Automatsko detektiranje rukometnog terena na slici

Mravić, Filip

Sveučilište u Rijeci – Odjel za informatiku

Studij poslovne informatike

Filip Mravić

Automatsko detektiranje rukometnog terena na slici

Diplomski rad

Mentor: doc. dr. sc. Miran Pobar

Rijeka, 14.06.2020.g.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 01.06.2020.

Zadatak za diplomski rad

Pristupnik: Filip Mravić

Naziv diplomskog rada: Automatsko detektiranje rukometnog terena na slici

Naziv diplomskog rada na eng. jeziku: Automatic handball court detection

Sadržaj zadatka:

Jedan od problema koji se javlja kod automatske analize videa iz domene sporta je detekcija sportskog terena. Automatskom detekcijom terena omogućuje se praćenje kada igrači napuštaju teren te isključivanje detektiranih osoba koje ne sudjeluju u igri iz daljnje analize. Detektirane granice terena mogu služiti i za geometrijsku kalibraciju kamere čime se omogućava izračun pozicije igrača unutar terena za vrijeme igre.

Zadatak ovog diplomskog rada je proučiti postupke obrade i segmentacije slike kojima se mogu izlučiti ključne točke slike koje su pogodne za detekciju terena korištenjem postupaka strojnog učenja ili pak izravno odrediti granice terena. Postupke treba u primijeniti za detekciju rukometnog terena u video snimkama treninga nastalim u amaterskim uvjetima i testirati njihovu uspješnost s obzirom na točnost detekcije.

Mentor: doc. dr. sc. Miran Pobar

Voditeljica za diplomske radove: izv. prof. dr. sc. Ana Meštrović

Komentor:

Zadatak preuzet: 01.06.2020.

(potpis pristupnika) F. Mravić

Sadržaj po poglavljima

Sažetak

U ovom diplomskom radu su prezentirana i opisana 3 eksperimenta (2 od njih su vezana za detekciju terena dok je jedan vezan za segmentaciju terena). Prvi eksperiment ne koristi postupke strojnog učenja nego samo funkcije za obradu slike koristeći OpenCV biblioteku dok ostala dva koriste strojno učenje; jedan od njih koristi konvolucijsku neuronsku mrežu dok drugi koristi istu takvu neuronsku mrežu ali koristeći napredniju metodu koja je nazvana Mask R-CNN. Sva 3 eksperimenta koriste različite metode za dobivanje konačnog rezultata koji će se na kraju usporediti sa ostalim metodama.

Ključne riječi

Detekcija, segmentacija, neuronska mreža, konvolucijska neuronska mreža, U-NET, Detectron, Mask-RCNN

Uvod

Jedan od problema u domeni sporta je objektivna analiza sportaša prikupljanjem različitih statistika. Zbog povećanja profitabilnosti u nekim sportovima došlo je do većeg strategiziranja u kojem se pokušavaju provesti metode analize videa kako bi se dobili podaci kao što je primjerice prevaljena dužina tijekom igre, formacija igrača u nekom određenom trenutku, broj akcija određenog igrača itd. [1].

Cilj ovog rada je pokušati detektirati i segmentirati objekt – rukometni teren na slici. Detekcija terena uključuje određivanje vanjskih granica objekta (terena) koje mogu biti prikazane oko pronađenog objekta u obliku pravokutnika koji sadrži vidljivi dio terena (eng. boundary box). Preciznije određivanje oblika traženog objekta, gdje se za svaki piksel na slici treba odrediti pripada li objektu ili ne, je problem segmentacije objekta.

Rad se u većem dijelu sastoji od praktičnog dijela u kojem se ispituju 2 različite metode: pokušaj segmentacije rukometnog terena korištenjem Houghove transformacije te detekcija i segmentacija terena pomoću neuronske mreže. Metoda u kojem se koriste neuronske mreže također ispituje dvije različite arhitekture: U-NET i Mask R-CNN. Nakon toga se sve metode analiziraju i evaluiraju.

Korišteni materijali

Materijali koji su korišteni u ovom radu su potrebni za učenje modela i testiranje/validaciju za metode koje koriste neuronsku mrežu i oni su sljedeći:

- 15 kratkih videa trajanja od 1-2 sekunde koji su pretvoreni u slike (uzet je početni frame)
- Desetak videa trajanja nekoliko minuta koje su snimljene u dvije različite dvorane u Rijeci od kojih je uglavnom izdvojeno 3 slike (početak, sredina i kraj videa) ali ponekad i više po videu. Na taj način je prikupljeno 57 slika.
- 55 slika je prikupljeno sa Youtube-a pomoću alata koji preuzme sliku u određenom frame-u koji su snimljeni na olimpijskim igrama u Londonu (2012) i Riu de Janeiru (2016) te na svjetskom prvenstvu 2019.g.

Sveukupno je korišteno 127 slika. U dijelu kada se treba podijeliti skup slika na učenje i testiranje korišten je standard 80% / 20%, gdje je 80% slika korišteno za učenje a ostalih 20% za testiranje.

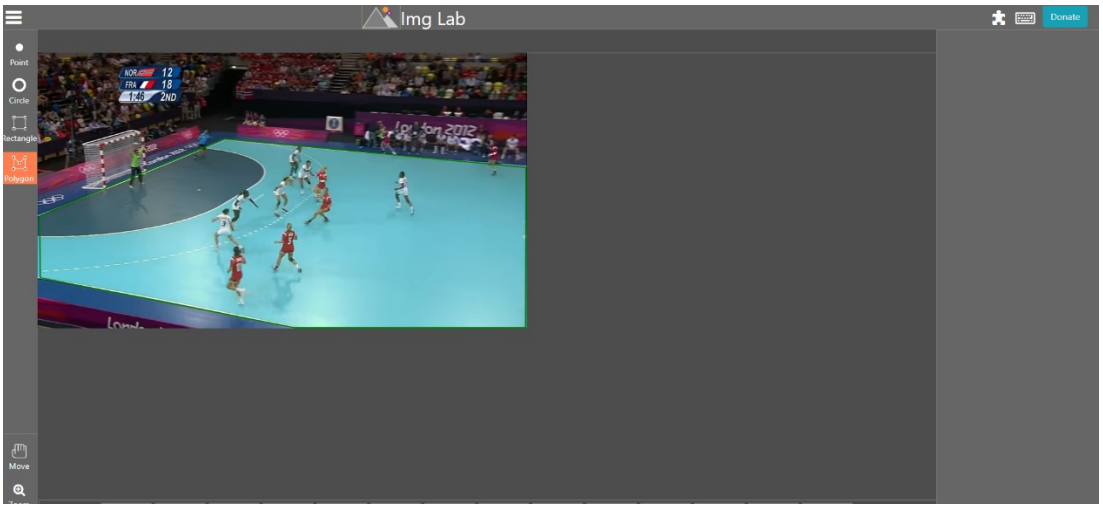
Neke slike koje se koriste kao ulaz za detekciju terena su slikane sa kamerom koja ima širokokutnu leću (eng. fisheye lens camera) pa zbog toga je bilo potrebno popraviti slike u procesiranju jer bi moglo dati bolje rezultate u detekciji terena ako se normaliziraju. Korišten je program “Korekcija radijalne distorzije” (eng. Radial correction distortion) [4]. Program se pokreće kao već kompajlirani C++ program koji pokušava napraviti korekciju slike ako je moguće. Od 15 slika program je uspješno popravio 10 slika. Slike su se popravile na način da su nakon korekcije linije terena postale ravne dok u originalnim slikama slika one ne izgledaju posve ravne već malo zaobljeno. Rezultat koji se dobije je normalizirana slika koja će ispraviti lagane zaobljenosti linija terena. Na slici 1 i 2 možemo vidjeti primjer originalne slike sa radijalnom distorzijom i ispod nje je normaliziranu sliku.

Slika 1 Originalna slika dodavanje_KS_11

Slika 2 Popravljanje radijalne distorzije slike 1

Proces anotacije slika za daljnju digitalnu obradu

Jedan važan pripremni korak u učenju modela za detekciju objekata je anotacija objekata na slici. Strojno učenje modela za detekciju objekata zahtjeva parove ulaznih slika i željenih oznaka (anotacija), na temelju kojih će naučiti određivati odgovarajuće oznake i za slike koje nisu korištene za učenje. Bez skupa anotiranih slika, računalo ne može naučiti raspoznati razlike između ostalih objekata i objekta kojeg se traži. U ovom radu klasa objekta kojeg se traži je rukometni teren. Anotacija klasa je obavljena u Open Source softverskom alatu zvanom Imglab koji je napisan u Javascript programskom jeziku tako da se program pokreće u web pregledniku. Proces anotiranja slike se u ovom radu obavio na način da se opisao poligon oko rukometnog terena. Slika 3 prikazuje sučelje Imglab aplikacije.



Slika 3 Primjer anotacije rukometnog terena u programu Imglab

Kada je završen proces anotiranja klase/a na slikama tada je potrebno prevesti to u nekakav format koji će aplikacija u kojoj se radi strojno učenje prepoznati. U ovom radu je korišten COCO-ov JSON format koja je obična JSON datoteka u kojim su sve slike opisane dodatnim parametrima od kojih je najvažniji „bounding box“ koji opisuje poligon sa svim koordinatama kao točkama tog poligona. Potrebno je proces anotiranja napraviti i na skupu za testiranje kako bi program mogao provjeriti svoje rezultate te dobiti nekakvu statistiku svojih pokušaja.

Detekcija granica terena korištenjem Houghove transformacije

Cilj prvog iskušanog postupka je detekcija granica terena pomoću detektiranih ravnih linija na slici korištenjem Houghove transformacije (detaljnije objašnjena u potpoglavlju).

Koraci obrade se mogu podijeliti na:

1. Pretvorbu iz RGB sustava u HSV sustav boja
2. Dobivanje binarne maske (ili maski) plavih i smeđih područja slike
3. Spajanje maske i originalne slike
4. Dobivanje rubova na slici sa Canny Edge algoritmom
5. Detekcija linija Houghovom transformacijom
6. Detekcija granica terena

Rezultat koji se dobije pokretanjem programa je slika koja bi zelenom bojom trebala ocrtati granice terena. Slike međukoraka u tom procesu se spremaju u izlaznoj mapi koja je nazvana “output”.

Pretvorba iz RGB sustava u HSV sustav boja

Modul cv2 se pretežito koristi u ovom programu tako se već i u početku koristi za učitavanje slika za daljnje procesiranje. Nakon učitavanja slike, potrebno je pretvoriti sliku iz BGR (blue-green-red slikovnog modela) u HSV (hue, saturation, value slikovni model). HSV model je bitan zbog lakše izrade binarne maske. HSV slikovni model se sastoji od 3 različite komponente: H (hue, hrv. Nijansa), S (saturation, hrv. Zasićenje) i V (value, hrv. Vrijednost). Raspon vrijednost komponenti se razlikuje u drugim aplikacijama. Glavna razlika između ovog modela i primjerice RGB modela i zašto se često koristi za manipulaciju slika je što je dovoljno navesti samo jedan kanal i već se zna koja boja se koristi dok recimo u RGB modelu se mora navesti 3 kanala [5]. U ovom eksperimentu je korištena nijansa plave i smeđe boje. Ako se koristi RGB kanal nijanse tih boja se drastično mijenjaju ako se promijeni bilo koja

vrijednost od tih 3 kanala dok to nije slučaj u HSV modelu. Za recimo plavu boju, vrijednost H (hue) komponente koristeći cv2 modul je izabran 96. Mijenjanjem komponenti S (saturation) ili V (value) još uvijek dobivamo plavu samo drukčiju svjetlinu.

