Primjena dubokog učenja u razvoju sustava za detekciju objekata u stvarnom vremenu

Nenadić, Lovre

Sveučilište u Zadru

Odjel za informacijske znanosti

Stručni prijediplomski studij

Informacijske tehnologije

Lovre Nenadić

Primjena dubokog učenja u razvoju sustava za detekciju objekata u stvarnom vremenu

Završni rad

Zadar, 2024.

Sveučilište u Zadru

Odjel za informacijske znanosti

Stručni prijediplomski studij

Informacijske tehnologije

Primjena dubokog učenja u razvoju sustava za detekciju objekata u stvarnom vremenu

Završni rad

Student/ica:

Lovre Nenadić

Mentor/ica:

Doc. dr. sc. Ante Panjkota

Komentor/ica:

Niko Vrdoljak mag. ing. el.

Zadar, 2024.

Izjava o akademskoj čestitosti

Ja, **Lovre Nenadić,** ovime izjavljujem da je moj **završni** rad pod naslovom **Primjena dubokog učenja u razvoju sustava za detekciju objekata u stvarnom vremenu** rezultat mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na izvore i radove navedene u bilješkama i popisu literature. Ni jedan dio mojega rada nije napisan na nedopušten način, odnosno nije prepisan iz necitiranih radova i ne krši bilo čija autorska prava.

Izjavljujem da ni jedan dio ovoga rada nije iskorišten u kojem drugom radu pri bilo kojoj drugoj visokoškolskoj, znanstvenoj, obrazovnoj ili inoj ustanovi.

Sadržaj mojega rada u potpunosti odgovara sadržaju obranjenoga i nakon obrane uređenoga rada.

Zadar, 28. rujna 2024.

Primjena dubokog učenja u razvoju sustava za detekciju objekata u stvarnom vremenu

SAŽETAK

U radu je prikazana primjena dubokog učenja u stvarnom vremenu, uključujući sve potrebne korake koji su doveli do izrade web aplikacije. Cilj rada je bio istrenirati model za detekciju ručnog sata i pametnog telefona, te ga implementirati u web aplikaciju.

Rad je podijeljen u tri dijela, gdje se svaki nadovezuje na prethodni. U prvom dijelu teorijski je opisan YOLOv8 model, te način na koji su fotografije pripremljene za detekciju i segmentaciju objekata. U drugom dijelu objašnjen je proces treniranja modela na Google Colab platformi, uz analizu dobivenih rezultata. Na kraju rada predstavljen je Streamlit okvir i način na koji je implementiran u web aplikaciju. Također, opisano je korisničko sučelje aplikacije te rezultati detekcije i segmentacije objekata koji se prikazuju u stvarnom vremenu.

Ključne riječi: duboko učenje, YOLOv8, detekcija, segmentacija, Google Colab, Streamlit

Using deep learning in the development of object detection systems in real time

SUMMARY

This thesis presents the application of deep learning in real-time, including all the necessary steps that led to the development of a web application. The aim of the thesis was to train a data model for detecting wristwatches and smartphones and implement it into a web application.

The thesis is divided into three parts, where each builds upon the previous one. The first part theoretically describes the YOLOv8 model and the process by which the images were prepared for object detection and segmentation. The second part explains the process of training the data model on the Google Colab platform, along with an analysis of the achieved results. The final part of the thesis presents the Streamlit framework and how it was implemented into the web application. Furthermore, the application's user interface is described, and the results of object detection and segmentation are shown in real-time.

Keywords: deep learning, YOLOv8, detection, segmentation, Google Colab, Streamlit

SADRŽAJ

UVOD

Duboko učenje i razvoj popratne tehnologije doprinijelo je napretku razvoja metoda u raznim područjima, uključujući detekciju objekta. Cilj ovog rada je koristeći duboko učenje istrenirati vlastiti model koji će biti korišten za detekciju i segmentiranje objekata. Za to će se koristiti YOLOv8 model, koji omogućava razvoj aplikacija koje mogu detektirati objekte u stvarnom vremenu. Da bi proces treniranja bio što uspješniji, zahtjeva detaljnu pripremu podataka i upotrebu odgovarajućih računalnih resursa.

Priprema podataka za treniranje vlastitog modela predstavlja prvi i ključan korak u cijelom procesu. To će uključivati prikupljanje, obradu i organizaciju fotografija na kojima će model biti treniran. Postupak prikupljanja slika za detekciju razlikovat će se od segmentacije zbog različitih zahtjeva u prikazu objekata.

Nakon pripreme podataka, slijedit će proces treniranja modela koji je jednako važan za uspješnost projekta. U tom dijelu rada objasnit će se različite platforme za treniranje i navesti prednosti korištenja Google Colab-a.

U posljednjem dijelu rada bit će objašnjen proces izrade same web aplikacije kroz koju će korisnici moći koristiti istrenirani model. Sučelje aplikacije bit će izrađeno koristeći Streamlit okvir te će isti biti korišten za postavljanje aplikacije na poslužitelj. U zadnjem koraku bit će objedinjeno duboko učenje i prikaz detekcije objekta u stvarnom vremenu.

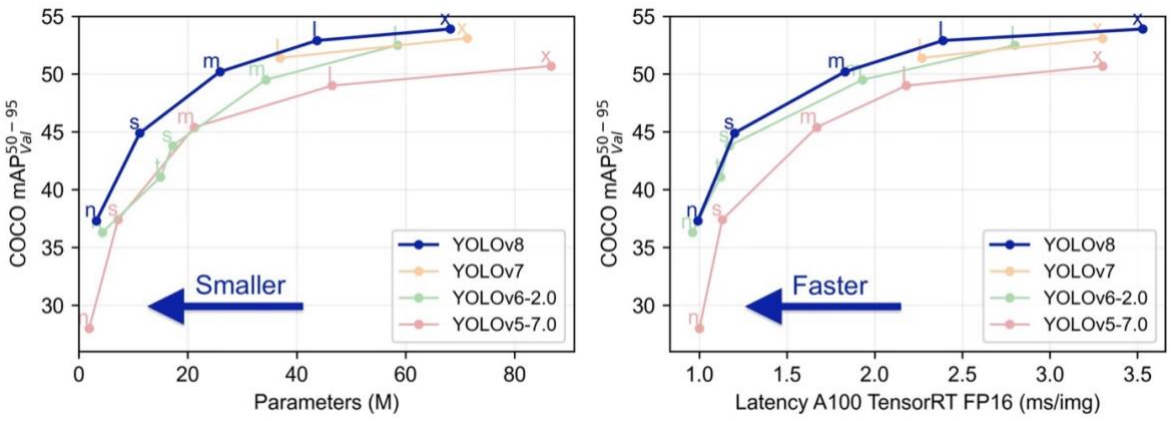
1. PREUZIMANJE FOTOGRAFIJA ZA TRENING MODELA

Priprema slika i podataka za treniranja vlastitog modela za detekciju ključan je korak u cijelom procesu. Naime, što se kvalitetnije odradi proces pripreme podataka, to će se na kraju dobiti kvalitetniji model za detekciju. Razlog tomu je što se model oslanja na te podatke kako bi naučio prepoznavati obrasce, značajke i razlike između različitih klasa objekata. Za treniranje postoje razni modeli, a za ovaj rad je odabran YOLOv8 model.

1.1. YOLOv8 model

YOLOv8 („You Only Look Once“) je računalni model za detekciju objekta koji pruža podršku za detekciju, segmentaciju i klasifikaciju objekata kroz Python pakete [1]. Također pruža podršku i za rad sa command line interface-om (sučelje naredbenog retka). Uz gotove modele, YOLO platforma služi i za treniranje naprednih vlastitih modela za duboko učenje. Neke od karakteristika koje YOLO platformu razlikuju od drugih su brzina, točnost, jednostavnost upotrebe i raznolikost primjena [2]. Ultralytics je tvrtka koja je razvila YOLO model 2015. godine na Sveučilištu u Washitonu, a osnivači su Joseph Redmon i Ali Farhadi [3]. Registriran je pod AGPL-3.0 licencom što bi značilo da je idealan za studente i entuzijaste koje zanima područje dubokog učenja. Također postoji i opcija komercijalnog korištenja.

Kako bi razumjeli arhitekturu YOLO modela, potrebno je objasniti od kojih glavnih dijelova se sastoji, a oni su: Backbone, Neck and Head [4]. Backbone je neuronska mreža i zadužena je za izdvajanje značajki iz ulaznih slika. Koristi prilagođenu mrežu kako bi poboljšala protok informacija između slojeva i povećala točnost [5]. Neck je dio koji služi kao veza između Backbone i Head dijelova. Neck objedinjuje i agregira podatke iz različitih faza Backbone-a. Također, spaja podatke različitih veličina kako bi mreža mogla detektirati objekte svih veličina. Head je završni dio i onaj koji korisnik vidi tijekom korištenja aplikacije. Zadužen je za prikaz okvira unutar kojih se detektiraju objekti, prikazuje rezultat točnosti detekcije te nazive objekata. Kao dokaz koliko je YOLOv8 model napredan govori činjenica da se koristi u aplikacijama za prepoznavanje automobilskih registracija, video nadzoru, robotici te čak u detekciji medicinskih slika.



