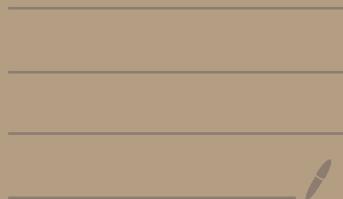


DL for CV

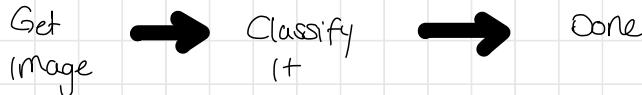
SS 20



Before:



2012 - Now:



=> End-to-End approach

Image Net Challenge since 2010

- Bewertung: Top 5 Klassen die ausgegeben werden müssen korrektes Label enthalten
(Top 5 da manchmal mehrere Objekte im Bild)
- 1000 Klassen

2012: Alex Net 15,3% Fehler vs. 26,1% Fehler

Computer Vision Tasks

single object

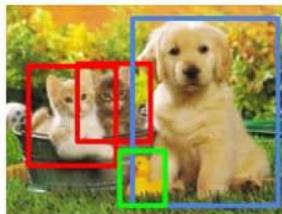


CAT



CAT

multiple objects



CAT, DOG, DUCK



CAT, DOG, DUCK

Classification

Classification + Localization

Object Detection

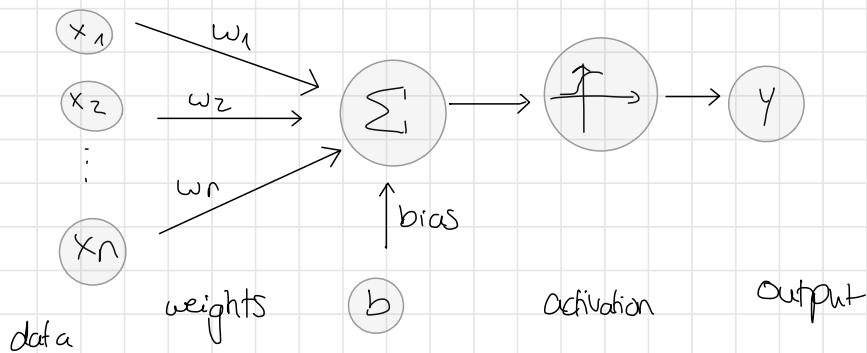
Instance Segmentation

NN Basics

Idee: End-to-end learning einer Hierarchie

- ↪ low-level features in frühen layers } für versch. Tasks
- mid-level mittleren } wiederverwenden
- high-level später

Single Layer Perceptron

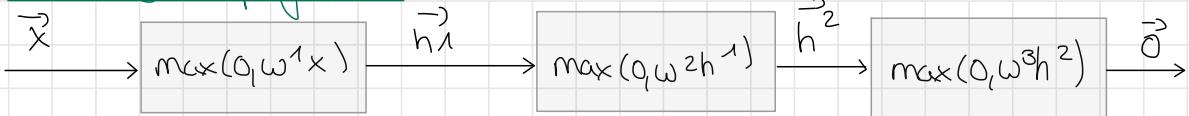


$$y = \text{activation} (w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b)$$

nicht-linear → sonst nur lineare Probleme lösbar auch mit mehreren Perceptrons / Layers

Multi-layer Perceptron

Forward Propagation:



$\vec{\theta} = \text{Modell} = w^i, b^i$ für alle Layer

$\vec{x} \in \mathbb{R}^D$ $D = \# \text{ Pixel (ausgerollt)}$

$w_i \in \mathbb{R}^{N_1, N_{i-1}}$ $N_i = \# \text{ Units in Layer i}$

$\vec{b}_i, h_i \in \mathbb{R}^{N_i}$ $\vec{o} \in \mathbb{R}^C$ $C = \# \text{ Klassen}$

Loss

Ausgabe \vec{o} als Wktverteilung: $\sum \text{aller Werte aus } \vec{o} = 1$

Softmax: Wkt von Klasse k geg. Eingabe \vec{x}

$$p(c_k=1 | \vec{x}) = \frac{e^{o_k}}{\sum_j e^{o_j}}$$
$$\vec{o} = \begin{pmatrix} o_1 \\ o_2 \\ \vdots \\ o_k \\ \vdots \\ o_C \end{pmatrix} = \text{Ausgabe}$$

Negative log likelihood: pro Sample \vec{x}

$$L(\vec{x}, \vec{y}, \vec{o}) = - \sum_j y_j \log p(c_j | \vec{x})$$

$\vec{y} = \begin{pmatrix} 0 & \dots & 0 & \xrightarrow{j} & 1 & \dots & 0 & \dots & 0 \end{pmatrix}_c$

tatsächliche Klasse j vorhergesagte Klasse j

Modellloss:

$$\vec{o}^* = \underset{\theta}{\operatorname{argmin}} \sum_n L(\vec{x}^n, \vec{y}^n, \vec{o})$$

Backprop Ziel: minimierte Loss
↳ propagiert Loss zum Anfang zurück

→ Berechne Ableitung des Loss nach Params

① Ableitung Loss nach output \vec{o}

$$\frac{\partial L}{\partial o} = \underbrace{p(c|\vec{x})}_{\text{Vorhersage}} - \underbrace{\vec{y}}_{\text{ground truth}}$$

② Ableitung loss nach $\vec{w}_3 \rightarrow \sim \text{kettenregel}$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial w_3} = (p(c|\vec{x}) - \vec{y}) \cdot \vec{h}^2$$

$\vec{o} = \vec{w}_3 \vec{h}_2$

③ Ableitung loss nach \vec{h}_2

$$\frac{\partial L}{\partial h_2} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial h_2} = (p(c|\vec{x}) - \vec{y}) \cdot w_3$$

④ Ableitung loss nach w_2

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial h_2} \cdot \frac{\partial h_2}{\partial w_2} = \frac{\partial L}{\partial h_2} \cdot \frac{\partial h_2}{\partial w_2} = \dots$$

- - -

Lernverfahren

Learning Rate η wichtigster Hyperparam

- zu hoch : pendelt um min / divergiert
- zu klein : langsame Konvergenz

Gradient Descent

Passe Modell Θ (Gewichte, Bias) entsprechend Loss an:

$$w_i = w_i - \eta \frac{\partial L}{\partial w_i}$$

Batch Gradient Descent: Modell mit Params Θ
n Trainingsdaten

\rightarrow minimierte $L(\Theta) = \sum_{i=1}^n L_i(\Theta)$ \rightsquigarrow alle Trainingsdaten in jedem Schritt benutzt

- (+) convergiert für konvexe Fehlerflächen ggf. globales und lokales
- (-) teuer falls n groß!

Mini-Batch GD (Stochastic GD)

approx. Σ mit mini-batch (32, 64, ...)

Momentum

Problem: Fehlverteilung nicht konvex
↳ viele lokale Optima



\rightarrow hyperparam μ (≈ 0.9)

$$\Delta_t = \mu \Delta_{t-1} - \eta \frac{\partial L(\Theta)}{\partial \Theta} ; \quad \Theta = \Theta + \Delta_t$$

↑
 update vec
 t zeit
 aus Schritt davor

\rightarrow wird schneller falls Gradient sich in gleiche Richtung ändert
 \rightarrow schneller Konvergenz
 langsamer andere

\rightarrow weniger Fluktuation

Learning Rate Annealing

- Lernrate verringern während Training

zB 0.1 am Anfang

0.01 nach 100 Epochen

0.001 nach 200 Epochen

Nesterov accelerated Gradient

≈ Momentum aber mit lookahead statt Δ_{t-1}

Adagrad

- Lernrate anders je Gewicht θ_i
- Gewichte mit hohem Gradient \rightarrow reduziere η
kleinem \rightarrow erhöhe η

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

Diag Matrix \nearrow

mit $G_{t,ii} \stackrel{?}{=} \text{Summe der Quadrate der Gradienten von } \theta_i \text{ bis Zeit } t$

Adadelta

≈ Adagrad mit weniger aggressivem Anpassen von η
 \rightarrow Anpassung von η nur in bestimmtem Fenster w

Adam

≈ Adadelta + Momentum

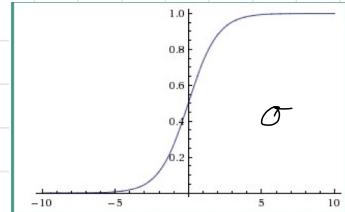
- Mittelwert der 2. Momenten der Gradienten

Activation Functions

- nicht-linear! sonst keine nicht-linearen Features lernbar
→ Komposition von Linearitaten = linear
- differenzierbar (da Backprop)

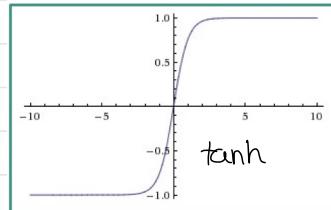
Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



- ⊖ Vanishing gradients:
Ableitung ≈ 0 für große / kleine x
- nicht σ -zentriert: nur positive Inputs: Gradienten alle pos oder alle neg → zick/zack Dynamik

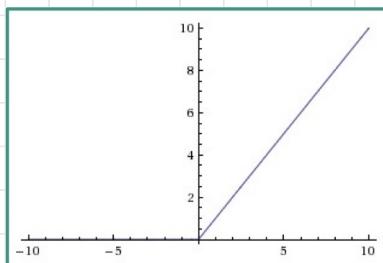
Tanh $\tanh(x)$



Relu

$$f(x) = \max(0, x)$$

- ⊕ kein vanishing gradient
- ⊖ dying Relu: starker Gradientenfluss kann Gewichte so updaten dass Gradient ab dann immer 0

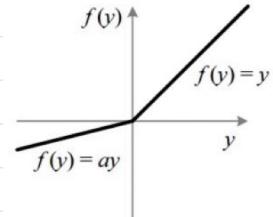


Leaky Relu

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

$\alpha = 0.1$ typisch

→ versucht dying Relu zu fixen



Maxout = Generalisierte Relu, leaky Relu

$$f(x) = \max(\omega_1^T x + b_1, \omega_2^T x + b_2)$$

- + kein dying Relu
- doppelt so viele Params

Anwendung: Relu → Leaky Relu / Maxout → tanh
 ↳ sigmoid die

Loss Funktionen

Abhängig vom Task

Classification

neg. log
Likelihood

Hinge
Loss

Regression

L1
MAE

L2
MSE

Metric / Similarity Learning

Triplet loss

Classification = Vorhersage eines Labels

Negative log-Likelihood (pro Sample x)

$$L(\vec{x}, y) = - \sum_j y_j \log p(c_j | x)$$

Hinge Loss (für SVMs)

$$L(x, y) = \sum_j \max(0, 1 - x_i y_i)$$

Regression = Vorhersage einer/mehrerer kontinuierlicher Features y_1, \dots, y_n

→ minimiere Distanz zw. Vorhersage x_j und Wahrheit y_j

L_1 -Loss Mean Absolute Error

$$L(\vec{x}, y) = \sum_j |y_j - x_j|$$

L_2 -Loss Mean Squared Error

$$L(\vec{x}, y) = \sum_j (y_j - x_j)^2$$

Metric / Similarity learning = Distanz zw. Objekten, zB Gesichtserkennung

Triplet Loss

Input = 3 Bilder a: Anchor { gleiches Gesicht }

p: positive

n: negative { anderes Gesicht }

$$\sum_{(a, p, n) \in T} \max \left\{ 0, \alpha - \underbrace{\|x_a - x_n\|_2^2}_{\text{normalenw.}} + \underbrace{\|x_a - x_p\|_2^2}_{\text{maximieren}} \right\}$$

Convolutional Neural Networks

Problem Fully Connected:

- zu viele Params
- ↳ verschwendung von Ressourcen
- zu wenige Trainingsdaten
- 40k hidden Units
2 Layer
≈ 1,6 Mrd Params
- keine lokalen Muster erkennbar

Warum jetzt?

