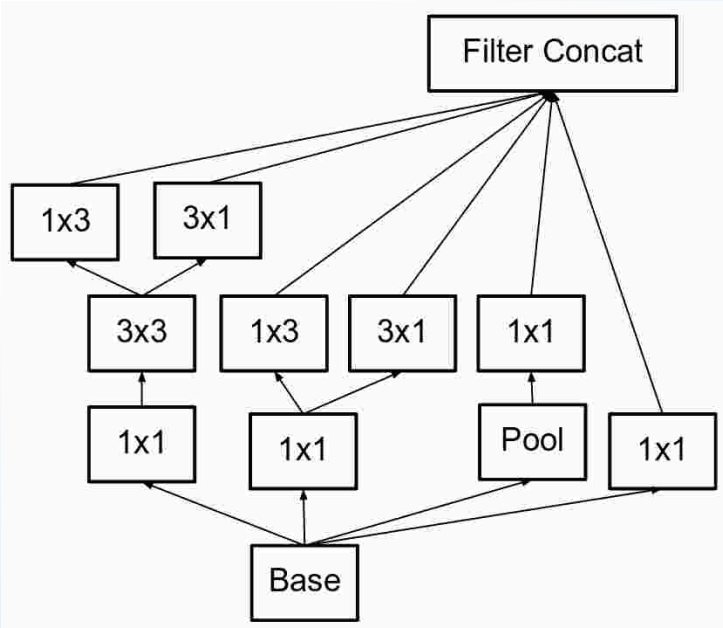
- More efficient
- More nonlinearity

5x5 conv Weights:
$C \times (5 \times 5 \times C) = 25C^2$

$2 \times$ (3x3) conv Weights:
$2 \times C \times (3 \times 3 \times C) = 18C^2$

Small filters

- Can go even smaller
- 1×3 conv and 3×1 conv

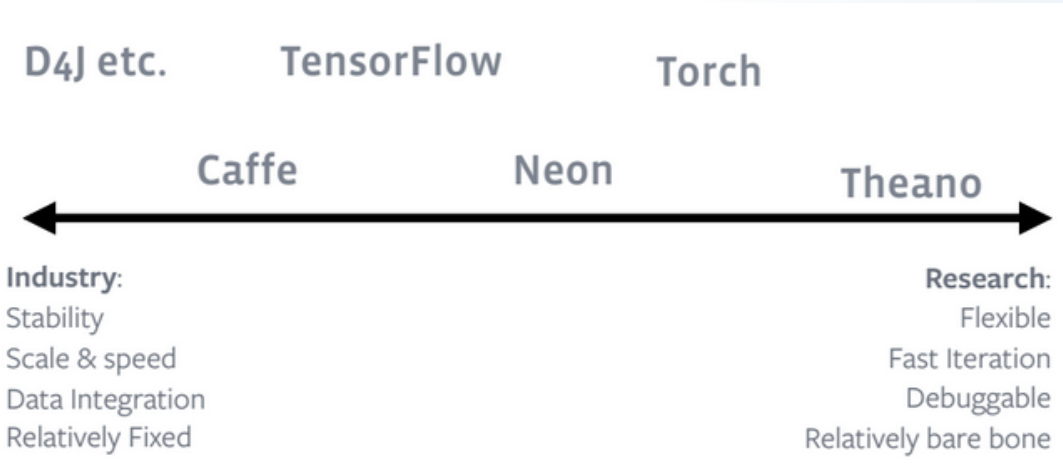


[Szegedy et al., 2015]

Software

Many software alternatives:

- Torch
- Caffe
- Theano
- Tensorflow
- Neon
- Keras



[Source: Alex Wiltschko]

Software

Caffe:

- + Fast
- + Feedforward
- + Finetuning
- + Easy to get started
- + Great model zoo
- - RNN
- - Extensibility (CUDA/C++)

Torch:

