Vorlesung 6: Convolutional Neural Networks - Grundlagen

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

Prof. Dr. Reinhard Klein

Nils Wandel

Informatik II, Universität Bonn

05.06.2024

Wiederholung

Letzte Woche

1. Parameter-Initialisierung
2. Regularisierungs-Terme in Loss-Funktionen (L_2 / L_1)
3. Early Stopping
4. Weight-Sharing
5. Data-Augmentation
6. Bagging
7. Dropout

Organisation

Today's Lecture

1. Transformationen als Gruppenoperation
2. Äquivarianz und Invarianz
3. Konvolution und Kreuzkorrelation
4. Convolutional Layers
5. Batch-Normalisierung
6. CNNs

Part 1

Grundlagen Konvolutionen

Gruppenoperationen

Gruppe

Sei G eine Menge und $\cdot : G \times G \to G$ eine 2-stellige Abbildung. Dann ist $(G, \cdot)$ eine Gruppe, wenn:

1. Es existiert ein neutrales Element $e \in G$, sodass $\forall a \in G : e \cdot a = a \cdot e = a$
2. Jedes Element $a \in G$ besitzt ein inverses Element $a^{-1} \in G$, sodass: $a \cdot a^{-1} = a^{-1} \cdot a = e$
3. Assoziativität gilt: $\forall a, b, c \in G : (a \cdot b) \cdot c = a \cdot (b \cdot c)$

Gruppenoperation

Sei $(G, \cdot)$ eine Gruppe und X eine Menge, dann ist $\triangleright : G \times X \to X$ eine Gruppen(-links-)operation, wenn gilt:

1. Identität: $\forall x \in X : e \triangleright x = x$, wobei e das neutrale Element von G ist
2. Verträglichkeit: $\forall a, b \in G, x \in X : (a \cdot b) \triangleright x = a \triangleright (b \triangleright x)$

Beispiele: Gruppenoperationen

Mit Hilfe von Gruppenoperationen lassen sich Transformationen elegant beschreiben:
Sei z.B. $X = [0,1]^{n \times n}$ ein Bildraum für Bilder der Größe $n \times n$.

1. $G = \{Id, Ref_y\}$ bildet eine Gruppe, welche das neutrale Element (Id) und eine Reflektion an der y-Achse (Ref_y) enthält. D.h. $Id \triangleright x = x$ und $Ref_y \triangleright x$ entspricht dem gespiegelten Bild von x.
2. $G = \{Id, Rot_{90}, Rot_{180}, Rot_{270}\}$ bildet eine Gruppe, welche sämtliche 90°-Rotationen des Bildes enthält. Was wären hier die inversen Elemente von Rot_{90},Rot_{180} und Rot_{270}?
3. $G = \{Rot_\theta : \theta \in \mathbb{R}\}$ bildet eine Gruppe, welche sämtliche Rotationen enthält. Welche Probleme könnten hier für einen diskreten und durch n begrenzten Bildraum auftreten?
4. $G = \{T(x,y) : x, y \in \mathbb{R}\}$ bildet eine Gruppe, welche sämtliche Translationen um x, y enthält. Welche Probleme könnten hier für einen diskreten und durch n begrenzten Bildraum auftreten?
5. $G = S(M)$ ist die Gruppe aller Permutationen einer Menge M. Diese Gruppe spielt eine wichtige Rolle z.B. auf Punktwolken-Daten.

Äquivarianz und Invarianz

Sei $f : X \to Y$ eine Abbildung (zum Beispiel ein neuronales Netz), $(G, \cdot)$ eine Gruppe und $\triangleright : G \times X \to X$ sowie $\square : G \times Y \to Y$ Gruppenoperationen auf X und Y.
f heisst *äquivariant* bezüglich G, wenn gilt:

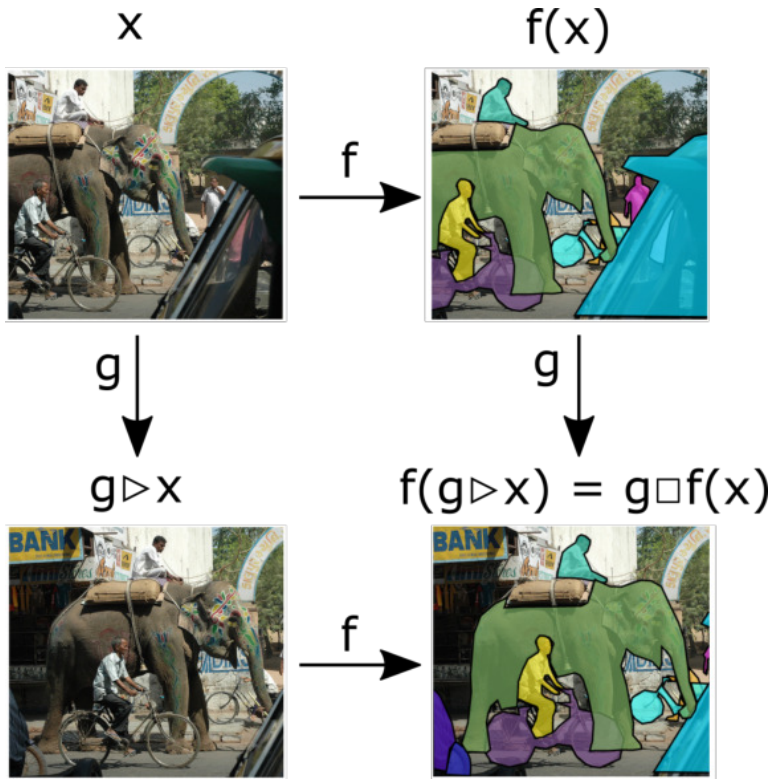
$$\forall g \in G, x \in X : f(g \triangleright x) = g \square f(x)$$

f heisst *invariant* bezüglich G, wenn gilt:

$$\forall g \in G, x \in X : f(g \triangleright x) = f(x)$$

Beispiel: Äquivarianz

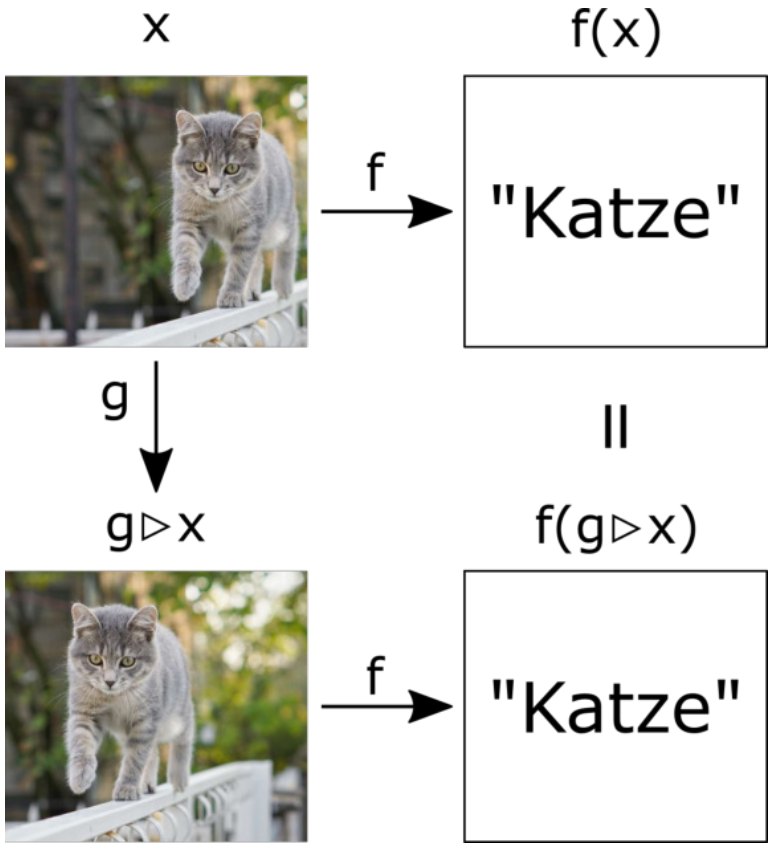
Sei $f : X \to Y$ ein neuronales Netz, welches Bilder X auf Bild-Segmentierungen Y abbildet und $(G, \cdot)$ eine Gruppe für Translationen. Dann möchten wir, dass sich die Bildsegmentierungen bei Translationen des Eingabebildes gleichsam transformieren:



