Izgradnja detektora objekata zasnovanih na dubokim neuronskim mrežama i implementacija na ugradbeni računalni sustav

Mučaj, Klaudija

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

IZGRADNJA DETEKTORA OBJEKATA ZASNOVANIH NA DUBOKIM NEURONSKIM MREŽAMA I IMPLEMENTACIJA NA UGRADBENI RAČUNALNI SUSTAV

Diplomski rad

Klaudija Mučaj

Osijek, 2021. godina

SADRŽAJ

1. UVOD

Od početka proizvodnje automobila, istraživači su nastojali razviti automobil kojim se može upravljati iz udaljenosti. Tako je još 1925. godine predstavljen prvi automobil bez vozača. Njime je pomoću radio valova upravljala osoba koja se nalazila u drugom automobilu, a vožnja je obavljena na Broadway ulici u New York Cityju [1]. Daljnjim razvojem automobila kojima je moguće upravljati iz velikih udaljenosti se uskoro potiče i razvoj nove vrste automobila, odnosno automobila kojima za uspješnu vožnju neće biti potrebna ljudska interakcija. Zajednički naziv za takva vozila je autonomna vozila. Kroz ostatak 20. stoljeća se tehnologija počela razvijati čime se sve više počelo vjerovati da će se jednoga dana uistinu uspjeti razviti potpuno autonomno vozilo. To vjerovanje se 1986. godine produbljuje dolaskom *NavLab*-a, prvog prototipa samovozećeg vozila [2]. Ovime se pokreće val akademskih i industrijskih istraživanja koji su usmjereni prema razvoju autonomnih vozila te u razdoblju do 2000. godine predstavljaju se razni projekti. Jedan od takvih projekta je i europski projekt *PROMETHEUS* kojim je razvijen automobil kojim je izvršena prva autonomna vožnja velikih udaljenosti i to od Münchena u Njemačkoj do Odense u Danskoj. Ovaj val traje i dan danas, jer se još uvijek nije postigla potpuna autonomna vožnja. Razlog leži u činjenici da vozilo mora imati mogućnost prikupljanja informacija iz okoline kako bi ono u potpunosti bilo autonomno, odnosno mora moći lokalizirati samog sebe u njoj i detektirati ostale sudionike u prometu te planirati kretanje. Iako se pomoću raznih senzora koji su ugrađeni u vozilima (poput LiDAR-a, RADAR-a, kamera i slično) može dobiti veliki broj informacija koji opisuju okolinu vozila, računalni sustav vozila ih mora obraditi kako bi na temelju njih donijelo odluke o svom kretanju.

Još se u zadnjoj četvrtini 20. stoljeća u komercijalne svrhe počinju pojavljivati prvi sustavi koji pomažu vozaču prilikom vožnje s glavnim ciljem povećavanja sigurnosti i udobnosti vožnje, a time i prometa. Ti se sustavi nazivaju napredni sustavi za podršku vozaču pri upravljanju vozilom, ili ADAS (engl. *advanced driving assistant system*). ADAS na temelju informacija o okolini i samom vozilu mogu donijeti odluke kojima će se minimizirati mogućnost ljudske pogreške i smanjiti vjerojatnost prometne nesreće, čime se povećava sigurnost samog vozača. Pravi *boom* u razvoju ADAS-a, a i autonomnih vozila se javlja sredinom 2010-ih godina. U to vrijeme se pojavljuju i prve duboke konvolucijske neuronske mreže, CNN (engl. *convolutional neural network*). Pojava CNN-a je omogućila veliki napredak u korištenju slika s kamera za lokalizaciju i klasifikaciju objekata. Naime slike koje se dobivaju npr. s kamerom montiranoj na prednjoj strani vozila moguće je procesirati s ciljem detekcije objekata, odnosno lokalizacije

objekata od interesa i njihovo klasificiranje, čime se nadalje omogućava sigurnija kretnja automobila kroz promet.

U posljednjih nekoliko godina se broj slojeva neuronske mreže povećava, zbog čega je za njihovo korištenje potrebno imati sustav s odgovarajućom količinom memorije, procesorskom snagom, a često i sa specijaliziranim dijelovima kako bi se postigao potreban rad u stvarnom vremenu. Računalni sustavi koji se nalaze u vozilima su ugradbeni, što znači da imaju ograničene računalne resurse. Ovo detekciju objekata u stvarnom vremenu čini vrlo zahtjevnom, jer algoritmi koji se implementiraju unutar njih moraju biti ne samo vremensko učinkoviti, već trebaju imati i mogućnost učinkovitog korištenja ostalih resursa. Drugim riječima, od njih se zahtjeva da koriste što je moguće manju količinu resursa. Iako su se nedavno počele razvijati mreže koje uz korištenje manjeg postotka resursa daju dovoljno dobre rezultate detektiranja objekata upravo kako bi se mogle primijeniti u stvarnom vremenu, velik broj tih rješenja se često testira na računalima, što ne pruža realnu sliku za korištenje u ugradbenim računalnim sustavima.

Zadatak ovog diplomskog rada je razviti dva moderna rješenja detekcije objekata u okolini vozila, YOLOv3 [3] i Faster R-CNN [4], i implementirati ih u ugradbeni računalni sustav, točnije NVIDIA Jetson Nano, kako bi se njihove performanse mogle usporediti na računalu i ugradbenom računalnom sustavu. Detektore se prvo pomoću skupa podataka BDD100K [5] trenira i validira, nakon čega se njihove performanse na računalu i NVIDIA Jetson Nanu testiraju pomoću CityScapes, ApolloScape i skupa podataka izgrađenog u okviru ovog rada. Testiranjem detektora na skupovima različitih od trening i validacijskog skupa se vrši kako bi se otkrila i robusnost izgrađenih detektora na promjenu uvjeta, na primjer: vremenskih uvjeta, geografske lokacije, kamere, i drugo.

Rad je strukturiran na sljedeći način. U drugom poglavlju su opisana neka od postojećih rješenja detekcije objekata u okolini vozila i navedeni su korišteni skupovi podataka. Treće poglavlje sadržava potrebnu teorijsku podlogu, dok četvrto sadrži opis korištene tehnologije. Unutar petog poglavlja opisane su postavke provedenih testiranja, te su dani i diskutirani njihovi rezultati, nakon čega slijedi zaključak.

2. POSTOJEĆA RJEŠENJA DETEKCIJE OBJEKATA ZA AUTONOMNU VOŽNJU

Tijekom godina provedena su razna istraživanja koja su bila orijentirana poboljšanju postojećih algoritama detekcije objekata za autonomnu vožnju ili razvoju novih algoritama. Tablicom 2.1. je dano nekoliko algoritama koji su se tijekom prijašnjih nekoliko godina razvili koristeći javno dostupne skupove podataka. U tablici su performanse pojedinog detektora dane AP-om (engl. *average precission*) kojom se prikazuje uspješnost detektiranja pojedine kategorije objekta (detaljnije objašnjeno u potpoglavlju 3.3.1.), mAP-om (engl. *mean average precission*) koja predstavlja srednju vrijednost dobivenih AP-ova (detaljnije objašnjeno u potpoglavlju 3.3.2.), te brojem obrađenih okvira u sekundi (engl. *frames per second* – FPS). Promatrajući tablicu može se primijetiti kako se uspješnost razvijenih detektora najčešće testira nad dva glavna sudionika prometa, vozilima i pješacima, dok tek manji broj promatra ostale sudionike, poput bicikala i biciklista, i objekte, npr. semafore i prometne znakove. Uspoređujući vrijednosti AP-a vozila i pješaka navedenih detektora može se primijetiti kako RRC [6], *Light Network* [7] i *Gaussian* YOLOv3 [8] postižu vrhunske rezultate, detektor razvijen u okviru rada [9] prosječne rezultate, a CFENet [10] postiže skoro tri puta lošije rezultate. Mogući razlog ovome leži u činjenici što su se RRC, *Light Network, Gaussian* YOLOv3 i *Waymo* detektori trenirali, validirali i testirali nad istim skupom podataka, dok se CFENet trenirao nad jednim i testirao nad drugim skupom. Skupovi podataka za detekciju objekata za autonomnu vožnju se izgrađuju uzorkovanjem okvira (engl. *frames*) iz videozapisa dobivenih snimanjem kamerom postavljenoj na unutarnjoj strani prednjeg vjetrobranskog stakla. Zbog ovoga skupovi često sadržavaju velik broj sličnih scena, što, i nakon podjele na trening, validacijski i testni skup, uvelike utječe na rezultate testiranja.

Tablica 2.1. Prikaz performansi nekoliko razvijenih algoritama detekcije objekata u okolini vozila

Model	**AP – vozilo**	**AP – pješak**	**AP – bicikl**	**AP – biciklist**	**AP – prometni znak**	**AP – semafor**	**mAP**	**FPS**
RRC[6]	89.85%	75.33%	-	76.47%	-	-	-	-
Light Network [7]	84.31%	72.18%	-	-	-	-	-	4-6
Gaussian YOLOv3 [8]	90.2%	79.57%	-	81.3%	-	-	83.61%	43.13
CFENet [10]	35.39%	17.62%	14.58%	12.73%	28.29%	8.67%	22.34%	20+
Waymo [9]	63.7%	55.8%	-	-	-	-	-	-

KITTI [11] je jedan od najpopularnijih skupa podataka za autonomnu vožnju koji sadrži podatke raznih senzora za obavljanje zadataka poput: stereovizije, *optical flow*-a, segmentacije, i mnogo drugo. Za detektiranje 2D objekata KITTI sadrži ~15000 slika kojima su sadržane oznake: automobila, kombija, kamiona, pješaka, osoba koje sjede, biciklista i tramvaja. Skup je podijeljen na trening i test skup, a test skup nije javno dostupan.

Mnoga istraživanja svoja rješenja za detekciju objekata na slikama testiraju pomoću KITTI skupa. Među njima je i [6] kojim je u svrhu poboljšavanja performansi pomoću RRC-a (engl. *recurrent rolling convolution*) razvijen novi pristup SSD (engl. *single shot detector*) modelu. Autori rada [6] su javno dostupan trening skup KITTI skup podataka podijelili na dva skupa i time stvorili skupove za trening i validaciju, a prije samog treniranja su nad podatcima obavili augmentaciju podataka. Augmentacija podataka predstavlja postupak kojim se postojeći skup za učenje povećava pomoću različitih transformacija, poput zamućivanja slike, rotacije, promjene zasićenosti boja i drugo. Za optimizaciju procesa učenja modela korišten je SGD (engl. *stohastic gradinet descent*) s momentom od 0.9, a stopa učenja (engl. *learning rate*) se tijekom procesa učenja mijenjala, točnije početna vrijednost je postavljena na 0.0005 te se svakih 40,000 iteracija dijelila s 10. Svojim prijedlogom su autori rada [6] postigli vrhunske rezultate u usporedbi s dotadašnjim modelima i to kroz sve tri razine težine koje je definirao KITTI. Za srednju (engl. *moderate*) težinu, na kojoj je IoU (engl. *Intersection over Union*), odnosno omjer između površine područja preklapanja i površine područja unije predviđenog BB-a (engl. *bounding box*, oznaka lokacije objekta na slici) i stvarnog BB-a, postavljen na 0.5, su postignuti idući rezultati AP-a: za automobil 89.85%, pješaka 75.33%, te biciklista 76.47%.

Radom [7] je predstavljen detektor koji je dizajniran za postizanje što je moguće boljih rezultata s ograničenim računalnim resursima. Autor predstavlja model koji se temelji na ShuffleNetv2 mreži, a sastoji se od izmijenjene R-CNN mreže i RPN-a u svrhu smanjivanja računalnih zahtjeva i zadržavanja što veće točnosti detekcija. Predstavljeno rješenje je trenirano na NVIDIA GTX 1080 GPU-u, a nad KITTI skupom podataka je trenirano za detekciju vozila i pješaka. Rezultati dobiveni za srednju težinu KITTI izazova su: 84.31% za vozila, te 74.98% za pješake. Kako bi se ocijenila mogućnost korištenja razvijenog modela u sustavima za rad u stvarnom vremenu autor uz mjerilo točnosti mreže promatra i potrebne memorijske resurse, GFLOPs (engl. *giga floating point operations per second*) i FPS. Autor je ustanovio da je veličina razvijenog modela je 30 MB, broj GFLOPs-a 1.12, a FPS za pojedini korišteni ugradbeni računalni

sustav je: za Raspberry Pi 4 – 4-6 FPS-a; NVIDIA Jetson TX2 10-12 FPS-a; čime je pokazano da model nije prikladan za korištenje u sustavima za rad u stvarnom vremenu.

BDD100K (engl. *Berkeley DeepDrive 100K*) [5] predstavlja trenutno najveći skup podataka razvijenog za različite zadatke autonomne vožnje, kao što su: detektiranje i praćenje objekata, određivanje oznaka traka, i mnogo drugo. Sastoji se od 100000 videozapisa skupljenih iz različitih američkih gradova i različitih vrsti scena (ulice grada, autoceste, stambena područja, i drugo). Za zadatak detektiranja objekata iz skupa videozapisa odabrano je 100000 okvira na kojima su označeni BB-ovi idućih objekata i sudionika prometa: automobil, autobus, kamion, motocikl, bicikl, vlak, prometni znak, semafor, osoba, te vozač (biciklist ili motociklist) [5]. Iako je BDD100K trenutno relativno mlad skup podataka, pomoću njega su provedena mnoga istraživanja, među kojima je i [10].

Autori [8] predlažu izmjenu YOLOv3 mreže kojom se poboljšava preciznost detekcije objekata, ali se zadržava i mogućnost primjene u sustavima za rad u stvarnom vremenu. Predložena izmjena odnosi se na način modeliranja BB-ova, točnije koordinate BB-ova se modeliraju pomoću Gauss-ove normalizacije. Predloženi model autori [8] nazivaju *Gaussian YOLOv3*. Treniranje modela se provelo na grafičkoj kartici NVIDIA GTX 1080 Ti s CUDA-om 8.0 i cuDNN v7, pri čemu je veličina *batch*-a postavljena na 64, a stopa učenja na 0.0001. Model je treniran nad KITTI skupom, a testiran je nad KITII i BDD100K skupom. Nad KITTI skupom model postiže mAP od 86.79% i FPS od 24.91, dok nad BDD100K ostvaruje uveliko nižu vrijednost mAP-a (18.4%) i sličnu vrijednost FPS-a (22.5).

U [10] autori predlažu novi detektor objekata: CFENet (engl. *comprehensive feature enhancment network*). CFENet se temelji na arhitekturi SSD-a. Uvođenjem modula za sveobuhvatno poboljšanje značajki u osnovnu arhitekturu SSD-a, autori [10] postižu bolje rezultate pri detektiranju malih objekta u odnosu na neke od suvremenih metoda. Autori detektora kroz dva testiranja uspoređuju rezultate CFENet-a sa SSD-om i RefineNet-om. Zbog ograničenih računalnih resursa, svim testiranim mrežama (CFENet, SSD, te RefineDet) je *backbone* VGG-16. Prvo testiranje se provodi nad MSCOCO [12] skupom podataka. Njime su autori potvrdili da CFENet s većom uspješnošću detektira ne samo objekata „normalne“ veličine, već i objekte manjih veličina. Drugo testiranje je provedeno nad BDD100K skupom podataka, za koje je IoU postavljen na 0.7, a veličina ulazne slike na 512x512. Rezultatima ovog testiranja se pokazalo da CFENet s 19.1% preciznosti nadmašuje RefineDet za 1.7%, te da je brzina CFENet-a (21 FPS-a) nešto sporija od RefineDet-a (22.3 FPS-a), što pokazuje da su oba modela primjenjiva u stvarnom

vremenu. Autori [10] su CFENet model prijavili i na natjecanje organizirano u okviru [5]. Kako bi na njemu ostvarili što bolje rezultate autori su nad CFENet-om izvršili odgovarajuće izmjene i postigli mAP 29.69%.

Radom [9] je model Faster R-CNN-a implementiran s ekstraktorom značajki ResNet-101. Skup podataka nad kojom se trenira model je razvijen u okviru rada [9] vožnjom kroz tri američka grada: Phoenix, Mountain View, San Francisco. Razvijeni skup sadržava oznake: vozila, pješaka i biciklista. Za testiranje rada modela autori koriste dvije razine težine: LEVEL_1 i LEVEL_2. Prilikom testiranja rada modela je za detektiranje vozila postavljen IoU na 0.7 čime se za AP na LEVEL_1 dobiva rezultat od 63.7%, a za LEVEL_2 53.3%. Za pješake je IoU postavljen na 0.5 te se za AP dobivaju rezultati od 55.8 pri LEVEL_1 te 52.7 na LEVEL_2. Za bicikliste nisu objavljeni rezultati. Kao jedan od razloga nižih rezultata autori navode činjenicu da su slike iz Phoenixa i Mountain Viewa dobivene vožnjom predgrađem, dok su slike iz San Francisca dobivene vožnjom gradom, tj. kroz urbano područje.

Autori rada [13] treniranjem YOLO modela nad različitim skupovima podataka razvijaju nekoliko različitih sustava za detektiranje objekata. Sustavi su testirani kroz nekoliko eksperimenata. Prvim eksperimentom se sustav treniran nad PASCAL VOC skupom [14] testirao nad KITTI skupom podataka. Ovim eksperimentom su autori pokazali kako mreža koja nije specifično dizajnirana za autonomnu vožnju ima relativno zadovoljavajuće rezultate s prosječnim IoU-om oko 40%. Za provođenje drugog eksperimenta su autori dostupne slike KITTI skupa podijelili na trening i testni skup, te su mrežu dobivenu prvim eksperimentom dodatno trenirali pomoću dobivenog trening skupa, točnije obavljeno je fino podešavanje (engl. *fine-tuning*) mreže. Dobivena mreža se zatim testirala nad izdvojenim slikama KITTI skupa, a dobiveni rezultati preciznosti i odziva su značajno viši u odnosu na mrežu prvog eksperimenta. Autori su primijetili i kako je zbog relativno velikih dimenzija slika KITTI skupa (320x1060) dobivena vrlo niska brzina inferencije od 2.5 FPS-a. Kako bi potvrdili hipotezu proveli su treći eksperiment. Mreža dobivena prvim eksperimentom se fino podešava i testira pomoću slika iz Udacity skupa podataka (koji sadrži oznake za: automobile, kamione i pješake) smanjenih dimenzija. Mreža time postiže 9.58 FPS-a. Ovim eksperimentima su autori pokazali da za primjene u autonomnoj vožnji YOLO treba proći kroz još mnoga poboljšanja.

2.1. Odabrani skupovi podataka

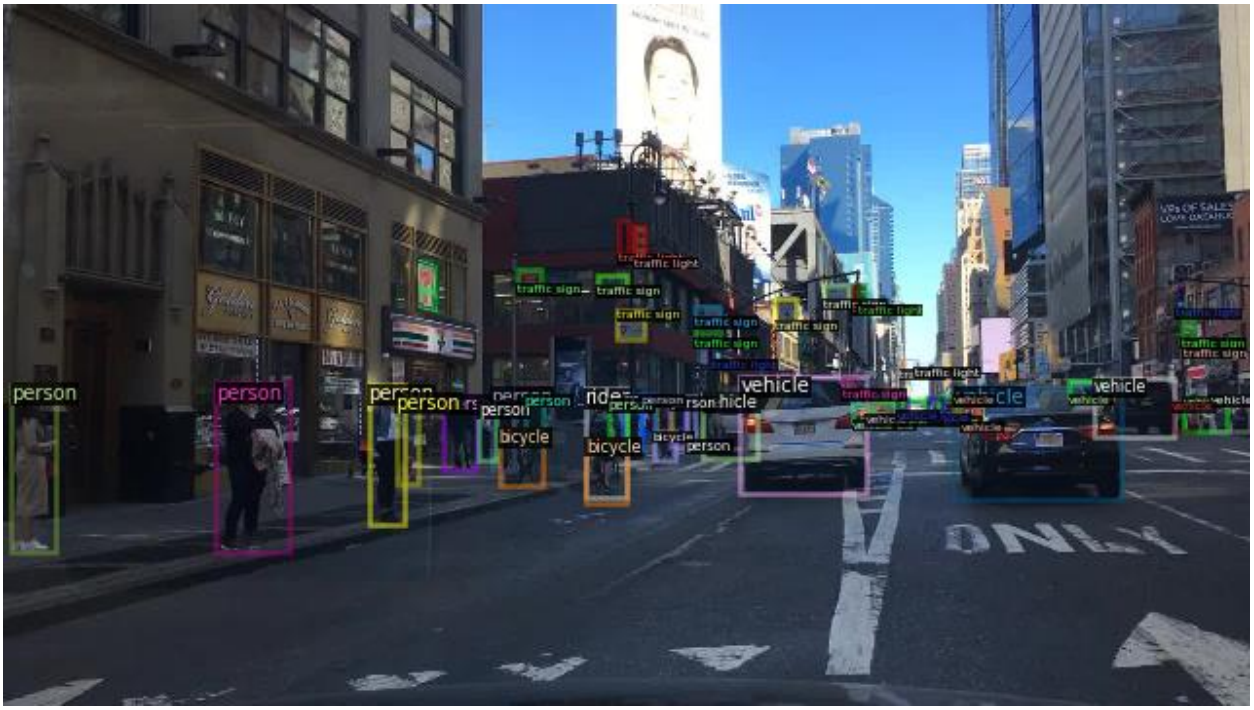
Detektori koji će se razviti u okviru ovog rada moći će detektirati najvažnije objekte i sudionike prometa: vozila (odnosno, automobili, kombiji, kamioni, autobusi i motocikli će se promatrati kao jedna klasa), bicikle, pješake, bicikliste, prometne znakove te semafore. Kako bi se ovo omogućilo potrebno je odabrati odgovarajući skup podataka koji sadrži oznake navedenih objekata. U tablici 2.2. prikazano je nekoliko postojećih skupova podataka za detekciju objekata u autonomnoj vožnji, kao i broj oznaka koje pojedini skup sadrži za odabrane klase. Osim skupova prikazanih unutar tablice 2.2. razvijene su i mnoge druge, kao na primjer D^2-City [15], no informacije o postojećim klasama kao i o broju njihovih oznaka unutar skupa nisu dostupne, zbog čega ti skupovi podataka nisu navedeni u tablici. Promatrajući tablicu je moguće vidjeti kako svaki skup sadrži oznake za vozila i pješake, ali ne i za ostale odabrane klase. Iako i BDD100K i A2D2 sadrže i oznake za bicikle i bicikliste, A2D2 ne sadrži prometne znakove ni semafore, zbog čega je za treniranje detektora odabran BDD100K skup podataka.

Tablica 2.2. Popis nekoliko postojećih skupova podataka za detekciju objekata i postojanja odabranih klasa. Znak '-' označava nepostojanje klase unutar odgovarajućeg skupa.

