Primjena YOLO algoritma za prepoznavanje i analizu parkirnih mjesta

Kovačević, David

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

David Kovačević

PRIMJENA YOLO ALGORITMA ZA PREPOZNAVANJE I ANALIZU PARKIRNIH MJESTA

Završni rad

Pula, 2024.

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

PRIMJENA YOLO ALGORITMA ZA PREPOZNAVANJE I ANALIZU PARKIRNIH MJESTA

Završni rad

JMBAG: 0303100702, redovni student
Studijski smjer: Informatika
Kolegij: Skladišta i rudarenje podataka
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija

Mentor: izv. prof. dr. sc. Goran Oreški
Komentor: Romeo Šajina, mag. inf.

Pula, lipanj 2024.

Sadržaj

1. SAŽETAK

U ovom radu istraživao sam primjenu YOLO (You Only Look Once) algoritma za prepoznavanje i analizu parkirnih mjesta. Uvodni dio rada obuhvaća osnovne pojmove i metode detekcije objekata, te pregled dostupnih algoritama u ovom području. Odlučio sam se za YOLO algoritam zbog njegove učinkovitosti i preciznosti. U teorijskom dijelu rada opisao sam YOLO algoritam, uključujući njegovu arhitekturu i radni princip. Posebna pažnja posvećena je YOLOv8 verziji algoritma, koja predstavlja najnoviju iteraciju ovog pristupa. Korištene su različite varijante YOLOv8 algoritma, uključujući YOLOv8s, YOLOv8n i YOLOv8m s različitim hiperparametrima treniranja. Implementacija je provedena na različitim YOLO modelima, koristeći unaprijed trenirane modele na COCO skupu podataka, kao i modele trenirane od nule, ali svi su kasnije dodatno trenirani (fine-tuning) na specifičnom skupu podataka. Validacija i testiranje provedeni su kako bi se utvrdile performanse svakog modela.

Ključne riječi: YOLO algoritam, računalni vid, detekcija objekata, parkirna mjesta

Poveznica na GitHub repozitoriji: https://github.com/Kovinejtor/Parking-Slot-Detection

SUMMARY

In this paper, I explored the application of the YOLO (You Only Look Once) algorithm for the recognition and analysis of parking spaces. The introductory part of the paper covers basic concepts and methods of object detection, as well as an overview of available algorithms in this field. I chose the YOLO algorithm due to its efficiency and accuracy. In the theoretical part of the paper, I described the YOLO algorithm, including its architecture and working principle. Special attention was given to the YOLOv8 version of the algorithm, which represents the latest iteration of this approach. Different variants of the YOLOv8 algorithm were used, including YOLOv8s, YOLOv8n, and YOLOv8m with different training hyperparameters. Implementation was carried out on various YOLO models, using pre-trained models on the COCO dataset, as well as models trained from scratch, but all were later fine-tuned on a specific dataset. Validation and testing were conducted to determine the performance of each model.

Keywords: YOLO algorithm, computer vision, object detection, parking spaces GitHub Repository Link: https://github.com/Kovinejtor/Parking-Slot-Detection

2. UVOD

2.1. Motivacija završnog rada

U današnje vrijeme, problem parkiranja predstavlja značajan izazov u urbanim sredinama. S porastom broja vozila, dostupnost parkirnih mjesta postaje ograničeno, što dovodi do povećanja zagušenja prometa, zagađenja zraka i stresa među vozačima. Tradicionalni pristupi praćenju i upravljanju parkirnim mjestima često su neučinkoviti i skupi za implementaciju. S druge strane, napredak u tehnologijama računalnog vida i umjetne inteligencije nudi nove mogućnosti za automatizaciju i optimizaciju tih procesa. YOLO (You Only Look Once) algoritam za detekciju objekata ističe se po svojoj brzini i točnosti, što ga čini idealnim rješenjem za prepoznavanje i analizu parkirnih mjesta. Motivacija za ovaj završni rad leži u potencijalu YOLO algoritma da unaprijedi sustave upravljanja parkiranjem, smanji prometnu zagušenost i poboljša iskustvo vozača.

2.2. Ciljevi završnog rada

Glavni ciljevi ovog završnog rada su:

- Razviti te usporediti više modela za prepoznavanje i analizu parkirnih mjesta koristeći YOLO algoritam**:** implementirati i prilagoditi YOLO algoritam za specifične potrebe detekcije slobodnih i zauzetih parkirnih mjesta
- Procijeniti učinkovitost razvijenih modela**:** detaljno analizirati performanse modela na njemu nepoznatom skupu podataka kako bi se utvrdila njegova točnost i brzina u stvarnim uvjetima
- Iznijeti preporuke za daljnji rad

2.3. Organizacija rada

Ovaj rad je organiziran prema sljedećim poglavljima:

- Pregled literature: prikupljanje i analiza postojećih istraživanja i publikacija na temu detekcije parkirnih mjesta, računalnog vida i YOLO algoritma. Identificiranje trenutnih radova i rješenja u ovom području.

- Prikupljanje i priprema podataka: prikupljanje relevantnih skupova podataka koji sadrže slike parkirališta te oznake o tome je li parkirno mjesta zauzeto ili slobodno
- Implementacija YOLO algoritma: prilagodba i treniranje YOLO modela za specifične zadatke detekcije parkirnih mjesta
- Evaluacija performansi: testiranje razvijenih modela na njima nepoznatom skupu podataka i evaluacija njihovih performansi u smislu točnosti, brzine i robusnosti.
- Analiza rezultata: kvalitativna i kvantitativna analiza dobivenih rezultata. Identifikacija prednosti i nedostataka razvijenih modela.
- Zaključak i smjernice za budući rad: sažimanje ključnih nalaza provedenog rada, zaključci i preporuke za buduće radove i preporuke implementacije sustava u praksi.

3. PREGLED LITERATURE

3.1. Detekcija objekta: osnovni koncepti i metode

Detekcija objekata (engl. object detection) je ključni zadatak računalnog vida koji omogućuje prepoznavanje instanci vizualnih objekata određenih klasa (poput ljudi, životinja ili bicikla) na slikama ili u videozapisima. Svrha detekcije objekata je razviti modele koji mogu pružiti osnovne informacije potrebne za primjene u računalnom vidu (engl. computer vision): "Koji objekti se nalaze gdje?". Detekcija objekata je osnovni problem računalnog vida i temelj je za mnoge druge zadatke poput segmentacije instanci i slika, opisa slika, praćenja objekata i drugih. Primjene detekcije objekata uključuju detekciju pješaka, životinja, vozila, brojanje ljudi, detekciju lica i sl [1].

Modeli za detekciju objekata mogu identificirati prisutnost i lokaciju objekta na slici ili u videozapisu. Oni mogu prepoznati više različitih objekata na istoj slici ili videozapisu. Klasifikacijski modeli identificiraju kojoj kategoriji slika pripada, dok modeli za detekciju objekata identificiraju što je na slici i gdje se nalazi svaki objekt [30].

U računalnom vidu, slike se prikazuju kao kontinuirane funkcije na 2D koordinatnom sustavu, *f(x,y)*. Slike prolaze kroz procese uzorkovanja (engl. sampling) i kvantizacije (engl. quantization), koji pretvaraju kontinuiranu funkciju slike u diskretnu mrežu

pikselnih elemenata. Računalo tada može segmentirati sliku u diskretne regije na temelju vizualne sličnosti i blizine piksela. Kroz anotacijsko sučelje, korisnici označavaju određene objekte kao regije specifičnih značajki na razini piksela. Kada se unese slika, model za detekciju objekata prepoznaje regije sa značajkama (engl. features) sličnim onima u skupu podataka za treniranje kao iste objekte. Detekcija objekata je oblik prepoznavanja uzoraka, gdje modeli prepoznaju agregate svojstava poput veličine, oblika, boje i klasificiraju regije prema vizualnim uzorcima iz ručno anotiranih podataka za treniranje [29].

Napredak u tehnikama dubokog učenja (engl. deep learning) u posljednjih nekoliko godina ubrzao je razvoj tehnologije detekcije objekata. Mreže za detekciju objekata temeljene na dubokom učenju i GPU računalna snaga značajno su poboljšale performanse detektora i praćenja objekata. Strojno učenje (engl. machine learning, ML) je grana umjetne inteligencije (engl. artificial intelligence, AI) koja uključuje učenje obrazaca iz podataka, dok duboko učenje predstavlja specijalizirani oblik strojnog učenja s višestupanjskim učenjem. Detekcija i praćenje objekata putem dubokog učenja osnova su mnogih modernih primjena računalnog vida, poput inteligentnog praćenja zdravlja, autonomne vožnje, pametnog video nadzora, detekcije anomalija, robotskog vida i drugih. Svaka AI aplikacija za vid obično zahtijeva kombinaciju različitih algoritama koji čine višestupanjsku obradu. Brzi napredak dubokih konvolucijskih neuronskih mreža (engl. convolutional neural network, CNN) i poboljšana računalna snaga GPU-ova glavni su pokretači velikog napretka u detekciji objekata temeljenoj na računalnom vidu. Glavni nedostatak detektora objekata je velika računalna zahtjevnost koja zahtijeva značajnu procesorsku snagu [1].

Detektori objekata temeljeni na dubokom učenju izvlače značajke iz ulazne slike ili video okvira. Detektor objekata rješava dva zadatka: pronalaženje objekata (možda čak i nula) i klasificiranje svakog objekta te procjenu njegove veličine s pomoću okvira za obuhvat (engl. bounding box) [1].

3.2. Pregled algoritama za detekciju objekata

Moguće je algoritme svrstati u kategoriju jednostupanjskih (engl. one-stage) algoritama ili u kategoriju dvostupanjskih (engl. two-stage) algoritama. U jednostupanjskoj detekciji objekata, ključna strategija je izravno izdvajanje značajki

(engl. features) pomoću konvolucijske neuronske mreže (engl. convolutional neural network, CNN). Te značajke se mogu dalje koristiti za klasifikaciju i lociranje objekata. Kod dvostupanjskih algoritama, model najprije izdvaja prijedloge regija, a zatim konvolucijska neuronska mreža predviđa klasifikaciju i lokalizaciju cilja. Obje strategije imaju svoje prednosti i nedostatke. Jednostupanjska metoda je vrlo brza i nije lako ometana pozadinom te uči većinu osnovnih značajki. Međutim, njezina točnost općenito je niža kada se radi o detekciji manjih objekata. Dvostupanjska metoda ima visoku točnost u prepoznavanju većine objekata, ali zahtijeva veliki skup podataka i dugo vrijeme za identifikaciju. Točnost detekcije jednostupanjskih algoritama nastavlja se poboljšavati, čak premašujući točnost dvostupanjske detekcije. Jednostupanjski algoritmi su od velike važnosti u praksi jer brzina detekcije može zadovoljiti potrebe aplikacija u stvarnom vremenu [28].

Najvažniji dvostupanjski algoritmi detekcije objekata su: RCNN i SPPNet (2014), Fast RCNN i Faster RCNN (2015), Mask R-CNN (2017), Pyramid Networks/FPN (2017) te G-RCNN (2021). Najvažniji jednostupanjski algoritmi detekcije objekata su: YOLO (2016), SSD (2016), RetinaNet (2017), YOLOv3 (2018), YOLOv4 (2020), YOLOR (2021), YOLOv7 (2022), YOLOv8 (2023) i YOLOv9 (2024) [1].

Ren, Shaoqing, i suradnici u [23] predlažu Faster R-CNN, algoritam za detekciju objekata koji koristi mrežu za prijedlog regija (engl. region proposal network, RPN) za generiranje prijedloga objekata i konvolucijsku neuronsku mrežu za klasifikaciju prijedloga i rafiniranje njihovih okvira za obuhvat. Faster R-CNN je pokazao obećavajuće rezultate u detekciji automobila na parkiralištima te može podnijeti različite rasporede parkirališta, uvjete osvjetljenja i orijentacije automobila [11].

Kaiming He i suradnici u [24] predlažu Mask R-CNN, varijanta Faster R-CNN koja također može predviđati maske objekata uz okvire za obuhvat i klasne oznake. Mask R-CNN je primijenjen na zadatke detekcije parkiranja za segmentaciju i praćenje automobila na parkiralištima. Oni se često koriste u sustavima detekcije parkiranja za prepoznavanje prisutnosti automobila na parkirnim mjestima [11].

Wei Liu i suradnici u [25] predlažu SSD (Single Shot Detector), jednofazni sustav za identifikaciju objekata koji koristi nekoliko konvolucijskih mapa značajki različitih skala za predviđanje vrste i lokacije objekata. SSD može detektirati objekte različitih veličina

i omjera stranica te je primijenjen na zadatke detekcije parkiranja s visokom točnošću i učinkovitošću [11].

Tsung-Yi i suradnici u [26] predlažu RetinaNet koji je nedavni algoritam za detekciju objekata koji koristi novu funkciju fokalnog gubitka kako bi riješio problem neravnoteže klasa u zadacima detekcije objekata. RetinaNet može detektirati objekte s visokom preciznošću i odzivom te je primijenjen na zadatke detekcije parkiranja s obećavajućim rezultatima [11].

Joseph Redmon i suradnici u [27] predlažu YOLO (You Only Look Once), popularan algoritam za detekciju objekata koji može detektirati objekte u stvarnom vremenu dijeleći sliku na mrežu ćelija i predviđajući klasu i lokaciju objekta u svakoj ćeliji. YOLO može detektirati više objekata na slici i primijenjen je na zadatke detekcije parkiranja s velikom točnošću i brzinom. Pregled YOLO algoritma te njegovih novih verzija je dan u nastavku [11].

3.3. Pregled dosadašnjih istraživanja na temu prepoznavanja parkirnih mjesta

Prepoznavanje parkirnih mjesta predstavlja izazov u urbanim sredinama gdje je dostupnost parkirnog prostora često ograničena. Dosadašnja istraživanja u ovom području uglavnom su se fokusirala na razvoj tehnologija i algoritama za prepoznavanje slobodnih parkirnih mjesta koristeći različite pristupe, uključujući obradu slike, duboko učenje, senzorske mreže i kombinaciju ovih metoda.

Jedan od pionirskih radova u ovom području je istraživanje iz 2004. godine koje su proveli **Funck i sur.** u [7], gdje su koristili kamere montirane na vozila za prepoznavanje slobodnih parkirnih mjesta s pomoću algoritama obrade slike. Značajan je u području računalnog vida i detekcije objekata jer predstavlja inovativan pristup automatskom praćenju zauzetosti parkirališta koristeći samo pojedinačne slike. Ovaj rad postavlja temelje za razvoj sustava koji mogu učinkovito i precizno detektirati prisutnost automobila na parkiralištu bez potrebe za skupim i složenim hardverskim instalacijama, kao što su senzori ili kamere s više pogleda. Postavio je temelje za kasnija istraživanja koja su integrirala naprednije metode računalnog vida.

S razvojem dubokog učenja, došlo je do značajnog napretka u točnosti i efikasnosti sustava za prepoznavanje parkirnih mjesta. **Amato i sur.** u [8], razvili su 2017. godine

sustav baziran na konvolucijskim neuronskim mrežama koji može precizno detektirati slobodna parkirna mjesta na osnovu analize snimaka nadzornih kamera. Njihov rad pokazuje kako primjena dubokog učenja može poboljšati performanse u stvarnim uvjetima.

Također, korištenje senzorskih mreža postaje sve popularnije u ovom području. **Lin i sur.** u [9], 2018. godine predstavili su sustav koji koristi senzore ugrađene u asfalt parkinga za detekciju prisutnosti vozila. Ovaj pristup omogućava kontinuirano praćenje i visoku preciznost bez potrebe za vizualnim pregledom.

Sve ove studije naglašavaju važnost razvoja naprednih tehnologija za prepoznavanje parkirnih mjesta kako bi se odgovorilo na sve veće potrebe urbanih sredina. Kontinuirana istraživanja i inovacije u ovom polju su ključne za postizanje efikasnijih i održivijih rješenja za parkiranje.

