

# Understanding Convolutional Neural Networks

David Stutz

July 24th, 2014

# Table of Contents

- 1 Motivation
- 2 Neural Networks and Network Training
  - Multilayer Perceptrons
  - Network Training
  - Deep Learning
- 3 Convolutional Networks
- 4 Understanding Convolutional Networks
  - Deconvolutional Networks
  - Visualization
- 5 Conclusion

# Table of Contents

- 1 Motivation
- 2 Neural Networks and Network Training
  - Multilayer Perceptrons
  - Network Training
  - Deep Learning
- 3 Convolutional Networks
- 4 Understanding Convolutional Networks
  - Deconvolutional Networks
  - Visualization
- 5 Conclusion

# Motivation

Convolutional networks represent specialized networks for application in computer vision:

- ▶ they accept images as raw input (preserving spatial information),
- ▶ and build up (learn) a hierarchy of features (no hand-crafted features necessary).

Problem: Internal workings of convolutional networks not well understood ...

- ▶ Unsatisfactory state for evaluation and research!

Idea: Visualize feature activations within the network ...

# Motivation

Convolutional networks represent specialized networks for application in computer vision:

- ▶ they accept images as raw input (preserving spatial information),
- ▶ and build up (learn) a hierarchy of features (no hand-crafted features necessary).

Problem: Internal workings of convolutional networks not well understood ...

- ▶ Unsatisfactory state for evaluation and research!

Idea: Visualize feature activations within the network ...

# Table of Contents

- 1 Motivation
- 2 Neural Networks and Network Training
  - Multilayer Perceptrons
  - Network Training
  - Deep Learning
- 3 Convolutional Networks
- 4 Understanding Convolutional Networks
  - Deconvolutional Networks
  - Visualization
- 5 Conclusion

# Multilayer Perceptrons

A multilayer perceptron represents an adaptable model  $y(\cdot, w)$  able to map  $D$ -dimensional input to  $C$ -dimensional output:


$$y(\cdot, w) : \mathbb{R}^D \rightarrow \mathbb{R}^C, x \mapsto y(x, w) = \begin{pmatrix} y_1(x, w) \\ \vdots \\ y_C(x, w) \end{pmatrix}. \quad (1)$$

In general, a  $(L + 1)$ -layer perceptron consists of  $(L + 1)$  layers, each layer  $l$  computing linear combinations of the previous layer  $(l - 1)$  (or the input).

## Multilayer Perceptrons – First Layer

On input  $x \in \mathbb{R}^D$ , layer  $l = 1$  computes a vector  $y^{(1)} := (y_1^{(1)}, \dots, y_{m^{(1)}}^{(1)})$  where

$$y_i^{(1)} = f(z_i^{(1)}) \quad \text{with } z_i^{(1)} = \sum_{j=1}^D w_{i,j}^{(1)} x_j + w_{i,0}^{(1)}. \quad (2)$$

  $i^{\text{th}}$  component is called “unit  $i$ ”

where  $f$  is called activation function and  $w_{i,j}^{(1)}$  are adjustable weights.



# Multilayer Perceptrons – First Layer

What does this mean?

Layer  $l = 1$  computes linear combinations of the input and applies an (non-linear) activation function ...

The first layer can be interpreted as generalized linear model:

$$y_i^{(1)} = f \left( \left( w_i^{(1)} \right)^T x + w_{i,0}^{(1)} \right). \quad (3)$$

Idea: Recursively apply  $L$  additional layers on the output  $y^{(1)}$  of the first layer.

## Multilayer Perceptrons – Further Layers

In general, layer  $l$  computes a vector  $y^{(l)} := (y_1^{(l)}, \dots, y_{m^{(l)}}^{(l)})$  as follows:

$$y_i^{(l)} = f\left(z_i^{(l)}\right) \quad \text{with } z_i^{(l)} = \sum_{j=1}^{m^{(l-1)}} w_{i,j}^{(l)} y_j^{(l-1)} + w_{i,0}^{(l)}. \quad (4)$$

Thus, layer  $l$  computes linear combinations of layer  $(l - 1)$  and applies an activation function ...

