# Prepoznavanje i praćenje objekata kamerom pomoću strojnog vida

**Bačevina, Luka**

**Master's thesis / Diplomski rad**

**2024**

*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:235:568753




SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

# DIPLOMSKI RAD

**Luka Bačevina**

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

# DIPLOMSKI RAD

Mentor:
prof. dr. sc. Željko Šitum, dipl. ing.

Student:
Luka Bačevina

Zagreb, 2024.

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

Zahvaljujem se svojem mentoru prof. dr. sc. Željku Šitumu na njegovoj pomoći i susretljivosti prilikom pisanja ovog rada.

Također se zahvaljujem svojoj obitelji i kolegama na podršci tijekom studiranja.

Luka Bačevina

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<br>Fakultet strojarstva i brodogradnje | |
|---|---|
| Datum | Prilog |
| Klasa: **602 - 04 / 24 - 06 / 1** | |
| Ur.broj: **15 - 24 -** | |

# DIPLOMSKI ZADATAK

Student: **Luka Bačevina** JMBAG: 0035219467

Naslov rada na hrvatskom jeziku: **Prepoznavanje i praćenje objekata kamerom pomoću strojnog vida**

Naslov rada na engleskom jeziku: **Recognition and tracking of objects with a camera using machine vision**

Opis zadatka:

Računalni (strojni) vid predstavlja granu umjetne inteligencije koja omogućuje računalima da analiziraju i interpretiraju vizualne informacije iz digitalnih slika ili videozapisa. Koristeći kompleksne algoritme i tehnike obrade slika, strojni vid omogućuje računalima da identificiraju obrasce, objekte, pa čak i određene akcije unutar slika ili videozapisa na temelju njihove veličine, oblika, boje, teksture, lokacije i drugih značajki. Ova tehnologija ima široku primjenu u različitim područjima, uključujući medicinu, sigurnost, promet, robotiku, proizvodnju i mnoga druga te ima potencijal transformirati naše živote i dosadašnje načine poslovanja mnogih gospodarskih grana. Svrha ovog rada je implementirati sustav koji može kamerom fizički pratiti objekte u realnom vremenu u stvarnoj okolini uz pomoć algoritma strojnog vida.

U radu je potrebno:

- projektirati sustav koji može kamerom pratiti osobe ili objekte koristeći algoritam strojnog vida, a prilagođen je za ergonomsko nošenje na glavi operatera
- integrirati konstrukcijske, upravljačke i senzorske komponente u funkcionalnu cjelinu i izraditi sustav za prepoznavanje i praćenje objekata kamerom
- izraditi algoritam strojnog vida za izdvajanje značajki, klasifikaciju, prepoznavanje i praćenje objekata
- istražiti tržišne mogućnosti komercijalizacije razvijenog sustava.

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

| Zadatak zadan: | Datum predaje rada: | Predviđeni datumi obrane: |
|---|---|---|
| 9. svibnja 2024. | 11. srpnja 2024. | 15. – 19. srpnja 2024. |

Zadatak zadao:

Prof. dr. sc. Željko Šitum

Predsjednik Povjerenstva:

Prof. dr. sc. Ivica Garašić

# SADRŽAJ

# POPIS SLIKA

# POPIS TABLICA

# SAŽETAK

Računalni vid interdisciplinarno je područje koje se bavi obradom vizualnih podataka za računalnu interpretaciju. Uz pomoć naprednih algoritama i strojnog učenja sustavi računalnog vida imaju mogućnost prepoznavati uzorke i detektirati objekte i radnje u stvarnom vremenu. Danas je u uporabi u raznim poljima, od medicinske robotike do autonomnih vozila.

U ovom radu predstavljen je postupak razvoja sustava računalnog vida za prepoznavanje stanja na semaforima za pješake u svrhu pomaganja slijepim osobama prilikom prelaska ceste. Potrebno je razviti prijenosni sustav koji će se sastojati od kamere i integriranog računalnog sustava na kojem će se obrađivati podaci sa kamere u stvarnom vremenu. Takav sustav moći će pomoći slijepim osobama da se lakše i neovisnije snalaze u urbanim sredinama.

Ključne riječi: računalni vid, detekcija objekata u stvarnom vremenu , prijenosni sustav, pomoć slijepim osobama

# SUMMARY

Computer vision is an interdisciplinary field which is concerned with processing visual data for use in computer interpretation. With the help of advanced algorithms and machine learning, computer vision systems have the ability to detect patterns, objects and other actions in real time. Today it is used in various fields, from medical robotics to autonomous vehicles.

This thesis will concern itself with developing a system of machine vision for detecting the state of crosswalk signals for the purpose of assisting blind persons when crossing the road. It is necessary to develop a portable system that will consist of a camera and an integrated computer on which the data from the camera will be processed in real-time. Such a system will be able to assist blind persons to navigate in urban environments more easily and independently.

Key words: computer vision, real-time object detection, portable system, blind person assistance

# 1. UVOD

## 1.1. Strojni i računalni vid

Za početak, potrebno je razlikovati računalni vid od strojnog vida. Računalni vid je informatička disciplina koja se koristi kamerama kako bi se dobili i interpretirali podaci iz stvarnog svijeta. Bavi se metodološkim i algoritamskim problemima i temama povezanim s implementacijom razvijenih sustava [1]. Strojni vid je proučavanje metoda i tehnika pomoću kojih se mogu konstruirati umjetni vizijski sustavi te korisno primijeniti u praktičnim aplikacijama. Kao takav, obuhvaća znanost i inženjerstvo vida [2]. U kontekstu ovog rada koristit će se pojam strojni vid jer nakon obrade podataka ipak postoji praktični zadatak koji sustav mora obaviti kako bi bio funkcionalan.

## 1.2. Kratki povijesni pregled

Računalni vid započeo je 1960-ih s osnovnom obradom slika i prepoznavanjem uzoraka. U 1980-ima, uvođenje algoritama za detekciju rubova i ekstrakciju značajki predstavljalo je značajnu prekretnicu. Optički tok i tehnike 3D rekonstrukcije dodatno su napredovali 1990-ih. Početkom novog tisućljeća uvedeni su pristupi strojnog učenja, ali pravi proboj dogodio se 2010-ih s dubokim učenjem i konvolucijskim neuronskim mrežama (eng. *convlonutional neural network, CNN*). To je omogućilo značajna poboljšanja u klasifikaciji slika, detekciji objekata i segmentaciji, otvarajući put za primjene u autonomnim vozilima, prepoznavanju lica i medicinskoj dijagnostici. Danas je računalni vid brzo rastuće područje s mnogobrojnim primjenama u stvarnom svijetu. [3]

## 1.3. Uporaba strojnog vida u industriji i ostalim sektorima

Strojni vid u industriji danas igra ključnu ulogu u automatiziranim procesima i zadacima kontrole kvalitete. U takvim primjenama bitno je da strojni vid izvršava zadatke u stvarnom vremenu i bez grešaka. Za to se koriste napredne kamere i razvijeni algoritmi za računalni vid. Razne industrije kao što su automobilska, farmaceutska i elektronička primjenjuju strojni vid kako bi osigurali konzistenciju proizvoda, detektirali defekte, navodili robotske sustave, itd. Strojni vid također se koristi u drugim sektorima, gdje je jedan od istaknutijih prijevozni sektor. Najnovije inovacije dovode nas sve bliže autonomnim automobilima, a pametni semafori s kamerama za prepoznavanje količine prometa već su neko vrijeme u uporabi. Još jedno

neizostavno područje strojnog vida je medicina. Koristi se prilikom operacija s robotom, ili kao pomoćni alat za prepoznavanje tumora i ostalih dijagnoza prilikom iščitavanja rezultata RTG-a, magnetske rezonance, itd. [4].

## 1.4. Poteškoće slijepih osoba u svakodnevnom životu

Slijepe osobe svakodnevno se susreću s mnogim izazovima i poteškoćama, od uporabe raznih svakodnevnih predmeta do snalaženja u prostoru. Ove poteškoće uvelike im mogu ograničiti slobodu i neovisnost, te im također čine život opasnijim, posebno u urbanim sredinama. Strojni vid nudi razna obećavajuća rješenja na način da uređaj umjesto osobe vrši zadatke koji su za slijepe i slabovidne osobe nemogući. Na primjer, sustav koji može umjesto osobe čitati i interpretirati tekst uklanja potrebu za drugom osobom koja bi to morala raditi. Problem kojim će se baviti ovaj rad je izrada sustava koji pomaže slijepim osobama prilikom prelaska ceste na označenom prijelazu sa semaforom. Pošto nisu svi semafori opremljeni sa zvučnim signalima za slijepe osobe, ovaj sustav u teoriji ima primjenjivost u stvarnom svijetu ako se izvede na način da bude dovoljno pouzdan.

## 1.5. Koncepcija sustava za asistenciju slijepim osobama prilikom prelaska ceste

Sustav je koncipiran tako da bude jednostavan za uporabu i ergonomski nenametljiv. Sastojao bi se od kamere, nosača, računalne jedinice, zvučnika i/ili neke vrste aktuatora u svrhu da je osoba i nijema (npr. može biti jednostavna vibracija telefona).

Kamera treba biti dovoljno dobra da sustav može s visokom preciznošću i sigurnošću detektirati signalizaciju semafora prilikom prelaska osobe preko ceste. Tu se nameće jedan bitan argument: zašto ne iskoristiti kameru od pametnog telefona? U današnje doba praktički svaka osoba ima pametni telefon s kamerom koja i premašuje zahtjeve zamišljenog sustava.

Postoji više razloga zašto ne koncipirati sustav na taj način. Prvi bi bio taj da bi osobe s nedostatnim uređajima bile zakinute, te bi za njih sustav bio neprimjenjiv. Također, ukoliko se pametni telefon stavi na jako vidljivo mjesto na tijelu on postaje lakša meta za krađu, posebice ako je skuplji model. S druge strane ne treba ni zanemariti argument da bi kompletna implementacija na pametnom telefonu bila najjednostavnija i zasigurno se treba razmotriti. Može se uvijek ostaviti opcija da sustav bude *phone-only* u slučaju da je to ostvarivo (npr. za modele mobitela s boljom kamerom).

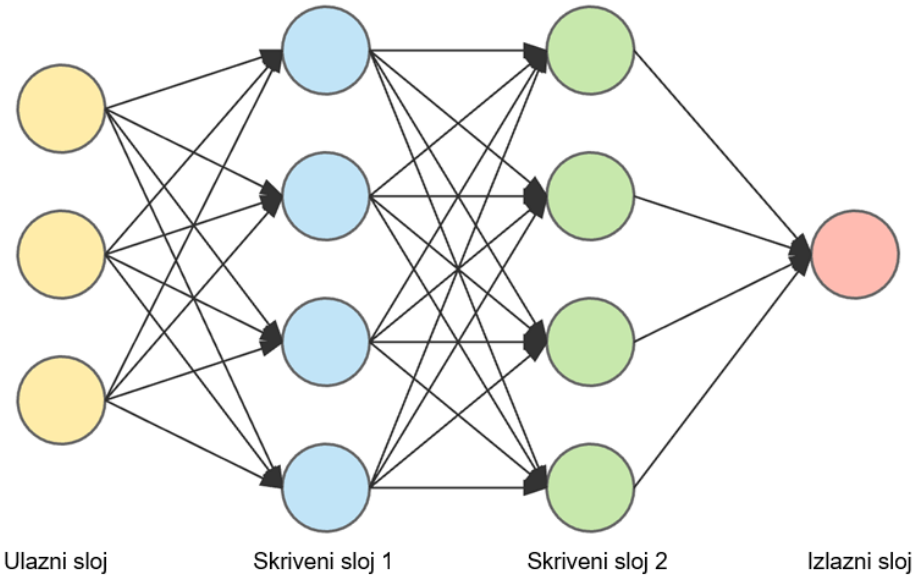
Nosač za kameru trebao bi biti ergonomski nenametljiv, prilagodljiv za razne tjelesne konstitucije i estetski prihvatljiv. Također mora biti postavljen tako da je kamera usmjerena u pravcu kretanja korisnika. Dvije se opcije ističu: prsa i glava. Položaj na prsima omogućavao bi lakše i nenametljivije postavljanje kamere i manji put kabla. Nadalje, postoje već takvi sustavi koji se koriste kod policije u nekim državama. Nedostatci tog položaja bili bi to što je niže na tijelu i kamera bi u situaciji gdje ima puno ljudi ili prepreka imala lošu preglednost okoline. Drugi položaj je na glavi osobe. Taj položaj imao bi bolju preglednost, a na glavi korisnika ima više položaja na koje može biti postavljena. Jedna od opcija bila bi na vrh glave (npr. GoPro), ili pozicionirana na naočalama, što predstavlja estetski najbolju opciju. Problem takvog pozicioniranja bila bi činjenica da korisnik ne mora uvijek gledati ravno, no u većini slučajeva hoće, posebice ako zna da rad sustava ovisi o tome. Još jedna mana je što bi kamera bila uvijek lako vidljiva svima oko korisnika.

Računalna jedinica koja će obrađivati podatke s kamere u ovom slučaju bio bi pametni telefon. Velika većina današnjih pametnih telefona sposobna je obrađivati slike u stvarnom vremenu i također slati zvučne ili taktilne signale pomoću vlastitog zvučnika ili bluetooth uređaja, vibracijama. Još jedna velika prednost korištenja pametnog telefona je što sustav ne bi zahtijevao vanjski izvor napajanja već bi se služio baterijom pametnog telefona.

# 2. RAČUNALNI VID

## 2.1. Neuronske mreže

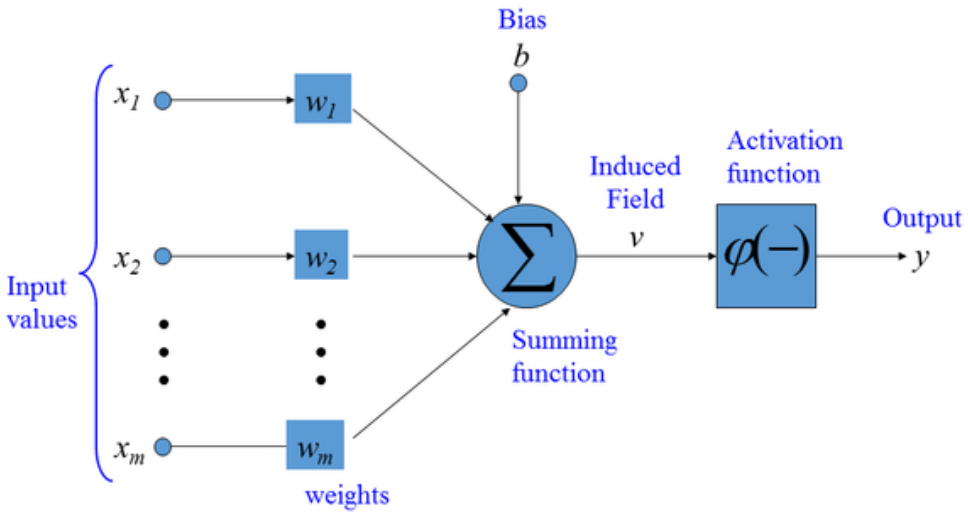
U kontekstu strojnog učenja neuronske mreže su računalni modeli koji rade odluke na način sličan ljudskom mozgu. Koristeći umjetne neurone kako bi oponašali način na koji pravi neuroni rade, može se dobiti model koji ima sposobnost prepoznavanja, odlučivanja i zaključivanja [5]. Imaju slojevitu neuronsku strukturu sastojeći se načelno od ulaznog sloja, jednog ili više skrivenih slojeva te izlaznog sloja. Svaki umjetni neuron povezan je s jednim ili više drugih umjetnih neurona u idućem sloju. Na slici 2.1 prikazana je struktura jedne osnovne neuronske mreže.




**Slika 2.1 Prikaz strukture neuronske mreže [6]**

Kada na ulaz neurona dođe neka vrijednost ona se množi s težinama. Ti umnošci zatim se sumiraju u prijenosnoj funkciji i ako je potrebno dodaje se *bias*. Ta vrijednost prosljeđuje se kao ulaz u aktivacijsku funkciju koja shodno tome računa izlaz. Izlaz se uspoređuje s pragom aktivacije i ukoliko je veći prosljeđuje se kao ulaz na idući neuron. Na slici 2.2 prikazan je princip rada umjetnog neurona. Izlazni sloj neuronske mreže daje informacije koje su potrebne korisniku modela. U fazi učenja ti će se podaci uspoređivati s podacima iz skupa za učenje te će se pomoću funkcije gubitka računati greška. Zatim će se težine u neuronima podešavati tako

da se izračunata greška pokuša smanjiti. Na taj način neuronska mreža postaje bolja u smanjivanju greške, odnosno postizanju boljih rezultata na izlazu.




**Slika 2.2 Prikaz strukture umjetnog neurona [7]**

## 2.2. Konvolucijske neuronske mreže

Konvolucijske neuronske mreže (eng. *convolutional neural networks, CNN*) su tip neuronskih mreža posebno razvijen za obradu podataka koji imaju rešetkastu topologiju (2D), no mogu se koristiti i za jednodimenzionalne i trodimenzionalne strukture [8]. Konvolucijske neuronske mreže razlikuju se od drugih vrsta neuronskim mreža zbog boljih performansi prilikom obrade slike, govora i zvuka. Zbog toga se koriste za zadatke kao što su računalni vid i procesiranje prirodnog govora (eng. *natural language processing*) [9].

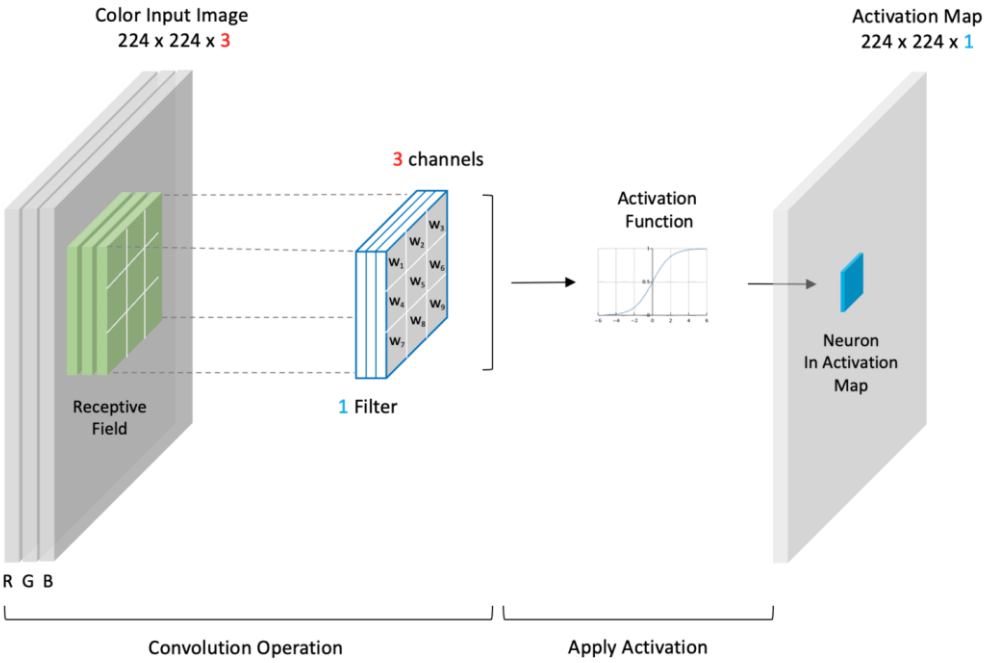
Konvolucijske neuronske mreže sastoje se od 3 glavna tipa sloja.

### *2.2.1. Konvolucijski sloj*

Konvolucijski sloj prvi je sloj u konvolucijskim neuronskim mrežama i njihova osnovna gradivna jedinica. Za sliku u boji sloj funkcionira na sljedeći način:

- Slika u boji reprezentirana je kao 3D matrica gdje su prve dvije dimenzije širina i visina slike u pikselima dok treća dimenzija „dubina“ predstavlja RGB vrijednosti piksela.
- Filterom (ili kernelom) određenih dimenzija (često 3x3) određenim korakom prolazi se po slici i računa se dot produkt. Filter također mora imati „dubinu“ jednaku ulaznoj slici. Rezultat prolaska filtra je aktivacijska mapa (*eng. feature map*) čija je dubina „1“ zato što se radi prostorni dot produkt. Pošto jedan konvolucijski sloj može imati više filtara

nova slika (sve aktivacijske mape) može biti veće dubine od ulazne. Prava moć konvolucijskih neuronskih mreža je što prilikom učenja one uče podešavati parametre filtara kako bi oni naglašavali određene značajke, što omogućuje prepoznavanje i smanjuje izlaz funkcije gubitka.




