

# Videoanalyse und Objekttracking

1<sup>st</sup> Bartolovic Eduard  
Hochschule München  
München, Deutschland  
eduard.bartolovic0@hm.edu

2<sup>nd</sup> Thomas Willeit  
Hochschule München  
München, Deutschland  
XXXXX@hm.edu

3<sup>rd</sup> Schäfer Julia  
Hochschule München  
München, Deutschland  
j.schaefer0@hm.edu

## Zusammenfassung—

### I. VIDEOMATERIAL

Für dieses Projekt wurden mehrere Aufnahmen aufgenommen. Ziel war es keine zu chaotischen aber auch nicht zu einfache Aufnahmen zu erstellen. Auch sollten verschiedene Szenarien abgebildet werden.

#### Verkehrsaufnahmen am Brudermühlertunnel am Mittleren Ring:

Dieser Teil des Mittleren Rings ist sehr verkehrsreich und Autobahnähnlich ausgebaut. Es gibt mehrere Fahrspuren. Alle Fahrzeuge fahren im Sichtfeld in eine Richtung. Es sind ein paar Laternen und Bäume in der Aufnahme die die Sicht etwas beeinträchtigen. Es sind nur Pkw im Datensatz enthalten. Teilweise missachten Fahrzeuge den Sicherheitsabstand und fahren sehr nah aufeinander auf. Das Video hat 60 Bilder pro Sekunde und hat eine Bildauflösung  $3840 \times 2160$ . Es wurde kein Stativ verwendet sondern nur die interne Stabilisierung des Smartphones genutzt.



Abbildung 1. Verkehrsaufnahmen am Brudermühlertunnel

#### Verkehrsaufnahmen am Canditunnel:

Eine stark befahrene Straße mit einer vielzahl von Verkehrsteilnehmern. Es herrscht zwei Richtungsverkehr mit insgesamt sechs Fahrspuren. In eine Richtung stehen die Fahrzeuge in einem Stau. Durch Regen spiegeln viele Oberflächen. Die Bäume bewegen sich aufgrund von stärkeren Wind und verdecken einen Teil der Fahrbahn. Diese Aufnahme ist die Anspruchsvollste. In diesem Video sind neben Pkw auch Busse enthalten. Das Video hat 60 Bilder pro Sekunde und hat eine Bildauflösung  $3840 \times 2160$ . Es wurde kein Stativ verwendet sondern nur die interne Stabilisierung des Smartphones genutzt.

#### Zugaufnahmen an der Donnersbergerbrücke:



Abbildung 2. Verkehrsaufnahmen am Canditunnel.

Eine Aufnahme der einer der am stärksten befahrenen Bahnstrecken Europas [1]. Hier existiert viel Zugverkehr auf mehreren Gleisen mit unterschiedlichen Zugtypen. Beobachtet werden die vier Gleise der zwei Bahnsteige der S-Bahnhalts Donnersbergerbrücke. Ein Hindernis sind die Masten, Signale und das Bahnsteigdach welche einen guten Blick auf die Züge erschweren. In dieser Aufnahme sind Züge und Fußgänger enthalten. Der Fokus liegt aber bei den Zügen. Das Video hat 30 Bilder pro Sekunde und hat eine Bildauflösung  $1920 \times 1080$ . Es wurde ein Stativ verwendet.



Abbildung 3. Zugaufnahmen an der Donnersbergerbrücke

### II. KONZEPT

Ziel des Projektes ist es klassische Verfahren mit neueren DeepLearning Methoden zu vergleichen. Dafür ist für beide Arten eine Pipeline geschaffen worden.

Für die Klassische Methoden wurde für die Objekterkennung das Gausmixture Verfahren verwendet und für das Tracking den SORT Algorithmus. Für die DeepLearning Verfahren

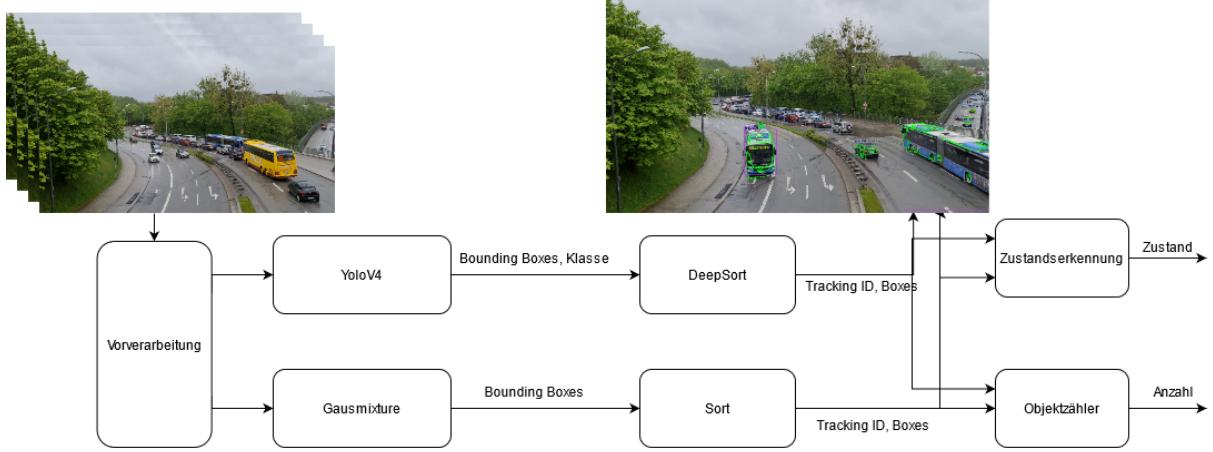


Abbildung 4. Konzept für das Projekt

wurde für die Objekterkennung ein Neuronales Netz und für das Tracking wurde Deepsort verwendet.

