

# Vorlesung Computational Intelligence:

## Teil 3: Künstliche Neuronale Netze

Kohonen-Karten, Deep Learning, Kommentare

**Ralf Mikut, Wilfried Jakob, Markus Reischl**

Karlsruher Institut für Technologie, Institut für Automation und angewandte Informatik

E-Mail: [ralf.mikut@kit.edu](mailto:ralf.mikut@kit.edu), [wilfried.jakob@kit.edu](mailto:wilfried.jakob@kit.edu)

jeden Donnerstag 14:00-15:30 Uhr, Nusselt-Hörsaal

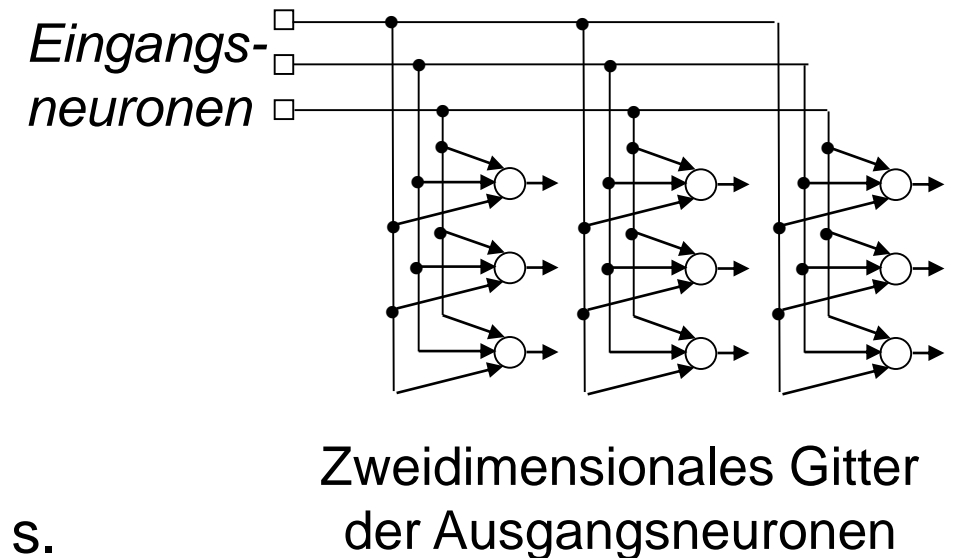
- 3 Künstliche Neuronale Netze
  - 3.1 Vom Biologischen zum Künstlichen Neuronalen Netz
  - 3.2 Struktur
  - 3.3 Lernverfahren (Fortsetzung)
  - 3.4 Multi-Layer-Perceptron-Netze (MLP-Netze)
  - 3.5 Radial-Basis-Funktions-Netze (RBF-Netze)
  - 3.6 Kohonen-Karten**
  - 3.7 Deep Learning & Convolutional Neural Networks
  - 3.8 Kommentare

- Synonyme
  - Self-Organizing Map (SOM)
  - Self-Organizing Feature Map (SOFM)
- Biologische Motivation:
  - Aufbau und Funktion der (menschlichen) Hirnrinde (cerebral cortex)
  - nur ca. 2 mm dick, enthält Milliarden von Neuronen und Hunderte von Milliarden von Synapsen
  - Bereiche können bestimmten senso-motorischen Funktionen zugeordnet werden, z. B. dem Hören, dem Sehen, der Motorik.
  - Ähnliche Muster werden auf räumlich benachbarte Neuronen abgebildet
- Hieraus wurde durch Kohonen das Prinzip der Bildung topografischer Karten abgeleitet:

*Die räumliche Lage eines Ausgangsneurons in der topografischen Karte entspricht einem Bereich oder einem Merkmal in den Eingangsdaten.*
- Durch SOM werden Eingangsdaten mit hoher Dimension (viele Einzelmerkmale) häufig auf Karten niedriger Dimension (meist ein- oder zweidimensional) abgebildet.

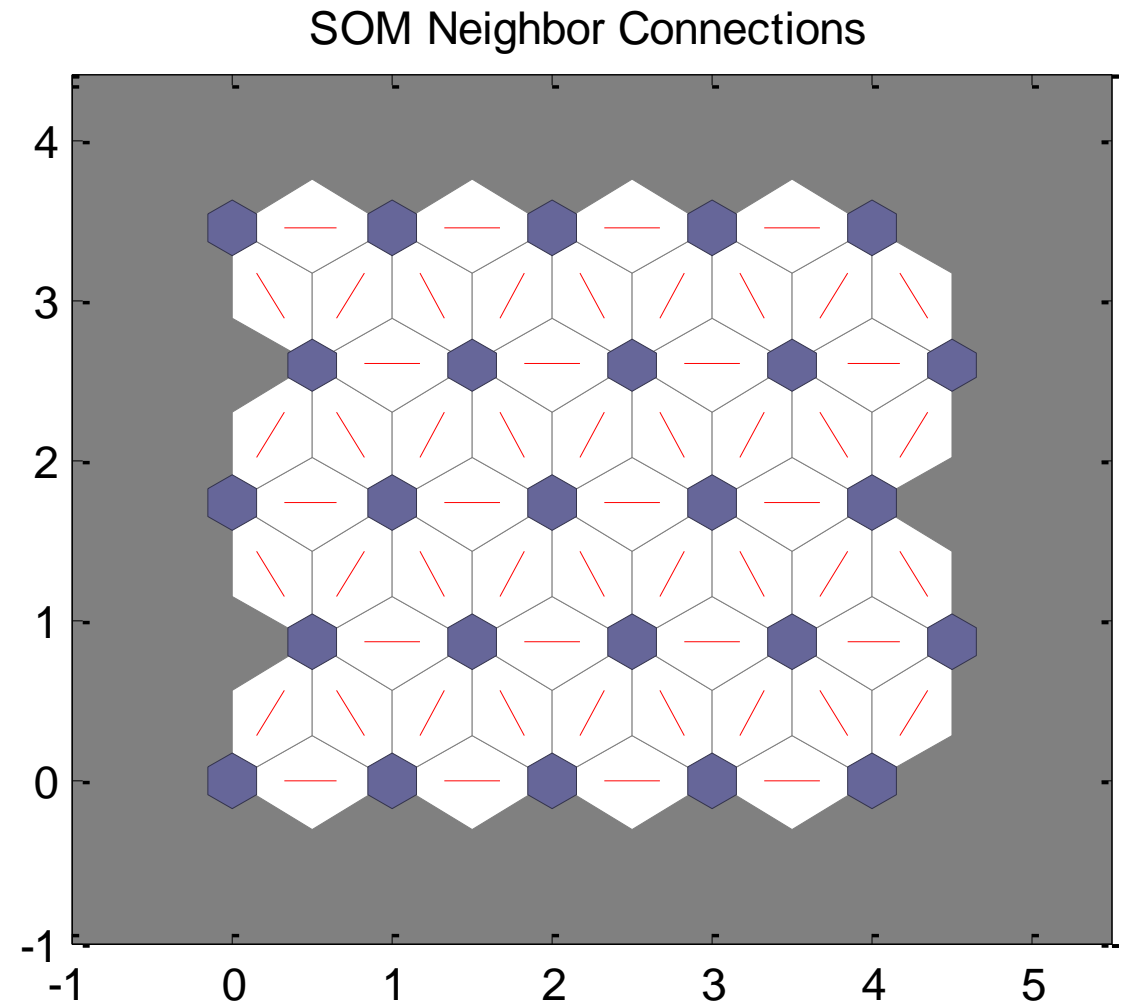
# Aufbau und Funktionsweise von SOMs

- Entsprechend dem biologischen Vorbild werden in einem SOM die Ausgangsneuronen in einem ein- oder zweidimensionalen Gitter angeordnet.
- Feedforward-Netz
- Der Eingangsvektor  $\mathbf{x}$  hat  $s$  Komponenten. Dementsprechend hat der Parametervektor des  $j$ -ten Ausgangsneurons  $\mathbf{w}_j$  die Dimension  $s$ .
- Wird dem SOM ein Eingangsvektor vorgelegt, wird bei einem angelerten Netz lediglich eine Gruppe benachbarter Ausgangsneuronen aktiviert.
- Berechnung des Zustands:
  - über Distanz, meist Verwendung des Euklidischen Abstands:  $z_j = \|\mathbf{x} - \mathbf{w}_j\|$
  - Alternative: Verwendung des inneren Produkts bei normalisierten Gewichtsvektoren  $\mathbf{w}$  und Eingangsvektoren  $\mathbf{x}$ :  $z_j = \mathbf{x}^T \mathbf{w}_j$
- Ausgang des Netzes: Gewinnerneuron  $j(\mathbf{x})$  (*winner-takes-all*-Prinzip) ergibt sich mit  $j(\mathbf{x}) = \operatorname{argmin} z_j$ .