Slika 1. Usporedba verzija YOLO modela

(Izvor: https://docs.ultralytics.com/models/yolov8/#overview)

Iz grafikona se može zaključiti kako je YOLOv8 najbrži i najmanji model do sad. To ga čini idealnim modelom za prepoznavanje objekata u stvarnom vremenu. Postiže veću *mAP* (Mean Average Precision) vrijednost uz manji broj resursa. Takva visoka razina optimizacije ovaj model čini idealnim za primjenu na mobilnim uređajima i ugradbenih sustava.

1.2. Preuzimanja fotografija za detekciju objekata

Jedan od ključnih koraka u procesu treniranja vlastitog modela je preuzimanje i priprema velikog skupa podataka. Kako bi model bio što točniji, potrebno je imati velik broj označenih fotografija. U tom slučaju pomažu nam servisi koji imaju velike skupove podataka i besplatni su za korištenje.

Open Images Dataset V7 je jedan od najpoznatijih i najvećih skupova podataka za ovu svrhu, a sadrži ukupno oko 9 milijuna fotografija. Također sadrži 16 milijuna okvira za prepoznavanje modela te 600 različitih objekata [6]. Projekt održava Google te je trenutno aktualna verzija 7. U ovom radu odabrani su objekti ručni sat i mobilni uređaj. Postoje još brojni modeli, a ovi su odabrani radi lakšeg testiranja modela. Pored jednostavnosti, ovi objekti su izabrani i zbog svoje široke upotrebe i dostupnosti u svakodnevnom životu.

Korist će se Python skripta za preuzimanje velikog broja fotografija s Open Images Dataset. Takav pristup je važan jer se pripremaju podaci za treniranje putem YOLOv8 modela. Nakon

završetka izvođenja skripte, na računalno se preuzima data.zip datoteka. U njoj se nalaze dva direktorija: *images* i *labels*. U direktoriju *images* nalaze se još 3 direktorija: train (sadrži sve fotografije specificiranih objekata), val (sve fotografije za validaciju objekata) i test (sve fotografije za test objekata). Fotografije su u .jpg formatu i svaka ima svoj ID. U direktoriju *labels* nalaze se također 3 direktorija istog imena ali s različitim podacima: train (sadrži sve datoteke sa anotacijama za trening fotografija), val (sve datoteke sa anotacijama za validiranje fotografija) i test (sve datoteke sa anotacijama za test fotografija). Svaka tekstualna datoteka je .txt formata i ID odgovara fotografiji koju predstavlja. U datotekama su zapisane koordinate koje se kasnije koriste za prikaz objekata unutar *bounding boxes*. To su pravokutnici koji se koriste u algoritmima za detekciju objekata kako bi se precizno označili položaji objekata unutar slike. Oni definiraju granice objekta pomoću koordinata i za cilj imaju minimizirati prostor izvan objekta. Bounding boxes su ključni za treniranje i evaluaciju modela prepoznavanja objekata u slikama.

Prema službenoj dokumentaciji, za svaku klasu preuzeto je 1500 fotografija kako bi model bio što bolje istreniran i osigurao visoku preciznost u prepoznavanju različitih objekata [7]. Količina podataka ključna je za povećanje točnosti modela jer veći broj primjera omogućava bolje učenje raznolikih karakteristika unutar svake klase.

download_data_google_open_images_v7_object_detection_dataset.py

download_data_google_open_images_v7_object_detection_dataset.py > ...

```
164 if __name__ == "__main__":
165     parser = argparse.ArgumentParser()
166     parser.add_argument('--classes', default=['Watch', 'Mobile phone'])
167     parser.add_argument('--out-dir', default='./data')
168     parser.add_argument('--yolov8-format', default=True)
169     parser.add_argument('--max-number-images-per-class', default=1500)
170     args = parser.parse_args()
171 
172     classes = args.classes
173     if type(classes) is str:
174         classes = ast.literal_eval(classes)
175 
176     out_dir = args.out_dir
177 
178     yolov8_format = True if args.yolov8_format in [
179         'T', 'True', 1, '1'] else False
180 
181     max_number_images_per_class = int(args.max_number_images_per_class) \
182         if args.max_number_images_per_class is not None else None
183 
184     process(classes, out_dir, yolov8_format, max_number_images_per_class)
185 
```

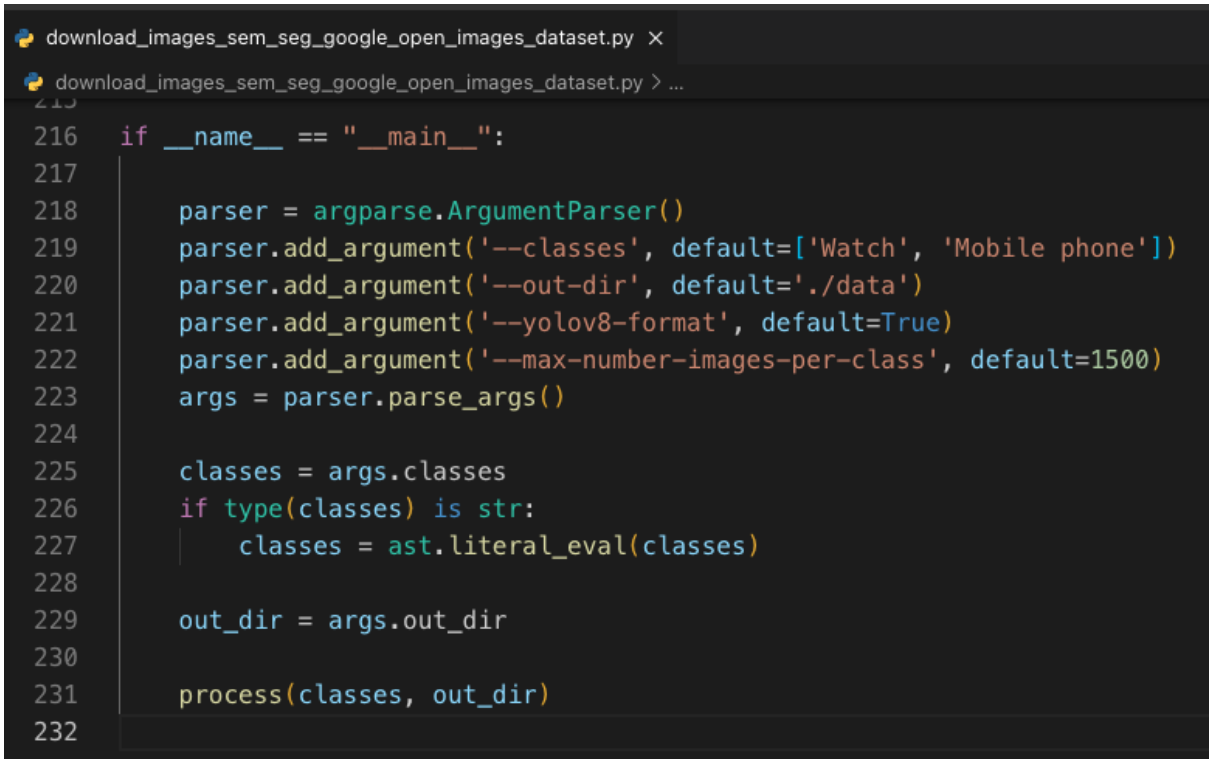
Slika 2. Dio koda skripte za preuzimanje fotografija za detekciju

1.3. Preuzimanje fotografija za segmentaciju objekata

Za preuzimanje fotografija potrebnih za treniranje segmentacije objekata koristit će se Python skripta koja se sastoji od nekoliko koraka. Prvo se preuzimaju CSV datoteke koje sadrže informacije o segmentaciji objekata (train-annotations-object-segmentation.csv, validation-annotations-object-segmentation.csv), a potom downloader.py koja se koristi za preuzimanje slika iz Open Images Dataset-a. Skripta provjerava postoje li navedene klase (Watch, Mobile phone) te pohranjuje identifikatore (ID) tih klasa za daljnju obradu. Zatim skripta čita CSV datoteke s informacijama o segmentaciji i generira popis slika koje sadrže klase za preuzimanje. Popis fotografija je zapisan u datoteku image_list_file.

Maska za segmentaciju je binarna slika koja koristi piksele kako bi označila položaj ciljanih objekata unutar fotografija [8]. U ovim maskama, pikseli koji pripadaju objektu od interesa imaju vrijednost 1, dok svi ostali pikseli imaju vrijednost 0. Na taj način, maska omogućava izoliranje određenog objekta iz slike. Ove anotacije se spremaju u tekstualne datoteke, u formatu koji je prikladan za treniranje YOLOv8 modela. Skripta zatim preuzima .zip arhiv koji sadrži maske za segmentaciju za sve fotografije u direktoriju data. Navedeni direktorij dijeli se na images i labels direktorije. To je inače standardni YOLOv8 format za treniranje. Images direktorij sadrži fotografije raspoređene u direktorije train, validation i test. Labels direktorij također slijedi navedenu strukturu te se unutar svakog poddirektorija nalaze .txt datoteke koje sadrži anotacije za odgovarajuće fotografije.