**Slika 2.3 Prikaz principa rada konvolucijskog filtera [10]**

- Kako bi se spriječio gubitak rubnih piksela i smanjivanja dimenzije često se ulazna matrica proširuje sa nulama tako da izlazna matrica aktivacijske mape bude jednake širine i visine (eng. *zero-padding*) [11].
- Aktivacijska funkcija koja se koristi u ovom sloju je ReLu (eng. *rectified linear unit*) funkcija.

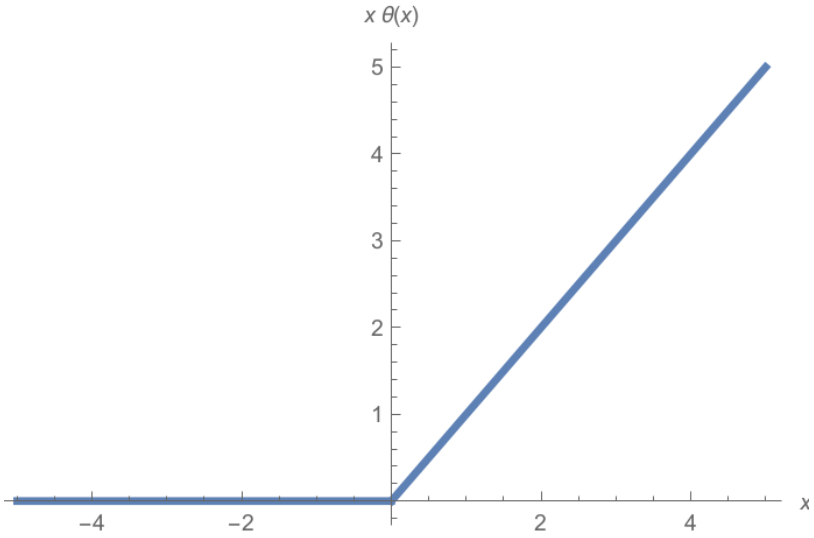
### *2.2.2. ReLu aktivacijska funkcija*

ReLu funkcija je najčešće korištena aktivacijska funkcija kod konvolucijskih neuronskih mreža zbog svoje jednostavnosti i učinkovitosti. Prvi put je definirana 1969. godine no tek se u istraživanju 2011. godine [12] pokazalo da ima odlična svojstva za ubrzavanje učenja neuronskih mreža,

Definirana je kao funkcija koja vraća ulaznu vrijednost ako je pozitivna, a ako je negativna vraća nulu. Prikazana je izrazom (2.1).

$$f(x) = \begin{cases} x \; ako \; je \; x > 0 \\ 0 \; ako \; je \; x \leq 0 \end{cases} \tag{2.1}$$

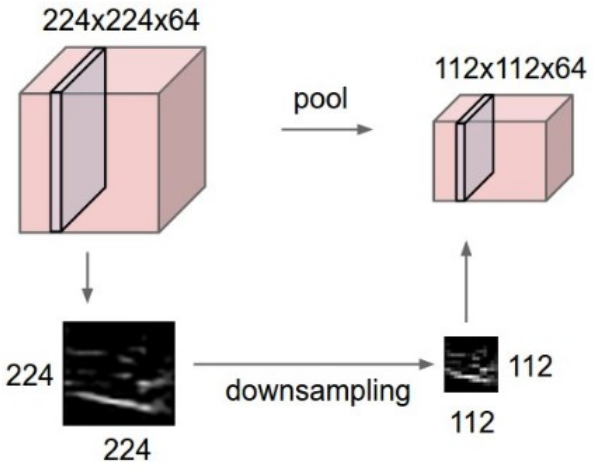
Iznimno je jednostavna za prikazivanje grafom, jer za negativne vrijednosti x funkcija je 0, te kada je x > 0 je jednostavan pravac s nagibom iznosa 1. Prikazana je grafom na slici 2.4.




**Slika 2.4 ReLu aktivacijska funkcija [13]**

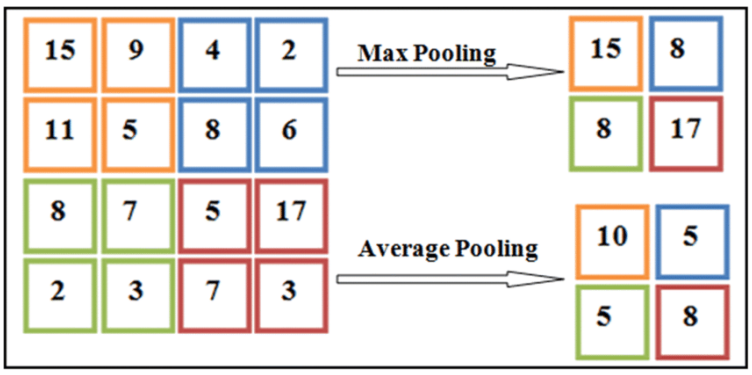
### *2.2.3. Sloj sažimanja*

Sloj sažimanja (eng. *pooling layer*) je sloj koji služi kako bi se reducirala kompleksnost neuronske mreže smanjivanjem dimenzija aktivacijskih mapa. Također koristi filter i djeluje preko svake aktivacijske mape neovisno. Aktivacijska funkcija slojeva sažimanja također je ReLu funkcija. Slika 2.5 grafički prikazuje princip na kojem sloj sažimanja radi uz primjer.




**Slika 2.5 Prikaz principa rada sloja sažimanja [14]**

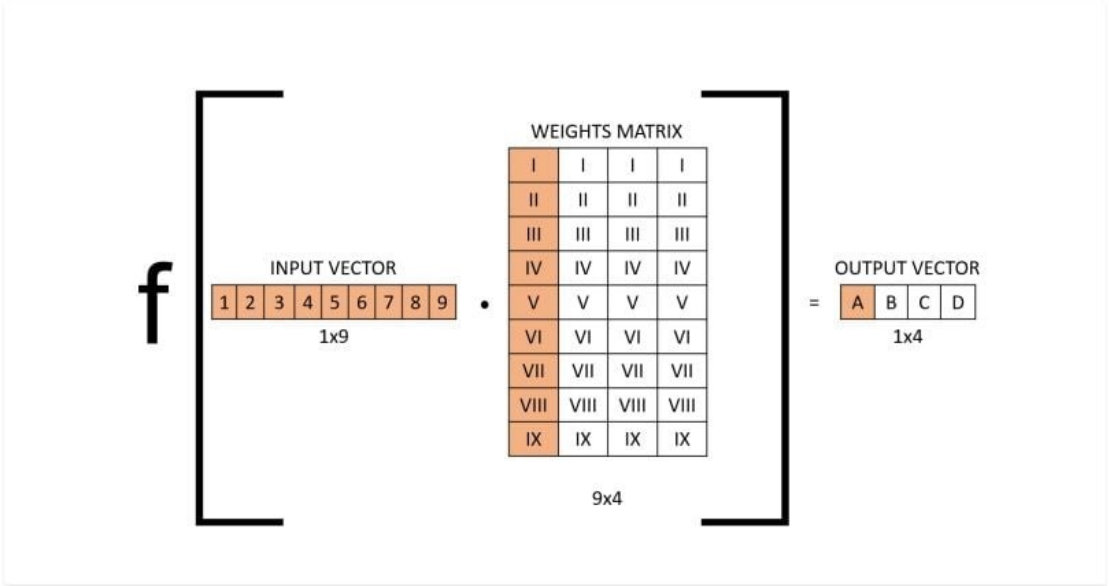
*Max pooling* je vrsta sažimanja kada filter koji prolazi preko aktivacijske mape pamti samo najveću vrijednost sadržanu u dimenziji filtera. Ova vrsta sažimanja je korištena češće. *Average pooling* računa srednju vrijednost te ju šalje u iduću aktivacijsku mapu. Unatoč tome što ovim postupkom gubimo informacije, on je koristan kako bi pojednostavili mrežu radi boljih performansi. Slika 3.6. prikazuje princip rada ova dva filtera na 4x4 matrici.




**Slika 2.6 Prikaz *Max pooling*-a i *average pooling*-a [15]**

### *2.2.4. Potpuno povezani sloj*

Potpuno povezani sloj (eng. *fully connected (FC) layer*) je sloj koji obavlja zadatak klasifikacije na temelju značajki izvučenih kroz prethodne slojeve i njihove različite filtre. U potpuno povezanom sloju svi su ulazni neuroni povezani sa svim izlaznim neuronima. Slika koja dolazi na ulaz FC sloja mora se „spljoštiti“ u stupac vektor. Potpuno povezani slojevi obično koriste *softmax* aktivacijsku funkciju za pravilno klasificiranje ulaza [16], vraćajući na izlazu vjerojatnost klase od 0 do 1. Ovaj sloj je zadnji sloj u svakoj konvolucijskoj neuronskoj mreži. Na slici 3.7. prikazan je princip njegovog rada.




**Slika 2.7 Princip rada potpuno povezanog sloja [17]**

### *2.2.5. Softmax aktivacijska funkcija*

*Softmax* funkcija u pravilu je zadnja aktivacijska funkcija neuronske mreže koja služi za klasifikaciju objekata, posebno u slučajevima gdje ima puno klasa. Ona transformira skup linearnih rezultata u vjerojatnosti koje se zbrajaju do jedan, omogućujući interpretaciju izlaza modela kao vjerojatnosti pripadnosti svakoj klasi.

Svaki ulaz podiže se na eksponencijalnu vrijednost, a zatim se dijeli sa zbrojem svih eksponencijalnih vrijednosti, čime se osigurava da rezultirajuće vjerojatnosti budu u rasponu od 0 do 1. Time se također daje prednost većim ulaznim vrijednostima nad manjima. Formula za računanje izlaza *softmax* aktivacijske funkcije dana je u jednadžbi (2.2):

$$\sigma(z)_i = \frac{e^{\beta z_i}}{\sum_{j=1}^{K} e^{\beta z_j}} \tag{2.2}$$

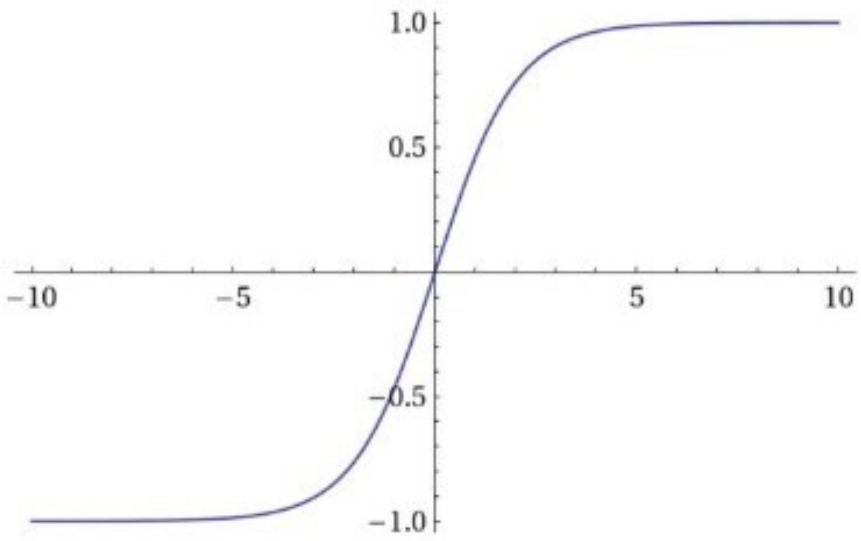
gdje su:

$\sigma$ – vektor vjerojatnosti klase,

$z_i$ – ulazni vektor,

$\beta$ – opcionalni parametar, služi za podešavanje vrijednosti koje želimo naglasiti funkciji (ako je $\beta \geq 0$ naglašavaju se veće vrijednosti, ako je $\beta < 0$ naglašavaju se manje vrijednosti).

Graf *softmax* aktivacijske funkcije prikazan je na grafu na slici ispod:



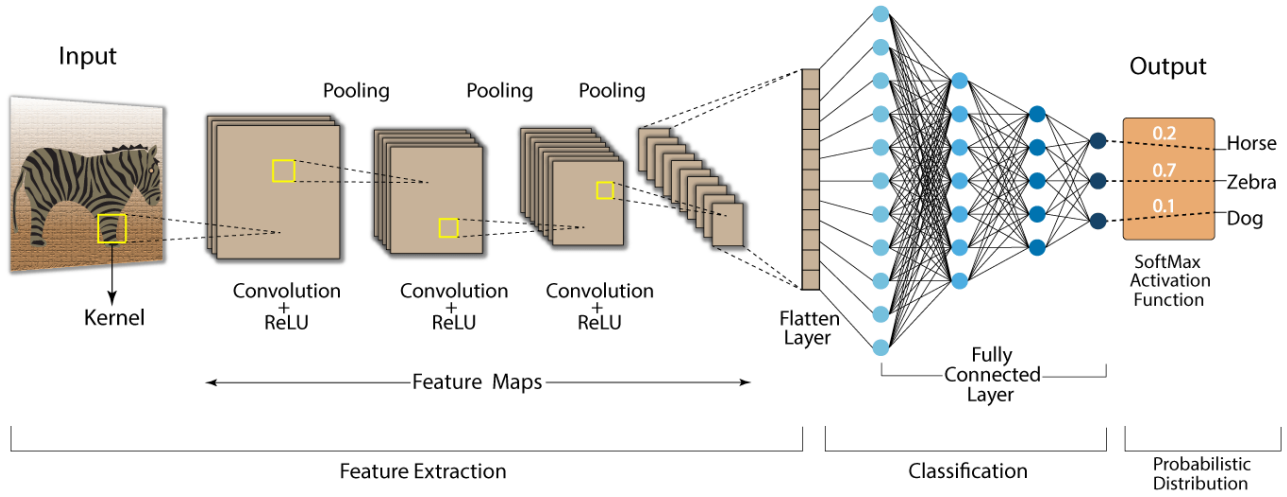

**Slika 2.8 *Softmax* aktivacijska funkcija [18]**

## 2.3. Vrste konvolucijskih neuronskih mreža za računalni vid

Konvolucijske neuronske mreže su se kao novitet u području umjetne inteligencije pojavile 1980-ih, no njihov puni potencijal se nije mogao potpuno iskoristiti do sredine 2000-ih godina na grafičkim procesorima (GPU)[19]. Danas postoji mnoštvo CNN-ova koje se koriste u svrhu detekcije, a neki od najpoznatijih su:

- Region-based Convolutional Neural Networks (R-CNN), Fast R-CNN, Faster R-CNN,
- You Only Look Once (YOLO),
- Single Shot Detector (SSD),
- RetinaNet.

Na slici 2.9 prikazan je dijagram toka koji prikazuje rad konvolucijskih neuronskih mreža.




**Slika 2.9 Prikaz rada konvolucijske neuronske mreže [20]**

## 2.4. Ultralytics YOLO

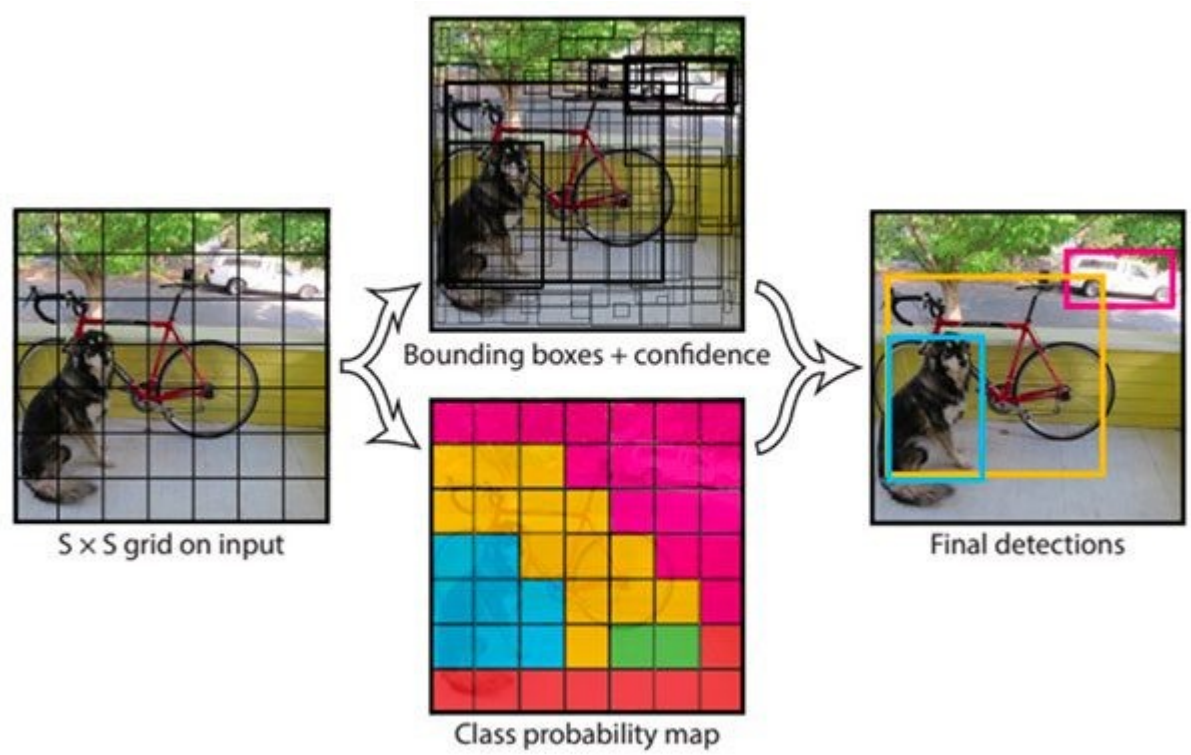
YOLO (eng. *You Only Look Once*) je algoritam za detekciju objekata u stvarnom vremenu kojeg su razvili Joseph Redmon i Ali Farhadi 2015. godine. To je detektor objekata u jednoj fazi koji koristi konvolucijsku neuronsku mrežu (CNN) za predviđanje granica okvira i klasnih vjerojatnosti objekata na ulaznim slikama. YOLO algoritam dijeli ulaznu sliku na mrežu ćelija, i za svaku ćeliju predviđa vjerojatnost prisutnosti objekta i koordinate okvira objekta. Također predviđa klasu objekta. Za razliku od detektora objekata u dvije faze kao što su R-CNN i njegove varijante, YOLO obrađuje cijelu sliku u jednom prolazu, što ga čini značajno bržim i učinkovitijim. YOLO se koristi u raznim primjenama kao što su autonomna vozila i sustavi nadzora. Za razliku od tradicionalnih algoritama koji koriste dvostupanjski pristup (prvo generiraju prijedloge regija, a zatim klasificiraju te regije), YOLO koristi jednostupanjski pristup koji istovremeno predviđa više okvira (bounding boxes) i njihove klasne vjerojatnosti u regijama u jednoj evaluaciji. Ova metoda treniranja i detekcije omogućuje YOLO-u da obrađuje slike brže, što ga čini vrlo pogodnim za primjene koje zahtijevaju trenutnu detekciju i odgovor. YOLO je od svojeg začetka 2015. godine prošao kroz puno iteracija i nadogradnji, a posljednja od njih (u trenutku pisanja ovog rada) je YOLOv8. [21]

Na slici 2.10 je vidljiva robusnost algoritma jer prepoznaje ljudsku glavu naslikanu nerealističnim umjetničkim stilom s pouzdanosti od 80 %, unatoč tome što je u setu podataka za učenje imao samo fotografije ljudskih glava.