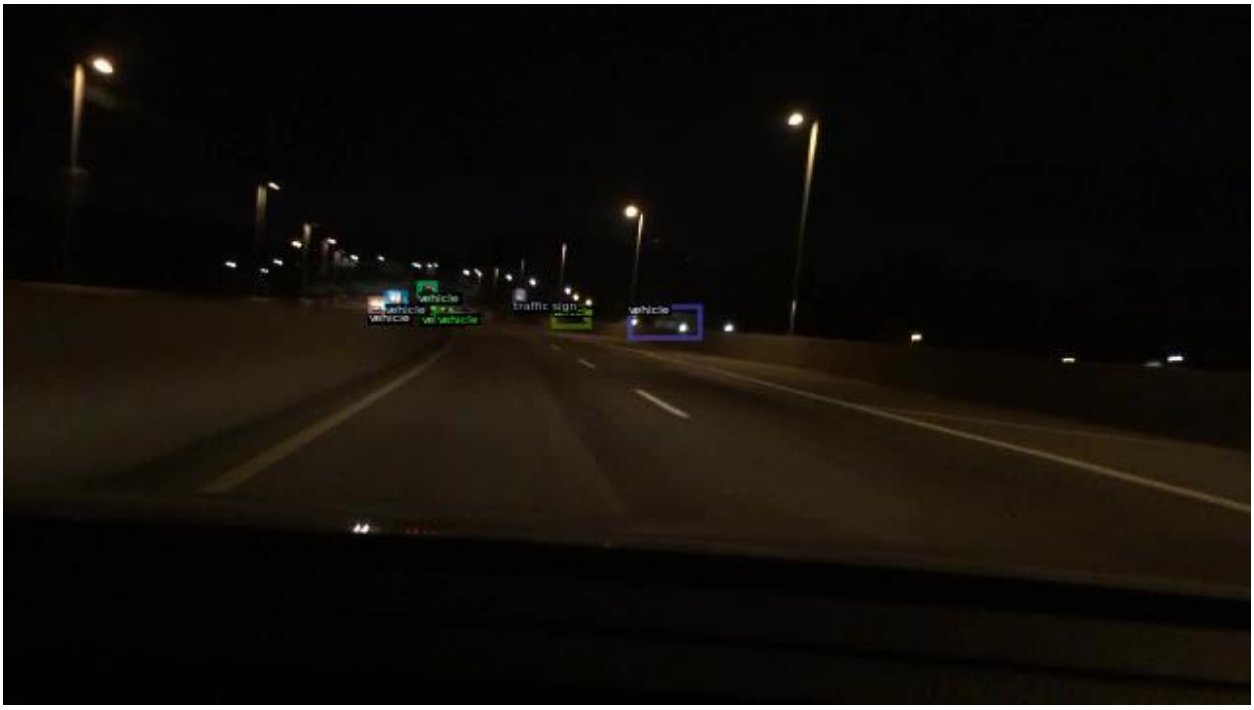
Skup podataka / Klasa	Vozilo	Bicikl	Pješak	Biciklist	Prometni znak	Semafor
KITTI [11]	~205k	-	~115k	~20k	-	-
Waymo [9]	9M	-	~2.7M	~81k	-	-
NuImages [16]	322095	17060	168580	-	-	-
DAWN [17]	7362	-	483	-	-	-
BDD100K [5]	1085800	10229	129262	6461	343777	265906

BDD100K [5] predstavlja trenutno najveći skup podataka razvijen za različite zadatke autonomne vožnje (detektiranje i praćenje objekata, određivanje oznaka traka, i mnogo drugo). On se sastoji od 100000 videozapisa prikupljenih iz različitih američkih gradova i različitih vrsta scena (ulice grada, autoceste, stambena područja, i drugo). Za zadatak detektiranja objekata iz skupa videozapisa odabrano je 100000 okvira na kojima su označeni BB-ovi sljedećih objekata: automobil, autobus, kamion, motocikl, bicikl, vlak, prometni znak, semafor, pješak, te biciklist [5]. Od navedenih 100000 okvira javno je dostupno 70000 slika, koji su nadalje podijeljeni na skup za treniranje (60000 slika) i skup za validaciju (10000 slika). Kako detektori razvijeni ovim radom neće morati moći detektirati sve klase koje pruža BDD100K skup podataka, tako je suvišne klase bilo potrebno filtrirati. Osim toga, detektori će morati detektirati automobile, kamione, autobuse i motocikle kao jednu klasu, točnije vozilo, zbog čega je bilo potrebno svaku oznaku tih

klasa promijeniti u vozilo. Na slici 2.1. je prikazano nekoliko primjera slika iz trening skupa, dok je na slici 2.2. moguće vidjeti primjere slike iz validacijskog skupa. Na navedenim slikama se može vidjeti kako BDD100k skup sadrži velik broj raznolikih slika, kao što su slike urbanih područja, autoceste, različitih doba dana (dan, predvečer, večer), različitih vremenskih uvjeta (sunčano, oblačno, kišovito, snježno, i dr.).



a)



b)

c)

Slika 2.1. Prikaz nekoliko slika iz BDD100K trening skupa podataka


vehicle
traffic sign

a)


vehicle
vehicle
vehicle
vehicle
vehicle

b)

c)

Slika 2.2. Prikaz nekoliko slika iz BDD100K validacijskog skupa podataka

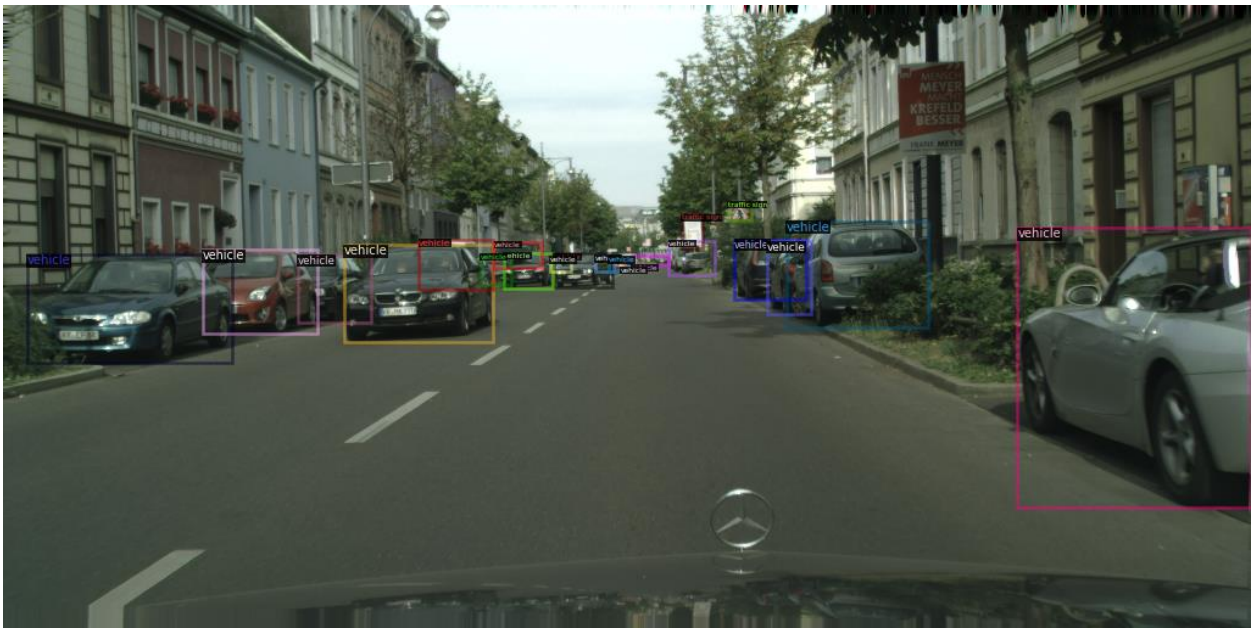
Osim što je potrebno odrediti skup za treniranje detektora, potrebno je odrediti i skup za testiranje. Promatrajući tablicu 2.2. može se uočiti kako veliki broj postojećih skupova podataka za detektiranje objekata ne sadržavaju oznake odabranih klasa. Iz tog razloga se proučavaju i postojeći skupovi podataka za semantičku segmentaciju. Iako je semantička segmentacija poseban zadatak računalnog vida, postojeći podatci se mogu prilagoditi za zadatak detekcije objekata. Ovo je moguće postići tako što se oznake na razini piksela pretvaraju u BB-ove na idući način: za odgovarajući objekt traže se točke segmenta koje se nalaze krajnje lijevo, desno, gore i dolje, čime se omogućava određivanje gornje lijeve točke i donje desne točke BB-a, a time nadalje i njegova širina i visina. Popis nekoliko postojećih skupova podataka za semantičku segmentaciju je moguće vidjeti u tablici 2.3. Postoji mnogo razvijenih skupova podataka za semantičku segmentaciju, no zbog nedostupnosti informacija o postojećim klasama kao i broju njihovih oznaka ti skupovi nisu navedeni unutar tablice 2.3. (poput Synthia-e [18]). Uspoređujući tablice 2.2. i 2.3. primjetno je kako skupovi podataka za semantičku segmentaciju u odnosu na skupove za detekciju objekata pružaju puno veći opseg označenih objekata. Uzimajući u obzir odabrane klase objekata može se uočiti kako se za testiranje detektora može koristiti svaki skup podataka iz tablice. Za testiranje detektora su u ovom radu odabrani skupovi CityScapes [19] i ApolloScape [20].

Tablica 2.3. Popis nekoliko postojećih skupova podataka za semantičku segmentaciju i postojanja odabranih klasa. Znak '*' označava postojanje navedene klase unutar odgovarajućeg skupa, ali je informacija o broju oznaka nedostupna.

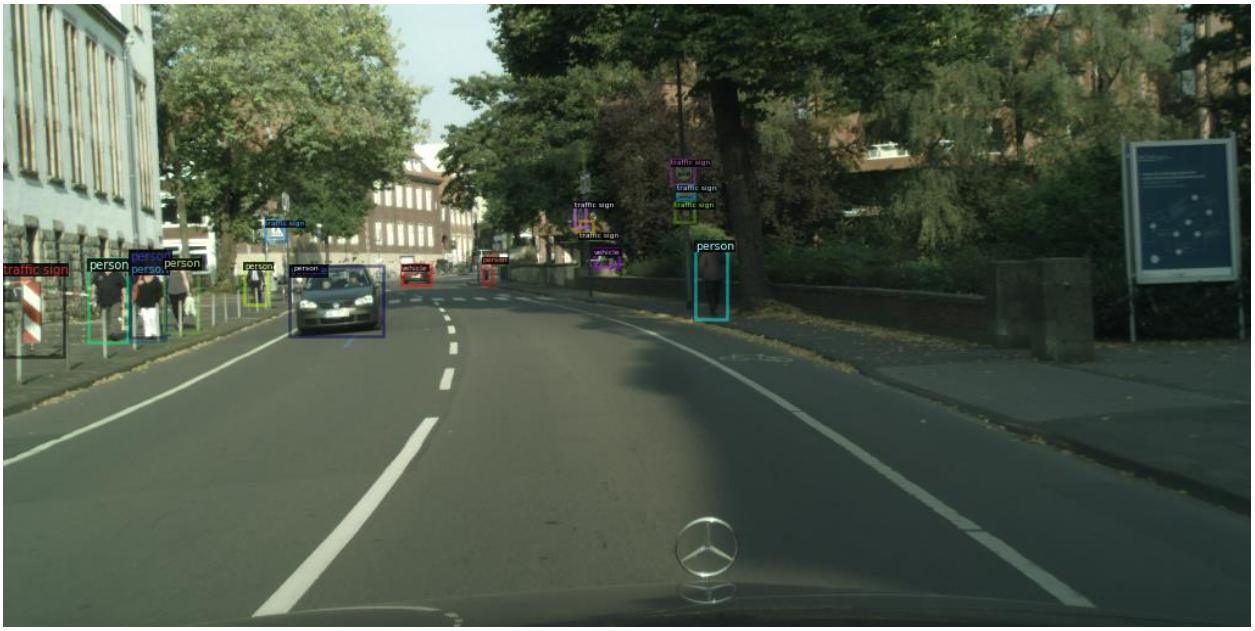
Skup podataka / Klasa	**Vozilo**	**Bicikl**	**Pješak**	**Biciklist**	**Prometni znak**	**Semafor**
CityScapes [19]	~41k		~24.4k		*	*
ApolloScape [20]	~1.9M		~543k		*	*
Mapillary Vistas [21]	~170k	~7k	~55k	~4k	~100k	~80k
BDD100K [17]	64647	426	6989	295	18243	14606
A2D2 [22]	~485k	~30k	~35k	*	*	*

CityScapes [19] predstavlja skup slika koji je namijenjen zadatku semantičke segmentacije. Sastoji se od velikog broja slika dimenzija 2048*x*1024 snimljenih vožnjom kroz 50 različitih gradova (ponajviše Njemačkih gradova) kroz period od nekoliko mjeseci, čime obuhvaća visoku varijabilnost scena. Unutar skupa je označeno 30 različitih klasa, među kojima su (osim klasa odabranih ovim radom): cesta, pločnik, zgrada, zid, ograda, nebo, vegetacija, i mnoge druge. CityScapes skup podataka je ovisno o kvaliteti oznaka segmenata podijeljen na dva manja skupa:

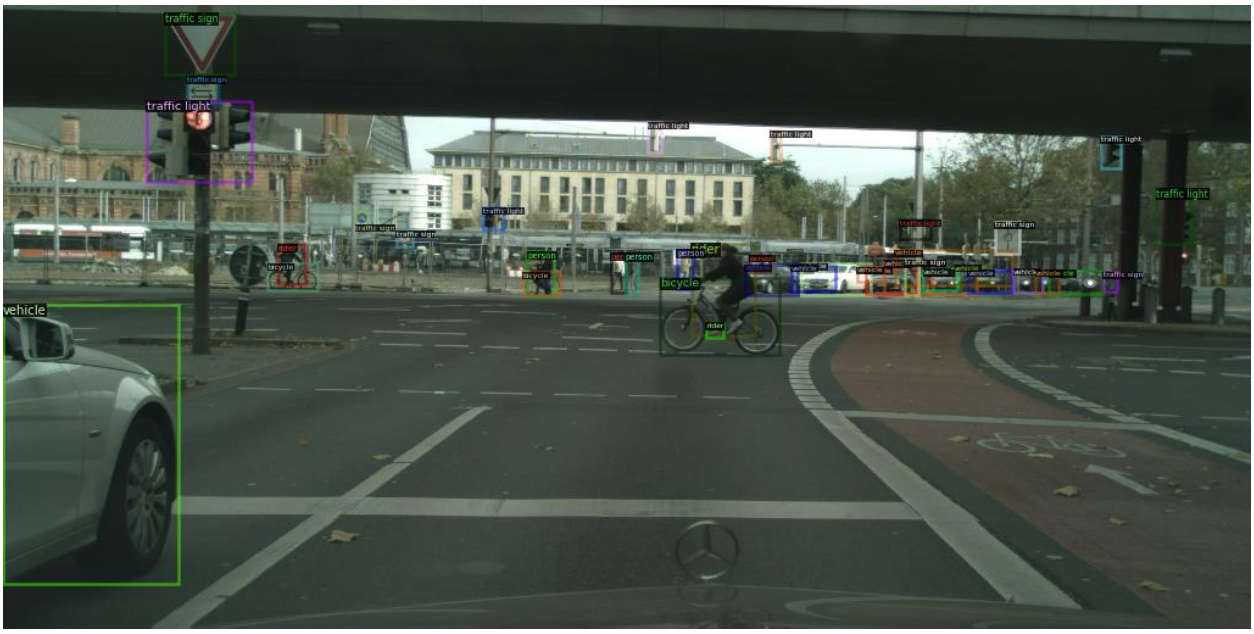
- Fine oznake (engl. *fine annotations*) – predstavlja skup od 5000 slika (podijeljenih u trening, validacijski i testni skup) u kojem su segmenti na razini piksela vrlo precizno označeni. Za testiranje detektora razvijenih ovim radom odabran je ovaj skup. Potrebno je napomenuti kako od tih 5000 slika CityScapes javno pruža oznake za 3475 slika, odnosno ne pruža oznake testnog skupa. Stoga se za testiranje detektora koriste slike s dostupnim oznakama. Primjer oznaka transformiranih u BB-ove prikazano je na slici 2.3.
- Grube oznake (engl. *coarse annotations*) – predstavlja skup od 20000 slika u kojem su segmenti na razini piksela grubo označeni, odnosno neprecizno.


vehicle
vehicle
vehicle
vehicle
vehicle
vehicle
vehicle
vehicle

a)


traffic sign
person
person
person
person
traffic sign
traffic sign
traffic sign
traffic sign
person

b)



c)

Slika 2.3. Prikaz nekoliko slika iz CityScapes skupa podataka

ApolloScape [20] je jedan od najvećih skupova podataka orijentiran prema rješavanju različitih zadataka autonomne vožnje, među kojima su: predikcija kretnje, 3D percepcija, razumijevanje scene i drugo. Među tim podatcima je i skup za raščlanjivanje scene (engl. *scene parsing*) koji sastoji od 146997 slika dimenzija 3384*x*2710 snimljenih u različitim gradovima. Skup za raščlanjivanje scene sadržava oznake na razini piksela, informacije o pozama, te mape dubine za statičku pozadinu. Skup je podijeljen na trening, validacijski i testni skup (čiji podatci nisu javno dostupni). Unutar skupa je označeno 25 različitih klasa među kojima su (osim klasa odabranih ovim radom): cesta, pločnik, most, zgrada, ograda, nebo, vegetacija, i mnoge druge. Kako su za CityScapes skup dostupne fine oznake za 3475 slika, tako je iz ApolloScape skupa nasumično odabrano 3475 slika nad kojima će se detektori razvijeni ovim radom testirati. Nekoliko primjera odabranih slika s njihovim oznaka prikazano je na slici 2.4.

a)

b)

c)

Slika 2.4. Prikaz nekoliko slika iz ApolloScape skupa podataka

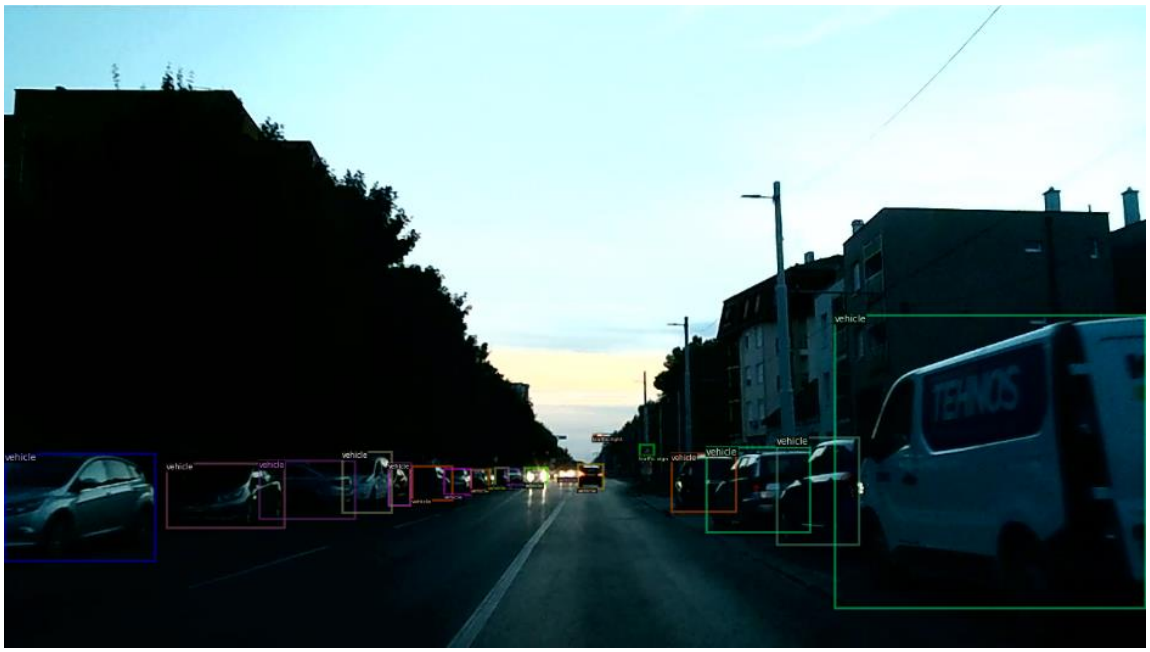
2.2. Izgrađeni skup podataka

U okviru ovog rada se za testiranje razvijenih detektora razvija vlastiti skup podataka koji sadrži oznake svih kategorija za koje se ovim radom oni razvijaju detektori. Za kreiranje skupa podataka korištena je *Trust Trino HD* web kamera rezolucije 1280 x 720. Navedena kamera korištena je kako bi se snimilo nekoliko videozapisa prometa u Osijeku pri 30 FPS-a, tako što je postavljena na unutarnjoj strani prednjeg vjetrobranskog stakla. Iz dobivenih videozapisa izdvojeno je ukupno 3000 okvira na način da je za svaku sekundu vožnje izdvojen prvi okvir, a označavanje objekata na slikama se provodilo pomoću alata *LabelImg*. Na slici 2.5. moguće je vidjeti nekoliko primjera slika iz razvijenog skupa s njihovim oznakama. Iz slike je moguće vidjeti kako se unutar skupa nalaze slike različitih scena (stambeno područje, obilaznica) snimljene u različitim dobima dana (poslijepodne, predvečer) i različitih vremenskih uvjeta (sunčano, oblačno). Tablicom 2.4. dan je broj oznaka pojedine kategorije koje sadrži izgrađeni skup podataka. Uspoređujući tablicu 2.4. s tablicama 2.2. i 2.3. može se primijetiti kako broj instanci pojedine kategorije prati trend postojećih skupova podataka, točnije objekt koji sadržava najveći

broj instanci je automobil, a biciklisti, bicikli i pješaci predstavljaju objekte najmanje reprezentacije. Iako broj instanci ostalih sudionika i objekata prometa nije u desecima tisuća, kao što je slučaj s automobilima, njihov broj instanci daje adekvatnu reprezentaciju istih u prometu.

a)

b)



c)

Slika 2.5. Prikaz nekoliko primjera slika iz izgrađenog skupa podataka.

