Detekcija lica korištenjem neuronske mreže YOLO v3 tiny

Rogina, Marko

Master's thesis / Diplomski rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University North / Sveučilište Sjever**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:122:704940

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-18**

Repository / Repozitorij:

University North Digital Repository

SVEUČILIŠTE SJEVER
SVEUČILIŠNI CENTAR VARAŽDIN

DIPLOMSKI RAD br. 006/MMD/2020

DETEKCIJA LICA KORIŠTENJEM NEURONSKE MREŽE YOLO V3 TINY

Marko Rogina

Varaždin, rujan 2020.

SVEUČILIŠTE SJEVER
SVEUČILIŠNI CENTAR VARAŽDIN

Studij: diplomski sveučilišni studij Multimedija

DIPLOMSKI RAD br. 006/MMD/2020

DETEKCIJA LICA KORIŠTENJEM NEURONSKE MREŽE YOLO V3 TINY

Student:

Marko Rogina: 0899/336D

Mentor:

izv. prof. dr. sc. Emil Dumić

Varaždin, rujan 2020.

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR-42000 Varaždin

Prijava diplomskog rada

Definiranje teme diplomskog rada i povjerenstva

ODJEL: Odjel za multimediju

STUDIJ: diplomski sveučilišni studij Multimedija

PRISTUPNIK: Marko Rogina | MATIČNI BROJ: 0899/336D

DATUM: 16.06.2020. | KOLEGIJ: Računalni vid

NASLOV RADA: Detekcija lica korištenjem neuronske mreže YOLO v3 tiny

NASLOV RADA NA ENGL. JEZIKU: Face detection using YOLO v3 tiny neural network

MENTOR: Emil Dumić | ZVANJE: doc.dr.sc.

ČLANOVI POVJERENSTVA:
1. doc.art.dr.sc. Mario Periša - predsjednik
2. doc.art. Robert Geček - član
3. doc.dr.sc. Emil Dumić - mentor
4. doc.dr.sc. Andrija Bernik - zamjenski član
5.

Zadatak diplomskog rada

BROJ: 006/MMD/2020

OPIS:

U ovom radu će biti napravljen program za detekciju lica pomoću učenja neuronske mreže YOLO v3 tiny. Detekcija objekata danas predstavlja važan korak u analizi slika i videozapisa. Detekcija objekata općenito uključuje njihovu klasifikaciju i lokalizaciju. Opisat će se različiti načini za detekciju objekata na slici: Haarove kaskade (Viola-Jones), histogram orijentiranih gradijenata (HOG) i metode potpornih vektora (SVM), te metode dubokog učenja. Specifično, sljedeće neuronske mreže će biti detaljnije objašnjene: CNN, R-CNN, Fast R-CNN, Faster R-CNN, YOLO (i YOLO tiny), Mask R-CNN. Bit će objašnjena razlika između detektora u 2 koraka i SSD detektora. Bit će objašnjena razlika između okvira objekata i maski objekata. Potom će se dati opis nekih metoda za mjerenje točnosti detekcije, poput IoU i mAP. Bit će objašnjeno i zašto se koristi povećanje broja podataka prilikom treniranja. Bit će objašnjen algoritam za odabir okvira objekta s najvećom vjerojatnošću (NMS). Bit će dan i pregled nekih postojećih baza s već unaprijed označenim i klasificiranim objektima.
Praktični dio zadatka je vezan za treniranje YOLO v3 tiny neuronske mreže ze detekciju lica. Bit će istraženi različiti parametri učenja koji mogu utjecati na krajnji rezultat, poput skalirane veličine slike, broja iteracija za učenje, načina povećanja broja podataka, početnih veličina blokova za traženje objekta (anchor). Nakon treniranja, validacije i testiranja, rezultati će biti uspoređeni s nekim od postojećih metoda za detekciju lica.

ZADATAK USUČEN: 16.6.2020. | POTPIS MENTORA: Emil Dumić

SVEUČILIŠTE SJEVER

Zahvala

Zahvaljujem se svojem mentoru izv. prof. dr. sc. Emilu Dumić koji je cijelo vrijeme nadzirao proces izrade te je svojim savjetima i prijedlozima pomogao u svladavanju prepreka koje su nastale tijekom izrade diplomskog rada. Zahvaljujem se na njegovom angažmanu i razumijevanju.

Marko Rogina

Sažetak

U ovom radu baviti ćemo se temom detekcije lica korištenjem neuronske mreže YOLO v3 tiny. Istražiti ćemo različite parametre učenja koji mogu utjecati na krajnji rezultat. Provest ćemo treniranje, validaciju i testiranje te napraviti usporedbu rezultata s nekim od postojećih metoda za detekciju lica. Također, definirati ćemo ključne pojmove računalnog vida. U ovom radu dan je i pregled ostalih sustava za detekciju objekata kako bi se uočili prednosti i nedostaci te spomenule neke ključne razlike te dobila ideja na koji način sustavi za detekciju najčešće rade i koje metode koriste.

Ključne riječi: računalni vid, klasifikacija, lokalizacija, sustavi za detekciju, duboko učenje, konvolucijske neuronske mreže.

Summary

In this paper, we will study the topic of face detection using the neural network YOLO v3 tiny. We will research different learning parameters that may affect the end result. We will carry out training, validation and testing and compare the results with some of the existing methods for face detection. We will also define the key terms of computer vision. This paper also provides an overview of other object detection systems in order to idenify advantages and disadvantages and mention some key differences to get an idea of how detection systems work and what methods they use.

Key words: computer vision, classification, localization, detection systems, deep learning, convolutional neural networks.

Popis korištenih kratica

HOG	Histogram orijentiranih gradijenata
SVM	Metoda potpornih vektora
CNN	Konvolucijska neuronska mreža
R-CNN	Konvolucijska neuronska mreža s prijedlozima regija
RPN	Mreža prijedloga regija
IoU	Presjek preko unije
RoI	Regija interesa
FPN	Piramidalna mreža značajki
YOLO	Pogledaj samo jednom
DPM	Modeli s deformiranim dijelovima
AP	Prosječna preciznost
mAP	Srednja prosječna preciznost
SSD	Jednonamjenski detektor

Sadržaj

1. Uvod ... 1
1.1 Predmet i cilj rada ... 2
1.2 Metode istraživanja ... 2
1.3 Struktura rada ... 2
2. Osnovni načini detekcije objekata na slici ... 3
2.1 Haarove kaskade (Viola-Jones) ... 3
2.2 HOG + SVM ... 8
3. Metode dubokog učenja ... 11
3.1 CNN ... 11
3.2 R-CNN ... 17
3.3 Brzi R-CNN ... 21
3.4 Brži R-CNN ... 24
3.5 Maskirani R-CNN ... 28
3.6 YOLO ... 31
3.7 Mjerenje točnosti detekcije ... 38
3.8 YOLO v3 ... 41
3.9 YOLO v3 tiny ... 45
4. Praktični dio zadatka ... 47
4.1 Treniranje s jednom klasom ... 47
4.2 Validacija i testiranje s jednom klasom ... 50
4.3 mAP rezultati za 1 klasu (evaluacija) ... 52
4.4 Treniranje s dvije klase ... 54
4.5 Validacija i testiranje s dvije klase ... 57
4.6 mAP rezultati za dvije klase (evaluacija) ... 59
4.7 Usporedba performansi i rezultata između YOLO v3 tiny i YOLO v4 tiny ... 60
4.8 Treniranje YOLO v3 tiny s 80 klasa ... 64
5. Zaključak ... 66
6. Literatura ... 69
Popis slika ... 73

1. Uvod

Danas je računalni vid jedno od najzanimljivijih polja umjetne inteligencije i strojnog učenja s obzirom da ima raznoliku primjenu i ogroman potencijal [1]. Računalni vid je disciplina koja obuhvaća polje umjetne inteligencije te uči strojeve da vide [2]. Cilj računalnog vida je replicirati snažne kapacitete ljudskog vida [1]. To se najčešće postiže osmišljavanjem računalnih modela baziranih na ljudskom vidnom sustavu [2]. Ljudi su sposobni razumjeti i opisati ono što vide na slici. To su ujedno i vještine koje sustav računalnog vida treba. Glavni problem koji se rješava pomoću računalnog vida može se sažeti u nekoliko riječi. S obzirom na dvodimenzionalnu sliku, sustav računalnog vida mora prepoznati prisutne predmete i njihove karakteristike poput oblika, teksture, veličine, prostornog rasporeda. Važno je razumjeti da računalni vid postiže puno više od same obrade slike ili strojnog vida [1].

Detekcija objekata danas predstavlja važan korak u analizi slika i videozapisa. Detekcija objekata općenito uključuje njihovu klasifikaciju i lokalizaciju. Važno je napomenuti da postoji razlika između algoritma za otkrivanje objekata i algoritma za klasifikaciju. Algoritmi za otkrivanje objekata pokušavaju napraviti granični okvir oko objekta kako bi ga locirali unutar slike dok algoritmi za klasifikaciju pokušavaju klasificirati objekt [3].

U prvom dijelu rada detaljnije su opisani sustavi za detekciju objekata na slici. Opisana ja arhitektura pojedinog sustava, odnosno način na koji je pojedini sustav osmišljen, koje su karakteristike tog sustava te način na koji sustav radi klasifikaciju i lokalizaciju kako bi detektirao objekt na slici. Također, opisane su prednosti i mane pojedinog sustava. Najprije su opisani osnovni sustavi poput haarovih kaskada, histograma orijentiranih objekata te metode potpornih vektora budući da su ove metode i sustavi bili odgovorni za daljnji napredak računalnog vida. Nadalje, opisuju se modernije metode za detekciju objekata poput metoda dubokog učenja. Metode dubokog učenja su izvedene na slične načine s par ključnih razlika. Također, možemo vidjeti da metode koji su osnovni sustavi koristili nisu bile najbrže i najpreciznije kada se usporede s modernijim metodama kao što su metode dubokog učenja koje koriste konvolucijske neuronske mreže. Opisane su neuronske mreže poput CNN, R-CNN, brzog R-CNNa, bržeg R-CNNa, maskirani R-CNN, YOLO i YOLO tiny. Objašnjena je razlika između detektora u 2 koraka i SSD detektora. Također, objašnjena je razlika između okvira objekata i maski objekata. Dan je opis nekih metoda za mjerenje točnosti detekcije, poput IoU i mAP. Objašnjeno je i zašto se koristi

povećanje broja podataka prilikom treniranja. Dan je i pregled nekih postojećih baza s već unaprijed označenim i klasificiranim objektima.

U drugom dijelu rada, fokus će biti na programu za detekciju lica pomoću učenja neuronske mreže YOLO v3 tiny. Istraženi su različiti parametri učenja koji mogu utjecati na krajnji rezultat, poput skalirane veličine slike, broja iteracija za učenje, načina povećanja broja podataka. Nakon treniranja, validacije i testiranja, napravljena je usporedba rezultata YOLO v3 tiny s jednom klasom, s dvije klase te je napravljena usporedba s nekim od postojećih metoda za detekciju lica.

1.1 Predmet i cilj rada

Predmet istraživanja je računalni vid. Preciznije, detekcija lica korištenjem neuronske mreže YOLO v3 tiny. Također, istražene su i ostale metode za detekciju objekata kako bi se dobio bolji uvid u ovu temu.

Cilj rada je istražiti i prikazati mogućnosti računalnog vida, tj. istražiti i prikazati mogućnosti neuronske mreže YOLO v3 tiny. Bit će istraženi različiti parametri te prednosti i nedostaci.

1.2 Metode istraživanja

Prilikom izrade rada korištena je znanstvena i stručna literatura s područja računalnog vida. Također, jedan dio istraživanja proveden je na računalu pomoću neuronske mreže YOLO v3 tiny. Provedena je usporedba dobivenih rezultata s već postojećim metodama za detekciju lica.

1.3 Struktura rada

Rad je podijeljen na 6 dijela. U prvom poglavlju je uvod gdje će se čitatelja detaljnije upoznati s predmetom, svrhom i ciljevima diplomskog rada. U drugom poglavlju reći će se nešto o načinu detekcije objekata na slici te osnovnim sustavima detekcije. U trećem poglavlju detaljnije će se objasniti neuronske mreže, pojašnjavaju se način i princip rada te ključne razlike. U četvrtom poglavlju objašnjeno je na koji način se proveo postupak istraživanja za detekciju lica te što je sve bilo istraženo. U petom poglavlju donosi se zaključak. U šestom poglavlju je navedena literatura.

2. Osnovni načini detekcije objekata na slici

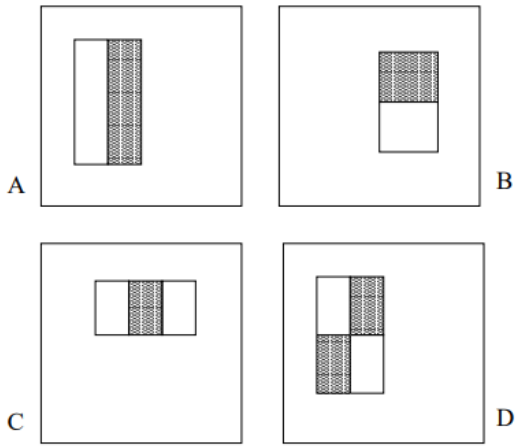
Opisat će se različiti načini za detekciju objekata na slici: Haarove kaskade, histogram orijentiranih gradijenata, metode potpornih vektora.

Osnovni modeli za klasifikaciju slika su poprilično jednostavni. Kao ulaz koriste sliku te im je cilj klasificirati tu sliku kao jednu od mnogih mogućih izlaznih klasa [4]. Klasifikacija slike uključuje predviđanje klase jednog objekta na slici dok se lokalizacija objekta odnosi na identificiranje položaja jednog ili više objekata na slici [5].

2.1 Haarove kaskade (Viola-Jones)

Sustav za klasifikaciju slika koji omogućava odrediti koji objekt se nalazi na slici. Objedinjuje algoritme i uvide kako bi se konstruirao okvir za robusnu i izuzetno brzu detekciju objekta. Kod ostalih sustava za prepoznavanje lica koriste se pomoćne informacije poput: razlika među slikama u video sekvencama ili boja piksela kod slika u boji. Ovaj sustav postiže visoke brzine u obradi iz razloga što radi samo s informacijama prisutnim u jednoj slici svjetline [6].

Ovaj postupak detekcije lica klasificira slike na osnovi vrijednosti jednostavnih značajki. Postoji mnogo razloga za korištenje značajki, a ne piksela. Najčešći razlog je taj što značajke mogu djelovati na način da prema potrebi kodiraju znanje o domeni koje je teško naučiti koristeći ograničenu količinu podataka prilikom treniranja. Drugi razlog je taj što sustav temeljen na značajkama djeluje puno brže od sustava koji je temeljen na pikselima. Jednostavne značajke koje ovaj sustav koristi podsjećaju na osnovne *Haar* funkcije. Konkretnije, ovaj sustav koristi tri glavne značajke koje pomažu prilikom detekcije objekata. Vrijednost značajke s dva pravokutnika koja predstavlja razliku između zbroja piksela unutar dva pravokutna polja. Pravokutna polja su iste veličine i istog oblika te su vodoravno ili okomito susjedni. Značajka s tri pravokutnika izračunava vrijednost zbroja između dva vanjska pravokutnika te oduzima od zbroja u središnjem pravokutniku. Na kraju, značajka s četiri pravokutnika izračunava razliku između dijagonalnih parova pravokutnika. S obzirom da je osnovna razlučivost detektora 24x24, cjelokupni skup značajki pravokutnika je poprilično velik [7]. To možemo vidjeti na slici 2.1.



Slika 2.1

Zbroj piksela koji se nalaze unutar bijelog pravokutnika oduzimaju se od zbroja piksela koji se nalaze unutar sivog pravokutnika. Značajka s dva pravokutnika prikazana je u (A) i (B), (C) prikazuje značajku s tri pravokutnika, a (D) prikazuje značajku s četiri pravokutnika.

http://www.vision.caltech.edu/html-files/EE148-2005-Spring/pprs/viola04ijcv.pdf, pristup: 15.6.2020.

Značajke pravokutnika mogu se izračunati vrlo brzo koristeći posredni prikaz slike koji se zove integralna slika (formula 2.1). Integralna slika na mjestu *x,y* sadrži broj piksela s gornje i s lijeve strane od *x,y* uključujući: [7]

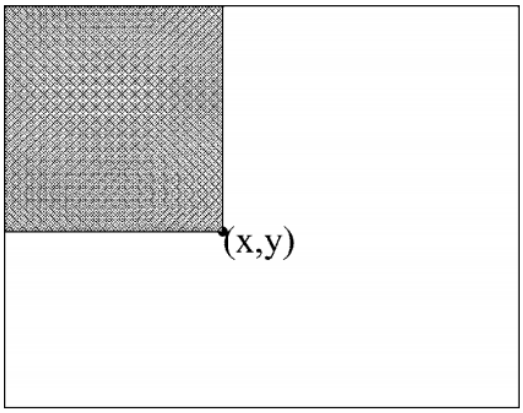
$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x',y'), \tag{2.1}$$

pri čemu je $ii(x,y)$ integralna slika, a $i(x,y)$ je izvorna slika, korištenjem: [7]

$$s(x,y) = s(x,y-1) + i(x,y), \tag{2.2}$$

$$ii(x,y) = ii(x-1,y) + s(x,y), \tag{2.3}$$

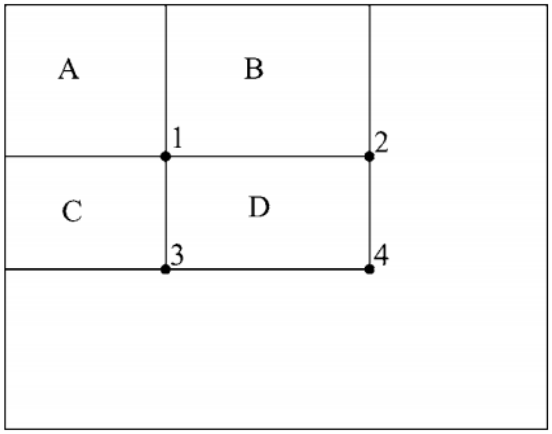
(pri čemu je $s(x,y)$ cjelokupni zbroj redaka, $s(x,-1) = 0$, i $ii(-1,y) = 0$) integralna slika se može izračunati u jednom prolazu preko originalne slike (formule 2.2 – 2.3) [7]. To možemo vidjeti na slici 2.2.



Slika 1.2

Vrijednost integralne slike u točki (x, y) je zbroj svih piksela s gornje i s lijeve strane.
http://www.vision.caltech.edu/html-files/EE148-2005-Spring/pprs/viola04ijcv.pdf, pristup: 15.6.2020.

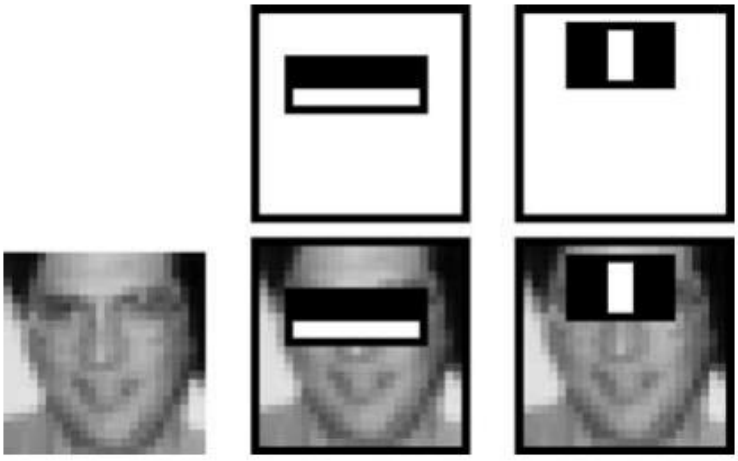
Pomoću integralne slike svaki se pravokutni zbroj može izračunati u četiri referentna polja. Razlika između dva pravokutna zbroja može biti izračunata u osam referentna polja. Budući da značajka s dva pravokutnika uključuje susjedne pravokutnike, zbrojevi se mogu izračunati u šest referentnih polja, osam u slučaju značajke s tri pravokutnika te devet za značajke s četiri pravokutnika [7]. To prikazuje slika 2.3.



Slika 2.3

Zbroj piksela unutar pravokutnika D može se izračunati s četiri referentna polja. Vrijednost integralne slike na lokaciji 1 je zbroj piksela u pravokutniku A. Vrijednost na lokaciji 2 je A + B, na lokaciji 3 je A + C, a na lokaciji 4 je A + B + C + D.
http://www.vision.caltech.edu/html-files/EE148-2005-Spring/pprs/viola04ijcv.pdf, pristup: 15.6.2020.

Kako bi se osigurala brza klasifikacija, proces učenja mora isključiti veliku većinu dostupnih značajki i usredotočiti se na mali skup kritičnih značajki. Ovaj sustav koristi metodu konstruiranja klasifikatora odabirom malog broja važnih značajki pomoću *AdaBoost* algoritma za učenje. *AdaBoost* algoritam za učenje koristi se za poboljšanje klasifikacije jednostavnog algoritma za učenje (npr. može se koristiti za poboljšanje performansi jednostavnog perceptrona[1]). To radi na način da kombinira skup slabih klasifikacijskih funkcija kako bi stvorio jači klasifikator. Algoritam za učenje pretražuje skup mogućih perceptrona te vraća perceptron s najnižim pogreškama klasifikacije. U ovom smislu, jednostavan algoritam za učenje ili perceptron naziva se *slabim učenikom* jer ne očekujemo čak ni najbolju klasifikacijsku funkciju da dobro klasificira podatke o učenju. Da bi se *slab učenik* poboljšao, on se poziva da riješi niz problema s učenjem. Nakon prvog kruga učenja, primjeri se ponovno pregledavaju kako bi se naglasili oni koji su bili pogrešno klasificirani od prethodnog slabog klasifikatora. Na kraju, jak klasifikator postaje perceptron [7]. To možemo vidjeti na slici 2.4.



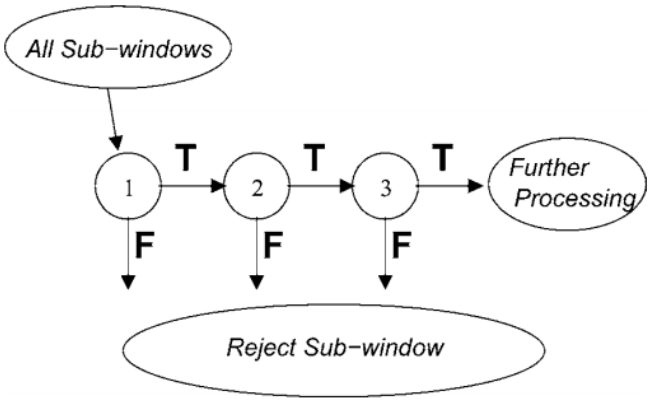
Slika 2.4

Prva i druga značajka odabrana od AdaBoost. Dvije značajke prikazane su u gornjem redu, a zatim su te dvije značajke postavljene na lice za treniranje u donjem redu. Prva značajka mjeri razliku u intenzitetu između područja oko očiju i području oko gornjih obraza. Značajka dobiva prednost na tome što je područje oko očiju najčešće tamnije od obraza. Druga značajka uspoređuje intenzitet oko područja očiju s intenzitetom gornjeg dijela nosa.

http://www.vision.caltech.edu/html-files/EE148-2005-Spring/pprs/viola04ijcv.pdf, pristup: 15.6.2020.

[1] Perceptron je algoritam koji se koristi za učenje s nadzorom binarnih klasifikatora. Binarni klasifikatori odlučuju da li ulaz, koji je obično predstavljen nizom vektora, pripada određenoj klasi. (https://deepai.org/machine-learning-glossary-and-terms/perceptron, pristup: 15.6.2020.)

Sustav također ima algoritam koji omogućava kombiniranje složenijih klasifikatora u kaskadnu strukturu što izuzetno povećava brzinu detekcije, a samim time smanjuje samo vrijeme računanja. Ključna stvar je ta što se može napraviti manji te efikasniji i poboljšani klasifikatori koji odbacuju sve negativne slučajeve koji nisu lice, dok detektiraju gotovo sve pozitivne slučajeve. Najčešće, jednostavniji klasifikatori služe za odbijanje većinu negativnih slučajeva, a nakon toga se pozivaju kompleksniji klasifikatori kako bi se postigle što niže stope lažno pozitivnih slučajeva. Zapravo, cjelokupan proces detekcije je baziran na odlukama i mogućim posljedicama. Pozitivan ishod iz prvog klasifikatora pokreće procjenu drugog klasifikatora. Pozitivan ishod iz drugog klasifikatora pokreće treći klasifikator. Negativni ishod u bilo kojoj fazi dovodi do izravnog prekidanja procesa detekcije. Kaskadna struktura zapravo pokušava odbaciti što više negativnih slučajeva u najranijoj fazi detekcije. Sljedeći klasifikator je uvijek složeniji od prethodnog što znači da složeniji klasifikatori imaju veće stope otkrivanja lažno pozitivnih slučajeva [7]. To možemo vidjeti na slici 2.5.



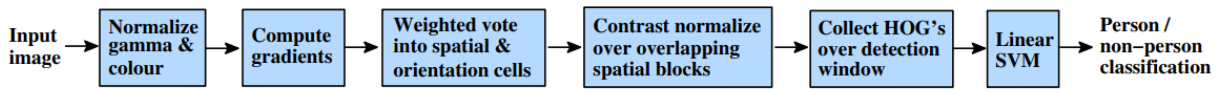
Slika 2.5

Shematski prikaz kaskadne detekcije. Serija klasifikatora koja se primjenjuje na svaki slučaj. Početni klasifikator eliminira velik broj negativnih slučajeva s vrlo malo obrade. Sljedeći klasifikatori eliminiraju dodatne negativne slučajeve no zahtijevaju dodatno proračunavanje. Nakon nekoliko faza obrade broj slučajeva se radikalno smanjio.

http://www.vision.caltech.edu/html-files/EE148-2005-Spring/pprs/viola04ijcv.pdf, pristup 15.6.2020.