... hier soll f *äquivariant* bezüglich Translationen sein.

Beispiel: Invarianz

Sei nun $f : X \to Y$ ein neuronales Netz, welches Bilder X auf Klassen-labels Y abbildet. Dann möchten wir, dass sich die Klassifizierung bei Translationen des Eingabebildes nicht ändert:



... hier soll f *invariant* bezüglich Translationen sein.
Frage: Für welche Fälle / Transformationen könnte Invarianz problematisch sein?

Was ist eine Konvolution / Kreuzkorrelation?

Eine *Konvolution (oder Faltung)* einer Funktion $x(t)$ mit einer Funktion $y(t)$ ist definiert als:

$$x * y(t) = \int_{\mathbb{R}^n} x(\tau)y(t-\tau)d\tau = \int_{\mathbb{R}^n} x(t-\tau)y(\tau)d\tau$$

Eine *Kreuzkorrelation* einer Funktion $x(t)$ mit einer Funktion $y(t)$ ist definiert als:

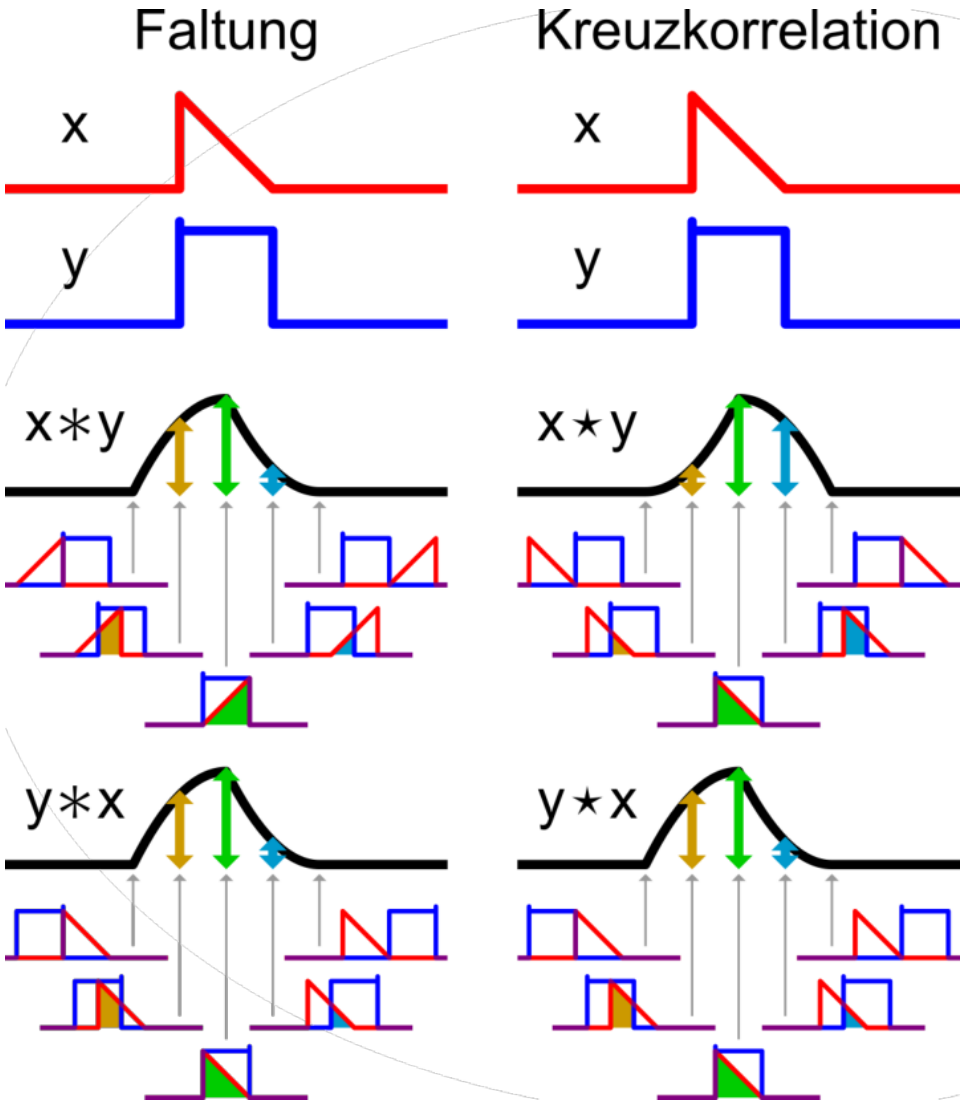
$$x \star y(t) = \int_{\mathbb{R}^n} x(\tau)y(t+\tau)d\tau = \int_{\mathbb{R}^n} x(\tau-t)y(\tau)d\tau = (x(-\tau) * y(\tau))(t)$$

Anschaulich kann man sich vorstellen, dass $y(t)$ ein Signal ist, welches wir durch einen Kernel (oder Filter) $x(t)$ "abtasten". Dazu verschieben wir den Kernel x entlang des Signals y um t und berechnen dann das Integral über das Produkt aus den beiden Funktionen. Der Unterschied zwischen Faltung und Kreuzkorrelation ist, dass bei der Faltung der Kernel am Ursprung gespiegelt betrachtet wird.

(Anmerkung: In Convolutional Neural Networks werden üblicherweise Kreuzkorrelationen verwendet)

Was ist eine Konvolution / Kreuzkorrelation?

Veranschaulichendes Beispiel in 1D:



Warum Konvolution / Kreuzkorrelation?

Wichtige Eigenschaften der Konvolution / Faltung:

- Kommutativ: $x * y = y * x$
- Assoziativ: $x * (y * z) = (x * y) * z$
- Distributiv: $x * (y + z) = x * y + x * z$
- Faltungstheorem: $\mathcal{F}(x * y) = (2\pi)^{n/2}\mathcal{F}(x) \cdot \mathcal{F}(y)$, wobei

$$\mathcal{F}(y)(f) = (2\pi)^{-n/2} \int_{\mathbb{R}^n} y(t)e^{-ift}dt$$

  die Fourier-Transformation ist.
- Äquivariant bezüglich Translationen von x und y

Wichtige Eigenschaften der Kreuzkorrelation:

- Distributiv: $x \star (y + z) = x \star y + x \star z$
- Äquivariant bezüglich Translationen von y

Was ist eine diskrete Konvolution / Kreuzkorrelation?