Mit einem Objektzähler können am Ende der beiden Pipelines die Objekte gezählt oder mit einer Zustandserkennung die Bewegungen und Stillstand erfasst werden. Die Abbildung 4 visualisiert das Konzept.

Bilder werden mittels der OpenCV Bibliotheksmethoden Frame für Frame aus einem Video ausgelesen. Des weiteren ist es möglich Bilder vor der weiteren Verarbeitung noch beliebig zuzuschneiden um nur relevant Bereiche zu betrachten. Dies hat den Vorteil das die Performance steigt und auch die Genauigkeit mancher Algorithmen zunimmt. Diese Bilder werden dann in die Pipeline gelegt. Es muss die Objektzählerbox definiert werden.

### III. OBJEKTERKENNUNG: GAUSS MIXTURE

#### A. Rauschminderung

#### IV. OBJEKTERKENNUNG: YOLOV4

YOLO ist ein Neuronales Netz für die Echtzeit-Objekterkennung. In diesem Projekt verwendeten wir die 4te und damit aktuellste Version von Yolo [4]. So bietet jede Version inkrementelle Verbesserung zum jeweiligen Vorgänger. In dem folgenden Abschnitt werden die wichtigsten Punkte aller Versionen erläutert.

YOLO ist die Abkürzung für 'You Only Look Once' was übersetzt 'Man sieht nur einmal hin' heißt. Die Aufgabe der Objekterkennung besteht darin, den Ort und die Größe der Bounding Box im Bild zu bestimmen, sowie die Objekte zu klassifizieren. Frühere Methoden, wie R-CNN und seine Variationen, verwendeten eine Pipeline, um diese Aufgabe in mehreren Schritten durchzuführen. Dies ist in der Ausführung langsam und aufwendiger zu trainieren, da jede einzelne Komponente separat trainiert werden muss. YOLO, erledigt beide Aufgaben mit einem einzigen Convolutional Neural Network. Es betrachtet dabei die Objekterkennung als ein Regressionsproblem auf räumlich getrennte Bounding Boxes

und zugehörige Klassenwahrscheinlichkeiten.

Die verwendete Loss-Funktion betrachtet drei Komponenten:

- 1) **Localization loss:** Differenz zwischen vorhergesagten Bounding-Box-Werten ( $x, y, w$  und  $h$ ) und tatsächlichen Bounding-Box-Werten.

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$

$$+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

Wenn  $\mathbb{1}_{ij}^{obj} = 1$  dann ist die Boundingbox  $j$  in Zelle  $i$  verantwortlich das Objekt zu erkennen.  $\lambda_{coord}$  ist eine Gewichtung um den Localization loss wichtiger zu machen. Standardmäßig ist  $\lambda_{coord} = 5$  [7].

- 2) **Confidence loss:** Fehler im Konfidenzwert.

$$\sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2$$

$\hat{C}_i$  ist der Konfidenzwert der Boundingbox  $j$  in Zelle  $i$ . Wenn  $\mathbb{1}_{ij}^{obj} = 1$  dann ist die Boundingbox  $j$  in Zelle  $i$  verantwortlich das Objekt zu erkennen.

Wenn das Objekt nicht erkannt wurde ist der Confidence loss:

$$\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2$$

$\lambda_{noobj}$  ist eine Gewichtung um den Confidence loss unwichtiger zu machen. Standardmäßig ist  $\lambda_{noobj} = 0.5$  [7].

- 3) **Classification loss:** Differenz zwischen den vorhergesagten Klassenwahrscheinlichkeiten und den tatsächlichen Klassenwahrscheinlichkeiten:

$$\sum_{i=0}^{S^2} \mathbb{1}_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

Wenn  $\mathbb{1}_i^{obj} = 1$  dann wurde ein Objekt in Zelle  $i$  gefunden.  $\hat{p}_i(c)$  beschreibt die Klassenwahrscheinlichkeit für die Klasse  $c$  in der Zelle  $i$  [7].

Zusammengesetzt sieht der Loss wie folgt aus.

$$\begin{aligned} & \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\ & + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 \\ & + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\ & + \sum_{i=0}^{S^2} \sum_{c \in classes} \mathbb{1}_i^{obj} (p_i(c) - \hat{p}_i(c))^2 \end{aligned}$$

Das Bild ist in ein  $S \times S$ -Gitter mit den Residualblöcken aufgeteilt. Wenn der Mittelpunkt eines Objekts in eine Gitterzelle fällt, ist diese Gitterzelle für die Erkennung dieses Objekts zuständig. Jede Gitterzelle sagt  $B$  Bounding Boxes und  $C$  Klassenkonfidenzwerte für diese Boxen voraus [3]