# Nachbarschaften

- Nachbarschaftsstruktur ist fest vorgegeben
- Beispiel:
  - zweidimensional
  - je 5 Neuronen pro Dimension
  - Neuronen in der Mitte haben 6 Nachbarn
  - Neuronen am Rand haben 2-4 Nachbarn



1. Festlegen der Dimension des Gitters und der Anzahl der Ausgangsneuronen
2. zufällige Initialisierung der Gewichte  $\mathbf{w}_{SOM,i}[0]$
3. zufällige Auswahl eines Datentupels  $\mathbf{x}[n]$
4. Bestimmung des Gewinnerneuron wird für dieses Datentupel bestimmt
5. Bestimmung der Nachbarn für das Gewinnerneuron
6. Gewinnerneuron (besonders stark) und dessen Nachbarn (etwas weniger) werden in Richtung von  $\mathbf{x}[n]$  verschoben,  $k$ : Iterationsschritt:

$$\mathbf{w}_{SOM,i}[k+1] = \mathbf{w}_{SOM,i}[k] + \rho_{i,i_G[k]}[k](\mathbf{x}[k] - \mathbf{w}_{SOM,i}[k])$$

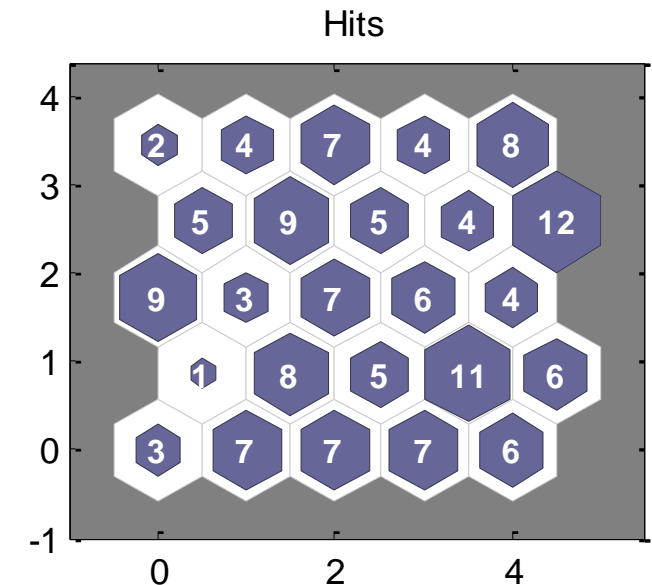
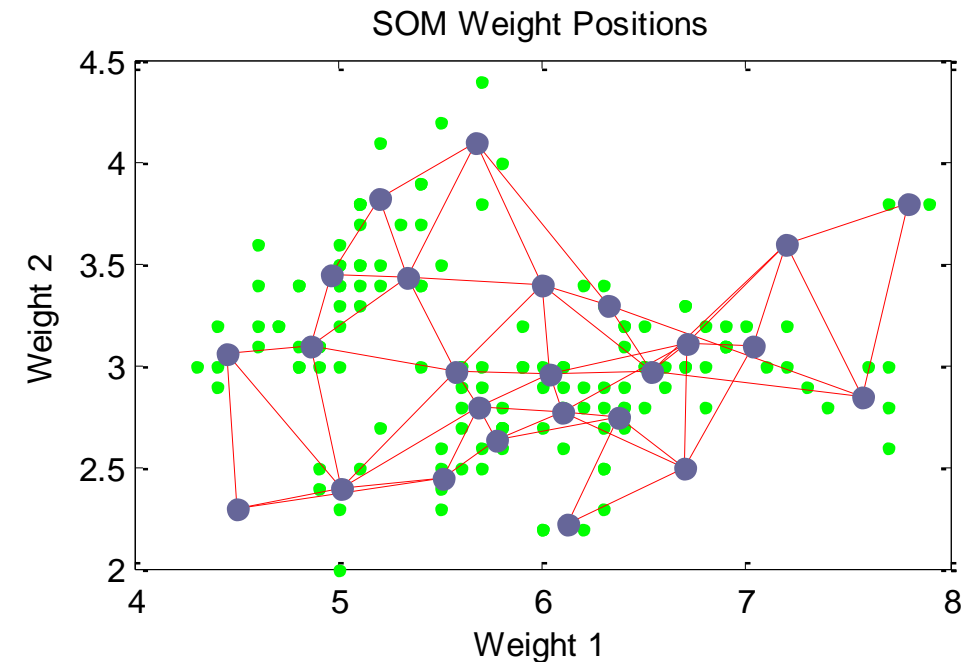
$$i_G[k] = \operatorname{argmin}_i d(\mathbf{w}_{SOM,i}[k], \mathbf{x}[k])$$

$$\rho_{i,j}[k] = \rho_0[k] \cdot \exp(-d(\mathbf{p}_i, \mathbf{p}_j)) \text{ mit } \rho_{i_G[k],i_G[k]}[k] = \rho_0[k] \geq \rho_{i,i_G}[k]$$

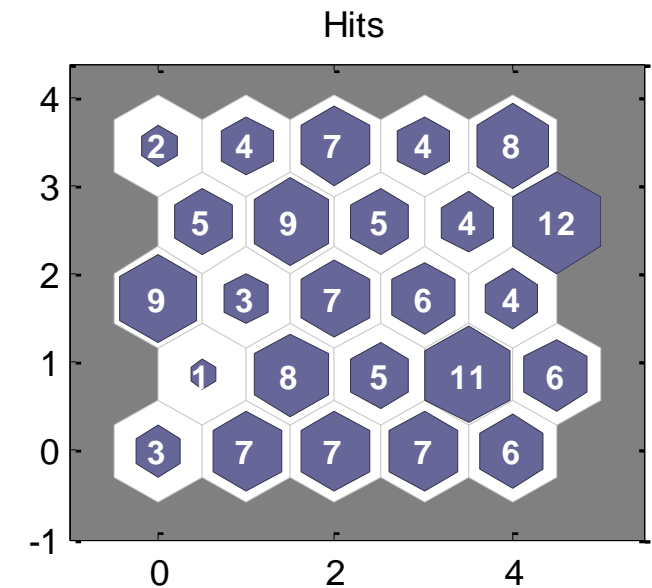
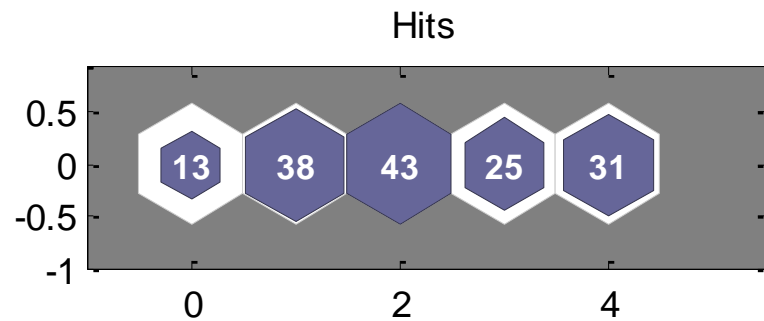
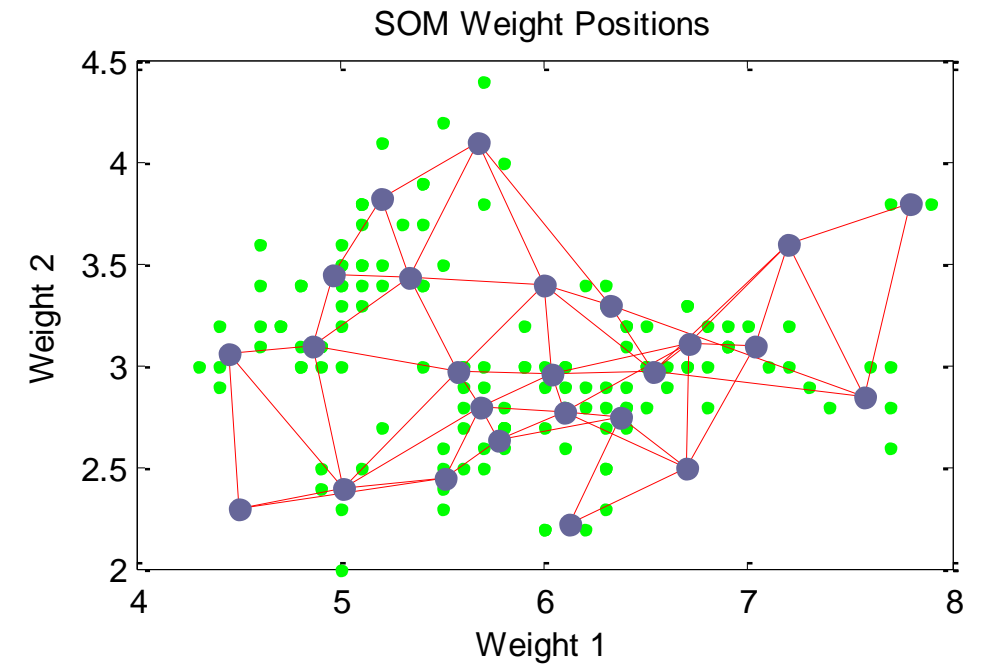
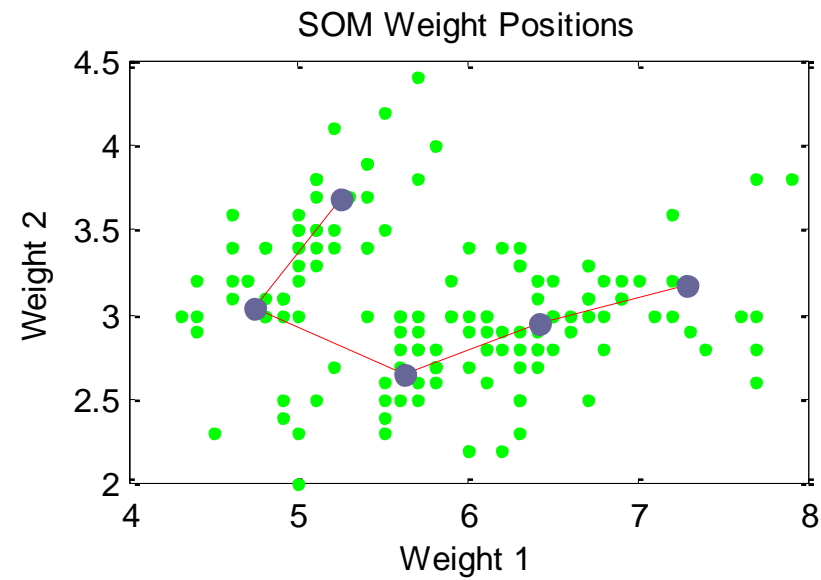
7. Berechnung eines Gütekriteriums (basierend auf dem durchschnittlichen Abstand der letzten Datentupel zum Gewinnerneuron)
  8. Abbruch, wenn Güteanforderung erfüllt, sonst  $k=k+1$  und Fortsetzen mit 3.
- Erweiterungen möglich, z.B. Änderung Nachbarschaft über  $k$  usw.

