Detekcija objekata na slikama temeljena na konvolucijskoj neuronskoj mreži

Caplić, Andrea

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Andrea Caplić

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

Doc. dr. sc. Tomislav Stipančić, dipl. ing.

Student:

Andrea Caplić

Zagreb, 2022.

Izjavljujem da sam ovaj rad izradila samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem mentoru Doc. dr. sc. Tomislavu Stipančiću na ukazanom povjerenju, pomoći i dostupnosti koju mi je pružio tijekom izrade ovog rada.

Također, zahvaljujem se svojim roditeljima i prijateljima na razumijevanju tijekom studiranja i izrade ovoga rada, a posebno se od srca zahvaljujem svom bratu Andreju koji mi je tijekom cijelog školovanja bio uzor, bezuvjetna podrška i neopisiva motivacija.

Andrea Caplić

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 22 – 6 / 1**	
Ur.broj: **15 - 1703 - 22 -**	

ZAVRŠNI ZADATAK

Student: **Andrea Capalić** JMBAG: **0035213117**

Naslov rada na hrvatskom jeziku: **Detekcija objekata na slikama temeljena na konvolucijskoj neuronskoj mreži**

Naslov rada na engleskom jeziku: **Detection of objects in images based on a convolutional neural network**

Opis zadatka:

Zadnjih godina tehnike dubokog učenja postižu zapažene rezultate kod problema detekcije objekata na slikama. Detekcija objekata je zadatak iz domene računalnog vida koji uključuje određivanje prisutnosti, lokalizacije te identifikacije jednog ili više objekata na slici od interesa. YOLO skupina konvolucijskih neuronskih mreža snažan je skup unaprijed treniranih računalnih modela temeljen na KERAS biblioteci strojnog učenja.

U radu je potrebno:

- odabrati te pripremiti prikladan unaprijed trenirani YOLO model mreže,
- odabrati skup slika na kojima je proveden trening unaprijed treniranog modela mreže,
- napraviti predviđanja o prisutnosti objekata na odabranom skupu slika te interpretirati rezultate,
- na ulaz u mrežu dovesti sliku koja nije bila uključena u testni skup slika te dati kritički osvrt na rezultate.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:
30. 11. 2021.

Zadatak zadao:
Doc. dr. sc. Tomislav Stipančić

Datum predaje rada:
1. rok: 24. 2. 2022.
2. rok (izvanredni): 6. 7. 2022.
3. rok: 22. 9. 2022.

Predviđeni datumi obrane:
1. rok: 28. 2. – 4. 3. 2022.
2. rok (izvanredni): 8. 7. 2022.
3. rok: 26. 9. – 30. 9. 2022.

Predsjednik Povjerenstva:
Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

SAŽETAK

Detekcija objekata je aktualna tehnologija povezana s računalnim vidom kojoj su cilj identifikacija i klasifikacija objekata te lociranje njihovih položaja u digitalnim slikama i videozapisima. Ima primjenu u mnogim poljima svakodnevnog života kao što su prepoznavanje lica na nadzornim videokamerama i samovozeći automobili. Čovjek nakon kratkog pogleda na sliku odmah može prepoznati koji su objekti na slici, kako su pozicionirani i u kakvom su međusobnom odnosu s ostalim objektima. Čovjekov vid je brz i precizan, a to nam omogućuje obavljanje složenih zadataka poput vožnje automobila. Brzi algoritmi za otkrivanje objekata omogućavaju i automobilima autonomnu vožnju koja trenutno postoji, no i dalje ima puno ograničenja zbog kojih ta vožnja nije moguća u svim prilikama. Postoji nekoliko metoda dubokog učenja koje se temelje na konvolucijskim neuronskim mrežama, a za potrebe ovog rada korištena je YOLOv3 metoda. U ovom radu su predstavljeni unaprijed trenirani model koji već posjeduje velik broj klasa koje prepoznaje, a potom dodatno trenirani model kako bi mogao prepoznati neke nove objekte.

Ključne riječi: YOLOv3, strojno učenje, duboko učenje, konvolucijske neuronske mreže, Keras

SUMMARY

Object detection is a current technology related to computer vision that aims to identify and classify objects and locate their positions in digital images and videos. It has applications in many fields of everyday life such as facial recognition on surveillance cameras and self-driving cars. After a quick glance at the picture, a person can immediately recognize which objects are in the picture, how they are positioned and what kind of relationship they have with other objects. Human vision is fast and precise and this allows us to perform complex tasks such as driving a car. Fast algorithms for detecting objects enable cars to drive autonomously, which currently exists, but there are still many limitations that do not make such driving possible in all situations. There are several deep learning methods based on convolutional neural networks but the YOLOv3 method was used for the purpose of this paper. This paper presents a pre-trained model that already has a large number of classes it recognizes and additionally trained model in order to be able to recognize some new objects.

Key words: YOLOv3, machine learning, deep learning, convolutional neural networks, Keras

1. UVOD

Za neke osobe umjetna inteligencija je znanstvena fantastika, za neke je ona tek daleka budućnost, no umjetna inteligencija je već sad sastavni dio naših života.

Računalni vid je grana umjetne inteligencije koja strojevima daje sposobnost vida. On podrazumijeva prepoznavanje i obradu slika ili videozapisa. Jedno od područja računalnog vida su umjetne neuronske mreže koje su trenutno aktualne u mnogim tehnologijama poput prepoznavanja prometnih znakova tijekom autonomne vožnje. Čovjeku to djeluje kao jednostavan proces jer ga mi obavljamo prirodno, no da bi računalo moglo razumjeti i opisati ono što vidi na slici moramo ga naučiti detektirati objekte, prepoznavati lica i uklanjati šumove kako bi ono to moglo ubuduće samo raditi. Detektiranje objekta i njegova klasifikacija podrazumijevaju dva različita algoritma. Dok detektiranjem lociramo objekt unutar cijele slike tako da ga smjestimo unutar njegovog graničnog okvira, klasifikacijom određujemo što je taj objekt zapravo. Ovaj završni rad opisuje kako detektirati objekte na slikama pomoću dubokog učenja, a fokus će biti na YOLO skupini konvolucijskih neuronskih mreža.

2. TEORIJSKA OSNOVA RADA

2.1. Strojno učenje

Strojno učenje je vrsta umjetne inteligencije koja omogućuje softverskim aplikacijama da postanu preciznije u predviđanju ishoda bez da su za to eksplicitno programirane. Algoritmi strojnog učenja koriste podatke od prije kao ulaz za predviđanje novih izlaznih vrijednosti. Stoga je cilj strojnog učenja programirati računala da koriste primjere podataka ili prošlih iskustava za rješavanje određenog problema. Strojno učenje je temelj novih tehnologija kao što su samovozeći automobili, aplikacije za prepoznavanje govora, filtriranje elektroničke pošte i prevođenje. Neke implementacije strojnog učenja koriste podatke i neuronske mreže po uzoru na rad ljudskog mozga. Strojno učenje često se kategorizira prema tome kako algoritam uči da bi postao točniji u svojim predviđanjima, a poznajemo četiri osnovna pristupa: nadgledano učenje (engl. *supervised learning*), nenadgledano učenje (engl. *unsupervised learning*), polunadgledano (engl. *semi-supervised learning*) učenje i pojačano učenje (engl. *reinforcement learning*). Vrsta algoritma koji znanstvenici odluče koristiti ovisi o vrsti podataka koju žele predvidjeti. [1]

Nadgledano učenje

U ovoj vrsti strojnog učenja znanstvenici za podatke opskrbljuju algoritme označenim podacima o obuci i definiraju varijable za koje žele da algoritam procijeni korelacije. Specificirani su i ulaz i izlaz algoritma.

Nenadgledano učenje

Ova vrsta strojnog učenja uključuje algoritme koji treniraju na neoznačenim podacima. Algoritam skenira skupove podataka tražeći bilo kakvu smislenu vezu. Podaci na kojima algoritmi treniraju, kao i predviđanja ili preporuke koje izlaze unaprijed su određeni.

Polu-nadgledano učenje

Ovaj pristup strojnom učenju uključuje mješavinu dva prethodna tipa. Znanstvenici koji se bave podacima mogu dati algoritam uglavnom označene podacima za obuku, ali model može samostalno istraživati podatke i razvijati vlastito razumijevanje skupa podataka.

Pojačano učenje

Znanstvenici koji se bave podacima obično koriste učenje s pojačanjem kako bi naučili stroj da dovrši proces od više koraka za koji postoje jasno definirana pravila. Znanstvenici koji se bave podacima programiraju algoritam za dovršetak zadatka i daju mu pozitivne ili negativne znakove dok odrađuju kako izvršiti zadatak. Ali većinom algoritam sam odlučuje koje će korake poduzeti na tom putu.

2.2. Duboko učenje

Duboko učenje je vrsta strojnog učenja koja oponaša način na koji ljudi stječu određene vrste znanja. Izuzetno je korisna tehnika znanstvenicima koji se bave podacima koji imaju zadatak prikupljanja, analiziranja i tumačenja velikih količina podataka. Duboko učenje čini te procese bržim i lakšim.

Kako bismo jednostavnije razumjeli što je to duboko učenje, za primjer ćemo uzeti dijete koje tek uči svoje prve riječi. Zamislimo da je njegova prva riječ bila auto. Dijete će potom pokazivati na mnoge svakodnevne objekte oko sebe, upirati prstom u njih i ponavljati tu riječ. Ovdje glavnu ulogu igra roditelj koji će djetetu reći „Da, to je auto“ ili „Ne, to nije auto“. Što će dijete nalaziti više objekata, postajati će sve svjesnije da auti mogu biti raznih veličina, oblika i boja, ali da svi imaju neke zajedničke karakteristike. Dijete iz ove analogije je računalo, a roditelj je korisnik koji trenira računalo da bi ubuduće samo moglo prepoznati naučeni objekt.

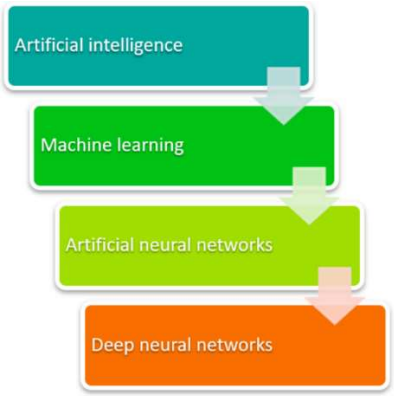
U tradicionalnom strojnom učenju proces učenja se nadzire, a korisnik mora biti krajnje specifičan kada kaže računalu koje karakteristike treba tražiti da odluči sadrži li slika auto ili ne. Ovo je iscrpljujući proces koji se naziva ekstrakcija značajki, a stopa uspjeha računala u potpunosti ovisi o sposobnosti korisnika da točno definira skup značajki za auto. Prednost dubokog učenja je što program sam gradi skup značajki bez nadzora. Učenje bez nadzora ne samo da je brže, već je i točnije.