Diese Klassenkonfidenzwerte zeigen, wie sicher das Modell ist, dass die Boundingbox ein Objekt enthält und auch wie genau die Box das Objekt beschreibt. Die Konfidenz wird wie folgt definiert:

$$c = Pr(\text{Objekt}) * IoU_{pred}$$

Die IoU ist zwischen zwei Boxen A und B ist wie folgt definiert:

$$IoU = \frac{A \cap B}{A \cup B}$$

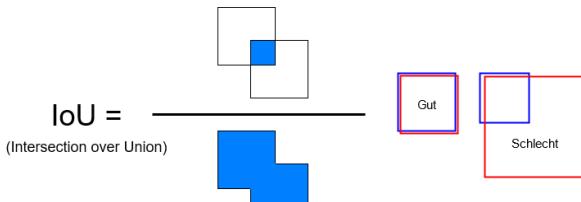


Abbildung 5. Intersection-Over-Union

In Abbildung 5 ist die Berechnung der Intersection-Over-Union dargestellt. Sie beschreibt das Verhältnis der Schnittfläche einer Bounding Box mit der Grundwahrheit bezüglich der Gesamtfläche von Bounding Box und Grundwahrheit. Je genauer die beiden Flächen übereinander liegen desto höher der Wert, desto besser ist die Vorhersage der Bounding Box. ++QUELLE++

In der zweiten Version von Yolo wurde an mehreren Stellen etwas verbessert. So wurde zum Beispiel Batch-Normalization in Convolutioonal-Layers eingefügt.

Die Inputgröße des Netzes wurde von  $224 * 224$  auf  $448 * 448$  erhöht.

Eine weitere wichtige Änderung sind Ankerboxen. Da Objekte meist immer ähnliche Formen haben kann man eine gewisse Menge an sogenannten Ankerboxen definieren welche als Basis Boundingboxen fungieren. So wird anstatt die absolute Größen von Boxen in Bezug auf das gesamte Bild vorherzusagen, eine Ankerbox ausgewählt die am besten zu dem gewünschten Objekten passende verwendet. Die Abbildung 6 zeigt dieses Konzept. Diese Ankerboxen

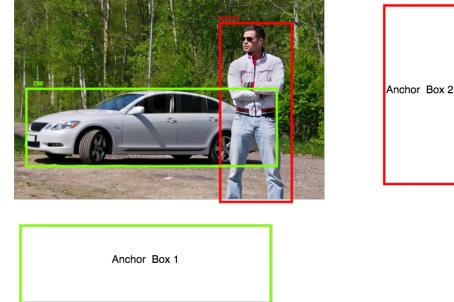


Abbildung 6. Passende Ankerboxen werden ausgewählt [2]

kann man mittels Clusteralgorithmen wie K-Means und den Boundingboxen aus dem Datensatz berechnen. Dies hat vor allem den Vorteil das die Boundingboxen besser zu den Objekten des jeweiligen Datensatzes besser passen.

Jede Vorhersagen für eine Boundingbox enthält die 4 Koordinaten:  $t_x, t_y, t_w, t_h$ .  $t_x$  und  $t_y$  sind die x an y Koordinaten, relative Zellen Mittelpunkt.  $t_w$  und  $t_h$  sind die Skalierungsfaktoren relativ zu Ankerboxdimensionen.  $p_w$  und  $p_h$  sind die Breite und Höhe der Ankerbox. Dabei gibt es noch einen Gitterzellenoffset  $c_x$  und  $c_y$ .

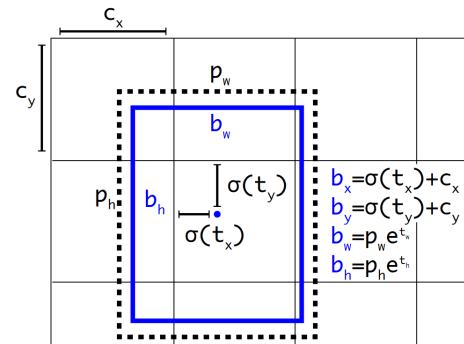


Abbildung 7. Zusammensetzung einer Boundingbox [5]

Außerdem wir der *objectness score*  $t_o$  vorhergesagt.

$$P_r(\text{Object}) * IoU_b^{object} = \sigma(t_o)$$

$\sigma(t_o)$  entspricht dem Box Konfidenz Score. Der Output des Modells enthält für jede Gitterzelle n Boxen, wobei n der

Anzahl der Ankerboxen entspricht. Jede Box ist ein Vektor aus 4 Koordinaten objectness score und der Klassen C. Der Vektor enthält:  $[b_x, b_y, b_w, b_h, b_o, C]$  [5].

$$C = [c_1, c_2, \dots, c_{80}]^T$$

$$P = [P_r * c_1, P_r * c_2, \dots, P_r * c_{80}]$$

$$\text{class} = \text{argmax}_i(C)$$

$$\text{score} = \max(P)$$

In der dritten Version wurde vor allem die Genauigkeit weiter verbessert. So wurden Restblöcke, Skip-Verbindungen und Upsampling Techiken hinzugefügt. Das Netzwerk wuchs damit drastisch auf insgesamt 106 Convolutional-Layer. Der Featureextraktor Darknet19 aus YoloV2 wurde durch Darknet-53 ersetzt. Darknet-53 besitzt 53 Convolutional-Layer und beinhaltet zusätzlich vereinzelte Abkürzungen im Netz. Eine weitere Änderung betrifft den Softmax layer für die Bestimmung der Zielklasse. Früher wurde immer nur die Klasse ausgewählt die den höchsten Score erzielte. Dies hat sich geändert. Dadurch das die Klassenvorhersage mit Hilfe einer Logistische Regression stattfindet kann ein Objekt mehrere Label gleichzeitig haben wie zum Beispiel Person und Frau. Diese Multilabelklassifikation wird für unser Projekt aber nicht benötigt.

