FH JOANNEUM
University of Applied Sciences

Objekte finden, verfolgen, verstehen: Computer Vision praktisch

Praxisnaher Einstieg in die Grundlagen der Computer Vision mit Python

DI Dr. Hannah Wimmer, BSc.
Data Science and Artificial Intelligence
Institut für Wirtschaftsinformatik und Data Science, FH Joanneum
Eckertstraße 30i, 8020 Graz
hannah.wimmer@fh-joanneum.at

Objekte finden, verfolgen, verstehen: Computer Vision praktisch

Bevor wir anfangen…

- Was ist Ihr aktuelles **Verständnis** von Computer Vision (CV)?
- Was wären Ihrer Meinung nach **Anwendungsgebiete** der CV?
- Wie **funktioniert** CV Ihrer Meinung nach?
- Was sind **Möglichkeiten** bzw. **Grenzen** der CV?

Kursüberblick – worüber wir diskutieren werden…

1. **Einführung und Grundbegriffe**
   - Romantische Erwartungshaltung versus Realität
   - Grundlagen und Methoden der Computer Vision

2. **Klassische Fähigkeiten von CV-Modellen**
   - Bildklassifizierung: *„Welches Objekt ist am Bild zu sehen?“*

   → **Coding Beispiel:** Objektklassifizierung

   - Objektdetektion: *„Welches Objekt ist zu sehen und wo?“*

   → **Coding Beispiel:** Objektdetektion

   - Segmentierung, Pose Estimation, Oriented Bounding Boxes

   → **Coding Beispiel:** Segmentierung, Pose Estimation, OBB

3. **Objektverfolgung**
   - Verschiedene Methoden und ihre Vor- und Nachteile

   → **Coding Beispiel:** Objektverfolgung mittels ByteTrack / DeepSort

4. **Grenzen der CV**
   - Was ist möglich, und woran werden wir scheitern? Warum?

5. **Diskussion & Wrap-Up**

OBJEKTE FINDEN, VERFOLGEN, VERSTEHEN: COMPUTER VISION PRAKTISCH

1. Einführung und Grundbegriffe

- Grundidee und klassische Problemstellungen
- Romantische Erwartungshaltung versus Realität
- Methoden der Computer Vision

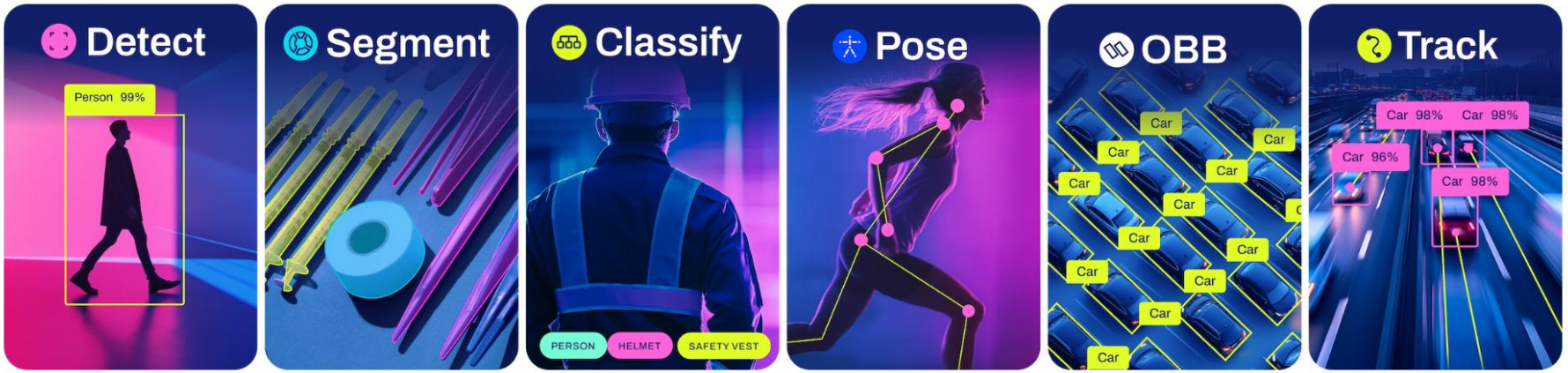
1. Was ist Computer Vision *wirklich*?

Grundidee und klassische Problemstellungen

- Informationen sind nicht immer gleich in **optimal nutzbarer Form** vorhanden
- Überwachungsvideos, Verkehrsaufnahmen, etc.: eine **manuelle** Sichtung ist bei heutigen Datenmassen **undenkbar**
- **Computer Vision:** Methoden, **automatisiert** Informationen aus Bildern oder Videoframes zu extrahieren

Problemstellungen im Bereich Computer Vision

- Objektklassifizierung: einem Bild (im Gesamten!) ein Label zuordnen
- Objekterkennung: ein Objekt im Bild erkennen und identifizieren
- Bildsegmentierung: Klassifizierung einzelner Bildbestandteile auf der Pixel-Ebene
- Posenerkennung: Erkennung und Nachverfolgung der Position und Orientierung von Körperteilen



1. Was ist Computer Vision *wirklich*?

Romantische Erwartungshaltung versus Realität

Menschliche Vorstellung vom ‚Sehen'

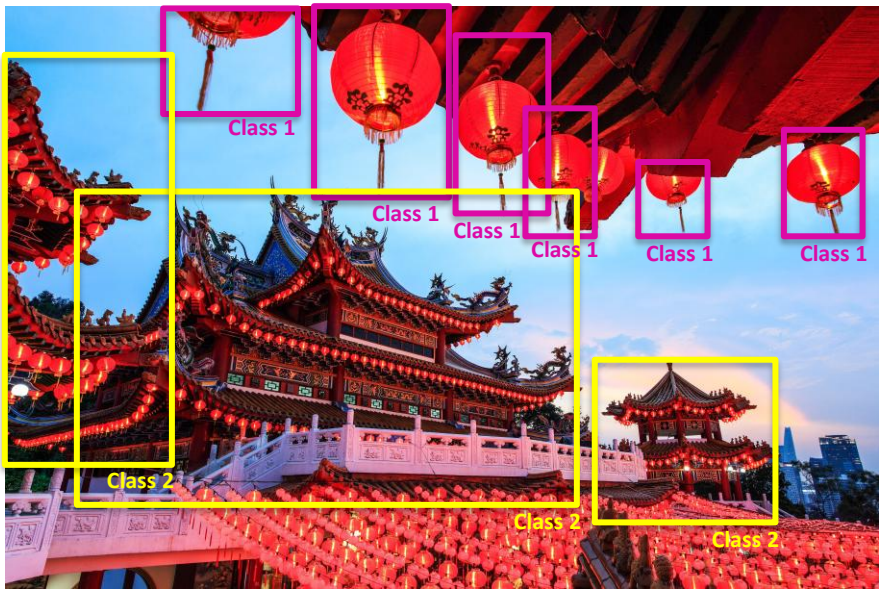
Was wir wahrnehmen...

+ Einzelne **Objekte** (Tempel, Lichter, Wolken, Hochhäuser, ..)
+ schöne Atmosphäre, gute Beleuchtung, **ästhetische Details**
+ „**Gesamteindruck**": was das Bild in uns auslöst

→ **„Die Maschine versteht Bilder" – Nein**

→ **„Die Maschine sieht, was ich sehe" – Nein**

Maschinelles ‚Sehen'



Was die Maschine wahrnimmt…

+ Pixelanordnungen und Muster, **auf die trainiert wurde**
- **Keine** generelle Wahrnehmung
- **Kein** Verständnis der Situation

1. Was ist Computer Vision *wirklich*?

Romantische Erwartungshaltung versus Realität

Kurze Anmerkung dazu:

- **Nicht** als philosophische ‚menschliche Seele versus Maschine'-Diskussion zu verstehen
- Menschliche Wahrnehmung ist hier **nichts besonderes** (wir arbeiten auch nur auf Basis eines neuronalen Netzes)
- Genau wie eine Maschine, lernen auch wir **Klassen von Objekten**, die wir später unterscheiden können:

Was ist bspw. das für ein Tier?

