Vizijski sustav za prebrojavanje osoba temeljen na YOLOv8 modelu za prepoznavanje objekata

Lovreković, Lora

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:979223

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Lora Lovreković

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor: Student:

izv. prof. dr. sc. Tomislav Stipančić, dipl. ing. Lora Lovreković

Zagreb, 2023.

Izjavljujem da sam ovaj rad izradila samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se mentoru izv. prof. dr. sc. Tomislavu Stipančiću na pruženoj potpori i savjetima tijekom pisanja rada, ali i na pozitivnoj energiji koju širi kroz svoj rad na fakultetu. Hvala Robertu na savjetu za reference, te Dominiku i Matei što su bili statisti u jednom od videa u ovome radu. Tijekom diplomskog studija najviše sam zahvalna na iskustvu studiranja u inozemstvu, stoga hvala i Ani koja je sa mnom proživjela tih pet mjeseci.

Zahvaljujem i svim onim prijateljima koji su mi uvijek podrška i na koje se uvijek mogu osloniti. Veliko hvala naravno i mojoj obitelji, a najviše mami Ani.

Lora Lovreković

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
Proizvodno inženjerstvo, inženjerstvo materijala, industrijsko inženjerstvo i menadžment, mehatronika i robotika, autonomni sustavi i računalna inteligencija

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 - 04 / 23 - 6 / 1**	
Ur.broj: **15 - 23 -**	

DIPLOMSKI ZADATAK

Student: **Lora Lovreković** JMBAG: 0035214434

Naslov rada na hrvatskom jeziku: **Vizijski sustav za prebrojavanje osoba temeljen na YOLOv8 modelu za prepoznavanje objekata**

Naslov rada na engleskom jeziku: **A vision system for counting people based on the YOLOv8 object recognition model**

Opis zadatka:

Modeli strojnog vida omogućuju računalima da prepoznaju značajke na slikama te ih koriste kao gradivne elemente kod složenijih primjena u sklopu stvarne okoline. U tu se svrhu često koriste različite metode umjetne inteligencije temeljene na pred treniranim modelima pomoću kojih vizijski sustavi vrše akviziciju informacija, te uče ili identificiraju objekte, pojave ili osobe.
YOLO je model temeljen na konvolucijskoj neuronskoj mreži koji služi za otkrivanje različitih objekata na slici u ovisnosti o skupu podataka za trening. YOLO model čak može identificirati više objekata unutar iste slike te ih klasificirati u različite kategorije.
U radu je potrebno izraditi cjelovito softversko rješenje za prebrojavanje osoba u sklopu stvarne primjene temeljeno na YOLOv8 modelu za prepoznavanje, koje uključuje:

- model za pronalaženje i praćenje osoba u pokretu bez obzira na trenutnu orijentaciju te perspektivu gledanja, te
- algoritam za prebrojavanje pronađenih osoba.

Dobiveno softversko rješenje je potrebno eksperimentalno evaluirati uključivši ljudske subjekte te dati kritički osvrt na rad aplikacije.

U radu je potrebno navesti korištenu literaturu te eventualno dobivenu pomoć.

Zadatak zadan: 4. svibnja 2023.

Zadatak zadao: Izv. prof. dr. sc. Tomislav Stipančić

Datum predaje rada: 6. srpnja 2023.

Predviđeni datumi obrane: 17. – 21. srpnja 2023.

Predsjednik Povjerenstva: Prof. dr. sc. Ivica Garašić

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

SAŽETAK

Razvitak računalnog vida, grane dubokog učenja, zadnjih je godina doveo do napretka u području prepoznavanja objekata, koje računalima omogućuje identifikaciju, klasifikaciju i lokaciju objekata na slikama i videima. Za prepoznavanje objekata u ovome je radu korišten YOLOv8 algoritam, te je implementiran i model DeepSORT za praćenje objekata. U ovome radu prepoznavanje i praćenje bit će primijenjeno na ljudske osobe, koje će u konačnici biti prebrojane na temelju svojih kretanja. Nakon što se model primijeni na nekoliko primjera, slijedi analiza rezultata i prostori za poboljšanje rada modela.

Ključne riječi: duboko učenje, računalni vid, prepoznavanje objekata, praćenje objekata, lokalizacija, klasifikacija, YOLOv8, DeepSORT

SUMMARY

The development of computer vision, a branch of deep learning, in the last years has led to development of object detection, that enables computers to identify, classify and localize objects in photos and videos. In this paper, YOLOv8 algorithm is used to detect objects and the DeepSORT model for object tracking is implemented as well. Detection and tracking in this paper will be applied to people, who are going to be counted based on their movements. After the model has been applied to several examples, results will be analyzed and space for improvements will be presented.

Key words: deep learning, computer vision, object detection, object tracking, localization, classification, YOLOv8, DeepSORT

1. UVOD

Razvitak dubokog učenja, no posebice računalnog vida u zadnjih je nekoliko godina doveo do napretka u području prepoznavanja objekata na slikama i videozapisima. Prepoznavanje objekata omogućuje računalima da identificiraju, klasificiraju, ali i lokaliziraju objekte. Jedan od ključnih izazova s kojima se prepoznavanje objekata suočava primjena je u stvarnom vremenu, a uspješno prepoznavanje ljudskih osoba u stvarnome vremenu može imati brojne korisne primjene. Primjerice, može se koristiti za sigurnosno nadziranje, za promet, analizu ponašanja osoba, prebrojavanje osoba na nekoj lokaciji. U ovome radu za prepoznavanje osoba koristi se YOLOv8 (You Only Look Once) algoritam temeljen na metodama dubokog učenja. Ovaj je model jedan od najnaprednijih detektora objekata, a popularnost je stekao dajući izvrsne rezultate u primjeni u stvarnom vremenu, te zbog jednostavnosti i pristupačnosti korištenja.

Cilj ovoga rada primjena je modela za prepoznavanje i praćenje objekata na nekoliko specifičnih primjera. Na videozapisima koji su u radu korišteni, potrebno je prepoznati ljudske osobe pomoću YOLOv8 algoritma, te ih potom pratiti iz kadra u kadar implementacijom DeepSORT alata za praćenje. Rad je nakon uvodnog poglavlja podijeljen na šest poglavlja. U prvome poglavlju predstavljeni su razni modeli dubokog učenja, od čega su konvolucijske mreže najdetaljnije predstavljene, zbog činjenice da su temelj YOLO modela. U radu su predstavljeni razni detektori objekata, primjenjivi na slikama i videima. Njihovo kronološko predstavljanje u radu sa svakim modelom donosi određena unaprjeđenja, uz naglasak na YOLO modele i na promjene koji svaka nova verzija ovog modela nosi. Budući da zadatak nije moguće obaviti bez praćenja i DeepSORT modela, i ta će grana biti pobliže objašnjena.

Nakon teorijskog dijela fokusiranog na detektore objekata, slijedi detaljan opis implementacije YOLOv8 i DeepSORT modela na specifične primjere. Primjeri se sastoje od nekoliko videozapisa koji prikazuju ljude koji se kreću hodajući ili po pokretnim stepenicama, te je potrebno izvršiti prebrojavanje ljudi na temelju tih kretnji. U radu je korišten GitHub repozitorij koji je prvo testiran slijedeći upute kreatora repozitorija. Nakon što je potvrđeno da repozitorij radi ispravno, izvršena je primjena na videozapise ljudi preuzete s interneta, ali i na samostalno snimljene primjere. Osim objašnjenja najbitnijih linija koda, predstavljeni su i analizirani rezultati, ali i problemi koji postoje u radu aplikacije.

2. DUBOKO UČENJE

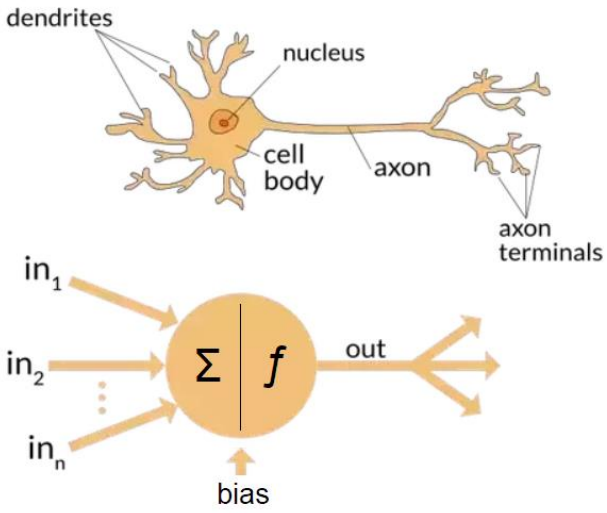
Strojno učenje u današnje doba uvelike pomaže društvu i pridonosi kvalitetnijem načinu života. Tehnike strojnog učenja implementirane su u uređajima koje svakodnevno koristimo, poput kamera, pametnih telefona te u algoritmima za prikazivanje ciljanog sadržaja na društvenim mrežama. Duboko učenje vrsta je strojnog učenja i dio umjetne inteligencije koja s ciljem oponašanja ljudskog mozga gradi duboke arhitekture i tako usvaja i apstraktne informacije čime se otvara širok spektar mogućnosti praktične primjene. [1] Zadnjih godina duboko učenje sve je korišteniji alat zahvaljujući svojim rezultatima kojima premašuje i dominira nad ostalim metodama umjetne inteligencije. Sve većoj popularnosti pridonijeli su razni faktori od kojih je najbitnije istaknuti razvoj velikih visokokvalitetnih javno dostupnih setova podataka te osnaživanje GPU računalstva, čime se značajno ubrzalo treniranje dubokih modela učenja. [2] Unaprjeđenje algoritama strojnog učenja, te pristupačne cijene računalnog hardvera također su odigrali bitnu ulogu u popularizaciji ovog alata. Duboko učenje ima veliku primjenu u računalnom vidu, poput prepoznavanja objekata, praćenje pokreta, prepoznavanje govora, prepoznavanje lica…

U nastavku ove cjeline objašnjene su neke od najpopularnijih vrsta neuronskih mreža za duboko učenje, od čega su za ovaj rad najbitnije konvolucijske neuronske mreže, stoga će one biti detaljnije predstavljene.

2.1. Konvolucijske neuronske mreže

Konvolucijske neuronske mreže (eng. Convolutional Neural Networks – CNN) vrste su neuronskih mreža koje su specijalizirane za obradu dvodimenzionalnih ulaznih podataka, točnije slika, iako primjenu pronalaze i u radu s jednodimenzionalnim, ali i trodimenzionalnim podatcima. [3] Samo ime ovih mreža potječe iz njihovog specifičnog konvolucijskog sloja, koji je objašnjen u nastavku poglavlja.

Arhitektura konvolucijskih neuronskih mreža inspirirana je organizacijom i funkcijom ljudskog vidnog korteksa [4], odnosno primarnog kortikalnog područja mozga, koje prima, integrira i obrađuje vizualne informacije prenesene s mrežnice oka. [5] Veze između umjetnih neurona (gradivnih jedinica neuronskih mreža) konvolucijskih mreža uvelike podsjećaju na veze između neurona u ljudskome mozgu.



Slika 1. Usporedba prirodnog i umjetnog neurona [6]

Bitna pretpostavka kod rješavanja problema pomoću ovakvih mreža je ta da objekti ne bi smjeli imati prostorno ovisne karakteristike. Primjerice, prilikom detekcije osobe na slici, položaj osobe u odnosu na kadar ne bi smio utjecati na točnost prepoznavanja osobe. Još jedna od karakteristika konvolucijskih mreža učenje je apstraktnih značajki što se dublje ulazni podatci provode kroz mrežu. Na primjer, u klasifikaciji slika u prvim slojevima mreže bit će prepoznati rubovi na slici, u sljedećim slojevima bit će prepoznati jednostavniji oblici, a u najdubljim slojevima prepoznat će se obilježja poput ljudskog oka, mačjeg uha, pseće njuške ili znaka za obavezno zaustavljanje.

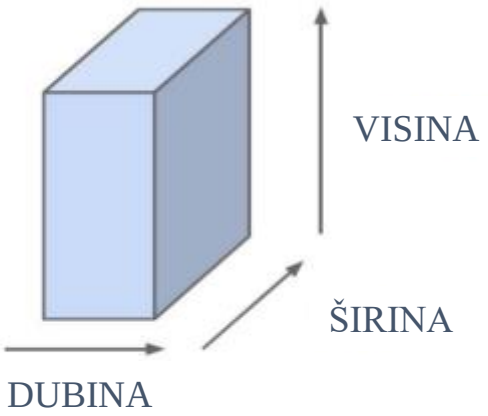
Konvolucijske neuronske mreže zadnjih petnaestak godina postižu revolucionarne rezultate vezane uz prepoznavanje uzoraka; od obrade slika do prepoznavanja glasa. [7] Neke od primjena ovih mreža su klasifikacija podataka, računalni vid, obrada teksta i govora smislenog čovjeku…

2.1.1. Arhitektura konvolucijskih neuronskih mreža

Budući da su CNN prvenstveno specijalizirane za obradu slika, njihova arhitektura u nastavku pobliže je opisana na primjeru obrade slike.

Konvolucijske neuronske mreže prikazuju sliku u tri dimenzije, koje predstavljaju visinu, širinu te dubinu. Visina i širina ovise o broju piksela po visini i širini slike, a dubina definira boje na slikama te najčešće ima tri vrijednosti za RGB sustav ili samo jednu vrijednost za crno-bijele i jednobojne slike. Pikseli su gradivne jedinice računalnih slika, a njihova vrijednost može

varirati u rasponu od 0 do 255, budući da je to raspon svake od boja u RGB sustavu. Pikseli su preko slike raspoređeni kao matrica.



Slika 2. Trodimenzionalna reprezentacija ulaznih slikovnih podataka

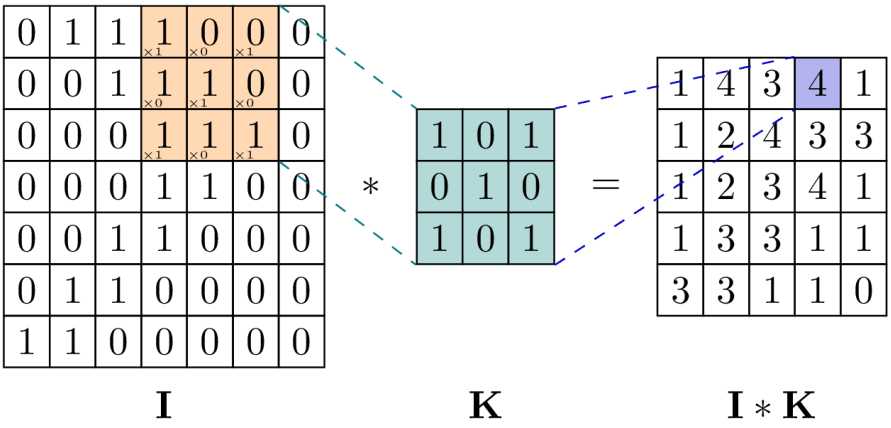
2.1.2. Konvolucijski sloj

Konvolucija je linearna operacija kojom se provodi množenje težinskih faktora s ulaznim podatcima. [3]

Ako je input na primjer slika dimenzija 32x32x3, bez provedbe konvolucije svaki od piksela smatrao bi se ulaznim podatkom, te bi za rad mreže bilo potrebno 3072 (32x32x3=3072) težinska faktora. Treba napomenuti kako bi se ovaj broj odnosio samo na jedan neuron u skrivenom sloju, te ukoliko se doda još jedan neuron, broj potrebnih težina bi se udvostručio, odnosno dodavanjem svakog neurona u skriveni sloj, broj težina bi rastao i postao nepraktičan za izvođenje.

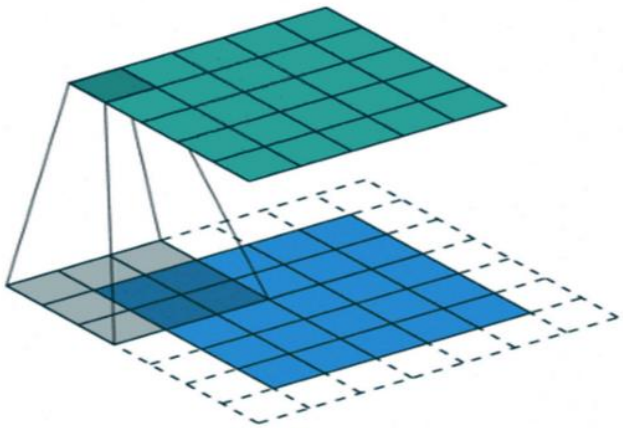
Umjesto analiziranja slike kao cjeline, puno efikasnija metoda pokazalo se promatranje manjih segmenata slike, pri čemu se koristi filter, odnosno kernel. Kerneli sadrže težine koji se uče tijekom procesa treniranja mreže. Dimenzije kernela manje su od dimenzija slike, najčešće 3x3 ili 5x5, a njihova dubina ekvivalentna je dubini slike koja se obrađuje. Konvolucijom koja se provodi između kernela i slike dobije se dot produkt iliti skalarni produkt jer je rezultat samo jedna vrijednost, što se može vidjeti i iz slikovnog prikaza na Slici 3. Cijeli proces odvija se tako što se kernel pomiče po slici počevši od lijevog gornjeg kuta te se potom kreće po širini i visini ulaznih dimenzija slike. Pomicanje kernela ovisi o koraku, odnosno za koliko će se piksela pomaknuti kernel. Ne preskaču li se pikseli prilikom pomaka, korak će biti jednak 1. Svakim pomakom provodi se konvolucija i u konačnici će rezultat konvolucijskog sloja biti dvodimenzionalna tzv. aktivacijska mapa, na Slici 3. označena kao I*K. Ovakav način primjene

kernela preko slike omogućava sistematično traženje i shodno tome pronalaženje određenog obilježja ili objekta bilo gdje na slici.



Slika 3. Konvolucija

Iz Slike 3. [8] može se uočiti i da će zbog manjih dimenzija kernela, i aktivacijska mapa, odnosno izlaz iz konvolucijskog sloja biti manjih dimenzija od originalnih. Ukoliko se žele zadržati originalne dimenzije slike ili čak dobiti dimenzije veće od istih, oko rubnih piksela može se dodati dopuna (eng. padding), kao na Slici 4. [4] Dopuna su zapravo pikseli vrijednosti 0, stoga neće puno utjecati na rezultat konvolucije.



Slika 4. Dopuna

Kako bi se izračunale dimenzije aktivacijske mape, nužno je poznavati sljedeće parametre: **veličinu filtera** (koliko je težina u filteru, odnosno kernelu potrebno trenirati, npr. kernel

dimenzija 3x3 imat će 9 težina), **dubinu slike** (ako je slika u boji, dubina će biti 3, što ekvivalentno znači da će postojati i 3 kernela), **dopunu** te **korak**. Dimenzije aktivacijske mape tada se računaju prema sljedećoj formuli:

$$D = \frac{D_s - F + 2P}{S} + 1 \qquad \textbf{(1)}$$

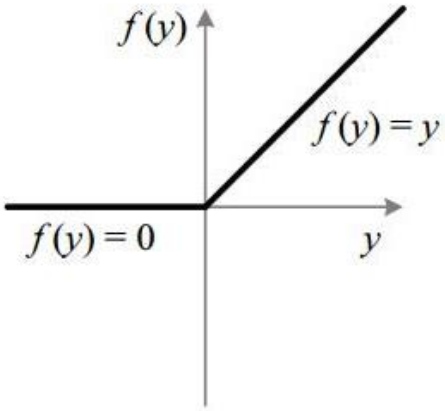
pri čemu je D_s stara dimenzija slike, F dimenzija filtera, P broj nadopunjenih nula, a S korak. Dubina izlazne mape jednaka je broju filtera koji se koriste.

2.1.3. Aktivacijska funkcija

Nakon konvolucijskog sloja uobičajeno slijedi aktivacijska funkcija, korištena za izračunavanje ponderiranog zbroja ulaza i pristranosti (eng. bias) nakon čega se odlučuje može li neuron biti pokrenut ili ne. [9] Drugim riječima, koristi se za kontroliranje izlaza, pretežno radi prilagodbe (podešenja zasićenosti) ili radi isključivanja (ograničavanja) generiranog izlaza. [7] Aktivacijska funkcija može biti linearna ili nelinearna, ovisno o njenoj prirodi, a u modernim metodama dubokog učenja koriste se nelinearne funkcije, koje pružaju bolje mogućnosti učenja. Dugo su najpopularnije u upotrebi bile sigmoidna i tanh funkcija, ali zadnjih godina daleko najkorištenija aktivacijska funkcija je ReLU (Rectified Linear Unit). Neki od razloga njene popularnosti su:

- Definicije funkcije i gradijenta mnogo su jednostavnije.
- Kod funkcija poput sigmoidne i tanh javlja se problem nestajućeg gradijenta, dok ReLU funkcija zadržava konstantni gradijent za pozitivni ulaz.
- ReLU stvara rjeđi prikaz.

ReLU funkcija određena je na sljedeći način: uzme li se da je x ulaz funkcije, ukoliko je x pozitivan broj, utoliko će i izlaz biti jednak x, a ako je x negativan, izlaz će biti 0, što se može i iščitati iz grafa funkcije na Slici 5 (preuzeta s https://www.quora.com/What-is-leaky-ReLU). Upravo na taj način aktivira se manji broj neurona, rjeđi prikaz i mreža je rasterećena. [10]



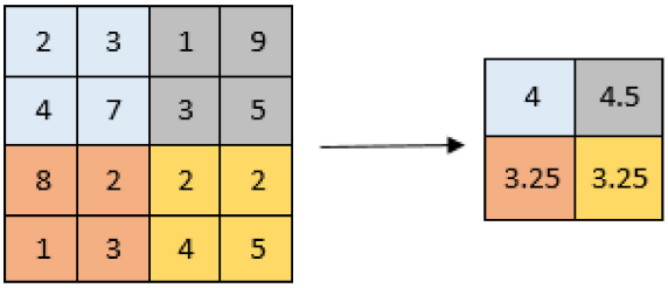
Slika 5. ReLU aktivacijska funkcija

2.1.4. Sloj sažimanja

Zadaća sloja sažimanja (eng. pooling layer) redukcija je prostornih dimenzija podataka, odnosno smanjenje visine i širine ulaza u sljedeći konvolucijski sloj. Važno je napomenuti da pritom veličina dubine ostaje nepromijenjena, dakle i dalje će postojati tri filtera. Redukcija prostornih dimenzija uzrokuje smanjenje potrebne snage računala za obradu podataka (zbog manje količine računanja i potrebnih težina), te doprinosi prepoznavanju vodećih karakteristika neovisno o njihovom položaju i zakrenutosti. Najčešći oblici slojeva sažimanja su sažimanje maksimalnih vrijednosti i prosječno sažimanje.

Prosječno sažimanje:

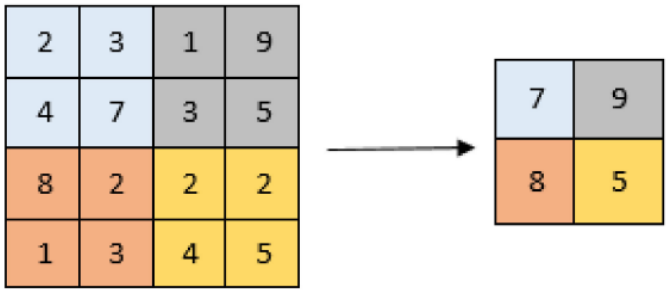
Promotri li se pobliže Slika 6. [4], može se uočiti da se i u slojevima sažimanja javljaju filteri, uz razliku da oni ne nose težine. Ako se filter veličine 2x2 provede po aktivacijskoj mapi 4x4, uz korak 2, rezultat će biti matrica veličine 2x2. Kako se filter pomiče po slici, svakim se pomakom računa prosječna vrijednost svih piksela obuhvaćenih tim filterom te se ta vrijednost zapisuje na ekvivalentno mjesto u novoj matrici.



Slika 6. Prosječno sažimanje

Sažimanje maksimalnih vrijednosti:

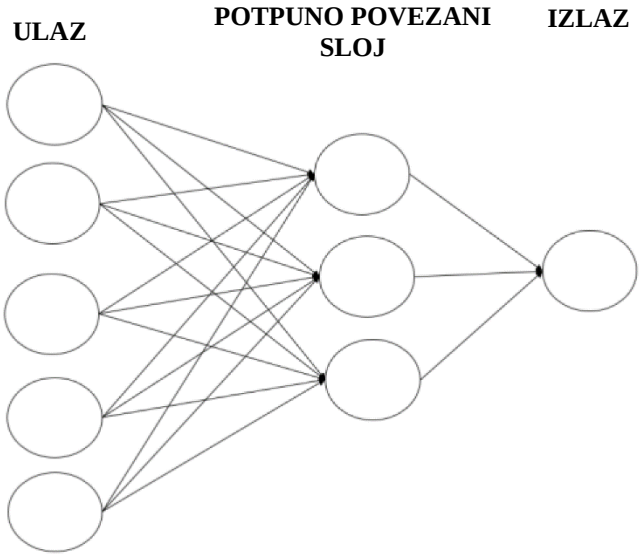
Filter se pomiče po matrici na isti način kao kod prosječnog sažimanja, ali ne računa se prosječna vrijednost, već se jednostavno u novu matricu na odgovarajuće mjesto zapisuje maksimalna vrijednost točaka obuhvaćenih filterom. Ovom metodom dolazi se do brže konvergencije, lakše se prepoznaju superiorna svojstva i poboljšava se generalizacija. [2]



Slika 7. Sažimanje maksimalnih vrijednosti [4]

2.1.5. Potpuno povezani slojevi

Potpuno povezani slojevi (eng. Fully Connected Layer) u neuronskoj mreži pojavljuju se tek u posljednjim slojevima. Svaki od neurona u jednom potpunom povezanom sloju povezan je sa svakim neuronom prethodnog, ali i sljedećeg sloja (Slika 8. (preuzeta s https://www.mdpi.com/2073-8994/14/4/658)). Ovaj sloj prima input od sloja sažimanja ili konvolucijskog sloja te ga "ravna" u jednodimenzionalni izlaz, vektor. Jedan od nedostataka potpuno povezanih slojeva vrlo je velik broj parametara koji zahtijevaju složeno računanje tijekom treniranja mreže. Stoga se pomoću tehnike ispadanja mogu reducirati brojevi čvorova i veza.



