DL2 - Objekterkennung

Niklas Heiber
Hochschule Hamm Lippstadt
Lippstadt, Deutschland
Niklas.Heiber@stud.hshl.de

***Abstract*—Diese Dokumentation enthält meinen Lösungsansatzt für die Aufgabe. Außerdem werden die verschidenen Modelle kurz beschrieben und meine Auswahl des Modells begründet. Auch gibt der Text wieder, wie YOLO antreniert und benutzt wird.**

I. Motivation

Ziel der Objekterkennung wärend der Fahrt ist, Objekte auf der Straße wie Tiere zu erkennen, um möglicherweise Unfälle zu vermeiden. Indem man mehrere Bilder macht, kann man die Distanz des Autos zu den Objekten bestimmen und es dem Auto ermöglichen, rechtzeitig zu bremsen. Außerdem hat man so vergleichswerte, wommit die bereits vorhandenen Systeme im Auto, wie zum Beispiel Entfernungssensoren, das Radar oder die Software des Autos, verbessern kann. Die Daten können auch dazu verwendet werden, häufige Gefahrenstellen auf der Straße zu finden, um andere früher warnen zu können.

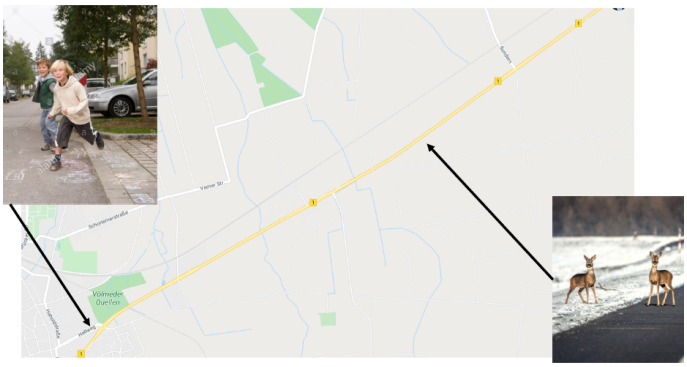

Fig. 1. Motivation

II. Use Case

In Fig. 2 ist das Usecase zusehen. Außerhalb ist das Auto, dass von der Objekterkennungssoftware PNG Bilder mit Zeitstempel und Coordinaten bekommt. Innerhalb ist die Funktion der Objekterkennung zu sehen. Die Funkltionen sind mit inludes oder extends untereinander verbunden. Der Aktor ist durch eine Assoziation mit der Funktion ”transmit information” verbunden. Wärend das Fahrzeug fährt beobachtet und identifiziert die Objekterkennungssoftware Objekte auf der Straße, macht Bilder und fügt diesen aktuelle Coordinaten des Autos sowie die aktuelle Zeit hinzu. Diese werden dann an das Auto übermittelt.

