Computer Vision

Objekterkennung - Convolutional Neural Networks and YOLO

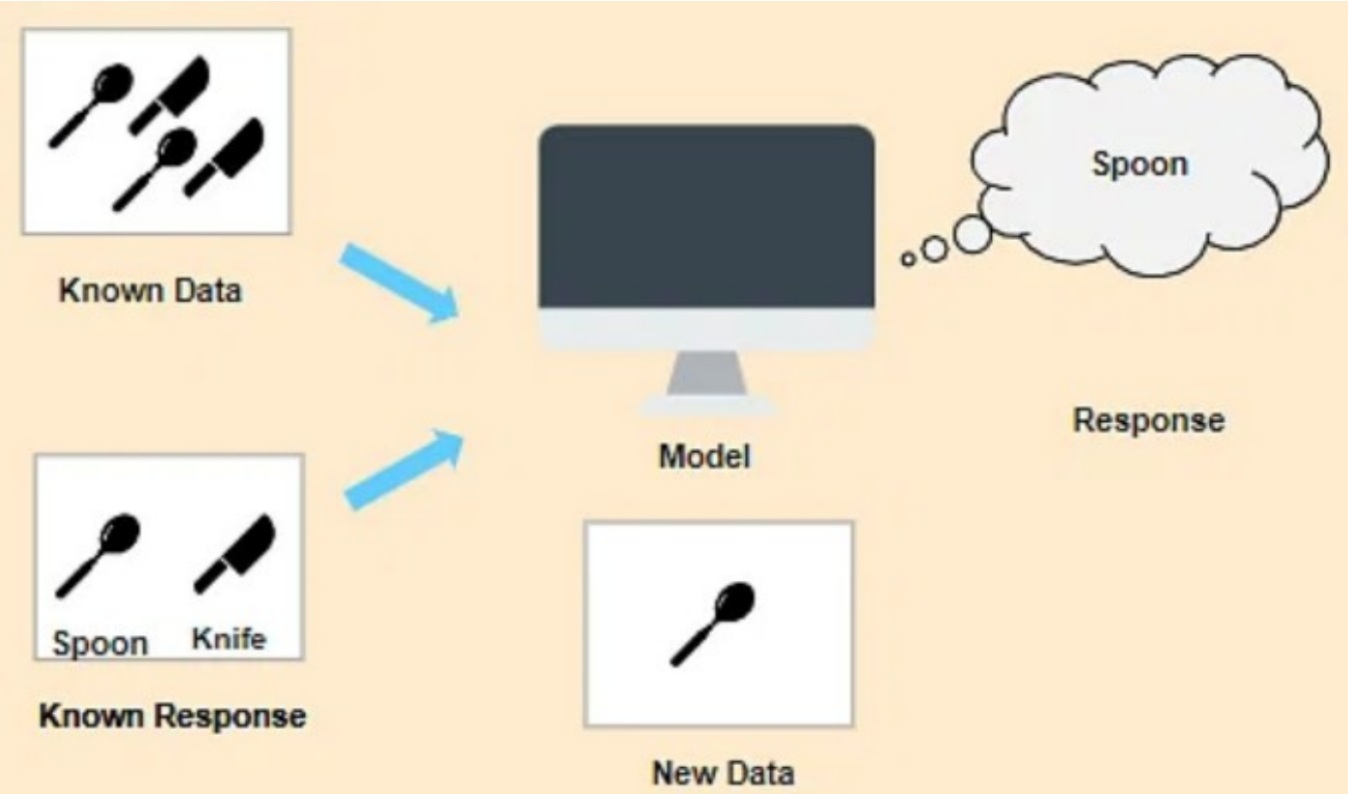
Alexandra Posekany

December 2025

Machine Learning und Klassische Objekterkennung HOG + SVM

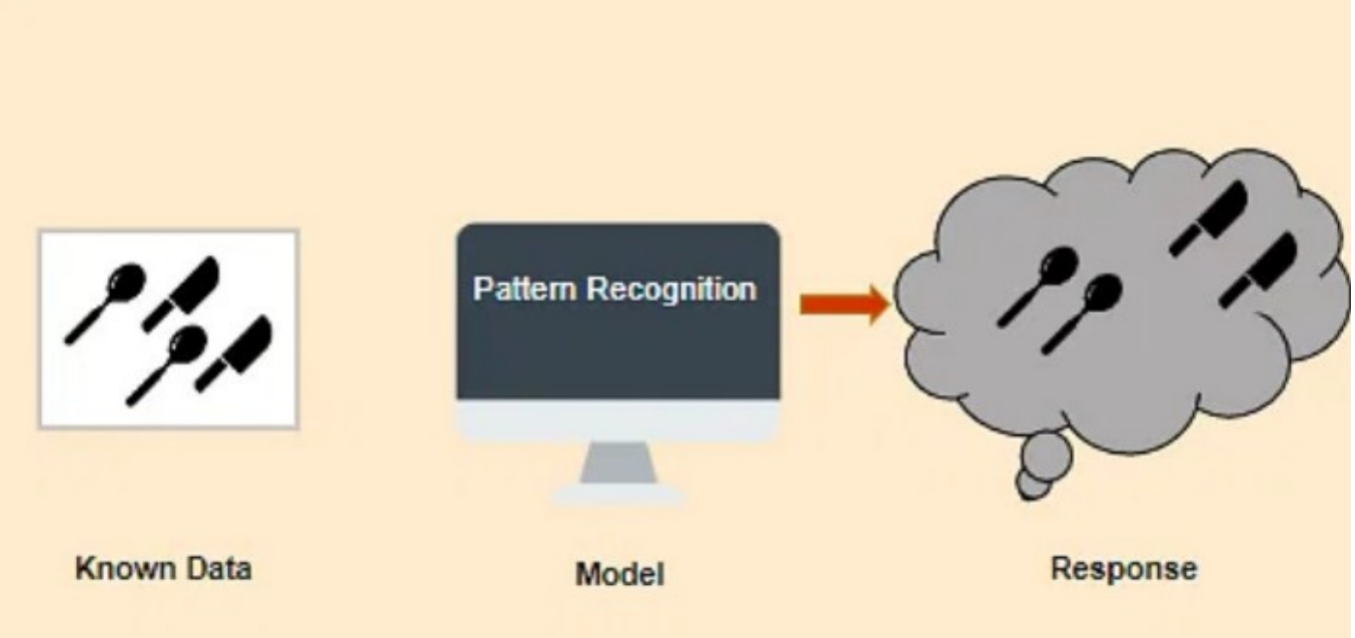
Überwachtes Lernen (Supervised Learning)

Definition: Beim überwachten Lernen werden dem Modell sowohl die Eingabedaten als auch die gewünschten Ausgabedaten (Labels) zur Verfügung gestellt.



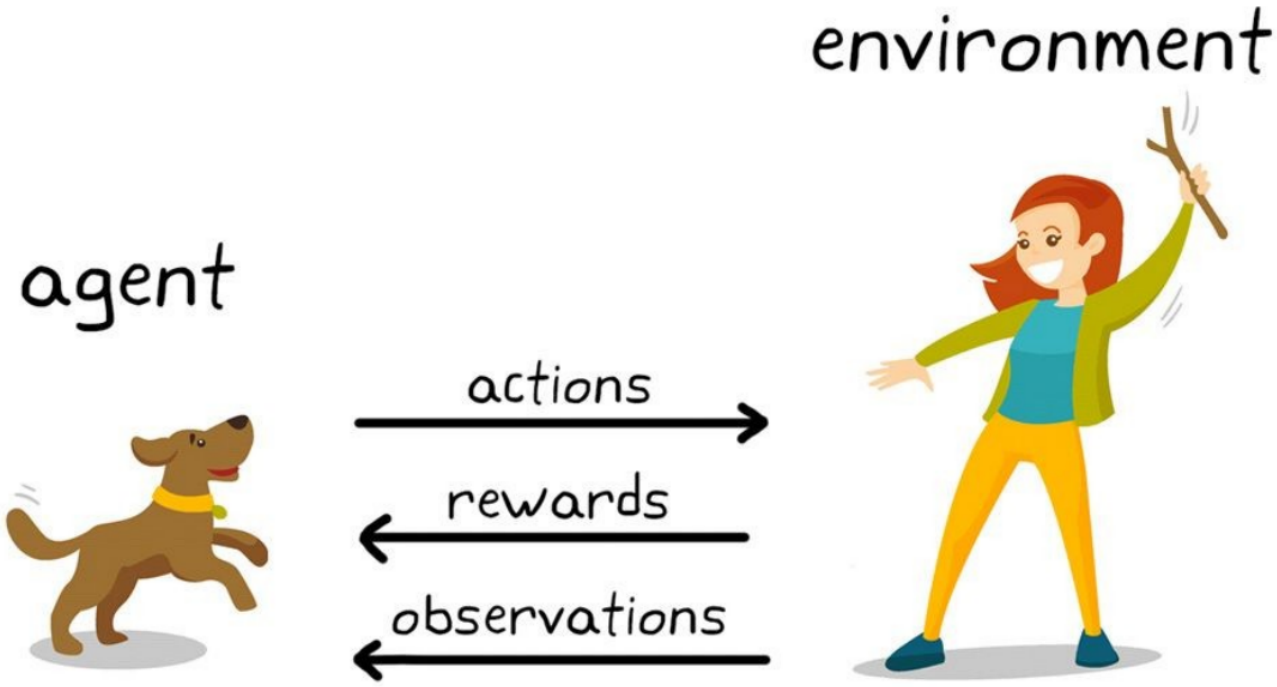
Unüberwachtes Lernen

Definition: Beim unüberwachten Lernen werden dem Modell nur Eingabedaten ohne Labels zur Verfügung gestellt.

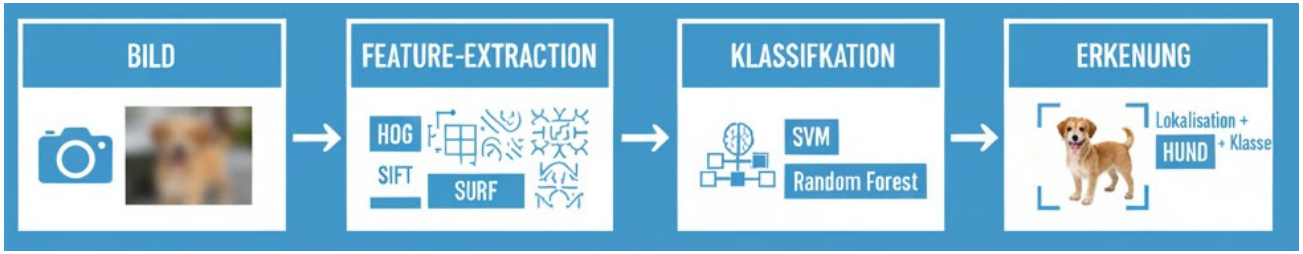


Verstärkendes Lernen (Reinforcement Learning)

Definition: Beim verstärkenden Lernen lernt ein Agent durch Interaktion mit einer Umgebung und erhält Belohnungen für gewünschtes Verhalten.



Von Merkmalen zur (klassischen) Objekterkennung



Vorteile:

- Interpretierbar (wir wissen, was die Features sind)
- Weniger Daten benötigt (keine riesigen Datensätze)
- Schnell zu trainieren
- Gut für spezifische, eingeschränkte Aufgaben

Histogram of Oriented Gradients (HOG)

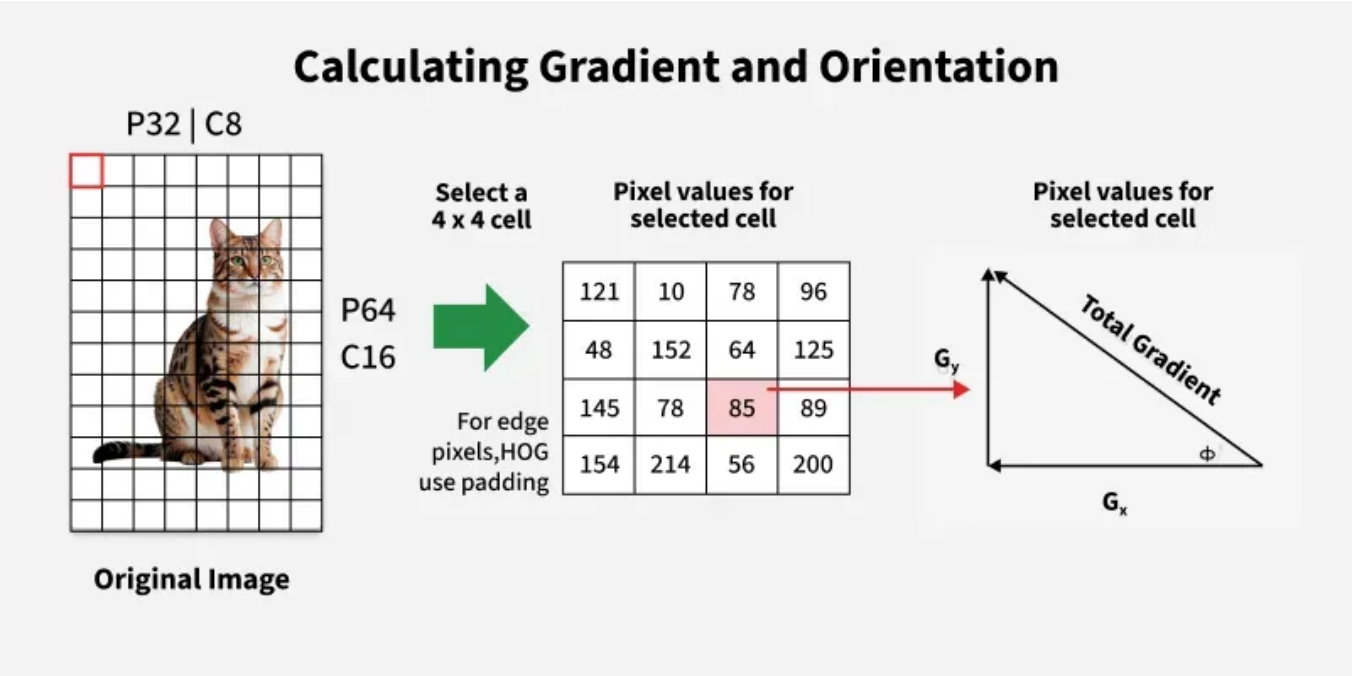
Motivation: Menschliche Wahrnehmung

Idee: Die Form eines Objekts (z.B. Mensch) lässt sich sehr gut durch die Verteilung von Kantenrichtungen beschreiben.

