Detekcija dinamičkih objekata u prometu u ovisnosti o njihovoj udaljenosti i vremenskim uvjetima

Đurašinović, Luka

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

LUKA ĐURAŠINOVIĆ

Detekcija dinamičkih objekata u prometu u ovisnosti o njihovoj udaljenosti i vremenskim uvjetima

Diplomski rad

Pula, rujan, 2023. godine

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

LUKA ĐURAŠINOVIĆ

Detekcija dinamičkih objekata u prometu u ovisnosti o njihovoj udaljenosti i vremenskim uvjetima

Diplomski rad

JMBAG: 0069064282, redoviti student
Studijski smjer: Informatika
Kolegij: Neuronske mreže i duboko učenje
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: doc.dr.sc Goran Oreški

Pula, rujan, 2023. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani **Luka Đurašinović**, kandidat za **magistra informatike,** ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljeni način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

Đurašinović

U Puli, 05.09.2023.

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, **Luka Đurašinović** dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom

“ **Detekcija dinamičkih objekata u prometu u ovisnosti o njihovoj udaljenosti i vremenskim uvjetima**“

koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 05.09.2023.

Potpis

Đurašinović

Sažetak

Cilj ovog rada bio je ispitati uspješnost detekcije pješaka i vozila u prometu putem slika prikupljenih iz CARLA simulatora, odnosno ispitivanje primjenjivosti CARLA simulatora za primjenu u stvarnim situacijama treniranja i testiranja sustava unutar autonomnih vozila. Ispitivanje uspješnosti detekcije objekata odrađeno je u dvije kategorije: detekcija objekata u zavisnosti o njihovoj udaljenosti, te detekcija objekata u zavisnosti o vremenskim uvjetima. Svi podaci prikupljeni iz CARLA simulatora su trenirani i testirani putem YOLO algoritma koji je jedan od najpoznatijih i najpouzdanijih algoritama za detekciju objekata u stvarnom vremenu.

Ključne riječi: CARLA, YOLO, klasa, vozilo, pješak, slika, detekcija objekata, klasifikacija, udaljenost, vremenski uvjeti, trening, validacija, test, učenje, preciznost, odziv, gubitak

Summary

The goal of this paper was to examine the success of pedestrian and vehicle detection in traffic by using images collected from CARLA simulator, in other words - to examine applicability of CARLA simulator for usage in real-life situations of training and testing systems within autonomous vehicles. Successfullness test of object detection was done in two categories: detection of objects depending of their distance and detection of objects depending on weather conditions. All the data collected from CARLA simulator was trained and tested through YOLO algorithm which is one of most recognised and reliable algorithms for real-time object detection.

Keywords: CARLA, YOLO, class, vehicle, pedestrian, image, object detection, classification, distance, weather conditions, training, validation, test, learning, precision, recall, loss

Sadržaj

1. Uvod

Većina prometnih nesreća uzrokovana je ljudskom pogreškom. Čovjek često, svjesno ili nesvjesno, precjenjuje svoje vozačke sposobnosti i brzinu reagiranja što uslijed nenadanih situacija i prilika na cesti dovodi do lakših ili težih prometnih nezgoda.

Prvenstveno iz tih razloga, ali i iz potrebe za što većim rasterećenjem čovjeka od svakodnevnih obaveza i briga počelo se sve više ulagati u razvoj tzv. autonomnih tj. „samovozećih“ vozila.

Prema općoj definiciji: *„Autonomno vozilo sposobno se navoditi i kretati bez ljudskog utjecaja“ (*Ilková, 2017). Iz navedenog jasno je na koji način nam takva vozila mogu znatno olakšati svakodnevni život te ga učiniti sigurnijim, pošto autonomna vozila ne bi trebala biti podložna ljudskoj grešci. Ipak, je li to stvarno tako?

Naravno da je svako računalo prvenstveno stvoreno od strane čovjeka, pa su tako i računalni sustavi unutar autonomnih vozila postavljeni i trenirani od strane čovjeka. Jasno je stoga da u tom procesu može doći do određenih nepravilnosti ili previda.

Svako autonomno vozilo funkcionira na principu senzora, tj. određenog broja specifično postavljenih kamera raznih vrsta, kao i velikog broja senzora pokreta, zvuka i sličnog. Sve navedene kamere i senzori moraju prvenstveno biti fizički precizno kalibrirani i postavljeni na samo vozilo, a potom i računalno „trenirani“ kako bi pravilno raspoznavali svoju okolinu i pravovremeno reagirali na sve svakodnevne situacije i neočekivane ishode u cestovnom prometu. Sva ta postavljanja izvode se od strane čovjeka.

Kako je navedeno, mogućnost ljudske pogreške, koliko god pojedinac bio precizan i stručan u svom poslu je neisključiva. Stoga, prilikom ispitivanja računalnih modela za upravljanje autonomnim vozilima potrebno je koristiti različite simulatore i testne alate kako bi se minimizirala mogućnost materijalne štete i potencijalnih ljudskih žrtvi prilikom stvarnog cestovnog ispitivanja vozila.

Upravo za te namjene koristi se CARLA simulator razvijen od strane Sveučilišta u Barceloni. Unutar CARLA simulatora moguće je vjerno simulirati prikaz odvijanja prometa i kretanja različitih tipova vozila i pješaka u nekom od računalno generiranih gradova. Naime, CARLA i njoj slični simulatori koriste se za prikupljanje podataka tj. slika, videa i zvukova, u svrhu strojnog učenja, odnosno treniranja autonomnog vozila za pravovremene i valjane reakcije u prometu.

U nastavku ovog rada koristit će se upravo CARLA u svrhu postizanja što veće točnosti i sposobnosti prepoznavanja pješaka i drugih vozila u prometu prilikom kretanja autonomnog vozila. Cilj je ispitati uspješnost detekcije objekata u prometu u zavisnosti o njihovoj udaljenosti od našeg autonomnog vozila, kao i u zavisnosti o vremenskim prilikama postavljenim unutar samog simulatora.

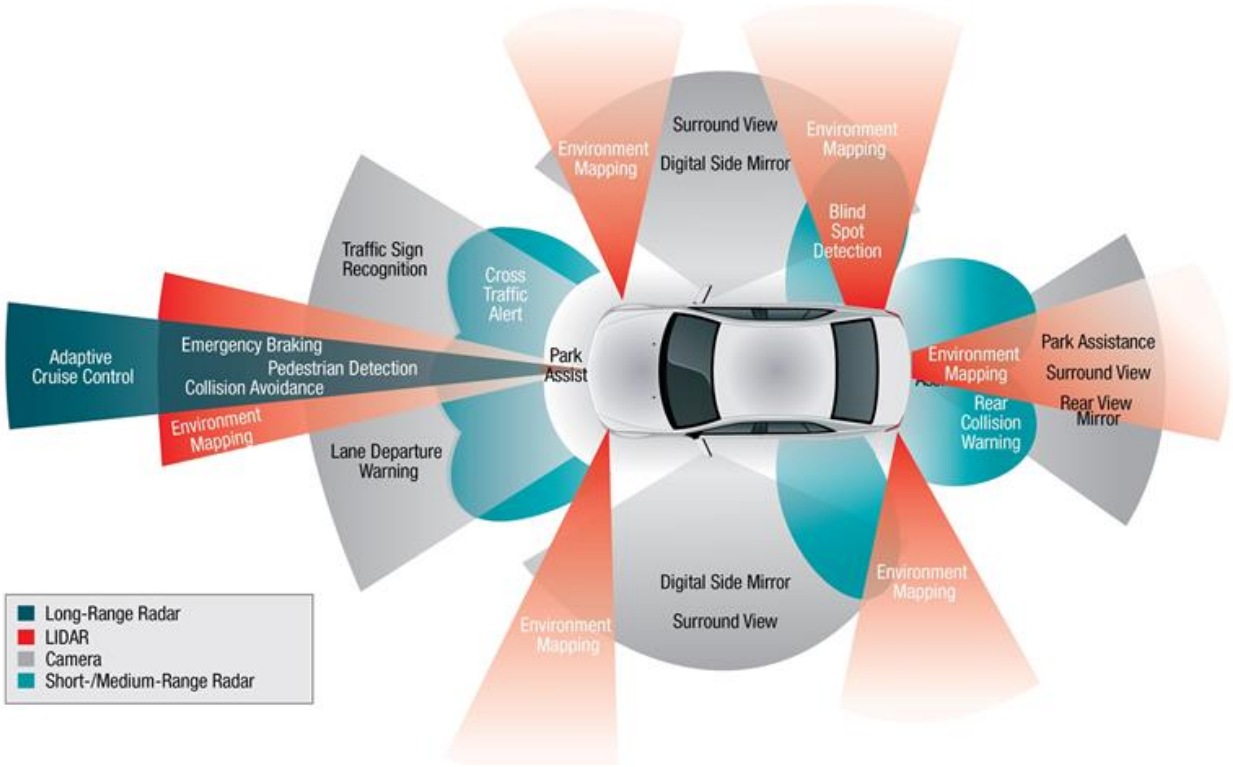
Kako bi se korisnost dobivenih podataka uspješno ispitala koristit će se YOLO *(“You Only Look Once”*) algoritam koji se smatra jednim od najboljih i najpouzdanijih modernih algoritama za detekciju objekata. Ispitivanje podataka vršit će se tako da se kroz neuronsku mrežu YOLO algoritma propuste dobivene slike iz CARLA simulatora, te se kroz treniranje modela i njegovu završnu validaciju dobiju rezultati koji pokazuju kolika se uspješnost detekcije pojedinih objekata može postići s obzirom na postavljene uvjete.

Ako se postigne zadovoljavajuća razina točnosti prilikom treniranja i validacije nad dobivenim podacima, to bi ujedno dokazalo korisnost i primjenjivost CARLA simulatora za stvarna, „ozbiljnija“ testiranja prilikom razvoja takvih vozila od strane njihovih proizvođača.

2. CARLA simulator i autonomna vozila

2.1. Princip rada autonomnih vozila

Kao što je u rečeno u uvodu, autonomna vozila su ona vozila koja zahtijevaju minimalnu ili nikakvu interakciju čovjeka pri njihovom upravljanju. Kako bi se osigurala najviša moguća razina autonomnosti i sigurnosti u prometu - kako za putnike unutar samog vozila, tako i za ostala vozila i pješake u blizini autonomnog vozila, potrebna je najsofisticiranija tehnologija unutar i van vozila. To podrazumijeva veliki broj kamera i senzora raznih namjena koji s najvišom mogućom preciznošću tijekom vožnje promatraju i osluškuju okolinu vozila u punom kružnom radijusu od 360° (Slika 1).



Slika 1 - Prikaz kamera i senzora autonomnog vozila,

Izvor: https://towardsdatascience.com/how-to-make-a-vehicle-autonomous-16edf164c30f

Naravno, sve te kamere i senzori ne mogu sami po sebi ispravno prepoznavati objekte u prometu, niti znati sva prometna pravila i moguće opasnosti bez da ih se prethodno ne isprogramira i podesi da rade to za što su namijenjeni. Tu dolazimo do onoga što je unutar vozila (i samih senzora), a to su njihovi trenirani sustavi.

Sva znanja koja imaju vanjske kamere i senzori vozila dobivena su „treniranjem“ računalnih modela putem strojnog učenja, koji se potom ugrađuju u same senzore i računalne sustave unutar vozila. Autonomna vozila sa svim svojim dijelovima moraju proći intenzivne i dugotrajne faze ispitivanja i treniranja, prvenstveno preko za to namijenjenih simulatora, a potom i u strogo kontroliranim stvarnim uvjetima, kako bi mogla dobiti dopuštenje za prometovanje stvarnim prometnicama (Wu et al.,2023).

Sustav je potrebno naučiti da koristi podatke sa senzora. To se radi tako da se svakom od sustava unutar vozila (ovisno o njihovoj namjeni), predaje ogromna količina primjerenih simuliranih i stvarnih sadržaja u obliku zvukova, slika i video sadržaja kako bi kroz proučavanje velikih količina tih podataka dosegli najveću moguću točnost prepoznavanja željenih prometnih stavki u stvarnim uvjetima.

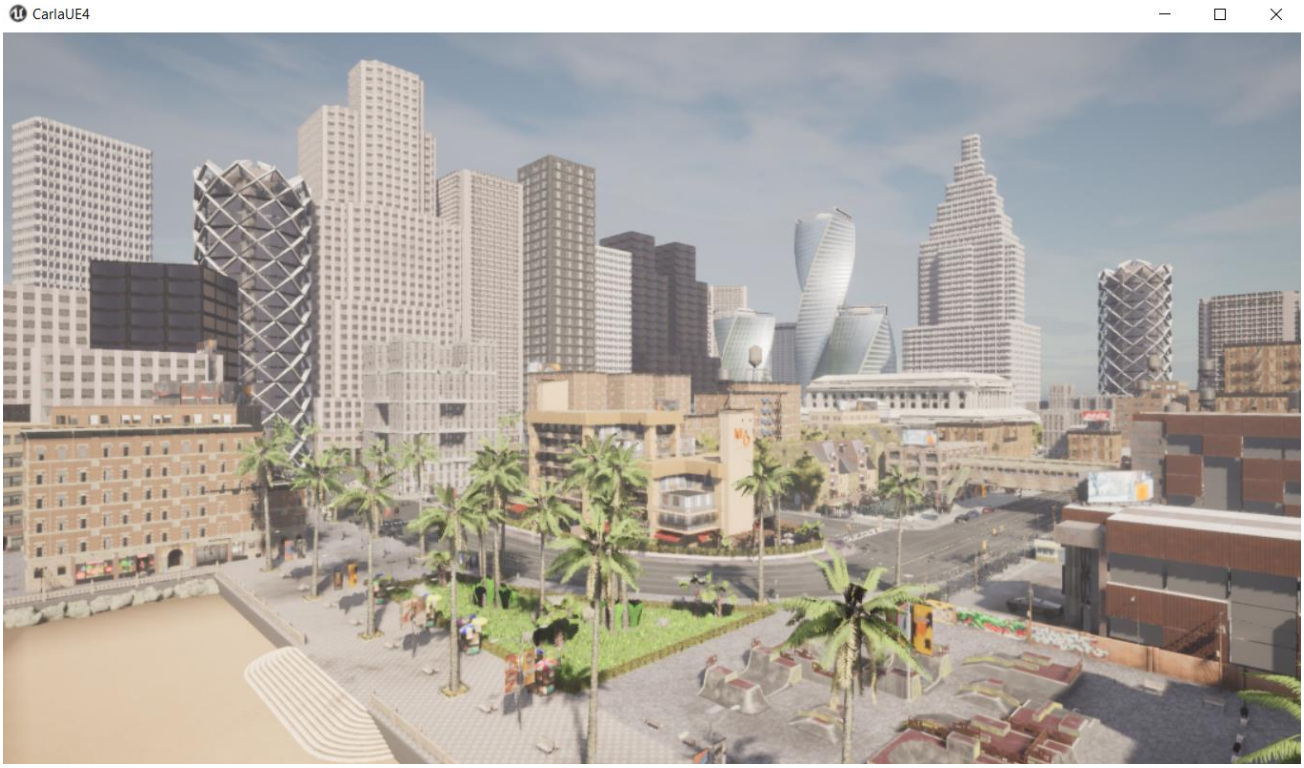
Kada je svaki od sustava unutar autonomnog vozila istreniran do najviših mogućih razina, vozilo se može početi ispitivati na stvarnim prometnicama, te ako tada opravda očekivanja - može ga se pustiti u promet.

2.2. CARLA simulator

CARLA simulator razvijen je od strane *„Computer Vision Centera“* sveučilišta u Barceloni uz pomoć i sponzorstva brojnih svjetskih kompanija kao što su Intel, Toyota, Nvidia i mnoge druge [32].

Simulator je razvijen unutar *„Unreal Engine 4“* razvojnog okruženja koje je u vlasništvu kompanije *Epic Games. Unreal Engine* je razvojno okruženje namijenjeno prvenstveno razvoju video igara, ali s obzirom na to da i sam simulator na trenutke podsjeća na video igru, isti je bio prikladan za razvoj CARLE. Ipak, CARLA simulator nije video igra i kao takvu ga ne treba promatrati, već je to sofisticirani simulator namijenjen treniranju raznih sustava unutar autonomnih vozila.

CARLA se pokreće izravno iz operacijskog sustava (*Windows, Linux ili macOS*) i potom se putem prilagođenih *Python* naredbi upravlja svim događanjima unutar nje. Prikaz početnog sučelja CARLA simulatora nakon pokretanja prikazan je na slici ispod (Slika 2).



Slika 2 - Prikaz početnog sučelja nakon pokretanja CARLA simulatora

Kao što je prikazano na slici, nakon pokretanja simulatora otvara se prazan grad *(„Town01“)* kojim se pomoću tipkovnice možemo slobodno kretati i promatrati njegove ulice, zgrade i okolinu sa željene visine.

Dodatno, preko terminala ili povezanog alata za uređivanje koda, može se putem prilagođenih *Python* naredbi „stvoriti“ željeni broj vozila i pješaka (tzv. *Actora*) koji će se slobodno kretati gradom poštujući pritom sve prometne propise kao što su semafori, prometni znakovi, pješački prijelazi i slično. Drugim riječima, moguće je stvoriti realističnu simulaciju odvijanja prometa na gradskim ulicama.

Ključni dio CARLA simulatora je mogućnost postavljanja vlastitog vozila na prometnicu u svrhu prikupljanja podataka iz okoline. Postavljenim vozilom možemo sami upravljati (kao

da igramo video igru) ili ga možemo postaviti u autonomni mod gdje će se ono samo kretati po prometnici prateći prometne propise, te obraćajući pažnju na ostala vozila i pješake (Slika 3).

Slika 3 - Prikaz autonomnog vozila koje se kreće kroz grad

Vozilu se prilikom vožnje u autonomnom načinu rada na gornjoj slici vidi samo prednji kraj, to je iz razloga jer je na prednji dio automobila postavljena kamera koja nam uzima slike onoga što vozilo vidi ispred sebe, a ovo je samo jedan od načina pogleda iz vozila u svrhu prikupljanja slika.

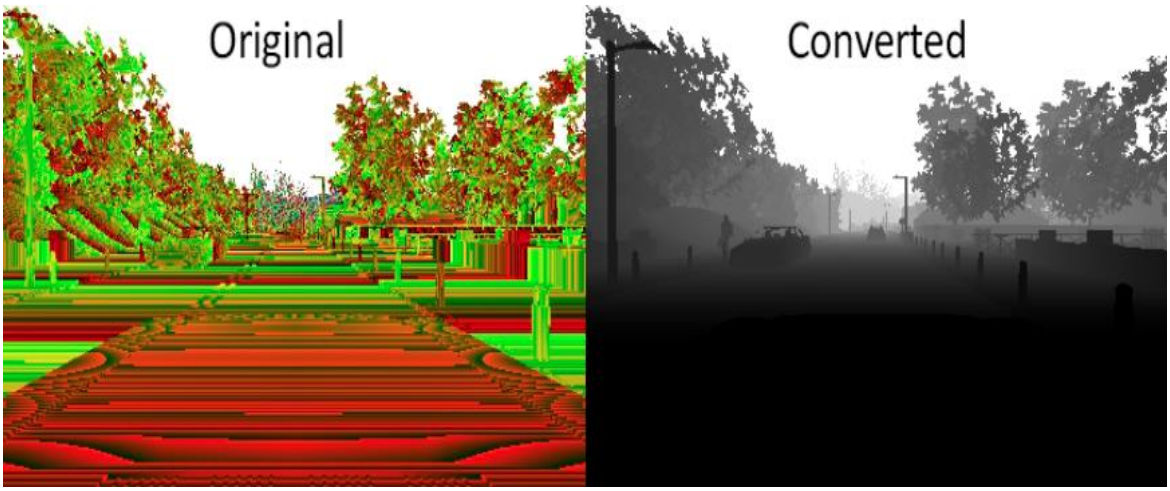
Unutar CARLE postoji više tipova kamera koji se mogu pozicionirati na vozilo u svrhu prikupljanja različitih vrsta slika ili videa u svrhu treniranja različitih tipova senzora autonomnih vozila.

2.3. Vrste kamera u CARLA simulatoru

Unutar CARLE postoji više različitih tipova senzora kao što su: kamere, detektori kolizije, detektori prelazaka preko linija, detektori zapreka, geolokatori, radari i mnogi drugi.

Najbitnije i najkorištenije su svakako kamere, a unutar CARLE one se dijele na šest tipova:

1. ***Depth camera*** prikazuje „dubinu“, tj. udaljenost elemenata u polju kroz različite nijanse sivih tonova koji su dobiveni pretvorbom (normalizacijom) RGB slike (Slika 4).



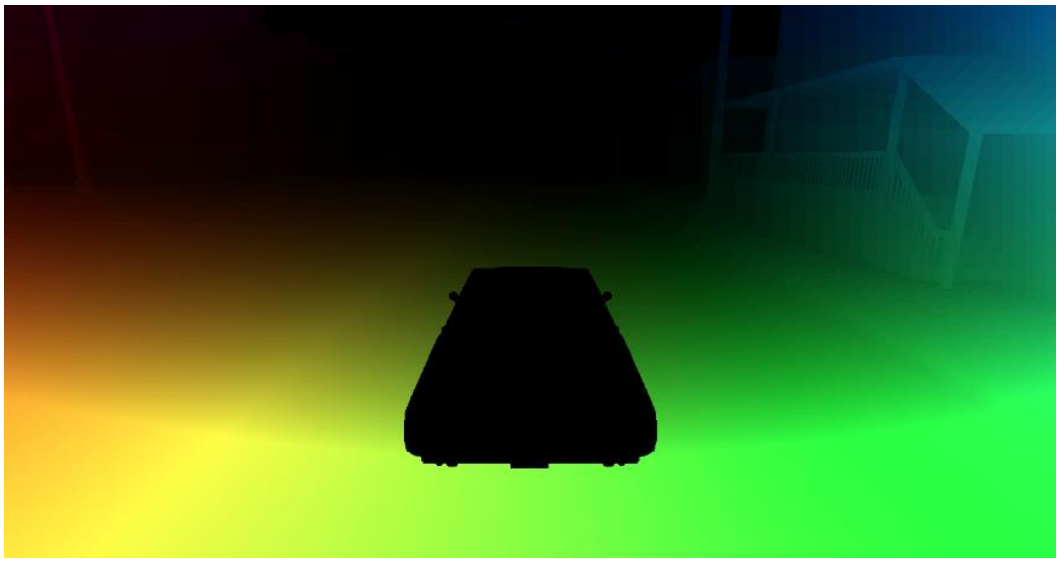
Slika 4 - Prikaz načina rada Depth kamere u CARLA simulatoru, Izvor: https://carla.readthedocs.io/en/latest/ref_sensors/#depth-camera

Ova kamera bitna je i za treniranje modela unutar ovog rada jer su se uz pomoć nje doznale udaljenosti objekata u vidnom polju kamere, što je bio ključni podatak za treniranje modela mreže gdje se ispitivala uspješnost detekcije objekata u ovisnosti o njihovoj udaljenosti.

2. ***RGB camera*** prikazuje sliku u „normalnom“ obliku, tj. kao da je uslikana sa klasičnim fotoaparatom ili kamerom na pametnom telefonu (vidjeti - Slika 3).

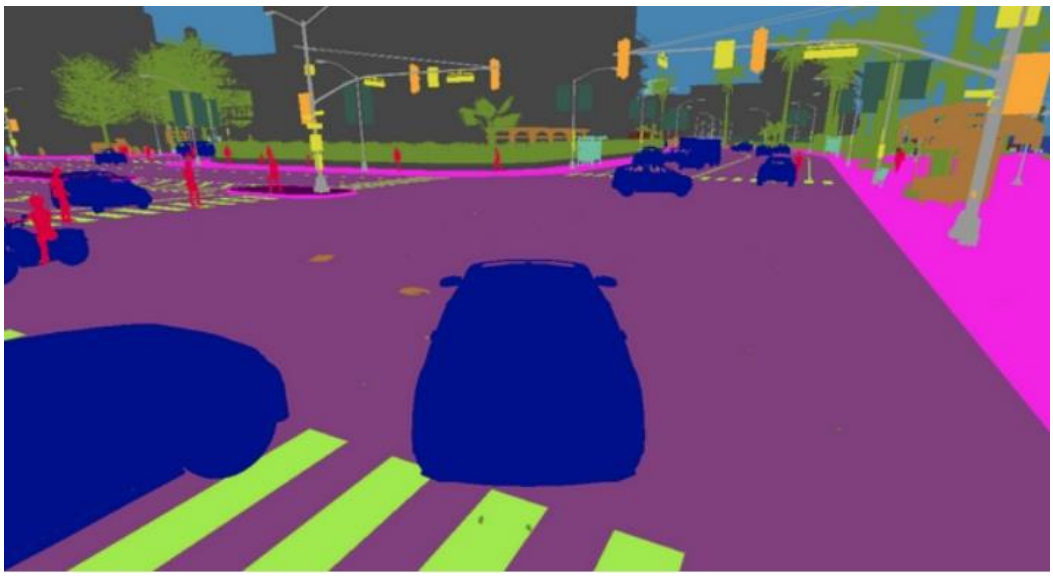
   To je ujedno i tip kamere koji se koristio za potrebe prikupljanja podataka za treniranje modela u sklopu ovoga rada.

3. ***Optical Flow camera*** prikazuje kretanje i intenzitet tog kretanja u vidnom polju kamere (Slika 5).



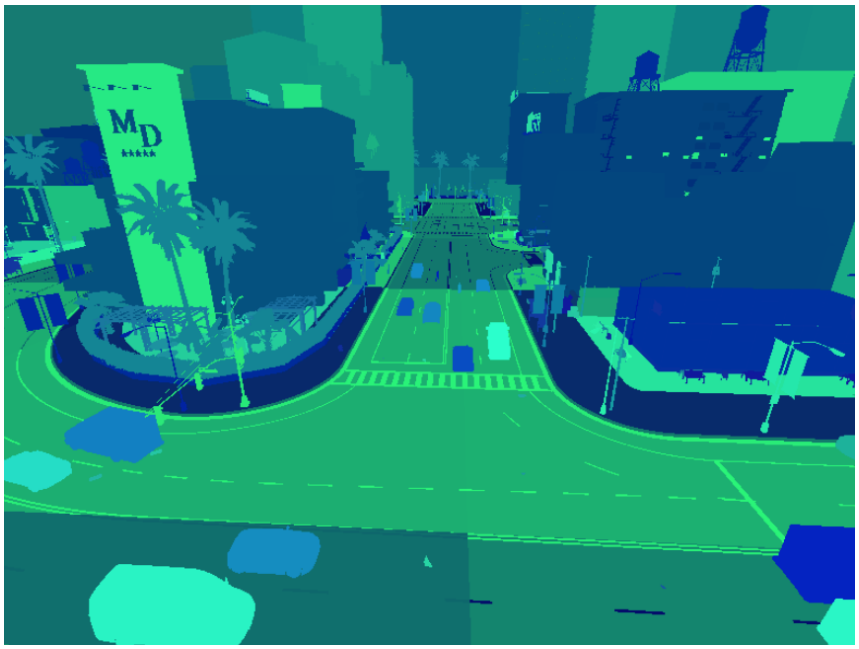
Slika 5 - Prikaz načina rada Optical Flow kamere u CARLA simulatoru, Izvor: https://carla.readthedocs.io/en/latest/ref_sensors/#optical-flow-camera

4. ***Semantic segmentation camera*** prikazuje i odvaja svaki objekt u vidnom polju kamere po jedinstvenim bojama kako bi se razlikovali različiti objekti (Slika 6).



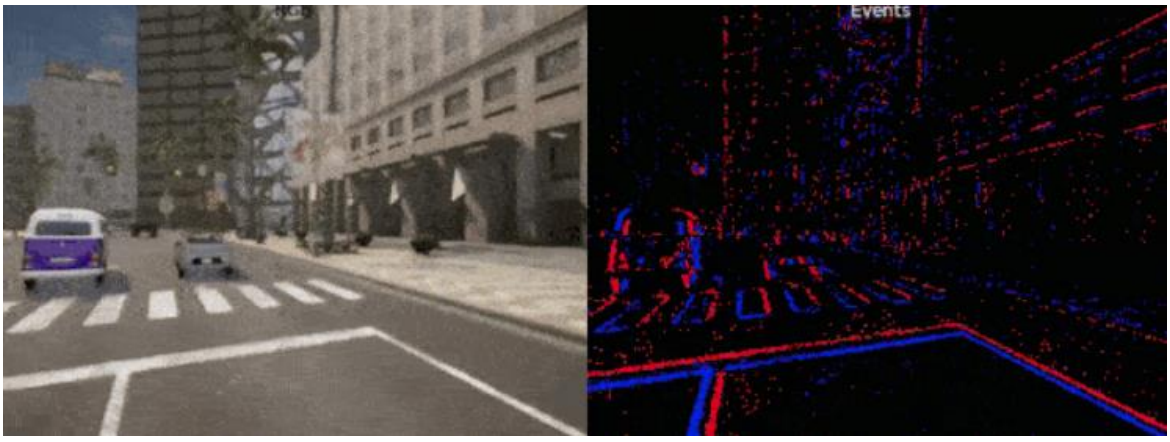
Slika 6 - Prikaz načina rada kamere semantičke segmentacije u CARLA simulatoru, Izvor: https://carla.readthedocs.io/en/latest/ref_sensors/#semantic-segmentation-camera

5. ***Instance segmentation camera*** klasificira svaki objekt u vidnom polju prema njegovoj klasi i ID-u tog specifičnog objekta (Slika 7).



Slika 7 - Prikaz načina rada kamere instance segmentacije u CARLA simulatoru, Izvor: https://carla.readthedocs.io/en/latest/ref_sensors/#instance-segmentation-camera

6. ***Dynamic Vision Sensor (DVS) camera*** je tzv. „kamera događaja“, ona ne uzima slike u fiksno postavljenom vremenu, već bilježi i snima samo kada se dogode veće promjene u vidnom polju kamere (Slika 8).



Slika 8 - Prikaz načina rada DVS kamere u CARLA simulatoru, na lijevoj slici prikazano je ono što se događa na sceni, a na desnoj slici prikazan je način na koji to DVS kamera percipira, Izvor: https://carla.readthedocs.io/en/latest/ref_sensors/#dvs-camera

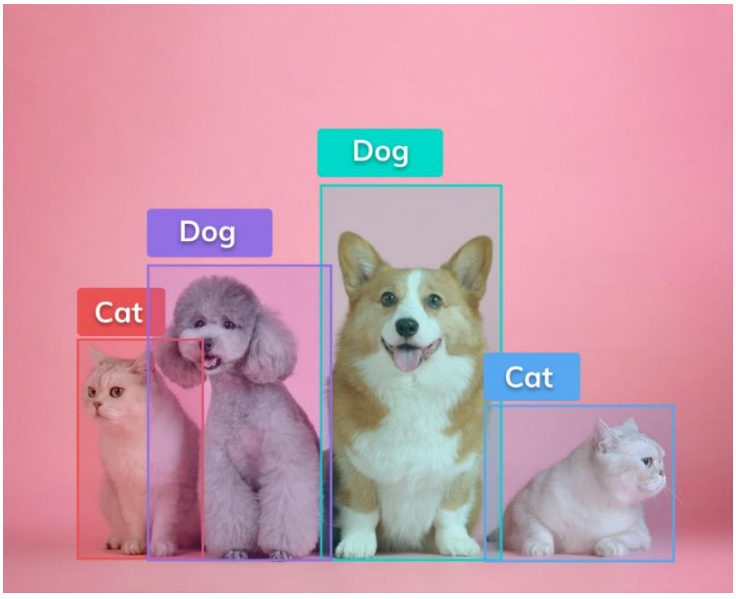
3. YOLO algoritam

3.1. Način rada i primjene YOLO algoritma

Prva verzija *„You Only Look Once“* (YOLO) algoritma razvijena je od strane Josepha Redmona i Ali Farhadija na Sveučilištu u Washingtonu 2015. godine (Redmon, et al., 2015). Sa svojim izumom YOLO algoritma napravili su revoluciju u polju strojnog učenja pošto je YOLO bio prvi *one-stage* algoritam koji se mogao mjeriti sa dotadašnjim *two-stage* algoritmima kao što su *Fast R-CNN* ili *Faster R-CNN*.

