

Videoanalyse und Objekttracking

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

2rd Schäfer Julia
Hochschule München
München, Deutschland
j.schaefer0@hm.edu

3nd Willeit Thomas
Hochschule München
München, Deutschland
twilleit@hm.edu@hm.edu

Zusammenfassung—In Zeiten, in denen Neuronale Netze immer mehr Bedeutung erlangen ist es wichtig weiterhin die klassischen Algorithmen zu berücksichtigen, da diese oftmals ähnlich gute Ergebnisse erzielen. Im Rahmen der Veranstaltung Videoanalyse und Objekttracking wurden daher zwei mögliche Umsetzungen für Objekt-Detektion und zwei für Objekt-Tracking näher betrachtet und implementiert. Jeweils eine der Umsetzungen je Themenbereich arbeitet auf Basis von Neuronalen Netzen. Anschließend wurden für beide Varianten ein Objektzhäler implementiert der die erkannten Objekte zählt. Ein wichtiger Bereich sind ebenfalls Fehler und Probleme der Algorithmen, welche anhand von aufgezeichneten Daten erläutert werden. Zuletzt wird ein Vergleich von klassischen Algorithmen mit Algorithmen auf Basis von Neuronalen Netzen gegeben.

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. Es wurden drei Aufnahmen angefertigt.

Verkehrsaufnahmen am Brudermühlentunnel 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 einander auf. Dies erschwert die Detektion. Das Video wurde in 60 Bilder pro Sekunde aufgezeichnet und hat eine Bildauflösung 3840×2160 . Es wurde kein Stativ verwendet, sondern nur die interne Stabilisierung des Smartphones genutzt.



Abbildung 1. Verkehrsaufnahmen am Brudermühlentunnel

Verkehrsaufnahmen am Canditunnel: Eine stark befahrene Straße mit einer Vielzahl von Verkehrsteilnehmern. Es herrscht

Zweirichtungsverkehr 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 eine Bildauflösung von 3840×2160 . Es wurde kein Stativ verwendet sondern nur die interne Stabilisierung des Smartphones genutzt.



Abbildung 2. Verkehrsaufnahmen am Canditunnel

Zugaufnahmen an der Donnersbergerbrücke: Eine Aufnahme einer der am stärksten befahrenen Bahnstrecken Europas [5]. Hier existiert viel Zugverkehr auf mehreren Gleisen mit unterschiedlichen Zugtypen. Beobachtet werden die vier Gleise der zwei Bahnsteige des S-Bahnhofs Donnersbergerbrücke. Ein Hindernis sind die Masten, Signale, Bäume und das Bahnsteigdach welche einen guten Blick auf die Züge erschweren. In dieser Aufnahme sind Züge und Fußgänger enthalten. Der Fokus liegt aber auf den Zügen. Das Video hat 30 Bilder pro Sekunde und eine Bildauflösung 1920×1080 . Es wurde ein Stativ verwendet.



Abbildung 3. Zugaufnahmen an der Donnersbergerbrücke

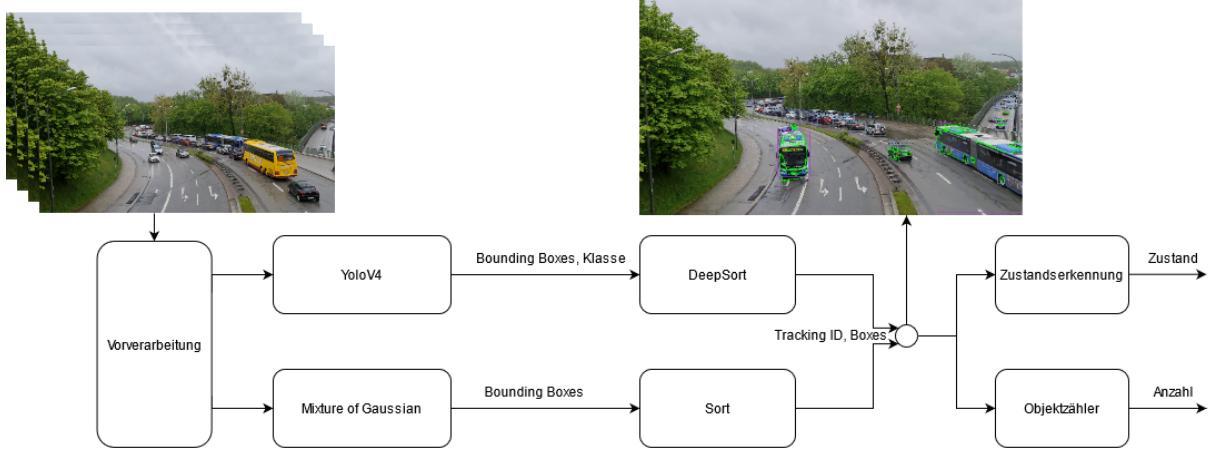


Abbildung 4. Konzept für das Projekt mit zwei Pipelines.

II. KONZEPT

Ziel des Projektes ist es klassische Verfahren mit neueren Deep Learning Methoden zu vergleichen. Dafür ist für beide Arten eine Pipeline geschaffen worden. Für die klassischen Methoden wurde für die Objekterkennung das Gausmixture Verfahren verwendet und für das Tracking der SORT Algorithmus. Für die Deep Learning Verfahren 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. Bevor mit der Erkennung von Objekten begonnen werden kann, werden die Videoaufnahmen so erstellt, angepasst und eingeschränkt, dass bessere Ergebnisse bei der Erkennung erzielt werden können.

III. PREPROCESSING

Es ist möglich Bilder vor der weiteren Verarbeitung noch beliebig zuzuschneiden, um nur relevant Bereiche zu betrachten. Dies hat den Vorteil, dass die Performance steigt und auch die Genauigkeit mancher Algorithmen zunimmt. Diese Bilder werden anschließend in die Pipeline gelegt.

Bevor das Zuschneiden beginnt, kann während und nach der Videoaufnahme, das Video stabilisiert werden.

A. Videostabilisierung

Es gibt folgende Ansätze zur Videostabilisierung:

Mechanische Videostabilisierung:

Im ersten Ansatz versucht man unerwünschte Bewegungen im Video, die durch das Wackeln der Hand entstehen, durch ein Kamerastativ oder ein Gimbal, welches mit Hilfe von Sensoren Bewegungen ausgleicht, zu vermeiden. Diese Art der Stabilisierung findet vor bzw. während der Aufnahme statt. In unserer Aufnahme der Donnersbergerbrücke wurde ein Stativ verwendet.

Optische Videostabilisierung:

Die zweite Vorgehensweise bewegt, anders als im vorherigen Absatz genannt, nicht die gesamte Kamera, sondern nur Teile im Objektiv oder beim Bildsensor. Es wird also entweder der Bildkreis über dem Bildsensor oder der Bildsensor unter dem Bildkreis verschoben. Beschleunigungssensoren erkennen hier die Bewegung der Kamera und passen dementsprechend die optischen Elemente an, damit das Bild stabilisiert wird. Das Ganze findet während der Aufnahme statt.

