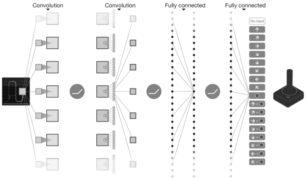
Machine Learning

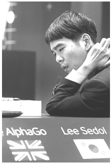
– winter term 2016/17 –

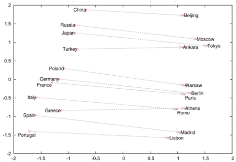
Chapter 09: Deep Learning

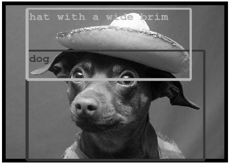
Prof. Adrian Ulges
Masters "Computer Science"
DCSM Department
University of Applied Sciences RheinMain

Deep Learning Applications images from [14] [16] [1] [13] [18]









In this Chapter

- Why deep learning is hard
- Tricks to make it work
- Convolutional neural networks
- State-of-the-art in Neural Networks

Outline

1. Why Deep Learning is Hard

2. Tricks to make Deep Learning Work

3. Convolutional Neural Networks

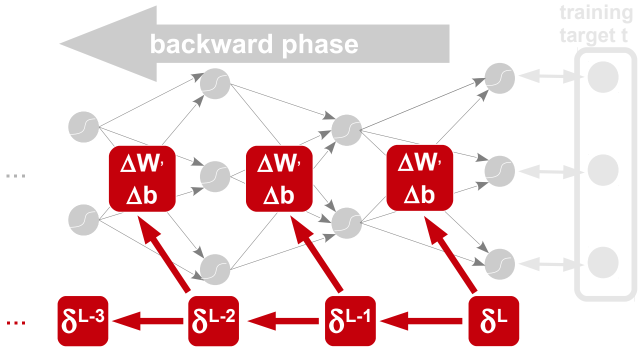
4. Deep Learning: Sample Models

Deep Learning: Characterization

Deep Learners are models which ...

- ... consist of "many" layers of nonlinear units (=neurons)
  (many = at least 3?)
- ... are in contrast to "shallow" learners
  (e.g., logistic regression, SVMs → 1 layer)
- ... learn representations of data whose *abstraction increases through the layers*
- ... use these representations instead of *hand-crafted features*
- ... often learn these representations in an *unsupervised* manner on large-scale datasets

Backpropagation (Reprise)



Backprop Formulas

$$\boldsymbol{\delta}^L = \left(\mathbf{a}^L - \mathbf{t}\right) \odot f'(\mathbf{z}^L) \qquad \Delta w_{ij}^l = -\lambda \cdot \delta_j^l \cdot a_i^{l-1}$$

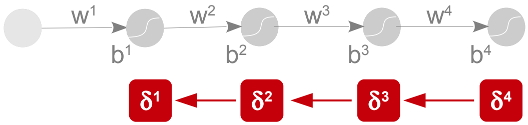
$$\boldsymbol{\delta}^l = \left(W^{l+1} \cdot \boldsymbol{\delta}^{l+1}\right) \odot f'(\mathbf{z}^l) \qquad \Delta b_j^l = -\lambda \cdot \delta_j^l$$

Key Problem: Unstable Gradients

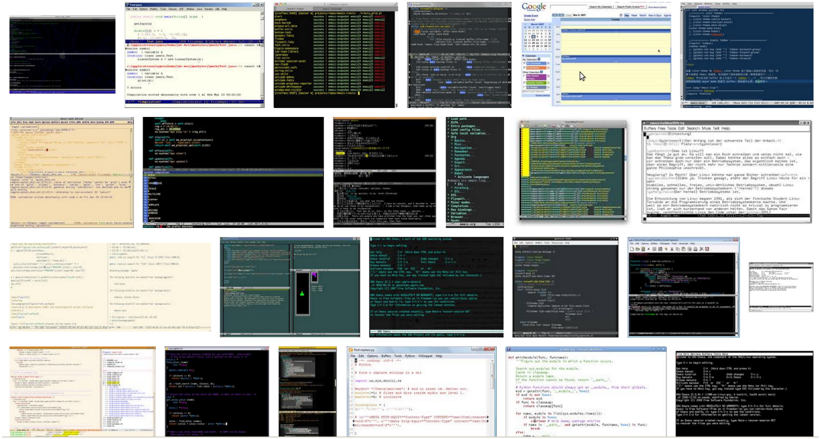
"As we move from the output layer to earlier layers the gradient tends to either vanish (the **vanishing gradient problem***) or explode (***the exploding gradient problem***). Since the gradient is the signal we use to train, this causes problems."*

(Nielsen, "Neural Networks and Deep Learning")

Dummy Network (1 neuron per layer, sigmoid activation f, see [15])

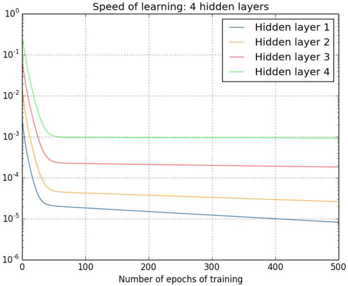


Unstable Gradients: Example



Vanishing Gradients: Example image from [15]

- a neural network trained on MNIST data
  (30 neurons per hidden layer, 4 hidden layers, fully connected)
- The delta-values in the different layers, $\boldsymbol{\delta}^1, \boldsymbol{\delta}^2, ..., \boldsymbol{\delta}^L$, indicate how strong the weights change during learning.
- We measure this **"speed" of learning** in the different layers by $||\boldsymbol{\delta}^1||, ||\boldsymbol{\delta}^2||, ..., ||\boldsymbol{\delta}^L||$.