- viele Daten verfügbar
- CPU ~ GPU → deutlich schnelleres Training

Faltung / convolution:

= Gewichtete Summe des Inputs

Bild:

p_{11}	p_{12}	p_{13}	p_{14}
p_{21}	..	:	:
p_{31}	..	p_{33}	:
p_{41}	p_{44}

Filter:

f_{11}	f_{12}	f_{13}
f_{21}	f_{22}	f_{23}
f_{31}	f_{32}	f_{33}

$$\rightarrow \text{output } \vec{o} : o_{11} = f_{11} \cdot p_{11} + f_{12} \cdot p_{12} + \dots + f_{33} \cdot p_{33}$$

Conv Layer

Input $W \times H \times C \leftarrow \# \text{ Kanäle}$
z.B. 3 bei RGB

- > jeden Kanal als einzelne Eingabeschicht
- je Kanal ein Filter

z.B. Input $224 \times 224 \times 3$

↳ Filter ist dann $f \times f \times 3$

Conv Operation

- Filter über Bild schieben mit Stride S
- jeder Kanal bekommt eigenen Filter
- neuer Pixelwert ist Summe über alle Kanäle

→ Ausgabe: Activation Map

→ je "Filtersatz" eine über alle Kanäle

Bsp.: Input $224 \times 224 \times 3$
Filter $5 \times 5 \times 3$

→ Ausgabe ist eine Activation Map $w \times h \times \underline{1}$

→ normalerweise mehr als 1 Filter pro Layer

zB 96 Filter der Größe $5 \times 5 \times \text{#Channels}$ Layer davor

→ Ausgabe $w \times h \times 96 \leftarrow 96$ activation maps

das ist Eingabe für Layer $l+1$

Formel

$$h_j^l = \max(0, \sum_{k=1}^{n_c} h_k^{l-1} * w_{kj}^l)$$

↑
output
feature map

↑
Summe
über alle
Kanäle der
Eingabe

↑
relu
Filter
input feature map

Dimensionen Layer l

Input: $H^{l-1} \times W^{l-1} \times n_C^{l-1}$

→ Output $H^l \times W^l \times n_C^l < \hat{\approx} \# \text{Filter in Layer } l$

$$H^l = \left\lceil \frac{H^{l-1} + 2P^l - f^l}{S^l} + 1 \right\rceil$$

↓ padding ↓ filter size
 ↓ stride

analog für W

Parameter Layer l

$$n_C^l \times \underbrace{f^l \times f^l}_{\substack{\uparrow \\ \# \text{filter}}} \times n_C^{l-1}$$

↑ filter size ↑
 ↑ $\# \text{Kanäle Eingabe}$

Pooling Layer

- kompatible Zusammenfassung von Infos
↪ kombiniere Bildbereiche
- bringt Invarianz gegenüber Bildtransformationen
Robustheit Rauschen

üblich: max / avg Pooling
Stride ≥ 2

Overlapping Pooling hilft gegen Overfitting
üblich: Stride = 2, Kernel Size = 3

Input: $W^{l-1} \times H^{l-1} \times n_C^{l-1}$
Output: $W^l \times H^l \times n_C^l$

$$W^l = \left\lfloor \frac{W^{l-1} - f^l}{S^l} + 1 \right\rfloor$$

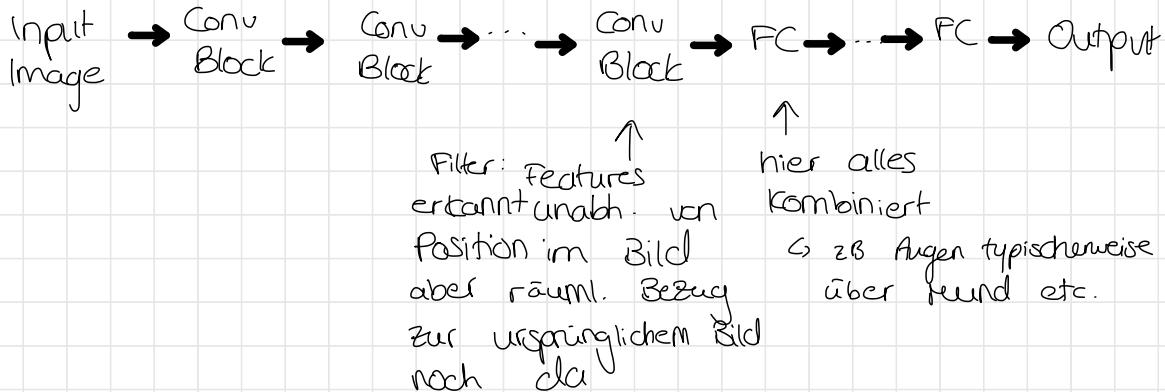
↓ bleibt gleich
 ↓ analog

Conv Net

Conv Block

Conv → Normalization → Non-linearity → Pooling → Feature maps

Net



Data Augmentation

- erzeuge mehr Trainingsdaten durch:
 - horz./vert. Flipping
 - Rotation
 - Verzerrung
 - Rauschen
 - Farbintensität (einzelner Kanäle) ändern
 - Bildausschnitte verwenden

Dropout

- = zufällig ausgehende Gewichte eines Neurons auf 0 setzen → Verbindungen kappen
- jedes Neuron muss lernen da die anderen weg sein könnten

rate = 0.5 \Rightarrow jedes Neuron nur halbe Zeit aktiv

Batch Normalization

Problem: Internal Covariance Shift

- Layer l bekommt als Eingabe die Ausgabe von Layer $l-1$
 - durch Grad Dec. ändern sich diese Werte ständig
 - je höher Learning rate desto stärker die Änderung
 - Layer l lernt langsam da sich Input ständig ändert
- Lösung: Normalisiere Eingaben auf $\mu=0, \sigma^2=1$

Alg.: Normalisiere pro Featutemap über alle Elemente eines Batches

1) Berechne empirisches Mittel μ_k Varianz σ^2_k des Batches für jede Dimension (Featutemap) einzeln

2) Normalisiere:

$$\hat{x}_k = \frac{x_k - \mu_k}{\sqrt{\sigma^2_k + \epsilon}} \quad \text{kleine konst.} \rightarrow \text{numerische Stab.}$$

3) $y_k = \gamma_k \hat{x}_k + \beta_k \rightarrow$ alle Activations auf $\mu=0, \sigma^2=1$
 ↑ ↗
 lernbare Params limitiert Netz
 \rightarrow erlaubt Reskalierung durch Netz
 \rightarrow lernt Netz selbst durch γ_k, β_k
 (β_k macht b' überflüssig)

Test-Time

μ_k, σ_k^2 nicht auf Test-Batch berechnen

↪ nutze fixe μ, σ^2 aus Trainingsdaten

↪ werden geschätzt während Training mit running average



- Layer l unabhängiger von vorherigen Layers
→ lernt unabhängiger
- höhere Learning rate möglich, da sich hidden activations weniger stark ändern
- weniger sensibel bzgl. Gewichtsinit
- zusätzliche Regularization, weniger Overfitting
↪ zusätzliches "Rauschen" in den mittleren Layers

Implementierung

einfache Additionen / Multipl. → GPUs

Datenparallelismus:

- Worker trainieren gleiches Modell
- teilen sich Gradienten, passen Gewichte an
→ effizient falls Gradienten sparse sind

Modell parallelismus:

- Worker trainieren versch. Teile des Modells auf gleichen Daten
- teilen Neuronen Aktivierungen

Neuronen Visualisierung

- Conv Layers: Filter ausgeben lassen
- andere Layer (Pooling, ...) : Top 100 Bilder mit starker Aktivierung \rightarrow mean

Feature Extraction

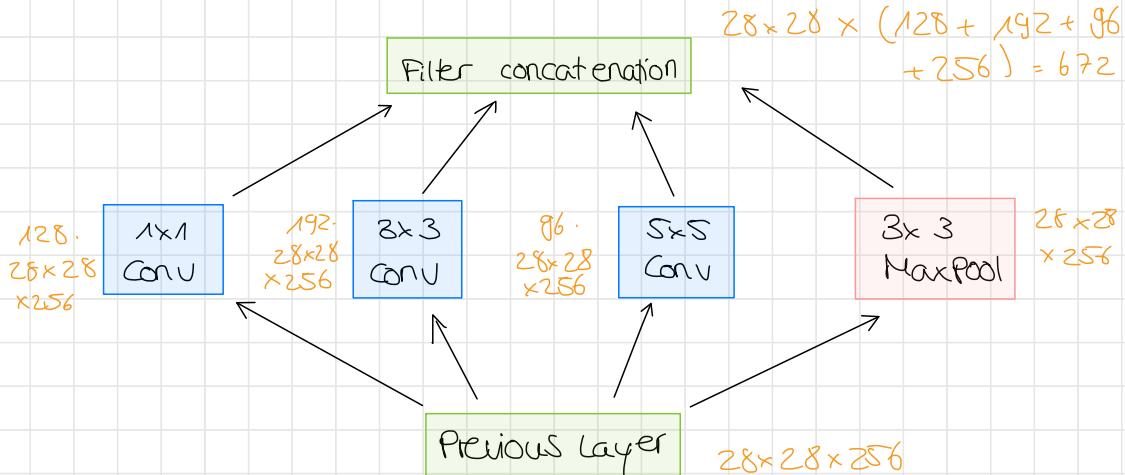
- Features = Beschreibung der Bildinhalte (Kanten, ...)
- DNN lernen automatisch gute Features
 - \hookrightarrow hierarchisch: frühe layers \rightarrow einfache Kanten, ...
mittlere \rightarrow Augen, Reisen, ...
späte \rightarrow Objekte
- letztes softmax Layer abschneiden \rightarrow Feature extractor

DeCAF: Transfer learning

- ① Trainiere NN end-to-end mit Bildklassif. (zB ImageNet)
- ② Nutze pretrained net...
 - a) für andere Klassifikations Tasks
 - \hookrightarrow letztes Layer austauschen
 - nur letztes Layer für ein paar Epochen trainieren
 - b) Feature Extractor
 - \hookrightarrow entferne letztes Layer
 - hidden values als Features

Inception Modules

Idee: parallele filter ops auf Eingabe innerhalb eines layers



Conv Ops

$$\begin{aligned} & 128 \cdot 28 \times 28 \times 256 \cdot 1 \times 1 \\ & + 192 \cdot 28 \times 28 \times 256 \cdot 3 \times 3 \\ & + 96 \cdot 28 \times 28 \times 256 \cdot 5 \times 5 \\ & = 854 \text{ Mid} \end{aligned}$$

$$\begin{aligned} & 1 \times 1 \text{ conv} \\ & 3 \times 3 \text{ conv} \\ & 5 \times 5 \text{ conv} \end{aligned}$$

- mehrere receptive fields
- 3×3 pooling
- alle output featuremaps hintereinander konkatenieren

→ Problem output size gross → computational complexity

→ pooling layer behält Größe

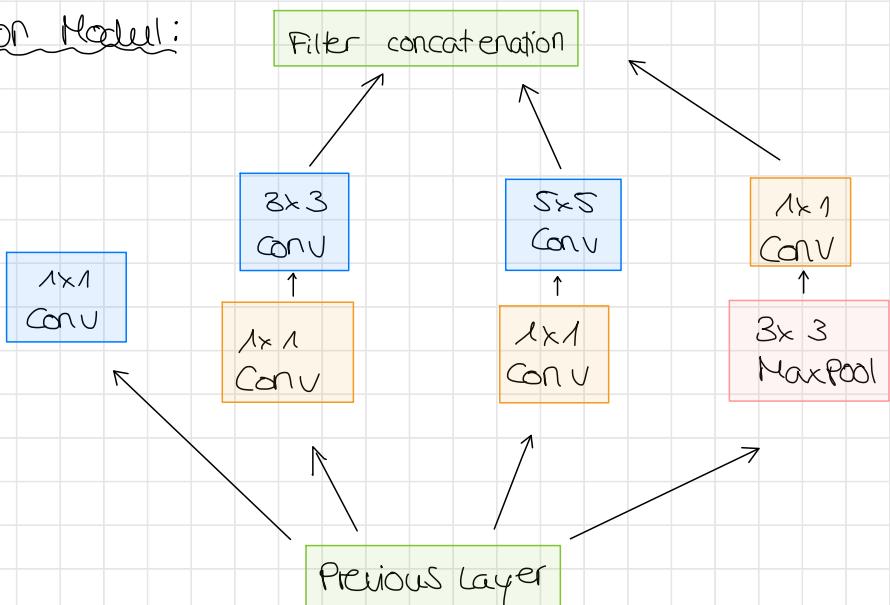
↪ je tiefer das Netz desto größere Tiefe der Daten

→ Bottleneck Layer

- 1×1 Filter zur Dimensionsreduktion der Features
- ↪ kombiniert Feature maps (Linearkomb.)

$$\begin{array}{c} 56 \times 56 \times 64 \\ \text{tiefe} \nearrow \\ * \end{array} \quad \begin{array}{c} 1 \times 1 \times 64 \\ 32 \text{ Filter} \end{array} \quad \rightarrow 56 \times 56 \times 32$$

Inception Modul:



Auxiliary Loss Layer

Vanishing Gradient:

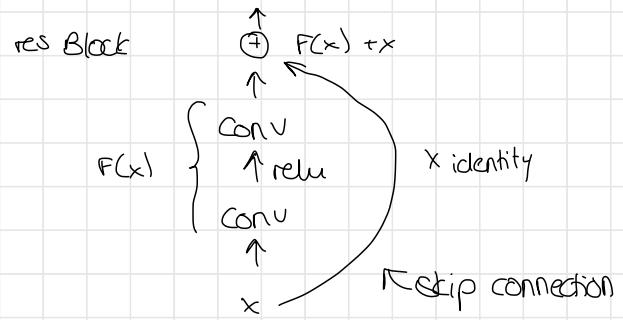
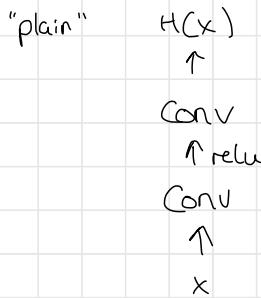
bei tiefen Netzen: kein Gradient mehr "übrig" in frühen Schichten bei Backprop.
→ lernen nicht / sehr langsam