Iako YOLO nije bio prvi ikad kreirani *one-stage* algoritam, može se reći da je bio prvi koji je svoje zadatke odrađivao sa zadovoljavajućim omjerom točnosti i brzine, pošto su se dotadašnji *one-stage* algoritmi mučili upravo s balansiranjem ta dva parametra.

Cilj algoritama za detekciju objekata je pronalaženje objekata na slici ili videu i iscrtavanje tzv. *„bounding boxeva“* oko istih s ciljem prepoznavanja o kojoj je točno klasi objekta riječ (Slika 9). Klasa je ustvari ono što sam objekt predstavlja, drugim riječima – ako je na slici mačka, ispravan naziv klase tog objekta je „mačka“.



Slika 9 - Bazični prikaz načina rada algoritma za detekciju objekata,

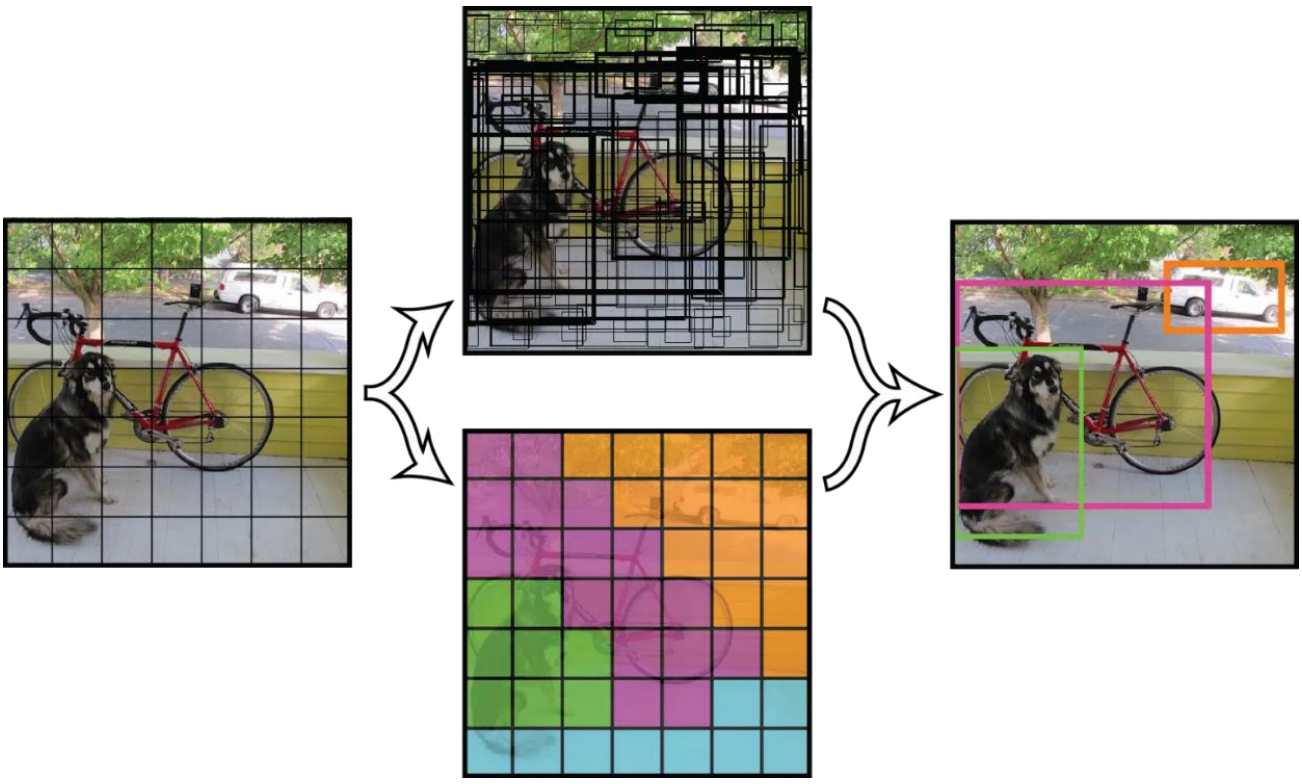
Izvor: https://www.v7labs.com/blog/object-detection-guide

YOLO algoritam znatno je poboljšao kvalitetu i brzinu detekcije objekata u stvarnom vremenu pošto je kao *one-stage* algoritam bio znatno brži od dotadašnjih *two-stage* algoritama. Iz tog razloga je YOLO algoritam u većini slučajeva bolji odabir pri treniranju modela gdje je bitna brza detekcija objekata kao što su sigurnosni sustavi ili autonomna vozila.

3.2. Razlike između načina rada one-stage i two-stage algoritama

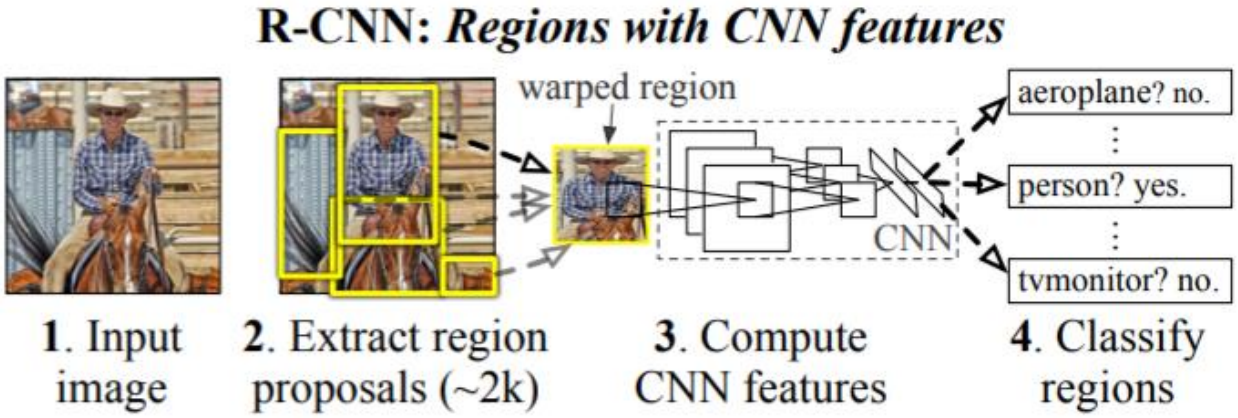
Glavna razlika između one-stage i two-stage algoritama je u načinu na koji pristupaju detekciji i klasifikaciji objekata na slici.

One-stage algoritmi funkcioniraju tako da se u jednom prolasku kroz neuronsku mrežu modela istovremeno predviđaju *bounding boxevi* i klase objekata na slici (Slika 10).



Slika 10 - Način na koji YOLO algoritam procesira sliku u svrhu iscrtavanja bounding boxeva, Izvor: https://www.v7labs.com/blog/object-detection-guide

Two-stage algoritmi istu radnju izvršavaju u dva koraka. U prvom koraku predviđaju „kandidate“ tj. područja na slici podobna za potencijalno iscrtavanje *bounding boxeva*, a potom se u drugom koraku svaki od kandidata detaljnije analizira kako bi se utvrdilo je li to zapravo objekt i koja je njegova klasa (Slika 11).



Slika 11 - Način na koji R-CNN algoritam procesira sliku u svrhu iscrtavanja bounding boxeva, Izvor: https://medium.com/codex/a-guide-to-two-stage-object-detection-r-cnn-fpn-mask-r-cnn-and-more-54c2e168438c

Iz prikazanih slika lako se da zaključiti kako je YOLO u većini slučajeva brži algoritam od bilo kojeg *two-stage* algoritma, ali to nikako ne umanjuje vrijednost *two-stage* algoritama s obzirom na to da su njihova predviđanja klasa i iscrtavanja *bounding boxeva* oko objekata često točnija i preciznija od YOLO algoritma i ostalih *one-stage* algoritama. Naime, svaki tip algoritma ima svoje ciljano područje primjene.

3.3. Povijesni pregled YOLO algoritama

YOLO je zamišljen kao *„open-source“* alat što znači da svatko može preuzeti neku od verzija algoritma i dodatno je modificirati prema vlastitim potrebama, te takvu modificiranu verziju ponovo podijeliti s ostatkom korisnika na slobodno preuzimanje i korištenje. Iz tog razloga danas postoji veliki broj različitih korisničkih verzija YOLO algoritma.

Usprkos velikom broju korisničkih modifikacija, YOLO algoritam se može podijeliti na osam osnovnih verzija (generacija) koje su donijele značajnije napretke i promjene u radu samog algoritma (Jocher et. al., 2023). Kratke značajke svake od generacija prikazane su u nastavku [31] :

- **YOLO** je prvi puta predstavljen 2015. godine, te je brzo stekao popularnost zbog svoje brzine i preciznosti

- **YOLOv2** druga je generacija YOLO algoritma predstavljena 2016. godine, dodatno je poboljšala performanse algoritma uvođenjem batch normalizacije, *anchor boxeva* i dimenzionalnih *clustera*

- **YOLOv3** treća je generacija YOLO algoritma predstavljena 2018. godine, poboljšala je performanse algoritma manjim promjenama u samoj arhitekturi algoritma

- **YOLOv4** četvrta je generacija YOLO algoritma predstavljena 2020. godine, uvela je brojne inovacije u sam algoritam kao što su mozaički *data augmentation* i novi tipovi funkcija gubitka

- **YOLOv5** donio je dodatna poboljšanja u performansama samog algoritma, a ujedno je uveo mnoge novitete za bolja praćenja treniranih modela te tumačenja završnih rezultata u vidu integriranog praćenja treninga i automatskog izvoza podataka (rezultata) u veliki broj različitih formata za korištenje u drugim aplikacijama ili uređajima

- **YOLOv6** razvijen je od strane kineske dostavne kompanije „Meituan“ 2022. godine prvenstveno u svrhu treniranja njihovih autonomnih dostavnih robota i vozila

- **YOLOv7** donio je novitete u vidu točnijih procjena položaja i orijentacije objekata

- **YOLOv8** najnovija je verzija YOLO algoritma razvijena od strane *Ultralyticsa*, uvela je brojna poboljšanja u performansama, fleksibilnosti i stabilnosti samog algoritma, šireći tako raspon primjenjivosti YOLO algoritma

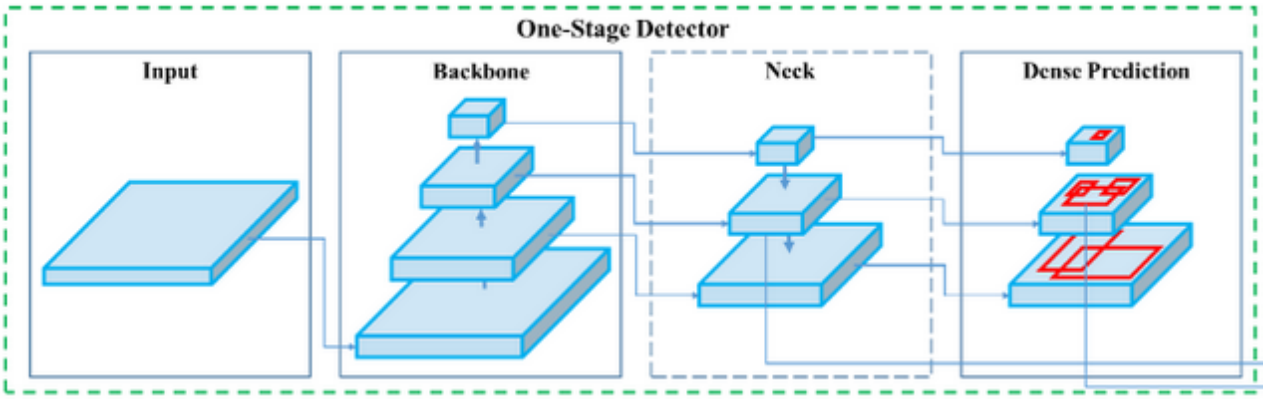
Za potrebe treniranja modela u nastavku korištena je YOLOv5 verzija algoritma iz razloga što su u trenutku odlučivanja o algoritmu koji će se koristiti YOLOv7 i YOLOv8 bile relativno nove i neprovjerene verzije algoritma, a YOLOv6 algoritam se koristi prvenstveno u industrijske svrhe.

3.4. Arhitektura YOLOv5 algoritma

YOLOv5 razvijen je od strane istog tima ljudi koji su razvili i originalni YOLO algoritam, a održava se od strane *Ultralytics* platforme koja je kao takva glavni izvor dokumentacije i informacija o samom algoritmu.

Za razliku od prošlih verzija YOLO algoritma, YOLOv5 prva je verzija algoritma koja je softverski zasnovana na *PyTorch frameworku* [29] koji je pisan u *Python* programskom jeziku. Osim što mu je to poboljšalo performanse pri izvođenju, YOLOv5 je time postao i znatno pristupačniji i lakši za korištenje od dotadašnjih verzija YOLO algoritma koje su radile na *Darknet frameworku* pisanom u C programskom jeziku (Keita, 2022).

Arhitektura neuronskih mreža većine *one-stage* algoritama, pa tako i YOLO algoritma, sastoji se od tri dijela kroz koja prolaze svi ulazni podaci. Ti dijelovi nazivaju se slijedno: kičma (*eng. backbone*), vrat (*eng. neck*) i glava (*eng. head*) (Slika 12).



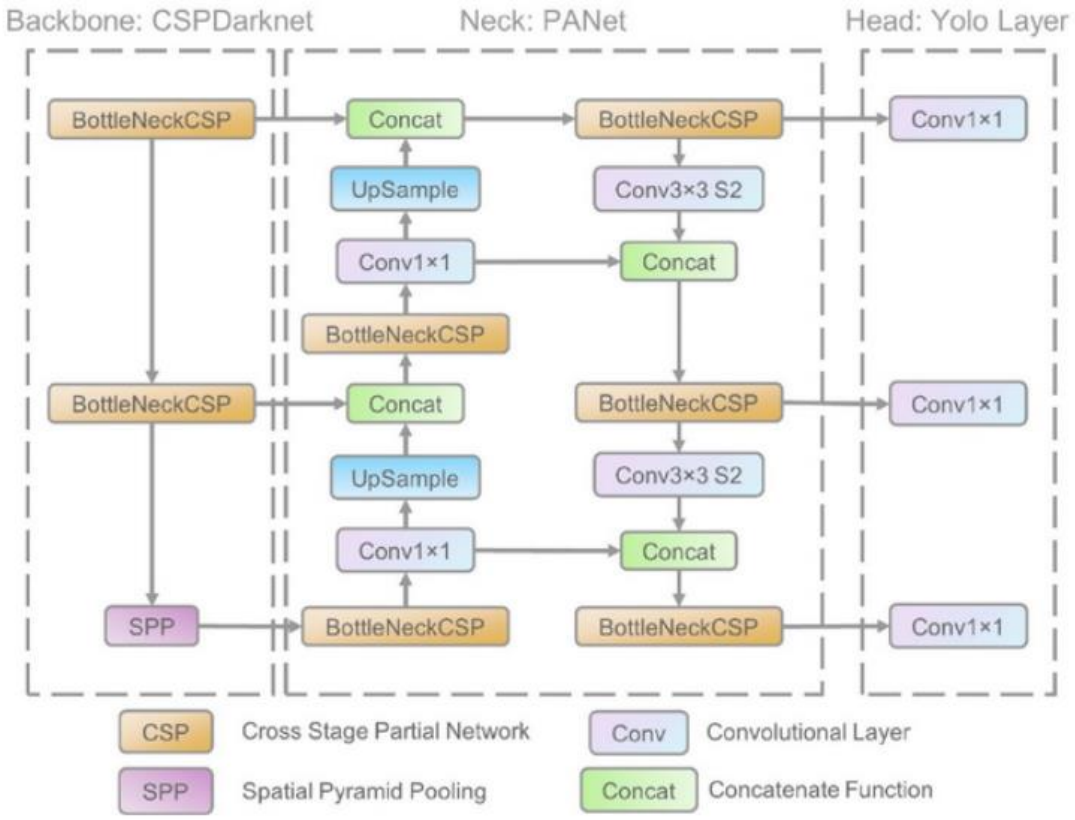
Slika 12 - Prikaz arhitekture neuronskih mreža one-stage algoritama, Izvor: https://iq.opengenus.org/yolov5/

Kičma je zadužena za izvlačenje ključnih značajki iz ulaznih podataka na način da kreira mapu značajki (*eng. Feature map*) koja bilježi najbitnije informacije o objektima i njihovom kontekstu unutar slike.

Vrat sa svojim dodatnim konvolucijskim slojevima dodatno produbljuje znanje o prethodno definiranim značajkama što pomaže kvaliteti i brzini detekcije samog algoritma.

Glava je završni dio modela neuronske mreže *one-stage* algoritma. Ona je odgovorna za predviđanje lokacija objekata oko kojih se trebaju iscrtati *bounding boxevi*, kao i za predviđanje klasa tih objekata.

YOLO algoritam građen je na istoj takvoj arhitekturi. Arhitektura YOLOv5 mreže prikazana je na slici ispod (Slika 13).



Slika 13 - Prikaz sastavnica neuronske mreže YOLOv5 algoritma, Izvor: https://iq.opengenus.org/yolov5/

YOLOv5 koristi CSP-Darknet53 kao kičmu svoje neuronske mreže. Isti je korišten još kod YOLOv3 verzije, ali ovdje je primijenjena *„Cross Stage Partial“* (CSP) strategija mreža koja koristi *residua*l i *dense* blokove kako bi prevladala problem nestajućih i redundantnih gradijenata. Navedeno pomaže pri smanjivanju broja parametara unutar mreže što znatno smanjuje i broj izračuna pri prolasku podataka kroz mrežu. Na taj način se znatno smanjuje i brzina zaključivanja, koja je najvažniji parametar pri detekciji objekata u stvarnom vremenu.

Najveće promjene u strukturi YOLO algoritma sa YOLOv5 generacijom dogodile su se u vratu same mreže. Prva bitna promjena je kvalitetnije korištenje *Spatial Pyramid Poolinga* (SPP) koje započinje još u kičmi, a druga promjena je korištenje *Path Aggregation Networka* (PANet). Iako su oba ova pristupa korištena i u ranijim generacijama YOLO mreže, ovdje su nad njima napravljene značajne nadogradnje koje su pospješile lakšu agregaciju i protok informacija kroz mrežu, što je uzročno-posljedično poboljšalo i brzinu same mreže.

Glava mreže YOLOv5 modela ostala je ista kao i u YOLOv3 i YOLOv4 modelima, odnosno i dalje koristi tri konvolucijska sloja koja predviđaju lokacije bounding boxeva na slici i klase kojima objekti unutar tih bounding boxeva pripadaju.

3.5. Veličine mreža unutar YOLOv5 algoritma

YOLOv5 algoritam izgrađen je na prethodno prikazanoj arhitekturi, ipak unutar samog algoritma imamo različite veličine njegovih prethodno treniranih modela mreža. Sve su one izgrađene na istom principu, ali se razlikuju u svojoj dubini tj. broju parametara koje posjeduju. Što je mreža „dublja“, odnosno veća, to bi rezultati treninga trebali biti bolji, ali korištenjem dubljih mreža znatno se produljuje i vrijeme samog treniranja. Osim duljeg treninga, korištenje dubljih mreža zahtijeva i veću hardversku moć stroja (računala) na kojem se trening odvija, pošto brzina i mogućnost treniranja direktno ovisi i o hardveru samog računala.

YOLOv5 posjeduje pet osnovnih veličina mreža koje se koriste prilikom treniranja korisničkih modela. Iste su prikazane u nastavku:

- **n (nano) model** mreže koristi oko 1.9 milijuna parametara
- **s (small) model** mreže koristi oko 7.2 milijuna parametara
- **m (medium) model** mreže koristi oko 21.2 milijuna parametara
- **l (large) model** koristi oko 46.5 milijuna parametara
- **x (extra large) model** koristi oko 86.7 milijuna parametara

Postoji još nekolicina podvrsta veličina mreža tj. proširenih verzija svake od već navedenih veličina, ali to su tek blage varijacije navedenih modela.

3.6. Hiperparametri i metrike YOLOv5 algoritma

YOLOv5 trenira „adaptivno“, odnosno kroz epohe treniranja algoritam automatski prilagođava svoje parametre kako bi postigao što bolje rezultate sa svakom sljedećom epohom.

Ono što se ne mijenja tijekom treninga je njegov optimizator i funkcija gubitka. YOLOv5 koristi „*Stohastic Gradient Descent“* (SGD) kao uobičajeni optimizator, a kao uobičajenu funkciju gubitka koristi *„Generalised Intersection over Union“* (GioU).

Svi hiperparametri, pa tako i optimizator i funkcija gubitka, mogu se podesiti i promijeniti ručno unutar konfiguracijske datoteke prije početka željenog treninga.

Prilikom treniranja YOLOv5 prati više različitih kategorija metrika koje po završetku treninga i validacije daje kao rezultate u vidu brojčanih podataka i prikladnih grafova.

Metrike koje YOLOv5 prati tijekom treninga su sljedeće:

1) Funkcije gubitka tijekom treninga:

    - ***Train/box_loss*** mjeri koliko dobro model predviđa lokacije (koordinate) bounding boxeva oko objekata prilikom treninga. Što je *box_loss* manji to su koordinate točnije.

    - ***Train/obj_loss*** mjeri koliko dobro model predviđa je li objekt prisutan unutar iscrtanog bounding boxa prilikom treninga. Što je *obj_loss* manji to je veća vjerojatnost da je traženi objekt prisutan unutar *bounding boxa*.

    - ***Train/cls_loss*** mjeri koliko točno model predviđa ispravne klase objekata unutar *bounding boxeva* prilikom treninga. Što je *cls_loss* manji to je veća šansa da je predviđena klasa objekta ispravna.

Navedeni gubitci samo su komponente funkcije gubitka, odnosno na njih se referira kao na „gubitke“, a ne kao na cjelokupne funkcije gubitka.

2) Evaluacijske metrike:

- ***Metrics/precision*** odgovara na pitanje „Koliko smo sigurni da su objekti koje je model označio *bounding boxevima* točno klasificirani ?“. Naime, ako je model nešto prepoznao kao vozilo, to isto ne mora stvarno i biti vozilo, a *precision* u tom slučaju vraća podatak koliki je udio točnih prepoznavanja klase vozila model imao u ukupnom broju prepoznavanja. Drugim riječima, *precision* metrika mjeri omjer stvarnih (istinitih) detekcija među svim detekcijama (istinitim i lažnim). Izražava se jednadžbom:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Jednadžba 1 - Precision (Izvor: Fränti i Istodor, 2023)

- ***Metrics/recall*** mjeri „osjetljivost“ modela prilikom detekcije, odnosno govori koliki je udio objekata svih klasa model točno detektirao, bez obzira na to je li pritom klasa objekta točno predviđena. Izražava se jednadžbom:

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Jednadžba 2 - Recall (Izvor: Fränti i Istodor, 2023)

- ***Metrics/mAP_0.5*** je *„Mean Average Precision“* metrika koja računa prosječnu vrijednost uspješne detekcije i klasifikacije svih klasa objekata uz *Intersection over Union (IoU)* postavljen na 0.5. Drugim riječima - ova metrika govori koliko je objekata točno prepoznato uz definiran uvjet da se predviđanje od strane modela (iscrtan *bounding box*) preklapa s bar 50% površine stvarnog objekta na slici. Ako npr. *mAP_0.5* iznosi 0.87, to znači da je 87% objekata zadovoljavalo uvjet da se predikcija od strane modela preklapa s barem 50% površine stvarnog objekta na slici, dok preostalih

13% detekcija objekata nije zadovoljilo taj uvjet, odnosno bili su lažno pozitivni (neispravno klasificirani) ili nisu zadovoljavali uvjet preklapanja. Izražava se formulom:

$$mAP = \frac{1}{n}\sum_{k=1}^{k=n} AP_k$$

Jednadžba 3 - mAP (Izvor: Padilla, et. el., 2020)

Pri čemu **n** označava broj klasa, a $\mathbf{AP_k}$ je prosječna vrijednost klase k

- ***metrics/mAP_0.5:0.95*** ima istu ulogu kao i mAP_0.5, ali uz strože definiran IoU koji varira u rasponu od 0.5 do 0.95.

3) Funkcije gubitka tijekom privremene validacije:

- ***Val/box_loss*** ima istu funkciju kao i *train/box_loss*, samo što se ovdje mjeri na dotad neviđenim podacima tijekom procesa privremene validacije.
- ***Val/obj_loss*** ima istu funkciju kao i *train/obj_loss*, samo na validacijskim podacima korištenima prilikom privremene validacije.
- ***Val/cls_loss*** ima istu funkciju kao i *train/cls_loss*, samo na validacijskim podacima korištenim prilikom privremene validacije.

Bitno je naglasiti da se „prava“ validacija u svom punom obujmu odvija nakon završetka svih trening epoha, ali u svrhu što boljeg praćenja treninga i utjecaja aktualnih hiperparametara, po završetku svake trening epohe odvija se tzv. „privremena validacija“ koja na malom nasumičnom uzorku slika iz validacijskog skupa podataka govori koliko dobro treniranje modela napreduje i kakvi bi rezultati mogli biti prilikom završne validacije.

4) Metrike stope učenja (*eng. learning rate metrics*):

- ***x/lr0*** je vrijednost hiperparametra *learning ratea* na početku i u početnim fazama treniranja.

- ***x/lr1*** je vrijednost hiperparametra *learning ratea* usred procesa treninga.
- ***x/lr2*** je završni *learning rate* pri kraju treniranja ili u posljednjoj epohi treninga.

Vrijednosti learning ratea *lr0*, *lr1* i *lr2* se pomoću definiranog optimizatora (*SGD*) nakon određenog vremena u procesu treninga izjednače i imaju iste vrijednosti sve do kraja treninga, stoga će se u kasnijim prikazima i komentarima dobivenih rezultata ove tri metrike objediniti i prikazati kao jedinstvena vrijednost ***lrX*** zbog ljepšeg i jednostavnijeg prikaza.

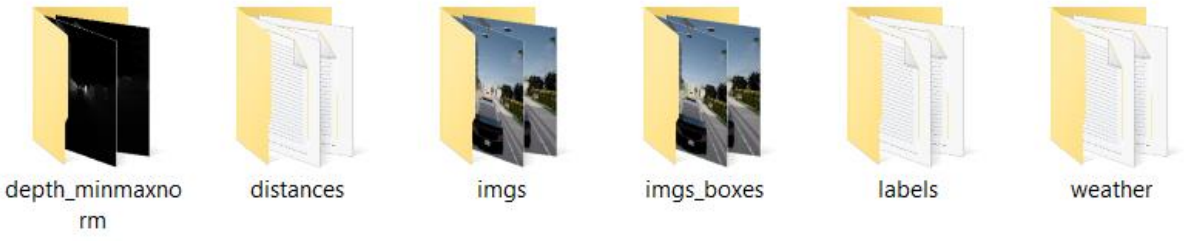
4. Prikupljanje i obrada podataka

4.1. Prikupljanje slika pomoću CARLA simulatora

U svrhu prikupljanja slika iz CARLA simulatora korištena je modificirana verzija CARLA simulatora koja sadrži već integrirane, prilagođene *Python* skripte, u svrhu prikupljanja podataka uz pomoć *RGB i depth* senzora iz CARLA simulatora. Iako modificirana verzija simulatora ne radi na najnovijoj verziji CARLE (0.9.14), već na nešto starijoj 0.9.7 verziji, to u slučaju prikupljanja podataka za potrebe naših modela nije imalo pretjerane važnosti ni utjecaja na kvalitetu prikupljenih podataka, a pritom je korištenje modificirane verzije znatno olakšalo i skratilo proces prikupljanja podataka s obzirom na to da bi se u suprotnom sve skripte za prikupljanje podataka unutar aktualne verzije CARLE trebale pisati ručno i prilagođavati specifičnim potrebama svakog skupa podataka [28].

Na ovaj način podaci su jednostavno prikupljeni preuzimanjem modificirane verzije s *GitHub* repozitorija i pokretanjem iste, koja potom većinu posla odrađuje automatski pritom spremajući podatke (slike) sve dok je korisnik sam ne prekine u toj radnji.

Ova verzija simulatora paralelno prikuplja slike snimljene *RGB* i *depth* tipovima kamere, te ih potom sprema na željenu lokaciju unutar računala pritom ih odvajajući u zasebne direktorije (Slika 14).



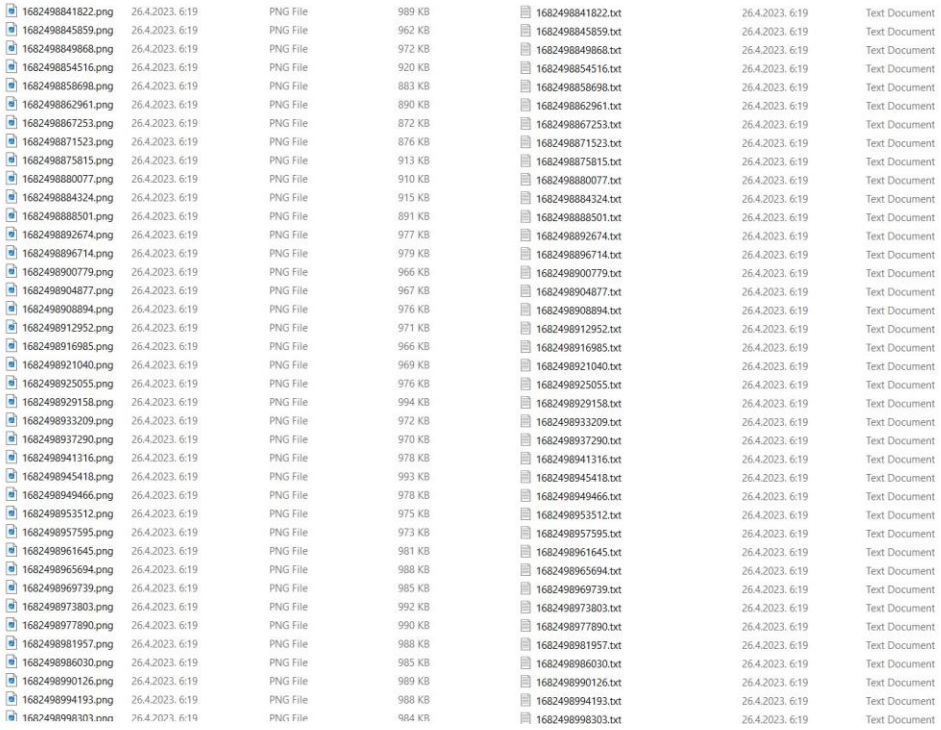
Slika 14 - Prikaz izgleda datoteke sa prikupljenim podacima putem modificiranog simulatora

Ono što znatno olakšava daljnji posao prilagodbe podataka je to što su sve datoteke u kreiranim direktorijima istoga imena, odnosno dodjeljuje im se ista vremenska oznaka kao naziv datoteke. Drugim riječima, unutar direktorija *"depth_minmaxnorm", "imgs"* i *"imgs_boxes"* postoji identična slika u *.png* formatu, slikana u istom trenutku i putem *RGB*

i putem *depth* kamere, koja je pritom istog naziva unutar sva tri direktorija, ali sukladno njenoj namjeni odvojena je i paralelno kreirana u tri različita direktorija zbog kasnije lakše primjene.

