Technology
Arts Sciences
TH Köln

Bildbasierte Computergrafik

Verhalten von Objektdetektoren bei Nacht am Beispiel von YOLOv7 und Detectron2

Projektdokumentation

von

Benjamin Pagelsdorf & Moritz Langer

vorgelegt an der

Technischen Hochschule Köln
Campus Gummersbach
Fakultät für Informatik und
Ingenieurwissenschaften

im Studiengang

Medieninformatik

Gummersbach, im Februar 2023

Inhaltsverzeichnis

1 Einleitung

In der heutigen Welt ist die computergestützte Objekterkennung ein wichtiger Bereich der Künstlichen Intelligenz. Die Fähigkeit, Objekte in Echtzeit genau zu erkennen und zu lokalisieren, ist von großer Bedeutung für viele Anwendungen, zum Beispiel Überwachung, Sicherheit, Verkehr und Robotics. Es gibt viele verschiedene Algorithmen, die für die computergestützte Objekterkennung verwendet werden können, und es ist wichtig zu untersuchen, wie sie sich verhalten, um die bestmögliche Lösung für eine gegebene Anwendung zu finden. Einer dieser Algorithmen, der seit langem an der Spitze sitzt, ist YOLO (für das Projekt Yolo v7)[1]. Ein anderer Algorithmus, der in diesem Bereich immer beliebter wird, ist Facebooks Detectron2[2].

Es ist wichtig zu untersuchen, wie sich YOLO v7 und Detectron2 bei der Objekterkennung bei Nacht verhalten, da dies eine Herausforderung darstellt, insbesondere für Computer Vision-Systeme. Nachtbedingungen können dazu führen, dass Bilder undeutlich und schwer zu interpretieren sind, wegen der schlechteren Lichtbedingungen und Details. Die Frage, die deshalb gestellt werden, muss ist, ob ein spezielles Training für Nacht Verbesserungen bringt und vonnöten ist.

In diesem Projekt und der dazugehörigen Dokumentation werden YOLO v7 und Detectron 2 auf jeweils zwei identischen Datensätzen, und einer Kombination aus beiden, trainiert. Der Vergleich wird die Fähigkeit beider Algorithmen untersuchen, Objekte in dunklen Umgebungen genau zu erkennen und zu lokalisieren, und bewerten, wie gut jeder Algorithmus in Bezug auf Geschwindigkeit, Genauigkeit und Robustheit bei Nacht funktioniert und ob es sinnvoll ist, extra Trainings für die Objekterkennung bei Nacht durchzuführen.

[1] https://github.com/WongKinYiu/yolov7 (last accessed 27.02.2023)
[2] https://github.com/facebookresearch/detectron2 (last accessed 27.02.2023)

2 Übersicht der benutzen Objektdetektoren

Für die Untersuchungen im Projekt wurde sich aufgrund der Vorarbeit im Seminarteil und der technischen Vorteile dazu entschieden YOLO zu verwenden. Es wurde sich auf Version 7 verständigt, da diese zum Zeitpunkt des Projektes eine der neusten und am besten untersuchteste und verwendete Versionen von Yolo ist. Ein weiterer Pluspunkt war der Fakt, dass diese Version von Yolo auf Pytorch lauft. Für andere Vergleiche wurden sich zunächst Abwandlungen von Yolo angeschaut, wie z. B. Yolo X oder auch eine auf Nacht trainierte Yolo Variante. Diese Versionen lieferten jedoch, aufgrund der älteren Yolo Versionen auf denen sie basieren, schlechtere Ergebnisse als Yolo v7.
Für einen Vergleich wurde deshalb ein Two-Stage Detector gesucht. Bei der Research wurde dann Detectron2 gefunden, eine Weiterentwicklung von Faster-RCNN. Da die Ergebnisse aus diversen Test sehr vielversprechend aussahen, wurde sich entschieden diesen Objektdetektor zu verwenden.

2.1 YOLO v7

You Only Look Once (YOLO) v7 ist ein weit verbreiteter Objektdetektor für schnelle und präzise Objekterkennung in Echtzeit. Er ist für seine einfache Architektur und hohe Geschwindigkeit bekannt.