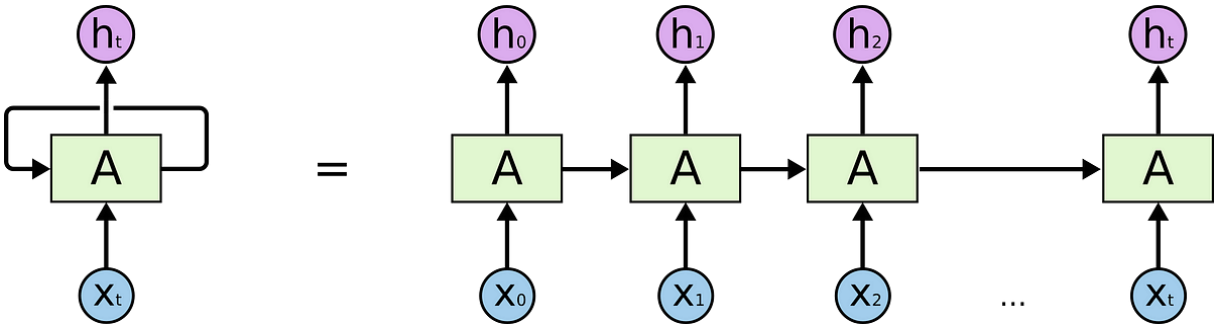
Slika 8. Potpuno povezni sloj

Prilikom rješavanja problema klasifikacije, u posljednjem je sloju uobičajena primjena Softmax klasifikacijske funkcije. Ona transformira vektor izlaza posljednjeg sloja mreže u vjerojatnosti, čineći ih tako pogodnijima za klasifikaciju.

2.2. Rekurzivna ili ponavljajuća neuronska mreža

Rekurzivne ili ponavljajuće neuronske mreže (eng. Recurrent Neural Networks – RNN) vrsta su umjetnih neuronskih mreža prilagođenih za obradu sekvencijalnih podataka i rad s vremenskim serijama podataka. Ime su dobile po tome što izvode isti zadatak za svaki element sekvence, pri čemu izlazni podatci ovise o prethodnim kalkulacijama. Drugim riječima, RNN posjeduju koncept pamćenja koje im pomaže pohraniti informacije prethodnih ulaznih podataka kako bi generirale sljedeći izlaz u sekvenci. Teoretski bi RNN mogle koristiti podatke kroz relativno duge sekvence, ali u praksi se pokazalo da koriste podatke od samo nekoliko prethodnih koraka, sve dok nije izumljena dugotrajna kratkoročna memorija (eng. Long Short Term Memory – LSTM). Temeljna ideja iza arhitekture ove najkorištenije RNN, memorijska je ćelija koja može održavati svoje stanje tijekom vremena te nelinearne jedinice koje reguliraju protok informacija kroz ćelije. [1] Prednosti RNN leže u sposobnosti rukovanja sekvencijskim podatcima, unosima varirajućih duljina te sposobnosti pohrane informacija. S druge strane, provedba može biti vrlo spora, te mreža ne uzima u obzir buduće unose pri donošenju odluka.

Nadalje, još jedan nedostatak bio bi nestajući gradijent, odnosno gradijenti za računanje težinskih faktora približavaju se nuli, što mreži onemogućuje učenje novih težinskih faktora. [11] Grafički prikaz RNN mreže prikazan je na Slici 9 (preuzeta s https://aditi-mittal.medium.com/understanding-rnn-and-lstm-f7cdf6dfc14e).



Slika 9. RNN model

2.3. Ograničeni Boltzmannov stroj

Ograničeni Boltzmannov stroj (eng. Restricted Boltzmann Machine – RBM) predstavljen je osamdesetih godina prošlog stoljeća, a danas pronalazi svoju primjenu u klasifikaciji slika, obradi medicinskih fotografija… RBM je svojom strukturom plitka neuronska mreža sa samo dva sloja – ulazni sloj i skriveni sloj. Neuroni u pojedinom sloju ove mreže nemaju međusobnih veza, ali su povezani sa svim neuronima spomenutog drugog sloja. Informacije u ovoj mreži tijekom treniranja i korištenja teku u oba smjera pri čemu su i težinski faktori u oba smjera jednaki. Tijekom prolaza unaprijed RBM ulazne podatke prevodi u set brojeva, koje tijekom prolaza unazad prevodi natrag generirajući tako restrukturirane ulazne podatke. Zanimljiva karakteristika ovih mreža je ta da podatci ne trebaju imati pripadajuće oznake, što je važno u primjeni na stvarnim setovima podataka. [1]

RBM je dobio na popularnosti kada je postao gradivna jedinica za višeslojnu strukturu učenja Deep Belief Network (DBN).

2.4. Autoenkoderi

Autoenkoderi vrsta su neuronskih mreža koje su prvenstveno konstruirane za komprimiranje ulaznih podataka, koji se potom dekodiraju natrag na način da rekonstruirani ulaz bude što sličniji originalnom. Zbog toga mreža mora odlučiti koje su značajke ulaznih podataka

najbitnije. Ulazni sloj i izlazni sloj autoenkodera su jednaki, te autoenkoderi mogu biti trenirani s kraja na kraj (eng. end-to-end) ili postupno sloj po sloj, u kojem se slučaju njihovi slojevi slažu u dublje višeslojne neuronske mreže. Duboki autoenkoderi su posebno korisni u smanjenju dimenzionalnosti, a manje dimenzije ulaznih podataka vode do velikih računalnih ubrzanja. [1] Glavna svrha autoenkodera nenadzirano je učenje informativnog prikaza podataka koji ima razne primjene, poput na primjer, klasterizacija. Osim za klasterizaciju koriste se i pri klasifikaciji, za detekciju anomalija, smanjenje dimenzija… [12]

3. RAČUNALNI VID

Računalni vid dio je umjetne inteligencije koji uključuje korištenje računala u svrhu dobivanja detaljnog razumijevanja vizualnih podataka, pritom rabeći metode slične ljudskim prirodnim vizualnim sustavima. [13] To je izuzetno raznoliko polje koje pruža puno potencijala, stoga ne čudi činjenica da se u današnje vrijeme u tom području izvodi velik broj istraživanja.

Prije razvoja dubokog učenja, računalni vid imao je velika ograničenja te je zahtijevao veliku količinu ručnog kodiranja. Nije bilo implementirano mnogo automatizacije, te su greške u rezultatima bile velike. Zahvaljujući napretku umjetne inteligencije, posebice inovacijama na području dubokog učenja i neuronskih mreža, zadnjih godina računalni vid prati velik razvitak te pronalazi svoje primjene u brojnim područjima. Neki od glavnih čimbenika koji stoje iza ovog rasta količina je podataka koji se mogu generirati te potom koristiti za treniranje i unaprjeđenje sustava računalnog vida, ali i snaga današnjih računala kojom je omogućena brza obrada podataka.

Strojno učenje uvelike je pomoglo rješavanju brojnih povijesnih problema. Na primjer, razvijen je softver za predviđanje vjerojatnosti prebolijevanja raka dojke, uspješniji od stručnjaka iz tog područja medicine. Ipak, za razvoj takvog softvera bilo je uloženo mnogo rada od strane inženjera i medicinskih stručnjaka. Drugačiji pristup strojnom učenju javio se s razvojem dubokog učenja i neuronskih mreža koje ono koristi. Trenira li se mreža na dovoljnoj količini označenih primjera iz nekog specifičnog područja, mreža će sama biti sposobna prepoznati obilježja koja povezuju slike istih oznaka. Te ponavljajuće uzorke transformirat će u matematički zapis koji će koristiti za buduću klasifikaciju.

3.1. Primjena računalnog vida

Primjena računalnog vida može biti prepoznata u brojnim proizvodima koje ljudi koriste svakodnevno. Primjerice, prilikom prepoznavanja lica računalni vid prepoznaje karakteristike lica koje tada uspoređuje s onima u bazi podataka na kojoj je mreža trenirana. Ovakva metoda koristi se kod pametnih telefona za identifikaciju vlasnika uređaja, u aplikacijama društvenih mreža kako bi se prepoznali i označili korisnici na fotografijama, a također prepoznavanje lica pronalazi primjenu i u identifikaciji zločinaca snimljenih na videu.

Jedna od bitnijih primjena svakako je i zdravstvo. Na primjer, računalni vid može biti izuzetno koristan pri detekciji kancerogenih madeža na koži ili dijagnostika simptoma prepoznatih na rendgenskim ili MR skenovima.

Velika uloga računalnog vida može se pronaći i u proširenoj i mješovitoj stvarnosti, koja omogućuje pametnim uređajima (pametni telefoni, tableti, pametne naočale) da u sliku stvarnog svijeta ubace virtualne objekte. Pomoću računalnog vida, oprema za proširenu stvarnost otkriva objekte u stvarnom svijetu kako bi odredila lokaciju na zaslonu gdje će biti postavljen virtualni objekt. [14]

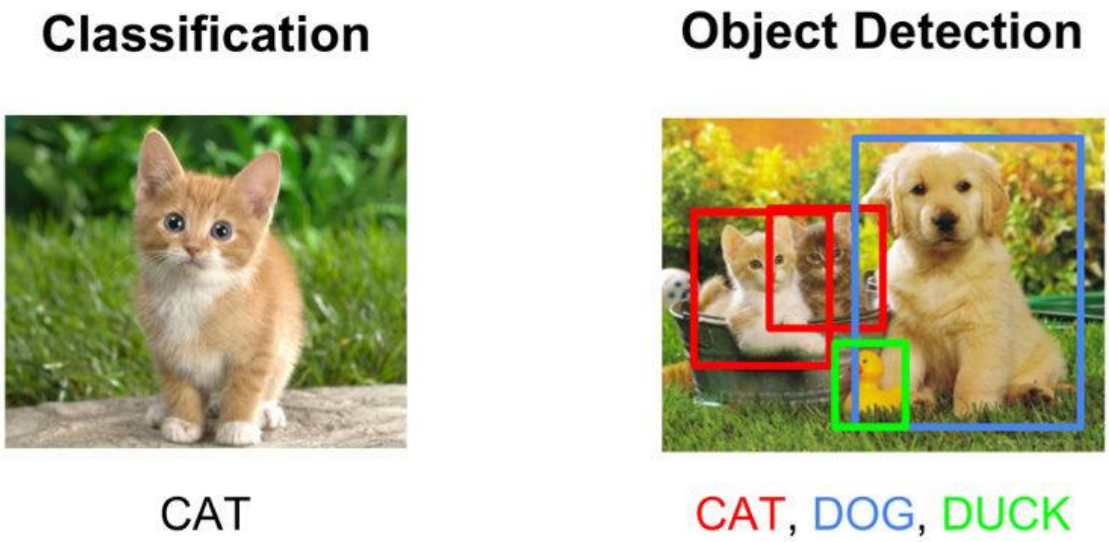
Može se reći da su glavna polja na koja se dijeli računalni vid: prepoznavanje objekata, detekcija objekata, video praćenje, segmentacija objekata, procjena pozicije i pokreta, modeliranje scene te restauracija slike. [15] Budući da se ovaj rad temelji na prepoznavanju i praćenju objekata, u nastavku slijedi opis tehnika koje se koriste za obavljanje tih zadataka.

4. PREPOZNAVANJE OBJEKATA

Prepoznavanje objekata vrlo je važno polje unutar domene računalnog vida. Ono se odnosi na proces detekcije instanci (poput, na primjer, ptice, automobila, konja ili osobe) na slikama ili videima. U početcima razvoja prepoznavanja objekata korišteni su i bili su vrlo popularni dvofazni detektori objekata. No, razvojem jednofaznih detektora i algoritama koje oni koriste, premašili su popularnost dvofaznih detektora. Štoviše, usponom YOLO algoritama za prepoznavanje objekata implementacija jednofaznih detektora daleko je premašila primjene dvofaznih, a njihov je rad pružao izuzetne rezultate. [15]

Radi kvalitetnijeg razumijevanja prepoznavanja objekata, prvo treba objasniti pojmove klasifikacija objekata (eng. Object Classification), lokalizacija (eng. Object Localization) te detekcija objekata (eng. Object Detection).

Klasifikacija slika odnosi se na dodjelu oznake klase pojedinoj slici na temelju prepoznatog objekta, dok lokalizacija uključuje crtanje graničnih okvira oko jednog ili više objekata na slici. Drugim riječima, ulazni podatak za klasifikaciju bit će slika s jednim objektom na njoj, a izlaz će biti oznaka klase. Ulaz za lokalizaciju bit će slika s jednim ili više objekata, a izlaz će biti jedan ili više graničnih okvira. Detekcija objekata kombinira ova dva zadatka. Nakon što je na slici prepoznat objekt, određuje se njegova točna lokacija na slici crtanjem graničnog okvira oko objekta. Skupni je naziv za sve ove radnje zajedno prepoznavanje objekata. [16]



Slika 10. Razlika između klasifikacije i detekcije [17]

Slika 11. Razlika između lokalizacije i detekcije [18]

Detektori objekata mogu se podijeliti na jednofazne i dvofazne, a njihova je glavna razlika u tome hoće li se generirati prijedlog interesnih područja ili ne.

4.1. Izazovi prilikom prepoznavanja objekata

Primjene prepoznavanja objekata su brojne; samovozeći automobil, prepoznavanje teksta, nadzorni sustavi, robotika, prepoznavanje lica, prepoznavanje pješaka, i mnoge druge. Iako su primjene već ovako raznovrsne, sustavi za prepoznavanje objekata u svom radu nailaze i na brojne izazove. Glavni izazovi uključuju:

- **Višerazmjerno treniranje**: većina detektora objekata trenirani su na određenoj rezoluciji ulaza te proizvode lošije rezultate ako su ulazne slike drugačije rezolucije ili drugačijih dimenzija od onih na kojima se provodio trening mreže.
- **Disbalans klasa prednjeg i pozadinskog plana**: neravnoteža i disproporcije među objektima različitih kategorija mogu značajno utjecati na izvedbu.
- **Detekcija relativno malih objekata**: svi algoritmi za pronalaženje objekata dobre će rezultate pokazivati za veće objekte, budući da je i mreža trenirana na većim modelima. Međutim, takvi modeli rezultiraju lošijom izvedbom u primjeni na objektima manjih dimenzija.
- **Potreba za velikim skupovima podataka i snagom računala**: algoritmi detektora objekata u dubokom učenju trebaju veće skupove podataka za računanje i velike snage računala za obradu podataka. Budući da su brojevi podataka za trening zadnjih godina

eksponencijalno porasli, označavati svaki pojedini objekt na slikovnom prikazu postao je vrlo zamoran zadatak.

- **Manje veličine skupova podataka**: usprkos činjenici da modeli dubokog učenja mogu nadigrati tradicionalne metode strojnog učenja, pokazalo se da su im za to potrebni veliki skupovi podataka te da učenje na manjem skupu podataka rezultira lošijom izvedbom modela.
- **Netočna lokacija prilikom predviđanja**: granični okviri (eng. bounding box) predstavljaju predviđanja temeljne istine, odnosno stvarne klase objekta na slici. Tijekom predviđanja uključeni su i okolni pikseli, što utječe na točnost algoritma. [15]

4.2. Tradicionalne metode detekcije objekata

Tradicionalne tehnike prepoznavanja objekata obično su se sastojale od tri faze. U prvoj fazi označena su potencijalna područja na slici i lociran je objekt. Za tu fazu koristio se klizni prozor koji se pomicao po cijeloj slici. Druga faza odnosila se na izdvajanje značajki iz generiranih potencijalnih područja. Glavni alati za pronalaženje značajki bili su SIFT (Scale Invariant Feature Transform) i HOG (Histogram of Oriented Gradients). Treća faza bila je primjena treniranog klasifikatora koji u konačnici provodi klasifikaciju. Glavni su klasifikatori bili SVM (Support Vector Machine), AdaBoost (Adaptive Boosting) itd. Tradicionalni detektori objekata uključuju HOG + Cascade algoritam za otkrivanje, HOG + SVM algoritam i DPM algoritam.

Usporedba prva dva navedena tradicionalna detektora vidljiva je u Tablici 1:

Metoda	Kvaliteta detekcije	Vrijeme detekcije
HOG + Cascade	Lošija kvaliteta (slaba sposobnost generalizacije rezultira niskom stopom detekcije)	Kratko (u prosjeku 50 ms, za primjene u stvarnom vremenu)
HOG + SVM	Dobra kvaliteta (niska stopa pogreške)	Dugo (u prosjeku 500 ms, nije primjenjivo u stvarnom vremenu)

Tablica 1. Usporedba tradicionalnih metoda detekcije

DPM algoritam robusniji je za promjene položaja pješaka, što znači da ima veću točnost detekcije u kompleksnim scenama. Međutim, ovaj algoritam je vrlo spor tako da daje slabe rezultate u stvarnom vremenu. Zbog disbalansa ovih metoda u točnosti i vremenu detekcije, javila se potreba za razvojem dvofaznih modela.

4.3. Dvofazni detektori objekata

Dvofazni detektori objekata obavljaju zadatak prepoznavanja objekata u dvije faze. Prvom fazom generiraju se interesna područja (eng. Regions of Interest – RoI) pomoću mreže za predlaganje područja (eng. Region Proposal Network – RPN). To znači da se u ovoj fazi odabiru područja u kojima je moguće da će biti prepoznat objekt. Druga faza bit će odgovorna za predviđanje objekata i stvaranje graničnih okvira za predložena interesna područja. Najpoznatiji su modeli u ovoj kategoriji Konvolucijske neuronske mreže temeljene na područjima (eng. Region Based Convolutional Neural Networks – RCNN), Brze RCNN (eng. Fast RCNN) i Brže RCNN (eng. Faster RCNN).

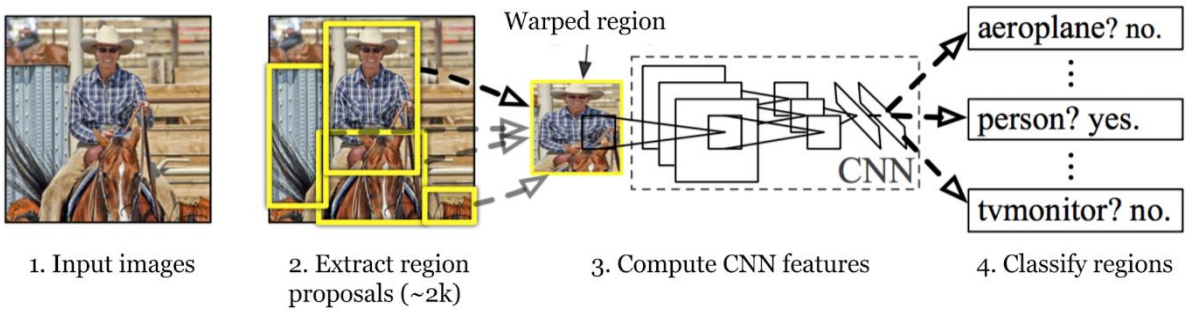
4.3.1. R-CNN

R-CNN konvolucijska je neuronska mreža temeljena na područjima, a osmislio ju je Ross Girshick 2013. godine. [19] Predložio je rješenje problema stvaranja velikog broja potencijalnih područja time što bi se selektivnim traženjem generiralo 2000 područja na slici koja imaju najveću vjerojatnost da će sadržavati objekt, što bi uvelike smanjio posao klasifikacije. [20] Ovih 2000 područja provlače se kroz neuronsku mrežu koja obavlja zadatak ekstrakcije obilježja sa slike koja onda prolaze kroz SVM klasifikator ne bi li se klasificirao objekt unutar potencijalnog područja. Uz predviđanje postojanja objekta unutar potencijalnog područja, algoritam pomaže i povećanju preciznosti graničnih okvira.

Rad R-CNN mreže može biti sažet na sljedeći način:

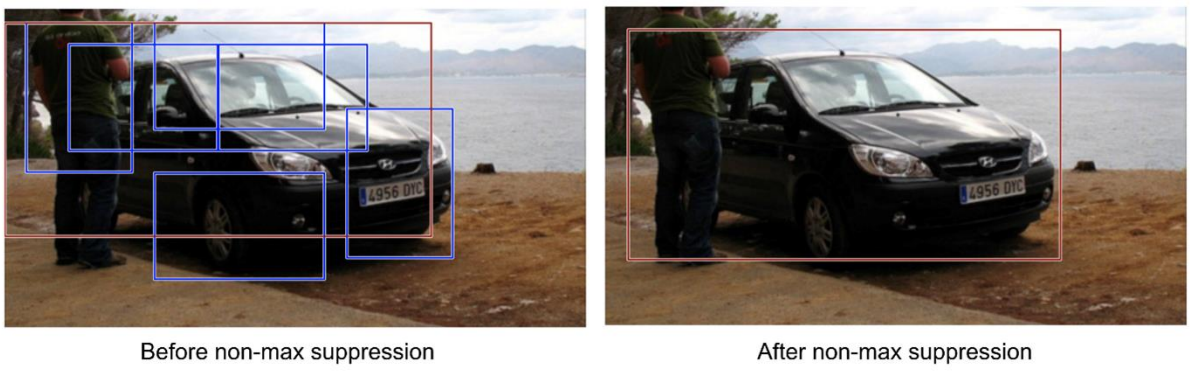
1. Prethodno treniranje konvolucijske mreže na slikovnom setu podataka s K brojem klasa.
2. Prijedlog 2000 potencijalnih područja slike koje mogu sadržavati tražene objekte. Područja ne moraju biti jednakih veličina.
3. Nastavak podešavanja konvolucijske mreže na predloženim područjima za K + 1 klasa (jedna klasa odnosi se na pozadinu, odnosno na nepostojanje traženog objekta).

4. Za svako područje jednim unaprijednim prolaskom kroz mrežu generira se vektor koji potom prolazi kroz SVM klasifikator.
5. Kako bi se smanjile pogreške lokalizacije, ispravljaju se predviđanja graničnih okvira.



Slika 12. Arhitektura R-CNN [19]

Kod R-CNN mreža i sličnih modela pronalaženja objekata koristi se ne-maksimalno potiskivanje (Non-Maximum Suppression) kojim se izbjegava ponovljena detekcija jedne te iste instance. Nakon što se generira skup graničnih okvira za objekt u istoj kategoriji, zanemaruju se okviri s niskom razinom sigurnosti te se bira onaj okvir s najvećom sigurnošću. Primjer za ovu pojavu prikazan je na Slici 13. [21]



Slika 13. Ne-maksimalno potiskivanje

Još jedan koristan alat R-CNN mreža svakako je isključivanje teških negativa (eng. Hard Negative Mining). Ako unutar graničnog okvira ne postoji objekt, takav okvir nazivamo negativnim. Ako je unutar takvog okvira potpuno prazna pozadina, to se može nazvati „lakim negativom“. Međutim sadržava li takav okvir nejasne teksture ili malen dio objekta, negativ može biti teško prepoznati, odakle i dolazi naziv „teški negativ“. Njih može biti lako pogrešno

klasificirati kao pozitive, stoga se oni trebaju eksplicitno izdvojiti tijekom treniranja mreže i uvrstiti ih u podatke za treniranje kako bi se unaprijedio rad klasifikatora. [21]

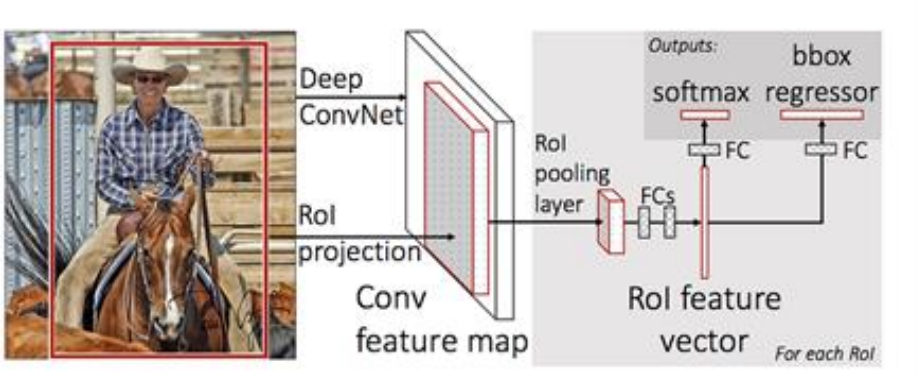
Neki od problema koji se javljaju u radu R-CNN mreže su:

- Potrebno je mnogo vremena za treniranje mreže jer je potrebno klasificirati 2000 prijedloga područja po slici.
- Ne može se implementirati u detekciju u stvarnom vremenu.
- Algoritam selektivnog pretraživanja je fiksan, odnosno u toj fazi se ne odvija nikakvo učenje mreže što može dovesti do velikog broja loše predloženih područja. [22]

4.3.2. Brza R-CNN

Ross Girshick bio je svjestan nedostataka u radu R-CNN modela, stoga je 2015. predložio novi model nazvan Brza R-CNNN (eng. Fast R-CNN). U ovome modelu, slika je prva koja se provlači kroz konvolucijsku mrežu, a iz posljednje mape značajki generiraju se prijedlozi područja. Ulazna slika i interesna područja (RoI) ulazi su u konvolucijsku mrežu. Svako interesno područje sažima se u mapu značajki fiksne veličine nakon čega se potpuno povezanim slojevima pretvara u vektor obilježja. Mreža za svako interesno područje ima dva izlaza; softmax vjerojatnosti za predviđanje klase te vrijednosti pomaka graničnih okvira.

