

5.7) Bilderkennung mit Convolutional Neural Networks

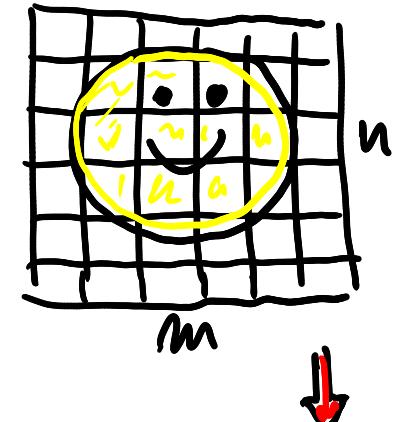
357

Die bisher betrachteten FFNNs (Feedforward NNs) erwarten einen Input-Vektor als Eingabe im die Input-Layer!

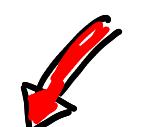


$$\vec{x} = (x_1, \dots, x_n)$$

Falls z.B. ein Bild klassifiziert werden soll hat man aber 2D-Daten



Um so einen Input zu verarbeiten muss man also einen Vektor daraus machen

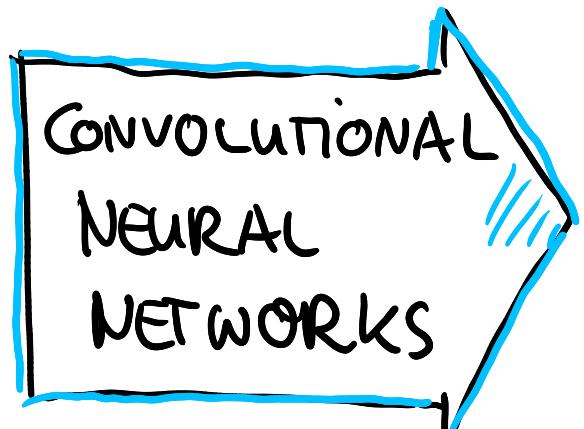
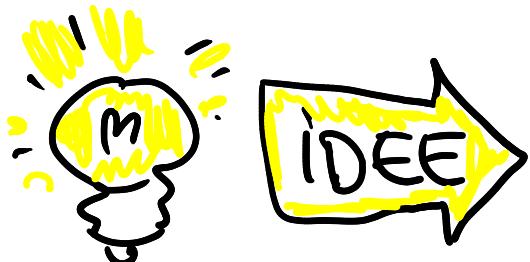


PROBLEM:

Nachbarschaftsbeziehungen zwischen den Pixeln gehen verloren!

Außerdem:

- Bilder stellen einen extrem hochdimensionalen Input dar
- viel redundante oder nützlose Information

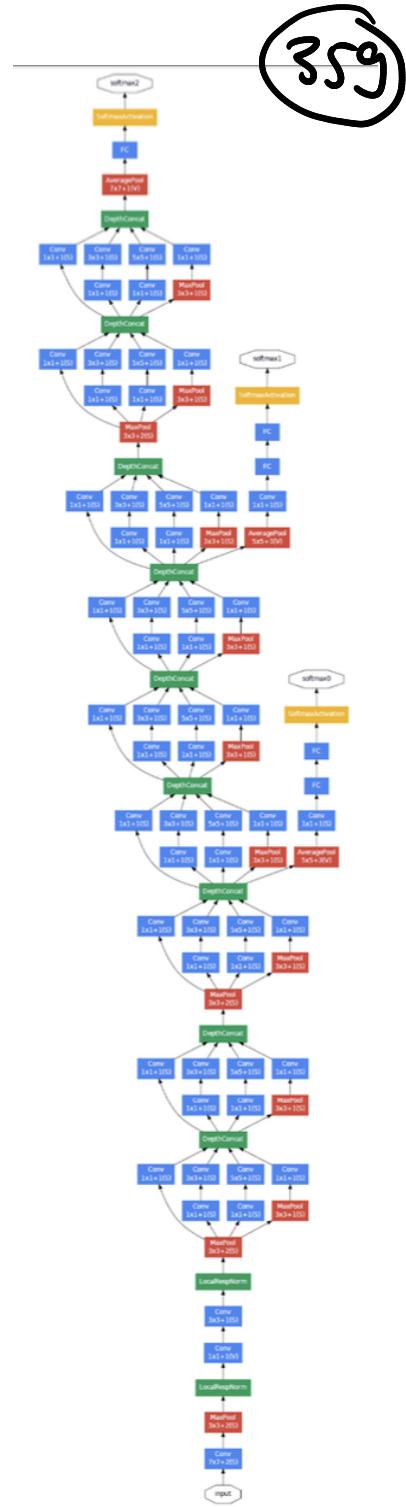


- überträgt Ideen aus der klassischen Bildverarbeitung (z.B. Filter) auf NNs
- Lerne eine effiziente Vorverarbeitung des originalen 2D - Inputs um relevante Features zu identifizieren und Unnötiges auszusortieren