YOLO v7 nutzt ein einziges CNN-Netzwerk, das das gesamte Bild auf einmal verarbeitet, anstatt es in kleinere Bereiche zu unterteilen (Wang u. a., 2022). Zudem benutzt Yolo auch Anchor Boxen, diese sind vordefinierte Boxen, die verwendet werden, um potenzielle Objekte im Bild zu identifizieren (Redmon u. a., 2015). In YOLO v7 werden sie trainiert, um die Größen und Formen von Objekten zu erlernen, die häufig in den Bilddaten vorkommen.

Anchor Boxen werden auf jeder Zelle im YOLO v7-Netzwerk angewendet, um potenzielle Objekte zu identifizieren. Jede Zelle im Netzwerk erzeugt mehrere Vorhersagen, die eine Klassenprobabilät und eine Bounding Box-Koordinate für jedes potenzielle Objekt im Bild enthalten. Die Klassenprobabilität beschreibt die Wahrscheinlichkeit, dass ein bestimmtes Objekt in der Zelle vorhanden ist, und die Bounding Box-Koordinaten beschreiben die exakte Position des Objekts im Bild (Redmon u. a., 2015).

Die Anchor Boxen werden so trainiert, dass sie die möglichen Formen und Größen von Objekten abbilden, die in den Bilddaten vorkommen. Daher kann das YOLO v7-Netzwerk die Anchor Boxen verwenden, um potenzielle Objekte im Bild zu identifizieren, indem es diejenigen auswählt, die am besten zu den tatsächlichen Objekten

im Bild passen (Wang u. a., 2022).

YOLO v7 nutzt auch Non-Maxima Suppression (NMS), um sicherzustellen, dass jedes Objekt im Bild nur einmal erkannt wird, auch wenn es von mehreren Zellen identifiziert wurde. Die NMS wählt die höchste Klassenprobabilität aus, um das endgültige Objekt zu bestimmen und verwirft alle übrigen Vorhersagen (Wang u. a., 2022). Insgesamt ist YOLO v7 ein schnelles und einfaches Objektdetektions-Framework, das eine hohe Leistung bei Echtzeitanwendungen bietet. Es ist bekannt für seine Fähigkeit, eine Vielzahl von Objekten gleichzeitig zu erkennen, und seine Fähigkeit, komplexe Bildmuster mit einer einfachen Architektur zu erlernen und zu identifizieren.

2.2 Detectron2

Detectron2 ist eine von Facebooks/Metas KI Abteilung entwickelte zweistufige Objekterkennungssoftware. Detectron2 nutzt eine Region-based Convolutional Neural Network (R-CNN) Architektur zur Objekterkennung. Die R-CNN-Architektur besteht aus drei Hauptkomponenten: Region Proposal Network (RPN), Klassifikations-CNN (C-CNN) und Bounding Box Regression (Wu u. a., 2019).

1. Region Proposal Network (RPN): Das RPN nutzt eine Slide-Window-Technik, um das Bild in kleinere Bereiche zu unterteilen, und bewertet jeden Bereich auf seine Wahrscheinlichkeit, ein Objekt zu enthalten. Diese Bewertung wird durch den Einsatz eines kleinen CNN-Netzwerks durchgeführt, das eine Vielzahl von Filter-Features nutzt, um Muster in den Bilddaten zu erkennen. Die Bereiche mit der höchsten Wahrscheinlichkeit, ein Objekt zu enthalten, werden als Objektvorschläge beibehalten (Ren u. a., 2015).

2. Klassifikations-CNN (C-CNN): Die Objektvorschläge werden dann von einem größeren CNN-Netzwerk klassifiziert, um festzustellen, ob sie tatsächlich ein Objekt enthalten. Dieses Netzwerk nutzt eine Vielzahl von Schichten, um hochkomplexe Muster in den Bilddaten zu erlernen und zu identifizieren. Die CNN-Schichten können Konvolutionale Schichten, Pooling-Schichten und vollständig verbundene Schichten enthalten (Ren u. a., 2015).

