Matematički fakultet, Univerzitet u Beogradu

Računaranska inteligencija: Instance segmentation

Autori:
Marko Nikitović
Kristijan Petronijević

Predmetni nastavnici:
Stefan Kapunac
Ivan Pop-Jovanov

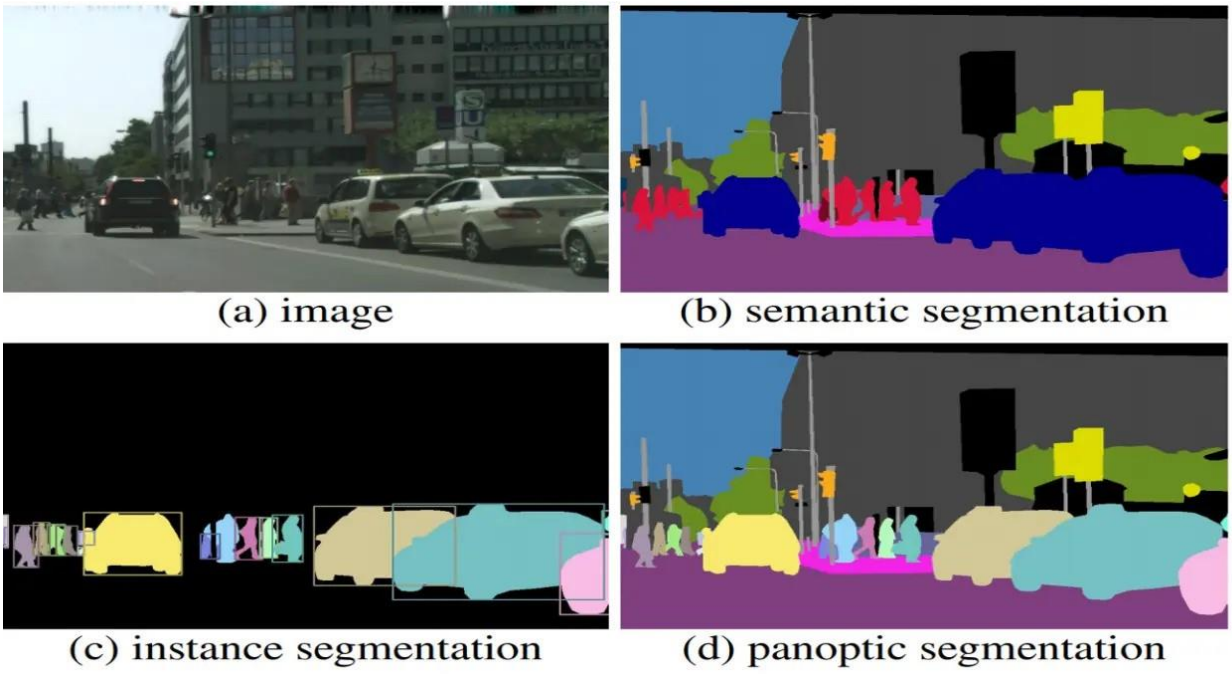
Školska godina 2023/2024

1. Uvod

Instance segmentation je računarski zadatak koji kombinuje klasične probleme detekcije objekata i segmentacije slika. Cilj instance segmentacije je ne samo prepoznavanje i lokalizacija objekata na slici, već i njihova precizna piksel-po-piksel segmentacija. Za razliku od semantičke segmentacije, gde su svi objekti iste klase obeleženi kao jedna celina, instance segmentation razlikuje pojedinačne objekte unutar iste klase. Ova tehnika je ključna u aplikacijama poput autonomne vožnje, medicinske analize slika i analize sportskih događaja, gde je važno razumeti i izolovati specifične objekte u složenim scenama. Moderni modeli, poput Mask R-CNN-a i varijanti YOLO modela, koriste se za rešavanje ovog problema, postižući visoku preciznost i efikasnost.

1.1. Slični problemi

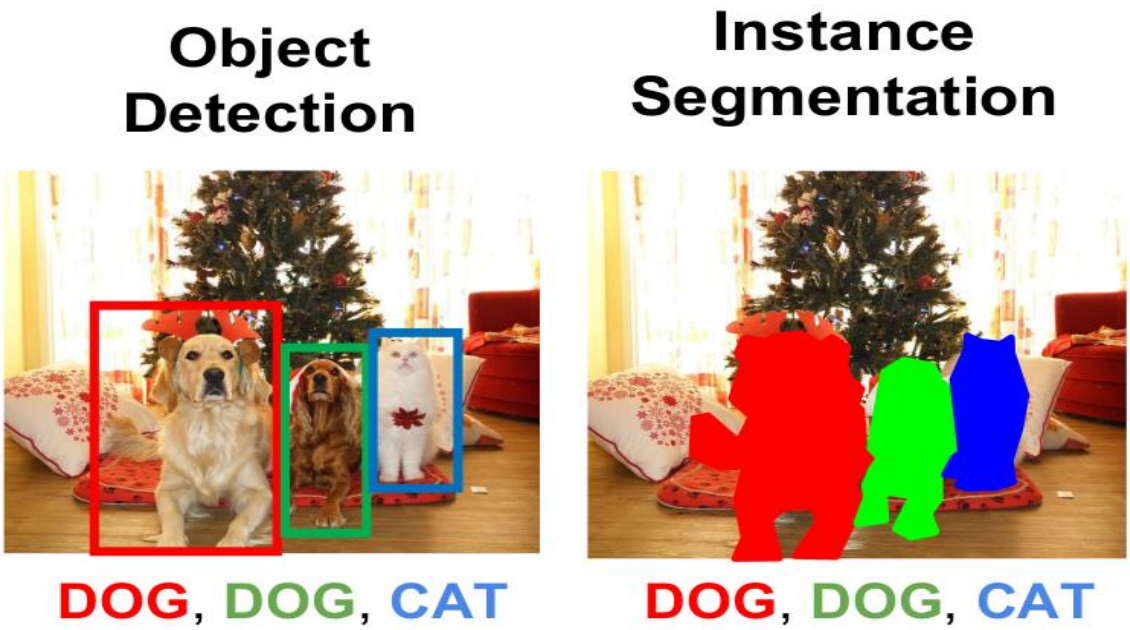
U oblasti računarskog vida javlja se odredjen broj relativno sličnih problema, te ćemo ukratko razmotriti njihove razlike.



Slika: Razlike izmedju sličnih problema

Razlika izmedju **instance segmentation** i **semantic segmentation**: glavna razlika predstavlja to što kod semantic segmentation prepoznajemo samo različite klase objekata na slici, ali ne i različite instance u okviru njih.

Panoptic segmentation predstavlja spoj prethodne dve tehnike i to radi nivou cele slike, vršeći sveobuhvatno prepoznavanje objekata na slikama.



Slika: Instance segmentation vs Object detection

Kod **object detection** prepoznajemo samo “kutije” ali ne i same maske objekata na slikama.