4. TEORIJSKE OSNOVE

4.1. Tehnološki aspekti YOLO algoritma

YOLO je popularan model za detekciju objekata i segmentaciju slika, a razvili su ga Joseph Redmon i suradnici na Sveučilištu Washington te je objavljen 2015. godine u [27]. YOLO je brzo stekao popularnost zbog svoje velike brzine i točnosti. **YOLOv2**, objavljen 2016., poboljšao je originalni model uključivanjem normalizacije serije (engl. batch normalization), referentnog okvira (engl. anchor box) te klastera dimenzija. **YOLOv3**, objavljen 2018., dodatno je unaprijedio performanse modela korištenjem učinkovitije mreže za kralježnicu (engl. backbone network), više referentnih okvira i prostornog piramidalnog sažimanja. **YOLOv4**, objavljen 2020. godine, uvodi inovacije poput Mosaic augmentacije podataka, nove detekcijske glave bez sidra (engl. anchor-free detection head) i nove funkcije gubitka. **YOLOv5** dodatno je poboljšao performanse modela i dodao nove značajke kao što su optimizacija hiperparametara (engl. hyperparameters), integrirano praćenje eksperimenata i automatski izvoz u popularne formate. **YOLOv6** je razvijen od strane Meituana 2022. godine i koristi se u mnogim autonomnim dostavnim robotima te kompanijama. **YOLOv7** dodao je dodatne zadatke kao što je procjena položaja (engl. pose estimation) na COCO keypoints skupu podataka. **YOLOv8** je najnovija verzija YOLO-a razvijena od strane Ultralyticsa.

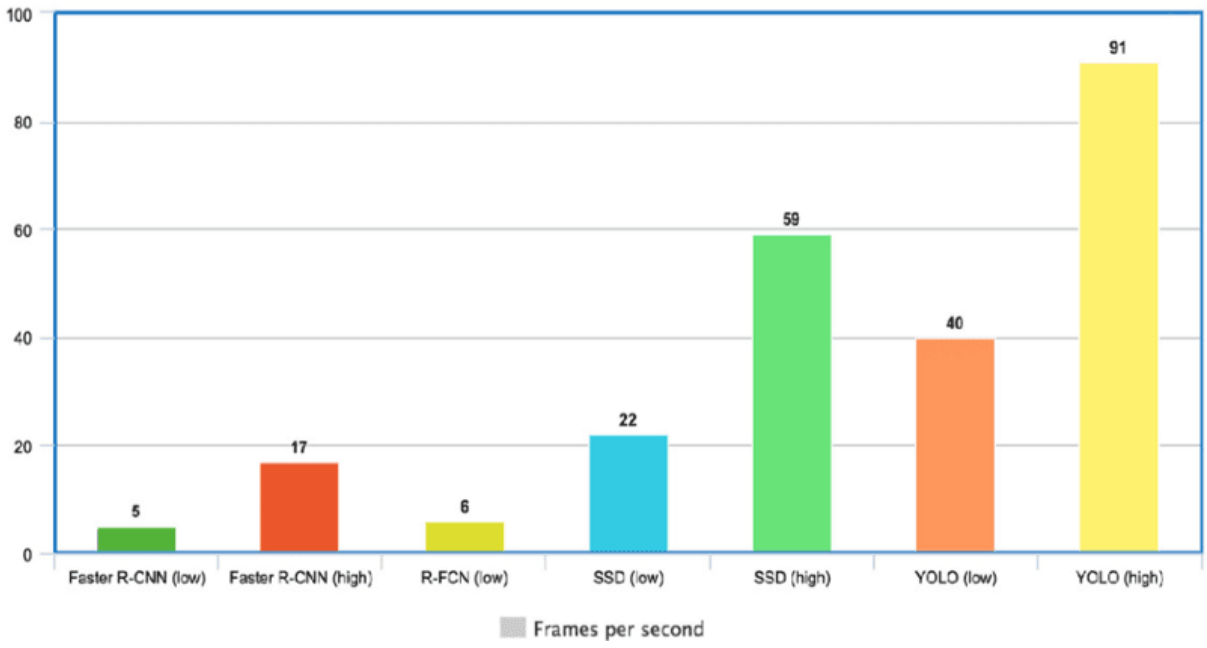
Kao vrhunski model (state-of-the-art, SOTA), YOLOv8 nadograđuje uspjeh prethodnih verzija, uvodeći nove značajke i poboljšanja za bolju performansu, fleksibilnost i učinkovitost. YOLOv8 podržava cijeli niz zadataka računalnog vida, uključujući detekciju, segmentaciju, procjenu položaja, praćenje i klasifikaciju. Ova svestranost omogućava korisnicima korištenje mogućnosti YOLOv8 u raznolikim aplikacijama i domenama. **YOLOv9** uvodi inovativne metode kao što su programabilne gradijentne informacije (engl. Programmable Gradient Information, PGI) i generalizirana efikasna mreža za agregaciju slojeva (engl. Generalized Efficient Layer Aggregation Network, GELAN). **YOLOv10** je razvijen od strane istraživača sa Sveučilišta Tsinghua koristeći Ultralytics Python paket. Ova verzija uvodi napredak u stvarnoj vremenskoj detekciji objekata uvođenjem kraj-do-kraja (engl. end-to-end) glave koja eliminira potrebu za ne-maksimalnim suzbijanjem (engl. Non-Maximum Suppression, NMS)[3].

Ultralytics nudi dvije opcije licenciranja, a to su **AGPL-3.0 licenca** i **poslovna licenca**. AGPL-3.0 licenca je licenca koju odobrava OSI, a idealna je za studente i entuzijaste jer potiče otvorenu suradnju i dijeljenje znanja. Poslovna licenca je namijenjena za komercijalnu upotrebu te ova licenca omogućuje besprijekornu integraciju Ultralytics softvera i AI modela u komercijalne proizvode i usluge, zaobilazeći zahtjeve otvorenog koda AGPL-3.0 [3].

Jedan od razloga zašto YOLO prednjači u konkurenciji uključuju brzinu, točnost detekcije, dobru generalizaciju te otvoren izvor (engl. open source). YOLO je izuzetno brz jer ne koristi složene tokove (engl. pipelines). Može obraditi slike brzinom od 45 okvira po sekundi (FPS). Osim toga, YOLO postiže više od dvostruke prosječne preciznosti (mAP) u usporedbi s drugim sustavima u stvarnom vremenu, što ga čini izvrsnim kandidatom za obradu i detekciju [19], [20].

Izvor 1: Sergio Antonio Sánchez Hernández,

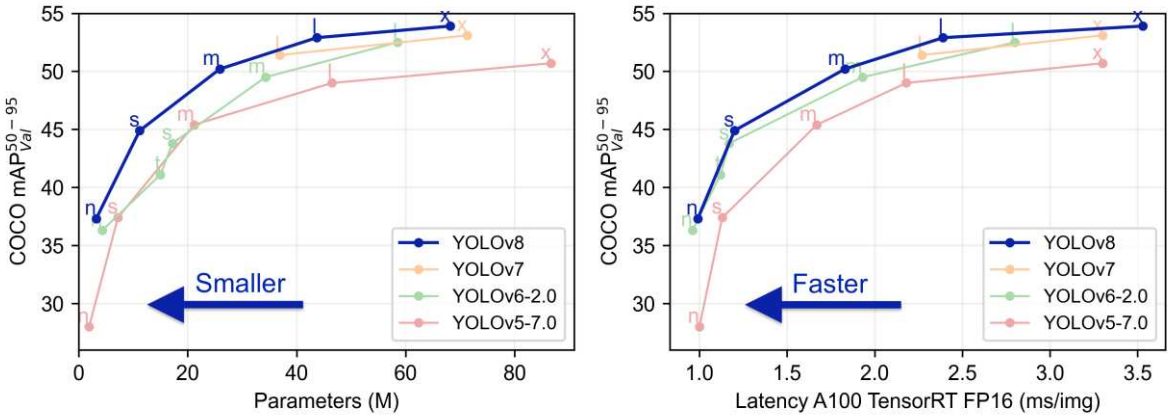
Comparison of performance metrics of pretrained models for object detection using the TensorFlow framework [20]



Slika 1: Usporedba YOLO algoritma s ostalim algoritmima detekcije objekata

YOLO je daleko ispred drugih najmodernijih modela po pitanju točnosti s vrlo malo grešaka u pozadini. S tim napretkom, YOLO je dodatno unaprijedio generalizaciju za nove domene, što ga čini izvrsnim za primjene koje se oslanjaju na brzu i robusnu detekciju objekata. Činjenica da je YOLO otvorenog koda omogućila je zajednici da stalno unapređuje model. To je jedan od razloga zašto je YOLO napravio toliko poboljšanja u tako kratkom vremenu. Među svim algoritmima za detekciju objekata odlučio sam se za YOLO jer je otvorenog izvora i praktičan pri korištenju, a za potrebe ovog rada odlučio sam se za YOLOv8 [19], [20].

Izvor 2: Ultralytics YOLO dukomentacija, YOLOv8 [20]

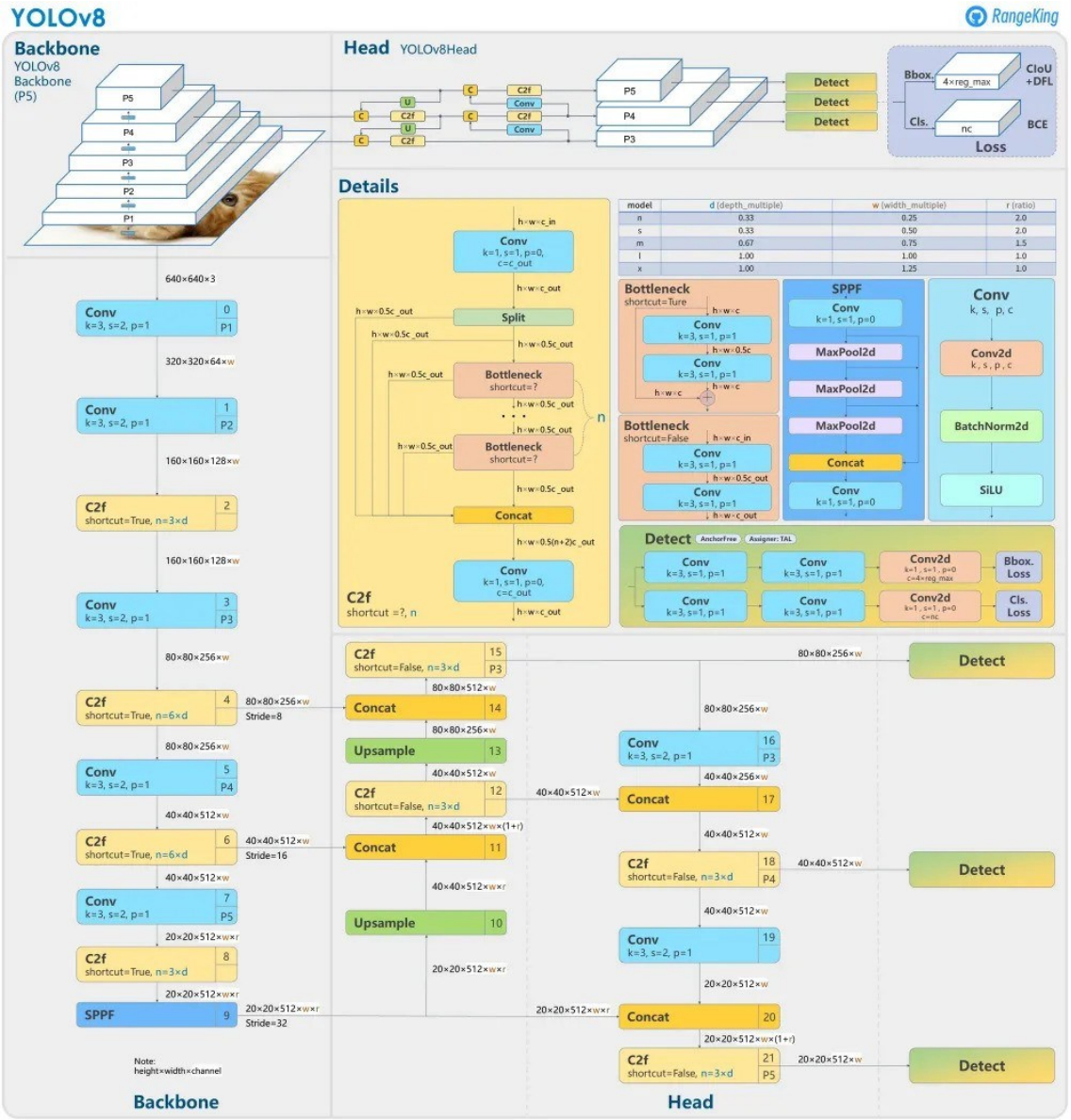


Slika 2: Usporedba YOLOv8 s prethodnim YOLO modelima

4.2. YOLOv8 algoritam: arhitektura i radni principi

Važno je naglasiti da su temelji YOLO arhitekture uspostavljeni inicijalno u radu Josepha Redmona i suradnika u [27], a službeni papir za YOLOv8 u vrijeme pisanja ovog završnog rada još uvijek nije objavljen. Arhitektura YOLO algoritma dizajnirana je za učinkovitost i efektivnost. Sastoji se od konvolucijske neuronske mreže koja dijeli ulaznu sliku na mrežu i predviđa granice okvira i vjerojatnosti za svaku ćeliju mreže. Ovaj pojednostavljeni pristup omogućuje YOLO-u da istovremeno detektira više objekata s visokom točnošću. Postoje tri ključna bloka u algoritmu, a sve će se odvijati unutar tih blokova, a to su: **kralježnica** (**engl. backbone**), **vrat (engl. neck)** i **glava (engl. head)** [12], [13].

Izvor 3: Jacob Solawetz, Francesco, What is YOLOv8? The Ultimate Guide. [13]



Slika 3: Arhitektura YOLO algoritma

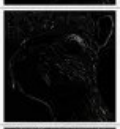
U YOLO arhitekturi kralježnica (engl. backbone), također poznata kao ekstraktor značajki, odgovorna je za ekstrakciju značajnih značajki iz ulaza. Kralježnica hvata jednostavne uzorke u početnim slojevima, poput rubova i tekstura, te može imati više razina reprezentacije kako ide dublje, hvatajući značajke s različitih razina apstrakcije, pružajući bogatu, hijerarhijsku reprezentaciju ulaza. Vrat (engl. neck) djeluje kao most između kralježnice i glave (engl. head), obavljajući operacije fuzije značajki i integrirajući kontekstualne informacije. Vrat sastavlja piramide značajki agregiranjem mapa značajki dobivenih od kralježnice, provodeći spajanje ili fuziju značajki različitih skala kako bi se osiguralo da mreža može otkriti objekte različitih veličina. Vrat također

integrira kontekstualne informacije kako bi poboljšao točnost detekcije uzimajući u obzir širi kontekst scene te smanjuje prostornu rezoluciju i dimenzionalnost resursa radi olakšavanja računalne obrade, što povećava brzinu, ali može smanjiti kvalitetu modela. Glava je završni dio mreže i odgovoran je za generiranje izlaza mreže, kao što su okviri ograničenja (engl. bounding box) i pouzdanosti za detekciju objekata. Glava generira okvire ograničenja povezane s mogućim objektima na slici, dodjeljuje pouzdanost svakom okviru ograničenja kako bi se naznačilo koliko je vjerojatno da je objekt prisutan te sortira objekte u okvirima ograničenja prema njihovim kategorijama [12].

