# Programsko rješenje za detekciju zaštitnih maski na licu

**Pirjać, Jure**

SVEUČILIŠTE U DUBROVNIKU

ODJEL ZA ELEKTROTEHNIKU I RAČUNARSTVO
PRIMIJENJENO/POSLOVNO RAČUNARSTVO

# DIPLOMSKI RAD

## PROGRAMSKO RJEŠENJE ZA DETEKCIJU ZAŠTITNIH MASKI NA LICU

Dubrovnik, srpanj, 2021.

SVEUČILIŠTE U DUBROVNIKU

ODJEL ZA ELEKTROTEHNIKU I RAČUNARSTVO
PRIMIJENJENO/POSLOVNO RAČUNARSTVO

# DIPLOMSKI RAD

## PROGRAMSKO RJEŠENJE ZA DETEKCIJU ZAŠTITNIH MASKI NA LICU

Mentor:
**izv.prof.dr.sc. Mario Miličević**

Diplomant:
**Jure Pirjać**

Dubrovnik, srpanj, 2021.

# SAŽETAK

Širenje bolesti koju uzrokuje COVID 19 je promijenilo način na koji ljudi funkcioniraju i dalo nove obveze kojih se ljudi moraju pridržavati kako bi zaštitili sebe i druge. Tu spada držanje distance, kao i obvezno nošenje zaštitne maske za lice u zatvorenim prostorima. U ovom radu je objašnjena izrada i treniranje modela za objektnu detekciju nošenja zaštitnih maski na licu te usporedba s različitom konfiguracijom istog modela i usporedba s drugim modelom. U prošlosti je stvaranje prilagođenog objektnog detektora trebalo velike količine resursa, ljudi i vremena. Sada pomoću alata TensorFlow 2 API-a za objektnu detekciju se može brže i efikasnije stvoriti pouzdane modele. U ovom radu će biti objašnjeno sljedeće: instalacija TensorFlow API-a za objektnu detekciju na Google Colab-u, instalacija svih potrebnih alata i biblioteka potrebnih za točan rad modela, priprema slika (koja će se obavljati lokalno korištenjem LabelImg alata), treniranje modela uz praćenje grafova na TensorBoard-u, testiranje 3 konfiguracije modela (pomoću slika) na Google Colab-u i za kraj usporedba i objašnjenje rezultata. Također, objašnjene su metrike korištene u radu, kao i konvolucijske mreže te ostala terminologija potrebna za razumijevanje tematike.

*Ključne riječi: konvolucijske mreže, objektna detekcija, TensorFlow 2, zaštitna maska*

# ABSTRACT

The spread of the disease caused by COVID 19 has changed the way people function and has given new obligations that people must adhere to in order to protect themselves and others. This includes keeping a distance, as well as the mandatory wearing of a protective face mask indoors. This paper explains the development and training of a model for object detection of wearing face masks and a comparison with a different configuration of the same model and a comparison with another model. In the past, creating a custom object detector required large amounts of resources, people and time. Reliable models can now be created faster and more efficiently with the TensorFlow 2 Object Detection API. This paper will explain the following: installation of the TensorFlow Object Detection API on Google Colab, installation of all necessary tools and libraries needed for accurate model operation, image preparation (to be performed locally using the LabelImg tool), model training with tracking graphs on TensorBoard, testing 3 model configurations (using images) on Google Colab, and finally comparing and explaining the results. The metrics used in the paper are also explained, as well as the convolutional networks and other terminology needed to understand the topic.

*Keywords: convolutional networks, object detection, TensorFlow 2, protective mask*

# SADRŽAJ:

# 1. UVOD

Strojno učenje je jedno od najpopularnijih područja računalne znanosti. To je grana umjetne inteligencije koja se bavi oblikovanjem algoritama koji svoju učinkovitost poboljšavaju na temelju učenja pomoću prijašnjih rezultata. Cilj strojnog učenja je davanje potrebnih podataka i rezultata strojevima s ciljem da otkriju sličnosti kako bi došli do učinkovitog rješenja. Strojno učenje se primjenjuje u mnogim znanstvenim granama kao što su: medicina, molekularna biologija, statistika i brojne druge grane.

Primjeri zadataka strojnog učenja bi bili: klasifikacija, predikcija, detekcija i ostali problemi koji se mogu učinkovitije riješiti strojnim učenjem u odnosu na klasične algoritme.

Klasifikacija se odnosi na svrstavanje objekata u odgovarajuće kategorije. Klasifikacija može uključivati neke od tehnika prepoznavanja objekata poput umjetnih neuronskih mreža (engl. *Artificial Neural Network*, ANN) i metode potpornih vektora (engl. *Support Vector Machine*, SVM).

Predikcijom se žele predvidjeti buduće vrijednosti iz prethodnih vrijednosti, dok se objektnom detekcijom pokušava otkriti objekt na temelju njegovih osobina.

Detekcija se može opisati kao proces određivanja položaja objekata unutar slike (lokalizacija) i identifikacije označenog objekta s obzirom na ponuđene klase (klasifikacija).

## 1.1. Definicija rada

Predmet istraživanja diplomskog rada je postupak rješavanja problema detekcije nošenja zaštitne maske za lice u vrijeme pandemije bolesti koju uzrokuje COVID 19 pomoću objektne detekcije. U vrijeme zaraze potrebno je i obvezno nošenje zaštitne maske kako bi se zaštitilo sebe i druge. Zbog toga je bitno napraviti funkcionalno programsko rješenje koje radi detekciju nošenja zaštitne maske. Za istraživanje je korišteno TensorFlow okruženje napisano u Python programskom jeziku.

## 1.2. Svrha i ciljevi rada

Svrha rada je objasniti princip objektne detekcije više objekata na slici, koja uključuje lokalizaciju i klasifikaciju. Tu spada prikupljanje slika za vlastiti skup podataka, njihovo označavanje u LabelImg alatu i pisanje programskog rješenja za treniranje. Cilj je usporediti više različitih modela te preko rezultata testiranja odabrati model koji daje najbolje rezultate. Također, tijekom rada će biti objašnjen koncept umjetne inteligencije, duboko učenje, konvolucijske neuronske mreže, korišteni modeli i druge teorijske stavke potrebne za razumijevanje tematike.

## 1.3. Metodologija rada

U radu će se opisati TensorFlow okruženje za treniranje objektne detekcije, opis više modela i njihove arhitekture te objašnjenje rezultata dobivenih treniranjem prilagođenih modela.

## 1.4. Struktura rada

Rad je podijeljen na dva dijela. U prvom dijelu je opisana teorija strojnog učenja i dubokih neuronskih mreža gdje je posebna važnost dana opisu konvolucijskih neuronskih mreža. Također, opisani su osnovni pojmovi, korištene metrike kao i modeli korišteni pri treniranju modela za objektnu detekciju. Nadalje, opisani su alati korišteni za objektnu detekciju poput TensorFlow-a i TensorFlow API-a za objektnu detekciju (engl. *TensorFlow Object Detection API*). Drugi dio rada se sastoji od opisa korištenja Google Colab alata gdje je opisan svaki programski kod korišten za izradu navedenog rješenja za treniranje modela. Također, prikazani su i uspoređeni rezultati različitih rješenja.

# 2. STROJNO UČENJE

## 2.1. Umjetna inteligencija

Umjetna inteligencija (engl. *Artificial intelligence*, skraćeno AI) je dio računalnih znanosti koja se bavi izradom inteligentnih računalnih sustava koji predočavaju karakteristike koje se povezuje s inteligencijom u ljudskom ponašanju. Pojam se može primijeniti na bilo koji stroj koji pokazuje osobine povezane s ljudskim umom poput učenja i rješavanja problema.

Glavna karakteristika umjetne inteligencije bi bila sposobnost poduzimanja onih radnji koji imaju najbolje šanse za postizanje određenog cilja. Glavni podskup umjetne inteligencije bi bilo strojno učenje koje se odnosi na koncept gdje računalni programi mogu automatski učiti i prilagoditi se novim podacima bez ljudske pomoći. Time se, pomoću dubokog učenja, može automatski učiti puno podataka poput teksta, slika ili videa.

Istraživači i programeri na polju dubokog učenja čine iznimno brze korake oponašajući funkcioniranje ljudskog mozga poput učenja, rasuđivanja i percepcije. Ono što je prije 10 godina bilo na vrhu sada se gleda kao zastarjela tehnologija što je moguće vidjeti kod dubokih konvolucijskih mreža koje će biti objašnjene u radu. Samim time, AI se stalno razvija u korist mnogih znanosti i industrija.

Umjetnu inteligenciju dijelimo na dvije kategorije: slabu i jaku.

Slaba umjetna inteligencija (engl. *Weak artificial intelligence*) se sastoji od sustava koji su osposobljeni i usredotočeni za izvršavanje određenog skupa zadataka. Ovaj tip umjetne inteligencije pokreće većinu umjetne inteligencije koja nas okružuje. Primjer sustava koji utjelovljuju slabu umjetnu inteligenciju bi bili: Apple Siri, Amazon Alexa, Google Assistant i slično.

Snažnu umjetnu inteligenciju čine sustavi koji izvršavaju zadatke koji se smatraju ljudskim. To su složeniji sustavi kojih čine umjetna opća inteligencija (engl. *Artificial General Intelligence*, skraćeno AGI) i umjetna super inteligencija (engl. *Artificial Super Intelligence,* skraćeno ASI).

Umjetna opća inteligencija je teorijski oblik umjetne inteligencije gdje bi stroj imao inteligenciju jednaku kao čovjek. Takva inteligencija bi imala svijest sa sposobnošću rješavanja problema, učenja i planiranja budućnosti.

Umjetna super inteligencija bi nadmašila sposobnost mozga čovjeka. Ovakav tip umjetne inteligencije je još uvijek u potpunosti teoretski i nema praktičnih primjera u današnjoj upotrebi.

Primjeri snažne umjetne inteligencije bi bili autonomna vozila i strojevi u bolničkim operacijskim salama [1] [2].

### 2.1.1. Strojno učenje i duboko učenje

Iako se često pojmovi strojno učenje i duboko učenje koriste naizmjenično potrebno je navesti razlike. Riječ je o dijelovima umjetne inteligencije gdje je duboko učenje podskup strojnog učenja. Hijerarhiju podskupova umjetne inteligencije prikazuje slika 1.

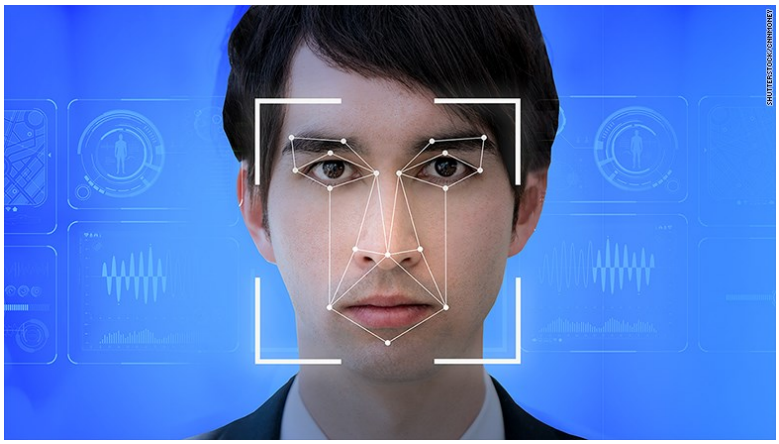

**Slika 1** Hijerarhija umjetne inteligencije

Ono po čemu se strojno učenje i duboko učenje razlikuju je po tome što se duboko učenje sastoji od neuronskih mreža. „Duboko“ u riječi duboko učenje predstavlja neuronsku mrežu koja se sastoji od 3 ili više slojeva [1] [2]. Duboke neuronske mreže će biti objašnjene u poglavlju 2.5. „Duboke neuronske mreže“.

