Razvoj aplikacije za detekciju i raspoznavanje prometne signalizacije iz video signala korištenjem modela dubokog učenja

Franjković, Jure

Undergraduate thesis / Završni rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:168:715808

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-24**

Repository / Repozitorij:

FER Repository - University of Zagreb Faculty of Electrical Engineering and Computing repozitory

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1431

RAZVOJ APLIKACIJE ZA DETEKCIJU I RASPOZNAVANJE PROMETNE SIGNALIZACIJE IZ VIDEO SIGNALA KORIŠTENJEM MODELA DUBOKOG UČENJA

Jure Franjković

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1431

RAZVOJ APLIKACIJE ZA DETEKCIJU I RASPOZNAVANJE PROMETNE SIGNALIZACIJE IZ VIDEO SIGNALA KORIŠTENJEM MODELA DUBOKOG UČENJA

Jure Franjković

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1431

Pristupnik:	**Jure Franjković (0036543210)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	doc. dr. sc. Marko Horvat

Zadatak: **Razvoj aplikacije za detekciju i raspoznavanje prometne signalizacije iz video signala korištenjem modela dubokog učenja**

Opis zadatka:

Neuronske mreže dubokog učenja naziv je za skupa modela umjetnih neuronskih mreža koje kroz višeslojnu obradu podataka konstruiraju apstrakcije visoke razine. Ti su se modeli pokazali osobito učinkoviti u nizu zadataka iz domene računalnog vida. Posebice su pokazali svoju visoku primjenjivost u detekciji i raspoznavanju objekata u video signalu u stvarnom vremenu i realističnim uvjetima korištenja. Cilj ovog rada je izraditi aplikaciju koja korištenjem naprednih inačica radnog okvira otvorenog koda YOLO (You Only Look Once), temeljenog na dubokom učenju, detektira i raspoznaje vertikalnu prometnu signalizaciju iz video signala. Upoznati se s radnim okvirom YOLO i modelima dubokog učenja. Izraditi dovoljno reprezentativne video isječke prometne signalizacije u stvarnim uvjetima prometnica, te ih pohraniti i obraditi. Koristeći razvojno okruženje u programskom jeziku Python izraditi aplikaciju koja će implementirati traženu funkcionalnost. Provesti eksperimentalno vrednovanje, uključujući i referentni model te statistički obraditi rezultate. Prikazati komponente izrađenog sustava, bitne isječke izvornog programskog koda te definirati korištena programska sredstva i potrebne postupke. Radu priložiti izvorni i izvršni kod razvijenog sustava uz potrebna dodatna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 14. lipnja 2024.

Iskreno zahvaljujem doc. dr. sc. Marku Horvatu na njegovoj pomoći i podršci tijekom izrade mog završnog rada.

Sadržaj

Uvod 1

1. Detekcija prometnih znakova i semafora 3

1.1. Značaj i primjena modela za detekciju prometnih znakova i semafora 3

1.2. Model detekcije prometnih znakova i semafora 4

2. Korištene tehnologije 7

2.1. YOLOv8 7

2.2. Kaggle 8

2.3. Programski jezik Python 12

2.4. Ostale tehnologije 13

3. Implementacija programskog rješenja 15

3.1. Ultralytics 15

3.1.1. Kako koristiti Ultralytics YOLO? 16

3.2. Treniranje modela 17

3.3. Aplikacija 21

3.4. Upute za pokretanje 23

4. Rezultati 25

4.1. Korištene metrike evaluacije detektora objekata 25

4.2. Rezultati dobivenog modela 28

4.3. Završna usporedba s drugim radom 32

Zaključak 33

Literatura 34

Sažetak 37

Summary 38

Uvod

Metode dubokog učenja su podskup metoda strojnog učenja koje su bazirane na umjetnim neuronskim mrežama [1]. Pod pojam „duboko“, podrazumijeva se veliki broj skrivenih slojeva neuronske mreže koja oponaša složeno ponašanje ljudskog mozga [2]. Već od otkrića računala (prvo računalo *ENIAC* [3] proizvedeno je 1955. godine), i zbog njegove sposobnosti za brzo računanje niza matematičkih operacija, pokušavao se pronaći način kako simulirati računalo kao ljudski mozak [4]. Prva su se istraživanja i pokušaji dubokog učenja javili 1960-ih godina, a danas, nešto više od 60 godina kasnije, imamo modele dubokog učenja koji su sposobni razgovarati i razumjeti čovjeka, naučiti nešto u vremenu puno kraćem nego što bi ljudskom biću trebalo. Računalni vid (engl. *Computer vision*) [5] je područje umjetne inteligencije koje se bavi prepoznavanje dvodimenzionalnih odnosno trodimenzionalnih predmeta. U početku, ono što je dijelilo računalni vid od običnog procesiranja digitalnih fotografija je bio cilj razumijevanja cijelog sadržaja fotografije. 1990-ih, prvi su se puta krenule koristiti statističke metode u učenju računalnog vida za prepoznavanje ljudskih lica što je ubrzalo njegov razvoj [5].

Razvoj računalnog vida u posljednjih desetak godina znatno je ubrzan zahvaljujući napretku u dubokom učenju odnosno korištenjem konvolucijskih neuronskih mreža (engl. *Convolutional neural network, CNN*) [6]. Danas je računalni vid jedno od najbrže rastućih područja u računarstvu zbog svoje široke primjenjivosti u raznim industrijama. Automobilska je industrija jedan od primjera industrije koja koristi računalni vid za navigaciju, prepoznavanje prepreka, prometnih znakova i omogućuje pomoćne sustave vozaču. U konačnici, računalni vid je razlog zašto danas na svijetu postoje autonomna vozila (engl. *Self-driving vehicle*) [7] koja imaju sposobnost upravljanja vozilom s minimalnom ili bez čovjekove pomoći. Duboko učenje može trajati jako dugo ako nemamo grafičku karticu (engl. *graphics processing unit, GPU*) odgovarajuće brzine. Razvoj grafičkih kartica omogućio je složenije modele i algoritme, čime su se značajno poboljšale performanse i točnost sustava računalnog vida. Iako grafičke kartice značajno poboljšavaju performanse, još uvijek postoje izazovi vezani za troškove i potrošnju energije. Treniranje velikih modela zahtijeva značajna financijska ulaganja u hardver i može biti energetski intenzivno.

Računalni vid će zasigurno nastaviti imati ključnu ulogu u budućim inovacijama, mijenjajući različite industrije i svakodnevni život.

Cilj ovog rada je napraviti računalnu aplikaciju koja primjenjuje model dubokog učenja treniran na skupu podataka i koja omogućuje detekciju prometnih znakova i semafora iz video signala. Izradom ove aplikacije, moguće ju je implementirati u druge korisne projekte poput sustava za detekciju prometnih znakova u vozilima i raditi analize o brojnim parametrima (svjetlina, oštrina fotografije…) prisutnim u video signalu. Unutar ovog rada bit će objašnjeni postupci i tehnologije koji su omogućili izradu modela za detekciju objekata iz video signala. Bit će navedena točnost dobivenog modela i njegove metrike koje će se usporediti s drugim znanstvenim radovima na istu tematiku i s istim modelom dubokog učenja kako bi imali što precizniju usporedbu. Također, bit će objašnjene korištene tehnologije koje su uvelike pomogle i ubrzale razvoj ovog projekta, a posebice web aplikacija *Kaggle* [8] koja nudi puno mogućnosti svojim korisnicima, a jedna od njih je korištenje efikasne grafičke kartice.

1. Detekcija prometnih znakova i semafora

1.1. Značaj i primjena modela za detekciju prometnih znakova i semafora

Cestovni promet je najrazvijeniji oblik prometa i nema države na svijetu koja nema ceste, pa tako i prometne znakove i semafore. Detekcija prometnih znakova i semafora ima veliku primjenu u današnjem svijetu. Navedimo neke primjere:

- Autonomna vozila
  - Autonomna vozila koriste modele za detekciju prometnih znakova i semafora kako bi sigurno upravljala vozilom na cestama, poštivala prometna pravila, prepoznala zaustavne znakove, ograničenje brzine i signalizaciju na semaforima
- Napredni sustavi pomoći vozaču
  - Napredni sustavi pomoći vozaču (engl. *Advanced driver-assistance system*, *ADAS*) koriste ovakve modele za pružanje podrške pri vožnji, poput upozorenja na prebrzu vožnju, automatskog kočenja pri crvenom svjetlu na semaforu, ili asistencije pri zadržavanju ograničenja brzine
- Pametni gradovi
  - U pametnim gradovima (engl. *Smart city*), detekcija prometnih znakova i semafora koristi se za optimizaciju prometa, smanjenje gužvi i poboljšanje sigurnosti na cestama kroz prilagodljive sustave za upravljanje prometom
- Sustavi za nadzor i provođenje zakona
  - Policija i druge službe koriste modele za detekciju prometnih znakova i semafora za nadzor prometa, identifikaciju prekršaja poput prolaska kroz crveno svjetlo ili prekoračenja brzine, te za opću regulaciju prometa
- Mobilne navigacijske aplikacije
  - Navigacijske aplikacije koriste ove modele za pružanje ažurnih informacija korisnicima, popu upozorenja na nadolazeće prometne znakove, ograničenja brzine, i promjene u signalizaciji na semaforima.