2.2 HOG + SVM

Metoda histograma orijentiranih gradijenata (*eng. Histogram of Oriented Gradients*) temelji se na ocjenjivanju dobro normaliziranih lokalnih histograma orijentacijskih gradijenata slike u gustoj mreži. Osnovna ideja je da se izgled i oblik lokalnog objekta često mogu okarakterizirati poprilično dobro raspodjelom lokalnih intenziteta gradijenata ili položaja ruba, čak i ako ne znamo točan položaj odgovarajućih gradijenata ili rubova. To se najčešće provodi na način da se slika podijeli na male prostorne ćelije, gdje svaka ćelija akumulira smjerove gradijenata ili orijentaciju ruba. Kako se osvjetljenje i sjenčanje ne bi previše promijenilo, također je korisno normalizirati kontrast. To se može postići na način da se energija lokalnog histograma donekle akumulira na veće prostorne blokove te da se koriste rezultati kako bi se normalizirale sve ćelije u bloku. Normalizirani blokovi deskriptora se još mogu nazvati i deskriptori histograma orijentiranih gradijenata. Spajanjem detektora s gustom mrežom HOG deskriptora te koristeći metodu potpornih vektora (*eng. Support Vector Machines*) može se detektirati objekt na slici [8]. Način procesa možemo vidjeti na slici 2.6.

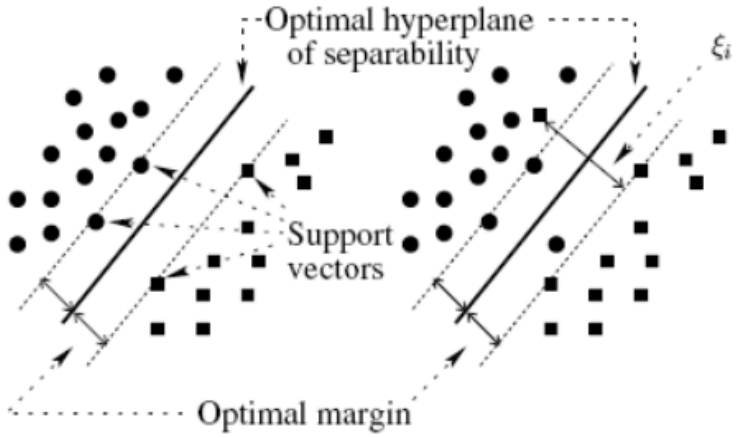


Slika 2.6

Pregled procesa za otkrivanje objekata

http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf, pristup: 17.6.2020.

SVM pristup nastoji pronaći optimalnu hiper-ravninu između klasa fokusirajući se na slučajeve koji su prilikom treniranja stavljeni na rub deskriptora klase. Ti slučajevi prilikom treniranja se nazivaju potporni vektori. Slučajevi koji nisu potporni vektori se odbacuju. Na ovaj način se koriste učinkovitiji uzorci prilikom treniranja koji omogućavaju da se s malim brojem uzoraka postigne visoka točnost klasifikacije. Može se koristiti kod binarne klasifikacije kod koje su klase linearno odvojive. To znači da je moguće pronaći barem jednu hiper-ravninu definiranu vektorom koji može razdvojiti klasu bez pogreške [9]. To možemo vidjeti na slici 2.7.

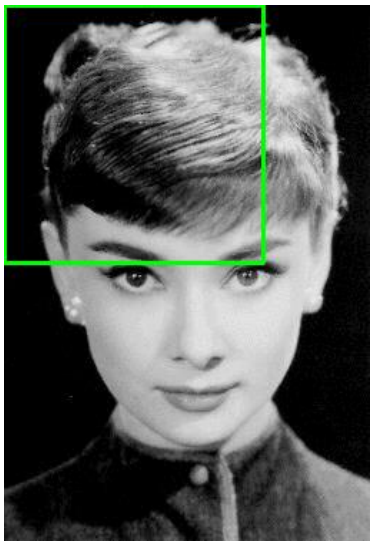


Slika 2.7

Lijevo: slučaj linearnih klasa koje se mogu odvojiti. Desno: slučaj nelinearnih klasa koje se mogu odvojiti. ξ mjeri pogrešku hiper-ravnine.

https://www.researchgate.net/publication/225929583_Support_Vector_Machine_Classification_for_Object-Based_Image_Analysis, pristup: 17.6.2020.

Općeniti proces treniranja detektora za objekt pomoću histograma orijentiranih gradijenata bi izgledao ovako: Uzorak *P* predstavlja pozitivne uzorke objekata koje želimo prepoznati te izdvojiti HOG deskriptore iz tih uzoraka. Uzorak *N* predstavlja negativne uzorke koji ne sadrže nijedan objekt koji želimo detektirati. Također, izdvojimo HOG deskriptore iz tih uzoraka. Nakon toga, pomoću pozitivnih i negativnih uzoraka, istrenira se linearna metoda potpornih vektora. Za svaku sliku i svaku moguću veličinu slike u negativnom setu uzoraka aplicira se tehnika klizajućeg prozora. Za svaki prozor računa se HOG deskriptor te se primjenjuje klasifikator. Ako klasifikator pogrešno klasificira pojedini prozor kao objekt, zapiše se vektor koji je povezan s lažno pozitivnim rezultatima zajedno s mogućom klasifikacijom. Zatim se uzimaju lažno pozitivni uzorci pronađeni tijekom gore navedene tehnike koji se razvrstaju prema pouzdanosti te se radi reklasifikacija klasifikatora pomoću tih uzoraka. Klasifikator je istreniran i može se primijeniti na testni skup podataka [10]. Primjer kliznog prozora možemo vidjeti na slici 2.8.



Slika 2.8

Primjer kliznog prozora gdje se prozor pomiče s lijeva na desno i odozgo prema dolje

https://www.pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/, pristup: 17.6.2020.

3. Metode dubokog učenja

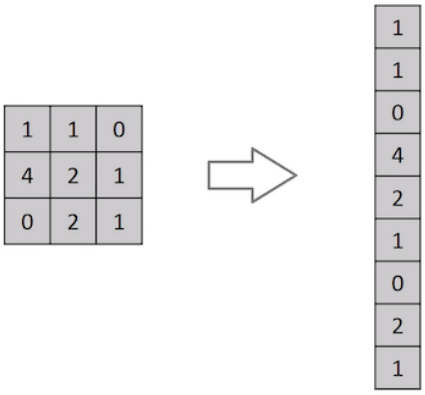
Opisat će se metode dubokog učenja. Specifično, sljedeće neuronske mreže će biti detaljnije objašnjene: CNN, R-CNN, brzi R-CNN, brži R-CNN, maskirani R-CNN, YOLO i YOLO tiny.

Metoda dubokog učenja je tehnika strojnog učenja koja uči računalo da radi ono što mogu raditi ljudi. Računalni model uči izvoditi klasifikacije izravno iz slike. Modeli dubokog učenja mogu postići vrhunsku preciznost, ponekad prelazeći performanse samog čovjeka. Modeli se treniraju pomoću velikog skupa podataka i neuronskih mreža koje sadrže mnogo slojeva. Izraz *duboko* obično se odnosi na broj skrivenih slojeva neuronske mreže [11].

3.1 CNN

Konvolucijska neuronska mreža (*eng. Convolutional Neural Network*) algoritam je dubokog učenja koji pomoću ulazne slike može dodijeliti važnost različitim aspektima, tj. objektima na slici te ih je u mogućnosti razlikovati jedan od drugoga. Predobrada koja je potrebna u konvolucijskoj mreži znatno je manja u usporedbi s ostalim algoritmima za klasifikaciju. Također, konvolucijske mreže imaju mogućnost naučiti karakteristike za razliku od primitivnih metoda. Arhitektura konvolucijskih mreža je slična arhitekturi neurona u ljudskom mozgu. Pojedini neuroni reagiraju na podražaje u ograničenom području vidnog polja. Skup takvih polja preklapa se kako bi pokrili čitavo vizualno polje [12].

Slika nije ništa drugo nego matrica vrijednosti piksela, zar ne? Zašto ne bismo jednostavno spljoštili sliku (npr. 3x3 matricu slike u 9x1 vektor) i plasirali ju u više stupanjski perceptor kako bi napravili klasifikaciju? Ovo baš i ne bi bilo dobro. Kod ekstremno osnovnih slučajeva kao što su binarne slike, ova metoda bi mogla pokazati prosječnu preciznost tijekom predviđanja klasa ali bi imala slabu ili nikakvu točnost kada je riječ o složenim slikama koje ovise o pikselima [12]. Spljoštavanje matrice u vektor možemo vidjeti na slici 3.1.

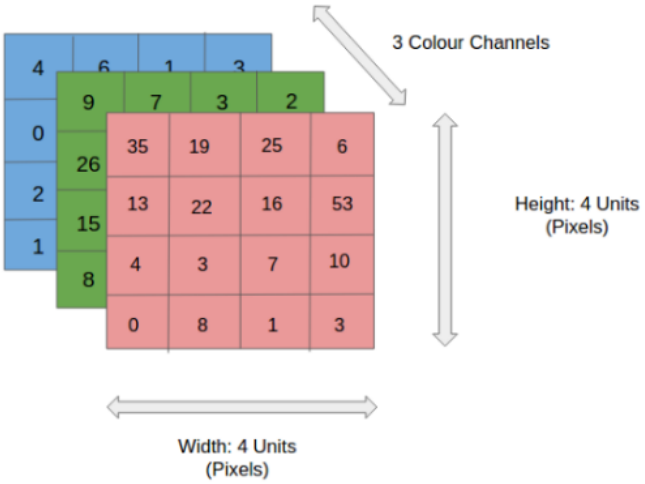


Slika 3.1

Spljoštavanje 3x3 matrice u 9x1 vektor

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, pristup: 17.6.2020.

Konvolucijska mreža može uspješno obuhvatiti prostorne i vremenske ovisnosti na slici pomoću primjene relevantnih filtera. Ova arhitektura je bolje prilagođena skupu podataka slike jer uključuje manji broj parametara. Uloga konvolucijske mreže je smanjiti sliku u oblik koji je lakši za obradu bez da se izgube značajke koje su ključne za dobivanje dobrih pretpostavki [12]. Ovo možemo vidjeti na slici 3.2.

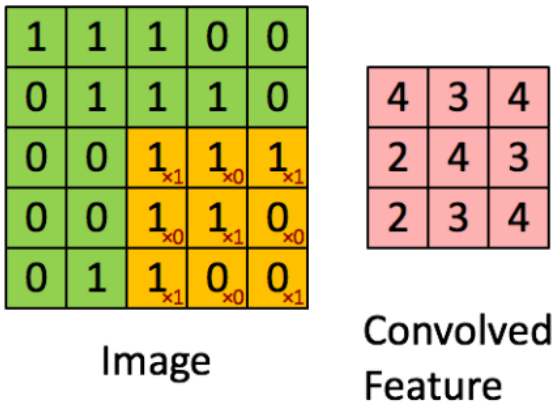


Slika 3.2

RGB slika koja je podijeljena na tri boje - crvena, plava i zelena. Zamislite kako bi se zakomplicirale stvari prilikom izračunavanja da je slika dimenzija 8k.

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, pristup: 17.6.2020.

Ako imamo sliku dimenzija 5 (visina) x 5 (širina) x 1 (broj kanala, npr. RGB), na slici dolje zeleni dio predstavlja 5x5x1 ulaznu sliku *I*. Element koji sudjeluje u izvođenju operacije konvolucije u prvom dijelu konvolucijskog sloja se naziva kernel *K*, koji je prikazan žutom bojom. *K* je odabran kao 3x3x1 matrica. Kernel se pomiče 9 puta budući da je *stride*[2] 1 prilikom izračunavanja. Filter se pomiče u desno s određenom vrijednošću pomaka tako dugo dok ne prođe čitavu širinu [12]. Ovo možemo vidjeti na slici 3.3.



Slika 3.3

5x5x1 slika s 3x3x1 kernelom kako bi se dobila 3x3x1 konvolucijska značajka

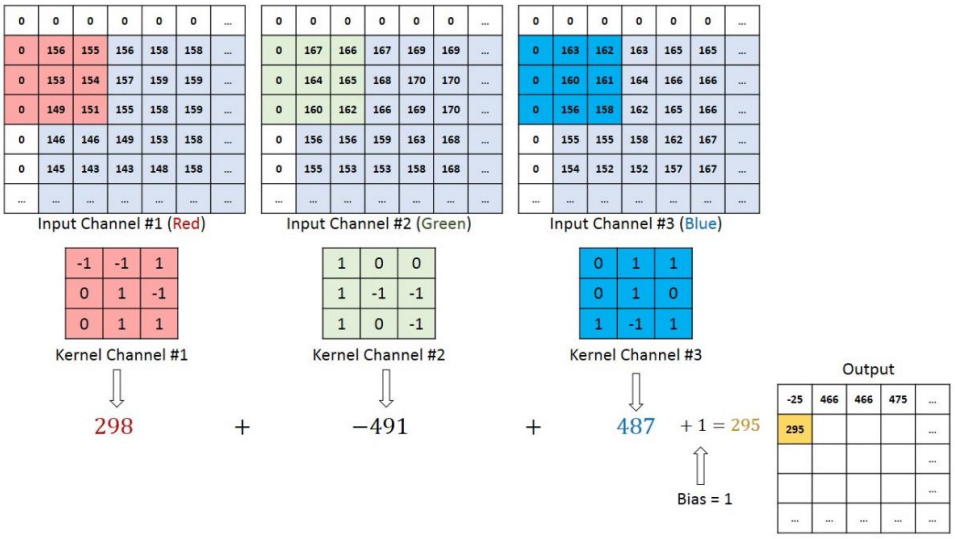
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, pristup: 17.6.2020.

U slučaju slika s više kanala (npr. RGB), kernel ima istu dubinu kao i na ulaznoj slici (formula 3.1). Računanje matrice između *Kn* i *In* se izvodi na sljedeći način: [12]

$$([K1, I1]; [K2, I2]; [K3, I3]) \tag{3.1}$$

te su svi rezultati zbrojeni s biasom kako bi se dobio kanal s jednom dubinom na izlazu konvolucije [12]. Ovo je prikazano na slici 3.4.

[2] *Stride* je sastavni dio konvolucijskih neuronskih mreža ili neuronskih mreža podešenih za kompresiju slika. *Stride* je parametar koji modificira broj pomaka preko slike. Na primjer, ako je pomak postavljen na 1, filter će se svaki put pomicati za jedan piksel. *Stride* je najčešće postavljen na cijeli broj, a ne na decimalni broj. (https://deepai.org/machine-learning-glossary-and-terms/stride, pristup: 17.6.2020.)



Slika 3.4

Konvolucijska operacija na MxNx3 matrici slike s kernelom 3x3x3

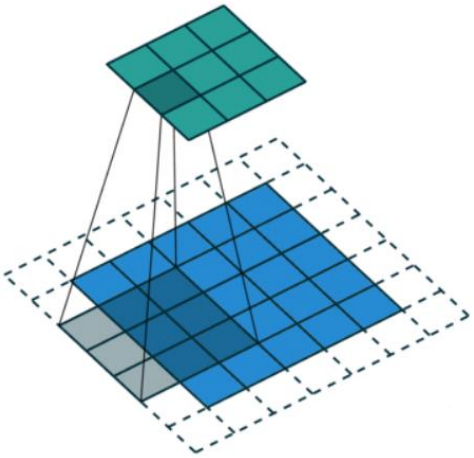
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, pristup: 17.6.2020.

Cilj konvolucijske operacije je da iz ulazne slike izdvoji značajke visoke razine poput rubova. Konvolucijska mreža ne mora biti ograničena na samo jednom konvolucijskom sloju. Najčešće, prvi konvolucijski sloj odgovoran je za značajke niske razine kao što su boja i orijentacija gradijenta. Uz dodatne slojeve arhitektura se prilagođava i značajkama visoke razine, pružajući nam mrežu koja ima dobro razumijevanje slika. Postoje dvije vrste rezultata – prvi kod kojeg je konvolucijska značajka smanjena u dimenzijama u odnosu na ulaz, a kod drugog je dimenzija ili povećana ili ostaje ista. Kod prvog ovo postižemo primjenom *valid padding*[3], a kod drugog to postižemo primjenom *same padding*[4]. Kada 5x5x1 sliku povećamo na 6x6x1 sliku te na nju primijenimo 3x3x1 kernel, vidjet ćemo da je konvoluirana matrica dimenzija 5x5x1. Odatle naziv *same padding*. S druge strane, ako istu operaciju izvodimo bez da koristimo *padding*[5], dobit ćemo matricu koja ima dimenzije kernela (3x3x1 ili drugim riječima *valid padding*) [12]. *Same padding* možemo vidjeti na slici 3.5.

[3] *Valid padding* znači da nema *padding* te se pretpostavlja da su sve dimenzije validne pa je ulazna slika u potpunosti prekrivena filtrom te ima *stride* kojeg smo postavili. (https://intellipaat.com/community/558/what-is-the-difference-between-same-and-valid-padding-in-tf-nn-maxpool-of-tensorflow, pristup: 17.6.2020.)

[4] *Same padding* primjenjuje *padding* na ulaznu sliku tako da je ulazna slika u potpunosti prekrivena filtrom te ima zadani *stride*. Za svaki *stride* 1, izlaz će biti isti kao i ulaz. (https://intellipaat.com/community/558/what-is-the-difference-between-same-and-valid-padding-in-tf-nn-maxpool-of-tensorflow, pristup. 17.6.2020.)

[5] *Padding* je izraz relevantan za konvolucijske neuronske mreže jer se odnosi na količinu piksela koji se dodaju slici prilikom CNN obrade. (https://deepai.org/machine-learning-glossary-and-terms/padding, pristup: 17.6.2020.)



Slika 3.5

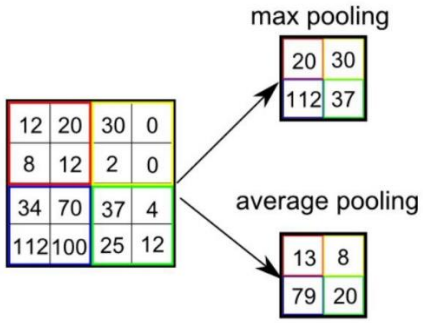
Same padding

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, pristup: 17.6.2020.

Slično konvolucijskom sloju, *pooling* sloj je odgovoran za smanjenje prostorne veličine značajke konvolucije. To se radi da bi se smanjila računska snaga potrebna za obradu podataka. Nadalje, korisno je za izdvajanje dominantnih značajki kojima se ne mijenja rotacija i pozicija kako bi se očuvao proces učinkovitog treniranja modela. Postoje dvije vrste *poolinga*: *max pooling*[6] i *average pooling*[7]. *Max pooling* vraća maksimalnu vrijednost iz dijela slike koji pokriva kernel. *Max pooling* suzbija šum te ujedno smanjuje dimenzije. S druge strane, *average pooling* jednostavno smanji dimenzije te mu to ujedno služi kao mehanizam suzbijanja šuma. *Max pooling* je puno bolji od *average poolinga*. Konvolucijski sloj i *pooling* sloj zajedno tvore *I-ti* sloj konvolucijske neuronske mreže. Ovisno o složenosti slike, broj takvih slojeva može se povećati za još bolje obuhvaćanje detalja na niskim razinama. Nakon gornjih postupaka, modelu je uspješno omogućeno razumijevanje značajki. Nakon toga, izlaz se plasira u neuronsku mrežu kako bi se napravila klasifikacija [12]. Vrste *poolinga* prikazane su na slici 3.6.

[6] *Max pooling* operacija za svaki blok jednostavno izvodi izračun maksimalne vrijednosti te na taj način dobivamo smanjeni prikaz. (https://www.machinecurve.com/index.php/2020/01/30/what-are-max-pooling-average-pooling-global-max-pooling-and-global-average-pooling/, pristup: 17.6.2020.)

[7] *Average pooling* umjesto maksimalne vrijednosti izračunava prosjek za svaki blok pa sadržava puno više informacija o elementima bloka koje su manje važne. (https://www.machinecurve.com/index.php/2020/01/30/what-are-max-pooling-average-pooling-global-max-pooling-and-global-average-pooling/, pristup: 17.6.2020.)

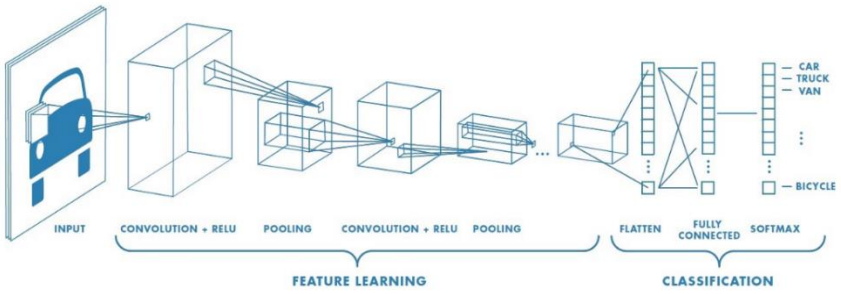


Slika 3.6

Vrste poolinga

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, pristup: 17.6.2020.

Dodavanjem potpuno povezanog sloja je jeftin način učenja nelinearnih kombinacija značajki visokih razina. Potpuno povezani sloj uči moguću nelinearnu funkciju u tom prostoru. Kada je ulazna slika konvertirana u prikladan oblik za više stupanjski perceptron, slika se spljošti u vektorski stupac. Spljoštena slika (izlaz) plasira se u neuronsku mrežu te se primjenjuje *backpropagation*[8] za svaku iteraciju treniranja. Nakon nekog vremena, model je u stanju razlikovati dominirajuće i uobičajene značajke niske razine na slikama te i klasificirati tehnikom *softmax*[9] klasifikacije [12]. Prikaz mreže možemo vidjeti na slici 3.7.



Slika 3.7

Neuronska mreža s puno konvolucijskih slojeva

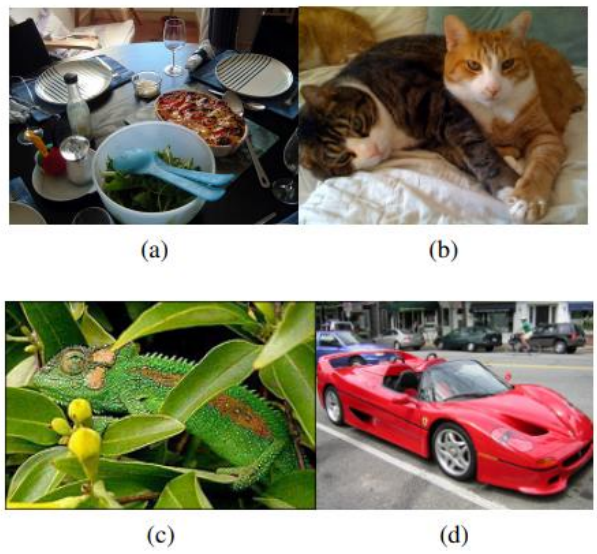
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, pristup: 17.6.2020.

[8] *Backpropagation* je algoritam za učenje s nadzorom kod neuronske mreže koji izračunava nagib gradijenta. Još se naziva i *backward propagation of errors* budući da se pogreške izračunavaju na izlazu te se distribuiraju natrag kroz sloj mreže. (https://deepai.org/machine-learning-glossary-and-terms/backpropagation, pristup: 17.6.2020.)

[9] *Softmax* klasifikatori daju vjerojatnosti za svaku oznaku klase. (https://www.pyimagesearch.com/2016/09/12/softmax-classifiers-explained/, pristup: 17.6.2020.)

3.2 R-CNN

Metoda konvolucijske neuronske mreže s prijedlozima regija (*eng. Region Based Convolutional Neural Network*) postiže izvrstnu točnost detekcije objekata pomoću klasične konvolucijske neuronske mreže [13]. Dugo vremena, objekti su se morali detaljno opisati kako bi se oni mogli identificirati. Zbog toga je nastala segmentacija kojoj je cilj jedinstvena podjela slike pomoću općeg algoritma. No, slike su najčešće hijerarhijske prirode. Slika 3.8 (a) prikazuje salatu i žlice unutar zdjele koja se nalazi na stolu. Nadalje, ovisno o kontekstu, pojam stola na ovoj slici može se odnositi na drvo ili može uključivati sve na stolu. Iz toga možemo zaključiti da sama priroda slike i način kategorizacije objekata su zapravo hijerarhijski. To onemogućava jedinstvenu podjelu objekata za većinu slučajeva. Ovaj problem se rješava korištenjem hijerarhijske podjele. Osim toga što bi segmentacija trebala biti hijerarhijska, koristeći samo jednu strategiju prilikom segmentacije najčešće nije najbolje rješenje. Postoji više razloga zašto bi se regija slike morala grupirati. Na slici 3.8 (b) mačke se mogu odvojiti pomoću boje no tekstura im je ista. Na slici 3.8 (c) kameleon je na slici sličan lišću koje ga okružuje no njegova tekstura se razlikuje. Kod slike 3.8 (d) kotači se razlikuju od automobila i po boji i po teksturi no oni su i dalje sastavni dijelovi automobila. Pojedinačne vizualne karakteristike ne mogu riješiti problem segmentacije [14].



Slika 3.8

Problem segmentacije

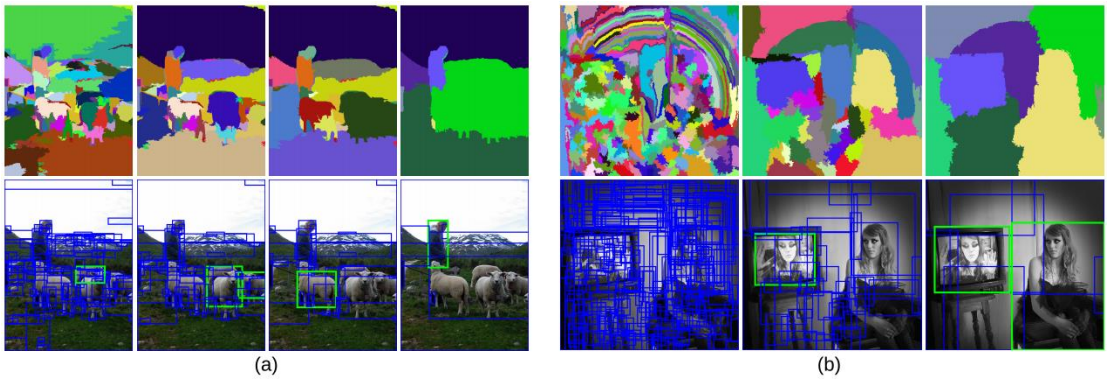
https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013/UijlingsIJCV2013.pdf, pristup: 18.6.2020.