## 2.2. Primjene umjetne inteligencije i strojnog učenja

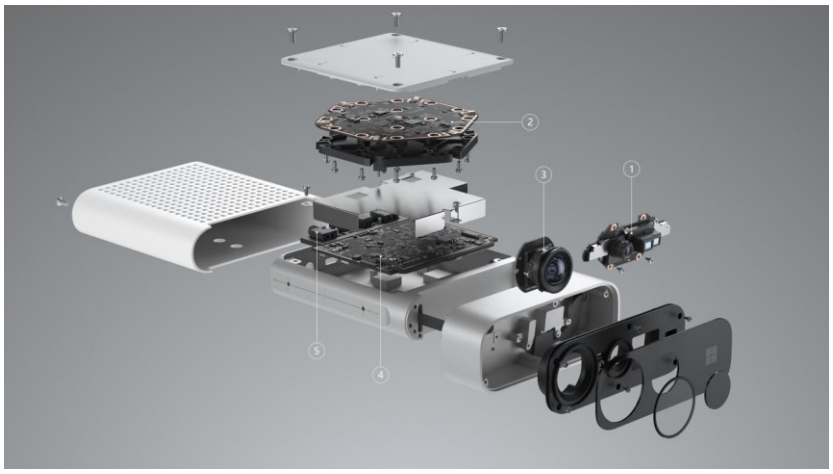
Strojno učenje ima veliku ulogu u poslovanju i korištenju svega povezanog s modernom tehnologijom. Koristi se u svim granama za poboljšanje konkurentnosti i rješavanja problema koji drugačije ne bi bili mogući.

Primjer primjene strojnog učenja bi bilo Facebook prepoznavanje lica na fotografijama koje ljudi svakodnevno susreću koristeći Facebook platformu. Riječ je o objektnoj detekciji koju je Facebook napravio radi lakšeg označavanja prijatelja na slikama. Slika 2 prikazuje objektnu detekciju lica.



**Slika 2** Prepoznavanje lica [3]

Drugi primjer bi bio Xbox Kinect (slika 3) koji, koristeći algoritam „Random Forest“, ima sposobnost prepoznavanja gesti, glasovnih naredbi i objekata u videu. Sve te potrebne podatke hvata pomoću raznih senzora koji skupljaju veliku količinu podataka. Ti podaci se zatim kroz algoritam umjetne inteligencije prosljeđuju te se stvara klasifikacija objekata potrebna za prepoznavanje gesti koje korisnik učini. Riječ je o uređaju od tvrtke Microsoft koji je služio za drugačiji oblik igranja igara bez direktnog kontakta s tipičnom periferijom za igranje [4].



**Slika 3** Xbox Kinect [4]

Među primjene ubrajamo i autonomna vozila koja zahvaljujući dubokom učenju mogu raspoznati prometne znakove, pješake, druga auta i kretati se u prometnoj okolini. Kod autonomnih automobila računalni sustav mora moći uzeti u obzir sve vanjske podatke, izračunati ih i kretati se uz minimalnu posljedicu mogućeg sudara.

Također, u umjetnu inteligenciju spadaju i brojne druge stavke kao što su: algoritmi za praćenje korisnika kako bi im se prikazali relevantni oglasi, pretvaranje teksta u govor (s točnim naglaskom), u medicini, u prijevodu stranih jezika i sličnim područjima gdje se nalaze velike količine podataka.

### 2.3. Funkcija gubitka (engl. *Loss function*)

Strojevi uče pomoću funkcije gubitka. To je metoda koja procjenjuje koliko dobro specifični algoritam modelira dane podatke. Kada bi predviđanja previše odstupala od stvarnih rezultata, onda bi funkcija gubitka iznosila veliki broj. Nadalje, uz pomoć funkcije optimizacije, funkcija gubitka uči smanjivati pogrešku u predviđanju.

Postoji više različitih funkcija gubitka, a odabir određene ovisi o brojnim stavkama poput vrste odabranog algoritma za učenje kao primjera jedne od njih.

Funkcije gubitka mogu se podijeliti u dvije kategorije: klasifikacijski gubici (engl. *Classification losses*) i regresijski gubici (engl. *Regression losses*).

Regresijski gubitak koristimo kada predviđamo kontinuirane vrijednosti poput cijene nekretnina ili prodaje tvrtke. Jedan od primjera funkcije regresijskog gubitka bi bila srednja kvadratna pogreška (engl. *Mean squared error*).

Formula srednje kvadratne pogreške bi glasila:

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n} \quad (1)$$

Srednja kvadratna pogreška se mjeri kao prosjek razlike između stvarne i predviđene vrijednosti. Zbog kvadrata, one vrijednosti koje su dosta različite od stvarnih vrijednosti bivaju strogo kažnjavane u usporedbi s vrijednostima s manjim odstupanjima.

Klasifikacijski gubitak se koristi kada se pokušavaju predvidjeti izlazi iz skupa konačnih vrijednosti kategorija.

Primjer funkcije klasifikacijskog gubitka bi bio gubitak unakrsne entropije (engl. *Cross entropy loss*). Gubitak unakrsne entropije raste kako se predviđena vjerojatnost odudara od stvarne oznake. Stvarna oznaka može imati vrijednosti 0 i 1. Dakle predviđanje vjerojatnosti od .01 kada je stvarna oznaka promatranja 1 bi bilo loše i rezultiralo velikom vrijednošću gubitka.

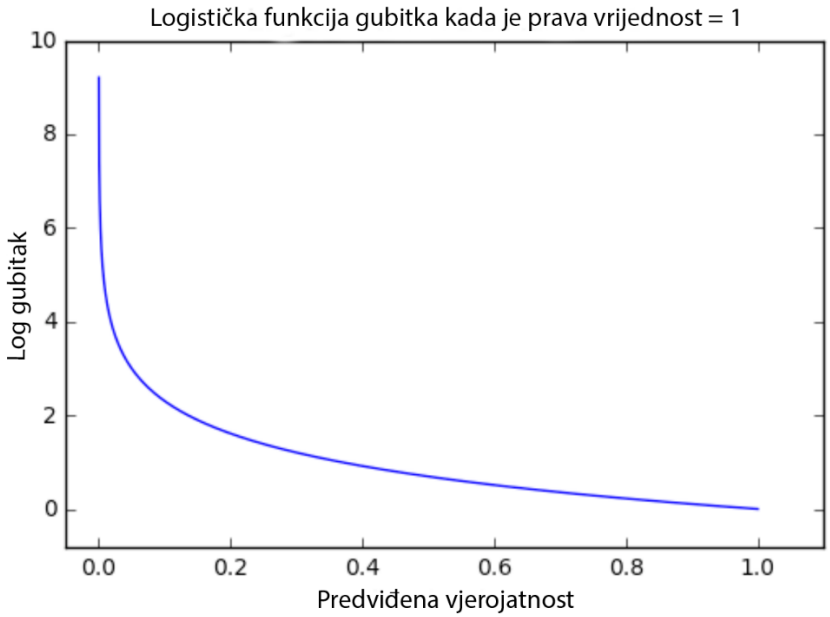
Formula gubitka unakrsne entropije bi glasila:

$$Gubitak\ unakrsne\ entropije = -(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)) \quad (2)$$

Ono što se može primijetiti je da ako je stvarna oznaka 1 ($y_i = 1$) druga polovica funkcije nestaje, dok ako je stvarna oznaka 0 ($y_i = 0$) prva polovica funkcije nestaje.

Unakrsna entropija je također povezana i često se miješa s logističkom funkcijom gubitka. Iako je riječ o mjerama koje su izvedene iz različitog izvora, kada se koriste kao funkcije gubitka za klasifikacijske modele obje mjere obavljaju jednaku funkciju.

Slika 4 prikazuje raspon mogućih vrijednosti gubitaka s obzirom na istinito opažanje, tj. u kojem je prava vrijednost 1 (na primjer osoba nosi masku = 1).




**Slika 4** Logistička funkcija gubitka [5]

Kako se predviđena vjerojatnost približava 1, gubitak se polako smanjuje. Međutim, ako se predviđena vjerojatnost smanjuje, gubitak brzo raste. Time je moguće zaključiti da navedena funkcija jako kažnjava vrijednosti koje su pogrešne s velikim odstupanjem [5] [6].

### 2.4. Srednja prosječna preciznost (engl. *mean average precision,* skraćeno mAP) i srednji prosječni odziv (engl. *average recall,* skraćeno mAR)

U računalnom vidu, točnije objektnoj detekciji, koristi se srednja prosječna preciznost (engl. *Mean average precision*, skraćeno mAP) za procjenu točnosti otkrivanja objekata (klasifikacija i lokalizacija). Klasifikacija određuje klasu objekta (npr. s maskom ili bez maske), dok lokalizacija određuje samu lokaciju objekta (koordinate okvira unutar kojeg se nalazi objekt).

Mnogi algoritmi koriste navedenu metriku za određivanje svoje preciznosti. Neki od njih bi bili: SSD ResNet, EfficientDet, *Faster* R-CNN i YOLO.

Da bi se mogla opisati srednja prosječna preciznost potrebno je objasniti preciznost i odziv.

Preciznost definira koliko se može pouzdati u pozitivno predviđanje, tj. koliko je uzoraka za koje je model rekao da su pozitivni stvarno točno.

To bi se prikazalo preko formule:

$$Preciznost = \frac{TP}{TP + FP} \tag{3}$$

Gdje bi kratica TP značila točno pozitivni (engl. *true positive*) i FP lažno pozitivni (engl. *false positive*).

Međutim, navedena formula nije točno prosjek preciznosti za objektnu detekciju. To se može objasniti na primjeru u kojem imamo detekciju koja je ispravno, ali ne u potpunosti lokalizacijom točno, označena.

Time bi se moglo označiti da je TP = 1, dok je FP = 0. Kad bi se to uvrstilo u formulu dobilo bi se:

$$Preciznost = \frac{TP}{TP + FP} = \frac{1}{1 + 0} = 1 \tag{4}$$

S obzirom na to da imamo samo vrijednost klasifikacije, srednja preciznost bi bila jednaka 1 što nije u potpunosti točno. Objektna detekcija, osim klasifikacije, također obavlja i lokalizaciju okvira. Samim time, potrebno je za svaki okvir mjeriti preklapanje predviđenog graničnog okvira i već označenog graničnog okvira temeljne istine (engl. *Ground Truth*). To se mjeri putem metrike IoU (omjer presjeka i unije, engl. *Intersection over Union*) [7].

Formula bi glasila:

$$IoU = \frac{Presjek\ graničnih\ okvira}{Unija\ graničnih\ okvira} \tag{5}$$

Za zadatke objektne detekcije, srednja prosječna preciznost se računa koristeći IoU vrijednost uz odgovarajući IoU prag. Na primjer, ako je IoU prag 0.5, a IoU vrijednost za predviđanje je 0.8, tada predviđanje klasificiramo kao točno pozitivno (TP). S druge strane, ako je IoU 0.4, klasificiramo ga kao lažno pozitivno (FP). Sam prag se može mijenjati te tako IoU 0.4 kod praga 0.3 bi predstavljao točno pozitivnu, dok kod praga 0.5 lažno pozitivnu detekciju.

Osim IoU metrike, za srednju prosječnu preciznost je važna i ocjena pouzdanosti (engl. c*onfidence score*) koja bi označavala vjerojatnost da okvir sadrži objekt.

Samim time, da bi detekcija bila točno pozitivna mora zadovoljavati tri kriterija. Prvi bi bio da je ocjena pouzdanosti veća od zadanog praga. Drugi kriterij bi bio da predviđena klasa odgovara pravoj klasi, dok bi treći kriterij bio da predviđeni okvir ima IoU veći od praga.

U slučaju da je bilo koji od zadnja dva kriterija netočan, predviđena detekcija se smatra lažno pozitivnom. U slučaju da je prvi kriterij netočan, to jest da je ocjena pouzdanosti detekcije manja od praga, a trebao bi označavati istinitu detekciju, detekcija se smatra lažno negativnom.

Za mjerenje srednje prosječne preciznosti, osim preciznosti, potrebna je i metrika odziva (engl. *recall*).