→ flux loss layer: "Hilfsloss" an zwischen (layers) gegen vanishing Gradient
G berechne prediction an mittleren (layers)
+ LOSS → füllt Gradient bei Backprop mit rein

Residual Blocks

Problem: tiefere Modelle lernen schlechter

↳ lerne Residual mapping statt direktes Mapping



\rightarrow layers fitten $F(x) = H(x) - x$ statt $H(x)$ direkt

⊕ einfacherer Gradientenfluss

Object Detection

- mehrere Instanzen lokalisieren + klassifizieren
↳ Output: Bounding Box ($x, y, width, height$)
Klassenlabel

2 Ansätze:

a) als Regressionsproblem

- ↳ für jedes Objekt im Bild BBox + Klasse ausgeben
- ⊖ Objekte verlieren nicht bekannt
- ↳ braucht variable Ausgabe Größe
→ geht nicht

b) als Klassifikationsproblem

- Sliding Window: schiebe Suchfenster übers Bild
→ Objekt im Fenster ja/nein?
- Bildpyramide: Sliding Window auf versch. Größen
 - ⊖ langsam
- Histogram of Oriented Gradients + SVM (für Klassif.)
 - ⊕ schnell
 - ⊖ erkennt nur statische Objekte

Region Proposals

Idee: finde "unruhige" Bildregionen die vermutlich Objekt enthalten
→ reduziere Suchraum auf wenige Bildbereiche

Objectness

- = Wert der angibt wie wahrscheinlich Bildbereich Objekt enthält
- zB mit Kontendetektion → ⊕ schnell

Selective Search

- basiert auf Segmentierung
↳ hierarchische Gruppierung (bottom up)
- Bild segmentieren in homogene Bereiche
↳ iterativ Nachbarregionen mergen

Ground Truth, loss

· Ground Truth: BBoxen + Label

$$\frac{|A \cap B|}{|A \cup B|} = \text{Überlappung}$$

· BBox korrekt falls "Intersection over Union" mit Prediction $\geq 50\%$.

Recall: wie viele BBoxen von allen vorhandenen habe ich gefunden

$$R = \frac{\# \text{ gefunden}}{\# \text{ ground truth}}$$

Precision: Von den gefundenen BBoxen: wie viele waren an der richtigen Stelle

$$P = \frac{\# \text{ korrekter BBoxen}}{\# \text{ gefunden}}$$

F (F1) measure:

$$F = \frac{2 \cdot P \cdot R}{P + R}$$

Mean Average Precision:

AP: Avg der Prec in der PrecRec Kurve

Mean AP = mean der AP aller Klassen

R-CNN Region Proposal CNN

- ① generiere Region Proposals (externe Methode)
 - ② Schicke jede Region einzeln durch CNN (pretrained)
 - ③ Für jede Ausgabe aus ② (für jede Region):
 - BBox Regression
 - SVM zur Klassifizierung (eine SVM je Klasse)
-
- sehr langsam:
 - jede Region einzeln durch CNN
 - Eine SVM pro Klasse

Training:

- 1) Trainiere CNN, z.B. AlexNet auf ImageNet
- 2) Transfer Learning: reinitialisiere letztes Layer + trainiere
- 3) Trainiere classifier: SVM je Klasse
- 4) Region Proposals verbessern: Regressionsmodell
→ lerne Korrekturparam (dx, dy, dw, dh) für BBox
Input: Features der proposed regions (! nicht FC da in FC räumlicher Bezug weg)

Fast R-CNN

Idee: betrachte einzelne Regionen erst nachdem gesamtes Bild durch CNN
↪ statt alle Regionen einzeln durch CNN

- ① Bild durch CNN schicken
↪ Output: Featuremap für alle Regionen
- ② generiere ROI (externe Methode)
- ③ ROI pooling
- ④ FC layers
Linear + softmax → Klassifikation (statt SVMs)
Linear → BBox Regression

ROI pooling:

Input: Feature maps mit Größe $W \times H \times C$
Region proposals

↪ FC braucht fixe Größe $w \times h \times C$

- ① Splitte ROI in $w \times h$ Subregions
② (Max)Pooling der Subregions auf $w \times h$ Werte
③ FC layers

Training:

2 Losses: Klassifikation (neg log likel,...)
Regression (L_1 / L_2)

↪ gewichtet

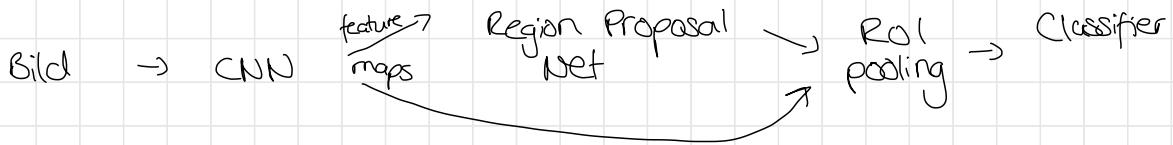
↪ damit lernen = Multi-task learning

• Kerne erst FC Gewichte, Fine Tuning: retraining CNN

- meiste Zeit verloren für region proposals
- kein End-to-end Training
- ↳ externe Region proposals

Faster R-CNN

Idee: kein externes region proposal
→ alles in einem Netz!



Region Proposal Network:

Input: Featuremaps $W \times H \times C$ aus CNN

Output: p Region proposals
je Objectness score $p \times 6$ →

- 4 für BBox
- 2 für score

- ① RPN (= fully conv net) über featurmaps schieben
- ② an jeder Position:
evaluiere Anchors auf Objectness

Anchor:

• Voredefinierte Referenzboxen = "Formen" die Objekte haben können

• auf 3 Skalierungen + 3 Seitenverhältnissen
→

• erlauben Erkennung von Objekten auf versch. Skalierungen auf den Feature maps

Loss:

- 2 losses: Regr. : regression der Anchors \rightarrow passe Anchor BBoxes an
 Class. : objectness score

\rightarrow brauche Label für objectness der Anchors

\rightarrow labeling:

- positive: größter IoU mit ground truth
 $\text{oder } \text{IoU} > 0.7$

- negative: $\text{IoU} < 0.3$

\rightarrow Rest unwichtig für Training

\rightarrow Multitask loss:

$$\mathcal{L}(\{p_i\}, \{t_i\}) = \frac{1}{N_{\text{cls}}} \sum_{\substack{\text{klass.} \\ \text{Regr.}}} \mathcal{L}_{\text{cls}} + \frac{\lambda}{N_{\text{reg}}} \sum_{\substack{\# \text{klassifk.} \\ = \text{Batch size}}} \mathcal{L}_{\text{reg}}$$

Gewichtungsfaktor

\downarrow

\uparrow

$\# \text{window positions}$

Training

- beide Netze zsm
- alle 4 losses gleichzeitig
- \rightarrow Multitask learning

4 losses:

- objectness classific.
- anchor regression
- object class classific.
- detection regression

beeinflusst

welche Features
für Detection
benutzt
werden

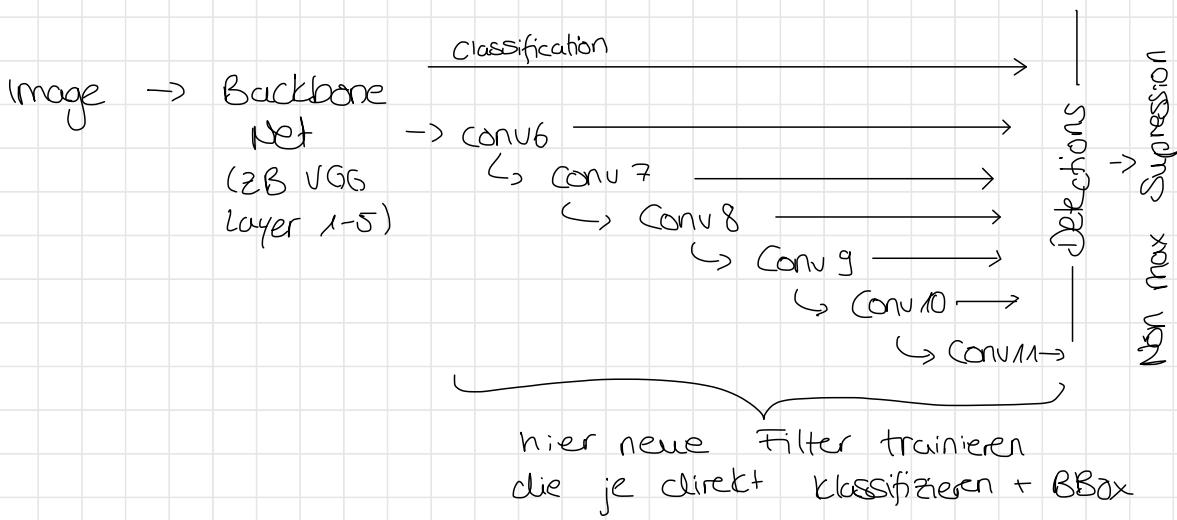
verbessert
finale Lokalisierung

SSD Detektor

Single shot Multibox Detector

Problem: Resampling auf einheitliche Größe (ROI Pooling)
langsam

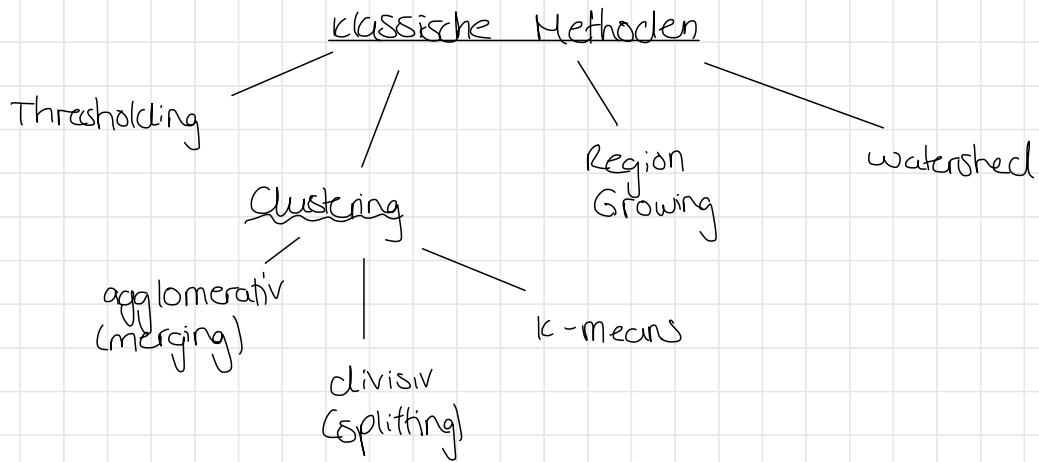
- Idee:
- nutze "default region proposals" in jeder Position einer Feature map = **Detector Element**
 - klassifiziere Objekt → box regression für jede default box
 - wende Boxen in versch. Layers im CNN an
 - ↳ auf versch. Skalierungen
 - kein Rescaling nötig
 - Spätere Schichten: grobe räuml. Auflösung
→ größerer Bildbereich abgedeckt aber räuml. Bezug noch vorhanden



- Ergebnisse mit non-max suppression reduzieren
- > keine Region Proposals
keine FC

Semantic Segmentation

- unterteile Bild in aussagekräftige Bildregionen abh. von Anwendung \rightarrow ordne pixelweise Label zu
- \rightarrow einfachere Analyse der wenigen übrigen Komponenten



Naiv: Sliding window

label je Pixel im Fenster

⊖ ineffizient: viele Forward passes
kein Reuse der Features

Fully Convolutional NNs

- keine FC layers mehr
- Conv Net + DeConv Net ($\hat{=}$ umgedrehtes Conv Net)
- DownSampling: Conv + Pooling
- UpSampling auf ursprüngliche Bildgröße

Output: $H \times W \times \# \text{ classes} = \# \text{ bekannte Klassen}$
 \uparrow $+1$ für unbekanntes

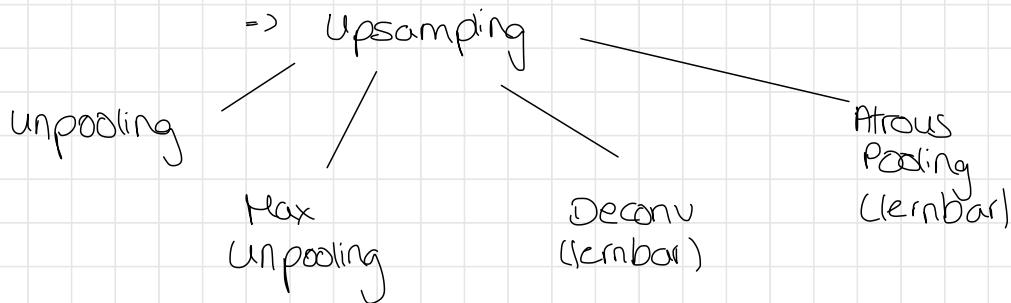
\hookrightarrow jeder Channel c zeigt wieviel dass Pixel zu dieser Klasse gehört