Korištenjem Open Images Dataset V7 i preciznim filtriranjem maski prema relevantnim klasama, skripta osigurava da se samo relevantni podaci koriste u procesu treniranja. Time se poboljšava kvaliteta modela i segmentacija objekata.



Slika 3. Dio koda skripte za preuzimanje fotografija za segmentaciju

Ova Python skripta koristi Google Open Images Dataset za preuzimanje slika i pripadajućih maski segmentacije prema unaprijed definiranim klasama objekata, poput "Watch" I "Mobile phone". Koristeći biblioteku *argparse* omogućuje prilagodbu postavki poput odredišnog direktorija, formata kompatibilnog s YOLOv8 modelom te maksimalnog broja slika po klasi.

2. PROCES TRENIRANJA MODELA ZA DETEKCIJU OBJEKATA

Treniranje modela za detekciju objekata predstavlja ključan korak u razvoju modela, a koji za krajnji cilj ima prepoznavanje objekta unutar fotografije ili videa. Proces uključuje rad s velikim brojem fotografija koje su ranije pripremljenije prema YOLOv8 modelu. Tijekom treniranja model prilagođava svoje parametre kroz brojna ponavljanja, sve kako bi postigao što bolje rezultate na nepoznatim fotografijama. Konačna provjera performansi i točnosti modela vrši se na testnom skupu podataka. Na kraju postupka dobiva se .pt (PyTorch Tensors) datoteka koja sadržava istrenirani model [9]. Takve vrste datoteka čuvaju arhitekturu modela i trenirane parametre u obliku tenzora, višedimenzionalnih nizova koji su ključni za duboko učenje.

PyTorch je popularna biblioteka otvorenog koda za *machine-learning* koja omogućuje: podršku za rad na grafičkim karticama (GPU), mogućnost paralelne obrade podataka i neuronskih mreža te fleksibilnost za izmjene [10].Ova biblioteka ima dobru integraciju s drugim koje su korištene u ovom radu. NumPy biblioteka je korištena za izračun numeričkih operacija, normalizaciju slika i transformaciju koordinata bounding box-ova što uvelike ubrzava treniranje modela [11]. Pandas se koristi za manipulaciju i analizu podataka, posebno kada se radi s CSV tipom datoteka. Ključan je za učitavanje i prethodnu obradu skupova podataka prije nego što se oni pretvore u PyTorch tensore (generalizacija matrica i vektora) [12]. Uz navedene biblioteke, bit će korištene još: OpenCV, Pillow, Ultralytics, Torchvision te mnogi ostali. Sve biblioteke nalazit će se u datoteci requirements.txt. koja će kasnije biti važna za pokretanje aplikacije te postavljanje iste na server.

Treniranje modela može se provoditi lokalno na računali ili na *cloud* poslužitelju, ovisno o zahtjevima i potrebama korisnika. Svaki pristup ima svoje prednosti i nedostatke, razlike u brzini izvođenja i potrebnim resursima te jednostavnosti korištenja.

Za lokalno treniranje potrebna nam je NVIDIA grafička kartica visokih performansi koja podržava CUDA, što više RAM-a i što bolji CPU (procesor). Iako je moguće treniranje modela bez grafičke kartice, ne preporuča se jer se vrijeme izvođenja naglo povećava s obzirom na to da sav posao obavlja procesor. Potrebno je instalirati sve biblioteke te konfigurirati CUDA jezgre. Fotografije i svi ostali podaci za treniranje se pohranjuju lokalno što može zauzeti i nekoliko stotina gigabajta prostora na disku. Ako bi svi navedeni uvjeti bili zadovoljeni, cijena

računala bila bi značajno veća, no istovremeno bi se i vrijeme potrebno za treniranje modela osjetno skratilo.

U ovom radu primijenit će se alternativni pristup treniranju modela, putem cloud poslužitelja. Google Colab je besplatna online platforma koju je razvila tvrtka Google, a služi za izvođenje koda direktno u internet pregledniku [13]. Također pruža besplatan pristup GPU-ovima (NVIDIA Tesla K80) i TPU-ovima (T4, P100), što ubrzava proces dubokog i strojnog učenja. Projekti se mogu međusobno dijeliti kako bi više osoba moglo raditi istovremeno na projektu. No, postoje i brojni nedostaci. Trening može trajati maksimalno 12 sati, a već nakon 90 minuta neaktivnosti server prestaje s radom. Resursi su limitirani na 12GB radne memorije i 70GB memorije na disku. Podaci se ne pohranjuju trajno nego ih je potrebno učitati na Google Drive koji ima ograničene od 15GB memorije u besplatnoj verziji. Serversko okruženje se resetira sa svakom sesijom što zahtjeva ponovno instaliranje paketa. Iako Google Colab ima svojih ograničenja, njegova pristupačnost i besplatna upotreba čine ga izuzetno vrijednim alatom za istraživanje.

2.1. Treniranje modela za detekciju

Treniranje modela na pažljivo pripremljenim fotografijama predstavlja završni korak u cjelokupnom procesu, a kvalitetno provedeni postupak direktno utječe na performanse u aplikaciji. Prije korištenja Google Colab okruženja, potrebno je napraviti direktoriji na Google Drive-u i učitati prethodno pripremljene fotografije za YOLOv8 model. Sve fotografije se nalaze u direktoriju data koji smo prethodno raspakirali jer se bio preuzet kao .zip arhiv.

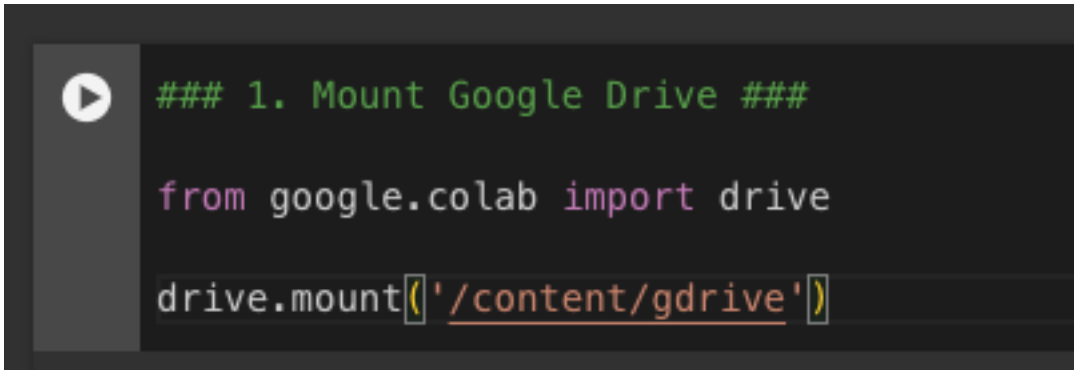
Uz podatke za treniranje, potreban nam je i google_colab_config.yaml datoteka. Jako je važna jer sadrži apsolutnu putanju do *root* direktorija, te putanju do direktorija train i val. Na taj način YOLOv8 model zna gdje tražiti podatke za treniranje. Prilikom definiranja putanja za treniranje modela, potrebno je navesti i klase koje model treba prepoznati. Ove klase se zapisuju kao lista, čime se jasno specificira skup kategorija na kojima će se model obučavati.

Slika 4. Konfiguracijska YAML datoteka

YAML (Yet Another Markup Language) je format za serijalizaciju podataka koji se najčešće koristi za konfiguracijske datoteke [14]. Lagan je za čitanje i pisanje jer koristi strukturu uvlačenja te podržava osnovne tipove podataka kao što su stringovi, brojevi i liste. Po potrebi može se jednostavno konvertirati u druge popularne formate kao što su JSON ili XML.

U direktoriju na Google Drive-u pod nazivom „ObjectDetectionYOLOv8“ dodajemo .ipynb datoteku s pomoću koje treniramo model na Google Colab okruženju. To je zapravo Jupyter Notebook (bilježnica) otvorenog koda koja sadrži Python kod i ćelije za tekst [15]. Koristi se za duboko učenje, analiziranje podataka i strojno učenje. Može se pokrenuti online koristeći Google Colab i JupyterHub ili lokalno u editoru kao što je VSCode.

Proces treniranja modela započinje povezivanjem (montiranjem) Google Drive-a s Google Colab okruženjem, što se postiže izvršavanjem koda:

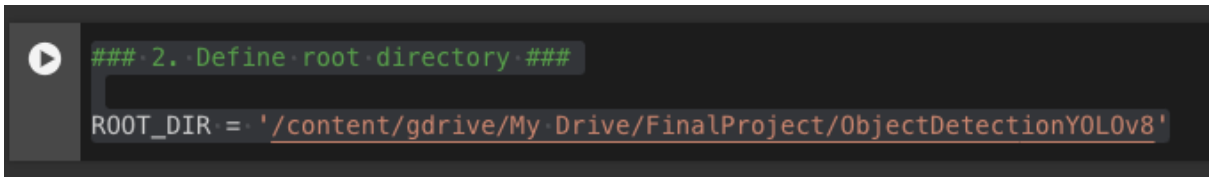


Slika 5. Integracija Google Drive-a u Google Colab okruženju

Google zahtijeva prijavu korisnika putem korisničkog računa kako bi omogućio pristup podacima pohranjenim na Google Drive-u. Nakon uspješne autentifikacije, korisnik dobiva potpuni pristup svim podacima. Ovaj pristup predstavlja značajno poboljšanje u odnosu na učitavanje podataka za svaku pojedinačnu radnu sesiju. Dodatno, svi sudionici na dijeljenom

projektu automatski dobivaju pristup podacima, što olakšava suradnju i poboljšava učinkovitost.