3. Bounding Box Regression: Schließlich werden die Objekte, die von dem C-CNN als tatsächliche Objekte identifiziert wurden, weiter verfeinert, um ihre genauen Positionen und Formen zu bestimmen. Dieser Schritt nutzt eine Technik namens "bounding box regression", bei der die ursprünglichen Objektvorschläge weiter verfeinert werden, um die endgültigen Objekte und ihre Positionen im Bild genauer zu bestimmen (Wu u. a., 2019).

Insgesamt nutzt Detectron2 eine Kombination aus RPN, C-CNN und Bounding Box Regression, um eine hohe Genauigkeit bei der Objekterkennung zu erreichen. Es kann auch mehrere bekannte CNN-Architekturen wie FPN (Feature Pyramid Network), RetinaNet und Mask R-CNN unterstützen, um eine Vielzahl von Anwendungen zu ermöglichen, einschließlich Instance Segmentation, Object Detection und Keypoint Detection (Wu u. a., 2019).

3 Durchführung

3.1 Datensätze

Im Rahmen des Objekterkennungsvergleichs wurden zwei Datensätze verwendet: ein Tag-Datensatz (khermz, 2022) mit 5256 Bildern und ein Nacht-Datensatz (Uni, 2022)mit 5000 Bildern. Ein Vorteil hierbei ist, dass diese Datensätze in verschiedene Formate exportiert werden können. Somit können beide Datensätze für YOLO und Detectron2 eingesetzt werden. Um eine optimale Trainings-, Validierungs- und Testmenge zu gewährleisten, wurden die Daten im Verhältnis 70% Trainings-, 20% Validierungs- und 10% Testmenge aufgeteilt.

Da die beiden Datensätze eine unterschiedliche Anzahl von Klassen aufwiesen, mussten die Labels angepasst werden. Unter Klassen werden die verschiedenen Arten von marktieren Objekten verstanden. Die ersten Fünf waren in beiden Datensätzen deckungsgleich. Alle anderen Klassen wurden entfernt, und ein entsprechendes Skript wurde geschrieben, um dies automatisch zu erledigen. Zudem wurden Labels im Nacht-Datensatz umbenannt, um sie an den Tag-Datensatz anzugleichen. „Other person"wurde zu „Person“, „Bikecycle"wurde zu „Bike“und „Motorcycle"wurde zu „Motor“ umbenannt.

Zusätzlich wurden die Tag- und Nacht-Datensätze zusammengeführt, um den Vergleich der Objekterkennungsalgorithmen in beiden Szenarien zu ermöglichen. Dank der zuvor durchgeführten Anpassungen waren die beiden Datensätze direkt und ohne Probleme verbindbar. Ebenso wie die anderen beiden Datensätze, wurde der kombinierte Datensatz in Trainings-, Validierungs- und Testmengen aufgeteilt.

3.2 Trainngsumgebung

In der Evaluation von Cloud-basiertem Training wurden zwei Plattformen betrachtet: Google Colab und Amazon Sage.

Google Colab ist eine Plattform, die in der kostenfreien Version längere und intensivere Trainings, welche länger als etwa drei Stunden dauern, nicht erlaubt. Diese Erfahrung sammelten wir als unser Training nach genannten drei Stunden nicht mehr fortzusetzen war. Das Kostenmodell von Google Colab ist auch undurchsichtig, da es etwa 10 € pro Monat für 100 Recheneinheiten berechnet. Allerdings ist nicht klar, wie viel eine Recheneinheit genau ist und wie viel schneller das Training auf Google Colab im Vergleich zu einer lokalen GPU ist.

Amazon Sage ist für Neukunden im ersten Monat für ein bestimmtes Kontingent kostenfrei. Allerdings ist die Einrichtung der Plattform unübersichtlich, und es gibt wenig verfügbare Hilfestellung. Die Kostentransparenz von Amazon Sage ist ebenfalls schlecht. Aus diesen Gründen wurde das Training auf einem lokalen PC mit einer Nvidia GeForce RTX 2060 6GB GPU durchgeführt, da dies die beste Option in Bezug auf Kosten, Geschwindigkeit und Transparenz war.