Brza R-CNN brža je od R-CNN jer se ne treba procesirati 2000 regija, već se konvolucija provodi samo jednom za svaku sliku iz čega se kao rezultat dobije mapa značajki. Ovim poboljšanjem postižu se brža vremena treniranja modela i vremena donošenja zaključaka. [22]



Slika 14. Arhitektura brze R-CNN [23]

Jedna od inovacija u ovom modelu sloj je sažimanja nazvan RoI sažimanje (eng. RoI Pooling). To je vrsta sažimanja maksimalnih vrijednosti korištena za pretvorbu značajki iz jednog

interesnog područja sa slike u mapu značajki s fiksnim dimenzijama V x Š, pri čemu su V i Š hiperparametri ovisni o svakom pojedinom području interesa. Svaki „prozor“ interesnog područja veličine v x š dijeli se na V x Š mrežu podprozora veličina v/V x š/Š nakon čega se provodi maksimalno sažimanje svakog podprozora. [23]

Tijek rada ovog modela je kako slijedi [21]:

1. Predtreniranje konvolucijske mreže.
2. Prijedlog regija selektivnim traženjem.
3. Izmjene predtrenirane konvolucijske mreže:
    a. Zamjena posljednjeg sloja sažimanja maksimalnih vrijednosti s RoI slojem sažimanja.
    b. Zamjena posljednjeg potpuno povezanog sloja i posljednjeg softmax sloja za K broj klasa, s potpuno povezanim slojem i softmax funkcijom za K + 1 klasa.
4. Model se grana u dva izlazna sloja:
    a. Softmax predviđanja za K + 1 klasa.
    b. Vrijednosti pomaka graničnih okvira.

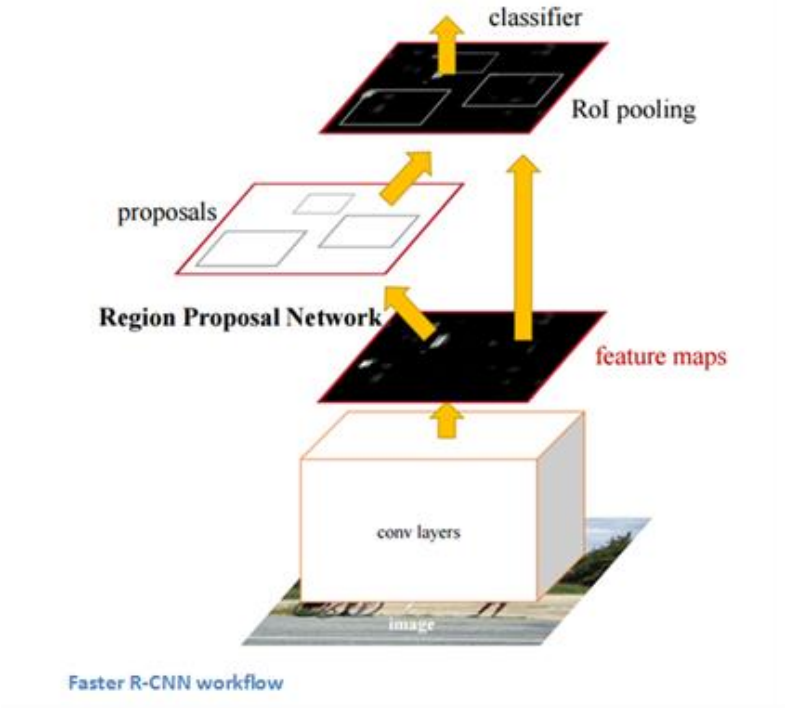
4.3.3. Brža R-CNN

Prethodno navedeni modeli (R-CNN i Brza R-CNN) za prijedloge područja koriste selektivno traženje koje je sporo i vremenski skupo. Zbog toga je Shaoqing Ren [24] osmislio algoritam za detekciju objekata koji se ne oslanja na selektivno traženje, već koristi mrežu za predlaganje područja (RPN) kojom mreža uči kako predložiti područja. Takav algoritam naziva se Brža R-CNN (eng. Faster R-CNN). [22]

Struktura Brže R-CNN sastoji se od sljedećih komponenata:

1. Konvolucijski sloj kojim se dobiva mapa značajki za cijelu sliku.
2. RPN generira težine (eng. anchors) prilikom stvaranja prijedloga područja. Funkcija prosudbe određuje jesu li težine u prednjem planu ili su u pozadini, te ih podešava kako bi se dobio što točniji prijedlog područja.
3. RoI sažimanje.
4. Sloj klasifikacije za predviđanje klase objekta i sloj regresije za podešavanje interesnih područja. [24]

Na Slici 15. [16] prikazana je i shema arhitekture ovog modela.



Slika 15. Arhitektura brže R-CNN

U Tablici 2. ispisane su prosječne testne brzine svake od obrađenih metoda detekcije objekata R-CNN algoritmima. Vrlo se jasno može iščitati kako je Brža R-CNN znatno brža od svojih prethodnika. Toliko kratko vrijeme izvođenja čini ju prikladnom za primjenu u stvarnom vremenu.

	R-CNN	**Brza R-CNN**	**Brža R-CNN**
Vrijeme testiranja po slici	50 s	2 s	0,2 s
Ubrzanje	1x	25x	250x

Tablica 2. Usporedba R-CNN metoda

4.3.4. R-FCN

Proces korišten u prethodnim algoritmima, koji uključuje RPN, RoI sažimanje, potpuno povezane slojeve, vremenski je skup, a potpuno povezani slojevi povećavaju broj parametara za učenje što dodatno pridonosi kompleksnosti mreže.

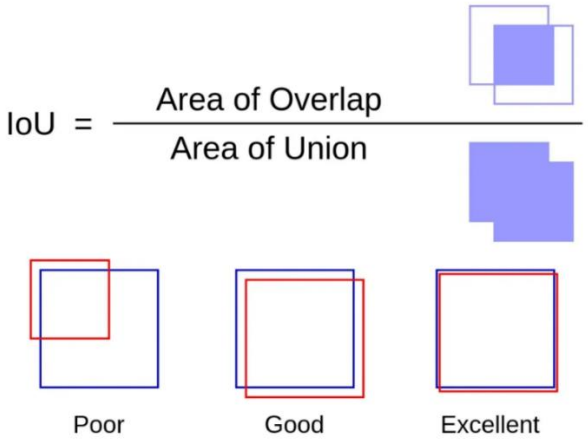
Kod modela Potpuno konvolucijske mreže temeljene na područjima (eng. Region-based Fully Convolutional Network – R-FCN) i dalje postoji RPN za prijedlog područja, ali ne postoje više

potpuno povezani slojevi nakon RoI sažimanja. Svi kompleksni slojevi, koji nose ulogu učenja u mreži, prebačeni su prije RoI sažimanja, što znači da nakon sažimanja više nema slojeva koji moraju učiti, odnosno slojevi mogu međusobno razmjenjivati informacije o naučenim parametrima. [25] Položaj RoI sloja u mreži uvelike utječe na rad mreže, pa tako ako je RoI sloj bliži inputu, na kraju će algoritam raditi s većom točnošću prepoznavanja objekata, ali radit će sporije. Što je RoI sloj bliži izlazu mreže, to će njegova podmreža biti plića i efikasnija, ali može se dogoditi da mreža daje rezultate manje točnosti. Budući da je u R-FCN mreži RoI sloj prebačen na sam kraj mreže, konvolucijski slojevi mogu međusobno izmjenjivati parametre. Posljedično, R-FCN još je brža i efikasnija od Brže R-CNN s pogodnom točnošću obavljanja zadataka.

No ovakav položaj RoI sloja u mreži može uzrokovati i neosjetljivost mreže na lokaciju objekata. Zbog toga R-FCN primjenjuje mape rezultata osjetljive na položaj (eng. position-sensitive score maps), koje sadrže informacije o lokaciji. Kako bi se to ostvarilo, u mrežu je ubačen poseban konvolucijski sloj nakon svih ostalih konvolucijskih slojeva, odgovoran za generiranje mapa rezultata osjetljivih na položaj. Na koncu mreža izvršava RoI sažimanje osjetljivo na položaj (eng. Position-Sensitive RoI pooling), koje nije isto kao RoI sažimanje u Bržoj R-CNN mreži. [26]

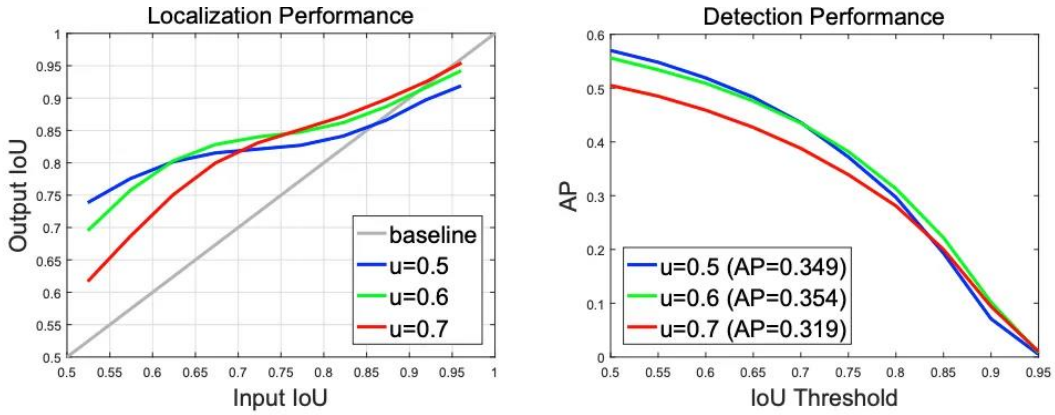
4.3.5. Kaskadna R-CNN

U prepoznavanju objekata nužno je postojanje presjeka iznad unije (eng. Intersection over Union – IoU) kako bi se mogli definirati pozitivi i negativi. Ova operacija prikazana je na Slici 16. (preuzeta s https://idiotdeveloper.com/what-is-intersection-over-union-iou/), na kojoj je i objašnjena razlika između loše, dobre i izvrsne operacije IoU.



Slika 16. IoU

Detektor koje treniran s niskim pragom IoU, na primjer 0,5, uobičajeno generira detekcije uz pozadinsku buku (eng. noise). S druge strane, poveća li se prag IoU, degradirat će se kvaliteta izvedbe detekcije, što je prikazano i na grafovima na Slici 17 [27].



Slika 17. Izvedba regresora (lijevo) i detektora (desno) u ovisnosti o pragu IoU

Za ovu pojavu odgovorna su dva ključna faktora:

- Problem prenaučenosti (eng. overfitting) mreže u fazi treniranja.
- Neusklađenost vremena zaključivanja koji je IoU optimalan za rad detektora.

Kaskadna R-CNN preložena je kao rješenje ovih problema. Sastoji se od više detektora treniranih s povećavajućim pragovima IoU, što omogućuje uspješniju eliminaciju lažnih pozitiva. Detektori su trenirani fazu po fazu, pri čemu je izlaz iz jednog modela detekcije ujedno i ulaz u sljedeći model, a prag IoU postupno raste. [27] Sukladno tome, modeli koji se pojavljuju kasnije u mreži imat će više tragove IoU. Ovakva kaskadna struktura postoji iz dva razloga:

1. Prag IoU odabran za treniranje modela za pronalaženje objekata trebao bi biti što bliži IoU vrijednosti prijedloga unesenog u ovaj model, u svrhu poboljšanja učinkovitosti detekcije.
2. Za različite pragove izlazni IoU generalno će biti veći od ulaznog IoU. Zbog toga se izlaz prethodne faze IoU koristi kao ulaz u sljedeću fazu, tako da dobiveni IoU postaje sve veći. Kaskadna R-CNN čini da se detektor na svakom stupnju usredotoči na detekciju vrijednosti IoU unutar zadanog raspona, čime se poboljšava kvaliteta detekcije. [26]

4.3.6. Usporedba

Proveden je eksperiment detekcije brodova iz skupa podataka HSRC2016 koji sadrži 256 slika za trening i 98 slika za validaciju. [26] U eksperimentu je testirano pet različitih modela brodova. Skup podataka treniran je na sljedećim modelima: Brža R-CNN, R-FCN, FPN, Kaskadna R-CNN, a u konačnici je provedena i validacija. Sposobnost pronalaženja objekata pojedinog modela izražena je preko srednje prosječne preciznosti (eng. mean average precision – mAP). Što je veća mAP, to model ima bolju izvedbu prilikom detekcije objekata na zadanom skupu podataka. Iz Tablice 3. može se iščitati kako je najlošije rezultate u ovom eksperimentu postizala Brža R-CNN, a najbolji rezultati proizlaze iz Kaskadne R-CNN, koja je od najlošije bolja za 1,69%.

model	**mAP**
Brža R-CNN	93,63%
R-FCN	94,20%
FPN	94,72%
Kaskadna R-CNN	95,32%

Tablica 3. Usporedba dvofaznih modela detekcije objekata

4.4. Jednofazni detektori objekata

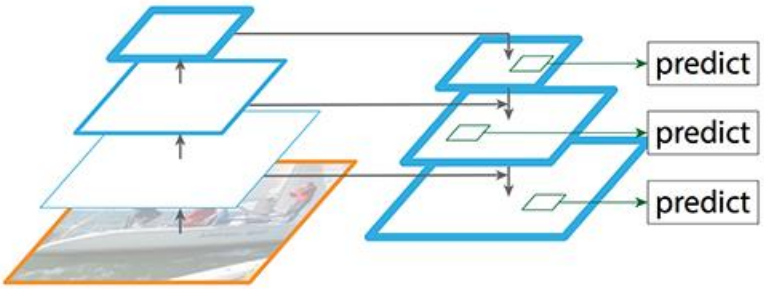
Glavna razlika koja postoji između jednofaznih i dvofaznih detektora objekata je u tome što jednofazni ne generiraju prijedlog područja, nego direktno mogu provesti klasifikaciju i koordinate položaja objekta.

Jednofazni detektori imaju jednostavniju arhitekturu. Uzimaju u obzir sve prostorne veličine na slici, a kao izlaz generiraju granične okvire oko pronađenog objekta, kao i njihovu klasu. Dvofazni detektori vrlo su kompleksne arhitekture, ali su i moćniji, tako da generalno daju bolje rezultate od jednofaznih detektora. Generalno se može reći da jednofazni detektori imaju prednost u pogledu brzine, a dvofazni imaju prednost u pogledu točnosti. Međutim, s pojavom i razvojem YOLO (You Only Look Once) algoritma za pronalaženje objekata i jednofazni detektori u današnje vrijeme mogu parirati dvofaznima. U ovim algoritmima zadatak se obavlja u jednoj fazi/kadru pri čemu je problem lokalizacije oblikovan kao problem regresije uz korištenje dubokih neuronskih mreža. YOLO nije prvi algoritam koji je koristio detektor jednog

kadra (eng. Single Shot Detector – SSD), nego je on bio implementiran i u njegove prethodnike kao što su Dekonvolucijski detektor jednog kadra (eng. Deconvolution Single Shot Detector – DSSD), RetinaNet, RefineDet++… [15]

4.4.1. FPN

Iako nije detektor objekata, za razumijevanje nastavka ovog rada, bitno je opisati Piramidalnu mrežu značajki (eng. Feature Pyramid Network – FPN), koja se koristi za ekstrakciju obilježja različitih dimenzija na slici. [26] FPN ima arhitekturu odozgo prema dolje s bočnim vezama, koja je predočena i slikovnim prikazom na Slici 18 (preuzeta s https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c). Plavi obrisi na prikazu označavaju mape značajki, a deblji obrisi semantički jača obilježja. [28] Razvijena je za generiranje visokokvalitetnih mapa značajka te pokazuje značajna poboljšanja kao ekstraktor značajki. Ovim piramidama mjerilo se ne mijenja, odnosno promjena mjerila objekta kompenzira se njegovim premještanjem na neku drugu razinu u piramidi, čime je omogućeno otkrivanje objekata u velikom rasponu mjerila.



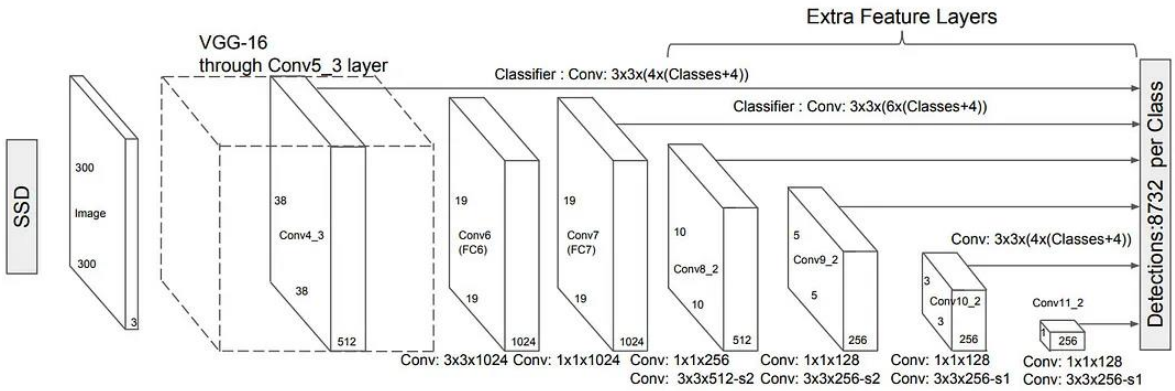
Slika 18. FPN model

Glavna pogodnost određivanja značajki na svakoj razini piramide je ta što se proizvodi prikaz značajki različitih mjerila pri čemu su sve razine jake, uključujući razine visokih rezolucija. S druge strane, ovakav pristup ima i svoja ograničenja. Značajno je povećano vrijeme zaključivanja, zbog čega je onemogućena primjena u stvarnom vremenu. Uz to, učenje dubokih neuronskih mreža na ovaj način nije izvedivo u kontekstu memorije, stoga su slikovne piramide korištene samo tijekom testiranja. Zbog toga dolazi do nedosljednosti između vremena treniranja i vremena testiranja. [29]

4.4.2. SSD

Korištenjem SSD metode relativno se pojednostavljuje proces detekcije objekata jer je eliminiran korak generiranja prijedloga područja i naknadne faze ponovnog uzorkovanja značajki. Svi proračuni obavljeni su u jednoj mreži. Upravo je zbog toga SSD jednostavan za treniranje i izravan u kontekstu integracije u sustave koji sadrže komponentu detekcije objekata. SSD diskretizira granične okvire u skup zadanih okvira preko različitih omjera i mjerila po lokaciji mape značajki. Tijekom vremena predviđanja, mreža generira rezultate koji se odnose na postojanje svake kategorije objekta koja se ispituje unutar svakog pojedinog zadanog okvira. Na temelju toga proizvodi prilagodbe okvira kako bi on što bolje odgovarao obliku detektiranog objekta. Uz to, mreža kombinira predviđanja iz nekoliko mapa značajki različitih rezolucija kako bi prirodnije obrađivala objekte različitih veličina. [30]

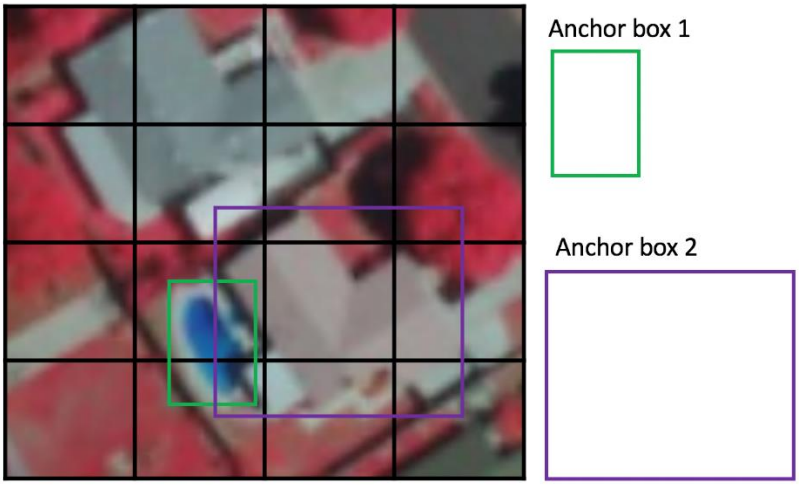
SSD se sastoji od dva dijela. Prvi dio služi za ekstrakciju mapa značajki i uobičajeno ga čini prethodno trenirana neuronska mreža, poput na primjer mreže ResNet [31] ili VGG16 [32]. Drugi dio SSD modela zapravo su nadodani konvolucijski slojevi koji u konačnici (posljednji aktivacijski slojevi) rezultiraju graničnim okvirima i klasama detektiranih objekata.



Slika 19. SSD model s VGG16 konvolucijskom mrežom [33]

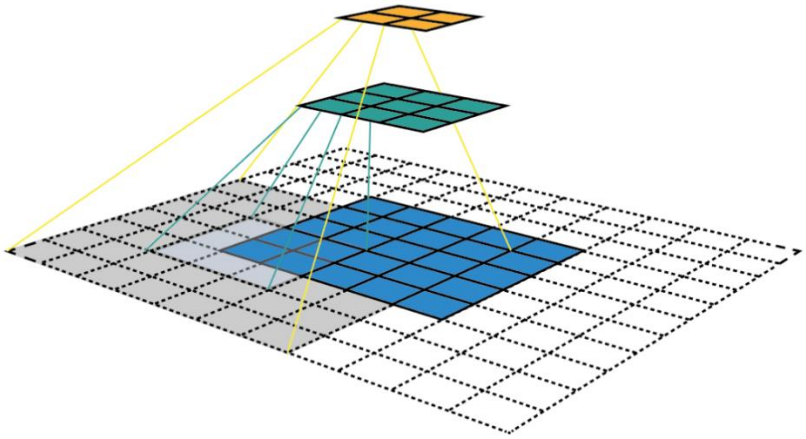
Umjesto pomičnog prozora, SSD koristi mrežu kojom podijeli sliku na ćelije, pri čemu će svaka od ćelija biti odgovorna za detekciju objekata (lokacije i oblika) unutar svojih okvira. Za slučajeve kada se unutar jedne ćelije nalazi više od jednog objekta služe usidreni okviri (eng. anchor box) i receptivna polja (eng. receptive field). Na Slici 20. [31]bazen odgovara duguljastom usidrenom okviru 1, a zgrada odgovara širem okviru 2. U fazi treniranja SSD modela, koristi se faza podudaranja (eng. matching phase) kako bi se odgovarajući usidreni

okviri podudarili s graničnim okvirima temeljne istine objekata sa slike. Usidreni okvir s najvećim stupnjem preklapanja s objektom bit će odgovoran za lokalizaciju i klasifikaciju tog objekta.



Slika 20. Usidreni okviri (eng. anchor boxes)

Receptivno polje odnosi se na područje u ulaznom prostoru koje određena značajka CNN-a promatra, odnosno na koju utječe. Zbog operacije konvolucije, značajke na različitim slojevima predstavljat će različite veličine područja ulazne slike. Što se dublje prolazi kroz mrežu, veličina predstavljena značajkom bit će sve veća.



Slika 21. Receptivna polja

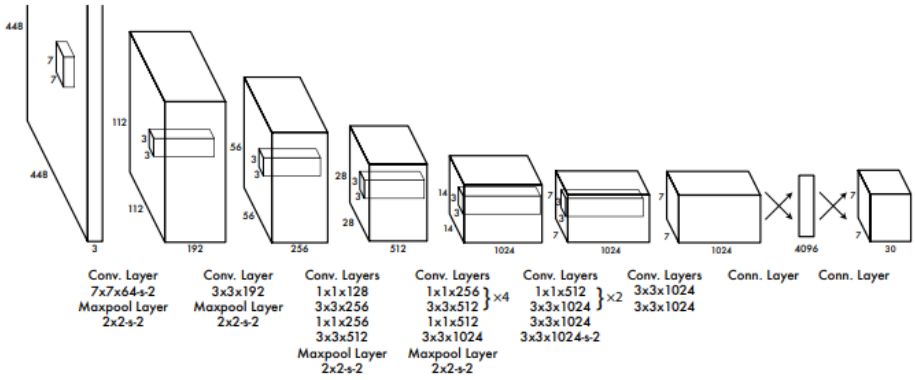
Na Slici 21.[31] ova je pojava jasnije objašnjena. Počne li sa slojem 5x5 te se primijeni konvolucija, dobit će se mapa značajki veličine 3x3 u kojem jedan piksel (jedna značajka)

predstavlja područje ulaznog sloja veličine 3x3. Primijeni li se ponovno konvolucija, dobit će se mapa veličine 2x2, u kojem će svaki piksel korespondirati s područjem ulaza veličine 7x7. Značajke u istim mapama značajki imaju ista receptivna polja i traže iste uzorke, ali na drugim lokacijama. Budući da raniji slojevi nose manja receptivna polja, njima se mogu bolje prepoznati objekti manjih dimenzija na slici. Zbog toga se može definirati hijerarhija ćelija mreže, na primjer 4x4 mreža za pronalaženje manjih objekata, 2x2 mreža za objekte srednjih dimenzija, a 1x1 mreža koristila bi se za pronalaženje objekata čija površina prekriva cijelo područje slike. [31]

4.4.3. YOLO

You Only Look Once (YOLO) vrlo je popularan algoritam za jednofazno prepoznavanje objekata. Prva verzija YOLO algoritma predstavljena je 2016. [34], a od tada je izašlo još nekoliko njegovih varijanti, od kojih je svaka nosila određena poboljšanja. Danas YOLO može parirati ne samo jednofaznim detektorima, već i dvofaznim detektorima u smislu točnosti, ali i u smislu vremena zaključivanja. U ovome modelu pronalaženje objekata shvaćeno je kao pojedinačni problem regresije, direktno od piksela do koordinata graničnih okvira i vjerojatnosti klase. Ovom metodom, na temelju samo jednog pogleda na sliku provodi se predviđanje koji su objekti prisutni na slici i gdje se na njoj nalaze.

