SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6992

IZLUČIVANJE KANDIDATA U DVOPROLAZNIM LOKALIZACIJSKIM MODELIMA

Leo Pleše

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6992

IZLUČIVANJE KANDIDATA U DVOPROLAZNIM LOKALIZACIJSKIM MODELIMA

Leo Pleše

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 13. ožujka 2020.

ZAVRŠNI ZADATAK br. 6992

Pristupnik:	**Leo Pleše (0036505670)**
Studij:	Računarstvo
Modul:	Računarska znanost
Mentor:	prof. dr. sc. Siniša Šegvić

Zadatak: **Izlučivanje kandidata u dvoprolaznim lokalizacijskim modelima**

Opis zadatka:

Lokalizacija objekata u prirodnim slikama važan je zadatak računalnog vida s mnogim zanimljivim primjenama. U posljednje vrijeme vrlo zanimljive rezultate postižu dvoprolazni konvolucijski modeli koji u prvom prolazu identificiraju kandidate, a u drugom donose konačnu odluku. Ovaj rad razmatra izvedbu prvog prolaza takvog modela čiji je cilj izlučiti okvire kandidata svih poznatih vrsta objekata. U okviru rada, potrebno je odabrati okvir za automatsku diferencijaciju te upoznati biblioteke za rukovanje matricama i slikama. Proučiti i ukratko opisati postojeće pristupe za lokalizaciju objekata. Predložiti arhitekturu dubokog modela za izlučivanje pravokutnih okvira koji odgovaraju kandidatima objekata. Uhodati postupke učenja modela i validiranja hiperparametara. Primijeniti naučene modele te prikazati i ocijeniti ostvarene rezultate.. Radu priložiti izvorni i izvršni kod razvijenih postupaka, ispitne slijedove i rezultate, uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 12. lipnja 2020.

Zahvaljujem se mentoru prof. dr. sc. Siniši Šegviću na svim smjernicama, savjetima, povjerenju i vremenu koje mi je pružio pod svojim mentorstvom. Također hvala i kolegi Bruni Kovaču na savjetima i podršci pri izradi rada.

Sadržaj

Uvod

Područje umjetne inteligencije (engl. *artificial intelligence*) danas je jedno od najperspektivnijih područja napretka ljudskog znanja i tehnologije. Ono istražuje ljudsku inteligenciju i inteligentna ponašanja koja pokušava simulirati koristeći napredak ostalih grana znanosti, od tehničkih znanosti pa sve do prirodnih znanosti kojima je inicijalno ovo područje bilo i inspirirano. Na isti način, umjetna inteligencija je radi stvaranja čim vjernije slike ljudske inteligencije došla u usku vezu s kognitivnom znanosti [1] što je interdisciplinarna znanosti koja uz elemente računarstva obuhvaća i ključne spoznaje iz psihologije, lingvistike, antropologije i drugih povezanih znanosti ključnih za razumijevanje čovjekova razmišljanja i ponašanja. Umjetna inteligencija tako nastoji uključiti osnovne koncept prirodne inteligencije kroz koncepte percepcije i razumijevanja svijeta oko nas, na koje se zatim nastavlja koncept učenja koje u konačnici omogućava donošenje odluka na temelju ulaznih percepcija iz vanjskog svijeta i već naučenog znanja koje inteligentni agent posjeduje. Isto tako, važne značajke koje svaki inteligentni agent mora imati kako bi učinkovito mogao prolaziti kroz te korake kako bi riješio pojedini problem jesu mogućnost planiranja i optimizacije odlučivanja kako bi u realnoj situaciji s ograničenim resursima u ograničenom vremenu mogao donijeti odluke koje čim više ispunjavaju njegove unutarnje predefinirane ciljeve. Prema opisanome, jasno je da je cilj umjetne inteligencije stvarati ponašanja agenata u realnim uvjetima tako da se u njih ugrade elementi inteligencije, te znanje, spoznaje, razumijevanja jezika i učenje koji su jednako važno kao i inteligencija. Kako ima mnoštvo aspekata inteligencije, postoji i mnogo područja umjetne inteligencije – od obrade prirodnih jezika, metoda pretraživanja prostora stanja, sustava temeljenih na činjenicama i pravilima, dokazivanja teorema u okviru logike, robotike pa sve do strojnog učenja i računalnog vida koja obuhvaća ovaj rad.

Strojno učenje (engl. *machine learning*) općenito je ekvivalent učenju i ono je vrsta programiranja računala gdje se učenje temeljeno na ulaznim podacima i stečenom iskustvu odnosno znanju optimizira kako bi se na kraju izvođenja implementiranog algoritma dobio model koji što bolje generalizira znanje naučeno u tom procesu. Nadalje, računalni vid (engl. *computer vision*) proučava razumijevanje slika i pomičnih slika i pritom se primarno bavi problemima klasifikacije, lokalizacije i detekcije odnosno segmentacije objekata na

slikama. Kako bismo dobili potpunu spoznaju o sadržaju slike, treba klasificirati objekte na slici što znači utvrditi koje su klase objekata prisutne na njoj, uz to lokalizirati i detektirati objekte što znači odrediti poziciju i okvir koji pristaje uz objekt kako bi se odredile granice područja objekta na slici, te na kraju segmentirati objekte što znači odrediti koji elementi digitalne slike pripadaju kojem objektu.

U ovom radu bavimo se modelima koji u dva prolaza kroz slojeve umjetne neuronske mreže lokaliziraju i prepoznaju objekte na slici. Model koji obuhvaća lokalizaciju objekata koji smo detaljno istražili i implementirali je RPN (engl. *Region Proposal Network*), mreža za predlaganje regija. Model RPN je pak elementarni dio modela Mask-RCNN čije smo sastavne elemente istražili kako bismo dobili širu sliku načina rada cijelog jednog sustava za segmentaciju objekata koja je konačni cilj razumijevanja slike.

Mask-RCNN [2] je radni okvir za segmentaciju objekata koji provodi klasifikaciju, lokalizaciju, detekciju i segmentaciju objekata. Mask-RCNN je temeljen na modelu za detekciju objekata Faster-RCNN [3] koji je uveo RPN na način da uz granu za predviđanje klase objekta ima i granu za predviđanje odnosno predlaganje interesnih regija koje potencijalno sadrže objekt i koje nisu fiksne već se predviđaju pomaci okvira kako bi se bolje prilagodili točnom okviru pojedinog objekta. Radi izlučivanja okvira kandidata (engl. *proposals*) koji prekrivaju objekte RPN ima dvije grane kojima predviđa istovremeno okvire tj. opisane pravokutnike (engl. *bounding boxes*) objekata i vjerojatnost (tzv. engl. *objectness score*) da se u određenom okviru (engl. *anchor*) nalazi objekt. Mask-RCNN je proširio Faster-RCNN dodavši granu mreže za predviđanje maske objekta radi segmentacije objekata. Faster-RCNN temeljen je dijelom na modelu Fast-RCNN. Fast-RCNN [4] je riješio jedan od glavnih problema R-CNN-a gdje je višerazinsko učenje vjerojatnosti klasa objekata i okvira objekata zamijenio jednorazinskim učenjem gdje se istovremeno dobivaju izlazi klasifikatora softmax i regresora okvira. RCNN [5] (engl. *Region-based Convolutional Neural Network*), od kojeg je počeo ovaj cijeli lanac novih inovativnih metoda lokalizacije i detekcije odnosno segmentacije objekata, kombinirao je ideju predlaganja regija s klasičnim konvolucijskim mrežama.

Pregledavši najbitnije napretke u području lokalizacije objekata, istražili smo arhitekturu i funkcioniranje osnovnih komponenata Mask-RCNN-a i implementirali model RPN koji smo ispitali učenjem nad više slika pojedinačno kako bismo se vizualno uvjerili u kvalitetu kojom RPN provodi lokalizaciju i sukladno tome označava okvire nad objektima.

1. Korišteni algoritmi i matematički aparat

Prije svega, kako bismo razumjeli algoritme dubokog učenja koje smo koristili u ovom radu i matematičku pozadinu na kojima se oni zasnivaju, dat ćemo pregled konteksta u kojem se duboko učenje nalazi, počevši od područja umjetne inteligencije odnosno strojnog učenja. Zatim ćemo dati pregled osnova strojnog i potom dubokog učenja s naglaskom na algoritmima korištenim u svrhu implementacije modela RPN.

1.1. Duboko učenje u kontekstu umjetne inteligencije

Umjetna inteligencija, kako je već opisano, područje je koje obuhvaća širok spektar primjena koje imaju zajednički cilj – oblikovati inteligentne strojeve koji bi razumjeli svijet oko nas, automatizirali pojedine procese i dr. Jedan od prvih pokušaja u tom smjeru bio je pristup umjetnoj inteligenciji kao bazi znanja (engl. *knowledge base*). U tom pristupu je računalo prepušteno automatsko logičko zaključivanje – inteligentni stroj na temelju danih činjenica (engl. *facts*) iz baze znanja i pravila zaključivanja (engl. *inference rules*) zapisanih u formalnom jeziku izvodi zaključke koje zatim dodaje kao nove činjenice u svoju bazu znanja. Ovaj pristup je, međutim, imao jedan značajan nedostatak jer nije omogućavao sustavu da usvaja nova znanja koja ne može steći isključivo iz činjenica inicijalno mu danih na raspolaganje. Stoga se javila ideja koja je bila korak dalje prema rješavanju tog problema: treba omogućiti sustavu pronalaženju uzoraka iz sirovih ulaznih podataka upravo kako bi inteligentni sustav mogao razviti višu razinu inteligencije u smislu usvajanja novih znanja. To je ujedno temeljna ideja i polazišna točka jedne od ključnih primjena umjetne inteligencije – strojnog učenja.

Korak dalje je uključenje reprezentacije podataka koji imaju određena svojstva (engl. *features*) u okviru reprezentacijskog učenja gdje sustav osim preslikavanja unaprijed dobivene reprezentacije u izlaz također uči i samu reprezentaciju podataka. To je važno radi automatizacije odabira prikladnog skupa svojstava što istovremeno smanjuje ulogu čovjeka te taj odabir čini učinkovitijim. Na kraju, iako je tako postavljen sustav učinkovit u razrješavanju pojedinih reprezentacija ulaznih podataka, on nije u stanju dovoljno dobro uključiti sve faktore varijacije (engl. *factors of variation*) koji su često velika smetnja u dobivanju kvalitetne reprezentacije. Na primjeru učenja prepoznavanja objekata na

digitalnim slikama, neki od primjera faktora varijacije su pozicija, obasjanost i boja objekta te kut upada svjetlosti na objekt i kut između kamere i objekta.

Taj glavni problem rješava duboko učenje (engl. *deep learning*) tako da složene reprezentacije razlaže na jednostavnije tj. pomoću više jednostavnijih koncepata gradi one složenije. Na taj način, duboki modeli nam omogućuju izgradnju kompleksnih inteligentnih sustava koji će biti u stanju stvoriti i vrlo kompleksne reprezentacije ulaznih podataka. U slučaju računalnog vida, primjer jednostavnih koncepata su rubovi objekta koji se mogu odrediti iz svjetline susjednih piksela, iz kojih se njihovim grupiranjem gradi složeniji koncept obrisa objekata i kuteva između njih, iz kojih se dalje grade dijelovi objekta i na kraju kao najsloženija apstrakcija je sam objekt koji se prepoznaje kao cjelina. Kako bi se omogućilo to postupno stvaranje sve složenijih reprezentacija podataka, grade se duboki modeli. Pritom dubinu modela možemo mjeriti na dva glavna načina: kao broj nelinearnih transformacija koje naš program provodi kako bi iz ulaza dobio izlaze modela ili kao dubinu grafa koji predstavlja međupovezanost koncepata uključenih u izračun.

Konačno, važno je napomenuti kako postoje i dva glavna pristupa razumijevanju dubokog učenja. Prvi, tradicionalni pristup inspiriran je biologijom živčanog sustava gdje neurone modeliraju umjetni neuroni (engl. *artificial neurons*) u umjetnim neuronskim mrežama (engl. *artificial neural networks*), osnovne računske jedinice koje se slažu u složene neuronske mreže. Pritom se posebice govori o slojevitim mrežama (engl. *layered networks*) koje imaju sloj ulaza (engl. *input layer*), sloj izlaza (engl. *output layer*) te obično jedan ili više skrivenih slojeva (engl. *hidden layers*) gdje se većim brojem skrivenih slojeva može izlučiti više složenijih svojstava ulaznih podataka.

Međutim, danas je dominantnim pristupom postao onaj intuitivniji programerima koji se zasniva na shvaćanju dubokih modela kao diferencijabilnih transformacija izraženih računalnim programima. Diferencijabilno programiranje [6] (engl. *differentiable programming*) je taj novi pristup dubokim modelima tj. novo shvaćanje programiranja u cijelosti gdje inteligentni strojevi koriste duboke modele kako bi rješavali sve naprednije probleme zahvaljujući mogućnosti učinkovitog učenja i optimizacije (engl. *software 2.0 stack* [7]). Taj pristup zahtijeva da sve funkcije s kojima se radi budu kontinuirane i diferencijabilne po ulaznim parametrima. Kako bi funkcije bile diferencijabilne, same operacije koje se izvode unutar nje trebaju biti također diferencijabilne – takve osnovne građevne jedinice diferencijabilnog modela su linearne kombinacije što su težinske sume koje se opet mogu kombinirati u nove linearne kombinacije. Kako bismo u sustav uveli

nelinearnost, uvodimo diferencijabilne funkcije aktivacije. Uz sve navedeno, kako bi duboki modeli dobili mogućnost kontrole toka diferencijabilnog programa, postoje i diferencijabilna grananja zasnovana na funkciji sigmoida gdje se u prvom slučaju kao težina uz prvu funkciju uzima izlaz sigmoida, a težina uz drugu funkciju je ta vrijednost oduzeta od jedan (kako se sigmoid kreće od 0 do 1). Slično, postoje diferencijabilne petlje čija se memorija zasniva na povratnim neuronskim mrežama. Također, postoji i niz drugih diferencijalnih operatora poput operatora konvolucije koji je ključan kod modela za topološki organizirane podatke poput slika, govora ili prirodnog jezika.

Naposljetku, duboko učenje je posebna vrsta strojnog učenja koja omogućuje reprezentaciju svijeta kao hijerarhiju složenijih koncepata definiranih putem onih jednostavnijih. Stoga objasnimo prvo temeljne koncepte strojnog učenja.

1.2. Strojno učenje

Prije svega, tri su osnovna koncepta strojnog učenja: ulazni podaci, algoritam i model.

1.2.1. Osnovni koncepti

1.2.1.1 Skupovi ulaznih podataka

Kako bismo učili model, a zatim ga ispitali, trebamo imati dovoljan skup prikladno odabranih podataka koji vjerno reprezentiraju podatke koji će biti ulaz našem model jednom kad ga isporučimo. Stoga trebamo prikupiti takve podatke i podijeliti ih unaprijed u tri skupa: skup za učenje odnosno trening (engl. *training set*), skup za provjeru odnosno validaciju (engl. *validation set*) i skup za ispitivanje odnosno testiranje (engl. *test set*). Pritom je nekoliko uobičajenih omjera podjele dostupnih ulaznih podataka u ta tri skupa kao npr. 70% u skup za učenje, 15% u skup za validaciju i 15% u skup za testiranje.

Na skupu za učenje model uči parametre npr. težine (engl. *weights*) koji su za njega definirani. Model se pritom uči najčešće u epohama (engl. *epochs*) gdje je jedna epoha jedan prolaz kroz sve primjere (engl. *examples*) iz skupa za učenje. Ovisno o tome u kojem trenutku odnosno koliko često se mijenjaju parametri modela tijekom učenja, postoje tri tipa učenja [8]: pojedinačno (engl. *on-line*) gdje se parametri mijenjaju nakon svakog primjera, učenje u mini-grupama (engl. *mini-batches*) nakon fiksnog predodređenog broja primjera i grupno učenje (engl. *batch*) nakon prolaza kroz sve primjere iz skupa za učenje.