- + Extensibility
- + Model zoo
- - RNN
- - Lua

Software

Theano:

- + Python
- + Good abstraction
- + RNN
- + Extensibility
- - Pretrained models
- - Debugging

Tensorflow:

- + Python
- + Good abstraction
- + RNN
- + Best parallelism
- - Performance
- - Flexibility

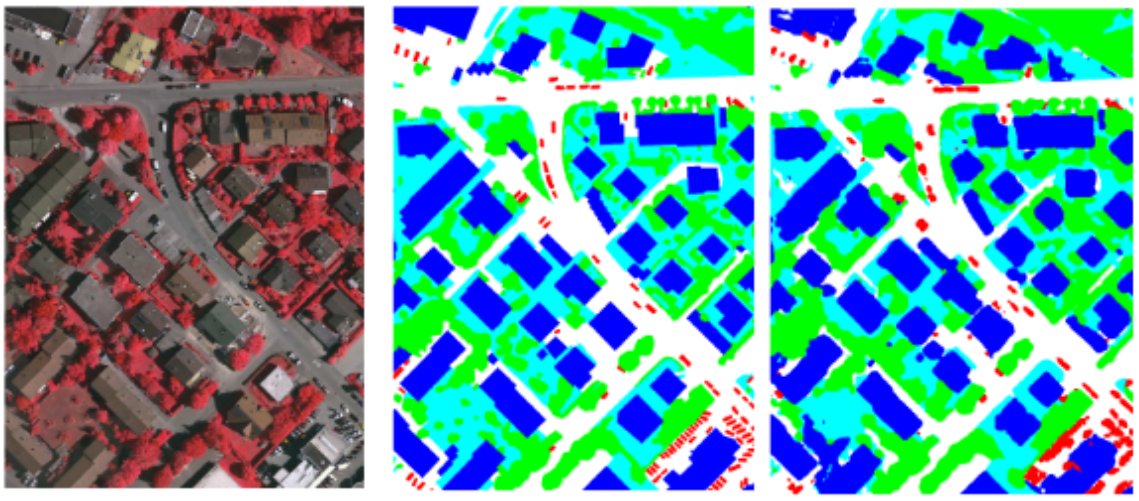
Caffe - Finetuning example

Possible to finetune network without writing code (using C++ api):

- Step 1: Download a pre-trained model from the model zoo. [https://github.com/BVLC/caffe/wiki/Model-Zoo]
- Step 2: Modify .prototxt and define solver.prototxt (Nice visualization http://ethereon.github.io/netscope/quickstart.html)
- Step 3: Run ./build/tools/caffe train -solver vggModel/solver.prototxt -weights vggModel/VGG_CNN_S.caffemodel -gpu 0

Segmentation using CNNs

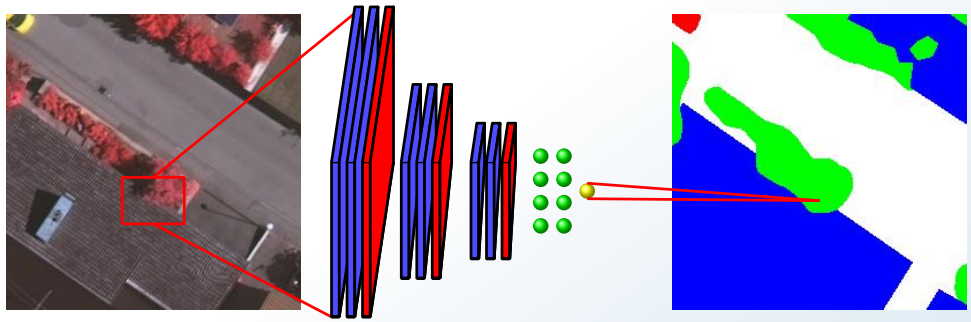
- Pixel-wise classification
- Want end-to-end learnable architecture
- Less sensitive than traditional segmentation methods