YOLO arhitektura ima pristup analize lokalnih značajki umjesto ispitivanja slike u cjelini, a cilj ove strategije je prvenstveno smanjiti računalni napor i omogućiti detekciju u stvarnom vremenu. Za ekstrakciju mapa značajki, konvolucije se koriste više puta u algoritmu. Konvolucija je matematička operacija koja kombinira dvije funkcije kako bi stvorila treću. U računalnom vidu i obradi signala, konvolucija se često koristi za primjenu filtera na slike ili signale, naglašavajući specifične uzorke. U konvolucijskim neuronskim mrežama, konvolucija se koristi za ekstrakciju značajki iz ulaza poput slika. Konvolucije su strukturirane s pomoću jezgri (engl. kernels, k), pomaka (engl. strides, s) i popunjavanja (engl. padding, p) [12].

Jezgra, također poznat kao filter, je mala matrica brojeva koja se pomiče preko ulaza (slike ili signala) tijekom operacije konvolucije. Cilj je primijeniti lokalne operacije na ulaz kako bi se otkrile specifične karakteristike. Svaki element u jezgri predstavlja težinu koja se množi s odgovarajućom vrijednošću u ulazu tijekom konvolucije [12].

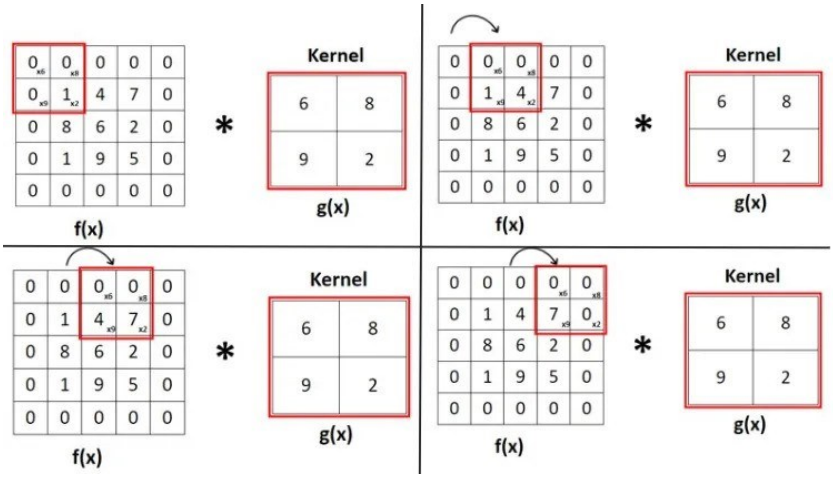
Izvor 4: Juan Pedro, Detailed Explanation of YOLOv8 Architecture. [12]

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Slika 4: Prikaz jezgre

Pomak je količina premještanja koju jezgra prolazi dok se kreće preko ulaza tijekom konvolucije. Pomak od 1 znači da se jezgra pomiče za jednu poziciju odjednom, dok pomak od 2 znači da jezgra preskače dvije pozicije sa svakim pomakom. Pomak izravno utječe na prostorne dimenzije izlaza konvolucije [12].

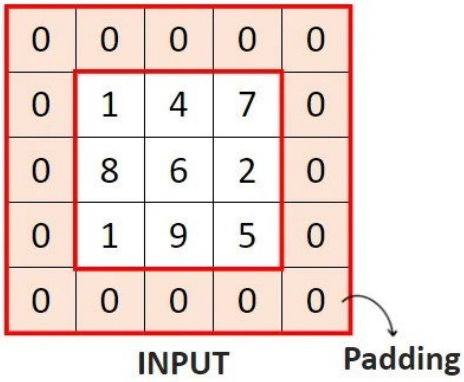
Izvor 5: Juan Pedro, Detailed Explanation of YOLOv8 Architecture. [12]



Slika 5: Prikaz pomaka

Popunjavanje se odnosi na dodavanje dodatnih piksela oko rubova ulazne slike (obično nula) prije primjene konvolucijskih operacija. Ovo se radi kako bi se osiguralo da se informacije na rubovima slike tretiraju na isti način kao i informacije u središtu tijekom konvolucijskih operacija [12].

Izvor 6: Juan Pedro, Detailed Explanation of YOLOv8 Architecture. [12]



Slika 6: Prikaz popunjavanja

Način treniranja (engl. train mode) koristi se za treniranje YOLOv8 modela na prilagođenom skupu podataka. U ovom načinu rada, model se trenira korištenjem specificiranog skupa podataka i hiperparametara. Proces treniranja uključuje optimizaciju hiperparametara modela kako bi mogao točno predvidjeti klase i lokacije objekata na slici. **COCO (Common Objects in Context)** je veliki skup podataka za detekciju objekata, segmentaciju i opisivanje. Dizajniran je kako bi potaknuo istraživanje na širokom spektru kategorija objekata i obično se koristi za ocjenjivanje

modela računalnog vida. COCO skup podataka se široko koristi za treniranje i evaluaciju modela dubokog učenja u detekciji objekata, segmentaciji instanci i detekciji ključnih točaka. Raznolik skup kategorija objekata, veliki broj anotiranih slika i standardizirani metrički sustavi evaluacije čine ga korisnim resursom u području računalnog vida. Moguće je koristiti unaprijed trenirani model (na primjer yolov8s.pt koji je unaprijed treniran na COCO skupu podataka) ili model od "nule". Za potrebe rada sam koristio oba pristupa te će u nastavku rada biti prikazana njihova primjena i rezultati treniranja različitih modela [2], [4], [5].

Način validacije (engl. validation mode) koristi se za validaciju YOLOv8 modela nakon što je treniran. U ovom načinu rada, model se evaluira na skupu za validaciju kako bi se izmjerila njegova točnost i sposobnost generalizacije. Ovaj način rada može se koristiti za fino podešavanje hiperparametara modela kako bi se poboljšala njegova izvedba [4].

Način predviđanja (engl. predict mode) koristi se za predikcije koristeći trenirani YOLOv8 model na novim slikama ili videozapisima. U ovom načinu rada, model se učitava iz datoteke kontrolne točke, a korisnik može pružiti slike ili videozapise za provođenje inferencije. Model predviđa klase i lokacije objekata na ulaznim slikama ili videozapisima [4].

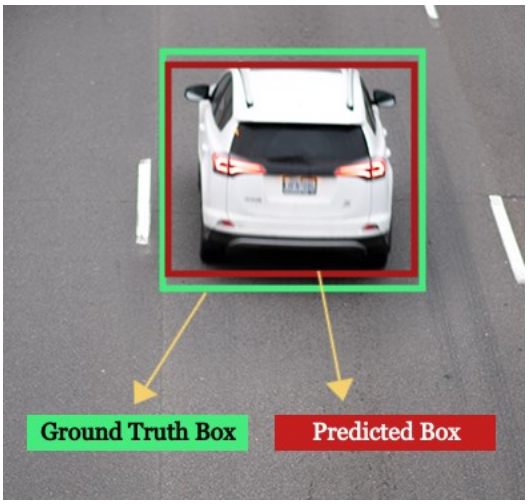
4.3. Evaluacijske metrike

Mjerni pokazatelji performansi ključni su alati za ocjenu točnosti i učinkovitosti modela za detekciju objekata. Oni pokazuju koliko učinkovito model može identificirati i lokalizirati objekte unutar slika. Osim toga, pomažu u razumijevanju kako model rukuje lažno pozitivnim i lažno negativnim rezultatima. Ovi uvidi su ključni za ocjenjivanje i poboljšanje performansi modela [15].

Metrike koje nisu važne samo za YOLOv8 već su općenito primjenjive na različite modele detekcije objekata su presjek nad unijom (engl. intersection over union, IoU), prosječna preciznost (engl. average precision, AP), srednja prosječna preciznost (engl. mean average precision, mAP), F1-vrijednost (engl. F1-score), preciznost (engl. precision) i odziv (engl. recall).

Presjek nad unijom (engl. intersection over union, IoU) je mjera koja kvantificira preklapanje između predviđenog okvira (engl. predicted box) ograničenja i stvarnog okvira (engl. ground truth box) ograničenja. Igra temeljnu ulogu u procjeni točnosti lokalizacije objekata [15].

Izvor 7: Deval Shah ,Mean Average Precision (mAP) Explained: Everything You Need to Know. [22]



Slika 7: Stvarni okvir i predviđenog okvira

Izvor 8: Gaudenz Boesch, What is Intersection over Union (IoU)? [21]

$$Intersection\ over\ Union = \frac{Area\ of\ Intersection}{Area\ of\ Union}$$

Formula 1: Matematička formula za presjek nad unijom

Površina presjeka (engl. area of intersection) je zajednička površina koju dijele dva granična okvira (preklapanje). Površina unije (engl. area of union) je ukupna površina pokrivena s dva granična okvira. Formula 1 proizvodi vrijednost između 0 i 1, gdje 0 označava da nema preklapanja, a 1 označava savršeno podudaranje između predviđenog okvira i stvarnih graničnih okvira [21], [22].

Prosječna preciznost (engl. average precision, AP) računa površinu ispod krivulje preciznost-odziv, pružajući jednu vrijednost koja obuhvaća preciznost i odziv modela [14], [15].

Izvor 9: Edmundo Casas, Leo Ramos, Eduardo Bendek and Francklin Rivas-Echeverría, Assessing the Effectiveness of YOLO Architectures for Smoke and Wildfire Detection. [15]

$$AP = \int_0^1 precision(r)\, d\, r$$

Formula 2: Matematička formula za prosječnu preciznost

Izraženo matematički, definira se kao što je prikazano u formuli 2, gdje *preciznost (r)* predstavlja preciznost na određenoj razini odziva (*r*) [15].

Srednja prosječna preciznost (engl. mean average precision, mAP) proširuje koncept *AP*-a računajući prosječne vrijednosti *AP*-a preko više objektnih klasa. To je korisno u scenarijima detekcije višeklasnih objekata kako bi se pružila sveobuhvatna evaluacija performansi modela [14], [15].

Izvor 10: Deval Shah ,Mean Average Precision (mAP) Explained: Everything You Need to Know. [22]

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i$$

Formula 3: Matematička formula za srednju prosječnu preciznost

U formuli 3 je vidljiva matematička formula gdje *N* označava ukupan broj klasa, AP_i označava prosječnu preciznost za *i*-tu klasu. Računanje *AP*-a je prethodno opisan te prikazan u formuli 2 [22].

Preciznost (engl. precision) i odziv (engl. recall) kvantificira udio pravih pozitiva među svim pozitivnim predikcijama, ocjenjujući sposobnost modela da izbjegne lažne pozitivne rezultate. S druge strane, odziv izračunava udio pravih pozitiva među svim stvarnim pozitivima, mjereći sposobnost modela da otkrije sve instance određene klase [14], [15].

Izvor 11: Edmundo Casas, Leo Ramos, Eduardo Bendek and Francklin Rivas-Echeverría, Assessing the Effectiveness of YOLO Architectures for Smoke and Wildfire Detection. [15]

$$Precision = \frac{TP}{TP + FP}$$

Formula 4: Matematička formula za preciznost

Izvor 12: Edmundo Casas, Leo Ramos, Eduardo Bendek and Francklin Rivas-Echeverría, Assessing the Effectiveness of YOLO Architectures for Smoke and Wildfire Detection. [15]

$$Recall = \frac{TP}{TP + FN}$$

Formula 5: Matematička formula za odziv

U matematičkim terminima, definicija preciznosti može se predstaviti formulom prikazanom u formuli 4, gdje *TP* predstavlja broj ispravno predviđenih pozitivnih slučajeva. Lažno pozitivni *FP* predstavlja broj slučajeva koji su pogrešno predviđeni kao pozitivni. Za formulu odziva *TP* također predstavlja broj ispravno predviđenih pozitivnih slučajeva, a *FN* predstavlja broj slučajeva koji su pogrešno predviđeni kao negativni [15].

F1-vrijednost (engl. F1-score) je harmonična sredina preciznosti i odziva, pružajući uravnoteženu procjenu performansi modela uzimajući u obzir i lažne pozitivne te lažne negativne rezultate [14], [15].

Izvor 13: Edmundo Casas, Leo Ramos, Eduardo Bendek and Francklin Rivas-Echeverría, Assessing the Effectiveness of YOLO Architectures for Smoke and Wildfire Detection. [15]

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision \times Recall}$$

Formula 6: Matematička formula za F1-vrijednost

Matematički gledano, njegova definicija izražena je jednadžbom prikazanom u formuli 6, gdje je preciznost (engl. precision) omjer istinskih pozitivnih (*TP*) i zbroja istinskih pozitivnih (*TP*) i lažno pozitivnih (*FP*), dok je odziv (engl. recall) omjer istinskih pozitivnih (*TP*) i zbroja istinskih pozitivnih (*TP*) i lažno negativnih (*FN*) [15].

Jedan od dijelova izlaza je detaljna analiza performansi po klasama. Ova detaljna informacija korisna je kada se pokušava razumjeti kako se model ponaša za svaku pojedinačnu klasu, posebno u skupovima podataka s raznolikim kategorijama objekata [14] , [15]. Za svaku klasu u skupu podataka pružaju se informacije o klasi, slici, instanci te okviru. **Klasa (engl. class)** označava naziv objektne klase, poput "osoba", "automobil" ili "pas". U ovom završnom radu će se koristiti klase “occupied” (zauzeto) i “empty” (slobodno) koje označavaju je li određeno parkirno mjesto slobodno ili zauzeto. **Slike (engl. image)** je mjera koja govori o broju slika u skupu koje sadrže objektnu klasu. **Instance (engl. instance)** pružaju broj pojavljivanja klase u svim slikama u skupu. **Okvir (engl. box, a sastoji se od: P, R, mAP50, mAP50-**

95) je mjera koja pruža uvide u performanse modela u detekciji objekata. **P** (preciznost) označava točnost detektiranih objekata, što ukazuje na to koliko je detekcija bila ispravna. **R** (odziv) je mogućnost modela da identificira sve instance objekata na slikama. **mAP50** je srednja prosječna preciznosti koja je izračunata na pragu preklapanja (IoU) od 0,50. To znači da se prosječna preciznost izračunava uzimajući u obzir samo one detekcije gdje se predviđeni okvir i stvarni okvir preklapaju s najmanje 50% (IoU ≥ 0,50). Prethodno je opisan IoU te njegovo računanje. Ovo je mjera točnosti modela koja uzima u obzir samo "jednostavne" detekcije. **mAP50-95** je prosječna srednja preciznost izračunata na različitim pragovima preklapanja, koji variraju od 0,50 do 0,95. Ovdje se prosječna preciznost izračunava na različitim pragovima preklapanja koji variraju od 0,50 do 0,95 u koracima od 0,05 (0,50, 0,55, 0,60, ..., 0,95). To znači da se procjenjuju performanse modela na različitim razinama preciznosti, od relativno "jednostavnih" detekcija (0,50) do vrlo "strogih" detekcija (0,95). Pruža sveobuhvatan pregled performansi modela na različitim razinama teškoća detekcije [14], [15].