Računalni program u početku dobiva podatke o treniranju, a to je skup slika za koje je čovjek unaprijed označio svaku sliku sadrži li auto ili ne takozvanim meta oznakama. Program koristi informacije koje prima iz podataka o treniranju za stvaranje skupa značajki za auto i izradu prediktivnog modela. U ovom slučaju, model koji računalo prvo stvori mogao bi predvidjeti da sve što na slici ima četiri gume treba biti označeno kao auto. Sa svakom iteracijom, prediktivni model postaje složeniji i točniji. Za razliku od djeteta kojem će trebati tjedni ili čak mjeseci da shvati koncept auta, računalni program koji koristi algoritme dubokog učenja može proći kroz set za treniranje i sortirati kroz milijune slika, točno identificirajući na kojim slikama se nalaze

auti u roku od nekoliko minuta. Da bi se postigla prihvatljiva razina točnosti, programi dubokog učenja zahtijevaju pristup golemim količinama podataka za treniranje i velike procesorske snage, od kojih nijedno nije bilo lako dostupno programerima sve do doba velikih podataka i računarstva u takozvanom oblaku (engl. *Cloud*). Budući da programiranje dubokog učenja može stvoriti složene statističke modele izravno iz vlastitog iterativnog izlaza, može stvoriti točne prediktivne modele iz velikih količina neoznačenih, nestrukturiranih podataka. Ovo je važno jer Internet stvari (engl. *Internet of things*) postaje sve raširenij jer je većina podataka koje stvaraju ljudi i strojevi nije strukturirana i nije označena. [2]

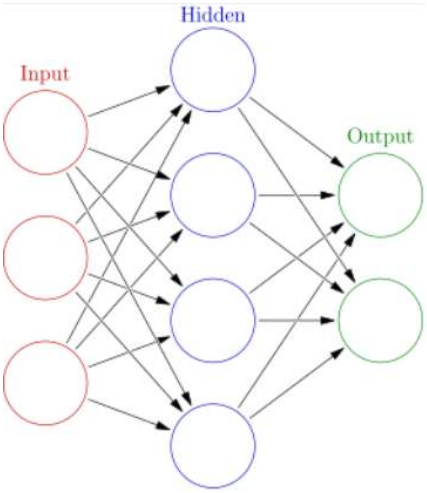
2.3. Duboke neuronske mreže [3]

Neuronske mreže su metoda u umjetnoj inteligenciji koja uči računala da obrađuju podatke na način koji je inspiriran ljudskim mozgom. To je vrsta procesa strojnog učenja, nazvanog duboko učenje, koji koristi međusobno povezane čvorove ili neurone u slojevitoj strukturi koja nalikuje ljudskom mozgu. Duboka neuronska mreža je neuronska mreža s određenom razinom složenosti, odnosno neuronska mreža s više od dva sloja. Duboke neuronske mreže koriste sofisticirano matematičko modeliranje za obradu podataka na složene načine.



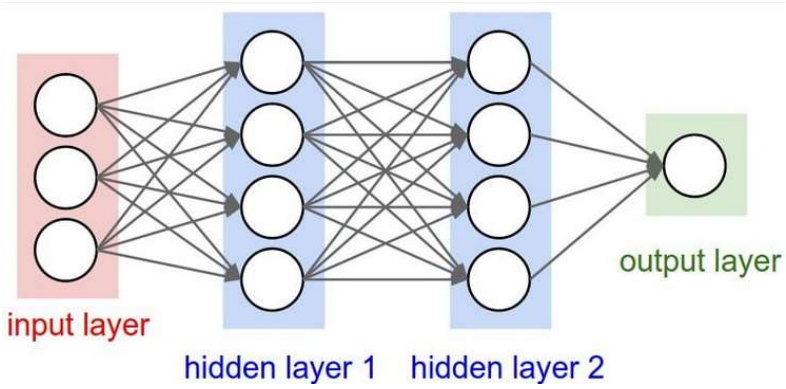
Slika 1. Evolucija dubokih neuronskih mreža

Neuronske mreže koriste skriveni sloj kao mjesto za pohranu i procjenu koliko je jedan od ulaza značajan za izlaz. Skriveni sloj pohranjuje informacije o važnosti unosa, a također uspostavlja veze između važnosti kombinacija unosa.



Slika 2. Shema rada neuronskih mreža

Duboka neuronska mreža ima više skrivenih slojeva. 'Duboko' se odnosi na slojeve modela koji su višeslojni.



Slika 3. Shema rada dubokih neuronskih mreža

2.4. YOLOv3 [4]

Tijekom proteklih nekoliko godina u strojnom učenju vidjeli smo dramatičan napredak u području otkrivanja objekata. Iako postoji nekoliko različitih modela detekcije objekata, ovdje ćemo se usredotočiti na samo jedan model koji se zove "You Only Look Once" ili skraćeno YOLO. YOLO je izgrađen na Darknet klasifikatoru, a izumili su ga Joseph Redmon, Santosh Divvala, Ross Girshick i Ali Farhadi 2015. godine te do sada ima već 3 različite verzije. YOLO kao što mu ime sugerira, „Pogledaj samo jednom“ primjenjuje konvolucijsku neuronsku mrežu s jednim prolazom na cijelu sliku i predviđa granične okvire i njihove vjerojatnosti klasa. Ono

po čemu se YOLO ističe u usporedbi s konkurentima je njegova izuzetna brzina. YOLO može izvesti točan zaključak u manje od pola vremena u odnosu na ostale, postižući sposobnost za praćenje objekata koji se kreću u stvarnom vremenu u videozapisima od 30 sličica u sekundi pomoću grafičkog procesora potrebnog za računalne igre. YOLOv3 ima 53 konvolucijska sloja koji se nazivaju Darknet-53 kao što je prikazano na [Slika 4.].

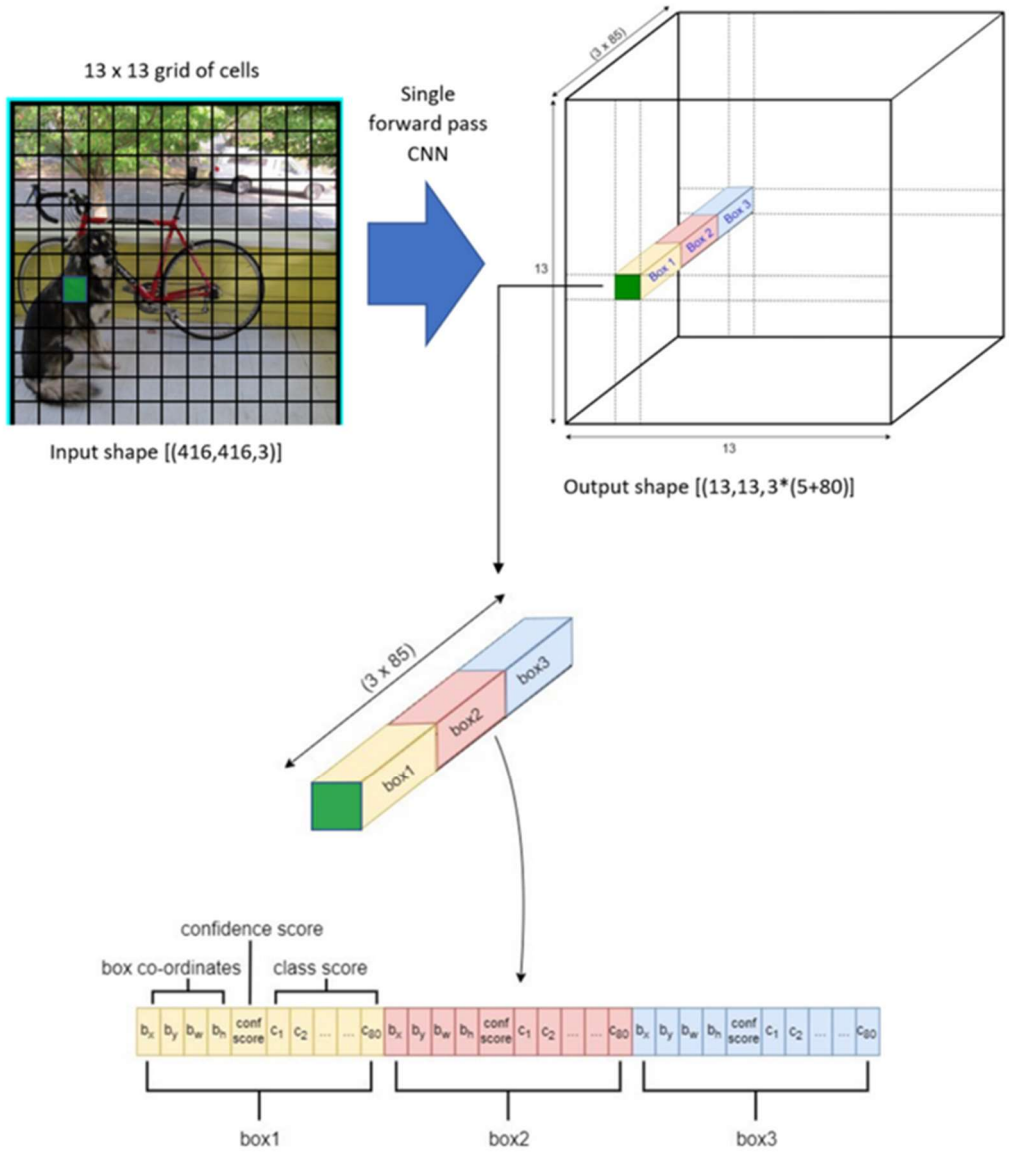
	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Slika 4. Konvolucijski slojevi

YOLOv3 dijeli ulaznu sliku u *S* x *S* mrežu ćelija i predviđa granične okvire kao i vjerojatnosti klase za svaku mrežu. Svaka mrežna ćelija odgovorna je za predviđanje *B* graničnih okvira i *C* klasa vjerojatnosti objekata čija središta padaju unutar mrežne ćelije. Ograničavajući okviri su područja interesa objekata kandidata. *B* je povezan s brojem korištenja „sidara“ (engl. *anchor*). Svaki granični okvir ima (5 + *C*) atribute. Vrijednost 5 odnosi se na 5 atributa graničnog okvira, to su središnje koordinate (b_x, b_y) i oblik (b_h, b_w) graničnog okvira, te jedan rezultat pouzdanosti. *C* je broj klasa. Rezultat pouzdanosti odražava koliko pouzdano okvir sadrži objekt. Ocjena pouzdanosti je u rasponu od 0 – 1 (0-100%).

S obzirom da imamo *S* x *S* mrežu ćelija, nakon pokretanja jedne konvolucijske neuronske mreže YOLOv3 proizvodi 3-D tenzor oblika [*S*, *S*, *B* * (5 + *C*)].

Sljedeća slika [Slika 5.] ilustrira osnovni princip YOLOv3 gdje je ulazna slika podijeljena u mrežu ćelija 13 x 13. Mreža ćelija 13 x 13 koristi se za prvu ljestvicu, dok YOLOv3 zapravo koristi tri različite ljestvice.



