Deep Learning

Lecture 4 – Convolutional Neural Networks 1

UPPSALA
UNIVERSITET

Joakim Lindblad
Department of Information Technology
Uppsala University

joakim.lindblad@it.uu.se
cb.uu.se/~joakim

UPPSALA UNIVERSITET

Outline

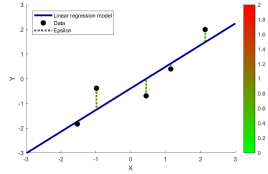
Further reads/links

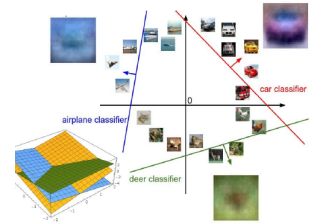
- Get going in MATLAB https://se.mathworks.com/help/nnet/examples/create-simple-deep-learning-network-for-classification.html
- Stanford CS231n deep learning course by Fei Fei's group, 2016 version (skip to 2nd lecture, w. Andrej Karpathy) https://www.youtube.com/watch?v=g-PvXUjD6qg&list=PL1Jy-eBtNFt6EuMxFYRiNRS07MCWN5UIA&index=1 2017 version https://www.youtube.com/playlist?list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv
- Machine learning by Andrew Ng (Coursera) https://www.youtube.com/playlist?list=PLZ9qNFMHZ-A4rycgrgOYma6zxF4BZGGPW http://deeplearning.stanford.edu/tutorial/
- Recent deep learning summer school in Toronto http://videolectures.net/DLRLsummerschool2018_toronto/
- Ian Goodfellows book on deep learning http://www.deeplearningbook.org/
- Stat212b: Topics Course on Deep Learning http://joanbruna.github.io/stat212b/
- Yann LeCun's "Gradient-based learning applied to document recognition" http://ieeexplore.ieee.org/document/726791/?arnumber=726791
- An overview of gradient descent optimization algorithms http://ruder.io/optimizing-gradient-descent/
- WILDML http://www.wildml.com/
- Deep Learning Glossary http://www.wildml.com/deep-learning-glossary/
- Colah's blog http://colah.github.io/
- https://icml.cc/Conferences/2018/Schedule?type=Tutorial, https://icml.cc/Conferences/2017/Tutorials
- https://arxiv.org & http://www.arxiv-sanity.com/
- The annual report of https://aiindex.org/
- And many many more ... (Google is your friend :-))

Recap

Recap

- Linear regression $y = Wx + b$
  - Maximum likelihood $\rightarrow$ Least squares optimization problem: $\widehat{w}, \widehat{b} = \underset{w,b}{\operatorname{argmin}} \sum_{i=1}^{n} (wx_i + b - y_i)^2$
- A linear classifier $y = f(Wx + b)$ – *Logistic regression*
  - Multi-class case encoding as a "one hot" vector
- Send the output through a nonlinearity (activation function) $y = f(Wx + b)$
- Send the output to another classifier, and another... $y = f(W^{(3)} f(W^{(2)} f(W^{(1)} x + b^{(1)}) + b^{(2)}) + b^{(3)})$ = A *Neural network*
- Measure performance with a *loss function*
- Use Stochastic Gradient Descent (SGD) to minimize the loss
  - Take small steps in the direction of the negative gradient $\theta_{k+1} = \theta_k - \lambda \cdot \nabla_\theta L$, where stepsize (a.k.a. learning rate) λ typically in the range $[0.001, 0.01]$





Recap – Linear models

Linear regression

Output

$y_i \in \mathbb{R}$

Model

$\hat{y}_i = \mathbf{w}^\mathsf{T}\mathbf{x}_i + b$

Loss

$L_i = (y_i - \hat{y}_i)^2$

Logistic regression

Output

$y_i \in \{0, 1\}$

Model

$p_i = \mathsf{Pr}(y_i = 1|\mathbf{x}_i) = \frac{e^{\mathbf{w}^\mathsf{T}\mathbf{x}_i + b}}{1 + e^{\mathbf{w}^\mathsf{T}\mathbf{x}_i + b}}$

Loss

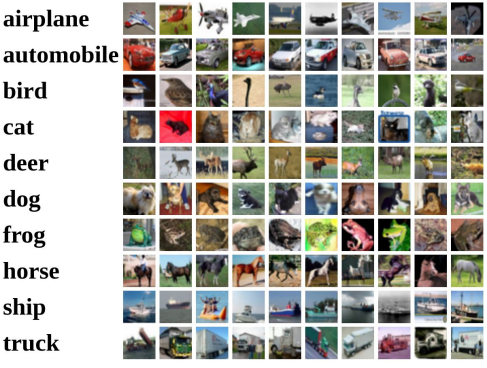
$L_i = -y_i \ln(p_i) - (1 - y_i) \ln(1 - p_i)$

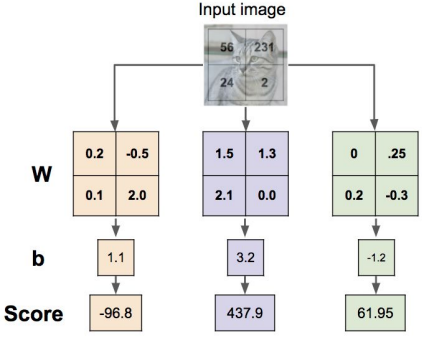
We find $\mathbf{w}$ and b by minimizing sum of the losses

$$\widehat{\mathbf{w}}, \widehat{b} = \underset{\mathbf{w}, b}{\mathsf{argmin}} \frac{1}{n} \sum_{i=1}^{n} L_i$$

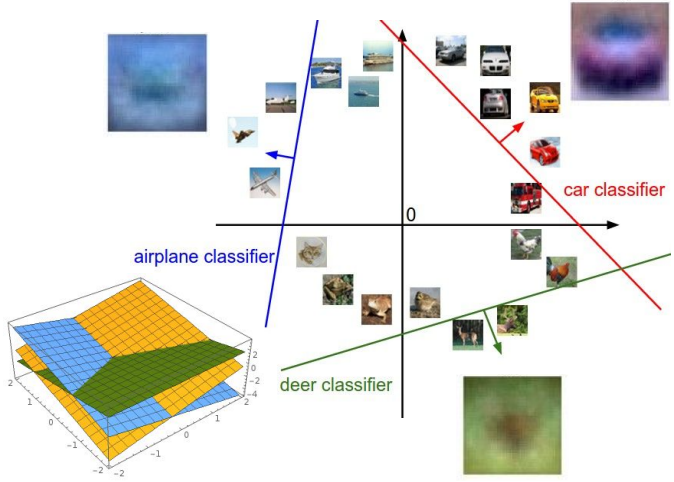
- For linear regression the problem can be solved analytically.
- For logistic regression we need to use numerical optimization.

Interpreting a Linear Classifier: Visual Viewpoint





Interpreting a Linear Classifier: Geometric Viewpoint



$$f(x,W) = Wx + b$$

Array of **32x32x3** numbers
(3072 numbers total)

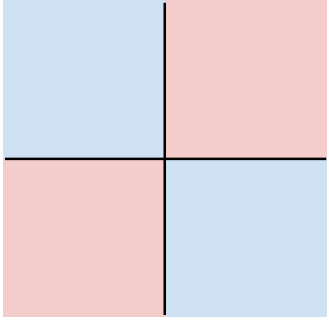
Plot created using Wolfram Cloud

Cat image by Nikita is licensed under CC-BY 2.0

Hard cases for a linear classifier

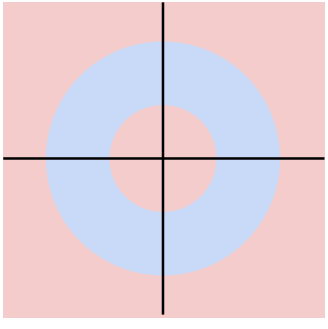
Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants



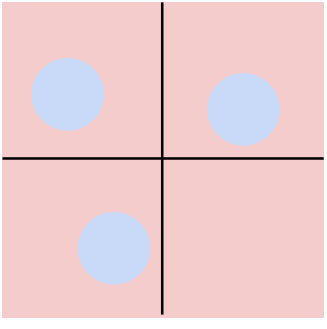
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



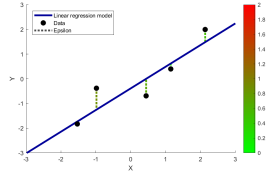
Class 1:
Three modes

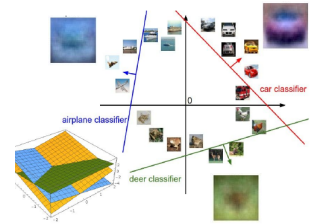
Class 2:
Everything else



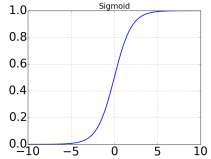
Recap

- Linear regression $y = Wx + b$
  - Maximum likelihood $\rightarrow$ Least squares optimization problem: $\widehat{w}, \widehat{b} = \underset{w,b}{\operatorname{argmin}} \sum_{i=1}^{n} (wx_i + b - y_i)^2$
- A linear classifier $y = f(Wx + b)$ – *Logistic regression*
  - Multi-class case encoding as a "one hot" vector
- Send the output through a nonlinearity, *activation function* $y = f(Wx + b)$
- Send the output to another classifier, and another... $y = f(W^{(3)} f(W^{(2)} f(W^{(1)} x + b^{(1)}) + b^{(2)}) + b^{(3)})$ = A *Neural network*
- Measure performance with a *loss function*
- Use Stochastic Gradient Descent (SGD) to minimize the loss
  - Take small steps in the direction of the negative gradient $\theta_{k+1} = \theta_k - \lambda \cdot \nabla_\theta L$, where stepsize (a.k.a. learning rate) λ typically in the range $[0.001, 0.01]$

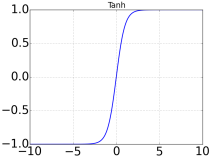




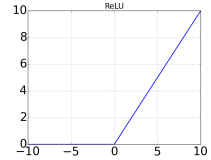
Recap – Activation functions $\sigma(x)$



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



$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2sigmoid(2x) - 1$$



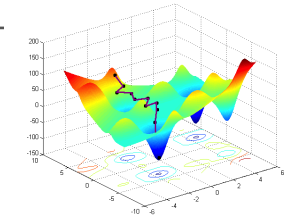
$$ReLU(x) = max(0, x)$$

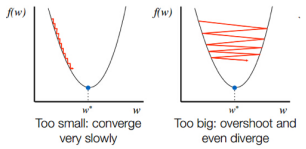
Recap

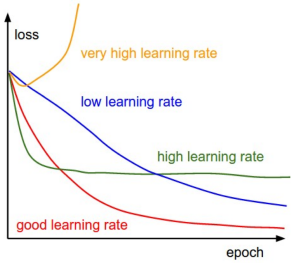
- A neural network is a sequential construction of several generalized linear regression models.

$$\mathbf{h}^{(0)} = \mathbf{x}$$
$$\mathbf{h}^{(l)} = \sigma\left(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}\right), \quad l = 1, \ldots, L-1$$
$$\hat{y} = \mathbf{W}^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)}$$

- All weight matrices and offset vectors in all layers combined are the parameters of the network
  $\boldsymbol{\theta} = \left[\text{vec}(\mathbf{W}^{(1)})^\mathsf{T}, \quad \mathbf{b}^{(1)\mathsf{T}}, \quad \ldots, \quad \text{vec}(\mathbf{W}^{(L)})^\mathsf{T}, \quad \mathbf{b}^{(L)\mathsf{T}}\right]^\mathsf{T}$,
  which constitutes the parametric model $\hat{y} = f(\mathbf{x}; \boldsymbol{\theta})$.
- We train our models by considering the optimization problem $\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta})$
- Optimization using Stochastic Gradient Descent (SGD)







Recap – Minibatch SG

1. Initialize $\boldsymbol{\theta}_0$, set $k \leftarrow 1$, choose batch size n_b and number of epochs E.
2. For $i = 1$ to E
   (a) Randomly shuffle the training data $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{n}$.
   (b) For $j = 1$ to $\frac{n}{n_b}$
      (i) Approximate the gradient of the loss function using the current minibatch $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=(j-1)n_b+1}^{jn_b}$
      $\widehat{\mathbf{g}}_k = \frac{1}{n_b} \sum_{i=(j-1)n_b+1}^{jn_b} \nabla_{\boldsymbol{\theta}} L_i(\boldsymbol{\theta})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k} \approx \mathbf{g}_k.$
      (ii) Take a step in the gradient direction $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \epsilon \widehat{\mathbf{g}}_k$.
      (iii) Update the iteration index $k \leftarrow k + 1$.

At each iteration we get a stochastic approximation $\widehat{\mathbf{g}}_k$ of the true gradient

$$\mathbf{g}_k = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L_i(\boldsymbol{\theta})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k}$$

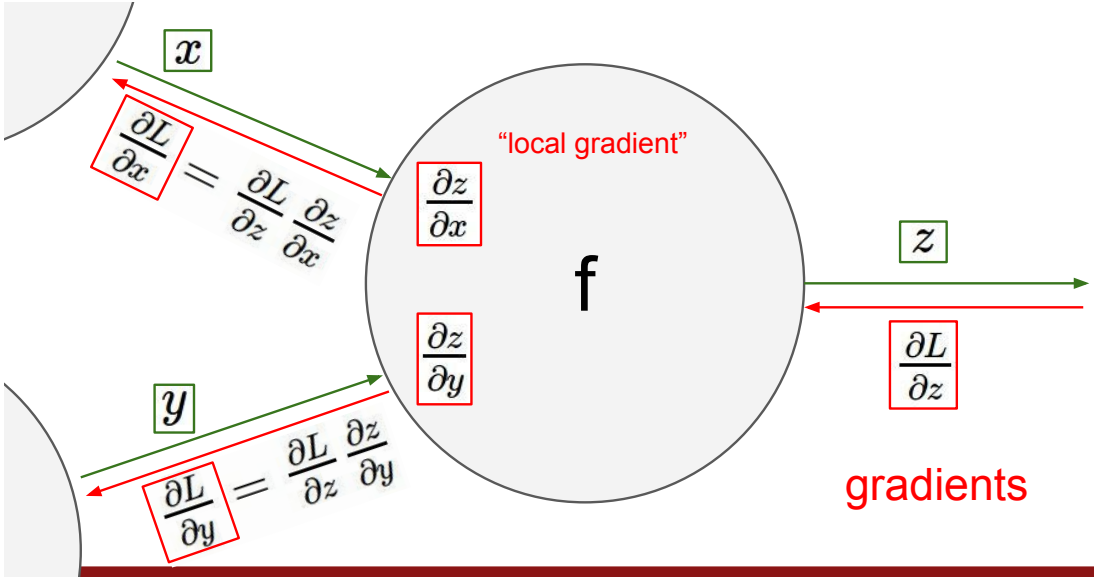
Recap – Back-propagation

Back-propagation - "Backprop" computing partial derivatives (e.g. in a *computational graph*) $\nabla_\theta L(\theta; \mathbf{x}) = (\frac{\partial L}{\partial \theta_1}, \frac{\partial L}{\partial \theta_2}, \ldots)$

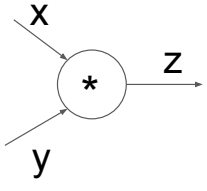
- Forward pass, backward pass, parameter update, and iterate...
- Follow the chain rule
- Find the analytical gradient of each component (incl. activation and loss functions)
- Total gradient is the product of the gradient at that node multiplied with the gradient coming to the node from the output side (propagating backwards)
- If connected to multiple nodes, add the gradients
- For ReLU activations gradient pass to positive outputs only
- For deep networks, very many values are multiplied together $\rightarrow$ problematic with vanishing or exploding gradients

Become a backprop ninja!