- Simulacije i treniranje autonomnih sustava
  - U simulacijama za treniranje autonomnih vozila i sustava, detekcija prometnih znakova i semafora koristi se za stvaranje realističnog okruženja u kojem sustavi mogu učiti i testirati svoje sposobnosti prije primjene u stvarnom svijetu
- Edukacija i obuka vozača
  - Modeli za detekciju prometnih znakova i semafora koriste se u edukacijskim programima i simulacijama za obuku novih vozača. Ovi sustavi mogu pružiti realistične scenarije u kojima se vozači uče prepoznati i pravilno reagirati na prometne znakove i semafore. Također, simulacije mogu uključivati ocjenjivanje reakcije vozača u različitim prometnim situacijama, što pomaže u poboljšanju njihovih vozačkih vještina i pripremi za stvarne uvjete na cesti
- Poboljšanje infrastrukture i planiranje gradova
  - Urbanistički arhitekti i inženjeri koriste modele za detekciju prometnih znakova i semafora za analizu trenutne prometne infrastrukture i identificiranje područja koja zahtijevaju poboljšanja. Na temelju podataka prikupljenih ovim modelima, mogu se donijeti informirane odluke o postavljanju novim prometnih znakova, semafora, i drugih elemenata prometne signalizacije kako bi se povećala sigurnost i učinkovitost prometa u gradovima

1.2. Model detekcije prometnih znakova i semafora

Modeli dubokog učenja mogu doći unaprijed trenirani (engl. *pretrained model*) ili moraju biti učeni od početka na određenom skupu podataka. Model dubokog učenja koji se koristi u ovom projektu zove se *YOLO* (You Only Look Once) i služit će za detekciju prometnih znakova i semafora jednom kada je istreniran.

U svijetu postoji jako puno različitih prometnih znakova i vrsta prometnih znakova. Različite zemlje imaju znakove koji označavaju iste stvari, ali ipak se u nekim detaljima razlikuju (Sl. 1.1).

Slika 1.1 Prometni znak sporedne ceste u Hrvatskoj (lijevo) i Poljskoj (desno)

Kako bi model dubokog učenja mogli istrenirati da prepoznaje prometne znakove, potreban mu je veliki broj fotografija koje sadrže prometne znakove i semafore. U ovom projektu bit će korištene slike prometnih znakova u Hrvatskoj.

Jednom kada imamo potrebne fotografije za obradu, potrebno je modelu dubokog učenja označiti objekte koji nas zanimaju unutar slike s pomoću graničnih okvira (engl. *bounding box*). To radimo tako da unutar fotografije ucrtavamo granične okvire tako da uzimamo koordinate točaka koje će predstavljati određeni pravokutnik i brojem označavamo klasu objekta koji je obuhvaćen unutar označenog pravokutnika. Jedna stvar koja se treba odlučiti prije samog obilježavanja fotografija je broj klasa koje želimo da naš model, jednom kad je istreniran, može prepoznati. Taj podatak se modelu dubokog učenja najčešće zadaje s pomoću *YAML* datoteke unutar koje se navodi broj klase i ime klase. Oblik oznaka koje *YOLO* prima moraju imati strogo zadani oblik koji je sljedeći: <ime_klase> <x-centar> <y-centar> <širina> <visina> [9], kao na primjer: „0 0.598508 0.611257 0.170863 0.222513“. Oznaka „x-centar“ predstavlja „x“ koordinatu središta nacrtanog pravokutnika, „y-centar“ „y“ koordinatu središta nacrtanog pravokutnika, dok preostala dva parametra predstavljaju širinu i visinu pravokutnika. Sve oznake su normalizirane unutar raspona [0, 1] s obzirom na širinu i visinu cijele fotografije. Treba napomenuti da YOLO može obrađivati oblike koji nisu pravokutnici, ali zbog jednostavnosti obilježavanja i brzine učenja (omogućavanje oblika koji nisu pravokutnici usporava učenje modela) modela, koristit ćemo samo pravokutne oblike unutar ovog projekta. Za imena klasa ovog projekta, koristit će se „traffic_sign“ za prometni znak i „traffic_light“ za semafor. Razlog zbog kojeg svaki znak nije stavljen u posebnu klasu ili grupa znakova u svoju klasu je velika količina posla kod označavanja fotografije što nije glavna tema ovog projekta.

Nakon što su sve fotografije prikupljene i označene, valja ih podijeliti u skup fotografija za treniranje, validaciju i testiranje. Skup za treniranje treba biti najveći, ali ne smijemo zanemariti niti skup za validaciju i testiranje. Validacijski skup nam omogućuje da pratimo koliko dobro model generalizira objekte za detekciju i iznimno je bitan kako ne bi došlo do prenaučenosti (engl. *overfitting*). U ovom projektu, skup za treniranje bio je 80 % ukupnog broja slika, skup za validaciju 10 % ukupnog broja slika i 10 % slika je bio skup za testiranje.

Sljedeći problem koji predstavlja treniranje modela dubokog učenja je brzina učenja [25]. Kao što je navedeno u uvodu, potrebna je efikasna grafička kartica za dovoljno brzo

izvođenje dijela treniranja. S obzirom na to da su grafičke kartice skupe i energetski su iznimno neefikasne, u ovom projektu koristit će se web platforma *Kaggle* koja nudi najefikasnije grafičke kartice trenutno dostupne na tržištu za besplatno. Kasnije ćemo detaljnije proći kroz mogućnosti *Kagglea*. Nakon što su svi prethodni koraci ostvareni, odabrani model spreman je za treniranje.

2. Korištene tehnologije

2.1. YOLOv8

Programski okvir YOLOv8 (engl. *You Only Look Once*), druga je najnovija iteracija YOLO algoritma, poznata po svojoj brzini i točnosti u detekciji objekata [10]. Ovaj model koristi jedinstvenu arhitekturu neuronske mreže koja omogućuje brzu i efikasnu detekciju objekata unutar jedne mrežne prolaznosti. Ključne značajke YOLO v8 su [11]:

- Brzina
  - Model je dizajniran za rad u stvarnom vremenu. Njegova arhitektura omogućuje obradu preko 500 sličica po sekundi (engl. *Frames per second, FPS*) [12], što ga čini idealnim za primjene koje zahtijevaju visoku preciznost i brz odziv, kao što su autonomna vozila i sustavi nadzora.
- Točnost
  - Donosi poboljšanja u preciznosti detekcije objekata u odnosu na prethodne inačice modela zahvaljujući unaprijeđenim tehnikama treniranja, optimiziranoj arhitekturi mreže i boljoj generalizaciji na različitim skupovima podataka.
- Jednostavnost korištenja
  - *YOLO* modeli su poznati po svojoj jednostavnosti implementacije i primjene. Postoje brojni alati i resursi, poput *Roboflowa,* uključujući unaprijed trenirane modele koji se mogu lako prilagoditi specifičnim zadacima.
- Jednostavna arhitektura
  - Koristi konvolucijske neuronske mreže za efikasno pronalaženje značajki na slikama. Njegova arhitektura omogućuje detekciju višestrukih objekata u jednoj slici uz zadržavanje visokih performansa u području brzine i preciznosti [13].

Arhitektura modela podijeljena je na tri dijela [14]:

- *Backbone* mreža
  - YOLOv8 koristi varijantu Darknet arhitekture nazvanu CSPDarknet53, koja uključuje *Cross-Stage Partial* (CSP) veze. Ove veze poboljšavaju protok informacija između različitih slojeva mreže i olakšavaju gradijentni tok tijekom treniranja, što rezultira boljom točnošću i efikasnošću
- *Neck* struktura
  - *Path Aggregation Network* (PANet) koristi se za bolji protok informacija između različitih prostornih rezolucija, omogućujući modelu da efikasno hvata višeslojne značajke. Ovo je ključno za prepoznavanje objekata različitih veličina i u različitim scenarijima
- *Detection Head*
  - Dinamičko dodjeljivanje sidra [15] (engl. *Dynamic Anchor Feature Selection*): YOLO v8 koristi dinamičko dodjeljivanje sidra i novu *Intersection over Union* (IoU) funkciju gubitka, što poboljšava točnost predviđanja granica okvira i bolje rukovanje preklopljenim objektima

2.2. Kaggle

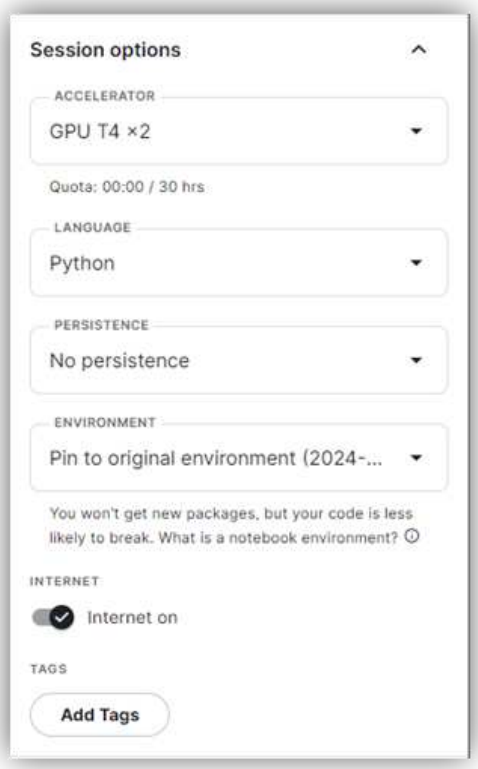
Kaggle je online platforma koja omogućava znanstvenicima, analitičarima i programerima da se međusobno povežu, natječu, uče i dijele svoja rješenja vezana uz strojno učenje i podatke [16]. Osnovan, je 2010. godine, a 2017. godine ga je preuzeo Google.

