Usporedba metoda dubokog učenja za detekciju objekata u sklopu strojnog vida

Vuletić, Kristian

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Kristian Vuletić

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

Izv. prof. dr. sc. Tomislav Stipančić, dipl. ing.

Student:

Kristian Vuletić

Zagreb, 2024.

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

Zahvaljujem se mentoru na savjetima i usmjeravanju i kolegama na poslu za pristup hardveru za izradu eksperimenta. Posebno hvala obitelji i kolegama na fakultetu na podršci.

Kristian Vuletić

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

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

ZAVRŠNI ZADATAK

Student: **Kristian Vuletić** JMBAG: **0035221082**

Naslov rada na hrvatskom jeziku: **Usporedba metoda dubokog učenja za detekciju objekata u sklopu strojnog vida**

Naslov rada na engleskom jeziku: **Comparison of deep learning methods for object detection in machine vision**

Opis zadatka:

U svijetu algoritama i umjetne inteligencije u tijeku je ubrzani razvoj novih metoda i pristupa. Pred znanstvenom i istraživačkom zajednicom svakim danom se pojavljuju unaprijeđeni i novi modeli umjetne inteligencije u sklopu različitih primjena.
U radu je potrebno istražiti pristupe i trendove razvoja metoda dubokog učenja kod detekcije objekata na slikama. U sklopu toga potrebno je:

- identificirati i opisati osnovne principe kod razvoja metoda dubokog učenja kroz povijest tako da se procijeni uporabljivost te dobre i loše strane tih principa kod razvoja budućih modela
- identificirati barem dva pristupa kod razvoja modela dubokog učenja te ih usporediti i evaluirati koristeći modele dubokog učenja koji ih predstavljaju
- trenirati i testirati odabrane modele koristeći javno dostupne baze podataka kao što su ImageNet ili COCO.

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

Zadatak zadan: 30. 11. 2023.

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

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

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

Predsjednik Povjerenstva: Prof. dr. sc. Damir Godec

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

Oznaka	Opis
AdaBoost	Adaptive Boosting
AP	Average Precision
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CUDA	Compute Unified Device Architecture
FPN	Feature Pyramid Network
FPS	Frames Per Second
HOG	Histogram of Oriented Gradient
mAP	Mean Average Precision
R-CNN	Regions with CNN features
ResNet	Residual Network
R-FCN	Region-based Fully Convolutional Network
RoIAlign	Region of Interest Align
RoIPool	Region of Interest Pooling
RPN	Region Proposal Network
SIFT	Scale Invariant Feature Transform
SSD	Single Shot multibox Detector
SVM	Support Vector Machine
VGG	Visual Geometry Group
VOC	Visual Object Classes
VGG	Visual Geometry Group
YOLO	You Only Look Once

SAŽETAK

U svijetu detekcije objekata došlo je do velikog porasta popularnosti metoda dubokog učenja, u tijeku je ubrzani razvoj novih metoda. U sklopu rada identificirani su temeljni pristupi tih metoda, sličnosti i razlike identificiranih pristupa te su opisane razne metode razvijene kroz povijest. Nakon toga, proveden je eksperiment u kojem su dvije metode trenirane i testirane na javnim bazama podataka.

Ključne riječi: detekcija objekata, duboko učenje

SUMMARY

In the world of object detection there has been a big surge in popularity of deep learning methods, new methods are being developed at a very fast pace. This paper identifies the fundamental approaches of those methods, their similarities and differences and then describes the various methods developed throughout history. After that, and experiment is conducted in which two methods are trained and tested on publically available databases.

Key words: object detection, deep learning

1. UVOD

Detekcija objekata jedan je od kompleksnijih aspekata strojnog vida. Uključuje dvije kategorije strojnog vida: lokalizaciju objekata i klasifikaciju objekata. Detekcijom objekata dobijaju se lokacije objekata na slici i njihove klase što ju čini korisnom u raznim primjenama na primjer autonomnoj vožnji [1], detekciji registracijskih tablica [2] i proizvodnim sustavima [3].

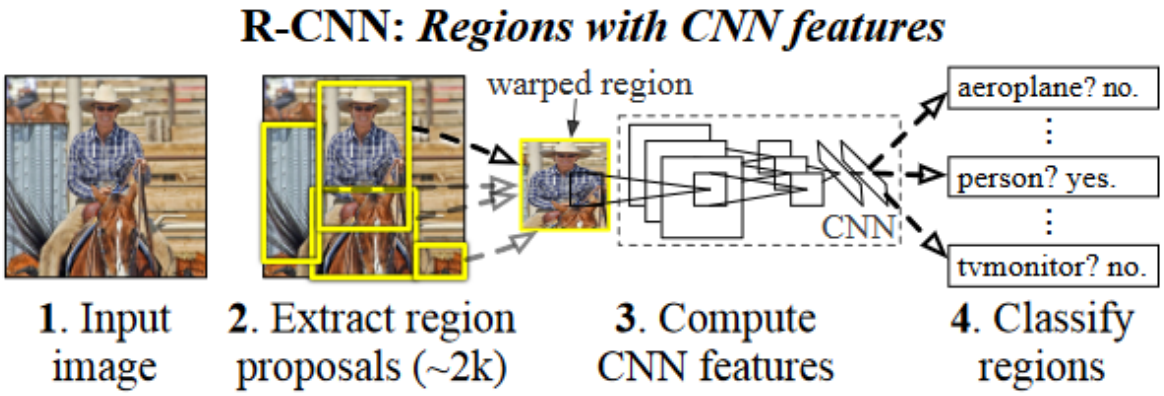
Prethodno, u detekciji objekata, koristile su se takozvane tradicionalne metode koje su opisane u tri koraka: informativni odabir regije, izvlačenje značajki i kasifikacija.

Informativnim odabirom regije pronalaze se dijelovi slike u kojima se nalaze objekti, to se postiže skeniranjem čitave slike kliznim prozorom višestrukih omjera. Nedostatak ovakvog pristupa je u tome što je zahtjevan i neefikasan, proizvesti će veliki broj nepotrebnih prozora, a smanjenje broja generiranih prozora može pogoršati rezultate.

Izvlačenjem značajki dobija se semantički opis svih objekata koje je potrebno detektirati izvlačenjem njihovih vizualnih značajki. Više metoda je razvijeno u ovu svrhu kao što su na primjer Scale Invariant Feature Transform (SIFT) [4] i Histogram of Oriented Gradient (HOG) [5]. Nedostatak tih metoda je što im varijabilnost u slikama uzrokovana na primjer različitim pozicijama objekata, šumom ili različitim osvjetljenjem znatno utječe na rezultate.

Klasifikacija je konačni korak detekcije objekata kojim se nakon pronalaska objektata oni dijele u klase kako bi se mogli razlikovati jedni od drugih, dajući potpunije informacije o slici. Postoji više rješenja kao što su na primjer Support Vector Machine (SVM) [6] i AdaBoost [7]. Njihov nedostatak je što daju loše rezultate na većim skupovima podataka sa više šuma to jest gdje dolazi do preklapanja klasa.

Do velikog napretka u strojnom vidu došlo je pojavom konvolucijskih neuronskih mreža (CNN) [8], što dovodi do razvoja takozvanih metoda dubokog učenja strojnog vida. Veći pomak ostvaren je razvojem metode Regions with CNN features (R-CNN) [9] koja, umjesto metode kliznog prozora, odabir regije postiže generiranjem oko 2000 prijedloga regija dobivenih segmentacijom slike i tradicionalne metode izvlačenja značajki zamjenjuje konvolucijskom neuronskom mrežom koja je sposobna izvući kompleksnije značajke iz slika u usporedbi sa tradicionalnim metodama. Slika 1. prikazuje princip rada R-CNN metode.



Slika 1. Princip rada R-CNN metode [9]

Daljnjim razvojem metoda dubokog učenja istaknula su se dva pristupa: dvokoračne metode na temelju prijedloga regija (Faster-RCNN) [10] i jednokoračne metode koje regresiju i klasifikaciju obavljaju na čitavoj slici, bez prijedloga regija (RetinaNet) [11]. Oba pristupa imaju svoje prednosti i nedostatke međusobno u usporedbi. Koji pristup odabrati uvelike ovisi o slučaju upotrebe, potrebama korisnika, dostupom hardveru, potencijalnom pretprocesiranju slika, naknadnoj obradi detekcija itd., zbog čega identificiranje optimalnog pristupa može biti teško. U ovom radu obraditi će se oba pristupa te zatim testirati na javno dostupnim bazama podataka i usporediti njihove rezultate i performanse u svrhu procjene njihove upotrebljivosti na raznim slučajevima upotrebe.

2. KONVOLUCIJSKE NEURONSKE MREŽE U DETEKCIJI OBJEKATA

Konvolucijske neuronske mreže pojavile su se mnogo godina prije razvoja R-CNN metode, no dugo vremena nisu imale koristi u detekciji objekata, niti u drugim područjima, prvenstveno zbog nedostatka velikih baza podataka i nedovoljno snažnog hardvera. Posljednjih godina došlo je do mnogo razvoja što je dovelo do povećanog interesa prema metodama dubokog učenja u detekciji objekata od strane istraživačke zajednice: snažniji hardver, javno dostupne velike baze podataka i biblioteke za lakši rad s podacima i treniranje modela sa istim.

Hardver je neprekidno u razvoju, svake godine izlaze nove inačice hardvera što dovodi do učestalih poboljšanja u snazi i efikasnosti, za duboko učenje najvažnija su unaprijeđenja ostvarena na grafičkim procesorima koja, u usporedbi sa središnjim procesorima, omogućuju višestruko bržu obradu podataka i treniranje modela [12]. Također, znatan je i razvoj Compute Unified Device Architecture (CUDA) [13] alata koji omogućava uporabu Nvidia grafičkih procesora u programiranju.

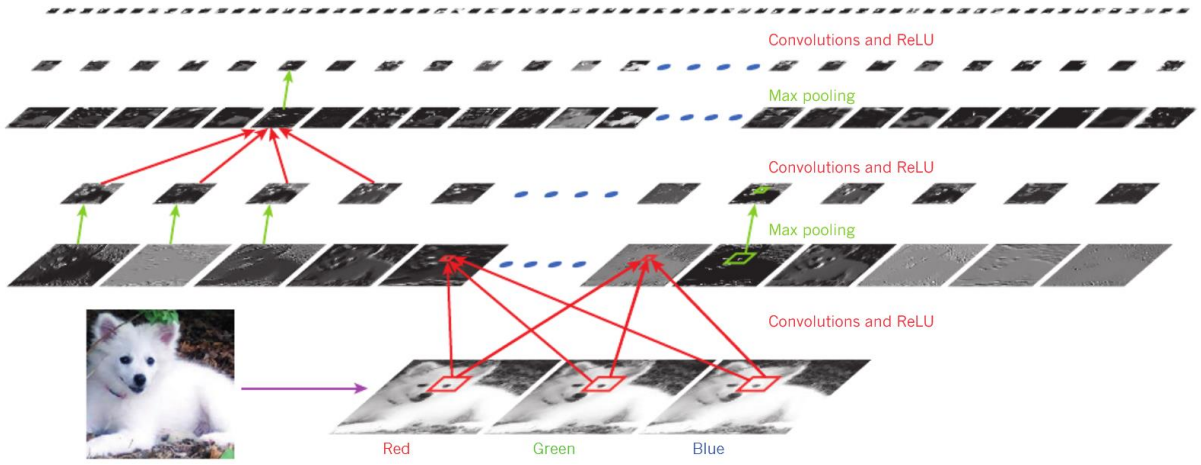
U treniranju modela metodama dubokog učenja, jako je važno koristiti što veći i što kvalitetniji skup podataka kako bi se ostvarili što bolji rezultati. Danas je dostupno mnogo javno dostupnih velikih baza podataka za razne svrhe, kod detekcije objekata ističu se Common Objects in Context (COCO) [14] koji sadrži 330 tisuća slika sa 1,5 milijuna instanci objekata i Pascal Visual Object Classes (VOC) [15] koji sadrži 11 tisuća slika sa 27 tisuća instanci objekata. Ove dvije baze podataka su od velike važnosti u istraživačkim radovima gdje se često koriste za evaluaciju novorazvijenih metoda [9-11].

Veliki značaj u razvoju metoda dubokog učenja nose i biblioteke razvijene za projekte strojnog učenja. One omogućuju lakše rukovanje sa višedimenzionalnim tenzorima, kreiranje novih i učitavanje postojećih metoda strojnog učenja i treniranje modela. Ističu se TensorFlow [16] i PyTorch [17].

Navedenim napretcima, uporaba konvolucijskih neuronskih mreža postaje sve češća i nove metode se neprestano razvijaju. Konvolucijske neuronske mreže dovele su do velikih poboljšanja u detekciji objekata, razlog tome je kompleksna struktura takvih mreža što ih čini prikladnim za složenije probleme.

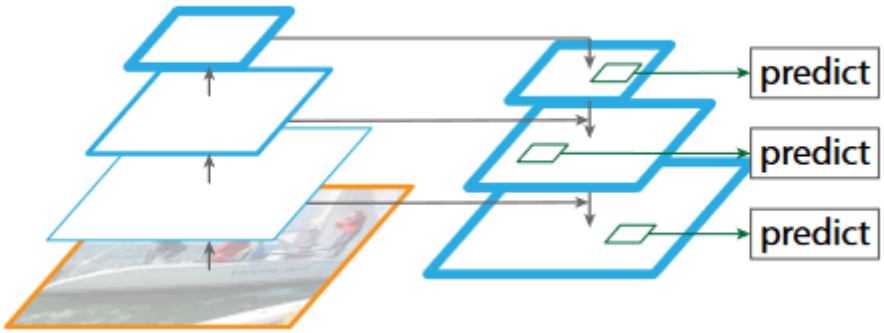
Prvi veći napredak ostvaren je pojavom R-CNN metode koja je tradicionalne metode izvlačenja značajki slika zajmijenila konvolucijskom neuronskom mrežom koja prolaskom slike kroz svaki njen sloj primjenjuje filtere na sliku i umanjuje njenu rezoluciju čime se iz slike

dobiju abstraktne značajke visoke razine koje su manje osjetljive na šum od značajki niske razine izvučenih na razini piksela. Slika 2. prikazuje način rada konvolucijske neuronske mreže u izvlačenju značajki.