# Ergebnisse

- Neuronen werden in die Nähe von Datentupeln gezogen
- Gewichte  $\mathbf{w}_{SOM,i}$  werden so bestimmt, dass sie wichtige Bereiche des Eingangsraum abdecken
- 1D- bzw. 2D-Verbindungsstruktur bleibt erhalten:  
"Topologieerhaltende Abbildung"
- projiziert bei Bedarf höherdimensionale auf niedrigdimensionale Eingangsräume
- Ergebnis kann im Bereich der Verbindungsstruktur analysiert werden, z.B. wieviele Datentupel pro Neuron (siehe Beispiel)



# 1D- und 2D-SOMs





- *Problemtypen*
  - Dimensionsreduktion bei höherdimensionalen Merkmalsräumen: z. B. in der Spracherkennung, Klassifizierung von Phonemen (Transformation der durch Fourier-Transformation gewonnen Merkmale auf zweidimensionale Phonemkarte)
  - Approximation und Visualisierung von höherdimensionalen nichtlinearen Zusammenhängen: z. B. Robotersteuerungen
- *Anwendungsfelder*
  - Sprachverarbeitung
  - Bildverarbeitung
  - Robotik/autonome Systeme
  - Telekommunikation

# Beispiel: Spracherkennung

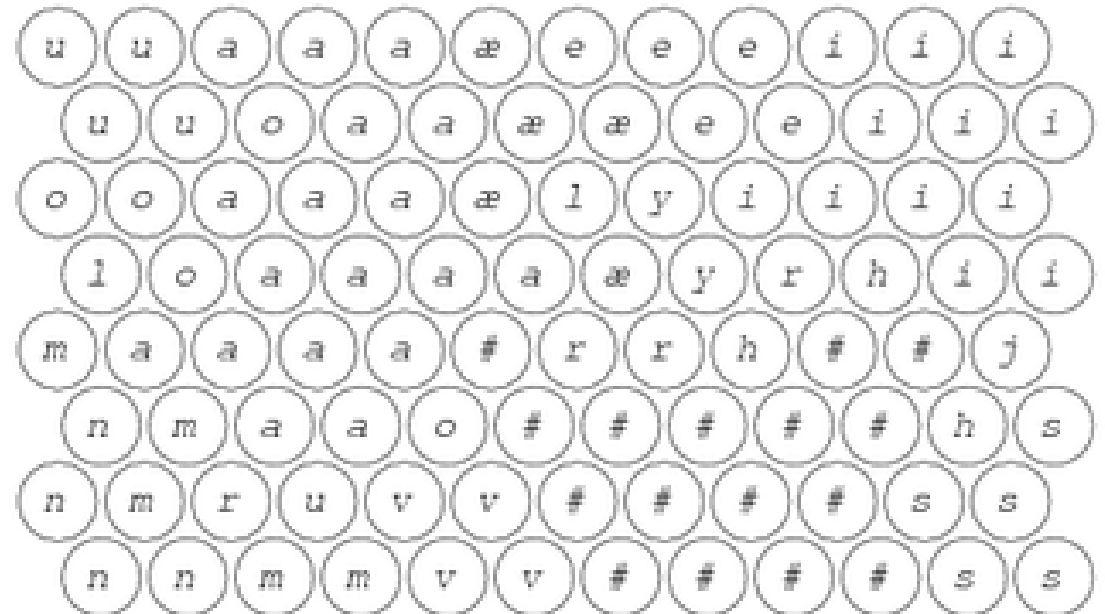
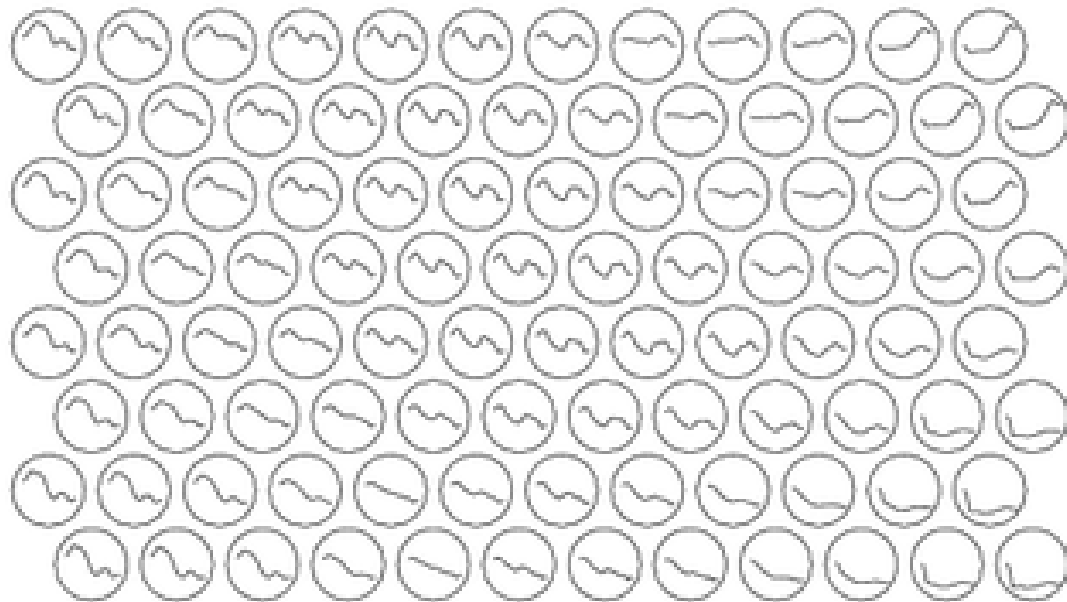
- Finnische Phonemkarte

Bildquelle: [http://www.scholarpedia.org/article/Kohonen\\_network](http://www.scholarpedia.org/article/Kohonen_network)

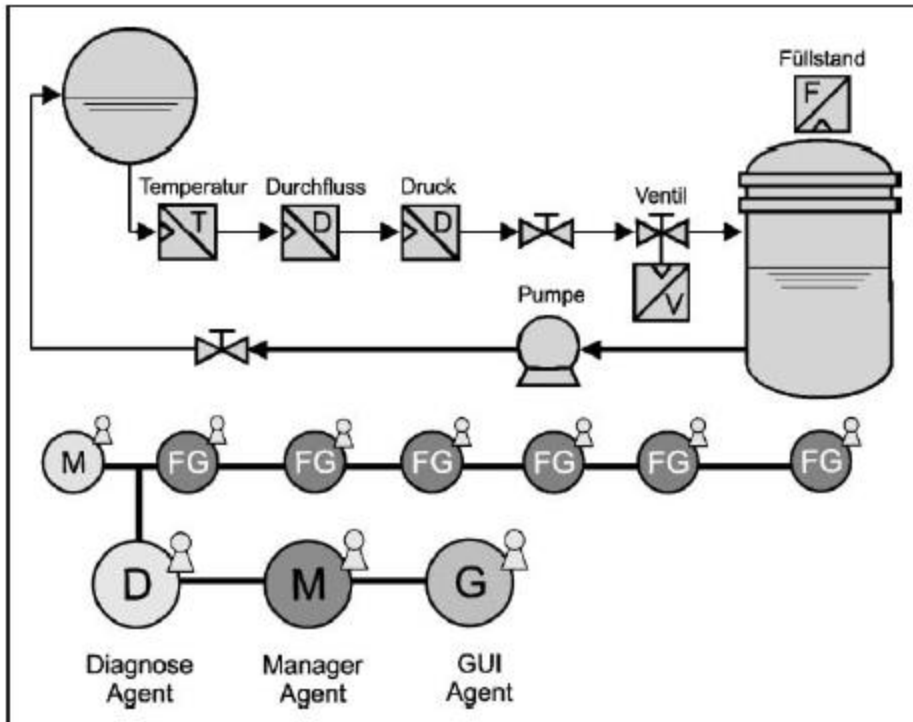
(Teuvo Kohonen)

Links: Spektrum

Rechts: Phoneme (# Stoppkonsonanten k,p,t)



# Beispiel: Visualisierung von Prozessphasen (1)



Quelle:

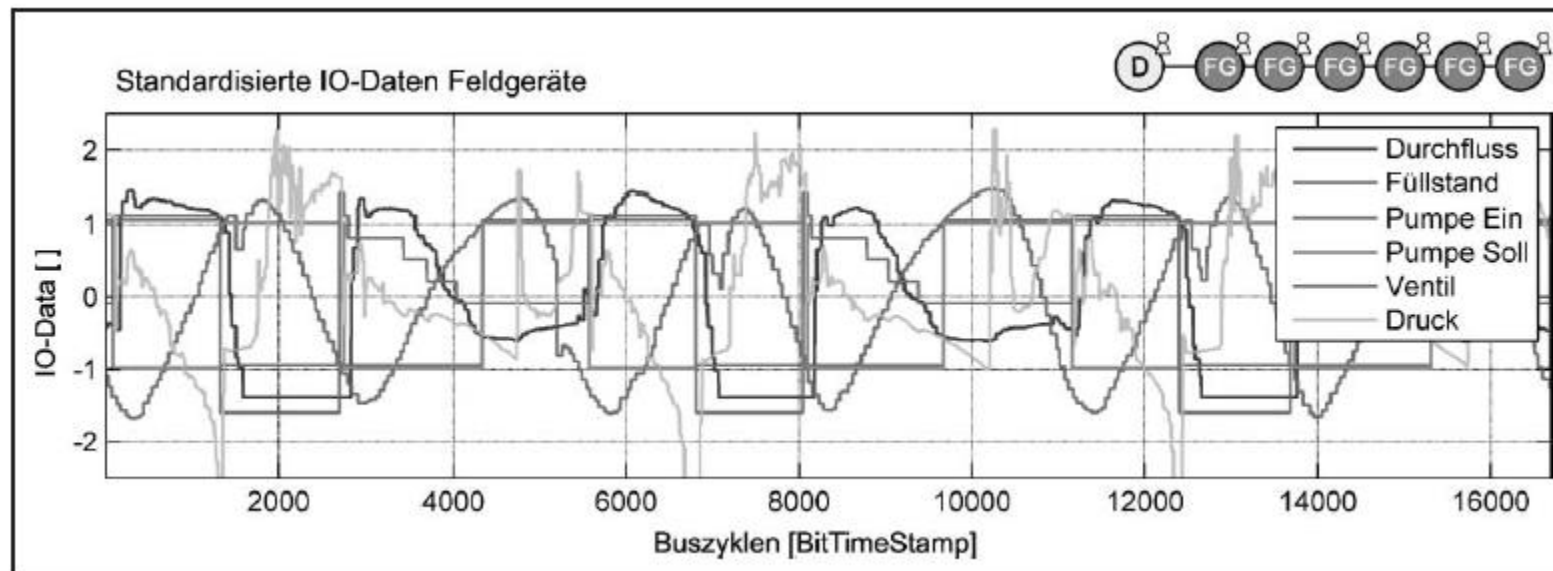
Frey, C.: Prozessdiagnose und Monitoring feldbusbasierter Automatisierungsanlagen mittels selbstorganisierender Karten und Watershed-Transformation. *at-Automatisierungstechnik*, Oldenbourg, **2008**, 56, 374-380