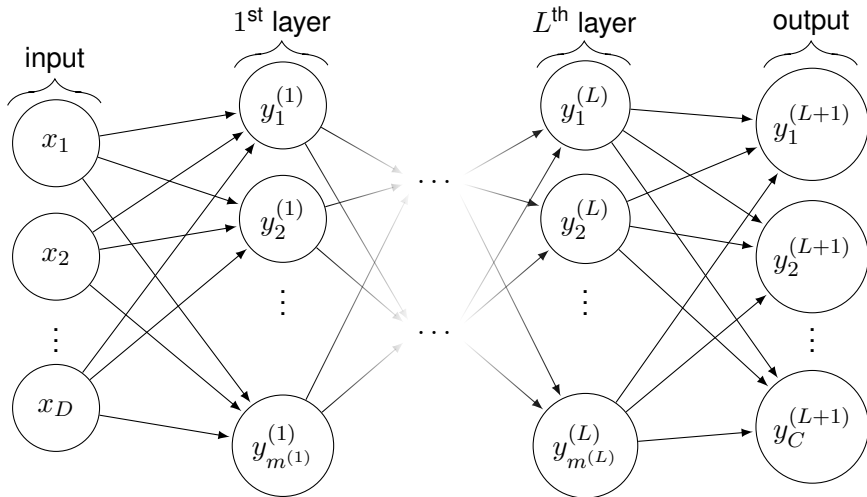
## Multilayer Perceptrons – Output Layer

Layer  $(L + 1)$  is called output layer because it computes the output of the multilayer perceptron:

$$y(x, w) = \begin{pmatrix} y_1(x, w) \\ \vdots \\ y_C(x, w) \end{pmatrix} := \begin{pmatrix} y_1^{(L+1)} \\ \vdots \\ y_C^{(L+1)} \end{pmatrix} = y^{(L+1)} \quad (5)$$

where  $C = m^{(L+1)}$  is the number of output dimensions.

## Network Graph



# Activation Functions – Notions

How to choose the activation function  $f$  in each layer?

- ▶ Non-linear activation functions will increase the expressive power: Multilayer perceptrons with  $L + 1 \geq 2$  are universal approximators [HSW89]!
- ▶ Depending on the application: For classification we may want to interpret the output as posterior probabilities:

$$y_i(x, w) \stackrel{!}{=} p(c = i|x) \quad (6)$$

where  $c$  denotes the random variable for the class.

# Activation Functions – Notions

How to choose the activation function  $f$  in each layer?

- ▶ Non-linear activation functions will increase the expressive power: Multilayer perceptrons with  $L + 1 \geq 2$  are universal approximators [HSW89]!
- ▶ Depending on the application: For classification we may want to interpret the output as posterior probabilities:

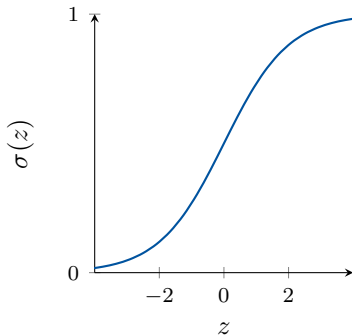
$$y_i(x, w) \stackrel{!}{=} p(c = i|x) \quad (6)$$

where  $c$  denotes the random variable for the class.

# Activation Functions

Usually the activation function is chosen to be the logistic sigmoid:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



which is non-linear, monotonic and differentiable.

# Activation Functions

Alternatively, the hyperbolic tangent is used frequently:

$$\tanh(z). \quad (7)$$

For classification with  $C > 1$  classes, layer  $(L + 1)$  uses the softmax activation function:

$$y_i^{(L+1)} = \sigma(z^{(L+1)}, i) = \frac{\exp(z_i^{(L+1)})}{\sum_{k=1}^C \exp(z_k^{(L+1)})}. \quad (8)$$

Then, the output can be interpreted as posterior probabilities.



## Activation Functions

Alternatively, the hyperbolic tangent is used frequently:

$$\tanh(z). \quad (7)$$

For classification with  $C > 1$  classes, layer  $(L + 1)$  uses the softmax activation function:

$$y_i^{(L+1)} = \sigma(z^{(L+1)}, i) = \frac{\exp(z_i^{(L+1)})}{\sum_{k=1}^C \exp(z_k^{(L+1)})}. \quad (8)$$

Then, the output can be interpreted as posterior probabilities.

# Network Training – Notions

By now, we have a general model  $y(\cdot, w)$  depending on  $W$  weights.

Idea: Learn the weights to perform

- ▶ regression,
- ▶ or classification.

We focus on classification.

## Network Training – Training Set

Given a training set

$C$  classes:

1-of- $C$  coding scheme

$$U_S = \{(x_n, t_n) : 1 \leq n \leq N\}, \quad (9)$$

learn the mapping represented by  $U_S$  ...

by minimizing the squared error

$$E(w) = \sum_{n=1}^N E_n(w) = \sum_{n=1}^N \sum_{i=1}^C (y_i(x_n, w) - t_{n,i})^2 \quad (10)$$

using iterative optimization.

