Lecture 7: Convolutional Neural Networks - Architekturen

BA-INF 153: Einführung in Deep Learning für Visual Computing

Prof. Dr. Reinhard Klein

Nils Wandel

Informatik II, Universität Bonn

12.06.2024

Wiederholung

Grundlagen CNNs

1. Äquivarianz und Invarianz
2. (kontinuierliche / diskrete) Konvolution und Kreuzkorrelation
3. Convolutional Layers (weight sharing, sparsity, local receiptive field)
4. Dilation, Padding, Stride, Pooling
5. Transposed Convolutional Layers
6. Batch Normalisierung
7. Convolutional Neural Networks

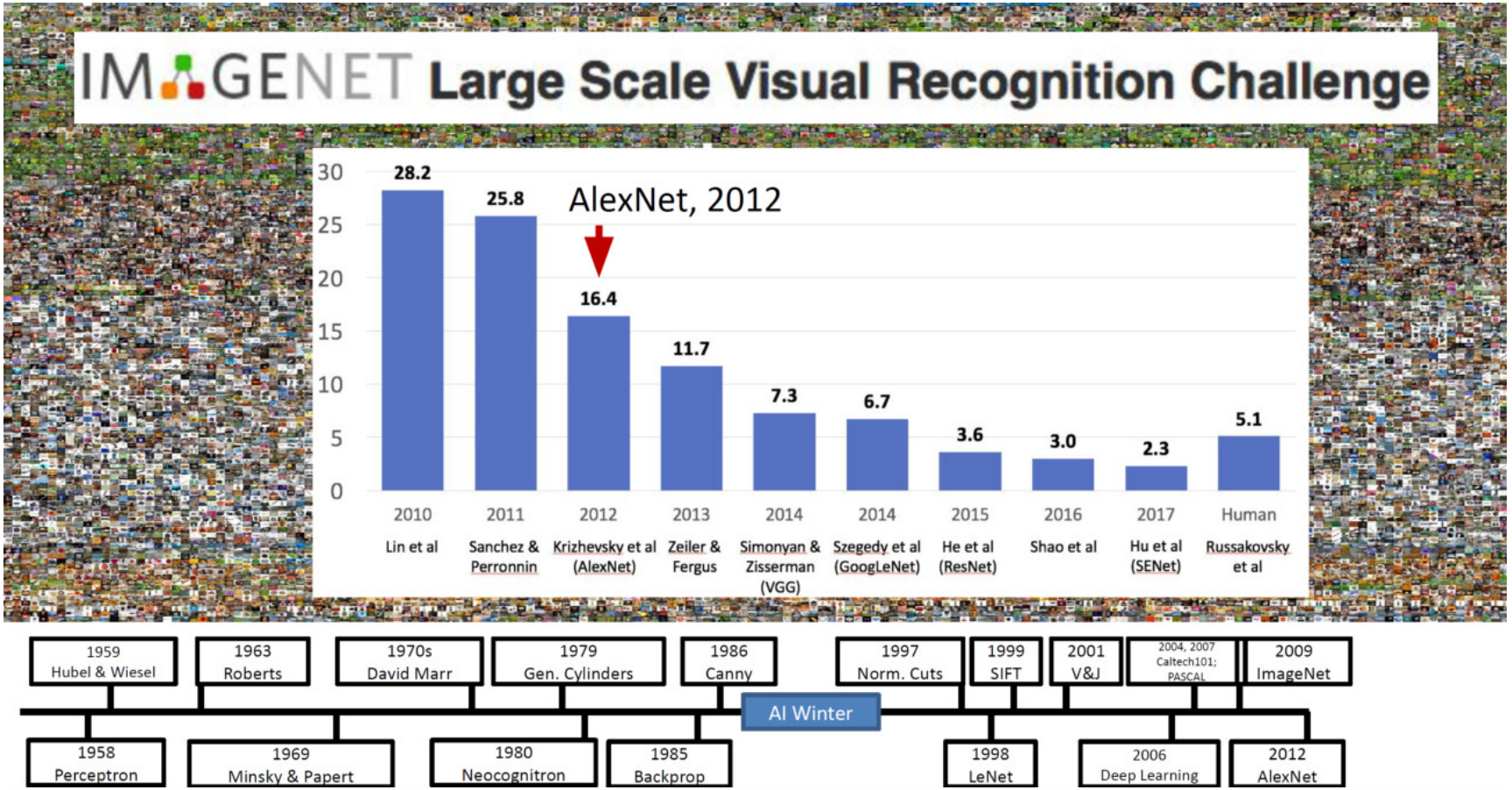
Today's Lecture:

1. CNN - Architectures:
    1. AlexNet
    2. VGG
    3. ResNet
2. CNN - Visualisierung und Analyse
    1. Activations
    2. Nearest Neighbors im Feature-Space
    3. Occlusion Experiments
    4. Saliency Maps
    5. Feature Visualisierung
    6. Texture Synthesis
    7. Style Transfer
    8. Adversarial Attacks
3. Transfer-Learning

Part 1

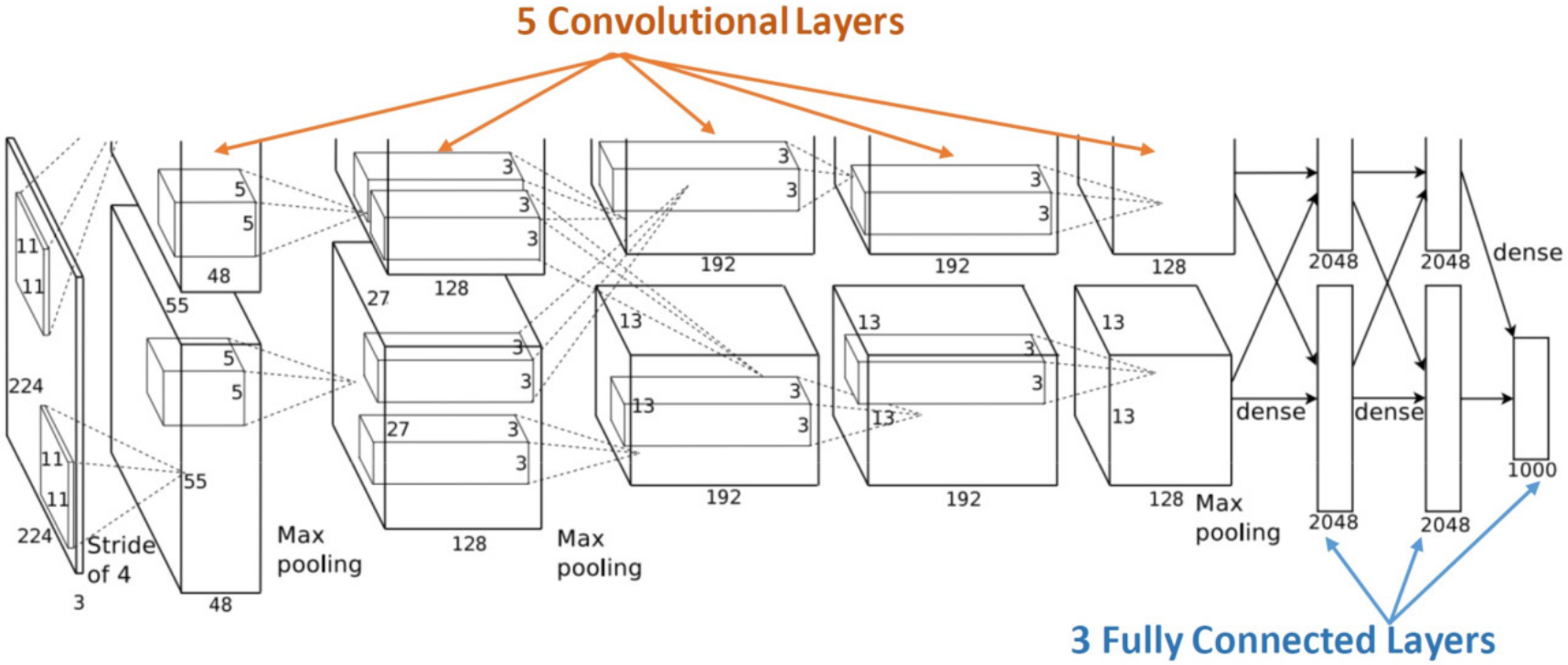
CNN Architekturen

ILSVRC: Top-5 error rates over time



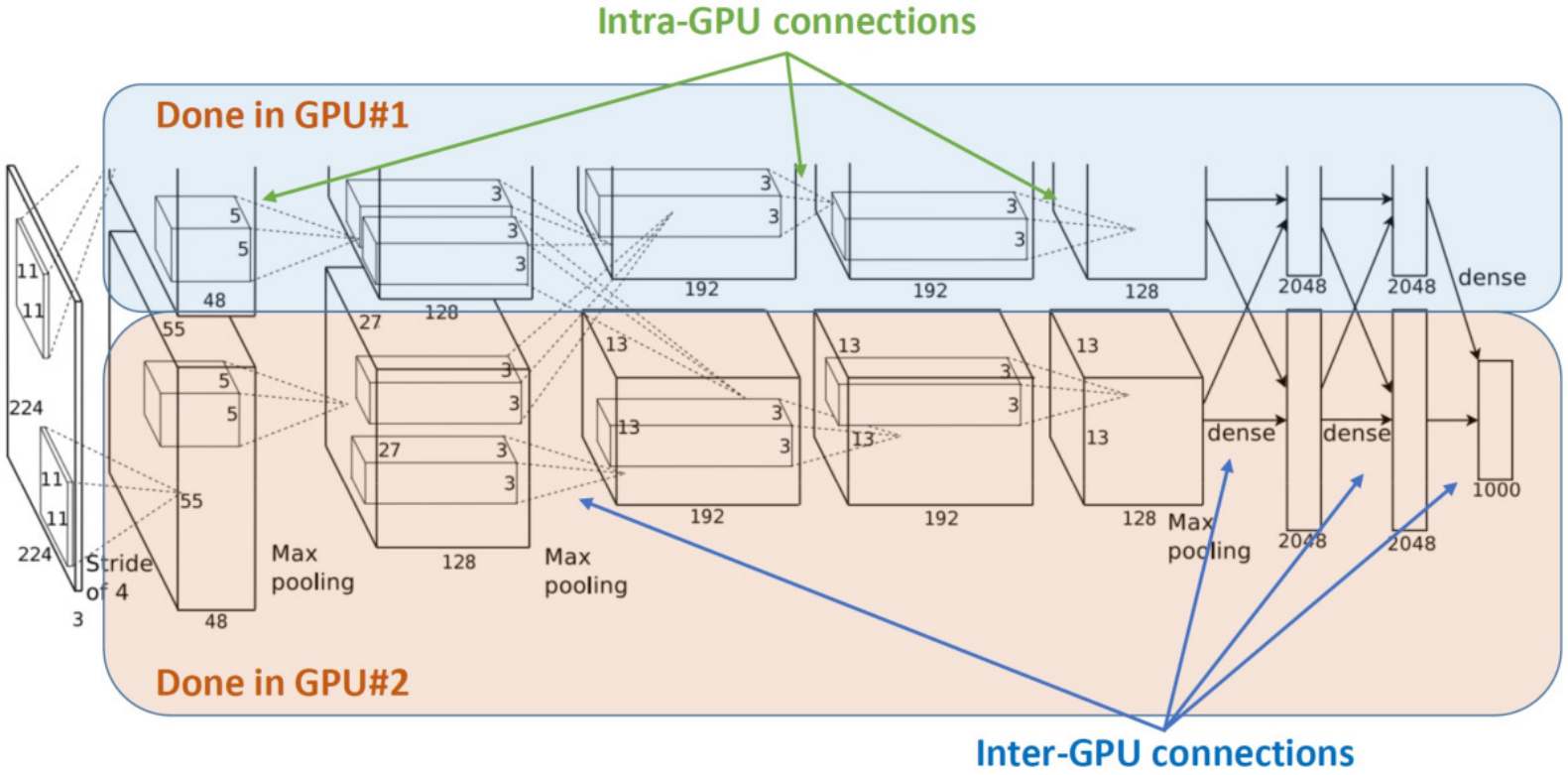
[Hier] gibt es eine interaktive Übersicht zum derzeitigen SOTA der ILSVRC.

AlexNet Architecture



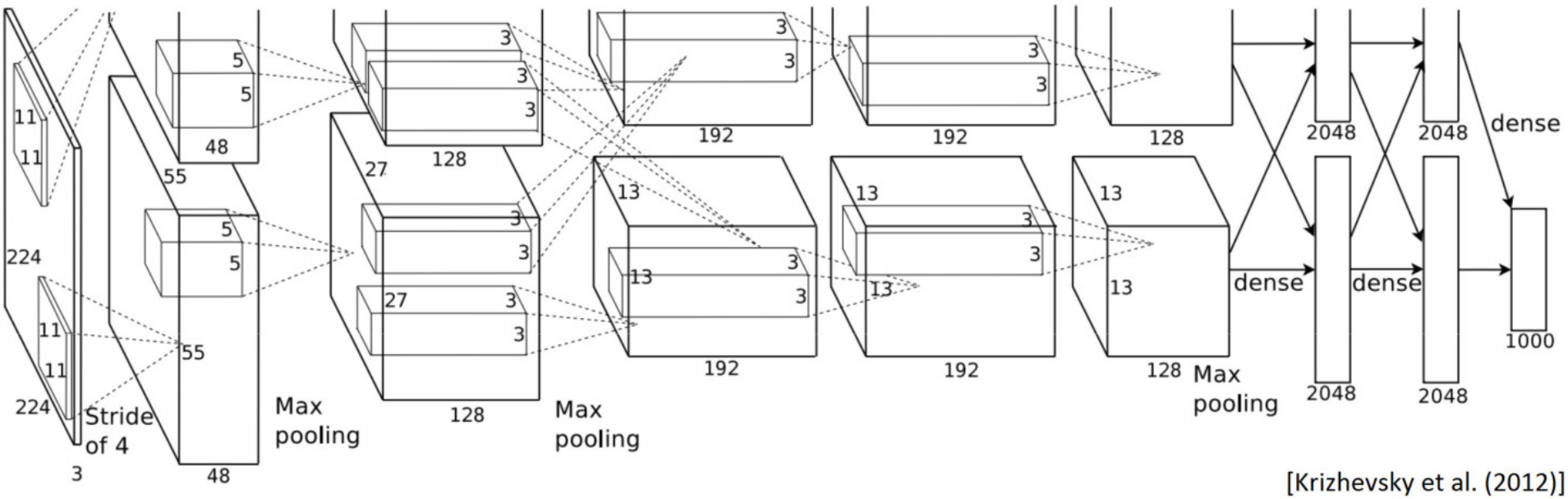
Output-Layer: Softmax für Wahrscheinlichkeitsfunktion über 1000 Klassen.

AlexNet Architecture



- Verteile Training auf 2 GPUs ⇒ Mehr Speicher und parallele Berechnungen
- Verbindungen zwischen verschiedenen GPUs sind nur im 3.,6.,7.,8. Layer vorhanden
- Verwendet ca 2 GB RAM auf jeder GPU und ca 5 GB System Memory während des Trainings

AlexNet Architecture



- Trainiert mit Stochastic Gradient Descent für 5-6 Tage auf 2 NVIDIA GTX 580 3GB GPUs.
- 650.000 Neuronen, 60 Mio Parameter, 640 Mio Verbindungen
- Letztes Feature-Layer: 4096 Neuronen
- Output-Layer: 1000 Neuronen für 1000 Klassen-Labels
- Eingabebilder werden auf eine Auflösung von 224×224 downsampled

Rectified Linear Units

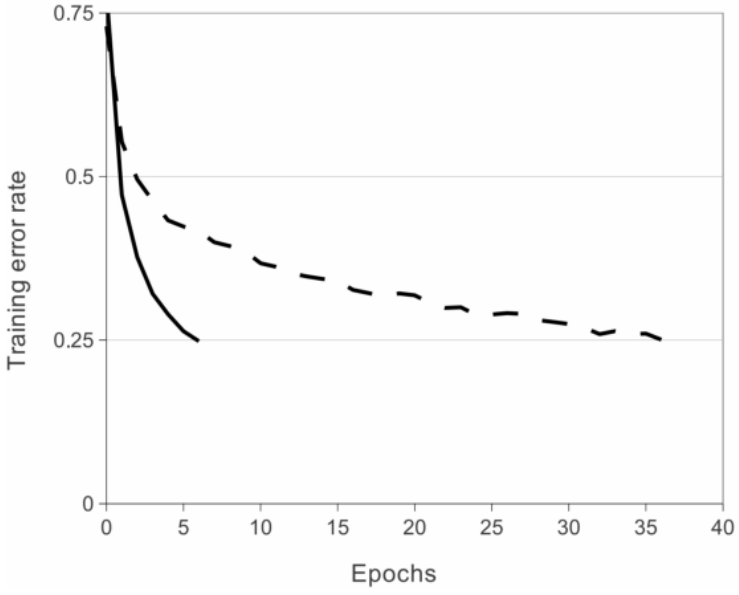


Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons **(dashed line)**.

Figure: Im Vergleich zu tanh führen ReLU-Aktivierungsfunktionen zu schnellerem Training, da diese für große Aktivierungen nicht sättigen.