- Note that the scale is logarithmic
  (Layer 1 learns 100× slower than Layer 4)

Outline

Deep Learning: What to do?

Improving Optimization (= avoid unstable gradients)

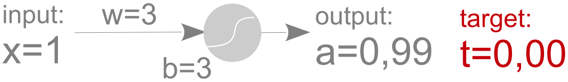
- different loss function *(→ cross-entropy)*
- different activation function *(→ RELU)*
- variations to backpropagation *(→ momentum, Chapter 08)*
- advanced techniques

Improving Generalization (= avoid overfitting)

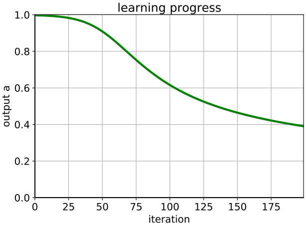
- regularization + dropout
- network topology (CNNs)
- more processing power (GPUs)
- larger training sets
  - Pascal VOC Challenge (2005-2012): 11K training images
  - ILSVRC (2012-...): 1,3 mio. training images

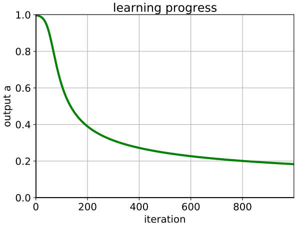
Trick 1: Cross-Entropy Cost

Example: a poorly initialized neuron (see [15])



- Our old cost function (squared error): $E = \frac{1}{2}(a-t)^2$ ...
- ... leads to weight updates of $\frac{\partial E}{\partial w} = (a-t) \cdot f'(z)$
- ... and $f'(z)$ is very small!
- We plot the learning progress over the iterations: How fast does the neuron move towards the desired output 0?



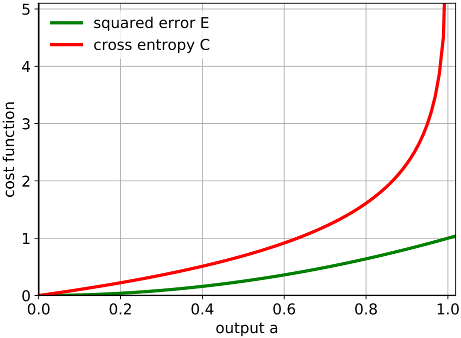


Trick 1: Cross-Entropy Cost

- **Idea**: Our **cost** must **compensate** for small values of f'
- Use the **Cross Entropy** as cost *(see Chapter 02)*

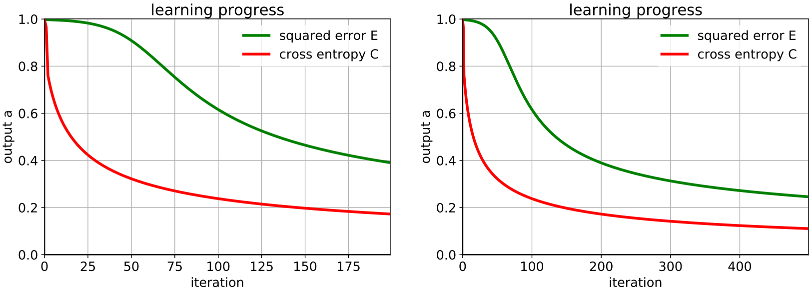
$$C(\mathbf{a}^L) = -\sum_k t_k \cdot log(a_k^L) + (1 - t_k) \cdot log(1 - a_k^L)$$
$$= -log(1-a) \quad // \text{ in our case}$$

- C penalizes our 'far off' neuron **much stronger**!



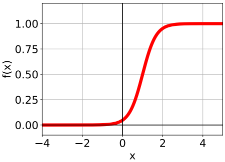
Trick 1: Cross-Entropy Cost

With cross-entropy, our neuron learns much faster!

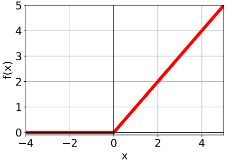


Trick 2: Rectified Linear Units (RELUs)

sigmoid activation



RELU activation



Backpropagation works with RELUs just like with sigmoids - just with a different f' term.

Sigmoid

- learning slows down for small and large inputs

Rectified Linear Unit

- learning is fast for positive inputs
- the neuron stops learning entirely for negative inputs
- (much) more efficient computation

Trick 2: Rectified Linear Units (RELUs)

Practical Advice

- The input of RELU neurons should *tend to be* larger than zero $\rightarrow$ initialize with a slightly higher bias!
- Let $a_1, a_2, \ldots, a_n$ be the outputs of a RELU layer. If we want them to be **scaled to $[0, 1]$** (say, in classification), we simply rescale the RELU unit's output using a so-called *softmax*

$$\left(a_1, a_2, \ldots, a_n\right) \mapsto \left(\frac{e^{a_1}}{\sum_i e^{a_i}}, \frac{e^{a_2}}{\sum_i e^{a_i}}, \ldots, \frac{e^{a_n}}{\sum_i e^{a_i}}\right)$$

Example

$$\begin{aligned} 1,\quad 3,\quad 1,\quad 7 &\mapsto 2\%,\ 11\%,\ 2\%,\ 85\% \\ -3,\quad 0,\quad 0.5,\quad -15 &\mapsto 2\%,\ 37\%,\ 61\%,\ 0\% \end{aligned}$$