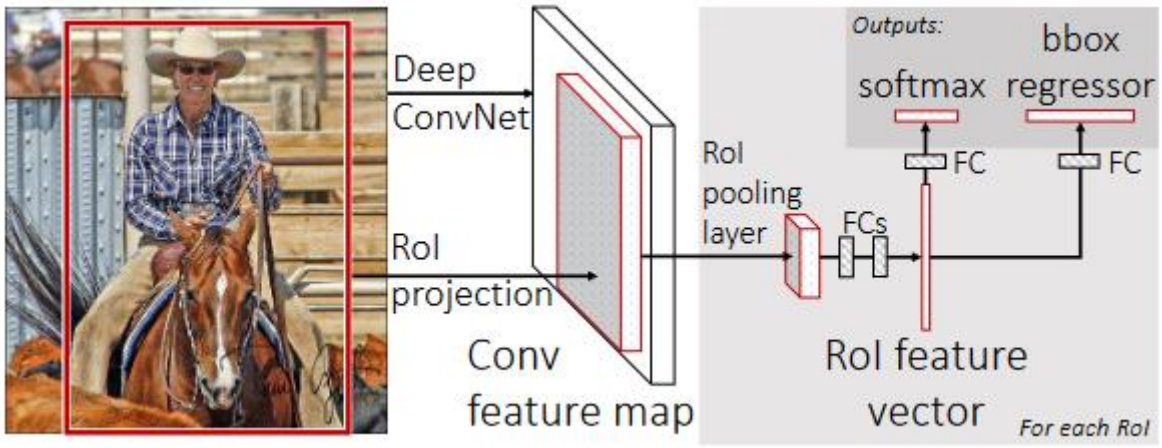
Slika 2. Izvlačenje značajki visoke razine konvolucijskom neuronskom mrežom [8]

Iako navedena metoda generalno znatno poboljšava rezultate detekcije objekata, poboljšanja se najviše vide na objektima srednje i velike veličine (objekti koji zauzimaju veći postotak piksela slike, u COCO dataset-u definirani kao objekti veći od 32x32 piksela), manji objekti imaju znatno lošije rezultate [11]. Razlog tome je što, pri smanjenju rezolucije slike, mali objekti su reducirani na jako mali broj piksela te mreža ne može izvući mnogo značajki o njima. Kako bi se poboljšali rezultati za objekte raznih veličina, razvijena je Feature Pyramid Network (FPN) arhitektura [18]. Ova mreža kao ulazni podatak uzima izlazni podatak konvolucijske mreže za izvlačenje podataka, sliku najmanje rezolucije, i generira slike veće rezolucije kako bi izvukla više značajki za male objekte. Ovom metodom se izvlače značajke iz svih rezolucija koje mreža generira, ne samo zadnje, čime se dobijaju značajke za razne veličine objekata (Slika 3.).



Slika 3. Princip rada Feature Pyramid Network arhitekture [18]

Konvolucijske mreže također su dovele do napretka u regresiji graničnih okvira i klasifikacije objekata. Metodom Fast-RCNN [19] tradicionalne metode regresije okvira i klasifikacije zamijenjene su dvjema podmrežama koje rade paralelno te se istovremeno mogu trenirati, zbog čega značajke slika nije potrebno pohranjivati na računalo, što je znatno ubrzalo treniranje modela i poboljšalo rezultate. Slika 4. prikazuje strukturu Fast-RCNN metode.



Slika 4. Struktura Fast-RCNN metode [19]

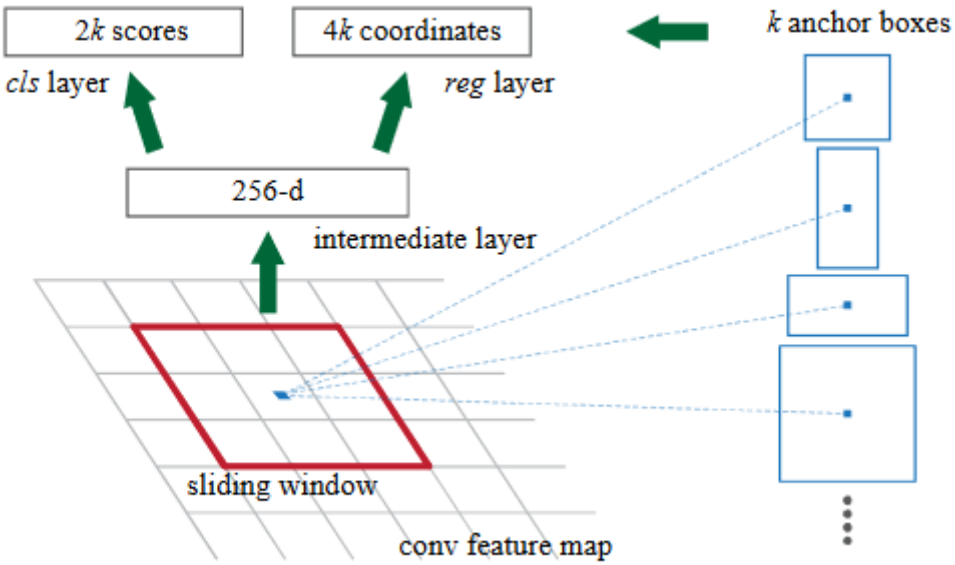
Svi navedeni doprinosi su, zbog poboljšanja do kojih su doveli, postali temelj metoda dubokog učenja u detekciji objekata. U tijeku je neprestani razvoj novih metoda no, neovisno o tome koja se moderna metoda primijenjuje u projektima detekcije objekata, ovi doprinosi se mogu pronaći u gotovo svakom projektu koji koristi moderne metode dubokog učenja.

3. MODERNE METODE DUBOKOG UČENJA ZA DETEKCIJU OBJEKATA

Sa rastućom popularnošću metoda dubokog učenja u detekciji objekata, neprestano se razvijaju nove metode, svaka donoseći nove ideje u područje. Unatoč tome, te metode imaju i neke sličnosti u svojim pristupima detekciji objekata, što dovodi do podijele metoda dubokog učenja na dvije vrste: dvokoračne metode temeljene na prijedlozima regija i jednokoračne metode koje detekcije obavljaju na čitavoj slici, ne služeći se prijedlozima regija. Te dvije vrste razlikuju se najviše u jednoj stvari: načinu na koji traže objekte po slikama. Nijedan pristup nije objektivno bolji od drugog, svaki ima svoje prednosti i nedostatke.

3.1. Dvokoračni pristup

Metode koje koriste dvokoračni pristup rade na način da, nakon izvlačenja značajki slike, mape značajki dijele na regije te se onda na regijama traže lokacije i klase objekata. Moderne metode to postižu zasebnom mrežom za prijedloge regija (RPN) [10] koja se, u fazi treniranja modela, trenira istovremeno uz mrežu za detekciju te uči prepoznati na kojim regijama u slici je najveća vjerojatnost da sadrže tražene objekte. Slika 5. prikazuje princip rada mreže za prijedloge regija. Neke od istaknutiji metoda dvokoračnog pristupa su: R-CNN, Fast R-CNN, Faster R-CNN, R-FCN [20] i Mask R-CNN [21].



Slika 5. Princip rada mreže za prijedloge regija [10]

R-CNN prva je metoda koja je pokazala potencijal dubokog učenja u detekciji objekata. No taj potencijal demonstrira u izvlačenju značajki, pri generiranju prijedloga regija ne uvodi konvolucijsku neuronsku mrežu. R-CNN prijedloge regija generira metodom selektivne

pretrage [22]. Ova metoda generira prijedloge regija segmentacijom slike [23] te kombinira regije koje se preklapaju koristeći više strategija kako bi pronašla objekte raznih veličina. R-CNN metodom selektivne pretrage generira ručno određeni broj regija, koji je u izvornom radu postavljen na 2000. Metoda je postigla srednju prosječnu preciznost (*mAP*) od 53,7% na Pascal VOC 2010 bazi podataka, što je veliko poboljšanje na tada najbolji rezultat na istoj bazi podataka od 40,4% *mAP* [9]. Slična poboljšanja ostvarena su i na drugim često korištenim bazama podataka poput Pascal VOC 2007 i 2012.

Fast R-CNN, osim već spomenutih promjena u drugom poglavlju, uvodi promjene u načinu rukovanja sa prijedlozima regija. Metoda generira prijedloge regija selektivnom pretragom isto kao i R-CNN no, umjesto izvlačenja značajki iz svake regije, izvlači značajke iz čitave slike te prijedloge regija udružuje sa dobivenom mapom značajki i generira vektore značajki, jedan za svaki prijedlog regija. Zbog doprinosa ove metode navedenih u drugom poglavlju i činjenice da izvlačenje značajki obavlja jednom po slici umjesto jednom za svaki prijedlog regija, Fast R-CNN dovodi do velikog ubrzanja i poboljšane preciznosti naspram R-CNN, ostvarujući *mAP* veći od 68% na Pascal VOC 2007, 2010 i 2012 bazama podataka [19].

Faster R-CNN uvodi velike promjene u načinu generiranja prijedloga regija u usporedbi sa R-CNN i Fast R-CNN metodama. Staru metodu mijenja mrežom za prijedloge regija, konvolucijskom neuronskom mrežom koja iz mape značajki generira skupinu pravokutnih prijedloga objekata i iznos koji predstavlja vjerojatnost da ta regija predstavlja objekt, ne pozadinu. Te regije su zatim predane Fast R-CNN mreži koja na regijama obavlja klasifikaciju i regresiju graničnih okvira. Mrežu za prijedloge regija potrebno je trenirati poput mreže za detekciju objekata. Mreža tijekom treninga vraća dva gubitka, jedan za regresiju granica regija i jedan za klasifikaciju regija. Ta dva gubitka se zatim zbrajaju te se ta ukupna vrijednost koristi dalje u treniranju modela. Jednadžba (1) prikazuje način računanja gubitka mreže za prijedloge regija.

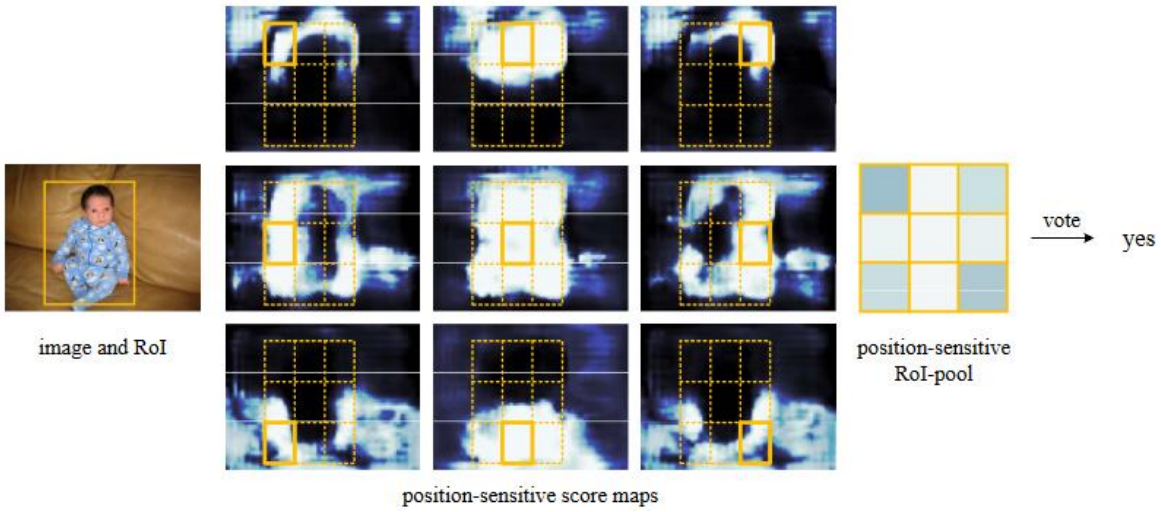
$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*) \qquad \textbf{(1)}$$

i predstavlja index granice okvira koje predviđa mreža, p_i je vjerojatnost da granica okvira *i* sadrži objekt. Varijabla p^*_i predstavlja oznaku graničnih okvira temeljne istine, iznosi 1 ako je predviđeni granični okvir pozitivan i 0 ako je negativan. Vektor t_i sadrži 4 koordinate koje opisuju lokaciju i veličinu predviđenih granica dok t^*_i sadrži koordinate okvira temeljne istine povezanog sa pozitivnim predviđenim okvirom. L_{cls} je gubitak u klasifikaciji na dvije klase,

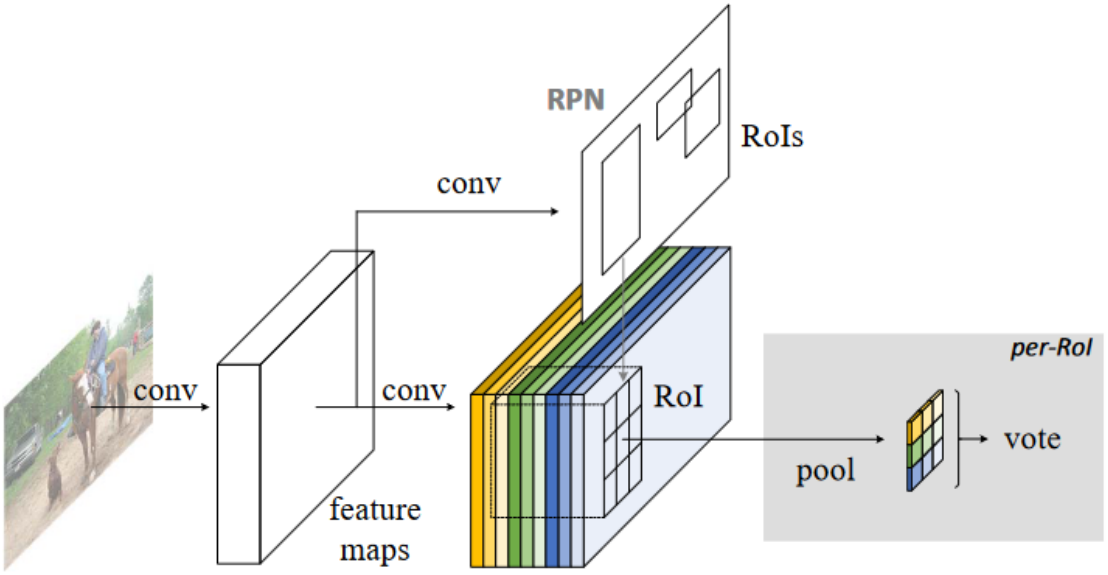
objekt i pozadina, dok je L_{reg} gubitak regresije graničnih okvira. N_{cls} i N_{reg} su normalizacijski faktori pripadajućih gubitaka i λ je balansirajuća težina.

Uvođenjem mreže za prijedloge regija, Faster R-CNN je u odnosu na prethodne metode ostvario poboljšanja i u učinkovitosti i preciznošću. Metoda je testirana na bazama podataka Pascal VOC 2007 i 2012, u oba slučaja ostvaruje *mAP* veći od 70% [10].

Razvoj Region-based Fully Convolutional Network (R-FCN) metode nadahnut je pojavom novih klasifikacijskih mreža (koje u kontekstu detekcije objekata u ovom radu zovemo mrežama za izvlačenje značajki). Prethodno su se u ovu svrhu koristile mreže koje su sadržavale više potpuno povezanih slojeva poput AlexNet [24] i VGG mreža [25] no poslije su razvijene mreže koje su potpuno konvolucijske, to jest sadrže samo jedan potpuno povezani sloj, njihov poslijednji sloj. Među takvim mrežama ističu se rezidualne mreže (ResNet) [26] i GoogLeNet [27]. R-FCN primjenom ResNet mreže pokušava postići efikasniju detekciju objekata, također uvodi promjene u radu sa prijedlozima regija, na mapi značajki, paralelno dok mreža za prijedloge regija generira prijedloge, konvolucijskim slojem generira mape vjerojatnosti koje pokazuju vjerojatnost da je objekt sadržan na određenoj lokaciji na slici. Kada se regije interesa generiraju, na temelju mape vjerojatnosti, dobija se izračun vjerojatnosti da regija interesa sadržava traženi objekt te se samo regije sa dovoljno visokom vjerojatnošću prosljeđuju dalje na regresiju graničnih okvira i klasifikaciju (Slika 6.). R-FCN metoda je testirana na Pascal VOC 2007 i 2012 bazama podataka gdje je ostvarila *mAP* preko 80%, također je testirana na COCO bazi podataka gdje je ostvarila *AP* od 31,5%. U svakom testu metoda je imala lošiju preciznost od Faster R-CNN modela koji je koristio istu ResNet mrežu, no znatno je efikasnija obrađujući slike brzinom od 0,17 sekudni po slici u usporedbi sa Faster R-CNN modelom čija je brzina iznosila 3,36 sekundi po slici [20]. Slika 7. prikazuje strukturu R-FCN metode.



