Detekcija i klasifikacija vozila i pješaka korištenjem Haar kaskadnog klasifikatora

Batur, Jure

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Jure Batur

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor:

Izv. prof. dr. sc. Tomislav Stipančić

Student:

Jure Batur

Zagreb, 2024.

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

Zahvaljujem se mentoru dr. sc. Tomislavu Stipančiću na pruženoj pomoći, strpljenju, uloženom vremenu i suradnji tijekom izrade ovog rada.

Zahvaljujem se obitelji koji su mi cijelo vrijeme bili podrška za vrijeme trajanja studija kao i mojim prijateljima i kolegama s fakulteta koji su mi mnogo pomogli i olakšali studiranje.

Jure Batur

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 24 – 06 / 1**	
Ur.broj: **15 – 24 –**	

ZAVRŠNI ZADATAK

Student: **Jure Batur** JMBAG: **0035226895**

Naslov rada na hrvatskom jeziku: **Detekcija i klasifikacija vozila i pješaka korištenjem haar kaskadnog klasifikatora**

Naslov rada na engleskom jeziku: **Detection and classification of vehicles and pedestrians using haar cascade classifier**

Opis zadatka:

Svojstva te kvaliteta rada računalnog modela uvelike ovisi o kvaliteti korištenih podataka u fazi treninga te metodi strojnog učenja koja je pritom korištena. Strojni vid je aktivno istraživačko područje već dugi niz godina, te igra vitalnu ulogu kod sustava za upravljanje i nadzor prometa. U tim sustavima algoritmi i računalni modeli za detekciju i klasifikaciju objekata i živih bića pronalaze vrlo važne primjene. Skup podataka za trening je ovdje vrlo često videozapis prometnica u urbanim okruženjima snimljen u raznim gradovima što u pravilu vodi ka izradi robusnih klasifikatora.
Haar kaskada je pristup temeljen na strojnom učenju, gdje se kaskadna funkcija trenira iz pozitivnih i negativnih slika. Nakon faze treninga računalni model se potom koristi na proizvodnom skupu slika prilikom prepoznavanja i klasifikacije. Kaskada uključuje nekoliko slabih klasifikatora, što finalnom klasifikatoru daje snagu i robusnost da klasificira više značajki na slici.

U radu je potrebno:

- istražiti dostupne klasifikatore za detekciju i klasifikaciju
- razviti programsko rješenje te primijeniti klasifikator temeljen na haar kaskadama kod prepoznavanja automobila, vozila na dva kotača, autobusa i pješaka
- testirati razvijeno programsko rješenje koristeći dostupne video materijale.

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

Zadatak zadan: 30. 11. 2023.

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

Datum predaje rada:
1. rok: 22. i 23. 2. 2024.
2. rok (izvanredni): 11. 7. 2024.
3. rok: 19. i 20. 9. 2024.

Predviđeni datumi obrane:
1. rok: 26. 2. – 1. 3. 2024.
2. rok (izvanredni): 15. 7. 2024.
3. rok: 23. 9. – 27. 9. 2024.

Predsjednik Povjerenstva: Prof. dr. sc. Damir Godec

SADRŽAJ

POPIS SLIKA

POPIS OZNAKA

- **YOLO** – You Only Look Once
- **IoU** – Intersection over Union
- **CNN** – Convolutional Neural Network
- **R-CNN** – Region-Based Convolutional Neural Networks
- **RPN** – Region Proposal Networks
- **RoI** – Region of Interest
- **RoIPool** – Region of Interest Pooling
- **SSD** – Single Shot Multibox Detector
- **NMS** – Non-maximum Suppression
- **XML** – Extensible Markup Language

SAŽETAK

U radu je detaljno objašnjen koncept prepoznavanja ljudi i objekata pomoću Haar kaskadnog klasifikatora. U ovom radu je glavni fokus na prepoznavanju vozila i pješaka u prometu. Cilj je stvoriti što bolji klasifikator korištenjem prikupljenih podataka te na taj način omogućiti precizno prepoznavanje vozila i pješaka. Kako bi se dobio što bolji klasifikator, potrebno je prikupiti veliki broj korisnih podataka koji će kasnije proći kroz određeni trening te će oni tvoriti kaskadu pomoću kojih će se pokretati programi za detekciju željenih objekata. Metoda je isprobana na više različitih videozapisa i slika te s više različitih kaskada kako bi se rezultati lakše usporedili.

Na kraju, uzimajući u obzir i ostale klasifikatore osim Haar kaskadnog klasifikatora, donesen je zaključak.

Ključne riječi: Haar kaskadni klasifikator, vozila, pješaci, detekcija

SUMMARY

The paper explains in detail the concept of recognizing people and objects using the Haar cascade classifier. In this paper, the main focus is recognition of vehicles and pedestrians in traffic. The goal is to create the best possible classifier using the collected data and enable precise recognition of vehicles and pedestrians. In order to obtain the best possible classifier, it is necessary to collect a large number of useful data that will later go through specific training and they will form a cascade with which the programs for detecting the desired objects will be launched. The method was tested on several different videos and images and with several different cascades to make the results easier to compare.

Finally, taking into account other classifiers besides the Haar cascade classifier, a conclusion was made.

Key words: Haar cascade classifier, vehicles, pedestrians, detection

1. UVOD

Detekcija vozila i pješaka u prometu danas ima veliki značaj i svakim danom je sve više zastupljena. Na sigurnosnim kamerama se sve više primjenjuje kako bi nivo sigurnosti i praćenja prometa porastao. Također, sve je više zastupljena i u samim vozilima te se na taj način teži unaprjeđenju autonomne vožnje. Budući da je u pitanju ljudska sigurnost, detekcija mora biti bez greške te za detekciju vozila i pješaka postoji nekoliko metoda koje se primjenjuju.

Neke od metoda koje se mogu koristiti za detekciju vozila i pješaka u prometu su:

1. YOLO (You Only Look Once)
2. Mask R-CNN
3. SSD (Single Shot MultiBox Detector)
4. Haar kaskadni klasifikator

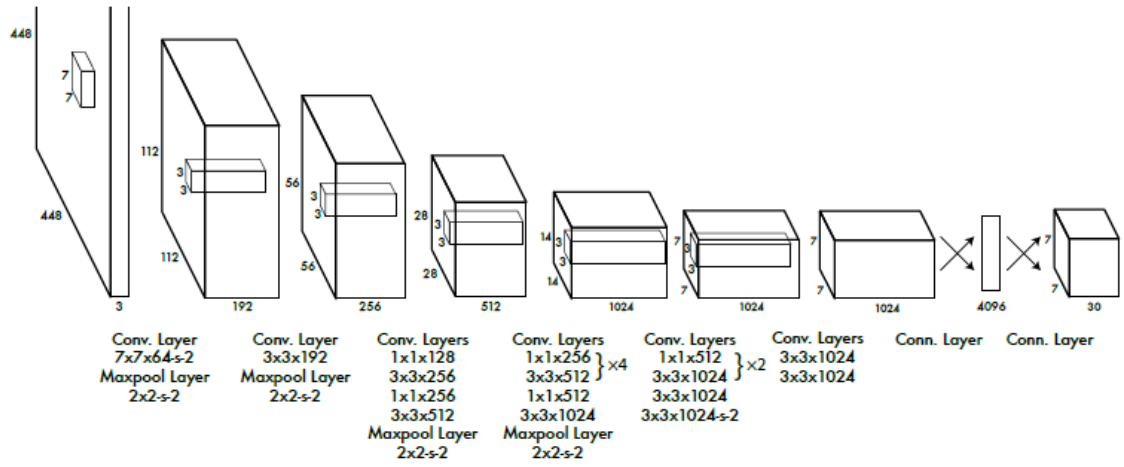
O prve tri metode u radu su navedene samo najosnovnije karakteristike dok se rad više fokusira na Haar kaskadni klasifikator te njegovu primjenu u zadanim situacijama.

2. KLASIFIKATORI ZA DETEKCIJU

2.1. YOLO (You Only Look Once)

YOLO koristi neuronsku mrežu koja predviđa vjerojatnost graničnih okvira i klasa odjednom. Razlikuje se od ostalih detekcijskih algoritama koji su svoje klasifikatore prenamijenili za obavljanje detekcije. YOLO sva predviđanja obavlja pomoću jednog potpuno povezanog sloja. Metode koje koriste mreže za prijedlog regija obavljaju više iteracija na istoj slici dok YOLO isti taj posao napravi u jednoj iteraciji.

YOLO algoritam uzima sliku kao ulaz i koristi duboku konvolucijsku neuronsku mrežu za detekciju objekata na slici. Arhitektura modela konvolucijske neuronske mreže koja tvori okosnicu YOLO algoritma je prikazana na slici (Slika 1.).



Slika 1. Arhitektura YOLO metode

Prvih 20 konvolucijskih slojeva modela su prethodno obučeni pomoću ImageNet-a uključivanjem privremenog prosječnog skupnog sloja i potpuno povezanog sloja. Zatim je taj prethodno obučen model pretvoren da obavlja detekciju jer su prethodna istraživanja pokazala da dodavanje konvolucijskog i povezanog sloja prethodno obučenoj mreži poboljšava izvedbu. Posljednji potpuno povezani sloj predviđa obje klasne mogućnosti i koordinate graničnog okvira.

YOLO dijeli ulaznu sliku u S x S koordinatnu mrežu. Ako centar objekta upada u ćeliju koordinatne mreže, ta ista ćelija je odgovorna za detekciju tog objekta. Svaka ćelija predviđa B graničnih okvira i iznos pouzdanosti za te okvire. Iznosi pouzdanosti govore koliko je model pouzdan da okvir sadrži objekt i koliko je zapravo predviđeni okvir precizan.

YOLO predviđa više graničnih okvira po ćeliji koordinatne mreže. Za vrijeme treninga, potreban je samo jedan granični okvir koji je zadužen za detekciju svih objekata. YOLO dodjeljuje jednom

prediktoru da bude „odgovoran“ za predviđanje objekata na temelju čijeg predviđanje ima najveći trenutni IoU s osnovnom istinom. Ovo vodi do specijalizacije između prediktora graničnog okvira. Svaki prediktor postaje bolji u predviđanju određenih veličina, omjera slike ili klasa objekata, poboljšavajući ukupni rezultat prisjećanja.

Ključna tehnika koja se koristi u YOLO modelu je ne-maksimalno potiskivanje (eng. non-maximum suppression, NMS). NMS je korak naknadne obrade koji se koristi za poboljšanje točnosti i učinkovitosti detektiranja objekata. Kod detekcije objekata, normalno je da se generira nekoliko graničnih okvira za jedan objekt na slici. Ti granični okviri mogu se preklapati ili mogu biti locirani na različitim pozicijama, ali svi oni predstavljaju isti objekt. NMS se koristi za identifikaciju i uklanjanje suvišnih ili netočnih graničnih okvira i za ispis jednog graničnog okvira za svaki objekt na slici. [1]

2.2. Mask R-CNN

Mask R-CNN je konvolucijska neuronska mreža (eng. Convolutional Neural Network, CNN) i najsuvremenija je metoda u segmentaciji slike. Ova varijanta duboke neuronske mreže detektira objekte na slici i generira segmentacijsku masku visoke kvalitete za svaki slučaj.

Kako bi se Mask R-CNN metoda bolje razumjela, potrebno je objasniti osnovne koncepte:

- Konvolucijska neuronska mreža (CNN)
- Regionalne konvolucijske neuronske mreže (R-CNN)
- Brži R-CNN s mrežom prijedloga regija (RPN)
- Način rada Mask R-CNN-a

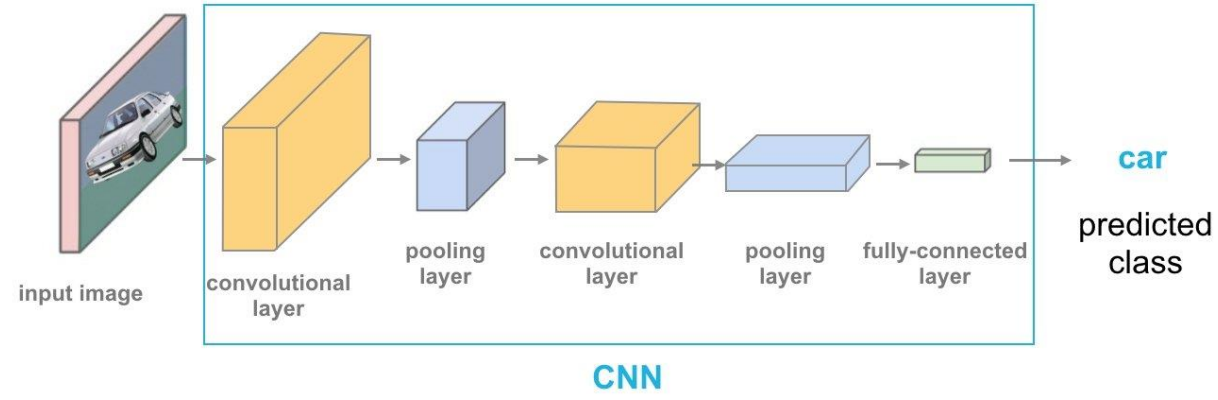
2.1.1. Konvolucijska neuronska mreža (CNN)

Konvolucijska neuronska mreža je tip umjetne neuronske mreže koja se koristi u procesima prepoznavanja slike i obradi koja je optimizirana za obradu podataka o pikselima. Stoga, konvolucijske neuronske mreže su temeljne i osnovne komponente za zadatke računalnog vida u segmentaciji slike.