Ein weitere Neuerung ist das Vorhersagen über drei verschiedene Skalierungen. Das System extrahiert Merkmale mit einem ähnlichen Konzept wie bei Merkmalspyramiden (FPN). Diese Technik verbessert die Genauigkeit vor allem bei kleineren Objekten. So werden Feature-Map aus einer früheren Schicht aus dem Netzwerk extrahiert und mit höher gesampelten Features durch Verkettung zusammen gefügt. Mit weiteren Faltungsschichten werden diese kombinierte Feature-Map verarbeitet. So profitiert man von Vorhersagen für alle der drei Skalierungen und damit von allen vorherigen Berechnungen sowie von den feinkörnigen Merkmalen aus der frühen Phase des Netzwerks. Die Abbildung 8 zeigt das Konzept eines Merkmalspyramiden netzwerks.

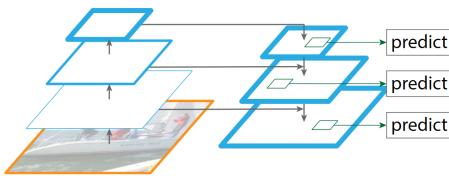


Abbildung 8. Darstellung eines Merkmalspyramiden netzwerks [8]

Die letzte dieser Schichten der jeweiligen Skalierungen sagt einen 3-D-Tensor voraus, der Bounding Box, Objekthaftigkeit und Klassenvorhersagen enthält. Es werden aber für jede der drei Skalierung drei Boxen vorausgesagt. Der Tensor entspreche dann für 4 Bounding-Box-Offsets, 1 Objektvorhersage und 80 Klassenvorhersagen

$$N * N * [3 * (4 + 1 + 80)]. [6]$$

YoloV4 hat die Geschwindigkeit und Genauigkeit verbessert. So nennt das Paper [4] zum Beispiel *Bag of Freebies*. Damit sind Verbesserungen gemeint die die Genauigkeit des Modells zur Trainingszeit verbessern aber die Inferenzzeit nicht beeinflussen. So wurde stark das Training optimiert. Bei dem Training werden Bilder zusätzlich aus dem Datensatz generiert. Existierende Bilder werden dupliziert und dann rotiert, zugeschnitten, verzerrt, verdunkelt, etc. und dann auch durch das Netz geschickt. Auch Techniken wie *Cut Mix* und *Mosaic Dataaugmentation* werden angewendet. Bei älteren Verfahren werden Bildteile gelöscht. Bei den beiden verwendeten Verfahren werden stattdessen Bildteile aus den Trainingsdaten ausgeschnitten und kombiniert, wobei die Ground-Truth-Labels ebenfalls angepasst werden müssen. Diese Data Augmentation sorgt für eine verbesserte Generalisierungsfähigkeit des Modells.

Regularisierungsmethoden wie *DropOut*, *DropConnect* und *DropBlock* wurden gegen Overfitting eingesetzt.

Auch wird die Lossfunktion angepasst. Bei den bisherigen Versionen wurde die Mittlere quadratische Abweichung (MSE) für das Regressionsproblem verwendet. Diese Regression wird dann unabhängig für alle Vorhersagen von  $t_x, t_y, t_w, t_h$  verwendet. Sinnvoller ist es die Koordinaten zusammen als die IoU der Boundingboxen in Betracht zu ziehen. Deshalb verwendet YoloV4 *CIoU* als Lossfunktion um die Integrität der Objekte mit einzubeziehen. Weitere Verbesserungen die auf Kosten der Laufzeit gehen werden *Bag of Specials* genannt. Dazu gehören zum Beispiel *Squeeze-and-Excitation*, *Spatial Attention Module*, Vergrößerung des Rezeptionsfeldes des Modells und Stärkung der Fähigkeit zur Merkmalsintegration.

Im Backbone wird eine *MISH*-Aktivierungsfunktion verwendet. Diese ist eine glatte und nicht-monotone neuronale Aktivierungsfunktion, die wie folgt definiert werden kann:

$$f(x) = x * \tanh(\ln(1 + e^x))$$

Sie arbeitet besser als zum Beispiel *ReLU* oder *Swish* [9]. Die Komponenten in der Architektur wurde auch ersetzt. Im Backbone läuft nun *CSPDarknet53*. Dies basiert auf dem Darknet53 aus YoloV3 und wird mit Cross-Stage-Partial-Networks erweitert. Im Hals des Netzes wird *Spatial pyramid pooling* (SPP), *Path Aggregation Network* (PANet) und *Spatial Attention Module* (SAM) statt den Merkmalspyramiden (FPN) aus YoloV3 verwendet.

SPP wendet verschiedene Strategien bei unterschiedlichen Skalierungen an. Sie ersetzt den letzten Pooling-Layer durch einen räumlichen Pyramiden-Pooling-Layer. Die Feature-Maps werden räumlich in  $m \times m$  Bins unterteilt. In Yolo wird eine modifizierte Version von SPP verwendet die die räumliche Dimension der Ausgabe beibehalten. In FPN werden die Objekte separat und unabhängig voneinander auf verschiedenen Skalenebenen erkannt. Dies kann zu doppelten Vorhersagen führen. PAN kombiniert deshalb die Informationen aus allen Ebenen.

SAM verbessert die räumliche Aufmerksamkeit. Am *Head* der für Prädiktion verantwortlich ist wurde nichts geändert und ist damit identisch zu YoloV3 [4] [10]. Diese Architektur mit der viel zahl an Komponenten wird in er Abbildung 9 dargestellt.