Digitale Videostabilisierung:

Der letzte Ansatz erfolgt nach der Aufnahme über Software und es ist keine zusätzliche Hardware nötig. Mit Hilfe von Algorithmen werden die Unterschiede zwischen zwei hintereinander folgenden Frames herausgefunden, unerwünschte Bewegungen entfernt und Anschluss zu einem stabilisierten Video zusammengefügt. Im Folgenden soll diese Art der Stabilisierung vorgestellt werden. Dabei wird nicht genauer auf die mathematischen Hintergründe eingegangen, sondern nur die Vorgehensweise beschrieben und auf die Algorithmen dahinter verwiesen [27][3].

Videostabilisierung - Implementierung:

- 1) Im ersten Schritt werden die Frames eingelesen und in ein Grauwertbild umgewandelt.
- 2) Anschließend wird die Bewegung zwischen dem aktuellen und dem vorherigen Frame bestimmt. Theoretisch würden 2 Punkte zur Bestimmung reichen, allerdings sollten für eine stabile Schätzung deutlich mehr Punkte verwendet werden. Bildregionen mit besonders vielen Ecken eignen sich hier äußerst gut, weshalb mithilfe von goodFeaturesToTrack() von OpenCV die gewünschten Features detektiert werden. Diese Funktion besteht aus einer Modifikation des Harris-Corner-Detectors durch Shi-Tomasi [24].
- 3) Danach kann mit der Lucas-Kanade Methode der Optical Flow [20] bestimmt werden. Wie in der Abbildung 6 zu sehen ist werden sowohl die Bewegungen der Fahrzeuge als auch Bewegungen in den Bäumen

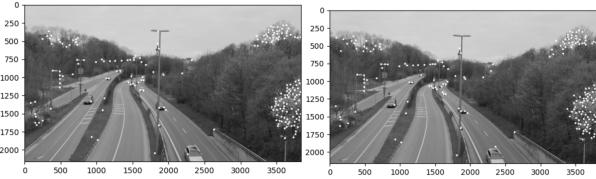


Abbildung 5. Shi-Tomasi Corner Detector: Good Features to Track

oder den Straßenlaternen im Hintergrund eingezeichnet. Diese sind geringer als die Bewegung der Fahrzeuge und entstehen durch das Wackeln der Hand beim Filmen. Sie könnten allerdings auch durch Wind verursacht werden, wodurch es zu einem Fehler bei der Stabilisierung kommen könnte.



Abbildung 6. Lucas-Kanade Optical Flow: Bewegungen im Bild

- 4) Da nun die Features im aktuellem und dem Frame davor bekannt sind, lässt sich daraus die affine Transformation zwischen den beiden Bildern mit `estimateRigidTransform()` berechnen. Daraus wird dann die Translation in x, y-Richtung und die Rotation herausgeholt und für den späteren glatteren Übergang abgespeichert.
- 5) Nach Durchlaufen aller Frames wird mit `np.cumsum` die kumulierte Summe aller gespeicherten Transformationen und somit die Trajektorie erstellt.
- 6) Nun liegen 3 Kurven vor, die die Veränderung der Bewegung (x, y, Winkel) über die Zeit darstellen. Diese Kurven werden nun mit dem Bewegten Mittelwert geglättet. Um den Bewegten Mittelwert zu verstehen, betrachten wir das folgende Beispiel: Eine Kurve ist in einem Array `c` gespeichert und die Filtergröße ist 5. Möchten wir nun das k-te Element glätten ergibt sich die folgende Berechnung:

$$f[k] = \frac{c[k-2] + c[k-1] + c[k] + c[k+1] + c[k+2]}{5}$$

- 7) Im letzten Schritt wird der Unterschied zwischen der originalen und der geglätteten Trajektorie berechnet und auf die ursprüngliche Transformation addiert, wodurch

man eine geglättete Transformation erhält. Diese wird nun auf die Frames angewandt. Um Artefakte zu vermeiden, werden die Bilder um den Bildmittelpunkt minimal skaliert, dadurch können schwarze „Flecken“ an den Bildrändern vermieden werden, solange die Kamerabewegungen nicht zu extrem sind [19].

Videostabilisierung – Ergebnis:

In der Abbildung 7 sieht man links die Detektion auf dem Originalvideo und rechts die Detektion auf dem stabilisierten Video. Während die Fahrzeugdetektion fast identisch ist, fällt auf, dass die Bewegungen in den Bäumen und der restlichen Umgebung deutlich geringer sind. Zwar konnten nicht alle unerwünschten Bewegungen entfernt werden, aber es konnte eine sichtbare Verbesserung erzielt werden.

Videostabilisierung – Alternativer Ansatz:

Die restlichen weißen Bildpunkte werden durch Anwendung einer Erosion entfernt und die übrig gebliebenen Fahrzeuge wieder mit einer Dilatation deutlich gemacht. Wie in der folgenden Bilderfolge zu sehen ist, verschwinden durch die Erosion die unerwünschten Bewegungen, die nicht durch die Videostabilisierung entfernt werden konnten. Die nun kaum sichtbaren Flächen der Fahrzeuge werden mit der Dilatation so weit vergrößert, dass diese wieder gut zu detektieren sind [7]. Die Schritte werden in den Grafiken 8 bis 11 gezeigt.

IV. OBJEKTERKENNUNG: MOG-MIXTURE OF GAUSSIAN

Ziel des Mixture of Gaussian Algorithmus [26] ist, den unbewegten Hintergrund zu erkennen und diesen vom Bild zu subtrahieren. Das heißt alle Pixel, die zum Hintergrund gehören, werden schwarz und alle anderen weiß dargestellt. Anschließend können mit `cv2.findContours()` zusammenhängende weiße Pixel zu Objekten hinzugefügt werden. Wenn die Fläche der Pixel einen definierten Threshold überschreitet, wird um diese eine Bounding Box eingezeichnet.

Funktionsweise MOG

Im ersten Schritt wird der Funktion eine Anzahl an Frames zum „Lernen“ zugeteilt. Aus diesen wird der unbewegte Hintergrund bestimmt, welcher später subtrahiert wird, sodass nur die bewegten Objekte übrigbleiben. Der Algorithmus modelliert für jeden Pixel den Hintergrund mit unterschiedlich gewichteten Gauss-Funktionen η . Diese K-Funktionen enthalten eine Gewichtung (ω_i, t), den durchschnittlichen Intensitätswert (μ_i, t) und die Standardabweichung (σ_i, t) für den i-ten Gaussian zum Zeitpunkt t .

$$\sum_{i=1}^K \omega_i, t * \eta(u; \mu_i, \sigma_i)$$

Mit jedem neuen Pixel (z.B. Kamerabewegung oder Fahrzeugbewegung) werden die Gauss-Funktionen frameweise aktualisiert. Dabei wird die Pixelhistorie und der neue Pixelwert berücksichtigt. Falls nun zu einem neuen Pixel keiner der Gauss-Funktionen innerhalb eines Thresholds passt, wird der Pixel als Vordergrund deklariert, ansonsten



Abbildung 7. Links: ohne Videostabilisierung; Rechts: mit Videostabilisierung

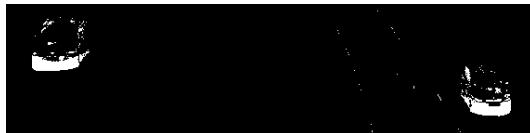


Abbildung 8. Mixture of Gaussian



Abbildung 9. Ergebnis der Erosion auf Mixture of Gaussian

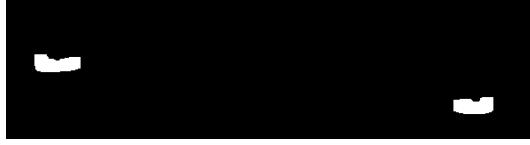


Abbildung 10. Ergebnis der Dilatation auf Erosion



Abbildung 11. Objekterkennung

als Hintergrund. Diese adaptive Hintergrundsubtraktion [1][10][13] mit MOG wird in Abbildung 12 gezeigt.