Arhitektura konvolucijske neuronske mreže sastoji se od tri osnovna sloja:

- **Konvolucijski sloj:** Ovaj sloj pomaže u sažimanju ulazne slike kao karte značajki korištenjem filtera i jezgri.
- **Sloj udruživanja:** Pomoću ovog sloja se smanjuje uzorkovanje karti značajki sažimanjem prisutnosti značajki u dijelovima karte značajki.
- **Potpuno povezani sloj:** Potpuno povezani slojevi povezuju svaki neuron u jednom sloju sa svakim neuronom u drugom sloju.

Kombiniranjem CNN slojeva, neuronskoj mreži se omogućuje da nauči kako identificirati i prepoznati traženi objekt na slici. Jednostavne konvolucijske neuronske mreže su napravljene za klasifikaciju slika i detekciju objekata s jednim objektom na slici.

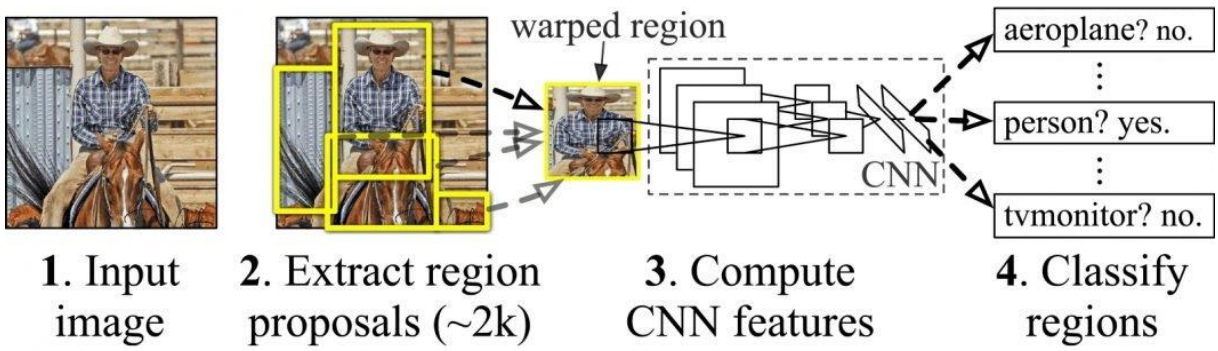


Slika 2. Arhitektura konvolucijske neuronske mreže (CNN)

U kompleksnijim slučajevima s više objekata na slici, jednostavna CNN arhitektura nije optimalna pa se onda koristi Mask R-CNN koji se temelji na R-CNN-u.

2.1.2. Regionalne konvolucijske neuronske mreže (R-CNN)

Regionalne konvolucijske neuronske mreže su tip modela strojnog učenja koji se koristi za zadatke računalnog vida u detekciji objekata. Ovaj pristup koristi granične okvire preko područja objekata koji zatim neovisno procjenjuju konvolucijske mreže neovisno o svim regijama interesa (RoI) kako bi klasificirali više područja slike u predloženu klasu.

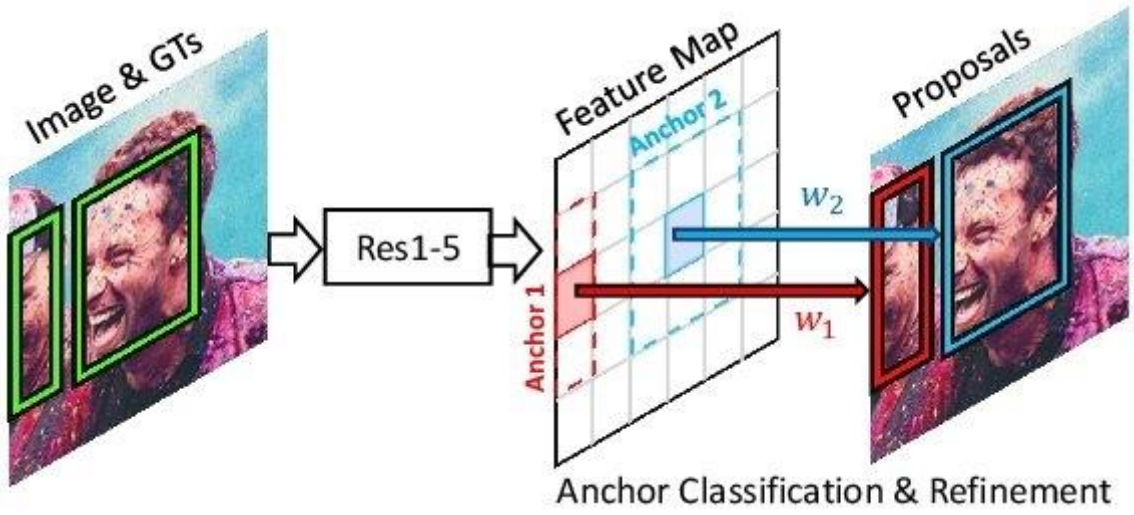


Slika 3. Prikaz R-CNN koncepta

2.1.3. Brži R-CNN s mrežom prijedloga regija (RPN)

Brzi R-CNN je poboljšana verzija R-CNN arhitekture s dvije faze:

- **Mreža prijedloga regija (RPN):** Neuronska mreža koja predlaže višestruke objekte koji su dostupni unutar određene slike



Slika 4. Prikaz RPN koncepta

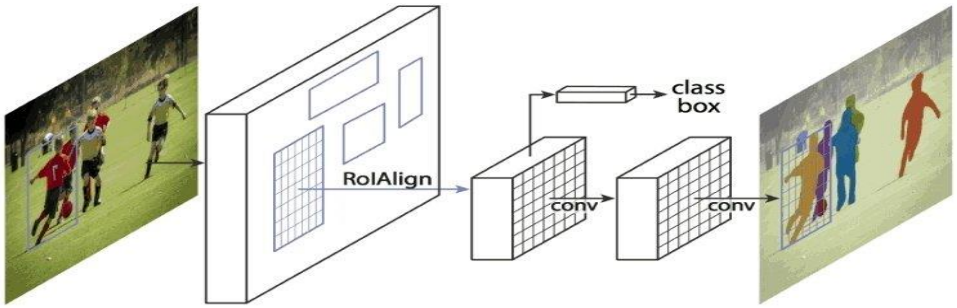
- **Brzi R-CNN:** Na ovaj način se izvlače značajke korištenjem RoIPool (Region of Interest Pooling) iz svakog kandidatskog okvira i izvodi klasifikaciju i regresiju graničnog okvira. RoIPool je operacija za izvlačenje male karte značajki iz svake regije interesa u detekciji.

Slika 5. Prikaz koncepta brzog R-CNN-a

2.1.4. Način rada Mask R-CNNa

Mask R-CNN je napravljen koristeći brži R-CNN. Dok brži R-CNN ima 2 izlaza za svakog objekta kandidata (oznaka klase i pomak graničnog okvira), Mask R-CNN je dodatni treći izlaz koji je različit od prva dva izlaza te zahtjeva izdavanje mnogo finijeg prostornog izgleda objekta.

Mask R-CNN je ekstenzija bržeg R-CNN-a i radi dodavanjem grane za predviđanje maske objekta (RoI) paralelno s postojećom granom za prepoznavanje graničnog okvira. [3]



Slika 6. Okvir Mask R-CNN-a za segmentaciju slike

2.3. SSD (Single Shot MultiBox Detector)

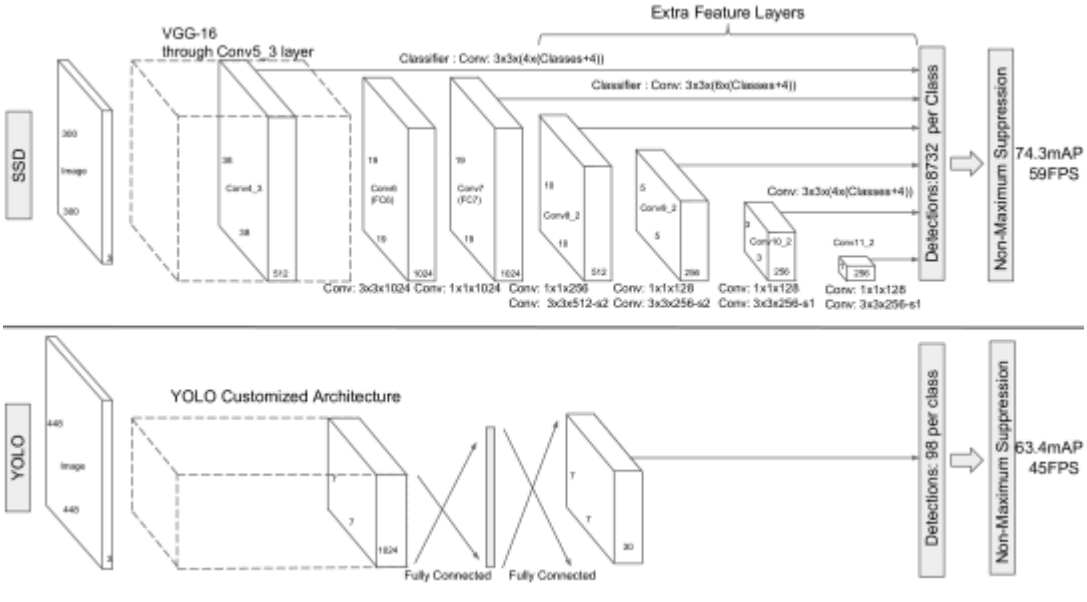
SSD pristup se temeljni na konvolucijskim mrežama koje proizvode kolekciju graničnih okvira fiksne veličine i bodove za prisutne objektne klase u tim okvirima.

Rani sloj mreže se temelji na standardnoj arhitekturi temeljenoj na visokoj kvaliteti klasifikacije slika (okrnjen prije ostalih klasifikacijskih slojeva) te se taj sloj mreže naziva osnovna mreža.

Zatim se dodaje pomoćna struktura na mrežu kako bi se potakla detekcija sa sljedećim ključnim značajkama:

- Karte značajki za detekciju u više razmjera – dodani su konvolucijski slojevi značajki na kraj okrnjene osnovne mreže. Ti slojevi se postupno smanjuju i dopuštaju predviđanje detekcija u više mjerila. Konvolucijski model za predviđanje detekcija je različit za svaki sloj značajki (dok YOLO predviđanja vrši u jednom mjerilu karte značajki).

- Konvolucijski prediktori za detekciju – svaki dodani sloj značajki (ili postojeći sloj značajki iz osnovne mreže) može proizvesti fiksni skup detekcijskih predviđanja pomoću skupa konvolucijskih filtera. Oni su naznačeni na vrhu arhitekture SSD mreže na slici (Slika 7.). Za sloj značajki veličine m x n koji ima p kanala, osnovni element za predviđanje parametara potencijalne detekcije je 3 x 3 x p mala jezgra koja proizvodi ili rezultat za kategoriju ili oblik pomaka relativnog na koordinatni sustav osnovnog okvira. Na svakom m x n mjestu gdje se jezgra primjenjuje, ispisuje se izlazna vrijednost. Izlazne vrijednosti pomaka graničnog okvira mjere se u odnosu na zadani položaj okvira u odnosu na svaki lokaciju karte značajki (dok YOLO za ovaj korak koristi posredni potpuno povezani sloj umjesto konvolucijskih filtera).
- Zadani okviri i omjeri slike – pridružuje se skup zadanih graničnih okvira svakoj ćeliji karti značajki za više karata značajki na vrhu mreže. Osnovni okvir slaže kartu značajki na konvolucijski način tako da je pozicija svakog okvira u odnosu na odgovarajuću ćeliju fiksna. U svakoj ćeliji karte značajki se predviđaju pomaci u odnosu na osnovni oblik okvira u ćeliji kao i rezultati po klasama koji ukazuju na prisutnost klase u svakom od tih okvira. [5]