Osim generiranja numeričkih metrika moguće je također proizvoditi vizualne izlaze koji pružaju intuitivnije razumijevanje performansi modela. U nastavku ovog odlomka se nalazi pregled vizualnih izlaza koji se mogu očekivati te nazivi datoteka u kojima se pojedini nalazi nakon generiranja. **Krivulja F1 vrijednosti (engl. F1 score curve, F1_curve.png) je** krivulja koja predstavlja F1 vrijednost pri različitim pragovima. Interpretacija ove krivulje može pružiti uvide u ravnotežu modela između lažno pozitivnih i lažno negativnih rezultata pri različitim pragovima. **Krivulja preciznost-odziv (engl. precision-recall curve, PR_curve.png)** je bitna vizualizacija za svaki problem klasifikacije, ova krivulja prikazuje kompromise između preciznosti i odziva pri različitim pragovima. Postaje posebno značajna kada je riječ o neizbalansiranim klasama. **Krivulja preciznosti (engl. precision curve, P_curve.png)** je grafički prikaz vrijednosti preciznosti pri različitim pragovima. Ova krivulja pomaže u razumijevanju kako se preciznost mijenja s promjenom pragova. **Krivulja odziva (engl. recall curve, R_curve.png)** je graf koji ilustrira kako se vrijednosti odziva mijenjaju pri različitim pragovima. **Matrica zabune (engl. confusion matrix, confusion_matrix.png)** pruža detaljan prikaz rezultata, prikazujući brojeve pravih pozitiva, pravih negativa, lažnih pozitiva i lažnih negativa za svaku klasu. **Normalizirana matrica zabune (engl. normalized confusion matrix,**

confusion_matrix_normalized.png) prikazuje podatke u proporcijama umjesto u sirovim brojkama. Ovaj format olakšava usporedbu performansi između klasa. **Oznake validacijskih serija (engl. validation batch labels, val_batchX_labels.jpg)** su slike koje prikazuju stvarne oznake za različite serije (engl. batch) iz validacijskog skupa podataka. Pružaju jasnu sliku o tome što su objekti i njihovim lokacijama prema skupu podataka. **Predikcije validacijskih serija (engl. validation batch predictions, val_batchX_pred.jpg)** su vizualni elementi koji prikazuju predikcije koje je model YOLOv8 napravio za odgovarajuće serije. Usporedbom s oznakama može se lako procijeniti koliko dobro model detektira i klasificira objekte vizualno [14].

Gubitak (engl. loss) se odnosi na kvantitativnu mjeru neusklađenosti između predviđenih izlaza modela i stvarnih ili očekivanih vrijednosti. U kontekstu detekcije objekata, gubitak se koristi za procjenu točnosti detekcija koje model napravi u usporedbi sa stvarnim lokacijama objekata i oznakama na slici. Niža vrijednost gubitka ukazuje na veće slaganje između predviđanja i stvarnih vrijednosti, a cilj je smanjiti gubitak tijekom procesa treniranja kako bi se poboljšala izvedba modela. Različite verzije YOLO algoritma razlikuju se po funkcijama gubitka koje koriste. YOLOv5 koristi gubitak okvira (engl. box loss), gubitak objektivnosti (engl. objectness (obj) loss) i gubitak klasifikacije (engl. classification (cls) loss). YOLOv6 koristi gubitak presjeka nad unijom (engl. intersection over union (IoU) loss), gubitak klasifikacije i distribucijski fokalni gubitak (engl. distributional focal loss (dfl)). YOLOv7 koristi gubitak okvira, gubitak objektivnosti i gubitak klasifikacije. YOLOv8 koristi gubitak okvira, gubitak klasifikacije i distribucijski fokalni gubitak. U nastavku će biti opisani različiti tipovi gubitka jer su prisutni kod različitih verzija YOLO algoritma, ali naglašavam važnost gubitka okvira, gubitka klasifikacije i distribucijskog fokalnog gubitka jer su to tipovi gubitka koji u ovom radu implementirani YOLOv8 model koristi [15].

Gubitak okvira (engl. box loss) je funkcija gubitka koja mjeri neslaganje između predviđenih koordinata graničnog okvira i stvarnih koordinata okvira. Obično koristi metrike poput srednje kvadratne pogreške (MSE) ili glatkog L1 gubitka. **Gubitak objektivnosti (engl. objectness (obj) loss)** procjenjuje točnost predviđanja objektivnosti, koja određuju sadrži li zadani granični okvir objekt ili ne. Uobičajeno koristi binarnu unakrsnu entropiju za gubitak (engl. binary cross-entropy loss). **Gubitak klasifikacije (engl. classification (cls) loss)** mjeri razliku između

predviđenih vjerojatnosti klasa i stvarnih oznaka klasa. Koristi kategorijsku unakrsnu entropiju za gubitak (engl. categorical cross-entropy loss). **Gubitak presjeka nad unijom (engl. intersection over union (IoU) loss)** procjenjuje konzistentnost između predviđenih graničnih okvira i stvarnih okvira koristeći IoU metriku. Kvantificira preklapanje između predviđenih i stvarnih graničnih okvira. **Distribucijski fokalni gubitak (engl. distributional focal loss (dfl))** je proširenje fokalnog gubitka (engl. focal loss) i koristi se za rješavanje neravnoteže klasa u detekciji objekata. Dodjeljuje veće težine izazovnijim primjerima koje je teže točno klasificirati [15].

Instanca se obično odnosi na pojedinačne primjerke ili instance objekata koji se detektiraju na slikama. Na primjer, ako model detektira tri osobe i dva automobila na jednoj slici, to znači da postoji pet instanci objekata na toj slici (tri instance osobe i dvije instance automobila). Slojevi su osnovni građevni blokovi neuronskih mreža te slojevi u neuronskoj mreži obavljaju različite zadatke obrade podataka [15], [19].

Performanse računala i superračunala izražavaju se korištenjem standardne stope koja označava broj aritmetičkih izračuna s pomičnim zarezom koje sustavi mogu izvršiti u sekundi. Ta stopa, operacije s pomičnim zarezom po sekundi, skraćeno je FLOPS. Računalni sustav s 1 gigaFLOPS (GFLOPS) sposoban je izvršiti milijardu operacija s pomičnim zarezom u sekundi. Da bi se postiglo ono što računalni sustav s 1 GFLOPS može učiniti u samo jednoj sekundi, morali bismo izvoditi jedan izračun svake sekunde tijekom 31,69 godina [17].

5. METODOLOGIJA ZAVRŠNOG RADA

5.1. Skup podataka

Izvor skupa podataka (engl. dataset) je *Muhammad Syihab, Parking Space Image Dataset* [10]. Skup podataka se sastoji od tri direktorija: train, valid i test te od datoteka data.yaml, README.dataset i README.roboflow. README.dataset naglašava da skup podataka ima licencu “CC BY 4.0 DEED” što omogućava da se slobodno može:

- dijeliti - kopirati i redistribuirati materijal u bilo kojem mediju ili formatu u bilo koju svrhu, čak i komercijalno [6].
- prilagoditi - remiksati, transformirati i nadograđivati materijal u bilo koju svrhu, čak i komercijalno [6].

Mora se dati odgovarajuće priznanje, pružiti poveznica na licencu i naznačiti jesu li napravljene izmjene [6]. Naglašavam da je u sklopu izrade ovog završnog rada jedino data.yaml izmijenjen, a detalje izmjene ću naglasiti u nastavku rada. U README.roboflow se može vidjeti da skup podataka uključuje 7801 slika te da su vozila označena u YOLOv8 formatu.

U skupu podataka *Muhammad Syihab, Parking Space Image Dataset* [10] autor je primijenio sljedeće predobrade za slike te njihove oznake: automatska orijentacija pikselskih podataka (s uklanjanjem EXIF orijentacije), promjena veličine slike (engl. image size, skraćeno je imgsz) na 640x640 (proporcionalno rastezanje) i za stvaranje tri verzije svake izvorne slike primijenjena je dodatna augmentacija. Korišteno je: nasumično izrezivanje između 0 i 20 posto slike, nasumična rotacija između -20 i +20 stupnjeva, nasumična prilagodba svjetline između -25 i +25 posto, nasumična prilagodba ekspozicije između -10 i +10 posto, nasumično je primijenjeno Gaussovo zamućenje (engl. Gaussian blur) između 0 i 1 piksela te sol i papar šum primijenjen je na 2 posto piksela.

Svaki direktoriji *train*, *valid* i *test* ima svoja dva direktorija zvana *labels* i *images*. Subdirektorij images ima slike u formatu jpg koje sadrže parkirna mjesta i vozila u slučaju da ih ima.

Izvor 14: Muhammad Syihab, Parking Space Image Dataset [10]

Slika 8: Slika s parkirnim mjestima i vozilima

Izvor 15: Muhammad Syihab, Parking Space Image Dataset [10]

Slika 9: Slika s klasnim oznakama

Subdirektorij labels sadrži tekstualne datoteke u kojima svaki red ima numeričke vrijednosti koje označavaju danim redoslijedom: <klasa>, <x_centroid>, <y_centroid>, <širina> i <visina>. <klasa> je indeks klase objekta (u ovom slučaju, "0" za slobodno parkirno mjesto i "1" za zauzeto parkirno mjesto). <x_centroid> i <y_centroid> su koordinate središta okvira za obuhvat (engl. bounding box) normalizirane u rasponu od 0 do 1, u odnosu na širinu i visinu slike. <širina> i <visina> su normalizirane dimenzije okvira za obuhvat u odnosu na širinu i visinu slike. Iz dolje priložene slike za prvi red (1 0.3 0.303125 0.03203125 0.1) se može zaključiti da: 1 predstavlja klasu (zauzeto parkirno mjesto), 0.3 predstavlja normaliziranu x-koordinatu središta okvira (30% širine slike), 0.303125 je normalizirana y-koordinata središta okvira (30.3125% visine slike), 0.03203125 označava normaliziranu širinu okvira (3.203125% širine slike), a 0.1 je normalizirana visina okvira (10% visine slike). Isti princip očitavanja vrijednosti vrijedi za ostale redove te ostale datoteke [16].

```
1 0.3 0.303125 0.03203125 0.1
1 0.3359375 0.34453125 0.05546875 0.08828125
1 0.4953125 0.36328125 0.2546875 0.2796875
1 0.7515625 0.44296875 0.49609375 0.70078125
0 0.49921875 0.48125 0.17578125 0.225
```

Slika 10: Tekstualni prikaz oznake slike

Važno je naglasiti da je autor skupa podataka odredio omjer i kvantitativnu distribuciju unutar skupa podataka *Muhammad Syihab, Parking Space Image Dataset* [10]. Direktorij train sadrži 7017 slika, a samim time se može zaključiti da ima 7017 datoteka s oznakama za slike. Validacijski set sadrži 558 slika tj 558 datoteka s oznakama za slike, a set za testiranje sadrži 226 slika i 226 datoteka s oznakama. Omjer između

broja slika tj oznaka je zadovoljavajući, a sveukupni broj slika je dovoljan za treniranje modela za potrebe ovog rada.

Izvor 16: Muhammad Syihab, Parking Space Image Dataset

Slika 11: Omjer i kvantitativna distribucija unutar skupa podataka

Datoteka data.yaml konfiguracijska je datoteka koja se koristi za YOLO model prilikom treniranja, validacije i testiranja. Datoteka definira putanje do skupova podataka i druge važne informacije potrebne za trening modela.

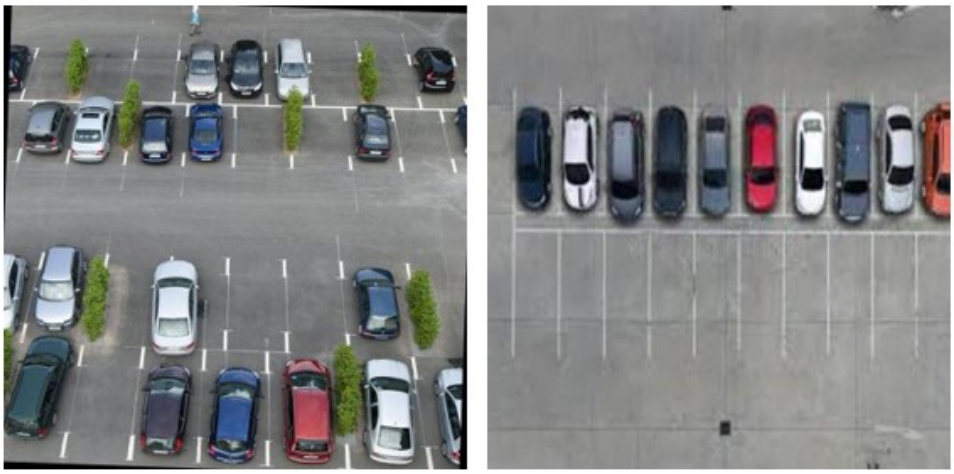
Datoteka data.yaml sadrži *train: ../train/images*, a to je putanja do direktorija koji sadrži slike za trening. Putanja do direktorija koji sadrži slike za validaciju je *val: ../valid/images*, a putanja do direktorija koji sadrži slike za testiranje je *test: ../test/images*. Sadrži *nc: 2* koji označava broj klasa koje model treba prepoznati (u ovom slučaju, dvije klase: "empty" i "occupied"). Linija *names: ['empty', 'occupied']* predstavlja nazive klasa koje model treba prepoznati. Klasa "empty" predstavlja slobodno parkirno mjesto, a klasa "occupied" zauzeto parkirno mjesto. Linija *roboflow:* sadrži informacije o projektu i verziji skupa podataka iz Roboflow-a [10], [16].

```yaml
train: ../train/images
val: ../valid/images
test: ../test/images

nc: 2
names: ['empty', 'occupied']

roboflow:
  workspace: muhammad-syihab-bdynf
  project: parking-space-ipm1b
  version: 4
  license: CC BY 4.0
  url: https://universe.roboflow.com/muhammad-syihab-bdynf/parking-space-ipm1b/dataset/4
```

Slika 12: Datoteka data.yaml

Važno je naglasiti da u skupu podataka ima kvalitetnih te manje kvalitetnih slika. Naglašavam da nije riječ o kvaliteti specifikacije kamere i slike već o samoj kvaliteti pozicioniranja kamera za izradu potrebne slike. Postoji dobar ugao i lošiji ugao te drugi aspekti pozicioniranja kamere. U određenim slikama model neće pronaći sva parkirna

mjesta ne jer je model slab već zato što je riječ o lošijem pozicioniranju slike što čini izazovnijim prepoznavanje parkirnih mjesta.



Slika 13: Primjer dobro pozicioniranih kamera

Slika 14: Primjer loše pozicioniranih kamera

Možemo primijetiti da je u prvom redu prethodnog prilaganja u slici 13 jasno vidljiv parking te njegova mjesta, a jedini je nedostatak da ne obuhvaća sva mjesta te se može pretpostaviti da ima još kamera. Lagano se može raspoznati parkirno mjesto, njihov broj te je li mjesto zauzeto ili slobodno. U slici 14 se može vidjeti da uočavanje pozadinskih parkirnih mjesta te vozila može biti izazovno kada je prednje parkirno mjesto bilo zauzeto.

5.2. Priprema okruženja za implementaciju YOLO algoritma

Implementacija YOLO algoritma zahtijeva pripremu okruženja kako bi se osiguralo učinkovito treniranje, validacija i testiranje modela. U ovom slučaju, lokalno okruženje VS code-a je korišteno za izvođenje svih faza razvoja modela, čime je osigurana potpuna kontrola nad procesom i mogućnost optimizacije prema specifičnim zahtjevima projekta. Treniranje modela izvedeno je na laptopu s 16GB RAM-a, koristeći Ultralytics YOLOv8.2.22, Python 3.10.11, s CUDA 11.8 podrškom, uz NVIDIA GeForce RTX 3050 Ti Laptop GPU s 4096MiB memorije.

Ovaj hardverski sklop pruža solidnu osnovu za izvođenje zadataka dubokog učenja. NVIDIA GeForce RTX 3050 Ti Laptop GPU, iako ne spada u vrhunske grafičke kartice, pruža dovoljno računalne snage za treniranje kompleksnih modela poput YOLOv8. Njegovih 4GB memorije omogućava rad s modelima srednje veličine i skupovima podataka koji nisu previše veliki, no može naići na ograničenja kod izuzetno velikih skupova podataka ili vrlo kompleksnih modela. Unatoč tome, za većinu standardnih zadataka u detekciji objekata, ovaj GPU pruža zadovoljavajuće performanse, omogućujući učinkovito izvođenje iteracija treniranja uz razumnu brzinu.

