SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1542

Duboki konvolucijski modeli za praćenje objekata

Ivan Fabijanić

Zagreb, lipanj 2017.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 15. ožujka 2017.

DIPLOMSKI ZADATAK br. 1542

Pristupnik: **Ivan Fabijanić (0119007943)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Duboki konvolucijski modeli za praćenje objekata**

Opis zadatka:

Praćenje objekata u slijedu slika važan je problem računalnog vida s mnogim zanimljivim primjenama. Do sada su u tu svrhu uglavnom korišteni kriteriji temeljeni na izravnoj usporedbi piksela u susjednim slikama. Međutim, recentan uspjeh klasifikacijskih dubokih modela navodi na ideju da bi se obrasci kretanja objekata mogli predvidjeti nadziranim učenjem. Tema ovog rada je istražiti prednosti i nedostatke takvog pristupa za praćenje osoba.

U okviru rada, potrebno je proučiti i ukratko opisati postojeće pristupe za praćenje objekata u slijedu slika utemeljene na dubokom učenju. Predložiti arhitekturu dubokog modela za predviđanje budućeg položaja objekata. Uhodati postupke učenja i validiranja hiperparametara. Primijeniti naučene modele na snimkama nogometnih susreta. Prikazati i ocijeniti ostvarene rezultate. Predložiti pravce budućeg razvoja.

Radu priložiti izvorni i izvršni kod razvijenih postupaka, ispitne slijedove i rezultate, uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 10. ožujka 2017.
Rok za predaju rada: 29. lipnja 2017.

Mentor:

Izv. prof. dr. sc. Siniša Šegvić

Predsjednik odbora za diplomski rad profila:

Prof. dr. sc. Siniša Srbljić

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

Zahvaljujem mentoru izv. prof. dr. sc. Siniši Šegviću koji je svojim bogatim znanjem, korisnim savjetima i utrošenim vremenom puno pripomogao prilikom izrade ovog rada.

Contents

1. Uvod

Računalni vid je područje umjetne inteligencije koje se bavi prikupljanjem, obradom, analiziranjem te razumijevanjem slika i općenito višedimenzionalnih podataka iz stvarnog svijeta sa svrhom izvlačenja korisnih numeričkih ili simboličkih podataka. Računalni vid pokriva mnogo područja koja uključuju detekciju objekata na slikama ili videu, semantičku segmentaciju (dodavanje razreda svakom pikselu slike), praćenje objekata, kalibraciju kamera, popravak slike, itd. Ovaj se bavi problemima praćenja objekata na videu, detekcijom i semantičkom segmentacijom sa posebnim naglaskom na problem praćenja objekata. Praćenje objekata na videu ili između slika koje su slikane u slijedu je jedan od najstarijih i najkompliciranijih problema u području računalnog vida još začetka tog područja računarske znanosti u 1960.-im godinama prošlog stoljeća. Od ranih početaka pa do sada tom problemu se pristupalo na razne načine koji su često ovisili o procesorskoj snazi tadašnjih računala, ali i o samom načinu pristupa problemu te samom njegovom razumijevanju. Od početka pa sve do prije par godina ljudima najlogičniji i najbolji pristup je bio praćenje ručno razvijenim metodama koje su iterativno poboljšavane kroz godine. Te metode su sve do prije par godina redovito davale, a i danas su često u vrhu po performansama za dobar dio situacija, najbolje i najtočnije rezultate praćenja. Ručno razvijenim metodama praćenja spadaju praćenje gustim optičkim tokom, rijetkim optičkim tokom (npr. KLT algoritam), praćenje Kalmanovim filterom, metoda prosječnog pomaka (*eng. meanshift*), itd. Prije par godina sve se to promijenilo kada su duboke konvolucijske neuronske mreže daleko nadmašile uobičajene metode u detekciji objekata na slikama te semantičkoj segmentaciji. Razlozi za taj nagli rast u performansama neuronskih mreža su razvitak novih algoritama učenja dubokih neuronskih mreža (npr. Adam, Adagrad i LMSProp), velik porast u performansama procesora, pogotovo grafičkih kartica, te ogroman broj kvalitetno označenih skupova za učenje. Tada je odjednom bilo moguće naučiti mreže velikog kapaciteta u nekom razumnom vremenu da rješavaju taj problem. Na krilima tog uspjeha danas se sve veći broj istraživača odlučuje uključiti konvolucijske mreže i u problem praćenja objekata na videu. Na velikom dijelu skupa videa za testiranje kvalitete praćenja te metode postižu vrhunske rezultate, ali i dalje je to još vrlo eksperimentalno područje bez dobro uhodanih smjernica. Glavni razlozi za to su mnogo veći nedostatak kvalitetnih skupova za učenje u odnosu na problem detekcije ili semantičke segmentacije te sam problem praćenja koji je osjetno drugačiji. Zbog toga je broj radova napravljenih na tu temu osjetno manji u odnosu na radove temeljene na detekciji. Ovaj rad se stoga bavi upravo tim problemom primijenjenim na specifičan problem praćenja

igrača na nogometnim utakmicama. Iako je problem kojim se bavi ovaj rad specifičan rješenja koja su dobivena bi trebala dobro generalizirati i na drugim problemima praćenja objekata koji nisu nužno igrači. S tim na umu razvijena su 2 modela praćenja koji se u manjoj ili većoj mjeri oslanjaju na konvolucijske neuronske mreže. Svi modeli su trenirani i testirani na videu nogometne utakmice 1. HNL između Dinama i Splita koji je sniman iz ptičje perspektive i koji dobro simulira realne uvjete. Za svaki model je mjerena performansa koliko dobro prati objekte i brzina izvođenja te na kraju izveden zaključak.

Slika 1: Primjer praćenja objekata na videu [1]. Objekti koji se prate su označeni plavim pravokutnicima.

2. Osnove konvolucijskih neuronskih mreža

2.1. Operacija konvolucije i konvolucijski sloj

Konvolucija [2] je matematička operacija definirana nad dvije funkcije realne ili kompleksne varijable koja kao izlaz daje treću funkciju koja je definirana kao količina preklapanja između prve funkcije te okrenute i pomaknute druge funkcije. Prva funkcija u konvoluciji se najčešće označuje slovom f, druga funkcija sa slovom g, a pomak sa slovom t. Konvolucija realne neprekinute funkcije je definirana formulom (2.1):

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{2.1}$$

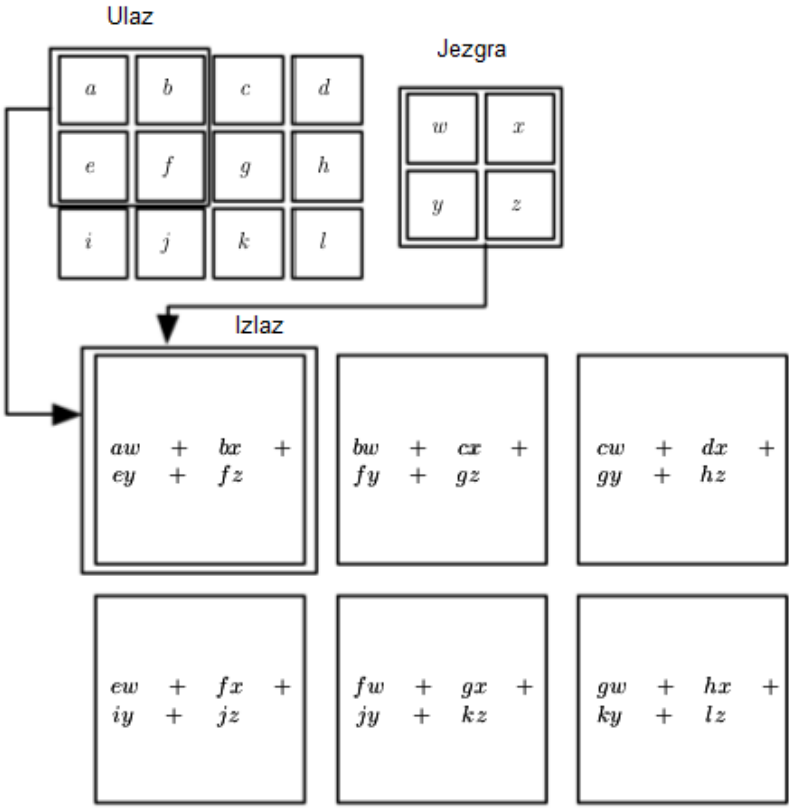
Operacija konvolucije je komutativna, asocijativna, distributivna i asocijativna sa skalarnim množenjem. Radi svih tih svojstava ima veliku uporabu u područjima poput obrade signala, statistike, računalnog vida, itd. Ukoliko se konvolucija koristi u području računalnog vida tada se koristi njezina diskretna varijanta jer su podaci najčešće diskretizirani i ima ih konačan broj (npr. slika u boji veličine 3000 x 2000 piksela sa tri kanala za crvenu, plavu i zelenu boju od kojih svaki kanal ima veličinu od 1 bajta). Konvolucija sa diskretnim varijablama gdje je funkcija g definirana konačnim skupom brojeva $\{-M, -M+1, \dots, M-1, M\}$ je dana formulom (2.2):

$$(f * g)[n] = \sum_{m=-M}^{M} f[n-m]g[m] \tag{2.2}$$

U računalnom vidu kod konvolucije u neuronskim mrežama funkcija f se naziva ulazom, funkcija g jezgrom (*eng. kernel)*, a izlaz operacije konvolucije mapom značajki. Ukoliko su ulazni podaci dvodimenzionalni, što je u računalnom vidu čest slučaj, konvolucija se radi po obje dimenzije. Također, pošto je konvolucija komutativna najčešće se radi jednostavnosti računalne implementacije jezgra okreće s obzirom na ulaz kao što je vidljivo u formuli (2.3):

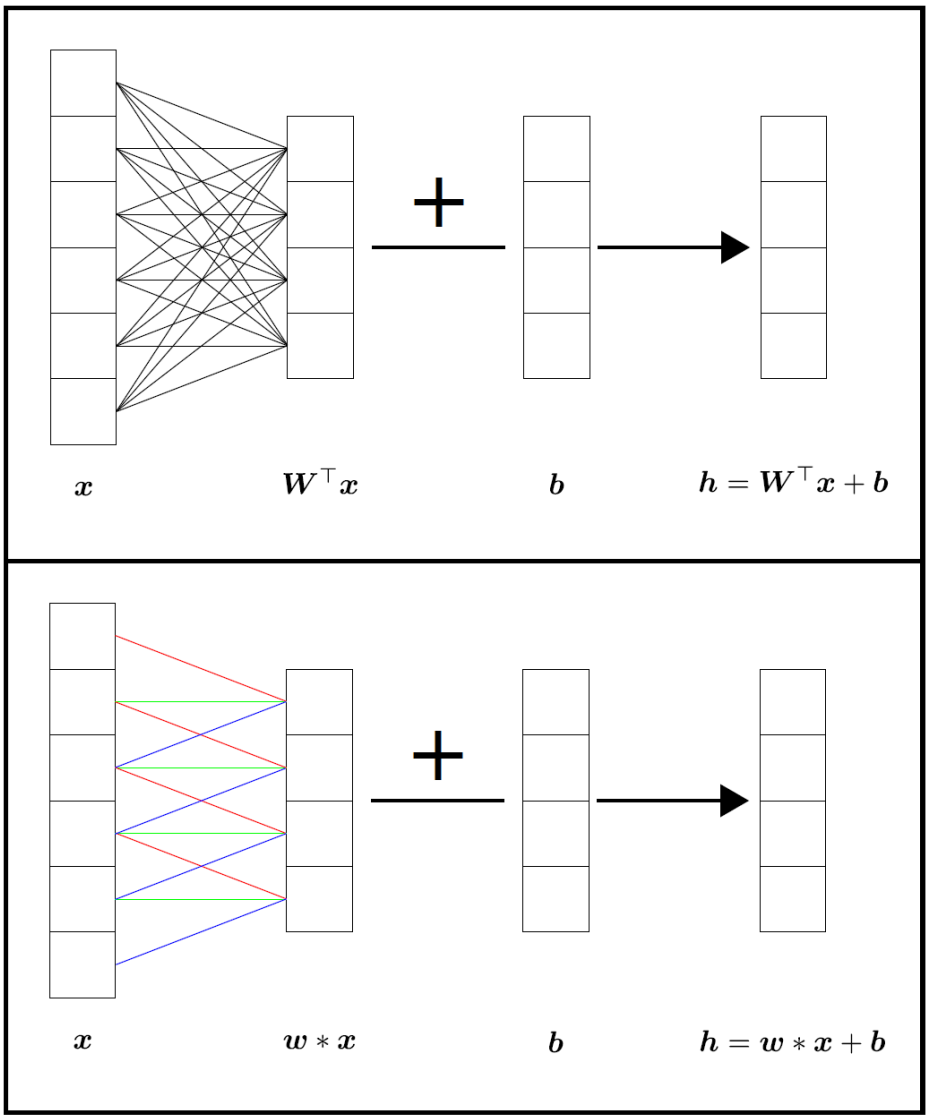
$$(f * g)(i,j) = \sum_{m=m_{min}}^{m_{max}} \sum_{n=n_{min}}^{n_{max}} g[i-m, j-n]f[m,n] \tag{2.3}$$

U primjeni u računalnom vidu jezgra je najčešće kvadratnog oblika te se njezina veličina najčešće izražava širinom u pikselima. Grafički primjer dvodimenzionalne konvolucije definirane u prethodnoj formuli sa kvadratnom jezgrom širine 2 je prikazan na slici ispod:



Slika 2: Rezultat dvodimenzionalne konvolucije [3] bez nadopune.

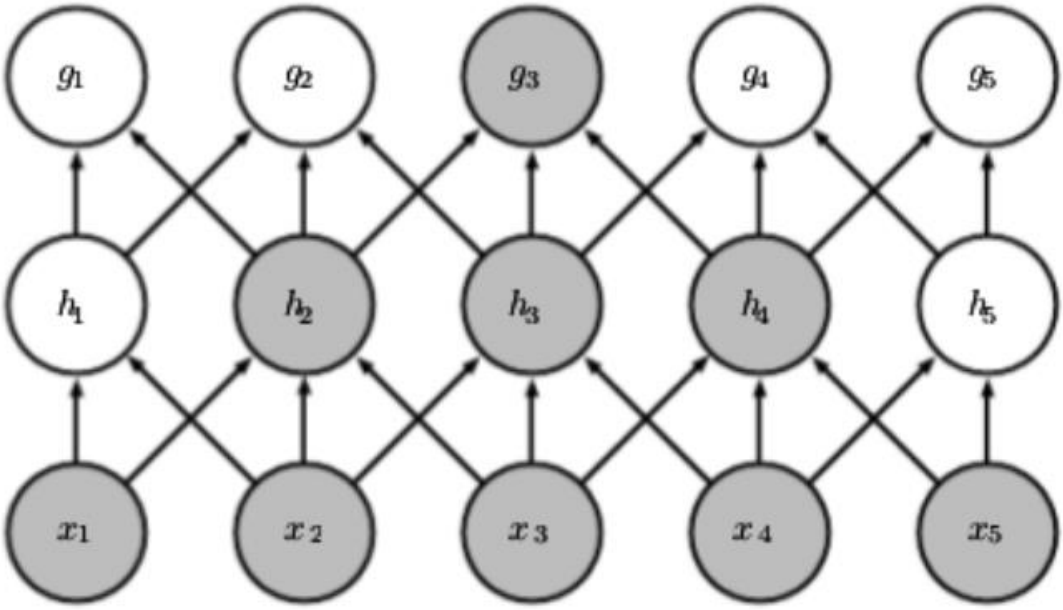
Konvolucija je veoma korisna u neuronskim mrežama koje su namijenjene za rad sa dvodimenzionalnim podacima poput slika jer je jako slična potpuno povezanom sloju pa je jednostavna za integraciju unutar mreže, a ima svojstvo da modelira samo lokalne interakcije (jedan neuron nije povezan sa svima prethodnima) i podržava dijeljenje parametara što znači da je reprezentacija izlaza konvolucije ekvivarijantna s obzirom na pomak [3]. Stoga sloj u neuronskoj mreži koji je u potpunosti sastavljen od konvolucijskih operacija nazivamo **konvolucijskim slojem** neuronske mreže. Primjer razlike konvolucijskog sloja u odnosu na potpuno povezani sloj je prikazan na slici 3, sa oznakama x za ulazni vektor, W za težine potpuno povezanog sloja, w za težine u konvolucijskom sloju, b za varijable pristranosti u potpuno povezanom i konvolucijskom sloju te h kao izlaz konvolucijskog i potpuno povezanog sloja:



Slika 3: Usporedba potpuno povezanog (gore) i konvolucijskog (dolje) sloja [3]. Vidljivo je osjetno smanjenje povezanosti između potpuno povezanog i konvolucijskog sloja. To vodi k osjetno manjem broju parametara za učenje.

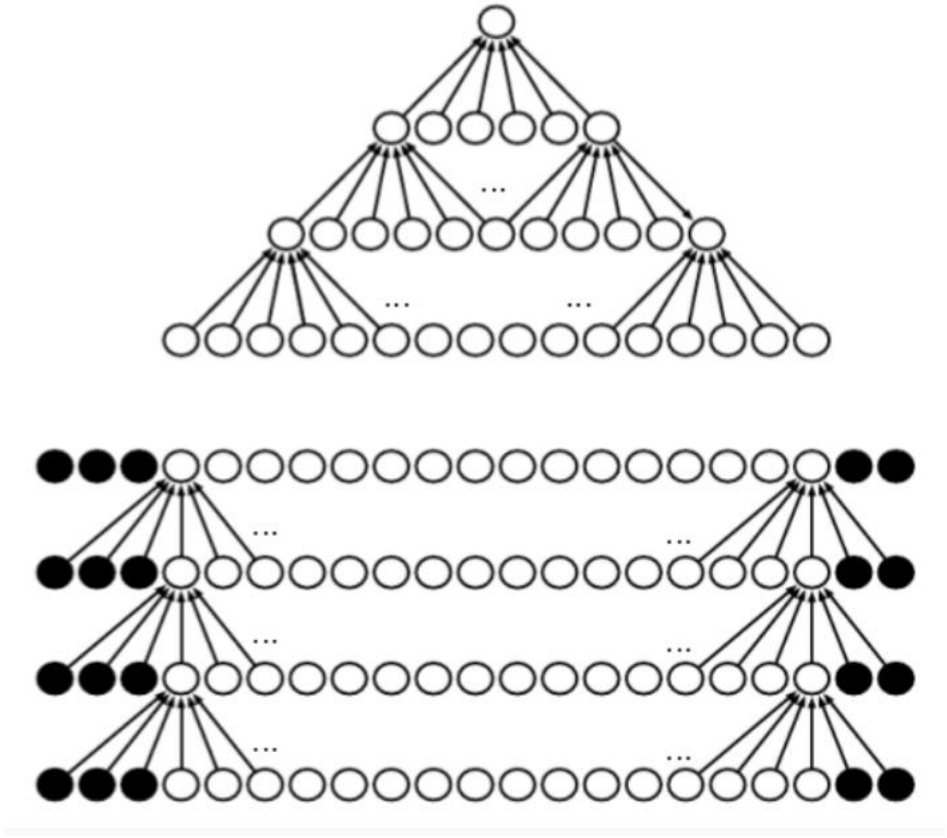
Kao što je iz slike vidljivo konvolucijski sloj je povezan sa mnogo manjim brojem ulaza što donosi prednost mnogo manjeg broja parametara u tom sloju. Prva prednost toga je brža evaluacija podataka kroz mrežu jer sada imamo složenost $O(kn)$ umjesto $O(mn)$, gdje je n veličina trenutnog sloja, k veličina jezgre konvolucijskog sloja, a m veličina ulaznog vektora te vrijedi $k \ll m$. Druga prednost manjeg broja parametara je osjetno lakše i brže učenje parametara konvolucijskog sloja prilikom faze učenja same mreže [3]. Još jedno jako korisno svojstvo konvolucijskih slojeva u neuronskoj mreži jest da značajke (pojedinačni neuroni) modeliraju indirektnu interakciju većeg dijela ulaznih značajki. To znači da na pojedinačni neuron u konvolucijskoj neuronskoj mreži ustvari utječu značajke prijašnjih slojeva gdje širina interakcije ovisi o tome koliko je široka jezgra prethodnih konvolucijskih slojeva. Te

značajke se nazivaju **receptivnim poljem** konvolucijske neuronske mreže. Ilustracija receptivnog polja je prikazana na slici 4:



Slika 4: Receptivno polje neurona g_3 [3]. Sa slike je vidljivo da receptivno polje ovisi o veličine konvolucijske jezgre i broju konvolucijskih slojeva prije sloja u kojem se gleda receptivno polje.

Još jedna optimizacija koja se može napraviti u konvolucijskim slojevima neuronskih mreža je **dijeljenje parametara** [3]. To znači da je skup težina koje koristi jedan neuron isti kao i skup težina koje koristi i drugi neuron u rešetci. Prednost toga je da umjesto da učimo odvojen skup parametara za svaki od n izlaza, svaki izlaz dijeli jedan skup parametara. S time se dobiva još manji model sa k parametara što znači da se model još jednostavnije i brže nauči. Računska složenost ostaje i dalje $O(kn)$. Dodatna bitna stvar kod konvolucijskog sloja je rješavanje problema **nadopune** (*eng. padding*). Ukoliko nema nadopune rubova ulaznog vektora prilikom operacije konvolucije tada se reprezentacija smanjuje sa dubinom mreže. To znači da je vektor izlaza svaki put sve kraći pa je i dubina same mreže ograničena time, npr. za vektor ulaza dužine m i jezgru širine k izlaz je duljine $m - k + 1$. Ukoliko se rubovi ulaza nadopune nulama taj se problem rješava i izlaz je uvijek jednake širine pa i sama mreža nije ograničene dubine. Glavni nedostatak te metode je da elementi na rubovima manje utječu na izlaz nego elementi u sredini. Konvolucija sa nadopunom i bez nje je prikazana na slici 5:



Slika 5: Konvolucija bez nadopune (gore) i sa nadopunom (dolje) [3]. Vidljivo je da ukoliko se konvolucija radi bez nadopune da je krajnja dubina mreže ograničena zbog smanjivanja širine slojeva prilikom svake operacije konvolucije. Taj problem se može riješiti sa konvolucijom sa nadopunom.

2.2. Funkcija i sloj sažimanja

Funkcija sažimanja je (*eng. pooling function)* je funkcija koja preslikava skup prostorno bliskih značajki na ulazu u jednu značajku na izlazu [3]. Ona obično računa neku statističku vrijednost skupa vrijednosti značajki na ulazu kao npr. srednju, maksimalnu ili L2 vrijednost. Glavni razlog zašto se ona koristi u konvolucijskim mrežama jest povećanje invarijantnosti na pomak. Definicija invarijantnosti je definirana na ovaj način. Uzmimo funkciju $f(x)$ i $g(x)$. Funkcija $f(x)$ je invarijantna s obzirom na g ako vrijedi izraz (2.4):

$$f\big(g(x)\big) = f(x) \tag{2.4}$$

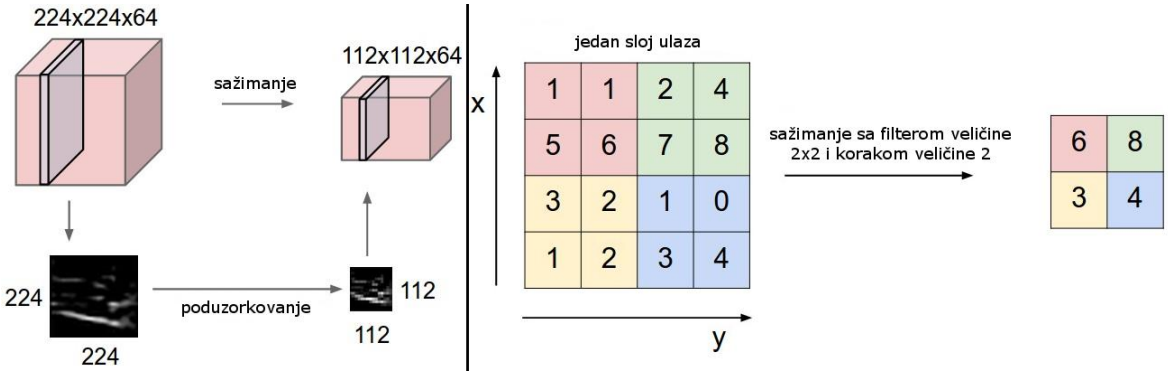
Invarijantnost na pomak je vrlo korisno svojstvo ako pretpostavimo da je za raspoznavanje bitnije detektirati prisutnost neke značajke nego njezinu točnu lokaciju. Veličina regije sažimanja određuje količinu invarijantnosti sa pravilom ako je regija veća onda invarijantnost vrijedi za veće pomake. Drugi razlozi za uključivanje funkcije sažimanja u zaseban sloj neuronske mreže jest da se progresivno smanji veličina reprezentacije ulaza,

smanji broj parametara u mreži, skrati vrijeme računanja izlaza iz mreže te kao kontrola za pretreniranost mreže [4]. Svaki sloj sažimanja prima dva hiperparametra (parametra na koje možemo utjecati). Prvi od njih je **veličina filtera** (*eng. spatial extent)*. On određuje koliko ćemo susjednih članova ulaza uzeti u obzir prilikom računanja sažimanja. Drugi hiperparametar je **korak** (*eng. stride*). On određuje za koliko se filter pomiče prilikom računanja izlaza. Veličina izlaza za ulaz veličine $W \times H$ sa veličinom filtera F i veličinom koraka S se računa prema sljedećoj formuli (2.5):

$$W_{izlaz} = \frac{W - F}{S} + 1$$

$$H_{izlaz} = \frac{H - F}{S} + 1 \tag{2.5}$$

Grafički prikaz sažimanja funkcijom maksimalne vrijednosti je prikazan na slici 6:



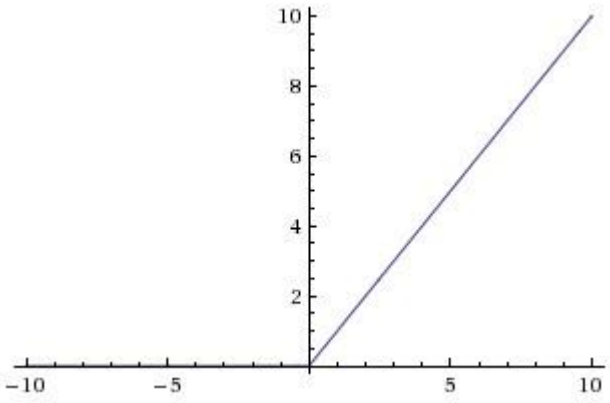
Slika 6: Sažimanje funkcijom maksimalne vrijednosti sa filterom veličine 2x2 i korakom veličine 2 [4].

