

# Künstliche Intelligenz / Maschinelles Lernen



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

## Tiefes Lernen — Deep Learning

### Teil 2 - Trainieren von faltenden Neuronalen Netzwerken



Basierende auf Folien von Viktoriia Sharmanska, Geoff Hinton, Fei-Fei Li und viele anderen.  
Danke fürs Offenlegen ihrer Folien



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT





# Trainieren des AlexNets: Überblick

---

- Große Datenmenge: ImageNet
- Zwei NVIDIA GTX 580 3GB GPUs
- Für circa eine Woche
- **Mittels stochastischem Gradientenabstieg und Backpropagation**



# ImageNet

- 15 Millionen Bilder
- 22.000 Kategorien / Labels
- Bilder sind aus dem Internet
- Menschen haben die Bilder annotiert (Amazons Mechanical Turk)
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2010)
  - 1.000 Kategorien
  - 1.2 Millionen Bilder zum Trainieren (~1000 pro Kategorie)
  - 50,000 Bilder zur Validierung
  - 150,000 Bilder zum Testen
- RGB Bilder; Durchschnittsbild wurde von allen Bildern abgezogen
- Unterschiedliche Auflösungen, aber AlexNet skaliert die Bilder auf 256x256



# ImageNet: Klassifikationsaufgabe

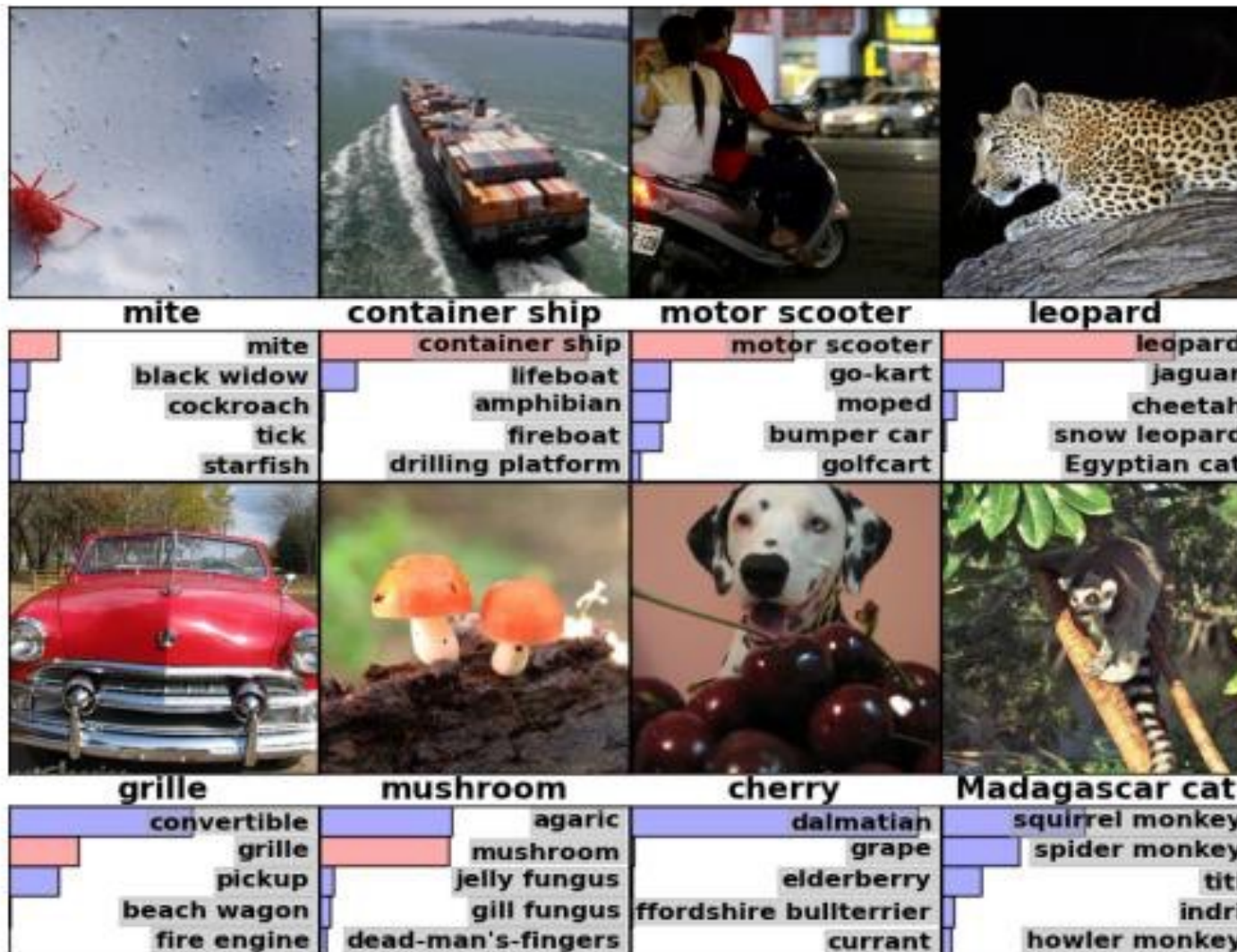
- Eine Vorhersage pro Bild (Top-1 error)
- Sage 5 Labels pro Bild vorher (Top-5 error)





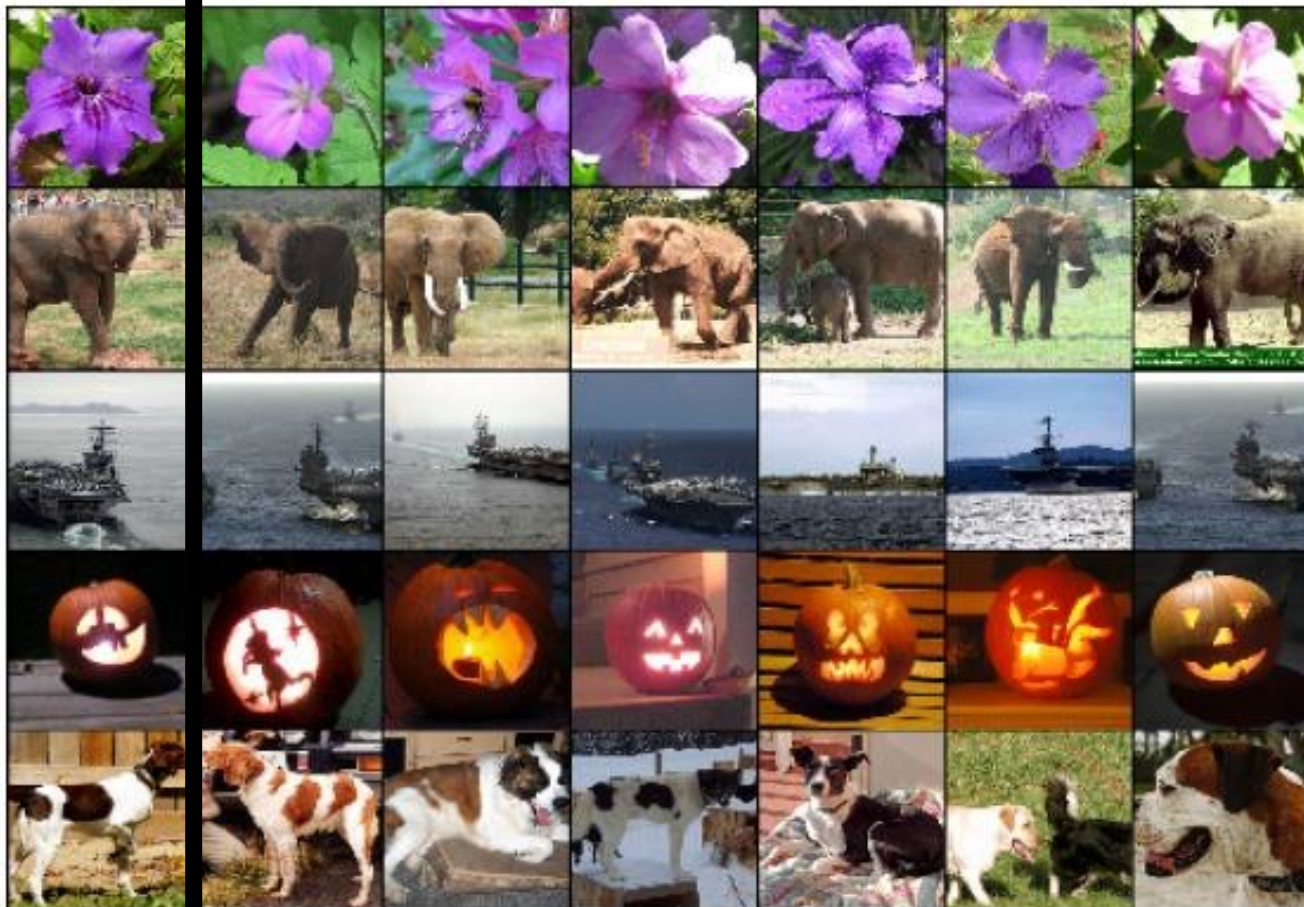


# ImageNet: Ergebnisse von AlexNet





# AlexNet: Suche von ähnlichen Bildern



Anfragebild

Sechs Trainingsbilder, die zu Merkmalsvektoren in der letzten, unbeobachteten Schicht führen, die den kleinsten Euklidischen Abstand zum Anfragebild hatten

