Detekcija i praćenje kretanja igrača u videozapisima sportskih utakmica

Pongrac, Marko

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1610

DETEKCIJA I PRAĆENJE KRETANJA IGRAČA U VIDEOZAPISIMA SPORTSKIH UTAKMICA

Marko Pongrac

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1610

DETEKCIJA I PRAĆENJE KRETANJA IGRAČA U VIDEOZAPISIMA SPORTSKIH UTAKMICA

Marko Pongrac

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1610

Pristupnik:	**Marko Pongrac (0036544062)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	izv. prof. dr. sc. Tomislav Hrkać

Zadatak: **Detekcija i praćenje kretanja igrača u videozapisima sportskih utakmica**

Opis zadatka:

Jedna od primjena računalnog vida je automatska analiza video snimaka sportskih utakmica. U okviru završnoga rada treba proučiti pristupe detekciji i praćenju objekata opisane u literaturi te programski ostvariti sustav za detekciju i praćenje igrača u video snimkama sportskih utakmica. Analizirati ponašanje ostvarenog sustava te prikazati i ocijeniti ostvarene rezultate. Radu priložiti izvorni i izvršni kod razvijenih postupaka, ispitne sekvence i rezultate, uz potrebna objašnjenja i dokumentaciju te navesti korištenu literaturu.

Rok za predaju rada: 14. lipnja 2024.

Zahvaljujem mentoru izv. prof. dr. sc. Tomislav Hrkać na pomoći i savjetima prilikom izrade rada. Zahvaljujem se svojoj obitelji na bezuvjetnoj podršci te ohrabrenjima tijekom mojeg obrazovanja.

Sadržaj

Uvod

Računalni vid je područje umjetne inteligencije koje izučava primjenu strojnog učenja i umjetnih neuronskih mreža, s ciljem prepoznavanja i razumijevanja sadržaja slika i videozapisa. Između ostalog, bavi se problemima detekcije i segmentacije objekata iz slike, te praćenjem objekata na videozapisu [1]. U ovome radu posebna pažnja je posvećena detekciji objekata na slici primjenom strojnoj učenja te njihovom praćenju na videozapisu.

Razvojem umjetnih neuronskih mreža, one su postale najpopularniji alat za detekciju objekata na slikama. Međutim, odabir pravog modela umjetne neuronske mreže često predstavlja problem. Razlog tomu je vremenska složenost detekcije objekata na slici, ako se ne koristi računalni akcelerator koji omogućava paralelizaciju procesa detekcije objekata, poput grafičkih procesorskih jedinica.

Praćenje objekata na videozapisu je složen proces s mnogo različitih pristupa rješavanju, od analiziranja promjene središnjih točaka objekata, do naprednijih metoda poput Kalmanovog filtera[2].

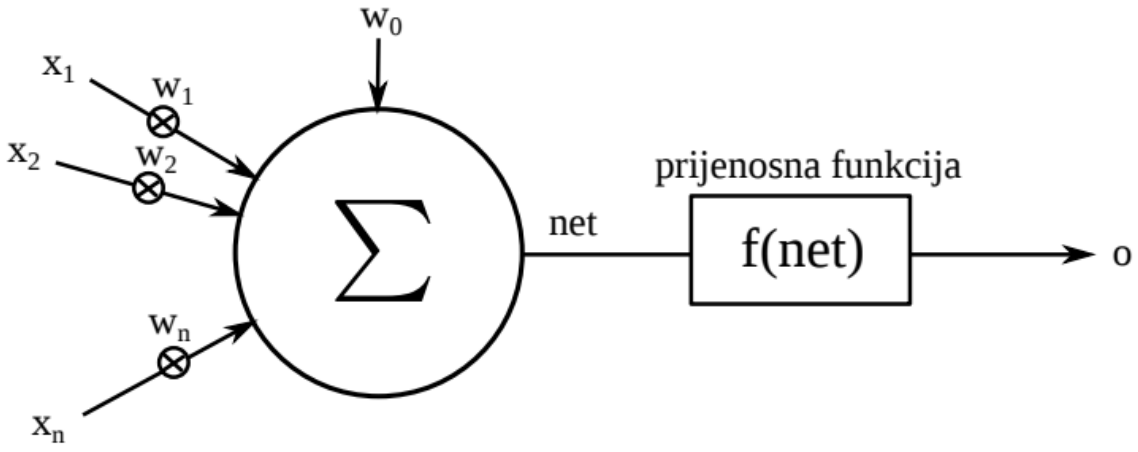
U ovom radu proučavamo izgradnju sustava za detekciju i praćenje igrača na snimkama sportskih utakmica. Prvo je potrebno odabrati prikladan model neuronske mreže te ga trenirati za rješavanje problema detekcije igrača te analizirati njegove metrike. Podatkovni skup za treniranje modela pripremamo sami. Zatim je potrebno implementirati praćenje objekata na videozapisu korištenjem njihovih središnjih točaka. Na kraju trebam osigurati da sustav bude osposobljen za rad u stvarnom vremenu.

1. Umjetne neuronske mreže

Umjetna neuronska mreža je skup međusobno povezanih jednostavnih procesnih jedinica, nazvanih umjetni neuron. Za razliku od simbolističkog pristupa umjetnoj inteligenciji, temelje se na konektivističkom pristupu. Takav pristup umjetnoj inteligenciji prirodan je za masovno raspodijeljeno i paralelno računanje[3]. Također, bitno je napomenuti da se umjetnoj neuronskoj mreži ne definira način obrade podataka. Ona se jednostavno uči na temelju njoj dostupnih podataka, odnosno iskustveno.

1.1. Umjetni neuron

Model umjetnog neurona prvi su definirali 1943. godine znanstvenici Warren McCulloch i Walter Pitts, na temelju modela biološkog neurona. Svaki umjetni neuron sastoji se od n ulaza, preko kojih prima ili vrijednosti neurona iz prethodnog sloja ili vrijednosti ulaznih podataka u neuronsku mrežu. Svaka ulazno vrijednost x_1, x_2, … do x_n množi se sa pripadnom težinom w_i. Skalarnom produktu vektora ulaznih podataka u neuron i pripadnih težina pridružuje se težina w_0, te se rezultat propušta kroz prijenosnu funkciju i prosljeđuje neuronima u sljedećem sloju. Važno je napomenuti da se u zadnjem sloju umjetne neuronske mreže ne koriste prijenosne funkcije[3]. Skica modela umjetnog neurona prikazana je slikom (Slika 1.1).



Slika 1.1 Model umjetnog neurona. Preuzeto sa:[3].

1.2. Treniranje umjetne neuronske mreže

Proces podešavanja težina, odnosno parametara svakog umjetnog neurona naziva se treniranje umjetne neuronske mreže. Na početku procesa treniranja parametri imaju nasumične vrijednost te se procesom učenja prilagođavaju ulaznim podacima. U slučaju kada model izgubi svojstvo generalizacije, odnosno dobro radi na podatkovnom skupu za treniranje ali se loše snalazi na dotad nikad viđenim podacima, kažemo da je došlo do prenaučenosti umjetne neuronske mreže. Cijeli podatkovni skup na kojem se provodi proces učenja uobičajeno se dijeli na 3 odvojena skupa:

- Skup za treniranje
- Skup za provjeru
- Skup za testiranje

1.3. Funkcija gubitka

Funkcije gubitka jedan su od osnovnih alata prilikom analize kvalitete naučene umjetne neuronske mreže, a govori nam koliko je model izgubio na točnosti[4]. Prilikom procesa treniranja, jedan od ciljeva je minimizirati funkciju gubitka modela. Na skupu za treniranje funkcija gubitka pada sa svakom epohom učenja, zbog čega se za analizu kvalitete naučenog modela koristi vrijednost funkcije gubitka na skupu za provjeru. U trenutku kada funkcija gubitka počinje rasti na skupu za provjeru, smatra se da je model izgubio mogućnost generalizacije te se prekida proces učenja.

1.4. Matrica zabune

Matrica zabune jedan je od češće korištenih alata za analizu kvalitete treniranih modela umjetnih neuronskih mreža. Prilikom analize modela za proces detekcije, ako je potrebno detektirati objekte iz n klasa, broj stupaca i redaka matrice jednak je broju klasa uvećanom za 1. Razlog tomu je što postoji mogućnost da se dio slike koji ne pripada nijednoj klasi, odnosno pripada pozadini slike, detektira kao da pripada nekoj klasi. Također je moguće da model ne detektira objekt koji je trebao detektirati, odnosno odredi da pripada pozadini slike.

2. Praćenje objekata na videozapisu

2.1. Euklidska udaljenost

Euklidska udaljenost 2 točke u dvodimenzionalnom Kartezijevom koordinatnom sustavu predstavlja udaljenost između 2 točke u koordinatnom sustavu[5]. U završnom radu koristi se za praćenje objekata na videozapisu.

2.2. Praćenje objekata na temelju njihovih središta

Algoritam za praćenje objekata na temelju njihovih središta računa euklidske udaljenosti između objekata detektiranih na 2 uzastopna okvira videozapisa te slijedno uparuje objekte s najmanjim euklidskim udaljenostima. Svaki objekt iz trenutnog okvira može biti uparen s najviše jednim objektom iz prethodnog okvira i obratno. U slučaju da je više objekata detektirano u trenutnom okviru u odnosu na prethodni, smatramo da su se pojavili novi objekti na videozapisu. U slučaju da je više objekata detektirano u prethodnom okviru u odnosu na trenutni, smatramo da su neki objekti nestali iz videozapisa.

3. Korišteni alati

3.1. Python

3.1.1. Općenito

Python je programski jezik više razine, razvijen tijekom 80-tih i 90-tih godina prošlog stoljeća od strane Guida van Rossuma. Međutim, tijekom vremena u njega su dodane mnoge nove funkcionalnosti. Zbog svoje jednostavnosti i lake čitljivosti jedan je od najpopularnijih programskih jezika današnjice[6]. Zbog svoje jednostavnosti, često se primjenjuje za treniranje neuronskih mreža te u sustavima računalnog vida. Međutim, velika mana je njegova relativna sporost u odnosu na ostale programske jezike, pogotovo jezike niže razine (npr. C i C++). Jedan od problema prilikom razvoja sustava bilo je upravo pisanje programskog koda na takav način da sustav može obrađivati videozapise u stvarnom vremenu.