Iako je sloj sažimanja vrlo često u upotrebi u konvolucijskim mrežama postoje mreže koje ga uopće ne koriste. Umjesto njega se koriste samo konvolucijski filteri sa većim korakom svakih nekoliko slojeva. Pokazalo se da to vodi k jako dobrim rezultatima prilikom treniranja generativnih modela poput varijacijskih autoenkodera. Zbog toga je moguće da će buduće duboke neuronske mreže imati manji broj slojeva sažimanja ili će oni biti čak skroz izostavljeni [4].

2.3. Konvolucijske neuronske mreže

Konvolucijske neuronske mreže su jako slične uobičajenim neuronskim mrežama što znači da su sastavljene od neurona koji se sastoje od varijabla težina (*eng. weights*) i pristranosti (*eng. bias*) koje se mogu podešavati prilikom učenja mreža. To također znači da svaki neuron prihvaća ulazni vektor, računa skalarni umnožak sa vlastitim težinama, pridodaje tom iznosu varijablu pristranosti i na kraju dodaje, opcionalnu, nelinearnost. Funkcija nelinearnost koja se u praksi pokazala najboljom za duboke neuronske mreže je **ReLU**. Ona je definirana formulom (2.6) i prikazana je na slici (7):

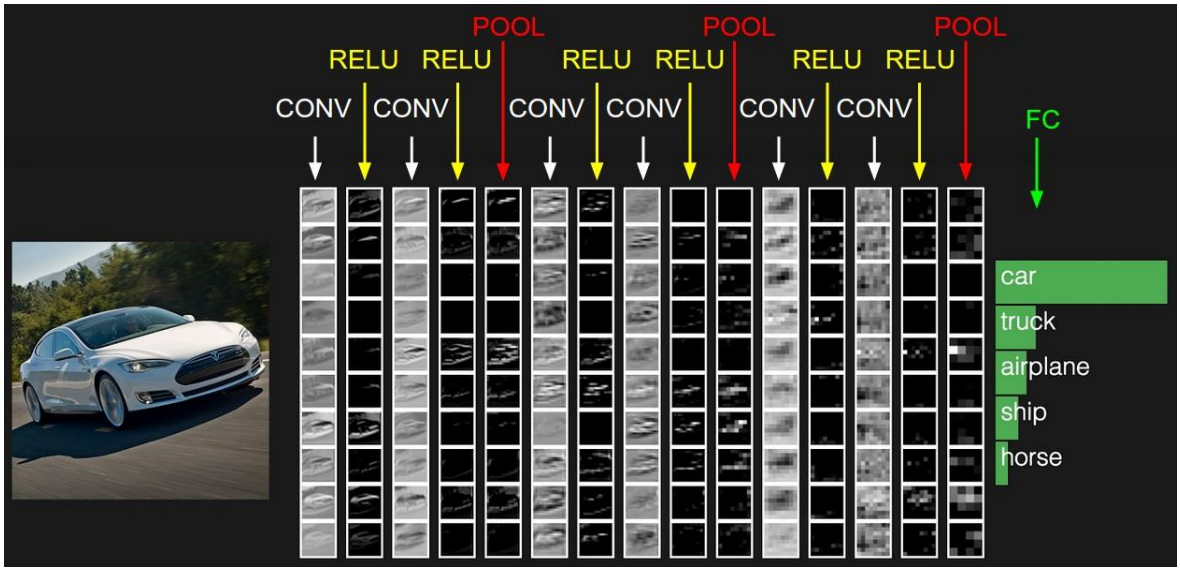
$$ReLU(x) = \max(0, x) \tag{2.6}$$



Slika 7: Graf funkcije ReLU [5].

Glavne prednosti korištenja funkcije ReLU kao nelinearnosti na izlazu iz neurona su veliko ubrzanje konvergencije prilikom učenja mreže što se smatra da je radi toga što ona ne ulazi u područje zasićenja za pozitivne brojeve te mnogo efikasnije i brže računanje izlaza jer se ne trebaju računati računalno skupe operacije poput eksponenciranja [5]. Sam izlaz iz obične konvolucijske mreže je i dalje vjerojatnost pripadanja ulaza nekom razredu tako da sva pravila i algoritmi učenja mreža rade dobro na njima. Ono što razlikuje običnu od konvolucijske neuronske mreže jest to što konvolucijska mreža ima jedan ili više konvolucijskih slojeva u sebi dok obična mreža nema nijedan. Druga stvar koja razlikuje ta dva tipa mreža jest to što su konvolucijske mreže najčešće usko specijalizirane za ulazne podatke tako da primaju samo slike na ulaz. Ta eksplicitna pretpostavka omogućava dodavanje nekih svojstava u mrežu koja ju čine mnogo efikasnijom [5]. Prva stvar koju kovolucijske mreže rješavaju jest problem skaliranja običnih mreža ako je ulaz cijela slika. Potpuno povezani slojevi imaju ogroman parametara te stoga pate od problema teškog treniranja i lakog pretreniranja. Npr. za sliku veličine 200x200x3 piksela prvi skriveni sloj

bi imao 120 000 težina, a pošto bi bilo potrebno dodati još slojeva taj broj bi jako brzo narastao daleko više [5]. Konvolucijske neuronske mreže taj problem rješavaju uz pomoć konvolucijskih slojeva. Kao što je objašnjeno prije oni nemaju potpunu povezanost sa svakim elementom ulaza te skoro uvijek imaju implementirano i dijeljenje parametara te tako imaju mnogo manji broj težina za učenje. Također konvolucijske mreže iskorištavaju dimenzije ulaza u mrežu te neurone slažu u trodimenzionalne tenzore što daje i trodimenzionalne (ili čak i višedimenzionalne) izlaze u idući slojeve. S time se ostvaruje arhitektura mreže koja je bliska tipu ulaza što omogućuje osjetno bolje performanse. Iz svega do sad navedenog se da zaključiti da se obična konvolucijska mreža sastoji od tri vrste slojeva: konvolucijskih slojeva, slojeva sažimanja i potpuno povezanih slojeva na izlazu. Tipičan primjer konvolucijske neuronske mreže prikazan je na slici 8:



Slika 8: Primjer konvolucijske neuronske mreže [4]. Na slici se vide konvolucijski slojevi (CONV), slojevi nelinearnosti (RELU), slojevi sažimanja (POOL) te potpuno povezani sloj (FC). Izlaz iz mreže su vjerojatnosti pripadanja slike određenom razredu.

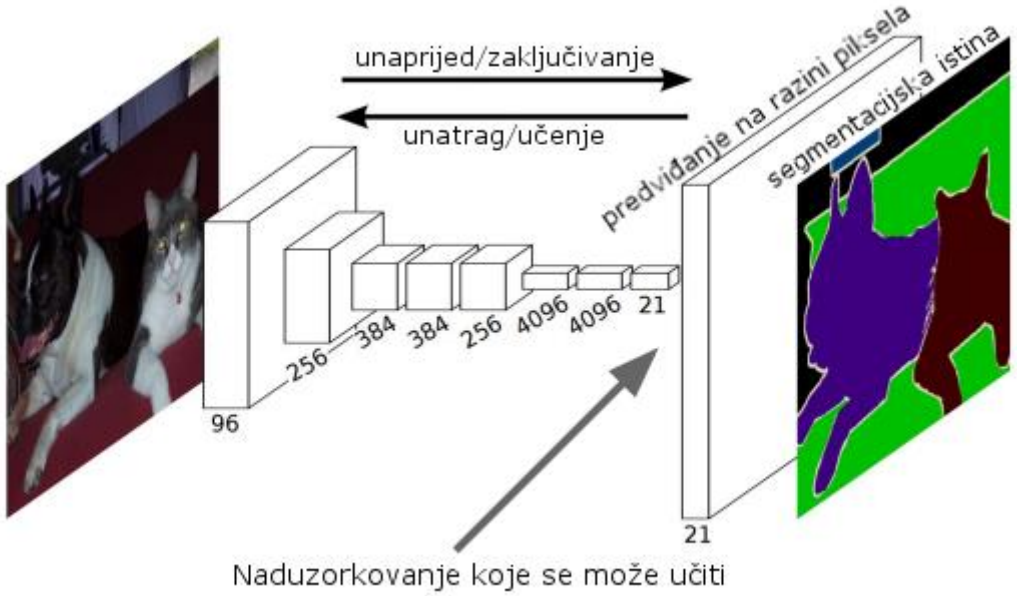
Kao što je iz slike vidljivo ulaz u mrežu je slika koja se šalje na konvolucijski sloj. Svaki konvolucijski sloj može, ali i ne mora imati nelinearnost na izlazu. Uobičajena je praksa staviti par konvolucijskih slojeva jedan iza drugog prije sloja sažimanja. Na taj način se povećava kapacitet mreže, receptivno polje neurona i poboljšava detekcija. Svaki sloj sažimanja smanjuje veličinu slike za daljnje ulaze te također povećava receptivno polje. Tako se najčešće slaže par slojeva jedan za drugim prije dolaska do potpuno povezanog sloja. Potpuno povezani slojevi su obično na kraju mreže te služe za dobivanje krajnjeg izlaza kao vjerojatnosti za svaki razred objekta zasebno. To se može vidjeti na slici 8. Takve konvolucijske mreže se najčešće koriste za klasifikaciju cijelih slika u zasebne razrede.

Tipičan primjer za to je recimo testni skup ImageNet[1]. Duboke mreže ovakvog tipa su prve polučile osjetno bolje rezultate u području računalnog vida u odnosu na metode koje su se do tad koristile. Primjeri takvih mreža su VGG19[2], GoogleNet[3] i ResNet[4]. Iako ovakve mreže daju odlične rezultate, sve više rezultata pokazuje prema smjeru smanjivanja broja slojeva sažimanja te broja potpuno povezanih slojeva. Razlog micanje slojeva sažimanja je naveden u prošlom poglavlju, a razlog za micanje potpuno povezanih slojeva jest osjetno smanjenje broja težina za učenje i evaluaciju. Najbolji primjer za to jest usporedba mreža VGG19 i ResNet. VGG19 ima dubinu od 19 slojeva te uključuje 3 potpuno povezana sloja na kraju. Većinom radi njih mreža ima 144 milijuna parametara uz top-5 grešku na setu ImageNet od 6.8%. Za usporedbu, mreža ResNet ima dubinu od 152 sloja sa 1,7 milijuna parametara i samo jednim potpuno povezanim slojem. Njena top-5 greška iznosi 3,6%. Iz toga je vidljivo da potpuno povezani slojevi ne pridonose puno kvaliteti klasifikacije te se mogu izostaviti za razliku od konvolucijskih slojeva kojih je bolje staviti čim više. Kompletno micanje potpuno povezanih slojeva iz konvolucijske mreže vodi k novom tipu mreža koje se zovu potpuno konvolucijske mreže.

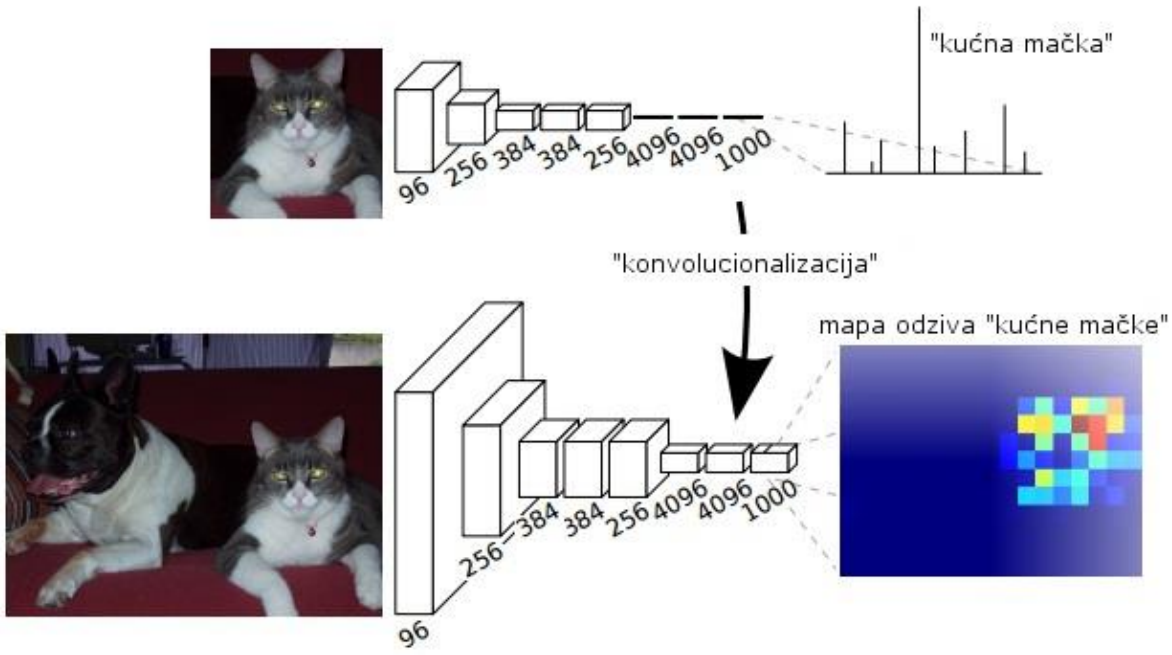
2.4. Potpuno konvolucijske mreže

Potpuno konvolucijske mreže su neuronske mreže koje se sastoje samo od konvolucijskih slojeva uz opcionalne slojeve sažimanja. U praksi to znači da je zadnji potpuno povezani sloj zamijenjen sa još jednim konvolucijskim slojem koji ima veliko receptivno polje. Glavna ideja je uhvatiti globalni kontekst scene. Također ako se zadnji potpuno povezani sloj zamijeni sa konvolucijskim s time se dobije neki oblik lokalizacije ukoliko se gledaju aktivacije u tom sloju. Na kraju je glavni cilj napraviti zadnji konvolucijski sloj dovoljno velikim da se taj lokalizacijski efekt može skalirati na veličinu ulazne slike. Prikaz potpuno konvolucijske mreže i aktivacije u zadnjem sloju prikazane su na slikama 9 i 10:

[1] http://www.image-net.org
[2] https://arxiv.org/abs/1409.1556
[3] https://arxiv.org/abs/1409.4842
[4] https://arxiv.org/abs/1512.03385



Slika 9: Potpuno konvolucijska mreža za semantičku segmentaciju [6]. Izlaz iz potpuno konvolucijske mreže jest slika sa odzivima svakog piksela koja se tad prilikom učenja uspoređuje sa stvarnim oznakama piksela na toj slici. Nakon faze učenja mreža daje odzive svakog piskela za svaki razred koji ona treba raspoznati.



Slika 10: Razlika između aktivacija izlaza obične i potpuno konvolucijske mreže [6]. Aktivacije obične konvolucijske neuronske mreže su vjerojatnost pripadanja razredu dok su za potpuno konvolucijsku mrežu to regije na slici koje imaju iznos odziva veći od 0.

Kao što je vidljivo iz primjera na slikama potpuno konvolucijske neuronske mreže imaju svoju glavnu primjenu u semantičkoj segmentaciji slika. Ako se mreža koristi za tu svrhu kao izlaz iz mreže dobiva se izlazni tenzor odziva gdje su u matricama raspoređeni odzivi

za svaki pojedinačni razred. Sve što je tada potrebno je pronaći najveći odziv za svaki razred i spojiti to u izlaznu segmentiranu sliku. Veoma često se mreže namijenjene za klasifikaciju trenirane na setu ImageNet pretvaraju u potpuno konvolucijske modele namijenjene za segmentaciju. To se može napraviti u 3 koraka. Prvi korak je uzeti konvolucijsku mrežu koja nam odgovara po svojstvima ili rezultatima. U drugom koraku se uzmu svi potpuno povezani slojevi i pretvore se u konvolucijske slojeve veličine 1×1. Već smo s ovim korakom dobili potpuno konvolucijsku mrežu. Dobro svojstvo koje se također s ovim korakom dobilo je da se sad također mogu koristiti slike proizvoljne veličine. U skladu s tim je moguće i promijeniti veličine slojeva da bolje odgovaraju ulazu. Obično je dobra praksa staviti veličine slojeva na veličinu originalnog ulaznog podijeljenog sa 2^n. Zadnji korak jest dodavanje sloja „dekonvolucije“ („transponirane konvolucije“) da se aktivacijski položaji stave u kontekst ulazne slike. Pojednostavljeno, to znači da se aktivacije skaliraju na veličinu ulazne slike. Taj sloj također ima parametre za učenje. Prikaz pretvaranja obične konvolucijske mreže u potpuno konvolucijsku mrežu prikazan je na slici 11 koja se nalazi na sljedećoj stranici:

Slika 11: Pretvaranje obične konvolucijske mreže u potpuno konvolucijsku mrežu [6]. Prvi korak je pretvaranje potpuno povezanih slojeva u konvolucijske slojeve veličine 1×1. Idući korak je podešavanje veličine tih slojeva ovisno o dimenzijama ulaza. Zadnji korak je dodavanje sloja sa naduzorkovanjem koji daje izlaz iste veličine kao i ulaz.

Na taj se način uz malo ponovnog učenja mreže dobivaju jako dobri rezultate u području semantičke segmentacije. Ono što je također bitno napomenuti jest da su funkcije gubitka s kojim se uči potpuno konvolucijska mreža potpuno iste kao i za običnu konvolucijsku mrežu. To znači da se mogu i dalje koristiti gubitak unakrsne entropije kao i gubici izvedeni iz nje.

2.5. Funkcije gubitaka konvolucijskih neuronskih mreža

Svaka neuronska mreža mora imati definiranu funkciju gubitka prilikom faze učenja i validacije. To je funkcija preko koje se određuje količina greške koju mreža radi prilikom svake iteracije učenja te uz pomoć koje se mogu provesti metode učenja gradijentnim spustom. Daleko najpopularnija funkcija gubitka potpuno konvolucijskih neuronskih mreža jest unakrsna entropija i njezine modificirane verzije. **Gubitak unakrsne entropije** (*eng. cross-entropy loss*) jest funkcija koja je preuzeta iz područja teorije informacije i ona je definirana formulom (2.7) [7]:

$$L(w) = -\frac{1}{N}\sum_{n=1}^{N}[y_n \log(\hat{y}_n) + (1 - y_n)\log(1 - \hat{y}_n)] \tag{2.7}$$

w su težine u mreži koje se optimiziraju, $\hat{y}_n \equiv g(w \cdot x_n)$, $g(z)$ je logistička funkcija, a y_n su željene vrijednosti izlaza. Ova funkcija je vrlo čest odabir za učenje svih vrsta neuronskih mreža te se nalazi implementirana u velikom broju biblioteka za rad sa neuronskim mrežama. Druga funkcija gubitka koja se često upotrebljava jest **uravnoteženi gubitak unakrsne entropije**. To je modifikacija gubitka unakrsne entropije modificirana za rad sa skupovima podataka gdje su određeni razredi podataka mnogo češće pojavljuju od drugih. Taj gubitak je definiran formulom (2.8):

$$L(w) = -\frac{1}{N}\sum_{n=1}^{N}[y_n \log(\hat{y}_n) + (1 - y_n)\log(1 - \hat{y}_n)] \cdot \frac{1}{p_{apriori}(y_n)} \tag{2.8}$$

Ta funkcija je definirana jako slično kao i funkcija gubitka unakrsne entropije, ali sa razlikom da uzima u obzir i frekvenciju pojavljivanja svakog razreda, $p_{apriori}(y_n)$, u podacima za učenje te na taj način pojačava važnost učenja razreda koji imaju nižu frekvenciju pojavljivanja te koji bi inače bili gotovo zanemareni prilikom učenja. S njom se dobiva ravnomjernije učenje u situacijama kada razredi nisu podjednako raspoređeni. Problem koji se javlja prilikom korištenja ove funkcije jest ako je neki razred izrazito rijedak u podacima za učenje. U tom slučaju ova funkcija će forsirati učenje mreže za taj razred na štetu svih ostalih. Taj problem se obično rješava na način da se preveliki gradijent podreže (*eng. clipping*) na neki maksimalno određeni iznos.