Im Bereich von Visual Computing werden Kernels x und Signale y oft auf diskreten Pixel-Rastern ($D \subset \mathbb{Z}^n$) gespeichert. In diesem Fall kann man die diskrete Konvolution bzw Kreuzkorrelation analog zum kontinuierlichen Fall einführen.

Die *diskrete Konvolution* ist definiert als:

$$x * y(n) = \sum_{k \in D} x(k)y(n-k) = \sum_{k \in D} x(n-k)y(k)$$

Die *diskrete Kreuzkorrelation* ist definiert als:

$$x \star y(n) = \sum_{k \in D} x(k)y(n+k) = \sum_{k \in D} x(k-n)y(k) = (x(-k) * y(k))(n)$$

Was ist eine diskrete Kreuzkorrelation?

Die diskrete Kreuzkorrelation bildet die Basis für konvolutionale neuronale Netze. Anbei eine Visualisierung für die diskrete Kreuzkorrelation in 2D:

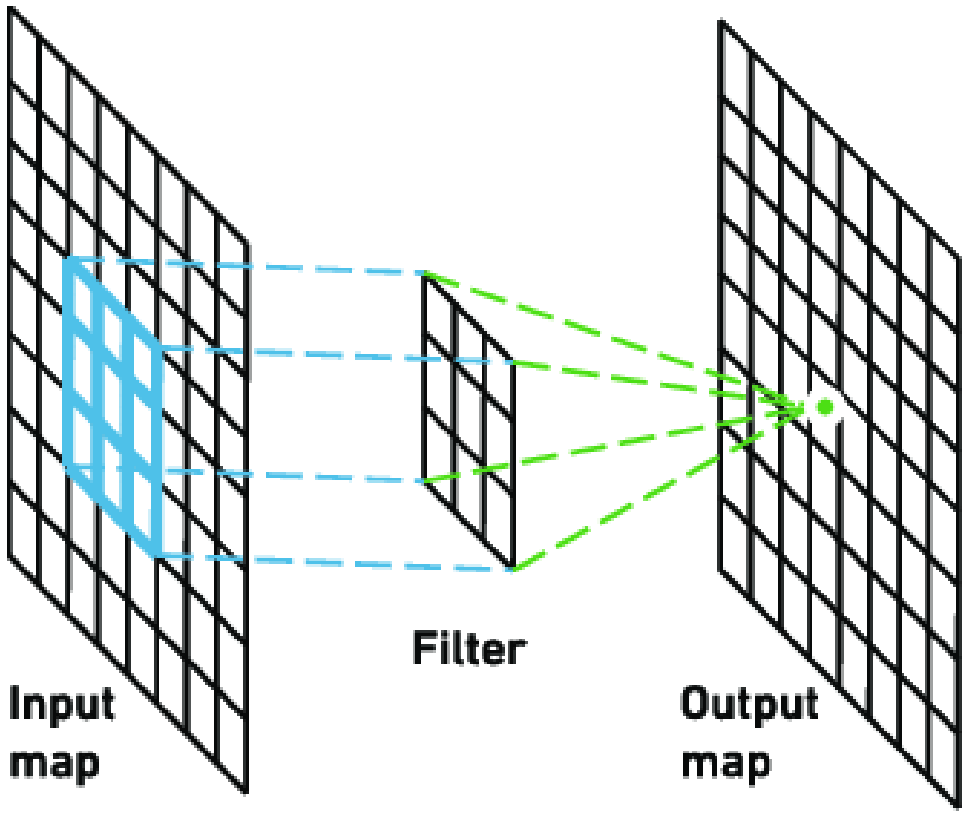


Figure: Diskrete Kreuzkorrelation in 2D. Indem der Filter (x) verschoben und mit der Inputmap (y) mulitpliziert wird, kann die Inputmap "abgetastet" werden. Die Resultate werden in der Outputmap ($s = x \star y$) gespeichert.

Part 2

Convolutional Layers

Convolutional Layer in 1D

Ein convolutional Layer berechnet die Aktivierungen s als diskrete Kreuzkorrelation $s = x \star y$ eines Kernels x mit dem Eingangssignal y:

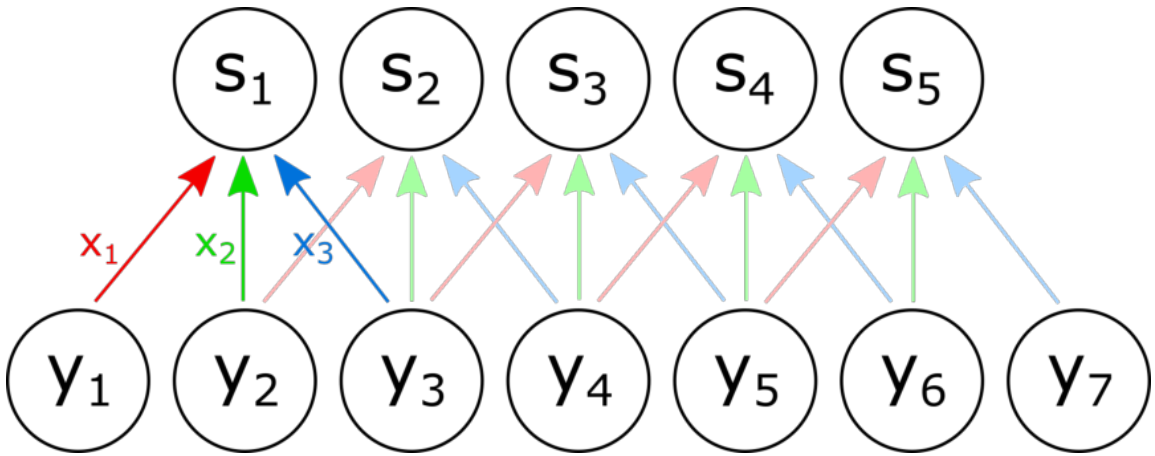
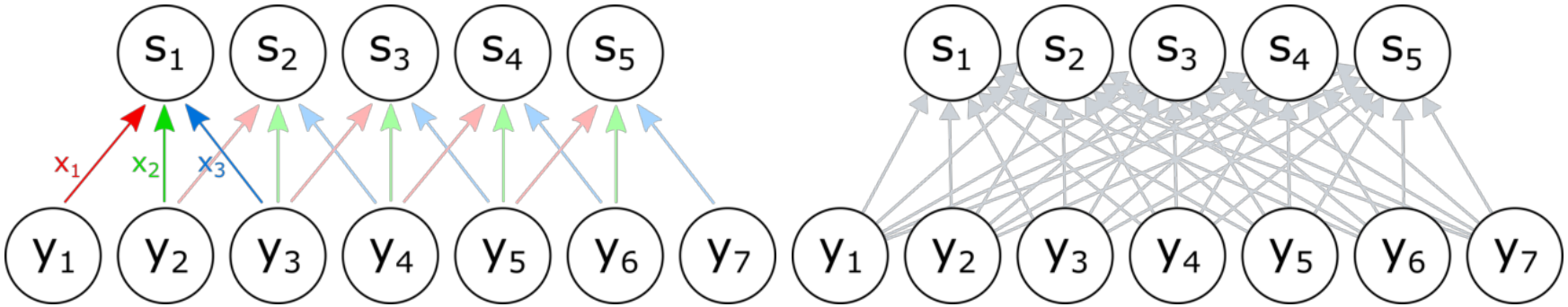


Figure: 1D Convolutional Layer. $s = x \star y$ wird aus der Kreuzkorrelation von x mit y berechnet. Hierbei entspricht s den Aktivierungen der Units des convolutional Layers, x entspricht den Gewichten der Units und y entspricht den Eingabesignalen. Beachte, dass die Gewichte zwischen den Units identisch sind ("weight sharing").