Vereinfacht beschrieben misst der Algorithmus in den Lernbildern die Zeit, in welcher ein Pixel den gleichen Intensitätswert hat und bestimmt so den Hintergrund. Ändert sich der Wert, wird an dem Pixel eine Bewegung erkannt. Dieser wird als Vordergrund bestimmt und die Parameter aktualisiert.

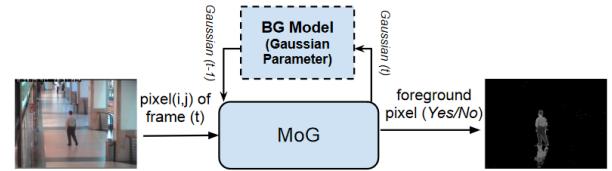


Abbildung 12. Adaptive Hintergrundsubtraktion mit MOG [26]

A. Fehler in der Erkennung

Eine Schwierigkeit bei der Objekterkennung ist, dass die Bewegungen hauptsächlich bei großen Intensitätsunterschieden wie der Wechsel von Straße zu Fahrzeuganfang und von Fahrzeugende zur Straße, erkannt werden. Große Flächen wie die Motorhaube, Windschutzscheibe und die Dachfläche werden nicht erkannt, da diese so lange einen Pixel mit einem sehr ähnlichen Intensitätswert „bedecken“, dass dieser zwischendurch als Hintergrund festgelegt wird. Dabei kann es passieren, dass je nach Thresholdgröße mehrere Objekte in einem Fahrzeug erkannt werden. Dieses Verhalten zeigt sich in Abbildung 13, 14 und 15. Ein weiteres Problem ist es, wenn neben den Objekten markante Kanten liegen. Diese Kanten sorgen zum Beispiel dafür, dass der Bus in der Grafik 14 und 15 zweigeteilt wird. Der Grund hierfür dürfte die Bordsteinkante und der Asphaltflick auf dem Radweg sein. All diese Probleme führen dazu, dass der Bus, der in den Abbildungen 14 und 16 zu sehen ist, insgesamt dreimal erkannt wird.



Abbildung 13. Schwierigkeiten bei der Objekterkennung

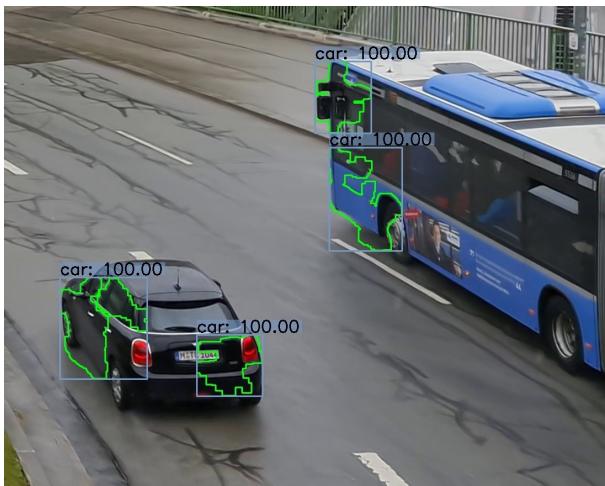


Abbildung 14. Schwierigkeiten bei der Objekterkennung



Abbildung 15. Schwierigkeiten bei der Objekterkennung

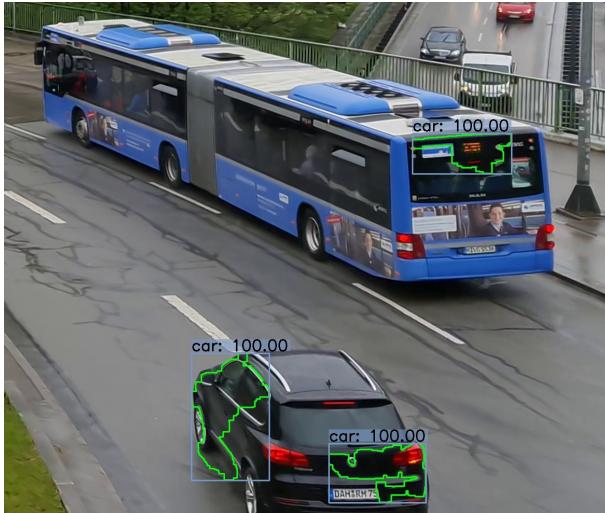


Abbildung 16. Schwierigkeiten bei der Objekterkennung

Um diese Probleme zu lösen, steigern wir den Einfluss der Dilatation. Damit werden die Flächen, die als Basis der Erkennung dienen verbunden. Dies resultiert in einer verbesserten Erkennung, wie gut in den Abbildungen 17 und 18 zu sehen ist. Das ein Bus zwei mal erkannt wird wurde damit nicht gelöst.

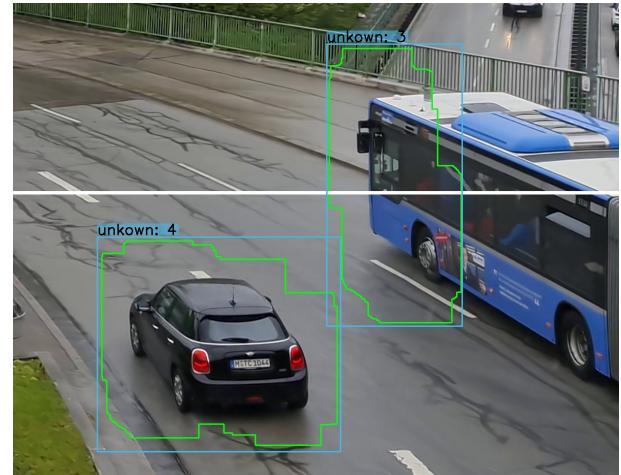


Abbildung 17. Optimierung der Parameter verbessert die Ergebnisse

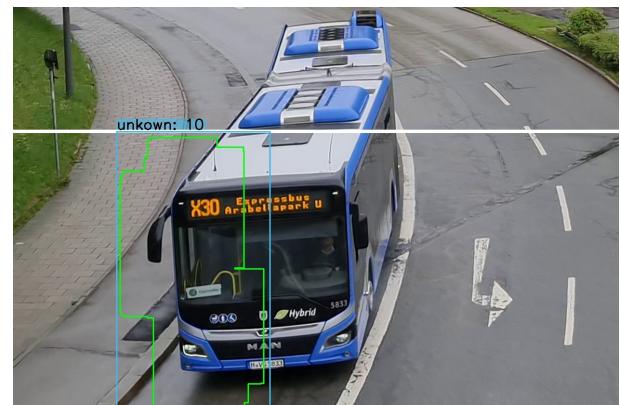


Abbildung 18. Optimierung der Parameter verbessert die Ergebnisse

V. OBJEKTERKENNUNG: YOLOV4

YOLO [23] ist ein Neuronal Netz für die Echtzeit-Objekterkennung. In diesem Projekt wurde die vierte und damit aktuellste Version von YOLO [4] verwendet. 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 [9] 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 Boxen und zugehörige Klassenwahrscheinlichkeiten.