Yes you should understand backprop!
https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b


x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Modularized implementation: forward / backward API



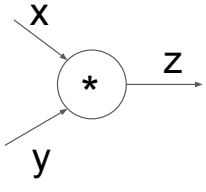
(x,y,z are scalars)

```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        return z
    def backward(dz):
        # dx = ... #todo
        # dy = ... #todo
        return [dx, dy]
```

$\frac{\partial L}{\partial z}$

$\frac{\partial L}{\partial x}$

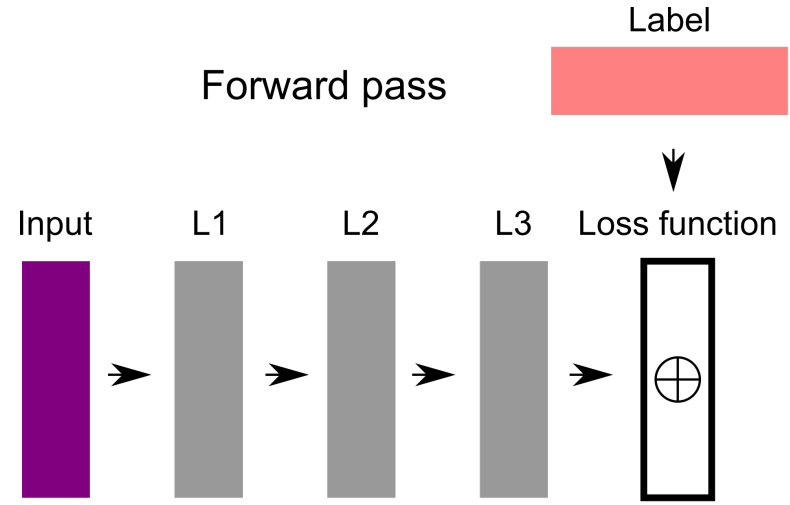
Modularized implementation: forward / backward API