Postoji još osnovniji problem. Regije s vrlo različitim karakteristikama, poput lica i džempera, mogu se kombinirati u jedan objekt tek nakon što se utvrdi da je objekt zapravo čovjek. Dakle, bez prethodnog prepoznavanja teško je prepoznati da su lice i džemper dio jednog objekta. To je dovelo do novog modernijeg pristupa, a to je da se lokalizacija napravi kroz identifikaciju objekata gdje model uči pomoću primjera te pregledava svaku lokaciju unutar slike kako bi mogao odrediti mjesto objekta. Ova metoda ima nekoliko nedostataka. Pretraživanje svake moguće lokacije je teško izvedivo. Prostor za pretraživanje mora se smanjiti koristeći regularnu mrežu, fiksno skaliranje i omjere. U većini slučajeva broj lokacija koje se moraju pregledati je i dalje ogroman pa postoje alternativna ograničenja koja treba nametnuti. Međutim, uniformirano uzorkovanje stvara više prozora što nam odmah daje do znanja da ne podržavaju objekt. Umjesto da na slijepo radimo uzorkovanje lokacije, pitanje je: Da li možemo upravljati uzorkovanjem pomoću analize podataka [14]?

Kako se objekt može nalaziti na bilo kojem položaju i u bilo kojoj skali na slici, normalno je da pretražujemo svaki dio slike. Međutim, vizualni prostor je ogroman, čineći detaljnu računalnu pretragu nepraktičnom. Iz ovog razloga koristi se selektivno pretraživanje. Kod selektivnog pretraživanja koristi se temeljna struktura slike kako bi se generirala lokacija objekta. Ova metoda daje skup lokacija potpuno neovisnih o klasi. Budući da ne koristi fiksni omjer slike, metoda nije ograničena na objekte. Ova metoda generira manje lokacija što olakšava problem pošto je varijabilnost uzoraka manja te se samim time oslobađa računalna snaga koja bi se mogla koristiti za snažnije tehnike strojnog učenja [14].

Selektivni algoritam pretraživanja podliježe slijedećim razmatranjima: [14]

1. Snimanje svake veličine. Objekti se mogu pojavljivati u bilo kojoj veličini unutar slike. Nadalje, pojedini objekti imaju manje jasnu granicu od ostalih objekata. Dakle, u selektivnom pretraživanju sve veličine objekata moraju biti uzete u obzir. To se najbolje postiže pomoću hijerarhijskog algoritma. Ovo možemo vidjeti na slici 3.9.



Slika 3.9

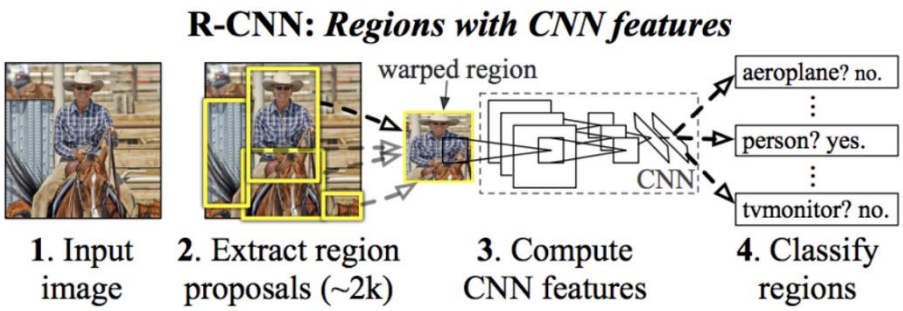
Dva primjera selektivnog pretraživanja koji pokazuju nužnost različitih veličina

https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013/UijlingsIJCV2013.pdf, pristup: 18.6.2020.

2. Raznovrsnost. Ne postoji jedinstvena optimalna strategija za grupiranje regija. Kao što je prikazano na slici 16, regije mogu formirati objekt zbog boje, zbog teksture ili zbog toga što su dijelovi spojeni. Nadalje, uvjeti osvjetljenja kao što su sjene i boja svjetla također mogu utjecati na to kako regije tvore objekt. Stoga, umjesto jedne strategije koja djeluje dobro u većini slučajeva, želimo imati raznovrstan skup strategija za rješavanje svih slučajeva.

3. Brzo za računanje. Cilj selektivnog pretraživanja je dobiti niz mogućih lokacija objekata za praktičnu upotrebu prepoznavanja objekata. Algoritam je relativno brz te stvaranje ovog skupa ne bi trebalo uzeti previše vremena.

Da bi zaobišli problem ogromnog broja regija koristimo metodu selektivnog pretraživanja kako bi smo iz slike izvukli samo 2 000 regija. Stoga, umjesto da pokušavamo klasificirati ogroman broj regija, jednostavno možemo raditi s 2 000 regija. Prijedlog kojih će se 2 000 regija koristiti je generiran pomoću algoritma za selektivno pretraživanje. Selektivna pretraga stvara početnu podsegmentaciju gdje se generiraju regije objekta. Nakon toga se koristi algoritam za rekurzivno kombiniranje sličnih regija u veće regije. Generirane regije koriste se za izradu konačnih prijedloga za regiju nekog objekta [3]. Proces možemo vidjeti na slici 3.10.

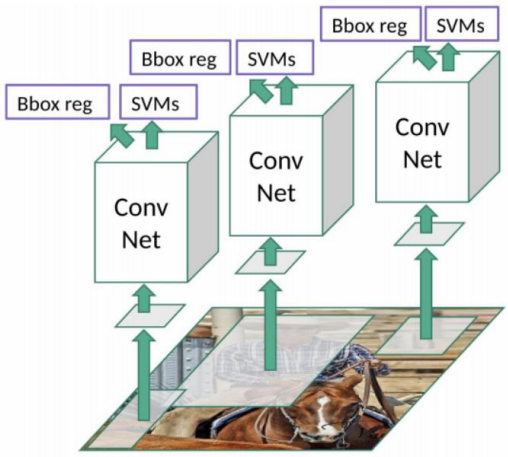


Slika 3.10

R-CNN proces

https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e, pristup: 18.6.2020.

Ovih 2 000 regija posloženo je u kvadrat koji je plasiran u konvolucijsku neuronsku mrežu koja daje 4096-dimenzionalni vektor kao izlaz. CNN se ponaša kao ekstraktor značajki. Izlazni gusti sloj sastoji se od značajki koje su izvađene iz slike. Izdvojene značajke plasiraju se u SVM radi klasificiranja prisutnosti objekta unutar regije. Kako bi se predvidjela prisutnost objekta unutar regije, algoritam također predviđa četiri dodatne vrijednosti te na taj način povećava preciznost graničnog okvira. Na primjer, s obzirom na regiju, algoritam bi predvidio prisustvo osobe. Može se dogoditi da se lice te osobe unutar regije prepolovi. Da se to ne događa, dodatne vrijednosti pomažu u prilagođavanju graničnog okvira te regije [3]. Ovo možemo vidjeti na slici 3.11.



Slika 3.11

R-CNN

https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e, pristup: 18.6.2020.

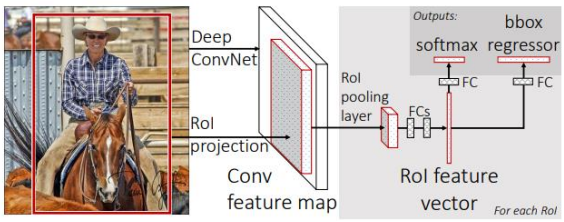
R-CNN još uvijek ima problema s brzinom budući da je potrebno mnogo vremena za osposobljavanje mreže iz razloga što bi smo morali klasificirati 2 000 regija po slici. Također, ne može se implementirati u stvarnom vremenu jer je potrebno oko 47 sekundi za svaku testnu sliku. Selektivni algoritam pretraživanja je fiksni algoritam pa se u toj fazi ne događa nikakvo učenje pa bi moglo doći do stvaranja loših regija [3].

3.3 Brzi R-CNN

Novi algoritam za treniranje koji popravlja nedostatke R-CNNa te istovremeno poboljšava brzinu i točnost. Metoda brzog R-CNNa ima nekoliko prednosti: [13]

1. Bolja kvaliteta detekcije
2. Treniranje se odvija u jednoj fazi
3. Treniranje može ažurirati sve mrežne slojeve
4. Nije potreban disk za spremanje značajki

Brzi R-CNN kao ulaz uzima cijelu sliku te skup objektnih prijedloga. Mreža prvo obrađuje cijelu sliku pomoću više konvolucijskih i *max pooling* slojeva kako bi se izradila konvolucijska mapa značajki. Zatim, za svaki prijedlog objekta, RoI (*eng. Region of Interest*) sloj izdvaja vektor značajki fiksne duljine iz mape značajki. Svaki vektor značajki plasira se u niz potpuno povezanih slojeva koji se granaju u dva izlazna sloja. Jedan sloj koji daje procjene vjerojatnosti nad klasama objekata (*softmax* klasifikator) te drugi sloj koji daje četiri broja za svaku klasu objekta. Svaki skup od četiri vrijednosti predstavlja granični okvir za jednu od klasa [13]. Arhitektura je prikazana na slici 3.12.

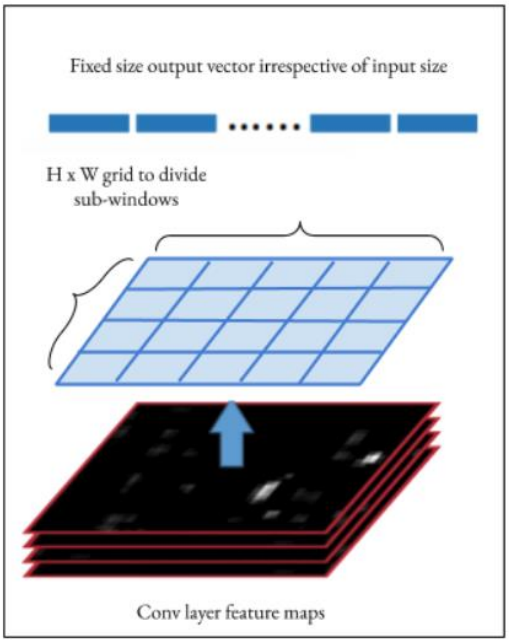


Slika 3.12

Brza R-CNN arhitektura

https://arxiv.org/pdf/1504.08083.pdf, pristup: 19.6.2020.

RoI *pooling*[10] sloj koristi *max pooling* kako bi konvertirao značajke unutar bilo koje valjane regije interesa u manju mapu značajki s fiksnim prostornim opsegom H x W (npr. 7 x 7), gdje su H i W hiper-parametri koji su neovisni o bilo kojem određenom RoIu. RoI je pravokutni prozor koji se nalazi unutar konvolucijske mape značajki. Svaki je RoI definiran s četiri vrijednosti (*r*,*c*,*h*,*w*) koje specificiraju njegov gornji lijevi kut (*r*,*c*) te visinu i širinu (*w*,*h*). RoI *max pooling* radi na način da razdjeli h x w RoI prozor na H x W mrežu potprozora čija je veličina približno h/H x w/W te vrijednosti svakog potprozora stavlja u odgovarajuću ćeliju izlazne mreže [13]. Ovo osigurava da je izlazna značajka fiksne veličine te je neovisna o ulaznoj veličini. H i W odabrani su tako da je izlaz kompatibilan s prvim povezivim slojem mreže. RoI *pooling* provodi se na svakom kanalu pojedinačno [15]. Ovo možemo vidjeti na slici 3.13.



Slika 3.13

RoI pooling

https://towardsdatascience.com/fast-r-cnn-for-object-detection-a-technical-summary-a0ff94faa022, pristup: 19.6.2020.

[10] RoI *pooling* služi kako bi se iskoristila jedinstvena mapa značajki za sve prijedloge koje generira RPN. Kod mreže za otkrivanje objekta rješava problem potrebe fiksne veličine slike. (https://towardsdatascience.com/region-of-interest-pooling-f7c637f409af, pristup: 19.6.2020.)

Za vrijeme treniranja svaka mini serija izgrađena je iz dvije slike. Mini serija sastoji se od 64 RoIa po slici. Prije nego dođe do RoI *pooling* sloja, cijela slika prolazi kroz CNN. Na taj način sve regije interesa s iste slike računaju se zajedno te dijele memoriju kada ulaze i izlaze iz CNNa. Strategija uzorkovanja (uzorkovanje svih RoIa u seriji s vrlo malo slika) omogućava da se cijeli modul (generacija značajki, klasifikacija i regresija) trenira zajedno. Treba imati na umu da je veličina mreže vrlo mala kod CNNa sve do RoI *pooling* sloja te je puna veća kod *softmax* i regresijskih slojeva [15].

Umjesto treniranja linearnog SMVa, brzi R-CNN koristi *softmax* klasifikator koji nauči prilikom treniranja. *Softmax* klasifikator malo nadmašuje SVM klasifikator za sve tri mreže. Ovaj učinak je mali no pokazuje se boljim budući da je moguće dobiti bolje rezultate pomoću samo jedne faze treniranja [13]. Rezultati su prikazani na slici 3.14.

method	classifier	S	M	L
R-CNN [9, 10]	SVM	**58.5**	**60.2**	66.0
FRCN [ours]	SVM	56.3	58.7	66.8
FRCN [ours]	softmax	57.1	59.2	**66.9**

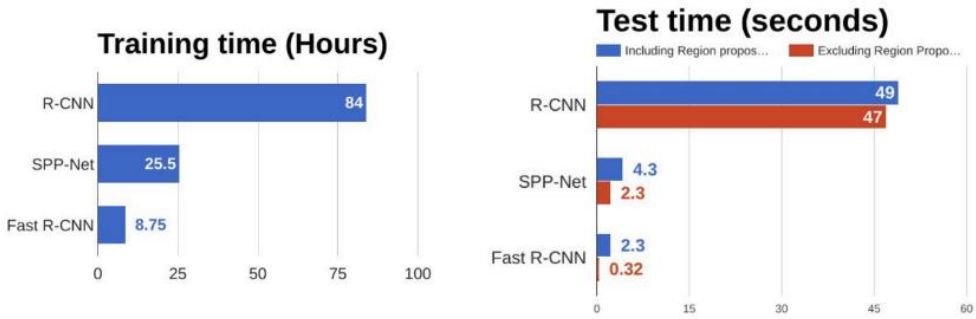
Slika 3.14

Rezultati dobiveni koristeći različite klasifikatore

https://arxiv.org/pdf/1504.08083.pdf, pristup: 19.6.2020.

Što se tiče *multi-task* treniranja, ne samo da je treniranje lakše, već i poboljšava izvedbu iz razloga što zadaci utječu jedni na druge tijekom samog treniranja. Stoga, uvježbavanje različitih faza mreže u potpunosti poboljšava proces [15]. Također, *multi-task* treniranje poboljšava točnost klasifikacije što pokazuje na pozitivan učinak *multi-task* učenja. Brzi R-CNN obrađuje slike 146 puta brže od R-CNNa, a vrijeme treniranja je 9 puta brže [13].

Razlog zašto je brzi R-CNN brži od R-CNNa je taj što ne moramo svaki put unositi 2 000 prijedloga regija u konvolucijsku neuronsku mrežu. Umjesto toga, konvolucija se vrši jednom po slici te se iz nje generira mapa značajki [3]. Usporedba je prikazana na slici 3.15.



Slika 3.15

Usporedba algoritama za detekciju objekata

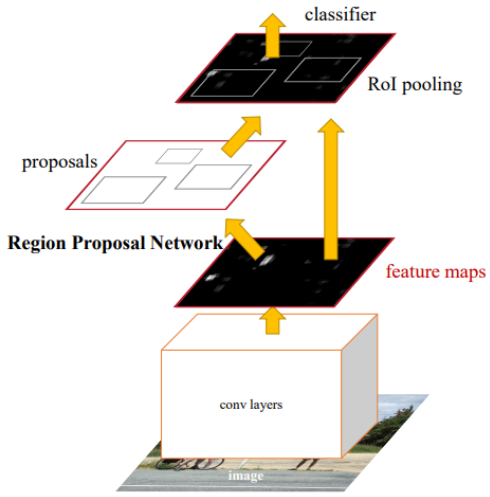
https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e, pristup: 19.6.2020.

Iz gornjih grafova može se zaključiti da je brzi R-CNN znatno brži u treniranju i testiranju. Kada se pogledaju performanse brzog R-CNNa, uključujući prijedloge za regije, vidimo da to znatno usporava algoritam u odnosu na performanse prilikom nekorištenja prijedloga za regije [3].

3.4 Brži R-CNN

Algoritmi poput R-CNNa i brzog R-CNNa koriste selektivno pretraživanje kako bi pronašli prijedloge regija. Selektivna pretraga je spor i dugotrajan proces koji utječe na performanse mreže. Iz tog razloga, osmišljen je novi algoritam za detekciju objekata koji eliminira selektivni algoritam pretraživanja te omogućava mreži da nauči prijedloge regija [3].

Brži R-CNN sastoji se od dva modula. Prvi modul je duboka konvolucijska mreža koja predlaže regije. Drugi modul je detektor brzog R-CNNa koji koristi predložene regije [16]. Kao i kod brzog R-CNNa, kod bržeg R-CNNa slika služi kao ulaz u konvolucijsku mrežu koja pruža konvolucijsku mapu značajki. Umjesto korištenja algoritma selektivnog pretraživanja na mapi značajki za identificiranje prijedloge regija, koristi se zasebna mreža za predviđanje prijedloga regija [3]. Cijeli sustav je jedinstvena, objedinjena mreža za otkrivanje objekata. RPN (*eng. Region Proposal Network)* modul govori brzom R-CNN modulu gdje treba tražiti. Iz ovoga se razvila terminologija neuralnih mreža s mehanizmima pozornosti [16]. Prijedlozi predviđenih regija zatim se preoblikuju pomoću RoI *pooling* sloja koji se zatim koristi kako bi se klasificirale slike unutar predloženog područja te predvidjele vrijednosti pomaka graničnih okvira [3]. Brži R-CNN je prikazan na slici 3.16.

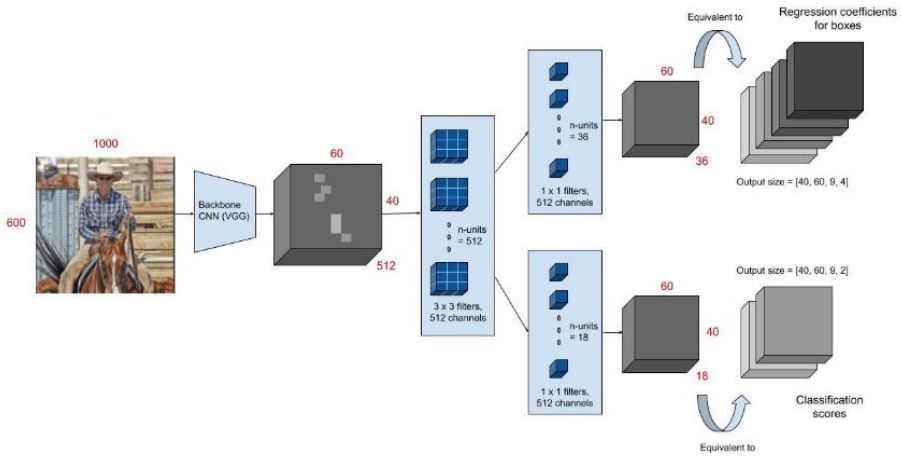


Slika 3.16

Brži R-CNN

https://arxiv.org/pdf/1506.01497.pdf, pristup: 20.6.2020.

Mreža prijedloga regija (RPN) započinje s radom kada se ulazna slika plasira u konvolucijsku neuronsku mrežu. Ulaznoj slici prvo se promijeni veličina na način da je najkraća strana veličine 600px, a dulja strana ne prelazi veličinu od 1 000px. Izlazne značajke mreže (označene s H x W) obično su znatno manje od ulazne slike. Za svaku točku izlazne mape značajki, mreža mora saznati postoji li objekt na ulaznoj slici koji odgovara lokaciji objekta na izlaznoj slici te mora procijeniti njegovu veličinu. To se postiže postavljanjem niza sidrišta na ulaznu sliku za svaku lokaciju izlazne mape značajki. Ta sidrišta označavaju moguće objekte različitih veličina i omjera. Prilikom prolaska kroz svaki piksel na izlaznoj mapi značajki, mreža mora provjeriti da li odgovarajuća sidrišta, koja obuhvaćaju ulaznu sliku, zapravo sadrže objekte te preraditi koordinate tih sidrišta kako bi ograničavajuće okvire interpretirali kao prijedloge objekata ili regije interesa (RoI). Na mapu značajki najprije se primjenjuje 3 x 3 konvolucija s 512 jedinica kako bi se dobila 512-d mapa značajki za svaku lokaciju. Nakon toga slijede dva srodna sloja: 1 x 1 konvolucijski sloj s 18 jedinica za klasifikaciju objekata te 1 x 1 konvolucijski sloj s 36 jedinica za regresiju graničnog okvira. 18 jedinica daje izlaz veličine (W, H, 18). Ovaj izlaz koristi se kako bi procijenio koja je vjerojatnost da svaka točka unutar mape značajke sadrži objekt unutar svih 9 sidrišta u toj točki. 36 jedinica daje izlaz veličine (W, H, 36). Ovaj se izlaz koristi za davanje 4 koeficijenata regresije za svih 9 sidrišta na svakoj točki mape značajki. Ovi koeficijenti regresije koriste se kako bi se poboljšale koordinate sidrišta koje sadrže objekte [17]. Arhitektura je prikazana na slici 3.17.



Slika 3.17

RPN arhitektura

https://towardsdatascience.com/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46, pristup: 20.6.2020.

Izlazna mapa značajki sastoji se od oko 40 x 60 lokacija koje se podudaraju s otprilike 20 000 sidrišta (40*60*9). Prilikom treniranja, sva sidrišta koja prijeđu granicu se zanemaruju kako ne bi doprinijeli gubitku. Time se broj sidrišta koji se uzima po slici znatno smanjuje (s 20 000 sidrišta po slici na 6 000 sidrišta po slici). Sidrište se smatra pozitivnim uzorkom ako zadovoljava bilo koji od dva uvjeta: sidrište ima najveći IoU (*eng. Intersection over Union*) s okvirom ili sidrište ima IoU veći od 0,7 s bilo kojim okvirom. Sidrište se označuje kao negativno ako je njegov IoU s okvirom manji od 0,3. Preostala sidrišta (ni pozitivna ni negativna) ne uključuju se u RPN treniranje. Svaka mini serija za RPN treniranje dolazi iz jedne slike. Uzimanje uzoraka svih sidrišta sa slike usmjerilo bi proces učenja prema negativnim uzorcima. Iz tog razloga, nasumično se odabire 128 pozitivnih i 128 negativnih uzoraka. Ako nema dovoljno pozitivnih uzoraka, uzimaju se dodatni negativni uzorci [17].

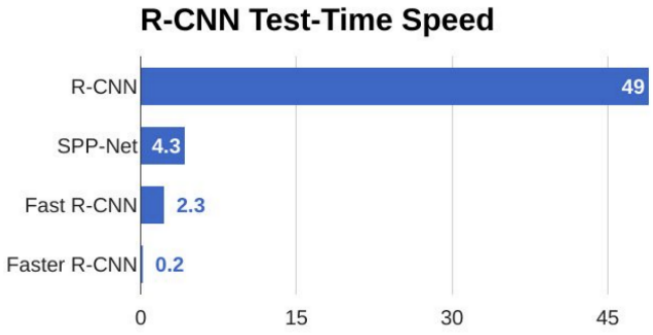
Pošto se RPN i brzi R-CNN treniraju samostalno, to znači da će oni izmijeniti svoje konvolucijske slojeve na različite načine. Iz tog razloga moramo razviti tehniku koja omogućuje dijeljenje konvolucijskih slojeva između dviju mreža. Postoje tri načina treniranja mreža s podijeljenim značajkama: [16]

1. Naizmjenično treniranje. Prvo treniramo RPN te koristimo prijedloge za obuku brzog R-CNNa. Mreža podešena pomoću brzog R-CNNa se koristi kako bi se inicijalizirao RPN te se taj proces ponavlja.

2. Približno zajedničko treniranje. Ovdje se mreže RPNa i brzog R-CNNa spajaju u jednu mrežu tijekom treniranja. Stvaraju se prijedlozi za regije koji se tretiraju kao fiksni te unaprijed izračunati prijedlozi prilikom treniranja brzog R-CNN detektora.

3. Zajedničko treniranje koje nije približno. Granični okviri predviđeni RPNom su ujedno i funkcije ulaza. RoI *pooling* sloj kod brzog R-CNNa prihvaća konvolucijske značajke i predviđene granične okvire kao ulaz. Kod zajedničkog treniranja koje nije približno, treba nam RoI *pooling* sloj koji se razlikuje od *w.r.t.* koordinata okvira.

Brži R-CNN je znatno brži od brzog R-CNNa. Stoga, može se koristiti za detekciju objekata u realnom vremenu [3]. Usporedba brzina je prikazana na slici 3.18.



Slika 3.18

Usporedba brzine između algoritama za detekciju objekata

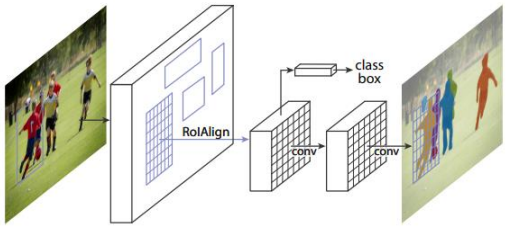
https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e, pristup: 20.6.2020.