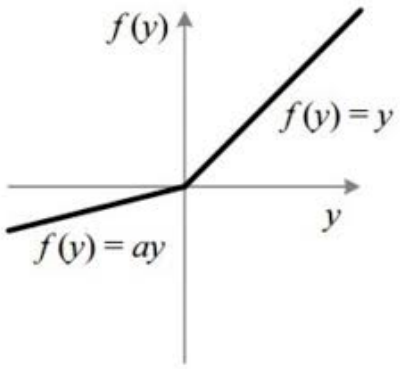
Arhitektura YOLO algoritma vrlo je jednostavna, a inspirirana je GoogLeNet neuronskom mrežom, čiji se grafički prikaz može vidjeti na Slici 22 [34]. U YOLO modelu postoje 24 konvolucijska sloja za kojima slijede 2 potpuno povezana sloja. Od ovih 24 sloja, nakon samo njih 4 slijede slojevi sažimanja maksimalnih vrijednosti. Korišteni su 1x1 konvolucijski slojevi kojima se smanjuju prostori značajki dobiveni iz prethodnih slojeva. Konvolucijski slojevi prethodno su trenirani na ImageNet skupu podataka na pola rezolucije, nakon čega se rezolucija udvostručuje za fazu detekcije.



Slika 22. Arhitektura YOLO modela

Ulazna slika podijeli se na mrežu veličine SxS. Ako centralna koordinata nekog objekta upada u određenu ćeliju, ta ćelija bit će odgovorna za detekciju tog objekta. Svaka ćelija predviđa granične okvire i vrijednost sigurnosti za te okvire. Sigurnost daje informaciju o tome koliko je model uvjeren da se unutar okvira nalazi objekt te koliko je točan položaj okvira. U YOLO modelu može se predviđati više graničnih okvira po jednoj ćeliji. Često dolazi do pojave generiranja više graničnih okvira za jedan te isti objekt, stoga se tijekom treniranja mreže traži samo jedan granični okvir koji će biti odgovoran za pojedini objekt. Na temelju najvišeg IoU predviđanja i temeljne istine, jednom prediktoru dodijeli se uloga prepoznavanja pojedinog objekta. Jedna od ključnih tehnika korištenih u YOLO modelima svakako je ne-maksimalno potiskivanje (eng. Non-Maximum Suppression – NMS) koje pridonosi unaprjeđenju točnosti i efikasnosti prepoznavanja objekata. Pomoću NMS prepoznaju se i uklanjaju suvišni granični okviri tako da se za jedan objekt dobije samo jedan granični okvir. [35] U sloju sažimanja koristi se globalno prosječno sažimanje (eng. Global Average Pooling).

Aktivacijska funkcija korištena u YOLO modelu je Leaky Rectified Linear Unit (LReLU) čiji je graf prikazan na Slici 23 (preuzeta s https://www.quora.com/What-is-leaky-ReLU). Korištena je za sve slojeve, osim posljednjeg, kod kojeg se koristi linearna aktivacijska funkcija. Upravo je taj posljednji sloj odgovoran za predviđanje klasa i koordinata graničnih okvira za detektirane objekte.



Slika 23. Leaky ReLU

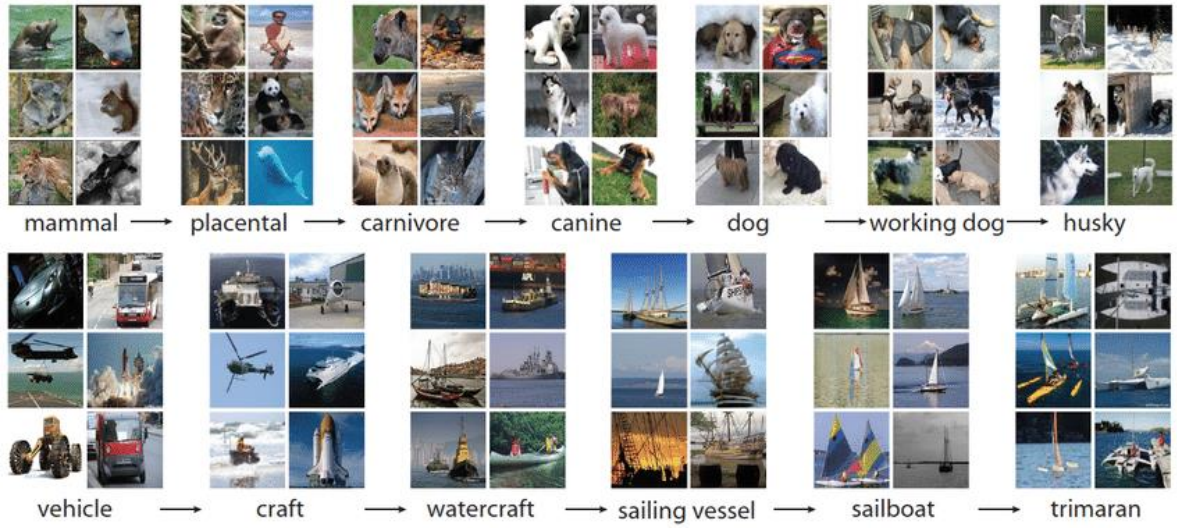
Prednosti YOLO algoritma pred ostalim dotadašnjim metodama detekcije objekata su brojne. YOLO je vrlo brz alat jer se detekcija svodi na problem regresije. Osnovni YOLO model obrađuje slike brzinom od 45 kadrova po sekundi. Nadalje, ovaj alat sagleda cijelu sliku prilikom treniranja i testiranja, stoga globalno razmišlja o svojim predviđanjima. Na primjer, Brza R-CNN mreža ne vidi veći kontekst slike, stoga pogrešno prepoznaje objekte na dijelu slike gdje je vidljiva samo pozadina. YOLO u usporedbi s Brzom R-CNN mrežom čini manje od pola grešaka takve prirode. Prosječna srednja preciznost YOLO algoritma više je od duplo veća nego kod ostalih dotadašnjih sustava prepoznavanja objekata u stvarnome vremenu. Posljednja prednost ovog algoritma njegova je sposobnost da uči generalizirane prikaze objekata. Drugim riječima, treniramo li algoritam na prirodnim slikama, a potom testiramo na umjetničkim djelima, rezultati će biti znatno bolji s YOLO modelom nego s DPM ili R-CNN modelima. [34]

U usporedbi s dvofaznim detektorima objekata, glavni nedostatak prve verzije YOLO-a svakako su velike greške prilikom lokalizacije i manji opoziv (eng. recall – omjer broja objekata točno pretpostavljene klase i ukupnog broja te klase [36]). [15]

4.4.4. YOLOv2

Iste godine kao i YOLO izrađeno je i njegovo prvo poboljšanje – YOLOv2, poznat i kao YOLO9000. [35] Njegovi autori primijenili su skalabilnost u prepoznavanju objekata, za što su koristili poznate skupove podataka ImageNet i COCO (Common Objects in Context). Njihovom su kombinacijom uspjeli dobiti preko 9000 kategorija objekata, čime količina podataka za klasifikaciju postaje ogromna. Temelj arhitekture ove verzije je Darknet-19, varijanta VGGNet arhitekture. Sastoji se od 19 konvolucijskih slojeva i 5 slojeva sažimanja

maksimalnih vrijednosti, a sadržava i mnoge dodatne značajke koje prva verzija YOLO modela ne uključuje. [15]



Slika 24. Primjeri klasa za ImageNet skup podataka [37]

Slika 25. Primjeri klasa za COCO skup podataka [38]

YOLOv2 za predviđanje graničnih okvira koristi kombinaciju usidrenih okvira i predviđene pomake. Time algoritam postaje sposoban obrađivati veći spektar veličina objekata, što je ujedno jedno od glavnih unaprjeđenja ove verzije.

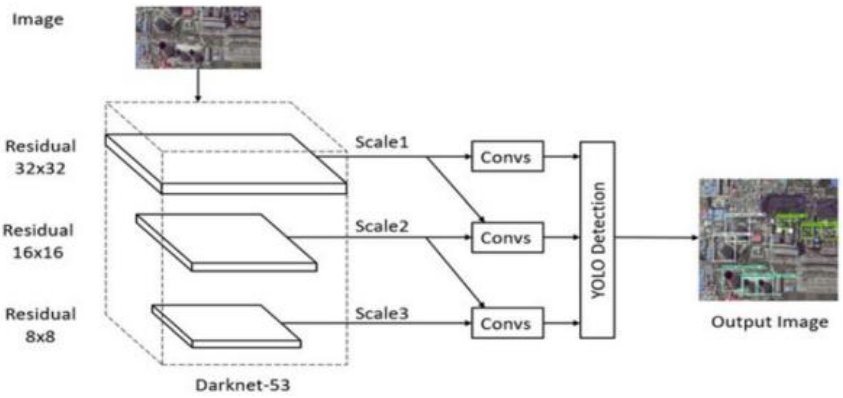
Normalizacija serije (eng. batch normalization) pomaže u povećanju točnosti i stabilnosti modela. Implementira se kako bi se normalizirali izlazi svakog od skrivenih slojeva radi postizanja dosljedne raspodjele težinskih matrica kroz različite slojeve.

Prilikom treniranja YOLOv2 koristi strategiju višerazmjernog treniranja (eng. multi-scale training), što znači da je model treniran na slikama različitih dimenzija, nakon čega se dobiju

prosječna predviđanja. Dimenzije mogu varirati od 320x320 do 608x608. [15] Ova tehnika pridonosi fleksibilnosti modela, posebice u kontekstu predviđanja objekata manjih dimenzija.
Još jedno od poboljšanja ove verzije svakako je nova funkcija gubitka, temeljena na sumi kvadrata pogreške između predviđenih graničnih okvira i temeljne istine, te vjerojatnosti predviđene klase. [35]

4.4.5. YOLOv3

Prve dvije verzije YOLO algoritma nisu imale rješenja za nerijetke greške lokalizacije, te su bile relativno neuspješne u detekciji objekata manjih dimenzija. Kako bi se popravili ti nedostatci, osmišljena je treća verzija YOLOv3 2018. godine, trenirana i evaluirana na COCO skupu podataka. Jedno od glavnih poboljšanja ove verzije upotreba je konvolucijske neuronske mreže temeljene na Darknet-53 modelu. Mreža se sastoji od 53 konvolucijska sloja te koristi 3x3 i 1x1 kernele uz još dodatne prečace (eng. shortcut connections). Uz ovu mrežu, model YOLOv3 sadrži još dodatna 53 sloja.



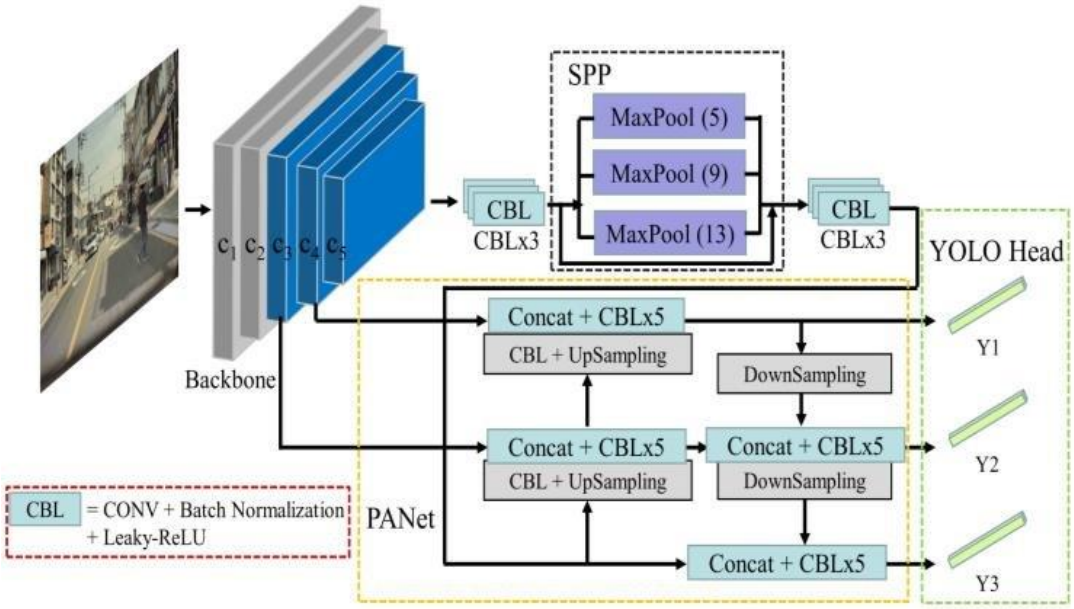
Slika 26. Arhitektura YOLOv3

Ova verzija YOLO detektora inspirirana je FPN – piramidalnom mrežom značajki. Poput FPN modela YOLOv3 koristi 1x1 konvoluciju na mapama značajki kako bi detektirao objekte. Mape značajki generirane su u tri različita razmjera, uz faktore 32, 16 i 8.
Još jedno od unaprjeđenja primjena je usidrenih okvira različitih veličina, za razliku od YOLOv2 koji je koristio usidrene okvire jednakih dimenzija. Stoga je poboljšana mogućnost detekcije objekata različitih veličina, ali je postignuta i veća točnost dimenzija graničnih okvira za odgovarajući objekt.

4.4.6. YOLOv4

YOLOv4 četvrta je verzija YOLO algoritma za detekciju objekata, predstavljena 2020. godine. Ovim modelom postignute su znatne promjene, predstavljene su nove metode i prikazane razne kompleksne i moćne tehnike. Rezultati dobiveni ovom verzijom, u usporedbi s prethodnima, mnogo su bolji u pogledu vremena izvođenja zadataka, ali i točnosti. Također, ova verzija postaje dostupna svima koji koriste grafičku procesorsku jedinicu (eng. Graphics Processing Unit – GPU). [39]

Arhitektura YOLOv4 modela može se svesti na okosnicu CSPDarknet53, dodatni modul SPP, PANet kao vrat mreže i YOLOv3 kao glavu modela, a shematski je prikazana na Slici 27 (preuzeta s https://www.researchgate.net/figure/Overall-structure-of-YOLOv4-including-CSPDarknet-backbone-SPPnet-PANet-and-3-YOLO_fig2_344919620). CPSNet odnosi se na Cross Stage Partial Network, odnosno „Djelomična mreža između faza“, koja je dizajnirana specifično za obavljanje zadataka prepoznavanja objekata. CSPDarknet53 relativno je plitka mreža koja sadrži 29 konvolucijskih slojeva s kernelom 3x3, receptivno polje (eng. receptive field) veličine 725x725 i 27,6 milijuna parametara. Tvorci YOLOv4 modela eksperimentalno su došli do zaključka da je ova arhitektura optimalna za četvrtu verziju YOLO-a.



Slika 27. Arhitektura YOLOv4

Prostorno piramidalno sažimanje (eng. Spatial Pyramid Pooling – SPP), postavljeno je između neuronske mreže i potpuno povezanoga sloja [28], gdje preslikava bilo koju ulaznu veličinu u

izlaz fiksne veličine. Primjena SPP znatno povećava receptivno polje, izdvaja najznačajnije značajke konteksta slike, pri čemu ne uzrokuje gotovo nikakvo smanjenje brzine rada mreže.
Za razliku od YOLOv3 koji koristi FPN, YOLOv4 koristi PANet (Path Aggregation Network) kao metodu agregacije parametara s različitih razina okosnice za različite razine detektora. [39]
Bitna korištena metoda svakako je i Modul prostorne pažnje (eng. Spatial Attention Module – SAM) koji doprinosi tome da se mreža može fokusirati na bitnije informacije umjesto da uči nekorisne pozadinske informacije. [40]
Još jedna novost u YOLOv4, je ta da za razliku od YOLOv3, gdje je samo jedna usidrena točka (eng. anchor point) bila odgovorna za jednu temeljnu istinu, sada je za jednu temeljnu istinu odgovorno njih nekoliko. Broj usidrenih okvira ostaje nepromijenjen, ali razmjer odabira pozitivnih uzoraka se povećao. Implementirana je i funkcija gubitka Potpuni presjek preko unije (eng. Complete Intersection over Union – CioU), koja brzo konvergira. [41]

4.4.7. YOLOv5

Model YOLOv5 predstavljen je kao projekt otvorenog koda, održavan od strane Ultralytics platforme za strojno učenje, koja je orijentirana dostupnosti umjetne inteligencije svima [42]. Ova je verzija izdana iste godine kao i YOLOv4, od strane istog tima. Mnogi su bili skeptični oko novog modela jer se čini manje inovativan od svog prethodnika, no YOLOv5 pokazuje poboljšanja performansi na više polja:

- PyTorch okvir jednostavan je za korištenje i za obuku vlastitog skupa podataka.
- Čitanje koda je lagano, integracija velikog broja tehnologije računalnog vida.
- Lako konfiguriranje okruženja, brzo treniranje modela.

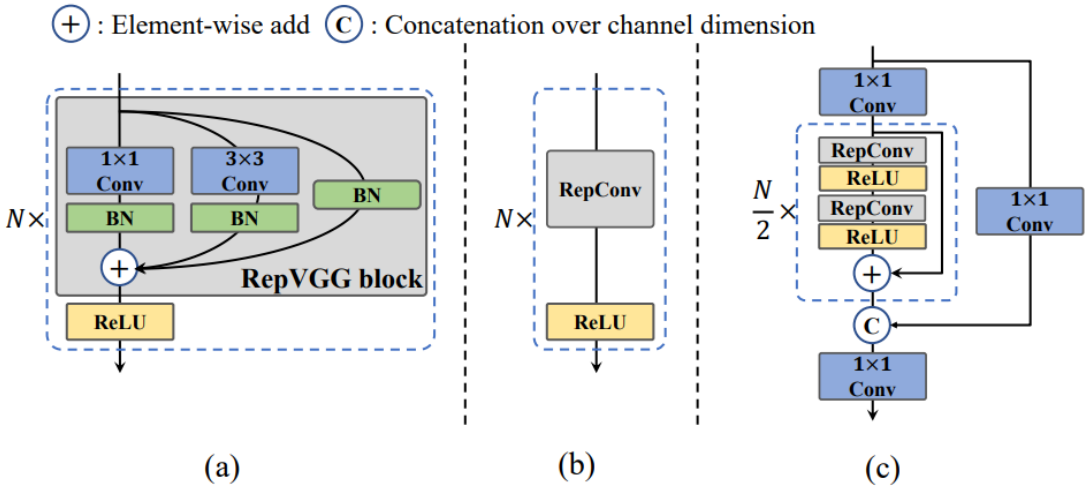
Svaka serija podataka za trening prolazi kroz učitavač (eng. data loader) koji na slikama obavlja poboljšanja: skaliranje, podešavanje prostora boja i poboljšanje mozaika. Rezultatima je dokazano da poboljšanje mozaika rješava problem pronalaženja objekata malih dimenzija na slici. [41]
Okosnica YOLOv5 modela kao i kod YOLOv4 je CSPDarknet53, nakon koje slijedi modul SPP koji u ovoj verziji nosi poboljšanja arhitekture zahvaljujući kojima postiže bolje rezultate. Za metodu agregacije parametara ponovno se koristi PANet, koji uključuje značajke naučene u okosnici i skraćuje put informacija između nižih i viših slojeva. Glava YOLOv5 modela sastoji se od tri grane od kojih svaka obavlja predviđanja na različitim mjerilima, odnosno koriste se mreže veličina 13x13, 26x26 i 52x52. Svaka ćelija predviđa tri granična okvira. Svaki od tih

ogranaka glave modela rezultira graničnim okvirima, vjerojatnošću klasa i iznosom sigurnosti. Kako bi se riješio problem preklapajućih graničnih okvira, koristi se NMS metoda. [43]

YOLOv5 koristi i usidrene okvire, ali metoda njihovog generiranja nešto je drugačija nego kod prethodnih verzija. Tako zvani „dinamični usidreni okviri“ (eng. dynamic anchor boxes) koriste algoritam klasterizacije kako bi se grupirali granični okviri jednake temeljne istine u klastere nakon čega će težišta klastera postati usidreni okviri. Ovakav pristup omogućuje usidrenim okvirima da budu bolje usklađeni s veličinom i oblikom detektiranih objekata. [35]

4.4.8. YOLOv6

YOLOv6 izdan je 2022. godine kao unaprjeđenje prethodnih verzija. U ovoj je verziji predstavljena nova okosnica EfficientRep, temeljena na RepVGG i CSPStackRep blokovima. Uočeno je da RepVGG imaju veću snagu predstavljanja značajki u malim mrežama, pri sličnoj brzini donošenja odluka, dok se za dobivanje većih modela teško mogu skalirati zbog velikog porasta broja parametara i računalnih troškova. Upravo se zato RepBlokovi uzimaju kao osnove manjih modela tijekom faze treniranja. Svaki RepBlok pretvoren je u skup 3x3 konvolucijskih slojeva s ReLU aktivacijskom funkcijom, što omogućuje mreži dostatnu uporabu snage računala. No, s povećanjem kapaciteta mreže rastu troškovi i broj parametara, stoga na snagu stupa CSPStackRep Blok. Ovaj blok sastoji se od tri konvolucijska 1x1 sloja i od skupa podblokova koje predstavljaju dva RepConv bloka. Za jasnije objašnjenje ovakve arhitekture može se promotriti prikaz na Slici 28 [44].



Slika 28. (a) RepBlok; (b) RepBlok konvertiran u RepConv; (c) CSPStackRep Blok

Vrat mreže temeljen je na PAN modelu uz poboljšanja kojima se dobiva RepPAN. Glava mreže pojednostavljena je kako bi postala učinkovitija, koristi strategiju hibridnih kanala (eng. hybrid-channel strategy), te dobiva naziv Učinkovita Odvojena Glava (eng. Efficient Decoupled Head). [44]

Za dodjelu oznaka klasa koristi se pristup Učenja usklađivanja zadataka (eng. Task Alignment Learning) – TOOD. Koriste se i nove funkcije gubitaka za klasifikaciju i regresiju. Za klasifikaciju koristi se VariFocal gubitak, a za problem regresije SIoU ili GIoU funkcije. [45]

U YOLOv6 algoritmu javlja se i strategija samodestilacije za regresiju i klasifikaciju, kako bi se dodatno unaprijedila točnost modela. Ova tehnika svodi se na to da se učitelj ograniči da bude i sam učenik, ali prethodno obučen, odakle i potječe izraz samodestilacija. Još jedna novost koja je implementirana u novoj verziji promijenjena je kvantizacijska shema za detekciju koristeći RepOptimizer i kanalna destilacija (eng. channel-wise distillation). Njihova uporaba dovodi mrežu do još boljih rezultata brzine i točnosti rada modela. [44]

Zaključno, YOLOv6 u smislu točnosti i brzine premašuje sve svoje prethodnike.

4.4.9. YOLOv7

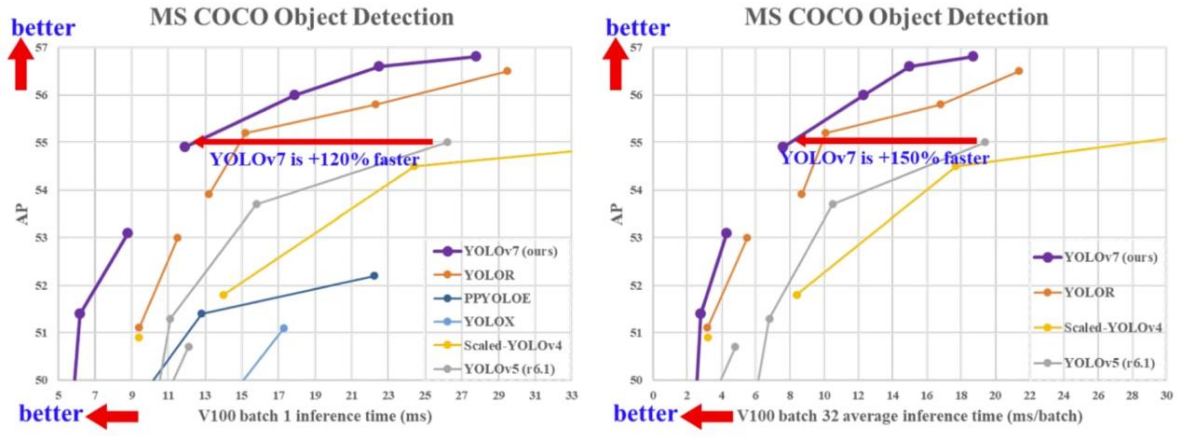
YOLOv7, objavljen 2022. godine [46], nadmašuje sve dotadašnje detektore u brzini i točnosti i to u rasponu od 5 FPS do 160 FPS (frames per second – slika u sekundi). Poput YOLOv4 treniran je koristeći samo MS COCO skup podataka bez predtreniranih okosnica. Ova verzija uvodi nekoliko promjena u arhitekturi modela te seriju tako zvanih bag-of-freebies koja povećava točnost bez da utječe na vrijeme zaključivanja, nego samo na vrijeme treniranja.