# Das Trainieren von faltenden Neuronales Netzwerken (CNNs) im Detail

- Training
  - Stochastischer Gradientenabstieg
  - Backpropagation
  - Initialisierung
- Vermeidung von Überanpassung
  - Dropout Regularisierung
  - Datenaugmentierung
- Fine-tuning
- Visualisierung von CNNs



# Training CNNs

- ☐ Stochastischer Gradientenabstieg
- ☐ Backpropagation
- ☐ Initialisierung





# Stochastic gradient descent (SGD)

## (Mini-batch) SGD

Initialisierung der Parameter

Gehe über alle Daten (mehrmals):

- **Sample** einen (Batch von) Datenpunkt(en)
- **Vorwärtspropagierung** der Daten durch das Netzwerk, berechne die Klassifikationsgüte / den Verlust (loss).
- **Rückwärtspropagierung** des Gradientens der Parameter bezüglich der Verlustfunktion
- **Update** der Parameter mittels des Gradientens





# Stochastic gradient descent (SGD)

## (Mini-batch) SGD

Initialisierung der Parameter **zufällig aber auf intelligente Art**

Gehe über alle Daten (mehrmals):

- **Sample** einen (Batch von) Datenpunkt(en)
- **Vorwärtspropagierung** der Daten durch das Netzwerk, berechne die Klassifikationsgüte / den Verlust (loss). **Z.B.** 
$$E = \frac{1}{2} (y_{\text{predicted}} - y_{\text{true}})^2$$
- **Rückwärtspropagierung** des Gradientens der Parameter bezüglich der Verlustfunktion

- **Update** der Parameter mittels des Gradientens. **SGD**

$$w^{t+1} = w^t - \alpha \cdot \frac{dE}{dw}(w^t)$$





# Backpropagation

Backpropagation ist die rekursive Anwendung der Kettenregel zur Differenzierung entlang des Berechnungsgraphens des Netzwerkes zur Berechnung des Gradientes der Verlustfunktion nach allen Variablen, auch der Zwischenergebnisse

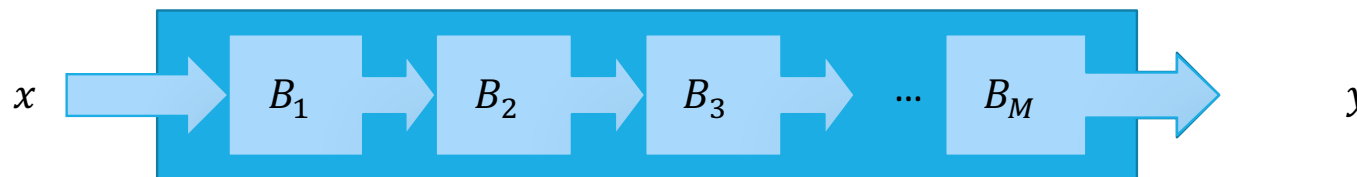
Die Lego-Architektur von tiefen Netzwerken spiegelt sich auch auf im Trainieren wider



# Backpropagation

Implementierungen sind modular aufgebaut. Knoten/ Leogbausteine implementieren den Vorwärts- und Rückwärtspropagierungen

## Sequential brick



### Propagation

- Apply propagation rule to  $B_1, B_2, B_3, \dots, B_M$ .

### Back-propagation

- Apply back-propagation rule to  $B_M, \dots, B_3, B_2, B_1$ .





# Backpropagation

Letzte Schicht implementiert die Klassifikation

## Square loss brick



Propagation

$$E = y = \frac{1}{2}(x - d)^2$$

Back-propagation

$$\frac{\partial E}{\partial x} = (x - d)^T \frac{\partial E}{\partial y} = (x - d)^T$$



# Backpropagation

Typische Legobausteine

## Loss bricks

		Propagation	Back-propagation
Square		$y = \frac{1}{2}(x - d)^2$	$\frac{\partial E}{\partial x} = (x - d)^T \frac{\partial E}{\partial y}$
Log	$c = \pm 1$	$y = \log(1 + e^{-cx})$	$\frac{\partial E}{\partial x} = \frac{-c}{1 + e^{cx}} \frac{\partial E}{\partial y}$
Hinge	$c = \pm 1$	$y = \max(0, m - cx)$	$\frac{\partial E}{\partial x} = -c \mathbb{I}\{cx < m\} \frac{\partial E}{\partial y}$
LogSoftMax	$c = 1 \dots k$	$y = \log(\sum_k e^{x_k}) - x_c$	$\left[\frac{\partial E}{\partial x}\right]_s = (e^{x_s} / \sum_k e^{x_k} - \delta_{sc}) \frac{\partial E}{\partial y}$
MaxMargin	$c = 1 \dots k$	$y = \left[ \max_{k \neq c} \{x_k + m\} - x_c \right]_+$	$\left[\frac{\partial E}{\partial x}\right]_s = (\delta_{sk^*} - \delta_{sc}) \mathbb{I}\{E > 0\} \frac{\partial E}{\partial y}$

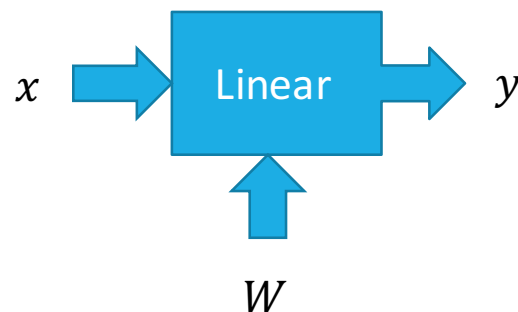




# Backpropagation

Vollständig-verbundene Schichte, Faltungsschichten (Skalarprodukt)

## Linear brick



Propagation

$$y = Wx$$

Back-propagation

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} W$$

$$\frac{\partial E}{\partial W} = x \frac{\partial E}{\partial v}$$





# Backpropagation

Nicht-lineare Aktivierungen

## Activation function brick



Propagation

$$y_s = f(x_s)$$

Back-propagation

$$\left[ \frac{\partial E}{\partial x} \right]_s = \left[ \frac{\partial E}{\partial y} \right]_s f'(x_s)$$