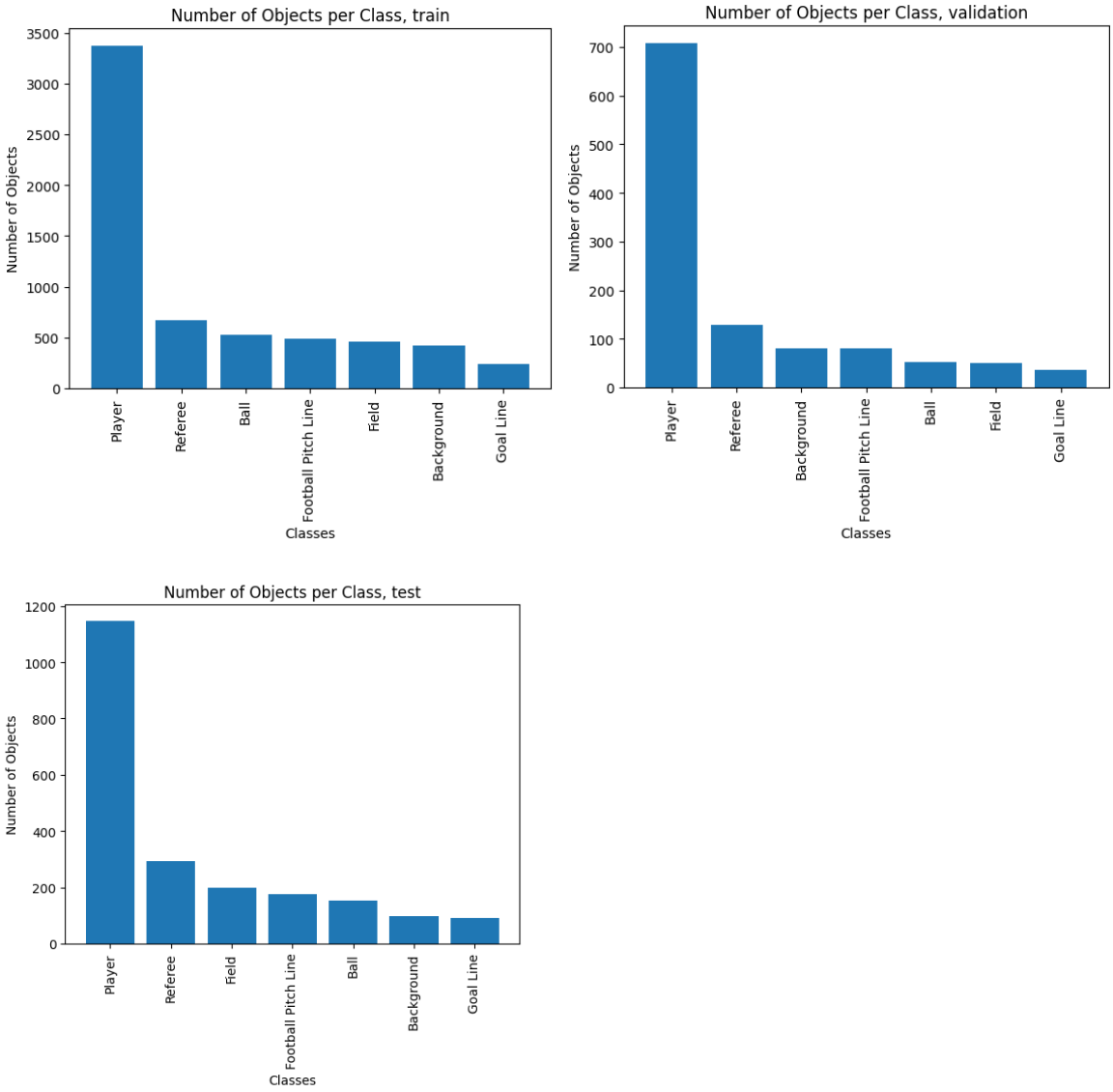
1.2. Podaci

Za konkretan domen primene uzeli smo fudbalsku utakmicu, gde je potrebno razlikovati igrače, loptu, sudiju itd.

U našem slučaju podaci su podeljeni u tri **batch**-a, i svaki batch je jedan video snimak. Prvi pokusaj treniranja je bio da po jedan deo ostavimo za trening, test i validaciju. Ispostavilo se da u tom slučaju model nije uspeo da dovoljno nauci iz trening skupa, i videćemo kasnije da je davao lose rezultate.

Takodje, taktika da se svi **frame**-ovi podele nasumično u trening, validaciju i test može dovesti do nerealnih rezultata, jer razlike izmedju pojedinačnih frejmova su cesto minimalne.

Naposletku, da bismo omogućili modelu da dovoljno nauci ali i da ne dobijamo lažne rezultate snimke smo podelili po kadrovima i uniformno ih dodelili u svaki od skupova, vodeći računa da odnosi budu približno 70:10:20. Na ovaj način izbegli smo **data leakage,** a modelu obezbedili dovoljnu količinu za učenje.



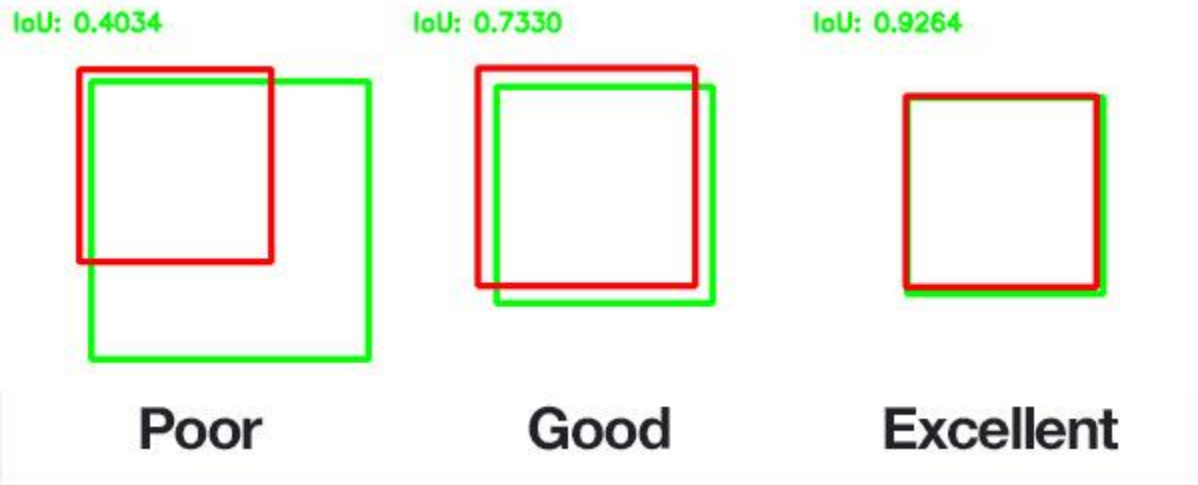
Slika: Zastupljenost odredjenih klasa za svaki test, trening i validaciju

Klase su slično rasporedjene u sva tri skupa, uz minimalne varijacije.

1.3. Metrike modela

Za evaluacija kvaliteta modela uglavnom smo se oslanjali na **mAP** vrednost (eng. Mean average precision). Ova statistika se racuna kao prosecna vrednost **AP** (eng. Average precision) vrednosti za svaku od klasa, gde se AP izracunava kao povrsina ispod **PR** (eng. Precision - recall) krive.

IoU (eng. Intersection over union) predstavlja odnos preseka izmedju preseka i unije, od nje zavisi da li cemo odredjenu predikciju uzimati u obzir ili ne. Prilikom racunanja mAP mi smo korisili IoU iz opsega [0.05:0.95] sa korakom 0.05, i na kraju uzimali srednju vrednost.



Slika: Instance segmentation vs Object detection

Još jedna metrika se javlja za ispitivanje modela, i to uglavnom za deo modela koji predlaze regione, u pitanju je **mAR** (eng. Mean average recall). Za svaku klasu se prosečno izračunava **recall** na svakoj IoU vrednosti, a zatim se uzima prosek tog proseka za svaku klasu. To daje **mAR^D** za D detekcija (D može biti broj poput 1, 10, 100 detekcija itd.).