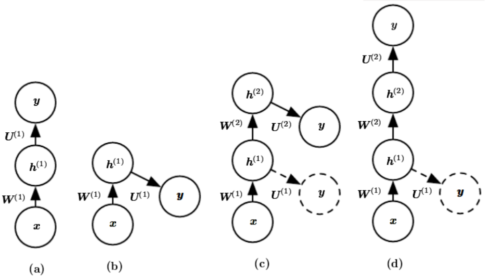
Remarks

- RELU activations have been vital to image recognition [10, 11]
- "We do not yet have a solid theory of how activation functions should be chosen." [15]

Advanced Techniques image from [5]

More Complicated Ways to Facilitate Deep Learning

- pretraining: start training with a simple network, then add **incremental layers** [5]
- **linear (sub-)paths** through the network *(prevent the gradient from dying off)*
- **skip connections** bypassing several layers
- adding extra **copies of the output** to early layers [19] *(makes the lowest layers receive a large gradient)*



Outline

Convolution for Images

We view images as **discrete 2D signals** $s : \mathbb{Z} \times \mathbb{Z} \to \{1, ..., M\}$.
Filters transform images s into other images s'. We focus on a particular kind of filter: **FIR (finite-impulse-response) filters**:

Definition (FIR Filter for Images)

Let s be a (2D) signal (i.e., an image), $M \in \mathbb{N}$, and

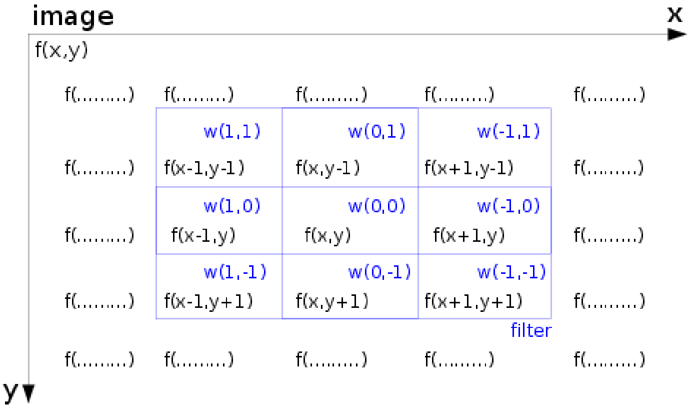
$$\begin{pmatrix} w_{-M,-M} & \dots & w_{-M,0} & \dots & w_{-M,M} \\ \dots & & & & \\ w_{0,-M} & \dots & w_{0,0} & \dots & w_{0,M} \\ \dots & & & & \\ w_{M,-M} & \dots & w_{M,0} & \dots & w_{M,M} \end{pmatrix}$$

be a **filter mask**. *Then, an* **finite impulse response filter** *computes:*

$$s'(x, y) = \sum_{u=-M}^{M} \sum_{v=-M}^{M} s(x-u, y-v) \cdot w_{u,v}$$

Convolution for Images

- We place the mask at every position of the image
- We compute the **weighted sum** of the pixel intensities, weighted by the mask's values



Convolution for Images: Example 1

image: Christoph Lampert

The **mean filter blurs** the input image

$$\begin{pmatrix} w_{-2,-2} & w_{-2,-1} & w_{-2,0} & w_{-2,1} & w_{-2,2} \\ w_{-1,-2} & w_{-1,-1} & w_{-1,0} & w_{-1,1} & w_{-1,2} \\ w_{0,-2} & w_{0,-1} & w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,-2} & w_{1,-1} & w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,-2} & w_{2,-1} & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix} = \frac{1}{25} \cdot \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

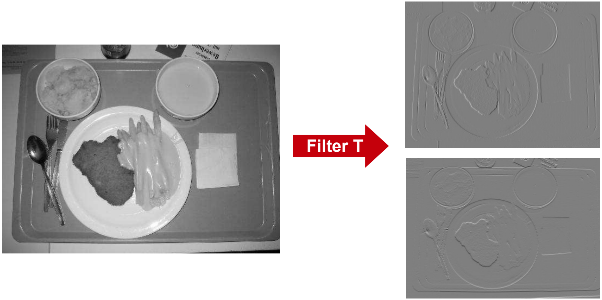
Filter T

Convolution for Images: Example 2

What do these Filters do?

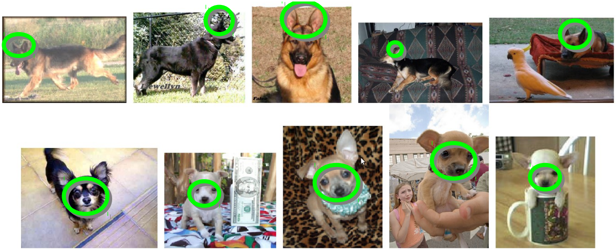
$$\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

These are the **Sobel filters**: They are commonly used to compute the partial derivatives $\frac{\partial s(x,y)}{\partial x}$, $\frac{\partial s(x,y)}{\partial y}$ of an image *(which indicate the* **edges** *of an image)*



Traditional Use of Convolution/Filters image from [4]

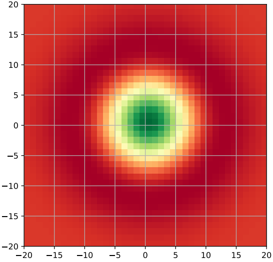
Key Idea: Even when images from the same class are not **globally** similar, they share certain **local characteristics**