Massnahmen gegen Overfitting

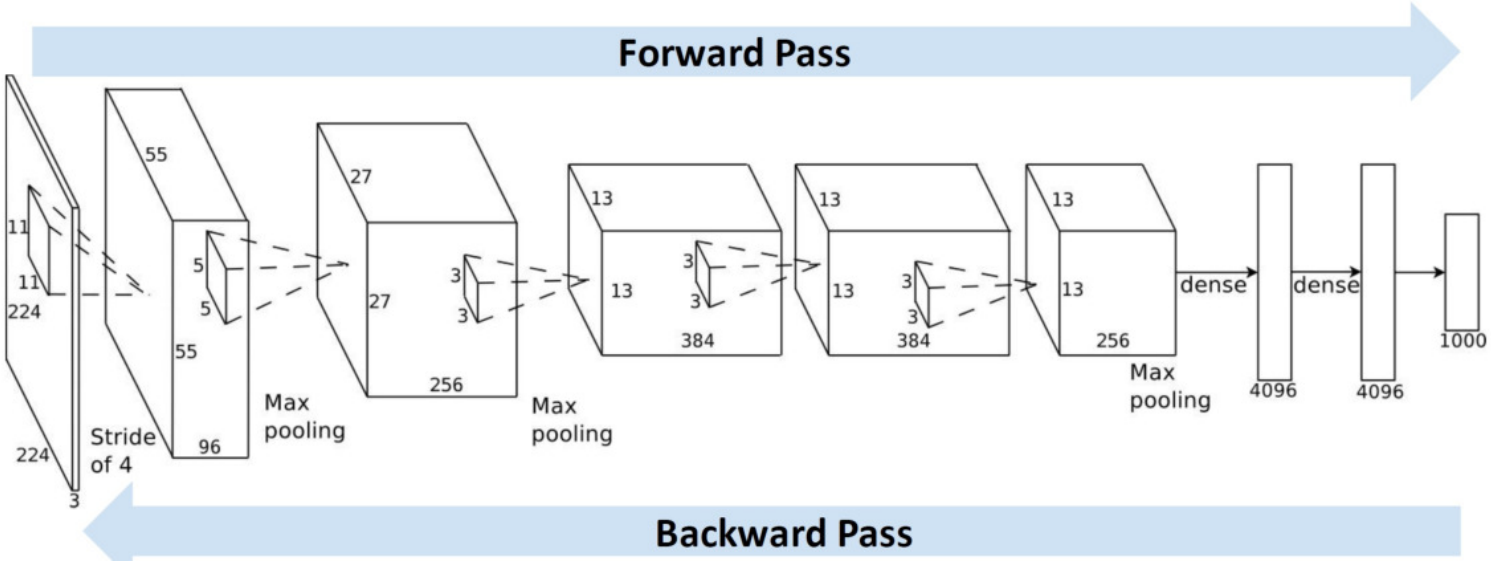
Data Augmentation

- Translationen und horizontale Spiegelungen indem zufällige 224×224 Bildausschnitte (und ihre Reflektionen) aus den 256×256 Trainings-Bildern gewählt werden.
- Dadurch wird die Grösse des Trainingssets um den Faktor 2048 vergrößert.

Dropout 0.5

- Dropout wird in den ersten beiden Fully-Connected Layern verwendet.
- Setze Outputs von einzelnen Units in den hidden Layern mit Wahrscheinlichkeit 0.5 auf 0.
- Durch Dropout werden ca doppelt so viele Trainingsschritte bis zu Konvergenz benötigt.

Training



- Modell wurde mit SGD (Batchsize=128, Initial learning rate=0.01, Momentum=0.9) trainiert.
- Regularisierung mit Weight decay = 0.0005.
- Normalverteilte Gewichtsinitialisierung ($\mu = 0, \sigma = 0.01$)
- Loss entspricht Cross-Entropy von Ground truth label mit Vorhersagen

Validation classification

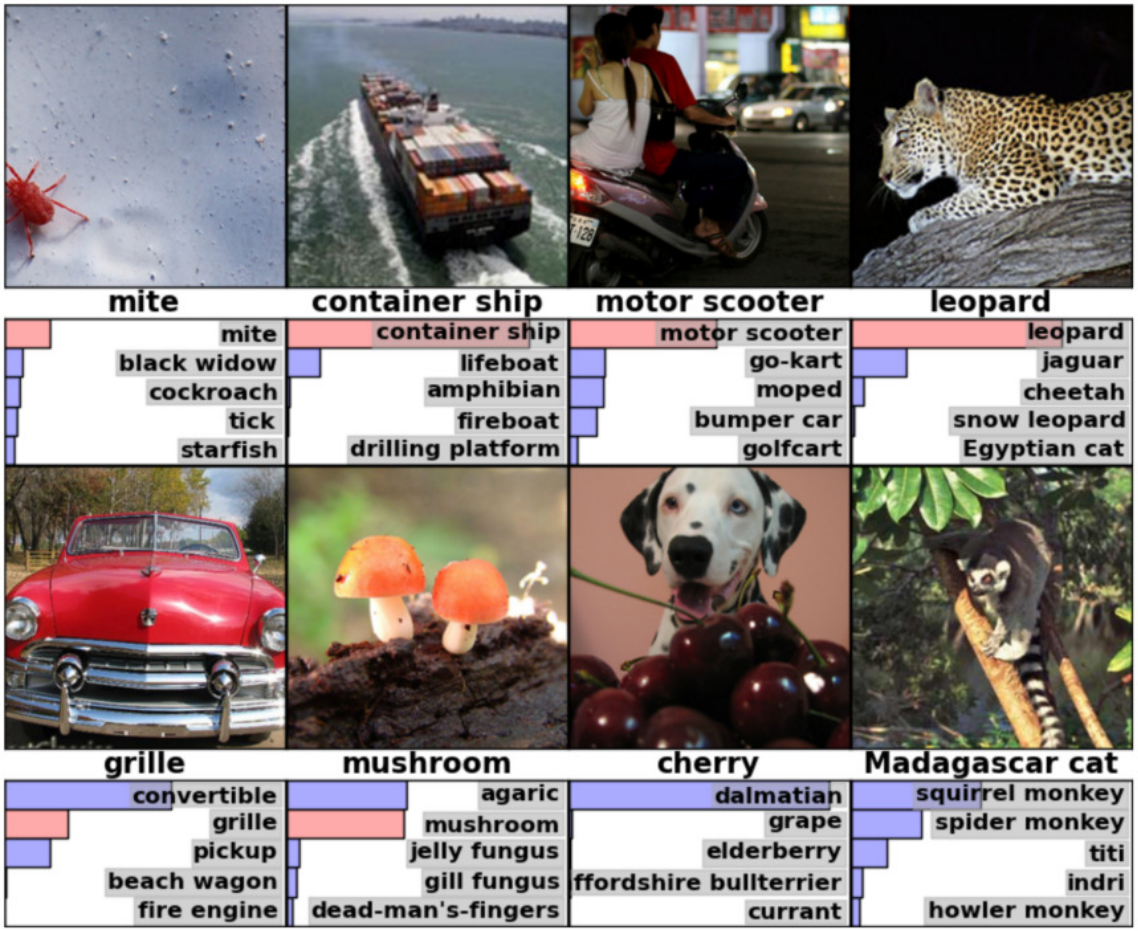
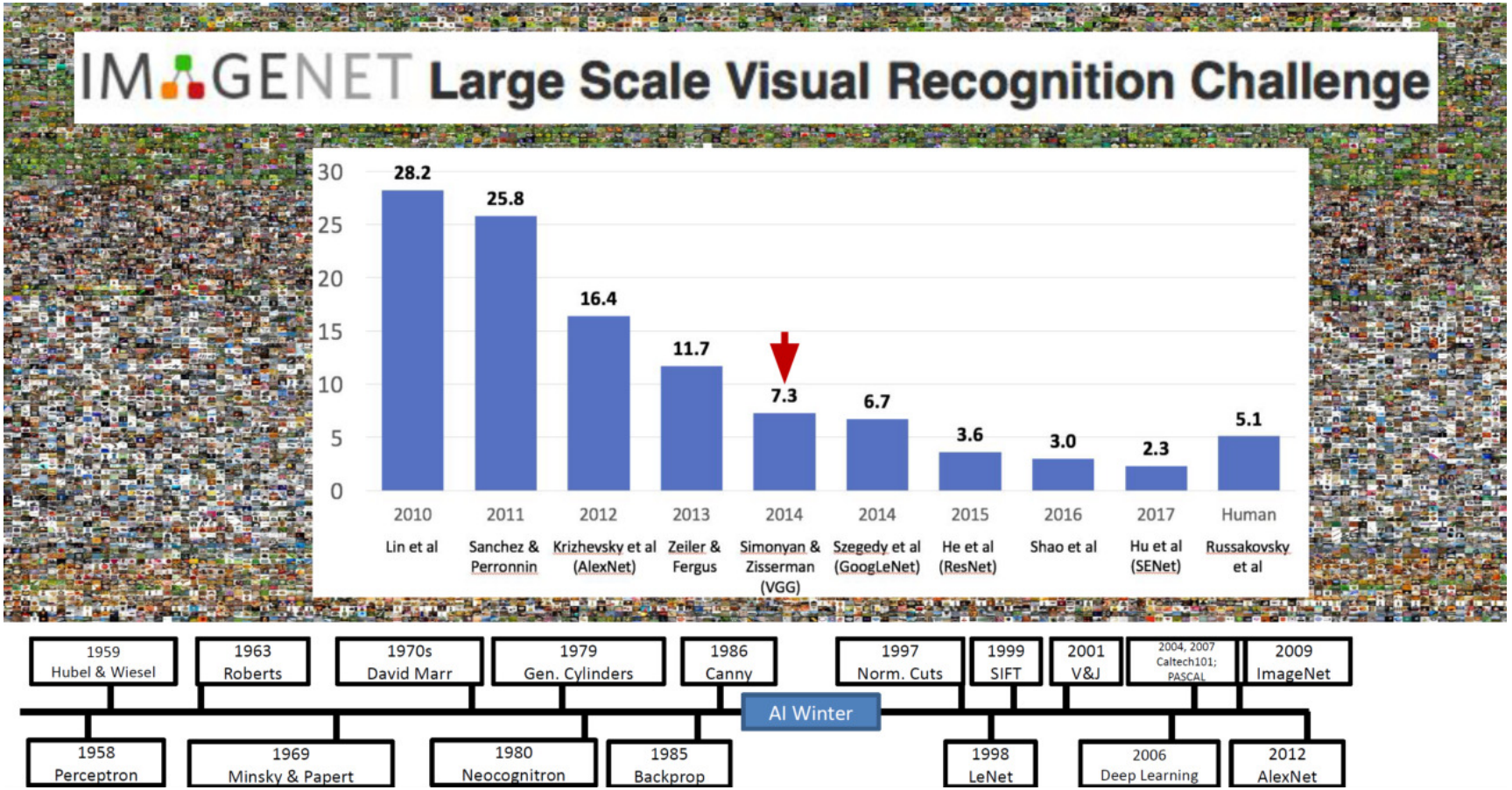


Figure: Ground truth labels und Vorhersagen des Alexnet für 8 Testbilder aus der ILSVR Challenge [Quelle]

ILSVRC: Top-5 error rates over time



[Hier] gibt es eine interaktive Übersicht zum derzeitigen SOTA der ILSVRC.

VGG-Net

Very Deep CNNs for Large-Scale Image Recognition

Figure: Slide von [hier]

Wichtiger Beitrag: Netzwerktiefe ist bedeutsam! 19 Layers.

VGGNet Architecture

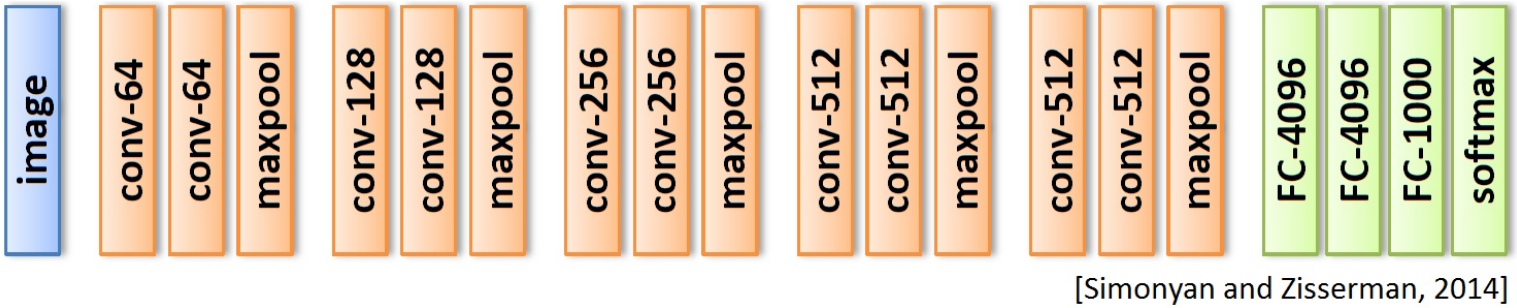


Figure: slide from [here]

Wichtige Design Entscheidungen:

- Convolutional Layers:
  - 3×3 conv. filters - kleinste Filtergröße, um einen Eindruck von links/rechts, oben/unten, mitte zu bekommen.
  - Stride 1 mit 1 Pixel Padding - keine Verkleinerung der hidden Repräsentation
- Max-pooling
  - 2×2 Pooling-Fenster mit Stride 2
  - 5 max-pool Layers mit jeweils Faktor x2 Reduktion
- Zum Schluss: 3 fully-connected (FC) Layers

Warum 3x3 convolutional Layers?

- Aufeinanderfolgende convolutional Layers haben ein größeres Receiptive Field
  - Zwei 3x3 convolutional Layers ⇒ 5x5 receptive field
  - Drei 3x3 convolutional Layers ⇒ 7x7 receptive field

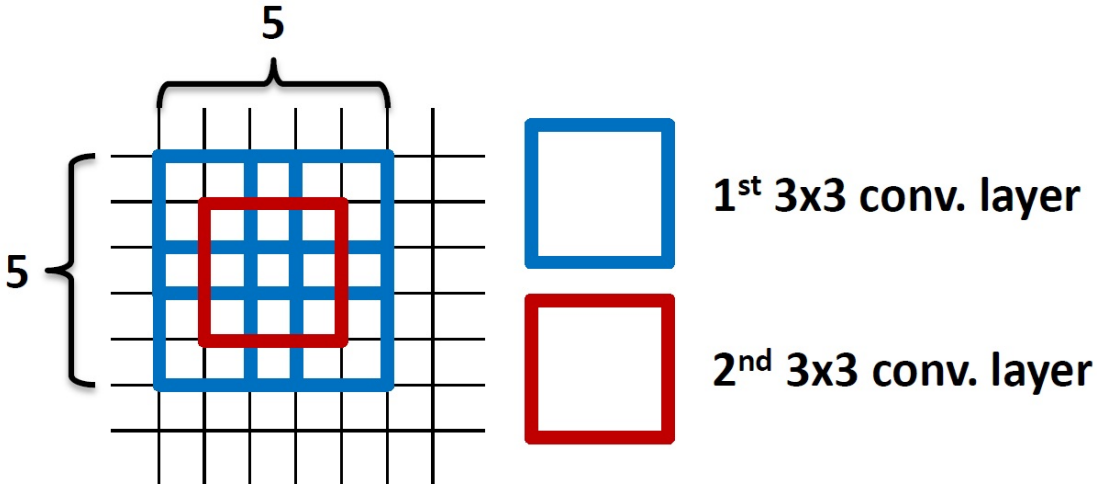


Figure: Slide von [here]

- Umso tiefer das Netzwerk → Mehr Nicht-Linearität / komplexere Features
- Mehr Parameter: ∼ 140 M vs ∼ 62 M für AlexNet

VGG-Net Konfigurationen

Um das Skalierungsverhalten von NNs zu untersuchen, werden häufig mehrere verschieden große Modelle untersucht (siehe Spalten A bis E).

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

[Simonyan and Zisserman, 2014]

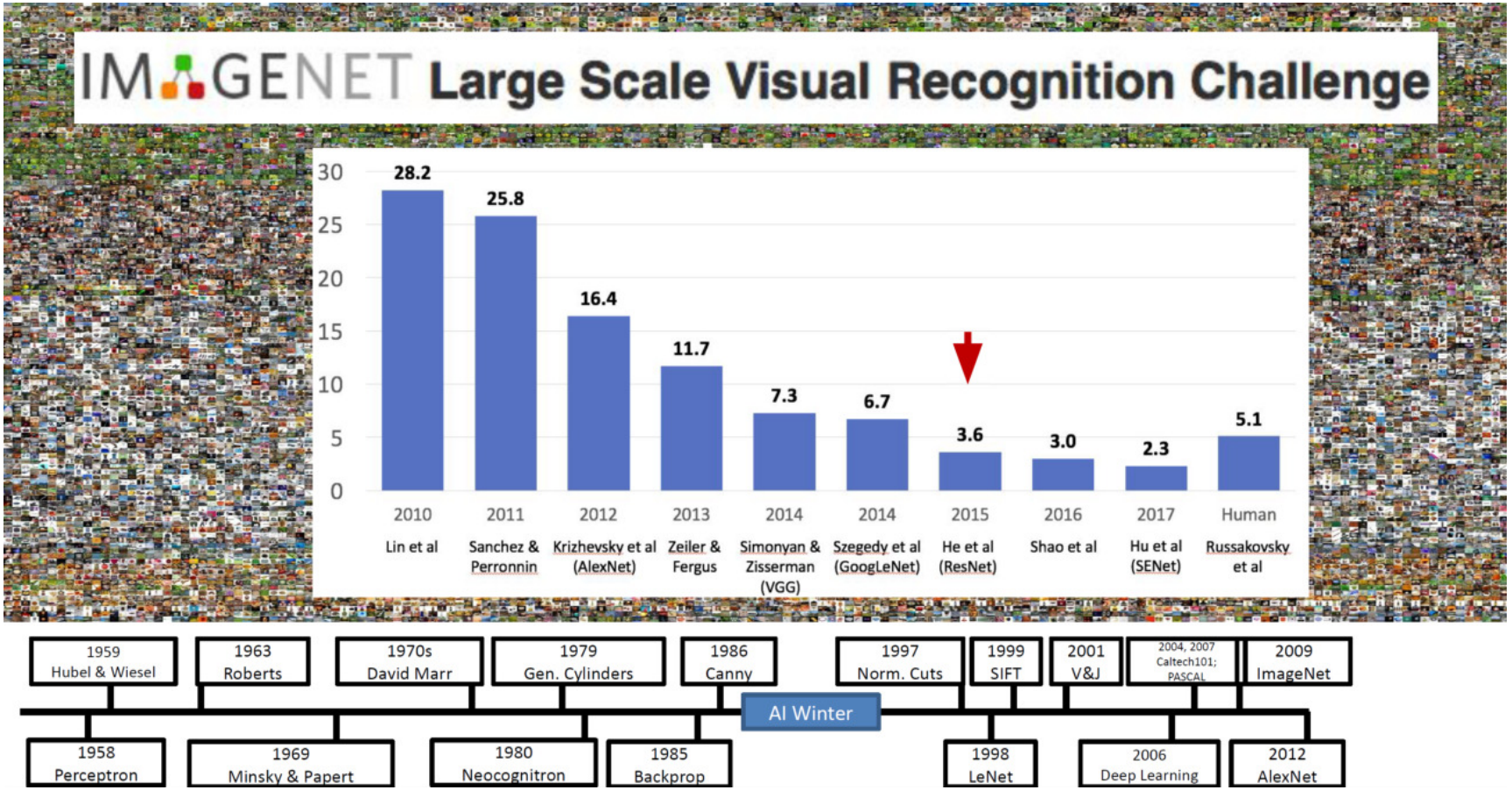
Number of parameters (in millions).