Arhitektura YOLOv7 modela temelji se na Proširenoj učinkovitoj mreži agregacije slojeva (eng. Extended efficient layer aggregation network – E-ELAN). E-ELAN radi za modele bez ograničenja na broj računalnih blokova, kombinira obilježja različitih grupa miješanjem i spajanjem kardinalnosti kako bi se unaprijedilo učenje mreže.

Skaliranjem se generiraju modeli različitih veličina tako što se prilagode neki atributi modela. Arhitektura YOLOv7 temeljena je na ulančavanju (eng. concatenation-based), zbog čega tipične tehnike skaliranja uzrokuju promjenu u omjeru između kanala ulaza i kanala izlaza iz prijelaznog sloja, što dovodi do smanjenja upotrebe hardvera modela. Zato YOLOv7 predlaže novu strategiju skaliranja u kojoj se dubina i širina bloka skaliraju istim faktorom u svrhu održanja optimalne strukture modela.

U YOLOv7 korištena je bag-of-freebies radi povećanja točnosti detekcije bez povećanja vremena zaključivanja. Ono što je uključeno u bag-of-freebies planirana je reparametrizirana konvolucija, kojom se uklanja veza identiteta prilikom konverzije RepBloka i time se dobiva RepConvN. Nadalje, uključena je gruba dodjela oznaka (eng. coarse label assignment) za pomoćnu glavu, koja pomaže tijekom treniranja mreže, te fina dodjela (eng. fine label assignment) za vodeću glavu, koja je odgovorna za konačni izlaz. Normalizacija serije integrira prosjek i varijancu normalizacije u pristranost (eng. bias) i težine konvolucijskog sloja u fazi zaključivanja. Bag-of-freebies sadrži i implicitno znanje inspirirano 2021. godine izdanim YOLOR (You Only Learn One Representation), koji se odnosi na učenje više zadataka odjednom, te sadrži konačno i prosjek eksponencijalnog kretanja kao konačni model zaključivanja. [45]

Jedna od glavnih prednosti YOLOv7 modela njegova je brzina koja ga čini primjenjivim za probleme u stvarnom vremenu, kao na primjer nadzorne kamere, samovozeći automobili i slični slučajevi u kojima je vrlo važno da brzine obrade podataka budu velike. Što se tiče rezultata, YOLOv7 na skupu podataka COCO postiže prosječnu preciznost od 37,2% s pragom IoU od 0,5, što je odličan rezultat, ali i dalje ne dostiže preciznost dvofaznih objekata. [35] Na Slici 29. [46] prikazana je prosječna preciznost u ovisnosti o vremenu zaključivanja (lijevo) i vremenu zaključivanja po seriji (desno). Na grafovima su prikazani rezultati za različite detektore primijenjene u stvarnom vremenu te se može zaključiti kako YOLOv7 svojim rezultatima nadmašuje svaki od modela s kojima je uspoređen.



Slika 29. Usporedba YOLO modela na MS COCO skupu podataka

S druge strane, ono što nedostaje YOLOv7 modelu i dalje je točnost u detekciji manjih objekata, ali i objekata znatno većih dimenzija od ostalih objekata na slici. Uz navedeno, ovaj model

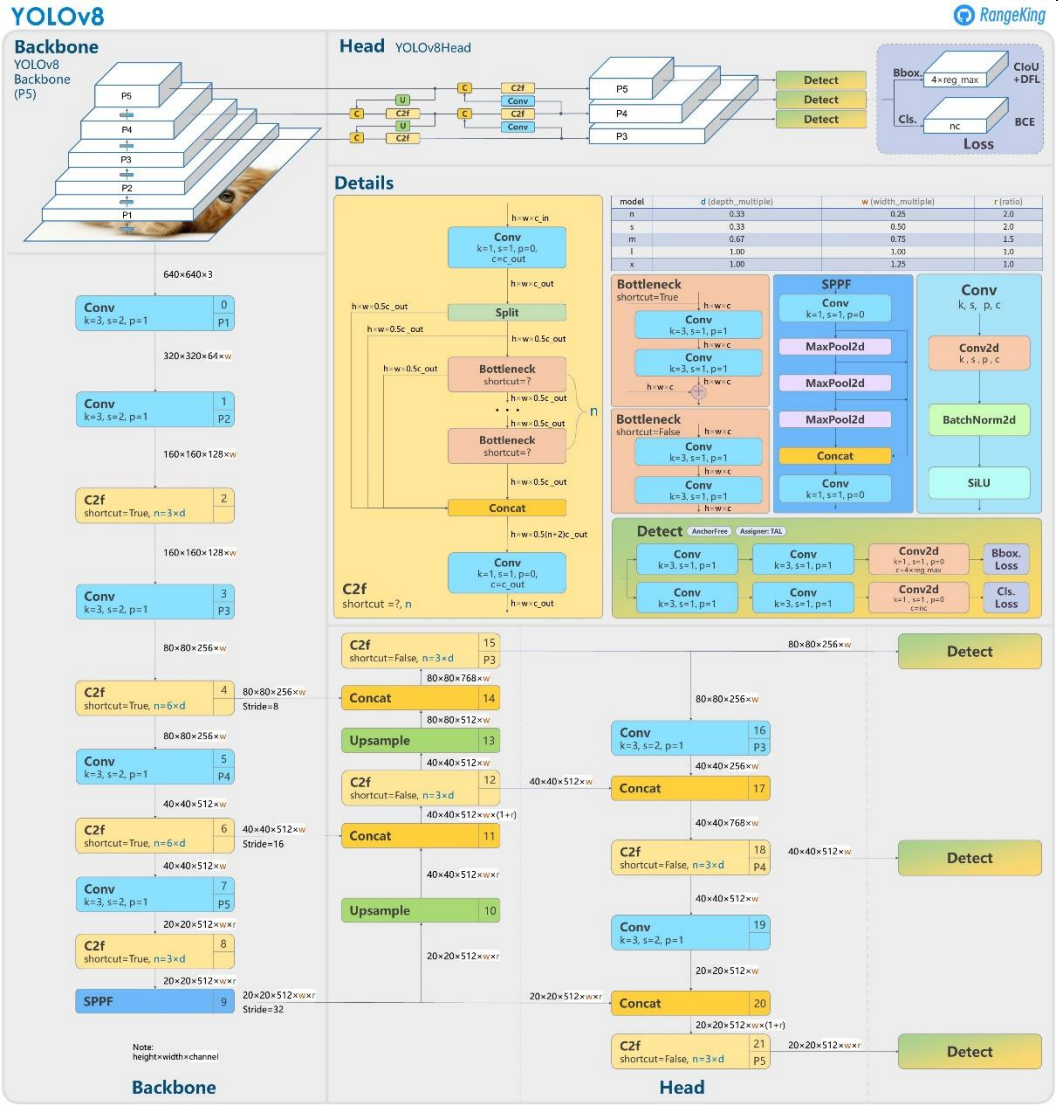
može biti osjetljiv i na promjene osvjetljenja i neke druge uvjete okoline, stoga ga može biti nepraktično koristiti u uvjetima stvarnog svijeta gdje su uvjeti promjenjivi. [35]

4.4.10. YOLOv8

YOLOv8 najnovija je verzija YOLO detektora objekata izdana u siječnju 2023. godine od strane Ultralytics-a. Ova verzija ima sličnu arhitekturu kao i njeni prethodnici, ali predstavlja i mnoga unaprjeđenja u odnosu na njih. YOLOv8 nudi pet skaliranih verzija: YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), YOLOv8l (large) te YOLOv8x (extra large). Ovom verzijom mogu se obavljati razni vizijski zadatci poput detekcije objekata, segmentacije, procjene položaja, praćenje i klasifikacija. [45]

Arhitektura YOLOv8 može se podijeliti u dvije glavne komponente: okosnica i glava. Okosnica je slična kao i u YOLOv5 modelu uz određene promjene. CSPDarknet53 arhitektura sadrži 53 konvolucijska sloja, kao i u YOLOv5 modelu, no promjena nastaje u gradivnim blokovima modula koji slijedi nakon te mreže. Modul se sada naziva C2f, a ne više C3. U C2f svi su izlazi iz vrata (eng. bottleneck – 3x3 konvolucije s rezidualnim vezama) ulančani, dok je u C3 bio korišten samo izlaz iz zadnjeg vrata. U vratu su značajke povezane direktno bez forsiranja istih dimenzija kanala, što umanjuje broj parametara i ukupnu veličinu tenzora. [47]

Glava YOLOv8 sadrži nekoliko konvolucijskih slojeva, koje potom slijede potpuno povezani slojevi, odgovorni za predviđanje graničnih okvira, objektivnosti te vjerojatnosti klasa za prepoznate objekte. Za vjerojatnosti klasa koristi se softmax funkcija, a kao aktivacijska funkcija u izlaznom sloju koristi se sigmoidna funkcija za objektivnost (vjerojatnost da granični okvir sadrži objekt). Kao funkcije gubitka koriste se CIoU i DFL funkcije za granične okvire, a binarna unakrsna entropija (eng. binary cross-entropy) za gubitke klasifikacije. U zadatcima regresije, klasifikacije i objektivnosti važnu ulogu ima i činjenica da se ne koriste usidreni okviri. Time se svakoj grani omogućuje da se usredotoči na svoj zadatak i poboljša ukupnu točnost modela.



Slika 30. Arhitektura YOLOv8 [47]

Jedna od bitnijih karakteristika glave uporaba je mehanizma samopažnje koji omogućava modelu da se usredotoči na različite dijelove slike i prilagodi važnost različitih značajki. Također važna značajka svakako je FPN model za detekciju objekata različitih veličina i mjerila na slici. [48]

Tijekom treniranja mreže, YOLOv8 povećava slike. U svakoj epohi, model vidi nešto drugačiju varijantu dostupne slike. Jedno od tih povećanja zove se povećanje mozaika (eng. mosaic augmentation). Ono uključuje spajanje četiriju slika zajedno, tjerajući tako model da uči prepoznati objekte na novim lokacijama, u djelomičnoj okluziji te u okolini različitih piksela. Međutim, ovakva povećanja dokazano degradiraju izvedbu modela ukoliko se koristi kroz cijelu fazu treniranja, stoga ih se isključuje u posljednjih deset epoha treninga mreže.

Tablica pokazuje točnosti YOLOv8 modela primijenjenih na MS COCO skupu podataka. Rezultati su izvrsni, primjerice YOLOv8m model dostiže srednju prosječnu preciznost mAP od 50,2, a uporabom YOLOv8x može se doseći mAP od čak 53,9.

Model	mAP
YOLOv8n	37,3%
YOLOv8s	44,9%
YOLOv8m	50,2%
YOLOv8l	52,9%
YOLOv8x	53,9%

Tablica 4. COCO točnosti YOLOv8 modela

5. PRAĆENJE OBJEKATA

Praćenje (eng. tracking) u kontekstu dubokog učenja zadatak je koji se odnosi na predviđanje položaja objekata na videu koristeći njihove prostorne i vremenske karakteristike. Proces praćenja generalno se sastoji od dva koraka:

1. Modul odgovoran za detekciju i lokalizaciju ciljanog objekta.
2. Prediktor pokreta (eng. motion predictor) odgovoran za predviđanje budućih pokreta na temelju dotadašnjih informacija.

Praćenje je korisno upotrebljavati uz prepoznavanje objekata iz više razloga. Prvo, detektori objekata samo prikazuju lokaciju objekta, no gledajući samo izlaze, ne može se znati koje koordinate pripadaju kojem objektu. Koristeći praćenje svakom objektu dodjeljuje se identifikacijski broj (ID) koji je pripisan objektu sve dok se on nalazi u kadru. Nadalje, tragači (eng. trackers), odnosno alati za praćenje objekata, generalno su brži od detektora zbog čega su vrlo pogodni za uporabu u stvarnom vremenu. Primjenu često pronalaze u praćenju prometa. Također u sportu kada je potrebno pratiti loptu ili igrača, kako bi se primjerice odredio prekršaj ili potvrdio gol. Vrlo je korisna i primjena nadzora pomoću više kamera, gdje praćena osoba, ukoliko izađe iz kadra jedne kamere, ali se pojavi u kadru druge kamere, zadržava svoj ID, olakšavajući tako praćenje te osobe.

5.1. Podjela

Tragači se mogu klasificirati na dva načina. Prva podjela, s obzirom na broj praćenih objekata, odnosi se na tragače jednog objekta i tragače više objekata. Druga podjela prema korištenju detekcije dijeli tragače na one koji koriste detekciju i one koji ju ne koriste.

Tragači jednog objekta (eng. single object tracker) prate samo jedan objekt na videu, čak i ako se na videu pojavljuje više objekata. Prvo lokaliziraju objekt te ga prate kroz daljnje kadrove, a njihov princip rada daje vrlo brze rezultate. Temeljeni na tradicionalnom računalnom vidu su CSRT i KCF tragači, no tragači temeljeni na dubokom učenju daju daleko bolje rezultate. U njih se ubrajaju primjerice SiamRPN i GOTURN.

Tragači više objekata (eng. multiple object tracker) mogu pratiti više objekata u kadru. Trenirani su na velikoj količini podataka, mogu istovremeno pratiti čak i objekte različitih klasa, ne gubeći na brzini i točnosti. Neki od najkvalitetnijih algoritama koji spadaju u ovu vrstu tragača su primjerice DeepSORT, JDE i CenterTrack.

Praćenje detekcijom (eng. tracking by detection) odnosi se na algoritme u kojima detektor prepozna objekt kroz kadrove te koristeći povezivanje podataka kroz te kadrove generira trajektorije kretanja objekta. Ova je metoda korisna pri praćenju više objekata te može biti od velike pomoći u slučajevima kada detektor objekata ne radi ispravno.

Praćenje bez detekcije (eng. tracking without detection) vrsta je algoritama u kojima se koordinate objekta ručno inicijaliziraju, nakon čega se objekt prati u daljnjim kadrovima. No, ova vrsta koristi se većinski u algoritmima tradicionalnog računalnog vida.

5.2. SORT

SORT (eng. Simple Online Realtime Tracking) način je praćenja objekata u kojem se koriste Kalman filteri i mađarski algoritmi (eng. Hungarian algorithms). Sastoji se od četiri glavne komponente koje su redom:

1. **Detekcija**: ovim prvim korakom detektor prepozna objekt koji će biti praćen.
2. **Procjena**: detekcija napreduje iz trenutnog kadra u sljedeći procjenom položaja ciljanog objekta u sljedećem kadru koristeći model konstantne brzine.
3. **Povezivanje podataka**: kako sada postoje granični okviri praćenja i granični okviri detekcije, koristi se IoU metoda i mađarski algoritam za optimizaciju rezultata.
4. **Nastanak i brisanje identiteta**: praćenim objektima dodjeljuju se pojedinačni identifikacijski brojevi – identiteti (ID). Objektu je pripisan ID sve dok je objekt u kadru, kada nestaje iz kadra taj ID se briše. [49]

5.3. DeepSORT

SORT algoritam daje dobre rezultate u kontekstu preciznosti i točnosti praćenja, no ima velike probleme s dodjelom ID brojeva i s okluzijom. DeepSORT kao ekstenzija SORT algoritma implementira duboko učenje tako što koristi deskriptor izgleda, čime rješava problem dodjele identiteta praćenim objektima. Dakle, DeepSORT ne temelji praćenje objekta samo na brzini kojom se objekt kreće, već i na izgledu objekta. To je iznimno moćan i brz alat koji je postao vodeći algoritam za praćenje objekata.

Slika 31. Identifikacijski brojevi korištenjem DeepSORT algoritma [50]

Za treniranje modela korištena je kosinusna metrika (eng. cosine) koja pomaže modelu da povrati identitete u slučaju dugotrajne okluzije objekta. Ipak, okluzija ostaje jedan od najvećih problema u modelima praćenja objekata, posebice primjerice tijekom praćenja osoba u gužvi, gdje se ljudi kreću iz i u kadar velikim brzinama. Drugi problem koji ovi modeli moraju prebroditi potreba je za velikom količinom podataka za učenje. Skup podataka za učenje mora se sastoji od velike količine različitih videa na kojima su prikazani razni pokreti različitih objekata na različitim lokacijama. [51]

6. VIZIJSKI SUSTAV ZA PREBROJAVANJE OSOBA

Za izradu ovog diplomskog rada korišten je Github repozitorij kreatora „MuhammadMoinFaisal“ na poveznici https://github.com/MuhammadMoinFaisal/YOLOv8-DeepSORT-Object-Tracking. Repozitorij sadrži sve datoteke potrebne za izradu vizijskog sustava za prepoznavanje i praćenje osoba. [52]

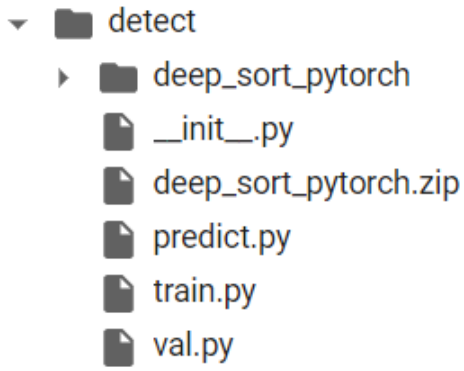
Projekt je moguće izraditi ili u PyCharm okruženju ili pomoću Google Colab poveznice koristeći Python programski jezik. U ovome slučaju projekt je izrađen kao Google Colab datoteka radi bolje preglednosti koraka koji su obavljeni.

Prije samog početka rada potrebno je namjestiti GPU kao vrijeme izvođenja. Nakon što se klonira GitHub javni repozitorij YOLOv8-DeepSORT-Object-Tracking, ulazi se u odabrani direktorij (eng. directory) na Slici 32. te se u njemu instaliraju sve potrebne ovisnosti (eng. dependencies). U ovisnosti se ubrajaju sve knjižnice koje je potrebno instalirati kako bi kod u konačnici radio.

Slika 32. Direktorij YOLOv8-DeepSORT-Object-Tracking

Potom se ulazi u mapu za detekciju „detect“ koja sadrži skripte za treniranje, validaciju i predviđanja, kao što je prikazano na Slici 33. Budući da je model u ovome radu predtrenirani, ne koriste se skripte za trening i validaciju. U spomenutoj mapi preuzme se DeepSORT datoteka, koja je potrebna kako bi bilo moguće izvršiti praćenje prepoznatih objekata na videu.

Kako bih stekla bolje znanje o YOLOv8 algoritmu za pronalaženje objekata, upoznala se s njegovim mogućnostima i područjima primjene, upisala sam tečaj kreatora GitHub repozitorija, Muhammada Moina na stranici Udemy [53]. U svojem tečaju objašnjava i spomenuti repozitorij te Python kod korišten za pronalaženje i praćenje objekata, stoga će se objašnjenja u nastavku većinski oslanjati na njegova predavanja s ovoga tečaja.

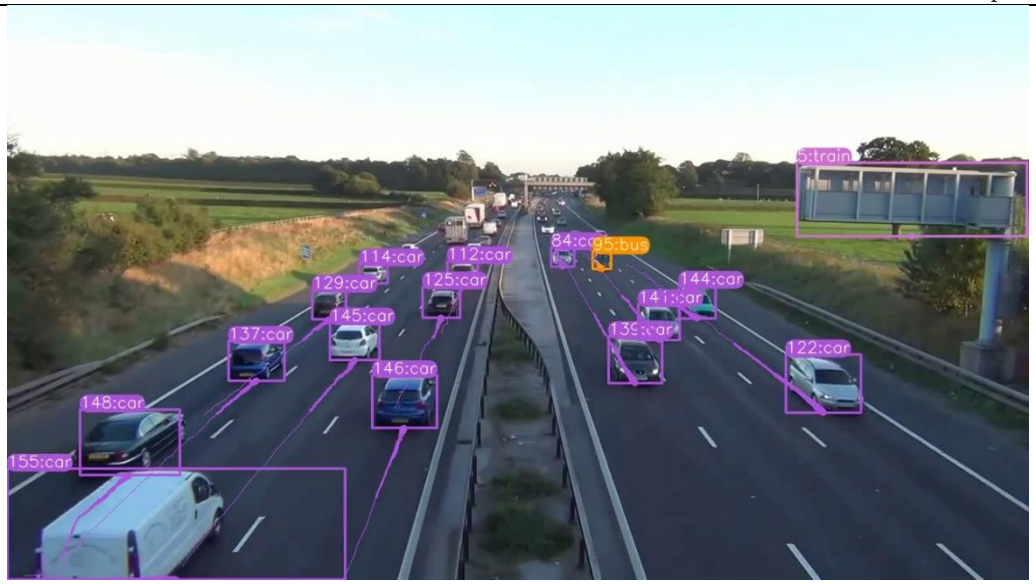


Slika 33. Mapa detect

6.1. Testiranje GitHub repozitorija na primjeru automobila

Radi potvrde da GitHub repozitorij radi kako je predviđeno, prvo su provedeni koraci točno kako su navedeni u repozitoriju. Video na koji je potrebno primijeniti model snimka je automobila na autocesti, čiji je jedan kadar vidljiv na Slici 34. Nakon provedenih svih koraka bez unosa ikakvih promjena dobije se rezultat kao na Slici 35. Bez dubinske analize već na prvi pogled može se uočiti kako model radi ispravno, odnosno prepoznaje sva vozila, no ne radi savršeno. Za neka vozila ne prepoznaje ispravnu vrstu vozila, na primjer jedan automobil prepoznat je kao autobus, a jedno kombi vozilo prepoznato je kao automobil. Također, u gornjem desnom uglu slike struktura koja uopće nije vozilo pogrešno je identificirana kao vlak. No, zaključak je prvog testa kako je repozitorij ispravno kreiran.

Slika 34. Prvi testni video



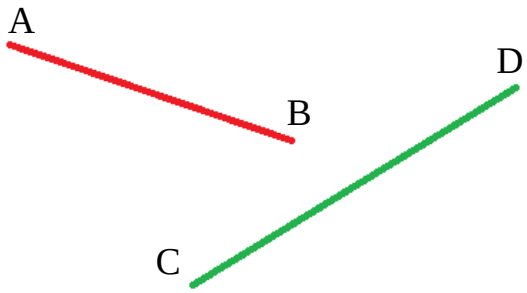
Slika 35. Rezultati prvog testa

6.2. Testiranje prebrojavanja vozila

Sljedeći korak bio je dodatan drugi test preuzetog koda, uz određene preinake. Naime, na sljedećem primjeru, osim prepoznavanja vozila i njihovog ispravnog svrstavanja, dodatno se provjerava u kojem se smjeru vozila kreću, te se nakon prelaska definirane linije pribrojavaju broju vozila koji se kreće prema dolje, odnosno vozilima koji se kreću prema gore. Kako bi to bilo moguće izvesti, prvo je potrebno dopisati linije koda kojima se provjerava je li objekt prešao liniju, odnosno je li došlo do intersekcije dviju linija. To se definira sljedećim dvjema funkcijama, čiji će izlaz na kraju biti True (točno) ukoliko je došlo do intersekcije:

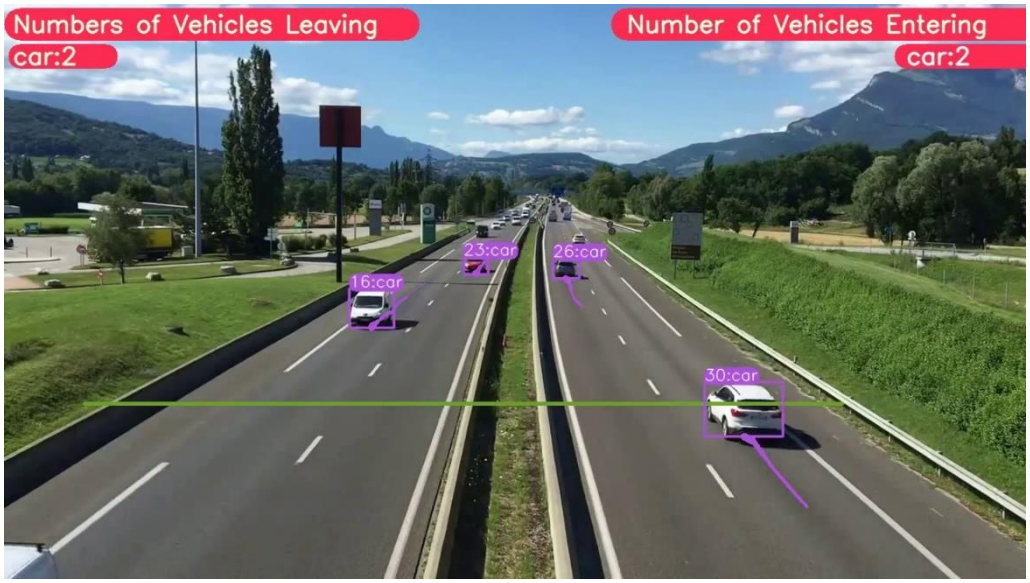
```
def ccw(A,B,C):
    return (C[1]-A[1]) * (B[0]-A[0]) > (B[1]-A[1]) * (C[0]-A[0])

def intersect(A,B,C,D):
    return ccw(A,C,D) != ccw(B,C,D) and ccw(A,B,C) != ccw(A,B,D)
```

Prema Slici 36. može se jasnije pojasniti funkcija intersekcije. Promatraju li se dvije linije, AB i CD, ukoliko dođe do njihovog presjeka, odnosno produži li se linija AB preko CD linije, doći će do intersekcije koja će biti prepoznata pomoću spomenutih linija koda.



Slika 36. Objašnjenje funkcije intersect

U ovom specifičnom primjeru, prva linija AB odnosi se na trag praćenog objekta, prikazanog na Slikama 35. i 37. kao trajektorije, a druga linija CD predstavljena je zelenom linijom na Slici 37. U nastavku ovoga poglavlja na primjeni koda za praćenje ljudi pobliže će biti objašnjeni i svi ostali relevantni elementi koda.