Approach: Hand-engineer Filters to detect Local Features

- robust to changes of illumination, pose, background, ...
- state-of-the-art until 2011 *(and still used frequently today)*
- **SIFT**, SURF, HoG, Canny, ORB, ...
- *more in Chapter 03*

Step 1: Local Feature Detection

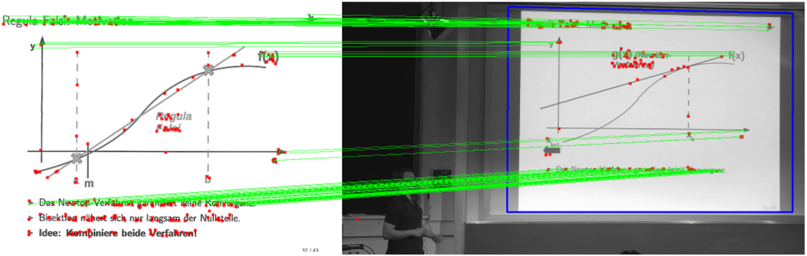


Example: The DoG ("difference-of-Gaussians") filter detects **blobs** *(dark regions surrounded by a bright background)*

- There are other detectors for corners, edges, etc.
- We usually apply filters of multiple sizes *(→ scale invariance)*

Step 2: Local Feature Matching image from [7]

After detecting local features, we *match* them to recognize objects



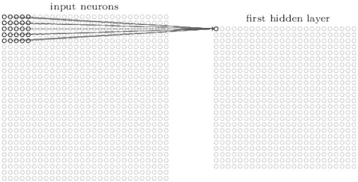
Filters in Neural Networks

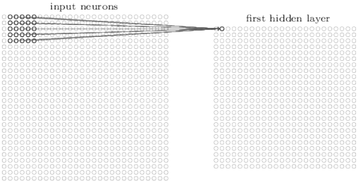
- By carefully designing the filter mask, we can scan the image for certain **features**
- Here, I designed a mask to detect the T-junction in the “4”.
- The result is called a **feature map**

Filters in Neural Networks

- **Layer 1**: run feature detectors over the image
- **Layer 2**: classify based on which features have been detected
- **This way, neural networks can learn their filters by backpropagation!**
- We call them **convolutional neural networks** (CNNs)

Convolutional Layers images from [15]





Convolutional Layers apply Filters

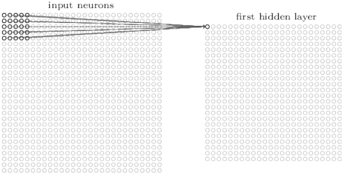
- the **input neurons** are the input image's pixels
- the **hidden neurons** (1st layer) are the feature map's pixels
- the **weights** are the entries of the (say, 5×5) filter mask
- the activation of neuron (or pixel) (j, k) in the feature map is

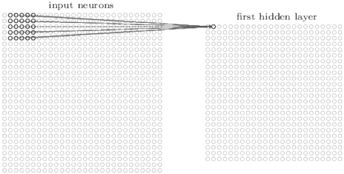
$$a_{jk} = f\left(b + \sum_{u=-2}^{2} \sum_{v=-2}^{2} w_{uv} \cdot x_{j+u,k+v}\right)$$

- short for the whole image *(with the convolution operator $*$)*:

$$\mathbf{a} = f\left(\mathbf{b} + (\mathbf{W} * \mathbf{x})\right)$$

Convolutional Layers images from [15]

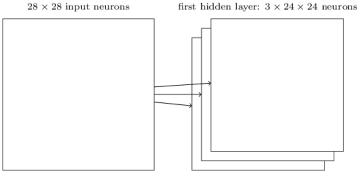




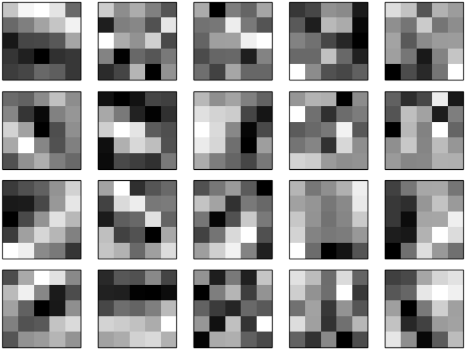
Discussion

- CNNs need far less weights: With a 28×28 input image and 24×24 output map, the number of weights is:
  - **fully connected layer**: $28 \times 28 \times 24 \times 24 \;\; (+24^2) \approx 450,000$
  - **convolutional layer**: $5 \times 5 \;\; (+1) = 26$
- This is called **weight sharing**, and it's great:
  less parameters $\rightarrow$ **less overfitting**!
- Convolutional neurons have a limited **receptive field** (e.g., 5×5) $\rightarrow$ instead of detecting **global** features, convolutional neurons detect **local** features.

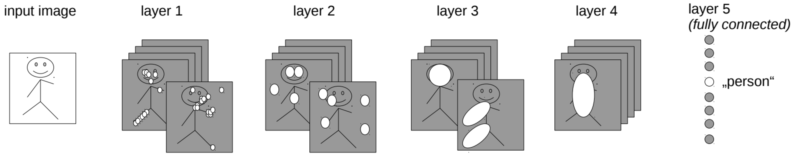
Convolutional Layers images from [15]



- Because we require far less weights, we can spend them on **multiple feature maps**!
- CNNs use **hundreds** of filters per layer.
- Some example of feature masks learned from MNIST data