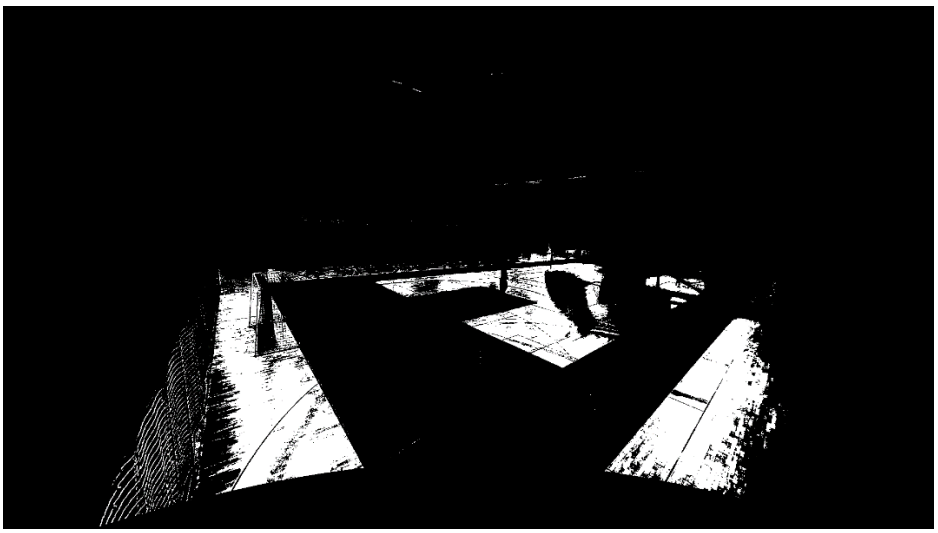
Dobivanje binarne maske

Binarna maska će koristiti dvije primarne boje koje će se razlikovati od ostalih, to je tamno plava koja bi trebala pokriti dio terena u radijusu od 7 metara (tzv. sedmerac) kod vratara te sve stranice cijelog terena koje su pokrivene svjetlo plavom bojom. Druga boja je smeđa boja koja većinom pokriva cijeli teren. Te dvije boje su spremljene u BGR modelu u obliku polja sa trima vrijednostima koristeći modul numpy.

Prije nego se dobije binarna maska potrebno je dobiti vrijednost plave i smeđe boje terena koju pokušavamo izolirati. To možemo dobiti koristeći jednostavan alat kao što je npr. Paint 3D, gdje se može koristiti funkcionalnost “color picker” (hrv. Izabirač boja) da bi dobili specifične vrijednosti boja na toj slici. Kada smo dobili zadane vrijednosti one se tada moraju pretvoriti u HSV model kako bi se dalje koristile za izradu maske. Još jedan korak prije izrade maske je uzimanje donjih i gornjih granica kod segmentacije/izolacije boja. Konvencija koja se inače uzima je da donja granica ima vrijednosti (H-10, S-40, V-40) dok gornja granica ima iste vrijednosti samo koristeći dodavanje vrijednosti umjesto oduzimanja. Nakon toga se izrađuju dvije maske, jedna koja će izolirati samo plavu boju na slici, a druga koja izolira samo smeđu boju. Kada se dobiju te dvije maske, može se dobiti konačna maska spajanjem dvije maske pomoću aritmetičkog operatora ‘+’. Adicija u cv2-u funkcionira na način da kao prvo, obje slike moraju koristiti isti slikovni model (npr. ako prva slika koristi HSV model onda mora i druga). Vrijednosti svakog kanala se samo pridodaju drugoj vrijednosti gdje, ako je vrijednost veća od najveće moguće vrijednosti kanala, tj. vrijednosti 255 tada se koristi operator “%” odnosno modulo. Recimo ako se vrijednosti 250 dodaje vrijednost 6 tada će konačna vrijednost na tom kanalu biti 1 [6].

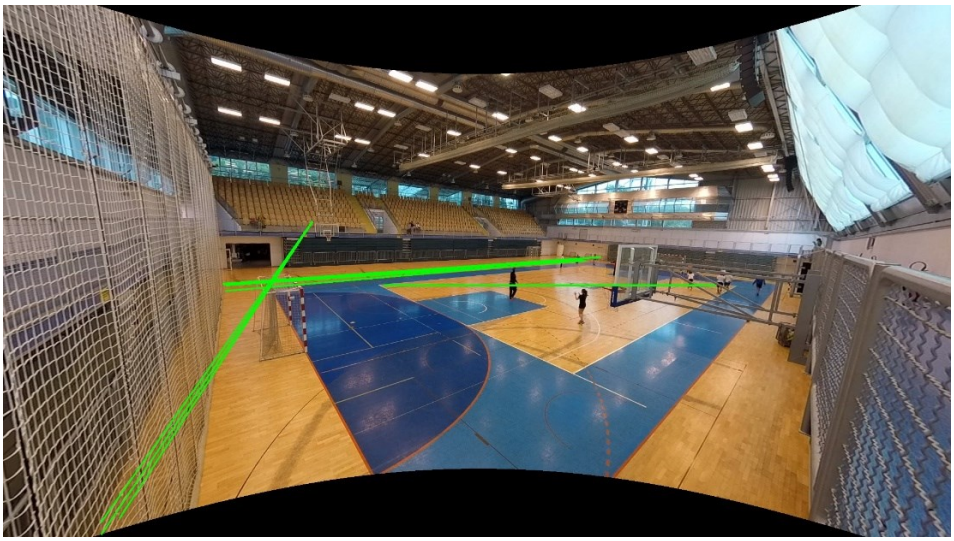
Na slikama 4 i 5 možemo vidjeti masku koju dobijemo. Bijela boja je izolirana boja dok je crnom obojano sve ostalo. Možemo vidjeti na slikama da su također većim dijelom komplementarne, što znači nakon adicije dviju slika dobit će se cijela slika terena.

Slika 4 Binarna maska dobivena segmentacijom plave boje



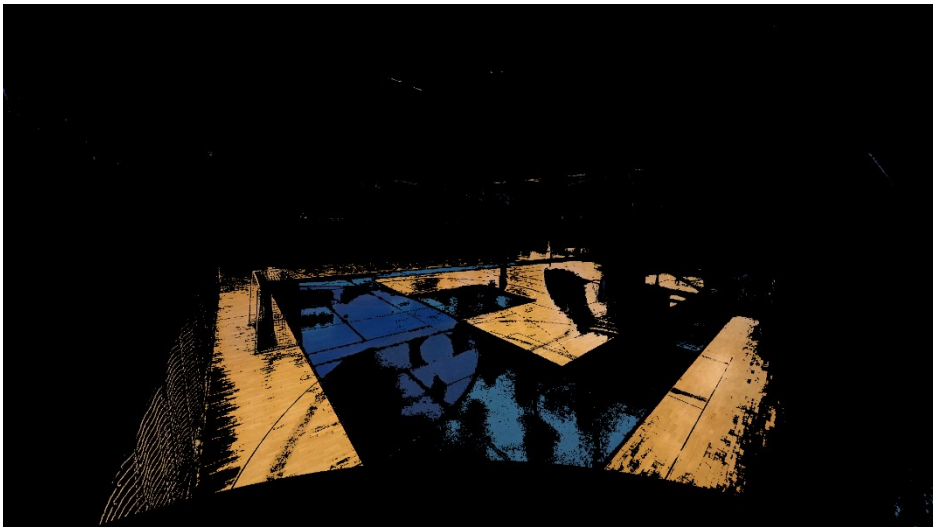
Slika 5 Binarna maska dobivena segmentacijom smeđe boje

Konačno moramo postaviti finalnu masku na originalnu sliku koristeći bitwise and(&) operator. Taj operator nam omogućuje da postavimo masku preko originalne slike tako da možemo odstraniti dijelove slike, artefakte koji nisu potrebni kao što je publika. Slika 6 prikazuje slučaj ako pokušamo dobiti granice terena bez spajanja maski. Možemo na kraju usporediti sliku 6 sa slikom 9 i zaključiti da je bilo potrebno napraviti adiciju maski kako bi dobili bolje rezultate.



Slika 6 Granice terena korištenjem samo maske koja filtrira smeđu boju

Slika 7 prikazuje rezultat dobivanja konačne maske.



Slika 7 Rezultantna slika dobivena adicijom/dodavanjem dviju maski sa ulaznom slikom

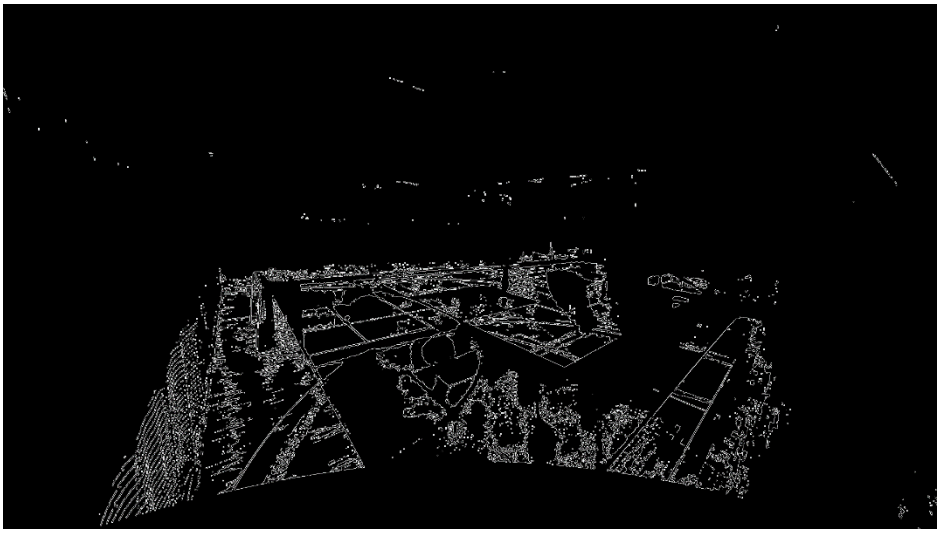
Naivni pokušaj detekcije terena se zove upravo zato jer se koriste točno izabrane maske za specifičnim BGR/HSV vrijednostima terena kako bi se dobila točna kombinacija boja terena. Za svaku drugu sliku će se javiti problem različitog osvjetljenja i bilo kakva promjena može izazvati promjenu boja terena gdje binarna maska na slici neće ispasti dobro.

Dobivanje rubova na slici sa Canny Edge algoritmom

Sljedeći korak je dobivanje rubova na slici, ali za to je potrebno pretvoriti sliku u crno-bijeli model. Funkcija Canny u cv2-u je jedan od algoritama koji se koristi u izdvajanju rubova na slici koji je razvio John F. Canny 1986.g. Algoritam se sastoji od nekoliko koraka [7]:

- Uklanjanje šumova/smetnji na slici (eng. noise reduction) koristeći Gaussian filtar.
- Traženje gradijenta intenziteta slike koji se dobije koristeći Sobel kernel u horizontalnom i vertikalnom smjeru (možda dodati funkcije)
- Supresija lokalnih ne-maksimuma, za svaki piksel se provjerava ako se u njegovoj okolini nalazi lokalni maksimum
- Uzimanje "sigurnih" rubova koji imaju vrijednosti u određenoj donjoj i gornjoj granici

Slika 8 prikazuje rezultat koji se dobije nakon operacije detektiranja rubova.