Slika 5. Podjela slike u mrežu ćelija

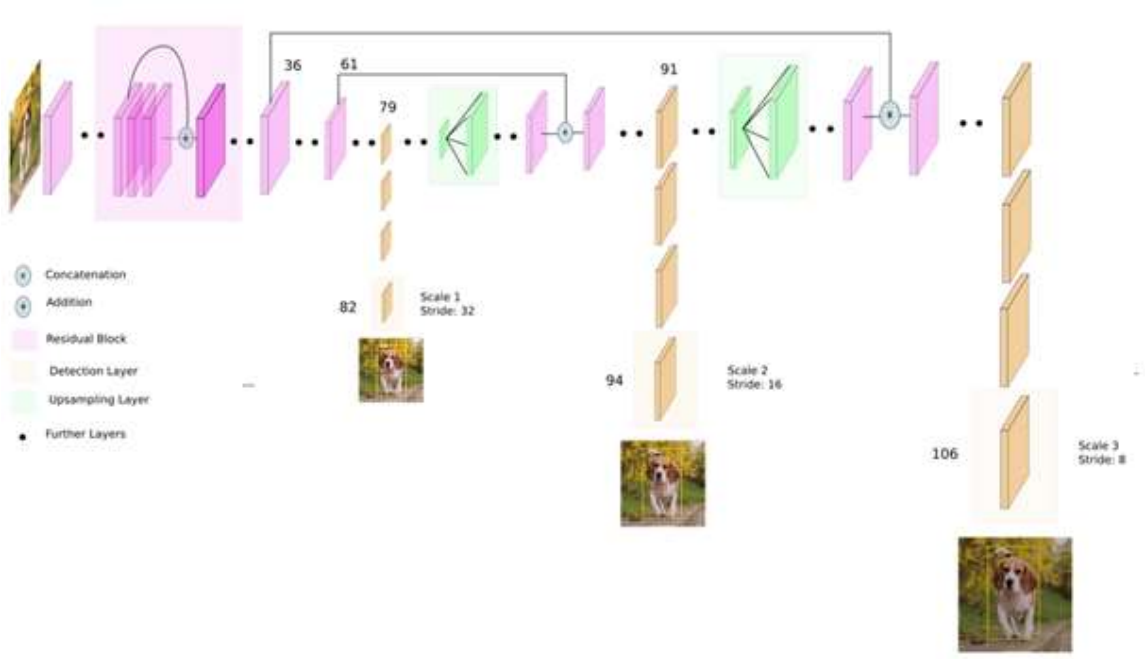
Jedna mrežna ćelija može otkriti samo jedan objekt čija središnja točka pada unutar ćelije, ali ako ćelija rešetke sadrži više od jedne središnje točke objekta to znači da postoji više objekata koji se preklapaju. Kako bi prevladao ovo stanje, YOLOv3 koristi 3 različite sidrene kutije (engl. *Anchor box algorithm)* za svaku ljestvicu detekcije. Sidrene kutije su skup unaprijed definiranih graničnih okvira određene visine i širine koji se koriste za hvatanje mjerila i različitih omjera širine i visine specifičnih klasa objekata koje želimo otkriti. Postoje tri predviđanja u cijeloj ljestvici, tako da je ukupni broj sidrenih kutija devet, oni su: (10×13),

(16×30), (33×23) za prvu ljestvicu, (30×61), (62× 45), (59×119) za drugu ljestvicu i (116×90), (156×198), (373×326) za treću ljestvicu.

YOLOv3 detektira u 3 različite ljestvice kako bi se prilagodio različitim veličinama objekata koristeći korake od 32, 16 i 8. To znači, ako unesemo ulaznu sliku veličine 416 x 416, YOLOv3 će detektirati na ljestvici 13 x 13 , 26 x 26 i 52 x 52. Za prvu ljestvicu, YOLOv3 smanjuje ulaznu sliku na 13 x 13 i čini predviđanje na 82. sloju. Prva ljestvica otkrivanja daje 3D tenzor veličine 13 x 13 x 255.

Nakon toga, YOLOv3 uzima mapu značajki iz sloja 79 i primjenjuje jedan konvolucijski sloj prije nego što ga poveća za faktor 2 kako bi imao veličinu od 26 x 26. Ova mapa značajki s povećanim uzorkovanjem zatim se povezuje s mapom značajki iz sloja 61. Ulančana mapa značajki se zatim podvrgava još nekoliko konvolucijskih slojeva dok se druga ljestvica detekcije ne izvede na sloju 94. Druga ljestvica predviđanja proizvodi 3D tenzor veličine 26 x 26 x 255.

Isti postupak ponovno se izvodi još jednom za predviđanje treće ljestvice. Karti značajki iz sloja 91 dodaje se jedan konvolucijski sloj, a zatim se povezuje s kartom značajki iz sloja 36. Konačni sloj predviđanja se radi na sloju 106 dajući 3D tenzor veličine 52 x 52 x 255. Dakle, YOLOv3 predviđa otkrivanje preko 3 različite ljestvica, pa ako unesemo sliku veličine 416 x 416, nastati će tri različita izlazna tenzora oblika: 13 x 13 x 255, 26 x 26 x 255 i 52 x 52 x 255.



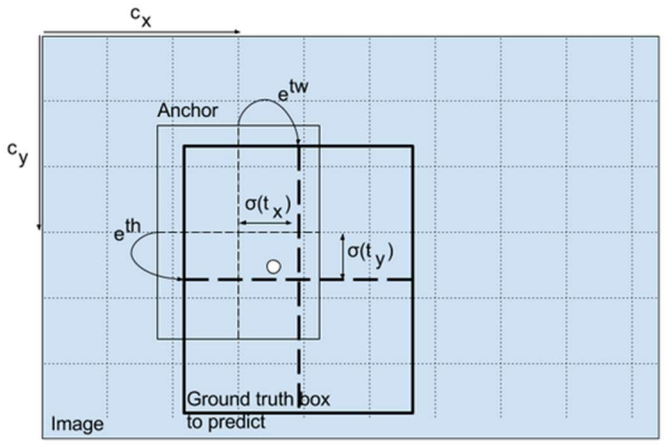
Slika 6. Arhitektura YOLOv3 mreže

Za svaki granični okvir YOLO predviđa 4 koordinate: t_x, t_y, t_w, t_h. Gdje su t_x i t_y su središnje koordinate graničnog okvira u odnosu na ćeliju rešetke čije središte pada unutra, a t_w i t_h su oblik, širina i visina graničnog okvira.

Konačni rezultat predviđanja graničnog okvira treba pročistiti na temelju ove formule:

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$

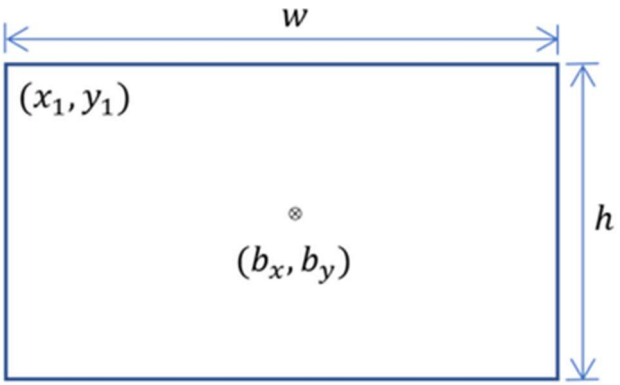
Gdje su p_w i p_h širina i visina sidrene kutije.



Slika 7. Sidrene kutije

YOLO algoritam vraća granične okvire u obliku b_x, b_y, b_w, b_h, pri čemu su b_x i b_y središnje koordinate kutija, a b_w i b_h oblik kutije (širina i visina). Općenito, za ishodište crtanja okvira koristimo gornju lijevu koordinatu (x_1, y_1) i oblik okvira (širina i visina). Da bismo to mogli učiniti, pretvorimo ih pomoću ove relacije:

$$x_1 = b_x - \frac{w}{2}$$
$$y_1 = b_y - \frac{h}{2}$$



Slika 8. Granični okvir

Nakon jednog prolaska konvolucijske neuronske mreže, ono što će se dogoditi jest da mreža YOLO pokušava predložiti više graničnih okvira za isti otkriveni objekt. Problem je kako odlučiti koji je od ovih graničnih okvira pravi. Da bi se prevladao ovaj problem, može se primijeniti metoda koja se naziva *non-maximum suppression* (NMS). U osnovi, ono što NMS radi je čišćenje tih detekcija. Prvi korak NMS-a je potiskivanje svih okvira predviđanja gdje je rezultat pouzdanosti ispod određene vrijednosti praga. Ako je prag pouzdanosti postavljen na 0,5, onda će svaki granični okvir gdje je rezultat pouzdanosti manji ili jednak 0,5 biti odbačen. No ova metoda još uvijek nije dovoljna za odabir odgovarajućih graničnih okvira, jer se ne mogu svi nepotrebni granični okviri eliminirati ovim korakom, pa se tada primjenjuje drugi korak NMS-a. Ostatak viših rezultata pouzdanosti sortira se od najvišeg do najnižeg, zatim se označi granični okvir s najvišim rezultatom kao ispravan granični okvir, a nakon toga se pronađu svi ostali granični okviri koji imaju visok presjek preko unije (IOU) s ovim označenim okvirom. Recimo da smo postavili prag IOU na 0,5, tako da se svaki granični okvir koji ima IOU veći od 0,5 mora ukloniti jer ima visok IOU koji odgovara istom istaknutom objektu. Ova nam metoda omogućuje izlaz samo jednog odgovarajućeg graničnog okvira za otkriveni objekt. Ovaj postupak se ponavlja za preostale granične okvire i uvijek se označi najviši rezultat kao odgovarajući granični okvir i tako sve dok svi granični okviri nisu ispravno odabrani.

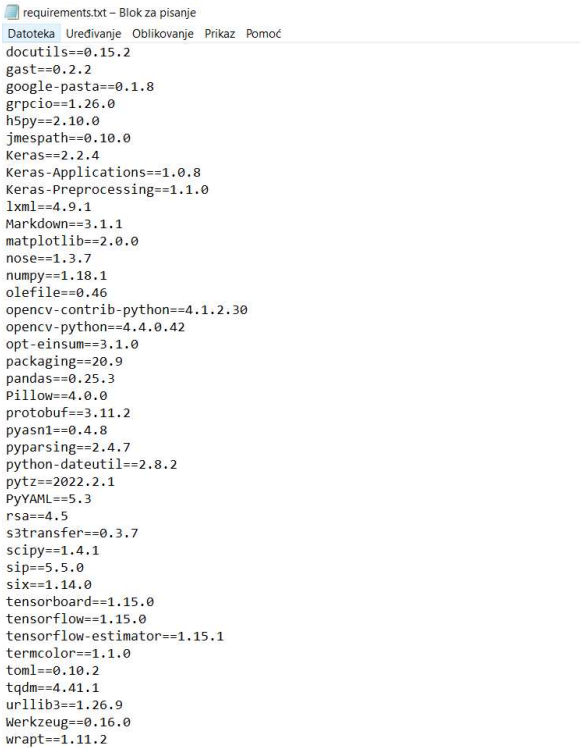
3. IZRADA VLASTITOG RJEŠENJA

3.1. Programska podrška

3.1.1. Python

Python je interpretirani programski jezik opće namjene i visoke razine kojeg je osmislio i stvorio Guido van Rossum 1990. godine, dok je prva javna inačica objavljena je u veljači 1991. godine. Python dopušta programerima korištenje nekoliko stilova programiranja. Objektno orijentirano, strukturno i aspektno orijentirano programiranje su stilovi dopušteni korištenjem Pythona te ova fleksibilnost čini Python programski jezik sve popularnijim. Python se najviše koristi na Linuxu, no postoje i inačice za druge operacijske sustave (Windows, Mac, Raspberry Pi). Python je trenutno najraširenij višenamjenski programski jezik visoke razine. Python programi općenito su manji od drugih programskih jezika poput Jave. Programerima je potrebno relativno manje da napišu kod, a zahtjev za uvlačenjem redaka čini ih čitljivima cijelo vrijeme. Python koriste gotovo svi tehnološki divovi kao što su: Google, Amazon, Facebook, Instagram, Dropbox i Uber. Najveća prednost Pythona je ogromna zbirka standardne biblioteke koja se može koristiti za : strojno učenje, GUI (engl. *Graphical user interface*) aplikacije (npr. Kivy, Tkinter, PyQt), web okviri kao naprimjer Django (koriste ih YouTube, Instagram, Dropbox), obrada slike (OpenCV, Pillow), web scraping (poput Scrapy, BeautifulSoup, Selenium), testni okviri, multimedija, znanstveno računalstvo i obrada teksta. [5]