5.7.1) Die Bausteine von CNNs

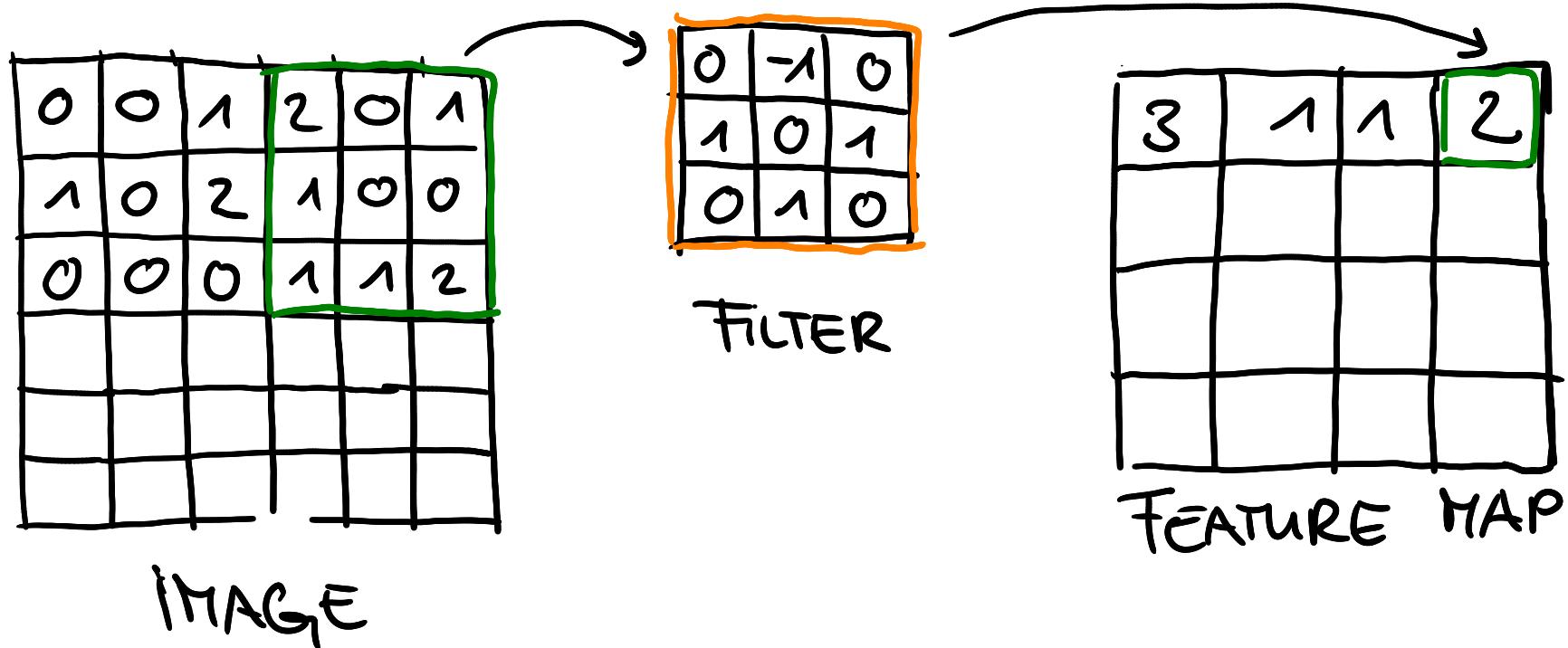
359

- CNNs sind tiefe neuronale Netze, die aus einem laugen Stack von interagierenden Schichten bestehen.
 - Der größte Teil des CNNs kann als dynamisch lernende Vorverarbeitung des 2D-Inputs verstanden werden
 - Der eigentliche Klassifikator (Regressor) macht nur einen winzigen Teil aus.
 - Die einzelnen Schichten von CNNs übernehmen ganz spezielle Aufgaben!

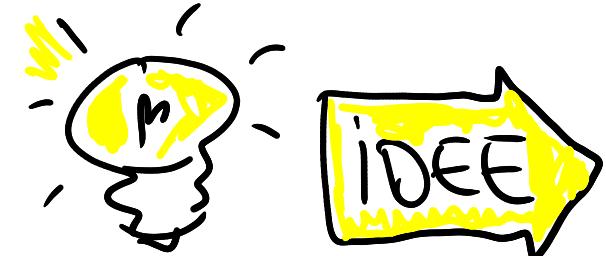


① Die Convolution Layer

Convolution Layers greifen die Idee von Filtern aus der klassischen Bildverarbeitung auf



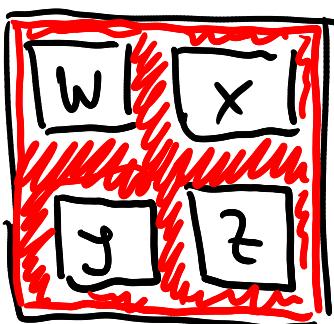
- Filter wandert mit festgelegter Schrittweite über das Bild
- Feature Map = Ergebnis einer Faltung



Statt den Filter *a priori* festzulegen
könnte man die optimale Struktur
auch lernen!