→ Tiger

…und das?

→ Axolotl

…und das…??

→ Fangschreckenkrebs

→ **Was wir nicht gelernt haben, können auch wir nicht zuordnen – wie die Maschine**

→ **Aber:** wir tendieren dazu, unser ‚Processing' zu romantisieren: emotionaler/ästhetischer Eindruck, Interpretation, etc.

→ Im maschinellen Sehen bleibt's **technisch**: Pixel, Muster, Klassen – keine Interpretation, keine kognitive Ebene

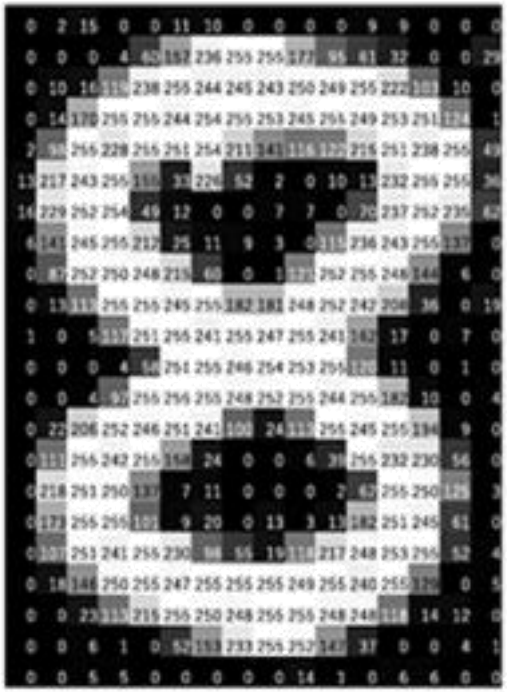
1. Was ist Computer Vision *wirklich*?

Methoden der Computer Vision

Pixel, Muster, Klassen – wie funktioniert das nun?

- Die digitale Auflösung ist endlich: jedes Bild hat eine gewisse Anzahl an Pixeln
- Jeder Pixel besitzt einen konkreten wert, klassischerweise ein Integer zwischen 0 und 255
- Der PC ‚sieht' das Pixel-Array

Entnommen aus: Understanding How Computers See Pictures

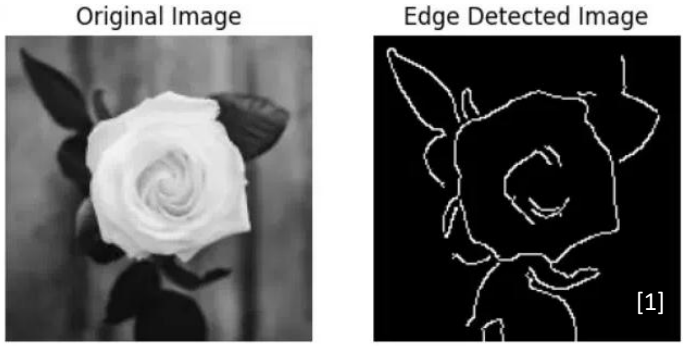


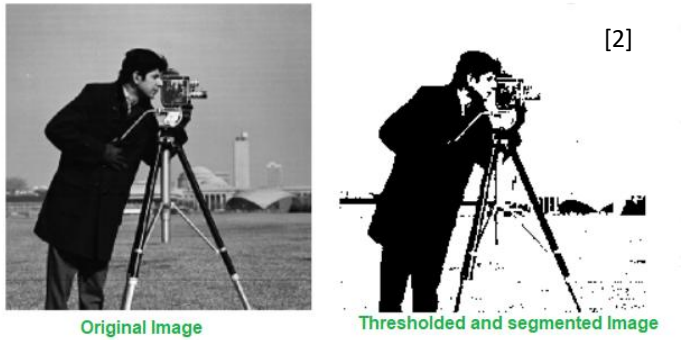
1. Was ist Computer Vision *wirklich*?

Methoden der Computer Vision

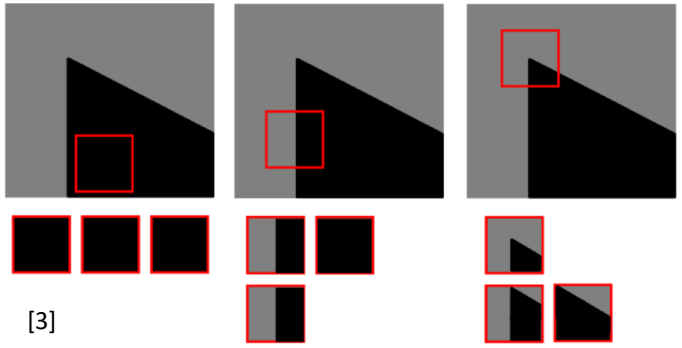
Pixel, Muster, Klassen – wie funktioniert das nun?

- Klassisch: Edge Detection, Corner Detection, Thresholding, etc.





- Methoden, die unabhängig von den Daten selbst operieren
- Verwendet, um Features für nachfolgende Klassifizierungsalgorithmen zu extrahieren
- „Regelbasiertes Machine Learning“!



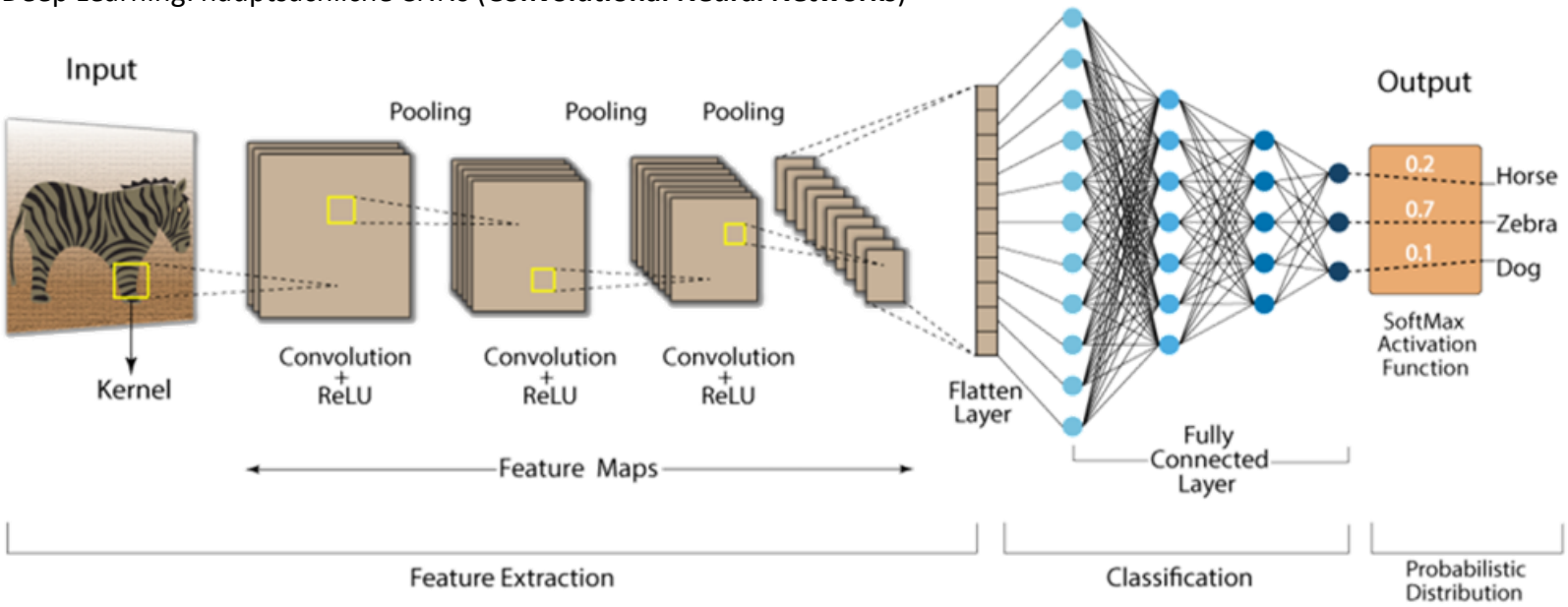
[1] What is Edge Detection in Image Processing? - GeeksforGeeks
[2] Thresholding-Based Image Segmentation – GeeksforGeeks
[3] Corner detection — Basics of Image Processing
[4] visual c++ - Scale space blob detection using opencv - Stack Overflow

1. Was ist Computer Vision *wirklich*?

Methoden der Computer Vision

Pixel, Muster, Klassen – wie funktioniert das nun?