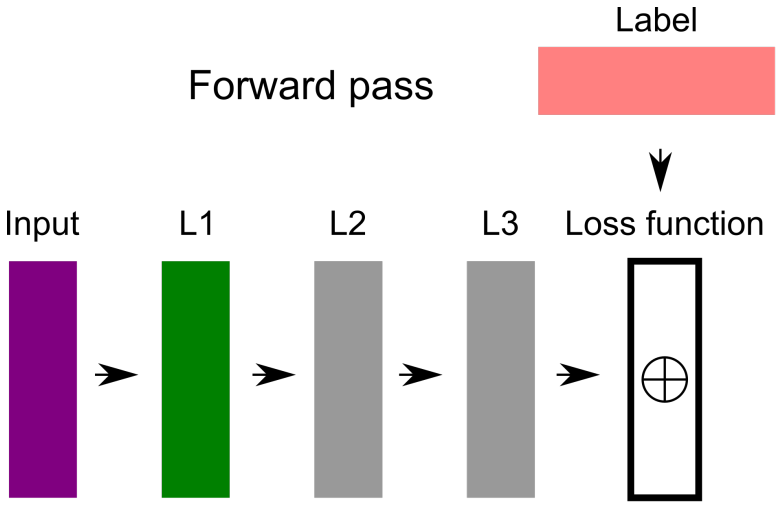
(x,y,z are scalars)

```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

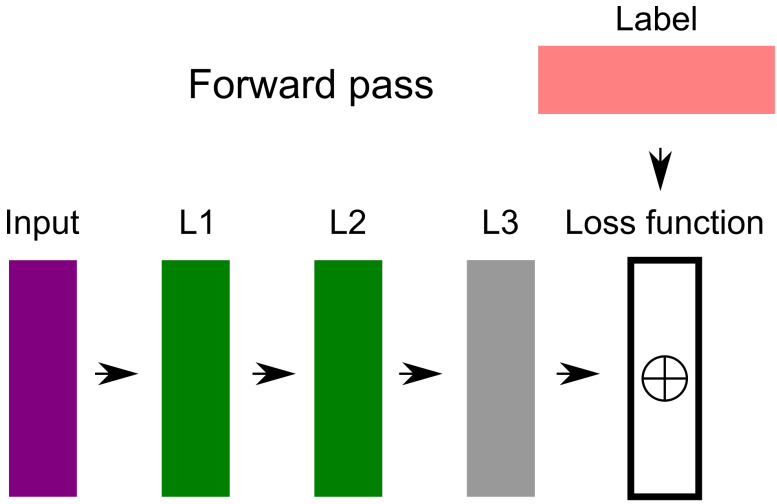
Forward pass



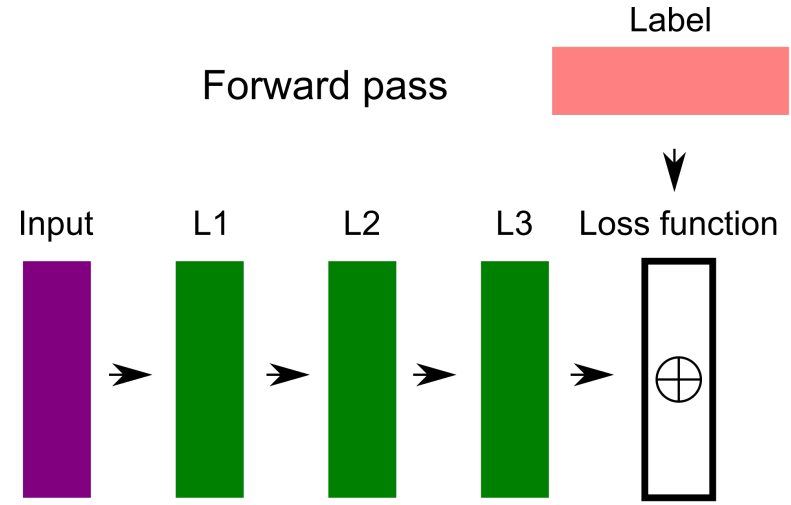
Forward pass



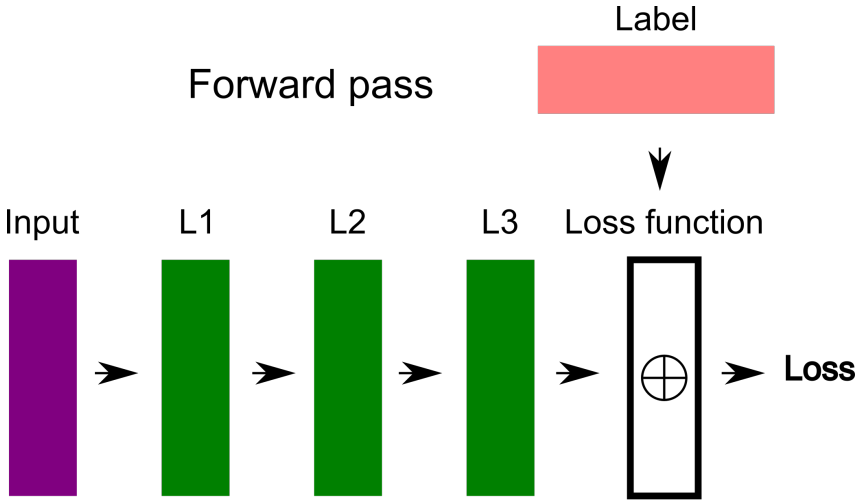
Forward pass



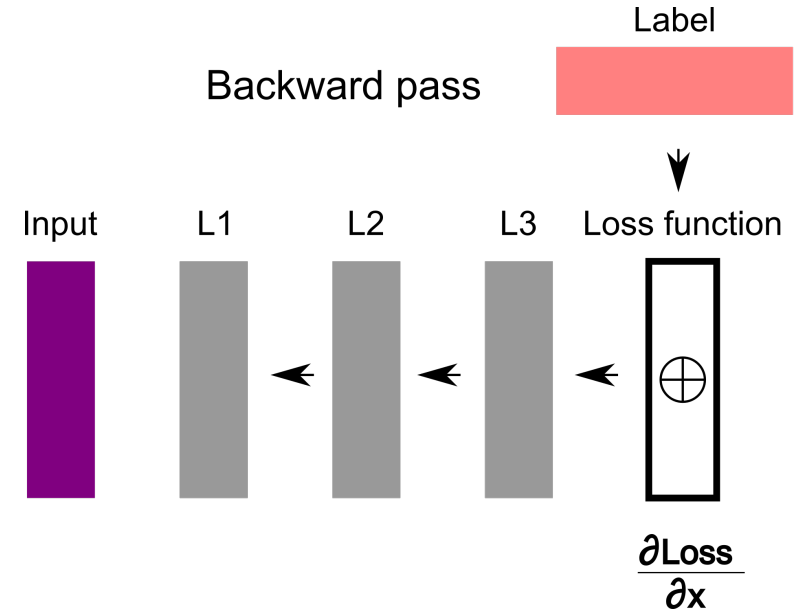
Forward pass



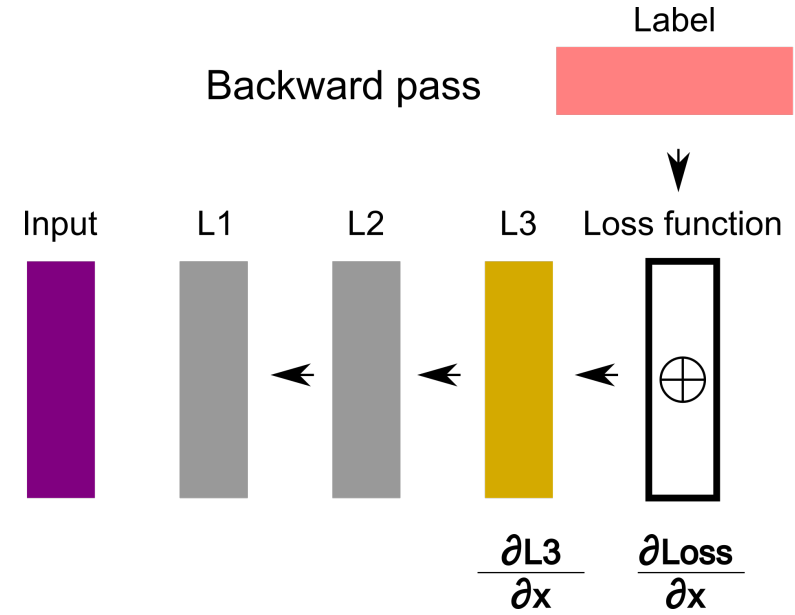
Forward pass



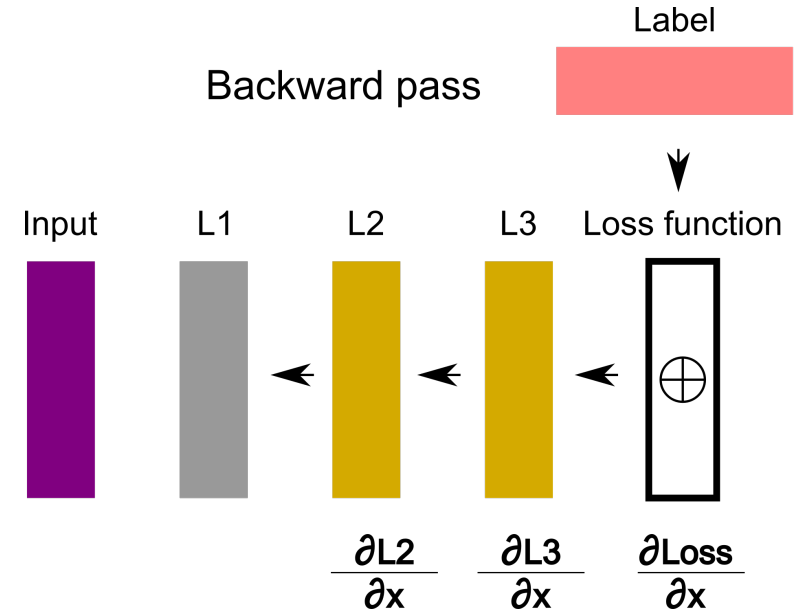
Backward pass



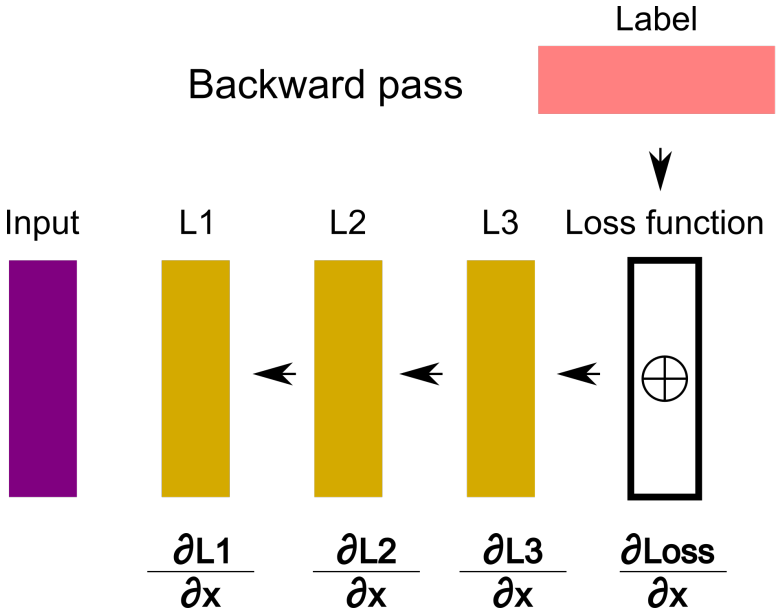
Backward pass



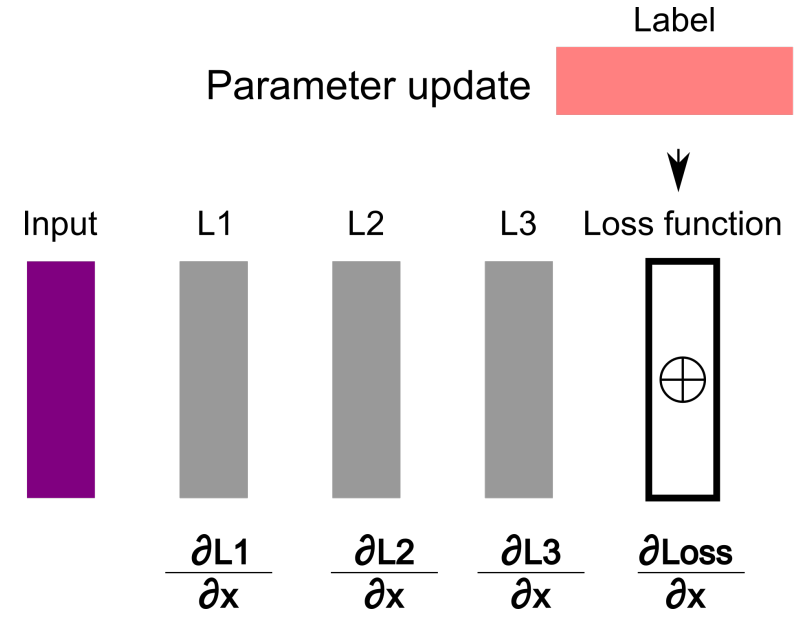
Backward pass



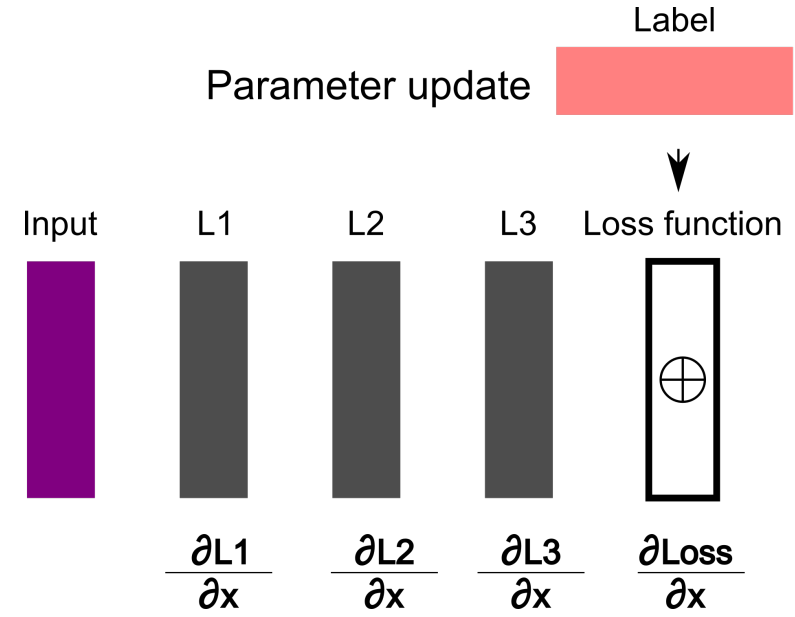
Backward pass



Backward pass

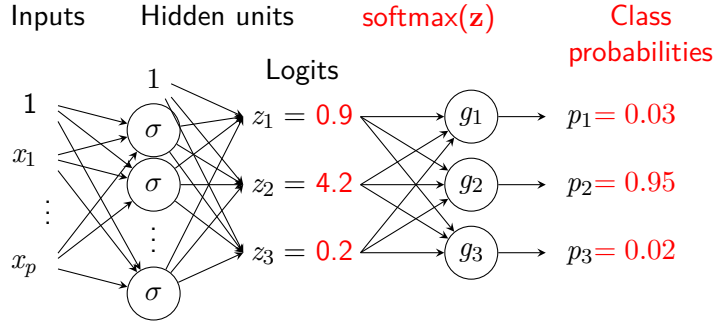


Backward pass



NN classification with $K = 3$ classes

Consider an example with three classes $K = 3$.

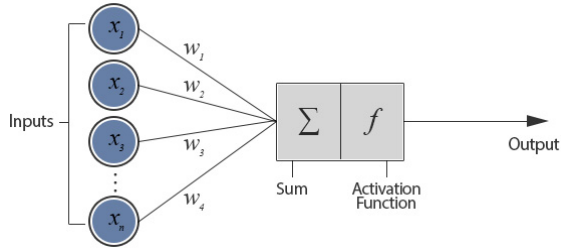


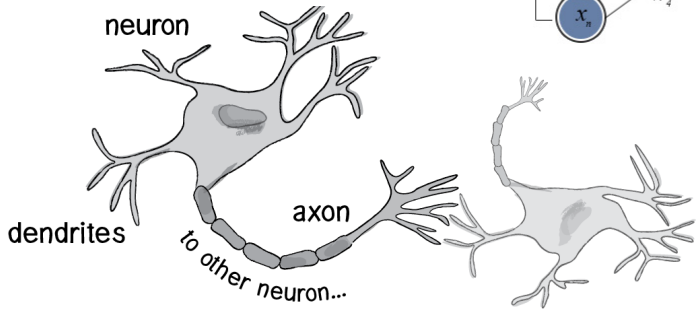
where

$$g_k(\mathbf{z}) = \frac{e^{z_k}}{\sum_{l=1}^{K} e^{z_l}}, \qquad \text{softmax}(\mathbf{z}) = [g_1(\mathbf{z}), \ldots, g_K(\mathbf{z})]^\mathsf{T}.$$

Deep convolutional neural networks

Artificial neuronal networks





What used to be seen as a deep neural network...

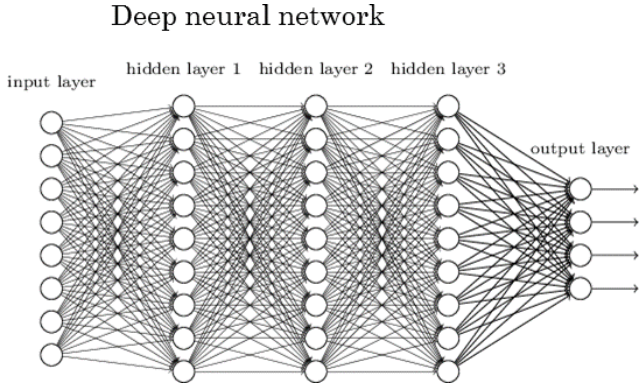


Figure: Fully connected Neural network

Src. http://www.rsipvision.com/exploring-deep-learning/

What used to be seen as a deep neural network...

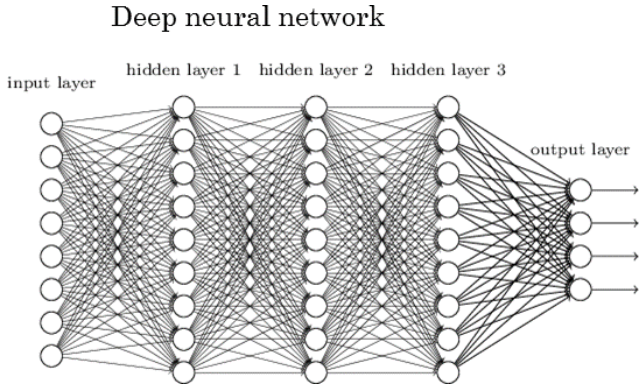


Figure: Fully connected Neural network

Src. http://www.rsipvision.com/exploring-deep-learning/

- The number of weights grows super fast!

What used to be seen as a deep neural network...

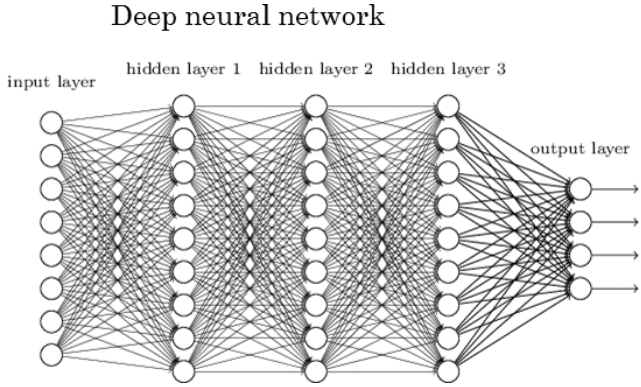


Figure: Fully connected Neural network

Src. http://www.rsipvision.com/exploring-deep-learning/

- The number of weights grows super fast!
  - Computationally heavy! Memory hungry! (Think of a 1000×1000 px image.)

What used to be seen as a deep neural network...

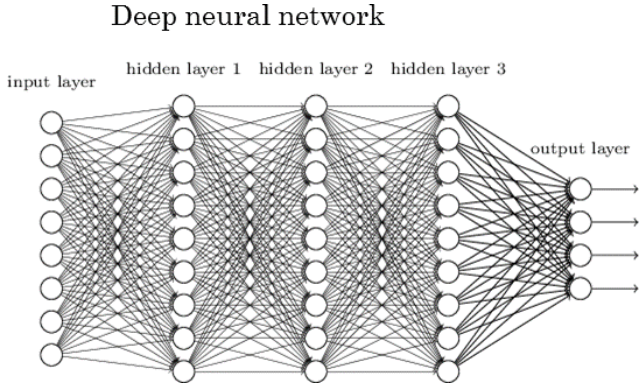
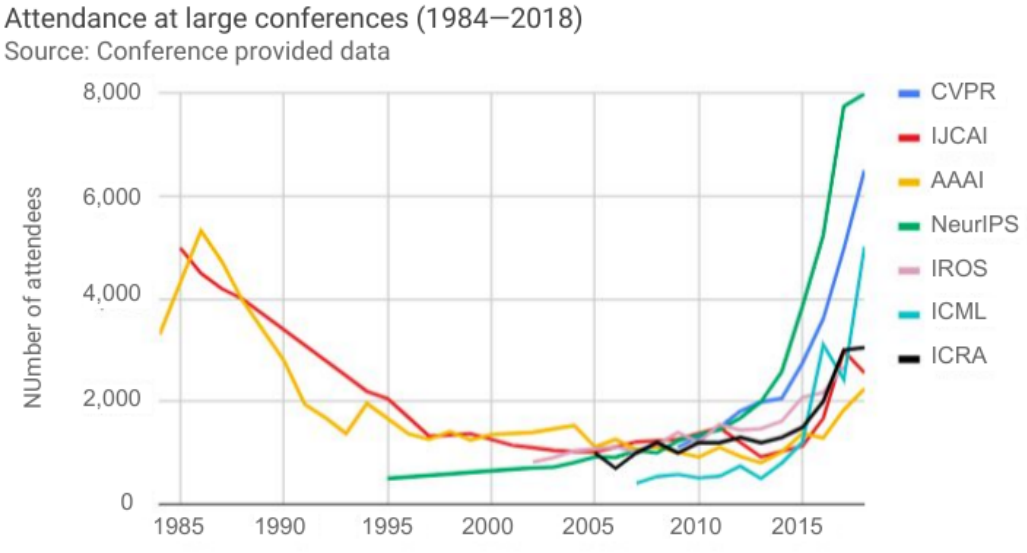


Figure: Fully connected Neural network

Src. http://www.rsipvision.com/exploring-deep-learning/

- The number of weights grows super fast!
  - Computationally heavy! Memory hungry! (Think of a 1000×1000 px image.)
  - Difficult to train! Slow convergence, needing very large amounts of training data!

A bit of history



Fully connected neural networks...

Fully connected neural networks

- connect every input with every other
- every input is treated equally, it is one dimension in a high dimensional space
- no concern for spatial structure
- every input is equally "near" any other input
- the upper left part of an image is completely independent from the upper right, and the lower left, and the lower right
- the network is forced to learn individually what a cat looks like in all parts of the image

UPPSALA UNIVERSITET

Fully connected neural networks...

Fully connected neural networks

- connect every input with every other
- every input is treated equally, it is one dimension in a high dimensional space
- no concern for spatial structure
- every input is equally "near" any other input
- the upper left part of an image is completely independent from the upper right, and the lower left, and the lower right
- the network is forced to learn individually what a cat looks like in all parts of the image

Can we do better?

A bit of history:

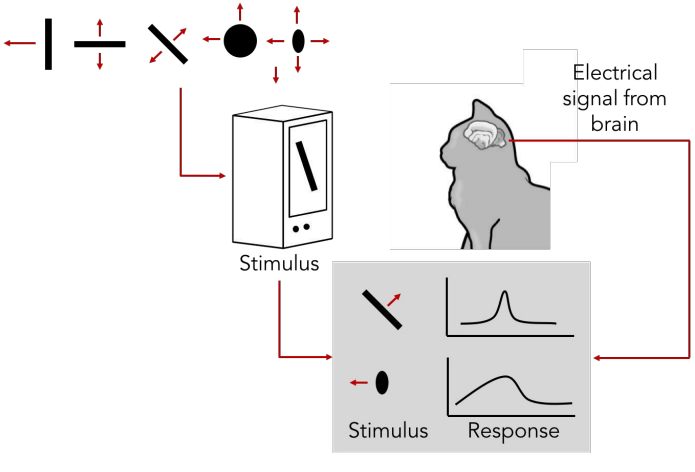
Hubel & Wiesel, 1959

RECEPTIVE FIELDS OF SINGLE NEURONES IN
THE CAT'S STRIATE CORTEX

1962

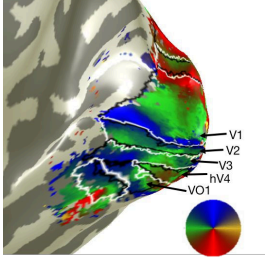
RECEPTIVE FIELDS, BINOCULAR INTERACTION
AND FUNCTIONAL ARCHITECTURE IN
THE CAT'S VISUAL CORTEX

1968...

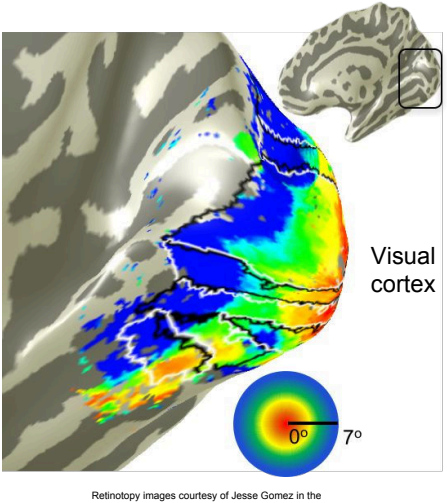


A bit of history

Topographical mapping in the cortex:
nearby cells in cortex represent
nearby regions in the visual field



Human brain



Retinotopy images courtesy of Jesse Gomez in the Stanford Vision & Perception Neuroscience Lab.

Hierarchical organization

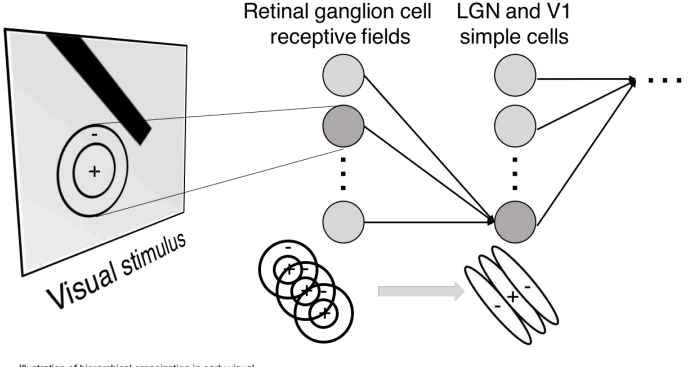
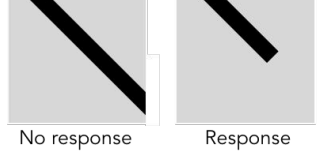


Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

Simple cells:
Response to light orientation

Complex cells:
Response to light orientation and movement

Hypercomplex cells:
response to movement with an end point

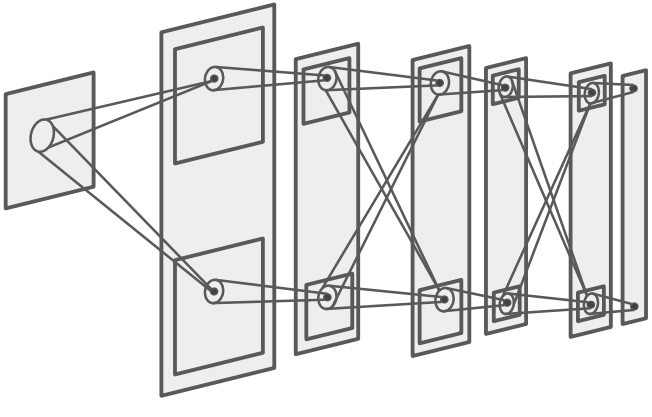


No response

Response (end point)

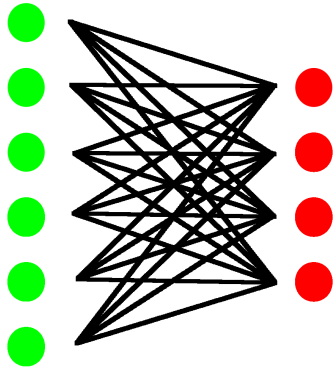
A bit of history:

Neocognitron
[Fukushima 1980]



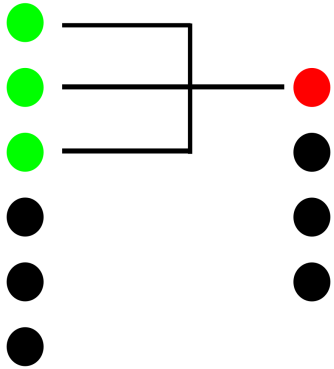
"sandwich" architecture (SCSCSC…)
simple cells: modifiable parameters
complex cells: perform pooling

Fully connected neural network



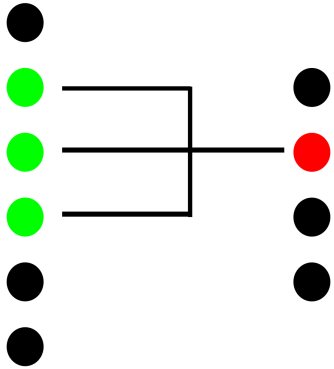
- $y_j = \sum_{i \in \text{input}} W_{ij} x_i$