Na skupu za validaciju određujemo optimalnu složenost modela. Validacija modela se provodi na kraju epohe u grupama validacijskih primjera u određenom broju iteracija. U validaciji uspješnost modela mjerimo prikladnom metrikom koja nam pokazuje kolika je „točnost“ odnosno greška modela jer tu grešku ne možemo promatrati na samom skupu za učenje – tamo greška konstantno pada jer naš model zasigurno postaje sve bolji na skupu podataka na kojima uči. Međutim, uspješnost se treba procijeniti na primjerima koje model nije vidio tijekom učenja. Takvi su podaci iz skupa za testiranje, no preporučljivo je dio primjera iz skupa za učenje unaprijed odrediti u primjere skupa za validaciju koje će model tijekom učenja koristiti ne za učenje parametara, već radi pronalaženja optimalnog broja epoha treniranja jer greška na validacijskom skupu od tog trenutka više neće padati nego će rasti. Prema tome, cilj je pronaći minimalnu grešku na validacijskom skupu i u tom trenutku zaustaviti učenje. Upravo opisan problem jedan je od glavnih u strojnom učenju – problem prenaučenosti (engl. *overfitting*) do kojeg dolazi ukoliko nastavimo trenirati model nakon dostignute optimalne složenosti zbog čega složenost postaje veća no što je potrebna što se odražava kako na validacijskom skupu, tako i na skupu za testiranje tj. model nam (kao i od početka) postaje sve uspješniji u predviđanjima na skupu za učenje, no očito na račun uspješnosti na svim drugim neviđenim podacima. To se rješava uvođenjem opisanog validacijskog skupa tehnikom validacije (engl. *validation*) gdje podatke podijelimo na npr. inicijalno 30% u skup za testiranje i 70% u skup za treniranje od kojeg onda uzmemo npr. 30% u skup za validaciju, a ostalih 70% u konačni skup za učenje koji je sada umanjen za primjere iz odabranog validacijskog skupa.

Na skupu za testiranje na kraju provjeravamo generalizacijsku moć modela. Za to odabiremo objektivnu metriku ovisno o prirodi problema koji se rješava tj. koji nam je dio kritičan. Gledamo odnos točnog (engl. *ground truth*) izlaza i predviđanja odnosno predikcije (engl. *prediction*) modela za dani ulaz, što izravno možemo prikazati npr. matricom zabune (engl. *confusion matrix*). Pritom uzimamo u obzir pojedine od sljedeće četiri moguće kategorije tih odnosa: TP (engl. *true positive*) – točni izlaz i predviđanje su pozitivni, FP (engl. *false positive*) – model predviđa pozitivan rezultat iako je točni negativan, TN (engl. *true negative*) – točni izlaz i predviđanje su negativni i FN (engl. *false negative*) – model predviđa negativan rezultat iako je točni pozitivan. Različite su metrike poput točnosti (engl. *accuracy*). preciznosti (engl. *precision*), odziva (engl. *recall*) i dr. koje uzimaju omjer pojedinih kombinacija pojedinih od četiriju navedenih omjera. Primjerice, preciznost kao omjer TP / (TP + FP) nam govori koliko je izlaza pozitivno od

ukupnog broja pozitivnih predikcija i stoga se koristi kad je cijena FP visoka tj. ne želimo da model predviđa da je izlaz pozitivan iako je on ustvari negativan. Primjer primjene preciznosti je detekcija neželjene (engl. *spam*) e-pošte gdje ne želimo da model predvidi da je neka (važna) e-pošta *spam* ukoliko nije.

1.2.1.2 Algoritmi strojnog učenja

Ovisno o problemu odabiremo tip algoritma, a zatim i konkretan algoritam strojnog učenja koji smatramo prikladnim za rješavanjem danog problema. Tri su osnovne kategorije strojnog učenja koje obuhvaćaju srodne kategorije algoritama strojnog učenja: nadzirano, nenadzirano i podržano učenje.

Nadzirano učenje (engl. *supervised learning*) bavi se nalaženjem funkcije koja iz danog ulaza daje dani izlaz. Pritom rješava dvije glavne vrste takvih problema obzirom na kontinuitet izlaza: klasifikaciju (engl. *classification*) u slučaju diskretnog izlaza odnosno regresiju (engl. *regression*) kad je izlaz kontinuiran. Nadzirano učenje koristimo kad za svaki ulaz u skupu za učenje imamo već pridijeljen obilježen izlaz tj. imamo na raspolaganju skup označenih podataka (engl. *labeled data*) u obliku parova (ulaz, željeni izlaz) pa model može uočavati uzorke u povezanosti ulaza s izlazom koji mu je već dan. Model tokom procesa učenja uči povezanost ulaza s danim izlazom radi dobivanja tražene funkcije koja ustvari predstavlja procjenu stvarnog preslikavanja u obliku predikcije modela klasifikatora odnosno regresora.

Nenadzirano učenje (engl. *unsupervised learning*) za razliku od nadziranog nema unaprijed definirane izlaze za dane ulaze, stoga je prvi korak naći uzorke u podacima kako bi se dobile grupe (engl. *clusters*) podataka koji su prema određenim uzorcima slični. Preciznije, nalaženje sličnosti u modernih modela koji uče nenadzirano svodi se na računanje izglednosti podatka i generiranje podataka uzorkovanjem naučene distribucije podataka za učenje. Zahvaljujući tome, nenadzirani modeli se upravo koriste u slučajevima kad je teško označiti sve podatke ili se želi pronaći uzorke u podacima koje bi čovjek puno teže opazio.

Podržano učenje (engl. *reinforcement learning*) je tip strojnog učenja gdje agent prolazi kroz prostor stanja donoseći odluke koje mu donose pozitivnu ili negativnu nagradu (engl. *reward*) s ciljem maksimiziranja kumulativne nagrade. Nagrada mjeri uspješnost akcije agenta u stanju. Podržano učenje zasniva se na Markovljevim lancima kao modelu koji obuhvaća skup stanja i prijelaze iz jednog u drugo stanja s određenom vjerojatnosti, uz to unoseći pojam akcije i nagrade čime se koncept proširuje na Markovljeve procese

odlučivanja (engl. *MDP*, *Markov decision process*) gdje onda sljedeće stanje ne ovisi samo o trenutnom stanju nego i o akciji koju se poduzme u trenutnom stanju.

1.2.1.3 Model strojnog učenja

Kada priredimo podatke i odaberemo i implementiramo algoritam strojnog učenja, pokrenemo algoritam i kao ulaz mu damo prikupljene podatke iz skupa za učenje i validaciju. Nakon procesa učenja dobijemo model koji predstavlja rezultat procesa učenja danim podacima i algoritmom pod određenim postavljenim hiperparametrima. Hiperparametri modela su vrijednosti koje se postavljaju izvana prije učenja prema određenoj heuristici i kojima kontroliramo samo učenje. Ukoliko smo nakon učenja zadovoljni dobivenim modelom, tada je sve u redu, no u protivnom je moguće da trebamo promijeniti određene hiperparametre tj. validirati ih (engl. *hyperparameter validation*) kako bismo nakon sljedećeg učenja dobili drugačiji, potencijalno uspješniji model. Osim što je važno validirati hiperparametre modela, na početku treba identificirati koji su hiperparametri modela i koji će parametri modela biti njima podložni kako u pojedinim slučajevima želimo da se pojedini parametri modela ne mijenjaju (engl. *untrainable parameters*). Tipičan primjer hiperparametra je stopa učenja (engl. *learning rate*) koja određuje koliko često će model mijenjati parametre tokom učenja. Ako je stopa učenja preniska, učenje će predugo trajati, a ako je previsoka, algoritam može promašiti optimalni postav težina te naći tek suboptimalno rješenje. Stoga je on jedan od ključnih hiperparametara u procesu učenja, posebice kod optimizacije modela koju ćemo objasniti.

1.2.2. Osnovne komponente

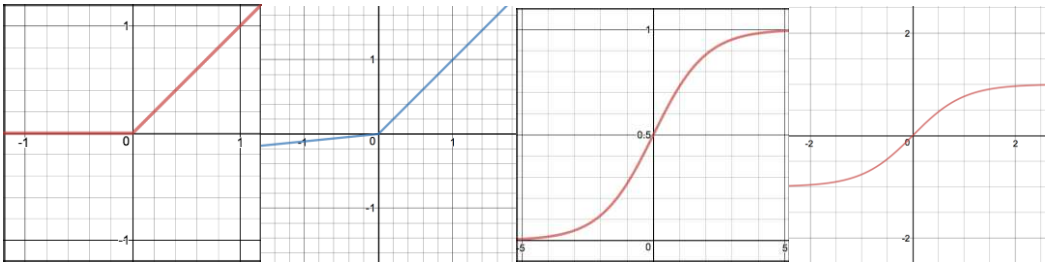
Upoznavši se s osnovnom arhitekturom modela koji se sastoje od ulaznog, skrivenih i izlaznog sloja u pristupu modelima kao slojevima neurona, odnosno modela kao diferencijabilnog programa u kojem ćemo odsad prvenstveno razmatrati svijet posebice dubokih modela, upoznajmo ostale ključne dijelove arhitekture i postupke koji nam daju efikasnije modele.

1.2.2.1 Aktivacija

Funkcije aktivacije (engl. *activation functions*) su nelinearnosti u našem modelu koje transformiraju težinske sume s izlaza u željeni izlaz. Uobičajeno korištene funkcije aktivacije (Slika 1.1) su:

- Softmax: $\sigma(x_i) = e^{xi} / \Sigma_{j=1,...,K} e^{xj}$, i = 1, ..., K – Funkcija koja daje izlaz iz intervala [0, 1] i stoga je prikladna za definiranje vjerojatnosti u višeklasnoj logističkoj regresiji. Diferencijabilna je što je čini općenito pogodnom funkcijom aktivacije.
- Sigmoid/logistička funkcija: $\sigma(x) = 1 / (1 + e^{-x})$ – Varijanta softmax funkcije, također diferencijabilna funkcija koja također daje izlaz iz intervala [0, 1] i prikladna je za definiranje vjerojatnosti u binarnoj logističkoj regresiji.
- ReLU/zglobnica (engl. *Rectified Linear Unit*): f(x) = max(0, x) – Danas sve više korištena aktivacijska funkcija jer je jednostavna i učinkovita, koristi je većina konvolucijskih mreža. Veći problem je jedino što negativne vrijednosti ne preslikava prikladno jer sve mapira u nulu, a kao rješenje tome je osmišljen tzv. *Leaky ReLU* koja linearno opada od 0 prema negativnim vrijednostima.
- tanh (tangens hiperbolni): $f(x) = 2\sigma(x) - 1$ – Kvalitativno slična sigmoidu, samo u širem intervalu izlaza [-1, 1]. Prednost nad sigmoidom je što se negativne vrijednosti preslikavaju u negativne, a ne blizu nule kao što je slučaj kod sigmoida.

U ovom radu korištena je aktivacija softmax radi dobivanja vrijednosti klasifikacija kao klasifikacijske mjere da je područje objekt odnosno pozadina.



Slika 1.1. Funkcije aktivacije – slijeva nadesno: *ReLU*, *Leaky ReLU*, *sigmoid*, *tanh* [9].

1.2.2.2 Gubitak

Funkcije gubitka (engl. *loss functions*) mjera su uspješnosti predviđanja modela. One računaju pogrešku (engl. *error*) koja se definira kao razlika između predviđenog i točnog izlaza. Ovisno o tipu problema odabire se prikladna funkcija gubitka.

U regresiji primjer funkcije gubitka je srednja apsolutna pogreška tj. L1 gubitak (engl. *MAE*, *Mean Absolute Error*, *L1 Loss*) koja računa prosjek apsolutnih vrijednosti odstupanja predviđenih od točnih izlaza. Njegova prednost je otpornost na stršeće vrijednosti budući da uzima isključivo apsolutnu razliku. Drugi tipičan gubitak je srednja kvadratna pogreška tj. L2 gubitak (engl. *MSE*, *Mean Square Error*, *L2 Loss*) koji računa prosjek kvadrata odstupanja. Kod njega stršeće vrijednosti koje su dalje od točne

vrijednosti doprinose kvadratno većim gubitkom što je puno jača kazna nego isključivo apsolutni iznos razlike. L2 gubitak je posebno pogodan za deriviranje pa tako i za izračun gradijenata koji je ključan dio unatražnog prolaza u učenju modela. Naposljetku, kao kombinacija gubitaka L1 i L2 postoji glatki-L1 gubitak (engl. *smooth L1 loss*) koji je za vrijednosti po apsolutnom iznosu manje od 1 zapravo L2 gubitak, a inače L1, čime je izbjegnut „šiljak" u ishodištu svojstven gubitku L1.

U klasifikaciji uobičajena funkcija gubitka je logistički gubitak odnosno gubitak unakrsne entropije (engl. *cross entropy loss*) u kojem se gubitak povećava s povećanjem razlike između predviđene i točne vjerojatnosti klase. Računa se kao negativna suma logaritama vjerojatnosti klasa gdje su u sumu uključeni samo članovi za koje je klasifikacija za promatrani primjer točna. Drugi klasični klasifikacijski gubitak je gubitak zglobnice (engl. *hinge loss*) kod koje točna klasifikacijska mjera treba biti veća od klasifikacijskih mjera kojima je podatak svrstan u sve ostale (netočne) klase. Za razliku od logističkog gubitka koji se zasniva na procjeni najveće izglednosti (engl. *Maximum Likelihood Estimation*, *MLE*) kojom se maksimizira izglednost parametara modela, gubitak zglobnice maksimizira marginu između decizijske granice i podataka radi točne i pouzdane klasifikacije.

U ovom radu korištena su dva gubitka: gubitak unakrsne entropije za predviđanje klase objekta – klasa objekta i klasa pozadine, i glatki-L1 gubitak koji je posebno koristan u problemima detekcije objekata u modelima Fast, Faster i Mask RCNN pa smo ga i mi koristili kao gubitak kod predviđanja okvira objekata modelom RPN.

1.2.2.3 Regularizacija

Regularizacija (engl. *regularization*) obuhvaća niz tehnika kojima se pokušava ublažiti i spriječiti jedan od najvećih problema u učenju modela u strojnom učenju – prenaučenost. Ideja regularizacije je spriječiti prekomjeran rast parametara modela radi izbjegavanja prenaučenosti odnosno radi postizanja veće generalizacijske moći modela. Time je ona povezana s već opisanom idejom validacije učenja na validacijskom skupu.

Regularizacija unosi dodatan član u izrazu za ukupni gubitak koji je u osnovnom obliku jednak iznosu odabrane funkcije gubitka. Regularizacijski član ima oblik sume koja se množi faktorom λ kojim se regulira jačina regularizacije. Izraz za gubitak treba minimizirati, no gubitak će biti veći što je regularizacijski član veći, a veći gubitak znači lošije predviđanje modela na skupu za učenje. Iz toga je očito da treba odabrati dovoljno

velik faktor λ kako bi se spriječila prenaučenost, ali istovremeno i podnaučenost ako je faktor premali.

Dva osnovna primjera regularizacije su: L1 regularizacija (engl. *lasso regression*) koja regularizacijski faktor λ množi sumom apsolutnih vrijednosti parametara modela, najčešće težina, te L2 regularizacija (engl. ridge regression) gdje se taj faktor množi sumom kvadrata težina.

Na kraju, osim spomenutih klasičnih tehnika regularizacije poput navedenih, postoje i ostale koje se ne temelje na dodavanju regularizacijskog člana ukupnom gubitku, već na drugi način rješavaju problem prenaučenosti. Jedna od takvih regularizacijskih tehnika je rano zaustavljanje (engl. *early stopping*). Glavna ideja ranog zaustavljanja je da se učenje modela prekine u trenutku kad model počinje biti prenaučen.

1.2.2.4 Optimizacija

Cilj optimizacije je minimizirati funkciju gubitka. Gradijentni spust je općeniti optimizacijski algoritam koji traži lokalni minimum diferencijabilne funkcije, stoga se kreće u smjeru negativnog gradijenta konveksne funkcije gubitka. Postoje mnogi optimizatori poput SGD-a, Adama, AdaGrada i dr.

Osnovni optimizacijski algoritam je stohastički gradijentni spust (engl. *SGD*, *stochastic gradient descent*). Konceptualno, stohastički gradijentni spust inicijalizira parametre na slučajne vrijednosti, izračuna gradijente svakog od parametara obzirom na funkciju gubitka i primijeni izračunate gradijente na trenutne vrijednosti. To se ponavlja sve dok se ne nađe lokalni minimum. Parametri modela se obično ažuriraju na kraju svake iteracije učenja, a koliko brzo će parametri konvergirati prema minimumu ovisi o stopi učenja. Nova vrijednost parametra x' će biti trenutna x umanjena za vrijednost gradijenta dx pomnoženog stopom učenja η (izraz (1)).

$$x' = x - \eta \cdot dx \tag{1}$$

U ovakvoj početnoj verziji SGD-a može doći do problema lokalnog optimuma (engl. *local optima problem*) u točkama sedla gdje je gradijent 0 pa SGD pronalazi minimum, no umjesto optimalnog rješenja u globalnom optimumu nađe tek suboptimalno rješenje za koje gubitak nije minimalan.