Python je odabran kao programski jezik zbog svoje trenutne dominacije u području strojnog učenja i dostupnosti opsežnog ekosustava biblioteka i alata. Sve ove komponente zajedno čine kohezivno i učinkovito okruženje za razvoj i implementaciju YOLO algoritma. U glavnom direktoriju projekta nalazi se direktorij *datasets* i skripta *Parking_Slot_Detection.ipynb* koja sadrži potrebni python kod za razvoj potrebnih modela. Za ovaj rad je korišten Jupyter notebook zato jer omogućuje interaktivnost, fleksibilnost i bolju dokumentaciju. U *datasets* se nalazi direktorij *Dataset* koji sadrži direktorije *train*, *valid* i *test* te *data.yaml*. Direktoriji *train*, *valid* i *test* svaki individualno sadrže svoje *images* te *labels* direktorije koji sadrže slike i oznake za objekte unutar slike. Prilikom pokretanja različitih faza razvoja modela se kreiraju određene datoteke za koje će se detaljnije diskutirati u nastavku rada.

```
train: C:/ParkingSlotDetection/datasets/Dataset/train/images
val: C:/ParkingSlotDetection/datasets/Dataset/valid/images
test: C:/ParkingSlotDetection/datasets/Dataset/test/images

nc: 2
names: ['empty', 'occupied']
```

Slika 15: Modificirani data.yaml

U data.yaml su napravljene manje promijene, a one su prisutne u putanjama (engl. path) za skupove podataka za treniranje, validaciju i testiranje. Modificirane su sukladno potrebama lokalnog okruženja za implementaciju algoritma.

5.3. Implementacija YOLO algoritma za prepoznavanje parkirnih mjesta

Za potrebu ovog rada odlučio sam se za YOLO algoritam verzija 8 tj YOLOv8 te je važno naglasiti da postoji više modela YOLOv8 algoritma, a to su YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l i YOLOv8x. YOLOv8n je najmanji po veličini modela, manje resursa i memorije troši, brži je, ali se smanjuje točnost u usporedbi s drugim modelima. YOLOv8x je potpuna suprotnost: najveći po veličini modela, više resursa i memorije troši, sporiji je, ali se povećava točnost u usporedbi s drugim modelima. Za modele koje sam naveo između YOLOv8n i YOLOv8x se navedene specifikacije povećavaju/smanjuju ovisno o njihovoj blizini između dva modela.

U ovom radu sam upotrijebio YOLOv8n, YOLOv8s i YOLOv8m jer su prisutna ograničenja po pitanju resursa te memorije prilikom treniranja modela. Za potrebe ovog rada ovi modeli će biti dovoljno dobri za prihvatljivu kvalitetu demonstracije prepoznavanja parkirnih mjesta. Napravio sam sedam treninga s različitim modelima i hiperparametrima te kod nekih sam koristio unaprijed treniran model, a kod nekih sam izvodio trening bez unaprijed treniranog modela. Htio sam pokazati razliku u kvaliteti između različitih modela s različitim hiperparametrima. U nastavku rada će se uspoređivati njihova kvaliteta prilikom validacije i testiranja. U tablici 1 su prikazani korišteni modeli i hiperparametri za izvedena treniranja.

Model YOLO-a / Hiperparametri treninga	**Korištenje unaprijed treniranog modela**	**Broj epoha**	**Veličina slike**	**Serija (engl. batch)**	**Stopa učenja (engl. learning rate, lr0)**
YOLOv8n	Ne	100	640	16	0.01
YOLOv8s	Ne	100	640	16	0.01
YOLOv8m	Ne	100	320	16	0.01
YOLOv8n	Da	100	640	16	0.01
YOLOv8s	Da	100	640	16	0.01
YOLOv8m	Da	100	320	16	0.01
YOLOv8m	Da	100	480	16	0.01

Tablica 1: Modeli i hiperparametri za pojedini trening

U *Parking_Slot_Detection.ipynb* je implementirano kreiranje modela, proces treniranja, validacije i testiranja za svaki individualni model. S *nazivVarijable = YOLO(“”)* se u toj varijabli sprema unaprijed trenirani model (na primjer *YOLO(“yolov8n.pt”)* koji je unaprijed treniran na COCO dataset ili na primjer yolov8n.yaml. Funkcija **train** služi za treniranje modela, a mora se naglasiti putanja za skup podataka s kojim će se trening izvoditi te je moguće navesti hiperparametre sa željenim vrijednostima. U *data.yaml* je naglašeno koje slike i oznake se koriste za treniranje, validaciju i testiranje. Za validaciju sam uzimao model koji je imao najbolje rezultate tijekom treniranja. Pokušao sam i s drugim modelima, ali *best.pt* iz svakog treninga je uvijek pokazao najbolji rezultat. Model *best.pt* je model koji je imao najbolje težine tijekom cijelog treniranja te se nalazi u putanji *runs/detect/trainBroj/weights/best.pt*. Funkcija **val** je izvodila proces validacije te su izlazne mjere validacije spremljene u varijablu *metrics*. Funkcija **predict** je s navedenim modelom(korišten je *best.pt*) izvodila predviđanje klasa te njihovih lokacija na njima neviđenoj slici na temelju sposobnosti detekcije modela. Izlaz funkcije je prikaz slike s detektiranim objektima i njihovim klasama. Ja sam napravio da se za model sve neviđene slike njezinog izlaza funkcije spreme u putanji *output/predictions* prilikom čega se kreira novi direktorij *predictBroj*.

Nakon završetka funkcije train se u glavnom direktoriju ParkingSlotDetection kreira direktorij *runs* a unutar njega se kreira direktorij *detect*. Unutar putanje *ParkingSlotDetection/runs/detect* se kreira direktorij za svaki trening (*trainBroj*) i validaciju (*valBroj*). Za svako testiranje/predviđanje se kreira direktoriji *predictBroj* koji

se sprema u putanju *runs/ detect*, a taj direktorij sadrži izlazne slike s oznakama za klase te se može zaključiti koliko je model kvalitetan.

5.4. Evaluacija rezultata

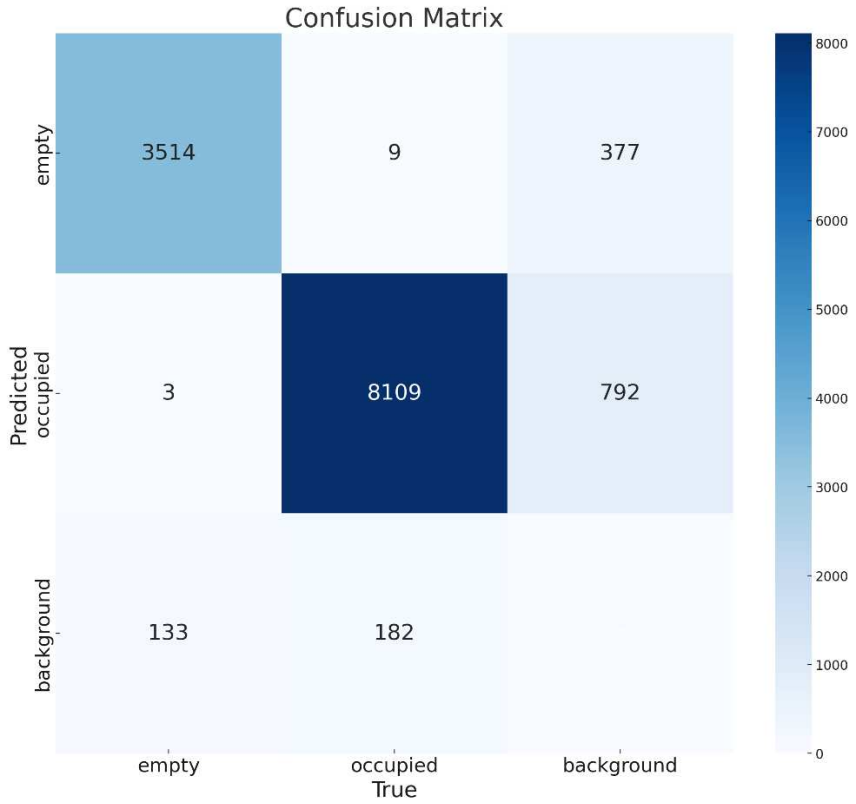
U tablici 2 su priloženi rezultati validacije različitih modela s različitim hiperparametrima od kojih su neki koristili unaprijed treniran model, a neki nisu koristili unaprijed treniran model koji je treniran na COCO skupu podataka. Svi modeli su prilikom validacije imali jednak broj instanci, neovisno o klasi.

Model YOLO-a / Specifikacije i metrike	**Korištenje unaprijed treniranog modela**	**Veličina slike**	**Instance (sve klase)**	**Instance (klasa “empty”)**	**Instance (klasa “occupied”)**	**Slojevi**	**Parametri (u mil.)**	**GFLOPs**	**Preciznost (sve klase)**	**Preciznost (klasa “empty”)**	**Preciznost (klasa “occupied”)**	**Odziv (sve klase)**	**Odziv (klasa “empty”)**	**Odziv (klasa “occupied”)**	**mAP50 (sve klase)**	**mAP50 (klasa “empty”)**	**mAP50 (klasa “occupied”)**	**mAP50-95 (sve klase)**	**mAP50-95 (klasa “empty”)**	**mAP50-95 (klasa “occupied”)**
YOLOv8n	Ne	640	11950	3650	8300	168	3.01	8.1	0.958	0.94	0.976	0.939	0.914	0.963	0.974	0.964	0.985	0.86	0.814	0.906
YOLOv8s	Ne	640	11950	3650	8300	168	11.13	28.4	0.957	0.944	0.971	0.945	0.934	0.956	0.979	0.972	0.986	0.894	0.849	0.939
YOLOv8m	Ne	320	11950	3650	8300	218	25.84	78.7	0.946	0.924	0.967	0.947	0.932	0.962	0.975	0.966	0.985	0.877	0.829	0.925
YOLOv8n	Da	640	11950	3650	8300	168	3.01	8.1	0.956	0.944	0.967	0.949	0.934	0.964	0.978	0.972	0.985	0.899	0.857	0.941
YOLOv8s	Da	640	11950	3650	8300	168	11.13	28.4	0.96	0.948	0.972	0.946	0.934	0.958	0.978	0.972	0.984	0.915	0.878	0.952
YOLOv8m	Da	320	11950	3650	8300	218	25.84	78.7	0.957	0.946	0.969	0.948	0.936	0.961	0.978	0.971	0.985	0.908	0.868	0.948
YOLOv8m	Da	480	11950	3650	8300	218	25.84	78.7	0.965	0.961	0.969	0.951	0.938	0.965	0.98	0.974	0.986	0.92	0.882	0.957

Tablica 2: Rezultati validacije pojedinog modela

Broj slojeva, parametara te GFLOPs-a se mijenjao ovisno o korištenom YOLOv8 modelu. Po preciznosti je najgori YOLOv8m model koji nije koristio unaprijed treniran model te je koristio slike veličine 320×320, a YOLOv8m s unaprijed treniranim modelom i veličinom slike od 480×480 je imao najbolji rezultat. Već kod preciznosti se može vidjeti značajna razlika između najboljeg te najgoreg modela koji se razlikuju samo oko veličine slike te možemo vidjeti da sama veličina slike ima veliko značenje prilikom treniranja. Po vrijednostima odziva se može zaključiti da su YOLOv8n i YOLOv8s bez unaprijed treniranog modela imali najgori rezultat, a YOLOv8m s unaprijed treniranim modelom i slikama 480×480 je ponovno imao najbolje vrijednosti. U tablici je moguće uočiti vidljivu razliku između modela koji su trenirani na unaprijed treniranom modelu te koji su od “nule” trenirani gdje treniranje na unaprijed treniranom (fine-tune) može dovesti do boljih rezultata. Za mAP50 se može uočiti isti trend koji je prethodno opisan. Za vrijednosti mAP50-95 se nastavlja trend gdje je model YOLOv8m s unaprijed treniranim modelom te korištenim slikama 480×480 imao najbolji rezultat među svim treniranim modelima.

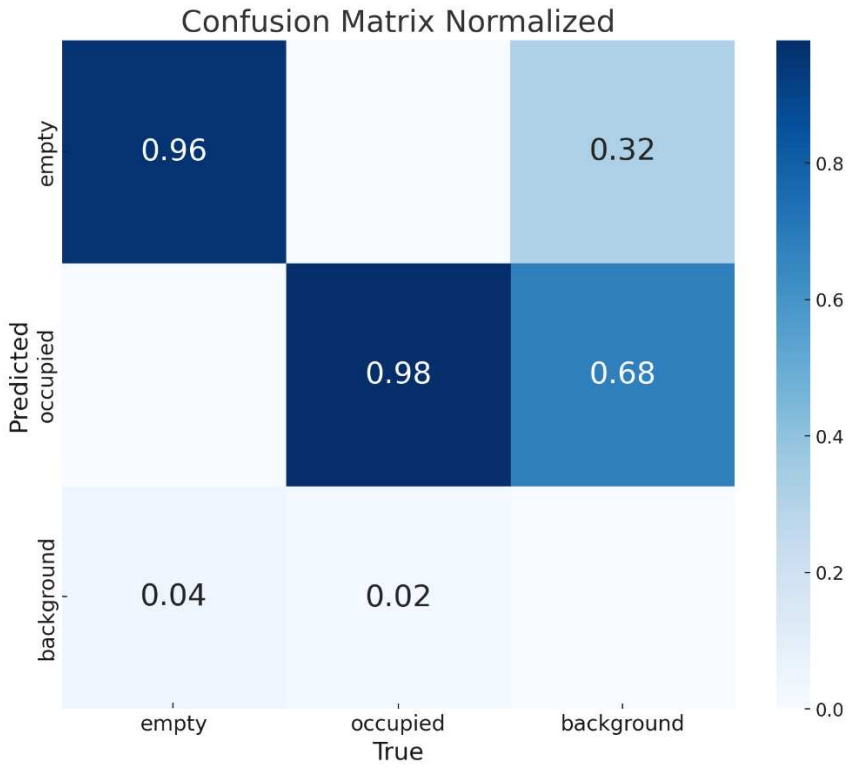
Na temelju slike 16 može se zaključiti da je model YOLOv8m s unaprijed treniranim modelom i slikama 480×480 prilikom klasifikacije "occupied" (zauzetih) mjesta imao 8109 ispravnih predikcija. U prepoznavanju "empty" (slobodnih) mjesta je imao 3514 ispravnih predikcija. Model je 3 puta pogrešno klasificirao "empty" kao "occupied" te je 9 puta pogrešno klasificirao "occupied" kao "empty". Broj pogrešnih klasifikacija u pozadinskoj kategoriji je 133 puta za "empty" i 182 puta za "occupied". Najveći broj točnih predikcija je za "occupied", što sugerira da model dobro prepoznaje zauzeta mjesta. Postoji mala, ali vidljiva pogreška između "empty" i "background" te između "occupied" i "background". Vrijednost 377 označava broj slučajeva gdje je model pogrešno klasificirao "background" (pozadinu) kao "empty" (prazno) mjesto. Vrijednost 792 označava broj slučajeva gdje je model pogrešno klasificirao "background" (pozadinu) kao "occupied" (zauzeto) mjesto. Kada je riječ o rezultatima nakon treninga i validacije vidljivo je da upravo prokomentirani model ima bolje rezultate od drugih. Donje desno polje je prazno jer "background" (pozadina) nije definirana kao klasa.



