

Videoanalyse und Objekttracking

1st Bartolovic Eduard
Hochschule München
München, Deutschland
eduard.bartolovic0@hm.edu

2nd Thomas Willeit
Hochschule München
München, Deutschland
XXXXX@hm.edu

3rd 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 Brudermühlentunnel am Mittleren Ring. Dieser Teil ist sehr verkehrsreich und Autobahnähnlich ausgebaut. Es gibt mehrere Fahrspuren. Alle Fahrzeuge fahren in eine Richtung. Es sind ein paar Laternen und Bäume in der Aufnahme. Es sind nur Pkw und kleine Transporter welche trotzdem als Pkw klassifiziert werden enthalten.



Abbildung 1. Verkehrsaufnahmen am Brudermühlentunnel

Verkehrsaufnahmen am Canditunnel. Eine stark befahrene Straße mit einer vielzahl von Verkehrsteilnehmern. Es herrscht zwei Richtungsverkehr mit insgesamt 4 Fahrspuren. In eine Richtung stehen die Fahrzeuge in einem Stau. Durch Regen spiegeln viele Oberflächen. Die Bäume bewegen sich aufgrund von stärkeren Wind. Diese Aufnahme ist die anspruchsvollste. In diesem Video sind neben Pkw auch Busse enthalten.

Zugaufnahmen an der Donnersbergerbrücke. Eine Aufnahme der am stärksten befahren Bahnstrecken Europas [1]. Hier existiert viel Zugverkehr auf mehreren Gleisen mit unterschiedlichen Zugtypen. Beobachtet werden die vier Gleise der zwei Bahnsteige der S-Bahnhalt 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.



Abbildung 2. Verkehrsaufnahmen am Canditunnel.



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 Verfahren wurde für die Objekterkennung Gausmixture verwendet und für das Tracking Sort. Für die DeepLearning Verfahren wurde für die Objekterkennung ein Neuronales Netz und für das Tracking wurde Deepsort verwendet. Mit einem Objektzähler können am Ende der Pipelines die Objekte gezählt oder mit einer Zustandserkennung die Bewegungen und Stillstand erfasst werden. Die Abbildung 4 visualisiert das Konzept.

Bilder werden mittels OpenCV Bibliotheken Frame für Frame aus einem Video ausgelesen. Es ist 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

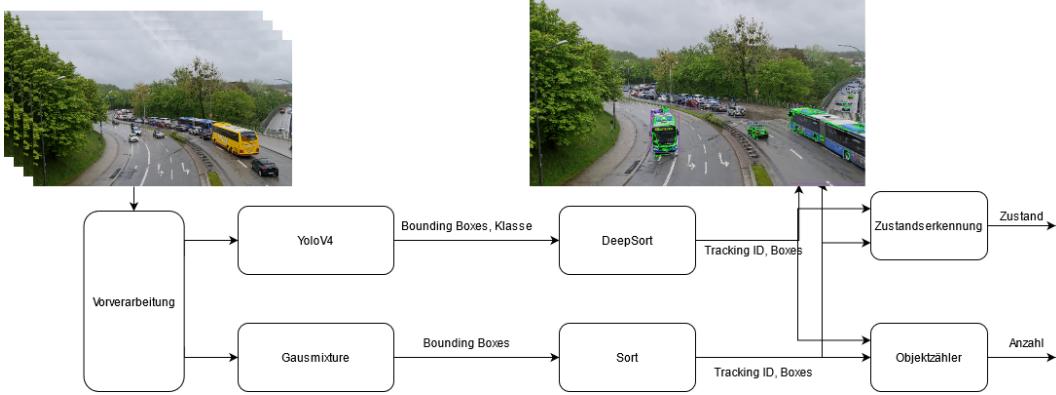


Abbildung 4. Konzept für das Projekt

Algorithmen zunimmt. Diese Bilder werden dann in die Pipeline gelegt.

III. OBJEKTERKENNUNG: GAUSS MIXTURE

A. Rauschminderung

IV. OBJEKTERKENNUNG: YOLOV4

YOLO ist ein Neuronal Netz für die Echtzeit-Objekterkennung. In diesem Projekt verwendeten wir die 4te und damit aktuell neueste Version von Yolo [4]. So bietet jede Version inkrementelle Verbesserung zum jeweiligen Vorgänger. In den 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 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 $\mathbb{1}_i^{obj} = 1$ dann ist die Boundingbox j in Zelle i verantwortlich das Objekt zu erkennen. λ_{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 $\mathbb{1}_i^{obj} = 1$ dann ist die Boundingbox j in Zelle i verantwortlich das Objekt zu erkennen.