# Backpropagation

Typische Aktivierungsfunktionen

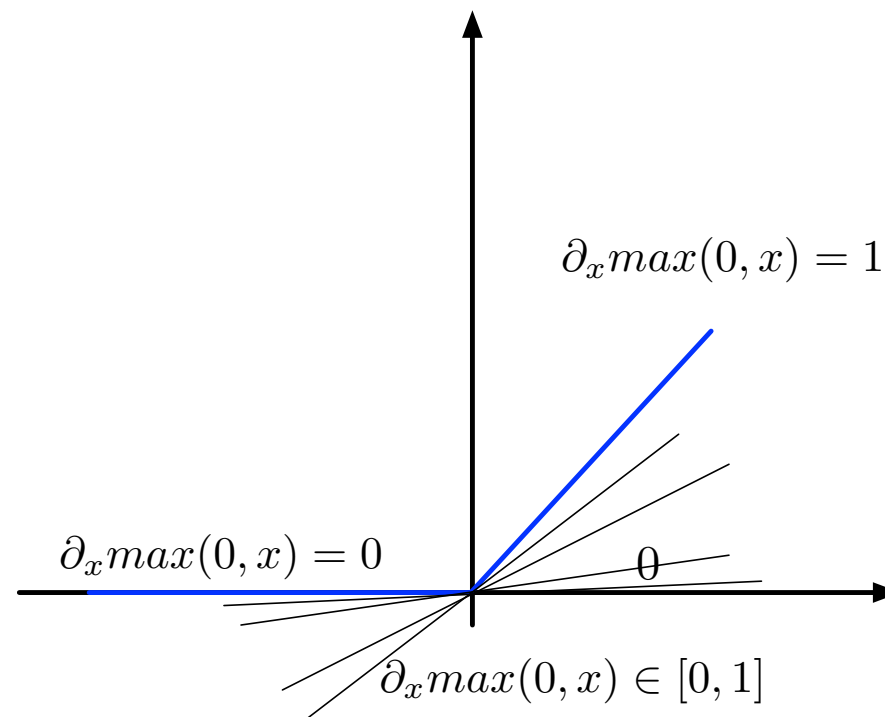
## Activation functions

	Propagation	Back-propagation
Sigmoid	$y_s = \frac{1}{1+e^{-x_s}}$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \frac{1}{(1+e^{x_s})(1+e^{-x_s})}$
Tanh	$y_s = \tanh(x_s)$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \frac{1}{\cosh^2 x_s}$
ReLu	$y_s = \max(0, x_s)$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \mathbb{I}\{x_s > 0\}$
Ramp	$y_s = \min(-1, \max(1, x_s))$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \mathbb{I}\{-1 < x_s < 1\}$



# Subgradienten

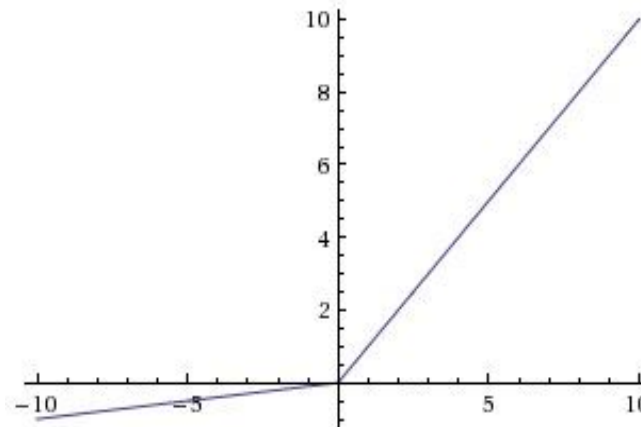
ReLU Gradient ist für  $x=0$  nicht definiert, daher benutzt man  
**Subgradient**





# Leaky ReLU

In der Praxis gerne benutzt: *Leaky ReLU*,  $f(x) = \max(0.01x, x)$   
Das vermeidet, dass der Gradient "saturiert", also überall gleich aussieht



## Leaky ReLU

$$f(x) = \max(0.01x, x)$$



# Training CNNs

- ☒ Stochastischer Gradientenabstieg
- ☒ Backpropagation
- ☐ Initialisierung





# Stochastic gradient descent (SGD)

## (Mini-batch) SGD

### Initialisierung der (Filter-)Gewichte

- nicht mit 0 initialisieren
- Nicht alle mit dem selben Wert initialisieren
- sample gleichmäßig  $U[-b,b]$  um Null herum oder von der Normalverteilung

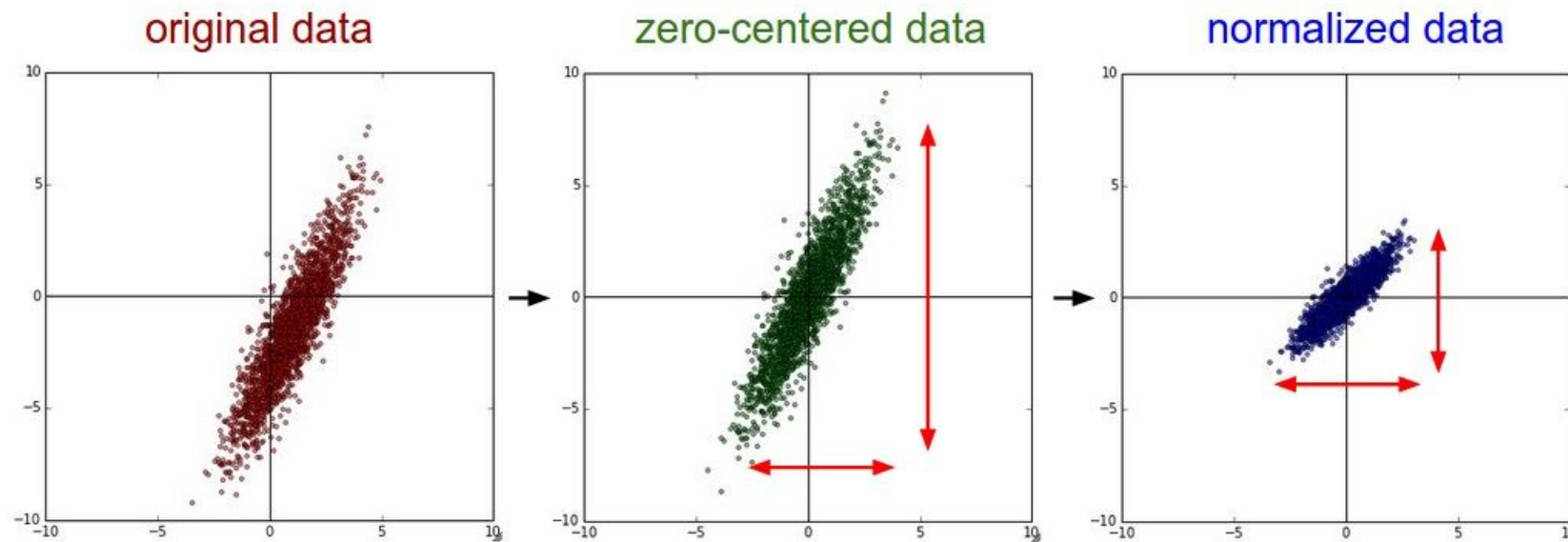
Abnahme (decay) der Schrittweite  $\alpha$    $w^{t+1} = w^t - \alpha \cdot \frac{dE}{dw}(w^t)$

Je näher wir am der “Lösung” sind, desto kleiner die Schrittweite

- Fange mit einer großen Schrittweite an (z.B. 0.1)
- Behalte diese bei, bis sich der Fehler auf der Validierungsmenge nicht mehr verbessert
- Halbiere die Schrittweite und verfolge wie zuvor

# Stochastic gradient descent (SGD)

## Normalisierung der Daten



Bei Bildern: Ziehe das Durchschnittsbild von allen Trainingsbildern ab.

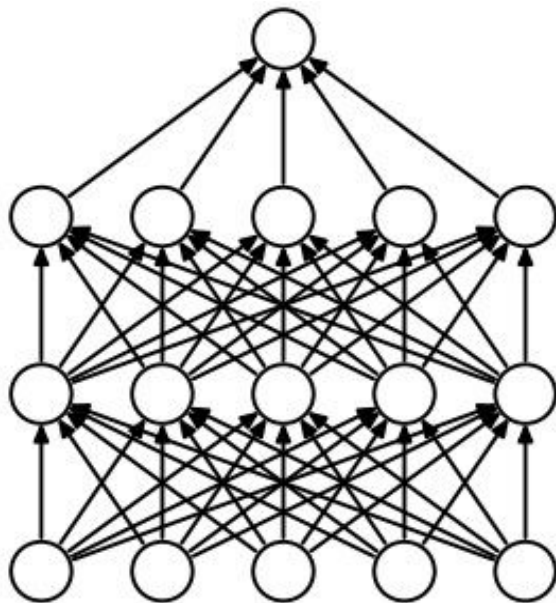


# Vermeide Überanpassung

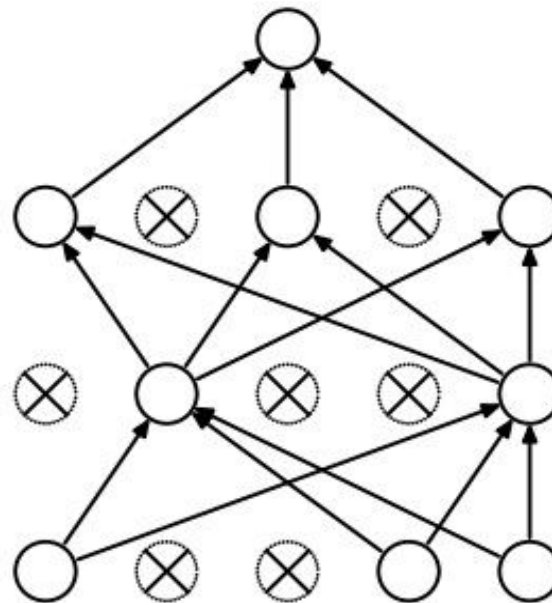
- ☐ Dropout regularization
- ☐ Data augmentation