Upsampling

Auflösung der CNNs werden immer kleiner, \Leftarrow Originalauflösung
= immer globalere Bildrepräsentation
= Infos über immer größere Bildbereiche

ABER:

brauche pixelweise Infos am Ende



Unpooling:

a) Nearest neighbor

$$\begin{array}{c} \begin{matrix} 3 & 1 \\ 7 & 9 \end{matrix} \\ 2 \times 2 \end{array} \sim \begin{array}{c} \begin{matrix} 3 & 3 & 1 & 1 \\ 3 & 3 & 1 & 1 \\ 7 & 7 & 9 & 9 \\ 7 & 7 & 9 & 9 \end{matrix} \\ 4 \times 4 \end{array}$$

b) Bilinear Interpolation \approx reverse mean pooling

$$\begin{array}{c} \begin{matrix} 3 & 1 \\ 7 & 9 \end{matrix} \\ 2 \times 2 \end{array} \quad \begin{array}{c} \begin{matrix} 3 & 2.5 & 1.5 & 1 \\ 4 & 3.75 & 3.25 & 3 \\ 6 & 6.25 & 6.75 & 7 \\ 7 & 7.5 & 8.5 & 9 \end{matrix} \\ 4 \times 4 \end{array}$$

Max Unpooling

$$\begin{matrix} 1 & 2 & \textcolor{orange}{6} & 3 \\ 3 & \textcolor{blue}{5} & 2 & 1 \\ 1 & 2 & 2 & 1 \\ 7 & 3 & 4 & 8 \end{matrix}$$

4×4

$$\rightarrow \begin{matrix} 5 & 6 \\ 7 & 8 \end{matrix}$$

$\rightarrow \dots \rightarrow$ rest vom
NN

2×2

rest vom

$$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix}$$

2×2

$$\begin{matrix} 0 & 0 & \textcolor{orange}{2} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \textcolor{green}{3} & 0 & 0 & 4 \end{matrix}$$

4×4

Max Pooling:
merke Pos
der Max Werte

----->

Max Unpooling:
füge Werte an
den Pos. ein

Matche Layer
gleicher Auflösung

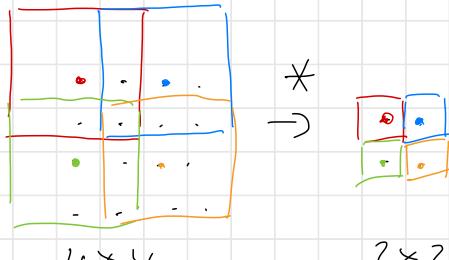
Transposed Convolution lernbar!

Convolution

3×3

$S=2$

$P=1$

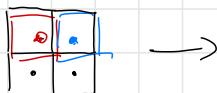


Deconvolution

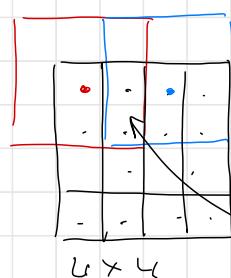
3×3

$S=2$

$P=1$



← transponierter
Faltungskernell
S Werte lernen
durch Backprop



Summe an
Überlappung

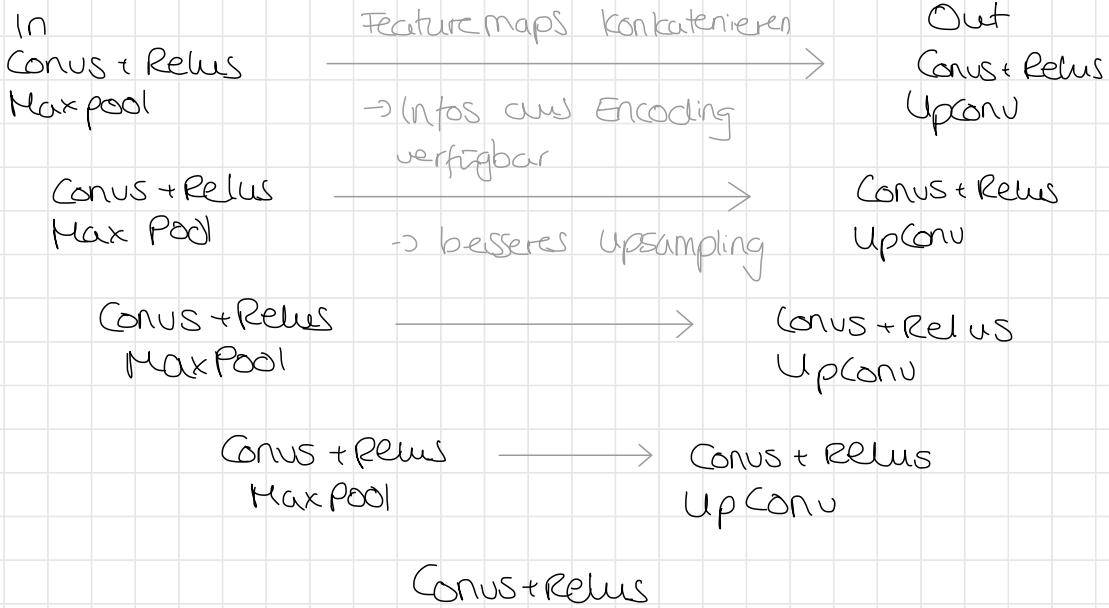
Skip Connections:

Upsampling verbessern: nutze Infos aus entsprechenden Layern des Downsamplings

↪ Skip connections

U-Net

- für medizinische Bilder
 - pretraining auf ImageNet bringt nichts
- kopiere Features aus Downsamplingpfad zu entspr. Layer im Upsampling
→ konkatenieren



Deeplab

Atrous Convolutions - Dilated Convs Deeplab V1

3 Wünsche:

1. bearbeite größeres rezeptives Feld mit 3×3 Convs

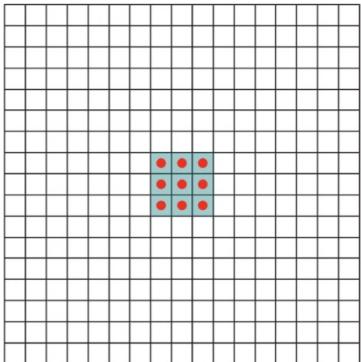
-> Größe vom rezeptiven Feld soll so sein wie mit
 $7 \times 7 / 9 \times 9 / 11 \times 11$ Filtern

2. Anzahl Params gleich wie ein 3×3 Layer

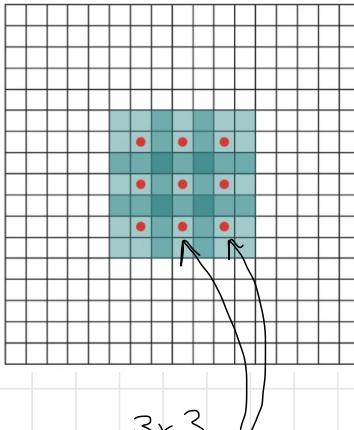
3. Stride = 1 nutzen damit Auflösung gleich bleibt

=> Werte 3×3 Filterkernels in größeren Abständen auf

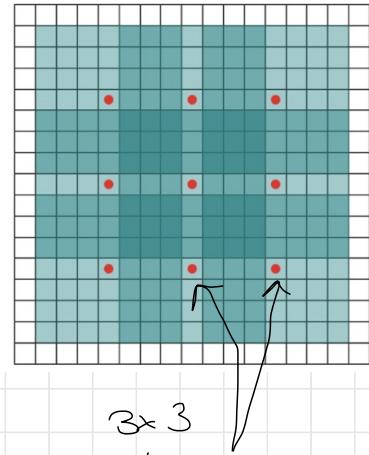
1 dilated Conv



2 dilated Conv



4 dilated Conv



3×3

rate = 1

= normaler Conv

3×3

rate > 2

3×3

rate = 4

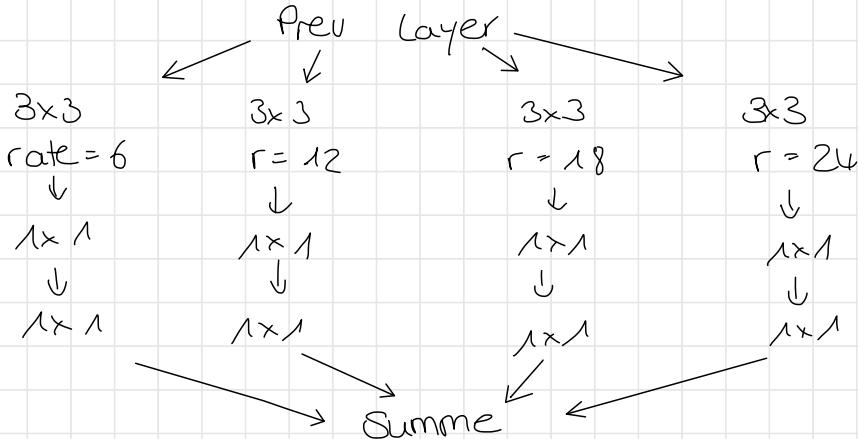
• klassische Pipeline: Auflösung wird reduziert durch Pooling oder $\text{Stride} > 1$

• Deeplab: erst ein paar Standard Convs, ab Wunschauflösung nur noch Dilated Convs
→ Sprungrate erhöhen damit Auflösung gleich bleibt

Atrous Spatial Pyramid Pooling DeepLab V2

= Atrous Convs + Inception Module

→ parallele Ausführung mehrerer Atrous Convs mit versch. rates in einem layer



- (-) große Sprungrate (zB z4) + kleine Auflösung (32×32)
 - ↳ brauche \uparrow (Zero-) Padding damit Auflösung gleich
 - ↳ Filter betrachten nur noch Mittleren Wert da Rest schon im Padding, also 0
 - ist dann quasi 1×1 Filter
 - kein räumlicher Kontext mehr

GAP - Features

DeepLab V3

Lösung: Global Average Pooling auf allen vorherigen Feature Maps

- berechne je Feat. Map Mittelwert
- weitergeben an 1×1 conv + upsampling

Conditional Random Fields

- Iteratives Postprocessing Verfahren
- glättet verwischte Segmentierungen
- Anwendung nach Netz-Ausgabe
- gehört nicht zum Training

Optimiere Energiefunktion

$$E(x) = \underbrace{\sum_i \Theta_i(x_i)}_{= -\log P(x_i)} + \underbrace{\sum_{ij} \Theta_{ij}(x_i, x_j)}_{\substack{\text{Nachbarpixel haben oft} \\ \text{gleiches Label} \\ \rightarrow belohne ähnliche \\ \text{Nachbarpixel}}}$$

\approx Konfidenz dass
Label :

Sem Seg Evaluation

Accuracy:

$$= \frac{TP + TN}{TP + TN + FP + FN} = \frac{\# \text{ korrekt}}{\# \text{ gesamt}}$$

(-) $\# TN$ wird überbewertet falls viele negative im Vergleich zu $\# TP$
zB viel Hintergrund (TN), wenig Kuh (TP)

Mean IoU:

$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|} = \frac{TP}{TP + FP + FN}$$

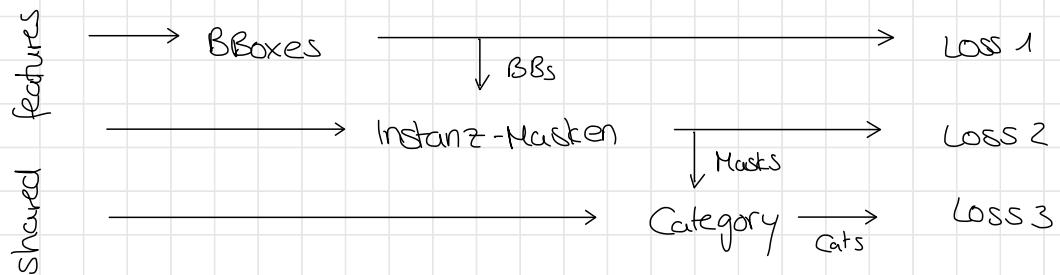
berechne IoU pro Klasse, dann Mittelwert

Dice Coefficient / F1 score:
oft bei medizinischen Bildern

$$DSC(A, B) = \frac{2 |A \cap B|}{|A| + |B|} = \frac{2TP}{2TP + FP + FN}$$

Instance Segmentation

- ordne Pixel Label und Instanz zu
- Multi-Task Network:



- 1) Regression BBoxes
In: conv Features
Out: BBoxes

Layer 1: BBox Locations
Layer 2: Objectness Score

2) generiere Instanz-Masken

= pixelweise Zuordnung zu Instanzen, ohne Label

In: Conv Features

BBoxes

Out: Instanz-Maske pixelweise

jede BBox einzeln durchs Netz propagiert

3) Klassifizierung

In: Conv Features

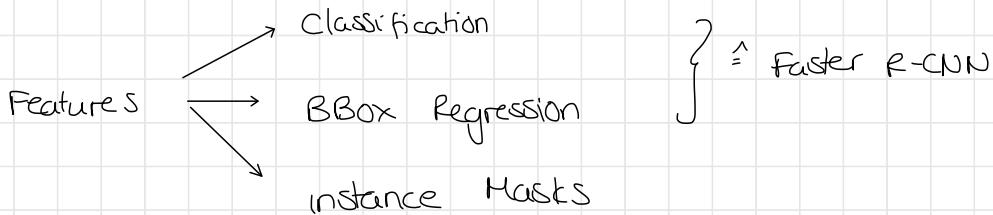
Masken

Out: Instanz-Segmentierung + Klassenlabel

Ordne jeder Instanz Klassenlabel zu

Mask R-CNN

= erweitertes Faster R-CNN



Roi Align

Problem Roi Pooling:

- NN gibt floats aus für BBoxes
- Roi Pooling runden auf int \rightarrow Koordinaten werden quantifiziert

\rightarrow Informationsverlust

\rightarrow Lösung: bilineare Interpolation statt Quantifizierung

\rightarrow lege Raster über Bild, finde 4 nächsten Rasterrecken

\rightarrow bilin. Int. der 4 Punkte

\rightarrow avg/max Pooling dieser Werte

Loss

3 Losses:

- Classification loss : cross entropy L_{cls}
- BBox loss : diff. zw. ground truth, pred L_{Box}
- Mask loss : cross entropy zw. GT, pred L_{mask}

$$\Rightarrow \mathcal{L} = L_{\text{cls}} + L_{\text{Box}} + L_{\text{mask}}$$

→ End 2 End Training

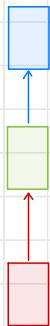
Recurrent Neural Networks

- zur Verarbeitung / Ausgabe von Datenstreams
→ Sequenzen von Daten
- innerer Zustand h_t abhängig von Eingabe x und h_{t-1}

RNN Modelle

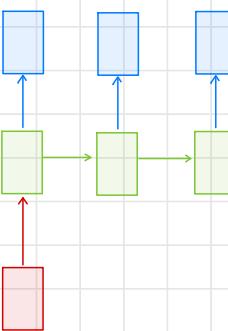
one to one

= standard MLP /
CNN



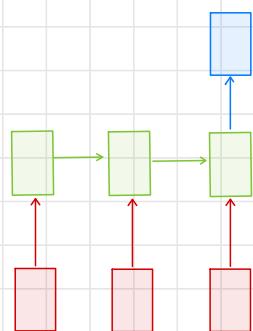
one to many

Image Captioning



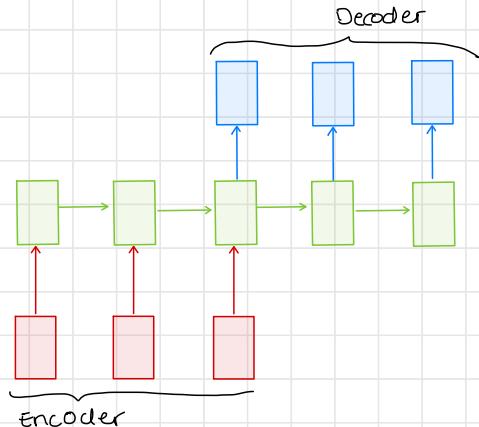
many to one

Action Classification



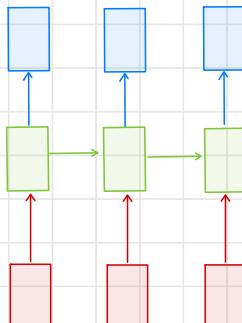
many to many

Machine Translation



many to many

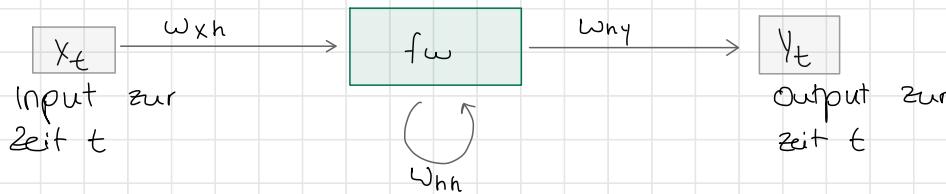
Frame level video Classification



Vanilla RNN

- hidden vector h
- output y ist abh. von h

ω_{xh} } geteilt für alle t
 ω_{hh} } Lernen durch
 ω_{hy} } Backprop



$$h_t = f_w(h_{t-1}, x_t) = \tanh(\omega_{hh} h_{t-1} + \omega_{xh} x_t)$$

$$y_t = \omega_{hy} h_t$$

Training

D = vocab size

H = hidden size = # hidden neurons

n = sequence length

① param init:

W : zufällig mit Gauß-Verteilung
 b : Nullen

$$\omega_{xh} \in \mathbb{R}^{H \times D}$$

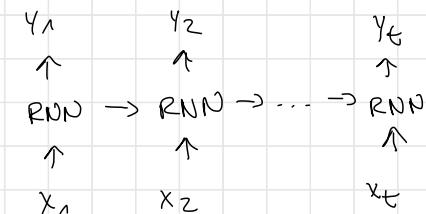
$$\omega_{hh} \in \mathbb{R}^{H \times H}, b_h \in \mathbb{R}^H$$

$$\omega_{hy} \in \mathbb{R}^{D \times H}, b_y \in \mathbb{R}^D$$

② Netzwerk über Zeit ausrollen
 · Fwd pass berechnen

· Output berechnen: Prediction scores
 + softmax

· Loss: Neg Log Lik. für jedes x_i



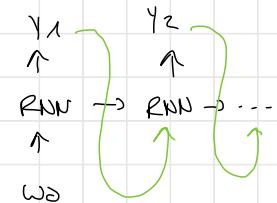
(abh. von Wortrepr!
 wzw: l_1/l_2 , OHE: CrossEntropy)

- ③ Backprop: Gradienten über alle vorherigen Zeitschritte zurück
 → spätere Losses haben größeren Einfluss
- Update Modellparams

Testing: Sampling

Seed = 1. Input w_0

→ berechne Output Scores y_t + softmax
 wähle char mit größter w_{kt}
 → als neue Eingabe w_t ins Netz



Vanishing / Exploding Gradients

$$\text{Gradient} \approx dh_{t-1} = W_{hh}^T \cdot dh_t \quad \text{State wird multiplikativ geändert}$$

↪ beim ausrollen auf große Länge (> 20):
 $W_{hh} < 1 \rightarrow$ Gradient wird 0
 $> 1 \rightarrow$ Inf

=> Lernen unmöglich!

Parameter:

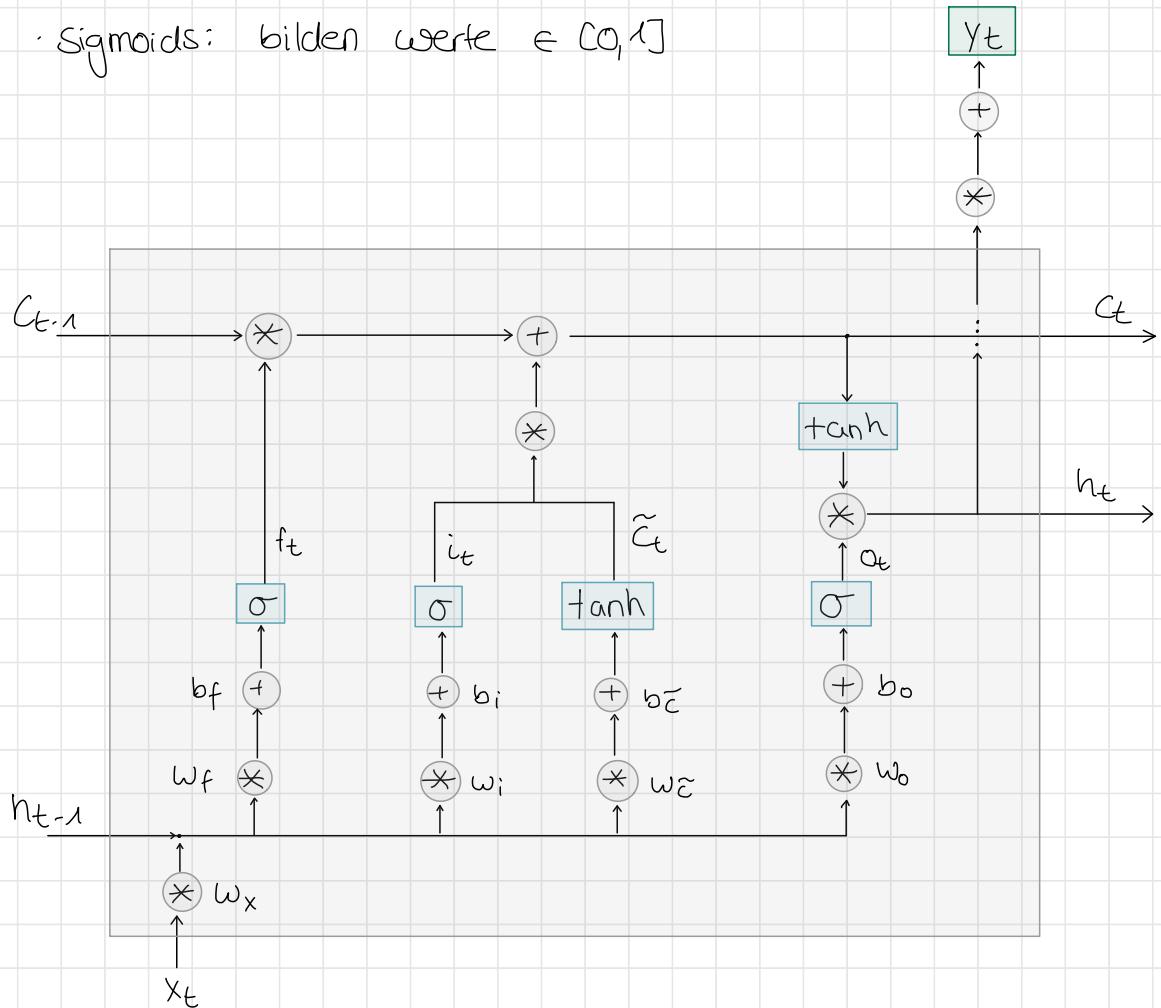
$H = \# \text{ hidden units} \rightarrow H \times 2H \text{ params}$

LSTM

→ ändere state additiv → + keine Gradientenprobleme
- hohe Komplexität des Modells

· 2 hidden states: c_t, h_t

· Sigmoids: bilden Werte $\in [0, 1]$



Forget gate: wie viel vom alten c möchte ich vergessen
 $f_t = \sigma(w_f \cdot [h_{t-1}, x_t] + b_f)$

Input gate: was aus dem proposal \tilde{c} wird übernommen
 $i_t = \sigma(w_i \cdot [h_{t-1}, x_t] + b_i)$

proposal: $\tilde{c}_t = \tanh(w_c \cdot [h_{t-1}, x_t] + b_c)$

Update c: $C_t = \underbrace{f_t * C_{t-1}}_{\text{vergesse Teile des alten Hems}} + \underbrace{i_t * \tilde{c}_t}_{\text{verknüpfe neue Hems}}$

output gate: was soll alles in neuen state h_t
 $O_t = \sigma(\omega_o [h_{t-1}, x_t] + b_o)$

update h : $h_t = O_t * \tanh(c_t)$

Parameter:

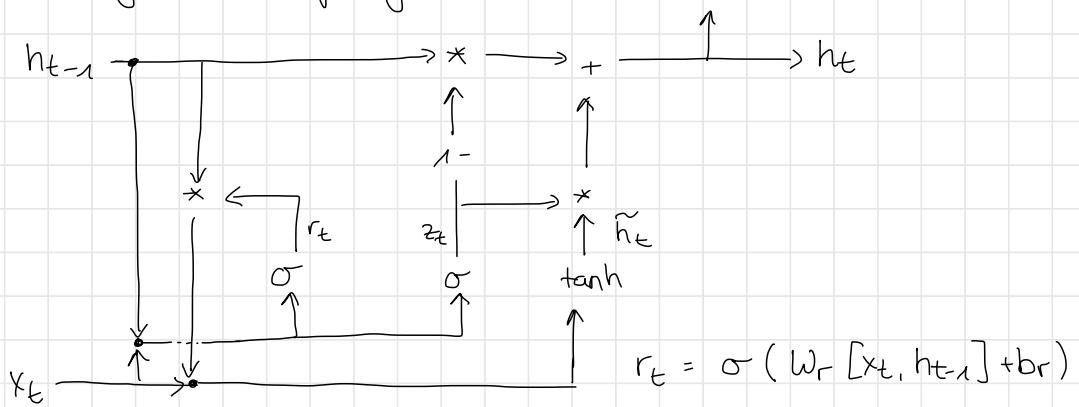
$H = \# \text{ hidden units} \rightarrow 4H \times 2H \text{ params}$