Metrika odziva (engl. r*ecall*) mjeri koliko je model dobar u pogađanju točne klase, tj. od pozitivnih klasa koliko ih je algoritam točno pogodio.

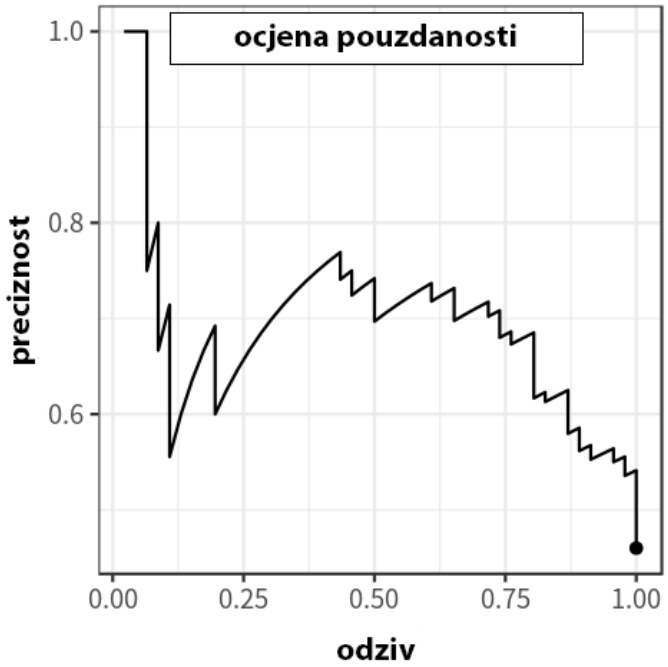
Jednadžba bi glasila:

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

Gdje bi kratica TP značila točno pozitivni, a FN lažno negativni (engl. *false negative*).

Zanemarujući lokalizaciju, razlika između preciznosti i odziva se može objasniti na sljedećem primjeru. Kad bi se gledalo na primjeru detekcije zaštitnih maski na licu da se na nekoj slici nalazi 5 osoba s maskom. Model predikcijom vrati tri detekcije, od kojih su dvije točne i jedna lažno pozitivna. U tom slučaju odziv bi bio 40% (dvije je detekcije algoritam pogodio od ukupno pet), dok bi preciznost iznosila 67% (od tri označene predikcije od strane modela, dvije su točno označene i klasificirane).

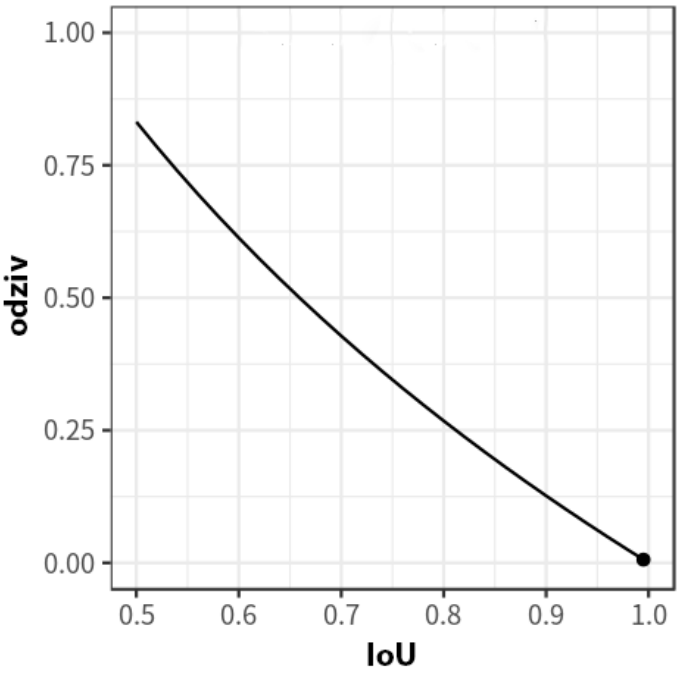
Postavljanjem različitih pragova za ocjenu pouzdanosti, dobivamo različite parove preciznosti i odziva. Time se može nacrtati krivulja preciznosti-odziva koja pokazuje povezanost između dvije metrike. Na navedenoj krivulji odziv bi se prikazivao na x osi, a y os bi prikazivala metriku preciznosti. Sljedeća slika prikazuje krivulju preciznost-odziv [7] [8].




**Slika 5** Krivulja preciznost-odziv [8]

Iz krivulje je moguće vidjeti da smanjivanjem praga za ocjenu pouzdanosti, metrika odziva raste, dok preciznost ima tendenciju da se smanjuje.

Osim navedene krivulje postoji i odziv-IoU krivulja koja je temelj metrike prosječnog odziva.




**Slika 6** Krivulja odziv-IoU [8]

Na krivulji je vidljivo da se metrika odziva smanjuje kako se IoU povećava [7] [8].

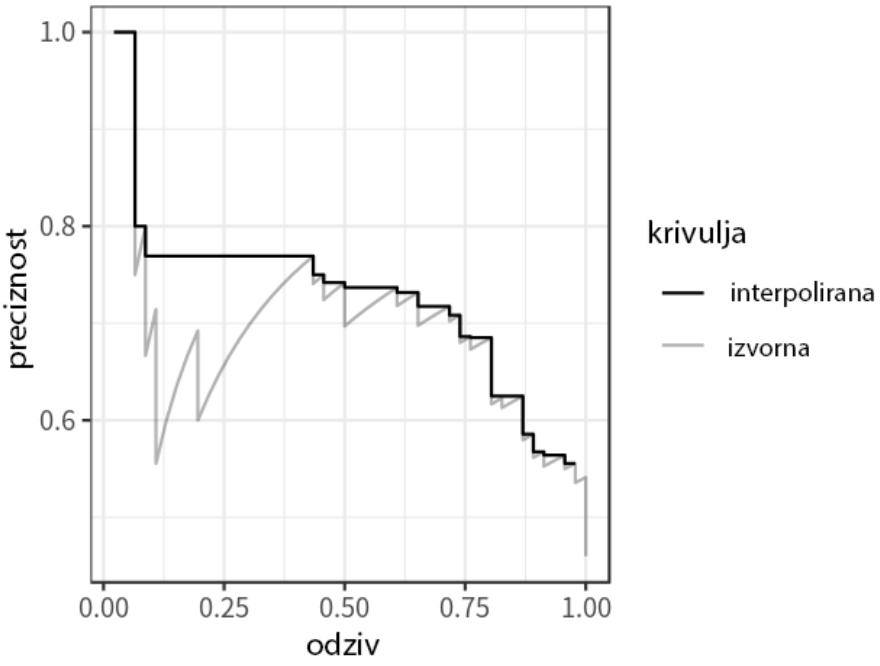
### 2.4.1. Prosječna preciznost (engl. *average precision,* skraćeno AP) i srednja prosječna preciznost (skraćeno mAP)

Prosječna preciznost je najvažnija metrika u objektnoj detekciji za provjeru performansi modela, tj. detektora. Zasniva se na krivulji preciznost-odziv (P-R). Riječ je o prosjeku preciznosti na svim jedinstvenim vrijednostima metrike odziva.

Zbog varijabilnosti preciznosti na krivulji preciznost-odziv, potrebno je krivulju interpolirati prije izračuna prosječne preciznosti. Interpolacija označava konstrukciju novih točaka unutar skupa poznatih točaka podataka te bi u ovom slučaju predstavljala prilagodbu krivulje. Interpolacija bi funkcionirala tako da se bilo koja preciznost na određenoj razini odziva r definira kao najveća preciznost za sve preciznosti nakon nje, tj. gdje je vrijednost odziva veća ili jednaka od r (r' >= r). Sljedeća jednadžba pokazuje opisani princip:

$$p_{interp} = \max_{r' \geq r} p\,(r') \tag{7}$$

Sljedeća slika prikazuje kako bi nakon interpolacije izgledala nova krivulja.




**Slika 7** Interpolacija krivulje preciznost-odziv [8]

Nakon interpolacije, prosječna preciznost bi se izračunala kao vrijednost ispod interpolirane krivulje preciznost-odziv preko sljedeće formule:

$$AP = \sum_{i=1}^{n-1} (r_{i+1} - r_i) p_{interp}(r_{i+1}) \tag{8}$$

Gdje bi $r_1$, $r_2$, …, $r_n$ označavali razine odziva (u rastućem redoslijedu) na kojima se preciznost prvo interpolira [7] [8].

S obzirom na to da detektor većinom ne detektira samo jednu klasu, najkorištenija metrika kod objektne detekcije bi bila srednja prosječna preciznost (engl. *mean average precision*, skraćeno mAP) koja bi predstavljala srednju vrijednost prosječne preciznosti za K razreda te bi jednadžba glasila:

$$mAP = \frac{\sum_{i=1}^{K} AP_i}{K} \tag{9}$$

### 2.4.2. Prosječni odziv (engl. *average recall,* skraćeno AR) i srednji prosječni odziv (skraćeno mAR)

Prosječni odziv je metrika korištena često u objektnoj detekciji uz metriku prosječne preciznosti. Prosječni odziv se može opisati kao prosjek odziva za sve IoU $\in [0.5, 1.0]$ i računa se kao dva puta veća površina od odziv-IoU krivulje preko sljedeće jednadžbe:

$$AR = 2 \int_{0.5}^{1} recall(o) do \tag{10}$$

gdje bi o predstavljalo IoU, a recall(o) odgovarajući odziv za navedeni IoU [7] [8].

Srednji prosječni odziv se definira kao srednja vrijednost prosječnog odziva za sve razrede i računa se preko sljedeće jednadžbe:

$$mAR = \frac{\sum_{i=1}^{K} AR_i}{K} \tag{11}$$

### 2.4.3. COCO varijante za metrike mAP i mAR

Za evaluaciju i testiranje skupa podataka za ovaj rad koristit će se COCO (engl. *Common Object in Context*) metrike koje uključuju mAP i mAR metrike u više varijanti.

S obzirom na IoU postoje sljedeće varijante metrike mAP:

- $mAP^{IoU = .50:.05:.95}$ što predstavlja mAP izračunatu iz prosjeka od 10 IoU pragova i primarna je metrika za provjeru
- $mAP^{IoU =.50}$ koji predstavlja mAP na IoU pragu od 0.5
- $mAP^{IoU =.75}$ koji predstavlja mAP na IoU pragu od 0.75

S obzirom na veličinu objekta postoje sljedeće varijante metrike mAP:

- $mAP^{small}$ koji predstavlja mAP za male objekte koji pokrivaju područje koje je manje od $32^2$ piksela
- $mAP^{medium}$ koji predstavlja mAP za srednje velike objekte koji pokrivaju područje manje od $96^2$ (a veće od $32^2$) piksela
- $Map^{large}$ koji predstavlja mAP za velike objekte koji pokrivaju područje veće od $96^2$ piksela

S obzirom na broj detekcija na slici postoje sljedeće varijante metrike mAR:

- $mAR^{max=1}$ koji predstavlja mAR s jednom detekcijom po slici
- $mAR^{max=10}$ koji predstavlja mAR s 10 detekcija po slici
- $mAR^{max=100}$ koji predstavlja mAR sa 100 detekcija po slici

S obzirom na veličinu objekta postoje jednake varijante metrike mAR kao kod metrike mAP (*small, medium, large*) [8].