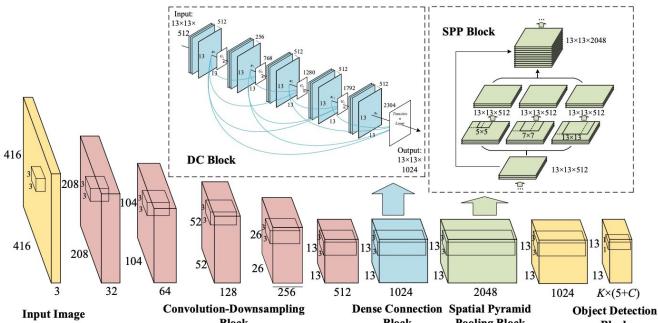


Abbildung 9. Das Konzept von YoloV4 [10]

Wir mussten unser Modell nicht mehr trainieren da es bereits mit dem Microsoft COCO Datensatz trainiert wurde. Es sind 80 verschiedene Klassen erkennbar. Wir interessieren uns aber nur für einen kleineren Teil wie:

- 1) Personen
- 2) Pkw
- 3) Lkw
- 4) Busse
- 5) Fahrräder
- 6) Züge

Das Modell könnte noch besser arbeiten wenn es nur auf die Klassen trainiert würde die für unser Projekt relevant sind. Dies hätte aber den Rahmen des Projektes gesprengt. Das Modell ist mit ein paar Ausnahmen für das Projekt ausreichend gut genug.

Dennoch kann man je nach Szenario per Parameter die relevanten Klassen auswählen. Das Modell prädikt trotzdem noch alle Klassen. Die nicht relevanten werden einfach nur im Postprocessing herausgefiltert.

Ein weitere Postprocesingschritt ist es die Detektion die eine zu geringe Sicherheit besitzen zu entfernen.

Die übrigen Boxen durchlaufen eine Non-Maxima-Suppression. Der Output kann überschneidende Boundingboxen besitzen die eigentlich zur selben Klasse gehören. Diese Duplikate können mittels der Non-Maxima-Suppression entfernt werden. Hierfür wird die IOU der Boxen berechnet. Sollte die IOU einen Threshold überschreiten dann wird das Duplikat entfernt. Die Grafik 10 zeigt wie zu viele überlappende Boxen mit einer NMS mit einer ersetzt werden kann.

Das Ergebnis einer Prädiktion ist eine Liste von Objekten welches aus den x,y Koordinate des Mittelpunktes der Boundingbox, der Breite w, der Höhe h, der Klasse und des Konfidenzwerts besteht.

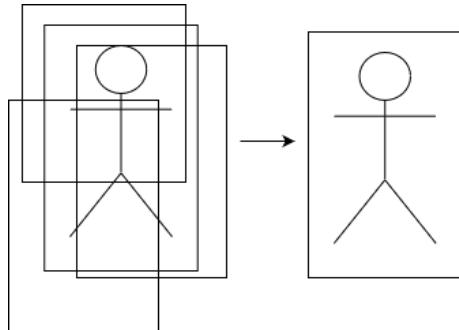


Abbildung 10. Non-Maxima-Suppression

#### A. Fehler in der Erkennung

ZUG FEHLT++++++

YoloV4 ist nicht perfekt. So gibt es gelegentlich Fehler:  
**Falschklassifizierung:**

Selten Klassifiziert das Netz falsch. So wie in der Abbildung 11 zu sehen ist wurde ein Bus als Pkw klassifiziert. Das Modell ist sich in diesem Beispiel auch nur unsicher mit 31%. Der Fehler lässt sich vermutlich erklären da die Frontpartie des Bus einem Pkw sehr ähnelt und der Rest des Fahrzeugs wegen einem Baum und einer Laterne verdeckt wird. Auch ist zu vermuten, dass das Netz falsche Merkmale gelernt hat. Dies könnte mit Sicherheit verbessert werden wenn das Modell nur auf die für uns relevanten Klassen trainiert worden wäre. Die folgenden Abbildungen 11, 12, 13 und 14 sind Beispiele für falsche Klassifikationen. Speziell ist vor allem der Fehler in Bild 14. Hier wurde ein Anhänger als Fahrzeug klassifiziert. Der Ursprung dieses Fehlers kann unterschiedliche Gründe haben. Dies liegt wahrscheinlich auch am Datensatz selbst.



Abbildung 11. Falschklassifizierung eines Bus wegen Verdeckung

#### Fehlende Detektion:

Ein etwas häufiger Fehler ist das Detektionen komplett fehlen. Dies kann mehrere Gründe haben. So zeigt die Abbildung 15 zwei Typische Gründe. Hauptgrund ist es das Objekte teilweise verdeckt sind. In Kombination wenn Objekte sehr klein sind ist es nahezu nicht mehr möglich