2. Modeli

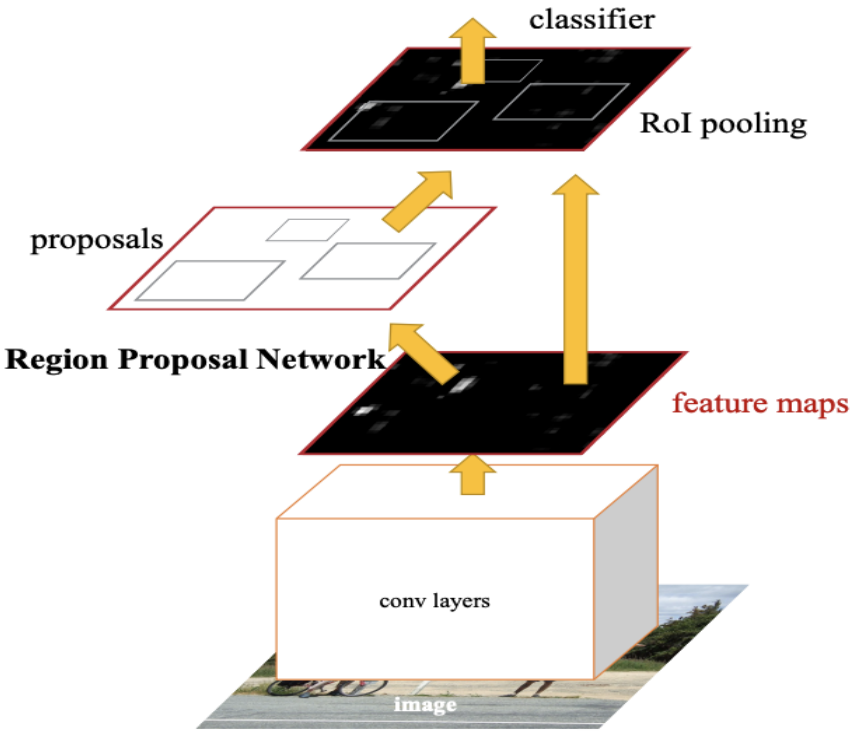
U ovom projektu pokrili smo sledeće modele za rešavanje problema prepoznavanja instanci:

- Mask RCNN (Mask Region-based Convolutional Neural Network)
- Cascade Mask RCNN (Detectron library)
- YOLOv8

U nastavku ćemo redom prolaziti kroz ove modele.

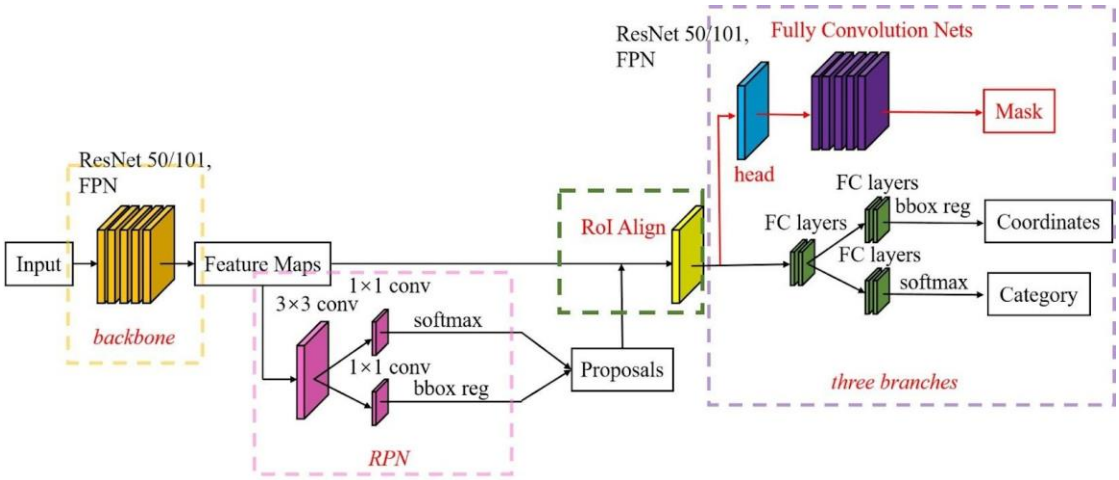
2.1. Mask RCNN

Prvi model koji smo razmtrali je **Mask RCNN**, koji se bazira na Faster RCNN arhitekturi za detekciju objekata.



Slika: Arhitektura Faster RCNN mreže

Kod Mask RCNN imamo dodatani konvulutivni sloj koji predvidja masku za svaku od feature mapa.



Slika: Arhitektura Mask RCNN mreže

U našem slučaju, kao **backbone** uzeli smo već istreniranu ResNet50 mrežu pripremljenu na imageNet skupu podataka. **Anchor_generator** podešava veličine kutija koje RPN mreža generise, dok **roi_pooler** definiše način na koji svodimo na feature mape jednake veličine.

Početne podele podataka po batch-ovima dale su los rezultat na test skup (mAP ≈ 0.05). Tek prilikom podele snimaka po scenama, model je uspeo da bolje uopštava. Tom prilikom smo koristili sledeću arhitekturu:

```
def get_instance_segmentation_model(num_classes):
    backbone = torchvision.models.resnet50(weights=ResNet50_Weights.IMAGENET1K_V1)
    backbone = torch.nn.Sequential(*(list(backbone.children()))[:-2]))
    backbone.out_channels = 2048

    anchor_generator = AnchorGenerator(
        sizes=((32, 64, 128, 256, 512),)
        aspect_ratios=((0.5, 1.0, 2.0),) * 5
    )

    roi_pooler = torchvision.ops.MultiScaleRoIAlign(
        featmap_names=['0'], output_size=7, sampling_ratio=2
    )

    model = MaskRCNN(backbone,
                     num_classes=num_classes,
                     rpn_anchor_generator=anchor_generator,
                     box_roi_pool=roi_pooler)

    return model
```

Slika: Korišćena arhitektura Mask RCNN mreže

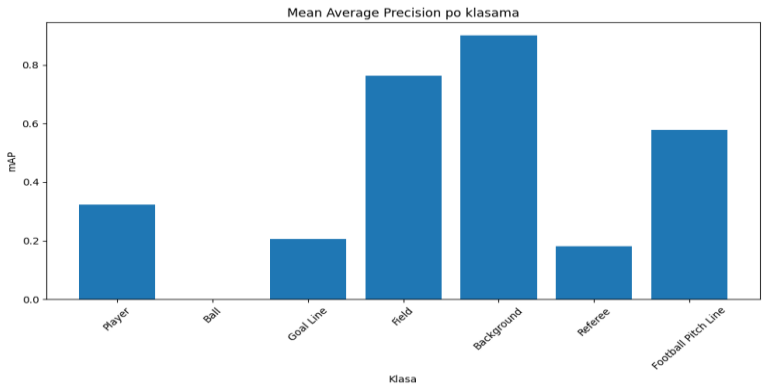
Ovaj optimizator koristi stohastički gradijentni spust (SGD) sa stopom učenja od 0.005, momentumom od 0.9 za ubrzanje konvergencije, i **weight decay**-om od 0.0005 za regularizaciju. Korišćen je **StepLR scheduler**, koji smanjuje stopu učenja za faktor 0.1 svakih 3 epohe, čime se postepeno finije prilagođava tokom treniranja.

```
optimizer = torch.optim.SGD(params, lr=0.005, momentum=0.9, weight_decay=0.0005)
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)
num_epochs = 7
```

Slika: Korišćeni optimizator

Na inicijalnom treniranju dobijeni su sledeći rezultati:

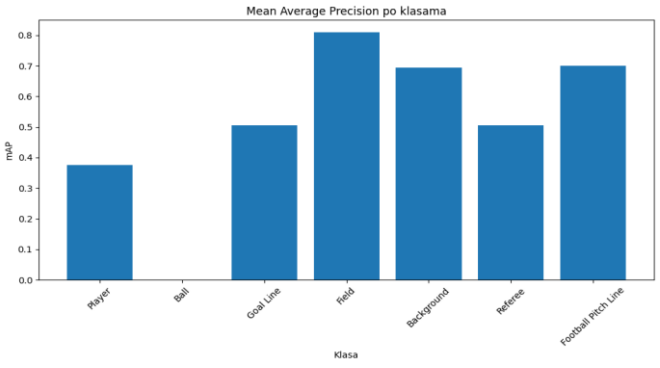
Metrička vrednost	Vrednost
map	tensor(0.4215)
map_50	tensor(0.6788)
map_75	tensor(0.3982)
map_small	tensor(0.)
map_medium	tensor(0.1148)
map_large	tensor(0.5269)
mar_1	tensor(0.3714)
mar_10	tensor(0.4822)
mar_100	tensor(0.4853)
mar_small	tensor(0.)
mar_medium	tensor(0.1584)
mar_large	tensor(0.5966)



Slika: Rezultati treniranja modela, (validacioni)