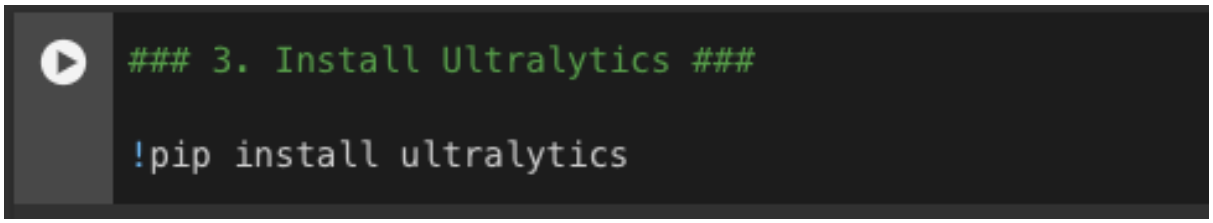
Zatim je potrebno je definirati putanju do glavnog direktorija na Google Drive-u, odnosno do lokacije na kojoj se nalaze fotografije potrebne za treniranje modela. Glavni direktoriji se zove „ObjectDetectionYOLOv8“, a smješten je unutar direktorija „FinalProject“. Potrebno je izvršiti sljedeći kod:



Slika 6. Postavljanje putanje glavnog direktorija

Po završetku treniranja, to je lokacija na koju će se spremiti rezultati treniranja modela. Rezultati će sadržavati: model za detekciju u prethodno spominjanom .pt formatu, metrike evaluacije te druge izlazne podatke potrebne za analizu i korištenje modela.

Treći korak u treniranju modela je instaliranje Ultralytics Python paketa pomoću pip paket instalera.



Slika 7. Instalacija Ultralytics paketa

Pokretanjem ove naredbe instaliramo sve ostale potrebne pakete kao što su PyTorch, OpenCV, NumPy, Pillow i mnoge drugi. Korištenjem Ultralytics paket nam omogućuje pristup već unaprijed treniranim modelima, čime značajno skraćujemo vrijeme potrebno za treniranje.
Pip je program za instaliranje Python paketa i ključan je za upravljanje raznim bibliotekama i njihovim verzijama [16]. Pojednostavljuje instaliranje i ažuriranje biblioteka, postavljanje razvojnog okruženja te se time osigurava da razvojno okruženje bude dosljedno i usklađeno. Ovakav pristup je posebno važan za ponovno kreiranje okruženja na računalima ili u timskom radu. Kasnije u radu, ovaj alat će pomoći u kreiranju virtualnog okruženja za razvoj aplikacije.

Ovo je izuzetno korisno jer omogućava rad sa različitim verzijama paketa na istom računalu, bez brige o sukobima između različitih projekata.

Četvrti korak je treniranje modela za detekciju objekata.

```
### 4. Train model ###

import os

from ultralytics import YOLO

ROOT_DIR = "/content/gdrive/My Drive/FinalProject/ObjectDetectionYOLOv8"

# Load a model
model = YOLO("yolov8n.pt")  # load pre trained model

# Use the model
results = model.train(data=os.path.join(ROOT_DIR, "google_colab_config.yaml"), epochs=20)  # train the model
```

Slika 8. Implementacija algoritma YOLOv8 za detekciju objekata

Postupak se započinje importiranjem modula os koji se koristi za rad sa direktorijima, odnosno putanjama do njih [17]. Iz Ultralytics biblioteke importira se YOLO gdje je sadržana implementacija modela za detekciju objekata. Također je potrebno ponovno definirati putanju do glavnog direktorija na Google Drive-u. Zatim se učitava unaprijed trenirani YOLOv8 model, konkretno "yolov8n.pt" verzija. To je *nano* verzija modela, koja je najmanja i najbrža, ali manje precizna u usporedbi sa većim verzijama [18]. Najbolje ju je koristiti na mobilnim uređajima gdje su brzina i efikasnost bitan faktor. *Small* verzija je malo veći i brži model od *nano* koji pruža dobar omjer brzine izvođenja i točnosti prilikom prepoznavanja objekata. Koristi se u nadzornim kamerama i jednostavnim industrijskim aplikacijama. *Medium* je veći i precizniji model od dva prethodna međutim i znatno sporiji, točnije 3x sporiji od *nano* modela. Koristi se za autonomnu vožnju, video analitiku i sigurnosne sustave. *Large* model prati metriku i veći je od svojih prethodnika i sporiji, no pruža veću točnost. Koristi se na snažim grafičkim karticama za praćenje objekata u zahtjevnim situacijama i za analitiku videa u visokoj rezoluciji. *XLarge* je najveći i najkompleksniji unaprijed trenirani YOLOv8 model i koristi se u okruženjima gdje je točnost ključna, a računalnih resursa ima na pretek.

Svi modeli primaju fotografije u veličini od 640 piksela i dijele sličnu arhitekturu kao što je backbone i head. Namijenjeni su za detekciju objekata i razlikuju se u brzini i točnosti.

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Slika 9. Usporedba performasni različitih YOLOv8 modela

(Izvor: https://docs.ultralytics.com/models/yolov8/#supported-tasks-and-modes)

Proučavanjem dokumentacije na kojoj je model treniran može se zaključiti da sadrži klase koje su potrebne u ovom radu, a to su mobilni uređaj i ručni sat [19]. To nam daje sigurnost da će trenirani model imati dobre rezultate prilikom detekcije tih objekata. Široka baza podataka platforme ImageNet, na kojoj je unaprijed trenirani YOLOv8 model, omogućava bolju generalizaciju, što znači da bi model trebao prepoznati različite varijacije u obliku, veličini i tipu uređaja.

Pozivanjem funkcije *model.train(...)* započinje treniranje modela za detekciju objekata. Argument *data* specificira putanju do YAML datoteke gdje su zapisane sve potrebne postavke za treniranje. Funkcija „data=os.path.join(ROOT_DIR, "google_colab_config.yaml")“ osigurava putanju do glavnog direktorija bez obzira o kojem operacijskom sustavu je riječ.

U strojnom učenju *ephos* se odnosi na jedan kompletan krug izvođenja treninga kroz cijelu bazu podataka [20]. Tijekom izvođenja jedne epohe, model obradi svaki podatak i prilagodi ga za daljnji postupak. Obično se modeli treniranju kroz veći broj epoha kako bi model bio što točniji i kako bi bio spreman na prepoznavanje novih objekata koje do tad nije vidio. Potrebno je pronaći balans u broju epoha jer s malim brojem dobiva se model koji je nedovoljno treniran (*underfitting*), a s velikim brojem pretreniran modela (*overfitting*) koji neće dobro generalizirati nove podatke. Radi lakšeg procesiranja podataka trening se odvija u manjim skupovima podataka, odnosno *batch-evima.* To su manji dijelovi podataka, a ovakav pristup smanjuje potrošnju radne memorije i procesorske snage.
Broj epoha je postavljen na 20 jer se model pokazao točnim i pouzdanim, a veći broj epoha nije osigurao značajan pomak u točnosti prepoznavanja objekata. Daljnje povećanje broja

epoha nije rezultiralo značajnim poboljšanjem u točnosti prepoznavanja objekata. Postupak je trajao oko 30 minuta, a razvojno okruženje na Google Colab-u je koristilo oko 5GB radne memorije i 3GB memorije za grafičku karticu. Ovi resursi nisu prelazili kapacitet unutar besplatnog okruženja platforme te su pružili dovoljno računalne snage za optimalan razvoj i testiranje modela.

Rezultati treniranja spremaju se u direktoriji train, a on sadrži: više verzija modela, metrike evaluacije, vizualizacije, grafike i konfiguracije. Model se sprema u dvije verzije: *best.pt* i *last.pt. Best* (najbolji) je proglašen najboljim modelom na osnovu performansi na validacijskom skupu podataka. U obzir se uzima metrika točnosti ili gubitka da se odredi kada je model postigao najbolje rezultate. Ovaj model je najbolji za korištenje u aplikacijama jer je postigao najbolje rezultate tokom treniranja. *Last* (posljednji) predstavlja posljednju verziju modela nakon završetka cjelokupnog procesa treniranja. To nije nužno najbolji model, ali se čuva kako bi se kasnije moglo nastaviti treniranje od te točke ili detaljno analizirati cijeli proces.