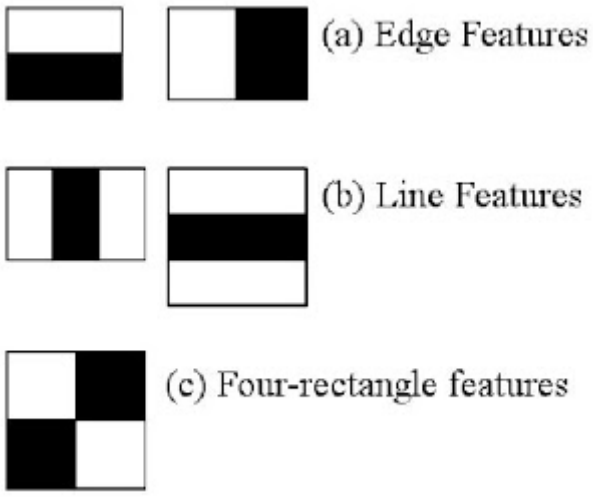


Slika 7. Usporedba SSD modela (gore) i YOLO modela (dolje)

2.4. Haar kaskadni klasifikator

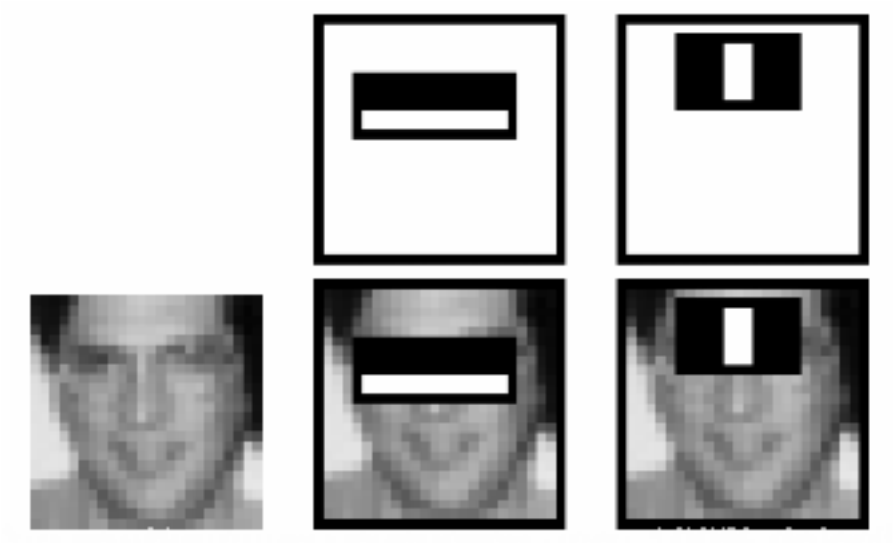
Ideju Haar kaskadnog klasifikatora razvili su Paul Viola i Michael Jones. Klasifikator se bazira na strojnom učenju gdje se kaskadna funkcija trenira pomoću pozitivnih i negativnih slika. Nakon toga se koristi za detekciju objekata na drugim slikam.

Haar kaskadni klasifikator se najčešće primjenjuje pri detekciji ljudskog lica. U slučaju prepoznavanja lica, pozitivne slike bi bile slike lica, a negativne slike su sve one koje ne sadržavaju ljudsko lice. Kada se prikupi dovoljan broj pozitivnih i negativnih slika, slijedi trening te stvaranje kaskade. Iz slika se izdvajaju značajke gdje svaka značajka ima jedinstvenu vrijednost koja se dobila oduzimanjem zbroja piksela ispod bijelog pravokutnika od zbroja piksela ispod crnog pravokutnika.



Slika 8. Tipovi Haar značajki

Prozor veličine 24x24 ima preko 160000 značajki što je prevelik broj za računanje svake značajke posebno, ali većina značajki koje se računaju su zapravo nebitne što uvelike olakšava cijeli postupak. Za primjer se može uzeti slika ispod.



Slika 9. Proces prepoznavanja slike kod Haar metode

Gornji red prikazuje dvije dobre značajke. Prva odabrana značajka se fokusira na svojstvo da je područje očiju češće tamnije od područja gdje su nos i obrazi koji su svijetliji. Druga odabrana značajka se odnosi na svojstvo da su oči tamnije od mosta na nosu. Svaki prozor primijenjen na obraze ili bilo koje drugo područje je nebitno. Kako bi se od preko 160000 značajki odabrale one najbolje, koristi se AdaBoost. Sve značajke primjenjujemo na svim slikama za trening. Za svaku značajku se pronalazi najbolji prag koji će klasificirati lica u pozitivno i negativno. Pojavljivat će se greške pa se uzimaju značajke s minimalnom učestalosti pogreške što znači da te značajke najpreciznije klasificiraju slike s licem ili bez njega. Cijeli proces nije jednostavan kao što se čini. U početku svaka slika ima jednaku težinu. Nakon svake klasifikacije, težina greške kod klasifikacije slike se povećava. Tada proces završava, a nova učestalost greške se računa te s tim i nova težina. Proces se nastavlja dok se ne ostvari zahtjevana preciznost ili učestalost greške ili dok se ne pronađe traženi broj značajki. Konačni klasifikator je ponderirani zbroj svih tih slabih klasifikatora. Nazivaju se slabim jer sami ne mogu klasificirati sliku, ali zajedno tvore snažan klasifikator. 200 značajki osigurava detekciju s 95% točnosti. Njihova konačna postavka ima oko 6000 značajki, a samo smanjenje od 160000+ značajki na njih 6000 je veliki dobitak. Ako bi se svih 6000 značajki primijenilo na slici, to bi uzelo previše vremena i postalo bi neučinkovito. Pošto većina slike ne sadrži područje lica, koristi se jednostavna metoda kojom će se vidjeti je li prozor slike unutar ili izvan regije lica. Ako se prozor ne nalazi u regiji lica, odbacuje se i više se ne procesuira. Umjesto toga, fokus se stavlja na regije gdje može biti lice. Na ovaj način se više vremena troši na provjeru regija lica, ali na kraju ipak ubrzava cijeli proces. Za ovo se koristi koncept Kaskada klasifikatora (Cascade of Classifiers). Umjesto primjene svih 6000 značajki na prozor, značajke se grupiraju u različite faze klasifikatora i primjenjuju se jedna po jedna. Obično

prvih par faza će sadržavati mnogo manje značajki. Ako prozor ne zadovolji prvu fazu, odbacuje se te se njegove značajke ne uzimaju u obzir. Ako prozor zadovolji prvu fazu, na njega se primjenjuje druga faza značajki i proces se nastavlja. Prozor koji zadovolji sve faze je regija lica. [6]

Iako se Haar kaskadni klasifikator često koristi za prepoznavanje lica, u sklopu ovog rada će se primijeniti u prepoznavanju vozila i pješaka u prometu te će se na kraju dobiti određeni zaključci.

3. IZRADA HAAR KASKADE

3.1. Prikupljanje podataka

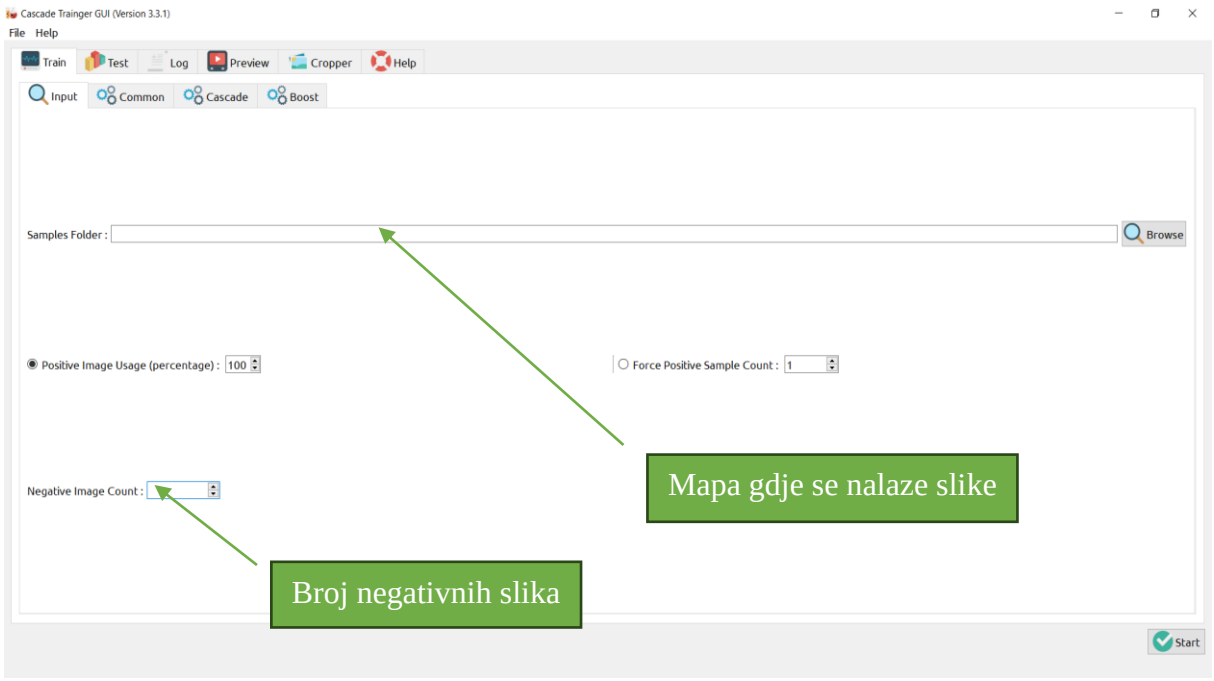
Najvažnija stvar za stvaranje snažne kaskade su kvalitetni podaci. Podaci se sastoje od pozitivnih i negativnih slika. U pozitivne slike se ubrajaju sve slike na kojima se nalaze objekti koje je potrebno detektirati, a to su u ovom slučaju automobili, motocikli, autobusi i pješaci (Slika 11.). Za svaki od navedenih objekata koje je potrebno detektirati radi se poseban trening, odnosno posebna kaskada što znači da su napravljena 4 različita treninga za 4 različite kaskade. Pod negativne slike spadaju sve slike koje ne sadrže nikakve željene objekte detekcije. To može biti bilo što, ali najbolje je koristiti objekte koji okružuju željene objekte detekcije. To mogu biti slike ceste bez prometnih vozila, prometni znakovi, semafori, pješački prijelazi, stabla, ulične rasvjete, itd. (Slika 10.)

Slika 10. Primjer negativne slike

Slika 11. Primjer pozitivne slike

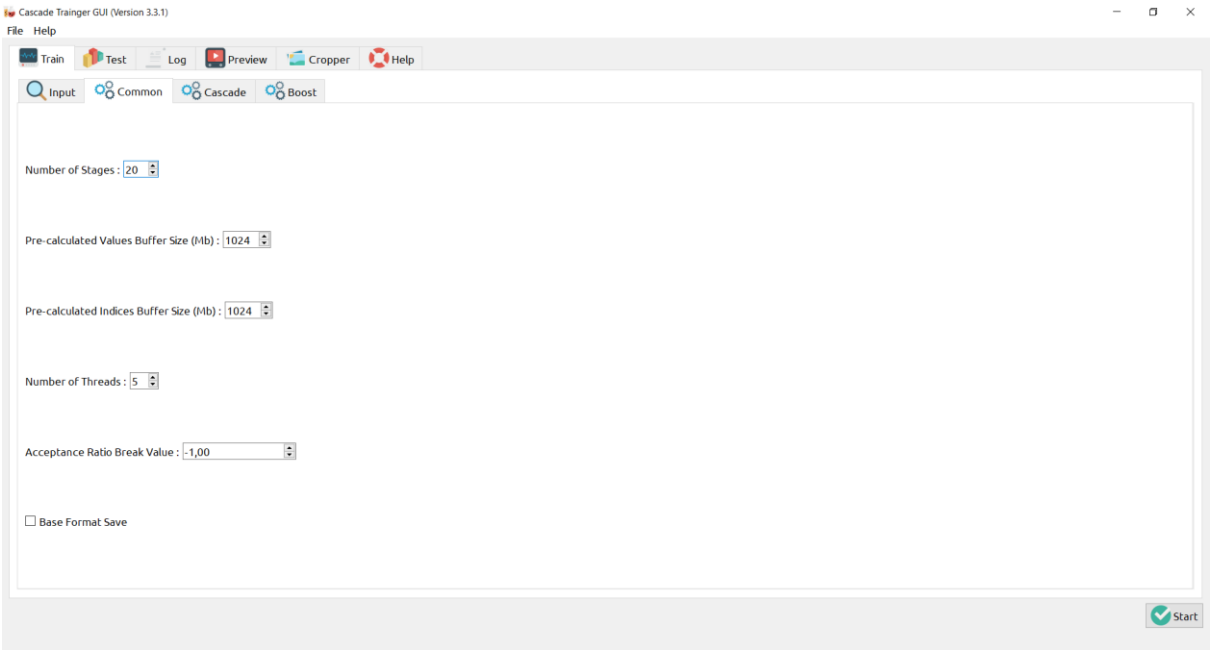
3.2. Stvaranje kaskade

Za stvaranje kaskade koristi se program „*Cascade-Trainer-GUI*“. Potrebno je namjestiti određene postavke prije samog početka treninga (Slika 12.). Prvo se mora odabrati mapa gdje su spremljene pozitivne i negativne slike. Jako važna stvar je da se pozitivne slike spremaju u mapu pod nazivom „*p*“, a negativne pod nazivom „*n*“. Za početak su to jedine dvije mape unutar glavne mape. Nakon toga se unosi broj negativnih slika koje će proći kroz proces treninga. Veći broj negativnih slika znači i veće opterećenje memorije računala, a s tim i trajanje samog procesa treninga.