Možemo da primetimo da model uopšte ne prepoznaje loptu, kao i da slabije prepoznaje odredjene klase. U pokusaju da obuhvatimo manje objekte, u atribut **sizes** smo dodali i veličinu 16 (predstavlja broj piksela za veličinu kutija) i dobili smo nešto bolje rezultate:

Metrička vrednost	Vrednost
map	tensor(0.5128)
map_50	tensor(0.7426)
map_75	tensor(0.5557)
map_small	tensor(0.)
map_medium	tensor(0.2955)
map_large	tensor(0.6338)
mar_1	tensor(0.4262)
mar_10	tensor(0.5666)
mar_100	tensor(0.5757)
mar_small	tensor(0.)
mar_medium	tensor(0.3293)
mar_large	tensor(0.7024)

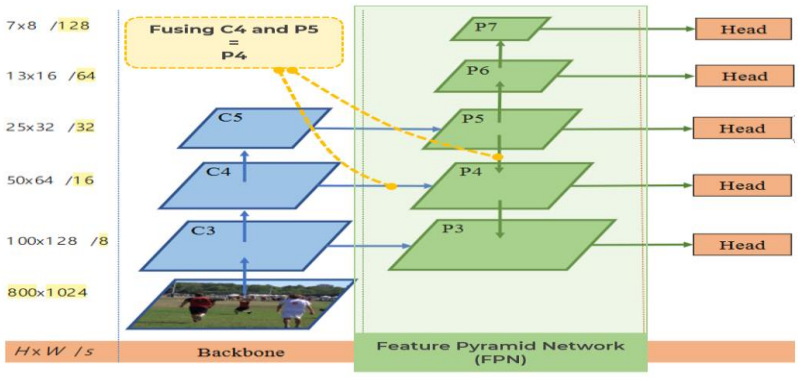


Slika: Rezultati treniranja modela nakon dodavanja veličine 16px u sizes, (validacioni)

Dobijaju se nešto bolji rezultati za klase igrači, sudije i gol linija, ali i dalje mnogo bolje prepoznaje velike objekte nego male i srednje. U nastavku ćemo pokusati da resimo taj problem. Ovo je standardni problem koji se javlja sa slikama koje sadrže veći broj različitih objekata.

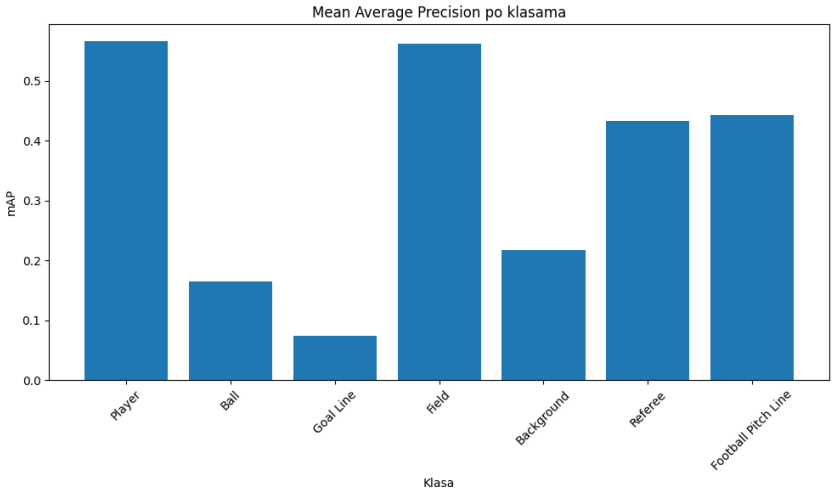
2.1.1. Menjanje arhitekture, korišćenje FPN

Prvo uvodimo **Feature Pyramid Network**, koji kombinuje informacije sa feature map-a koje se nalaze na različitim nivoima.



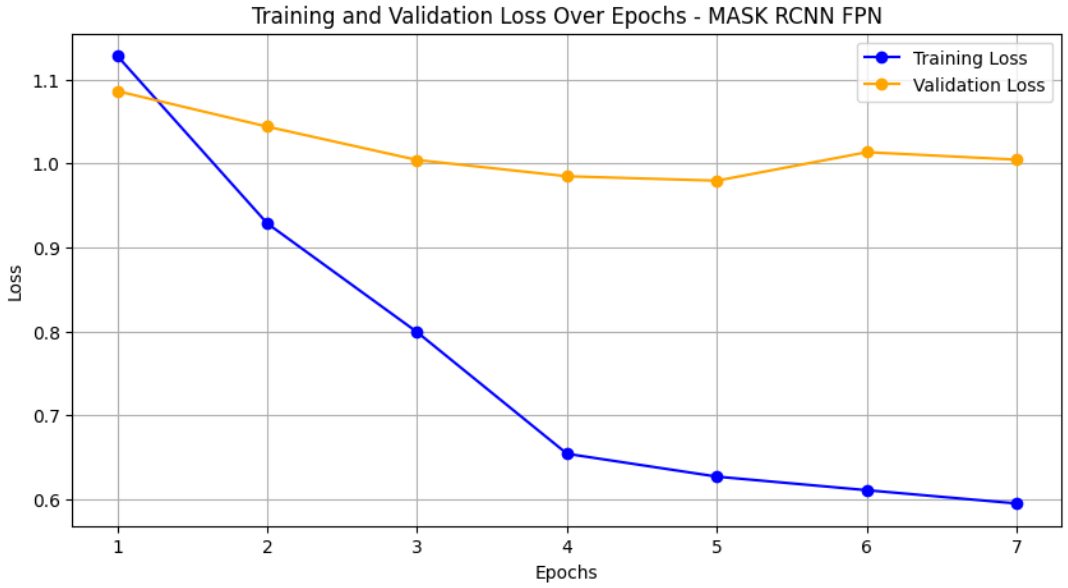
Slika: Feature pyramid network arhitektura

Na različitim nivoima definišemo različite vrednosti za veličinu "kutija" (eng. Bounding box), što je nivo apstraktniji odnosno sadrži vise semantičkih informacija na njemu čemu koristi veće veličine kutija (treba uzeti u obzir da FPN arhitektura kombinuje znanja sa različitih nivoa). Takodje smo promenili parametre koji kontrolišu broj predloga regiona (eng. **region proposals**) koji RPN generiše pre i posle **NMS**-a (Non-Maximum Suppression) tokom treniranja i testiranja. (Ista arhitektura se koristi do kraja, pogledati kod)



Slika: Nakon promene arhitekture, mAP po klasama

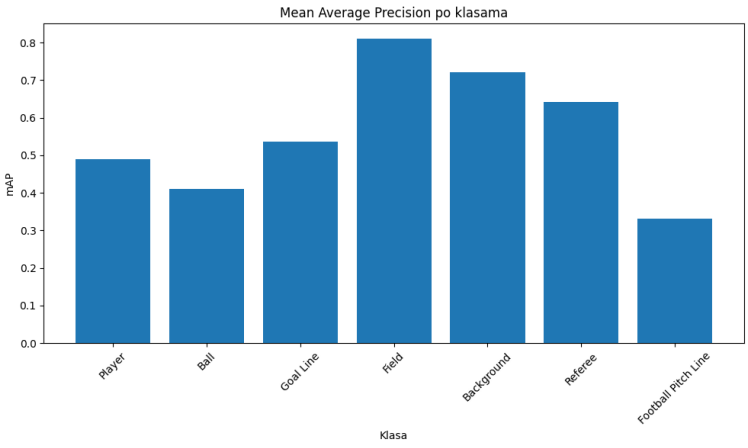
Možemo primetiti da je model počeo da prepoznaje loptu koja je najmanji objekat na slikama kao da su porasli i rezultati za klasu igrač. Ali da je smanjio kvalitet predikcije za gol liniju i pozadinu koji zauzimaju veći deo slike.