2.2. Treniranje modela za segmentaciju

Prilikom detekcije objekata model generira *bounding box* i na taj način prikazuje objekte koje je prepoznao. Međutim, takvi okviri neće pružiti informacije o stvarnom obliku objekta jer su uvijek pravokutni ili kvadratni. Ovaj način detekcije je ograničen i ne razumije strukturu objekta. U složenijim zadacima je potrebno znati točan oblik i rubove objekta, a za to nam je potrebna segmentacija objekta. Takav pristup nam omogućuje detaljniju analizu jer pruža masku na razini piksela za svaki objekt u slici. Svaki piksel se identificira kao dio određenog objekta ili pozadine, što omogućuje precizno određivanje oblika objekta i njegovih granica. Segmentacija se koristi za različite primjene gdje je potrebno imati razumijevanje o obliku objekta.

U kontekstu strojnog učenja, postoje dvije osnovne vrste segmentacije: semantička segmentacija i segmentacija instance [21]. Semantička segmentacija dodjeljuje istu oznaku za sve instance iste klase unutar fotografije, bez razlikovanja pojedinačnih instanci unutar te klase. Točnije rečeno, sve instance iste klase objekata u slici dijele istu oznaku. Na primjer, ako se na fotografiji nalazi više pametnih telefona, svaki će biti označen istom oznakom, ne vezano o

tome radi li se o različitim modelima ili pozicijama. Ovakav pristup segmentaciji koristi se kad nije potrebno razlikovati pojedinačne objekte unutar tih klasa. U primjeru autonomne vožnje, dovoljno je općenito prepoznati različite tipove objekata na cesti, bez potrebe za preciznom detekcijom svakog pojedinog vozila ili pješaka.

Segmentacija instance ide korak dalje i razlikuje svaku pojedinačnu instancu unutar iste klase. Takav pristup bi značio da ako imamo više pametni telefona na slici, svaki bi dobio zasebnu oznaku čime se mogu međusobno razlikovati. Brojni su primjeri gdje se koristi ovaj pristup: medicinska dijagnostika, poljoprivreda, prodaja, robotika i mnogi drugi.

Kao i kod treniranja modela za detekciju objekata, postupak kreće se sa povezivanjem (montiranjem) Google Drive-a s Google Colab okruženjem. Nakon autentifikacije dobiva se pristup fotografijama koje su ranije učitane na Google Drive prateći YOLOv8 strukturu. Drugi korak je postaviti točnu putanju do glavnog direktorija na Google Drive-u, a naziva se „ObjectSegmentationYOLOv8“. Kao i direktoriji za detekciju, nalazi se u unutar direktorija višeg reda „FinalProject“. Nakon toga, instalira se Ultralytics paket, zajedno sa svim potrebnim ovisnostima poput PyTorch-a, OpenCV-a i drugih, kako bi se osigurao konzistentan rad sa segmentacijskim modelima. Ultralytics omogućuje sve tehničke ovisnosti kako bi se korisnik mogao fokusirati na prilagodbu arhitekture modela.

Četvrti korak se razlikuje od treniranja modela za detekciju, jer je riječ o korištenju YOLOv8 za segmentaciju. Dok se kod detekcije model fokusira na prepoznavanje i označavanje pravokutnih okvira oko objekata, segmentacija prepoznaje precizne granice, identificirajući svaki piksel koji pripada tom objektu. Za treniranje segmentacijskog modela, koristi se posebna verzija YOLOv8 modela – YOLOv8-seg, koji je optimiziran za zadatke segmentacije. Ovaj model može istovremeno obavljati detekciju objekata i njihovu segmentaciju, što ga čini svestranim alatom. Oznaka -seg označava unaprijed trenirani YOLOv8 model koji je spreman za segmentaciju objekata.

Treniranje modela za segmentaciju traje duže u usporedbi s detekcijom objekata, jer segmentacija zahtijeva preciznije predikcije na razini piksela, dok se detekcija fokusira na prepoznavanje objekata kroz jednostavnije pravokutne okvire. Kod segmentacije, model mora definirati točne granice objekta unutar slike, što značajno povećava složenost zadatka.YOLOv8 koristi napredne slojeve za predikcije na razini piksela, što dodatno doprinosi povećanju

zahtjeva u procesiranju. Ova razina detalja znači da se obrađuje veća količina podataka, što rezultira većim potrebama za memorijom i računalnim resursima, ali i dužim vremenom treniranja.

Treniranje modela je trajalo oko sat vremena što bi značilo oko tri minute po jednoj epohi. Model dobro uči i generalizira, što se vidi iz podudaranja trendova između trening i validacijskih metrika. Nema značajnih znakova pretreniranosti (overfitting-a), a model postiže dobre rezultate u preciznosti i točnosti. Ako se nastavi treniranje, moguće je postići još bolje performanse, ali trenutni rezultati pokazuju da model već sada postiže solidne rezultate.

```
### 4. Train model ###

import os

from ultralytics import YOLO


# Load a model
model = YOLO("yolov8n-seg.pt")  # load pre trained model

# Use the model
results = model.train(data=os.path.join(ROOT_DIR, "google_colab_config.yaml"), epochs=20)  # train the model
```

Slika 10. Implementacija algoritma YOLOv8 za segementaciju objekata

2.3. Usporedba rezultata treniranja

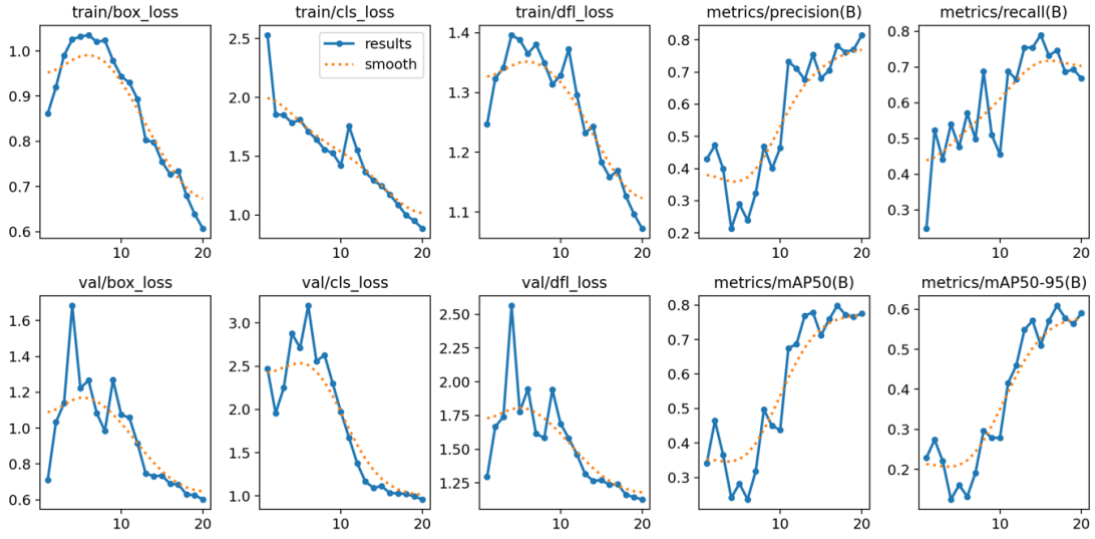
Nakon što je treniranje YOLOv8 modela završeno, generira se direktorij *detect*. U njemu se nalaze različiti grafovi, rezultati i istrenirani model. Grafovi pružaju vizualni uvid u ključne metrike performansi, poput preciznosti (*precision*), odziva (*recall*), rezultata i gubitka tijekom iteracija treninga. Ovi rezultati omogućuju dublje razumijevanje kako se model prilagođavao podacima tijekom treninga, kao i koliko je uspješan u predviđanju na novim podacima. U narednom dijelu, detaljno će se analizirati grafovi kako bi se procijenila uspješnost modela i mogućnost za daljnje optimizacije.

Kad je riječ o treningu detekcije objekata, vidljiv je dosljedan trend opadanja gubitaka i validacije kroz epohe. Takav trend upućuje na uspješnu optimizaciju modela u učenju detekcije objekata. U validaciji se također vidi opadanje, iako su prisutna određena odstupanja, što je očekivano jer model generalizira na nepoznatim podacima.

Preciznost i odziv u zadatku detekcije pokazuju rast tijekom epoha. Preciznost raste relativno stabilno, s malim odstupanjima, što pokazuje da model postaje sve bolji u identificiranju ispravnih objekata. Odziv također bilježi znatno poboljšanje, što sugerira da model postaje učinkovitiji u pronalaženju svih relevantnih objekata unutar slika.

mAP (*mean Average Precision*) je ključna metrika za ocjenu performansi u zadacima detekcije. *mAP50* i *mAP50-95* pokazuju stabilan rast, dok *mAP50* doseže visoku točnost u kasnijim epohama, što ukazuje da model ostvaruje visoku razinu preciznosti u detekciji. S druge strane, kontinuirani rast *mAP50-95* pokazuje da model zadržava visoku razinu preciznosti čak i pri zahtjevnijim uvjetima, što je vrlo pozitivan pokazatelj napretka modela.