Network	A,A-LRN	B	C	D	E
parameters	133	133	134	138	144

Layers - Anzahl Parameter (VGG Konfiguration D)

Layer	Memory	Params
image (224×224 RGB)	224*224*3=150K	0
conv3-64	224*224*64=3.2M	(3*3*3)*64 = 1K
conv3-64	224*224*64=3.2M	(3*3*64)*64 = 36K
pool2	112*112*64=800K	0
conv3-128	112*112*128=1.6M	(3*3*64)*128 = 73K
conv3-128	112*112*128=1.6M	(3*3*128)*128 = 147K
pool2	56*56*128=400K	0
conv3-256	56*56*256=800K	(3*3*128)*256 = 294K
conv3-256	56*56*256=800K	(3*3*256)*256 = 589K
conv3-256	56*56*256=800K	(3*3*256)*256 = 589K
pool2	28*28*256=200K	0
conv3-512	28*28*512=400K	(3*3*256)*512 = 1.1M
conv3-512	28*28*512=400K	(3*3*512)*512 = 2.3M
conv3-512	28*28*512=400K	(3*3*512)*512 = 2.3M
pool2	14*14*512=100K	0
conv3-512	14*14*512=100K	(3*3*512)*512 = 2.3M
conv3-512	14*14*512=100K	(3*3*512)*512 = 2.3M
conv3-512	14*14*512=100K	(3*3*512)*512 = 2.3M
pool2	7*7*512=25K	0
FC-4096	4096	7*7*512*4096 = 102M
FC-4096	4096	4096*4096 = 16.7M
FC-1000	1000	4096*1000 = 4M
Total	24M	138M

ILSVRC: Top-5 error rates over time



[Hier] gibt es eine interaktive Übersicht zum derzeitigen SOTA der ILSVRC.

ResNet: Revolution of Depth

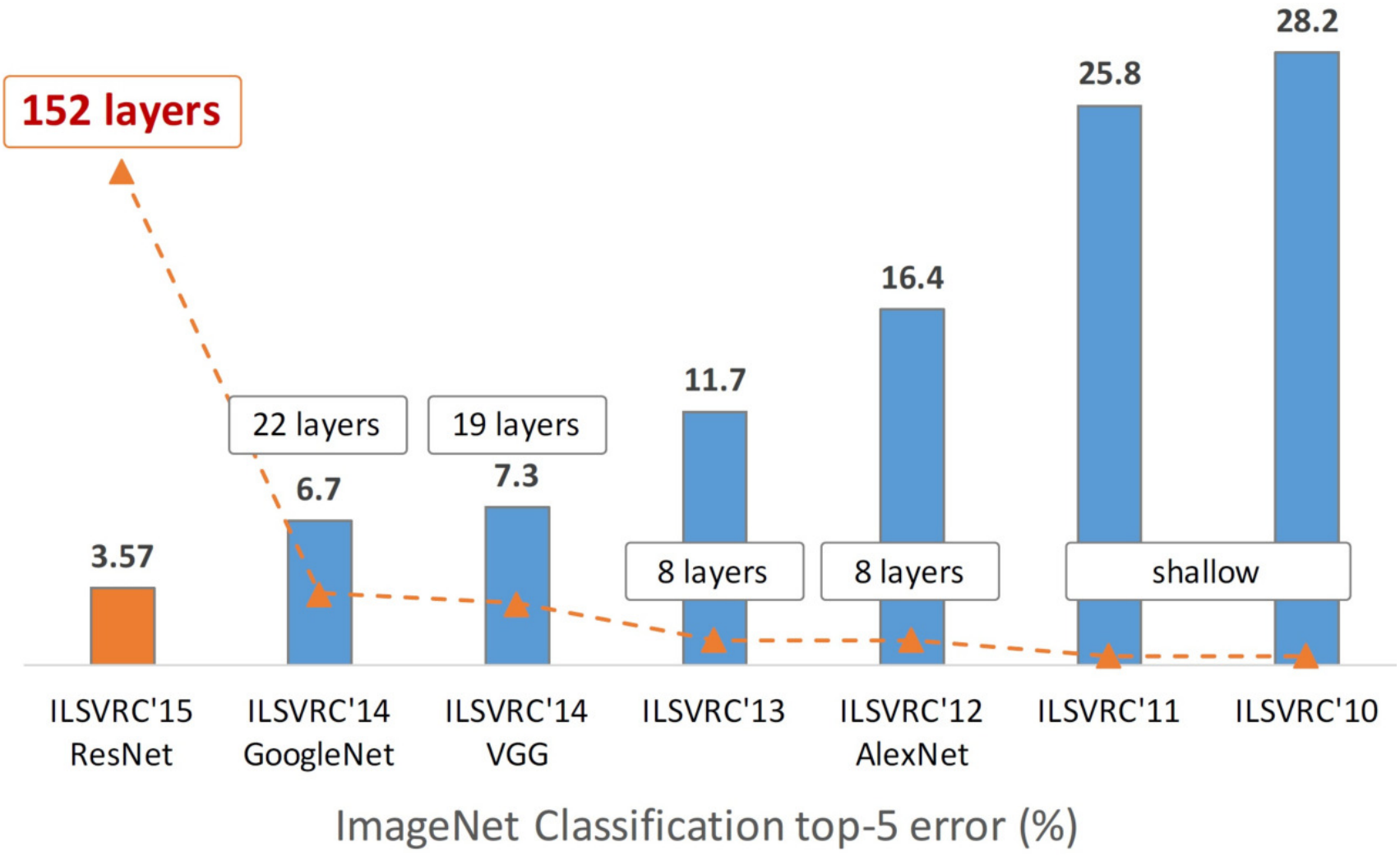


Figure: Slide von [hier]

ResNet: Revolution of Depth

Figure: Slide von [hier]

2-3 Wochen Trainingszeit auf 8 GPUs

ResNet: Revolution of Depth

ResNets @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: "*Ultra-deep*" 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

Figure: Slide von [hier]

Einfach mehr Layer verwenden?

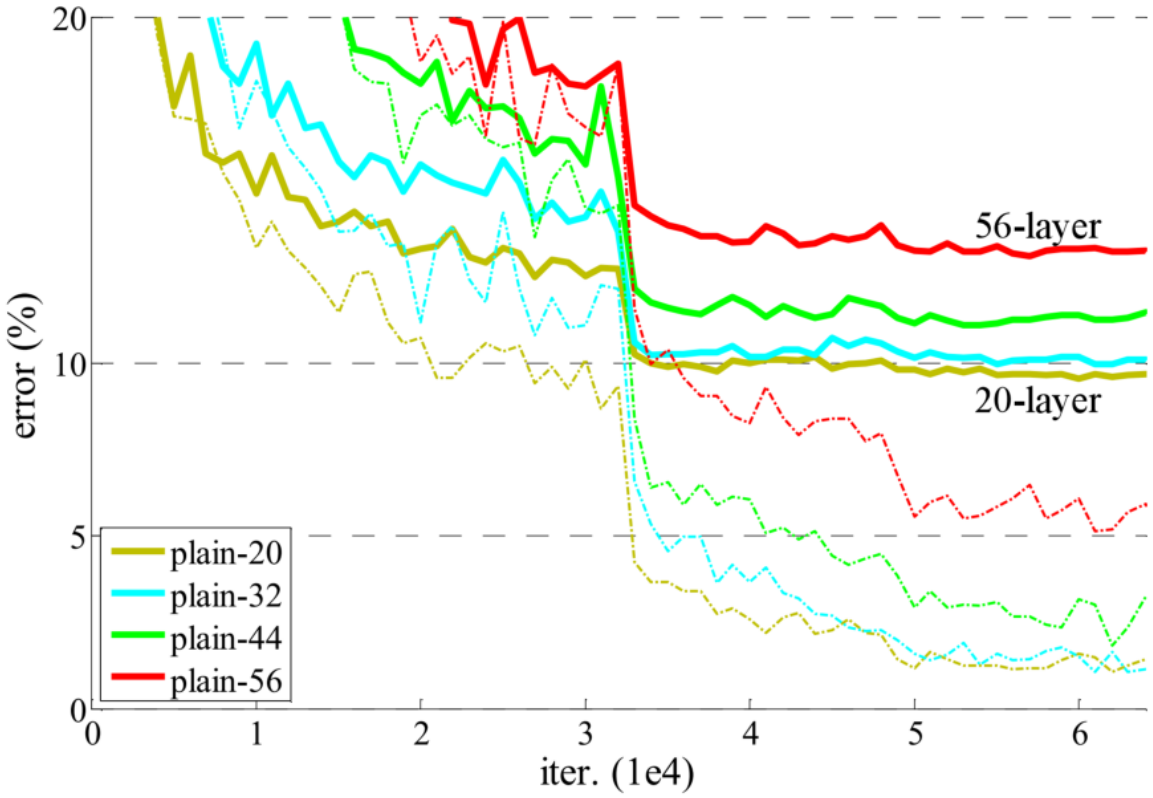
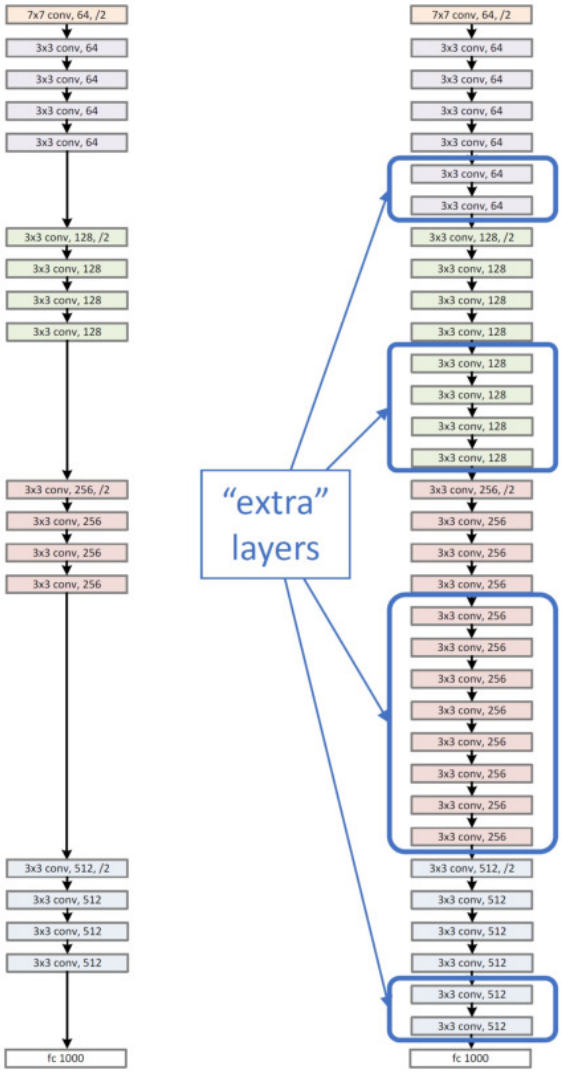


Figure: Abbildung aus [ResNet-Paper]. Solid lines: Test-Fehler; Dashed lines: Training-Fehler

... wie man sieht, führen mehr Layers in "naiven" feed-forward Netzen zu höheren Test und Trainings-Fehlern.
Warum ist das merkwürdig?

Einfach mehr Layer verwenden?

a shallower model (18 layers)



a deeper counterpart (34 layers)

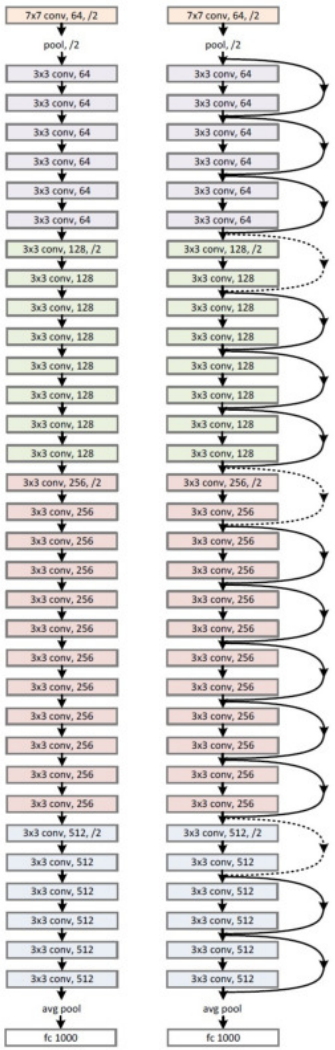
- A deeper model should not have **higher training error**
- A solution *by construction*:
  - original layers: copied from a learned shallower model
  - extra layers: set as identity
  - at least the same training error
- Optimization difficulties: solvers cannot find the solution when going deeper...

Figure: Slide von [hier]

Netzwerk Design

- Keep it simple
- Our basic design (VGG-style)
  - all 3x3 conv (almost)
  - spatial size /2 => # filters x2
  - Simple design; just deep!
- Other remarks:
  - no max pooling (almost)
  - no hidden fc
  - no dropout

plain net



ResNet

Figure: Slide von [hier]

Deep Residual Learning

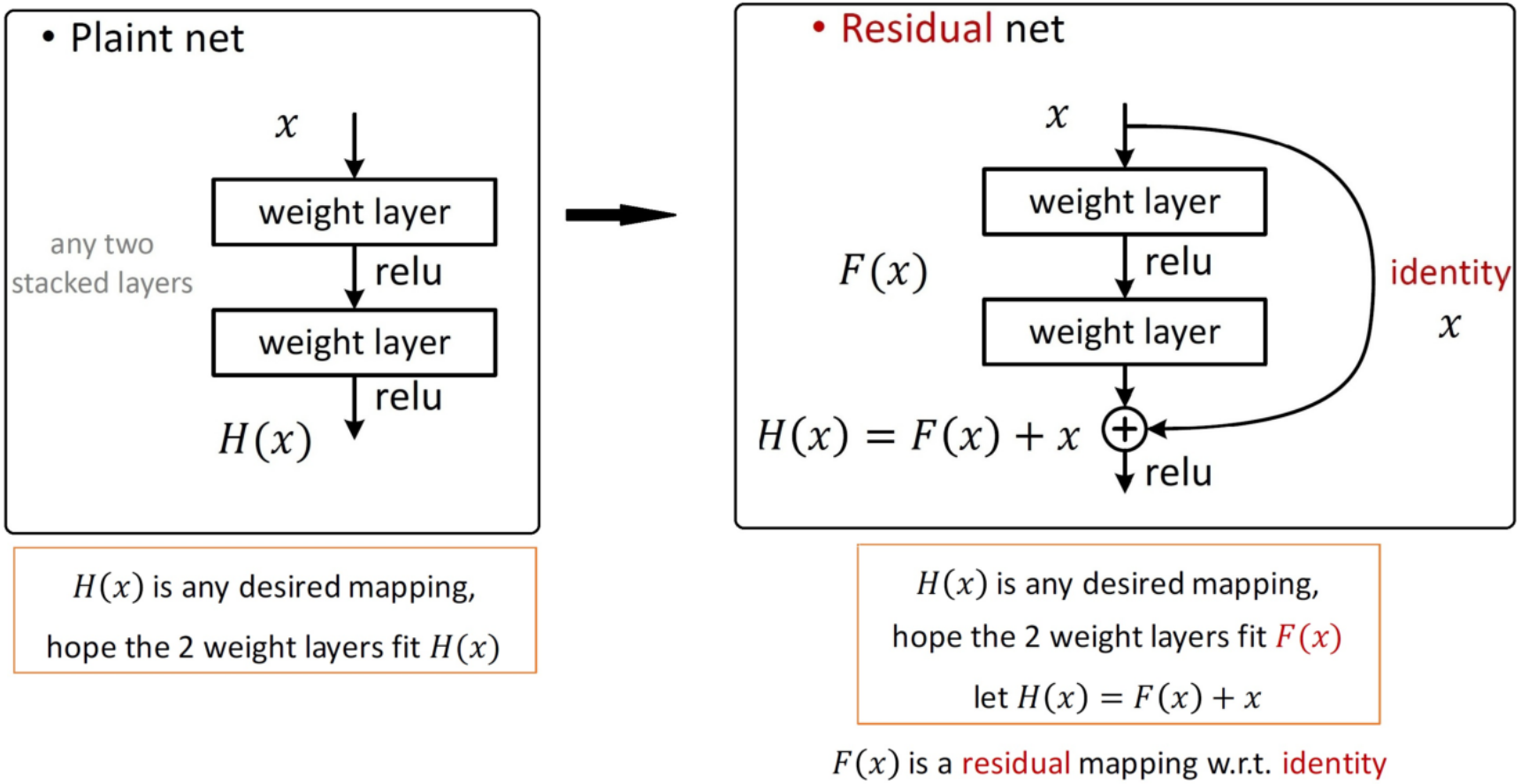
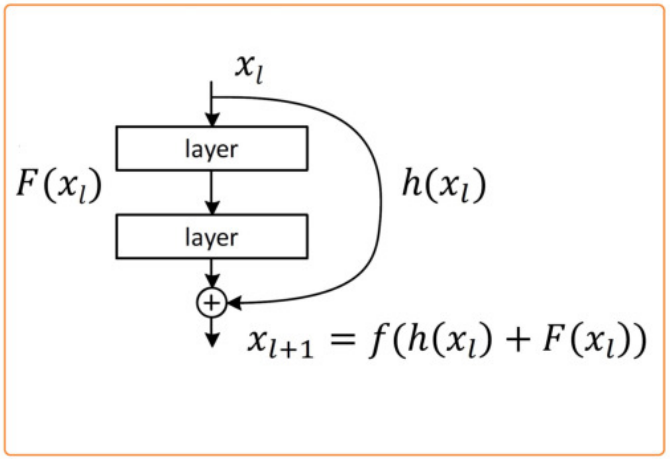