Slika: Kretanje greške tokom treniranja

Deluje kao da model pati od preuranjene konvergencije. Jer greška na validacionom skupu je u početku opadala ali ne previse, da bi posle toga stagnirala. Povećaćemo broj epoha na 10 (zbog ograničenih resursa) i podesićemo korak kada **lr_schreduler** menja korak učenja na 6.

```
Ukupne metrike:
Metrička vrednost      Vrednost
              map tensor(0.5631)
           map_50 tensor(0.8625)
           map_75 tensor(0.5900)
        map_small tensor(0.2070)
       map_medium tensor(0.3982)
        map_large tensor(0.6000)
            mar_1 tensor(0.4848)
           mar_10 tensor(0.6331)
          mar_100 tensor(0.6451)
       mar_small tensor(0.2240)
      mar_medium tensor(0.4430)
       mar_large tensor(0.6887)
```



Slika: Evaluacija modela nakon promene broj epoha

Dobijeni model je bolji od svih predjasnjih po dobijenim statistikama. Jedino gde je malo lošiji je u predvidjanju za igrače.

2.1.2. Proširivanje skupa podataka

Posto nas trening skup ima 567 slika, ima smisla na neki način povećati veličinu trening skupa novim podacima iz kojih bi model mogao bolje da nauci.

Probaćemo nad trening skupom da izvršimo neke osnovne geometrijske transformacije i da nad takvim podacima da pokrenemo prethodni model.

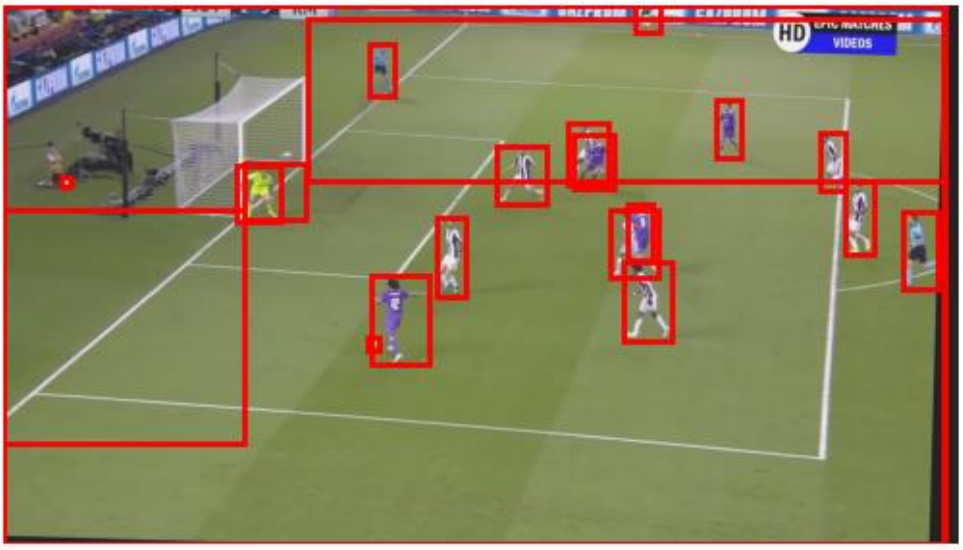
```
geom_transforms =  [A.ShiftScaleRotate(shift_limit=0.0625, scale_limit=0.1, rotate_limit=5,
                        interpolation=cv2.INTER_LINEAR, border_mode=cv2.BORDER_CONSTANT, value=0, mask_value=0, p=0.5),

                    A.HorizontalFlip(p=0.5),
                    A.RandomSizedCrop((800, 1070), height, width, w2h_ratio=1920/1080,
                                      interpolation=cv2.INTER_CUBIC, always_apply=False, p=0.5),
                    A.RandomBrightnessContrast(brightness_limit=0.2, contrast_limit=0.2, p=0.5)
                    ]
```

Slika: Korišćene transformacije nad podacima

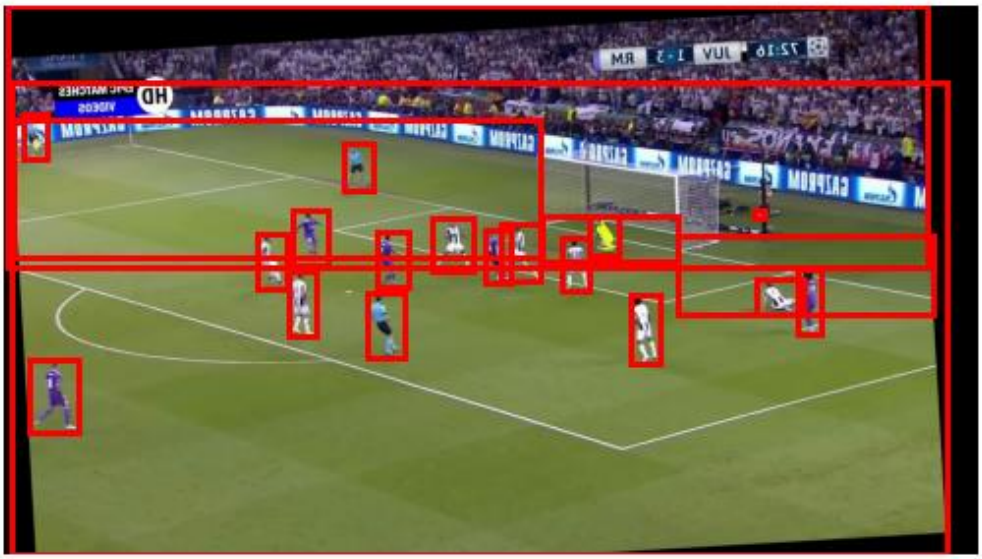
- **A.ShiftScaleRotate**: Nasumično pomera (shift), skalira i rotira sliku u datim granicama (shift za 6.25%, skaliranje do ±10%, rotacija do ±5 stepeni).
- **A.HorizontalFlip**: Nasumično horizontalno preokreće sliku sa verovatnoćom od 50%.
- **A.RandomSizedCrop**: Nasumično seče deo slike sa zadatim opsegom veličine (800-1070 px) i prilagođava ga na datu visinu i širinu, zadržavajući odnos širine i visine 1920:1080.
- **A.RandomBrightnessContrast**: Nasumično menja osvetljenost i kontrast slike unutar zadatih granica (±20%).

Ove transformacije su umerene, odnosno neće uvesti neke drastične promene u slike (imalo bi smisla probati i sa radikalnijim promenama).

Takodje vodili smo računa da se maske i granične kutije preslikaju na pravi način, za faktor uvećana smo koristili 2 (duplo vise slika nego u originalnom skupu).



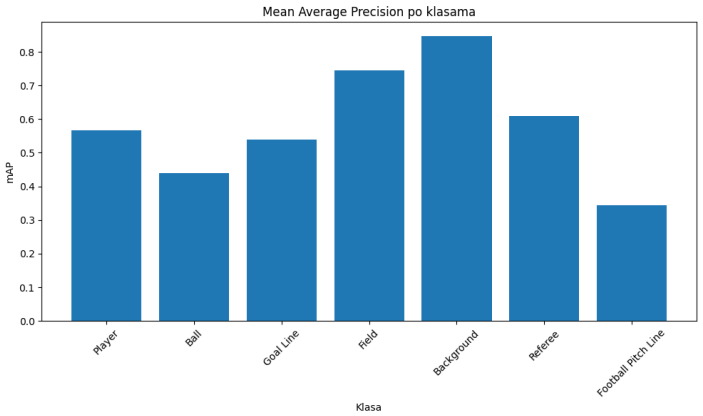
Slika: Primer primene transformacije na sliku