2.6. Moderni algoritmi učenja konvolucijskih neuronskih mreža

Moderni algoritmi učenja neuronskih mreža su se nedavno pojavili i to kao nadogradnja na metodu učenja gradijentnim spustom. Gradijenti spust je dugo vremena bio primarna metoda učenja neuronskih mreža, pogotovo ako se koristi u stohastičkom obliku. Iako je gradijentni spust najpopularnija metoda učenja mreža on ima dosta problema koji su posebno došli do izražaja sa dubokim modelima. Glavni problem je taj što su neuronske mreže problem nekonveksne optimizacije što znači da lokalni minimum u funkciji mreže nije nužno i globalni minimum. Taj problem je izrazito izražen u dubokim modelima sa mnogo lokalnim minimuma. Drugi problemi koje vrijedi spomenuti su problemi strmih regija u funkcijama gubitaka i eksplodirajućih gradijenata na njima, problem sedla i platoa, itd. [8]. Moderni algoritmi učenja pokušavaju riješiti te probleme uz pomoć adaptivnih stopa učenja. Tri najpopularnija moderna algoritma za učenje neuronskih mreža koji koriste te metode su AdaGrad, RMSProp i Adam. Algoritam **AdaGrad** je razvijen 2011. godine i on adaptira stope učenja parametara skalirajući ih inverzno proporcionalno kvadratnom korijenu sume svih kvadriranih vrijednosti pripadne parcijalne derivacije funkcije gubitaka kroz povijest [8]. To znači da će parametri koji imaju velike parcijalne derivacije brzo početi koristiti male stope učenja, a parametri koji imaju male parcijalne derivacije će dugo vremena koristiti razumno velike stope učenja. Skaliranje stope učenja je konzistentno sve jače i nema mogućnosti opravka što znači da taj algoritam može i dalje imati problem sa nekonveksnošću funkcije gubitaka neuronskih mreža. Pseudokod algoritma AdaGrad napisan na engleskom jeziku je prikazan na slici 12:

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, perhaps 10^{-7}, for numerical stability
  Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
    Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

Slika 12: Pseudokod algoritma AdaGrad [8]

Algoritam **RMSProp** je razvijen 2012. godine i radi na sličan način kao i AdaGrad, ali uz jedno bitno poboljšanje. On mijenja akumuliranje gradijenata tako da dodaje eksponencijalno prigušivanje starih vrijednosti što znači da novije vrijednosti imaju veći utjecaj. U praksi to znači da ako komponenta gradijenta u nekom trenutku padne, relativno brzo će pasti i njeno prigušivanje pa će korekcije ponovno povećati. Zbog toga je ovaj algoritam najčešće bolji odabir za nekonveksne probleme. Pseudokod algoritma RMSProp napisan na engleskom jeziku je prikazan na slici 13:

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ.
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers.
Initialize accumulation variables $\boldsymbol{r} = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
end while

Slika 13: Pseudokod algoritma RMSProp [8]

Algoritam **Adam** je razvijen 2014. godine i on je skraćenica od engleskih riječi „**Ada**ptive **M**oment Estimation“ [9]. Adam u sebi ima ugrađen efekt momenta (fizikalna ekvivalencija inercije, ali primijenjena za gradijent). Adam sprema eksponencijalno smanjujuće prosjeke prošlih kvadriranih gradijenata v_t i eksponencijalno smanjujući prosjek prošlih gradijenata m_t što se može interpretirati kao efekt momenta. Računanje se radi po izrazu (2.9):

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

(2.9)

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

m_t je procjena prvog momenta (srednje vrijednosti), a v_t je procjena drugog momenta (necentrirane varijance). Problem koji se javlja ako su ti vektori inicijalizirani na nulu, a koji je pristranost prema nuli, rješava se na način da se procjene prvog i drugog momenta računaju sa prepravljenom pristranošću prema izrazu (2.10):

$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t}$$

(2.10)

$$\hat{v}_t = \frac{m_t}{1-\beta_2^t}$$

Osvježavanje parametara se računa prema izrazu (2.11):

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t+\epsilon}}\widehat{m}_t \tag{2.11}$$

Preporučuju se uzeti vrijednosti od 0.9 za β_1, 0.999 za β_2 te 10^{-8} za ϵ. Pseudokod algoritma Adam napisan na engleskom jeziku je prikazan na slici 14:

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$
    Compute update: $\Delta\boldsymbol{\theta} = -\epsilon\frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}}+\delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

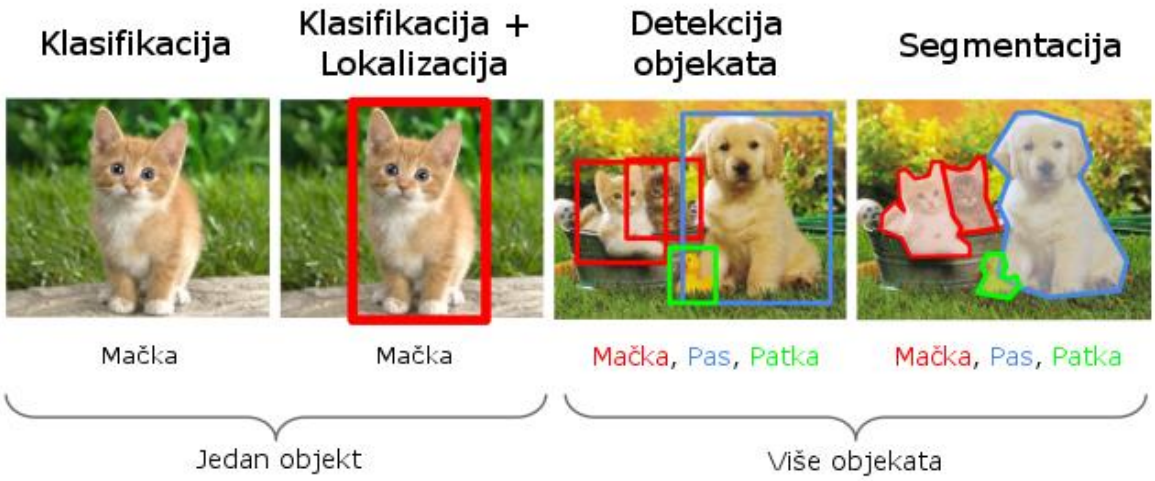
Slika 14: Pseudokod algoritma Adam [8]

Adam je najrobusniji od svih navedenih algoritama te se preporučuje kao najbolji odabir za učenje neuronskih mreža.

3. Problem detekcije i praćenja objekata

3.1. Definicija problema detekcije objekata

Problem detekcije objekata na slikama ili videu može se podijeliti u 4 različite kategorije ili problema, a to su: **klasifikacija**, **klasifikacija kombinirana sa lokalizacijom**, **detekcija objekata** i **segmentacija**. Razlika između tih problema je prikazana na slici 15:



Slika 15: Kategorije detekcije objekata na slikama [10] od najlakše do najteže (prema desno)

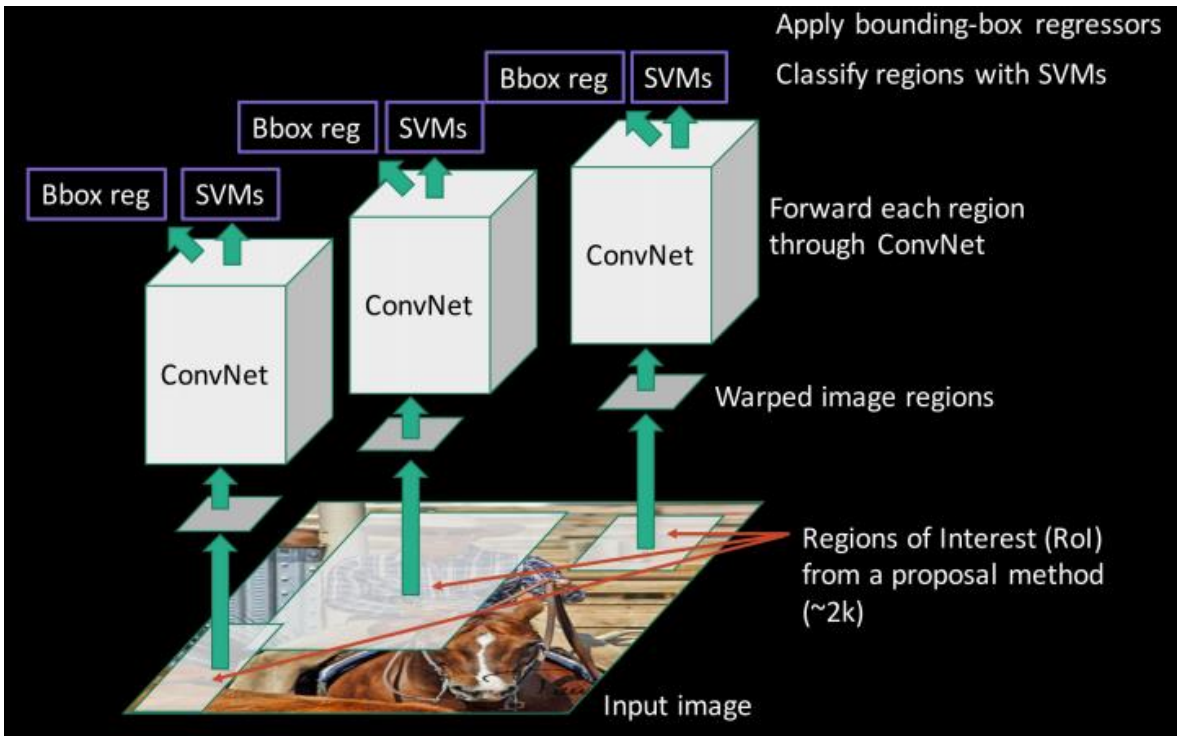
Najjednostavniji od ta četiri problema jest problem klasifikacije. U njemu je potrebno samo detektirati kojem razredu pripada slika ovisno o objektu koji se nalazi na njoj. Iako je najjednostavniji taj problem je i dalje veoma težak, pogotovo ukoliko ima mnogo različitih razreda za prepoznavanje. Dobar primjer testa za taj problem jest već ranije spomenuti skup podataka ImageNet. Malo kompliciraniji problem od čiste klasifikacije slike jest klasifikacija kombinirana sa lokalizacijom. U tom slučaju se osim klasifikacije razreda slike mora definirati i gdje se objekt koji pripada tom razredu nalazi na toj slici. To najčešće uključuje neku vrstu regresije koja se primjenjuje radi lokalizacije. Treći problem jest detekcija objekata. To je problem koji je veoma sličan klasifikaciji i lokalizaciji samo što je je još otežan zahtjevom da se na jednoj slici mora pronaći više objekata koji mogu pripadati različitim razredima. Segmentacija primjeraka jest problem detekcije koji se zahtjeva klasifikacija u pripadajući razred za svaki pojedinačni piksel na slici. Skup podataka koji je primjer testa za ova navedena zadnja tri problema jest Pascal VOC[5]. Ovi problemi su se ranije rješavali klasičnim metodama računalnog vida i strojnog učenja kao npr. detekcija uz pomoć HAAR metode ili HOG metode te klasifikacijom uz pomoć SVM-a ili slučajnih

[5] http://host.robots.ox.ac.uk/pascal/VOC/

šuma. Otkad su duboki konvolucijski modeli srušili rekorde u kvaliteti klasifikacije i detekcije većina istraživanja se okrenula prema tim metodama te je razvijeno mnogo novih metoda za duboke modele.

3.2. Detekcija modelima RCNN, Fast RCNN i Faster RCNN

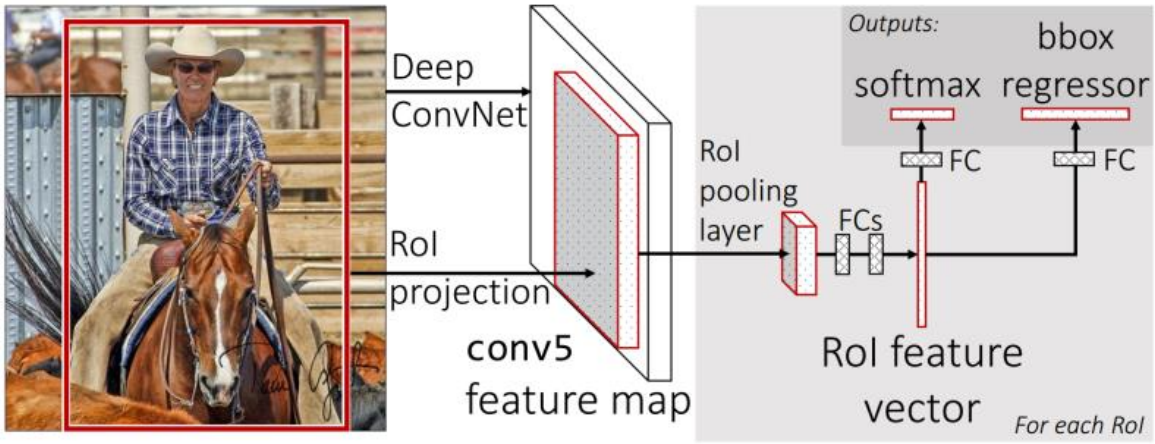
RCNN (***eng. Regions + CNN***) jest metoda detekcije koja uključuje konvolucijsku neuronsku mrežu zajedno sa vanjskom metodom predlaganja pogodnih regija. U njoj prvo vanjski predlagatelj regija da određen broj regija (pravokutnika) u slici. Najčešće taj broj iznosi oko 2000 pravokutnika. Sve regije se postavljaju na željenu veličinu i šalju na ulaz u konvolucijsku neuronsku mrežu. Izlaz iz mreže se šalje na SVM model koji razdvaja objekt od pozadine te se na kraju radi regresija veličine pravokutnika regije [10]. Rad RCNN-a napisan na engleskom jeziku je prikazan na slici 16:



Slika 16: RCNN model za detekciju objekata [10]. Prvo se generiraju regije interesa koje se postavljaju na veličinu ulaza u konvolucijsku mrežu. Te regije interesa tada se šalju na ulaz konvolucijske mreže te se izlaz iz mreže šalje na SVM modele koji određuju da li je regija interesa traženi objekt i na regresiju veličine pravokutnika koja radi fino ugađanje veličine te regije.

Trening RCNN modela izvodi se u pet koraka [10]. U prvom koraku se uzme neka konvolucijska mreža trenirana na setu podatak ImageNet. Drugi korak jest ponovno treniranje zadnjeg potpuno povezanog sloja sa slikama objekata koji moraju biti detektirani te sa slikama koje ne sadrže te objekte. Treći korak jest uzimanje svih regija za svaku sliku,

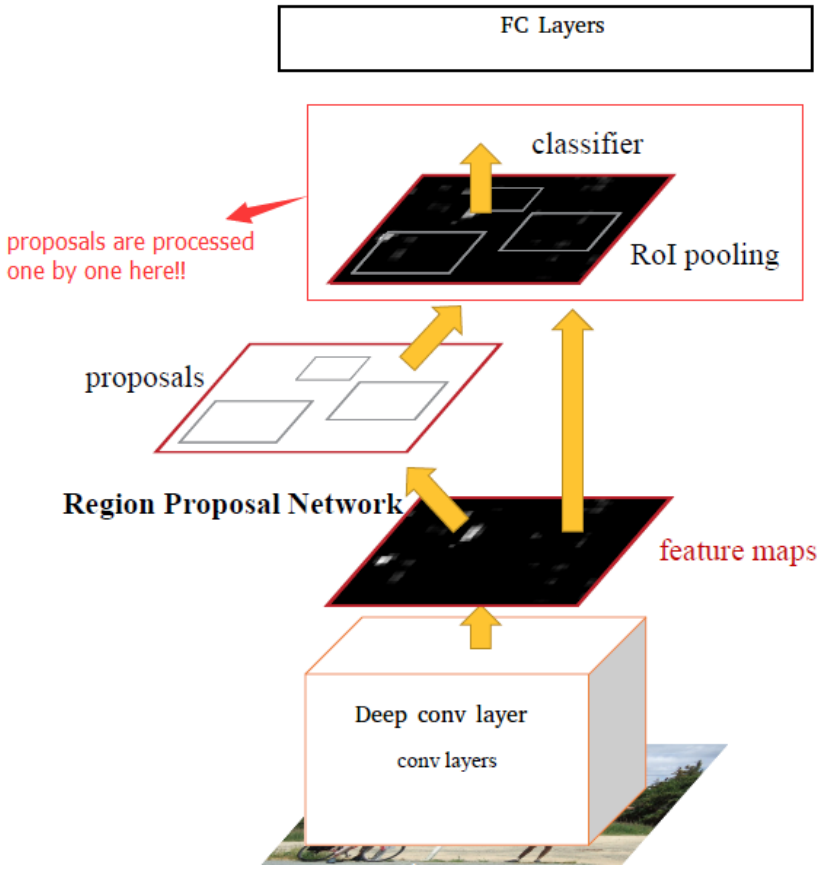
podešavanje veličine za svakog od njih te spremanje na disk. U četvrtom koraku se uz pomoć njih uče binarni SVM-ovi koji prepoznavaju objekt ili okolinu za svaki pojedinačni razred. U zadnjem koraku se uče model linearne regresije koji daje korekciju dobivenih pravokutnika. RCNN radi svih tih koraka daje dobre rezultate, ali ima nedostatak da je radi svih spomenutih koraka jako spor. Model koji pokušava ublažiti taj problem jest **Fast RCNN** [10]. Ta metoda isto ovisi o vanjskom predlaganju regija interesa, npr. preko selektivne pretrage. Predložene regije se zajedno sa cijelom slikom šalju na ulaz konvolucijske mreže te se izlaz iz mreže zajedno sa predloženim regijama šalje na sloj sažimanja regije interesa (*eng. ROI pooling layer*). Sloj sažimanja regije interesa daje izlaz za svaku regiju interesa te ih sve postavlja na istu veličinu zbog ulaza na potpuno povezani sloj mreže. Potpuno povezani sloj tada daje klasifikaciju preko vjerojatnosti da li je to traženi objekt te nakon toga radi regresija veličine pravokutnika. Rad modela Fast RCNN napisan na engleskom jeziku je prikazan na slici 17:



Slika 17: Fast RCNN model za detekciju objekata [10]. Na ulaz mreže šalje se cijela slika sa regijama interesa te se izlaz iz nje šalje na sloj sažimanja regije interesa. Izlaz iz njega je vektor koji se šalje na potpuno povezane slojeve za klasifikaciju i regresiju veličine pravokutnika.

Radi manjeg broja koraka ovaj model radi osjetno brže od modela RCNN. U praksi je taj model obično brži oko 25 puta od modela RCNN sa podjednakim performansama detekcije. Iako je taj model osjetno brži i dalje nije dovoljno brz jer mu i dalje u prosjeku treba preko 2 sekunde po slici veličine 500×500 piksela [10]. Nadogradnja na model **Faster RCNN** rješava taj problem jer se tada brzina izvođenja spušta na oko 0.2 sekunde za sliku iste veličine uz također jednake performanse detekcije. Taj model napušta vanjski model predlaganja regija interesa na slici prije ulaza u mrežu te ga premješta na izlaz konvolucijske mreže. U njemu se na ulaz mreže šalje slika sa objektom te se uzima izlaz zadnjeg konvolucijskog sloja mreže koja je naučena na setu ImageNet i šalje na mrežu koja predlaže

regije. Tamo se pomični prozor (najčešće veličine 3×3 piksela) pomiče po slici te provjerava da li su pikseli unutar njega traženi objekt. Također se za svaki prozor isprobavaju prije definirani pravokutnici fiksnih veličina te se klasificiraju kao objekt ili pozadina. Najbolji prijedlozi se na kraju šalju dalje u mrežu na klasifikaciju te regresiju veličine pravokutnika. Način rada Faster RCNN modela napisan na engleskom jeziku je prikazan na slici 18:

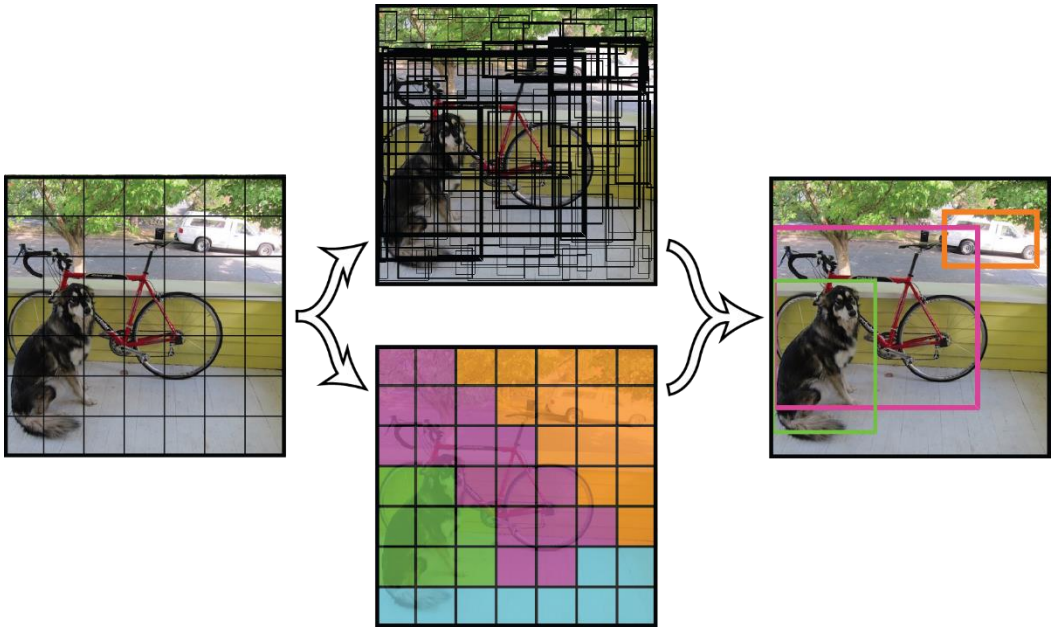


Slika 18: Faster RCNN model za detekciju objekata [10]. U njemu se sama slika šalje u potpuno konvolucijsku mrežu te se na izlazu radi predlaganje regije interesa preko pomičnog prozora sa različitim veličinama pravokutnika. Najbolji prijedlozi se tada šalju u mrežu za klasifikaciju te regresiju veličine pravokutnika.

Prilikom učenja ovog modela bitno je zapamtiti da ovaj model radi načina rada ima 4 funkcije gubitka: funkciju gubitka predlaganja regije interesa, gubitak klasifikacije prilikom predlaganja regije interesa, Fast RCNN gubitak klasifikacije te Fast RCNN gubitak regresije veličine pravokutnika.

3.3. Single Shot detektori (SSD i YOLO)

Sve do sad navedene metode se oslanjaju na vanjske predlagatelje područja interesa na slici te klasifikatore visoke preciznosti za te regije. Radi toga su jako precizni, ali sa problemom sporog izvođenja, što pogotovo vrijedi za modele RCNN i Fast RCNN. Radi toga se te dvije radnje mogu spojiti u jednu mrežu koja radi oboje. To se obično radi tako se definiraju već određene veličine pravokutnika te se uz pomoć njih traže objekti na izlazu iz konvolucijske neuronske mreže [11]. Ti modeli se zovu **Single Shot detektori** i svi oni rade na sličan način sa glavnom razlikom u dobivanju željenog pravokutnika. Oni rade tako da se slika pošalje na ulaz potpuno konvolucijske neuronske mreže te se uzima cijeli izlaz iz te mreže. Taj izlaz je obično radi slojeva sažimanja mnogo manje razlučivosti na izlazu, ali to ne smeta previše. Svaki od tih izlaznih piksela može sadržavati željeni razred te se radi toga on šalje na dodatni sloj koji daje predviđanja za razrede i veličinu pravokutnika. Jedna od metoda za to je **SSD** (***eng. Single Shot Detector***). U njoj se koriste aktivacijske mape različitih veličina za predviđanje razreda i veličine pravokutnika. Druga metoda za to je **YOLO** (***eng. You Look Only Once***). Nakon toga se svi dobiveni pravokutnici filtriraju metodom potiskivanja objekata koji nisu maksimum (*eng. non-maxima suppresion*) sa dobivenom detekcijom na izlaznim klasificiranim pikselima oko objekta. Najbolji dobiveni pravokutnici se uzimaju kao rezultat detekcije. Način dobivanja detekcije na slici metodom YOLO je prikazan na slici 19:



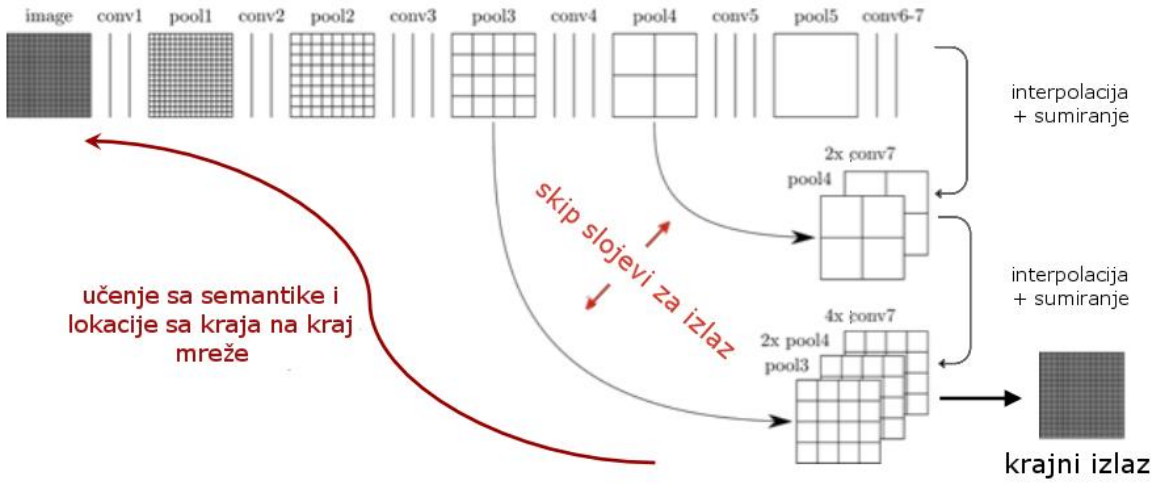
Slika 19: Detekcija objekata metodom YOLO [14] koja uključuje generiranje regija interesa (pravokutnika) i potiskivanje onih koji nisu postavljeni maksimalno dobro u odnosu na klasificirane regije slike.