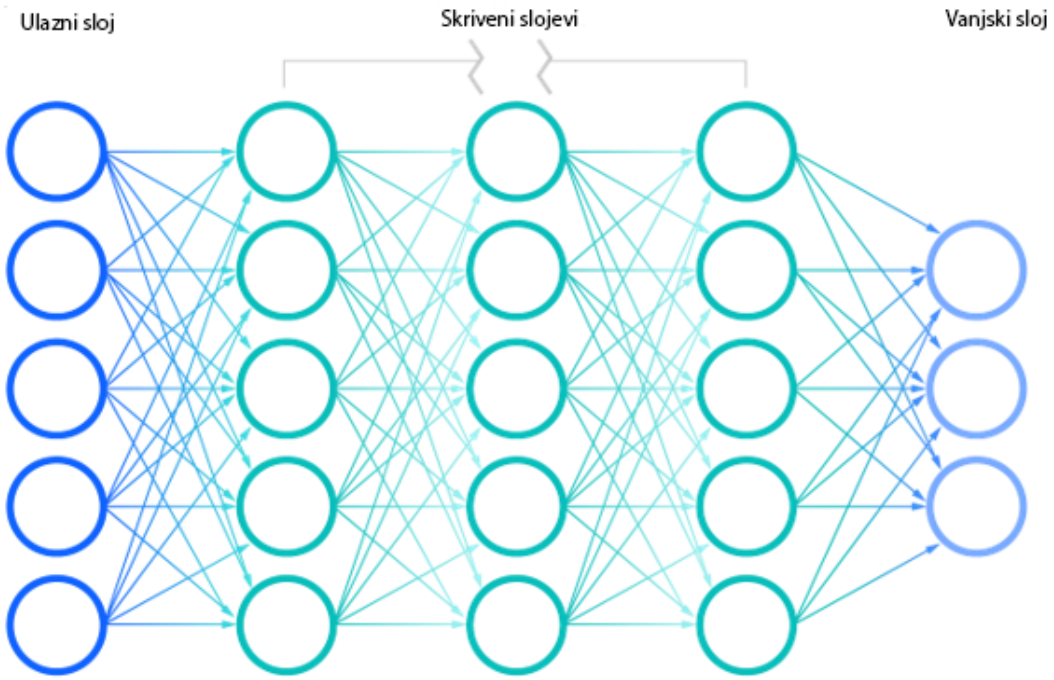
### 2.5. Umjetne neuronske mreže

Umjetne neuronske mreže se može svrstati u metode umjetne inteligencije, točnije dubokog učenja. Cilj im je preko oponašanja funkcioniranja ljudskog mozga prepoznati odnose između velikih količina informacija. Korišteni su u raznim područjima poput: financija, trgovanja dionicama, predviđanja meteoroloških podataka, kao i računalnom vidu za detekciju, klasifikaciju i brojnim drugim djelatnostima. Kod računalnog vida se govori o konvolucijskim neuronskim mrežama objašnjenima u sljedećem poglavlju.

Umjetne neuronske mreže se prvi put pojavljuju 1943. godine u radu napisanom od strane znanstvenika Warren McCullough-a i Walter Pitts-a. Navedeni znanstvenici su koristili vrlo

jednostavan model neurona koji, poput biološkog neurona, obrađuje signale putem sinaptičke i somatske operacije. Takav jednostavan model neurona je dobio naziv perceptron.

Neuronske mreže su sastavljene od tri sloja: ulaznog, jednog ili više skrivenih slojeva i izlaznog sloja. Ulazni sloj prima podatke, skriveni slojevi ih obrađuju, a izlazni sloj daje rezultat. Svaki umjetni neuron se povezuje s drugim i ima povezanu težinu i prag. Ako je izlaz bilo kojeg pojedinačnog neurona iznad navedene vrijednosti praga, taj neuron se aktivira i šalje podatke sljedećem sloju mreže. Inače, ako je izlaz ispod vrijednosti praga, podaci se ne prenose na sljedeći sloj mreže. Izgled neuronske mreže prikazuje slika 8.




**Slika 8** Neuronske mreže [9]

Svaki pojedinačni čvor se može zamisliti kao vlastiti model linearne regresije koji se sastoji od: ulaznih podataka, težina, odstupanja i rezultata. Formula bi se prikazala na sljedeći način:

$$\sum_{i=1}^{m} x_i w_i + b \tag{12}$$

Gdje bi $x_i$ predstavljao ulaz, w težinu, i b odstupanje (engl. *bias*).

Nakon što je određen ulazni sloj, težine su dodijeljene. Težine pomažu u određivanju važnosti bilo koje varijable, s tim da one veće doprinose značajnijem rezultatu u usporedbi s ostalima. Svi se ulazi pomnože s pripadajućim težinama i zatim zbroje. Nakon toga, izlaz se prenosi kroz

aktivacijsku funkciju, koja određuje izlaz. Ako izlaz pređe zadani prag, on aktivira čvor i prosljeđujući podatke sljedećem sloju u mreži. To rezultira izlazom jednog čvora na ulazu sljedećeg čvora [9].

Postoje brojne vrste neuronskih mreža koje se koriste za različite slučajeve. Jedan od primjera bi bile periodične neuronske mreže koje se koriste za obradu prirodnog jezika i prepoznavanje govora, dok se konvolucijske neuronske mreže (skraćeno ConvNets ili CNN) češće koriste za klasifikaciju i računalni vid.

### 2.5.1. Konvolucijske neuronske mreže

Konvolucijska neuronska mreža (engl. *Convolutional Neural Network*, skraćeno ConvNet / CNN) je algoritam dubokog učenja koji može uzeti ulaznu sliku, dodijeliti odstupanje i težine različitim objektima na slici i biti ih u stanju razlikovati. Predobrada potrebna u konvolucijskim mrežama je niža u usporedbi s drugim algoritmima klasifikacije. Iako su u primitivnim metodama filtri ručno projektirani, konvolucijske mreže imaju mogućnost naučiti te filtre preko obuke.

Konvolucijska neuronska mreža je slična samoj neuronskoj mreži po tome što je također sastavljena od neurona. Svaki neuron prima ulaze, računa zbroj umnožaka ulaza i težina što se šalje aktivacijskoj funkciji i vraća izlaz. Ono što je različito je da su ulazi u konvolucijskim neuronskim mrežama višekanalne slike, umjesto vektora kao u neuronskim mrežama.

Prije konvolucijskih neuronskih mreža koristile su se ručne metode ekstrakcije značajki za prepoznavanje objekata na slikama. Međutim, konvolucijske neuronske mreže sada pružaju skalabilniji pristup klasifikaciji slika i prepoznavanju objekata koristeći principe linearne algebre, kao što je umnožavanje matrice, kako bi se identificirali uzorci unutar slike.

Konvolucijske neuronske mreže imaju tri glavne vrste slojeva, a to su: konvolucijski sloj (engl. C*onvolutional layer)*, sloj sažimanja (engl. *Pooling layer)* i potpuno povezani sloj (engl. *Fully-connected layer,* skraćeno FC).

Konvolucijski sloj je prvi sloj konvolucijske mreže. Prate ga dodatni konvolucijski slojevi ili slojevi sažimanja, dok je potpuno povezani sloj završni sloj. Sa svakim slojem konvolucijska mreža se povećava u svojoj složenosti identificirajući veće dijelove slike. Početni slojevi baziraju se na jednostavnim značajkama poput boja i rubova. Kako podaci slike napreduju kroz

slojeve konvolucijske mreže tako ona počinje prepoznavati veće elemente ili oblike predmeta, dok konačno ne identificira željeni objekt.

U konvolucijskom sloju se javlja većina računanja. Zahtijeva nekoliko komponenata poput ulaznih podataka, mape značajki i filtera. Ako je ulazni podatak slika u boji koja se sastoji od matrice piksela u 3D-u, onda će ulaz imati tri dimenzije: visinu, širinu i dubinu. Dubina odgovara RGB-u na slici.

Filter je dvodimenzionalna matrica težina koja predstavlja dio slike. Filteri mogu biti različitih veličina, ali najčešća veličina je 3x3 matrica. Filter će se kretati po prihvatljivim poljima slike, provjeravajući je li značajka prisutna. Taj se postupak naziva konvolucija. Pojam konvolucija se odnosi na matematičku kombinaciju dviju funkcija kako bi se dobila treća funkcija. Odrađivanjem konvolucije stvara se mapa značajki.

Kako bi se objasnio postupak konvolucije za filter se može uzeti matrica prikazana na slici 9.

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

**Slika 9** 3x3 filter

Slika se može gledati kao matrica piksela (1 i 0) određene veličine. Slika 10 prikazuje primjer slike veličine matrice 5x5.

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

**Slika 10** Matrica slike veličine 5x5

Izvršavamo konvoluciju klizanjem filtera preko matrice slike. Na svakom mjestu se vrši množenje matrica i rezultat se sprema u mapu značajki. Tako u slučaju 3x3 filtera nad 5x5 matricom slike dobit će se 9 rezultata tj. mapa značajki veličine 3x3.

Slika 11 prikazuje princip dobivanja prvog rezultata mape značajki kod konvolucije.

| | | | | |
|---|---|---|---|---|
| 1x1 | 1x0 | 1x1 | 0 | 0 |
| 0x0 | 1x1 | 1x0 | 1 | 0 |
| 0x1 | 0x0 | 1x1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

**Slika 11** Primjer klizanja i umnoška matrica

Rezultat bi bio:

$$1 \times 1 + 1 \times 0 + 1 \times 1 + 0 \times 0 + 1 \times 1 + 1 \times 0 + 0 \times 1 + 0 \times 0 + 1 \times 1 = \mathbf{4} \qquad (13)$$

Nakon svih rezultata slika 12 prikazuje dobivenu mapu značajki.

| | | |
|---|---|---|
| 4 | 3 | 4 |
| 2 | 4 | 3 |
| 2 | 3 | 4 |

**Slika 12** Dobivena mapa značajki

U ovom primjeru je prikazana operacija u 2D-u, međutim konvolucije se odrađuju u 3D-u jer je svaka slika prikazana u dimenzijama: visina, širina i dubina gdje dubina predstavlja RGB kanale slike.

U konvolucijskom sloju odrađuje se više konvolucija na slici gdje svaka operacija ima drugačiji filter što rezultira u drugačijim mapama značajki. Na kraju se sve mape značajki uzmu i stave zajedno kao konačni izlaz konvolucijskog sloja.

Kao i u ostalim neuronskim mrežama, koristi se aktivacijska funkcija kako bi izlaz bio nelinearan. Tako će izlaz konvolucije proći kroz aktivacijsku funkciju. Primjer aktivacijske funkcije bi bila funkcija ReLU (engl. *Rectified Linear Unit*).

Postoje tri parametra koja utječu na veličinu izlaza koje treba postaviti prije početka treninga neuronske mreže. To su: broj filtera, *stride* i *padding.*

Broj filtera utječe na dubinu izlaza. Tako bi n filtera dalo n različitih značajki, stvarajući dubinu od n.

*Stride* je broj piksela koju filter prelazi preko ulazne matrice (slike). Broj piksela je obično jedan, tj. filter prelazi piksel po piksel.

*Padding* se koristi jer krajnji pikseli na slici ne dobivaju šansu da dođu do centra filtra. Filter počinje od početka slike gdje lijeva strana filtera stoji na krajnje lijevoj strani slike. Filter zatim „klizi“ slikom jedan po jedan piksel sve dok desna strana filtera ne stoji na krajnje desnoj strani slike. Alternativni način dodavanja filtera slici bi bio da svaki piksel slike ima šansu da dođe do centra filtra. Bez *padding-a* to nije moguće.

*Padding* stvara dodatne piksele vrijednosti 0 (koji tijekom množenja nemaju utjecaj) na krajnjim dijelovima slike. Na primjeru 8x8 slike, dodavanjem *padding-a* bi se stvorila 10x10 matrica slike. Time kad je odrađena konvolucija s 3x3 filterom stvorila bi se 8x8 mapa značajki. Takav *padding* bi se nazvalo *same padding* jer veličina mape značajki odgovara veličini slike.

Osim *same padding-a,* postoji *valid padding* kod kojeg se ne ubacuju dodatne vrijednosti (bez *padding-a*) i *full padding* kod kojeg se dodaje dodatni broj piksela takav da svaki krajnji piksel slike dobije jednak broj izlaznih piksela kao oni blizu središta.