**Slika 2.10 Primjer inferencije YOLOv8 algoritmom na fotografiju autoportreta Vincenta van Gogha koristeći algoritam učenja za prepoznavanje ljudske glave [21]**

### *2.4.1. Princip rada YOLO algoritma*

1. Ulaznoj slici mijenjaju se dimenzije ovisno o parametru ulazne veličine slike YOLO modela.
2. Slika se dijeli na mrežu ćelija i ulazi u CNN, pri čemu je svaka ćelija odgovorna za predviđanje okvira klasa i njihove pouzdanosti, ali i svaka ćelija vraća vrijednost vjerojatnosti klase što govori algoritmu koja se klasa nalazi u kojoj ćeliji slike.
3. Prije nego što se prikazuju rezultati, nepotrebni okviri filtriraju se metodom potiskivanja ne-maksimuma. Ta metoda će biti objašnjena kasnije u ovom radu. Sve što za sad treba znati je da ona filtrira nepotrebne okvire i vraća nazad čistu preglednu sliku s jednim okvirom najveće pouzdanosti za svaku instancu klase koju treba detektirati.
4. Konačni izlaz je skup predviđenih okvira i klasnih oznaka za svaki objekt na slici




**Slika 2.11 Pojednostavljeni prikaz rada YOLO algoritma [22]**

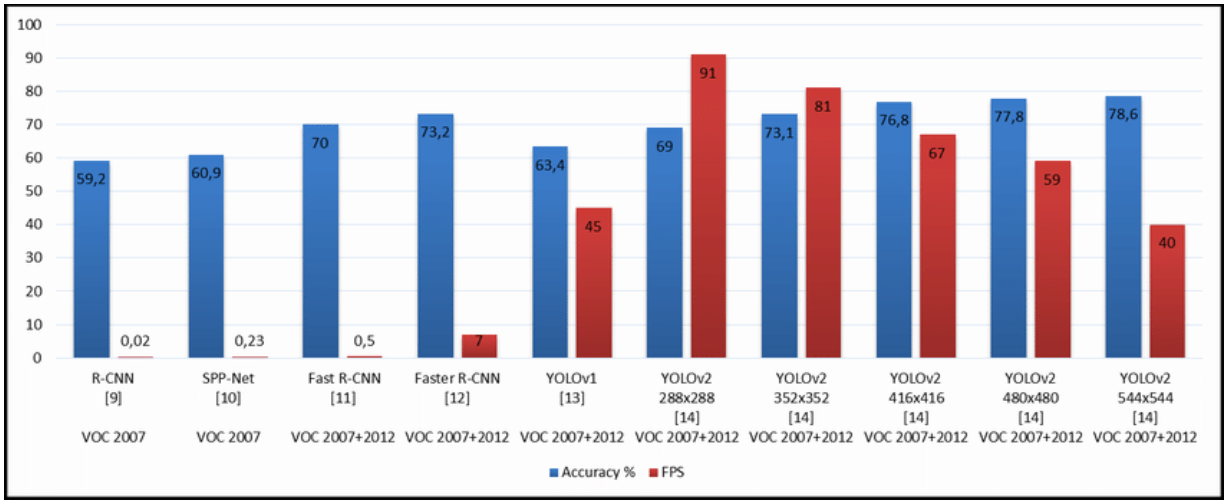
### *2.4.2. Prednosti i nedostatci YOLO algoritama nad drugim algoritmima za detekciju*

YOLO algoritam, kao što je spomenuto prije razlikuje se od ostalih algoritama za detekciju po tome što u samo jednom prolazu lokalizira okvire klasa i njihove vjerojatnosti da se nalaze u nekoj regiji (ćeliji). To ga čini značajno bržim od drugih algoritama za detekciju i to je ujedno njegova najveća prednost. Njegova brzina omogućava mu rad u realnom vremenu. U tablici 2.1 dana je detaljnija usporedba prednosti i mana YOLO algoritma.

**Tablica 2.1 Tablica prednosti i nedostataka YOLO algoritima [22][23]**

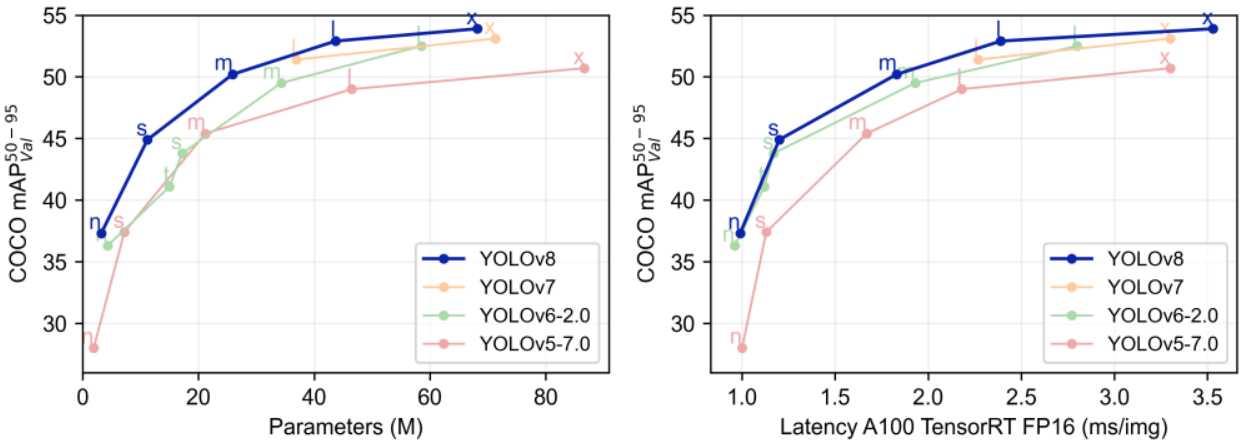
| Prednosti | Nedostatci |
|---|---|
| • Velika brzina i mogućnost obrade podataka u stvarnom vremenu<br>• Mogućnost obrade slika i videozapisa velikih dimenzija<br>• Jednostavnost jednostupanjskog pristupa<br>• Ujedinjena arhitektura – algoritam predviđa okvire i vjerojatnosti klasa istovremeno | • Točnost lokalizacije nekad narušena kod detekcije malih objekata u usporedbi s drugim algoritmima (R-CNN)<br>• Ako su objekti jako blizu jedan drugome može doći do problema u razlikovanju zbog pristupa temeljenog na mreži ćelija<br>• Naknadna dodatna obrada potrebna za filtriranje redundantnih okvira |

Na dijagramu na slici 2.12 prikazana je usporedba YOLOv1 i YOLOv2 algoritma s raznim R-CNN algoritmima. Plavi stupac predstavlja preciznost algoritma u obliku postotka dok crveni predstavlja brzinu rada algoritma u sličicama po sekundi (eng. *frames per second, FPS)*. Dok se R-CNN algoritmi ističu po preciznosti može se vidjeti kako su značajno sporiji i neprimjenjivi u uvjetima uporabe u realnom vremenu.




**Slika 2.12 Usporedba algoritama za detekciju [24]**

Na slici 2.13 je na grafovima prikazana usporedba raznih verzija YOLO modela različitih veličina. Usporedba je rađena na Microsoft-ovom COCO (Common objects in Context) skupu podataka. Na lijevom grafu uspoređuje se preciznost u ovisnosti od broja parametra modela (veći modeli kao što su l i x imaju veći broj parametara). Na desnom grafu uspoređuje se preciznost modela i njegova brzina.




**Slika 2.13 Usporedba broja parametara i brzine raznih YOLO modela [25]**

## 2.5. Vrednovanje rezultata algoritama za detekciju

Metrike po kojima se vizijski sustavi vrednuju nisu komplicirane, no svejedno ih je potrebno objasniti kako bi se pravilno mogli interpretirati rezultati učenja. Najčešće korištene metrike uključuju točnost (koja se u ovom radu neće razmatrati), preciznost, odziv i F1 rezultat. Kroz temeljitu evaluaciju, sustav se može optimizirati kako bi postigao najbolje moguće rezultate u stvarnim primjenama.

### *2.5.1. Prag pouzdanosti*

Prag pouzdanosti (eng. *confidence threshold*) je vrijednost koja se koristi u binarnoj klasifikaciji za određivanje granične točke pri kojoj model odlučuje hoće li klasificirati primjerak kao pripadnika pozitivne klase ili negativne klase, ovisno o njegovoj pouzdanosti (eng. *confidence score*). Ovaj prag se koristi za pretvorbu kontinuiranih izlaznih vrijednosti modela (koje su obično vjerojatnosti) u binarne klasifikacije. Prag pouzdanosti uvijek je broj između 0 i 1.

### *2.5.2. Matrica konfuzije*

Matrica konfuzije/zbrke (eng. *confusion matrix*) je vrsta tablice koja ima specifičnu ulogu u strojnom učenju i srodnom inženjerstvu. Pomaže u prikazivanju i klasifikaciji rezultata nakon odrađenog postupka učenja ili validacija.

Matrica konfuzije obavlja binarni postupak klasifikacije. Tablica se sastoji od minimalno dva retka i dva stupca. Ispunjena je sa četiri vrijednosti - istinski pozitivni, lažni pozitivni, istinski negativni i lažni negativni. Matrica konfuzije može biti i većih dimenzija ovisno o broju klasa u modelu. Sustav radi dobro ako se većina rezultata u matrici nalazi na njenoj dijagonali.

U tablici 2.2 prikazana je osnovna struktura matrice konfuzije, dok je u tablici 2.3 dan primjer načina popunjavanja u slučaju detekcije dabrova i svizaca radi lakšeg razumijevanja. Svaku od ćelija u matrici popunili bi brojem instanci u kojima se taj slučaj dogodio. Polja označeno zeleno su dobri ishodi, odnosno istinski pozitivni i istinski negativni.

**Tablica 2.2 Osnovna matrica konfuzije**

| **Predviđeno** / **Stvarno** | **Pozitivno** | **Negativno** |
|---|---|---|
| **Pozitivno** | **Istinski pozitivno (TP)** | **Lažno negativno (FN)** |
| **Negativno** | **Lažno pozitivno (FP)** | **Istinski negativno (TN)** |

**Tablica 2.3 Primjer matrice konfuzije**

| Predviđeno / Stvarno | **Dabar** | **Svizac** | **Pozadina** |
|---|---|---|---|
| **Dabar** | Dabar prepoznat kao dabar (TP) | Dabar prepoznat kao svizac (FP) | Dabar prepoznat kao pozadina (FN) |
| **Svizac** | Svizac prepoznat kao dabar (FP) | Svizac prepoznat kao svizac (TP) | Svizac prepoznat kao pozadina (FN) |
| **Pozadina** | Pozadina prepoznata kao dabar (FP) | Pozadina prepoznata kao svizac (FP) | Pozadina prepoznata kao pozadina (TN) |

Istinski pozitivni (eng. *true positive, TP*): Ovo su slučajevi kada je model ispravno predvidio pozitivan ishod (npr. model ispravno predviđa lokaciju neke klase na slici).

Lažno pozitivni (eng. *false positive, FP*): Ovo su slučajevi kada je model netočno predvidio pozitivan ishod (npr. model predviđa lokaciju klase na mjestu na kojem se ona ne nalazi).

Lažno negativni (eng. *false negative, FN*): Ovo su slučajevi kada je model netočno predvidio negativan ishod (npr. model predviđa da na nekoj lokaciji nema klase, ali ona je zapravo tamo).

Istinski negativni (eng. *true negative, TN*): Ovo su slučajevi kada je model ispravno predvidio negativan ishod (npr. model ispravno predviđa da na nekoj lokaciji nema klase).

### *2.5.3. Točnost*

Točnost (eng. *accuracy*) je metrika koja pokazuje koliko je istinski pozitivnih predviđanja bilo u svim predviđanjima modela. Jako je intuitivna metrika i sve što treba znati o njoj je da je model bolji što je veća točnost. Računa se prema jednadžbi (2.3) danoj ispod:

$$Točnost = \frac{TP}{TP + TN + FP + FN} \tag{2.3}$$

### *2.5.4. Preciznost*

Preciznost (eng. *precision*) je metrika koja pokazuje koliko su točna pozitivna predviđanja. Visoka preciznost ukazuje na točnost u pozitivnim predikcijama, čak i ako to znači propuštanje nekih pozitivnih instanci. Ovo je važno u scenarijima gdje su lažno pozitivni rezultati neprihvatljivi ili disruptivni. Preciznost se također može nazvati pozitivna prediktivna vrijednost (PPV). Računa se na način da se sva točna predviđanja podijele s ukupnim brojem predviđanja, kao što je prikazano u jednadžbi (2.3):

$$Preciznost = \frac{TP}{TP + FP} \tag{2.4}$$

### *2.5.5. Odziv*

Odziv (eng. *recall*) ili osjetljivost je mjera koja pokazuje koliko su istinski pozitivni slučajevi ispravno identificirani i računa se prema jednadžbi 2.5 danoj ispod:

$$Odziv = \frac{TP}{TP + FN} \tag{2.5}$$

Visok odziv ukazuje na prepoznavanje što više pozitivnih instanci, čak i ako to znači uključivanje nekih lažno pozitivnih. Ovo je ključno u situacijama gdje propuštanje pozitivne instance ima značajne posljedice. Odziv se također može nazivati stopa istinski pozitivnih rezultata (eng. t*rue positive rate, TPR*).

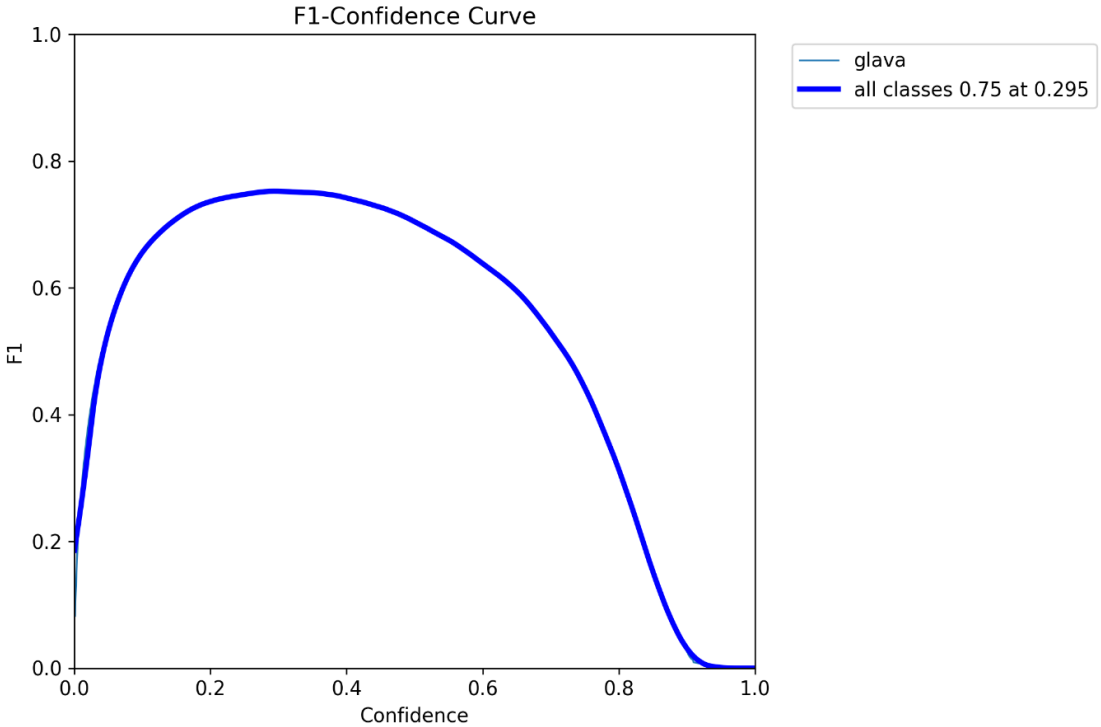
### *2.5.6. F1 rezultat*

Ova metrika predstavljena je harmonijskom sredinom preciznosti i odziva. Daje jedinstvenu mjeru koja balansira oba aspekta, računa se po sljedećoj formuli danoj u jednadžbi 2.6:

$$F1 = 2 \cdot \frac{Preciznost \cdot Odziv}{Preciznost + Odziv} \tag{2.6}$$

F1 rezultat se obično prikazuje u obliku F1 krivulje koja prikazuje F1 rezultat u odnosu na različite vrijednosti praga povjerenja. Ta krivulja bitno pomaže u ocjenjivanju performansi modela i pomaže u odabiru optimalnog praga koji maksimizira F1 rezultat kako bi se postigla ravnoteža između preciznosti i odziva.

Ravna ili polako mijenjajuća F1 krivulja kroz raspon pragova ukazuje na otpornost na izbor praga. Nagli pad F1 rezultata ukazuje na osjetljivost na izbor praga, sugerirajući da performanse modela jako ovise o pravilnom odabiru praga. Na slici 2.13. prikazana je F1 krivulja dobivena tokom učenja modela koji je prepoznao glavu na slici 2.10. S nje se može iščitati da je ravnoteža preciznosti i odziva postignuta za vrijednost praga pouzdanosti od 0.295.

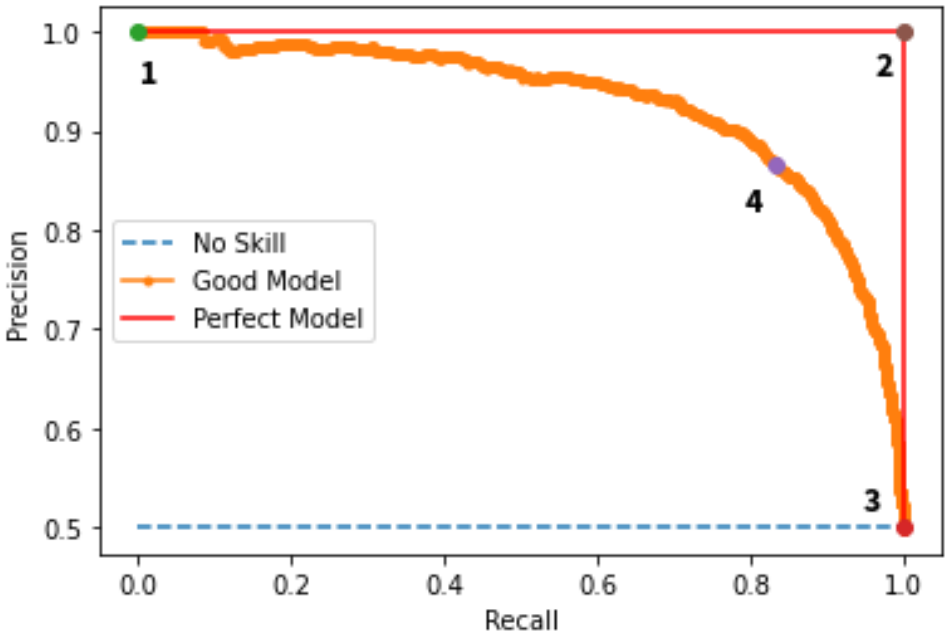



**Slika 2.14 Primjer grafa F1 krivulje**

### *2.5.7. Krivulja preciznost-odziv (PR krivulja)*

PR krivulja (eng. Precision-Recall curve) pomaže u razumijevanju kompromisa između preciznosti i odziva za različite vrijednosti praga. Posebno je korisna kada se radi s neuravnoteženim skupovima podataka gdje je broj pozitivnih instanci znatno manji od broja negativnih instanci. Krivulja pruža vizualni alat za procjenu performansi modela u smislu njegove sposobnosti da prepozna relevantne instance (odziv) uz minimiziranje lažno pozitivnih (preciznost).

PR krivulja dobije se tako da se za svaku vrijednost praga pouzdanosti od 0 do 1 izračunaju preciznost i odziv te se iscrtaju na grafu. Što je veća površina ispod PR krivulje to će performanse modela biti bolje. Idealni model na PR krivulji bi bio konstantan do vrijednosti (1,1) nakon čega bi se odmah spustio na x-os. Više modela može se usporediti prikazivanjem njihovih PR krivulja na istom grafu. Model čija je krivulja bliža gornjem desnom kutu obično ima bolje performanse. Ravne regije na PR krivulji ukazuju na to da preciznost ostaje relativno konstantna kroz raspon vrijednosti odziva, sugerirajući otpornost na promjene praga u tom području. Na grafu na slici 2.14 prikazane su idealna, realna i najlošija moguća PR krivulja.