3.4. Semantička segmentacija preko potpuno konvolucijskih mreža

Potpuno konvolucijske mreže se u zadnjih par godina uspješno koriste za semantičku segmentaciju slika. Osnovni princip rada potpuno konvolucijske mreže za semantičku segmentaciju jest objašnjen u poglavlju 2.4. te će ovdje samo biti proširen detaljima. Princip koji je tamo pokazan radi odlično, ali postoji jedan problem. To je problem krajnje razlučivosti jer se prilikom unaprijednog prolaza kroz mrežu ulaz periodički smanjuje zbog slojeva sažimanja pa krajnja operacija skaliranja nema sve informacije od prije. Taj problem je prikazan na slici 16:

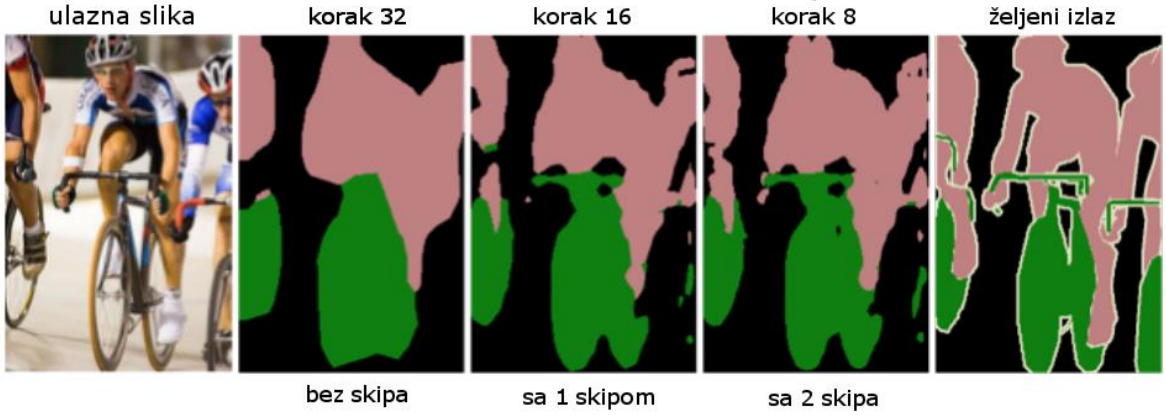
Slika 20: Problem gubitka razlučivosti prilikom semantičke segmentacije koji je vidljiv u nedostatku finih detalja prilikom segmentacije [6]

Za njega postoji dosta rješenja, ali najpopularnije i najjednostavnije je uzeti aktivacije iz prošlih slojeva te ih sumirati i interpolirati zajedno. Ta metoda se naziva *„**skip-connection**“*.Ona uzima izlaze iz konvolucijskih slojeva prije svakog sloja sažimanja te ih na kraju sve spaja uz pomoć sumacije i interpolacije sa izlazom u novi izlaz iz mreže. Svaki od slojeva naduzorkovanja za spajanje na kraju mreže je također moguće učiti. Prikaz arhitekture potpuno konvolucijske mreže sa *„skip-connection“* metodom implementiranom je prikazana na slici 17:



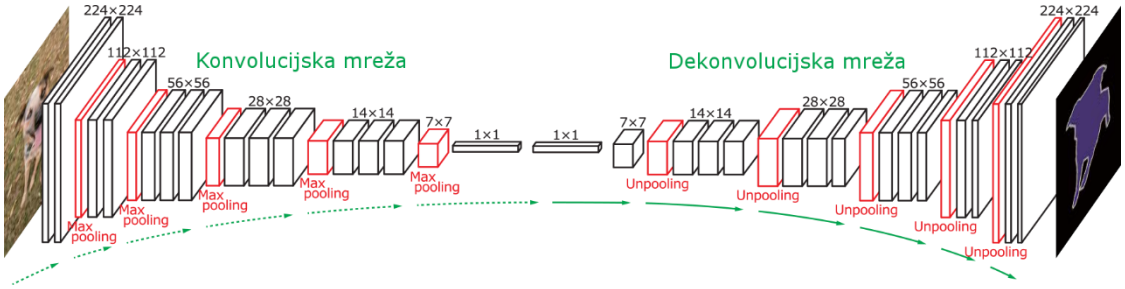
Slika 21: Potpuno konvolucijska mreža sa skip arhitekturom i načinom rada[6]

Krajnji rezultat te metode je često izlaz sa mnogo finije segmentiranim detaljima na slici kao što se to može jasno vidjeti na slici 18:



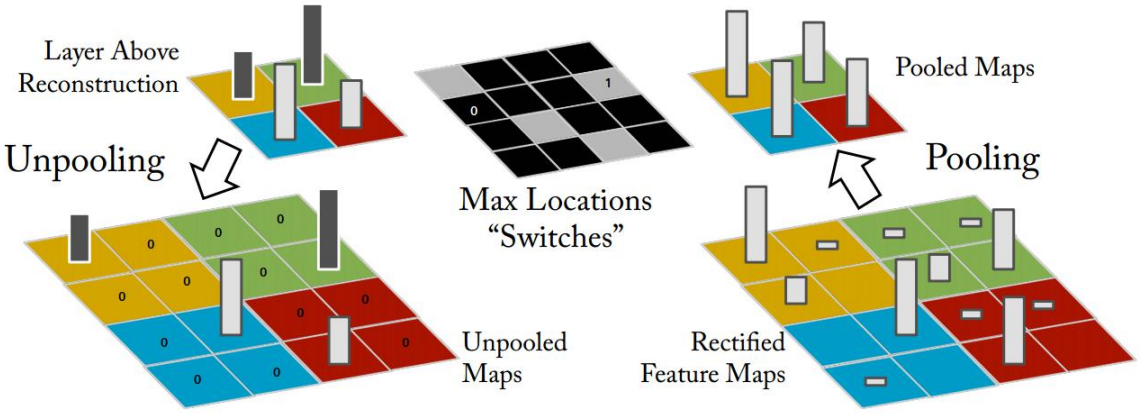
Slika 22: Razlika u razlučivosti između mreža bez skip povezanosti i sa njima [6]. Vidljiv je porast razlučivosti i količine detalja prilikom segmentacije slike ako se koriste skip povezanosti.

Drugi način rješavanja problema razlučivosti jest metoda koja se naziva **ekstremna segmentacija** (***eng. extreme segmentation***) ili **dekonvolucijska mreža** [6]. To je potpuno konvolucijska mreža koja je povezana na drugu potpuno konvolucijsku mrežu koja ima obrnute konvolucijske slojeve te slojeve sažimanja. Arhitektura te mreže jest prikazana na slici 19:



Slika 23: Arhitektura dekonvolucijske mreže [6]. Mreža se sastoji od 2 glavna dijela, konvolucijskog i dekonvolucijskog. Dekonvolucijski sloj također unosi i posebne „unpooling“ slojeve.

Rezultati koje daje ovakva mreža su odlični i pripadaju samom vrhu po performansama segmentacije. Problemi koje unosi ovakva arhitektura su dugotrajno učenje, učenje u 2 koraka gdje se mreža u prvom koraku uči na jednostavnim primjerima te drugi korak sa kompleksnim primjerima i problem uvođenja nove vrste sloja mreže. Taj sloj se naziva **sloj proširivanja** (***eng. unpooling layer***). To je sloj gdje se aproksimira inverzna operacija od sažimanja na način da se prilikom sažimanja spremaju pozicije maksimuma u prozoru sažimanja te se na tu lokaciju stavlja iznos iz prošlog sloja. Način rada sloja proširivanja napisan na engleskom jeziku prikazan je na slici 20:



Slika 24: Sloj proširivanja (eng. unpooling layer) [6]. On radi operaciju obrnutu od sloja sažimanja na način da zapamti lokacije maksimuma na mapama prilikom sažimanja te ih ponovno postavi te maksimume na iste te lokacije prilikom operacije proširivanja.

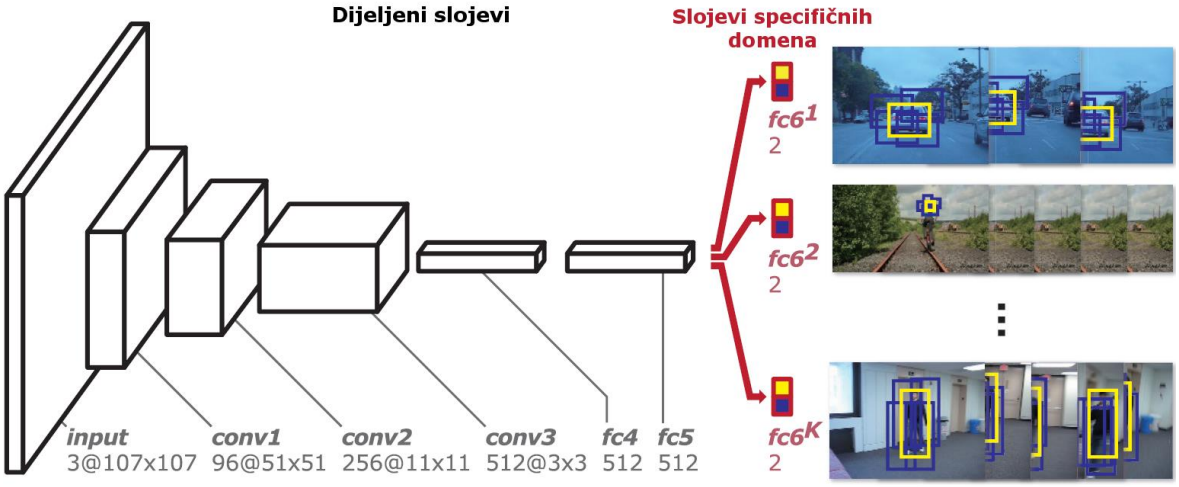
3.5. Definicija problema praćenja objekata

Algoritmi koji se koriste za praćenje kretanja objekata ili same kamere su jedna od najčešće korištenih metoda iz područja računalnog vida. Oni pokušavaju riješiti problem pomaka određenih objekata unutar slike ili pomaka cijele slike u odnosu na prošlu sliku u nizu. Algoritmi klasičnog računalnog vida koji se bave tim problemom najčešće se mogu podijeliti u par kategorija: algoritmi za gusti optički tok (oni pokušavaju procijeniti vektor kretanja za svaki piksel na slikama), algoritmi za rijetki optički tok (oni pokušavaju pratiti kretanje određenih točaka na slikama), Kalmanov filter (algoritam iz obrade signala koji pokušava naći lokaciju pratećeg objekta iz informacija prijašnjeg kretanja), *Meanshift* i *Camshift* algoritmi (algoritmi za pronalaženje maksimuma funkcije gustoće za praćenje), itd [18]. Samo praćenje objekata može se raditi i algoritmima detekcije što se dosta često i radi i ta metoda se zove praćenje detekcijom (*eng. tracking by detection*), ali najčešće se za tu primjenu koriste specijalno razvijeni algoritmi za praćenje. Osim toga problem detekcije je dosta često (ali ne uvijek) teži problem od praćenja jer u praćenju je najčešće samo bitno u lokalnoj okolini naći objekt koji najsličniji onome iz prošle sličice dok se u detekciji i lokalizaciji pretražuje cijela slika za traženim objektom. Stoga algoritmi za praćenje najčešće u početku rada trebaju rezultat detekcije i lokalizacije nekog drugog algoritma da bi nastavili praćenje između sličica koje slijede. Navedeni algoritmi koji pripadaju klasičnim metodama računalnog vida dan danas daju jako dobre rezultate, ali sa razvojem neuronskih mreža i njihovim odličnim rezultatima na područjima detekcije i lokalizacije dolazi do sve većeg interesa akademske zajednice i industrije da se one iskoriste i za zadatak praćenja objekata. Područje koje se bavi time još uvijek je jako mlado te nema mnogo nekih dogovora kojih se većina drži ni nekih izuzetno dobrih rezultata ili modela na koje bi se nastavila istraživanja.

3.6. Praćenje objekata konvolucijskim neuronskim mrežama

Praćenje objekata konvolucijskim mrežama jedan je od predloženih algoritama za praćenje objekata dubokim modelima. Glavna ideja u tom modelu jest uzeti klasifikacijsku konvolucijsku mrežu koja je naučena na skupu ImageNet te ju prenamijeniti za rad sa različitim razredima objekata koje treba pratiti na videu. U praksi to znači da mreža ima određen broj konvolucijskih slojeva nakon ulaza, par potpuno povezanih slojeva na izlazu iz njih te N izlaza svaki sa softmax funkcijom za svaku specifičnu domenu praćenja tj. broj različitih objekata za praćenje [15]. Softmax funkcija na svakom izlazu određuje razred objekata tj. da li je pravokutnik objekt koji se prati ili pozadina. Takva mreža najčešće je bazirana na VGG modelima jer najnovije generacije (npr. ResNet) klasifikacijskih

konvolucijskih mreža nemaju potpuno povezane slojeve na izlazu. Ovakva mreža obično ima mnogo manju arhitekturu od mreža za detekciju ili klasifikaciju jer je problem osjetno jednostavniji. Arhitektura takve mreže je prikazana na slici 25:



Slika 25: Arhitektura konvolucijske neuronske mreže za praćenje objekata [16]. Na slici se vidi da je izlaz mreže podijeljen na K domena gdje se za svaki video radi klasifikacija pravokutnika na traženi objekt i okolinu.

Prva stvar koju treba napraviti prilikom izgradnje ovakve mreže jest uzeti naučene konvolucijske filtere i naučiti regresijski model za veličinu pravokutnika te nakon toga naučiti potpuno povezane slojeve 4, 5 i 6. sloj za tu domenu uz pomoć pozitivnih i negativnih primjera. To se obično radi na prvih par sličica iz videa. Nakon toga se uzme početna lokacija objekta koji se prati bilo ručno ili preko nekog algoritma za lokalizaciju. Idući korak jest generiranje kandidata na novu lokaciju objekta preko Gaussove razdiobe sa centrom u prošloj lokaciji objekta. Također se radi i skaliranje veličine tih pravokutnika. Idući korak je pronalazak najboljeg kandidata za novu lokaciju. To se radi preko formule (3.1):

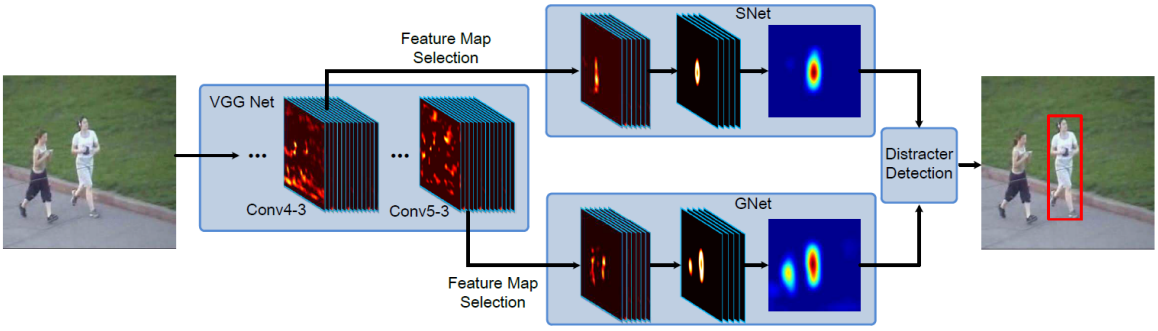
$$x^* = \underset{x^i}{\operatorname{argmax}} f^+(x^i) \tag{3.1}$$

gdje je x^* nova lokacija objekta, x^i predložene nove lokacije, a f^+ funkcija iznosa kvalitete nove lokacije (izlaz iz konvolucijske neuronske mreže). Kad se dobije najbolji kandidat, onda se provjerava iznos iz mreže. Ako je taj iznos veći od nekog definiranog praga (najčešće je to iznos od 0.5) onda se generiraju i spremaju pozitivni primjeri iz tog te se radi regresija veličine pravokutnika. Ako je iznos manji od praga onda se radi učenje sa pozitivnim primjerima dobivenim od prije te negativnim iz te iteracije. Također se svakih 10 sličica radi učenje mreže sa pozitivnim i negativnim primjerima [16]. Sama funkcija f^+ se uči sa 100 K iteracija, gdje je K broj različitih domena (videa) sa brzinom učenja od 0.0001 za

konvolucijske slojeve i 0.001 za potpuno povezane slojeve. Mreža se također uči na prvoj sličici svakog videa sa 30 iteracija sa istom brzinom učenja. Sama metoda učenja je stohastički gradijentni spust. Ovakva mreža za praćenje daje dobre rezultate, ali ima problem jako sporog izvođenja (1 sličica u sekundi na videu razlučivosti 111×111 piksela i to sve sa izvođenjem na grafičkoj kartici NVIDIA Tesla K20m). Jedan od glavnih razloga za to je mnogo provjera različitih veličina pravokutnika za svaku sličicu, što znači prolazak kroz cijelu ne baš malu mrežu, te radi samog učenja mreže tijekom rada.

3.7. Praćenje objekata potpuno konvolucijskim neuronskim mrežama

Drugi način praćenja objekata jest preko potpuno konvolucijskih neuronskih mreža. U tom slučaju se zadnji potpuno povezani slojevi miču i gledaju se samo odzivi zadnjih konvolucijskih slojeva. Arhitektura rada jednog takvog modela napisan na engleskom jeziku jest prikazan na slici 26:



Slika 26: Arhitektura potpuno konvolucijske neuronske mreže za praćenje objekata [15]. Vidi se da je mreža podjeljena na 2 dijela: SNet i GNet. SNet mreža pokušava pronaći specifične detalje objekta i maknuti šum u izlazu iz konvolucijskog sloja za objekt koji se prati dok GNet radi lokalizaciju.

Mreže koje koriste takvu ili sličnu arhitekturu najčešće koriste već naučene težine modela koji su naučeni na setu ImageNet. U primjeru na slici 26 to jest mreža VGG16 [15]. Nakon što se odabere mreža, prvi korak u radu jest detekcija početne lokacije objekta za praćenje. To se može raditi ručno (da sami označimo lokaciju objekta) ili preko nekog detektora (npr. SSD). Nakon toga se inicijaliziraju početne vrijednosti specifične mreže (SNet) i opće mreže (GNet). To se radi prema sljedećim funkcijama gubitka (3.2):

$$L = L_S + L_G \tag{3.2}$$

$$L_U = \left\| \widehat{M}_U - M \right\|_F^2 + \beta \| W_U \|_F^2$$

gdje je $U \in \{S, G\}$ tj. označava mreže SNet i GNet, $\widehat{M}_U$ predstavlja mapu odziva mreže za sve što nije pozadina, M je tražena mapa odziva, W_U su parametri težina konvolucijskih slojeva, a β je parametar za smanjivanje intenziteta težina. Nakon inicijalizacije idući korak jest izrezivanje pravokutnika oko zadnje poznate lokacije objekta i njegovo slanje kroz konvolucijsku mrežu. Lokalizacija objekta se radi na mreži GNet. Ona pretpostavlja da se objekt pomaknuo po normalnoj razdiobi prema izrazu (3.3):

$$p\left(X^t \middle| \hat{X}^{t-1}\right) = N(X^t; X^{t-1}, \sum) \tag{3.3}$$

gdje je $\sum$ dijagonalna kovarijacijska matrica koja sadrži varijance lokalizacijskih parametara. Taj izraz se kombinira sa svim odzivima GNet mreže te se kandidat sa najvećom vjerojatnošću uzima kao predviđena lokacija objekta u trenutnoj slici. Dobiveni izlaz se onda kombinira sa izlazom SNet mreže koja pokušava maknuti šum sa odziva GNet mreže te se dobiva konačna lokacija traženog objekta u trenutnoj slici u videu. Nakon toga se mreže GNet i SNet mogu prema potrebi osvježiti u radu sa novim podacima radi prilagodbe novim uvjetima praćenja objekata. Ova metoda je osjetno brža od metode praćenja sa običnom konvolucijskom mrežom pa da se izvodi u 3 sličice u sekundi na grafičkoj kartici NVIDIA GeForce TITAN sa usporedivim performansama praćenja objekata. Iako je to osjetno poboljšanje to i dalje nije dovoljno dobro za praćenje u realnom vremenu.

4. Praćenje objekata na sportskim susretima konvolucijskim modelima

4.1. Definicija problema

Problem kojim se bavi ovaj rad jest praćenje igrača na sportskim susretima uz opcionalnu detekciju koristeći konvolucijske neuronske mreže. U tu svrhu korištena je video snimka nogometne utakmice između klubova Dinamo i Split. Video snimka je snimana iz ptičje perspektive sa pogledom na cijeli teren. Razlučivost videa je 1632×288 piksela, trajanje mu je 5 minuta i sniman je u 25 sličica u sekundi. Isječak iz testnog videa je prikazan na slici 27:

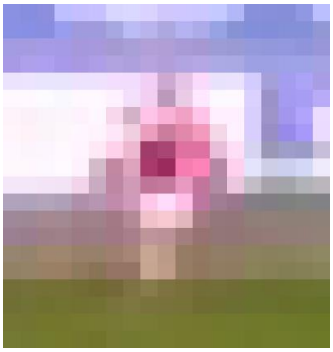
Slika 27: Primjer isječka iz testnog videa.

Prvi problem koji se javlja jest veličina igrača u odnosu na teren. To je problem koji podjednako pogađa detekciju i praćenje objekata. Sitni objekti na slici se često teže detektiraju nego objekt koji je jasno definiran na slici te stoga to treba uračunati u obzir. Idući problem koji se javlja jest promjena veličine igrača ovisno o tome na kojem dijelu terena se nalazi. Primjer tog problema jest prikazan na slici 28:

Slika 28: Problem različite veličine igrača na terenu. Taj problem se javlja prilikom različite udaljenosti od kamere.

Taj problem također podjednako pogađa detekciju i praćenje. U slučaju detekcije taj problem otežava učenje mreže jer model nije konzistentan, ali to se danas rješava kombinacijom konvolucijskih i slojeva sažimanja. Kod praćenja taj problem će se probati riješiti na drugi

način. Treći problem koji se javlja jest problem brzog kretanja igrača. To se manifestira velikim pomakom objekta između dva sličice u videu. Taj problem jest bitan samo za praćenje igrača dok za detekciju ne igra nikakvu ulogu. Najčešće se rješava povećanjem veličine susjedstva gdje se gleda postoji li pomak objekta. Četvrti problem koji se javlja jest problem stapanja igrača sa pozadinom. Primjer tog problema jest prikazan na slici 29:



Slika 29: Problem stapanja igrača sa pozadinom. Na slici je vidljivo da se igrač jedva prepoznaje u odnosu na pozadinu.

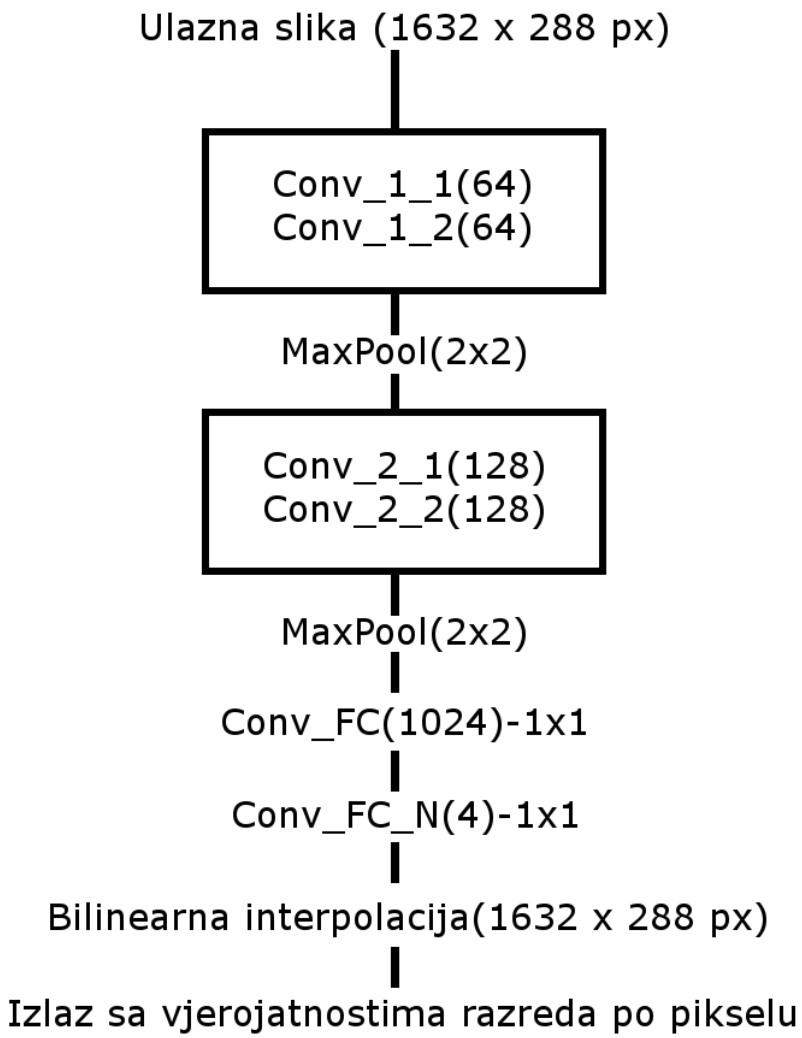
U tom slučaju teško je razlikovati objekt od pozadine i to podjednako pogađa detekciju i praćenje igrača. Taj problem nije jednostavno riješiti i najčešće je najbolje rješenje probati naučiti mrežu da radi dobro u mnogo različitih uvjeta. Zadnji i najteži problem jest problem međusobnog prekrivanja igrača. To je situacija kada se jedan igrač nađu ispred drugog te svojim tijelom potpuno ili većinom prekrije igrača koji se nalazi iza njega. Taj problem jest prikazan na slici 30:

Slika 30: Problem međusobnog prekrivanja igrača. Na slici je vidljivo da je crveni igrač gotovo u potpunosti ispred drugog igrača te time miče gotovo sve informacije o njemu sa trenutne slike.

Ovaj problem je jako nezgodan za rješavanje jer se prilikom prekrivanja u gotovo potpunosti gubi informacija o jednom igraču te je nemoguće napraviti predviđanje iz položaja ili npr. dotadašnjeg vektora brzine igrača. Taj problem će se također probati riješiti u ovom radu. Zadnji problem koji se javlja jest prilikom učenja modela zbog toga jer su oznake s kojima se uči model dosta neprecizne te stoga otežavaju učenje. S druge strane ta nepreciznost daje određeni regularizacijski efekt pa u nekim slučajevima to možda čak i pomaže.