Opisan model ažuriranja parametara je načelno jednostavan i usprkos problemu lokalnih optimuma u mnogim slučajevima učinkovito primjenjiv, no u dubokim modelima se javlja dodatan problem velikog broja parametara koji se trebaju ažurirati. Različiti parametri imaju različit doprinos funkciji gubitka i stoga gradijentni spust parametre ažurira različitim brzinama zbog čega se događa da se neki parametri puno povoljnije ažuriraju od drugih zbog čega dolazi do velikih oscilacija gradijentnog spusta.

Kako bi brže konvergirao tj. kako bi gradijenti brže išli u pravom smjeru, SGD-u se dodaje zalet (engl. *momentum*). Pritom se izrazu kojim se ažurira trenutni parametar ($-\eta \cdot dx$) pribraja umnožak tzv. faktora trenja (engl. *friction*) μ i zadnje ažurirane vrijednosti parametra brzine (eng. *velocity*) v (izraz (2)) [10].

$$v' = -\eta \cdot dx + \mu \cdot v \tag{2}$$

U ovom radu korišten je SGD s momentom kako se SGD obično koristi u detekciji objekata, a moment je korišten radi brže konvergencije i namješten je kao hiperparametar na vrijednost 0.9 koja se eksperimentalno pokazalo učinkovitom.

1.2.2.5 Algoritam propagacije greške unatrag

Algoritam propagacije greške unatrag (engl. *backpropagation algorithm*, skraćeno *backprop*) je osnova i za optimizaciju koja ažurira parametre pomoću izračuna gradijenta. Njime računamo gradijente parametara obzirom na funkciju gubitka u prolazu mrežom unatrag (engl. *backward pass*) nakon što smo unaprijednim prolazom mreže (engl. *forward pass*) dobili redom izlaze svih neurona svih slojeva našeg dubokog modela. U prolazu unatrag računamo greške neurona (gradijenti funkcije gubitka po svakom parametru) i iskoristimo ih da dobijemo vrijednosti kojima se parametri ažuriraju. Pritom se koristimo pravilom ulančavanja kod računanja parcijalnih derivacija koje konkretno koristimo kako bismo dobili parcijalnu derivaciju funkcije gubitka dL po određenoj težini $w_{i,j}$ koja spaja neuron i iz trenutnog s neuronom j iz sljedećeg sloja. Ovdje prvo računamo parcijalnu derivaciju gubitka po izlazu s_j iz neurona j za svaki uzorak s iz skupa D i zatim parcijalnu derivaciju tog izlaza po težini $w_{i,j}$. Te dvije dobivene parcijalne derivacije pomnožimo i dobijemo željenu parcijalnu derivaciju funkcije gubitka L po danoj težini $w_{i,j}$ (izraz (3)).

$$\frac{\partial L}{\partial w_{i,j}} = \frac{\partial L}{\partial s_j} \frac{\partial s_j}{\partial w_{i,j}} \tag{3}$$

Prvo se računaju gradijenti izlaznog sloja kao umnožak derivacije funkcije aktivacije (npr. sigmoid) i razlike očekivanog $t_{s,i}$ i dobivenog $o_{s,i}$ izlaza svakog neurona i za svaki uzorak s (izraz (4) [11]) iz skupa D. Zatim se računaju gradijenti svih slojeva redom od zadnjeg prema prvom tako što se gradijent aktivacije trenutnog sloja pomnoži sumom umnožaka težina kojima je trenutni sloj povezan s idućim $w_{i,d}$ i pripadnih izračunatih gradijenata neurona u sljedećem sloju $\delta_d^{(k+1)}$ koji su bili izračunati u prethodnoj iteraciji (izraz (5) [11]). Konačno, težina između neurona i i j se ažurira proporcionalno stopi učenja η, izlazu y_i neurona i i gradijentu tj. pogrešci neurona j koji slijedi iza neurona i (izraz (6) [11]).

$$\delta_i^K = o_{s,i}(1-o_{s,i})(t_{s,i}-o_{s,i})\delta_i, \quad \forall s,\ s \in D \tag{4}$$

$$\delta_i^{(k)} = y_i^{(k)}(1-y_i^{(k)}) \sum_{d \in downstream} w_{i,d}\delta_d^{(k+1)} \tag{5}$$

$$w_{i,j}^{(k)} \leftarrow w_{i,j}^{(k)} + \eta y_i^{(k)} \delta_j^{(k+1)} \tag{6}$$

Zahvaljujući ovoj rekurzivnoj prirodi algoritma pojedini gradijenti funkcije gubitka s obzirom na svaki parametar se samo jednom računaju, a zatim u svakom sljedećem sloju se koriste već izračunate parcijalne derivacije funkcije gubitka po aktivacijama primjenjujući lančano pravilo diferenciranja na prethodno opisan način. Upravo to algoritam *backprop* čini računski visoko učinkovitim, stoga se on koristi u radnim okvirima za automatsku diferencijaciju poput PyTorcha koji smo odabrali.

2. Opis metode

Glavna metoda kojom smo u ovom radu pristupili problemu detekcije objekata okvirima je metoda RPN-a. Također, opisat ćemo ukratko i metode drugih modela koje koristimo u našem modelu RPN-a. Pritom ćemo se referirati na dva glavna dijela arhitekture modela Mask-RCNN: kralješnicu (engl. *backbone*) modela kao modul za izlučivanje značajki te glavnu mrežu za prepoznavanje opisanih pravokutnika (engl. *network head*) koja provodi klasifikaciju okvira u klasu objekt/pozadina i regresiju koordinata vrhova okvira kojom se dobivaju potrebni pomaci okvira i predviđanje maske (engl. *mask prediction*) koja se primjenjuje na svaku regiju interesa pojedinačno.

2.1. Kralješnica modela

Kao kralješnicu modela odabrali smo model ResNet-34. Kako su modeli ResNet konvolucijske mreže (engl. *convolutional networks*) koje su poseban tip dubokih neuronskih mreža (engl. *deep neural networks*), objasnimo prvo osnovne koncepte konvolucije kao operacije koji se primjenjuju i u ovoj arhitekturi.

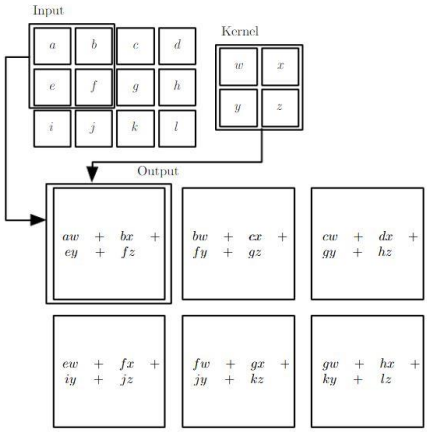
2.1.1. Osnovni koncepti konvolucije

Konvolucija (engl. *convolution*) je funkcija koja se dobiva kao rezultat operacije f * g nad dvjema funkcijama f i g kojom se izražava iznos preklapanja jedne funkcije g dok „prelazi“ preko druge funkcije f gdje je „prelaženje“ jedne funkcije preko druge zapravo njena translacija najčešće u smjeru osi x. Tako dobiven rezultat je integralna funkcija po parametru t gdje je konvolucija obavljena na intervalu [0, t] prema izrazu (7).

$$[f * g](t) = \int_0^t f(\tau)\, g(t-\tau)\, d\tau \tag{7}$$

Konvolucija u kontekstu računalnog vida najčešće je 2D konvolucija nad slikama kao 2D strukturama podataka tj. poljima piksela, no također postoji i 3D konvolucija koja nalazi svoje primjene u obradi 3D struktura podataka gdje je cilj dobiti volumetrijski prikaz strukture npr. u 3D ultrazvuku ili pak za rekonstrukciju 3D struktura kao u [12].

U našem slučaju, kako radimo sa slikama, na ulazu u konvolucijsku mrežu imamo matrice slika (engl. *image matrix*) koje se konvoluiraju (engl. *convolve*) matricama jezgri tj. filtera konvolucije (engl. *convolution kernel*, *filter*). Bit konvolucije jest izlučivanje značajki (engl. *feature extraction*) ulaznih podataka npr. slike kako bi se dobila semantički vrijedna reprezentacija slike u obliku mape značajki (engl. *feature map*) koja se dobiva operacijom konvolucije. Koje će se značajke slike ekstrahirati i time spremiti u mapi značajki, ovisi o dizajnu jezgre konvolucije određenih dimenzija i elemenata te pomaku tj. koraku (engl. *stride*) kao broju elemenata matrice odnosno piksela ulazne slike za koji će se ta jezgra pomicati horizontalno odnosno vertikalno nakon pojedine provedene konvolucije. Ilustracija konvolucije dana je Slika 2.1.



Slika 2.1. 2D konvolucija: jezgra (desno) konvoluira ulazom (lijevo) i dobivamo izlaz (dolje) [13].

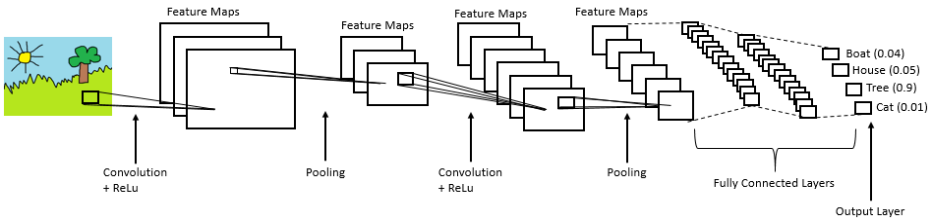
Sljedeći važan koncept koji treba spomenuti je nadopuna ulazne matrice najčešće nulama (engl. *zero-padding*). Taj se postupak radi kad je važno da se u reprezentaciji koju ćemo dobiti mapom značajki sačuvaju i semantički vrijedni rubovi slike koji bi inače bili zastupljeni manje nego pikseli u sredini slike jer bi ih jezgra konvolucije prošla manji broj puta što znači da bi sudjelovali u manje konvolucija. Ponekad je prihvatljivo da se ne radi nadopuna pa se dio piksela ulazne slike ili odbacuje ili nije toliko reprezentiran u mapi značajki (engl. *valid padding*), no često je ona nezaobilazna (engl. *same padding*) što će biti u većini konvolucijskih slojeva naše mreže gdje nam je svaki piksel ulazne slike važan.

Nakon opcionalne primjene nadopune primjenjuje se funkcija aktivacije, najčešće nelinearna npr. ReLU kojom se unosi nelinearnost u konvolucijsku mrežu. Postoji opcija primjene druge aktivacije poput sigmoida, u našem slučaju uzimamo ReLU i softmax.

Zadnji važan koncept je sažimanje (engl. *pooling*). Sažimanje se koristi kako bi se smanjio broj parametara i time dimenzije izlaza, no i dalje zadržavajući ključne informacije

dobivene konvolucijom. Pri sažimanju se definira filter sažimanja određenih dimenzija i pomaka te vrsta sumarne statistike kojom ćemo elemente dobivene konvolucijom ulazne slike pokrivene filterom sažimanja reducirati u jedan broj. Moguće statistike prostornog sažimanja kojima poduzorkujemo (engl. *subsample*) izlaz konvolucije su sažimanje maksimumom, prosjekom ili sumom. Jedna od najčešće korištenih tehnika je sažimanje maksimumom (engl. *max pooling*) koju i mi koristimo u radu.

Nakon konvolucijskog sloja u pojedinim slučajevima bi slijedio potpuno povezan npr. za klasifikaciju, no njega u našem slučaju nemamo jer su nam isključivo relevantni konvolucijski slojevi (s nadopunom nulama i sažimanjem maksimumom) kojima izlučujemo značajke u mape značajki. Opća arhitektura konvolucijske mreže prikazana je na Slika 2.2 gdje možemo napomenuti da se u konvolucijskim mrežama koristi uobičajeno više od jednog konvolucijskog sloja gdje se u svakom sljedećem sloju ekstrahiraju sve složenije značajke čime se dobivaju sve apstraktnije reprezentacije u mapama značajki.



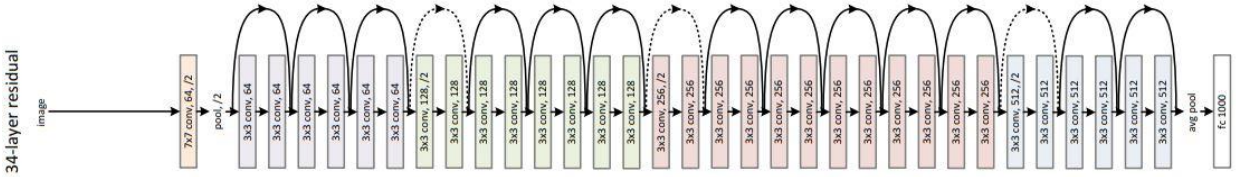
Slika 2.2. Prikaz opće arhitekture konvolucijske neuronske mreže [14].

2.1.2. Konvolucijska arhitektura ResNet-34

Kao što je spomenuto na kraju prethodnog potpoglavlja, prije je bilo općeprihvaćeno da čim više konvolucijskih slojeva imamo tj. čim je konvolucijska duboka neuronska mreža dublja, moći ćemo dobiti bolje reprezentacije ulaznih slika. Međutim, postavilo se pitanje postojanja određene dubine konvolucijske mreže nakon koje performanse počinju opadati. Kako bi se odgovorilo na to pitanje, najprije se trebalo naći rješenje problema nestajućeg odnosno eksplodirajućeg gradijenta (engl. *vanishing, exploding gradient*) gdje se zbog prevelike dubine mreže gradijenti pri izračunu funkcije gubitka smanjuju toliko da padnu blizu nule odnosno povećavaju do beskonačnosti pa čak može doći i do numeričkog podljeva (engl. *underflow*) odnosno preljeva (engl. *overflow*) u računalu u slučaju nestajućeg odnosno eksplodirajućeg gradijenta redom. Taj su problem uglavnom riješili modeli s normalizacijom ulaza i slojeva unutar mreže (engl. *batchnorm*) koji su ujedno stabilizirali gradijent te posljedično učinili optimizaciju (npr. SGD-om) učinkovitijom i

konačno učenje bržim [15]. Rješivši taj problem, pojavio se još jedan problem – problem degradacije točnosti na skupu za učenje: za veću dubinu mreže, točnost se „zasićuje“ i otad naglo počinje opadati. Međutim, neočekivana činjenica je bila da uzrok nije bio u prenaučenosti modela jer se pogreška na skupu za učenje počela odjednom povećavati. Taj je problem primijećen i u radu [16] koji je predstavio arhitekturu ResNet-34 u sklopu novog radnog okvira – dubokog rezidualnog učenja (engl. *deep residual learning*).

Idejna slika kojom je arhitektura ResNeta motivirana je da se ulaz prvog sloja može premostiti izravno na izlaz zadnjeg sloja modela koristeći funkciju identiteta: kako umjetne neuronske mreže mogu učinkovito aproksimirati funkcije, mogu i funkciju identiteta f(x) = x, kojoj onda ako pridodamo inicijalni ulaz mreže x, dobijemo funkciju h(x) = f(x) + x koja je nazvana rezidualna funkcija i na njoj se temelji radni okvir rezidualnog učenja. U okviru rezidualnog učenja funkcija identiteta se koristi kod uvođenja prečica između slojeva koje se mogu izravno koristiti kad su ulaz i izlaz jednakih dimenzija (slučaj 1)), no kad su oni različitih dimenzija (slučaj 2)) onda postoje dvije opcije obje s korakom konvolucije 2: prva je korištenje identiteta bez uvođenja dodatnih parametara pomoću nadopune ulaza nulama kako bi se dimenzije povećale, i druga gdje se konvolucijama 1x1 poklope njihove dimenzije. U radu smo odabrali drugu opciju gdje za slučaj različitog broja ulaznih i izlaznih kanala dodajemo jedan konvolucijski sloj jezgre veličine 1x1. Na Slika 2.3 prikazana je arhitektura ResNet34 gdje su strelicama označene prečice – punom linijom slučaj 1), a iscrtkanom slučaj 2) odnosa dimenzija ulaza i izlaza.