Nach der Berechnung der Kreuzkorrelation, wird üblicherweise eine nichtlineare Aktivierungsfunktion auf die Aktivierungen angewendet (z.B: ReLU).

Convolutional Layer versus fully connected Layer



Ein convolutional Layer (links) hat verglichen mit einem fully connected Layer (rechts) weniger Verbindungen (3×5 versus 7×5). Durch diese *Sparsity* kann der Rechenaufwand gesenkt werden.
Darüber hinaus wird durch *weight sharing* die Anzahl Parameter reduziert (3 versus $7 \times 5 = 35$) und die selben gelernten Features können an unterschiedlichen Orten der Eingabe extrahiert werden. Dadurch wird Äquivarianz bezüglich Translationen garantiert.

Receiptive Field

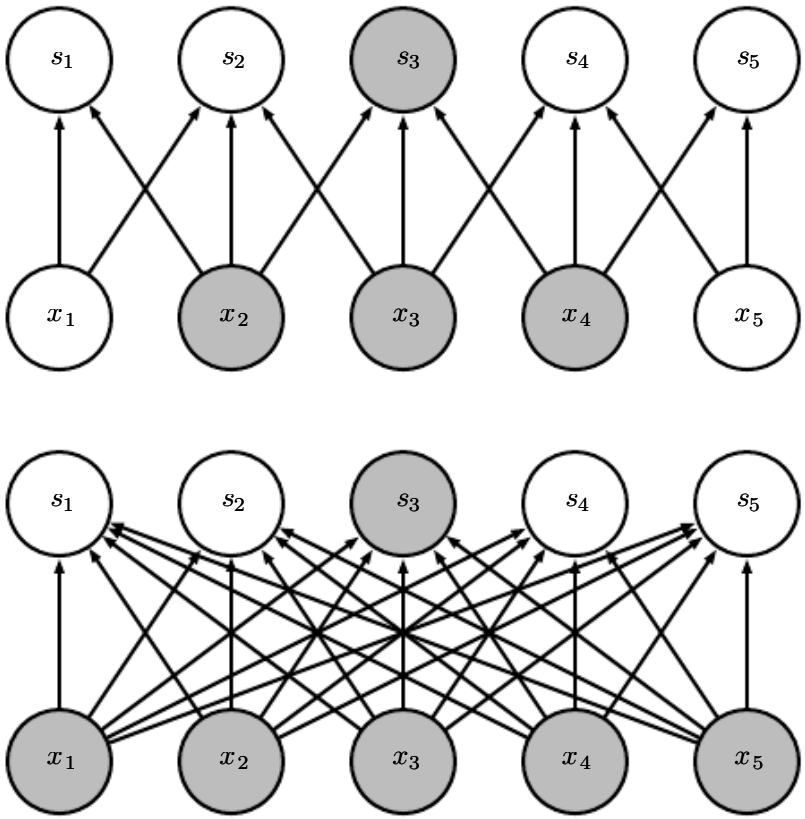


Figure: 9.3 von Goodfellow. Oben: Spärliche Verbindungen (Sparse Connectivity) durch Kernel der Grösse 3. Dadurch kann z.B. die Ausgabe von s_3 nur von den Nachbar-Units x_2, x_3, x_4 abhängen. Diese Nachbar-Units werden auch als *Receiptive Field* von s_3 bezeichnet. Unten: Bei einem Fully connected Layer kann jede Ausgabe in s von jeder Eingabe in x abhängen.

In einem Signal (z.B. Audio oder Bilder) sind benachbarte Werte (z.B. Pixel) oft stark korreliert. Diese lokalen Korrelationen können mit einem kleinen Receiptive Field bereits gut erfasst werden.

Receiptive Field

Beachte, dass auch bei kleinen Kernels Interaktionen mit einem grossen Gebiet entstehen können, wenn mehrere Layer in einem tiefen Netzwerk hintereinander angewendet werden. Dadurch kann Informationen von lokalen Features aus niedrigen Layern sukzessive zu komplexeren und globaleren Features in höheren Layern zusammengeführt werden.

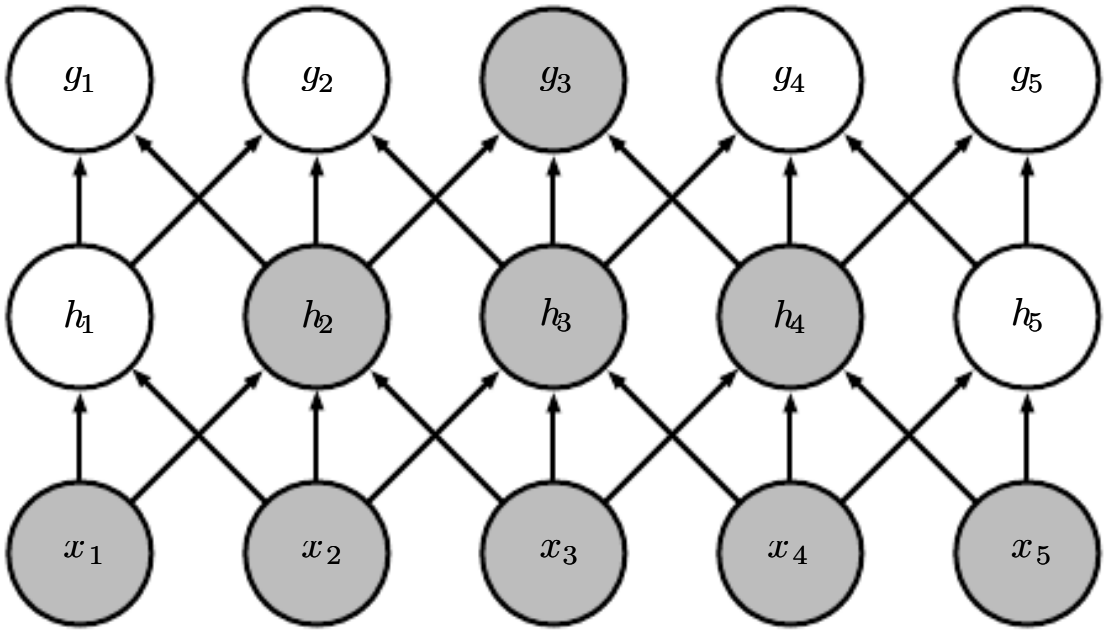


Figure: 9.4 von Goodfellow. Das Receiptive Field von g_3 wird durch mehrere Layer vergrössert.