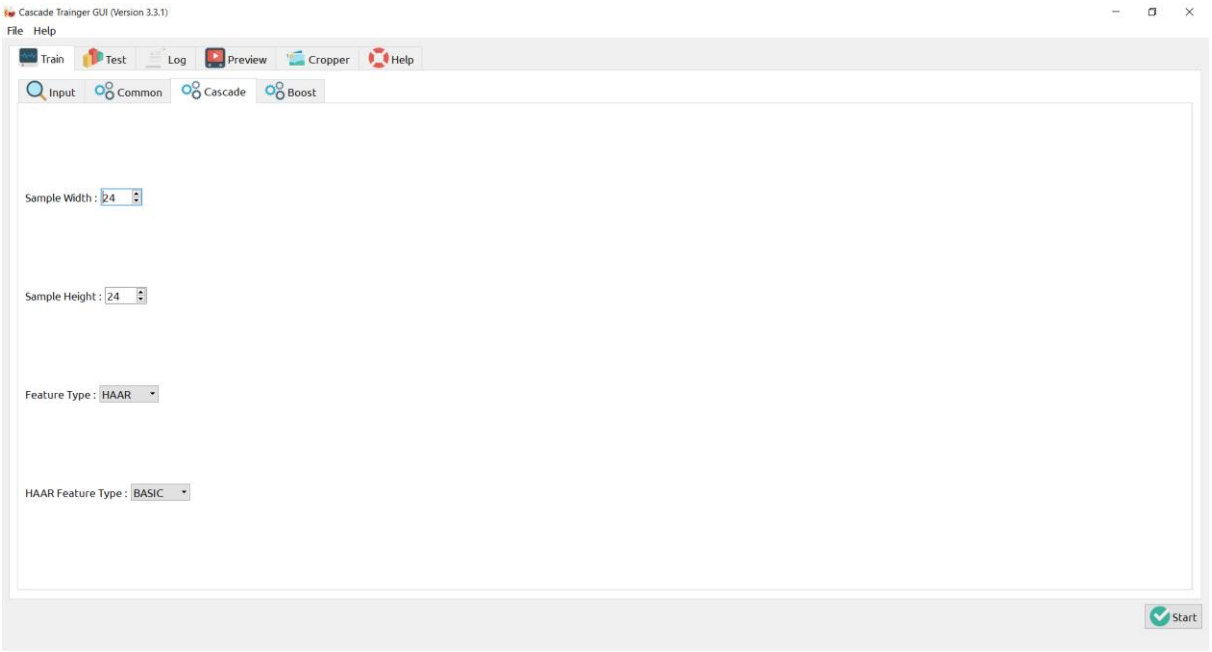
Slika 12. Cascade-Trainer-GUI Input

U prozoru „*Common*“ već su postavljene zadane vrijednosti, ali moguće ih je podešavati po vlastitoj želji (Slika 13.). Kako bi se dobila snažnija kaskada, onda je potrebno povećati broj faza (Number of Stages), ali treba imati na umu da s povećanjem broja faza raste vrijeme trajanja treninga kao i opterećenje memorije. Na slici su prikazane zadane vrijednosti kada se otvori prozor.



Slika 13. Cascade-Trainer-GUI Common

U prozoru *„Cascade"* se mogu podešavati dimenzije uzorka slike (Slika 14.). Također, određuje se i svojstvo rada što je u ovom slučaju Haar.



Slika 14. Cascade-Trainer-GUI Cascade

Trening može trajati od nekoliko sekundi pa i do nekoliko desetaka minuta, ovisno o unesenim parametrima, broju podataka i snazi računala. Treba imati na umu da se računalno ne preoptereti jer onda trening neće biti uspješan. Također, treba obratiti pažnju da se korištenjem većeg broja podataka i zahtjevnijih parametara poboljšava kaskada, odnosno dobivaju se bolji rezultati detekcije. Potrebno je pronaći idealan balans između efikasnosti i uloženog vremena.

Nakon što trening završi, stvara se nova mapa u početnoj mapi gdje se nalaze pozitivne i negativne slike. Nova mapa se zove *„classifier“* u kojoj se nalazi XML datoteka kaskade.

4. PROGRAM ZA DETEKCIJU

4.1. OpenCV kod u programskom jeziku Python

Kod za detekciju vozila i pješaka napisan je u programskom jeziku „*Python 3.11.4.“*. Za svaki videozapis je napravljen zaseban kod.

4.1.1. Detekcija automobila

Za početak je napisan jednostavan kod za detekciju automobila u prometu (Slika 15.) te je cilj vidjeti koliko učinkovito kaskada prepoznaje automobile.

```
import cv2

cascade_src = 'auta.xml'

video_src = 'video.avi'

cap = cv2.VideoCapture(video_src)

car_cascade = cv2.CascadeClassifier(cascade_src)


while True:
    ret, img = cap.read()

    if (type(img) == type(None)):
        break
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    cars = car_cascade.detectMultiScale(gray, 1.1, 2)


    for (x,y,w,h) in cars:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,255),2)

    cv2.imshow('video', img)


    if cv2.waitKey(33) == 27:
        break

cv2.destroyAllWindows()
```

Slika 15. Jednostavan Python kod za prepoznavanje automobila u prometu

Prije pisanja samog koda, potrebno je instalirati cv2 biblioteku koja omogućava korištenje naredbi za obradu slika i videa. To je najjednostavnije napraviti koristeći *„Command Prompt“* pomoću naredbe „*pip install opencv-python“*.

XML datoteka kaskade koja se nalazi u mapi *„classifier“* kopirana je u mapu gdje se nalazi python program za detekciju automobila te se ona u ovom slučaju zove *„auta.xml“*.

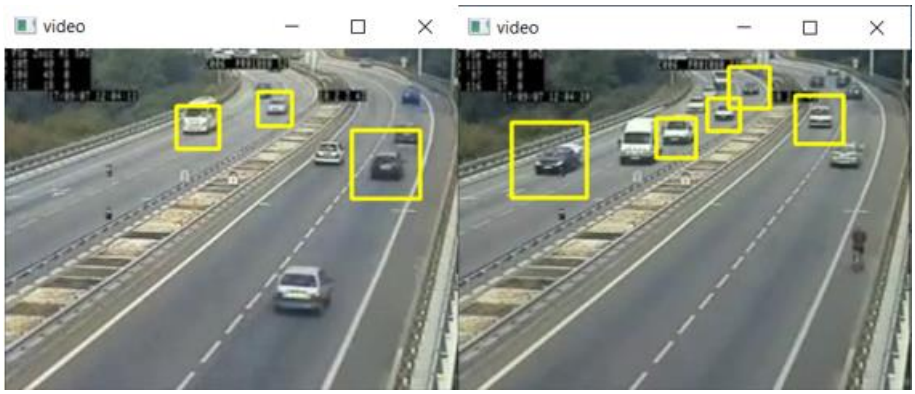
Osim kaskade potreban je i videozapis u kojem želimo detektirati automobile u prometu. U ovom slučaju je to *„video.avi“*.

„VideoCapture“ omogućuje čitanje slika unutar videozapisa te ih sprema u varijablu „*cap“*.

„CascadeClassifier“ stvara objekt „*car_cascade“* korištenjem kaskade koja je dobivena treningom *„auta.xml“*. Taj stvoreni objekt predstavlja Haar kaskadni klasifikator za detekciju automobila.

Sada se može pokrenuti while petlja koja traje sve dok ne završi videozapis ili dok se ne pritisne tipka Esc. Svaka slika iz videozapisa se čita te se ti podaci spremaju u varijablu „*img“* dok varijabla „*ret“* pokazuje da li je slika uspješno pročitana. Kada više nema slika za čitanje onda varijabla „*img“* poprima vrijednost „*None“* te petlja završava. Naredba „*cvtColor“* pretvara sliku unutar varijable „*img“* u sive nijanse kako bi se olakšao cjelokupni proces. Slijedi detekcija automobila pomoću naredbe „*detectMultiScale“* te se dobiva lista pravokutnika koji predstavljaju detektirane regije. Pomoću parametara 1.1 i 2 se namještaju željeni uvjeti za detekciju automobila. Oko detektiranih automobila se crtaju pravokutnici žute boje (0,255,255) te je debljina pravokutnika stavljena na 2. Da bi se vidio dobiveni rezultat potrebno je koristiti naredbu „*imshow“*.

Nakon što petlja završi, naredba „*destroyAllWindows“* zatvara sve OpenCV prozore koji su stvoreni za vrijeme egzekucije programa.



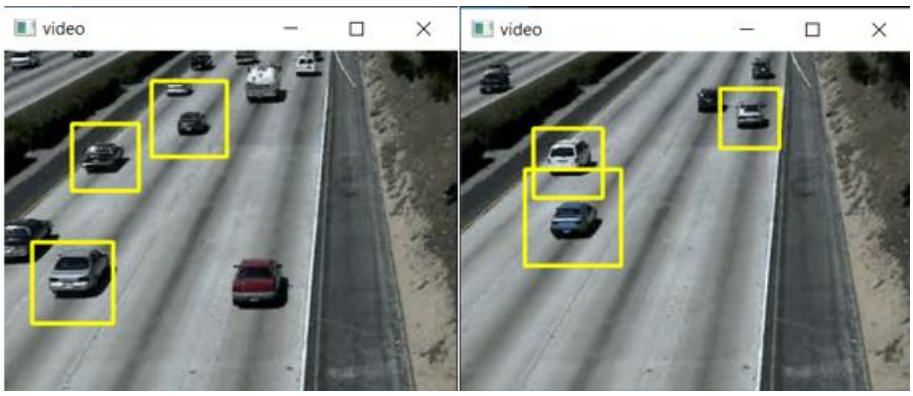
Slika 16. Rezultati detekcije automobila na prvom videozapisu

Kako bi se testirala valjanost kaskade, isti postupak je napravljen za novi videozapis s automobilima u prometu. Python kod je identičan, jedino što se u kodu mora promijeniti je naziv novog videozapisa koji se koristi (Slika 17.).

```
import cv2

cascade_src = 'auta.xml'

video_src = 'video1.avi'

cap = cv2.VideoCapture(video_src)

car_cascade = cv2.CascadeClassifier(cascade_src)


while True:
    ret, img = cap.read()

    if (type(img) == type(None)):
        break

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    cars = car_cascade.detectMultiScale(gray, 1.1, 2)


    for (x,y,w,h) in cars:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,255),2)

    cv2.imshow('video', img)

    if cv2.waitKey(33) == 27:
        break

cv2.destroyAllWindows()
```

Slika 17. Identičan Python kod prethodnom kodu s novim videozapisom

Prepoznavanje automobila na novom videozapisu radi na identičan način te se može zaključiti da kaskada i Python mogu prepoznavati automobile na više različitih videozapisa.



Slika 18. Rezultati detekcije automobila na drugom videozapisu

Prepoznavanje automobila radi relativno dobro, ali ako se želi poboljšati efikasnost same kaskade potrebno je napraviti još bolji trening. Za to je potreban jako veliki broj podataka kao i što veća radna memorija računala. Problem Haar kaskade je teško ostvarivanje idealne detekcije automobila. Prepoznati automobili na videozapisu ne ostaju detektirani cijelo vrijeme dok traje videozapis nego samo jedan dio vremena. Taj problem je detaljnije objašnjen u nastavku ovog rada.

4.1.2. Detekcija vozila na dva kotača

Kao i kod automobila, detekcija vozila na dva kotača (motocikli, mopedi, itd.) radi na istom principu. Glavna razlika je što se za trening kaskade ne koriste slike automobila nego slike vozila na dva kotača. Princip treninga je isti, prikupljanje podataka, izrada kaskade te pisanje Python koda. Python kod je isti kao i za detekciju automobila.

Ovdje je kaskada također isprobana na 2 različita videozapisa kako bi se mogli dobiti što precizniji rezultati. Za oba videozapisa se koristi isti Python kod (Slika 19. i Slika 20.). Ovdje je još korištena naredba za promjenu veličine prozora videozapisa jer su početne dimenzije prevelike pa se ne mogu vidjeti rezultati detekcije.