Slika 16: Matrica zabune za YOLOv8m (imgsz=480) s unaprijed treniranim modelom

U slici 17 je moguće vidjeti normaliziranu matricu zabune koja prikazuje omjer točnih i netočnih predikcija. Gledajući gornji lijevi kvadrant, vidimo da je model točno predvidio 96% praznih mjesta kao prazna. U gornjem srednjem te gornjem desnom kvadrantu nema vrijednosti, što znači da model nikad ili veoma rijetko nije predvidio zauzeto mjesto kao prazno i prazno mjesto kao zauzeto. Važno je napomenuti da se polje gleda prazno tj da je uvijek ispravno predviđeno samo u slučaju da je vrijednost 0 ili da je vrijednost znatno manja od 0.1 jer se onda zaokruži na manju vrijednost koja iznosi 0. U gornjem desnom kvadrantu model je 32% pozadinskih mjesta predvidio kao prazna. Središnji kvadrant ima vrijednost 0.98 što je indikacija da je model točno predvidio 98% zauzetih mjesta kao zauzeta. Srednji desni kvadrant otkriva da je model 68% pozadinskih mjesta predvidio kao zauzeta. U donjem lijevom kvadrantu, 4% praznih mjesta je predviđeno kao pozadina, dok je 2% zauzetih mjesta predviđeno kao pozadina, što je prikazano u donjem srednjem kvadrantu. Donji desni kvadrant nema vrijednosti, što znači da model nikad nije predvidio pozadinsko mjesto kao pozadinu jer "background" (pozadina) nije definirana kao klasa. Ova matrica je u postocima ponovno dokazala da model YOLOv8m s unaprijed treniranim modelom i

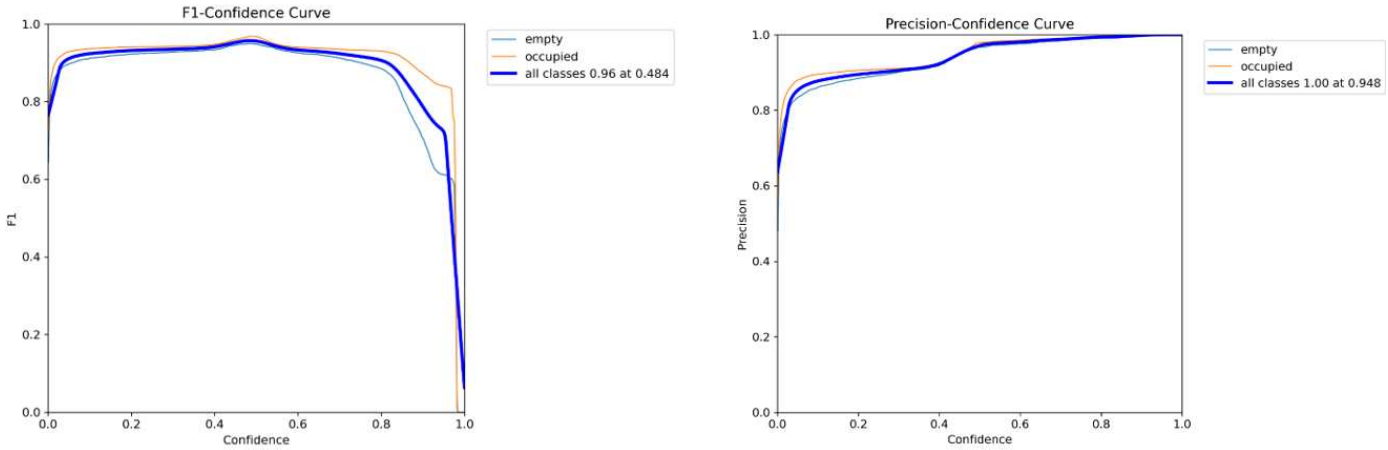
slikama 480×480 ima među svim svojim predviđanjima najbolji rezultat kod prepoznavanja zauzetih mjesta.



Slika 17: Normalizirana matrica zabune za YOLOv8m (imgsz=480) s unaprijed treniranim modelom

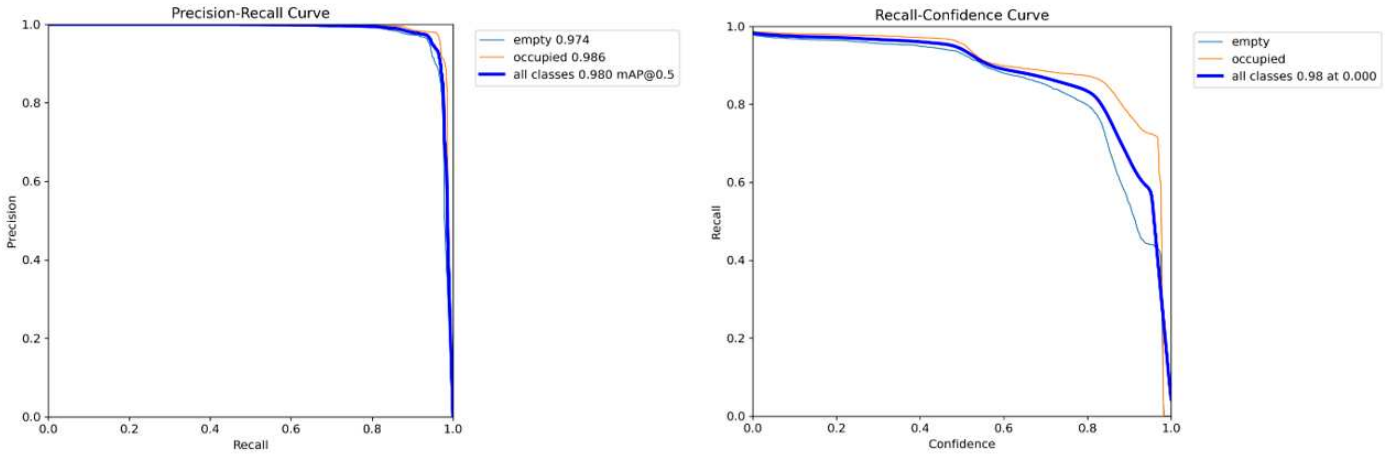
Slika 18 prikazuje krivulju pouzdanosti F1 i preciznost-pouzdanost krivulju za model YOLOv8m s unaprijed treniranim modelom i slikama 480×480. Iz krivulje pouzdanosti F1 može se zaključiti da model ima najbolje performanse pri srednjim vrijednostima pouzdanosti(engl. confidence) za obje klase, "empty" i "occupied", dok performanse padaju pri vrlo visokim vrijednostima pouzdanosti, pri “početku” kraja posebno za klasu "empty", a pri “kraju” kraja posebno za klasu “occupied”. Maksimalna vrijednost F1 za sve klase je 0.96 pri pouzdanosti od 0.484. Preciznost-pouzdanost krivulja pokazuje da preciznost za obje klase raste s povećanjem pouzdanosti, dosežući visoke vrijednosti preko 0.9. Maksimalna vrijednost preciznosti za sve klase je 1.00 pri pouzdanosti od 0.948. To sugerira da model ima visoku preciznost pri višim razinama pouzdanosti, ali može gubiti na F1 vrijednost zbog kompromisa između preciznosti i

pouzdanosti. Ovi grafovi pomažu u razumijevanju kako odabir različitih pragova pouzdanosti može utjecati na performanse modela.



Slika 18: Krivulja pouzdanosti F1 i krivulja preciznost-pouzdanost za YOLOv8m (imgsz=480) s unaprijed treniranim modelom

Slika 19 prikazuje preciznost-odziv krivulju i odziv-pouzdanost krivulju za model YOLOv8m s unaprijed treniranim modelom i slikama 480×480. Preciznost-odziv krivulja pokazuje da model postiže visoku preciznost i odziv za obje klase, "empty" i "occupied", s prosječnom preciznošću od 0.974 za "empty" i 0.986 za "occupied". Sveukupno, prosječna preciznost za sve klase (mAP@0.5) je 0.980, što ukazuje na izvrsne performanse modela. Odziv-pouzdanost krivulja prikazuje kako odziv varira s promjenom pouzdanosti, pokazujući da model održava visok odziv pri nižim razinama pouzdanosti, ali odziv opada pri višim razinama pouzdanosti, posebno za klasu "empty". Pri najvišim razinama pouzdanosti je vidljivo da odziv za klasu “occupied” strmovito te naglo opada. Maksimalni odziv za sve klase je 0.98 pri pouzdanosti od 0.000. Ovi grafovi sugeriraju da model ima visoku preciznost i dobar odziv, ali performanse u pogledu odziva opadaju pri višim pragovima pouzdanosti, što može utjecati na točnost modela u određenim scenarijima.



Slika 19: Krivulja preciznost-odziv i krivulja odziv-pouzdanost za YOLOv8m (imgsz=480) s unaprijed treniranim modelom

Jedini model koji je prokomentiran je model YOLOv8m s unaprijed treniranim modelom koji je koristio slike 480×480, a njega sam prokomentirao zato jer je imao najbolje rezultate prilikom validacije ali i kasnije prilikom testiranja/predviđanja. Prokomentirao sam vizualne elemente za prethodno navedeni model te naglašavam da se ostali vizualni elementi drugih treniranih modela nalaze u GitHub repozitoriju te da za njih vrijedi isti princip očitavanja vrijednosti i donošenja zaključaka na temelju toga. Poveznica na GitHub repozitoriji: https://github.com/Kovinejtor/Parking-Slot-Detection. Nakon završetka treninga se u putanji *runs/detect* generira direktoriji *trainBroj*. Nakon validacije se u putanji runs/detect generira *valBroj*, a nakon testiranja se u putanji *runs/detect* generira *predictBroj*. Svaki od njih sadrži određene elemente koji su izlazni rezultati njihovog procesa kao npr model, krivulja i sl. Navesti ću koji direktoriji su vezani uz koji model i parametre:

- YOLOv8n bez unaprijed treniranog modela: train7, val7, predict7
- YOLOv8s bez unaprijed treniranog modela: train5, val5, predict5
- YOLOv8m bez unaprijed treniranog modela i sa slikama 320×320: train6, val6, predict6
- YOLOv8n s unaprijed treniranim modelom: train2, val2, predict2
- YOLOv8s s unaprijed treniranim modelom: train, val, predict
- YOLOv8m s unaprijed treniranim modelom i sa slikama 320×320: train3, val3, predict3
- YOLOv8m s unaprijed treniranim modelom i sa slikama 480×480: train4, val4, predict4

6. RAZMATRANJE REZULTATA

6.1. Usklađenost rezultata s postavljenim ciljevima

Za provjeru kvalitete različitih modela se treba koristiti testni skup koji su za model neviđeni podaci. YOLO nudi mogućnost da se s funkcijom *predict* prikažu te spreme izlazne slike s oznakama koje su rezultat predviđanja modela na neviđeni skup, ali prilikom toga nemamo nikakve izlazne evaluacijske metrike na temelju kojih možemo usporediti modele. U nastavku rada ću demonstrirati rezultate funkcije *predict* koristeći najbolji model, ali prvo ću prikazati način za prikaz evaluacijskih metrika na temelju testnog skupa. U slici 20 je moguće vidjeti da sam promijenio za validaciju putanju i to tako da joj putanja referencira testni skup. Izvest će validaciju, ali na testnom skupu te će se saznati evaluacijske metrike. Na GitHub repozitoriju se nalazi nemodificirani *data.yaml* te u slučaju da čitatelj želi stvoriti validaciju za testni skup podataka onda treba samo promijeniti putanju za validaciju kao što je prikazano u slici 20. Za prikaz evaluacijskih metrika je potrebno pokrenuti postojeći kod za validaciju i to je potrebno napraviti za svaki model.

```
train: C:/ParkingSlotDetection/datasets/Dataset/train/images
val: C:/ParkingSlotDetection/datasets/Dataset/test/images   #prilikom testiranja staviti /test/ kako bi se na neviđenim podacima provjerili modeli
test: C:/ParkingSlotDetection/datasets/Dataset/test/images

nc: 2
names: ['empty', 'occupied']
```

Slika 20: Prikaz modificiranog data.yaml

```
best_model_path = 'runs/detect/train/weights/best.pt'
best_model = YOLO(best_model_path)
metrics = best_model.val(data='datasets/Dataset/data.yaml', imgsz=640)
print(metrics)
```

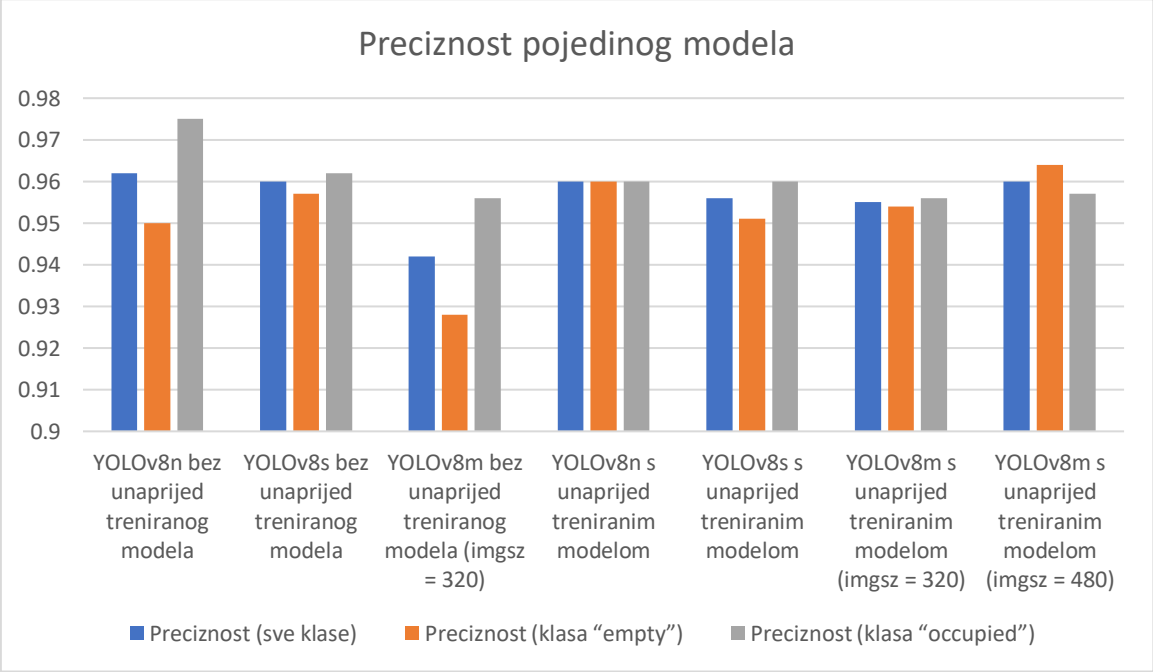
Slika 21: Prikaz koda za jednu validaciju

U tablici 3 mogu se uočiti rezultati evaluacijskog procesa testnog skupa podataka. YOLOv8n bez unaprijed treniranog modela je znatno bolji, ali vrijednosti mAP50-95 su znatno lošije u usporedbi s rezultatima iz validacijskog skupa podataka. YOLOv8s s unaprijed treniranim modelom je jednako/manje bolji, a YOLOv8m s unaprijed treniranim modelom te slikama 480×480 je i dalje najbolji model. Kod njega je jedino preciznost smanjena, ali su druge izlazne jedinice u boljim vrijednostima. Ostali modeli su po pitanju rezultata u jednako/malo lošijem stanju u usporedbi s rezultatom evaluacije nad validacijskim skupom podataka.