Dilation

Eine Alternative, um das Interaktionsgebiet (Receptive Field) zu vergrössern, ohne die Anzahl Parameter des Kernels zu verändern ist Dilatation (Dilation). Dabei wird der Kernel ausgedehnt, indem Lücken zwischen den Gewichten gelassen werden.
Anbei ein Beispiel in 2D:

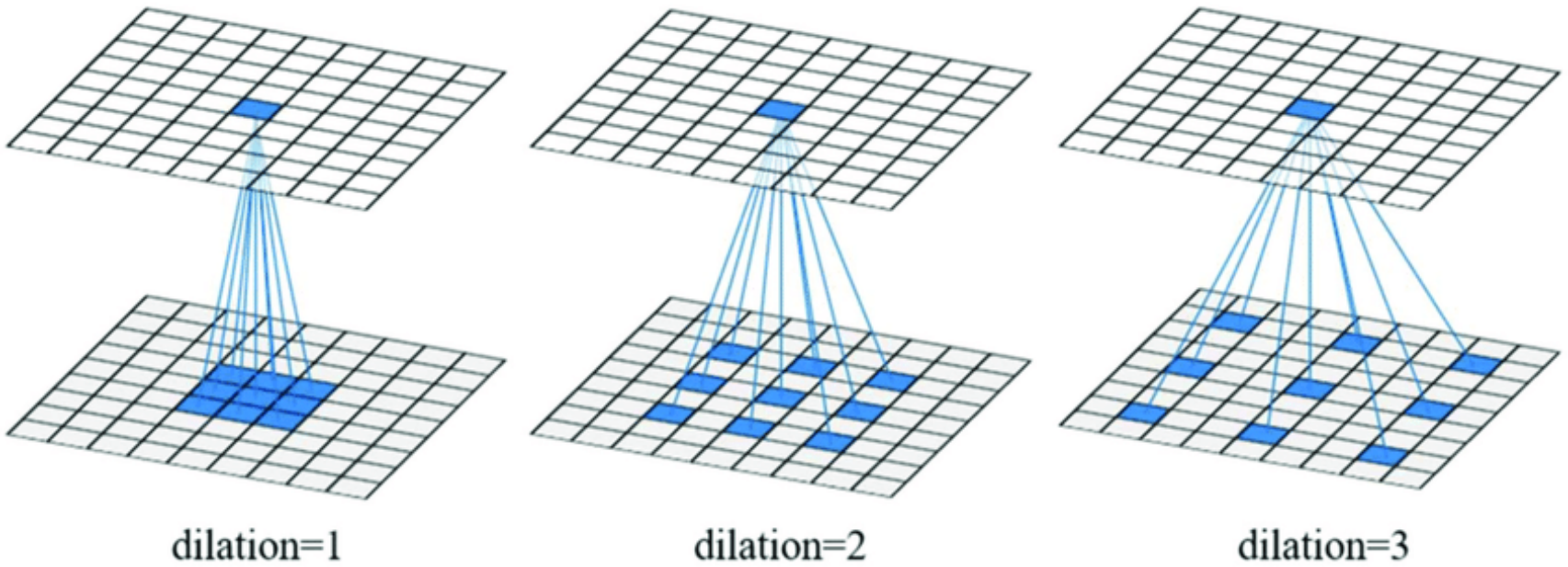


Figure: Dilation vergrössert das Interaktionsgebiet eines Kernels.

Padding

Bei einer Konvolutionsoperation wird die Grösse des Ausgabelayers mit der Grösse des Kernels x reduziert. Um dies zu verhindern können wir "Padding" an den Rändern der Eingabe einführen. Eine typische Wahl ist z.B. 0-Padding:

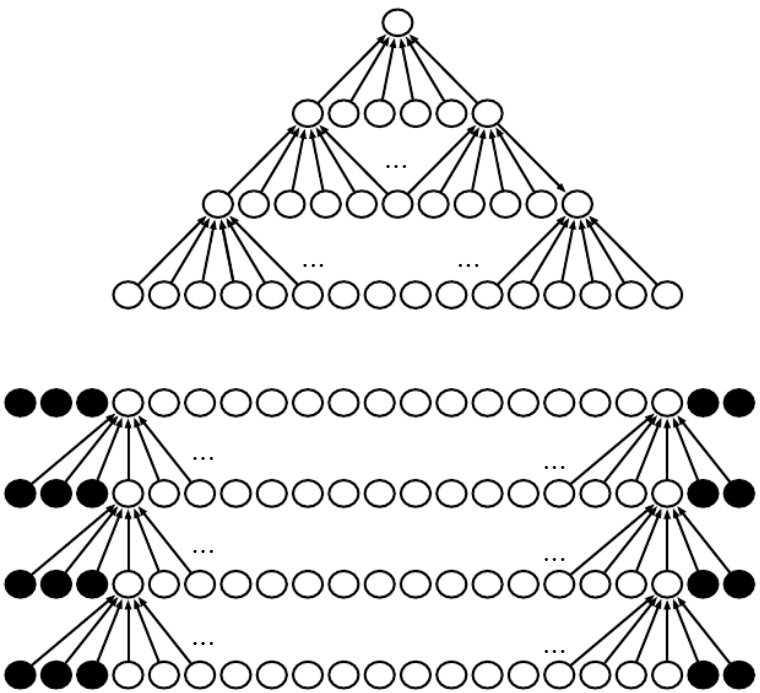


Figure: 9.13 von Goodfellow. Durch 0-Einträge an den Rändern kann die Grösse des Signals aufrechterhalten werden.

Padding-Varianten

Weitere Padding-Varianten sind:

- Spiegelung des Signals an den Rändern
- Periodisches Padding

Stride

Um den Rechenaufwand zu reduzieren, können wir die Ausgabegrösse verringern indem wir den Kernel nur alle n Positionen anwenden. Dies wird auch als Stride n bezeichnet und kann als Downsampling um Faktor n aufgefasst werden. Durch diese Stride-Operation kann zudem das Receiptive Field des nachfolgenden Layers um Faktor n vergrössert und somit globalere Features gelernt werden.

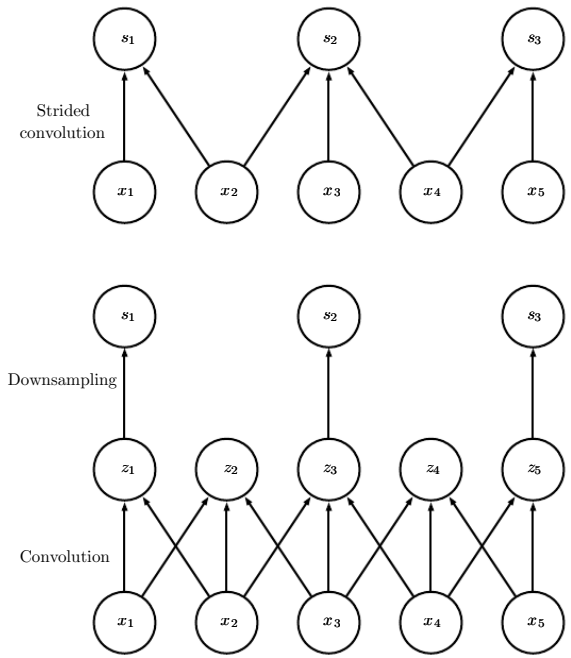


Figure: 9.12 von Goodfellow. Konvolutionen mit Stride 2 (oben) sind effizienter als naives Downsampling (unten), da unnötige Zwischenschritte (z_2, z_4) nicht berechnet werden müssen.