3.2.1 YOLO

Für das (Transfer-)Training der Yolov7-Modelle wurden die Standardparameter verwendet, die auf GitHub verfügbar sind. Wie auf der entsprechenden Seite[1] zu sehen ist. Während des Trainings mit YOLO wurden verschiedene Methoden ausprobiert, um das Training zu optimieren. Eine dieser Methoden war das Einfrieren von Schichten (Freezing von Layern). Diese Methode brachte jedoch keine guten Ergebnisse, wie auf GitHub zu sehen ist (Langer u. Pagelsdorf, 2023).

Insgesamt verlief das Training mit YOLO ziemlich gut und unkompliziert. Im Gegensatz zu anderen Objekterkennungsalgorithmen wie beispielsweise Detectron 2 benötigte YOLO kein extra Python-Skript, sondern konnte vollständig über die Befehlszeile aufgerufen werden. Das machte das Training einfach und transparent.

3.2.2 Detectron 2

Beim Training mit Detectron2 waren einige Herausforderungen zu bewältigen. Zunächst musste ein Python Code geschrieben werden, der die Datensätze lädt und ein Konfigurationsobjekt sowie die Instanz des Trainers erstellt. Da Detectron2 eine Vielzahl von Möglichkeiten bietet, wie zum Beispiel verschiedene Anwendungsgebiete und Netze, musste aus dem Modell Zoo das passende ausgewählt werden. Im Gegensatz zu YOLO arbeitet Detectron2 mit Iterationen statt Epochen. Bei der Definition der Anzahl der Klassen gibt es einen Unterschied zwischen YOLO und Detectron2. Letzteres benötigt eine zusätzliche Klasse, da es Superklassen gibt, was bedeutet, dass die Anzahl der Klassen bei Detectron2 um 1 höher ist als bei YOLO. Eine Auseinandersetzung mit den Konfigurationsparametern von Detectron2 ist unbedingt erforderlich, um das Training erfolgreich zu gestalten. Als Basis wurde das Modell [2] verwendet, das im Modell Zoo zu finden ist.

3.3 Trainingsdurchführung

Die Durchführung des Trainings der YOLOv7-Modelle hat für den Tagdatensatz insgesamt ca. 8 Stunden und 20 Minuten gedauert, bei einer Dauer von 2 Minuten und 30 Sekunden pro Epoche und insgesamt 200 Epochen. Für den Nachtdatensatz betrug die

[1] `https://github.com/WongKinYiu/yolov7#transfer-learning`(last accessed 21.02.2023)

[2] `https://github.com/facebookresearch/detectron2/blob/main/configs/COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml` (last accessed 22.02.2023) und `https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md#faster-r-cnn->R50-FPN` (last accessed 21.02.2023)

Trainingsdauer ca. 11 Stunden und 40 Minuten mit 3 Minuten und 30 Sekunden pro Epoche. Beim Training mit dem kombinierten Datensatz aus Tag und Nacht betrug die Dauer pro Epoche ca. 6 Minuten und 20 Sekunden und insgesamt ca. 21 Stunden und 7 Minuten für 200 Epochen.

Das Training von Detectron2 basiert nicht auf Epochen, sondern auf Iterationen. Für das Training der Modelle mit Detectron2 wurden insgesamt 100 Epochen trainiert. Bei einer Dauer von 0,61 Sekunden pro Iteration betrug die Trainingsdauer für den Tagdatensatz insgesamt ca. 15 Stunden und 30 Minuten. Für den Nachtdatensatz betrug die Dauer ca. 15 Stunden und 22 Minuten mit einer Iterationsdauer von 0,64 Sekunden. Um die Anzahl der Iterationen zu berechnen, wurde die Anzahl der Trainingsbilder, die Anzahl der Epochen und die Batch-Größe verwendet.