Direktoriji naziva *"distances"*, *"labels"* i *"weather"* slijedno sadrže podatke o udaljenosti pojedinih objekata od kamere (vozila i pješaka), anotacijama objekata tj. ispravnim lokacijama *bounding boxeva* objekata na slici, te informacije o vremenskim prilikama u trenutku prikazanom na slici. Datoteke u ova tri direktorija također su istog naziva kao i slikovne datoteke unutar tri prvonavedena direktorija, samo što su ovdje u pitanju tekstualni podaci, pa shodno tome ovi direktoriji ne sadrže slike, već datoteke u *.txt* formatu.

Na sljedećoj slici je paralelni prikaz početnih dijelova direktorija *"imgs"* i *"labels"* na kojem se vidi kako sve stavke unutar direktorija imaju isti naziv uz prikladan format (Slika 15).



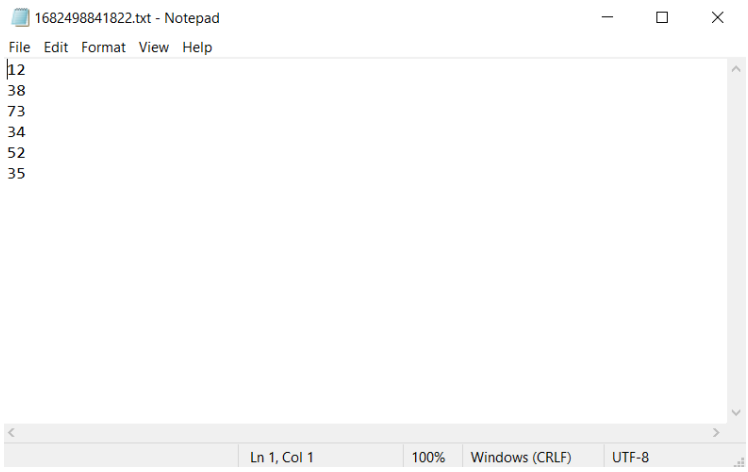
Slika 15 - Paralelni prikaz početnih dijelova direktorija „imgs" (lijevo) i „labels" (desno)

Detaljnija objašnjenja svakog pojedinog direktorija su sljedeća:

- ***depth_min_maxnorm*** direktorij sadrži slikovne *.png* datoteke koje su slikane sa *depth kamerom* (Slika 16)

Slika 16 - Prikaz slike unutar direktorija slikane sa depth kamerom

- ***distances*** direktorij sadrži *.txt* datoteke gdje svaki pojedini red unutar datoteke označava jedan određeni objekt (vozilo ili pješaka) sa iznosom njegove simulirane udaljenosti od kamere izražene u metrima (Slika 17)



Slika 17 - Prikaz jedne distance.txt datoteke gdje svaka linija prikazuje udaljenost jednog od objekata na slici

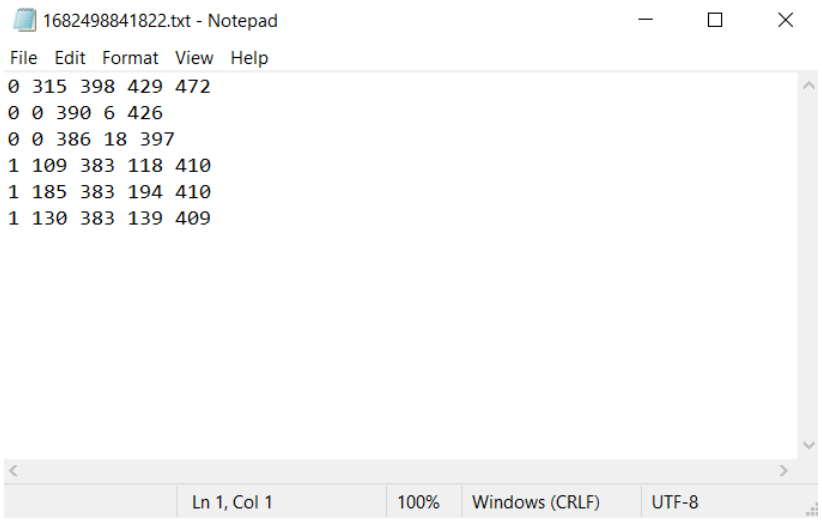
- ***imgs*** direktorij sadrži slikovne *.png* datoteke koje su slikane *RGB* kamerom (Slika 18)

Slika 18 - Prikaz slike unutar imgs direktorija slikane RGB kamerom

- ***imgs_boxes*** sadrži slikovne *.png* datoteke oko kojih su već iscrtani pravilni *bounding boxevi* oko objekata uz pomoć anotacija iz istoimenih *.txt* datoteka unutar „*labels*“ direktorija (Slika 19)

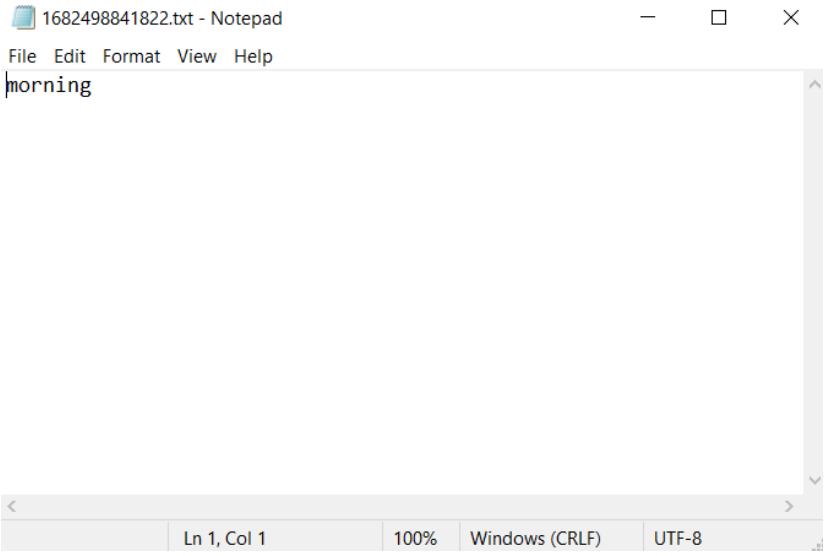
Slika 19 - Prikaz slike sa već iscrtanim bounding boxevima

- ***labels*** direktorij sadrži *.txt* datoteke u koje su zapisane anotacije objekata, odnosno točne koordinate za iscrtavanje *bounding boxeva* traženih objekata na slici, pri čemu prvi broj označava uvijek klasu objekta (Slika 20)



Slika 20 - Prikaz jedne labels.txt datoteke gdje svaka linija označava klasu objekta i prikazuje točne lokacije za iscrtavanje bounding boxa tog pojedinog objekta unutar slike

- ***weather*** direktorij sadrži *.txt* datoteke sa podatkom o dobu dana tj. vremenskim uvjetima na odgovarajućoj slici (Slika 21)



Slika 21 - Prikaz jedne weather.txt datoteke gdje je u jednom retku prikazano doba dana na slici što ujedno kazuje i specifične vremenske uvjete postavljene za to doba dana

Ono što je bitno naglasiti je to da su sadržaji unutar *.txt* datoteka međusobno povezani, odnosno prva linija bilo koje *distance.tx*t datoteke označava udaljenost za isti objekt za kojeg su ispisane *bounding box* koordinate unutar prve linije *labels.txt* datoteke istog imena. To znatno olakšava kasniju prilagodbu i razdjelu originalnog skupa podataka, što je bilo potrebno učiniti za potrebe treniranja modela različitih namjena u nastavku.

4.2. Prilagodba podataka za potrebe modela

Slike su prikupljane iz različitih gradova unutar CARLA simulatora kako bi se postigla što veća raznolikost okoline na slikama i mogućih situacija u prometu. Odabrano je šest različitih gradova za prikupljanje slika, a to su: *Town01, Town02, Town03, Town04, Town05* i *Town10*.

Iz svakog grada prikupljen je približno isti broj slika, a ukupan zbroj prikupljenih slika iz svih gradova iznosi 21450 slika. To je ukupan broj slika, ali pritom nije svaka pojedina slika prikladna za korištenje unutar svakog treniranog modela. Tako su se neke slike koristile možda i u 7 od 8 treniranih modela, a neke su bile iskorištene u svega jednom ili dva modela.

Navedeno je najviše ovisilo o samom sadržaju promatrane slike, odnosno ukoliko je treniran model s jutarnjim vremenskim prilikama - u istom se nisu mogle koristiti slike gdje su očigledni noćni uvjeti vožnje. Isto vrijedi i za modele s udaljenostima objekata, naime ne sadrži svaka slika objekte na istoj udaljenosti, odnosno na nekoj slici mogu biti prikazani samo objekti udaljeni do 10 metara, dok na nekoj drugoj mogu biti prikazani isključivo objekti na udaljenosti većoj od 50 metara. Jasno je da se prvonavedene slike ne mogu koristiti u treniranju modela gdje se ispituje uspješnost detekcije objekata udaljenijih od 50 metara od kamere, kao niti što se drugonavedene ne mogu koristiti za treniranje objekata koji su u neposrednoj blizini kamere, odnosno na udaljenostima manjim od 10 metara.

Iz tih razloga bilo je potrebno dodatno prilagoditi i razdvojiti početni skup podataka na 8 različitih manjih skupova koji zadovoljavaju potrebe svojih treninga. Kako je zbog veličine i složenosti originalnog skupa podataka to nemoguće ručno odraditi, za isto su se koristile ciljano napisane Python skripte koje su automatski odradile sva potrebna kopiranja i

razvrstavanja slika i dokumenata, te je na taj način dobiveno 8 konačnih trening i validacijskih skupova podataka koji su korišteni u kasnijim modelima. Prikaz i kratka objašnjenja rada i namjene svih kreiranih Python skripti prikazani su u poglavlju "DODATAK", dok će rad pojedinih skripti biti kratko objašnjen i u sljedećim poglavljima.

Broj slika unutar trening i validacijskih skupova, kao i ukupan broj slika za svaki od pojedinih modela prikazan je u tablici ispod (Tablica 1).

Tablica 1 - Prikaz broja slika unutar skupova podataka svih kreiranih modela

	Less10	**10to50**	**More50**	**Morning**	**Noon**	**Afternoon**	**Almost Night**	**Night**
Trening	9741	15842	6695	3567	3351	3400	3485	3397
Validacija	2437	3960	1674	850	850	850	850	850
UKUPNO	12178	19802	8369	4417	4201	4250	4335	4247

Kao što je vidljivo iz tablice, modeli za ispitivanje uspješnosti detekcije objekata u ovisnosti o vremenskim uvjetima su veličinama znatno izjednačeniji od modela za ispitivanje uspješnosti detekcije u ovisnosti o udaljenosti objekata od vozila. Razlog tome je što CARLA simulator u podjednakim vremenskim periodima raspoređuje trajanje svakog doba dana unutar simulacije, dok na to koliko će biti objekata na pojedinoj slici i na kojim udaljenostima će se isti nalaziti simulator ne može znatno utjecati, pošto se kreiranje i kretanje dinamičkih objekata u prometu odvija u potpunosti nasumično uz poštivanje prometnih propisa.

S obzirom na to da je cilj rada provjeriti uspješnost detekcije iz originalno prikupljenog skupa podataka sa što realističnijim uvjetima - tri prvonavadena skupa se nisu dodatno prilagođavala, tj. ručno proširivala ili smanjivala. Smatrano je da takav disbalans savršeno odgovara skupu slika koji bi se prikupio snimanjem prometa prilikom vožnje u stvarnim uvjetima.

4.3. Pokretanje treninga i dobivanje rezultata

Pokretanje treninga pomoću YOLOv5 algoritma može se izvesti na više različitih načina. Ipak, svakako najjednostavniji način je pokretanje putem terminala unutar operacijskog sustava.

Treniranje se pokreće jednostavnom naredbom čiji je primjer u nastavku:

```
python train.py --batch 32 --epochs 100 --data data_morning.yaml --weights yolov5s.pt --name morning_final
```

Stavke unutar ove naredbe su sljedeće:

- ***python train.py*** definira koja skripta YOLOv5 algoritma će se koristiti, tj. koja je korisnikova namjera, ovdje je to *„train.py“* skripta unutar YOLOv5 algoritma koja služi za pokretanje procesa treninga

- ***batch*** brojčano definira tzv. *„batch size“,* tj. koji će se broj slika procesirati unutar modela prije nego se isti ažurira

- ***epochs*** brojčano definira broj epoha, odnosno koliko će puta algoritam u procesu treninga proći kroz cijeli trening skup podataka (sve slike)

- ***data*** definira lokaciju tj. put do željene *.yaml* datoteke koji unutar sebe sadrži ispravno strukturirane lokacije trening i validacijskih skupova podataka unutar računala kako bi se iste mogle dohvatiti i koristiti u procesu treninga i validacije

- ***weights*** definira željenu veličinu korištene YOLO mreže

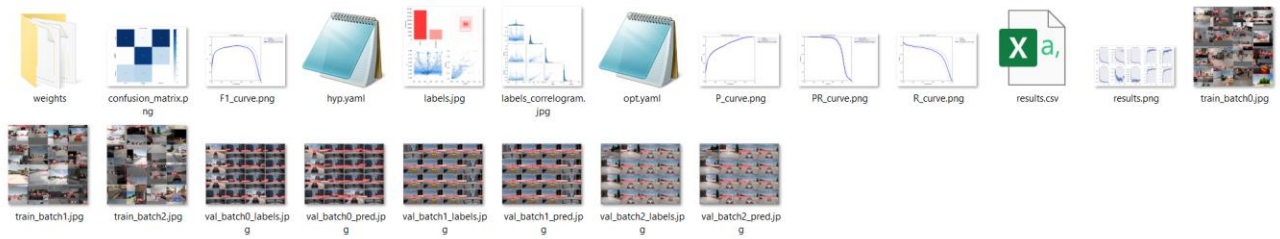
- ***name*** definira kako će se zvati direktorij u koji će se spremati rezultati treninga

Svi modeli unutar ovog rada trenirani su u 100 epoha, uz *batch size* 32 i *small* veličinu YOLOv5 mreže. S obzirom da se trenirani modeli sastoje od samo dvije klase i relativno

jednostavnog problema, nije bilo potrebe za većim modelom YOLO mreže. Treniranje je isprobano i na većim veličinama mreže (*medium* i *large*), ali isti nisu značajno pridonijeli na preciznosti modela, a istovremeno su treniranja bila znatno duža.

Po završetku 100 trening epoha, YOLO algoritam prolazi u jednoj dodatnoj epohi kroz validacijski skup podataka u kojem testira stvarnu uspješnost odrađenog treninga na način da pronalazi objekte na posebnom validacijskom skupu slika koje nije obradio prilikom treninga. Tom procedurom ispituje se stvarna kvaliteta odrađenog treninga, odnosno ukoliko je uspješnost detekcije na validacijskom skupu ista ili približna onoj postignutoj prilikom treninga – model se smatra uspješno istreniranim.

Nakon odrađenog treninga i validacije svi rezultati u vidu *excel* tablica, slika i grafova se spremaju u prethodno definirani direktorij za rezultate (Slika 22).



Slika 22 - Prikaz direktorija koji sadrži rezultate treninga

5. Detekcija dinamičkih objekata prema udaljenosti

5.1. Objašnjenje problema, prilagodba podataka i tumačenje rezultata

Prilikom vožnje, vozač mora budno paziti na prometne znakove, semafore i uvjete na cesti, ali svakako je najbitnija vlastita sigurnost te sigurnost ostalih sudionika u prometu. Iz tog razloga pravovremeno uočavanje ostalih sudionika u prometu i njihova udaljenost jedna je od glavnih briga i pri razvoju autonomnih vozila.

S tim ciljem provedeni su i treninzi modela u ovoj kategoriji. Pritom su kreirana tri skupa podataka s različitim grupacijama udaljenosti objekata (vozila i pješaka), odnosno:

- ***Less10*** kategorija sadrži slike i anotacijske podatke samo za bliske objekte, tj. objekte koji su na udaljenosti manjoj od 10 metara od autonomnog vozila

- ***10to50*** kategorija sadrži slike i anotacijske podatke samo za objekte koji su na srednjoj udaljenosti, odnosno udaljenosti od 10 do 50 metara od autonomnog vozila

- ***More50*** sadrži slike i anotacijske podatke samo za daleke objekte, odnosno objekte koji su na udaljenosti većoj od 50 metara od autonomnog vozila

S obzirom na to da jedna slika iz početnog (neobrađenog) skupa podataka može sadržavati objekte koji pripadaju različitim kategorijama udaljenosti, odnosno unutar jedne slike mogu postojati bliski, srednje udaljeni i daleki objekti - iste je trebalo odvojiti tako da se na pojedinoj slici promatraju samo objekti koji se i trebaju promatrati unutar te kategorije treninga, a oni koji ne pripadaju toj kategoriji se ignoriraju.

Početni skup podataka stoga se morao obraditi kroz za to namijenjene *Python* skripte kako bi računalo pravilno razvrstalo slike i njihove anotacije u svaku pojedinu kategoriju.

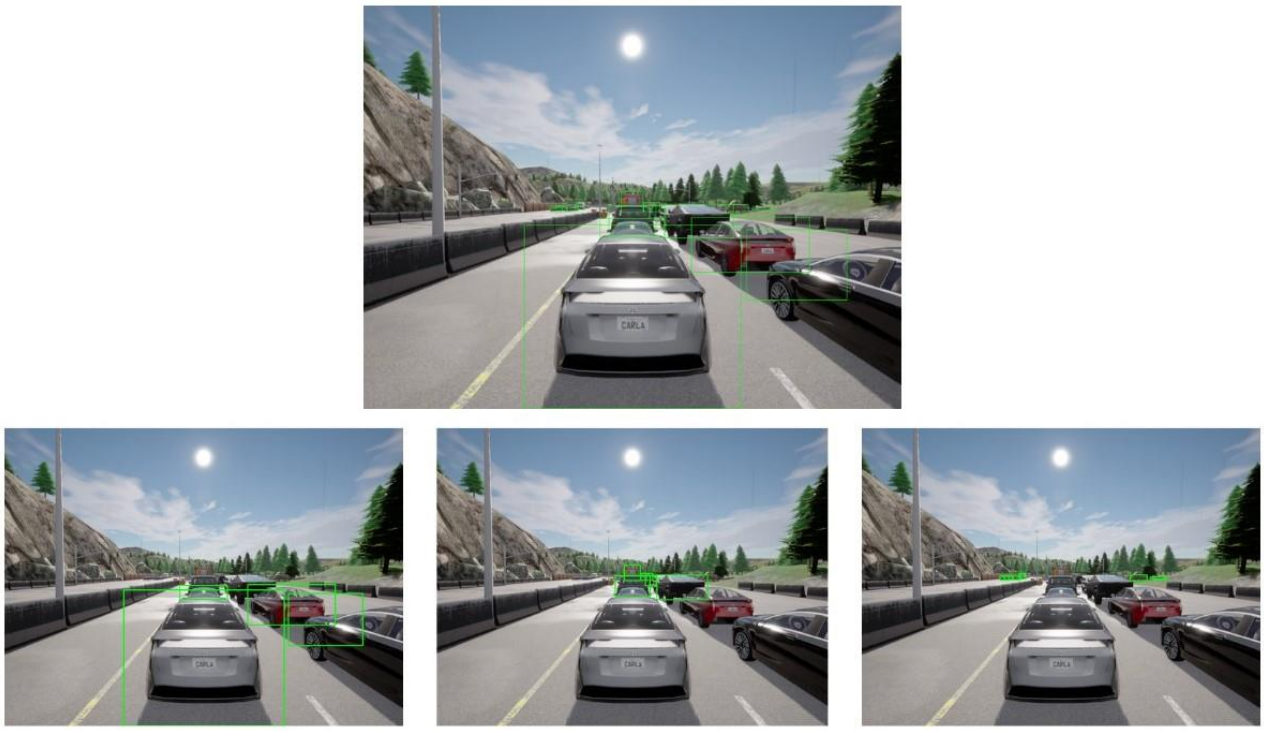
To je učinjeno tako da su se prvo uspoređivali nazivi datoteka iz početnog *depth* direktorija s nazivima datoteka iz početnog *labels* direktorija, a potom je računalo putem skripte svaku datoteku (*.txt file*) unutar oba direktorija otvaralo i čitalo. Ako je unutar pojedine *depth* datoteke zapisana vrijednost neke od linija teksta bila npr. manja od 10, linija teksta

unutar *labels* datoteke koja se nalazila na istoj poziciji tj. u istom redu kao u *depth* datoteci se potom kopirala u novu *labels* datoteku istoga imena, ali na novoj lokaciji primjerenoj za taj konkretni slučaj (npr. u *Less10* direktorij). Drugim riječima, kreirala se nova *labels* datoteka istoga naziva kao i originalna *labels* datoteka, ali nova datoteka je pritom sadržavala isključivo zapise iz originalne *labels* datoteke gdje je zadovoljen uvjet da je vrijednost u istoimenom *depth* dokumentu manja od 10. Sve ostale vrijednosti koje su veće od 10 su se ignorirale u tom slučaju, odnosno njihove linije teksta iz originalne *labels* datoteke također su se uspoređivale s vrijednostima iz *depth* datoteke, te su se potom kopirale i kreirale kao nove *labels* datoteke, ali u drugim odgovarajućim kategorijama tj. unutar 10to50 ili More50 direktorija.

Nakon što su se kreirale i pravilno razvrstale nove *labels* datoteke, druga *Python* skripta je potom čitala imena novokreiranih *labels* datoteka u svakom od tri kategorijska *labels* direktorija, te je potom kopirala slike iz *imgs* direktorija originalnog skupa podataka u ta tri direktorija. Pritom su se kopirale samo slike koje su imale svoj nazivni par unutar pojedinog *labels* direktorija, odnosno ako unutar *labels* direktorija nema *labels* datoteke istog imena kao što je originalna slika, ta slika se za taj direktorij zanemarila i nije se kopirala u njega.

Na taj način su se uz pomoć dvije jednostavne *Python* skripte brzo i efikasno kreirali novi, prilagođeni podatkovni skupovi spremni za pokretanje željenih treninga.

Iz navedenog je jasno da će pojedine originalne slike biti sadržane unutar sve tri kategorije, tj. kopirane u direktorije sva tri modela (ukoliko postoje pješaci ili vozila unutar slike na svim kategorijama udaljenosti), dok neke neće biti niti u jednoj od kategorija (ukoliko na slici nema nijednog pješaka niti vozila). Kako to izgleda kada se iscrtaju *bounding boxevi* oko objekata na slici koja sadrži objekte u sve tri daljinske kategorije prikazano je na slici ispod (Slika 23).



Slika 23 - Prikaz iste originalne slike unutar tri različite kategorije – originalna slika (gore), slika unutar Less10 skupa podataka (lijevo), slika unutar 10to50 skupa podataka (sredina) i slika unutar More50 skupa podataka(desno)

Na kolažnoj slici vidljivo je kako se na originalnoj slici iscrtavaju svi *bounding boxevi* oko vozila - neovisno o njihovoj udaljenosti, dok se na preostalim slikama promatraju i iscrtavaju samo *bounding boxevi* prikladni za promatranu kategoriju. Na isti način funkcioniraju i sami modeli prilikom svojih treninga.

Za svaku pojedinu kategoriju prikazati će se najvažniji grafovi kao i tablični prikazi praćenja metrika kroz procese treninga i validacije. Tablični prikazi sadržavaju prvu epohu svakog treninga, najbolju epohu svakog treninga te rezultate validacije. YOLOv5 automatski generira veliki broj slikovnih i grafičkih podataka, ali svakako je najbitnije izdvojiti grafove koji prikazuju kretanje svih metrika trening procesa, grafove odnosa *precisiona* i *recalla* (graf *mAP-a*), graf F1 mjere, te tzv. matricu zabune.

5.2. Less10 model

Unutar tablice (Tablica 2) prikazani su rezultati početne epohe, epohe u kojoj su se postigli najbolji rezultati te rezultati završne validacije. Svi rezultati zaokruženi su na 4 decimale.

Tablica 2 - Prikaz rezultata početne i najbolje epohe, te završne validacije Less10 modela

	Početna epoha	Najbolja epoha	Završna validacija
Train/box_loss	0.0637	0.0115	/
Train/obj_loss	0.0262	0.0061	/
Train/cls_loss	0.0079	0.0002	/
Precision	0.7328	0.9116	0.9090
Recall	0.6860	0.8009	0.7720
mAP_0.5	0.7122	0.8905	0.8600
mAP_0.5:0.95	0.3703	0.7120	0.6930
Val/box_loss	0.0470	0.0220	/
Val/obj_loss	0.0124	0.0135	/
Val/cls_loss	0.0026	0.0014	/
lrX	0.0700	0.0015	/

Sve metrike iz tablice mogle bi se izraziti i u postotcima na način da se navedene vrijednosti pomnože sa 100 zbog lakše predodžbe uspješnosti modela, ali zbog veće preciznosti zapisa, metrike su unutar tablice prikazane u originalnom decimalnom obliku. Za potrebe daljnjeg komentiranja rezultata po modelima, zbog prirodnijeg dojma stvarne uspješnosti modela svi decimalni zapisi rezultata pretvarati će se u postotke.

Iz tablice je vidljivo da su sve promatrane metrike imale ispravan i logičan rast ili pad, osim *object lossa* unutar privremene validacije. Iako bi u idealnom slučaju vrijednost svih vrsta gubitaka na modelu trebala padati, *val/obj_loss* je neznatno porastao. *Obj_loss* pokazuje koliko je model dobar u prepoznavanju da li se unutar *bounding boxa* nalazi objekt, pa zašto je onda u prvim fazama validacije bio bolji nego u kasnijim?

Pretpostavka je da se to događa zato što je unutar validacijskih epoha model puno osjetljiviji na greške zbog manjeg skupa podataka koje ujedno pritom i prvi put „vidi“. Drugim riječima, ukoliko model prilikom treniranja postane pretjerano „siguran“ u sebe, on tu samouvjerenost prenosi i na privremene validacije gdje se pritom može dogoditi da model npr. predvidi da se unutar određenog *bounding boxa* nalazi vozilo sa 95%-nom sigurnošću, a zapravo je taj *bounding box* prazan. Samo jedna takva kriva predikcija znatno više utječe na kretanje vrijednosti *object lossa* kod privremenih epoha nego što je to slučaj kod trening epoha. Ipak, kao što je rečeno - sam porast je u granicama početnih vrijednosti koje su pritom izrazito male, te kao takav ne utječe pretjerano na cjelokupnu kvalitetu modela. U nekom drugom slučaju ovaj porast *val/obj_loss-a* mogao bi se tumačiti kao pretreniranost (*eng. Overfitting*) modela, ali s obzirom da sve druge vrijednosti unutar modela ispravno rastu odnosno padaju, to ovdje nije slučaj.

Unutar procesa završne validacije ne prate se nikakvi gubitci niti *learning rate* što je i logično jer prilikom tog procesa model više ništa ne uči, već samo ispituje stvarnu kvalitetu treniranog modela. Stoga, unutar završne validacije prate se samo četiri metrike: *precision, recall*, *mAP_0.5* i *mAP_0.5:0.95*. Sve navedene metrike su se na validacijskim podacima zadržale na približnoj razini rezultata iz najbolje trening epohe, stoga se može reći da je treniranje modela uspješno provedeno.

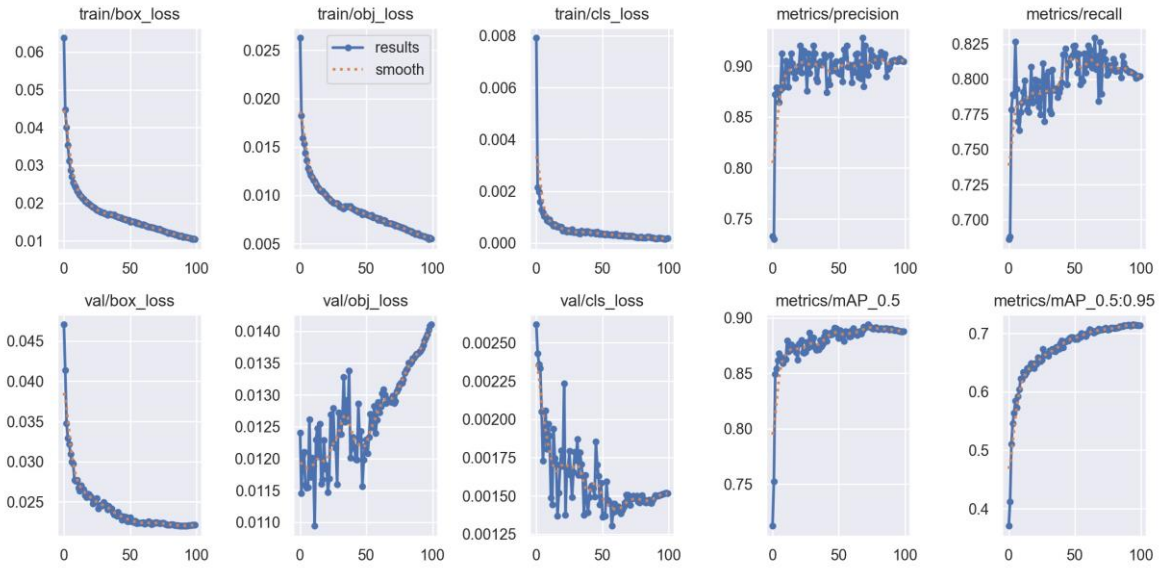
Iako su pri znanstvenom promatranju sve metrike modela najčešće podjednakih važnosti, kada bi jednom metrikom trebali nekome objasniti opću „uspješnost“ kojom određeni model predviđa objekte na slici, to bi svakako bila *mAP_0.5* metrika pošto ona najbolje sažima osnovnu ideju iza modela detekcije objekata. *Less10* model je u tom segmentu postigao odlične najviše rezultate koji iznose 89.05% u trening modelu i 86% prilikom validacije.

mAP_0.5:0.95 znatno je stroža metrika od *mAP_0.5*, pa iako je korisna, puno veća važnost se pridaje *mAP_0.5*, odnosno *IoU* postavljen na 0.5 se smatra dovoljnim za uspješnu detekciju. Ipak, zanimljivo je primjetiti je da je upravo *mAP_0.5:0.95* kao „najstroža“ metrika, postigla najveći porast u vrijednosti od početne epohe do najbolje epohe treninga što dodatno potvrđuje kvalitetu odrađenog treninga.

Precision i *recall* metrike su u pravilu izrazito međuzavisne, a važnost odnosa njihovih rezultata ovisi o samom modelu tj. krajnjem cilju onoga što želimo postići sa modelom. Iako su u većini modela važnosti ove dvije metrike podjednake, kod pojedinih modela se veća važnost pridaje *precision* metrici, dok kod nekih drugih modela veću važnost ima *recall* metrika. U ovom konkretnom modelu, iako je i *precision* izrazito bitan, veću važnost ipak ima *recall* metrika, što je i logično jer puno je bitnije da model točno detektira prisutnost objekta unutar pojedinog *bounding boxa,* nego što je bitna točna detekcija same klase unutar *bounding boxa,* pošto model sadrži samo dvije klase koje su zapravo podjednakih važnosti.