Slika 2.3. ResNet-34: 4 bloka od 3, 4, 6, 3 rezidualne jedinice s po 2 konvolucijska sloja i prečicama [17].

Na kraju, objasnimo na temelju gornje Slika 2.3 34 sloja arhitekture. Prvi konvolucijski sloj *conv1* radi konvoluciju jezgrom 7x7 i pomakom 2 nad ulazna 3 kanala (RGB komponente) i u izlazu daje 64 kanala. Izlaz se zatim nadopuni nulama i sažme maksimum jezgre 3x3 i korakom 2. Nakon toga slijede 32 konvolucijska sloja odnosno 16 tzv. rezidualnih jedinica (engl. *residual unit*, *bottleneck*) gdje svaka ima 2 konvolucijska sloja uz opcionalan treći konvolucijski sloj koji se dodaje kao prečica (engl. *shortcut*) kad su dimenzije ulaza i izlaza različite. Te su rezidualne jedinice organizirane u blokove od po 3, 4, 6 i 3 rezidualne jedinice koje označimo redom s *conv2*, *conv3*, *conv4* i *conv5*. Kao što

vidimo, izlaz jednog bloka ulaz je sljedećeg. Tako blok *conv2* ima 64 kanala dimenzija 128x128, *conv3* ima 128 kanala dimenzija 64x64, *conv4* ima 256 kanala dimenzija 32x32 te *conv5* ima 512 kanala dimenzija 16x16 za primjer ulazne slike 512x512 što je veličina koju na koju inicijalno u našem radu dovodimo ulaznu sliku u naš model RPN. Na prijelazima između ovih 4 bloka se dodatno rade konvolucije tako je npr. na prijelazu iz *conv2* u *conv3* konvolucija kojom iz ulaznih 64 dobivamo izlaznih 128 kanala čime iz bloka *conv2* sa 64 kanala prelazimo u blok *conv3* sa 128 kanala. Na kraju, 34. sloj je potpuno povezan sloj (engl. *fully connected layer*) koji, kao što je spomenuto, nemamo u našem modelu jer nije potreban.

Opišimo u nastavku potpunu arhitekturu kralješnice koju smo koristili u radu.

2.1.3. Arhitektura FPN

Arhitekturu ResNet, konkretno ResNet-34, koristili smo kao osnovni dio kralješnice našeg modela RPN, no kao „proširenu verziju“ te arhitekture možemo promatrati arhitekturu FPN opisanu u radu [18]. U radu smo implementirali kombinaciju te dvije arhitekture: ResNet-FPN, koja je osnova za model RPN-a, a dalje i za Mask-RCNN.

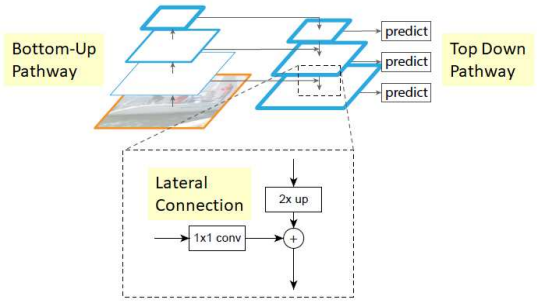
Arhitektura FPN-a uvela je inovativnu ideju u svijet provođenja konvolucije u dubokim konvolucijskim modelima i pokazala se učinkovitijom u detekciji objekata od klasifikacijskih arhitektura bez naduzorkovanja značajki (poput ResNeta) – uz prolaz mrežom odozdo prema gore (engl. *bottom-up*) uvela je dodatan prolaz odozgo prema dolje (engl. *top-down*) tvoreći tzv. piramidalnu arhitekturu. U piramidalnoj arhitekturi imamo piramidalnu hijerarhiju značajki gdje se predikcija provodi nad svakom mapom značajki tj. nakon svake konvolucije. Međutim, općeniti model takve arhitekture ne iskorištava mogućnost ponovnog iskorištavanja prethodno dobivenih mapa značajki radi dobivanja kvalitetnije predikcije. Model FPN (engl. *Feature Pyramid Network*) upravo koristi taj potencijal kombinacijom dvaju prolaza mrežom sljedećim pristupom koji komplementarno uzima dobre strane mapi značajki dobivenih prvim odnosno drugim prolazom. Želimo kombinirati mape značajke sve manjih dimenzija tj. manje rezolucije, no sve semantički sve jače iz drugog prolaza, s mapama koje su semantički slabije, no više rezolucije iz prvog prolaza. FPN to realizira lateralnim vezama (engl. *lateral connections*) između odgovarajućih mapi značajki u ta dva prolaza.

U prolazu odozdo prema gore radi se unaprijedni prolaz konvolucijskom mrežom (u našem slučaju ResNet-34) gdje se izlaz svake razine sprema kako bi se u drugom prolazu iskoristio radi obogaćivanja mapi značajki u tom prolazu.
U prolazu odozgo prema dolje koriste se već dobivene mape aktivacija iz prvog prolaza putem lateralnih veza, čime se istovremeno koriste aktivacijske mape dobivene u prvom prolazu s mapama koje dobivamo redom u ovom prolazu.

Prvo, kako bismo kombinirali mape značajki iz oba prolaza, trebamo izlaze svih blokova (*conv2*, *conv3*, *conv4*, *conv5*) postaviti na isti broj kanala npr. 256. To postižemo konvolucijom 1x1 s nadopunom nulama na izlazima tih blokova. Tako npr. na izlaz iz *conv5* primjenjujemo konvoluciju 1x1 s pomakom 1 s 512 ulaznih i 256 izlaznih kanala (*M5*), na *conv4* tu konvoluciju s 256 ulaznih i izlaznih kanala (*pre_M4_conv*), na izlaz iz *conv3* istu konvoluciju sa 128 ulaza i 256 izlaza (*pre_M3_conv*), na izlaz iz *conv2* tu konvoluciju sa 64 ulaza i 256 izlaza (*pre_M2_conv*), pri čemu sada sve ove konvolucije imaju broj isti broj izlaza pa možemo kombinirati mape.

Drugo, želimo kombinirati mape značajki iz dvaju prolaza kako bismo dobili bogate reprezentacije iz drugog prolaza zajedno s lokaliziranim značajkama iz prvog prolaza. To postižemo zbrajanjem trenutne mape značajki iz *top-down* prolaza s odgovarajućom mapom na istoj razine iz *bottom-up* prolaza, s napomenom da je mapa dobivena trenutnim (*top-down*) prolazom dvostruko manjih dimenzija po širini i visini zbog čega se ona najprije naduzorkuje (engl. *upsample*) faktorom 2 metodom najbližeg susjeda (engl. *nearest neighbor*) nakon čega se tek može provesti navedeno sumiranje (konceptualni prikaz na Slika 2.4). Takvom sumom iz mapa *pre_M4_conv* i naduzorkovane *M5* dobivamo mapu *M4*, iz *pre_M3_conv* i naduzorkovane *M4* mapu *M3*, i iz *pre_M2_conv* i naduzorkovane *M3* mapu *M2*. Još nakon toga trebamo dodatno, kako bismo ublažili neželjeni učinak uslijed naduzorkovanja, provesti konvoluciju 3x3 korakom 1 čime dobivamo konačnu mapu značajki na toj razini mreže. Tako iz mape *M5* tom konvolucijom dobivamo konačnu mapu *P5*, iz *M4* mapu *P4*, iz *M3* mapu *P3* i iz *M2* mapu *P2*. Prema tome, na kraju FPN-a dobivamo mape značajki *P2*, *P3*, *P4* i *P5* koje odgovaraju redom mapa dobivenim kao izlazi iz *bottom-up* prolaza *conv2*, *conv3*, *conv4* i *conv5* koje su bile ulazi *top-down* prolaza i jednakih su dimenzija (redom 128x128, 64x64, 32x32 i 16x16). Te mape značajki upravo su ulaz sljedećem i ključnom modelu – RPN-u. Osim navedenih mapi značajki, u svrhu dobivanja još više potencijalnih kandidata u RPN-u stvorimo još i

mapu *P6* iz mape *P5* primjenjujući sažimanje maksimumom jezgrom 1x1 i korakom 2 čime dobivamo značajke s također 256 kanala no dvostruko manje širine i visine (8x8).



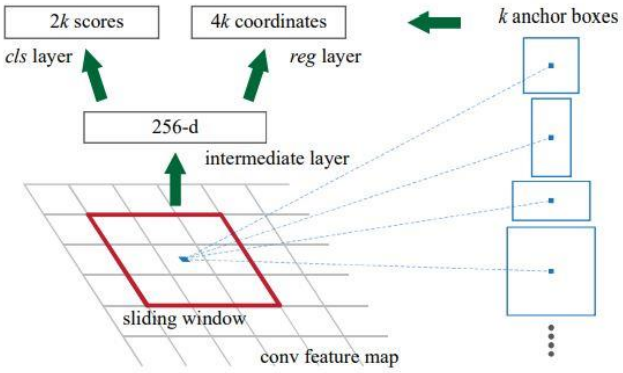
Slika 2.4. Zbroj 2x naduzorkovane mape iz *top-down* i odgovarajuće mape iz *bottom-up* prolaza [19].

2.1.4. Model RPN

RPN je model, jedan od središnjih dijelova modela Mask-RCNN, koji smo najdublje istražili i implementirali. Pritom su mape značajki *P2-P6* dobivene FPN-om na svakoj od razina ulaz u RPN. Koristeći te aktivacijske mape, RPN predviđa dvije stvari: klasifikacijsku mjeru kojom svrstava sadržaj okvira u klasu objekt/pozadina (engl. *„objectness" score*) i na izlazu daje te dvije vjerojatnosti, te pomake središta okvira u smjeru x i y osi kao i potrebne modifikacije širine i visine okvira. Stoga opišimo prvo pojam okvira.

Okvir (engl. *anchor*) je pravokutnik unaprijed određenog odnosa širine i visine čije je središte poravnato s određenim slikovnim elementom. Pritom broj okvira te dimenzije svakog okvira ovise o dimenzijama ulazne slike (u našem slučaju 512x512), omjerima širine i visine (kod nas 3 omjera – 1:1, 1:2 i 2:1) te površinama okvira (odabrali smo okvire površina 64^2, 128^2, 256^2, 512^2 i 1024^2). Navedeni omjeri širine i visine i površine okvira vežu se uz pojedine razine FPN-a čiji je izlaz ulaz u RPN (mape *P2*, *P3*, *P4*, *P5* i *P6*) gdje je površina aktivacijske mape na višoj razini manja pa je broj okvira manji, no ti su okviri veće površine. Tako je na najnižoj razini mapa *P2* najveće dimenzije (128x128) s najviše okvira najmanje površine (64^2), dok je na najvišoj razini mapa P6 najmanje dimenzije (8x8) s najmanje okvira najveće površine (1024^2). Slika 2.5 ilustrira rad i rezultate RPN-a: na svakoj predodređenoj poziciji mape značajki na određenoj razini FPN-a nalazi se *k* okvira predodređenih omjera širina i visina te površine, svaka mapa ima 256 izlaza i za svaki od njih se klasifikatorom predviđa vjerojatnost klase (te dvije vjerojatnosti za *k* okvira daju 2*k* klasifikacijskih mjera odnosno izlaza klasifikatora) i regresorom se predviđa

pomak centra okviru u x i y smjeru te promjena širine i visine okvira (ta 2 pomaka i 2 modifikacije dimenzija okvira su 4 broja za svaki od *k* okvira što daje 4*k* izlaza regresora).



Slika 2.5. Na danoj poziciji mape značajki je *k* okvira: za svaki se predviđa mjera klase objekt/pozadina (2*k* mjera) i promjene centra (x, y) i dimenzija (širine i visine) okvira (4*k* brojeva) [3].

Dobivši sve okvire, prije početka učenja prvi ključni korak nam je još saznati koliko se pojedini generirani okvir preklapa s kojim točnim okvirom (engl. *ground truth bounding box*) i na temelju toga za svaki okvir inicijalno odrediti klasu objekt/pozadina kao i promjenu pozicije centra i dimenzija okvira.

Kad dobijemo te klase i promjene okvira, damo ih RPN-u za učenje. Kao rezultat jedne iteracije učenja dobijemo klasifikacijske mjere i vjerojatnosti za klase objekt/pozadina te pomake okvira nakon čije bi primjene okviri naučeni iz inicijalnih okvira trebali biti bolje prilagođeni točnim okvirima. Zatim izračunamo gubitak klasifikatora i regresora, te izračunavši gradijente provedemo unazadni prolaz algoritmom *backprop* i naposljetku provedemo korak optimizacije modela odabranim optimizatorom SGD. Učenje provedemo u potrebnom broju iteracija te na kraju određene iteracije dobivamo model RPN-a uspješno prilagođen skupu za učenje.
Stoga opišimo prvo što treba napraviti prije učenja, a zatim i arhitekturu i učenje RPN-a.

2.1.4.1 Koraci prije učenja RPN-a

Glavni podatak koji sadrži oznaka (engl. *annotation*) svih primjera slika iz skupa podataka za učenje RPN-a je popis objekata tj. odgovarajućih točnih okvira određenih minimalnom i maksimalnom x i y koordinatom (pozicija gornjeg lijevog i donjeg desnog vrha okvira). Stoga nam je prvi korak saznati koliko se preklapaju točni okvir (dostupan iz oznake) s okvirima koje smo mi generirali na način opisan u uvodnom dijelu ovog poglavlja. U tu svrhu kao mjeru preklapanja tj. prilagođenosti naših generiranih okvira točnom okviru

koristit ćemo mjeru IoU (engl. *Intersection over Union*) koja se računa kao omjer presjeka i unije površina dvaju okvira.

Dobivši matricu preklapanja svakog okvira sa svakim točnim okvirom (u retku koji određuje okvir i stupcu koji određuje točni okvir zapisana je mjera preklapanja tih dvaju okvira), treba odrediti klasu objekta. Prvo, za svaki okvir odredimo s kojim točnim okvirom ima najveću mjeru preklapanja IoU i kolika ona iznosi te ako je veća od predodređenog praga (odabrano 0.6) onda je taj okvir klase 1 tj. klase objekta (pozitivan), ako je manja od predodređenog praga (odabrano 0.3) onda je klase -1 tj. klase pozadine (negativan), a inače (ako je IoU između ove dvije granice) kažemo da je okvir neutralan. Drugo, za svaki točan okvir odredimo s kojim okvirom ima najveću vrijednost IoU i tamo postavimo vrijednost klase 1 (objekt) – to je nužno kako bismo osigurali da u slučaju ako točan okvir nema nijedan okvir koji se s njime preklapa s IoU >= 0.6, izaberemo onaj okvir koji se s njime preklapa s najvećim IoU iako ta vrijednost ne treba biti veća od 0.6. Premda smo kroz ova dva koraka odredili klase svakom okviru, trebamo se još pobrinuti da uravnotežimo odnos okvira klase objekta i pozadine u predodređenom omjeru npr. 1:1 što znači da treba slučajno odabrati okvire klase 1 odnosno -1 da bi u tom slučaju pola okvira bilo pozitivno i pola negativno. Ovo balansiranje je važno jer u protivnom bi okviri klase pozadine bili dominantni što ne bi bilo dobro za proces učenja. Nakon ovog koraka smo odredili klase okvira u -1 odnosno 1 te imamo sve pozitivne i najpozitivnije negativne okvire na kojima ćemo učiti RPN.

Trebamo još odrediti pomake okvira. Stoga uzmemo sve pozitivne okvire i za svaki odredimo točni okvir s kojim ima najveći IoU. Za oba okvira pojedinačno odredimo njihovu širinu (razlika xmin i xmax), visinu (razlika ymin i ymax), (x, y) koordinatu centra (xmin + širina/2, ymin + visina/2) i na temelju toga 4 broja: *dx* – relativan pomak centra okvira po x osi kao omjer razlike x koordinata centara tih okvira i širine samog okvira, *dy* – analogan pomak kao *dx* samo po y-osi, *dw* – logaritam omjera širine točnog okvira i okvira i *dh* – logaritam omjera visine točnog okvira i okvira.

Odredivši klase i pomake okvira priredili smo oznake u prikladnom obliku za učenje.