Analiza grafova i rezultata pokazuje povremena odstupanja u ranim epohama, što sugerira da bi moglo biti potrebno bolje podešavanje hiperparametara, poput brzine učenja ili regularizacije podataka. Manja odstupanja u validacijskim gubicima ili *mAP* metrikama ukazivalo bi na stabilniju generalizaciju modela. Iako *mAP50-95* raste, i dalje postoji prostor za poboljšanje pri višim razinama preklapanja. To znači da model ponekad daje manje precizne predikcije pod strožim uvjetima, što može ukazivati na to da *bounding boxes* nisu optimalno postavljene.



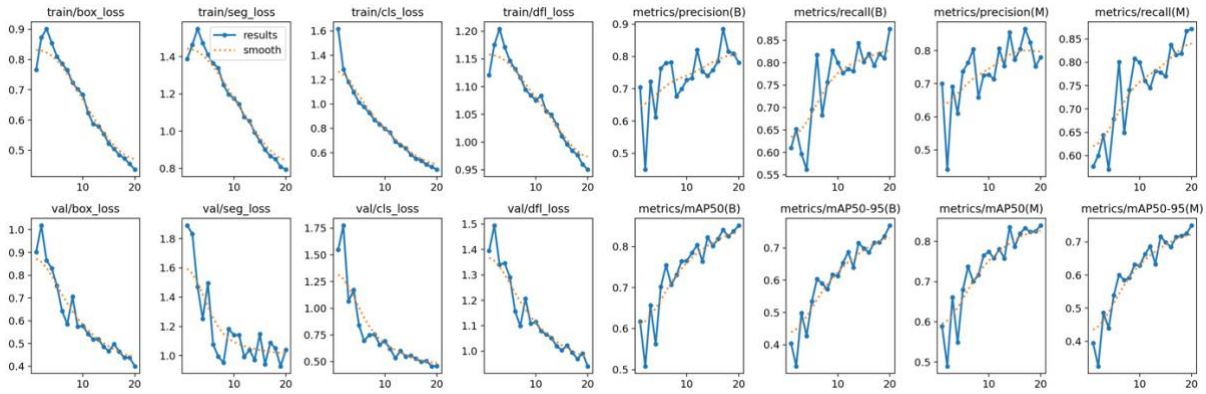
Slika 11. Rezultati treniranja modela za detekciju objekata

Kad je riječ o treniranju objekata za segmentaciju, gubitak segmentacije (*seg_loss*) pokazuje dosljedan trend opadanja tijekom treninga i validacije. Ovo upućuje na to da model dobro uči oblike i granice objekata te da postepeno poboljšava svoja predviđanja za segmentacijske maske. Brz pad segmentacijskog gubitka u ranim epohama pokazuje učinkovitu optimizaciju modela.

Slično kao kod detekcije, preciznost *(precision)* i odziv *(recall)* za segmentaciju pokazuju uzlazne trendove. Preciznost segmentacije je naročito važna jer ukazuje na to koliko dobro model može razlikovati objekt od pozadine, a ovdje vidimo da ta sposobnost jača kako epohe prolaze. Rast odziva znači da model postaje sve bolji u segmentiranju svih objekata unutar slike.

mAP za segmentaciju pokazuje pozitivan napredak. *mAP50*, koji mjeri preciznost kad maske objekata imaju barem 50% preklapanja, pokazuje visoku točnost, što znači da model dobro prepoznaje oblike objekata. Također, rast *mAP50-95*, koji mjeri preciznost pri različitim razinama preklapanja, pokazuje sve bolju preciznost, iako ima još prostora za poboljšanja.

Odstupanja u gubitku i preciznosti u validacijskim rezultatima može značiti da model nije stabilan kod složenijih objekata. Detaljnije podešavanje hiperparametara može pomoći smanjiti odstupanja. Iako raste, *mAP50-95* još ima prostora za poboljšanje na višim razinama preklapanja, što znači da model ponekad ne prepoznaje precizne rubove objekata.

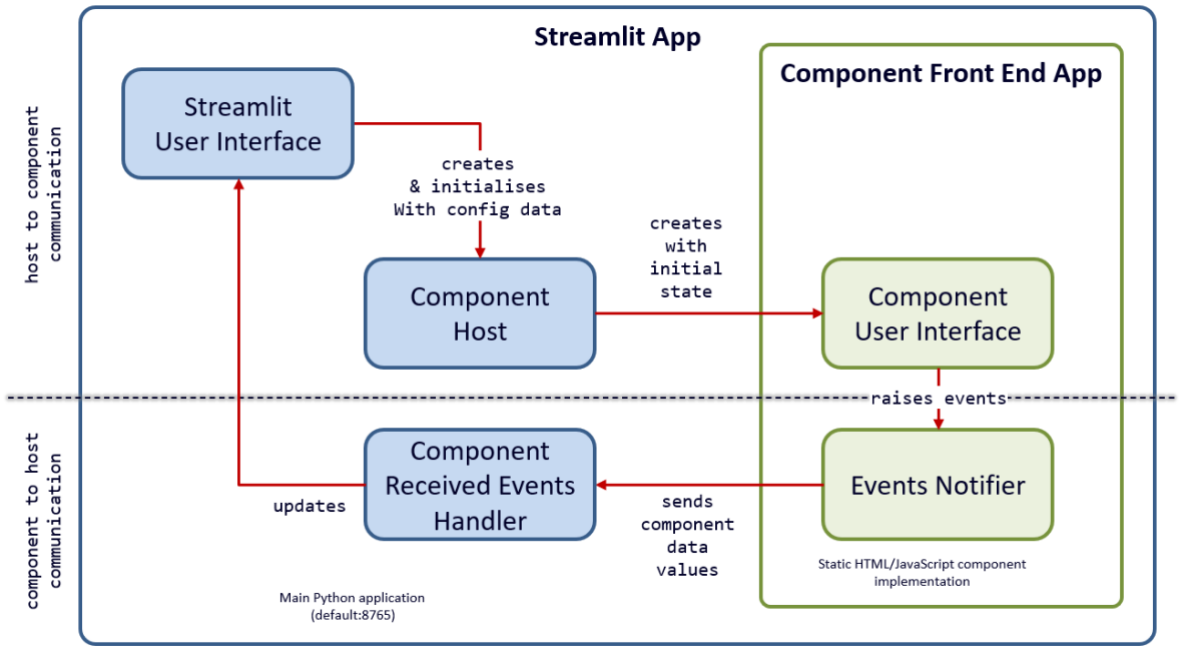


Slika 12. Rezultati treniranja modela za segmentaciju objekata

3. IZRADA WEB APLIKACIJE

U ovom poglavlju detaljno će se obraditi proces povezivanja već treniranog modela sa korisničkim sučeljem koristeći Streamlit *framework.* To je okvir koji omogućava brzu i jednostavnu izradu interaktivnih web aplikacija, što je idealno za prikazivanje modela strojnog učenja. Korisnicima će se omogućiti unos fotografija za detekciju na intuitivan način, nakon čega će aplikacija proslijediti te podatke treniranom modelu za izvođenje detekcije u stvarnom vremenu. Takav pristup je važan jer treniranje modela zahtijeva visok nivo tehničkog znanja, dok web aplikacije omogućuju jednostavnu interakciju s modelom krajnjim korisnicima koji nemaju nužno tehničku pozadinu.

Kad govorimo o razvoju softvera i programiranju, okvir (*framework*) je zbirka softverskih komponenti koje olakšavaju razvoj novih aplikacija. Ponovna upotreba postojećih komponenti ključan je princip u svim inženjerskim disciplinama [22]. Okviri također mogu postaviti i provoditi pravila vezana uz arhitekturu ili poslovne procese, čime se omogućuje dosljedan i standardiziran razvoj novih aplikacija. Također, korištenjem okvira osigurava se lakše održavanje koda, jer okvir pruža predefinirane komponente koje smanjuju potrebu za pisanjem koda od nule.



Slika 13. Arhitektura Streamlit aplikacije s komponentama

(Izvor: https://auth0.com/blog/introduction-to-streamlit-and-streamlit-components/)

3.1. Streamlit okvir

Streamlit je projekt otvorenog koda koji za cilj ima olakšavanje izrade web aplikacija za inženjere podataka i programere. Započeo je 2018. godine, a osnovala su ga tri bivša Google-ova razvojna inženjera: Adrien Treuille, Thiago Teixeira i Amanda Kelly. Glavna im je vodilja bilo kreirati okvir koji će podatkovnim znanstvenicima (*data scientists)* olakšati razvoj web aplikacija.

Prva službena verzija (0.49) predstavljena je javnosti 2019. godine i ubrzo je postala popularna među programerima. Razlog tome je jednostavnost transformiranja Python koda u interaktivne web komponente. Druga ključna prekretnica u razvoju okvira dogodila se 2020. godine kad je predstavljen „*Streamlit Cloud*“. Ova značajka omogućuje jednostavno postavljanje aplikacija na web platformu, čineći ih javno dostupnima i pristupačnima široj publici. Osim toga, Streamlit Cloud unaprijedio je timsku suradnju, omogućujući više korisnika da istovremeno rade na istom projektu, dijeleći resurse i zajednički unaprjeđujući aplikaciju [23].