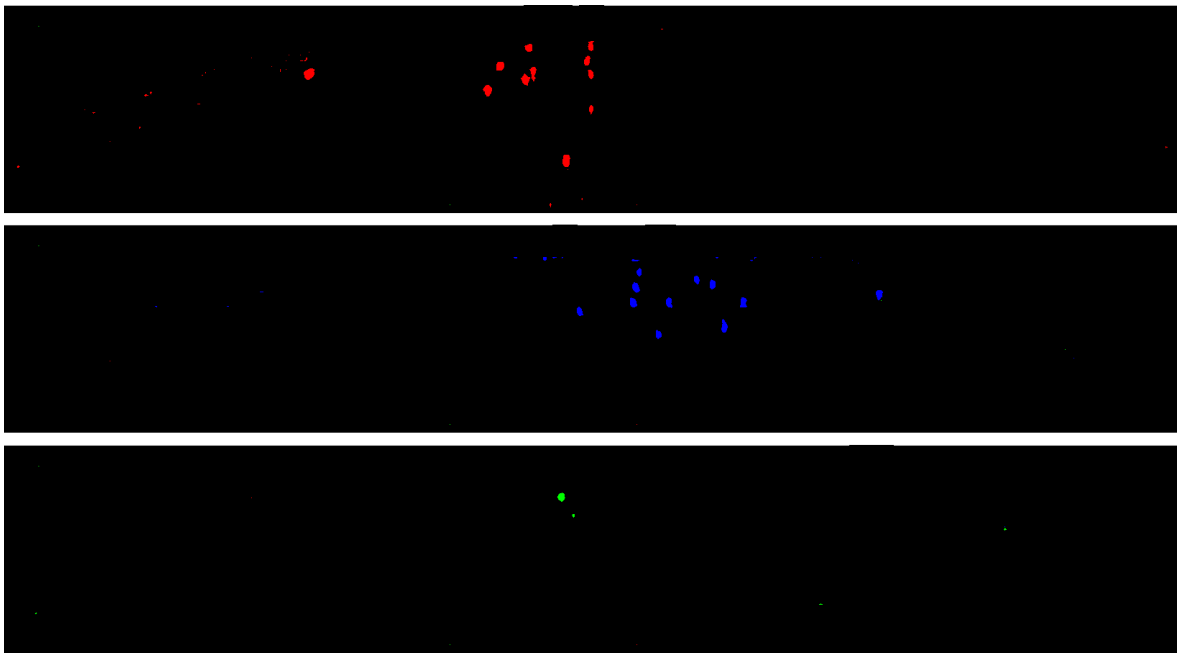
4.2. Detekcija igrača na terenu konvolucijskom neuronskom mrežom

Problem detekcije igrača na terenu je riješen konvolucijskom neuronskom mrežom. Mreža je potpuno konvolucijskog tipa i izvodi semantičku segmentaciju. Razlog za takav odabir jest taj što se segmentacijom igrača na terenu oni lako razaznaju od pozadine te ih je lako grupirati. Mreža je konstruirana za prepoznavanje 4 razreda objekata: nogometni tim A, nogometni tim B, sudac u terenu i razred ostalo (pozadina). Broj razreda se može promijeniti, ali bi tad bilo preporučljivo i promijeniti arhitekturu mreže radi dobivanja optimalnih performansi. Arhitektura korištene mreže je prikazana na slici 31:



Slika 31: Arhitektura konvolucijske neuronske mreže za detekciju igrača.

Kao što je vidljivo iz slike mreža nema veliku dubinu za što postoji par razloga. Prvi od njih taj što su igrači obično dobro definirani na terenu te stoga nije potreban veliki kapacitet mreže da nauči gdje su igrači. Drugi razlog male dubine jest brzina izvođenja, jer kraća mreža u pravilu ima kraće vrijeme izvođenja na računalu. Treći razlog jest taj što su igrači dosta sitni na terenu pa stoga nije potrebno da mreža ima veliko receptivno polje i mnogo parametara za učenje. Ova mreža također nema implementirane „*skip-connection*“ slojeve niti dekonvolucijske slojeve. Razlog za takvu odluku jest nebitnost segmentacije sitnih detalja na igraču, tj. bitno je samo naći njegovu poziciju, a detalji nisu toliko bitni. Stoga mreža može biti bez takvih dodatnih slojeva što također ubrzava vrijeme izvođenja. Mreža također koristi već dobivene težine iz VGG16 modela u prva dva konvolucijska sloja koji se samo po potrebi fino ugode sa učenjem modela. Razlog za takav odabir jest taj što VGG16 mreža ima odlične rezultate na ImageNet setu podataka te nije potrebno mnogo dodatnog učenja da model dobro radi. Izlaz iz mreže jest vjerojatnost pripadanja razredu pojedinog piksela. Idući korak jest dobivanje tri pojedinačne slike gdje su detektirane najveće vjerojatnosti za svaki pojedini razred kao što je prikazano na slici 32:



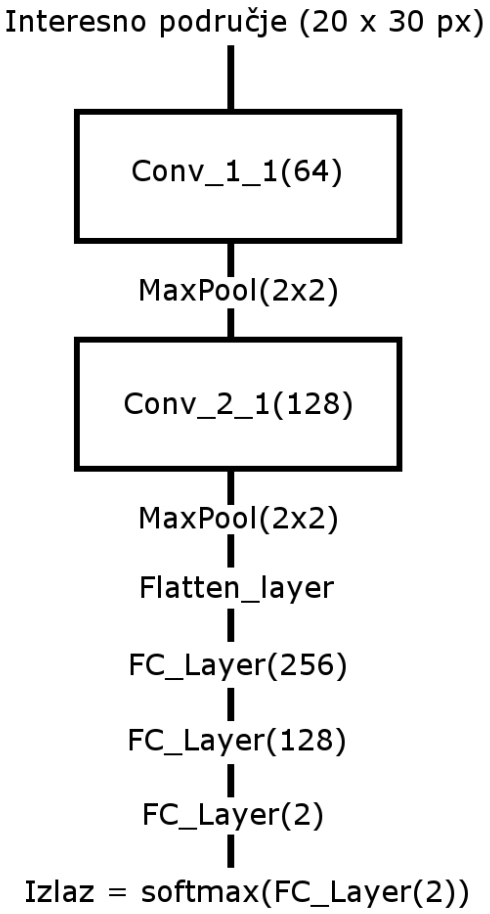
Slika 32: Dobivene slike za grupiranje. Prva slika odozgo prikazuje detektirane piksele za igrače prvog tima, slika u sredini za igrače drugog tima, a zadnja slika za sudca.

U idućem koraku se pokreće KNN algoritam grupiranja za svaku pojedinačnu sliku. Način dobivanja lokacije pojedinačnog igrača jest dobivanje N centroida, gdje je N hiperparametar koji je odabran radi moduliranja očekivanog broja objekata na slici, te filtriranje samo onih sa najvećim lokalnim susjedstvom. Na taj način se dobiju lokacije pojedinačnih igrača te se

oko njih rade pravokutnici proizvoljne veličine koji tada mogu poslužiti za daljnje praćenje ili ponovnu detekciju. KNN algoritam grupiranja jest odabran radi dobrih performansi izvođenja i radi odličnog grupiranja na ovom tipu problema. Kvaliteta detekcije na ovaj način podosta ovisi o kvaliteti segmentacije slike te je stoga preporučljivo napraviti kvalitetno učenje mreže. Jedan od preporučenih načina poboljšanja detekcije bio bi također prelazak na SSD ili YOLO modele detektora koji bi mogli dati bolje rezultate. Taj način detekcije je bio u planu za implementaciju, ali se na kraju od njega odustalo radi kombinacije kašnjenja samih podataka i početnih loših rezultata SSD detektora na njima. Mreža se uči na ručno označenim objektima na terenu koji su označeni u formatiranoj datoteci. Ti podaci imaju dosta šuma te stoga učenje nije idealno, ali s tim se ujedno unosi određena regularizacija u učenje. Primjeri za učenje se stoga dobiveni na način da se uzmu pravokutnici za svakog igrača na terenu, nađe se središte pravokutnika te se nacrta elipsa oko tog središta sa dužom poluosi u ravnini igrača. Svaki razred objekta ima drugačije označenu elipsu (različit intenzitet piskela) na crnoj podlozi (razredu pozadine) tako da se tako označene slike oznake mogu tretirati kao podaci za učenje pomoću kojih je moguće učiti mrežu da radi semantičku segmentaciju.

4.3. Praćenje igrača na terenu konvolucijskom neuronskom mrežom

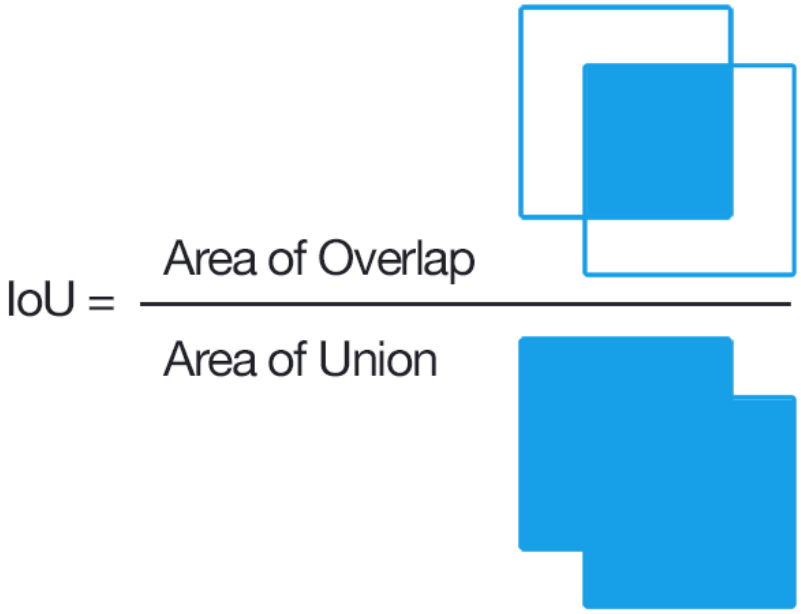
Problem praćenja igrača na terenu također je riješen konvolucijskom neuronskom mrežom, ali u ovom slučaju se radi o običnom, a ne potpuno konvolucijskom modelu. Pošto kvalitetnih radova na ovu temu nema puno u ovom radu se odlučilo raditi sa originalno osmišljenom metodom praćenja objekata uz pomoć konvolucijskog modela. Ideja koju koristi ovaj model praćenja jest pohlepna ponovna detekcija područja interesa u lokalnom susjedstvu prošlog područja interesa uz pomoć konvolucijske neuronske mreže. U praksi to znači da se uzme prošla lokacija područja interesa, u ovom slučaju igrača, te se u okolini prošle lokacije na pohlepan način traži lokacija koja ima najveći odziv na izlazu iz te konvolucijske neuronske mreže. Ova metoda pretpostavlja da se objekt prilikom kretanja ne mijenja previše svoj oblik koji je mreža naučila prilikom faze učenja, što je u praksi skoro uvijek točno. Također pretpostavlja da je objekt negdje u blizini mjesta gdje se nalazio u prošloj slici, iako se to može lako promijeniti sa povećanjem susjedstva pretraživanja. Ta metoda također uz konvolucijsku mrežu koristi i dodatnu logiku kako bi riješila navedene probleme koji se javljaju prilikom praćenja igrača na terenu. Konvolucijska mreža koja se koristi ima arhitekturu prikazanu na slici 33:



Slika 33: Arhitektura konvolucijske neuronske mreže za praćenje.

Iz slike se može vidjeti da ovo mreža ima manje konvolucijskih slojeva od mreže za detekciju, a razlog za to je što je problem praćenja lakši od problema detekcije, a i radi bržeg izvođenja. Ova mreža također kao i mreža za detekciju ima modificirane težine iz VGG16 mreže u konvolucijskim slojevima radi bržeg i lakšeg učenja. Na izlazu se radi softmax funkcija koja daje dva broja od kojih jedan označava slaganje izgleda područja interesa sa onim što je mreža naučila u fazi učenja, a drugi obrnuto. Broj 1 na prvom izlazu označava potpuno slaganje sa traženim objektom, a broj 0 potpuno neslaganje. Razlog za to je način određivanja kvalitete okolnih interesnih područja. To se moglo napraviti i sa jednim izlazom, ali se izlaz sa dva neurona pokazao boljim u praksi. Na ulaz se pak dovodi slika razlučivosti 20×30 piksela. Ta slika je interesno područje na slici koje želimo pratiti. To područje ne mora nužno imati točno tu razlučivost, ali se prije ulaza na mrežu ono mora staviti na tu razlučivost. To se također radi i prilikom učenja mreže. Prilikom samog učenja mreže na ulaz se dovodi pravokutnici područja interesa koji su navedeni u formatiranoj datoteci. Prvo se uzme pravokutnik koji predstavlja točan izgled područja interesa koje želimo pratiti te se

uzmu i pravokutnici iste veličine oko te točke. Iznos točnosti izgleda područja interesa određuje se preko **presjeka unije pravokutnika** (***eng. intersection over union – IoU***). Izraz za računanje presjeka unije pravokutnika na engleskom jeziku dan je na slici 34:

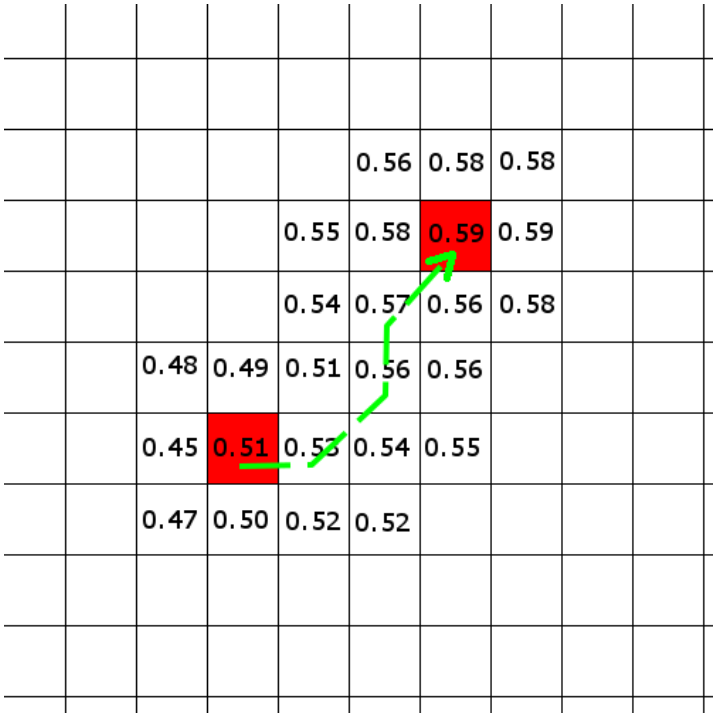


Slika 34: Presjek unije pravokutnika koji je definiran je kao omjer preklapanja pravokutnika i unije pravokutnika [17].

Po tom izrazu se vidi da će pravokutnik koji savršeno priliježe na područje interesa imati izlaz 1, a svaki koji je pomaknut će imati manje. Broj koji se dobije preko tog izraza za određeno područje interesa se šalje kao željeni izlaz prilikom učenja na usporedbu izlaza mreže. Nakon toga mreža računa grešku unakrsne entropije i korigira težine ovisno o greški. Kad se mreža nauči osnovnom izgledu područja interesa onda može početi praćenje objekata. Ovakva mreža za praćenje može raditi za mnogo objekata na videu jer gleda samo lokalne promjene na svakoj idućoj slici. Početne lokacije objekata mogu se dobiti ili ručno ili preko nekog detektora. Prilikom praćenja za svaku novu sličicu u nizu uzima se prošla lokacija područja interesa te se gleda njezina okolina. Veličina okoline je hiperparametar koji je dodan radi ovisnosti veličine objekta u odnosu na razlučivost videa, npr. okolina od 15 piksela je puno veća na razlučivosti 640×480 nego na 3840×2160 piksela. Za svaku lokaciju oko piksela gdje se nalazilo središte pravokutnika u prošloj slici računa se izlaz iz mreže te se nova lokacija pomiče na lokaciju novog piksela prema izrazu (4.1):

$$nova\ lokacija = \underset{lokacija_i}{\text{argmax}}(izlazMreže(lokacija_i)) \tag{4.1}$$

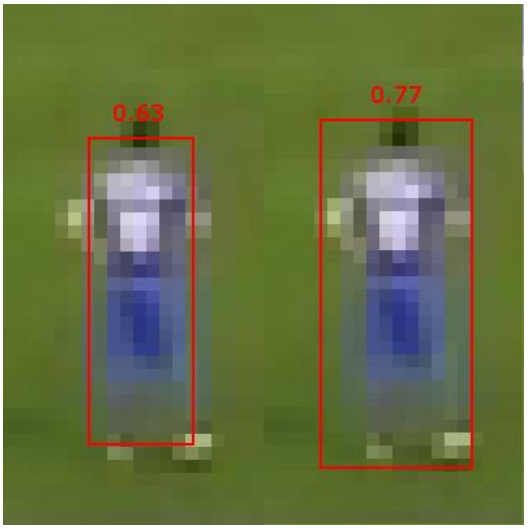
To se radi sve dok se ne dođe u točku gdje više nema rasta iznosa izlaza iz mreže ili dok se ne dođe do krajnje granice veličine susjedstva. Demonstracija takvog načina pretraživanja je prikazana na slici 35:



Slika 35: Način pretraživanja okoline prošle lokacije područja interesa. Kao što je sa slike vidljivo, zbog pohlepnog načina rada metode postoji realna mogućnost zaglavljivanja u lokalnom minimumu tj. da se ne nađe idealan kandidat za postavljanje pravokutnika.

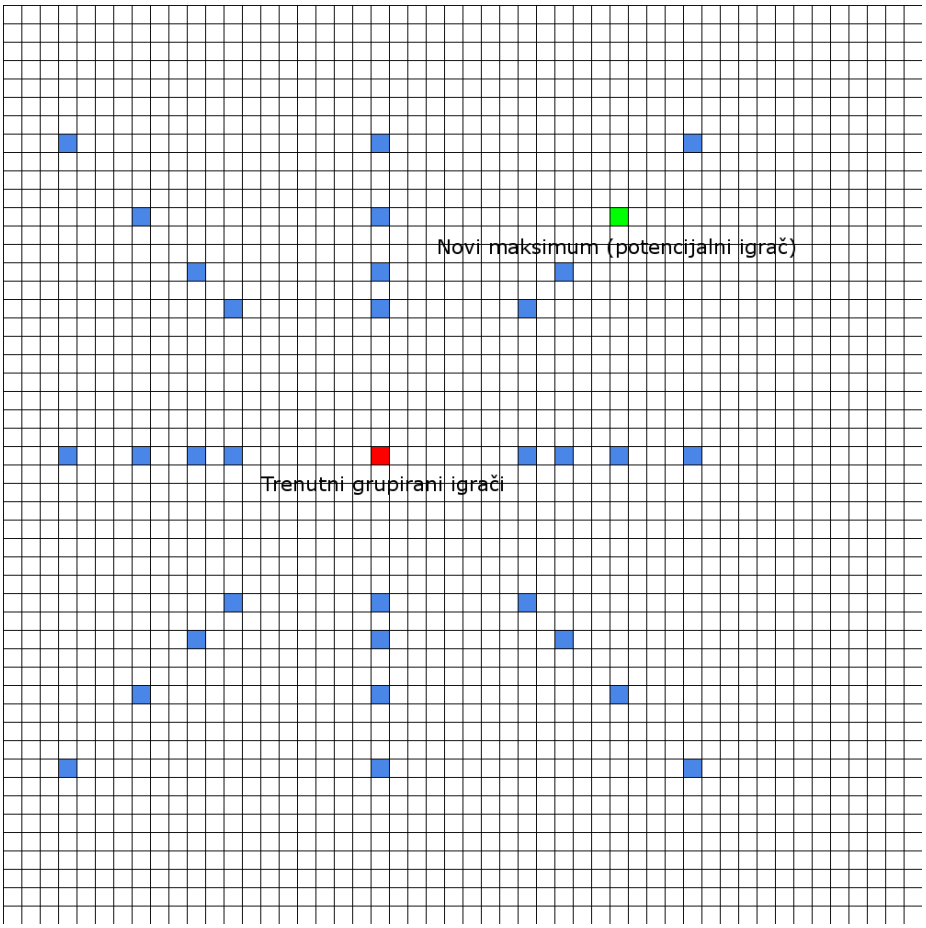
Ovaj način pretraživanja okoline je pohlepan i kreće se samo u smjeru najbržeg rasta, ali u praski to obično odgovara smjeru kretanja objekta. Kad stigne do maksimuma metoda i ako je taj maksimum veći od neke postavljene granice k (također hiperparametar kojim se može podesiti prag za detekciju igrača, što može biti jako korisno ovisno o tipu okoline) to se onda uzima kao nova lokacija područja interesa za iduću sliku. Ukoliko je maksimum manji od hiperparametra k tada se smatra da se objekt izgubio. Tada se može nastaviti praćenje bez njega ili se može ponovno pozvati detektor koji će probati ponovno pronaći objekt (ako je i dalje u sceni). Sa dobrim postupkom učenja mreže i dovoljno velikim susjedstvom oko piksela na ovaj način se rješavaju problemi brzog kretanja igrača i stapanja igrača sa okolinom. Problem različite veličine igrača koji se javlja kada se npr. igrač pomakne na dio terena dalji od kamere rješava se na način da se svakih n (također hiperparametar kojim se može podesiti razlučivost pretraživanja veličina pravokutnika što može biti od koristi ako se veličina objekta mijenja jako brzo ili jako sporo) sličica radi provjera odziva određenog broja većih i manjih pravokutnika koji predstavljaju područje interesa oko objekta koji pratimo.

Ova metoda pretpostavlja da će odziv pravokutnika koji pravilnije obuhvaća igrača dati bolji odziv mreže i time se poboljšati praćenje. Najbolja veličina se računa također preko izraza (15). Prikaz načina te metode je dan na slici 36:



Slika 36: Primjeri odziva različitih veličina pravokutnika na objektima. Pravokutnik koji bolje opisuje položaj igrača najčešće ima i veći iznos izlaza iz mreže za praćenje.

Na ovaj način se može učinkovito pratiti promjena veličine područja interesa (objekta) na terenu. Zadnji problem koji je preostao za riješiti jest problem međusobnog prekrivanja igrača. To je težak problem u kojem se gube svi podaci o igraču koji se nalazi straga te se ne može pretpostaviti njegovo buduće kretanje, pogotovo ako su igrači dugo međusobno prekriveni (kao npr. prilikom udarca iz kuta). Taj problem se u ovoj metoda praćenja pokušava riješiti detekcijom sudara između igrača. Sudar igrača se detektira kada su centri pravokutnika koji predstavljaju pozicije igrača bliže od neke udaljenosti d (najčešće 2-3 piksela, ali ovisi o razlučivosti videa i obliku objekta). Kada se dogodi kolizija dva (ili više) igrača se grupiraju u jednu grupu koja se prati. Nakon toga se periodično, svakih m (također hiperparametar koji je uveden radi pokušavanja razrješenja kolizije, manji broj daje brže performanse, ali tada je veća mogućnost da se objekt ponovno ne detektira i obrnuto) sličica radi pokušaj ponovne detekcije igrača koji se pomaknuo uz pomoć mreže za praćenje. To se radi u malo daljoj okolini grupiranih igrača jer se tako izbjegava situacija da se igrač pronađe jako blizu igrača iza kojeg se nalazio te da detektor ponovno krene pratiti tog igrača. Ponovna detekcija se radi ponovno preko izraza (15) i te se radi usporedba sa proizvoljnim pragom detekcije. Ako je iznos manji od praga, smatra se da je objekt i dalje iza drugog objekta. U suprotnom se objekt miče iz kolizijske grupe i nastavlja se njegovo praćenje. Rad metode ponovne detekcije je prikazan na slici 37:



Slika 37: Rad metode za ponovnu detekciju objekata nakon kolizije. Metoda pretražuje malo dalju okolinu od grupiranih objekata te pokušava pronaći maksimum koji ukazuje na moguće micanje igrača iz grupe. Ova metoda ima nedostatak da ponekad u rijetkim slučajevima pronađe bliskog igrača koji uopće nije ni grupiran.

Nakon izvršavanja ove metode nastavlja se normalno praćenje objekta u videu. Uz ovu navedenu metodu moguće je pratiti objekte na videu proizvoljnih oblika i veličina u poprilično zahtjevnim uvjetima sa dobrim performansama izvođenja, pogotovo ako se metoda izvodi na nekom snažnijem GPU-u.