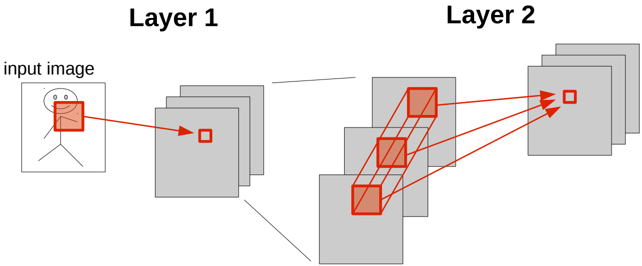


CNNs: Layer Stacking



- A **single convolutional layer** is quite **limited**:
  Its receptive fields are tiny and prone to noise.
- Idea: Feed feature maps to a **subsequent layer**, which constructs more complex features *($\rightarrow$ abstraction)*
- **Multiple layers**: edges $\rightarrow$ pupils $\rightarrow$ eyes $\rightarrow$ faces $\rightarrow$ persons
- With increasing layers ...
  - ... the **level of abstraction** increases
  - ... the **accuracy of localization** decreases

CNNs: Layer Stacking



- The second layer has not one input image, but multiple ones (namely, the feature maps from the first layer)!
- A **neuron n in the second layer** should be allowed to **combine** inputs from multiple feature maps of Layer 1
- Solution: **n** can access **all feature maps** within a local area, i.e. **n**'s local receptive field has size $5 \times 5 \times 20$:

$$a_{jk}^{p+1} = f\left(b + \sum_{u=-2}^{2} \sum_{v=-2}^{2} \sum_{f=1}^{20} w_{uvf} \cdot a_{j+u,k+v,f}^{p}\right)$$

CNNs: Layer Stacking

Example

- Layer 1 takes a 28×28 input image and filters it with 20 masks of size 5×5, obtaining 20 feature maps.
- Note: with a 5×5 convolution, the image reduces to 24×24 *(the filter mask must fit image)*.
- We add a second convolutional layer to the CNN

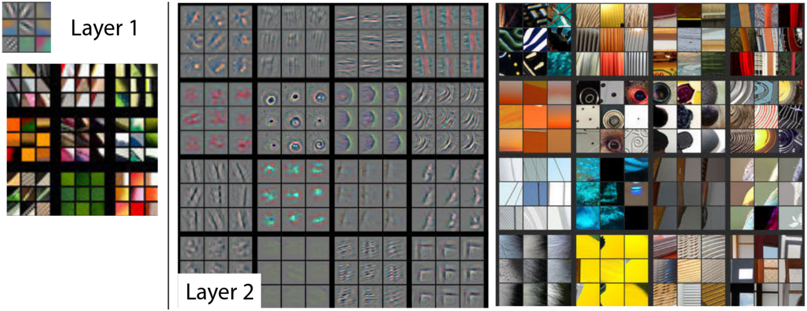
Layer	dims(in)	mask	#filters	dims(out)
1	28×28	5×5	20	$24 \times 24 \times 20$
2	$24 \times 24 \times 20$	$5 \times 5 \times 20$	30	$20 \times 20 \times 30$

Remarks

- Each layer's feature maps form a "3D matrix" (or **tensor**)
- This is why Google's deep learning library is called **tensorflow**.

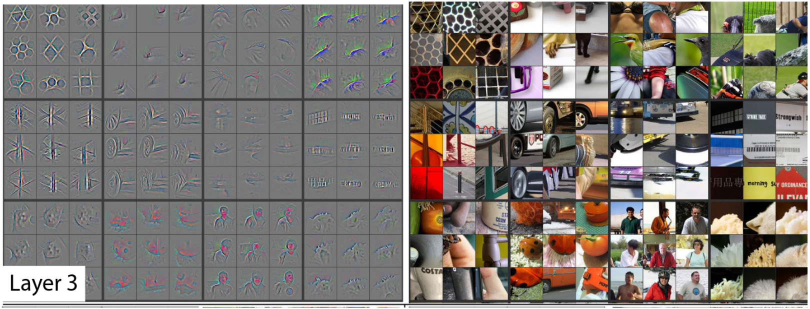
CNNs: Layer Stacking → Visualization images from [20]

- A CNN trained on 1000 object categories with 1,3 mio. images
- We **visualize** the **features** the CNN has learned, by …
  - … feeding the network input images
  - … recording the strongest activation in a given layer
  - … projecting this activation back to pixel space using *deconvolution*
- We start with Layers 1 and 2 …



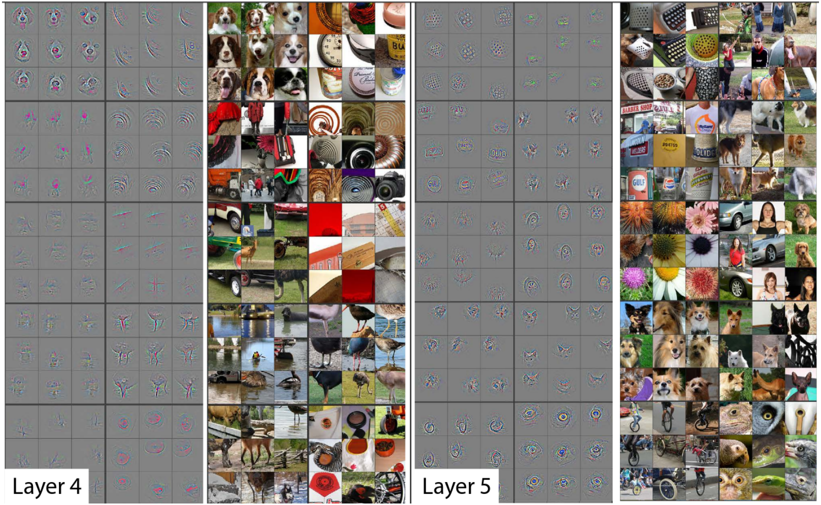
CNNs: Layer Stacking → Visualization images from [20]

- … and continue with Layers 3 …



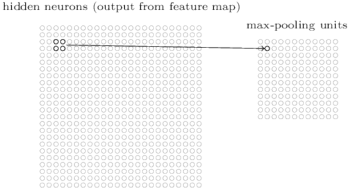
CNNs: Layer Stacking → Visualization images from [20]

- ... to Layers 4 and 5.