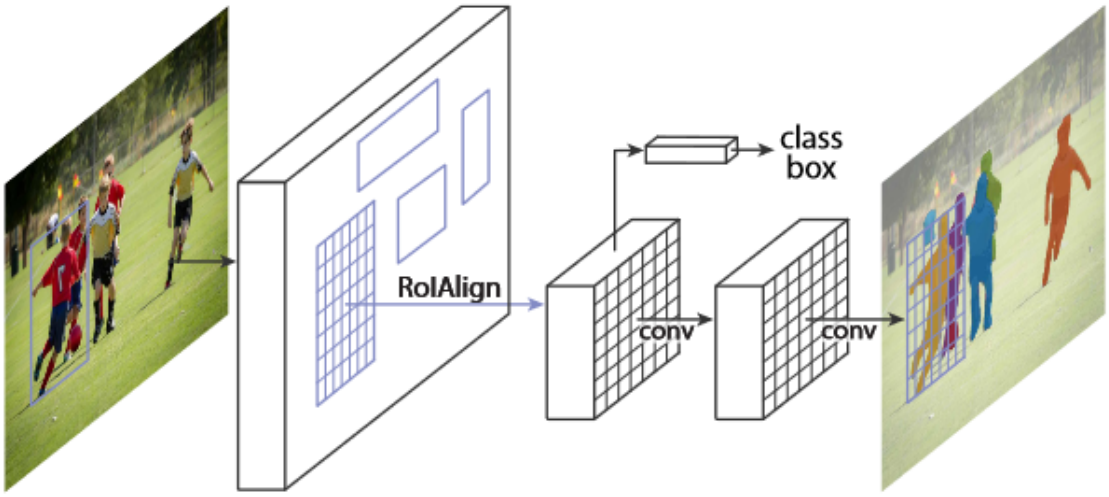
Slika 6. Primjena regije interesa na mape vjerojatnosti [20]



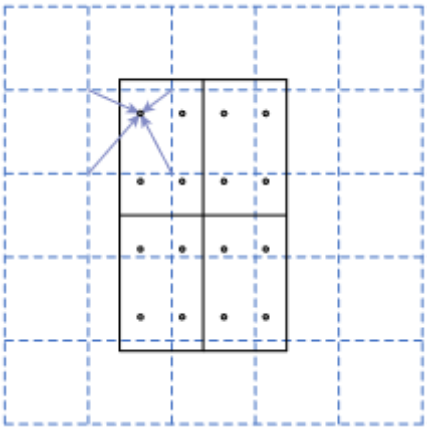
Slika 7. Struktura R-FCN metode [20]

Mask R-CNN metoda modificira Faster R-CNN metodu dodavajući mogućnost segmentacije slika uz klasifikaciju i lokalizaciju, što postiže uvođenjem konvolucijskih slojeva koji, paralelno uz klasifikaciju i regresiju graničnih okvira, izrađuje binarne segmentacijske maske na svaku regiju interesa (Slika 8.). Također, bitno je spomenuti modifikaciju koju metoda uvodi u načinu rada sa regijama interesa. Naime, metoda koju koriste Fast R-CNN i Faster R-CNN za stvaranje mapa značajki, RoIPool [19] može uzrokovati neusklađenost između mape značajki i regije interesa radi načina računanja koordinata mapa značajki, koordinate originalne slike dijeli sa brojem konvolucija i taj broj zaokružuje na cijeli broj. Ta neusklađenost nije stvarala probleme u Fast i Faster R-CNN metodama jer ne utječe mnogo na klasifikaciju no može značajno utjecati na segmentaciju koja mora biti precizna na razini piksela. Kako bi se riješio taj problem Mask R-CNN metoda uvodi RoIAlign [21], metodu koja pri dijeljenju ne zaokružuje vrijednost na cijeli broj već zadržava decimalne vrijednosti te udruživanje regija interesa ostvaruje bilinearnom interpolacijom (Slika 9.). Unatoč tome što je ovaj rad bio usredotočen na segmentaciju slika doveo je do poboljšanja i u detekciji objekata, na COCO bazi podataka testirane su dvije verzije modela, jedna u kojoj se uopće ne treniraju slojevi za segmentaciju, što u radu zovu Faster R-CNN sa RoIAlign i verzija u kojoj treniraju i slojeve za segmentaciju no u testiranju uzimaju u obzir samo granične okvire i klase. Oba modela dala su bolje rezultate od Faster R-CNN metode ostvarujući bolji *AP* na objektima male i srednje veličine. Nedostatak Mask R-CNN metode je što joj je za treniranje potrebna baza podataka koja, osim graničnih okvira i klasa, treba imati i maske za sve objekte. Takve baze podataka su

rijeđe i ručna izrada takvih baza uzima mnogo vremena zbog čega Mask R-CNN metoda često može biti nepraktična no i u takvim slučajevima, mreža je korisna jer se može trenirati i bez slojeva za segmentaciju što bi, prema rezultatima rada originalnog rada, trebalo dati bolje rezultate od Faster R-CNN metode zbog RoIAlign metode.



Slika 8. Struktura Mask R-CNN metode [21]



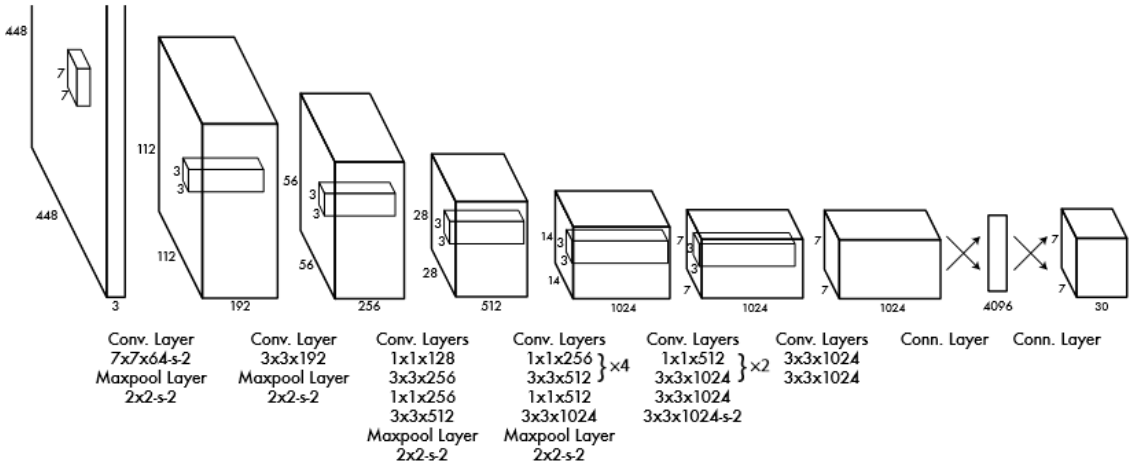
Slika 9. RoIAlign metoda [21]

3.2. Jednokoračni pristup

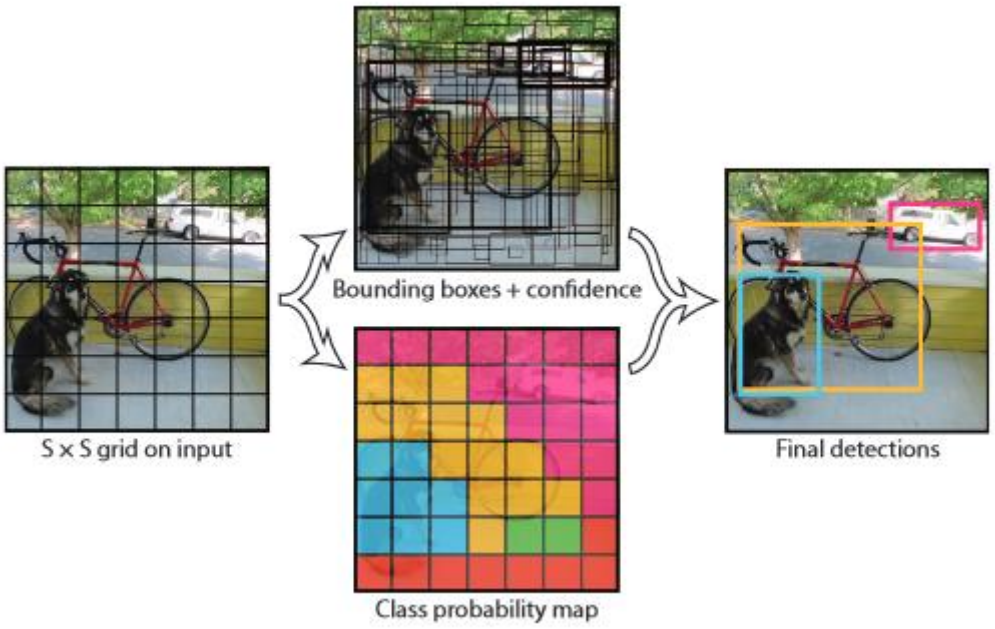
Metode koje koriste jednokoračni pristup, za razliku od metoda dvokoračnog pristupa, rade na način da, nakon izvlačenja mapi značajki iz slika, klasifikaciju i regresiju graničnih okvira obavljaju na čitavim mapama značajki umjesto pretraživanja regija interesa. Ideja ovakvog pristupa je postići bržu i efikasniju detekciju objekata uz što manji gubitak preciznosti. Struktura ovakvih metoda je generalno slična metodama dvokoračnog pristupa uz nedostatak mreže za prijedloge regija. Neke od značajnih metoda s ovim pristupom su: You Only Look

Once (YOLO) [28], Single Shot multibox Detector (SSD) [29], YOLOv2 [30], RetinaNet i YOLOv3 [31].

YOLO metoda je mnogo jednostavnija po strukturi i principu rada od dvokoračnih metoda, stvorena je s ciljem ubrzanja detekcije objekata i ostvarenja detekcije u stvarnom vremenu. Mreža se sastoji od 24 konvolucijska sloja i 2 potpuno povezana sloja (Slika 10.), za razliku od dvokoračnih metoda nema mrežu za generiranje prijedloga regija već detekciju izvodi na čitavoj slici, to postiže primjenjivanjem rešetke dimenzija *S* * *S* na mape značajki te u svakoj ćeliji radi predikcije graničnih okvira i vjerojatnosti za klase (Slika 11.). Prednosti ovakvog pristupa su najočitiji u performansama, metoda radi mnogo brže od dvokoračnih metoda, toliko brže da čak omogućava i detekciju u stvarnom vremenu. Najveći razlog tomu je rad sa jednom neuronskom mrežom, zbog nedostatka mreže za prijedloge regija metoda po slici obavlja samo jedno prosljeđivanje prema naprijed što metodu čini puno bržom i u treniranju i validaciji. Nedostatak ovog pristupa je u preciznosti, metoda generira ćelije samo u omjeru 1:1 i za svaku ćeliju generira samo dva granična okvira i jednu klasu. To dovodi do velikog broja ograničenja u detekciji, metoda ima općenito problema s detekcijom objekata male veličine, uz to, loše detektira objekte raznih omjera, objekte koji se preklapaju ili skup objekata koji su blizu jedan drugom i objekte različitih klasa koji se nalaze u istoj ćeliji. Metoda je testirana na Pacal VOC 2007 i 2012 bazama podataka, na bazi podataka iz 2007. mjeri *mAP* i brzinu u slikama po sekundi (FPS), a na 2012. samo *mAP*. Na Pascal VOC 2007 ostvaruje *mAP* od 63,4%, što je dosta niže od 73,2% koje ostvaruje Faster R-CNN no, s druge strane, YOLOv1 ostvaruje 45 FPS-a što je mnogo brže od 7 koje ostvaruje Faster R-CNN. Na Pascal VOC 2012 YOLOv1 ostvaruje *mAP* od 57,9% što je opet dosta niže od Faster R-CNN metoda koja je ostvarila 70,4% [28].

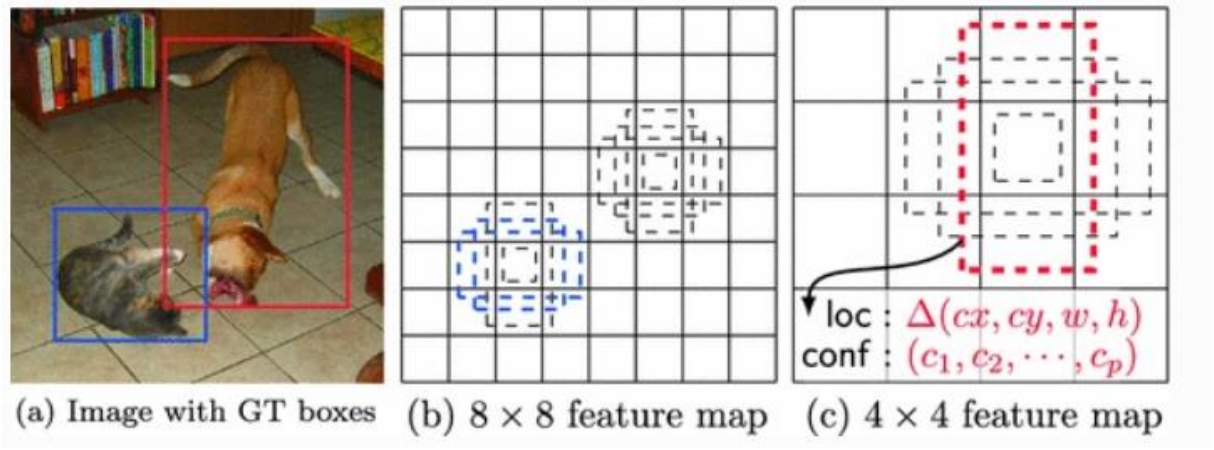


Slika 10. Struktura YOLOv1 mreže [28]

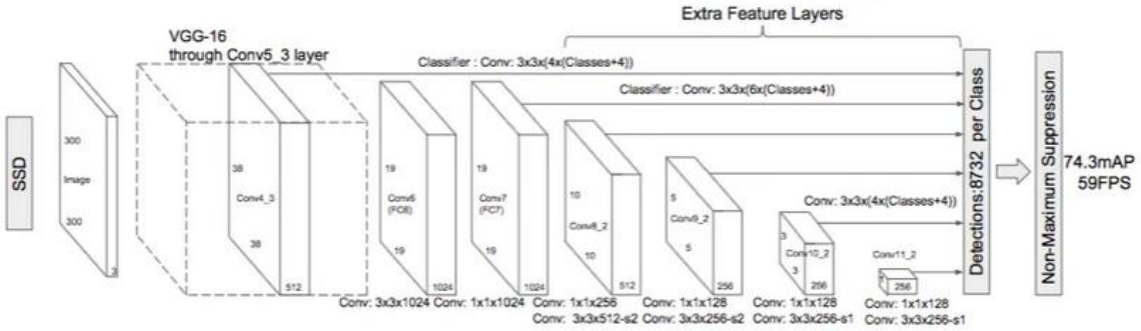


Slika 11. Princip rada YOLOv1 metode [28]