- Lokale Gradientenrichtungen sind robust gegenüber Beleuchtungsänderungen
- Die Verteilung (Histogramm) dieser Richtungen beschreibt die Form

HOG Algorithmus visuell

1. Bild in kleine Zellen (z.B. 8×8 Pixel) unterteilen.



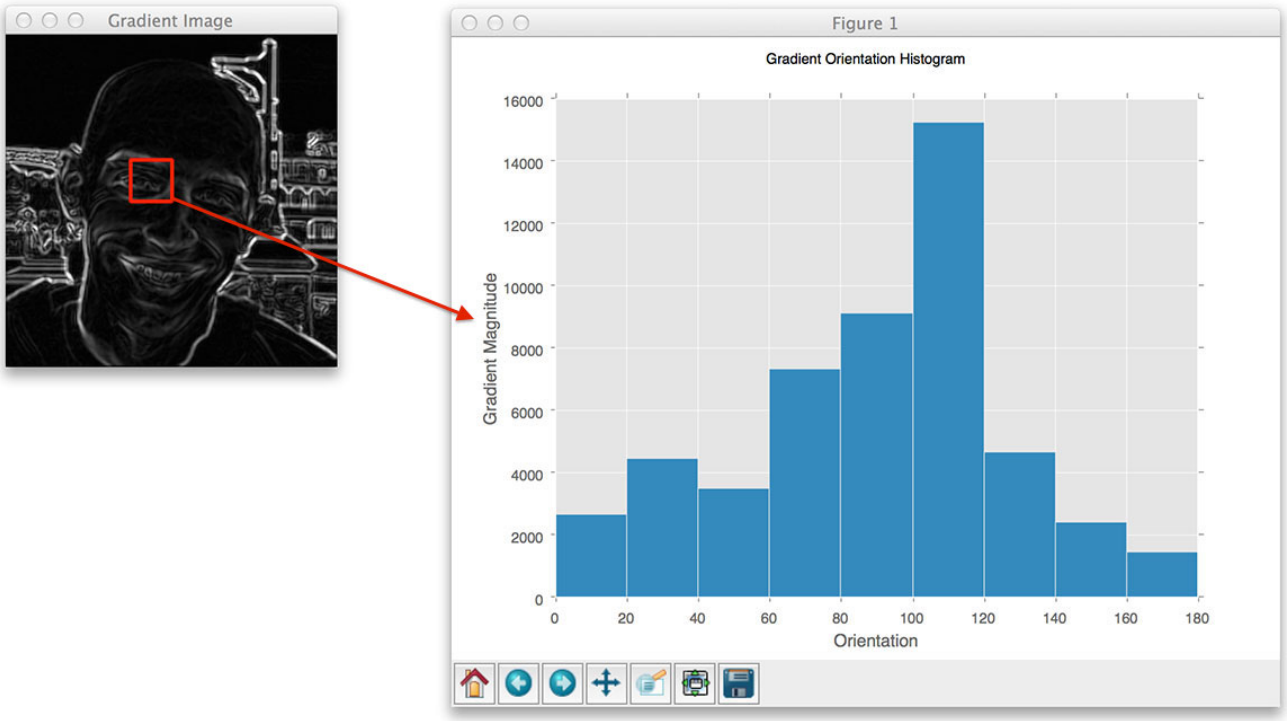
2. In jeder Zelle: Berechne Gradienten (Richtung und Stärke).

$$G_x = I * [-1,\ 0,\ 1], \qquad G_y = I * \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

$$|G| = \sqrt{G_x^2 + G_y^2}, \qquad \theta = \arctan\left(\frac{G_y}{G_x}\right)$$

HOG Algorithmus visuell

3. Erstelle ein Histogramm der Richtungen (z.B. “Viel vertikal, wenig horizontal”).



Ergebnis: Eine vereinfachte Darstellung, die den “Umriss” einfängt, aber Farben ignoriert.

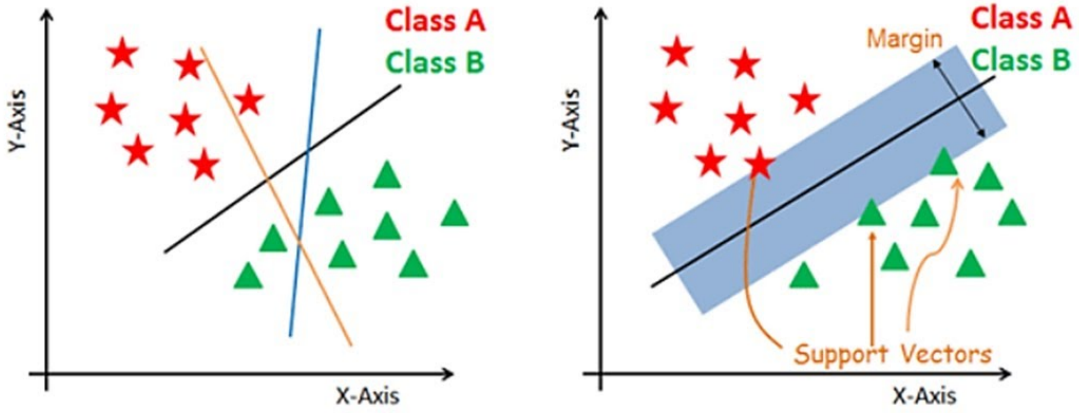
Support Vector Machine

Wir haben den HOG-Vektor (Features). Jetzt brauchen wir eine Entscheidung, welches “Label” (Klasse) wir vergeben.

Linear SVM (Support Vector Machine):

Findet die Hyperebene (Trennlinie), die die Klassen voneinander im hochdimensionalen Raum am besten trennt (größtmöglicher Sicherheitsabstand / Margin).

SUPPORT VECTOR MACHINE



Evaluierung - Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

- TP (True Positive): Korrekt als Objekt erkannt
- FP (False Positive): Falscher Alarm (Hintergrund als Objekt)
- FN (False Negative): Objekt übersehen
- TN (True Negative): Korrekt als Hintergrund erkannt

Metriken

- Genauigkeit (Accuracy): “wieviel insgesamt richtig”

$$\frac{TP + TN}{TP + TN + FP + FN}$$

- Präzision (Precision): “wieviel der erkannten Objekte sind wirklich Objekte”

$$\frac{TP}{TP + FP}$$

- Recall (Sensitivität): “wieviel der echten Objekte wurden erkannt”

$$\frac{TP}{TP + FN}$$

- F1-Score: “Harmonic Mean von Precision und Recall”

$$2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Precision-Recall Trade-off

Das fundamentale Dilemma

Bei hohem Confidence-Threshold: - Weniger Detektionen - Aber diese sind sehr sicher

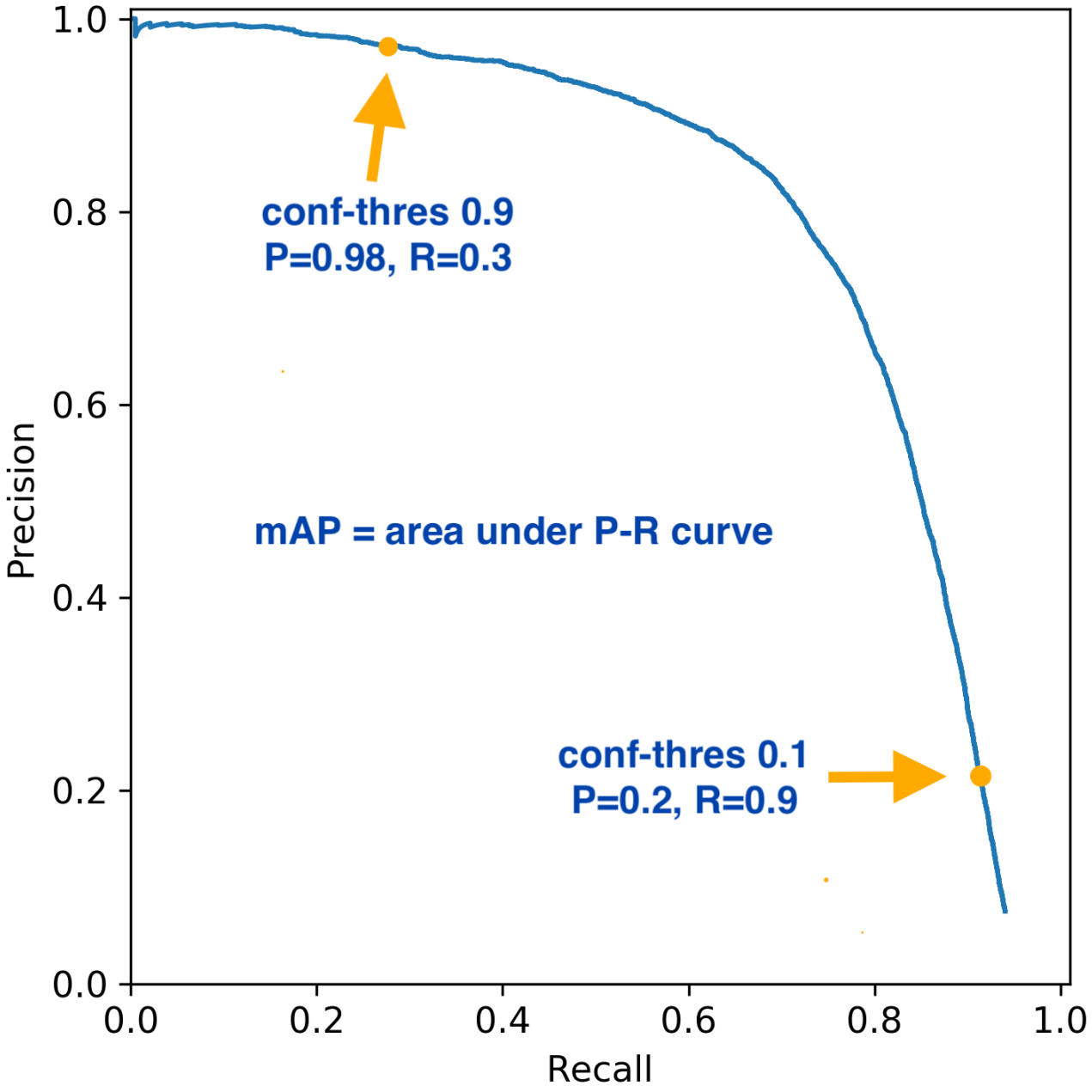
HOHE PRECISION, NIEDRIGER RECALL

Bei niedrigem Confidence-Threshold: - Viele Detektionen
- Auch unsichere dabei

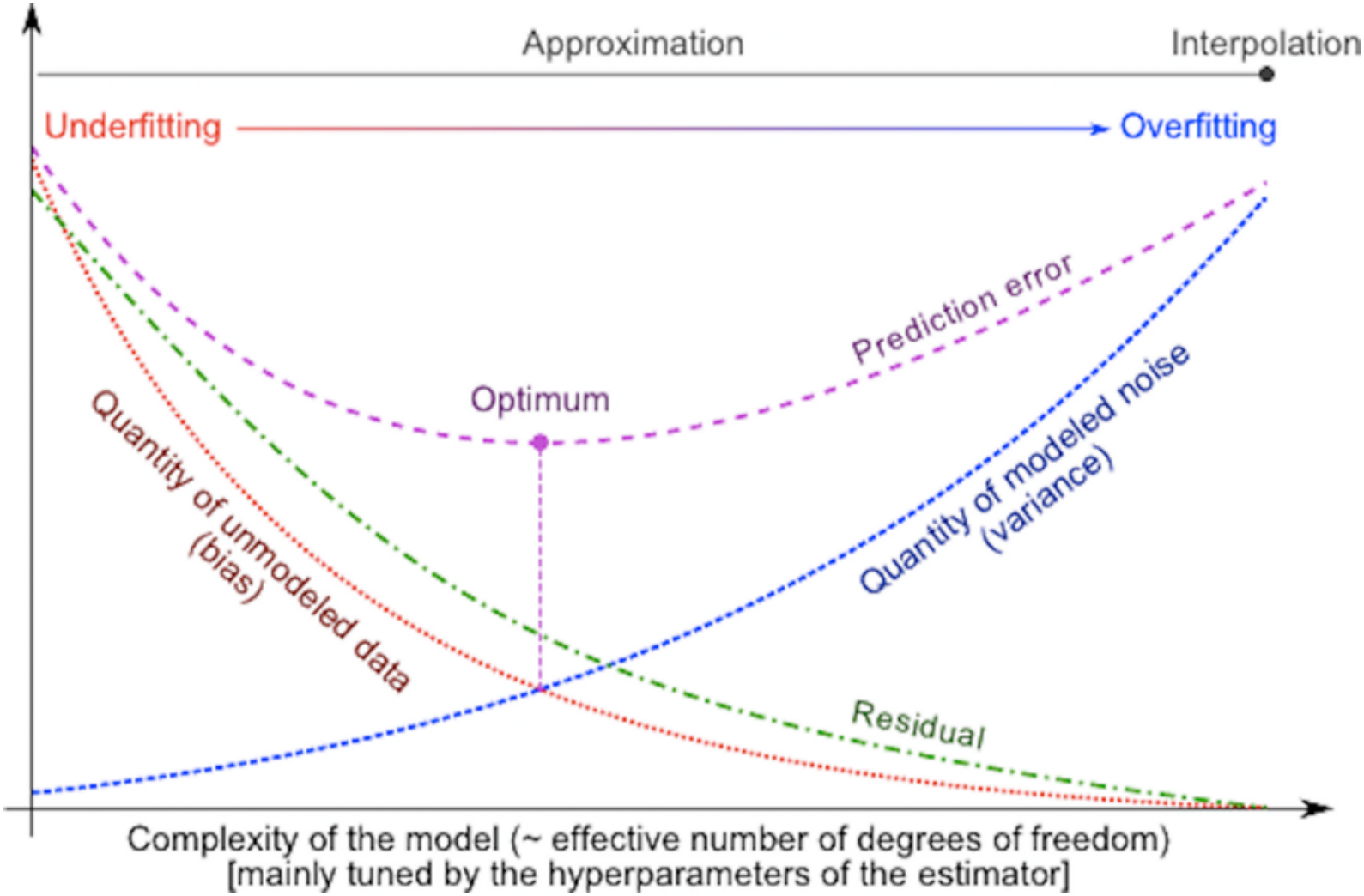
NIEDRIGE PRECISION, HOHER RECALL

- Autonomes Fahren: Hohe Precision (keine falschen Fußgänger!)
- Überwachungskameras: Hoher Recall (kein Einbrecher übersehen!)