CNNs: Pooling Layers image from [15]

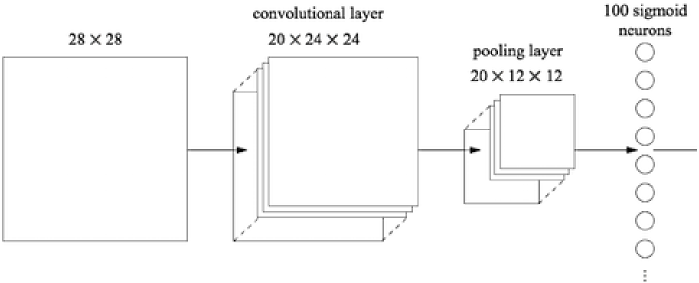
- We introduce **pooling** layers between the convolutional layers
- These **scale down** the feature maps *(it is enough to know roughly where a feature occurs)*.



Variations of Pooling

- **Max-Pooling**: take the maximum activation of the feature detector in the receptive field.
- **L2-Pooling**: take the L2 norm of the activations in the receptive field

CNNs: Minimal Architecture image from [15]

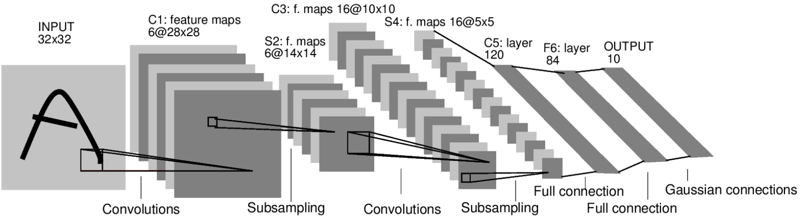


Remarks

- This CNN can be trained using plain backpropagation
  (see [9] for details)
- For **convolutional layers**, the error Δw_{uvf} is collected from all pixels in the output mask.
- For **pooling layers**, the error is just forwarded to the exact pixel where it came from.

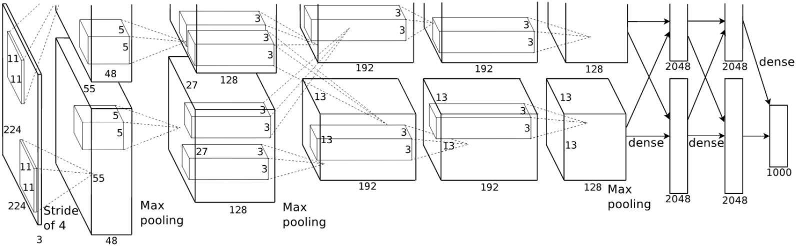
Outline

Example (Object Recognition): LeNet image from [12]

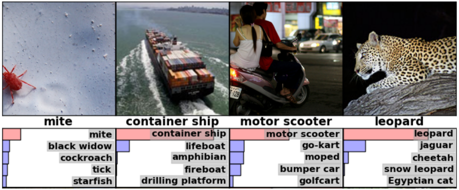


- 341K connections but only 90K parameters *(weight sharing)*
- applied to handwriting recognition
  (Demo: `http://yann.lecun.com`*)*
- 1998 (when SVMs were the method of choice...)

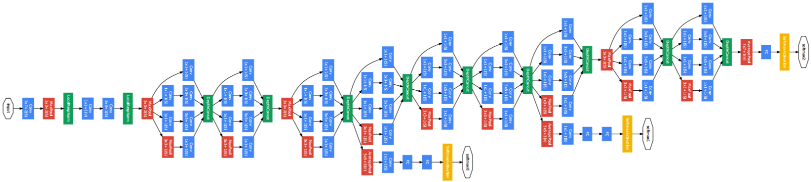
Example (Object Recognition): AlexNet images from [11], [2]



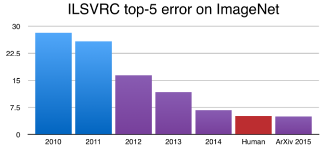
- **key trigger** for deep learning boom
- Layers: 5 $\times$ convolution, 3 FC layers, RELUs, dropout
- **GPU** implementation, network partitioned *(did not fit 1 GPU)*
- outstanding **winner of ILSVRC'12** *(top-5-error: 15.3%, second-best: 26.2%)*



Example (Object Recognition): GoogLeNet images from [19] [3]



- **increased depth** (22 layers) and width of network
- but: 12 × fewer **parameters** than AlexNet *(1×1 convolutions)*
- Codename: *Inception (a network within a network)*
- **human-level object recognition** (ILSVRC: 6.8% top-5-error)
- A. Karpathy: *I sat down and went through the [...] careful annotation process myself. [...] I became very good at identifying breeds of dogs. [...] My own error in the end turned out to be 5.1%.*



Transfer Learning

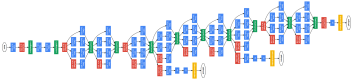
> *"Transfer learning is the improvement of learning in a* **new task** *through the* **transfer of knowledge** *from a* **related task** *that has already been learned."*
>
> (L. Torrey, J. Shavlik)

- Deep Learning allows us to train strong, complex models on large-scale training sets
- Key question: Can I adapt existing models to new domains *(where little training data may be available)?*

Examples

- I have trained a deep network for **keyword detection** on Wikipedia. Can I apply that to my customer's E-Mails?
- Can I reuse GoogLeNet (trained on cars, cats, dogs, etc.) to identify **other objects**?