SSD metoda koristi sličan pristup YOLOv1 metodi, no ostvaruje znatno poboljšanje u preciznosti i to na nižoj rezoluciji ulaznih slika (300x300 za SSD, 448x448 za YOLOv1) čime postiže i bolje performanse. Problem loše detekcije objekata raznih veličina rješava dijeleći mape značajki rešetkama različitih dimenzija. Time metoda generira ćelije različith veličina što omogućava detekciju objekata raznih veličina. Slijedeće, lošu detekciju objekata raznih omjera rješava na način da svakoj ćeliji zadaje granične okvire raznih veličina i omjera, ovaj pristup također uklanja ograničenje od dvije detekcije po ćeliji (Slika 12.). Model je testiran na bazama podataka Pascal VOC 2007, 2012 i COCO. U svim testovima navodi dvije verzije modela: SSD300 i SSD512 nazvani po rezoluciji ulaznih podataka za koju su konfigurirani. Na Pascal VOC 2007 bazi podataka uspoređuje se sa Faster R-CNN metodom, SSD300 ostvaruje 79,6% *mAP*, a SSD512 81,5%, oba rezultata nadmašuju Faster R-CNN koji je ostvario 78,8%. Na Pascal VOC 2012 bazi podataka u test uvode i YOLOv1 metodu, SSD300 ostvaruje *mAP* od 77,5%, a SSD512 80,0%, oba rezultata nadmašuju i Faster R-CNN koji je ostvario 75,9% i YOLOv1 koji je ostvario 57,9%. Na COCO bazi podataka metoda se opet uspoređuje samo sa Faster R-CNN metodom, sa presjekom preko unije od 0,5 kao graničnom vrijednošću, SSD300 ostvaruje *mAP* od 41,2%, a SSD512 46,4%, Faster R-CNN ostvaruje rezultat između prethodna dva sa *mAP* od 45,3%. Osim bolje preciznosti, značajno je što je metoda također i mnogo brža od navedenih metoda s kojima se uspoređivala ostvarujući brzinu od 59 FPS [29]. Slika 13. prikazuje strukturu SSD mreže.



Slika 12. Princip rada SSD metode [29]



Slika 13. Struktura SSD mreže [29]

YOLOv2 metoda, poznata i pod imenom YOLO9000, unaprijeđena je verzija prve YOLO metode. Cilj ove verzije je poboljšanje YOLOv1 metode oslovljavanjem njenih prethodno navedenih nedostataka. YOLOv2 uvodi nekoliko poboljšanja s tim ciljem. Prvo su uvedene promjene u mreži za izvlačenje značajki slika, te mreže su u ostalim metodama prethodno trenirane na bazama podataka za klasifikaciju slika na nižim rezolucijama što znači da pri treniranju modela za detekciju objekata, mreža za izvlačenje značajki se mora prilagoditi ne samo detekciji objekata već i povećanoj rezoluciji ulaznih slika. YOLOv2 umjesto toga mrežu prethodno trenira na povećanoj rezoluciji na ImageNet [32] bazi podataka za klasifikaciju. Slijedeće, kako bi riješila problem detekcije objekata raznih veličina i omjera, YOLOv2 metoda uvodi dvije velike promjene: izvlačenje mapi značajki iz zadnja dva sloja konvolucijske mreže umjesto samo iz zadnjeg kako bi izvukla mape veće rezolucije i upotreba graničnih okvira raznih veličina i omjera koje računa primjenom k-means algoritma na bazu podataka za treniranje. I konačno, kako bi riješila problem klasifikacije objekata koji su blizu jedan drugom ali su različitih klasa, metoda obavlja klasifikaciju na svakom graničnom okviru zasebno umjesto stvaranja mape vjerojatnosti klasa. Metoda je testirana na Pascal VOC 2007 i 2012 i COCO bazama podataka. Model koji radi sa slikama rezolucije 544x544 ostvaruje 40 FPS-a i

na Pascal VOC 2007 ostvaruje najbolji *mAP* dok na Pascal VOC 2012 i COCO bazama podataka ima lošije rezultate od SSD512 modela [30].

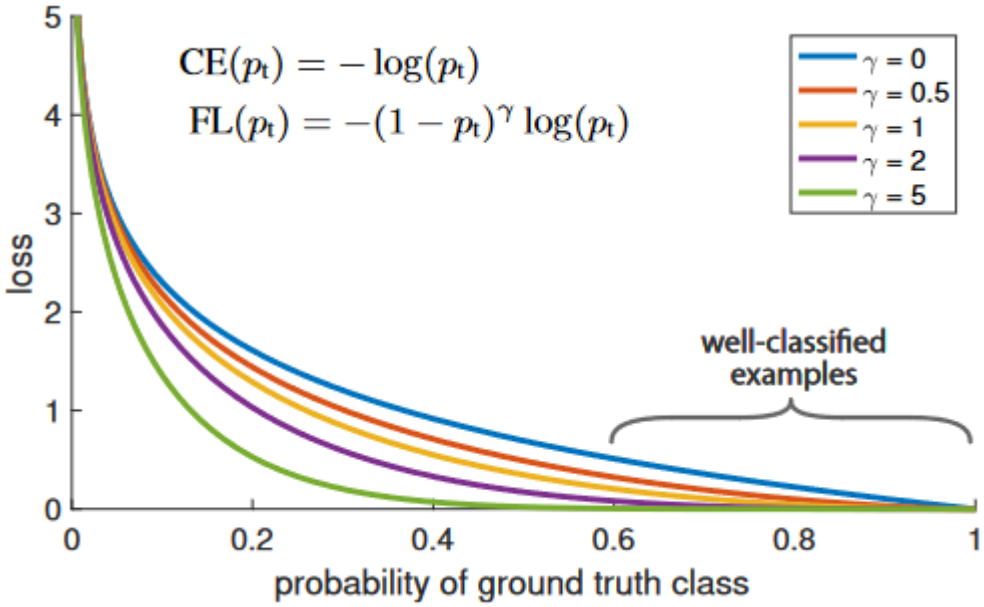
RetinaNet metoda usredotočuje se na jedan veliki nedostatak jednokoračnog pristupa, neuravnoteženost u reprezentaciji klasa. U većini slučajeva većinu površine slika ne zauzimaju objekti koje je potrebno detektirati već ju zauzima pozadina, što dovodi do neuravnoteženosti između pozadinske klase i ostalih klasa tj. pozadinska klasa ima veći utjecaj na izračun gubitaka u treniranju modela. U dvokoračnim metodama to ne predstavlja problem zato što mreža za prijedloge regija prvo pronalazi dijelove slika na kojima se traženi objekti nalaze i tek se na tim dijelovima odrađuju regresija i klasifikacija, uklanjajući većinu pozadine iz izračuna gubitaka. S druge strane, u jednokoračnim metodama situacija je potpuno suprotna. Takve metode obavljaju detekciju na čitavim slikama, što znači da pozadina čini većinu ulaznih podataka i ima najveći utjecaj izračun gubitaka. Kako bi se riješio taj problem, u RetinaNet metodi predstavljena je nova jednadžba (2) za izračun gubitaka nazvana žarišni gubitak (focal loss).

$$FL(p_t) = -(1 - p_t)^{\gamma} \log(p_t) \tag{2}$$

$\gamma \geq 0$ predstavlja podesivi koeficijent koji određuje utjecaj neravnoteže klasa na gubitak (Slika 14.), p_t je definiran jednadžbom (3):

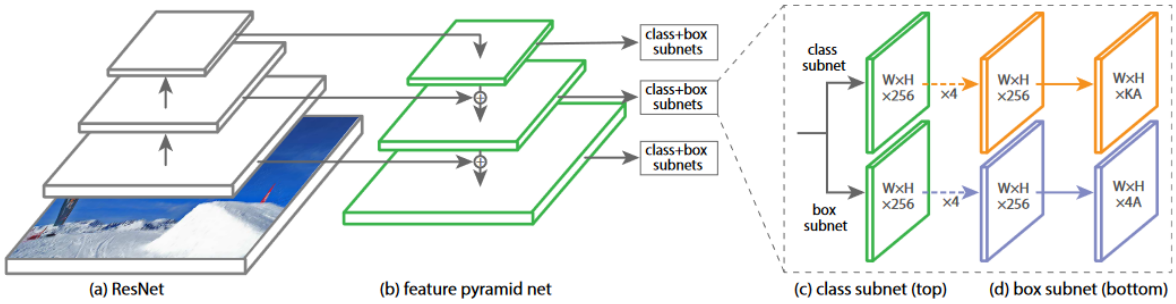
$$p_t = \begin{cases} p, & y = 1 \\ 1 - p, & y \neq 1 \end{cases} \tag{3}$$

gdje $y \in \{\pm 1\}$ predstavlja klasu temeljne istine, a $p \in [0,1]$ predstavlja vjerojatnost koju model procjenjuje za klasu s oznakom $y = 1$.



Slika 14. Utjecaj parametra γ na žarišni gubitak [11]

Princip rada žarišnog gubitka je da klase s kojima model ima više poteškoća imaju veći utjecaj na gubitak dok klase s kojima model nema poteškoća, prvenstveno pozadina, imaju manji utjecaj, rješavajući problem neuravnoteženosti klasa. RetinaNet model je u arhitekturi generalno nepromijenjen u odnosu na prethodno spomenute metode, za izvlačenje značajki koristi ResNet mrežu u kombinaciji sa FPN-om te koristi podmreže za regresiju i klasifikaciju. Slika 15. prikazuje arhitekturu RetinaNet mreže. Metoda je testirana na COCO bazi podataka, uspoređena je sa Faster R-CNN, YOLOv2 i SSD metodama. Za sve metode korištena je ResNet101 mreža za izvlačenje značajki, RetinaNet ostvaruje najveći *AP*. Ističe se *AP* na objektima male veličine, koji su prethodno jednokoračnim metodama predstavljale problem, gdje RetinaNet ostvaruje najbolje rezultate od 21,8%, drugi najbolji rezultat je Faster R-CNN od 18,2% [11].



Slika 15. Struktura RetinaNet mreže [11]

YOLOv3 metoda uvodi dodatna unaprijeđenja na YOLOv2 metodu s ciljem dodatnog poboljšanja preciznosti u detekciji. Dvije najveće promjene koje uvodi su predviđanje graničnih okvira na 3 različita razmjera, u usporedbi sa jednim koji se koristio u prethodnim verzijama, i nova mreža za izvlačenje značajki. Nova mreža sastoji se od 53 konvolucijska sloja (Slika 16.), u odnosu na 24 u prvoj verziji i 19 u drugoj, i uvodi FPN kako bi izvukla mape značajki raznih rezolucija kako bi metoda bila bolja u detekciji objekata raznolikih veličina. Metoda je testirana na COCO bazi podataka, iako daje bolje rezultate od prethodne YOLOv2 metode ima dosta lošije rezultate od RetinaNet metode [31].

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Slika 16. Mreža za izvlačenje značajki u YOLOv3 metodi [31]

4. EKSPERIMENT

Kako bi se utvrdila upotrebljivost metoda oba pristupa potrebno je provesti eksperiment koji uključuje treniranje i testiranje mreža na velikim bazama podatka. U sklopu ovog rada provesti će se eksperiment u kojem će se jedna dvokoračna metoda i jedna jednokoračna metoda trenirati na javno dostupnoj Pascal VOC bazi podataka za detekciju objekata.

U provođenju tog eksperimenta koristiti će se PyTorch biblioteka, odabrane su metode Faster R-CNN i RetinaNet te se koriste baze podataka Pascal VOC 2007 i 2012. Trening uključuje treniranje stopom učenja od 10^{-4} uz korištenje planera stope učenja koji će stopu učenja spuštati kada evaluacijski gubitak počne rasti, također je kreiran mehanizam ranijeg zaustavljanja koji će treniranje zaustaviti ako u 7 epoha model ne ostvari novi minimum evaluacijskog gubitka. Zbog navedene konfiguracije modeli će se jako kratko trenirati, razlog takvog pristupa su vremenska i hardverska ograničenja.

Radi navedenih ograničenja također se upotrebljavaju prethodno trenirani model dostupni u PyTorch biblioteci, koriste se Faster R-CNN i RetinaNet modeli sa Resnet50 mrežom za izvlačenje značajki sa FPN-om. Same mreže za detekciju su prethodno trenirane na COCO bazi podataka dok je ResNet mreža prethodno trenirana na ImageNet bazi podataka za klasifikaciju slika. Ostali parametri su također identični kod oba modela radi jednostavnije usporedbe, uključujući rezoluciju ulaznih slika koja je postavljena da manju dimenziju slike promijeni na 800 piksela, a drugoj dimenziji promijeni veličinu za isti faktor kako bi se očuvao omjer izvorne slike. Preuveličavanje slike je odrađeno bilinearnom interpolacijom.

Nakon treniranja modeli se testiraju, PyTorch ima ugrađenu klasu za pristup Pascal VOC bazama podataka no ne sadrži način testiranja izračunom *mAP* koji se tipično primjenjuje na Pascal VOC bazama podatka zbog čega je taj izračun ručno implementiran. Jednadžba (4) prikazuje način izračuna *mAP*:

$$mAP = \frac{1}{n}\sum_{i}^{n} AP_i \tag{4}$$

gdje *n* predstavlja broj klasa u bazi podatka, a AP_i predstavlja prosječnu preciznost za klasu *i* te se računa jednadžbom (5):

$$AP = \frac{1}{11}\sum_{r=0,0}^{1,0} p(r) \tag{5}$$

gdje *p(r)* predstavlja preciznost koja odgovara pokrivenosti *r* koja ima pomak od 0,1 po iteraciji, kako bi se dobila *p(r)* krivulja računaju se preciznost i pokrivenost za razne

vjerojatnosti klasa kao graničnih vrijednosti, od ≥ 95% do ≥ 5% sa razmakom od 5% po iteraciji, granična vrijednost omjera presjeka i unije je fiksirana na 0,5.

Konačno, nakon definiranja funkcije za izračun *mAP* obavlja se samo testiranje modela i izračun vremena predikcije iz kojeg se računa FPS. Za svaki provedeni test navode se *mAP* i brzina modela u FPS-u. Čitav eksperiment je odrađen na Nvidia A100 80GB SXM grafičkom procesoru. Sav kod napisan za eksperiment nalazi se u prilogu.

4.1. Pascal VOC 2007

Pascal VOC baza podataka iz 2007. na PyTorch-u sadrži već definirane baze podataka za treniranje, validaciju i testiranje. Baza podataka se sastoji od ukupno 9963 slika: 2501 slika za treniranje, 2510 slika za validaciju i 4952 slike za testiranje. Na tim slikama nalazi se ukupno 24640 instanci objekata 20 različitih klasa. Tablica 1. prikazuje ostvarene rezultate.