5. Programska podrška

5.1. Programski jezik Python

Python[6] je programski jezik visoke razine namijenjen za opću namjenu kojeg je stvorio Guido van Rossum 1991.g. To je jezik koji sadrži dinamičko tipiziranje podataka te automatsko upravljanje memorijom. Od programskih paradigmi podržava proceduralnu, objektno orijentiranu, funkcionalnu i imperativnu [12]. Python je jezik koji je interpretiran te stoga nema visoku efikasnost koda. Taj nedostatak Python nadoknađuje sa izuzetno bogatom standardnom bibliotekom, visokom prenosivošću koda između sustava, velikim brojem neovisno razvijenih biblioteka za razne namjene, sintaksom koja omogućuje laku čitljivost koda te ekstenzijskim modulima koji mogu biti pisani u visoko efikasnim jezicima poput C-a ili C++-a. Python dolazi u verzijama Python 2 kojoj vrijeme podrške traje do 2020. godine te Python 3 koja je pod aktivnim razvojem. Zadnja stabilna verzija u vrijeme pisanja ovog rada je Python 3.6.1. Interpreteri za Python su dostupni za sve bitne operacijske sustave poput Windowsa, Linuxa, MacOS-a, BSD-a te još mnogo manje poznatih operacijskih sustava.

5.2. Programska biblioteka TensorFlow

TensorFlow[7] je programska biblioteka otvorenog koda namijenjena za numeričko računanje uz pomoć grafova toka podataka [13]. Razvijena je u Google-ovom razvojnom timu „Google's Machine Intelligence“ za potrebe strojnog učenja i istraživanja dubokih modela neuronskih mreža. U njemu čvorovi grafa predstavljaju matematičke operacije dok rubovi grafa predstavljaju višedimenzionalna polja podataka (tenzore) koji komuniciraju između njih. Samim načinom rada se vidi da je primarno namijenjen za učenje i korištenje dubokih neuronskih mreža svih oblika i vrsta te zbog toga ima podržane mnoge često korištene algoritme učenja, definiranja i spremanja neuronskih mreža. TensorFlow ima fleksibilnu arhitekturu koja omogućava računanje na jednom ili više računalnih procesora ili grafičkih kartica na računalu, serveru ili na mobilnom uređaju. TensorFlow je podržan na operacijskim sustavima Linux, Windows te MacOS. Podržava rad sa grafičkim karticama koje podržavaju tehnologiju CUDA sa mogućom podrškom za OpenCL u budućnosti. TensorFlow je primarno namijenjen za rad sa programskim jezikom Python jer za njega

[6] https://www.python.org
[7] https://www.tensorflow.org

postoji najdetaljnija dokumentacija, ali podržani su i jezici C (C++), Java i Go. U trenutku pisanja ovog rada zadnja verzija TensorFlow biblioteke je 1.2rc0.

5.3. Programski okvir Qt

Qt[8] je moćan aplikacijski programski okvir koji je namijenjen za lako prenosiv kod i rad na više platformi. Napisan je u jeziku C++ i podržava sve popularne operacijske sustave i mobilne sustave. Može se koristiti za pisanje cijelih aplikacija, ali najčešće se upotrebljava za izradu korisničkih grafičkih sučelja (GUI-a). Podržani su svi popularni programski jezici kao npr. Java, C++, Python, itd. Zadnja stabilna verzija Qt programskog okvira za vrijeme pisanja ovog testa je 5.9. Primjer aplikacije napisane sa programskim okvirom Qt za više platformi je prikazan na slici 27:

Slika 38: Višeplatformska aplikacija napisana uz pomoć programskog okvira Qt[9].

5.4. Programska biblioteka OpenCV

OpenCV[10] je programska biblioteka sa funkcijama računalnog vida koja je namjenjena za rad u realnom vremenu. Izvorni autor je Intel i danas je to višeplatformska biblioteka otvorenog koda. Ona u sebi sadrži implementirane mnoge algoritme računalnog vida, rad sa videom i slikama te neke modele strojnog učenja. Napisana je u jeziku C++ i to je primarni

[8] https://www.qt.io
[9] https://www.qt.io/ui/
[10] http://opencv.org

podržani jezik. Osim njega službeno su podržani i jezici Java, Python i MATLAB/OCTAVE. Zadnja verzija programske biblioteke OpenCV u vrijeme pisanja ovog rada je 3.2.0.

5.5. Programska biblioteka Scikit-learn

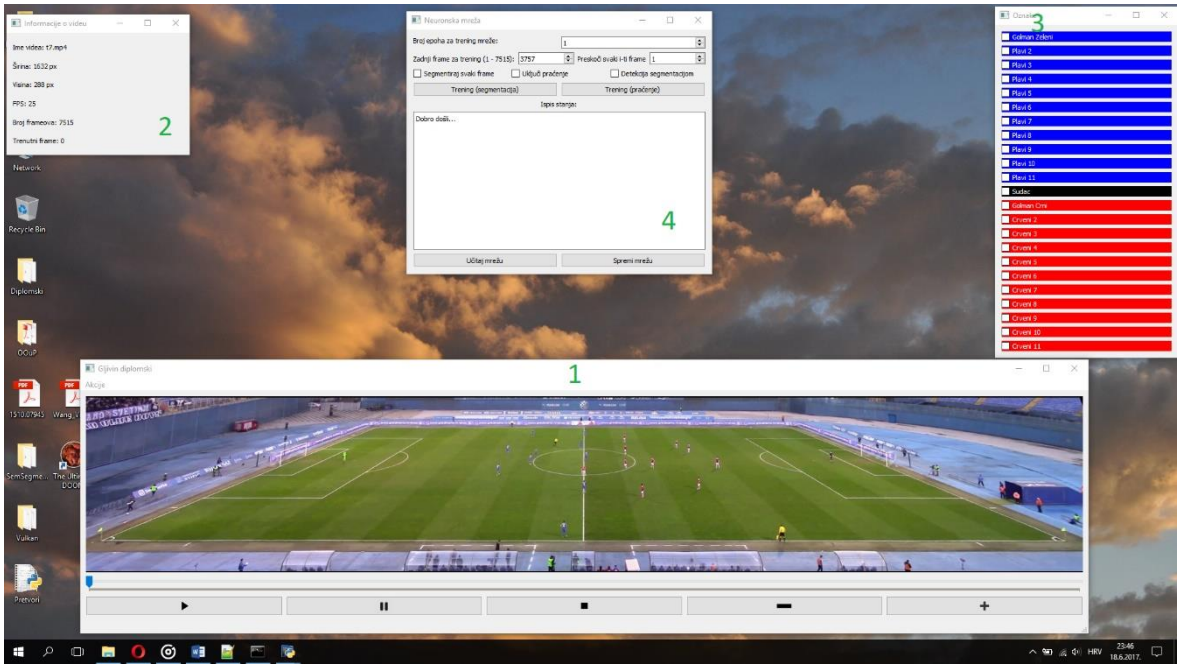
Scikit-learn[11] je besplatna programska biblioteka strojnog učenja namjenjena za jezik Python. U njoj se nalaze razni algoritmi strojnog učenja poput SVM-a, slučajnih šuma, k-means, logističke regresije, DBScana i slično. Ta biblioteka je implementirana za rad sa drugim Pythonovim znanstvenim bibliotekama poput NumPy-a i SciPy-a. Zadnja stabilna verzija Sckit_learn-a u vrijeme pisanja ovog rada je 0.18.1.

[11] http://scikit-learn.org/stable/

6. Programska implementacija

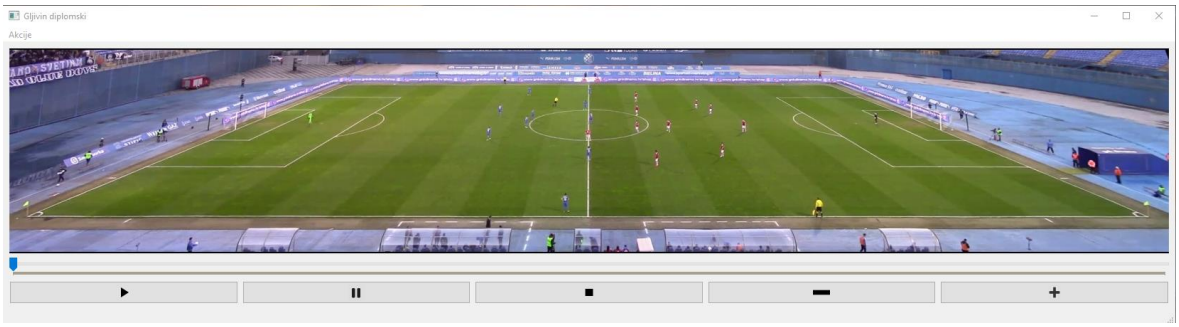
6.1. Glavni prozori

Glavni program je implementiran u jeziku Python verzije 3.5, uz pomoć biblioteke OpenCV 3.2 i programskog okvira Qt 5.9. Biblioteka OpenCV je korištena radi jednostavnog manipuliranja sa slikama i videom dok je uz pomoć programskog okvira Qt napravljeno grafičko sučelje za rad s programom. Program je bez ikakvih modifikacija isproban i testiran na operacijskim sustavima Windows 10 i Linux Mint 18.1. Glavni program se sastoji od 4 prozora prikazanih na slici 39:



Slika 39: Glavni program. Na slici je vidljivo da se on sastoji od 4 prozora sa različitim funkcionalnostima.

Glavni prozor je prozor broj 1. On se otvara prilikom pokretanja programa. Funkcionalnosti koje on posjeduje su otvaranje video zapisa u različitim formatima, osnovne funkcije za rad sa videom kao što su puštanje videa, pauza, prekidanje videa i pomak za jednu sliku unaprijed i unatrag. Također podržava mnogobrojne tipkovničke kratice za sve te funkcije. Zadnja mogućnost koja je implementirana jest mogućnost otvaranja formatiranih zapisa kretanja pojedinačnih igrača na terenu. Preko ovog prozora se prilikom otvaranja obje vrste datoteka dolazi do preostala tri prozora. Kod koji čini rad prozora jest poprilično dugačak i većinom se bavi definiranjem sučelja i načinom baratanja događaja (klikova mišem i tipkovnicom) bez zanimljivih algoritama te ga stoga nema smisla navoditi ovdje. Prozor 1 je prikazan na slici 40:

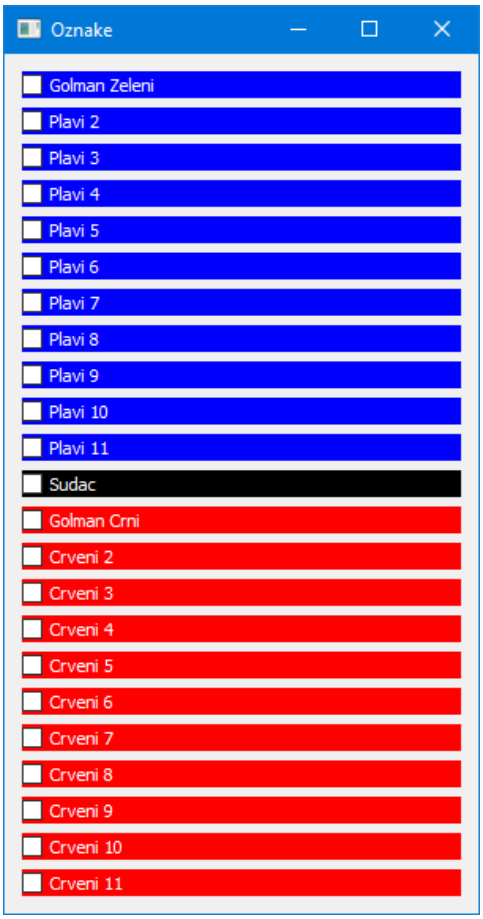


Slika 40: Prozor 1. Na slici su vidljive opcije rada sa video datotekom.

Drugi prozor se prikazuje prilikom otvaranja datoteke sa videom i on sadrži neke podatke o tom otvorenom videu. Ti podaci su: ime datoteke, razlučivost videa, broj sličica u sekundi, ukupan broj sličica u sekundi te redni broj trenutne sličice na prvom prozoru. Redni broj trenutne sličice se osvježava prilikom pomicanja kroz video. Kod za drugi prozor sadrži manje-više samo dodavanje elemenata grafičkog sučelja tako da se ovdje ne navodi. Prozor broj 2 je prikazan na slici 41:

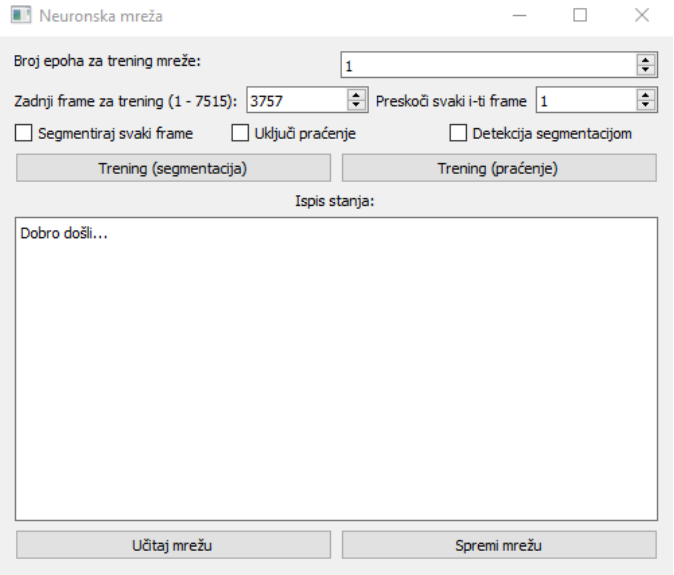
Slika 41: Prozor 2. Vide se informacije o razlučivosti videa, duljini videa, broju sličica u sekundi te trenutna sličica.

Treći prozor se prikazuje prilikom otvaranja formatirane datoteke sa pozicijama igrača. On služi za prikazivanje pozicija svakog igrača na terenu. To se radi klikom na potvrdni okvir sa imenom razreda koji se želi označiti. Time se pojavljuje zeleni pravokutnik oko označenog objekta. Kod programa se nalazi na priloženom mediju uz rad. Prozor broj 3 je prikazan na slici 42:



Slika 42: Prozor 3. Boje potvrdnih okvira označavaju pripadnosti razredima objekata.

Četvrti prozor se također prikazuje prilikom otvaranja formatirane datoteke sa pozicijama igrača. Taj prozor se koristi za većinu rada sa mrežama za detekciju i praćenje. Prikazan je na slici 43:



Slika 43: Prozor 4. Na slici se vidi da on sadrži grupirane sve opcije za rad sa programom.

Tu se može namjestiti broj epoha za učenje mreže. Također se može podesiti na koliko će se prvih slika u videu koristiti za učenje te da li će se preskakati svaki n-ti video. Učenje mreže za detekciju se pokreće klikom na gumb „*Trening (segmentacija)*“. Prilikom učenja greške se ispisuju na konzoli, a kraj treninga se javlja porukom na prozoru 4. Učenje mreže za praćenje se pokreće klikom na gumb „*Trening (praćenje)*“. Tu se također prilikom učenja greška ispisuje na konzoli, a kraj treninga se javlja porukom na prozoru 4. Naučene težine mreža se mogu učitati klikom na gumb „*Učitaj mrežu*“, a naučene težine nakon faze učenja se mogu spremiti klikom na gumb „*Spremi mrežu*“. Nakon što se mreže nauče ili učitaju klikom na potvrdne okvire se mogu odabrati načini rada. Potvrdni okvir „*Segmentiraj svaki frame*“ radi i prikazuje semantičku segmentaciju na svakoj sličici u nizu. Idući potvrdni okvir „*uključi praćenje*“ pokreće praćenje objekata na slikama u videu. Zadnji potvrdni okvir „*Detekcija segmentacijom*“ radi detekciju igrača u početnoj slici preko mreže za detekciju umjesto učitavanja iz formatirane datoteke. Nakon odabira željenih načina rada (mogu se međusobno kombinirati) pokretanje rada programa se radi klikom na gumb „*Play*“ u prozoru 1. Kod prozora 4 se nalazi na mediju koji je priložen uz rad.

6.2. Implementacija mreže za detekciju

Mreža za detekciju igrača je implementirana u programskom jeziku Python verzije 3.5 uz pomoć programske biblioteke TensorFlow verzije 1.1. i Scikit-learn programske biblioteke verzije 0.18.1. Model je definiran sljedećim dijelovima koda:

```
IMG_WIDTH = 1632
IMG_HEIGHT = 288
FLAGS = tf.app.flags.FLAGS
tf.app.flags.DEFINE_integer('img_width', IMG_WIDTH, '')
tf.app.flags.DEFINE_integer('img_height', IMG_HEIGHT, '')
tf.app.flags.DEFINE_integer('img_depth', 3, '')
tf.app.flags.DEFINE_integer('batch_size', 1, '')
tf.app.flags.DEFINE_integer('num_classes',4, '')

tf.app.flags.DEFINE_float('initial_learning_rate', 1e-4, """Initial learning rate.""")
tf.app.flags.DEFINE_float(
  'learning_rate_decay_factor',
  0.5,
  """Learning rate decay factor.""")
tf.app.flags.DEFINE_float('moving_average_decay', 0.9999, '')
```

```
data_shape = (FLAGS.batch_size, FLAGS.img_height, FLAGS.img_width, FLAGS.img_depth)
labels_shape = (FLAGS.batch_size, FLAGS.img_height, FLAGS.img_width)
self.__image = tf.placeholder(tf.float32, shape=data_shape)
self.__labels = tf.placeholder(tf.int32, shape=labels_shape)
self.__num_labels = tf.placeholder(tf.int32, shape=())
self.__apriori = tf.placeholder(tf.float32, shape=(4,))

config = tf.ConfigProto()
config.gpu_options.allocator_type = 'BFC'

self.__sess = tf.Session(config = config)

self.__global_step = tf.get_variable(
  'global_step',
  [],
  initializer = tf.constant_initializer(0),
  trainable = False)

decay_steps = 1000

lr = tf.train.exponential_decay(
  FLAGS.initial_learning_rate,
  self.__global_step,
  decay_steps,
  FLAGS.learning_rate_decay_factor,
  staircase=True)

with scopes.arg_scope([ops.conv2d, ops.fc], stddev=0.01, weight_decay=0.0005):
  with scopes.arg_scope([ops.conv2d, ops.fc, ops.dropout], is_training=True):
    net = conv_layer(self.__image, 'conv1_1', 'conv1_1_classify', dillate = False)
    net = conv_layer(net, 'conv1_2', 'conv1_2_classify', dillate = False)
    net = ops.max_pool(net, [2, 2], scope='pool1')
    net = conv_layer(net, 'conv2_1', 'conv2_1_classify', dillate = False)
    net = conv_layer(net, 'conv2_2', 'conv2_2_classify', dillate = False)
    net = ops.max_pool(net, [2, 2], scope='pool2')
    with scopes.arg_scope([ops.conv2d, ops.fc]):
      net = ops.conv2d(net, 1024, [1, 1], scope='fc6')
```

```
    net = ops.conv2d(net, FLAGS.num_classes, [1, 1], activation=None, scope='score'
    logits_up = tf.image.resize_bilinear(
      net,
      [FLAGS.img_height, FLAGS.img_width],
      name='resize_scores')

self.__logitsClassify = logits_up

self.__lossClassify = self.__bayesian_loss(
  self.__logitsClassify,
  self.__labels,
  self.__num_labels,
  self.__apriori)

self.__optClassify = tf.train.AdamOptimizer(lr).minimize(
  self.__lossClassify,
  aggregation_method = tf.AggregationMethod.EXPERIMENTAL_TREE)

self.__saver = tf.train.Saver()

init = tf.global_variables_initializer()
self.__sess.run(init)
```

U kodu je vidljivo da je ulaz u mrežu slika veličine 1632×288 piksela sa tri kanala boje, kao varijabla self.__image, a željeni izlaz mreže se predaje kao slika veličine 1632×288 piksela sa jednim kanalom u varijabli self.__labels. Varijabla self.__num se koristi za određivanje broja razreda za segmentaciju. Iz koda se također vidi da mreža koristi učenje sa eksponencijalnim padom kroz iteracije. Sama mreža je definirana u TensorFlowu uz pomoćne funkcije poput *conv_layer* i *ops.max_pool*. Greška koja se koristi je uravnoteženi gubitak koja je definirana funkcijom self.__bayesian_loss. Također se vidi da se mreža uči preko metode učenja Adam. Sama mreža se uči sljedećom funkcijom *„trainClassify*“:

```
def trainClassify(self, X, Y_):
CLASS_APRIORI = np.array([0, 0, 0, 0])

start_time = time.time()
run_ops = [
  self.__optClassify,
```

```
  self.__lossClassify,
  self.__logitsClassify,
  self.__global_step]

X = X.astype(np.float32)

for i in range(3):
  X[:,:,i] -= VGG_MEAN[i]

X = X.reshape(1, FLAGS.img_height, FLAGS.img_width, 3)
Y_ = Y_.astype(np.int32)
Y_ = Y_.reshape(1, FLAGS.img_height, FLAGS.img_width)
num_labels = (Y_ < 255).sum()

CLASS_APRIORI[0] = (Y_ == 0).sum()
CLASS_APRIORI[1] = (Y_ == 30).sum()
CLASS_APRIORI[2] = (Y_ == 60).sum()
CLASS_APRIORI[3] = (Y_ == 90).sum()
CLASS_APRIORI = 1 / (CLASS_APRIORI / np.sum(CLASS_APRIORI))
np.clip(CLASS_APRIORI, 0, 100, out = CLASS_APRIORI)

ret_val = self.__sess.run(
  run_ops,
  feed_dict={
    self.__image : X,
    self.__labels : Y_,
    self.__num_labels : num_labels,
    self.__apriori : CLASS_APRIORI}
)

(_, loss_val, scores, global_step_val) = ret_val

duration = time.time() - start_time
sec = float(duration)

format_str = 'frame: %d, loss = %.6f, time = %.3f sec'
print(format_str % (self.__step, loss_val, sec))
self.__step += 1
```

Iz koda se vidi da se ulaznoj slici prvo oduzmu srednje vrijednosti kanala za VGG16 mrežu, a zatim se dobije broj razreda tako da se izračuna broj elemenata manjih od maksimalne moguće vrijednosti. Idući korak je dobivanje broja pojedinačnih različitih razreda za uravnoteženi gubitak. Nakon toga se vrši učenje mreže. Funkcija „*evalClassify*“ radi segmentaciju slike i detekciju igrača te je definirana na ovaj način:

```
def evalClassify(self, X, Y_):
CLASS_APRIORI = np.array([0, 0, 0, 0])

start_time = time.time()
run_ops = [self.__lossClassify, self.__logitsClassify]

X = X.astype(np.float32)

for i in range(3):
  X[:,:,i] -= VGG_MEAN[i]

X = X.reshape(1, FLAGS.img_height, FLAGS.img_width, 3)
Y_ = Y_.astype(np.int32)
Y_ = Y_.reshape(1, FLAGS.img_height, FLAGS.img_width)
num_labels = (Y_ < 255).sum()

CLASS_APRIORI[0] = (Y_ == 0).sum()
CLASS_APRIORI[1] = (Y_ == 30).sum()
CLASS_APRIORI[2] = (Y_ == 60).sum()
CLASS_APRIORI[3] = (Y_ == 90).sum()
CLASS_APRIORI = 1 / (CLASS_APRIORI / np.sum(CLASS_APRIORI))
np.clip(CLASS_APRIORI, 0, 100, out = CLASS_APRIORI)

ret_val = self.__sess.run(
  run_ops,
  feed_dict={
    self.__image : X,
    self.__labels : Y_,
    self.__num_labels : num_labels,
    self.__apriori : CLASS_APRIORI}
```

```
)

(loss_val, out_logits) = ret_val
min = 0.05 * np.amax(out_logits[0])
max = np.amax(out_logits[0], axis = 2)

y = out_logits[0].argmax(2).astype(np.int32)
y[max < min] = 0
indexes = []

for i in range(3):
  indexes.append(np.where(y == (i + 1)))

retCentroids = np.empty(shape=(0,2),dtype = np.int32)

for classNum in range(3):
  Y = np.vstack(indexes[classNum])

  if (Y.shape[1] != 0):
    numCluster = int(50)

    if (Y.shape[1] < numCluster):
      numCluster = int(Y.shape[1] / 2)

    kmeans = KMeans(n_clusters = numCluster, max_iter = 200).fit(Y.T)
    unique, count = np.unique(kmeans.labels_, return_counts = True)
    centroids = kmeans.cluster_centers_.astype(np.int32)

    if (count[np.argmax(count)] < CLUSTER_COUNT):
      retCentroids = np.vstack(
        (retCentroids,centroids[np.argmax(count)].astype(np.int32)))
    else:
      while (True):
        minimum = np.argmin(count)
        if (count[minimum] < CLUSTER_COUNT):
          centroids = np.delete(centroids, [minimum], axis = 0)
          count = np.delete(count, [minimum], None)
          unique = np.delete(unique, [minimum], None)
```

```
        else:
          break
      while (len(centroids) > 0):
        centroidList = []

        for centroidIndex in range(len(centroids)):
          if (distance(centroids[0], centroids[centroidIndex]) < RANGE):
            centroidList.append(centroidIndex)
          max = count[0]

          if (len(centroidList) < 2):
            centroids = np.delete(centroids, centroidList, axis = 0)
            count = np.delete(count, centroidList, None)
            unique = np.delete(unique, centroidList, None)
          else:
            centroidMax = 0

            for centroid in centroidList:
              if (count[classNum] > max):
                max = count[classNum]
                centroidMax = centroid

            retCentroids = np.vstack(
              (retCentroids, centroids[centroidMax].astype(np.int32)))

            centroids = np.delete(centroids, centroidList, axis = 0)
            count = np.delete(count, centroidList, None)
            unique = np.delete(unique, centroidList, None)

width = y.shape[1]
height = y.shape[0]
y_rgb = np.empty((height, width, 3), dtype=np.uint8)

for cid in range(len(CLASS_INFO)):
  cpos = np.repeat((y == cid).reshape((height, width, 1)), 3, axis=2)
  cnum = cpos.sum() // 3
  y_rgb[cpos] = np.array(CLASS_INFO[cid][:3] * cnum, dtype=np.uint8)
```

```
duration = time.time() - start_time
sec = float(duration)

format_str = 'loss = %.6f, time = %.3f sec'
print(format_str % (loss_val, sec))

return y_rgb, retCentroids
```

Iz koda je vidljivo da funkcija „*EvalClassify*“ prima ulaznu sliku te radi semantičku segmentaciju. Nakon što se dobije izlaz gleda se na kojim su pikselima najveći odzivi te ih se klasificira po razredima u nove slike. Nakon toga se u novu listu spremaju lokacije tih piksela, Za svaku sliku sa razredima tada se radi KNN algoritam grupiranja. Napravi se više centroida nego što se smatra da ustvari ima objekata na slici te se radi filtriranje. To se radi radi toga što se mogu pojaviti duplikati na nekom mjestu, a potpuno zanemarivanje drugih mjesta. Nakon toga se vrši bojanje slike tj. vizualizacija semantičke segmentacije te se preostali centroidi i segmentirana slika vraćaju kao izlaz iz funkcije.