Naime, puno veći problem i opasnost u slučaju autonomnog vozila predstavljala bi situacija kada se vozilo ili pješak ne bi uopće detektirali prilikom vožnje, nego što bi to bio slučaj kada bi model pogrešno odredio klasu objekta, odnosno vozilo detektiralo kao pješaka ili pješaka kao vozilo. Obje navedene metrike u ovom modelu postižu visoke i zadovoljavajuće razine točnosti pri čemu je razlika između njih iznosi otprilike 10%, tj. *precision* postiže točnost od oko 90% na treningu i validaciji, dok *recall* postiže točnost od oko 80% na treningu i validaciji.

Svi podaci prikazani unutar tablice mogu se vidjeti i na skupu grafičkih prikaza koje YOLO automatski generira po završetku treninga (Slika 24).



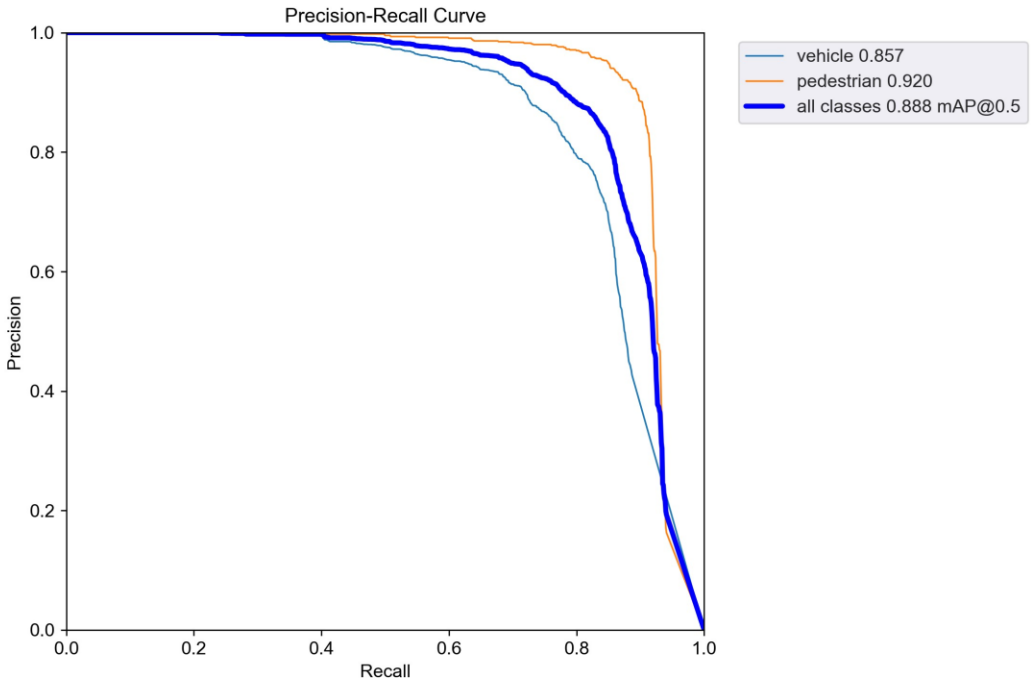
Slika 24 - Prikaz kretanja vrijednosti svih metrika po epohama kroz proces treniranja Less10 modela

Ovakav skup grafova izrazito je koristan kako bi se vidjele nepravilnosti koje nije uvijek lako isčitati iz tabličnih prikaza. Vidimo kako svi gubitci unutar trening epoha imaju lijep i pravilan pad, dok je kod privremenih validacija situacija ipak znatno drugačija. Jedini validacijski gubitak koji ima ispravan pad je *val/box_loss*, dok *cls_loss* iako ima tendenciju pada, taj pad sadrži znatno veći šum i odskakanje vrijednosti. *Val/obj_loss* je jedina vrsta gubitka koja raste kao što je prethodno objašnjeno.

Iako na grafu *val/obj_loss* i *val/cls_loss* mogu izgledati pomalo zabrinjavajuće, takva odstupanja su u potpunosti normalna i mogu se pripisati promjenama vrijednosti *learning ratea* između epoha treninga, te disbalansu u količini podataka, kao i disbalansu samih instanci klasa između trening i validacijskog skupa podataka. Da su ta odstupanja znatno veća u iznosima to bi bio znatno veći problem, ali u ovom slučaju ta odstupanja su u postotnim promilima što nema gotovo nikakvog utjecaja na sami model.

Sve ostale metrike kao što su *precision*, *recall* i *mAP*-ovi imaju tendenciju rasta što je i poželjno. *Precision* i *recall* pritom imaju puno veći šum kroz proces treninga, ali to je i očekivano s obzirom da YOLOv5 trenira adaptivno trudeći se postići najbolje moguće rezultate navedenih metrika. Veće promjene u vrijednostima *learning ratea* te korištenje različitih regularizacijskih tehnika (npr. *weight decay)* između epoha mogu uzrokovati pojavu nešto većeg šuma u određenim periodima treninga. Ipak, čak niti taj prividno veliki šum ne treba uzimati kao preveliki problem pošto se razlike u vrijednostima među epohama gdje je prisutan šum na grafu kreću u rasponu manjem od 5%.

YOLOv5 generira i neke pojedinačne grafove, a svakako najbitniji takav graf je graf međuodnosa *precisiona* i *recalla* (Slika 25).



Slika 25 - Prikaz odnosa precisiona i recalla u Less10 modelu

Prikazani graf na jednostavan način stavlja u međuodnos *precision* i *recall*, te na taj način daje maksimalne vrijednosti *mAP*-a svake pojedine klase, kao i najveću vrijednost *mAP*-a za sve klase zajedno.

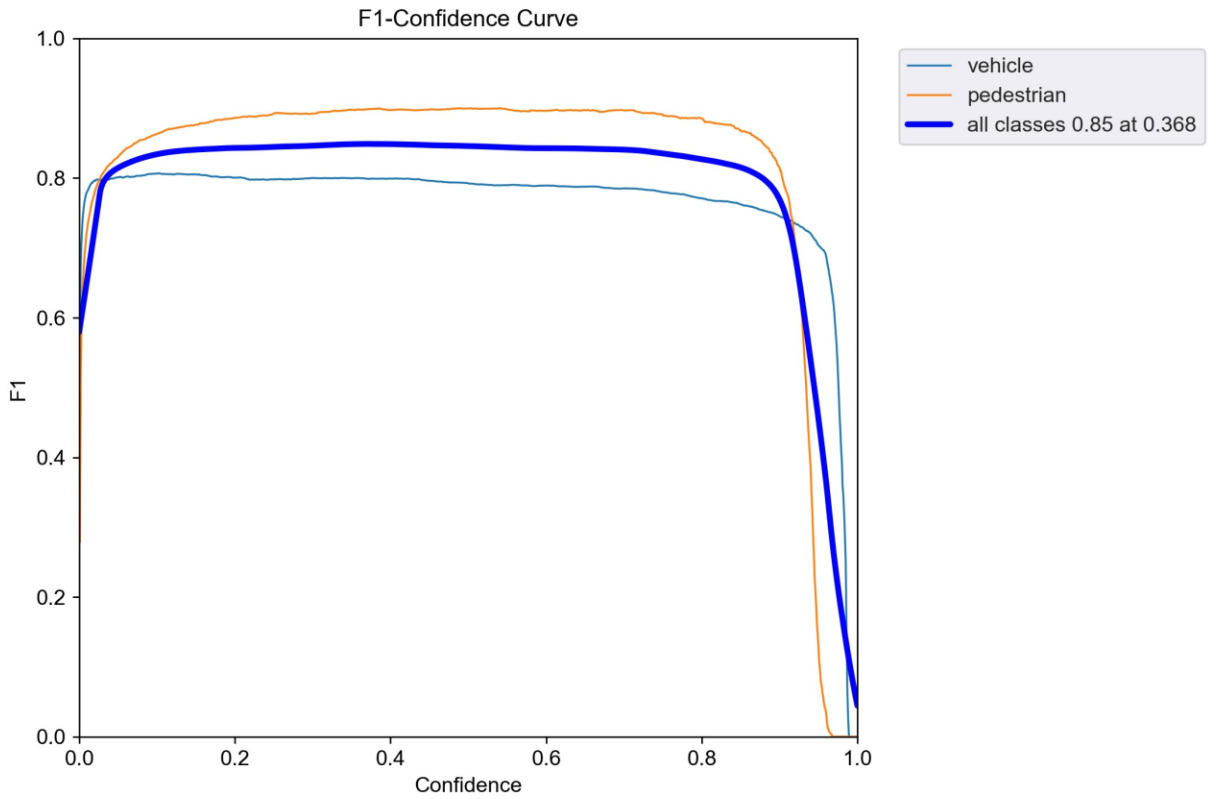
Na navedenom grafu vidimo da *mAP*_0.5 za sve klase iznosi 88% kada je *precision* na oko 90% i *recall* na oko 80% što odgovara i rezultatima iz tablice sa početka poglavlja.

Ono što je iznenađujuće je to da klasa pješaka doseže veći *mAP_0.5* od klase vozila. Kod klase pješaka iznosi 92%, dok je kod klase vozila na oko 85%. Taj međuodnos je pomalo iznenađujući, ali biti će jasnije zašto je to tako na kasnijem prikazu matrice zabune.

Drugi bitan graf ovog tipa je F1 graf koji prikazuje F1 mjeru koja označava harmonijsku sredinu između *precisiona* i *recalla*, odnosno u kakvom su međuodnosu *precision* i *recall* tijekom procesa treninga (Slika 26). Formula F1 mjere glasi:

$$F1\ score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Jednadžba 4 - F1 mjera (Izvor: Seo, et. el., 2021)



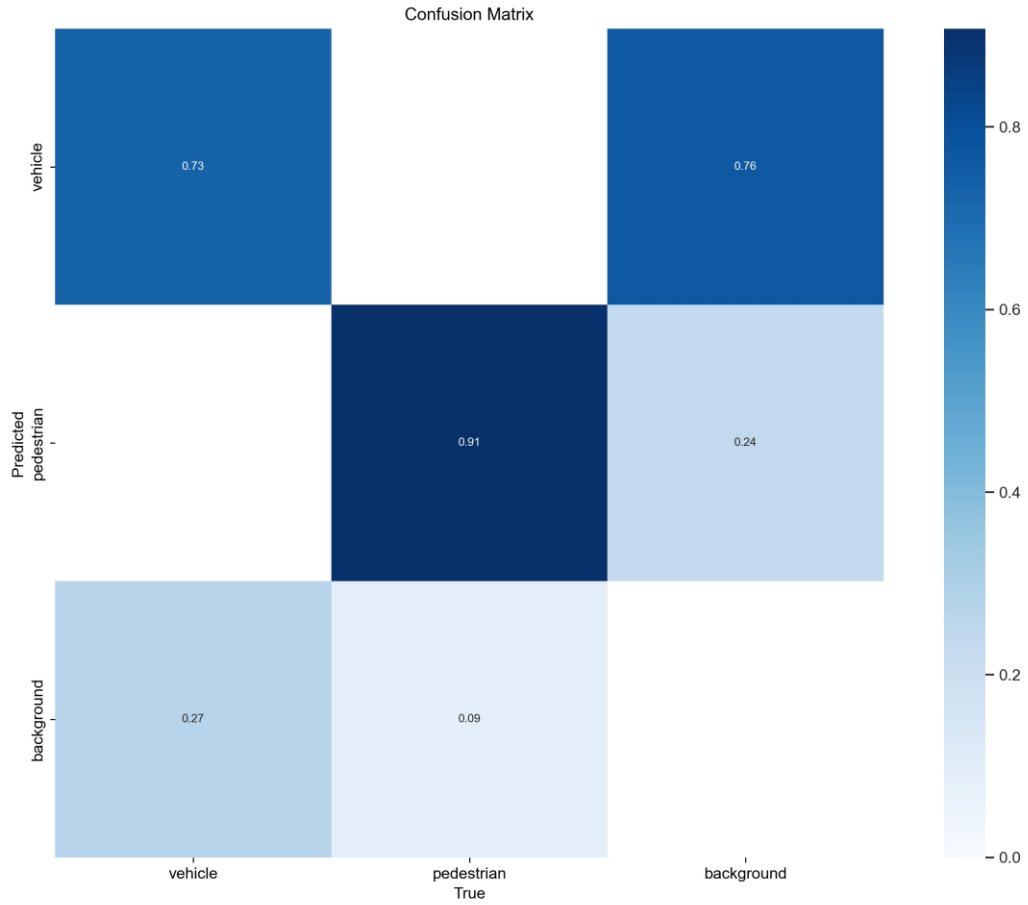
Slika 26 - F1 mjera harmonijske sredine precisiona i recalla u Less10 modelu

Kod F1 grafova bitno je da linije na grafu što više teže gornjem vrhu grafa i da su pritom što ravnije, pošto to označava dobar međuodnos *precisiona* i *recalla* kroz cijeli proces treninga. U našem slučaju vidimo da se harmonija održava konstantno i da u najboljem slučaju iznosi oko 85%, što je objašnjivo i time da je *precision* u svojim najboljim trenutcima iznosio oko 90%, dok je *recall* iznosio oko 80%.

Ono što je važno za objasniti je funkcija *x* osi prikazanog F1 grafa, tj. vrijednost „samopouzdanja“ (*eng. Confidence*). Upravo to je vrijednost koja govori o kvaliteti balansa između *precisiona* i *recalla*. Najlakše je promatrati parametar samopouzdanja kao svojevrsnu „klackalicu“, naime u rasponu samopouzdanja od 0 do 1, ako je nivo samopouzdanja prenizak to ujedno može označavati i nisku razinu *precisiona* samoga modela, dok ukoliko bi nivo samopouzdanja bio previsok to bi označavalo nizak *recall*. Stoga, najbolji slučaj je kada se nivo samopouzdanja kreće što bliže vrijednosti od 0.5.

Na gornjem grafu vidimo da se najveća harmonija između *precisiona* i *recalla* postigla kada je nivo samopouzdanja iznosio 0.368, odnosno tada su uspješnosti detekcije i klasifikacije bile najizjednačenije. Također, to nam govori da u tom trenutku *recall* vjerojatno nije bio na svojoj najvišoj razini koja je iznosila oko 80%, ali je zato vrijednost *precisiona* bila nešto viša od prosječne, odnosno u tim trenucima model je naginjao prema *precision* vrijednostima.

Najčešće i najzanimljiviji grafički prikaz za proučavanje je tzv. matrica zabune (*eng. Confusion Matrix*). To je matrica koja prikazuje koliki je omjer stvarnih i lažnih (eng. *True Positive* i *False Positive*) predviđanja koja je model napravio po svakoj klasi (Slika 27).



Slika 27 - Prikaz matrice zabune za Less10 model

Redci unutar matrice prikazuju predviđene klase, dok stupci prikazuju stvarne klase objekata. Unutar matrice zabune uvedena je i klasa pozadine (*eng. background*), iako to nije službena klasa našega modela. To je u potpunosti normalno kod svake matrice

zabune u algoritmima za detekciju objekata, upravo zato što se klase na slici pokušavaju uspješno izdvojiti od pozadine, odnosno osim što model može krivo odrediti klase prepoznatih objekata, isto tako se svaki objekt na slici može zamijeniti za pozadinu tj. neklasificiranu okolinu.

Matricu je najlakše promatrati i objasniti stupčano, pa tako krećući od prvoga stupca koji predstavlja klasu vozila vidimo da je stvarnih tj. točnih klasifikacija vozila bilo 73%, dok je u 27% slučajeva model za vozilo krivo pretpostavio da je ono pozadina. Ovako visok postotak miješanja vozila s pozadinom i nije toliko čudan, s obzirom na to da vozila na udaljenostima manjima od 10 metara se mogu nalaziti neposredno ispred ili pored kamere, pri čemu kamera ne vidi cijelo vozilo, već vidi veliku nedefiniranu plohu pa ga stoga prepoznaje kao pozadinu. Odlično je to što model nijednom nije zamijenio vozilo za pješaka ili obratno.

Točnost klasifikacije pješaka je znatno veća i ona iznosi 91%, dok je u preostalih 9% slučajeva model zamijenio pješaka za pozadinu. Ovako visoka razlika u točnosti naspram vozila je pomalo iznenađujuća pošto bi se očekivalo da će vozila imati puno veću točnost detekcije s obzirom na to da su obujmom veća i prepoznatljivija, ali vidimo da to ipak nije slučaj. Razlog toga je što su pješaci uvijek jasno vidljivi na tako malim udaljenostima i gotovo nikad ne ulaze u neposrednu blizinu kamere što znatno olakšava njihovu detekciju za razliku od vozila.

Posljednji stupac matrice prikazuje u kojem je postotku pozadina pogrešno klasificirana kao pješak ili vozilo. Iz matrice je vidljivo da je u 76% slučajeva pozadina krivo prepoznata kao vozilo, a u 24% slučajeva kao pješak.

Moglo bi se pomisliti da je loše to što je toliko visoka razina lažnog prepoznavanja vozila ili pješaka, ali zapravo je to i dalje relativno pozitivan pokazatelj kvalitete treniranog modela. Razlog zašto je to pozitivan pokazatelj leži u tome što je svakako bolje da autonomno vozilo nešto lažno detektira kao vozilo ili pješaka i samim time uspori i prilagodi svoju brzinu, nego što bi bio slučaj da tih lažnih detekcija nema i vozilo pritom uopće ne detektira vozilo ili pješaka pred sobom.

Naravno, idealan slučaj bio bi kada nikakvih lažnih detekcija ne bi bilo i da model pritom savršeno u 100% slučajeva prepoznaje i vozila i pješake. Ipak, takvu razinu savršene detekcije je i u stvarnim modelima autonomnih vozila još uvijek nemoguće postići. Ono što je pozitivno je to da usprkos lažnim prepoznavanjima model i dalje zadržava visoke razine *precisiona*, *recalla* i *mAP-a*.

Uz sve grafove i tablice, YOLOV5 sprema i odabrane prikaze detekcije objekata na slikama u periodima treninga i validacije. Jedan takav kolaž slika u procesu validacije prikazan je na slici ispod (Slika 28).

Slika 28 - Prikaz detekcije objekata na uzorku validacijskog skupa podataka Less10 modela

Tijekom detekcije objekata na slikama validacijskog skupa, objektima se dodjeljuje njihova predviđena klasa i pretpostavka modela kolika je vjerojatnost da pojedini objekt pripada toj klasi. U procesu treninga princip rada je isti, osim što se objekti tada ne definiraju po njihovim nazivima klasa, već po njihovoj oznaci klase, odnosno vozila se prilkom treniranja označavaju sa oznakom 0, a pješaci sa oznakom 1.

5.3. 10to50 model

Kao i kod prethodnog modela, putem tablice prikazani su rezultati prve epohe treninga, najbolje epohe treninga i završne validacije (Tablica 3).

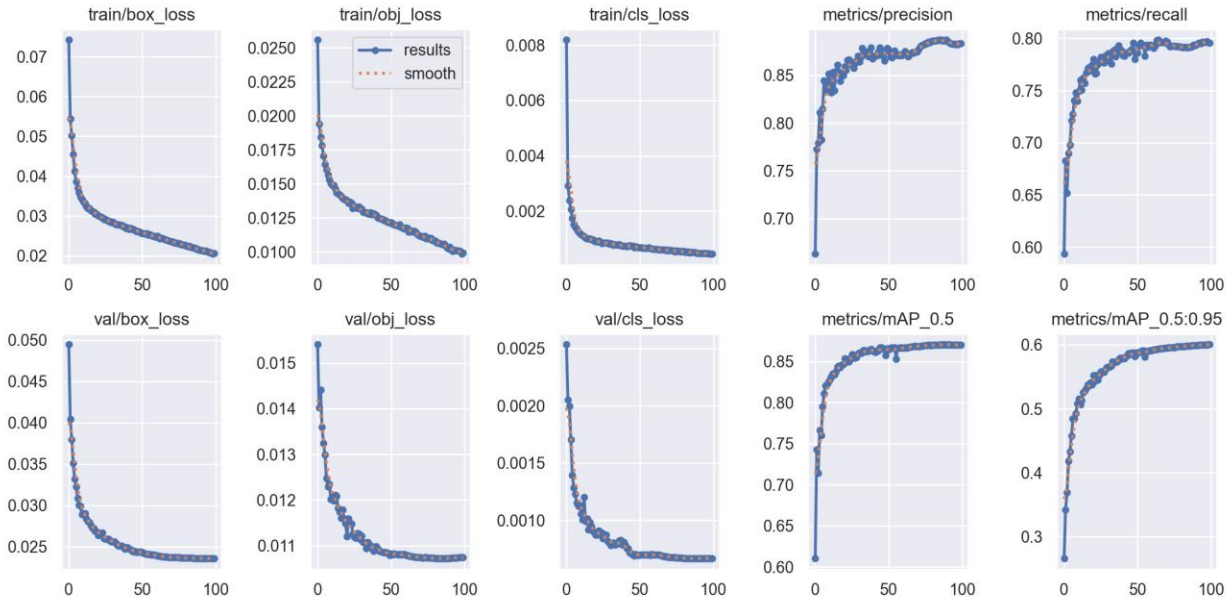
Tablica 3 - Prikaz rezultata početne i najbolje epohe, te završne validacije 10to50 modela

	Početna epoha	Najbolja epoha	Završna validacija
Train/box_loss	0.0742	0.0205	/
Train/obj_loss	0.0256	0.0098	/
Train/cls_loss	0.0082	0.0004	/
Precision	0.6629	0.8834	0.8840
Recall	0.5934	0.7960	0.7960
mAP_0.5	0.6102	0.8705	0.8700
mAP_0.5:0.95	0.2658	0.6005	0.6000
Val/box_loss	0.0494	0.0236	/
Val/obj_loss	0.0154	0.0108	/
Val/cls_loss	0.0025	0.0007	/
lrX	0.0700	0.0004	/

Iz tablice se može zaključiti da je trening bio još uspješniji od treninga *Less10* modela. Naime, apsolutno sve vrijednosti gubitka su padale tijekom treninga i svedene su na neznatnu razinu. Metrike *precisiona*, *recalla* i *mAP_0.5* su na zadovoljavajuće visokim razinama, dok je *mAP_0.5:0.95* nešto niži, ali s obzirom na svoju strogoću *IoU*-a i dalje je točnost te metrike na primjerenoj razini. *Precision* doseže razinu od oko 88% uspješnosti u najboljem slučaju, *recall* varira oko 80% točnosti, dok je *mAP_0.5* na visokih 87%.

Ono što je najbitnije za primijetiti je to da su vrijednosti metrika unutar završne validacije na skoro pa identičnim razinama kao i vrijednosti istih metrika unutar najbolje trening epohe modela. Kod *Less10* modela razlike navedenih metrika između najbolje epohe treninga i završne validacije kretale su se unutar 3%, dok ih ovdje praktički i nema. To je ujedno glavni pokazatelj uspješnosti odrađenog treninga.

Graf praćenja metrika u procesu treninga prikazan je na slici ispod (Slika 29).



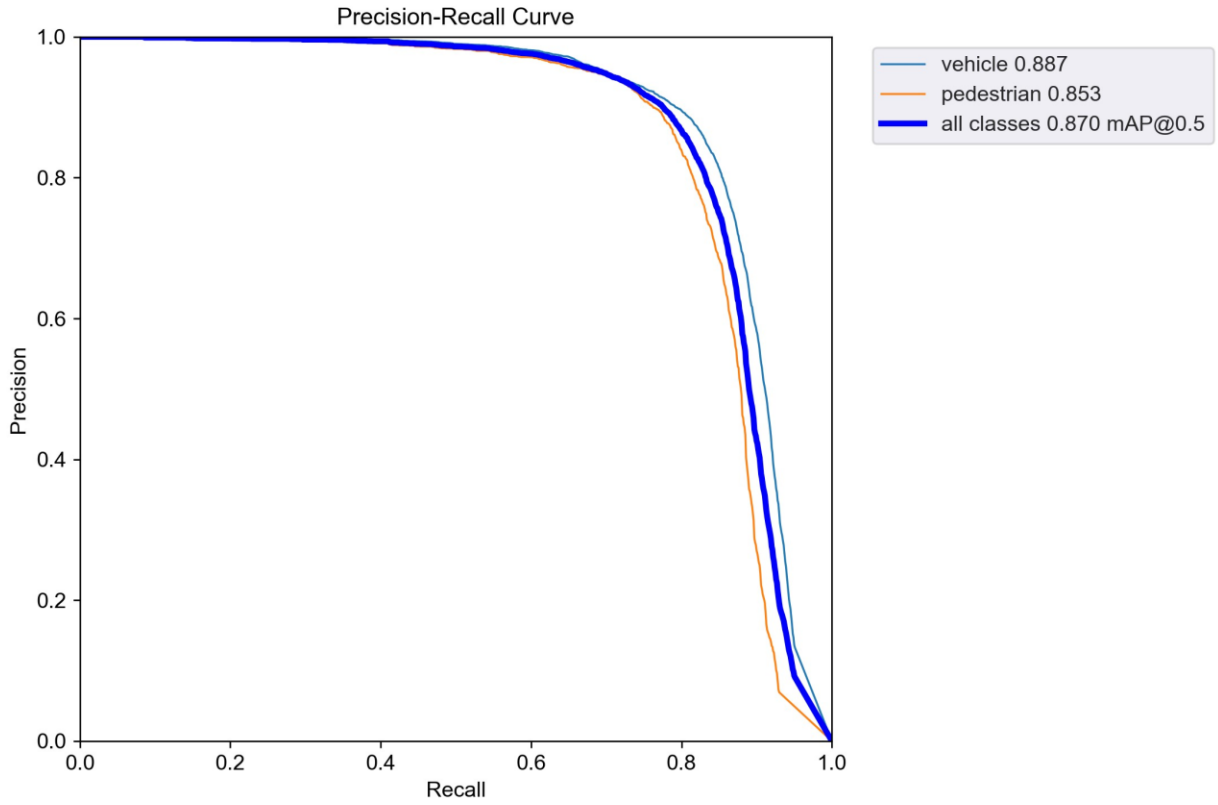
Slika 29 - Prikaz kretanja vrijednosti svih metrika po epohama kroz proces treniranja 10to50 modela

Iz grafa je vidljivo da je proces treniranja bio skoro pa savršen. Sve metrike gubitaka imaju tendenciju pada bez ikakvih šumova na ijednom od grafova. Sve ostale metrike imaju tendenciju stabilnog rasta uz minimalne doze šuma na *precision* i *recall* krivuljama koje pritom nisu pretjerano velike niti neuobičajene da bi predstavljale ikakav problem.

Razlog stabilnosti treninga i validacije unutar ovoga modela leži u više faktora:

- sadržava najveći broj slika od sva tri modela u ovoj kategoriji
- najveći broj slika znači i najveći broj instanci klasa (količine objekata za detekciju)
- objekti nisu u nijednom trenutku niti predaleko niti preblizu kamere

Graf međuodnosa *precisiona* i *recalla*, odnosno *mAP-a* je prikazan na slici ispod (Slika 30).

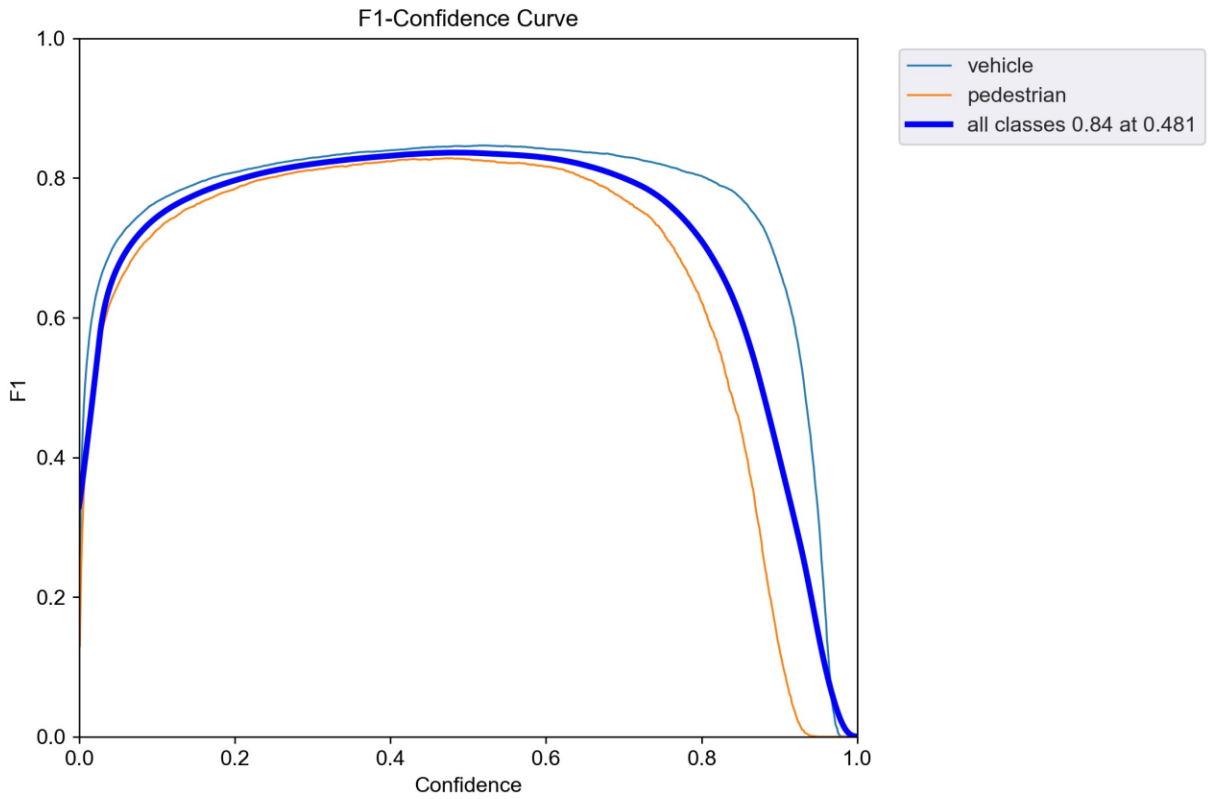


Slika 30 - Prikaz odnosa precisiona i recalla u 10to50 modelu

Na navedenom grafu vidljivo je da *mAP_0.5* za sve klase iznosi 87% kada točnost klasifikacije *precisiona* iznosi oko 88% te kada je točnost *recalla* na oko 80%, što odgovara i rezultatima iz tablice sa početka poglavlja. Pozitivno kod ovog modela je što vidimo da nema znatnih odstupanja u odnosima pojedinih klasa, tj. klasa vozila doseže svoj najveći *mAP* od oko 88%, dok je kod klase pješaka to oko 85%. Samim time se te dvije krivulje prate i preklapaju.

Na ovom modelu vidimo da osim grafički i rezultatski pravilnijih rezultata, ujedno je i sama razina točne klasifikacije puno „logičnija“, odnosno model ovdje ipak lakše i točnije prepoznaje vozila nego pješake, dok je kod *Less10* modela to bilo obratno.

F1 graf, odnosno graf harmonijske sredine *precision* i *recall* metrika prikazan je na sljedećoj slici (Slika 31).