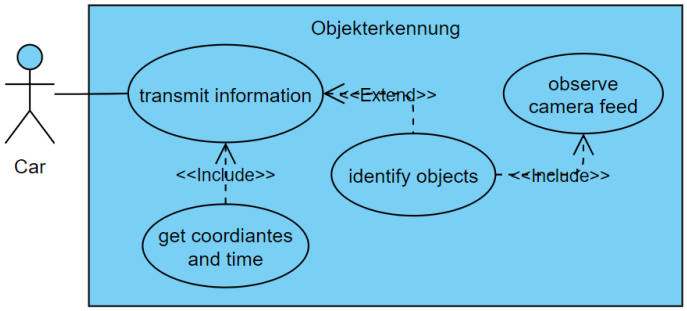

Fig. 2. Usecase

III. Aktivitätsdiagramm

Fig. 3 Zeigt das Aktivitätsdiagramm zu der Software. Zunächst wird die Kamera in dem Fahrzeug gestartet. Die überträgt dann ein Video-Feed an das Objekterkennungsprogramm. Dieses sucht dann Objekte in jedem Videobild des Videos. Dies ist mit einer Bildrate von bis 30 Bildern pro Sekunde möglich. Falls die Kamera mit erner BNildrate von 60 Bildern pro Sekunde aufnimmt, wird nur jedes zweite Bild nach Objekten untersucht. Sobald ein alle (für das Programm erkennbaren) Objekte im Bild erkennt wurden, wird das Bild zusammen mit den aktuellen Koordinaten und der aktuellen Zeit an das Auto gesendet. Sommit werden in einer Minute bis zu 1800 Bilder analysiert und an das Auto gesendet. Natürlich ist das Programm nur so schnell wie der Prozessor bzw. die GPU, die das Rechnen übernimmt.

IV. Modelle

In den letzten Jahren hat Objekterkennung viel an Bedeutung gewonnen und selbstfahrende Autos werden immer beliebter. Der Unterschied zwischen Objekterkennungsalgorithmen und Klassifizierungsalgorithmen besteht darin, dass wir bei Erkennungsalgorithmen versuchen, einen Begrenzungsrahmen um das interessierende Objekt zu zeichnen, um es im Bild zu lokalisieren. Außerdem muss bei der Objekterkennung nicht unbedingt nur einen Begrenzungsrahmen gezeichnen werden, sondern können viele Begrenzungsrahmen vorhanden sein, die verschiedene Objekte von Interesse im Bild darstellen. Es konnten keine herkömmlichen Modelle für diese Aufgabe genutzt werden, da es verschiden viele Objekte im Bild gibt und diese an verschiden stellen vorkommen und verschiedene größen aufweisen. Wenn versucht wird die Aufgabe auf herkömmliche

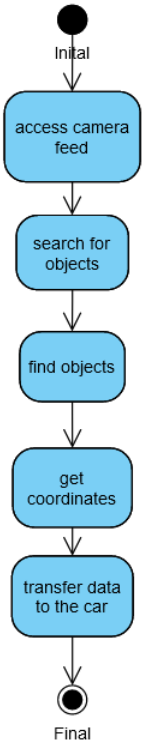

Fig. 3. Aktivitätsdiagramm

Weise zu lösen, muss das Bild in viele Regionen aufgeteilt werden, und die wird rechnerisch schnell explodieren. Deshalb wurden Algorithmen wie R-CNN, YOLO usw. entwickelt, um die Objekte im Bild zu finden, und das schnell.

A. R-CNN

Um das Problem der Regionen zu umgehen hat Ross Girshick vorgeschalgen, eine Methode zu nutzen, die nur 2000 Regionen dem Bild zuweist. Anstelle tausender Regionen im Bild zu identifizieren, werden hier nur 2000 identifiziert. Erst werden viele Regionen generiert. Diese werden mithilfe des Greedy Algorithmus in größere Regionen zusammengefast. Diese werden dann einzelnt identifiziert. Das Problem mit R-CNN ist, dass es immer noch fast eine Minute pro Bild benötigt und sommit nicht für Echtzeitanwendungen zu gebrauchen ist [1].

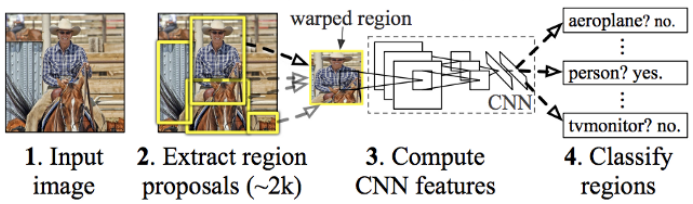

Fig. 4. R-CNN [2]