Slika 37. Rezultati drugog testa

Provedbom drugog testa zaključuje se da linije koda koje se odnose na intersekciju rade ispravno, jer je prepoznat svaki slučaj kada vozilo prijeđe definiranu granicu – zelenu liniju. Ono što je nedostatak YOLOv8, kao i prethodnih verzija YOLO algoritama, svakako je nemogućnost prepoznavanja objekata koji su relativno malih dimenzija u odnosu na dimenzije slike. Sukladno tom nedostatku, u oba testna primjera, vozila koja su u daljini nisu prepoznata

sve dok se ne približe, odnosno ona koja se kreću prema gore nakon nekog trenutka više se ne mogu detektirati, zbog čega gube granične okvire i trag.

6.3. Prebrojavanje osoba – primjeri preuzeti s interneta

Cijeli proces prepoznavanja, praćenja i prebrojavanja osoba odvija se na sljedeći način:

1. Implementira se YOLOv8 algoritam za prepoznavanje objekata koji će u ovome radu biti korišten za prepoznavanje osoba, te će rezultat biti granični okviri oko prepoznate osobe i odgovarajuća oznaka klase (eng. label) koja će u slučaju ispravne detekcije glasiti „person“.
2. Pronalazak centralne koordinate graničnog okvira.
3. Implementacija praćenja objekata pomoću DeepSORT-a kojim se svakom prepoznatom objektu dodjeljuje jedinstveni nasumično generirani identifikacijski broj ID.
4. Crtanje linija koje će predstavljati granice. Kada praćeni objekt prijeđe granicu, bit će pribrojen ovisno o smjeru kretanja.

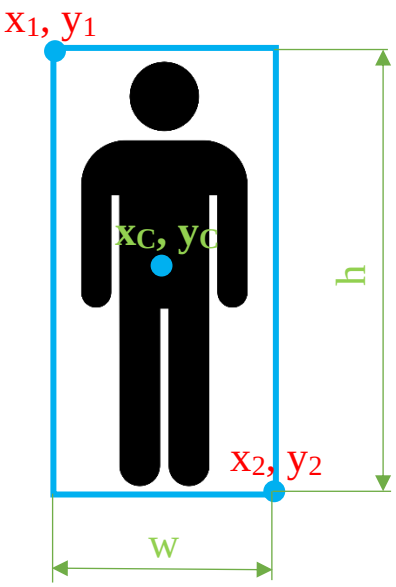
Za praćenje kretanja objekata, u ovome su radu korištena dva pristupa zadatku. Jedan pristup odnosio se na kretanje kada se sve osobe kreću po istoj stazi, pri čemu se neke kreću prema dolje, a neke prema gore. Drugi slučaj odnosi se kada se osobe kreću u definiranom smjeru po pojedinoj stazi, primjerice na pokretnim stepenicama, koje imaju definiranu putanju za osobe koje se kreću prema gore, te drugu putanju za osobe koje se kreću prema dolje. Ta dva pristupa su vrlo slična, no imaju manje razlike, stoga će kasnije u poglavlju te iste razlike biti pobliže objašnjene.

6.3.1. Objašnjenje koda

Kako bi cijeli kod radio ispravno, potrebno je prije svega importirati sve potrebne knjižnice i module. Za funkcioniranje YOLOv8 algoritma potrebno je preuzeti torch te ultralytics knjižnice. Također, za implementaciju prepoznavanja objekata, ali i praćenje, potrebno je imati importiranu cv2 knjižnicu. Praćenje objekata u ovom slučaju zahtjeva i DeepSORT module. Za generiranje nasumičnih vrijednosti boja za različite klase, potrebno je importirati random modul, a modul deque bit će pobliže opisan u nastavku.

Jedna od bitnijih funkcija korištenih u kodu funkcija je `xyxy_to_xywh(*xyxy)`. Prati li se prikaz na Slici 38. može se vrlo lako shvatiti i zašto. Naime, YOLOv8 algoritam svoja

predviđanja daje u obliku rubnih koordinata graničnih okvira, odnosno koordinate (x_1, y_1) za gornji lijevi vrh te koordinate (x_2, y_2) za donji desni vrh, na slici označene crvenom bojom.



Slika 38. Predviđanja YOLOv8 u odnosu na DeepSORT

Problem je u tome što DeepSORT model za praćenje objekata svoja predviđanja daje na temelju centralne koordinate graničnih okvira (x_C, y_C) te visine (eng. height – h) i širine (eng. width – w); na slici prikazani zeleno. Sljedećim linijama koda definirana je funkcija kojom se obavlja konverzija YOLOv8 zapisa u DeepSORT zapis, pri čemu su izlazi redom x koordinata centralne točke, y koordinata centralne točke, širina i u konačnici visina graničnog okvira:

```
def xyxy_to_xywh(*xyxy):
    bbox_left = min([xyxy[0].item(), xyxy[2].item()])
    bbox_top = min([xyxy[1].item(), xyxy[3].item()])
    bbox_w = abs(xyxy[0].item() - xyxy[2].item())
    bbox_h = abs(xyxy[1].item() - xyxy[3].item())
    x_c = (bbox_left + bbox_w / 2)
    y_c = (bbox_top + bbox_h / 2)
    w = bbox_w
    h = bbox_h
    return x_c, y_c, w, h
```

Unutar funkcije `UI_box` odvijaju se tri vrlo važne radnje. Pozivanjem `cv2.rectangle` crtaju se granični okviri oko prepoznatog objekta, a pomoću `cv2.circle` centralna koordinata. Zatim se pozivanjem prethodno definirane funkcije `draw_border` iznad graničnog okvira ucrta zaobljeni pravokutnik, te se u njega pozivanjem `cv2.putText` ubacuje tekst koji se sastoji od

jedinstvenog identifikacijskog broja i oznake klase. Primjer za izvršenu `UI_box` funkciju može se vidjeti na Slici 39.

Slika 39. Izvršena funkcija UI_box

Treba napomenuti da je ono što može varirati unutar ove funkcije točka koja se ucrtava. Iako DeepSORT obavlja svoje praćenje na temelju centralne koordinate graničnih okvira, prebrojavanje ljudi može se izvršiti na temelju neke druge točke. Primjerice točke koja predstavlja polovište donje stranice graničnog okvira, što u nekim slučajevima ima više smisla nego praćenje centralne koordinate. U slučaju kada se prati centralna koordinata, linija koda za definiranje koordinata ucrtane točke izgleda ovako:

```
cx, cy = int(x[0]) + w // 2, int(x[1]) + h // 2
```

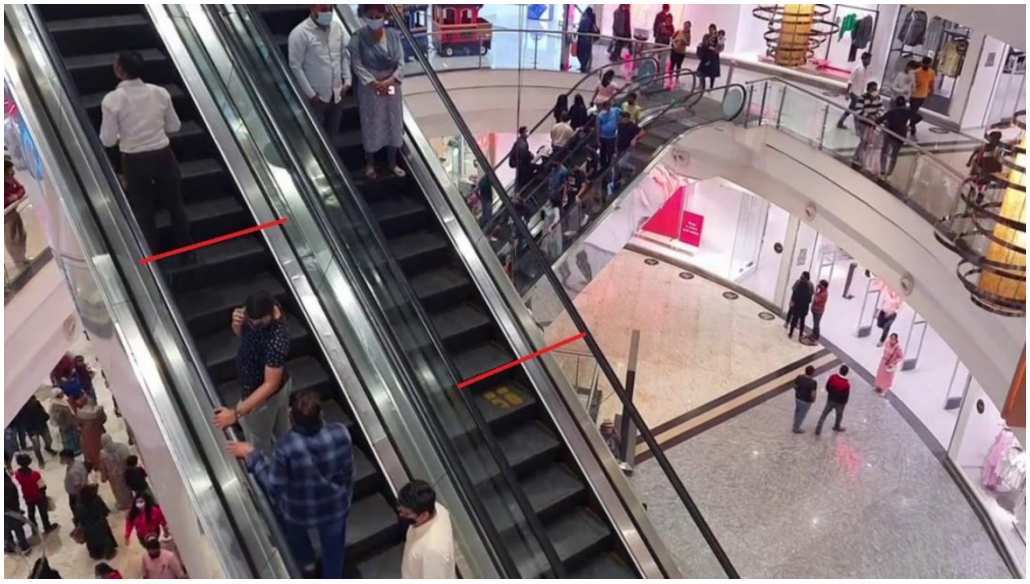
No, želi li se pratiti drugi spomenuti slučaj, koristit će se sljedeća linija koda:

```
cx, cy = int(x[0]) + w // 2, int(x[1]) + h
```

Slijedi objašnjenje sadržaja funkcije `draw_boxes`. Unutar ove funkcije izvršava se funkcija `UI_box`, te se svakom prepoznatom objektu dodjeljuje jedinstveni identifikacijski broj – identitet (ID). Izvan ove funkcije prethodno je otvoren novi rječnik (eng. dictionary) - `data_deque = {}`, u koji će identiteti biti pohranjeni. Razlog zašto se koristi rječnik, a ne lista, je taj što su rječnici prikladniji za korištenje u slučajevima kada treba brzo obaviti radnje

dodavanja i izbacivanja s oba kraja spremnika. U `data_deque` ID objekta bit će pohranjen dokle god je objekt u kadru. Nakon što je prepoznat novi objekt i dodijeljen mu je identitet, stvara se novi međuspremnik s maksimalnom duljinom 64. To znači da ako se dodaje 65. element u međuspremnik, 1. element će biti izbačen. Elementi unutar ovog međuspremnika zapravo su koordinate točke koja se prati, bila to koordinata središta graničnog okvira, ili njegovog donjeg ruba. Dakle, `data_deque` služi za praćenje kretanja i položaja prepoznatog objekta. Ako praćeni objekt nestane iz kadra, bit će jednostavno izbrisan iz rječnika, a pojavi li se novi objekt, njegov ID ubacuje se u rječnik.

Nakon što je kreiran ovaj sustav za praćenje, potrebno je pronaći način kako prebrojiti prepoznate objekte. Pozivanjem `cv2.line` crta se granična linija koja će služiti za prebrojavanje osoba koje tu istu liniju prijeđu. Već je spomenuto kako su u ovome radu implementirana dva načina prebrojavanja osoba, koja će sada biti i objašnjena. Ukoliko postoje dva zadana pravca kretanja osoba, kao što su primjerice pokretne stepenice, utoliko će biti dovoljno definirati dvije različite granične linije, na Slici 40. označene crvenom bojom.



Slika 40. Granične linije – primjer 1

U prethodno definirane liste `totalCountUp` i `totalCountDown` bit će dodani objekti koji prijeđu granične linije. U `totalCountUp` dodaju se objekti koji su prešli na Slici 40. definiranu lijevu liniju, odnosno objekti koji se kreću prema gornjem katu, a u `totalCountDown` dodaju se objekti koji se spuštaju na niži kat, odnosno prelaze desnu liniju. U poglavlju 5.2. objašnjena je funkcija intersekcije, koja se i ovdje koristi. Liniju AB u ovome slučaju predstavlja putanja

objekta, praćena unutar međuspremnika `data_deque`, a liniju CD predstavlja pojedina granična linija. Trajektorije kretanja praćenih objekata u ovome radu nisu potrebne, stoga je njihov prikaz izostavljen.

Moguć je i slučaj kada se objekti ne kreću po zadanim putanjama. U tome se slučaju definira samo jedna linija, kao na Slici 41.

Slika 41. Granične linije – primjer 2

Budući da je definirana samo jedna linija, prelaskom objekta preko linije ne može se direktno zaključiti kreće li se osoba prema gore ili prema dolje. Kako bi se riješio ovaj problem, definira se funkcija `get_direction`. Njome se provjerava je li prethodna y koordinata praćene točke veća ili manja od sljedeće. Ako je prethodna koordinata veća, može se zaključiti da se objekt kreće prema dolje, stoga mu se dodjeljuje oznaka `"Down"` iliti „dolje“, a ako je prethodna koordinata manja od sljedeće, objekt se kreće prema gore i dodjeljuje mu se oznaka `"Up"`, odnosno „gore“:

```
def get_direction(point1, point2):
    direction_str = ""

    if point1[1] > point2[1]:
        direction_str += "Down"
    elif point1[1] < point2[1]:
        direction_str += "Up"
    else:
        direction_str += ""
```

Ukoliko dođe do intersekcije praćenog objekta i definirane granične linije, utoliko se provjerava koji je smjer kretanja objekta. Kreće li se objekt prema gore, bit će dodan u listu `totalCountUp`, a kreće li se prema dolje bit će dodan u listu `totalCountDown`:

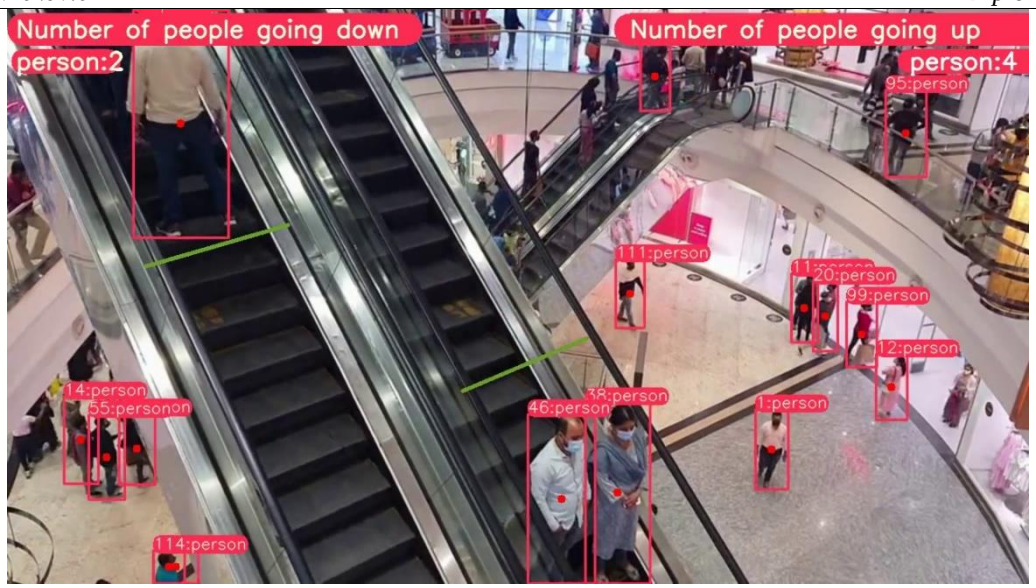
```
            if intersect(data_deque[id][0], data_deque[id][1], line[0], l
ine[1]):
                cv2.line(img, line[0], line[1], (255, 255, 255), 3)
                if "Up" in direction:
                  if obj_name not in totalCountUp:
                      totalCountUp[obj_name] = 1
                  else:
                      totalCountUp[obj_name] += 1
                if "Down" in direction:
                  if obj_name not in totalCountDown:
                      totalCountDown[obj_name] = 1
                  else:
                      totalCountDown[obj_name] += 1
          UI_box(box, img, label=label, color=color, line_thickness=2)
```

Ono što se još odvija unutar `draw_boxes` funkcije, prikaz je rezultata na zaslonu. U gornjem lijevom kutu prikazuje se broj ljudi koji su prešli liniju i kreću se prema dolje, a u gornjem desnom kutu broj ljudi koji se kreću prema gore.

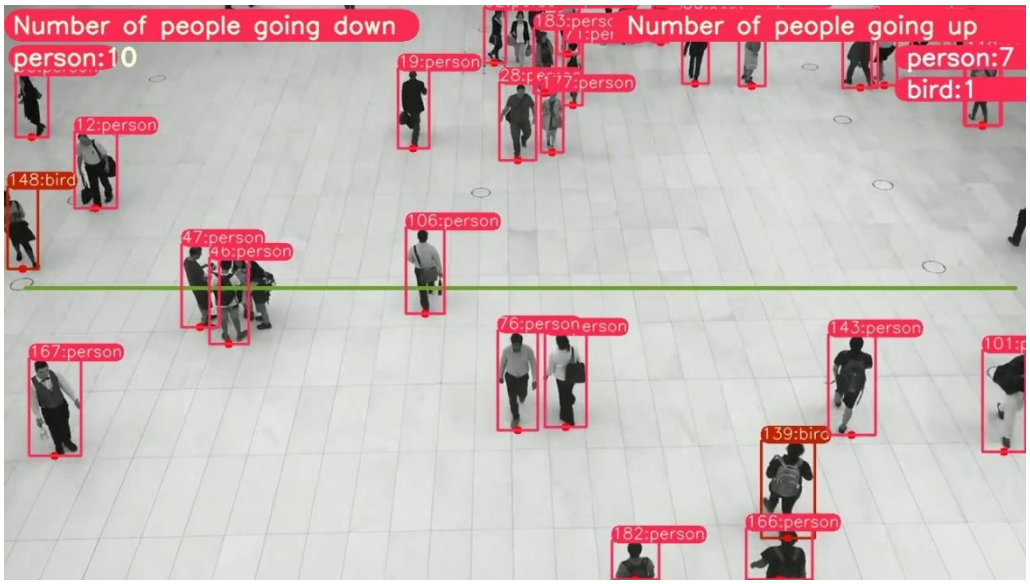
6.3.2. Rezultati

U primjeru 1 [54], koji se odnosi na video preuzet s interneta na kojemu se osobe kreću pokretnim stepenicama, sve osobe ispravno su prepoznate, te se ispravno pribrajaju broju ljudi koji se kreće prema gore, odnosno dolje.

Promotri li se drugi primjer videa ljudi koji se ne kreću po zadanoj putanji [55], uočava se da dolazi do određenih nepravilnosti u radu. Naime, većina osoba uistinu i jest prepoznata ispravno, no kod nekih objekata dolazi do pogrešaka. Na Slici 43. vidljivo je kako su čak dva objekta pogrešno prepoznata kao ptice, odnosno „bird“. Također, javlja se i problem kod grupice ljudi, pozicioniranih na lijevom središnjem predjelu slike. U grupici se nalaze četiri osobe, no samo su dvije prepoznate i dodijeljen im je identifikacijski broj. Iako na cjelokupnom videu nisu svi objekti ispravno identificirani, svaki objekt koji je prešao naznačenu graničnu liniju ispravno je pribrojen u listu, ovisno o tome kreće li se prema gore ili prema dolje.



Slika 42. Rezultati – primjer 1



Slika 43. Rezultati – primjer 2

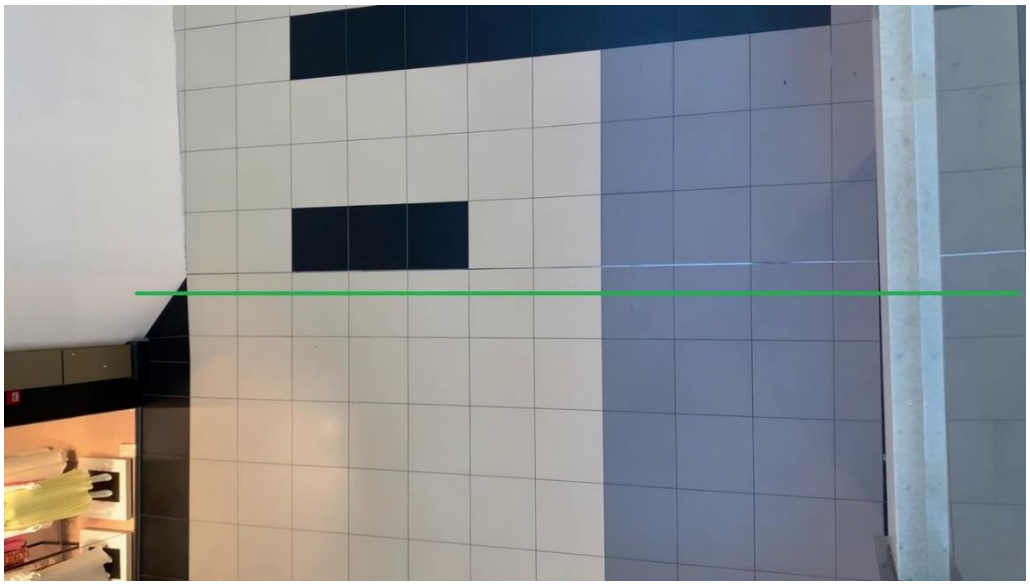
6.4. Prebrojavanje osoba – samostalno snimljeni primjeri

Za dodatne primjere primjene koda, videi u ovome poglavlju snimljeni su samostalno, koristeći dvojnu kameru mobilnog telefona iPhone XS, rezolucije 12 MP. Format snimljenih videa je 720x1280, a broj slika u sekundi 30. Video materijali snimani su na lokaciji Arena Centar.

Na Slikama 44. i 45. prikazani su kadrovi snimanja i pripadajuće granične linije. U oba je primjera odabran pristup sa samo jednom graničnom linijom za kretanje u oba smjera, čak i za

primjer na pokretnim stepenicama, jer je kut snimanja takav da bi bilo teško izvedivo postaviti granice za svaki smjer pojedinačno, a da model radi ispravno.

Slika 44. Primjer Arena Centar 1 [56]



Slika 45. Primjer Arena Centar 2 [57]

Iako je na ove videe primijenjen identičan kod kao i na videe preuzete s interneta, naravno uz izvršene potrebne promjene za koordinate graničnih linija, dobiveni rezultati izuzetno su loši.

Na kadrovima na Slici 46. može se uočiti jedna od nepravilnosti u radu modela. Jedan od objekata prepoznat je kao osoba te mu je dodijeljen identifikacijski broj 3, no granični okviri

ne obuhvaćaju cijeli objekt. Već je u sljedećem kadru taj isti objekt prepoznat tako da ga granični okviri u potpunosti obuhvaćaju, no identifikacijski broj više nije 3, nego 50. Također, nakon što je taj isti objekt prešao liniju uopće nije pribrojen na listu osoba koje se kreću prema dolje. Sličan problem na istim kadrovima može se vidjeti i s objektom identifikacijskog broja 7. U jednom kadru nije prepoznat, u sljedećem jest, no na zadnjem kadru ponovno nije prepoznat. Za vrijeme prelaženja tog istog objekta preko granične linije, objekt je ispravno pribrojen u broj osoba koje se kreću prema gore, no kada bi se linija nalazila na nekoj drugoj poziciji, velika je mogućnost da taj objekt ne bi bio pribrojen.

Slika 46. Problem – primjer Arena Centar 1

Kod tako učestale promjene graničnih okvira javlja se i pojava da jedan objekt bude pribrojen više od jednog puta, iako je samo jednom prešao graničnu liniju, upravo zato što se granični okviri povećavaju i smanjuju te tako neovisno o objektu više puta mogu preći liniju.

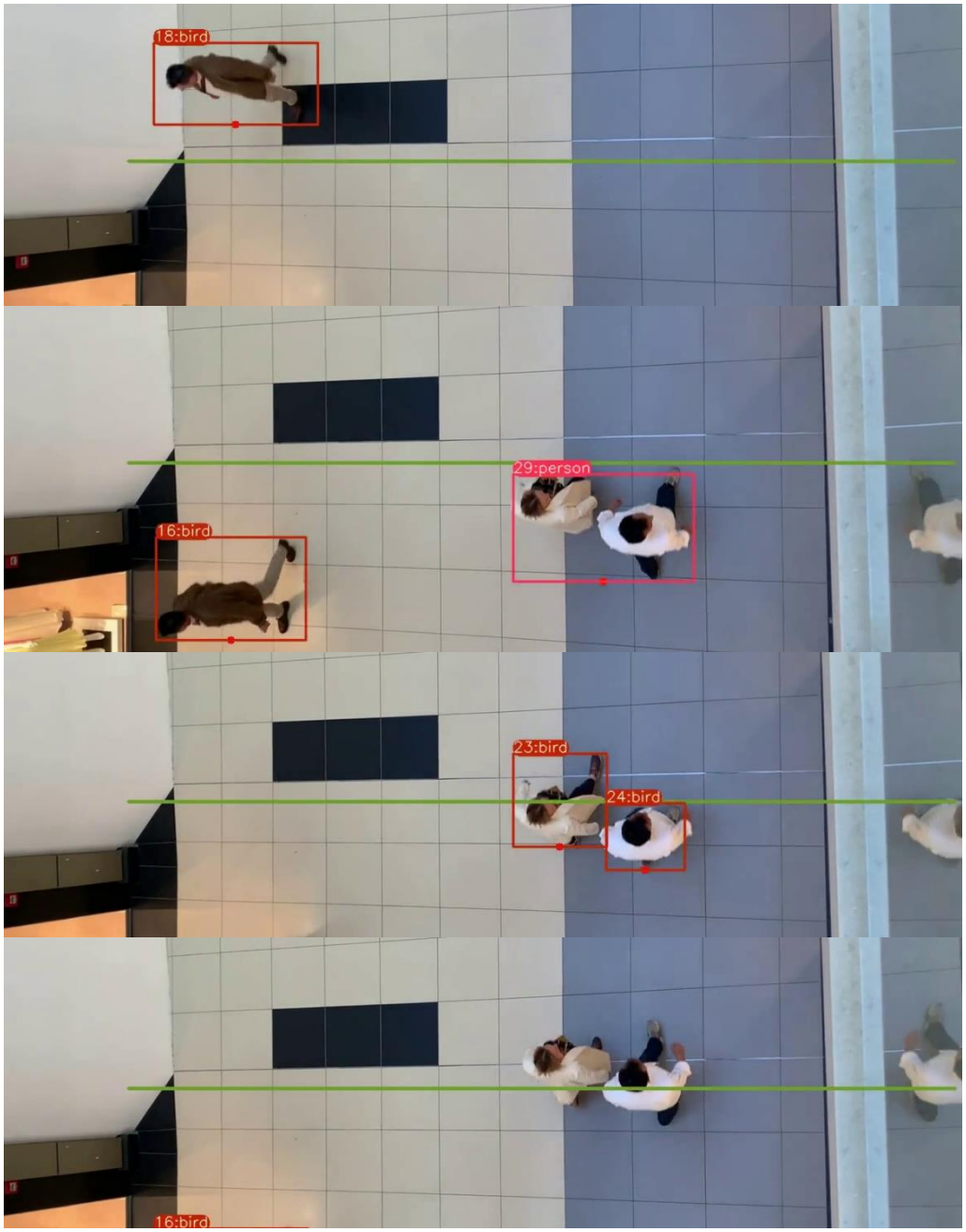
Konačan rezultat prvog samostalno snimljenog primjera prikazan je na Slici 47., a budući da rezultati samostalno snimljenih videa odstupaju od stvarnosti više nego primjeri preuzeti s interneta, pobliže će biti objašnjeni i tabličnim prikazima.