Kaggle se koristi za natjecanja u strojnom učenju gdje korisnici mogu sudjelovati u rješavanju stvarnih problema koristeći stvarne skupove podataka. Natjecanja su često sponzorirana od strane velikih tvrtki koje nude novčane nagrade za najbolje modele i rješenja. Osim velikih natjecanja s velikim nagradama sponzoriranih od strane tvrtki, postoje natjecanja fokusirana na akademska istraživanja s manjim nagradama, natjecanja namijenjena početnicima kako bi naučili osnove i manja natjecanja za eksperimentiranje bez veliki nagrada.

Kaggle omogućuje pristup brojnim skupovima podataka koje korisnici mogu koristiti za vježbu, istraživanje i razvoj svojih modela. Skupovi podataka su raznoliki, uključujući sve od zdravstvenih podataka do podataka o tržištu dionica.

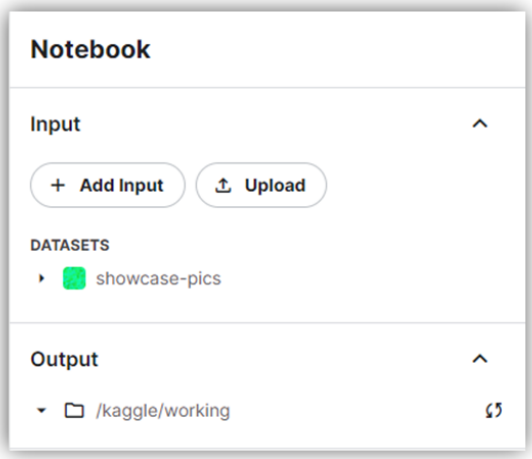
Još jedna prednost Kagglea je da omogućuje korisnicima da dijele svoje kodove i analize putem takozvanih „Kernels“ ili bilježnica (engl. *Notebook*). Ova online okruženja za programiranje koriste Python ili R programske jezike i mogu uključivati sve od jednostavnih skripti do složenih modela dubokog učenja. Dakle, koriste se dvije vrste bilježnica: „Jupyter bilježnica“ (engl. *Jupyter Notebook*) za izvođenje Python koda, vizualizaciju podataka i dokumentiranje svog rada i „R bilježnica“ (engl. *R Notebook*) za pisanje koda u R jeziku, omogućavajući identičnu funkcionalnost kao i „Jupyter bilježnica“. Velika prednost ovih online okruženja za programiranje i pisanje koda je u tome što Kaggle pruža besplatnu računalnu snagu, uključujući procesor, grafičku karticu i TPU (engl. *Tensor processing unit*), što omogućava treniranje kompleksnih modela bez lokalnih resursa. Osim toga, moguće je namjestiti postavke programskog jezika koji se koristi, trajnost podataka između pojedinih sesija bilježnice (isključiti ako nam stari podaci ne trebaju pri svakom pokretanju), okruženje koje određuje specifične verzije softvera i internet koji može biti omogućen za korištenje unutar bilježnice ili ne (Sl. 2.1). Treba napomenut da se grafička kartica ne može koristiti cijelo vrijeme već ima ograničenje od 30 sati uporabe koji se ponovno dodijele korisniku svakih tjedan dana. Isto tako, Kaggle omogućava pokretanje dvije paralelne bilježnice koje, obje mogu koristiti dodatne računalne resurse. Ako imamo dva paralelna programa pokrenuta jednu minutu i svaki od njih koristi grafičku karticu, naša uporaba grafičke kartice bit će zabilježena kao period od dvije minute, a ne od jedne.

Osim toga, postoji ograničenje na maksimalno izvršavanje pojedine bilježnice koje iznosi 12 sati i koje ako se prijeđe zaustavlja izvršavanje programa i ne sprema nikakve rezultate pod korisnikov profil (korisnički račun). Treba biti pažljiv s vremenom potrebnim za treniranje modela kako ne bi prekoračilo zadani limit, jer iako na kraju kod nije dovršen do kraja, Kaggle to i dalje računa kao korištenje grafičke kartice.



Slika 2.1 Postavke akceleratora, jezika, trajnosti i okruženja za bilježnicu

Unutar Kaggle notebooka, mogu se zadati podaci koji će se koristiti unutar koda, odnosno *input* prostor koji je uvijek mapiran na putanju „kaggle/input“. Isto tako, postoji rezervirani prostor za podatke generirane kroz kod koji je mapiran na putanju „kaggle/ouput“ (Sl. 2.2). Pod *input* možemo dodavati naše skupove podataka, ali i ostale podatke po želji. Kaggle *input* također ima limit od 20 gigabajta isto kao i *output* prostor.



Slika 2.2: Prikaz input i output prostora za podatke

Svaka bilježnica ima verzioniranje (Sl. 2.3) što omogućava da se svaka promjena sprema kao nova verzija, omogućavajući korisnicima povratak na prethodne verzije ako je potrebno.

Slika 2.3: Verzije istog Notebooka

Veliki broj popularnih paketa za strojno učenje i podatkovnu znanost je unaprijed instaliran, ali dopušteno je preuzimanje i korištenje velikog broja ostalih paketa koji su dostupni na internetu ili putem programa *pip* (engl. *preferred installation program*).

Kaggle ima aktivnu zajednicu gdje korisnici mogu postavljati pitanja, dijeliti savjete i surađivati na projektima. Diskusije su koristan resurs za učenje i rješavanje problema. Platforma nudi brojne tečajeve koji pokrivaju osnove znanosti o podacima, strojnom učenju i naprednim tehnikama. Ovi tečajevi su dizajnirani kako i za početnike i za napredne korisnike.

2.3. Programski jezik Python

Python je programski jezik opće namjene kojeg je dizajnirao Guido van Rossum i prvi put objavio 1991. godine. Python je poznat po svojoj čitljivosti i jednostavnosti sintakse što ga čini popularnim izborom među programerima [17]. Python omogućava dinamičko tipiziranje podataka što daje veću fleksibilnost programerima, ali i povećava šansu za pogreškama. Velika standardna biblioteka koju Python omogućava je dovoljna za rješavanje većine osnovnih zadataka i pisanja skripti za privatnu upotrebu. Još jedna prednost Pythona je podržavanje različitih paradigmi programiranja, uključujući proceduralno, objektno-orijentirano (engl. *Object Oriented Programming, OOP*) i funkcionalno programiranje.

Guido van Rossum je počeo raditi na Pythonu kao nasljedniku jezika ABC, a 1991. je objavljena prva verzija Pythona (0.9.0), koja je uključivala funkcije poput upravljanja iznimkama (engl. *Exception handling*), funkcija i osnovnih tipova podataka. 2000. godine je objavljena Python 2.0 verzija koja je donijela mnoge nove značajke uključujući potporu za Unicode. 2020. Python 2 službeno je prestao biti podržavan, a Python 3 s još više novih mogućnosti je postao standard koji se i danas koristi.

Python je postao jedan od vodećih jezika za strojno učenje i duboko učenje iz nekoliko ključnih razloga.

- Bogati ekosustav biblioteka i okvira
  - *TensorFlow*: Open-source biblioteka za duboko učenje razvijena od strane Googlea, omogućuje treniranje i implementaciju velikih neuronskih mreža
  - *Keras*: API visoke razine za duboko učenje razvijen od strane Googlea za implementaciju neuronskih mreža
  - *Pytorch*: Razvijen od strane Facebooka, *PyTorch* je popularan za istraživanje i razvoj zbog svoje fleksibilnosti i jednostavnosti korištenja.
  - *Scikit-learn*: Biblioteka za strojno učenje koja nudi jednostavne i efikasne alate za analizu podataka i modeliranje
  - *Pandas*: Alat za manipulaciju podacima i analizu, koji je posebno koristan za pripremu podataka u projektima strojnog učenja
- Jednostavnost i čitljivost koda
  - Pythonova jednostavna sintaksa omogućava brži razvoj prototipova i olakšava suradnju među timovima koji uključuju članove s različitim razinama programerskog znanja
- Integracija s drugim jezicima i alatima
  - Python se lako integrira s jezicima nižih razina kao što su C/C++ (za optimizaciju performansi) i s alatima kao što su *Apache Spark* za obradu velikih podataka.