Tablica 1. Rezultati testiranja na Pascal VOC 2007 bazi podatka

Modeli	*mAP*(%)	FPS
RetinaNet	18,91	7,15
Faster R-CNN	33,09	5,69

4.2. Pascal VOC 2012

Za sve Pascal VOC baze podatka nakon 2007. temeljna istina za slike za testiranje nije učinjena javno dostupnom stoga je za Pascal VOC bazu podataka iz 2012. potrebno testiranje odraditi na dostupnim podacima za treniranje i validaciju. U tu svrhu, u PyTorch biblioteci učitana je baza podataka koja sadrži i podatke za treniranje i podatke za validaciju u jednom skupu te se dijele u podskupove za treniranje, validaciju i testiranje. Podjela skupa je ručno odrađena na način da udjeli skupova u ukupnom broju slika budu slični istima u bazi podataka iz 2007., također se izbjegava miješanje podataka prije podjele na podskupove kako bi se izbjegao utjecaj nasumičnosti u treniranjima modela što bi onemogućilo bilo kakvu usporedbu. Baza podataka se sastoji od ukupno 11540 slika podijeljenih na: 2880 slika za treniranje, 2890 slika za validaciju i 5770 slika za testiranje. Na tim slikama se nalazi ukupno 27450 instanci objekata istih 20 klasa kao i u bazi podataka iz 2007. Tablica 2. prikazuje ostvarene rezultate.

Tablica 2. Rezultati testiranja na Pascal VOC 2012 bazi podataka

Modeli	*mAP*(%)	FPS
RetinaNet	27,57	7,15
Faster R-CNN	27,46	5,74

4.3. Rezultati eksperimenta

Eksperimentom se pokazalo da jednokoračne metode mogu ostvariti preciznost detekcije koja konkurira dvokoračnim metodama, no ne bez žrtvovanja brzine. RetinaNet metoda ostvaruje rezultate komparabilne sa dvokoračnom Faster R-CNN metodom, no njena brzina je samo blago viša i niti približno dovoljno visoka za detekciju u stvarnom vremenu, gubeći glavnu prednost koju tipična jednokoračna metoda ima nad dvokoračnim metodama. U primjenama gdje je preciznost ključna najprikladnije su dvokoračne metode ili RetinaNet, a u primjenama gdje je ključna brzina poput detekcije u stvarnom vremenu ili ako je korisnik hardverski ograničen potrebno je koristiti jednokoračne metode kojima je efikasnost glavni fokus poput SSD ili YOLO metoda.

5. ZAKLJUČAK

Pojavom javno dostupnih baza podataka i napretkom hardvera dolazi do velikog porasta popularnosti metoda dubokog učenja u detekciji objekata. U neprestanom razvoju novih metoda ističu se dva temeljna pristupa: jednokoračni i dvokoračni.

U sklopu rada definirani su temeljni doprinosi prisutni u svakim modernim metodama dubokog učenja, opisane su temeljne razlike između dva pristupa te zatim identificirane i opisane razvijene metode kroz povijest za oba pristupa. Konačno, proveden je eksperiment kako bi se utvrdile prednosti i nedostatci navedenih pristupa te utvrdila njihova uporabljivost u različitim slučajevima uporabe treniranjem i testiranjem pripadajućih metoda na PascalVOC bazama podataka.

U eksperimentu, odabrana jednokoračna metoda ostvaruje preciznost usporedivu sa dvokoračnim metodama no ostvaruje i sličnu brzinu. Odabir prikladne metode svodi se na odabir između preciznosti i performansa. U slučajevima uporabe gdje je ključna preciznost najprikladnije su dvokoračne metode te u slučajevima uporabe gdje je ključna brzina najprikladnije su jednokoračne metode kojima je glavni fokus efikasnost.

LITERATURA

[1] Balasubramaniam, A., & Pasricha, S.: Object Detection in Autonomous Vehicles: Status and Open Challenges, arXiv, 2022.

[2] Laroca, R., Severo, E., Zanlorensi, L. A., Oliveira, L. S., Gonçalves, G. R., Schwartz, W. R., & Menotti, D.: A robust real-time automatic license plate recognition based on the YOLO detector, 2018 international joint conference on neural networks (ijcnn), IEEE, 2018.

[3] Malburg, L., Rieder, M. P., Seiger, R., Klein, P., & Bergmann, R.: Object detection for smart factory processes by machine learning, Procedia Computer Science, 2021.

[4] Lowe, D. G.: Distinctive image features from scale-invariant keypoints, International journal of computer vision, 2004.

[5] Dalal, N., & Triggs, B.: Histograms of oriented gradients for human detection, 2005 IEEE computer society conference on computer vision and pattern recognition, Ieee, 2005.

[6] Cortes, C., & Vapnik, V.: Support-vector networks, Machine learning, 1995.

[7] Freund, Y., & Schapire, R. E.: A desicion-theoretic generalization of on-line learning and an application to boosting, European conference on computational learning theory, Berlin, 1995.

[8] LeCun, Y., Bengio, Y., & Hinton, G.: Deep learning, nature, 2015.

[9] Girshick, R., Donahue, J., Darrell, T., & Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation, Proceedings of the IEEE conference on computer vision and pattern recognition, 2014.

[10] Ren, S., He, K., Girshick, R., & Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks, Advances in neural information processing systems, 2015.

[11] Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P.: Focal loss for dense object detection, Proceedings of the IEEE international conference on computer vision, 2017.

[12] Raina, R., Madhavan, A., & Ng, A. Y.: Large-scale deep unsupervised learning using graphics processors, Proceedings of the 26th annual international conference on machine learning, 2009.

[13] Harris, M.: Many-core GPU computing with NVIDIA CUDA, Proceedings of the 22nd annual international conference on Supercomputing, 2008.

[14] Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L.: Microsoft coco: Common objects in context, Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, Springer International Publishing, 2014.

[15] Everingham, M., Van Gool, L., Williams, C. K., Winn, J., & Zisserman, A.: The pascal visual object classes (voc) challenge, International journal of computer vision, 2010.

[16] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Zheng, X.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems, arXiv, 2016.

[17] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S.: Pytorch: An imperative style, high-performance deep learning library, Advances in neural information processing systems, 2019.

[18] Lin, T. Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S.: Feature pyramid networks for object detection, Proceedings of the IEEE conference on computer vision and pattern recognition, 2017.

[19] Girshick, R.: Fast r-cnn, Proceedings of the IEEE international conference on computer vision, 2015.

[20] Dai, J., Li, Y., He, K., & Sun, J.: R-fcn: Object detection via region-based fully convolutional networks, Advances in neural information processing systems, 2016.

[21] He, K., Gkioxari, G., Dollár, P., & Girshick, R.: Mask r-cnn, Proceedings of the IEEE international conference on computer vision, 2017.

[22] Uijlings, J. R., Van De Sande, K. E., Gevers, T., & Smeulders, A. W.: Selective search for object recognition, International journal of computer vision, 2013.

[23] Felzenszwalb, P. F., & Huttenlocher, D. P.: Efficient graph-based image segmentation, International journal of computer vision, 2004.

[24] Krizhevsky, A., Sutskever, I., & Hinton, G. E.: Imagenet classification with deep convolutional neural networks, Advances in neural information processing systems, 2012.

[25] Simonyan, K., & Zisserman, A.: Very deep convolutional networks for large-scale image recognition, arXiv, 2014.

[26] He, K., Zhang, X., Ren, S., & Sun, J.: Deep residual learning for image recognition, Proceedings of the IEEE conference on computer vision and pattern recognition, 2016.

[27] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A.: Going deeper with convolutions, Proceedings of the IEEE conference on computer vision and pattern recognition, 2015.

[28] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A.: You only look once: Unified, real-time object detection, Proceedings of the IEEE conference on computer vision and pattern recognition, 2016.

[29] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C.: Ssd: Single shot multibox detector, Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, 2016.

[30] Redmon, J., & Farhadi, A.: YOLO9000: better, faster, stronger, Proceedings of the IEEE conference on computer vision and pattern recognition, 2017.

[31] Redmon, J., & Farhadi, A.: Yolov3: An incremental improvement, arXiv, 2018.

[32] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Fei-Fei, L.: Imagenet large scale visual recognition challenge, International journal of computer vision, 2015.

PRILOZI

Kod za treniranje i testiranje RetinaNet modela na Pacal VOC 2007

```
EPOCHS = 100
BATCH_SIZE = 8
LEARNING_RATE=1e-4
PATIENCE = 7
MODEL_PATH = "./retinanet_voc_2007.pth"
NUM_WORKERS = 10
DATA_PATH = "./data"
TRAINABLE_BACKBONE_LAYERS = 5

import torch
import torchvision
from torchvision.transforms import transforms
from torchvision.datasets import VOCDetection
from torch.utils.data import DataLoader
import torch.optim as optim
import torch.optim.lr_scheduler as ls
import logging
import numpy as np
from tqdm import tqdm
from torchvision.models.detection import RetinaNet_ResNet50_FPN_V2_Weights
import math

device = 'cuda' if torch.cuda.is_available() else 'cpu'
device


VOC_CLASSES = (  # always index 0
    'aeroplane', 'bicycle', 'bird', 'boat',
    'bottle', 'bus', 'car', 'cat', 'chair',
    'cow', 'diningtable', 'dog', 'horse',
    'motorbike', 'person', 'pottedplant',
    'sheep', 'sofa', 'train', 'tvmonitor')

classes_dict = {'aeroplane': 1, 'bicycle': 2, 'bird': 3, 'boat': 4,
    'bottle': 5, 'bus': 6, 'car': 7, 'cat': 8, 'chair': 9,
    'cow': 10, 'diningtable': 11, 'dog': 12, 'horse': 13,
    'motorbike': 14, 'person': 15, 'pottedplant': 16,
    'sheep': 17, 'sofa': 18, 'train': 19, 'tvmonitor': 20}


transform = transforms.Compose([
    transforms.ToTensor(),
])

def collate_fn(batch):
    return list(zip(*batch))
```

```
def target_transform(targets_dict):
    bboxes = []
    labels = []

    for instance in targets_dict['annotation']['object']:

        bbox = [float(instance['bndbox']['xmin']),
                float(instance['bndbox']['ymin']),
                float(instance['bndbox']['xmax']),
                float(instance['bndbox']['ymax'])]
        bboxes.append(bbox)

        label = classes_dict[instance['name']]
        labels.append(label)

    bboxes = torch.tensor(bboxes, dtype=torch.float)
    labels = torch.tensor(labels, dtype=torch.int64)

    targets = {}
    targets['boxes'] = bboxes
    targets['labels'] = labels

    return(targets)

train_dataset = VOCDetection(root=DATA_PATH, year="2007", image_set="train",
download=True, transform=transform, target_transform=target_transform)
val_dataset = VOCDetection(root=DATA_PATH, year="2007", image_set="val",
download=True, transform=transform, target_transform=target_transform)
test_dataset = VOCDetection(root=DATA_PATH, year="2007", image_set="test",
download=True, transform=transform, target_transform=target_transform)


train_loader = DataLoader(train_dataset, batch_size=1, shuffle=False,
collate_fn=collate_fn, num_workers=NUM_WORKERS)
val_loader = DataLoader(val_dataset, batch_size=1, shuffle=False,
collate_fn=collate_fn, num_workers=NUM_WORKERS)
test_loader = DataLoader(test_dataset, batch_size=1, shuffle=False,
collate_fn=collate_fn, num_workers=NUM_WORKERS)

# Print sizes of all datasets
print(f'Size of train dataset: {len(train_loader)}')
print(f'Size of validation dataset: {len(val_loader)}')
print(f'Size of test dataset: {len(test_loader)}')

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True,
collate_fn=collate_fn, num_workers=NUM_WORKERS)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False,
collate_fn=collate_fn, num_workers=NUM_WORKERS)
```

```
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False,
collate_fn=collate_fn, num_workers=NUM_WORKERS)


for imgs, targets in train_loader:
    print(targets[0])
    break


model = torchvision.models.detection.retinanet_resnet50_fpn_v2(weights =
RetinaNet_ResNet50_FPN_V2_Weights.COCO_V1,
                                                               trainable_backb
one_layers=TRAINABLE_BACKBONE_LAYERS)

num_classes = 21
out_channels = model.head.classification_head.conv[0].out_channels
num_anchors = model.head.classification_head.num_anchors
model.head.classification_head.num_classes = num_classes

cls_logits = torch.nn.Conv2d(out_channels, num_anchors * num_classes,
kernel_size=3, stride=1, padding=1)
torch.nn.init.normal_(cls_logits.weight, std=0.01)  # as per pytorch code
torch.nn.init.constant_(cls_logits.bias, -math.log((1 - 0.01) / 0.01))  # as
per pytorcch code
# assign cls head to model
model.head.classification_head.cls_logits = cls_logits

model = model.to(device)


optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE, weight_decay=1e-
5)
scheduler = ls.ReduceLROnPlateau(optimizer, 'min', patience=2)


class EarlyStopping:
    """Early stops the training if validation loss doesn't improve after a
given patience."""

    def __init__(self, path, patience: int = 7, min_delta: float = 0,
trace_func=print):
        """
        Args:
            patience (int): How long to wait after last time validation loss
improved.
                            Default: 7
            min_delta (float): Minimum change in the monitored quantity to
qualify as an improvement.
                            Default: 0
            path (str): Path for the model to be saved to.
            trace_func (function): trace print function.
```

```
                            Default: print
        """
        self.counter = 0
        self.min_loss = None
        self.early_stop = False
        self.val_loss_min = np.Inf
        self.path = path
        self.trace_func = trace_func

        if not callable(trace_func):
            raise TypeError("Argument trace_func should be a function")

        if patience < 1:
            raise ValueError("Argument patience should be positive integer")
        self.patience = patience

        if min_delta < 0.0:
            raise ValueError(
                "Argument min_delta should not be a negative number")
        self.delta = min_delta

        logging.info('EarlyStopping(), path=%s patience=%d min_delta=%f',
                     path, patience, min_delta)

    def __call__(self, loss, model):
        '''Check whether early stopping is needed '''
        print("loss:", loss, "min_loss:", self.min_loss)
        if self.min_loss is None:
            self.min_loss = loss
            self.save_checkpoint(loss, model)
        elif loss >= self.min_loss - self.delta:
            self.counter += 1
            self.trace_func(
                f'EarlyStopping counter: {self.counter} out of
{self.patience}')
            if self.counter >= self.patience:
                self.early_stop = True
        else:
            self.min_loss = loss
            self.save_checkpoint(loss, model)
            self.counter = 0

    def save_checkpoint(self, val_loss, model):
        '''Saves model when validation loss decreases.'''
        logging.info('save_checkpoint(), loss=%f', val_loss)
        torch.save(model, self.path)


def train_fn():
```

```
    global iteration
    total_loss = 0.0
    model.train()
    for data in tqdm(train_loader, desc="train   "):
        imgs, targets = data

        imgs = list(image.to(device) for image in imgs)
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

        losses = model(imgs, targets)

        loss = sum(loss for loss in losses.values())
        total_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    return total_loss/len(train_loader)


@torch.no_grad()
def eval_fn():
    total_loss = 0.0
    model.train()
    for data in tqdm(val_loader, desc="validate"):
        imgs, targets = data
        imgs = list(image.to(device) for image in imgs)
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]
        losses = model(imgs, targets)
        loss = sum(l for l in losses.values())
        total_loss += loss.item()

    return total_loss/len(val_loader)


def train(model):
    train_losses = []
    valid_losses = []
    early_stopping = EarlyStopping(MODEL_PATH, patience=PATIENCE,
min_delta=0.0)

    for i in range(0, EPOCHS):
        train_loss = train_fn()
        train_losses.append(train_loss)
        valid_loss = eval_fn()
        valid_losses.append(valid_loss)

        scheduler.step(valid_loss)
        early_stopping(valid_loss, model)
```

```
        if early_stopping.early_stop:
            print("early stopping ...")
            break

    return model, train_losses, valid_losses


torch.cuda.empty_cache()


model, tlosses, vlosses = train(model)


def calculate_iou(pred_box, gt_box):
    """
    Calculate intersection over union (IoU) between two bounding boxes.
    """
    # Calculate intersection area
    x_left = max(pred_box[0], gt_box[0])
    y_top = max(pred_box[1], gt_box[1])
    x_right = min(pred_box[2], gt_box[2])
    y_bottom = min(pred_box[3], gt_box[3])

    intersection_area = max(0, x_right - x_left + 1) * max(0, y_bottom - y_top
+ 1)

    # Calculate union area
    pred_area = (pred_box[2] - pred_box[0] + 1) * (pred_box[3] - pred_box[1] +
1)
    gt_area = (gt_box[2] - gt_box[0] + 1) * (gt_box[3] - gt_box[1] + 1)
    union_area = pred_area + gt_area - intersection_area

    # Calculate IoU
    iou = intersection_area / union_area

    return iou

def compute_ap(precisions, recalls):
    """
    Compute Average Precision (AP) from precision and recall lists.
    """
    # Add edge values to prevent out of range errors
    precisions = np.concatenate(([1], precisions, [0]))
    recalls = np.concatenate(([0], recalls, [1]))

    recall_levels = np.linspace(0, 1, 11)
    interpolated_precisions = []

    for recall_level in recall_levels:
```

```
        # Find the highest recall value less than or equal to the current
recall level
        recall_mask = recalls >= recall_level
        if np.any(recall_mask):
            precision_at_recall_level = np.max(precisions[recall_mask])
            interpolated_precisions.append(precision_at_recall_level)

    # Compute the average interpolated precision
    average_precision = np.mean(interpolated_precisions)

    return average_precision

def compute_map(data, iou_threshold=0.5):
    """
    Compute Mean Average Precision (mAP) for object detection.
    """
    average_precisions = []

    for class_idx in tqdm(range(1, 21), desc='Calculating mAP'):
        # Initiate per score precision and recall lists for AP calculation
        per_score_precision = np.zeros(19)
        per_score_recall = np.zeros(19)

        for i, score in enumerate(range(95, 0, -5)):
            true_positives = 0
            false_positives = 0
            total_class_gt = 0

            for img_dict in data:
                predictions = img_dict['predictions']
                ground_truth = img_dict['ground_truth']

                # Get predictions and ground_truth for the current class
                class_predictions = predictions[(predictions[:, 5] ==
class_idx) & (predictions[:, 4] >= score/100)]
                class_gt = ground_truth[ground_truth[:, 4] == class_idx]

                total_class_gt += len(class_gt)
                detected_gt_boxes = []

                for pred in class_predictions:
                    pred_box = pred[:4]
                    max_iou = 0

                    for i, gt in enumerate(class_gt):
                        gt_box = gt[:4]
                        iou = calculate_iou(pred_box, gt_box)

                        if iou > max_iou:
```

```
                            max_iou = iou
                            best_gt_idx = i

                    if max_iou > iou_threshold and best_gt_idx not in
detected_gt_boxes:
                        true_positives += 1
                        detected_gt_boxes.append(best_gt_idx)
                    else:
                        false_positives += 1

            per_score_precision[i] = true_positives / (true_positives +
false_positives + 1e-16)
            per_score_recall[i] = true_positives / (total_class_gt + 1e-16)

        # Compute AP
        ap = compute_ap(per_score_precision, per_score_recall)
        average_precisions.append(ap)

    print(average_precisions)

    # Compute mAP
    mAP = np.mean(average_precisions)

    return mAP


@torch.no_grad()
def test_fn():
    model.eval()
    all_data = []
    for data in tqdm(test_loader, desc="testing"):
        imgs, targets = data
        imgs = list(image.to(device) for image in imgs)
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]
        batch_predictions = model(imgs)

        for img_predictions, img_ground_truth in zip(batch_predictions,
targets):
            img_dict = {'predictions': [], 'ground_truth': []}

            for prediction in zip(*img_predictions.values()):
                prediction_tensor = torch.cat((prediction[0],
prediction[1].reshape(1), prediction[2].reshape(1)), dim=0)
                img_dict['predictions'].append(prediction_tensor)

            for ground_truth in zip(*img_ground_truth.values()):
                gt_tensor = torch.cat((ground_truth[0],
ground_truth[1].reshape(1)), dim=0)
                img_dict['ground_truth'].append(gt_tensor)
```

```
            if len(img_dict['predictions']) == 0:
img_dict['predictions'].append(torch.tensor([0, 0, 0, 0, 0, 0]))

            img_dict['predictions'] = torch.stack(img_dict['predictions'])
            img_dict['ground_truth'] = torch.stack(img_dict['ground_truth'])
            all_data.append(img_dict)

    return all_data


torch.cuda.empty_cache()
model = torch.load(MODEL_PATH)

timings=np.zeros(len(test_loader))

model.eval()

start_event = torch.cuda.Event(enable_timing=True)
end_event = torch.cuda.Event(enable_timing=True)

with torch.no_grad():
    for i, data in enumerate(test_loader):
        imgs, _ = data
        imgs = list(image.to(device) for image in imgs)
        start_event.record()
        output = model(imgs)
        end_event.record()
        torch.cuda.synchronize()
        current_time = start_event.elapsed_time(end_event)
        timings[i] = current_time

# Calculate inference time and frames per second
inference_time = np.sum(timings) / len(test_loader) / 1000.0 # Convert to
seconds
FPS = 1 / inference_time
print("Inference time:", inference_time)
print("FPS:", FPS)


model = torch.load(MODEL_PATH)

data = test_fn()

mAP = compute_map(data)
print(mAP)
```