B. Fast- und Faster R-CNN

Der Ansatz ist ähnlich wie bei R-CNN. Aber anstatt die Regionsvorschläge in das CNN einzuspeisen, wird das Eingabebild in das CNN eingeführt, um eine Faltungsmerkmalskarte zu erzeugen. Aus der Faltungsmerkmalskarte wird die Region der Vorschläge identifiziert, in Quadrate verzerrt und mithilfe eines RoI-Pooling-Layers auf eine feste Größe umgeformt, damit sie in einen vollständig verbundenen Layer eingespeist werden können. Aus dem RoI-Feature-Vektor wird einen Softmax-Layer verwendet, um die Klasse der vorgeschlagenen Region und auch die Offset-Werte für die Bounding Box vorherzusagen. Anstatt jede Region einzelnt nach Objekten zu überprüfen wird nur eine Faltungsoperation gemacht, woraus dann eine Feature-Map generiert wird. Faster R-CNN benötigt nur 1/20 der Zeit des R-CNNs [2]. Faster R-CNN ist ähnlich des Fast R-CNN Models, nutzt aber ein seperates Netzwerkum, die Vorschläge für die Region vorherzusagen. Diese Modell ist nochmal 10x schneller als Fast R-CNN und ist somit im Real Time nutzbar [3].

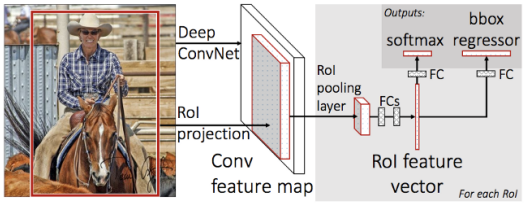

Fig. 5. Fast R-CNN [3]

C. YOLO

Frühere Erkennungssysteme verwenden Klassifizierer oder Lokalisierer um, um eine Erkennung durchzuführen. Diese wenden das Modell an mehreren Stellen und Maßstäben auf ein Bild an. Bereiche des Bildes mit hoher Bewertung werden als Erkennungen betrachtet. YOLO verwendet einen ganz anderen Ansatz. Das Modell wendet ein einzelnes neuronales Netz auf das vollständige Bild an. Dieses Netzwerk unterteilt das Bild in Regionen und sagt Bounding Boxes und Wahrscheinlichkeiten für jede Region voraus. Diese Bounding Boxes werden mit den vorhergesagten Wahrscheinlichkeiten gewichtet. Das YOLO Modell hat mehrere Vorteile gegenüber Klassifikator-basierten Systemen. Es betrachtet das gesamte Bild zur Testzeit, sodass seine Vorhersagen durch den globalen Kontext im Bild beeinflusst werden. Es macht auch Vorhersagen mit einer einzigen Netzwerkauswertung im Gegensatz zu Systemen wie R-CNN, die Tausende für ein einzelnes Bild benötigen. Yolo wurde von Joseph Chet Redmon entwickelt. Er hat aber das Entwickeln von YOLO nach der v3 aufgrund von Bedenken hinsichtlich der möglichen negativen Auswirkungen seiner Werke aufgegeben. Die drei Authoren von YOLO v4 sind Alexey Bochkovskiy, Chien-Yao Wang, und Hong-Yuan Mark Liao.

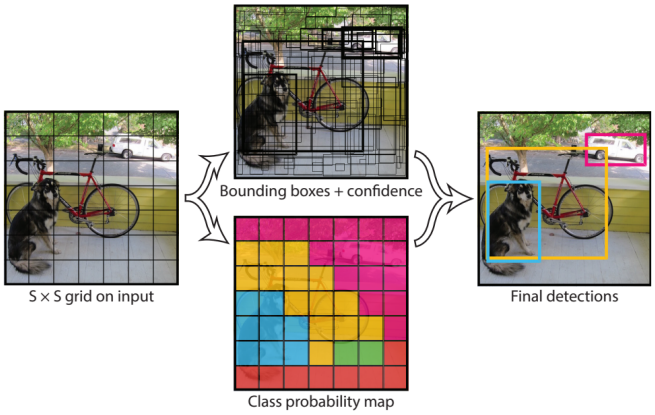

Fig. 6. YOLO [5]