Figure: Slide von [hier]

On identity mappings for optimization



Very smooth forward propagation

$$x_{l+1} = x_l + F(x_l)$$

$$x_{l+2} = x_{l+1} + F(x_{l+1})$$

$$x_{l+2} = x_l + F(x_l) + F(x_{l+1})$$

$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i)$$

Very smooth forward propagation

$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i)$$

- Any x_l is directly forward-prop to any x_L, plus residual.
- Any x_L is an additive outcome.
  - in contrast to multiplicative: $x_L = \prod_{i=l}^{L-1} W_i x_l$

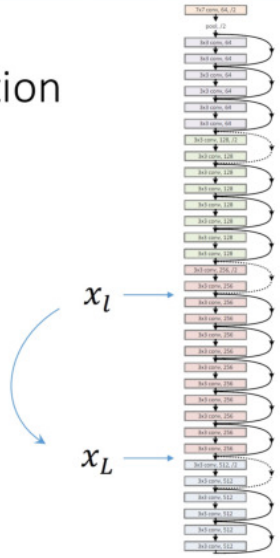


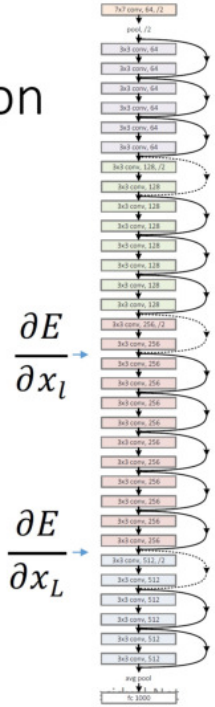
Figure: Slide von [hier]

On identity mappings for optimization

Very smooth backward propagation

$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i)$$

$$\frac{\partial E}{\partial x_l} = \frac{\partial E}{\partial x_L}\frac{\partial x_L}{\partial x_l} = \frac{\partial E}{\partial x_L}\left(1 + \frac{\partial}{\partial x_l}\sum_{i=1}^{L-1} F(x_i)\right)$$



Very smooth backward propagation

$$\frac{\partial E}{\partial x_l} = \frac{\partial E}{\partial x_L}\left(1 + \frac{\partial}{\partial x_l}\sum_{i=1}^{L-1} F(x_i)\right)$$

- Any $\frac{\partial E}{\partial x_L}$ is directly back-prop to any $\frac{\partial E}{\partial x_l}$, plus residual.
- Any $\frac{\partial E}{\partial x_l}$ is additive; unlikely to vanish
  - in contrast to multiplicative: $\frac{\partial E}{\partial x_l} = \prod_{i=l}^{L-1} W_i \frac{\partial E}{\partial x_L}$

Figure: Slide von [hier]

Einfach mehr Layer verwenden!

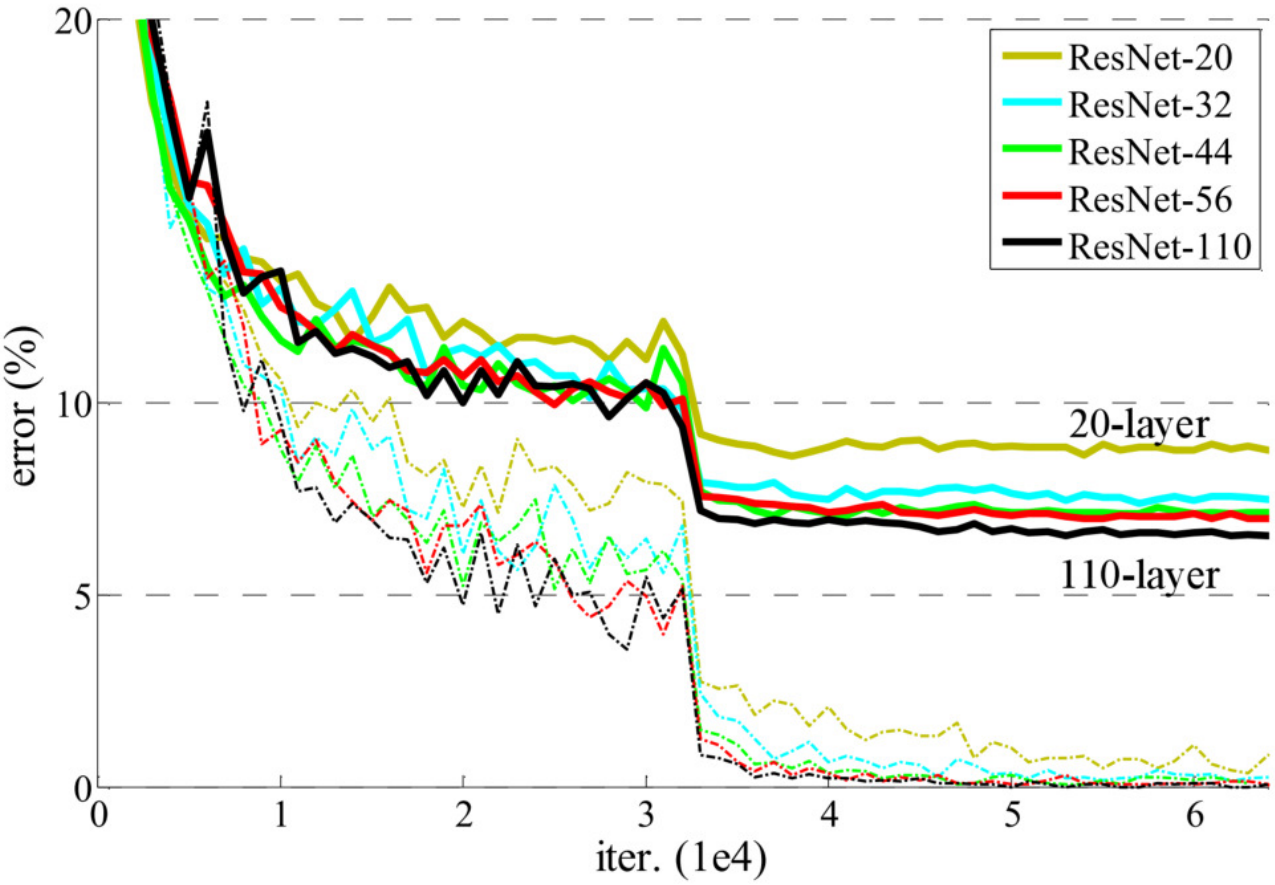


Figure: Abbildung aus [ResNet-Paper]. Solid lines: Test-Fehler; Dashed lines: Training-Fehler

... mit residual Connections können wesentlich bessere Trainings- und Testfehler auch bei sehr tiefen Netzen erzielt werden.

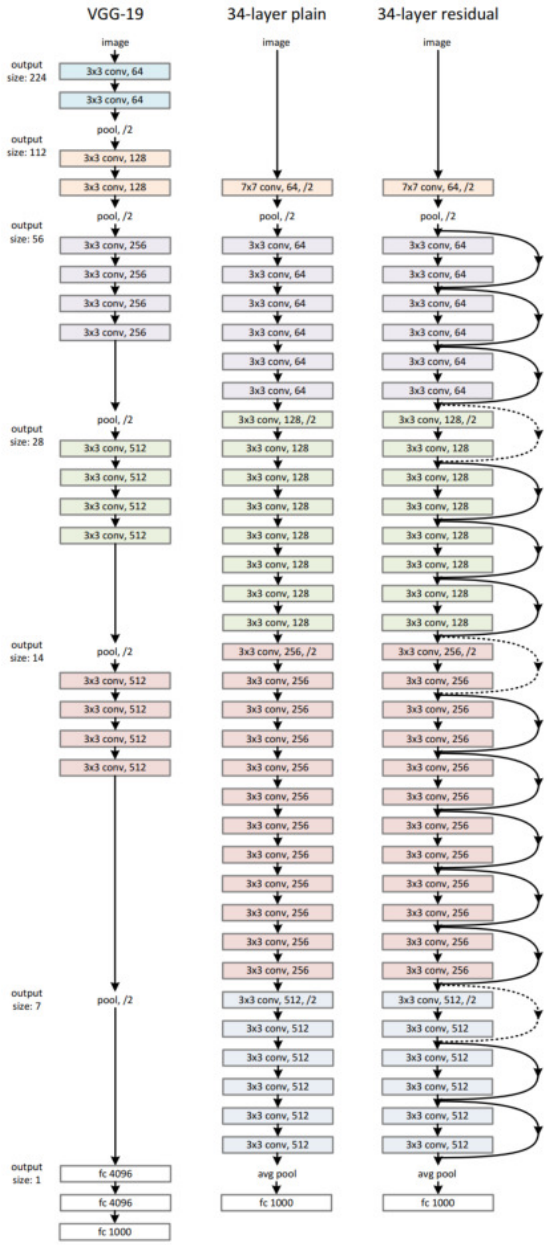
ResNet Konfigurationen

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure: ResNet-Bausteine sind in Klammern geschrieben × Stacking-Anzahl. Downsampling findet nach conv3_1, conv4_1, und conv5_1 mit Stride=2 statt. Slide von [hier]

.

ResNet vs VGG



Comparison				
Network	Year	top5 accuracy	Parameters	FLOP
AlexNet	2012	84.70%	62M	1.5B
VGGNet	2014	92.30%	138M	19.6B
ResNet-152	2015	95.51%	60.3M	11B

Figure: Obwohl das ResNet-152 über 8x mehr Layers als das VGG-19 besitzt, ist das ResNet effizienter bzgl. Speicher und Rechenaufwand (es benötigt weniger Parameter und weniger FLOPs für die Inferenz).

Woran liegt das?

Weitere Architekturen

Die Entwicklung von CNN-Architekturen ist ein offenes Forschungsgebiet.
Weitere Architekturen sind z.B.:

- Inception
- DenseNet
- ResNeXt
- EfficientNet
- MobileNet
- ConvNeXt
- ...

Auf Pytorch.org sind Implementierungen von vielen dieser Modelle frei verfügbar.

Part 2

CNN Visualisierung

Aktivierungen in verschiedenen Layern

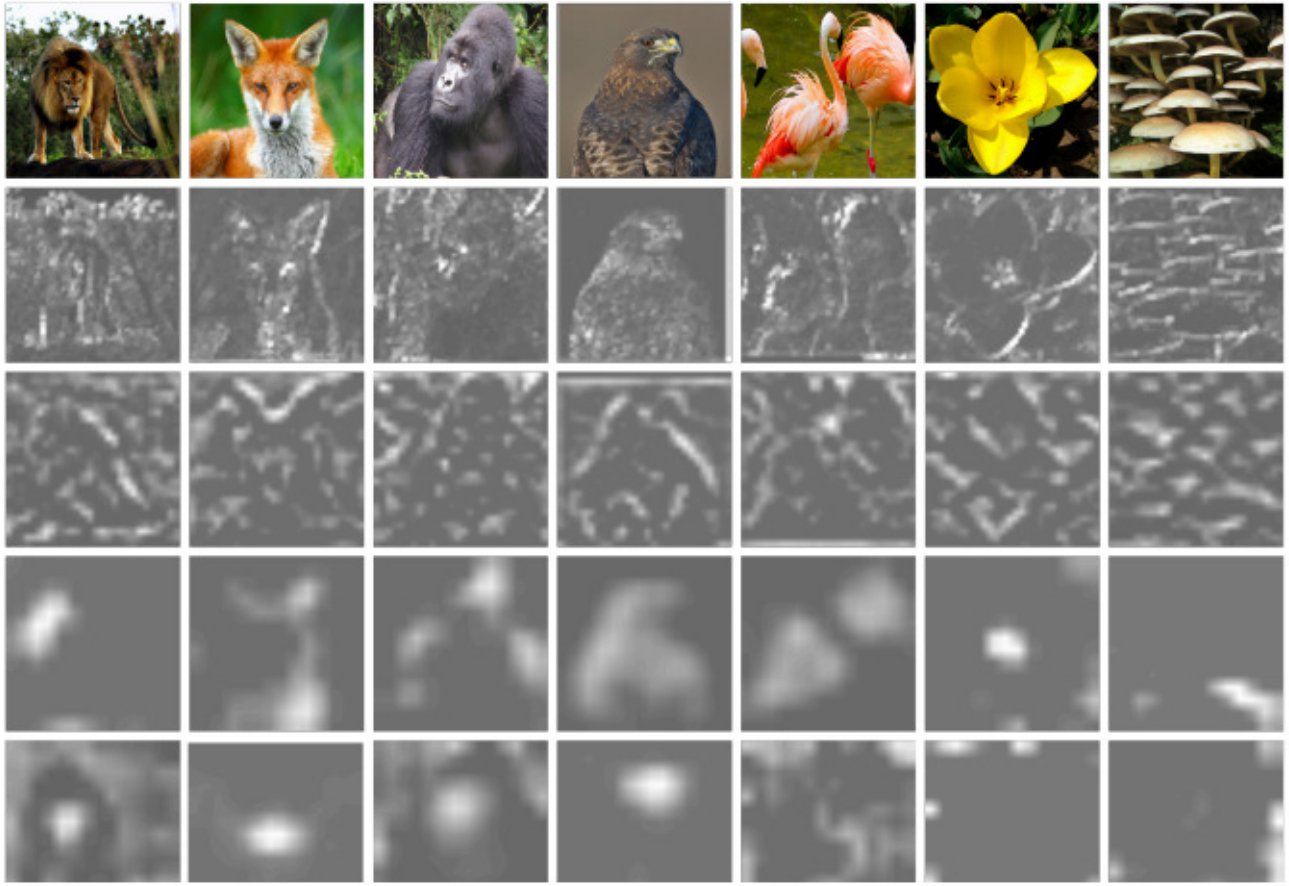


Figure: Abbildung von [hier]. Einfache Features (z.B. Kanten) werden in den ersten Layern extrahiert (siehe oben). Komplexere Features werden in späteren Layern erkannt (siehe unten).