Pooling

Eine weitere Möglichkeit, um eine kleinere Repräsentation zu erhalten ist Pooling. Dabei werden Features aus einer Nachbarschaft extrahiert (z.B. der Mean oder Max über alle Werte aus der Nachbarschaft) gefolgt von einer Stride-Operation.

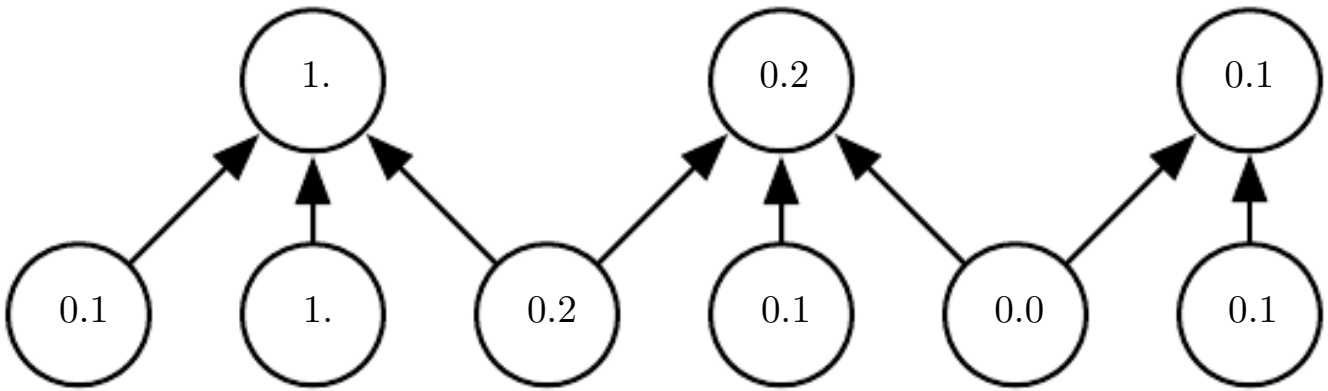


Figure: 9.10 von Goodfellow. Max-Pooling mit Stride 2 über eine Nachbarschaft der Grösse 3.

Max Pooling in 2D

Beispiel für Max-Pooling in 2D:

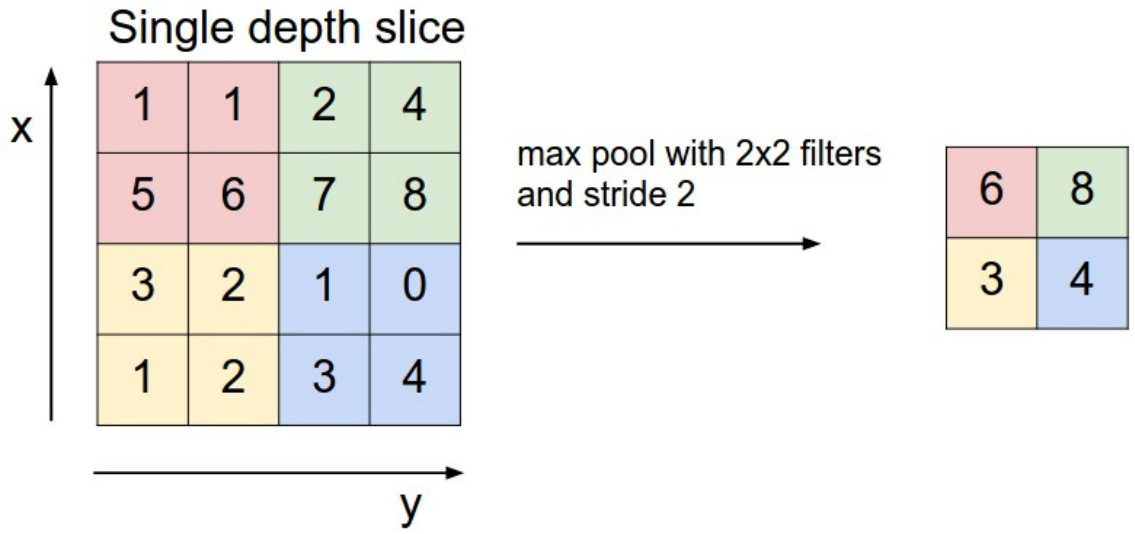


Figure: von github. Beispiel für Max-pooling in 2D über 2×2 Nachbarschaften mit Stride 2.

Maxpooling kann etwas Toleranz bezüglich Translationen erzielen, da Translationen des Maximalen Wertes innerhalb einer Nachbarschaft keine Auswirkungen auf die Ausgabe haben.
Durch Variation der Pooling-Grösse können zudem Eingabebilder mit unterschiedlichen Grössen durch dasselbe Netzwerk gehandhabt werden.

Convolutional Layer mit mehreren Kanälen

Oft besitzen die Eingabedaten mehrere Kanäle ("Channels"). Bei Farbbildern könnten dies zum Beispiel 3 Kanäle für RGB (Rot, Grün, Blau) oder HSV (Farbwert, Sättigung, Helligkeit) sein. In höheren Layern eines CNNs werden eine Vielzahl von Channels verwendet um unterschiedliche Features zu extrahieren.
Wenn ein Convolutional Layer n Input-channels auf m Output-Channels abbilden soll, dann werden für jeden Output-Channel n Kreuzkorrelationen für sämtliche Input-Channels berechnet und aufaddiert.
Der Kernel ist dann ein 4D-Tensor mit Grösse: $m \times n \times h \times w$
Wobei m der Anzahl Output-Channels, n der Anzahl Input-Channels, h der Kernel-Höhe und w der Kernel-Breite entspricht.

Transposed Convolutional Layer

Das transponierte (transposed) convolutional Layer funktioniert ganz ähnlich wie das convolutional Layer aber genau "umgekehrt". D.h. der Kernel wird mit der Eingabe gewichtet und dann auf die Ausgabe projeziert.

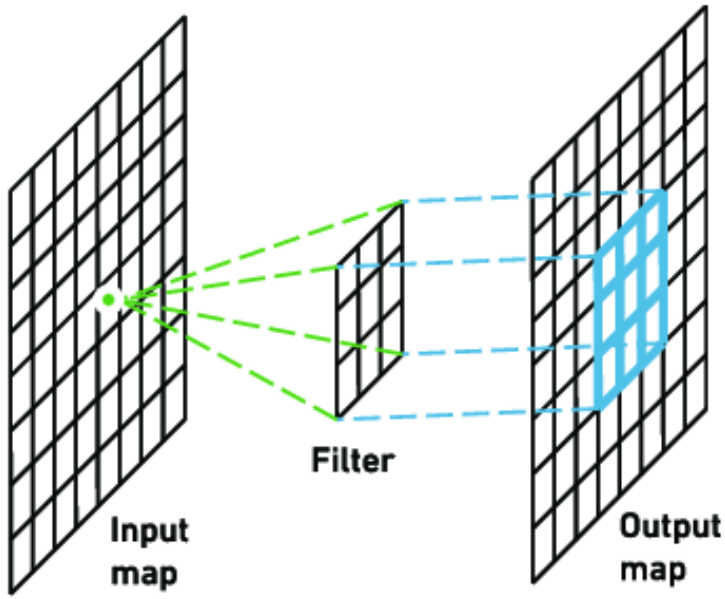


Figure: Transponierte Konvolution in 2D. Konzepte wie Stride und Dilation funktionieren ebenfalls analog aber genau umgekehrt zum convolutional Layer.