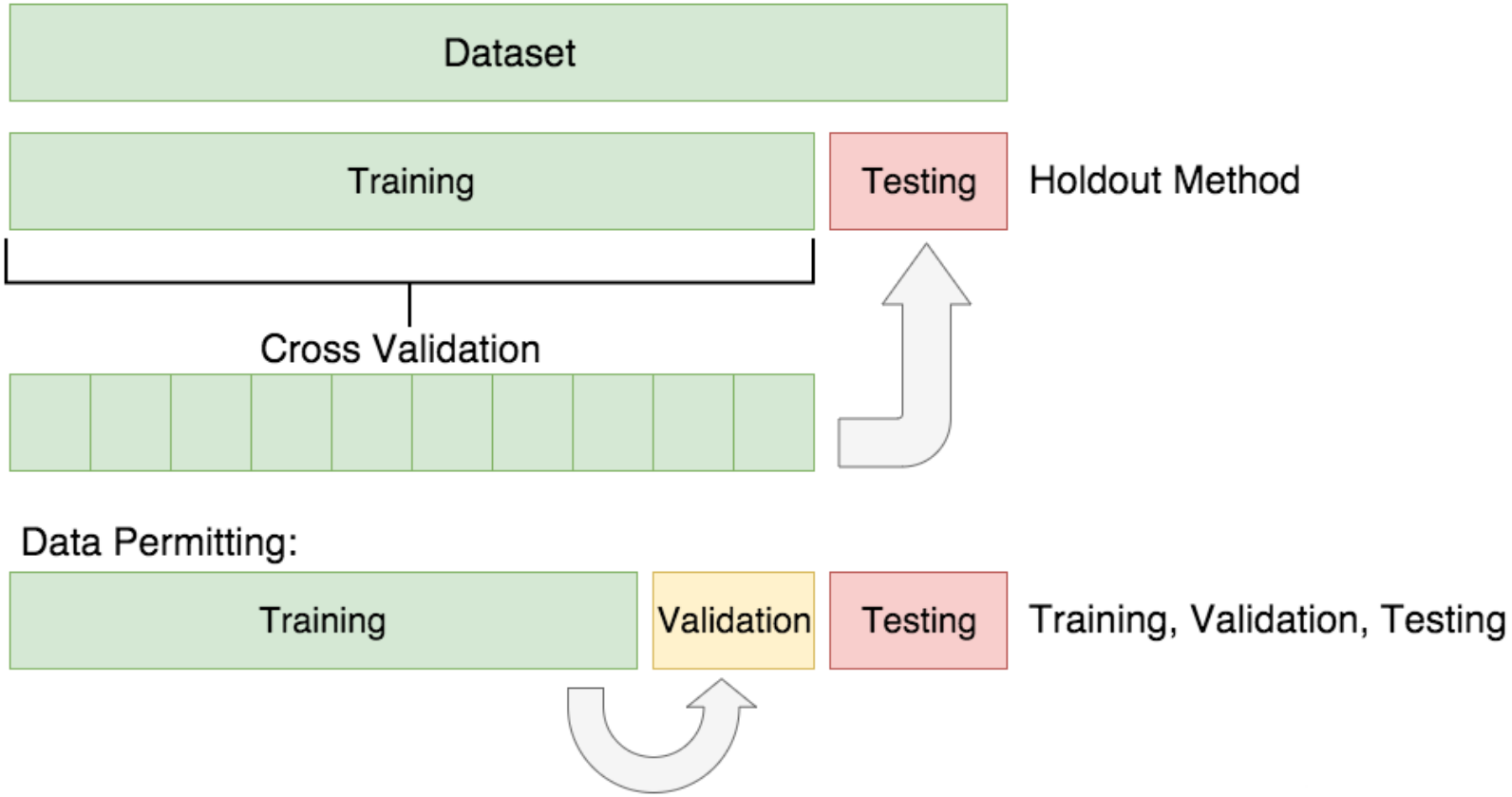
Precision Recall Curve



Probleme der Modellanpassung



Datenaufteilung zur Validierung



Das Such-Problem (Sliding Window)

Ein Klassifikator (SVM/CNN) kann nur sagen "Ja/Nein" für einen Bildausschnitt.

Wie finden wir das Objekt im großen Bild?

Sliding Window Ansatz:

- Schiebe ein Fenster fester Größe über das Bild (von links oben nach rechts unten).
- Wiederhole das für verschiedene Skalierungen (Bildpyramide), um kleine und große Objekte zu finden.

Problem:

Extrem rechenintensiv!

Ein VGA Bild (640×480) generiert zehntausende Fenster.

99,9% der Fenster sind Hintergrund.

Region Proposals (Der smarte Weg)

Statt “dumm” überall zu suchen, suchen wir nur dort, wo “etwas” ist.

Selective Search:

- Pixel basierend auf Farbe/Textur zu Blobs zusammenfassen (Superpixel).
- Generiert ca. 2.000 Vorschläge (Proposals) statt 100.000 Fenster.

Diese Vorschläge (“Blobs”) werden dann dem Klassifikator übergeben.

Vorläufer von R-CNN: R-CNN nutzte genau diesen Selective Search Ansatz.

Sliding Window vs. Region Proposals

Aspekt	Sliding Window	Region Proposals
Anzahl Fenster	Sehr viele ($10^4 - 10^5$)	Wenige ($10^2 - 10^3$)
Geschwindigkeit	Langsam	Schneller
Vollständigkeit	Garantiert vollständig	Kann Objekte verpassen
Präzision	Niedrig (viel Hintergrund)	Hoch (wenig Hintergrund)
Implementierung	Einfach	Komplex

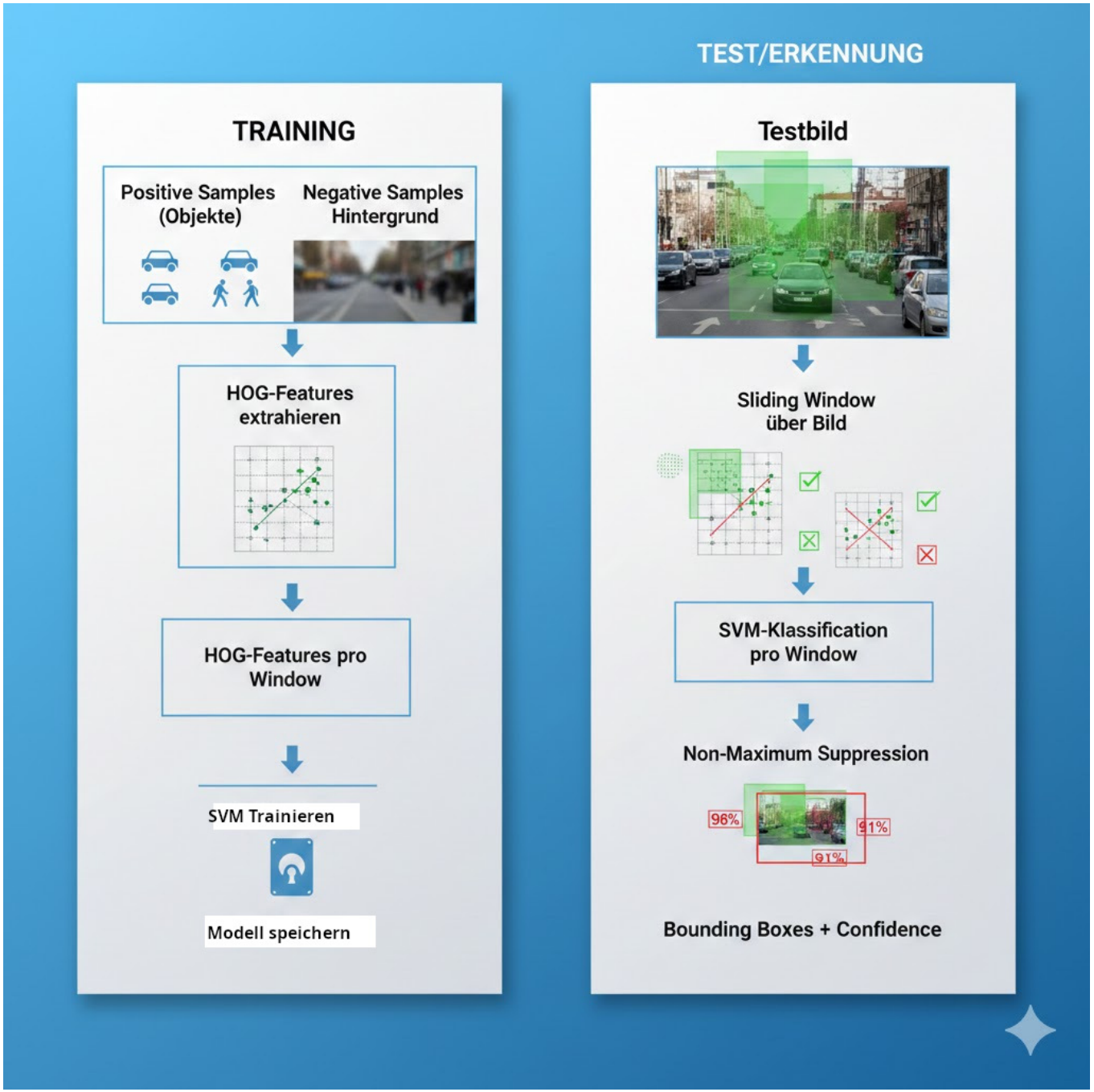
Non-Maximum Suppression (NMS)

Ziel: Redundante Detektionen entfernen

Problem: Ein Objekt wird oft durch mehrere überlappende Fenster detektiert

Lösung: Behalte nur die beste Detektion pro Objekt

Ablauf

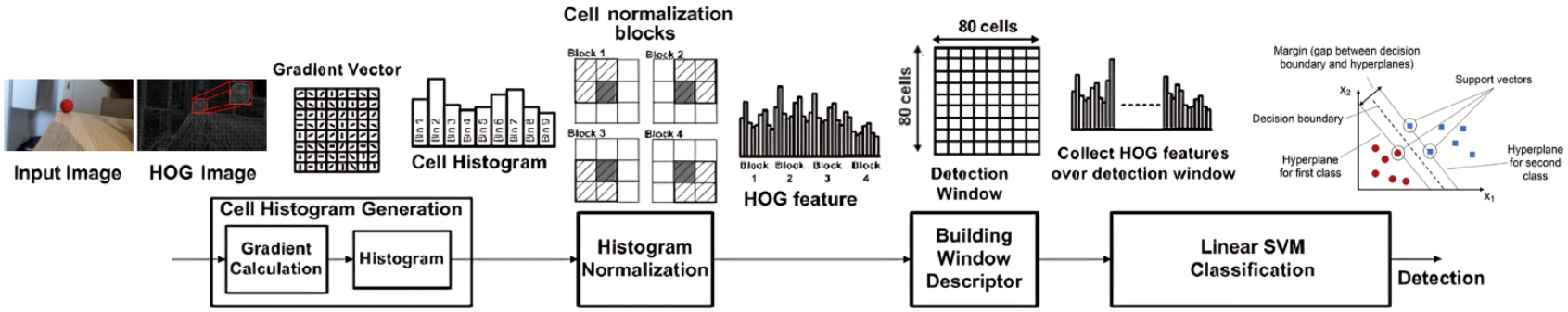

TEST/ERKENNUNG
TRAINING
Positive Samples (Objekte)
Negative Samples Hintergrund
HOG-Features extrahieren
HOG-Features pro Window
SVM Trainieren
Modell speichern
Testbild
Sliding Window über Bild
SVM-Klassification pro Window
Non-Maximum Suppression
96%
91%
91%
Bounding Boxes + Confidence

Warum reicht das nicht? (Limitierungen)

- Geschwindigkeit: Sliding Window ist zu langsam für Echtzeit.
- Region Proposals (Selective Search) sind schneller, aber immer noch ein Flaschenhals (CPU-basiert).
- Hand-Crafted Features: HOG ist gut für stehende Menschen, aber schlecht für deformierbare Objekte (z.B. schlafende Katze).
- Pipeline-Trennung:
  - Schritt 1 (Proposals) und Schritt 2 (Klassifikation) sind getrennt.
  - Der Klassifikator kann den Proposal-Algorithmus nicht korrigieren ("Du hast den Kopf abgeschnitten!").

Lösung: Wir brauchen eine Methode, die gleichzeitig guckt (wo?) und erkennt (was?). $\rightarrow$ YOLO (You Only Look Once).