[Kampffmeyer et al. 2016]

Segmentation using CNNs - Patch based

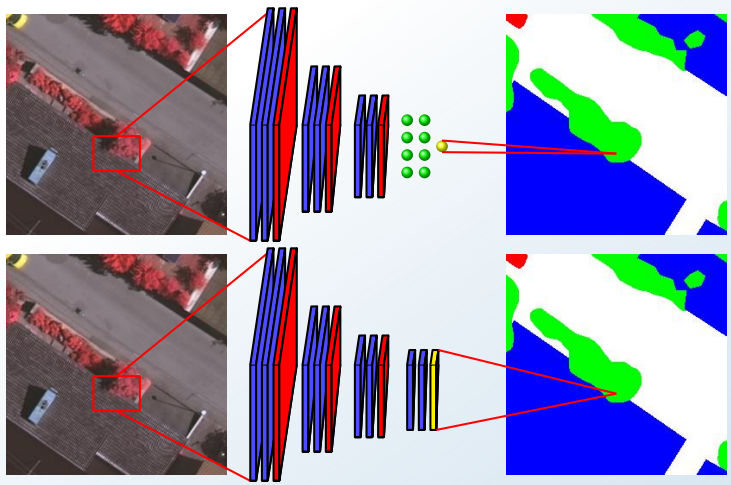
- Patch-based approach
- Very intuitive
- Very computationally expensive



Segmentation using CNNs - Patch based

Replace fully connected layer with convolutions [Sermanet et al., 2013]

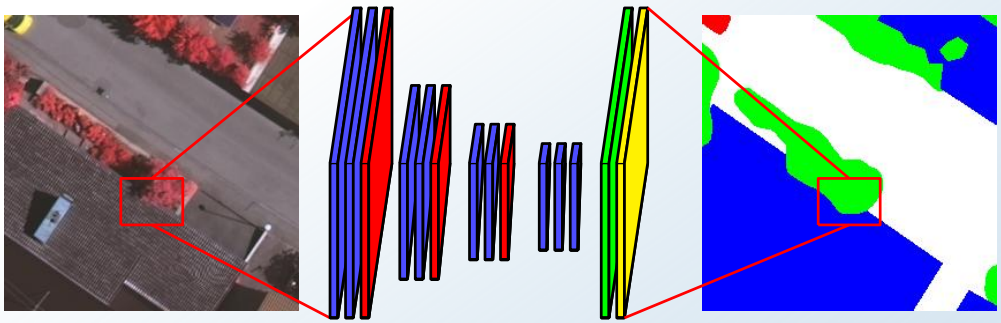
- More efficient
- Not dependent on image size



Segmentation using CNNs - Fully convolutional

Learn an upsampeling from feature representation back to pixel space [Long et al. 2015]

- Efficient
- Not dependent on image size
- End-to-end learning on whole images
- Better accuracy