# Regularisierung: DropOut

Dropout: „setze die Eingabe für einige Neuronen auf 0 (mit Wahrscheinlichkeit 0.5)“



(a) Standard Neural Net



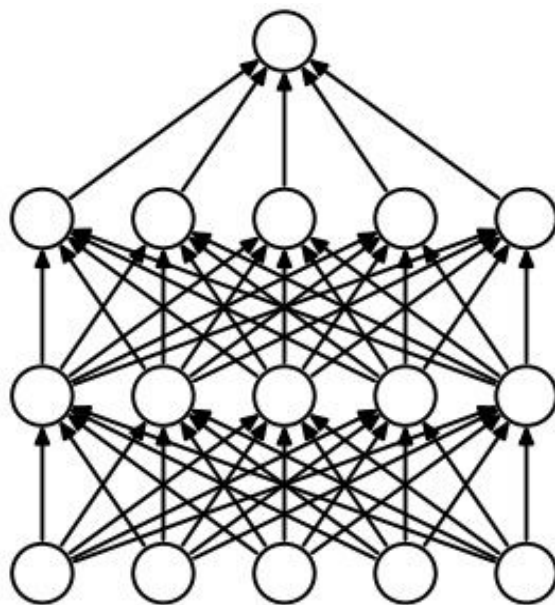
(b) After applying dropout.

*[Srivastava et al., 2014]*

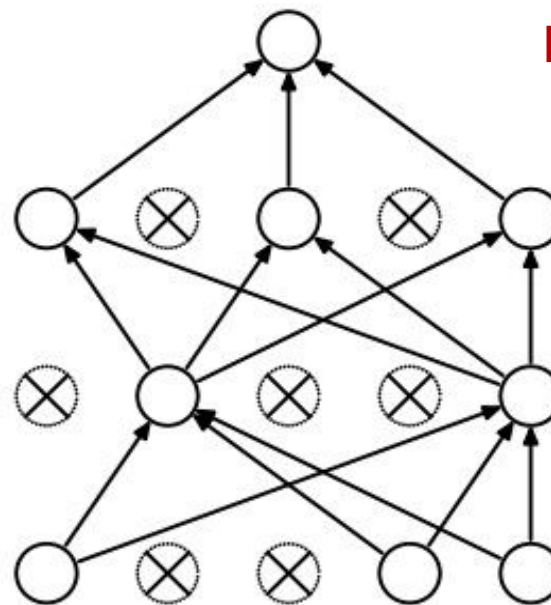


# Regularisierung: DropOut

Dropout: „setze die Eingabe für einige Neuronen auf 0 (mit Wahrscheinlichkeit 0.5). Die „dropped out“ Neuronen tragen nicht zur Vorhersage bei und werden auch nicht bei Backpropagation beachtet.



(a) Standard Neural Net



(b) After applying dropout.

**Es wird für jede Eingabe  
eine andere Netzwerk-  
Architektur benutzt.  
(Ensembles)**

**Zur Vorhersage auf der  
Testmenge benutze den  
Durchschnitt aller Modelle**

*[Srivastava et al., 2014]*



# Regularisierung: DropOut

Verringert komplexe Ko-Adaptionen von Neuronen; ein einzelnes Neuron kann sich nicht mehr auf andere andere Neuronen verlassen

Jedes Neuron muss robuste Merkmale lernen, die in einem komplexen Zusammenspiel mit anderen Neuronen gut funktionieren

Ohne DropOut zeigen CNNs starke Überanpassung

Aber DropOut verdoppelt (Pi\*Daumen) den Trainingsaufwand

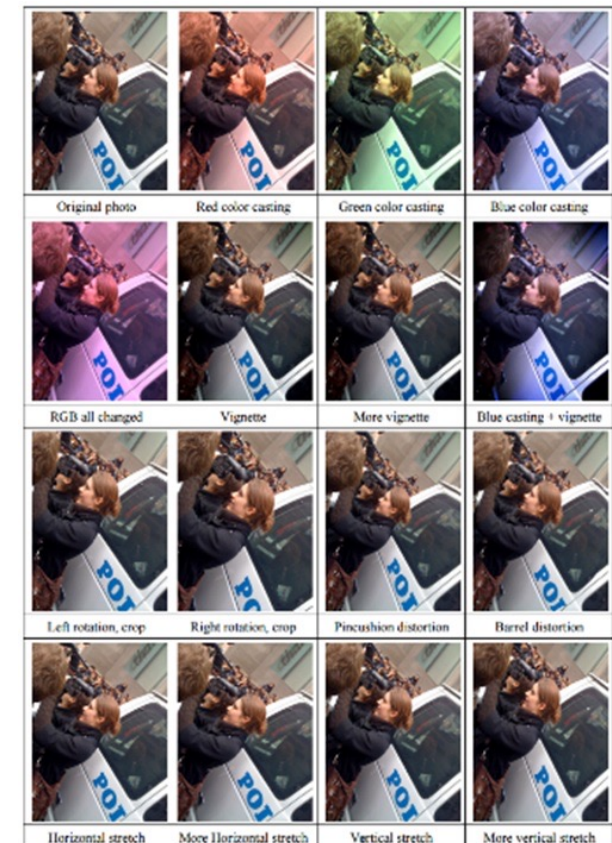
Alternative: Standard Regularisierungen wie z.B. L2-Regularisierung

# Regularisierung: Datenaugmentierung

Die einfachste Art der Regularisierung ist die Datenaugmentierung. Wir bereichern die Trainingsdaten mit vielen Varianten der Originaldaten an.

Typischerweise

- horizontale Spiegelungen
- Zufällige Ausschnitte
- Änderungen der RGB\_Werte
- Bildtransformationen