Aktivierungen in verschiedenen Layern

Figure: ... eine Visualisierung sämtlicher Aktivierungen in den verschiedenen Layern eines NN kann sehr aufwändig werden. (Deutsches Museum, Bonn)

Visualisierung: Nearest Neighbors im Feature Space des letzten Layers

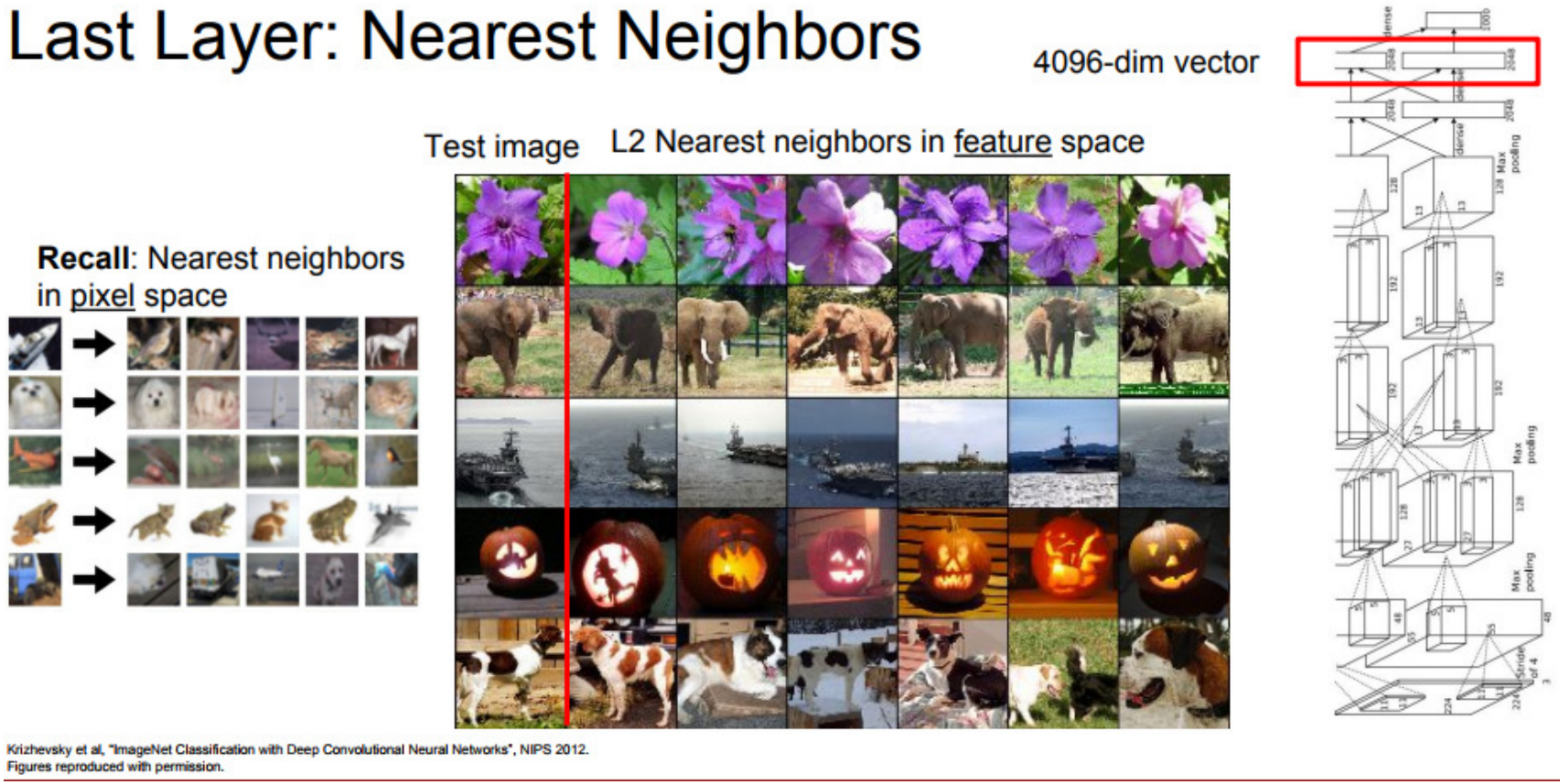


Figure: Slide von [hier].

Occlusion Experiments

Indem wir Bereiche der Eingabe verdecken, und die jeweilige Ausgabe des Netzes beobachten, können wir Rückschlüsse über den Ort von Objekten ziehen.

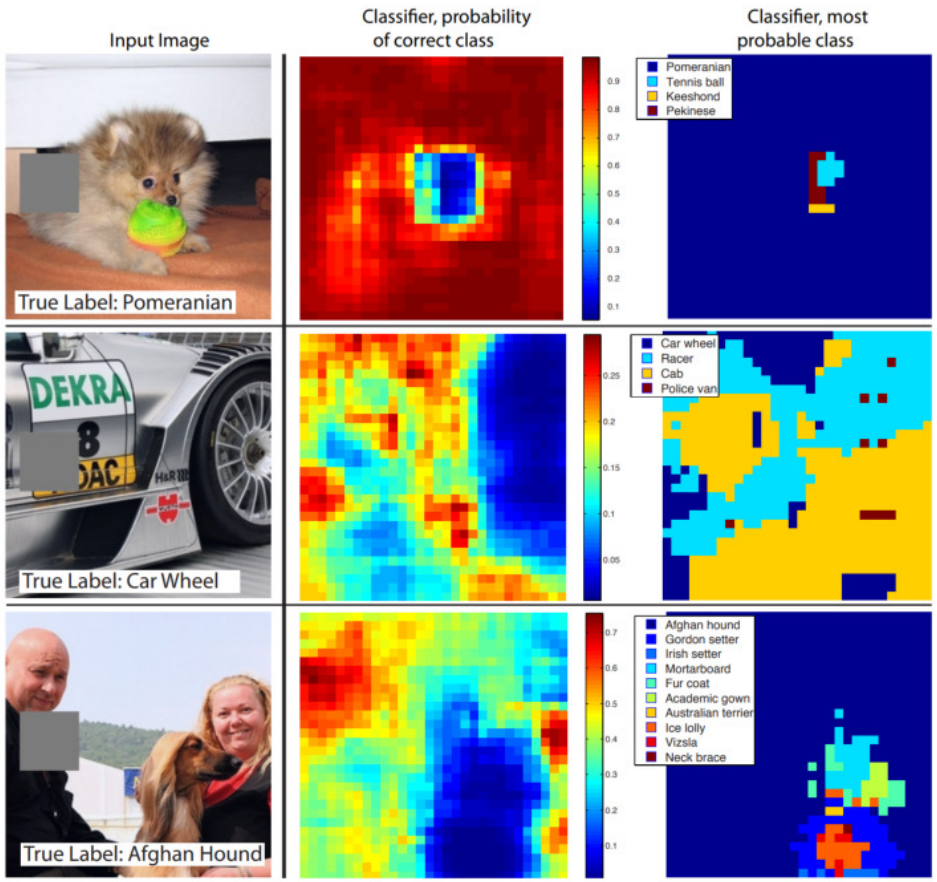


Figure: Abbildung aus [Visualizing and Understanding CNNs]

Visualisierung: Saliency Maps

Eine weitere (effizientere) Möglichkeit um Rückschlüsse über den Ort von Objekten zu ziehen ist, mit Hilfe von Backpropagation den Gradient der Ausgabeklasse bezüglich der Pixelwerte des Eingabebildes zu berechnen.

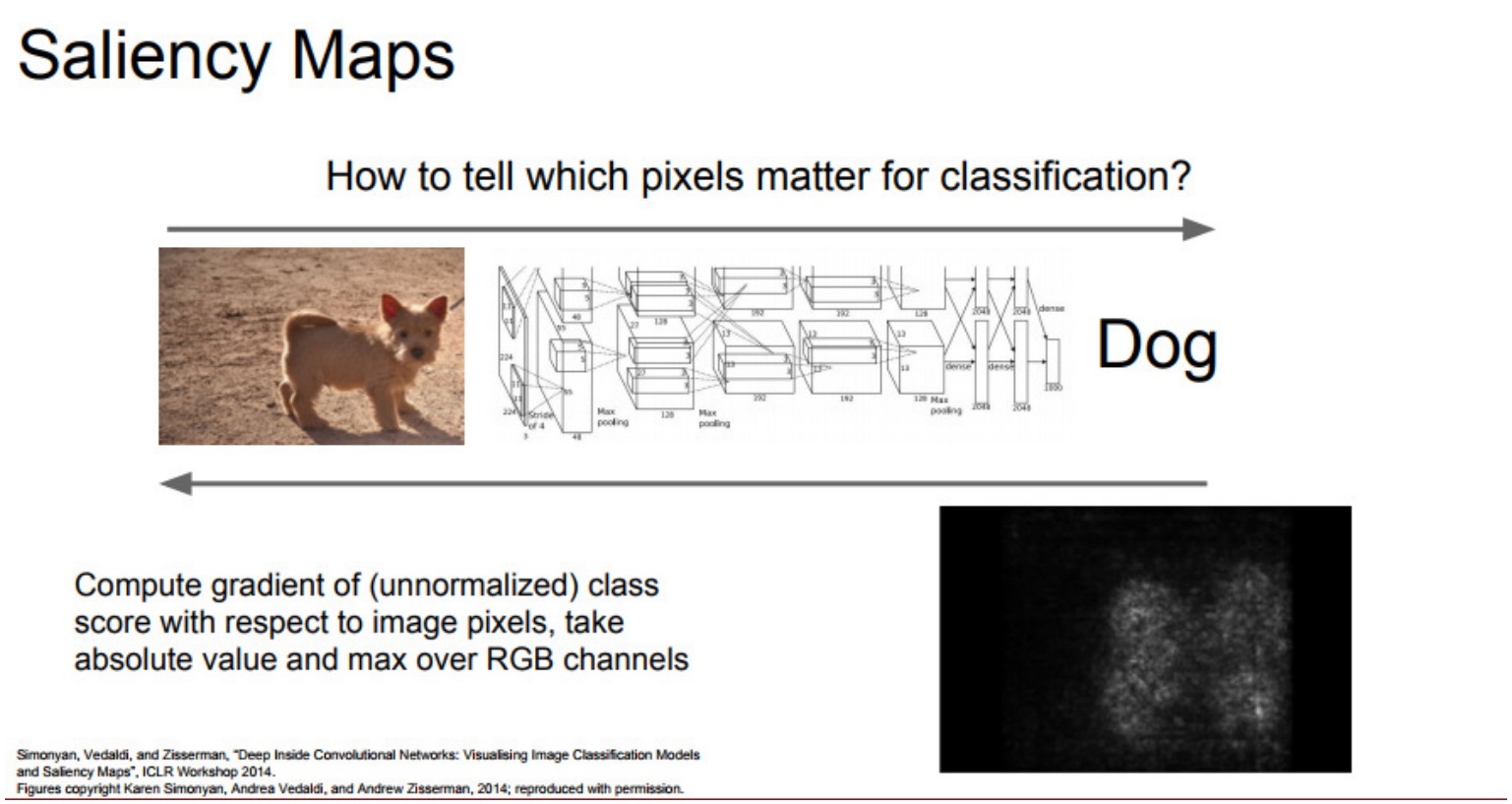


Figure: Slide von [hier]

Saliency Maps

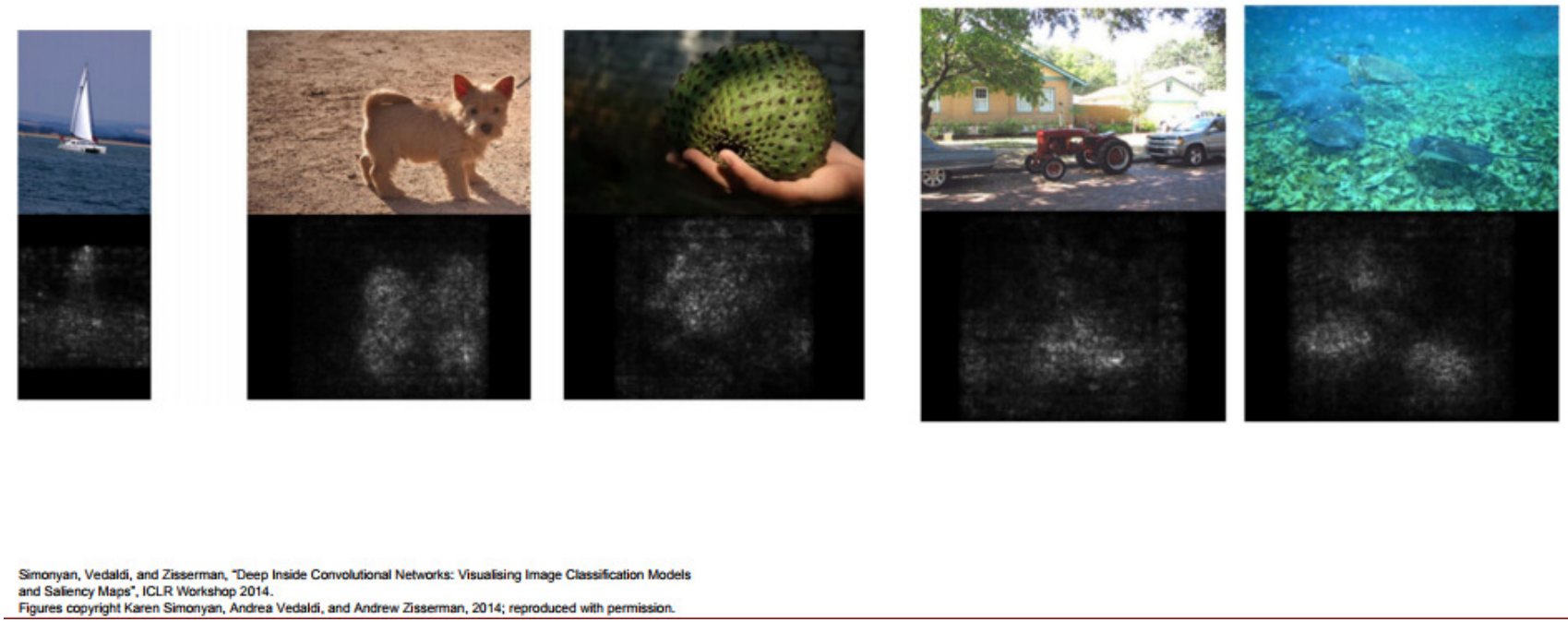


Figure: Slide von [hier]

Visualisierung: Filter Kernel des ersten Layers

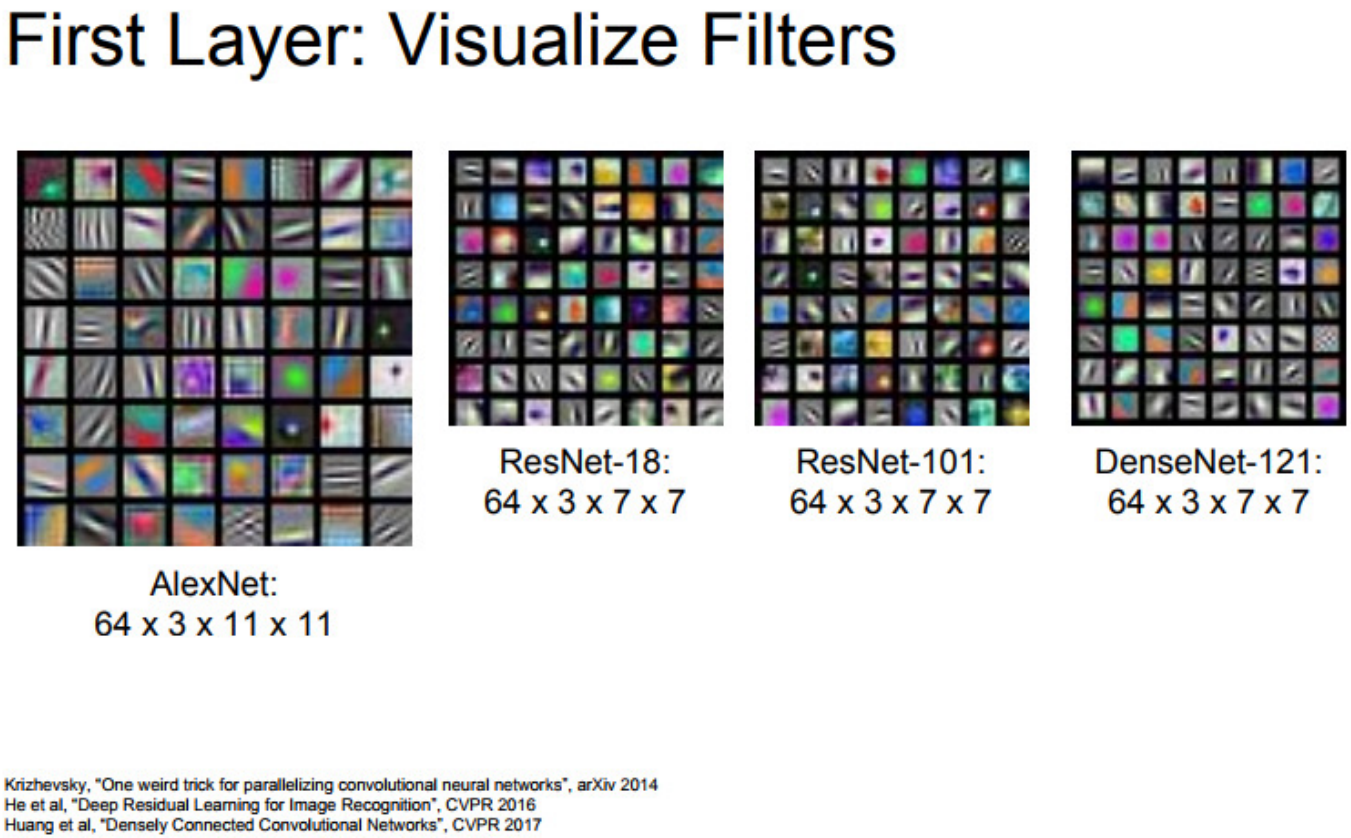


Figure: Slide von [hier]

Visualisierung: Filter Kernels in höheren Layern

Interpretation von Filter-Kerneln in höheren Layern ist schwierig:

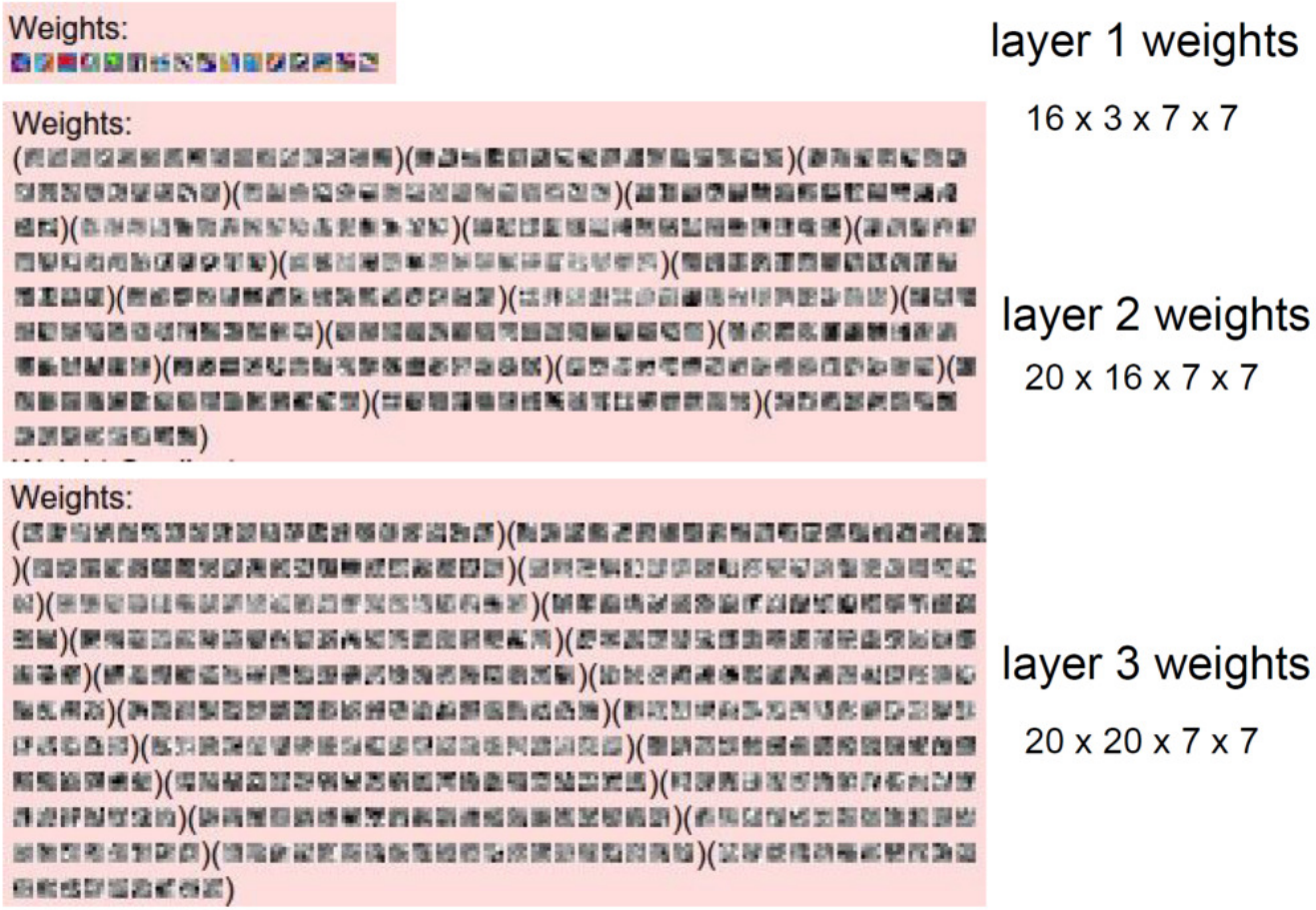


Figure: Slide von [hier]

Visualization: Gradient Ascent

Visualizing CNN features: Gradient Ascent

(Guided) backprop:
Find the part of an image that a neuron responds to

Gradient ascent:
Generate a synthetic image that maximally activates a neuron

$$I^* = \arg\max_I \boxed{f(I)} + \boxed{R(I)}$$

Neuron value

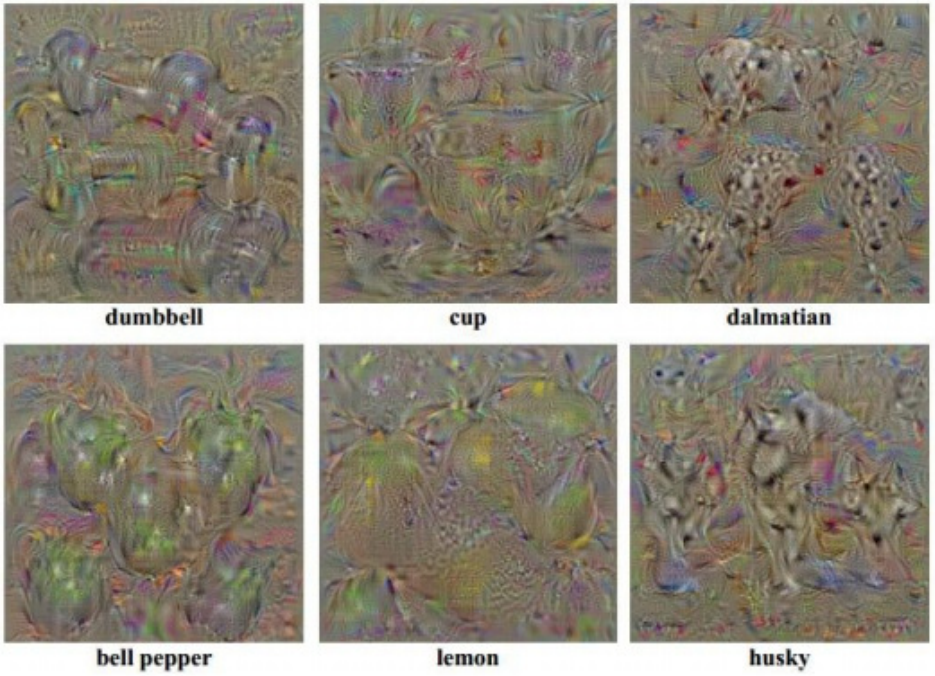
Natural image regularizer

Figure: Slide von [hier]

Visualization: Gradient Ascent

$$\arg\max_I S_c(I) - \boxed{\lambda \|I\|_2^2}$$

Simple regularizer: Penalize L2 norm of generated image



Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.
Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.

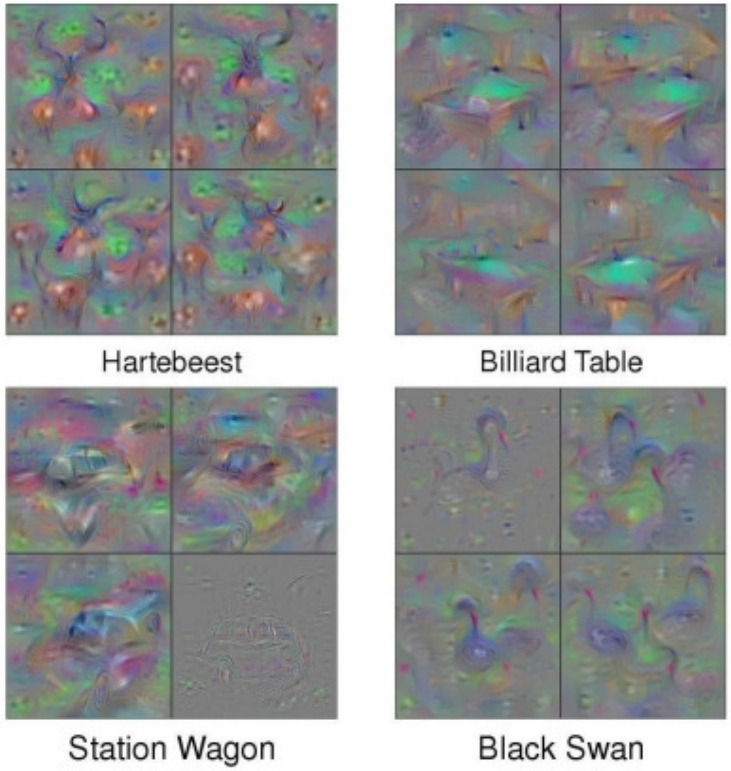
Figure: Slide von [hier]

Visualization: Gradient Ascent

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

Better regularizer: Penalize L2 norm of image; also during optimization periodically

(1) Gaussian blur image
(2) Clip pixels with small values to 0
(3) Clip pixels with small gradients to 0



Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.
Figure copyright Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson, 2014. Reproduced with permission.

Figure: Slide von [hier]

Visualization: Gradient Ascent

In höheren Layern werden die Features immer abstrakter:

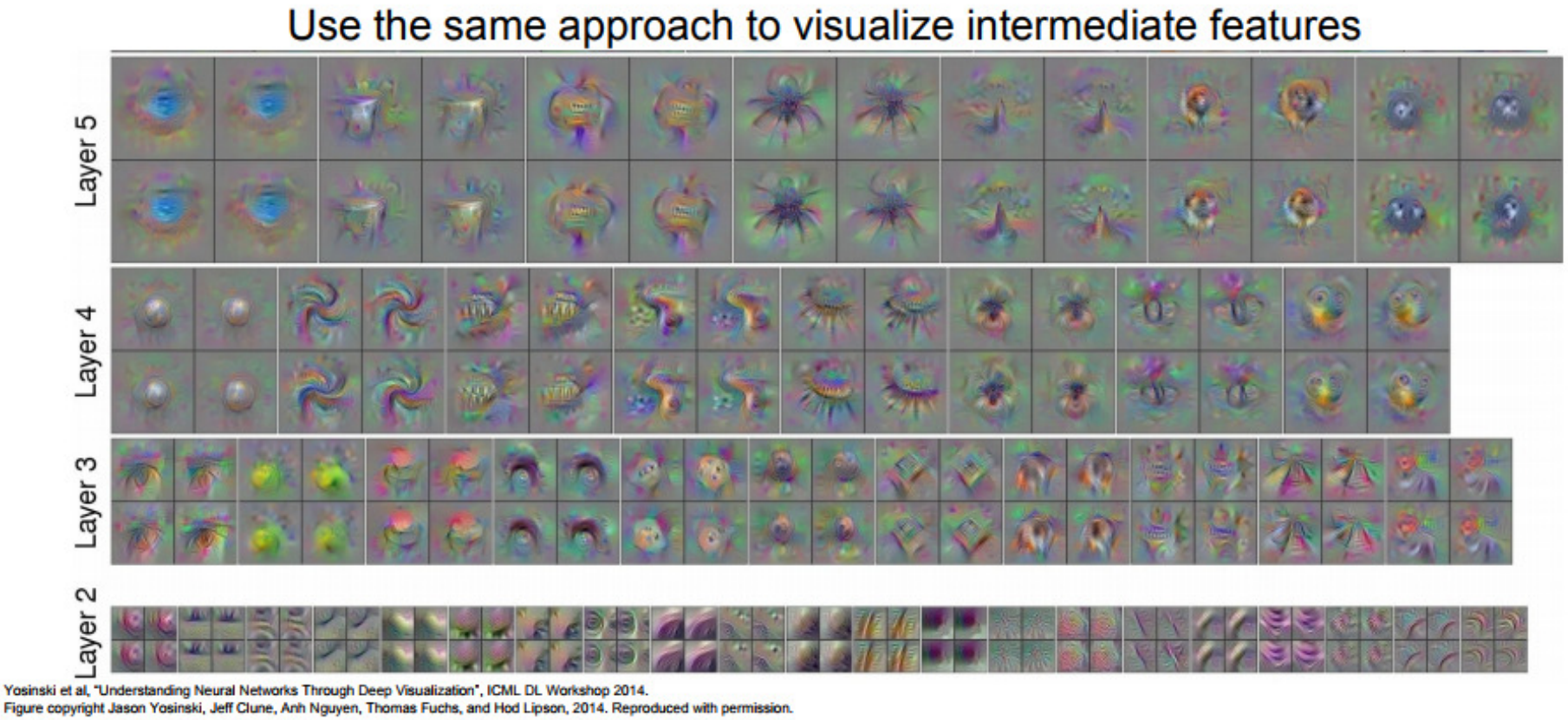
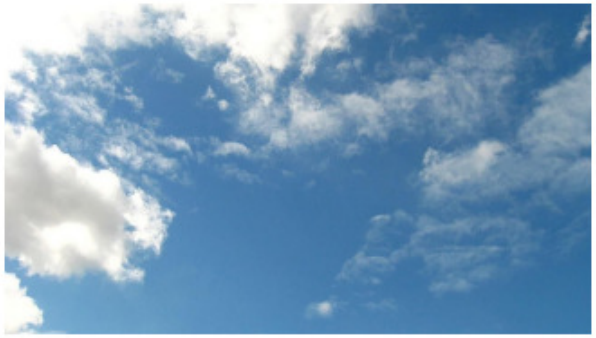


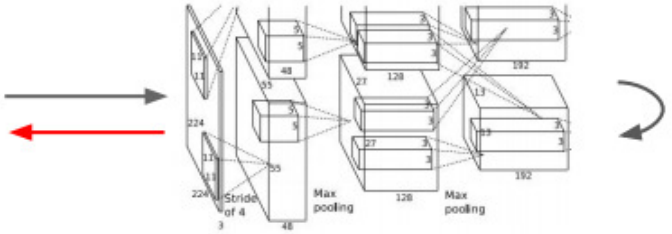
Figure: Slide von [hier]

Visualization: DeepDream

DeepDream: Amplify existing features

Rather than synthesizing an image to maximize a specific neuron, instead try to **amplify** the neuron activations at some layer in the network





Choose an image and a layer in a CNN; repeat:

1. Forward: compute activations at chosen layer
2. Set gradient of chosen layer *equal to its activation*
3. Backward: Compute gradient on image
4. Update image

Equivalent to:

$$I^* = \arg\max_I \sum_i f_i(I)^2$$

Mordvintsev, Olah, and Tyka, "Inceptionism: Going Deeper into Neural Networks", Google Research Blog. Images are licensed under CC-BY

Figure: Slide von [hier]

Visualization: DeepDream

Figure: Slide von [hier]

Visualization: DeepDream

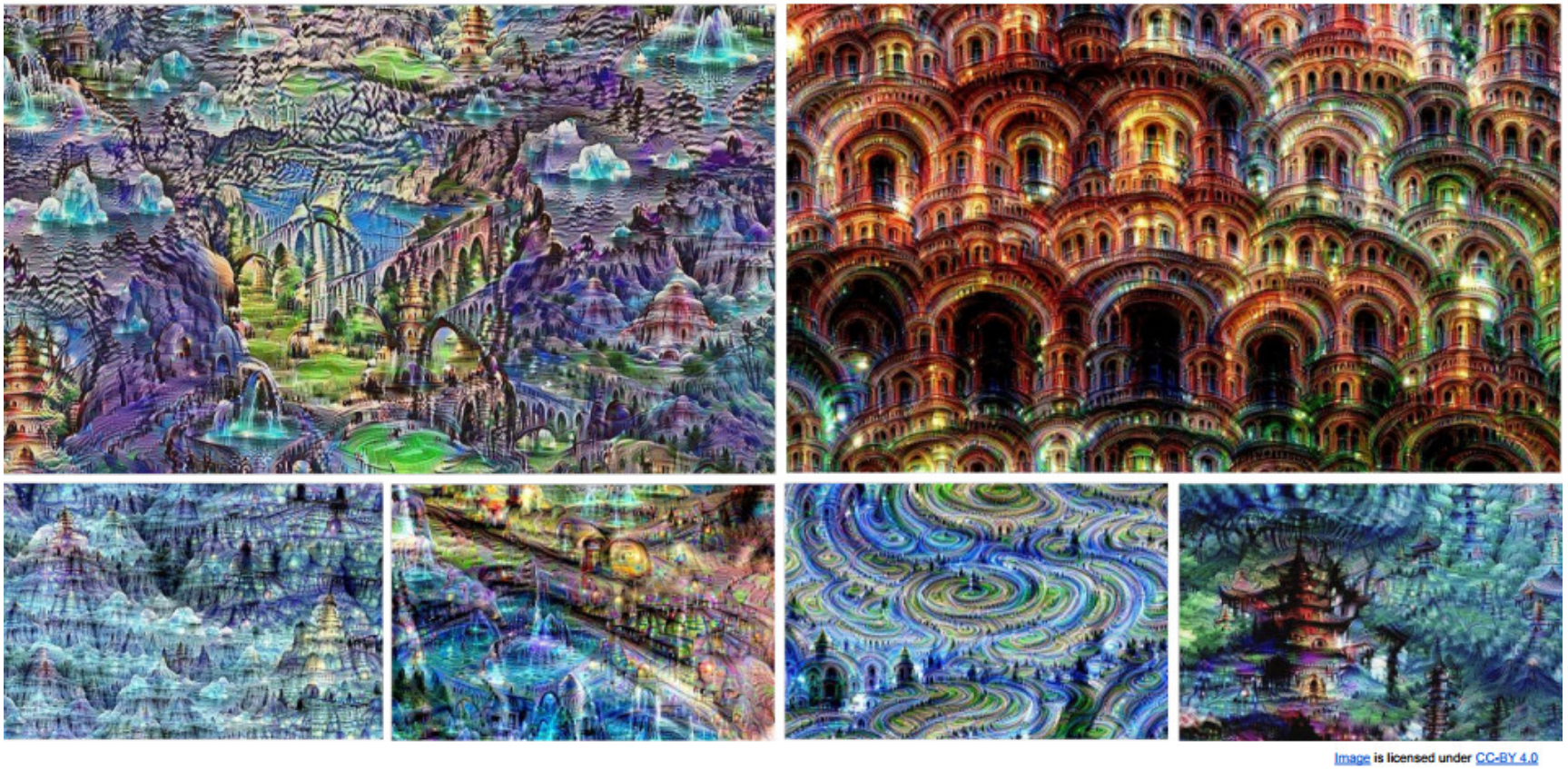


Figure: Slide von [hier]

Hier gibt es ein [Video], welches mit Hilfe von DeepDream Features aus verschiedenen Layern eines CNNs visualisiert.