Slika 8 Rubovi dobiveni Canny Edge algoritmom

Detekcija linija terena Houghovom transformacijom

Rezultantna slika će biti ulazni parametar u pozivanje zadnje funkcije cv2.HoughLinesP(). Hough transformacija je popularna tehnika za detektiranje bilo kakvog oblika koji se može matematički prikazati. Ako treba prepoznati linije na slici tada će koristiti matematičku formulu za pravac $y = mx + c$ odnosno u parametarskoj formi $\rho = x\cos\theta + y\ \sin\theta$ [7]. Funkcija koja se koristi je optimizirana verzija, tzv. probabilistička Hough transformacija. Razlika ove funkcije je što već za manje slike treba duže vremena za procesiranje. Ova funkcija ne uzima sve točke na slici u obzir nego samo podskup nekih točaka. Za ovaj

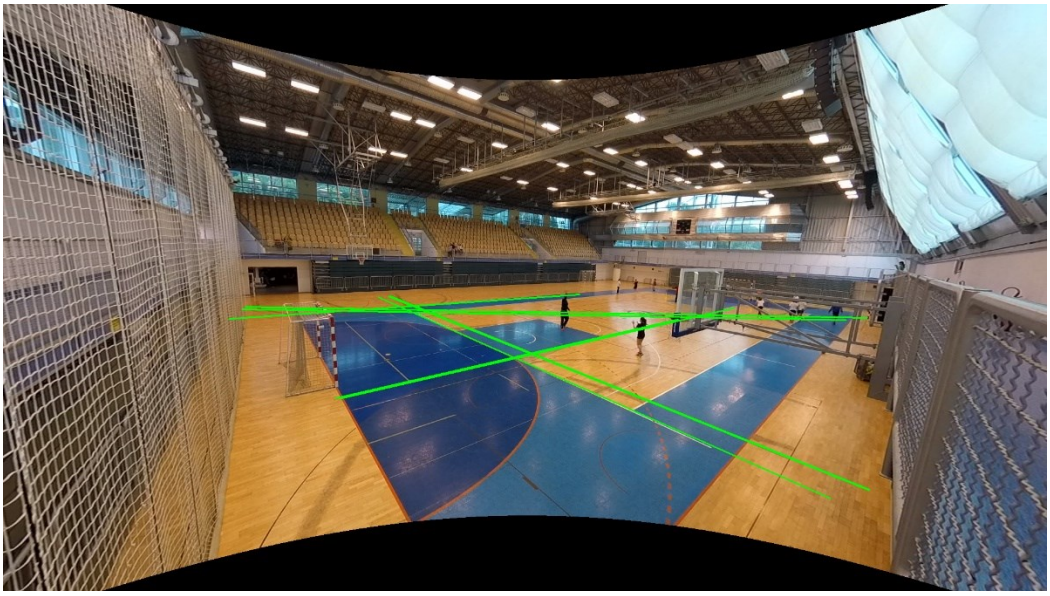
postupak je onda potrebno smanjiti granice za pronalaženje linija jer inače ne bi pronašlo niti jednu liniju.

Svaku točku slike možemo podijeliti u ćeliju gdje se prati vrijednosti: (ρ, θ). Linija će se pronaći akumuliranjem vrijednosti (tzv. broj glasova, eng. number of votes) gdje pronađe niz točaka pod nekim pravim kutem.

Preciznost pronalaženja linija u ovoj funkciji se vidi u sljedećim parametrima:

- Rho (ρ) – u pronalaženju linija se uvijek postavlja na vrijednost 1
- Theta (θ) – u pronalaženju linija se uvijek postavlja na vrijednost $\frac{\pi}{180}\ rad$
- Threshold (hrv. prag, granica) – najmanji broj glasova za signaliziranje da je pronašao liniju
- Minimum line length – najmanji broj u pikselima koji će činiti liniju
- Max line gap – najveći broj u pikselima između linija koje se prema tome mogu tretirati kao zasebne linije ili kao jedna cjelina

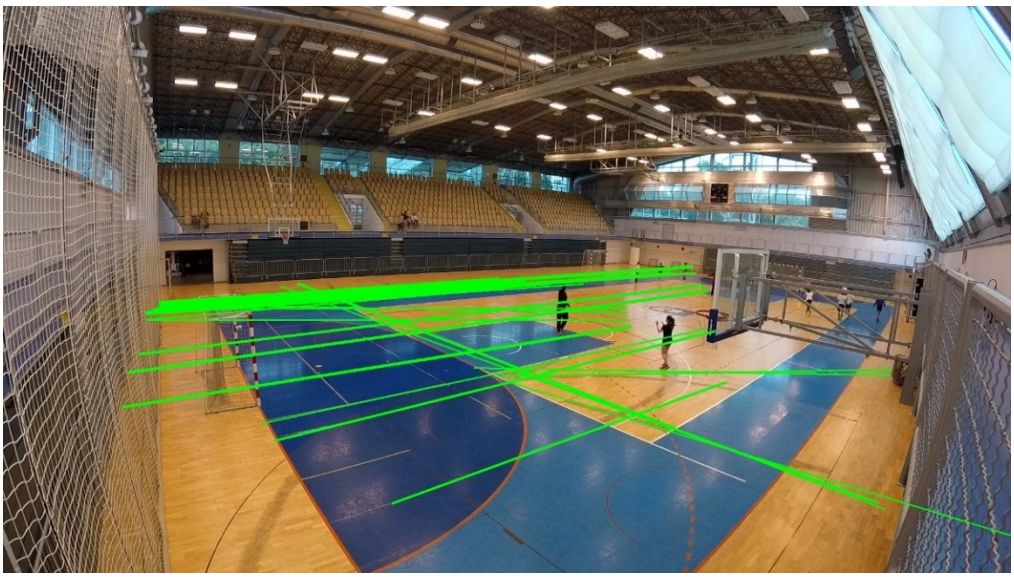
Slika 9 prikazuje rezultat koji se ostvario funkcijom HoughLinesP



Slika 9 Granice terena sa Houghovom transformacijom

Parametri bi se trebali još ugladiti kako bi se dobilo malo bolje rezultate nego što je sad. Problem je što i segmentirana slika ne izgleda baš najbolje zbog osvjetljena na teren pa zbog toga je teško dobiti cijelokupni teren izoliran i na koncu nije moguće dobiti linije koje bi detektirale teren kako treba.

Na kraju možemo također usporediti rezultate i na originalnoj slici kojoj nije ispravljena radijalna distorcija. Može se na slici 10 vidjeti da je iscrtano puno više linija bez previše značenja što znači da je korekcija slike bila ispravan korak u cijeloj metodi.



Slika 10 Granice terena sa Houghovom transformacijom bez ispravljanja slike

Implementacija

Program je napisan u Pythonu (koji se dosta često koristi za procesiranje kompjuterske grafike) sa sljedećim korištenim modulima:

- Modul **cv2 –** primarno korišteni modul u ovom projektu koji omogućuje procesiranje i manipulaciju kompjuterske grafike
- Modul **numpy** – jedan od popularnih modula zbog velikog broja funkcija sa poljima
- Modul **matplotlib** – modul koji je u ovom projektu korišten za prikaz rezultantnih slika u nekim određenim koracima programa
- Modul **argparse** – modul koji se koristi samo za jednostavno pokretanje programa sa više parametara

Za korištenje ovog programa je potrebno imati instalirano: Python3 i sve ostale navedene module iznad. Glavni program koji se zove **main.py** se pokreće naredbom **python main.py --input putanja_do_slike.png**.

U programu se najprije učitava slika koristeći funkciju cv2.*imread().* Sliku koja je učitana je u RGB modelu te ju je potrebno pretvoriti u HSV model koje se može dobiti funkcijom *cv2.cvtColor(img, COLOR_BGR2HSV)*. Nakon toga je potrebno dobiti HSV vrijednosti smeđe i plave boje. To se može dobiti alatom kao što je color picker ali se treba pripaziti jer vrijednosti svih komponenti HSV modela u cv2 modulu se kreću u rasponu od [0-180, 0-255, 0-255] dok se negdje drugdje koristi raspon od [0-360, 0-100, 0-100] koje se obilježavaju sa

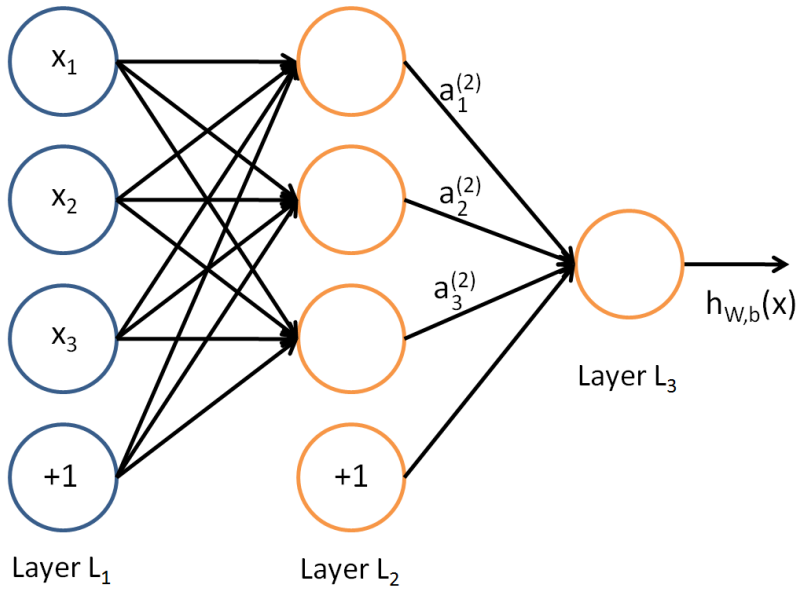
stupnjevima. Sljedeći korak je dobivanje dviju maski kako bi filtrirali smeđu i plavu boju, a to se može dobiti funkcijom *cv2.inRange().* Na kraju se konačnu masku može dobiti adicijom dobivenih dviju maski koristeći funkciju *cv2.bitwise_and()*. Sljedeći korak je dobivanje rubova na slici pozivanjem funkcije *cv2.Canny()* i postavljanjem donjeg i gornjeg ruba na određene vrijednosti, ali prije toga je potrebno pretvoriti BGR sliku u crno-bijelu sliku. To se dobije pozivanjem funkcije *cv2.cvtColor(img, COLOR_BGR2GRAY).* Posljednji korak je dobivanje granica terena koristeći funkciju *cv2.HoughLinesP()* koju možemo zapisati na računalo funkcijom *cv2.imwrite().*

Detekcija i segmentacija terena pomoću neuronske mreže

Povijest i općenito o neuronskim mrežama

Neuronska mreža je pojam koji se koristi već 70 godina sa počecima sredinom 40-ih godina prošlog stoljeća koju su predložili kao ideju dva američka znastvenika sa Sveučilišta u Chicagu: Warren McCullough i Walter Pitts koji su kasnije se preselili na MIT [9]. Neuronska mreža je jedan od najpoznatijih arhitektura u području strojnog učenja. Neke od vrsta neuronskih mreža su: konvolucijska neuronska mreža (CNN), povratna neuronska mreža (RNN), Autoenkoderi, itd…

Arhitektura neuronske mreže je inspirirana građom neurona i njihovim vezama u mozgu. Glavni element neuronske mreže je neuron kojih može biti na tisuće ili i više. Funkcionira na isti način kao i u ljudi, uzima nekakav ulazni podatak te daje izlaz sljedećem neuronu. U području strojnog učenja neuron je samo matematička funkcija koja uzima vrijednost te daje rezultat sljedećem neuronu. Kolekcija neurona se zove sloj (eng. layer) i oni su podjeljeni u 3 skupine: ulazni sloj (eng. input layer), skriveni slojevi (eng. hidden layers) kojih može biti 1 ili više i izlazni sloj (eng. output layer). Na slici 11 ispod možemo vidjeti najjednostavniju neuronsku mrežu koja će onda imati 3 sloja.