Wenn das Objekt nicht erkannt wurde ist der Confidence loss:

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

λ_{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:

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

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.

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

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

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

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 Klassen Konfidenzwerte für diese Boxen voraus [3]. Diese Klassen Konfidenzwerte 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:

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

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

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

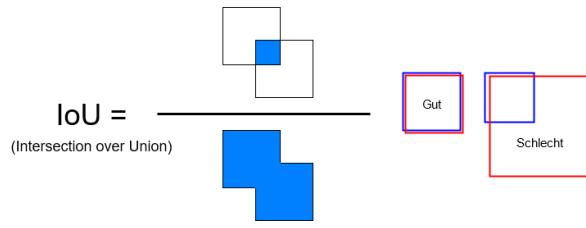


Abbildung 5. Intersection-Over-Union

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.

In der zweiten Version von Yolo wurde an mehreren Stellen etwas verbessert. So wurde zum Beispiel Batch-Normalization in Convolutional-Layers eingefügt. 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 absoluten 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 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. [5]

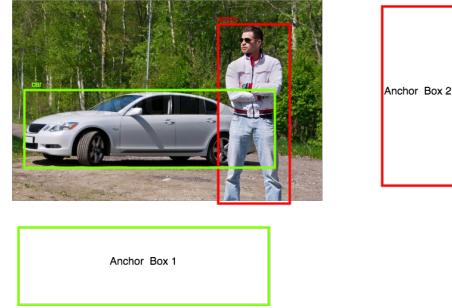


Abbildung 6. Passende Ankerboxen werden ausgewählt [2]

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 .

$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$

++++++GEHT BESTIMMT BESSER+++++ Jede

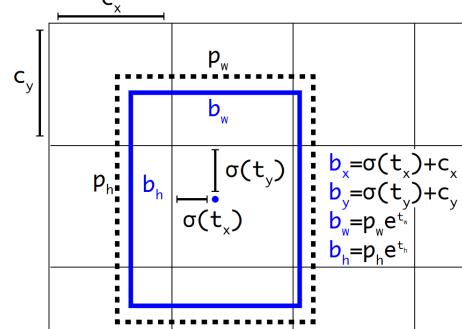


Abbildung 7. Konstruktion einer Boundingbox [5]

Gitterzelle sagt auch bedingte Klassenwahrscheinlichkeiten $Pr(Class_i|Object)$, voraus. Diese Wahrscheinlichkeiten beziehen sich auf die Gitterzelle, die ein Objekt enthält. Zum Testzeitpunkt wird die bedingten Klassenwahrscheinlichkeiten und die individuellen Box-Vertrauensvorhersagen multipliziert, wodurch wir klassenspezifische Vertrauenswerte für jede Box erhalten. Diese Werte kodieren sowohl die Wahrscheinlichkeit, dass diese Klasse in der Box vorkommt, als auch, wie gut die vorhergesagte Box zum Objekt passt.

$$Pr(Class_i|Object) * Pr(Object) * IoU_{pred}^{truth} = Pr(Class_i) * IoU_{truth}^{pred}$$

++++++

In der dritten Version wurde vor allem die Genauigkeit weiter verbessert. So wurden Restblöcke, Skip-Verbindungen und

Upsampling Techinken hinzugefügt. Das Netzwerk wuchs damit drastisch auf insgesamt 106 Convolutional-Layer. Der Featureextraktor Darknet-53 besitzt 53 Convolutional-Layer und 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 Regession 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 netzen. Hinter dem Featureextraktor liegen mehrere Faltungsschichten. Die letzte dieser Schichten sagt einen 3-D-Tensor voraus, der Bounding Box, Objekthaftigkeit und Klassenvorhersagen enthält. In dem Fall für den COCO Datensatz werden pro Skalierung 3 Boxen vorausgesagt. Der Tensor entspreche dann für 4 Bounding-Box-Offsets, 1 Objektvorhersage und 80 Klassenvorhersagen $N * N * [3 * (4 + 1 + 80)]$. In YOLOv3 erfolgt die Erkennung durch Anwendung von 1×1 Erkennungs Kernel auf Feature-Maps dreier verschiedener Skalierungen an drei verschiedenen Stellen im Netzwerk. SKALIERUNG. Diese Technik verbessert die Genauigkeit vor allem bei kleineren Objekten.