6.3. Implementacija mreže za praćenje

Mreža za praćenje je također implementirana u programskom jeziku Python verzije 3.5 te uz pomoć programske biblioteke TensorFlow 1.1. Model je definiran sljedećim kodom:

```
self.__trackData = tf.placeholder(tf.float32, shape=(None, 30, 20, 3))
self.__trackLabels = tf.placeholder(tf.float32, shape=(None, 2))

image = tf.reshape(self.__trackData, shape = [-1, 30, 20, 3])

config = tf.ConfigProto()
config.gpu_options.allocator_type = 'BFC'

self.__sess = tf.Session(config = config)

self.__global_step_track = tf.get_variable(
  'global_step_track',
  [],
  initializer = tf.constant_initializer(0),
  trainable = False)

decay_steps = 1000
```

```
lrTrack = tf.train.exponential_decay(
  5e-5,
  self.__global_step_track,
  decay_steps,
  FLAGS.learning_rate_decay_factor,
  staircase=True)

with scopes.arg_scope([ops.conv2d, ops.fc], stddev=0.01, weight_decay=0.0005):
  with scopes.arg_scope([ops.conv2d, ops.fc, ops.dropout], is_training=True):
    netTrack = conv_layer(self.__trackData, 'conv1_1', 'conv1_1_track', dillate = False)
    netTrack = ops.max_pool(netTrack, [2, 2], scope='poolTrack1')
    netTrack = conv_layer(netTrack, 'conv2_1', 'conv2_1_track', dillate = False)
    netTrack = ops.max_pool(netTrack, [2, 2], scope='poolTrack2')
    netTrack = ops.flatten(netTrack, scope='flatten1')
    netTrack1 = ops.fc(netTrack, 256, scope='fc1')
    netTrack2 = ops.fc(netTrack1, 128, scope='fc2')
    logits_track = ops.fc(netTrack2, 2, activation = None, scope='logits')

self.__logTrack = tf.nn.softmax(logits_track)
self.__lossTrack = tf.reduce_mean(
  tf.nn.softmax_cross_entropy_with_logits(
    labels=self.__trackLabels,
    logits=logits_track))

self.__optTrack = tf.train.AdamOptimizer(lrTrack).minimize(
  self.__lossTrack,
  aggregation_method = tf.AggregationMethod.EXPERIMENTAL_TREE)

self.__saver = tf.train.Saver()

init = tf.global_variables_initializer()
self.__sess.run(init)
self.__stepTrack = 1
```

Iz koda je vidljivo da mreža za praćenje ima sličan izgled kao i mreža za detekciju. Razlike su u veličini ulazne slike, dubini i strukturi same mreže, nešto manjem koraku učenja te

funkciji gubitka koja je sada gubitak unakrsne entropije. Mreža se uči funkcijom „*trainTracker*“ koja je definirana sljedećim kodom:

```
def evalTracker(self, x, y_):
run_ops = [self.__optTrack, self.__lossTrack, self.__logTrack, self.__global_step_track]


ret_val = self.__sess.run(run_ops, feed_dict={self.__trackData : x, self.__trackLabels : y_})
(_, loss_val, scores, global_step_val) = ret_val


self.__stepTrack += 1
return loss_val
```

Iz koda se vidi da je funkcija jako jednostavna i da joj je jedina svrha učenje mreže za praćenje. Mreža evaluira ulaz funkcijom „*evalTracker*“ koja je definirana sljedećim kodom:

```
def evalTracker(self, x, y_):
run_ops = [self.__lossTrack, self.__logTrack]


ret_val = self.__sess.run(run_ops, feed_dict={self.__trackData : x, self.__trackLabels : y_})
(loss_val, out_logits) = ret_val


return out_logits
```

Ova funkcija je također jako jednostavna i jedina joj je svrha evaluacija ulaza koji dobije.

6.4. Implementacija algoritma praćenja

Algoritam praćenja je napisan u programskom jeziku Python verzije 3.5. Ovaj algoritam se nalazi u kodu za „prozor 4“ jer se tamo nalaze objekti za mreže za detekciju i za prećenje. Pošto bi cijeli kod algoritma zauzimao previše mjesta u radu ovdje će biti naveden njegov pseudokod:

```
Uzmi trenutnu sliku iz videa;
Stvori praznu listu novih lokacija;


Za svaki od objekata koji se prate:
	Uzmi područje interesa objekta;
	Modificiraj veličinu područja interesa da odgovara ulazu u mrežu;
	Za maksimalni izlaz postavi izlaz iz mreže za praćenje za područje interesa;
	Dok je apsolutni pomak ulijevo ili udesno manji od ograničenja:
```

```
                Izračunaj izlaze iz mreže za praćenje za okolinu trenutne lokacije;
                Nađi novi maksimalni izlaz;
                Ako je novi maksimalni izlaz veći prošlog maksimalnog izlaza:
                        Pomakni se u smjeru novog maksimalnog izlaza;
                        Postavi maksimalni izlaz na novi maksimalni izlaz;
        Ako je maksimalni iznos veći od praga:
                Dodaj lokaciju maksimalnog izlaza u listu novih lokacija;

Provjeri postoji li kolizija između igrača i ako ima grupiraj ih u kolizijsku listu;

Ako postoji kolizija i u trenutnoj slici se provjerava ima li kolizije:
        Stvori praznu listu pronađenih;
        Za svaku grupiranu koliziju:
                Izračunaj izlaze mreže za daljnju okolinu oko grupe;
                Nađi maksimalni izlaz;
                Ako je maksimalni izlaz veći od praga:
                        Pomakni grupiranog igrača na lokaciju maksimalnog izlaza;
                        Dodaj grupiranog igrača natrag u listu novih lokacija;

Ako se u trenutnoj slici provjerava skaliranje pravokutnika regija interesa:
        Za svaki objekt koji se prati:
                Uzmi područje interesa objekta;
                Modificiraj veličinu područja interesa da odgovara ulazu u mrežu;
                Za maksimalni izlaz postavi izlaz iz mreže za praćenje za područje interesa;
                Izračunaj izlaze iz mreže za skalirane pravokutnike oko objekata;
                Nađi maksimalni izlaz za skalirane pravokutnike;
                Ako je maksimalni izlaz skaliranih pravokutnika veći od originalnog:
                        Postavi trenutnu veličinu pravokutnika na skaliranu veličinu;
Vrati listu novih lokacija;
```

Kao što se može vidjeti iz pseudokoda algoritam za praćenje je podijeljen u 3 koraka. U prvom koraku se radi praćenje prema algoritmu koji je opisan u poglavlju 4.3. Nakon toga se radi detekcija kolizije između igrača te ako ona postoji onda se ide u drugi korak. U drugom koraku se provjerava da li su se igrači koji su sad u kolizijskoj grupi pomaknuli iz nje te ako jesu onda ih se miče iz te grupe natrag u grupu za praćenje pojedinačnih igrača. Treći korak se radi periodično da se provjeri da li se igrač povećao ili smanjio na terenu te da se ovisno o tome podesi veličine regije interesa.

7. Eksperimentalni rezultati

7.1. Opis metoda i definiranje funkcija greški

Eksperimentalna evaluacija je provedena na videu „*t7.mp4*“ koji sadrži već prije spomenutu nogometnu utakmicu. On je za sitno duži od 5:00 minuta te sadrži 7515 sličica i brzina izvođenja mu je 25 sličica u sekundi. Razlučivost mu je 1632×288 piksela i sadrži 3 8-bitna kanala u boji. Model je učen i provjeren na slikama iz već spomenutog videa. Razlog za takvu metodu umjesto korištenja više različitih videa sličnih karakteristika jest taj što bi se model za praćenje u stvarnim uvjetima trebao naučiti na prvih nekoliko slika na početku videa te kasnije nastaviti praćenje. Bitna stvar za provođenje evaluacije jest definiranje načina mjerenja greške prilikom praćenja. Postoji nekoliko načina za mjerenje kvalitete praćenja, a jedan od popularnijih načina jest mjerenje sa presjekom unije pravokutnika za izlaz modela praćenja i prave lokacije u svakoj slici u videu. Ovdje se ta metoda neće koristiti jer će se kasnije uspoređivati praćenje sa detektorom koji radi na malo drugačiji način. Stoga su odabrane dvije funkcije greške praćenja. Prva od njih je nazvana **apsolutna greška**. Ona je definirana izrazom (7.1):

$$Error = \sum_i^N [\![d(p_i, q_i) < \delta]\!] \tag{7.1}$$

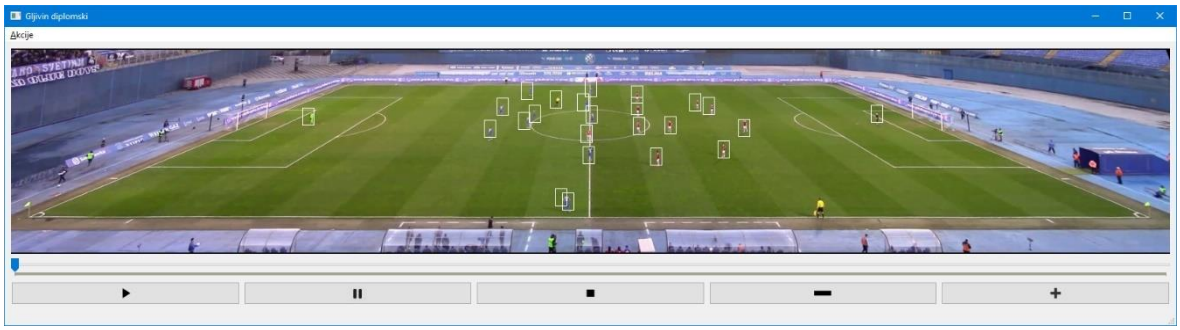
gdje je N broj objekata za praćenje, funkcija d je euklidska udaljenost, p_i lokacija trenutnog objekta dobivena modelom praćenja, q_i stvarna lokacija trenutnog objekta, a δ maksimalna dopuštena udaljenost koja je u ovom slučaju 10 piksela. Ova funkcija greške ustvari daje ocjenu da li se neki objekt izgubio ili nije u trenutnoj slici u videu. Druga greška je nazvana **ukupna greška**. Ona je definirana izrazom (7.2):

$$Error = \sum_i^N \frac{\|p_i - q_i\|}{\delta} \tag{7.2}$$

gdje je N broj objekata za praćenje, p_i lokacija trenutnog objekta dobivena modelom praćenja, q_i stvarna lokacija trenutnog objekta, a δ skaliranje greške koje je u ovom slučaju također 10 radi bolje usporedbe sa iznosom apsolutne greške. Druga funkcija greške za razliku mjeri pikselnu preciznost praćenja za svaki objekt u trenutnoj slici. Iz tog razloga ona puno više kažnjava svako odstupanje od stvarne lokacije te je za očekivati da će ona biti većeg iznosa od apsolutne greške. Sama eksperimentalna evaluacija je izvršena na 3 načina. Prvi je usporedba kvalitete praćenje sa detektorom, drugi je usporedba kvalitete praćenja u ovisnosti o dužini učenja i treći je usporedba brzine izvođenja detekcije i praćenja na istom računalu.

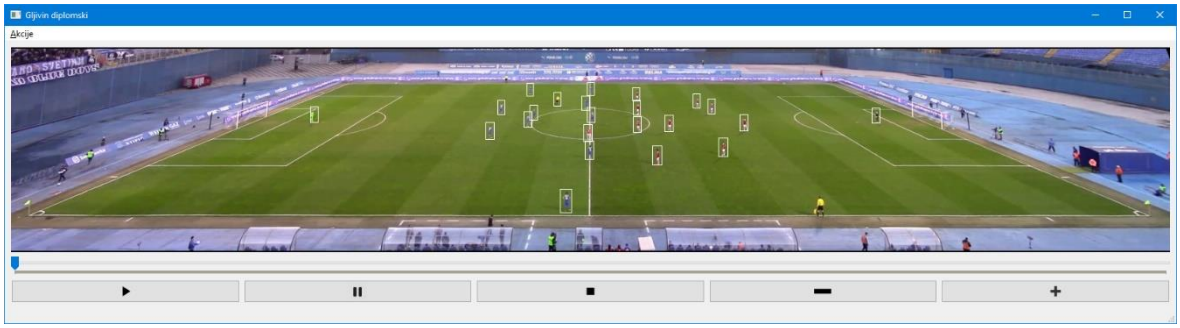
7.2. Usporedba kvalitete praćenja modela za detekciju i modela za praćenje

Prva usporedba kvalitete praćenja uključuje usporedbu modela za detekciju i modela za praćenje. Kao što je poznato metoda praćenja se može u potpunosti zamijeniti samo uzastopnim ponavljanjem metode detekcije na svakoj sličici u videu. To se ponekad stvarno i radi, ali se pokušava izbjeći bržim i točnijim modelima koji su namijenjeni upravo za praćenje objekata. Ovdje se upravo to želi testirati, tj. vidjeti da li je model za praćenje točniji od modela za čistu detekciju. Korišteni model za detekciju je objašnjen u poglavlju 4.2. i izlaz mu je prikazan na slici 44:



Slika 44: Izlaz iz modela za detekciju objekata (igrača). Vide se pravokutnici koji označavaju detektirane igrače.

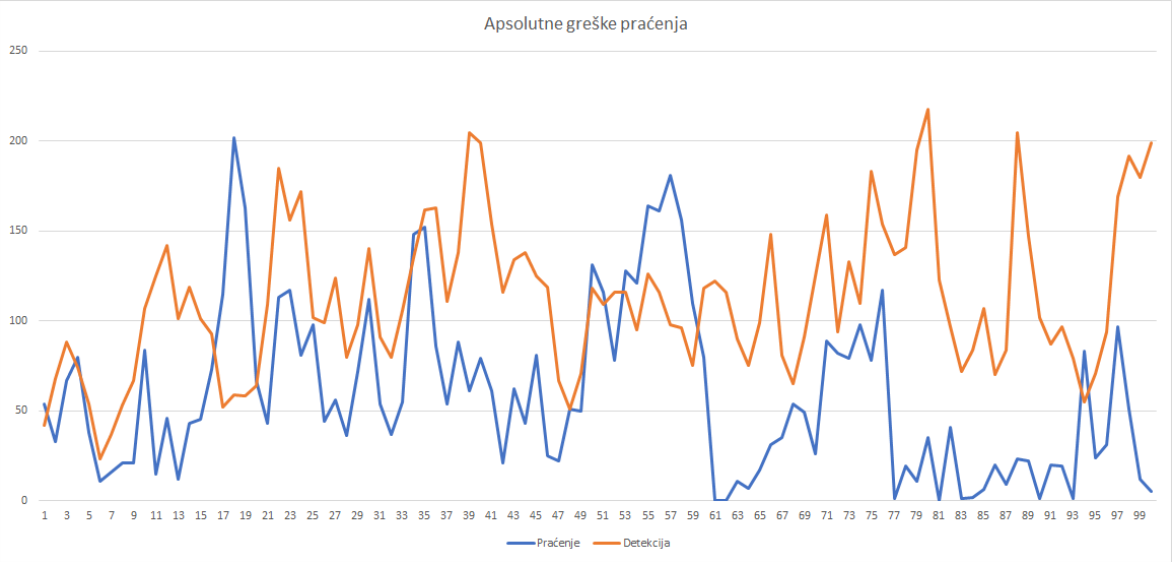
Usporedba se radila sa modelom za praćenje koji je objašnjen u poglavlju na 4.3. i izlaz mu je prikazan na slici 45:



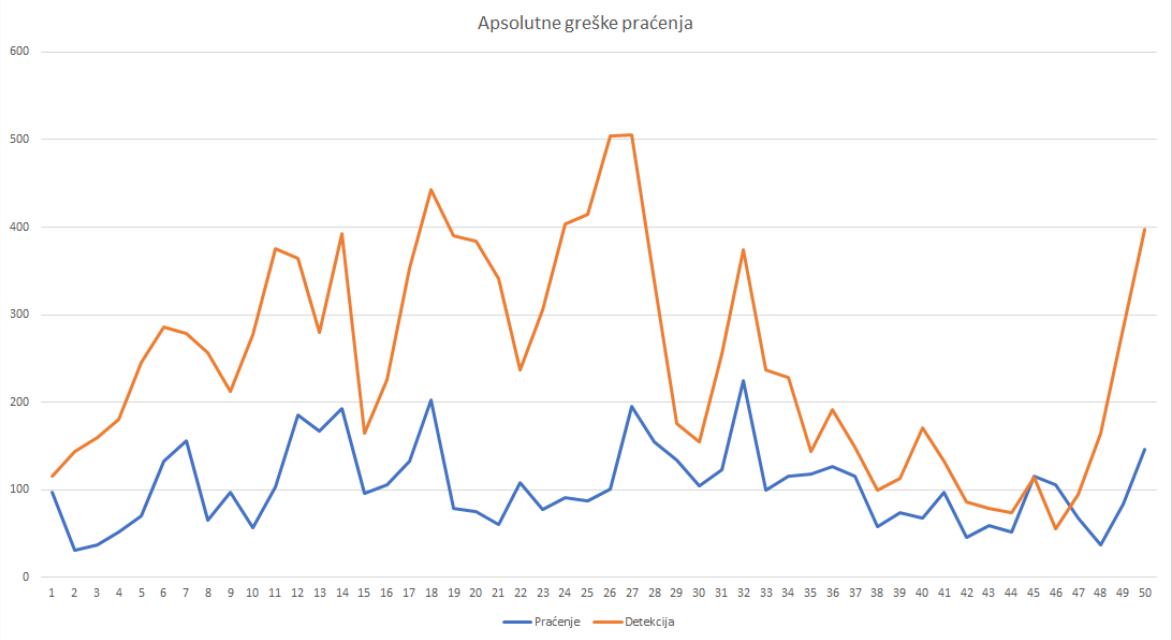
Slika 45: Izlaz iz modela za praćenje objekata (igrača). Također se vide pravokutnici oko igrača koji se prate.

Način testiranja je bio takav da se prilikom praćenja modelom za detekciju svaku sliku radila detekcija dok se za model detekcije u početnoj te svakih 50 sličica za lokaciju svakog igrača postavljala njegova stvarna lokacija iz formatirane datoteke pa se radilo praćenje. To se radi načina rada detektora objekata nažalost nije moglo napraviti tako da se ta metoda koristila samo za model za praćenje. Greška se mjerila preko metoda koje su navedene u poglavlju 7.1. što znači da su se mjerile apsolutna i ukupna greška. Oba modela su učena na prvih 5000 slika u videu te su testirani na idućih 2500 slika. Prilikom faze učenja model za detekciju se učio sa 1 slikom po iteraciji učenja dok se model za praćenje učio sa 81 slikom u grupi za

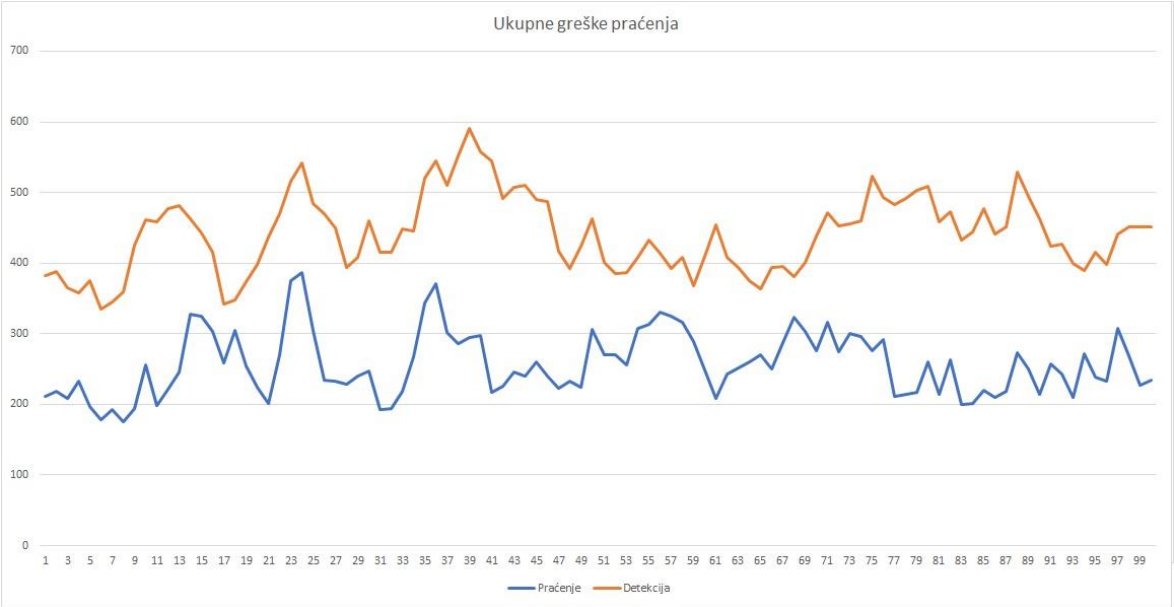
učenje (*eng. batch*) po iteraciji. Za regularizaciju i pokušaj smanjivanja pretreniranosti modela je korištena blaga L2 regularizacija sa iznosom od 0.0005. Apsolutna i ukupna greška su sumirane na svakih 50 sličica sve do kraja testiranja. Dobiveni rezultati su prikazani na slikama 46, 47, 48 i 49:



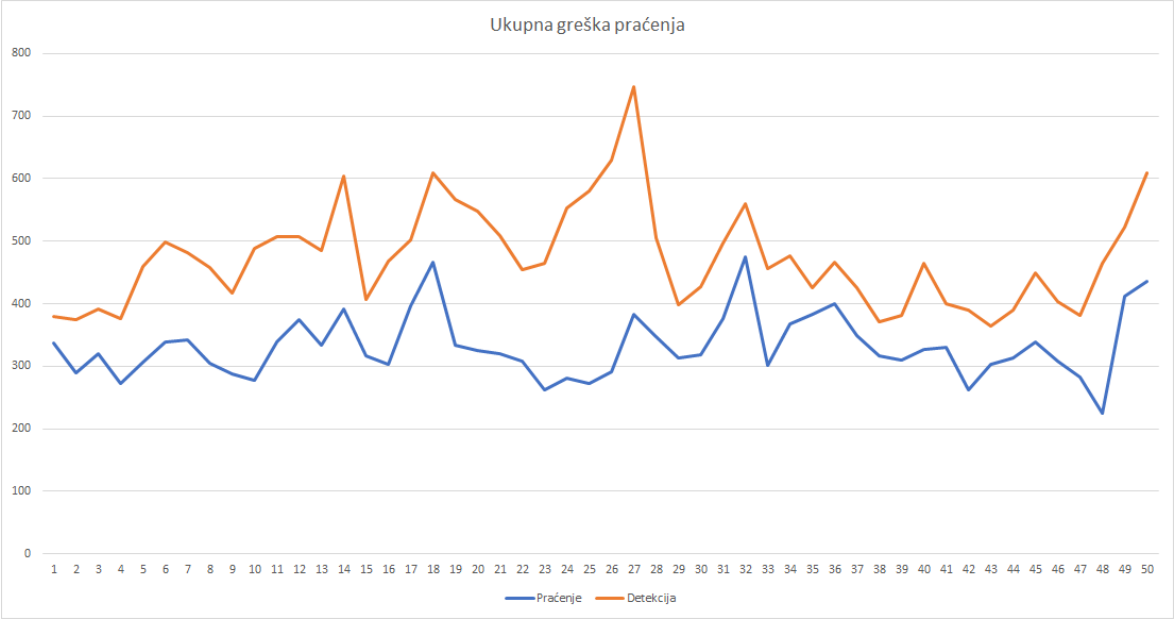
Slika 46: Usporedba apsolutnih grešaka oba modela na setu za učenje.