Tablica 2.4. Prikaz broja instanci pojedine kategorije koje sadržava razvijeni skup podataka

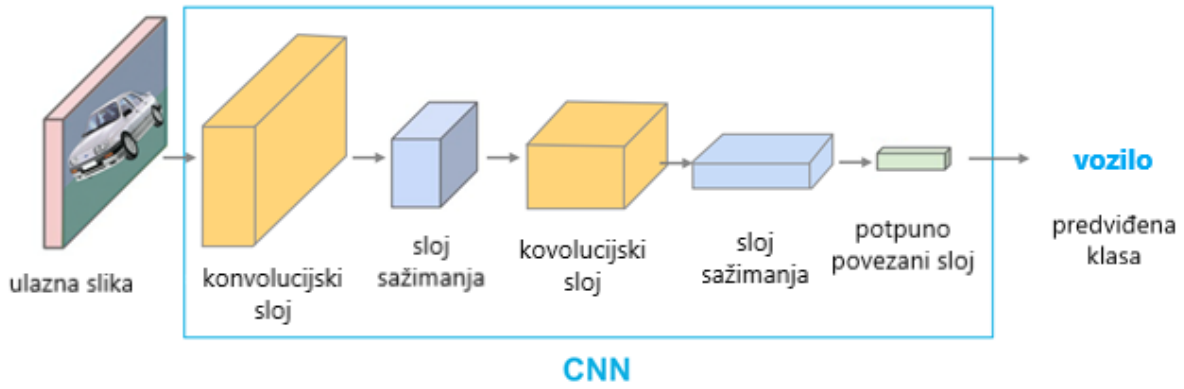
Kategorija	**Broj instanci**
Vozilo	19801
Bicikl	902
Pješak	1679
Biciklist	381
Prometni znak	5746
Semafor	6196

3. DETEKCIJA OBJEKATA

Detekcija objekata je jedan od zadataka računalnog vida. Njome se na slici lokalizira varijabilni broj objekata od interesa, točnije određuju se područja na kojima se nalaze objekti pomoću tzv. graničnih pravokutnika (engl. *bounding box* – BB) te se tim pronađenim objektima pridružuje klasa. Dok je čovjeku ovaj zadatak intuitivan, računalima je zbog velike varijabilnosti slika (okluzija objekata, promjene osvjetljenja, položaja kamere i drugo) ovo složen zadatak pa se zato problemu pristupa pomoću strojnog učenja. Detekcija objekata pomaže pri razumijevanju i analizi scena u slikama ili videozapisima te je stoga njena primjena vrlo raširena: detekcija lezija, tumora i slično (medicina); prepoznavanje lica; prepoznavanje znakova; prepoznavanje aktivnosti; praćenje objekata; i mnogo drugo [23]. Iako je duboko učenje jedna od mnogih grana strojnog učenja, razlika između načina razvoja detektora strojnog učenja i dubokog učenja je značajna. Standardni pristup detekciji objekata u području strojnog učenja podrazumijeva izdvajanjem značajki nekom od postojećih metoda (SIFT, HOG), a zatim se najčešće plitka mreža koristi za potrebe klasifikacije određenog dijela slike. Kod dubokog učenja je korak ekstrakcije značajki sadržan u samoj mreži. Vrlo često modeli razvijeni strojnim učenjem sadržavaju velik broj parametara i nisu optimirane za dani problem. Arhitektura modela dubokog učenja se obično sastoji od dva dijela: kodera (engl. *encoder*) koji ulazne podatke provlači kroz slojeve mreže s ciljem izlučivanja značajki; i dekodera (engl. *decoder*) koji na temelju izlaznih podataka kodera daje predikciju [24]. Tijekom godina razvili su se razni pristupi izgradnji takvih detektora kojima se nastoji riješiti problem detektiranja objekata u stvarnom vremenu, od kojih su najpopularniji: *single shot* detektori, ili SSD, za kojeg se najčešće kao predstavnik uzima YOLO (engl. *You Only Look Once*); te RPN (engl. *region proposal network*) na kojem se bazira Faster R-CNN (engl. *Faster Region-based Convolutional Neural Networks*) [24].

I u YOLO-u i u Faster R-CNN-u se kao koder podataka primjenjuje konvolucijska neuronska mreža. CNN su vrsta dubokih neuronskih mreža (engl. *deep neural network,* DNN) koje se primarno u području računalnog vida. One pomoću niza međusobno povezani slojeva dobiveni ulaz pretvaraju u odgovarajući izlaz. Najčešće se unutar njih razlikuje tri vrste sloja: konvolucijski sloj (engl. *convolutional layer*), sloj sažimanja (engl. *pooling layer*, neobavezan sloj) i potpuno povezani sloj (engl. *fully connected layer*). Potpuno povezani sloj skupa sa *softmax* aktivacijskom funkcijom se naziva *softmax* sloj. *Softmax* sloj se upotrebljava kao izlazni sloj CNN-a koji je namijenjen višeklasnoj klasifikaciji, zbog čega broj izlaza iz sloja mora biti jednak broju klasa od interesa. Uporabom *softmax*-a se za detektirani objekt dobiva vjerojatnost pripadnosti pojedinoj klasi od interesa, pri čemu je zbroj vjerojatnosti uvijek jednak jedan. Ulazni podatci CNN-a su

organizirani u topologiji nalik mreže (npr. slike) iz kojih CNN različitim izdvojenim značajkama dodjeljuje važnosti, odnosno određuje težine (engl. *weights*) i *bias*. U slučaju slike, CNN koriste strategiju podjele težine kako bi se na različitim lokacijama slike mogle prepoznati slične strukture. Drugim riječima, kako se konvolucijski sloj CNN-a temelji na matematičkoj operaciji konvolucije (engl. *convolution*), tako se postiže hijerarhijsko sastavljanje značajki podataka i to od značajki najmanje važnosti prema značajkama veće važnosti. Osim toga, uporabom konvolucije postiže se drastično smanjenje broja parametara koje je potrebno trenirati što omogućava bolje prilagođavanje modela predanom mu skupu podataka [25]. Na slici 3.1. moguće je vidjeti primjer jednostavne arhitekture CNN-a koja se sastoji od dva konvolucijska sloja, dva sloja sažimanja i jednog potpuno povezanog sloja (ukupno 5 skrivenih sloja) [26].



Slika 3.1. Primjer arhitekture jednostavnog CNN-a. Slika preuzeta iz [26].

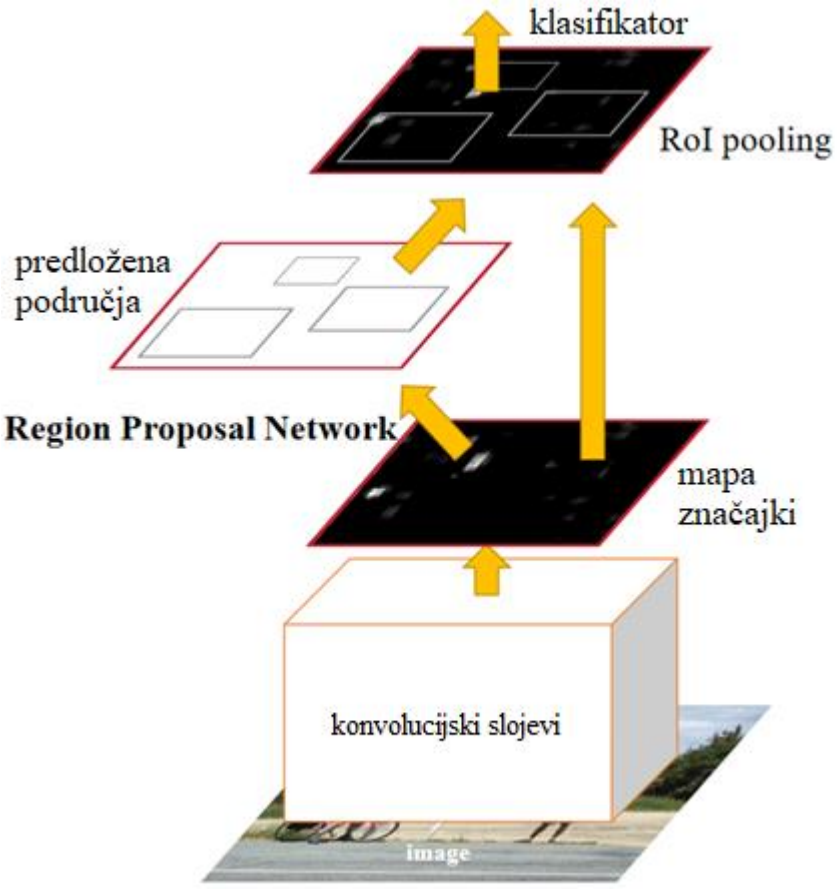
3.1. Faster R-CNN

Faster R-CNN detektoru su prethodila dva modela detektiranja objekata: R-CNN (engl. *Region-based Convolutional Neural Network*) i Fast R-CNN. Pojavom R-CNN-a pokrenuo se razvoj modela za detektiranje objekata koji se temelje na područjima (engl. *region-based*). R-CNN-om se pomoću algoritma *Selective Search* prvo predlažu područja slike u kojima postoji mogućnost pronalaska objekta. Zatim se predložena područja predaju CNN-u kojim se izvlače značajke pojedinog područja. Na kraju se dobivene značajke prosljeđuju: SVM-u – koji klasificira područja; i modelu linearne regresije – koji granice predloženog područja prilagođava dimenzijama objekta (ako se objekt u njemu nalazi) [27]. Može se reći kako se algoritmom R-CNN-om problem detektiranja objekta sveo na problem klasifikacije. No, zbog toga što se svako predloženo područje pojedinačno predaje CNN-u, R-CNN-a nije pogodan za uporabu u stvarnom

vremenu. Stoga je razvijen Fast R-CNN koji cijelu sliku odmah prosljeđuje CNN-u, čime se on primjenjuje samo jednom. To je omogućeno tako što se prije predlaganja područja obavlja izvlačenje značajki na temelju kojih se predlažu područja nalaska objekta. Osim toga, i klasifikacija pojedinog objekta je spojena s CNN-om tako što su modeli SVM-a zamijenjeni *softmax* slojem [28]. Iako su ovime postignute mnogo bolje performanse u odnosu na R-CNN, zbog daljnje uporabe algoritma *Selective Search* za predlaganje područja Fast R-CNN još uvijek nije dovoljno dobar za uporabu u stvarnom vremenu. Faster R-CNN rješava ovaj problem na način da *Selective Search* zamjenjuje s neuronskom mrežom za predlaganje područja, točnije uvodi RPN. Prema izvornom radu [4] Faster R-CNN predstavlja sustav za detektiranje objekata koji se sastoji od dva dijela:

- RPN – konvolucijska mreža koja predlaže područja u kojima se potencijalno nalazi neki objekt;
- Fast R-CNN – detektor koji za detektiranje objekta koristi predložena područja.

Osnovna arhitektura Faster R-CNN-a prikazana je na slici 3.2.



Slika 3.2. Prikaz osnovne arhitekture Faster R-CNN sustava za detektiranje objekata. Slika preuzeta iz [4].

3.1.1. *Backbone* mreže

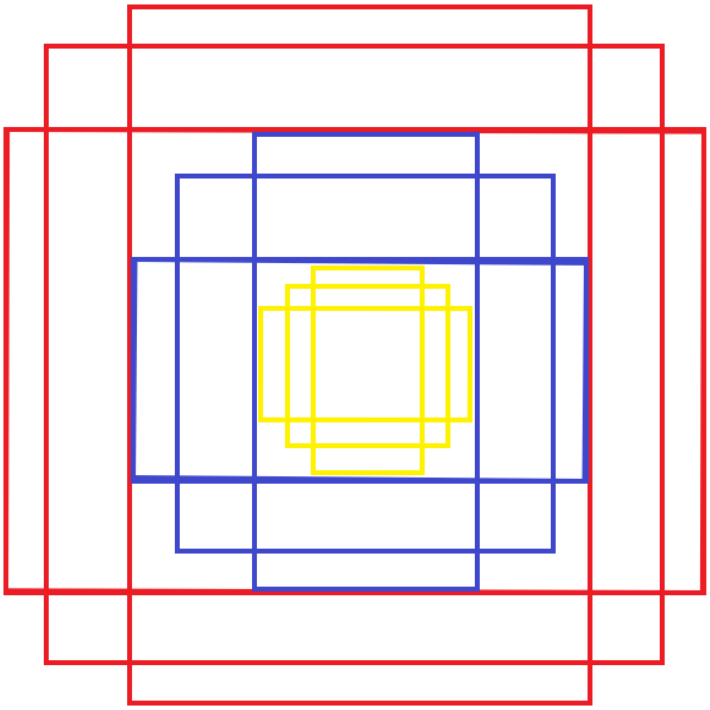
Na slici 3.2. moguće je vidjeti kako se slika (dimenzija: visina x širina x dubina) prije predaje RPN-u predaje unaprijed treniranom CNN-u, takozvanom *backbone*-u. Zadatak ovog CNN-a je izvlačenje značajki iz slike, odnosno stvaranje takozvane mape značajki (engl. *feature map*) koja se nadalje prosljeđuje RPN-u i Fast R-CNN-u. Kao *bacbone* se često koriste konvolucijske mreže koje su već istrenirane za klasifikaciju nad nekim skupom podataka, što znači da je krajnji izlaz iz tih mreža odgovarajuća klasa. Kako mape značajki zapravo predstavljaju rezultate operacije konvolucije, da bi se kao izlaz iz *backbone-a* dobila mapa značajki potrebno je odrediti sloj te mreže čiji će se izlaz u daljnjoj obradi koristiti. Izlaz iz svakog sloja predstavlja apstrakciju predanih joj informacija, što znači da se kompleksnost značajki povećava nakon svake konvolucije (npr. prvi slojevi najčešće pronalaze rubove, drugi pronalaze uzorke u njima, i sl.). U originalnom radu [4] su kao *backbone* korištene mreže ZF i VGG16 koje su već bile istrenirane nad skupom slika ImageNet [29] i koristio se izlaz njihovog petog sloja. Ali u originalnom radu nije striktno određeno da se kao *backbone* Faster R-CNN-a moraju koristiti ZF u VGG16, zbog čega se u novije vrijeme koriste druge mreže koje su manje i brže, što ih čini pogodnijim za korištenje u sustavima za rad u stvarnom vremenu (npr. MobileNetV2, ResNet-50, i druge).

3.1.2. *Region proposal network*

Kao što je ranije rečeno, Faster R-CNN je oblik detektora čiji je jedan od glavnih dijelova RPN. RPN je konvolucijska mreža koja istovremeno predviđa BB-ove objekta i sigurnost mreže da se unutar odgovarajućeg BB-a nalazi neki objekt (engl. *objectness score*). To znači da model na temelju dobivene mape značajki predlaže područja slike na kojima se najvjerojatnije nalazi odgovarajući objekt. Može se reći da RPN klasificira područja na: objekt i pozadina. Područja za koje je RPN odredio da sadržavaju objekt se zatim predaju klasifikatoru (engl. *classifier*) koji ili određuje odgovarajuću klasu objekta ili odbija područje (odnosno pretpostavlja da na tom području nema objekata od interesa). Ovime se dobiva precizniji i fleksibilniji model kojim je na slici moguće detektirati više objekata.

Mogućnost detektiranja više objekata na slici proizlazi iz činjenice da RPN promatrana područja od interesa (engl. *region of interest* ili ROI) slike određuje pomoću takozvanih *anchor box-ova*. *Anchor*-i predstavljaju fiksne BB-ove različitih dimenzija koji su rasprostranjeni kroz cijelu sliku, odnosno referentne BB-ove. Poznato je da su BB-ovi pravokutnog oblika te da postoje razni načini njihovog definiranja. Jedan od načina njihovog definiranja je korišten unutar RPN-a za definiranje *anchor*-a. Svaki *anchor* se definira pomoću četiri broja:

$\text{visina}, \text{širina}, x_{\text{centar}}, y_{\text{centar}}$. Ovim načinom se korištenjem *anchor*-a postiže da mreža uči predviđati odstupanja od *anchor*-a, odnosno vrijednosti za koliko je potrebno podesiti parametre odgovarajućeg *anchor*-a kako bi on što bolje lokalizirao objekt. Na slici 3.3. moguće je vidjeti primjer *anchor*-a. Na slici je prikazano sveukupno devet referentnih BB-ova. Crveni BB-ovi su pravokutnici čiji je omjer visine i širine: 1:1, 1:2 i 2:1. Omjeri stranica žutih i plavih BB-ova su jednaki kao i kod crvenih, a razlikuju se u ukupnoj površini na koju prikazuju. Ovako definirana skupina anchor-a se postavlja na svaki element mape značajki [30].

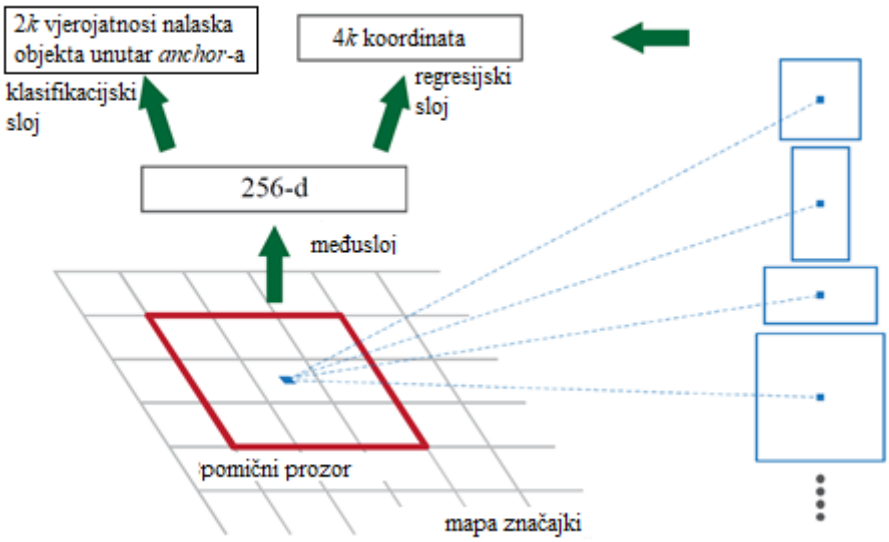


Slika 3.3. Prikaz primjera *anchor*-a. Slika preuzeta iz [30].

Dimenzije *anchor*-a je potrebno unaprijed odrediti, a postavljaju se na svaki element mape značajki dobivene *backbone*-om (vidi sliku 3.4.). Točnije rečeno, pomoću algoritma pomičnog prozora (engl. *sliding-window*) se za svaki element mape značajki vrši predikcija o postojanju objekta unutar jednog od referentnog BB-a *anchor*-a. To znači da će se za pojedinu poziciju pomičnog prozora vršiti maksimalno k predikcija, gdje k označava broj referentnih BB-ova. Kako se svaki referentni BB *anchor*-a opisuje pomoću četiri broja, tako je veličina izlaza predikcija BB-ova biti $4k$, a veličina izlaza vjerojatnosti nalazi li se unutar pripadnog referentnog BB-a objekt ili pozadina $2k$ (jedna vrijednost je vjerojatnost postojanja objekta, a druga je vjerojatnost postojanja pozadine unutar odgovarajućeg *anchor*-a).

Korištenjem *anchor*-a su autori rada [4] uspjeli osigurati iduća dva svojstva:

- Invarijantnost na translaciju (engl. *translation-invariance*) – ako se objekt u ulaznoj slici translatira, onda će se i predviđeni BB također translatirati
- *Multi-Scale Anchors* – svojstvo koje omogućava da se oslanjanjem na slike i mape značajki jedne skale mogu jednostavno vršiti klasifikacija i predikcija položaja referentnih BB-ova unutar *anchor*-a.



Slika 3.4. Prikaz RPN-a. Slika preuzeta iz [4].

3.1.3. Fast R-CNN

RPN-om se dobivaju ROI koji se predaju klasifikatoru čiji je zadatak odrediti kojoj klasi pripada mogući objekt i točnija procjena BB-a tog objekta. Osim dobivenih ROI-a, klasifikatoru se predaje i mapa značajki koja je dobivena pomoću *backbone*-a. Klasifikator zatim promatra ROI na mapi značajki te na temelju nje određuje klasu objekta i poboljšanje BB-a. Klasifikator Faster R-CNN-a je Fast R-CNN-a [4]. Sam Fast R-CNN se u suštini sastoji od ROI Pooling-a, ili ROIP, te R-CNN-a.

Kako su veličine ROI-a međusobno različite tako će i isječci mape značajki biti različitih veličina, što predstavlja problem prilikom razvoja klasifikatora. Autori [28] ovaj problem rješavaju implementiranjem ROIP-a, kojim se promatrani ROI-i svode na istu predefiniranu veličinu *NxN*. ROIP to omogućava tako što se za svaki promatrani ROI provode idući koraci [31]:

1. Promatrani ROI se dijeli na *NxN* jednakih dijelova
2. Nad svakim se dobivenim dijelom provodi *Max Pooling* kako bi se dobio izlaz iz ROIP-a, tj. pronalazi se maksimalna vrijednost svakog dijela, gdje svaka ta vrijednost predstavlja odgovarajući element izlaza.

Ovako dobivene konvolucijske mape značajki se dalje predaju R-CNN-u koji ima iduća dva zadatka:

- Klasificiranje ROI-a u klase od interesa (plus pozadina koja služi za uklanjanje loših prijedloga).
- Prilagođavanje predloženog BB-a da bolje uokviruje objekt.

3.1.4. Funkcija troška

Mreži je tijekom treniranja potrebno davati informacije o njenim performansama, odnosno o točnosti predviđanja kako bi se na temelju mogli prilagođavati njeni parametri. Zadatak pružanja tih informacija ima funkcija troška (engl. *loss function*). Funkcija troška mjeri udaljenost između vrijednosti koju je predvidjela mreža i očekivane vrijednosti izlaza. Funkcija troška Faster R-CNN mreže je optimizirana za više zadataka. Njome se kombiniraju gubitci klasifikacije i regresije BB-a. Funkcija troška Faster R-CNN mreže dana je formulom [4]:

$$L\left(\{p_i\},\{t_i\}\right) = \frac{1}{N_{klas.}}\sum_i \mathcal{L}_{klas.}(p_i, p_i^*) + \frac{\lambda}{N_{BB}}\sum_i p_i^* \cdot L_1^{smooth}(t_i - t_i^*) \tag{3-1}$$

gdje su:

- p_i – predviđena vjerojatnost mreže o tome nalazi li se unutar *i*-tog *anchor*-a objekt,
- p_i^* – jednaka je jedan ako se unutar *i*-tog *anchor*-a nalazi objekt, u suprotnom je nula,
- t_i – predviđene koordinate BB-a,
- t_i^* – označene koordinate BB-a,
- $N_{klas.}, N_{BB}$ – konstante normalizacije,
- λ – konstanta pomoću koje se postiže uravnoteženost gubitaka klasifikacije i regresije BB-a,
- $\mathcal{L}_{klas.}$ – logaritamska funkcija troška,
- $\mathrm{L}_1^{\mathrm{smooth}}$ – *smooth* L_1 gubitci.

3.2. YOLOv3

YOLO (engl. *You Only Look Once*) [3] predstavlja sustav za detekciju objekata u stvarnom vremenu. YOLO je detektor objekata koji za prepoznavanje objekata na slici koristi značajke koje su naučene pomoću DNN-a. On dobivenu sliku prvotno dijeli na više regija, a zatim za svaku regiju predviđa BB-ove i vjerojatnosti pripadnosti promatranih klasa. Dok mnoge druge do tada razvijene neuronske mreže za detekciju objekata, poput R-CNN-a, detekciju obavljaju u nekoliko koraka, YOLO ju obavlja u jednom tako što problem detekcije svodi na problem regresije (engl. *regression problem*) prostorno odvojenih BB-ova i njihovih pripadajućih vjerojatnosti pripadnosti klasa. YOLO je potpuno kovolucijska mreža (engl. *fully convolutional network*), što znači da mu se arhitektura sastoji od samo konvolucijskih slojeva i upravo ova jednostavnost arhitekture omogućava detektiranje objekata u stvarnom vremenu. Tijekom godina autori [3] su algoritam nastavili razvijati te se time pojavljuju nove poboljšane verzije YOLO-a. Jedna od njih je i YOLOv3 kojom je poboljšana detekcija objekata različitih veličina na način da uvodi poboljšanja unutar dijela mreže koji služi za izvlačenje značajki i promjene u računanju funkcije troška [3].