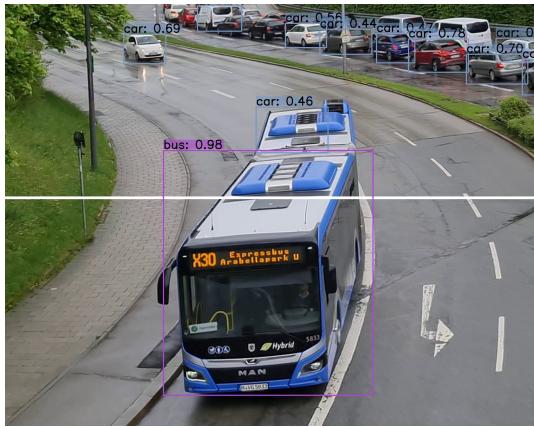


Abbildung 12. Falschklassifizierung des Dachs als Pkw



Abbildung 15. Fehlende Klassifikationen



Abbildung 13. Falschklassifizierung eines Hausdachs



Abbildung 16. Fehlende Klassifikationen

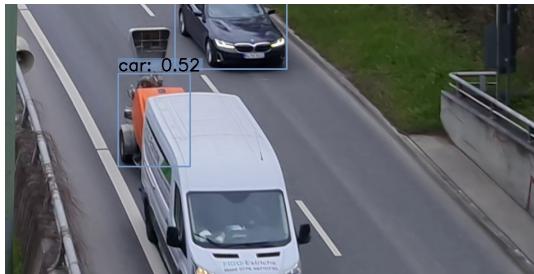


Abbildung 14. Falschklassifizierung eines Anhängers

zuverlässig Detektion zu bekommen. Auch wenn Objekte das Bild verlassen oder gerade betreten kann es noch kurz zu fehlenden Detektionen führen.

#### Unpassende Boundingboxen:

Yolo hat durch seine Eigenschaft nur rechteckige Boundingboxen vorhersagen zu können eine große Schwäche. Das resultiert in manchen Szenarien zu sehr unpassenden Boundingboxen. Beispielsweise kann man in der Abbildung 17 gut erkennen das wegen dem Busses welcher Diagonal steht die Boxen zu groß werden. Der Effekt tritt noch verstärkter auf bei noch längere Objekte wie die Zügen.++++++ZUG

BILD++. Auch kann die Box für das Objekt etwas falsch anliegen wenn Bäume oder Laternen ein Teil verdecken wie in Abbildung 18 zu sehen ist. Diese unpassenden Boxen erschweren das Tracking ungemein. Dazu später aber noch mehr.

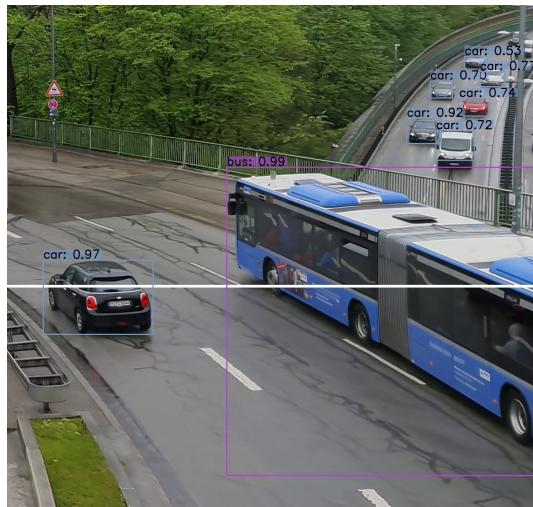


Abbildung 17. Unpassende Boundingboxen

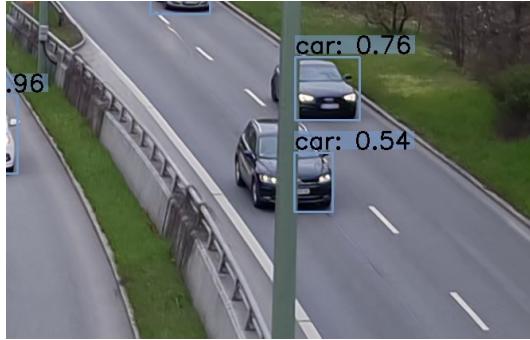


Abbildung 18. Unpassende Boundingboxen

## V. TRACKING: SORT

SORT berechnet auf Basis des Kalman Filters und der Hungarian Methode die Bewegung des Objekts und weist je nach Status eine eindeutige Identifikationsnummer zu. Die Hungarian Methode, auch Kuhn-Munkres-Algorithmus genannt, wird für das Lösen von Zuordnungsproblemen in einem gewichteten kompletten bipartiten Graph genutzt. Hierbei wird die Zuordnung anhand der minimalen Gewichtung, zwischen den Elementen gelöst. Gegeben ist ein Graph  $G = (A \cup B, E)$ . Zwischen  $A$  und  $B$  wird nun die Verbindung mit der kleinsten Gewichtung gesucht. Hierfür wird eine Kostenmatrix  $C$  erstellt mit  $c_{ij}$  als Kosten um  $a_i \in A$ ,  $b_j \in B$  zuzuordnen.

Der Kalman-Filter funktioniert für die Problemstellung sehr gut, da dieser ein lineares System mit gaußischem Rauschen voraussetzt. Dies ist gegeben durch die gleichmäßige Bewegung der Züge/ Autos in eine Richtung. Der Kalmanfilter kann aus verrauschten, teils redundanten Messungen die Zustände und Parameter des aktuellen Systems schätzen. Die Zustände des aktuellen Systems sind die Positionen der Objekte angegeben in Bounding Boxen. Durch Berücksichtigung des letzten Zustand des Systems, (die letzte Position), der Schätzung der nächsten Position und Abgleich mit der aktuellen Messung (die aktuelle Detektion) und anschließender Aktualisierung des Systems, kann eine zuverlässige Verfolgung des Objekts berechnet werden. Probleme die den Kalmanfilter stören sind zum Beispiel ungleichmäßige Bewegungen von Fahrzeugen, dies wird „Process Noise“ genannt, sowie Ungenauigkeiten in Messungen, diese werden als „Measurement Noise“ bezeichnet. Kann zu dem aktuell getrackten Objekt in dem Frame keine Bounding Box zugeordnet werden, wird der Zustand des Systems auch ohne die Korrektur durch die Messung berechnet. Kann in einem späteren Frame wieder eine Bounding Box zugeordnet werden, wird der Zustand durch diese Detektion erneut aktualisiert. Die Hungarian Methode wird verwendet, um die Detektion einem existierenden getrackten Objekt zuzuordnen. Hierfür wird die Intersection-over-Union Distance zwischen jeder Detektion und allen bereits erkannten Objekten verglichen und als Gewichtung verwendet.