Für die Regression 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 Bounding Box j in Zelle i verantwortlich das Objekt zu erkennen. λ_{coord} ist eine Gewichtung um den Localization loss wichtiger zu machen. Standardmäßig ist $\lambda_{coord} = 5$ [14].

- 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 Bounding Box j in Zelle i . Wenn $\mathbb{1}_i^{obj} = 1$ dann ist die Bounding Box 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 $\lambda_{noobj} = 0.5$ [14].

- 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 [14].

Zusammengesetzt sieht der Loss wie folgt aus:

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

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 Boxen und C Klassenkonfidenzwerte für diese Boxen voraus [23].

Diese Klassenkonfidenzwerte zeigen, wie sicher das Modell ist, dass die Bounding Box 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 zwischen zwei Boxen A und B ist wie folgt definiert:

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

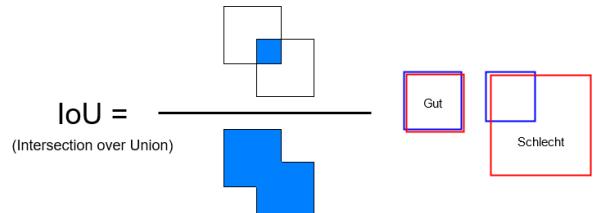


Abbildung 19. Intersection-Over-Union

In Abbildung 19 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 [21] wurde an mehreren Stellen Verbesserungen erreicht. So wurde beispielsweise Batch-Normalization in Convolutional-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 eine gewisse Menge an sogenannten Ankerboxen definiert werden, welche als Basis Bounding Boxen 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 20 zeigt dieses Konzept. Diese Ankerboxen kann

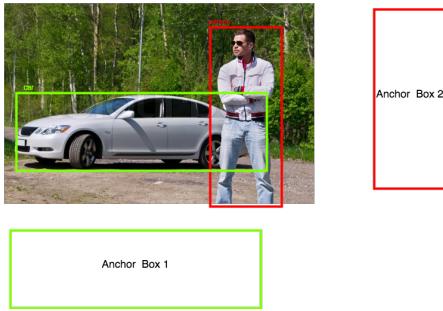


Abbildung 20. Passende Ankerboxen werden ausgewählt [34]

man mittels Clusteralgorithmen wie K-Means [30] und den Bounding Boxen aus dem Datensatz berechnen. Dies hat vor allem den Vorteil, dass die Bounding Boxen besser zu den Objekten des jeweiligen Datensatzes passen.

Jede Vorhersagen für eine Bounding Box 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 .

Außerdem wir der *objectness score* t_o vorhergesagt.

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

$\sigma(t_o)$ 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 vier Koordinaten, dem *Objectnessscore* und der Klassenwahrscheinlichkeiten C [21]. Der Vektor enthält: $[b_x, b_y, b_w, b_h, b_o, C]$.

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

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

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

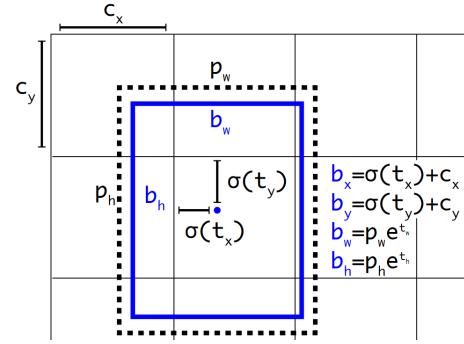


Abbildung 21. Zusammensetzung einer Bounding Box [21]

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

In der dritten Version wurde vor allem die Genauigkeit weiter verbessert. So wurden Restblöcke, Skip-Verbindungen und Upsampling Techniken 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 nur die Klasse ausgewählt, die den höchsten Score erzielte. Dies hat sich geändert. Dadurch, dass die Klassenvorhersage mit Hilfe einer Logistische Regression stattfindet, kann ein Objekt mehrere Label gleichzeitig haben wie zum Beispiel Person und Frau. Diese Multikelabelklassifikation wird für unser Projekt aber nicht benötigt.

Ein weitere Neuerung ist das Vorhersagen über drei verschiedener Skalierungen. Das System extrahiert Merkmale mit einem ähnlichen Konzept wie bei Merkmalspyramiden (FPN)[16]. 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 wird von Vorhersagen für alle der drei Skalierungen profitiert und damit von allen vorherigen Berechnungen, sowie von den feinkörnigen Merkmalen aus der frühen Phase des Netzwerks. Die Abbildung 22 zeigt das Konzept eines Merkmalspyramidenetzwerks.

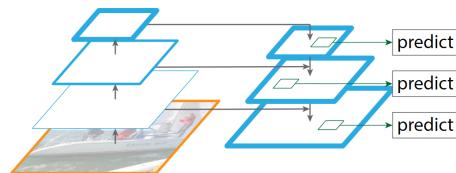


Abbildung 22. Darstellung eines Merkmalspyramidenetzwerks [16]

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 enthält dann die vier Bounding-Box-Offsets, eine Objektvorhersage und 80 Klassenvorhersagen $N * N * [3 * (4 + 1 + 80)]$ [22].

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* [33] und *Mosaic Data Augmentation* 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. Die Data Augmentation sorgt für eine verbesserte Generalisierungsfähigkeit des Modells.

Regularisierungsmethoden wie *DropOut* [25] und *DropBlock* [8] wurden gegen Overfitting eingesetzt.

Auch wird die Lossfunktion aus YOLOv1 angepasst. Bei den bisherigen Versionen wurde die mittlere quadratische Abweichung (MSE) für das Regressionsproblem verwendet. Die 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 Bounding Boxen in Betracht zu ziehen. Deshalb verwendet YOLOv4 *CIoU* [35] 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* [32], Vergrößerung des Rezeptionsfeldes des Modells und Stärkung der Fähigkeit zur Merkmalsintegration.

Im Backbone wird eine *MISH*-Aktivierungsfunktion [18] 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* [18]. 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 (CSP)[28] erweitert. Im Hals des Netzes wird *Spatial pyramid pooling* (SPP)[11] verwendet. *Path Aggregation Network* (PANet)[17] und *Spatial Attention Module* (SAM)[32] ersetzen die Merkmalspyramiden (FPN) die YOLOv3 verwendet.

Convolutional-Layer brauchen keine feste Bildgröße und erzeugen deshalb Feature-Maps beliebiger Größe. *Fullyconnected-Layer* benötigen stattdessen per Definition eine Eingabe mit fester Länge. SPP entfernt genau diese

Restriktion. SPP wendet verschiedene Strategien bei unterschiedlichen Skalierungen an. Die Feature-Maps werden räumlich in $m \times m$ Bins unterteilt. Dann wird ein Maximumpool auf jeden Bin für jeden Kanal angewendet. In YOLOv4 wird eine modifizierte Version von SPP verwendet die die räumliche Dimension der Ausgabe beibehaltet [11].