3.2.1. Osnovni princip rada

Kako bi se omogućilo istovremeno predviđanje BB-ova i klasa objekata, YOLO koristi značajke cijele slike. Ovo se ostvaruje tako što se slika prvotno dijeli na mrežu veličine *S x S*. Na centar svake ćelije tako dobivene mreže se postavlja *B anchor*-a, koji su objašnjeni u poglavlju 3.1.2. U originalnom radu [3] se unutar svake ćelije postavlja 3 *anchor*-a, ali broj istih se po potrebi može mijenjati. YOLO nad dobivenim slikama predviđa BB-ove. Ako se centar predviđenog BB-a nalazi unutar jedne od ćelija, YOLO pomoću *anchor*-a te ćelije mreže vrši iduće predikcije:

- Položaj i dimenzije BB-a – koriste se četiri broja: *x*, *y*, *w* i *h*; gdje (*x*, *y*) predstavljaju lokaciju centra BB-a, a (*h*, *w*) visinu i širinu BB-a. Potrebno je napomenuti kako se ta četiri broja normaliziraju u odnosu na veličinu slike radi omogućavanja primjene YOLO-a nad slikama različitih dimenzija.
- Pouzdanost – predstavlja vjerojatnost da se unutar odgovarajućeg BB-a nalazi neki objekt.
- Klasu objekta – za svaki BB daje vjerojatnost o pripadnosti pojedinoj klasi, čiji se broj označava s *C*.

To znači da se za jednu ćeliju dobiva *B*5 + C* predikcija, odnosno za cijelu sliku *S*S*(B*5+C)* predikcija. Ovime se dobiva velik broj BB-ova na slici, zbog čega velik broj njih ili

ne sadržava neki objekt ili se međusobno preklapaju. Kako bi riješili ovaj problem autori [32] na kraju mreže primjenjuju algoritam *Non-Maximum Suppression*, ili NMS, kojim se BB-ovi koji imaju IoU iznad određenog praga spajaju u jedan. Dobiveni BB se zatim unutar procesa treniranja koristi za prilagođavanje parametara mreže pomoću funkcije troška. Funkcija troška YOLOv3 mreže nije dana radom [3], već su njime opisane promjene u odnosu na funkciju troška YOLOv2 mreže [33]. Primjenjujući izmjene opisane radom [3] nad funkcijom rada troška [33] za funkciju troška YOLOv3 mreže dobiva se formula [34]:

$$
\begin{aligned}
&Ukupni\ gubitak \\
&= \lambda_{koord.} \sum_{I=0}^{SxS} \sum_{J=0}^{B} 1_{i,j}^{obj} \left[(t_x - \widehat{t_x})^2 + \left(t_y - \widehat{t_y}\right)^2 + (t_w - \widehat{t_w})^2 \right. \\
&\left. + (t_h - \widehat{t_h})^2 \right] + \sum_{i=0}^{SxS} \sum_{j=0}^{B} 1_{i,j}^{obj} \left[-log(\sigma(t_o)) + \sum_{k=1}^{C} BCE\left(\widehat{y_k}, \sigma(s_k)\right) \right] \\
&+ \lambda_{nobj} \sum_{i=0}^{SxS} \sum_{j=0}^{B} 1_{i,j}^{nobj} [-log(1 - \sigma(t_o))]
\end{aligned}
\tag{3-2}
$$

gdje su:

- SxS – ukupan broj ćelija na koje je podijeljena slika,
- B – broj predviđenih BB-ova unutar *i*-te ćelije,
- $\lambda_{koord.}, \lambda_{nobj}$ – konstante,
- $1_{i,j}^{obj}$ – jednaka jedan za *j*-ti BB ćelije *i* čija je pouzdanost o postojanju objekta najviša, u suprotnom je nula,
- t_x, t_y, t_w, t_h, t_o – predviđene koordinate i dimenzije BB-a, kao i pouzdanost mreže o postojanju objekta unutar predviđenog BB-a,
- $\widehat{t_x}, \widehat{t_y}, \widehat{t_w}, \widehat{t_h}$ – koordinate i dimenzije označenog BB-a,
- C – ukupan broj klasa od interesa,
- BCE – binarna unakrsna entropija,
- $\widehat{y_k}$ – označena klasa objekta, ako se unutar odgovarajućeg BB-a nalazi objekt klase *k*, onda će $\widehat{y_k}$ biti jednaka jedan, u suprotnom je nula,
- $1_{i,j}^{nobj}$ – jednaka jedan kada se unutar ćelije *i* ne nalazi neki objekt, u suprotnom je nula.

Moguće je primijetiti kako se formula sastoji od sljedećih dijelova [35]:

- Gubitaka lokalizacije (engl. *localization loss*) – mjeri pogreške predviđenih koordinata središta BB-a i njegove veličine. Računa se samo za okvire koji su detektirali objekt.
- Gubitaka klasifikacije (engl. *classification loss*) – ako je detektiran objekt, gubitak unutar svake ćelije se računa pomoću binarne unakrsne entropije (engl. *binary cross entropy* – BCE). BCE prvo svaku predviđenu vjerojatnost pripadnosti pojedine klase uspoređuje sa stvarnom izlaznom klasom. Nakon toga svaku predviđenu vjerojatnost „kažnjava“ na temelju udaljenosti od očekivane vrijednosti.
- Gubitaka pouzdanosti (engl. *confidence loss*) – kojim se pouzdanost mreže o postojanju objekta unutar predviđenog BB-a želi približiti nuli kada unutar odgovarajuće ćelije nema objekata.

3.2.2. Arhitektura

Gledajući s visoke razine, arhitektura YOLO-a se sastoji od dva dijela: ekstraktora značajki i detektora. Ulazna se slika prvo predaje ekstraktoru, a njegovi se rezultati (odnosno mape značajki) zatim prosljeđuju detektoru kojim se dobivaju informacije o BB-u i klasi objekta.

Ekstraktor značajki (engl. *feature extractor*) čini prvih 74 slojeva YOLOv3 detektora. Od tih 74 slojeva 53 su kovolucijska zbog čega se ekstraktor značajki YOLOv3 detektora naziva Darknet-53 (prikazan na slici 3.5.) [3]. Osim konvolucijskih slojeva, Darknet-53 sadržava i rezidualne slojeve (engl. *residual layers*) kojim se izlazi dva konvolucijska sloja (aktivacijske mape) zbrajaju i predaju idućem konvolucijskom sloju. Na kraju Darknet-53 se postavlja *Global Average* sloj sažimanja i *softmax* sloj. *Global Average* sloj sažimanja za svaku klasu generira jednu mapu značajki, nakon čega se za svaku mapu značajki izračunava njena prosječna vrijednost, a dobiveni se vektor predaje *softmax* sloju. Darkent-53 detektoru ne predaje samo jednu mapu značajki, već tri kako bi se time omogućilo detektiranje objekata različitih veličina. Mape značajki koje se prosljeđuju detektoru zapravo predstavljaju rezultate posljednjih tri rezidualnih slojeva. Sam detektor se sastoji od više konvolucijskih slojeva, gdje posljednji sloj izvršava konvoluciju filtrom veličine 1x1.

	Tip	Filteri	Veličina	Izlaz
	Konvolucijski	32	3x3	256x256
	Konvolucijski	64	3x3 / 2	128x128
1x	Konvolucijski	32	1x1	
	Konvolucijski	64	3x3	
	Rezidualni			128x128
	Konvolucijski	128	3x3 / 2	64x64
2x	Konvolucijski	64	1x1	
	Konvolucijski	128	3x3	
	Rezidualni			64x64
	Konvolucijski	256	3x3 / 2	32x32
8x	Konvolucijski	128	1x1	
	Konvolucijski	256	3x3	
	Rezidualni			32x32
	Konvolucijski	512	3x3 / 2	16x16
8x	Konvolucijski	256	1x1	
	Konvolucijski	512	3x3	
	Rezidualni			16x16
	Konvolucijski	1024	3x3 / 2	8x8
4x	Konvolucijski	512	1x1	
	Konvolucijski	1024	3x3	
	Rezidualni			8x8
	Average polling		*Global*	
	Potpuno povezani sloj		1000	
	Softmax			

Slika 3.5. Prikaz arhitekture ekstraktora značajki Darknet-53 za ulaznu sliku dimenzija 256x256

3.3. Evaluacija detektora objekata

Svaki detektor bi trebao moći lokalizirati sve objekte od interesa na slici i pravilno odrediti njihove klase. Detektori su u tome manje ili više uspješni, stoga se za evaluaciju koriste različite metrike koje upravo kvantificiraju koliko je točno navedeni zadatak učinjen za određen skup slika. Kako se na svakoj slici može nalaziti različit broj objekata različitih klasa i na drugačijim mjestima, tako ocjenjivanje rada detektora objekata nije jednostavan zadatak. Iako su tijekom godina različitim istraživanjima razvili mnogi načini za evaluaciju detektora (odnosno različite matematičke formule), najčešće korištene metrike su AP i mAP su razvijene u okviru dva različita

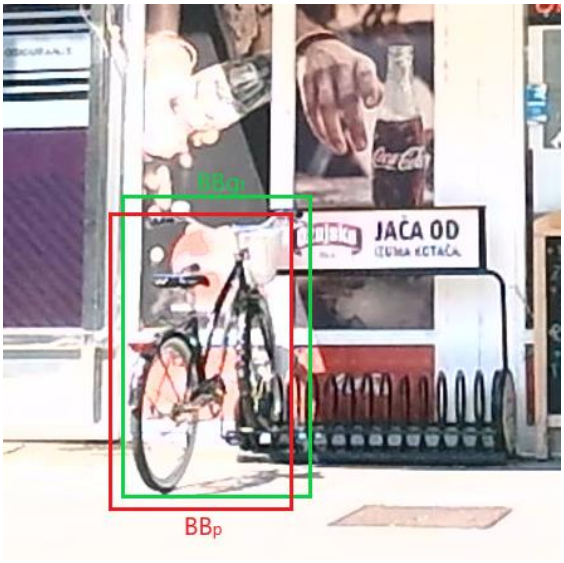
natjecanja, PASCAL VOC [14] i COCO [12]. Osim što je u obzir potrebno uzeti uspješnost detektiranja objekata, za detektore koji su razvijeni za primjenu u sustavima za rad u stvarnom vremenu je pri evaluaciji potrebno uzeti u obzir i samo vrijeme koje je potrebno za detektiranje objekata na pojedinoj slici, točnije brzinu obrade pojedine slike. Za primjenu u autonomnim vozila se najčešće smatra da brzina obrade treba biti minimalno 20 FPS-a.

Prije izračunavanja AP-a i mAP-a potrebno je odrediti iduće vrijednosti:

- Ocjenu pouzdanosti (engl. *confidence score*) – broj koji predstavlja vjerojatnost da se unutar *anchor*-a detektora nalazi neki objekt
- *Intersection over Union*, ili IoU – definiran je kao omjer između površine područja preklapanja i površine područja unije predviđenog BB-a (engl. *predicted* BB – BB_p) i stvarnog BB-a (engl. *ground-truth* BB – BB_{gt}), odnosno:

$$IoU = \frac{BB_p \cap BB_{gt}}{BB_p \cup BB_{gt}} \tag{3-3}$$

gdje BB_{gt} predstavlja površinu BB-a danog objekta koji je dan u skupu podataka (na slici 3.6. prikazano zelenom bojom), a BB_p površinu BB-a kojeg je detektor za dani objekt predvidio (na slici 3.6. prikazano crvenom bojom).



Slika 3.6. Primjer BB_{gt}-a i BB_p-a

Osim toga, za određivanje AP-a i mAP-a potrebno je izračunati i preciznost (engl. *precision*) i odziv (engl. *recall*) detektora. Kako bi se navedene metrike mogle izračunati, potrebno

je definirati broj stvarno pozitivnih rezultata (engl. *true positive*, TP). Predikcija detektora se smatra TP-om ako su redom zadovoljena iduća tri uvjeta [36]:

1. Vrijednost ocjene pouzdanosti je veća od predefiniranog praga
2. Klasa BB_p je jednaka klasi BB_{gt}
3. Vrijednost IoU-a je veća od predefiniranog praga.

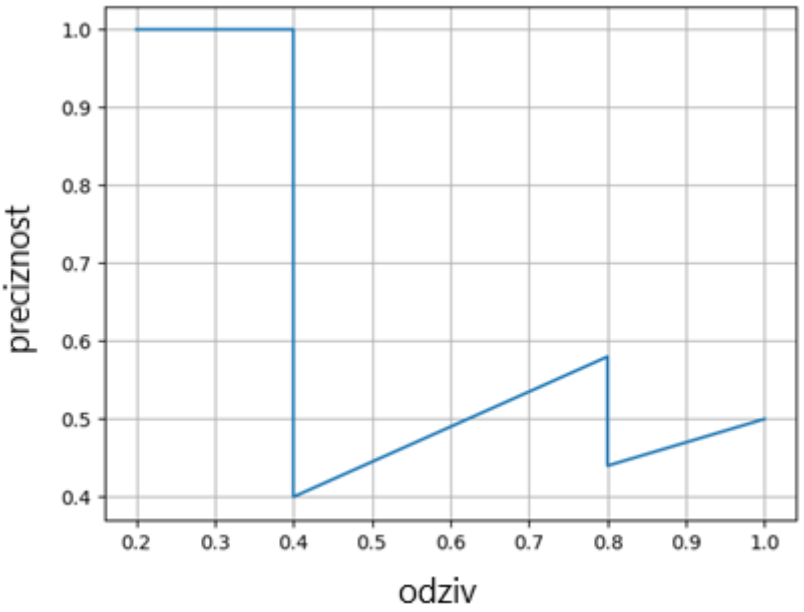
Ako je prvi uvjet ispunjen i jedan od druga dva nije, onda se predikcija detektora smatra lažno pozitivnom (engl. *false positive*, FP). Ako je detektor dao predikciju da se na odgovarajućem položaju na slici ne nalazi nikakav objekt, odnosno ako je ocjena pouzdanosti manja od praga, onda će se predikcija smatrati stvarno negativnom (engl. *true negative*, TN) ako uistinu na tom području nema objekta, u suprotnom se predikcija smatra lažno negativnom (engl. *false negative*, FN). Pomoću dobivenih vrijednosti se nadalje preciznost računa prema formuli 3-4, a odziv prema formuli 3-5. Preciznost (engl. *precision* – P) je mjera koja predstavlja vjerojatnost da je klasifikator točno odredio klasu detektiranog objekta. Odziv (engl. *recall* – R) predstavlja mjeru pomoću koje se otkriva koliko dobro klasifikator prepoznaje određenu klasu.

$$P = \frac{TP}{TP + FP} \tag{3-4}$$

$$R = \frac{TP}{TP + FN} \tag{3-5}$$

3.3.1. AP

Prosječna preciznost (engl. *average precision* – AP) je mjera koje se koristi za evaluaciju modela strojnog učenja, a time i modela detekcije objekata. Njome se za dani detektor prikazuje uspješnost točnog detektiranja pojedine klase objekta, što znači da se za svaku klasu izračunava pojedinačno. Izračunavanje AP-a se temelji na krivulji preciznosti i odziva (engl. *precision/recall curve*). Da bi se dobila krivulja preciznosti i odziva potrebno je za različite vrijednosti praga pouzdanosti izračunati preciznost i odziv. Tako dobivene vrijednosti se crtaju na graf na način da se preciznost postavlja na os ordinata, a odziv na os apscisa. Također, praksa je da se vrijednosti crtaju tako da s lijeva na desno vrijednost praga pouzdanosti smanjuje. Na slici 3.7. prikazan je primjer krivulje preciznosti i odziva. Moguće je primijetiti kako se vrijednost odziva povećava smanjenjem praga pouzdanosti, dok vrijednost preciznosti može varirati od praga do praga, ali načelno opada [36].



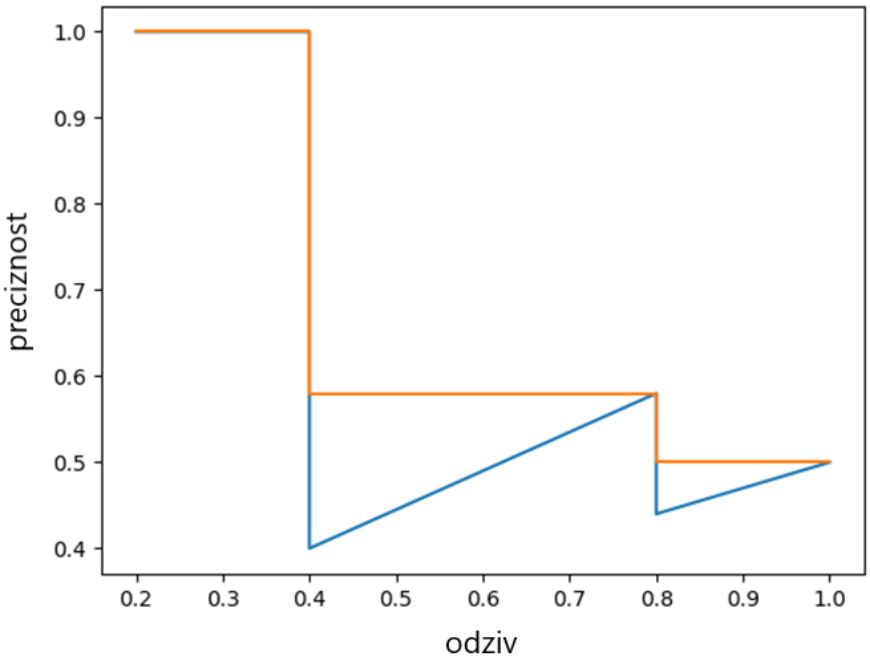
Slika 3.7. Primjer krivulje preciznosti i odziva. Slika preuzeta iz [36].

Nakon što je dobivena krivulja preciznosti i odziva, potrebno je istoj interpolirati preciznost. Interpolacija preciznosti se radi prema [36]:

$$p_{interp}(r) = \max_{r' \geq r} p(r') \tag{3-6}$$

gdje *p* označava preciznost na razini odziva *r* pri kojoj se vrši interpolacija. Iz formule 3-6 slijedi da se interpolacija vrši tako što se trenutnoj razini odziva *r* pridružuje ona preciznost koja je lokalni maksimum gledajući graf s lijeva na desno. Primjer prikaza interpolacije na grafu moguće je vidjeti na slici 3.8. Potrebno je napomenuti kako postoji više načina biranja razine odziva pri kojoj će se interpolirati preciznost:

- Biranjem 11 jednako udaljenih razina odziva; u ovom slučaju će vrijednosti razina odziva biti: 0.0, 0.1, 0.2, … , 1.0.
- Biranjem 101 jednako udaljenih razina odziva, kako je predstavljeno u okviru natjecanja COCO.
- Biranjem svih razina odziva kojima su podatci predstavljeni (prikazano na slici 3.8.).



Slika 3.8. Prikaz interpolacije (narančasta linija) preciznosti uzimajući u obzir svaku razinu odziva. Slika preuzeta iz [36].

Nakon dobivanja interpolirane krivulje preciznosti i odziva računa se AP na način da se izračunava površina ispod dobivene krivulje. To se postiže pomoću formule [36]:

$$AP = \sum_{i=1}^{n-1} (r_{i+1} - r_i) p_{interp}(r_{i+1}) \tag{3-7}$$

gdje r_1, r_2, …, r_n predstavljaju razine odziva na kojima se vrši interpolacija preciznosti, i to tako da je r_1 najmanja, a r_n najveća razina.

3.3.2. mAP

Iako se AP-om dobro opisuje rad detektora za pojedinu klasu, sveukupnu točnost detektora nije moguće iz tih informacija lako vizualizirati, zbog čega se izračunava srednja prosječna preciznost. Srednja prosječna preciznost (engl. *mean average precision*, ili mAP) predstavlja mjerilo kojim se dobiva uvid u točnost modela nad svim klasama. Računa se tako što se zbrajaju svi AP-ovi dobiveni za svaku klasu, nakon čega se dobiveni rezultat dijeli s ukupnim brojem klasa, točnije računa se srednja vrijednost svih dobivenih AP-ova, odakle je i dobila svoj naziv. mAP se izračunava na sljedeći način:

$$mAP = \frac{\sum_{i=1}^{K} AP_i}{K} \tag{3-8}$$

gdje K označava broj klasa koje detektor može detektirati, a AP_i prosječnu preciznost i-te klase.

4. KORIŠTENI SOFTVERSKI PAKETI I RAZVOJNA OKRUŽENJA

U ovom poglavlju opisani su softverski paketi i razvojna okruženja koja su korištena za treniranje YOLOv3 i Faster R-CNN detektora, proces treniranja pojedinog detektora i odabir krajnjih detektora za testiranje. Prikaz njihovih logoa korištenih softverskih paketa i razvojnih okruženja moguće je vidjeti na slici 4.1.

a) Logo Google Colaboratory-a

b) Logo Darkneta

c) Logo Detecron2 biblioteke

Slika 4.1. Korišteni softverski paketi i razvojna okruženja.

4.1. Google Colaboratory

Google Colaboratory, ili Colab, je alat kojeg razvija Google Research. Predstavlja besplatno okruženje koje svima omogućava pisanje i izvršavanje *python3* koda. Temelji se na *Jupyter* bilježnicama, točnije Colab je besplatno okruženje *Jupyter* bilježnica koji rade u oblaku. Colab pruža besplatni pristup računalnim resursima, među kojima je i grafička procesorska jedinica (engl. *graphics processing unit* – GPU). Iz tog razloga je Colab pogodan za analizu podataka i strojno učenje. On nudi nekoliko tipova GPU-a, među kojima su: NVIDIA K80, NVIDIA T4, NVIDIA P4, te NVIDIA P100. No, potrebno je napomenuti kako korisnik prilikom spajanja na okruženje ne može sam izabrati GPU kojeg želi koristiti. Time se žele postići ciljevi sprječavanja iskorištavanja resursa i postizanja ravnomjerne raspodjele resursa. Kako bi se osigurali navedeni ciljevi Colab postavlja mnoga ograničenja, kao što su na primjer [37]:

- Ograničeni vijek trajanja okruženja – varira ovisno o upotrebi, ali maksimalno može biti pokrenuto 12 sati unutar jednog dana
- Prekidanje veze s okruženjem ako je vrijeme mirovanja predugo
- Dostupnost memorije – varira ovisno o upotrebi.

Za korisnike kojima su potrebni visoki računalni resursi i koji žele imati veća i stabilnija ograničenja mogu to postići pretplatom na Colab Pro [37].

U ovom radu se zbog nedostupnosti sklopovske podrške koristi Colab kako bi se omogućilo efikasnije treniranje detektora. Zbog raznih ograničenja Colaba i velikog broja slika korištenih za treniranje (iz BDD100K skupa podataka 69863 slika za treniranje i 10000 slika za validaciju) vrijeme treniranja detektora je znatno povećano. Osim toga, kako bi se olakšao proces prenošenja trening skupa slika na Colab, skup je bilo potrebno postaviti na Google Drive.