## Network Training – Training Set

Given a training set

$C$  classes:

1-of- $C$  coding scheme


$$U_S = \{(x_n, t_n) : 1 \leq n \leq N\}, \quad (9)$$

learn the mapping represented by  $U_S$  ...

by minimizing the squared error

$$E(w) = \sum_{n=1}^N E_n(w) = \sum_{n=1}^N \sum_{i=1}^C (y_i(x_n, w) - t_{n,i})^2 \quad (10)$$

using iterative optimization.

# Training Protocols

We distinguish ...

**Stochastic Training** A training sample  $(x_n, t_n)$  is chosen at random, and the weights  $w$  are updated to minimize  $E_n(w)$ .

**Batch and Mini-Batch Training** A set  $M \subseteq \{1, \dots, N\}$  of training samples is chosen and the weights  $w$  are updated based on the cumulative error  $E_M(w) = \sum_{n \in M} E_n(w)$ .

Of course, online training is possible, as well.

# Training Protocols

We distinguish ...

**Stochastic Training** A training sample  $(x_n, t_n)$  is chosen at random, and the weights  $w$  are updated to minimize  $E_n(w)$ .

**Batch and Mini-Batch Training** A set  $M \subseteq \{1, \dots, N\}$  of training samples is chosen and the weights  $w$  are updated based on the cumulative error  $E_M(w) = \sum_{n \in M} E_n(w)$ .

Of course, online training is possible, as well.

# Training Protocols

We distinguish ...

**Stochastic Training** A training sample  $(x_n, t_n)$  is chosen at random, and the weights  $w$  are updated to minimize  $E_n(w)$ .

**Batch and Mini-Batch Training** A set  $M \subseteq \{1, \dots, N\}$  of training samples is chosen and the weights  $w$  are updated based on the cumulative error  $E_M(w) = \sum_{n \in M} E_n(w)$ .

Of course, online training is possible, as well.

# Iterative Optimization

Problem: How to minimize  $E_n(w)$  (stochastic training)?

- ▶  $E_n(w)$  may be highly non-linear with many poor local minima.

Framework for iterative optimization: Let ...

- ▶  $w[0]$  be an initial guess for the weights (several initialization techniques are available),
- ▶ and  $w[t]$  be the weights at iteration  $t$ .

In iteration  $[t + 1]$ , choose a weight update  $\Delta w[t]$  and set

$$w[t + 1] = w[t] + \Delta w[t] \quad (11)$$



# Iterative Optimization

Problem: How to minimize  $E_n(w)$  (stochastic training)?

- ▶  $E_n(w)$  may be highly non-linear with many poor local minima.

Framework for iterative optimization: Let ...

- ▶  $w[0]$  be an initial guess for the weights (several initialization techniques are available),
- ▶ and  $w[t]$  be the weights at iteration  $t$ .

In iteration  $[t + 1]$ , choose a weight update  $\Delta w[t]$  and set

$$w[t + 1] = w[t] + \Delta w[t] \quad (11)$$

# Gradient Descent

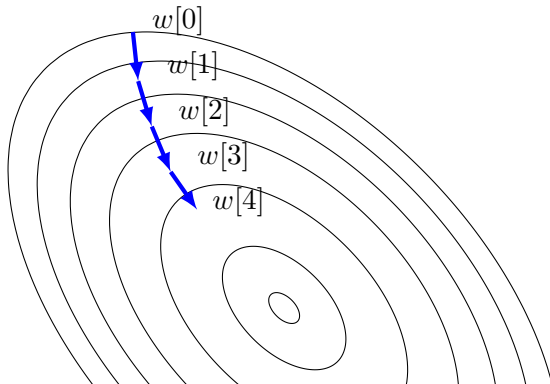
Remember:

Gradient descent minimizes the error  $E_n(w)$  by taking steps in the direction of the negative gradient:

$$\Delta w[t] = -\gamma \frac{\partial E_n}{\partial w[t]} \quad (12)$$

where  $\gamma$  defines the step size.

# Gradient Descent – Visualization



# Error Backpropagation

Problem: How to evaluate  $\frac{\partial E_n}{\partial w[t]}$  in iteration  $[t + 1]$ ?

- ▶ “Error Backpropagation” allows to evaluate  $\frac{\partial E_n}{\partial w[t]}$  in  $\mathcal{O}(W)$ !

Further details ...

- ▶ See the original paper “Learning Representations by Back-Propagating Errors,” by Rumelhart et al. [RHW86].

# Deep Learning

Multilayer perceptrons are called deep if they have more than three layers:  $L + 1 > 3$ .