Kod za treniranje i testiranje RetinaNet modela na Pacal VOC 2012

```
EPOCHS = 100
BATCH_SIZE = 8
LEARNING_RATE=1e-4
PATIENCE = 7
MODEL_PATH = "./retinanet_voc_2012.pth"
NUM_WORKERS = 10
DATA_PATH = "./data"
TRAINABLE_BACKBONE_LAYERS = 5


import torch
import torchvision
from torchvision.transforms import transforms
from torchvision.datasets import VOCDetection
from torch.utils.data import DataLoader
import torch.optim as optim
import torch.optim.lr_scheduler as ls
import logging
import numpy as np
from tqdm import tqdm
from torchvision.models.detection import RetinaNet_ResNet50_FPN_V2_Weights
import math

device = 'cuda' if torch.cuda.is_available() else 'cpu'
device


VOC_CLASSES = (  # always index 0
    'aeroplane', 'bicycle', 'bird', 'boat',
    'bottle', 'bus', 'car', 'cat', 'chair',
    'cow', 'diningtable', 'dog', 'horse',
    'motorbike', 'person', 'pottedplant',
    'sheep', 'sofa', 'train', 'tvmonitor')

classes_dict = {'aeroplane': 1, 'bicycle': 2, 'bird': 3, 'boat': 4,
    'bottle': 5, 'bus': 6, 'car': 7, 'cat': 8, 'chair': 9,
    'cow': 10, 'diningtable': 11, 'dog': 12, 'horse': 13,
    'motorbike': 14, 'person': 15, 'pottedplant': 16,
    'sheep': 17, 'sofa': 18, 'train': 19, 'tvmonitor': 20}


transform = transforms.Compose([
    transforms.ToTensor(),
])

def collate_fn(batch):
    return list(zip(*batch))

def target_transform(targets_dict):
```

```
    bboxes = []
    labels = []

    for instance in targets_dict['annotation']['object']:

        bbox = [float(instance['bndbox']['xmin']),
                float(instance['bndbox']['ymin']),
                float(instance['bndbox']['xmax']),
                float(instance['bndbox']['ymax'])]
        bboxes.append(bbox)

        label = classes_dict[instance['name']]
        labels.append(label)

    bboxes = torch.tensor(bboxes, dtype=torch.float)
    labels = torch.tensor(labels, dtype=torch.int64)

    targets = {}
    targets['boxes'] = bboxes
    targets['labels'] = labels

    return(targets)

test_dataset = VOCDetection(root=DATA_PATH, year="2012", image_set="trainval",
download=True, transform=transform, target_transform=target_transform)

# Split the dataset into train, validation and train datasets with a similar
split to Pascal VOC 2007
train_indices = range(2880)
val_indices = range(2880, 5770)
test_indices = range(5770, 11540)

train_dataset = torch.utils.data.Subset(test_dataset, train_indices)
val_dataset = torch.utils.data.Subset(test_dataset, val_indices)
test_dataset = torch.utils.data.Subset(test_dataset, test_indices)


train_loader = DataLoader(train_dataset, batch_size=1, shuffle=False,
collate_fn=collate_fn, num_workers=NUM_WORKERS)
val_loader = DataLoader(val_dataset, batch_size=1, shuffle=False,
collate_fn=collate_fn, num_workers=NUM_WORKERS)
test_loader = DataLoader(test_dataset, batch_size=1, shuffle=False,
collate_fn=collate_fn, num_workers=NUM_WORKERS)

# Print sizes of all datasets
print(f'Size of train dataset: {len(train_loader)}')
print(f'Size of validation dataset: {len(val_loader)}')
print(f'Size of test dataset: {len(test_loader)}')
```

```
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True,
collate_fn=collate_fn, num_workers=NUM_WORKERS)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False,
collate_fn=collate_fn, num_workers=NUM_WORKERS)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False,
collate_fn=collate_fn, num_workers=NUM_WORKERS)


for imgs, targets in train_loader:
    print(targets[0])
    break


model = torchvision.models.detection.retinanet_resnet50_fpn_v2(weights =
RetinaNet_ResNet50_FPN_V2_Weights.COCO_V1,
                                                               trainable_backb
one_layers=TRAINABLE_BACKBONE_LAYERS)

num_classes = 21
out_channels = model.head.classification_head.conv[0].out_channels
num_anchors = model.head.classification_head.num_anchors
model.head.classification_head.num_classes = num_classes

cls_logits = torch.nn.Conv2d(out_channels, num_anchors * num_classes,
kernel_size=3, stride=1, padding=1)
torch.nn.init.normal_(cls_logits.weight, std=0.01)  # as per pytorch code
torch.nn.init.constant_(cls_logits.bias, -math.log((1 - 0.01) / 0.01))  # as
per pytorcch code
# assign cls head to model
model.head.classification_head.cls_logits = cls_logits

model = model.to(device)


optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE, weight_decay=1e-
5)
scheduler = ls.ReduceLROnPlateau(optimizer, 'min', patience=2)


class EarlyStopping:
    """Early stops the training if validation loss doesn't improve after a
given patience."""

    def __init__(self, path, patience: int = 7, min_delta: float = 0,
trace_func=print):
        """
        Args:
            patience (int): How long to wait after last time validation loss
improved.
                            Default: 7
```

```
            min_delta (float): Minimum change in the monitored quantity to
qualify as an improvement.
                               Default: 0
            path (str): Path for the model to be saved to.
            trace_func (function): trace print function.
                               Default: print
        """
        self.counter = 0
        self.min_loss = None
        self.early_stop = False
        self.val_loss_min = np.Inf
        self.path = path
        self.trace_func = trace_func

        if not callable(trace_func):
            raise TypeError("Argument trace_func should be a function")

        if patience < 1:
            raise ValueError("Argument patience should be positive integer")
        self.patience = patience

        if min_delta < 0.0:
            raise ValueError(
                "Argument min_delta should not be a negative number")
        self.delta = min_delta

        logging.info('EarlyStopping(), path=%s patience=%d min_delta=%f',
                     path, patience, min_delta)

    def __call__(self, loss, model):
        '''Check whether early stopping is needed '''
        print("loss:", loss, "min_loss:", self.min_loss)
        if self.min_loss is None:
            self.min_loss = loss
            self.save_checkpoint(loss, model)
        elif loss >= self.min_loss - self.delta:
            self.counter += 1
            self.trace_func(
                f'EarlyStopping counter: {self.counter} out of
{self.patience}')
            if self.counter >= self.patience:
                self.early_stop = True
        else:
            self.min_loss = loss
            self.save_checkpoint(loss, model)
            self.counter = 0

    def save_checkpoint(self, val_loss, model):
        '''Saves model when validation loss decreases.'''
```

```
        logging.info('save_checkpoint(), loss=%f', val_loss)
        torch.save(model, self.path)


def train_fn():
    global iteration
    total_loss = 0.0
    model.train()
    for data in tqdm(train_loader, desc="train   "):
        imgs, targets = data

        imgs = list(image.to(device) for image in imgs)
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

        losses = model(imgs, targets)

        loss = sum(loss for loss in losses.values())
        total_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    return total_loss/len(train_loader)


@torch.no_grad()
def eval_fn():
    total_loss = 0.0
    model.train()
    for data in tqdm(val_loader, desc="validate"):
        imgs, targets = data
        imgs = list(image.to(device) for image in imgs)
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]
        losses = model(imgs, targets)
        loss = sum(l for l in losses.values())
        total_loss += loss.item()

    return total_loss/len(val_loader)


def train(model):
    train_losses = []
    valid_losses = []
    early_stopping = EarlyStopping(MODEL_PATH, patience=PATIENCE,
min_delta=0.0)

    for i in range(0, EPOCHS):
        train_loss = train_fn()
        train_losses.append(train_loss)
```

```
        valid_loss = eval_fn()
        valid_losses.append(valid_loss)

        scheduler.step(valid_loss)
        early_stopping(valid_loss, model)
        if early_stopping.early_stop:
            print("early stopping ...")
            break

    return model, train_losses, valid_losses


torch.cuda.empty_cache()


model, tlosses, vlosses = train(model)


def calculate_iou(pred_box, gt_box):
    """
    Calculate intersection over union (IoU) between two bounding boxes.
    """
    # Calculate intersection area
    x_left = max(pred_box[0], gt_box[0])
    y_top = max(pred_box[1], gt_box[1])
    x_right = min(pred_box[2], gt_box[2])
    y_bottom = min(pred_box[3], gt_box[3])

    intersection_area = max(0, x_right - x_left + 1) * max(0, y_bottom - y_top
+ 1)

    # Calculate union area
    pred_area = (pred_box[2] - pred_box[0] + 1) * (pred_box[3] - pred_box[1] +
1)
    gt_area = (gt_box[2] - gt_box[0] + 1) * (gt_box[3] - gt_box[1] + 1)
    union_area = pred_area + gt_area - intersection_area

    # Calculate IoU
    iou = intersection_area / union_area

    return iou

def compute_ap(precisions, recalls):
    """
    Compute Average Precision (AP) from precision and recall lists.
    """
    # Add edge values to prevent out of range errors
    precisions = np.concatenate(([1], precisions, [0]))
    recalls = np.concatenate(([0], recalls, [1]))
```

```
    recall_levels = np.linspace(0, 1, 11)
    interpolated_precisions = []

    for recall_level in recall_levels:
        # Find the highest recall value less than or equal to the current
recall level
        recall_mask = recalls >= recall_level
        if np.any(recall_mask):
            precision_at_recall_level = np.max(precisions[recall_mask])
            interpolated_precisions.append(precision_at_recall_level)

    # Compute the average interpolated precision
    average_precision = np.mean(interpolated_precisions)

    return average_precision

def compute_map(data, iou_threshold=0.5):
    """
    Compute Mean Average Precision (mAP) for object detection.
    """
    average_precisions = []

    for class_idx in tqdm(range(1, 21), desc='Calculating mAP'):
        # Initiate per score precision and recall lists for AP calculation
        per_score_precision = np.zeros(19)
        per_score_recall = np.zeros(19)

        for i, score in enumerate(range(95, 0, -5)):
            true_positives = 0
            false_positives = 0
            total_class_gt = 0

            for img_dict in data:
                predictions = img_dict['predictions']
                ground_truth = img_dict['ground_truth']

                # Get predictions and ground_truth for the current class
                class_predictions = predictions[(predictions[:, 5] ==
class_idx) & (predictions[:, 4] >= score/100)]
                class_gt = ground_truth[ground_truth[:, 4] == class_idx]

                total_class_gt += len(class_gt)
                detected_gt_boxes = []

                for pred in class_predictions:
                    pred_box = pred[:4]
                    max_iou = 0

                    for i, gt in enumerate(class_gt):
```

```
                        gt_box = gt[:4]
                        iou = calculate_iou(pred_box, gt_box)

                        if iou > max_iou:
                            max_iou = iou
                            best_gt_idx = i

                    if max_iou > iou_threshold and best_gt_idx not in
detected_gt_boxes:
                        true_positives += 1
                        detected_gt_boxes.append(best_gt_idx)
                    else:
                        false_positives += 1

            per_score_precision[i] = true_positives / (true_positives +
false_positives + 1e-16)
            per_score_recall[i] = true_positives / (total_class_gt + 1e-16)

        # Compute AP
        ap = compute_ap(per_score_precision, per_score_recall)
        average_precisions.append(ap)

    print(average_precisions)

    # Compute mAP
    mAP = np.mean(average_precisions)

    return mAP


@torch.no_grad()
def test_fn():
    model.eval()
    all_data = []
    for data in tqdm(test_loader, desc="testing"):
        imgs, targets = data
        imgs = list(image.to(device) for image in imgs)
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]
        batch_predictions = model(imgs)

        for img_predictions, img_ground_truth in zip(batch_predictions,
targets):
            img_dict = {'predictions': [], 'ground_truth': []}

            for prediction in zip(*img_predictions.values()):
                prediction_tensor = torch.cat((prediction[0],
prediction[1].reshape(1), prediction[2].reshape(1)), dim=0)
                img_dict['predictions'].append(prediction_tensor)
```

```
            for ground_truth in zip(*img_ground_truth.values()):
                gt_tensor = torch.cat((ground_truth[0],
ground_truth[1].reshape(1)), dim=0)
                img_dict['ground_truth'].append(gt_tensor)

            if len(img_dict['predictions']) == 0:
img_dict['predictions'].append(torch.tensor([0, 0, 0, 0, 0, 0]))

            img_dict['predictions'] = torch.stack(img_dict['predictions'])
            img_dict['ground_truth'] = torch.stack(img_dict['ground_truth'])
            all_data.append(img_dict)

    return all_data


torch.cuda.empty_cache()
model = torch.load(MODEL_PATH)

timings=np.zeros(len(test_loader))

model.eval()

start_event = torch.cuda.Event(enable_timing=True)
end_event = torch.cuda.Event(enable_timing=True)

with torch.no_grad():
    for i, data in enumerate(test_loader):
        imgs, _ = data
        imgs = list(image.to(device) for image in imgs)
        start_event.record()
        output = model(imgs)
        end_event.record()
        torch.cuda.synchronize()
        current_time = start_event.elapsed_time(end_event)
        timings[i] = current_time

# Calculate inference time and frames per second
inference_time = np.sum(timings) / len(test_loader) / 1000.0 # Convert to
seconds
FPS = 1 / inference_time
print("Inference time:", inference_time)
print("FPS:", FPS)


model = torch.load(MODEL_PATH)

data = test_fn()

mAP = compute_map(data)
print(mAP)
```