Za potrebe rješavanja ovog zadatka korištena je verzija Python 3.5, a dodatne potrebne biblioteke su bile dodatno instalirane pomoću Naredbenog retka (engl. *Command prompt*). Popis potrebnih biblioteka i njihovih verzija prikazano je na [Slika 9.]. [5]



Slika 9. Zahtjevi za potrebnim bibliotekama

3.1.2. Keras

Keras je aplikacijsko programsko sučelje (engl. *application programming interface* – API) visoke razine za neuronske mreže u Pythonu koji se može koristi se u kombinaciji s tri različita *backenda*. To uključuje TensorFlow, CNTK i Theano. S ovim okvirom, fokus na razvoj je na jednostavnom sučelju i omogućavanju izvršenja brzog testiranja. Keras podržava i konvolucijske i rekurentne mreže i njihovu kombinaciju. Procesor (CPU) ili grafički čip (GPU) također mogu biti korišteni za izvršenje. Najveće prednosti Kerasa su jednostavnost korištenja, modularnost i jednostavna proširivost novim modulima. Da bi mogli koristiti cijeli asortiman Kerasa, potrebno je instalirati dodatni softver ili preuzeti i instalirati dodatne Python module. [6]

3.1.3. TensorFlow

TensorFlow je platforma za strojno učenje otvorenog koda koja nudi različite razine apstrakcije. Dakle, ovaj okvir preporučuje se i početnicima i iskusnim korisnicima koji žele više kontrole nad mrežom. Službeni Keras API visoke razine može se koristiti za stvaranje neuronske mreže. Za veće zadatke strojnog učenja može se koristiti Distribution Strategy API koji je dizajniran za distribuciju obuke na različite hardverske uređaje bez potrebe za promjenom modela.

Nadalje, TensorFlow je neovisan o jeziku i platformi. Dakle, moguće ga je koristiti za stolno računalo, mobitel, web i Cloud aplikacije. TensorFlow podržava programske jezike kao što su Python, C++ i R. Okvir također ima alat koji se zove TensorBoard, koji nudi učinkovit prikaz mrežnog modeliranja i izvođenje. [7]

3.1.4. NumPy

Skraćenica NumPy je akronim za "Numeric Python". To je programska biblioteka koja proširuje programski jezik s funkcijama za numeričke izračune i matematičke rutine. NumPy je biblioteka za programski jezik Python, koja dodaje podršku za velike, višedimenzionalne nizove i matrice, a s velikom zbirkom matematičkih funkcija omogućuje visoke razine za rad s tim nizovima. Preteču NumPyja - Numeric, izvorno je stvorio Jim Hugunin uz doprinose nekoliko drugih programera. Godine 2005. Travis Oliphant stvorio je NumPy uključivanjem značajki konkurentskog Numarraya u Numeric, uz opsežne izmjene. NumPy je softver otvorenog koda i ima mnogo suradnika. Sve aplikacije za velike podatke i duboko učenje programirane s Pythonom koriste NumPy matematičke funkcije, jer programski jezik Python nije optimiziran za numeričke proračune. [8]

3.1.5. Pip

Pip je akronim od izraza "Preferred Installer Program". Ovaj alat naredbenog retka omogućuje operacijskom sustavu instalaciju, ponovnu instalaciju ili deinstaliranje dijelova Python paketa jednim jednostavnim i jasnim naredbama: “pip install ____”.Naredbu pip također može pratiti broj, na primjer "pip3" što označava verziju Pythona koja se koristi. [9]

3.1.6. Microsoft COCO

Microsoft COCO skup podataka (engl. dataset) je veliki skup podataka o otkrivanju objekata, segmentaciji, otkrivanju ključnih točaka i opisima. Skup podataka sastoji se od 328.000 slika. Prva verzija MS COCO skupa podataka objavljena je 2014., a sadržavala je 164.000 slika podijeljenih u skupove za obuku (83.000), validaciju (41.000) i test (41.000). U 2015. objavljen je dodatni testni set od 81.000 slika, uključujući sve prethodne testne slike i 40.000 novih slika. Na temelju povratnih informacija zajednice, 2017. godine podjela obuke/validacije promijenjena je s 83.000/41.000 na 118.000/5.000. Nova podjela koristi iste slike i bilješke. Testni set iz 2017. podskup je od 41.000 slika testnog skupa iz 2015. godine. Osim toga, izdanje

iz 2017. sadrži novi neoznačeni skup podataka od 123.000 slika. Skup podataka ima napomene za: otkrivanje objekata , natpisi, otkrivanje ključnih točaka, segmentacija stvari na slikama, panoptički (puna segmentacija scene s 80 kategorija stvari kao što su osoba, bicikl i slon i podskupom od 91 kategorije stvari kao trava, nebo, cesta..) i DensePose - više od 39.000 slika i 56.000 instanci osoba označenih komentarima, odnosno svaka osoba označena je ID-om instance i preslikavanjem između piksela slike koji pripadaju tijelu te osobe i 3D modela predloška. Bilješke su javno dostupne samo za slike za obuku i provjeru valjanosti. [10]

3.2. Detekcija objekata na slikama

Kao baza za rješavanje ovog zadatka koristio se već unaprijed trenirani model. Radi se o izazovnom modelu za implementaciju od nule, posebno za početnike jer zahtijeva razvoj mnogih prilagođenih elemenata modela za obuku i predviđanje. Na primjer, čak i izravna upotreba prethodno obučenog modela zahtijeva sofisticirani kod za pročišćavanje i interpretaciju predviđenih graničnih okvira koje daje model.

```
labels = ["person", "bicycle", "car", "motorbike", "aeroplane", "bus", "train", "truck",
    "boat", "traffic light", "fire hydrant", "stop sign", "parking meter", "bench",
    "bird", "cat", "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra", "giraffe",
    "backpack", "umbrella", "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard",
    "sports ball", "kite", "baseball bat", "baseball glove", "skateboard", "surfboard",
    "tennis racket", "bottle", "wine glass", "cup", "fork", "knife", "spoon", "bowl", "banana",
    "apple", "sandwich", "orange", "broccoli", "carrot", "hot dog", "pizza", "donut", "cake",
    "chair", "sofa", "pottedplant", "bed", "diningtable", "toilet", "tvmonitor", "laptop", "mouse",
    "remote", "keyboard", "cell phone", "microwave", "oven", "toaster", "sink", "refrigerator",
    "book", "clock", "vase", "scissors", "teddy bear", "hair drier", "toothbrush"]
```

Slika 10. Objekti unaprijed treniranog modela [11]

Prije svega potrebno je weights format pretvoriti u h5 format da bi Keras biblioteka mogla koristiti već poznate COCO klase.

```
C:\Users\andre\Desktop\FAKS\ZAVRSNI>cd YOLOv3-custom-training

C:\Users\andre\Desktop\FAKS\ZAVRSNI\YOLOv3-custom-training>python convert.py model_data/yolov3.cfg model_data/yolov3.weights model_data/yolo_weights.h5
Using TensorFlow backend.
```

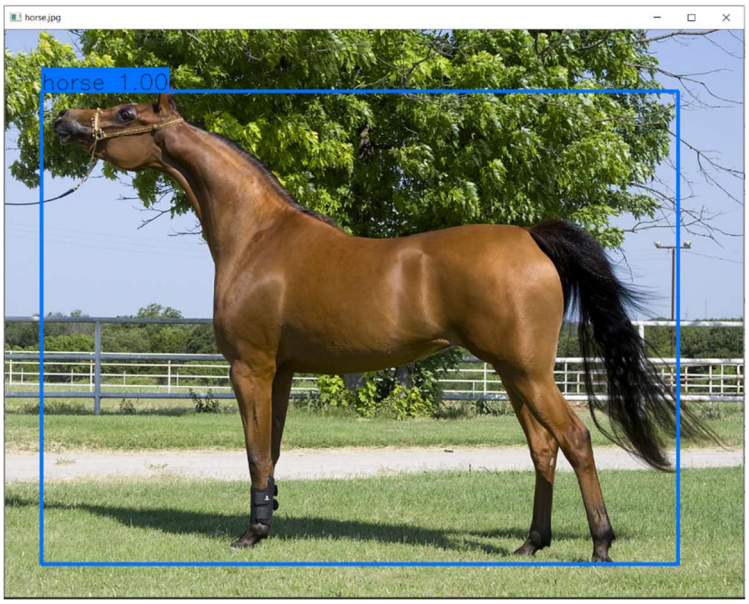
Slika 11. Pretvaranje weights u h5 format

Bitno je da slike učitavamo u JPG formatu i da pritom pazimo da dimenzije slike nisu prevelike. Da provjerimo funkcionira li unaprijed trenirani model učitati ćemo nekoliko različitih slika. Slika se učitava tako da se u kodu napiše njezino ime, pritom se mora paziti da se željena slika i kod nalaze u istoj datoteci. Učitavanje željene slike prikazano je na [Slika 12.].

```
168 if __name__=="__main__":
169     yolo = YOLO()
170     image = 'traffic.jpg'
171     r_image, ObjectsList = yolo.detect_img(image)
172     #print(ObjectsList)
173     cv2.imshow(image, r_image)
174     if cv2.waitKey(25) & 0xFF == ord("q"):
175         cv2.destroyAllWindows()
176     yolo.close_session()
177
```

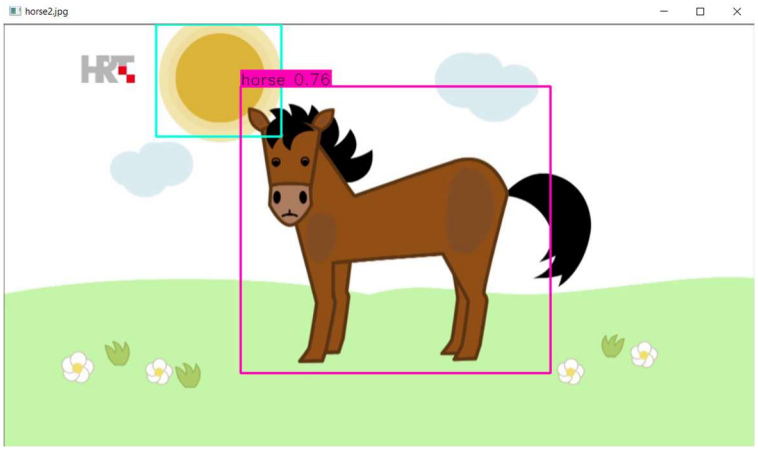
Slika 12. Učitavanje slike

Model prepoznate objekte smješta unutar graničnih okvira, dodjeljuje im naziv i informaciju o tome s kolikom sigurnošću tvrdi da se na slici nalazi upravo taj objekt. [Slika 13.]