3.5 Maskirani R-CNN

Konceptualno jednostavan, fleksibilan i općenit način za segmentaciju instanci objekata. Ovaj pristup učinkovito otkriva objekte na slici te istovremeno stvara visokokvalitetnu segmentacijsku masku za svaku instancu. Metoda maskiranog R-CNNa proširuje brži R-CNN dodavanjem dijela koji predviđa masku objekta paralelno s postojećim dijelovima za prepoznavanje graničnog okvira. Maskirani R-CNN je jednostavan za treniranje i ima malu prednost nad bržim R-CNNom. Osim toga, maskirani R-CNN lako se generalizira za druge zadatke, npr. omogućava nam procjenu poza ljudi na istoj slici [18].

Zajednica koja se bavi računalnim vidom je u vrlo kratkom vremenu poboljšala otkrivanje objekata i semantičku segmentaciju. Velikim dijelom, ova brza napredovanja su postignuta pomoću moćnih sustava kao što su brzi R-CNN, brži R-CNN i potpuno konvolucijske mreže. Segmentacija instance je izazovna jer zahtjeva ispravno otkrivanje svih objekata na slici te precizno segmentiranje svake instance. Iz tog razloga kombinira klasične elemente koje sadrži računalni vid prilikom otkrivanja objekata, gdje je cilj klasificirati pojedinačne objekte te lokalizirati taj objekt pomoću graničnog okvira zajedno s semantičkom segmentacijom gdje je cilj klasificirati svaki piksel u fiksni skup kategorija vez razdvajanja instanci objekta [18].

Kao što smo već napomenuli maskirani R-CNN je konceptualno jednostavan: brži R-CNN ima dva izlaza za svaki objekt, oznaku klase i odstupanje graničnog okvira. Na ovo se dodaje treći dio koji daje masku objekta. Maska R-CNNa je intuitivna ideja no dodatni izlaz maske razlikuje se od izlaza klase i okvira pa zahtijeva izdvajanje objekta s dobrim prostornim rasporedom. Zatim, uvode se ključni elementi maskiranog R-CNNa, uključujući *pixel-to-pixel* poravnanje koji je glavni nedostatak brzog i bržeg R-CNNa. Maskirani R-CNN adoptira procedure bržeg R-CNNa s identičnom prvom fazom (a to je RPN). U drugoj fazi, paralelno s predviđanjem klase i pomaka okvira, maskirani R-CNN također daje binarnu masku za svaki RoI. Nakon toga slijedi i pristup koji paralelno primjenjuje klasifikaciju i regresiju graničnog okvira [18]. Maskirani R-CNN prikazan je na slici 3.19.



Slika 3.19

Maskirani R-CNN

https://arxiv.org/pdf/1703.06870.pdf, pristup: 21.6.2020.

Kod treninga, definira se *multi-task* gubitak za svaki uzorkovani RoI (formula 3.2): [18]

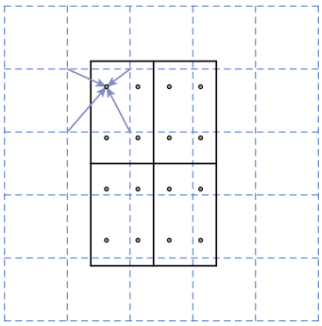
$$L = L_{cls} + L_{box} + L_{mask}, \tag{3.2}$$

gdje L_{cls} predstavlja gubitak klasifikacije, a L_{box} gubitak graničnog okvira. Maska ima $K\ m^2$ dimenzionalni izlaz za svaki RoI koji kodira K binarnih maski rezolucije $m \times m$, po jednu za svaku od K klasa. Na to apliciramo *per-pixel sigmoid* te definiramo L_{mask} kao prosječni binarni *cross-entropy*[11] gubitak. Za RoI povezan s *ground-truth*[12] klasom k, L_{mask} je definiran samo na k-toj maski (ostali izlazi maske ne doprinose gubitku). Ovakva L_{mask} definicija omogućava mreži generiranje maske za svaku klasu. To se postiže pomoću namjenske klasifikacije, kako bi se predvidjela oznaka klase koja se koristi za odabir izlazne maske [18].

Maska kodira prostorni raspored ulaznih objekata. Dakle, izdvajanje prostorne strukture maske može se riješiti pomoću *pixel-to-pixel* korespondencije koju nam daje konvolucija. Konkretno, predviđamo $m \times m$ masku za svaki RoI koristeći potpunu konvolucijsku mrežu. To svakom sloju u maski omogućava da održi određen $m \times m$ prostorni raspored objekta bez da ga pretvara u vektorski prikaz koji nema prostornu dimenziju. Potpuno konvolucijska mreža zahtjeva manje parametara te je točnija. *Pixel-to-pixel* zahtjeva da se značajke RoIa, koje su zapravo male mape značajki, dobro poravnaju kako bi očuvali *pixel-to-pixel* prostornu korespondenciju. RoI *align*[13] sloj igra ključnu ulogu u predviđanju maski [18].

[11] *Cross-entropy* je mjera koja se temelji na entropiji te općenito izračunava razliku između dvije vjerojatnosti. (https://machinelearningmastery.com/cross-entropy-for-machine-learning/, pristup: 21.6.2020.)
[12] *Ground-truth* provjerava ispravnost rezultata strojnog učenja u odnosu na stvarni svijet. (https://www.techopedia.com/definition/32514/ground-truth, pristup: 21.6.2020.)
[13] Glavna razlika između RoI *pooling* i RoI *align* je u kvantizaciji. (https://towardsdatascience.com/understanding-region-of-interest-part-2-roi-align-and-roi-warp-f795196fc193, pristup: 21.6.2020.)

RoI *pooling* je standardni postupak koji se koristi za izdvajanje mape značajki iz svakog RoIa. No, kod njega se događa kvantizacija što zapravo dovodi do neusklađivanja između RoIa i izdvojenih značajki. Iako to možda ne utječe na klasifikaciju, imati će negativan utjecaj na predviđanje *pixel-accurate* maski. Iz ovog razloga, uvodi se RoI *align* sloj (slika 3.20). On uklanja kvantizaciju koja se događa kada koristimo RoI *pooling* postupak te pravilno poravnava izdvojene značajke s ulazom. Zapravo time izbjegavamo bilo kakvo kvantiziranje RoI granica. Za računanje točnih vrijednosti ulaznih značajki koristi se bilinearna interpolacija. Uzimaju se četiri uzorkovane lokacije u svakom RoIu te se računa rezultat (koristeći maksimalnu ili srednju vrijednost) [18].



Slika 3.20

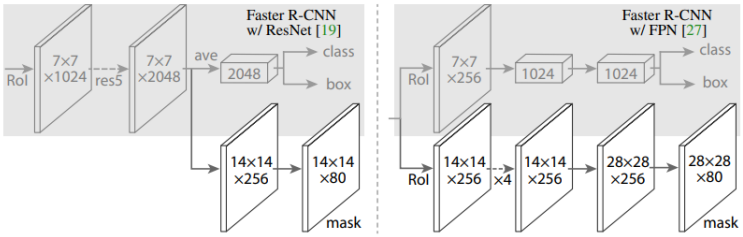
Isprekidana mreža predstavlja mapu značajki, pune linije predstavljaju RoI, a točke predstavljaju četiri uzorkovane lokacije. RoI align izračunava vrijednost svake lokacije uzorkovanja pomoću bilinearne interpolacije. Kvantizacija se ne provodi niti u jednoj fazi.

https://arxiv.org/pdf/1703.06870.pdf, pristup: 21.6.2020.

Kako bi demonstrirali ovaj pristup, instancirati ćemo maskirani R-CNN s više arhitektura. Da bi bilo jasnije, napravljena je razlika između: Konvolucijske *backbone*[14] arhitekture koje se koriste za izdvajanje značajki preko cijele slike, glave mreže za prepoznavanje graničnog okvira (klasifikacija i regresija) te predikcije maske koja se primjenjuje zasebno za svaki RoI. Definiramo *backbone* arhitekturu koristeći značajke dubinske mreže. Procjenjujemo 50 ili 101 sloj ResNeta i ResNeXt mreža. Originalna implementacija bržeg R-CNNa sa ResNet izdvajala je značajke iz završnog konvolucijskog sloja četvrte faze koju mi još nazivamo i *C4*. *Backbone* sa ResNet-50, na primjer, definiran je sa ResNet-50-C4 [18].

[14] *Backbone* se odnosi na mrežu koja uzima sliku kao ulaz te izvlači mapu značajki na kojoj se temelji ostatak mreže. (https://stats.stackexchange.com/questions/397767/how-do-backbone-and-head-architecture-work-in-mask-r-cnn, pristup: 21.6.2020.)

Također istražujemo i još jedan učinkovitiji *backbone* koji se naziva FPN (*eng. Feature Pyramid Network*). FPN koristi *top-down*[15] arhitekturu s bočnim vezama kako bi se napravila piramidalna mreža značajki iz ulaza jedne skale. Brži R-CNN s FPN *backbone* izdvaja RoI značajke iz različitih razina piramidalnih značajki u skladu s njihovom skalom. Koristeći ResNet-FPN *backbone* za izdvajanje značajki zajedno s maskiranim R-CNNom dobivamo izvrsne rezultate što se tiče točnosti i brzine. Za glavu mreže pomno pratimo arhitekture koje su navedene iznad te dodajemo konvolucijsko predviđanje maske. Zapravo, proširujemo brži R-CNN (slika 3.21) [18].



Slika 3.21

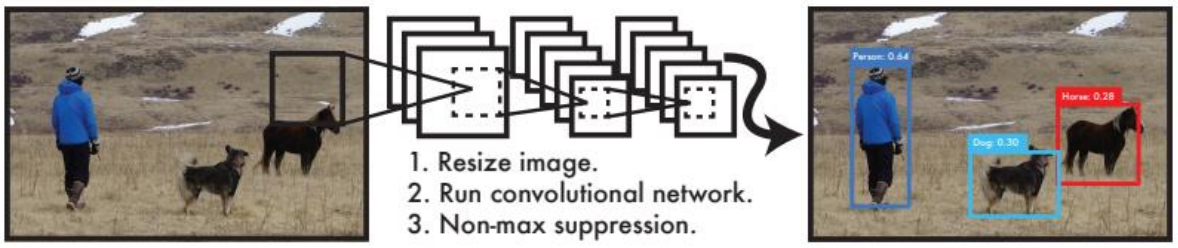
Arhitektura glave. Proširujemo dvije postojeće brže R-CNN glave. Slike prikazuju glave s ResNet C4 i FPN backbones, brojevi označavaju prostornu rezoluciju i kanale, strelice označavaju konvolucijske ili potpuno konvolucijske slojeve. https://arxiv.org/pdf/1703.06870.pdf, pristup: 21.6.2020.

3.6 YOLO

Ljudi bace pogled na sliku i odmah znaju koji se objekti nalaze na slici, gdje se nalaze te koje interakcije rade. Ljudski vizualni sustav je brz i precizan, omogućava nam obavljanje složenih zadataka poput vožnje bez previše razmišljanja. Brzi i precizni algoritmi za otkrivanje objekata omogućili bi računalima da upravljaju automobilima bez specijaliziranih senzora. Trenutni sustavi za detekciju prilagođavaju klasifikatore prilikom obavljanja detekcije. Za otkrivanje objekata, takvi sustavi uzimaju klasifikator za taj objekt te ga procjenjuju na raznim lokacijama i skalama na testnoj slici. Noviji pristupi poput R-CNNa koriste metode prijedloga regije kako bi generirali potencijalni granični okvir na slici te nakon toga pokreću klasifikator na tim predloženim okvirima. Ovaj sustav je složen i spor te ih je teško optimizirati budući da se svaka pojedinačna komponenta mora zasebno trenirati [19].

[15] Top-down arhitektura služi za raščlanjivanje problema na manje uporabljive cjeline ili dijelove. (https://www.computerscience.gcse.guru/theory/top-down-design, pristup: 18.9.2020.)

Pomoću sustava YOLO (*eng. You Only Look Once*) slika se pogleda samo jednom kako bi se predvidjelo koji su predmeti prisutni na toj slici te gdje se nalaze. Detekcija objekta rekonstruirana je kao jedinstveni regresijski problem, izravno od piksela slike do koordinata graničnog okvira i vjerojatnosti klasa. YOLO je osvježavajuće jednostavan. Jedna konvolucijska mreža istovremeno predviđa više graničnih okvira te vjerojatnost klase za te okvire. YOLO radi treniranje na cjelovitim slikama te izravno optimizira performanse detekcije. Ovaj unificirani model ima nekoliko prednosti nad tradicionalnim metodama za otkrivanje objekata. Najprije, YOLO je izuzetno brz budući da se detekcija smatra kao regresijski problem pa ne moramo koristiti složeni sustav. Prilikom testiranja slike, jednostavno pokrenemo neuronsku mrežu kako bi predvidjeli detekcije. Osnovna mreža radi na 45 sličica po sekundi bez serijske obrade. To znači da možemo obraditi videozapis u stvarnom vremenu koji ima kašnjenje manje od 25 milisekundi. YOLO postiže dvostruku prosječnu preciznost od ostalih sustava. Također, kada YOLO radi pretpostavku o slici, on sliku razmatra na globalnoj razini. Za razliku od kliznih prozora i prijedloga regija, YOLO vidi cijelu sliku prilikom treniranja pa samim time implicitno kodira kontekstualne informacije o klasama i izgled klasa. YOLO radi manje od polovice pozadinskih grešaka u odnosu na brzi R-CNN. Budući da je YOLO dobro generaliziran, manja je vjerojatnost da neće raditi kada se na njega primjeni nove domene ili neočekivani unosi. YOLO još uvijek zaostaje za najsuvremenijim sustavima detekcije kad je u pitanju točnost detekcije. Iako može brzo prepoznati objekte na slikama, još uvijek se muči prilikom preciznog lokaliziranja pojedinih objekata [19]. YOLO sustav prikazan je na slici 3.22.



Slika 3.22

YOLO sustav za detekciju. Najprije se mijenja veličina ulazne slike, zatim se pokreće jedna konvolucijska mreža te se formiraju rezultati detekcije.

https://pjreddie.com/media/files/papers/yolo_1.pdf, pristup: 24.6.2020.

Kao što je već spomenuto, kod YOLO sustava objedinimo zasebne komponente detekcije objekata u jednu neuronsku mrežu. Mreža koristi značajke cijele slike kako bi predvidjela svaki granični okvir. Također, istovremeno predviđa sve granične okvire svih klasa za svaku sliku. To znači da mreža razmatra sliku i sve predmete na slici na globalnoj razini. YOLO sustav dijeli ulaznu sliku na *S x S* mrežu. Ako središte objekta padne na ćeliju mreže ta ćelija će biti odgovorna za otkrivanje tog objekta. Svaka ćelija mreže predviđa granične okvire *B* i pouzdanost rezultata za te okvire. Pouzdanost rezultata (formula 3.3) odražava koliko je model uvjeren da okvir sadrži neki objekt te kolika je točnost okvira. Pouzdanost rezultata možemo definirati kao: [19]

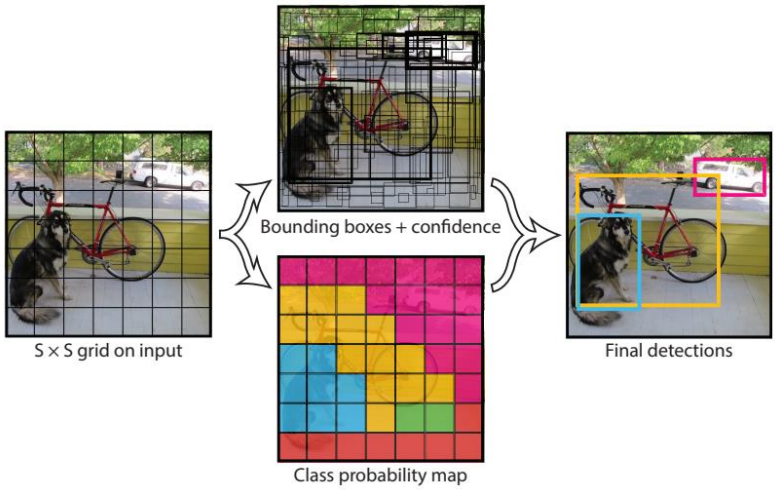
$$Pr(Object) * IoU_{pred}^{truth}, \tag{3.3}$$

gdje želimo da je pouzdanost rezultata jednaka sjecištu nad unijom (IoU) između predviđenog okvira i *ground-truth*. Ako u toj ćeliji ne postoji niti jedan objekt, pouzdanost rezultata bi trebala biti nula. Svaki granični okvir sastoji se od 5 predviđanja: *x,y,w,h* te pouzdanost. Koordinate *(x,y)* predstavljaju središte okvira u odnosu na graničnu ćeliju mreže. Širina i visina se predviđa u odnosu na cijelu sliku. Svaka ćelija mreže također predviđa vjerojatnosti uvjetne klase $Pr(Class_i|Object)$. Vjerojatnosti su uvjetovane na mreži ćelije koja sadrži objekt. Predviđa se samo jedan skup vjerojatnosti klase po ćeliji mreže bez obzira na broj okvira *B* [19].

Za vrijeme testiranja množe se vjerojatnosti uvjetne klase i predviđanje pouzdanosti za pojedinačne okvire (formula 3.4): [19]

$$Pr(Class_i|Object) * Pr(Object) * IoU_{pred}^{truth} = Pr(Class_i) * IoU_{pred}^{truth}, \tag{3.4}$$

što nam daje pouzdanost rezultata za svaku klasu okvira. Ovi rezultati kodiraju vjerojatnost te klase koja se pojavljuje u okviru i koliko dobro predviđeni okvir odgovara objektu na slici [19]. Ovo možemo vidjeti na slici 3.23.

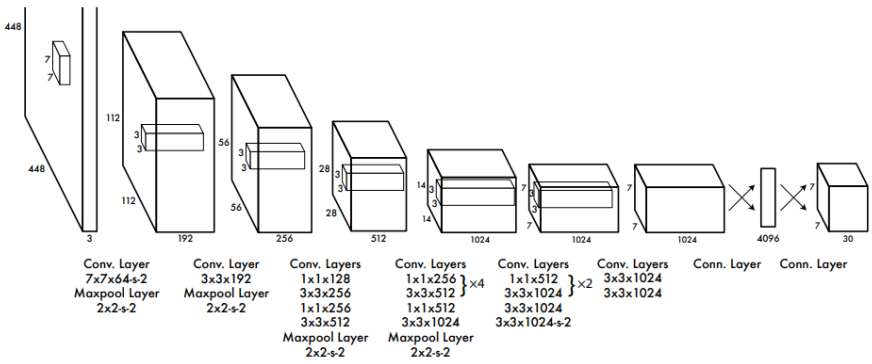


Slika 3.23

Sustav podijeli sliku na S x S mrežu te za svaku ćeliju mreže predviđa granične okvire B, pouzdanost za te okvire i vjerojatnost klase C

https://pjreddie.com/media/files/papers/yolo_1.pdf, pristup: 24.6.2020.

Ovaj model je implementiran kao konvolucijska neuronska mreža. Početni konvolucijski sloj izdvaja značajke sa slike dok potpuno povezani slojevi predviđaju izlazne vrijednosti i koordinate. Ova mreža ima 24 konvolucijska sloja i dva potpuno povezana sloja. Jednostavno se upotrebljavaju *1x1* redukcijski slojevi koji se nastavljaju na *3x3* konvolucijska sloja. Također je i trenirana i brza verzija YOLOa koja je namijenjena kako bi pomakli granice kod brzog otkrivanja objekata. Brzi YOLO koristi neuronsku mrežu s manje konvolucijskih slojeva (koristi 9 slojeva umjesto 24) te manje filtera u tim slojevima. Svi ostali parametri prilikom treniranja i testiranja su isti između YOLOa i brzog YOLOa [19]. Ova arhitektura je prikazana na slici 3.24.



Slika 3.24

Arhitektura mreže

https://pjreddie.com/media/files/papers/yolo_1.pdf, pristup: 24.6.2020.

Joseph Redmon, Santoshi Divvala, Ross Girshick i Ali Farhadi za treniranje koriste 20 konvolucijskih slojeva zajedno s prosječnim *pooling* slojem i potpuno povezanim slojem. Mrežu treniraju otprilike tjedan dana te koriste *Darknet* za treniranje i iznošenje zaključaka. Zatim, model prilagode kako bi otkrivao objekte. Pokazalo se da dodavanjem konvolucijskih i povezanih slojeva na trenirane mreže može poboljšati performanse. Iz tog razloga, dodaju 4 konvolucijska sloja i dva potpuno povezana sloja s nasumično inicijaliziranim važnostima. Detekcija najčešće zahtjeva dobre vizualne informacije pa se ulazna razlučivost mreže poveća s *224x224* na *448x448*. Završni sloj predviđa vjerojatnost klase i koordinate graničnog okvira. Normalizira se visina i širina graničnog okvira prema visini i širini slike tako da se oni nalaze između vrijednosti 0 i 1. Nakon toga, dodaju se parametri graničnom okviru za x i y koordinate na način da predstavljaju odstupanja od određene lokacije ćelije mreže što znači da su ćelije mreže također ograničene između vrijednosti 0 i 1. Na završnom sloju koriste linearnu aktivacijsku funkciju dok na svim ostalim slojevima koriste propusno ispravljenu linearnu aktivacijsku funkciju (formula 3.5): [19]

$$\emptyset(x) = \begin{cases} x, & if\ x > 0 \\ 0.1\,x, & otherwise \end{cases}. \tag{3.5}$$

Izlaz modela optimiziraju pomoću pogreške zbroja kvadrata. Pogreška zbroja kvadrata se lako optimizira no ne podudara se savršeno s maksimiziranjem prosječne preciznosti. Lokacijska greška je ujednačena greškom klasifikacije što nije idealno. Također, na svakoj slici mnoge ćelije mreže ne sadrže ni jedan objekt pa to gura rezultate pouzdanosti prema nuli, često nadvladavajući gradijent ćelija koje sadrže objekte. To može dovesti do nestabilnosti modela. Da bi smo to riješili, povećamo gubitak predviđanja koordinata graničnog okvira te smanjimo gubitak pouzdanog predviđanja za okvire koji ne sadrže objekte. Za to koristimo dva parametra, λ_{coord} i λ_{noobj}. Postavimo $\lambda_{coord} = 5$ te $\lambda_{noobj} = 0.5$. Pogreška u zbroju kvadrata jednako važi za pogreške u velikim i malim okvirima. Ovakav pokazatelj pogreške trebao bi pokazati da su manja odstupanja kod velikih okvira manje važna od odstupanja u malim okvirima. Kako bi to djelomično riješili, predviđa se kvadratni korijen širine i visine graničnog okvira umjesto izravne visine i širine [19].

YOLO predviđa više graničnih okvira po ćeliji mreže. Za vrijeme treninga želimo da samo jedan granični okvir bude odgovoran za predviđanje svakog objekta. Da bi to postigli, dodjeljuje se jedan prediktor čije predviđanje ima najveći IoU. To dovodi do specijalizacije između prediktora graničnih okvira. Svaki prediktor postaje bolji u predviđanju određenih veličina, omjera i klasa objekata [19].

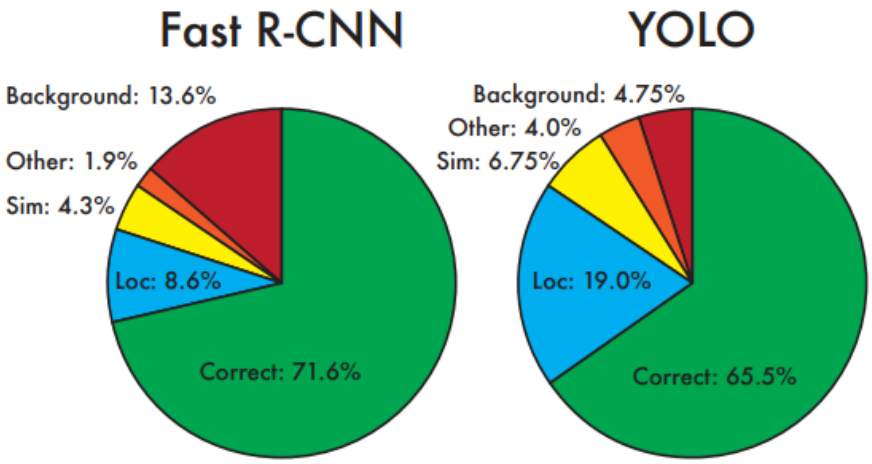
Tijekom treniranja optimizira se višedijelna funkcija gubitka. Funkcija gubitka sankcionira grešku klasifikacije samo ako je objekt prisutan u toj ćeliji mreže. Također, sankcionira i grešku koordinate graničnog okvira ako prediktor, koji je odgovoran za taj okvir, ima najveći IoU od bilo kojeg prediktora u toj ćeliji mreže [19].

Detekcija objekata je temeljni problem računalnog vida. Sustav za detekciju uobičajeno započinje s izdvajanjem skupa robusnih značajka ulazne slike. Zatim se koriste klasifikatori ili lokalizatori kako bi se identificirali objekti. Ti klasifikatori ili lokalizatori pokreću se na kliznom prozoru preko cijele slike ili na nekim područjima na slici. Usporedimo YOLO sustav za otkrivanje objekata s nekoliko ključnih sličnosti i razlika. DPM (*eng. Deformable part models*) koriste klizni prostor za otkrivanje objekata Također, DPM koristi odvojeni sustav kako bi izdvojio statične značajke, klasificirao regije, predvidio granične okvire. YOLO sustav zamjenjuje sve te različite dijelove s jednom konvolucijskom neuronskom mrežom. Mreža istovremeno provodi izdvajanje značajki, predviđanje graničnog okvira, kontekstualno razumijevanje. Mreža trenira značajke te ih optimizira za detekciju. Ovakva objedinjena arhitektura dovodi do bržeg i preciznijeg modela nego DPM. R-CNN i njegove inačice koriste prijedloge regija umjesto kliznih prozora kako bi pronašle objekte na slikama. Selektivnom pretragom generiraju se potencijalni granični okviri, mreža izdvaja značajke, SVM daje okvire, a linearni model podešava granične okvire dok *non-max supression*[16] uklanja duple detekcije. Svaka faza ovog složenog sustava mora se zasebno prilagoditi što rezultira time da je sustav dosta spor. YOLO dijeli neke sličnosti s R-CNNom. Svaka ćelija mreže predlaže potencijalne granične okvire i bilježi te okvire pomoću konvolucijskih značajki. Međutim, ovaj sustav stavlja prostorna ograničenja na prijedloge ćelija mreže što pomaže da se smanji višestruko otkrivanje istog objekta. Ovaj sustav također predlaže daleko manje graničnih okvira, samo 98 po slici u usporedbi s 2 000 kod selektivne pretrage [19].