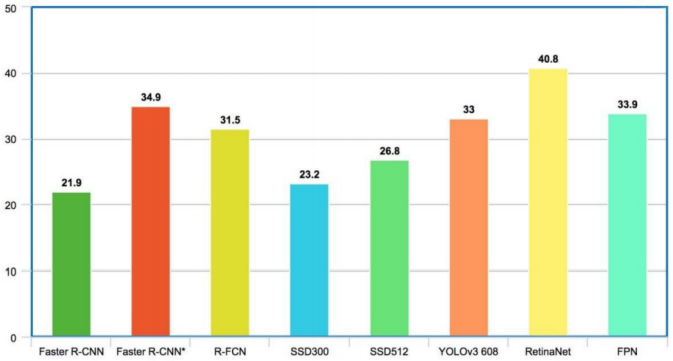

Fig. 8. Genauigeitsvergleich bei gleichem Dataset [6]

V. Warum YOLO

YOLO ist opensource, bis zu 1000 mal schneller als R-CNN und bis zu 100 mal schneller als Fast R-CNN. Die mAP (mean Average Precision) liegt bei .5 IOU (Intersection over union) und ist damit so gut wie Focal Loss, nur 4 mal schneller. Auch kann das Model schneller gemacht werden, ohne das Modell neu anzutrainieren, nur wird dabei die Genauigkeit verringert. Allerdings ist nicht alles besser. Im gegensatz zu den R-CNN Modellen ist YOLO zwar schneller, aber auch ungenauer und hat schwierigkeiten, kleine Objekte richtig zu identifizieren.

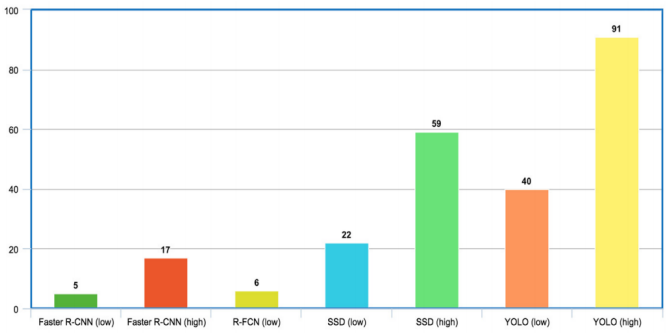

Fig. 7. Bilder pro Sekunde [6]

VI. Programm

Für die Implementierung wird Python 3, sowie OpenCV und Darknet benötigt. Es kann in Wondows MS Visual Studio, Mac oder Linux benutzt werden. Für eine wesentlich schnellere Berechnung kann man auch mit Hilfe von Nvidias CUDA Toolkit (nur Windows und Linux) die GPU statt der CPU nutzen. Allerdings werden nur eine begrenzte Menge an Software unterstüzt. Da ich eine virtuelle Maschine nutze, um Linux (Ubuntu) auszuführen, habe ich mich dagegen entschieden.