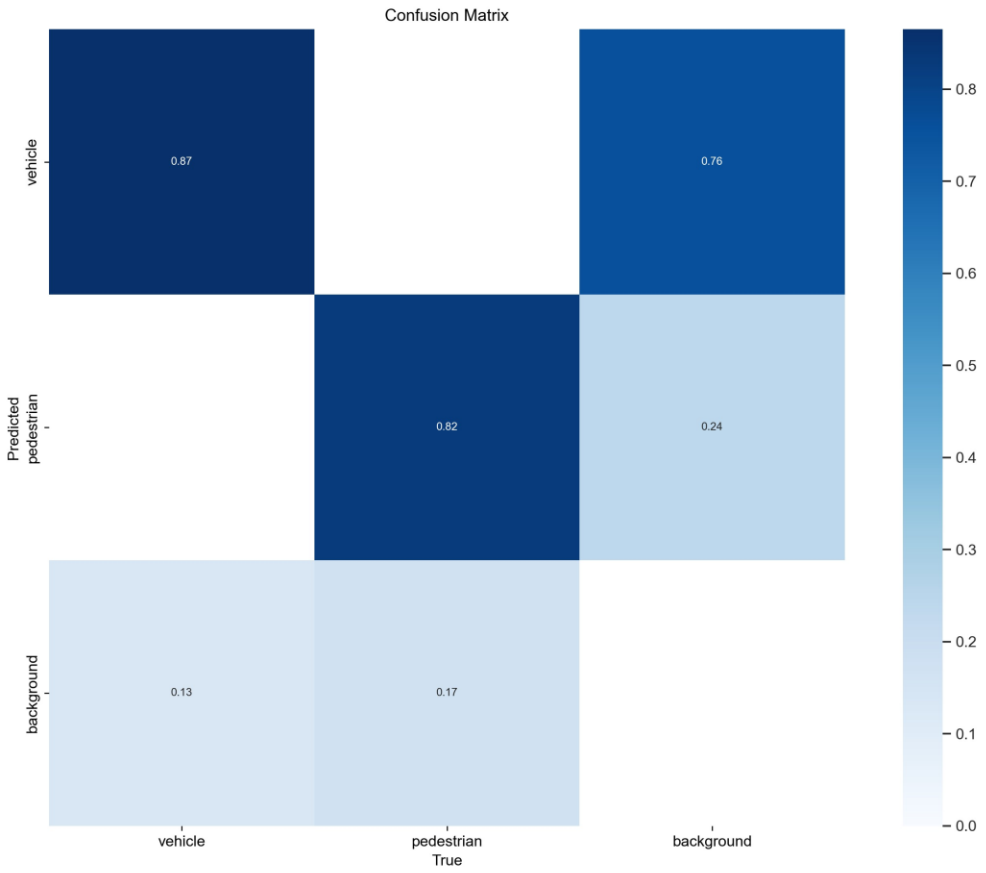


Slika 31 - F1 graf 10to50 modela

F1 graf ovog modela podosta je sličan i F1 grafu *Less10* modela, s time da ovdje graf usprkos tome što izgleda manje harmonično, ima pozitivnu tendenciju nagiba ka gornjem vrhu grafa što je uvijek poželjno za F1 mjeru. Najveća harmonijska vrijednost *precisiona* i *recalla* iznosi 84% što opravdava najveće vrijednosti precisiona u modelu od oko 88% i vrijednosti recalla od 80%.

Vidljivo je i da je nivo samopouzdanja u trenutku najveće harmonijske vrijednosti iznosio 0.48 (48%), što je vrlo blizu idealne vrijednosti koja iznosi 0.50 i označava savršen balans između *precisiona* i *recalla.*

Matrica zabune modela prikazana je na slici ispod (Slika 32).



Slika 32 - Prikaz matrice zabune za 10to50 model

Promatrajući prvi stupac vidi se da je model imao 87% točnih predviđanja klase vozila, dok je u 13% slučajeva vozilo zamijenio za pozadinu. Drugi stupac koji se odnosi na pješake kazuje kako je model imao 82% točnih predviđanja klase pješaka dok je u 18% slučajeva iste zamijenio za pozadinu. Završni, treći stupac ima identične rezultate kao i kod *Less10* modela, odnosno u 76% slučajeva je nešto što je zapravo bila pozadina model prepoznao kao vozilo, dok je u preostalih 24% slučajeva pozadinu lažno prepoznao kao pješaka.

Znatan skok se dogodio u uspješnosti točnih predviđanja klase vozila naspram *Less10* modela gdje je ista iznosila 73% točnih predviđanja, ali je pritom točnost predviđanja za klasu pješaka pala za oko 10%. Ipak, može se reći da su takvi rezultati i očekivani s obzirom da je u ovom modelu vozila znatno lakše za detektirati zbog jasnije vidljivosti istih, dok je pješake znatno teže za detektirati s porastom njihove udaljenosti od kamere.

5.4. More50 model

Tablica posljednjeg modela iz ove kategorije prikazana je u nastavku (Tablica 4).

Tablica 4 - Prikaz rezultata početne i najbolje epohe, te završne validacije More50 modela

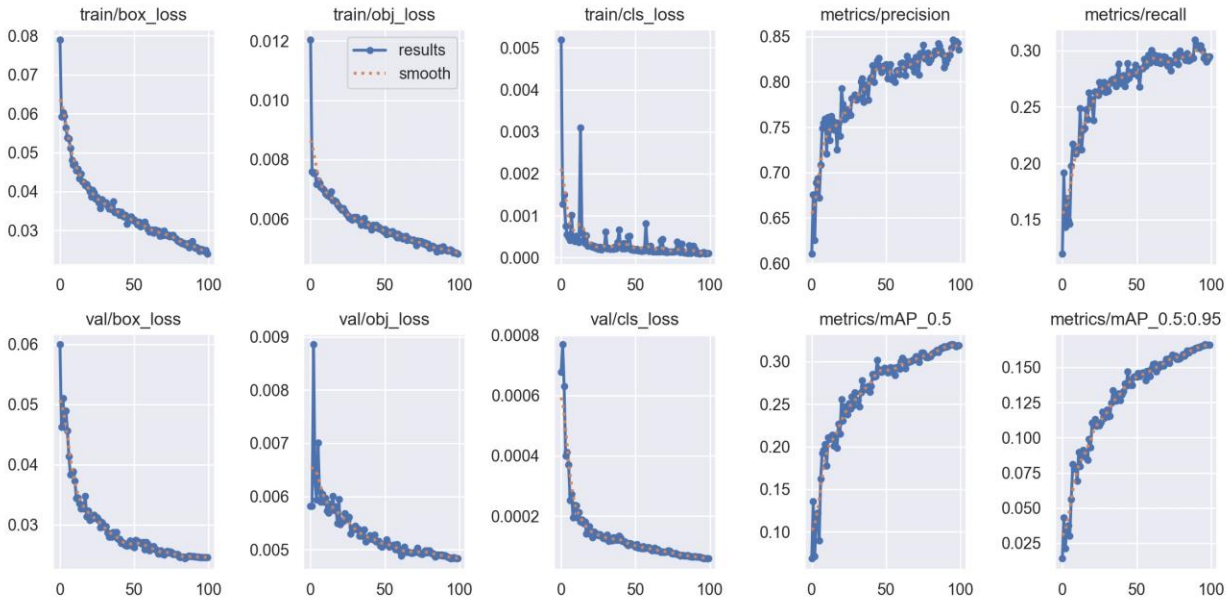
	Početna epoha	Najbolja epoha	Završna validacija
Train/box_loss	0.0789	0.0239	/
Train/obj_loss	0.0120	0.0048	/
Train/cls_loss	0.0052	9.8704e-05	/
Precision	0.6103	0.8355	0.8410
Recall	0.1196	0.2946	0.2950
mAP_0.5	0.0687	0.3187	0.3190
mAP_0.5:0.95	0.01414	0.1659	0.1660
Val/box_loss	0.0599	0.0247	/
Val/obj_loss	0.0058	0.0049	/
Val/cls_loss	0.0007	5.9296e-05	/
lrX	0.0033	0.0003	/

Iako bi se gledajući gubitke moglo reći da je ovo najbolji model s obzirom da su iznosi gubitaka u obje skupine najniži od sva tri promatrana modela, to ipak nije slučaj. *Precision* metrika također postiže visoku točnost od oko 84% što je odlično s obzirom na udaljenost objekata. Ipak, promotre li se *recall* i *mAP* metrike vidljivo je da je model u potpunosti neiskoristiv.

Razlog toga krije se u tome što CARLA na udaljenostima većima od 50 metara uopće ne generira pješake. Unutar 1674 slike validacijskog skupa podataka za *More50* model pronađeno je 2650 vozila i jedan jedini pješak.

Stoga, objašnjivo je da je *precision* metrika pritom vrlo visoka jer je model relativno uspješno klasificirao objekte (vozila) na slikama, ali su sve ostale bitne metrike izrazito niske s obzirom na nesrazmjer broja vozila i pješaka na slikama. Model naime prilikom svog procesa validacije nije uspio niti tog jednog pješaka točno prepoznati, dakle *recall* u slučaju klase pješaka iznosi 0.

Kako kretanje metrika kroz proces treninga izgleda na grafu metrika prikazano je na slici ispod (Slika 33).

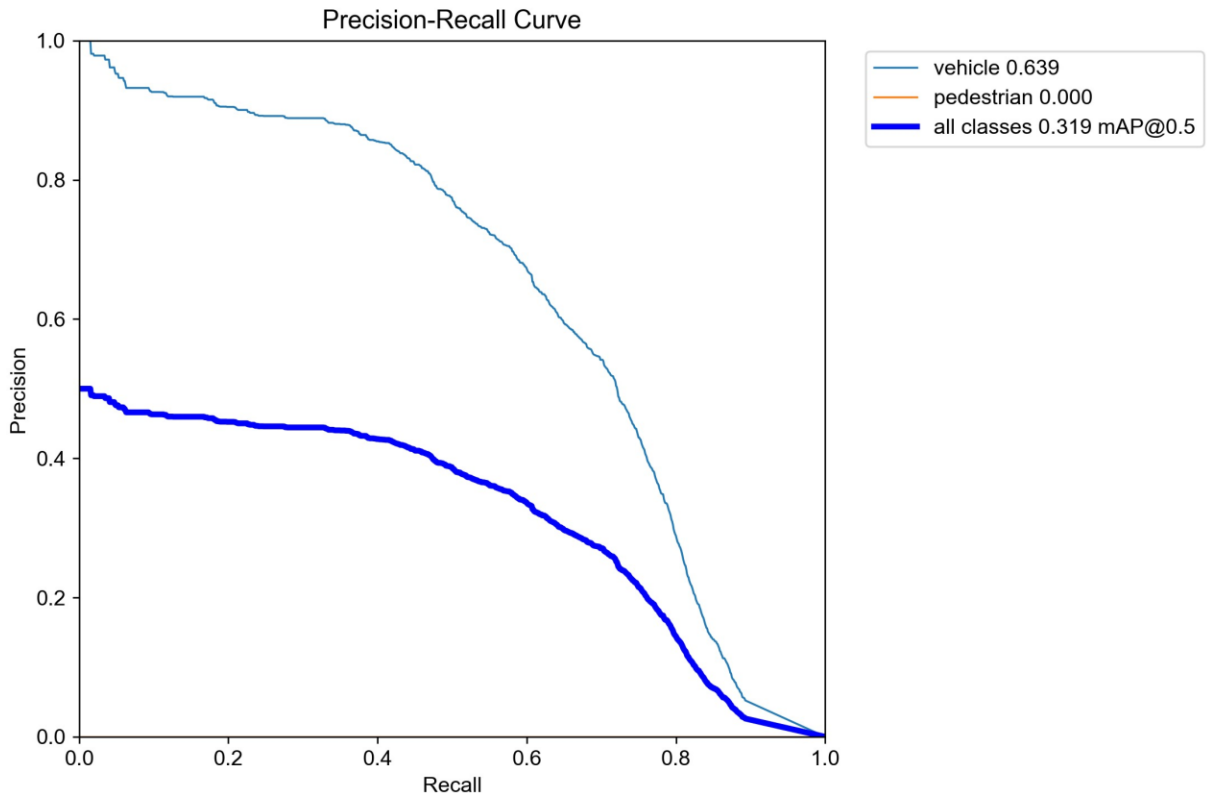


Slika 33 - Prikaz kretanja vrijednosti svih metrika po epohama kroz proces treniranja More50 modela

Gledajući isključivo graf moglo bi se reći da model zapravo ima ispravno kretanje svih metrika, ali znajući točne brojke iz prethodne tablice i promatrajući koje vrijednosti zapravo sve te metrike postižu jasno je da je model zapravo neiskoristiv.

Znatna količina šuma na svakom od grafova metrika je prisutna što je objašnjivo uz sve prethodno navedene zaključke, kao i to da je model prilikom treniranja „razumio“ da nešto nije uredu i trudio se svim silama podesiti hiperparametre kako bi se postigla maksimalna vrijednost navedenih mjera.

Graf odnosa *precisiona* i *recalla* također je zanimljiv za promatranje u slučaju ovoga modela (Slika 34).



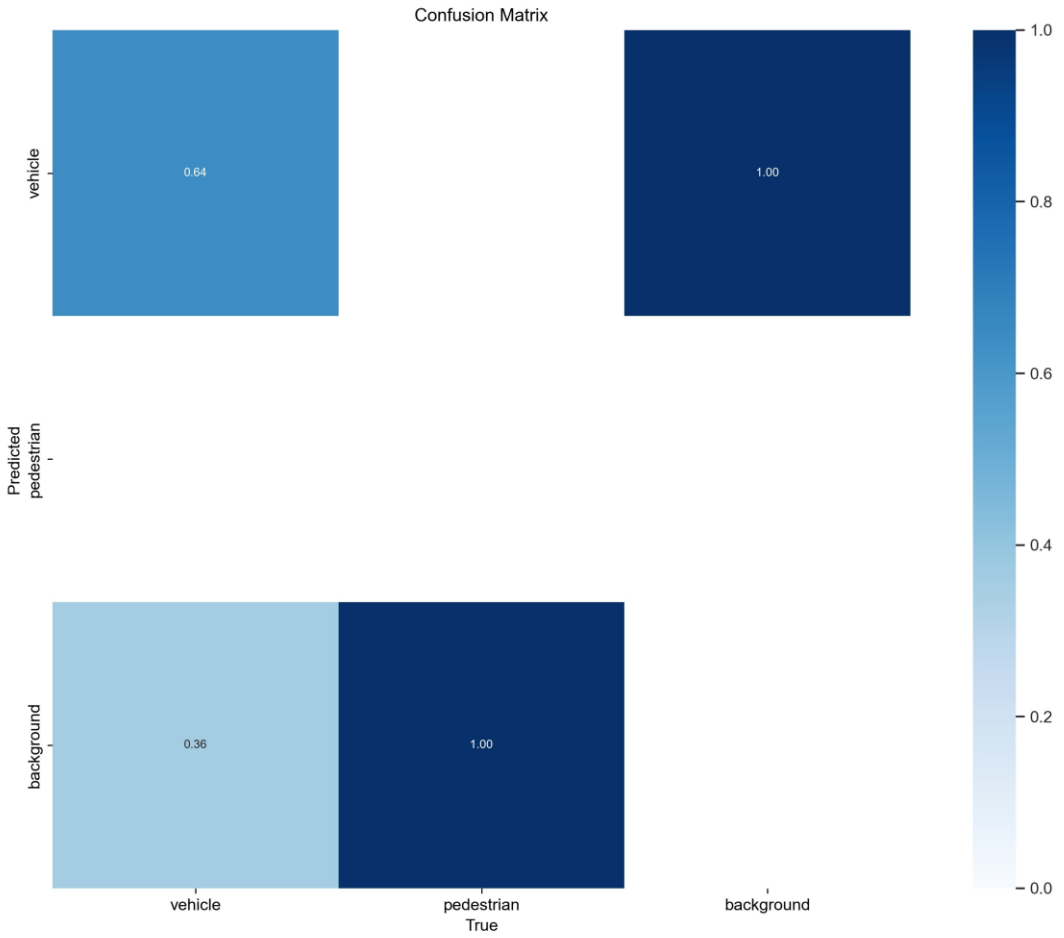
Slika 34 - Prikaz odnosa precisiona i recalla u More50 modelu

Usporedi li se ovaj graf *precisiona* i *recalla* sa grafovima prethodna dva modela vidljivo je koliko je zapravo ovaj model podbacio u svom cilju. Naime, čak i klasa vozila nema visoku vrijednost *mAP_0.5*, odnosno ona u najboljem slučaju iznosi oko 64% što je neprihvatljivo za potrebe korištenja u autonomnim vozilima.

Krivulja za pješake očekivano ni ne postoji na grafu pošto joj je iznos 0, dok obje klase u zbroju postižu najveći *mAP_0.5* od oko 32% što je i logično s obzirom da je upravo to aritmetička sredina zbroja *mAP*-a obje klase.

F1 graf u ovom slučaju nema ni smisla prikazivati s obzirom da je identičnog izgleda kao i *Precision-Recall* krivulja.

Ono što najbolje dočarava neuspjeh ovoga modela je matrica zabune koja je prikazana na slici ispod (Slika 35).



Slika 35 - Prikaz matrice zabune za More50 model

Matrica je u slučaju ovog modela poprilično jasna, ali i zanimljiva za vidjeti.

Model je u 64% slučajeva točno prepoznao vozilo, dok je u preostalih 36% slučajeva vozilo prepoznao kao pozadinu. Kod klase pješaka nije imao nijedno točno predviđanje, odnosno onog jednog pješaka koji je „zalutao“ na neku od slika model je prepoznao kao pozadinu.

Najzanimljivije je to da niti pri lažnim predikcijama pozadine model nije u nijednom slučaju pomislio da je pozadina zapravo pješak, već je u 100% slučajeva pri pogrešnim predikcijama pozadinu prepoznao kao vozilo.

5.5. Objedinjavanje rezultata i načini poboljšavanja modela

Nakon prikazivanja detaljnih rezultata treninga i validacije svakog pojedinog modela, poželjno je paralelno prikazati njihove završne validacijske rezultate u svrhu lakše usporedbe uspješnosti modela i donošenja zaključaka.

U donjoj tablici prikazani su rezultati završnih validacija za sva tri modela (Tablica 5).

Tablica 5 - Zajednički prikaz rezultata završnih validacija svih modela u kategoriji modela za detekciju objekata u ovisnosti o njihovoj udaljenosti

	Precision	***Recall***	***mAP_0.5***	***mAP_0.5:0.95***
Less10	0.9090	0.7720	0.8600	0.6930
10to50	0.8840	0.7960	0.8700	0.6000
More50	0.8410	0.2950	0.3190	0.1660

Iz tablice je vidljivo da su *Less10* i *10to50* modeli opravdali očekivanja, te se u tim slučajevima CARLA simulator pokazao kao kvalitetan način prikupljanja podataka. Ono što je zanimljivo za izdvojiti kod ta dva modela je to što je *mAP_0.5:0.95* postigao gotovo 10% bolje rezultate kod *Less10* modela nego kod *10to50* modela. To se može objasniti na način da je *bounding boxeve* oko objekata ipak znatno lakše precizno iscrtati što su objekti bliže kameri, a kako se ta daljina povećava tako i mogućnost modela za precizno iscrtavanje *bounding boxeva* opada.

Osim povećanjem broja slika i dubine mreže, metrike na *Less10* modelu mogle bi se dodatno poboljšati i drugačijim pozicioniranjem kamere na vozilu unutar CARLA simulatora u svrhu izbjegavanja „blokade“ kamere od strane ostalih vozila na izrazito malim udaljenostima.

10to50 model mogao bi se poboljšati smanjivanjem raspona udaljenosti unutar same kategorije, odnosno dijeljenjem navedene kategorije na više manjih raspona udaljenosti. Drugim riječima, *10to50* model bi se mogao npr. podijeliti na dva manja *10to30* i *31to50* modela s obzirom da je u trenutnom slučaju objekte na 15-ak metara udaljenosti znatno lakše točno detektirati i klasificirati nego one koji se nalaze na gornjoj granici udaljenosti modela, odnosno na udaljenostima većima od 40 metara. Upravo taj veliki raspon

udaljenosti objekata je zakočio ovaj model da ne ode sa svojim mjerama na vrlo visoke razine točnosti koje bi iznosile preko 90%.

CARLA simulator se pokazao kao loš način prikupljanja podataka u svrhu treniranja uspješnosti detekcije objekata u prometu na udaljenostima većima od 50 metara. Niti proširenje skupa podataka putem CARLE, niti dublja mreža ne bi pomogli u postizanju zadovoljavajućih razina točnosti za eventualnu primjenu u treniranju kamera i senzora u stvarnim autonomnim vozilima.

Moglo bi se pomisliti da bi se More50 model mogao primijeniti barem za treniranje detekcije vozila na autocestama gdje nema prisutnosti pješaka, ali *mAP_0.5* od svega 64% na klasi vozila također je nedovoljna za ikakvu ozbiljniju primjenu.

6. Detekcija dinamičkih objekata prema vremenskim uvjetima

6.1. Objašnjenje problema, prilagodba podataka i obilježja kategorija

Ideja pri treniranju modela u ovoj kategoriji bila je ispitati uspješnost detekcije objekata pri različitim vremenskim uvjetima, odnosno kako i koliko različiti vremenski uvjeti utječu na uspješnost detekcije.

Skupovi podataka za modele vremenskih uvjeta su unutar CARLA simulatora prikupljeni i nazvani po engleskim nazivima za doba dana. Tako su modeli unutar ove kategorije sljedećih naziva: *Morning*, *Noon*, *Afternoon*, *Almost Nigh*t i *Night*.

Svih pet skupova podataka kreirano je iz početnog skupa podataka, kao što je bio slučaj i kod modela za detekciju objekata prema njihovoj udaljenosti.

Za razvrstavanje slikovnih i *labels* datoteka također se koristila *Python* skripta koja je čitala vrijednost tj. doba dana koja su zapisana u svakoj datoteci unutar „*weather*“ direktorija koji se nalazi u sklopu početnog skupu podataka. Pošto su sve *weather* datoteke istih naziva kao i njihove odgovarajuće *labels* i slikovne datoteke, skripta je iste automatski razvrstavala u pravovaljane nove direktorije - ovisno o dobu dana zapisanom u pojedinoj *weather* datoteci. Ukoliko je u pojedinoj *weather* datoteci pisala vrijednost „*morning*“, *label* i slikovnu datoteku istoga naziva skripta je kopirala u direktorij namijenjen za „*Morning*“ skup podataka. Skripta je na taj način čitala sve *weather* datoteke i pritom automatski kopirala sve odgovarajuće *labels* i slikovne datoteke u njihove prikladne direktorije tj. podskupove podataka. Skripta je priložena i detaljnije objašnjena unutar poglavlja „DODATAK“ na kraju ovoga rada.

Osim po količini svjetlosti i kutu sunca s obzirom na različita doba dana, u svakoj od kategorija vrijede i drugačije postavke vremenskih prilika. Unutar svake od kategorija postavljene su različite postavke naoblake, oborina, oborinskih naslaga te intenziteta vjetra.

Sve postavke vremenskih prilika na slikama svakog od kreiranih modela prikazane su u tablici (Tablica 6).

Tablica 6 - Prikaz vremenskih postavki svih modela u kategoriji

	Naoblaka	Oborine	Oborinske naslage	Intenzitet vjetra	Kut sunca
Morning	20	90	30	30	30
Noon	30	0	60	30	80
Afternoon	-1	-1	-1	-1	-1
Almost Night	30	30	0	30	-40
Night	50	0	40	30	-60

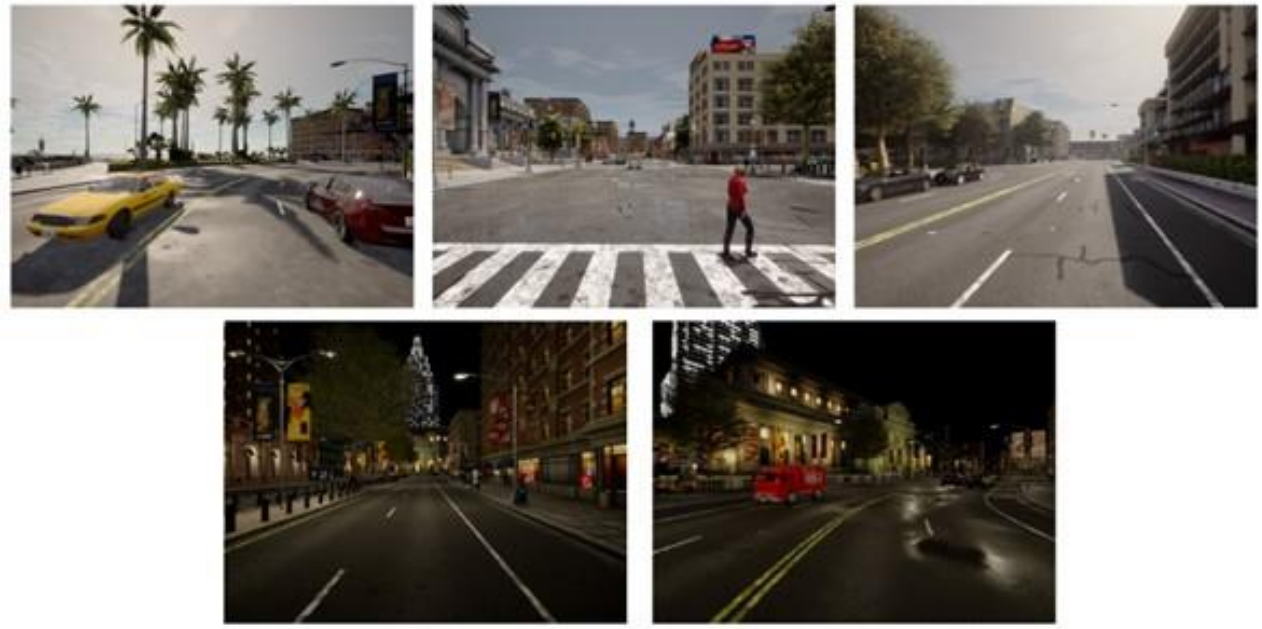
Sve vrijednosti vremenskih postavki kreću se u rasponu od 0 do 100, osim vrijednosti kuta sunca koja je u rasponu od -90 do 90. Što su vrijednosti veće to je veća jačina i količina navedene vremenske prilike odnosno neprilike.

Iz tablice je vidljivo da su sve postavke *Afternoon* modela postavljene na -1. To označava neku vrstu „isključenja“ svih vremenskih postavki, odnosno predstavlja idealne vremenske uvjete koji nemaju nikakvog utjecaja na okolinu ni odvijanje prometa.

CARLA kao simulator ne mora funkcionirati u svakoj situaciji niti prilikom prilagodbe svih svojih parametara savršeno, niti po nekoj općoj ljudskoj logici. Tako je npr. količina naoblake kod jutarnjih postavki najniža od svih preostalih modela (ako izuzmemo Afternoon model), a pritom je količina oborina daleko najveća. Takvo nešto nije nemoguće i u stvarnim uvjetima, ali je rijetkost i malo je vjerojatno da će takvi uvjeti stvarno i vladati. Ipak, u slučaju modela u ovoj kategoriji ta formalna točnost nije toliko bitna, koliko je bitno da su uvjeti vidljivosti i stanja prometnica što raznolikiji kako bi se što vjernije ispitala razlika u uspješnosti detekcije pri različitim vremenskim uvjetima.

Procesi treniranja i validacije odvijaju se identično kao i kod modela za detekciju objekata u ovisnosti o njihovoj udaljenosti. Vrste rezultata koje YOLOv5 algoritam dostavlja po završetku treninga također su identične kao i kod tih modela.

Na primjeru po jedne od slika iz svakog od kreiranih modela mogu se vidjeti razlike u vidljivosti i vremenskim uvjetima između modela (Slika 36).



Slika 36 - Primjerci slika iz svakog pojedinog modela, Morning (gore lijevo), Noon (gore sredina), Afternoon (gore desno), Almost Night (dolje lijevo) i Night (dolje desno)

6.2. Morning model

Jutra su u CARLI podosta mokra i vlažna po pitanju oborina i uvjeta na cesti što je vidljivo iz prethodno prikazane tablice. Usprkos tome, dnevno svjetlo je podosta jako, pa bi i vidljivost u prometu trebala biti dobra. Ipak, koliko će uspješna biti razina detekcije u jutarnjim uvjetima najbolje se vidi iz tablice metrika (Tablica 7).

Tablica 7 - Prikaz rezultata početne i najbolje epohe, te završne validacije Morning modela

	Početna epoha	Najbolja epoha	Završna validacija
Train/box_loss	0.0854	0.0211	/
Train/obj_loss	0.0405	0.0156	/
Train/cls_loss	0.0129	0.0004	/
Precision	0.3258	0.8758	0.8760
Recall	0.4521	0.7243	0.7220
mAP_0.5	0.3223	0.8030	0.8030
mAP_0.5:0.95	0.1067	0.5041	0.5040
Val/box_loss	0.0675	0.0287	/
Val/obj_loss	0.0240	0.0204	/
Val/cls_loss	0.0047	0.0018	/
lrX	0.0703	0.0024	/

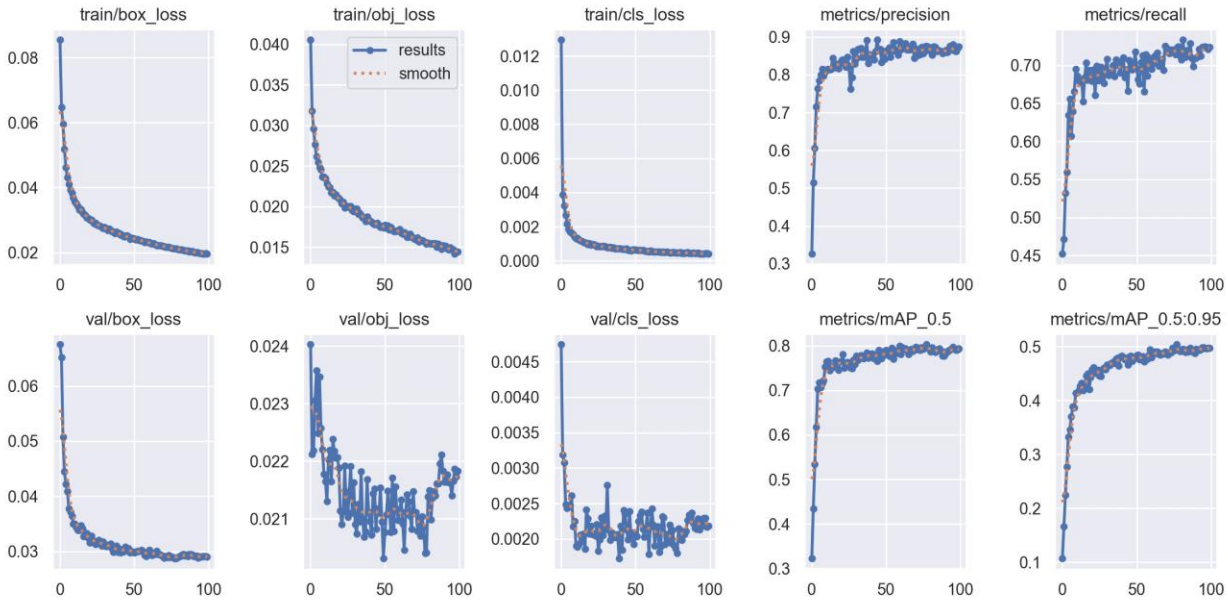
Sve vrste mjerenih gubitaka doživjele su pad, te se kreću u rasponu od 0.1% do otprilike 2% vrijednosti što je utjecajno zanemariva razina gubitka.

Precision i *mAP_0.5* metrike su u početnoj epohi imale slične vrijednosti, te su upravo to vrijednosti koje su doživjele najznačajniji rast kroz trening. Rezultati obje metrike su visoki i zadovoljavajući, te im se vrijednosti podudaraju u najboljoj epohi treninga i unutar završne validacije, što je pozitivan pokazatelj kvalitete odrađenog treninga.