Kao što je već spomenuto, drugi sloj konvolucije može slijediti početni. Kada se to dogodi, struktura CNN-a može postati hijerarhijska jer kasniji slojevi mogu vidjeti piksele unutar receptivnih polja prethodnih slojeva. Tako, ako se pretpostavi da slika sadrži automobil može se razmišljati o automobilu kao skupu dijelova. Sastoji se od okvira automobila, upravljača, guma i slično. Svaki pojedini dio automobila činio bi uzorak niže razine u neuronskoj mreži, a kombinacija njegovih dijelova predstavlja uzorak više razine, stvarajući hijerarhiju značajki

unutar CNN-a. U konačnici konvolucijski sloj omogućuje neuronskoj mreži da izdvoji uzorke na slici.

Nakon konvolucijskog sloja slijedi sloj sažimanja. Sloj sažimanja provodi smanjivanje dimenzije smanjujući broj parametara na ulazu slike. Slično kao i konvolucijski sloj, operacija sažimanja (engl. *Pooling operation*) provlači filter preko ulaza, ali razlika je u tome što filter nema težinu. Umjesto toga filter primjenjuje funkciju agregiranja na vrijednosti unutar receptivnog polja.

Dvije najčešće korištene vrste sažimanja su: maksimalno sažimanje (engl. *Max pooling*) i prosječno sažimanje (engl. *Average pooling*).

Maksimalno sažimanje funkcionira tako da, dok se filter kreće po matrici slike, odabire piksel s maksimalnom vrijednošću koju će poslati u izlazni niz. Suprotno, prosječno sažimanje izračunava prosječnu vrijednost unutar receptivnog polja za slanje u izlazni niz.

Prednost sloja za sažimanje je što omogućuje smanjenje kompleksnosti, popravlja efikasnost i smanjuje rizik *overfitting*-a. Nedostatak bi bio gubljenje dosta informacija.

I za kraj, kod potpuno povezanog sloja, svaki čvor u izlaznom sloju se povezuje izravno s čvorom u prethodnom sloju. Ovaj sloj izvršava zadatak klasifikacije na temelju značajki koje su izdvojene u prethodnim slojevima i njihovih različitih filtera. Kao funkciju aktivacije, potpuno povezani sloj najčešće koristi SoftMax funkciju [10] [11].

### 2.6. Alati za strojno učenje

Za strojno učenje je moguće koristiti različite programske jezike i alate. Trenutno najpopularniji programski jezici za strojno učenje su Python i R.

Od najpopularnijih alata je vrijedno spomenuti Anaconda okruženje koje koristi alate Jupyter Notebook i Spyder za pisanje koda. Riječ je o okruženju koje se bazira na Python programskom jeziku.

U ovom radu će se koristiti Python programski jezik i Google Collaboratory (skraćeno Google Colab) okruženje napravljeno od strane tvrtke Google za pisanje i izvođenje koda u stvarnom vremenu na oblaku koristeći se Google-ovim serverima.

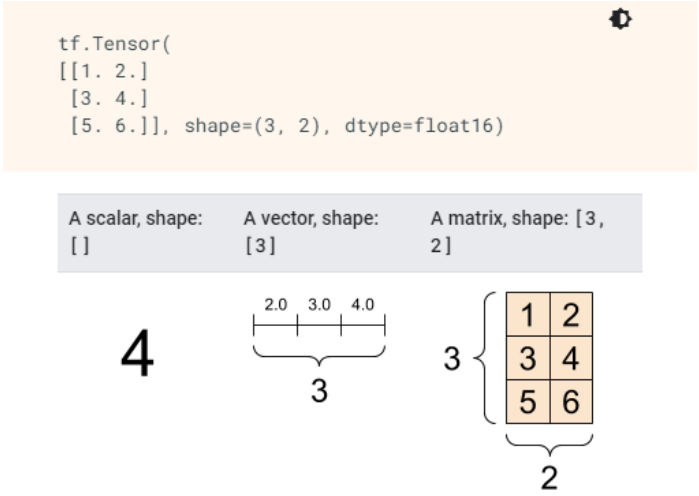
# 3. OPIS ALATA I OKRUŽENJA KORIŠTENIH ZA IZRADU PROGRAMSKOG RJEŠENJA

Za izradu modela objektne detekcije i strojnog učenja je korišten TensorFlow API za objektnu detekciju (engl. *TensorFlow Object Detection API*) baziran na platformi TensorFlow.

## 3.1. TensorFlow

TensorFlow je cjelovita platforma otvorenog koda za strojno učenje izrađena od tvrtke Google. Sadrži sveobuhvatan, fleksibilan ekosustav alata, knjižnica i resursa koji omogućuje programerima da lakše izgrade i implementiraju aplikacije pogonjene strojnim učenjem [12].

TensorFlow za izračune koristi *tensor*-e. *Tensor*-i su višedimenzionalna polja uniformnog tipa (nazvani dtype). Nalik su na numpy.array. Svi *tensor*-i su nepromjenjivi poput Python brojeva i nizova. To znači da se nikada ne može ažurirati sadržaj *tensor*-a, već samo stvoriti novi. Primjer jednostavnog *tensor*-a prikazuje slika 13.



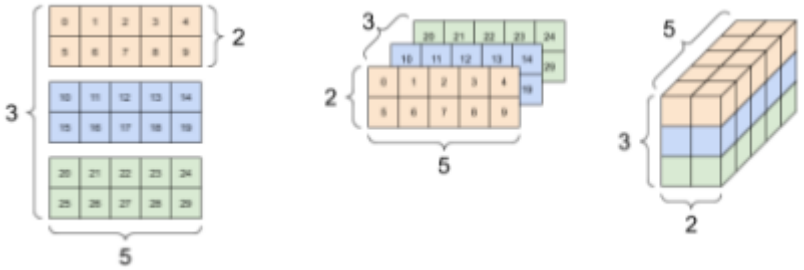

**Slika 13** Primjer *tensor*-a [12]

*Tensor*-i mogu imati i više osi. Sljedeća slika prikazuje *tensor* s tri osi.

```
tf.Tensor(
[[[ 0  1  2  3  4]
  [ 5  6  7  8  9]]

 [[10 11 12 13 14]
  [15 16 17 18 19]]

 [[20 21 22 23 24]
  [25 26 27 28 29]]], shape=(3, 2, 5), dtype=int32)
```

**Slika 14** *Tensor* s tri osi [12]

Sam *tensor* se može vizualizirati na brojne načine. Sljedeća slika prikazuje tri najčešća oblika vizualizacije *tensor*-a.




**Slika 15** Vizualizacija *tensor*-a [12]

TensorFlow spaja mnoštvo modela i algoritama strojnog učenja i dubokog učenja (tzv. neuronsko umrežavanje). Koristi Python za pružanje prikladnog *front-end* API-ja za izgradnju programa, istodobno izvršavajući te programe u C++ dobivajući visoke performanse.

Čvorovi i *tensor*-i u TensorFlow-u su Python objekti, a TensorFlow programi su Python programi, međutim stvarne matematičke kalkulacije se ne izvode u Python-u. Posebne biblioteke dostupne putem TensorFlow-a su napisane kao C binarne datoteke visokih performansi te putem njih Python služi kao usmjerivač prometa.

Uz sve navedeno, značajke koje bi se dobile korištenjem TensorFlow platforme bi bile: jednostavna izrada modela, robusno strojno učenje bilo gdje (bilo na lokalnom računalu, oblaku ili serveru) i jednostavna, fleksibilna infrastruktura za istraživanje i eksperimentiranje. Pod

jednostavna izrada modela misli se na korištenje velikog opusa biblioteka i API-a dostupnih na TensorFlow platformi kao što su Keras koji se koristi kod klasifikacije i TensorFlow *Object Detection* API kod objektne detekcije [12].

U radu će se objasniti korištenje TensorFlow platforme uz TensorFlow API za objektnu detekciju koji također koristi Keras kao dio svog opusa.

### 3.2. TensorFlow API za objektnu detekciju (engl. *TensorFlow Object Detection API*)

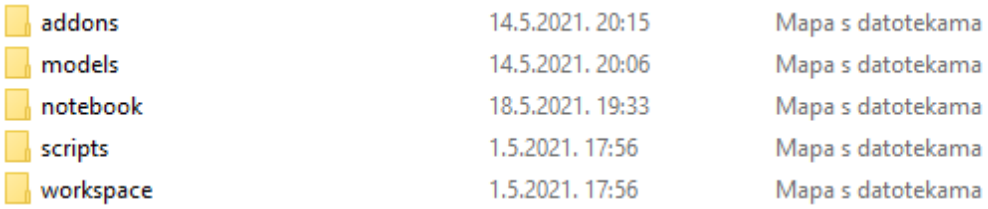
Stvaranje točnih modela strojnog učenja koji mogu lokalizirati i identificirati više objekata na jednoj slici ostaje glavni izazov u računalnom vidu. TensorFlow API za objektnu detekciju je okvir otvorenog koda izrađen na platformi TensorFlow od tvrtke Google koji olakšava izgradnju, obuku i primjenu modela za objektnu detekciju. TensorFlow API za objektnu detekciju podržava i TensorFlow 2 (skraćeno TF2) i TensorFlow 1 (skraćeno TF1) [12].

U radu će biti korišten TensorFlow druge generacije (skraćeno TF2).

# 4. PRIPREMA PODATAKA

## 4.1. Priprema mape

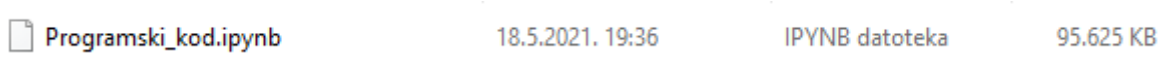
Prije treniranja modela je potrebno pripremiti mapu koja sadrži potrebne mape i datoteke koje će se koristiti za izradu i treniranje modela objektne detekcije. Sadržaj mape je prikazan na slici 16.




**Slika 16** TensorFlow mapa

Mapa je strukturirana od sljedećih mapa*: addons, models, scripts, workspace* i *notebook.*

*Notebook* mapa sadrži Google Colab/Jupyter datoteku koja sadrži sve potrebne linije koda za izradu modela. Sadržaj *notebook* mape prikazuje slika 17.

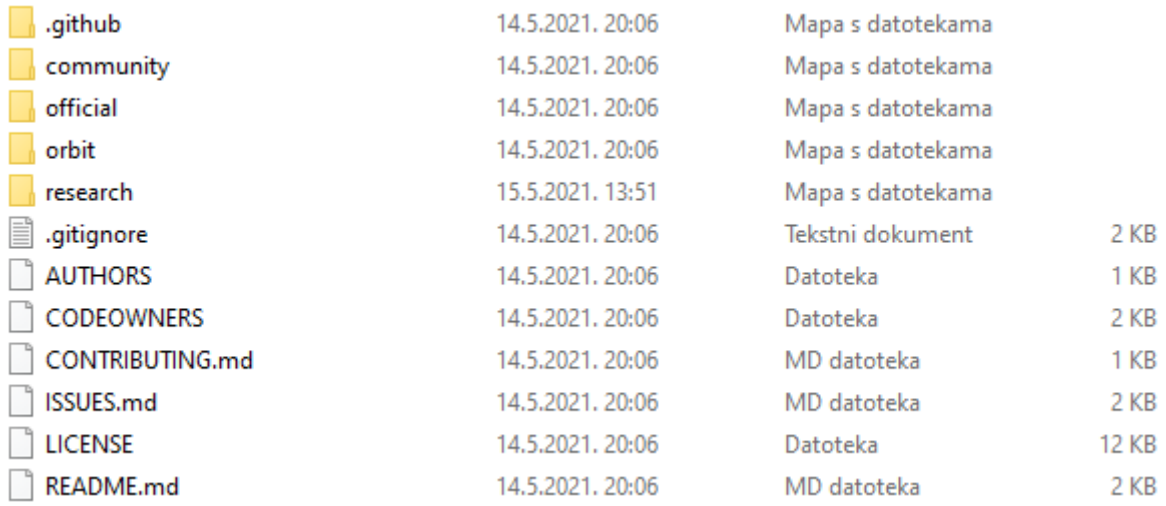



**Slika 17** *Notebook* mapa

*Addons* mapa služi za dodavanje alata koji će se koristiti kod istraživanja. Primjer bi bio LabelImg alat za izradu oznaka (*with-mask, no-mask*), s potrebnim x i y ordinatama na slici koje pokazuju osobu koja nosi zaštitnu masku na licu. Slika 18 prikazuje sadržaj *addons* mape.