3.1.2. Python moduli

Osnovna ideja prilikom izgradnje programskih sustava u Python-u je korištenje manjih programskih komada, zvanih moduli, koji se potom učitavaju u veće programe navođenjem ključne riječi `import`. Prilikom implementacije programskog rješenja korišteni su sljedeći javno dostupni moduli:

- scipy
- collections
- numpy
- ultralytics
- torch
- sys
- cv2

Također, napravljeni su i vlastiti moduli:

- videoGenerator
- centroidTracker

Modul `videoGenerator.py` sadrži klasu `VideoGenerator`, zaduženu za učitavanje videozapisa i modela za detekciju te obradu i stvaranje novog videozapisa. Modul `centroidTracker.py` sadrži implementaciju algoritma za praćenje objekata na temelju euklidskih udaljenosti središta vanjskih okvira objekata između uzastopnih okvira iz videozapisa.

3.1.3. Modul venv

S porastom broja dostupnih Python modula i paketa, porasla je i vjerojatnost da verzije javno dostupnih paketa dođu u konflikt, zbog različitih inačica istoimenih modula, koji se s vremenom nadograđuju te se uklanjaju uočene pogreške u njima. Zbog toga je izrazito bitan modul `venv`, iako se on u programsko rješenju ovog rada nigdje eksplicitno ne uvozi ključnom riječi `import`. Naime, prilikom organizacije radne okoline za razvoj sustava pomoću modula `venv` stvoreno je novo Python virtualno okruženje `zav_rad`. Svrha tog virtualnog okruženja je izolacija naše Python okoline od drugih mogućih Python projekata na našem računala. To nam omogućuje da prilikom razvoja programskog sustava uključimo točno one inačice Python modula koje nas zanimaju, te ne postoji opasnost od sukoba između inačica modula koje sada koristimo, te njihovih inačica koje se možda koriste u nekim drugim, već postojećim ili kasnije kreiranim, Python projektima.

3.2. OpenCV

OpenCV je javno dostupna biblioteka optimirana za rješavanje problema vezanih uz računalni vid. Biblioteka je otvorenog koda. Sadrži velik broj funkcija različitih namjena poput:

- Raspoznavanje boja na slikama
- Detekcija objekata
- Segmentacija slike
- Generiranje novih slika i videozapisa
- Crtanje po učitanim slikama
- Praćenje objekata na videozapisima[7]

Prilikom izrade završnog rada, korištene su dostupne funkcije za generiranje novog videozapisa te crtanje po učitanim slikama.

Budući da je sama biblioteka napisana u programskom jeziku C++, prilikom pozivanja funkcija u programskom jeziku Python zadržava se brzina izvođenja programskog jezika C++ [7].

3.3. NumPy

NumPy je biblioteka otvorenog koda korištena u programskom jeziku Python, prvi put objavljena 2005. godine. Osnovni razlog razvoja biblioteke je bilo ubrzavanje postupaka računanja u programskom jeziku Python[8].

3.4. PyTorch

PyTorch je popularan radni okvir otvorenog koda namijenjen korišten za razvoj aplikacija u Python-u koje koriste strojno učenje. Razvijen je u Pythonu, a održava ga Linux Foundation[9]. Iako se inače najčešće koristi za razvoj i treniranje modela neuronskih mreža, prilikom izrade ovog završnog rada primarno je korišten zbog svojeg efikasnog korištenja grafičkih procesorskih jedinica prilikom treniranja modela umjetnih neuronskih mreža te korištenja modela za detekciju objekata na novim videozapisima. Aktivacijom grafičkih procesorskih jedinica značajno ubrzava proces obrade svake pojedine slike, budući da posao detekcije prebacuje s centralne procesorske jedinice računala, koja nije prilagođena obradi slika korištenjem umjetnih neuronskih mreža. Ovo svojstvo omogućuje rad sustava za detekciju i praćenje u stvarnom vremenu.

3.5. Ultralytics YOLOv8

YOLOv8 je algoritam za detekciju i segmentaciju objekata na slikama te klasifikaciju slika. Pripada familiji YOLO (You Only Look Once) algoritama. Implementiran je koristeći biblioteku PyTorch. Time je omogućeno mnogo različitih mogućnosti prilikom treniranja vlastitih modela neuronskih mreža, poput prijenosa težina te do-treniranja prije naučenih modela vlastitim podatkovnim skupovima. Tvrtka Ultralytics, tvrtka koja je razvila algoritam te ga održava, dizajnirala je nekoliko različitih YOLOv8 arhitektura, koje

se međusobno razlikuju po svojoj kompleksnosti i brzini izvođenja prilikom detekcije objekata na slikama. Tablica usporedbe različitih YOLOv8 arhitektura vidljiva je na slici .

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

Slika 3.1 Usporedba različitih YOLOv8 arhitektura treniranih na COCO podatkovnom skupu. Vidljivo je da arhitektura YOLOv8n ima najveću brzinu detekcije te najmanji broj parametara za treniranje, skoro 22 puta manje parametara od modela YOLOv8x. Preuzeto sa:[11].

3.6. Roboflow

Roboflow je alat za stvaranje i dijeljenje podatkovnih skupova namijenjenih treniranju različitih modela neuronskih mreža, a koje se koriste za rješavanje problema iz područja računalnog vida. Također nudi mogućnost augmentacije postojećeg skupa za treniranje stvaranjem novih slika primjenom odabranih transformacija nad početnim slikama. Razvila ga je te ga održava istoimena tvrtka.

4. Implementacija sustava

4.1. Priprema skupa podataka za treniranje modela

Prvi korak prilikom implementacije rješenja bio je odabir pogodnog modela koji bi se trenirao za detekciju igrača te ostalih bitnih aktera na slikama, za što je odabran YOLOv8n model. Nakon odabira modela bilo je potrebno pripremiti podatkovni skup te ga podijeliti na sljedeće skupove, u omjeru 70-20-10:

- Skup za treniranje
- Skup za provjeru
- Skup za testiranje

Kako bi se postigla maksimalna nasumičnost te isključila moguća pristranost autora rada, prepušteno je alatu Roboflow da on sam podijeli slike na zadane skupove, u zadanom omjeru. Kao konačan cilj postavljeno je označavanje 1000 slika sa 4 različite utakmice, a trenirani su različiti modeli na podatkovnim skupovima bez dodatnih augmentacija, ali i oni s augmentacijama. Videozapisi utakmica su javno dostupni, a snimljeni su 2023. godine.

S obzirom na prirodu sporta te česte kontakte među igračima, često se događa da jedan dio igrača zakriven drugim igračem, ili nekim drugim akterom na slici. Prilikom označavanja vanjskih okvira objekata, označavani su i skriveni dijelovi objekata, zbog čega često dolazi do preklapanja između područja koja pripadaju različitim objektima.

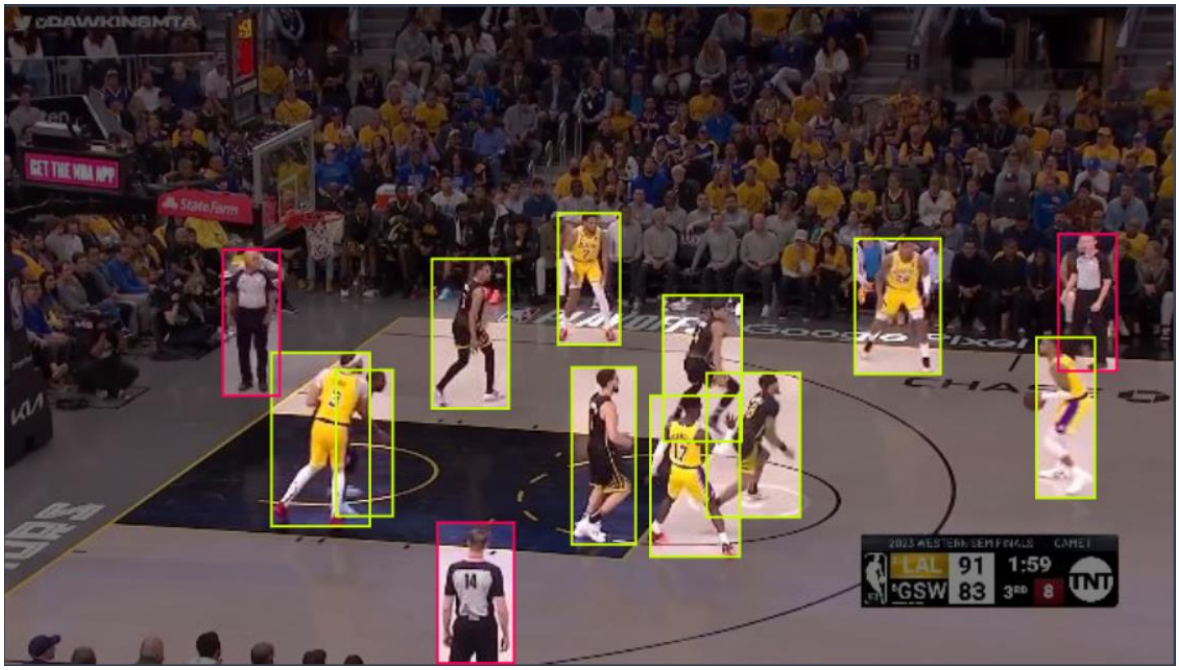
Prilikom analize problema identificirane su 4 klase zanimljive našem sustavu:

- *player*
- *coach*
- *referee*
- *scored_basket*

Klasa *player* predstavlja igrače koji su u aktivnoj ulozi na terenu, odnosno u skupu se nalaze svi igrači koji sudjeluju u igri, a ignoriraju se oni igrači na klupi, izvan terena. Ta klasa nam je najbitnija prilikom kreiranja podatkovnih skupova. Klasa *coach* predstavlja

sve trenere momčadi prisutne na slikama, a uvedena je budući da treneri često provode vrijeme hodajući uz teren. Uvođenjem ove klase uvodimo dodatnu distinkciju sustavu, kako bismo potencijalno smanjili broj lažno-pozitivnih detekcija klase *player*. S istom namjerom uvedena je klasa *referee*, kojoj pripadaju svi košarkaši suci na terenu. Oni se, naime, od igrača razlikuju po svojim karakterističnim radnim uniformama crne i sive boje. Posljednja klasa, *scored_basket*, služi za identifikaciju trenutaka u kojima su postignuti koševi. To se može raspoznati karakterističnim oblicima košarkaških mrežica u trenutku prolaska lopte kroz obruč i mrežicu. Ovoj klasi nije posvećena velika pozornost prilikom kreiranja podatkovnih skupova, ali se može iskoristiti prilikom nadogradnje sustava praćenja igrača.