- Vizualizacija podataka
  - Biblioteke kao što su *Matplotlib* i *Seaborn* omogućuju naprednu vizualizaciju podataka, što je ključno za analizu i interpretaciju rezultata strojnog učenja

Model dubokog učenja korišten u ovom projektu, *YOLO*, također dolazi s bibliotekom *Ultralytics* koja omogućava da se model vrlo jednostavno postavi u kodu, a zatim da se s tim istim modelom omogući izvršavanje različitih zadataka poput detekcije, predviđanja podataka, treniranja modela i mnogo drugih mogućnosti.

2.4. Ostale tehnologije

Za izradu aplikacije i testiranje samog Python koda korišteno je integrirano razvojno okruženje (engl. *integrated development environment, IDE*) *PyCharm.* On korisniku daje veliki izbor mogućnosti koje olakšavaju razvoj Python aplikacija, posebno za velike projekte. Neke od glavnih značajki su:

- Sintaksno isticanje za poboljšanje čitljivosti koda
- Automatsko dovršavanje koda
- Način rada za otkrivanje i ispravljanje grešaka (engl. Debugging mode)
- Integracija sa sustavom za praćenje verzija koda (engl. *version control system*)
- Podrška za web razvoj uz alate za velik broj web okvira (engl. *web framework*)

Za upravljanje verzijama koda i suradnju, unutar ovog projekta se koristi aplikacija Github koja koristi Git sustav kontrole verzija. Github je ključno sredstvo za timski rad i otvoreni razvoj softvera, omogućavajući učinkovitu suradnju i dijeljenje koda. Neke značajke koje Github daje svojim korisnicima su:

- Pohrana i upravljanje kodom putem udaljenog repozitorija
- Suradnju među programerima kroz *pull requestove* i recenzije koda
- Integracija CI/CD cjevovoda i testiranja
- Praćenje problema i upravljanje zahtjevima za funkcionalnost
- Veliki broj projekata za otvoreni razvoj unutar zajednice programera i inženjera

Jedna vrlo korisna Python biblioteka korištena u ovom projektu je *Albumentations*. Služi za augmentaciju slika koja je posebno korisna u području računalnog i vida i dubokog učenja.

Nudi razne tehnike augmentacije poput rotacija, translacija, skaliranja, zamućenja i puno drugih. Kompatibilna je s bibliotekama *PyTorch* i *TensorFlow* i optimizirana je za brzo izvođenje augmentacija zato što podržava obradu na grafičkoj kartici. Također, omogućuje kombiniranje više transformacija u jednoj operaciji.

Neke od tehnika augmentacije koje su korištene u ovom projektu su:

- RandomBrightnessContrast(*probability*)
  - Ova tehnika dodaje ili oduzima određeni intenzitet svjetlosti na cijeloj slici, čime se mijenja ukupna svjetlina slike. Također, mijenja razliku između najtamnijih i najsvjetlijih dijelova slike što dovodi do povećanja ili smanjenja kontrasta slike
- GaussNoise(*probability*)
  - Šum koji je dobiven iz normalne (Gaussove) distribucije dodaje se svakom pikselu slike, čime se simulira slučajni šum koji može nastati tijekom snimanja slike. Pomaže modelu da dobro generalizira objekte bez obzira na prisutan šum
- GaussianBlur(*probability*)
  - Primjenjuje Gaussovo zamućenje na sliku čime se smanjuju detalji i šum, rezultirajući glatkom slikom
- Sharpen(*probability*)
  - Povećava kontrast rubova na slici, čineći ih jasnijima i bolje definiranim. Ova tehnika povećava sposobnost modela da prepozna detalje u slikama
- RandomGamma(*probability*)
  - Mijenja svjetlinu slike tako da generira slučajnu gama vrijednost koja će posvijetliti sliku ako je pozitivna ili potamniti sliku ako je negativna
- CLAHE(*probability*)
  - Primjenjuje CLAHE (*Contrast Limited Adaptive Histogram Equalization*) na sliku. CLAHE ograničava kontrast, što sprječava prekomjerno pojačanje šuma u homogenim područjima

3. Implementacija programskog rješenja

U ovom će poglavlju biti objašnjeno kako je model za detekciju prometnih znakova i semafora implementiran, kako koristiti aplikaciju i bit će objašnjena biblioteka *Ultralytics* koja omogućuje korištenje *YOLO* modela dubokog učenja unutar Python koda.

3.1. Ultralytics

Ultralytics je kompanija poznata po razvoju alata za duboko učenje, posebno u području računalnog vida [18]. Njihova glavna biblioteka, *Ultralytics* je skup naprednih alata koji omogućavaju treniranje, korištenje i pregled različitih modela unutar Python koda. Ultralytics je razvila različite verzije YOLO modela, s ciljem poboljšanja performansi u detekciji objekata. Najnovije verzije, poput YOLOv8 i YOLOv9, donose značajna poboljšanja u brzini, točnosti i jednostavnosti korištenja [24].

Ključne značajke ove biblioteke su:

- Jednostavnost korištenja
  - Ultralytics ima jednostavan API koji omogućava brzo pokretanje i implementaciju modela
  - Podrška za Jupyter bilježnice omogućava korisnicima lako eksperimentiranje i razvoj unutar interaktivnog okruženja
- Performanse
  - YOLO modeli su poznati po svojoj brzini, omogućavajući detekciju objekata u stvarnom vremenu, što ih čini idealni za aplikacije poput autonomnih vozila i video nadzora
  - Modeli poput YOLOv8 koriste napredne tehnike za poboljšanje točnosti detekcije, uključujući dinamičko dodjeljivanje sidra i optimizirane funkcije gubitka
- Fleksibilnost i prilagodljivost
  - Korisnici mogu lako prilagoditi modele svojim specifičnim potrebama, koristeći unaprijed trenirane modele kao osnovu ili trenirajući nove modele od početka
  - Modeli mogu raditi s različitim veličinama ulaznih slika, omogućavajući veću fleksibilnost u različitim aplikacijama

- Biblioteka podržava veliki broj hiperparametara koji se mogu predati modelu pri treniranju i znatno poboljšati performanse tog modela
- Podrška za različite formate modela dubokog učenja
  - Glavni format za Ultralytics modele je PyTorch, koji je poznat po svojoj fleksibilnosti i dinamičnom grafu izvođenja
  - Modeli se mogu dobiti u različitim formatima, uključujući ONNX, CoreML, TensorFlow Lite, što omogućava njihovu implementaciju na različitim platformama i uređajima
- Vizualizacija i analitika
  - Biblioteka nudi alate za vizualizaciju i analizu performansi modela, uključujući grafike točnosti, gubitka i drugih ključnih metrika
  - Sadrži ugrađene alate za prikazivanje rezultata detekcije na slikama i video zapisima, što olakšava evaluaciju i prezentaciju rezultata

3.1.1. Kako koristiti Ultralytics YOLO?

Prvi korak je instalacija biblioteke ultralytics koja se može napraviti s pomoću programa pip:

```
pip install ultralytics
```

Jednom kada smo instalirali biblioteku, ona je spremna za korištenje i možemo je uvesti u Python kod s pomoću importa. Jednom kada je uvezemo, možemo definirati model, koristiti taj model za detekciju i puno drugih mogućnosti:

```
from ultralytics import YOLO
model = YOLO('yolov8n.pt')
results = model('path/to/image.jpg')
results.show()
```

Kod 3.1 Uvoz biblioteke unutar Python koda

Model možemo po želji i trenirati:

```
model = YOLO('yolov8n.pt')
model.train(dana='path/to/dataset.yaml', epochs=100)
```

Kod 3.2 Treniranje YOLO modela

Kod treniranja modela, možemo mu zadati veliki broj hiperparametara od kojih su neki veoma bitni poput broja epoha, veličine slike i parametri brzine učenja. Tablica s nekima od svih hiperparametara nalazi se dolje i prikazuje imena parametra, njegovu

pretpostavljenu vrijednost, ako se ne preda neka druga vrijednost i objašnjenje tog parametra (Tablica 3.2).