GRU Gated Recurrent Units

ähnliche Performance wie LSTM

Params : $3H \times 2H$

reset gate + output gate



$$r_t = \sigma(\omega_r [x_t, h_{t-1}] + b_r)$$

$$\tilde{h}_t = \tanh(\omega_{xh} x_t + \omega_{hh} (r_t * h_{t-1}))$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Word Embedding

One - Hot - Encoding

- $D = \text{len}(\text{vocab})$

- repräsentiert Wort / Buchstabe als D -dim Vektor mit einer 1, Rest 0

- \ominus großes Vokabular \rightarrow riesiger Vektor
- keine Semantik repräsentiert

Word2Vec

LOSS: L1 / L2

- lerne semantische Darstellung im Vektorraum
 $\sim \|v(\text{zebra}) - v(\text{pferd})\| < \|v(\text{zebra}) - v(\text{bus})\|$

(1) supervised: zu teuer, subjektives Labeling

(2) unsupervised: lerne basierend auf großem Textkorporus
 \rightarrow lerne Semantik basierend auf Wortbezug in Texten

Skip-Gram Modell

- Wörter im gleichen Kontext haben ähnliche Semantik

- Fully Connected MLP

- Input: 1 Wort als One-Hot vector

- Output: benachbarte Wörter

- Embedding Layer: bottleneck mit Size 100 - 1000

Training:

- auf zB Wikipedia
- lerne Emb Layer

\rightarrow Nach Training: verwende Emb Layer repr. als Wortrepräsentation

Skip-Thoughts \equiv Skip-Gram für Sätze

Image Captioning

- generiere Beschreibung des Bildinhalts

Image \longrightarrow CNN \longrightarrow RNN

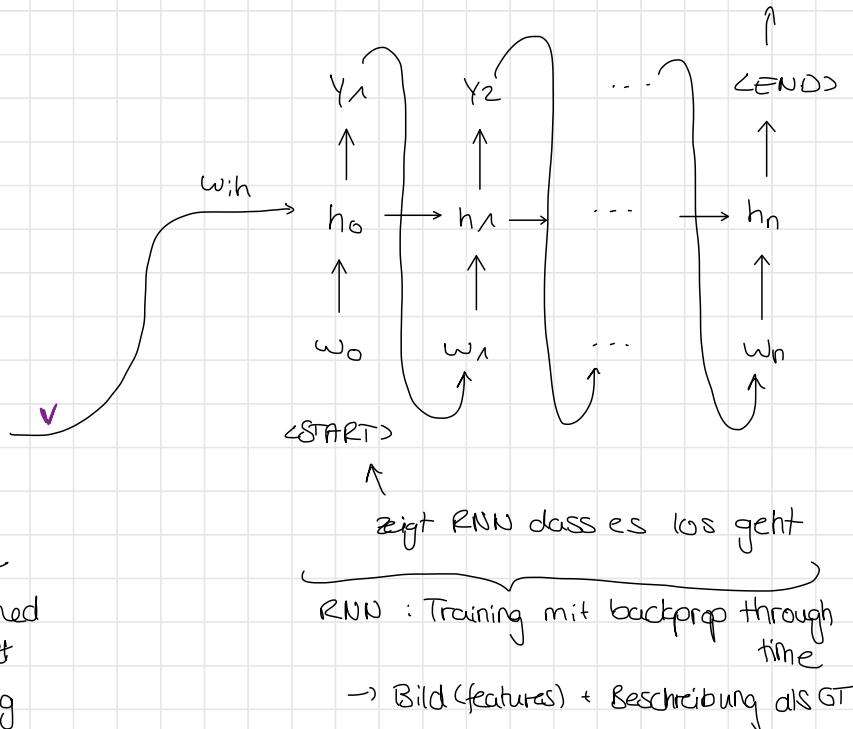
VGG CNN + RNN

Abgabe durch
Encoder Token
beenden

Image
Conv
Conv
Maxpool
 \vdots
Conv
Maxpool
FC 4096
FC 4096
~~FC 1000~~
~~Softmax~~

V

VGG pretrained auf ImageNet ggf. fine-tuning



$$v = \text{Bildrepräsentation} \quad v \in \mathbb{R}^{4096}$$

$$h_0 = \tanh(\omega_{xh} x_0 + \omega_{vh} v)$$

Metriken

nutze Metriken auch machine translation

↪ wie gut passt Hyp zu Ref? subjektiv!

BLEU score Bilingual Evaluation Understudy

misst Precision

wie viele n-Gramme aus Hyp finde ich in Ref

- keine Betrachtung der n-Gramme die NICHT in Hyp sind
- favorisiert kürzere Sätze → keine Semantik

BLEU-4 für "gutes Englisch" benutzt

METEOR score Metric for Evaluation of Translation with Explicit Ordering

Reihenfolge der Wörter berücksichtigt

favorisiert längere Sätze

$$\text{Meteor} = (1 - \text{mapping_penalty}) \cdot \text{Fmean} = (1 - \text{map_pen}) \cdot \frac{10 \cdot \text{P.R}}{\text{R} + \text{gP}}$$

⇒ am besten BLEU + METEOR

Deep Visual Semantic Alignments

Architektur verwenden um Bildbereiche zu beschreiben

Image → R-CNN → ordne Satzteile den Regionen zu



bidirectional RNN



Sentence Embeddings aus Beschreibung

Attention Modelle

Attention: Menschen nutzen hervorstehende Bildbereiche als "Orientierung"

-> Ziel: lerne worauf Fokus zu legen ist

Translation:

- merke frühere Worte bei langen Sätzen
- ↳ generiere nächstes Wort basierend auf vorherigen Wörtern + Kontextvektor

$$p(y) = \prod_{t=1}^{T_2} p(y_t | \underbrace{y_1, \dots, y_{t-1}}_{\text{Context vector}}, c)$$

Kontextvektor ist letzter hidden Zustand nach Encoder Teil

$$c_t = \sum_{j=1}^{T_1} \alpha_{tj} h_j$$

α_{tj} gibt an wie wichtig Input j für Output t

wobei $\sum_j \alpha_{tj} = 1 \rightarrow \alpha$ ändert sich je nach Input zum Zeitpunkt j

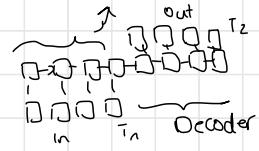
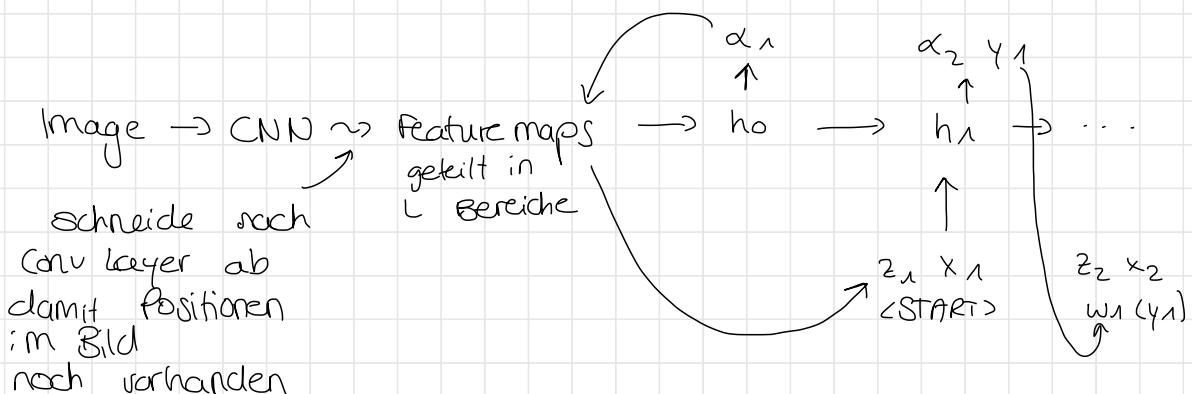


Image Captioning

- berechne Softmax der angibt wo im Bild Fokus liegt
- lerne auf welchen Bereich sich Netz wann fokussiert



y = Verteilung über Vokabular

a = Verteilung über L Bildbereiche (a_i = Bereich i)

$\alpha_{t,i}$ = softmax (attention-modell (a_i , h_{t-1}))

z_i = wie stark sollen Bereiche gewertet werden bei Vorhersage des nächsten Worts

↪ hard attention: nur $\max\{\alpha_{t,i}\}$ → also Bereich mit max Wert

soft attention: gewichtet alle $\sum_{i=1}^L \alpha_i a_i$

Video Captioning

· generiere videobeschreibung

Ansatz 1:

- jedes Frame durch CNN
- mean pooling ↗ ein Descriptor für alle frames
- RNN

Ansatz 2:

- Encoder / Decoder RNN (Seq-to-Seq)
- erst jedes Frame in Encoder geben
- dann Decoder Teil der Beschreibung ausgibt

Visual Question Answering

Bild + Frage → Antwort

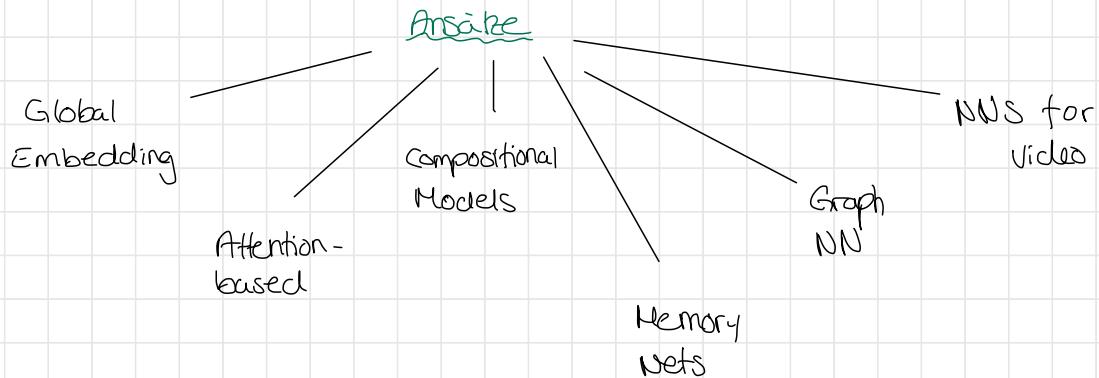
- QA beantworten → bessere Als für andere Anwendungen
- Mensch - Robo - Kollaboration: Robo erkennt was Mensch macht

Arten

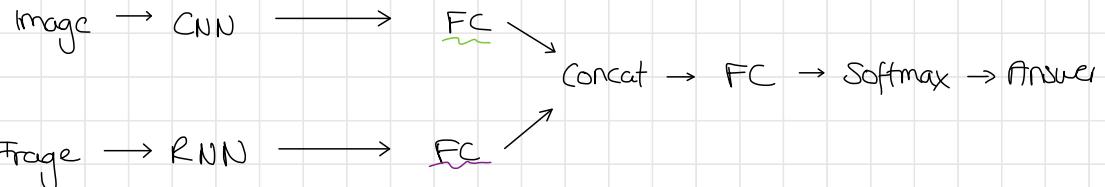
- Open - Ended: alle Wörter aus Vokabular für Antwort zur Verfügung
In: Bild + Frage
Out: Antwort
- Multiple - Choice:
In: Bild + Frage + Aw - Möglichkeiten
Out: Zahl / Buchstabe ↑

Evaluation

Accuracy über alle Fragen



Global Embedding



CNN (zB VGG)

- pretrained auf ImageNet
- letzte Softmax Schicht abschneiden \rightarrow 4096 Vektor
- FC macht daraus 1024 dim Bildrepräsentation

RNN: (zB 2layer LSTM)

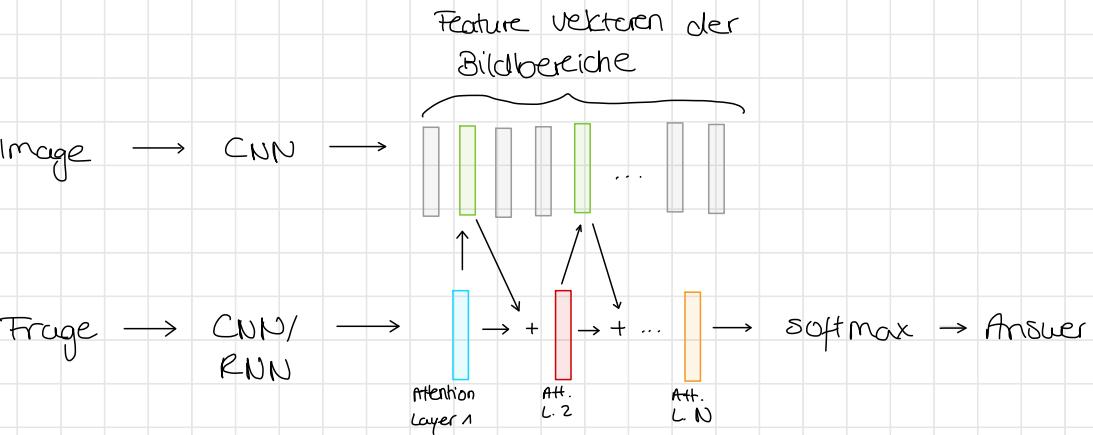
- end-to-end trainiert
- FC kombiniert c_t, h_t zu 1024 dim encodierter Satz