Očekivani broj detektiranih objekata na slici je 0 do 10 objekata klase *player*, 0 do 3 objekta klase *referee*, 0 do 2 objekta klase *coach* te 0 do 1 objekt klase *scored_basket*. Primjer označavanja područja koja pripadaju pojedinim klasama na jednoj slici iz podatkovnog skupa prikazan je slikom (Slika 4.1). Vidljivo je 10 objekata klase *player*, označenih pravokutnicima sa žutim bridovima, te 3 objekta klase *referee*, označenih pravokutnicima sa crvenim bridovima.

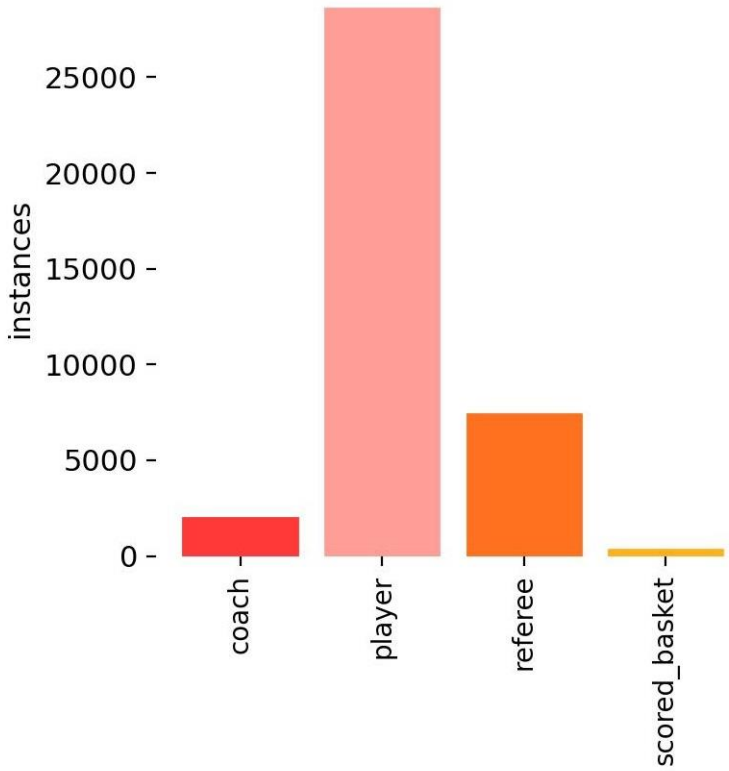


Slika 4.1 Primjer označenih objekata klase player te klase referee na jednoj slici, u alatu Roboflow

Konačan podatkovni skup sadržavao je 1000 vlastito označenih slika, te sadrži sljedeći broj objekata svake klase:

- 8647 objekata klase *player*
- 2237 objekata klase *referee*
- 588 objekata klase *coach*
- 103 objekta klase *scored_basket*.

Međutim, primjenom augmentacije *Change exposure* broj slika je povećan na 3612. Stoga konačan podatkovni skup sadrži znatno veći broj objekata svake klase, koji je vidljiv na slici (Slika 4.2).



Slika 4.2 zastupljenost pojedinih klasa u skupu za treniranje

4.2. Treniranje modela

4.2.1. Odabir modela i strategija razvoja modela

Za treniranje modela konkretno je odabran model YOLOv8n iz familije YOLOv8 modela. U odnosu na ostale modele, najjednostavniji je te ima najmanji broj parametara za

treniranja, ali je isto tako i najbrži prilikom detektiranja objekata na slici. To nam je izuzetno bitno, budući da je poželjno svojstvo našeg sustav sposobnost rada u stvarnom vremenu.

Programski kod za treniranje modela sastoji se od 3 dijela

- Učitavanje potrebnih modula te pripremanje grafičke procesorske jedinice za rad, radi ubrzavanja proces treniranja
- Izgradnja yolov8 modela te prijenos već dostupnih težina svakog neurona
- Treniranje modela na našem podatkovnom skupu

```
from ultralytics import YOLO
import torch
device = "0" if torch.cuda.is_available() else "cpu"
if device == "0":
    torch.cuda.set_device(0)

model = YOLO('yolov8n.yaml').load('yolov8n.pt')
results = model.train(data='NBAv3/data.yaml', epochs = 20,
imgsz = 640, amp = False, val = True, batch = 2)
```

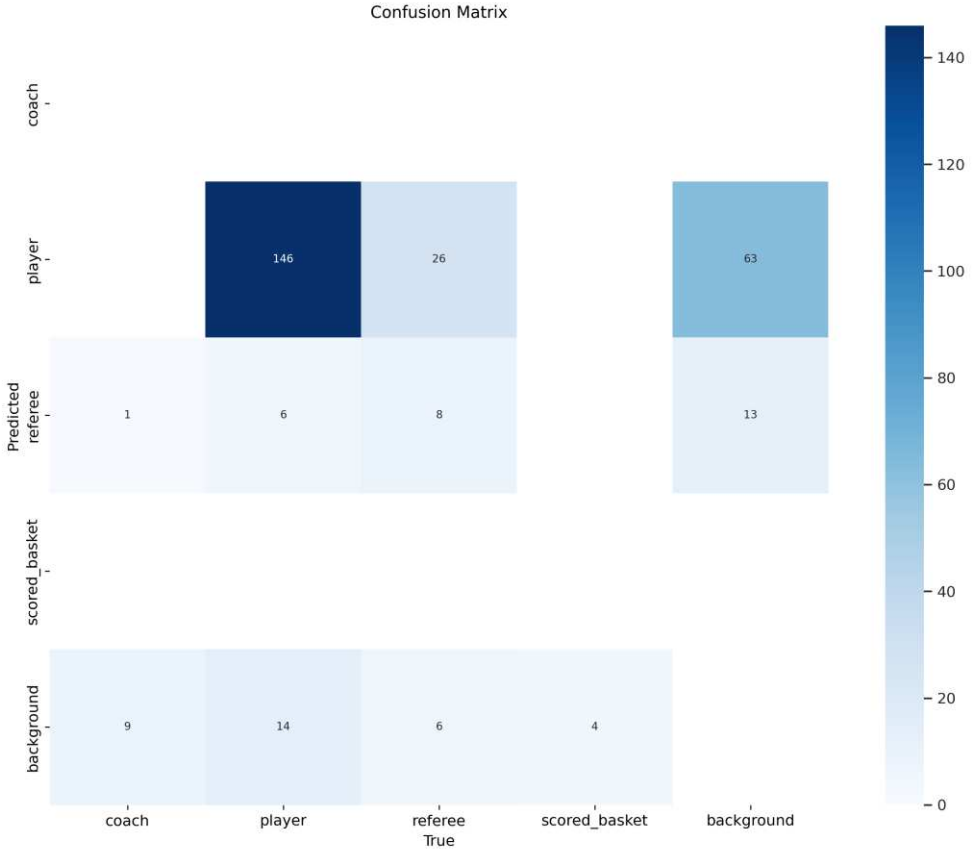
Gornji isječak koda prikazuje treniranje modela na jednom podatkovnom skupu. Nakon pripreme grafičke kartice modela s njegovim početnim težinama za treniranja, pokreće se sam process treniranja modela pozivanjem metode `train`. Kao početni broj epoha za treniranje odabran je broj 20, što se pokazalo dovoljno dobrim za to da model nauči dovoljno, ali opet zadrži sposobnost generalizacije. Predviđena veličina ulaznih slika je 640x640 piksela, ali to nema nikakav utjecaj prilikom proslijeđivanja modelu dotad neviđenih slika, budući da im on sam promijeni veličinu. Također, svakom modelu je prag pouzdanosti automatski postavljen na 0.25.

Sam proces razvoja modela za detekciju objekata podijelili smo u dvije faze. U prvoj fazi model je treniran na manjem podatkovnom skupu od 70 slika (70% podatkovnog skupa koji sadrži ukupno 100 slika). Razlog za to je bila identifikacija slabosti modela, potencijalno uzrokovanih lošim odabirom strategije označavanja objekata na slikama, te primjerena reakcija prilikom označavanja ostatka slika od ciljanih 1000. Prilikom analize modela najveća je pozornost posvećena sljedećim metrikama:

- Matrica zabune
- Normalizirana matrica zabune
- F1-pouzdanost krivulja

4.2.2. Analiza metrika prve varijante modela

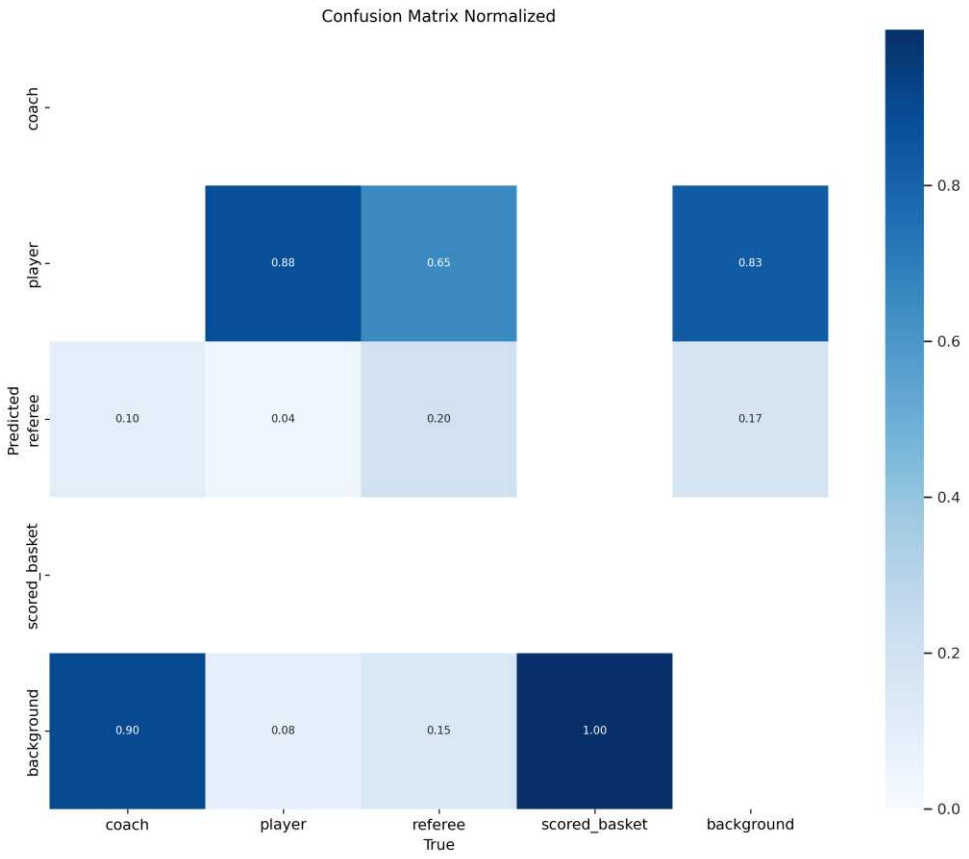
Iz matrice zabune prvog treniranog modela, prikazane slikom (Slika 4.3), može se zaključiti da model dobro detektira gdje se nalaze objekti klase *player*, ali također ima zamjetan broj lažno-pozitivnih detekcija, budući da često zamjenjuje pozadinu za klasu *player*. Budući da je naš uzorak malen, zbog toga što naš skup za provjeru u ovom trenutku sadrži samo 20 slika, analizirajmo normaliziranu matricu zabune, prikazanu slikom (Slika 4.4).



Slika 4.3 Matrica zabune prve varijante modela

Proučavanjem normalizirane matrice zabune prve varijante modela, možemo zaključiti da velik udio lažno-pozitivnih detekcija pozadine slika pripada upravo klasi *player*. Također, detekcija ostalih objekata na slikama ne radi dobro. Tomu je najvjerojatniji uzrok upravo veličina podatkovnog skupa, budući da na objektima kojih ima više, poput klase *player*,

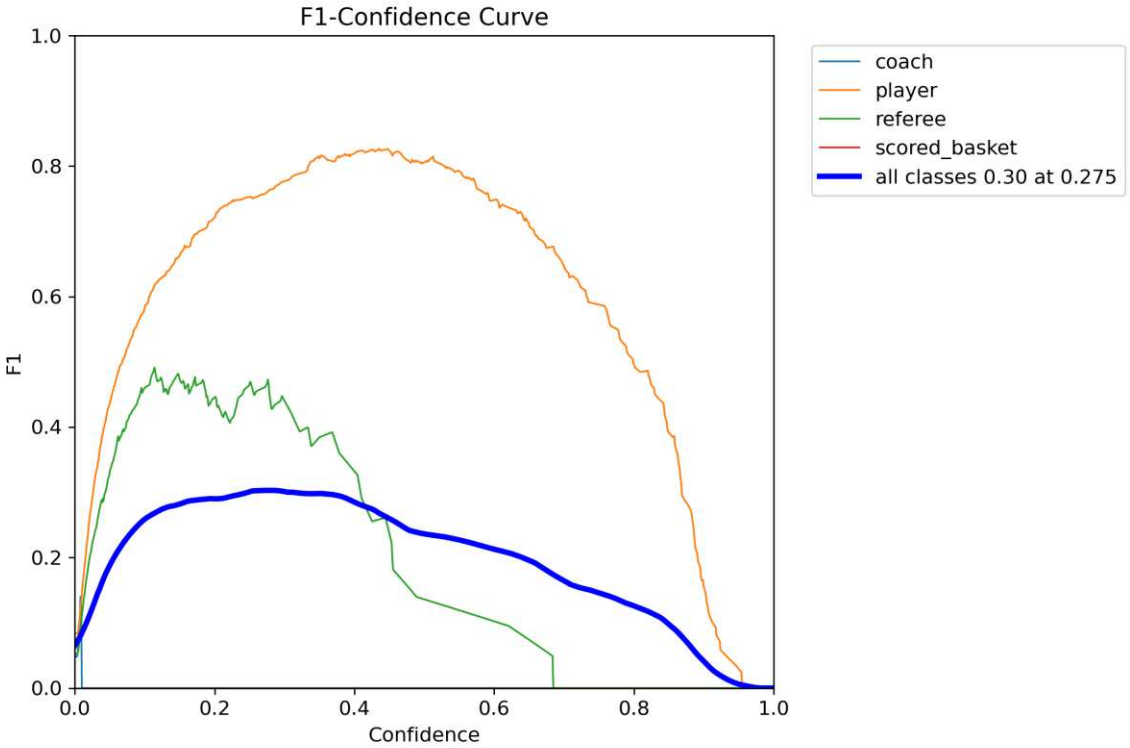
model 88% objekata koji zaista pripadaju toj klasi ispravno detektira. Odnosno, djeluje da je naša strategija označavanja objekata na slikama ispravna, ali da ipak trebamo povećati podatkovni skup.



Slika 4.4 Normalizirana matrica zabune prve varijante modela

Međutim, kako bismo bili sigurni u tu odluku, analizirajmo još krivulju F1 - pouzdanost, prikazanu slikom (Slika 4.5).

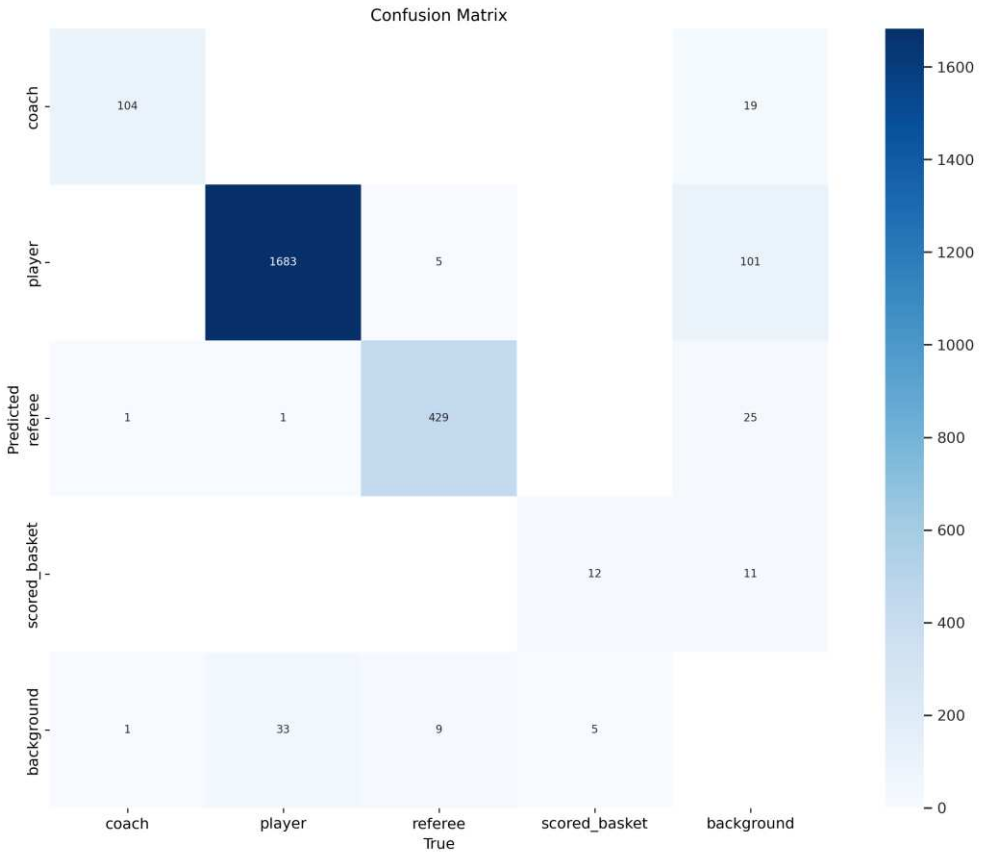
Na temelju F1 - confidence krivulje, vidimo da je optimalna vrijednost minimalne pouzdanosti našeg modela 0.275, zato što model za tu vrijednost postiže najveću F1 vrijednost, iznosa 0.3. Međutim, isto tako je primjetno da objekti klase *player* postižu primjetno višu F1 vrijednosti na skoro cijelom intervalu od 0 do 1. Također, objekti klase *coach* na skoro cijelom intervalu postižu vrijednost 0. To definitivno ide u prilog našoj tezi da je strategija označavanja objekata na slikama dobra te da samo trebamo povećati naš podatkovni skup. Donosimo konačan zaključak da je naša strategija dobra te uz ista pravila označavanja objekata kao i do sada dodajemo nove slike u podatkovni skup.