Recall i *mAP_0.5:0.95* su postigle nešto manje vrijednosti, tj. vrijednosno su ispod očekivanja. Za *mAP_0.5:0.95* to i nije toliko čudno niti zabrinjavajuće s obzirom na njena stroga pravila. Ono što je bitno je da je prethodno navedeni *mAP_0.5* postigao zadovoljavajuće vrijednosti od preko 80%.

Ono što blago zabrinjava je vrijednost *recall* metrike u najboljoj epohi i završnoj validaciji. Iako 72% uspješnih detekcija nije neuspjeh, za slučaj modela koji sadrži samo dvije klase to je ipak prenizak rezultat. Uzroci nižih *recall* vrijednosti trenutno se mogu samo nagađati, konkretniji zaključak moći će se donijeti tek po pregledu svih grafova unutar modela, te usporedbom sa rezultatima *recall* metrike iz preostalih modela u kategoriji.

Graf praćenja metrika za vrijeme treniranja modela prikazan je na slici ispod (Slika 37).



Slika 37 Prikaz kretanja vrijednosti svih metrika po epohama kroz proces treniranja Morning modela

Svi trening gubitci imaju pravilan i „čist“ pad cijelim svojim tokom, također to uključuje i *val/box_loss*, dok kod *val/obj_lossa* i *val/cls_lossa* vidimo znatnu količinu šuma i skokove u vrijednostima.

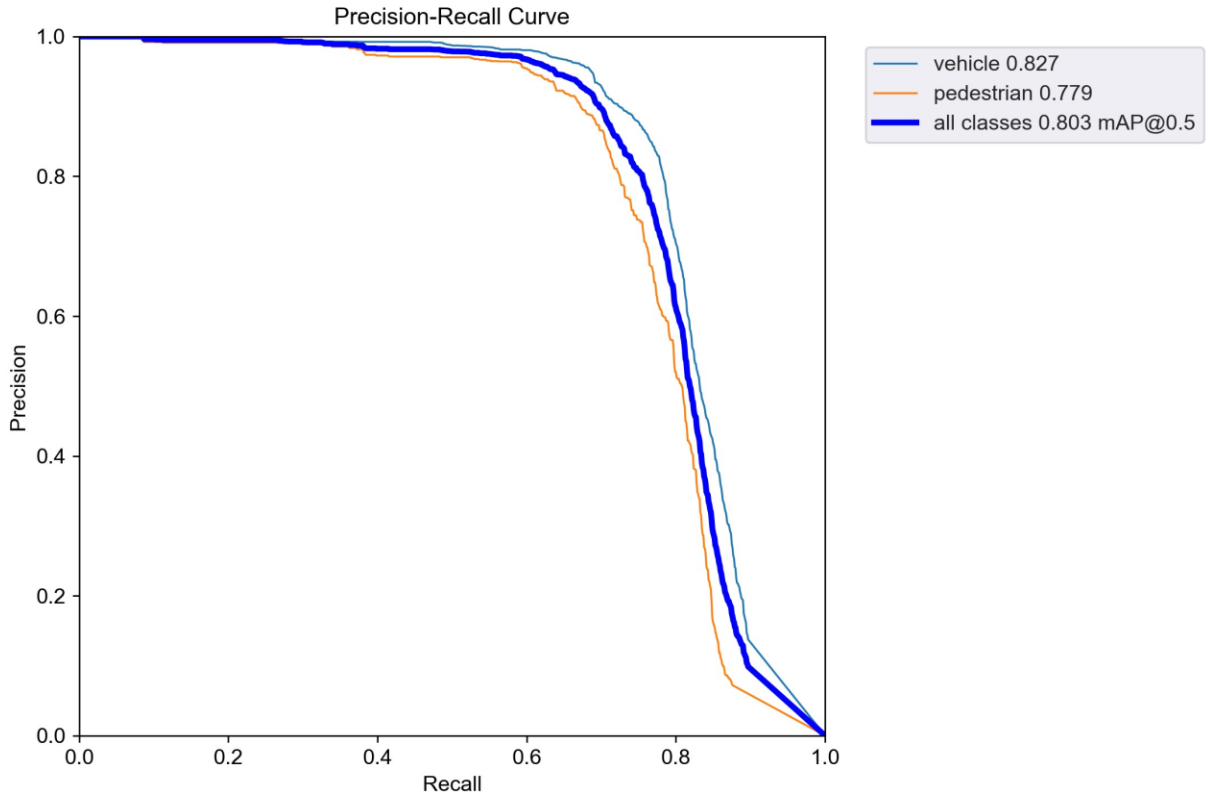
Kod *val/obj_lossa* taj šum je znatno izraženiji, ali krivulja gubitka svejedno ima tendenciju pada, a skokovi u kojima vrijednosti variraju su u rasponu od 0.003, što bi izraženo u postotku značilo raspon skokova od 0.3%. Kod *val/cls_lossa* taj šum i raspon vrijednosti još je i manji. Jasno je stoga da sa tako niskim vrijednostima gubitaka i niskim rasponima skokova vrijednosti, isti nemaju prevelikog utjecaja na metrike *precisiona*, *recalla* ili *mAP-a* unutar validacijskog skupa podataka, odnosno nisu direktan uzročnik nešto niže vrijednosti *recalla* što bi se prilikom prvog pogleda na graf moglo pomisliti.

Da su vrijednosti ta dva gubitka znatno veće i da se pritom događaju takvi šumovi, to bi već bio ozbiljan problem koji bi znatno utjecao i na druge rezultate pri validaciji, te bi bio pokazatelj vjerojatne pretreniranosti modela.

U ovom slučaju to se može pripisati disbalansu broja instanci klasa unutar validacijskog skupa podataka gdje je omjer broja vozila naprema broju pješaka otprilike 4:1. Također, validacijski skup na relativno malom broju slika sadrži veliku količinu vozila i pješaka koji su prikazani na tim slikama, što može dovesti do preklapanja bounding boxeva i zbunjivanja modela na malom uzorku.

Vrijednosti *precisiona*, *recalla* i oba *mAP*-a imaju pravilnu tendenciju rasta sa nešto većom razinom šuma na prikazu *recall* metrike što dodatno ukazuje na određeni problem pri ispravnom određivanju kojoj klasi predviđeni objekt pripada.

Graf odnosa *precisiona* i *recalla* prikazan je na slici ispod (Slika 38).

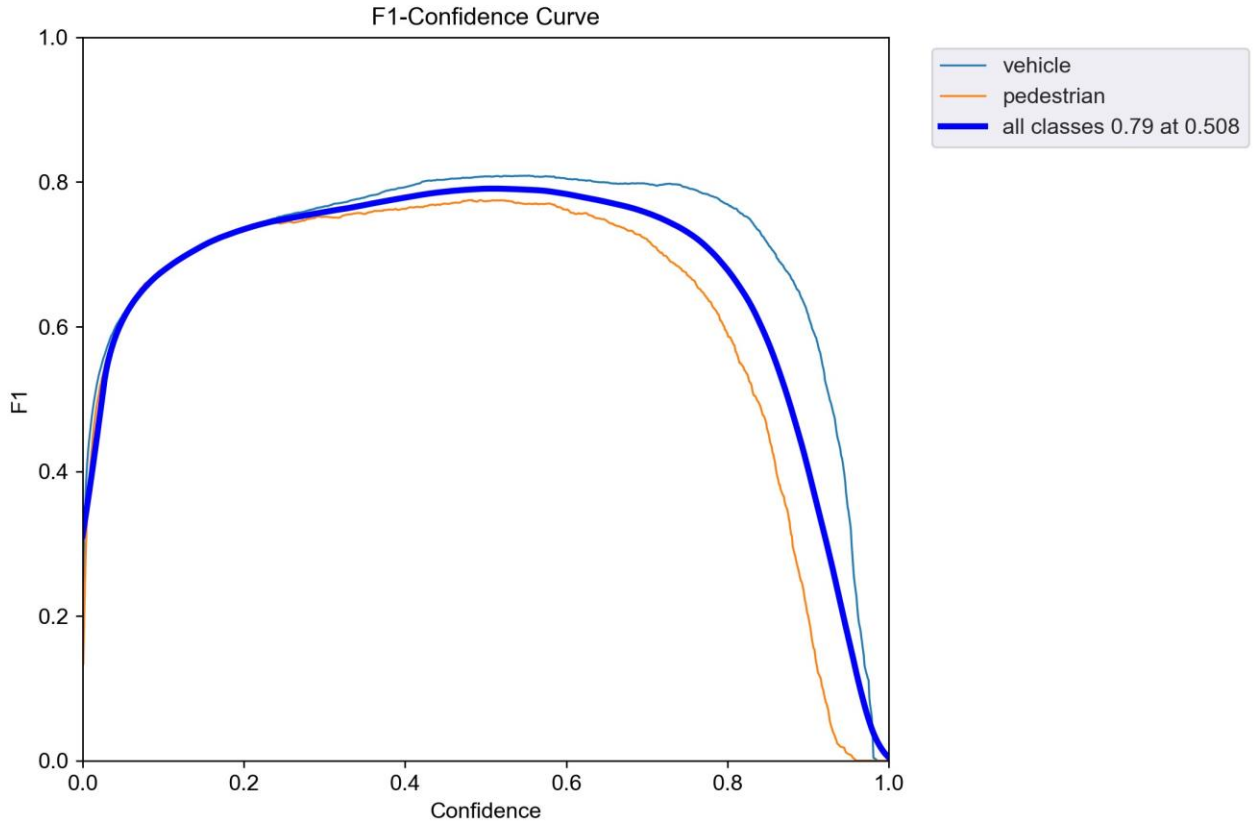


Slika 38 - Prikaz odnosa precisiona i recalla u Morning modelu

Usprkos nešto nižim vrijednostima *recalla*, ovo je jedan od „uglađenijih“ grafova međuodnosa *precisiona i recalla* od svih do sada promatranih modela. Najveći *mAP_0.5* klase vozila iznosi 82.7%, dok je kod pješaka to približno 78%. Očekivano, vozila se nešto

preciznije i točnije klasificiraju od pješaka, ali ta razlika u točnosti njihovih detekcija nije toliko izražena kao što je to znao biti slučaj u modelima za detekciju objekata u ovisnosti o njihovoj udaljenosti. Kada se sagledavaju obje klase, najveći *mAP_0.5* model doseže kada je *precision* metrika na oko 87% točnosti, a tada je *recall* metrika na oko 72% točnosti što odgovara i brojkama iz početne tablice. U tim trenucima *mAP_0.5* iznosi oko 80% što se također podudara sa tabličnim podacima.

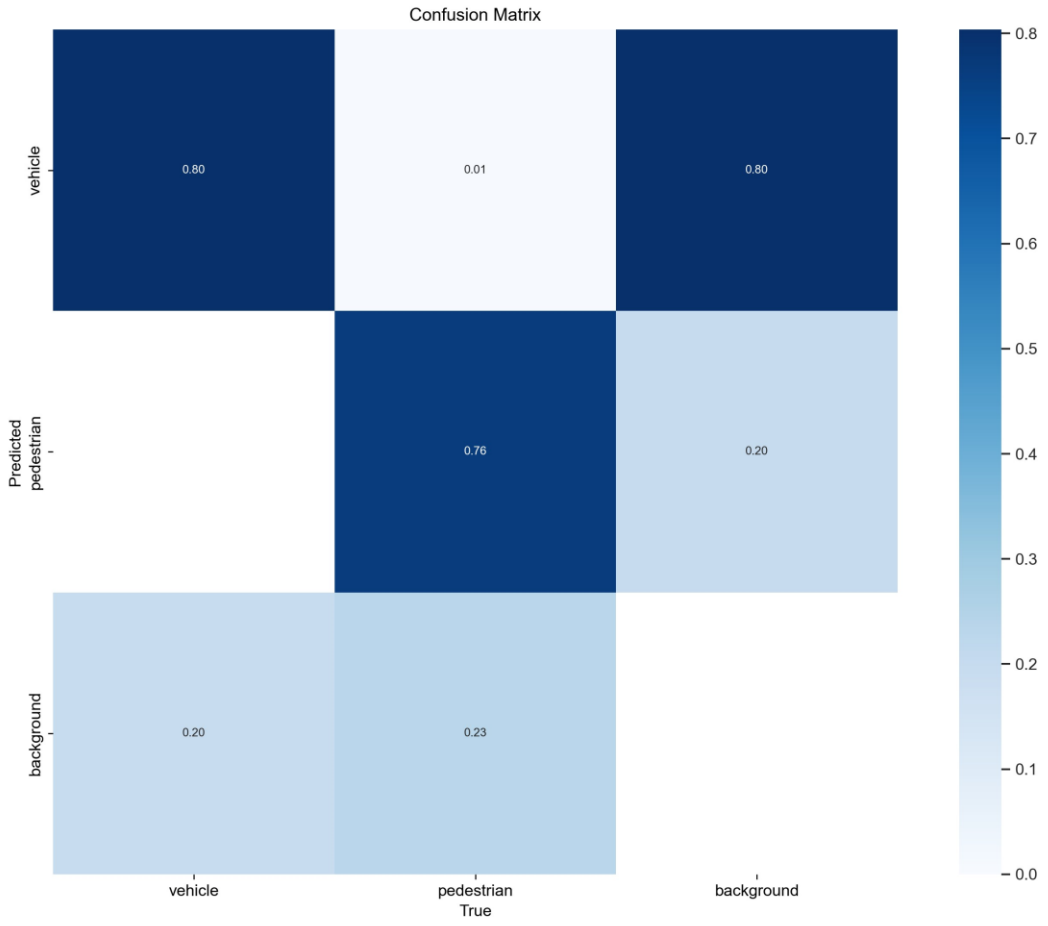
Graf F1 mjere prikazan na sljedećoj slici nema tako savršen luk krivulje kao u nekim prethodnim modelima, ali postiže gotovo savršen harmonijski balans između *precisiona* i *recalla* pri razini samopouzdanja od 0.508 (Slika 39). Drugim riječima - graf u trenutku najboljeg harmonijskog omjera za obje klase ne naginje previše niti na stranu *precisiona,* niti na stranu *recalla*.



Slika 39 - F1 graf Morning modela

Matrica zabune ovoga modela izrazito je zanimljiva za promatranje iz razloga što uvodi svojevrsnu „novinu“ u do sada promatrane modele (Slika 40).

Navedena novina vidljiva je u drugom stupcu matrice koji predstavlja klasifikacije klase pješaka. Naime u slučaju ovoga modela 1% ukupnog broja pješaka pogrešno je klasificirano kao vozilo, što se u nijednom dosadašnjem modelu nije dogodilo. Usprkos toj krivoj klasifikaciji pješaka kao vozila, model je točno klasificirao pješake u 76% slučajeva što odgovara i podacima sa prethodnih F1 i *Precision-Recall* grafova.



Slika 40 - Matrica zabune „Morning“ modela

U prvom stupcu matrice vidljivo je da je klasa vozila zadržala visoku 80% točnost klasifikacije, dok su u 20% slučajeva vozila zamijenjena za pozadinu.

U trećem stupcu matrice već se može primijetiti uzorak da model uvijek puno više lažno prepoznaje pozadinu kao vozilo nego kao pješaka. Kako u prethodnim modelima, tako i u ovome - model je krivo prepoznao pozadinu kao vozilo u oko 80% slučajeva, dok je u preostalih 20% slučajeva pozadinu zamijenio za pješaka.

6.3. Noon model

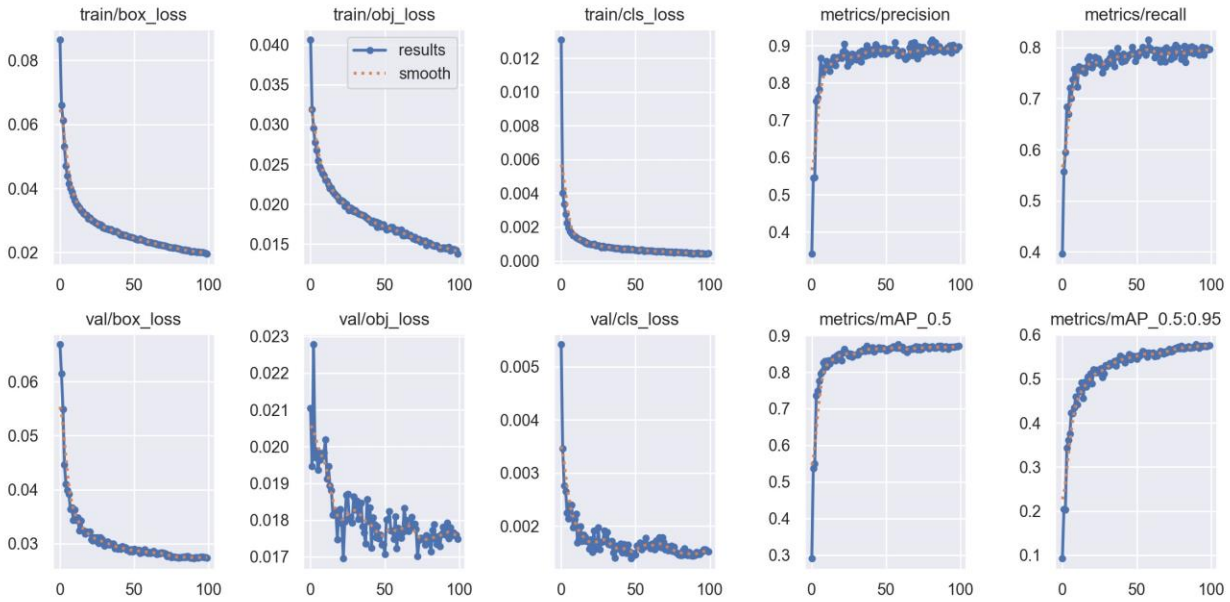
Na slikama *Noon* modela sunce je pod najvišim kutem od svih pet modela unutar ove kategorije što ujedno znači i najveće osvjetljenje scene. Oborine nisu prisutne, ali su oborinske naslage kao i naoblaka nešto većih vrijednosti nego kod *Morning* modela. Koliko je to sve utjecalo na rezultate vidljivo je iz tablice u nastavku (Tablica 8).

Tablica 8 - Prikaz rezultata početne i najbolje epohe, te završne validacije Noon modela

	Početna epoha	Najbolja epoha	Završna validacija
Train/box_loss	0.0864	0.0204	/
Train/obj_loss	0.0406	0.0144	/
Train/cls_loss	0.0130	0.0004	/
Precision	0.3415	0.8907	0.8870
Recall	0.3960	0.8020	0.7950
mAP_0.5	0.2905	0.8710	0.8700
mAP_0.5:0.95	0.0925	0.5704	0.5780
Val/box_loss	0.0668	0.0275	/
Val/obj_loss	0.0210	0.0173	/
Val/cls_loss	0.0054	0.0015	/
lrX	0.0702	0.0015	/

Svi gubitci unutar treninga i privremenih validacija su u padu, a *precision* i *mAP_0.5* metrike su slične rezultatima iz *Morning* modela. Veliki pomak nastao je u *recall* i *mAP_0.5:0.95* metrikama koje su obje porasle za približno 8% u usporedbi sa *Morning* modelom. Na taj način *recall* je probio granicu od 80% točnosti što se može nazvati nekakvim imaginarnim ciljem i standardom kojem težimo u rezultatima tri najbitnije metrike, odnosno za vrijednosti *precision*, *recall* i *mAP_0.5* metrike.

Graf metrika iz gornje tablice prikazan je na slici ispod (Slika 41).



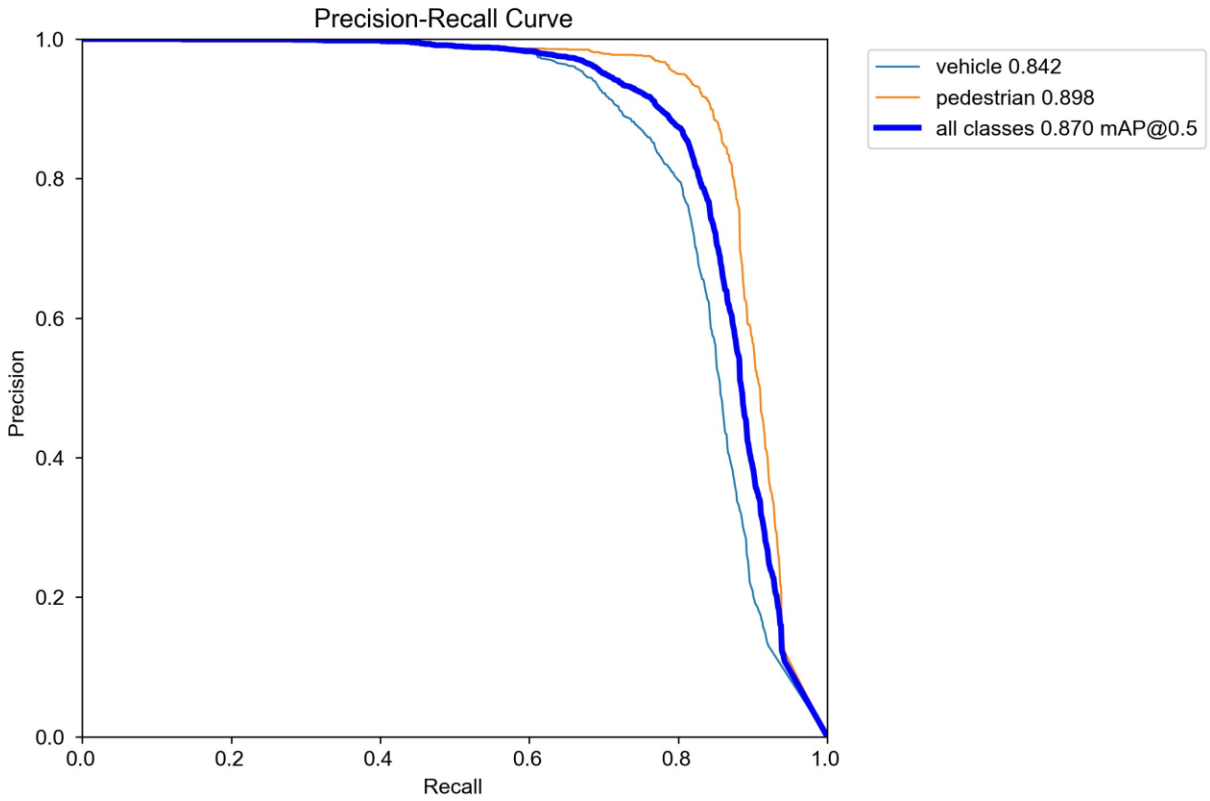
Slika 41 - Prikaz kretanja vrijednosti svih metrika po epohama kroz proces treniranja Noon modela

Osim prema brojevnim rezultatima, grafički je također ovaj model znatno „pravilniji“ od *Morning* modela.

To se ponajprije vidi prema grafovima za *val/obj_loss* i *val/cls_loss*, gdje je unutar *val/obj_loss* grafa tendencija pada puno pravilnija, te je šum znatno manji nego kod *Morning* modela, dok je u *val/cls_loss* grafu šum gotovo potpuno eliminiran.

Graf *recall* metrike također je izgubio znatnu količinu šuma, što opravdava i prethodno prikazan rast točnosti naspram *Morning* modela.

Na grafu međuodnosa *precision* i *recall* metrika, krivulje koje predstavljaju klase se ne preklapaju u tolikoj mjeri kao što je to bio slučaj kod *Morning* modela, ali ta nesavršenost se opravdava znatno boljim rezultatima svake pojedine klase, a i njihove *mAP_0.5* vrijednosti (Slika 42).



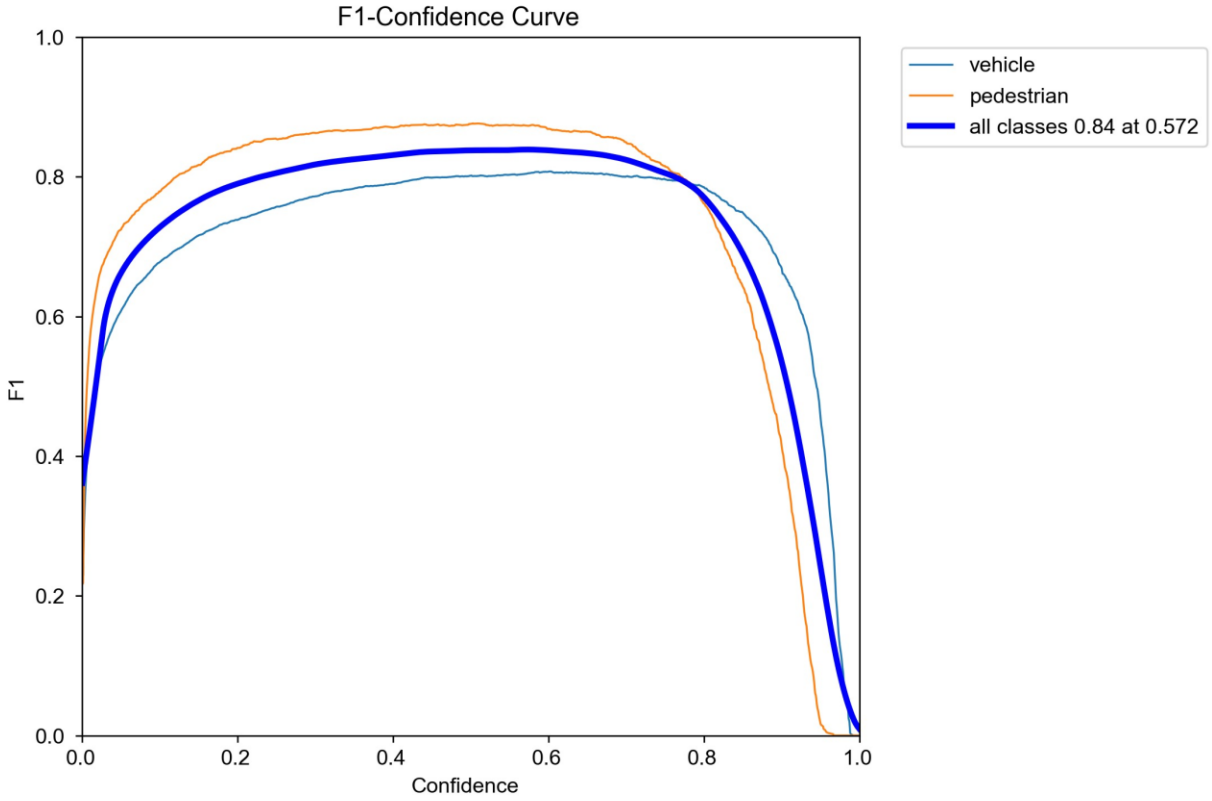
Slika 42 - Prikaz odnosa precisiona i recalla u Noon modelu

Ono što je zanimljivo za primijetiti je nešto veća razina ispravne detekcije pješaka nego vozila. To je do sada bio slučaj samo kod *Less10* modela gdje je to opravdano boljim pregledom staze nego ceste od strane kamere na vozilu.

Iako je ovaj rezultat pozitivan, svejedno se kosi sa do sada stečenom logikom. Razlog zašto se to u ovom slučaju događa može se pronaći ponajviše u položaju sunca, odnosno kutu pada sunčevih zraka na scenu. Naime, sunce ovdje znatno jače obasjava cestu nego stazu, ali pritom to stvara kontraefekat na detekciju vozila, pošto se često stvara odsjaj koji zasljepljuje kameru i otežava jasnu detekciju objekata koji se kreću po cesti, što su najčešće upravo vozila.

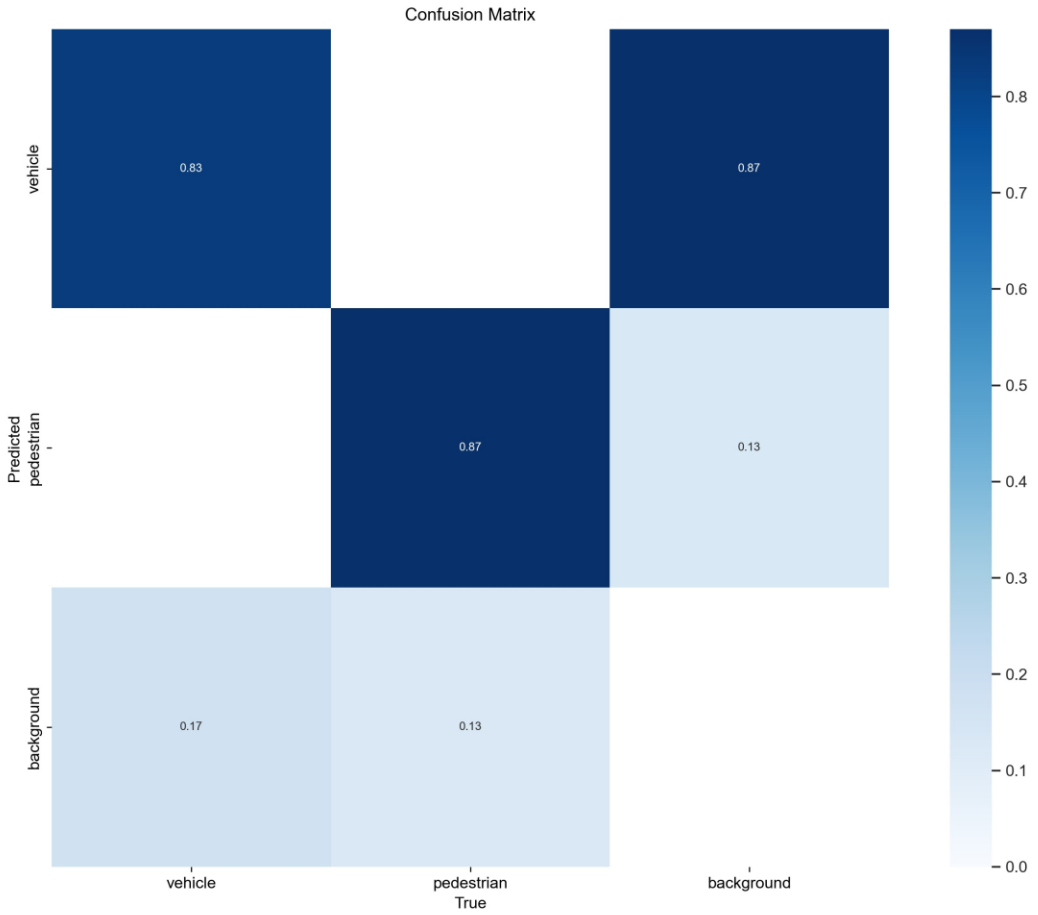
Usprkos tome, rezultati su odlični - vozila se uspješno prepoznaju u otprilike 84% slučajeva, dok se pješaci uspješno prepoznaju u gotovo 90% slučajeva. Prosjek *mAP_0.5* detekcije obje klase stoga iznosi 87%.

Iz F1 grafa može se isčitati da model najbolju harmonijsku sredinu postiže u trenucima kada je *recall* metrika u nešto boljem položaju od *precision* metrike, pošto se to događa pri iznosu samopouzdanja od 0.572 označavajući nešto veće naginjanje modela ka većoj *recall* vrijednosti što je u našim modelima i poželjno (Slika 43).



Slika 43 - F1 graf Noon modela

Na matrici zabune ovoga puta nema nikakvih „iznenađenja“. Prikazane su očekivane razine točnih klasifikacija koje su se mogle isčitati i iz prethodnih grafova (Slika 44).



Slika 44 - Matrica zabune Noon modela

6.4. Afternoon model

Kako je ovo model sa „idealnim“ vremenskim postavkama, odnosno postavkama koje ne bi trebale utjecati na kvalitetu detekcije objekata i odvijanja prometa, ovo je ujedno i model gdje se očekuju najbolji rezultati. Stoga je za očekivati da će rezultati biti slični *Noon* modelu, čak možda i malo bolji.

Kakvi su stvarni rezultati nakon odrađenog treninga i validacije ovoga modela prikazano je u tablici u nastavku (Tablica 9).