- Deep Learning: hauptsächliche CNNs (**Convolutional Neural Networks**)



- Methoden, die Anhand von Trainingsdaten lernen (unterschiedliche Daten → unterschiedliche Modelle!)
- State-of-the-Art Resultate, allerdings oft ‚Block Box'
- Features werden nach und nach extrahiert (erst lokal, dann global), dann in späteren Layers klassiert

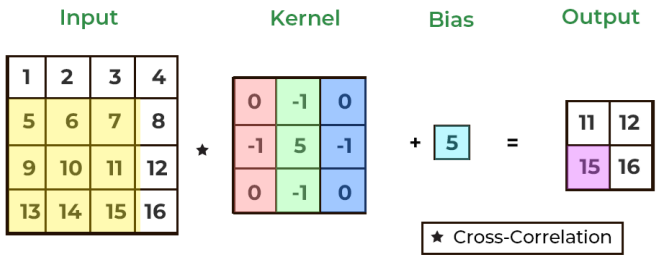
[1] Convolutional Neural Network | Deep Learning | Developers Breach

1. Was ist Computer Vision *wirklich*?

Methoden der Computer Vision

Zu Convolutional Neural Networks…

- Basieren auf wiederholten Anwendungen von a) **Convolution**, b) **Activation Function,** und c) **Pooling**
- **Convolution:** Faltung eines Inputs mit einem sog. ‚kernel'
  im Wesentlichen eine Filterung (bzw. Glättung) des Inputbildes



- **Activation Funtion**: bringt eine Nichtlinearität in das Neuronale Netz
  sorgt dafür, dass auch nichtlineare Zusammenhänge gelernt werden können
- **Pooling**: extrahiert Max oder Avg über Fenster einer gewissen Größe
  übernimmt eine Art ‚Downsampling' im Sinne der klassischen Signalverarbeitung
  komprimiert die Information über Layers

→ Im Fall von Farbbildern hat bereits der Input drei ‚**Schichten**' (rot, blau, grün)

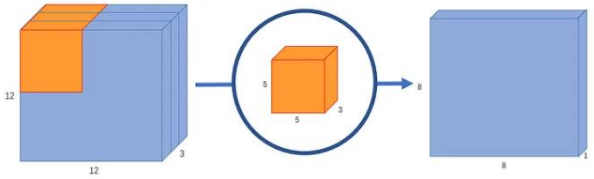


Image 4: A normal convolution with 8×8×1 output

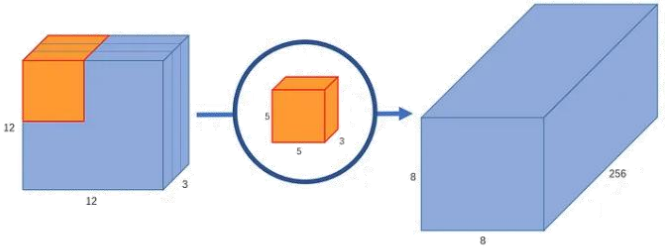


Image 5: A normal convolution with 8×8×256 output

OBJEKTE FINDEN, VERFOLGEN, VERSTEHEN: COMPUTER VISION PRAKTISCH

2. Klassische Fähigkeiten von CV-Modellen

- Die Anfänge: Objektklassifizierung
- Steigende Komplexität: Objektdetektion
- Weitere Fähigkeiten: Segmentierung, Pose Estimation, Oriented Bounding Boxes
- Die YOLO-Modellfamilie

2. Klassische Fähigkeiten von CV-Modellen

Die Anfänge: Bildklassifizierung

Eine der einfachsten (unkonkretesten) Fragestellungen:

„Welche Objektklasse ist diesem Bild zuzuordnen?“

→ **Katze**

→ **Löwe**

→ **Fuchs**

Was uns interessiert:

- Die **Objektklasse** für das **gesamte Bild** (Katze, Löwe, Fuchs)

Was uns (noch) nicht interessiert:

- **Wo genau** am Bild sich die Objekte befinden
- **Wie viele** Objekte sich am Bild befinden

2. Klassische Fähigkeiten von CV-Modellen

Die Anfänge: Objektklassifizierung

Eine der einfachsten (unkonkretesten) Fragestellungen:

„Welche Objektklasse ist diesem Bild zuzuordnen?“

→ **Katze**

→ **Löwe, Löwe**

→ **Fuchs**

Wie kann Objektklassifizierung mit einem CNN aussehen:

- **Input**: Farbbild mit Breite w und Höhe h in drei ‚Farbkanälen‘ (für rot, blau, grün). **Beispiel**: $\text{input} = (\underbrace{1080}_{h}, \underbrace{1920}_{w}, \underbrace{3}_{Farbkanäle})$
- **Output**: Vektor mit zugeordneten Wahrscheinlichkeiten in der Länge aller dem Modell bekannten Klassen
  Beispiel: für ein Modell, das auf drei Objektklassen trainiert wurde: $\text{output} = (\underbrace{0.98}_{Katze}, \underbrace{0.01}_{Hund}, \underbrace{0.01}_{Vogel})$

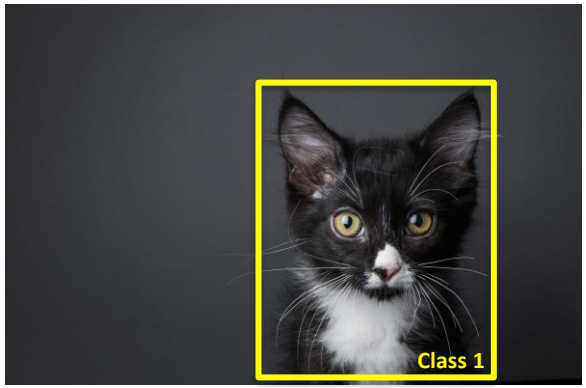
- **Coding Beispiel: Objektklassifikation**

2. Klassische Fähigkeiten von CV-Modellen

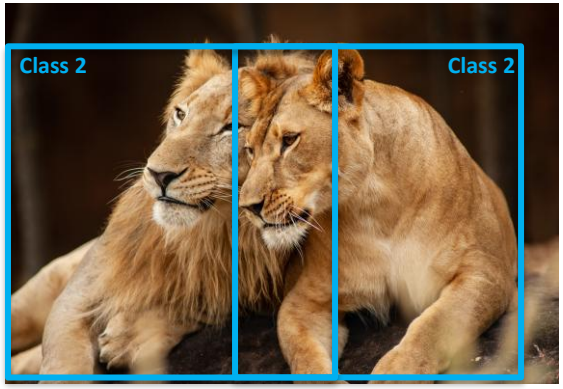
Steigende Komplexität: Objektdetektion

Oft hätten wir gerne mehr Information:

„Welche Objekte befinden sich auf diesem Bild und wo?“



→ **Katze + Box**



→ **Löwe + Box, Löwe + Box**

→ **Fuchs + Box**

Was uns interessiert:

- Die **Objektklasse** (Katze, Löwe, Fuchs)
- Die dazugehörige **Bounding Box** (kleinstes Rechteck, das das Objekt einschließt)
- Wichtig: erneut für **jedes** erkannte Objekt am Bild

Was uns (noch) nicht interessiert:

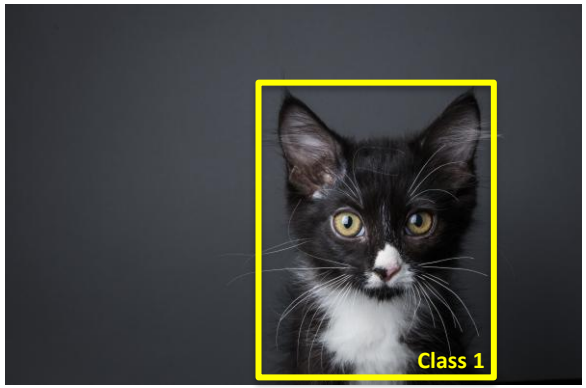
- **Welche individuellen Pixel** am Objekt ‚beteiligt‘ sind

2. Klassische Fähigkeiten von CV-Modellen

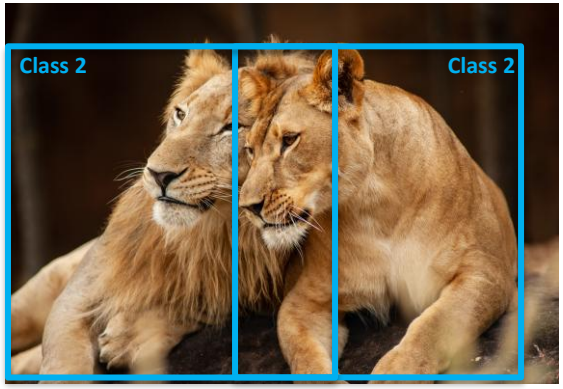
Steigende Komplexität: Objektdetektion

Oft hätten wir gerne mehr Information:

„Welche Objekte befinden sich auf diesem Bild und wo?“



→ **Katze + Box**



→ **Löwe + Box, Löwe + Box**

→ **Fuchs + Box**

Wie kann Objektdetektion mit einem CNN aussehen:

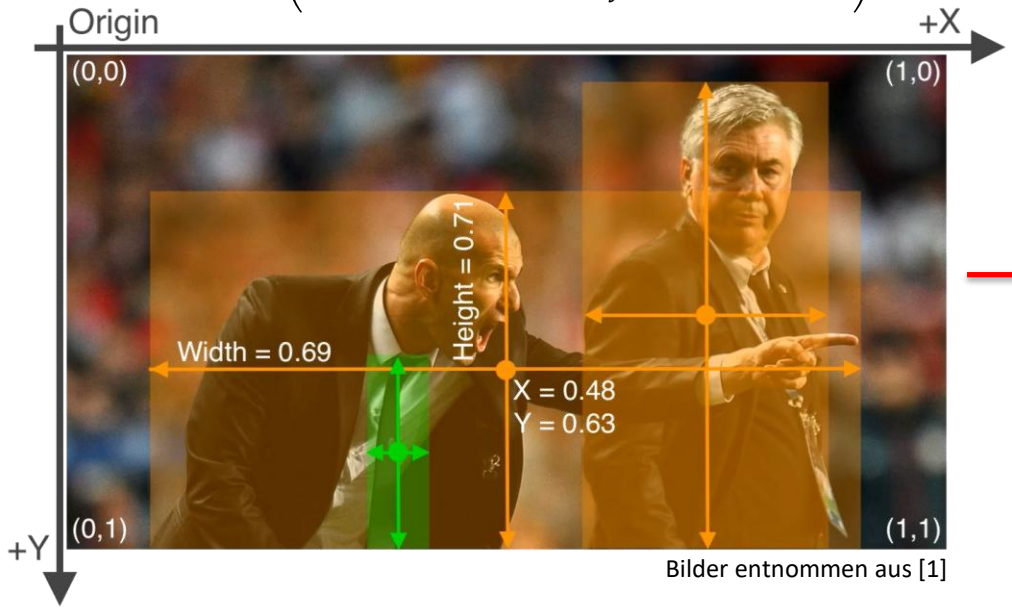
- **Input**: Farbbild mit Breite w und Höhe h in drei ‚Farbkanälen‘ (für rot, blau, grün). **Beispiel**: $\text{input} = (\underbrace{1080}_{h}, \underbrace{1920}_{w}, \underbrace{3}_{Farbkanäle})$
- **Output:** Tupel mit (**Klassenlabel, x_center, y_center, width, height)** für jedes einzelne erkannte Objekt.
  Beispiel: $\text{output} = \left(\underbrace{1}_{Katze}, \underbrace{0.65}_{\substack{Box\\Mittelpunkt\\x}}, \underbrace{0.65}_{\substack{Box\\Mittelpunkt\\y}}, \underbrace{0.4}_{\substack{Box\\Breite}}, \underbrace{0.8}_{\substack{Box\\Höhe}} \right)$

2. Klassische Fähigkeiten von CV-Modellen

Steigende Komplexität: Objektdetektion

Wie kann Objektdetektion mit einem CNN aussehen:

- **Input**: Farbbild mit Breite w und Höhe h in drei ‚Farbkanälen' (für rot, blau, grün). **Beispiel**: $\text{input} = (\underbrace{1080}_{h}, \underbrace{1920}_{w}, \underbrace{3}_{Farbkanäle})$
- **Output:** Tupel mit (**Klasse, x_center, y_center, width, height)** für jedes einzelne erkannte Objekt.
  Beispiel: $\text{output} = \begin{pmatrix} \underbrace{1}_{Katze} & , & \underbrace{0.65}_{\substack{Box\\Mittelpunkt\\x}} & , & \underbrace{0.65}_{\substack{Box\\Mittelpunkt\\y}} & , & \underbrace{0.4}_{\substack{Box\\Breite}} & , & \underbrace{0.8}_{\substack{Box\\Höhe}} \end{pmatrix}$



Bilder entnommen aus [1]

zidane.txt

```
0  0.481719 0.634028 0.690625 0.713278
0  0.741094 0.524306 0.314750 0.933389
27 0.364844 0.795833 0.078125 0.400000
```

[1] Überblick über Objekterkennungs-Datensätze – Ultralytics YOLO-Dokumentation

- **Coding Beispiel: Objektdetektion**

2. Klassische Fähigkeiten von CV-Modellen

Weitere Fähigkeiten: Segmentierung, Pose Estimation, Oriented Bounding Boxes

‚Raffiniertere' Entscheidungen: **Bildsegmentierung**

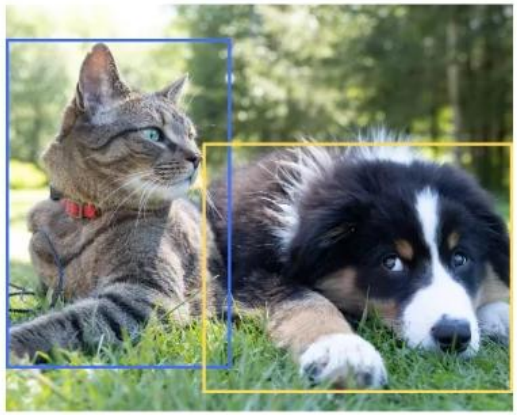
„Welche Objekte befinden sich auf diesem Bild, und welcher Bildbereich ist ihnen zuzuordnen?"

Welches Objekt?

↓

Objektklassifizierung

Welches Objekt und wo?



↓

Objektdetektion

Welches Objekt und in welchem Pixel?

Bild adaptiert aus [1]

↓

Bildsegmentierung

Was uns interessiert:

- Die **Objektklasse** aller erkannten Objekte
- Die dazugehörige **Pixelmaske** (relevant wenn konkrete Größe/Form/Ausdehnung von Bedeutung für uns ist)

Was uns (noch) nicht interessiert:

- Die **Orientierung/Positionierung** der Objekte im Raum

[1] Image segmentation detailed overview [Updated 2024] | SuperAnnotate

2. Klassische Fähigkeiten von CV-Modellen

Weitere Fähigkeiten: Segmentierung, Pose Estimation, Oriented Bounding Boxes

‚Raffiniertere' Entscheidungen: **Bildsegmentierung**

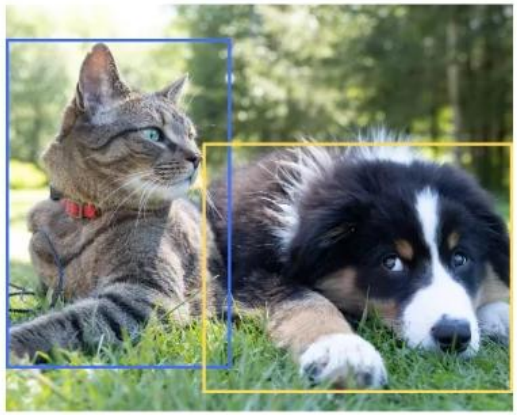
„Welche Objekte befinden sich auf diesem Bild, und welcher Bildbereich ist ihnen zuzuordnen?"