Feature Inversion

Given a CNN feature vector for an image, find a new image that:

- Matches the given feature vector
- "looks natural" (image prior regularization)

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathbb{R}^{H \times W \times C}}{\operatorname{argmin}} \ \ell(\Phi(\mathbf{x}), \Phi_0) + \lambda \mathcal{R}(\mathbf{x})$$

Given feature vector

Features of new image

$$\ell(\Phi(\mathbf{x}), \Phi_0) = \|\Phi(\mathbf{x}) - \Phi_0\|^2$$

$$\mathcal{R}_{V^\beta}(\mathbf{x}) = \sum_{i,j} \left((x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2 \right)^{\frac{\beta}{2}}$$

Total Variation regularizer
(encourages spatial smoothness)

Mahendran and Vedaldi, "Understanding Deep Image Representations by Inverting Them", CVPR 2015

Figure: Slide von [hier]

Feature Inversion

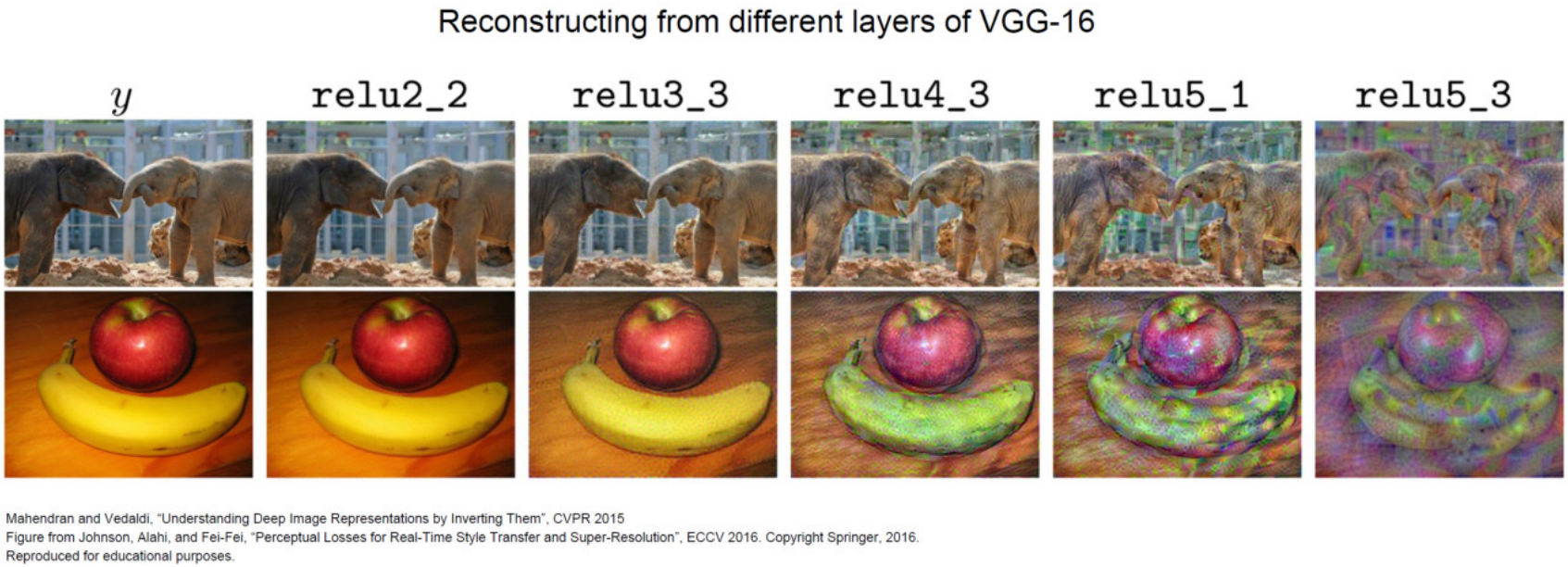


Figure: Slide von [hier]

Textur-Synthese

Given a sample patch of some texture, can we generate a bigger image of the same texture?

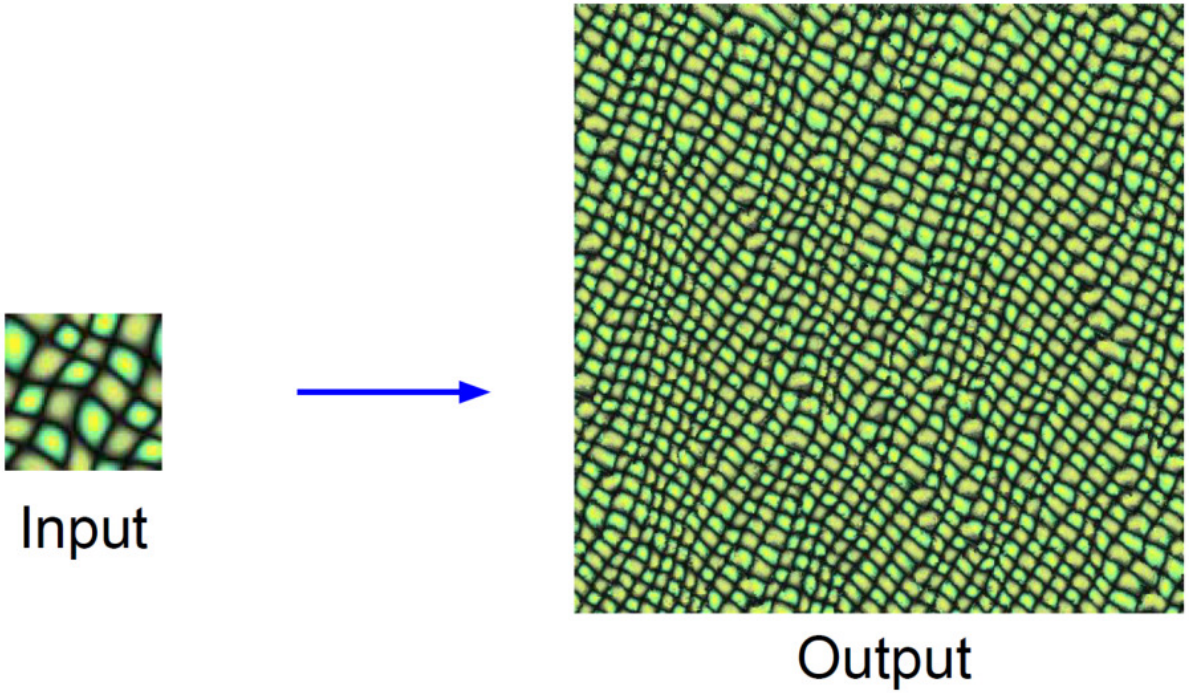
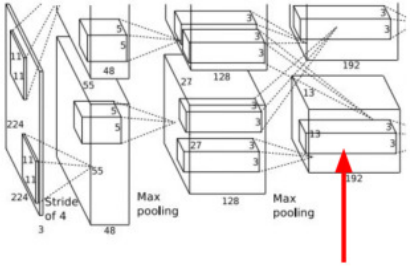
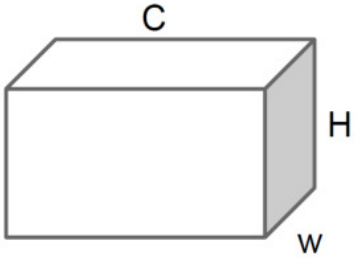


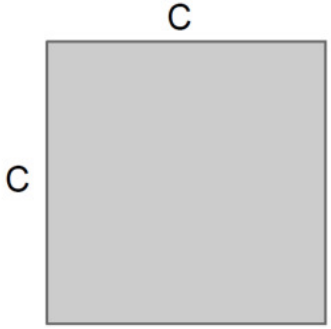
Figure: Slide von [hier]

Textur-Synthese

This image is in the public domain.







Gram Matrix

Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

Outer product of two C-dimensional vectors gives C x C matrix measuring co-occurrence

Average over all HW pairs of vectors, giving **Gram matrix** of shape C x C

Figure: Slide von [hier]

Textur-Synthese

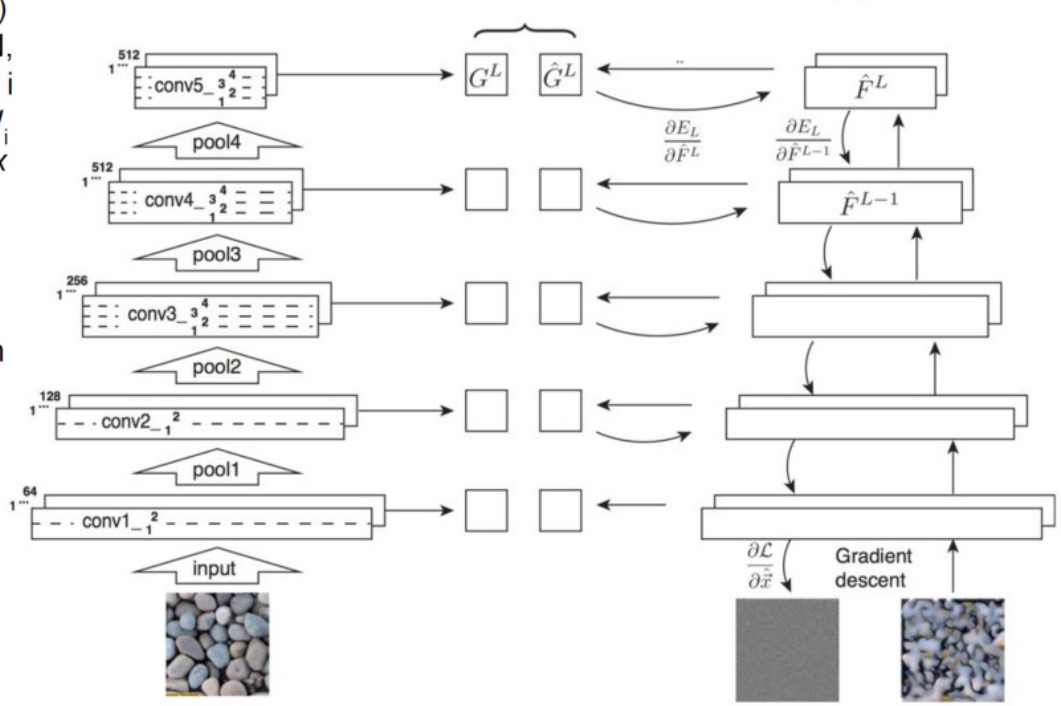
Neural Texture Synthesis

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - \hat{G}_{ij}^l\right)^2 \qquad \mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^{L} w_l E_l$$

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the *Gram matrix* giving outer product of features:

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l \text{ (shape } C_i \times C_i\text{)}$$

4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5

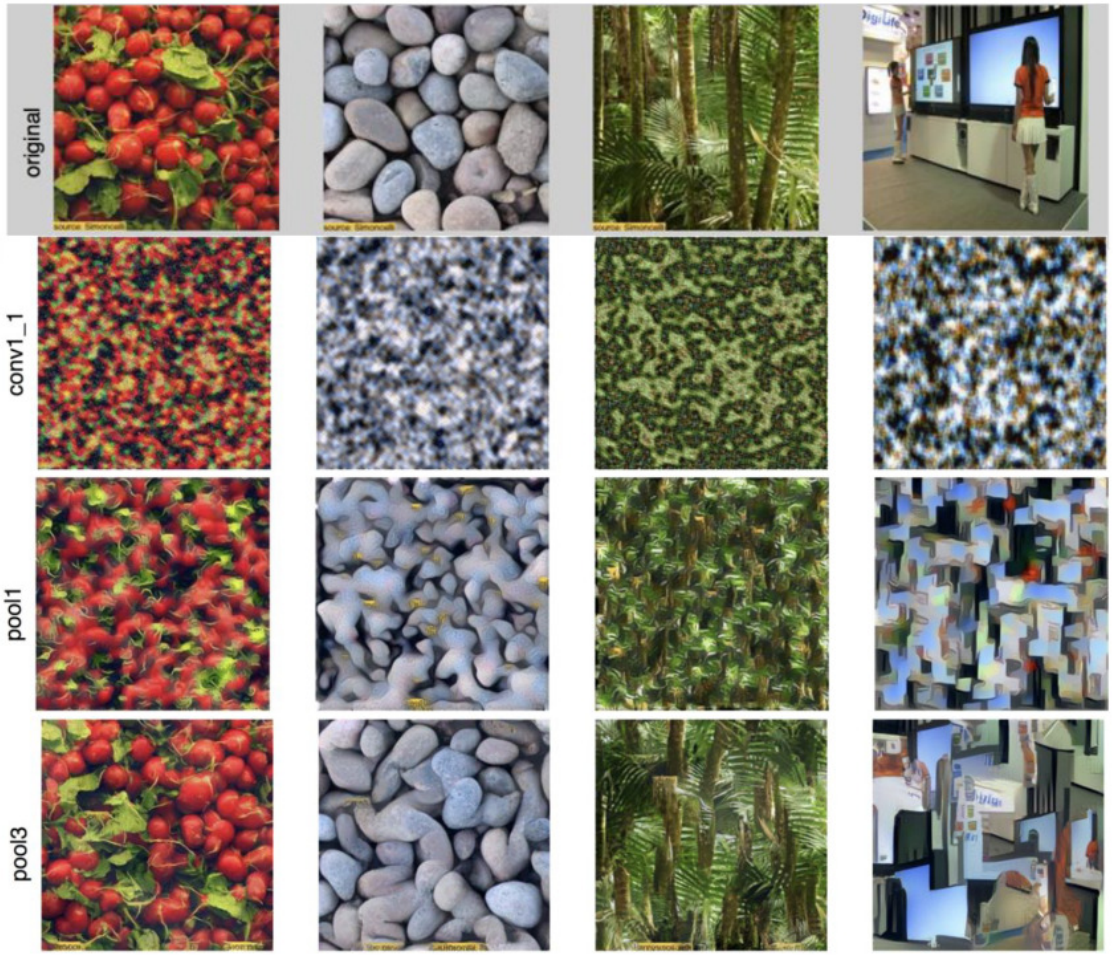


Gatys, Ecker, and Bethge, "Texture Synthesis Using Convolutional Neural Networks", NIPS 2015
Figure copyright Leon Gatys, Alexander S. Ecker, and Matthias Bethge, 2015. Reproduced with permission.

Figure: Slide von [hier]

Textur-Synthese

Textur-Synthese aus höheren Layern erzeugt grössere / abstraktere Features aus der Eingabetextur.



Gatys, Ecker, and Bethge, "Texture Synthesis Using Convolutional Neural Networks", NIPS 2015
Figure copyright Leon Gatys, Alexander S. Ecker, and Matthias Bethge, 2015. Reproduced with permission.

Figure: Slide von [hier]

Style Transfer

Idee: Kombiniere Textur-Synthese mit Feature-Rekonstruktion

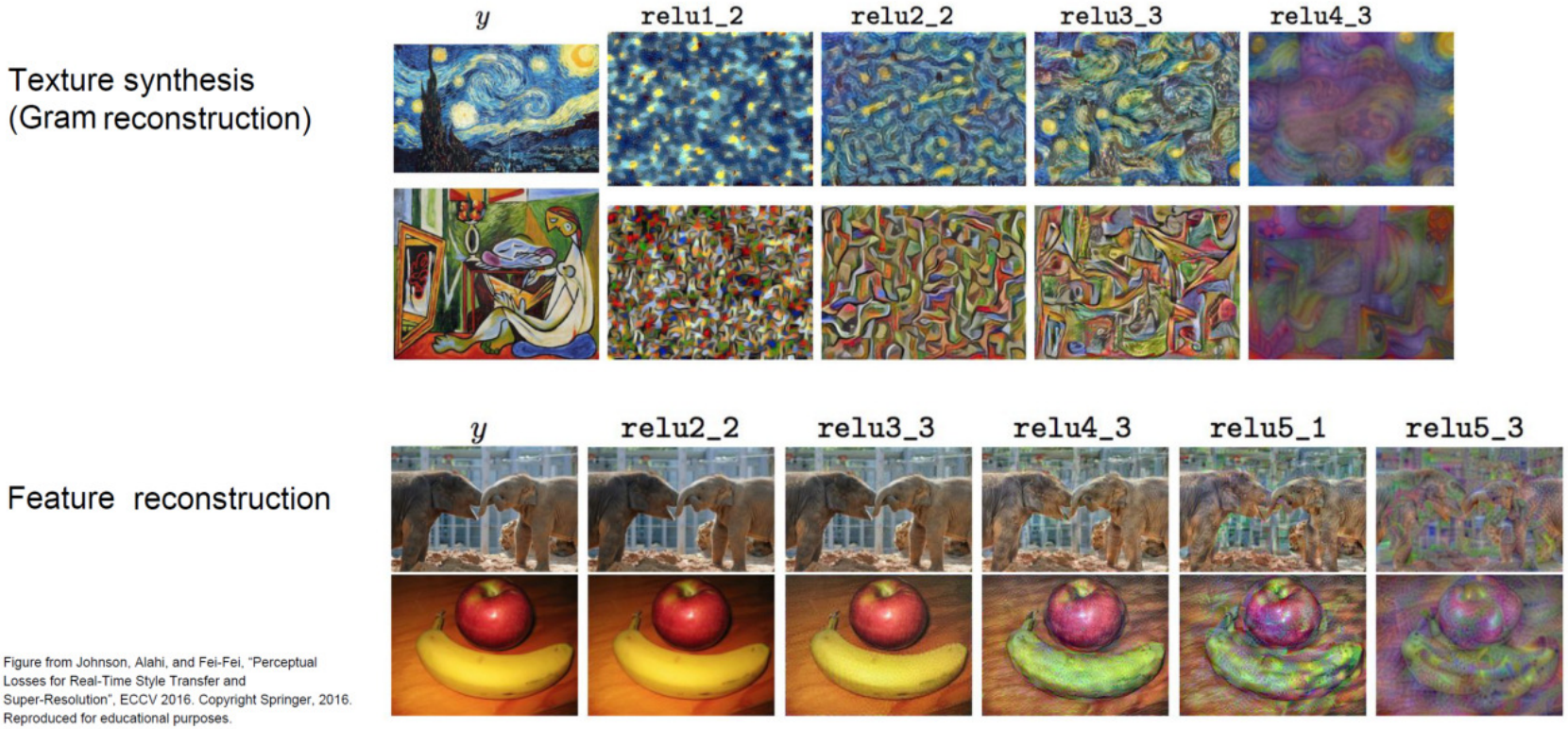


Figure: Slide von [hier]