A. Installation (Linux)

Zunächst wird Python installiert. Dazu muss man legendlich die Conole öffnen und 'sudo apt-get install python3' eingeben. Um zu prüfen, ob es erfolgreich installiert wurde oder um die version zu überpüfen, kann man den Befehl 'python3 -V' ausführen. Dies gibt die Python Version aus. OpenCV wird auch benötigt. Also wieder mit einem weiteren Befehl 'sudo apt install libopencv-dev python3-opencv' OpenCV installieren. Auch hier kann die Version überpüft werden: 'opencv_version'. Für CPU und GPU support wird noch CMake benötigt. Dis wird einfach durch den Befehl 'sudo apt install cmake' installiert und kann auch mit 'cmake –version' auf die Version geprüft werden. Fall die GPU für das Modell genutzt werden soll, muss auch CUDA von Nvidia istalliert werden. CUDA ist eine von Nvidia entwickelte Parallel-Computing-Plattform und ein Schnittstellenmodell für die Anwendungsprogrammierung. Es ermöglicht Softwareentwicklern und Softwareingenieuren, eine CUDA-fähige Grafikverarbeitungseinheit für die allgemeine Verarbeitung zu verwenden. Ich werde aber nicht weiter auf CUDA eingehen, da ich es selber nicht nutze. Für informationen zur installtion von SUDA siehe: https://linuxconfig.org/how-to-install-cuda-on-ubuntu-20-04-focal-fossa-linux".
Für die GPU wird auch cuDNN benötigt: https://docs.nvidia.com/deeplearning/cudnn/archives/cudnn_741/cudnn-install/index.htm. Für CPU benutzer wird OpenMP benötigt. Der Befehl lautet 'sudo apt install libomp-dev'. Zulezt die Eigentliche Bibliothek von YOLO. Dabei werden die Dateien von Github in das Verzeichnis kopiert, auf das sich die Konsole befindet. Darknet kann mit dem Befehl 'git clone https://github.com/AlexeyAB/darknet" heruntergeladen werden. Darknet wird als Framework für das Training von YOLO verwendet, d.h. es legt die Architektur des Netzwerks fest. Nachdem die Daten heruntergeladen wurden, kann man den Order öffnen und die Datei "Makefile" bearbeiten, um z.B. OpenCV nutzen zu können oder die GPU für die berechnung zu nutzen. Falls die GPU nicht genutzt wird, sollte nur 'OPENCV', 'AVX' (CPU sppedup), 'OPENMP' (CPU speedup) und 'LIBSO' auf '1' gestellt sein. Falls mit eine ZED Kamera und SDK benutzt werden sollte, kann man auch 'ZED_Camera' auf '1' setzten. Falls die GPU genutzt wird muss 'AVX' und 'OPENMP' auf 0 stehen und stattdessen

'GPU', 'CUDNN' (GPU speedup) und 'CUDNN_HALF' (weiterer speedup für die GPU) auf '1' stehen. Sobald die Einstellungen gespeichert wurden, muss man mit Hilfe der Konsole im Verzeichnis darknet den Befehl 'make' ausführen. Jetzt können die verschiedenen bereits vorhandenen Modelle mit ihren Gewichtungen ausgefürt werden. Die Bibliothek hat bereits viele vorgefertigte Modelle zur auswahl. Darunter ist das standart YOLO, aber auch YOLOlite. Yololite hat eine bessere Performance als YOLO, aber auch einen kleineren Dataset.

```
GPU=0
CUDNN=0
CUDNN_HALF=0
OPENCV=1
AVX=1
OPENMP=1
LIBSO=1
ZED_CAMERA=0
ZED_CAMERA_v2_8=0
```

Fig. 9. CPU Make Datei

```
GPU=1
CUDNN=1
CUDNN_HALF=1
OPENCV=1
AVX=0
OPENMP=0
LIBSO=1
ZED_CAMERA=0
ZED_CAMERA_v2_8=0
```

Fig. 10. GPU Make Datei

B. Trainingsbilder

Zum starten werden Trainingsbilder benötigt. Desto weniger verschidene Objekte YOLO später erkennen soll, desto schneller und genauer wird es. Um den Objektdetektor zu trainieren, muss das lernen mit Bounding-Box-Annotationen überwacht werden. Dafür wird ein Kästchen um jedes Objekt, das der Detektor sehen soll, gezeichnet und jedes Kästchen mit der Objektklasse, die der Detektor vorhersagen soll, beschriftet. Dafür können die Programme CVAT, LabelImg oder VoTT benutzt werden. Beim Zeichen der Boxen sollte das ganze Objekt in der Box sein, aber sollte nicht viel platz zwischen der Box und dem Objekt gelassen werden.