Transposed Convolutional Layer werden insbesondere in Decodern / generativen Modellen benötigt - z.B. wenn wir aus einer kompakten Repräsentation ein Bild erzeugen möchten.

Part 3

Batch Normalisierung

Chapter 8.7.1 of **Goodfellow-et-al-2016-Book**.

Batch Normalisieurng

Batch-Normlalisierung reparametriesiert ein Netzwerk. Die Normalisierung kann sowohl auf Eingaben als auch auf hidden Layers angewendet werden und reduziert (insbesondere bei CNNs) Probleme im Gradientenabstiegsverfahren.

Sei $\boldsymbol{H}$ eine Matrix, welche die Aktivierungen eines Minibatches für ein Layer, das normalisiert werden soll enthält (die Spalten der Matrix entsprechen einzelnen Units bzw Channels und die Zeilen entsprechen einzelnen Batch-Samples). Dann normalisiert $\boldsymbol{H}'$ die ursprünglichen Aktivierungen $\boldsymbol{H}$ wie folgt:

$$\boldsymbol{H}' = \frac{\boldsymbol{H} - \boldsymbol{\mu}}{\boldsymbol{\sigma}},$$

wobei μ und σ spaltenweise Mittelwerte und Standardabweichungen sind:

$$\boldsymbol{\mu} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{H}_{i,:} \qquad \boldsymbol{\sigma} = \sqrt{\delta + \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{H} - \boldsymbol{\mu})_i^2}$$

wobei δ ein kleiner positiver Wert (z.B. 10^{-8}) ist, um undefinierte Gradienten zu vermeiden, wenn die Wurzel gleich 0 ergibt.

Batch Normalisierung

Batch-Normalisierung führt zu besseren Initialisierungen

Wie bereits letzte Woche erwähnt, sollten die Eingaben für die Neuronen normalisiert sein, damit die Aktivierungsfunktionen zu Beginn des Trainings nicht direkt gesättigt werden und somit die Jacobians zu kleine Eigenwerte erhalten.

Batch-Normalisierung macht Gradient Descent effizienter

Durch Batch-Normalisierung verschwendet Gradient Descent keine Zeit mehr um einfach nur den Mittelwert / die Standardabweichung anzupassen, da der Normalisierungsschritt sich bereits darum kümmert.

Inferenz mit Batch-Normalisierung

Um ein Netzwerk mit Batch-Normalisierungs Layern auf einzelnen Bildern auszuwerten, muss während des Trainings ein running average für den Mittelwert μ und die Standardabweichung σ berechnet werden, welcher für die Inferenz verwendet werden kann.

Batch Normalisierung

Die Operation der Batch-Normalisierung reduziert die Aussagekraft eines Neurons, da Mittelwert und Standardabweichung der Ausgaben fixiert werden. Um dies zu verhindern, ist es üblich, nicht direkt $\boldsymbol{H'}$ sondern $\boldsymbol{\gamma H' + \beta}$ auszugeben, wobei $\boldsymbol{\gamma}$ und $\boldsymbol{\beta}$ gelernte Parameter sind, welche die Standardabweichung und den Mittelwert der Ausgabe separat lernen.

Für ein Layer der Form $\phi(\boldsymbol{XW} + \boldsymbol{b})$, wobei ϕ eine Aktivierungsfunktion ist, wird empfohlen, die Batchnormalisierung auf $\boldsymbol{XW}$ anzuwenden und den Bias $\boldsymbol{b}$ zu entfernen, da der Bias redundant mit dem $\boldsymbol{\beta}$ der Batch-Normalisierung wäre.

Part 4

Convolutional Neural Networks (CNNs)

LeNet5 - ein frühes CNN zur Klassifizierung von Ziffern

Ein frühes Beispiel eines CNN ist LeNet5 von Yann LeCun (1998):

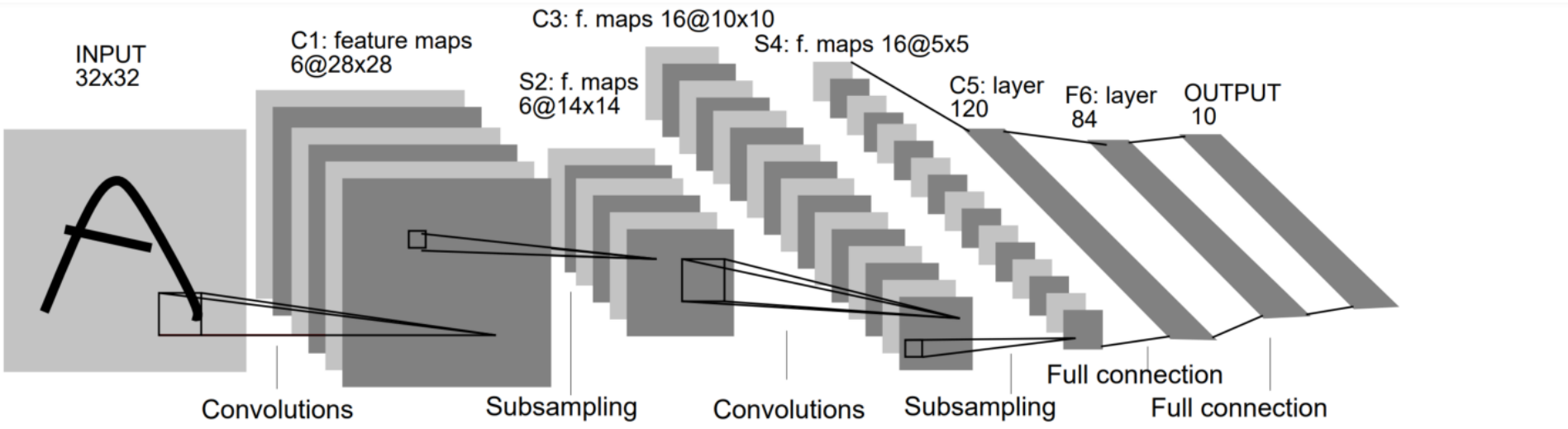


Figure: von Murphy. LeNet5 ist ein CNN zur Klassifikation von handschriftlichen Ziffern (LeCun 1998).

... modernere Architekturen werden wir nächste Woche besprechen.