Model YOLO-a / Specifikacije i metrike	**Korištenje unaprijed treniranog modela**	**Veličina slike**	**Instance (sve klase)**	**Instance (klasa “empty”)**	**Instance (klasa “occupied”)**	**Preciznost (sve klase)**	**Preciznost (klasa “empty”)**	**Preciznost (klasa “occupied”)**	**Odziv (sve klase)**	**Odziv (klasa “empty”)**	**Odziv (klasa “occupied”)**	**mAP50 (sve klase)**	**mAP50 (klasa “empty”)**	**mAP50 (klasa “occupied”)**	**mAP50-95 (sve klase)**	**mAP50-95 (klasa “empty”)**	**mAP50-95 (klasa “occupied”)**
YOLOv8n	Ne	640	6404	2010	4394	0.962	0.95	0.975	0.949	0.922	0.975	0.982	0.974	0.991	0.849	0.8	0.898
YOLOv8s	Ne	640	6404	2010	4394	0.96	0.957	0.962	0.942	0.921	0.963	0.979	0.97	0.988	0.882	0.836	0.927
YOLOv8m	Ne	320	6404	2010	4394	0.942	0.928	0.956	0.944	0.919	0.97	0.973	0.96	0.987	0.858	0.805	0.911
YOLOv8n	Da	640	6404	2010	4394	0.96	0.96	0.96	0.949	0.92	0.977	0.978	0.968	0.988	0.886	0.841	0.932
YOLOv8s	Da	640	6404	2010	4394	0.956	0.951	0.96	0.952	0.931	0.973	0.983	0.978	0.989	0.911	0.875	0.948
YOLOv8m	Da	320	6404	2010	4394	0.955	0.954	0.956	0.953	0.93	0.977	0.982	0.975	0.989	0.903	0.863	0.942
YOLOv8m	Da	480	6404	2010	4394	0.96	0.964	0.957	0.959	0.939	0.979	0.987	0.984	0.99	0.921	0.89	0.952

Tablica 3: Rezultati evaluacije pojedinih modela s testnim skupom podataka

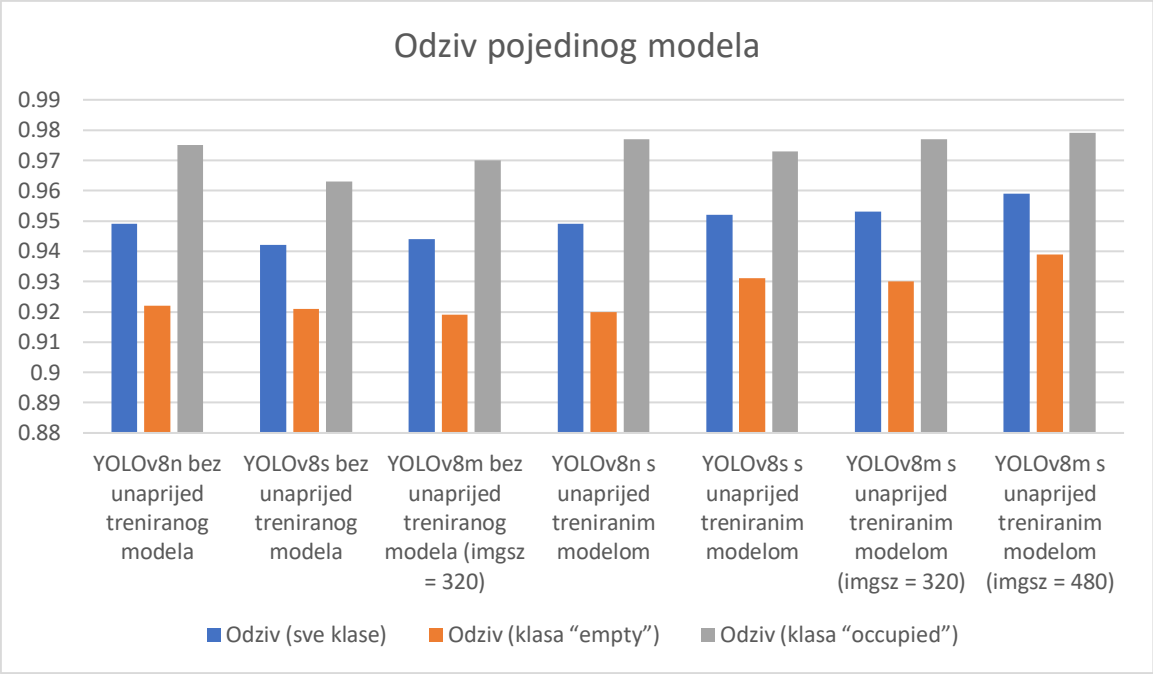
U slikama s grafovima sam koristio englesku skraćenicu *imgsz* koja označava veličinu slike te se numerička vrijednost *n* nakon jednakosti čita kao *n* × *n*. Neprisutnost *imgsz*-a znači da je korištena slika veličine 640 × 640. Na temelju slike 22 moguće je zaključiti da je YOLOv8n bez unaprijed treniranog modela najbolji kada je riječ o vrijednosti preciznosti koja je dobivena nakon testiranja modela. Ima sjajnu preciznost za klasu "occupied", a znatno lošiju za klasu "empty". Važno je naglasiti da je imao sjajan rezultat kada je riječ o preciznosti unatoč tome što je najmanji YOLOv8 model. Suprotno tome, dva od tri YOLOv8m modela su imali najgore vrijednosti preciznosti unatoč tome što su oni veći modeli od ostalih korištenih. U ovom slučaju je veličina slike imala važnu ulogu jer je YOLOv8n bez unaprijed treniranog modela koristio veličinu slike 640×640 prilikom treniranja, a YOLOv8m modeli su koristili slike manjih dimenzija. YOLOv8m s unaprijed treniranim modelom i korištenim slikama 480×480 u usporedbi s drugim modelima ima najveću vrijednost za preciznost klase "empty".



Slika 22: Grafički prikaz vrijednosti preciznosti pojedinog modela nakon testiranja

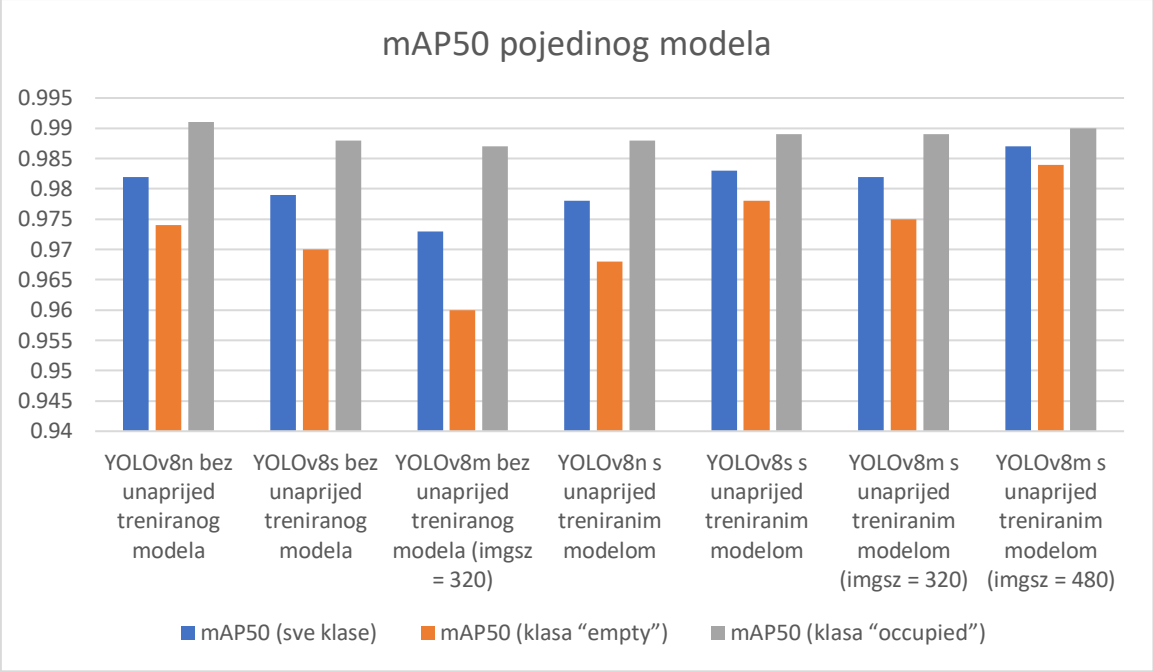
U slici 23 je moguće uočiti da svi modeli imaju sličnu vrijednost odziva. Može se jedino naglasiti da je model YOLOv8m s unaprijed treniranim modelom i korištenim slikama 480×480 bolji od drugih model, ali da je riječ o veoma maloj razlici u vrijednostima

odziva. Značajna je razlika u vrijednostima odziva između klase “occupied” i klase “empty”.



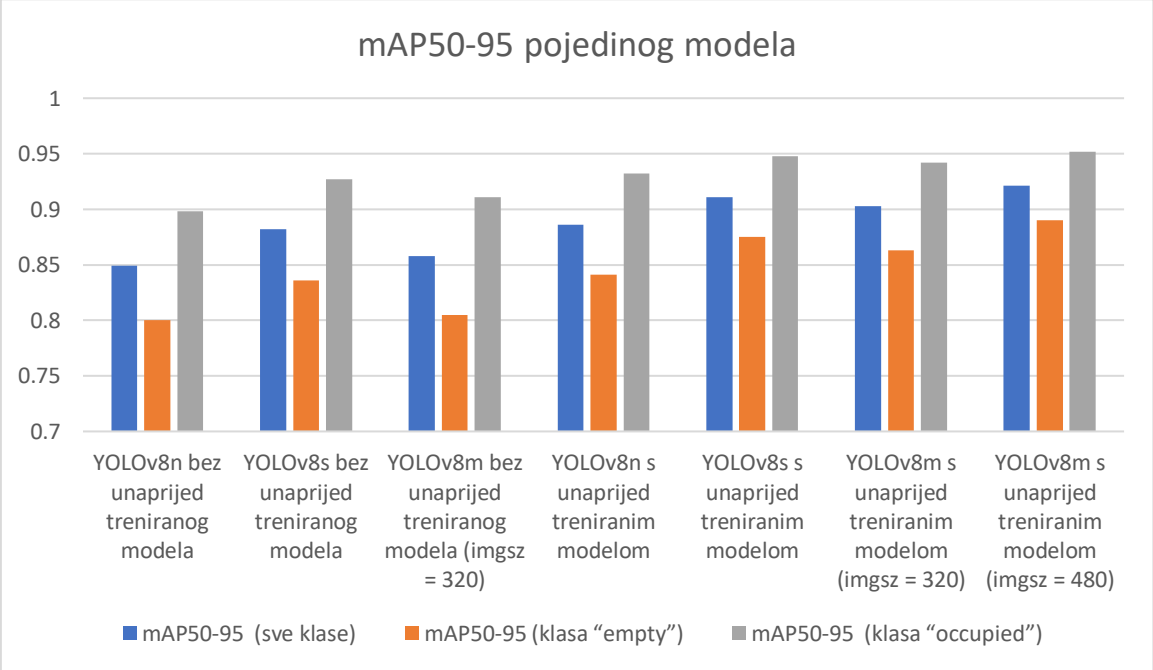
Slika 23: Grafički prikaz vrijednosti odziva pojedinog modela nakon testiranja

Na temelju slike 24 se može zaključiti da model YOLOv8m s unaprijed treniranim modelom i korištenim slikama 480×480 ima ponovno najbolje rezultate, a ovaj put je riječ o vrijednosti mAP50. YOLOv8n bez unaprijed treniranog modela ima veoma visoku vrijednost mAP50 za klasu “occupied” koja skoro dostiže 100%-tnu vrijednost tj 1. YOLOv8m bez unaprijed treniranog modela i korištenim slikama 320×320 ima najniže vrijednosti mAP50.



Slika 24: Grafički prikaz vrijednosti mAP50 pojedinog modela nakon testiranja

U slici 25 moguće je uočiti da je za vrijednost mAP50-95 vidljiva značajna razlika između modela koji su koristili unaprijed trenirani model te onih koji nisu koristili unaprijed trenirani model. Nastavlja se trend gdje se YOLOv8m s unaprijed treniranim modelom i korištenim slikama 480×480 ponovno pokazuje kao najbolji, a YOLOv8n bez unaprijed treniranog modela je imao najniže vrijednosti.



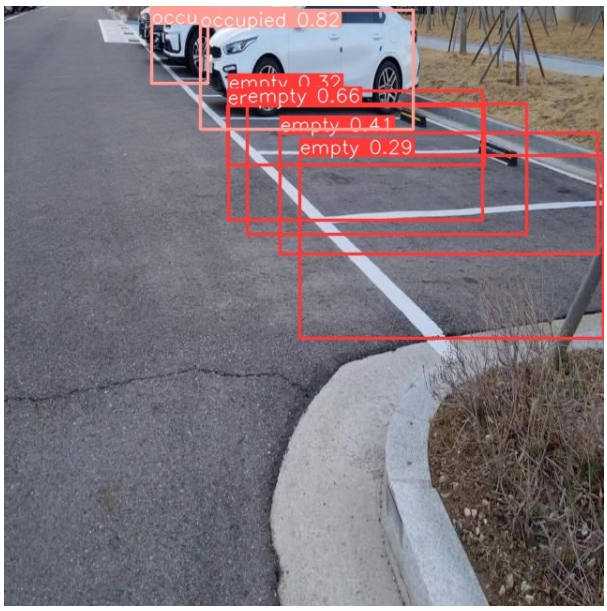
Slika 25: Grafički prikaz vrijednosti mAP50-95 pojedinog modela nakon testiranja

Model YOLOv8m koji je treniran s unaprijed treniranim modelom i s veličinom slika 480×480 je pokazao najbolji rezultat tijekom procesa treninga te validacije, ali i evaluacije s testnim skupom podataka što je bilo očekivano sukladno tome što je najveći te najprecizniji model među korištenima. Korištenje unaprijed treniranog modela je bilo ključno za bolji uspjeh modela, a finalni produkt je model koji bi iz slike mogao raspoznati slobodna ili zauzeta parkirna mjesta sa zadovoljavajućim rezultatima. Opisat ću i prikazati slikama rezultate najboljeg modela, a oni su rezultat funkcije *predict* koja je koristila testni skup podataka te prikazala rezultat u slikama s oznakama. Slike će se spremiti u putanji koja je navedena u kodu u novogenerirani direktorij *predictBroj.*

```
test_results = best_model.predict(source='datasets/Dataset/test/images', imgsz=640, conf=0.25, save=True, save_dir='output/predictions')
```

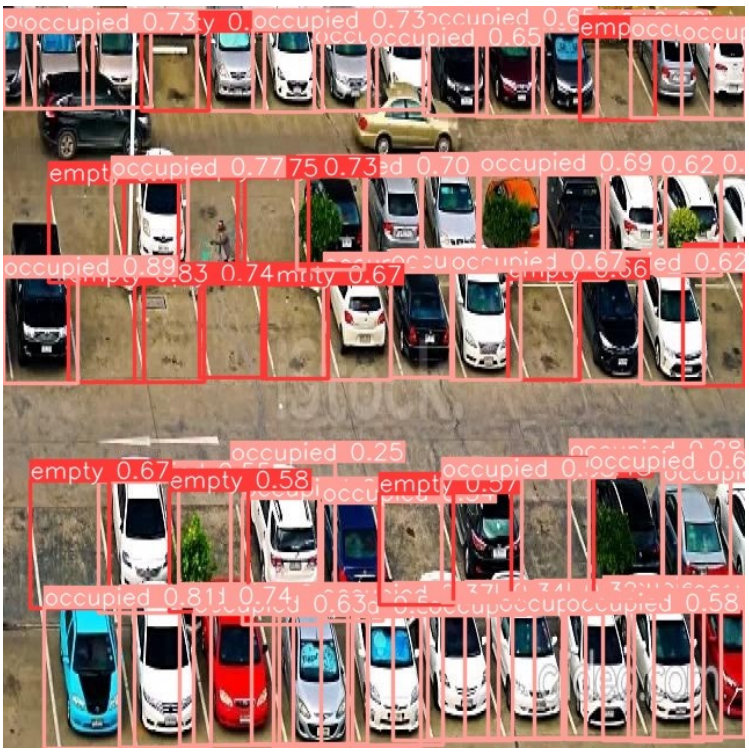
Slika 26: Prikaz koda za testiranje/predviđanje

Važno je naglasiti da je skup podataka bio izazovan za model kada je riječ o raspoznavanju parkirnih mjesta upravo jer su kod određenih slika kutovi postavljenih kamera bili postavljeni lošije te je stoga teže raspoznati pozadinska parkirna mjesta. Unatoč tome model je može raspoznati parkirna mjesta u izazovnim situacijama, ali u nekim jednostavnim situacijama ponekad griješi što ukazuje da unatoč validacijskim rezultatima i dalje ima izraziti prostor za napredak. Iz slike 27 se može zaključiti da je sjajno prepoznao stanje parkirnih mjesta uz izuzetak jednog/dva koji nisu ni ljudskom oku iz prvog pogleda vidljivi.