**Slika 2.15 Grafovi idealne, dobre i najlošije moguće PR krivulje [26]**

### *2.5.8. Srednja prosječna preciznost*

Srednja prosječna preciznost (eng. *mean average precision, mAP*) je sveobuhvatna metrika koja evaluira performanse modela kroz različite razine preciznosti i odziva. Kombinira preciznost i

odziv u jednu metriku. Često se koristi kao glavna metrika za usporedbu različitih modela strojnog vida.

Kako bismo izračunali mAP moramo prvo izračunati prosječnu preciznost (*eng. average precision, AP*) na sljedeći način za svaku klasu u modelu jednadžbom (2.7):

$$AP = \sum_{n} (R_n - R_{n-1}) P_n \tag{2.7}$$

gdje su:

$R_n$ – odziv na *n*-tom pragu,

$R_{n-1}$ – odziv na *(n-1)*-tom pragu,

$P_n$ – preciznost na razini odziva $R_n$ (odnosno na *n*-tom pragu).

Nadalje, mAP se računa izrazom za aritmetičku sredinu svih prosječnih preciznosti (za sve klase) danoj u sljedećoj jednadžbi (2.8):

$$mAP = \frac{1}{k} \sum_{i=1}^{k} AP_i \tag{2.8}$$

gdje su:

$k$ – broj klasa,

$AP_i$ – prosječna preciznost *i*-te klase.

Srednja prosječna preciznost će se prilikom vrednovanja modela prikazivati na dva različita načina:

- **mAP@50 -** odnosi se na srednju prosječnu preciznost izračunatu na IoU pragu od 0.50. Ova metrika je dobar početak i pruža jednostavniju, blažu evaluaciju.
- **mAP@50-95** - odnosi se na još jednu aritmetičku sredinu od svih srednjih prosječnih preciznosti izračunatih na više IoU pragova, od 0.50 do 0.95 u koracima od 0.05. Ova metrika daje detaljniju i strožu procjenu, naglašavajući modele koji dosljedno dobro rade na različitim razinama točnosti detekcije.

### *2.5.9. Presjek preko unije*

Presjek preko unije (eng. *intersection over union, IoU*) je metrika koja se koristi za evaluaciju preklapanja između predviđenog okvira detekcije i stvarnog okvira (označenog u za učenje/validaciju). Određivanjem IoU praga možemo filtrirati nedostatne ili nepotrebne rezultate.

Računa se prema jednadžbi 2.9:

$$IoU = \frac{A_i}{A_u} \tag{2.9}$$

gdje su:

$A_i$ – površina preklapanja predviđenog i stvarnog okvira,

$A_u$ – površina unije predviđenog i stvarnog okvira.

Ako je prag definiran kao IoU = 0.5 onda se predviđeni okvir smatra istinski pozitivnim (TP) jedino ako je njegov IoU sa pravim okvirom veći od te vrijednosti. Na slici 2.15 su prikazana tri slučaja preklapanja i njihove IoU vrijednosti.


**Slika 2.16 Primjeri preklapanja okvira i njihove IoU vrijednosti [27]**

### *2.5.10. Funkcije gubitka*

Funkcija gubitka (eng. *loss function*) služi kako bi se model mogli učiti novim podacima. Njezina glavna uloga je minimalizacija greške. Kada se greška u modelu smanjuje ova funkcija zna podesiti težine u neuronima na ispravan način. U ovom radu neće biti prikazano kako se računaju funkcije gubitka. Razlog tome je što detalji tih formula nisu dani u YOLO-ovoj dokumentaciji kako bi se izbjegla krađa intelektualnog vlasništva [28]. Bitno je samo znati da algoritam prilikom prikazivanja rezultata vraća rezultate tri funkcije gubitka koje su box_loss, cls_loss, i dfl_loss. Ukoliko te vrijednosti kroz epohe učenja padaju znači da se greška modela smanjuje.

- Box_loss pokazuje koliko dobro se predviđeni okviri preklapaju sa zadanim u setu za učenje. Računa se pomoću srednje kvadratne pogreške između parametara predviđenog okvira i istinskog okvira.
- Cls_loss (eng. *classification loss*) pokazuje koliko dobro model predviđa klasu detektiranog objekta.
- Dfl_loss (eng. *objectness loss*) daje uvid o tome koliko je model sposoban prepoznati objekt u određenoj regiji.

# 3. PRAKTIČNI DIO – RAZVIJANJE SUSTAVA STROJNOG VIDA

## 3.1. Zadatak računalnog vida i odabir YOLO modela

Računalni vid će primati izravni prijenos slike sa kamere u stvarnom vremenu. Njegov zadatak bit će detektirati signalizaciju semafora za pješake što znači da je nužno da ima mogućnost detekcije crvenog i zelenog svjetla na semaforu. Ključno je da raspoznaje razliku između crvenog i zelenog svjetla pješačkih semafora od crvenog i zelenog svjetla automobilskih semafora, te bilo kakvih drugih svjetala koja se mogu naći u cestovnom prometu (automobilska, biciklistička, svjetla na ulici..).

Postojala je mogućnost da sustav prepoznaje i pješačke semafore i automobilske semafore, ali u tom slučaju bi trebao proći puno detaljniji postupak učenja kako bi ih mogao razlikovati.

Konačni algoritam za detekciju koji je odabran je YOLOv8n model koji je na COCO skupu podataka koristeći AT100 TensorRT grafički procesor pokazao brzinu inferencije od oko 1 ms uz srednju prosječnu preciznost od oko 38% (slika 2.13). Unatoč tome što ima lošiju preciznost i manju kompleksnost od YOLOv8s modela, brži je, a to će biti bitnije prilikom implementacije na pametni telefon.

## 3.2. Baza podataka za strojno učenje

Kako bi algoritam za računalni vid imao mogućnost detekcije moramo ga naučiti tome. Za to nam je potrebna baza podataka sa slikama na kojima su okvirima označeni objekti koje želimo detektirati koji će se od sada nazivati klasama. Taj prvi skup podataka s označenim klasama je skup za učenje.

Idući skup podataka je validacijski skup. On je drugačiji od skupa za učenje no na njemu su još uvijek označene sve klase. Služi kako bismo evaluirali parametre strojnog učenja i potvrdili da sustav ne radi isključivo na podatcima iz skupa za učenje.

Treći skup je skup za ispitivanje. Na njemu se nalaze dosad nepoznati podaci (nove slike) bez ikakvih oznaka i služi isključivo za vizualnu provjeru sustava.

U slučaju ovog rada, skup za učenje sastoji se od 1026 slika, skup za validaciju od 116, dok skup za ispitivanje ne postoji već će ono biti odrađeno naknadno.

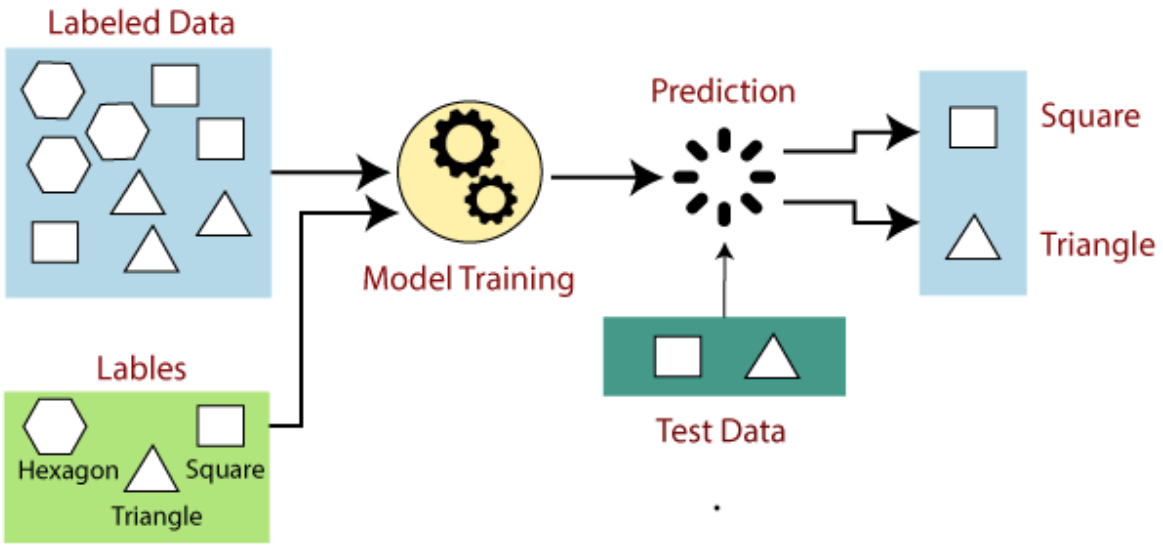
Slike koje se koriste za učenje preuzete su sa besplatne internetske baze podataka Roboflow [29]. Iz skupa podatka naknadno je uklonjena klasa „semafor“.

### 3.3. Nadgledano učenje modela

Nakon organizacije skupa podataka za učenje (slike i označeni okviri) u pripadajući format i lokacije na računalu pokreće se python skripta za učenje.

Ovaj tip strojnog učenja spada pod nadgledano učenje jer su svi podaci označeni. Minimizacija greške je srž treniranja modela u nadgledanom strojnome učenju. Kroz iterativni proces prilagodbe parametara modela, optimizacijski algoritmi smanjuju grešku između predviđenih i stvarnih vrijednosti, čime omogućuju modelu da precizno generalizira nove podatke.

Učenje se vrši kroz 300 epoha te se poslije vrednuju rezultati učenja i validacije.

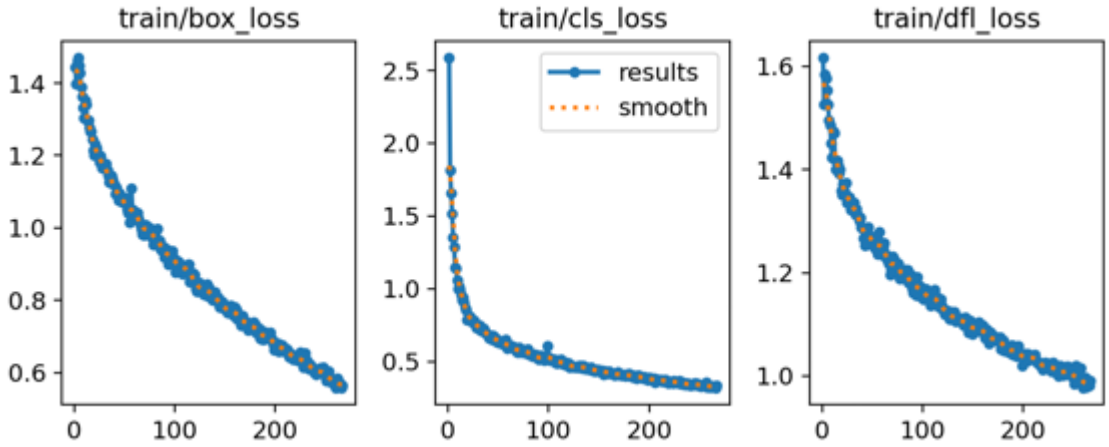



**Slika 3.1 Princip nadgledanog strojnog učenja**

## 3.4. Vrednovanje rezultata učenja

### *3.4.1. Funkcije gubitka*

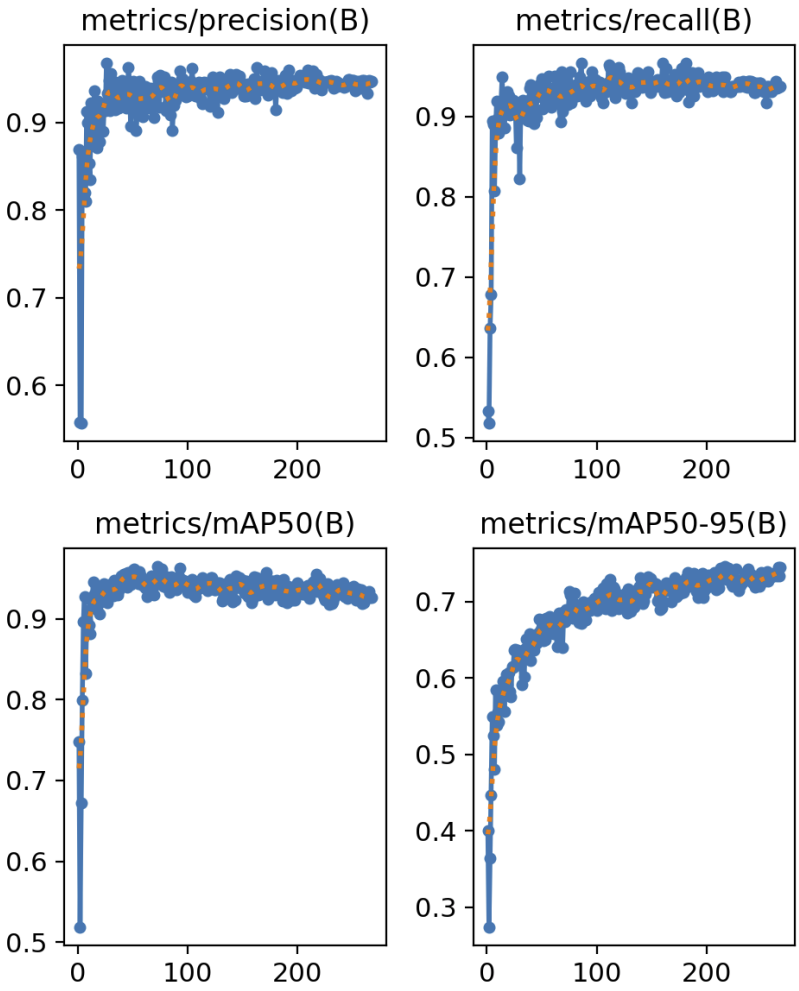
Model je sveukupno od 300 planiranih epoha učenja odradio 267. Razlog tome je što je programiran na način da ukoliko ne vidi napredak u rezultatima učenja u zadnjih 50 epoha, sam stane. Na grafovima na slici 3.2 nalaze se vrijednosti funkcija gubitaka kroz epohe učenja prikazanima u grafu. Vidljivo je da se vrijednost funkcija gubitka konstantno smanjivala bez puno skokova što ukazuje na dobro odrađeno učenje. Za funkcije box_loss i dfl_loss nagib se krenuo smanjivati, ukazujući na skoru konvergenciju, dok je funckija cls_loss praktički konvergirala. Sve u svemu rezultati funkcija gubitaka su zadovoljavajući.




**Slika 3.2 Grafovi vrijednosti funkcija gubitka kroz epohe učenja**

### *3.4.2. Preciznost, odziv i srednja prosječna preciznost (mAP@50 i mAP@50-95)*

Vrijednosti preciznosti, odziva i srednje prosječne preciznosti su konvergirale kroz jako mali broj epoha na visoke vrijednosti, kao što je vidljivo na grafovima na slici 3.3. To pokazuje da je model pouzdan i učinkovit u zadacima prepoznavanja objekata. Fluktuacije u odzivu i preciznosti sugeriraju da postoji prostor za mala poboljšanja u osiguravanju dosljednog prepoznavanja svih relevantnih instanci. Sveukupno gledano, proces treniranja se čini uspješnim.




**Slika 3.3 Grafovi vrijednosti preciznosti (gore lijevo), odziva (gore desno), mAP@50 (dolje lijevo) i mAP@50-95 (dolje desno)**

Ostale metrike ima više smisla provjeravati i vrednovati na validacijskom skupu, jer se u njemu nalaze neviđeni podaci i dat će puno bolji uvid u rad sustava.

## 3.5. Vrednovanje rezultata validacije

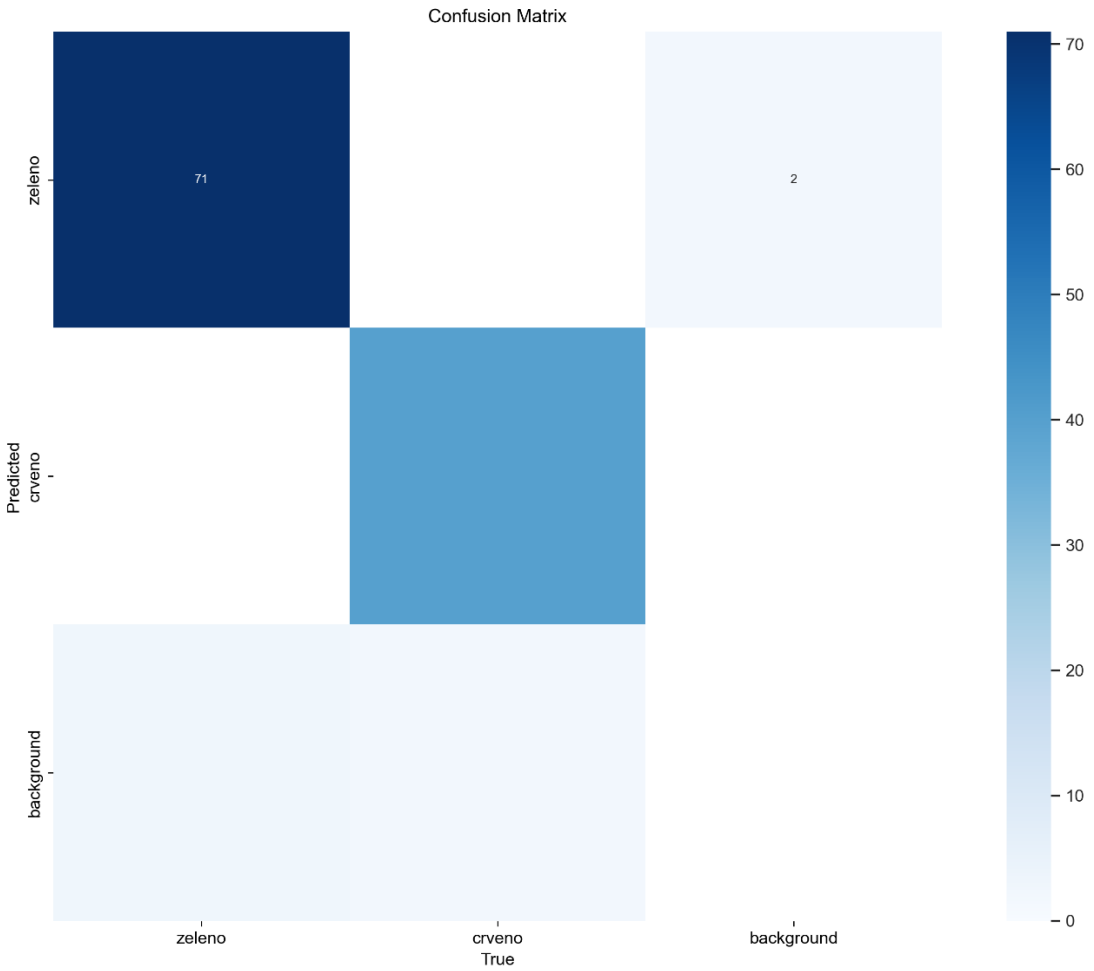
### *3.5.1. Osnovni pregled*

Model pri završetku validacije stvori kolaž slika sa označenim klasama i identičan kolaž sa detektiranim klasama. Pomoću toga se može na brzinu vidjeti rad sustava te ako su rezultati očigledno loši, ne trebaju se provjeravati ostale metrike. Na slici 3.4 u lijeva dva stupca prikazane su detekcije klasa koje je model predvidio s njihovim vrijednostima pouzdanosti, dok su u desna dva stupca prikazane klase sa svojim okvirom iz skupa podataka za validaciju. Sustav je uspješno i jako pouzdano predvidio položaje objekata.