Welches Objekt?

↓

Objektklassifizierung

Welches Objekt und wo?



↓

Objektdetektion

Welches Objekt und in welchem Pixel?

Bild adaptiert aus [1]

↓

Bildsegmentierung

Wie kann Bildsegmentierung mit einem CNN aussehen:

- **Input**: Farbbild mit Breite w und Höhe h in drei ‚Farbkanälen' (für rot, blau, grün). **Beispiel**: $\text{input} = (\underbrace{1080}_{h}, \underbrace{1920}_{w}, \underbrace{3}_{Farbkanäle})$
- **Output: Masken / Kontouren** jedes erkannten Objekts, **Objektklasse**, **confidence_score**

[1] Image segmentation detailed overview [Updated 2024] | SuperAnnotate

- **Coding Beispiel: Bildsegmentierung**

2. Klassische Fähigkeiten von CV-Modellen

Weitere Fähigkeiten: Segmentierung, Pose Estimation, Oriented Bounding Boxes

‚Raffiniertere' Entscheidungen: **Pose Estimation**

„Welche Objekte befinden sich am Bild und wie ist ihre Positionierung?"

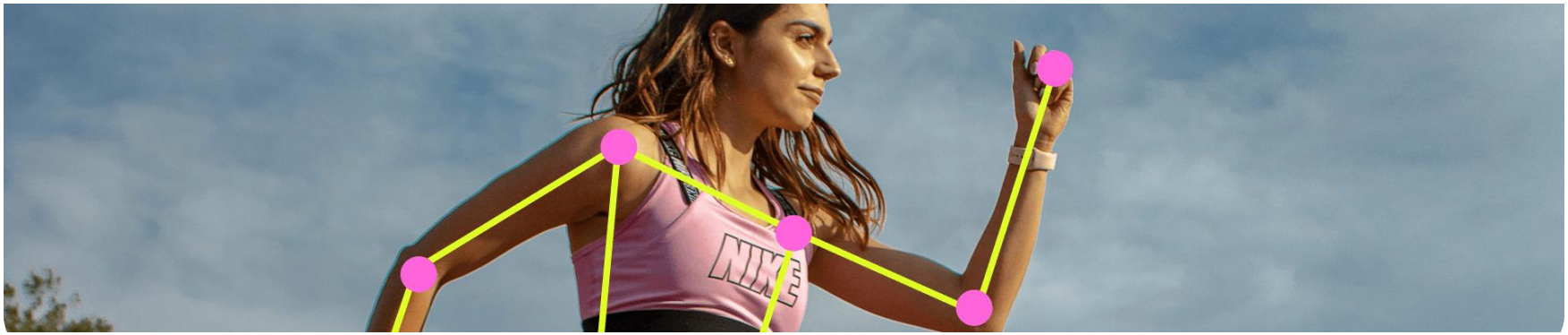


Bild adaptiert aus [1]

Was uns interessiert:

- Die **Objektklasse** aller erkannten Objekte
- Die dazugehörige **Bounding Box** (kleinstes Rechteck, das das Objekt einschließt)
- Die **Position** von **17 anatomischen Orientierungspunkten:**

Nase	Linkes/Rechtes Auge	Linkes/Rechtes Ohr
Linke/Rechte Schulter	Linker/Rechter Ellbogen	Linkes/Rechtes Handgelenk
Linke/Rechte Hüfte	Linkes/Rechtes Knie	Linker/Rechter Knöchel

Was uns nicht interessiert:

- Hier ist die Objektklasse nicht explizit dabei
- Die Modelle werden aber auf konkrete Anatomien (Personen, Hunde, etc.) trainiert – das funktioniert dann ohnehin nur bei diesen ‚Objekten' gut…

[1] Pose Estimation - Ultralytics YOLO Docs

2. Klassische Fähigkeiten von CV-Modellen

Weitere Fähigkeiten: Segmentierung, Pose Estimation, Oriented Bounding Boxes

‚Raffiniertere' Entscheidungen: **Pose Estimation**

„Welche Objekte befinden sich am Bild und wie ist ihre Positionierung?"

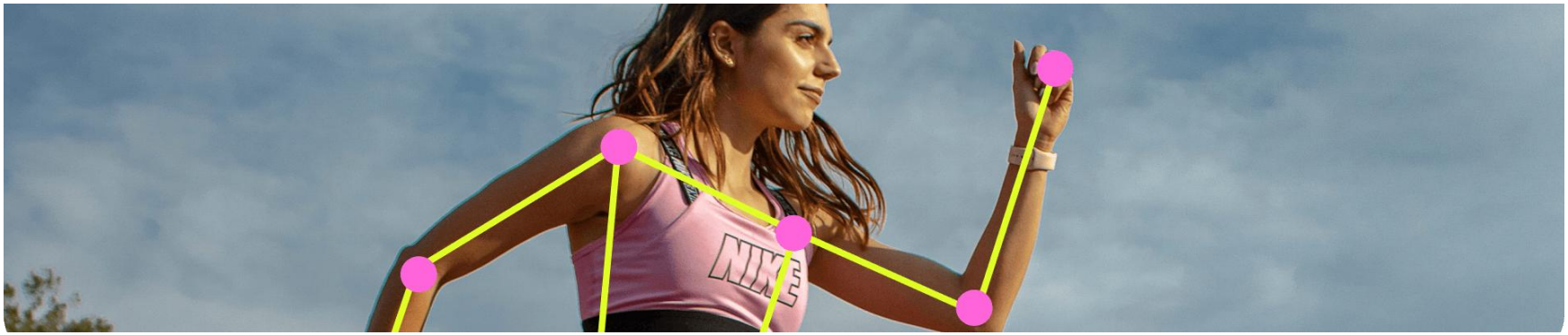


Bild adaptiert aus [1]

Wie kann Bildsegmentierung mit einem CNN aussehen:

- **Input**: Farbbild mit Breite w und Höhe h in drei ‚Farbkanälen' (für rot, blau, grün).
  Beispiel: $\text{input} = \left(\underbrace{1080}_{h}, \underbrace{1920}_{w}, \underbrace{3}_{Farbkanäle}\right)$
- **Output:** Position [x,y] für jeden anatomischen Orientierungspunkt, sowie die dazugehörigen confidence_scores
- Kommentar: es gibt oft auch noch zusätzlich eine **visibility_flag,** die Informationen über die Sichtbarkeit einzelner Punkte gibt.
  (0: nicht sichtbar und nicht gelabelt, 1: nicht sichtbar aber gelabelt, 2: sichtbar und gelabelt)

[1] Pose Estimation - Ultralytics YOLO Docs

- **Coding Beispiel: Pose Estimation**

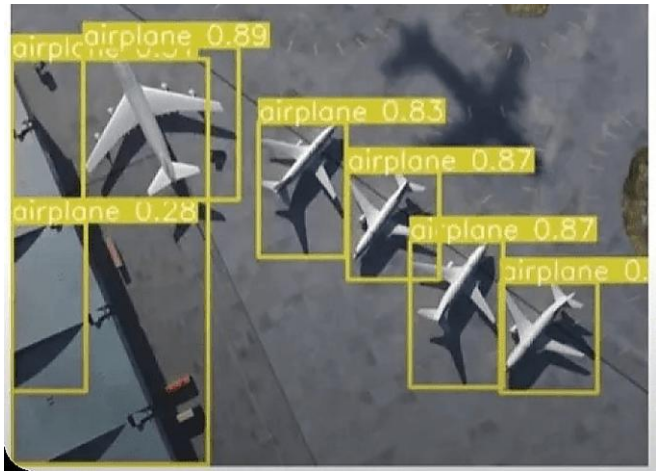
2. Klassische Fähigkeiten von CV-Modellen

Weitere Fähigkeiten: Segmentierung, Pose Estimation, Oriented Bounding Boxes

‚Raffiniertere' Entscheidungen: **Oriented Bounding Boxes**

- **Frage:** Ist Ihnen bisher ein ‚Shortcoming' in den Fähigkeiten aufgefallen? Was könnten wir noch wollen?