[16] *Non-max supression* je tehnika koja se koristi u mnogim algoritmima računalnog vida. To je klasa algoritma pomoću koje se odabire jedan entitet (npr. granični okvir) iz mnogih preklapajućih entiteta. (https://medium.com/@whatdhack/reflections-on-non-maximum-suppression-nms-d2fce148ef0a, pristup: 24.6.2020)

YOLO sustav kombinira pojedinačne komponente u jedinstveni, zajednički optimizirani model. Brzi i brži R-CNN usredotočeni su na ubrzavanje R-CNNa na način da koriste neuronsku mrežu umjesto da predlažu regije selektivne pretrage. Iako nude brzinu i poboljšanje točnosti, jedno i drugo je daleko od performansi u stvarnom vremenu [19].

YOLO se još uvijek muči s ispravnom lokalizacijom objekata. Pogreške lokalizacije kod YOLOa su veće od svih ostalih sustava kombinirano. Brzi R-CNN radi znatno manje pogreške kod lokalizacije no radi puno više pozadinskih pogrešaka. 13.6% najboljih detekcija je lažno pozitivnih te ne sadrže objekte [19]. Analizu pogreške možemo vidjeti na slici 3.25.



Slika 3.25

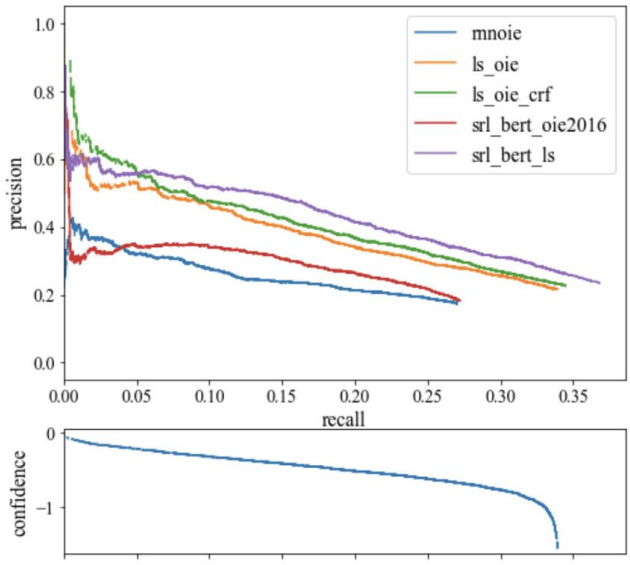
Analiza pogreške:Brzi R-CNN u odnosu na YOLO

https://pjreddie.com/media/files/papers/yolo_1.pdf, pristup: 24.6.2020.

3.7 Mjerenje točnosti detekcije

Kako bi usporedili performanse sustava za detekciju objekata kod računalnog vida, koristimo mjernu jedinicu mAP (*eng. mean Average Precision*). U nastavku je objašnjeno na koji načina se mAP izračunava te zašto je mAP mjerna jedinica postala preferirana kod detekcije objekata [20].

Da bismo razumjeli mAP, najprije moramo pojasniti *precision-recall* krivulju. *Precision* je mjera koja predstavlja koliko često naš model ispravno procjenjuje kada radi pretpostavku. *Recall* je mjera koja govori da li je naš model napravio pretpostavku svaki puta kada je trebao napraviti pretpostavku. Modeli koji uključuju i element pouzdanosti mogu povećati *precision* prilagođavanjem razine pouzdanosti koja im je potrebna kako bi se napravila procjena. Drugim riječima, ako je model u situaciji u kojoj je izbjegavanje lažno pozitivnih rezultata važnije od izbjegavanja lažno negativnih, može se postaviti viši prag pouzdanosti kako bi se model potaknuo da radi procjenu s visokom preciznošću. Naravno, time se smanjuje *recall* [20]. Ovo možemo vidjeti na slici 3.26.



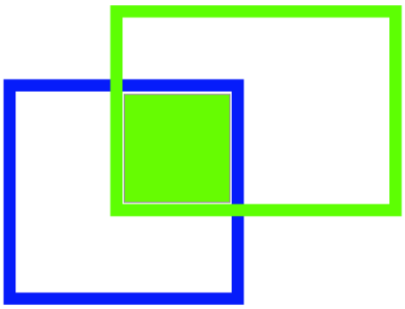
Slika 3.26

Precision-recall krivulja i pouzdanost

https://towardsdatascience.com/what-is-mean-average-precision-map-in-object-detection-8f893b48afd3, pristup: 27.6.2020.

Precision-recall krivulja je silazna iz razloga što se pouzdanje više smanjuje to se više pretpostavki napravi (pomažući *recall*) te je manja preciznost procjene (šteti *precision*). O ovome možete razmišljati na način da ako neko kaže da nabrojite svaku vrstu morskog psa, počeli biste s očiglednim (veća preciznost) no postali biste manje samopouzdani sa svakom dodatnom vrstom morskog psa koji biste mogli imenovati (veći *recall* s manjom preciznošću). Kako je model sve manje uvjeren, krivulja se naginje prema dolje. Ako model ima uzlaznu krivulju, tada najčešće model ima problema s procjenom pouzdanosti. *Precision-recall* krivulja daje AP (*eng. Average Precision*) [20].

Sustavi za detekciju objekata rade pretpostavke pomoću graničnog okvira i oznake klase. U praksi, granični okviri koji su predviđeni u koordinatama X1, X2, Y1, Y2 su uvijek malo pomaknuti od *ground-truth* oznake. Znamo da bismo predviđeni granični okvir trebali smatrati netočnim ako je klasa pogrešna, no gdje bismo trebali povući crtu prilikom preklapanja graničnog okvira? IoU je mjerna jedinica za postavljanje ove granice. IoU se računa kao količina graničnog okvira koji se preklapa s *ground-truth* graničnim okvirom podijeljen s ukupnom površinom oba okvira [20]. Grafički prikaz za IoU možemo vidjeti na slici 3.27.

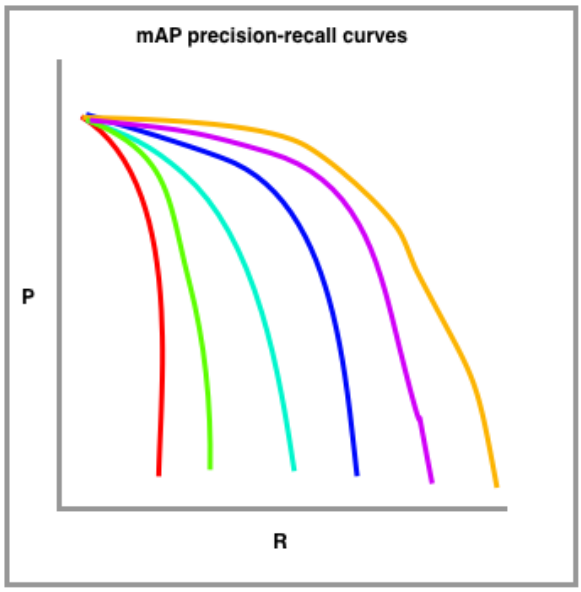


Slika 3.27

Grafički prikaz IoU

https://towardsdatascience.com/what-is-mean-average-precision-map-in-object-detection-8f893b48afd3, pristup: 27.6.2020.

Da bismo izračunali mAP, moramo imati niz *precision-recall* krivulja s IoU pragom postavljenim na različitim razinama [20]. Ovo je prikazano na slici 3.28.



Slika 3.28

mAP precision-recall krivulje

https://towardsdatascience.com/what-is-mean-average-precision-map-in-object-detection-8f893b48afd3, pristup: 27.6.2020.

Crvena krivulja predstavlja najviše zahtjeve za IoU (oko 90%), narančasta krivulja predstavlja najniže zahtjeve za IoU (oko 10%). Broj krivulja koje treba nacrtati obično se postavlja *izazovom.* Na primjer, COCO *izazov* postavlja 10 različitih pragova za IoU počevši od 0,5 te se povećava sve do 0,95 u koracima od 0,05. Na kraju, ove krivulje se nacrtaju za svaku vrstu klase. Izračunava se AP za svaku klasu pojedinačno za sve IoU pragove. Nakon toga izračunava se mAP za sve klase kako bi se došlo do konačne procjene [20].

3.8 YOLO v3

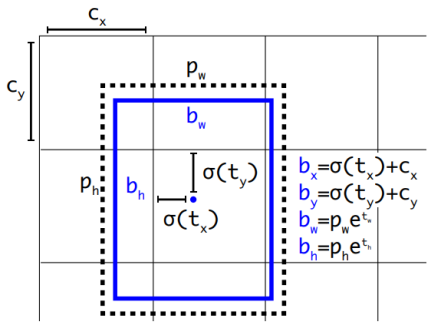
Sadrži novu mrežu klasifikatora koji su bolji od ostalih. Sustav predviđa granične okvire pomoću dimenzijskih klastera koji služe kao sidrišta (slika 3.29). Mreža predviđa 4 koordinate za svaki granični okvir *tx, ty, tw, th* (formule 3.6 – 3.9). Ako se ćelija pomakne od gornjeg lijevog kuta slike (*cx, cy*), a prethodni granični okvir ima širinu i visinu *pw, ph* tada predviđanja korespondiraju: [21]

$$b_x = \sigma(t_x) + c_x \tag{3.6}$$

$$b_y = \sigma(t_y) + c_y \tag{3.7}$$

$$b_w = p_w e^{t_w} \tag{3.8}$$

$$b_h = p_h e^{t_h} \tag{3.9}$$



Slika 3.29

Granični okviri s dimenzijama i predviđenim lokacijama

https://pjreddie.com/media/files/papers/YOLOv3.pdf, pristup: 29.6.2020.

YOLO v3 predviđa *objectness score*[17] za svaki granični okvir koristeći logističku regresiju. *Objectness score* bi trebao biti 1 ako se prethodni granični okvir preklapa s *ground-truth* objektom bolje od bilo kojeg prethodnog graničnog okvira. Ako prethodni granični okvir nije najbolji ali se preklapa s *ground-truth* objektom koji prelazi granicu tada zanemarujemo predviđanje. Koristi se granica od 0.5. YOLO v3 dodjeljuje samo jedan prethodni granični okvir za svaki *ground-truth* objekt. Ako prethodni granični okvir nije dodijeljen *ground-truth* objektu tada to ne predstavlja nikakav gubitak za predviđanje koordinata ili klase, već samo za objekte [21].

[17] *Objectness score* je mjera koja definira koliko dobro detektor prepoznaje lokacije i klase objekta. (https://arxiv.org/ftp/arxiv/papers/1909/1909.05626.pdf, pristup: 29.6.2020.)

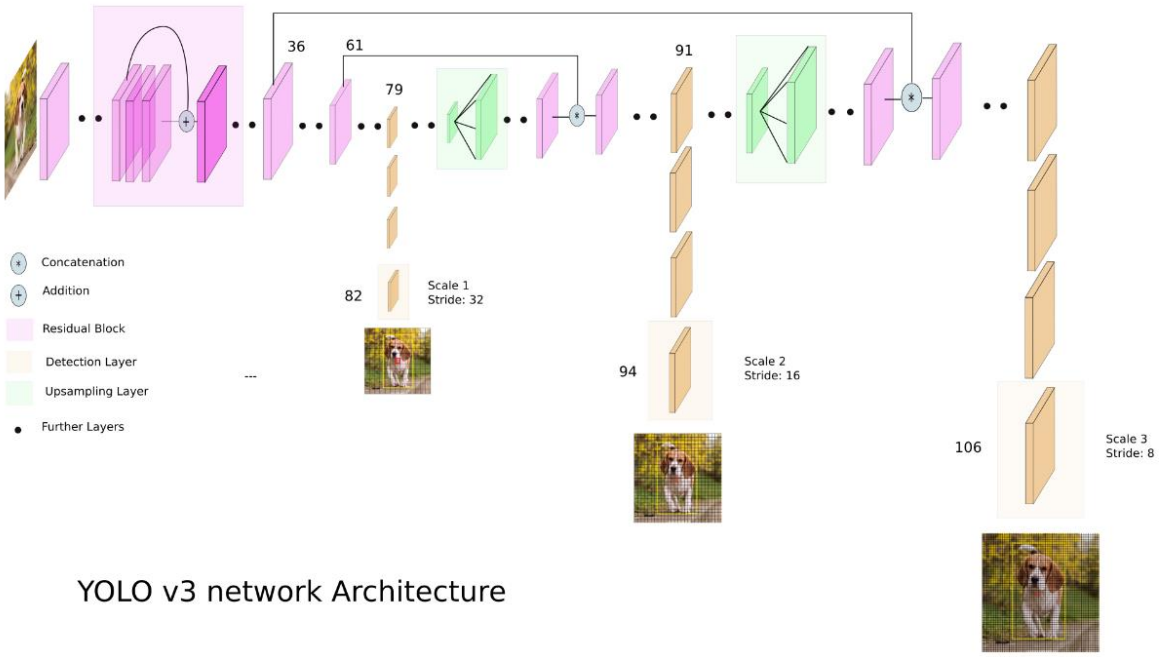
Pomoću višestruke klasifikacije, svaki okvir predviđa klase koje bi granični okvir mogao sadržavati. *Softmax* se ne koristi jer nije potreban za dobre performanse. Umjesto toga koristi se binarni *cross-entropy* gubitak za predviđanje klase. Ova formulacija pomaže kada prelazimo na složenije domene poput skupa podataka. U tom skupu podataka postoji mnogo oznaka koje se preklapaju (npr. žena i osoba). Upotreba *softmaxa* nameće pretpostavku da svaki okvir ima točno jednu klasu što najčešće nije slučaj. YOLO v3 radi pretpostavke okvira na temelju 3 veličine. Sustav izvlači značajke tih veličina pomoću koncepta sličnog FPNu. Iz osnovnog izdvajanja značajki dodaje se nekoliko konvolucijskih slojeva. Zadnji predviđa *3-d* tenzor kodiranja graničnog okvira, objekte i klase. Joseph Redmon i Ali Farhadi koriste COCO kako bi predvidjeli 3 granična okvira za svaku veličinu pa tenzor izgleda (formula 3.10): [21]

$$N\ x\ N\ x\ [3 * (4 + 1 + 80)], \tag{3.10}$$

za 4 odstupanja graničnih okvira, 1 predikciju objekata te 80 predikcija klasa. Također se u mrežu uzima i mapa značajki od prije te se spoji s povećanim značajkama koristeći metodu spajanja. Ova metoda nam omogućuje da dobijemo više smislene semantičke informacije iz povećanih značajka te podataka iz prethodne mape značajki. Nakon toga se dodaje još nekoliko konvolucijskih slojeva kako bi se obradila ova kombinirana mapa značajki te na kraju predviđamo sličan tenzor koji je dvostruko veći. Zatim ovo ponovimo još jednom kako bi predvidjeli okvire za zadnju veličinu. Na taj način, predviđanje za treću veličinu beneficira od svih prethodnih računanja [21].

Predviđanje na različitim slojevima pomaže u rješavanju problema koji se javlja prilikom detekcije malih objekata. Povećani slojevi spojeni s prethodnim slojevima pomažu u očuvanju značajki koje zatim pomažu kod detekcije malih objekata. Prvi sloj je odgovoran za detekciju velikih objekata, drugi sloj za detekciju srednjih objekata te teći sloj za detekciju manjih objekata. YOLO v3 ukupno koristi 9 sidrišta, 3 za svaku veličinu. Ako se YOLO trenira na vlastitom skupu podataka trebali bi koristiti *k-means clustering*[18] kako bi generirali 9 sidrišta. Zatim se sidrišta slože u silaznom redoslijedu dimenzija te se dodijele 3 najveća sidrišta za prvu veličinu, sljedeća 3 za drugu veličinu te posljednja 3 za treću veličinu [22]. Arhitekturu možemo vidjeti na slici 3.30.

[18] *K-means clustering* jedan je od najjednostavnijih i najpopularnijih algoritama kod strojno učenja bez nadzora te se odnosi na zbirku podataka koji se spajaju zajedno zbog određenih sličnosti. (https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1, pristup: 29.6.2020.)



Slika 3.30

Arhitektura YOLO v3 mreže

https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b, pristup: 29.6.2020.

Kada je rađen YOLO v3 isprobavale su se različite stvari koje nisu uspjele: [21]

1. Predviđanje pomaka sidrišta okvira. Pokušao se upotrijebiti normalan mehanizam predviđanja sidrišta okvira gdje se predviđa *x,y* pomak kao cijela visina i širina okvira pomoću linearne aktivacije. Ova formulacija smanjuje stabilnost modela i nije najbolje funkcionirala.

2. Linearno *x,y* predviđanje umjesto logističkog predviđanja. Pokušala se upotrijebiti linearna aktivacija kako bi se izravno predvidjelo *x,y* odstupanje. To je dovelo do pada kod mAP.

3. Žarišni gubitak. Pokušao se iskoristiti žarišni gubitak no to je dovelo do pada mAP. YOLO v3 možda već rješava problem koji žarišni gubitak pokušava riješiti budući da ima zasebno predviđanje objekata i uvjetno predviđanje klasa.

YOLO v3 je poprilično dobar. AP vrijednost je usporediva s SSD (*eng. Single Shot Detector*) varijantama te je 3 puta brži. Još uvijek dosta zaostaje za ostalim modelima poput Retinaneta. Međutim, kada pogledamo „stariju“ mjernu jedinicu mAP kod IoU = .5 (ili AP_{50} u grafikonu) vidimo da je YOLO v3 vrlo snažan. Gotovo je izjednačen s RetinaNetom te je daleko iznad SSD varijante. Ovo pokazuje da je YOLO v3 vrlo snažan detektor kojem je odlika stvaranje poprilično dobrih okvira za objekte. Međutim, performanse znatno padaju kako IoU prag raste. Što pokazuje da se YOLO v3 muči da okvire savršeno uskladi s objektom. Sa novim predviđanjima s više veličina vidimo da YOLO v3 ima relativno visoku AP_S performansu. Međutim, ima dosta loše performanse kada su u pitanju objekti srednje ili veće veličine. Vidimo da YOLO v3 značajne prednosti u odnosu na druge sustave detekcije što ga samim time čini brži i boljim [21]. Usporedbu možemo vidjeti na slici 3.31.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [5]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [8]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [6]	Inception-ResNet-v2 [21]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [20]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [15]	DarkNet-19 [15]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [11, 3]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [3]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [9]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [9]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Slika 3.31

Usporedba modela

https://pjreddie.com/media/files/papers/YOLOv3.pdf, pristup: 29.6.2020.

3.9 YOLO v3 tiny

Detektor za objekte, YOLO, često se navodi kao jedan od najbržih detektora temeljenih na dubokom učenju. Međutim, čak i uz svu tu brzinu, YOLO v3 još uvijek nije dovoljno brz za korištenje na određenim uređajima poput Raspberry Pi. Kako bi YOLO v3 bio još brži, Redmon je definirao varijaciju YOLO arhitekture koja se zove YOLO v3 tiny. Naravno, loša strana je ta što je YOLO v3 tiny manje precizan od YOLO v3, budući da je YOLO v3 tiny manja verzija od YOLO v3 verzije. Samo kao referenca, prema Redmonovom izvješću, YOLO v3 ima 51-58% mAP na COCO datasetu, dok YOLO v3 tiny iznosi samo 33,1% mAP, što je gotovo napola manje točnosti od YOLO v3. Međutim, 33% mAP još uvijek je dovoljno dobar kod nekih primjena [23].

Za razliku od YOLO v3, koji koristi DarkNet53, YOLO v3 tiny koristi Darknet19 *backbone* [23]. Strukturu YOLO v3 tiny mreže možemo vidjeti na slici 3.32.

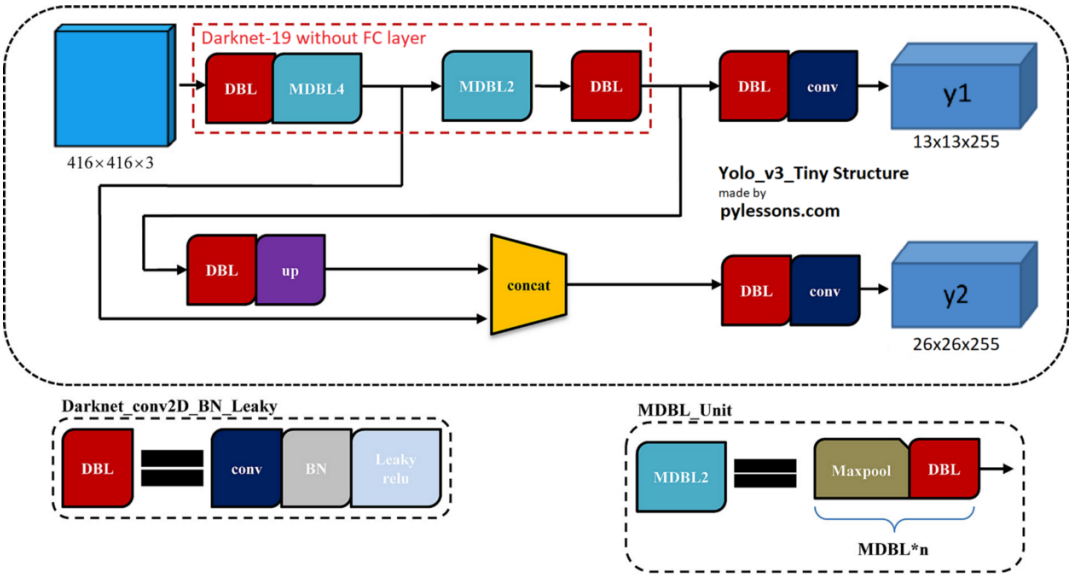
Layer	Type	Filters	Size/Stride	Input	Output
0	Convolutional	16	3 × 3/1	416 × 416 × 3	416 × 416 × 16
1	Maxpool		2 × 2/2	416 × 416 × 16	208 × 208 × 16
2	Convolutional	32	3 × 3/1	208 × 208 × 16	208 × 208 × 32
3	Maxpool		2 × 2/2	208 × 208 × 32	104 × 104 × 32
4	Convolutional	64	3 × 3/1	104 × 104 × 32	104 × 104 × 64
5	Maxpool		2 × 2/2	104 × 104 × 64	52 × 52 × 64
6	Convolutional	128	3 × 3/1	52 × 52 × 64	52 × 52 × 128
7	Maxpool		2 × 2/2	52 × 52 × 128	26 × 26 × 128
8	Convolutional	256	3 × 3/1	26 × 26 × 128	26 × 26 × 256
9	Maxpool		2 × 2/2	26 × 26 × 256	13 × 13 × 256
10	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
11	Maxpool		2 × 2/1	13 × 13 × 512	13 × 13 × 512
12	Convolutional	1024	3 × 3/1	13 × 13 × 512	13 × 13 × 1024
13	Convolutional	256	1 × 1/1	13 × 13 × 1024	13 × 13 × 256
14	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
15	Convolutional	255	1 × 1/1	13 × 13 × 512	13 × 13 × 255
16	YOLO				
17	**Route 13**				
18	Convolutional	128	1 × 1/1	13 × 13 × 256	13 × 13 × 128
19	Up-sampling		2 × 2/1	13 × 13 × 128	26 × 26 × 128
20	**Route 19 8**				
21	Convolutional	256	3 × 3/1	13 × 13 × 384	13 × 13 × 256
22	Convolutional	255	1 × 1/1	13 × 13 × 256	13 × 13 × 256
23	YOLO				

Slika 3.32

YOLO v3 tiny struktura mreže

https://medium.com/analytics-vidhya/tensorflow-2-yolov3-tiny-object-detection-implementation-c3ea5d4d0510, pristup: 13.8.2020.

Gledajući YOLO v3 tiny strukturu, ne možemo u potpunosti razumjeti sve slojeve te kako ih implementirati. Iz ovog razloga, donja slika prikazuje cijelu arhitekturu mreže. Na slici vidimo da ulazna slika veličine 416x416, nakon ulaska u Darknet19 mrežu, dobiva 2 grane. Te grane prolaze kroz niz konvolucija, spajanja i drugih operacija [23]. Arhitektura je prikazana na slici 3.33.



Slika 3.33

Arhitektura YOLO v3 tiny

https://medium.com/analytics-vidhya/tensorflow-2-yolov3-tiny-object-detection-implementation-c3ea5d4d0510, pristup: 13.8.2020.

Ako bi uspoređivali rezultate s izvornim YOLO v3 modelom, YOLO v3 tiny radi puno lošije. Iako je 70% brži od YOLO v3, točnost otkrivanja objekata je dosta loša. Prije nego što počnemo koristiti jedan od ovih modela, moramo odlučiti što nam je više potrebno, brzina ili točnost [23].