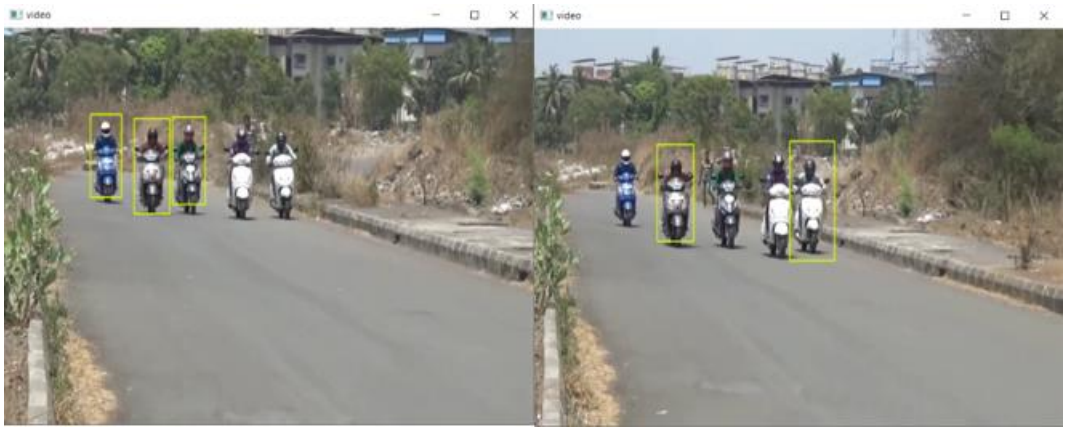
```
import cv2

cascade_src = 'two_wheeler.xml'

video_src = 'video.mp4'

cap = cv2.VideoCapture(video_src)

bike_cascade = cv2.CascadeClassifier(cascade_src)

new_window_size = (640, 480)

while True:
    ret, img = cap.read()

    if (type(img) == type(None)):
        break

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    bikes = bike_cascade.detectMultiScale(gray,1.1, 1)


    for (x,y,w,h) in bikes:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,215),2)

    resized_img = cv2.resize(img, new_window_size)

    cv2.imshow('video', resized_img)

    if cv2.waitKey(33) == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

Slika 19. Python kod za detekciju vozila na dva kotača na prvom videozapisu

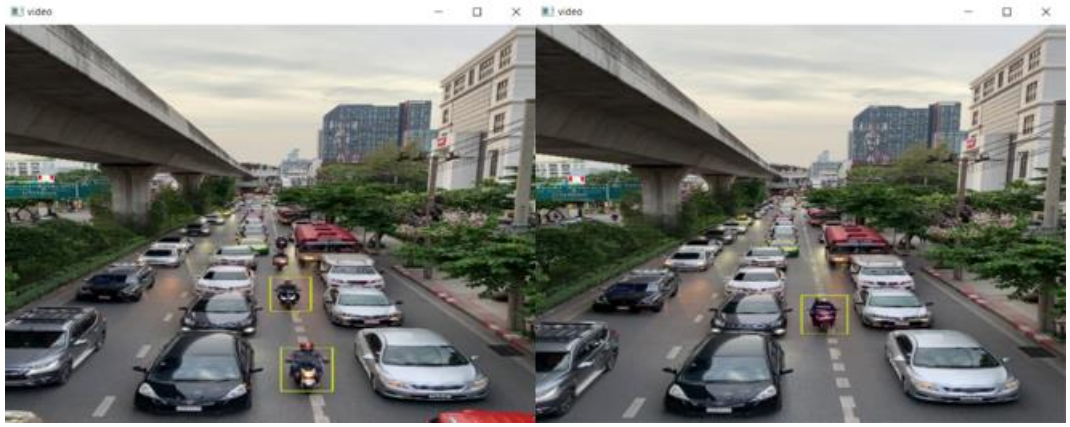
```
import cv2

cascade_src = 'two_wheeler.xml'

video_src = 'video1.mp4'

cap = cv2.VideoCapture(video_src)

bike_cascade = cv2.CascadeClassifier(cascade_src)

new_window_size = (640, 480)

while True:
    ret, img = cap.read()

    if (type(img) == type(None)):
        break

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    bikes = bike_cascade.detectMultiScale(gray,1.2, 1)


    for (x,y,w,h) in bikes:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,215),2)

    resized_img = cv2.resize(img, new_window_size)

    cv2.imshow('video', resized_img)

    if cv2.waitKey(33) == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

Slika 20. Python kod za detekciju vozila na dva kotača na drugom videozapisu

Kao što se može vidjeti, oba koda su identična, jedina razlika je videozapis koji se testira. Sada slijedi pokretanje koda kako bi se vidjela učinkovitost kaskade za prepoznavanje vozila na dva kotača.



Slika 21. Rezultati detekcije vozila na dva kotača na prvom videozapisu



Slika 22. Rezultati detekcije vozila na dva kotača na drugom videozapisu

Nakon pregledanih rezultata, može se uočiti da kaskada za prepoznavanje vozila na dva kotača radi jako slično kao i kaskada za prepoznavanje automobila. U oba videozapisa su prepoznata sva motorna vozila na dva kotača, ali opet je problem što vozila nisu cijelo vrijeme označena žutim pravokutnicima te će se za vrijeme trajanja videozapisa isto vozilo detektirati više puta.

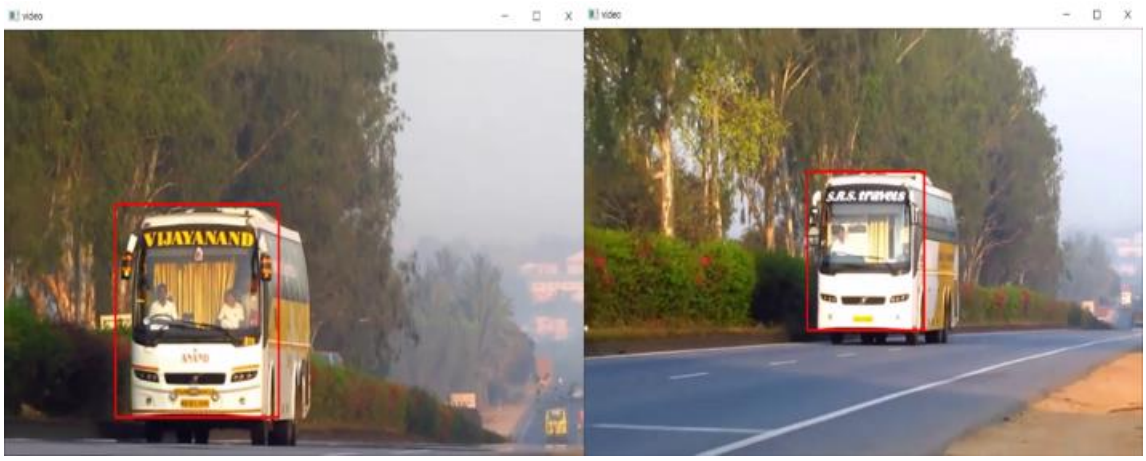
4.1.3. Detekcija autobusa

Kaskada za detekciju autobusa radi na principu prepoznavanja prednje strane vozila. Budući da su autobusi povišeni i imaju specifičan izgled prednje strane, to ne predstavlja prevelik problem. Detekcija autobusa može biti vrlo korisna na autobusnim kolodvorima i stanicama kako bi se mogao pratiti vozni red kao i sigurnost sudionika u prometu.

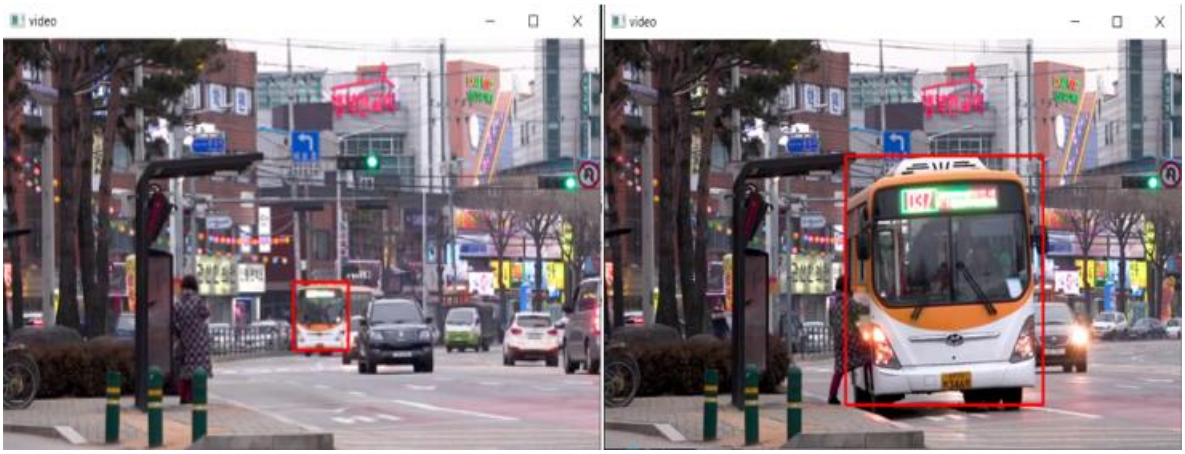
Pošto se u odabranim videozapisima ne nalazi mnogo autobusa kao što je to bio slučaj kod automobila i motocikala, ovdje su korištena 3 videozapisa kako bi se dobili što bolji rezultati. Python kodovi za sva 3 videozapisa su identični stoga je prikazan samo jedan kod (Slika 23.).

```
import cv2

cascade_src = 'Bus_front.xml'

video_src = 'bus1.mp4'

cap = cv2.VideoCapture(video_src)

bus_cascade = cv2.CascadeClassifier(cascade_src)

while True:
    ret, img = cap.read()

    if (type(img) == type(None)):
        break

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    buses = bus_cascade.detectMultiScale(gray, 1.16, 1)

    for (x,y,w,h) in buses:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255),2)

    cv2.imshow('video', img)

    if cv2.waitKey(33) == 27:
        break

cv2.destroyAllWindows()
```

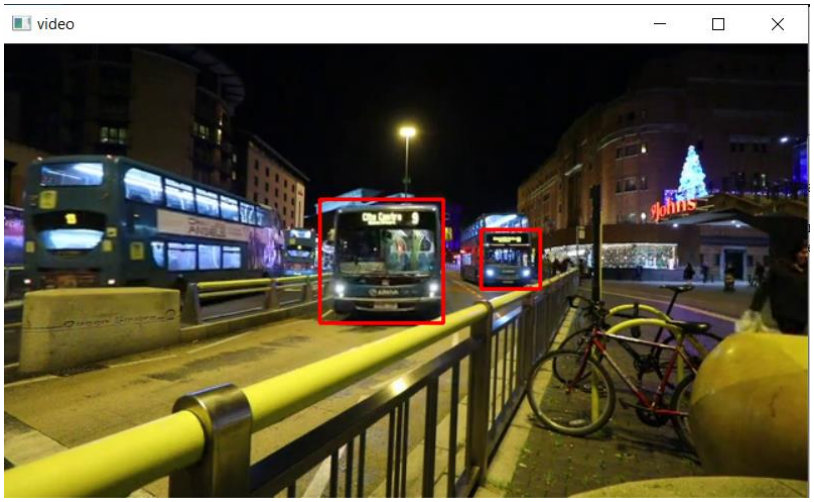
Slika 23. Python kod za detekciju autobusa u prometu



Slika 24. Rezultati detekcije autobusa na prvom videozapisu



Slika 25. Rezultati detekcije autobusa na drugom videozapisu



Slika 26. Rezultati detekcije autobusa na trećem videozapisu

Vidljivo je da kaskada dosta učinkovito prepoznaje autobuse s prednje strane te nema problema kod prepoznavanja autobusa na većim udaljenostima.

4.1.4. Detekcija pješaka

Osim detekcije vozila, jako je važna i detekcija pješaka. Ovo je posebno korisno na prometnim mjestima i pješačkim prijelazima gdje svakodnevno prolazi velik broj ljudi, ali i na centrima okupljanja ljudi kao što su trgovi, trgovinski centri, zabavna mjesta, itd.