In den FPN aus YOLOv3 werden die Objekte separat und unabhängig voneinander auf verschiedenen Skalenebenen erkannt. Dies kann zu doppelten Vorhersagen führen. PAN erweitert FPN mit einem Bottom-up-Pfad der die Weitergabe von Informationen aus den unteren Ebenen erleichtern. Durch adaptives Featurepooling kann auf jede Vorhersagen aus allen Ebenen zugriffen werden [17].

SAM verbessert die räumliche Aufmerksamkeit.

Am *Head* der für Prädiktion verantwortlich ist wurde nichts geändert und ist damit identisch zu YOLOv3 [4][15]. Diese Architektur mit der Vielzahl an Komponenten wird in Abbildung 23 dargestellt.

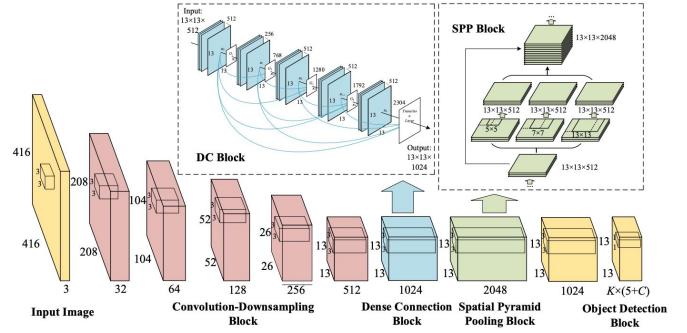


Abbildung 23. Das Konzept von YOLOv4 [15]

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 im Postprocessing herausgefiltert.

Ein weiterer Postprocessingschritt ist es die Detektion die eine zu geringe Sicherheit besitzen zu entfernen.

Die übrigen Boxen durchlaufen eine Non-Maximum-Suppression (NMS)[12]. Der Output kann überschneidende

Bounding Boxen besitzen die eigentlich zur selben Klasse gehören. Diese Duplikate können mittels der Non-Maximum-Suppression (NMS) entfernt werden. Hierfür wird die IoU der Boxen berechnet. Sollte die IoU einen Threshold überschreiten dann wird das Duplikat entfernt. Die Grafik 24 zeigt wie zu viele überlappende Boxen mit einer NMS mit einer ersetzt werden kann.

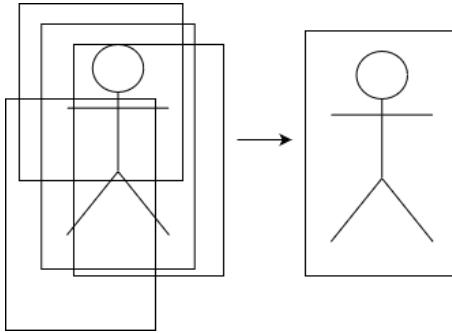


Abbildung 24. Non-Maxima-Suppression

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

A. Fehler in der Erkennung

YOLOv4 ist nicht perfekt. So gibt es gelegentlich Falschklassifizierungen, fehlende Detektionen oder unpassende Bounding Boxen. Fehlerursachen sind aber sehr schwer zu erklären, da das Netz eine Blackbox ist. Dieser Abschnitt zeigt Beispiele für diese Art von Fehler auf.

Falschklassifizierung:

Selten Klassifiziert das Netz falsch. So wie in der Abbildung 25 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 damit das die Frontpartie des Busses einem Pkw sehr ähnelt und der Rest des Fahrzeugs wegen eines Baums 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 das Projekt relevanten Klassen trainiert worden wäre. Die folgenden Abbildungen 25, 26, 27 und 28 sind Beispiele für falsche Klassifikationen. Speziell ist vor allem der Fehler in Bild 28. Hier wurde ein Anhänger als Fahrzeug klassifiziert. Der Ursprung dieses Fehlers kann unterschiedliche Gründe haben. Dies liegt wahrscheinlich auch am Datensatz selbst.

Fehlende Detektion:

Ein etwas häufigerer Fehler ist das Detektionen komplett fehlen. So zeigt die Abbildung 29 und 30 zwei Typische Gründe. Hauptgrund ist es das Objekte teilweise verdeckt sind. In Kombination, wenn Objekte sehr klein sind, ist es nahezu nicht mehr möglich zuverlässig Detektion zu



Abbildung 25. Falschklassifizierung eines Bus wegen Verdeckung

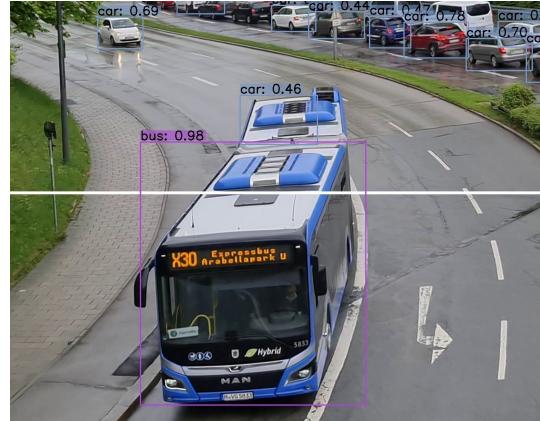


Abbildung 26. Falschklassifizierung des Dachs als Pkw



Abbildung 27. Falschklassifizierung eines Hausdachs



Abbildung 28. Falschklassifizierung eines Anhängers

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



Abbildung 29. Fehlende Klassifikationen



Abbildung 30. Fehlende Klassifikationen

Unpassende Bounding Boxen:

YOLO hat durch seine Eigenschaft nur rechteckige Bounding Boxen vorhersagen zu können eine große Schwäche. Das resultiert in manchen Szenarien zu sehr unpassenden Bounding Boxen. Beispielsweise kann man in der Abbildung 31 gut erkennen das wegen dem Busses welcher diagonal steht die Boxen sehr groß werden können. Der Effekt tritt noch verstärkter auf bei noch längere Objekten wie Zügen wie in Bild 32 zusehen. Der Fehler der im Abbildung 34 zu sehen ist hängt womöglich von Fehlern im Training zusammen. Züge wurden vielleicht an Bahnsteigen schlecht gelabelt was dem Netz dann beigebracht hat Bahnsteige als Züge zu klassifizieren. Auch kann die Box für das Objekt etwas falsch anliegen wenn Bäume oder Laternen ein Teil verdecken wie in Abbildung 33 zu sehen ist. Diese unpassenden Boxen erschweren das Tracking ungemein. Dazu später aber noch mehr.

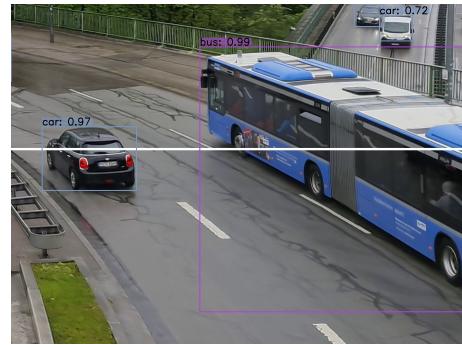


Abbildung 31. Unpassende Bounding Box



Abbildung 32. Unpassende Bounding Box



Abbildung 33. Unpassende Bounding Box



Abbildung 34. Unpassende Bounding Box

VI. TRACKING: SORT

SORT [2] berechnet auf Basis des Kalman Filters [29] und der Ungarische Methode [6] die Bewegung des Objekts und weist je nach Status eine eindeutige Identifikationsnummer zu.