Koliko je ovaj okvir popularan govori činjenica da su prikupili donacije u vrijednosti od čak 62 milijuna američkih dolara. To im je uveliko pomoglo za razvoj Streamlit Cloud projekta koji je skup i zahtjevan za održavanje. Također valja naglasiti da je Streamlit repozitorij jedan je od najpopularnijih Python okvira na GitHub-u. Dokaz tome je brojna zajednica programera koja je u konstantom rastu, što potvrđuje činjenica da su programeri izradili brojne komponente po mjeri te ih dijele s ostalima.

U ožujku 2022. godine, kompanija Snowflake preuzela je Streamlit za nepoznati iznos. Međutim, projekt je i dalje otvorenog koda te ne naplaćuje svoje korištenje. Koriste ga brojne kompanije i organizacije, pogotovo u sektorima kao što su: financije, zdravstvo i školstvo. Pomaže im prilikom vizualizacije podataka i za prikaz raznih grafikona. Programeri ga koriste za prikaz modela za detekciju objekta i izradu analitičkih prikaza.

Ono što najviše odlikuje Streamlit okvir je njegova jednostavnost za izradu korisničkog sučelja. Nije potrebno poznavati web tehnologije kao što su HTML, CSS, ili JavaScript, dovoljno je imati znanje Python programerskog jezika. Cijeli API je dizajniran da bude napisan u Python-u, koji je najpopularniji jezik među podatkovnim inženjerima. Za razliku od tradicionalnih web

okvira kao što su Flask ili Django, Streamlit nema potrebe za pisanjem složenog *backend* koda. Također nisu potrebne posebne konfiguracije za postavljanje ruta, baza podataka i *frontend* elemenata. Streamlit omogućava korisnicima da se fokusiraju na izradu prototipa, umjesto na infrastrukturne detalje aplikacije.

Streamlit koristi deklarativni stil programiranja što bi značilo da programeri opisuju što žele, umjesto da se fokusiraju na način na koji će se prikazati na webu. To čini okvir jednostavan za početnike te ga ubrzo mogu koristiti u izradi aplikacija. Uz navedeno, minimalni su zahtjevi za šablonskim kodom. To bi značilo da, na primjer, za prikaz podatkovne tablice korisnik može trebati samo jednu liniju Python koda, umjesto izgradnje složene back-end ili front-end infrastrukture.

Kao što je već spomenuto, Streamlit je projekt potpuno otvorenog koda, što bi značilo da je cijeli kod dostupan na platformi GitHub. Takav pristup rezultira aktivnom zajednicom programera koji doprinose zajednici, a mnoge značajke dolaze iz njihovih povratnih informacija. Korisnici mogu direktno komunicirati s razvojnim timom, testirati nove značajke i prijavljivati greške, što osigurava kontinuirano poboljšavanje.

Kao zaključak možemo navesti da je Streamlit revolucionirao način na koji znanstvenici podataka i inženjeri razvijaju i dijele interaktivne aplikacije. Jednostavnost pri izradi aplikacija koristeći samo Python uklanja tradicionalne prepreke web razvoja. Kroz otvorenu suradnju platforme i programera stvorena je živa zajednica koja zajedno pridonosi napretku projekta.

3.2. Izrada korisničkog sučelja

Korisničko sučelje aplikacije mora biti jasno i dobro definirano kako bi bila jednostavna i intuitivna za korištenje. Dizajn treba omogućiti brz pristup svim funkcijama, bez pretrpanosti informacijama koje bi mogle zbuniti korisnika. Elementi sučelja trebaju biti lako prepoznatljivi, a navigacija logična i prilagođena potrebama korisnika. Sve to pružna okvir Streamlit jer se pomoću njega kreiraju interaktivne web aplikacije. Uz pomoć tog okvira nije potrebno puno znanja u dizajnu korisničkih sučelja kako bi aplikacija zadovoljila sve navedeno. Valja naglasiti kako je bitno strukturirati kod u skladu s dobrom praksom razvoja softvera. To je ključno za održivost, skalabilnost i lakoću održavanja aplikacije. Korištenjem modularnog

pristupa, kod se može podijeliti u zasebne funkcije koje se odnose na različite dijelove korisničkog sučelja. Jasno imenovanje varijabli i funkcija u Streamlit aplikacijama, zajedno s dokumentacijom, omogućuje bolju razumljivost koda drugim programerima.

Aplikacija je koncipirana na način da osigura maksimalnu funkcionalnost i jednostavnost korištenja. Na lijevoj strani su smješteni ključni alati za rad s treniranim modelom, što omogućuje korisniku brz i jednostavan pristup svim funkcijama. Centralni dio aplikacije rezerviran je za prikaz dobivenih rezultata. Takav pristup naglašava rezultate kao ključni ishod interakcije s modelom. Također, ovakav dizajn omogućuje intuitivno upravljanje procesom analize i manipulacije modelom. Korisnici mogu lako pristupiti potrebnim funkcijama, što doprinosi učinkovitosti i preciznosti u radu s treniranim modelima.

Na početku korištenja aplikacije potrebno je odabrati vrstu prepoznavanja objekata: detekcija ili segmentacija. Razlike između te dvije vrste su ranije navedene i objašnjenje, a zadana vrijednost je detekcija objekta. Odabir se vrši tako da se pritisne jedan od dva *radio button-a* koji su inače standardni tipovi grafičkog prikaza dugmadi. Nakon toga potrebno je pomoću klizača postaviti granicu između 25% i 100%, što služi za određivanje preciznosti prepoznavanja objekata. Početna vrijednost je postavljena na 40%, što je dobar omjer između strogosti prilikom detekcije i fleksibilnosti. Ova postavka omogućuje korisnicima da prilagode performanse aplikacije prema svojim potrebama, omogućujući im da balansiraju između brzine obrade i točnosti prepoznavanja objekata.

Korisnici mogu izabrati izvor podataka za prepoznavanja objekata: fotografija, videozapis ili web kamera. Nakon odabira, aplikacija dinamički reagira na izbor, prikazujući relevantne elemente. Po odabiru opcije za učitavanje fotografija, korisnicima se nudi opcija da sa svog uređaja učitaju fotografiju na kojoj žele izvršiti prepoznavanje objekata. Dozvoljeni formati su: JPG, JPEG, PNG ili HEIC. Nakon što se fotografija učita, potrebno je pritisnuti gumb „Detect Objects“. S lijeve strane prikazuje se originalna fotografija, a s desne strane okviri oko prepoznatih objekta. Okviri prikazuju naziv klase koje su prepoznali i postotak sigurnosti. Ako je odabrana segmentacija objekata, tad će objekt biti jasno odvojen od pozadine. Bit će vidljive točne konture objekata, čime se stvara detaljniji prikaz svih elemenata unutar fotografije. Rezultati su prikazani jasno i mogu se preuzeti ili podijeliti s drugima.

Korisnici ne mogu birati vlastite videozapise za detekciju, već su dostupni unaprijed definirani videozapisi koji su odabrani kako bi se omogućila optimalna demonstracija funkcionalnosti aplikacije. Naime, obrada dugih videozapisa u visokoj rezoluciji zahtijeva značajne računalne resurse, što može predstavljati izazov za *cloud* servise poput Streamlit Clouda. Prilikom prepoznavanja objekata, videozapis automatski započinje reprodukciju, a korisnici mogu pratiti označene okvire koji se prikazuju oko detektiranih objekata. Ovi okviri jasno vizualiziraju prepoznate klase u stvarnom vremenu.

Treća i posljednja opcija za izvor podataka za prepoznavanje objekata jest web kamera, koja se prvenstveno odnosi na kamere u prijenosnim računalima. Međutim, aplikacija je responzivna te također funkcionira na mobilnim uređajima. Prvo je potrebno web pregledniku dopustiti pristup kameri, a zatim uslikati fotografiju. Aplikacija će nakon toga analizirati snimljenu sliku i prikazati rezultate detekcije, ističući prepoznate objekte kroz označene okvire i pridružene klase u stvarnom vremenu. Na taj način omogućuje korisnicima brzu i vizualno jasnu povratnu informaciju o prepoznatim elementima. Ako korisnik odabere opciju segmentacije objekata, aplikacija će dodatno prikazati segmentirane dijelove slike, bojeći svaki prepoznati objekt i vizualno odvajajući ga od pozadine. Na ovaj način korisnici mogu preciznije analizirati oblik i granice objekata.

Slika 14. Izgled korisničkog sučelja aplikacije

(Izvor: https://objecttrackerapp.streamlit.app/)

3.3. Postupak postavljanja aplikacije na web poslužitelj

Streamlit okvir razvio je Streamlit Cloud platformu preko koje se aplikacije mogu postaviti na poslužitelj i na taj način biti dostupne svima na Internetu. Postupak je relativno jednostavan i sastoji se od više koraka. No na samom početku najbitnije je lokalno testirati aplikaciju, kako bi izbjegli greške prilikom postavljanja aplikacije na poslužitelj.