- Everything connected
- Both local and non-local connections
- Does not care for spatial relationships
- Very many parameters, difficult to train
- Rarely more than a few (last) layers in a network

Convolutional neural network



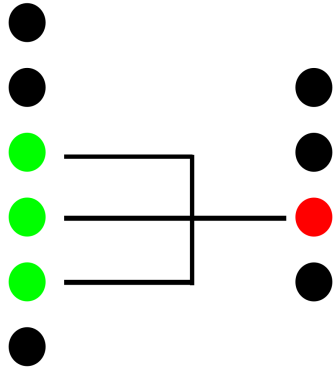
- Contains convolutional layers
- Only local connections
- Spatial relationship is preserved
- Parameter sharing
- $y_j = \sum_{i=-k}^{k} W_k x_{j+k}$
- Widely used in image analysis

Convolutional neural network



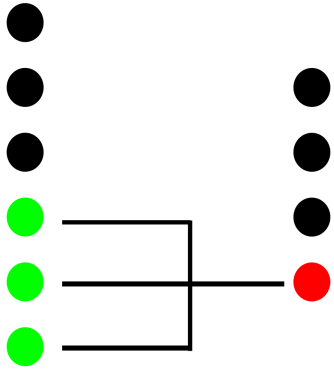
- Contains convolutional layers
- Only local connections
- Spatial relationship is preserved
- Parameter sharing
- $y_j = \sum_{i=-k}^{k} W_k x_{j+k}$
- Widely used in image analysis

Convolutional neural network



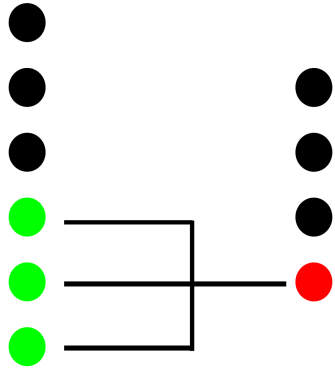
- Contains convolutional layers
- Only local connections
- Spatial relationship is preserved
- Parameter sharing
- $y_j = \sum_{i=-k}^{k} W_k x_{j+k}$
- Widely used in image analysis

Convolutional neural network



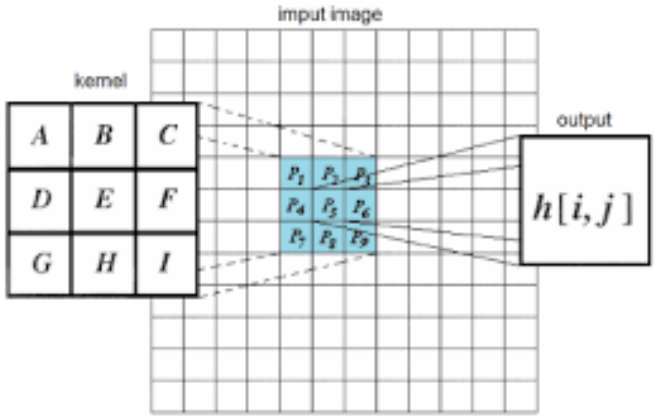
- Contains convolutional layers
- Only local connections
- Spatial relationship is preserved
- Parameter sharing
- $y_j = \sum_{i=-k}^{k} W_k x_{j+k}$
- Widely used in image analysis

Convolutional neural network

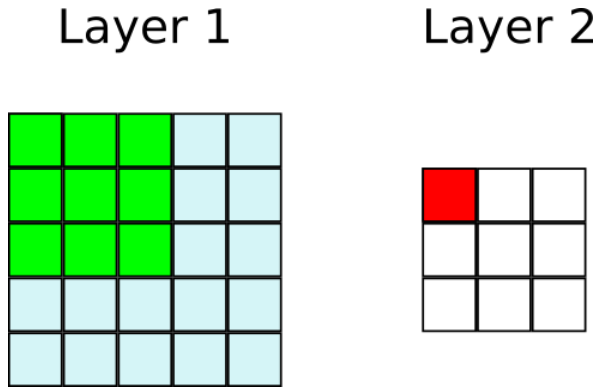


- Contains convolutional layers
- Only local connections
- Spatial relationship is preserved
- Parameter sharing
- $y_j = \sum_{i=-k}^{k} W_k x_{j+k}$
- Widely used in image analysis
- A linear finite impulse response (FIR) filter

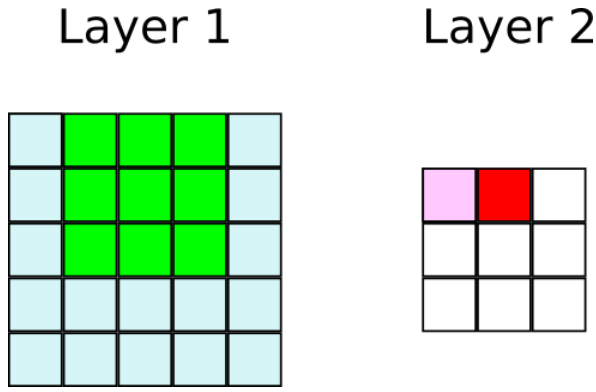
2d filtering – Cross correlation



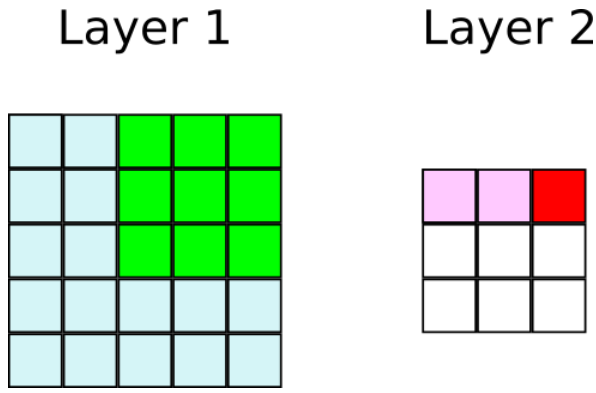
2d filtering



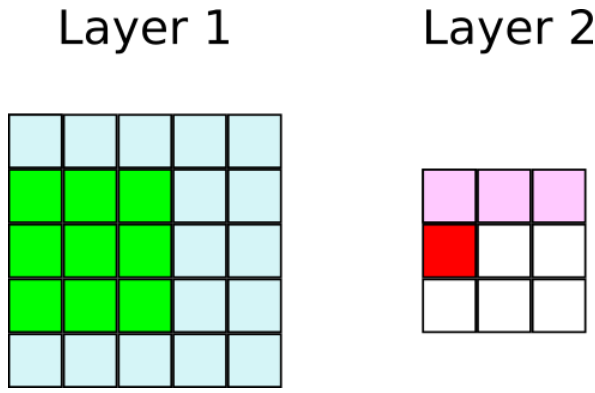
2d filtering



2d filtering

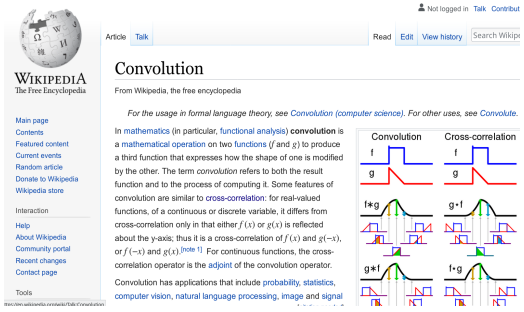


2d filtering



Convolution – Linear Shift-Invariant (LSI) system

Convolution – Mirror the filter in all axes (in 2D this is a 180 degree rotation).



Not logged in Talk Contribut

Article Talk Read Edit View history Search Wikipe

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools

Convolution

From Wikipedia, the free encyclopedia

For the usage in formal language theory, see Convolution (computer science). For other uses, see Convolute.

In mathematics (in particular, functional analysis) **convolution** is a mathematical operation on two functions (f and g) to produce a third function that expresses how the shape of one is modified by the other. The term *convolution* refers to both the result function and to the process of computing it. Some features of convolution are similar to cross-correlation: for real-valued functions, of a continuous or discrete variable, it differs from cross-correlation only in that either $f(x)$ or $g(x)$ is reflected about the y-axis; thus it is a cross-correlation of $f(x)$ and $g(-x)$, or $f(-x)$ and $g(x)$.[note 1] For continuous functions, the cross-correlation operator is the adjoint of the convolution operator.

Convolution has applications that include probability, statistics, computer vision, natural language processing, image and signal

Definition [edit]

The convolution of f and g is written $f{*}g$, using an asterisk or star. It is defined as the integral of the pro one is reversed and shifted. As such, it is a particular kind of integral transform:

$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau) g(t-\tau)\, d\tau.$$

Properties [edit]

Algebraic properties [edit]

See also: Convolution algebra

The convolution defines a product on the linear space of integrable functions. This product satisfies the following algebra product given by convolution is a commutative associative algebra without identity (Strichartz 1994, §3.3). Other linear sp are closed under the convolution, and so also form commutative associative algebras.

Commutativity

$$f * g = g * f$$

Proof: By definition

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t-\tau)\, d\tau$$

Changing the variable of integration to $u = t - \tau$ and the result follows.

Associativity

$$f * (g * h) = (f * g) * h$$

Proof: This follows from using Fubini's theorem (i.e., double integrals can be evaluated as iterated integrals in either ord

Distributivity

$$f * (g + h) = (f * g) + (f * h)$$

Proof: This follows from linearity of the integral.

Associativity with scalar multiplication

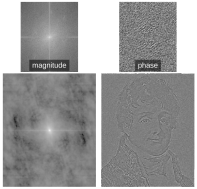
$$a(f * g) = (af) * g$$

for any real (or complex) number a.

Convolutions

Convolutions are very common operations in Image processing.

Fast computation of large convolutions can be done via the Fourier domain:



$$f * g = \mathcal{F}^{-1}\left\{\mathcal{F}(f) \cdot \mathcal{F}(g)\right\}$$