2.1.4.2 Arhitektura i učenje RPN-a

Arhitektura RPN-a naslanja se na arhitekturu FPN-a jer za svaku razinu FPN-a dobivamo mapu značajki (mape *P2-P6*) i za svaku određujemo klase objekta, klasifikacijske mjere i pomake okvira. Prvi sloj je konvolucijski gdje radimo konvoluciju 3x3 korakom 1 s

nadopunom nulama s 256 ulaznih i izlaznih kanala, te drugi je aktivacija ReLU koja se primjenjuje na izlaz te konvolucije. Taj je dio zajednički za klasifikacijski dio RPN-a koji predviđa klase i klasifikacijske mjere i za regresijski koji predviđa pomake okvira. Zatim se grana na dvije upravo spomenute grane – klasifikacijsku i regresijsku. Klasifikacija se odvija dodatnom konvolucijom 1x1 pomakom 1 s ulaznih 256 kanala i izlaznom broju kanala jednakom dvostrukom broju okvira koji pak odgovara broju omjera stranica okvira koje imamo (u našem slučaju 3 za 3 različita omjera – 1:1, 1:2, 2:1). Izlaz je dvostruki broj okvira jer se za svaki okvir predviđa vjerojatnost da objekt pripada klasi objekta (1) odnosno klasi pozadine (-1). Nakon te konvolucije primjenjuje se još aktivacija softmax kako bi se dobile te dvije vjerojatnosti kao konačni izlaz klasifikatora. Regresija se odvija dodatnom konvolucijom, slično kao i za klasifikaciju jezgrom 1x1 korakom 1, no broj izlaznih kanala je jednak četiri puta uvećanom broju okvira jer se predviđaju 4 broja *dx*, *dy*, *dw*, *dh* opisani na kraju prethodnog poglavlja 2.1.4.1.

Dobivši klasifikacijske mjere okvira za klasu -1/1, odgovarajuće vjerojatnosti i pomake okvira, možemo izračunati klasifikacijski i regresijski gubitak koji onda propagiramo unatražnim prolazom modela, optimiziramo parametre i tako iterativno učimo model. Konkretno, za klasifikaciju koristimo gubitak unakrsne entropije. Pritom ga računamo samo nad objektima klasa -1 i 1 a neutralne okvire ne uračunavamo u ovaj gubitak. Za regresiju računamo gubitak glatki-L1 i to samo nad okvirima klase 1.

2.1.4.3 Zaključivanje RPN-a

Naučivši RPN, treba još objasniti kako RPN provodi zaključivanje (engl. *inference*). Ovdje ćemo opisati koncepte korištene pri zaključivanju, a konkretni rezultati ispitivanja zaključivanja RPN-a dani su u poglavlju 5.

Od kandidata koje smo izlučili RPN-om želimo odabrati one koji najbolje pristaju točnim okvirima objekata na slici. Stoga prvo silazno sortiramo okvire kriterijem najveće klasifikacijske mjere za klasu 1. Zatim na okvire primjenjujemo pomake okvira dobivene učenjem: uzimamo širinu, visinu i položaj centra pravokutnika pomoću kojih odredimo nove vrijednosti istih veličina – x koordinatu centra računamo tako da staroj x koordinati pribrojimo umnožak širine i relativnog pomaka *dx*, analogno za novu y koordinatu centra preko visine i *dy*, a novu širinu okvira odredimo kao prijašnju pomnoženu s exp(*dw*), analogno za novu visinu preko *dh*. Zatim iz nove pozicije centra i dimenzija okvira odredimo točke (xmin, ymin) i (xmax, ymax).

Nakon toga, primijenimo algoritam NMS i kao rezultat uzmemo prvih 1000 kandidata koje njime dobijemo. Algoritam NMS (engl. *Non-maximum Suppression*) na temelju mjere IoU između okvira (kojima su prethodno promijenjene dimenzije i pozicija) odabire okvire na taj način da prvenstvo daje onima s većom klasifikacijskom mjerom (tj. onima koji imaju veću vjerojatnost preklapanja s točnim okvirom objekta), dok ostale okvire manje klasifikacijske mjere koji se preklapaju s tim okvirima veće klasifikacijske mjere s vrijednosti IoU većom od određenog praga (u našem slučaju 0.5) iterativno odbacuje. Treba još reći da još uvijek postoji šansa da i nakon NMS-a ostanu okviri koji izlaze van slike ili pak imaju negativne dimenzije ili sl. jer se uklanjanje neispravnih okvira odvija u kasnijoj fazi Mask-RCNN-a. U svakom slučaju, nakon primjene NMS-a nad kandidatima koje je RPN izlučio dobivamo prilično zadovoljavajuće rezultate koje ocjenjujemo vizualno i mjerom prosječnog odziva na 1000 kandidata (AR1000, engl. *average recall*) usrednjenog preko pragova IoU iz intervala [0.5, 0.95] s korakom 0.05 (za IoU = 0.50, 0.55, …, 0.95). Odabiremo mjeru AR (omjer TP / (TP + FN)) kao mjeru uspješnosti jer nam je najveća cijena FN (engl. *False Negative*) predikcija – ne želimo da RPN predvidi klasu -1 ako je točna 1 tj. da na području nekog okvira model predvidi da nema objekta iako je on pokriven tim okvirom.

3. Programska izvedba i vanjske biblioteke

3.1. Vanjske biblioteke

Od vanjskih biblioteka koje smo koristili u implementaciji modela RPN-a najvažnija je svakako biblioteka *torch*. Ta je biblioteka ključan dio radnog okvira strojnog učenja za automatsku diferencijaciju koji smo koristili – PyTorch. Neke od ostalih poznatijih okvira za automatsku diferencijaciju su npr. TensorFlow, Theano, Keras i Caffe.

Središnji paket koji neprestano koristimo za automatsku diferencijaciju nad tenzorima je *autograd* [20]. Sam pojam tenzora je ključan općenito u okvirima za automatsku diferencijaciju pa tako je i kao osnovni element operacija diferencijacija u PyTorchu izveden klasom *torch.Tensor*. Praćenje operacija nad pojedinim tenzorom započinje postavljanjem njegovog atributa *.requires_grad* na vrijednost *True*, a na kraju pozivom funkcije *.backward()* provodi se automatska diferencijacija tj. automatski se izračunavaju gradijenti definirane funkcije gubitka koji se zatim spreme u atribut *.grad* tenzora. U istom paketu, osim tenzora važna je i funkcija kojom je tenzor stvoren i koja zajedno s pripadnim tenzorom čini graf modela koji razvijamo kroz učenje. Ta je funkcija implementirana klasom *Function*, a za pojedini tenzor zapisuje se u njegovo polje *grad_fn*.

Druga najvažnija biblioteka je NumPy [21] s modulom *numpy* (često označeno skraćeno s *np* pri uvozu biblioteke u vlastiti modul). Ta biblioteka vezana uz jezik Python jedna je od osnovnih biblioteka za učinkovit rad s matricama pa se koristi za složene znanstvene izračune, stoga smo je i mi koristili u tu svrhu. Osnovna klasa koja se pritom koristi je *numpy.ndarray* [22] kao reprezentacija višedimenzionalnog homogenog polja elemenata fiksne veličine. Samo polje *numpy* se pritom stvara funkcijom *array* (npr. iz liste), *zeros* (polje nula) ili *empty* (stvara polje, ali ne mijenja sadržaj alocirane memorije) pri čemu se s *dtype* može specificirati tip podataka u polju. Primjer dvije osnovne operacije koje smo provodili na poljima *numpy* poljima su *np.where(condition[, x, y])* [23] koja na temelju ispunjenosti uvjeta *condition* za pojedini element polja provodi operaciju `x` odnosno `y` ili samo vraća polje s elementima koji zadovoljavaju uvjet ako operacije nisu specificirane, i *np.arange([start,]stop, [step,]dtype=None)* [24] koja vraća polje *numpy* s uniformno raspoređenim vrijednostima od *start* (pretpostavljeno 0) do *stop* s korakom *step*

(pretpostavljen 1) tipa *dtype.* Neke od ostalih funkcija koje smo često koristili su *np.exp*, *np.log*, *np.sqrt*, *np.minimum*, *np.maximum*, *np.random.choice*, *np.argmax* i *np.sum*.

Posljednje, od biblioteka za rad sa slikama koristili smo funkciju *io.imread* iz paketa *skimage* za učitavanje slika, što je dio biblioteke za obradu slika scikit-image [25]. Također, za slike na kojima smo iscrtavali okvire koristili smo: funkciju *Image.open* za otvaranje slike i zatim funkciju *rectangle* za iscrtavanje pravokutnika određenih dimenzija u boji te funkciju *show* za prikaz slike nad objektom klase *ImageDraw.Draw* iz paketa *PIL*, što je dio biblioteke za rad sa slikama Pillow [26].

3.2. Programska izvedba

Model RPN programski smo izveli u programskom jeziku Python korištenjem vanjskih modula opisanih u prethodnom potpoglavlju. Ovdje ćemo dati pregled najvažnijih dijelova koda i ukratko ih objasniti. Kako smo već detaljno opisali teorijsku pozadinu modela i uveli oznake glavnih slojeva arhitekture u poglavlju 2., ovdje ćemo dati kratke dopune tih objašnjenja u smislu konkretne implementacije. Pomoć pri pisanju implementacije bile su implementacije [27] te rad i implementacija priložena uz rad [28] koju je autor ustupio.

3.2.1. Organizacija koda

Kod je organiziran u četiri paketa:

- *dataset* – Sadrži podpaket *VOC2012* s paketima *Annotations* i *JPEGImages* koji sadrže redom anotacije (oznake) i slike iz ispitnog skupa VOC2012 koje se mogu koristiti za učenje RPN-a.
- *rpn_net* – Sadrži glavne module arhitekture RPN-a *backbone.py* i *rpn.py* uz konfiguracijsku datoteku *config.py* i pomoćne funkcije unutar podpaketu *utils* u modulima *anchor_util.py*, *image_util.py* i *xml_util.py*.
- *tests* – Sadrži module za testiranje (*test1.py*, *test2.py*, *test3.py*, *test4.py*, *test5.py*) i izračun metrike za uspješnost modela (*calculate_ar.py*, *calculate_avg_ar.py*) na temelju primjera slika na kojima je model učen popisanih u datoteci *train_list_100.txt* unutar ovog paketa.
- *weights* – Sadrži modele RPN-a koji se generiraju tijekom učenja.

3.2.2. Glavni moduli

U nastavku ćemo dati prikaz najvažnijih dijelova koda i popratna objašnjenja. Zbog preglednosti u slučaju pojedinih modela bit će prikazan samo konstruktor modela (gdje je implementirana arhitektura modela) bez metode *forward* (unaprijedni prolaz) – za detalje pogledati dokumentaciju koda u prilogu radu.

3.2.2.1 Kralješnica Resnet-34 + FPN (modul backbone.py)

Ovaj modul sadrži dvije glavne klase (izvedene iz *torch.nn.Module* – temeljne klase za izvođenje vlastitih modula neuronskih mreža):

- *Resnet34_FPN* – kralješnica RPN-a: model FPN s modelom ResNet-34 kao bazom
- *ResidualBlock* – rezidualna jedinica, osnovna jedinica 32 glavna konvolucijska sloja unutar 4 grupe konvolucijskih slojeva modela ResNet-34 (poglavlje 2.1.2)

Metoda *forward* je metoda klase *torch.nn.Module* koju koristimo za unaprijedni prolaz.

Klasa *ResidualBlock* (Kôd 3.1) predstavlja rezidualnu jedinicu modela ResNet-34 što znači da ima 2 konvolucijska sloja *c1* i *c2* koji rade konvoluciju 3x3 korakom 1. Prije tih konvolucija potrebna je nadopuna nulama za što je implementirana pomoćna klasa *SamePad2d*. Prisutni su i odgovarajući slojevi za normalizaciju mini-grupa (engl. *batch-normalization*) *bn1* i *bn2* koji su predviđeni za slučaj kad će se implementirani model proširiti na treniranje na mini-grupama slika. Nakon njih slijedi još aktivacija ReLU (*relu*). Na kraju, metodom *shortcut_method* se dodaje dodatan konvolucijski sloj koji radi konvoluciju 1x1 u slučaju kad je broj ulaznih različit od broja izlaznih kanala. Izlaz tog dodatnog konvolucijskog sloja (u slučaju različitog broja ulaznih i izlaznih kanala) odnosno ulaz u rezidualnu jedinicu (za jednak broj ulaza i izlaza), pribraja se izlazu zadnje aktivacije ReLU čime se dobiva konačni izlaz jedinice.

Sama klasa se instancira u *__init__* argumentima: *filters* – broj filtera koji je jednak broju ulaznih odnosno izlaznih kanala konvolucija, *kernel_size* – dimenzija jezgre konvolucije (jednaka 3) i *channels_in* i *channels_out* su broj ulaznih i izlaznih kanala jedinice.

```
class ResidualBlock(nn.Module):

    def __init__(self, filters, kernel_size, channels_in, channels_out):
        super().__init__()
```

```
        self.relu = nn.ReLU()

        self.pad1 = SamePad2d(kernel_size=kernel_size, stride=1)
        self.c1 = nn.Conv2d(in_channels=filters, out_channels=filters,
        kernel_size=kernel_size)
        self.bn1 = nn.BatchNorm2d(filters, eps=0.001, momentum=0.01)

        self.pad2 = SamePad2d(kernel_size=kernel_size, stride=1)
        self.c2 = nn.Conv2d(in_channels=filters, out_channels=filters,
        kernel_size=kernel_size)
        self.bn2 = nn.BatchNorm2d(filters, eps=0.001, momentum=0.01)

        self.shortcut = self.shortcut_method(channels_in, channels_out)
        if isinstance(self.shortcut, nn.Conv2d):
            self.pad_shortcut = SamePad2d(kernel_size=1, stride=1)

    def shortcut_method(self, channels_in, channels_out):
        if channels_in != channels_out:
            return nn.Conv2d(in_channels=channels_out,
            out_channels=channels_out, kernel_size=1)
        else:
            return lambda x: x

    def forward(self, x):
        ... # unaprijedni prolaz, izlaz dobiva u varijabli y
        if isinstance(self.shortcut, nn.Conv2d):
            y_add = self.pad_shortcut(x)
            y += self.shortcut(y_add)
        else:
            y += self.shortcut(x)

        y = self.relu(y)

        return y
```

Kôd 3.1. Klasa *ResidualBlock*, reprezentacija rezidualne jedinice modela ResNet-34.