4.2. DARKNET

Darknet je *framework* otvorenog koda pisan u C-u i CUDA-i [38]. Razvijen je isključivo za neuronske mreže, te podržava izvođenje i na GPU i središnja procesorska jedinica (engl. *central processing unti* – CPU). Darknet sadržava YOLO, a ima i mogućnost izvođenja RNN-a (engl. *recurrent neural networks*) koje se koriste za procesiranje sekvenci podataka (govor, tekst i sl.). Također, korisnicima omogućava i korištenje neuronskih mreža u video igrama, a značajkom *Nightmare* omogućava unazadno pokretanje neuronskih mreža [38]. Darknet je moguće instalirati uz dvije opcionalne ovisnosti [38]:

- OpenCV – *computer vision* biblioteka s veliki brojem funkcija za obradu slike
- CUDA – platforma kojom se omogućava izvođenje na GPU (isključivo za NVIDIA-u).

Iako je testiran samo na Linux-u i MAC OS-u, moguće je pronaći razne *tutorial*-e za instaliranje na Windows, kao što je [39].

4.2.1. Treniranje YOLOv3 detektora

Darknet je u okviru ovog rada korišten za treniranje YOLOv3 detektora. Detektor se trenirao na Colabu, a korištena *.ipynb* bilježnica nalazi se u prilogu P.4.1. Mreža je trenirana pomoću Tesla T4 GPU-a od 15109 MB. U tablici 4.1. moguće je vidjeti kratki opis konfiguracije YOLOv3 mreže za treniranje, dok cjelokupna korištena konfiguracija YOLOv3 mreže dana u prilogu P.4.2. Veličina *batch*-a od 64 slika je izabrana kako bi se mreža istrenirala na što većem

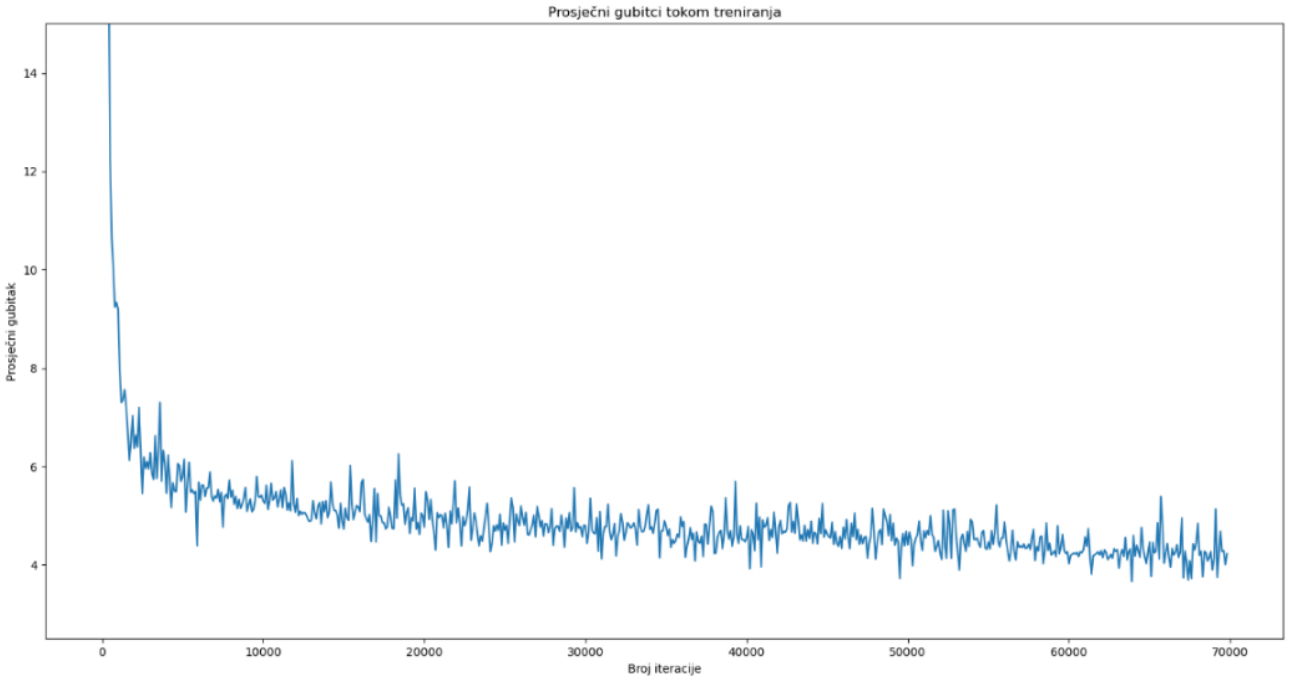
broju slika, i jer dostupni resursi to dozvoljavaju. Što su dimenzije ulaznih slika mreže veće, to će detekcija biti uspješnija, ali vrijeme obrade jedne slike će se također povećati. Obrnuto vrijedi za manje dimenzije ulaznih slika. Kako bi se postigla veća brzina obrade i točnost detekcija za veličinu ulaznih slika mreže odabrano 640x640 piksela. Autor Darknet repozitorija [40] napominje da je za uspješno treniranje mreže potrebno maksimalan broj iteracija treniranja postaviti na vrijednost koja je veća ili jednaka broju slika ukupnom broju slika trening skupa. Broj slika korištenog trening skupa jednak je 69863, i maksimalan broj iteracija je postavljen na 69863.

Tablica 4.1. Kratki prikaz konfiguracije treniranja YOLOv3 mreže. Potrebno je napomenuti kako su korištene konfiguracije koje su predložene originalnim radom [40].

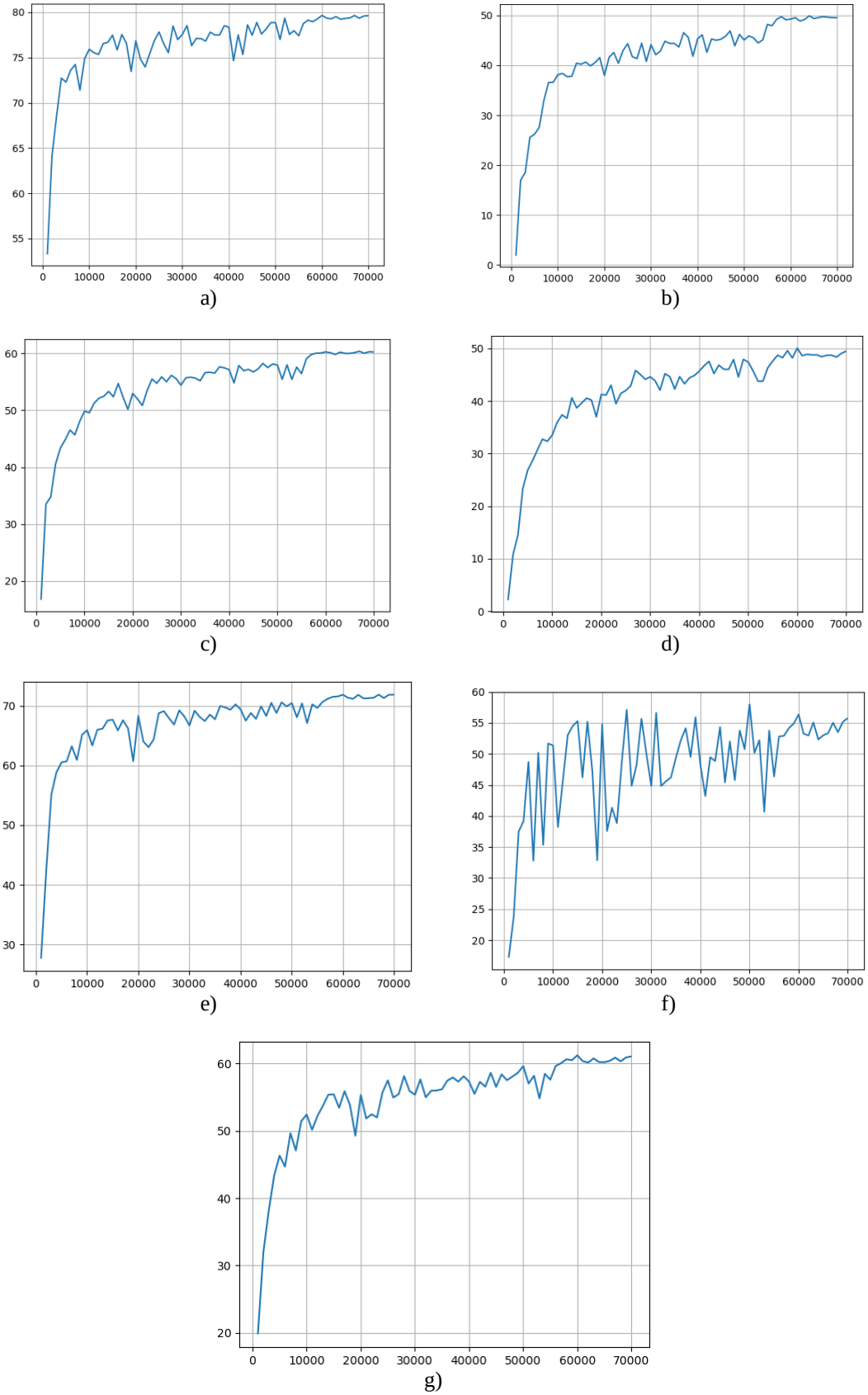
Veličina batcha	64
Dimenzije ulazne slike	640x640 piksela
Maksimalan broj iteracija	69863

Na slici 4.2. prikazan je graf mijenjanja vrijednosti funkcije troška tijekom procesa treniranja YOLOv3 detektora. Potrebno je napomenuti kako su na grafu prikazani rezultati za svakih 100 iteracija i da su podešene granice osi ordinata kako bi se dobio prikladniji prikaz rezultata. Promatrajući sliku moguće je primijetiti kako se vrijednost funkcije troška mijenja kroz proces treninga. Njena vrijednost varira od iteracije do iteracije, ali globalno se ona smanjuje. To znači da se parametri mreže sve bolje prilagođavaju trening skupu slika. Ovo može prouzrokovati *overfitting*, odnosno pretjerano usklađivanje parametara mreže na slike trening skupa. *Overfitting* predstavlja situaciju u kojoj trenirani model postiže vrhunske rezultate nad trening skupom podataka, ali nad novim podatcima postiže loše rezultate. Jedan od načina pomoću kojeg se može spriječiti *overfitting* je korištenjem validacijskog skupa kako bi se i nad njim vršila evaluacija mreže. Ako kroz više iteracija mreža nad trening skupom postiže sve bolje rezultate, a na validacijskom sve gore, onda se može zaključiti da je došlo do *overfitting*-a i proces učenja se može zaustaviti. Kako bi se spriječio *overfitting* YOLOv3 mreže u ovom je radu tijekom treniranja za dodatnu evaluaciju korišten validacijski skup. Na slici 4.3. su za svakih 1000 iteracija prikazani grafovi vrijednosti AP-a pojedine klase i mAP-a dobivene evaluacijom detektora nad validacijskim skupom. Proučavajući slike AP-a i mAP-a moguće je primijetiti kako se, kao i u slučaju vrijednosti funkcije troška, one mijenjaju kroz vrijeme. Gledajući ukupno, vrijednosti AP-a i mAP-a na validacijskom skupu rastu, a znatan skok u vrijednostima je vidljiv u 57000. iteraciji, što znači mreža sve bolje generalizira objekte od interesa. Iako je na slici 4.2. moguće primijetiti kako je više iteracija postiglo manje gubitke prilikom treniranja, za testiranje je odabrana

posljednja iteracija jer je na slici 4.3. vidljivo kako su vrijednosti AP-a i mAP za validacijske slike znatno više za nju (na primjer, uspoređujući 40000. iteraciju sa zadnjom moguće je vidjeti da ona ima manje gubitke na trening skupu, ali također na validacijskom skupu postiže i manje vrijednosti AP-a i mAP-a u odnosu na posljednju iteraciju).



Slika 4.2. Prikaz grafa promjene prosječnih gubitaka tijekom treniranja modela YOLOv3



Slika 4.3. Prikaz dobivenih rezultata AP za pojedinu klasu i mAP za validacijski skup slika. Gledajući od gore prema dolje te s lijeva na desno: a) vozilo, b) bicikl, c) pješak, d) biciklist, e) prometni znak, f) semafor, g) mAP

4.3. DETECTRON2

Detectron2 predstavlja biblioteku otvorenog koda koja pruža gotove implementacije suvremenih algoritama za detekciju objekata i segmentaciju slika [41]. Nasljednik je biblioteke Detectron, a razvija ju FAIR (engl. *Facebook AI Research*). Implementirana je u *PyTorch*-u, te podržava samo rad s CUDA-om, odnosno NVIDIA GPU-ima. Sastoji se od [41]:

- Implementacija suvremenih modela za detekciju objekata;
- Unaprijed istreniranih modela za detekciju objekata, semantičku segmentaciju, segmentaciju instanci, određivanja ključnih točaka čovjeka, i mnogo drugih;
- Podrške za popularno korištene skupove podataka, npr.: COCO, CityScapes, PASCAL VOC, LVIS;

zbog čega se koristi u raznim projektima računalnog vida, ponajviše unutar *Facebook*-a. Osim što pruža već gotove implementacije raznih modela, Detectron2 pruža i mogućnost njihovih izmjena, odnosno nadogradnji tako što korisnicima omogućava dodavanje vlastitih modula. Ovime se istraživačima nastoji pružiti mogućnost lakše implementacije novih algoritama [41].

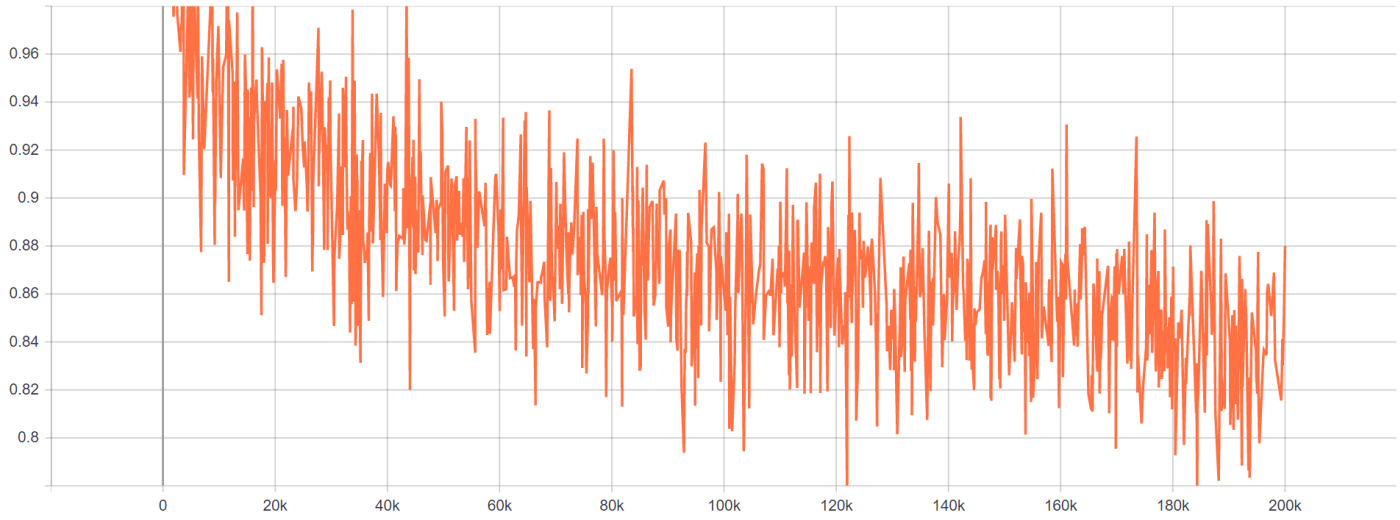
4.3.1. Treniranje Faster R-CNN detektora

Detectron2 je u ovom radu korišten u svrhu treniranja Faster R-CNN detektora. Zbog potrebe za treniranjem YOLOv3 detektora i zbog raznih ograničenja Colaba, Faster R-CNN detektor je treniran na dostupnom fakultetskom GPU-u: NVIDIA RTX 2080TI s 12GB memorije. U tablici 4.2. prikazan je kratki opis konfiguracije treniranja Faster R-CNN-a, u prilogu P.4.3. dana je *.py* datoteka koja je korištena za treniranje detektora, a cjelokupnu konfiguraciju moguće je proučiti u prilogu P.4.4. Zbog veličine memorije korištenog GPU-a za treniranje, veličina *batch*-a je postavljena na 8 slika. Ovo je znatno manja veličina *batch*-a u odnosu na YOLOv3 *batch* veličinu. Zato se maksimalan broj iteracija postavlja na 200000. Dimenzije ulaznih slika su postavljene na 640x640 piksela, radi postizanja većih brzina inferencije i većeg broja točnih detekcija.

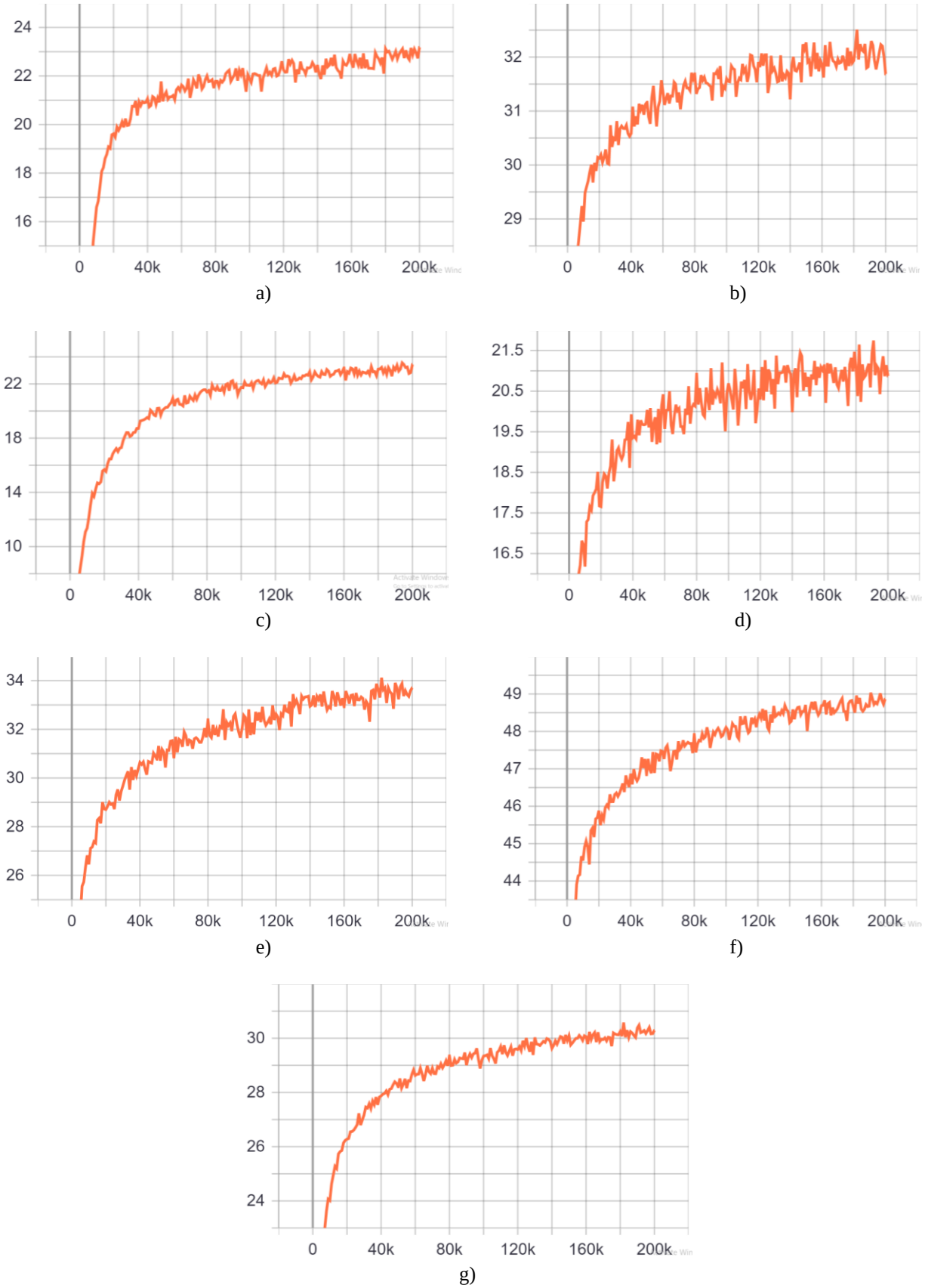
Tablica 4.2. Kratki prikaz konfiguracije treniranja Faster R-CNN mreže.

Veličina batcha	8
Dimenzije ulazne slike	640x640
Maksimalan broj iteracija	200000

Na slici 4.4. prikazan je gubitak u svakoj iteraciji treniranja Faster R-CNN detektora. Proučavajući sliku može se vidjeti kako se vrijednost funkcije troška mijenja iz iteracije u iteraciju, ali i da se, globalno gledajući, ona smanjuje. Kako bi se izbjegao problem overfitting-a, Faster R-CNN se svakih 1000 iteracija evaluirao i nad validacijskim skupom podataka, isto kao i YOLOv3 mrežE. Na slici 4.5. moguće je vidjeti AP pojedine klase, kao i ukupni mAP dobivenih evaluacijom mreže pomoću validacijskog skupa. Promatrajući sliku primjetno je da vrijednosti AP-a i mAP-a variraju iz iteracije u iteraciju, ali i da se ukupno gledajući njihove vrijednosti povećavaju, što znači da model uspješnije detektira objekte od interesa na neviđenim slikama. Gledajući samo vrijednost funkcije troška moguće je primijetiti kako detektor u 188000. iteraciji postiže manje gubitke zbog čega bi se taj detektor mogao izabrati. No, proučavanjem i prikazanih vrijednosti AP-a i mAP-a dobivenih za validacijski skup slika uviđa se da 191000. iteracija ostvaruje znatno bolje rezultate u odnosu na 188000-u iteraciju, što znači da iako 188000. iteracija ima manje ukupne gubitke, ali 191000. iteracija bolje detektira. Stoga je za daljnje testiranje odabran detektor dobiven 191000. iteracijom treniranja.



Slika 4.4. Prikaz grafa promjene prosječnih gubitaka tijekom treniranja modela Faster R-CNN. Na osi apscisa prikazan je broj iteracija, dok je na osi ordinata prikazana pripadajući gubitak iteracije.