Klassische Pipeline Objekterkennung



Convolutional Neural Networks (CNNs) für Objekterkennung

Motivation für CNNs

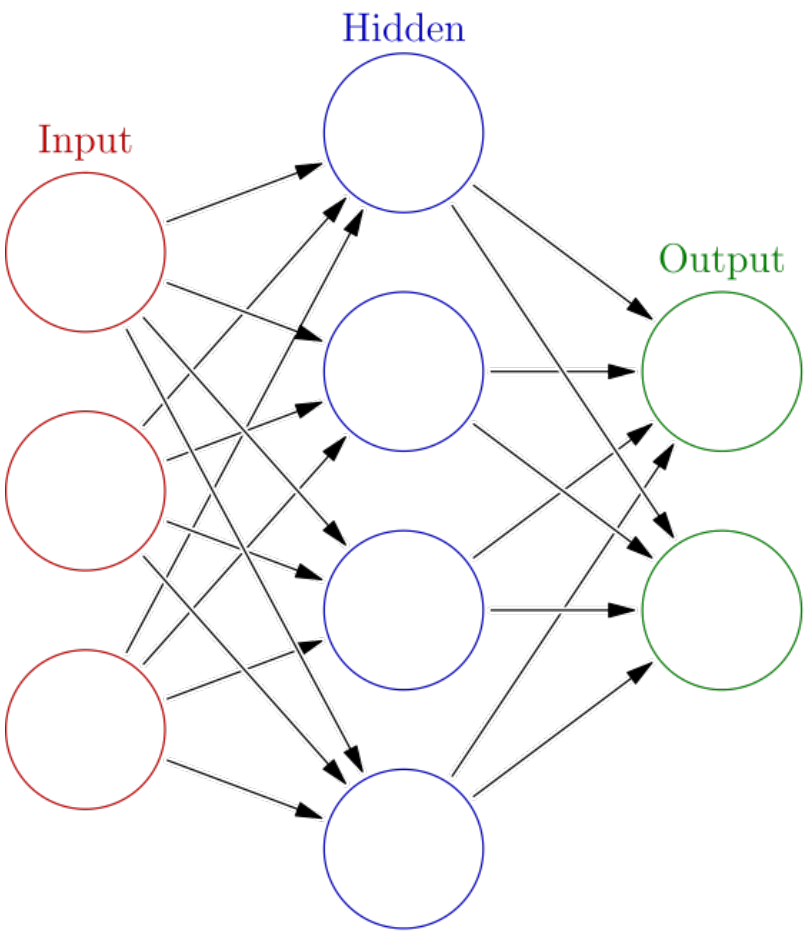
- Hand-Crafted Features (HOG, SIFT, etc.) sind limitiert und domänenabhängig:
  - HOG: Gut für Personen, schlecht für Gesichter
  - SIFT: Gut für Texturen, schlecht für Objektteile

  Jede Aufgabe benötigt eigenen Feature-Engineering-Ansatz
- CNNs lernen Features automatisch aus Daten.
- Bessere Generalisierung auf verschiedene Objekte und Bedingungen.

Neuronale Netzwerke - Neural networks

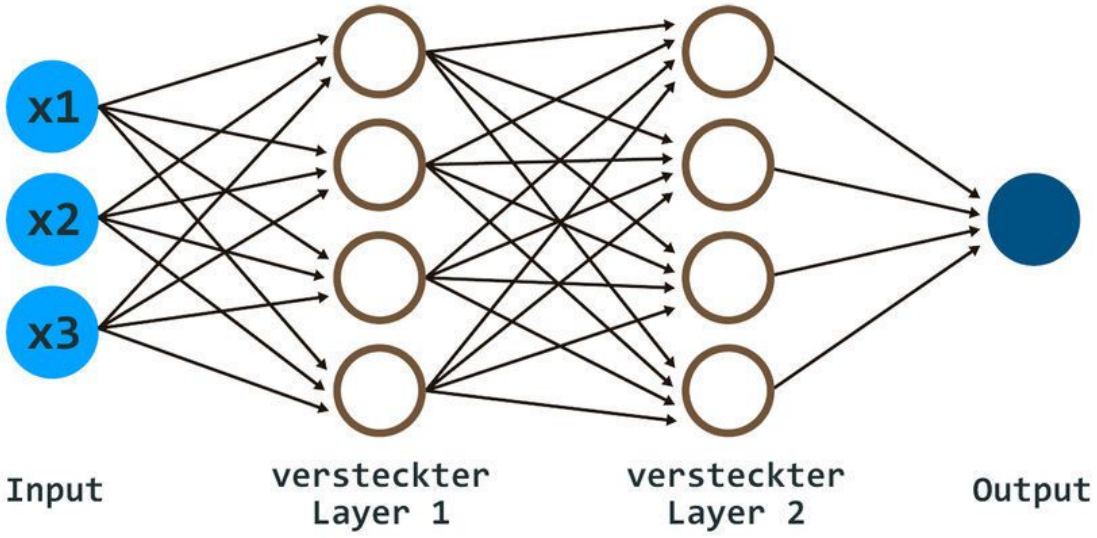
Ablauf des Algorithmus

- Simulation von kognitiven Lernprozessen im Gehirn und Versuch der Übertragung auf Maschinen
- Hohe Prädiktabilität
  gute Modellvorhersagen
- schlechte Interpretierbarkeit der Entscheidungsfindungsprozesse

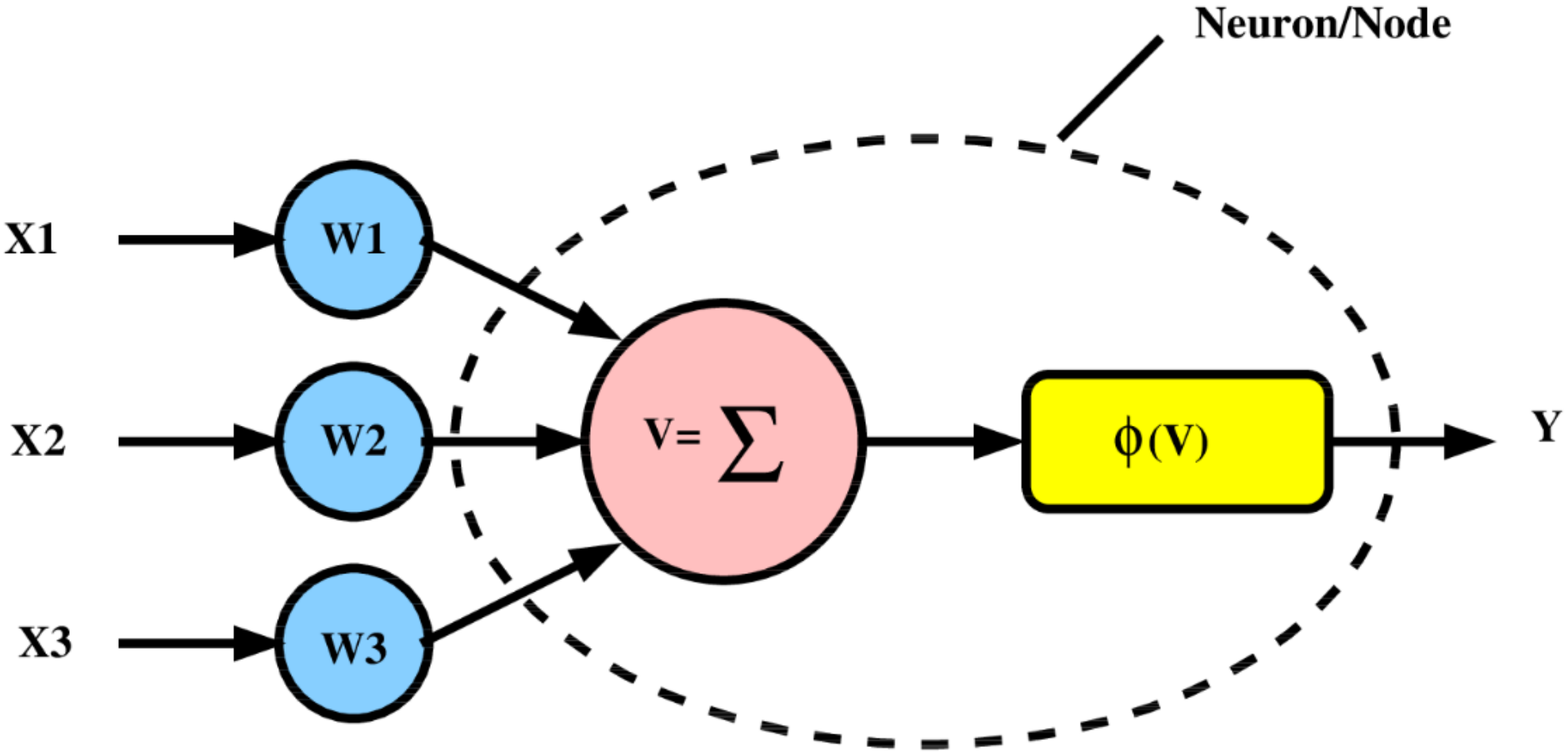


Neuronale Netzwerke - Neural networks

Ein neuronales Netz besteht aus Knoten (engl. nodes), die Neuronen genannt werden, in mehreren Schichten (engl. layers). Jede einzelne Schicht funktioniert wie ein Perceptron, nur dass sich die Inputdaten ändern, im Sinne, dass sie auf den Output der zugrunde liegenden Schicht aufbauen.



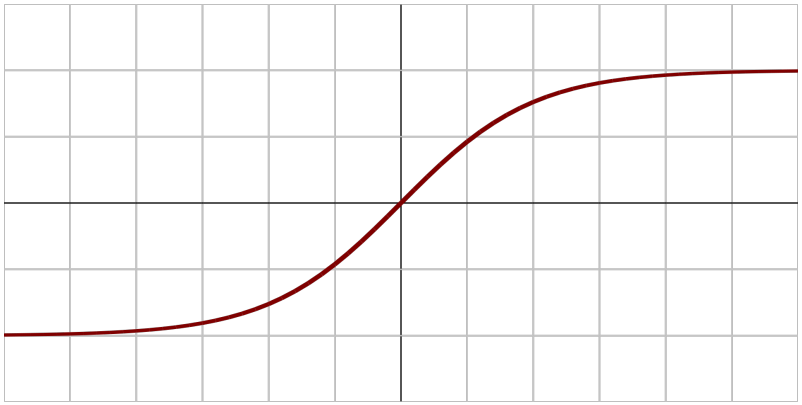
Künstliche neuronale Netzwerke



Gehirn (biologisches NN)	$\longrightarrow$	künstliches Neuronales Netzwerk
Neuronen	$\longrightarrow$	Knoten (Nodes)
Stärke von Synapsen	$\longrightarrow$	Gewichte (Weights)

Konzeptueller Aufbau

- linearer Teil: $v = \sum_i w_i \cdot x_i = w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_n \cdot x_n$
- Nichtlinearität: Aktivierungsfunktion



- Verlustfunktion

absolute lineare Verlustfunktion, L1 $L(x) = |x|$

quadratische Verlustfunktion, L2 $L(x) = x^2$

Klassifikation - Perceptron

Ablauf des Algorithmus

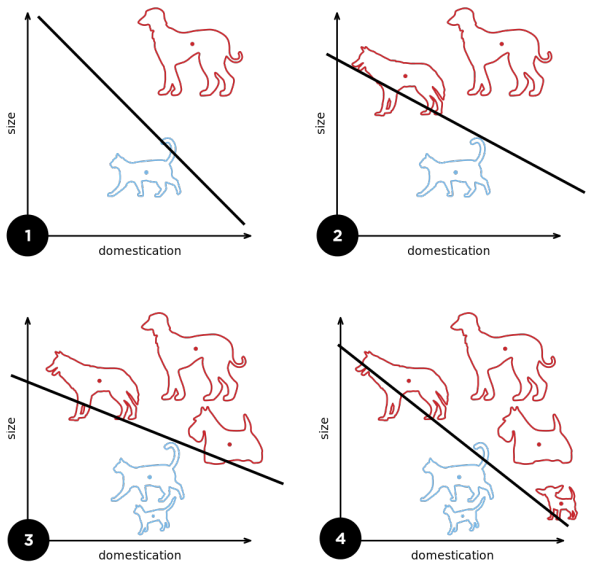
1. Initialisiere die Gewichte der Neuronen (z.B. alle $= 0$)
2. Klassifiziere, indem iterativ
   2.1 die Thresholdfunktion

   $$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$

   berechnet wird

   2.2 für alle Punkte aus dem Trainingsdatensatz ermittle, ob die Klassifikation stimme

   2.3 passe die Gewichte an, sodass $\sum_{i=1}^{m} w_i x_i$ entsprechend erhöht oder verringert wird, um eine andere Klassifikation zu erzielen
3. Validiere, indem du die Klassifikation mit den finalen Gewichten auf Testdaten anwendest und die wahren Labels und die vom Algorithmus zugewiesenen Labels verglichen werden



Ablauf des Perceptron Algorithmus

Ermittlung der neuen Schätzwerte in jedem Schritt

1. Mithilfe der **Gewichte (weights)** und des **Bias (offset)** werden die Prädiktionen $z = \mathbf{w} \cdot \mathbf{x} + b$ ermittelt.

2. Die numerischen Werte z werden in Gruppen umgerechnet mithilfe einer nicht-linearen **Aktivierungsfunktion**. Die Wahl dieser Funktion hat große Auswirkung auf die Performanz und Lerngeschwindigkeit des neuronalen Netzwerks!