Motivation: Lower layers can automatically learn a hierarchy of features or a suitable dimensionality reduction.

- ▶ No hand-crafted features necessary anymore!

However, training deep neural networks is considered very difficult!

- ▶ Error measure represents a highly non-convex, “potentially intractable” [EMB<sup>+</sup>09] optimization problem.

# Deep Learning

Multilayer perceptrons are called deep if they have more than three layers:  $L + 1 > 3$ .

Motivation: Lower layers can automatically learn a hierarchy of features or a suitable dimensionality reduction.

- ▶ No hand-crafted features necessary anymore!

However, training deep neural networks is considered very difficult!

- ▶ Error measure represents a highly non-convex, “potentially intractable” [EMB<sup>+</sup>09] optimization problem.

# Approaches to Deep Learning

Possible approaches:

- ▶ Different activation functions offer faster learning, for example

$$\max(0, z) \quad \text{or} \quad |\tanh(z)|; \quad (13)$$

- ▶ unsupervised pre-training can be done layer-wise;
- ▶ ...

Further details ...

- ▶ See “Learning Deep Architectures for AI,” by Y. Bengio [Ben09] for a detailed discussion of state-of-the-art approaches to deep learning.

# Summary

The multilayer perceptron represents a standard model of neural networks. They ...

- ▶ allow to tailor the architecture (layers, activation functions) to the problem;
- ▶ can be trained using gradient descent and error backpropagation;
- ▶ can be used for learning feature hierarchies (deep learning).

Deep learning is considered difficult.



# Table of Contents

- 1 Motivation
- 2 Neural Networks and Network Training
  - Multilayer Perceptrons
  - Network Training
  - Deep Learning
- 3 Convolutional Networks
- 4 Understanding Convolutional Networks
  - Deconvolutional Networks
  - Visualization
- 5 Conclusion

# Convolutional Networks

Idea: Allow raw image input while preserving the spatial relationship between pixels.

Tool: Discrete convolution of image  $I$  with filter  $K \in \mathbb{R}^{2h_1+1 \times 2h_2+1}$  is defined as

$$(I * K)_{r,s} = \sum_{u=-h_1}^{h_1} \sum_{v=-h_2}^{h_2} K_{u,v} I_{r+u,s+v} \quad (14)$$

where the filter  $K$  is given by

$$K = \begin{pmatrix} K_{-h_1,-h_2} & \dots & K_{-h_1,h_2} \\ \vdots & K_{0,0} & \vdots \\ K_{h_1,-h_2} & \dots & K_{h_1,h_2} \end{pmatrix}. \quad (15)$$

# Convolutional Networks

Idea: Allow raw image input while preserving the spatial relationship between pixels.

Tool: Discrete convolution of image  $I$  with filter  $K \in \mathbb{R}^{2h_1+1 \times 2h_2+1}$  is defined as

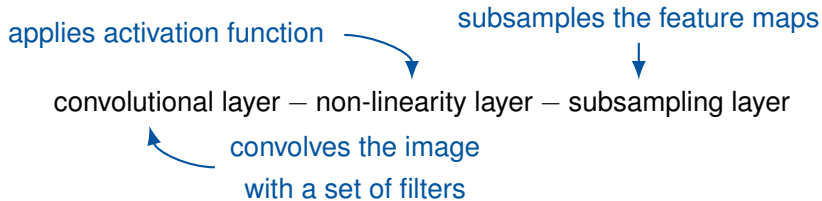
$$(I * K)_{r,s} = \sum_{u=-h_1}^{h_1} \sum_{v=-h_2}^{h_2} K_{u,v} I_{r+u,s+v} \quad (14)$$

where the filter  $K$  is given by

$$K = \begin{pmatrix} K_{-h_1,-h_2} & \dots & K_{-h_1,h_2} \\ \vdots & K_{0,0} & \vdots \\ K_{h_1,-h_2} & \dots & K_{h_1,h_2} \end{pmatrix}. \quad (15)$$

# Convolutional Networks – Architectures

Original Convolutional Network [LBD<sup>+</sup>89] aims to build up a feature hierarchy by alternating



followed by a multilayer perceptron for classification.

## Convolutional Layer – Notions

Central part of convolutional networks: convolutional layer.

- ▶ Can handle raw image input.

Idea: Apply a set of learned filters to the image in order to obtain a set of feature maps.