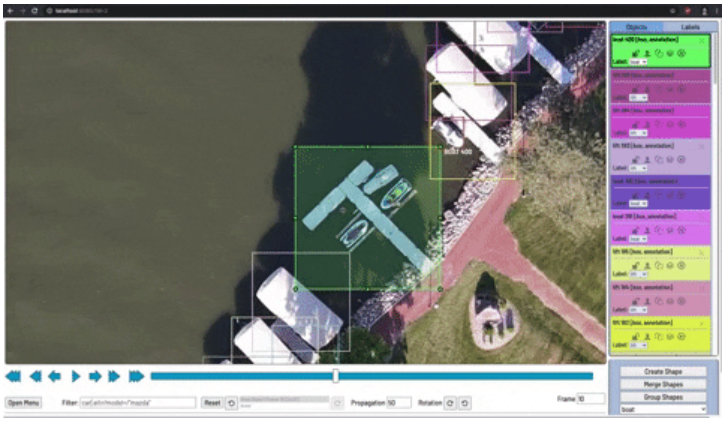

Fig. 11. Beschriften von Docks, Booten und Jetskis in CVAT

Es müssen zwei Dateien namens train.txt und test.txt erstellt werden. Diese Dateien müssen den Pfad der Trainings- und Testbildern in dem Datensatz enthalten.

C. YOLO Modell trainieren

YOLOv3 benötigt bestimmte spezifische Dateien, um zu wissen, wie und was trainiert werden soll. Es müssen diese drei Dateien erstellt werden: (.data, .names und .cfg). Zu Beginn wird die .data Datei mit dem vollgendem Inhalt erstellt. 'Classes' gibt die Anzahl der Klassen im Dataset an. 'train' und 'valid' geben den Datenpfad der Trainings- und Testdaten an. 'names' ist die .names Datei und 'backup' ist der Speicherort der YOLO Gewichtsdatei. Jetz wird noch die Yolov3.cfg benötigt. Hierfür kann man die Yolov3.cfg aus dem Darknet Order kopieren. Diese muss noch bearbeitet werden. Bei Linie 3 kann man die Anzahl der Trainingsbilder einstellen (z.B. batch=24). Bei Linie 603, 689 und 776 muss bei 'filters' eine Nummer stehen, Diese wird so berechnet: (Anzahl Klassen + 5) x3. Linie 610, 696 und 783 muss 'classes' auf die Anzahl vorhandener Klassen gesetzt werden. Jetzt müssen noch vortrainierte Geweichte heruntergelanden werden. https://pjreddie.com/media/files/darknet53.conv.74

Um das Modell zu trainieren muss legendlich der Befehl './darknet detector train train.data cfg/yolov3.cfg darknet53.conv.74' ausgefürt werden.

```
classes= 15
train  = train.txt
valid  = test.txt
names = traffic-sign.names
backup = backup/
```

Fig. 12. Die .data Datei

D. YOLO Modell benutzen

Bei der .names Datei sollte jede Kategorie in einer neuen Splate stehen. Die Spaltennmmer sollte die der .txt Datei der Testbilder übereinstimmen.

```
keine Zufahrtsstraße
Halt
Bahnhof
Kein Parken
Höchstgeschwindigkeitsbegrenzung (30 km/h)
Rechts abbiegen verboten
Keine Linkskurve
Parkplatz
Vorwärts und obligatorische richtige Richtung
Höchstgeschwindigkeitsbegrenzung (20 km/h)
Weg in den Verkehr
Obligatorische Richtung von vorne nach links
Zwangsrichtung vorwärts rechts forward
Vorwärts und links obligatorisch
Ende der Geschwindigkeitsbegrenzung (20 km/h)
```