Kod za treniranje i testiranje Faster R-CNN modela na Pacal VOC 2007

```
EPOCHS = 100
BATCH_SIZE = 8
LEARNING_RATE=1e-4
PATIENCE = 7
MODEL_PATH = "./faster_rcnn_voc_2007.pth"
NUM_WORKERS = 10
DATA_PATH = "./data"
TRAINABLE_BACKBONE_LAYERS = 5


import torch
import torchvision
from torchvision.transforms import transforms
from torchvision.datasets import VOCDetection
from torch.utils.data import DataLoader
import torch.optim as optim
import torch.optim.lr_scheduler as ls
import logging
import numpy as np
from tqdm import tqdm
from torchvision.models.detection import FasterRCNN_ResNet50_FPN_V2_Weights
import math

device = 'cuda' if torch.cuda.is_available() else 'cpu'
device


VOC_CLASSES = (  # always index 0
    'aeroplane', 'bicycle', 'bird', 'boat',
    'bottle', 'bus', 'car', 'cat', 'chair',
    'cow', 'diningtable', 'dog', 'horse',
    'motorbike', 'person', 'pottedplant',
    'sheep', 'sofa', 'train', 'tvmonitor')

classes_dict = {'aeroplane': 1, 'bicycle': 2, 'bird': 3, 'boat': 4,
    'bottle': 5, 'bus': 6, 'car': 7, 'cat': 8, 'chair': 9,
    'cow': 10, 'diningtable': 11, 'dog': 12, 'horse': 13,
    'motorbike': 14, 'person': 15, 'pottedplant': 16,
    'sheep': 17, 'sofa': 18, 'train': 19, 'tvmonitor': 20}


transform = transforms.Compose([
    transforms.ToTensor(),
])

def collate_fn(batch):
    return list(zip(*batch))

def target_transform(targets_dict):
```

```
    bboxes = []
    labels = []

    for instance in targets_dict['annotation']['object']:

        bbox = [float(instance['bndbox']['xmin']),
                float(instance['bndbox']['ymin']),
                float(instance['bndbox']['xmax']),
                float(instance['bndbox']['ymax'])]
        bboxes.append(bbox)

        label = classes_dict[instance['name']]
        labels.append(label)

    bboxes = torch.tensor(bboxes, dtype=torch.float)
    labels = torch.tensor(labels, dtype=torch.int64)

    targets = {}
    targets['boxes'] = bboxes
    targets['labels'] = labels

    return(targets)

train_dataset = VOCDetection(root=DATA_PATH, year="2007", image_set="train",
download=True, transform=transform, target_transform=target_transform)
val_dataset = VOCDetection(root=DATA_PATH, year="2007", image_set="val",
download=True, transform=transform, target_transform=target_transform)
test_dataset = VOCDetection(root=DATA_PATH, year="2007", image_set="test",
download=True, transform=transform, target_transform=target_transform)


train_loader = DataLoader(train_dataset, batch_size=1, shuffle=False,
collate_fn=collate_fn, num_workers=NUM_WORKERS)
val_loader = DataLoader(val_dataset, batch_size=1, shuffle=False,
collate_fn=collate_fn, num_workers=NUM_WORKERS)
test_loader = DataLoader(test_dataset, batch_size=1, shuffle=False,
collate_fn=collate_fn, num_workers=NUM_WORKERS)

# Print sizes of all datasets
print(f'Size of train dataset: {len(train_loader)}')
print(f'Size of validation dataset: {len(val_loader)}')
print(f'Size of test dataset: {len(test_loader)}')

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True,
collate_fn=collate_fn, num_workers=NUM_WORKERS)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False,
collate_fn=collate_fn, num_workers=NUM_WORKERS)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False,
collate_fn=collate_fn, num_workers=NUM_WORKERS)
```

```
for imgs, targets in train_loader:
    print(targets[0])
    break


model =
torchvision.models.detection.fasterrcnn_resnet50_fpn_v2(weights=FasterRCNN_Res
Net50_FPN_V2_Weights.COCO_V1,
                                                         trainable_back
bone_layers=TRAINABLE_BACKBONE_LAYERS)

num_classes = 21
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
in_features = model.roi_heads.box_predictor.cls_score.in_features
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

model = model.to(device)


optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE, weight_decay=1e-
5)
scheduler = ls.ReduceLROnPlateau(optimizer, 'min', patience=2)


class EarlyStopping:
    """Early stops the training if validation loss doesn't improve after a
given patience."""

    def __init__(self, path, patience: int = 7, min_delta: float = 0,
trace_func=print):
        """
        Args:
            patience (int): How long to wait after last time validation loss
improved.
                            Default: 7
            min_delta (float): Minimum change in the monitored quantity to
qualify as an improvement.
                            Default: 0
            path (str): Path for the model to be saved to.
            trace_func (function): trace print function.
                            Default: print
        """
        self.counter = 0
        self.min_loss = None
        self.early_stop = False
        self.val_loss_min = np.Inf
        self.path = path
        self.trace_func = trace_func

        if not callable(trace_func):
```

```
            raise TypeError("Argument trace_func should be a function")

        if patience < 1:
            raise ValueError("Argument patience should be positive integer")
        self.patience = patience

        if min_delta < 0.0:
            raise ValueError(
                "Argument min_delta should not be a negative number")
        self.delta = min_delta

        logging.info('EarlyStopping(), path=%s patience=%d min_delta=%f',
                     path, patience, min_delta)

    def __call__(self, loss, model):
        '''Check whether early stopping is needed '''
        print("loss:", loss, "min_loss:", self.min_loss)
        if self.min_loss is None:
            self.min_loss = loss
            self.save_checkpoint(loss, model)
        elif loss >= self.min_loss - self.delta:
            self.counter += 1
            self.trace_func(
                f'EarlyStopping counter: {self.counter} out of
{self.patience}')
            if self.counter >= self.patience:
                self.early_stop = True
        else:
            self.min_loss = loss
            self.save_checkpoint(loss, model)
            self.counter = 0

    def save_checkpoint(self, val_loss, model):
        '''Saves model when validation loss decreases.'''
        logging.info('save_checkpoint(), loss=%f', val_loss)
        torch.save(model, self.path)


def train_fn():
    global iteration
    total_loss = 0.0
    model.train()
    for data in tqdm(train_loader, desc="train   "):
        imgs, targets = data

        imgs = list(image.to(device) for image in imgs)
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

        losses = model(imgs, targets)
```

```
        loss = sum(loss for loss in losses.values())
        total_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    return total_loss/len(train_loader)


@torch.no_grad()
def eval_fn():
    total_loss = 0.0
    model.train()
    for data in tqdm(val_loader, desc="validate"):
        imgs, targets = data
        imgs = list(image.to(device) for image in imgs)
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]
        losses = model(imgs, targets)
        loss = sum(l for l in losses.values())
        total_loss += loss.item()

    return total_loss/len(val_loader)


def train(model):
    train_losses = []
    valid_losses = []
    early_stopping = EarlyStopping(MODEL_PATH, patience=PATIENCE,
min_delta=0.0)

    for i in range(0, EPOCHS):
        train_loss = train_fn()
        train_losses.append(train_loss)
        valid_loss = eval_fn()
        valid_losses.append(valid_loss)

        scheduler.step(valid_loss)
        early_stopping(valid_loss, model)
        if early_stopping.early_stop:
            print("early stopping ...")
            break

    return model, train_losses, valid_losses


torch.cuda.empty_cache()


model, tlosses, vlosses = train(model)
```

```
def calculate_iou(pred_box, gt_box):
    """
    Calculate intersection over union (IoU) between two bounding boxes.
    """
    # Calculate intersection area
    x_left = max(pred_box[0], gt_box[0])
    y_top = max(pred_box[1], gt_box[1])
    x_right = min(pred_box[2], gt_box[2])
    y_bottom = min(pred_box[3], gt_box[3])

    intersection_area = max(0, x_right - x_left + 1) * max(0, y_bottom - y_top
+ 1)

    # Calculate union area
    pred_area = (pred_box[2] - pred_box[0] + 1) * (pred_box[3] - pred_box[1] +
1)
    gt_area = (gt_box[2] - gt_box[0] + 1) * (gt_box[3] - gt_box[1] + 1)
    union_area = pred_area + gt_area - intersection_area

    # Calculate IoU
    iou = intersection_area / union_area

    return iou

def compute_ap(precisions, recalls):
    """
    Compute Average Precision (AP) from precision and recall lists.
    """
    # Add edge values to prevent out of range errors
    precisions = np.concatenate(([1], precisions, [0]))
    recalls = np.concatenate(([0], recalls, [1]))

    recall_levels = np.linspace(0, 1, 11)
    interpolated_precisions = []

    for recall_level in recall_levels:
        # Find the highest recall value less than or equal to the current
recall level
        recall_mask = recalls >= recall_level
        if np.any(recall_mask):
            precision_at_recall_level = np.max(precisions[recall_mask])
            interpolated_precisions.append(precision_at_recall_level)

    # Compute the average interpolated precision
    average_precision = np.mean(interpolated_precisions)

    return average_precision

def compute_map(data, iou_threshold=0.5):
```

```
    """
    Compute Mean Average Precision (mAP) for object detection.
    """
    average_precisions = []

    for class_idx in tqdm(range(1, 21), desc='Calculating mAP'):
        # Initiate per score precision and recall lists for AP calculation
        per_score_precision = np.zeros(19)
        per_score_recall = np.zeros(19)

        for i, score in enumerate(range(95, 0, -5)):
            true_positives = 0
            false_positives = 0
            total_class_gt = 0

            for img_dict in data:
                predictions = img_dict['predictions']
                ground_truth = img_dict['ground_truth']

                # Get predictions and ground_truth for the current class
                class_predictions = predictions[(predictions[:, 4] ==
class_idx) & (predictions[:, 5] >= score/100)]
                class_gt = ground_truth[ground_truth[:, 4] == class_idx]

                total_class_gt += len(class_gt)
                detected_gt_boxes = []

                for pred in class_predictions:
                    pred_box = pred[:4]
                    max_iou = 0

                    for i, gt in enumerate(class_gt):
                        gt_box = gt[:4]
                        iou = calculate_iou(pred_box, gt_box)

                        if iou > max_iou:
                            max_iou = iou
                            best_gt_idx = i

                    if max_iou > iou_threshold and best_gt_idx not in
detected_gt_boxes:
                        true_positives += 1
                        detected_gt_boxes.append(best_gt_idx)
                    else:
                        false_positives += 1

            per_score_precision[i] = true_positives / (true_positives +
false_positives + 1e-16)
            per_score_recall[i] = true_positives / (total_class_gt + 1e-16)
```

```
        # Compute AP
        ap = compute_ap(per_score_precision, per_score_recall)
        average_precisions.append(ap)

    print(average_precisions)

    # Compute mAP
    mAP = np.mean(average_precisions)

    return mAP


@torch.no_grad()
def test_fn():
    model.eval()
    all_data = []
    for data in tqdm(test_loader, desc="testing"):
        imgs, targets = data
        imgs = list(image.to(device) for image in imgs)
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]
        batch_predictions = model(imgs)

        for img_predictions, img_ground_truth in zip(batch_predictions,
targets):
            img_dict = {'predictions': [], 'ground_truth': []}

            for prediction in zip(*img_predictions.values()):
                prediction_tensor = torch.cat((prediction[0],
prediction[1].reshape(1), prediction[2].reshape(1)), dim=0)
                img_dict['predictions'].append(prediction_tensor)

            for ground_truth in zip(*img_ground_truth.values()):
                gt_tensor = torch.cat((ground_truth[0],
ground_truth[1].reshape(1)), dim=0)
                img_dict['ground_truth'].append(gt_tensor)

            if len(img_dict['predictions']) == 0:
img_dict['predictions'].append(torch.tensor([0, 0, 0, 0, 0, 0]))

            img_dict['predictions'] = torch.stack(img_dict['predictions'])
            img_dict['ground_truth'] = torch.stack(img_dict['ground_truth'])
            all_data.append(img_dict)

    return all_data


torch.cuda.empty_cache()
model = torch.load(MODEL_PATH)
```

```
timings=np.zeros(len(test_loader))

model.eval()

start_event = torch.cuda.Event(enable_timing=True)
end_event = torch.cuda.Event(enable_timing=True)

with torch.no_grad():
    for i, data in enumerate(test_loader):
        imgs, _ = data
        imgs = list(image.to(device) for image in imgs)
        start_event.record()
        output = model(imgs)
        end_event.record()
        torch.cuda.synchronize()
        current_time = start_event.elapsed_time(end_event)
        timings[i] = current_time

# Calculate inference time and frames per second
inference_time = np.sum(timings) / len(test_loader) / 1000.0 # Convert to
seconds
FPS = 1 / inference_time
print("Inference time:", inference_time)
print("FPS:", FPS)


model = torch.load(MODEL_PATH)
model.to(device)

data = test_fn()

mAP = compute_map(data)
print(mAP)
```