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 ungarische 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 Ungarische Methode, auch Kuhn-Munkres-Algorithmus genannt. Diese 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. Die Matrix C wird anschließend reduziert. In diesem Beispiel werden Objekte den Detektionen zugeordnet. Als Gewichtung dienen die Distanzen der IoU, welche minimal sein soll [6].

VII. 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 erweitert in Form von DeepSORT [31], welches sowohl einerseits die Mahalanobis-Distanzmetrik für die Assoziation nutzt, als auch die Form des Objekts in Form eines durch das hinzugefügte 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 alle Features eines gegebenen Bilds enthält. Ursprünglich wird das Neuro-nale 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 diese durch die (quadrierte) Mahalanobis Distanz ersetzt. 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. Die Mahalanobis Distanz berechnet in dieser Anwendung die Distanz zwischen der detektierten Bounding Box und der gaußverteilten Vorsage des Zustands des Kalman-Filters:

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

Hierbei wird die Verteilung des i-ten verfolgen Objekts (zuvor acht-dimensional) in den vier-dimensionalen Raum der Messungen projiziert (y_i, S_i). Die 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

Ist die Bewegung des Objekts nicht linear steigt die Unsicherheit in der Vorhersage der Position durch den Kalman-Filter. Treten zudem Bewegungen durch die Kamera auf, kommt es zu starken Verschiebungen innerhalb der Bildebene. Die Mahalanobis Distanz ist lediglich für Bewegungen mit geringer Unsicherheit geeignet, wodurch die Nutzung dieser Metrik für die Verfolgung verdeckter Objekte ungeeignet ist, da hier die Bewegungsunsicherheit besonders groß ist. Es wird daher eine weitere Metrik eingeführt, welche die Form der Objekte vergleicht. Die Beschreibung ist ein Feature-Vektor, welcher durch ein CNN berechnet wird.

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 [31].

1) *Ergebnisse des Trackings mit DeepSORT:* Das Tracking basiert auf YOLO, daher wird es beeinträchtigt wenn YOLO Objekte nicht erkennt. Der Kalman-Filter kann zwar in Fällen von fehlenden Detektionen weiterhin eine Vorhersage liefern, allerdings nur bei gleichbleibender Bewegung in eine Richtung.

Geprüft wurde wie lange der Tracker erfolgreich stehende und fahrende Autos verfolgen kann. In den meisten Fällen ist dabei kein Problem festzustellen. Problematisch ist, wenn der Objekt-Detektor über mehrere Frames ein Objekt nicht erkennt.

In den Aufzeichnung des Brudermühl-Tunnels zeigt sich die Problematik der unregelmäßigen Detektionen deutlich. Ein Teil der Straße wird durch die Laterne verdeckt, wodurch für einige Frames keine Detektion möglich ist. Die Fahrzeuge erhalten eine neue ID. Nachdem die Laterne jedoch passiert wurde wird die alte Nummerierung teilweise wieder zurückgesetzt, sodass lediglich eine Unterbrechung erfolgt.

Des weiteren entfernt sich teilweise das Fahrzeug zu schnell während die Bounding Boxen fehlen und kann nicht mehr der zuvor festgelegten Tracking-ID zugeordnet werden. Hierbei wird ebenfalls eine neue ID zugewiesen.

Objekte können durch zu schnelle Bewegung die ID des zuvor sich an der Stelle befindenden Objekts übernehmen, dies ist zu sehen in Abbildung 36-38. Dabei wechselt Fahrzeug 6 auf die ID 69. Da ist im Abbildung 37 wieder zurück auf 6 wechselt ist die ID 69 offen. Da Fahrzeug 63 für kurze Zeit hinter der Laterne keine Bounding Box erhält, wird fälschlicherweise die ID 69 zugewiesen. Dies wird in der nächsten Abbildung 38 nicht mehr korrigiert, da es sich zu schnell entfernt.



Abbildung 35. Im Hintergrund zu sehen Auto 6, 24 und 48 die die Lampe passieren



Abbildung 36. Tracken nach passieren der Lampe von 24 und 48 erfolgreich, wechselt der ID von Auto 6 auf 69, Fahrzeug 63 im Hintergrund



Abbildung 37. Wechselt von ursprünglich 6 von 69 zurück auf 6

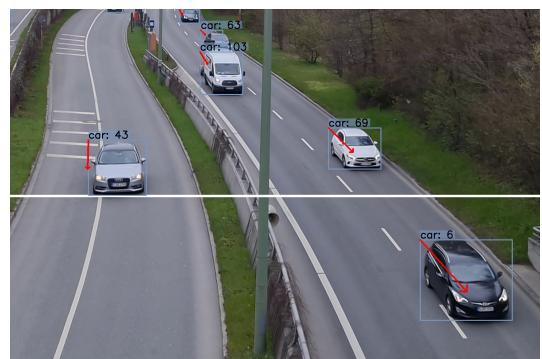


Abbildung 38. Wechsel von Fahrzeug 63 auf 69, dahinterliegende Fahrzeug erhält 63

In den Aufzeichnungen des Candid-Tunnels zeigen sich Probleme bei der Verfolgung von weiter entfernten und stehenden Fahrzeugen.

Die Verfolgung der nahen stehenden Fahrzeuge funktioniert gut, allerdings kommt es zu häufigen Änderungen der ID bei den weiter entfernten stehenden Fahrzeugen. Nahe sich bewegende Fahrzeuge können zuverlässig verfolgt werden, ebenfalls die sich gleichmäßig weiter entfernen. Im Bild groß erscheinende und sich bewegende Objekte können zuverlässig verfolgt werden.

Stehende Fahrzeuge mit fehlenden Detektionen erhalten neue IDs nach mehreren Frames. Die vorherigen IDs werden zu schnell verworfen, wodurch neue IDs erstellt werden. Der Wechsel auf neue IDs ist in den Abbildungen 39 bis 41 gezeigt.



Abbildung 39. Track-ID bei stehenden Fahrzeugen 6 und 30



Abbildung 40. Keine Detektion der Fahrzeuge 6 und 30



Abbildung 41. Änderung der Track-ID bei stehenden Fahrzeugen - 6 und 30 wechseln zu 95 und 115

In den Abbildungen 42-44 ist der Verlust der ID des blauen Trucks zu beobachten. Gleichzeitig zeigt der Richtungspfeil in die entgegengesetzte Richtung. Hier versagt der Kalman-Filter in der Richtungsbestimmung. Die Vorhersage kann nicht mehr der Detektion zugeordnet werden und eine neue ID wird gewählt



Abbildung 42. Fahrzeug mit ID 61, Richtungspfeil zeigt in entgegengesetzte Richtung



Abbildung 43. Keine Detektion des Fahrzeugs 61



Abbildung 44. Neuzuweisung der ID

Parameter die konfiguriert werden können sind die Zahl der Frames, ab wann der Track gelöscht wird. Zudem die Distanz zwischen einer Detektion und der von dem Kalman-Filter bestimmten Position. Zusätzlich kann der Wert der IoU-Distanz angegeben werden. Durch die Anpassung der maximalen IoU-Distanz kann ein häufiger Wechsel der ID mit direkt benachbarten Fahrzeugen vermieden werden.