3.4 Probleme

Einige Probleme traten während der Durchführung des Trainings auf. Zunächst funktionierte das Training am PC nicht, was zur Suche nach einer geeigneten Cloudplattform führte und extra Zeit in Anspruch genommen hat. Des Weiteren ergab sich aus der Erkenntnis, dass die Klassen beider Datensätze angeglichen werden mussten eine weitere Iteration. Da diese Erkenntnis erst nach dem Training einiger YOLO-Modelle gewonnen wurde, musste später alles neu trainiert werden. Die Trainings auf dem PC dauerten zudem sehr lange, wie aus den Trainingszeiten hervorgeht. Das Angleichen der Objektklassen für Detectron2 erforderte auf Grund eines anderen Formats zur Annotation der Objekte mehr Zeitaufwand als erwartet. Hierzu benötigt man ein eigenes Script, welches die entsprechenden Objektklassen aus der entsprechenden Datei entfernt. Genauso stellte die Installation von YOLOv7 mit CUDA eine Herausforderung dar. Hierfür waren einige Recherchen bezüglich Nvidia CUDA Toolkit mit GPU-Kompatibilität und PyTorch erforderlich. Ähnlich verhielt es sich bei der Installation von Detectron2 auf Windows, da es hierfür keinen offiziellen Support für Windows gibt.

4 Ergebnisse

Die Ergebnisse basieren auf den fünf besten Modellen der Objektdetektoren. Drei davon sind YOLO Modelle, die beiden anderen Modelle resultieren aus Detectron2. Bei YOLO sind die Modelle jeweils mit dem Datensatz für Tag, Nacht oder kombiniert trainiert worden. Mit Detectron2 verhält es sich ähnlich. Ausnahme hingegen ist, dass aufgrund der genannten Probleme kein Modell für den kombinierten Datensatz möglich war.

4.1 YOLO

Die nachfolgenden Tabellen 4.1, 4.2, 4.3 zeigen, wie präzise ein Netz auf dem jeweiligen Testdatensatz arbeitet. Die Spalten Tag, Nacht und Kombiniert geben dabei an, welches Modell validiert wurde. Die Spalte Labels gibt an, wie viele Objektannotationen es im Testdatensatz gibt. Der Testdatensatz für Tag beträgt 719 Bilder, für Nacht 1000 Bilder und kombiniert entsprechend 1719 Bilder.

Klassen	**Labels**	**Tag mAP50**	**Nacht mAP50**	**Kombiniert mAP50**
all	9703	56,90%	*37,10%	56,50%
bike	87	46,10%	25,10%	44,10%
bus	184	55,50%	31,80%	55,60%
car	8005	80,70%	70,90%	81,70%
motor	35	39,20%	20,60%	40,70%
person	1392	62,90%	0%	60,30%

Tabelle 4.1: YOLOv7 Ergebnisse mit Testdatensatz für Tag
*eigentlich beträgt die mAP50 29.70%

Die Tabelle 4.1 zeigt die Ergebnisse mit dem Testdatensatz für Tag. Hierbei ist zu erkennen, dass das für ausschließlich Nacht trainierte Modell weniger genau ist als die anderen. Dennoch liefert dieses Modell relativ gute Ergebnisse. Hierzu betrachten ist die Objektklasse *car*, die eine Genauigkeit von fast 71% aufweist. Ein Grund zugunsten des Ergebnisses könnte im Traningsdatensatz begründet sein, da, im Vergleich zu den anderen Klassen, viele Annotationen zu *car* existieren. Weiterhin anzumerken ist die korrigierte mAP, in der die Klasse *person* nicht enthalten ist. Hierzu sind im Datensatz für Nacht kaum *person* Annotationen zu finden.

In Tabelle 4.2 sind die Ergebnisse mit dem Testdatensatz für Nacht enthalten. Die Klasse *person* besitzt keine Ergebnisse, da, wie erwähnt, im Datensatz für Nacht fast keine Annotationen dieser Klasse gibt. Hier ist es interessant, wie genau das Netz für Tag auf den Nachtdaten arbeitet und sogar zum Teil, das für Nacht trainierte Modell

übertrifft. Darüber hinaus arbeitet das kombinierte Modell über allen Klassen hinweg am besten.