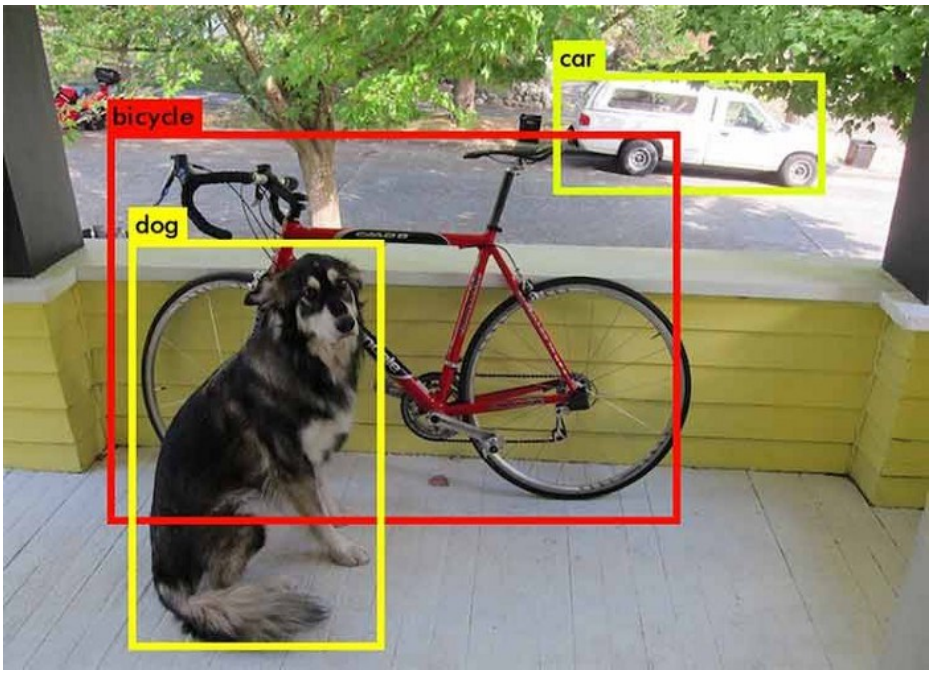
Slika 11 Primjer jednostavne neuronske mreže [16]

Zadnji važan element u modelu neuronske mreže je težina (eng. weight) koja je dodjeljena svakoj vezi između dva sloja i oni su uvijek nasumično dodjeljeni prije početka treniranja modela. Oni se u trenutku treniranja konstantno mijenjaju kako bi transformacija podataka bila najpreciznija [16].

Neuronska mreža „uči“ na način da unaprijed dobije skup podataka gdje su točni odgovori na problem već dani (recimo u obliku anotacija svakog objekta bitnog za klasifikaciju). Neuronska mreža, kada dobije izlaznu vrijednost, to usporedi sa već dobivenim rješenjem pomoću funkcije troška (eng. cost function). Rezultat te funkcije se prosljeđuje natrag po svim neuronima mijenjajući vrijednosti težine na svakoj konekciji neurona. Taj proces se naziva propagacija unatrag (eng. back propagation) i to je ključan proces kako neuronska mreža uči [17].

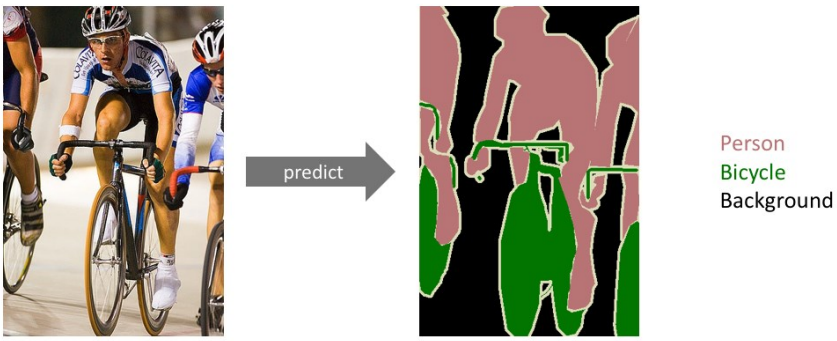
Semantička segmentacija i detekcija objekata

Jedni od glavnih problema u području računalnog vida su je detekcija objekata na slici. Problem detekcije objekata uključuje problem klasifikacije (određivanje vrste objekta) i lokalizacije objekta (određivanje lokacije gdje se objekt nalazi na slici), za sve objekte na slici. Rezultat detekcije je tipično u obliku ocrtanih pravokutnika oko detektiranih objekata i pripadajuće oznake imena klase tog objekta. Na slici 12 ispod možemo vidjeti primjer detekcije 3 objekata na slici.



Slika 12 Primjer detekcije objekata na slici [12]

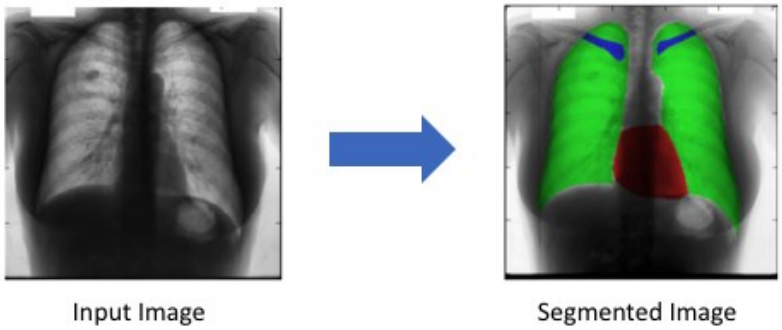
Semantička segmentacija se razlikuje od problema detekcije na način što sada svaki piksel pripada nekoj klasi na slici. Taj zadatak se još naziva i "gusta predikcija" (eng. Dense prediction). Na slici 13 možemo vidjeti segmentaciju osobe i bicikla od pozadine.



Slika 13 Primjer semantičke segmentacije objekata na slici [12]

Primjene semantičke segmentacije/detekcije objekata se može vidjeti recimo u području medicine kod dijagnoze pacijenata, npr. računala mogu poboljšati analizu rendgena tako da smanjuju potrebno vrijeme koje doktori potroše radeći ispitivajući dijagnozu pacijenta. U području autonomne vožnje se izrazito koristi segmentacija i detekcija kako bi lakše analizirali kako se kretati na cesti itd. [12]

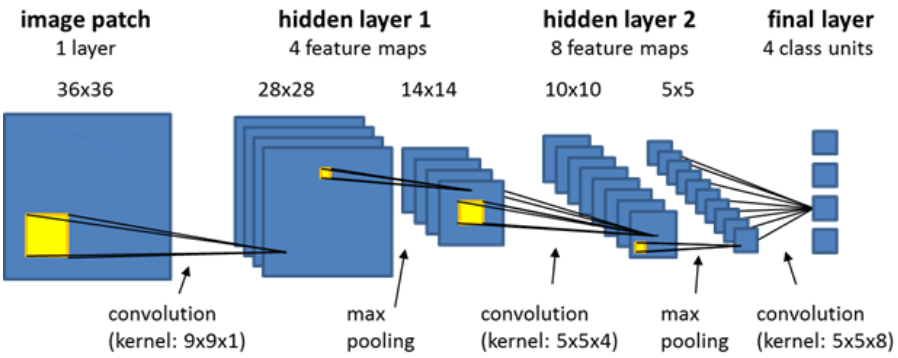
Na slic 14 ispod možemo vidjeti kako je uspješno segmentirano područje pluća, srca i ključne kosti:



Slika 14 Segmentacija pluća, srca i ključne kosti pacijenta [18]

O konvolucijskoj neuronskoj mreži

CNN ili ConvNet (convolutional neural network, hrv. Konvolucijska neuronska mreža) je jedna od vrsta neuronske mreže koja je najpoznatija u području digitalne obrade slika. Prednost nad jednostavnom neuronskom mrežom, tzv. “proslijedi-dalje” neuronskom mrežom (eng. feed-forward neural network), kao što je npr. jednoslojni i višeslojni (eng. Single/multi layer) perceptron, je što je izlučivanje značajki integrirano u samu mrežu. "Klasične" neuronske mreže su kao ulaz morale koristiti ručno dorađene filtere/karakteristike dok CNN ima sposobnost naučiti to “sam” [10]. Na slici 15 se vidi arhitektura CNN-a.



Slika 15 Arhitektura konvolucijske neuronske mreže [8]

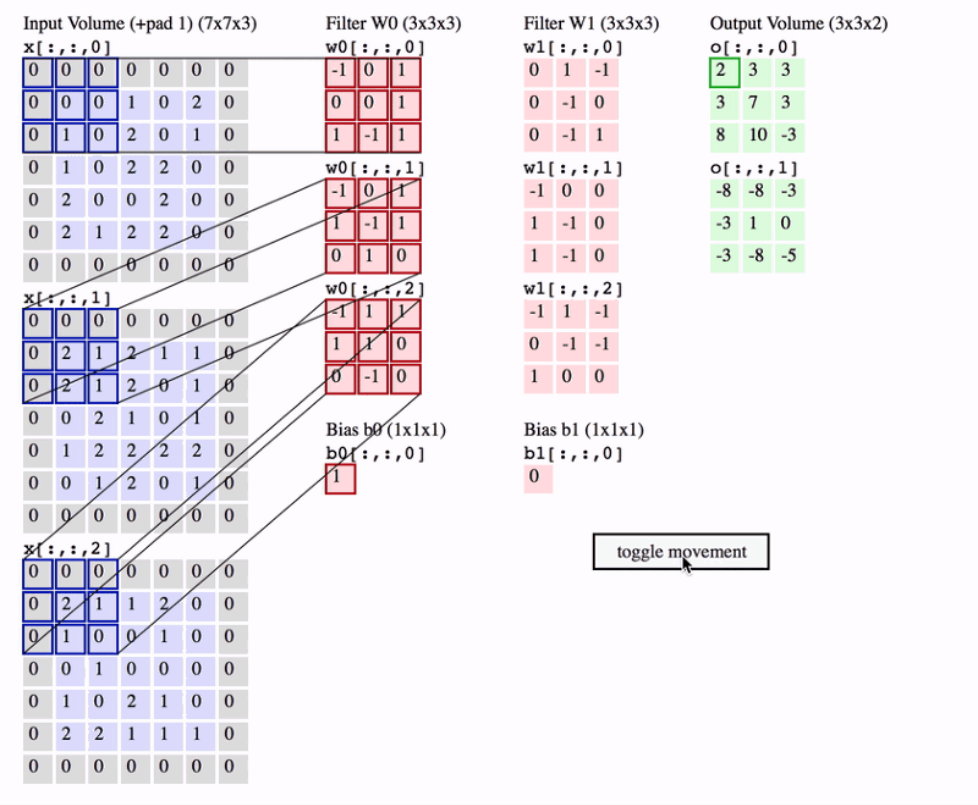
Arhitektura konvolucijske neuronske mreže je slična "običnoj" neuronskoj mreži (eng. Multi layer perceptron, MLP) gdje na ulazu (početni, prvi sloj) imamo onoliko neurona koliko cijela slika ima piksela puta broj kanala na slici. Svaki neuron će imati vrijednost između 0 i 1. Zadnji sloj ima neurona koliko klasa ima u skupu mogućih klasa. Između početnog i zadnjeg sloja se nalazi nekoliko slojeva koji se inače zovu "skriveni" slojevi u MLP-u ili konvolucijski slojevi u CNN-u. Ti se sastoje od nekoliko filtera: od najosnovnijih geometrijskih filtera koji detektiraju linije na slici (eng. Edge detection) do zadnjih slojeva koji mogu na kraju detektirati lice osobe/životinje i ostale značajke preko kojih se na kraju može klasificirati objekt na slici.