Slika 47. Rezultati – primjer Arena Centar 1

	Stvaran broj osoba	**Broj točno pribrojenih osoba**	**Broj pribrojenih objekata**
Objekti koji se kreću prema gore	4	2	3
Objekti koji se kreću prema dolje	4	2	3
Omjer broja točno pribrojenih osoba i ukupnog broja stvarnih osoba	50%		

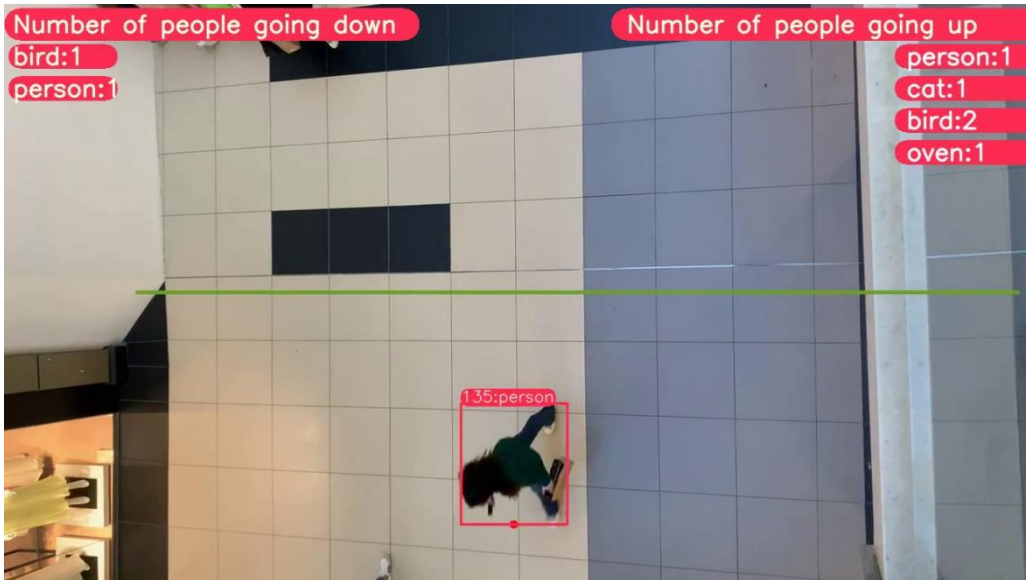
Tablica 5. Rezultati – primjer Arena Centar 1

Još lošiji rezultati ispadaju na sljedećem primjeru, kojem je položaj snimanja za neke osobe bio iznad njihovih glava. Ovaj kut snimanja pokazao se da daje iznimno loše rezultate, koji su ovim pristupom gotovo neupotrebljivi. Algoritam umjesto osoba prepoznaje ptice, mačke, pa čak i pećnice. Promotri li se niz kadrova obuhvaćenih Slikom 48., uočava se da je objekt lijevo prepoznat, ali ne kao osoba, već kao ptica. Usprkos činjenici da krivo prepoznata klasa ne utječe na pribrojavanje objekta koji prijeđe liniju, ovaj objekt nije pribrojen objektima koji se kreću prema dolje.



Slika 48. Problem – primjer Arena Centar 2

Na istim kadrovima, na desnoj polovici kadra prolaze muškarac i žena, koji su prvo zajedno prepoznati kao jedna osoba, no već u sljedećem kadru identificirani su pojedinačno kao ptice, dok prelazeći liniju njihovi granični okviri nestaju i ne bivaju pribrojeni ukupnom broju objekata koji se kreću prema gore. Konačni rezultati primjera 2 prikazani su Slikom 49. i Tablicom 6.



Slika 49. Rezultati – primjer Arena Centar 2

	Stvaran broj osoba	**Broj točno pribrojenih objekata**	**Ukupan broj pribrojenih objekata**
Objekti koji se kreću prema gore	17	5	5
Objekti koji se kreću prema dolje	6	1	2
Omjer broja točno pribrojenih osoba i ukupnog broja stvarnih osoba	8,7%		

Tablica 6. Rezultati – primjer Arena Centar 2

U Tablici 6. treći stupac naziva se „Broj točno pribrojenih objekata“, a ne „osoba“, zato što je velik broj pogrešno klasificiranih objekata, stoga je u tablicu upisan broj svih pojedinačnih objekata koji su ispravno prepoznati da su prešli graničnu liniju, bez obzira na pogrešnu klasifikaciju. U posljednjem retku tablice, omjer se odnosi samo na ispravno identificirane osobe. Svakako se može zaključiti kako kut snimanja odozgo nikako ne odgovara radu algoritma, kao ni brzo kretanje objekata.

Iako postoje osobe koje se na prvom primjeru ne kreću pokretnim stepenicama, stoga niti ne mogu biti prebrojane, algoritam bi ih neovisno o tome trebao prepoznati. Budući da velik broj pozadinskih osoba nije prepoznat, zaključuje se i kako udaljenost od kamere, odnosno relativne dimenzije promatranog objekta u odnosu na dimenzije videa, isto utječu na kvalitetu rada modela. Pokušavajući dobiti bolje rezultate, na trećem je primjeru ponovno odabrana scena pokretnih stepenica, budući da objekti ovdje imaju manju brzinu kretanja, te bi kut snimanja trebao biti pogodniji za prepoznavanje ispravne klase objekta. Položaj kamere u odnosu na stepenice ipak je nešto izmijenjen, nego što je bio u primjeru 1. Kamera je postavljena više frontalno na stepenice, što bi trebalo omogućiti jasniji prikaz osobe. Također, kako bi se isprobao i pristup s dvije granične linije umjesto jedne, dodana je i ta promjena. Istom pozicijom kamere i s istim uvjetima izrađena su dva videa, stoga će biti nazvani primjer 3a [58] i primjer 3b [59], a uvjeti snimanja i granične linije prikazane su Slikom 50.

Slika 50. Primjer 3

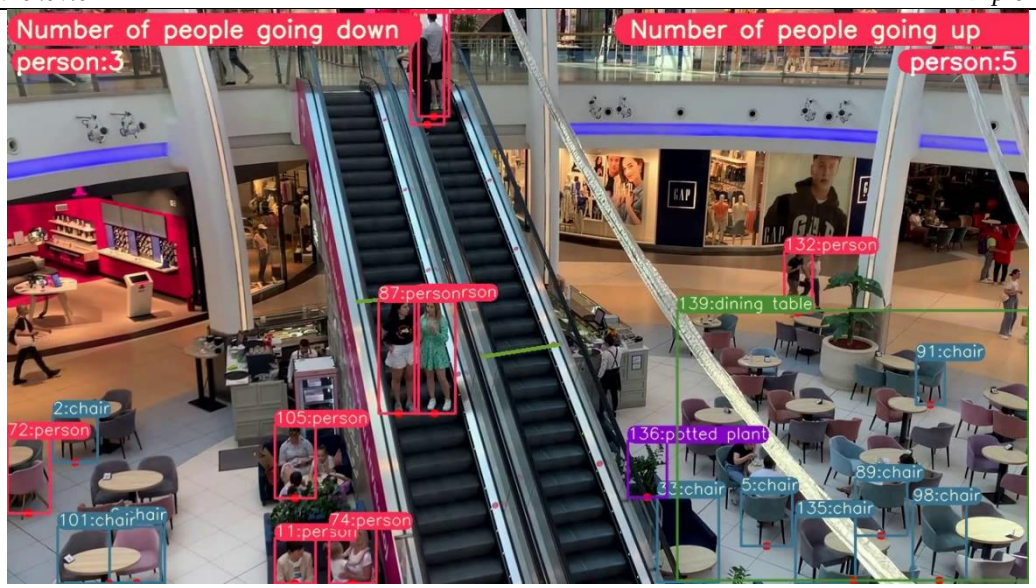
Rezultati primjera 3a i 3b nešto su bolji od primjera 1 i primjera 2, sve osobe ispravno su klasificirane, odnosno ne pojavljuju se neispravne klase u pribrojavanju. Konačni rezultati

primjera 3a prikazani su Slikom 51. i Tablicom 7., a primjera 3b Slikom 52. i Tablicom 8. Nakon rezultata slijedi opis glavnih problema koji ostaju prisutni.

Slika 51. Rezultati – primjer 3a

	Stvaran broj osoba	**Broj točno pribrojenih osoba**	**Ukupan broj pribrojenih objekata**
Objekti koji se kreću prema gore	5	1	1
Objekti koji se kreću prema dolje	7	5	5
Omjer broja točno pribrojenih osoba i ukupnog broja stvarnih osoba	50%		

Tablica 7. Rezultati – primjer 3a



Slika 52. Rezultati – primjer 3b

	Stvaran broj osoba	**Broj točno pribrojenih osoba**	**Ukupan broj pribrojenih objekata**
Objekti koji se kreću prema gore	4	4	5
Objekti koji se kreću prema dolje	3	3	3
Omjer broja točno pribrojenih osoba i ukupnog broja stvarnih osoba	100%		

Tablica 8. Rezultati – primjer 3b

Ukupan omjer točno pribrojenih osoba i ukupnog broja osoba za primjer 3 (3a i 3b zajedno) iznosi 68,42%. Rezultat je znatno bolji od rezultata dobivenih primjerima 1 i 2. Zaključuje se da kut snimanja, udaljenost kamere od objekata te brzina kretanja objekata uvelike utječu na točnost rada algoritma. Međutim to nisu jedini čimbenici. Primjerom 3 potvrđeni su određeni problemi, koji ne ovise o poziciji kamere niti o drugim uvjetima koji se mogu kontrolirati više no što je učinjeno, već ovise o samim algoritmima za prepoznavanje i praćenje objekata. Algoritam nije prilagođen za slučajeve okluzije, odnosno kada se objekti međusobno prekrivaju

i ne vide u potpunosti jasno. Problemi također nastaju i ako praćeni objekt pretjerano mijenja položaj, odnosno ako njegovo držanje nije statično. Nizom kadrova na Slici 53. prikazan je problem veće grupe ljudi.

Slika 53. Problem – primjer 3a

Iako se grupa sastoji od četvero osoba, u prvom kadru ispravno je prepoznata samo jedna osoba identifikacijskog broja 23, a identifikacijski broj 20 odnosi se na dvije osobe, stoga nije ispravna detekcija. Već u sljedećem kadru nije prepoznata nijedna osoba, a u posljednjem kadru prepoznata je jedna osoba, ona koja u prvome kadru uopće nije bila detektirana. Nakon što je cijela grupa ljudi prešla graničnu liniju, nije bio pribrojen niti jedan objekt.

Slika 54. Problem – primjer 3b

Nekonzistentnost graničnih okvira može stvoriti dodatne probleme. Prvi problem bio bi prelazak istog identiteta s jednog objekta na drugi, a drugi problem višestruko je pribrojavanje istog objekta. Ovaj je problem već ranije spomenut, a oba problema sada su i predočena kadrovima na Slici 54. Naime, u prvome kadru identifikacijski broj 69 dodijeljen je ženskoj

osobi, a u sljedećem kadru isti identifikacijski broj prelazi na mušku osobu, što bi trebalo označavati kako je to jedan te isti objekt, a očigledno nije. U sljedećem kadru prepoznata je i ženska osoba, no identifikacijski su brojevi zamijenjeni u odnosu na prvi kadar. Kako se dimenzije graničnih okvira iz kadra u kadar mijenjaju, javlja se slučaj da je muškarac pribrojen dva puta, što se vidi u dva posljednja kadra slike. Granična linija postane bijela ukoliko dođe do intersekcije praćene točke i linije, a za isti praćeni objekt ta je linija dva puta postala bijela, odnosno pribrojan je jednom previše.

Najbolji i najtočniji rezultati u ovim uvjetima dobiju se ukoliko su objekti statični i potpuno vidljivi bez međusobnih preklapanja, kao na primjerima na Slici 55.

Slika 55. Idealni položaj objekata

6.5. Objašnjenje rezultata i moguća poboljšanja

Kroz rezultate uočeno je da uvjeti snimanja i uvjeti okruženja utječu na rad YOLOv8 algoritma, odnosno neprikladan kut snimanja, loše osvjetljenje, okluzija, kompleksne pozadine i slični uvjeti nepovoljno utječu na rezultate modela. No, problemi mogu proizlaziti i iz drugih faktora. Primjerice model može biti treniran na premalom skupu podataka ili skupu podataka koji ne sadržava dovoljno varijacija u prikazu objekata. U fazi treniranja modela može se dogoditi i neadekvatno vrijeme treniranja, ali i podtreniranost odnosno pretreniranost. Pretreniranost se pojavljuje kada model postaje previše specijaliziran za skup podataka za treniranje, zbog čega ima poteškoće u prepoznavanju varijacija modela. S druge strane, podtreniranost se odnosi na nemogućnost modela da prepozna kompleksnije objekte. Također tijekom treniranja, ako se ne bilježe ispravne koordinate graničnih okvira, model može krivo naučiti lokalizirati objekte, što će također utjecati na točnost modela. Nadalje, problem može nastati u samoj arhitekturi i kapacitetu modela. YOLOv8 relativno je manji model od svojih prethodnika, kao što su

primjerice YOLOv4 ili YOLOv5. Hiperparametri kao što je primjerice veličina serije, ako su neprimjereno definirani također mogu dovesti do lošijih rezultata.

Ono na što se može utjecati dodatno, a da ne mijenja strukturu YOLOv8, bilo bi uvođenje većeg i raznolikijeg skupa podataka za trening modela, podešavanje hiperparametara, prikladno vrijeme treniranja, osiguranje zapisa ispravnih bilješki o lokaciji graničnih okvira te evaluacija rada modela na uvjetima iz stvarnog svijeta. Iako je YOLOv8 jedan od vodećih detektora objekata, a DeepSORT vodeći alat za praćenje, može se dogoditi da njihov rad ne zadovoljava prethodno postavljene kriterije. U tome slučaju potrebno je pokušati maksimalno prilagoditi uvjete snimanja i okoliša kako bi objekti bili što lakše uočljiviji. Ako model i dalje ne daje zadovoljavajuće rezultate, treba dobro razmisliti o primjeni nekih drugih modela na zadani problem. Dvofazni modeli točniji su od jednofaznih, stoga treba procijeniti koliko je za zadani problem bitna primjena u stvarnom vremenu, a koliko je bitna točnost rada modela.

7. ZAKLJUČAK

Računalni vid grana je umjetne inteligencije koju je bitno razvijati zbog svojih brojnih primjena, kako u strojarstvu i tehničkim područjima, tako i u medicini, ali i svakodnevnom životu. Za sigurnosni nadzor, analizu prometa, analizu ponašanja ljudi, prebrojavanja na slikama i videozapisima, ali i za druge slične zadatke vrlo važnu ulogu ima prepoznavanje ili detekcija objekata. Detektori objekata podijeljeni su na dvofazne i jednofazne. Dvofazni objekti provode postupak prepoznavanja objekata u dva koraka, prvi korak odnosi se na generiranje interesnih područja na slikama, a drugi korak na samo prepoznavanje i generiranje graničnih okvira. Jednofazni detektori, s druge strane, ne stvaraju interesna područja na slikama i videima, već direktno iz jednog pogleda mogu dati predviđanja za klase i granične okvire prepoznatih objekata. Arhitektura dvofaznih detektora složenija je nego kod jednofaznih, što ih čini moćnijim alatima, odnosno njima se postižu bolje točnosti i preciznosti prepoznavanja objekata. Istovremeno ih tako složena arhitektura čini i sporijima, stoga u pogledu brzine izvršavanja zadataka prednost imaju jednofazni detektori. Najpopularniji jednofazni detektor svakako je YOLO algoritam čiji je prvi model predstavljen 2016. godine. Od tada, ovaj model više je puta unaprjeđivan kroz razne verzije, a najnovija verzija, izdana u siječnju 2023. godine, je YOLOv8. Ova je verzija vrlo jednostavna i dostupna za korištenje, za razliku od prethodnih modela ne koristi usidrene okvire, omogućujući tako zadatcima klasifikacije, lokalizacije i objektivnosti da postižu bolje rezultate. Verzija sagledava objekte u kontekstu slike, te se pomoću svojstva samopažnje određuje važnost pojedinih obilježja objekata na slici. U fazi treniranja, koriste se i razne tehnike kojima se povećava skup podataka za trening i pokušava smanjiti problem okluzije.

YOLOv8 model za prepoznavanje objekata i DeepSORT model za praćenje objekata u ovome su radu primijenjeni na videima ljudi koji se kreću u jednom smjeru po zadanim putanjama ili po zajedničkoj putanji u više smjerova. Na videima su određene granične linije, kada ih osoba prijeđe, trebala bi biti pribrojana u pripadajuće liste ovisno o smjeru kretanja. Korištena su dva videa preuzeta s interneta, te su na njima rezultati bili vrlo dobri. Svi objekti koji su prešli granične linije su pribrojani, no postoje manje greške u klasifikaciji objekata, te greške uzrokovane okluzijom osoba. Kod je primijenjen i na samostalno snimljene videe, ali na ovim videima rezultati su bili znatno lošiji. Primjenom nekoliko različitih perspektiva gledanja utvrđeno je kako YOLO algoritam nije neosjetljiv na položaj kamere u odnosu na objekte. Iako je to vrlo napredan algoritam, još uvijek je potrebno da objekti budu vidljivi u cijelosti, da budu relativno statični i da okluzija objekata bude svedena na minimum. U suprotnom rezultati

detekcije mogu biti nezadovoljavajući. Nakon što su pojedini faktori modificirani, na neke je ipak teže utjecati. Kako bi se u budućnosti unaprijedila preciznost detekcije, moguće je provesti nekoliko poboljšanja. U fazi treniranja potrebno je pronaći adekvatno vrijeme treniranja modela te model opskrbiti dovoljno velikim skupom podataka koji sadrži razne varijante prikaza položaja i kretnji osoba u različitim uvjetima. Također, moguće je unaprjeđenje izmjenom hiperparametara u samoj arhitekturi modela. Međutim, ako su rezultati i dalje nezadovoljavajući, potrebno je uzeti u obzir primjenu dvofaznih detektora, ovisno o potrebnoj točnosti, ali i brzini primjene.

LITERATURA

[1] Ö. Y. A. U. and Y. D. M. Coşkun, "'AN OVERVIEW OF POPULAR DEEP LEARNING METHODS,'" *European Journal of Technique (EJT)*, vol. 7, no. 2, pp. 165–176, Dec. 2017.

[2] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep Learning for Computer Vision: A Brief Review," *Comput Intell Neurosci*, vol. 2018, p. 7068349, 2018, doi: 10.1155/2018/7068349.

[3] J. Brownlee, "How Do Convolutional Layers Work in Deep Learning Neural Networks?," *Machine Learning Mastery*, Apr. 17, 2020.

[4] D. Bhatt *et al.*, "CNN Variants for Computer Vision: History, Architecture, Application, Challenges and Future Scope," *Electronics (Basel)*, vol. 10, no. 20, 2021, doi: 10.3390/electronics10202470.

[5] T. Huff, N. Mahabadi, and P. Tadi, *Neuroanatomy, Visual Cortex*. 2023.

[6] "https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7".

[7] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, IEEE, Aug. 2017, pp. 1–6. doi: 10.1109/ICEngTechnol.2017.8308186.

[8] "https://tikz.net/conv2d/".

[9] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation Functions: Comparison of trends in Practice and Research for Deep Learning," Nov. 2018.

[10] A. F. Agarap, "Deep Learning using Rectified Linear Units (ReLU)," Mar. 2018.

[11] Mehreen Saeed, "An introducion to recurrent neural networks and the math that powers them," *machinelearningmastery.com* , Jan. 06, 2023.

[12] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," Mar. 2020.

[13] S. Xu, J. Wang, W. Shou, T. Ngo, A.-M. Sadick, and X. Wang, "Computer Vision Techniques in Construction: A Critical Review," *Archives of Computational Methods in Engineering*, vol. 28, no. 5, pp. 3383–3397, 2021, doi: 10.1007/s11831-020-09504-3.

[14] I. Mihajlović, "Everything You Ever Wanted To Know About Computer Vision.," *Towards Data Science*, Apr. 25, 2019.

[15] T. Diwan, G. Anirudh, and J. V Tembhurne, "Object detection using YOLO: challenges, architectural successors, datasets and applications," *Multimed Tools Appl*, vol. 82, no. 6, pp. 9243–9275, 2023, doi: 10.1007/s11042-022-13644-y.

[16] J. Brownlee, "A Gentle Introduction to Object Recognition With Deep Learning," *Deep Learning for Computer Vision*, May 22, 2019.

[17] "https://ambolt.io/en/image-classification-and-object-detection/".

[18] "https://medium.com/analytics-vidhya/image-classification-vs-object-detection-vs-image-segmentation-f36db85fe81".

[19] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," Nov. 2013.

[20] U. Farooq, "From R-CNN to Mask R-CNN," Feb. 15, 2018.

[21] L. Weng, "Object Detection for Dummies Part 3: R-CNN Family," Dec. 31, 2017.

[22] Rohith Gandhi, "R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms," *Towards Data Science*, Jul. 09, 2018.

[23] R. Girshick, "Fast R-CNN," Apr. 2015.

[24] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," Jun. 2015.

[25] S.-H. Tsang, "Review: R-FCN — Positive-Sensitive Score Maps (Object Detection)," *Towards Data Science*, Oct. 20, 2018.
[26] L. Du, R. Zhang, and X. Wang, "Overview of two-stage object detection algorithms," *J Phys Conf Ser*, vol. 1544, no. 1, p. 012033, May 2020, doi: 10.1088/1742-6596/1544/1/012033.
[27] Z. Cai and N. Vasconcelos, "Cascade R-CNN: Delving into High Quality Object Detection," Dec. 2017.
[28] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, "Object Detection With Deep Learning: A Review," *IEEE Trans Neural Netw Learn Syst*, vol. 30, no. 11, pp. 3212–3232, 2019, doi: 10.1109/TNNLS.2018.2876865.
[29] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature Pyramid Networks for Object Detection," Dec. 2016.
[30] W. Liu *et al.*, "SSD: Single Shot MultiBox Detector," Dec. 2015, doi: 10.1007/978-3-319-46448-0_2.
[31] developers.arcgis.com, "How single-shot detector (SSD) works?"
[32] J. Hui, "SSD object detection: Single Shot MultiBox Detector for real-time processing," Mar. 14, 2018.
[33] "https://iq.opengenus.org/single-shot-detection-ssd-algorithm/".
[34] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788. doi: 10.1109/CVPR.2016.91.
[35] R. Kundu, "YOLO: Algorithm for Object Detection Explained [+Examples]," Jan. 17, 2023.
[36] P. Fränti and R. Mariescu-Istodor, "Soft precision and recall," *Pattern Recognit Lett*, vol. 167, pp. 115–121, 2023, doi: https://doi.org/10.1016/j.patrec.2023.02.005.
[37] "https://devopedia.org/imagenet".
[38] "https://www.researchgate.net/figure/Samples-of-annotated-images-in-the-MS-COCO-dataset-from-12_fig1_334867305".
[39] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," Apr. 2020.
[40] S. Trivedi, "Understanding Attention Modules: CBAM and BAM — A Quick Read," *VisionWizard*, Jun. 12, 2020.
[41] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, "A Review of Yolo Algorithm Developments," *Procedia Comput Sci*, vol. 199, pp. 1066–1073, 2022, doi: https://doi.org/10.1016/j.procs.2022.01.135.
[42] "https://ultralytics.com/."
[43] D. Reis, J. Kupec, J. Hong, and A. Daoudi, "Real-Time Flying Object Detection with YOLOv8," May 2023.
[44] C. Li *et al.*, "YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications," Sep. 2022.
[45] J. Terven and D. Cordova-Esparza, "A Comprehensive Review of YOLO: From YOLOv1 and Beyond," Apr. 2023.
[46] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," Jul. 2022.
[47] J. Solawetz, "What is YOLOv8? The Ultimate Guide.," *blog.roboflow.com*, Jan. 11, 2023.
[48] A. Mehra, "Understanding YOLOv8 Architecture, Applications & Features," *www.labellerr.com*, Apr. 16, 2023.
[49] Sanyam, "Understanding Multiple Object Tracking using DeepSORT," *https://learnopencv.com/*, Jun. 21, 2022.