Slika 4.5. Prikaz dobivenih rezultata AP za pojedinu klasu i mAP za validacijski skup slika. Gledajući od gore prema dolje te s lijeva na desno: a) bicikl, b) pješak, c) biciklist, d) prometni znak, e) semafor, f) vozilo, g) mAP

4.4. NVIDIA Jetson Nano

NVIDIA Jetson predstavlja obitelj ugradbenih računalnih sustava malih snaga. Razvija ih NVIDIA u svrhu ubrzanja aplikacija strojnog učenja. NVIDIA Jetson korisnicima pruža mogućnost ubrzavanja AI (engl. *artificial intelligence*) mreža, lakog uvođenja novih značajki, te korištenje istog softvera u različitim proizvodima i aplikacijama. Osim toga, zahvaljujući detaljnom dizajnu hardvera, dostupnim primjerima softvera, detaljnoj dokumentaciji, te aktivnoj zajednici se primjena NVIDIA Jetson modula unutar raznih industrija sve više povećava (npr. logistika, poljoprivreda, pametni gradovi, zdravstvo, i mnoge druge) [42]. Svaki Jetson modul podržava *cloud-native* tehnologije, kao što su izgradnja, implementacija i upravljanje AI-a na *edge*-u. Također, svaki modul koristi isti NVIDIA CUDA-X softver, a arhitektura CPU-a svakog modula se temelji na ARM arhitekturi [42]. U okviru ovog rada korišten je modul NVIDIA Jetson Nano (slika 4.6.). NVIDIA Jetson Nano pruža CSI sučelje koji se koristi za povezivanje kamere, 4 USB 3.0 sučelja koji su korišteni za: Wi-Fi adapter, bežično spajanje miša i tipkovnice, spajanje USB *stick*-a na kojem se nalaze testni skupovi. Osim toga, korištenjem HDMI priključka omogućeno je korištenje NVIDIA Jetson Nana s NVIDIA JetPack SDK-om, odnosno s grafičkim korisničkim sučeljem. Detaljne specifikacije NVIDIA Jetson Nana dane su tablicom 4.3.

Slika 4.6. Nvidia Jetson Nano

Tablica 4.3. Specifikacije NVIDIA Jetson Nana [43]

GPU	NVIDIA Maxwell arhitektura s 128 NVIDIA CUDA jezgrama
CPU	Quad-core ARM Cortex-A57 MPCore
Memorija	4 GB 64-bit LPDDR4, 1600MHz 25.6 GB/s
Pohrana	16 GB eMMC 5.1
Enkoder videozapisa	250MP/sec 1x 4K @ 30 (HEVC) 2x 1080p @ 60 (HEVC) 4x 1080p @ 30 (HEVC) 4x 720p @ 60 (HEVC) 9x 720p @ 30 (HEVC)
Dekoder videozapisa	500MP/sec 1x 4K @ 60 (HEVC) 2x 4K @ 30 (HEVC) 4x 1080p @ 60 (HEVC) 8x 1080p @ 30 (HEVC) 9x 720p @ 60 (HEVC)
Kamera	12 lanes (3x4 or 4x2) MIPI CSI-2 D-PHY 1.1 (1.5 Gb/s per pair)
Povezanost	Gigabit Ethernet, M.2 Key E
Zaslon	HDMI 2.0 i EDP 1.4
USB	4x USB 3.0, USB 2.0 Micro-B
Ostalo	GPIO, I^2C, I^2S, SPI, UART
Mehaničke informacije	69.6 mm x 45 mm konektor od 260 *edge* pinova

5. REZULTATI TESTIRANJA IZGRAĐENIH DETEKTORA

U nastavku su objašnjene postavke provođenja testiranja na računalu i na NVIDIA Jetson Nanu. Prvo se provodi testiranje oba detektora na računalu i uspoređuju se dobiveni rezultati, nakon čega se provodi testiranje na NVIDIA Jetson Nanu. Na kraju se uspoređuju rezultati rada pojedinog detektora na računalu i NVIDIA Jetson Nanu. Za testiranje su odabrane slike iz skupova podataka Cityscapes, Apolloscape i razvijeni skup podataka od 3000 slika. Oznake slika Cityscapesa se dijele na detaljne i grube, a za svrhe testiranja odabrane su detaljne oznake. Cityscapes dozvoljava pristup samo oznakama trening skupa, točnije 3475 slika, što znači da je bilo potrebno nasumično odabrati isti broj slika iz Apolloscape skupa koji dozvoljava pristup oznakama svih slika.

5.1. Testiranje izgrađenih detektora na računalu

Testiranje na računalu se provodi zbog usporedbe performansi oba detektora kada je dostupna jaka sklopovska podrška, i zbog mogućnosti usporedbe performansi s NVIDIA Jetson Nanom. Testiranje na računalu se zbog neposjedovanja odgovarajućeg GPU-a, točnije neposjedovanja NVIDIA GPU-a koji podržava Cudu, provodi na Google Colaboratory-u. Iz tog razloga performanse pojedinog detektora ovise o dodijeljenom GPU-u.

5.1.1. YOLOv3

Testiranje odabranog YOLOv3 detektora provedeno je nad Tesla T4 GPU od 15109 MB. Dobiveni rezultati testiranja su dani tablicom 5.1., dok je na slikama 5.1., 5.2. i 5.3. prikazano nekoliko primjera detektiranih objekata na slikama iz CityScapes, ApolloScape i skupa razvijenog u okviru ovog rada. Proučavanjem rezultata iz tablice moguće je primijetiti kako se za skup podataka CityScapes dobivaju znatno bolji rezultati u odnosu na ApolloScape skup podataka, npr. detektor je vozilo unutar CityScapes skupa detektiralo s prosječnom preciznošću od ~64%, dok je u ApolloScape ~33%, što je skoro dva puta manja preciznost u odnosu na CityScapes. Također je primjetno kako su rezultati dobiveni za ostale klase znatno niži u usporedbi s vozilom, točnije prosječna preciznost detektiranja prometnog znaka unutar skupa CityScape je ~37%. Jedan od mogućih razloga uspješnije detekcije vozila leži u činjenici da su veličine vozila znatno veće u odnosu na ostale objekte od interesa, poput prometnih znakova i semafora, odnosno u činjenici da je detektoru jednostavnije prepoznati veće objekte Mogući razlozi ovome najvjerojatnije leže i u nekoliko nedostataka korištenih skupova podataka:

- Unutar trening skupa se pojavljuje znatno veći broj primjera vozila u odnosu na ostale klase objekata, zbog čega mreža bolje detektira vozila
- CityScapes i ApolloScape skupovi su namijenjeni za semantičku segmentaciju; kako oba skupa sadržavaju klase poput ograde koje se označavaju kao jedna instanca (bilo da su one mrežaste, prozirne ili ne) tako se prilikom označavanja slike objekti koji se nalaze iza takvih objekata ne označavaju ili se označava samo vidljivi dio (na primjer, ako je ograda mrežasta i ako se iza nje nalazi, npr. bicikl, onda se bicikl neće označiti jer on ima manju razinu prioriteta od ograde, tj. kako je ograda bliža vozilu tako ona ima veći prioritet); ovo uveliko utječe na dobivene rezultate. Utjecaj ovog nedostatka može se primijetiti usporedbom slike 2.3. c) i 5.1. c), točnije moguće je primijetiti kako vozila iza ograde nisu označena, ali YOLOv3 je detektirao jedno od njih.
- Iako se na službenim stranicama skupa ApolloScape [44] navodi da skup sadržava oznake prometnih znakova i semafora, detaljnijim prolaskom kroz dostupne slike i njihove oznake otkriva suprotno, što predstavlja razlog zbog čega za te klase nisu dobiveni validni rezultati.
- Osim toga, daljnjim proučavanjem ApolloScape skupa primijećeno je kako su vrlo često bicikli i biciklisti skupa označeni klasom motocikl (odnosno vozilo). To je uzrokovalo vrlo loše rezultate detektiranja bicikala i biciklista.

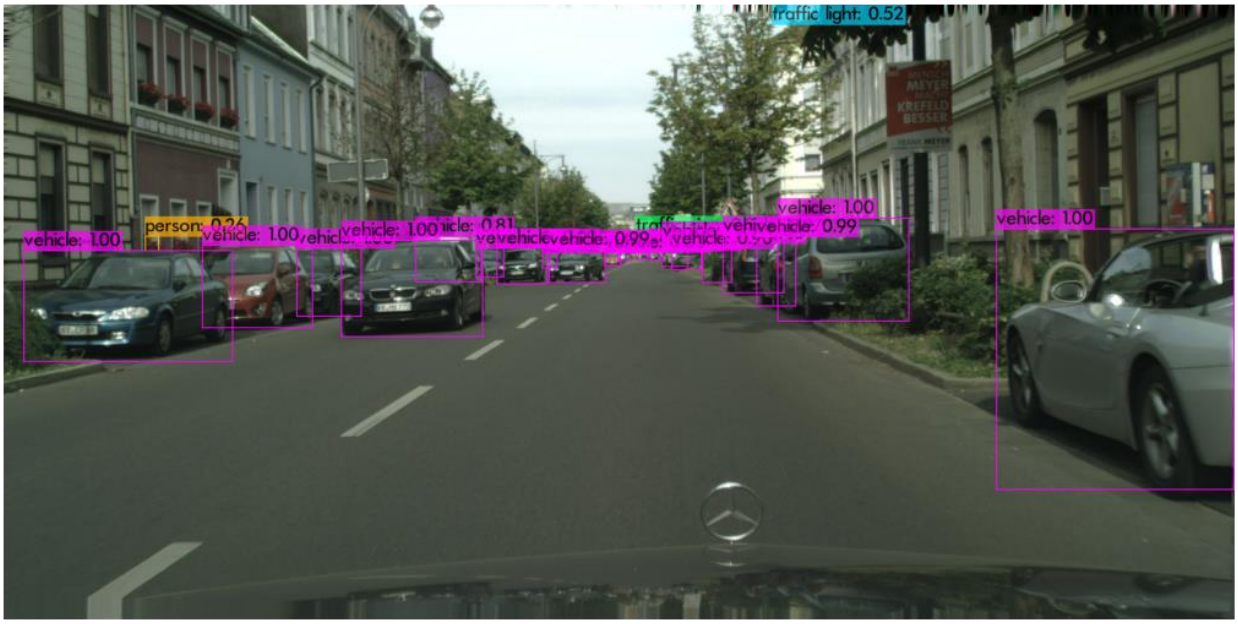
Usporedbom rezultata postignutih na vlastitom razvijenom skupu slika s rezultatima postignutim s rezultatima CityScapes i ApolloScape skupa moguće je vidjeti kako navedeni nedostatci CityScapes i ApolloScape skupa uvelike utječu na performanse detektora. Nad vlastito razvijenim skupom detektor u usporedbi s ApolloScape skupom za sve klase postiže bolje rezultate, posebice za bicikle (oko pedeset puta bolji AP) i pješake (oko sedam puta bolji AP). U odnosu na CityScapes skup detektor nad vlastito razvijenim skupom za većinu klasa postiže bolje performanse, ponajviše za semafore nad kojima je preciznost ~53%, što je 23% više u usporedbi s CityScapes rezultatima Jedino je kategorija biciklist pokazala lošije rezultate, jer se preciznost smanjila za ~11% u usporedbi s CityScapes skupom.

Za inferenciju 3475 slika iz CityScapes skupa detektoru je bilo potrebno 316 s, što znači oko 11 FPS-a, za ApolloScape 550 s, to jest oko 6.3 FPS-a, te za 3000 slika skupa razvijenog ovim radom 141 s, odnosno 21.28 FPS-a. Ovo je očekivano jer su rezolucije slika vlastito razvijenog skupa 1280x720 piksela, CityScapes skupa 2048*x*1024 piksela, a Apolloscape skupa 3384*x*2710 piksela. Promatrajući rezultate dobivenih brzina inferencije detektora nad CityScapes i

ApolloScape skupa može se zaključiti kako detektor nije primjenjiv u sustavima za rad u stvarnom vremenu. No, potrebno je naglasiti da je na brzinu uvelike utjecala dimenzije slika pojedinog skupa, što je primjetno i samom činjenicom da je za ApolloScape slike, čije su dimenzije najveće, dobiven najmanji FPS.

Tablica 5.1. Rezultati testiranja YOLOv3 na računalu.

Metrika / Skup podataka	**CityScapes**	**ApolloScape**	**Izgrađeni skup podataka**
AP – vozilo	64.64%	33.71%	75.55%
AP –bicikl	35.49%	0.84%	41.32%
AP – pješak	46.60%	7.20%	51.35%
AP – biciklist	50.63%	0%	39.96%
AP – prometni znak	37.36%	N/A	40.17%
AP – semafor	29.42%	N/A	52.95%
mAP	44.02%	10.44%	50.22%



a)

b)

c)

Slika 5.1. Objekti koje je detektirao YOLOv3 pokrenut na računalu detektirao na tri primjera slika CityScapes skupa.


vehicle: 0.97

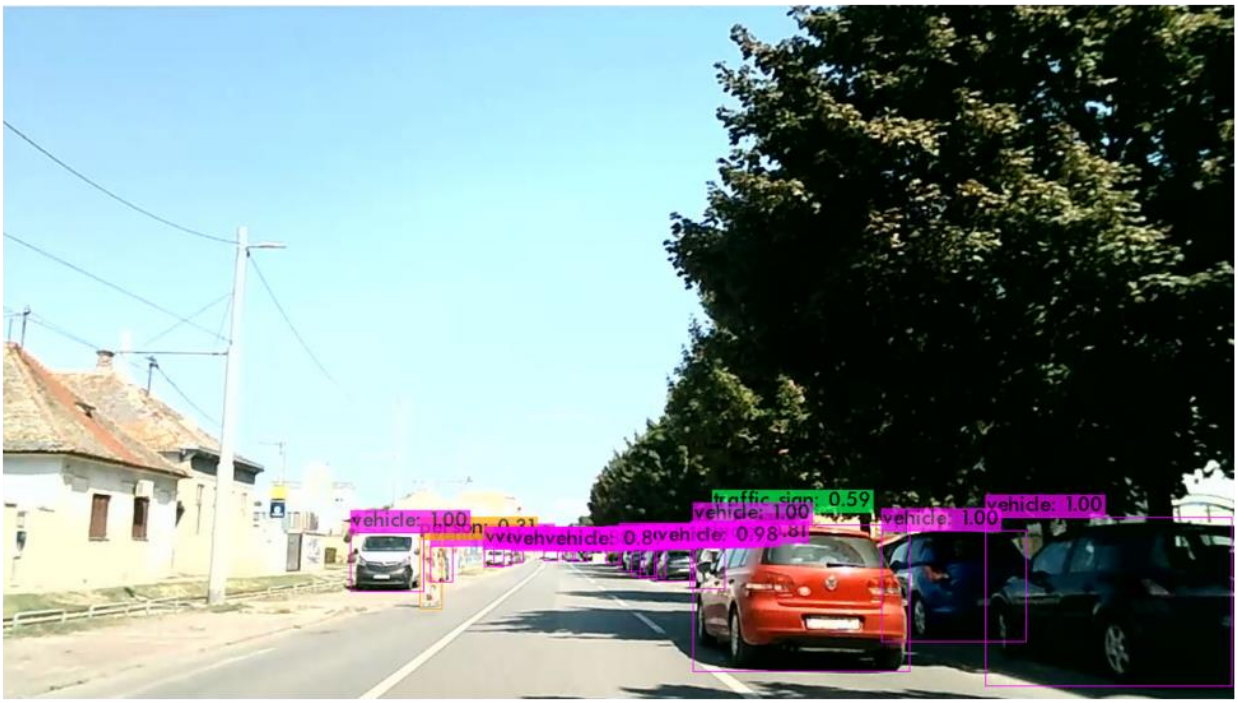
a)


traffic sign: 0.56
traffic sign: 0.74
vehicle: 0.98
traffic sign: 0.97

c)

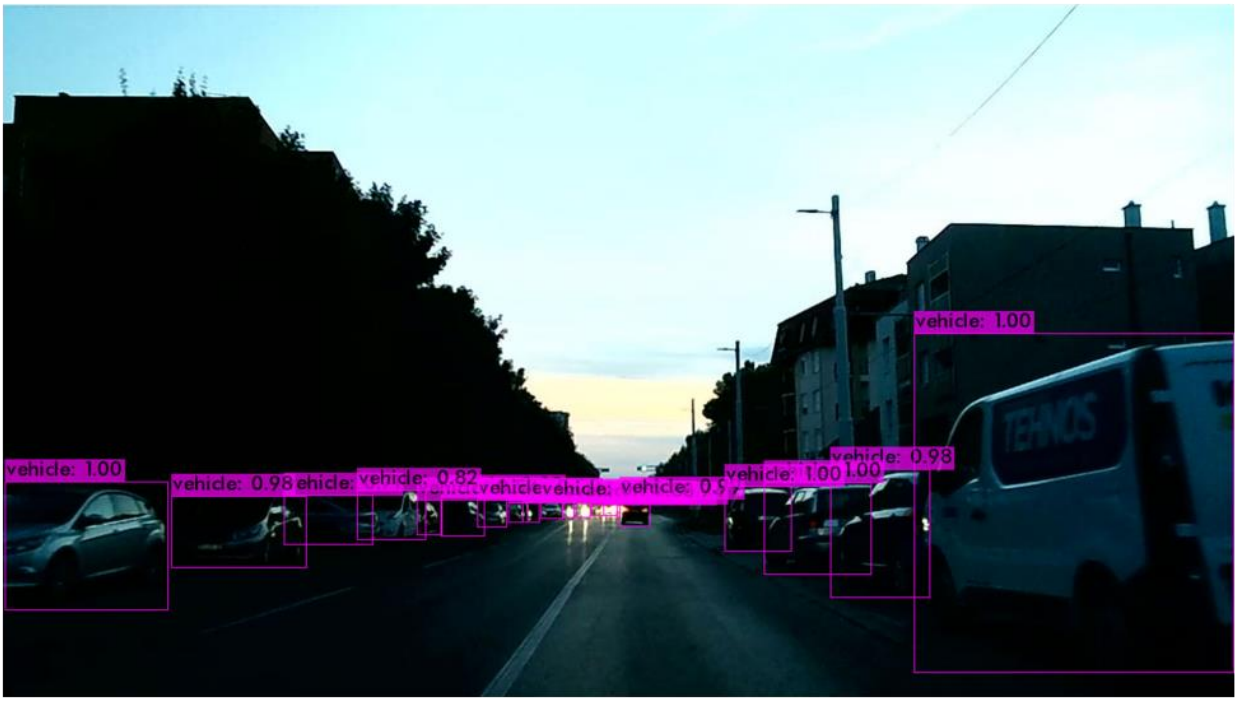
c)

Slika 5.2. Objekti koje je detektirao YOLOv3 pokrenut na računalu detektirao na tri primjera slika Apolloscape skupa.



a)

b)



c)

Slika 5.3. Objekti koje je detektirao YOLOv3 pokrenut na računalu detektirao na tri primjera slika vlastitog izgrađenog skupa.

Uspoređujući rezultate razvijenog YOLOv3 detektora s rezultatima detektora iz [6] (dani tablicom 2.1. u poglavlju 2) moguće je primijetiti kako YOLOv3 postiže znatno lošije rezultate. Dok se RRC detektorom [6] vozila detektiraju s preciznošću od skoro 90% te pješaci i biciklisti sa uspješnošću od oko 75%, dobiveni YOLOv3 detektor iste objekte detektira sa skoro dva puta manjom uspješnošću u slučaju CityScapes skupa, točnije vozila s oko 64%, pješake s oko 46% te bicikliste s oko 50% uspješnošću. Sličan je trend moguće primijetiti i usporedbom s *Ligth Network* [7] i *Gaussian* YOLOv3 [8], no u usporedbi s Waymo [9] postiže približno jednake rezultate. Ovime bi se moglo zaključiti da razvijeni YOLOv3 detektor ne pruža vrhunske rezultate, no prilikom ocjenjivanja je potrebno uzeti u obzir i činjenicu da je dobiveni YOLOv3 detektor testiran nad skupovima podataka koji su namijenjeni semantičkoj segmentaciji. Iz ovog razloga se razvijeni detektor testirao i nad samostalno razvijenim skupom podataka čime se dobivaju rezultati koji su prikazani tablicom 5.1. Nad vlastitim skupom se pokreće i *Gaussian* YOLOv3 detektor [8]. *Gaussian* YOLOv3 je u okviru rada [8] treniran za prepoznavanje deset klasa objekata od interesa, zbog čega je bilo potrebno prilagoditi izvorni kod radi dobivanja AP-a za klase od interesa ovog rada. Pokretanjem *Gaussian* YOLOv3 detektora nad vlastitim skupom podataka dobivaju se idući rezultati AP-a: vozilo 69.78%; bicikl 25.21%; pješak 35.52%; biciklist 20.56%; prometni znak 32.42%; te semafor 38.33%. Vidljivo je, kako u usporedbi s YOLOv3 detektorom, detektor rada [8] za svaku klasu daje lošije rezultate. Razlog ovome leži u činjenici da su autori [8] detektor trenirali i testirali nad istim skupom podataka, koji u sebi sadržava velik broj sličnih scena (slika) izdvojenih iz mnoštvo različitih scenarija. Uzimajući u obzir sve dobivene rezultate moguće je zaključiti kako je razvijeni YOLOv3 detektor uspješno treniran.

5.1.2. Faster R-CNN

Testiranje odabranog Faster R-CNN detektora provedeno je nad Tesla T4 GPU od 15109 MB. Tablica 5.2. prikazuje dobivene rezultate, dok su na slikama 5.4., 5.5. i 5.6. prikazani primjeri detektiranih objekata i njihovih BB-ova na primjerima slika iz CityScapes, ApolloScape i vlastito razvijenog skupa. Faster R-CNN, kao i YOLOv3 detektor, daje bolje rezultate u slučaju testiranja nad CityScapes skupom podataka u odnosu na ApolloScape. Faster R-CNN najbolje detektira vozila (~44% CityScapes, ~19% ApolloScape). Kod Faster R-CNN-a se javljaju problemi tijekom detekcije semafora i prometnih zakova (~17% prometni znak i ~11% semafor), i to iz istih razloga kao i kod YOLOv3 modela. Rezultati navedenog testiranja su također prikazani tablicom 5.2., te je usporedbom s mAP-om postignutim na skupu CityScapes i ApolloScape moguće zaključiti da detektor vrši bolje detekcije nad skupom razvijenim ovim radom. No, promatrajući vrijednosti AP-a pojedine klase vidljivo je kako samo neke klase (vozila, pješake, prometne znakove i semafore)

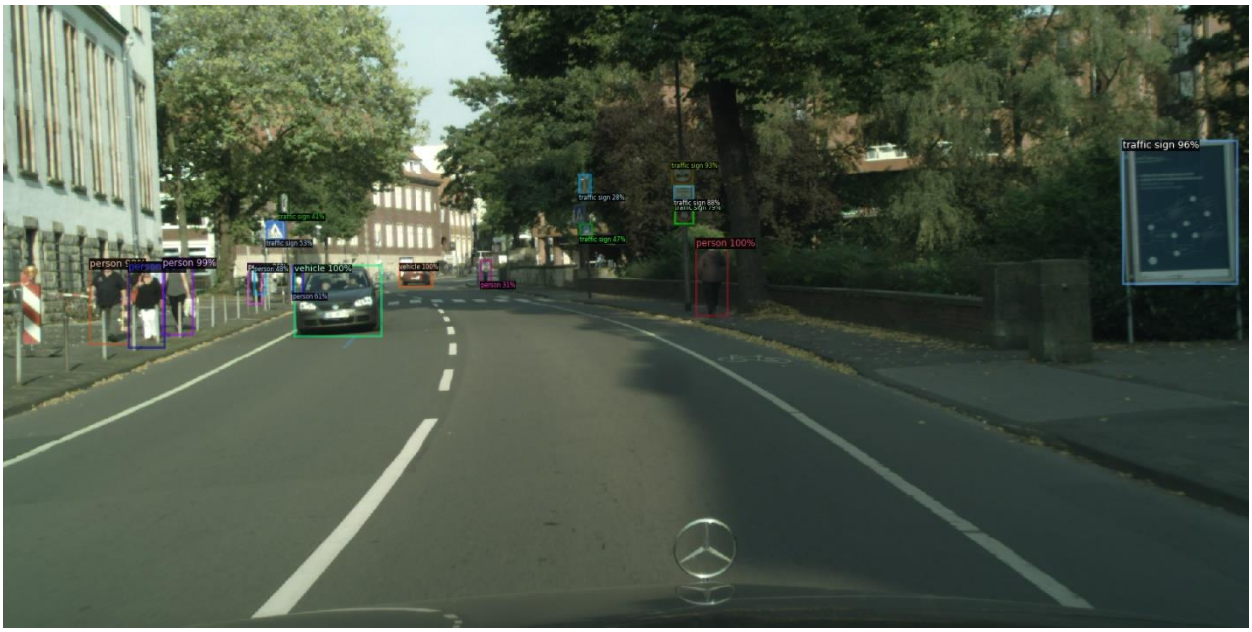
detektor uspješnije detektira u usporedbi s CityScapes testiranjem, dok ostale (biciklei bicikliste) s manjom uspješnošću.