Kod za treniranje i testiranje Faster R-CNN modela na Pacal VOC 2012

```
EPOCHS = 100
BATCH_SIZE = 8
LEARNING_RATE=1e-4
PATIENCE = 7
MODEL_PATH = "./faster_rcnn_voc_2012.pth"
NUM_WORKERS = 10
DATA_PATH = "./data"
TRAINABLE_BACKBONE_LAYERS = 5


import torch
import torchvision
from torchvision.transforms import transforms
from torchvision.datasets import VOCDetection
from torch.utils.data import DataLoader
import torch.optim as optim
import torch.optim.lr_scheduler as ls
import logging
import numpy as np
from tqdm import tqdm
from torchvision.models.detection import FasterRCNN_ResNet50_FPN_V2_Weights
import math

device = 'cuda' if torch.cuda.is_available() else 'cpu'
device


VOC_CLASSES = (  # always index 0
    'aeroplane', 'bicycle', 'bird', 'boat',
    'bottle', 'bus', 'car', 'cat', 'chair',
    'cow', 'diningtable', 'dog', 'horse',
    'motorbike', 'person', 'pottedplant',
    'sheep', 'sofa', 'train', 'tvmonitor')

classes_dict = {'aeroplane': 1, 'bicycle': 2, 'bird': 3, 'boat': 4,
    'bottle': 5, 'bus': 6, 'car': 7, 'cat': 8, 'chair': 9,
    'cow': 10, 'diningtable': 11, 'dog': 12, 'horse': 13,
    'motorbike': 14, 'person': 15, 'pottedplant': 16,
    'sheep': 17, 'sofa': 18, 'train': 19, 'tvmonitor': 20}


transform = transforms.Compose([
    transforms.ToTensor(),
])

def collate_fn(batch):
    return list(zip(*batch))

def target_transform(targets_dict):
```

```
    bboxes = []
    labels = []

    for instance in targets_dict['annotation']['object']:

        bbox = [float(instance['bndbox']['xmin']),
                float(instance['bndbox']['ymin']),
                float(instance['bndbox']['xmax']),
                float(instance['bndbox']['ymax'])]
        bboxes.append(bbox)

        label = classes_dict[instance['name']]
        labels.append(label)

    bboxes = torch.tensor(bboxes, dtype=torch.float)
    labels = torch.tensor(labels, dtype=torch.int64)

    targets = {}
    targets['boxes'] = bboxes
    targets['labels'] = labels

    return(targets)

test_dataset = VOCDetection(root=DATA_PATH, year="2012", image_set="trainval",
download=True, transform=transform, target_transform=target_transform)

# Split the dataset into train, validation and train datasets with a similar
split to Pascal VOC 2007
train_indices = range(2880)
val_indices = range(2880, 5770)
test_indices = range(5770, 11540)

train_dataset = torch.utils.data.Subset(test_dataset, train_indices)
val_dataset = torch.utils.data.Subset(test_dataset, val_indices)
test_dataset = torch.utils.data.Subset(test_dataset, test_indices)


train_loader = DataLoader(train_dataset, batch_size=1, shuffle=False,
collate_fn=collate_fn, num_workers=NUM_WORKERS)
val_loader = DataLoader(val_dataset, batch_size=1, shuffle=False,
collate_fn=collate_fn, num_workers=NUM_WORKERS)
test_loader = DataLoader(test_dataset, batch_size=1, shuffle=False,
collate_fn=collate_fn, num_workers=NUM_WORKERS)

# Print sizes of all datasets
print(f'Size of train dataset: {len(train_loader)}')
print(f'Size of validation dataset: {len(val_loader)}')
print(f'Size of test dataset: {len(test_loader)}')
```

```
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True,
collate_fn=collate_fn, num_workers=NUM_WORKERS)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False,
collate_fn=collate_fn, num_workers=NUM_WORKERS)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False,
collate_fn=collate_fn, num_workers=NUM_WORKERS)


for imgs, targets in train_loader:
    print(targets[0])
    break


model =
torchvision.models.detection.fasterrcnn_resnet50_fpn_v2(weights=FasterRCNN_Res
Net50_FPN_V2_Weights.COCO_V1,
                                                         trainable_back
bone_layers=TRAINABLE_BACKBONE_LAYERS)

num_classes = 21
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
in_features = model.roi_heads.box_predictor.cls_score.in_features
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

model = model.to(device)


optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE, weight_decay=1e-
5)
scheduler = ls.ReduceLROnPlateau(optimizer, 'min', patience=2)


class EarlyStopping:
    """Early stops the training if validation loss doesn't improve after a
given patience."""

    def __init__(self, path, patience: int = 7, min_delta: float = 0,
trace_func=print):
        """
        Args:
            patience (int): How long to wait after last time validation loss
improved.
                            Default: 7
            min_delta (float): Minimum change in the monitored quantity to
qualify as an improvement.
                            Default: 0
            path (str): Path for the model to be saved to.
            trace_func (function): trace print function.
                            Default: print
        """
        self.counter = 0
```

```
        self.min_loss = None
        self.early_stop = False
        self.val_loss_min = np.Inf
        self.path = path
        self.trace_func = trace_func

        if not callable(trace_func):
            raise TypeError("Argument trace_func should be a function")

        if patience < 1:
            raise ValueError("Argument patience should be positive integer")
        self.patience = patience

        if min_delta < 0.0:
            raise ValueError(
                "Argument min_delta should not be a negative number")
        self.delta = min_delta

        logging.info('EarlyStopping(), path=%s patience=%d min_delta=%f',
                     path, patience, min_delta)

    def __call__(self, loss, model):
        '''Check whether early stopping is needed '''
        print("loss:", loss, "min_loss:", self.min_loss)
        if self.min_loss is None:
            self.min_loss = loss
            self.save_checkpoint(loss, model)
        elif loss >= self.min_loss - self.delta:
            self.counter += 1
            self.trace_func(
                f'EarlyStopping counter: {self.counter} out of
{self.patience}')
            if self.counter >= self.patience:
                self.early_stop = True
        else:
            self.min_loss = loss
            self.save_checkpoint(loss, model)
            self.counter = 0

    def save_checkpoint(self, val_loss, model):
        '''Saves model when validation loss decreases.'''
        logging.info('save_checkpoint(), loss=%f', val_loss)
        torch.save(model, self.path)


def train_fn():
    global iteration
    total_loss = 0.0
    model.train()
```

```
    for data in tqdm(train_loader, desc="train   "):
        imgs, targets = data

        imgs = list(image.to(device) for image in imgs)
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

        losses = model(imgs, targets)

        loss = sum(loss for loss in losses.values())
        total_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    return total_loss/len(train_loader)


@torch.no_grad()
def eval_fn():
    total_loss = 0.0
    model.train()
    for data in tqdm(val_loader, desc="validate"):
        imgs, targets = data
        imgs = list(image.to(device) for image in imgs)
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]
        losses = model(imgs, targets)
        loss = sum(l for l in losses.values())
        total_loss += loss.item()

    return total_loss/len(val_loader)


def train(model):
    train_losses = []
    valid_losses = []
    early_stopping = EarlyStopping(MODEL_PATH, patience=PATIENCE,
min_delta=0.0)

    for i in range(0, EPOCHS):
        train_loss = train_fn()
        train_losses.append(train_loss)
        valid_loss = eval_fn()
        valid_losses.append(valid_loss)

        scheduler.step(valid_loss)
        early_stopping(valid_loss, model)
        if early_stopping.early_stop:
            print("early stopping ...")
            break
```

```
    return model, train_losses, valid_losses


torch.cuda.empty_cache()


model, tlosses, vlosses = train(model)


def calculate_iou(pred_box, gt_box):
    """
    Calculate intersection over union (IoU) between two bounding boxes.
    """
    # Calculate intersection area
    x_left = max(pred_box[0], gt_box[0])
    y_top = max(pred_box[1], gt_box[1])
    x_right = min(pred_box[2], gt_box[2])
    y_bottom = min(pred_box[3], gt_box[3])

    intersection_area = max(0, x_right - x_left + 1) * max(0, y_bottom - y_top
+ 1)

    # Calculate union area
    pred_area = (pred_box[2] - pred_box[0] + 1) * (pred_box[3] - pred_box[1] +
1)
    gt_area = (gt_box[2] - gt_box[0] + 1) * (gt_box[3] - gt_box[1] + 1)
    union_area = pred_area + gt_area - intersection_area

    # Calculate IoU
    iou = intersection_area / union_area

    return iou

def compute_ap(precisions, recalls):
    """
    Compute Average Precision (AP) from precision and recall lists.
    """
    # Add edge values to prevent out of range errors
    precisions = np.concatenate(([1], precisions, [0]))
    recalls = np.concatenate(([0], recalls, [1]))

    recall_levels = np.linspace(0, 1, 11)
    interpolated_precisions = []

    for recall_level in recall_levels:
        # Find the highest recall value less than or equal to the current
recall level
        recall_mask = recalls >= recall_level
        if np.any(recall_mask):
```

```
            precision_at_recall_level = np.max(precisions[recall_mask])
            interpolated_precisions.append(precision_at_recall_level)

    # Compute the average interpolated precision
    average_precision = np.mean(interpolated_precisions)

    return average_precision

def compute_map(data, iou_threshold=0.5):
    """
    Compute Mean Average Precision (mAP) for object detection.
    """
    average_precisions = []

    for class_idx in tqdm(range(1, 21), desc='Calculating mAP'):
        # Initiate per score precision and recall lists for AP calculation
        per_score_precision = np.zeros(19)
        per_score_recall = np.zeros(19)

        for i, score in enumerate(range(95, 0, -5)):
            true_positives = 0
            false_positives = 0
            total_class_gt = 0

            for img_dict in data:
                predictions = img_dict['predictions']
                ground_truth = img_dict['ground_truth']

                # Get predictions and ground_truth for the current class
                class_predictions = predictions[(predictions[:, 4] ==
class_idx) & (predictions[:, 5] >= score/100)]
                class_gt = ground_truth[ground_truth[:, 4] == class_idx]

                total_class_gt += len(class_gt)
                detected_gt_boxes = []

                for pred in class_predictions:
                    pred_box = pred[:4]
                    max_iou = 0

                    for i, gt in enumerate(class_gt):
                        gt_box = gt[:4]
                        iou = calculate_iou(pred_box, gt_box)

                        if iou > max_iou:
                            max_iou = iou
                            best_gt_idx = i
```

```
                    if max_iou > iou_threshold and best_gt_idx not in
detected_gt_boxes:
                        true_positives += 1
                        detected_gt_boxes.append(best_gt_idx)
                    else:
                        false_positives += 1

            per_score_precision[i] = true_positives / (true_positives +
false_positives + 1e-16)
            per_score_recall[i] = true_positives / (total_class_gt + 1e-16)

        # Compute AP
        ap = compute_ap(per_score_precision, per_score_recall)
        average_precisions.append(ap)

    print(average_precisions)

    # Compute mAP
    mAP = np.mean(average_precisions)

    return mAP


@torch.no_grad()
def test_fn():
    model.eval()
    all_data = []
    for data in tqdm(test_loader, desc="testing"):
        imgs, targets = data
        imgs = list(image.to(device) for image in imgs)
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]
        batch_predictions = model(imgs)

        for img_predictions, img_ground_truth in zip(batch_predictions,
targets):
            img_dict = {'predictions': [], 'ground_truth': []}

            for prediction in zip(*img_predictions.values()):
                prediction_tensor = torch.cat((prediction[0],
prediction[1].reshape(1), prediction[2].reshape(1)), dim=0)
                img_dict['predictions'].append(prediction_tensor)

            for ground_truth in zip(*img_ground_truth.values()):
                gt_tensor = torch.cat((ground_truth[0],
ground_truth[1].reshape(1)), dim=0)
                img_dict['ground_truth'].append(gt_tensor)

            if len(img_dict['predictions']) == 0:
img_dict['predictions'].append(torch.tensor([0, 0, 0, 0, 0, 0]))
```

```
            img_dict['predictions'] = torch.stack(img_dict['predictions'])
            img_dict['ground_truth'] = torch.stack(img_dict['ground_truth'])
            all_data.append(img_dict)

    return all_data


torch.cuda.empty_cache()
model = torch.load(MODEL_PATH)

timings=np.zeros(len(test_loader))

model.eval()

start_event = torch.cuda.Event(enable_timing=True)
end_event = torch.cuda.Event(enable_timing=True)

with torch.no_grad():
    for i, data in enumerate(test_loader):
        imgs, _ = data
        imgs = list(image.to(device) for image in imgs)
        start_event.record()
        output = model(imgs)
        end_event.record()
        torch.cuda.synchronize()
        current_time = start_event.elapsed_time(end_event)
        timings[i] = current_time

# Calculate inference time and frames per second
inference_time = np.sum(timings) / len(test_loader) / 1000.0 # Convert to
seconds
FPS = 1 / inference_time
print("Inference time:", inference_time)
print("FPS:", FPS)


model = torch.load(MODEL_PATH)

data = test_fn()

mAP = compute_map(data)
print(mAP)
```