Slika: Primer primene transformacije na sliku

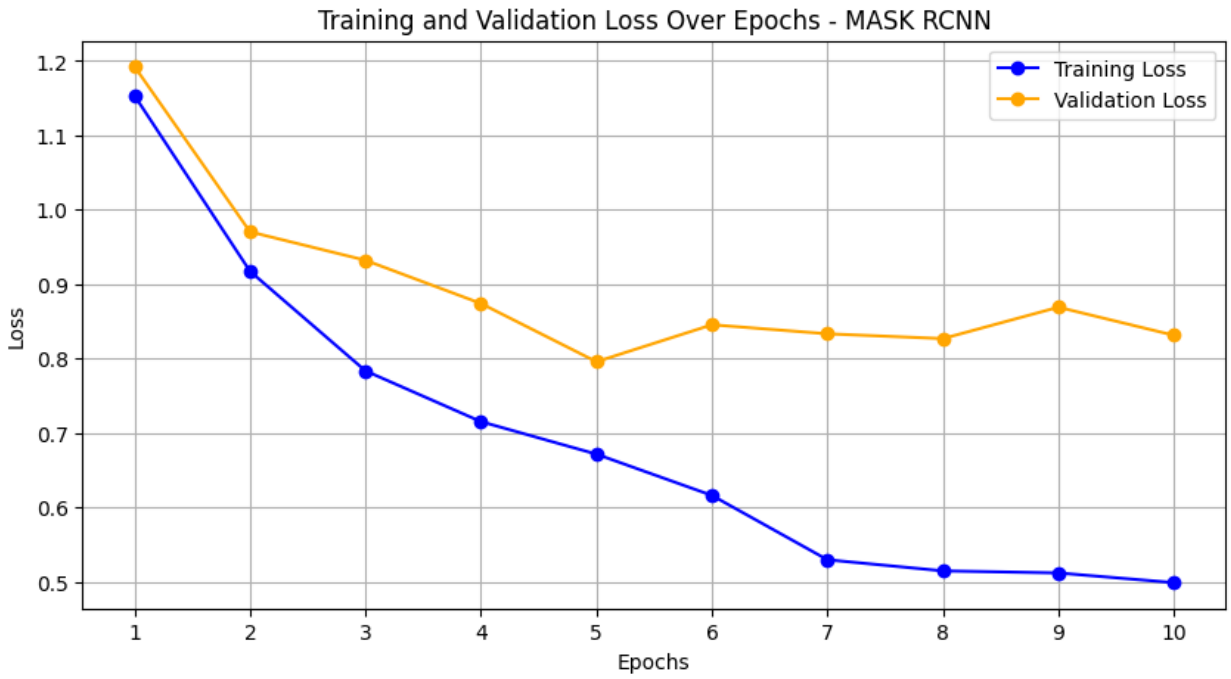
Sledi prikaz dobijenih rezultata:

```
Ukupne metrike:
Metrička vrednost      Vrednost
             map tensor(0.5843)
          map_50 tensor(0.8870)
          map_75 tensor(0.5841)
       map_small tensor(0.2204)
      map_medium tensor(0.4023)
       map_large tensor(0.6295)
           mar_1 tensor(0.4625)
          mar_10 tensor(0.6376)
         mar_100 tensor(0.6541)
       mar_small tensor(0.2510)
      mar_medium tensor(0.4678)
       mar_large tensor(0.6917)
```



Slika: Dobijeni rezultati nakon proširivanja podataka

Ukoliko pogledamo kako se kreće greška za validacioni i trening skup.



Slika: Trening i test greška tokom epoha

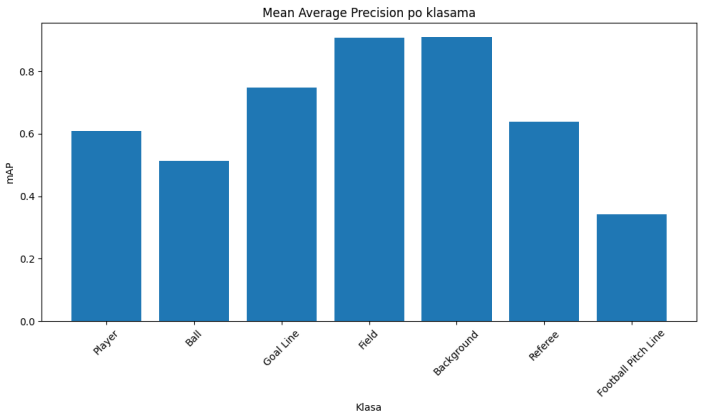
Epoha	1	2	3	4	5	6	7	8	9	10
mAP	0.2287	0.3537	0.5099	0.5160	0.5312	0.5707	0.5710	0.5832	0.5840	0.5843

Slika: mAP vrednost tokom različitih epoha

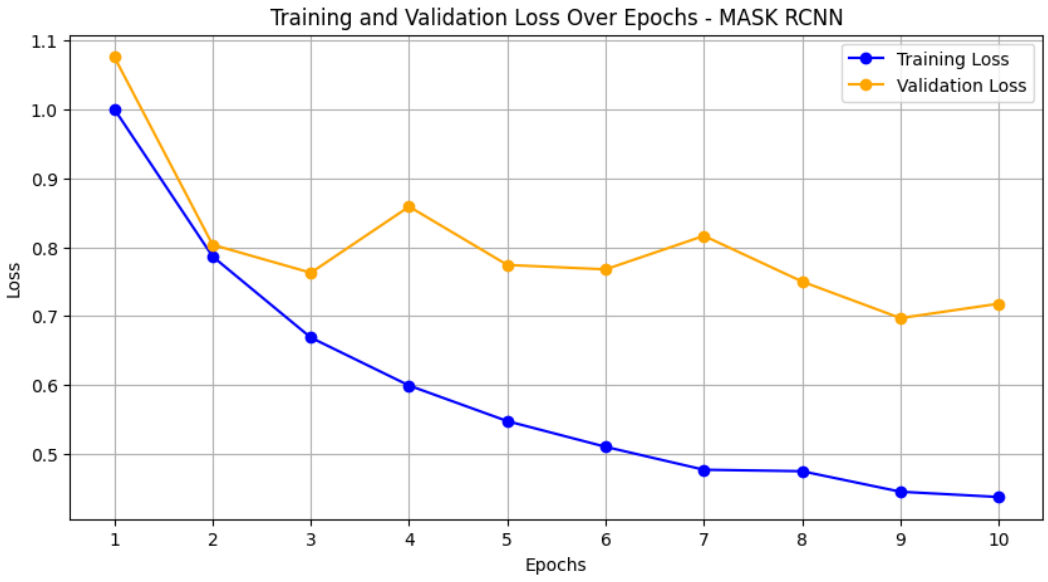
Iz ovih podataka deluje da je opet došlo do preuranjene konvergencije. Ako pogledamo uzrok tome je postavljanje **step_size=6**. Trebalo bi poraditi na optimizatoru.

U nastavku koristićemo **Adam** (eng. Adaptive Moment Estimation) koji je dosta fleksibilniji i nije potrebno vršiti ažuriranje preko **scheduler**-a.

Metrička vrednost	Vrednost
map	tensor(0.6669)
map_50	tensor(0.9028)
map_75	tensor(0.7390)
map_small	tensor(0.2595)
map_medium	tensor(0.4288)
map_large	tensor(0.7137)
mar_1	tensor(0.5398)
mar_10	tensor(0.7091)
mar_100	tensor(0.7278)
mar_small	tensor(0.2819)
mar_medium	tensor(0.4872)
mar_large	tensor(0.7648)



Slika: Rezultati na validacionom skupu



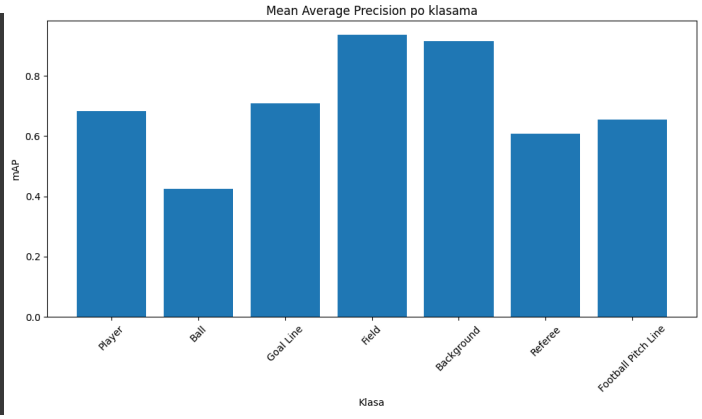
Slika: Tok treniranja

2.1.3. Testiranje modela

Krajnje testiranje kvaliteta modela vrsimo na posebno odvojenom test skupu koji smo na početku. Pošto se poslednji model pokazao kao najbolji njega ćemo koristi za finalne rezultate.