Slično kao u slučaju testiranja YOLOv3 detektora, tako se i za testiranje Faster R-CNN detektora može zaključiti da veličina ulaznih slika uvelike utječe na vrijeme inferencije. Tako je za inferenciju skupa razvijenog ovim radom detektoru bilo potrebno 380 s, odnosno 7.9 FPS-a, CityScapes skupa 508 s, odnosno 6.8 FPS-a, te za ApolloScape 1140 s, točnije 3 FPS-a.

Tablica 5.2. Rezultati testiranja Faster R-CNN-a na računalu.

Metrika / Skup podataka	CityScapes	ApolloScape	Izgrađeni skup podataka
AP – vozilo	43.99%	19.44%	44.91%
AP –bicikl	18.72%	0.96%	13.57%
AP – pješak	28.02%	2.65%	28.25%
AP – biciklist	25.85%	0%	16.82%
AP – prometni znak	17.02%	N/A	21.78%
AP – semafor	11.16%	N/A	22.11%
mAP	24.13%	5.76%	24.57%

a)



b)

c)

Slika 5.4. Objekti koje je detektirao Faster R-CNN pokrenut na računalu detektirao na tri primjera slika CityScapes skupa.

a)

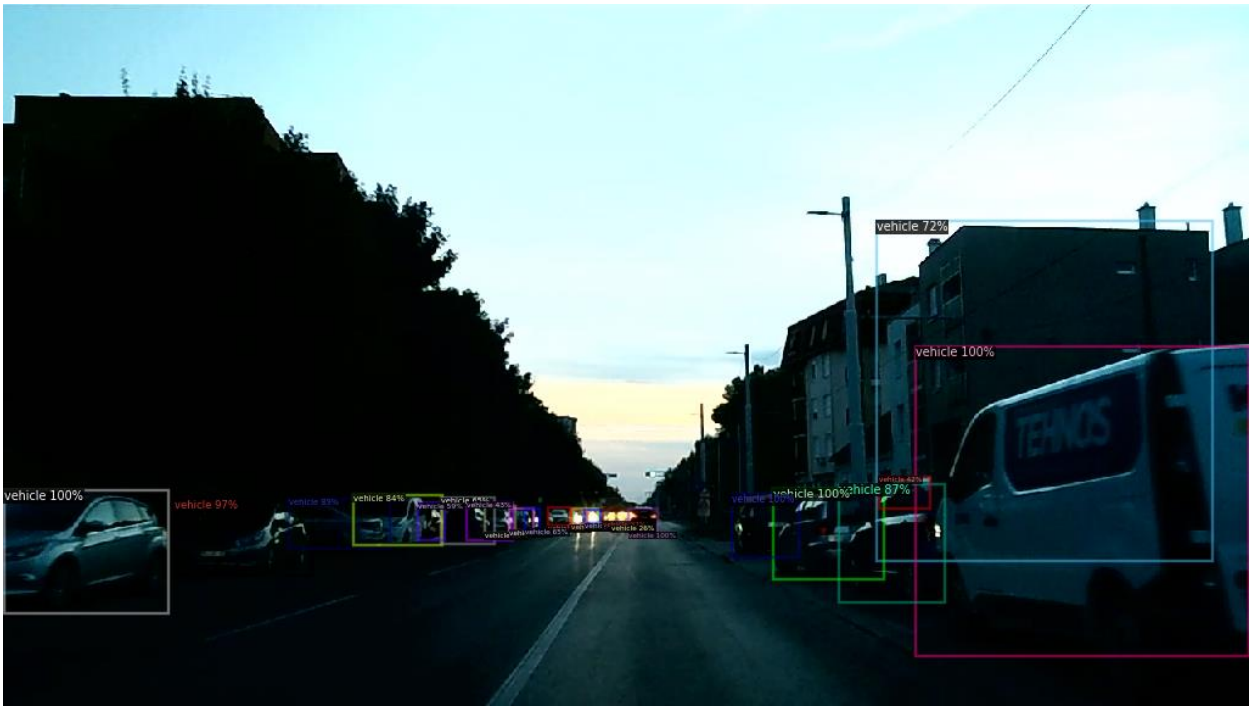
b)

c)

Slika 5.5. Objekti koje je detektirao Faster R-CNN pokrenut na računalu detektirao na tri primjera slika ApolloScape skupa.

a)

b)



c)

Slika 5.6. Objekti koje je detektirao Faster R-CNN pokrenut na računalu detektirao na tri primjera slika vlastito razvijenog skupa.

Usporedbom dobivenog Faster R-CNN detektora s detektorima prikazanim tablicom 2.1.(poglavlje 2.) primjećuje se kako Faster R-CNN postiže čak i do četiri puta manje rezultate AP-a. Dobiveni Faster R-CNN detektira objekte vozila, pješake i bicikliste s uspješnostima 43.99%, 28.02% i 25.85% redom (na slikama CityScapesa), a *Gaussian* YOLOv3 detektor [8] iste objekte redom detektira sa uspješnošću od 90.2%, 79.57% i 81.3% (na slikama BDD100K skupa), odnosno uspješnost detektiranja biciklista je čak tri puta veća. Na temelju ovih rezultata se može zaključiti kako je razvijeni Faster R-CNN detektor neuspješan, no kao što je to bio slučaj s YOLOv3 detektorom i u ovom je slučaju potrebno uzeti u obzir da Faster R-CNN detektor nije testiran nad skupom podataka namijenjen detekciji objekata. Iz tog se razloga i Faster R-CNN detektor testira nad samostalno razvijenim skupom podataka čiji su rezultati dani tablicom 5.2. Moguće je primijetiti kako se u odnosu na ApolloScape postignuti značajno bolji rezultati za svaku kategoriju, no u usporedbi s CityScapes skupom Faster R-CNN detektor nad razvijenim skupom postiže slični mAP. Usporedbom Faster R-CNN s rezultatima *Gaussian* YOLOv3 detektora dobivenih za vlastito razvijenim skupom, koji su dani u potpoglavlju 5.1.1., može se zaključiti da Faster R-CNN model postiže zadovoljavajuće rezultate.

Uspoređivanjem rezultata YOLOv3 i Faster R-CNN detektora dokazano je kako je YOLOv3 bolji detektor za dane slučajeve. YOLOv3 nad CityScapes skupom za svaku klasu postiže oko dva puta veće vrijednosti AP-a, što se odražava i na dobivene vrijednosti mAP-a za koju YOLOv3 postiže 44.02%, a Faster R-CNN 24.13%. Isti trend je moguće primijetiti i gledajući dobivene rezultate testiranja nad ApolloScape skupom (YOLOv3 postiže mAP od 10.44%, a Faster R-CNN 5.76%) i skupom razvijenim ovim radom (mAP YOLOv3 detektora je 50.22%, a Faster R-CNN-a 24.57%). Jedan od glavnih razloga zašto su dobiveni rezultati takvi je što je YOLOv3 „vidio“ veći broj slika u odnosu na Faster R-CNN, odnosno YOLOv3 je treniran s *batch*-om od 64 na skoro 70000 iteracija, dok je Faster R-CNN treniran s *batch*-om od 8 na 200000 iteracija. Osim toga, YOLOv3 za svaki testirani skup podataka postiže viši FPS, odnosno brže vrši detekcije u odnosu na Faster R-CNN. Gledajući rezultate ApolloScape skupa podataka moguće je primijetiti kako i u ovom slučaju YOLOv3 nadmašuje Faster R-CNN. No, vrijednost AP-a bicikla unutar ApolloScape skupa je za oba detektora vrlo niska. Razlog tome je činjenica da su bicikli vrlo često naslonjeni na ograde ili slično što rezultira time da u semantičkoj segmentaciji nije vidljiv njihov cijeli oblik, kako je navedeno unutar potpoglavlja 5.1.1. Iz istog razloga su i vrijednosti AP-a vozila oko dva puta manje u odnosu na CityScapes skup podataka i skup koji je razvijen ovim radom, dok su za pješaka oko trinaest puta manje. Također, na niske vrijednosti mAP-a oba detektora utječe i činjenica da se unutar odabranih testnih slika ApolloScape skupa podataka ne pojavljuje niti jedan prometni znak ni semafor.

5.2. Testiranje izgrađenih detektora na NVIDIA Jetson Nanu

U ovom potpoglavlju su predstavljeni rezultati postignuti testiranjem detektora na NVIDIA Jetson Nanu. Testovi su provedeni na isti način kao i na računalu. Potrebno je napomenuti kako je zbog dimenzija slika bilo potrebno privremeno zauzeti dodatnu memoriju za izvođenje detekcije.

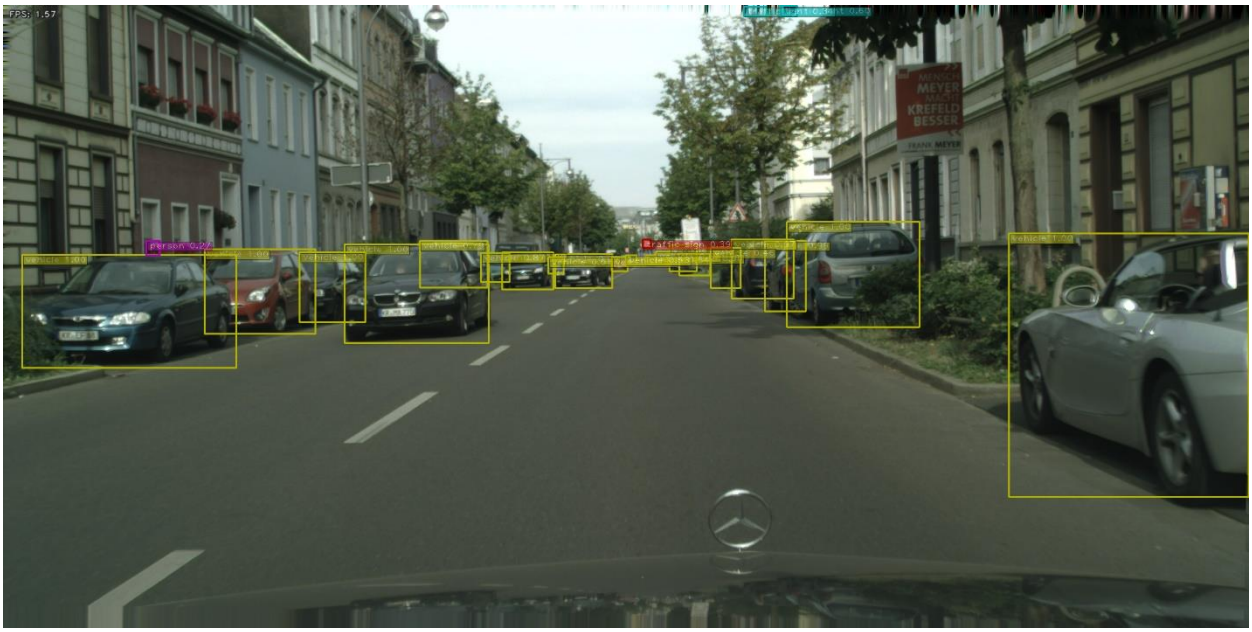
5.2.1. YOLOv3

Pokretanjem YOLOv3 detektora u originalnom okruženju na NVIDIA Jetson Nano moguće je primijetiti kako se i za CityScapes i ApolloScape i razvijeni skup podataka dobivaju identični rezultati AP-a i mAP-a kao i na računalu. Ovi rezultati su očekivani jer se detektor pokrenuo s vrijednostima težina koje su dobivene procesom učenja. No, do primjetnih razlika dolazi prilikom mjerenja vremena inferencije. Vrijeme koje je bilo potrebno za inferenciju skupa razvijenog ovim radom je 4635 s, odnosno 0.65 FPS-a, CityScapes skup 5448 s, odnosno oko 0.64 FPS-a, dok je za ApolloScape bilo potrebno 5891 s, odnosno oko 0.59 FPS-a. Zbog dimenzija slika je moguće opet primijetiti kako je za inferenciju ApolloScape skupa bilo potrebno najviše vremena.

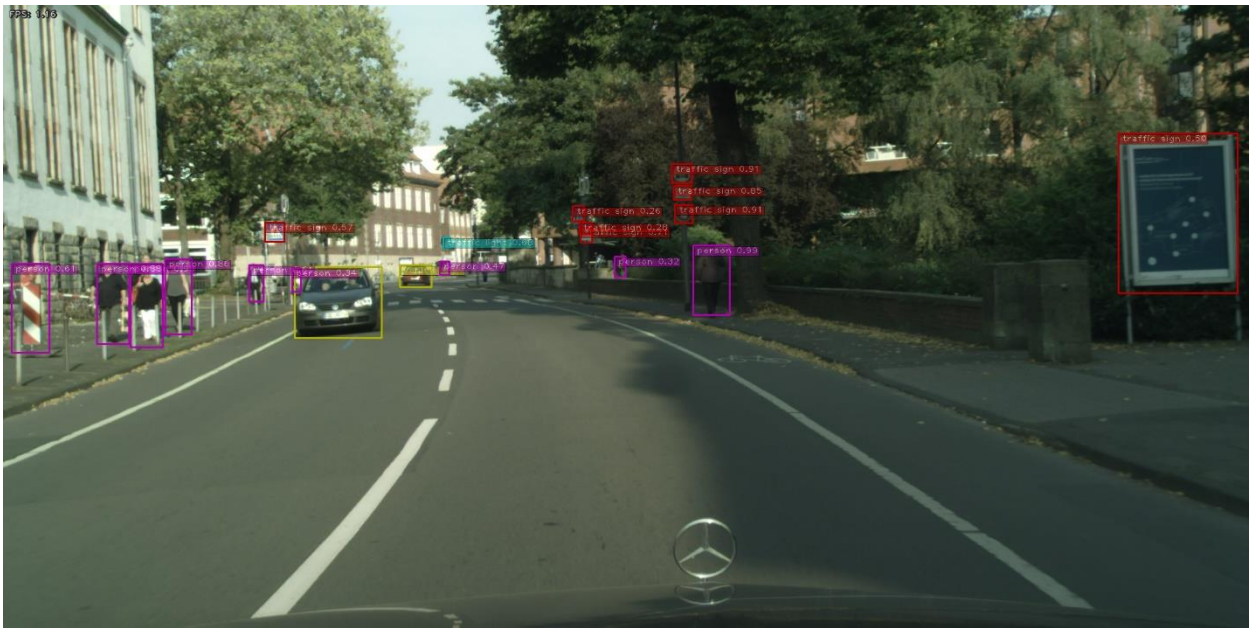
Pokretanje YOLOv3 detektora na NVIDIA Jetson Nanu rezultira vrlo niskim FPS-om što znači da nije primjenjiv u sustavima za rad u stvarnom vremenu. Inferenciju detektora moguće je ubrzati *export*-anjem treniranog modela u TensorRT, koji predstavlja SDK (engl. *software development kit*) za postizanje inferencije visokih performansi. Za *export* detektora korišten je repozitorij [45], kojim je prvo potrebno YOLOv3 detektor *export*-irati u ONNX format (format za modele strojnog učenja kojim se omogućava njihova razmjena između različitih *framework*-a), nakon čega se ONNX format *export*-ira u TensorRT. Ovom pretvorbom se brzina inferencije znatno povećala, do oko 2.5 FPS. Povećanje brzine nije bilo dovoljno kako bi detektor bio primjeren za upotrebu u sustavima za rad u stvarnom vremenu. Daljnje povećanje brzine moguće je postići i manjim dimenzijama ulaznih slika. Tablicom 5.3. su dani rezultati AP-a i mAP-a dobivenih pomoću dobivenog detektora. Moguće je primijetiti kako su dobiveni rezultati znatno manji u odnosu na originalni detektor. Razlog ovome leži u činjenici da TensorRT YOLOv3 koristi takozvani *half-precision floating-point* format, kojim se za zapis decimalnih brojeva ne koristi standardnih 32 bita, već 16 bita. Nekoliko primjera detektiranih objekata moguće je vidjeti na slikama 5.7., 5.8. i 5.9.

Tablica 5.3. Rezultati testiranja TensorRT YOLOv3 detektora na NVIDIA Jetson Nanu.

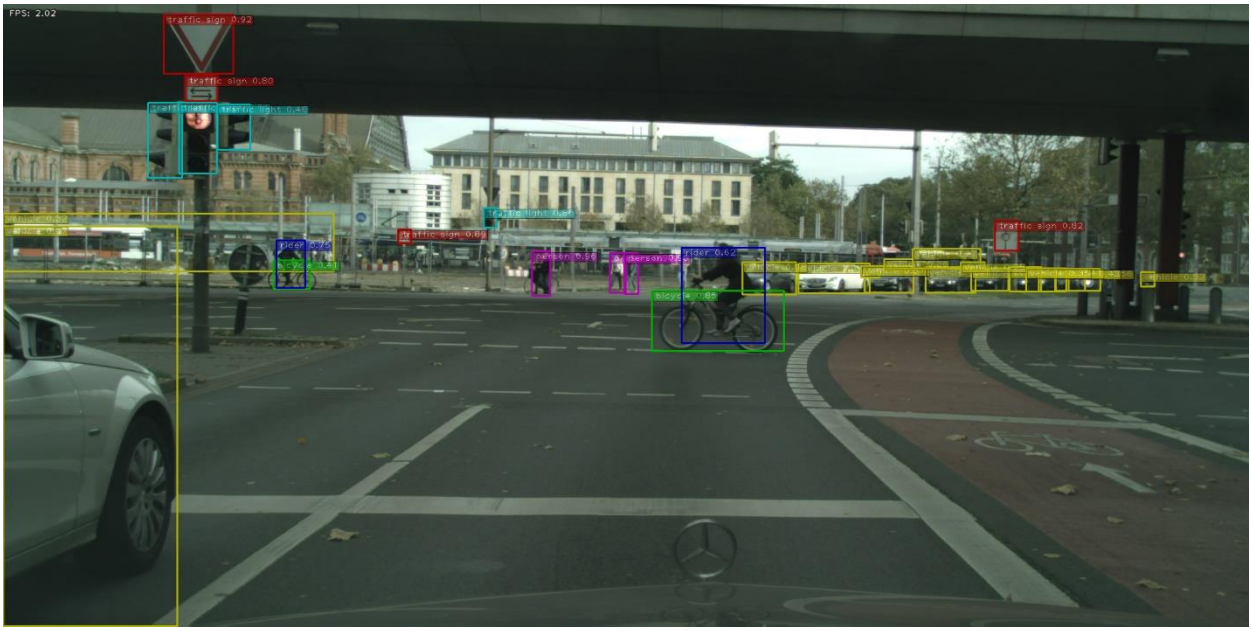
Metrika / Skup podataka	CityScapes	ApolloScape	Izgrađeni skup podataka
AP – vozilo	59.0%	31.6%	68.3%
AP –bicikl	26.8%	0.6%	22.9%
AP – pješak	38.8%	5.6%	41.6%
AP – biciklist	39.8%	0%	26.2%
AP – prometni znak	28.4%	N/A	35.1%
AP – semafor	24.4%	N/A	40.6%
mAP	36.2%	7.75%	39.12%



a)



b)



c)

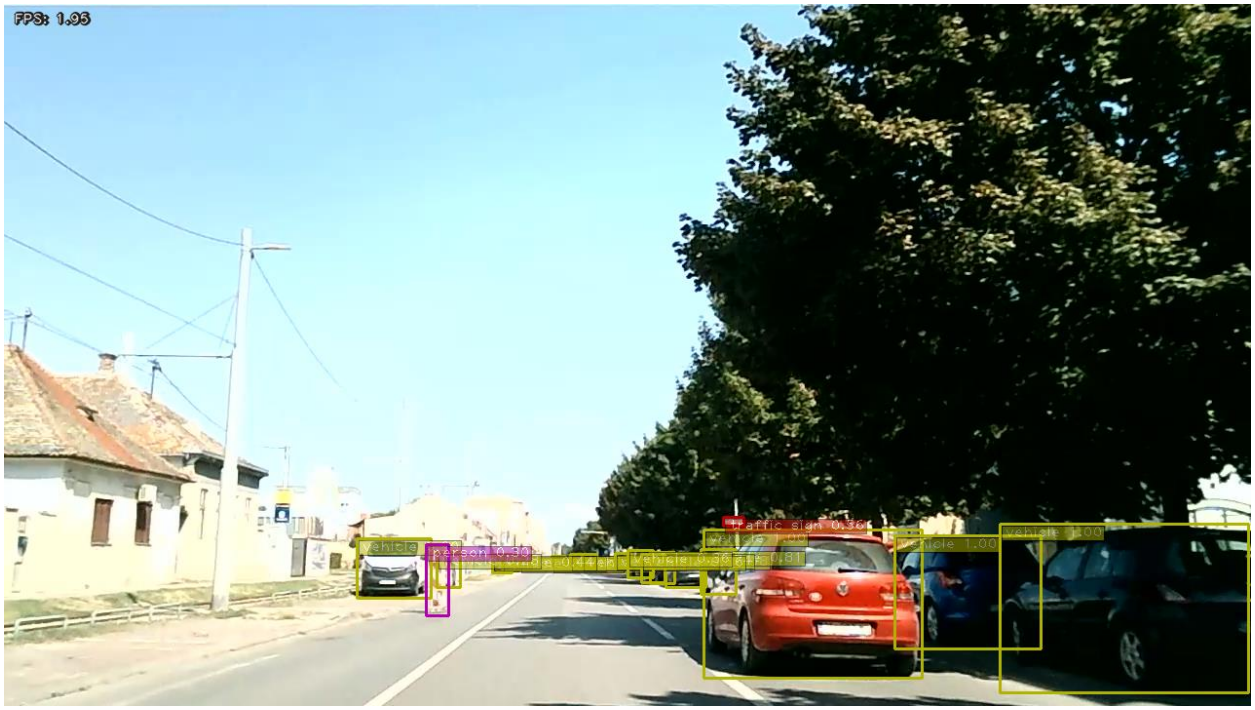
Slika 5.7. Objekti koje je detektirala TensorRT implementacija YOLOv3 modela pokrenuta na NVIDIA Jetson Nanu na tri primjera slika CityScapes skupa.

a)

b)

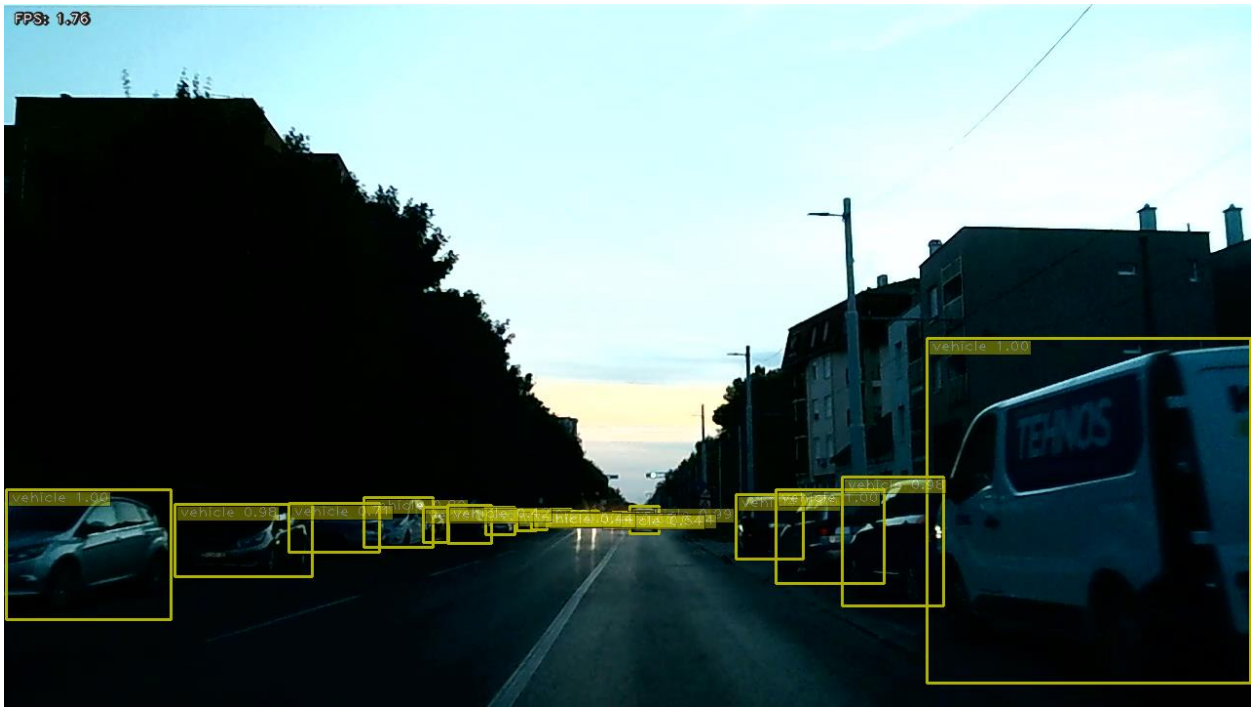
c)

Slika 5.8. Objekti koje je detektirala TensorRT implementacija YOLOv3 modela pokrenuta na NVIDIA Jetson Nanu na tri primjera slika ApolloScape skupa.



a)

b)



c)

Slika 5.9. Objekti koje je detektirala TensorRT implementacija YOLOv3 modela pokrenuta na NVIDIA Jetson Nanu na tri primjera vlastito razvijenog skupa.