Slika 4.5 F1 - pouzdanost krivulja prve varijante modela

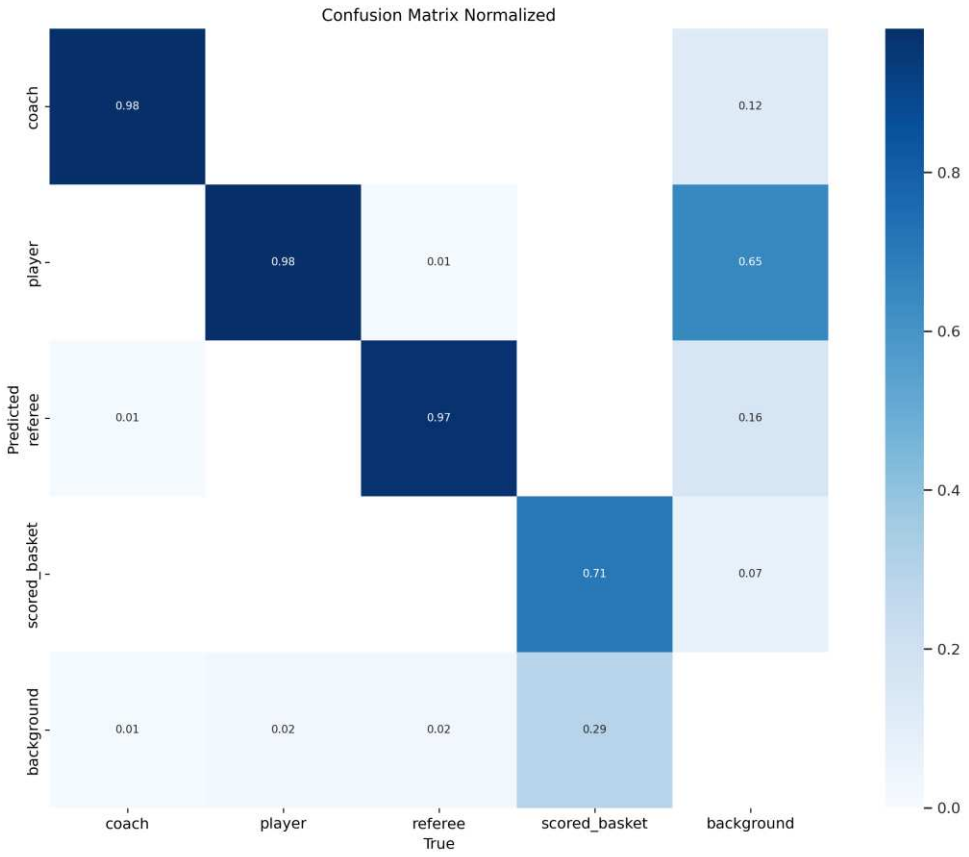
4.2.3. Analiza metrika druge varijante modela

Iz matrice zabune druge varijante modela, prikazane slikom (Slika 4.6), odmah se može primijetiti da sada imam puno veći i relevantniji skup za provjeru. Primjetan je pad udjela lažno-pozitivnih detekcija objekata iz klase *player*. Naime, nakon treniranja prve verzije modela taj udio je iznosio 0.38, a sada iznosi 0.06. 101 put je objekt iz klase *background* pogrešno označen kao da pripada klasi *player* te je 5 puta objekt klase *referee* označen kao da pripada klasi *player*. Međutim, čak 1683 puta je objekt klase *player* ispravno detektiran i označen. Međutim, udio lažno-pozitivnih detekcija objekata klase *scored_basket* je i dalje dosta visok te iznosi 0.48. Sve lažne-pozitivne detekcije ove klase svode se na situaciju da model pogrešno detektira pozadinu kao postignut koš, odnosno nikad ne označi pogrešno objekt klase *player*, *coach* ili *referee* kao objekt koji pripada klasi *scored_basket*.



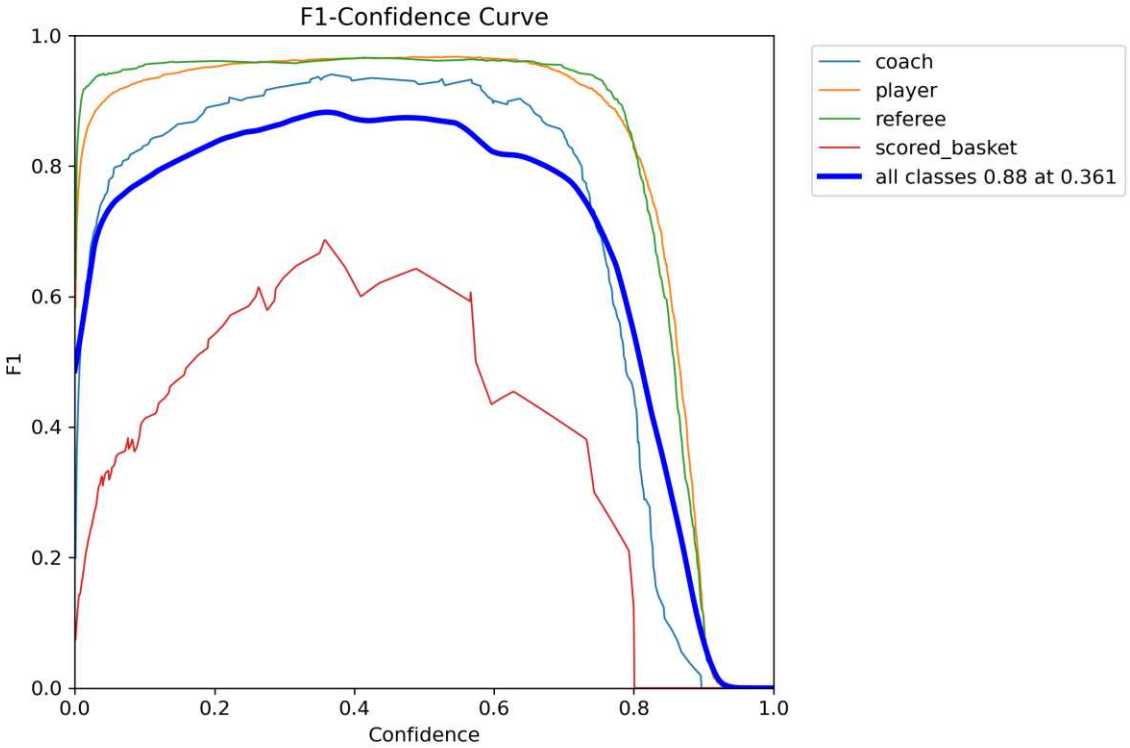
Slika 4.6 Matrica zabune druge varijante modela

Analizom normalizirane matrice zabune druge varijante modela, prikazane slikom (Slika 4.7), primjetan je značajan skok u točnosti detekcije za sve označene klase. Udio točno detektiranih objekata koji pripadaju klasama *player*, *coach* i *referee* iznosi preko 0.95. Jedina klasa za koju je postignut nešto lošiji rezultat je *scored_basket*, za koju je udio ispravno detektiranih objekata 0.71. Najveći skok u odnosu na prvu varijantu ostvaren je za klase *coach* i *referee,* kod kojih je u prvoj varijanti modela udio Istinito-pozitivnih objekata iznosio 0.1 odnosno 0.2, a kod druge varijante iznosi 0.98, odnosno 0.97. Kod pogrešnih detekcija pozadine, i dalje se najčešće detektira da dio pozadine pripada klasi *player*, što čini čak 0.65 svih pogrešnih detekcija pozadine. Međutim, koristeći znanje stečeno analizom obične matrice zabune ovog modela, znamo da je taj udio jako malen u ukupno broju objekata za koje je detektirano da pripadaju klasi *player*.



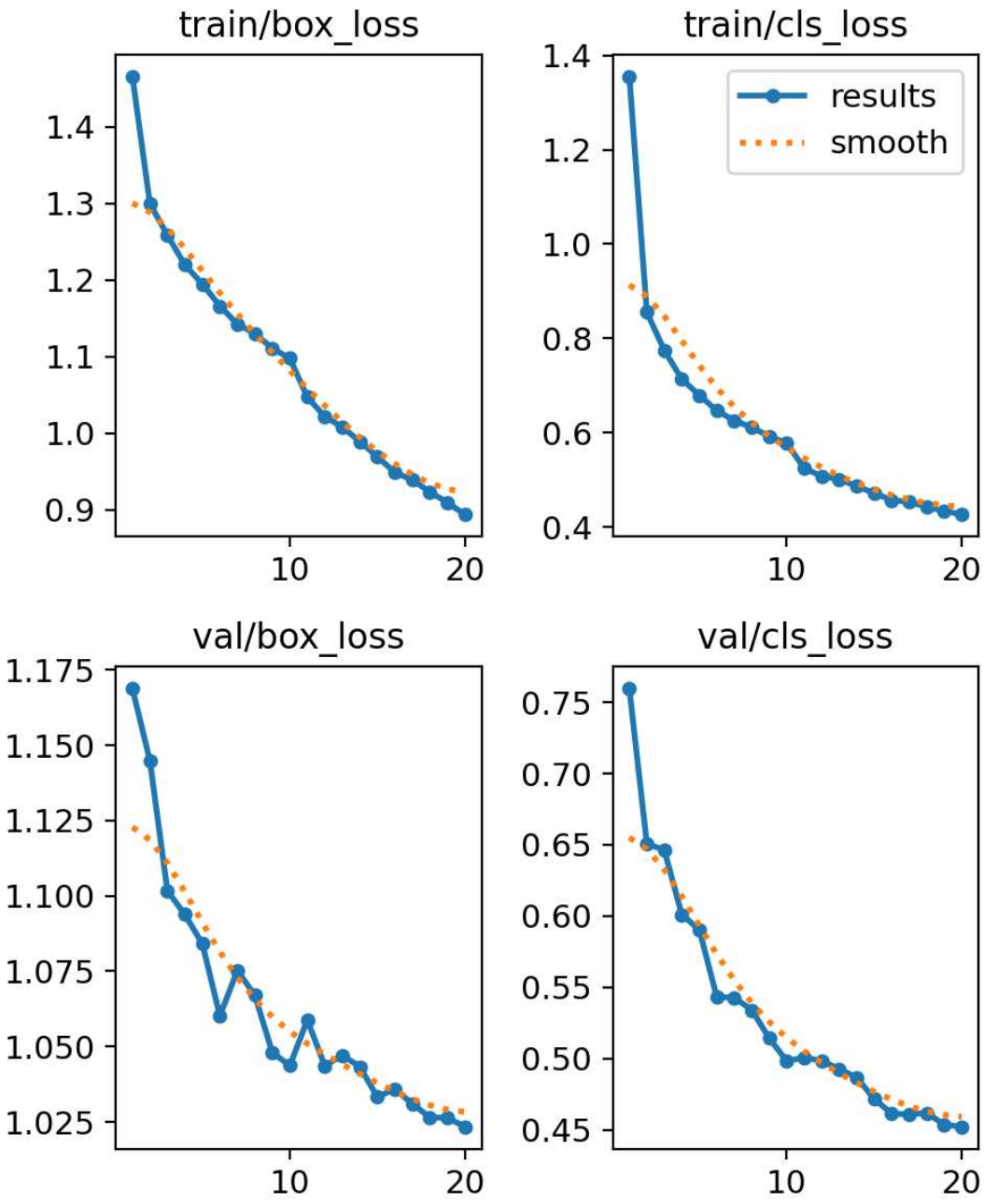
Slika 4.7 Normalizirana matrica zabune druge varijante modela

Konačno, analizom F1 – pouzdanost krivulje, primjetan je značajan rast F1 vrijednosti svih klasa za vrijednosti varijable Confidence manje od 0.8. Najveću F1 vrijednost cijeli model postiže za vrijednost varijable Confidence 0.361, a F1 vrijednost je 0.88, što je 2.96 puta veće od F1 vrijednosti prve varijante modela. Na temelju analize sve tri metrike, možemo zaključiti da performanse modela s većim podatkovnim skupom te iskorištenim augmentacijama nadmašuju performanse prve varijante modela. Također, naša strategija označavanja slika se pokazala dobrom.