**Slika 3.4 Detekcije modela na nekim slikama iz validacijskog skupa**

### *3.5.2. Matrica konfuzije*

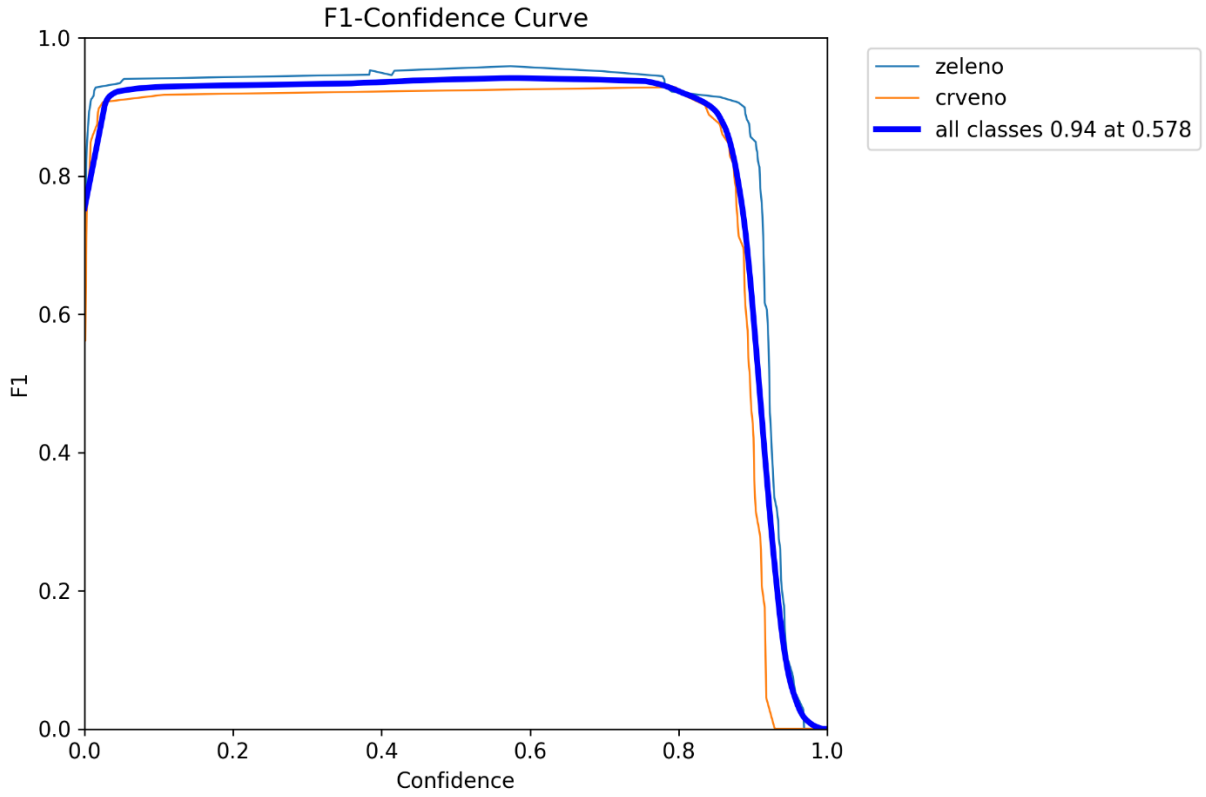
Sljedeće što je potrebno provjeriti je matrica konfuzije. Iz nje se može jasno iščitati koliko je model napravio grešaka i koje vrste. Na slici 3.5 nalazi se matrica i iz nje se može iščitati da je model točno predvidio sva crvena i zelena svjetla. Također se vidi da je model na dvije slike dao lažno pozitivni (FP) rezultat, predviđajući zeleno svjetlo tamo gdje se ono zapravo ne nalazi. U idealnom slučaju takvih grešaka ne bi bilo, te one ne bi sigurno smjele postojati u sustavu koji pomaže slijepim ljudima. Ovu grešku moguće je ukloniti proširujući set podataka za učenje. No ipak, pošto je greška jako mala, rezultati će se smatrati zadovoljavajućima. Jako je teško uočljivo na slici zbog blijede boje, ali došlo je do svega nekoliko lažno negativnih predviđanja i za crveno i za zeleno svjetlo. Iako predstavljaju problem ne moraju nužno biti uzrok za zabrinutost, jer neće predstavljati velik utjecaj prilikom kontinuiranog prijenosa slike s kamere.




**Slika 3.5 Matrica konfuzije modela**

### *3.5.3. F1 rezultat*

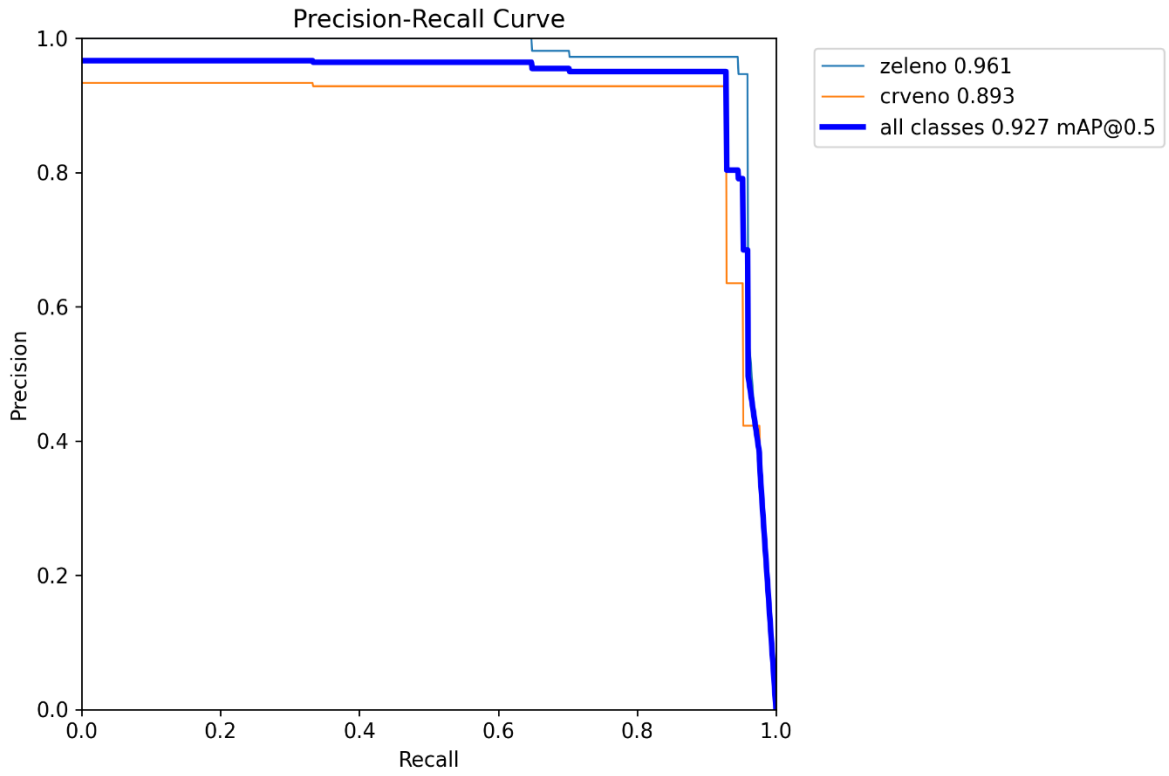
Iz grafa F1 krivulje na slici 3.6 vidljivo je da čak i za niske pragove pouzdanosti model ima poprilično konstantan odziv i preciznost. Krivulja je više-manje konstanta na intervalu od 0.1 do 0.8 što ukazuje da podešavanje praga pouzdanosti na bilo koju od tih vrijednosti će vratiti zadovoljavajuće rezultate, no optimalna vrijednost bila bi za prag pouzdanosti 0.578 gdje F1 vrijednost dostiže 0.94. Za pouzdanosti veće od 0.9 krivulja očekivano pada. Sve u svemu F1 krivulja zadovoljava zahtjeve modela. Prag pouzdanosti za detekciju u modelu bit će podešen na 0.5.




**Slika 3.6 F1 krivulja**

### *3.5.4. PR krivulja*

Iz grafa PR krivulje na slici 3.7 jasno se vidi da model radi s visokom preciznošću i odzivom za obje klase. Klasa zeleno kao i kod F1 krivulje ima bolje rezultate od klase crveno. Velika površina ispod krivulje također ukazuje na dobre performanse modela. Pri visokom odzivu krivulja očekivano pada. Rezultati ove metrike također su zadovoljavajući.




**Slika 3.7 PR krivulja**

### *3.5.5. Rezultati inferencije na realnim slikama*

Budući da je skup podataka za učenje i validaciju bio sastavljen od preuzetih slika, potrebno je dodatno provjeriti hoće li model funkcionirati u realnim uvjetima. Shodno tome, potrebno je provjeriti njegov rad na slikama (i videozapisima) koje su načinjene kamerom pametnog telefona na lokacijama na kojima će sustav raditi (križanja sa semaforima).

Slika korištena za usporedbu inferencije je dimenzija 3000x4000 piksela i ona predstavlja minimum na kojem bi sustav trebao pouzdano funkcionirati. Na njoj se nalazi pješački semafor koji signalizira crveno, a udaljen je 4 automobilske trake (oko 12m) u idealnim uvjetima osvjetljenja (oblačno, nema sjena i velikih razlika u osvjetljenju). U primjerima ispod slike su obrezane i smanjene kako ne bi zauzimale previše prostora.

Na slici 3.8 lijevo prikazana je inferencija na slici koristeći model sa istim ulaznim parametrom veličine slike na kojem je vršeno učenje (640x640 piksela). To znači da će model uzeti originalnu sliku veličine 3000x4000 i smanjit će je na 640x640. U tom procesu izgubit će se veliki broj detalja tako da neće doći do detekcije. Ova inferencija traje najkraće i iznosi 150 ms, no rezultat je neprihvatljiv jer s ovim parametrima ne dolazi do detekcije.

Lijeva slika također prikazuje rezultat još jedne inferencije, ali je parametar veličine slike koju model obrađuje ovaj put postavljen na 1440x1440 piksela. Niti ova veličina modela nije dovoljna da detektira prvo crveno svjetlo koje se nalazi preko puta ceste. Vrijeme inferencije za ovu veličinu slike je 461 ms, no rezultat je opet neprihvatljiv, jer sa zadanim parametrima ne dolazi do detekcije.

Na desnoj slici prikazana je inferencija koristeći istu sliku, ali s parametrom veličine slike koju model obrađuje postavljenu na 3008x3008 piksela. U ovom slučaju model je u mogućnosti detektirati crveno svjetlo s visokom pouzdanošću, no za to mu je potrebno 1841 ms, što ga čini nepraktičnim, posebno kada se uzme u obzir da će model obrađivati podatke na pametnom telefonu što će dodatno produžiti vrijeme obrade, stoga je i ovaj model neprihvatljiv.


**Slika 3.8 Usporedba inferencije na istoj slici s 3 različito podešena YOLOv8n modela (640x640 i 1440x1440 lijevo i 3008x3008 desno)**

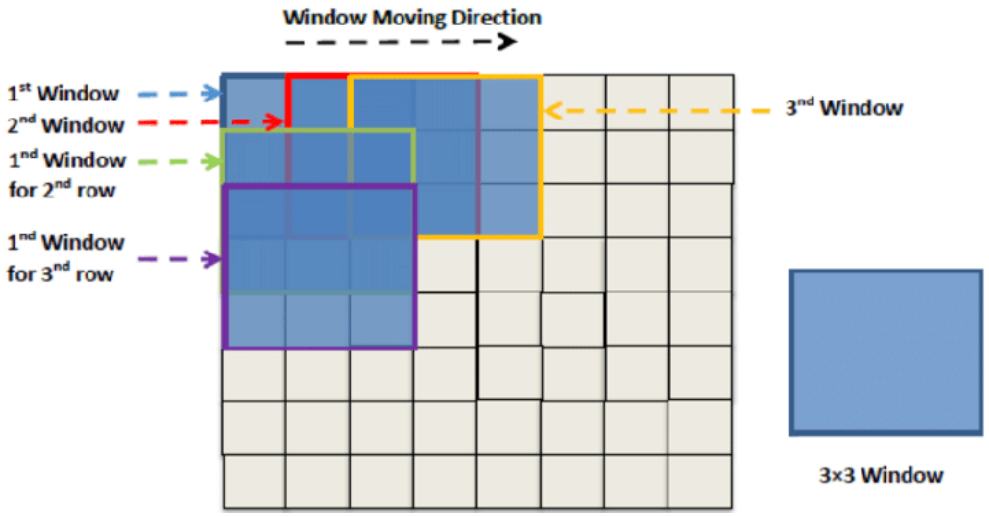
Rad modela, naravno, provjeren je na puno više slika nego što je ovdje prikazano, no u svrhu preglednosti samo je jedan primjer prikazan jer su ostali rezultati bili identični.
Pošto su se rezultati inferencije na realnim slikama pokazali nezadovoljavajućima potrebno je poduzeti mjere kako bi model funkcionirao. Spomenute mjere mogle bi podrazumijevati:

- Dodavanje mnoštva realnih slika većih dimenzija u skup podataka za učenje i ponovno učenje.
- Povećavanje dimenzija slika na kojima predmet uči i ponovno učenje (znatno povećava vrijeme učenja).
- Uvođenje metode klizećeg prozora
- Optičko uvećanje – ne dolazi do gubitka informacija, slika zadržava veličinu
- Digitalno uvećanje – dolazi do gubitka informacija, no sliku je lakše obraditi jer je manja

Kako bi se izbjeglo ponovno učenje modela problem će u biti riješen metodom klizećeg prozora.

### *3.5.6. Metoda klizećeg prozora*

Metoda klizećeg prozora (eng. *sliding window*) je metoda koja se koristi u računalnom vidu kako bi se na velikim slikama uspješno vršila detekcija objekata koji su u odnosu na veličinu slike relativno malih dimenzija [30]. Može se zamisliti prozor određenih dimenzija u gornjem lijevom kutu slike. Prozor zatim krene klizati prema desno i svakih $x$ piksela (korak u smjeru $x$) analizira odabranim modelom računalnog vida (u ovom slučaju YOLOv8n) sve što se nalazi u njemu. Kada dođe do kraja vrati se na lijevu stranu slike, no također se pomakne za $y$ piksela prema dolje (korak u smjeru $y$). To radi dok ne prođe cijelu sliku. Tim postupkom umjesto analiziranja cijele slike, analizira puno manjih slika koje zajedno čine prvobitnu sliku. Treba uzeti u obzir da ova metoda višestruko povećava vrijeme obrade zato što se sve manje slike moraju obraditi na isti način kao i cijela slika. Na slici 3.9 prikazan je princip rada metode klizećeg prozora.




**Slika 3.9 Princip rada metode klizećeg prozora [31]**

Inkremente pomaka i veličinu prozora bitno je pravilno podesiti. Ukoliko je veličina prozora prevelika može ponovno doći do lažno negativnih (FN) ishoda. Ukoliko je korak pomaka premali računalu će trebati znatno više vremena da obradi sve podatke. Također treba uzeti u obzir da korak ne bude prevelik kako ne bi došlo do presijecanja objekta kojeg želimo detektirati. Slike koje prozor uzima trebale bi se preklapati tako da nikad ne mogu presjeći predmet na pola. Cilj je naći optimalnu granicu na kojoj pouzdano detektiramo objekte i ne gubimo vrijeme nepotrebno obrađujući podatke.

Uzimajući u obzir da će svjetlo semafora za pješake zauzimati mali dio ukupne slike čak i kada mu se korisnik približi (a tada više nije ni toliko bitno svjetlo jer je korisnik skoro prešao cestu) možemo sa sigurnošću koristiti ovu metodu.

### *3.5.7. Potiskivanje ne-maksimuma*

Moguće je da s metodom klizećeg prozora isti objekt bude detektiran više puta i da algoritam želi iscrtati više okvira detekcije što bi posljedično dovelo do nereda na ekranu. Uvođenjem još jednog koraka obrade podataka može se filtrirati višak okvira tako da ostane samo onaj s najvećim pragom povjerenja.

To se radi metodom potiskivanja ne-maksimuma (eng. *non-maxima suppression, NMS*).

Funkcionira na sljedeći način:

1. Bira se okvir s najvećim pragom povjerenja.
2. Svi drugi okviri uspoređuju se s njim na način da im se gleda preklapanje, odnosno vrijednost njihovih IoU-a.

3. Ukoliko je vrijednost IoU-a veća od postavljenog praga okvir se smatra nepotrebnim te se uklanja.
4. Bira se sljedeći okvir s najvećim pragom povjerenja i metoda se ponavlja dok ne ponestane okvira za ukloniti.

### *3.5.8. Rezultati inferencije na realnim slikama s metodom klizećeg prozora*

Ispitivanje s klizećim prozorom je rađeno na istim slikama kao i prvobitno ispitivanje. Za ispitivanje je korišten parametar veličine ulazne slike u YOLOv8 model od 1440x1440 piksela. Veličina svake slike koju klizeći prozor odabere je sukladno tome 1440x1440 piksela što je optimalno jer algoritam onda ne mora trošiti vrijeme na prilagođavanje veličine slike. Korak koji prozor napravi prije uzimanja nove slike je 1000 piksela kako bi se osigurao preklop od 440 piksela u oba smjera. Pošto su objekti koji se žele detektirati mali to je sasvim dovoljna količina preklopa da se on ne nađe na presjeku slika.
Na slici 3.10. nalazi se usporedba YOLOv8 modela istog parametra veličine ulazne slike sa i bez klizećeg prozora.


**Slika 3.10 Rezultati inferencije s klizećim prozorom (lijevo) i bez (desno)**

Vidljivo je da metoda klizećim prozorom daje puno bolji rezultat. Jedini problem je što je vrijeme inferencije za sliku bez klizećeg prozora 461 ms dok je s klizećim prozorom 3.5 s. Unatoč tome što je takvo vrijeme inferencije neiskoristivo u svrhu primjene ovoga rada, treba uzeti u obzir da je korištena slika jako velikih dimenzija i stoga su se podijelile na puno više manjih slika. Kod primjene na pametnom telefonu slika će biti puno manjih dimenzija i podijeljena na minimalno dva i maksimalno četiri dijela (ovisno o daljnjem podešavanju). Ispitivanjem na pametnom telefonu ovaj pristup se pokazao boljim od pristupa bez klizećeg prozora i sa većim ulaznim parametrom veličine slike u YOLOv8 model.

Potrebno je još napomenuti kako je model provjeren i na skupu videozapisa. Videozapisi su bili različitih kvaliteta: 8K, UHD, FHD i HD. Cilj je bio optimizirati parametre koji su određeni prijašnjim korakom na malo realnijem skupu podataka. Lakše je provjeriti je li vrijeme inferencije zadovoljavajuće ako se to radi na stvarnom videozapisu, zato što daje bolji kontekst od samog broja milisekundi. Rezultati ove provjere pridonijeli su u određivanju parametara veličine ulazne slike u model, te veličine i koraka klizećeg prozora.

## 3.6. Razvijanje Android aplikacije

### *3.6.1. Općenito*

Pošto su se rezultati inferencije na fotografijama na računalu pokazali zadovoljavajućima može se prijeći na sljedeći korak što je razvijanje aplikacije za pametni telefon kako bi se inferencija mogla vršiti na njemu. Uzimajući u obzir slabiju brzinu obrade podataka na pametnim telefonima, potrebno je provjeriti hoće li model raditi jednako brzo i pouzdano kada se pokrene na pametnom telefonu.

### *3.6.2. Android studio*

Android studio je službeno integrirano razvojno okruženje (eng. *Integrated Development Environmnet, IDE*) razvijeno od strane Google-a za razvoj Android aplikacija [32]. Na tržište je u svojoj prvoj kompletnoj verziji izašao u prosincu 2014. godine [33]. Primarno koristi Kotlin ili Java programske jezike, ali i C++ je podržan.