Die Zuweisung der ID erfolgt anschließend über die optimale Lösung der Hungarian Methode.

## VI. TRACKING: DEEPSORT

SORT an sich ist bereits ein Tracker, das Problem ist jedoch, dass die ID-Änderungen zu oft auftreten. Das heißt die Objekt-ID wechselt während der Verfolgung. Ausgelöst wird dies zum Beispiel durch die teilweise Verdeckung des Objekts. Deshalb wurde SORT um ein CNN erweitert, welches sowohl einerseits die Mahalanobis-Distanzmetrik für die Assoziation nutzt, als auch die Form des Objekts in Form eines durch das hinzugefügten CNN berechneten Feature-Vektor verfolgt. Durch Verwendung einer geeigneteren Distanzmetrik und Beschreibung des Objekts kann dieses nun besser verfolgt werden. Das CNN erstellt einen Vektor, der enthält alle Features eines gegebenen Bilds. Ursprünglich wird das Neuronale Netz als Klassifikator trainiert. Indem die Klassifikations Layer weggelassen werden, können die resultierenden Feature-Vektoren als Wiedererkennungsform verwendet werden.

Anstatt bei der Ungarischen Methode wie bei SORT die IoU-Distanz zu verwenden, wird nun die (quadrierte) Mahalanobis Distanz verwendet, welche die Distanz zwischen der detektierten Bounding Box und der gaußverteilten Vorsage des Zustands des Kalman-Filters berechnet:

$$d^{(1)}(i, j) = (d_j - y_i)^T S_i^{-1} (d_j - y_i) \quad (1)$$

Hierbei wird die Verteilung, zuvor acht-dimensional des i-ten verfolgen Objekts in den vier-dimensionalen Raum der Messungen projiziert ( $y_i, S_i$ ). Diese Bounding Box der Detektion wird mit  $d_j$  bezeichnet.

|   |  |
|---|--|
| Kalman-Filter Zustand:                      | $(u, v, \gamma, h, \dot{x}, \dot{y}, \dot{\gamma}, \dot{h})$ |
| Detektion:                                  | $(u, v, \gamma, h)$  |
| $u, v :$                                    | Mittelpunkt der Bounding Box                                 |
| $h :$                                       | Skalierungsfaktor  |
| $\dot{x}, \dot{y}, \dot{\gamma}, \dot{h} :$ | Geschwindigkeiten in Bildkoordinaten                         |

Die Mahalanobis Distanz berechnet den Abstand von einem Punkt zu einer Verteilung. Der Abstand ist dabei wie viele Standardabweichungen der Verteilung dieser Punkt vom Mittelwert der Verteilung entfernt ist. Er wird somit durch die Standardabweichung der Verteilung normiert. Die Detektion stellt hierbei den Punkt da und die Unsicherheit der Zustandsvorhersage, beziehungsweise die Unsicherheit der Position des getrackten Objekts, die Verteilung. Dadurch wird bei der Berechnung der Distanz die Unsicherheit der Position miteinbezogen.

Für die Distanzmetrik ergibt sich dann folgende Gleichung  $D = \lambda * D_k + (1 - \lambda) * D_a$ . Hierbei ist  $D_k$  die Mahalanobis Distanz und  $D_a$  die kleinste Kosinus-Distanz zwischen den Feature-Vektoren.  $\lambda$  dient als ein Gewichtungsfaktor.

Kann DeepSORT erfolgreich ein bereits verfolgtes Objekt zu einer Detektion assoziieren, wird anschließend eine TrackingID erstellt. Hierbei muss beachtet werden, dass im Zuge der Verbesserung von SORT weitere Sicherheiten eingebaut wurden. Ein neu getracktes Objekt erhält nicht sofort eine

endgültige Identifikationsnummer. Erst nachdem das Objekt in drei hintereinander liegenden Frames erfolgreich getrackt werden konnte erhält es eine Nummer. Das Objekt erhält des weiteren einen Zähler  $a$ , der die Anzahl an Frames zählt, seitdem das Objekt nicht mehr erfolgreich assoziiert werden konnte. Ist eine maximale Anzahl  $Age_{max}$  erreicht gilt das Objekt als verloren oder außer Reichweite der Kamera und wird aus der Objektliste entfernt. Dieser Zähler wird jedoch zurückgesetzt, sollte zuvor das Objekt erneut erfolgreich assoziiert werden.

## VII. ZÄHLEN VON OBJEKten

Wir testeten verschiedene Verfahren zu Zählen der gefundenen Objekte die das Video passieren.

**Zählen der vergebenen TrackingID's.** Unser erster Plan war das Zählen aller vergebenen Tracking ID's. Dies hätte den Vorteil das sobald ein Objekt gefunden wird auch gezählt wird. Wir stellten aber schnell fest das Aufgrund von Fehlern in der Pipeline Objekten mehrfach neue IDs zugewiesen wurde. Diese Probleme entstehen in Kombination aus Fehlern in der Objekterkennung und dem Tracker. Je zuverlässiger das Tracking funktioniert desto geringer ist der Fehler im Zählen. Da aber ein perfekter Tracker nicht immer realistisch ist müssen bessere Lösungen gefunden werden.