Transfer Learning with GoogLeNet



- Recall GoogLeNet's architecture: multiple convolutional layers (f_{conv}), followed by a fully-connected layer + softmax (f_{class})

$$
\begin{array}{ccccc}
\mathbb{R}^{224\times 224} & \xrightarrow{f_{conv}} & \mathbb{R}^{1024} & \xrightarrow{f_{class}} & \mathbb{R}^{1000} \\
\mathbf{x} & \mapsto & \mathbf{x}' & \mapsto & \mathbf{y} \\
\text{(input image)} & & \text{(bottleneck layer)} & & \text{(classes)}
\end{array}
$$

- We can think of f_{conv} as a **very elaborate feature transformation**: $\mathbf{x}'$ is a 1024-dimensional feature representing the image.
- We call $\mathbf{x}'$ the **bottleneck layer**.
- $\mathbf{x}'$ is highly adapted to the classification problem GoogLeNet has been trained on: Its features are very helpful to recognize cats, dogs, cars, etc.!

Transfer Learning with GoogLeNet

Training

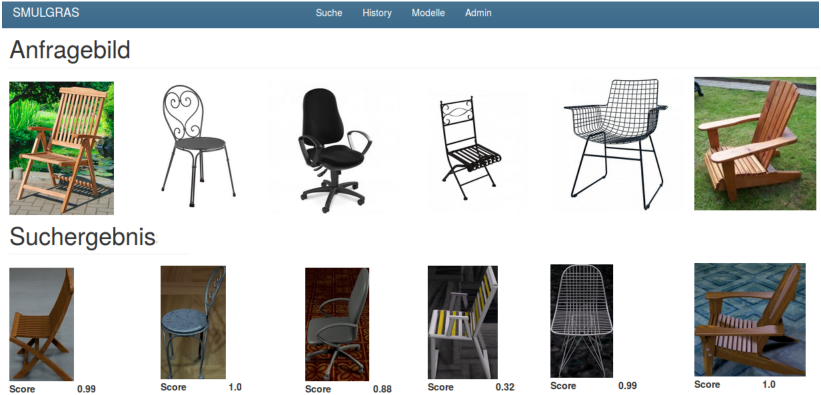
- We want to recognize 200 **new objects** (say, chairs). Of each, we have 100 training images.
- We apply the **convolutional layers** f_{conv} to all images
- We cache the resulting **bottleneck vectors** $\mathbf{x}'$
- We train a **new** (1-layer!) **classification layer** f'_{class} on those bottlenecks
- During training, errors are **not** propagated back into the convolutional layers. **Only the last layer** is trained.

Application

- Given a new image $\mathbf{x}$, its classification result is $f'_{class}(f_{conv}(\mathbf{x}))$
- This means: We use GoogleNet and simply replace the final layer with a 'chair-specific' one!

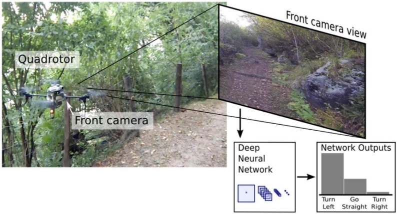
Transfer Learning with GoogLeNet: Results[1]

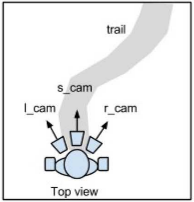
- transfer learning on **3D CG models** of chairs *(200 views each)*
- test photos of chairs *similar* to a 3D model



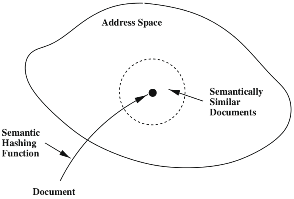
[1] Nadja Kurz, "Ein CNN zur view-basierten 3D-Modell-Suche", Bachelor's Thesis, HSRM, 2016.

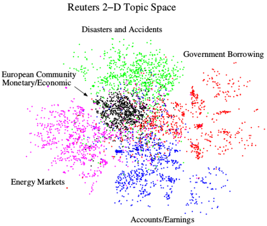
Image Classification Example: Path Following images from [8]

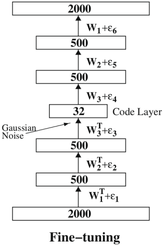




Text Compression Example: Semantic Hashing images from [17]







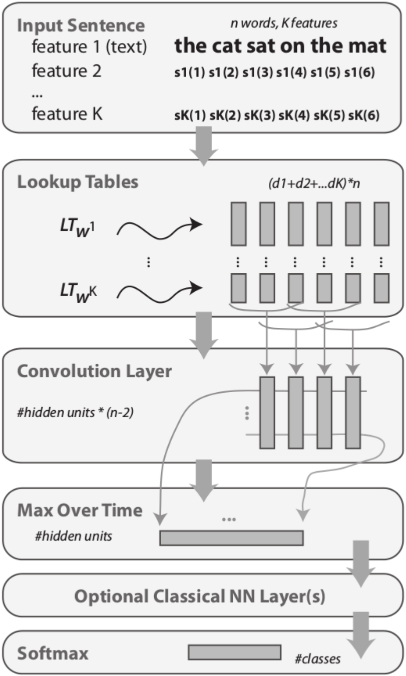
- Neural networks for (text) information retrieval
- Multiple layers of **Restricted Boltzmann Machines** (RBMs), trained incrementally
- **Learning problem**: **Compress** high-dimensional bag-of-words vectors to 32 bits, and **reconstruct** the original data
- Retrieval quality with 32-bit vectors about as good as **(tf-idf) bag-of-words**.