Tablica 3.1 Hiperparametri kod treniranja modela

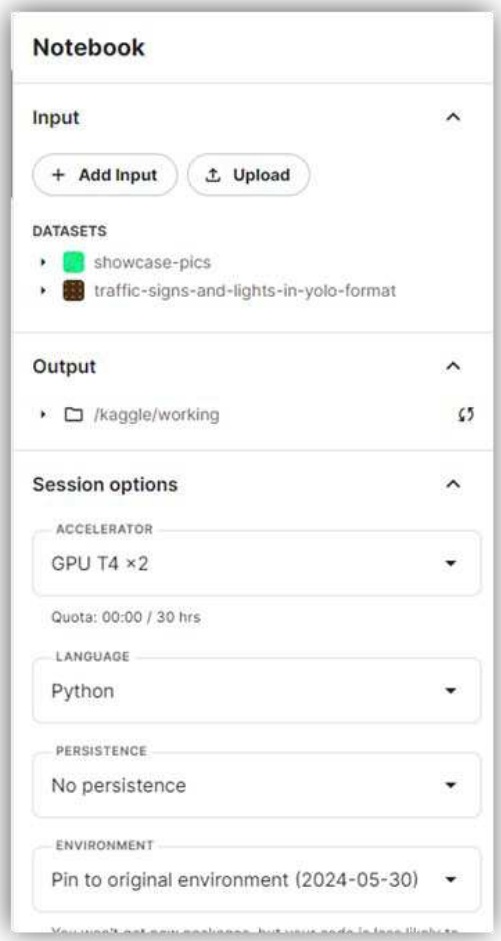
Ime hiperparametra	Pretpostavljena vrijednost	Opis
model	None	Specificira model koji želimo trenirati
data	None	Putanja do konfiguracijske datoteke
epochs	100	Koliko puta će model proći sve slike za vrijeme treniranja
imgsz	640	Sve slike se skaliraju na ovu dimenziju prije samog treniranja
optimizer	'auto'	Program koji nastoji poboljšati parametre učenja
lr0	0.01	Početna brzina učenja
lrf	0.01	Konačna brzina učenja

3.2. Treniranje modela

Kako bi mogli koristiti naš model u Python aplikaciji prvo ga moramo trenirati na napravljenom skupu podataka. Treniranje se, kao što je navedeno ranije u dokumentu izvodi na web platformi Kaggle.

Za treniranje je potreban skup slika koje ćemo koristiti, stoga možemo izraditi novi „Dataset“ i prebaciti sve potrebne slike na Kaggle u novostvoreni skup podataka. Stvoreni „Dataset“ zatim stavljamo u input prostor „Jupyter bilježnice“ koju ćemo koristiti za

treniranje. Također, odmah na početku možemo namjestiti da se koristi grafička kartica koju nudi Kaggle.



Slika 3.1 Postavke JuPyter bilježnice

Nakon što je okruženje namješteno, preostaje napisati kod kojim ćemo odlučiti kako želimo trenirati naš model. Početak koda sastoji se od funkcije za augmentaciju podataka i funkcije za raspodjelu slika na skup za treniranje, skup za validaciju i skup za testiranje. Također, ta funkcija te skupove sprema u radni direktorij naše bilježnice kako bi model za treniranje kasnije mogao lagano dohvatiti te podatke. Zbog veličine funkcije za raspodjelu slika po skupovima, ovdje ćemo je izostaviti.

```
def augment_image(image_path):
    image = cv2.imread(image_path)
    transform = A.Compose([
        A.RandomBrightnessContrast(p=0.2),
        A.GaussNoise(p=0.2),
        A.GaussianBlur(p=0.2),
        A.Sharpen(p=0.2),
        A.RandomGamma(p=0.2),
        A.CLAHE(p=0.2)
    ])
    augmented = transform(image=image)
    return augmented['image']
```

Kod 3.3 Funkcija za augmentaciju slika

Nakon što su svi podaci za treniranje i testiranje pripremljeni, model se može krenuti trenirati. Kao odabir modela koji ćemo trenirati u ovom projektu odabran je već trenirani model **yolov8x.pt**. Također, definira se konfiguracijska datoteka s pomoću koje se modelu daje do znanja gdje se nalaze setovi slika i koji broj predstavlja koju klasu unutar oznaka slika.

```
from ultralytics import YOLO

os.environ['WANDB_DISABLED'] = 'true'
model = YOLO("yolov8x.pt")

config_content = """path: /kaggle/working/datasets
train: images/train
val: images/valid
test: images/test

# Classes
names:
  0: traffic_sign
  1: traffic_light
"""
with open("/kaggle/working/data_config.yaml", "w") as f:
    f.write(config_content)

model.train(data="/kaggle/working/data_config.yaml",
epochs=200)
metrics = model.val()
path = model.export(format="onnx")
```

Kod 3.4 Konfiguracijska datoteka i treniranje modela

Nakon toga sve je spremno za trening modela koje ćemo započeti tako da spremimo verziju bilježnice i predamo joj željeno ime. U sklopu ovog projekta dodane su još određene funkcije za predikciju slika unutar iste bilježnice, ali one se neće koristiti kasnije. Još jedna stvar, koja je korisna i koja je dodana prije pokretanja cijelog koda je funkcija za ispis metrika na skupu za treniranje i skupu za testiranje. S pomoću tih metrika možemo vidjeti koliko dobro je model istreniran.

```
metrics = model.val()
print(metrics.box.map)
print(metrics.box.map50)

metrics = model.val(split='test')
print(metrics.box.map)
print(metrics.box.map50)
```

Kod 3.5 Funkcije za ispis metrika

Rezultati treniranja bit će komentirani u kasnijem dijelu dokumenta, dok ćemo istrenirani model preuzeti iz output prostora naše bilježnice i koristit ga dalje za izradu aplikacije za detekciju semafora i prometnih znakova.

3.3. Aplikacija

Aplikacija je jednostavna Python aplikacija koja se pokreće iz terminala odnosno komandne linije. Ako je grafička kartica koju sadrži računalo na kojem se pokreće aplikacija, preporučuje se koristiti virtualnu grafičku karticu zbog brzine obrade. Unutar aplikacije se koriste sljedeće biblioteke koje nisu dio Pythonove standardne biblioteke:

- *ultralytics*
- *matplotlib*
- *torch*
- *torchvision*
- *onnxruntime*

Navedene biblioteke se preuzimaju na način prikazan na slici broj 7.

```
!pip install ultralytics
!pip install matplotlib
!pip install torch
!pip install torchvision
!pip install onnxruntime
```

Kod 3.6 Instalacija potrebnih paketa

Dvije biblioteke koje su dio standardne Pythonove biblioteke su *cv2* i *sys* koje će nam koristiti za baratanje slikama i datotekama koje su na Kaggleu. Aplikacija sadrži klasu Model koja prima putanju do željenog istreniranog modela i vraća instancu klase koja je u pozadini YOLO model.

```
class Model:
    IMAGE = 'IMAGE'
    VIDEO = 'VIDEO'

    def __init__(self, model_path):
        self.model = YOLO(model_path)
```

Kod 3.7 Inicijalizacija klase Model

Nakon što je instanca klase Model dobivena putem poziva u glavnom dijelu programa, možemo koristiti metodu te klase „predict_media“ koja služi za generiranje slika odnosno videa s detektiranim prometnim znakovima i semaforima. Metoda kao parametre prima sadržaj koji se predaje (ili „image“ ili „video“), putanju do sadržaja i kao zadnji argument prima putanju do mjesta gdje želimo spremiti obrađenu sliku ili video.

```
def predict_media(self, media, media_path, output_path):
        if media.upper() == Model.VIDEO:
            self.predict_video(media_path, output_path)
        elif media.upper() == Model.IMAGE:
            self.predict_image(media_path)
        else:
            print("Wrong media argument: <image|video>")
            sys.exit(1)
```

Kod 3.8 Funkcija predict_media

Funkcija „predict_media“ zove funkcije „predict_video“ odnosno „predict_image“, ovisno o vrsti medijskog sadržaja koji je predan.

```
def predict_image(self, image_path):
        image = cv2.imread(image_path)

        results = self.model.predict(image, show=False)

        plt.figure(figsize=(10, 10))
        plt.imshow(cv2.cvtColor(results[0].plot(),
cv2.COLOR_BGR2RGB))
        plt.axis('off')
        plt.show()
```

Kod 3.9 Funkcija predict_image

```
def predict_video(self, video_path, output_path):
        video = cv2.VideoCapture(video_path)

        if not video.isOpened():
            print("Error: Could not open video.")
        else:
            fourcc = cv2.VideoWriter_fourcc(*'mp4v')
            out = cv2.VideoWriter(output_path, fourcc, 20.0,
(int(video.get(3)), int(video.get(4))))

            while video.isOpened():
                ret, frame = video.read()
                if ret:
                    results =
self.model.predict(source=frame, show=False)

                    processed_frame = results[0].plot()
                    processed_frame =
cv2.cvtColor(processed_frame, cv2.COLOR_RGB2BGR)
                    out.write(processed_frame)
                else:
                    break

            video.release()
            out.release()
```

Kod 3.10 Funkcija predict_video

Treba naglasiti da se video sprema na mjesto zadano s „output_path“ parametrom dok se slika zbog jednostavnosti odmah prikazuje korisniku bez spremanja na računalo.