2) *Ergebnisse des Trackings mit SORT:* Erneut basiert der Tracking-Algorithmus auf den Erkennungen des Objekt-Detektors. In Falle von SORT handelt es sich dabei um Gaussmixture. Dieser liefert teilweise für ein Objekt mehrere

Erkennung. Dies liegt daran, dass Gaussmixture bei großen Objekten nicht das gesamte Objekt erkennt, sondern einzelne Flächen und die definierten Bounding Boxen nur diese Flächen umschließen. SORT verarbeitet nur die erhaltenen Bounding Boxen und kann daher nicht erkennen ob mehrere Bounding Boxen zu einem Objekt gehören. Zu sehen ist die Problematik in Abbildung 45. Hierbei wurden zwei Bounding Boxen für den Bus definiert. Ebenfalls zu erkennen ist, dass weder das gezeigte Auto, noch der Bus von einer Bounding Box einheitlich umschlossen sind. SORT kann allerdings die definierten Boxen korrekt verfolgen bis der Bus beziehungsweise das Auto den durch Gaussmixture abgedeckten Bereich verlässt.

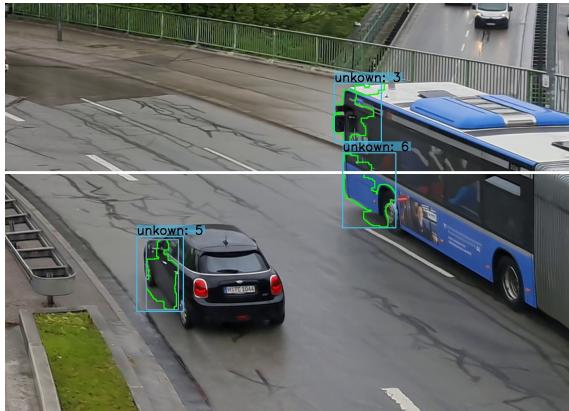


Abbildung 45. Doppelte Erkennung eines Objekts. SORT nimmt pro Bounding Box ein Objekt an, daher werden zwei Tracking-IDs für den Bus erstellt

Da dieser Bereich sehr klein ist, ist auch die zurückgelegte Strecke der Fahrzeuge nicht ausreichend, um eine Aussage über das Tracking mit SORT über längere Distanz zu treffen. Im Beispiel der Aufnahmen des Candid-Tunnels konnte außer den doppelten Erkennung keiner Fehler erkannt werden, da es hier keine Verdeckungen innerhalb des definierten Bereichs gab.

VIII. ZÄHLEN VON OBJEKTN

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, dass sobald ein Objekt gefunden wird auch gezählt wird. Wir stellten aber schnell fest, dass 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 wurde eine Zähllinie eingebaut, die sobald der Mittelpunkt der Bounding Box diese schneidet einmal hochzählt. Dies hatte noch immer Probleme wie zum Beispiel das manche Boxen doppelt gezählt wurden

weil der Mittelpunkt kurz wieder nach oben rutschte. Manchmal wurde die Linie auch direkt übersprungen.

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 Bounding Box 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 46. Darstellung einer Zählbox

IX. ZUSTANDSERKENNUNG ZÜGE

Da nun ein Objekt erfolgreich erkannt, verfolgt und seine Bewegungsrichtung erkannt werden kann, wird nun für Objekte 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. Im Verlauf des Projekts wurde die Richtungsbestimmung durch den Kalman Filter getestet. Dieser berechnet zusammen mit dem nächsten Zustand auch den Richtungsvektor, welcher genutzt werden könnte. Allerdings erkennt der Kalman-Filter nur die Bewegung der Bounding Box. Bewegt sich das Objekt direkt von der Kamera weg oder auf die Kamera zu, ändert sich die Position der Bounding Box nicht, trotz der Bewegung des Objekts innerhalb der Box. Ebenfalls erkennt der Kalman-Filter Bewegung, wenn fälschlicherweise die Bounding Box skaliert, da durch die Skalierung der Mittelpunkt der Bounding Box verschoben werden kann. Deshalb wurde weiterhin Optical Flow verwendet. Fährt der Zug ein, hat der gemittelte Richtungsvektor des Objekts durch die Bewegung eine Magnitude größer eines festgelegten Threshold. 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 eine festgelegte Anzahl an Frames geprüft ob die Richtung gleichbleibend ist. Sollte für einen bestimmten Zeitraum die Magnitude unter

den festgelegten Threshold 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.

X. VERGLEICH ZWISCHEN KLASSISCHEM METHODEN MIT DEEP-LEARNING VERFAHREN

Beide Verfahren haben ihre Stärken und Schwächen. Im Trackingteil sind beide Verfahren sich relativ ähnlich. Vor allem unterscheiden sich unsere Pipelines in der Objekterkennung welches dann aber auch maßgeblich die Trackingergebnisse beeinflusst. In der folgenden Liste werden die Vorteile und Nachteile beider Verfahren nach unserer Erkenntnissen aufgeschlüsselt.

- 1) **Genauigkeit:** Die Deep-Learning Verfahren verzeichnen vor allem gute Out-Of-The-Box Leistung. Ohne große Hyperparameteroptimierung erreicht man schnell eine ausgezeichnete Performance. Klassische Verfahren können dabei auch sehr gute Leistung erzielen wenn genug Optimierung stattfindet. Der Aufwand der Optimierung ist aber stark von der Situation abhängig.
So stellten wir fest das die klassischen Verfahren deutlich fehleranfälliger bei schlechten Videoaufnahmen waren. Verwackelte Aufnahmen stellen eine große Herausforderung dar. Gegenmaßnahmen dafür sind ein Stativ oder andere auf Software basierende Verfahren. Die Objekterkennung basierend auf YOLOv4 hat überhaupt kein Problem mit verwackelten Bildern da es jeden Frame unabhängig von einander betrachtet. Die Tracking-Algorithmen sind bezüglich starker Bewegung jedoch anfällig, da die Vorhersage der nächsten Position und der Assoziation der Detektion erschwert wird. Deep-SORT nutzt für die Assoziation bei starker Bewegung daher hauptsächlich den Feature-Vektor um die Objekte zuzuordnen. Es lässt sich sagen das die meisten Fehler der Tracker auf schlechter Objekterkennung basieren. DeepSORT ist dabei etwas weniger fehleranfällig als SORT.
- 2) **Laufzeit:** Bei der Laufzeit haben die klassischen Verfahren einen klaren Vorteil. Hier erzielten wir etwa doppelt so viele Bilder pro Sekunde als bei den Deep Learning Methoden. Außerdem dürfte es schwieriger sein Deep-Learning Verfahren auf Embedded Geräten lauffähig zu bekommen. Prinzipiell sollten beide Methoden auf Grafikkarten beschleunigt werden können.
- 3) **Nachvollziehbarkeit:** Bei Fehlern in der Objekterkennung bei YOLO ist es häufig schwierig nachzuvollziehen wie diese entstanden. Auch ist das behandeln dieser Fehler schwieriger. Bei MOG ist dies deutlich einfacher, da man nur das Graubild untersuchen muss und dann die Hyperparameter anpassen muss.
- 4) **Implementierungsaufwand:** Es kann gesagt werden das die Implementierung der klassischen Verfahren deutlich einfacher ist wie die der Deep Learning Methoden. Außerdem muss beachtet werden das ein trainiertes

Netz vorhanden sein muss. Ist dieses nicht vorhanden muss erst ein Datensatz aufwendig erstellt werden mit welchen dann ein Netz trainiert werden muss.