Style Transfer

Figure: Slide von [hier]

Style Transfer

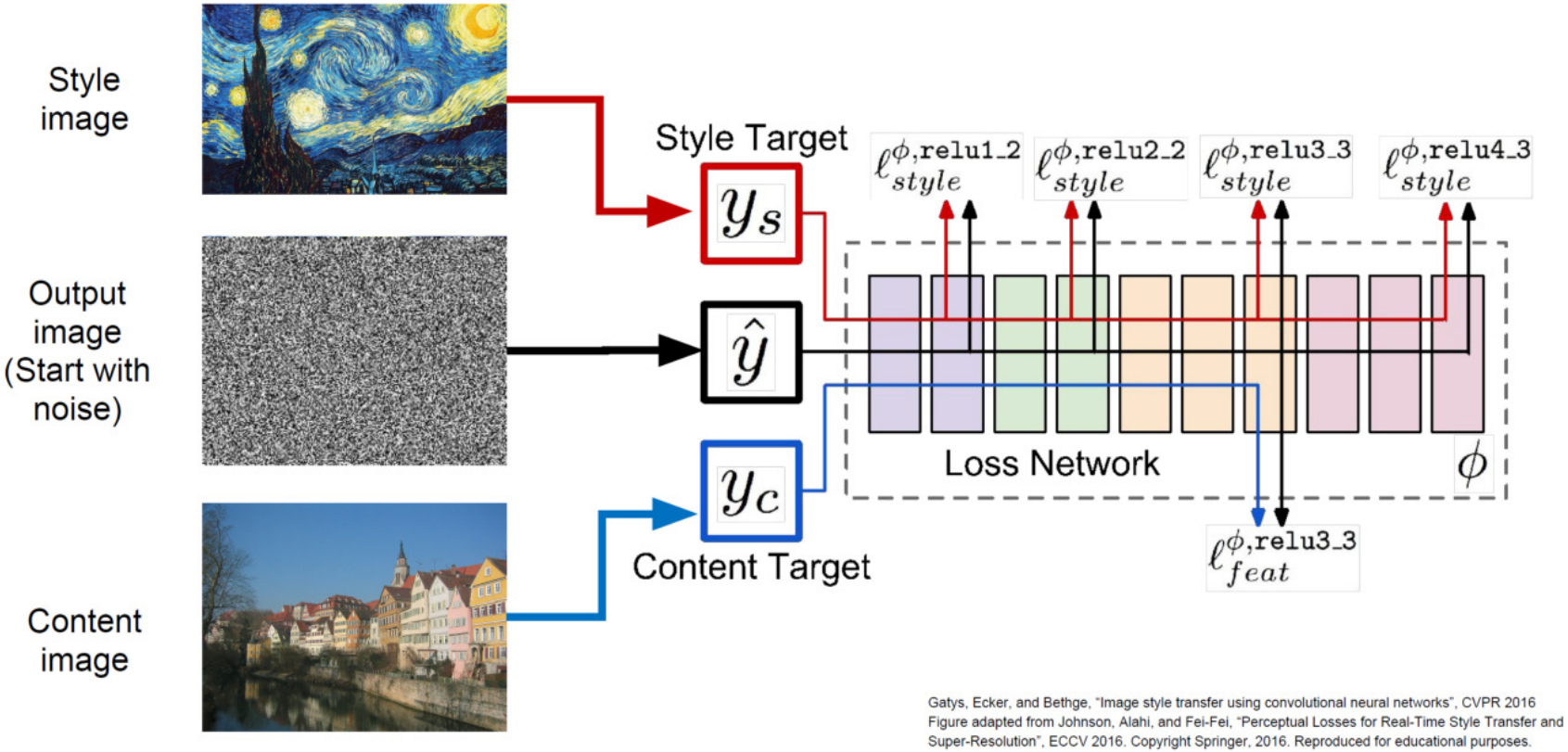


Figure: Slide von [hier]

Style Transfer

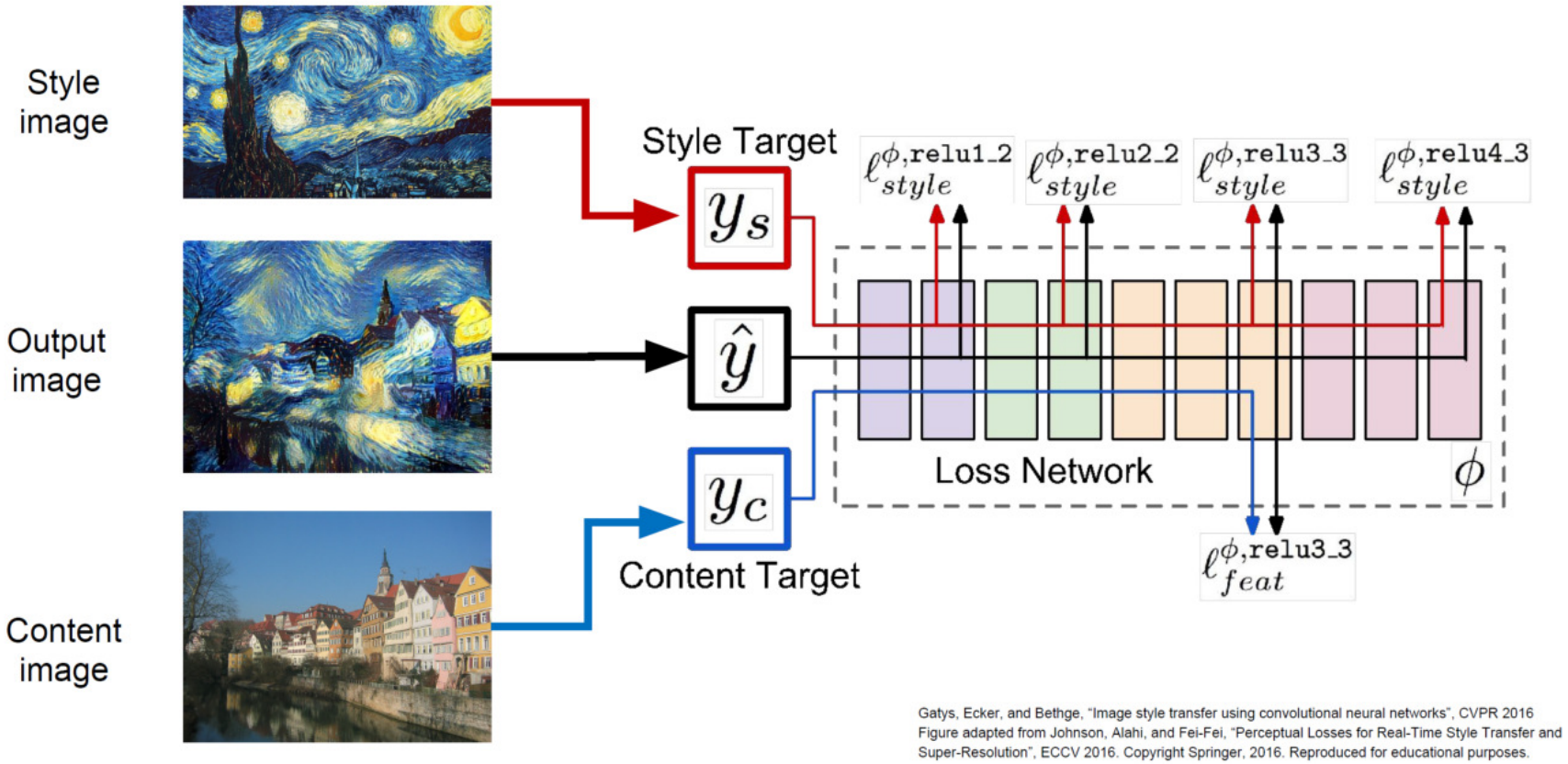


Figure: Slide von [hier]

Style Transfer

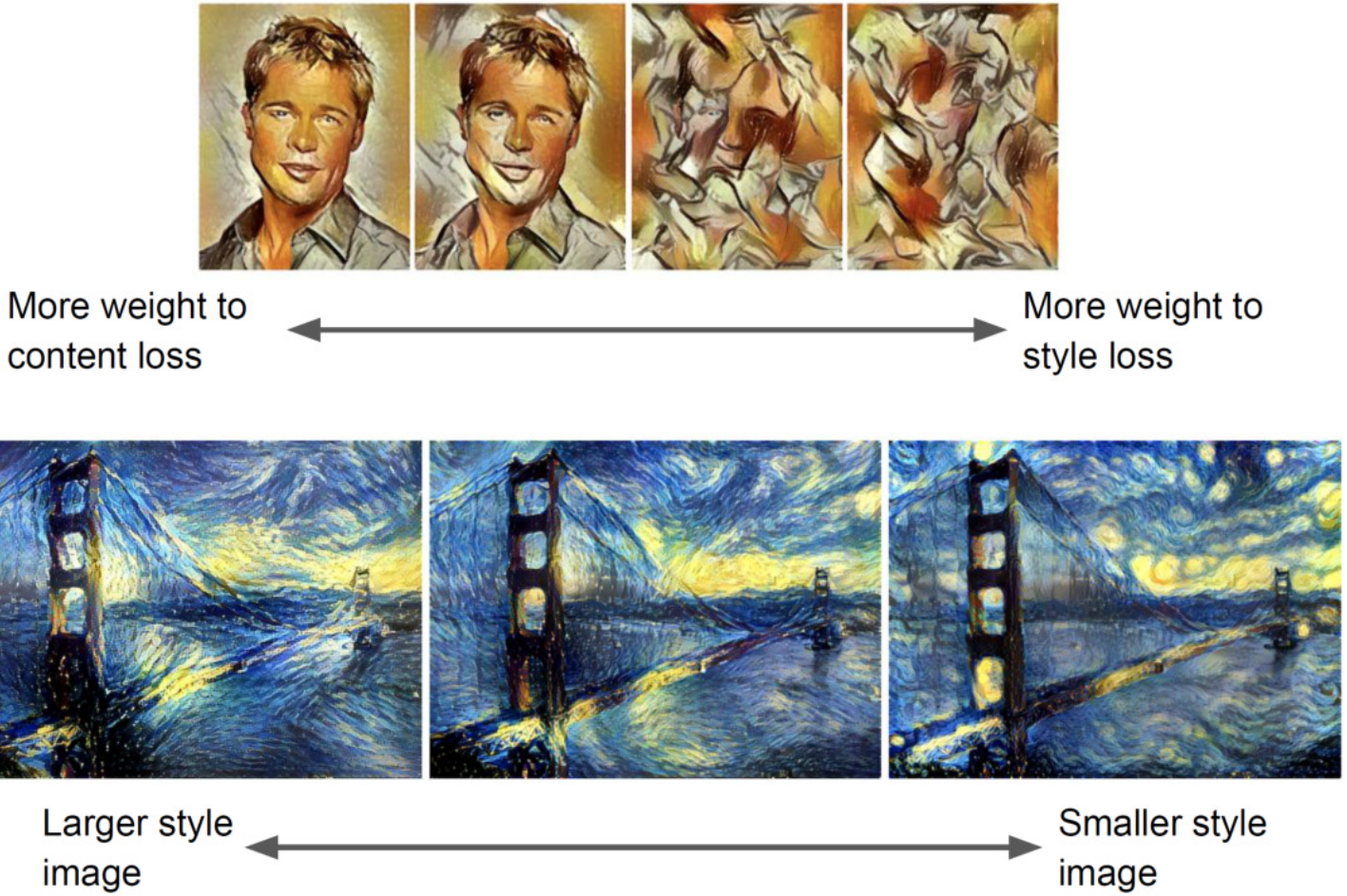


Figure: Slide von [hier]

Style Transfer

Mit Hilfe von z.B. Deepart.io kann man einfach eigene Bilder stilisieren.

Figure

Adversarial Attack

Mit Hilfe von Gradient Descent kann das Eingabebild bereits mit sehr wenig Noise so modifiziert werden, dass ein CNN eine beliebige falsche Klasse vorhersagt. Solche Angriffe werden auch als "adversarial attacks" bezeichnet.

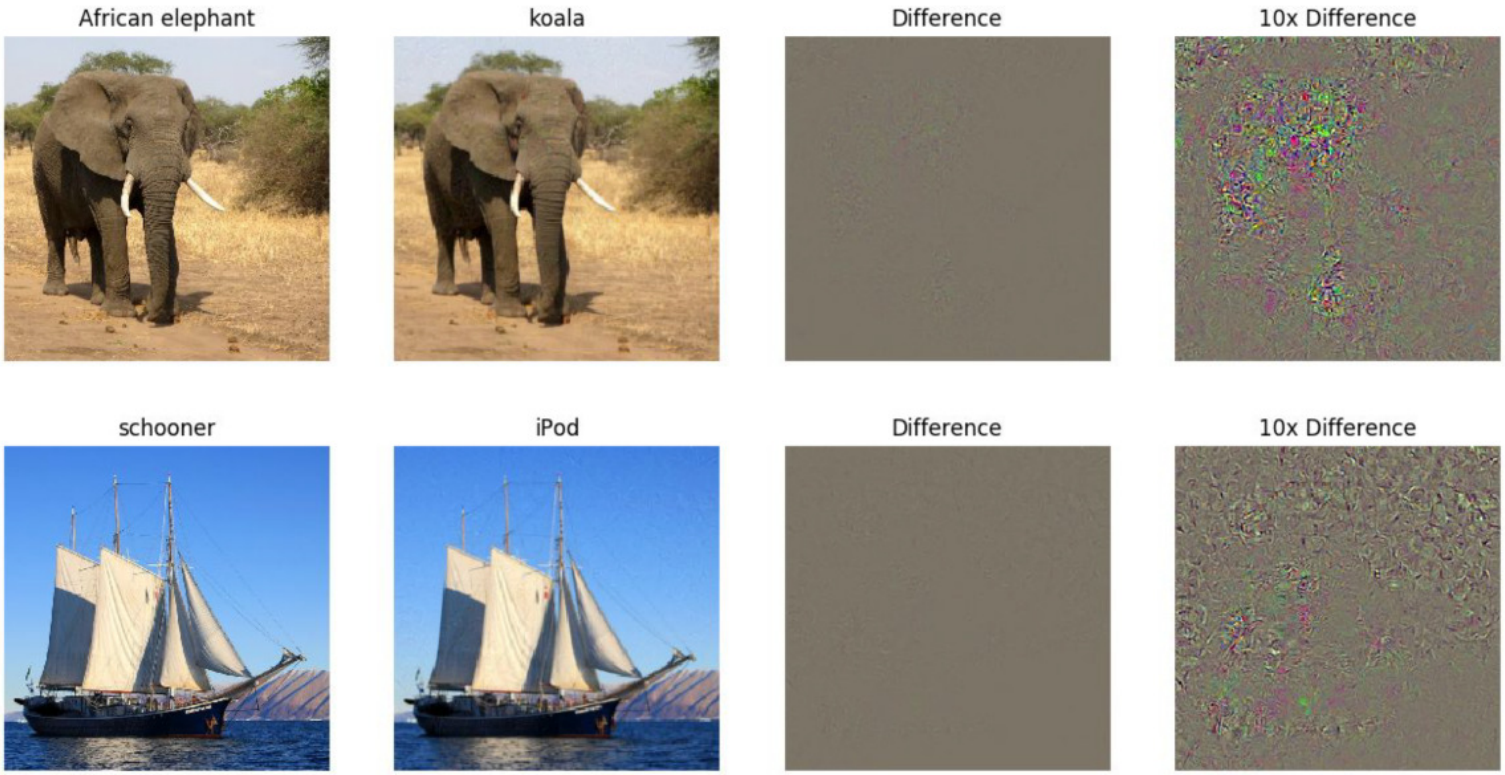


Figure: Beispiele für eine Adversarial Attack (von links: ursprüngliches Eingabebild, modifiziertes Bild, Differenz zwischen dem ursprünglichen und dem modifizierten Bild, Differenz x 10). Slide von [hier]

Part 3

Transfer Learning

Transfer Learning

CNNs als Feature-Extractors

Trainierte CNNs extrahieren zahlreiche Features aus Bildern, welche auch in weiteren neuen Kontexten, welche nicht im Training enthalten sind, nützlich sein können. (z.B. können Features zur Klassifizierung von Katzen auch nützlich zur Klassifizierung von Hunden sein)
Wichtige Punkte, die man bei Transfer Learning beachten sollte:

- wie ähnlich sind die Aufgaben / Datensets?
- wie gross ist das neue Datenset?

Fine-tuning von vortrainierten Modellen

Das "Vorwissen" eines vortrainierten Modells kann auf verschiedene Möglichkeiten auf eine neue Aufgabe / Datenset transferiert und wiederverwendet werden:

- Gewichtsinitialisierung: vortrainierte Modelle können einen guten Startpunkt für die gradientenbasierte Optimierung bilden und somit den Trainingsaufwand verringern.
- Fixierung von Netzwerkgewichten: Durch Fixierung von Netzwerkgewichten in den frühen Layern des NN können sinnvolle Features extrahiert und gleichzeitig die Modellkapazität reduziert werden. Dadurch kann Overfitting auf kleineren Datensätzen vermieden werden.

Transfer Learning

Vortrainierte Modelle

Es existieren zahlreiche Websiten, auf denen vortrainierte Modelle heruntergeladen werden können. z.B.:

- Pytorch bzw Pytorch Hub
- OpenMMLab
- HuggingFace
- Modelzoo
- ...

../deeplearn