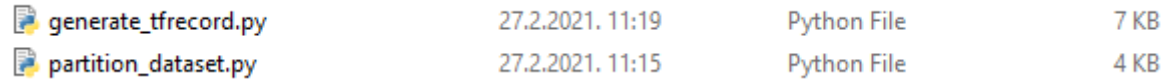

**Slika 18** *Addons* mapa

*Models* mapa služi za spremanje potrebnih alata i mapa od TensorFlow API-a za objektnu detekciju koje će se putem Google Colab *notebook*-a preuzeti. Sadržaj *models* mape je prikazan na slici 19.



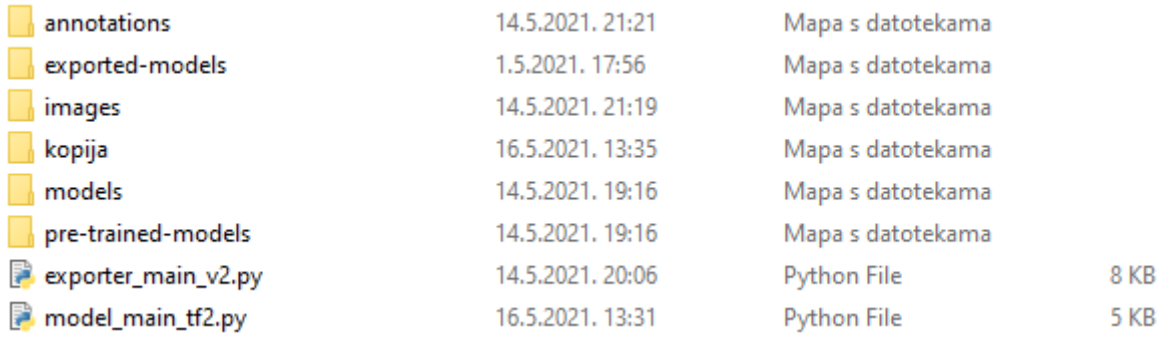

**Slika 19** *Models* mapa

*Scripts* mapa sadrži sve skripte koje su korištene u istraživanju. Unutar navedene mape nalazi se *preprocessing* mapa (slika 20) koja sadrži skripte kojima se XML datoteke koje sadrže oznake za slike, dobivene korištenjem LabelImg alata, pretvaraju u .tfrecord format specificiran od strane TensorFlow platforme. Sljedeća slika prikazuje sadržaj *scripts* mape.




**Slika 20** *Preprocessing* mapa

*Workspace* mapa sadrži *images* mapu koja sadrži slike koje su samostalno označene korištenjem LabelImg alata. Također, sadrži *annotations* mapu koja u sebi sadrži test.record i train.record datoteke koji su dobivene korištenjem LabelImg alata te sadrže oznake za vlastiti skup podataka od 1700 slika. Također, tu se nalazi i label_map.pbtxt datoteka koja sadrži nazive klasa (*with-mask*, *no-mask*) u formatu sličnom JSON formatu. Slika 21 prikazuje strukturu *workspace* mape.




**Slika 21** *Workspace* mapa

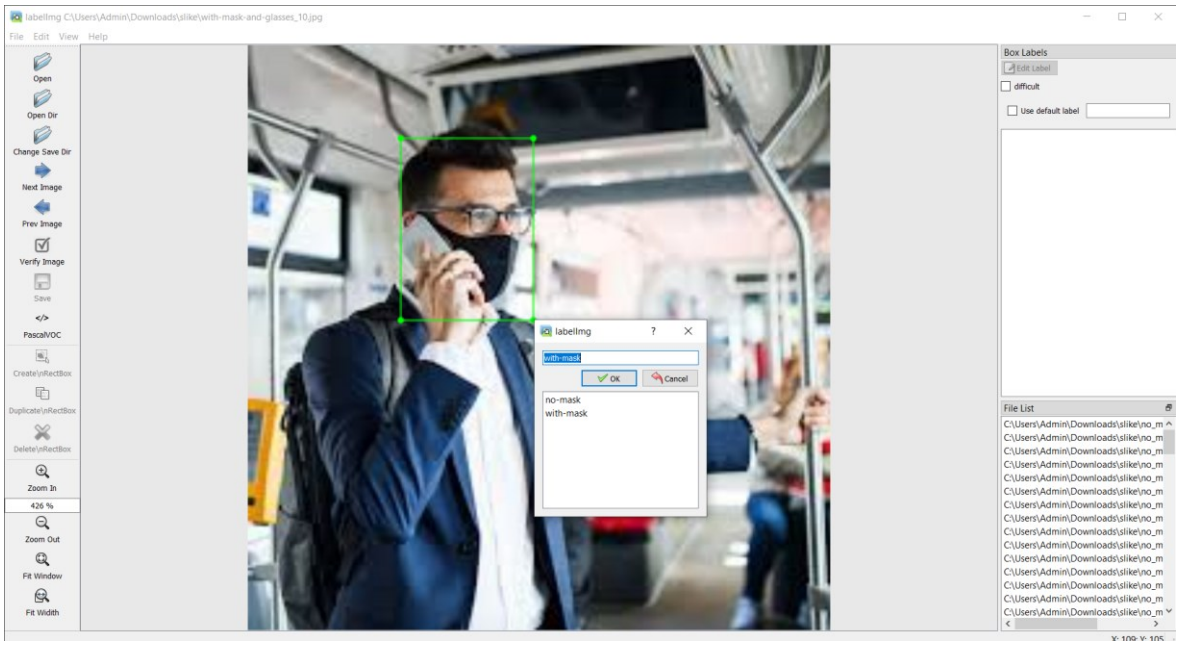
U *workspace*-u, osim *annotations* i *images* mape, nalaze se i sljedeće mape: *exported-models*, *models*, pre-*trained models*.

*Exported-models* sadrži trenirane modele izvezene u Google Colab-u spremne za testne i finalne programe. *Models* sadrži prilagođene modele za treniranje, dok pre-*trained models* sadrži modele bez uređivanja s vlastitim kontrolnim točkama (engl. c*heckpoint*) koji služe za nasljedno učenje (engl. *transfer learning*).

### 4.2. Prikupljanje i označavanje skupa podataka

Slike su prikupljene korištenjem Google slika, a označene korištenjem LabelImg alata. Ukupno je prikupljeno 1700 slika za istraživanje.

LabelImg je grafički alat za označavanje slika. Napisan je u Pythonu i koristi Qt za svoje grafičko sučelje. Bilješke se spremaju kao XML datoteke u PASCAL VOC formatu koji koristi ImageNet. Osim toga, podržava i YOLO format. Sam izgled sučelja LabelImg alata je prikazan na slici 22.



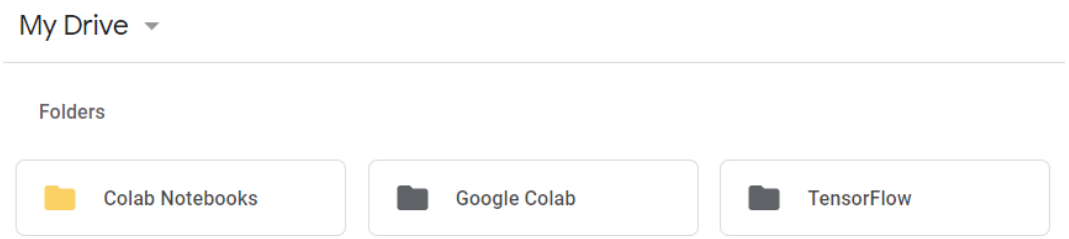

**Slika 22** Sučelje LabelImg alata

Na lijevom stupcu se nalaze sve potrebne komande u obliku tipki kojima se određuje mapa gdje se nalaze slike, mapa gdje se spremaju XML datoteke, tipke za listanje slika, tipka za označavanje detekcije i ostale komande potrebne za snalaženje u sučelju.

### 4.3. Izvoz mape na Google Disk

Za objektnu detekciju, zbog nedostatka resursa na lokalnom kompjuteru, korišten je Google Colab.

Prije korištenja *notebook*-a na navedenoj platformi potrebno je na početnu stranicu Google Disk-a prenijeti mapu objašnjenu u prethodnom poglavlju. Početne mapa Google Disk-a bi trebala sada imati TensorFlow mapu kao što prikazuje slika 23.




**Slika 23** Google Disk početna mapa

# 5. MODELI KORIŠTENI KOD ISTRAŽIVANJA

Za treniranje će se koristiti sljedeći modeli: **SSD ResNet50 V1 (RetinaNet50)** i **EfficientDet D1 640X640**. Modeli su dobiveni s *Google Models Garden* GitHub repozitorija.

Navedene (ili druge) modele moguće je preuzeti sa sljedeće poveznice: https://github.com/TensorFlow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md.

Prikaz nekih modela dostupnih na navedenoj poveznici je moguće vidjeti na slici 24.

| | | | |
|---|---|---|---|
| EfficientDet D0 512x512 | 39 | 33.6 | Boxes |
| EfficientDet D1 640x640 | 54 | 38.4 | Boxes |
| EfficientDet D2 768x768 | 67 | 41.8 | Boxes |
| EfficientDet D3 896x896 | 95 | 45.4 | Boxes |
| EfficientDet D4 1024x1024 | 133 | 48.5 | Boxes |
| EfficientDet D5 1280x1280 | 222 | 49.7 | Boxes |
| EfficientDet D6 1280x1280 | 268 | 50.5 | Boxes |
| EfficientDet D7 1536x1536 | 325 | 51.2 | Boxes |
| SSD MobileNet v2 320x320 | 19 | 20.2 | Boxes |
| SSD MobileNet V1 FPN 640x640 | 48 | 29.1 | Boxes |
| SSD MobileNet V2 FPNLite 320x320 | 22 | 22.2 | Boxes |
| SSD MobileNet V2 FPNLite 640x640 | 39 | 28.2 | Boxes |
| SSD ResNet50 V1 FPN 640x640 (RetinaNet50) | 46 | 34.3 | Boxes |
| SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50) | 87 | 38.3 | Boxes |
| SSD ResNet101 V1 FPN 640x640 (RetinaNet101) | 57 | 35.6 | Boxes |
| SSD ResNet101 V1 FPN 1024x1024 (RetinaNet101) | 104 | 39.5 | Boxes |
| SSD ResNet152 V1 FPN 640x640 (RetinaNet152) | 80 | 35.4 | Boxes |
| SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152) | 111 | 39.6 | Boxes |