3.4. Upute za pokretanje

Aplikacija se skida s repozitorija na GitHubu [19]. Može se preuzeti u obliku zip datoteke ili se može klonirati repozitorij. Nakon toga se treba pozicionirati u direktorij „application“ gdje se nalazi aplikacija. Prvo treba preuzeti sve potrebne biblioteke koje nisu dio Pythonove standardne biblioteke. To se može napraviti sljedećom naredbom:

```
pip install -r requirements.txt
```

Naravno, uz gore navedeni zahtjev, pretpostavka je da računalo na kojem se pokreće aplikacija ima instaliran Python inače se kod neće moći pokrenuti. Nakon što su sve dodatne biblioteke preuzete, aplikacija je spremna za upotrebu. Potrebno je otvoriti terminal ili komandnu liniju i pozicionirati se u direktorij s aplikacijom, ako se već ne nalazimo u njemu. Nakon toga zadaju se parametri sljedećim redom:

```
python model.py <model_path> <image|video> <image_path|video_path> <output_path>
```

- model_path
  - putanja do modela kojeg želimo koristiti
- image|video
  - vrsta medija kojeg želimo procesirati; mora biti točno ili „image“ ili „video“
- image_path|video_path
  - putanja do medija kojeg želimo procesirati
- output_path
  - putanja do mjesta na računalu gdje želimo spremiti video

Primjer poziva aplikacije i rezultat:

Slika 3.2 Primjer poziva aplikacije i rezultat

4. Rezultati

U ovom će poglavlju bit objašnjene metrike korištene za provjeru kvalitete modela i na kraju će biti uspoređene sa drugim radovima na istu temu te prokomentirane.

4.1. Korištene metrike evaluacije detektora objekata

Strojno učenje ima veliki broj metrika koje se mogu koristiti za opis preciznosti i učinkovitosti modela. YOLO modeli nakon treniranja generiraju grafove i metrike koje su statistički obrađene. Metrike evaluacije detektora objekata na koje ćemo se fokusirati su [20]:

- *box_loss*
  - Gubitak kutije mjeri pogrešku između predviđenog ograničavajućeg okvira i stvarnog ograničavajućeg okvira. Obuhvaća pogreške u lokalizaciji, poput netočne veličine ili pozicije predviđenog okvira.
  - Ova funkcija će se koristiti i na setu podataka za treniranje i na setu podataka za validaciju
  - Za model YOLOv8, nije objavljen točan algoritam koji se koristi za ovu metriku, ali većina modela dubokog učenja koriste L1 ili L2 gubitak ili IoU (*Intersection over Unit*)
  - Matematika iza ovih metrika je sljedeća:

$$L1LossFunction = \sum_{i=1}^{n} \left| y_{true} - y_{predicted} \right| \tag{1}$$

$$L2LossFunction = \sum_{i=1}^{n} \left(y_{true} - y_{predicted} \right)^2 \tag{2}$$

  - *Y* predstavlja dobivenu, odnosno predviđenu vrijednost za određeni objekt. Ove formule također objašnjavaju zašto *box_loss* parametar može biti veći 1, što možda na prvu, nije za očekivati.

- *cls_loss*
  - Ova metrika mjeri pogrešku u predviđanju klasnih oznaka detektiranih objekata. Procjenjuje koliko dobro model razlikuje različite klas objekata.

- Ovaj gubitak se obično izračunava korištenjem *cross-entropy* gubitka, koji se bavi neravnotežom klasa fokusirajući se više na teško klasificirajuće primjere. Manji gubitak klasifikacije ukazuje na bolju točnost u klasificiranju objekata.
- Matematička formula za *cross-entropy* gubitak je:

$$H(p,q) = -\sum_{\forall x} p(x)\log\big(q(x)\big) \tag{3}$$

- U ovoj je matematičkoj funkciji *p(x)* prava distribucija, dok je *q(x)* predviđena distribucija s obzirom na podatke unutar modela. Ova funkcija, s obzirom na to da je suma, također pokazuje da vrijednost može biti i veća od 1.
- Ova metrika će se također koristiti i na validacijskom skupu i na skupu za treniranje

- *dfl_loss*
  - *Distribution Focal Loss* (DFL) je metrika korištena u naprednim modelima detekcije objekata poput YOLOv8 za poboljšanje točnosti lokalizacije zadanih okvira
  - Matematička formula na DFL se razlikuje od modela do modela, ali nekakav glavni oblik je:

$$DFL(y,\hat{y}) = -\alpha\big(1-\sigma(\hat{y})\big)^{\gamma}\log\big(\sigma(y)\big) \tag{4}$$

  - α i γ su hiperparametri koji se određuju unutar modela, a određuju koliko će svaki primjer pridonositi pogrešci
  - σ(y) je softmax funkcija koja pretvara stvarnu koordinatu y u distribuciju vjerojatnosti. Ovo omogućuje modelu da nauči nesigurnost i fokusira se na vjerojatnija područja.
  - $(1-\sigma(\hat{y}))^2$ je izraz koji smanjuje utjecaj lako klasificiranih primjera i povećava utjecaj teških, krivo klasificiranih primjera. To znači da će primjeri koji su modelu teški (oni koje model klasificira s niskom sigurnošću) imati veći utjecaj na ukupni gubitak, što pomaže modelu da se bolje nauči na tim primjerima

- *precision*(B)
  - Preciznost je omjer točnih pozitivnih detekcija u odnosu na ukupan broj pozitivnih predikcija (točni pozitivni + lažni pozitivni)

- Visoka preciznost znači da model predviđa objekt većinom točno. Preciznost je ključna u slučajevima gdje želimo minimizirati broj lažnih pozitivnih rezultata
- Matematička formula za preciznost je:

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

- *recall*
  - Opoziv (*recall*) je omjer točnih pozitivnih detekcija u odnosu na ukupan broj stvarnih pozitivnih (točni pozitivni + lažni negativni). Mjeri sposobnost modela da identificira sve relevantne objekte u skupu podataka.
  - Ako je ova metrika visoka (blizu 1), model je dobar u pronalaženju svih instanci objekta
  - Matematička formula za *recall* je:

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

- *mAP50*(B)
  - *Mean Average Precision* (mAP) na *Intersection over Union* (IoU) 0.5 mjeri prosječnu preciznost modela preko svih klasa, koristeći prag od 0.5 za IoU između predviđenih i stvarnih okvira
  - Viša vrijednost ukazuje na bolju detekciju od strane modela
  - IoU je metrički pokazatelj koji se koristi za evaluaciju točnosti predviđenih okvira u odnosu na stvarne okvire. Izračunava se kao omjer područja preklapanja između predviđenog okvira i stvarnog okvira prema ukupnom području koje obuhvaća oba okvira.

$$IoU = \frac{Područje\ Preklapanja}{Područje\ Unije} \tag{7}$$

  - *Područje preklapanja* je površina gdje se predviđeni okvir i stvarni okvir preklapaju
  - *Područje Unije* je ukupna površina koja obuhvaća oba okvira

    Dakle, formula za ovu metriku može se iskazati kao:

$$mAP = \frac{1}{K}\sum_{k=1}^{K} AP_k \tag{8}$$

  - Konačno, ako je IoU između predviđenog i stvarnog okvira veći ili jednak 0.5, rezultat se smatra točnim

- *mAP50-95*(B)

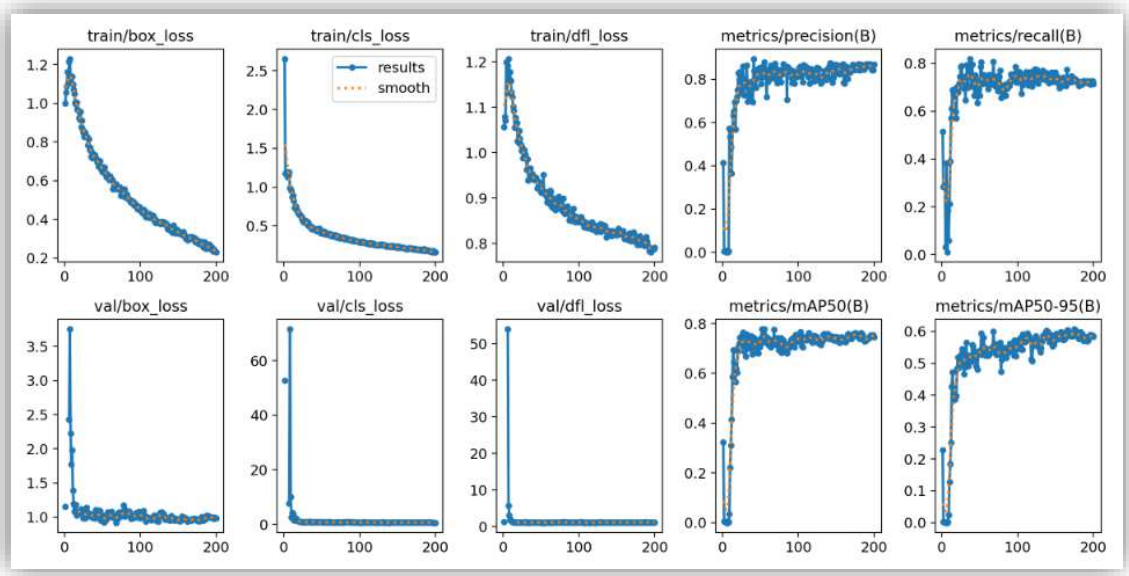
- Ovaj metrički parametar je jako sličan prethodnom *mAP50*, ali je stroži jer uzima u obzir više IoU pragova, pružajući detaljniji pregled sposobnosti modela da predvidi točne ograničavajuće okvire.
- *mAP50-95* mjeri prosječnu preciznost na više IoU pragova (korak može biti za 0.05 ili više). Dakle AP se izračunava za svaki IoU prag, a zatim se prosječna vrijednost koristi za izračun *mAP50-95*

$$mAP50 - 95 = \frac{1}{K}\sum_{k=0.}^{K} AP_k \tag{9}$$

4.2. Rezultati dobivenog modela

Prvo ćemo prikazati grafove koji prikazuju vrijednosti metrika kroz različite epohe. Zatim ćemo prikazati konfuzijsku matricu i na kraju ćemo prikazati brojčano metrike koje na kraju ima naš model. Svaki od ovih dijelova bit će i komentiran i objašnjen.