Klasa *Resnet34_FPN* (Kôd 3.2) definira kralješnicu našeg modela RPN – FPN s baznom arhitekturom ResNet-34. Prvi dio je sam ResNet-34: ulazni konvolucijski sloj (konvolucija 7x7 korakom 2 na 3 ulazna (RGB) sa 64 izlazna kanala) *conv1* koji je prije

toga nadopunjen nulama u *pad1* i još sažet maksimumom slojem *pool1*, zatim 4 bloka od redom 3, 4, 6 i 3 rezidualnih jedinica (*block2*, *block3*, *block4*, *block5*) ispred kojih se nalaze konvolucijski slojevi *block3_pre_conv*, *block4_pre_conv* i *block5_pre_conv* koji provode konvoluciju 3x3 korakom 2, na kraju blokova se još primjenjuje aktivacija ReLU. Izlazi zadnje rezidualne jedinice četiriju blokova su *conv2*, *conv3*, *conv4*, *conv5* što čine glavna 32 konvolucijska sloja opisana u poglavlju 2.1.2. (ukupno 3 + 4 + 6 + 3 = 16 rezidualnih jedinica gdje svaka ima 2 konvolucijska sloja *c1* i *c2* (iz klase *ResidualBlock*)). Drugi je dio FPN kao nadogradnja na ResNet-34 – ovdje ćemo se pozvati na iste oznake uvedene u poglavlju 2.1.3.: konvolucije 1x1 korakom 1 za dovođenje izlaza svih blokova na izlaznih 256 kanala (*M5*, *pre_M4_conv*, *pre_M3_conv*, *pre_M2_conv*), konvolucije 3x3 korakom 1 koje su izlaz RPN-a (*P5*, *P4*, *P3*, *P2*) uz *P6* što je sažimanje izlaza iz *P5* maksimumom jezgrom 1x1 korakom 2 koja je također izlaz RPN-a. Pojedini konvolucijski slojevi imaju nadopunu nulama (imena slojeva završavaju na „*_pad*“). Važno je još napomenuti da se iz konstruktora arhitekture ne mogu vidjeti međurezultati dobiveni u *M4*, *M3*, *M2* opisani u 2.1.3., a koji su onda ulaz u konvolucije *P4*, *P3*, *P2*. Radi ilustracije, na isječku Kôd 3.3 prikazano je kako se mapa značajki *M3* dobiva zbrojem izlaza konvolucije 1x1 *pre_M3_conv* nad izlazom *y3* iz 3. bloka *block3* i 2 puta (*scale_factor=2*) naduzorkovane mape *M4* metodom najbližeg susjeda (*mode="nearest"*).

```
class Resnet34_FPN(nn.Module):

    def __init__(self):
        super().__init__()

        self.relu = nn.ReLU()

        self.pad1 = SamePad2d(kernel_size=7, stride=2)
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=64,
        kernel_size=7, stride=2)

        self.bn1 = nn.BatchNorm2d(64, eps=0.001, momentum=0.01)
        self.pool1 = nn.Sequential(SamePad2d(kernel_size=3, stride=2),
        nn.MaxPool2d(kernel_size=3, stride=2))

        self.block2 = [ResidualBlock(64, 3, 64, 64).cuda() for _ in
        range(3)]
        self.conv2 = self.block2[-1].relu
```

```
self.block3_pre_pad = SamePad2d(kernel_size=3, stride=2)
self.block3_pre_conv = nn.Conv2d(in_channels=64,
out_channels=128, kernel_size=3, stride=2)
self.block3 = [ResidualBlock(128, 3, 64, 128).cuda() for _ in
range(4)]
self.conv3 = self.block3[-1].relu

self.block4_pre_pad = SamePad2d(kernel_size=3, stride=2)
self.block4_pre_conv = nn.Conv2d(in_channels=128,
out_channels=256, kernel_size=3, stride=2)
self.block4 = [ResidualBlock(256, 3, 128, 256).cuda() for _ in
range(6)]
self.conv4 = self.block4[-1].relu

self.block5_pre_pad = SamePad2d(kernel_size=3, stride=2)
self.block5_pre_conv = nn.Conv2d(in_channels=256,
out_channels=512, kernel_size=3, stride=2)
self.block5 = [ResidualBlock(512, 3, 256, 512).cuda() for _ in
range(3)]
self.conv5 = self.block5[-1].relu

self.M5_pad = SamePad2d(kernel_size=1, stride=1)
self.M5 = nn.Conv2d(in_channels=512, out_channels=256,
kernel_size=1, stride=1)
self.P5_pad = SamePad2d(kernel_size=3, stride=1)
self.P5 = nn.Conv2d(in_channels=256, out_channels=256,
kernel_size=3, stride=1)

self.P6 = nn.MaxPool2d(kernel_size=1, stride=2)

self.pre_M4_pad = SamePad2d(kernel_size=1, stride=1)
self.pre_M4_conv = nn.Conv2d(in_channels=256, out_channels=256,
kernel_size=1, stride=1)

self.P4_pad = SamePad2d(kernel_size=3, stride=1)
self.P4 = nn.Conv2d(in_channels=256, out_channels=256,
kernel_size=3, stride=1)

self.pre_M3_pad = SamePad2d(kernel_size=1, stride=1)
self.pre_M3_conv = nn.Conv2d(in_channels=128, out_channels=256,
```

```
        kernel_size=1, stride=1)

        self.P3_pad = SamePad2d(kernel_size=3, stride=1)
        self.P3 = nn.Conv2d(in_channels=256, out_channels=256,
        kernel_size=3, stride=1)

        self.pre_M2_pad = SamePad2d(kernel_size=1, stride=1)
        self.pre_M2_conv = nn.Conv2d(in_channels=64, out_channels=256,
        kernel_size=1, stride=1)

        self.P2_pad = SamePad2d(kernel_size=3, stride=1)
        self.P2 = nn.Conv2d(in_channels=256, out_channels=256,
        kernel_size=3, stride=1)
```

Kôd 3.2. Klasa *Resnet34_FPN*, reprezentacija modela ResNet-34 + FPN.

```
        upsampled4 = F.upsample(m4, scale_factor=2, mode="nearest")
        m3 = self.pre_M3_conv(self.pre_M3_pad(y3)) + upsampled4
```

Kôd 3.3. Dobivanje mape značajki M3 zbrojem 2 puta naduzorkovane mape M4 metodom najbližeg susjeda i konvoluiranog izlaza 3. bloka.

3.2.2.2 RPN i učenje RPN-a (modul rpn.py)

Klasa *RPN* (Kôd 3.4) definira model RPN. U konstruktoru prima kralješnicu ResNet-34 + FPN *backbone* i broj okvira na fiksnim pozicijama piksela *anchors_num*. Prvi sloj je zajednička konvolucija 3x3 korakom 1 s 256 ulaza i izlaza *conv* prije nadopunjena nulama, i zatim slijede dvije grane: klasifikacijska – konvolucija 1x1 s *2*anchors_num* (predviđa vjerojatnost da objekt pripada klasi objekta/pozadine) izlaza i zatim aktivacijom *softmax* i regresijska – konvolucija 1x1 s *4*anchors_num* izlaza (predviđa brojeve koji kvantificiraju pomake dx, dy i promjene dimenzija okvira dw, dh). U unaprijednom prolazu ove se operacije obavljaju redom na svakoj razini FPN-a (*P2-P6*).

```
class RPN(nn.Module):

    def __init__(self, backbone, anchors_num):
        super().__init__()

        self.backbone = backbone
        self.anchors_per_pix = anchors_num
        self.conv_pad = SamePad2d(kernel_size=3, stride=1)
```

```
self.conv = nn.Conv2d(in_channels=256, out_channels=256,
kernel_size=3)

self.relu = nn.ReLU()

self.cls_conv = nn.Conv2d(in_channels=256,
out_channels=self.anchors_per_pix * 2, kernel_size=1)

self.cls_softmax = nn.Softmax(dim=2)

self.bb_conv = nn.Conv2d(in_channels=256,
out_channels=self.anchors_per_pix * 4, kernel_size=1)
```

Kôd 3.4. Klasa *RPN*, reprezentacija modela RPN.

Konačno, isječak iz glavne funkcije učenja RPN-a dan je u isječku Kôd 3.5. Model RPN instancira se kralješnicom *Resnet34_FPN* s 3 okvira (3 omjera dimenzija okvira) sa središtima poravnatim s predodređenim pikselima slike. Radi učinkovitijeg izračuna funkcijom *cuda* izračun se prebacuje na grafičku karticu CUDA. Inicijalizira se optimizator SGD s parametrima stope učenja *lr=0.001*, momentom *momentum=0.9* i faktorom L2-regularizacije *"weight_decay": 1e-4* (regulariziraju se svi slojevi osim slojeva za normalizaciju mini-grupa). Dobivaju se svi okviri za konfigurirane dimenzije slike *IMAGE_SIZE*, površine okvira *ANCHOR_SCALES* i omjere dimenzija okvira *ANCHOR_RATIOS* u *anchors*. Učitaju se težine modela iz konfigurirane lokacije u *load_weights* iz modela dobivenog zadnjom epohom učenja RPN-a (ako postoji).

U *for* petlji model se uči u zadanom broju epoha: gradijenti se postavljaju na 0 radi ispravnog ažuriranja parametara (*optimizer.zero_grad()*), nad slikom se zove unaprijedni prolaz RPN-a čime se dobivaju klasifikacijske mjere *cls_classes*, vjerojatnosti kao izlaz aktivacije softmax *cls_softmaxes* i pomaci okvira *bb_deltas*, nad njima se računaju se klasifikacijski *rpn_class_loss* i regresijski *rpn_bbox_loss* gubitak čime se dobiva ukupni gubitak *loss* čijim se unatražnim prolazom *loss.backward()* akumuliraju gradijenti i prikladno se ažuriraju parametri u *optimizer.step()*. Gubitci se ispisuju, a svakih 10 iteracija snimamo model RPN-a.

```
rpn = RPN(Resnet34_FPN(), 3)
rpn.cuda()
... # spremanje parametara u: trainable_params_no_batchnorm (svi
slojevi osim batchnorm), trainable_params_only_batchnorm (batchnorm)
```

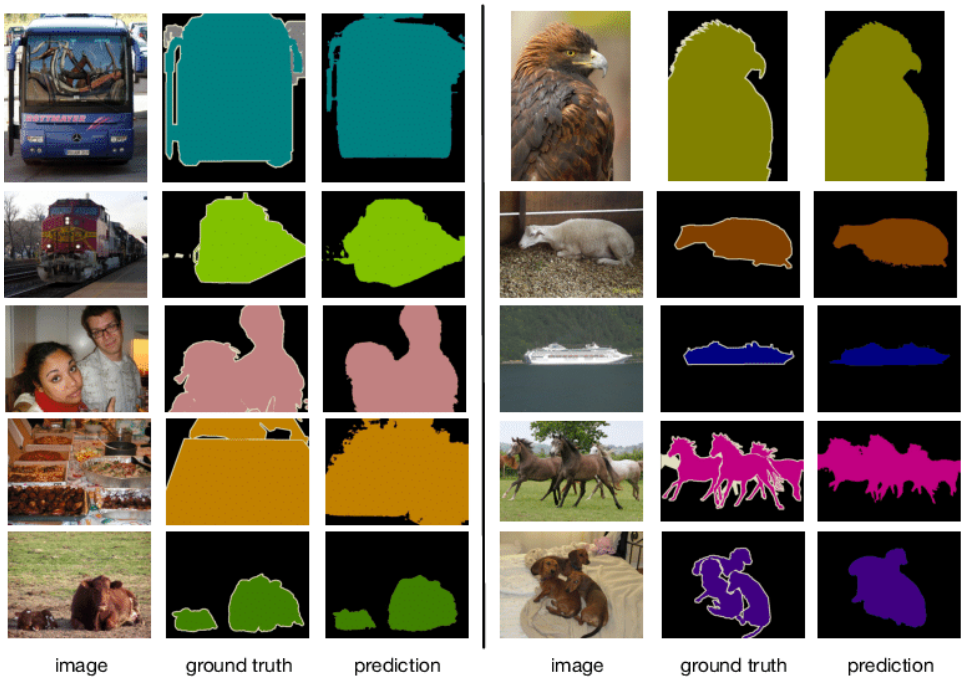
```
optimizer = torch.optim.SGD(params=
            [{"params": trainable_params_no_batchnorm,
             "weight_decay": 1e-4},
            {"params": trainable_params_only_batchnorm}],
            lr=0.001, momentum=0.9)

anchors = anchor_util.get_anchors(config.IMAGE_SIZE,
            config.ANCHOR_SCALES, config.ANCHOR_RATIOS)
... # učitavanje .jpg slike u img1 i .xml anotacija slike u bboxes1
... # određivanje zadnje epohe učenja modela u last_epoch_trained_num
rpn.load_weights("..\\" + config.WEIGHTS_FILE_PATH_BASE +
str(last_epoch_trained_num) + ".pth")

for epoch in range(start_epoch_num, end_epoch_num):
    print("Epoch", (epoch + 1))
    optimizer.zero_grad()

    cls_classes, cls_softmaxes, bb_deltas = rpn(img1_arr)

    rpn_cls, rpn_box_deltas =
    anchor_util.get_rpn_cls_and_box_deltas(anchors,
    np.array([bboxes1]))

    rpn_class_loss = calc_rpn_class_loss(rpn_cls, cls_classes)
    rpn_bbox_loss = calc_rpn_bbox_loss(rpn_box_deltas, rpn_cls,
    bb_deltas)
    loss = rpn_class_loss + rpn_bbox_loss

    torch.cuda.empty_cache()    # čišćenje memorije CUDA-e
    # backprop
    loss.backward()
    optimizer.step()

    print("rpn cls loss = {}, rpn bbox loss = {}, rpn total loss =
          {}".format(rpn_class_loss, rpn_bbox_loss, loss))
    if epoch % 10 == 9:
        torch.save(rpn.state_dict(), "..\\" +
        config.WEIGHTS_FILE_PATH_BASE + str(1 + epoch) + ".pth")
```

Kôd 3.5. Učenje modela RPN.

4. Ispitni skup izvornih slika

Za učenje modela RPN korišten je podatkovni skup PASCAL VOC2012 (engl. *Visual Object Classes Challenge 2012*) [29]. VOC2012 jedan je od standardnih skupova podataka za detekciju objekata koji sadrži velik skup slika za učenje i validaciju modela. Slike za detekciju su formata .jpg i za segmentaciju .png, dok su anotacije u obliku .xml datoteka.

Osim za detekciju i klasifikaciju objekata, VOC2012 koristi se i za segmentaciju objekata što je konačni cilj i samog Mask-RCNN-a. Primjeri segmentacije na VOC2012 prikazani su na Slika 4.1 gdje je redom prikazana slika, oznaka (segmentiran objekt boje koja označava njegovu klasu) i predviđena oznaka.



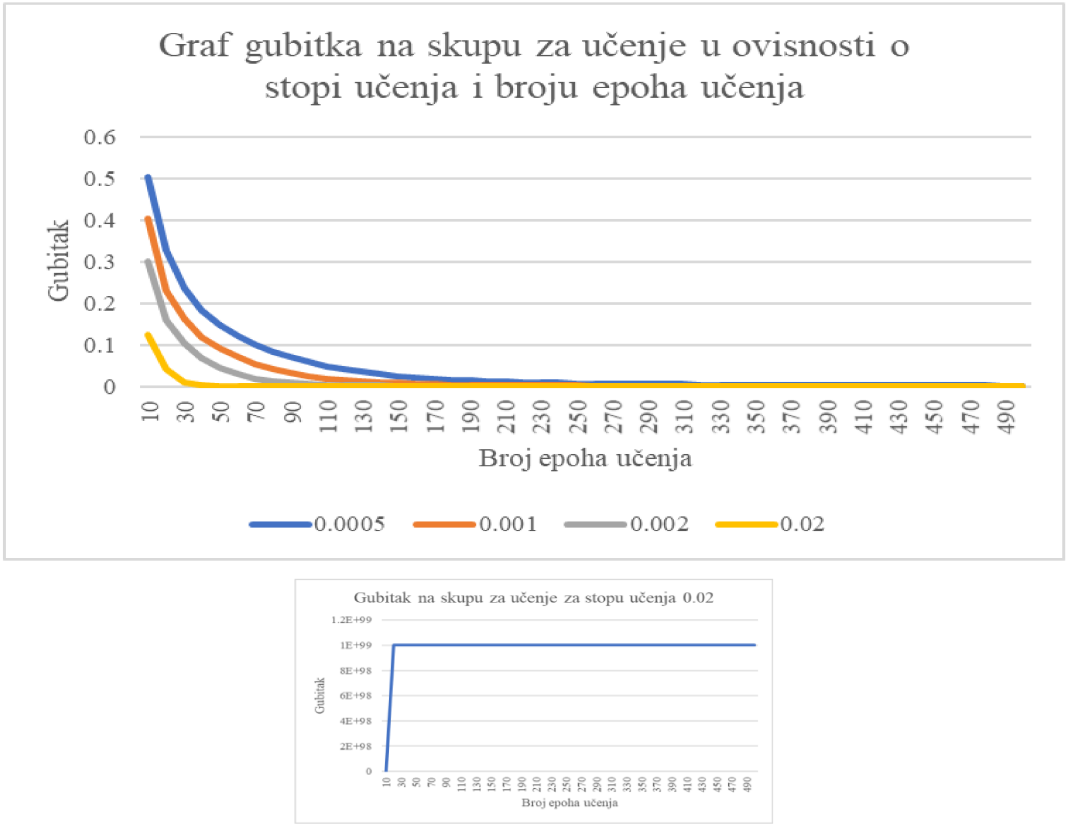
Slika 4.1. Primjeri segmentacije objekata – 3 stupca: slika, oznaka i predikcija [30].

U našem slučaju učili smo model na podskupu slika čiji se popis nalazi u datoteci *train_list_100.txt* što je podskup od 100 nasumično odabranih slika. Iz tog podskupa slika učili smo model na svakoj slici pojedinačno i zatim proveli ispitivanje na istoj. Rezultati su mjereni mjerom prosječnog odziva na 1000 kandidata (AR1000) koji je objektivna metrika prikladna za detekciju objekata.

5. Eksperimentalni rezultati

Ovdje ćemo prikazati rezultate dobivene eksperimentima učenja nakon 500 epoha nad 100 slučajno odabranih slika iz skupa VOC2012 s hiperparametrima modela za koje se model pokazao najuspješnijim. Uspješnost modela smo pritom ocijenili vizualno i objektivno gdje smo kao metriku odabrali AR1000 (opisano u 2.1.4.3).