Can be repeated: Apply a different set of filters to the obtained feature maps to get more complex features:

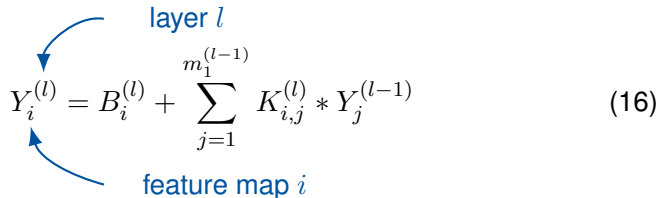
- ▶ Generate a hierarchy of feature maps.

# Convolutional Layer

Let layer  $l$  be a convolutional layer.

Input:  $m_1^{(l-1)}$  feature maps  $Y_i^{(l-1)}$  of size  $m_2^{(l-1)} \times m_3^{(l-1)}$  from the previous layer.

Output:  $m_1^{(l)}$  feature maps of size  $m_2^{(l)} \times m_3^{(l)}$  given by


$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} K_{i,j}^{(l)} * Y_j^{(l-1)} \quad (16)$$

where  $B_i^{(l)}$  is called bias matrix and  $K_{i,j}^{(l)}$  are the filters to be learned.

# Convolutional Layer – Notes

## Notes:

- ▶ The size  $m_2^{(l)} \times m_3^{(l)}$  of the output feature maps depends on the definition of discrete convolution (especially how borders are handled).
- ▶ The weights  $w_{i,j}^{(l)}$  are hidden in the bias matrix  $B_i^{(l)}$  and the filters  $K_{i,j}^{(l)}$ .

## Non-Linearity Layer

Let layer  $l$  be a non-linearity layer.

Given  $m_1^{(l-1)}$  feature maps, a non-linearity layer applies an activation function to all these feature maps:

$$Y_i^{(l)} = f\left(Y_i^{(l-1)}\right) \quad (17)$$

where  $f$  operates point-wise.

Usually,  $f$  is the hyperbolic tangent.

Layer  $l$  computes  $m_1^{(l)} = m_1^{(l-1)}$  feature maps unchanged in size ( $m_2^{(l)} = m_2^{(l-1)}$ ,  $m_3^{(l)} = m_3^{(l-1)}$ ).



# Subsampling and Pooling Layer

Motivation: Incorporate invariance to noise and distortions.

Idea: Subsample the feature maps of the previous layer.

Let layer  $l$  be a subsampling and pooling layer.

Given  $m_1^{(l-1)}$  feature maps of size  $m_2^{(l-1)} \times m_3^{(l-1)}$ , create  $m_1^{(l)} = m_1^{(l-1)}$  feature maps of reduced size.

- For example by placing windows at non-overlapping positions within the feature maps and keeping only the maximum activation per window.

# Putting it All Together

Remember: A convolutional network alternates

convolutional layer — non-linearity layer — subsampling layer

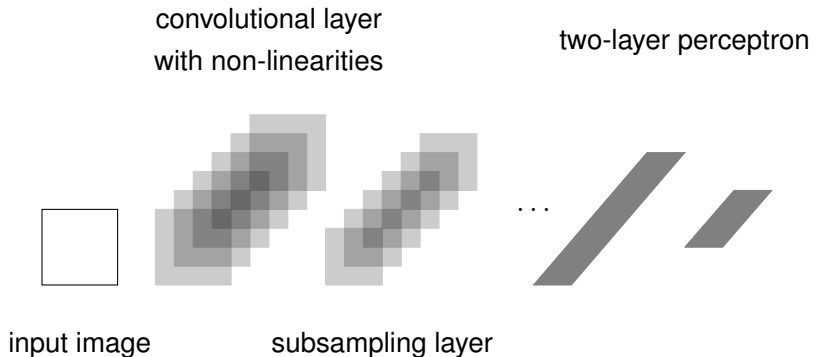
to build up a hierarchy of feature maps...

and uses a multilayer perceptron for classification.

Further details ...

- ▶ LeCun et al. [LKF10] and Jarrett et al. [JKRL09] give a review of recent architectures.

## Overall Architecture



## Additional Layers

Researchers are constantly coming up with additional types of layers ...

Example 1: Let layer  $l$  be a rectification layer.

Given feature maps  $Y_i^{(l-1)}$  of the previous layer, a rectification layer computes

$$Y_i^{(l)} = \left| Y_i^{(l-1)} \right| \quad (18)$$

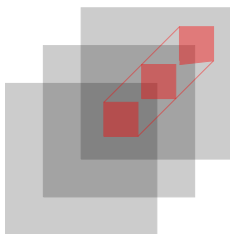
where the absolute value is computed point-wise.

Experiments show that rectification plays an important role to achieve good performance.