Klassen	Labels	Tag mAP50	Nacht mAP50	Kombiniert mAP50
all	9238	41,00%	47,60%	51,70%
bike	86	38,10%	34,00%	39,90%
bus	103	39,40%	44,70%	49,90%
car	9035	69,20%	76,00%	78,40%
motor	14	17,30%	35,80%	38,50%
person	-	-	-	-

Tabelle 4.2: YOLOv7 Ergebnisse mit Testdatensatz für Nacht

Die Tabelle 4.3 stellt die Ergebnisse für den kombinierten Testdatensatz dar. Hier siegt ebenfalls das Modell mit den kombinierten Trainingsdaten, gefolgt vom Tag-Modell. Den Schluss macht das auf Nacht trainierte Netz. Auffallend sind die Genauigkeiten bei der Klasse *car*. Tag und Nacht Modell haben nur eine Differenz von 1,5% und sind ebenso nahe an den 80%, die das beste Modell liefert. Ursache kann wieder die Anzahl der Labels für *car* sein.

Klassen	Labels	Tag mAP50	Nacht mAP50	Kombiniert mAP50
all	18941	48,30%	32,10%	54,40%
bike	173	41,50%	26,20%	41,40%
bus	287	49,10%	36,10%	53,10%
car	17040	74,90%	73,40%	80,00%
motor	49	34,70%	24,60%	39,40%
person	1392	45,70%	0,07%	58,10%

Tabelle 4.3: YOLOv7 Ergebnisse bei kombinierten Testdaten

Abschließend betrachtet besitzt das Modell mit beiden Datensätzen die besten Ergebnisse, benötigt hingegen auch fast die doppelte Trainingszeit. Im Vergleich Tag und Nacht hat das auf Tag trainierte Netz die höhere Genauigkeit.

4.2 Detectron 2

Die Tabellen 4.4 und 4.5 zeigen, wie präzise Detectron2 auf dem jeweiligen Testdatensatz arbeitet. Die Spalten Tag und Nacht geben dabei an, welches Modell validiert wurde. Detectron2 liefert eine leicht andere Metrik zu Evaluation als YOLO. Der AP Wert ist der Mittelwert aller Präzisionen mit einer *Intersection over Union* (IoU) von 50% bis 95%. Die Schrittweite beträgt dabei 5%. Der AP50 Wert ist die mittlere Präzision bei einer IoU von 50%. In den Tabellen sind die AP50 Werte für die einzelnen Klassen auf Basis der AP und AP50 Werte von *all* interpoliert. Insgesamt beinhaltet der Testdatensatz für Tag 526 Bilder und für Nacht 500 Bilder.

Klassen	Labels	Tag AP	Tag AP50	Nacht AP	Nacht AP50
all	9369	17,76%	33,38%	8,56%	15,76%
bike	101	9,96%	*18,72%	4,80%	*8,84%
bus	138	21,49%	*40,40%	6,94%	*12,78%
car	7877	31,66%	*59,52%	25,17%	*46,36%
motor	66	9,99%	*18,78%	5,87%	*10,81%
person	1187	15,67%	*29,46%	0%	*0%

Tabelle 4.4: Detectron2 Ergebnisse mit Testdatensatz für Tag
*Werte sind auf Basis von *AP all* und *AP50 all* interpoliert

In der Tabelle 4.4 sind die Ergebnisse mit dem Testdatensatz für Tag. Das Nacht-Modell liefert im Gesamtvergleich um Tag-Modell nur halb so gute Werte. Unter Betrachtung der Objektklasse *car* ist die Differenz beider Netze nicht so groß wie im Gesamtvergleich. Hier spielt wieder die starke Repräsentation von *car* im Trainingsdatensatz eine Rolle.

Klassen	Labels	Tag AP	Tag AP50	Nacht AP	Nacht AP50
all	4450	15,68%	30,51%	22,10%	40,14%
bike	21	11,04%	*21,48%	9,45%	*17,16%
bus	56	19,90%	*38,72%	26,99%	*49,02%
car	4362	28,68%	*55,80%	37,71%	*68,48%
motor	11	3,42%	*6,65%	14,26%	*25,90%
person	-	-	-	-	-

Tabelle 4.5: Detectron2 Ergebnisse mit Testdatensatz für Nacht
*Werte sind auf Basis von *AP all* und *AP50 all* interpoliert

Die Tabelle 4.5 liefert die Ergebnisse mit dem Testdatensatz für Nacht. Allgemein spiegeln sich die Resultate im Vergleich zu Tabelle 4.4. Es lässt sich jedoch feststellen, dass das Modell für Tag insgesamt bessere Genauigkeiten produziert als das Modell für Nacht auf dem Tag-Datensatz.