Slika 13. Unaprijed trenirani model – konj

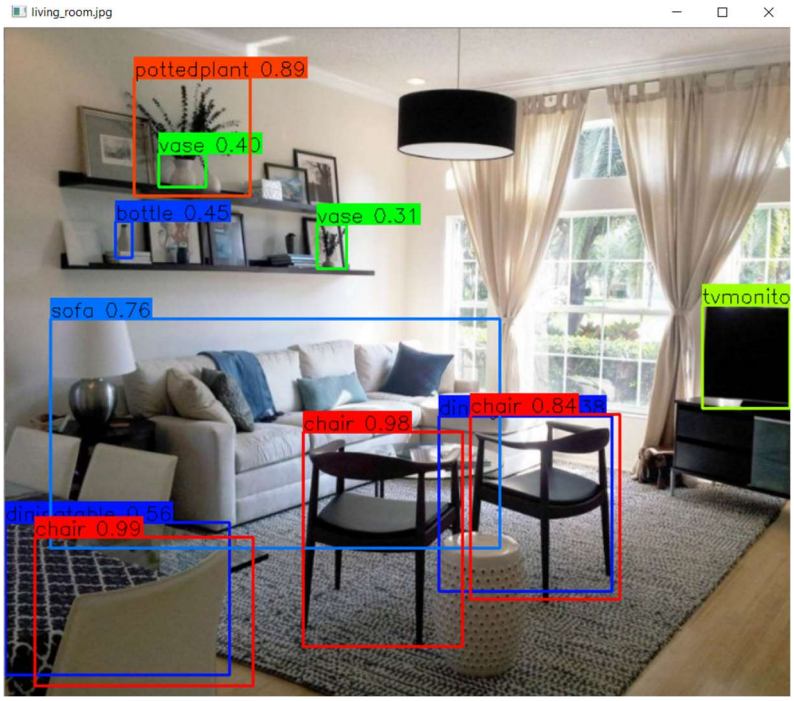
Unaprijed trenirani model ima mogućnost prepoznavati ne samo stvarne, već i nacrtane objekte. [Slika 14.]



Slika 14. Unaprijed trenirani model – nacrtani konj

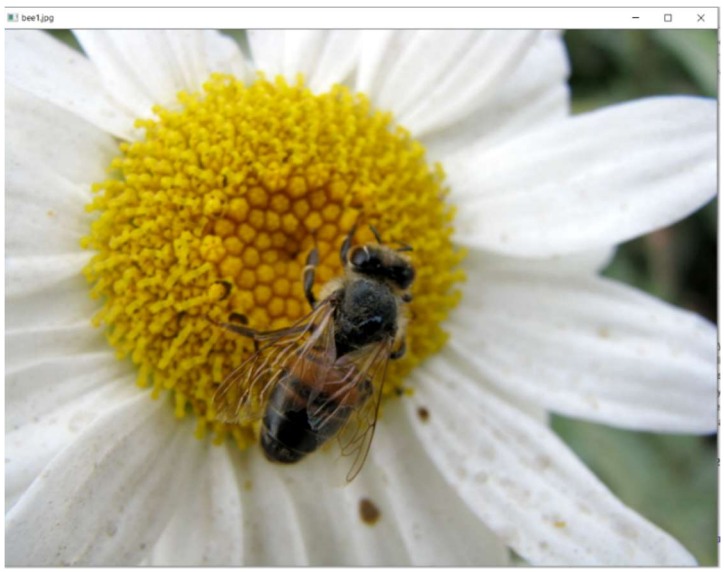
Slika 15. Unaprijed trenirani model - kolona automobila

Model može prepoznavati i više objekata istovremeno, svejedno radi li se o istim ili različitim objektima. [Slika 15.] i [Slika 16.]

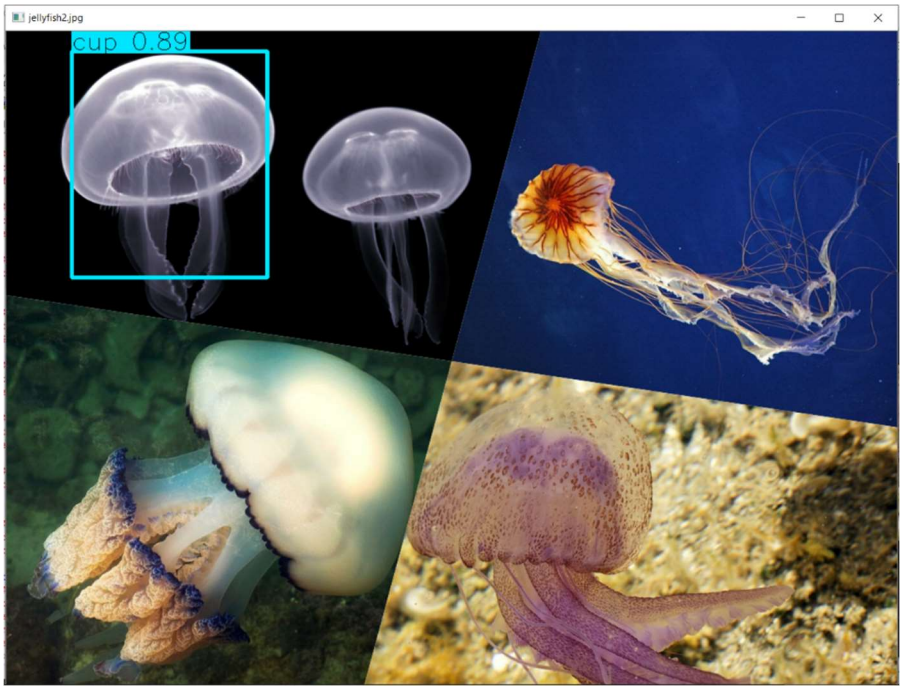


Slika 16. Unaprijed trenirani model - dnevni boravak

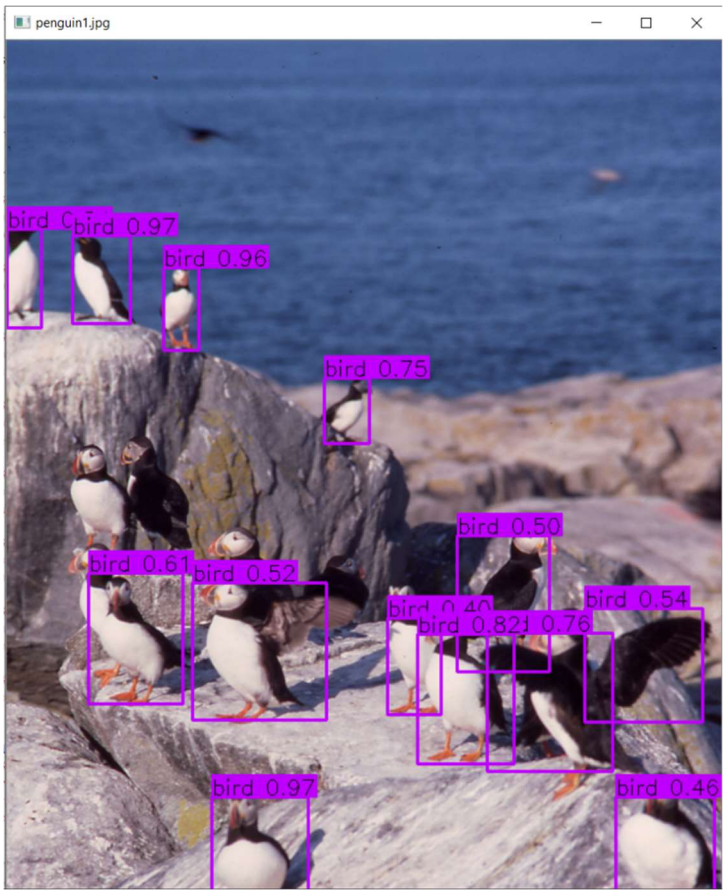
No problem nastaje kada modelu dovedemo neki objekt kojeg još uvijek nije naučio prepoznavati. U tom slučaju ili neće dati nikakvu povratnu informaciju ili će objekt klasificirati kao nešto njemu slično što već prepoznaje. Za primjer ćemo učitati tri nova objekta koja ćemo kasnije ponovo provjeriti i u treniranom modelu, a to su pčela, pingvin i meduza. Rezultati su vidljivi na [Slika 17.], [Slika 18.] i [Slika 19.].



Slika 17. Unaprijed trenirani model – pčela



Slika 18. Unaprijed trenirani model – meduza



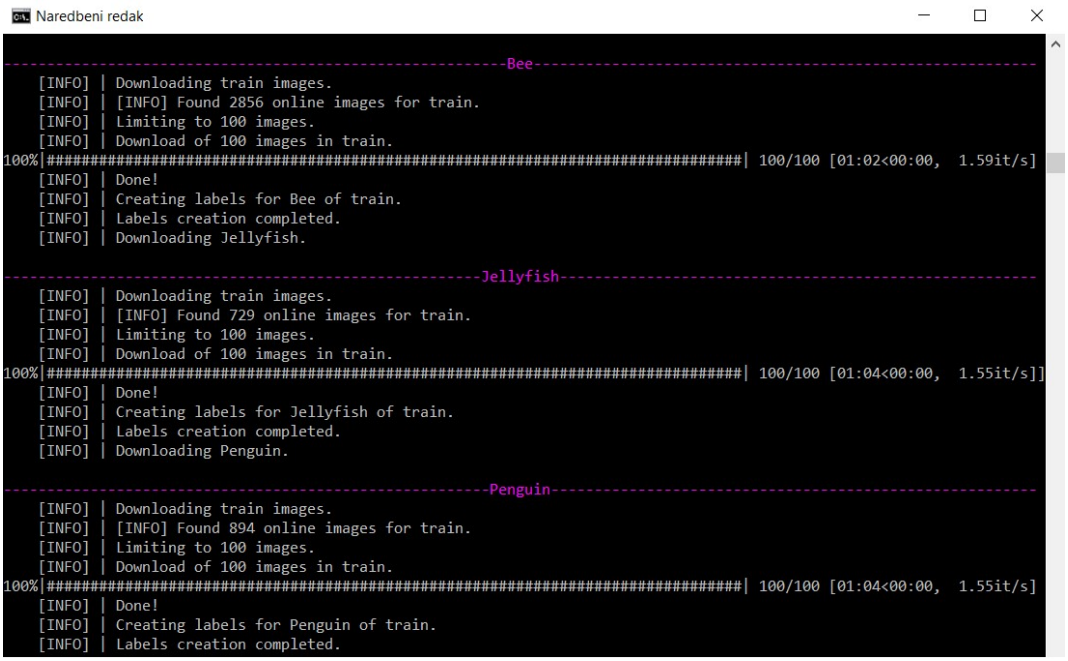
Slika 19. Unaprijed trenirani model - pingvini

S gornjih slika je vidljivo kako pčelu uopće nije detektirao, meduzu je klasificirao kao šalicu sa sigurnošću od 89%, a pingvine je prepoznao kao ptice i to u rasponu od 40-97%.