Standard Bounding Boxes (Objektdetektion)



Oriented Bounding Boxes (OBB)

Bild adaptiert aus [1]

- Die Bounding Boxes bisher waren **gerade ausgerichtete** Rechtecke
- Für Analysen von Satellitenaufnahmen ist das **suboptimal**, da die Objekte hier beliebig ausgerichtet sein können

[1] YOLO11 OBB Detection Guide | Ultralytics

2. Klassische Fähigkeiten von CV-Modellen

Weitere Fähigkeiten: Segmentierung, Pose Estimation, Oriented Bounding Boxes

‚Raffiniertere' Entscheidungen: **Oriented Bounding Boxes (OBB)**

Wie kann OBB mit einem CNN aussehen:

- **Input**: Farbbild mit Breite w und Höhe h in drei ‚Farbkanälen' (für rot, blau, grün).

  Beispiel: $\text{input} = (\underbrace{1080}_{h}, \underbrace{1920}_{w}, \underbrace{3}_{Farbkanäle})$

- **Output:** Position [x,y] jedes Eckpunktes der Bounding Box (point-based OBB)
- **Kommentar:** es gibt noch andere Varianten, OBB zu repräsentieren:
  - ***θ*-based OBB:** (Mittelpunktkoordinaten, Höhe, Breite, Drehung)
  - ***h*-based OBB:** über Höhe/Breite der Box (trigonometrische Überlegungen..)

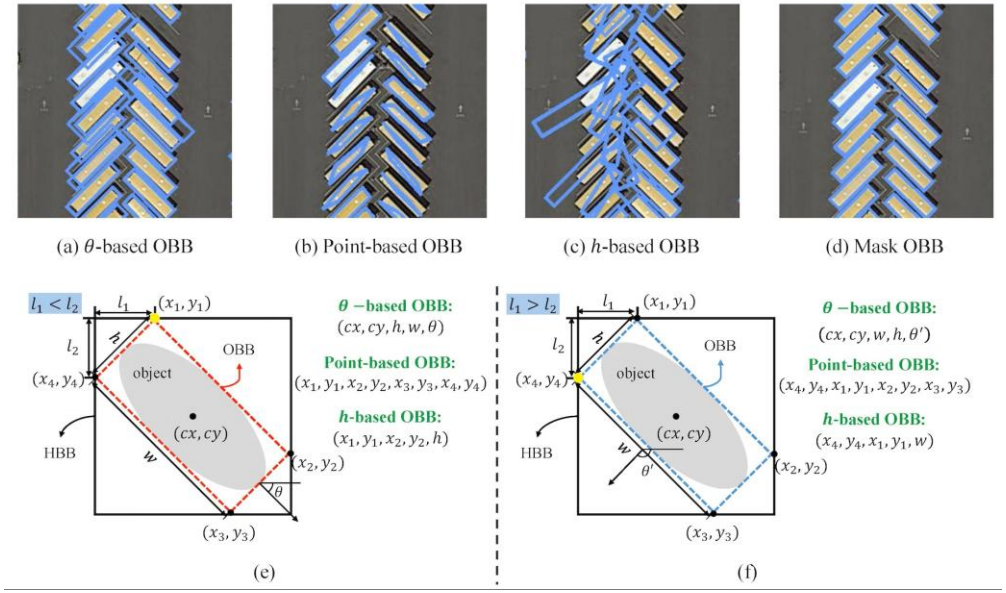


Bild aus [1]

[1] Oriented Bounding Box (OBB) Datasets Overview - Ultralytics YOLO Docs

- **Coding Beispiel: Oriented Bounding Boxes (OBB)**

2. Klassische Fähigkeiten von CV-Modellen
Die YOLO-Modellfamilie

Wir haben jetzt durchgehend YOLO-Modelle verwendet. **Was ist das eigentlich?**

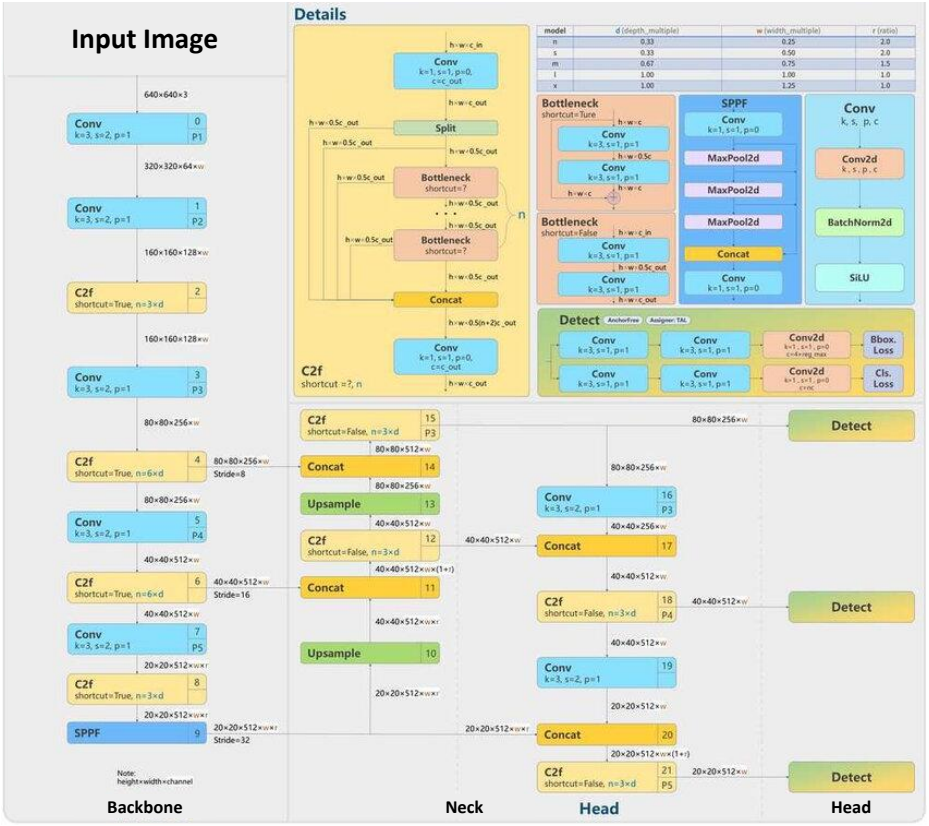
- Mittlerweile: zahlreiche vortrainierte CV-Modelle frei verfügbar
- Eine der bekanntesten (erfolgreichsten) Modellfamilien: **YOLO-Modelle**

Die YOLO-Modellfamilie

- **Name: „**You Only Look Once“: Bilder werden in einem einzigen „Pass-Through“ verarbeitet
- **Fähigkeiten:**
  - schnell und leistungsstark
  - Architektur ermöglicht Echtzeitanwendungen
  - Detektion, Klassifikation, Segmentierung, Pose Estimation, OBB, Tracking, etc.
- **Historisches**
  - Erstes Modell (YOLO) 2015 entwickelt
  - mittlerweile bereits YOLO12 (Version 12) verfügbar
  - immer in verschiedenen Größen (# Parameter) verfügbar: nano, small, medium, large, extra-large
- **Anwendungsgebiete:** Selbstfahrende Autos, Bildgebende Verfahren, Security, Robotik, etc.