5.2.2. Faster R-CNN

Faster R-CNN je pokretanjem originalnog detektora na Jetson Nanu dao identične vrijednosti AP-a pojedine klase i mAP kao i pokretanjem na računalu, što je bilo očekivano. Primjetne se razlike pojavljuju prilikom mjerenja vremena inferencije. Detektoru je na Jetson Nanu za Cityscapes skup bilo potrebno 11534 s, što je oko 0.3 FPS-a, ApolloScape skupu 16983 s, odnosno 0.22 FPS, a skupu razvijenog ovim radom 10662 s, točnije 0.28 FPS-a.

Kako su vrijednosti AP-a i mAP-a dobivene pokretanjem originalnih detektora na Jetson Nano identične rezultatima dobivenih na ralunalu, izvode se isti zaključci do kojih se došlo uspoređujući uspješnost detektiranja YOLOv3 i Faster R-CNN detektora na računalu (potpoglavlje 5.1.2.). Usporedbom brzina detektora moguće je vidjeti kako je YOLOv3 detektor i u ovom slučaju oko dva puta brži od Faster R-CNN detektora. Osim toga, ponovno se može primijetiti kako zbog veličina slika ApolloScape skupa podataka oba detektora postižu najnižu vrijednost FPS-a.

Iz dobivenih rezultata je moguće primijetiti kako brzina inferencije Faster R-CNN detektora nije dovoljna za primjenu u sustavima za rad u stvarnom vremenu, kao što je slučaj i s YOLOv3 detektorom. Iz tog razloga je originalni Faster R-CNN detektor potrebno pretvoriti u TensorRT detektor. No, arhitektura Faster R-CNN detektora je kompliciranija od YOLOv3 detektora i sadrži slojeve, odnosno matematičke operacije koje TensorRT još uvijek ne podržava, zbog čega Faster R-CNN detektor nije uspješno implementiran u TensorRT.

6. ZAKLJUČAK

Razvoj sustava s neuronskim mrežama u posljednjih nekoliko godina dovodi do naglog povećanja slojeva unutar mreže što dovodi do veće potrebe za snagom i resursima korištenog sustava. Problem proizlazi iz činjenice što ugradbeni računalni sustavi nemaju zadovoljavajuće resurse i snagu za primjenu u stvarnom vremenu. Također, isti skup podataka (podijeljen na trening, validacijski i testni skup) se vrlo često koristi za treniranje, validaciju i testiranje novo razvijenih mreža, čime se ne provjerava njihova robusnost na promjene uvjeta (npr. promjenu kamere, geografske lokacije i drugo). U radu je provjera uspješnosti izvođenja modela za detekciju objekata u stvarnom vremenu izvršena na računalu i na NVIDIA Jetson Nanu pomoću YOLOv3 i Faster R-CNN detektora. Detektori su trenirani i validirani na BDD100K skupu, a testirani na CityScapes, ApolloScape i vlastito izgrađenom skupu. Izgrađeni skup podataka dobiven je označavanjem prvog okvira svake sekunde videozapisa osječkog prometa.

Dobiveni rezultati testiranja potvrdili su utjecaj manjka resursa na performanse detektora. Pokretanjem detektora na računalu i NVIDIA Jetson Nanu dobivaju se jednake vrijednosti AP-a svake testirane klase (time mAP-a). Razlika je u vremenu potrebnom za detektiranje objekata na jednoj slici gdje je izvođenje detektora na NVIDIA Jetson Nanu dalo značajno lošije rezultate. Vrijeme inferencije YOLOv3 mreže na Jetson Nanu je do 33 puta sporije u odnosu na računalo, a Faster R-CNN mreže do 28 puta sporije. To znači da detektori u svom originalnom obliku nisu u mogućnosti zadovoljiti uvjete rada u sustavima u stvarnom vremenu. Kako bi se poboljšala brzina inferencije koristi se TensorRT SDK pomoću kojeg se ubrzavaju detektori. TensorRT ne podržava sve slojeve Faster R-CNN detektora, zbog čega on nije uspješno implementiran u TensorRT i nije se mogla izvršiti provjera njegovog mogućeg ubrzanja. YOLOv3 detektor je uspješno implementiran u TensorRT. Iako se brzina inferencije YOLOv3 mreže implementacijom u TensorRT povećala za oko 5 puta, ona i dalje nije dovoljno visoka za primjenu u sustavima za rad u stvarnom vremenu. Iz ovoga je moguće zaključiti kako YOLOv3 nije primjeren za uporabu u autonomnoj vožnji, iako na računalu postiže zadovoljavajuće rezultate.

Faster R-CNN mreža postiže značajno lošije vrijednosti AP-a (a time i mAP-a) u odnosu na YOLOv3, jer je veličina *batch*-a za Faster R-CNN bila 8 puta manja, čime je Faster R-CNN tijekom treniranja vidio manji broj slika unatoč tome što je maksimalan broj iteracija treniranja bio veći od YOLOv3 mreže. Ovo je moguće ispraviti daljnjim povećanjem maksimalnog broja iteracija treniranja Faster R-CNN-a, ili povećanjem veličine *batch*-a.

Vrijednosti AP-a i mAP-a dobivenih prilikom testiranja YOLOv3 mreže su niži u odnosu na rezultate validacije, što je s obzirom na promjene uvjeta očekivano. Kako su CityScapes i ApolloScape originalno razvijeni za semantičku segmentaciju, a vlastito izgrađeni skup za problem detekcije objekata, tako su vrijednosti AP-a i mAP-a vlastito izgrađenog skupa značajno bolje u odnosu na CityScapes i ApolloScape. Nedostatke CityScapes i ApolloScape skupa moguće je ukloniti ispravljanjem oznaka objekata unutar skupa. Za Faster R-CNN je moguće vidjeti sličan trend, odnosno rezultati dobiveni testiranjem su lošiji u odnosu na rezultate validacije. Također, rezultati izgrađenog skupa su znatno bolji u odnosu na CityScapes i ApolloScape skup. TensorRT implementacijom YOLOv3 modela se za svaki skup i svaku klasu dobivaju lošiji AP i mAP, što je očekivano jer TensorRT implementacija YOLOv3 mreže koristi half-precision floating-point format. Kako TensorRT implementacija Faster R-CNN mreže nije bila uspješna, tako nije bilo moguće testirati utjecaj korištenja half-precision floating-point formata na vrijednosti AP-a i mAP-a.

LITERATURA

[1] „Science: Radio Auto“, *Time*, kol. 10, 1925. Pristupljeno: tra. 28, 2021. [Na internetu]. Dostupno na: http://content.time.com/time/subscriber/article/0,33009,720720,00.html

[2] „What to Know Before You Get In a Self-driving Car“, *MIT Technology Review*. https://www.technologyreview.com/2016/10/18/69901/what-to-know-before-you-get-in-a-self-driving-car/ (pristupljeno tra. 28, 2021).

[3] J. Redmon i A. Farhadi, „YOLOv3: An Incremental Improvement“, *ArXiv180402767 Cs*, tra. 2018, Pristupljeno: srp. 25, 2021. [Na internetu]. Dostupno na: http://arxiv.org/abs/1804.02767

[4] S. Ren, K. He, R. Girshick, i J. Sun, „Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks“, str. 14.

[5] F. Yu *i ostali*, „BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning“, *ArXiv180504687 Cs*, tra. 2020, Pristupljeno: srp. 04, 2021. [Na internetu]. Dostupno na: http://arxiv.org/abs/1805.04687

[6] J. Ren *i ostali*, „Accurate Single Stage Detector Using Recurrent Rolling Convolution“, u *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, srp. 2017, str. 752–760. doi: 10.1109/CVPR.2017.87.

[7] H. Nguyen, „REAL-TIME VEHICLE AND PEDESTRIAN DETECTION ON EMBEDDED PLATFORMS“, *Vol 98*, izd. 21, str. 11, stu. 2020.

[8] J. Choi, D. Chun, H. Kim, i H.-J. Lee, „Gaussian YOLOv3: An Accurate and Fast Object Detector Using Localization Uncertainty for Autonomous Driving“, u *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Seoul, Korea (South), lis. 2019, str. 502–511. doi: 10.1109/ICCV.2019.00059.

[9] P. Sun *i ostali*, „Scalability in Perception for Autonomous Driving: Waymo Open Dataset“, *ArXiv191204838 Cs Stat*, svi. 2020, Pristupljeno: srp. 03, 2021. [Na internetu]. Dostupno na: http://arxiv.org/abs/1912.04838

[10] Q. Zhao, T. Sheng, Y. Wang, F. Ni, i L. Cai, „CFENet: An Accurate and Efficient Single-Shot Object Detector for Autonomous Driving“, *ArXiv180609790 Cs*, lis. 2018, Pristupljeno: srp. 04, 2021. [Na internetu]. Dostupno na: http://arxiv.org/abs/1806.09790

[11] A. Geiger, P. Lenz, i R. Urtasun, „Are we ready for autonomous driving? The KITTI vision benchmark suite“, u *2012 IEEE Conference on Computer Vision and Pattern Recognition*, Providence, RI, lip. 2012, str. 3354–3361. doi: 10.1109/CVPR.2012.6248074.

[12] T.-Y. Lin *i ostali*, „Microsoft COCO: Common Objects in Context“, *ArXiv14050312 Cs*, velj. 2015, Pristupljeno: srp. 03, 2021. [Na internetu]. Dostupno na: http://arxiv.org/abs/1405.0312

[13] „paper.pdf“. Pristupljeno: srp. 27, 2021. [Na internetu]. Dostupno na: https://ddd.uab.cat/pub/tfg/2017/tfg_71066/paper.pdf

[14] „The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Development Kit“. http://host.robots.ox.ac.uk/pascal/VOC/voc2012/htmldoc/devkit_doc.html#SECTION00054000000000000000 (pristupljeno srp. 14, 2021).

[15] Z. Che *i ostali*, „D^2-City: A Large-Scale Dashcam Video Dataset of Diverse Traffic Scenarios“, *ArXiv190401975 Cs Stat*, lip. 2019, Pristupljeno: kol. 10, 2021. [Na internetu]. Dostupno na: http://arxiv.org/abs/1904.01975

[16] „nuScenes dataset - Nuimages“. https://www.nuscenes.org/nuimages (pristupljeno kol. 10, 2021).

[17] M. A. Kenk i M. Hassaballah, „DAWN: Vehicle Detection in Adverse Weather Nature Dataset“, *ArXiv200805402 Cs*, ožu. 2020, doi: 10.17632/766ygrbt8y.3.

[18] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, i A. M. Lopez, „The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes“, u *2016*

IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, lip. 2016, str. 3234–3243. doi: 10.1109/CVPR.2016.352.
[19] M. Cordts *i ostali*, „The Cityscapes Dataset for Semantic Urban Scene Understanding“, *ArXiv160401685 Cs*, tra. 2016, Pristupljeno: kol. 10, 2021. [Na internetu]. Dostupno na: http://arxiv.org/abs/1604.01685
[20] X. Huang, P. Wang, X. Cheng, D. Zhou, Q. Geng, i R. Yang, „The ApolloScape Open Dataset for Autonomous Driving and its Application“, *IEEE Trans. Pattern Anal. Mach. Intell.*, sv. 42, izd. 10, str. 2702–2719, lis. 2020, doi: 10.1109/TPAMI.2019.2926463.
[21] G. Neuhold, T. Ollmann, S. R. Bulo, i P. Kontschieder, „The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes“, u *2017 IEEE International Conference on Computer Vision (ICCV)*, Venice, lis. 2017, str. 5000–5009. doi: 10.1109/ICCV.2017.534.
[22] J. Geyer *i ostali*, „A2D2: Audi Autonomous Driving Dataset“, *ArXiv200406320 Cs Eess*, tra. 2020, Pristupljeno: kol. 10, 2021. [Na internetu]. Dostupno na: http://arxiv.org/abs/2004.06320
[23] „What Is Object Detection?“ https://www.mathworks.com/discovery/object-detection.html (pristupljeno tra. 30, 2021).
[24] „Object Detection Guide | Fritz AI“. https://www.fritz.ai/object-detection/ (pristupljeno lip. 23, 2021).
[25] X. Jiang, A. Hadid, Y. Pang, E. Granger, i X. Feng, Ur., *Deep Learning in Object Detection and Recognition*. Singapore: Springer Singapore, 2019. doi: 10.1007/978-981-10-5152-4.
[26] C. Camacho, „Convolutional Neural Networks“. https://cezannec.github.io/Convolutional_Neural_Networks/ (pristupljeno kol. 11, 2021).
[27] R. Girshick, J. Donahue, T. Darrell, i J. Malik, „Rich feature hierarchies for accurate object detection and semantic segmentation“, *ArXiv13112524 Cs*, lis. 2014, Pristupljeno: srp. 06, 2021. [Na internetu]. Dostupno na: http://arxiv.org/abs/1311.2524
[28] R. Girshick, „Fast R-CNN“, *ArXiv150408083 Cs*, ruj. 2015, Pristupljeno: srp. 06, 2021. [Na internetu]. Dostupno na: http://arxiv.org/abs/1504.08083
[29] „ImageNet“. https://www.image-net.org/ (pristupljeno ruj. 07, 2021).
[30] H. Gao, „Faster R-CNN Explained“, *Medium*, lis. 05, 2017. https://medium.com/@smallfishbigsea/faster-r-cnn-explained-864d4fb7e3f8 (pristupljeno srp. 06, 2021).
[31] T. Grel, „Region of interest pooling explained“, *deepsense.ai*, velj. 28, 2017. https://deepsense.ai/region-of-interest-pooling-explained/ (pristupljeno lip. 28, 2021).
[32] J. Redmon, S. Divvala, R. Girshick, i A. Farhadi, „You Only Look Once: Unified, Real-Time Object Detection“, *ArXiv150602640 Cs*, svi. 2016, Pristupljeno: srp. 25, 2021. [Na internetu]. Dostupno na: http://arxiv.org/abs/1506.02640
[33] J. Redmon i A. Farhadi, „YOLO9000: Better, Faster, Stronger“, *ArXiv161208242 Cs*, pros. 2016, Pristupljeno: srp. 25, 2021. [Na internetu]. Dostupno na: http://arxiv.org/abs/1612.08242
[34] „neural networks - YOLOv3 loss function“, *Cross Validated*. https://stats.stackexchange.com/questions/380012/yolov3-loss-function (pristupljeno ruj. 13, 2021).
[35] J. Hui, „Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3“, *Medium*, kol. 27, 2019. https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088 (pristupljeno ruj. 13, 2021).
[36] N. Zeng, „An Introduction to Evaluation Metrics for Object Detection“, *NickZeng|曾广宇*, pros. 16, 2018. https://blog.zenggyu.com/en/post/2018-12-16/an-introduction-to-evaluation-metrics-for-object-detection/ (pristupljeno srp. 14, 2021).
[37] „Colaboratory – Google“. https://research.google.com/colaboratory/faq.html (pristupljeno srp. 10, 2021).

[38] „Darknet in 2021 - Reviews, Features, Pricing, Comparison“, *PAT RESEARCH: B2B Reviews, Buying Guides & Best Practices*, sij. 27, 2017. https://www.predictiveanalyticstoday.com/darknet/ (pristupljeno srp. 10, 2021).
[39] G. Mazumdar, „Guide To Darknet Installation On Windows 10 – CPU Version“, *Analytics India Magazine*, sij. 01, 2021. https://analyticsindiamag.com/guide-to-darknet-installation-on-windows-10-cpu-version/ (pristupljeno srp. 10, 2021).
[40] Alexey, *AlexeyAB/darknet*. 2021. Pristupljeno: srp. 27, 2021. [Na internetu]. Dostupno na: https://github.com/AlexeyAB/darknet
[41] „Detectron2: A PyTorch-based modular object detection library“. https://ai.facebook.com/blog/-detectron2-a-pytorch-based-modular-object-detection-library-/ (pristupljeno srp. 11, 2021).
[42] „Jetson Modules“, *NVIDIA Developer*, lis. 05, 2020. https://developer.nvidia.com/embedded/jetson-modules (pristupljeno srp. 11, 2021).
[43] „Jetson Nano“, *NVIDIA Developer*, ožu. 06, 2019. https://developer.nvidia.com/embedded/jetson-nano (pristupljeno srp. 11, 2021).
[44] „Apollo Scape“. http://apolloscape.auto/ (pristupljeno kol. 05, 2021).
[45] J. K. Jung, *tensorrt_demos*. 2021. Pristupljeno: kol. 06, 2021. [Na internetu]. Dostupno na: https://github.com/jkjung-avt/tensorrt_demos

SAŽETAK

Implementacija detektora za detektiranje najvažnijih sudionika i predmeta prometa u sustave za autonomnu vožnju je zahtjevan zadatak jer se oni sastoje od ugradbenih računalnih sustava koji imaju ograničene resurse. Osim toga, vrlo često se kod novo razvijenih mreža ne testira robusnost na promjene uvjeta. Ovim radom se YOLOv3 i Faster R-CNN mreže treniraju za detektiranje šest najvažnijih sudionika i predmeta prometa. Testiranje performansi razvijenih detektora provodi se na računalu i NVIDIA Jetson Nanu. Kako bi se testirala robusnost na promjene uvjeta, detektori se treniraju koristeći BDD100K skup podataka, a testiraju na CityScapes, ApolloScape i vlastito izgrađenom skupu podataka. Testiranja su pokazala da i YOLOv3 i Faster R-CNN ostvaruju lošije rezultate na testnim skupovima u odnosu na rezultate validacijskog skupa. Pokazalo se kako niti YOLOv3 niti Faster R-CNN implementacija na NVIDIA Jetson Nanu ne pružaju dovoljno visoku brzinu detekcije objekata za primjenu u sustavima za rad u stvarnom vremenu. Implementacijom YOLOv3 detektora u TensorRT SDK se smanjuje točnost detektiranja, ali brzina inferencije raste. Implementacija Faster R-CNN detektora u TensorRT SDK nije bila uspješna.

Ključne riječi: detekcija objekata, Faster R-CNN, NVIDIA Jetson Nano, YOLOv3

TITLE: Construction and implementation of deep neural network based object detectors on an embedded computer system

ABSTRACT

Since autonomous driving systems consist of embedded computer systems that have limited resources, implementing detectors for detecting the most important participants and objects in traffic is a challenging task. Furthermore, robustness to changing conditions of newly developed networks is not often tested. This work trains a YOLOv3 and a Faster R-CNN network to detect the six most important participants and objects in traffic. Performance testing of the developed detectors is conducted on a computer and NVIDIA Jetson Nano. Robustness testing is performed by training the detectors using the BDD100K dataset and testing them on CityScapes, ApolloScape and a self-built dataset. Compared to the results obtained with the validation set, the test results are worse. Neither YOLOv3 nor the Faster R-CNN implementation on NVIDIA Jetson Nano have provided a sufficiently high object detection rate for application in a real-time system. While the TensorRT SDK implementation of the YOLOv3 detector increased the rate of inference, the detection accuracy decreased. The implementation of the Faster R-CNN detector in the TensorRT SDK was not successful.

Keywords: Faster R-CNN, NVIDIA Jetson Nano, object detection, YOLOv3

ŽIVOTOPIS

Klaudija Mučaj je rođena 11. siječnja 1998. godine u Rijeci, Hrvatska. Prva dva razreda osnovne škole pohađa u Crikvenici, nakon čega se seli u Osijek. Ondje završava osnovnu školu, nakon čega upisuje III. gimnaziju Osijek. 2016. godine na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija upisuje preddiplomski sveučilišni studij računarstva. 2019. godine upisuje diplomski sveučilišni studij: Automobilsko računarstvo i komunikacije.

PRILOZI

P.4.1. .ipynb bilježnica korištena za treniranje YOLOv3 detektora

P.4.2. konfiguracijske datoteke YOLOv3 detektora

P.4.3. .py datoteka korištena za treniranje Faster R-CNN detektora

P.4.4. konačna konfiguracija Faster R-CNN detektora