\Rightarrow Convolution layer

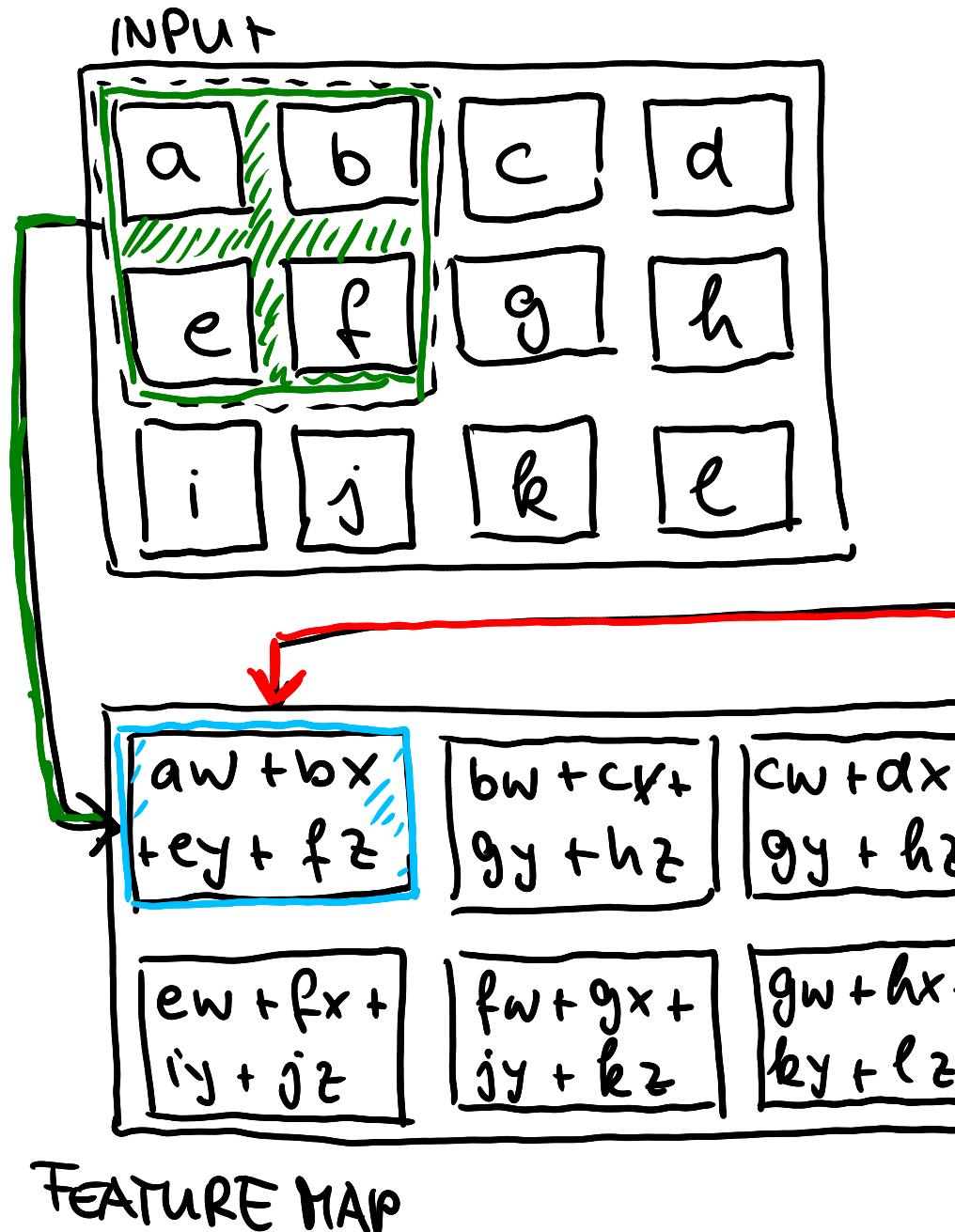
Bsp. (2×2) -Filter



im Gegensatz zur klassischen
Bildverarbeitung ist der Filter
zun Anfang noch nicht fixiert.

(2×2) -Filter \Rightarrow 4 unbekannte Einträge,
müssen optimiert werden!

Faltung des Filters mit dem Input



Das Ergebnis hängt ab von:

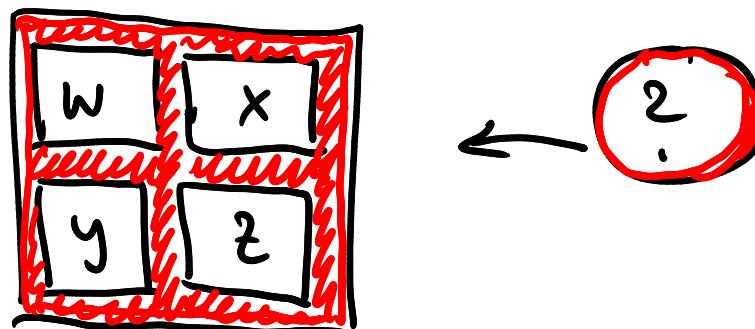
- Größe des Filters
 - Schrittweite
 - darf der Filter den Bereich nicht (teilweise) verlassen?
- Wenn nicht :
VALID CONVOLUTION

Bemerkung:

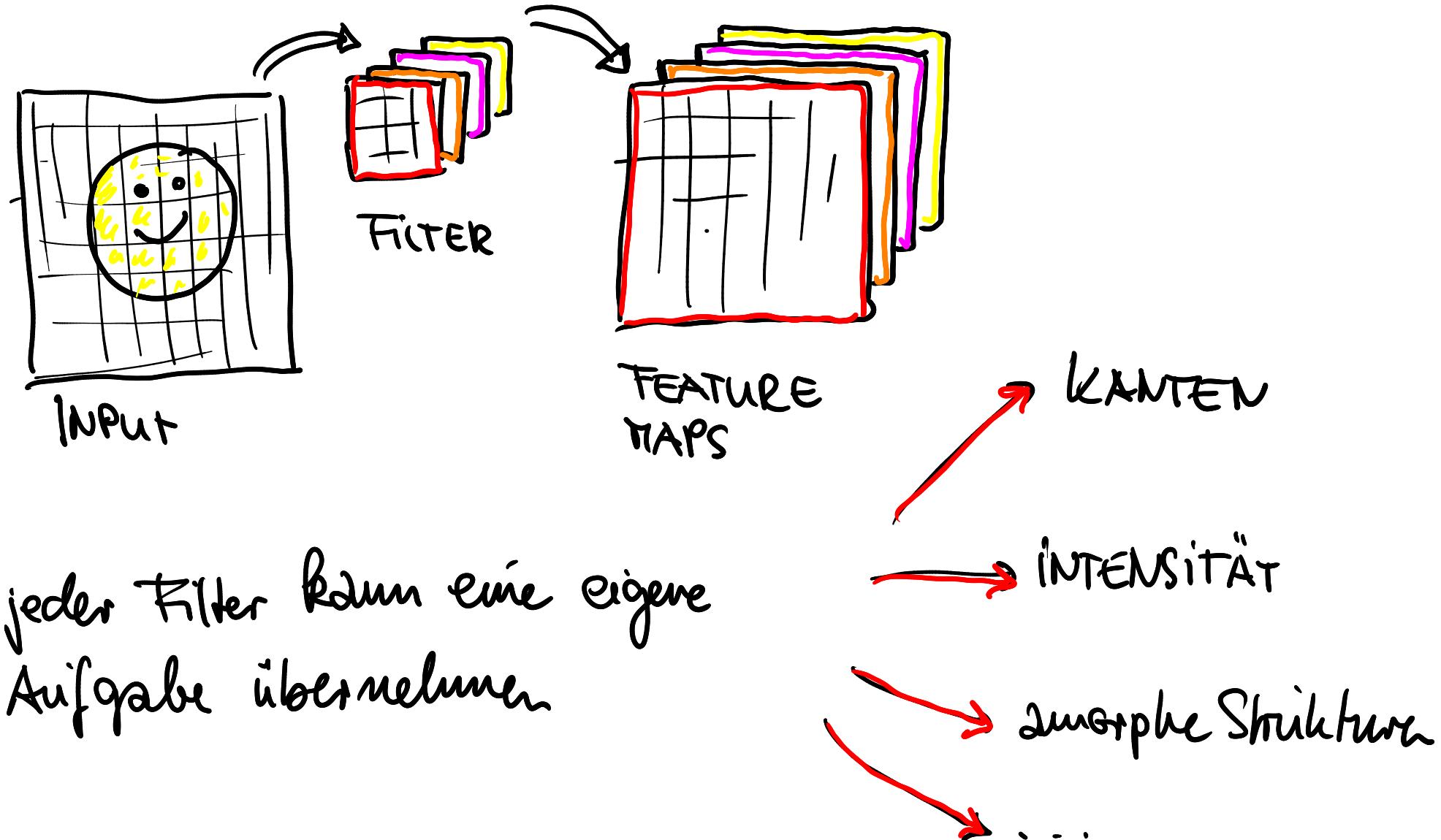
zu Beginn des Lernens startet man mit einer zufälligen Belegung des Filters

→ w, x, y, z sind Modellparameter

→ Das Modell soll während des Trainings selbst herausfinden, wie ein Filter aussehen soll, der die Info aus dem Input optimal verarbeitet.



... im der Regel wird ein Filter nicht ausreichen,
sondern man wird mehrere brauchen ...



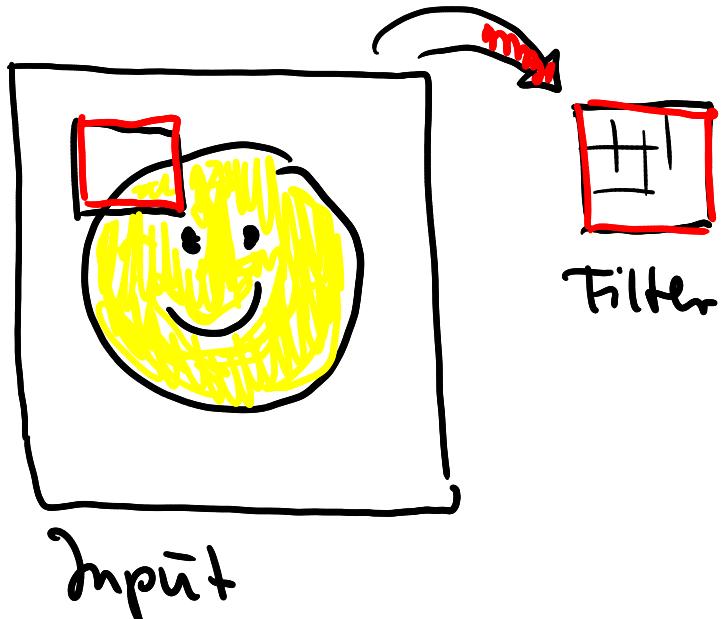
Bei der Umsetzung der Convolutional -NNs werden
3 grundlegende Prinzipien addressiert:

- ① SPARSE INTERACTIONS
- ② SHARED WEIGHTS
- ③ EQUIVARIANT REPRESENTATION

→ nur mit Hilfe des Zusammenspiels dieser 3 Komponenten lassen sich CNNs überhaupt realisieren