**Zähllinie.** Um Trackingfehler zu umgehen bauten wir eine Zähllinie ein die sobald der Mittelpunkt der Boundingbox diese schneidet einmal hochzählt. Dies hatte noch immer Probleme wie zum Beispiel das manche Boxen doppelt gezählt wurden ohne manche die Linie direkt übersprungen haben.  
WAS WAR DA NOCHMAL FALSCH Thomas?

**Zählung der Tracking IDs in einer Zählbox.** Um die beiden Techniken zu vereinen wurde eine Zählbox entwickelt. So kann für jedes Szenario per Parameter die Position und Größe der Zählbox definiert werden. Sobald der Mittelpunkt einer Boundingbox in den die Zählbereich eindringt wird überprüft ob diese Tracking ID schon in einem Set enthalten ist. Ist sie es noch nicht wird hochgezählt. Es ist sinnvoll die Zählbox in Bereiche zu setzen in denen das Tracking zuverlässig funktioniert.



Abbildung 19. Darstellung einer Zählbox

## VIII. ZUSTANDSERKENNUNG ZÜGE

Da nun ein Objekt erfolgreich erkannt, verfolgt und seine Bewegungsrichtung erkannt werden kann, wird nun für Objek-

te die der Klasse Zug zugeordnet sind, die aktuelle Zugphase bestimmt. Durch YOLO werden die Bounding Boxen pro Frame bestimmt. Durch Abgleich der TrackingID kann festgestellt werden, ob es sich immer um den selben Zug handelt. Für jede der Detektionen des ausgewählten Zugs wird durch Optical Flow die Magnitude und die Richtungswinkel bestimmt. Dabei wurde die durchschnittliche Richtung über die gesamten Richtungsvektoren der Bounding Box des Zuges bestimmt. Dies wurde im Verlauf des Projekts durch die Richtungsbestimmung des Kalmanfilters ersetzt. Dieser berechnet zusammen mit dem nächsten Zustand auch die den Richtungsvektor. Fährt der Zug ein, hat dieses Objekt durch die Bewegung eine Magnitude größer 1. Der Richtungsvektor wird zur Visualisierung in das Bild als Pfeil eingefügt. Sobald der Zug erkannt wird und eine Richtung bestimmt wird, wechselt dessen Zustand von „Kein Zug“ zu „Einfahrend“. Dabei wird immer für 5 Frames geprüft ob die Richtung gleichbleibend ist. Sollte für eine Länge von 10 Frames die Magnitude unter 0 fallen ändert sich der Zustand des Zuges zu „Haltend“. Beginnt der Zug sich wieder zu bewegen und die Bewegung ist gleichbleibend in eine Richtung wird der Zustand auf „Abfahrend“ gesetzt, bis der Zug außerhalb des Bildes ist und nicht mehr erkannt wird.

## IX. VERGLEICH KLASSISCHEN METHODEN MIT DEEP-LEARNING

### Eigenschaften der klassischen Verfahren:

- Klassische Verfahren sind deutlich besser nach zu vollziehen. Dies gilt vor allem für Fehler....
- Klassische Verfahren sind deutlich schneller. Diese Verfahren sind um den Faktor X schneller als DeepLearning-verfahren.....
- Klassische Verfahren können auch gute Leistung erzielen mit genug feinschliff.....
- Die Klassische Verfahren haben schwächen bei verwackelten Aufnahmen. Gegenmaßnahmen sind ein Stativ oder auf Software basierende Verfahren.

### Eigenschaften der Deep-Learning Verfahren:

- Deep-Learning Verfahren sind deutlich schlechter nach zu vollziehen. Dies gilt vor allem für Fehler....
- Klassische Verfahren sind deutlich schneller. Diese Verfahren sind um den Faktor X schneller als DeepLearning-verfahren.....
- Die Deep-Learning Verfahren verzeichnen vor allem gute Out-Of-The-Box Leistung.
- Die Objekterkennung YoloV4 hat überhaupt kein Problem mit verwackelten Bildern da es jeden Frame unabhängig von einander betrachtet.

## LITERATUR

- [1] <https://www.zeit.de/news/2020-10/18/s-bahn-stammstrecke-in-muenchen-fuer-arbeiten-gesperrt>
- [2] <https://medium.com/@sarangzambare/object-detection-using-non-max-suppression-over-yolov2-382a90212b51>
- [3] <https://arxiv.org/pdf/1506.02640v5.pdf>
- [4] <https://arxiv.org/pdf/2004.10934.pdf>
- [5] <https://arxiv.org/pdf/1612.08242.pdf>

- [6] <https://pjreddie.com/media/files/papers/YOLOv3.pdf>
- [7] <https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088>
- [8] [https://openaccess.thecvf.com/content\\_cvpr\\_2017/papers/Lin\\_Feature\\_Pyramid\\_Networks\\_CVPR\\_2017\\_paper.pdf](https://openaccess.thecvf.com/content_cvpr_2017/papers/Lin_Feature_Pyramid_Networks_CVPR_2017_paper.pdf)
- [9] <https://arxiv.org/vc/arxiv/papers/1908/1908.08681v1.pdf>
- [10] <https://jonathan-hui.medium.com/yolov4-c9901eaa8e61>

## X. ANHANG