Prvo prikažimo graf gubitka u ovisnosti o broju epoha učenja i stopi učenja (Slika 5.1). Jedan od glavnih hiperparametara koji smo trebali pažljivo odabrati bila je stopa učenja i stoga je izdvajamo kao jedan od ključnih hiperparametara.



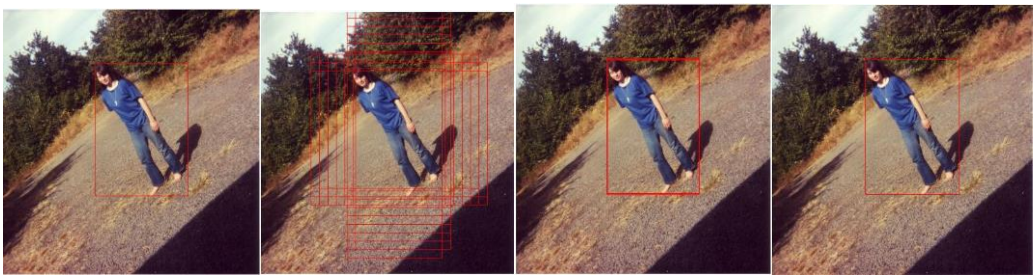
Slika 5.1. Graf gubitka na skupu za učenje u ovisnosti o stopi učenja i broju epoha učenja (gore) i gubitak na skupu za učenje za stopu učenja 0.02 (dolje) na skupu od 100 slika s popisa u datoteci *train_list_100_LR.txt*. Napomena 1: legenda na grafu gore pokazuje vrijednosti odgovarajućih stopa učenja. Napomena 2: kako gubitak za stopu učenja 0.02 je za 4 od 100 slika dao beskonačni gubitak u pojedinim epohama, on je zapravo jednak beskonačno. Ta beskonačnost je u grafu dolje samo simbolički prikazana kao velik broj 10^{99}, a u grafu gore je prikazan teoretski slučaj gdje su uzeti rezultati za preostalih 96 slika (oduzete su 4 slike koje daju beskonačne gubitke).

S grafa gubitka (Slika 5.1) najprije je vidljivo da se svi gubitci smanjuju s porastom broja epoha što je u redu. Zatim, specifična je stopa učenja 0.02 za koju gubitak na učenju 4 od 100 slika za koje je graf iscrtan ide u beskonačnost. Zbog toga je na gornjoj slici iscrtan teoretski slučaj gdje su te 4 slike izbačene iz razmatranja, dok je na slici dolje izdvojen stvaran gubitak za tu stopu učenja gdje je vidljivo da gubitak postaje beskonačan već nakon desetak epoha čemu je uzrok beskonačan gubitak na spomenute 4 od 100 slika (velik broj 10^{99} na grafu zapravo samo simbolično predstavlja beskonačnost).

U pogledu ostale tri stope učenja (0.0005, 0.001, 0.002), sve daju dobre rezultate: nakon 200. epohe gubitci su redom 0.01337, 0.00461, 0.001418, te nakon 500. epohe su redom 0.00277, 0.00095, 0.00034 za stope učenja 0.0005, 0.001, 0.002. Svi su gubitci relativno zadovoljavajući, a mi smo se odlučili za stopu učenja 0.001. Iako stopa 0.002 daje na danom uzorku od 100 slika manji gubitak na skupu za učenje, odlučili smo se za stopu učenja 0.001 kako nam ona daje također relativno zadovoljavajuć gubitak istog reda veličine (10^{-4} nakon 500. epohe), a dodatno, ta je stopa odabrana i u [27], službenoj implementaciji Mask-RCNN-a u PyTorchu, što nam u kombinaciji ukazuje da bi ta stopa učenja (0.001) mogla biti pravi izbor.

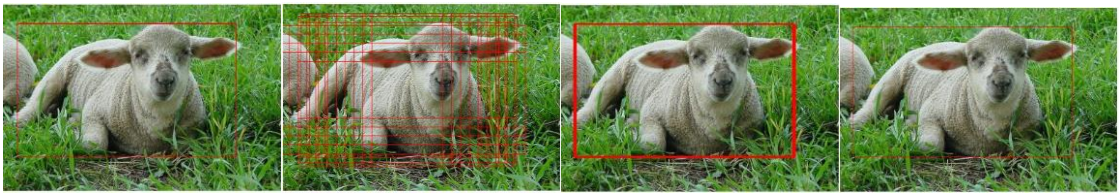
Prikažimo sada rezultate učenja na pojedinim slikama. Na primjeru svake slike dat ćemo: a) originalnu sliku s točnim okvirima, b) sliku s okvirima za učenje koji s točnima imaju mjeru preklapanja IoU >= 0.6, c) okvire iz b) nakon učenja i d) konačne okvire nakon primjene NMS-a koji će izbaciti one okvire manje klasifikacijske mjere koji se s onima veće klasifikacijske mjere preklapaju s IoU >= 0.5 i za rezultat dati najboljih 1000 okvira. Također ćemo dati iznos mjere AR1000.

Dajmo prvo nekoliko primjera s jednim objektom. Prvi je na Slika 5.2. Oko osobe imaju prikladni okviri prije učenja (b)) koji su se dobro prilagodili (c)) i konačno nakon NMS-a (d)) gdje su svi okviri manje klasifikacijske mjere koji se s onima veće klasifikacijske mjere preklapaju s IoU >= 0.5 uklonjeni, dobivamo dobar rezultat s AR = 1. Slično, primjer na Slika 5.3 ima približno dobar odziv AR = 0.9. Također, primjer na Slika 5.4 daje vrlo dobar rezultat i ima AR = 1.



Slika 5.2. 1 objekt, AR = 1.0 (slijeva nadesno: slučaj a) točni okviri, b) okviri za učenje koji s točnima imaju IoU >= 0.6, c) okviri iz b) nakon učenja, d) konačni okviri nakon NMS-a – izbačeni okviri manje klasifikacijske mjere koji s onima veće klasifikacijske mjere imaju IoU >= 0.5.

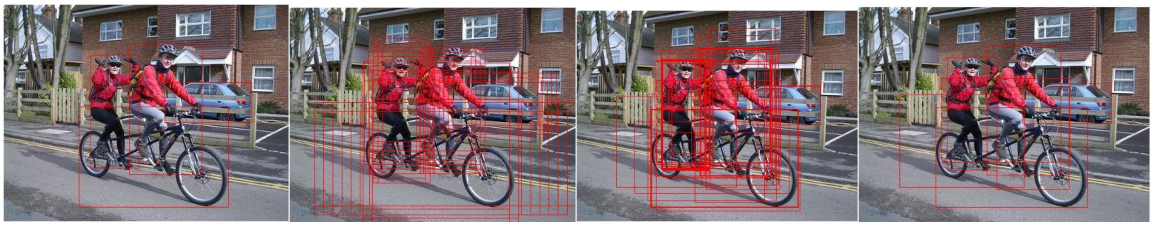
Slika 5.3. 1 objekt, AR = 0.9 (slijeva nadesno: slučaj a), b), c), d)).



Slika 5.4. 1 objekt, AR = 1.0 (slijeva nadesno: slučaj a), b), c), d)).

Prikažimo sada primjere slika s više objekata. Primjer na Slika 5.5 ima 5 objekata gdje svaki ima odgovarajuć broj prikladnih okvira. Nakon učenja dobiva se prilično velik broj okvira, a prema obrisima „gušćih“ dijelova vidi se ipak da 4 od 5 osoba imaju prikladne okvire, dok jedna osoba (u gornjem redu 3. slijeva nadesno) ima manji broj odgovarajućih okvira. Ipak nakon NMS-a dobiva se prilično dobar rezultat gdje je većini osoba pridijeljen jasan okvir uz nekoliko suvišnih. Prosječni odziv 0.8 je također zadovoljavajuć. Slično je s primjerom na Slika 5.6 gdje je situacija i ponešto bolja jer sva 4 objekta nakon učenja imaju dostatan broj okvira te i nakon NMS-a daju dobar rezultat s AR = 1. Sljedeći je primjer na Slika 5.7 koji ima manji broj objekata (dva) i dosad vizualno najbolji rezultat s također dobrim odzivom AR = 1. Ovdje nakon NMS-a uspješno dobivamo okvire koji su se vrlo dobro prilagodili točnima i, uz to, njihov broj je jednak broju objekata.

Slika 5.5. 5 objekata, AR = 0.8 (slijeva nadesno: slučaj a), b), c), d)).

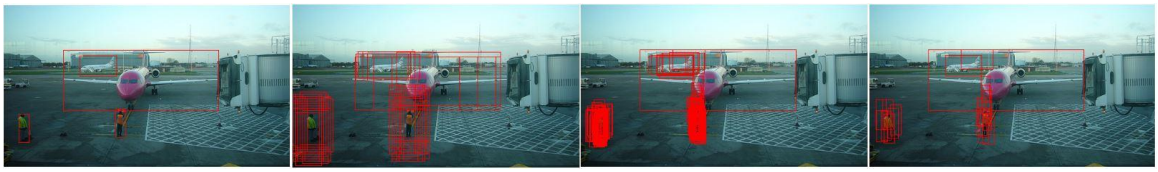


Slika 5.6. 4 objekta, AR = 1.0 (slijeva nadesno: slučaj a), b), c), d)).

Slika 5.7. 2 objekta, AR = 1.0 (slijeva nadesno: slučaj a), b), c), d)).

Sljedećih nekoliko primjera pokazuje jednu zanimljivu pojavu koja je primjetna na slikama s više objekata gdje su neki relativno dosta veći u odnosu na druge. Takav prvi primjer je na Slika 5.8 gdje imamo jedan veliki objekt (veliki avion naprijed) te 3 puno manja objekta (mali avion u pozadini i dvije osobe naprijed). Sve je u redu na početku gdje je svakom objektu dan odgovarajuć broj prikladnih okvira iako manji objekti imaju više okvira koji tijesno pristaju uz njih, no nakon učenja manji objekti, posebno najmanja dva (dvije osobe) dobivaju najveći broj okvira. To je vidljivo u krajnjem rezultatu gdje te dvije osobe imaju najveći broj okvira. Iako su svi objekti dobili zadovoljavajuće okvire, ta dva objekta mogla su imati i manji broj okvira. Konačni dobiveni AR je ipak dobar i jednak 0.975. Sljedeći takav primjer je na Slika 5.9 gdje vozilo koje je najdalje kao najmanji objekt slično dobiva nakon učenja velik broj okvira od kojih mnogi ostaju i nakon NMS-a. Kako je svaki objekt dobro pokriven prikladnim okvirima, odziv je 1.

Moguće rješenje dobivenoj pojavi da dobiveni model „preferira“ manje objekte mogla bi biti bolja prilagodba hiperparametara veličina odnosno površina okvira ili pak omjera dimenzija okvira čime bi eventualno mogli dobiti homogeniju situaciju s brojem okvira predloženih po objektu.

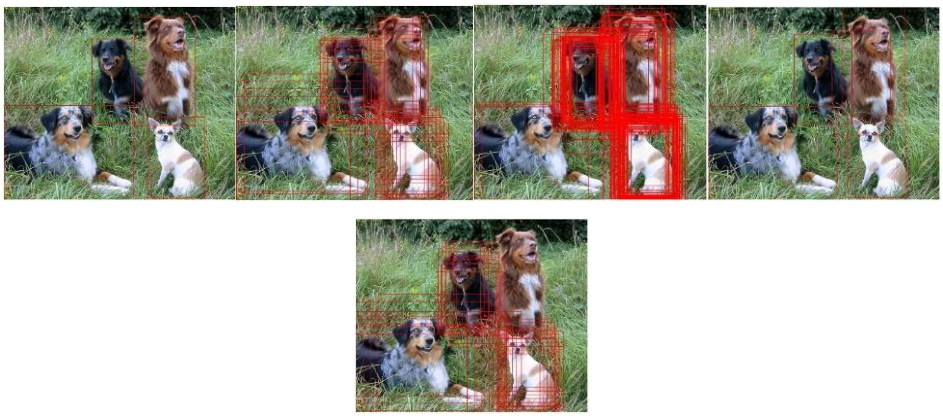


Slika 5.8. 4 objekta, AR = 0.975 (slijeva nadesno: slučaj a), b), c), d)).

Slika 5.9. 4 objekta, AR = 1.0 (slijeva nadesno: slučaj a), b), c), d)).

Sljedeći primjer specifičan je po jednom detalju koji je bilo važno uzeti u obzir pri implementaciji: iako za neki točni okvir ne postoji nijedan okvir koji se s njime preklapa s IoU većim od definiranog praga (određeno 0.6), svakom se točnom okviru trebaju pridijeliti oni okviri koji se s njime preklapaju najvećim IoU te se trebaju uzeti svi takvi okviri koji imaju tu maksimalnu vrijednost preklapanja s točnim okvirom. Primjer tome je Slika 5.10. Prvo, ovaj je primjer donekle sličan prethodnim dvama jer je vidljivo da 3 objekta od 4 dobivaju inicijalno dobre okvire koji dobro pristaju uz njih, dok preostali pas (dolje lijevo) ima okvire koji pristaju uz njega, ali ne toliko usko uz sam objekt. To se onda odražava i nakon učenja gdje svi osim tog psa dobivaju velik broj prilagođenih okvira, dok taj pas ima nekoliko okvira koji su ipak zadovoljavajuće prilagođeni. Nakon NMS-a je dobiven dobar rezultat s AR = 0.725.

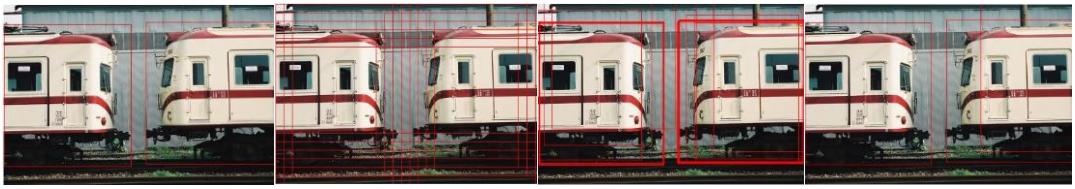
Drugo, ovaj primjer ilustrira činjenicu utvrđenu na početku odlomka što je prikazano na Slika 5.10 na slici u donjem redu gdje je dana slika s okvirima koje bi imali u slučaju pod b) tj. onih koji se s odgovarajućim točnim okvirima preklapaju s IoU >= 0.6. Vidljivo je da pas gore desno nema nijedan takav okvir. Stoga je potrebno osigurati da mu se ipak pridijele okviri s najvećom mjerom IoU preklapanja s njime. To smo implementirali i dobili dobar rezultat, a zasebna slika na Slika 5.10 dolje prikazuje što bi bilo u protivnom (što smo simulirali komentiranjem dijela koda koji to inače ostvaruje).



Slika 5.10. 4 objekta, AR = 0.725 (slijeva nadesno: gore – slučaj a), b), c), d), dolje – b) bez okvira koji se ne preklapaju s točnim okvirima s IoU >= 0.6, ali se od svih s njima najviše preklapaju).

Prikažimo još nekoliko primjera slika s više objekata. Na Slika 5.11 je primjer gdje su inicijalno svakom objektu pridijeljeni odgovarajući okviri gdje opet manji objekti (dvije osobe) imaju okvire koji ih uže opisuju pa nakon učenja imaju više okvira. Nakon NMS-a se dobiva rezultat koji je uglavnom u redu osim za desnog konja kojem zadnji dio tijela nije unutar okvira, zbog čega je i odziv manje zadovoljavajuć (AR = 0.65). Primjer na Slika 5.12 ima 2 objekta čije je okvire model dobro naučio te nakon NMS-a također dobivamo dobre rezultate osim što za desni objekt (vlak) ima okvir koji mu baš ne pristaje. Sukladno tome, odziv je nešto manji, no relativno zadovoljavajuć AR = 0.7.

Slika 5.11. 4 objekta, AR = 0.65 (slijeva nadesno: slučaj a), b), c), d)).