Grafički prikaz vrijednosti metrika kroz epohe:



Slika 4.1 Gtafički prikaz metrika kroz epohe

Iz grafa *train/box_loss* možemo vidjeti da gubitak značajno opada s povećanjem broja epoha, što ukazuje na to da model uči bolje predviđati zadane okvire tijekom vremena.

Val/box_loss pokazuje da je gubitak također u padu s porastom broja epoha, ali vidljivi su određeni skokovi. To može biti pokazatelj na nejednolike performanse na validacijskom

skupu podataka, a razlog tome može biti nedovoljno velik broj slika za učenje ili loše slike za validaciju. Isto tako, gubitci na validacijskom setu nikad ne prolaze ispod 1, dok su na setu za treniranje na 0.2 na kraju.

Slično je i kod *train/cls_loss* i *val/cls_loss* gdje također imamo pad gubitaka s porastom epoha. Treba napomenuti da *val/cls_loss* također ima skokove i nejednolikost pri zadnjim epohama, ali ih je teško uočiti zbog raspodjele na y osi.

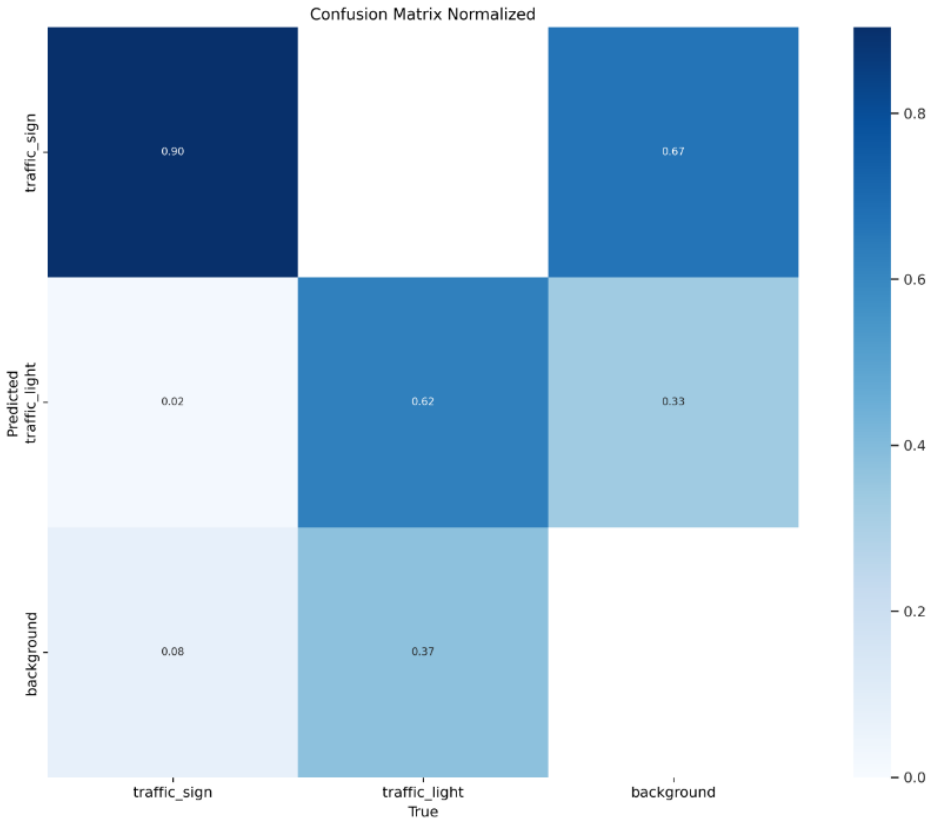
Train/dfl_loss i *val/dfl_loss* također pokazuju jako puno sličnosti prethodnim opisanim grafovima, ali uz više nejednolikosti među epohama.

Metrics/precision(B) prikazuje preciznost modela tijekom vremena. Preciznost za naš model brzo raste i stabilizira se na visokoj razini, što ukazuje na to da model ima visoku točnost u predikcijama.

Metrics/recall(B) pokazuje opoziv (*recall*) modela tijekom vremena. Slično kao i kod preciznosti, opoziv brzo raste i stabilizira se pri vrijednosti od 0.8 iz čega možemo zaključiti da model dobro identificira većinu objekata.

Metrics/mAP50(B) graf pokazuje da doseže preciznost od 0.8 što također ukazuje na visoku ukupnu preciznost modela. *Metrics/mAP50-95(B)* isto brzo raste, ali se stabilizira na oko 0.5 što je očekivano jer uključuje strože IoU pragove.

Sljedeće ćemo pokazati konfuzijsku matricu:



Slika 4.2: Konfuzijska matrica

Konfuzijska matrica pokazuje ipak drukčiju situaciju nego preciznost. Model u 90 % slučajeva pretpostavlja da je detektirani objekt prometni znak kada detektirani objekt zaista je prometni znak. Semafor detektira u samo 62 % slučajeva kada se radi o semaforu, dok ga puno puta, u čak 37 % slučajeva, zamijeni s pozadinom. Dok je ukupna preciznost bila visoka za naš model, konfuzijska matrica otkriva da model jako dobro prepoznaje prometne znakove, zna raspoznavati semafore od prometnih znakova, ali semafore zna često zamijeniti za pozadinu. Prema konfuzijskoj matrici, pozadina stvara problem.

Za kraj ćemo još prikazati metrike brojčano:

```
Run summary:
                    lr/pg0 2e-05
                    lr/pg1 2e-05
                    lr/pg2 2e-05
         metrics/mAP50(B) 0.76326
      metrics/mAP50-95(B) 0.61148
     metrics/precision(B) 0.84737
        metrics/recall(B) 0.73713
             model/GFLOPs 258.128
         model/parameters 68154534
  model/speed_PyTorch(ms) 34.766
           train/box_loss 0.23069
           train/cls_loss 0.16044
           train/dfl_loss 0.79058
             val/box_loss 0.98603
             val/cls_loss 0.58439
             val/dfl_loss 1.19049
```

Slika 4.3: Brojčani opis metrika

Lr/pg0 predstavlja stopu učenja u početnih slojevima, *lr/pg1* stopu učenja u srednjim slojevima neuronske mreže i *lrpg2* u završnim slojevima mreže [21]. Stope učenja su u našem slučaju kroz cijelu neuronsku mrežu jednake i iznose 0.00002. Još jednom vidimo metrike *box_loss*, *cls_loss*, *dfl_loss*, *precission, recall* i *mAP50, mAP50-90* kojima smo vrijednosti komentirali u poglavlju prije. Ovdje imamo nekoliko novih pojmova, a to su *model/parameters* koji predstavlja ukupan broj parametara u modelu, *model/speed_PyTorch(ms)* što označava brzinu izvođenja modela u milisekundama koristeći PyTorch. *Model/GFLOPs* (*Giga Floating Point Operations per Second*) mjera je računalne složenosti modela i veći broj označava veći broj operacija i složeniji model.

4.3. Završna usporedba s drugim radom

Zadnja stvar koja je preostala za napraviti je usporediti naše rezultate s rezultatima drugih istraživača na istu temu. Za našu usporedbu koristit ćemo također Kaggle Notebook kojeg je napravio drugi korisnik [22], a u kojem je korišten isti model dubokog učenja. Poveznica na rad bit će ostavljena pod literaturom.

Run summary:

lr/pg0	2e-05
lr/pg1	2e-05
lr/pg2	2e-05
metrics/mAP50(B)	0.93608
metrics/mAP50-95(B)	0.82598
metrics/precision(B)	0.9493
metrics/recall(B)	0.88844
model/GFLOPs	8.209
model/parameters	3013773
model/speed_PyTorch(ms)	1.448
train/box_loss	0.42934
train/cls_loss	0.25742
train/dfl_loss	0.8378
val/box_loss	0.49943
val/cls_loss	0.28454
val/dfl_loss	0.86544

Slika 4: Rezultati drugog rada

Može se uočiti da ovaj rad ima bolje konačne metrike od ovog projekta, iako u nekim metrikama poput *cls_loss*, *box_loss* i *dfl_loss* su slični na skupu za treniranje, ali na validacijskom skupu, koji je bitniji su puno bolje. Može se zaključiti da ovaj model bolje generalizira problem u odnosu na model ovog projekta. S obzirom na to da su korišteni slični parametri i hiperparametri, razlog zbog kojeg je došlo do ovakvih razlika je u skupu podataka. Iz broja serija, kojih je bilo 17 po epohi u našem projektu, a 56 u projektu drugog korisnika, možemo izvesti zaključak da se u ovom projektu koristio puno manji skup podataka i slika. To je vrlo vjerojatno i glavni razlog lošijih metrika.