## Additional Layers (cont'd)

### Example 2:

Local contrast normalization layers aim to enforce local competitiveness between adjacent feature maps.



ensure that values  
are comparable

- ▶ There are different implementations available, see Krizhevsky et al. [KSH12] or LeCun et al. [LKF10].

# Summary

A basic convolutional network consists of different types of layers:

- ▶ convolutional layers;
- ▶ non-linearity layers;
- ▶ and subsampling layers.

Researchers are constantly thinking about additional types of layers to improve learning and performance.

# Table of Contents

- 1 Motivation
- 2 Neural Networks and Network Training
  - Multilayer Perceptrons
  - Network Training
  - Deep Learning
- 3 Convolutional Networks
- 4 Understanding Convolutional Networks
  - Deconvolutional Networks
  - Visualization
- 5 Conclusion

# Understanding Convolutional Networks

State: Convolutional networks perform well without requiring hand-crafted features.

- ▶ But: Learned feature hierarchy not well understood.

Idea: Visualize feature activations of higher convolutional layers ...

- ▶ Feature activations after first convolutional layer can be backprojected onto the image plane.

Zeiler et al. [ZF13] propose a visualization technique based on *deconvolutional* networks.



# Understanding Convolutional Networks

State: Convolutional networks perform well without requiring hand-crafted features.

- ▶ But: Learned feature hierarchy not well understood.

Idea: Visualize feature activations of higher convolutional layers ...

- ▶ Feature activations after first convolutional layer can be backprojected onto the image plane.

Zeiler et al. [ZF13] propose a visualization technique based on *deconvolutional* networks.

# Deconvolutional Networks

*Deconvolutional networks aim to build up a feature hierarchy ...*

- ▶ by convolving the input image by a set of filters – like convolutional networks;
- ▶ however, they are fully unsupervised.

Idea: Given an input image (or a set of feature maps), try to reconstruct the input given the filters and their activations.

Basic component: *deconvolutional* layer.

# Deconvolutional Layer

Let layer  $l$  be a *deconvolutional* layer.

Given feature maps  $Y_i^{(l-1)}$  of the previous layer, try to reconstruct the input using the filters and their activations:

$$Y_i^{(l-1)} \stackrel{!}{=} \sum_{j=1}^{m_1^{(l)}} \left( K_{j,i}^{(l)} \right)^T * Y_j^{(l)}. \quad (19)$$

*Deconvolutional* layers ...

- ▶ are unsupervised by definition;
- ▶ need to learn feature activations *and* filters.

# Deconvolutional Networks

Deconvolutional networks stack *deconvolutional* layers and are fully unsupervised.

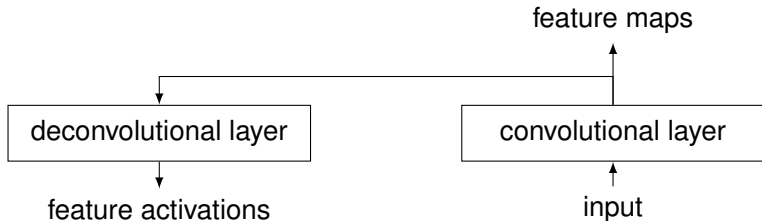
Further details ...

- ▶ See “Deconvolutional Networks,” by Zeiler et al. [ZKTF10] for details on how to train deconvolutional networks.

# Deconvolutional Layers for Visualization

Here: *Deconvolutional* layer used for visualization of trained convolutional network ...

- ▶ filters are already learned – no training necessary.



# Deconvolutional Layers for Visualization

Problem: Subsampling and pooling in higher layers.

Remember: Placing windows at non-overlapping positions within the feature maps, pooling is accomplished by keeping one activation per window.

Solution: Remember which pixels of a feature map were kept using so called “switch variables”.

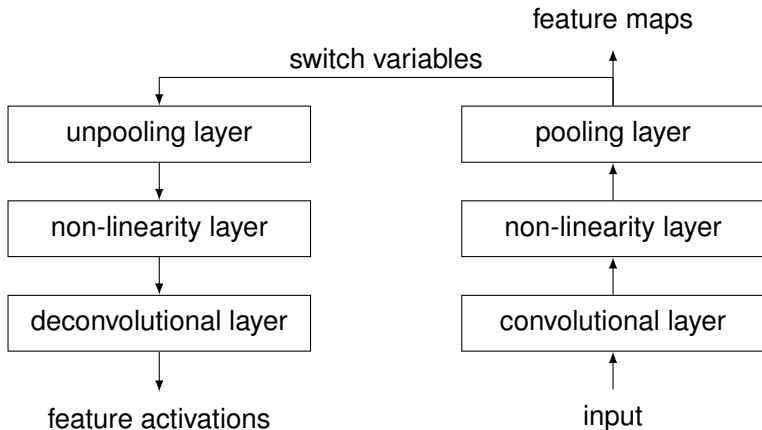
# Deconvolutional Layers for Visualization

Problem: Subsampling and pooling in higher layers.