Tablica 9 - Prikaz rezultata početne i najbolje epohe, te završne validacije Afternoon modela

	Početna epoha	Najbolja epoha	Završna validacija
Train/box_loss	0.0862	0.0196	/
Train/obj_loss	0.0371	0.0133	/
Train/cls_loss	0.0135	0.0005	/
Precision	0.4058	0.9029	0.9090
Recall	0.4598	0.8024	0.7880
mAP_0.5	0.3709	0.8716	0.8740
mAP_0.5:0.95	0.1395	0.5730	0.5780
Val/box_loss	0.0595	0.0259	/
Val/obj_loss	0.0175	0.0139	/
Val/cls_loss	0.0045	0.0006	/
lrX	0.0702	0.0010	/

Iz tablice je vidljivo da su rezultati vrlo slični rezultatima *Noon* modela.

Razlike svih dobivenih rezultata i unutar tablice i na iscrtanim grafovima kreću se u rasponu od otprilike 1%. Iz navedenog razloga grafovi ovoga modela neće se prikazati, pošto su poput brojčanih podataka gotovo identični onima već prikazanima za *Noon* model.

Kako se uvjeti na slikama unutar *Noon* modela mogu smatrati gotovo pa savršenim uvjetima, ovakvi gotovo identični rezultati modela u kojem bi „neutralne“ vremenske prilike trebale pogodovati neometanom odvijanju prometa nisu iznenađujući.

6.5. Almost Night model

Ovo je prvi model u ovoj kategoriji u kojem se može očekivati nešto znatniji pad i razlika u rezultatima praćenih metrika pošto je ovo prvi model gdje kut sunca ima negativnu vrijednost, odnosno razina osvjetljenja je znatno niža i zavisi ponajviše o gradskom osvjetljenju i svjetlima automobila. Je li se taj rezultatski pad modela stvarno i dogodio najlakše je provjeriti putem tablice rezultata (Tablica 10).

Tablica 10 - Prikaz rezultata početne i najbolje epohe, te završne validacije Almost Night modela

	Početna epoha	Najbolja epoha	Završna validacija
Train/box_loss	0.0878	0.0213	/
Train/obj_loss	0.0380	0.0140	/
Train/cls_loss	0.0139	0.0004	/
Precision	0.3259	0.8629	0.8260
Recall	0.3348	0.7233	0.7590
mAP_0.5	0.2319	0.8066	0.8180
mAP_0.5:0.95	0.0679	0.4947	0.4970
Val/box_loss	0.0699	0.0306	/
Val/obj_loss	0.0169	0.0170	/
Val/cls_loss	0.0055	0.0013	/
lrX	0.0702	0.0016	/

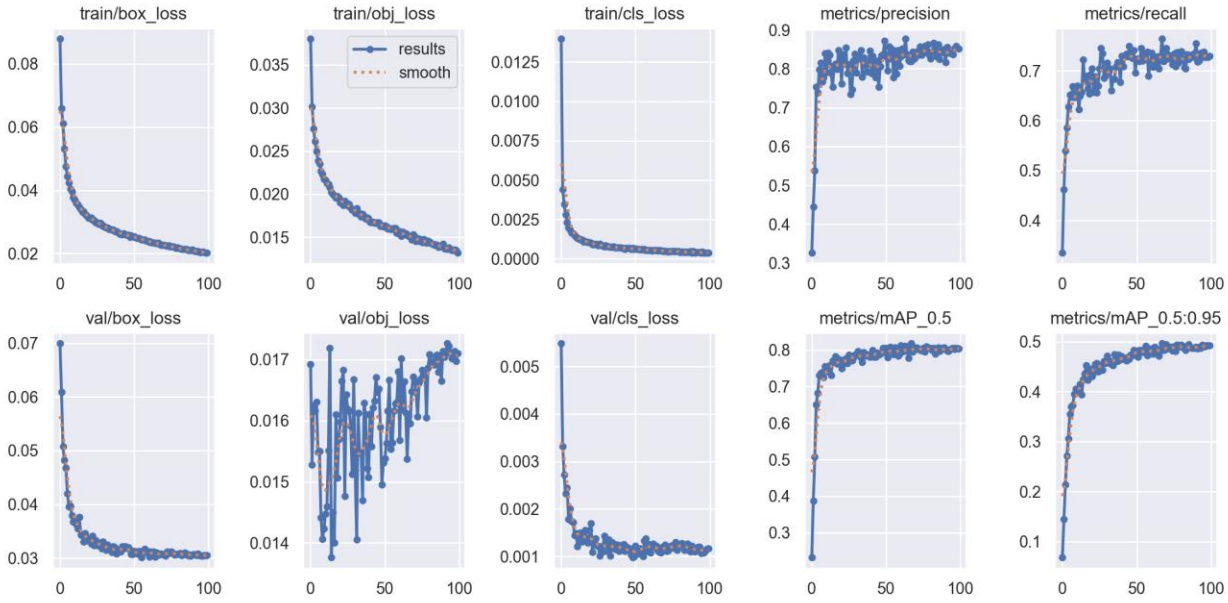
Prikazani rezultati zanimljivi su iz više razloga, ponajprije zato što se prvi put događa porast neke od komponenata funkcije gubitka, a to je u ovom slučaju *val/obj_loss*. Iako je taj porast zapravo više stagnacija vrijednosti početne epohe pošto isti iznosi svega +0.0001, ipak strogo matematički gledano to je porast vrijednosti. Kako se ta vrijednost zapravo kretala kroz epohe vidjeti će se najbolje na kasnijem grafičkom prikazu.

Puno važnije i veće razlike dogodile su se kod evaluacijskih mjera koje su rezultatski dosta slične onima u „*Morning*“ modelu, ali uz jednu bitnu razliku, a to je da rezultati završne validacije odstupaju od najbolje epohe znatno više nego što je to bio slučaj u bilo kojem prethodnom modelu.

Razlog takvih odstupanja vrijednosti između najbolje epohe i završne validacije najvjerojatnije se krije u različitoj količini osvjetljenja na slikama trening i validacijskog skupa podataka. Naime, pošto je trening skup slika četiri puta veći od validacijskog skupa, u slikama trening skupa vozilo prolazi kroz puno više različitih tipova i količina različitih osvjetljenja scene tj. grada, nego što je to slučaj kod validacijskog skupa gdje je vozilo prilikom prikupljanja slika moglo proći isključivo kroz jedan ili dva različita tipa osvjetljenja što može uzrokovati nešto bolje ili lošije rezultate prilikom validacije. Kod preostalih

modela se to nije moglo dogoditi pošto je svjetlost dolazila prvenstveno od Sunca, te se na taj način ikakav tip svjetlosnog disbalansa izbjegao.

Kako su se vrijednosti metrika kretale kroz proces treninga prikazano je na objedinjenom grafičkom prikazu (Slika 45).

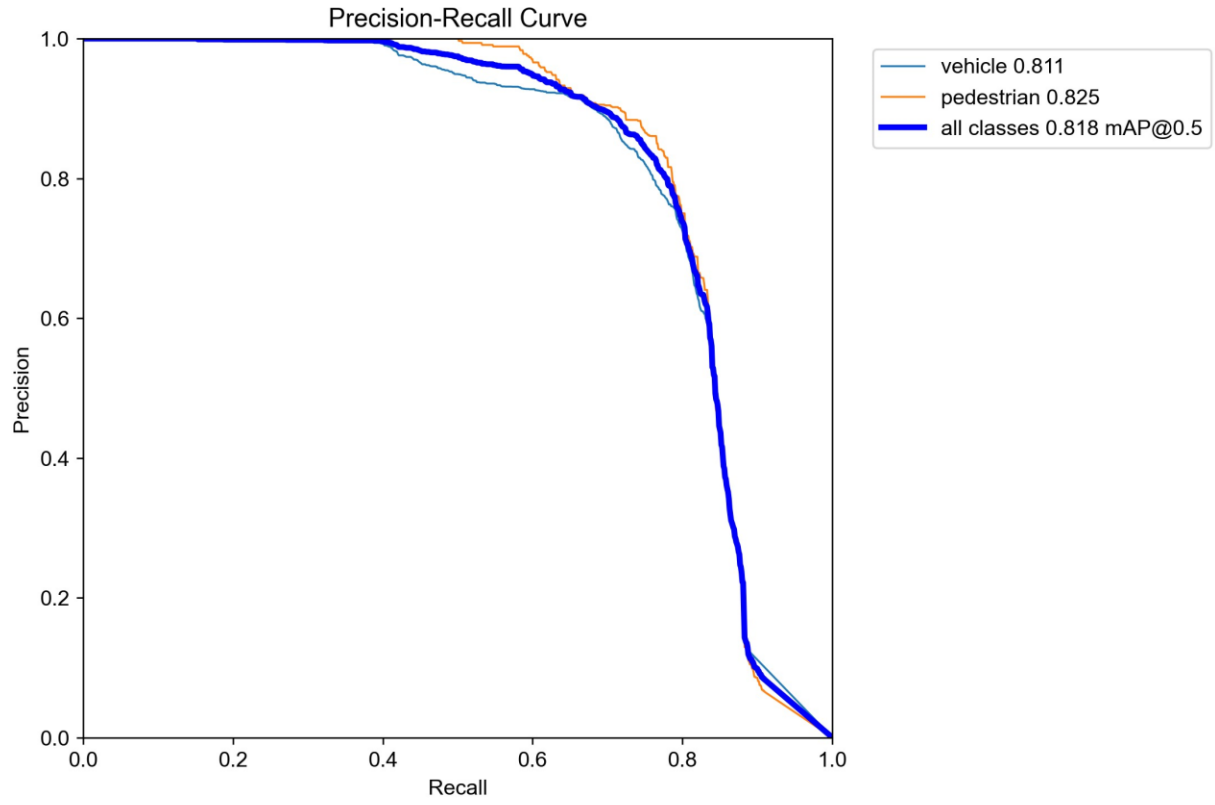


Slika 45 - Prikaz kretanja vrijednosti svih metrika po epohama kroz proces treniranja Almost Night modela

Graf potvrđuje ono što je primjećeno u tabličnom prikazu, a to je da *val/obj_loss* ima tendenciju rasta u procesu treninga. Iako ovo nije poželjan znak, ponovo treba naglasiti to da je ta vrijednost zapravo izrazito mala i nema pretjeranog utjecaja na završne rezultate, barem ne pri ovako niskim vrijednostima. Ono što je zanimljivo na *val/obj_loss* grafu je prisutnost velike količine šuma, a pošto je to graf koji je vezan uz validacijski skup podataka tu možemo naći i povezanost sa prethodno navedenim razlikama u osvjetljenju na podacima validacijskog skupa.

Sve ostale metrike imaju ispravnu tendenciju rasta i pada uz ponešto veći šum na *precision* i *recall* grafovima, što može objasniti i razliku u rezultatima unutar završne validacije, s obzirom da je i u procesu treniranja dolazilo do povremenih većih odstupanja u vrijednostima između epoha.

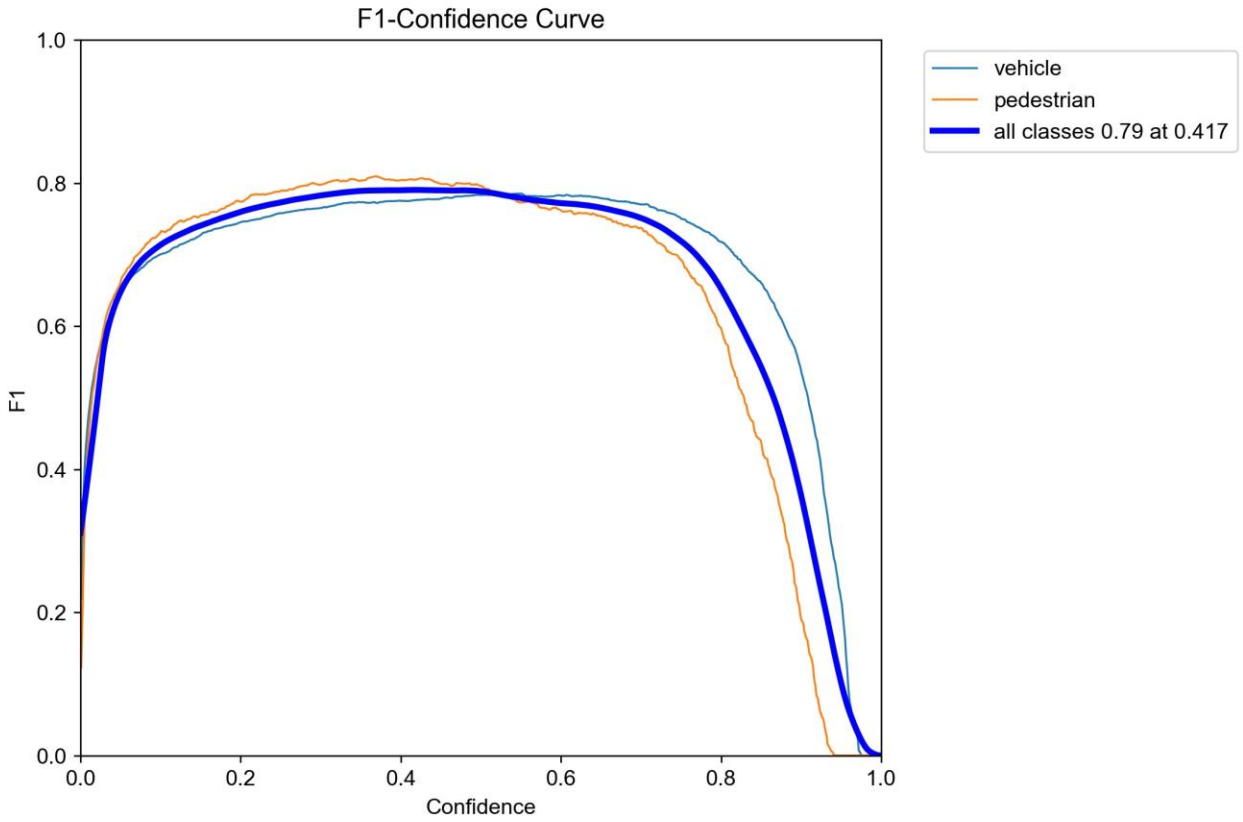
Na *Precision-Recall* grafu vidljiv je gotovo savršen međuodnos *mAP-a* između klasa, s obzirom da se krivulje gotovo u potpunosti preklapaju (Slika 46).



Slika 46 - Prikaz odnosa precisiona i recalla u Almost Night modelu

Klase vozila i pješaka kreću se otprilike na istim vrijednostima najvećeg *mAP_0.5*, što pri njihovoj aritmetičkoj sredini daje najviši *mAP_0.5* od 81,8% koji se događa pri vrijednosti *precisiona* od oko 86% i vrijednosti *recalla* od oko 72% što odgovara i podacima iz tablice.

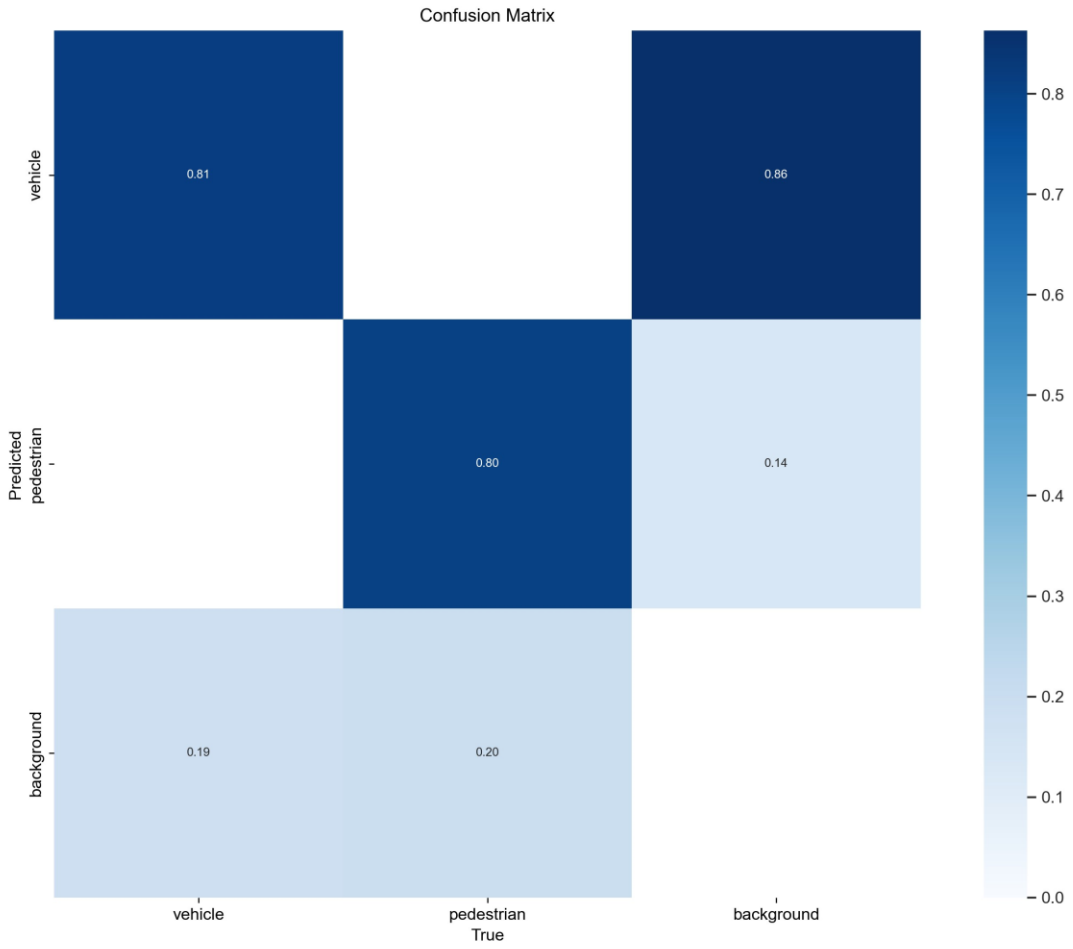
F1 graf ukazuje na to da su *precision* i *recall* metrike u najboljoj harmoniji kada razina samopouzdanja iznosi 0.417 što govori da je model više orijentiran prema preciznosti klasifikacije nego prema točnoj predikciji objekata (Slika 47).



Slika 47 - F1 graf „Almost night“ modela

Kada razina samopouzdanja iznosi 0.417, klase dosežu najbolju F1 vrijednost od 79%.

Matrica zabune blago se kosi sa prikazima iz prethodnih grafova gdje su pješaci imali nešto veću najvišu vrijednost točnih predviđanja od vozila. Ipak, pošto matrica zabune uzima prosjek vrijednosti iz cijeloga modela, a ne samo u najboljem slučaju - to i nije toliko čudno pošto se i iz prethodnih grafova može vidjeti da je kroz cijeli trening skup razina točnih predviđanja vozila kroz veći dio treninga ipak bila nešto viša od one za klasu pješaka. Matrica zabune prikazana je na slici ispod (Slika 48).



Slika 48 - Matrica zabune Almost Night modela

Iz matrice je vidljivo da razina točnih predviđanja klase vozila iznosi 81%, dok kod klase pješaka iznosi 80%. Shodno tome vozila su u 19% slučajeva zamijenjena za pozadinu, dok se kod pješaka to dogodilo u 20% slučajeva.

Pogrešna predviđanja pozadine kao vozila i pješaka zadržala su očekivanu razinu od 86% pogrešnih predviđanja pozadine kao klase vozila te 14% pogrešnih predviđanja pozadine kao klase pješaka.

6.6. Night model

Night model ima nešto veću razinu naoblake nego *Almost Night* model, oborina nema, ali zato ima oborinskih naslaga koje mogu stvoriti odsjaj na cesti i dodatno otežati kvalitetu detekcije objekata. Kut sunca još je niži, čime je osvjetljenje još slabije i oslanja se u potpunosti na gradsku rasvjetu i svjetla automobila.

Kako sve navedene vremenske postavke utječu na same rezultate vidljivo je iz tablice (Tablica 11).

Tablica 11 - Prikaz rezultata početne i najbolje epohe, te završne validacije Night modela

	Početna epoha	**Najbolja epoha**	**Završna validacija**
Train/box_loss	0.0879	0.0205	/
Train/obj_loss	0.0355	0.0127	/
Train/cls_loss	0.0144	0.0004	/
Precision	0.3768	0.8828	0.8820
Recall	0.3528	0.6735	0.6740
mAP_0.5	0.2684	0.7636	0.7640
mAP_0.5:0.95	0.0879	0.4836	0.4840
Val/box_loss	0.0662	0.0308	/
Val/obj_loss	0.0226	0.0206	/
Val/cls_loss	0.0052	0.0018	/
lrX	0.0703	0.0017	/

Sve komponente funkcije gubitka imale su tendenciju pada, uključujući i *val/obj_loss* koja se pokazala kao najnepravilnija komponenta u sklopu modela ove kategorije. Iako je ona po rezultatima iz tablice doživjela blagi pad, taj pad je toliko neznatan da se vjerojatno mogu ponovo očekivati velike količine šuma i skokova u njenim vrijednostima kroz procese privremenih validacija, ali to će se najbolje vidjeti na kasnijem grafu.

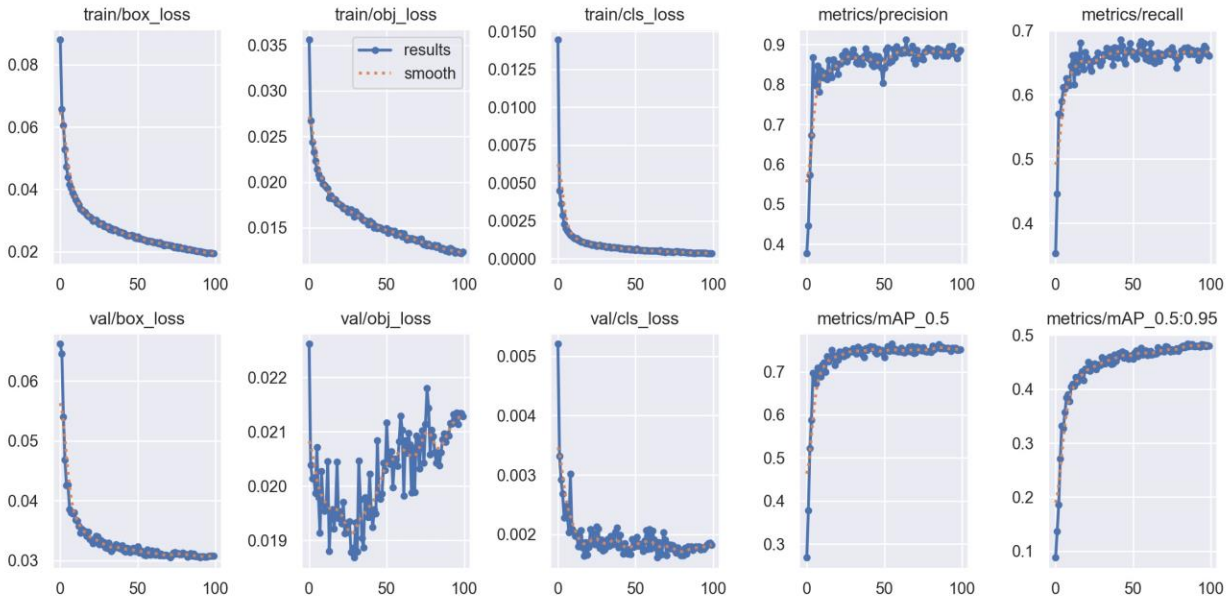
Precision metrika zadržala je svoju visoku razinu točnosti kao i u preostalim modelima, dok su sve ostale evaluacijske mjere doživjele znatan pad spram ostalih modela. Tako

recall ne doseže niti 70%-nu točnost, već se kreće na točnosti od oko 67.3%. Obje vrste *mAP* metrika također imaju do sada najniže vrijednosti u ovoj kategoriji modela.

Ipak, ne može se reći da je model neiskoristiv i da je pokazao neefikasnost CARLA simulatora za prikupljanje podataka u ovom slučaju, kao što je to npr. bio slučaj kod *More50* modela iz prethodne kategorije.

Vrijednosti jesu nešto niže nego u dosadašnjim modelima u ovoj kategoriji, ali s obzirom na postavljene vremenske uvjete i potpuno odsutstvo sunčeve svjetlosti to je bilo i očekivano.

Graf praćenja svih metrika kroz proces treniranja i privremenih validacija prikazan je na slici ispod (Slika 49).



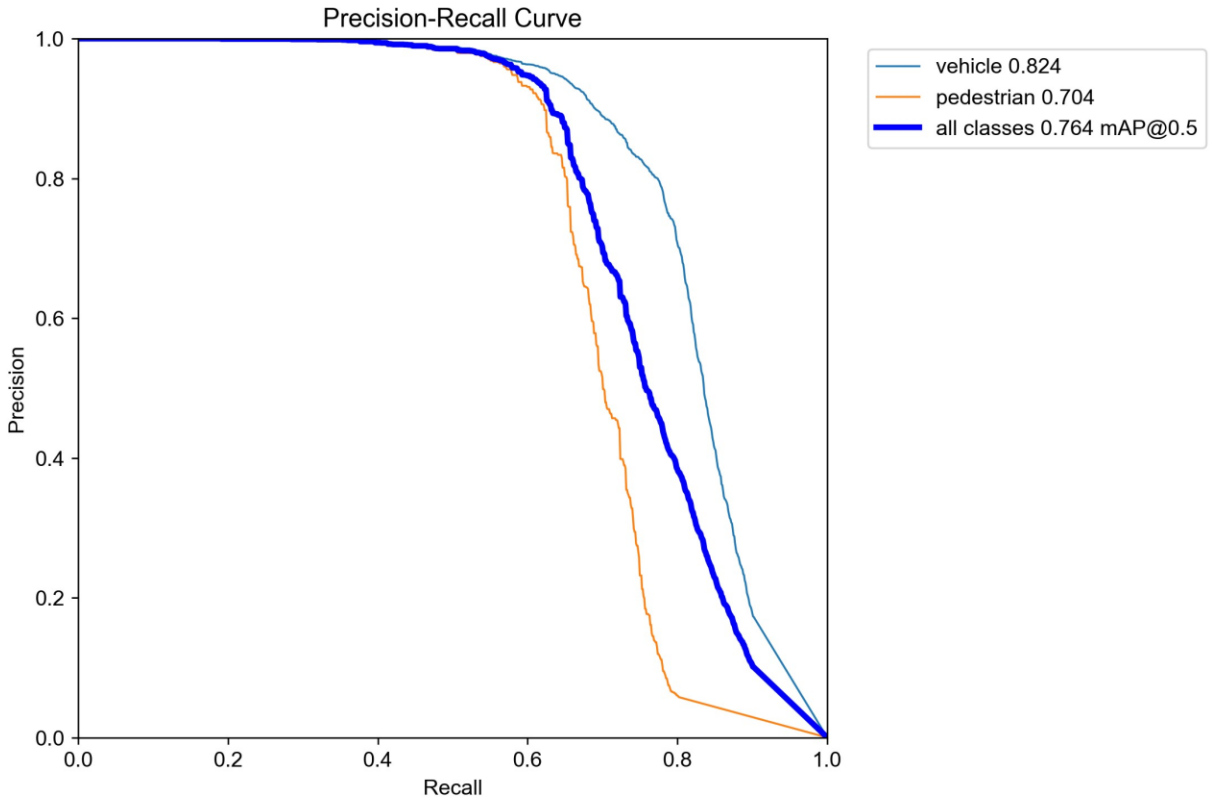
Slika 49 - Prikaz kretanja vrijednosti svih metrika po epohama kroz proces treniranja Night modela

Prema očekivanju *val/obj_loss* je ponovo imao velike količine šuma i skokova, te iako je u početku doživio znatan pad, vidljivo je da prema kraju treniranja modela ima tendenciju rasta što bi u dužem treniranju na većem skupu podataka i pri njegovim većim vrijednostima predstavljalo problem i značilo vjerojatnu pretreniranost modela. Ipak, šum i rast u ovom modelu su ipak znatno manje izraženi nego kod prošlog *„Almost Night“*

modela što je svakako pozitivan pomak. U slučaju ovoga modela, s obzirom da su vrijednosti *val/obj_lossa* i dalje izrazito male, to se može pripisati kao i kod prethodnog modela nesrazmjeru slika i instanci klasa unutar trening i validacijskog skupa, te različitim periodičkim kvalitetama osvjetljenja u slikama validacijskog skupa.

Sve ostale mjere su pravilnog oblika i tendencija rasta, odnosno pada.

Graf odnosa *precisiona* i *recalla* najzanimljiviji je za promatranje od svih do sada prikazanih modela u ovoj kategoriji (Slika 50).



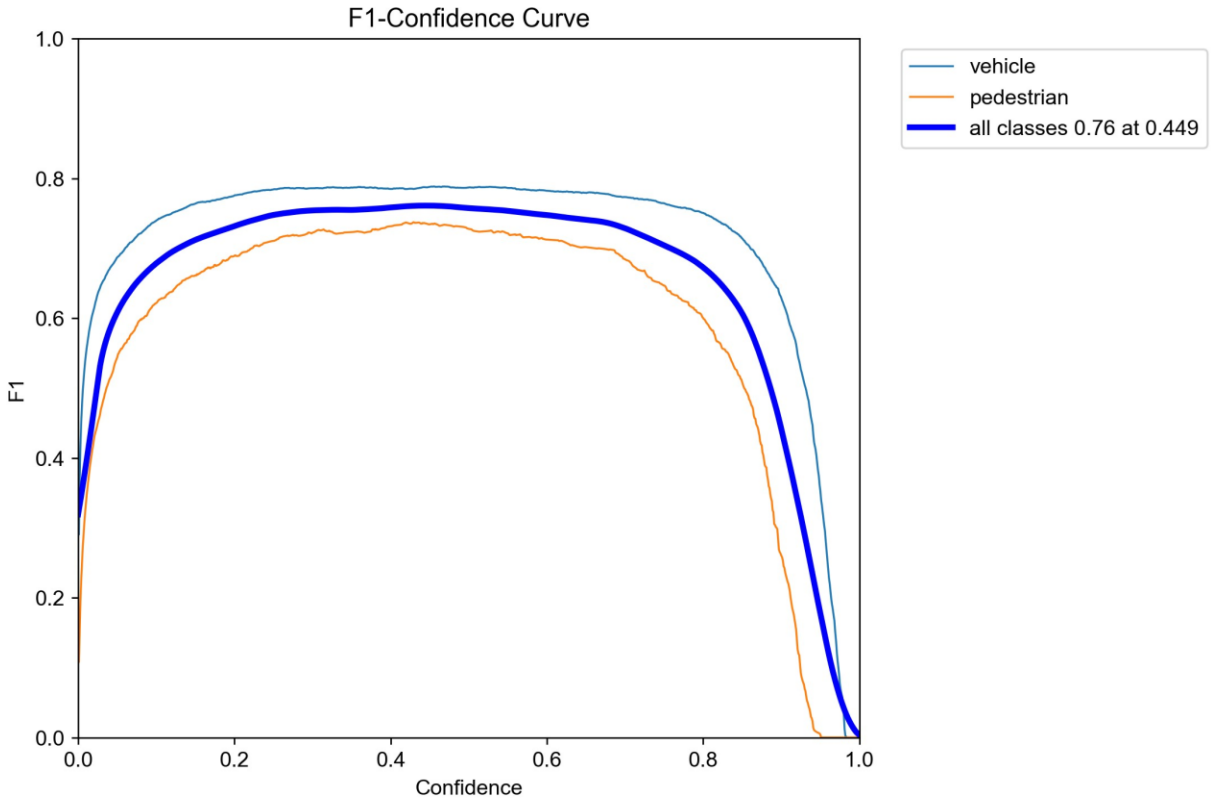
Slika 50 - Prikaz odnosa precisiona i recalla u Night modelu

Iz grafa je vidljivo da su vozila znatno bolje prepoznata od pješaka, odnosno da su upravo pješaci glavni uzrok nešto nižih vrijednosti *recall* i *mAP* mjera. To je i logično s obzirom da vozila imaju vlastiti izvor osvjetljenja u vidu prednjih i stražnjih svjetala, dok vidljivost pješaka isključivo ovisi o tome jesu li na osvjetljenom dijelu ulice ili su u tami. Također,

vidljivost pješaka može ovisiti i o njihovoj odjeći i obući, ali u slučaju podataka prikupljenih CARLA simulatorom taj utjecaj je zanemariv.