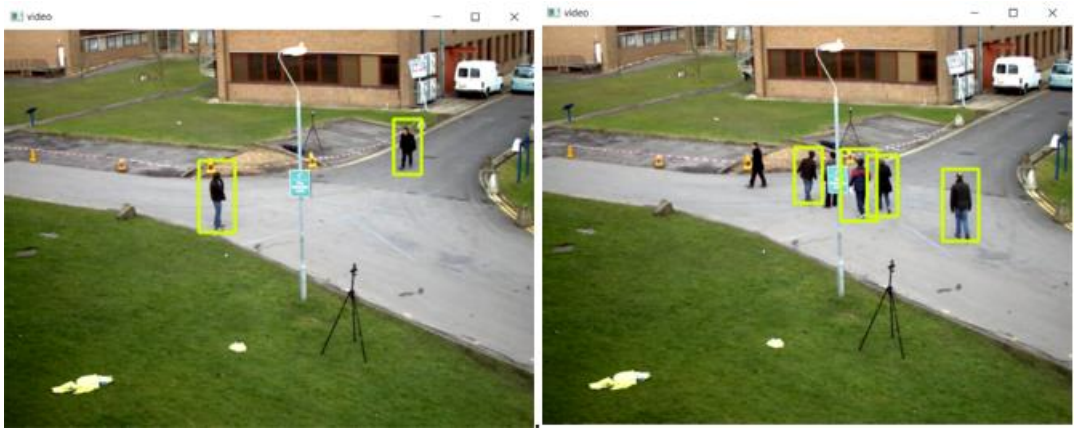
```
import cv2

video_src = 'video1.AVI'

cap = cv2.VideoCapture(video_src)

pedestrian_cascade = cv2.CascadeClassifier('pedestrian.xml')

new_window_size = (640, 480)

while True:
    ret, img = cap.read()


    if (type(img) == type(None)):
        break

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    pedestrians = pedestrian_cascade.detectMultiScale(gray,1.3,2)

    for(x,y,w,h) in pedestrians:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,210),4)

    resized_img = cv2.resize(img, new_window_size)

    cv2.imshow('video', resized_img)

    if cv2.waitKey(33) == 27:
        break

cv2.destroyAllWindows()
```

Slika 27. Python kod za detekciju pješaka na prvom videozapisu

```
import cv2

video_src = 'video2.mp4'

cap = cv2.VideoCapture(video_src)

pedestrian_cascade = cv2.CascadeClassifier('pedestrian.xml')

new_window_size = (640, 480)

while True:
    ret, img = cap.read()


    if (type(img) == type(None)):
        break

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    pedestrians = pedestrian_cascade.detectMultiScale(gray,1.3,2)

    for(x,y,w,h) in pedestrians:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,210),4)

    resized_img = cv2.resize(img, new_window_size)

    cv2.imshow('video', resized_img)

    if cv2.waitKey(33) == 27:
        break

cv2.destroyAllWindows()
```

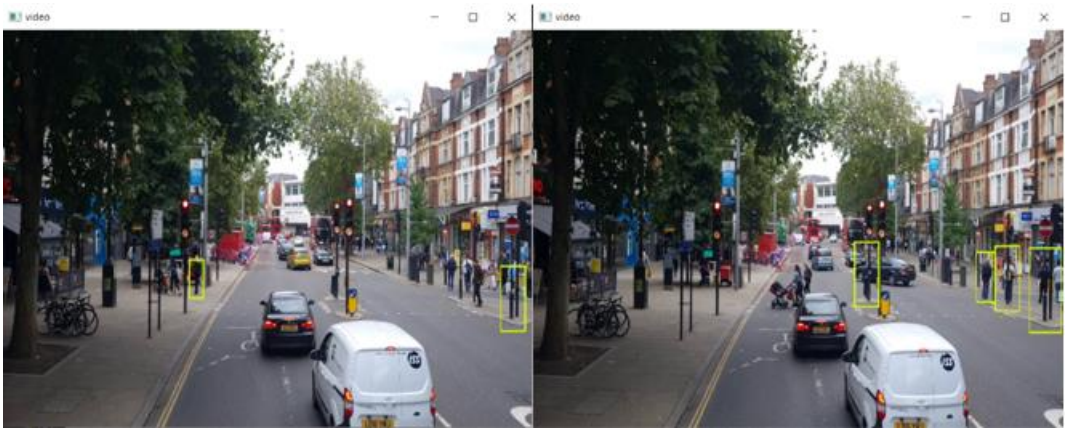
Slika 28. Python kod za detekciju pješaka na drugom videozapisu

Kaskada za detekciju pješaka isprobana je na dva različita videozapisa te su Python kodovi i u ovom slučaju identični (Slika 27. i Slika 28.). Na prvom videozapisu nema prometnih vozila koji se kreću nego samo pješaci koji hodaju po cesti. Na drugom videozapisu je prikazana cesta sa semaforima gdje vozila normalno prometuju, a po nogostupu hodaju pješaci.



Slika 29. Rezultati detekcije pješaka na prvom videozapisu

Na prvom videozapisu (Slika 29.) kaskada vrlo dobro prepoznaje pješake te nikakve druge objekte ne miješa s njima kao što to inače mogu biti stupovi i stalci koji imaju oblik ljudskih udova pa može doći do zabune u samoj detekciji.



Slika 30. Rezultati detekcije pješaka na drugom videozapisu

Na drugom videozapisu (Slika 30.) kaskada ima većih problema kod prepoznavanja pješaka. Iako je kroz videozapis većina pješaka detektirana (pješaci koji nisu zaklonjenim objektima), problem stvaraju stupovi na snimci. Na snimci stupovi i ljudi za kaskadu mogu izgledati dosta slično pa kaskada zabunom stupove detektira kao pješake. Da bi se to popravilo, potreban je trening s puno više kvalitetnih podataka što bi zahtijevalo veliku količinu vremena.

4.2. Problemi kod detekcije vozila i pješaka

U prethodno prikazanim rezultatima mogu se uočiti pojedini problemi kod detekcije vozila i pješaka. Kod svake testirane kaskade se pojavio problem **titranja**, odnosno svako vozilo ili pješak su bili detektirani više puta.

Još jedan od problema je i **detekcija pogrešnog objekta**, odnosno da se traženi objekt detekcije zamijeni nekim drugim objektom. Taj problem je viđen kod detekcije pješaka u prometu.

4.2.1. Problem titranja

Ovaj problem se spominjao kod detekcije automobila, ali pojavljuje se i kod svih drugih slučajeva. Problem je što se isti objekt detekcije prepoznaje više puta za vrijeme snimke te ispada da se detektira novi objekt. Zbog toga nije moguće pratiti broj detektiranih objekata.

Kada se željeni objekt detektira, on bude označen žutim pravokutnikom. Međutim, taj pravokutnik vrlo brzo nestane te kaskada više ne prepoznaje objekt. Kroz par milisekundi kaskada opet prepoznaje taj isti objekt te ga opet označi sa žutim pravokutnikom i to se ponavlja u krug s velikom brzinom. Da bi se problem lakše prikazao, ubačen je brojač koji je zadužen za brojanje broja automobila koji su detekritani na snimci (Slika 31.). Na taj način se vidi da je detektiran puno veći broj automobila od stvarnog broja automobila na slici.

```
import cv2

cascade_src = 'cars.xml'

video_src = 'video1.avi'

cap = cv2.VideoCapture(video_src)

car_cascade = cv2.CascadeClassifier(cascade_src)

counter = 0

while True:
    ret, img = cap.read()

    if (type(img) == type(None)):
        break

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    cars = car_cascade.detectMultiScale(gray, 1.1, 2)


    for (x,y,w,h) in cars:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,255),2)

    counter += len(cars)

    cv2.putText(img, f'Car Count: {counter}' , (10,30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

    cv2.imshow('video', img)

    if cv2.waitKey(33) == 27:
        break

cv2.destroyAllWindows()
```

Slika 31. Python kod za detekciju automobila s brojačem

U Python kod su dodane 3 nove linije. Varijabla brojača pod nazivom „*counter*“, funkcija za povečavanje brojača svaki put kad se automobil detektira „*counter += len(cars)*“ te ispis broja detektirnih automobila na videozapisu „*cv2.putText*“.

Slika 32. Rezultat brojača automobila

Na ovom videozapisu (Slika 32.) stvarni broj automobila koji su se pojavili iznosi oko 50 dok brojač pokazuje da je detektirano preko 800 automobila. Ispada da je svaki automobil detektiran oko 16 puta.

Za ovaj problem ne postoji konkretno rješenje, jedini način da se ovo popravi je kvalitetnija kaskada što zahtjeva bolji trening s više podataka, ali gotovo je nemoguće dobiti savršene rezultate bez titranja.

4.2.2. Detekcija pogrešnog objekta

Ovdje je problem što se može dobiti pogrešan rezultat detekcije. Kaskada ponekad može traženi objekt detekcije zamijeniti nekim sličnim objektom i tako dati pogrešan rezultat detekcije. Sve ovisi o snazi kaskade i koliko je podataka i vremena uloženo u trening. Što je kaskada jača, šansa za pogreškom je manja.

Kod detekcije vozila ovaj problem se nije primijetio, ali kod detekcije pješaka se dogodio nekoliko puta. Budući da na snimci tijelo pješaka dok stoji ima sličan oblik kao stupovi te su još istih ili jako sličnih boja (pješaci u crnoj odjeći i crni stup semafora), vrlo lako dolazi do zabune i pogreške.

Problem se najbolje vidi na (Slika 30.) gdje se može vidjeti semafor na sredini ceste koji je prepoznat kao pješak.

5. SPAJANJE VIŠE KASKADA U JEDAN PROGRAM ZA DETEKCIJU

5.1. Ideja programa

Većina prometnih snimki sadržava više vrsta vozila (automobili, motocikli, autobusi) i pješake (najčešće na pješačkim prijelazima i nogostupu). Ideja je da se jedan Python kod sastoji od više kaskada te tako detektira više različitih vrsta objekata na istoj snimci. Za početak je odabran videozapis „*bus2.mp4*“ koji se već koristio u prethodnom primjeru za detekciju autobusa. Taj videozapis je odabran iz razloga jer se na njemu nalaze automobili, autobus i pješak te vozila nisu previše udaljena od kamere što kaskadi omogućuje lakšu detekciju.

5.2. Stvaranje programa

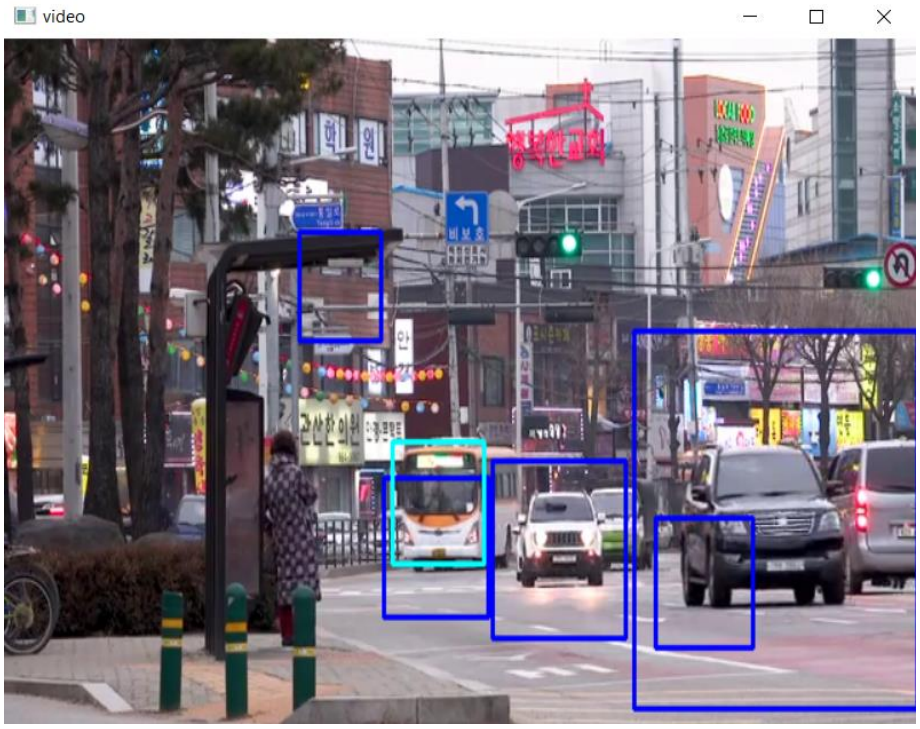
Korištene su kaskade za detekciju automobila, autobusa i pješaka iz prethodnih primjera. Sve 3 kaskade su prebačene u novu praznu mapu. Nakon toga je video „*bus2.mp4*“ prebačen u tu istu mapu te je još samo potrebno napisati kod za detekciju vozila. Radi lakšeg praćenja procesa, prvo su ubačene dvije kaskade (za detekciju autobusa i automobila) te je na kraju ubačena kaskada za detekciju pješaka.

```
import cv2
cascade1_src = 'cars.xml'
cascade2_src = 'Bus_front.xml'
cascade3_src = 'pedestrian.xml'
video_src = 'bus2.mp4'
cap = cv2.VideoCapture(video_src)
car_cascade = cv2.CascadeClassifier(cascade1_src)
bus_cascade = cv2.CascadeClassifier(cascade2_src)
ped_cascade = cv2.CascadeClassifier(cascade3_src)
new_window_size = (640, 480)
while True:
    ret, img = cap.read()
    if (type(img) == type(None)):
        break
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    cars = car_cascade.detectMultiScale(gray, 1.1, 2)
    buses = bus_cascade.detectMultiScale(gray, 1.1, 2)
    peds = ped_cascade.detectMultiScale(gray, 1.1, 2)
    for (x,y,w,h) in cars:
        cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
    for (x,y,w,h) in buses:
        cv2.rectangle(img, (x, y), (x+w, y+h), (255, 255, 0), 2)
    for (x,y,w,h) in peds:
        cv2.rectangle(img, (x, y), (x+w, y+h), (0, 0, 255), 2)
    resized_img = cv2.resize(img, new_window_size)
    cv2.imshow('video', resized_img)
    if cv2.waitKey(33) == 27:
        break
cap.release()
cv2.destroyAllWindows()
```

Slika 33. Python kod s 3 kaskade

5.3. Rezultati programa

Prvo je testiran Python kod s dvije kaskade („*cars.xml*“ i „*Bus_front.xml*“).