   Als Beispiel betrachten wir die Sigmoidfunktion (=logistische Funktion) zur Klassifikation in 2 Gruppen:

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Ablauf des Perceptron Algorithmus

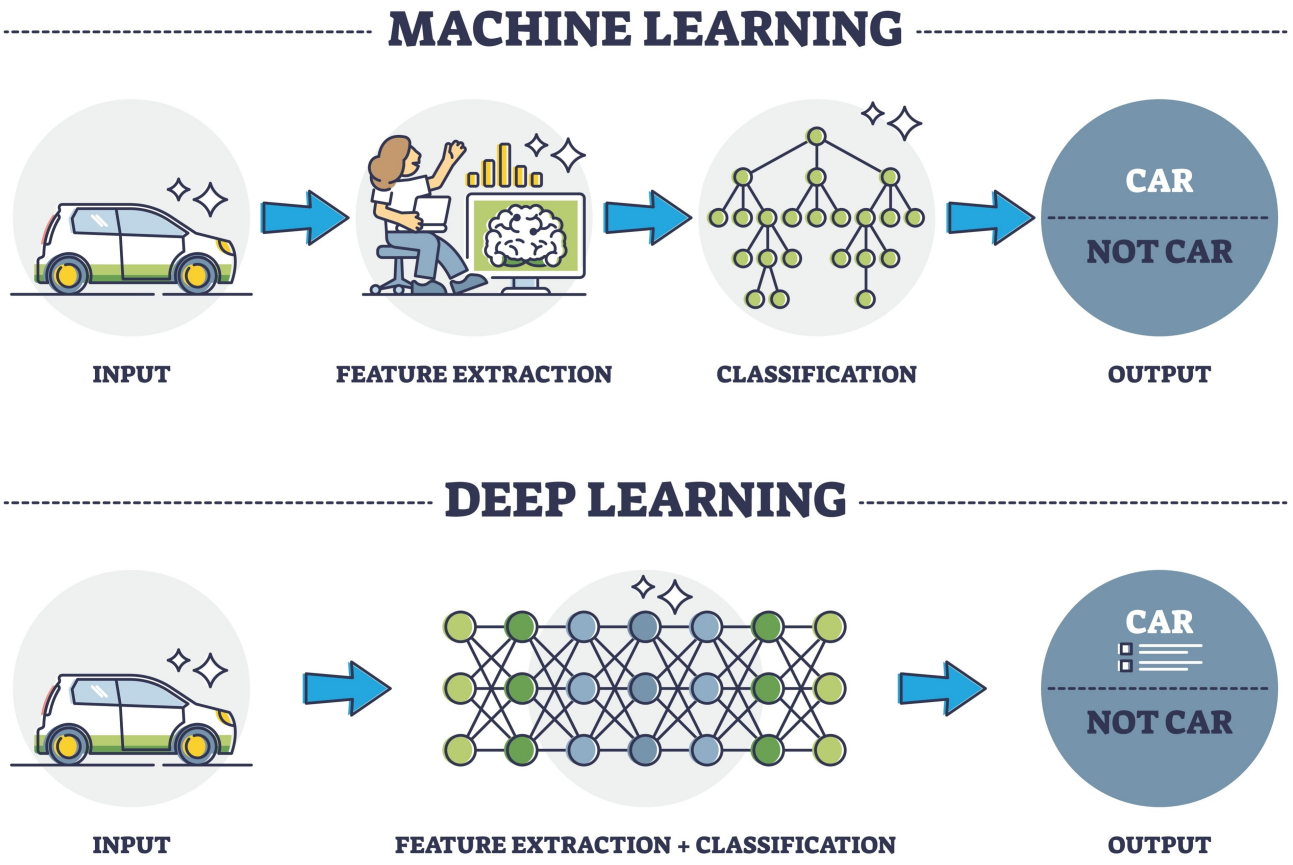
Lernen aus den eigenen Fehlern

3. Backpropagation (backward propagation of errors): Der Gradient der Verlustfunktion in Bezug auf die Gewichte $\mathbf{w}$ entscheidet wie stark diese verändert werden müssen.

   Wir kennen bereits die Verlustfunktion von Regressionsmodellen, nämlich die quadratische Verlustfunktion $L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$.

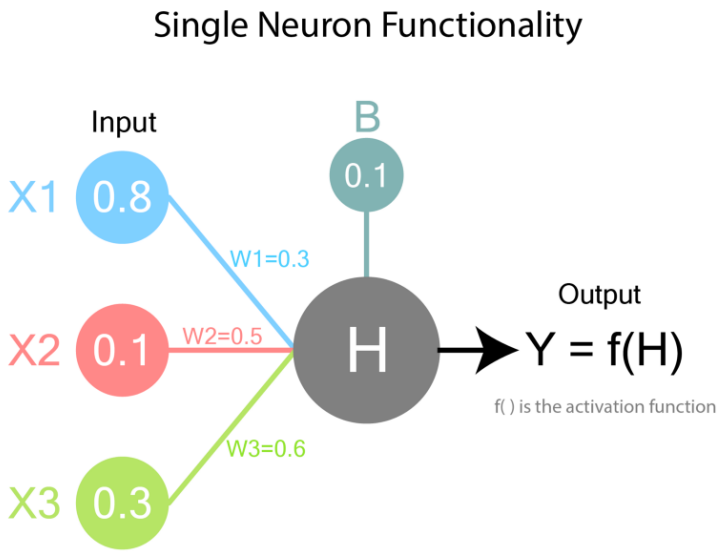
4. Optimierung, also Finden der "Senke" der mehrdimensionalen Verlustfunkion im hochdimensionalen Raum mit den Gewichten $\mathbf{w}_i$ als Raumdimensionen.

Deep Learning 1.0.1



Informationsübertragung

In jeder Einzelschicht geschieht die Approximation wie beim Perceptron basierend den den jeweiligen Inputs.
Ein wichtiger Aspekt dabei ist das **Verstärken von Knoten mit hohem Gewicht**, wenn diese in mehreren Durchläufen als relevant gefunden werden, und andererseits das **Ausdünnen von Knoten mit niedrigem Gewicht**. Dazu dienen die Verlustfunktionen!



$$H = X1*W1 + X2*W2 + X3*W2 + B$$

$$Y = f(H)$$

Backpropagation (backward propagation of errors)

Der Gradient der Verlustfunktion in Bezug auf die Gewichte **w** entscheidet wie stark diese verändert werden müssen.

Wie funktioniert das?

Alle Knoten werden durch folgende Eigenschaften charakterisiert:

- Input Vektor $\mathbf{x} = (x_1, \cdot, x_p)$
- Output Vektor $y = (y_1, \ldots, y_n)$
- Verlustfunktion C "cost function"
- Anzahl der Layers L
- Matrix der Gewichte $W^l = \begin{matrix} w_{1,1} & \ldots & w_{1,\mathcal{I}} \\ w_{2,1} & \ldots & w_{2,\mathcal{I}} \\ \vdots & \ddots & \vdots \\ w_{\mathcal{J},1} & \ldots & w_{\mathcal{J},\mathcal{I}} \end{matrix}$
  der $\mathcal{I}$ Knoten von Schicht l-1 mit $\mathcal{J}$ Knoten in Schicht l
- Aktivierungsfunktion f^l auf Schicht l

Fully Connected Networks (FCN) auf Bildern?

Warum nehmen wir nicht einfach ein normales neuronales Netz (MLP)?

Rechenbeispiel: Bild: 1000×1000 Pixel RGB (3 Mio. Inputs).

Hidden Layer mit 1000 Neuronen.

Gewichte: $3.000.000 \times 1.000 = 3$ Milliarden Parameter!

Problem:

- Zu wenige Daten, um derartig viele Parameter zu trainieren (Overfitting auf Trainingsdaten)
- Verlust der räumlichen Struktur (Pixel links oben hat keinen Bezug zu Pixel rechts unten, wenn man alles "plattklopft"). Keine Features wie Kanten, Ecken, Texturen etc. erkennbar.

Die Lösung – Lokale Verbindungen (Convolution)

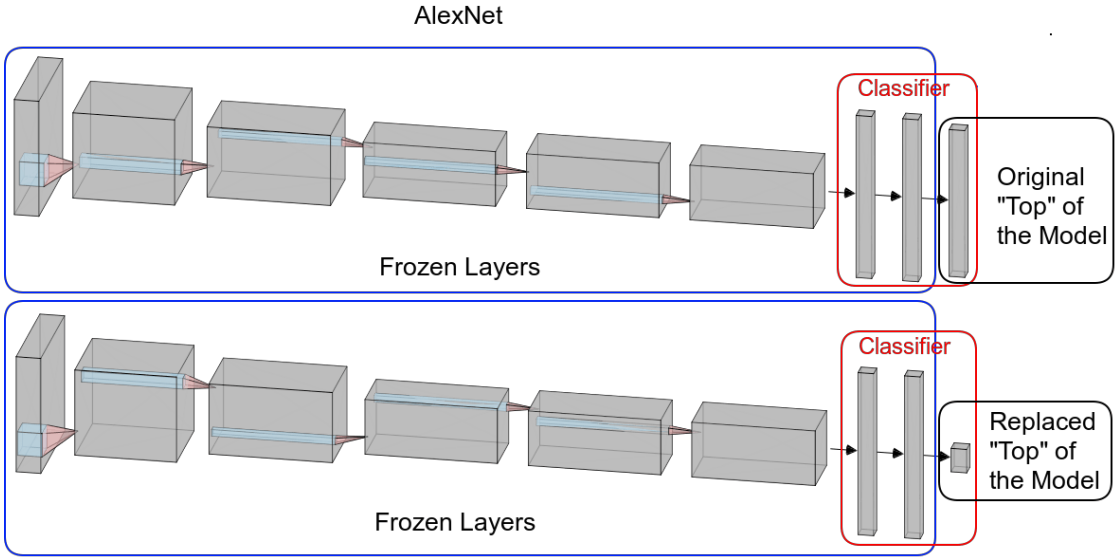
Idee: Ein Neuron muss nicht das ganze Bild sehen, sondern nur einen kleinen Ausschnitt (Receptive Field).

Analogie: Wenn wir eine Kante suchen, schauen wir nur auf 3×3 Pixel, nicht auf das ganze Bild.

Weight Sharing: Wir suchen die gleiche Kante überall im Bild. Also benutzen wir denselben Filter für das ganze Bild.

Schlüsselerkenntnis: Netzwerk lernt bessere Features selbst!

Aufbau AlexNet



Der Convolutional Layer

<u>Faltung</u> (Convolution) = Filter über Bild schieben

Der Unterschied:

- **Sobel**: Kernel war fix $[-1, 0, 1]$.
- **CNN**: Kernel ist variabel $[w_1, w_2, w_3]$.

Ziel: Das Netzwerk lernt die Zahlen w_i (Gewichte) so, dass der Fehler am Ende minimal ist.

Konzept: Ein CNN lernt quasi, "Sobel-Filter" zu bauen!

Hyperparameter der Convolution

Filter Size (Kernel): Meist 3×3 oder 5×5.

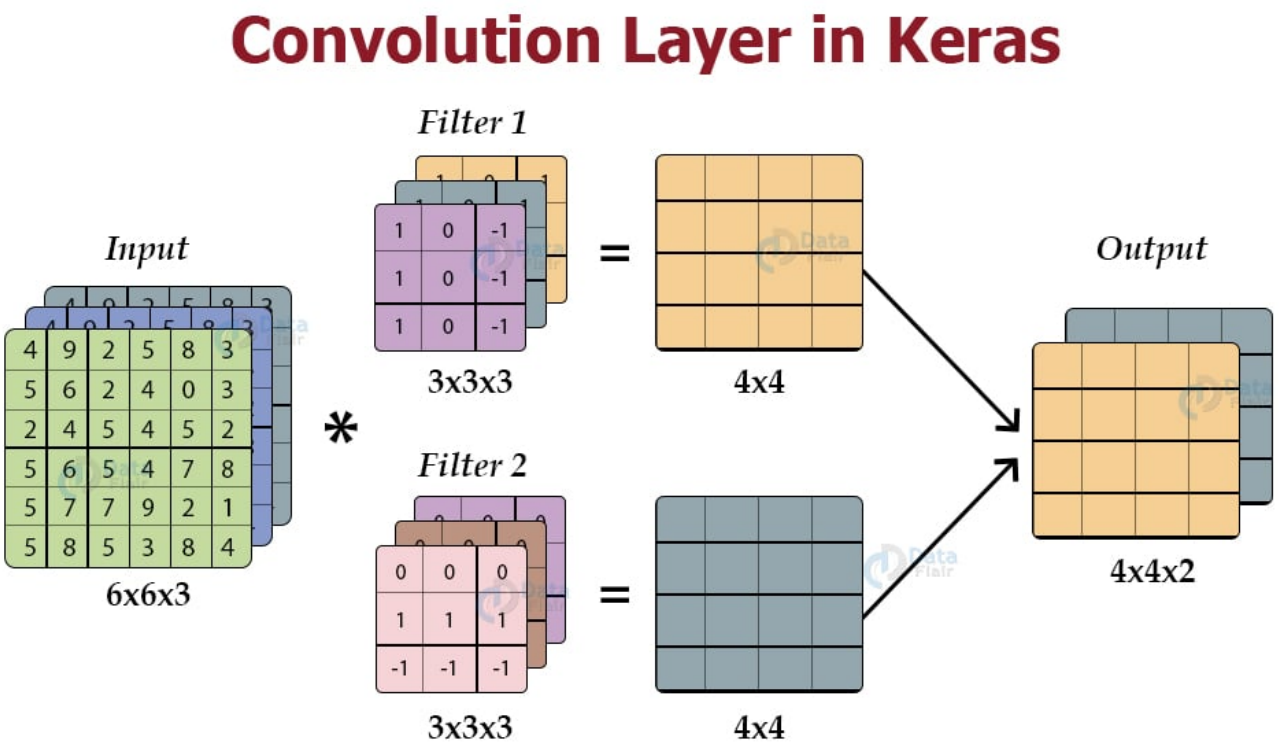
Stride (Schrittweite):

- Stride 1: Filter rutscht 1 Pixel weiter (genau).
- Stride 2: Filter springt (reduziert Bildgröße).

Padding (Randauffüllung):

- Valid: Kein Padding (Bild wird kleiner).
- Same: Rand mit Nullen füllen (Bildgröße bleibt gleich).