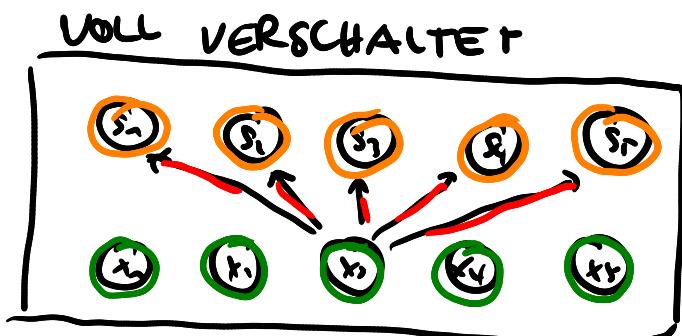
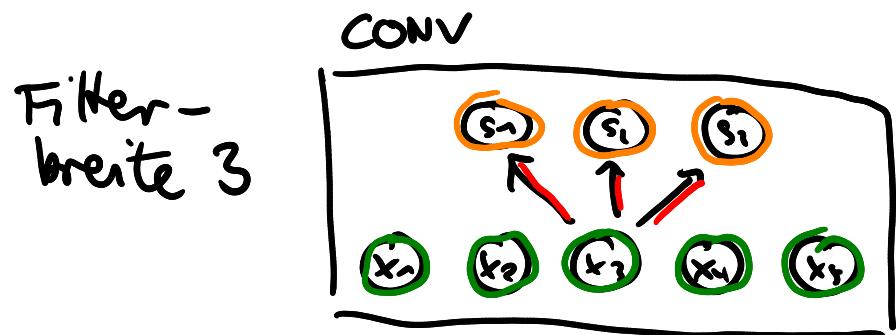
① SPARSE INTERACTION

366



Input

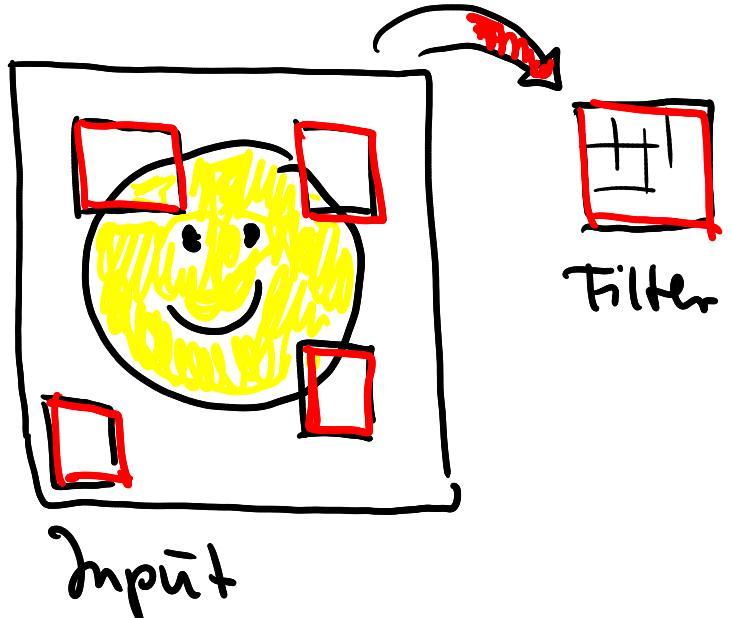
← der Filter interagiert NIE mit allen Input-Pixeln gleichzeitig, sondern sieht immer nur einen kleinen Ausschnitt davon \Rightarrow weniger Parameter im Netz!



... viel weniger Konnektoren!!

② SHARED WEIGHTS

367



- ein Filter wird an jeder Stelle des Inputs benutzt
- für eine Convolution werden 1, 2, 3, ... fest belegte Filter gelernt
- der komplette Input "teilt" sich die Filterelemente (= weights) jedes Filters

⇒ SHARED WEIGHTS

Dahinter steckt folgende Logik:

- ein Filter, der z.B. Kräuter erkennen\scharfen kann soll das auf dem ganzen Bild (gleich) tun
 - die resultierende Feature-Map ist dann das geschärfte Bild
- ⇒ die Art der (am sinnvollsten) zu extrahierenden Features ist überall im Bild gleich wichtig
- ⇒ die Features sollen überall gleich abgeleitet werden
- ⇒ gleiche Filter-Struktur!



Funktioniert ein gelernter Filter immer noch so gut, wenn das Eingangsbild verschoben oder rotiert wird?



bilden die Basis für eine Klassifikation



Invarianz bezüglich dieser Transformationen ist entscheidend für eine brauchbare Klassifikation

③ EQUIVARIANT REPRESENTATION

370

→ Translations - bzw. Relationsinvarianz?

... erreicht man durch Cov-Lager alleine nicht.

Man braucht dazu noch einen weiteren Baustein:

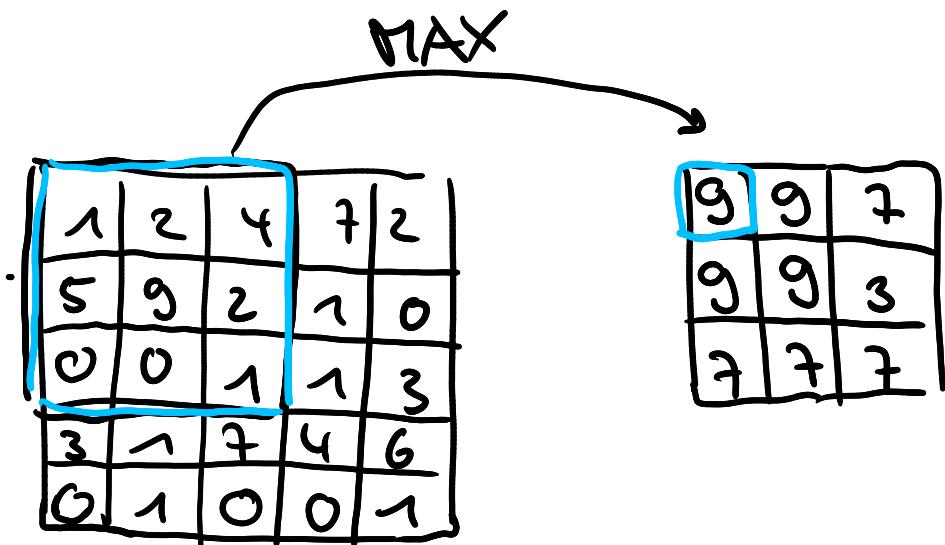
② Die Pooling - Lagen



Ursprünglich eingeführt um
eine Dimensionsreduktion
sehr großer Feature-Maps
zu erreichen

	1	2	4	7	2
	5	9	2	1	0
	0	0	1	1	3

↳ Maske:
- min Maximum } Max
- min Average } Pooling
 Average



Reduktion von
 $5 \times 5 = 25$ auf
 nur noch $3 \times 3 = 9$

→ schiebe die Maske mit einer gegebenen Schrittweite über die Feature-Map.

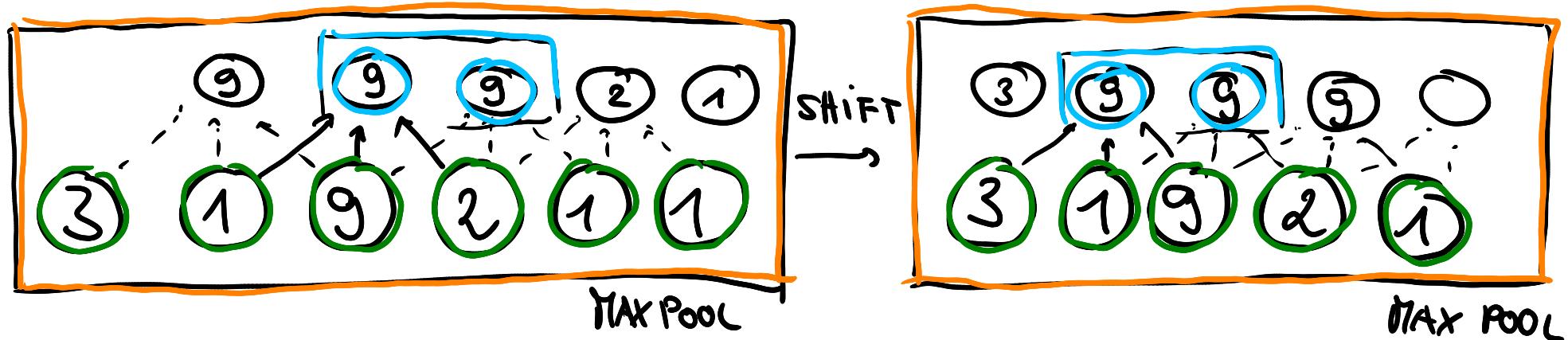
Der Output der Pooling-Schicht ist eine Art statischer Zusammenfassung der entsprechenden Einträge

→ entweder Maximum (Max Pooling) oder Mittelwert (Average Pooling)

Weiterer Effekt:

372

TRANSLATIONSINVARIANZ



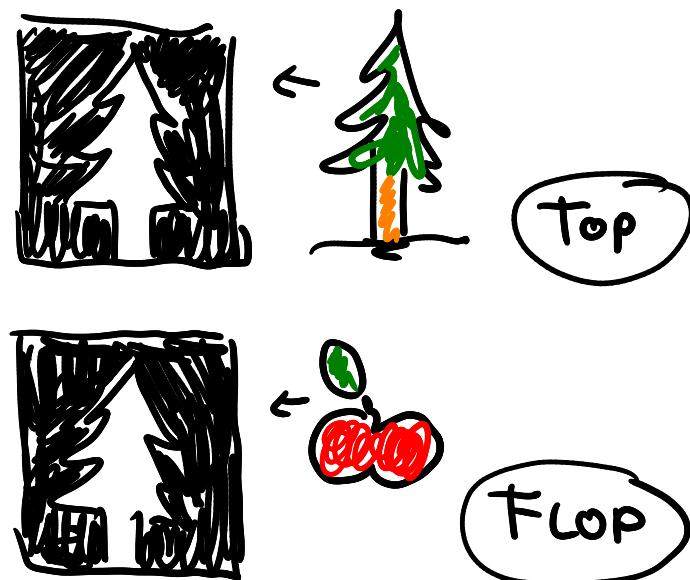
- verschiebt man das Eingangssignal um ein kleines Stück nach rechts, dann verändert sich die Repräsentation der Pooling-Schicht wenig
- ⇒ Ergebnis ist stabil gegen kleine Verschiebungen
- ⇒ sinnvolle (stabile) Feature Map für die Klassifikation

ROTATIONSSINVARIANZ

↓
Zusammenispiel von:

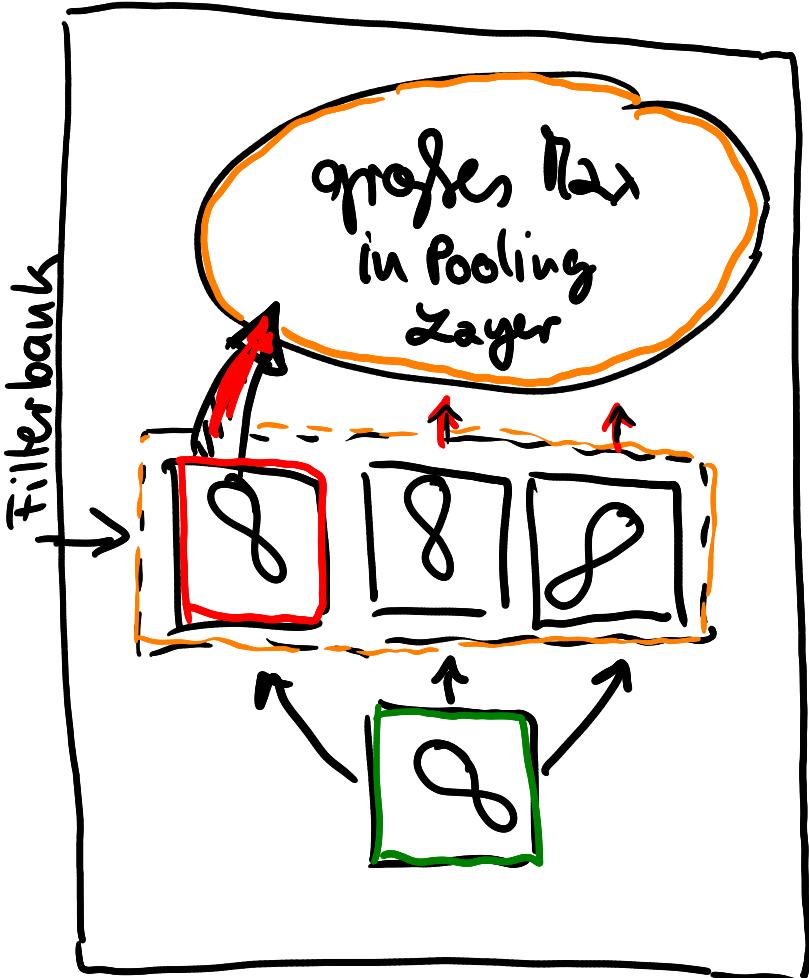
- CONV-LAYERN mit mehreren Filtern
- POOLING-LAYERN

Vorstellung: jeder Filter stellt eine Art "Lochblende" oder "Maske" dar, der nur bestimmte Signale durchlässt