Slika 4.8 F1 - pouzdanost krivulja druge varijante modela

Međutim, dosad analizirane metrike nam ne daju potpunu informaciju o tome kolika je sposobnost generalizacije našeg modela, te je li možda potrebno smanjiti broj epoha, kako bismo povećali njegovu moć generalizacije, odnosno funkcionalnosti na videozapisima utakmica koje dotad nije vidio. Kako bismo donijeli odluku o tome generalizira li naš model dovoljno dobro, pogledajmo vrijednost funkcije gubitka u ovisnosti i trenutnoj epohi treniranja modela (Slika 4.9). Za vrijednost funkcije gubitka prilikom određivanja klase objekta, primjećujemo da vrijednost konzistentno pada i na testu za treniranje i na testu za validaciju. Odnosno naša model je i dalje zadržao sposobnost generalizacije prilikom određivanja klase kojoj detektirani objekt pripada, te bi trebao raditi i na nikad viđenoj slici. Ista stvar vrijedi i za određivanja vanjskih okvira detektiranih objekata. Vrijednost funkcije gubitka konzistentno pada i na podatkovnom skupu za treniranje i na podatkovnom skupu za provjeru. Donosimo zaključak da naš model radi dobro te da nije prenaučen, što znači da ne moramo smanjiti broj epoha prilikom proces treniranja modela.



Slika 4.9 Graf ovisnosti funkcije gubitka o broju epoha treniranja

4.3. Detekcija i praćenje igrača

Dio programa koji generira obrađen videozapis s oznakama praćenih igrača nalazi se u modulu `videoGenerator.py`. U tom modulu odvija se učitavanje svakog pojedinog okvira videozapisa te detekcija ključnih aktera u sličici, a algoritam za praćenje je implementiran u modulu `centroidTracker.py`.

4.3.1. Praćenje objekata pomoću njihovih središnjih točaka

Za praćenje objekata na dva uzastopna okvira videozapisa zadužen je objekt iz klase `CentroidTracker`. Prilikom konstrukcije objekta iz pripadne klase, definiraju se atributi nužni za ispravan rad. Atribut `next_player_id` ima početnu vrijednost 0, a bilježi brojčanu oznaku koju ćemo pridružiti sljedećem, do tog trenutka nedetektiranom objektu kojeg je YOLO8 model pridružio klasi *player*. Atribut `detected_players` uređeni je rječnik, gdje se kao ključ sprema identifikator koji smo pridružili praćenom objektu, a vrijednost je uređeni par kojim je određeno središte objekta. Budući da je moguće da model neuronske mreže na nekim kadrovima pogriješi te ne detektira neki objekt klase *player*, definiramo kao jedan od atributa uređeni rječnik `disappeared_players`, kojemu je također ključ identifikator igrača kojeg smo detektirali na prošlim sličicama, a vrijednost trenutni uzastopni broj sličica na kojima objekt nije bio detektiran. Kako ne bismo trajno spremali sve igrače, pa čak i one koji su zaista izašli iz okvira videozapisa, definiramo atribut `maximum_frames_before_declared_disappeared`. On određuje tijekom koliko najviše uzastopnih okvira videozapisa objekt može ostati nedetektiran, prije nego što ga prestanemo pratiti.

Glavni dio algoritma za praćenje nalazi se u metodi `updateTrackingInfo`. Ona kao argument prima listu uređenih četvorki, a u svakoj uređenoj četvorci su prva 2 elementa koordinate gornjeg desnog kuta te druga 2 elementa koordinate donjeg desnog kuta. Ako naš model nije detektirao nijedan objekt u trenutnom okviru, jednostavno svim objektima detektiranima u prošlom okviru povećamo broj uzastopnih okvira na kojima nisu bili detektirani za 1.

```
if len(bounding_boxes) == 0:
      for player_ID in list(self.detected_players.keys()):
                self.disappeared_players[player_ID] += 1
```

Dodatno, ako je broj uzastopnih okvira na kojima igrač nije detektiran dosegnuo prag definiran atributom `maximum_frames_before_declared_disappeared`, izbrišemo zapis o njemu pozivanjem metode `deregisterPlayer`.

```
def deregisterPlayer(self, player_ID):
        del self.detected_players[player_ID]
        del self.disappeared_players[player_ID]
```

U slučaju da su u trenutnom okviru detektirani traženi objekti, na temelju koordinata gornjeg lijevog te donjeg desnog kuta izračunamo koordinate središta objekta kao aritmetičke sredine odgovarajućih parova koordinata.

Dalje razlikujemo 2 slučaja:

- Trenutno ne pratimo nijedan objekt te ne postoje objekti koji su nam zapisani kao privremeno nestali, odnosno navedeni u atributu `disappeared_players`
- Postoji barem jedan objekt detektiran na prethodnom okviru ili detektiran u zadnjih `maximum_frames_before_declared_disappeared` okvira

Ako vrijedi prvi slučaj, jednostavno slijedno prolazimo po svakome objektu, pridružimo mu njegov jedinstveni identifikacijski broj te zabilježimo koordinate njegova središta u prikladnu strukturu podataka, koristeći metodu `registerPlayer`.

```
def registerPlayer(self, centroid):
        self.detected_players[self.next_player_ID] = centroid
        self.disappeared_players[self.next_player_ID] = 0
        self.next_player_ID += 1
```

U slučaju da postoje objekti detektirani u prošlom okviru, ili objekti koji se smatraju privremeno nestalima, potrebno je prvo izračunati Euklidske udaljenosti između središta trenutno detektiranih objekata te onih detektiranih na prethodnom okviru.

```
centroid_distances =
distance.cdist(numpy.array(old_player_centroids),
input_centroids)
```

Pretpostavimo da su indeksi redaka označeni slovom *i*, a indeksi stupaca označeni slovom *j*. Rezultat poziva funkcije `distance.cdist` je matrica u kojoj je vrijednost elementa u *i*-tom retku i *j*-tom stupcu jednaka euklidskoj udaljenosti između elementa liste `old_player_centroids` s indeksom *i* te elementa liste `input_centroids` s indeksom *j*.

U svakom retku pronalazimo najmanju udaljenost te zabilježimo u listu indekse redaka sortirane uzlazno po najmanjim vrijednostima u pojedinom retku. Odnosno, za svako

prethodno zabilježeno središte objekta odredimo koja je najmanja udaljenost između njega i bilo kojeg novo detektiranog središta objekta te sortiramo indekse redaka u kojim se nalazi pojedini prethodno zabilježeni objekt uzlazno po minimalnim udaljenostima.

```
distances_rows = centroid_distances.min(axis = 1).argsort()
```

Zatim slično pronalazimo najmanje udaljenosti u svakom stupcu, odnosno za svaki novo detektirani objekt tražimo indeks retka prethodno detektiranog objekta koji mu je najbliži. Odabrane retke zatim poredamo onim redoslijedom kojim su navedeni indeksi redaka u listi `distances_rows`. Na primjer, ako je prvi element liste `distances_rows` broj 2, onda će prvi element liste `distances_columns` biti indeks retka s najmanjom vrijednosti u trećem stupcu matrice.

```
distances_columns = centroid_distances.argmin(axis =
1)[distances_rows]
```

Preostaje upariti objekte detektirane u prethodnom okviru s onima detektiranima u trenutnome. Pri tome moramo paziti na to se svaki indeks retka može upariti isključivo s jednim indeksom stupca i obratno. Odnosno, svakom objektu detektiranom u trenutnom okviru odgovara najviše jedan objekt detektiran u prethodnomu. Kako bismo sačuvali podatke o već iskorištenim redcima i stupcima, definiramo dvije strukture podataka tipa skup, `used_rows` i `used_columns`.

```
for (row, column) in zip(distances_rows, distances_columns):
      if row in used_rows or column in used_columns:
            pass
      else:
            player_ID = old_player_IDs[row]
            self.detected_players[player_ID] = input_centroids[column]
            self.disappeared_players[player_ID] = 0

            used_rows = used_rows | {row}
            used_columns = used_columns | {column}
```

S obzirom da broj detektiranih objekata u uzastopnim okvirima ne mora biti konstantan, razlikujemo 2 moguće situacije:

- Broj detektiranih objekata u trenutnom okviru je veći od broja detektiranih objekata u prethodnome

- Broj detektiranih objekata u trenutnom okviru je manji ili jednak broju detektiranih objekata u prethodnome

Ako je broj detektiranih objekata u trenutnom kadru veći od broja detektiranih objekata u prethodnom okviru, odnosno ako je broj stupaca matrice udaljenosti veći od broja redaka matrice, jednostavno prolazimo po svim objektima koji nisu pridruženi nijednom prethodnom detektiranom objektu te ih zabilježimo pozivom metode `registerPlayer`.

```
for column in unused_columns:
      self.registerPlayer(input_centroids[column])
```

Inače, prolazimo po svim neiskorištenim redcima matrice te zabilježimo da objekt nije detektiran u trenutnom okviru. Dodatno, ako je broj uzastopnih sličica na kojima se objekt nije pojavio dosegnuo vrijednost određenu atributom `maximum_frames_before_declared_disappeared`, potrebno je izbrisati podatke o tom objektu.

```
 for row in unused_rows:
      disappeared_player_ID = old_player_IDs[row]
      self.disappeared_players[disappeared_player_ID] += 1

      if self.disappeared_players[disappeared_player_ID] >=
self.maximum_frames_before_declared_disappeared:

            self.deregisterPlayer(disappeared_player_ID)
```

Metoda na kraju vraća informacije o trenutnom praćenim objektima, atribut `detected_players`.