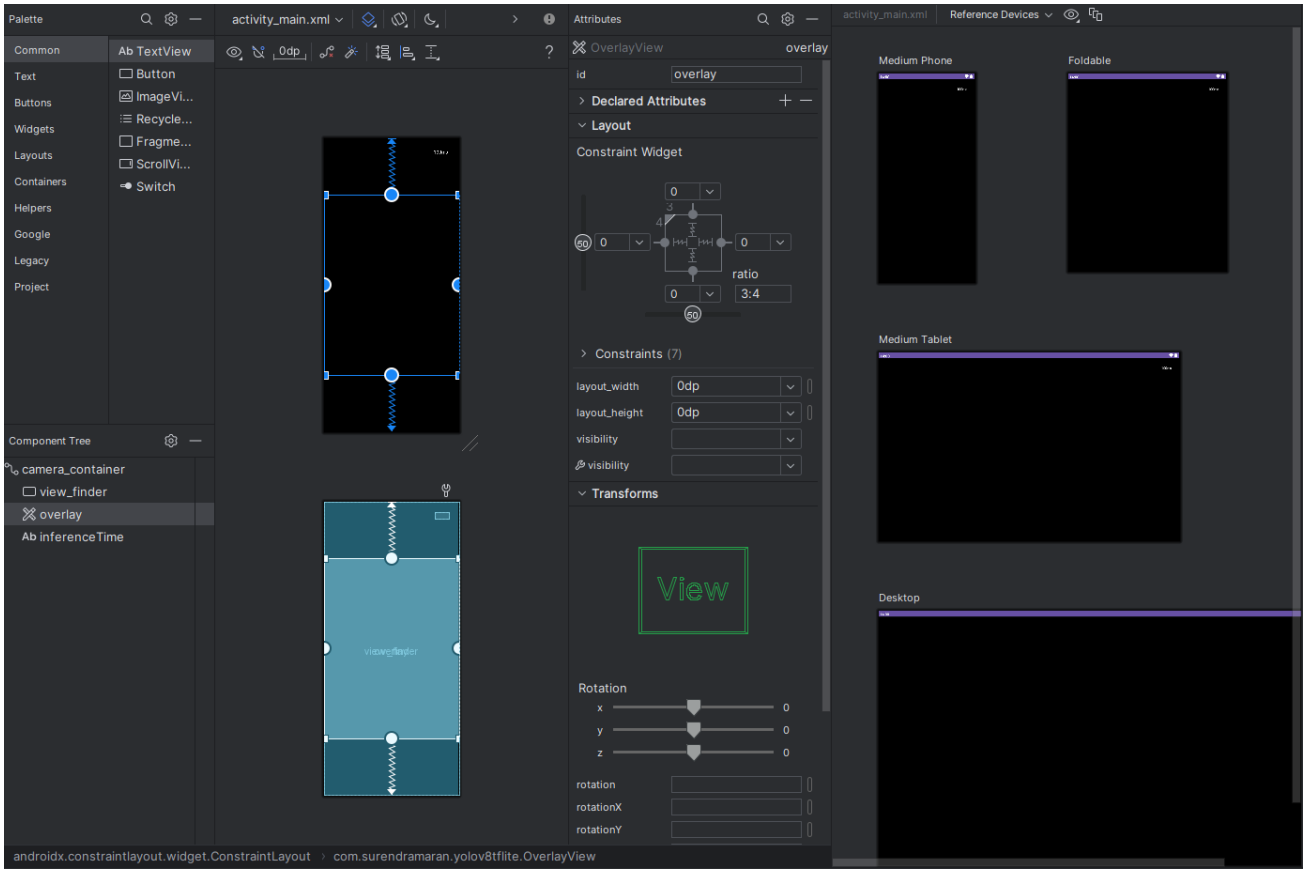
Kotlin je službeni jezik za Android razvoj i preferirani izbor zbog svoje moderne sintakse i sigurnosnih značajki. Ističe se jednostavnošću, sigurnošću i interoperabilnošću s Java ekosustavom. Google je 2019. izjavio da je to preferirani jezik za programiranje Android aplikacija [34]. Neke od najpopularnijih aplikacija kao što su *Google karte* su napisane u

Kotlinu. Kotlin podržava funkcionalne i objektno-orijentirane paradigme, te ima sintaksu koja omogućava sažeto pisanje koda. Osim toga, nudi sigurnost od null referenci i poboljšava čitljivost i održavanje koda, čineći ga omiljenim izborom među programerima.

Ukratko, Android studio je ključni alat za programere koji žele stvarati aplikacije za Android platformu, pružajući sve potrebne alate i resurse na jednom mjestu za efikasan razvoj, testiranje i distribuciju aplikacija.

Aplikacija korištena u ovom radu razvijena je na Jellyfish verziji (travanj 2024.).

Na slici 3.11 nalazi se prikaz korisničkog sučelja prilikom stvaranja korisničkog sučelja aplikacije za pametni telefon. Također je prikazana kartica koja prikazuje kako bi aplikacija izgledala na sučeljima raznih uređaja koji koriste operativni sustav Android.




**Slika 3.11 Korisničko sučelje programa Android studio**

### *3.6.3. Struktura aplikacije*

Za početak treba napomenuti kako u ovom radu aplikacija nije razvijena od početka već se nadogradila na postojeću aplikaciju preuzetu s interneta [35]. Sam kod sastoji se od 3 glavne skripte:

- **MainActivity.kt** je glavni dio koda, zaslužan za procese uključivanja i isključivanja, te ponašanja aplikacija. U njemu se inicijalizira kamera. Slika sa kamere prije nego se pošalje u idući dio programa digitalno se uvećava. Jedna instanca slike se tada šalje direktno na ekran mobitela dok druga prolazi kroz filtere za povećavanje saturacije i kontrasta nakon čega se šalje u Detector.kt.
- **Detector.kt** je dio koda koji je zaslužan za detekciju. Prvo prelazi klizećim prozorom po dobivenoj slici kako bi ju podijelio na manje komade. Zatim sliku obrađuje YOLOv8s.tflite modelom. Taj korak izvršava se pomoću grafičkog procesora (GPU) telefona kako bi se ubrzala obrada podataka. Program također mjeri vrijeme potrebno za inferenciju slike te ga šalje na zaslon. Nakon što model vrati koordinate okvira program na njima vrši NMS kako bi se uklonili nepotrebni okviri. Kada su koordinate dobivene šalju se u skriptu OverlayView.kt.
- **OverlayView.kt** je dio koda koji služi isključivo za dodavanje grafičkih elemenata na zaslon mobitela. Ulazni podaci su mu koordinate okvira s identifikacijskim brojem klase detektirane u tom okviru. Program na određenu poziciju na zaslonu crta okvir detekcije te dodatni tekstni okvir u kojem se nalazi ime klase.

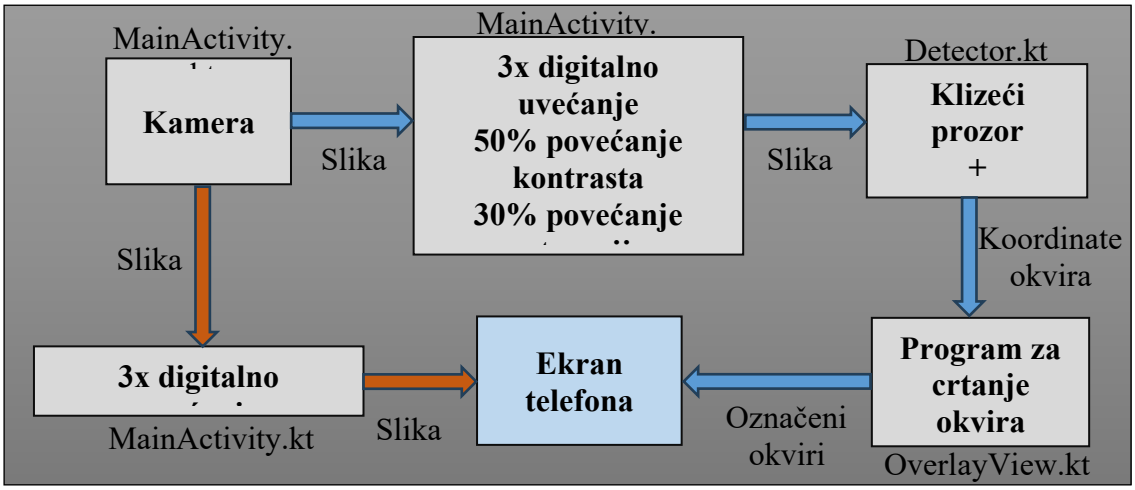
Da bi se YOLOv8 model mogao pokrenuti u aplikaciji potrebno ga je na računalu pretvoriti iz Pytorch (.pt) modela u TensorFlow Lite model (.tflite). Podaci sa kamere obrađuju se u stvarnom vremenu i na zaslonu se iscrtava okvir ukoliko je došlo do detekcije klase. Kao i na računalu, koristi se metoda klizećeg prozora radi bolje detekcije. Aplikacija također u gornjem desnom kutu ispisuje podatak o vremenu inferencije. Taj podatak služi za ocjenjivanje brzine rada modela mobitelu i ukoliko je ono preveliko potrebno je smanjiti veličinu modela.

### *3.6.4. Podešavanje parametara aplikacije*

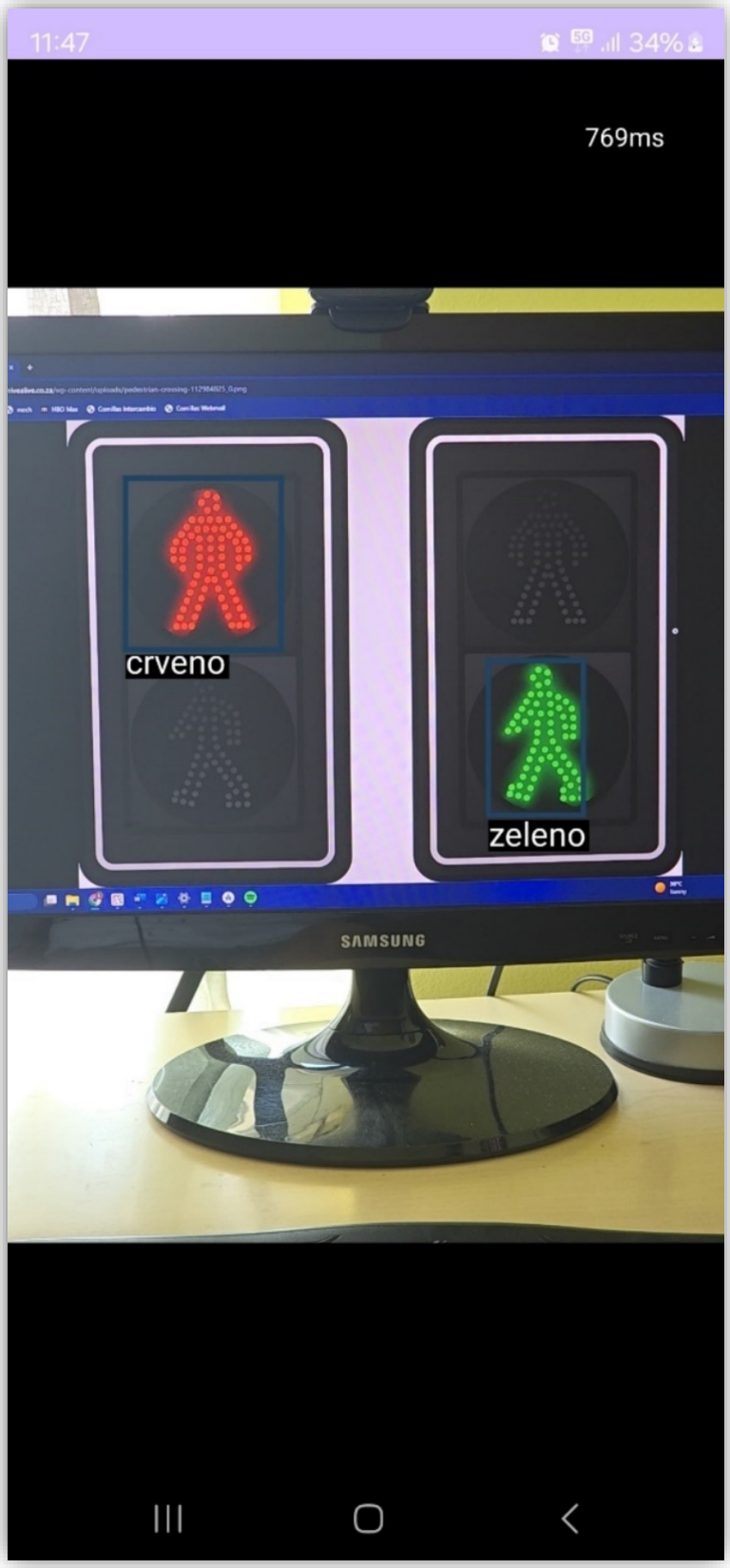
Kao što je spomenuto ranije, aplikacija koristi prednju kameru od mobitela i snima u omjeru 4:3. Također je primijenjeno trostruko digitalno uvećanje slike, kako bi se uklonile nepotrebne informacije sa slike i poboljšala detekcija. Prije slanja informacija modelu slike prolaze kroz dva filtra. Jedan im povećava kontrast dok im drugi povećava zasićenje. Ta dva filtra pokazala

su prilikom testiranja bolju detekciju zelenog svjetla bez da su utjecala na detekciju crvenog. Klizećim prozorom slika se dijeli na 4 manje koje onda YOLOv8 model analizira (parametar veličine slike je 1440x1440 piksela) i za svaku vraća informacije o lokacijama klasa, te potiskivanjem ne-maksimuma (NMS) rješava problem preklapajućih detektiranih okvira. Vrijeme inferencije mjeri se na način da program mjeri vrijeme između izlaza novih podataka iz modela. Podatci o okvirima tada se šalju u iduću funkciju koja služi za njihovu vizualizaciju na ekranu na kojem se prikazuje kontinuirani prijenos. Vrijeme inferencije ispisuje se također u gornjem desnom kutu. Prag pouzdanosti potreban za detekciju predmeta podešen je na 0.3.

Na slici 3.12. prikazan je dijagram toka aplikacije. U kutijama su označene glavne radnje i aktivnosti koje izvršavaju programi čija se imena nalaze iznad kutija. Na strelicama su označeni podaci koji se šalju između njih.




**Slika 3.12 Dijagram toka aplikacije**




**Slika 3.13 Korisničko sučelje aplikacije.**

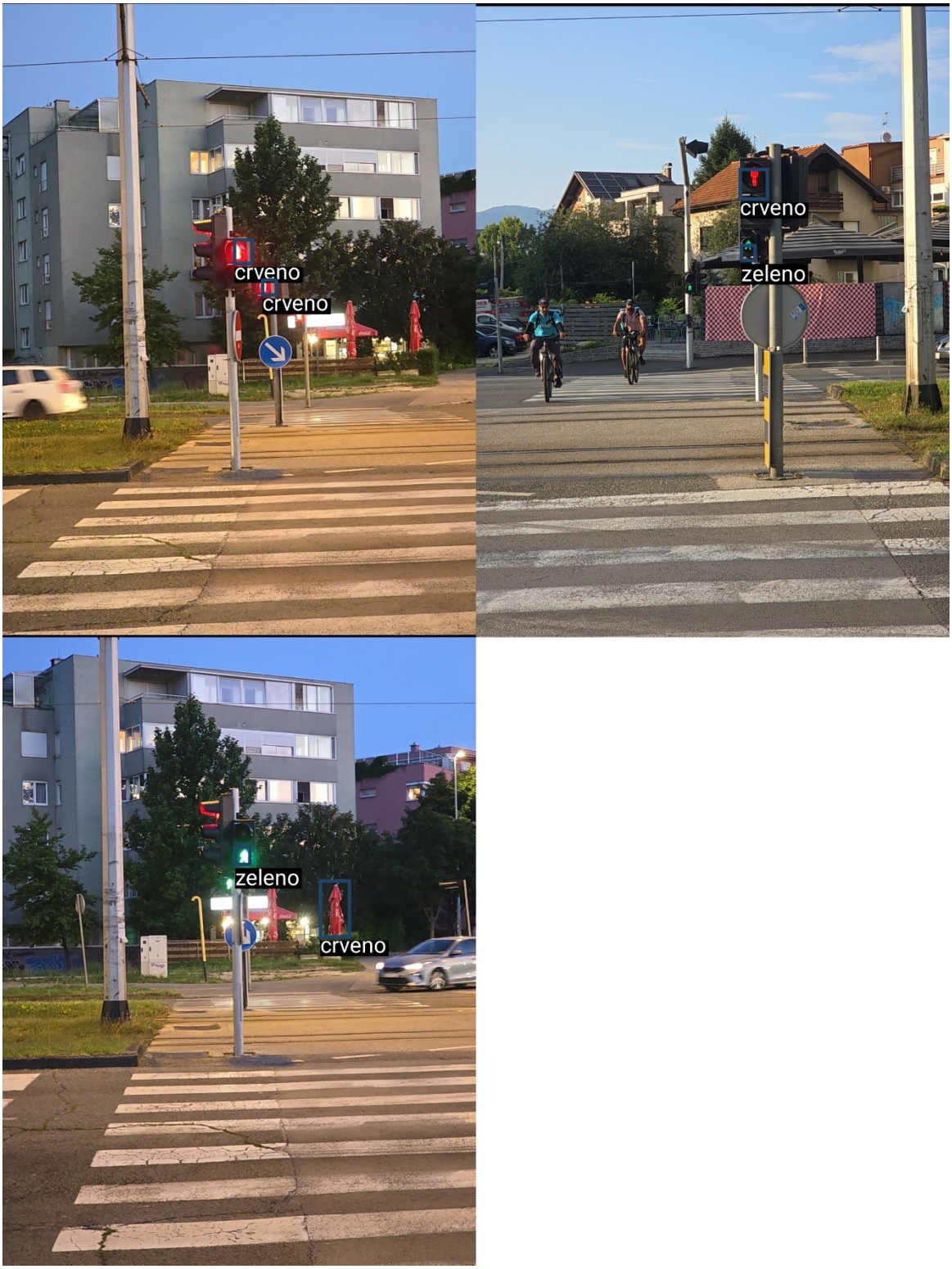
### *3.6.5. Provjera funkcionalnosti aplikacije i vrednovanje rezultata*

Kada je aplikacija dovršena instalira se na pametni telefon koji koristi operativni sustav Android. U slučaju ovog rada to je Samsung Galaxy S24 Ultra. Ispitivanja su izvršena na istim lokacijama na kojima su snimljene slike i videozapisi za prijašnja ispitivanja na računalu. Rezultati su prikazani u obliku snimaka zaslona s pametnog telefona. Također treba opet napomenuti da program crta nove okvire tek kada dobije informacije od modela ostavljajući prijašnje detektirane na ekranu unatoč pomicanju kamere. Kombinacija toga i autorove nemogućnosti da drži telefon mirno razlog su zašto okviri na slikama nisu najbolje pozicionirani, ali su blizu tome.

Na slici 3.13 nalaze se primjeri dobre detekcije. Aplikacija je uspješno prepoznala svjetlo na semaforima u zadovoljavajućem vremenskom roku (oko 850 ms). Također nije došlo do lažno pozitivnih ishoda niti je detektirala bilo koja druga semaforska svjetla osim onoga koje je bitno korisniku. Aplikacija se također pokazala robusnom jer je uspjela izvršiti uspješne detekcije na raznim semaforima u različitim uvjetima osvjetljenja.

Na slici 3.14 prikazani su neki od lošijih rezultata. Unatoč tome, zato što aplikacija radi u skoro-realnom vremenu, povremeno dođe do grešaka u detekciji ili detektiranja semafora koji nije bitan i za koji bi trebala biti dodana dodatna logika kako bi se osigurao siguran prelazak ceste. Također može doći do pojave lažno pozitivnih rezultata (npr. na slici se vidi da je aplikacija detektirala crveni suncobran kao crveno svjetlo). Premda nisu idealni, ovi slučajevi ne predstavljaju preveliki problem kod rada aplikacije zato što se u velikoj većini slučajeva otklone unutar 600-1000 ms kada se aplikacija „osvježi“. Dodavanjem sigurnosne provjere da aplikacija javlja zeleno samo kada je ono detektirano uzastopno u dvije instance bi, makar usporilo rad, spriječilo problem lažno pozitivnih rezultata. Nadalje, naknadnim ispitivanjem pokazano je da lažno pozitivni rezultati praktički nestanu kada se prag pouzdanosti podigne na vrijednost od 0.5.


**Slika 3.14 Primjeri ispravnog funkcioniranja aplikacije**




**Slika 3.15 Primjeri neispravnog funkcioniranja aplikacije**

## 3.7. Uporaba eksterne kamere

Kao što je rečeno prilikom koncepcije sustava, idealno bi bilo da funkcionira s eksternom kamerom (ne kamerom od mobitela). Tim pristupom mogli bi skup podataka za učenje temeljiti isključivo na fotografijama s iste kamere koje bi se koristila u realnim uvjetima, čime bi primjetno poboljšali metrike kao što su preciznost i odziv. Korisničko iskustvo bi generalno također bilo sličnije zato što se ne bi koristile kamere s različitim svojstvima. Korisnik bi bio sigurniji pošto ne mora izlagati svoj pametni telefon svima na ulici, a i nošenje istoga na držaču koji se nalazi na prsima ili glavi je jako nepraktično i onemogućava uporabu.