Remember: Placing windows at non-overlapping positions within the feature maps, pooling is accomplished by keeping one activation per window.

Solution: Remember which pixels of a feature map were kept using so called “switch variables”.

## Deconvolutional Layers for Visualization





# Feature Activations

How does this look?

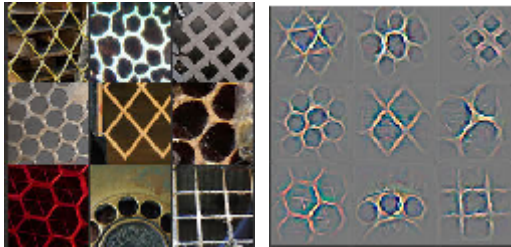
Examples in [ZF13]: Given a validation set, backproject a single activation within a feature map in layer  $l$  to analyze which structure excites this particular feature map.

Layer 1: Filters represent Gabor-like filters (for edge detection).

Layer 2: Filters for corners.

Layers above layer 2 are interesting ...

## Feature Activations (cont'd)

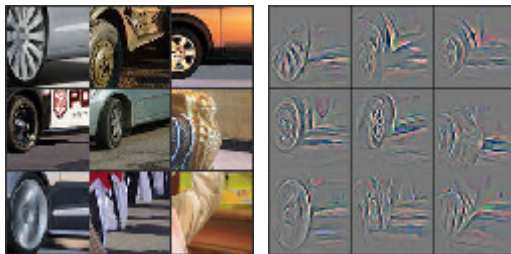


(a) Images.

(b) Activations.

Figure: Activations of **layer 3** backprojected to pixel level [ZF13].

## Feature Activations (cont'd)

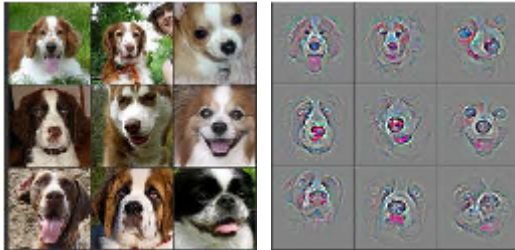


(a) Images.

(b) Activations.

Figure: Activations of **layer 3** backprojected to pixel level [ZF13].

## Feature Activations (cont'd)



(a) Images.

(b) Activations.

Figure: Activations of **layer 4** backprojected to pixel level [ZF13].

## Feature Activations (cont'd)



(a) Images.

(b) Activations.

Figure: Activations of **layer 4** backprojected to pixel level [ZF13].

# Table of Contents

- 1 Motivation
- 2 Neural Networks and Network Training
  - Multilayer Perceptrons
  - Network Training
  - Deep Learning
- 3 Convolutional Networks
- 4 Understanding Convolutional Networks
  - Deconvolutional Networks
  - Visualization
- 5 Conclusion

# Conclusion

Convolutional networks perform well in computer vision tasks as they learn a feature hierarchy.

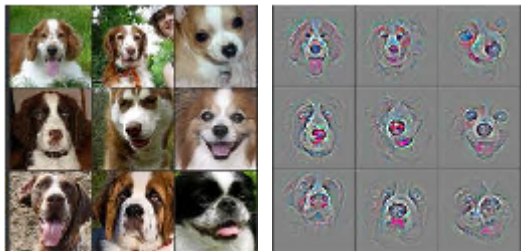
Internal workings of convolutional networks are not well understood.

- ▶ [ZF13] use *deconvolutional* networks to visualize feature activations;
- ▶ this allows to analyze the feature hierarchy and to increase performance.
  - ▶ For example by adjusting the filter size and subsampling scheme.

## The End

# Thanks for your attention!

Paper available at <http://davidstutz.de/seminar-paper-understanding-convolutional-neural-networks/>



## Questions?





Y. Bengio.

Learning deep architectures for AI.

*Foundations and Trends in Machine Learning*, (1):1–127, 2009.



C. Bishop.

Exact calculation of the hessian matrix for the multilayer perceptron.

*Neural Computation*, 4(4):494–501, 1992.



C. Bishop.

*Neural Networks for Pattern Recognition*.

Clarendon Press, Oxford, 1995.



C. Bishop.

*Pattern Recognition and Machine Learning*.

Springer Verlag, New York, 2006.



S. Becker and Y. LeCun.

Improving the convergence of back-propagation learning with second-order methods.