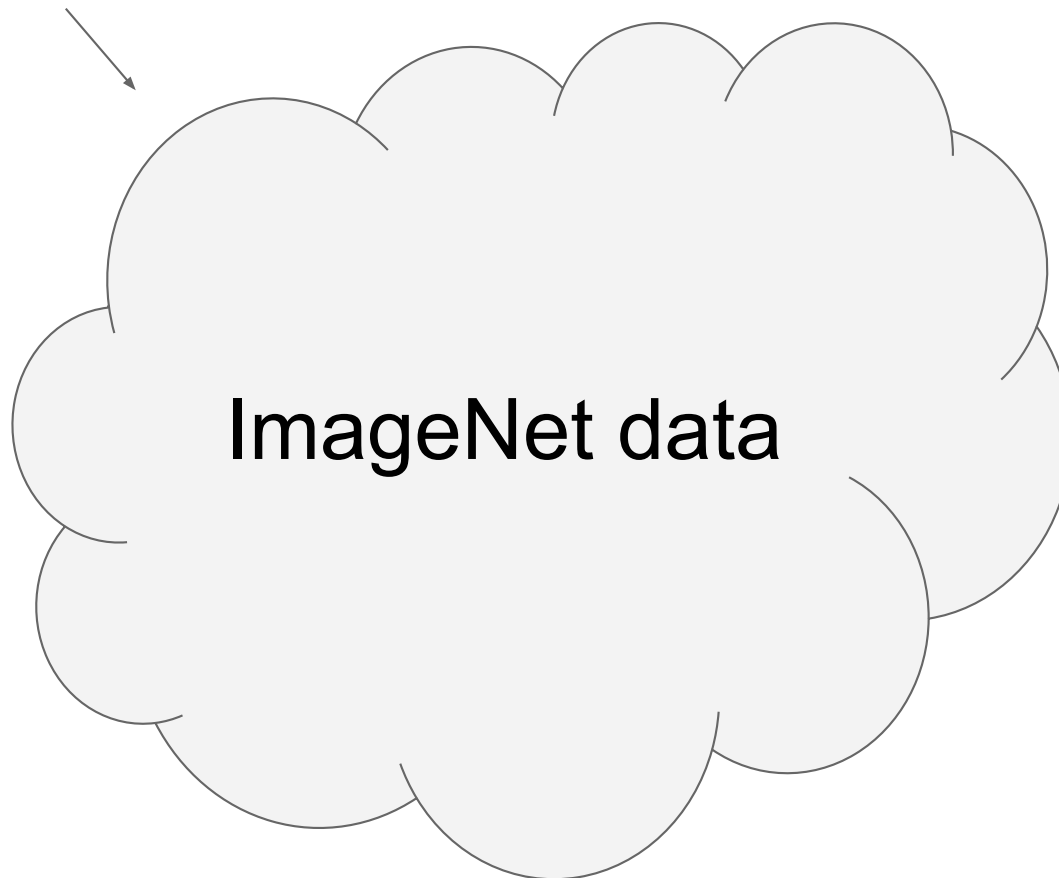


# Fine-Tuning

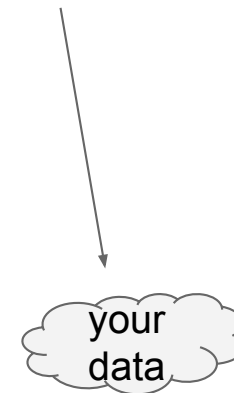


# Fine-Tuning

1. Train on ImageNet

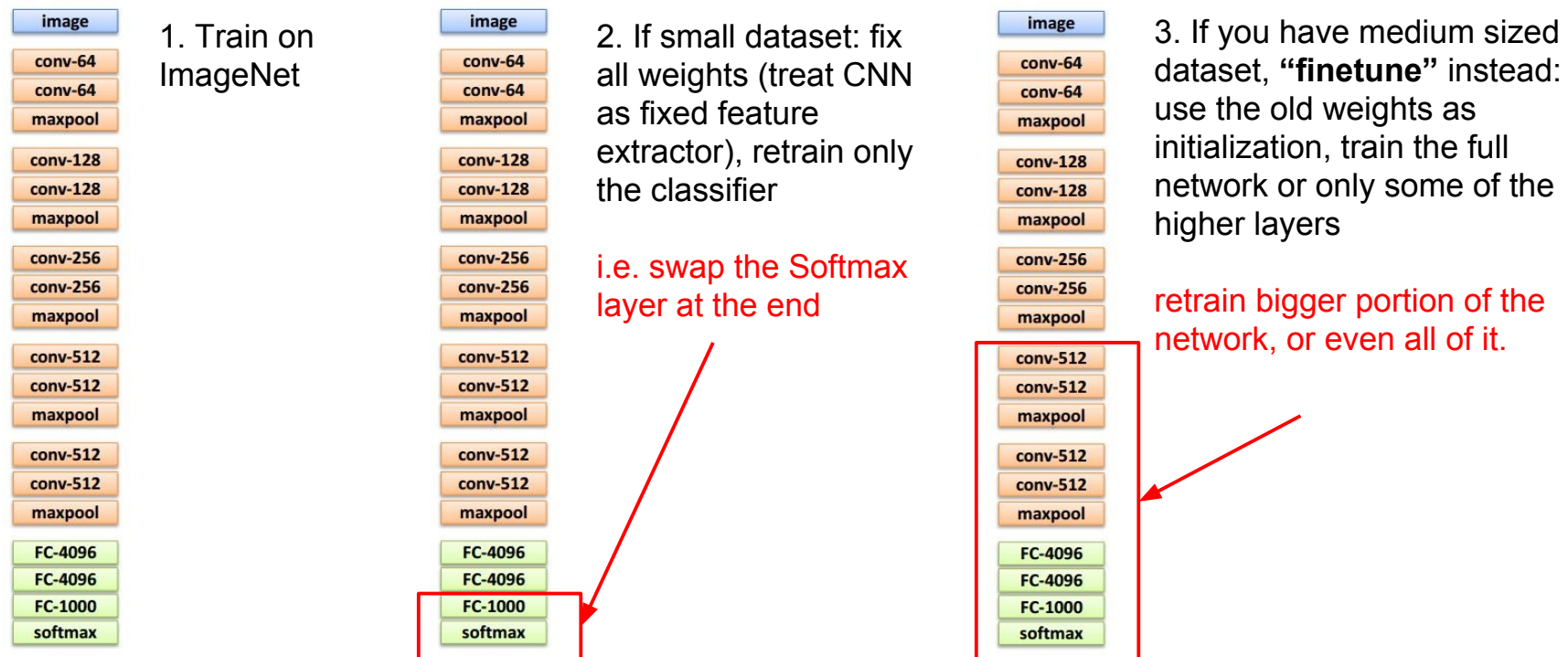


2. Finetune network on  
your own data



# Fine-Tuning

## Transfer Learning with CNNs



Im Internet findet man viele pre-trained Modelle



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

# Visualisierung



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT





# Erste Faltungsschicht

## AlexNet



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



96 Faltungskerne ( $11 \times 11 \times 3$ ) gelernt durch die erste Faltungsschicht auf  $227 \times 227 \times 3$  Eingabebildern.



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

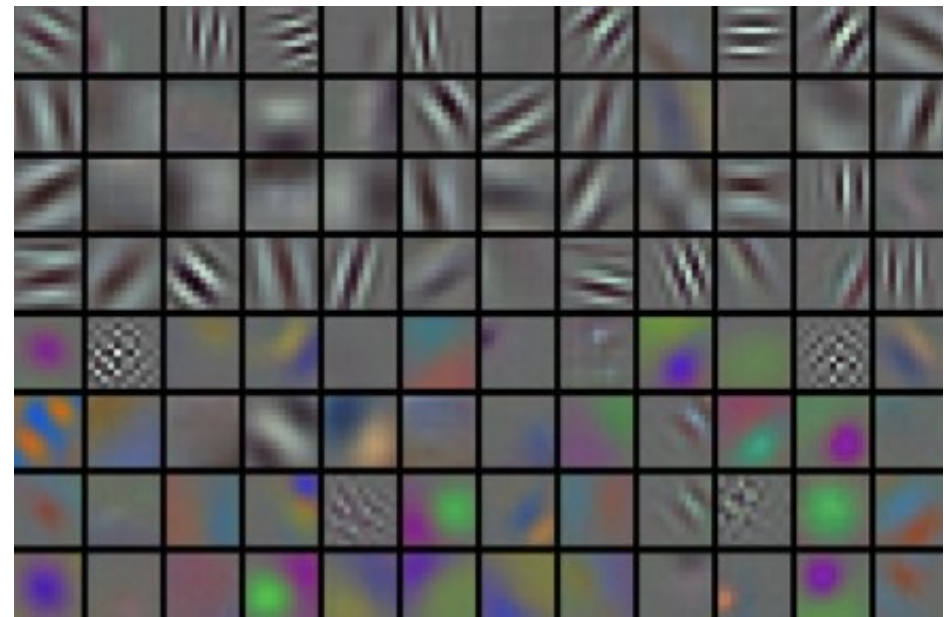
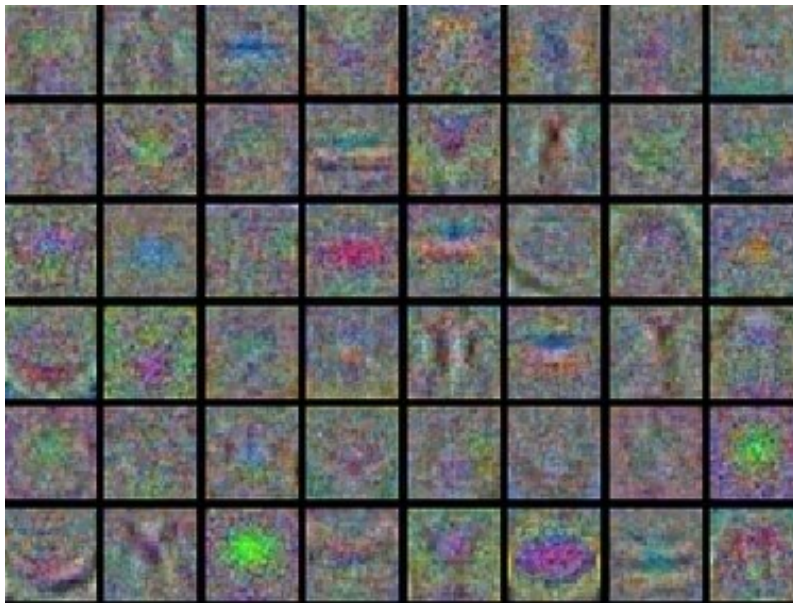


# Erste Faltungsschicht

## AlexNet



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



Welches Netzwerk ist besser? Links oder rechts?