4.3.2. Detekcija igrača u kadru i generiranje videozapisa

Za analizu i generiranje novog videozapisa, uz objekt klase `CentroidTracker`, korišteni su moduli `cv2` i `ultralytics`. Modul `cv2` sadrži funkcije potrebne za čitanje videozapisa i generiranje novog videozapisa, a `ultralytics` sadrži funkcije potrebne za detekciju objekata. Prilikom inicijalizacije objekta klase VideoGenerator, definira se vrijednost atributa koji određuje kanal iz kojeg program čita videozapis.

```
self.input_video = cv2.VideoCapture(input_video)
```

Osim toga, potrebno je postaviti vrijednosti koje određuje korišteni model neuronske mreže za detekciju, te parametre vezane uz crtanje vanjskih okvira na generiranom videozapisu. Atribut `model` je instanca modela za detekciju objekata koji se koristi, a `tracker` je instanca klase `CentroidTracker`, zadužene za praćenje objekata na videozapisu.

```
self.model = YOLO(input_model, task = "detect")
self.tracker = CentroidTracker(10)
self.color = (255, 255, 0)
self.thickness = 2
```

Metoda `processVideo` obavlja samu obradu videozapisa, te kao rezultat sprema na računalo i vraća obrađeni videozapis, spremljen u .mp4 formatu. Na početku je potrebno definirati parametre novog videozapisa:

- Ime datoteke gdje se sprema generirani videozapis
- Format videozapisa
- Broj sličica u sekundi
- Veličinu svake pojedine sličice u videozapisu u pikselima

```
self.fps = self.input_video.get(cv2.CAP_PROP_FPS)
self.frame_size = (int(self.input_video.get(3)),
int(self.input_video.get(4)))
self.processed_video =
cv2.VideoWriter(self.processed_video_path,
cv2.VideoWriter_fourcc('m', 'p', '4', 'v'), int(self.fps),
self.frame_size)
```

Broj sličica u sekundi rezultantnog videozapisa jednak je onome kod ulaznog videozapisa, kao i dimenzije svake pojedine sličice. Jedino po čemu se ulazni i generirani videozapis mogu razlikovati je format spremanja.

Metoda prolazi slijedno po videozapisu, sličicu po sličicu, te unutar okvira detektira objekte zadane modelom. Budući da je naš model treniran za detekciju više vrsta objekata, a ne samo igrača, potrebno je filtrirati dobivene rezultat vanjske okvire. Nakon što odaberemo one vanjske okvire objekata koji su određeni da pripadaju klasi *player*, nacrtamo vanjske okvire na sličici te ih proslijedimo atributu `tracker`, zaduženom za praćenje objekata.

```
bounding_boxes_tensors = frame_detections[0].boxes.xyxy
bounding_boxes_classes = frame_detections[0].boxes.cls
bounding_boxes_tuples = [tuple(int(coordinate) for coordinate
in box) for box in bounding_boxes_tensors.cpu().numpy()]
bounding_boxes_classes = [int(predicted_class) for
predicted_class in bounding_boxes_classes]
player_bounding_boxes = [bounding_boxes_tuples[idx] for idx
in range(len(bounding_boxes_tuples)) if
bounding_boxes_classes[idx] == 1]

for player in player_bounding_boxes:
      self.current_frame = cv2.rectangle(self.current_frame,
player[:2], player[2:], self.color, self.thickness)

tracked_players_IDs =
self.tracker.updateTrackingInfo(player_bounding_boxes)
```

Nakon što atribut `tracker` iz klase `CentroidTracker` vrati uređeni rječnik u kojem je ključ identifikacijska oznaka igrača, a vrijednost koordinate središta vanjskog okvira koji određuje igrača, potrebno je na sličicu novogeneriranog videozapisa staviti pripadni tekst, kako bismo mogli pratiti igrače.

```
for (player_ID, coordinates) in tracked_players_IDs.items():
      player_text = f"player_{player_ID}"
      cv2.putText(self.current_frame, player_text,
tuple(coordinates), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
self.color, self.thickness)

self.processed_video.write(self.current_frame)
```

Na kraju, kada su obrađene sve sličice videozapisa, prekida se čitanje i pisanje po datotekama te se otpuštaju korištene datoteke.

```
self.processed_video.release()
self.input_video.release()
```

4.4. Aplikacija za obradu videozapisa

Programski kod samo aplikacije, koja koristi usluge modula VideoGenerator.py, nalazi se u glavnom modulu `player_tracking.py`. Nakon što se učitaju potrebni moduli te se aktivira grafička procesorska jedinica korištenjem modula `torch`, započinje sama obrada videozapisa.

```
if __name__ == "__main__":
    device = "0" if torch.cuda.is_available() else "cpu"
    if device == "0":
        torch.cuda.set_device(0)
```

Kako bismo pripremili grafičku procesorsku jedinicu za obradu videozapisa, prvo se YOLOv8n modelu *v1_best.pt* proslijedi slika iz direktorija, nebitno o njenom sadržaju. Razlog za to je što grafičkoj procesorskoj jedinici, neovisno o tome koji model koristi, na početku uvijek treba znatno više vremena za obradu slike. Time postižemo da obrada videozapisa teče neometano, s najboljim mogućim performansama, a sve kako bi sustav radio u stvarnom vremenu.

```
bootup_model = YOLO('models/v1_best.pt')
bootup_results = bootup_model(["GSWLakers_6005.jpg"])
```

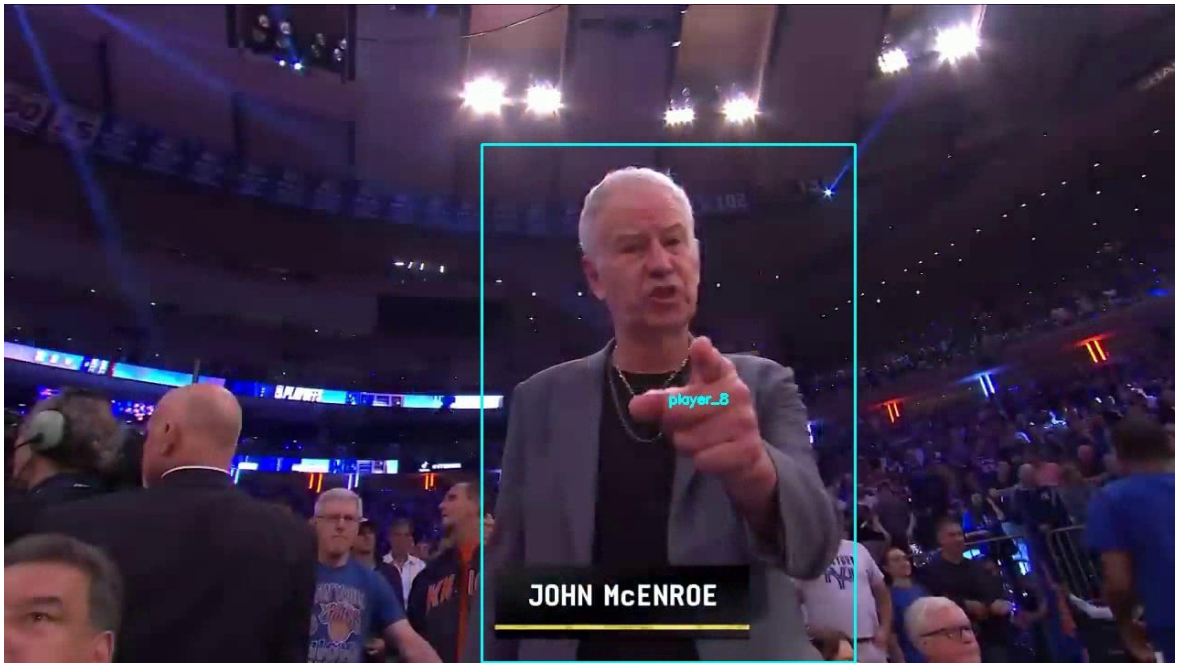
Na kraju preostaje samo učitati argumente poslane preko naredbene linije aplikaciji, proslijediti ih objektu zaduženom za obradu proslijeđenog i generiranje novog videozapisa, te konačno pozvati naredbu za obrađivanje videozapisa.

```
model_path = sys.argv[1]
videofile_path = sys.argv[2]
videoGenerator = VideoGenerator(videofile_path, model_path)
processed_video = videoGenerator.processVideo()
```

5. Primjeri obrađenih videozapisa

Obrađeni videozapis trebao bi imati istu rezoluciju i broj sličica u sekundi kao i ulazni videozapis te bi na njemu igrači trebali biti uokvireni pravokutnicima plave boje sa svojim jedinstvenim oznakama u središtu. U nastavku su prikazani primjeri slika jednog obrađenog javno dostupnog videozapisa, snimljenog tijekom 2024. godine.