```
I =
     1     4     7
     2     5     8
     3     6     9
>> full(convmtx2(I,4,4))
ans =
     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     2     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     3     2     1     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     3     2     1     0     0     0     0     0     0     0     0     0     0     0     0
     4     0     3     2     0     0     0     0     0     0     0     0     0     0     0     0
     5     4     0     3     2     0     0     0     0     0     0     0     0     0     0     0
     6     5     4     0     3     2     0     0     0     0     0     0     0     0     0     0
     0     6     5     4     0     3     2     0     0     0     0     0     0     0     0     0
     7     0     6     5     0     0     3     2     0     0     0     0     0     0     0     0
     8     7     0     6     5     0     0     3     2     0     0     0     0     0     0     0
     9     8     7     0     6     5     0     0     3     2     0     0     0     0     0     0
     0     9     8     7     0     6     5     0     0     3     2     0     0     0     0     0
     0     0     9     8     0     0     6     5     0     0     3     2     0     0     0     0
     0     0     0     9     0     0     0     6     5     0     0     3     2     0     0     0
     0     0     0     0     0     0     0     0     6     5     0     0     3     2     0     0
     0     0     0     0     0     0     0     0     0     6     5     0     0     3     2     0
     0     0     0     0     0     0     0     0     0     0     6     5     0     0     3     2
     0     0     0     0     0     0     0     0     0     0     0     6     5     0     0     3
     0     0     0     0     0     0     0     0     0     0     0     0     6     5     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     6     5     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     6     5
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     6
```

convmtx2

2-D convolution matrix

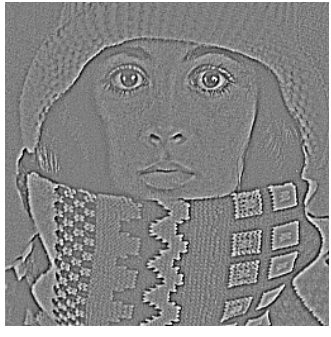
Syntax