Slika 47: Usporedba apsolutnih grešaka oba modela na testnom setu.



Slika 48: Usporedba ukupnih grešaka oba modela na setu za učenje.



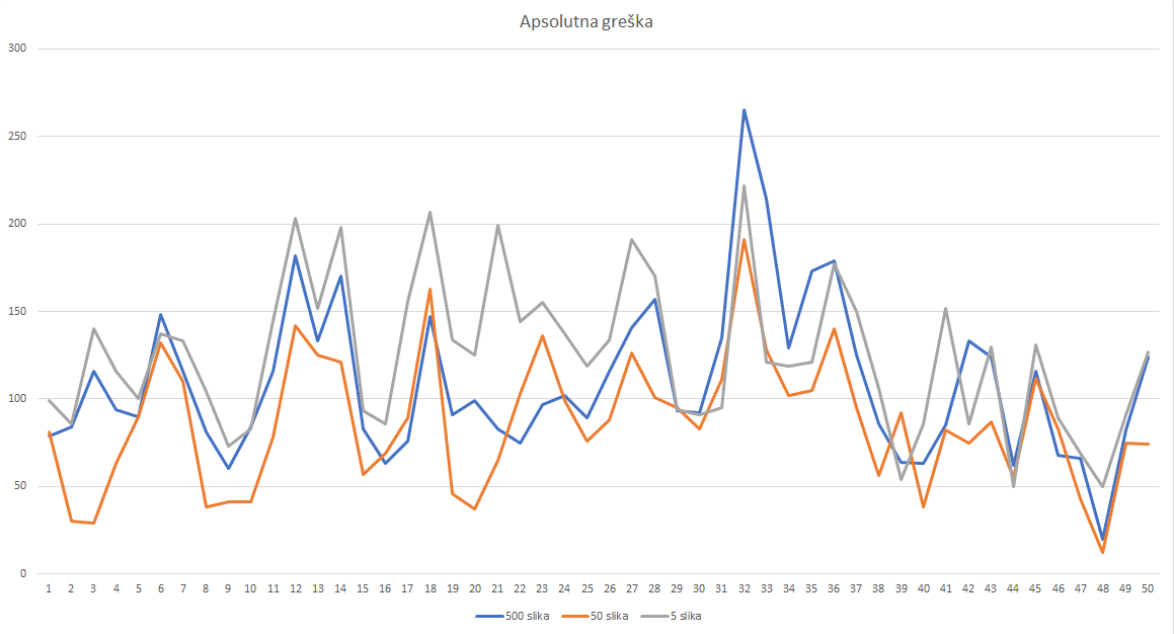
Slika 49: Usporedba ukupnih grešaka oba modela na testnom setu.

Iz rezultata se da vidjeti da je model za praćenje u svim slučajevima imao osjetno bolje rezultate, pogotovo od 950-e do 1450-e sličice u dijelu za testiranje. Ti rezultati se mogu pripisati tome što model za detekciju radi detekciju svakog igrača svaku sličicu te tome što ponekad zna grupirati igrače istog tima kada su jako blizu jedan drugoga kao jednog igrača. Također model za praćenje ima par metodu kojom može izbjeći tu situaciju i radi pretraživanje okoline prošle lokacije igrača te ima veću mogućnost pronaći kuda se igrač pomaknuo u sljedećoj sličici. Ono što se također može uočiti su bolji rezultati na testu za učenje što ukazuje na laganu pretreniranost oba modela. To znači da bi povećanje

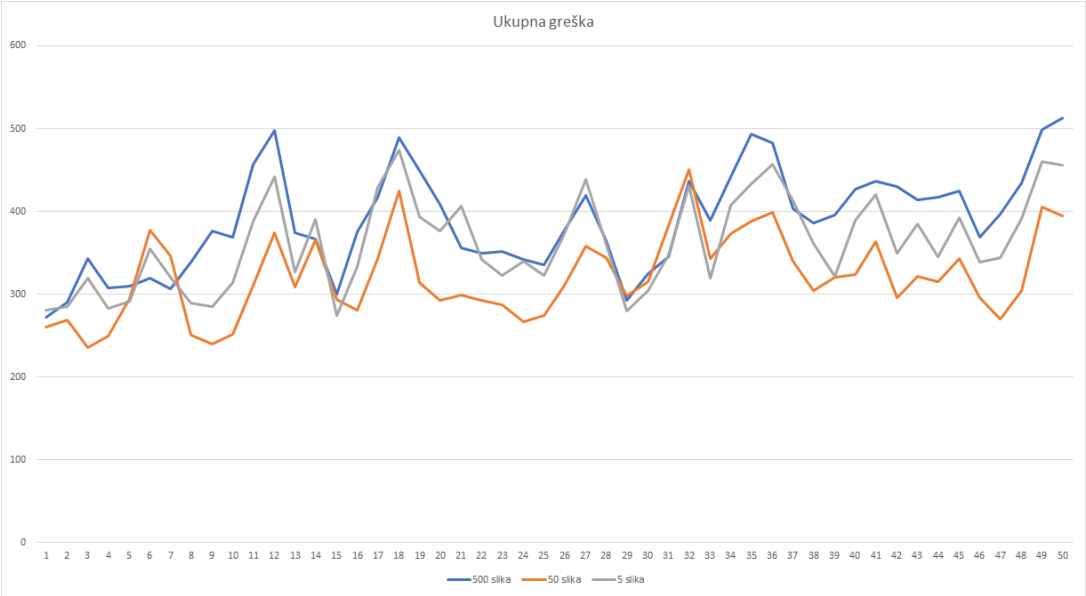
regularizacije oba modela moglo dati veća poboljšanja prilikom praćenja na testnom setu podataka. Osim toga to bi vrlo vjerojatno poboljšalo rezultate na onih 500 sličica za model sa detekcijom.

7.3. Usporedba kvalitete praćenja u ovisnosti o dužini faze učenja

Druga usporedba kvalitete praćenja jest usporedba kvalitete praćenja u ovisnosti o dužini faze učenja. Taj test obuhvaća samo mjerenja na modelu za praćenje. Pretpostavka je da bi dobar model za praćenje trebalo biti relativno lako naučiti tj. da mu treba relativno mali broj ulaznih slika da bi naučio dobro pratiti objekte. Zato se u ovom testu koriste brojke od 5, 50 i 500 iteracija učenja. Svaka od ih brojki također označava i koliko se početnih slika koristilo za učenje modela za praćenje. Nakon što je model naučen sa tim brojem slika testiranje se vršilo ponovno na zadnjih 2500 slika u videu sa istom metodologijom kao i u prošlom testu radi konzistentnosti. Dobiveni rezultati su prikazani na slikama 50 i 51:



Slika 50: Apsolutna greška u ovisnosti o broju iteracija učenja.

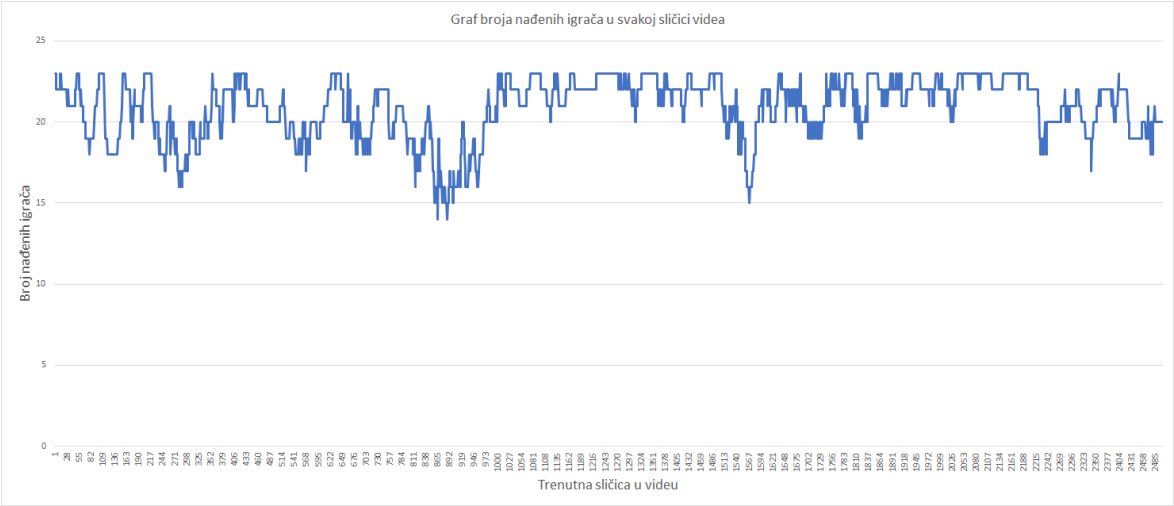


Slika 51: Ukupna greška u ovisnosti o broju iteracija učenja.

Iz rezultata se vidi da povećanje broja iteracija ima pozitivan utjecaj na kvalitetu praćenja, ali sve do neke granice. Prilikom pomaka sa 5 iteracija na 50 vidi se osjetno poboljšanje u kvaliteti praćenja, dok sa povećanjem na 500 dolazi do laganog pogoršanja rezultata. To se može objasniti ili statističkom greškom koja bi se ispravila većim brojem testova ili učenjem na kasnijem skupu slika koje nisu toliko pogodne za učenje praćenja, npr. učenje na igračima koji se međusobno prekrivaju. Ali i sa tim rezultatom se može zaključiti da mreža dobro prati i sa molim brojem iteracija, ali i da povećanje broja iteracija definitivno pomaže.

7.4. Mjerenje broja izgubljenih objekata u ovisnosti o trenutnoj sličici

Zadnje testiranje koje je izvršeno jest mjerenje trenutnog broja objekata (igrača) koji se prate u svakoj sličici videa. Mreža se prvo naučila na početne dvije trećine videa nogometne utakmice te se na drugoj trećini vršilo praćenje. Svakih 1000 sličica se vršila ponovna detekcija igrača da se izgubljeni igrači ponovno počnu pratiti. Broj praćenih objekata se mjerio sa apsolutnom greškom tj. svaki objekt koji se po toj vrsti greške izgubio se računa da se više ne prati. Maksimalan broj objekata koji su se pratili je 23. Dobiveni rezultati su prikazani na slici 52:



Slika 52: Graf broja nađenih igrača u svakoj sličici videa. 23 je maksimum, a svaki broj ispod njega označava da su se neki igrači izgubili prilikom praćenja.

Iz rezultata se vidi da mreža u većini situacija jako dobro prati objekte na terenu. Lošiji rezultati između sličica 600 i 1000 se daju objasniti sa situacijom na terenu kada su svi igrači blizu gola te se javlja gužva te se mora raditi mnogo ponovnih detekcija međusobno prekrivenih igrača. Također je vidljivo da, ako se zanemare takve jako teške situacije, sustav za praćenje i ponovnu detekciju rade jako dobro.

7.5. Usporedba brzine izvođenja modela za detekciju i modela za praćenje

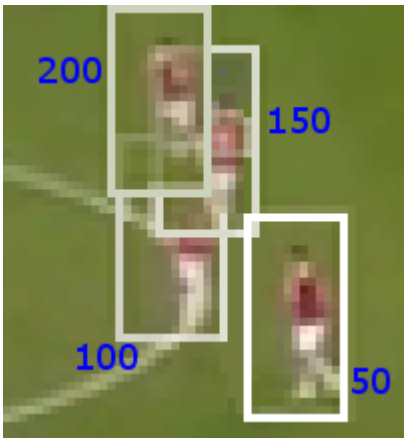
Treća usporedba kvalitete praćenja jest usporedba brzine izvođenja modela za detekciju i modela za praćenje. Jedna od glavnih karakteristika dobrih modela za praćenje jest brže izvođenje u odnosu na modele za detekciju objekata. Ovdje će se stoga usporediti modeli za detekciju i praćenje objekata koji su objašnjeni u poglavlju 4.2. i 4.3. Test je izvršen na prvih 100 slika u videu gdje se mjerila brzina izvođenja za svaku sličicu te je na kraju napravljen prosjek izvođenja. Test je izvršen na računalu sa procesorom Intel Core i5-6300HQ, 8 GB RAM memorije i grafičkom karticom NVIDIA GeForce GTX 950M sa 4 GB video memorije. Na računalo je također snimljena verzija TensorFlowa 1.1 koja se izvršava na grafičkoj kartici računala i pratilo se 23 objekata po slici. Dobiveni rezultati su navedeni u sljedećoj tablici:

Način praćenja objekata	**Prosječno vrijeme izvođenja po sličici**
Metoda detekcije	0.681 s
Metoda praćenja	0.198 s

Kao što je vidljivo iz rezultata mjerenja metoda praćenja je osjetno brža od metode detekcije (tj. 3.44 puta je brža). To se može objasniti plićom arhitekturom modela, osjetno manjim ulazom u mrežu te sa ako se sve to spoji osjetno manjim brojem operacija koje se moraju izvršiti.

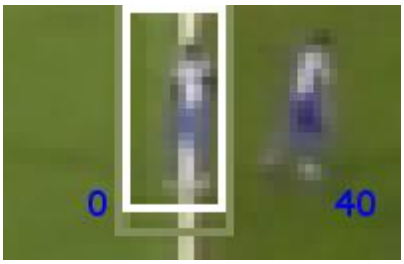
7.6. Pozitivni i negativni primjeri praćenja objekata

Model za praćenje koji je dobro naučen općenito nema problema sa praćenjem više objekata koji su dobro definirani u okolini neovisno o tome da li im se mijenja veličina ili oblik. To se može dobro vidjeti na pozitivnom primjeru praćenja na slici 53 koja prikazuje praćenje objekta na 200 sličica zaredom bez osvježavanja detektorom ili točnom lokacijom iz formatirane datoteke.



Slika 53: Pozitivan primjer praćenja objekata na 200 sličica zaredom. Na slici je prikazana lokacija objekta u trenutnoj sličici u videu.

Puno bitnije situacije od ovih su one negativne i savjeti kako ih izbjeći. Ponekad se zna dogoditi da detektor odluta od objekta kojeg prati makar je relativno dobro definiran u okolini. Primjer za to je prikazan na slici 54:

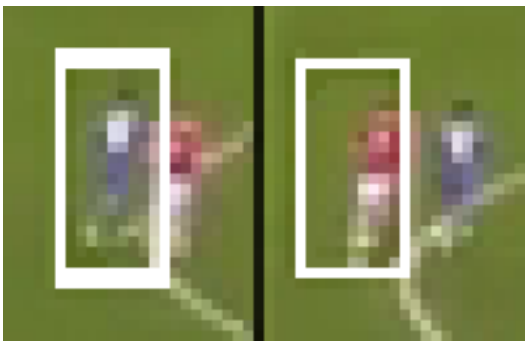


Slika 54: Gubitak objekta između 40 sličica.

Taj problem se najčešće javlja radi loše provedenog učenja modela za praćenje. Primjer iste situacije, ali sa bolje provedenom fazom učenja prikazana je na slici 55:

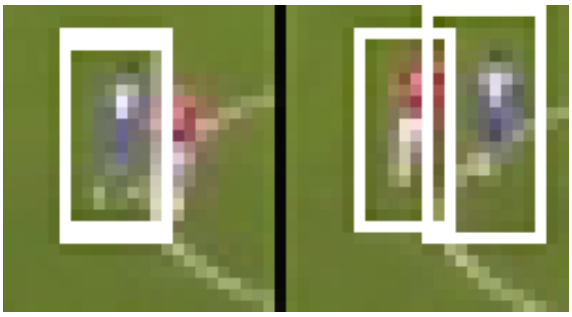
Slika 55: Primjer sa boljom fazom učenja. Vidljiv je uspješan nastavak praćenja u istoj situaciji u odnosu na sliku 51.

Drugi čest primjer gubitka objekta prilikom praćenja jest prilikom međusobnog preklapanja dvaju objekata (igrača). Taj problem je prikazan na slici 56:



Slika 56: Neuspješno razrješenje kolizije. Model za praćenje je prebacio oba pravokutnika na igrača crvenog tima te ga nastavio pratiti.

Taj problem se javlja radi toga što se informacije o igraču u pozadini gube te se mora ponovno naći kad se maknu jedan od drugoga. Problem što je teško predvidjeti kad će se oni razdvojiti i na koji način će se razdvojiti stoga je to jedan od težih problema u ovom radu. On se pokušava riješiti metodom razrješenja kolizije koja je opisana u poglavlju 4.3. koja radi dobro, ali bi se prva nadogradnja modela trebala baviti poboljšanjem te metode. Primjer uspješno razriješene kolizije je prikazan na slici 57:



Slika 57: Uspješno razriješena kolizija. Model je uspješno vratio pravokutnik za praćenje na igrača plavog tima.

8. Zaključak

Područje računalnog vida koje se bavi praćenjem objekata uz pomoć dubokih konvolucijskih modela je još uvijek jako nedorečeno u vrijeme pisanja ovog rada. Radi toga se u ovom radu morao razviti novi model za praćenje objekata uz pomoć neuronskih mreža. Kao što se iz rezultata vidi taj model dokazuje da se konvolucijske neuronske mreže mogu uspješno koristiti i za problem praćenja objekata. Mreža odlično prati objekte u većini normalnih situacija kada se objekt barem relativno dobro ističe u okolini i ako ima oblik koji se ne mijenja u potpunosti prilikom tijeka videa. Također performanse izvođenja su osjetno bolje nego za metodu detekcije te bi se video ove razlučivosti sa jačom grafičkom karticom mogao izvoditi i u stvarnom vremenu. Problemi koji se javljaju poput međusobnog preklapanja objekata ili skaliranja objekata u videu se mogu relativno uspješno riješiti metodama nakon izlaza iz konvolucijske neuronske mreže. U budućem radu bi se kao prvo poboljšanje mogla napraviti robusnija metoda ponovne detekcije objekata nakon kolizije (prekrivanja). Drugo poboljšanje bi moglo biti optimizacija arhitekture modela sa više konvolucijskih slojeva gdje svaki od njih ima manji broj izlaza nego sada jer su takvi modeli pokazali odlične rezultate prilikom testiranja na setu podataka ImageNet. To poboljšanje bi lako dovelo i do boljih performansi izvođenja. Treće poboljšanje bi moglo biti poboljšanje modela za detekciju tako da se implementira SSD ili YOLO model. Četvrto poboljšanje bi moglo biti dodavanje vektora brzine svakom objektu. S tim bi se mogla raditi predikcija lokacije objekta u idućoj slici te se tako potencijalno smanjiti broj iteracija traženja objekta. Zadnje poboljšanje bi mogao biti potpuni redizajn mreže tako da se ona uči sa kraja na kraj. To bi značilo da bi se mreža naučila detektirati kretanje, promjenu veličine objekta i njihovo međusobno preklapanje prilikom faze učenja. To je veoma težak problem koji je bio isproban prilikom izrade ovog rada, ali je radi loših performansi odbačen, ali bi se poboljšanjem tog modela možda mogle dobiti jako dobre performanse te to vrijedi pokušati u budućnosti.

9. Literatura

1. Mei X., Porikli F.: Joint Tracking and Video Registration by Factorial Hidden Markov Models; ICASSP 2008; Las Vegas, Nevada, SAD; 31.3. - 4.4. 2008.g.;str. 6.

2. Definition, Discrete Convolution; 13.5.2017.g.; *Wikipedia: Convolution*; https://en.wikipedia.org/wiki/Convolution; 4.6.2017.g.

3. Josip Krapac: Konvolucijske neuronske mreže, 2016.g., *prezentacija sa predmeta Duboko učenje na Fakultetu elektrotehnike i računarstva*; http://www.zemris.fer.hr/~ssegvic/du/du2convnet.pdf; 4.6.2017.g.

4. Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition; Pooling layer; proljeće 2017.g.; *Convolutional Neural Networks (CNNs, ConvNets)*; http://cs231n.github.io/convolutional-networks/; 5.6.2017.g.

5. Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition; Commonly used activation functions; proljeće 2017.g.; *Neural Networks*; http://cs231n.github.io/neural-networks-1/; 6.6.2017.g.

6. Leonardo Araujo dos Santos; Fully convolutional network for segmentation, Conversion from normal CNN to FCN; 6.6.2017.g.; *Image Segmentation*; https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/image_segmentation.html; 8.6.2017.g.

7. Cross-entropy error function and logistic regresion; 25.4.2017.g.; *Wikipedia: Cross entropy*; https://en.wikipedia.org/wiki/Cross_entropy; 9.6.2017.g.

8. Marko Čupić: Optimizacija parametara modela, 2016.g., *prezentacija sa predmeta Duboko učenje na Fakultetu elektrotehnike i računarstva*; http://www.zemris.fer.hr/~ssegvic/du/du3optimization.pdf; 9.6.2017.g.

9. Sebastian Ruder; Gradient descent optimization algorithms: Adam; 19.1.2016.g.; *An overview of gradient descent optimization algorithms*; http://sebastianruder.com/optimizing-gradient-descent/; 9.6.2017.g.

10. Leonardo Araujo dos Santos; Introduction, RCNN, Fast RCNN, Faster RCNN; 6.6.2017.g.; *Object Localization and Detection*; https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/object_localization_and_detection.html; 12.6.2017.g.

11. Leonardo Araujo dos Santos; Introduction, Localizing with Convolution neural network, How to get bounding box; 6.6.2017.g.; *Single Shot detectors*; https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/single-shot-detectors.html; 12.6.2017.g.

12. Python: Overview; 7.6.2017.g.; *Wikipedia: Python (programming language)*: https://en.wikipedia.org/wiki/Python_(programming_language); 9.6.2017.g.

13. About TensorFlow; proljeće 2017.g.; *TensorFlow*; https://www.tensorflow.org; 9.6.2017.g.

14. Joseph Redmon; How it works; 23.8.2016.g.;*YOLO: Real-Time Object Detection*; https://github.com/pjreddie/darknet/wiki/YOLO:-Real-Time-Object-Detection; 12.6.2017.g.

15. L. Wang, W. Ouyang, X. Wang, H. Lu; Visual Tracking with Fully Convolutional Networks: Proposed Algorithm; ICCV; Santiago, Chile; 2015.g.; str. 3122-3123

16. H. Nam, B. Han; Multi-Domain Network (MDNet), Online Tracking using MDNet; 6.1.2016.g.; *Learning Multi-Domain Convolutional Neural Networks for Visual Tracking*; https://arxiv.org/abs/1510.07945; 13.6.2017.g.

17. A. Rosenbrock; Machine learning, Object Detection; 7.11.2016.g.; *Intersection over Union (IoU) for object detection*; http://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/; 15.6.2017.g.

18. S. Mallick: What is object tracking ?; 13.2.2017.g.; *Object Tracking using OpenCV (C++/Python)*; https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/; 15.6.2017.g.

Duboki konvolucijski modeli za praćenje objekata

Sažetak

Ovaj rad se bavi problemom detekcije i praćenja sa dubokim konvoluciiskim modelima. Objašnjeni su koncepti dubokih konvolucijskih modela, metode učenja te metode detekcije i praćenja objekata. Također su opisani modeli detekcije i praćenja objekata koji su razvijeni u okviru ovog rada. Modeli su evaluirani tako da se mjerila kvaliteta praćenja modela za praćenje u odnosu na model za detekciju, mjerila se kvaliteta praćenja u odnosu na duljinu faze učenja i mjerila se brzina izvođenja ta dva modela. Testovi i faze učenja su provedeni na video snimci nogometne utakmice koja je snimana iz ptičje perspektive gdje su kao objekti za praćenje bili igrači oba tima i sudac.

Ključne riječi: praćenje objekata; detekcija objekata; konvolucijske neuronske mreže; računalni vid; duboko učenje; TensorFlow

Deep convolution models for object tracking

Abstract

This paper examines the problems of object tracking and detection with deep convolutional models. Concepts of deep convolutional networks, learning methods and methods for detection and tracking have been explained as well. Detection and tracking models developed for this paper have also been explained. Models are evaluated with measurement of tracking quality of tracking model against detection model, measurement of tracking quality with different length of learning phase and with measurement of execution performance of those two models. Learning phases and tests have been executed on video file containing the football match which have been filmed in bird's eye perspective where the tracking objects have been players of both teams and the referee.

Keywords: object tracking; object detection; convolutional neural networks; computer vision; deep learning; TensorFlow