Svaki od tih slojeva na sebi ima nekoliko bitnih operacija gdje rezultat šalju kompleksnijem sloju. Konvolucijski slojevi su dimenzija *n x n x broj kanala* gdje je n najčešće uzet kao 3 ili 5. Prva operacija koja se događa na tom sloju se naziva konvolucijska operacija koja je vrlo slična produktu matrica i definirana je na sljedeći način [2]:

$$C(j,k) = \sum_{p}\sum_{q} A(p,q)B(j-p+1, k-q+1)$$

C označava rezultantnu matricu sa retkom j i stupcem k. Generalno, veličina matrice C će biti suma veličine matrice dijela slike (matrica A) i kernela (matrica B) te umanjena za 1 ali se rezultantna matrica uvijek postavlja na istu veličinu kao i ulazna matrica (ulazna slika). Indeks sume p se kreće od 0 do size(A) – 1, a indeks sume q se analogno tomu kreće od 0 do size(B) – 1.

Recimo da je konvolucijski sloj veličine *3 x 3 x 3,* operacija se može vizualizirati na slici 16. Prvi korak je vertikalno i horizontalno okretanje matrice kernela, odnosno okretanje matrice po retcima i onda okretanje matrice, koju smo dobili okretanjem po retcima, po stupcima. Nakon toga uzmemo veličinu matrice kernela na ulaznoj matrici te krećući se od prvog elementa silazno po stupcima i retcima do zadnjeg dijela matrice množimo sve elemente i zbrojimo. Konačni element zapisujemo kao prvi element rezultantne matrice. Taj postupak se nastavlja dok se ne napravi operacija na svim dijelovima ulazne matrice.



Slika 16 Vizualizacija konvolucijske operacije, input je prvi sloj a filter W0 je konvolucijski sloj [3]

Ostale operacije koje slijede nakon konvolucijske operacije su:

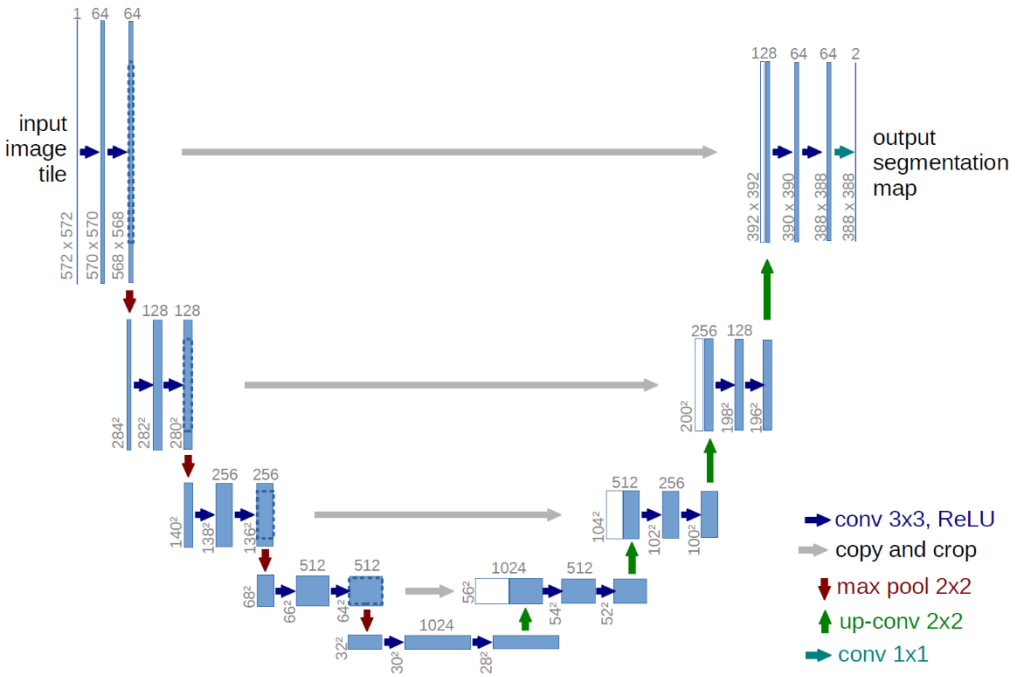
- Max pooling operacija – reducira veličina matrice značajke tako da imamo manje parametara, drugim riječima veličina slike se smanjuje (eng. down-sampling)
- Transponirana konvolucija ili dekonvolucija – metoda za pretvaranje u veću rezoluciju slike koja je bitna za dobivanje segmentacije slike (eng. Up-sampling)

Segmentacija terena pomoću neuronske mreže U-Net

U prvom eksperimentu, metoda koja je korištena za segmentaciju objekta na slici koristi U-NET arhitekturu konvolucijske neuronske mreže. U-NET je razvio Olaf Ronneberger sa timom u svrhe segmentacije slike u području biomedicine 2015.g [19].

Ulazni podatak je RGB slika i nakon toga slijedi proces. Proces se sastoji od 2 faze: kontrakcija (enkoder) koji traži kontekst odnosno što je to na slici i ekspanzija (dekoder) koji traži gdje se točno nalazi na slici. Prva faza smanjuje slike na jako malu veličinu koristeći normalne konvolucijske operacije i max-pooling povećavajući dubinu (kanal) slike, recimo RGB sliku veličine 128 x 128 px (128x128x3) pretvara u sliku veličine 8 x 8px (8x8x256). U drugoj fazi se događa obrnuti proces, koristeći konvolucijske operacije zajedno sa transponiranom konvolucijom dobiva se početna slika sa dobivenom informacijom gdje se

nalazi taj objekt [19]. Na kraju kao izlaz se dobije binarna maska. Na slici 17 je prikazana cijela arhitektura U-Neta.



Slika 17 Arhitektura U-NET-a [22]

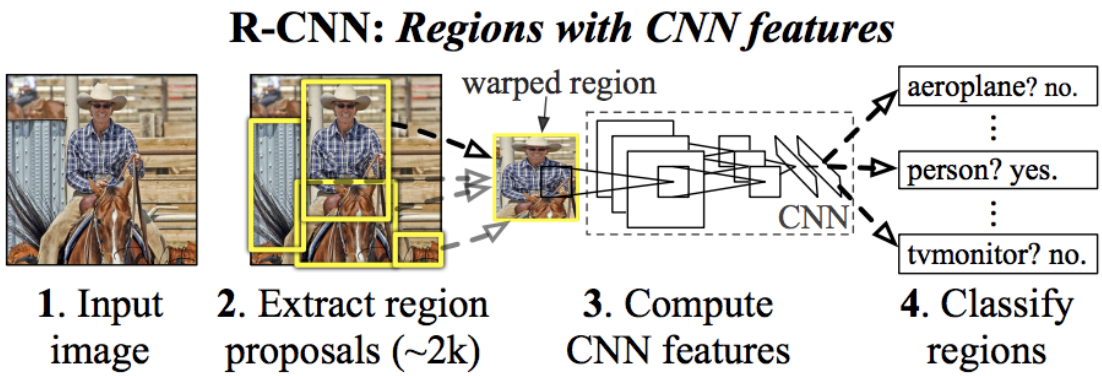
Detekcija i segmentacija terena pomoću mreže Mask-RCNN

U drugom eksperimentu, za detekciju i segmentaciju terena korištena je mreža Mask-RCNN implementirana u alatu Detectron. Detectron je open source projekt kojeg je razvio tim istraživača u Facebook-ovoj AI Research grupi 2019.g. U ovom radu je korištena poboljšana – druga verzija softvera nazvana Detectron v2 koja je implementirana pomoću PyTorch biblioteke za strojno učenje. Prednosti u odnosu na prvu verziju je što je više fleksibilan i pruža brže strojno učenje na jednoj ili više grafičkih kartica [11].

Programski kod za aplikaciju je napisan na Google Colab platformi zato jer pružaju opciju za pokretanje koda na njihovim serverima koji imaju opciju pokretanja na CUDA grafičkim karticama zato jer je PyTorch biblioteka koristi CUDA Toolkit.

Mask R-CNN je nadopuna na prethodne mreže od kojih je posljednja u nizu bila Faster R-CNN. Mask R-CNN nudi istovremeno rješavanje problema detekcije i segmentacije objekata. Mask R-CNN kreće od R-CNN-a (eng. Region-based CNN), nadopunu na konvolucijsku neuronsku mrežu koju su razvili R. Girshick et al., gdje se selektira točno 2000 regija na slici i onda se koristeći „pohlepne“ (eng. greedy) algoritme rekurzivno pozivaju slične regije na slici i spajaju se u veće regije koje se na kraju klasificiraju. Veliki problem za ovaj tip mreže je što je potrebno puno vremena za učenje modela i nije moguće implementirati ovu mrežu u

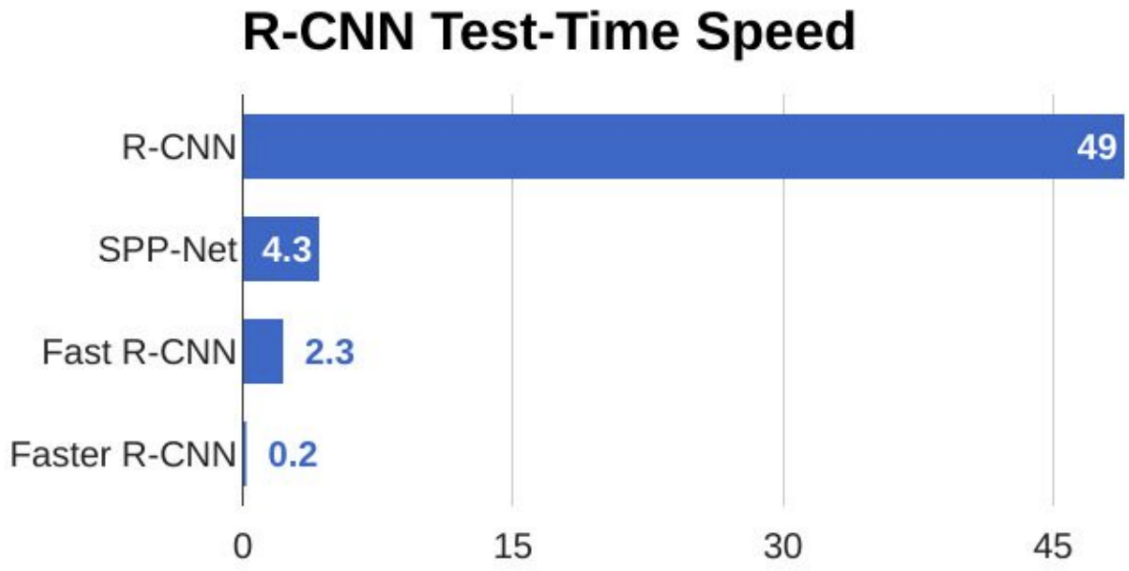
realnom vremenu jer u prosjeku je potrebno oko 47 sekundi za obradu jedne slike [21]. Na slici 18 je prikazana arhitektura R-CNN mreže.