Concat:

- Fusion Bild + Frage
- zB durch Konkatenation, Addition, Mult., ...
- FC mit # möglicher Antworten erzeugt Antwort

Stacked Attention Network

- finde mit Frage interessante Bildbereiche



Fragen Embedding:

- CNN als LSTM Alternative

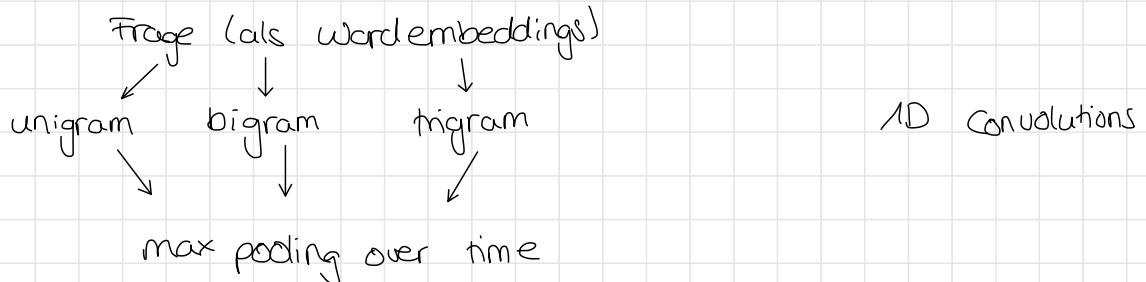


Bild Embedding:

- zB pretrained UGG
- abschneiden nach Conv → Bereichsbezug noch vorhanden
 - ↪ $w \times h \times \text{dim}$ feature maps
 - ↪ vector repräsentation $1 \times 1 \times \text{dim}$ für alle $w \cdot h$ Bereiche

Ablöse:

- N Mal (für N attention layers):
 - nutze aktuellen State um Bildvektoren abzufragen
→ in Layer 1 ist das einfach die Frage
 - gibt Attention distribution
→ wie wichtig ist in diesem Layer welcher Bildbereich
 - kombiniere aktuellen State mit Bildvektoren zum neuen State
- mit N^{th} state Frage beantworten

Compositional Models

= Neural Module Networks

- zersteile Frage in modulare Teile
↳ trainiere spezifische NNs für diese Teile

→ NNs für

- find : image \rightarrow attention
- transform : attention \rightarrow attention
- combine : attention \times attention \rightarrow attention
- describe : image \times attention \rightarrow label
- measure : attention \rightarrow label

- baue mit parser die NN chains aus den Fragen im Trainings-
set

→ trainieren der Chain mit Antworten

teile Gewichte zw gleichen Funktionen + Argumenten
→ find (red) überall gleich
find (blue) aber andere Gewichte



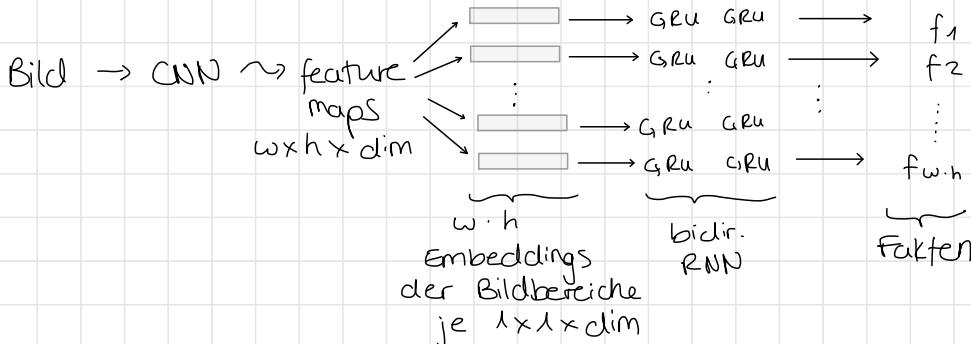
- abhängig von Parser

→ Netz kann Struktur auch lernen

↳ reinforcement learning statt backprop richtig

Dynamic Memory Networks

· Lesen + schreiben aus Memory



(Fragenmodul analog)

- Memory initialisiert mit Fragenrepräsentation
- in jedem Schritt: lesen + schreiben in Mem
- Output: Memory concatenated mit Frage

Graph Neural Networks

- relationale Fragen beantworten

- Feature maps nach CNN: $w \times h \times \text{dim}$
↳ feature vektoren des Bilds: $w \cdot h$ viele $1 \times 1 \times \text{dim}$
- immer 2 dieser Vektoren werden mit Fragenembedding konkateniert
- Re-Embedded mit MLP
- Summe bilden über alle VCs
- FC layer generiert Antwort

Video Question Answering

- Modelle müssen relevante Frames rausfinden

Problem: einige Modelle ignorieren Video, lemen Antworten zu "raten" basierend auf Trainingsdaten
zB semantische Ähnlichkeit zw. Frage + Aw

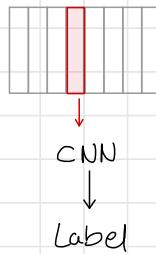
Activity Classification

• Klassifikation von actions in Video Stream

CNN + Temporal Connectivity

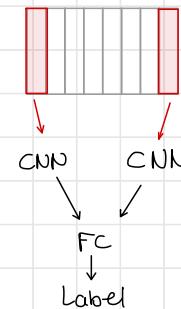
Single Frame

- keine zeitliche Info
wähle 1 Frame aus Video
→ CNN → Klassifikation



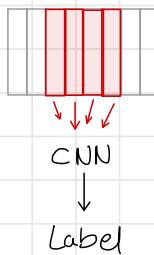
Late Fusion

- 2 Frames mit n frames Abstand
• letzte Layer haben zeitl. Info



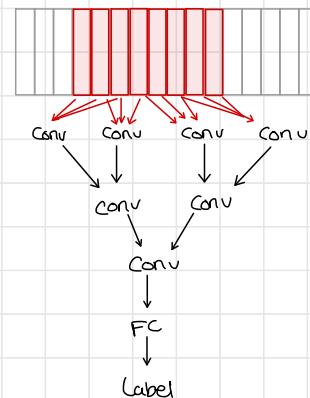
Early Fusion

- frühe Conv Layer haben zeitl. Info
• Filter der Größe $11 \times 11 \times 3 \times T$
 ↑
 # frames



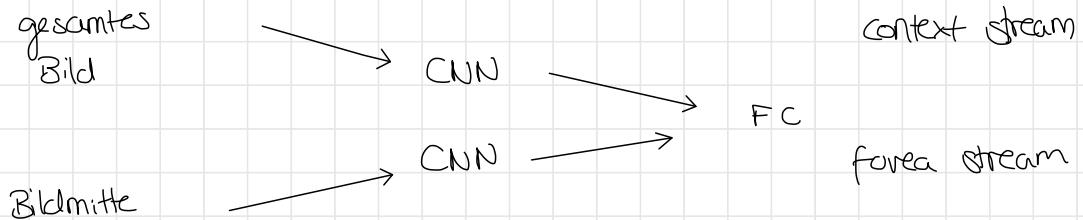
Slow Fusion

- Spätere Layer haben größerer zeitl. Kontext
↳ Bewegungsmuster auf versch. Hierarchien

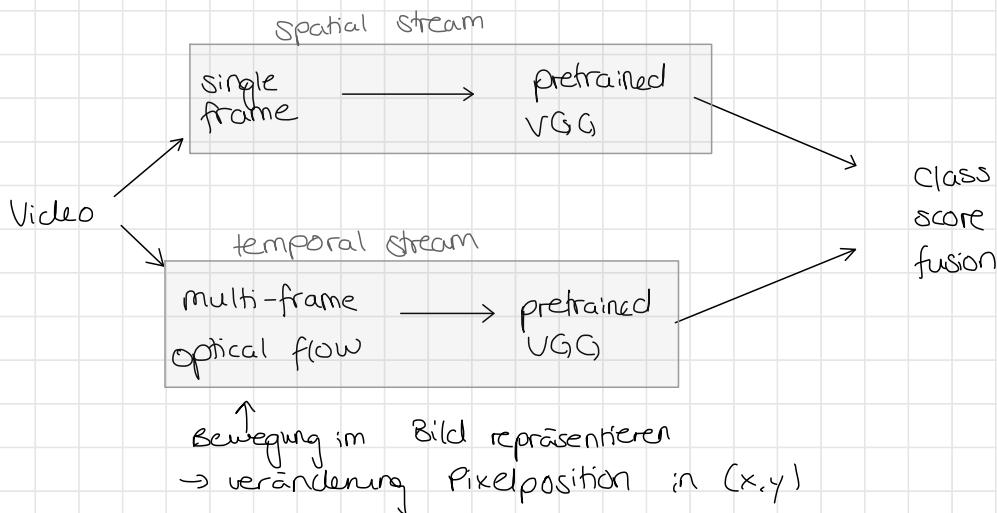


Multiresolution CNN

- schnelleres Training durch Reduzieren der Eingabegröße
- Annahme: relevante Aktionen in Bildmitte



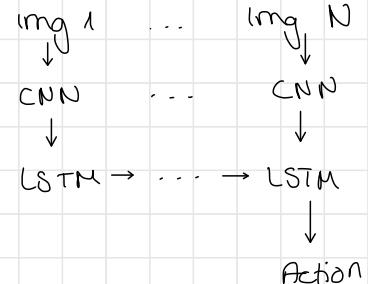
Two-stream ConvNets



- kein end-to-end Training

CNN + LSTM

- Training: Output pro timesteps für Loss
- Testing: Output last frame = prediction
- Retrained auf ImageNet



3D Convolutional Nets

3D conv in Raum-Zeit : frames stapeln

Input: $H \times W \times L$ $L = \# \text{frames} \cdot \text{channels pro frame}$
Filter: $K \times K \times d$ $d < L$
Output: 3D Tensor

Probleme

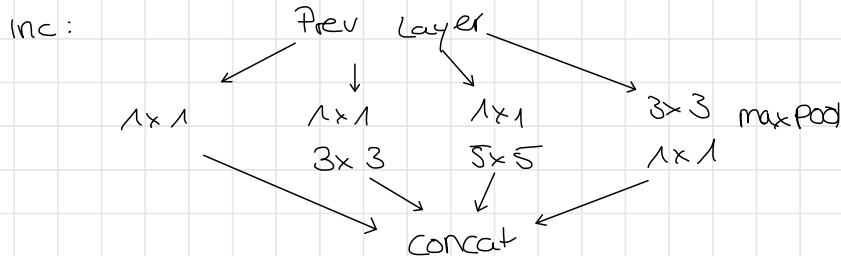
1. zu wenige Trainingsdaten für EZ-E Training
2. Vertraining auf ImageNet = "falsche" Aufgabe
3. keine "schlauen" temporalen Modelle

Inflated 3D CNN

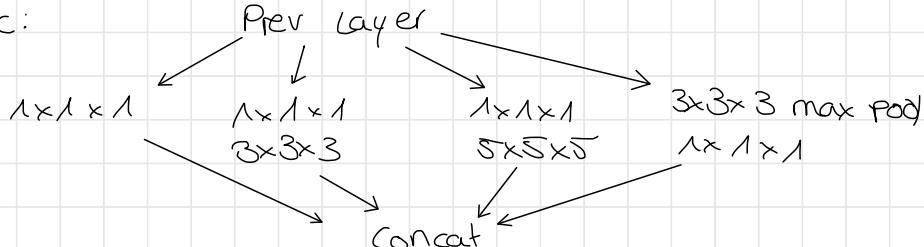
Wissenstransfer aus Image Recognition \rightsquigarrow 3D CNN

Trainiere 2D Inception Modul auf Image Net

\hookrightarrow 3D Inception Modul: Filter n mal hintereinander



3D Inc:



!3D

= state of the art
 \approx 3D version von GoogleNet

Pre-Training

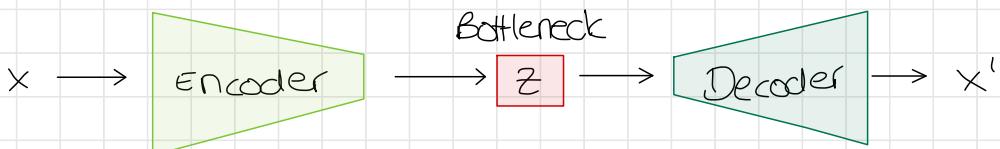
- pre-training auf ImageNet + Video dataset bringt viel v.a. bei 3D conv
- wichtiger als dynamic / long-term context!

Self-Supervised Learning als Pre-training

- Labels von Daten ggf. teuer/unmöglich z.B. Autounfälle?
 - Segmentierungen / Posen schwer zu annotieren → human error
↳ unklare medizinische Bilder?
 - Daten ist eigene Supervision
≈ lerne mit Proxy-Aufgaben
- Rekonstruktion Kolorisation Context/
 zeitliche
 Position Abfolge ...