Eingang:

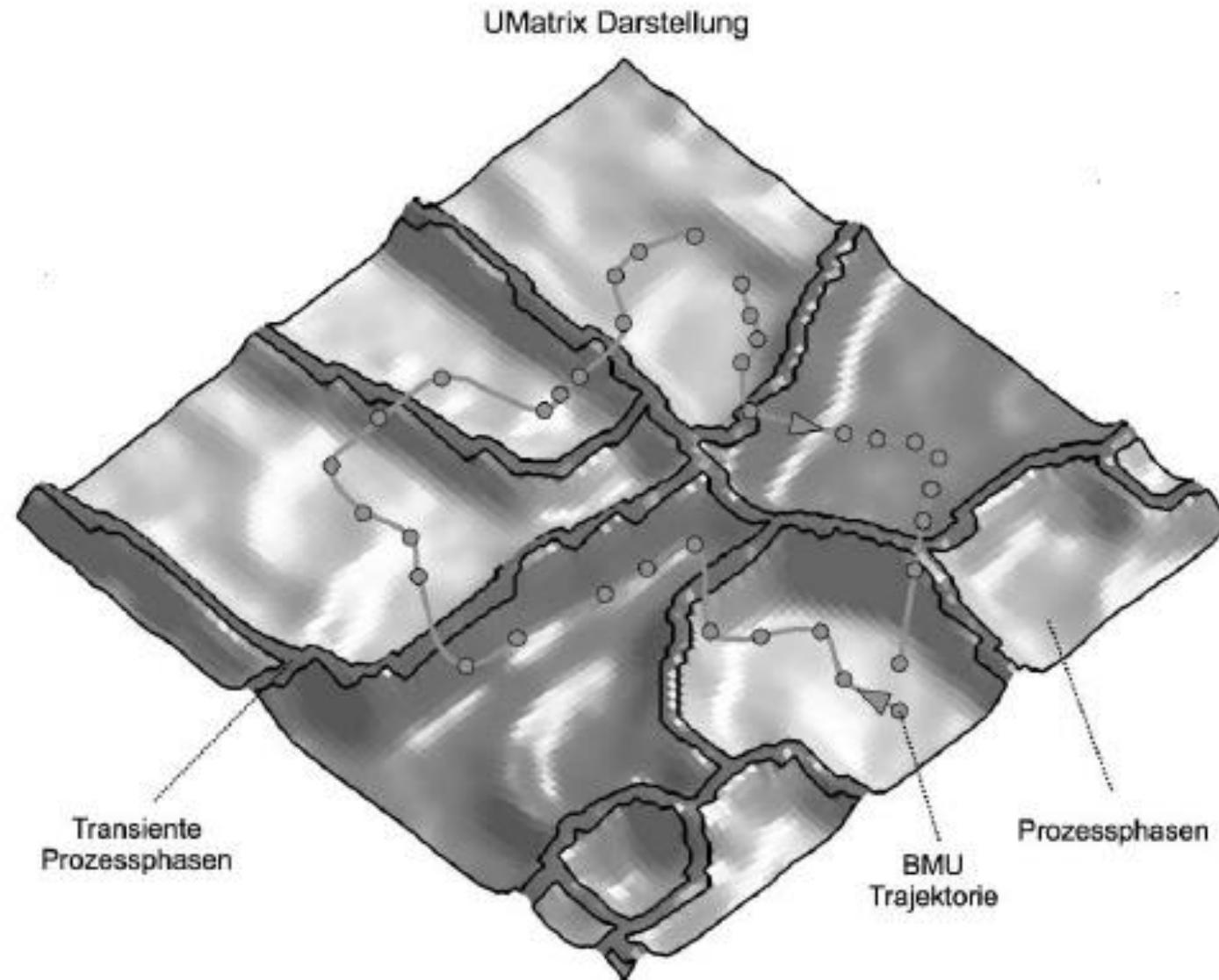
Rohdaten aus einem Prozess

Ausgang:

Visualisierung via SOM

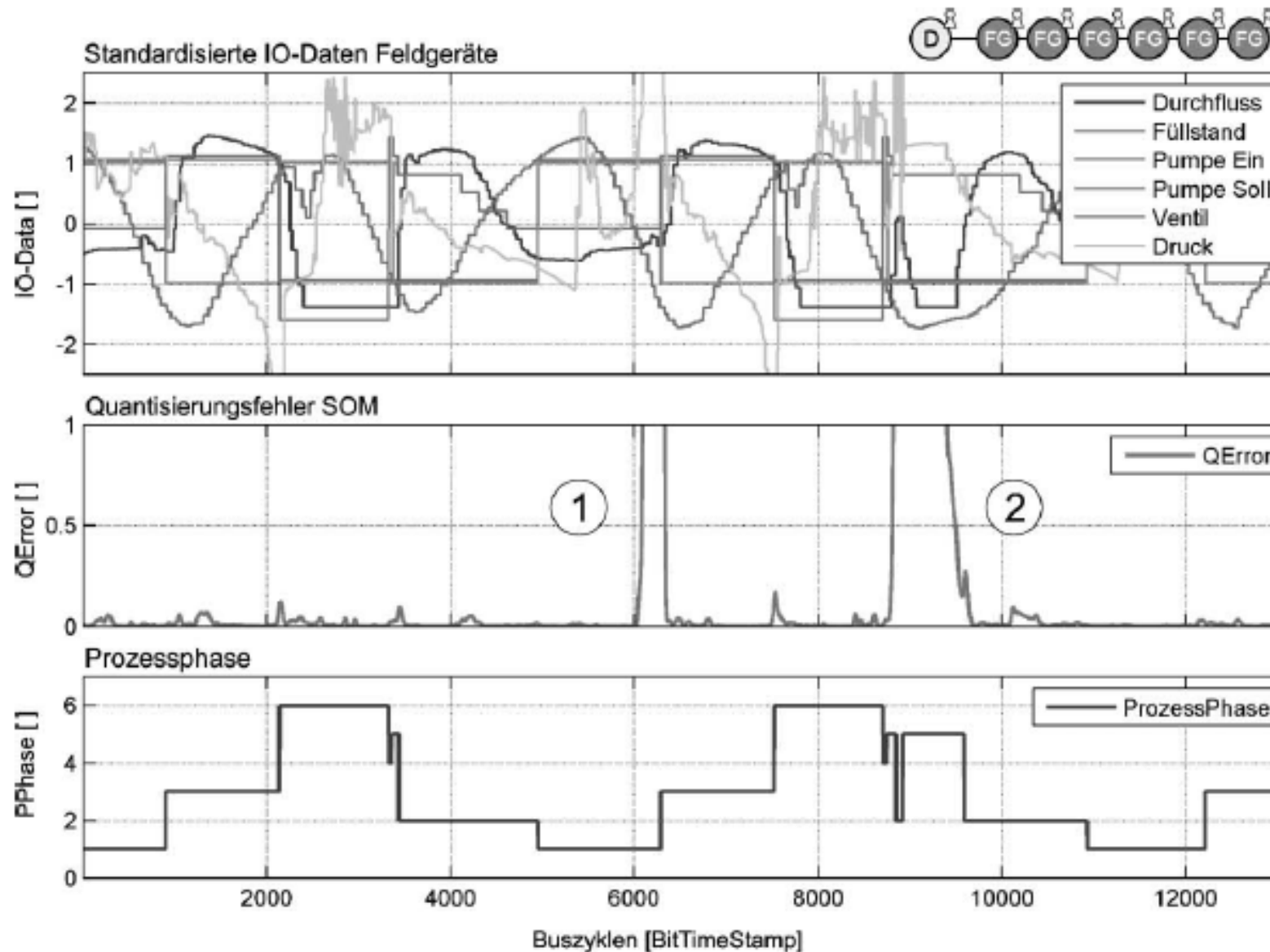


# Beispiel: Visualisierung von Prozessphasen (2)



Frey, C.: Prozessdiagnose und Monitoring feldbusbasierter Automatisierungsanlagen mittels selbstorganisierender Karten und Watershed-Transformation. *at-Automatisierungstechnik*, Oldenbourg, **2008**, 56, 374-380