Slika 18 Arhitektura Region-based CNN-a [21]

Autor R. Girshick je rješio neke probleme u R-CNN-u i novu mrežu nazvao Fast R-CNN. Napredak je što više nije potrebno svaki puta izdvojiti 2000 regija u svakom procesu izrade mape značajki (eng. feature map) nego je prvo iz ulazne slike stvorena konvolucijska mapa značajki pa je na toj mapi izdvojeno 2000 regija što je minimiziralo vrijeme učenja 10 puta i vrijeme testiranja 20 puta [22].

Selekcija regija je bio dugotrajan proces pa su zbog toga autori S. Ren et al. izradili novi tip CNN-a koji su nazvali Faster R-CNN koji je koristio posebnu neuronsku mrežu samo za učenje selektiranja regija na slici. Vrijeme testiranja je bilo 10 puta kraće od Fast R-CNN-a. Na slici 19 možemo vidjeti vrijeme testiranja nekoliko vrsta CNN-a.

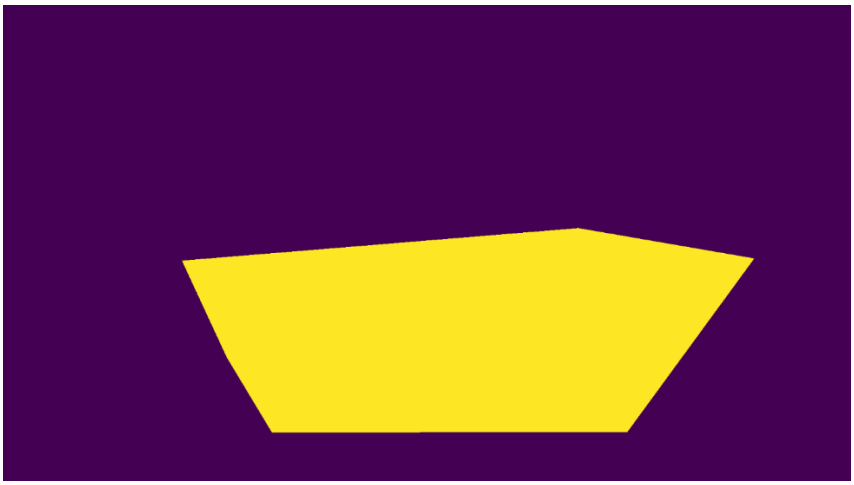


Slika 19 Performanse testiranja slike koristeći modifikacije CNN-a[24]

Faster R-CNN kao izlaz neuronske mreže daje ime klase objekta i pripadajući bounding box. Mask R-CNN izlaz nadopunjuje samo dodavajući segmentaciju objekta. Autori R. Girshick et al. navode kako bi ovaj pristup mogao omogućiti i predikciju poze osoba na slici [20].

Opis eksperimenta – segmentacija terena pomoću U-Net arhitekture

Kod problema segmentacije objekata na slici u procesu analize predikcije se uspoređuje maska koja je dobivena strojnim učenjem i generirana maska originalne slike. Prvi korak je zbog toga nekako generirati sve maske za pripadajuću sliku. Generirane maske će biti binarne slike (u ovom slučaju će pozadina biti ljubičaste boje a prvi plan će biti žute boje). Slika 20 prikazuje jednu od generiranih binarnih maski od rukometnog terena.



Slika 20 Primjer jedne od generiranih binarnih maska slike terena

Maska se generira pomoću COCO-ove *annToMask* funkcije gdje je samo potrebno učitati anotaciju za tu sliku. Koristeći petlju na taj način je u ovom radu u desetak sekundi izgenerirana maska za svaku pripadajuću sliku.

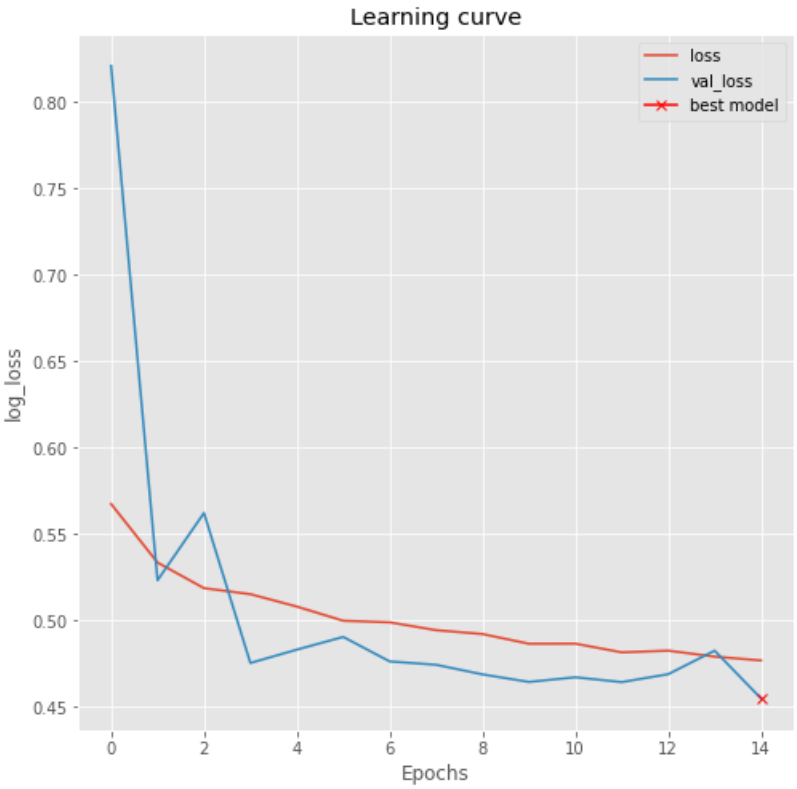
Kada su sve maske izgenerirane može se krenuti sa glavnim programom koji koristi U-NET arhitekturu kao model za konvolocijsku neuronsku mrežu. Program pretežito koristi Tensorflow sa Keras-om za dio strojnog učenja. Pomoćne biblioteke koje omogućuju osnovno manipuliranje slikama i ostale sitne svari uključuje: numpy, scipy, skimage, sklearn, … Rezolucija slika koja će se koristiti za strojno učenje je u ovom eksperimentu postavljena na 128x128 px kako učenje ne bi trajalo predugo. Još jedan parametar koji se koristi je *seed* koji je postavljen na neki broj tako da kada se bilo kada koristi funkcija koja daje nasumični rezultat, taj rezultat uvijek bude isti kod sljedećeg pokretanja aplikacije.

Nakon učitavanja slika i maska u program bitno je napraviti augmentaciju podataka kako bi dobili bolje rezultate u učenju. Augmentaciju slika nudi Keras u paketu *image preprocessing* koristeći ImageDataGenerator koji nudi mnogo opcija za transformaciju slike. Nažalost u ovom eksperimentu se koriste i maske pa u biti slike dolaze u binarnom paru (original, maska) a biblioteka koristi nasumični generator kao rezultat svih transformacija tako da su transformacije originalne slike i maske različite a to ne smije biti, one moraju biti iste. Zbog tog razloga je korišten „primitivniji" način transformacija slika. Svakoj originalnoj slici i maski su dodane 4 kopije, svaka sa svojom transformacijom. Transformacije čine: nasumična rotacija slike, zrcaljenje slike u horizontalnom smjeru, nasumično osvjetljenje i nasumični kontrast slike. Nakon toga u biti od 1 slike imamo 5 slika što na kraju znači da je od 127 slika umjetno dobiveno 508 slika i sada sveukupno je 635 slika.

U procesu strojnog učenja korištena je U-NET arhitektura sa 4 kontrakcijska sloja i 4 ekspanzijska sloja te jednim dodatnim slojem između što daje 9 slojeva. Još jednom za napomenuti, u kontrakcijskom sloju se radi konvolucija (smanjenje, eng. down-sampling) slike kako bi se saznalo o kojem objektu se radi . Nakon toga dolazi se u ekspanzijski sloj gdje se radi povećanje slike kako bi se saznalo gdje se točno taj objekt nalazi na slici (eng. up-sampling). Svaki konvolucijski sloj koristi kernel (filter) odnosno matricu veličine 3x3 gdje obavlja konvolucijske i max pooling operacije.

Hiperparametri (parametri koji su u modelu dani proizvoljno) koji su bitni za ovaj model su: *batch_size* i *epochs*. Prvi hiperparametar daje oznaku modelu koliko iteracija treba proći dok ne pokuša napraviti nekakvu predikciju dok drugi, broj epoha, označava broj koliko puta treba ponoviti prolazak kroz sve ulazne podatke. Svaki put kada započne sljedeću epohu program pokušava popraviti svoje interne parametre kako bi se povećala preciznost predikcija. Za ovaj model je korišten *batch_size=2* i *epochs=15*. Zbog malo manjeg broja slika moguće je postaviti jako mali *batch_size* (najmanje moguće je 1, što znači nakon 1 slike će pokušati napraviti neku predikciju), što je veći *batch_size* to će učenje biti brže ali preciznost lošija. Ako se nakon brojnih pokretanja vidi da se preciznost previše ne mijenja u pozitivnom smjeru nakon završetka jedne epohe moguće je smanjiti taj broj.

Nakon završetka učenja u ovom eksperimentu je prikazana krivulja učenja radi lakše vizualizacije toka uspješnosti predikcija. Parametar koji se proučava je *loss* i *val_loss*. *Val_loss* je uvijek prioritet jer on proučava gubitak na skupu za validaciju. U početku će parametri imati vrlo visoke vrijednosti. Cilj je minimizirati te brojke da one budu što bliže 0. U ovom eksperimentu najbolji rezultat je dobiven u 14. epohi kao što se na vidi na slici ispod. Trajanje svake epohe je u prosjeku trajalo 68.8 sekundi što na kraju daje ukupno vrijeme učenja od 17 minuta. Slika 21 prikazuje krivulju učenja.