2. Klassische Fähigkeiten von CV-Modellen

Die YOLO-Modellfamilie



Adaptiert von [1]

- **Grobe Unterteilung**: Backbone, Neck, Head
- **Backbone**: Sequenz von
  - **Convolutional Layers:** Feature Extraction (Downsampling, Filter iterativ verdoppelt)
  - **Cross-Stage Partial (CSP) Layers:** Feature Diversität (robustere Features, verringert die Gefahr von Vanishing Gradients)
- **Neck**: Fügt einzelne Outputs der Backbone Layers wieder zusammen (außerdem: Upsampling)
  - Weitere Conv. / CSP Layers
- **Head**: Aufspaltung in mehrere Prädiktionspipelines
  - Bounding-Box Regression
    x, y: Mittelpunktkoordinaten der Box
    w, h: Breite bzw. Höhe der Box
  - Objektklassifizierung
    Auto, Radfahrer, etc.

Input: Bild (z.B. 1920 × 1020 × k) mit drei Farbkanälen k
Output: [x, y, w, h, confidenceScore, classID]

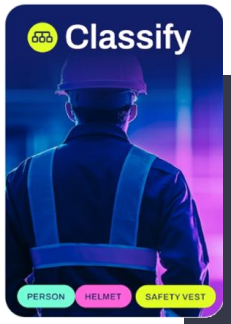
[1] Telaumbanua, A. P. H., Larosa, T. P. ., Pratama, P. D. ., Fauza, R. H. ., & Husein, A. M. (2023). „Vehicle Detection and Identification Using Computer Vision Technology with the Utilization of the YOLOv8 Deep Learning Method“. *Sinkron : Jurnal Dan Penelitian Teknik Informatika,* 7(4), 2150-2157. https://doi.org/10.33395/sinkron.v8i4.12787

2. Klassische Fähigkeiten von CV-Modellen

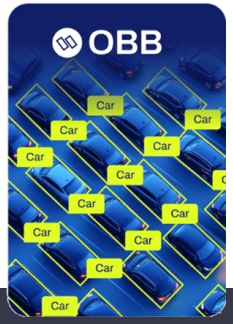
Die YOLO-Modellfamilie

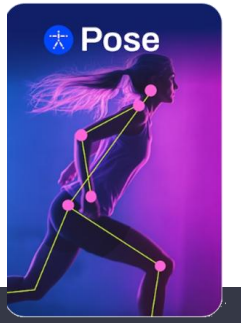
Je nach Funktionalität: Variation in der Architektur…
(nur bei Klassifizierung straight-forward nachvollziehbar..)

```
      (dfl): DFL(
        (conv): Conv2d(16, 1, kernel_size=(1, 1), stride=(1, 1), bias=False)
      )
    )
  )
)
```

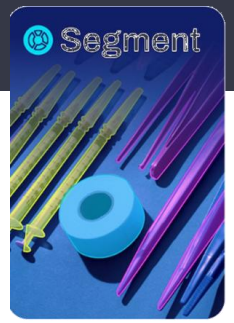



```
    )
  )
  (10): Classify(
    (conv): Conv(
      (conv): Conv2d(256, 1280, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(1280, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (act): SiLU(inplace=True)
    )
    (pool): AdaptiveAvgPool2d(output_size=1)
    (drop): Dropout(p=0.0, inplace=True)
    (linear): Linear(in_features=1280, out_features=1000, bias=True)
  )
)
```






```
  (1): Conv(
    (conv): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn): BatchNorm2d(32, eps=0.001, momentum=0.03, affine=True, track_running_stats=True)
    (act): SiLU(inplace=True)
  )
  (2): Conv2d(32, 32, kernel_size=(1, 1), stride=(1, 1))
)
```




```
      (2): Sequential(
        (0): Conv(
          (conv): Conv2d(256, 51, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (bn): BatchNorm2d(51, eps=0.001, momentum=0.03, affine=True, track_running_stats=True)
          (act): SiLU(inplace=True)
        )
        (1): Conv(
          (conv): Conv2d(51, 51, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (bn): BatchNorm2d(51, eps=0.001, momentum=0.03, affine=True, track_running_stats=True)
          (act): SiLU(inplace=True)
        )
        (2): Conv2d(51, 51, kernel_size=(1, 1), stride=(1, 1))
      )
    )
  )
)
```

OBJEKTE FINDEN, VERFOLGEN, VERSTEHEN: COMPUTER VISION PRAKTISCH

3. Objektverfolgung (Object Tracking)

- Was ist Objektverfolgung?
- Welche Methoden gibt es?

3. Objektverfolgung

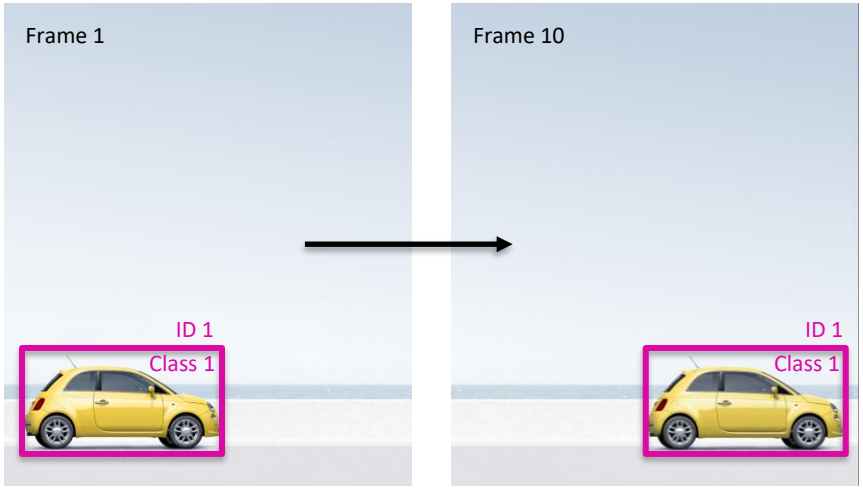
Was ist Object Tracking, und wie funktioniert es?

- Bisher haben wir uns mit **einzelnen Bildern** beschäftigt
- Im Normalfall sind wir aber daran interessiert, **Videos** automatisiert zu analysieren (Überwachungsaufnahmen, Verkehrsaufnahmen, Sportaufzeichnungen, etc.)

Die bisher besprochenen Methoden und Modelle lassen sich **Großteils** auf ‚fortlaufende Bilder' (Videos) anwenden.

➢ **Frage**: wo könnten wir allerdings ein Problem bekommen, wenn wir von Bildern zu Videos wechseln?

→ Beim Erkennen von Objekten, die **über mehrere Frames hinweg** im Bild sind.



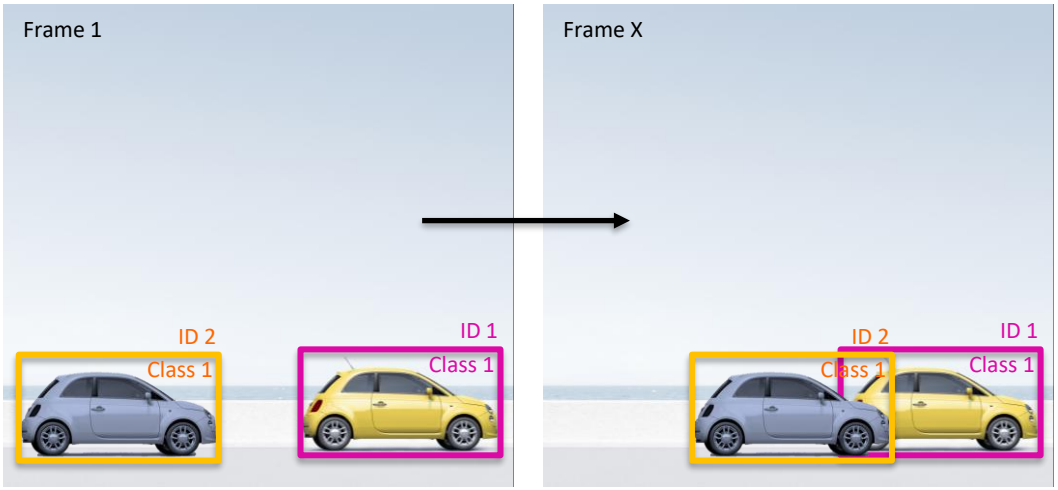
- **Objektdetektion:** „Dort ist ein Auto"
  - Erkennt ein Auto (+Bounding Box) in **jedem Frame**
  - Behandelt jedes Frame **unabhängig** vom anderen
  - Keine Wahrnehmung einer **Bewegung (!)**
    Das Modell weiß nicht, dass es dasselbe Auto ist!
- **Objektverfolgung:** „Dasselbe Auto hat sich bewegt"
  - Detektierte Objekte bekommen eine ID (hier ID 1)
  - In jedem Frame: neues oder bekanntes Objekt?
    → neue ID / Zuweisung bestehender ID

3. Objektverfolgung

Was ist Object Tracking, und wie funktioniert es?