4. Praktični dio zadatka

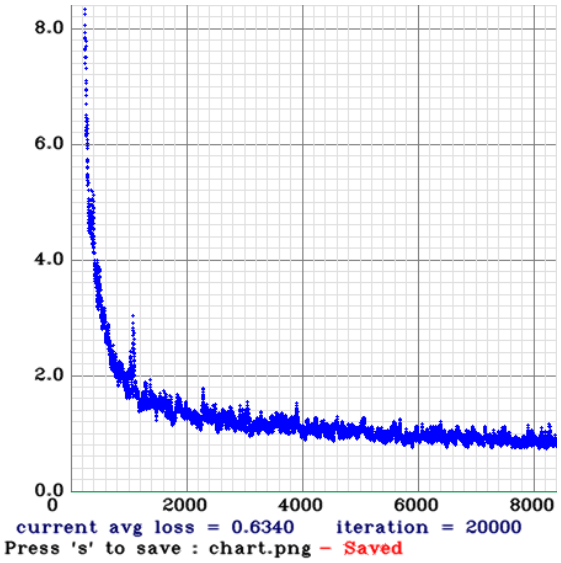
U praktičnom dijelu zadatka, fokus će biti na neuronskoj mreži YOLO v3 tiny. Kako bi YOLO v3 tiny neuronska mreža mogla prepoznati, tj. detektirati lice, neuronska mreža je najprije trenirana. Treniranje, validacija, testiranje i istraživanje se provelo pomoću: besplatnog radnog okruženja Darknet, interpretiranog i objektno orijentiranog programskog jezika Python, integriranog razvojnog okruženja Microsoft Visual Studio, besplatne zbirke datoteka (modula) za računalni vid OpenCV, Open Images V6 dataseta te komandne linije Windows PowerShell. Također, prilikom treniranja korišteno je CUDA ubrzanje kako bi se znatno smanjilo vrijeme pojedinog treninga.

Provedeno je više različitih treninga kako bi se istražili parametri koji mogu utjecati na krajnji rezultat. Kod treniranja su prikazani i uspoređeni grafovi koji su dobiveni prilikom provođenja treniranja s više različitih parametara, kod testiranja je prikazano kako ti rezultati izgledaju na stvarnoj slici, a kod evaluacije je prikazana i napravljena usporedba rezultata za mAP_{50} i mAP_{75}. Na kraju je napravljena usporedba s nekim od postojećih metoda za detekciju lica.

4.1 Treniranje s jednom klasom

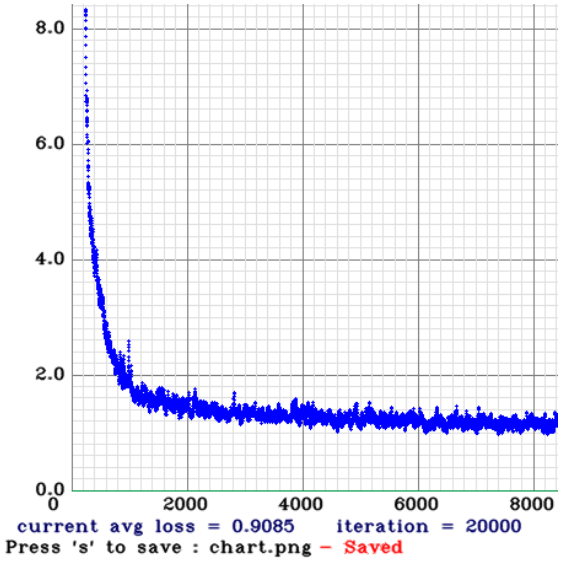
Slike za treniranje su preuzete sa Open Images V6 dataseta. Za dataset je korištena klasa ljudskog lica. Važno je napomenuti da prije korištenja dataseta moramo konvertirati anotacije za klasu koju koristimo u datasetu (u ovom slučaju anotacije za klasu ljudskog lica). Najprije je napravljen dataset s 1 000 slika po klasi. Taj dataset je korišten da se provede više treninga s različitim parametrima kako bi se istražilo da li različiti parametri utječu na krajnji rezultat. Nakon toga, napravljen je dataset s 5 000 slika po klasi kako bi se istražilo da li i broj slika utječe na krajnji rezultat. Također, istražilo se da li pomoću prosječnog gubitka prilikom treninga s jednom klasom možemo predvidjeti krajnji rezultat. Nakon što je napravljen dataset, u programu Microsoft Visual Studio editirana je *yolov3-tiny.cfg* datoteka. Prilikom ovog treniranja, važno je napomenuti da je datoteka *yolov3-tiny.cfg* bila samo oslonac te da se ona nije koristila za samo treniranje. Parametri datoteke *yolov3-tiny.cfg* su promijenjeni u svrhu ovog istraživanja te se prilikom ovog istraživanja koristila datoteka *yolov3-tiny-custom.cfg*. Kod treniranja, kao *backbone* je korišten *yolov3-tiny.conv.11*.

Neuronska mreža *yolov3-tiny-custom* s jednom klasom (ljudsko lice) ali s različitim parametrima (slike 4.1 – 4.6).



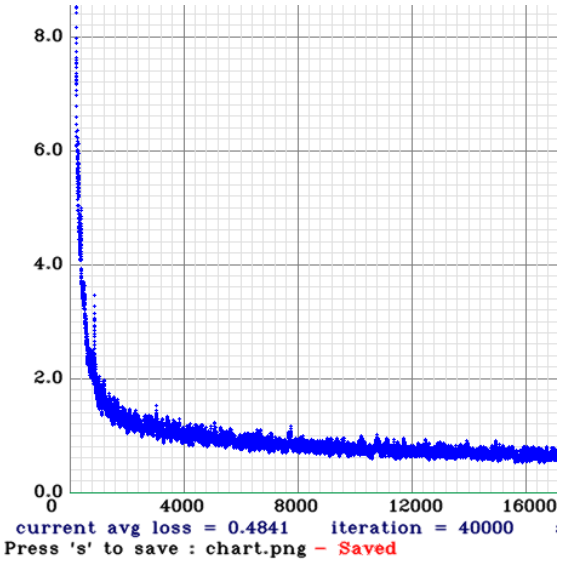
Slika 4.1

Prosječni gubitak za YOLO v3 tiny custom1



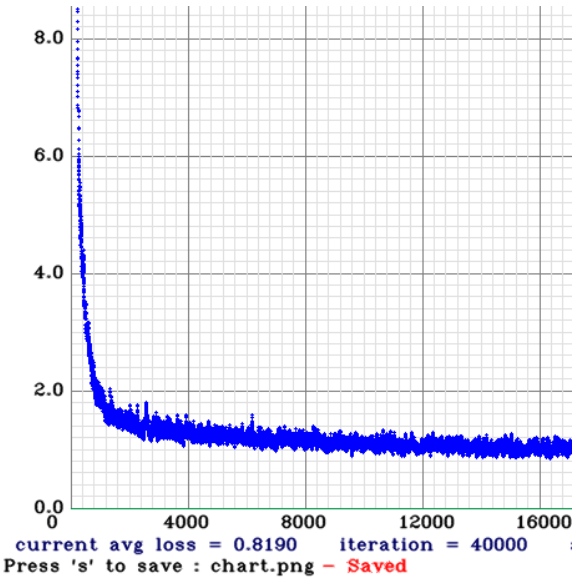
Slika 4.2

Prosječni gubitak za YOLO v3 tiny custom2



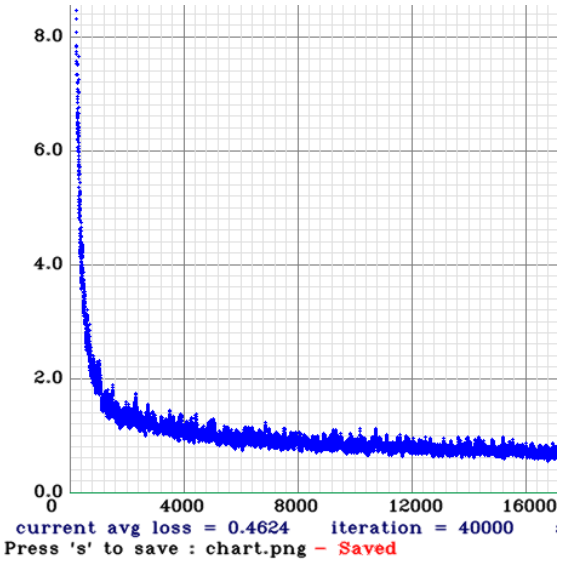
Slika 4.3

Prosječni gubitak za YOLO v3 tiny custom3



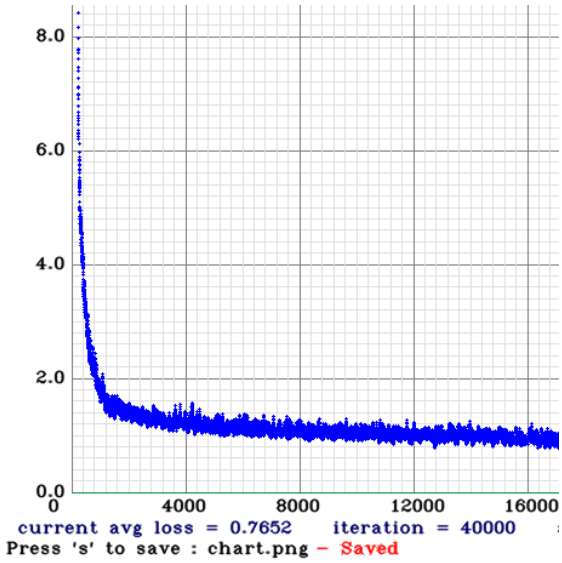
Slika 4.4

Prosječni gubitak za YOLO v3 tiny custom4



Slika 4.5

Prosječni gubitak za YOLO v3 tiny custom5



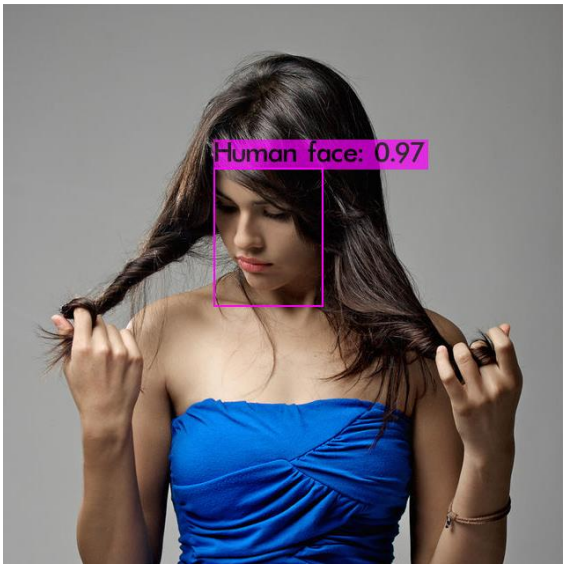
Slika 4.6

Prosječni gubitak za YOLO v3 tiny custom6

Prilikom *custom1* treninga korišteno je 1 000 slika po klasi s veličinom slika 416 x 416 te brojem iteracija od 20 000. Ako ovo usporedimo s *custom3* treningom kod kojega je za trening korišteno 1 000 slika po klasi s veličinom slika 416 x 416 te brojem iteracija od 40 000, možemo uočiti da je prosječni gubitak za *custom3* puno manji. Također, ako usporedimo i *custom3* trening s *custom5* treningom kod kojega je za trening korišteno 1 000 slika po klasi s veličinom slika 608 x 608 te brojem iteracija od 40 000, možemo uočiti da se prosječni gubitak još smanjio. No, ako *custom1* trening usporedimo s *custom2* treningom kod kojega je za trening korišteno 5 000 slika po klasi s veličinom slika 416 x 416 te brojem iteracija od 20 000, možemo uočiti da je prosječni gubitak kod *custom2* treninga puno veći. Isto se događa kada usporedimo *custom3* trening s *custom4* treningom (5 000 slika po klasi s veličinom slika 416 x 416 te brojem iteracija od 40 000) i kada usporedimo *custom5* trening s *custom6* treningom (5 000 slika po klasi s veličinom slika 608 x 608 te brojem iteracija od 40 000). Da li se iz toga može zaključiti da će detektor kod kojeg je korišteno 1 000 slika za treniranje biti precizniji od detektora kod kojeg je korišteno 5 000 slika za treniranje? Da li se prosječni gubitak treninga s 1 000 slika uopće može uspoređivati s prosječnim gubitkom treninga u kojem smo koristili 5 000 slika? Da li broj slika, veličina slike i broj iteracija utječu na samu preciznost detektora odnosno na krajnji rezultat? Da bi smo odgovorili na ova pitanja, nakon treninga najprije moramo provesti testiranje te nakon toga izračunati mAP.

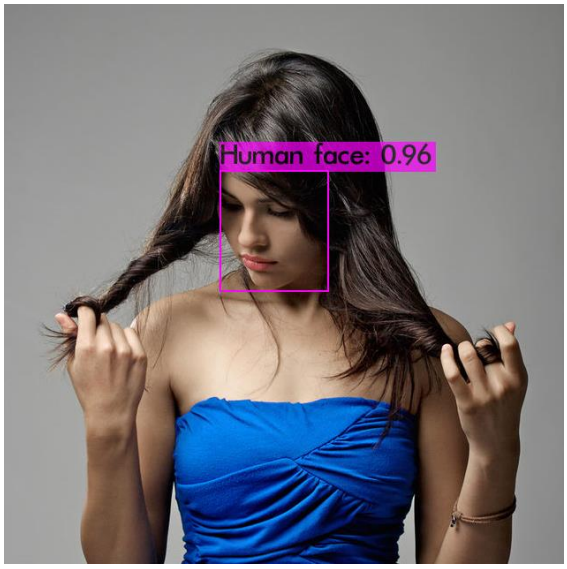
4.2 Validacija i testiranje s jednom klasom

Važno je napomenuti da je prilikom validacije i testiranja korišteno 10% slika od ukupnog dataseta. Također, važno je napomenuti da prilikom validacije i testiranja nisu korištene iste slike koje su korištene prilikom treniranja. Klasa, u ovom slučaju ljudsko lice, je ostala ista. Razlog tome je što ako za testiranje koristimo iste slike koje smo koristili i za treniranje, dobit ćemo bolje rezultate nego što oni zapravo jesu. U nastavku su prikazani i uspoređeni rezultati koji su dobiveni nakon testiranja. Važno je napomenuti da je ovdje izdvojena samo jedna slika, a ne cijeli testni dataset, kako bi se dao prikaz testiranja te kako bi utvrdili da li detektor detektira klasu za koju je istreniran. Ista testna slika analizirana je i s modelima koji su imali različite parametre.



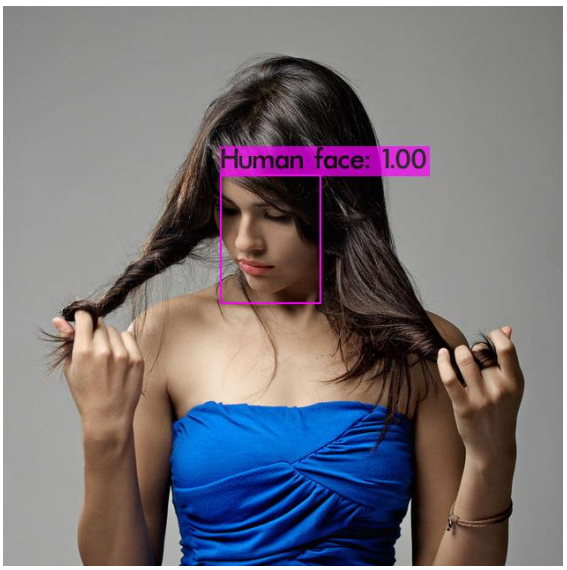
Slika 4.7

Testiranje za YOLO v3 custom1



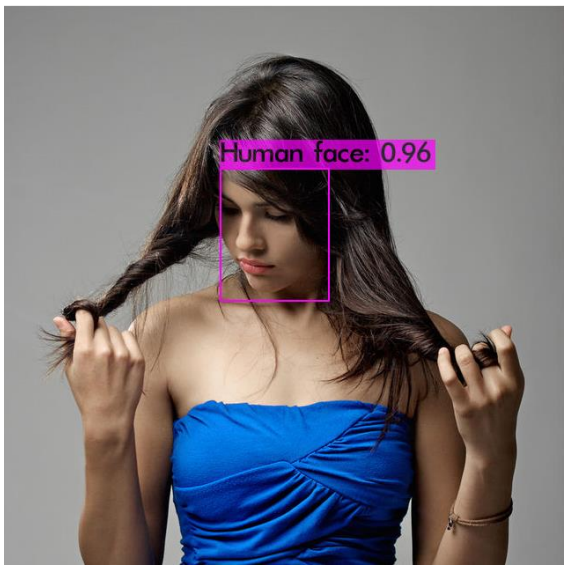
Slika 4.8

Testiranje za YOLO v3 custom2



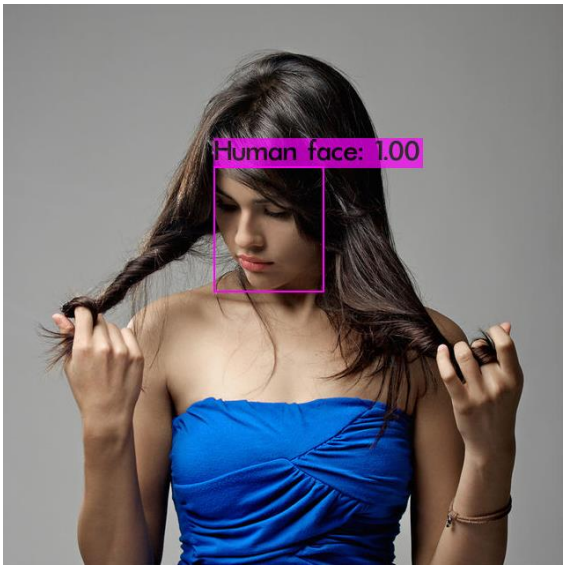
Slika 4.9

Testiranje za YOLO v3 custom3



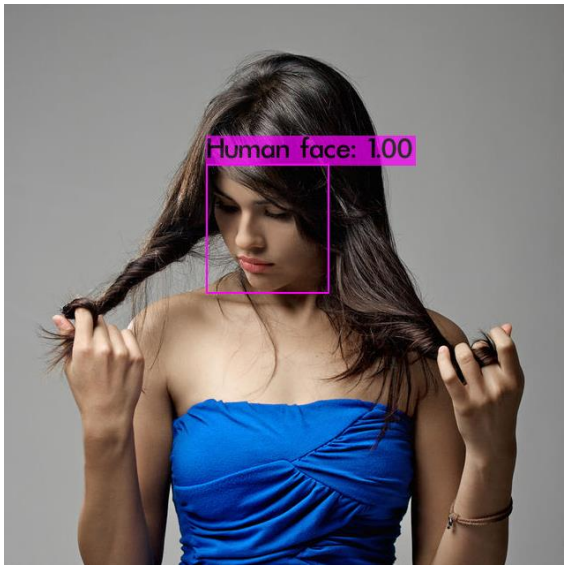
Slika 4.10

Testiranje za YOLO v3 custom4



Slika 4.11

Testiranje za YOLO v3 custom5



Slika 4.12

Testiranje za YOLO v3 custom6

Pomoću graničnog okvira možemo zaključiti da svih šest detektora detektiraju lice (slike 4.7 – 4.12). Također, svih šest detektora detektiraju lice s dosta velikom pouzdanošću (barem za ovu sliku). Vidimo da *cutom1* ima veću pouzdanost od *custom2* te *custom3* ima veću pouzdanost od *custom4*. Veća pouzdanost za jednu sliku ne mora nužno značiti da su modeli *custom1* i *custom3* bolji od modela *custom2* i *custom4*. Da bismo utvrdili koji su modeli bolji, moramo izračunati mAP.

4.3 mAP rezultati za 1 klasu (evaluacija)

Važno je napomenuti da je prilikom izračunavanja mAP vrijednosti korišteno 10% slika od ukupnog dataseta. Također, važno je napomenuti da prilikom evaluacije nisu korištene iste slike koje su korištene prilikom treniranja. Klasa, u ovom slučaju ljudsko lice, je ostala ista. Razlog tome je što ako za evaluaciju koristimo iste slike koje smo koristili i za treniranje, dobit ćemo bolje rezultate nego što oni zapravo jesu. U nastavku je izračunat mAP_{50} i mAP_{75} za jednu klasu te su analizirani i uspoređeni rezultati. Također, prikazano je i što se dogodi kada se za evaluaciju koriste iste slike koje su se koristile i za trening.

model	backbone	mAP_{50}	mAP_{75}
YOLO v3 tiny custom1	yolov3-tiny.conv.11	54.5	28.8
YOLO v3 tiny custom2		63.5	40.4
YOLO v3 tiny custom3		53.8	32.1
YOLO v3 tiny custom4		64.5	41.6
YOLO v3 tiny custom5		54.6	33.8
YOLO v3 tiny custom6		65.4	44.5

Slika 4.13

Usporedba YOLO v3 tiny modela s jednom klasom

Ako pogledamo sliku 4.13 te opet usporedimo model *custom1* (1 000 slika po klasi s veličinom slika 416 x 416 te brojem iteracija od 20 000) i *custotm3* (1 000 slika po klasi s veličinom slika 416 x 416 te brojem iteracija od 40 000) vidimo da je mAP@IoU=50 za *custom1* veća no mAP@IoU=75 za *custom1* je manja. Ako usporedimo *custom3* i *custom5* (1 000 slika po klasi s veličinom slika 608 x 608 te brojem iteracija od 40 000) vidimo da je mAP za *custom3* malo manja u oba slučaja. Također, ako usporedimo *custom2* (5 000 slika po klasi s veličinom slika 416 x 416 te brojem iteracija od 20 000) s *custom4* (5 000 slika po klasi s veličinom slika 416 x 416 te brojem iteracija od 40 000) vidimo da je mAP za *custom2* manja. Isto se događa kada usporedimo i *custom4* s *custom6* (5 000 slika po klasi s veličinom slika 608 x 608 te brojem iteracija od 40 000). mAP kod *custom4* je manja u oba slučaja.

Sada ćemo usporediti i rezultate modela koji su imali različiti broj slika prilikom treniranja. Ako opet pogledamo sliku 4.13 te usporedimo *custom1* s *custom2* vidimo da je mAP kod *custom1* manja u oba slučaja. Također, mAP je manja u oba slučaja kod *custom3* i *custom5* kada usporedimo *custom3* s *custom4* i *custom5* s *custom6* modelom.

Iz ovoga možemo vidjeti nekoliko stvari. Pomoću grafova koji pokazuju prosječni gubitak prilikom treniranja možemo otprilike znati koji detektor će biti bolji i precizniji ali samo u slučaju kada imamo isti broj slika po klasi. Ako pogledamo grafove gdje se za trening koristilo 1 000 slika (*custom1, custom3, custom5*) i grafove gdje se za trening koristilo 5 000 slika (*custom2, custom4, custom6*) te usporedimo s mAP rezultatima, vidimo da veći prosječni gubitak ne mora nužno značiti da će detektor biti lošiji i manje precizan. Također, pomoću rezultata možemo vidjeti da parametri poput broja slika, veličine slika te broja iteracija utječu na samu preciznost detektora, tj. na krajnji rezultat. Vidimo da što je broj slika, veličina slika i broj iteracija prilikom treniranja veći to je detektor bolji i precizniji.

Slika dolje prikazuje što se dogodi s rezultatima kada koristimo različite slike prilikom treniranja i evaluacije, a što se dogodi s rezultatima kada koristimo iste slike prilikom treniranja i evaluacije.

model	backbone	mAP_{50}	mAP_{75}
YOLO v3 tiny custom1 different images	yolov3-tiny.conv.11	54.5	28.8
YOLO v3 tiny custom1 same images		82.3	54.3

Slika 4.14

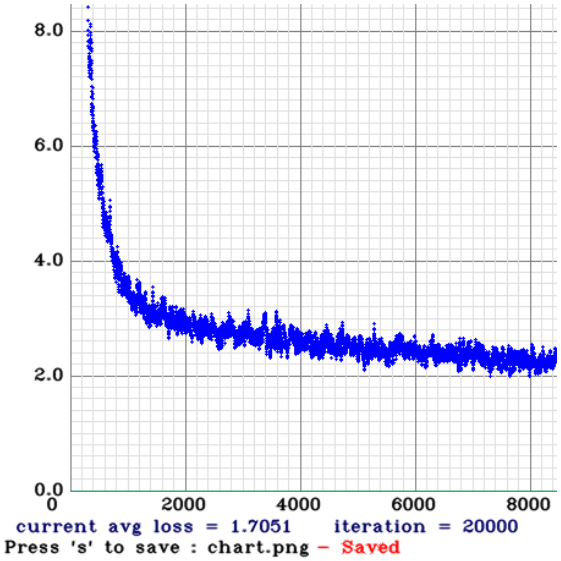
Usporedba mAP za isti model prilikom korištenja različitih i istih slika kod treninga i evaluacije

Kao što vidimo, model kod kojeg su trening i evaluacija provedeni s istim slikama daje puno bolje rezultate od modela kod kojeg su trening i evaluacija provedeni s različitim slikama. U stvarnosti model nikada neće trebati prepoznati iste slike s kojima je trenirao pa se iz tog razloga uzimaju različite slike kako bi se dobila realističnija slika rezultata.

4.4 Treniranje s dvije klase

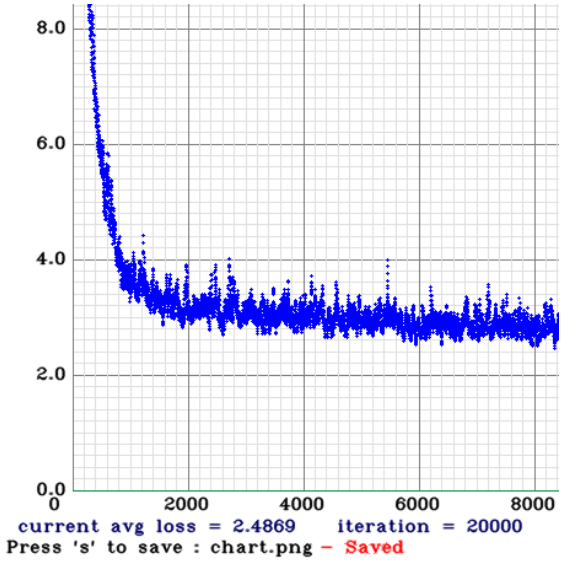
Isto kao i kod treniranja s jednom klasom, slike za treniranje su preuzete sa Open Images V6 dataseta. Za dataset su korištene dvije klase (ljudsko lice i ruke). Važno je napomenuti da prije korištenja dataseta moramo konvertirati anotacije za klase koje koristimo u datasetu (u ovom slučaju anotacije za klase ljudskog lica i ruku). Najprije je napravljen dataset s 1 000 slika po klasi. Taj dataset je korišten da se provede više treninga. Nakon toga, napravljen je dataset s 5 000 slika po klasi koji je također korišten za provođenje više treninga. Prilikom treniranja s dvije klase korišteni su isti parametri koji su korišteni i kod treniranja s jednom klasom. Željelo se istražiti da li treniranje s više klasa ima utjecaj na krajnji rezultat. Također, istražilo se da li pomoću prosječnog gubitka prilikom treninga s dvije klase možemo predvidjeti krajnji rezultat. Nakon što je napravljen dataset, u programu Microsoft Visual Studio editirana je *yolov3-tiny.cfg* datoteka. Kao i kod treniranja s jednom klasom, važno je napomenuti da je kod ovog treninga datoteka *yolov3-tiny.cfg* bila samo oslonac te da se ona nije koristila za samo treniranje. Parametri datoteke *yolov3-tiny.cfg* su promijenjeni u svrhu ovog istraživanja te se prilikom ovog istraživanja koristila datoteka *yolov3-tiny-custom.cfg*. Kod treniranja, kao *backbone* je korišten *yolov3-tiny.conv.11*.