Convolution visuell



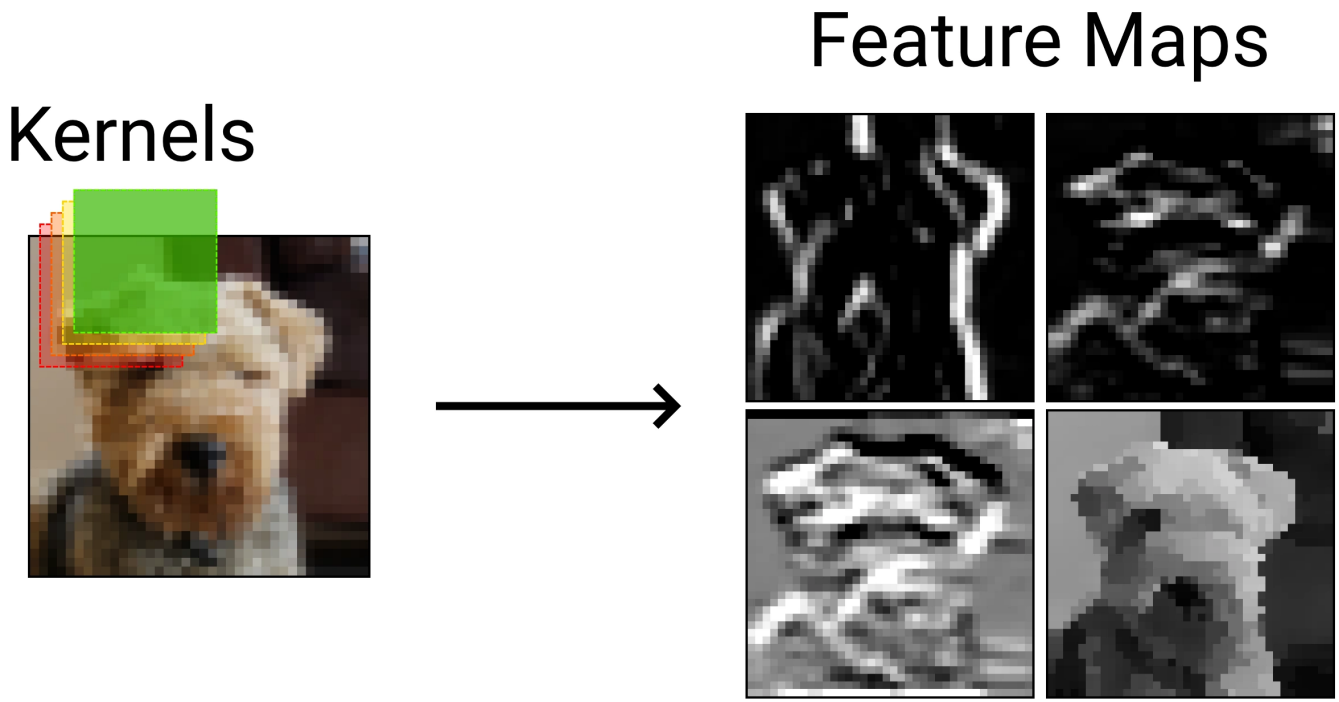
Feature Maps (Was sieht das Netz?)

Ein Filter erzeugt eine Feature Map.

Wir haben nicht nur einen Filter, sondern viele (z.B. 64 pro Layer).

Output-Dimension: Input $H \times W \times 3 \rightarrow$ Conv (64 Filter) $\rightarrow$ Output $H \times W \times 64$.

Das Bild wird "tiefer", aber behält (grob) die räumliche Struktur.



Pooling (Downsampling)

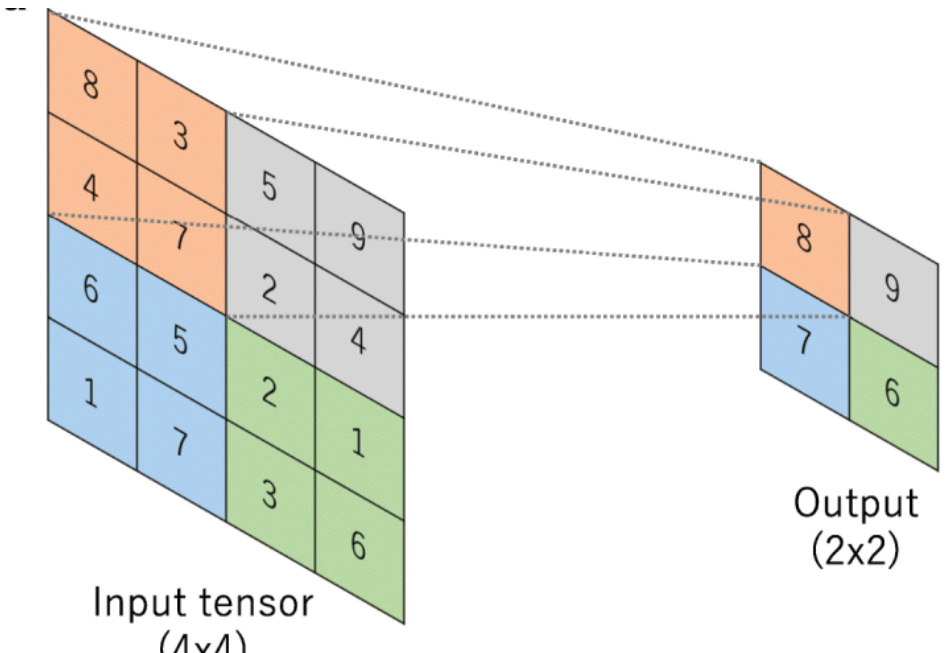
Ziel: Datenreduktion und Invarianz.

Wenn sich das Gesicht um 3 Pixel verschiebt, soll das Netz immer noch “Gesicht” sagen.

Max Pooling:Nimm ein 2×2 Fenster.

Behalte nur den größten Wert. Wirf 75% der Daten weg!

Intuition: Es ist egal wo genau im 2x2 Feld die Kante war, Hauptsache dass sie da war (Max-Wert).



Fully Connected Layer (Der Entscheidungsträger)

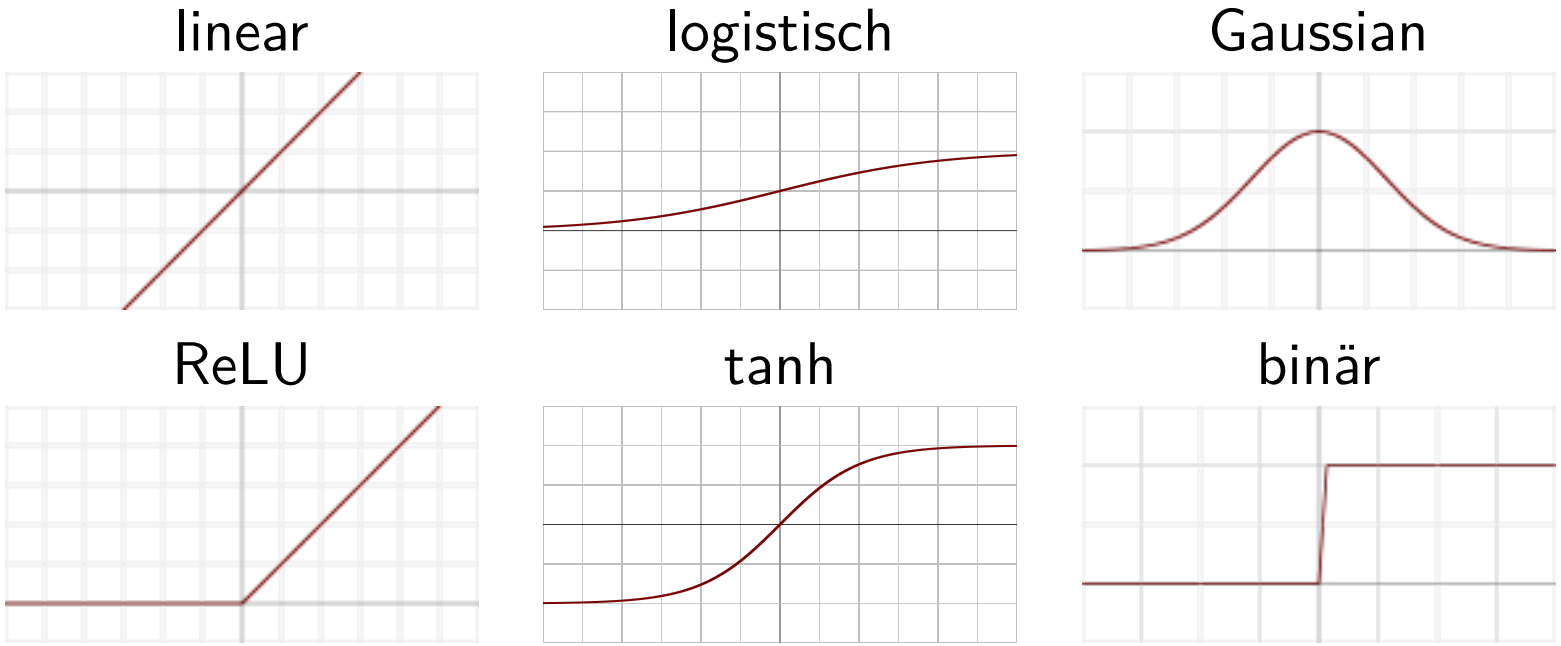
Am Ende haben wir kleine, tiefe Feature Maps (z.B. $7 \times 7 \times 512$).

Flattening: Wir machen daraus einen langen Vektor.

Dense Layer: Klassisches neuronales Netz, das diese High-Level Features verbindet.

"Wenn (Ohr da) UND (Schnurrhaar da) DANN (Katze)."

Nichtlinearität durch Aktivierungsfunktionen



SoftMax = normalisierte Exponentialfunktion $\frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$

Wandelt die rohen Zahlen (Logits) in Wahrscheinlichkeiten um (Summe = 100%).

Output: [Katze: 0.85, Hund: 0.10, Auto: 0.05]

Wie lernt das Ding? (Backpropagation Intuition)

Loss Function (Fehlerfunktion): Vergleicht Output mit Wahrheit.

Netz sagt: “Hund” (0.8). Wahrheit ist: “Katze”. Fehler ist groß!

Optimizer (SGD / Adam):

Wir müssen die Gewichte ändern, um den Fehler zu senken.

Backpropagation: Wir leiten den Fehler nach den Gewichten ab (Gradient) und gehen einen Schritt in die Gegenrichtung.

“Ändere den Filter in Layer 1 so, dass er Kanten besser findet, damit Layer 5 besser Ohren findet.”

Pyramiden-Struktur

Ein typisches CNN (z.B. VGG16) ist ein Stapel von Blöcken:

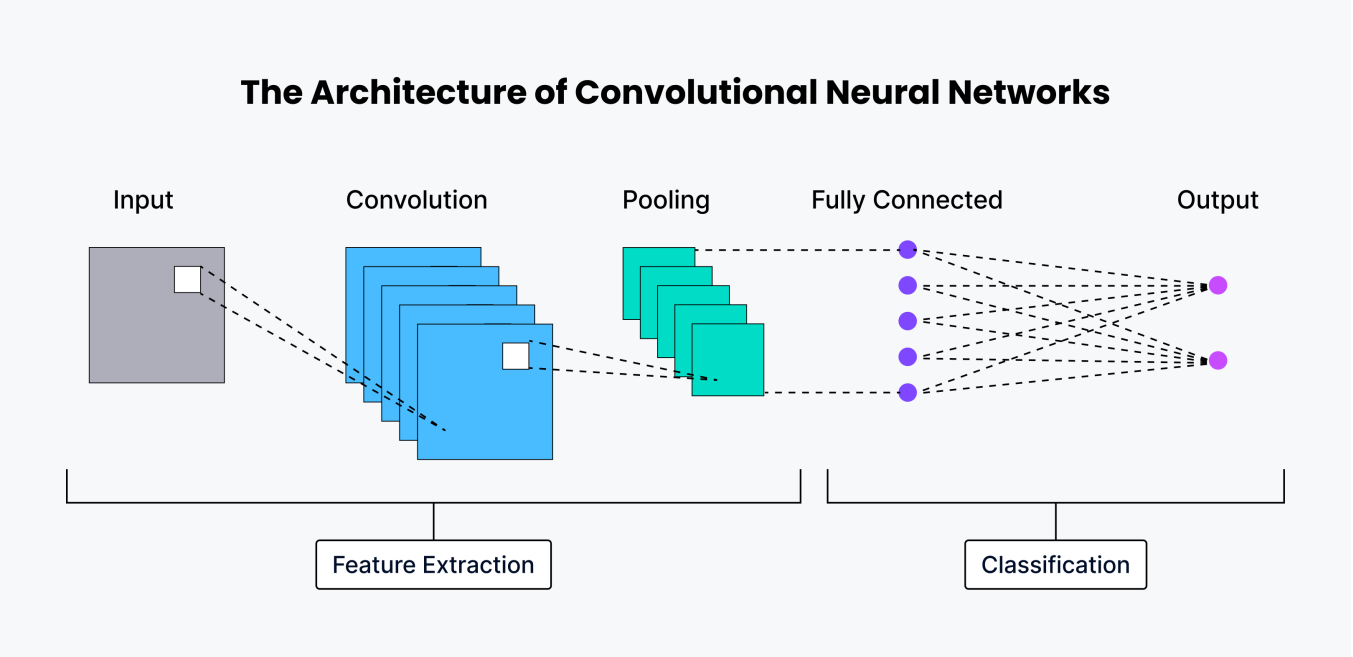
Conv + ReLU (Features finden)

Pooling (Bild kleiner machen)

Conv + ReLU (Komplexere Features finden)

Pooling

Dense Layers



Arten von CNN

2013: R-CNN - Regions with CNN Features

1. Region Proposals (Selective Search) $\rightarrow$ ~2000 Vorschläge
2. Jede Region durch CNN (AlexNet) $\rightarrow$ Feature Extraction
3. SVM für Klassifikation + Bounding Box Regression

2015: Fast R-CNN - Schneller durch Feature Sharing

1. Ganzes Bild durch CNN $\rightarrow$ Feature Map
2. Region Proposals auf Feature Map projizieren
3. ROI Pooling $\rightarrow$ feste Größe $\rightarrow$ Klassifikation + Regression

2015: Faster R-CNN - Noch schneller durch RPN

Region Proposal Network (RPN) lernt Regionen

Ethik und Erklärbarkeit von ML Modellen

Unterschiedliche Aspekte

- **Verifizierbarkeit**
  - die Möglichkeit Fehler der Modellierung und Prädiktion zu erkennen - Validierung, Missklassifikation
- **Modellierbarkeit**
  - die Möglichkeit die Struktur einer Problemstellung bzw. ihres Lösungsweges abzubilden
- **Erklärbarkeit**
  - die Entscheidungen und Schlussfolgerungen des Modells auch erklären können
- **Ethik** und **Verantwortung**
  - Ist soziales und ethisch korrektes Verhalten durch solche Algorithmen erlernbar?
  - Wer trägt die Verantwortung und juristische Haftung für Entscheidungen, welche von "künstlichen Intelligenzen getroffen werden"?