# Beispiel: Visualisierung von Prozessphasen (3)



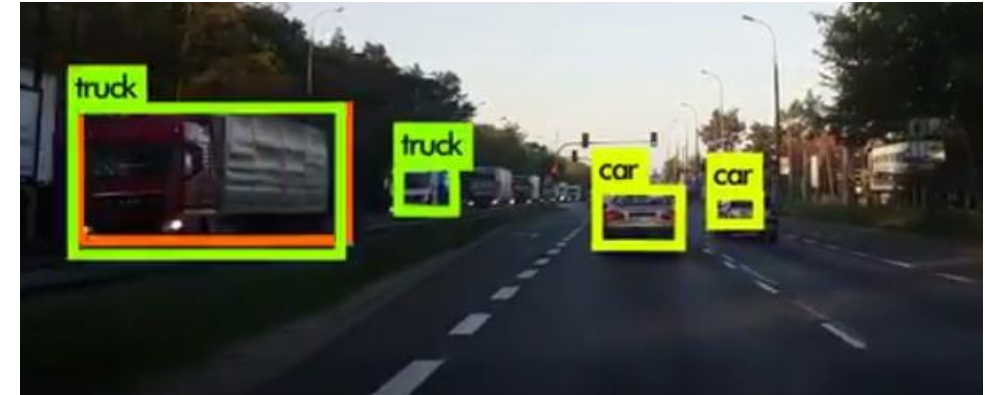
Frey, C.: Prozessdiagnose und Monitoring feldbusbasierter Automatisierungsanlagen mittels selbstorganisierender Karten und Watershed-Transformation. *at-Automatisierungstechnik*, Oldenbourg, **2008**, 56, 374-380

- 3 Künstliche Neuronale Netze
  - 3.1 Vom Biologischen zum Künstlichen Neuronalen Netz
  - 3.2 Struktur
  - 3.3 Lernverfahren (Fortsetzung)
  - 3.4 Multi-Layer-Perceptron-Netze (MLP-Netze)
  - 3.5 Radial-Basis-Funktions-Netze (RBF-Netze)
  - 3.6 Kohonen-Karten
  - 3.7 Deep Learning & Convolutional Neural Networks**
  - 3.8 Kommentare



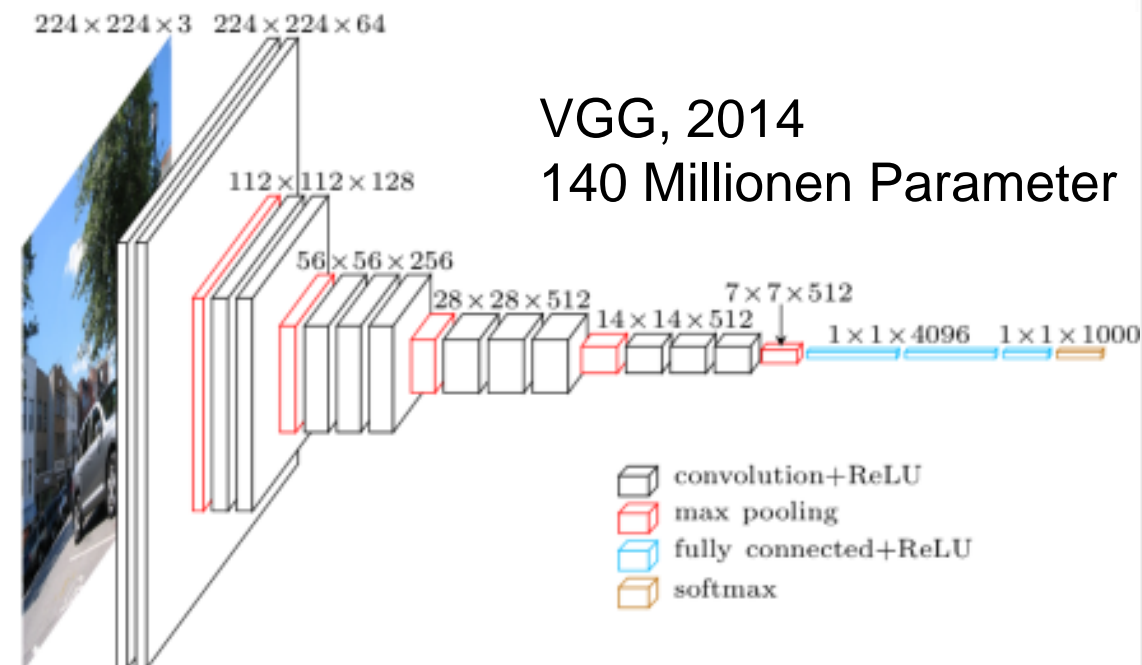
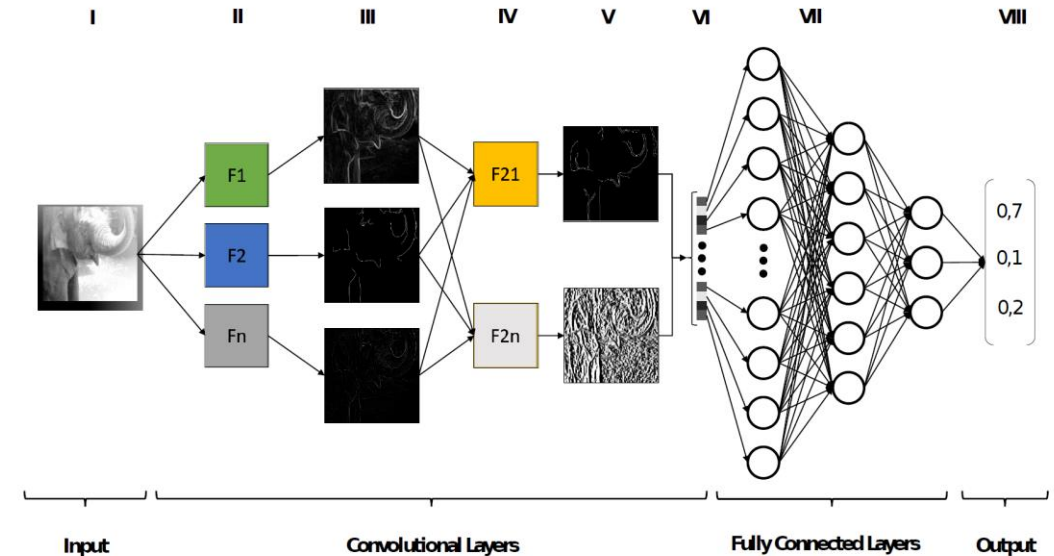
# Deep Learning (Historie)

- Geschichte
  - 2006 Prägung des Begriffs (Deep Belief Networks)
  - 2010: ImageNet Competition (große Bilddatenbank)
  - 2012 AlexNet: Durchbruch neuronaler Strukturen im Maschinellen Sehen
  - 2014: VGG
  - 2015: ResNet, Inception etc.
  - 2016: Alpha Go
- Idee:
  - Vorverarbeitung (z.B. bei Bildern, Zeitreihen, Strukturen) nicht manuell vorgeben, sondern mitlernen
  - Netze mit vielen verdeckten Schichten
  - spezielle Netzstrukturen und Lernstrategien



# Deep Learning Struktur

- oftmals Feedforward-Netze
- auf hochdimensionale Eingangsgrößen (meistens Bilder) angepasst, Pixel sind Eingang des Netzes
- Ausgang je nach Netz: Label für ganzes Bild, lokalisierte Objekte oder pixelweise Segmentierung
- geschickte Verbindung der Layer mit Vorstrukturierung:
  - Faltung (Convolution) mit gekoppelten Parametern und lokaler Zuweisung
  - Downsampling: Max-Pooling zur Reduktion der Auflösung
  - ReLU (weniger beteiligte Neuronen)
  - "Fully Connected Layer" als MLP
  - Wiederholung gleicher Layer



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).



# Deep Learning (Strukturelemente)

- Faltung (Convolution+Stride): Zusammenfassen benachbarter Regionen (Parameter: Wichtungsmatrizen der CNN-Schichten, Stride: Schrittweite der Faltungsmatrix)
- Batch-Normalization: Verbesserung der Parameteroptimierung
- Max-Pooling: Reduktion der Auflösung
- voll-vernetzte Schichten: „Interpretation / Abstraktion“ der erhaltenen Datenströme

Wichtungsmatrix

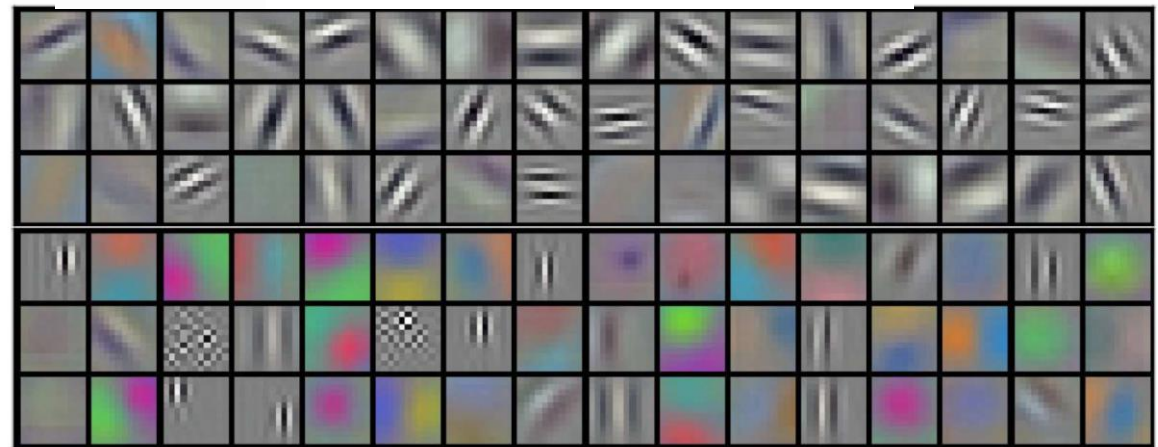
$$\begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix}$$

Eingang der Faltung

3	3 <sub>0</sub>	2 <sub>1</sub>	1 <sub>2</sub>	0
0	0 <sub>2</sub>	1 <sub>2</sub>	3 <sub>0</sub>	1
3	1 <sub>0</sub>	2 <sub>1</sub>	2 <sub>2</sub>	3
2	0	0	2	2
2	0	0	0	1