U nastavku su prikazani grafovi (slike 4.15 – 4.20) koji su dobiveni nakon provedenih treniranja neuronske mreže *yolov3-tiny-custom* s dvije klase (ljudsko lice i ruke).



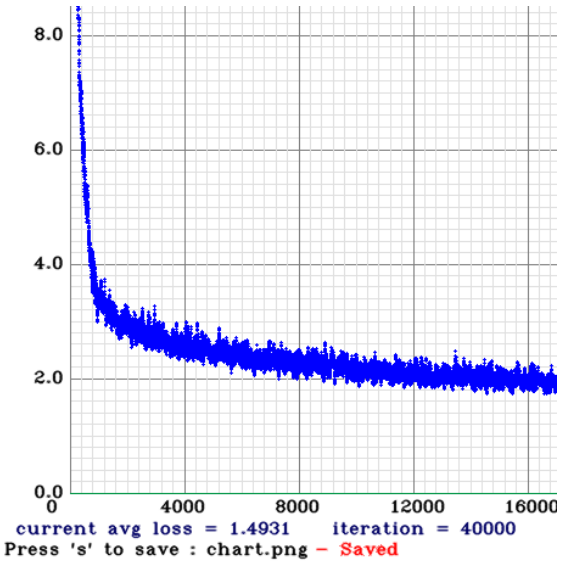
Slika 4.15

Prosječni gubitak za YOLO v3 tiny custom7



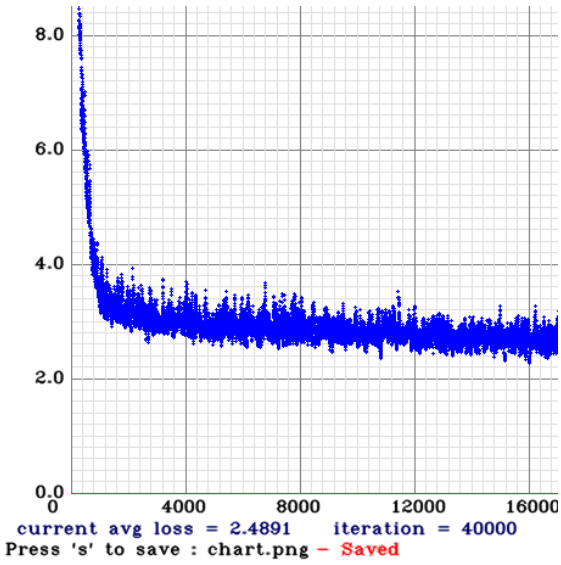
Slika 4.16

Prosječni gubitak za YOLO v3 tiny custom8



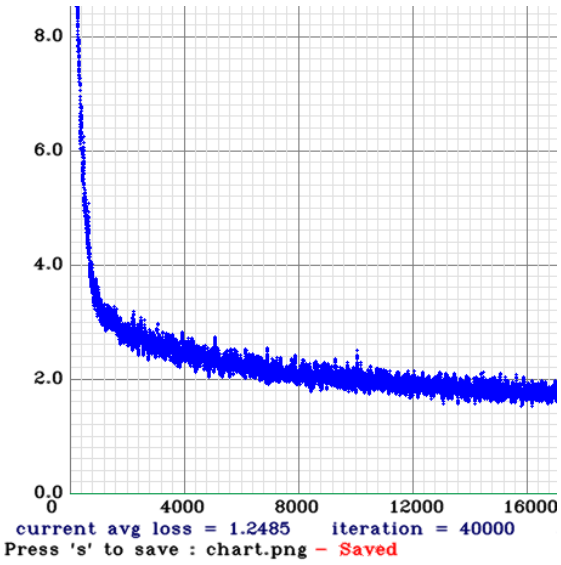
Slika 4.17

Prosječni gubitak za YOLO v3 tiny custom9



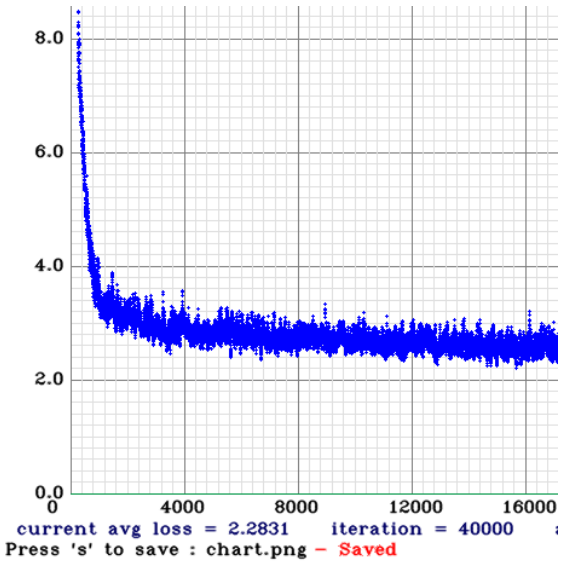
Slika 4.18

Prosječni gubitak za YOLO v3 tiny custom10



Slika 4.19

Prosječni gubitak za YOLO v3 tiny custom11



Slika 4.20

Prosječni gubitak za YOLO v3 tiny custom12

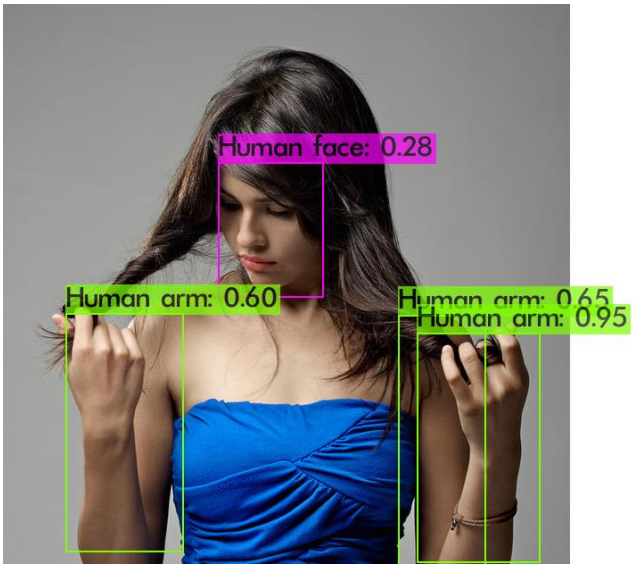
Najprije ćemo usporediti i analizirati grafove s dvije klase. Prilikom *custom7* treninga korišteno je 1 000 slika po klasi s veličinom slika 416 x 416 te brojem iteracija od 20 000. Ako ovo usporedimo s *custom9* treningom kod kojega je za trening korišteno 1 000 slika po klasi s veličinom slika 416 x 416 te brojem iteracija od 40 000, možemo uočiti da je prosječni gubitak kod *custom9* treninga puno manji. Također, ako usporedimo i *custom9* trening s *custom11* treningom kod kojega je za trening korišteno 1 000 slika po klasi s veličinom slika 608 x 608 te brojem iteracija od 40 000, možemo uočiti da se prosječni gubitak kod *custom11* još smanjio. Ako *custom7* trening usporedimo s *custom8* treningom kod kojega je za trening korišteno 5 000 slika po klasi s veličinom slika 416 x 416 te brojem iteracija od 20 000, možemo uočiti da je prosječni gubitak kod *custom8* treninga puno veći. Isto se događa kada usporedimo *custom9* trening s *custom10* treningom (5 000 slika po klasi s veličinom slika 416 x 416 te brojem iteracija od 40 000) i kada usporedimo *custom11* trening s *custom12* treningom (5 000 slika po klasi s veličinom slika 608 x 608 te brojem iteracija od 40 000). Zapravo, iz ovoga možemo vidjeti da grafovi koji su dobiveni pomoću treniranja s dvije klase prate logiku grafova koji su trenirani pomoću jedne klase. Ovo se moglo i očekivati budući da su kod treninga s dvije klase korišteni isti parametri kao i kod treninga s jednom klasom, razlika je samo bila u broju klasa. Budući da grafovi, koji su dobiveni pomoću treniranja s dvije klase, prate logiku grafova koji su trenirani pomoću jedne klase, najvjerojatnije pomoću prosječnog gubitka možemo otprilike znati koji detektor će biti bolji i precizniji ali samo u slučaju kada imamo isti broj slika po klasi. To ćemo saznati kada provedemo validaciju i testiranje te izračunamo mAP.

Sada ćemo napraviti usporedbu i analizu između grafova s jednom klasom i grafova s dvije klase. Podsjetimo, prilikom treniranja s jednom i s dvije klase korišteni su isti parametri. To znači da bi mogli usporediti graf *custom1* s grafom *cutom7,* graf *custom2* s grafom *cutom8,* graf *custom3* s grafom *cutom9,* graf *custom4* s grafom *cutom10,* graf *custom5* s grafom *cutom11,* graf *custom6* s grafom *cutom12*. Kada usporedimo navedene grafove, možemo vidjeti da je prosječni gubitak za grafove s jednom klasom puno manji od prosječnog gubitka grafova s dvije klase.

Da li to znači da treniranje s više klasa ima utjecaj na krajnji rezultat? Da li su rezultati s jednom klasom bolji od rezultata s dvije klase? Ako jesu, što je tome razlog? Da bi smo odgovorili na ova pitanja, nakon treninga moramo provesti validaciju i testiranje te izračunati mAP.

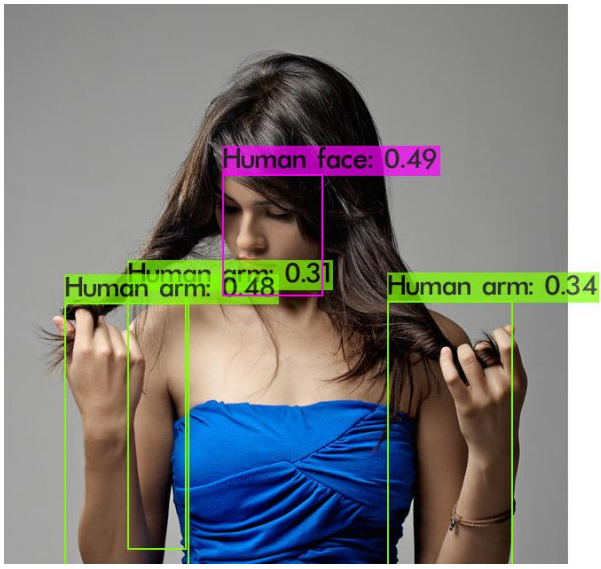
4.5 Validacija i testiranje s dvije klase

Kao i kod validacije i testiranja s jednom klasom, prilikom validacije i testiranja s dvije klase korišteno je 10% slika od ukupnog dataseta. Također, važno je napomenuti da prilikom validacije i testiranja nisu korištene iste slike koje su korištene prilikom treniranja. Klase, u ovom slučaju ljudsko lice i ruke, su ostale iste. U nastavku su prikazani i uspoređeni rezultati koji su dobiveni nakon testiranja. Važno je napomenuti da je ovdje izdvojena samo jedna slika, a ne cijeli testni dataset, kako bi se dao prikaz testiranja te kako bi utvrdili da li detektor detektira klasu za koju je istreniran budući da se na temelju jedne slike ne može utvrditi koji model ima bolje performanse (kad imamo različite slike, za svaku sliku daje drugačiju pouzdanost). Performanse pojedinog modela ćemo utvrditi kasnije prilikom evaluacije kada ćemo izračunati mAP za svaki model. Ista testna slika analizirana je s modelima koji su imali različite parametre.



Slika 4.21

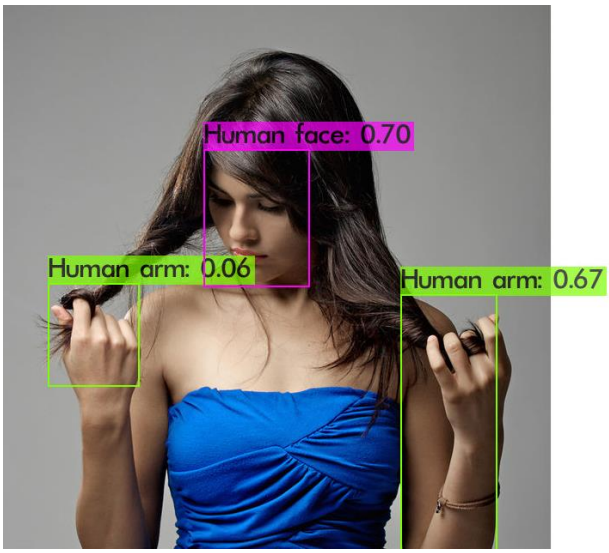
Testiranje za YOLO v3 custom7



Slika 4.22

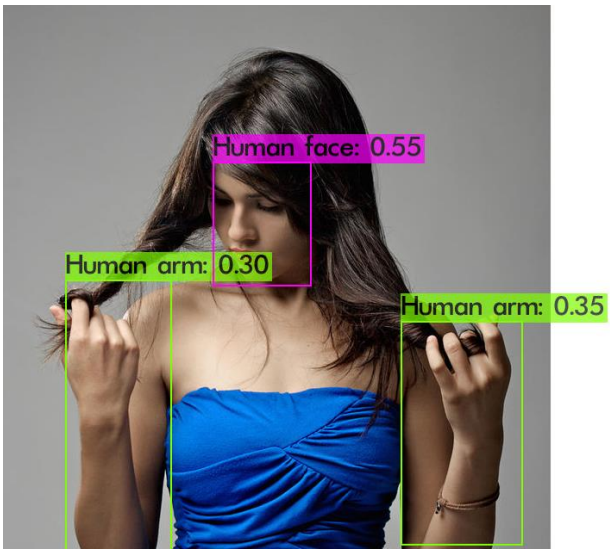
Testiranje za YOLO v3 custom8

Vidimo da *custom7* (slika 4.21) i *custom8* (slika 4.22) prepoznaje obje klase. Također, modeli *custom7* i *custom8* su napravili više predikcija prilikom detekcije ruku. To možemo zaključiti po tome što se na slikama nalaze više graničnih okvira za istu klasu. Također, granični okviri se baš i ne poklapaju s pozicijom ruku.



Slika 4.23

Testiranje za YOLO v3 custom9

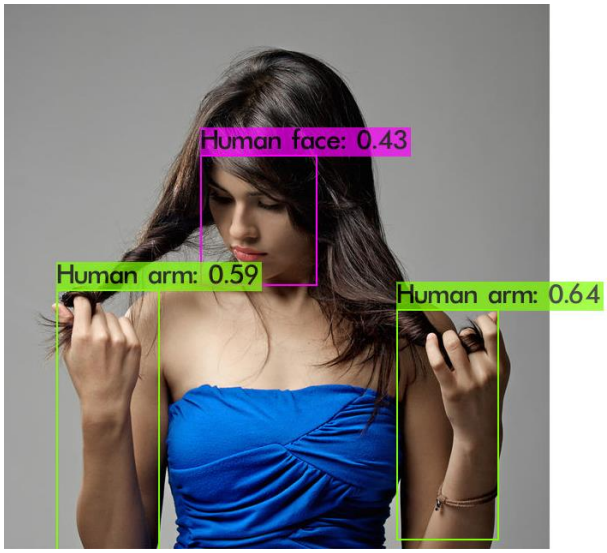


Slika 4.24

Testiranje za YOLO v3 custom10

Slika 4.25

Testiranje za YOLO v3 custom11



Slika 4.26

Testiranje za YOLO v3 custom12

Kada pogledamo *custom9* (slika 4.23) i *custom10* (slika 4.24) vidimo da detektor radi samo jednu predikciju za obje klase. Međutim, granični okvir i pouzdanost za *custom9* na jednoj ruci dosta odstupaju. Ako kasnije pogledamo rezultate za mAP mogli bismo reći da je to čak i normalno jer detektor prilikom testiranja cijelog dataseta nema istu pouzdanost za svaku sliku. Ako pogledamo *custom11* (slika 4.25) i *custom12* (slika 4.26), vidimo da detektor također radi samo jednu predikciju po klasi te granični okviri ne odstupaju puno od pozicija lica i ruku.

4.6 mAP rezultati za dvije klase (evaluacija)

Kao i kod evaluacije s jednom klasom, prilikom izračunavanja mAP vrijednosti je korišteno 10% slika od ukupnog dataseta. Također, važno je napomenuti da prilikom evaluacije nisu korištene iste slike koje su korištene prilikom treniranja. U nastavku je izračunat mAP_{50} i mAP_{75} za dvije klase te su analizirani i uspoređeni rezultati.

model	backbone	mAP_{50}	mAP_{75}
YOLO v3 tiny custom7	yolov3-tiny.conv.11	22.6	9.2
YOLO v3 tiny custom8		29.1	10.7
YOLO v3 tiny custom9		22.8	9.6
YOLO v3 tiny custom10		28.9	11.1
YOLO v3 tiny custom11		22.9	10.4
YOLO v3 tiny custom12		30.1	11.1

Slika 4.27

Usporedba YOLO v3 tiny modela s dvije klase

Ako pogledamo sliku 4.27 te usporedimo model *custom7* (1 000 slika po klasi s veličinom slika 416 x 416 te brojem iteracija od 20 000) i *custotm9* (1 000 slika po klasi s veličinom slika 416 x 416 te brojem iteracija od 40 000) vidimo da je mAP@IoU=50 i mAP@IoU=75 za *custom7* manja. Ako usporedimo *custom9* i *custom11* (1 000 slika po klasi s veličinom slika 608 x 608 te brojem iteracija od 40 000) vidimo da je mAP@IoU=50 i mAP@IoU=75 za *custom9* isto manja. Također, ako usporedimo *custom8* (5 000 slika po klasi s veličinom slika 416 x 416 te brojem iteracija od 20 000) s *custom10* (5 000 slika po klasi s veličinom slika 416 x 416 te brojem iteracija od 40 000) vidimo da je mAP@IoU=50 za *custom8* veća no mAP@IoU=75 je manja. Kada usporedimo i *custom10* s *custom12* (5 000 slika po klasi s veličinom slika 608 x 608 te brojem iteracija od 40 000), mAP@IoU=50 za kod *custom10* je manja no mAP@IoU=75 je ista kao i kod *custom12*.

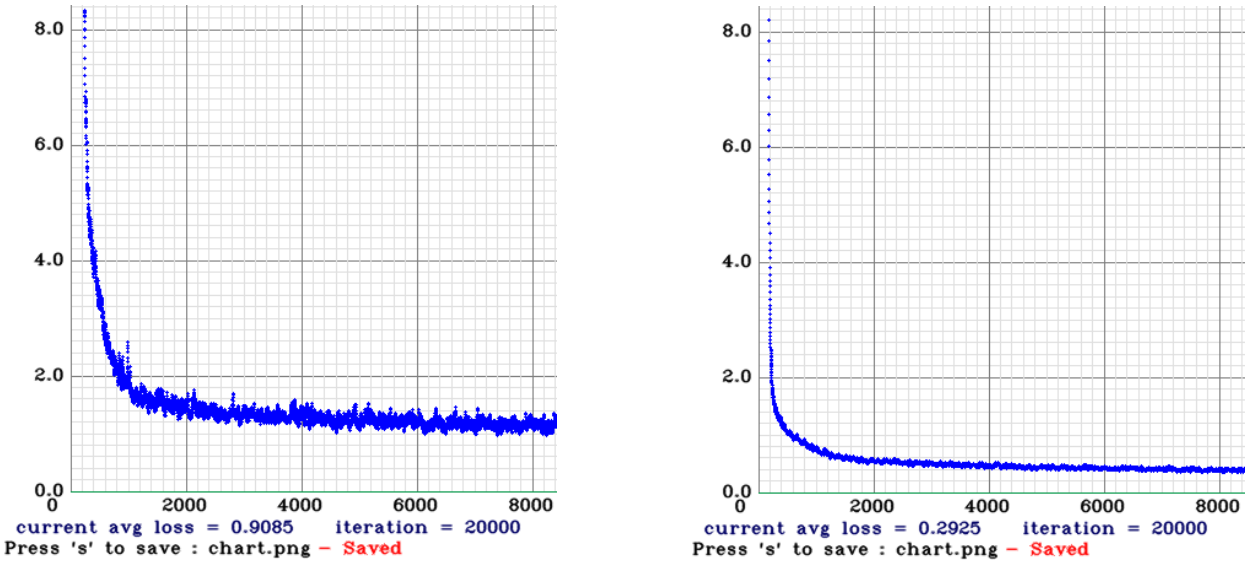
Sada ćemo usporediti i rezultate modela koji su imali različiti broj slika prilikom treniranja. Ako opet pogledamo sliku 4.27 te usporedimo *custom7* s *custom8* vidimo da je mAP kod *custom7* manja u oba slučaja. Također, mAP je manja u oba slučaja kod *custom9* i *custom11* kada usporedimo *custom9* s *custom10* i *custom11* s *custom12* modelom.

Ako usporedimo evaluaciju s jednom klasom i evaluaciju s dvije klase možemo vidjeti da treniranje s dvije klase ima utjecaj na krajnji rezultat. Rezultati kod treniranja s dvije klase su skoro 2 puta manji za mAP@IoU=50 te 3-4 puta manji za mAP@IoU=75. Ovo ukazuje da parametri prilikom treniranja s jednom klasom i prilikom treniranja s dvije klase ne bi trebali biti isti ako bismo željeli dobiti donekle slične rezultate koje smo dobili prilikom treniranja s jednom klasom. Iz gore navedenog jasno je da moramo promijeniti parametre prilikom treniranja s dvije klase. Prvi parametar koji bismo trebali promijeniti je broj slika (trebali bismo povećati broj slika po klasi koji koristimo u našem datasetu). Nakon toga, trebali bismo i promijeniti, tj. povećati broj iteracija.

Parametri poput broja klasa, broja slika, veličine slika i broja iteracija utječu na krajnji rezultat. Grafovi koji pokazuju prosječni gubitak prilikom treniranja mogu služiti za donošenje pretpostavki krajnjih rezultata no pomoću grafova ne možemo biti 100% sigurni što će dati krajnji rezultat.

4.7 Usporedba performansi i rezultata između YOLO v3 tiny i YOLO v4 tiny

Slike za treniranje su preuzete sa Open Images V6 dataseta. Za dataset je korištena klasa ljudskog lica. Prilikom treniranja korišten je isti dataset za YOLO v3 tiny i za YOLO v4 tiny. Također, korišteni su i isti parametri (5 000 slika po klasi s veličinom slika 416 x 416 te brojem iteracija od 20 000). Prilikom treniranja, kao *backbone* za neuronsku mrežu YOLO v3 tiny je korišten *yolov3-tiny.conv.11*, a kao *backbone* za neuronsku mrežu YOLO v4 tiny je korišten *yolov4-tiny.conv.29*. Na slici 4.28 možemo vidjeti prosječni gubitak za YOLO v3 tiny i YOLO v4 tiny.



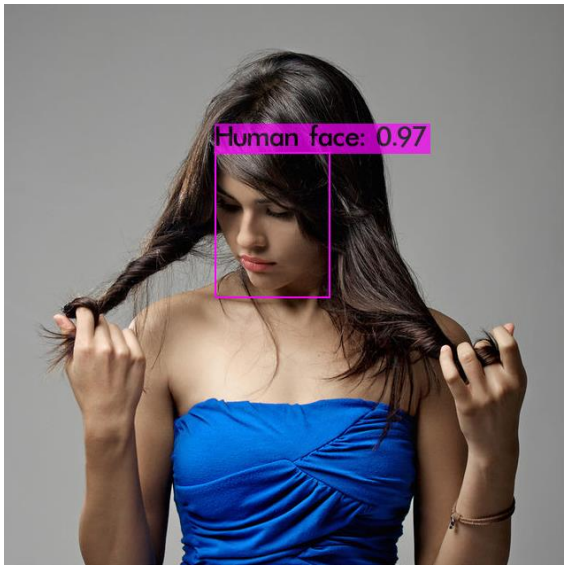
Slika 4.28 Usporedba prosječnog gubitka između YOLO v3 tiny i YOLO v4 tiny

Ako pogledmo grafove vidimo da je prosječni gubitak za YOLO v4 tiny puno manji od prosječnog gubitka za YOLO v3 tiny. Prema tome možemo pretpostaviti da će i krajnji rezultati za YOLO v4 tiny biti bolji. U nastavku je provedena validacija i testiranje te evaluacija.

Prilikom testiranja korišteno je 10% slika od ukupnog dataseta. Također, važno je napomenuti da prilikom testiranja nisu korištene iste slike koje su korištene prilikom treniranja. Klasa, u ovom slučaju ljudsko lice, je ostala ista. U nastavku su prikazani i uspoređeni rezultati koji su dobiveni nakon testiranja (slika 4.29 – 4.30). Kao i kod prijašnjih testiranja izdvojena je samo jedna slika, a ne cijeli testni dataset, kako bi se dao prikaz testiranja te kako bi utvrdili da li detektor detektira klasu za koju je istreniran.