Yolov3 ist damit in der Lage in jeder Zelle 9 Boundingboxen wegen der 3 Ankerboxen für jede der 3 Skalierungstufe vorhersagen. [6]

YoloV4 hat einige Neuerungen. So nennt das Paper [4] *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 *random erase* und *Cut Out* werden angewendet. Beide Löschen die Pixel aus zufälligen Rechtecken im Bildern und ersetzen sie mit einem zufälligen Wert. Diese Data Augmentation sorgt für eine verbesserte Generalisierungsfähigkeit des Modells.

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

SMOOTHLABEL???????

Auch wird die Lossfunktion angepasst. Bei den bisherigen Versionen wurde die Mittlere quadratische Abweichung 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 IoU der Objekte in Betracht zu ziehen. Deshalb verwendet YoloV4 *CIoU* als Lossfunktion um die Integrität der Objekte mit einzubeziehen. Weitere Verbesserungen die das Paper [4], die die Laufzeit leicht erhöhen 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:

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

Die Komponenten in der Architektur wurde auch ersetzt. Im Backbone läuft nun *CSPDarknet53*. Im Neck wird *Spatial pyramid pooling (SPP)* und *Path Aggregation Network (PAN)* verwendet.++++++

Am Head wurde nichts geändert und ist damit identisch zu YoloV3 [4]. Wir mussten unser Modell nicht mehr trainieren da

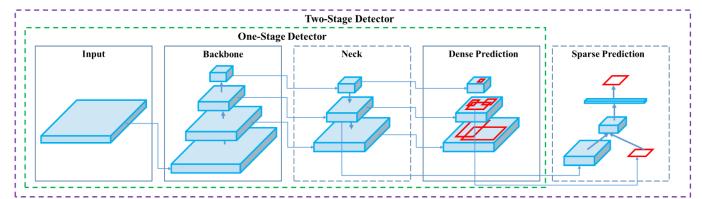


Abbildung 8. Das Konzept von YoloV4 [4]

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.

Deshalb lassen sich je nach Szenario per Parameter die relevanten Klassen auswählen. Das Modell prädikt trotzdem noch alle Klassen. Die nicht relevanten werden einfach nur 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 9 zeigt wie zu viele überlappende Boxen durch eine ersetzt werden.

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.

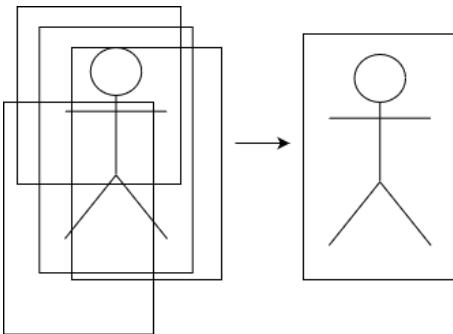


Abbildung 9. Non-Maxima-Suppression



Abbildung 12. Falschklassifizierung

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 10 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.



Abbildung 10. Falschklassifizierung

Außerdem DATENSATZ++++++

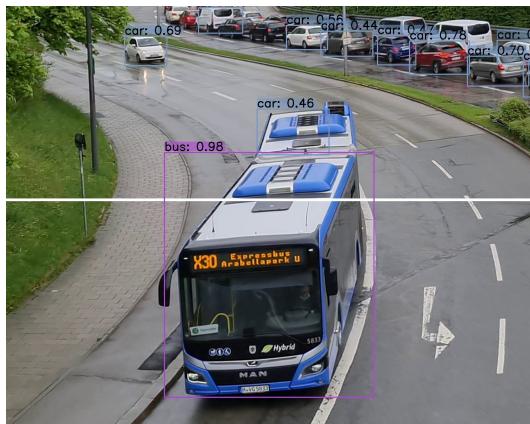


Abbildung 11. Falschklassifizierung



Abbildung 13. Falschklassifizierung

Fehlende Klassifikationen:

Ein etwas häufiger Fehler ist das Klassifikationen Fehlen. Dies kann mehrere Gründe haben. So zeigt die Abbildung ?? zwei Typische Gründe. Grund Nummer eins ist es das Objekte teilweise verdeckt sind. In Kombination wenn Objekte sehr klein sind ist es nahezu nicht mehr möglich zuverlässig Detektion zu bekommen. INPUT NETZ YOLO? Zu grobes GRID???