Zusammenfassend für Detectron2 zeigt sich ebenfalls, dass das auf Tag trainierte Netz im Vergleich genauere Werte in einer inversen Umgebung liefert als das Netz auf Nachtdaten.

5 Fazit

Im Verlauf des Projekt gab es einige Herausforderungen zu bewältigen, um das gesteckte Projektziel zu erreichen. Beginnend mit der Suche nach geeigneten Datensätzen. Wichtig dabei war es zwei Datensätze zu finden, welche gleich groß waren und den gleichen Anwendungskontext besitzen. Anschließend musste festgestellt werden, dass auch die Anzahl der Objektklassen eine wichtige Rolle bei der Evaluation der jeweiligen Netze spielt, weshalb in einer weiteren Iteration die Annotationen in den Datensätzen angepasst wurden. Darüber hinaus gestaltete sich das Finden nach einer passenden Plattform zum Trainieren der Objektdetektoren als zeitintensiv. Zu Beginn stand eine erfolglose Einrichtung auf dem lokalem Computer, weshalb nach einer Alternative gesucht wurde. Eine passende und kostenlose Cloud Plattform wurde nicht gefunden, weshalb ein erneuerter Versuch auf dem lokalen Computer unternommen wurde und mit Erfolg endete.
Neben den vielen Problemen konnten zusätzlich zu den Ergebnissen weitere Erkenntnisse gewonnen werden. Auf Basis der gewonnen Ergebnisse ist erkennbar, das Objekterkennung bei Nacht ohne explizites Training möglich ist. Es zeigt sich, dass das Modell auf Tag insgesamt besser auf Nacht arbeitet, als das Modell von Nacht auf Tag. Weiterhin konnte die Erkenntnis gewonnen werden, dass nicht nur die Anzahl der Bilder relevant ist, sondern auch die Anzahl der Annotationen zu einer Objektklasse eine wichtige Kennzahl ist.

Tabellenverzeichnis

Literaturverzeichnis

[khermz 2022] KHERMZ: *bdd Dataset*. https://universe.roboflow.com/khermz2012-gmail-com/bdd-ujnwc. Version: nov 2022. https://universe.roboflow.com/khermz2012-gmail-com/bdd-ujnwc. – visited on 2023-02-21

[Langer u. Pagelsdorf 2023] LANGER, Moritz ; PAGELSDORF, Benjamin: *Evaluation of Objectdetection by night*. https://github.com/BenPag/object_detection_by_night_eval. Version: 2023. – last accessed 28.02.2023)

[Redmon u. a. 2015] REDMON, Joseph ; DIVVALA, Santosh ; GIRSHICK, Ross ; FARHADI, Ali: *You Only Look Once: Unified, Real-Time Object Detection*. http://dx.doi.org/10.48550/ARXIV.1506.02640. Version: 2015

[Ren u. a. 2015] REN, Shaoqing ; HE, Kaiming ; GIRSHICK, Ross ; SUN, Jian: *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. http://dx.doi.org/10.48550/ARXIV.1506.01497. Version: 2015

[Uni 2022] UNI: *rmsw_5k_night Dataset*. https://universe.roboflow.com/uni-koped/rmsw_5k_night. Version: nov 2022. https://universe.roboflow.com/uni-koped/rmsw_5k_night. – visited on 2023-02-21

[Wang u. a. 2022] WANG, Chien-Yao ; BOCHKOVSKIY, Alexey ; LIAO, Hong-Yuan M.: *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. http://dx.doi.org/10.48550/ARXIV.2207.02696. Version: 2022

[Wu u. a. 2019] WU, Yuxin ; KIRILLOV, Alexander ; MASSA, Francisco ; LO, Wan-Yen ; GIRSHICK, Ross: *Detectron2: A PyTorch-based modular object detection library*. https://ai.facebook.com/blog/-detectron2-a-pytorch-based-modular-object-detection-library-/. Version: 2019. – last accessed 27.02.2023)