Kako bismo riješili taj problem potrebno je dodatno trenirati naš unaprijed trenirani model. To znači da ćemo mu morati dovesti prosječno 100 slika svakog novog objekta kako bi on mogao naučiti njegove glavne karakteristike da bi ga ubuduće sam mogao prepoznavati. Pritom postoje dvije mogućnosti: samostalno preuzeti 100 različitih slika istog objekta i ručno na svakoj slici označavati granične okvire unutar kojih je objekt smješten ili preuzeti već gotove datoteke gdje je to netko već učinio umjesto nas. [12]

Za preuzimanje je potrebna jednostavna naredba vidljiva na [Slika 20.] s kojom zatražimo željene nove klase, odredimo koliko slika da preuzme za treniranje, a osim treniranja moguće je preuzeti datoteke za testiranje i validaciju.

Slika 20. Naredba za preuzimanje novih klasa



Slika 21. Preuzimanje novih klasa

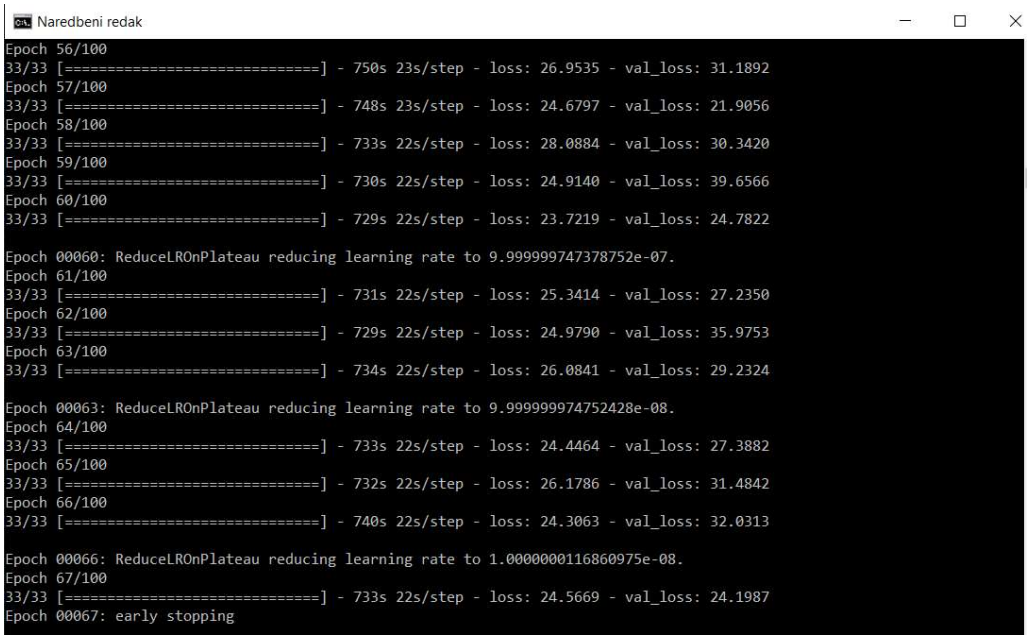
Slike će se potom preuzeti u JPG formatu, no programu slika kao takva ne znači ništa pa ju je potrebno pretvorili u XML (engl. *EXtensible Markup Language*) format. XML je jezik kojeg mogu čitati i ljudi i računala. XML oznake identificiraju podatke i koriste se za pohranjivanje, prijenos, rekonstrukciju i organiziranje podataka.

```
C:\Users\andre\Desktop\FAKS\ZAVRSNI\OIDv4_ToolKit-master>python oid_to_pascal_voc_xml.py
Currently in Subdirectory: test
Currently in Subdirectory: train

Creating PASCAL VOC XML Files for Class: Bee
100%|#################################################################################| 100/100 [00:02<00:00, 43.13it/s]

Creating PASCAL VOC XML Files for Class: Jellyfish
100%|#################################################################################| 100/100 [00:02<00:00, 43.56it/s]

Creating PASCAL VOC XML Files for Class: Penguin
100%|#################################################################################| 100/100 [00:02<00:00, 44.46it/s]
Currently in Subdirectory: validation
```

Slika 22. Pretvaranje JPG u XML format

Nakon toga će se u datotekama gdje su preuzete JPG slike novih klasa pojaviti i pripadajući XML formati koji sadrže informacije o koordinatama objekata unutar slika.

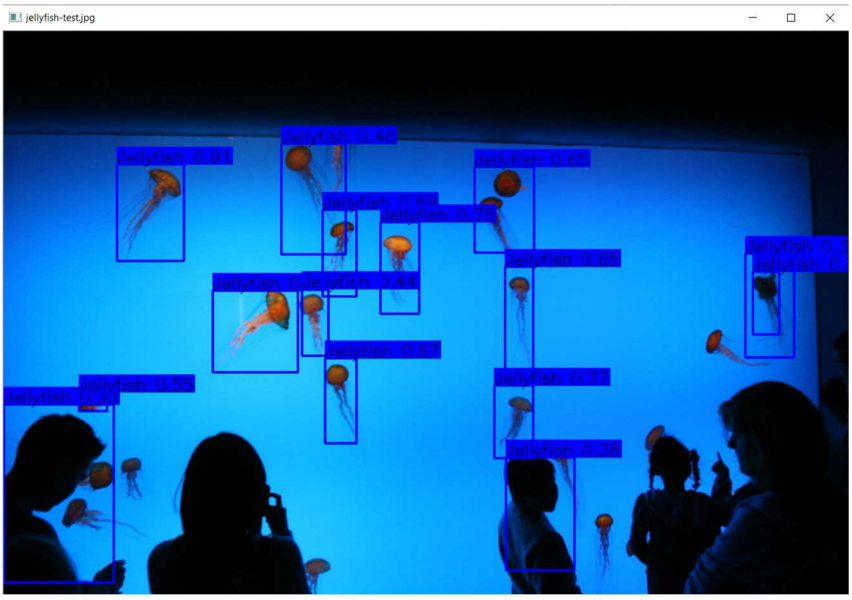
Zatim se pokreće skripta za treniranje . To je iscrpljujući višesatni postupak za računalo tijekom kojeg program prolazi kroz inicijalno određenih 100 epoha (engl. *epoch*). Jedna epoha predstavlja jedan prolazak kroz algoritam za treniranje. Program se sam zaustavio nakon 67. epohe jer u zadnja tri ponavljanja nije imao nikakav napredak.



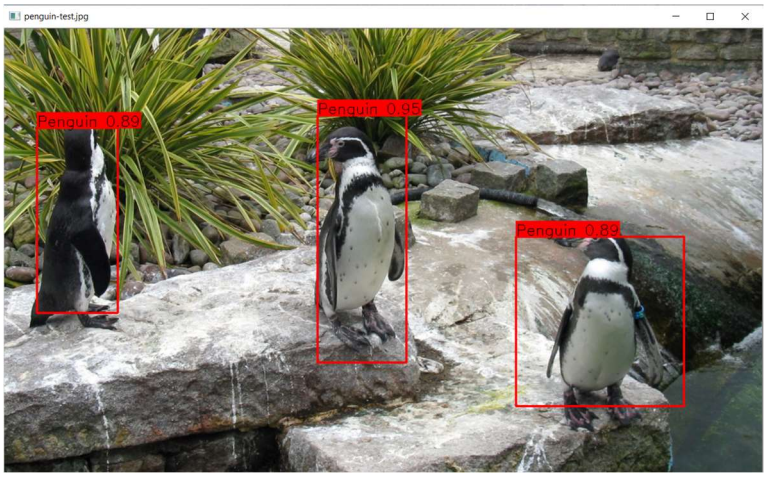
Slika 23. Epohe treniranja

Kada su sve epohe dovršene program je spreman za upotrebu, a u ranije korištenom kodu za detekciju objekata napravljena je mala izmjena kako bi iz iste skripte mogli pokretati i trenirani i unaprijed trenirani model. Zatim su napravljena predviđanja na odabranom skupu slika na temelju kojih je program trenirao. Kao što je vidljivo na [Slika 24.], [Slika 25.] i [Slika 26.], iako je program bio treniran na ovim slikama svejedno objekte nije prepoznao sa 100% sigurnosti.