Erklärbarkeit ?

Der Algorithmus lernt "Wolf"

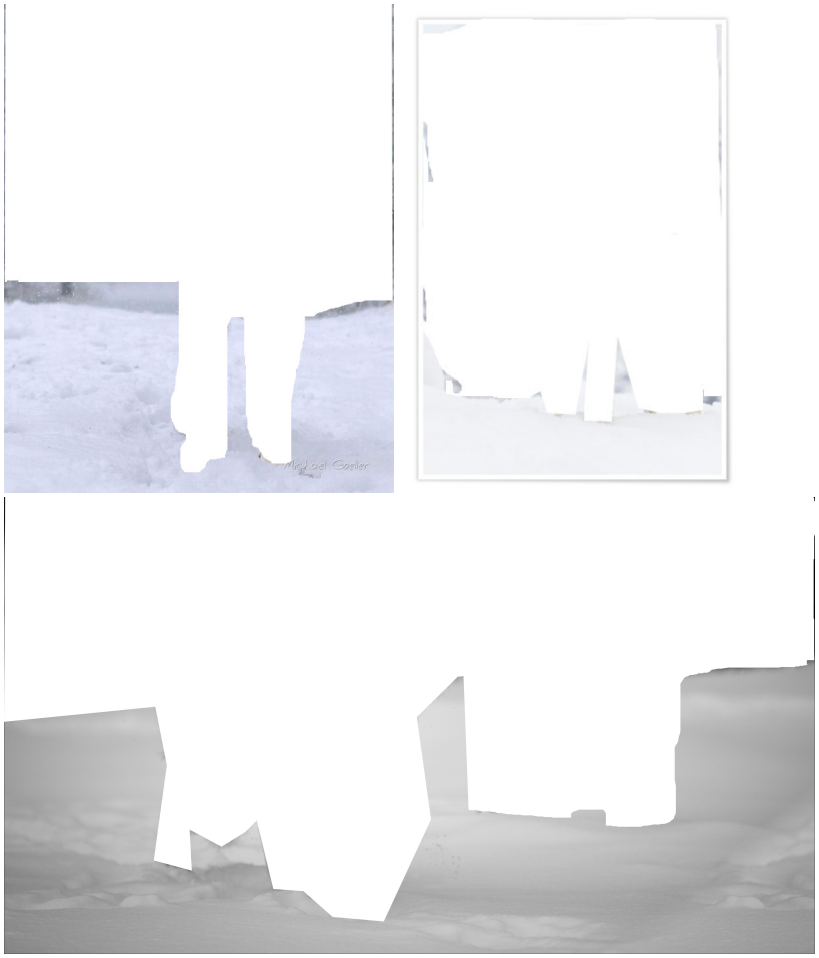
Der Algorithmus soll deklarieren

"Wolf"mit prob=0.97

WTF?

Erklärbarkeit ?

Was hat der Algorithmus gelernt?



Wie hat der Algorithmus deklariert?

"Wolf"mit prob=0.97

Rassismus bei Gesichtserkennung - NY Times (2018!)

Gender was misidentified in **up to 1 percent of lighter-skinned males** in a set of 385 photos.

Gender was misidentified in **up to 7 percent of lighter-skinned females** in a set of 296 photos.

Rassismus bei Gesichtserkennung - NY Times (2018!)

Gender was misidentified in **up to 12 percent of darker-skinned males** in a set of 318 photos.

Gender was misidentified in **35 percent of darker-skinned females** in a set of 271 photos.

Sexismus bei Gesichtserkennung - 2020

die meisten historisch berühmten weiblichen Persönlichkeiten werden als Männer klassifiziert

Data Augmentation (Datenvermehrung)

Problem: Zu wenig Daten.

Lösung: Wir generieren neue Daten aus den alten und erhalten mehr „effektive“ Trainingsdaten ohne neue Annotationen.

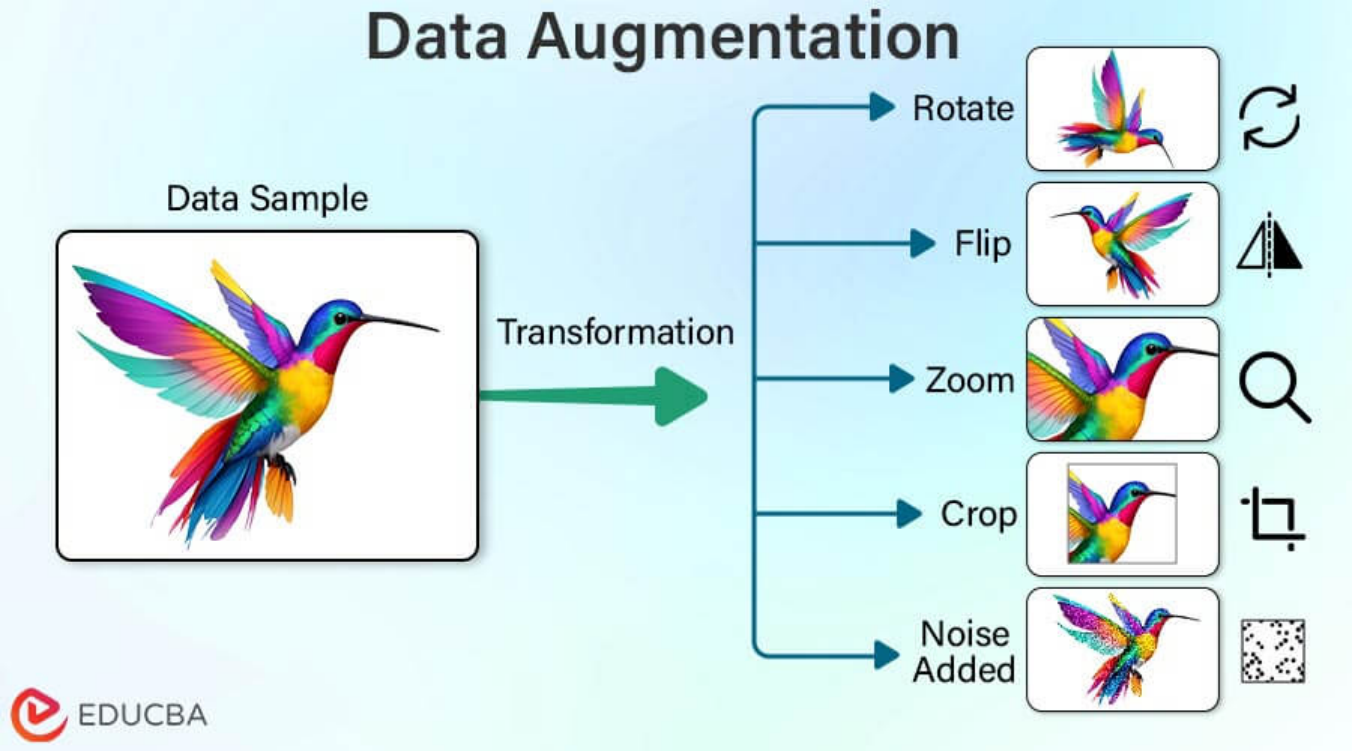
Reduziert Overfitting, verbessert Generalisierung & Robustheit

Effekt: Aus 1.000 Bildern werden virtuell 1.000.000 Variationen.

Das Netz lernt Invarianz: “Eine Katze auf dem Kopf ist immer noch eine Katze.”

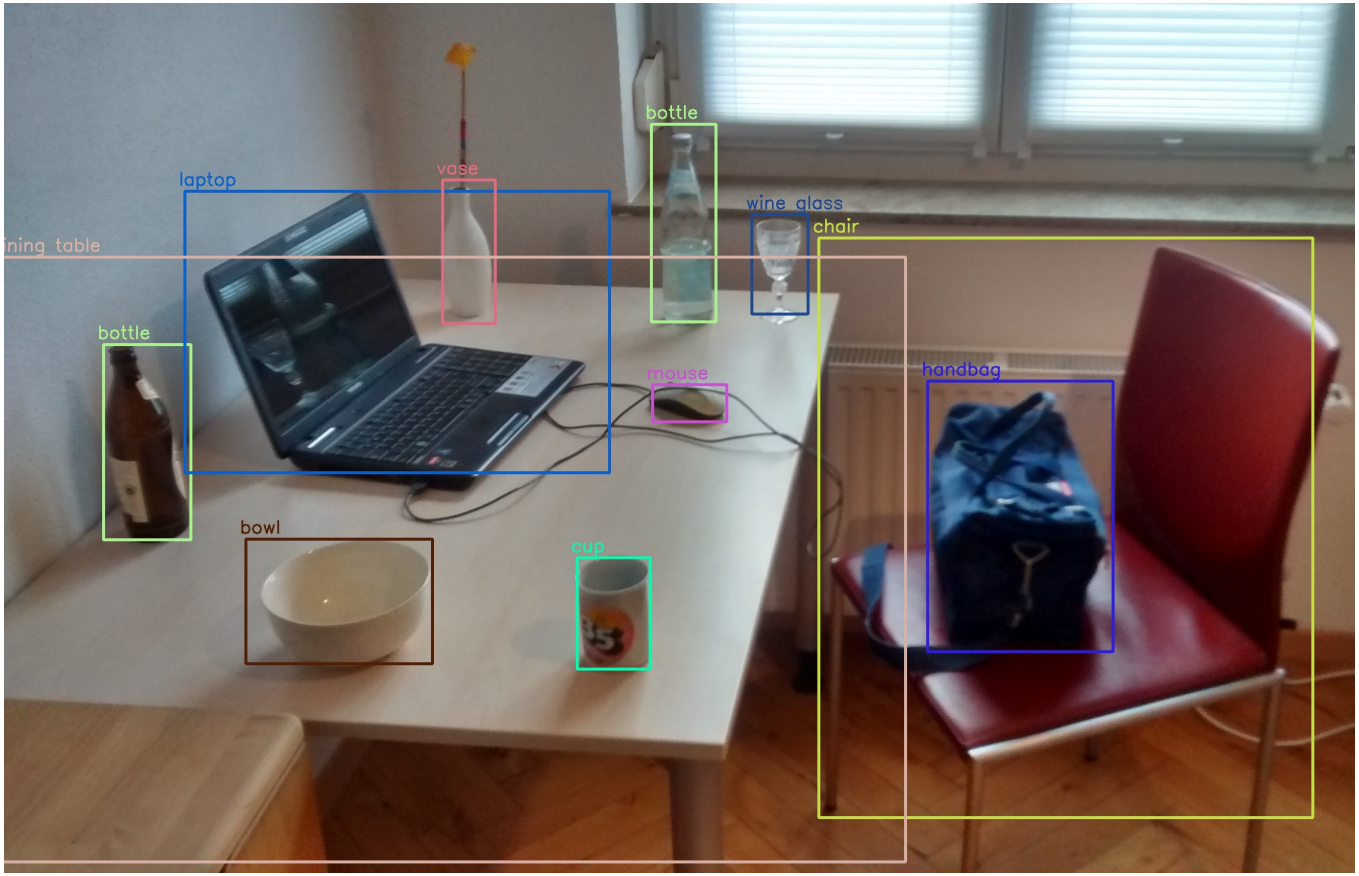
Data Augmentation

Drehen, Spiegeln (Flip), Zoomen, Helligkeit ändern, Rauschen hinzufügen.



YOLO: You Only Look Once

YOLO


bottle
laptop
vase
wine glass
chair
ining table
bottle
mouse
handbag
bowl
cup

Aufgaben in Computer Vision

Image Classification:

Frage: “Was ist auf dem Bild?” “Hund” (1 Label pro Bild)

Object Detection:

Frage: “Was ist wo?” “Hund bei (x1,y1,x2,y2), Katze bei . . . ” + Bounding Boxes + Klassen

Semantic Segmentation:

Frage: “Welches Pixel gehört zu was?”

Pixelweise Klassenzuordnung (Himmel, Straße, Auto, Fußgänger. . .)

Instance Segmentation:

Frage: “Welche Instanz gehört wohin?”

Pixelweise Zuordnung + Instanz-ID (Auto1, Auto2, Person1, Person2. . .)