Example: Term Embeddings image: [6]

"You shall know a word by the company it keeps"

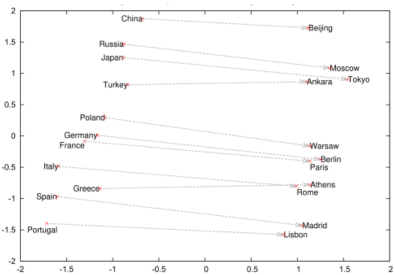
(J.R.Firth (1957))

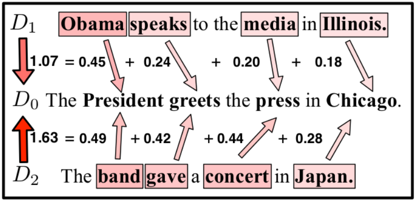
- **Stage 1 (Unsupervised)**: Context-based **prediction** of words. Given its neighbors, predict a word *(or given a word, predict its neighbors)*.
- **Stage 1 (Supervised)**: Classification of text subsequences
  - part-of-speech tagging (`noun vs. verb`)
  - named entity recognition (`person vs. company`)
  - selantic role labeling (`subject vs. object`)
  - synonym prediction



Example: Term Embeddings image: [6]

Byproduct: Term-level Feature Vectors





- Terms t are mapped to high-dimensional feature vectors $p(t)$
- relations between terms become shifts in vector space

$$p(uncle'') - p(''man'') + p(''woman'') \approx p(''aunt'')$$

- works for syntactic and semantic relations
- allows smarter machine learning on texts

References I

[1] Google DeepDream robot: 10 weirdest images produced by AI 'inceptionism' and users online (Photo: Reuters).
http://www.straitstimes.com/asia/east-asia/alphago-wins-4th-victory-over-lee-se-dol-in-final-go-match (retrieved: Nov 2016).

[2] Image Recognition (Tensorflow tutorial).
https://www.tensorflow.org/versions/r0.11/tutorials/image_recognition/index.html (retrieved: Nov 2016).

[3] Mocha.jl: Deep Learning for Julia.
https://devblogs.nvidia.com/parallelforall/mocha-jl-deep-learning-julia/ (retrieved: Nov 2016).

[4] picture shared by Christoph Lampert.
contact: http://pub.ist.ac.at/~chl/.

[5] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle.
Greedy Layer-Wise Training of Deep Networks.
In Advances in Neural Information Processing Systems 19, pages 153–160. 2007.

[6] R. Collobert and J. Weston.
A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning.
In Ann. Conf. on Neural Information Processing Systems (NIPS), 2008.

[7] M. Eberts.
Automatisierte Indexierung der Videoaufnahmen von Vorträgen mittels Bildmatching.
Bachelor's Thesis, RheinMain University of Applied Sciences, 2014.

[8] A. G. et al.
A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots.
IEEE Robotics and Automation Letters, 1(2), July 2016.

References II

[9] A. Gibiansky.
Convolutional Neural Networks (blog post).
http://andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks/ (retrieved: Nov 2016).

[10] X. Glorot, A. Bordes, and Y. Bengio.
Deep Sparse Rectifier Neural Networks.
In Proc. AISTATS-11, volume 15, pages 315–323, 2011.

[11] A. Krizhevsky, I. Sutskever, and G. Hinton.
ImageNet Classification with Deep Convolutional Neural Networks.
In Advances in Neural Information Processing Systems 25, pages 1097–1105. 2012.

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner.
Gradient-Based Learning Applied to Document Recognition.
In Proceedings of the IEEE, volume 86, pages 2278–2324, 1998.

[13] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean.
Distributed Representations of Words and Phrases and their Compositionality.
In Advances in Neural Information Processing Systems 26, pages 3111–3119. Curran Associates, Inc., 2013.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis.
Human-level control through deep reinforcement learning.
Nature, 518(7540):529–533, 02 2015.

[15] M. Nielsen.
Neural Networks and Deep Learning.
Determination Press, 2015.

References III

[16] M.-A. Russon.
Google DeepDream robot: 10 weirdest images produced by AI 'inceptionism' and users online.
http://www.ibtimes.co.uk/google-deepdream-robot-10-weirdest-images-produced-by-ai-inceptionism-users-online-1509518
(retrieved: Nov 2016).

[17] R. Salakhutdinov and G. Hinton.
Semantic Hashing.
International Journal of Approximate Reasoning, 50, 2009.

[18] C. Szegedy.
Building a deeper understanding of images (Google Research Blog).
https://research.googleblog.com/2014/09/building-deeper-understanding-of-images.html
(retrieved: Nov 2016).

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich.
Going Deeper with Convolutions.
In Computer Vision and Pattern Recognition (CVPR), 2015.

[20] M. D. Zeiler and R. Fergus.
Visualizing and Understanding Convolutional Networks.
CoRR, abs/1311.2901, 2013.