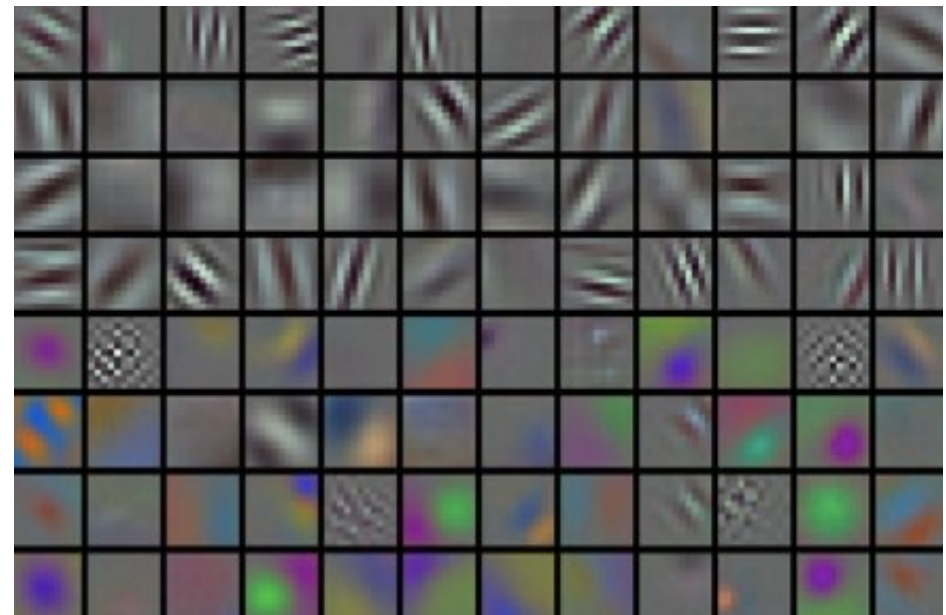
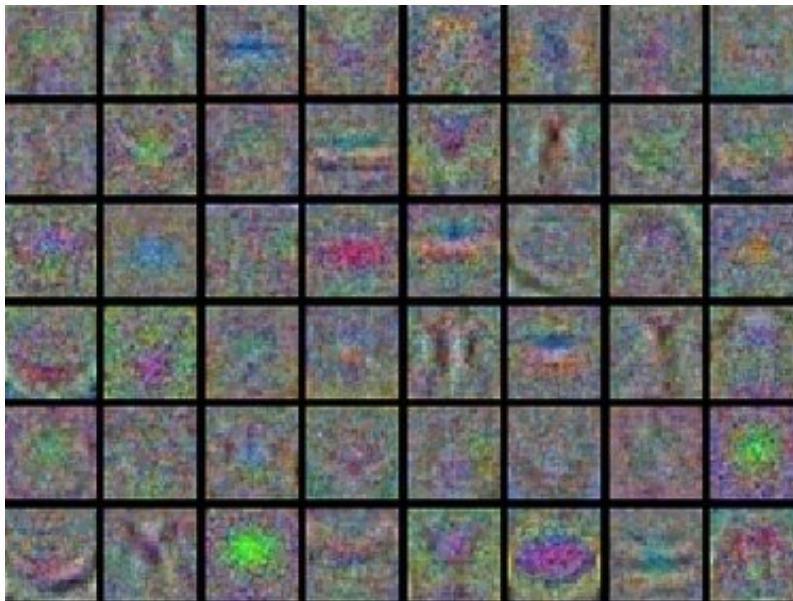


TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



# Erste Faltungsschicht

## AlexNet

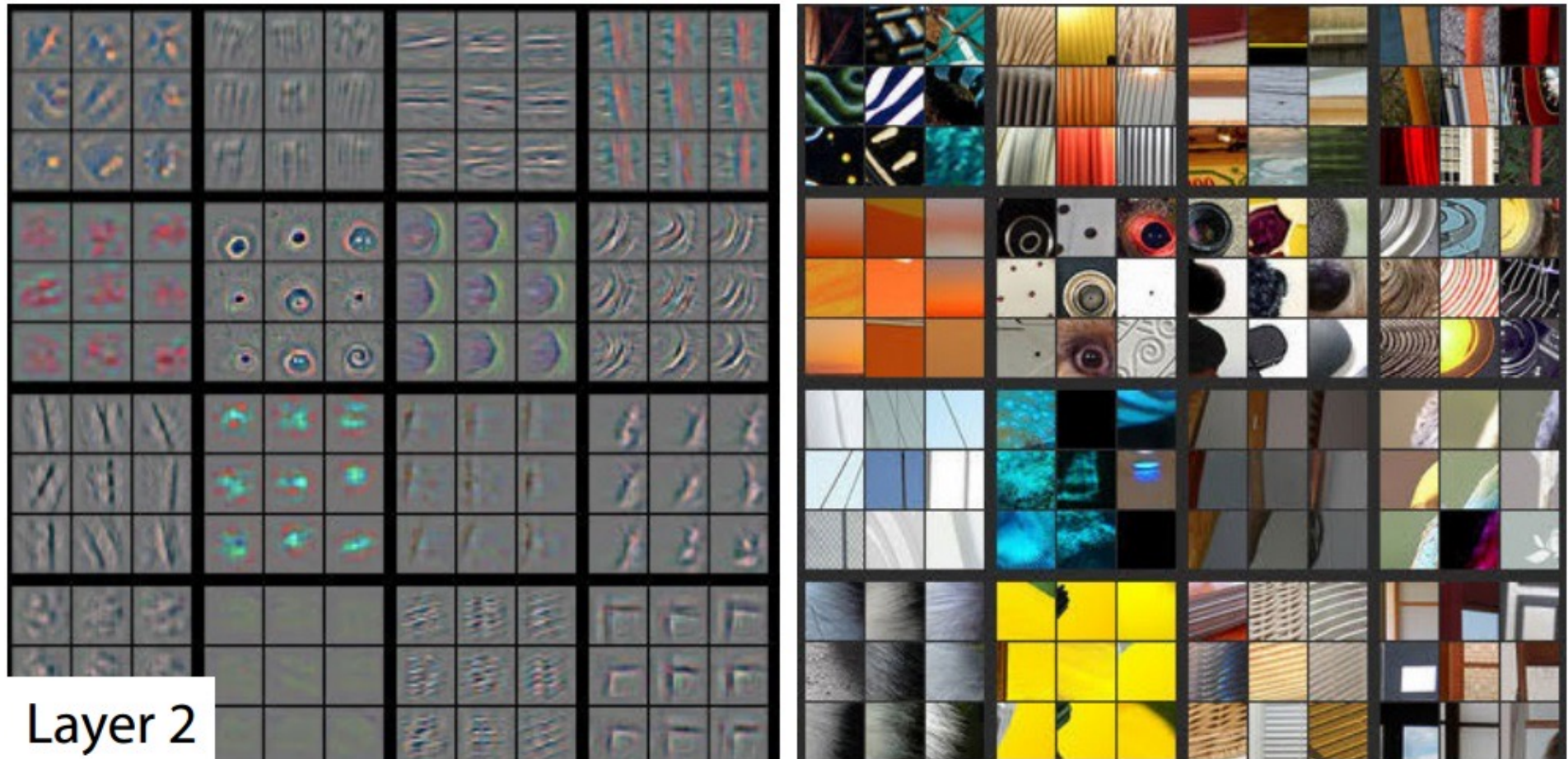


Links: vielleicht nicht konvergiert? Falsche Schrittweite? Nicht normalisiert?