Reconstruction

- Autoencoder = Encoder / Decoder mit Bottleneck layer
- Aufgabe: lerne Identität
 - x' möglichst gleich zu x
 - Netz lernt kompakte Repräsentation von x



- + · alle Labels möglich! Input = Label
· kein setup nötig
- · schwer für große Bilder mit vielen Details
· proxy loss kann oft realen Sachverhalt schwer trainieren

Colonization

- = Autoencoder aber label \neq input
- In: greyscale Bild
out: Farbbild
- \rightarrow Netz muss sinnvolle Repräsentation der Farben lernen

Kontext / Position

- = Puzzleproblem
- 2 Bildbereiche samplen aus 3×3 Grid
 \rightarrow Netz muss sagen wo 2. Bildbereich zu 1. gehört
= 8 Klassen Classification task
- Lücken + Jitter einfügen damit Netz nicht einfache Features matched (Kanten,...)

zeitliche Abfolge

- kurze Clips mit viel Motion
- \rightarrow positive: Clips in richtiger Reihenfolge
negative: Clips mit Frames geshuffelt

Generative Adversarial Networks

- Ziel: geg. Trainingsdaten generiere neue Daten aus gleicher Verteilung

Density Estimation

- explizit: definiere $p_{model}(x)$ und trainiere darauf
- implizit: trainiere Modell das aus $p_{model}(x)$ Samplen kann ohne $p_{model}(x)$ zu definieren

Implicit Generative Models

- trainiere Netz G das die Verteilung $p_{model}(x)$ erzeugt
 $\rightarrow p_{model}(x)$ soll nahe an $p_{data}(x)$ sein
(prüfen durch Sampling)

Input: code vector $z \rightarrow$ gesampelt aus einfacher, fester Verteilung; jede Dim eine Zahl

Output: $x = G(z)$
differenzierbare Funktion die z auf $x \in \text{Data Space}$ abbildet
 \uparrow
ZB $\mathbb{R}^{w \times h \times 3}$
für RGB Bilder

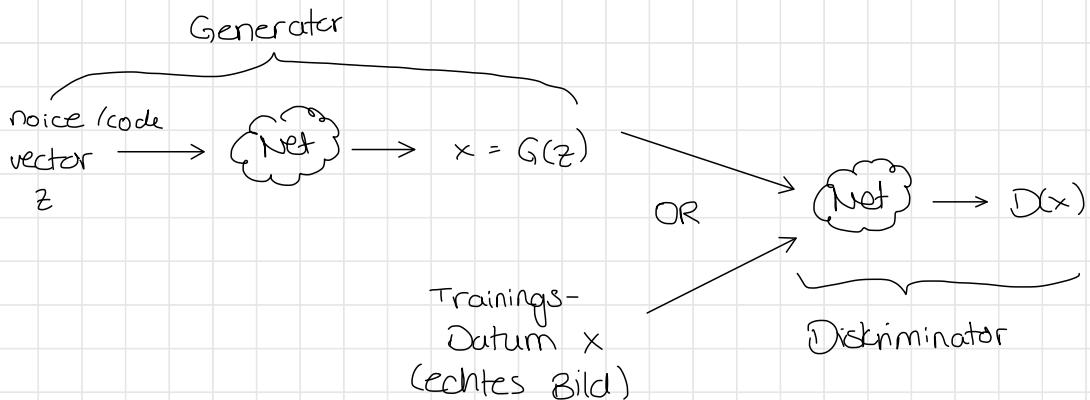
$z \rightarrow \text{Net} \rightarrow x = G(z)$

GAN

- Trainiere 2 Netze:
 - Generator G : versucht realistische Samples zu erzeugen
 - Diskriminator D : versucht Bilder von G und Bilder aus Trainingsdaten zu unterscheiden

$\rightarrow G$ versucht D auszutricksen

- Output von D : Wahrscheinlichkeit dass $G(z)$ echt
 $\hookrightarrow [0, 1]$



Training

G, D gemeinsam trainieren \rightarrow min max Spiel

$$\min_{\Theta_g} \max_{\Theta_d} \left[\underbrace{\mathbb{E}_{x \sim p_{\text{data}}} \log D_{\Theta_d}(x)}_{\text{Ausgabe von } D \text{ für echtes Bild}} + \underbrace{\mathbb{E}_{z \sim p(z)} \log (1 - D_{\Theta_d}(G_{\Theta_g}(z)))}_{\text{Ausgabe von } D \text{ für fake Bild von } G} \right]$$

$\rightarrow D$ maximiert sodass $D(x) \approx 1, D(G(z)) \approx 0$

\hookrightarrow gradient ascent

G minimiert

\hookrightarrow gradient descent

} alternierend
trainieren

Problem: G wird anfangs sehr schlechte Beispiele generieren

\rightarrow Gradienten nahe 0

\rightarrow langsames Lernen

\rightarrow ändere objective von G :

statt Wkt dass D korrekt zu minimieren $\min_{\Theta_g} \mathbb{E}_{z \sim p(z)} \log (1 - D_{\Theta_d}(G_{\Theta_g}(z)))$

Maximiere Wkt dass D falsch $\max_{\Theta_g} \mathbb{E}_{z \sim p(z)} \log (D_{\Theta_d}(G_{\Theta_g}(z)))$

\leadsto gleiches Ergebnis, besseres Training

Algo

- for num epochs : $k=1 / k>1$ unklar
- for $k \leftarrow$ steps :
- sample minibatch mit m noise vectors $\{z^1, \dots, z^{m^2}\}$ aus noise distribution $p_g(z)$
 - sample minibatch mit m Beispielen $\{x^1, \dots, x^{m^2}\}$ aus Data generating distribution $p_{\text{data}}(x)$
 - gradient ascent auf D
 - sample minibatch mit m noise vectors $\{z^1, \dots, z^{m^2}\}$ aus noise distribution $p_g(z)$
 - gradient ascent auf G

Evaluation

3 Eigenschaften messen:

- 1) Sample quality: sehen Bilder echt aus?
- 2) diverse Samples: generiert G immer nur ein Bild oder verschiedene
- 3) generalization: generiert G neue Bilder oder lernt es nur Trainingsbilder auswendig

Inception Score

- pre-trained Classifier klassifiziert Samples
 - vergleiche Label-distribution im Dataset und Label-distr. der erzeugten Bilder
- 1) 2) abgedeckt
3) fehlt

Fréchet Inception Distance

- vergleiche Verteilung der Features

Nearest neighbor

- betrachte NNs eines Samples im Trainingsset um 3) zu prüfen
- NN Suche im feature space

Deep Convolutional GAN DCGAN

Probleme standard GAN:

- (1) Oszillation ohne Konvergenz
- (2) Mode collapse: G modelliert nur kleines subset

→ Regeln:

- (i) ersetze Pooling Layer durch
 - in D: Strided Convs
 - in G: Strided transposed Convs
- (ii) BatchNorm in D, G
- (iii) keine FC hidden layers
- (iv) ReLU in G (außer tanh für output)
- (v) Leaky ReLU in D

Conditional GAN

- erlaube Sampling von conditional distribution
- Anwendung:
 - next frame prediction
 - image in-painting
 - sem seg
 - style transfer
 - img-to-img translation
- parametrisiere G, D mit conditioning quantity Y

Pix2Pix

- G = DCGAN + skip connections von layer i zu layer D
- D = reguläres ConvNet mit overlapping patches $N \times N$, aug am Ende
- ① benötigt Trainingsdaten mit Pixel-Pixel Annotation

Cycle GAN

- lerne Img 2 (Img Transl. von ungepaarten Daten)
 - 2 G Netze: G_A, G_B
 - c) 1 von Style $A \rightarrow B$ $A \rightarrow B \rightarrow A$ sollte konsistent sein
 - 2 $B \rightarrow A$ \hookrightarrow original wieder erzeugen
 - ! 2 D Netze \rightarrow damit $G(A)$ sehr ähnlich A_{real}
 $G(B)$ B_{real}

SPADE

semantische Bildsynthese

In: sem Maske

Out: Bild

Encoder /Decoder Struktur aber ohne Encoder

↳ füttet Sem. Maske auf versch. Hierarchien ins Netz

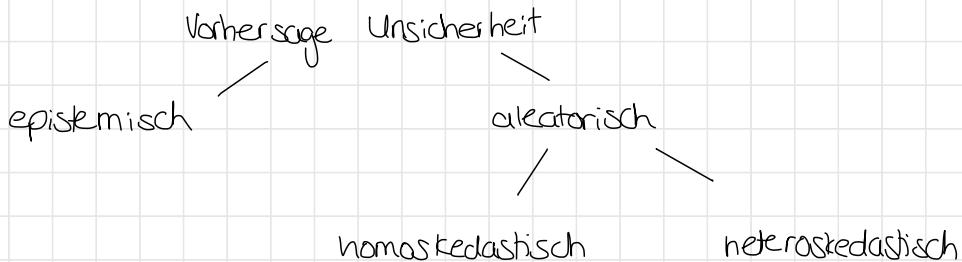
Advanced Topics

DL Challenges:

- supervised learning braucht sehr viele gelabelte Daten
→ Training mit weniger / ungenaugen Daten?
- Daten die Netz noch nicht kennt?

Unsicherheiten, zero-shot learning

- Modell sollte mit unbekannten Daten umgehen können
→ messen wie sicher sich Modell bei Vorhersage ist



aleatorisch:

- Unsicherheit in den Daten
 - Verrauschte Daten
 - Kann nicht reduziert werden, aber lernbar!
- zB Objekt zu weit weg von Kamera

Homoskedastisch: gleich für alle Eingaben (zB ungenauer Sensor)

Heteroskedastisch: Unsicherheit variiert je nach Eingabe

epistemisch:

Objekt unterrepr. in Trainingsdaten
↳ Modell erkennt es nicht

- Modellunsicherheit
- misst was das Modell nicht weiß
- theoretisch nicht vorhanden bei unbegrenzten Trainingsdaten

↳ Bayesian NNs: keine Verteilungen statt Gewichte

Confidence Calibration: zB Platt scaling mit validation set

Ensembles: mehrere Modelle trainieren, schauen wo uneinig

Bayesian Neural Networks

Softmax + wktverteilung

↪ Cross Entropy loss maximiert Top-1 Score

=> bias Richtung Aussagen mit hoher Konfidenz

=> Netz überschätzt Ausgaben!

→ lerne Gewichte als Verteilungen statt deterministisch / fix

↪ output: echte Wktverteilung

Problem: Inferenz schwierig

$$p(y^* | x^*, \underbrace{x, y}_{\text{Trainingsdaten}}) = \int p(y^* | x^*, \omega) \underbrace{p(\omega | x, y)}_{\substack{\uparrow \\ \text{Modellparams}}} d\omega$$

↑
keine analytische Evaluation möglich

vorhersage für

=> Approximieren mit einfacher Verteilung $q_\theta(\omega)$

$$= \int p(y^* | x^*, \omega) q_\theta(\omega) d\omega$$

=> Monte-Carlo Dropout $\approx \frac{1}{T} \sum_{t=1}^T p(y^* | x^*, \omega_t)$

$$\omega_t \sim q_\theta(\omega) = \text{Bernoulli Verteilung}$$

\rightarrow zufällig 0 oder 1

Training: als Dropout umsetzen

Testing: Dropout behalten!

T forward passes berechnen

→ Mittelwert = Vorhersage

Varianz = Maß für epistemische Unsicherheit
(je höher desto unsicherer)

Zero-Shot Learning

- unmöglich ALLE Klassen zu kennen / zu labeln
- erkenne nie vorher gesehene Klassen

$$\underbrace{y^{\text{unseen}} \cap y^{\text{seen}} = \emptyset}_{\text{zero-shot condition}}$$

Semantic embedding

- Beschreibungen der Klassen in Vektorraum

Möglichkeiten:

(i) Straightforward:

- trainiere Classifier auf bekannten Klassen um aus Bildern sem. emb. zu machen
- query mit unbekannter Klasse, suche nächstes sem. emb.

(ii) Transfer Knowledge:

(iii) Attribute discovery:

- text mining von zB Wikipedia
- lerne daraus optische / diskriminative Features

Few Shot Learning / Weak Supervision

- Modell soll gute Seg liefern gegeben sehr wenigen Daten

n-way k-shot Scenario:

Training: große Datenset mit Bild + Maske für manche Klassen

Testing: Support Set mit k Paaren für n neue Klassen

Ziel: segmentiere Testbilder in die n neuen Klassen

Weak Supervision

für Seg: liefere nicht gesamte Maske sondern nur

- BBoxen /
- Scribbles (gröbe Linien wo Objekt) /
- Single Points /
- Img level labels (mit attention models trainieren)

Domain Adaptation

- geg.
- Trainingsdaten aus source domain
 - ungelabelte Daten aus target domain
-] gleiche Klassen!

→ übertrage Model

Architekturen:

- GAN: Discriminator unterscheidet zw. source / target
- Cycle GAN: transferiere source Daten nach target space