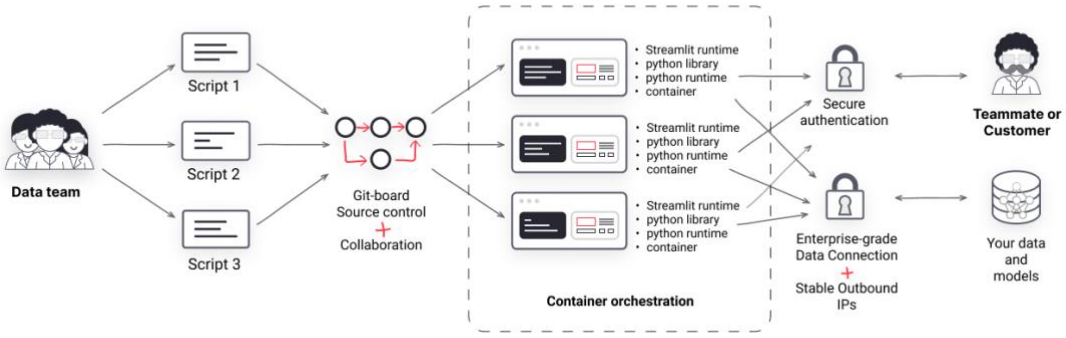
Postupak započinje pristupom na GitHub platformu te kreiranjem repozitorij u kojem će se nalaziti aplikacija. Potrebno je obratiti pažnju da se priloži i *requirements.txt* datoteka. Ona je ključna jer specificira sve Python biblioteke potrebne za funkcioniranje aplikacije. Ova datoteka se obično nalazi u glavnom direktoriju projekta i sadrži verzije potrebnih biblioteka za funkcioniranje aplikacije. U glavnom direktoriju se još nalaze pod direktoriji *images, videos* i *weights.* U *images* se nalazi fotografija za početnu stranicu aplikacije, dok se unutar *videos* nalaze videozapisi za detekciju objekta. Direktoriji *weights* služi za pohranu modela za detekciju i segmentaciju.

Potom je potrebno registrirati se na Streamlit Cloud platformu i to putem GitHub računa. Na taj način se daje pristup svim repozitorijima na tom korisničkom računu. GitHub omogućuje jednostavnu integraciju sa Streamlit Cloud-om, olakšavajući naknadno dodavanje novih značajki. Ova integracija omogućuje automatsko ažuriranje aplikacije na Streamlit Cloud-u svaki put kada se napravi „*push*“ na određenu granu repozitorija. Streamlit Cloud koristi promjene iz GitHub repozitorija kako bi u stvarnom vremenu rekonstruirao aplikaciju, što omogućuje brzu iteraciju i testiranje.

Jednom kada je kod na GitHub-u, sljedeći korak je odabrati glavnu datoteku koja pokreće aplikaciju. Najčešće ime za tu datoteku je *app.py.* Također, potrebno je odabrati glavnu granu repozitorija. Ova postavka osigurava da Streamlit Cloud zna koju datoteku koristiti za pokretanje aplikacije, što olakšava njeno automatsko ažuriranje i provođenje promjena svaki put kada se izvrši nova promjena u repozitoriju.

Ukoliko postupak prođe bez grešaka u kodu, aplikacije je vidljiva i javno dostupna svima putem poveznice. Također moguće je staviti poseban naziv aplikacije u poveznicu. Jednom kada je aplikacija objavljena, može se lako dijeliti s drugima, što olakšava suradnju i testiranje

na uređajima različitih rezolucija. Također, Streamlit Cloud pruža statistiku posjeta, omogućujući programerima da prate koliko ljudi koristi aplikaciju.



Slika 15. Arhitektura Streamlit Cloud okruženja

(Izvor: https://blog.streamlit.io/introducing-streamlit-cloud/)

ZAKLJUČAK

U ovom radu prikazano je kako temeljita priprema podataka i ispravna konfiguracija modela izravno utječu na uspješnost treniranja modela za detekciju i segmentaciju objekta. Primjena YOLOv8 modela pokazala se prikladnom za zadatke koji zahtijevaju brzinu i točnost. Uz to, Google Colab platforma omogućila je besplatno i optimizirano treniranje modela bez potrebe za visokim lokalnim računalnim resursima. Na taj način postupak treniranja dostupniji je široj zajednici programera i studenta što je bilo od iznimne važnosti za ovaj rad.

Postupak detekcije objekta dao je primarni uvid u trenirane objekte, u konkretnom slučaju ručni sat i mobilni uređaj, dok je segmentacija omogućila preciznije prepoznavanje oblika objekata na razini piksela. Navedeni postupci nam daju uvid u mogućnosti dubokog učenja, treniranja i prikaza modela u stvarnom vremenu.

Nakon završenog postupka treniranja modela, dan je uvid u rezultate čime se potvrđuje uspješnost provedenog projekta. Također, analizom rezultata dan je i zaključak u čemu se može napredovati kako bi trenirani model imao još bolje performanse.

Krajnji rezultat i svrha treniranja je web aplikacija u kojoj svi korisnici imaju mogućnost isprobati sve značajke aplikacije, odnosno isprobati detekciju i segmentaciju navedenih objekata.

Popis literature

[1] „Explore Ultralytics YOLOv8“, [Na internetu]. Dostupno: https://yolov8.com/ [pristupano 25.08.2024.]

[2] „YOLOv8“, [Na internetu]. Dostupno: https://docs.ultralytics.com/models/yolov8/ [pristupano 25.08.2024.]

[3] „Where to Start“, [Na internetu]. Dostupno: https://docs.ultralytics.com/#where-to-start [pristupano 26.08.2024.]

[4] Juan Pedro, „Detailed Explanation of YOLOv8 Architecture — Part 1“, 2023, [Na internetu]. Dostupno: https://medium.com/@juanpedro.bc22/detailed-explanation-of-yolov8-architecture-part-1-6da9296b954e [pristupano 26.08.2024.]

[5] Jane Torres, „YOLOv8 Architecture: A Deep Dive into its Architecture“, 2024, [Na internetu]. Dostupno: https://yolov8.org/yolov8-architecture/ [pristupano 26.08.2024.]

[6] „Overview of Open Images V7“, [Na internetu]. Dostupno: https://docs.ultralytics.com/models/yolov8/ [pristupano 01.09.2024.]

[7] „Tips for Best Training Results“, [Na internetu]. Dostupno: https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results/ [pristupano 01.09.2024.]

[8] „Isolating Segmentation Objects“, [Na internetu]. Dostupno: https://docs.ultralytics.com/guides/isolating-segmentation-objects/ [pristupano 01.09.2024.]

[9] Mysterious obscure, „A Deep Dive into Model Files (.pkl, .pt, .h5) and the Magic of Machine Learning“, 2024, [Na internetu]. Dostupno: https://medium.com/@mysterious_obscure/a-deep-dive-into-model-files-pkl-pt-h5-and-the-magic-of-machine-learning-740768317e76 [pristupano 04.09.2024.]

[10] „PyTorch documentation“, [Na internetu]. Dostupno: https://pytorch.org/docs/stable/index.html [pristupano 04.09.2024.]

[11] „NumPy: the absolute basics for beginners“, [Na internetu]. Dostupno: https://numpy.org/devdocs/user/absolute_beginners.html [pristupano 05.09.2024.]

[12] „Getting started“, [Na internetu]. Dostupno: https://pandas.pydata.org/docs/getting_started/index.html#getting-started [pristupano 05.09.2024.]

[13] „Introduction to Colab“, [Na internetu]. Dostupno: https://cloud.google.com/colab/docs/introduction [pristupano 08.09.2024.]

[14] „What is YAML?“, 2023, [Na internetu]. Dostupno: https://www.redhat.com/en/topics/automation/what-is-yaml [pristupano 10.09.2024.]
[15] „Project Jupyter Documentation“, [Na internetu]. Dostupno: https://docs.jupyter.org/en/latest/ [pristupano 10.09.2024.]
[16] „Overview of Python Packaging“, [Na internetu]. Dostupno: https://packaging.python.org/en/latest/overview/ [pristupano 14.09.2024.]
[17] „os — Miscellaneous operating system interfaces“, [Na internetu]. Dostupno: https://docs.python.org/3/library/os.html [pristupano 14.09.2024.]
[18] „Object Detection“, [Na internetu]. Dostupno: https://docs.ultralytics.com/tasks/detect/ [pristupano 16.09.2024.]
[19] „ImageNet.yaml“, [Na internetu]. Dostupno: https://github.com/ultralytics/ultralytics/blob/main/ultralytics/cfg/datasets/ImageNet.yaml [pristupano 16.09.2024.]
[20] „Epoch in Machine Learning“, 2024, [Na internetu]. Dostupno: https://www.geeksforgeeks.org/epoch-in-machine-learning/ [pristupano 20.09.2024.]
[21] Kanan Vyas, „Object Segmentation“, 2020, [Na internetu]. Dostupno: https://medium.com/visionwizard/object-segmentation-4fc67077a678 [pristupano 22.09.2024.]
[22] „What is a Framework in Programming and Engineering?“, [Na internetu]. Dostupno: https://aws.amazon.com/what-is/framework [pristupano 23.09.2024.]
[23] Raymond Lee, „A brief history of Streamlit“, 2023, [Na internetu]. Dostupno: https://ray3168.medium.com/abrief-history-of-streamlit-8d916c454429 [pristupano 25.09.2024.]

Popis slika