Problem položaja kamere također je već spomenut, no razmotrit će se detaljnije još jednom. Kamera može biti postavljena na glavu korisnika ili visoko na sredinu prsa. Bilo koja druga pozicija na tijelu bila bi nepouzdana zbog kretanja udova ili loše preglednosti. Čovjekov prirodni pravac kretanja podudara se s pravcem u koji gledaju njegova glava i prsa. Naravno, to ne mora uvijek biti istina, ali u realnoj situaciji niti jedna osoba neće hodati zakrenutog trupa ili glave, posebno ako zna da ima kameru na glavi čiji rad ovisi o tome da gleda ravno. Stoga se u Tablici 2. razmatraju prednosti i nedostaci svakog položaja.

**Tablica 3.1 Usporedba mogućih pozicija kamere**

| | **Kamera na glavi** | **Kamera na prsima** |
|---|---|---|
| **Prednosti** | • Bolja preglednost, visok položaj<br>• Više mogućih izvedbi: naočale, kaciga<br>• Lakše za postaviti na sebe (posebno ako je u obliku naočala)<br>• Mogućnost kontrole zakretanjem pogleda | • Čvrsto fiksirana, mala vjerojatnost ispadanja<br>• Veća vjerojatnost da će biti usmjerena paralelno s osobom.<br>• Ponovljivost položaja postavljanja<br>• Manje primjetljiva, manja vjerojatnost krađe<br>• Kraći put kabla do upravljačke jedinice |

| | | |
|---|---|---|
| **Nedostatci** | • Veća vjerojatnost lošeg usmjerenja (loše stavljeno, zakrenuti pogled osobe)<br>• Izvedba kacige može umoriti korisnika<br>• Može spasti ili zapeti za prepreku<br>• Primjetljiva, lakša za preoteti<br>• Veća nestabilnost | • Nizak položaj, lošija preglednost<br>• Veća vjerojatnost opstrukcije zbog prepreka ili drugih osoba<br>• Kompliciranije za postaviti na sebe<br>• Sam korisnik može slučajno prekriti kameru rukom ili saginjanjem |

U konačnici je odabran pristup kamere na glavi. Operativni sustav Android podržava uporabu plug-and-play kamera koristeći standardno Androidovo aplikacijsko programsko sučelje (eng. *application programming interface, API)*[36]. GoPro kamere (počevši s modelom Hero 9) imaju razvijen API i mogu se programirati [37]. Ispod ovog odjeljka dana su dva primjera vrsta kamera koje bi se mogle koristiti u svrhu realizacije ovog sustava. Bilo kakve kamere slične navedenima također bi funkcionirale, bitno je samo da zadovoljavaju gore navedene uvjete.

- Standardna web-kamera/ GoPro – Ovakve kamere trebale bi se nositi na nosaču na glavi operatera. Prednost im je što zbog svoje veličine imaju bolju kvalitetu od alternativa. Nedostatci su im što su jako vidljive i estetski neprivlačne kada se nose odmah iznad čela korisnika.
- Endoskopske kamere – ne funkcioniraju kao pravi endoskopi, to im je samo trgovačko ime. Zapravo su obične kamere koje su stavljene u valjkasto kućište s dugačkim kablom. Ova vrsta kamera je primamljiva zato što su male i neprimjetne. Primjerice, pričvršćivanjem takve kamere na naočale dobio bi se jednostavan i neprimjetan dizajn. Mana ove vrste kamera je lošija kvaliteta slike pošto su kompaktne i ne mogu imati veliki senzor.

**Slika 3.16 GoPro kamera (lijevo) [38] i endoskopska kamera (desno) [39]**

## 3.8. Analiza tržišta i mogućnosti komercijalizacije

### *3.8.1. Korisnici aplikacije*

Aplikacija bi bila prvenstveno namijenjena slijepim i slabovidnim osobama. Prema Internacionalnoj organizaciji za prevenciju sljepoće (IAPB) globalno postoji oko 43 milijuna slijepih osoba, dok je oko 295 milijuna osoba slabovidno [40]. U Europi, prema podacima Svjetske zdravstvene organizacije (WHO), ima oko 2550000 (podatak iz 2010.) slijepih osoba [41]. U Hrvatskoj, prema podacima IAPB-a, ima oko 11500 (podatak iz 2021.) [42] slijepih osoba.

### *3.8.2. Postojeća rješenja na tržištu*

- Seeing AI (Micorsoft)[43] i Google Lookout[44]: Aplikacije koje koriste kameru pametnog telefona za prepoznavanje objekata, čitanje teksta, prepoznavanje lica i opisa scena. Integrirane su s ostalim Microsoft/Google uslugama i besplatne. No, detekcija semafora nije im primarna funkcija, te se koriste kamerom od pametnog telefona.
- Envision AI[45]: Aplikacija i pametne naočale (prikazane na slici 3.16) koje koriste umjetnu inteligenciju za čitanje teksta, prepoznavanje lica, opis scena i prepoznavanje objekata. Također se može koristiti i bez naočala, te je besplatna u oba slučaja.

- OrCam MyEye[46]: Nosivi uređaj koji se pričvršćuje na naočale i koristi kameru za čitanje teksta, prepoznavanje lica i prepoznavanje objekata pomoću umjetne inteligencije. Kompaktan je, jednostavan za nošenje i neprimjetan. Mana mu je visoka cijena.

**Slika 3.17 Pametne naočale *Envision Glasses* [47]**

### *3.8.3. Zaključak analize tržišta*

Tržište već ima nekoliko rješenja koja djelomično pokrivaju potrebu za detekcijom svjetla na pješačkim semaforima, ali nijedno od njih nije u potpunosti specijalizirano za tu funkcionalnost. Postoji prostor za inovaciju i razvoj aplikacije koja bi se fokusirala isključivo na detekciju svjetla na pješačkim semaforima, pružajući visoko precizne i brze zvučne signale korisnicima.

### *3.9.* Osvrt i mogućnosti poboljšanja u budućnosti

Sustav u trenutnom stanju nije spreman za distribuciju na tržištu i uporabu u realnim uvjetima. Za početak bi se trebali uvesti standardi opreme tako da se sustav ne može koristiti ako nije dostupna određena moć obrade podataka i kamera s dovoljno velikom rezolucijom kako bi nesmetano dolazilo do detekcije. Također bi trebalo ponoviti fazu učenja, ali ovaj put puno detaljnije. To podrazumijeva proširivanje skupa podataka za učenje sa slikama koje će biti raznolikije u rezoluciji, sadržaju i kvaliteti. Učenje također mora biti postavljeno da traje više epoha. Sustav mora moći razlikovati prometna svjetla na puno većem intervalu udaljenosti i to bez ikakve mogućnosti da napravi grešku jer bi to moglo imati katastrofalne posljedice po korisnika. Ove mjere dovele bi i do smanjenja vremena inferencije zato što bi, ukoliko sustav bolje raspoznaje signalizaciju pješačkog semafora iz daljine, mogli smanjiti parametar veličine ulazne slike i izbaciti metodu klizećeg prozora koja znatno usporava postupak obrade.

Idealno bi bilo i da sustav može prepoznavati druge prometne objekte kao što je na primjer obilježeni pješački prijelaz. S dovoljno podataka sustav bi sam mogao zaključiti kada je korisnik došao do prijelaza sa semaforom.

Za aplikaciju bi trebalo razviti funkcionalno korisničko sučelje koje je jednostavno za upravljati kada osoba ne vidi ekran. Dodavanjem glasovnih naredbi i povratnih informacija slijepim osobama bilo bi omogućeno jednostavno i brzo upravljanje aplikacijom. Krajnji cilj aplikacije bila bi potpuna sposobnost pružanja navigacijskih uputa slijepim osobama. To znači da bi aplikacija morala ili imati u sebi integriranu kartu ili je dobiti na uporabu od treće stranke (kao što je Google), te pomoću GPS-a davati detaljne i precizne upute osobi gdje se kretati i kada treba imati povećani oprez jer se približava prometnici.

# 4. ZAKLJUČAK

Zbog napretka u poljima umjetne inteligencije, strojnog učenja i povećavane moći obrade podatka, budućnost u kojoj strojevi i računala preuzimaju sve veći teret posla od čovjeka postaje sve realnija. Umjetna inteligencija je pojam koji se danas, zahvaljujući svojoj generativnoj branši, spominje na praktički svakodnevnoj bazi i u laičkim grupama.

Cilj ovog rada bio je napraviti iskoristivi, prijenosni i intuitivni sustav za detekciju pomoću strojnog vida. Odlučeno je da će sustav biti izveden u obliku aplikacije za pomoć slijepim osobama pri prelasku ceste, ali ne treba zaboraviti činjenicu da se on može preobličiti da detektira što god ga se nauči. Za „mozak“ sustava odabrana je „nano“ inačica najnovijeg i najbržeg modela temeljenog na tehnologiji konvolucijskih neuronskih mreža, YOLOv8n. Model se pokazao iznimno jednostavan za implementaciju i učenje, a već je u testnoj fazi pokazivao odlične rezultate. Sustav je učenje odradio sa skupom podataka preuzetim s interneta, te je ispitan u stvarnim uvjetima rada. Kada se sustav pokazao zadovoljavajućim na računalu, krenulo se u razvoj Android aplikacije koja će omogućiti da se sustav koristi bilo kada i bilo gdje. Aplikacija je izgrađena na kosturu preuzete aplikacije, a dodane su joj funkcionalnosti predobrade slike, metoda klizećeg prozora i ubrzavanje obrade pomoću grafičkog procesora. Gotovi sustav pokazao se funkcionalnim u realnim situacijama, jednostavan za korištenje i dovoljno robustan za osnovnu primjenu.

Retrospektivno gledano, sustav je mogao biti izveden bolje da su se prilikom izrade poduzele neke dodatne mjere. Za početak, skup podataka za učenje trebao je barem sadržavati 50% slika sa kamere na kojoj je zamišljeno da sustav radi kako bi detekcija bila bolja. Činjenica da je sustav sam obustavio učenje radi prestanka napretka govori da su podaci u skupu bili međusobno preslični. Skup podataka za učenje također se trebao općenito proširiti (veći broj slika iz različitih udaljenosti) i omogućiti sustavu da detektira objekte iz daljine. To bi za posljedicu učinilo metodu klizećeg prozora nepotrebnom, te bi se vrijeme inferencije znatno smanjilo.

Sve u svemu, sustav je ostvario svoj primarni cilj i pokazao potencijal za daljnji razvoj i prilagodbu. Usprkos nekim ograničenjima, njegova upotrebljivost u stvarnim situacijama je potvrdila da je moguće izraditi učinkovit i prijenosni alat za pomoć slijepim osobama. Buduća istraživanja i unaprjeđenja trebala bi se usmjeriti na poboljšanje skupa podataka, optimizaciju aplikacije i proširenje funkcionalnosti sustava. Na taj način, slične tehnologije mogu dodatno

poboljšati kvalitetu života korisnika, istovremeno otvarajući vrata novim primjenama u raznim područjima.

# LITERATURA

[1] Reinhard Klette: *Undergraduate Topics in Computer Science, Concise Computer Vision, An Introduction into Theory and Algorithms*, Springer, London, 2014.

[2] E. R. Davies: *Signal Processing and its Applications, Machine Vision Theory, Algorithms, Practicalities*, Academic Press, 2004.

[3] *What is computer vision? (History, Applications ,Challenges),* dostupno na: https://medium.com/@ambika199820/what-is-computer-vision-history-applications-challenges-13f5759b48a5, pristupljeno 5.6.2024.

[4] Elyan E, Vuttipittayamongkol P, Johnston P, Martin K, McPherson K, Moreno-García CF, Jayne C, Mostafa Kamal Sarker M.: *Computer vision and machine learning for medical image analysis: recent advances, challenges, and way forward. Artificial intelligence surgery*, vol. 2, 2022.

[5] *What is a nerual network,* dostupno na: *https://www.ibm.com/topics/neural-networks,* pristupljeno 6.6.2024.

[6] *Everything you need to know about Neural Networks and Backpropagation — Machine Learning Easy and Fun,* dostupno na: *https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a,* pristupljeno 6.6.2024.

[7] *Artificial neuron,* dostupno na: https://www.gabormelli.com/RKB/Artificial_Neuron, pristupljeno 6.6.2024.

[8] *How Do Convolutional Layers Work in Deep Learning Neural Networks?,* dostupno na: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/, pristupljeno 8.6.2024.

[9] *What are convolutional neural networks,* dostupno na: https://www.ibm.com/topics/convolutional-neural-networks, pristupljeno 9.6.2024.

[10] *Understanding Convolutional Neural Networks (CNNs),* dostupno na: https://www.oksim.ua/2024/02/01/understanding-convolutional-neural-networks-cnns/, pristupljeno 9.6.2024.

[11] Richard Szeliski: *Computer Vision Algorithms and Applications,* Springer, London, 2011.

[12] Romanuke, Vadim: *Appropriate number and allocation of ReLUs in convolutional neural networks*, Research Bulletin of NTUU, Kyiv Polytechnic Institute, 2017.

[13] *On-line learning in neural networks with ReLU activations*, dostupno na: https://michielstraat.com/talk/mastertalk/, pristupljeno 10.6.2024.

[14] Tomislav Stipančić: *Predavanje iz kolegija Vizijski sustavi*, *Uvod u strojno učenje – CNN,* 2024.

[15] Rinku Datta Rakshit, Dakshina Ranjan Kisku, Phalguni Gupta, Jamuna Kanta Sing: *Cross-resolution face identification using deep-convolutional neural network,* Sprigner, Multimedia Tools and Applications, 2021.

[16] Ragav Venkatesan, Baoxin Li: *Convolutional Neural Networks in Visual Computing A Concise Guide,* Taylor & Francis, CRC Press, Boca Raton, London, 2017.

[17] *Fully Connected Layer vs. Convolutional Layer: Explained,* dostupno na: https://builtin.com/machine-learning/fully-connected-layer#:~:text=Fully%20Connected%20Layer%3F-,A%20fully%20connected%20layer%20refers%20to%20a%20neural%20network%20in,output%20of%20the%20output%20vector, pristupljeno 12.6.2024.

[18] *Softmax Activation Function, Everything You Need to Know,* dostupno na: https://insideaiml.com/blog/SoftMaxActivation-Function-1034, pristupljeno 12.6.2024.

[19] Dave Steinkraus, Patrice Simard, Ian Buck: *"Using GPUs for Machine Learning Algorithms",* 12th International Conference on Document Analysis and Recognition, 2005.

[20] *What is Convolutional Neural Network — CNN (Deep Learning),* dostupno na: https://nafizshahriar.medium.com/what-is-convolutional-neural-network-cnn-deep-learning-b3921bdd82d5, pristupljeno 13.6.2024.

[21] Vlastita fotografija

[22] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi: *You Only Look Once: Unified, Real-Time Object Detection,* University of Washington, Allen Institute for AI, Facebook AI Research, 2016.

[23] Srivastava, S., Divekar, A.V., Anilkumar, C. et al: *Comparative analysis of deep learning image detection algorithms,* Journal Big Data 8, čl. 66, 2021.

[24] Ceren Gülra Melek, Elena Battini Sőnmez, Songul Varlı Albayrak: *Object Detection in Shelf Images with YOLO,* IEEE EUROCON, Novi Sad, 2019.

[25] *YOLOv8 : Comprehensive Guide to State Of The Art Object Detection,* dostupno na: https://learnopencv.com/ultralytics-yolov8/, pristupljeno 14.6.2024.

[26] *What is Precision and Recall curves*, dostupno na: https://analyticsindiamag.com/topics/precision-and-recall-curves/, pristupljeno 17.6.2024.

[27] *Intersection over Union (IoU) for object detection,* dostupno na: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/, pristupljeno 17.6.2024.

[28] *Komentar Glenn Jochera, osnivača Ultralytics-a,* dostupno na: https://github.com/ultralytics/ultralytics/issues/2789#issuecomment-1560940213, pristupljeno 19.6.2024.

[29] Markus X: *Blind Assist1 Dataset,* Roboflow, dostupno na: https://universe.roboflow.com/markus-x-cgbeg/blind_assist1, pristupljeno 15.5.2024.

[30] *What is Sliding Window in Object Detection: Complete Overview of Methods & Tools,* dostupno na: https://supervisely.com/blog/how-sliding-window-improves-neural-network-models/, pristupljeno 15.6.2024.

[31] Rana Alaqil, Batool Alhumaidi ,Rahaf Alotaibi, Hafida Benhidour: *Automatic Gun Detection From Images Using Faster R-CNN,* SMARTTECH, Riyadh, 2020.

[32] *Meet Android Studio,* dostupno na: https://developer.android.com/studio/intro, pristupljeno 24.6.2024.

[33] *Google releases Android Studio 1.0, the first stable version of its IDE,* dostupno na: https://venturebeat.com/business/google-releases-android-studio-1-0-the-first-stable-version-of-its-ide/, pristupljeno 24.6.2024.

[34] *Kotlin is now Google's preferred language for Android app development,* dostupno na: https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/?guccounter=1, pristupljeno 24.6.2024.

[35] *Object detection app using YOLOv8 and Android,* dostupno na: *https://github.com/AarohiSingla/Object-Detection-Android-App,* pristupljeno 8.6.2024.

[36] *External USB Cameras,* dostupno na: https://source.android.com/docs/core/camera/external-usb-cameras, pristupljeno 29.6.2024.

[37] *Open GoPro,* dostupno na: https://gopro.com/en/us/info/open-gopro, pristupljeno 29.6.2024.

[38] *GoPro Head Strap 2.0,* dostupno na: https://gopro.com/en/us/shop/mounts-accessories/head-strap-2.0/ACHOM-002.html, pristupljeno 29.6.2024.

[39] *USB C Endoskopska kamera,* dostupno na: https://www.amazon.com/Inspection-Fantronics-Waterproof-Borescope-Adjustable/dp/B071HYRPND, pristupljeno 30.6.2024.

[40] Bourne R, Steinmetz J, Faxman S, et al.: *Trends in prevalence of blindness and distance and near vision impairment over 30 years: an analysis for the Global Burden of Disease Study,* Lancet Global Health, pristupljeno pomoću IAPB Vision Atlas-a 2020.

[41] *About Blindness and Partial Sight, Facts and Figures,* dostupno na: *https://www.euroblind.org/about-blindness-and-partial-sight/facts-and-figures#details,* pristupljeno 26.6.2024

[42] *IAPB Vision Atlas,* dostupno na: https://www.iapb.org/learn/vision-atlas/magnitude-and-projections/countries/croatia/, pristupljeno 26.6.2024.

[43] *Seeing AI,* dostupno na: https://www.seeingai.com/, pristupljeno 26.6.2024.

[44] *Lookout: an app to help blind and visually impaired people learn about their surroundings,* dostupno na: https://blog.google/outreach-initiatives/accessibility/lookout-app-help-blind-and-visually-impaired-people-learn-about-their-surroundings/, pristupljeno 26.6.2024.

[45] *Envision App,* dostupno na: https://www.letsenvision.com/app, pristupljeno 26.6.2024.

[46] *OrCam MyEye 3 Pro,* dostupno na: https://www.orcam.com/en-us/orcam-myeye-3-pro, pristupljeno 26.6.2024.

[47] *Envision Glasses,* dostupno na: https://visionmatters.net/product/envision-glasses/, pristupljeno 26.6.2024.