U nastavku prezentujemo rezultate.

Metrička vrednost	Vrednost
map	tensor(0.7041)
map_50	tensor(0.9371)
map_75	tensor(0.8293)
map_small	tensor(0.1416)
map_medium	tensor(0.4662)
map_large	tensor(0.7834)
mar_1	tensor(0.5303)
mar_10	tensor(0.7589)
mar_100	tensor(0.7640)
mar_small	tensor(0.1732)
mar_medium	tensor(0.5073)
mar_large	tensor(0.8244)



Slika: Rezultati na test skupu

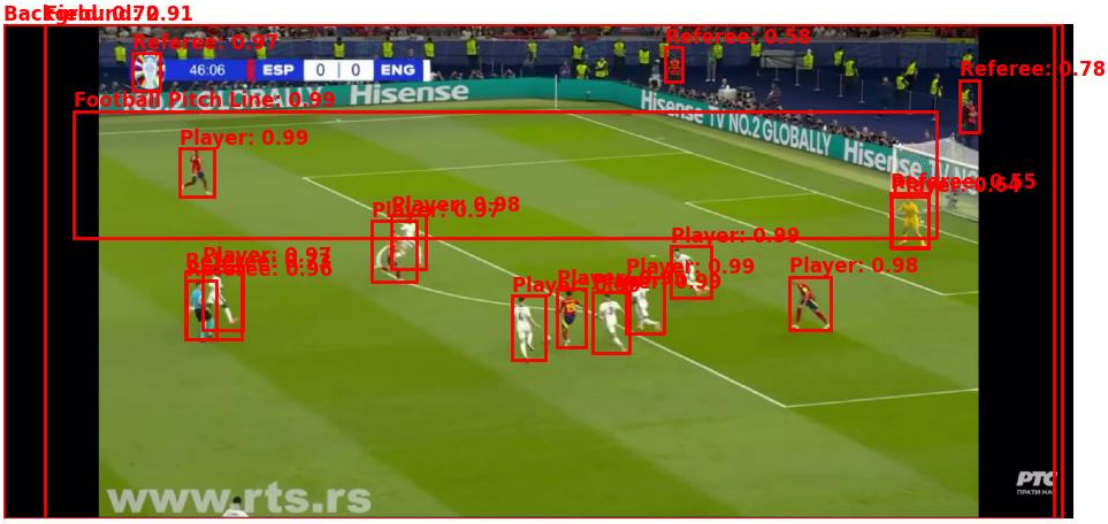
Model najbolje prepoznaje velike objekte, nešto lošije objekte srednje veličine a najlošije najmanje objekte sto je i očekivano. Najbolji rezultati su postignuti **Field** i **Background** (veliki objekti), dok su najlošiji na lopti koju i jeste najteže detektovati.

Klasa	map	mar_100
Player	0.6825	0.7499
Ball	0.4247	0.5196
Goal Line	0.7081	0.7484
Field	0.9358	0.9470
Background	0.9149	0.9444
Referee	0.6076	0.7027
Football Pitch Line	0.6549	0.7358

Slika: Detaljnije statistike po klasama

Rad modela na nepoznatim slikama sa interneta. Možemo primetiti da za dosta objekata koji nisu klase sudija, predvidja da to jesu što nam govori da model ima prostora za poboljšanje.

Slika: Prvi primer, sa maskama i bbox



Slika: Drugi primer, samo bbox

2.1.4. Zaključak ovog dela projekta

Cilj ovog dela projekta je da se postepenim razmatranjem modela i njegovom interpretacijom dodje do boljih rezultata, krećući od jednostavnijih modela i postepenim usložnjavanjem. Poslednji model je dao najbolje rezultate, koji su dobri za to koliko je model kompleksan i da je se koristio Mask RCNN. Daljom modifikacijom hiperparametara može se doci do još boljih rezultata, odnosno model ima prostora za napredak.

2.2. Kaskadni Mask RCNN (Detectron)

Detectron je open-source softverski paket razvijen od strane Facebook AI Research tima, namenjen za implementaciju algoritama za detekciju objekata, segmentaciju slika i druge zadatke iz oblasti kompjuterskog vida. Detectron je zasnovan na PyTorch-u i pruža jednostavan okvir za implementaciju naprednih modela za prepoznavanje objekata.

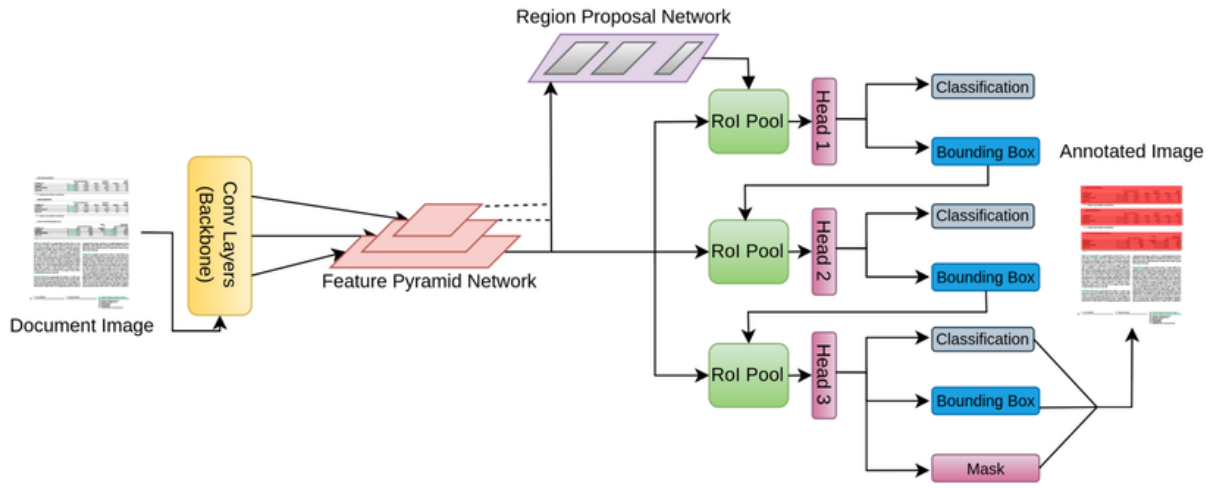
Kaskadni Mask R-CNN je napredna verzija Mask R-CNN arhitekture, koja se koristi za instance segmentation (prepoznavanje i obeležavanje objekata unutar slike). Glavna ideja je da koristi

više faza za preciznije predikcije objekata i njihovih maski.

1. Faze predikcije: Za razliku od standardnog Mask R-CNN-a, Kaskadni Mask R-CNN ima više "kaskadnih" faza. U svakoj fazi, predikcije objekata postaju preciznije, jer se svaka faza zasniva na ispravkama iz prethodne faze.

2. Razdvajanje objekata i maski: Algoritam ne samo da detektuje objekte unutar slike, već i kreira maske koje prikazuju oblik svakog objekta.

3. Upotreba: Kaskadni Mask R-CNN je veoma koristan za zadatke gde je potrebna visoka preciznost detekcije, kao što su autonomna vozila, nadzor ili napredna analiza slika.



Slika: Arhitektura Cascade RCNN

Prvo smo preko **Optuna** biblioteke pronašli najbolje parametre na trening skupu testirajući ih na validacionom skupu. Optuna radi na način da se zada veliki broj hiperpametara i broj pokušaja. Ona u svakom pokusaju pokušava da predvidi najbolje parametre preko odredjenih heuristika i rezultata iz prethodnog treniranja.

Nakon pronadjenih najboljih parametra istreniran je model zajedno na trening i validacionom skupu i testiran na test skupu. Greška modela je testirana preko mAP metrike.