Die Problematik:

- Jedem detektierten Objekt (Bounding Box!) muss eine **eindeutige** ID zugewiesen werden
- Die IDs dürfen nicht versehentlich vertauscht werden (bspw. beim Überholen →)
- Die Fortbewegung der Bounding Boxes muss **plausibel** sein



Ideen zur Umsetzung:

- Wir vergleichen konsekutive Frames $(t, t-1)$
- Für jede Bounding Box im einen Frame suchen wir nun eine ‚matching' Box im anderen Frame!

Methoden

- (Betrachtung des Euklidischen Abstands, der Größe, des Klassenlabels, … zwischen Paaren von Bounding Boxen)
- Betrachtung der **Intersection Over Union** (Überschneidung) zwischen Paaren von Bounding Boxen
- Betrachtung einer **Deep Association Metric** (Bewegungsinformation + visuelle Features des Objekts)

[1] A complete overview of Object Tracking Algorithms in Computer Vision & Self-Driving Cars

3. Objektverfolgung

Was ist Object Tracking, und wie funktioniert es?

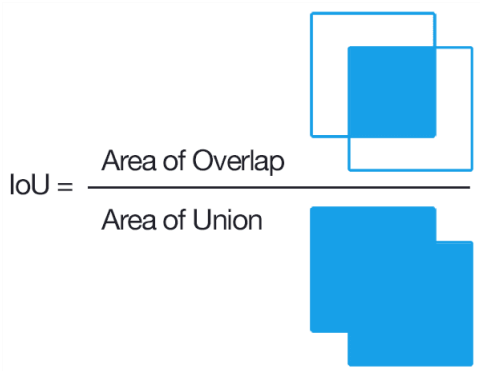
Methoden

- (Betrachtung des Euklidischen Abstands, der Größe, des Klassenlabels, … zwischen Paaren von Bounding Boxen)
- Betrachtung der **Intersection Over Union** (IoU; Überschneidung) zwischen Paaren von Bounding Boxen
- Betrachtung einer **Deep Association Metric** (Bewegungsinformation + visuelle Features des Objekts)

Zwei Beispiele: BYTETrack und Deep SORT

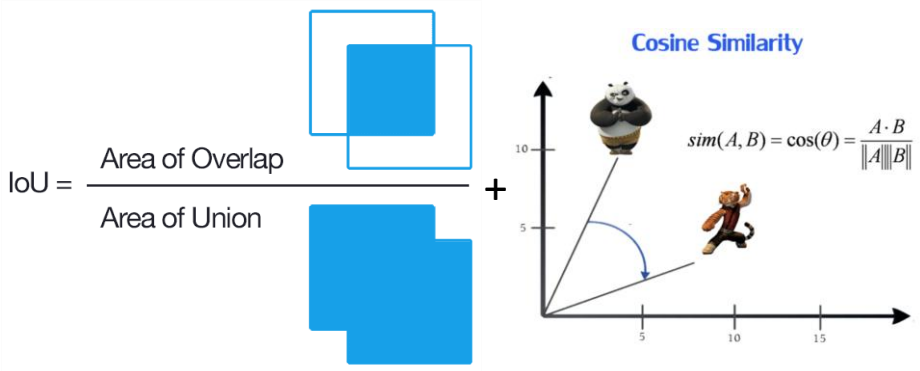
BYTETrack[1]: IoU-basieres Tracking

➢ Prozentuelle Überschneidung der Bounding Boxes in konsekutiven Frames (Bewegung)



Deep SORT[2]: Deep-Association-Metric-basiertes Tracking

➢ IoU (Bewegung) + Objektfeatures (visuelle Ähnlichkeit)



[1] An Introduction to BYTETrack: Multi-Object Tracking by Associating Every Detection Box | Datature Blog
[2] Deep SORT: Realtime Object Tracking Guide

- **Coding Beispiel: Objektverfolgung mittels BYTETrack / Deep SORT**

OBJEKTE FINDEN, VERFOLGEN, VERSTEHEN: COMPUTER VISION PRAKTISCH

4. Grenzen der Computer Vision

- Was ist möglich?
- Woran werden wir scheitern?

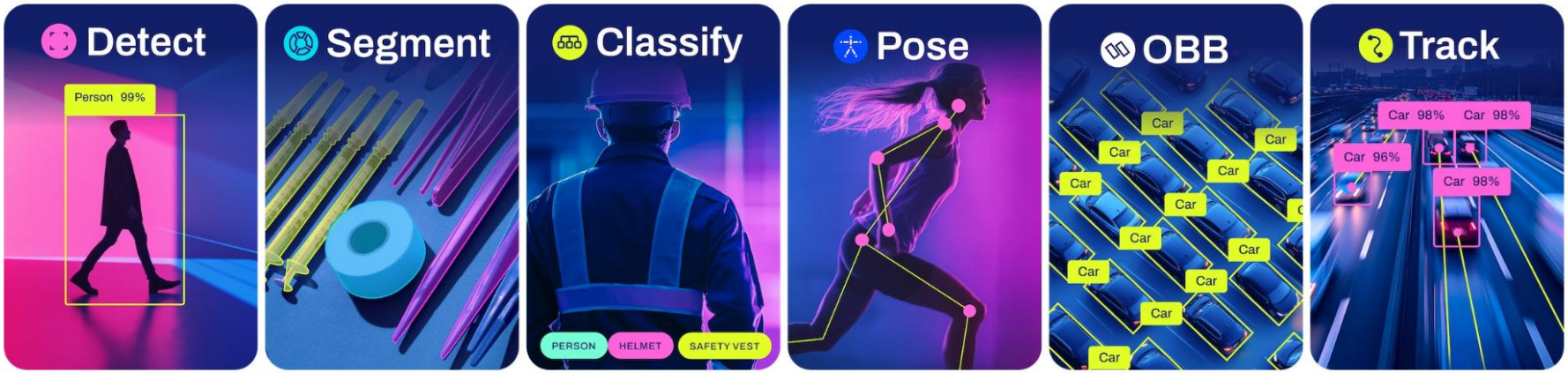
4. Grenzen der Computer Vision

Die Möglichkeiten auf einen Blick

Was CV-Modelle können (zusammengefasst):

- Objekte, **auf deren Erkennung trainiert wurde**, auf einem Bild erkennen (Objektklassifikation)
- Die **Position** von Objekten, auf die trainiert wurde, erkennen (Objektdetektion – „Bounding Boxes“)
- Eine zu einem erkannten Objekt gehörige **Pixelmaske** identifizieren (Segmentierung)
- Die **Posen** von Personen (je nach Trainingsdatensatz auch Hunden, etc.) erkennen (Pose Estimation)
- Die Position von **räumlich unterschiedlich ausgerichteten** Objekten erkennen (Oriented Bounding Boxes, OBB)
- Etc.

→ Viele Fähigkeiten, die im Normalfall von **unterschiedlich trainierten** Modellen abgedeckt werden.



4. Grenzen der Computer Vision

Die Möglichkeiten auf einen Blick

Zentrale Limitationen:

- Worauf nicht trainiert wurde, kann nicht erkannt werden

- Wenige Trainingsdaten → schlechte Performance

- Garbage in, garbage out

- **Kein „Röntgenblick“**

OBJEKTE FINDEN, VERFOLGEN, VERSTEHEN: COMPUTER VISION PRAKTISCH

5. Diskussion und Wrap-Up