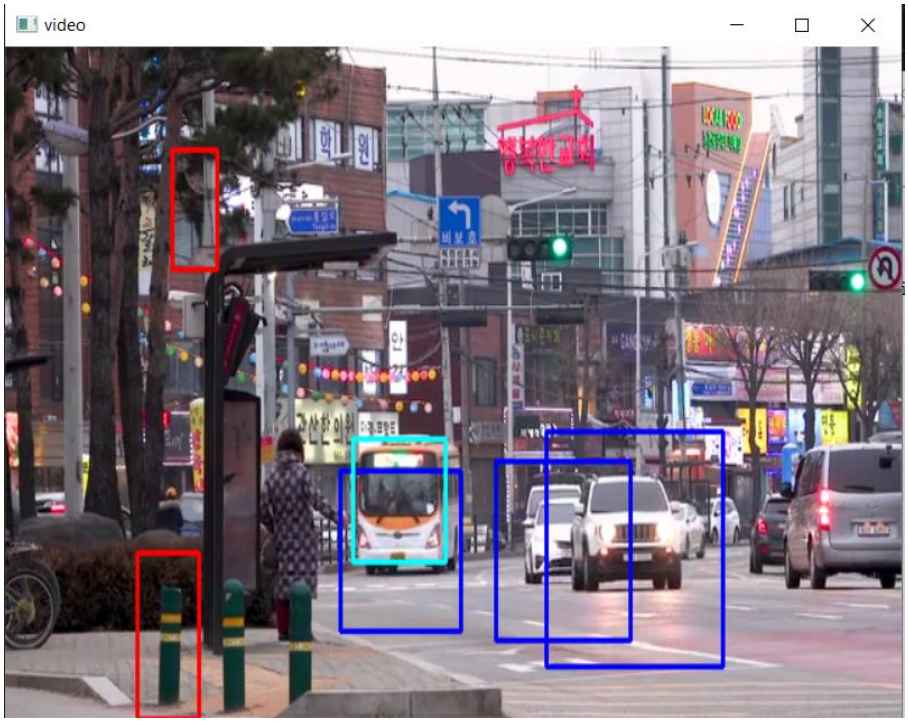


Slika 34. Rezultat višekaskadne detekcije 1

Iz slike (Slika 34.) je vidljivo da su rezultati detekcija automobila (tamnoplavi pravokutnici) precizni uz pojedine greške kao što je tamnoplavi pravokutnik na crvenoj zgradi što je uobičajeno za Haar kaskadni klasifikator koji vrlo lako traženi objekt može zamijeniti nekim pogrešnim objektom. Također je vidljivo da je autobus prepoznat i kao automobil i kao autobus (svijetloplavi pravokutnik).

Može se zaključiti da program s dvije kaskade relativno dobro prepoznaje različita vozila uz pojedine poteškoće koje su uobičajene za Haar kaskadni klasifikator.

Nakon testiranja programa s dvije kaskade, dodana je i treća kaskada – kaskada za prepoznavanje pješaka te sada Python kod izgleda kao na (Slika 33.).

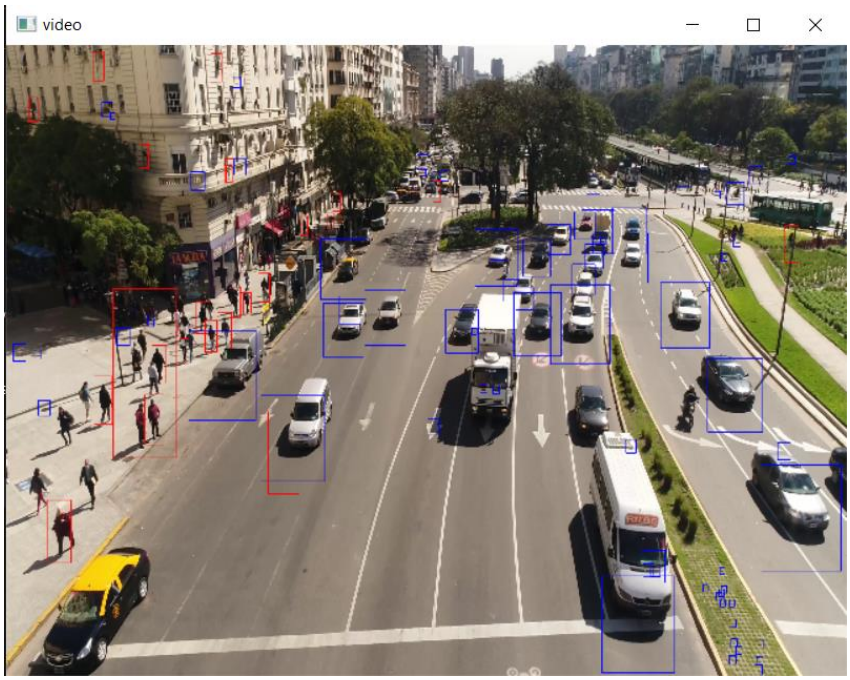


Slika 35. Rezultat višekaskadne detekcije 2

Rezultati (Slika 35.) su vrlo slični kao i u programu s dvije kaskade. Sve tri kaskade funkcioniraju te detektiraju tražene objekte iako je ovdje vidljivo da je detekcija pješaka (crveni pravokutnik) pogrešna. Taj problem je bio prisutan i u primjeru gdje je prikazana čista detekcija pješaka.

Što se tiče rada kaskada na snimci, može se zaključiti da dobro rade i da videozapis ne usporava zbog preopterećenja rada s 3 kaskade.

Kako bi se dobili što kvalitetniji rezultati, isti Python kod s 3 kaskade je primijenjen na još jednoj snimci na kojoj se nalazi veći broj vozila i pješaka.



Slika 36. Rezultat višekaskadne detekcije 3

Na ovoj slici (Slika 36.) je vidljivo da postoji puno više problema nego kod prethodnog primjera. Iako je većina automobila detektirana (plavi pravokutnik), vidljivo je da se plavi pravokutnici pojavljuju i na zelenoj površini, nogostupu i zgradi. Osim vozila, većina pješaka koji se nalaze bliže kameri su detektirani (crveni pravokutnik) dok se udaljeniji pješaci puno teže prepoznavaju ili se uopće ne prepoznaju. Kod detekcije pješaka također dolazi do problema pojavljivanja crvenih pravokutnika po zgradi.

Na temelju rezultata iz posljednje snimke vidljivo je da Haar kaskadni klasifikator neće jednako raditi na više različitih videozapisa. Potrebno je pronaći optimalnu snimku koja kaskadama neće stvarati probleme. Ako se pojavljuje jako veliki broj vozila i pješaka to uvelike otežava posao kaskadi te dolazi do problema kao što su prikazani iznad.

6. OSVRT NA OSTVARENE REZULTATE

6.1. Osvrt na rezultate Haar kaskadnog klasifikatora

Analizirajući dobivene rezultate, može se zaključiti da Haar kaskadni klasifikator relativno dobro detektira određenu vrstu objekata. Na primjer, ako je Haar kaskadni klasifikator zadužen samo za detekciju automobila, ta zadaća će biti vrlo dobro obavljena, odnosno velika većina automobila na videozapisu će biti prepoznata.

Kako se povećava broj kaskada u Haar kaskadnom klasifikatoru, tako se samom klasifikatoru otežava posao detekcije. Taj primjer se najbolje može uočiti na slici (Slika 36.) gdje je napravljen program za detekciju 3 različite vrste objekata (automobili, autobusi i pješaci). Klasifikator do određene mjere obavlja pravilnu detekciju, ali se pojavljuju i broje greške što sami klasifikator ne čini pouzdanim u prometnim detekcijama.

Osim otežane detekcije kod višekaskadnog programa, problem Haar kaskadnom klasifikatoru predstavlja i pogrešna detekcija sličnih objekata što je bio slučaj kod detekcije pješaka na slici (Slika 30.). Vidljivo je da se vrlo lako može dogoditi pogreška tako što će umjesto pješaka biti detektiran stup u ulici. Kod ostalih klasifikatora za detekciju također može doći do ovakvih pogrešaka, ali kod Haar kaskadnog sustava je to izraženije, pogotovo ako nije prije same detekcije napravljen dobar trening kojim bi se stvorila snažna kaskada.

Uz sve navedene probleme, pojavljuje se i problem titranja koji iznova detektira isti objekt za vrijeme trajanja videozapisa. Isti objekt se za vrijeme videozapisa detektira više puta što onemogućuje praćenje stvarnog broja detektiranih objekata (npr. automobila).

6.2. Usporedba rezultata Haar kaskadnog klasifikatora s ostalim metodama za detekciju

Budući da se kod detekcije vozila i pješaka u prometu pomoću Haar kaskadnog klasifikatora pojavljuju određene poteškoće, predložene su neke druge metode kao moguće rješenje. U uvodnom dijelu rada se spominjalo nekoliko metoda za detekciju (YOLO, SSD, R-CNN) pa je za kraj napravljena usporedba rezultata Haar kaskadnog klasifikatora i ostalih navedenih metoda.

Slika 37. Primjer upotrebe YOLO metode u prometu 1 [10]

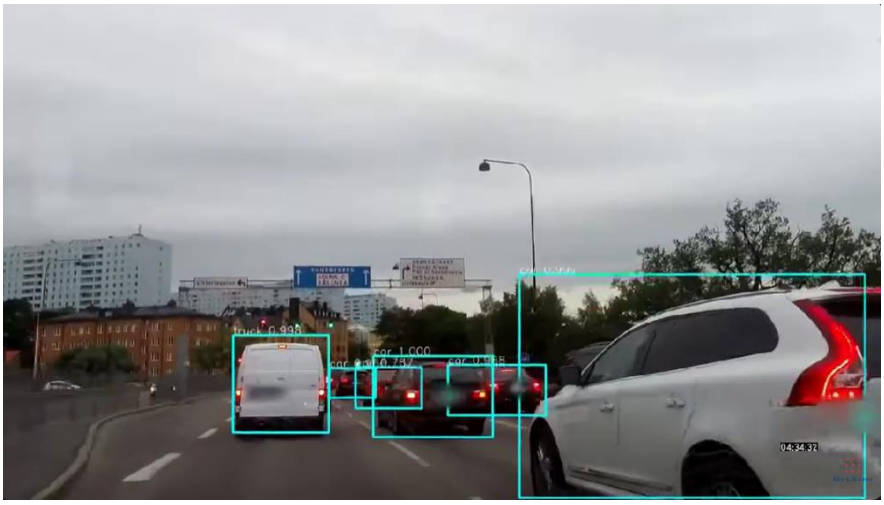
Slika 38. Primjer upotrebe YOLO metode u prometu 2 [10]

Yolo metoda je puno sigurnija i brža u detekciji objekata u prometu. Vozila na cesti prepoznaje dosta preciznije te za svaki detektirani objekt prikazuje o čemu je točno riječ (automobil, kamion, pješak, itd.). Također je dosta važno što se ne pojavljuje problem titranja te se lako može pratiti trenutni broj detektiranih objekata na kameri kao i ukupan broj detektiranih objekata za vrijeme trajanja videozapisa. (Slika 37. i Slika 38.)

Slika 39. Primjer upotrebe Mask R-CNN metode u prometu [11]

Mask R-CNN metoda u usporedbi s Haar kaskadnim klasifikatorom detekciju obavlja puno preciznije. Neuronske mreže omogućuju prepoznavanje detalja i oblika detektiranih objekata.

Mask R-CNN također može detektirati više objekata istovremeno. Kod Haar kaskadnog klasifikatora se iz prijašnjih rezultata moglo zaključiti da vrlo teško obavlja detekciju više objekata u gusto naseljenom području dok Mask R-CNN takve zadatke obavlja puno učinkovitije.



Slika 40. Primjer upotrebe SSD metode u prometu [12]

SSD metoda u usporedbi s Haar kaskadnim klasifikatorom je mnogo brža i preciznija. Također je skalabilna što znači da se može prilagoditi različitim veličinama slika i razlučivostima. Može detektirati više različitih klasa objekata istovremeno te je manje osjetljiva na varijacije u osvjetljenju.

7. ZAKLJUČAK

Haar kaskadni klasifikator se rijetko koristi za detekciju područja s gustim prometom. U radu se može vidjeti kako ova metoda može učinkovito prepoznavati vozila dok nije u pitanju povećana gustoća prometa. S povećanjem gustoće prometa, učinkovitost Haar kaskadnog klasifikatora opada.

Osim otežane detekcije pri povećanoj gustoći prometa, pojavljuju se i drugi problemi kao što su problem titranja i greške pri detekciji. Haar kaskadni klasifikator zahtijeva kvalitetan trening koji zahtjeva više vremena ako se žele postići kvalitetniji rezultati, ali ako je potrebno ostvariti rezultate koji su brzi i precizni, najbolje je koristiti metode kao što su YOLO, Mask R-CNN, SSD, itd.