Ausgang der Schicht

12	12	17
10	17	19
9	6	14



4	6	10	3
9	4	0	1
2	4	9	8
7	1	2	5

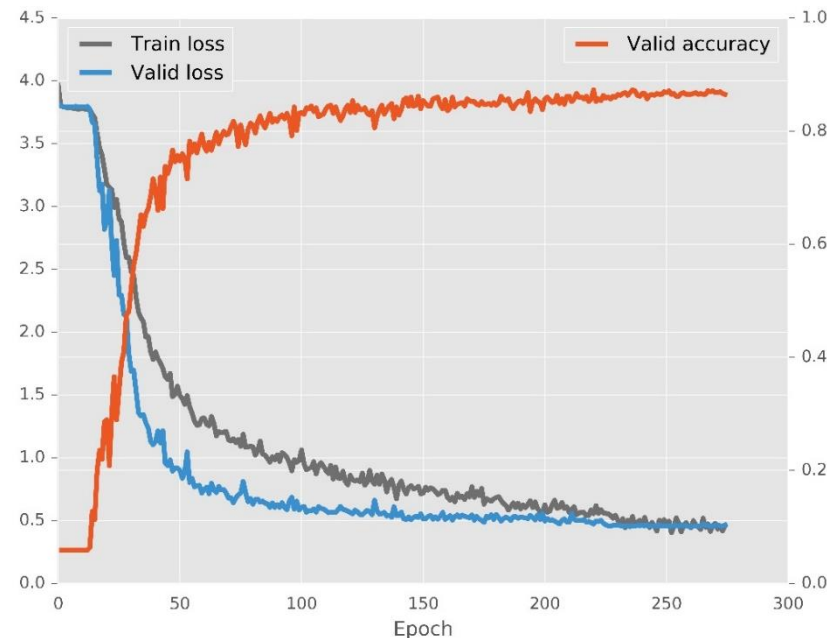
Max-Pooling

9	10
7	9

# Deep Learning (Beispiel)

## Training:

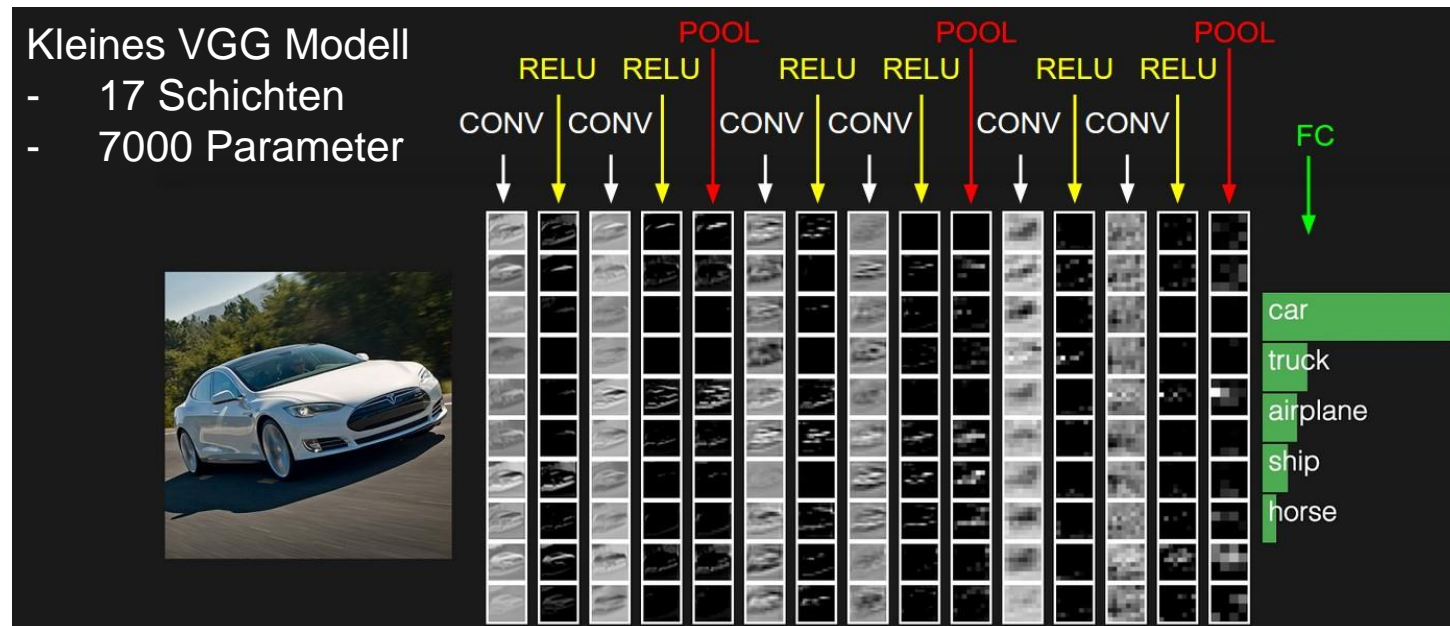
- Backpropagation wird auf mehreren Bildern (Batch) parallel berechnet
- verschiedene Gütemaße: Loss (spezielle Maße, verschiedene Varianten) und Accuracy (1-Klassifikationsfehler)
- ein Durchgang über alle Bilder ist eine Epoche



Typischer Verlauf für ein Training über eine Bildklassifikation mit Backpropagation,  
Quelle: [http://florianmuellerklein.github.io/cnn\\_streetview/](http://florianmuellerklein.github.io/cnn_streetview/)

## Zeitaufwand:

- abhängig von Daten und Architektur
- Beispiel:
  - 10000 Bilder
  - 1 High-End GPU
  - typisch: einige Minuten/Epoche



Quelle: <http://cs231n.github.io/convolutional-networks/>

# Datenaugmentierung

häufige Probleme:

- zu hohe Anzahl an Parametern
- Datensatz zu klein für verlässliche Schätzung

Idee:

- Erzeugen zusätzlicher Bilder beim Training oder beim Testen ("Test-Time-Augmentation")
- Maßnahmen: z.B. Rauschen, Drehen, Rotieren, Scharfzeichnen usw.
- Ziel:
  - Training: kleinerer Klassifikationsfehler der Bilder durch größere Robustheit
  - Testen: Validieren, ob immer das gleiche Ergebnis rauskommt



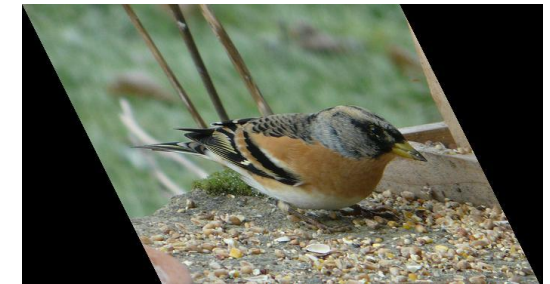
Original



Rauschen



Rotation



Scherung



Grauwertbild



Übersättigung



Weichzeichnung



Scharfzeichnen



# Verfügbare öffentlich zugängliche Datensätze

Bildklassifikation  
(weit verbreitete Benchmarks)

Datensatz	Ground truth
MNIST	<ul style="list-style-type: none"> <li>- Bild-Label</li> <li>- <a href="http://yann.lecun.com/exdb/mnist/">http://yann.lecun.com/exdb/mnist/</a></li> </ul>
CIFAR	<ul style="list-style-type: none"> <li>- Bild-Label</li> <li>- <a href="https://www.cs.toronto.edu/~kriz/cifar.html">https://www.cs.toronto.edu/~kriz/cifar.html</a></li> </ul>
IMAGENET	<ul style="list-style-type: none"> <li>- Bild-Label</li> <li>- Bounding Box</li> <li>- <a href="http://www.image-net.org/">http://www.image-net.org/</a></li> </ul>


größte Herausforderung: Bilder müssen gelabelt sein!

## Autonomes Fahren/Objekterkennung

Dataset	Ground truth
 <p>The KITTI Vision Benchmark Suite A project of Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago</p>	<ul style="list-style-type: none"> <li>- 3D/2D Object detection</li> <li>- 3D Object tracking</li> <li>- SLAM/visual odometry</li> <li>- Optical/scene flow</li> <li>- Depth</li> <li>- Lane detection, road estimation</li> <li>- <a href="http://www.cvlibs.net/datasets/kitti/">http://www.cvlibs.net/datasets/kitti/</a></li> </ul>
 <p>CITYSCAPES DATASET Semantic Understanding of Urban Street Scenes</p>	<ul style="list-style-type: none"> <li>- Object detection</li> <li>- Semantic segmentation</li> <li>- <a href="https://www.cityscapes-dataset.com/">https://www.cityscapes-dataset.com/</a></li> </ul>
 <p>BERKELEY DeepDrive</p>	<ul style="list-style-type: none"> <li>- Object detection</li> <li>- Freespace detection</li> <li>- Lane markings</li> <li>- Instance segmentation</li> <li>- <a href="https://bair.berkeley.edu/blog/2018/05/30/bdd/">https://bair.berkeley.edu/blog/2018/05/30/bdd/</a></li> </ul>
 <p>Mapillary</p>	<ul style="list-style-type: none"> <li>- Object detection</li> <li>- Semantic segmentation</li> <li>- Instance segmentation</li> <li>- <a href="https://www.mapillary.com/">https://www.mapillary.com/</a></li> </ul>
 <p>apolloscope</p>	<ul style="list-style-type: none"> <li>- Semantic segmentation</li> <li>- Object detection</li> <li>- Lane segmentation</li> <li>- Self localization</li> <li>- Car instance segmentation</li> <li>- Depth estimation</li> <li>- <a href="http://apolloscope.auto/">http://apolloscope.auto/</a></li> </ul>
 <p>COCO Common Objects in Context</p>	<ul style="list-style-type: none"> <li>- Object detection</li> <li>- Object recognition</li> <li>- Localization</li> <li>- Semantic segmentation</li> <li>- <a href="http://cocodataset.org/">http://cocodataset.org/</a></li> </ul>

# Deep Learning Software inkl. GPU-Unterstützung

- Python-Bibliotheken
  - integrierte GPU/TPU Unterstützung

- Tensorflow
  - Große Community
  - Gute Unterstützung für Hardware

- Keras Keras
  - High-level Bibliothek für Tensorflow und Microsoft Cognitive Toolkit
  - sehr kompakte Schreibweise

- PyTorch
  - Fokus auf wissenschaftliche Anwendungen

- Weitere Umgebungen: Microsoft Cognitive Toolkit , Caffe, Deeplearning4j, ...