Training

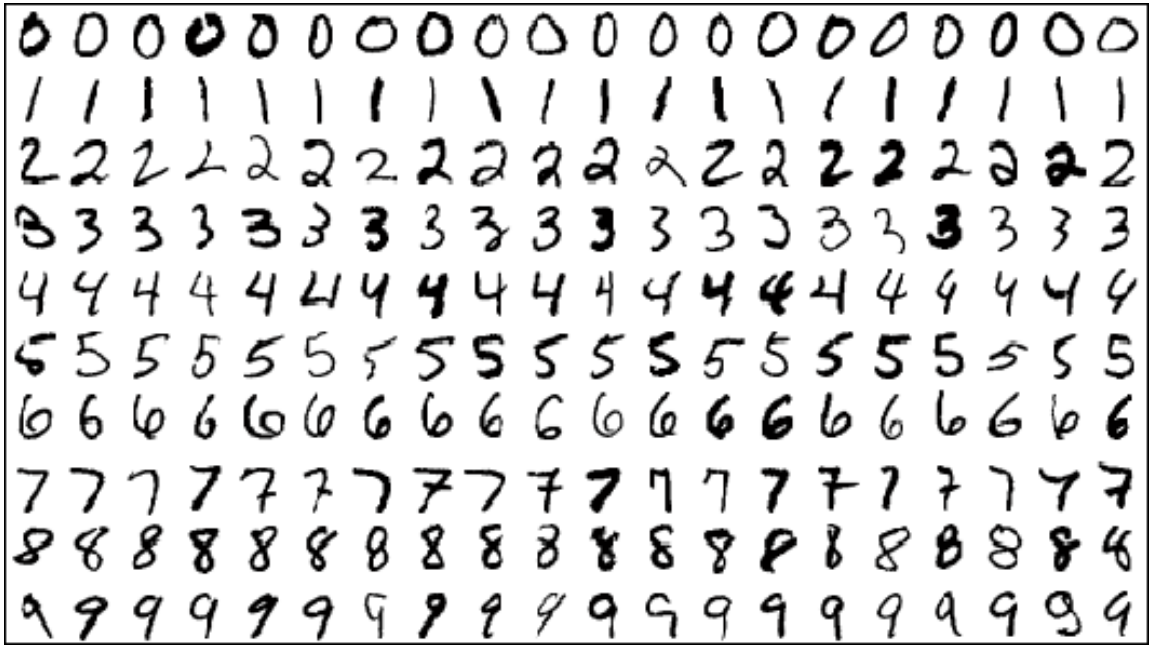


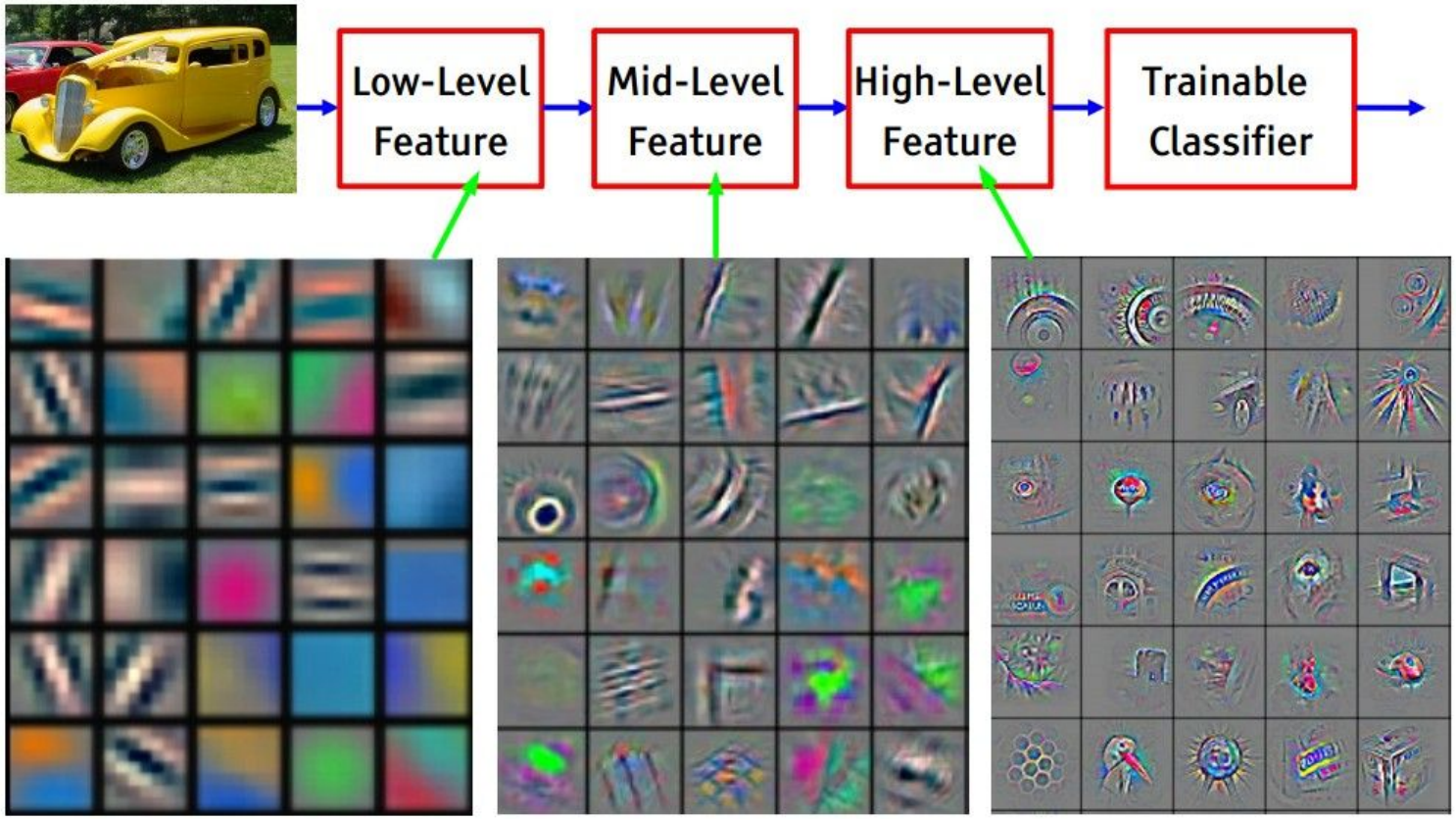
Figure: Beispiele aus dem MNIST dataset.

Das CNN wurde auf Ziffern aus dem MNIST datenset traininert. Damals (1998) hat das Training von LeNet5 2-3 Tage benötigt - heute kann ein CNN auf dem MNIST datenset in unter 1 Min Trainingszeit eine sehr gute Performance erzielen. Weitere Informationen dazu können [hier] gefunden werden.

LetNet5 Mistakes

Figure: von Murphy. Die 82 Fehler, die LeNet5 auf 10,000 Test-Daten des MNIST-Datensets gemacht hat (`korrektes Label` $\rightarrow$ `geschätztes Label`).

Hierarchie von Features in modernen CNNs



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

In den ersten Layern eines CNN werden Low-Level Features (z.B. Kanten) erkannt. In darauf folgenden Layern werden dann komplexere Mid-Level Features (z.B. Ecken / Kreise) erkannt und in den finalen Layern können High-Level Features (z.B. Texturen / Räder / etc) erkannt werden.

../deeplearn