Zaključak

Modeli za detekciju prometne signalizacije [23] već od prije imaju široku primjenu i upravo je to bio jedan od razloga za ovaj projekt u kojem je primijenjeno znanje umjetne inteligencije, dubokog učenja, statistike i programiranja. Rezultat je očit, i on je da ovakvi modeli imaju jako pozitivan utjecaj na današnji svijet i mijenjaju industriju iz dana u dan. Iako nije potvrđeno, vjerujem da ovakvi sustavi smanjuju broj nesreća na cesti i čine promet sigurnijim. Računalni vid i duboko učenje se razvijaju sve brže i brže i vrlo je vjerojatno kako ćemo jednog dana imati modele koji će imati točnosti blizu 100% s gubitcima gotovo 0. S porastom primjene umjetne inteligencije u prometnim sustavima, važno je adresirati i etničke aspekte. Potrebno je osigurati da tehnologija bude korištena na etičan način i u skladu s pravnim okvirima kako bi se zaštitili svi sudionici u prometu.

Literatura

[1] Planck Collaboration. (2015). Planck 2015 results: XIII. Cosmological parameters. *Nature*, 519(7543), 307-309. Poveznica: https://www.nature.com/articles/nature14539 (pristupljeno 12. 6. 2024.)

[2] IBM. (2024). What is deep learning? Poveznica: https://www.ibm.com/cloud/learn/deep-learning (pristupljeno 12. 6. 2024.)

[3] ENIAC, Wikipedia Poveznica: https://hr.wikipedia.org/wiki/ENIAC (pristupljeno 5. 6. 2024.)

[4] Perceptron Poveznica: https://machinelearningmastery.com/implement-perceptron-algorithm-scratch-python/ (pristupljeno 11. 6. 2024.)

[5] IBM. (2024). What is computer vision? Poveznica: https://www.ibm.com/blog/computer-vision/ (pristupljeno 12. 6. 2024.)

[6] Li, Y., Chen, Y., Wang, N., & Zhang, Z. (2023). Multi-Scale Cost Attention and Adaptive Fusion Stereo Matching Network. *Electronics, 12*(7), 1594. Poveznica: https://www.mdpi.com/2079-9292/12/7/1594 (pristupljeno 12. 6. 2024.)

[7] Autonomna vozila, Wikipedia Poveznica: https://en.wikipedia.org/wiki/Self-driving_car (pristupljeno 9. 6. 2024.)

[8] Built In. (2024). What Is Kaggle? How to Compete in Kaggle Competitions. Poveznica: https://builtin.com/data-science/what-is-kaggle (pristupljeno 5. 6. 2024.)

[9] Albumentation library i granični okviri Poveznica: https://albumentations.ai/docs/getting_started/bounding_boxes_augmentation/ (pristupljeno 9. 6. 2024.)

[10] Ultralytics YOLO dokumentacija, Poveznica: https://docs.ultralytics.com/ (pristupljeno 5. 6. 2024.)

[11] Hussain, M. (2023). YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection. *Machines*, 11(7), 677. Poveznica: https://www.mdpi.com/2075-1702/11/7/677 (pristupljeno 12. 6. 2024.)

[12] Neural Magic. (2024). YOLOv8 Detection 10x Faster with DeepSparse: 500 FPS on a CPU. Poveznica: https://neuralmagic.com/blog/yolov8-detection-10x-faster-with-deepsparse-500-fps-on-a-cpu (pristupljeno 12. 6. 2024.)

[13] YOLOv8 arhitektura Poveznica: https://yolov8.org/yolov8-architecture/ (pristupljeno 8. 6. 2024.)

[14] Juan Pedro. (2023). Detailed Explanation of YOLOv8 Architecture: Part 1. *Medium*. Poveznica: https://medium.com/@juanpedro.bc22/detailed-explanation-of-yolov8-architecture-part-1-6da9296b954e (pristupljeno 12. 6. 2024.)

[15] Li, Y., Chen, Y., Wang, N., & Zhang, Z. (2019). Dynamic Anchor Feature Selection for Single-Shot Object Detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Poveznica: https://openaccess.thecvf.com/content_ICCV_2019/papers/Li_Dynamic_Anchor_Feature_Selection_for_Single-Shot_Object_Detection_ICCV_2019_paper.pdf (pristupljeno 9. 6. 2024.)

[16] DevOpsSchool. (2023). What is Kaggle and Use Cases of Kaggle? Poveznica: https://www.devopsschool.com/blog/what-is-kaggle-and-use-cases-of-kaggle/ (pristupljeno 11. 6. 2024.)

[17] ProfileTree. (2024). Python Programming Language: A Comprehensive Overview. Poveznica: https://profiletree.com/python-programming-language-comprehensive-overview/ (pristupljeno 8. 6. 2024.)

[18] Ultralytics github dokumentacija Poveznica: https://github.com/ultralytics/docs (pristupljeno 9. 6. 2024.)

[19] Repozitorij aplikacije i treniranja modela Poveznica: https://github.com/JureJurko/Zavrsni-rad (10. 6. 2024.)

[20] Wang, Z., Li, Y., & Zhang, Z. (2022). Sustainability in the Digital Age: A Comprehensive Review and Future Directions. *Sustainability, 14*(19), 12274. Poveznica: https://www.mdpi.com/2071-1050/14/19/12274 (pristupljeno 4. 6. 2024.)

[21] Parametri učenja modela YOLOv8 Poveznica: https://github.com/ultralytics/ultralytics/issues/7424 (pristupljeno 6. 6. 2024.)

[22] Rad za usporedbu Poveznica: https://www.kaggle.com/code/pkdarabi/traffic-signs-detection-using-yolov8 (pristupljeno 6.6.2024.)

[23] Definicija prometne signalizacije Poveznica: https://enciklopedija.hr/clanak/prometna-signalizacija (pristupljeno 11.6.2024.)

[24] Horvat, M., Jelečević, L., & Gledec, G. (2023, September). Comparative Analysis of YOLOv5 and YOLOv6 Models Performance for Object Classification on Open Infrastructure: Insights and Recommendations. In *Proceedings of the 34th Central European Conference on Information and Intelligent Systems (CECIIS 2023), Dubrovnik, Croatia* (pp. 20-22)

[25] Horvat, M., & Gledec, G. (2022, September). A comparative study of YOLOv5 models performance for image localization and classification. In *33rd Central European Conference on Information and Intelligent Systems (CECIIS)* (p. 349)

Sažetak

RAZVOJ APLIKACIJE ZA DETEKCIJU I RASPOZNAVANJE PROMETNE SIGNALIZACIJE IZ VIDEO SIGNALA KORIŠTENJEM MODELA DUBOKOG UČENJA

U ovom radu korišteni su modeli dubokog učenja, konkretno YOLOv8, za detekciju prometne signalizacije. Primijenjeno je znanje iz umjetne inteligencije, dubokog učenja i računalnog vida. Model je treniran na skupu slika hrvatskih prometnih znakova koristeći platformu Kaggle, koja omogućuje upotrebu naprednih grafičkih kartica za brže treniranje. Evaluirane su različite metrike evaluacije detektora objekata, uključujući *box_loss*, *cls_loss*, *dfl_loss*, preciznost i opoziv. Rezultati su pokazali da model postiže visoku točnost, iako postoje izazovi u detekciji semafora. Dobiveni model je implementiran u aplikaciju koja prima slike i video signale i detektira objekte u njima. Projekt pokazuje potencijal za poboljšanje sigurnosti prometa kroz napredne tehnologije.

Ključne riječi

Duboko učenje, YOLOv8, prometna signalizacija, detekcija objekata, Kaggle, metrike evaluacije detektora objekata

Summary

DEVELOPMENT OF AN APPLICATION FOR TRAFFIC SIGN DETECTION AND RECOGNITION FROM VIDEO SIGNALS USING DEEP LEARNING MODELS

In this study, deep learning models, specifically YOLOv8, were used for the detection of traffic signals. Knowledge from artificial intelligence, deep learning, and computer vision was applied. The model was trained on a dataset of Croatian traffic sign images using the Kaggle platform, which enables the use of advanced graphics cards for faster training. Various object detection evaluation metrics were evaluated, including box_loss, cls_loss, dfl_loss, precision, and recall. The results showed that the model achieves high accuracy, although there are challenges in detecting traffic lights. The obtained model was implemented in an application that receives images and video signals and detects objects in them. The project demonstrates the potential for improving traffic safety through advanced technologies.

Keywords

deep learning, YOLOv8, traffic signals, object detection, Kaggle, object detection evaluation metrics