## Keras

Model  
Architektur

```
model = Sequential()  
model.add(Conv2D(32, kernel_size=(3, 3),  
                activation='relu',  
                input_shape=input_shape))  
model.add(Conv2D(64, (3, 3), activation='relu'))  
model.add(MaxPooling2D(pool_size=(2, 2)))  
model.add(Dropout(0.25))  
model.add(Flatten())  
model.add(Dense(128, activation='relu'))  
model.add(Dropout(0.5))  
model.add(Dense(num_classes, activation='softmax'))
```

Erstellung des  
Rechengraphs

```
model.compile(loss=keras.losses.categorical_crossentropy,  
             optimizer=keras.optimizers.Adadelta(),  
             metrics=['accuracy'])
```

Model Training

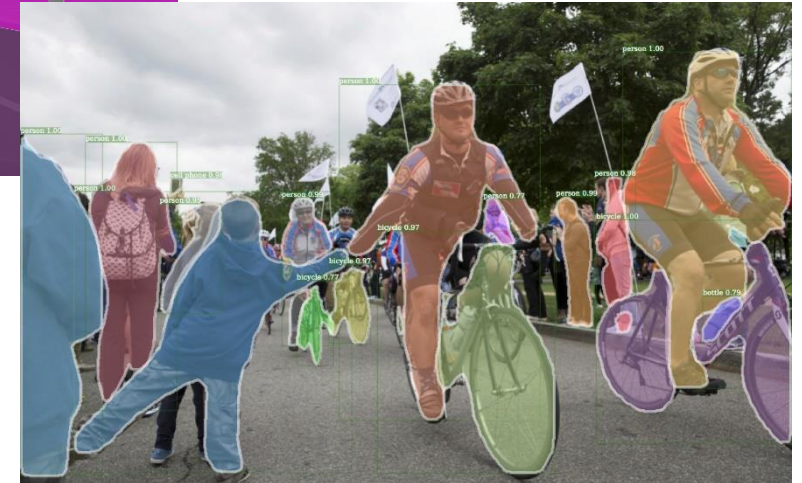
```
model.fit(x_train, y_train,  
        batch_size=batch_size,  
        epochs=epochs,  
        verbose=1,  
        validation_data=(x_test, y_test))  
score = model.evaluate(x_test, y_test, verbose=0)
```

## Anwendung:

- Segmentierung
  - Objekterkennung
  - Autonomes Fahren
  - Beispiel Architekturen:
    - ResNeXt
    - Faster R-CNN
    - Mask R-CNN
- Gute Verfügbarkeit von Daten



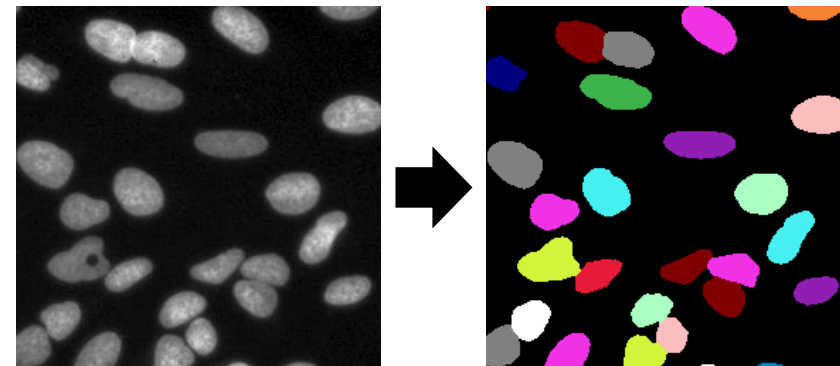
Quelle: Cityscapes,  
<https://www.cityscapes-dataset.com/>



Quelle: Facebook, Detectron,  
<https://github.com/facebookresearch/Detectron>

## Anwendungen:

- Zellsegmentierung
  - Gewebeerkennung
  - Beispiel Architekturen:
    - U-Net
    - Segnet
- Umfangreiche und gute Datensets  
schlecht verfügbar (auch immer für eigene  
Probleme!)



# GAN: Generative Adversarial Networks (1)

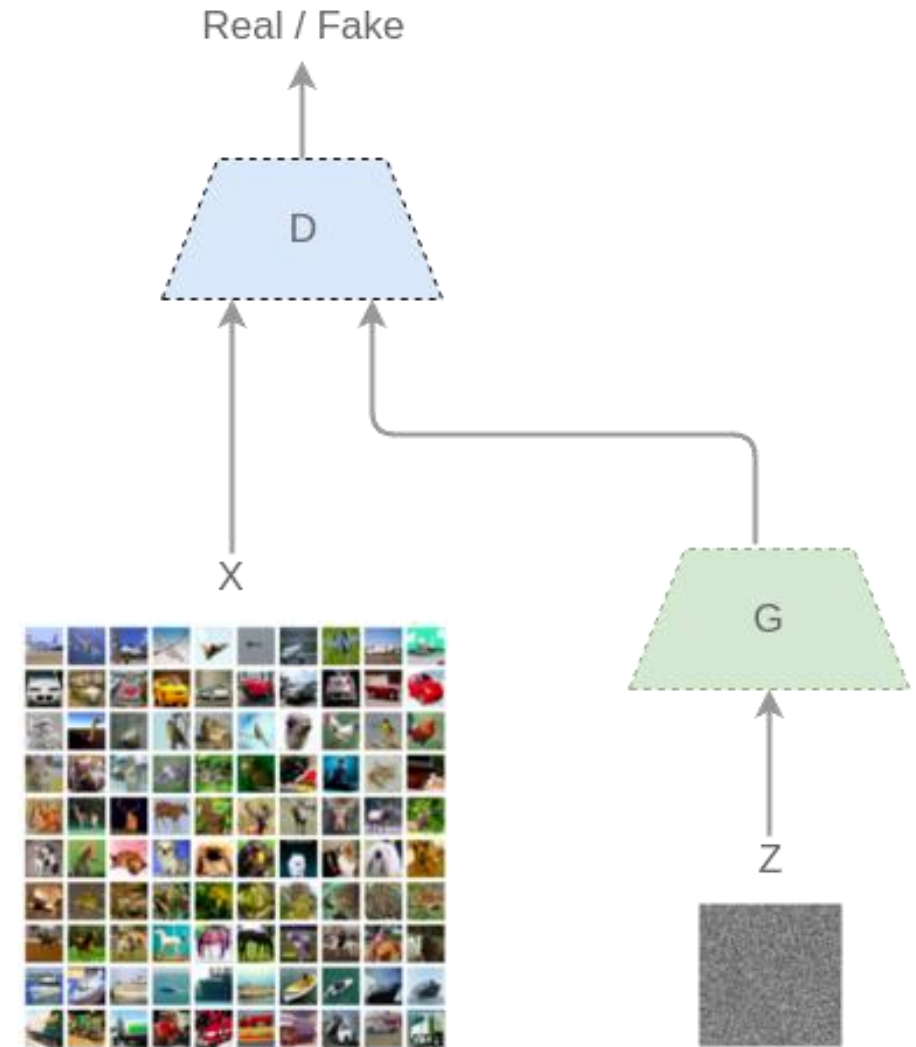
Ziel: Synthetisierung von Bildern, die von realen Bildern nicht unterscheidbar sind

Modell besteht aus 2 Netzen:

- G: Generator, der Bilder auf Basis von zufälligem Rauschen erstellt
- D: Diskriminator, binäre Klassifikation zwischen echten und generierten Bildern

Training:

- D: Binäre Klassifikation zwischen echten und von G generierten Bildern
- G: Anpassen der Gewichte, so dass der Diskriminator getäuscht wird
- abwechselndes Trainieren von G und D
- da die beiden Modelle gegensätzliche Ziele haben, spricht man von adversarial network.
- im Optimalfall werden beide Netze kontinuierlich besser und die Qualität der erzeugten Bilder steigt.





# GAN: Generative Adversarial Networks (2)

## Anwendung

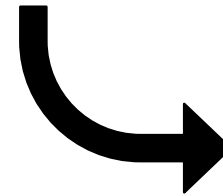
- Style-Transfer
- Erstellung von natürlichen Bildern
- Super-Resolution  
(Erhöhung der Auflösung von Bildern )
- Label2Image (<https://phillipi.github.io/pix2pix/>)

## Beispiel Architekturen (Conditional GAN):

- Vid2Vid  
<https://github.com/NVIDIA/vid2vid>
- CycleGAN  
<https://github.com/junyanz/CycleGAN>



Nacht



Style-Transfer auf Tag



Photograph



Monet



Van Gogh



Cezanne



Ukiyo-e

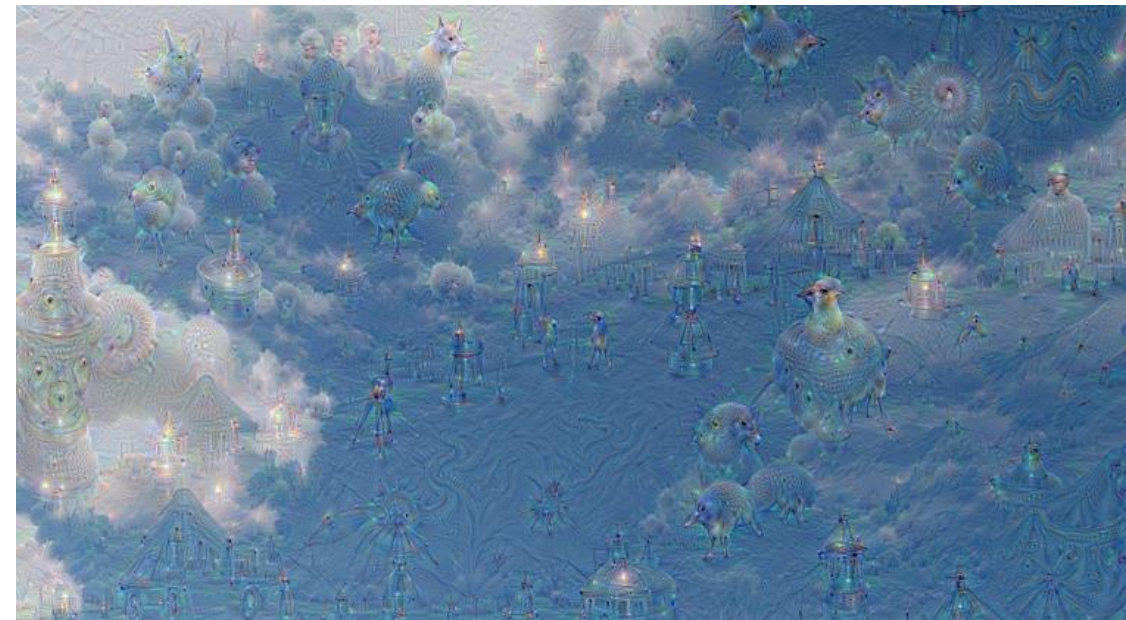
Style-Transfer: Quelle:Jun-Yan Zhu et al. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", ICCV, 2017