**Slika 24** Modeli dostupni za TensorFlow API za objektnu detekciju

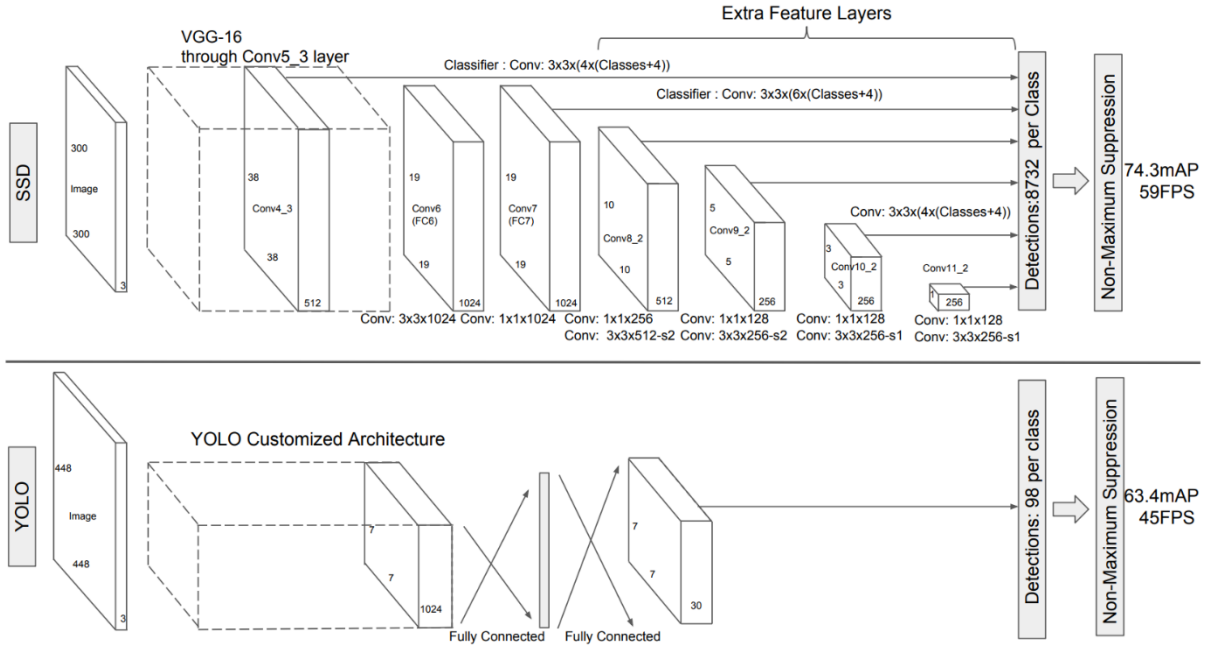
## 5.1. SSD ResNet 50 model

SSD ResNet 50 model je napravljen na SSD arhitekturi (engl. *Single-shot MultiBox Detector*) koja je bazirana na ResNet-50 mreži te je dostupan za preuzimanje s *Google Models Garden* GitHub repozitorija.

### 5.1.1. SSD (engl. *Single-shot MultiBox Detector)*

SSD je jednostupanjski algoritam za otkrivanje objekata. To znači da, za razliku od dvostupanjskih modela, SSD-ovi ne trebaju početni korak generiranja prijedloga objekata. To ga čini bržim i učinkovitijim od dvostupanjskih pristupa poput *Faster* R-CNN-a. Time žrtvuje performanse za otkrivanje malih predmeta kako bi se postigla brzina.

SSD se sastoji od dvije komponente: temeljnog modela i SSD zaglavlja. Temeljni model obično je unaprijed trenirana mreža za klasifikaciju slika. To je obično mreža poput ResNet-a trenirana na ImageNet-u s koje je uklonjen konačni potpuno povezani sloj klasifikacije. Time ostaje duboka neuronska mreža koja je sposobna izvući značajke iz ulazne slike, uz očuvanje prostorne strukture slike na nižoj rezoluciji. SSD zaglavlje je jedan ili više konvolucijskih slojeva dodanih glavnom dijelu mreže. Izlazi se tumače kao okviri i klase objekata na prostornom mjestu aktiviranja završnih slojeva. Arhitektura SSD algoritma je prikazana na slici 25.




**Slika 25** SSD arhitektura [13]

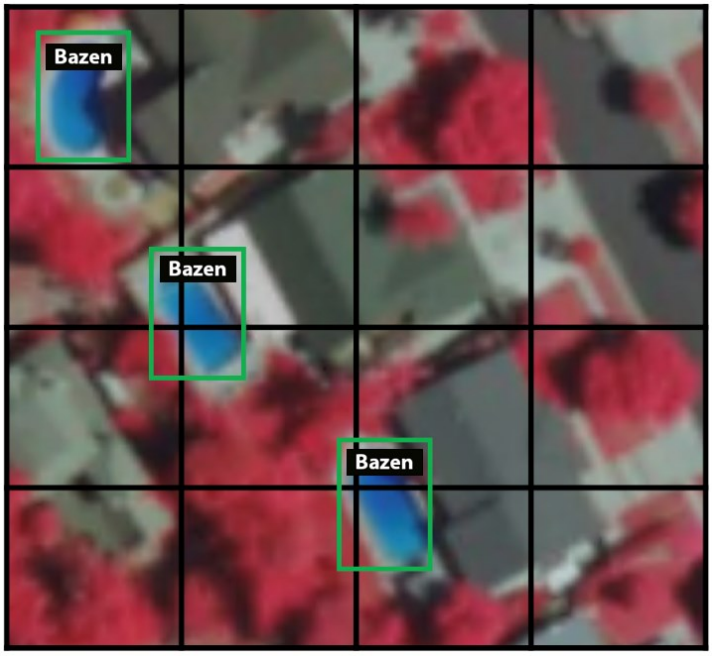
Kod izgradnje modela je moguće razmišljati o izgradnji modela za otkrivanja objekata koji se nalazi na modelu za klasifikaciju slika. Jednom kad imamo dobar model za klasifikaciju slika, jednostavan način otkrivanja objekata bi bio klizanje okvira preko slike i otkrivanje objekta.

Međutim, s takvim pristupom postoje barem dva problema. Prvo bi se postavilo pitanje kako znati veličinu okvira tako da uvijek sadrži predmet jer se u slici mogu nalaziti različite vrste

predmeta, ili isti predmet različitih veličina. Drugi problem bi bio sam omjer visine i širine graničnog okvira. Puno predmeta može biti prisutno u raznim oblicima i samim time nije moguće gledati sa samo jednim omjerom. Da bi se pogodila točna veličina okvira trebalo bi se pokušavati n puta dok se ne dobije točna veličina i omjer okvira što iziskuje veliku snagu kompjutera.

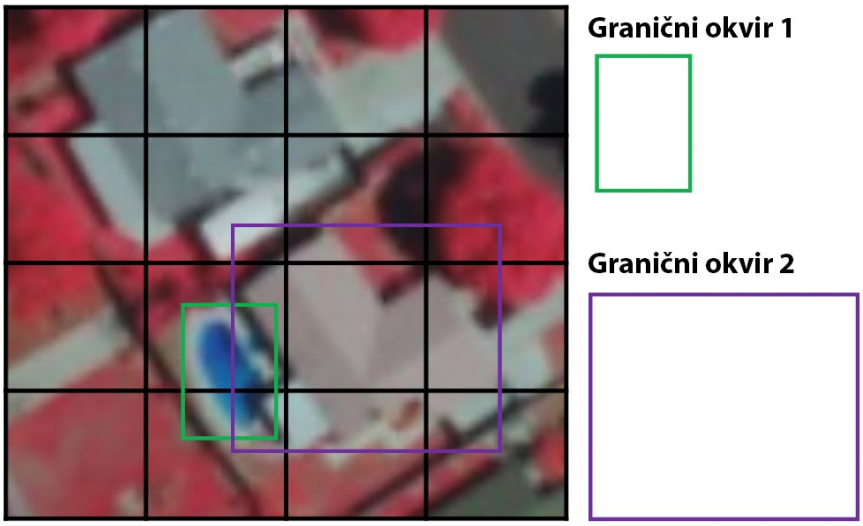
Zbog toga postoje dva oblika algoritama za objektnu detekciju. Algoritam kao što je *Faster* R-CNN koristi pristup u dva koraka. Prvo se identificiraju regije u kojima se očekuje otkrivanje objekata, a zatim otkrivaju objekte samo u tim regijama pomoću konvolucijskih mreža. S druge strane, algoritam poput SSD-a koristi potpuno konvolucijski pristup u kojem mreža može u jednom prolazu pronaći sve objekte na slici. Dvostupanjski algoritmi, poput R-CNN-a, obično imaju malo bolju preciznost, ali se sporije pokreću, dok su jednostupanjski algoritmi (SSD) učinkovitiji i imaju jednako dobru točnost [14].

SSD algoritam, umjesto korištenja kliznog okvira, dijeli sliku pomoću mreže ćelija gdje je svaka ćelija odgovorna za otkrivanje objekata u svom području slike. Slika 26 prikazuje raspored mreže s ćelijama.




**Slika 26** Primjer 4x4 mreže ćelijama [14]

Svakoj mrežnoj ćeliji može biti dodijeljeno više graničnih okvira. Ti graničnim okviri su unaprijed definirani i svaki je odgovoran za veličinu i oblik unutar mrežne ćelije. Primjer se može vidjeti na slici 27.




**Slika 27** Primjer graničnog okvira [14]

SSD koristi fazu podudaranja tijekom treninga kako bi podesio odgovarajući granični okvir s obzirom na objekt unutar slike.

U osnovi, onaj granični okvir s najvećim stupnjem preklapanja s objektom odgovoran je za predviđanje klase tog objekta i njegovog mjesta. Ovo se svojstvo koristi za obuku mreže i predviđanje otkrivenih objekata i njihovih lokacija nakon što je mreža obučena. To znači da, u vrijeme predviđanja, mreža generira bodove za prisutnost svake kategorije predmeta u svakom zadanom okviru i proizvodi prilagodbe okvira kako bi se bolje podudarao s oblikom objekta. U praksi je svaki granični okvir određen omjerom stranica i razinom zumiranja. Time imamo više graničnih okvira različitih omjera stranica i same veličine [13] [14].

### 5.1.2. ResNet 50

ResNet, skraćenica od *Residual Networks*, je jedna od najboljih dubokih neuronskih mreža. To je klasična neuronska mreža koja se koristi kao okosnica (engl. *backbone*) za mnoge zadatke računalnog vida. Postigla je izvrsne performanse i osvojila prvo mjesto na ImageNet detekciji, ImageNet lokalizaciji, COCO detekciji i COCO segmentaciji na natjecanjima ILSVRC i COCO 2015.

Postoje mnoge varijante arhitekture ResNet-a kao što su: ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-110, ResNet-152, ResNet-164, ResNet-1202.

Ono po čemu se razlikuju je po broju slojeva. U ovom radu za istraživanje je korištena, već spomenuta, ResNet-50 duboko neuronska mreža s SSD arhitekturom.

Temeljni razlog uspješnosti ResNet-a je bio taj što je omogućio uspješno treniranje izuzetno duboke neuronske mreže s više od 150 slojeva. Prije ResNet-a, trening vrlo dubokih neuronskih mreža je bio težak zbog problema nestajanja gradijenata.

ResNet arhitektura se koristi kao ekstraktor značajki za složenije arhitekture za rješavanje problema računalnog vida. U ovom slučaju je korištena sa SSD arhitekturom.

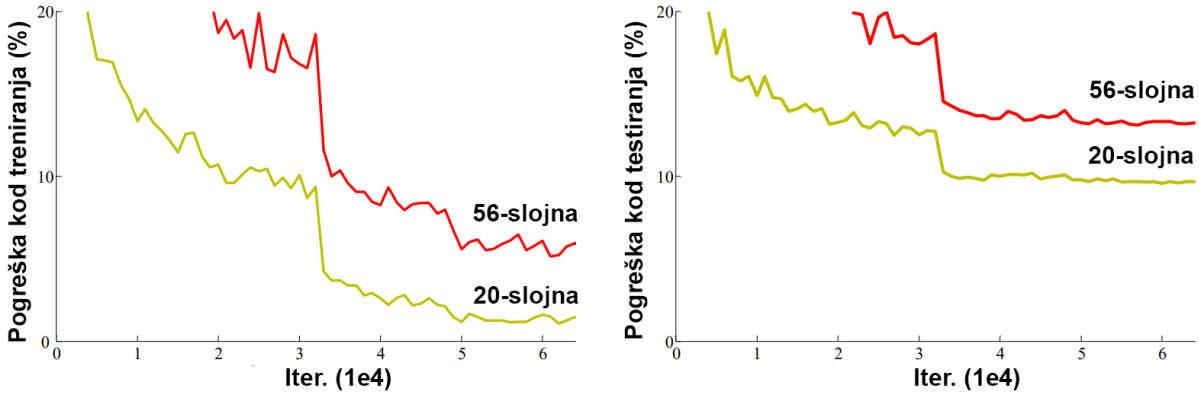
Prije opisa ResNet-a potrebno je navesti i ukratko objasniti duboke neuronske mreže koje su prethodile ResNet-u.