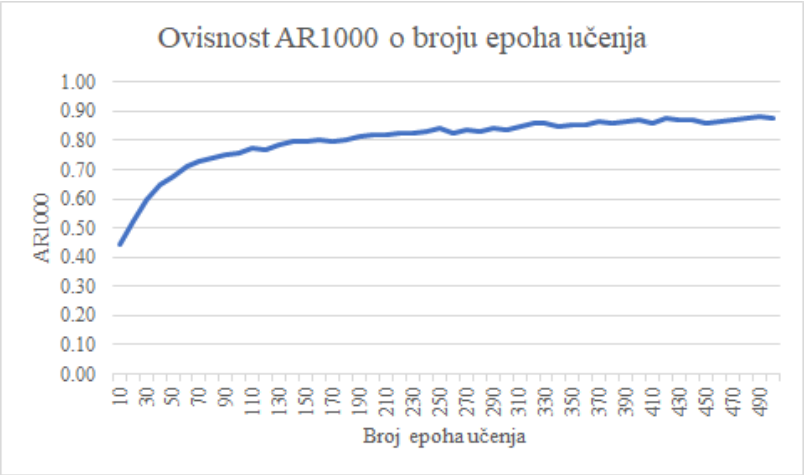
Slika 5.12. 2 objekta, AR = 0.7 (slijeva nadesno: slučaj a), b), c), d)).

Konačno, dajmo primjer jedne slike (Slika 5.13) s dosad najvećim brojem objekata – 7 objekata 4 različite vrste: 4 osobe, 1 televizor, 1 stolac i 1 boca. Inicijalno, svim su objektima pridijeljeni odgovarajući okviri. Nakon učenja okviri su se prilično uspješno prilagodili objektima. Najviše okvira imaju pritom 3 osobe, dok preostala osoba (2. slijeva

nadesno) kao i stolac, televizor i boca imaju znatno manji broj naučenih okvira. Nakon NMS-a većina objekata dobiva zadovoljavajuće okvire, osim televizora čiji se okvir mogao nešto bolje prilagoditi, a također postoji i nekoliko suvišnih okvira kao okvir koji preklapa dvije najdesnije osobe (većim dijelom lijevu). Međutim, sveukupno, možemo reći da je na ovakvoj složenijoj slici s većim brojem objekata različitih veličina i klasa rezultat ipak relativno zadovoljavajuć što pokazuje i AR od 0.8714 koji nam ukazuje na uspješnost modela RPN-a.

Slika 5.13. 7 objekata, AR = 0.8714 (slijeva nadesno: gore – slučaj a), b), dolje – c), d)).

Naposljetku, kako bismo se uvjerili da naš model RPN-a radi dobro na većem uzorku od prikazanih slika, naučili smo RPN na 100 slika zasebno i uzeli prosjek mjere AR1000 na modelima dobivenim učenjem nakon 10, 20, …, 480, 490, 500 epoha i dobili zadovoljavajuće rezultate (Slika 5.14): već nakon 200. epohe dostiže AR1000 = 0.82, a nakon 500. epohe AR1000 = 0.88.



Slika 5.14. Graf ovisnosti AR1000 o broju epoha učenja iz skupa {10, 20, …, 490, 500} na slikama iz PASCAL VOC2012 s popisa u datoteci *train_list_100.txt*.

Zaključak

Ovim radom predstavili smo model za predlaganje kandidata RPN kao ključnu komponentu modela za segmentaciju objekata Mask-RCNN. RPN je kao model predstavljen u modelu Faster R-CNN koji je omogućio istovremenu detekciju objekata i predlaganje kandidata okvira. Kao kralješnicu modela odabrali smo duboku konvolucijsku neuronsku mrežu ResNet-34 koja zajedno s modelom piramidalne strukture izlučivanja svojstava FPN omogućuje dobivanje semantički još bogatije reprezentacije ulazne slike. Stoga smo najprije implementirali arhitekturu i unaprijedni prolaz modelom ResNet-34 s njegovim blokovima rezidualnih jedinica, a zatim i sam model RPN. Nakon toga smo implementirali procedure za učenje modela i izračun klasifikacijskog i regresijskog gubitka. Također, implementirali smo mjeru uspješnosti modela AR1000 usrednjenu preko različitih pragova mjere preklapanja IoU. Tada smo krenuli učiti RPN.

Tijekom procesa učenja mjerili smo oba gubitka kako bismo bili sigurni da se proces dobro odvija, a nakon učenja uspješnost smo ocjenjivali vizualno i spomenutom mjerom prosječnog odziva. Radi dobivanja uspješnijeg modela validirali smo hiperparametre modela poput stope učenja, momenta u optimizaciji, praga IoU za koji će se okvir klasificirati u klasu objekta i dr. Ovaj dio je bio važan jer je trebalo oprezno postaviti vrijednosti tih hiperparametara kako bi RPN imao zadovoljavajuć gubitak i općenito davao zadovoljavajuće modele. Model je pritom bio neočekivano osjetljiv na stopu učenja koju smo trebali pažljivo odrediti. Na kraju smo dobili model koji, kao što je pokazano, ima relativno zadovoljavajuće rezultate vizualno i objektivno. Ipak, kako smo model uspjeli uspješno naučiti isključivo na 100 slika no pojedinačno jednu po jednu sliku, ima ograničenu primjenjivost, no može se iskoristiti u svrhu provjere ispravnosti osnovnog rada RPN-a i u edukativne svrhe na način koji smo ilustrirali u opisu eksperimentalnih rezultata.

Sljedeći korak bio bi naučiti model na mini-grupama slika čime bismo postigli generalizacijsku moć modela. Također, trebalo bi i validirati model kako bi se pronašla optimalna složenost RPN-a za koju bi bio najuspješniji na zasebnom validacijskom skupu i zatim testirati na skupu za testiranje. Kao konačni cilj, kako je RPN osnovna komponenta Mask-RCNN-a, mogli bismo implementirali cjelokupni model Mask-RCNN koji bi mogao segmentirati objekte.

Literatura

[1] Dalbelo Bašić B., Šnajder J., *1. Uvod u umjetnu inteligenciju*, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, (2019). Poveznica: https://www.fer.unizg.hr/_download/repository/UI-1-Uvod.pdf; pristupljeno 22. svibnja 2020.

[2] He, K., Gkioxari, G., Dollar, P., Girshick, R. *Mask R-CNN*, Facebook AI Research (FAIR), arXiv: 1703.06870v3 [cs.CV], (2018, siječanj). Poveznica: https://arxiv.org/pdf/1703.06870.pdf; pristupljeno 23. svibnja 2020.

[3] Ren, S., He, K., Girshick, R., Sun, J. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*, arXiv:1506.01497v3 [cs.CV], (2016, siječanj). Poveznica: https://arxiv.org/pdf/1506.01497.pdf; pristupljeno 23. svibnja 2020.

[4] Girshick, R. *Fast R-CNN*, Microsoft Research, arXiv:1504.08083v2 [cs.CV], (2015, rujan). Poveznica: https://arxiv.org/pdf/1504.08083.pdf; pristupljeno 24. svibnja 2020.

[5] Girshick, R., Donahue, J., Darnell, T., Malik., J. *Rich feature hierarchies for accurate object detection and semantic segmentation*, UC Berkeley, arXiv:1311.2524v5 [cs.CV], (2014, listopad). Poveznica: https://arxiv.org/pdf/1311.2524.pdf; pristupljeno 24. svibnja 2020.

[6] Olah C. *Neural Networks, Types, and Functional Programming*, (2015, rujan). Poveznica: http://colah.github.io/posts/2015-09-NN-Types-FP/; pristupljeno: 24. svibnja 2020.

[7] Karpathy A. *Software 2.0*, Medium, (2017, studeni). Poveznica: https://medium.com/@karpathy/software-2-0-a64152b37c35; pristupljeno: 24. svibnja 2020.

[8] Dalbelo Bašić B., Šnajder J., *10. Strojno učenje*, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, (2020). Poveznica: https://www.fer.unizg.hr/_download/repository/UI-2020-10-StrojnoUcenje.pdf; pristupljeno 24. svibnja 2020.

[9] Github kanal bfortuner, projekt ml-glossary, dokumentacija docs/acitvation_functions.rst, *Activation Functions*, (2018, kolovoz). Poveznica: https://github.com/bfortuner/ml-glossary/blob/master/docs/activation_functions.rst; pristupljeno: 24. svibnja 2020.

[10] Stanford University, SAD, stranice Predmeta CS231 Convolutional Neural Networks for Visual Recognition, *Learning*, *Parameter updates*, *First-order (SGD), momentum, Nesterov momentum*. Poveznica: https://cs231n.github.io/neural-networks-3/; pristupljeno: 3. lipnja 2020.

[11] Dalbelo Bašić B., Čupić, M., Šnajder J., *11. Umjetne neuronske mreže*, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, (2020). Poveznica: https://www.fer.unizg.hr/_download/repository/UI_12_UmjetneNeuronskeMreze[1].pdf; pristupljeno 24. svibnja 2020.

[12] Cao, Y.-P., Liu, Z.-N., Kuang, Z.-F., Kobbelt, L., Hu, S.-M. *Learning to Reconstruct High-quality 3D Shapes with Cascaded Fully Convolutional Networks*, Tsinghua University, Owlii Inc., RWTH Aachen University, The European Conference on Computer Vision (ECCV) 2018 papers, pp. 616-633, (2018). Poveznica: http://openaccess.thecvf.com/content_ECCV_2018/papers/Yan-Pei_Cao_Learning_to_Reconstruct_ECCV_2018_paper.pdf; pristupljeno: 11. lipnja 2020.

[13] Goodfellow, I., Bengio, Y., Courville, A. *Deep Learning*, MIT Press, (2016). Poveznica: https://www.deeplearningbook.org/; pristupljeno 25. svibnja 2020.

[14] Prabhu, R. *Understanding of Convolutional Neural Network (CNN) — Deep Learning*, Towards Data Science, Medium, (2018, ožujak). Poveznica: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148; pristupljeno 25. svibnja 2020.

[15] Santurkar, S., Tsipras, D., Ilyas, A., Madry, A. *How Does Batch Normalization Help Optimization?*, MIT, arXiv: 1805.11604v5 [stat.ML], (2019, travanj). Poveznica: https://arxiv.org/pdf/1805.11604.pdf; pristupljeno: 11. lipnja 2020.

[16] He, K., Zhang, X., Ren, S., Sun, J. *Deep Residual Learning for Image Recognition*, Microsoft Research, arXiv:1512.03385v1 [cs.CV], (2015, prosinac). Poveznica: https://arxiv.org/pdf/1512.03385.pdf; pristupljeno 25. svibnja 2020.

[17] Ruiz, P. *Understanding and visualizing ResNets*, Towards Data Science, Medium, (2018, listopad). Poveznica: https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8; pristupljeno 25. svibnja 2020.

[18] Lin, T.-Y., Dollar, P., Girshick, R., He, K., Hariharan, B., Belongie, S. *Feature Pyramid Networks for Object Detection*, Facebook AI Research (FAIR), Cornell University and Cornell Tech, arXiv:1612.03144v2 [cs.CV], (2017, travanj). Poveznica: https://arxiv.org/abs/1612.03144; pristupljeno 26. svibnja 2020.

[19] Tsang, S.-H. *Review: FPN — Feature Pyramid Network (Object Detection)*, Towards Data Science, Medium, (2019, siječanj). Poveznica: https://towardsdatascience.com/review-fpn-feature-pyramid-network-object-detection-262fc7482610; pristupljeno 26. svibnja 2020.

[20] PyTorch *Autograd: Automatic Differentiation*. Poveznica: https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html; pristupljeno 26. svibnja 2020.

[21] NumPy *NumPy*. Poveznica: https://numpy.org/; pristupljeno 26. svibnja 2020.

[22] NumPy *numpy.ndarray*. Poveznica: https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html; pristupljeno 26. svibnja 2020.

[23] NumPy *numpy.where*. Poveznica: https://numpy.org/doc/stable/reference/generated/numpy.where.html; pristupljeno 26. svibnja 2020.

[24] NumPy *numpy.arange*. Poveznica: https://numpy.org/doc/stable/reference/generated/numpy.arange.html; pristupljeno 26. svibnja 2020.

[25] Scikit-image *Scikit-image image processing in Python*. Poveznica: https://scikit-image.org/; pristupljeno 26. svibnja 2020.

[26] Pillow *Pillow*. Poveznica: https://pillow.readthedocs.io/en/stable/; pristupljeno 26. svibnja 2020.

[27] Matterport, Inc., GitHub kanal multimodallearning, projekt *PyTorch Mask RCNN*, implementacija modela Mask RCNN pod licencom The MIT License (MIT), (2017). Poveznica: https://github.com/multimodallearning/pytorch-mask-rcnn; pristupljeno 10. ožujka 2020.

[28] Kovač, B. *Lokalizacija i segmentacija objekata na slikama modelom Mask-RCNN*. Diplomski projekt. Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, siječanj 2020.

[29] PASCAL2 Pattern Analysis, Statistical Modelling and Computational Learning, *Visual Object Classes Challenge 2012 (VOC2012)*, (2012). Poveznica: http://host.robots.ox.ac.uk/pascal/VOC/voc2012/; pristupljeno 27. svibnja 2020.

[30] Shen, T., Lin, G., Liu, L., Shen, C., Reid, I. *Weakly Supervised Semantic Segmentation Based on Co-segmentation*, School of Computer Science, The University of Adelaide, Australia, School of Computer Science and Engineering, Nanyang Technological University, Singapore, arXiv:1705.09052v3 [cs.CV], (2017, kolovoz). Poveznica: https://www.researchgate.net/publication/317164080_Weakly_Supervised_Semantic_Segmentation_Based_on_Co-segmentation; pristupljeno 27. svibnja 2020.

Izlučivanje kandidata u dvoprolaznim lokalizacijskim modelima

Sažetak

Postoje mnogi modeli lokalizacije objekata: mi smo odabrali RPN. RPN je dvoprolazni lokalizacijski model duboke konvolucijske neronske mreže koji paralelno predviđa klasu objekta u razred objekt/pozadina te položaj i dimenzije okvira objekata. Pritom se oslanja na model FPN koji radi i prolaz konvolucijskom mrežom u suprotnom smjeru čime izlučuje još značajki spojivši lateralnim vezama mape značajki tih prolaza – lokalne informacije iz prvog i semantički vrednija svojstva iz drugog. Za osnovu FPN-a odabrali smo arhitekturu ResNet-34 koja rješava problem degradacije točnosti na skupu za učenje uslijed prevelike dubine konvolucijske mreže. Proučili smo ove modele i implementirali RPN. Naučili smo model na više slika iz podatkovnog skupa PASCAL VOC 2012 pojedinačno i eksperimentalno metrikom AR1000 potvrdili da učinkovito označava interesne regije okvirima kao rezultat učenja pozicija i dimenzija okvira radi još bolje prilagodbe objektu. Na kraju smo algoritmom NMS uspješno odabrali okvire koji se prema IoU najbolje preklapaju s točnima.

Ključne riječi: RPN, Mask-RCNN, FPN, ResNet-34, konvolucijske neuronske mreže, duboko učenje, diferencijabilno programiranje, strojno učenje, umjetna inteligencija, podatkovni skup PASCAL VOC 2012, AR1000, lokalizacija objekata, detekcija objekata, segmentacija objekata, dvoprolazni lokalizacijski modeli, IoU, interesne regije, okviri, točni okviri, NMS.

Candidate extraction in two-stage localization models

Abstract

There are many object localization objects: we have chosen RPN. RPN is a two-stage convolutional neural network localization model which in parallel predicts an object's class into object/background as well as position and dimension of its bounding boxes. Thereat it relies on FPN model, which does an additional pass through convolutional network in the opposite direction, which extracts more features by having lateral connections between the feature maps from the two passes – local information from the first and semantically more valuable features from the second one. As for the FPN groundwork we have chosen ResNet-34 architecture, which solves the training accuracy degradation problem appearing as a result of too large convolutional network depth. We have studied these models and implemented RPN. We have trained the model on multiple images from the PASCAL VOC 2012 dataset individually and by using AR1000 for the baseline experimentally confirmed it efficiently labels regions of interest with bounding boxes as a result of learning anchor positions and dimensions with aim of even better adjustment to the object. At the end, using NMS algorithm we have successfully selected the anchors which according to IoU have the largest overlap with the ground truth ones.

Keywords: RPN, Mask-RCNN, FPN, ResNet-34, convolutional neural networks, deep learning, differentiable programming, machine learning, artificial intelligence, PASCAL VOC 2012 dataset, AR1000, object localization, object detection, object segmentation, two-stage localization models, IoU, regions of interest, anchors, bounding boxes, NMS.