Slika 4.29

Testiranje YOLO v3



Slika 4.30

Testiranje YOLO v4

Prilikom evaluacije, odnosno prilikom izračunavanja mAP vrijednosti je korišteno 10% slika od ukupnog dataseta. Također, važno je napomenuti da prilikom evaluacije nisu korištene iste slike koje su korištene prilikom treniranja. U nastavku je izračunat mAP_{50} i mAP_{75} za YOLO v3 tiny i YOLO v4 tiny te su analizirani i uspoređeni rezultati.

model	backbone	mAP_{50}	mAP_{75}
YOLO v3 tiny	yolov3-tiny.conv.11	63.5	40.4
YOLO v4 tiny	yolov4-tiny.conv.29	68.1	49.6

Slika 4.31

Usporedba mAP između YOLO v3 tiny i YOLO v4 tiny

Iz slike 4.31 možemo vidjeti da YOLO v4 tiny nadmašuje YOLO v3 tiny za mAP@IoU=50 i za mAP@IoU=75. Za mAP_{50} razlika je 4.6% dok je za mAP_{75} razlika 9.2%. To je poprilično puno budući da se treniralo s istim parametrima i budući da je vrijeme treniranja za YOLO v4 tiny trajalo kraće. Iz ovoga možemo vidjeti da i arhitektura neuronske mreže utječe na krajnji rezultat.

Ako pogledamo sliku 4.32 koja prikazuje već izračunate rezultate za YOLO v3 i YOLO v4 te ako usporedimo te rezultate, možemo vidjeti da YOLO v4 također nadmašuje YOLO v3 za mAP@IoU=50 i za mAP@IoU=75.

Method	Backbone	Size	FPS	AP	AP_{50}	AP_{75}
YOLOv4: Optimal Speed and Accuracy of Object Detection						
YOLOv4	CSPDarknet-53	416	38 (M)	41.2%	62.8%	44.3%
YOLOv4	CSPDarknet-53	512	31 (M)	**43.0%**	**64.9%**	**46.5%**
YOLOv4	CSPDarknet-53	608	23 (M)	43.5%	65.7%	47.3%
YOLOv3: An incremental improvement [63]						
YOLOv3	Darknet-53	320	45 (M)	28.2%	51.5%	29.7%
YOLOv3	Darknet-53	416	35 (M)	31.0%	55.3%	32.3%
YOLOv3	Darknet-53	608	20 (M)	33.0%	57.9%	34.4%

Slika 4.32

Usporedba mAP između YOLO v3 i YOLO v4 koristeći MS COCO dataset

https://arxiv.org/pdf/2004.10934.pdf, pristup: 13.8.2020.

Važno je napomenuti da su A. Bochkovskiy, C.-Y. Wang i M. Hong-Yuan u istraživanju za YOLO v4 koristili CSPDarknet53 *backbone* koji je treniran na COCO datasetu [24]. Prilikom YOLO v3 treninga možemo vidjeti da je korišten uobičajen Darknet53 *backbone*. Ovo nam govori da rezultate za YOLO v3 tiny i YOLO v4 tiny ne možemo usporediti s YOLO v3 i YOLO v4 rezultatima. Ovime se samo željelo pokazati da su rezultati za YOLO v4 i YOLO v4 tiny bolji od YOLO v3 i YOLO v3 tiny rezultata.

4.8 Treniranje YOLO v3 tiny s 80 klasa

Slike za treniranje su preuzete sa Open Images V6 dataseta. Za dataset je korišteno 80 klasa te je za svaku klasu korišteno 1 000 slika. Klase koje su korištene prilikom ovog treniranja mogu se naći u GitHub repozitoriju[19]. Parametri s kojima se treniralo, odnosno arhitektura i parametri datoteke *yolov3-tiny.cfg* preuzeti su od Josepha Redmona s GitHub repozitorija darknet[20]. Prilikom treniranja, kao *backbone* za neuronsku mrežu YOLO v3 tiny je korišten *yolov3-tiny.conv.11*.

Nakon treniranja, provedena je validacija i testiranje kako bi utvrdili da li detektor detektira klase za koje je istreniran (slike 4.33 – 4.36). Važno je napomenuti da je u nastavku izdvojeno samo nekoliko slika, a ne cijeli testni dataset. Prilikom validacije i testiranja je također korišteno 10% slika od ukupnog dataseta.

Slika 4.33

Testiranje YOLO v3 tiny klase 16

Slika 4.34

Testiranje YOLO v3 tiny klase 44

[19] https://github.com/Marko-Rogina/YOLO-v3-tiny-80-classes/blob/master/OpenImagesV6Dataset, pristup: 8.9.2020.

[20] https://github.com/pjreddie/darknet/blob/master/cfg/yolov3-tiny.cfg, pristup: 28.8.2020.

Slika 4.35

Testiranje YOLO v3 tiny klase 54

Slika 4.36

Testiranje YOLO v3 tiny klase 61

Prilikom evaluacije, odnosno prilikom izračunavanja mAP vrijednosti je korišteno 10% slika od ukupnog dataseta. Također, važno je napomenuti da prilikom evaluacije nisu korištene iste slike koje su korištene prilikom treniranja. U nastavku je izračunat mAP_{50} za YOLO v3 tiny s 80 klasa.

model	backbone	mAP_{50}	mAP_{75}
YOLO v3 tiny	yolov3-tiny.conv.11	39.8	13.1

Slika 4.37

mAP rezultat za YOLO v3 tiny koji je treniran s 80 klasa

Iz slike 4.37 vidimo da YOLO v3 tiny koji je treniran s 80 klasa daje poprilično dobre rezultate. Najbolje je da prilikom treniranja s više klasa ne koristimo iste parametre koje koristimo i prilikom treniranja s jednom klasom kako bismo dobili donekle dobar rezultat.

5. Zaključak

Detekcija lica pomoću računalnog vida zanimljiva je i aktualna tema iz područja umjetne inteligencije s obzirom na to da ima raznoliku primjenu. Da bi računalo uspjelo prepoznati lice ili neki drugi objekt, potrebno je provesti niz različitih postupaka, poput odabira radnog okruženja, tj. računalnih programa koje ćemo koristiti, odabira arhitekture i dataseta te sastavljanje dataseta za treniranje, validaciju, testiranje i evaluaciju te konačno postavljanje parametara prilikom treniranja.

YOLO v3 tiny daje poprilično dobre rezultate kada je u pitanju detekcija lica prilikom treniranja s jednom, s dvije i s više klasa, čak onda kada se trenira i sa slabijim parametrima. Prilikom treniranja YOLO v3 tiny neuronske mreže moramo uzeti u obzir da će različiti parametri poput broja klasa, broja slika, veličine slika te broja iteracija dati različiti krajnji rezultat, odnosno utjecat će na samu preciznost detektora. Što je broj slika, veličina slika i broj iteracija prilikom treniranja veći, to će detektor biti bolji i precizniji. Prilikom provođenja validacije, testiranja i evaluacije valja pripaziti da se ne koriste iste slike. Modeli kod kojeg su trening, validacija, testiranje i evaluacija provedeni s istim slikama daju puno bolje rezultate nego što oni zapravo jesu. U stvarnosti model nikada neće trebati prepoznati iste slike s kojima je trenirao. Također, prilikom validacije, testiranja i evaluacije se uzima manji broj slika, to je najčešće 10 % od ukupnog dataseta. Pomoću validacije i testiranja ne možemo zaključiti koji model će dati bolje krajnje rezultate. Kako bismo zaključili koji model daje bolje krajnje rezultate, moramo izračunati mAP. Ako nismo zadovoljni s krajnjim rezultatima, uvijek možemo promijeniti parametre ili uzeti bolju arhitekturu te ponovno istrenirati model. Arhitektura neuronske mreže YOLO v4 tiny se pokazala boljom od YOLO v3 tiny arhitekture. Što se arhitekture neuronskih mreža više razvijaju i usavršavaju, to one uz puno slabije parametre (broj slika, broj iteracija, veličina slika) te kraće vrijeme treniranja daju bolji krajnji rezultat.

Iako su se sustavi za detekciju objekata brzo razvijali i napredovali u zadnjih nekoliko godina te su prošli kroz niz poboljšanja, YOLO v3 tiny još uvijek nije blizu ljudskoj sposobnosti prepoznavanja objekata, tj. u ovom slučaju prepoznavanja lica. Ljudi kada vide neku sliku lica, vrlo lako i veoma brzo mogu zaključiti nekoliko stvari odjednom o toj slici, poput toga nalazi li se na slici muškarac ili žena, koju boju očiju i kose imaju, koja je otprilike dob i rasa osobe koju vide na slici. Također, ako se u pozadini slike nalazi neki objekt ili neka znamenitost, ljudi mogu

zaključiti iz koje su države ili čak iz kojega grada lica koja se nalaze na slici. To nam govori da u računalnom vidu još uvijek ima mjesta za napredak kada je u pitanju detekcija objekata.

U Varaždinu, _____ 2020. godine _____

potpis studenta

Sveučilište
Sjever

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, MARKO DOGINA ***(ime i prezime)*** **pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom** DETEKCIJA LICA KORIŠTENJEM NEURONSKE MREŽE YOLO V3 TINY ***(upisati naslov)*** **te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.**

Student/ica:
(upisati ime i prezime)

Dogi
(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, MARKO DOGINA ***(ime i prezime)*** **neopozivo izjavljujem da sam suglasan/na s javnom objavom završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom** DETEKCIJA LICA KORIŠTENJEM NEURONSKE MREŽE YOLO V3 TINY ***(upisati naslov)*** **čiji sam autor/ica.**

Student/ica:
(upisati ime i prezime)

Dogi

6. Literatura

[1] Tryolabs, "An Introductory Guide to Computer Vision", izvor: https://tryolabs.com/resources/introductory-guide-computer-vision/#what-is-computer-vision, zadnji pristup: 30.6.2020.

[2] R. Singh, "Computer Vision — An Introduction", izvor: https://towardsdatascience.com/computer-vision-an-introduction-bbc81743a2f7, zadnji pristup: 30.6.2020.

[3] R. Gandhi, "R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms", izvor: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e, zadnji pristup: 30.6.2020.

[4] M. Venkatachalam, "Introduction to Object Detection", izvor: https://towardsdatascience.com/introduction-to-object-detection-943f21e26063, zadnji pristup: 1.7.2020.

[5] J. Brownlee, "A Gentle Introduction to Object Recognition With Deep Learning", izvor: https://machinelearningmastery.com/object-recognition-with-deep-learning/, zadnji pristup: 1.7.2020.

[6] P. Viola, M. Jones, "Robust Real-time Object Detection", Second International Workshop On Statistical And Computational Theories Of Vision - Modeling Learning Computing And Sampling, Vancouver, Kanada, str. 1-25, srpanj 2001.

[7] P. Viola, M.J. Jones, "Robust Real-Time Face Detection", International Journal of Computer Vision, vol. 57, str. 137–154, svibanj 2004., doi: 10.1023/B:VISI.0000013087.49260.fb

[8] N. Dalal, B. Triggs, "Histograms of oriented gradients for human detection", 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, SAD, lipanj 2005., str. 886-893 vol. 1, doi: 10.1109/CVPR.2005.177.

[9] A. Tzotsos, D. Argialas, "Support Vector Machine Classification for Object-Based Image Analysis", u: Blaschke T., Lang S., Hay G.J. (eds) Object-Based Image Analysis. Lecture Notes in Geoinformation and Cartography. Springer, Berlin, Heidelberg, 2008., doi: 10.1007/978-3-540-77058-9_36

[10] A. Rosebrock, "Histogram of Oriented Gradients and Object Detection", izvor: https://www.pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/, zadnji pristup: 17.6.2020.

[11] MathWorks, "What Is Deep Learning?", izvor: https://www.mathworks.com/discovery/deep-learning.html#whyitmatters, zadnji pristup: 1.7.2020.

[12] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks", izvor: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, zadnji pristup: 17.6.2020.

[13] R. Girshick, "Fast R-CNN", 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Čile, prosinac 2015., str. 1440-1448, doi: 10.1109/ICCV.2015.169.

[14] J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers et al., "Selective Search for Object Recognition", International Journal of Computer Vision, vol. 104, str. 154–171, rujan 2013., doi: 10.1007/s11263-013-0620-5

[15] S. Ananth, "Fast R-CNN for object detection", izvor: https://towardsdatascience.com/fast-r-cnn-for-object-detection-a-technical-summary-a0ff94faa022, zadnji pristup: 19.6.2020.

[16] S. Ren, K. He, R. Girshick, J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, br. 6, str. 1137-1149, lipanj 2017., doi: 10.1109/TPAMI.2016.2577031.

[17] S. Ananth, "Faster R-CNN for object detection", izvor: https://towardsdatascience.com/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46, zadnji pristup: 20.6.2020.

[18] K. He, G. Gkioxari, P. Dollár, R. Girshick, "Mask R-CNN", 2017 IEEE International Conference on Computer Vision (ICCV), Venecija, Italija, 2017., str. 2980-2988, doi: 10.1109/ICCV.2017.322.

[19] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, SAD, lipanj 2016., str. 779-788, doi: 10.1109/CVPR.2016.91.

[20] J. Solawetz, "What is Mean Average Precision (mAP) in Object Detection?", izvor: https://towardsdatascience.com/what-is-mean-average-precision-map-in-object-detection-8f893b48afd3, zadnji pristup: 27.6.2020.

[21] J. Redmon, Ali Farhadi, "YOLOv3: An Incremental Improvement", izvor: https://pjreddie.com/media/files/papers/YOLOv3.pdf, zadnji pristup: 29.6.2020.

[22] A. Kathuria, "What's new in YOLO v3?", izvor: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b, zadnji pristup: 29.6.2020.

[23] Python Lessons: Tensorflow 2, "YOLOv3-Tiny object detection implementation", izvor: https://medium.com/analytics-vidhya/tensorflow-2-yolov3-tiny-object-detection-implementation-c3ea5d4d0510, zadnji pristup: 13.8.2020.

[24] A. Bochkovskiy, C.-Y. Wang and M. Hong-Yuan, "YOLOv4: Optimal Speed and Accuracy of Object Detection", izvor: https://arxiv.org/abs/2004.10934, zadnji pristup: 13.8.2020.

Popis slika

Slika 2.1 Zbroj piksela koji se nalaze unutar bijelog pravokutnika oduzimaju se od zbroja piksela koji se nalaze unutar sivog pravokutnika. Značajka s dva pravokutnika prikazana je u (A) i (B), (C) prikazuje značajku s tri pravokutnika, a (D) prikazuje značajku s četiri pravokutnika. Izvor: http://www.vision.caltech.edu/html-files/EE148-2005-Spring/pprs/viola04ijcv.pdf, pristup: 15.6.2020. 4

Slika 2.2 Vrijednost integralne slike u točki *(x, y)* je zbroj svih piksela s gornje i s lijeve strane. Izvor: http://www.vision.caltech.edu/html-files/EE148-2005-Spring/pprs/viola04ijcv.pdf, pristup: 15.6.2020. 5

Slika 2.3 Zbroj piksela unutar pravokutnika D može se izračunati s četiri referentna polja. Vrijednost integralne slike na lokaciji 1 je zbroj piksela u pravokutniku A. Vrijednost na lokaciji 2 je A + B, na lokaciji 3 je A + C, a na lokaciji 4 je A + B + C + D. Izvor: http://www.vision.caltech.edu/html-files/EE148-2005-Spring/pprs/viola04ijcv.pdf, pristup: 15.6.2020. 5

Slika 2.4 Prva i druga značajka odabrana od AdaBoost. Dvije značajke prikazane su u gornjem redu, a zatim su te dvije značajke postavljene na lice za treniranje u donjem redu. Prva značajka mjeri razliku u intenzitetu između područja oko očiju i području oko gornjih obraza. Značajka dobiva prednost na tome što je područje oko očiju najčešće tamnije od obraza. Druga značajka uspoređuje intenzitet oko područja očiju s intenzitetom gornjeg dijela nosa. Izvor: http://www.vision.caltech.edu/html-files/EE148-2005-Spring/pprs/viola04ijcv.pdf, pristup: 15.6.2020. 6

Slika 2.5 Shematski prikaz kaskadne detekcije. Serija klasifikatora koja se primjenjuje na svaki slučaj. Početni klasifikator eliminira velik broj negativnih slučajeva s vrlo malo obrade. Sljedeći klasifikatori eliminiraju dodatne negativne slučajeve no zahtijevaju dodatno proračunavanje. Nakon nekoliko faza obrade broj slučajeva se radikalno smanjio.
Izvor: http://www.vision.caltech.edu/html-files/EE148-2005-Spring/pprs/viola04ijcv.pdf, pristup: 15.6.2020. 7

Slika 2.6 Pregled procesa za otkrivanje objekata
Izvor: http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf, pristup: 17.6.2020. 8

Slika 2.7 Lijevo: slučaj linearnih klasa koje se mogu odvojiti. Desno: slučaj nelinearnih klasa koje se mogu odvojiti. ξ mjeri pogrešku hiper-ravnine. Izvor: https://www.researchgate.net/publication/225929583_Support_Vector_Machine_Classification_for_Object-Based_Image_Analysis, pristup: 17.6.2020. 9

Slika 2.8 Primjer kliznog prozora gdje se prozor pomiče s lijeva na desno i odozgo prema dolje
Izvor: https://www.pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/, pristup: 17.6.2020. 10

Slika 3.1 Spljoštavanje 3x3 matrice u 9x1 vektor
Izvor: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, pristup: 17.6.2020. 12

Slika 3.2 RGB slika koja je podijeljena na tri boje - crvena, plava i zelena. Zamislite kako bi se zakomplicirale stvari prilikom izračunavanja da je slika dimenzija 8k.
Izvor: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, pristup: 17.6.2020. 12

Slika 3.3 5x5x1 slika s 3x3x1 kernelom kako bi se dobila 3x3x1 konvolucijska značajka
Izvor: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, pristup: 17.6.2020. 13

Slika 3.4 Konvolucijska operacija na MxNx3 matrici slike s kernelom 3x3x3
Izvor: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, pristup: 17.6.2020. 14

Slika 3.5 *Same padding*
Izvor: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, pristup: 17.6.2020. 15

Slika 3.6 Vrste poolinga
Izvor: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, pristup: 17.6.2020. 16

Slika 3.7 Neuronska mreža s puno konvolucijskih slojeva
Izvor: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, pristup: 17.6.2020. 16

Slika 3.8 Problem segmentacije
Izvor: https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013/UijlingsIJCV2013.pdf, pristup: 18.6.2020. 17

Slika 3.9 Dva primjera selektivnog pretraživanja koji pokazuju nužnost različitih veličina
Izvor: https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013/UijlingsIJCV2013.pdf, pristup: 18.6.2020. 19

Slika 3.10 R-CNN proces
Izvor: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e, pristup: 18.6.2020. 20

Slika 3.11 R-CNN
Izvor: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e, pristup: 18.6.2020. 20

Slika 3.12 Brza R-CNN arhitektura
Izvor: https://arxiv.org/pdf/1504.08083.pdf, pristup: 19.6.2020. 21

Slika 3.13 RoI *pooling*
Izvor: https://towardsdatascience.com/fast-r-cnn-for-object-detection-a-technical-summary-a0ff94faa022, pristup: 19.6.2020. 22

Slika 3.14 Rezultati dobiveni koristeći različite klasifikatore
Izvor: https://arxiv.org/pdf/1504.08083.pdf, pristup: 19.6.2020. 23

Slika 3.15 Usporedba algoritama za detekciju objekata
Izvor: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e, pristup: 19.6.2020. 24

Slika 3.16 Brži R-CNN
Izvor: https://arxiv.org/pdf/1506.01497.pdf, pristup: 20.6.2020. 25

Slika 3.17 RPN arhitektura
Izvor: https://towardsdatascience.com/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46, pristup: 20.6.2020. 26

Slika 3.18 Usporedba brzine između algoritama za detekciju objekata
Izvor: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e, pristup: 20.6.2020. 27

Slika 3.19 Maskirani R-CNN
Izvor: https://arxiv.org/pdf/1703.06870.pdf, pristup: 21.6.2020. 29

Slika 3.20 Isprekidana mreža predstavlja mapu značajki, pune linije predstavljaju RoI, a točke predstavljaju četiri uzorkovane lokacije. RoI *align* izračunava vrijednost svake lokacije uzorkovanja pomoću bilinearne interpolacije. Kvantizacija se ne provodi niti u jednoj fazi.
Izvor: https://arxiv.org/pdf/1703.06870.pdf, pristup: 21.6.2020. 30

Slika 3.21 Arhitektura glave. Proširujemo dvije postojeće brže R-CNN glave. Slike prikazuju glave s ResNet C4 i FPN *backbones*, brojevi označavaju prostornu rezoluciju i kanale, strelice označavaju konvolucijske ili potpuno konvolucijske slojeve.
Izvor: https://arxiv.org/pdf/1703.06870.pdf, pristup: 21.6.2020. 31

Slika 3.22 YOLO sustav za detekciju. Najprije se mijenja veličina ulazne slike, zatim se pokreće jedna konvolucijska mreža te se formiraju rezultati detekcije.
Izvor: https://pjreddie.com/media/files/papers/yolo_1.pdf, pristup: 24.6.2020. 32

Slika 3.23 Sustav podijeli sliku na *S x S* mrežu te za svaku ćeliju mreže predviđa granične okvire *B*, pouzdanost za te okvire i vjerojatnost klase *C*
Izvor: https://pjreddie.com/media/files/papers/yolo_1.pdf, pristup: 24.6.2020. 34

Slika 3.24 Arhitektura mreže
Izvor: https://pjreddie.com/media/files/papers/yolo_1.pdf, pristup: 24.6.2020. 34

Slika 3.25 Analiza pogreške: Brzi R-CNN u odnosu na YOLO
Izvor: https://pjreddie.com/media/files/papers/yolo_1.pdf, pristup: 24.6.2020. 37

Slika 3.26 *Precision-recall* krivulja i pouzdanost
Izvor: https://towardsdatascience.com/what-is-mean-average-precision-map-in-object-detection-8f893b48afd3, pristup: 27.6.2020. 38

Slika 3.27 Grafički prikaz IoU
Izvor: https://towardsdatascience.com/what-is-mean-average-precision-map-in-object-detection-8f893b48afd3, pristup: 27.6.2020. 39

Slika 3.28 mAP *precision-recall* krivulje
Izvor: https://towardsdatascience.com/what-is-mean-average-precision-map-in-object-detection-8f893b48afd3, pristup: 27.6.2020. 40

Slika 3.29 Granični okviri s dimenzijama i predviđenim lokacijama
Izvor: https://pjreddie.com/media/files/papers/YOLOv3.pdf, pristup: 29.6.2020. 41

Slika 3.30 Arhitektura YOLO v3 mreže
Izvor: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b, pristup: 29.6.2020. 43

Slika 3.31 Usporedba modela
Izvor: https://pjreddie.com/media/files/papers/YOLOv3.pdf, pristup: 29.6.2020. 44

Slika 3.32 YOLO v3 tiny struktura mreže
Izvor: https://medium.com/analytics-vidhya/tensorflow-2-yolov3-tiny-object-detection-implementation-c3ea5d4d0510, pristup: 13.8.2020. 45

Slika 3.33 Arhitektura YOLO v3 tiny
Izvor: https://medium.com/analytics-vidhya/tensorflow-2-yolov3-tiny-object-detection-implementation-c3ea5d4d0510, pristup: 13.8.2020. 46

Slika 4.1 Prosječni gubitak za YOLO v3 tiny *custom1* 48
Slika 4.2 Prosječni gubitak za YOLO v3 tiny *custom2* 48
Slika 4.3 Prosječni gubitak za YOLO v3 tiny *custom3* 48
Slika 4.4 Prosječni gubitak za YOLO v3 tiny *custom4* 48
Slika 4.5 Prosječni gubitak za YOLO v3 tiny *custom5* 49
Slika 4.6 Prosječni gubitak za YOLO v3 tiny *custom6* 49
Slika 4.7 Testiranje za YOLO v3 *custom1* 50
Slika 4.8 Testiranje za YOLO v3 *custom2* 50
Slika 4.9 Testiranje za YOLO v3 *custom3* 51

Slika 4.10 Testiranje za YOLO v3 *custom4* 51
Slika 4.11 Testiranje za YOLO v3 *custom5* 51
Slika 4.12 Testiranje za YOLO v3 *custom6* 51
Slika 4.13 Usporedba YOLO v3 tiny modela s jednom klasom 52
Slika 4.14 Usporedba mAP za isti model prilikom korištenja različitih i istih slika kod treninga i evaluacije 53
Slika 4.15 Prosječni gubitak za YOLO v3 tiny *custom7* 54
Slika 4.16 Prosječni gubitak za YOLO v3 tiny *custom8* 54
Slika 4.17 Prosječni gubitak za YOLO v3 tiny *custom9* 55
Slika 4.18 Prosječni gubitak za YOLO v3 tiny *custom10* 55
Slika 4.19 Prosječni gubitak za YOLO v3 tiny *custom11* 55
Slika 4.20 Prosječni gubitak za YOLO v3 tiny *custom12* 55
Slika 4.21 Testiranje za YOLO v3 *custom7* 57
Slika 4.22 Testiranje za YOLO v3 *custom8* 57
Slika 4.23 Testiranje za YOLO v3 *custom9* 58
Slika 4.24 Testiranje za YOLO v3 *custom10* 58
Slika 4.25 Testiranje za YOLO v3 *custom11* 58
Slika 4.26 Testiranje za YOLO v3 *custom12* 58
Slika 4.27 Usporedba YOLO v3 tiny modela s dvije klase 59
Slika 4.28 Usporedba prosječnog gubitka između YOLO v3 tiny i YOLO v4 tiny 61
Slika 4.29 Testiranje YOLO v3 62
Slika 4.30 Testiranje YOLO v4 62
Slika 4.31 Usporedba mAP između YOLO v3 tiny i YOLO v4 tiny 62
Slika 4.32 Usporedba mAP između YOLO v3 i YOLO v4 koristeći MS COCO dataset 63
Slika 4.33 Testiranje YOLO v3 klase 16 64
Slika 4.34 Testiranje YOLO v3 klase 44 64
Slika 4.35 Testiranje YOLO v3 klase 54 65
Slika 4.36 Testiranje YOLO v3 klase 61 65
Slika 4.37 mAP rezultat za YOLO v3 tiny koji je treniran s 80 klasa 65