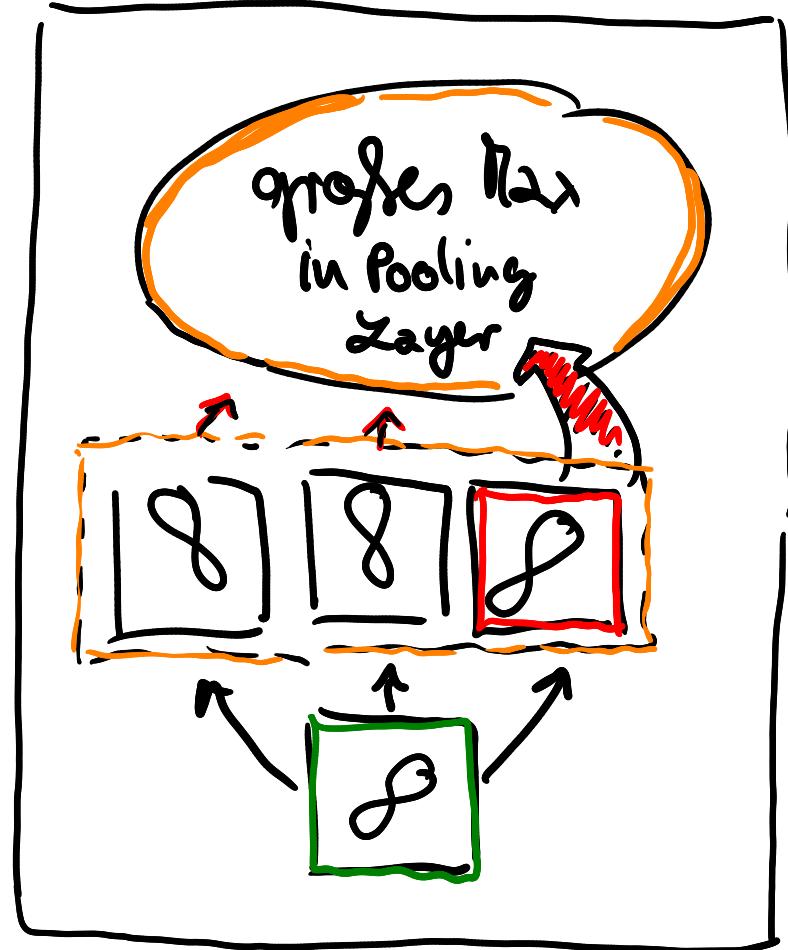
Bsp.: 3 geteilte Filter

374



Multi-Channel Approach

sichert die
Invarianz gegen-
über einer
Rotation der
Eingangssignale



- maximale Aktivierung im Detektor

5.7.2) Padding

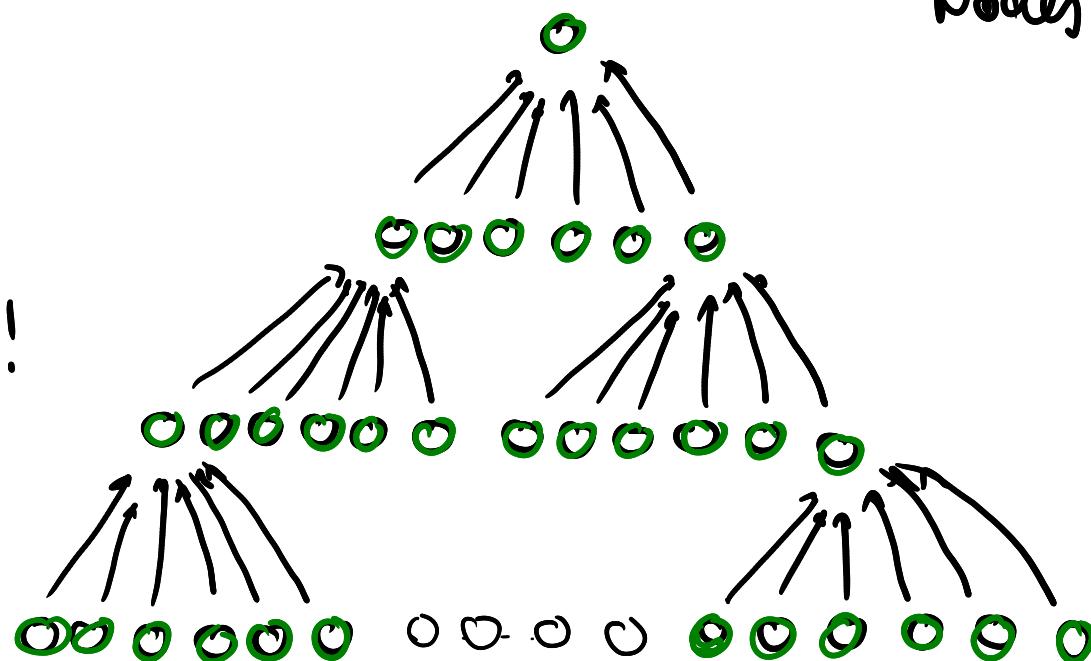
Bisher war bei der Convolution nur eine Position des Filters erlaubt, bei der das Fenster die Input-Matrix nicht verlässt.

Beobachtung

Filterbreite 6

⇒ Reduktion:

5 Pixel je Schritt!!

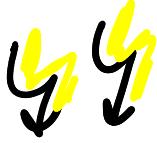


Die Convolution reduziert die Zahl der Nodes

→ nur 3 Schichten möglich!!

ABER

Wir wollen ja auch noch Pooling einsetzen
und eigentlich tiefe Netze konstruieren



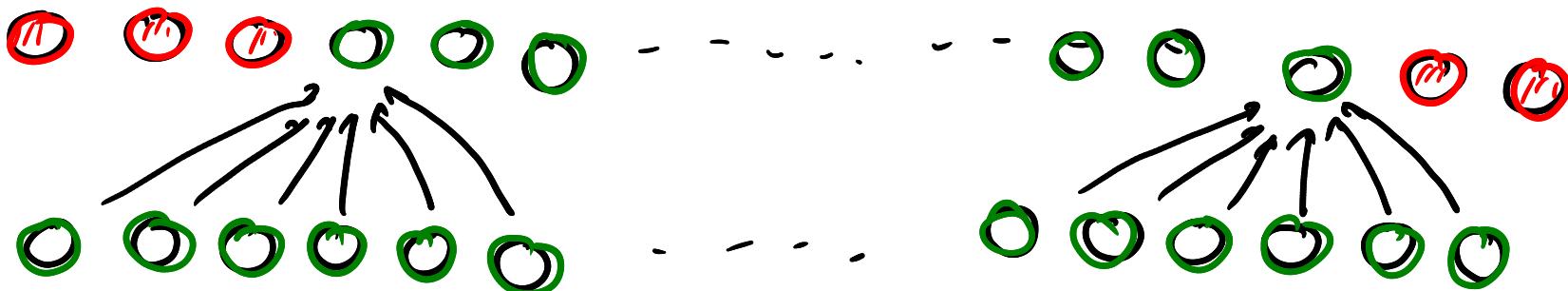
LÖSUNG

ergänze entsprechend viele Pixel mit Wert Null am Rand

⇒ beliebig viele Schichten möglich

= 0
"NEU":
synthet. Werte:

ÜSW.



→ Das sogenannte PADDING = Ergänzen mit 0er Pixeln
am Rand

377

ist ein essentielles Feature tiefer CNNs

3 Fälle:

① Kein Padding → Valid Convolution

- es sind nur komplette Fenster zugelassen
- für ein Bild der Breite m und einen Filter der Größe k gilt:
 $(m - k + 1)$ -dimensionaler Output
- die Anzahl der Schichten ist limitiert!

② Zero-Padding → Same Convolution

- der Output hat die gleiche Größe wie der Input
- die Inputs am Rand beeinflussen den Output weniger als die im Zentrum (kommen seltener dran beim Durchlaufen)
- beliebig viele Schichten sind möglich

③ Füll Convolution

- es werden so viele Nullen am Rand ergänzt, dass jedes Input-Pixel k-mal in jeder Richtung berücksichtigt wird
 - der Output hat dann die Dimension $m+k+1$
- Die optimale Lösung liegt (Empirie ☺) meistens irgendwo zwischen ① und ②