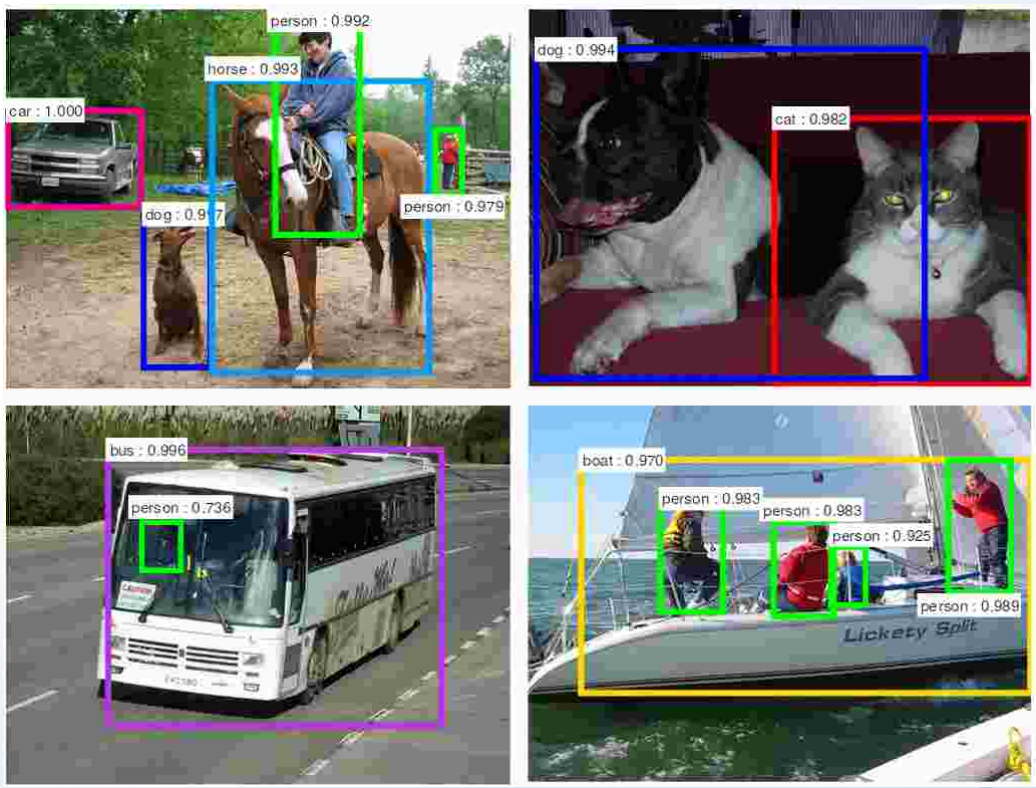


Object detection

Two main approaches

- Region Proposals
- Regression

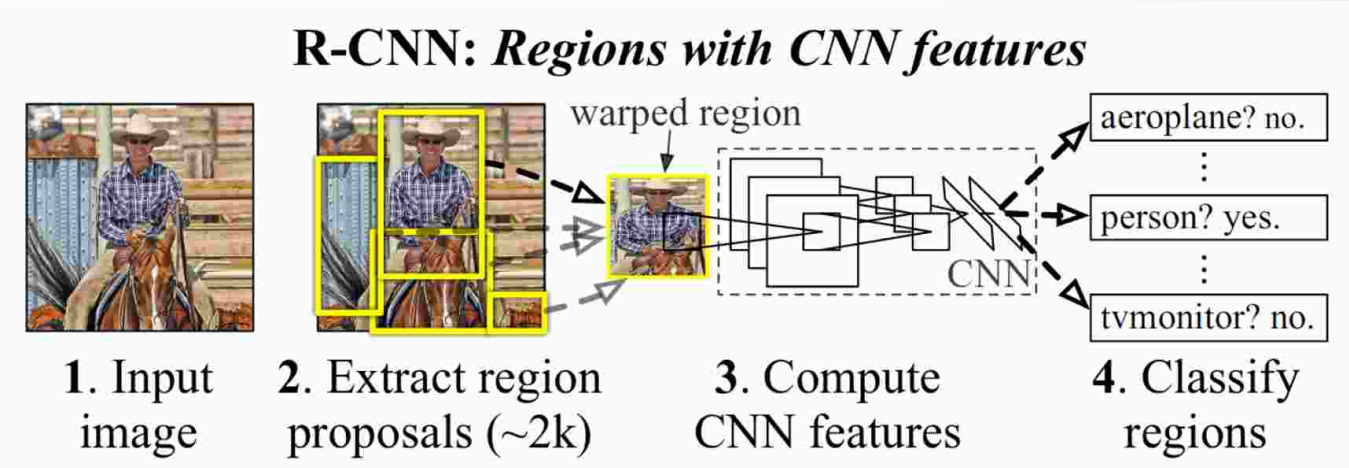
[Ren et al., 2016]