Rechts: Schön, glatt, unterschiedliche Filter. Lernen scheint gut funktioniert zu haben



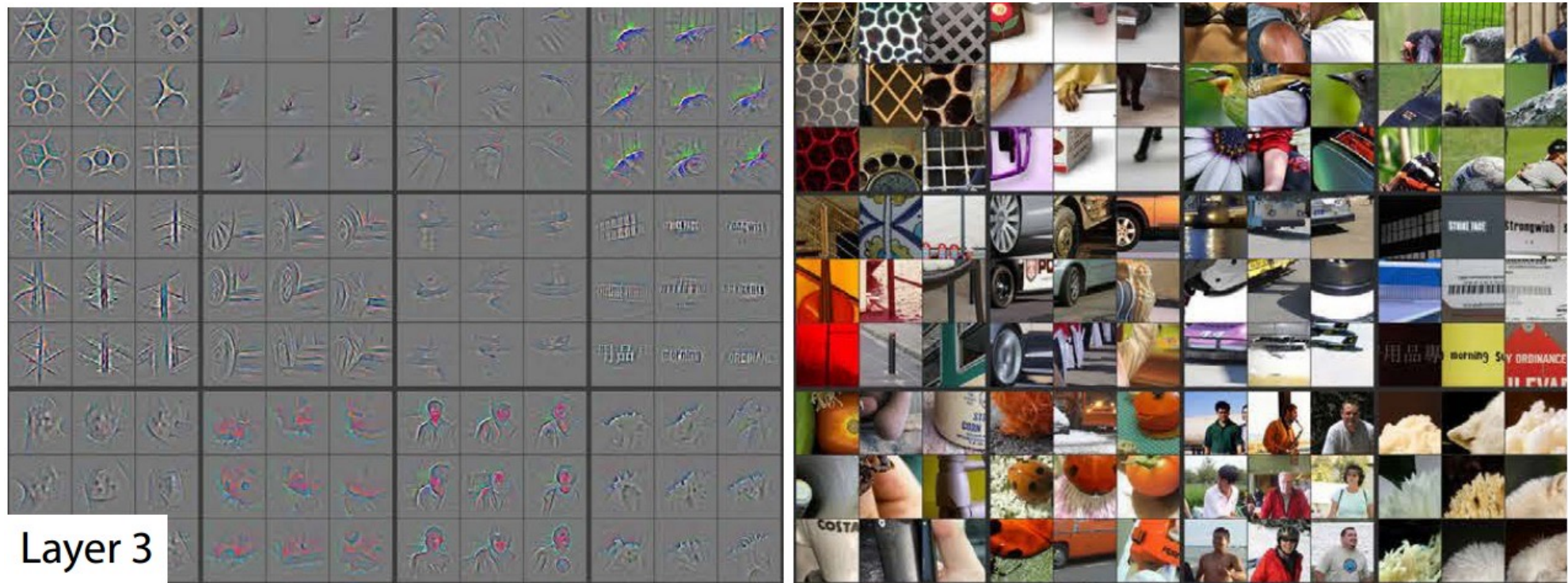
# Visualisierung von Faltungsnetzwerken



Visualizing and Understanding Convolutional Networks [[Zeiler and Fergus, ECCV 2014](#)]



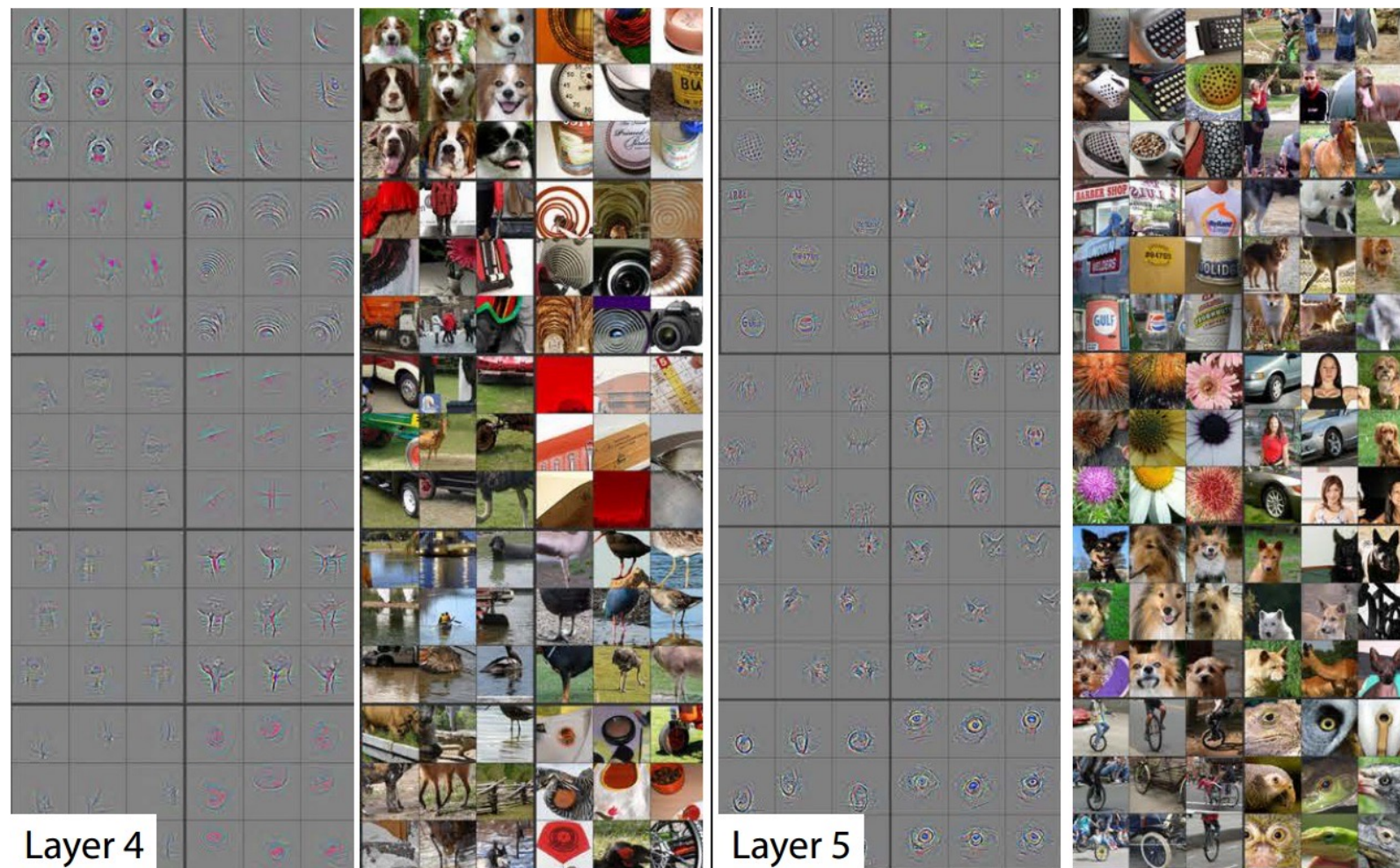
# Visualisierung von Faltungsnetzwerken



Visualizing and Understanding Convolutional Networks [[Zeiler and Fergus, ECCV 2014](#)]