Slika 24. Predviđanja na odabranom skupu slika – pčela

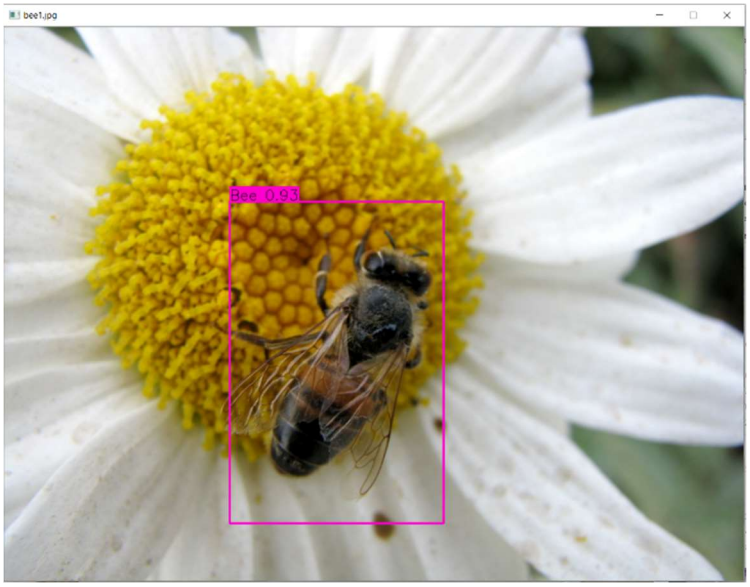


Slika 25. Predviđanja na odabranom skupu slika – meduze

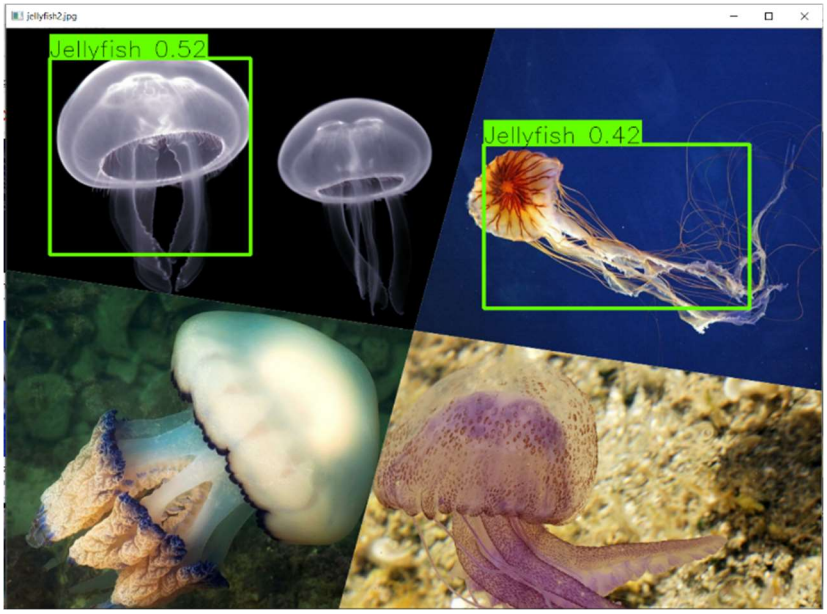


Slika 26. Predviđanja na odabranom skupu slika – pingvini

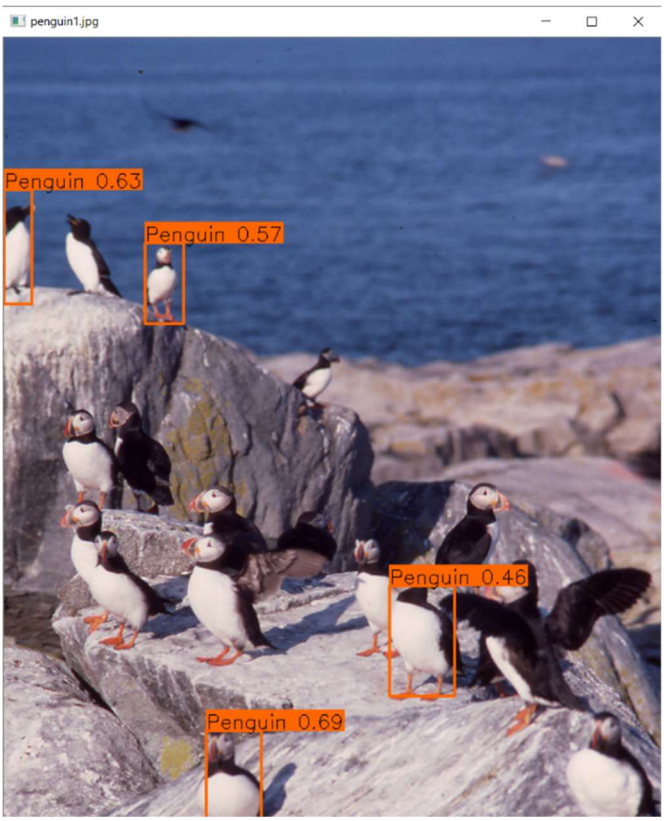
I naposlijetku provjereno je kakve će rezultate program dati kada mu se dovedu objekti koje je naučio prepoznavati, no u pitanju su slike koje se ne nalaze u odabranom skupu slika na temelju kojih je program bio treniran. Ponovo su korištene slike koje su bile dovedene unaprijed treniranom programu. Kao što se vidi na [Slika 27.] pčelu je prepoznao sa 93% sigurnosti, na [Slika 28.] neke meduze uopće nije detektirao, a bio je nesiguran i kod ovih koje je uspio prepoznati, a sličan rezultat se ponovio si s pingvinima na [Slika 29.].



Slika 27. Trenirani model – pčela



Slika 28. Trenirani model – meduze



Slika 29. Trenirani model – pingvini

4. KRITIČKI OSVRT

Kao najveću manu korištenja unaprijed treniranih modela navela bih pojavu takozvanog katastrofalnog zaboravljanja (engl. *catastrophic forgetting*). To je uobičajena pojava kod umjetnih neuronskih mreža kada unaprijed trenirani model nakon naknadnog treniranja zaboravi prethodno naučene informacije. U skladu s time moj model je nakon treniranja bio u mogućnosti prepoznavati nove tri klase, ali zaboravio je prethodnih 80. To se moglo spriječiti da smo u trening modela uključili ponovo tih 80 klasa, no to bi znatno opteretilo računalo. Kada je provedeno testiranje na skupu slika na temelju kojih je model bio treniran, u više slučajeva je rezultat predviđanja bio sa sigurnošću manjom od 50%. Također kada su bile testirane fotografije s više istovrsnih objekata na slici, model bi ih prepoznao tek nekoliko. Na temelju toga zaključujem da YOLO teže detektira i klasificira manje objekte u mnogobrojnim skupinama, što je uzrokovano jakim prostornim ograničenjima nametnutim predviđanjima graničnog okvira, a uz to teže se bori i s prepoznavanjem objekata u novim ili neuobičajenim omjerima, a i s objektima relativno grubih značajki zbog višestruke operacije smanjivanja uzorkovanja. Povećavanjem broja slika potrebnih za treniranje rezultati bi bili precizniji, no to zahtijeva više memorije, bolji grafički procesor i neusporedivo više vremena jer mu je za treniranje samo tri nove klase sa skupom od 100 slika po klasi bilo potrebno 7 sati. Nakon provođenja testiranja smatram da će upotreba ove tehnologije u budućnosti biti od krucijalne važnosti, no trenutno ima mnogo mana i mislim da umjetne neuronske mreže i računalni vid ne mogu komparirati čovjeku.

5. ZAKLJUČAK

Detekcija objekata je aktualno područje danas sveprisutne umjetne inteligencije. Iako čovjek tu radnju obavlja pasivno, računalno se mora trenirati da bi ubuduće samo moglo prepoznavati. To nije jednostavan proces jer računalo mora proći kroz velike baze slika kako bi na temelju njih naučilo kakve karakteristike pojedini objekt posjeduje. Na temelju rezultata je vidljivo da model ni nakon treniranja ne prepoznaje objekte sa 100% sigurnosti, ali postoje određeni parametri s kojima je moguće utjecati na poboljšavanje rezultata, no oni dodatno opterećuju računalo. Povećanjem odabranog skupa slika, veličine slika, rezolucije i broja iteracija potrebnih za treniranje vjerojatno bi se dobili znatno precizniji rezultati. Primjena ovog modela može biti vrlo korisna u mnogim tehnologijama, ali i dalje ne može zamijeniti čovjeka koji ne samo da može detektirati i klasificirati s većom vjerojatnošću nego proučavanjem ostatka slike može dobiti puno širu sliku o objektu poput lokacije objekta, doba dana ili godišnjeg doba za vrijeme kojeg je slikano, marke uređaja i dobi osobe.

LITERATURA

[1] Machine learning, https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML pristupljeno: 04.09.2022.

[2] Deep learning, https://www.techtarget.com/searchenterpriseai/definition/deep-learning-deep-neural-network pristupljeno: 04.09.2022.

[3] Deep neural network, https://www.bmc.com/blogs/deep-neural-network/ pristupljeno: 06.09.2022.

[4] The beginner's guide to implementing YOLOv3 in TensorFlow 2.0, https://machinelearningspace.com/yolov3-tensorflow-2-part-1/ pristupljeno: 07.09.2022.

[5] Python introduction, https://www.w3schools.com/python/python_intro.asp pristupljeno: 14.09.2022.

[6] Keras, https://keras.io/ pristupljeno: 14.09.2022.

[7] NumPy introduction, https://www.w3schools.com/python/numpy/numpy_intro.asp pristupljeno: 14.09.2022.

[8] What is TensorFlow? https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning- library-explained.html pristupljeno: 15.09.2022.

[9] Pip, https://pypi.org/project/pip/ pristupljeno: 15.09.2022.

[10] Microsoft common objects in context, https://paperswithcode.com/dataset/coco pristupljeno: 15.09.2022.

[11] How to perform object detection with YOLOv3 in Keras, https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/?fbclid=IwAR3kyxyl5jMWbtfI4dcG-80B-uRTwqF1ducxT7HMCEH8TFolroF8XNU8Aao pristupljeno: 08.08.2022.

[12] COCO - common objects in context, https://cocodataset.org/#home pristupljeno: 11.08.2022.

[13] How to train an object detection model with Keras, https://machinelearningmastery.com/how-to-train-an-object-detection-model-with-keras/ pristupljeno: 10.08.2022.

[14] YOLO: real-time object detection, https://pjreddie.com/darknet/yolo pristupljeno: 08.08.2022.

[15] Redmon J., Divvala S., Girshick R., Farhadi A.: You Only Look Once: Unified, Real-Time Object Detection, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016., str. 779-788