Object detection - Region Proposals - RCNN

RCNN [Girshick et al. 2014]

- Region proposal (e.g. selective search)
- Classify regions
- Computationally expensive
- Non-maximum suppression

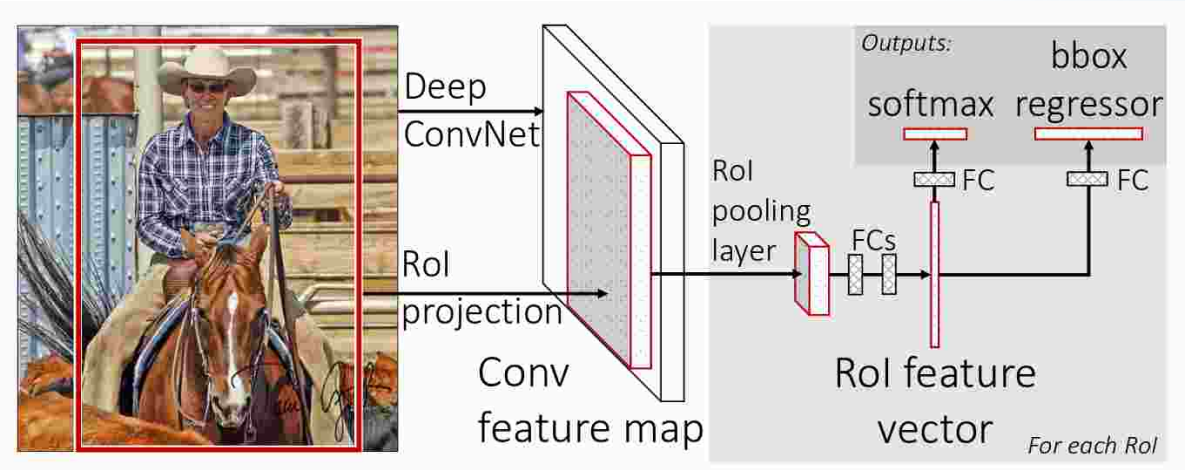


[Girshick et al. 2014]

Object detection - Region Proposals - Fast RCNN

Fast RCNN [Girshick, 2015]

- Use fully convolutional idea for efficiency
- Bounding box regression offsets
- Faster and better accuracy

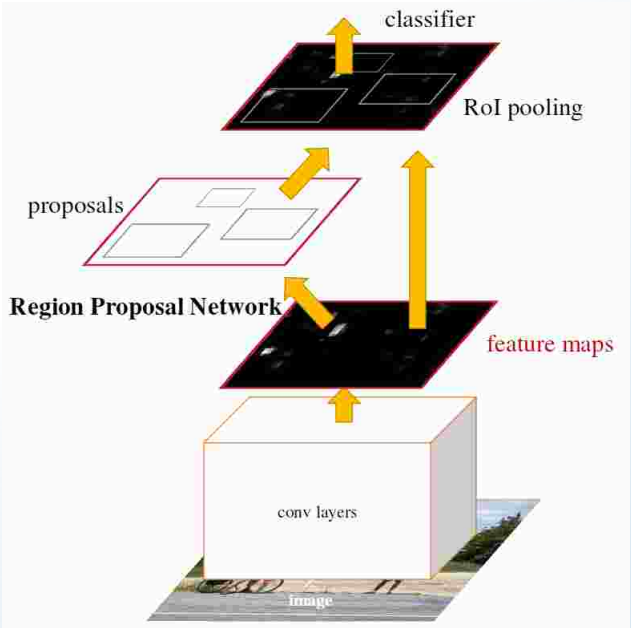


[Girshick, 2015]

Object detection - Region Proposals - Faster RCNN

Faster RCNN [Ren et al., 2016]

- Region proposal network
- Faster and improved overall accuracy



[Ren et al., 2016]

Take away message

- CNNs are powerful models
- State of the art on many tasks
- Don't require large datasets