Zusammenfassend lässt sich sagen das beide Verfahren ihre Stärken haben. Die Entscheidung welche Methode verwendet werden sollte hängt stark von der Situation und den verfügbaren Ressourcen ab. So kann das MOG-Verfahren keine still stehenden Objekte erkennen, was zu großen Problemen führen kann. Wenn die Klasse des zu erkennenden Objekts wichtig ist, ist Deep-Learning fast alternativ los.

LITERATUR

- [1] *Background Subtraction*. 2016. URL: https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_video/py_bg_subtraction/py_bg_subtraction.html.
- [2] Alex Bewley u. a. “Simple online and realtime tracking”. In: *2016 IEEE international conference on image processing (ICIP)*. IEEE. 2016, S. 3464–3468.
- [3] *Bildstabilisierung*. URL: <https://de.wikipedia.org/wiki/Bildstabilisierung>.
- [4] Alexey Bochkovskiy, Chien-Yao Wang und Hong-Yuan Mark Liao. “YOLOv4: Optimal Speed and Accuracy of Object Detection”. In: *CoRR* abs/2004.10934 (2020). arXiv: 2004.10934. URL: <https://arxiv.org/abs/2004.10934>.
- [5] dpa. *S-Bahn-Strammstrecke in München für Arbeiten gesperrt*. 2020. URL: <https://www.zeit.de/news/2020-10/18/s-bahn-stammstrecke-in-muenchen-fuer-arbeiten-gesperrt>.
- [6] K. Erciyes. “Matching”. In: *Guide to Graph Algorithms: Sequential, Parallel and Distributed*. Cham: Springer International Publishing, 2018, S. 263–303. ISBN: 978-3-319-73235-0. DOI: 10.1007/978-3-319-73235-0_9. URL: https://doi.org/10.1007/978-3-319-73235-0_9.
- [7] *Eroding and Dilating*. URL: https://docs.opencv.org/3.4.12/db/df6/tutorial_erosion_dilatation.html.
- [8] Golnaz Ghiasi, Tsung-Yi Lin und Quoc V. Le. “Drop-Block: A regularization method for convolutional networks”. In: *CoRR* abs/1810.12890 (2018). arXiv: 1810.12890. URL: <http://arxiv.org/abs/1810.12890>.
- [9] Ross Girshick u. a. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, S. 580–587.
- [10] Kalpana Goyal und Jyoti Singhai. “Review of background subtraction methods using Gaussian mixture model for video surveillance systems”. In: *Artificial Intelligence Review* 50.2 (2018), S. 241–259.
- [11] Kaiming He u. a. “Spatial pyramid pooling in deep convolutional networks for visual recognition”. In: *IEEE transactions on pattern analysis and machine intelligence* 37.9 (2015), S. 1904–1916.
- [12] Jan Hendrik Hosang, Rodrigo Benenson und Bernt Schiele. “Learning non-maximum suppression”. In: *CoRR* abs/1705.02950 (2017). arXiv: 1705.02950. URL: <http://arxiv.org/abs/1705.02950>.

- [13] *How to Use Background Subtraction Methods*. URL: https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html.
- [14] Jonathan Hui. *Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3*. 2018. URL: <https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088>.
- [15] Jonathan Hui. *YOLOv4*. 2020. URL: <https://jonathan-hui.medium.com/yolov4-c9901eaa8e61>.
- [16] Tsung-Yi Lin u. a. “Feature pyramid networks for object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, S. 2117–2125.
- [17] Shu Liu u. a. “Path aggregation network for instance segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, S. 8759–8768.
- [18] Diganta Misra. “Mish: A Self Regularized Non-Monotonic Neural Activation Function”. In: *CoRR* abs/1908.08681 (2019). arXiv: 1908.08681. URL: <http://arxiv.org/abs/1908.08681>.
- [19] *Object Tracking*. URL: https://docs.opencv.org/3.4/dc/d6b/group_video_track.html.
- [20] *Optical Flow*. URL: https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html.
- [21] Joseph Redmon und Ali Farhadi. “YOLO9000: Better, Faster, Stronger”. In: *arXiv preprint arXiv:1612.08242* (2016).
- [22] Joseph Redmon und Ali Farhadi. “YOLOv3: An Incremental Improvement”. In: *arXiv* (2018).
- [23] Joseph Redmon u. a. “You only look once: Unified, real-time object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, S. 779–788.
- [24] *Shi-Tomasi Corner Detector & Good Features to Track*. URL: https://docs.opencv.org/3.4/d4/d8c/tutorial_py_shi_tomasi.html.
- [25] Nitish Srivastava u. a. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), S. 1929–1958.
- [26] Hamed Tabkhi, Robert Bushey und Gunar Schirner. “Algorithm and architecture co-design of Mixture of Gaussian (MoG) background subtraction for embedded vision”. In: *2013 Asilomar Conference on Signals, Systems and Computers*. IEEE. 2013, S. 1815–1820.
- [27] Abhishek Singh Thakur. *Video Stabilization Using Point Feature Matching in OpenCV*. 2019. URL: <https://learnopencv.com/video-stabilization-using-point-feature-matching-in-opencv/>.
- [28] Chien-Yao Wang u. a. “CSPNet: A new backbone that can enhance learning capability of CNN”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*. 2020, S. 390–391.
- [29] Greg Welch, Gary Bishop u. a. “An introduction to the Kalman filter”. In: (1995).
- [30] Michael Wiedenbeck und Cornelia Züll. “Klassifikation mit Clusteranalyse: Grundlegende Techniken hierarchischer und k-means-Verfahren”. In: (2001).
- [31] Nicolai Wojke, Alex Bewley und Dietrich Paulus. “Simple online and realtime tracking with a deep association metric”. In: *2017 IEEE international conference on image processing (ICIP)*. IEEE. 2017, S. 3645–3649.
- [32] Sanghyun Woo u. a. “Cbam: Convolutional block attention module”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, S. 3–19.
- [33] Sangdoo Yun u. a. “CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features”. In: *CoRR* abs/1905.04899 (2019). arXiv: 1905.04899. URL: <http://arxiv.org/abs/1905.04899>.
- [34] Sarang Zambare. *Object detection: using non-max suppression over YOLOv2*. 2019. URL: <https://medium.com/@sarangzambare/object-detection-using-non-max-suppression-over-yolov2-382a90212b51>.
- [35] Zhaozheng Zheng u. a. “Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression”. In: *CoRR* abs/1911.08287 (2019). arXiv: 1911.08287. URL: <http://arxiv.org/abs/1911.08287>.