U radu se mogu vidjeti usporedbe rezultata ostalih metoda (YOLO, Mask R-CNN, SSD) s Haar kaskadnim klasifikatorom i može se zaključiti da su te metode puno učinkovitije te se one puno više koriste u prometnoj detekciji.

Haar kaskadni klasifikator se najčešće koristi za detekciju ljudskih lica i gesta, pojedinih predmeta, a ako se koristi u prometu, najčešće je slučaj kod prometa manje gustoće.

LITERATURA

[1] YOLO: Algorithm for Object Detection: https://www.v7labs.com/blog/yolo-object-detection#how-does-yolo-work-yolo-architecture (15.12.2023.)

[2] You Only Look Once: Unified, Real-Time Object Detection; Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi: https://pjreddie.com/media/files/papers/yolo_1.pdf (15.12.2023.)

[3] Everything about Mash R-CNN: A Beginner's Guide: https://viso.ai/deep-learning/mask-r-cnn/ (15.12.2023.)

[4] SSD: Single Shot MultiBox Detector; Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg: https://ar5iv.labs.arxiv.org/html/1512.02325 (16.12.2023.)

[5] Understanding SSD MultiBox — Real-Time Object Detection In Deep Learning: https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab (16.12.2023.)

[6] OpenCV:Haar_Cascade_Classifier: https://docs.opencv.org/4.x/db/d28/tutorial_cascade_classifier.html (10.12.2023.)

[7] Vehicle-And-Pedestrian-Detection-Using-Haar-Cascades,GitHub_link: https://github.com/AdityaPai2398/Vehicle-And-Pedestrian-Detection-Using-Haar-Cascades (1.12.2023.)

[8] Haar Cascade Object Detector On Windows, YouTube link: https://www.youtube.com/watch?v=ydSXgBZ1ybk (1.12.2023.)

[9] Haar Cascade Object Detector Using GUI In Windows, YouTube link: https://www.youtube.com/watch?v=4BVDR7ggseI (1.12.2023.)

[10] YOLOv3 + Deep Sort tracking, YouTube link: https://www.youtube.com/watch?v=LyOxclvbshU&list=PLAzJXCSg9-Ze-e6C9xYEZ5vR-THHk5sMn (20.1.2024.)

[11] Traffic Demo (Mask RCNN), YouTube link: https://www.youtube.com/watch?v=RyNp4cl7YoQ (20.1.2024.)

[12] Vehicle Detection with SSD network, YouTube link: https://www.youtube.com/watch?v=rU1QP_GMQ4A (20.1.2024.)

PRILOZI

Prilog 1

Brojač automobila – youtube video:

https://www.youtube.com/watch?v=hUH9qwlfxcw

Prilog 2

Python kod za detekciju automobila na prvom videozapisu:

```
import cv2
cascade_src = 'cars.xml'
video_src = 'video.avi'
cap = cv2.VideoCapture(video_src)
car_cascade = cv2.CascadeClassifier(cascade_src)

while True:
    ret, img = cap.read()
    if (type(img) == type(None)):
        break
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    cars = car_cascade.detectMultiScale(gray, 1.1, 2)
    for (x,y,w,h) in cars:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,255),2)
    cv2.imshow('video', img)
    if cv2.waitKey(33) == 27:
        break

cv2.destroyAllWindows()
```

Prilog 3

Python kod za detekciju automobila na drugom videozapisu:

```
import cv2
cascade_src = 'cars.xml'
video_src = 'video1.avi'
cap = cv2.VideoCapture(video_src)
car_cascade = cv2.CascadeClassifier(cascade_src)

while True:
    ret, img = cap.read()
```

```
    if (type(img) == type(None)):
        break
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    cars = car_cascade.detectMultiScale(gray, 1.1, 2)
    for (x,y,w,h) in cars:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,255),2)
    cv2.imshow('video', img)
    if cv2.waitKey(33) == 27:
        break

cv2.destroyAllWindows()
```

Prilog 4

Python kod za detekciju vozila na dva kotača na prvom videozapisu:

```
import cv2
cascade_src = 'two_wheeler.xml'
video_src = 'video.mp4'
cap = cv2.VideoCapture(video_src)
bike_cascade = cv2.CascadeClassifier(cascade_src)
new_window_size = (640, 480)

while True:
    ret, img = cap.read()
    if (type(img) == type(None)):
        break
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    bikes = bike_cascade.detectMultiScale(gray,1.1, 1)
    for (x,y,w,h) in bikes:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,215),2)
    resized_img = cv2.resize(img, new_window_size)
    cv2.imshow('video', resized_img)
    if cv2.waitKey(33) == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

Prilog 5

Python kod za detekciju vozila na dva kotača na drugom videozapisu:

```
import cv2
cascade_src = 'two_wheeler.xml'
video_src = 'video1.mp4'
cap = cv2.VideoCapture(video_src)
bike_cascade = cv2.CascadeClassifier(cascade_src)
new_window_size = (640, 480)

while True:
    ret, img = cap.read()
    if (type(img) == type(None)):
        break
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    bikes = bike_cascade.detectMultiScale(gray,1.2, 1)
    for (x,y,w,h) in bikes:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,215),2)
    resized_img = cv2.resize(img, new_window_size)
    cv2.imshow('video', resized_img)
    if cv2.waitKey(33) == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

Prilog 6

Python kod za detekciju autobusa na prvom videozapisu:

```
import cv2
cascade_src = 'Bus_front.xml'
video_src = 'bus1.mp4'
cap = cv2.VideoCapture(video_src)
bus_cascade = cv2.CascadeClassifier(cascade_src)

while True:
    ret, img = cap.read()
    if (type(img) == type(None)):
        break
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    buses = bus_cascade.detectMultiScale(gray, 1.16, 1)
    for (x,y,w,h) in buses:
```

```
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255),2)
    cv2.imshow('video', img)
    if cv2.waitKey(33) == 27:
        break

cv2.destroyAllWindows()
```

Prilog 7

Python kod za detekciju autobusa na drugom videozapisu:

```
import cv2
cascade_src = 'Bus_front.xml'
video_src = 'bus2.mp4'
cap = cv2.VideoCapture(video_src)
bus_cascade = cv2.CascadeClassifier(cascade_src)

while True:
    ret, img = cap.read()
    if (type(img) == type(None)):
        break
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    buses = bus_cascade.detectMultiScale(gray, 1.16, 1)
    for (x,y,w,h) in buses:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255),2)
    cv2.imshow('video', img)
    if cv2.waitKey(33) == 27:
        break

cv2.destroyAllWindows()
```

Prilog 8

Python kod za detekciju autobusa na trećem videozapisu:

```
import cv2
cascade_src = 'Bus_front.xml'
video_src = 'bus3.mp4'
cap = cv2.VideoCapture(video_src)
bus_cascade = cv2.CascadeClassifier(cascade_src)

while True:
    ret, img = cap.read()
    if (type(img) == type(None)):
```

```
        break
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    buses = bus_cascade.detectMultiScale(gray, 1.16, 1)
    for (x,y,w,h) in buses:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255),2)
    cv2.imshow('video', img)
    if cv2.waitKey(33) == 27:
        break

cv2.destroyAllWindows()
```

Prilog 9

Python kod za detekciju pješaka na prvom videozapisu:

```
import cv2
video_src = 'video1.AVI'
cap = cv2.VideoCapture(video_src)
pedestrian_cascade = cv2.CascadeClassifier('pedestrian.xml')
new_window_size = (640, 480)

while True:
    ret, img = cap.read()
    if (type(img) == type(None)):
        break
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    pedestrians = pedestrian_cascade.detectMultiScale(gray,1.3,2)
    for(x,y,w,h) in pedestrians:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,210),4)
    resized_img = cv2.resize(img, new_window_size)
    cv2.imshow('video', resized_img)
    if cv2.waitKey(33) == 27:
        break

cv2.destroyAllWindows()
```

Prilog 10

Python kod za detekciju pješaka na drugom videozapisu:

```
import cv2
video_src = 'video2.mp4'
cap = cv2.VideoCapture(video_src)
pedestrian_cascade = cv2.CascadeClassifier('pedestrian.xml')
```

```
new_window_size = (640, 480)

while True:
    ret, img = cap.read()
    if (type(img) == type(None)):
        break
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    pedestrians = pedestrian_cascade.detectMultiScale(gray,1.3,2)
    for(x,y,w,h) in pedestrians:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,210),4)
    resized_img = cv2.resize(img, new_window_size)
    cv2.imshow('video', resized_img)
    if cv2.waitKey(33) == 27:
        break

cv2.destroyAllWindows()
```

Prilog 11

Python kod za detekciju automobila s brojačem:

```
import cv2
cascade_src = 'cars.xml'
video_src = 'video1.avi'
cap = cv2.VideoCapture(video_src)
car_cascade = cv2.CascadeClassifier(cascade_src)
counter = 0

while True:
    ret, img = cap.read()
    if (type(img) == type(None)):
        break
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    cars = car_cascade.detectMultiScale(gray, 1.1, 2)
    for (x,y,w,h) in cars:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,255),2)
    counter += len(cars)
    cv2.putText(img, f'Car Count: {counter}' , (10,30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,
0, 255), 2)
    cv2.imshow('video', img)
    if cv2.waitKey(33) == 27:
        break
cv2.destroyAllWindows()
```

Prilog 12

Python kod s 2 kaskade na prvom videozapisu:

```
import cv2
cascade1_src = 'cars.xml'
cascade2_src = 'Bus_front.xml'
video_src = 'bus2.mp4'
cap = cv2.VideoCapture(video_src)
car_cascade = cv2.CascadeClassifier(cascade1_src)
bus_cascade = cv2.CascadeClassifier(cascade2_src)
new_window_size = (640, 480)

while True:
    ret, img = cap.read()
    if (type(img) == type(None)):
        break
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    cars = car_cascade.detectMultiScale(gray, 1.1, 2)
    buses = bus_cascade.detectMultiScale(gray, 1.1, 2)
    for (x,y,w,h) in cars:
        cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
    for (x,y,w,h) in buses:
        cv2.rectangle(img, (x, y), (x+w, y+h), (255, 255, 0), 2)
    resized_img = cv2.resize(img, new_window_size)
    cv2.imshow('video', resized_img)
    if cv2.waitKey(33) == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

Prilog 13

Python kod s 3 kaskade na prvom videozapisu:

```
import cv2
cascade1_src = 'cars.xml'
cascade2_src = 'Bus_front.xml'
cascade3_src = 'pedestrian.xml'
video_src = 'bus2.mp4'
cap = cv2.VideoCapture(video_src)
car_cascade = cv2.CascadeClassifier(cascade1_src)
bus_cascade = cv2.CascadeClassifier(cascade2_src)
```

```
ped_cascade = cv2.CascadeClassifier(cascade3_src)
new_window_size = (640, 480)

while True:
   ret, img = cap.read()
   if (type(img) == type(None)):
      break
   gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
   cars = car_cascade.detectMultiScale(gray, 1.1, 2)
   buses = bus_cascade.detectMultiScale(gray, 1.1, 2)
   peds = ped_cascade.detectMultiScale(gray, 1.1, 2)
   for (x,y,w,h) in cars:
      cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
   for (x,y,w,h) in buses:
      cv2.rectangle(img, (x, y), (x+w, y+h), (255, 255, 0), 2)
   for (x,y,w,h) in peds:
      cv2.rectangle(img, (x, y), (x+w, y+h), (0, 0, 255), 2)
   resized_img = cv2.resize(img, new_window_size)
   cv2.imshow('video', resized_img)
   if cv2.waitKey(33) == 27:
      break

cap.release()
cv2.destroyAllWindows()
```

Prilog 14

Python kod s 3 kaskade na drugom videozapisu:

```
import cv2
cascade1_src = 'cars.xml'
cascade3_src = 'pedestrian.xml'
video_src = 'video.mp4'
cap = cv2.VideoCapture(video_src)
car_cascade = cv2.CascadeClassifier(cascade1_src)
ped_cascade = cv2.CascadeClassifier(cascade3_src)
new_window_size = (640, 480)

while True:
   ret, img = cap.read()
   if (type(img) == type(None)):
      break
   gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
    cars = car_cascade.detectMultiScale(gray, 1.1, 2)
    peds = ped_cascade.detectMultiScale(gray, 1.1, 2)
    for (x,y,w,h) in cars:
        cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
    for (x,y,w,h) in peds:
        cv2.rectangle(img, (x, y), (x+w, y+h), (0, 0, 255), 2)
    resized_img = cv2.resize(img, new_window_size)
    cv2.imshow('video', resized_img)
    if cv2.waitKey(33) == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

Prilog 15

Originalni i obrađeni videomaterijali koji su se koristili u ovom radu:

Priloženo na USB uređaju.