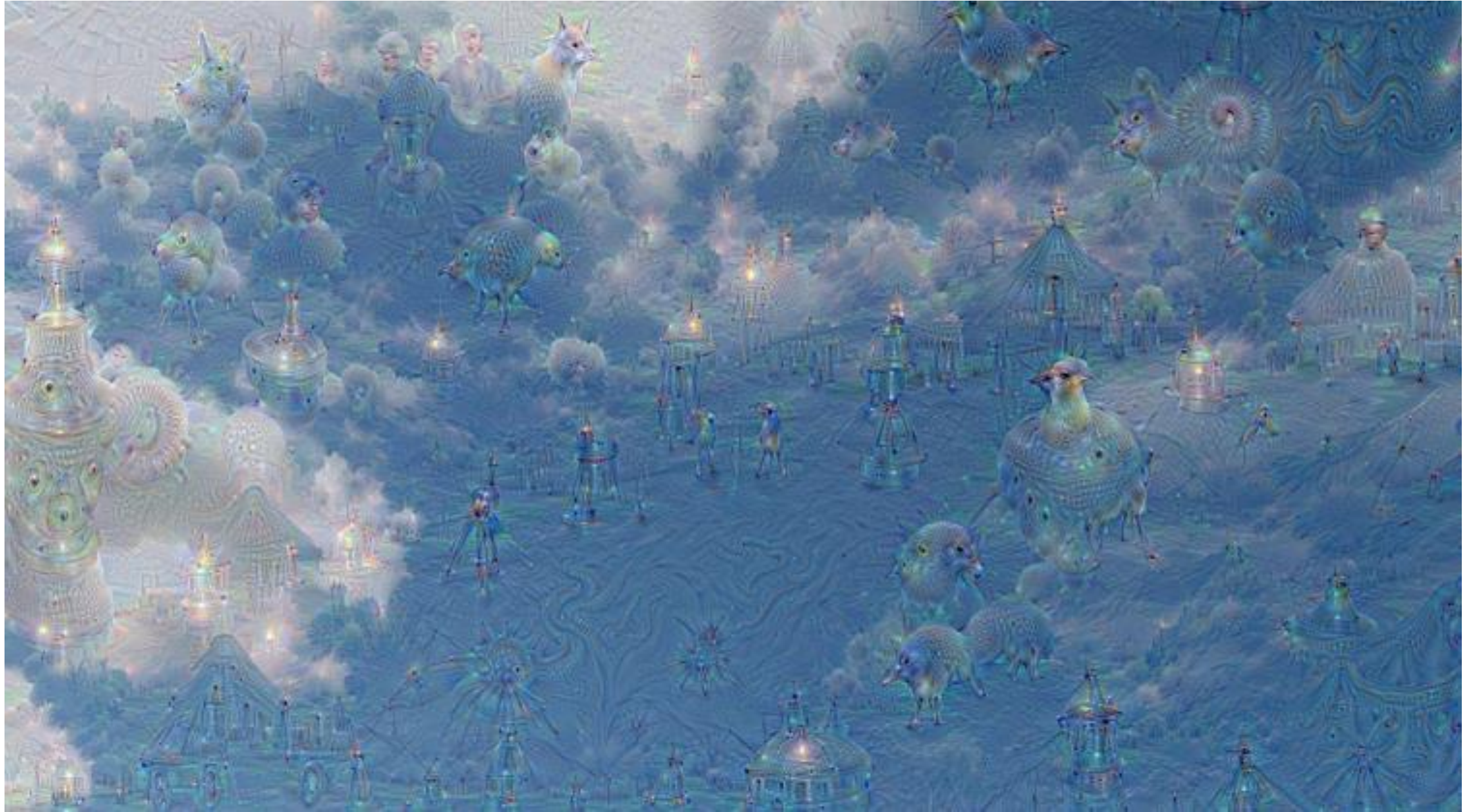
# Überinterpretation & Inceptionism

Ähnlich wie Style-Transfer  
("Deep Dream"):

- Eingang: Wolkenbild (oben)
- Netz: Trainiert auf Tieren
- Ausgang: Wolkenbild mit Tiermuster
- Lernstrategie: Modifizieren des EINGANGSBILDES so, dass es dem gewünschten Ausgang eines Netzes (vorletzte Schicht) nahe kommt
- Quelle:  
<https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>







Quelle: <https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>

# Was hat sich geändert, was vorher nicht ging?

- geschickte Verbindung der Layer mit Vorstrukturierung
  - Nutzung bekannter Vorverarbeitungsstrategien (z.B. Faltung)
  - Erzwingen von niederdimensionalen Repräsentationen (z.B. Autoencoder)
- Reduzieren der zu lernenden Parameter:
  - feste Kopplung von Parametern (z.B. gleiche Faltung für lokale Bildbereiche)
  - Downsampling (Verringerung der Auflösung)
  - Lernen nur eines Teils der Parameter (z.B. Transfer Learning)
- spezielle Lernstrategien, um mit vorhandenen Daten auszukommen:
  - Datenaugmentierung
  - Reinforcement Learning & Verkopplung von Ein- und Ausgangsgrößen mehrerer Netze (z.B. in Generative Adversarial Networks)
  - Arbeiten zur Konvergenz der Aktivierungsfunktion
- große gelabelte Datensätze für Training  
(mit anderen Daten als in der eigentlichen Aufgabe!)
- spezielle Deep Learning Software inkl. GPU-Unterstützung

- 3 Künstliche Neuronale Netze
  - 3.1 Vom Biologischen zum Künstlichen Neuronalen Netz
  - 3.2 Struktur
  - 3.3 Lernverfahren (Fortsetzung)
  - 3.4 Multi-Layer-Perceptron-Netze (MLP-Netze)
  - 3.5 Radial-Basis-Funktions-Netze (RBF-Netze)
  - 3.6 Kohonen-Karten
  - 3.7 Deep Learning & Convolutional Neural Networks
  - 3.8 Kommentare**

- Künstliche Neuronale Netze sind wegen ihrer Eigenschaft als "universelle Approximatoren" populär
- dann besonders sinnvoll, wenn strukturelle Zusammenhänge unbekannt sind
- Künstliche Neuronale Netze sind trotzdem eher kompliziert, deshalb unbedingt mit einfacheren Methoden (Polynomregression usw.) vergleichen
- MLP, RBF und SOM sind die Standardtypen für Künstliche Neuronale Netze
- Neuere Trends existieren, z.B. Deep Learning
- Rekurrente Netze stark umstritten
  - Pro: Abbildung dynamischer Zusammenhänge
  - Kontra: Konvergenzprobleme



- Software  
"Comparison of Neural Network Simulators" (U Colorado)  
[https://grey.colorado.edu/emergent/index.php/Comparison\\_of\\_Neural\\_Network\\_Simulators](https://grey.colorado.edu/emergent/index.php/Comparison_of_Neural_Network_Simulators)
- MATLAB:
  - Neuronale Netze Toolbox inkl. MLP, RBF, SOM usw., Hilfe mit  
`>> help nnet`
  - SciXMiner Implementierungen
    - MLP, RBF, SOM:  
(frei verfügbar unter <http://sourceforge.net/projects/scixminer>)
    - Projekte im ILIAS:  
1D, 2D Regression (künstliche Testdatensätze)
    - Projekte in SciXMiner:  
Building-Datensatz für Energieverbrauch in einem Gebäude

- Zeit und Ort:
  - Selbststudium (mit Hinweisen in der Vorlesung)
  - Beratungstermine im SCC-Pool Geb. 20.21, Termin wird noch per ILIAS bekannt gegeben (Pools G&H, 3 Termine), ein Termin pro Studierendem buchbar
- Ziele:
  - Kennenlernen MATLAB-Toolbox SciXMiner
  - Anlernen von MLPs im ein- und zweidimensionalen Fall
  - Zusatzaufgabe: Regression des Energieverbrauchs von Gebäuden

## ACHTUNG:

- im ILIAS: uebung\_ci.zip
  - Aufgabenstellung inkl. Installationshinweise SciXMiner
  - Beispieldatensätze und Makros
- bei Variante mit Laptop: vor der Übung: S. 2 der Übungsanleitung (Installation auf eigenem Laptop) abarbeiten (sonst WLAN-Probleme)

# Bedienung SciXMiner

- Funktionen über Menüs zugänglich (Callbacks...)
- Parameter über Oberfläche einstellbar (Callbacks...)
- Ergebnisse in Bilder und Dateien
- Englisch
- Automatisierung von Abläufen durch Makros
- Einbindung neuer Funktionen:
  - Makro
  - Plugin Merkmalsextraktion
  - Menüpunkt usw.
- Erweiterungen (Peptide, Bildverarbeitung, Datenqualität, Asphaltbilder usw.)
- Installationsdatei
- Handbuch (195 Seiten)