Abbildung 14. Fehlende Klassifikationen



Abbildung 15. Fehlende Klassifikationen

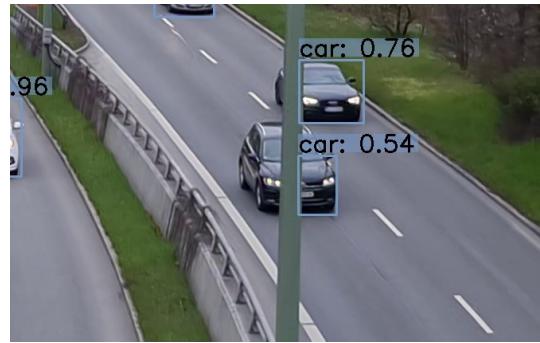


Abbildung 17. Unpassende Boundingboxen

Unpassende Boundingboxen:

Yolo hat eine große Schwäche da es nur rechteckige Boundingboxen vorhersagen kann. Das resultiert in manchen Szenarien zu sehr unpassenden Boundingboxen. Beispielsweise kann man in der Abbildung 16 gut erkennen das wegen dem Bus welcher diagonal steht die Boxen zu groß werden. Das selbe gilt für noch längere Objekte wie die Züge.++++++ZUG BILD++. Auch kann die Box für das Objekt etwas falsch anliegen wenn Bäume oder Laternen ein Teil verdecken.

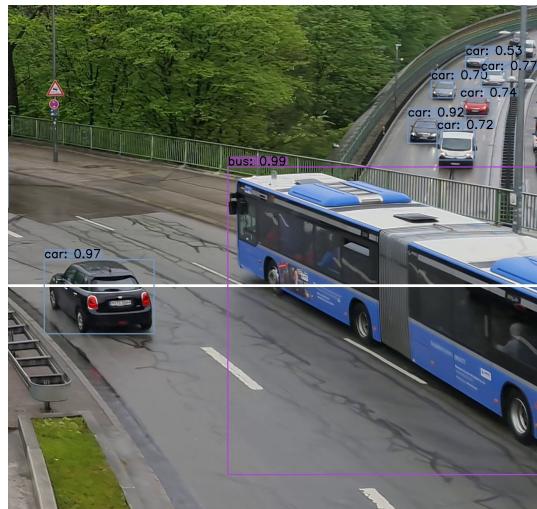


Abbildung 16. Unpassende Boundingboxen

...falsch Klassifizierung wegen schlechter Datensatz.... Zu großer Datensatz...

YOLO besitzt starke räumliche Beschränkungen für die Boundingbox-Vorhersagen, da es pro Gitterzelle nur eine begrenzte Anzahl an Vorhersagen treffen kann. Diese Anzahl ist direkt abhängig von der Anzahl der Ankerboxen [5]. Diese räumliche Beschränkung begrenzt die Anzahl der Objekte in der Nähe, die das Modell vorhersagen kann. —————

Problem zu viele Elemente nebeneinander..... wegen zu großen maschigen Grid zu wenig Ankerboxen????—————

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 gaußschem 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, γ, 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. 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. λ 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.

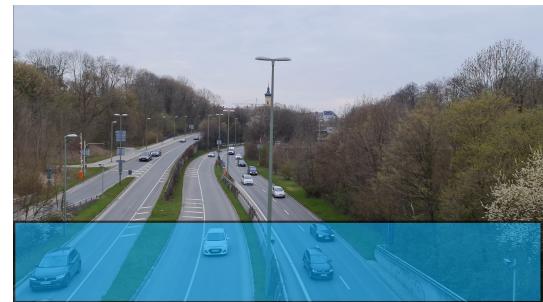


Abbildung 18. Darstellung einer Zählbox

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.

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-supression-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>

IX. VERGLEICH KLASSISCHEN METHODEN MIT DEEPLARNING

X. ANHANG