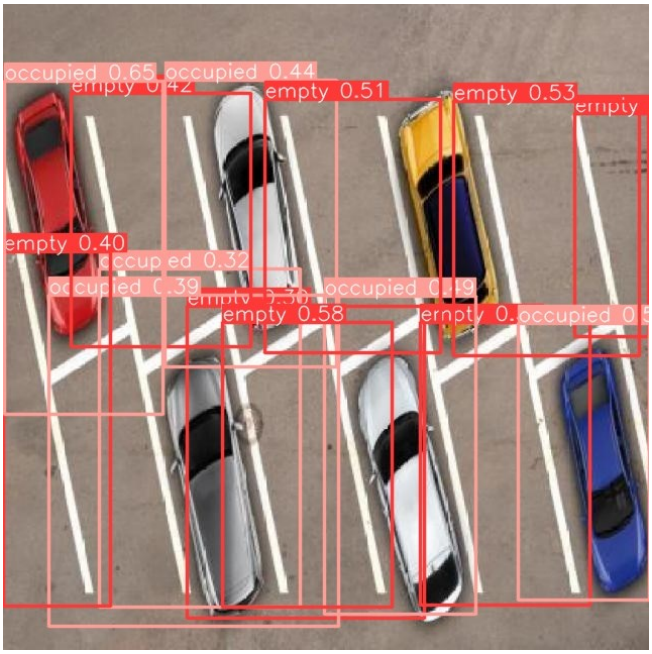
Slika 27: Prvi primjer rezultata predikcije na testnom skupu podataka

Iz slike 28 jasno je vidljivo da model sjajno prepoznaje stanja parkirnih mjesta uz izuzetak jednog praznog parkirnog mjesta. U lijevoj sredini slike se nalazi jedno dulje crno vozilo te upravo zbog njegove duljine model smatra da on zauzima i prostor iza njega.

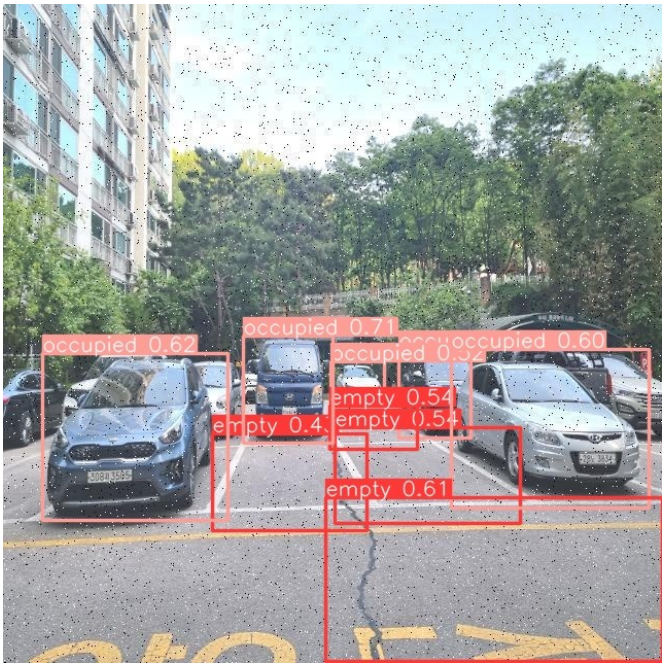


Slika 28: Drugi primjer rezultata predikcije na testnom skupu podataka

Slika 29 i slika 30 prikazuju da model prepoznaje većinu parkirnih mjesta ispravno, ali unatoč tome i dalje su prisutne zabune prilikom klasifikacije te je prisutno teže prepoznavanje parkirnog mjesta zbog ugla kamere.

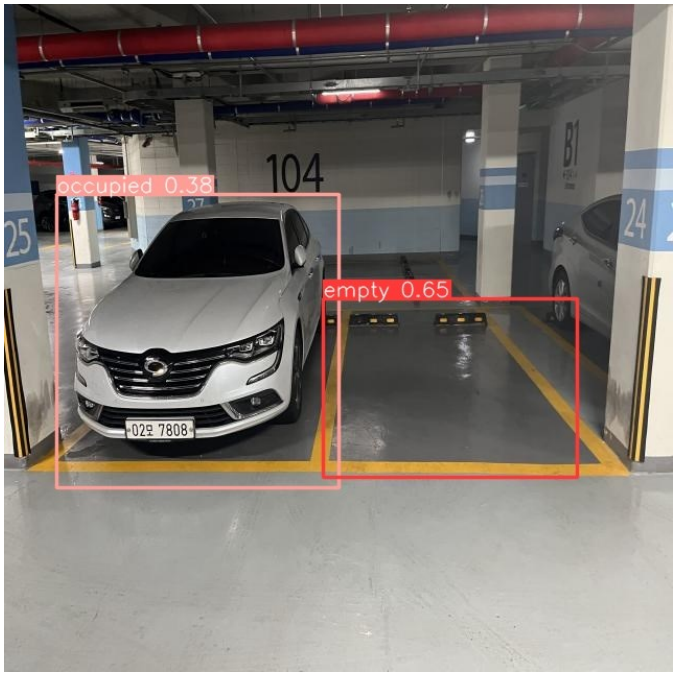


Slika 29: Treći primjer rezultata predikcije na testnom skupu podataka



Slika 30: Četvrti primjer rezultata predikcije na testnom skupu podataka

Iz slike 31 je vidljivo da model prepoznaje sva parkirna mjesta te ispravno pridružuje klase uz izuzetak bočnog auta kojeg je teško uočiti tj čije je parkirno mjesto teže za uočiti. Slika 32 ukazuje na to da model uopće nije prepoznao lijevo parkirno mjesto.



Slika 31: Peti primjer rezultata predikcije na testnom skupu podataka

Slika 32: Šesti primjer rezultata predikcije na testnom skupu podataka

6.2. Ograničenja rada

U ovom radu primijenjen je YOLO algoritam verzija 8 (YOLOv8) za prepoznavanje i analizu parkirnih mjesta. Treniranje modela izvedeno je na laptopu s ograničenim hardverskim resursima: 16 GB RAM-a, NVIDIA GeForce RTX 3050 Ti Laptop GPU s 4096 MiB memorije, te korištenjem Ultralytics YOLOv8.2.22, Python 3.10.11, s CUDA 11.8 podrškom. Ova konfiguracija uvodi nekoliko ključnih ograničenja koje treba uzeti u obzir prilikom evaluacije rezultata rada. Korišteno računalo ima ograničenu količinu RAM-a i GPU memorije, što direktno utječe na mogućnost treniranja većih modela i obrade velikih datasetova. U praksi, ovo je značilo da sam se morao odlučiti za manje modele YOLOv8 algoritma, konkretno YOLOv8n, YOLOv8s, i YOLOv8m. Ovi modeli su izabrani jer troše manje resursa i memorije te omogućuju brže treniranje, što je bilo ključno zbog ograničenja dostupnih računalnih resursa.

Odabirom manjih i osrednjih modela (YOLOv8n, YOLOv8s, YOLOv8m), postignut je kompromis između brzine treniranja i preciznosti modela. Manji modeli, kao što je YOLOv8n, troše manje memorije i resursa te omogućuju brže treniranje, ali to dolazi uz cijenu smanjenja preciznosti u usporedbi s većim modelima kao što su YOLOv8l i YOLOv8x. Na primjer, YOLOv8x, koji je najpreciznji model, zahtijeva značajno više memorije i resursa, te bi treniranje na korištenom računalu bilo izuzetno sporo i neefikasno.

Skup podataka korišten za treniranje modela je bio zadovoljavajući, ali s boljim hardverskim resursima bi izabrao veći i raznolikiji skup podataka. Treniranje modela na ograničenom hardveru zahtijeva više vremena, što može ograničiti broj eksperimenata i iteracija koje se mogu provesti unutar zadanog vremenskog okvira rada. Zbog toga je bilo teško provesti opsežnu evaluaciju i fino podešavanje hiperparametara modela, što bi potencijalno moglo poboljšati performanse.

7. ZAKLJUČAK

7.1. Implikacije rezultata za praktičnu primjenu

Rezultati primjene YOLOv8 algoritma za prepoznavanje i analizu parkirnih mjesta pokazali su da je model YOLOv8m s korištenjem unaprijed treniranog modela, 100 epoha, veličinom slike 480×480, i serije (engl. batch) veličine 16, dao najbolje

rezultate. Vrijednost preciznosti te ostalih važnih metrika ovog modela tijekom validacije i testiranja su bolje nego vrijednosti metrika drugih modela, što ga čini najprikladnijim za praktičnu primjenu među svim isprobanim modelima i hiperparametrima. Važno je naglasiti da su se kod određenih metrika manji YOLOv8 modeli pokazali bolji nego određeni veći tj osrednji YOLOv8 modeli, ali unatoč tome prethodno specificirani YOLOv8m model je sveukupno najbolji. Model YOLOv8m, zahvaljujući svojim sjajnim rezultatima, može se integrirati u sustave za upravljanje parkirnim mjestima u stvarnom vremenu. Ovo uključuje aplikacije za vođenje korisnika do slobodnih parkirnih mjesta, nadzor nad zauzetošću parkirnih mjesta i automatizirane sustave za naplatu parkiranja. Visoka točnost modela omogućava točnije informacije za korisnike, smanjujući frustracije uzrokovane netočnim podacima o slobodnim parkirnim mjestima. Ovo može povećati zadovoljstvo korisnika i efikasnost parkirnih sustava. Iako je YOLOv8m postigao visoku točnost, veći modeli poput YOLOv8l ili YOLOv8x, uz adekvatan hardver, mogu potencijalno postići još veću preciznost. Navedeni model ima dobru preciznost i ostale izlazne evaluacijske metrike, ali trebalo bi se težiti da se za praktičnu primjenu koristi još precizniji, robusniji te kvalitetniji model.

7.2. Preporuke za daljnji rad

Kao što je navedeno u poglavlju o ograničenjima provođenog rada, treniranje modela izvedeno je na računalu s ograničenim resursima. Korištenje naprednijeg hardvera s više GPU-ova ili s većom količinom RAM-a, omogućilo bi treniranje većih modela YOLOv8, kao što su YOLOv8l i YOLOv8x. Ovi modeli pružaju veću preciznost u prepoznavanju objekata, što bi moglo značajno poboljšati performanse sustava. Jedan od ključnih faktora za poboljšanje modela je korištenje većeg i raznovrsnijeg skupa podataka. Preporučio bih prikupljanje dodatnih podataka iz različitih izvora i uvjeta, uključujući slike parkirnih mjesta s različitih lokacija, pod različitim vremenskim uvjetima i u različito doba dana. Ovo će pomoći modelu da bolje generalizira i poveća njegovu robusnost u stvarnim situacijama. Korisno bi bilo istražiti mogućnost kombiniranja YOLO algoritma s drugim tehnikama strojnog učenja i računalnog vida. Na primjer, korištenje modela za praćenje objekata može pomoći u praćenju vozila koja ulaze i izlaze iz parkirnih mjesta, čime se dodatno povećava preciznost i korisnost sustava. Prepoznavanje anomalija, poput nepravilnog parkiranja ili zauzetosti mjesta koja su označena kao slobodna, može biti korisno za unapređenje sustava. Razvijanje

i implementacija algoritama za prepoznavanje ovakvih anomalija može pomoći u pružanju točnijih informacija korisnicima. Za daljnji rad preporučuje se implementacija i testiranje modela u stvarnim uvjetima. Ovo uključuje integraciju modela u sustav nadzora parkirnih mjesta u stvarnom vremenu, te evaluaciju njegove performanse na terenu. Ovime će se dobiti vrijedne povratne informacije koje mogu pomoći u daljnjem usavršavanju modela.

8. LITERATURA

[1] Gaudenz Boesch, Object Detection in 2024: The Definitive Guide. https://viso.ai/deep-learning/object-detection/

[2] YOLOv8 službena web stranica. https://yolov8.com/

[3] Ultralytics YOLO dokumentacija, home. https://docs.ultralytics.com/

[4] Ultralytics YOLO dokumentacija, modes. https://docs.ultralytics.com/modes/

[5] Ultralytics YOLO dokumentacija, COCO Dataset.
https://docs.ultralytics.com/datasets/detect/coco/

[6] CC BY 4.0, Attribution 4.0 International
https://creativecommons.org/licenses/by/4.0/

[7] Funck, W., Möhler, N., & May, F. (2004). "Determining Car-Park Occupancy from Single Images." Proceedings of the IEEE Intelligent Vehicles Symposium, 2004.

[8] Amato, G., Carrara, F., Falchi, F., Gennaro, C., & Vairo, C. (2017). "Car parking occupancy detection using smart camera networks and deep learning." Proceedings of the IEEE Symposium on Computers and Communications (ISCC), 2017.

[9] Lin, T., Rivano, H., & Le Mouel, F. (2018). "A survey of smart parking solutions." IEEE Transactions on Intelligent Transportation Systems, 2018.

[10] Muhammad Syihab, Parking Space Image Dataset
https://universe.roboflow.com/muhammad-syihab-bdynf/parking-space-ipm1b/dataset/4

[11] Pierluigi Siano, A Review of Parking Slot Types and their Detection Techniques for Smart Cities. https://www.mdpi.com/2624-6511/6/5/119

[12] Juan Pedro, Detailed Explanation of YOLOv8 Architecture. https://medium.com/@juanpedro.bc22/detailed-explanation-of-yolov8-architecture-part-1-6da9296b954e

[13] Jacob Solawetz, Francesco, What is YOLOv8? The Ultimate Guide. https://blog.roboflow.com/whats-new-in-yolov8/

[14] Ultralytics YOLO dokumentacija, Performance Metrics Deep Dive. https://docs.ultralytics.com/guides/yolo-performance-metrics/

[15] Edmundo Casas, Leo Ramos, Eduardo Bendek and Francklin Rivas-Echeverría, Assessing the Effectiveness of YOLO Architectures for Smoke and Wildfire Detection. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10239390

[16] Ultralytics YOLO dokumentacija , Train Custom Data. https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/

[17] University Information Technology Services, Understand measures of supercomputer performance and storage system capacity. https://kb.iu.edu/d/apeq

[18] Ultralytics YOLO dokumentacija, YOLOv8.

https://docs.ultralytics.com/models/yolov8/

[19] Zoumana KEITA, YOLO Object Detection Explained. https://www.datacamp.com/blog/yolo-object-detection-explained

[20] Sergio Antonio Sánchez Hernández, A review: Comparison of performance metrics of pretrained models for object detection using the TensorFlow framework. https://www.researchgate.net/figure/Comparison-of-frames-processed-per-second-FPS-implementing-the-Faster-R-CNN-R-FCN-SSD_fig6_342570032

[21] Gaudenz Boesch, What is Intersection over Union (IoU)?. https://viso.ai/computer-vision/intersection-over-union-iou/

[22] Deval Shah ,Mean Average Precision (mAP) Explained: Everything You Need to Know. https://www.v7labs.com/blog/mean-average-precision

[23] Ren, Shaoqing, et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks" Advances in neural information processing systems 28.

[24] Kaiming He et al. “Mask R-CNN” ICCV 2017.

[25] Wei Liu et al. “SSD: Single Shot MultiBox Detector” European Conference on Computer Vision.

[26] Tsung-Yi Lin et al. “Focal Loss for Dense Object Detection”.

[27] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”.

[28] Hanqing Bi et al. “Comparing one-stage and two-stage learning strategy in object detection”.

[29] Jacob Murel Ph.D., Eda Kavlakoglu “What is object detection?” https://www.ibm.com/topics/object-detection

[30] James Gallagher, “What is Object Detection? The Ultimate Guide”. https://blog.roboflow.com/object-detection/

9. POPIS ELEMENATA

9.1. Popis slika

9.2. Popis tablica

9.3. Popis formula