```
T = convmtx2(H,m,n)
T = convmtx2(H,[m n])
```

Description

`T = convmtx2(H,m,n)` returns the convolution matrix `T` for the matrix `H`. If `X` is an `m`-by-`n` matrix, then `reshape(T*X(:),size(H`

Laplace filter



Laplace operator: $\Delta = \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$

Sobel filter

Approximates the first derivatives: $\frac{\partial}{\partial x}$, $\frac{\partial}{\partial y}$

1	0	-1
2	0	-2
1	0	-1

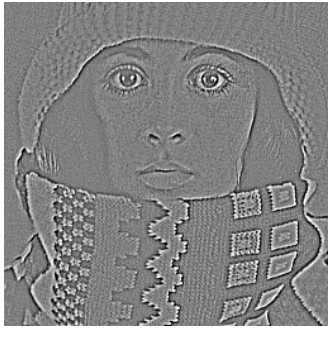
S_x

1	2	1
0	0	0
-1	-2	-1

S_y

Sharpening an image

$$\text{sharpened} = (1+\alpha) \cdot \text{original} - \alpha \cdot \text{smoothed}$$
$$\text{sharpened} = \text{original} + \alpha \cdot (\text{original} - \text{smoothed})$$



0	0	0
0	1	0
0	0	0

original

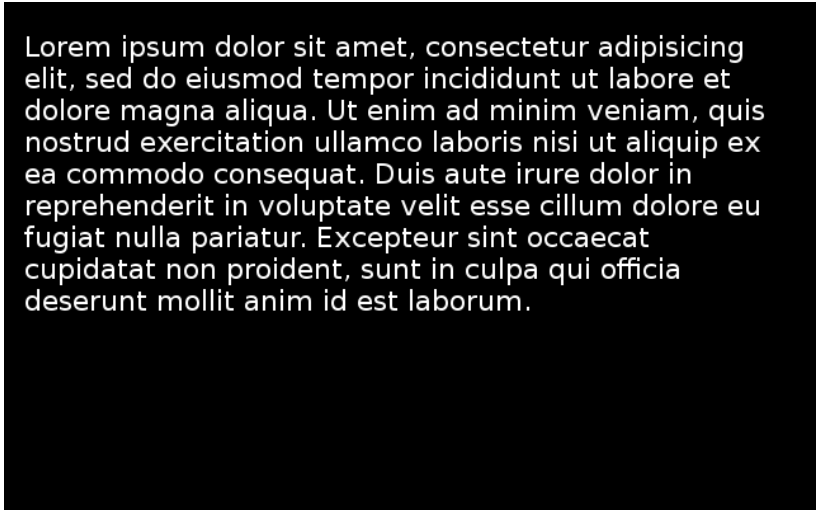
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

smoothed

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

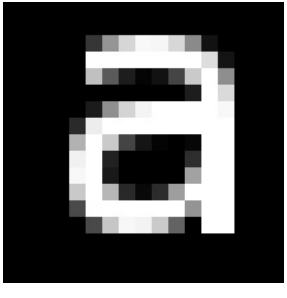
9 · diff.

Beyond smoothing and sharpening



An image of a piece of text.

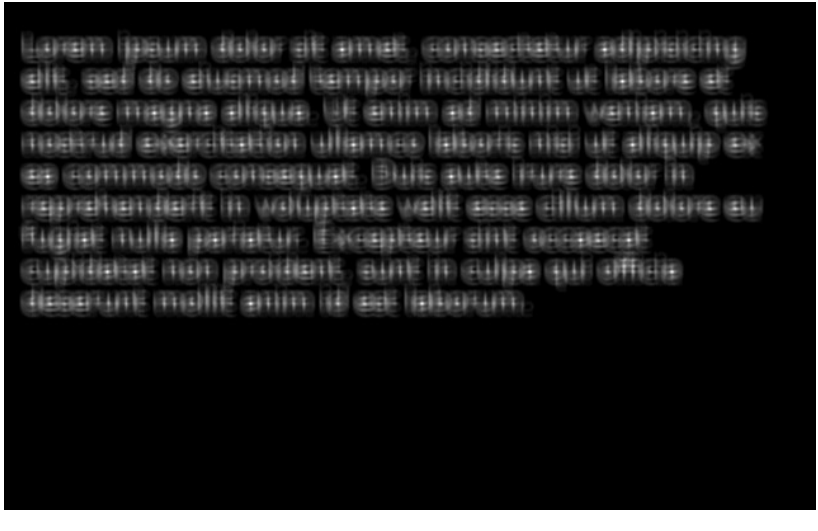
Beyond smoothing and sharpening



Filter kernel, image of letter “a”.

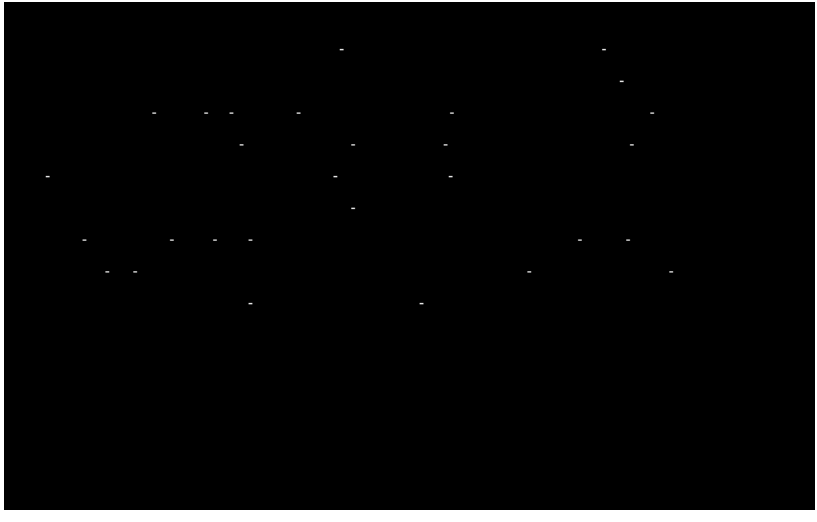
What happens when we apply this kernel as a linear filter?
When is the output of this filter maximum/minimum?

Beyond smoothing and sharpening



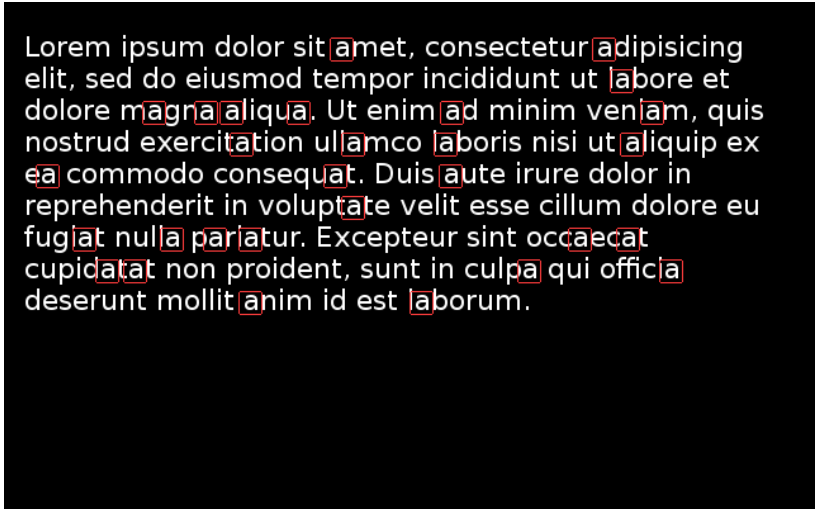
After linear filtering

Beyond smoothing and sharpening



Finding all pixels brighter than a manually selected threshold value

Beyond smoothing and sharpening



Detected instances of letter “a”.

Convolutional neural networks - born in 1989

Communicated by Dana Ballard

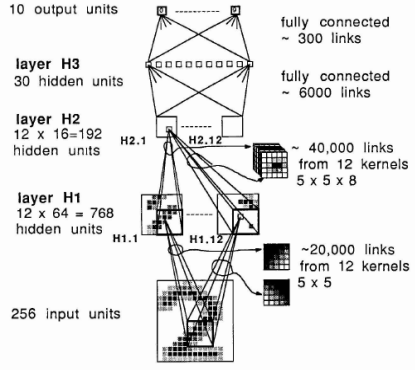
Backpropagation Applied to Handwritten Zip Code Recognition

Y. LeCun
B. Boser
J. S. Denker
D. Henderson
R. E. Howard
W. Hubbard
L. D. Jackel
AT&T Bell Laboratories Holmdel, NJ 07733 USA

The ability of learning networks to generalize can be greatly enhanced by providing constraints from the task domain. This paper demonstrates how such constraints can be integrated into a backpropagation network through the architecture of the network. This approach has been successfully applied to the recognition of handwritten zip code digits provided by the U.S. Postal Service. A single network learns the entire recognition operation, going from the normalized image of the character to the final classification.

1 Introduction

Previous work performed on recognizing simple digit images (LeCun 1989) showed that good generalization on complex tasks can be obtained



Deep convolutional neural networks - born in 1998

Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

Abstract—
Multilayer Neural Networks trained with the backpropagation algorithm constitute the best example of a successful Gradient-Based Learning technique. Given an appropriate network architecture, Gradient-Based Learning algorithms can be used to synthesize a complex decision surface that can classify high-dimensional patterns such as handwritten characters, with minimal preprocessing. This paper reviews various methods applied to handwritten character recognition and compares them on a standard handwritten digit recognition task. Convolutional Neural Networks, that are specifically designed to deal with the variability of 2D shapes, are shown to outperform all other techniques.

I. Introduction

Over the last several years, machine learning techniques, particularly when applied to neural networks, have played an increasingly important role in the design of pattern recognition systems. In fact, it could be argued that the availability of learning techniques has been a crucial factor in the recent success of pattern recognition applications such as continuous speech recognition and handwriting recognition.

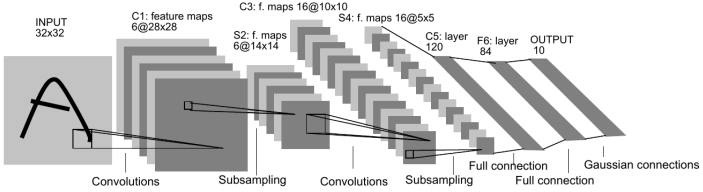
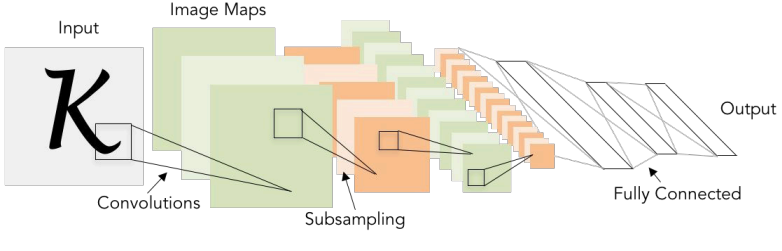


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

A bit of history:

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]



LeNet-5

A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]

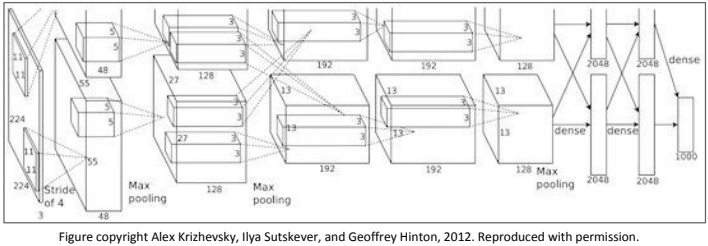
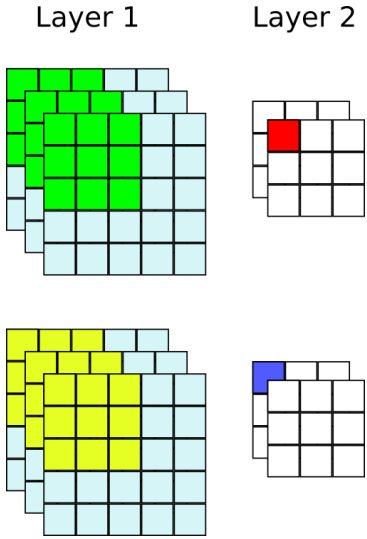


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

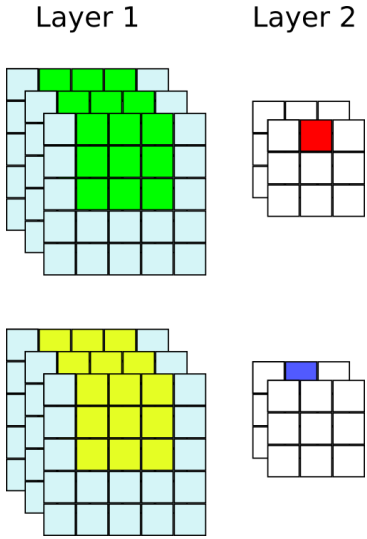
"AlexNet"

3d convolutions



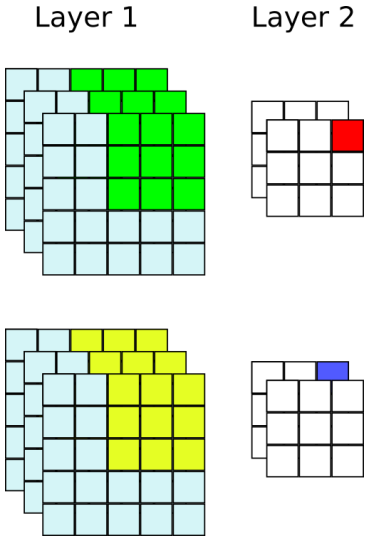