PRILOZI

I. Python kod – treniranje

```python
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '0'

import numpy as np
import keras.backend as K
from keras.layers import Input, Lambda
from keras.models import Model
from keras.optimizers import Adam
from keras.callbacks import TensorBoard, ModelCheckpoint, ReduceLROnPlateau, EarlyStopping

from yolo3.model import preprocess_true_boxes, yolo_body, tiny_yolo_body, yolo_loss
from yolo3.utils import get_random_data


def _main():
    annotation_path = '4_CLASS_test.txt'
    log_dir = 'logs/000/'
    classes_path = '4_CLASS_test_classes.txt'
    anchors_path = 'model_data/yolo_anchors.txt'
    class_names = get_classes(classes_path)
    num_classes = len(class_names)
    anchors = get_anchors(anchors_path)

    input_shape = (416,416) # multiple of 32, hw

    is_tiny_version = len(anchors)==6 # default setting
    if is_tiny_version:
        model = create_tiny_model(input_shape, anchors, num_classes,
            freeze_body=2, weights_path='model_data/yolo_weights.h5')
    else:
        model = create_model(input_shape, anchors, num_classes, freeze_body=2, weights_path='model_data/yolo_weights.h5')


    logging = TensorBoard(log_dir=log_dir)
    checkpoint = ModelCheckpoint(log_dir + 'ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5',
        monitor='val_loss', save_weights_only=True, save_best_only=True, period=3)
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=3, verbose=1)
    early_stopping = EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=1)

    val_split = 0.1
    with open(annotation_path) as f:
        lines = f.readlines()
    np.random.shuffle(lines)
    num_val = int(len(lines)*val_split)
    num_train = len(lines) - num_val

    # Train with frozen layers first, to get a stable loss.
    # Adjust num epochs to your dataset. This step is enough to obtain a not bad model.
    if True:
        model.compile(optimizer=Adam(lr=1e-3), loss={
            # use custom yolo_loss Lambda layer.
            'yolo_loss': lambda y_true, y_pred: y_pred})

        batch_size = 8
        print('Train on {} samples, val on {} samples, with batch size {}.'.format(num_train, num_val, batch_size))
        model.fit_generator(data_generator_wrapper(lines[:num_train], batch_size, input_shape, anchors, num_classes),
                steps_per_epoch=max(1, num_train//batch_size),
                validation_data=data_generator_wrapper(lines[num_train:], batch_size, input_shape, anchors, num_classes),
                validation_steps=max(1, num_val//batch_size),
                epochs=50,
                initial_epoch=0,
                callbacks=[logging, checkpoint])
        model.save_weights(log_dir + 'trained_weights_stage_1.h5')

    # Unfreeze and continue training, to fine-tune.
    # Train longer if the result is not good.
    if True:
        for i in range(len(model.layers)):
            model.layers[i].trainable = True
        model.compile(optimizer=Adam(lr=1e-4), loss={'yolo_loss': lambda y_true, y_pred: y_pred}) # recompile to apply the change
        print('Unfreeze all of the layers.')

        batch_size = 8 # note that more GPU memory is required after unfreezing the body
        print('Train on {} samples, val on {} samples, with batch size {}.'.format(num_train, num_val, batch_size))
        model.fit_generator(data_generator_wrapper(lines[:num_train], batch_size, input_shape, anchors, num_classes),
            steps_per_epoch=max(1, num_train//batch_size),
            validation_data=data_generator_wrapper(lines[num_train:], batch_size, input_shape, anchors, num_classes),
            validation_steps=max(1, num_val//batch_size),
            epochs=100,
            initial_epoch=50,
            callbacks=[logging, checkpoint, reduce_lr, early_stopping])
        model.save_weights(log_dir + 'trained_weights_final.h5')

    # Further training if needed.
```

```
87  def get_classes(classes_path):
88      '''loads the classes'''
89      with open(classes_path) as f:
90          class_names = f.readlines()
91      class_names = [c.strip() for c in class_names]
92      return class_names
93
94  def get_anchors(anchors_path):
95      '''loads the anchors from a file'''
96      with open(anchors_path) as f:
97          anchors = f.readline()
98      anchors = [float(x) for x in anchors.split(',')]
99      return np.array(anchors).reshape(-1, 2)
100
101
102 def create_model(input_shape, anchors, num_classes, load_pretrained=True, freeze_body=2,
103             weights_path='model_data/yolo_weights.h5'):
104     '''create the training model'''
105     K.clear_session() # get a new session
106     image_input = Input(shape=(None, None, 3))
107     h, w = input_shape
108     num_anchors = len(anchors)
109
110     y_true = [Input(shape=(h//{0:32, 1:16, 2:8}[l], w//{0:32, 1:16, 2:8}[l], \
111         num_anchors//3, num_classes+5)) for l in range(3)]
112
113     model_body = yolo_body(image_input, num_anchors//3, num_classes)
114     print('Create YOLOv3 model with {} anchors and {} classes.'.format(num_anchors, num_classes))
115
116     if load_pretrained:
117         model_body.load_weights(weights_path, by_name=True, skip_mismatch=True)
118         print('Load weights {}.'.format(weights_path))
119         if freeze_body in [1, 2]:
120             # Freeze darknet53 body or freeze all but 3 output layers.
121             num = (185, len(model_body.layers)-3)[freeze_body-1]
122             for i in range(num): model_body.layers[i].trainable = False
123             print('Freeze the first {} layers of total {} layers.'.format(num, len(model_body.layers)))
124
125     model_loss = Lambda(yolo_loss, output_shape=(1,), name='yolo_loss',
126         arguments={'anchors': anchors, 'num_classes': num_classes, 'ignore_thresh': 0.5})(
127         [*model_body.output, *y_true])
128     model = Model([model_body.input, *y_true], model_loss)
129
130     return model
131
132 def create_tiny_model(input_shape, anchors, num_classes, load_pretrained=True, freeze_body=2,
133             weights_path='model_data/tiny_yolo_weights.h5'):
134     '''create the training model, for Tiny YOLOv3'''
135     K.clear_session() # get a new session
136     image_input = Input(shape=(None, None, 3))
137     h, w = input_shape
138     num_anchors = len(anchors)
139
140     y_true = [Input(shape=(h//{0:32, 1:16}[l], w//{0:32, 1:16}[l], \
141         num_anchors//2, num_classes+5)) for l in range(2)]
142
143     model_body = tiny_yolo_body(image_input, num_anchors//2, num_classes)
144     print('Create Tiny YOLOv3 model with {} anchors and {} classes.'.format(num_anchors, num_classes))
145
146     if load_pretrained:
147         model_body.load_weights(weights_path, by_name=True, skip_mismatch=True)
148         print('Load weights {}.'.format(weights_path))
149         if freeze_body in [1, 2]:
150             # Freeze the darknet body or freeze all but 2 output layers.
151             num = (20, len(model_body.layers)-2)[freeze_body-1]
152             for i in range(num): model_body.layers[i].trainable = False
153             print('Freeze the first {} layers of total {} layers.'.format(num, len(model_body.layers)))
154
155     model_loss = Lambda(yolo_loss, output_shape=(1,), name='yolo_loss',
156         arguments={'anchors': anchors, 'num_classes': num_classes, 'ignore_thresh': 0.7})(
157         [*model_body.output, *y_true])
158     model = Model([model_body.input, *y_true], model_loss)
159
160     return model
161
162 def data_generator(annotation_lines, batch_size, input_shape, anchors, num_classes):
163     '''data generator for fit_generator'''
164     n = len(annotation_lines)
165     i = 0
166     while True:
167         image_data = []
168         box_data = []
169         for b in range(batch_size):
170             if i==0:
```

```
                np.random.shuffle(annotation_lines)
            image, box = get_random_data(annotation_lines[i], input_shape, random=True)
            image_data.append(image)
            box_data.append(box)
            i = (i+1) % n
        image_data = np.array(image_data)
        box_data = np.array(box_data)
        y_true = preprocess_true_boxes(box_data, input_shape, anchors, num_classes)
        yield [image_data, *y_true], np.zeros(batch_size)

def data_generator_wrapper(annotation_lines, batch_size, input_shape, anchors, num_classes):
    n = len(annotation_lines)
    if n==0 or batch_size<=0: return None
    return data_generator(annotation_lines, batch_size, input_shape, anchors, num_classes)

if __name__ == '__main__':
    _main()
```

II. Python kod – detekcija objekata

```python
import colorsys
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '0'
import cv2


import numpy as np
from keras import backend as K
from keras.models import load_model
from keras.layers import Input

from yolo3.model import yolo_eval, yolo_body, tiny_yolo_body
from yolo3.utils import image_preporcess

class YOLO(object):
    _defaults = {
        "anchors_path": 'model_data/yolo_anchors.txt',

#       Ako koristimo model s Interneta:
        "model_path": 'model_data/yolo_weights.h5',
        "classes_path": 'model_data/coco_classes.txt',

#       Ako koristimo naš model koji smo trenirali:
        "model_path": 'logs/000/trained_weights_final.h5',
        "classes_path": '4_CLASS_test_classes.txt',

        "score" : 0.3,
        "iou" : 0.45,
        "model_image_size" : (416, 416),
        "text_size" : 3,
    }

    @classmethod
    def get_defaults(cls, n):
        if n in cls._defaults:
            return cls._defaults[n]
        else:
            return "Unrecognized attribute name '" + n + "'"

    def __init__(self, **kwargs):
        self.__dict__.update(self._defaults) # set up default values
        self.__dict__.update(kwargs) # and update with user overrides
        self.class_names = self._get_class()
        self.anchors = self._get_anchors()
        self.sess = K.get_session()
        self.boxes, self.scores, self.classes = self.generate()

    def _get_class(self):
        classes_path = os.path.expanduser(self.classes_path)
        with open(classes_path) as f:
            class_names = f.readlines()
        class_names = [c.strip() for c in class_names]
        return class_names

    def _get_anchors(self):
        anchors_path = os.path.expanduser(self.anchors_path)
        with open(anchors_path) as f:
            anchors = f.readline()
        anchors = [float(x) for x in anchors.split(',')]
        return np.array(anchors).reshape(-1, 2)

    def generate(self):
        model_path = os.path.expanduser(self.model_path)
        assert model_path.endswith('.h5'), 'Keras model or weights must be a .h5 file.'

        # Load model, or construct model and load weights.
        num_anchors = len(self.anchors)
        num_classes = len(self.class_names)
        is_tiny_version = num_anchors==6 # default setting
        try:
            self.yolo_model = load_model(model_path, compile=False)
        except:
            self.yolo_model = tiny_yolo_body(Input(shape=(None,None,3)), num_anchors//2, num_classes) \
                if is_tiny_version else yolo_body(Input(shape=(None,None,3)), num_anchors//3, num_classes)
            self.yolo_model.load_weights(self.model_path) # make sure model, anchors and classes match
        else:
            assert self.yolo_model.layers[-1].output_shape[-1] == \
                num_anchors/len(self.yolo_model.output) * (num_classes + 5), \
                'Mismatch between model and given anchor and class sizes'

        print('{} model, anchors, and classes loaded.'.format(model_path))

        # Generate colors for drawing bounding boxes.
        hsv_tuples = [(x / len(self.class_names), 1., 1.)
                      for x in range(len(self.class_names))]
        self.colors = list(map(lambda x: colorsys.hsv_to_rgb(*x), hsv_tuples))
        self.colors = list(
            map(lambda x: (int(x[0] * 255), int(x[1] * 255), int(x[2] * 255)),
```

```
                self.colors))

        np.random.shuffle(self.colors)  # Shuffle colors to decorrelate adjacent classes.

        # Generate output tensor targets for filtered bounding boxes.
        self.input_image_shape = K.placeholder(shape=(2, ))
        boxes, scores, classes = yolo_eval(self.yolo_model.output, self.anchors,
                len(self.class_names), self.input_image_shape,
                score_threshold=self.score, iou_threshold=self.iou)
        return boxes, scores, classes

    def detect_image(self, image):
        if self.model_image_size != (None, None):
            assert self.model_image_size[0]%32 == 0, 'Multiples of 32 required'
            assert self.model_image_size[1]%32 == 0, 'Multiples of 32 required'
            boxed_image = image_preporcess(np.copy(image), tuple(reversed(self.model_image_size)))
            image_data = boxed_image

        out_boxes, out_scores, out_classes = self.sess.run(
            [self.boxes, self.scores, self.classes],
            feed_dict={
                self.yolo_model.input: image_data,
                self.input_image_shape: [image.shape[0], image.shape[1]],#[image.size[1], image.size[0]],
                K.learning_phase(): 0
            })

        #print('Found {} boxes for {}'.format(len(out_boxes), 'img'))

        thickness = (image.shape[0] + image.shape[1]) // 600
        fontScale=1
        ObjectsList = []

        for i, c in reversed(list(enumerate(out_classes))):
            predicted_class = self.class_names[c]
            box = out_boxes[i]
            score = out_scores[i]

            label = '{} {:.2f}'.format(predicted_class, score)
            #label = '{}'.format(predicted_class)
            scores = '{:.2f}'.format(score)

            top, left, bottom, right = box
            top = max(0, np.floor(top + 0.5).astype('int32'))
            left = max(0, np.floor(left + 0.5).astype('int32'))
            bottom = min(image.shape[0], np.floor(bottom + 0.5).astype('int32'))
            right = min(image.shape[1], np.floor(right + 0.5).astype('int32'))

            mid_h = (bottom-top)/2+top
            mid_v = (right-left)/2+left

            # put object rectangle
            cv2.rectangle(image, (left, top), (right, bottom), self.colors[c], thickness)

            # get text size
            (test_width, text_height), baseline = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, thickness/self.text_size, 1)

            # put text rectangle
            cv2.rectangle(image, (left, top), (left + test_width, top - text_height - baseline), self.colors[c], thickness=cv2.FILLED)

            # put text above rectangle
            cv2.putText(image, label, (left, top-2), cv2.FONT_HERSHEY_SIMPLEX, thickness/self.text_size, (0, 0, 0), 1)

            # add everything to list
            ObjectsList.append([top, left, bottom, right, mid_v, mid_h, label, scores])

        return image, ObjectsList

    def close_session(self):
        self.sess.close()

    def detect_img(self, image):
        image = cv2.imread(image, cv2.IMREAD_COLOR)
        original_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        original_image_color = cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB)

        r_image, ObjectsList = self.detect_image(original_image_color)
        return r_image, ObjectsList


if __name__=="__main__":
    yolo = YOLO()
    image = 'traffic.jpg'
    r_image, ObjectsList = yolo.detect_img(image)
    #print(ObjectsList)
    cv2.imshow(image, r_image)
    if cv2.waitKey(25) & 0xFF == ord("q"):
        cv2.destroyAllWindows()
    yolo.close_session()
```