Slika 21 Krivulja učenja u ovom eksperimentu

Opis eksperimenta – segmentacija terena pomoću Mask-RCNN arhitekture

Zbog manjeg broja ulaznih podataka za postupak strojnog učenja prvi korak u programu je bio napraviti funkciju koja će izvoditi nekoliko nasumičnih transformacijskih operacija na slikama, primjerice mijenjanje jačine svjetlosti (eng. Brightness), zrcaljenje slike (eng. Flip) i rezanje nekih dijelova slike (eng. Crop). Taj postupak se naziva augmentacija podataka ili je još poznato pod nazivom sintetiziranje podataka. Nažalost, dokumentacija alata Detectron za korištenje augmentacije nad podacima je dosta gruba i nema puno informacija i također pitanja na webstranicama StackOverflow i Github nisu davala potpuna rješenja. Zbog toga je odlučeno da se ne koristi augmentacija podataka (augmentacija je bila implementirana ali loger nije pokazivao točne transformacije koje su bile korištene pa je zbog toga odbačeno).

Nakon toga je bitno registrirati COCO instancu i kao ulazni parametar je bitno proslijediti JSON datoteka, odnosno anotacije slika na skupu za učenje i treniranje. Nakon toga funkcija stvara rječnik kojemu se može pristupati pozivanjem klase *DatasetCatalog* i *MetadataCatalog*.

Za prikaz segmentiranog objekta na slici se koristi klasa *Visualizer* te se može odmah provjeriti ako su anotacije dobro učitane u memoriju računala na način da se prvo inicijalizira instanca klase te nakon toga pozove metoda *draw_dataset_dict.* Na slici 22 je prikaz segmentiranih slika prije početka učenja.

Slika 22 Prikaz anotiranih slika prije strojnog učenja

Kada smo uvjereni da su anotacije korektno učitane na ulaznim podacima slijedi konfiguracija treninga za strojno učenje gdje je jako bitno da je korektno podešen inače predikcija nakon strojnog učenja će biti lošija. U konfiguraciji se mogu podesiti varijable kao što je: korak učenja (eng. Learning rate), broj iteracija u procesu učenja te promijeniti model neuralnih mreža koji je već istreniran na dosta objekata. U ovom radu je korišten ResNet50 koja je konvolucijska neuronska mreža sa 50 slojeva a nalazi se u Detectron-ovoj konfiguraciji napisan u YAML programskom jeziku koji se koristi za serijalizaciju podataka odnosno za konfiguraciju podataka i aplikacija. Na slici 23 se može vidjeti ispis cijelog procesa strojnog učenja:

```
[10/03 13:22:56 d2.engine.train_loop]: Starting training from iteration 0
[10/03 13:23:01 d2.utils.events]:  eta: 0:01:05  iter: 19  total_loss: 1.311  loss_cls: 0.379  loss_box_reg: 0.072  loss_mask: 0.619
[10/03 13:23:06 d2.utils.events]:  eta: 0:01:02  iter: 39  total_loss: 0.932  loss_cls: 0.147  loss_box_reg: 0.361  loss_mask: 0.422
[10/03 13:23:12 d2.utils.events]:  eta: 0:00:58  iter: 59  total_loss: 0.697  loss_cls: 0.094  loss_box_reg: 0.274  loss_mask: 0.265
[10/03 13:23:17 d2.utils.events]:  eta: 0:00:54  iter: 79  total_loss: 0.416  loss_cls: 0.058  loss_box_reg: 0.173  loss_mask: 0.193
[10/03 13:23:22 d2.utils.events]:  eta: 0:00:50  iter: 99  total_loss: 0.349  loss_cls: 0.037  loss_box_reg: 0.136  loss_mask: 0.147
[10/03 13:23:27 d2.utils.events]:  eta: 0:00:45  iter: 119  total_loss: 0.227  loss_cls: 0.030  loss_box_reg: 0.116  loss_mask: 0.082
[10/03 13:23:32 d2.utils.events]:  eta: 0:00:40  iter: 139  total_loss: 0.226  loss_cls: 0.028  loss_box_reg: 0.124  loss_mask: 0.062
[10/03 13:23:37 d2.utils.events]:  eta: 0:00:35  iter: 159  total_loss: 0.200  loss_cls: 0.031  loss_box_reg: 0.097  loss_mask: 0.067
[10/03 13:23:42 d2.utils.events]:  eta: 0:00:30  iter: 179  total_loss: 0.208  loss_cls: 0.025  loss_box_reg: 0.111  loss_mask: 0.058
[10/03 13:23:47 d2.utils.events]:  eta: 0:00:25  iter: 199  total_loss: 0.181  loss_cls: 0.029  loss_box_reg: 0.094  loss_mask: 0.050
[10/03 13:23:52 d2.utils.events]:  eta: 0:00:20  iter: 219  total_loss: 0.139  loss_cls: 0.019  loss_box_reg: 0.081  loss_mask: 0.031
[10/03 13:23:57 d2.utils.events]:  eta: 0:00:15  iter: 239  total_loss: 0.135  loss_cls: 0.020  loss_box_reg: 0.075  loss_mask: 0.037
[10/03 13:24:02 d2.utils.events]:  eta: 0:00:10  iter: 259  total_loss: 0.132  loss_cls: 0.017  loss_box_reg: 0.078  loss_mask: 0.029
[10/03 13:24:07 d2.utils.events]:  eta: 0:00:05  iter: 279  total_loss: 0.120  loss_cls: 0.014  loss_box_reg: 0.074  loss_mask: 0.028
[10/03 13:24:14 d2.utils.events]:  eta: 0:00:00  iter: 299  total_loss: 0.116  loss_cls: 0.016  loss_box_reg: 0.070  loss_mask: 0.024
[10/03 13:24:15 d2.engine.hooks]: Overall training speed: 297 iterations in 0:01:14 (0.2517 s / it)
[10/03 13:24:15 d2.engine.hooks]: Total training time: 0:01:18 (0:00:03 on hooks)
```

Slika 23 Tijek strojnog učenja nad 45 augmentiranih slika

Nakon učenja je potrebno napraviti predikciju na slikama koje se nalaze u skupu za testiranje. Kao i kod provjeranja korektnosti anotacija u procesu prije učenja, na isti način se može

vizualno provjeriti predikcija koju je računalo napravilo. Na sljedećoj slici se može vidjeti predikcija rukometnog terena. Može se vidjeti da je računalo pokušalo predikciju na 2 instance klase rukometnog terena zbog nekakvog razloga. Na slici 24 se može vidjeti vizualizacija predikcije jedne nasumične slike iz skupa za testiranje.

Slika 24 Vizualizacija predikcije nad jednom slikom u skupu za testiranje

Rezultati

Posljednji korak je evaluirati predikcije koje je računalo napravilo nad validacijskom skupu podataka. Dvije metrike koje tome služe su su **IoU** (eng. Intersection over union) poznata kao i Jaccardov indeks i **AP** (eng. average precision, prosječna preciznost). Definiraju se na sljedeći način:

$$IoU = \frac{original \cup predikcija}{original \cap predikcija},$$

gdje je *original* maska koja odgovara stvarnom stanju (ground truth), a *predikcija* maska koju je generirala mreža. Drugim riječima ova metrika je omjer broja piksela koji su na bilo kojoj od dvije maske i broja piksela koji se nalaze na obje maske. Drugi parametar, prosječna preciznost, se definira na sljedeći način:

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

TP (eng. true positive) je broj uspješnih predikcija (kada je IoU >= prag) dok **FP** (eng. false positive) je broj neuspješnih predikcija (IoU < prag) [14].

U procesu predikcije bitno je odabrati prag prolaznosti predikcije (eng. threshold) nad slikom. Konvencija je koristiti 3 različite mjere u statističkoj obradi slika u strojnom učenju: **AP**, **AP50**, **AP75**. AP50 i AP75 označavaju da se koristila granica od IoU = 0.5 ili 0.75. Mjera AP nam govori više globalno o IoU vrijednostima. Računa se na način da se uzme prosjek svih AP vrijednosti počevši od IoU = 0.5 do IoU = 0.95 sa pomakom od 0.05 što znači da se računa prosjek 10 različitih IoU vrijednosti nad cijelom validacijskom skupu što nam daje malo bolju sliku o konačnom rezultatu.

Mask R-CNN rezultati

Nakon učenja modela koristeći neuronsku mrežu potrebno je testirati model na slikama u skupu za testiranje (validaciju). COCO u sebi već posjeduje funkciju COCOEvaluator te zajedno sa funkcijom *inference_on_dataset* (eng. Inference, hrv. Zaključak) može se dobiti prosječna preciznost predikcija. Evaluator na kraju daje 2 skupa vrijednosti, jedan se odnosi na preciznost bounding box-a dok se drugi odnosi na segmentaciju objekta. Na slici 25 možemo vidjeti statističke podatke o evaluaciji predikcije.

```
[11/07 20:56:08 d2.evaluation.coco_evaluation]: Evaluation results for segm:
|   AP   |  AP50   |  AP75   |  APs  |  APm  |  APl   |
|:------:|:-------:|:-------:|:-----:|:-----:|:------:|
| 82.931 | 100.000 | 100.000 |  nan  |  nan  | 82.931 |
[11/07 20:56:08 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
OrderedDict([('bbox',
              {'AP': 86.89615014132993,
               'AP50': 100.0,
               'AP75': 96.03960396039604,
               'APl': 86.89615014132993,
               'APm': nan,
               'APs': nan}),
             ('segm',
              {'AP': 82.93078405584919,
               'AP50': 100.0,
               'AP75': 100.0,
               'APl': 82.93078405584919,
               'APm': nan,
               'APs': nan})])
```

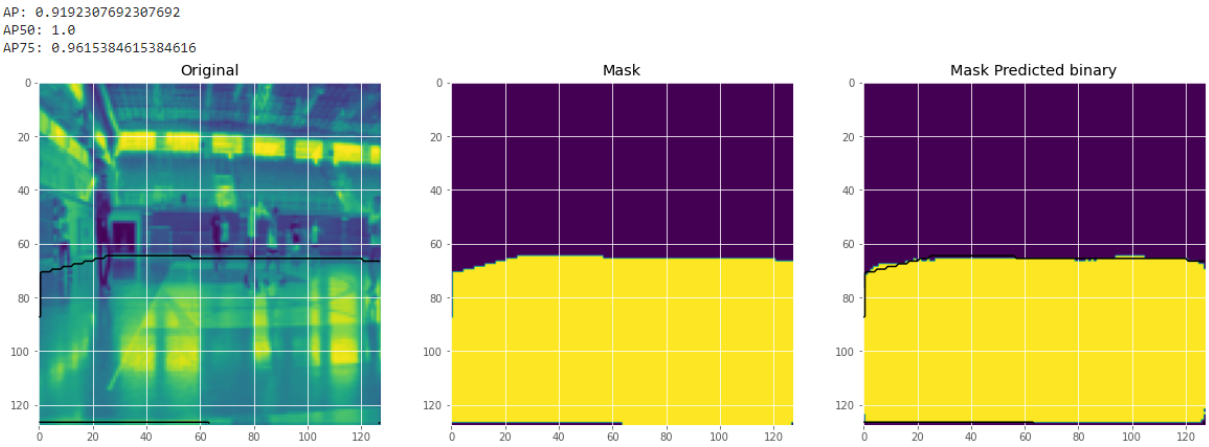
Slika 25 Prikaz prosječne preciznosti za detekciju i segmentaciju nad testnim podacima

Rezultat koji je bitan za procjenu konačnih rezultata je skup vrijednosti koji se dobije na kraju cijele evaluacije koji je tipa *OrderedDict* u Pythonu. Parametar koji se promatra je **AP** (eng. average precision, prosječna preciznost). Možemo vidjeti na validacijskom skupu od 26 slika, prosječna globalna preciznost za bounding box je 86.9% dok za problem segmentacije objekta je postignut rezultat od 83% što je vrlo zadovoljavajuće.