- Filter coefficients are learned from data
- Can be implemented as matrix multiplication (faster)
- Efficient GPU implementations are possible
- Implemented as tensor multiplications/additions
- Hierarchical feature extraction

3d convolutions



- Filter coefficients are learned from data
- Can be implemented as matrix multiplication (faster)
- Efficient GPU implementations are possible
- Implemented as tensor multiplications/additions
- Hierarchical feature extraction

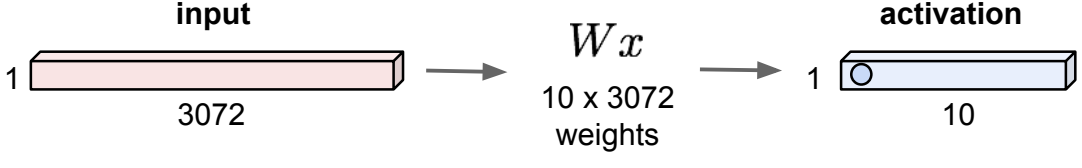
3d convolutions



- Filter coefficients are learned from data
- Can be implemented as matrix multiplication (faster)
- Efficient GPU implementations are possible
- Implemented as tensor multiplications/additions
- Hierarchical feature extraction

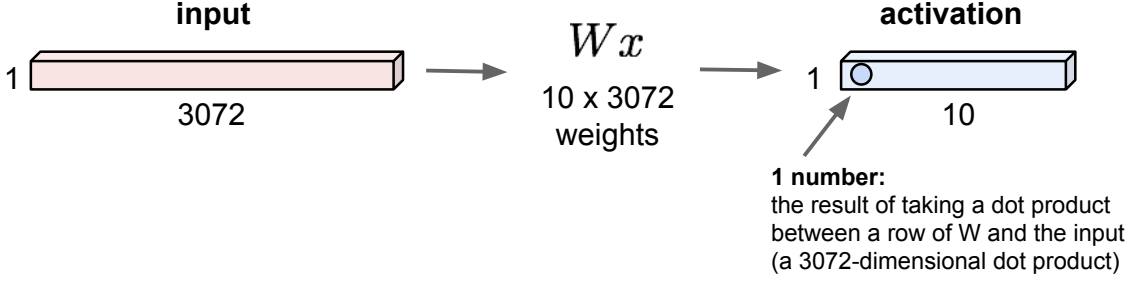
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



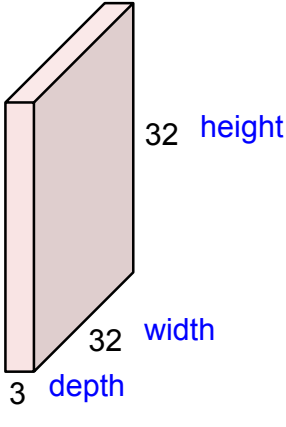
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



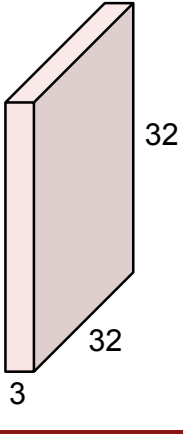
Convolution Layer

32x32x3 image -> preserve spatial structure

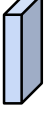


Convolution Layer

32x32x3 image



5x5x3 filter

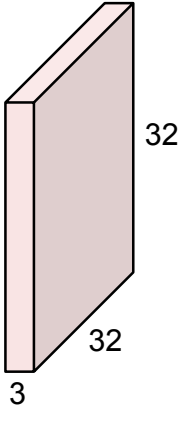


Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

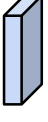
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

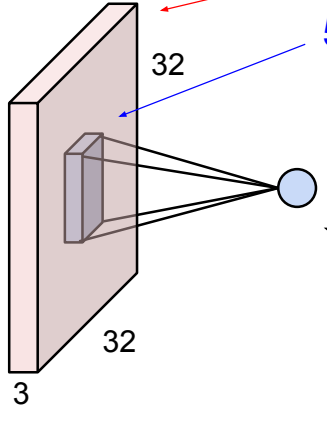


5x5x3 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer



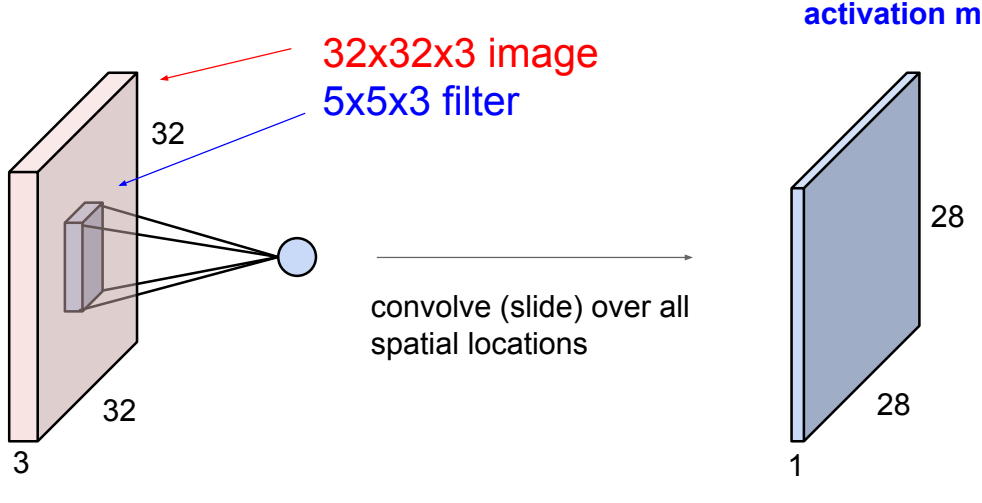
32x32x3 image

5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

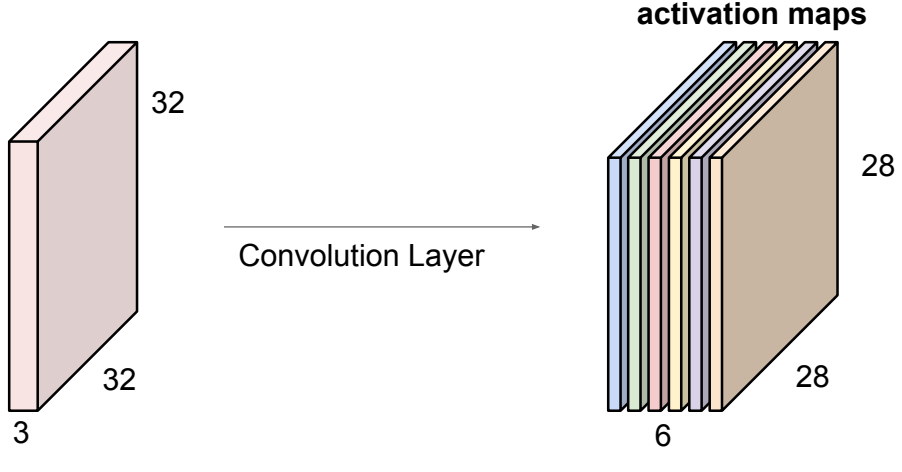
Convolution Layer



Convolution Layer

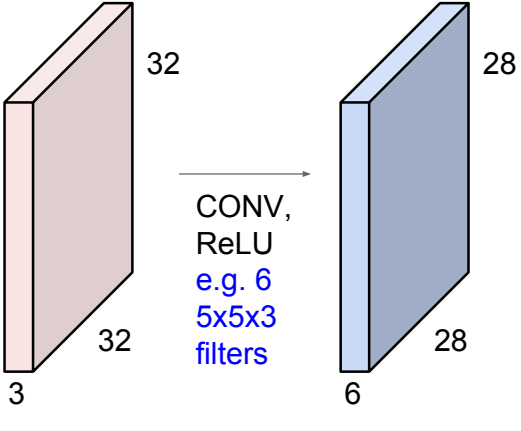
consider a second, green filter

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

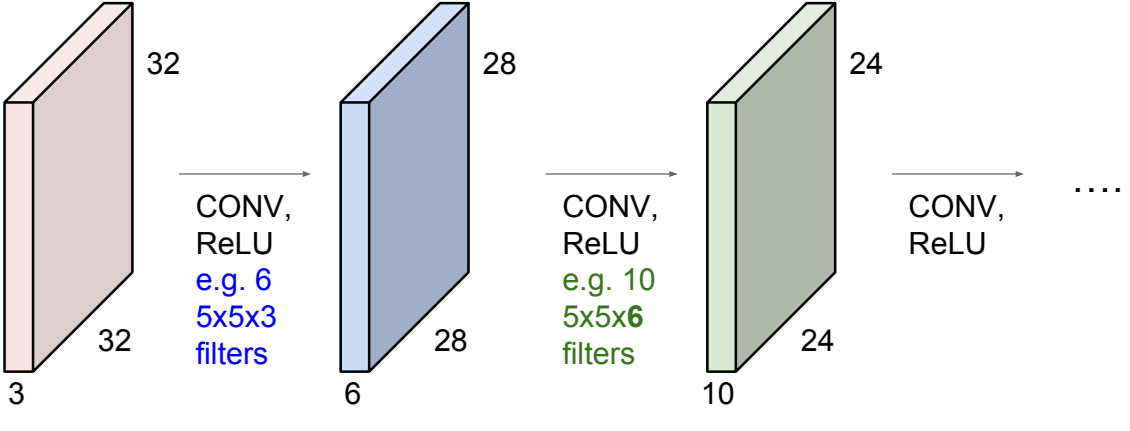


We stack these up to get a "new image" of size 28x28x6!

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



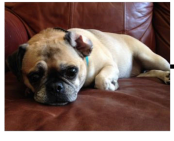
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



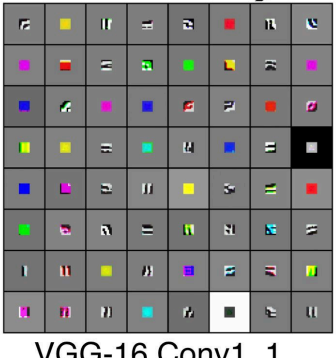
Preview

[Zeiler and Fergus 2013]

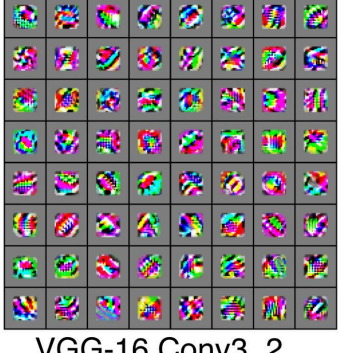
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



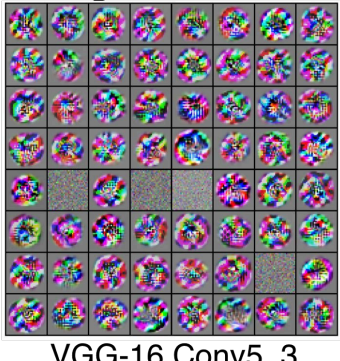
Low-level features → Mid-level features → High-level features → Linearly separable classifier



VGG-16 Conv1_1

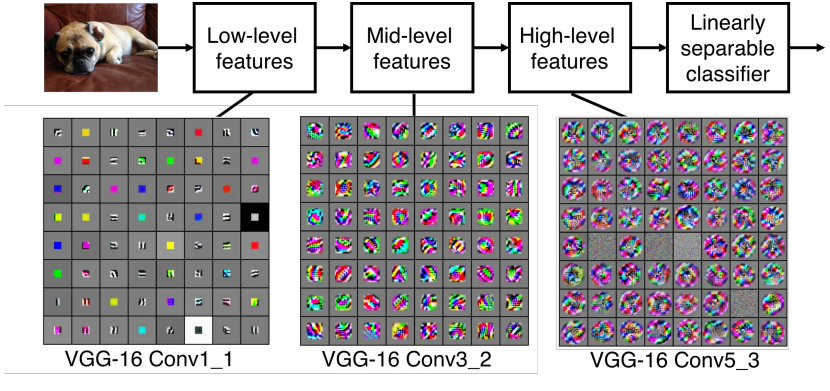


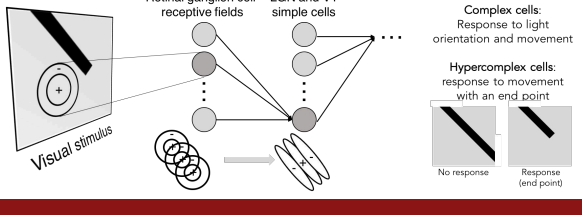
VGG-16 Conv3_2



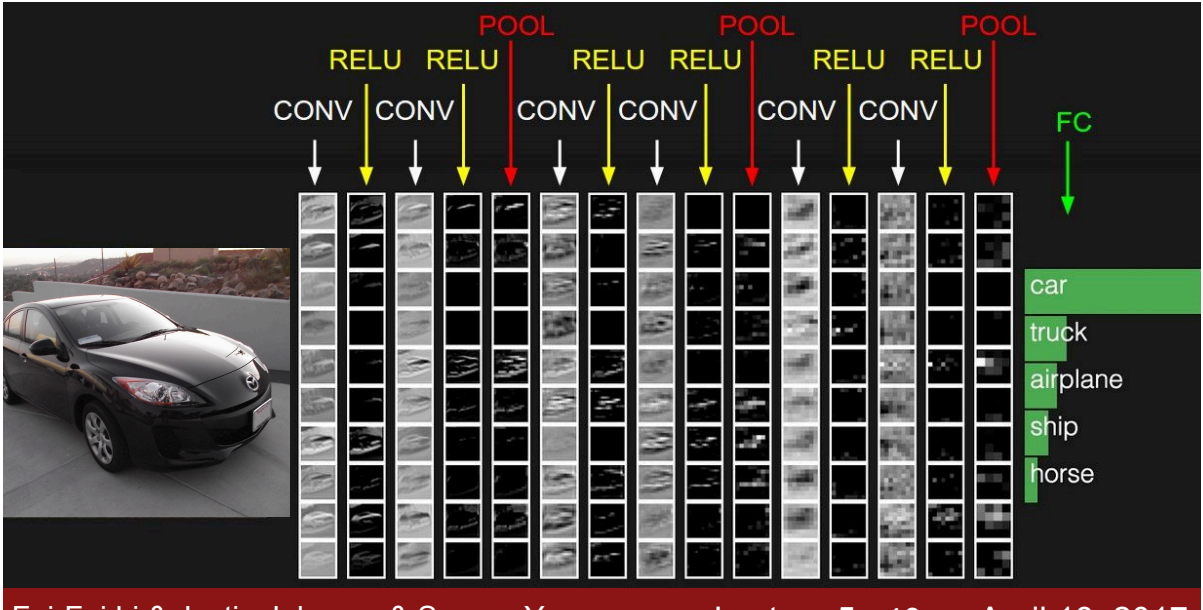
VGG-16 Conv5_3

Preview

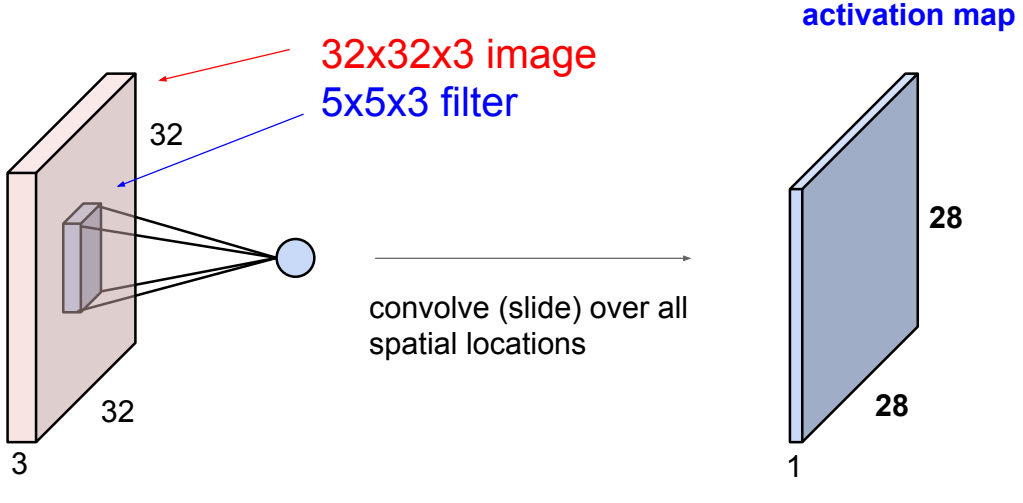

Low-level features
Mid-level features
High-level features
Linearly separable classifier
VGG-16 Conv1_1
VGG-16 Conv3_2
VGG-16 Conv5_3


Retinal ganglion cell receptive fields
LGN and V1 simple cells
Visual stimulus
Complex cells: Response to light orientation and movement
Hypercomplex cells: response to movement with an end point
No response
Response (end point)

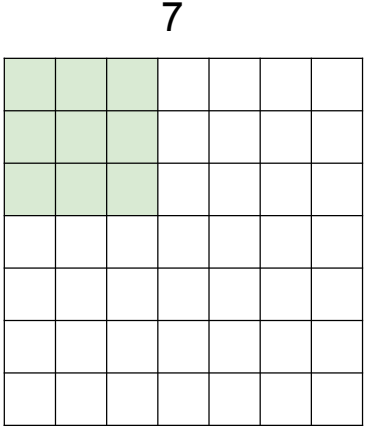
preview:


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

A closer look at spatial dimensions:

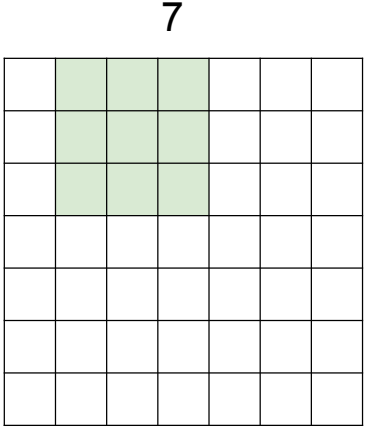


A closer look at spatial dimensions:



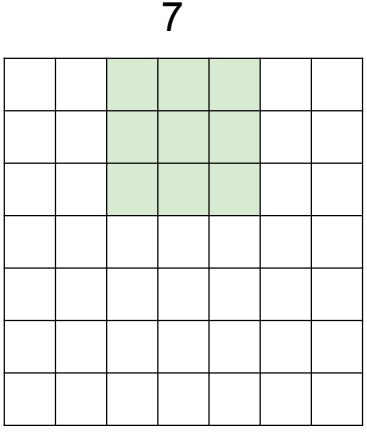
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



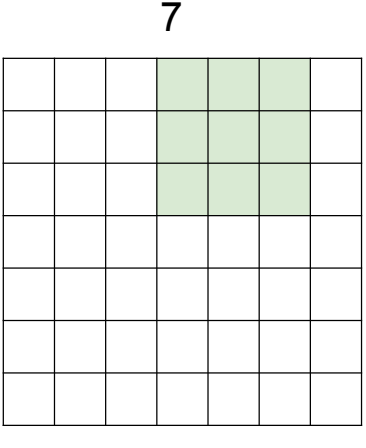
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



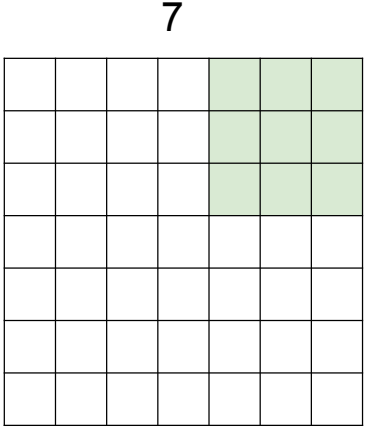
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

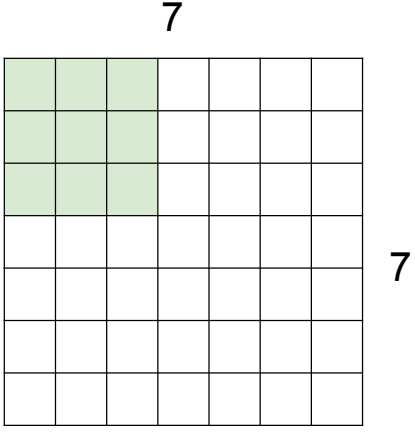
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

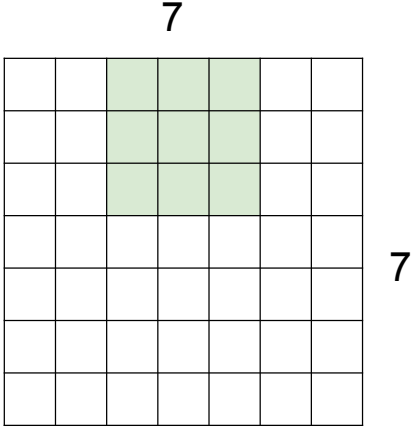
=> 5x5 output

A closer look at spatial dimensions:



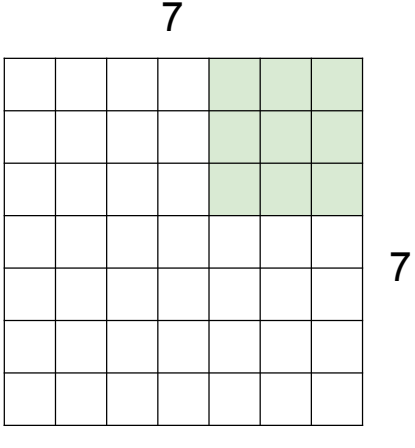
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



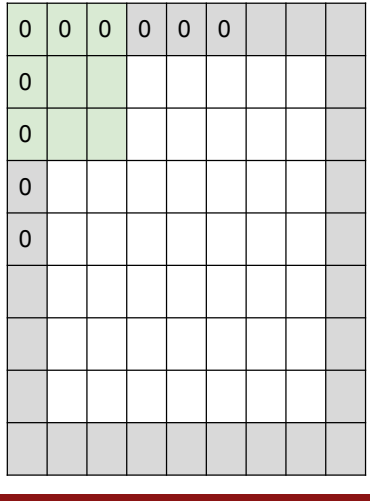
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

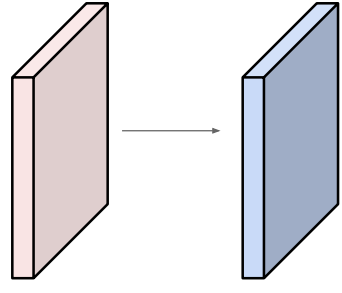
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

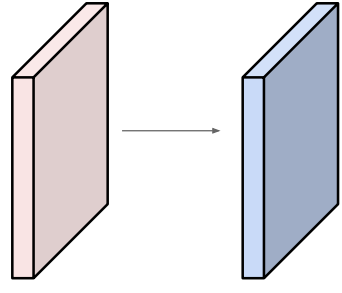
Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?

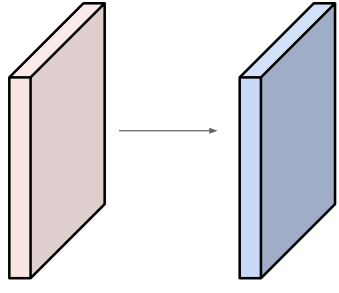
Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

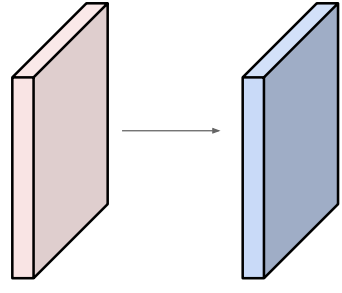
Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

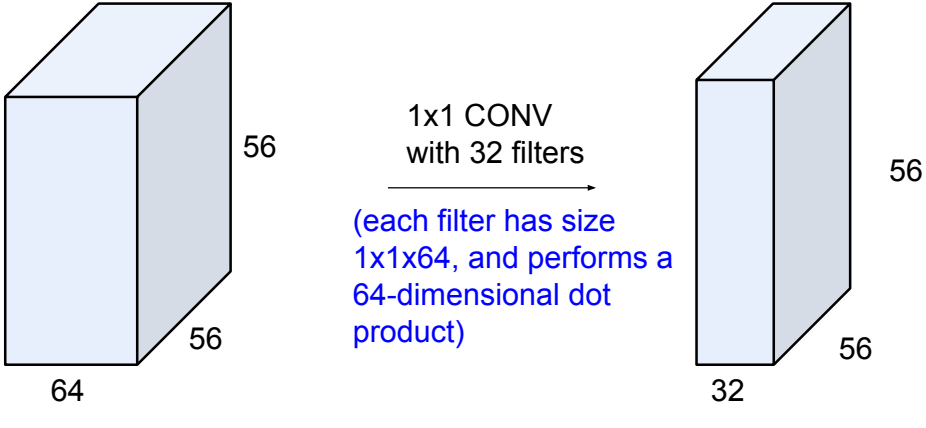