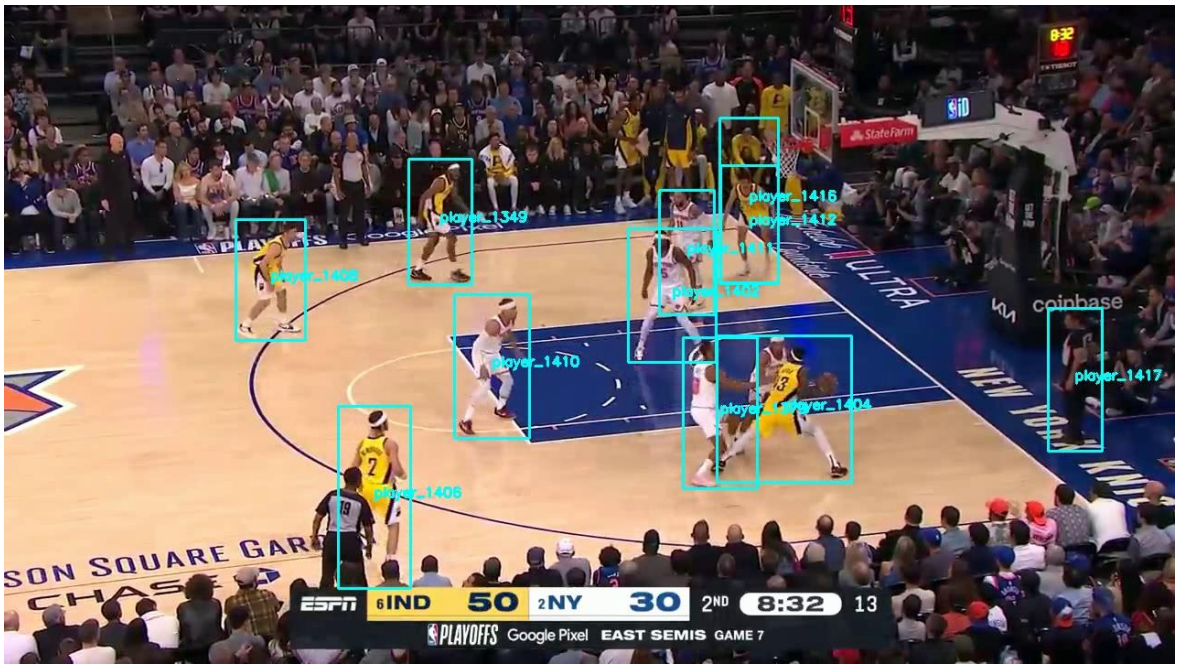
Jedna od situacija koje sustavu stvaraju problem je pogrešna detekcija publike kao igrača, u slučaju da je kamera u tom trenutku bila usmjerena isključivo prema publici(Slika 5.1.), iako se to ne događa često (Slika 5.2)



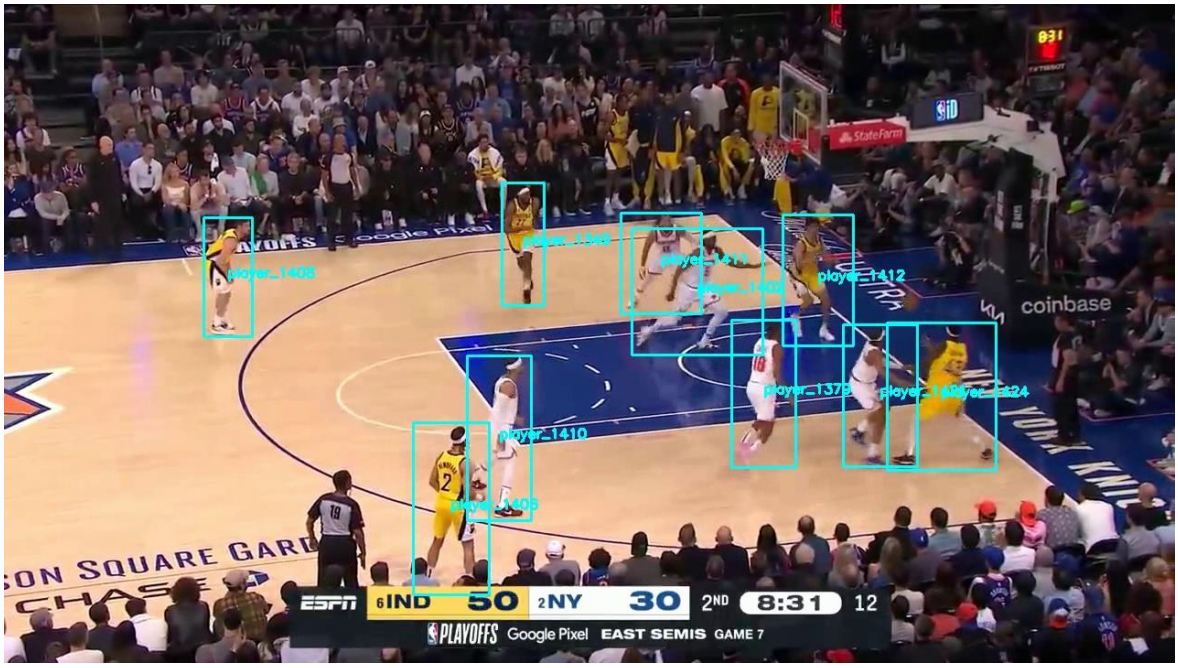
Slika 5.1 Primjer slike na kojoj je gledatelj utakmice pogrešno označen kao košarkaš sa jedinstvenom oznakom 8

Slika 5.2 Primjer slike na kojoj je taj isti gledatelj ispravno detektiran kao dio pozadine

Također, ponekad se događaju pogrešne detekcije sudaca kao košarkaša te igrača prisutnih na klupi košarkaša(Slika 5.3) Međutim, pokazuje se da sustav uspješno prati igrače te im ispravno zadržava oznake. Vidljivo je da je sustav sekundu kasnije ispravio obje greške prilikom detekcije te je svim igračima ostavio iste oznake, čak i ako se nalaze značajno promijene svoju poziciju u kratkom vremenskom intervalu, poput igrača s oznakom player_1410 i player_1412(Slika 5.4).



Slika 5.3 Primjer slike na kojoj je uspješno detektirano 9 igrača od mogućih 10, te su greškom dio pozadine i sudac detektirani kao košarkaši



Slika 5.4 Primjer slike na kojoj je svih 10 igrača uspješno detektirano te nema pogrešnih detekcija

Zaključak

Izradom ovog završnog rada dobiven je učinkovit sustav za detekciju i praćenje košarkaša na snimkama košarkaških utakmica, temeljen na modelu YOLOv8n i praćenju središnjih točaka vanjskih okvira objekata. Sustav teško detektira igrače koji su skriveni u trenucima kada se ispred njih nalazi neki drugi igrač, zbog čega bi se trebala doraditi detekcija igrača kojima je vidljiv tek manji dio tijela. Također, iako praćenje igrača funkcionira dobro, možda bi se moglo doraditi primjenom nekih drugih metoda, kako bi se poboljšale performanse praćenja prilikom kontakta među igračima. To bi se možda uspjelo poboljšati primjenom Kalmanovog filtera ili dubokog učenja za točnu identifikaciju svakog igrača. Međutim, razvijen je sustav koji je dovoljno vremenski efikasan te kvalitetno radi u stvarnom vremenu te je otvoren za daljnji razvoj i nadogradnje. Smjer u kojem bi se mogao razvijati je praćenje postignutih koševa pojedinog igrača te njemu pripadne momčadi, kao i praćenje ostalih bitnih metrika koje su korisne pri razvijanju strategije u košarci.

Literatura

[1] Marr B., 7 Amazing Examples Of Computer And Machine Vision In Practice, Forbes (2021, prosinac). Poveznica: https://www.forbes.com/sites/bernardmarr/2019/04/08/7-amazing-examples-of-computer-and-machine-vision-in-practice/#3dbb3f751018; pristupljeno 14. lipnja 2024.

[2] Nascimento, J. C., Abrantes, A. J., & Marques, J. S. (ožujak, 1999). An algorithm for centroid-based tracking of moving objects. In 1999 IEEE international conference on acoustics, speech, and signal processing. Proceedings. ICASSP99 (Cat. No. 99CH36258) (Vol. 6, pp. 3305-3308). IEEE.

[3] Čupić, M. Umjetne neuronske mreže, Umjetna inteligencija (2018, lipanj). Poveznica: http://java.zemris.fer.hr/nastava/ui/ann/ann-20180604.pdf; pristupljeno 13. lipnja 2024.

[4] Šnajder, J. Osnovni koncepti, Strojno učenje 1, (2022). Poveznica: https://www.fer.unizg.hr/_download/repository/SU1-2022-P02-OsnovniKoncepti.pdf; pristupljeno 13. lipnja 2024.

[5] Euclidean distance, Wikipedia, (2024, ožujak). Poveznica: https://en.wikipedia.org/wiki/Euclidean_distance; pristupljeno 13. lipnja 2024.

[6] Python (programming language), Wikipedia, (lipanj, 2024). Poveznica: https://en.wikipedia.org/wiki/Python_(programming_language); pristupljeno 12. lipnja 2024.

[7] OpenCV, Wikipedia, (veljača, 2024). Poveznica: https://en.wikipedia.org/wiki/OpenCV; pristupljeno 12. lipnja 2024.

[8] About Us, NumPy, (2024). Poveznica: https://numpy.org/about/; pristupljeno 12. lipnja 2024.

[9] Imambi, S., Prakash, K. B., & Kanagachidambaresan, G. R. (2021). PyTorch. Programming with TensorFlow: Solution for Edge Computing Applications, 87-104.

[10] Home – Ultralytics, Ultralytics Inc., (2024). Poveznica: https://docs.ultralytics.com/; pristupljeno 13. lipnja 2024.

[11] ultralytics, Ultralytics, (2024). Poveznica: https://github.com/ultralytics/ultralytics?tab=readme-ov-file; pristupljeno 13. lipnja 2024.

Sažetak

U završnom radu opisan je proces izrade sustava za detekciju i praćenje igrača na videozapisima košarkaških utakmica, odnosno detekciju i praćenje nekoliko objekata na videozapisu. Opisane su korištene tehnologije te način na koji one tvore jednu cjelinu za praćenje igrača u stvarnom vremenu. Prikazana je priprema podatkovnog skupa za treniranje modela YOLOv8n te dobivene metrike i rezultati modela. Detektirani igrači dalje se prate računanjem euklidskih udaljenosti na dvije uzastopne sličice. Sustav uspješno detektira i prati igrače na videozapisu, ali bi se sustav možda mogao poboljšati korištenjem drugih metoda praćenja.

Summary

This undergraduate thesis describes the process of building a system designed for detecting and tracking basketball players in video footages of basketball games, in other words, detecting and tracking multiple objects in a video file. Used technologies and way in which they form a functional system are described. Preparation of own dataset for YOLOv8n model trainging and trained model metrics are presented. Detected players are tracked on video by calculating the euclidean distance between objects on two consecutive frames. The system is successfully detecting and tracking basketball players on video files, but better results may be achieved by using the different object tracking methods.