[50] "https://medium.com/@mohit_gaikwad/deep-sort-simple-online-and-real-time-tracking-with-deep-associative-metric-94138d528ff1".
[51] M. I. H. Azhar, F. H. K. Zaman, N. M. Tahir, and H. Hashim, "People Tracking System Using DeepSORT," in *2020 10th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, 2020, pp. 137–141. doi: 10.1109/ICCSCE50387.2020.9204956.
[52] MuhammadMoinFaisal, "YOLOv8-DeepSORT-Object-Tracking," *https://github.com*, Jan. 25, 2023.
[53] M. Moin, "YOLOv8: Object Detection, Tracking & Web App in Python 2023," *www.udemy.com*, 2023.
[54] "https://drive.google.com/uc?id=1LLpEPJC2vS8HDxwgu61DqyeA1XyydS3u."
[55] "https://www.youtube.com/watch?v=ORrrKXGx2SE&ab_channel=MuhammedJamnas."
[56] "https://youtu.be/c3fc24y1dLA."
[57] "https://youtu.be/zEX0NjtP6b0."
[58] "https://youtu.be/XdY5JZJUvgs."
[59] "https://youtu.be/NW3UR7Mzm-I."

PRILOZI

I. predict.py skripta za primjer 1 – video preuzet s interneta

II. predict.py skripta za primjer 2 – video preuzet s interneta

I predict.py skripta za primjer 1 – video preuzet s interneta

```
import hydra
import torch
import argparse
import time
from pathlib import Path

import cv2
import torch
import torch.backends.cudnn as cudnn
from numpy import random
from ultralytics.yolo.engine.predictor import BasePredictor
from ultralytics.yolo.utils import DEFAULT_CONFIG, ROOT, ops
from ultralytics.yolo.utils.checks import check_imgsz
from ultralytics.yolo.utils.plotting import Annotator, colors, save_one
_box

import cv2
from deep_sort_pytorch.utils.parser import get_config
from deep_sort_pytorch.deep_sort import DeepSort
from collections import deque
import numpy as np

palette = (2 ** 11 - 1, 2 ** 15 - 1, 2 ** 20 - 1)
data_deque = {}

deepsort = None

totalCountUp = {}
totalCountDown = {}

limitsUp = [(173, 322), (352, 270)]
limitsDown = [(570, 478), (726, 412)]

def init_tracker():
    global deepsort
    cfg_deep = get_config()
    cfg_deep.merge_from_file("deep_sort_pytorch/configs/deep_sort.yaml"
)

    deepsort= DeepSort(cfg_deep.DEEPSORT.REID_CKPT,
                           max_dist=cfg_deep.DEEPSORT.MAX_DIST, min_co
nfidence=cfg_deep.DEEPSORT.MIN_CONFIDENCE,
                           nms_max_overlap=cfg_deep.DEEPSORT.NMS_MAX_O
VERLAP, max_iou_distance=cfg_deep.DEEPSORT.MAX_IOU_DISTANCE,
                           max_age=cfg_deep.DEEPSORT.MAX_AGE, n_init=c
fg_deep.DEEPSORT.N_INIT, nn_budget=cfg_deep.DEEPSORT.NN_BUDGET,
                           use_cuda=True)
```

```
##########################################################################
###################
def xyxy_to_xywh(*xyxy):
    bbox_left = min([xyxy[0].item(), xyxy[2].item()])
    bbox_top = min([xyxy[1].item(), xyxy[3].item()])
    bbox_w = abs(xyxy[0].item() - xyxy[2].item())
    bbox_h = abs(xyxy[1].item() - xyxy[3].item())
    x_c = (bbox_left + bbox_w / 2)
    y_c = (bbox_top + bbox_h / 2)
    w = bbox_w
    h = bbox_h
    return x_c, y_c, w, h


def compute_color_for_labels(label):
    if label == 0: #person
        color = (85,45,255)
    else:
        color = [int((p * (label ** 2 - label + 1)) % 255) for p in pal
ette]
    return tuple(color)

def draw_border(img, pt1, pt2, color, thickness, r, d):
    x1,y1 = pt1
    x2,y2 = pt2
    # Top left
    cv2.line(img, (x1 + r, y1), (x1 + r + d, y1), color, thickness)
    cv2.line(img, (x1, y1 + r), (x1, y1 + r + d), color, thickness)
    cv2.ellipse(img, (x1 + r, y1 + r), (r, r), 180, 0, 90, color, thick
ness)
    # Top right
    cv2.line(img, (x2 - r, y1), (x2 - r - d, y1), color, thickness)
    cv2.line(img, (x2, y1 + r), (x2, y1 + r + d), color, thickness)
    cv2.ellipse(img, (x2 - r, y1 + r), (r, r), 270, 0, 90, color, thick
ness)
    # Bottom left
    cv2.line(img, (x1 + r, y2), (x1 + r + d, y2), color, thickness)
    cv2.line(img, (x1, y2 - r), (x1, y2 - r - d), color, thickness)
    cv2.ellipse(img, (x1 + r, y2 - r), (r, r), 90, 0, 90, color, thickn
ess)
    # Bottom right
    cv2.line(img, (x2 - r, y2), (x2 - r - d, y2), color, thickness)
    cv2.line(img, (x2, y2 - r), (x2, y2 - r - d), color, thickness)
    cv2.ellipse(img, (x2 - r, y2 - r), (r, r), 0, 0, 90, color, thickne
ss)

    cv2.rectangle(img, (x1 + r, y1), (x2 - r, y2), color, -
1, cv2.LINE_AA)
```

```
    cv2.rectangle(img, (x1, y1 + r), (x2, y2 - r - d), color, -
1, cv2.LINE_AA)

    cv2.circle(img, (x1 +r, y1+r), 2, color, 12)
    cv2.circle(img, (x2 -r, y1+r), 2, color, 12)
    cv2.circle(img, (x1 +r, y2-r), 2, color, 12)
    cv2.circle(img, (x2 -r, y2-r), 2, color, 12)

    return img

def UI_box(x, img, color=None, label=None, line_thickness=None):
    tl = line_thickness or round(0.002 * (img.shape[0] + img.shape[1])
/ 2) + 1
    color = color or [random.randint(0, 255) for _ in range(3)]
    c1, c2 = (int(x[0]), int(x[1])), (int(x[2]), int(x[3]))
    w, h = int(x[2]) - int(x[0]), int(x[3]) - int(x[1])
    cx, cy = int(x[0]) + w // 2, int(x[1]) + h // 2
    cv2.circle(img, (cx, cy), 5, (0,0,255), cv2.FILLED)

    cv2.rectangle(img, c1, c2, color, thickness=tl, lineType=cv2.LINE_A
A)
    if label:
        tf = max(tl - 1, 1)
        t_size = cv2.getTextSize(label, 0, fontScale=tl / 3, thickness=
tf)[0]

        img = draw_border(img, (c1[0], c1[1] - t_size[1] -
3), (c1[0] + t_size[0], c1[1]+3), color, 1, 8, 2)

        cv2.putText(img, label, (c1[0], c1[1] - 2), 0, tl / 3, [225, 25
5, 255], thickness=tf, lineType=cv2.LINE_AA)

def intersect(A,B,C,D):
    return ccw(A,C,D) != ccw(B,C,D) and ccw(A,B,C) != ccw(A,B,D)

def ccw(A,B,C):
    return (C[1]-A[1]) * (B[0]-A[0]) > (B[1]-A[1]) * (C[0]-A[0])

def draw_boxes(img, bbox, names,object_id, identities=None, offset=(0,
0)):
    cv2.line(img, limitsUp[0], limitsUp[1], (46,162,112), 3)
    cv2.line(img, limitsDown[0], limitsDown[1], (46,162,112), 3)

    height, width, _ = img.shape
    # remove tracked point from buffer if object is lost
    for key in list(data_deque):
      if key not in identities:
        data_deque.pop(key)
```

```
    for i, box in enumerate(bbox):
        x1, y1, x2, y2 = [int(i) for i in box]
        x1 += offset[0]
        x2 += offset[0]
        y1 += offset[1]
        y2 += offset[1]

        # code to find center of the box
        center = (int((x2+x1)/ 2), int((y1+y2)/2))

        # get ID of object
        id = int(identities[i]) if identities is not None else 0

        # create new buffer for new object
        if id not in data_deque:
          data_deque[id] = deque(maxlen= 64)
        color = compute_color_for_labels(object_id[i])
        obj_name = names[object_id[i]]
        label = '{}{:d}'.format("", id) + ":"+ '%s' % (obj_name)

        # add center to buffer
        data_deque[id].appendleft(center)
        if len(data_deque[id]) >= 2:
          if intersect(data_deque[id][0], data_deque[id][1], limitsUp[0
], limitsUp[1]):
              cv2.line(img, limitsUp[0], limitsUp[1], (255, 255, 255),
3)

              if obj_name not in totalCountUp:
                   totalCountUp[obj_name] = 1
              else:
                   totalCountUp[obj_name] += 1

          if intersect(data_deque[id][0], data_deque[id][1], limitsDown
[0], limitsDown[1]):
              cv2.line(img, limitsDown[0], limitsDown[1], (255, 255, 25
5), 3)

              if obj_name not in totalCountDown:
                   totalCountDown[obj_name] = 1
              else:
                   totalCountDown[obj_name] += 1

        UI_box(box, img, label=label, color=color, line_thickness=2)


    # Display Count
        for idx, (key, value) in enumerate(totalCountUp.items()):
            cnt_str = str(key) + ":" +str(value)
```

```
            cv2.line(img, (width - 500,25), (width,25), [85,45,255], 40
)
            cv2.putText(img, f'Number of people going up', (width - 500
, 35), 0, 1, [225, 255, 255], thickness=2, lineType=cv2.LINE_AA)
            cv2.line(img, (width - 150, 65 + (idx*40)), (width, 65 + (i
dx*40)), [85, 45, 255], 30)
            cv2.putText(img, cnt_str, (width - 150, 75 + (idx*40)), 0,
1, [255, 255, 255], thickness = 2, lineType = cv2.LINE_AA)

        for idx, (key, value) in enumerate(totalCountDown.items()):
            cnt_str1 = str(key) + ":" +str(value)
            cv2.line(img, (20,25), (500,25), [85,45,255], 40)
            cv2.putText(img, f'Number of people going down', (11, 35),
0, 1, [225, 255, 255], thickness=2, lineType=cv2.LINE_AA)
            cv2.line(img, (20,65+ (idx*40)), (127,65+ (idx*40)), [85,45
,255], 30)
            cv2.putText(img, cnt_str1, (11, 75+ (idx*40)), 0, 1, [225,
255, 255], thickness=2, lineType=cv2.LINE_AA)


    return img

class DetectionPredictor(BasePredictor):

    def get_annotator(self, img):
        return Annotator(img, line_width=self.args.line_thickness, exam
ple=str(self.model.names))

    def preprocess(self, img):
        img = torch.from_numpy(img).to(self.model.device)
        img = img.half() if self.model.fp16 else img.float()  # uint8 t
o fp16/32
        img /= 255  # 0 - 255 to 0.0 - 1.0
        return img

    def postprocess(self, preds, img, orig_img):
        preds = ops.non_max_suppression(preds,
                                        self.args.conf,
                                        self.args.iou,
                                        agnostic=self.args.agnostic_nms
,
                                        max_det=self.args.max_det)

        for i, pred in enumerate(preds):
            shape = orig_img[i].shape if self.webcam else orig_img.shap
e
            pred[:, :4] = ops.scale_boxes(img.shape[2:], pred[:, :4], s
hape).round()
```

```
        return preds

    def write_results(self, idx, preds, batch):
        p, im, im0 = batch
        all_outputs = []
        log_string = ""
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim
        self.seen += 1
        im0 = im0.copy()
        if self.webcam:  # batch_size >= 1
            log_string += f'{idx}: '
            frame = self.dataset.count
        else:
            frame = getattr(self.dataset, 'frame', 0)

        self.data_path = p
        save_path = str(self.save_dir / p.name)  # im.jpg
        self.txt_path = str(self.save_dir / 'labels' / p.stem) + ('' if
 self.dataset.mode == 'image' else f'_{frame}')
        log_string += '%gx%g ' % im.shape[2:]  # print string
        self.annotator = self.get_annotator(im0)

        det = preds[idx]
        all_outputs.append(det)
        if len(det) == 0:
            return log_string
        for c in det[:, 5].unique():
            n = (det[:, 5] == c).sum()  # detections per class
            log_string += f"{n} {self.model.names[int(c)]}{'s' * (n > 1
)}, "
        # write
        gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gai
n whwh
        xywh_bboxs = []
        confs = []
        oids = []
        outputs = []
        for *xyxy, conf, cls in reversed(det):
            x_c, y_c, bbox_w, bbox_h = xyxy_to_xywh(*xyxy)
            xywh_obj = [x_c, y_c, bbox_w, bbox_h]
            xywh_bboxs.append(xywh_obj)
            confs.append([conf.item()])
            oids.append(int(cls))
        xywhs = torch.Tensor(xywh_bboxs)
        confss = torch.Tensor(confs)

        outputs = deepsort.update(xywhs, confss, oids, im0)
        if len(outputs) > 0:
```

```
            bbox_xyxy = outputs[:, :4]
            identities = outputs[:, -2]
            object_id = outputs[:, -1]

            draw_boxes(im0, bbox_xyxy, self.model.names, object_id,iden
tities)

        return log_string

@hydra.main(version_base=None, config_path=str(DEFAULT_CONFIG.parent),
config_name=DEFAULT_CONFIG.name)
def predict(cfg):
    init_tracker()
    cfg.model = cfg.model or "yolov8n.pt"
    cfg.imgsz = check_imgsz(cfg.imgsz, min_dim=2)  # check image size
    cfg.source = cfg.source if cfg.source is not None else ROOT / "asse
ts"
    predictor = DetectionPredictor(cfg)
    predictor()

if __name__ == "__main__":
    predict()
```

II predict.py skripta za primjer 2 – video preuzet s interneta

```
import hydra
import torch
import argparse
import time
from pathlib import Path

import cv2
import torch
import torch.backends.cudnn as cudnn
from numpy import random
from ultralytics.yolo.engine.predictor import BasePredictor
from ultralytics.yolo.utils import DEFAULT_CONFIG, ROOT, ops
from ultralytics.yolo.utils.checks import check_imgsz
from ultralytics.yolo.utils.plotting import Annotator, colors, save_one
_box

import cv2
from deep_sort_pytorch.utils.parser import get_config
from deep_sort_pytorch.deep_sort import DeepSort
from collections import deque
import numpy as np

palette = (2 ** 11 - 1, 2 ** 15 - 1, 2 ** 20 - 1)
data_deque = {}

deepsort = None

totalCountUp = {}
totalCountDown = {}

line = [(26, 354), (1267, 354)]

def init_tracker():
    global deepsort
    cfg_deep = get_config()
    cfg_deep.merge_from_file("deep_sort_pytorch/configs/deep_sort.yaml"
)

    deepsort= DeepSort(cfg_deep.DEEPSORT.REID_CKPT,
                           max_dist=cfg_deep.DEEPSORT.MAX_DIST, min_co
nfidence=cfg_deep.DEEPSORT.MIN_CONFIDENCE,
                           nms_max_overlap=cfg_deep.DEEPSORT.NMS_MAX_O
VERLAP, max_iou_distance=cfg_deep.DEEPSORT.MAX_IOU_DISTANCE,
                           max_age=cfg_deep.DEEPSORT.MAX_AGE, n_init=c
fg_deep.DEEPSORT.N_INIT, nn_budget=cfg_deep.DEEPSORT.NN_BUDGET,
                           use_cuda=True)
```

```
##########################################################################
###################
def xyxy_to_xywh(*xyxy):
    bbox_left = min([xyxy[0].item(), xyxy[2].item()])
    bbox_top = min([xyxy[1].item(), xyxy[3].item()])
    bbox_w = abs(xyxy[0].item() - xyxy[2].item())
    bbox_h = abs(xyxy[1].item() - xyxy[3].item())
    x_c = (bbox_left + bbox_w / 2)
    y_c = (bbox_top + bbox_h / 2)
    w = bbox_w
    h = bbox_h
    return x_c, y_c, w, h

def compute_color_for_labels(label):
    if label == 0: #person
        color = (85,45,255)
    else:
        color = [int((p * (label ** 2 - label + 1)) % 255) for p in pal
ette]
    return tuple(color)

def draw_border(img, pt1, pt2, color, thickness, r, d):
    x1,y1 = pt1
    x2,y2 = pt2
    # Top left
    cv2.line(img, (x1 + r, y1), (x1 + r + d, y1), color, thickness)
    cv2.line(img, (x1, y1 + r), (x1, y1 + r + d), color, thickness)
    cv2.ellipse(img, (x1 + r, y1 + r), (r, r), 180, 0, 90, color, thick
ness)
    # Top right
    cv2.line(img, (x2 - r, y1), (x2 - r - d, y1), color, thickness)
    cv2.line(img, (x2, y1 + r), (x2, y1 + r + d), color, thickness)
    cv2.ellipse(img, (x2 - r, y1 + r), (r, r), 270, 0, 90, color, thick
ness)
    # Bottom left
    cv2.line(img, (x1 + r, y2), (x1 + r + d, y2), color, thickness)
    cv2.line(img, (x1, y2 - r), (x1, y2 - r - d), color, thickness)
    cv2.ellipse(img, (x1 + r, y2 - r), (r, r), 90, 0, 90, color, thickn
ess)
    # Bottom right
    cv2.line(img, (x2 - r, y2), (x2 - r - d, y2), color, thickness)
    cv2.line(img, (x2, y2 - r), (x2, y2 - r - d), color, thickness)
    cv2.ellipse(img, (x2 - r, y2 - r), (r, r), 0, 0, 90, color, thickne
ss)

    cv2.rectangle(img, (x1 + r, y1), (x2 - r, y2), color, -
1, cv2.LINE_AA)
    cv2.rectangle(img, (x1, y1 + r), (x2, y2 - r - d), color, -
1, cv2.LINE_AA)
```

```
    cv2.circle(img, (x1 +r, y1+r), 2, color, 12)
    cv2.circle(img, (x2 -r, y1+r), 2, color, 12)
    cv2.circle(img, (x1 +r, y2-r), 2, color, 12)
    cv2.circle(img, (x2 -r, y2-r), 2, color, 12)

    return img

def UI_box(x, img, color=None, label=None, line_thickness=None):
    tl = line_thickness or round(0.002 * (img.shape[0] + img.shape[1])
/ 2) + 1
    color = color or [random.randint(0, 255) for _ in range(3)]
    c1, c2 = (int(x[0]), int(x[1])), (int(x[2]), int(x[3]))
    w, h = int(x[2]) - int(x[0]), int(x[3]) - int(x[1])
    cx, cy = int(x[0]) + w // 2, int(x[1]) + h
    cv2.circle(img, (cx, cy), 5, (0,0,255), cv2.FILLED)

    cv2.rectangle(img, c1, c2, color, thickness=tl, lineType=cv2.LINE_A
A)
    if label:
        tf = max(tl - 1, 1)
        t_size = cv2.getTextSize(label, 0, fontScale=tl / 3, thickness=
tf)[0]

        img = draw_border(img, (c1[0], c1[1] - t_size[1] -
3), (c1[0] + t_size[0], c1[1]+3), color, 1, 8, 2)

        cv2.putText(img, label, (c1[0], c1[1] - 2), 0, tl / 3, [225, 25
5, 255], thickness=tf, lineType=cv2.LINE_AA)

def intersect(A,B,C,D):
    return ccw(A,C,D) != ccw(B,C,D) and ccw(A,B,C) != ccw(A,B,D)

def ccw(A,B,C):
    return (C[1]-A[1]) * (B[0]-A[0]) > (B[1]-A[1]) * (C[0]-A[0])

def get_direction(point1, point2):
    direction_str = ""

    # calculate y axis direction
    if point1[1] > point2[1]:
        direction_str += "Down"
    elif point1[1] < point2[1]:
        direction_str += "Up"
    else:
        direction_str += ""

    return direction_str
```

```
def draw_boxes(img, bbox, names,object_id, identities=None, offset=(0,
0)):
    cv2.line(img, line[0], line[1], (46,162,112), 3)

    height, width, _ = img.shape
    # remove tracked point from buffer if object is lost
    for key in list(data_deque):
      if key not in identities:
        data_deque.pop(key)

    for i, box in enumerate(bbox):
        x1, y1, x2, y2 = [int(i) for i in box]
        x1 += offset[0]
        x2 += offset[0]
        y1 += offset[1]
        y2 += offset[1]

        # find center of bottom edge
        center = (int((x2+x1)/ 2), int((y2+y2)/2))

        # get ID of object
        id = int(identities[i]) if identities is not None else 0

        # create new buffer for new object
        if id not in data_deque:
          data_deque[id] = deque(maxlen= 64)
        color = compute_color_for_labels(object_id[i])
        obj_name = names[object_id[i]]
        label = '{}{:d}'.format("", id) + ":"+ '%s' % (obj_name)

        # add center to buffer
        data_deque[id].appendleft(center)
        if len(data_deque[id]) >= 2:
          direction = get_direction(data_deque[id][0], data_deque[id][1
])
          if intersect(data_deque[id][0], data_deque[id][1], line[0], l
ine[1]):
              cv2.line(img, line[0], line[1], (255, 255, 255), 3)
              if "Up" in direction:
                if obj_name not in totalCountUp:
                    totalCountUp[obj_name] = 1
                else:
                    totalCountUp[obj_name] += 1
              if "Down" in direction:
                if obj_name not in totalCountDown:
                    totalCountDown[obj_name] = 1
                else:
                    totalCountDown[obj_name] += 1
        UI_box(box, img, label=label, color=color, line_thickness=2)
```

```
    # Display Count
        for idx, (key, value) in enumerate(totalCountUp.items()):
            cnt_str = str(key) + ":" +str(value)
            cv2.line(img, (width - 500,25), (width,25), [85,45,255], 40
)
            cv2.putText(img, f'Number of people going up', (width - 500
, 35), 0, 1, [225, 255, 255], thickness=2, lineType=cv2.LINE_AA)
            cv2.line(img, (width - 150, 65 + (idx*40)), (width, 65 + (i
dx*40)), [85, 45, 255], 30)
            cv2.putText(img, cnt_str, (width - 150, 75 + (idx*40)), 0,
1, [255, 255, 255], thickness = 2, lineType = cv2.LINE_AA)

        for idx, (key, value) in enumerate(totalCountDown.items()):
            cnt_str1 = str(key) + ":" +str(value)
            cv2.line(img, (20,25), (500,25), [85,45,255], 40)
            cv2.putText(img, f'Number of people going down', (11, 35),
0, 1, [225, 255, 255], thickness=2, lineType=cv2.LINE_AA)
            cv2.line(img, (20,65+ (idx*40)), (127,65+ (idx*40)), [85,45
,255], 30)
            cv2.putText(img, cnt_str1, (11, 75+ (idx*40)), 0, 1, [225,
255, 255], thickness=2, lineType=cv2.LINE_AA)


    return img

class DetectionPredictor(BasePredictor):

    def get_annotator(self, img):
        return Annotator(img, line_width=self.args.line_thickness, exam
ple=str(self.model.names))

    def preprocess(self, img):
        img = torch.from_numpy(img).to(self.model.device)
        img = img.half() if self.model.fp16 else img.float()  # uint8 t
o fp16/32
        img /= 255  # 0 - 255 to 0.0 - 1.0
        return img

    def postprocess(self, preds, img, orig_img):
        preds = ops.non_max_suppression(preds,
                                        self.args.conf,
                                        self.args.iou,
                                        agnostic=self.args.agnostic_nms
,
                                        max_det=self.args.max_det)

        for i, pred in enumerate(preds):
```

```
            shape = orig_img[i].shape if self.webcam else orig_img.shap
e
            pred[:, :4] = ops.scale_boxes(img.shape[2:], pred[:, :4], s
hape).round()

        return preds

    def write_results(self, idx, preds, batch):
        p, im, im0 = batch
        all_outputs = []
        log_string = ""
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim
        self.seen += 1
        im0 = im0.copy()
        if self.webcam:  # batch_size >= 1
            log_string += f'{idx}: '
            frame = self.dataset.count
        else:
            frame = getattr(self.dataset, 'frame', 0)

        self.data_path = p
        save_path = str(self.save_dir / p.name)  # im.jpg
        self.txt_path = str(self.save_dir / 'labels' / p.stem) + ('' if
 self.dataset.mode == 'image' else f'_{frame}')
        log_string += '%gx%g ' % im.shape[2:]  # print string
        self.annotator = self.get_annotator(im0)

        det = preds[idx]
        all_outputs.append(det)
        if len(det) == 0:
            return log_string
        for c in det[:, 5].unique():
            n = (det[:, 5] == c).sum()  # detections per class
            log_string += f"{n} {self.model.names[int(c)]}{'s' * (n > 1
)}, "
        # write
        gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gai
n whwh
        xywh_bboxs = []
        confs = []
        oids = []
        outputs = []
        for *xyxy, conf, cls in reversed(det):
            x_c, y_c, bbox_w, bbox_h = xyxy_to_xywh(*xyxy)
            xywh_obj = [x_c, y_c, bbox_w, bbox_h]
            xywh_bboxs.append(xywh_obj)
            confs.append([conf.item()])
            oids.append(int(cls))
```

```
        xywhs = torch.Tensor(xywh_bboxs)
        confss = torch.Tensor(confs)

        outputs = deepsort.update(xywhs, confss, oids, im0)
        if len(outputs) > 0:
            bbox_xyxy = outputs[:, :4]
            identities = outputs[:, -2]
            object_id = outputs[:, -1]

            draw_boxes(im0, bbox_xyxy, self.model.names, object_id,iden
tities)

        return log_string

@hydra.main(version_base=None, config_path=str(DEFAULT_CONFIG.parent),
config_name=DEFAULT_CONFIG.name)
def predict(cfg):
    init_tracker()
    cfg.model = cfg.model or "yolov8n.pt"
    cfg.imgsz = check_imgsz(cfg.imgsz, min_dim=2)  # check image size
    cfg.source = cfg.source if cfg.source is not None else ROOT / "asse
ts"
    predictor = DetectionPredictor(cfg)
    predictor()

if __name__ == "__main__":
    predict()
```