Vozila tako postižu najvišu uspješnost detekcije i klasifikacije u 82.4% slučajeva, dok se kod pješaka to događa u samo 70.4% slučajeva. Najviša razina *mAP_0.5* iznosi 76.4% što je i aritmetička sredina ove dvije klase, te se podudara sa podacima iz prethodno prikazane tablice.

F1 graf ima iznenađujuće pravilan i „čist“ oblik krivulja s obzirom na sve do sada prikazane rezultate (Slika 51).

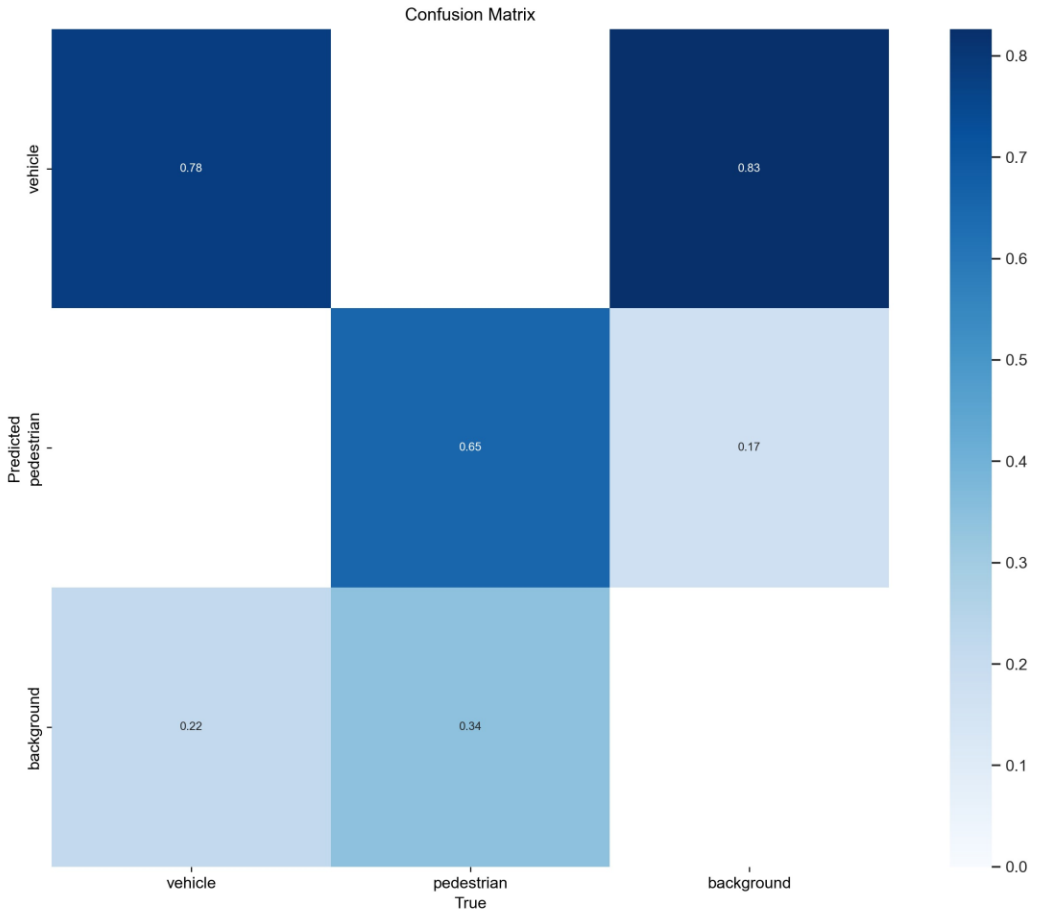


Slika 51 - F1 graf Night modela

Svoju najveću harmoniju *precision* i *recall* metrike dosežu kada razina samopouzdanja iznosi 0.449 što označava blagu naklonost grafa *precision* metrici.

Na F1 grafu jasno se može vidjeti kako je u cijelom tijeku treninga klasa vozila postizala znatno bolje vrijednosti od klase pješaka što dodatno potvrđuje prikaz vrijednosti sa *Precision-Recall* grafa.

Matrica zabune je zanimljiva s obzirom da je na njoj još znatno niža vrijednost točnih klasifikacija za obje klase (Slika 52).



Slika 52 - Matrica zabune Night modela

Iako je prethodno utvrđeno da je najviša razina točnih predviđanja vozila iznosila 82,4%, ovdje vidimo da je u cijelom procesu treninga ukupan broj točnih klasifikacija iznosio tek 78%, dok je u preostalih 22% slučajeva model klasificirao vozila kao pozadinu.

Kod klase pješaka situacija je još lošija s obzirom da je model kroz cijeli proces treninga uspješno klasificirao pješake samo u 65% slučajeva, što je za oko 5% lošije od

maksimalne razine točnih detekcija klase pješaka koja je prikazana na *Precision-Recall* grafu.

Lažne predikcije pozadine ostale su pri svojim uobičajenim vrijednostima.

6.7. Objedinjavanje rezultata i načini poboljšavanja modela

Kako bi se što bolje utvrdila i usporedila stvarna uspješnost modela unutar ove kategorije, potrebno je rezultate njihovih završnih validacija prikazati unutar jedne tablice (Tablica 12).

Tablica 12 - Zajednički prikaz rezultata završnih validacija svih modela u kategoriji modela za detekciju objekata u ovisnosti o vremenskim uvjetima

	Precision	Recall	mAP_0.5	mAP_0.5:0.95
Morning	0.8760	0.7220	0.8030	0.5040
Noon	0.8870	0.7950	0.8700	0.5780
Afternoon	0.9090	0.7880	0.8740	0.5780
Almost night	0.8260	0.7590	0.8180	0.4970
Night	0.8820	0.6740	0.7640	0.4840

Vidljivo je da je *precision* metrika zadržala visok postotak točnosti kroz svih pet modela što po pitanju kvalitete klasifikacije prepoznatih objekata opravdava korištenje CARLA simulatora za prikupljanje podataka u svrhu ovih tipova treninga.

mAP_0.5 metrika također je na zadovoljavajućoj razini kod svih pet modela, iako bi za stvarnu upotrebu trebala biti znatno veća, ali za ovu veličinu skupova podataka i veličinu YOLOv5 mreže koja je korištena - ovi rezultati su obećavajući. Druga *mAP* metrika, tj. *mAP_0.5:0.95* postigla je znatno lošije rezultate od *mAP_0.5* metrike, pa čak i od *mAP_0.5:0.95* metrika iz prve kategorije modela koji su trenirani prema udaljenostima objekata. Ipak, ova metrika se zbog svojih strogih *IoU* vrijednosti rijetko uzima kao relevantna, osim u modelima gdje je bitna visoka preciznost iscrtavanja *bounding boxeva* što u ovim modelima nije slučaj, već je sasvim dovoljna i nešto blaža *mAP_0.5* metrika.

Ono što je ustvari glavni uzročnik nešto nižih *mAP* vrijednosti je relativno niska vrijednost *recalla* u svih pet treniranih modela. Nešto niže vrijednosti *recalla* mogu se objasniti time

što su ovi modeli gledali objekte na svim udaljenostima, odnosno nisu se ograničavali samo na bliske objekte ili objekte srednjih udaljenosti, a iz prethodne kategorije treniranih modela vidjeli smo da sa porastom udaljenosti objekata opada sposobnost modela za točnost njihove detekcije.

Model koji je postigao najlošije rezultate je prema očekivanjima „*Night*“ model. Iako je preciznost tog modela i dalje na izrazito visokoj razini, vidljivost u CARLA simulatoru pri noćnim uvjetima je preslaba da bi se objekti mogli uvijek uspješno vidjeti, odnosno događa se veliki broj lažnih ili neuspješnih detekcija.

Usprkos nešto lošijim rezultatima *Night* modela, može se reći da je CARLA simulator u potpunosti primjenjiv za treniranje senzora i kamera autonomnih vozila pri različitim vremenskim uvjetima. Ipak, za slučajeve industrijske primjene trebale bi se uraditi brojne prilagodbe kako bi modeli mogli postići zadovoljavajuće rezultate.

Neke od tih prilagodbi odnose se na sam CARLA simulator koji bi trebao nadograditi svoju kvalitetu prikaza noćnih uvjeta u smislu poboljšanog gradskog osvjetljenja, ponajviše staza po kojima se pješaci najviše kreću.

Ostale bitne prilagodbe tiču se prikupljanja samih podataka za procese treniranja i validacije. Naime, prilikom kreiranja skupa podataka za potrebe ovakvog treninga trebalo bi se ručnim podešavanjima bolje izbalansirati odnos broja vozila i pješaka unutar grada iz kojeg se prikupljaju podaci, odnosno trebalo bi kreirati puno više pješaka kako bi se smanjio nesrazmjer među klasama, te bi se tako i kvaliteta njihove detekcije i klasifikacije podigla na višu razinu. Skup podataka bi također trebao biti znatno veći, te bi se trebala koristiti najveća dostupna veličina YOLO mreže u svrhu postizanja što boljih rezultata.

7. Zaključak

U današnje vrijeme autonomna vozila su još uvijek u svojim razvojnim povojima, no neupitno je da će ista ući u masovnu proizvodnju i uporabu u godinama koje dolaze.

Najveća kočnica njihovoj masovnoj proizvodnji i uporabi leži u visokim sigurnosnim standardima koji se moraju zadovoljiti prije nego bi navedena vozila smjela prometovati na stvarnim prometnicama.

Kako bi se njihova integracija u našu svakodnevicu olakšala i ubrzala, potrebno je razvijati i ulagati u različite tipove simulatora autonomne vožnje u svrhu ispitivanja i postizanja najviših mogućih razina sigurnosti senzora i sustava unutar autonomnih vozila.

Uvijek treba sagledavati širu sliku i razmišljati na koje sve načine možemo provesti korisna istraživanja i treniranja sustava uz primjerene alate. Dva takva istraživanja provedena su i u ovom radu gdje se CARLA simulator pokazao kao izvrstan način prikupljanja podataka za potrebe detekcije dinamičkih objekata u prometu u ovisnosti o njihovoj udaljenosti i vremenskim uvjetima.

Cilj ovoga rada nije bilo postizanje razina rezultata koji bi odgovarali industrijskoj primjeni, već ispitati da li bi CARLA simulator sa prikladnim hardverskim, vremenskim i financijskim sredstvima bio kvalitetan način prikupljanja podataka u svrhu treniranja sustava autonomnih vozila. Sveučilište u Barceloni stoga može biti ponosno na svoj simulator, s obzirom da se CARLA pokazala kao kvalitetan i jednostavan način prikupljanja podataka za potrebe treniranja sustava unutar autonomnih vozila.

Istina, CARLA simulator nije podoban za treniranje vozila za vožnju pri velikim brzinama na autocestama, pošto je detekcija objekata na udaljenostima većima od 50 metara izrazito loša, ali za testove umjerene gradske vožnje uz poštivanje prometnih propisa CARLA se pokazala kao u potpunosti primjenjiv i kvalitetan način prikupljanja podataka.

Ne treba zaboraviti ni YOLO algoritam koji svojom brzinom i lakoćom primjene, te kvalitetom povratnih informacija i rezultata znatno olakšava izvedbu kvalitetnih i efikasnih treninga i testova nad podacima prikupljenima u svrhu treniranja autonomnih vozila.

Sa puno većim skupovima podataka, te budućim nadogradnjama CARLE i YOLO algoritama, te uz financijske i hardverske mogućnosti kompanija koje se bave razvojem autonomnih vozila - CARLA simulator u kombinaciji sa YOLO algoritmom je zasigurno kvalietna, efikasna i pritom besplatna opcija koja može polučiti vrhunske rezultate pri razvoju željenih sustava unutar autonomnih vozila.

Ipak, sve dok autonomna vozila ne postanu standard i široko rasprostranjena pojava, moramo se i dalje uzdati u naše vozačke sposobnosti te paziti podjednako kako na nas same, tako i na ostale sudionike u prometu, jer trenutno su naša osjetila sluha i vida još uvijek najbolji i najtočniji senzori.

Popis korištene literature

Članci:

1. Ilková, V. ; Ilka, A. (2017) "Legal Aspects of Autonomous Vehicles – an Overview". Proceedings of the 2017 21st International Conference on Process Control (PC), Štrbské Pleso, Slovakia, June 6 – 9, pp. 428-433.
2. Xujein Wu; Guangming W.; Nachuan S. (2023) „Research on Obstacle Avoidance Optimiziation and Path Planning of Autonomous Vehicles Based on Attention Mechanism Combined with Multimodal Information Decision-making Thoughts of Robots“., Neurorobot, vol 17-2023
3. Dosovitskiy A.; German R.; Codevilla F.; Lopez A.; Koltun V. (2017) „CARLA: An Open Urban Drivin Simulator“. 1st Conference on Robot Learning (CoRL 2017), Mountain View, United States
4. Wu, Chien-Chung, Yu-Cheng Wu, and Yu-Kai Liang. 2023. "The Development of an Autonomous Vehicle Training and Verification System for the Purpose of Teaching Experiments" Electronics 12, no. 8: 1874.
5. Leudet, J., Mikkonen, T., Christophe, F., Männistö, T. (2018). Virtual Environment for Training Autonomous Vehicles. In: Giuliani, M., Assaf, T., Giannaccini, M. (eds) Towards Autonomous Robotic Systems. TAROS 2018. Lecture Notes in Computer Science(), vol 10965. Springer, Cham.
6. Peiyuan J.; Daji E.; Fangyao L.; Ying C.; Bo M. (2022) „A Review of YOLO Algorithm Developments“. Procedia Computer Science, Volume 199-2022, str. 1066-1073
7. Redmon J.; Divvala S.; Girshick R.; Farhadi A.; (2016) „You Only Look Once: Unified, Real-Time Object Detection“, University of Washington, Allen Institute for AI, Facebook AI Research
8. He, K., Zhang, X., Ren, S., & Sun, J. (2015) „Spatial pyramid pooling in deep convolutional networks for visual recognition“. IEEE transactions on pattern analysis and machine intelligence, 37(9), 1904–1916.
9. Park S. (2021) „A guide to Two-stage Object Detection: R-CNN, FPN, Mask R-CNN“. CodeX magazine

10. Chien-Yao W.; I-Hau Y.; Hong-Yuan M.; (2021) „You Only Learn One Representation: Unified Network for Multiple Tasks“, Institute of Information Science, Academia Sinica, Taiwan
11. WANG, Chien-Yao, LIAO, Hong-Yuan Mark, WU, Yueh-Hua, et al. 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. p. 390-391.
12. HE, Kaiming, ZHANG, Xiangyu, REN, Shaoqing, et al. Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE transactions on pattern analysis and machine intelligence, 2015, vol. 37, no 9, p. 1904-1916.
13. Xu, Renjie & Lin, Haifeng & Lu, Kangjie & Cao, Lin & Liu, Yunfei. (2021). A Forest Fire Detection System Based on Ensemble Learning. Forests. 12. 217. 10.3390/f12020217.
14. Hansen, Mark & Oparaeke, Alphonsus & Gallagher, Ryan & Karimi, Amir & Tariq, Fahim & Smith, Melvyn. (2022). Towards Machine Vision for Insect Welfare Monitoring and Behavioural Insights. Frontiers in Veterinary Science. 9. 10.3389/fvets.2022.835529.
15. Padilla, Rafael & Netto, Sergio & da Silva, Eduardo. (2020). A Survey on Performance Metrics for Object-Detection Algorithms. 10.1109/IWSSIP48289.2020.
16. Gupta A.; Anpalagan A.; Guan L.; Khwaja A.S.; (2017) „Deep learning for object detection and scene perception in self-driving cars: Survey, challenges and open issues“. Array-Volume 10, July 2021
17. Liu W.; Wang Z., Liu X.; Zeng N.; Liu Y.; Alsaadi F.; (2017) „A survey of deep neural network arhitectures and their applications“. Neurocomputing-Volume 234, 19 April 2017
18. S. -Y. Xu, X. -M. Chen, Z. -J. Wang, Y. -H. Hu and X. -T. Han, "Decision-Making Models for Autonomous Vehicles at Unsignalized Intersections Based on Deep Reinforcement Learning," 2022 International Conference on Advanced Robotics and Mechatronics (ICARM), Guilin, China, 2022, pp. 672-677, doi: 10.1109/ICARM54641.2022.9959664.

19. Doleron L.; „Detecting objects with YOLOv5, OpenCV, Python and C++“, Published in: Mlearning.ai, January 23,2022
20. Azzedine Boukerche and Zhijun Hou. 2021. Object Detection Using Deep Learning Methods in Traffic Scenarios. ACM Comput.Surv. 54, 2, Article 30 (March 2022), 35 page
21. Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, Silvio Savarese; „Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Rgression“.Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 658-666
22. Amit, Y., Felzenszwalb, P., Girshick, R. (2020). Object Detection. In: Computer Vision. Springer, Cham.
23. T. -H. Wu, T. -W. Wang and Y. -Q. Liu, "Real-Time Vehicle and Distance Detection Based on Improved Yolo v5 Network," 2021 3rd World Symposium on Artificial Intelligence (WSAI), Guangzhou, China, 2021, pp. 24-28, doi: 10.1109/WSAI51899.2021.9486316.
24. Francies M.; Ata M.; Mohamed M.; (2021) „A robust multiclass 3D object recognition based on modern YOLO deep learning algorithms“, Concurrency Computet Pract Exper. 2022, Wiley
25. Verma M and Singh M. (2023). Vehicle Object Detection Using Deep Learning-Based Anchor-Free Detector. Innovations in Computer Science and Engineering. 10.1007/978-981-19-7455-7_34. (455-465).
26. Koech K.; „Object Detection Metrics With Worked Example“, Published in: Towards Data Science, August 26,2020
27. Kanstren T.; „A Look at Precision, Recall and F1-Score“, Published in: Towards Data Science, September 11,2020

Internetski izvori:

28. Github repozitorij modificirane verzije CARLE: https://github.com/AlanNaoto/carla-dataset-runner
29. PyTorch: https://pytorch.org/
30. YOLOv5 repozitorij: https://github.com/ultralytics/yolov5
31. Ultralytics službena stranica: https://docs.ultralytics.com/modes/
32. CARLA simulator: https://carla.org/

Popis slika

Popis tablica

Popis jednadžbi

Popis kratica

YOLO – You Only Look Once

RGB – Red, Green, Blue

DVS – Dynamic Vision Sensor

R-CNN – Region-based Convolutional Neural Network

CSP – Cross Stage Partial

SPP – Spatial Pyramid Pooling

PANet – Path Aggregation Network

SGD – Stohastic Gradient Descent

GioU – Generalised Intersection over Union

IoU – Intersection over Union

mAP – Mean Average Precision

lr – Learning Rate

DODATAK

U sklopu ovog dodatka biti će prikazane i kratko objašnjene sve *Python* skripte koje su se koristile u svrhu prilagodbe podataka za korištenje pri treniranju modela. Uz osnovno objašnjenje, sve skripte objašnjene su i kroz svoje komentare u kodu (linije koje započinju znakom #).

1) ***YOLOv5_label_converter*** skripta služila je za pretvorbu vrijednosti originalnih koordinata unutar labels datoteka prikupljenih putem CARLA simulatora u oblik koji je čitljiv YOLOv5 algoritmu.

```
import os
import shutil

# Funkcija za konvertiranje koordinata bounding boxeva u YOLO format
def convert_coordinates(width, height, x_min, y_min, x_max, y_max):
    x = (x_min + x_max) / 2 / width
    y = (y_min + y_max) / 2 / height
    w = (x_max - x_min) / width
    h = (y_max - y_min) / height
    return x, y, w, h

# Lokacije izvornog i odredisnog direktorija
input_directory = 'D:/Diplomski_neuronske/50m_versions/weather_files_podjel
a/morning/labels'
output_directory = 'D:/Diplomski_neuronske/yolov5/data/labels/morning/train
'

# Kreiranje odredisnog direktorija ukoliko on ne postoji
if not os.path.exists(output_directory):
    os.makedirs(output_directory)

# Dohvacanje liste svih .txt datoteka u izvornom direktoriju
txt_files = [f for f in os.listdir(input_directory) if f.endswith('.txt')]

# Petlja kroz svaki dokument
for txt_file in txt_files:
    # Otvaranje izvornog dokumenta
    with open(os.path.join(input_directory, txt_file), 'r') as file:
        lines = file.readlines()


 # Kreiranje nove liste koja ce pohranjivati konvertirane linije
```

```
    converted_lines = []


# Petlja kroz svaku liniju u input dokumentu
    for line in lines:
        # Split the line into individual values
        label, x_min, y_min, x_max, y_max = line.strip().split(' ')

        # Konvertiranje koordinata bounding boxeva u YOLO format
        converted_coords = convert_coordinates(1024, 768, float(x_min), flo
at(y_min), float(x_max), float(y_max))
        converted_line = f"{label} {converted_coords[0]} {converted_coords[
1]} {converted_coords[2]} {converted_coords[3]}\n"

# Dodavanje konvertirane linije u novu listu
        converted_lines.append(converted_line)


    output_file = os.path.join(output_directory, txt_file)

    # Zapisivanje konvertiranih linija u output file
    with open(output_file, 'w') as file:
        file.writelines(converted_lines)
```

2) ***Picture_copy*** služi za kopiranje svih .png datoteka (slika) iz određenog izvornog direktorija u željeni odredišni direktorij, pod uvjetom da se za svaku sliku u odredišnom direktoriju nalazi odgovarajuća *.txt* datoteka.

```
import os
import shutil

# Kreiranje funkcije
def copy_images_with_txt(source_folder, destination_folder):
    for root, _, files in os.walk(source_folder):
        for file in files:
            if file.lower().endswith('.png'):
                image_path = os.path.join(root, file)
                txt_path = os.path.join(destination_folder, os.path.splitext(file
)[0] + '.txt')

                if os.path.exists(txt_path):
                    shutil.copy2(image_path, destination_folder)
                    print(f"Image {file} copied to {destination_folder}.")

# Postavljanje izvornog i odredisnog direktorija
source_folder = 'D:/Diplomski_neuronske/50m_versions/all_training_images'
destination_folder = 'D:/Diplomski_neuronske/50m_versions/weather_files_podjela/m
orning'

# Poziv funkcije na izvrsavanje
copy_images_with_txt(source_folder, destination_folder)
```

3) ***Distance-labels_sorter*** skripta čita sadržaj .txt datoteka iz „*distance*“ direktorija gdje su spremljeni podaci o udaljenostima objekata, te potom pronalazi datoteke istih naziva unutar „*labels*“ direktorija. Skripta potom čita vrijednosti iz svake pojedine *.txt* datoteke iz *„distance“* direktorija, te paralelno čita vrijednosti istih linija unutar odgovarajućih *.txt* datoteka u *„labels“* direktoriju. Nakon toga, ovisno o pročitanoj vrijednosti iz *distance* datoteke, uzima odgovarajuću liniju teksta iz *labels* datoteke i zapisuje tu liniju u novu *labels* datoteku pritom je svrstavajući jedan od tri odgovarajuća direktorija (ovisno o pročitanoj udaljenosti iz *distance* datoteke).

```
import os

def process_files(file1_path, file2_path):
    file1_name = os.path.basename(file1_path)
    file2_name = os.path.basename(file2_path)

    #Definiranje lokacija odredisnih direktorija
    output_folder_less_than_10 = "D:/Diplomski_neuronske/town_7/sorted_distanc
es/less10"
    output_folder_10_to_50 = "D:/Diplomski_neuronske/town_7/sorted_distances/1
0to50"
    output_folder_greater_than_50 = "D:/Diplomski_neuronske/town_7/sorted_dist
ances/more50"

    #Kreiranje trazenih odredisnih direktorija ukoliko ne postoje
    if not os.path.exists(output_folder_less_than_10):
        os.makedirs(output_folder_less_than_10)
    if not os.path.exists(output_folder_10_to_50):
        os.makedirs(output_folder_10_to_50)
    if not os.path.exists(output_folder_greater_than_50):
        os.makedirs(output_folder_greater_than_50)

    output_file1 = os.path.join(output_folder_less_than_10, file1_name)
    output_file2 = os.path.join(output_folder_10_to_50, file2_name)
    output_file3 = os.path.join(output_folder_greater_than_50, file2_name)

    lines_written_to_file1 = False
    lines_written_to_file2 = False
    lines_written_to_file3 = False
   #Kreiranje dva dokumenta iz kojih se cita i tri dokumenta za zapisivanje
    with open(file1_path, 'r') as f1, open(file2_path, 'r') as f2, \
         open(output_file1, 'w') as out1, open(output_file2, 'w') as out2, \
         open(output_file3, 'w') as out3:
```

```
        for line1, line2 in zip(f1, f2):
            value1 = int(line1.strip())

            if value1 < 10:
                out1.write(line2)  # Zapisuje cijelu liniju u output doce
                lines_written_to_file1 = True
            elif 10 <= value1 <= 50:
                out2.write(line2)  # # Zapisuje cijelu liniju u output doc
                lines_written_to_file2 = True
            else:
                out3.write(line2)  # # Zapisuje cijelu liniju u output doc
                lines_written_to_file3 = True

    if not lines_written_to_file1:
        os.remove(output_file1)  # Brise dokument ukoliko je prazan

    if not lines_written_to_file2:
        os.remove(output_file2)  # Brise dokument ukoliko je prazan

    if not lines_written_to_file3:
        os.remove(output_file3)  # Brise dokument ukoliko je prazan

    print(f"Processed files: {file1_path} and {file2_path}")
    print(f"Output files less than 10: {output_file1}")
    print(f"Output files between 10 and 50: {output_file2}")
    print(f"Output files greater than 50: {output_file3}")


# Specificiranje foldera koji sadrze lokacije parova dokumenata
folder1 = "D:/Diplomski_neuronske/town_7/distances"
folder2 = "D:/Diplomski_neuronske/town_7/labels"

# Petlja kroz svaki par dokumenata
for file1 in os.listdir(folder1):
    file2 = os.path.join(folder2, file1)
    if os.path.isfile(file2):
        file1_path = os.path.join(folder1, file1)
        file2_path = os.path.join(folder2, file2)
        process_files(file1_path, file2_path)
```

4) ***Label_sorter*** skripta razvrstava grupirane labels *.txt* datoteke po direktorijima na način da provjerava postoji li u pojedinom direktoriju *.png* datoteka (slika) koja je istog naziva kao i *.txt* datoteka koju želimo kopirati. Ukoliko podudaranje u nazivu postoji - *.txt* datoteka se kopira u taj direktorij, a ukoliko ne postoji kopiranje se preskače i provjerava se iduća datoteka.

```
 import os
import shutil

def copy_files(source_folder, target_folders):
    txt_files = []
    png_files = []

    # Dohvaca sve .txt datoteke iz izvornog direktorija
    for file in os.listdir(source_folder):
        if file.endswith(".txt"):
            txt_files.append(file)

    # Prolazi petljom kroz sve ciljane direktorije
    for folder in target_folders:
        # Dohvaca sve .png datoteke iz trenutnog direktorija
        for file in os.listdir(folder):
            if file.endswith(".png"):
                png_files.append(file)

        # Usporeduje imena i kopira datoteke
        for txt_file in txt_files:
            txt_name = os.path.splitext(txt_file)[0]
            for png_file in png_files:
                png_name = os.path.splitext(png_file)[0]
                if txt_name == png_name:
                    source_path = os.path.join(source_folder, txt_file)
                    target_folder = folder
                    shutil.copy2(source_path, target_folder)
                    print(f"Copied {txt_file} to {target_folder}")
                    break  # Prelazi na sljedecu .txt datoteku

        # Cisti listu .png datoteka za prijelaz na sljedeci direktorij
        png_files.clear()

#Definiranje lokacija direktorija
source_folder = "D:/Diplomski_neuronske/50m_versions/all_training_labels"
target_folders = [
```

```
    "D:/Diplomski_neuronske/50m_versions/weather_files_podjela/afternoon",
    "D:/Diplomski_neuronske/50m_versions/weather_files_podjela/almost_night",
    "D:/Diplomski_neuronske/50m_versions/weather_files_podjela/default",
    "D:/Diplomski_neuronske/50m_versions/weather_files_podjela/midday",
    "D:/Diplomski_neuronske/50m_versions/weather_files_podjela/morning"
]

copy_files(source_folder, target_folders)
```

5) ***Weather-file sorter*** skripta služi za kopiranje svih objedinjenih *weather* datoteka u direktorije koji odgovaraju vrijednosti (vrsti vremena) zapisanoj u pojedinoj *.txt weather* datoteci. Skripta usput i broji koliko se *weather* datoteka kopiralo u svaki direktorij zbog lakše usporedbe kasnijih skupova podataka.

```
import os
import shutil

# Specificira se lokacija izvornog direktorija
input_directory = "D:/Diplomski_neuronske/50m_versions_test/weather_test_all"

# Specificiraju se lokacije odredisnih direktorija
output_directories = {
    "morning": "D:/Diplomski_neuronske/50m_versions_test/weather_test_podjela/mor
ning",
    "noon": "D:/Diplomski_neuronske/50m_versions_test/weather_test_podjela/noon",
    "afternoon": "D:/Diplomski_neuronske/50m_versions_test/weather_test_podjela/a
fternoon",
    "almost_night": "D:/Diplomski_neuronske/50m_versions_test/weather_test_podjel
a/almost_night",
    "night": "D:/Diplomski_neuronske/50m_versions_test/weather_test_podjela/night
"
}

# Normalizacija puteva do odredisnih direktorija
output_directories = {key: os.path.normpath(path) for key, path in output_directo
ries.items()}

# Kreiranje rječnika u koji ce se pohranjivati broj kopiranih datoteka u svakom
od direktorija
files_copied_count = {directory: 0 for directory in output_directories.values()}

# Iteriranje preko svake datoteke u izvornom direktoriju
for filename in os.listdir(input_directory):
    file_path = os.path.join(input_directory, filename)

    # Provjera je li datoteka .txt oblika
    if filename.endswith(".txt"):
        with open(file_path, "r") as file:
            # Read the contents of the file
            content = file.read()
```

```
 # Provjera je li sadrzaj prazan
        if content:
            # Dohvacanje rijeci iz dokumenta
            word = content.split()[0].lower()


        # Provjera je li rijec sadrzana u odredisnom direktoriju
        if word in output_directories:
            # Dohvacanje odgovarajuceg odredisnog direktorija
            output_directory = output_directories[word]

            # Kreiranje odredisnog direktorija ukoliko isti ne postoji
            os.makedirs(output_directory, exist_ok=True)

            # Kopiranje datoteke u odredisni direktorij
            shutil.copy(file_path, output_directory)

            # Brojac kopiranih datoteka
            files_copied_count[output_directory] += 1

# Ispis prebrojanih datoteka koje su kopirane u svaki direktorij
for directory, count in files_copied_count.items():
    print(f"Files copied to {directory}: {count}")
```