Examples time:



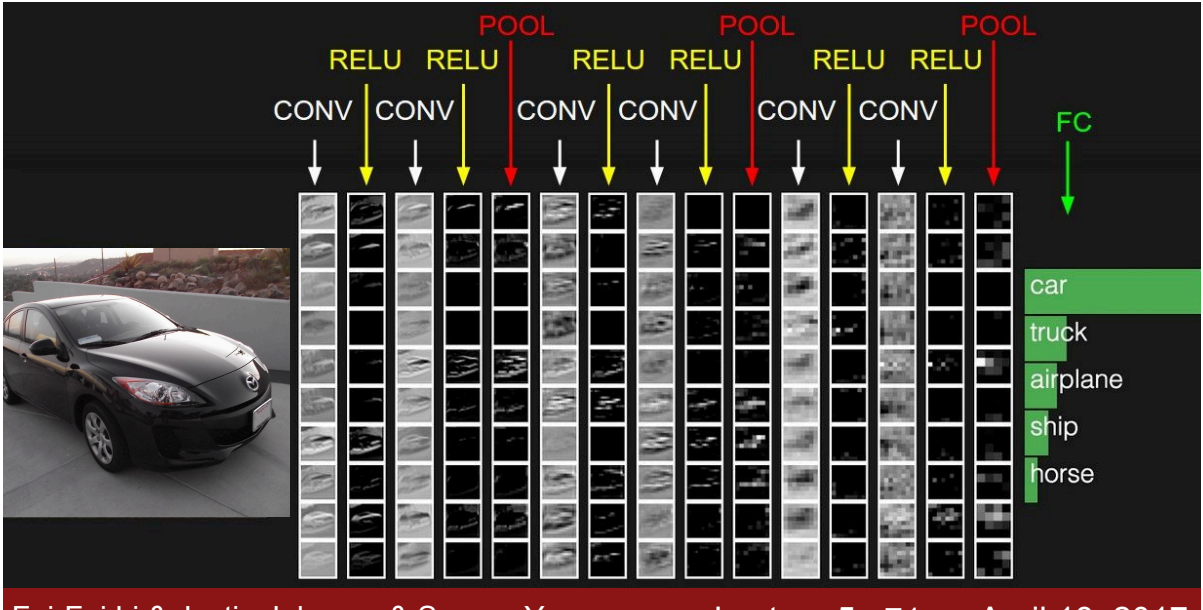
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

(btw, 1x1 convolution layers make perfect sense)

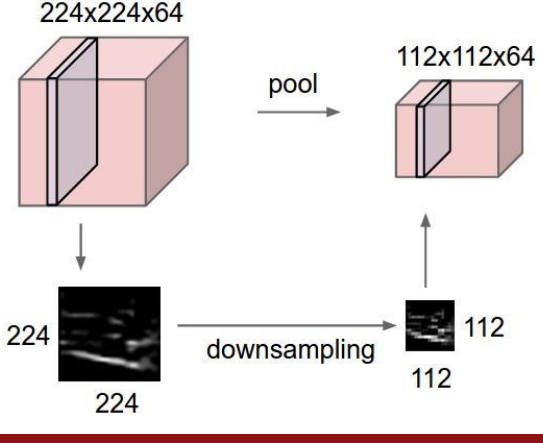


two more layers to go: POOL/FC

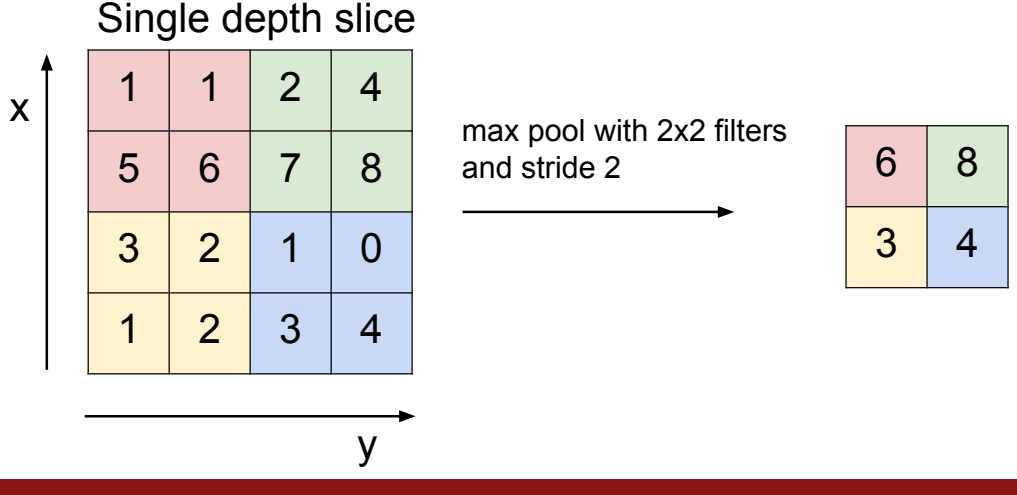


Pooling layer

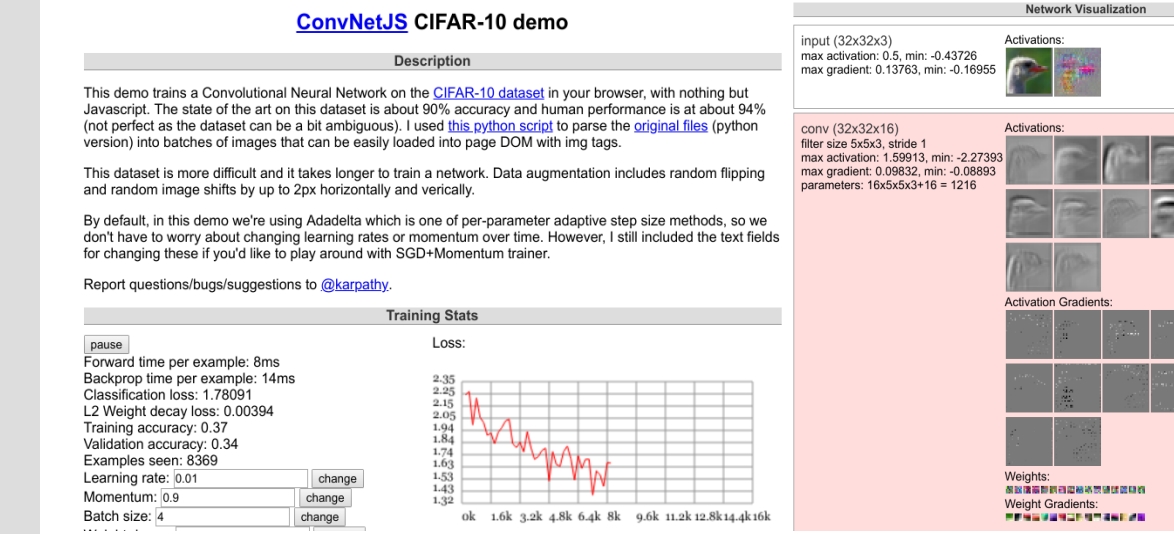
- makes the representations smaller and more manageable
- operates over each activation map independently:



MAX POOLING


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

ConvNetJS demo: training on CIFAR-10



https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

More about convolutions

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Filter visualization

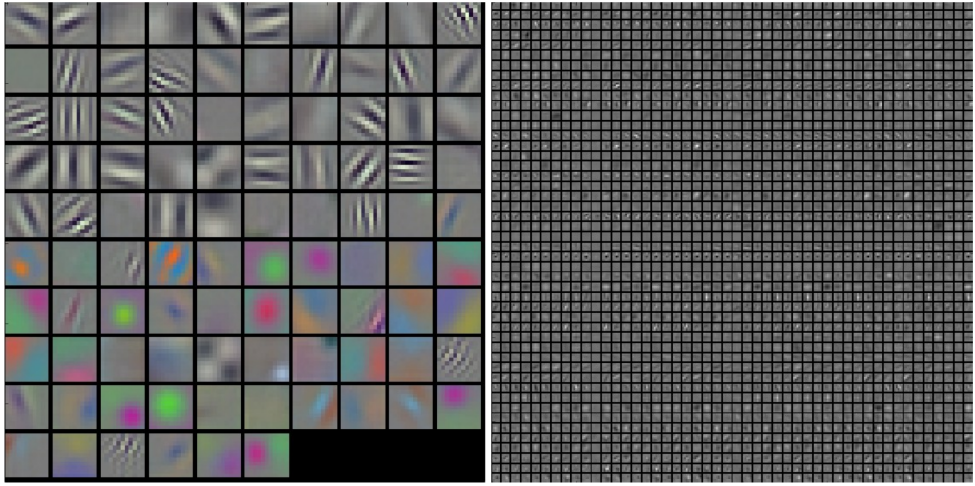


Figure: First and second layer features of Alexnet

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

Filter visualization

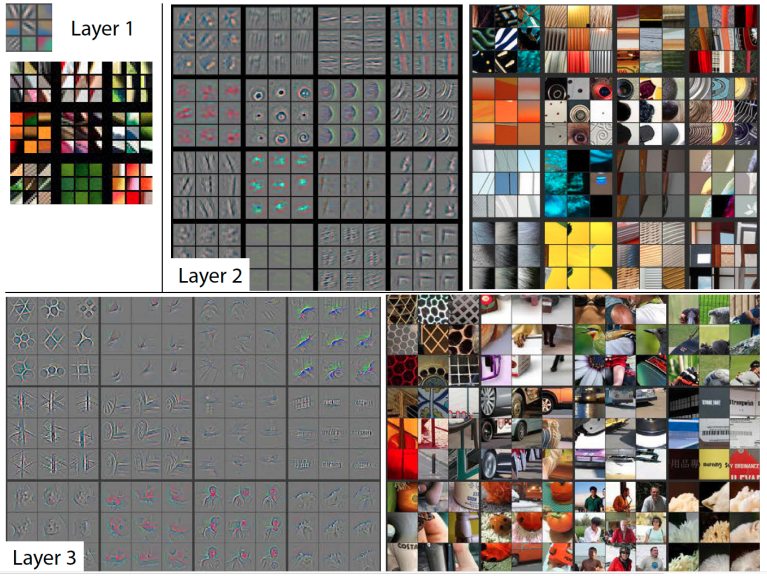


Figure: Src. Matthew D. Zeiler, et al, Visualizing and Understanding Convolutional Networks, ECCV 2014

Filter visualization

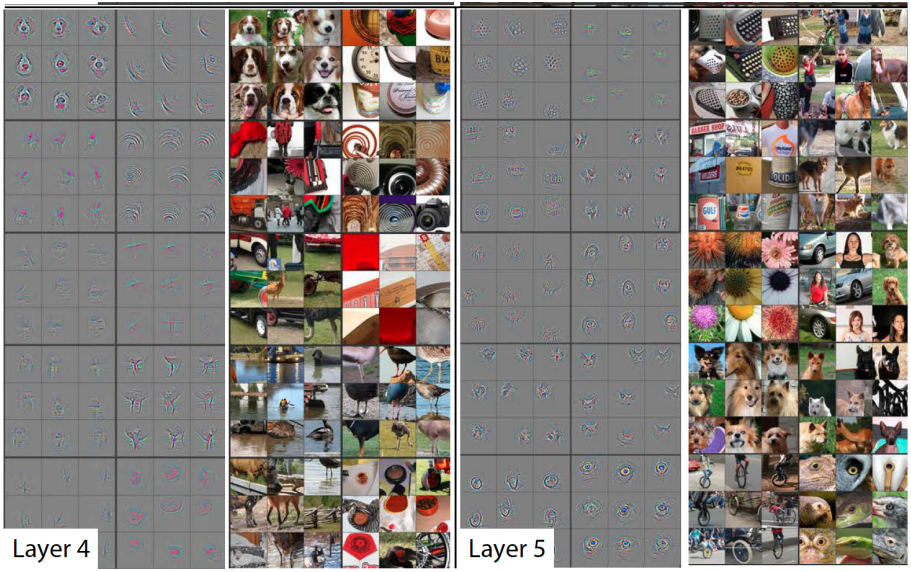


Figure: Src. Matthew D. Zeiler, et al, Visualizing and Understanding Convolutional Networks, ECCV 2014

Lenet

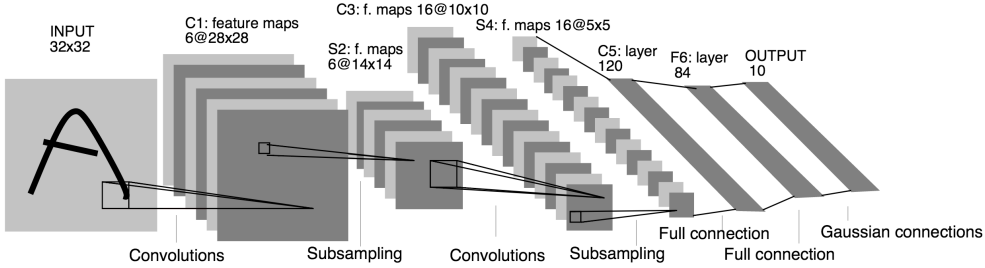


Figure: Src. Yann LeCun, et al, Gradient-based learning applied to document recognition, 1998

Alexnet

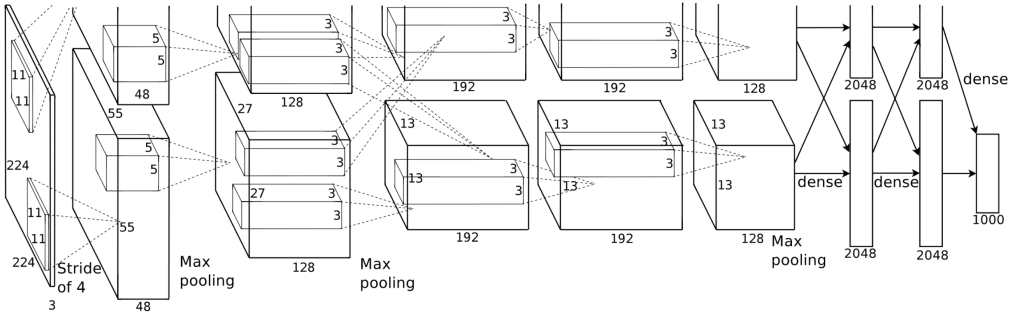


Figure: Src. Alex Krishevsky et al, ImageNet Classification with Deep Convolutional Neural Networks, 2012

Googlenet

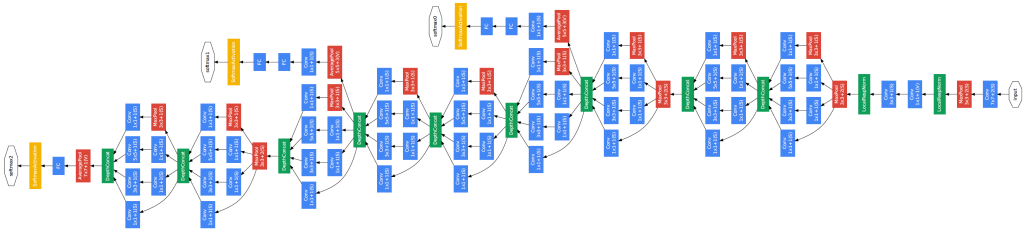


Figure: Src. Going deeper with convolutions [?]

Googlenet- inception layer

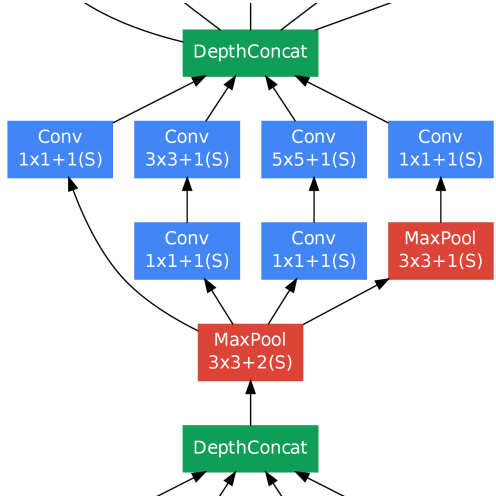


Figure: Src. Going deeper with convolutions [?]

U-Net

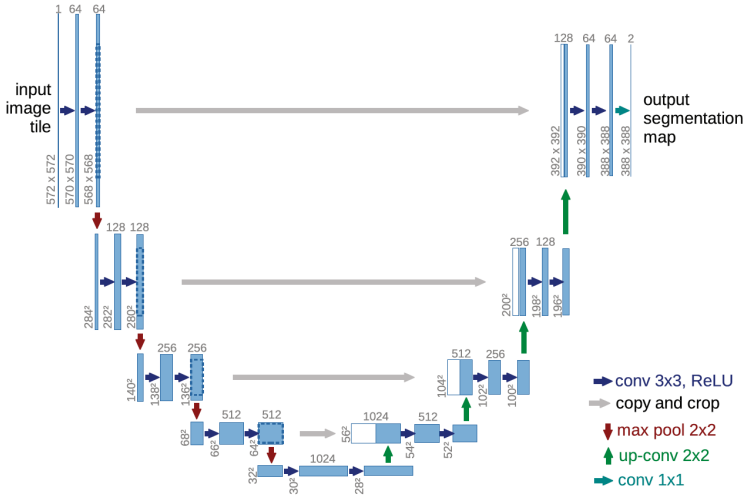


Figure: Src. Olaf Ronneberger, et al., U-Net: Convolutional Networks for Biomedical Image Segmentation, MICCAI 2015

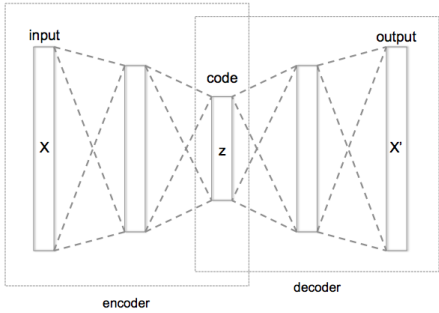
Types of deep neural networks

- Convolutional neural network
  - Residual networks
  - U-Net style (with transposed convolution and skip-links)
  - Region proposal networks (RPN, "Faster R-CNN")
- Recurrent neural network (RNN)
  - LSTM networks [Hochreiter and Schmidhuber (1997)]
- Auto encoders

Recurrent neural networks

- Connections between nodes form directed cycles.
- Create internal memory.
- Used to process sequence of data such as speech, text, video etc.
- Long short term memory (LSTM) commonly used now.

Auto encoders

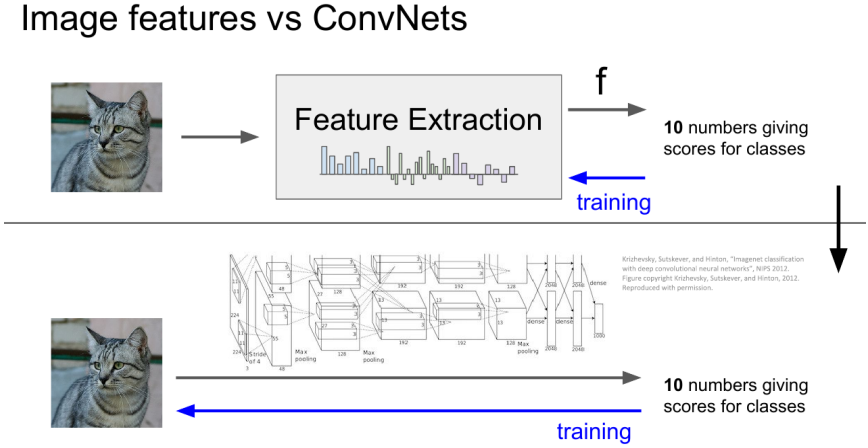


- Unsupervised feature learning
- Uses dimensionality reduction
- Tries to recreate input after mapping to lower dimension
- Can stack multiple auto encoders to create deeper network

Figure: Src. by Chervinskii - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45555552

Deep vs. shallow learning

Deep may mean "many layers" but may also mean "end-to-end learning" (avoiding engineered features).



A few popular DL frameworks

- Matlab, easy to get going, easy to install (on Windows), good examples, propriety, limiting flexibility after a while. Make sure to use the latest version!
- PyTorch is good for research, easy to modify – most popular at the Centre for Image Analysis!
- TensorFlow is a safe bet for most projects. Not perfect but has huge community, wide usage. Maybe pair with high-level wrapper (**Keras**, Sonnet, etc), good for one graph over many machines
- Consider Caffe2, or TensorFlow for production deployment and mobile

A few concepts to summarize lecture 4

Fully connected layer: Each output is connected to every input. (Input size plus one) $\times$ output size number of parameters.

Convolutional layer: Only local connections and weight sharing; sliding filter over the image. (Filter size plus one) $\times$ number of filters (depth) number of parameters.

Pooling layer: Aggregation of local information plus downsampling (striding). Most often max-pooling, but sometimes average pooling. No parameters.

Striding: When only computing output for every n-th pixel/voxel, reducing the spatial size of the activation. For example strided convolution.

Deep learning: End-to-end learning, where the machine learning methods "learns" features.

Shallow learning: Machine learning which relies on engineered features instead of the raw data.