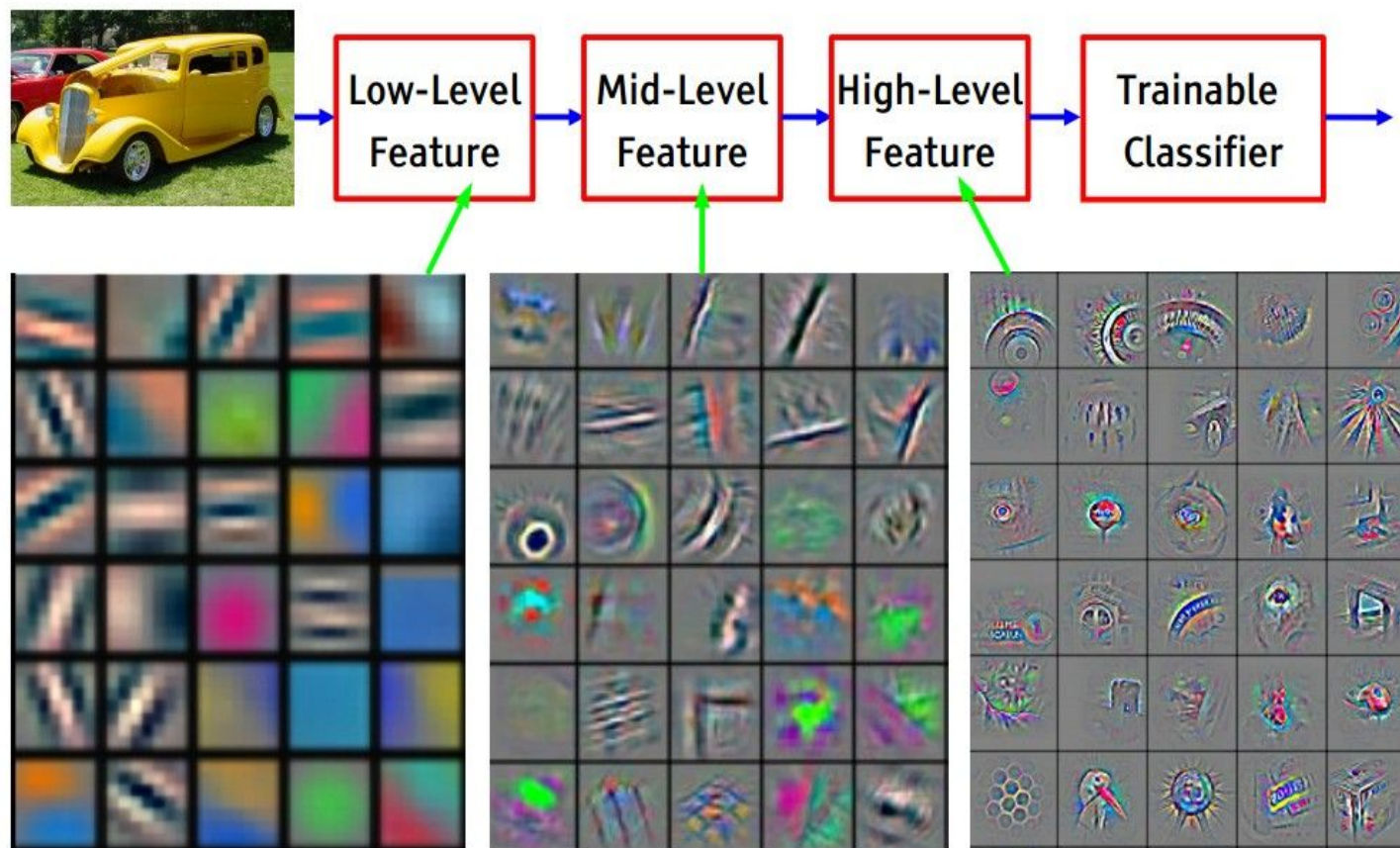
# Visualisierung von Faltungsnetzwerken



Visualizing and Understanding Convolutional Networks [[Zeiler and Fergus, ECCV 2014](#)]

# Visualisierung von Faltungsnetzwerken

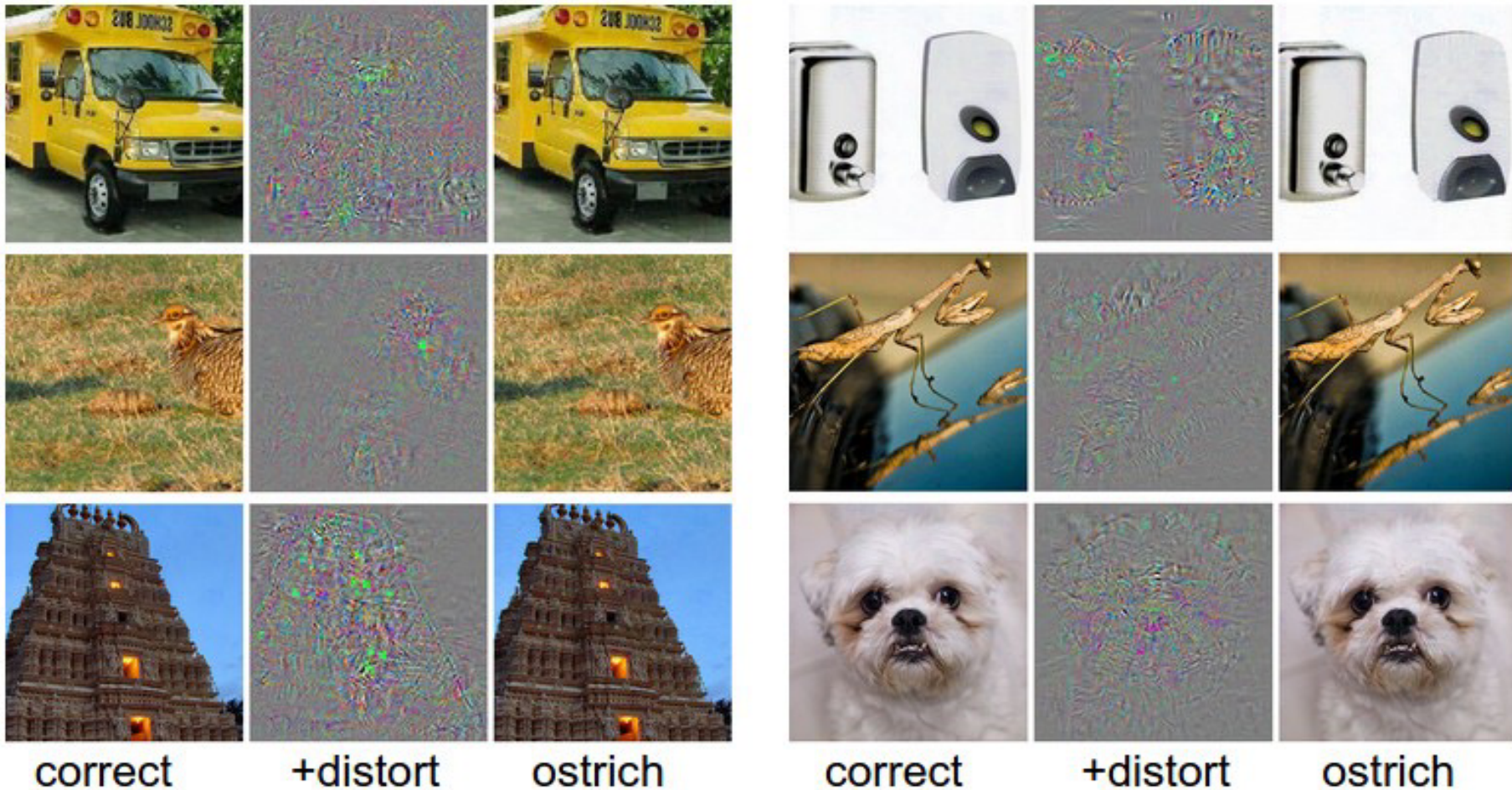
Die Schichten lernen hierarchische Repräsentationen der Eingabedaten



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



# Aber man kann CNNs auch „täuschen“



**Etwas Rauschen kann das Ergebnis komplett ändern**

## Aber man kann CNNs auch „täuschen“

“panda”  
57.7% confidence



+ .007 ×

“nematode”  
8.2% confidence



=

“gibbon”  
99.3 % confidence



Explaining and Harnessing Adversarial Examples [[Goodfellow ICLR 2015](#)]

## Etwas Rauschen kann das Ergebnis komplett ändern



# Was haben Sie bisher kennengelernt

---

- Das Trainieren von tiefen Neuronalen Netzwerken folgt dem Lego-Prinzip mittels Vorwärts- und Rückwärtspropagierung
- Meistens wird ein stochastischer Gradientenabstieg benutzt
- DropOut und Datenaugmentierung vermeiden Überanpassung
- Trainieren von tiefen Netzwerken kostet viel Zeit und Energie. Daher, wenn möglich, Transferlernen mittels pre-trained Modellen
- **Es gibt noch viel mehr zu sagen: Recurrent Networks, LSTM, Siam Networks, Variational Neural Networks, ...**