U-NET rezultati

Za razliku od Mask R-CNN arhitekture gdje je korišten Detectron kao framework, eksperiment u kojem se koristi U-NET arhitektura nema u sebi funkciju COCOEvaluator. Zbog toga je isprogramirana jednostavna funkcija *evaluate* koja samo vraća vrijednost IoU-a po formuli na početku ovog poglavlja.

Prosječna globalna preciznost predikcija na cijelom validacijskom skupu (26 slika) u ovom eksperimentu koristeći postupke augmentacije je 0.9192 što znači da je 91.92% slučajeva uspješno predviđeno. Na sljedećoj slici možemo vidjeti statističke podatke o uspješnosti predikcije koristeći augmentaciju.



Slika 26 Vizualizacija predikcije koristeći U-NET arhitekturu sa augmentiranim podacima i prikaz prosječne predikcije

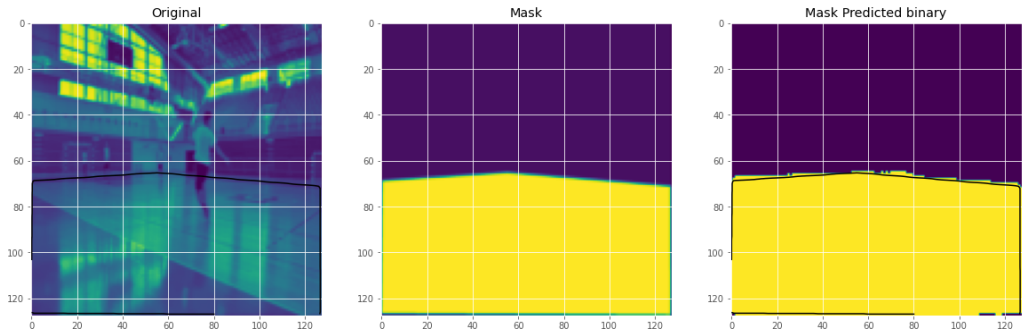
Usporedba rezultata

Na sljedećoj tablici 1 se uspoređuju rezultati vrijednosti parametra AP odnosno prosječne predikcije nad arhitekturama Mask R-CNN i U-NET koristeći 3 mjere: AP, AP50, AP75 koje su objašnjene na početku ovog poglavlja.

Tablica 1 Usporedba statističkih rezultata (prosječna predikcija) nad validacijskom skupu između U-NET-a i Mask R-CNN-a

	U-NET (sa augmentacijom)	U-NET (bez augmentacije)	Mask R-CNN
AP	91.92%	87.31%	84.5%
AP50	100%	100%	100%
AP75	96.15%	88.46%	91.9%

Na slikama 28 i 29 možemo vidjeti usporedbu rezultata segmentacije rukometnog terena na nasumičnoj slici u validacijskom skupu. U oba slučaja model je napravio dobru predikciju segmentacije.



Slika 27 Rezultat segmentacije koristeći U-NET

Slika 28 Rezultat segmentacije koristeći Mask R-CNN

Zaključak

U prvom eksperimentu je proveden naivni pokušaj detekcije terena, ne koristeći neuronsku mrežu za segmentaciju i detekciju terena. Zbog artefakata kao što je osvjetljenje i problema širina leće rezultat je bio jako loš. Parametri su se stalno morali ručno prilagoditi kako bi se dobili nekakvi rezultati (nekoliko ispravnih linija). Program koristi pretežito OpenCV biblioteku koja se koristila samo u svrhe obrade slike (i ne za strojno učenje).

Sljedeća dva eksperimenta koriste dosta slične metode za segmentaciju terena. U-NET je više orijentiran na problem segmentacije objekta dok alat Detectron rješava problem segmentacije i lokalizacije objekta odnosno pronalaženje bounding box-a. Prosječna globalna preciznost (AP) U-NET-a nad validacijskim skupom koristeći augmentaciju podataka u procesu treniranja podataka (4 transformacije po slici, odnosno 5 puta više podataka) je bila 91.92%. Ne koristeći augmentaciju točnost predikcija je bila za oko 5% manja što znači da je augmentacija podataka bila uspješna. Za razliku od U-NET-a gdje se koristi samo konvolucijska neuronska mreža, framework Detectron koristi nadopunjeni CNN sa bržim algoritmima/arhitekturom – Mask RCNN. Rezultat koji se može usporediti između U-NET-a i Mask R-CNN-a je segmentacija terena; Mask R-CNN je imao prosječnu globalnu preciznost od 84.5%, 3% manje od U-NET-a (uspoređuje se koristeći podatke koji nisu augmentirani) no ipak ima prednost što može i detektirati objekt što je bio primarni cilj R-CNN-a i drugih nadogradnji na tu arhitekturu (Fast R-CNN, Faster R-CNN).

Možemo usporediti vrijeme treniranja neuronskih mreža U-Net (sa i bez augmentacije) i Mask R-CNN (bez augmentacije):

- Vrijeme učenja koristeći Mask R-CNN: 7 minuta i 27 sekundi
- Vrijeme učenja koristeći U-Net (bez augmentacije): 4 minute
- Vrijeme učenja koristeći U-Net (sa augmentacijom): 17 minuta

Može se zaključiti da na istom skupu za treniranje (101 slika), U-Net daje bolje rezultate u brzini učenja nego Mask R-CNN kada se usporedi vrijeme učenja bez augmentiranih podataka. Vrijeme učenja sa augmentiranim podacima koristeći U-Net je dobar jer ako se uzme prosjek od 4 minute za svaki skup slika za treniranje to bi dalo prosječan rezultat od 20 minuta ako imamo 5 puta više podataka a rezultat daje čak 3 minute manje od prosječnog rezultata.

Literatura

1. E. Cheshire, C. Halasz, J. K. Perin, "Player tracking and analysis of Basketball Plays", 2015.
2. Matlab, "2-D Convolution", dostupno na: https://www.mathworks.com/help/matlab/ref/conv2.html, pristupljeno 16.11.2020.
3. Stanford University CS231, "Convolutional Neural Networks for Visual Recognition", dostupno na: https://cs231n.github.io/convolutional-networks/, pristupljeno 2.11.2020.
4. D. Santana-Cedres, L. Alvarez, L. Gomez, L. H. Palucci Vieira, P. R. Pereira Santiago, "Radial Distortion Correction of Sports Video Sequences"
5. S. Mallick, "Why does OpenCV use BGR color format?", 2015., dostupno na: https://www.learnopencv.com/why-does-opencv-use-bgr-color-format/, pristupljeno 20.08.2020.
6. OpenCV, "Changing colorspaces", dostupno na: https://docs.opencv.org/master/df/d9d/tutorial_py_colorspaces.html, pristupljeno 20.08.2020.
7. OpenCV, "Canny Edge Detection", dostupno na: https://docs.opencv.org/master/da/d22/tutorial_py_canny.html, pristupljeno 20.08.2020.
8. eCognition, "Convolutional Neural Network Algorithms", dostupno na: https://docs.ecognition.com/v9.5.0/eCognition_documentation/Reference%20Book/23%20Convolutional%20Neural%20Network%20Algorithms/Convolutional%20Neural%20Network%20Algorithms.htm, pristupljeno 2.11.2020.
9. L. Hardesty, MIT News, "Explained: Neural networks", dostupno na: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414, pristupljeno 15.10.2020.
10. S. Saha, Towards data science, "A Comprehensive Guide to Convolutional Neural Networks – the ELI5 way", dostupno na: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, pristupljeno 3.11.2020.
11. Y. Wu, A. Kirilov, F. Massa, W. Yen Lo, R. Girshick, AI Facebook Blog, "Detectron 2: A PyTorch-based modular object detection library", dostupno na: https://ai.facebook.com/blog/-detectron2-a-pytorch-based-modular-object-detection-library-/, pristupljeno 10.09.2020.
12. H. Lamba, Towards data science, "Understanding Semantic Segmentation with UNET", dostupno na: https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47, pristupljeno, 25.10.2020.
13. V. Patrawala, Towards data science, "Master the COCO Dataset for Semantic Image Segmentation", dostupno na: https://towardsdatascience.com/master-the-coco-dataset-for-semantic-image-segmentation-part-1-of-2-732712631047, pristupljeno 25.10.2020.

14. J. Jordan, “Evaluating image segmentation models”, dostupno na: https://www.jeremyjordan.me/evaluating-image-segmentation-models/, pristupljeno 29.10.2020.
15. P. Sharma, Analytics Vidhya, “Image classification vs. object detection vs. image segmentation”, dostupno na: https://medium.com/analytics-vidhya/image-classification-vs-object-detection-vs-image-segmentation-f36db85fe81, pristupljeno 18.10.2020.
16. V. Nigam, Towards data science, “Understanding neural networks. From neuron to RNN, CNN, and Deep Learning”, dostupno na: https://towardsdatascience.com/understanding-neural-networks-from-neuron-to-rnn-cnn-and-deep-learning-cd88e90e0a90, pristupljeno 16.11.2020.
17. M. Skirpan, Fast Forward Labs, “How do Neural Networks Learn?”, dostupno na: https://www.kdnuggets.com/2015/12/how-do-neural-networks-learn.html, pristupljeno 16.11.2020.
18. A. A. Novikov, D. Lenis, D. Major, J. Hladuvka, M. Wimmer, K. Buhler, “Fully Convolutional Architectures for Multi-Class Segmentation in Chest Radiographs”, 2017.
19. O. Ronneberger, P. Fischer, T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation”, 2015.
20. K. He, G. Gkioxari, P. Dollar, R. Girshick, Facebook AI Research, “Mask R-CNN”, 2018.
21. R. Girshick, J. Donahue, T. Darell, J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation”, 2014.
22. R. Girshick, “Fast R-CNN”, 2015.
23. S. Ren, K. He, R. Girshick, J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, 2016.
24. F. F. Li, J. Johnson, S. Yeung, Stanford University, “Lecture 11: Detection and Segmentation”, 2017.

Prilog - slike

Prilog – tablice

1. Usporedba statističkih rezultata (prosječna predikcija) nad validacijskom skupu između U-NET-a I Mask R-CNN-a

Prilog – alati korišteni u eksperimentima

1. COCO Split – programski razdvaja COCO JSON anotaciju na poseban JSON za učenje i poseban za testiranje, https://github.com/akarazniewicz/cocosplit
2. Imglab – za anotaciju slika, https://github.com/NaturalIntelligence/imglab
3. Detectron – framework za detekciju i segmentaciju objekata koristeći Mask R-CNN, https://github.com/facebookresearch/Detectron
4. Google Colab – softver koji omogućuje programsko kolaboriranje i pokretanje Pytorch-a koristeći virtualni CUDA GPU, https://colab.research.google.com/
5. Radial Distortion Correction – program za korekciju radijalne distorcije, https://ctim.ulpgc.es/demo112/
6. U-NET implementacija u Pythonu: https://github.com/hlamba28/UNET-TGS