# PRILOG – PROGRAMSKI KOD

## 1. Main.py

```
import os
import torch
torch. cuda. empty_cache()
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"

from ultralytics import YOLO

model = YOLO('yolov8n.pt')

results = model.train(data="config.yaml", device=0, workers=0,
epochs=300)
```

## 2. Detection.py

```
from ultralytics import YOLO
import cv2
import numpy as np
sl_size = 1440
sl_stride = 1000

model = YOLO('C:/Users/lukab/OneDrive/Radna
površina/Faks/!!!!DIPLOMSKI/Strojni vid - Semafori_Online -
kopija/runs/detect/train5/weights/last.pt')

# klizeći prozor
def sliding_window(image, model, window_size=sl_size,
stride=sl_stride):
    height, width, _ = image.shape
    detections = []

    for y in range(0, height, stride):
        for x in range(0, width, stride):
            window = image[y:y+window_size, x:x+window_size]

            if window.shape[0] != window_size or window.shape[1] !=
window_size:
                window = cv2.copyMakeBorder(window, 0, window_size -
window.shape[0], 0, window_size - window.shape[1], cv2.BORDER_CONSTANT,
value=(0,0,0))

            # inferencija na prozoru
            results = model(window,imgsz=1080)
            for result in results:
```

```
                # Koordinate okvira za prikaz na originalnoj slici
                for box, score, cls in
zip(result.boxes.xyxy.cpu().numpy(), result.boxes.conf.cpu().numpy(),
result.boxes.cls.cpu().numpy().astype(int)):
                    x1, y1, x2, y2 = box
                    width_box = x2 - x1
                    height_box = y2 - y1
                    adjusted_box = [x1 + x, y1 + y, width_box,
height_box, score, cls]
                    detections.append(adjusted_box)

    return detections

# Non-Maximum Suppression
def apply_nms(detections, iou_threshold=0.1):
    if len(detections) == 0:
        return []

    boxes = np.array([d[:4] for d in detections])
    scores = np.array([d[4] for d in detections])
    class_ids = np.array([d[5] for d in detections])

    indices = cv2.dnn.NMSBoxes(boxes.tolist(), scores.tolist(),
score_threshold=0.533, nms_threshold=iou_threshold)
    indices = indices.flatten() if len(indices) > 0 else []

    final_detections = [detections[i] for i in indices]
    return final_detections

# Učitaj sliku
image_path = '3.jpg'
image = cv2.imread(image_path)

window_size = sl_size
stride = sl_stride
detections = sliding_window(image, model, window_size, stride)

final_detections = apply_nms(detections)

# Crtanje prozora
for box in final_detections:
    x, y, width_box, height_box, score, cls_id = box
    x2, y2 = x + width_box, y + height_box
    label = f"{model.names[int(cls_id)]}: {score:.2f}"
    color = (0, 0, 255)
    cv2.rectangle(image, (int(x), int(y)), (int(x2), int(y2)), color,
5)
```

```
    cv2.putText(image, label, (int(x), int(y) - 10),
cv2.FONT_HERSHEY_SIMPLEX, 5, color, 15)

# Dimenzije prikaza
window_width = 900
window_height = 1000
image_height, image_width = image.shape[:2]
aspect_ratio = image_width / image_height

if image_width > window_width or image_height > window_height:
    if aspect_ratio > 1:
        new_width = window_width
        new_height = int(window_width / aspect_ratio)
    else:
        new_height = window_height
        new_width = int(window_height * aspect_ratio)
    resized_image = cv2.resize(image, (new_width, new_height))
else:
    resized_image = image

cv2.imshow('Detected Image', resized_image)
cv2.waitKey(0)
cv2.destroyAllWindows()

print(final_detections)
```

### 3. MainActivity.kt

```
package com.surendramaran.yolov8tflite

import android.Manifest
import android.content.pm.PackageManager
import android.graphics.Bitmap
import android.graphics.Canvas
import android.graphics.ColorMatrix
import android.graphics.ColorMatrixColorFilter
import android.graphics.Matrix
import android.graphics.Paint
import android.os.Bundle
import android.util.Log
import androidx.activity.result.contract.ActivityResultContracts
import androidx.appcompat.app.AppCompatActivity
import androidx.camera.core.AspectRatio
import androidx.camera.core.Camera
import androidx.camera.core.CameraSelector
import androidx.camera.core.ImageAnalysis
import androidx.camera.core.Preview
import androidx.camera.lifecycle.ProcessCameraProvider
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import com.surendramaran.yolov8tflite.Constants.LABELS_PATH
```

```
import com.surendramaran.yolov8tflite.Constants.MODEL_PATH
import com.surendramaran.yolov8tflite.databinding.ActivityMainBinding
import java.util.concurrent.ExecutorService
import java.util.concurrent.Executors

class MainActivity : AppCompatActivity(), Detector.DetectorListener {
    private lateinit var binding: ActivityMainBinding
    private val isFrontCamera = false

    private var preview: Preview? = null
    private var imageAnalyzer: ImageAnalysis? = null
    private var camera: Camera? = null
    private var cameraProvider: ProcessCameraProvider? = null
    private lateinit var detector: Detector

    private lateinit var cameraExecutor: ExecutorService

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        detector = Detector(baseContext, MODEL_PATH, LABELS_PATH, this)
        detector.setup()

        if (allPermissionsGranted()) {
            startCamera()
        } else {
            ActivityCompat.requestPermissions(this,
REQUIRED_PERMISSIONS, REQUEST_CODE_PERMISSIONS)
        }

        cameraExecutor = Executors.newSingleThreadExecutor()
    }

    private fun startCamera() {
        val cameraProviderFuture =
ProcessCameraProvider.getInstance(this)
        cameraProviderFuture.addListener({
            cameraProvider  = cameraProviderFuture.get()
            bindCameraUseCases()
        }, ContextCompat.getMainExecutor(this))
    }

    private fun bindCameraUseCases() {
        val cameraProvider = cameraProvider ?: throw
IllegalStateException("Camera initialization failed.")

        val rotation = binding.viewFinder.display.rotation

        val cameraSelector = CameraSelector
            .Builder()
            .requireLensFacing(CameraSelector.LENS_FACING_BACK)
            .build()

        preview = Preview.Builder()
            .setTargetAspectRatio(AspectRatio.RATIO_4_3)
```

```
            .setTargetRotation(rotation)
            .build()

        imageAnalyzer = ImageAnalysis.Builder()
            .setTargetAspectRatio(AspectRatio.RATIO_4_3)

.setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
            .setTargetRotation(binding.viewFinder.display.rotation)

.setOutputImageFormat(ImageAnalysis.OUTPUT_IMAGE_FORMAT_RGBA_8888)
            .build()

        imageAnalyzer?.setAnalyzer(cameraExecutor) { imageProxy ->
            val bitmapBuffer = Bitmap.createBitmap(
                imageProxy.width,
                imageProxy.height,
                Bitmap.Config.ARGB_8888
            )
            imageProxy.use {
bitmapBuffer.copyPixelsFromBuffer(imageProxy.planes[0].buffer) }
            imageProxy.close()

            val matrix = Matrix().apply {

postRotate(imageProxy.imageInfo.rotationDegrees.toFloat())

                if (isFrontCamera) {
                    postScale(
                        -1f,
                        1f,
                        imageProxy.width.toFloat(),
                        imageProxy.height.toFloat()
                    )
                }
            }

            val rotatedBitmap = Bitmap.createBitmap(
                bitmapBuffer, 0, 0, bitmapBuffer.width,
bitmapBuffer.height,
                matrix, true
            )

            val contrastSaturationBitmap =
adjustContrastAndSaturation(rotatedBitmap, 1.3f, 1.5f) // Adjust
contrast and saturation values as needed

            detector.detect(contrastSaturationBitmap)
        }

        cameraProvider.unbindAll()

        try {
            camera = cameraProvider.bindToLifecycle(
                this,
                cameraSelector,
                preview,
                imageAnalyzer
```

```
            )

            camera?.cameraControl?.setZoomRatio(3f)


preview?.setSurfaceProvider(binding.viewFinder.surfaceProvider)
        } catch (exc: Exception) {
            Log.e(TAG, "Use case binding failed", exc)
        }
    }

    private fun allPermissionsGranted() = REQUIRED_PERMISSIONS.all {
        ContextCompat.checkSelfPermission(baseContext, it) ==
PackageManager.PERMISSION_GRANTED
    }

    private val requestPermissionLauncher = registerForActivityResult(
        ActivityResultContracts.RequestMultiplePermissions()) {
        if (it[Manifest.permission.CAMERA] == true) { startCamera() }
    }

    override fun onDestroy() {
        super.onDestroy()
        detector.clear()
        cameraExecutor.shutdown()
    }

    override fun onResume() {
        super.onResume()
        if (allPermissionsGranted()){
            startCamera()
        } else {
            requestPermissionLauncher.launch(REQUIRED_PERMISSIONS)
        }
    }

    companion object {
        private const val TAG = "Camera"
        private const val REQUEST_CODE_PERMISSIONS = 10
        private val REQUIRED_PERMISSIONS = mutableListOf (
            Manifest.permission.CAMERA
        ).toTypedArray()
    }

    override fun onEmptyDetect() {
        binding.overlay.invalidate()
    }

    override fun onDetect(boundingBoxes: List<BoundingBox>,
inferenceTime: Long) {
        runOnUiThread {
            binding.inferenceTime.text = "${inferenceTime}ms"
            binding.overlay.apply {
                setResults(boundingBoxes)
                invalidate()
            }
        }
```

```
    }

    private fun adjustContrastAndSaturation(bitmap: Bitmap, contrast:
Float, saturation: Float): Bitmap {
        val colorMatrix = ColorMatrix().apply {

            set(arrayOf(
                contrast, 0f, 0f, 0f, 128 * (1 - contrast),
                0f, contrast, 0f, 0f, 128 * (1 - contrast),
                0f, 0f, contrast, 0f, 128 * (1 - contrast),
                0f, 0f, 0f, 1f, 0f
            ).toFloatArray())


            val saturationMatrix = ColorMatrix()
            saturationMatrix.setSaturation(saturation)
            postConcat(saturationMatrix)
        }

        val paint = Paint().apply {
            colorFilter = ColorMatrixColorFilter(colorMatrix)
        }

        val adjustedBitmap = Bitmap.createBitmap(bitmap.width,
bitmap.height, bitmap.config)
        val canvas = Canvas(adjustedBitmap)
        canvas.drawBitmap(bitmap, 0f, 0f, paint)

        return adjustedBitmap
    }
}
```

### 4. Detector.kt

```
package com.surendramaran.yolov8tflite

import android.content.Context
import android.graphics.Bitmap
import android.os.SystemClock
import org.tensorflow.lite.DataType
import org.tensorflow.lite.Interpreter
import org.tensorflow.lite.gpu.CompatibilityList
import org.tensorflow.lite.gpu.GpuDelegate
import org.tensorflow.lite.support.common.FileUtil
import org.tensorflow.lite.support.common.ops.CastOp
import org.tensorflow.lite.support.common.ops.NormalizeOp
import org.tensorflow.lite.support.image.ImageProcessor
import org.tensorflow.lite.support.image.TensorImage
import org.tensorflow.lite.support.tensorbuffer.TensorBuffer
import java.io.BufferedReader
import java.io.IOException
import java.io.InputStream
import java.io.InputStreamReader

class Detector(
    private val context: Context,
    private val modelPath: String,
```

```
    private val labelPath: String,
    private val detectorListener: DetectorListener
) {

    private var interpreter: Interpreter? = null
    private var labels = mutableListOf<String>()

    private var tensorWidth = 0
    private var tensorHeight = 0
    private var numChannel = 0
    private var numElements = 0

    private val imageProcessor = ImageProcessor.Builder()
        .add(NormalizeOp(INPUT_MEAN, INPUT_STANDARD_DEVIATION))
        .add(CastOp(INPUT_IMAGE_TYPE))
        .build()

    fun setup() {
        val compatList = CompatibilityList()
        val model = FileUtil.loadMappedFile(context, modelPath)
        val options = Interpreter.Options().apply{
            if(compatList.isDelegateSupportedOnThisDevice){
                val delegateOptions = compatList.bestOptionsForThisDevice
                this.addDelegate(GpuDelegate(delegateOptions))
            } else {
                this.setNumThreads(4)
            }
        }
        interpreter = Interpreter(model, options)

        val inputShape = interpreter?.getInputTensor(0)?.shape() ?: return
        val outputShape = interpreter?.getOutputTensor(0)?.shape() ?:
return

        tensorWidth = inputShape[1]
        tensorHeight = inputShape[2]
        numChannel = outputShape[1]
        numElements = outputShape[2]

        try {
            val inputStream: InputStream = context.assets.open(labelPath)
            val reader = BufferedReader(InputStreamReader(inputStream))

            var line: String? = reader.readLine()
            while (line != null && line != "") {
                labels.add(line)
                line = reader.readLine()
            }

            reader.close()
            inputStream.close()
        } catch (e: IOException) {
            e.printStackTrace()
        }
    }

    fun clear() {
        interpreter?.close()
        interpreter = null
    }
```

```
    fun detect(frame: Bitmap) {
        interpreter ?: return
        if (tensorWidth == 0) return
        if (tensorHeight == 0) return
        if (numChannel == 0) return
        if (numElements == 0) return

        var inferenceTime = SystemClock.uptimeMillis()

        val strideX = (tensorWidth / 1.3).toInt()
        val strideY = (tensorHeight / 1.3).toInt()

        val windows = mutableListOf<Bitmap>()
        for (y in 0 until frame.height step strideY) {
            for (x in 0 until frame.width step strideX) {
                val right = if (x + tensorWidth > frame.width) frame.width
else x + tensorWidth
                val bottom = if (y + tensorHeight > frame.height)
frame.height else y + tensorHeight
                windows.add(Bitmap.createBitmap(frame, x, y, right - x,
bottom - y))
            }
        }

        val allBoundingBoxes = mutableListOf<BoundingBox>()

        for (window in windows) {
            val resizedBitmap = Bitmap.createScaledBitmap(window,
tensorWidth, tensorHeight, false)

            val tensorImage = TensorImage(DataType.FLOAT32)
            tensorImage.load(resizedBitmap)
            val processedImage = imageProcessor.process(tensorImage)
            val imageBuffer = processedImage.buffer

            val output = TensorBuffer.createFixedSize(intArrayOf(1 ,
numChannel, numElements), OUTPUT_IMAGE_TYPE)
            interpreter?.run(imageBuffer, output.buffer)

            val bestBoxes = bestBox(output.floatArray)
            if (bestBoxes != null) {
                allBoundingBoxes.addAll(bestBoxes)
            }
        }

        inferenceTime = SystemClock.uptimeMillis() - inferenceTime

        if (allBoundingBoxes.isEmpty()) {
            detectorListener.onEmptyDetect()
            return
        }

        val finalBoxes = applyNMS(allBoundingBoxes)
        detectorListener.onDetect(finalBoxes, inferenceTime)
    }

    private fun bestBox(array: FloatArray) : List<BoundingBox>? {

        val boundingBoxes = mutableListOf<BoundingBox>()
```

```
        for (c in 0 until numElements) {
            var maxConf = -1.0f
            var maxIdx = -1
            var j = 4
            var arrayIdx = c + numElements * j
            while (j < numChannel){
                if (array[arrayIdx] > maxConf) {
                    maxConf = array[arrayIdx]
                    maxIdx = j - 4
                }
                j++
                arrayIdx += numElements
            }

            if (maxConf > CONFIDENCE_THRESHOLD) {
                val clsName = labels[maxIdx]
                val cx = array[c] // 0
                val cy = array[c + numElements] // 1
                val w = array[c + numElements * 2]
                val h = array[c + numElements * 3]
                val x1 = cx - (w/2F)
                val y1 = cy - (h/2F)
                val x2 = cx + (w/2F)
                val y2 = cy + (h/2F)
                if (x1 < 0F || x1 > 1F) continue
                if (y1 < 0F || y1 > 1F) continue
                if (x2 < 0F || x2 > 1F) continue
                if (y2 < 0F || y2 > 1F) continue

                boundingBoxes.add(
                    BoundingBox(
                        x1 = x1, y1 = y1, x2 = x2, y2 = y2,
                        cx = cx, cy = cy, w = w, h = h,
                        cnf = maxConf, cls = maxIdx, clsName = clsName
                    )
                )
            }
        }

        if (boundingBoxes.isEmpty()) return null

        return applyNMS(boundingBoxes)
    }

    private fun applyNMS(boxes: List<BoundingBox>) :
MutableList<BoundingBox> {
        val sortedBoxes = boxes.sortedByDescending { it.cnf
}.toMutableList()
        val selectedBoxes = mutableListOf<BoundingBox>()

        while(sortedBoxes.isNotEmpty()) {
            val first = sortedBoxes.first()
            selectedBoxes.add(first)
            sortedBoxes.remove(first)

            val iterator = sortedBoxes.iterator()
            while (iterator.hasNext()) {
                val nextBox = iterator.next()
                val iou = calculateIoU(first, nextBox)
```

```
                if (iou >= IOU_THRESHOLD) {
                    iterator.remove()
                }
            }
        }

        return selectedBoxes
    }

    private fun calculateIoU(box1: BoundingBox, box2: BoundingBox): Float {
        val x1 = maxOf(box1.x1, box2.x1)
        val y1 = maxOf(box1.y1, box2.y1)
        val x2 = minOf(box1.x2, box2.x2)
        val y2 = minOf(box1.y2, box2.y2)
        val intersectionArea = maxOf(0F, x2 - x1) * maxOf(0F, y2 - y1)
        val box1Area = box1.w * box1.h
        val box2Area = box2.w * box2.h
        return intersectionArea / (box1Area + box2Area - intersectionArea)
    }

    interface DetectorListener {
        fun onEmptyDetect()
        fun onDetect(boundingBoxes: List<BoundingBox>, inferenceTime: Long)
    }

    companion object {
        private const val INPUT_MEAN = 0f
        private const val INPUT_STANDARD_DEVIATION = 255f
        private val INPUT_IMAGE_TYPE = DataType.FLOAT32
        private val OUTPUT_IMAGE_TYPE = DataType.FLOAT32
        private const val CONFIDENCE_THRESHOLD = 0.5F
        private const val IOU_THRESHOLD = 0.1F
    }
}
```

### 5. OverlayView.kt

```
package com.surendramaran.yolov8tflite

import android.content.Context
import android.graphics.Canvas
import android.graphics.Color
import android.graphics.Paint
import android.graphics.Rect
import android.util.AttributeSet
import android.view.View
import androidx.core.content.ContextCompat

class OverlayView(context: Context?, attrs: AttributeSet?) : View(context,
attrs) {

    private var results = listOf<BoundingBox>()
    private var boxPaint = Paint()
    private var textBackgroundPaint = Paint()
    private var textPaint = Paint()

    private var bounds = Rect()

    init {
```

```
        initPaints()
    }

    fun clear() {
        textPaint.reset()
        textBackgroundPaint.reset()
        boxPaint.reset()
        invalidate()
        initPaints()
    }

    private fun initPaints() {
        textBackgroundPaint.color = Color.BLACK
        textBackgroundPaint.style = Paint.Style.FILL
        textBackgroundPaint.textSize = 50f

        textPaint.color = Color.WHITE
        textPaint.style = Paint.Style.FILL
        textPaint.textSize = 50f

        boxPaint.color = ContextCompat.getColor(context!!,
R.color.bounding_box_color)
        boxPaint.strokeWidth = 8F
        boxPaint.style = Paint.Style.STROKE
    }

    override fun draw(canvas: Canvas) {
        super.draw(canvas)

        results.forEach {
            val left = it.x1 * width
            val top = it.y1 * height
            val right = it.x2 * width
            val bottom = it.y2 * height

            canvas.drawRect(left, top, right, bottom, boxPaint)
            val drawableText = it.clsName
            textBackgroundPaint.getTextBounds(drawableText, 0,
drawableText.length, bounds)
            val textWidth = bounds.width()
            val textHeight = bounds.height()
            canvas.drawRect(
                left,
                bottom + BOUNDING_RECT_TEXT_PADDING,
                left + textWidth + BOUNDING_RECT_TEXT_PADDING,
                bottom + textHeight + 2 * BOUNDING_RECT_TEXT_PADDING,
                textBackgroundPaint
            )
            canvas.drawText(drawableText, left, bottom + textHeight +
BOUNDING_RECT_TEXT_PADDING, textPaint)
        }
    }
    fun setResults(boundingBoxes: List<BoundingBox>) {
        results = boundingBoxes
        invalidate()
    }
    companion object {
        private const val BOUNDING_RECT_TEXT_PADDING = 8
    }
}
```