*In Connectionist Models Summer School*, pages 29–37, 1989.



Y. bengio and Y. LeCun.

Scaling learning algorithms towards AI.

*In Large Scale Kernel Machines.* MIT Press, 2007.



D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber.

Flexible, high performance convolutional neural networks for image classification.

*In Artificial Intelligence, International Joint Conference*, pages 1237–1242, 2011.



D. C. Ciresan, U. Meier, and J. Schmidhuber.

Multi-column deep neural networks for image classification.

*Computing Research Repository*, abs/1202.2745, 2012.



R. Duda, P. Hart, and D. Stork.

*Pattern Classification.*

Wiley-Interscience Publication, New York, 2001.



D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio.

Why does unsupervised pre-training help deep learning?

*Journal of Machine Learning Research*, 11:625–660, 2010.



D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent.

The difficulty of training deep architectures and the effect of unsupervised pre-training.

*In Artificial Intelligence and Statistics, International Conference on*, pages 153–160, 2009.



D. Forsyth and J. Ponce.

*Computer Vision: A Modern Approach*.

Prentice Hall Professional Technical Reference, New Jersey, 2002.



X. Glorot and Y. Bengio.

Understanding the difficulty of training deep feedforward neural networks.

*In Artificial Intelligence and Statistics, International Conference on*, pages 249–256, 2010.



X. Glorot, A. Bordes, and Y. Bengio.

Deep sparse rectifier neural networks.

*In Artificial Intelligence and Statistics, International Conference on*,  
pages 315–323, 2011.



P. Gill, W. Murray, and M. Wright.

*Practical optimization*.

Academic Press, London, 1981.



S. Haykin.

*Neural Networks A Comprehensive Foundation*.

Pearson Education, New Delhi, 2005.



G. E. Hinton and S. Osindero.

A fast learning algorithm for deep belief nets.

*Neural Computation*, 18(7):1527–1554, 2006.



G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and  
R. Salakhutdinov.

Improving neural networks by preventing co-adaptation of feature detectors.

*Computing Research Repository*, abs/1207.0580, 2012.



K. Hornik, M. Stinchcombe, and H. White.

Multilayer feedforward networks are universal approximators.

*Neural Networks*, 2(5):359–366, 1989.



K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun.

What is the best multi-stage architecture for object recognition?

In *Computer Vision, International Conference on*, pages 2146–2153, 2009.



K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun.

Fast inference in sparse coding algorithms with applications to object recognition.

*Computing Research Repository*, abs/1010.3467, 2010.



A. Krizhevsky, I. Sutskever, and G. E. Hinton.

ImageNet classification with deep convolutional neural networks.

In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.



Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner.

Gradient-based learning applied to document recognition.

*Proceedings of the IEEE*, 86:2278–2324, 1998.



Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel.

Backpropagation applied to handwritten zip code recognition.

*Neural Computation*, 1(4):541–551, 1989.



H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin.

Exploring strategies for training deep neural networks.

*Journal of Machine Learning Research*, 10:1–40, 2009.



Y. LeCun.

Generalization and network design strategies.


In *Connectionism in Perspective*, 1989.




Y. LeCun, K. Kavukcuoglu, and C. Farabet.

Convolutional networks and applications in vision.

*In Circuits and Systems, International Symposium on*, pages 253–256, 2010.

 S. J. Nowlan and G. E. Hinton.  
Simplifying neural networks by soft weight-sharing.  
*Neural Computation*, 4(4):473–493, 1992.

 D. E. Rumelhart, G. E. Hinton, and R. J. Williams.  
Parallel distributed processing: Explorations in the microstructure of cognition.  
chapter Learning Representations by Back-Propagating Errors,  
pages 318–362. MIT Press, Cambridge, 1986.

 F. Rosenblatt.  
The perceptron: A probabilistic model for information storage and organization in the brain.  
*Psychological Review*, 65, 1958.

 D. Scherer, A. Müller, and S. Behnke.

Evaluation of pooling operations in convolutional architectures for object recognition.

*In Artificial Neural Networks, International Conference on*, pages 92–101, 2010.



P. Y. Simard, D. Steinkraus, and J. C. Platt.

Best practices for convolutional neural networks applied to visual document analysis.

*In Document Analysis and Recognition, International Conference on*, 2003.



M. D. Zeiler and R. Fergus.

Visualizing and understanding convolutional networks.

*Computing Research Repository*, abs/1311.2901, 2013.



M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus.

Deconvolutional networks.

*In Computer Vision and Pattern Recognition, Conference on*, pages 2528–2535, 2010.