Bounding box (bbox) metrika:

- Ukupni AP: 76.35
- AP50: 91.44
- AP75: 84.44
- AP za male objekte (APs): 43.40
- AP za srednje objekte (APm): 83.25
- AP za velike objekte (APl): 76.74

Specifčne klase:

- Igrač (AP-Player): 80.07
- Lopta (AP-Ball): 41.16
- Gol-linija (AP-Goal Line): 78.59
- Teren (AP-Field): 100.0
- Pozadina (AP-Background): 79.14
- Sudija (AP-Referee): 72.31
- Linija terena (AP-Football Pitch Line): 83.15

Segmentacija (segm) tehnika:

- Ukupni AP: 48.53
- AP50: 64.38
- AP75: 52.37
- AP za male objekte (APs): 12.02
- AP za srednje objekte (APm): 30.35
- AP za velike objekte (APl): 69.28

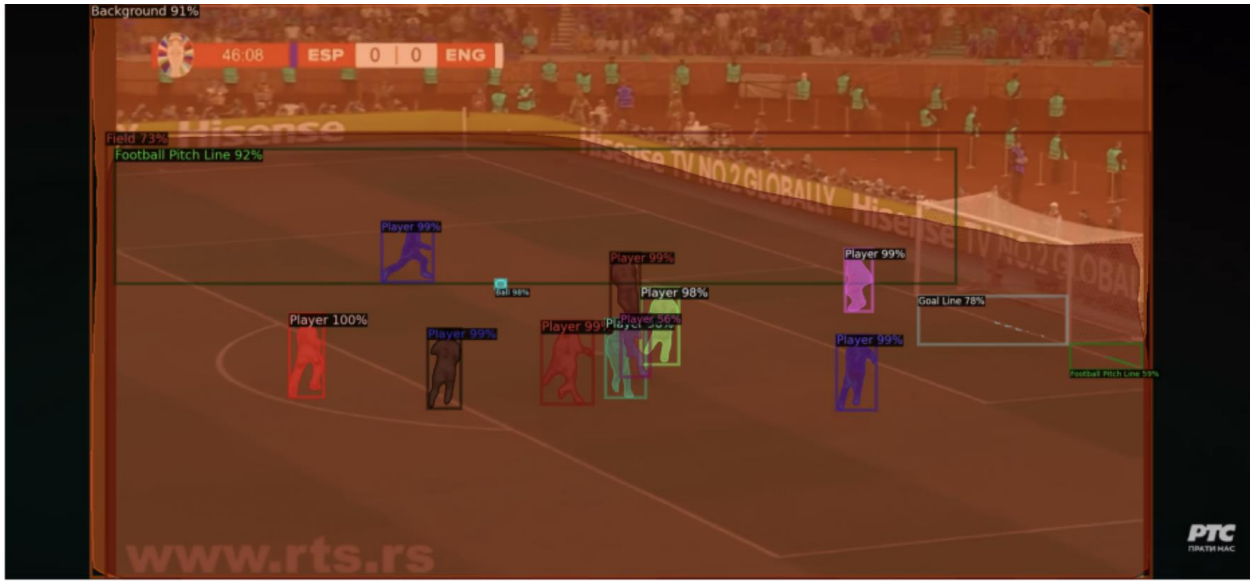
Specifične klase:

- Igrač (AP-Player): 66.28
- Lopta (AP-Ball): 41.28
- Gol-linija (AP-Goal Line): 0.0
- Teren (AP-Field): 100.0
- Pozadina (AP-Background): 77.14
- Sudija (AP-Referee): 55.01
- Linija terena (AP-Football Pitch Line): 0.0

Segmentacija malih objekata (APs: 12.02) i srednjih objekata (APm: 30.35) pokazuje značajno lošije performanse u poređenju sa velikim objektima (APl: 69.28).

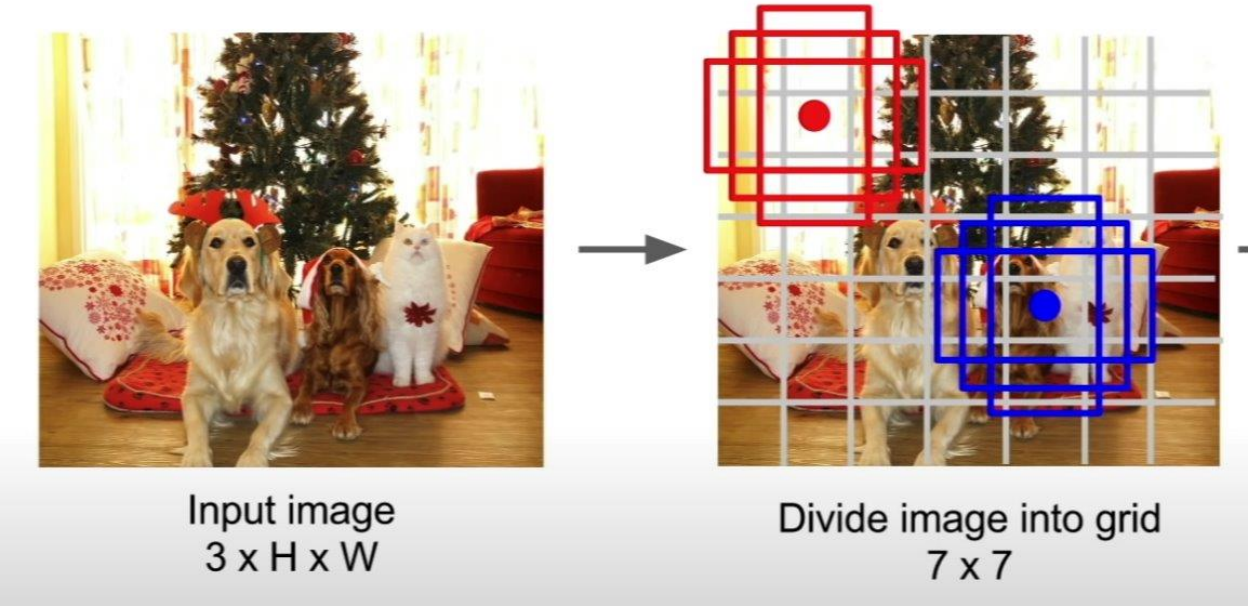
Nakon sto smo se uverili da model radi korektno na test skupu potrebno je trenirati model na celom skupu podataka. (Nismo bili u mogućnosti google colab nas je banovao zbog prevelike upotrebe).

Pronašli smo snimak fudbalske utakmice i izdvojili frejmove iz njega. Pustili smo model da izvrši predvidjanje nad tim frejmovima i dodatno testirali nas model vizuelno.



Slika: Primer rada modela na netreniranim podacima

2.3. YOLOv8



Slika: Podela slike na ćelije

Glavna ideja YOLO algoritma za prepoznavanje objekata je sledeća:

- Podele slike: Slika se deli na mrežu pravougaonih ćelija.
- Detekcija objekta: Svaka ćelija je odgovorna za prepoznavanje objekta čiji se centar nalazi unutar nje.
- Klasifikacija: Ćelija koristi klasifikator da prepozna klasu objekta i daje verovatnoću koliko je sigurna u tu klasifikaciju.
- Predikcija pravougaonika: Pored klasifikacije, svaka ćelija predviđa pravougaonik (bounding box) koji sadrži objekat, predviđajući koordinate centra objekta, kao i širinu i visinu tog pravougaonika.
- Skor pouzdanosti: Skor pouzdanosti detekcije je proizvod verovatnoće klasifikacije i IoU skora (preciznost preklapanja predviđenog pravougaonika sa stvarnim).

YOLO je efikasan jer istovremeno obrađuje celu sliku, a ne deo po deo, što ga čini vrlo brzim i preciznim za prepoznavanje objekata.

3. Literatura

1. Mask R-CNN, https://arxiv.org/abs/1703.06870
2. Feature Pyramid Networks for Object Detection, https://arxiv.org/abs/1612.03144
3. Stanford CS231n, https://cs231n.stanford.edu/
4. https://www.kaggle.com/code/blondinka/how-to-do-augmentations-for-instance-segmentation
5. Veštačka inteligencija, Predrag Janičić i Mladen Nikolić, https://poincare.matf.bg.ac.rs/~janicic//books/VI_A4.pdf
6. https://learnopencv.com/mean-average-precision-map-object-detection-model-evaluation-metric/