Gegenüberstellung

HOG + SVM

$$\text{Bild} \rightarrow \underbrace{\text{Sliding Window}}_{\text{manuelle Features}} \rightarrow \text{HOG Features} \rightarrow \text{SVM} \rightarrow \text{Detektion}$$

Two-Stage (Faster R-CNN)

$$\text{Bild} \rightarrow \text{CNN} \rightarrow \underbrace{\text{Feature Map}}_{\textit{gelernteFeatures}} \rightarrow \text{RPN}_{\text{Region Proposal Networks}} \rightarrow$$
$$\text{ROI Pooling} \rightarrow \text{Classifier}$$

One-Stage (YOLO)

$$\text{Bild} \rightarrow \text{CNN} \rightarrow \underbrace{\text{Feature Map}}_{\textit{gelernteFeatures}} \rightarrow \text{Detektion} + \text{Klassifikation}$$

Grundidee von YOLO

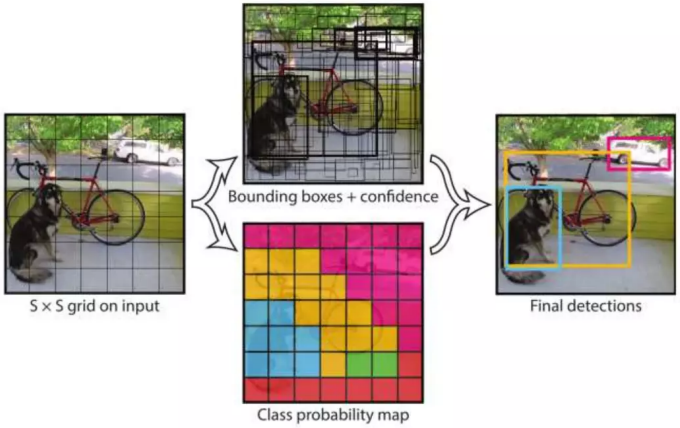
Traditionell

Bild $\rightarrow$ Viele Fenster $\rightarrow$ Klassifikation jedes Fensters $\rightarrow$ Zusammenführen

YOLO

Bild $\rightarrow$ Alle Bounding Boxes $+$ Klassen gleichzeitig

Bounding Box, Confidence and Class Probability



YOLO reframes **object detection** as a **regression** problem.

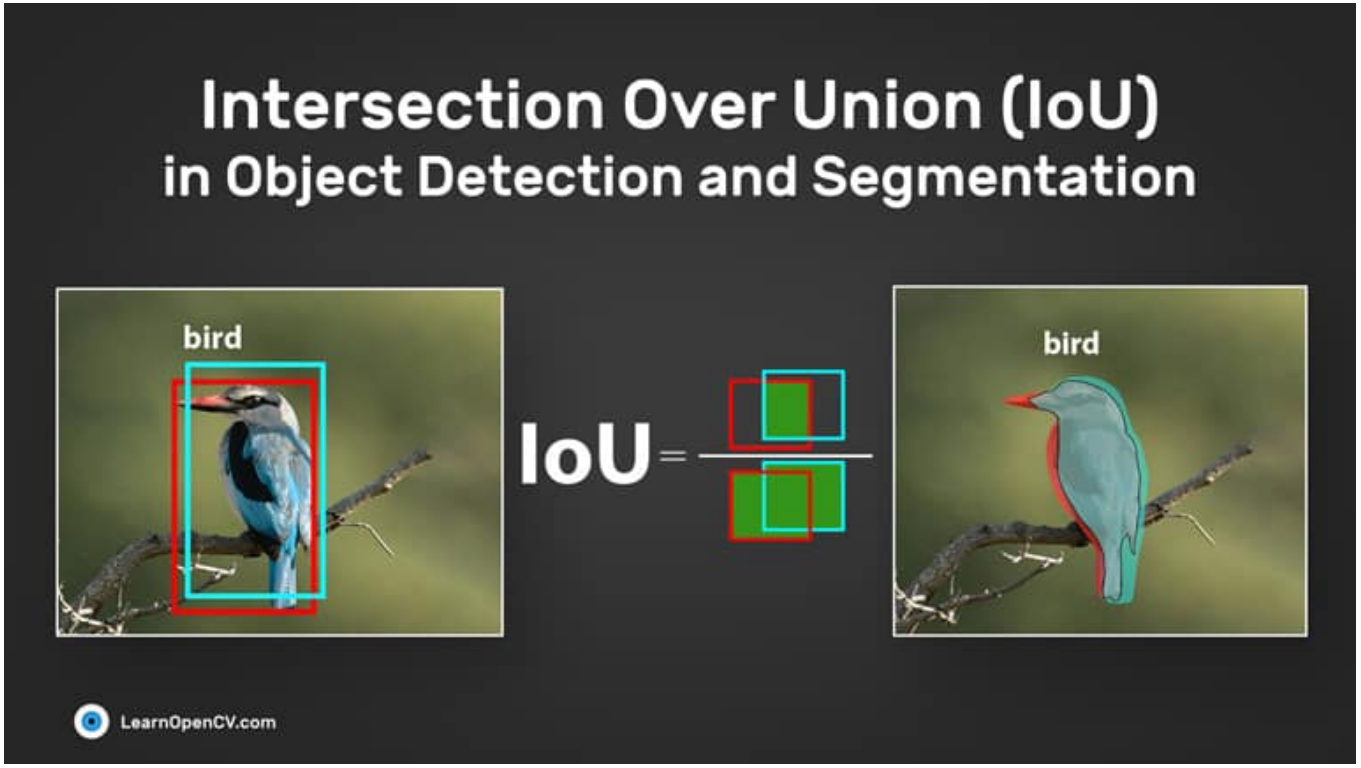
- The image is divided into an $S \times S$ grid and for each grid cell predicts B bounding boxes (x, y, w, h), confidence for those boxes, and C class probabilities.
- These predictions are encoded as an $\mathbf{S \times S \times (B * 5 + C)}$ tensor.

Confidence Score - Wie sicher ist die Detektion?

$$\text{Confidence} = P(\text{Object}) \times IoU_{\text{pred}}^{\text{truth}}$$

Pr(Object): Wahrscheinlichkeit, dass Objekt in Box ist (0 oder 1)

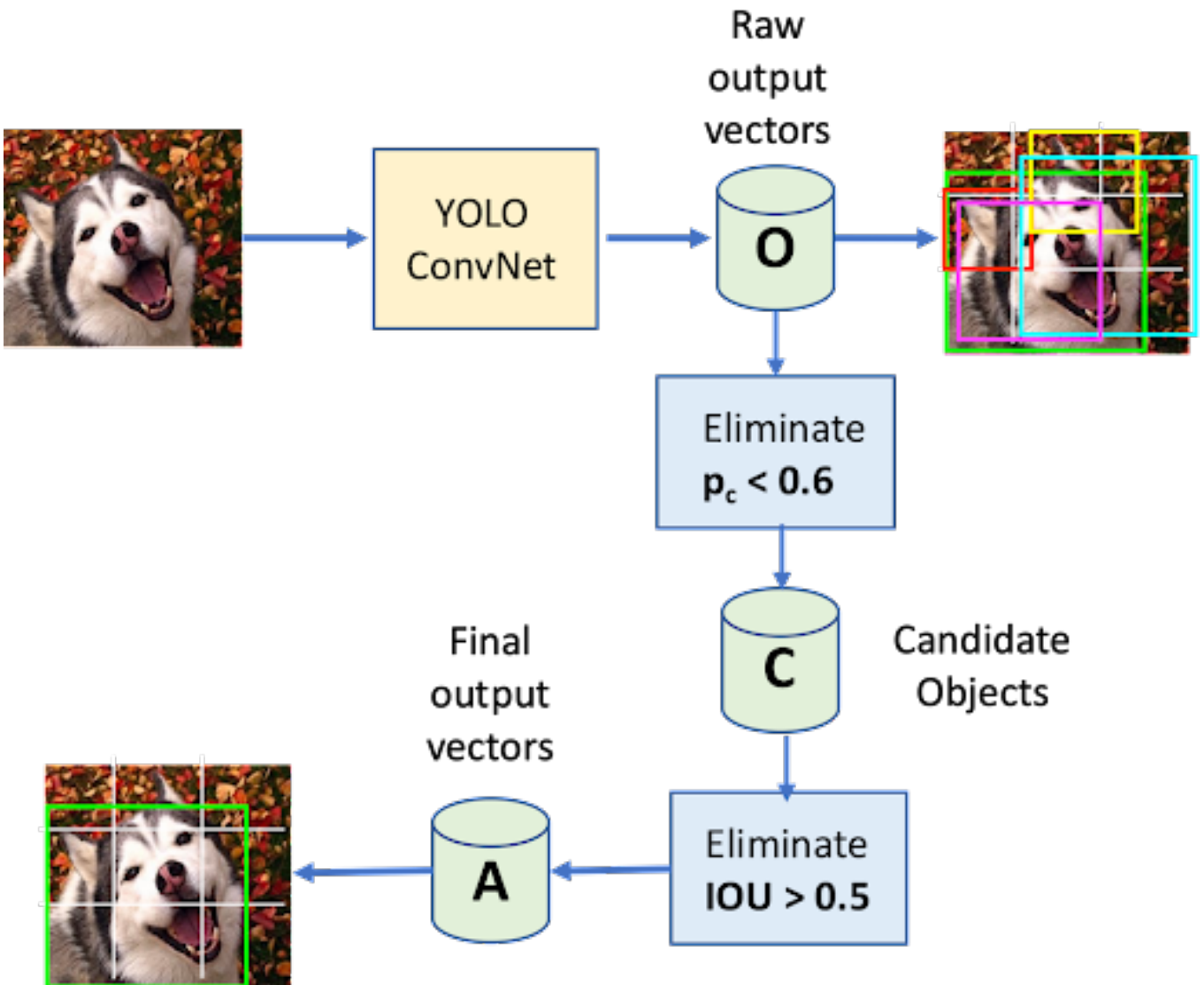
IoU: Intersection over Union (Überlappung mit Ground Truth)



Intersection over Union (IoU)

Wann ist eine Detektion "korrekt"?

Typischer Threshold: $IoU \geq 0.5$ für "korrekte" Detektion



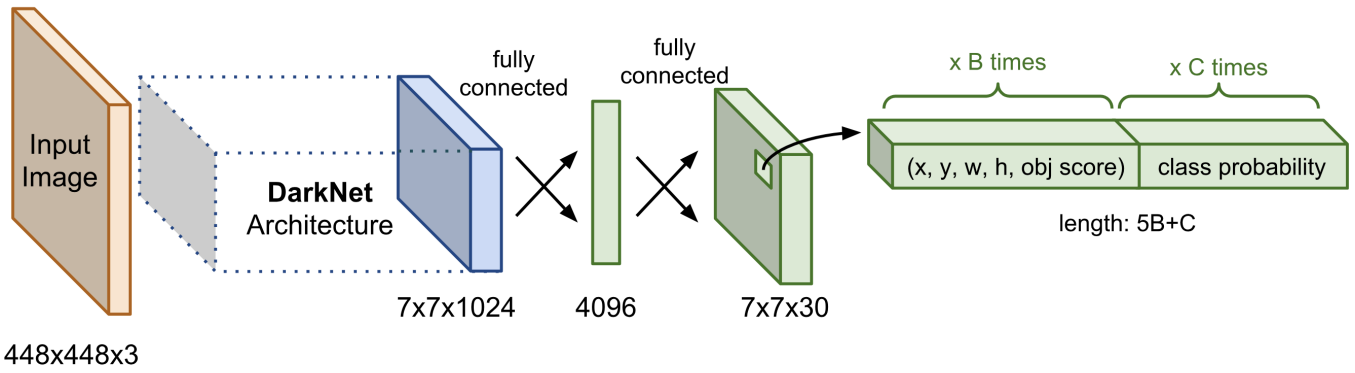
Klassenwahrscheinlichkeiten

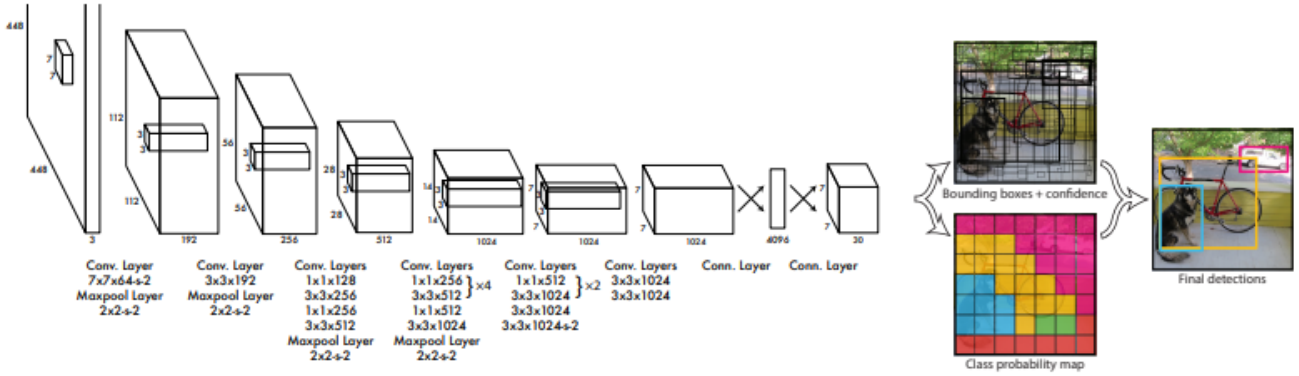
$P(Class_i|Object)$: Wahrscheinlichkeit der Klasse, gegeben dass ein Objekt vorhanden ist

Finale Wahrscheinlichkeit pro Klasse:

$$\begin{aligned} Pr(Class_i) &= Pr(Object) \times Pr(Class_i|Object) \times IoU \\ &= Confidence \times Pr(Class_i|Object) \end{aligned}$$

YOLO Architektur





YOLO Modelle im Überblick

Entwicklungstrends:

- 2016-2018: Fokus auf Echtzeitfähigkeit (v1-v3)
- 2020-2022: Trainingstechniken & Industrialisierung (v4-v7)
- 2023-2024: Multitask-Fähigkeiten & Edge-Optimierung (v8+)

Empfehlungen:

- Einführung: YOLOv5 oder YOLOv8
- Echtzeit auf Edge: YOLOv8n oder YOLOv10n
- Maximale Genauigkeit: YOLOv10x oder YOLO11x
- Produktionseinsatz: YOLOv8 (beste Dokumentation & Community)

Aktueller Stand (2024): YOLOv10 und YOLO11 bieten beste Balance aus Genauigkeit und Geschwindigkeit