Fig. 13. Die .names Datei

E. YOLO Modell benutzen

Um ein einzelnes Bild zu testen, wird der Befehl ’./ detect cfg/yolov4.cfg yolov4.weights data/person.jpg’ benutzt. Dabei ist ’cfg/yolov4.cfg’ die Konfiguration des Models und ’yolov4.weights’ das vortrainierte Modell. Das ’data/person.jpg’ ist das Bild das getestet werden soll. ’Detect’ heißt, es soll nur ein Bild getestet werden. Um mehrere Bilder oder ein Video zu testen, wir der Befehl ’./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights data/person.jpg -i 0 -thresh 0.25 ’ benötigt. Hier kommt die Variable ’i=0’ hinzu mit der Angabe der GPU-Nummer. Das ’thresh’ ist die Erkennungsschwelle. Die Genauigkeit der Erkennung variiert, wen dieser Wert verändert wird. Standardmäßig zeigt YOLO nur Objekte an, die mit einer Konfidenz von 0,25 oder höher erkannt wurden. Falls wie hier ’./darknet detector demo cfg/coco.data cfg/yolov4.cfg yolov4.weights test50.mp4 -i 0 -thresh 0.25’ das Wort demo eingefügt wird, öffnet sich beim ausführen ein weiters Fenster, dass den Output zeigt.

Mit ’-c 0’ am Ende kann man auch eine Kamera nutzen: ’./darknet detector demo cfg/coco.data cfg/yolov4.cfg yolov4.weights -c 0’.

VII. Schluss

YOLO ist eines der wenigen Modelle das in Echtzeit benutzt werden kann. Es kommt mit verschidenen Genauigkeiten und Geschwindigkeiten. Meiner meinung nach ist es das richtige Mittel für meine Aufgabe, auch wenn ich die Echtzeit nicht mit meinem Computer zeigen kann.

VIII. Referenzen

References

[1] Ross Girshick Jeff Donahue Trevor Darrell Jitendra Malik UC Berkeley, “Rich feature hierarchies for accurate object detection and semantic segmentation Tech report (v5)“, 22 Oct 2014

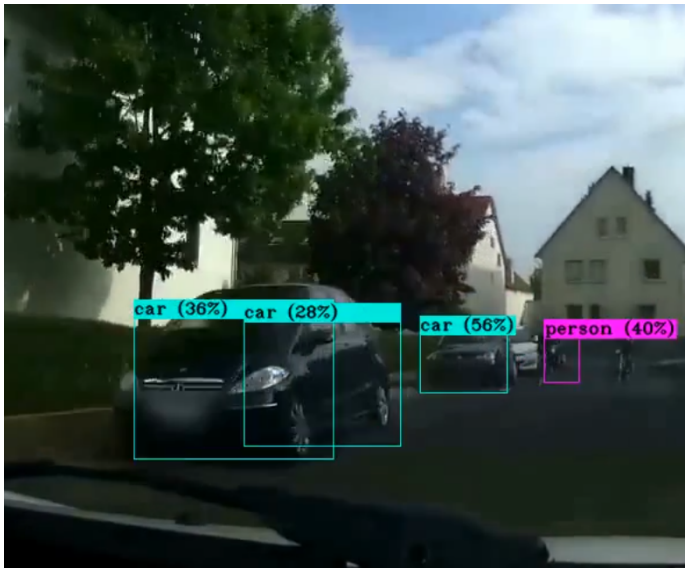

Fig. 14. YOLOlite Videobild ausgabe

[2] Ross Girshick, “Fast R-CNN“, 27 Sep 2015
[3] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks“, 06 Jan 2016
[4] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, “You Only Look Once: Unified, Real-Time Object Detection“, 09 May 2016
[5] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, “You Only Look Once: Unified, Real-Time Object Detection“, 09 May 2016
[6] S A Sanchez, H J Romero, y A D Morales, “A review: Comparison of performance metrics of pretrained models for object detection using the TensorFlow framework “, 2019

IX. Quellen

https://pjreddie.com/darknet/yolo/
https://arxiv.org/pdf/1504.08083.pdf
https://arxiv.org/pdf/1506.01497.pdf
https://arxiv.org/pdf/1506.02640v5.pdf
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf
https://arxiv.org/pdf/1311.2524.pdf
https://github.com/mathieuorhan/darknet/blob/master/README.md
https://iopscience.iop.org/article/10.1088/1757-899X/844/1/012024/pdf
https://blog.roboflow.com/cvat/