Prva od njih, 1997. godine, je bila LeNet. Imala je 7 slojeva koji su trebali prepoznati znamenke na bankovnim čekovima. S obzirom na slabost kompjutera u to doba, kao i mali broj slojeva, LeNet nije imala sposobnost detektiranja i klasifikacije slika.

Kao rješenje tome, 2012. godine se pojavljuje AlexNet mreža kao pobjednik, već spomenutog, ImageNet natjecanja koja je pokrenula princip dubokog učenja.

Uglavnom, u duboke neuronske mreže je potrebno slagati dodatne slojeve što rezultira poboljšanom preciznošću i performansama. Intuicija koja stoji iza dodavanja više slojeva jest ta da ti slojevi postepeno uče složenije značajke. Tako u slučaju prepoznavanja slika, prvi sloj može naučiti otkrivati rubove, drugi sloj može prepoznati teksture, dok treći sloj može naučiti otkrivati predmete.

Zbog toga je, nakon AlexNet-a, svaka mreža koja je slijedila je koristila više slojeva kako bi smanjila stopu pogrešaka. Međutim, povećanje velikog broja slojeva može dovesti do nestajanja gradijenta. Slika 28 prikazuje dva grafa koji opisuju postotak pogrešaka u podacima o treniranju i testiranju za 20-slojnu mrežu i 56-slojnu mrežu.

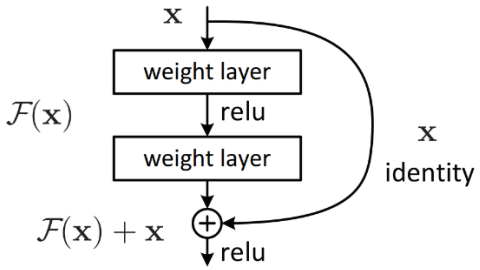



**Slika 28** Usporedba stope pogrešaka za podatke za treniranje i testiranje između 56-slojne i 20-slojne neuronske mreže [15]

Ono što je moguće vidjeti je da je postotak pogrešaka za 56-slojnu mrežu viši od 20-slojne mreže u oba slučaja (podataka o treniranju, kao i podataka o testiranju). To sugerira da se dodavanjem više slojeva na mrežu njena izvedba pogoršava. Tu može biti krivac funkcija optimizacije, inicijalizacija mreže ili nestajanje gradijenta. Netko može pomisliti da je riječ i o *overfitting-u,* ali ovdje je postotak pogreške 56-slojne mreže lošiji i u treniranju i testiranju što nije tipično za *overfitting.*

Gradijent pogreške je smjer i veličina koja se izračunava tijekom treninga neuronske mreže te se koristi za ažuriranje težina mreže prema točnom smjeru i za točan iznos. U dubokim neuronskim mrežama gradijenti pogrešaka se mogu nakupljati tijekom ažuriranja i rezultirati vrlo velikim gradijentima. To rezultira velikim ažuriranjima težina mreže što dovodi do nestabilne mreže. Te vrijednosti težina mogu postati toliko velike da dolazi do prelijevanja (engl. *overflow*) što rezultira u vrijednostima NaN (engl. *Not a Number*). Eksplozija se događa eksponencijalnim rastom uzastopnim množenjem gradijenata kroz mrežne slojeve koji imaju vrijednosti veće od 1. Slično tome, ako su vrijednosti male, gradijent će se eksponencijalno smanjivati dok se širimo kroz model dok na kraju ne nestane (vrijednost postane 0), a to je problem nestajanja gradijenta.

Kako bi se riješio problem nestajanja gradijenta, Kaiming He, Xiangyu Zhang, Shaoqing Ren i Jian Sun u svom radu „*Deep Residual Learning for Recognition Image*“ su uveli koncept nazvan *Residual Network*. U ovoj mreži se koristi tehnika koja se naziva preskakanje veza. Veza za preskakanje preskače trening iz nekoliko slojeva i povezuje se izravno s izlazom. To se može vidjeti kod *residual* blokova (slika 29) od kojih je ResNet mreža sastavljena [15].




**Slika 29** *Residual* blok [15]

Ono što se može uočiti je da postoji izravna veza (slika 19, x identity) koja preskače neke slojeve. Tu vezu se može nazvati 'preskoči vezu' i temelj je *residual* blokova. Zbog ove preskočene veze, izlaz sloja sada nije isti. Bez upotrebe ove veze za preskakanje, ulazni se 'x' pomnoži s težinama sloja nakon čega slijedi dodavanje odstupanja.

Time od formule:

$$H(x) = f(x) \tag{9}$$

Uvođenjem veze za preskakanje dolazimo do sljedeće formule:

$$H(x) = f(x) + x \tag{10}$$

Pristup koji stoji iza ove mreže je umjesto da slojevi nauče osnovno mapiranje, mi dopuštamo da mreža odgovara rezidualnom mapiranju. Prednost dodavanja preskakanja veze je ta da ako bilo koji sloj ošteti izvedbu arhitekture, tada će se preskočiti regularizacijom. Ovo rezultira treniranjem vrlo duboke neuronske mreže bez problema uzrokovanih nestajanjem gradijenta [15].

### 5.1.3. Konfiguracija SSD ResNet-50 modela

SSD ResNet 50 model je testiran u dvije konfiguracije. Obje konfiguracije modela su identično konfigurirane osim na dvjema stavkama: veličini slike i veličini grupe (engl. *batch size*).

Prva konfiguracija ima postavljenu veličinu slike na 640x640 i veličinu grupe na 8, dok druga konfiguracija ima postavljenu veličinu slike na 320x320 i veličinu grupe na 32.

Na lokaciji "./training_demo/models/my_ssd_resnet50_v1_fpn nalazi se datoteka pipeline.config na kojoj je potrebno promijeniti linije kako bi se dobio prilagođen model koji radi na navedenom skupu podataka od 1700 slika.

Stavke koje su promijenjene (ili dodane) su sljedeće:

- broj klasa
- veličina grupe
- lokacija slika za testiranje
- lokacija slika za treniranje
- lokacija dokumenata s nastavkom .tfrecord slika za testiranje i treniranje
- lokacija mape oznaka (label_map.pbtxt)
- rad na slikama (augmentacija slike)
    - okretanje slika horizontalno
    - okretanje slika pod 90 stupnjeva
    - nasumični isječak slike

Broj klasa je postavljen na 2 s obzirom na to da je riječ o objektnoj detekciji nošenja zaštitne maske (s maskom i bez maske).

Veličina grupe (engl. *Batch size)* definira količinu uzoraka kojom se „hrani“ model. Grupa (engl. *batch)* se može shvatiti kao petlja u kojoj se u svakoj iteraciji pojavljuje određeni broj uzoraka. Zatim se određuju predikcije i uspoređuje predikcija s očekivanim izlaznim varijablama nakon čega se računa pogreška (engl. *loss*). Ovu pogrešku algoritam koristi za poboljšanje modela. Veličina grupe može sadržavati cijeli set za treniranje, samo jednu sliku ili više od jednu, a manje od veličine seta za treniranje. Popularne veličine bi bile 16, 32 i 64 (ako to dopušta snaga grafičke kartice).

Lokacije slika i dokumenata koji sadrže oznake je potrebno postaviti kako bi model mogao pronaći navedene dokumente i pomoću njih trenirati.

Rad sa slikama je dodan kako bi se povećao broj primjeraka slika, tj. kako bi se slike nad kojima model trenira konstantno mijenjale.

Promjene na pipeline.config su vidljive na slici 30.

```
model {
  ssd {
    num_classes: 2
    image_resizer {
      fixed_shape_resizer {
        height: 640
        width: 640
      }
    }
    ...
}
train_config {
  batch_size: 8
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      ...
    }
  }
  data_augmentation_options {
    random_rgb_to_gray {
      probability: 0.3
    }
  }
  ...
  fine_tune_checkpoint: "pre-trained-models/
  ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/checkpoint/ckpt-0"
  num_steps: 25000
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "detection"
  use_bfloat16: false
  fine_tune_checkpoint_version: V2
}
train_input_reader {
  label_map_path: "annotations/label_map.pbtxt"
  tf_record_input_reader {
    input_path: "annotations/train.record"
  }
}
eval_config {
  metrics_set: "coco_detection_metrics"
  num_examples: 255
  use_moving_averages: false
  batch_size: 1;
}
eval_input_reader {
  label_map_path: "annotations/label_map.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "annotations/validation.record"
  }
}
```

**Slika 30** SSD ResNet-50 pipeline.config

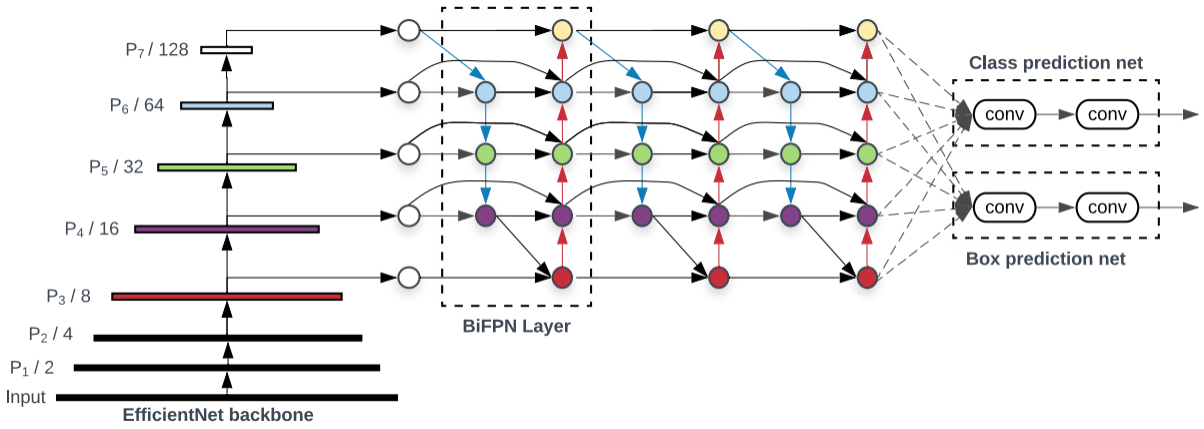
### 5.2. EfficientDet D1 640x640

Ono što je potrebno za dobar detektor za objektnu detekciju je da bude što precizniji i dovoljno brz za pokretanje u stvarnom vremenu. Ova dva aspekta imaju kompromis i većina detektora za objektnu detekciju mogu ispuniti jednu od tih stavki, bilo da je riječ o točnosti ili brzini. Kod preciznijih modela je potrebna velika snaga čime se i dolazi do smanjene brzine, što nije idealno kada je potrebno pronaći što učinkovitiji model.

Postoje određene tehnike poput globalnog obrezivanja filtera (engl. *Global Filter Pruning*) koje se koriste za komprimiranje velikih modela. Međutim, iako je moguće veće modele komprimirati navedenim postupkom, ipak je riječ o višestupanjskom (engl. *Multi stage*) postupku kod kojeg dolazi do pada točnosti.

Glavna inspiracija EfficientDet-u je EfficientNet koji je pokazao kako skalirati CNN-ove. Samim time, takva je tehnologija unaprijeđena i na detektore objekata poput EfficientDet-a za implementaciju detektora u stvarnim aplikacijama u robotici, autonomnim vozilima i slično. Uvođenjem okosnice EfficientNet moguće je postići puno bolju učinkovitost. Na primjer, počevši od početne vrijednosti RetinaNet-a koja koristi okosnicu ResNet-50, istraživanje Google tima je pokazalo da jednostavna zamjena ResNet-50 s EfficientNet-B3 može poboljšati točnost za 3%, a računanje smanjiti za 20% [16] [17].

Osim EfficientNet-a kao okosnice, EfficientDet za mrežu značajki koristi BiFPN. Slika 31 prikazuje arhitekturu EfficientDet modela.

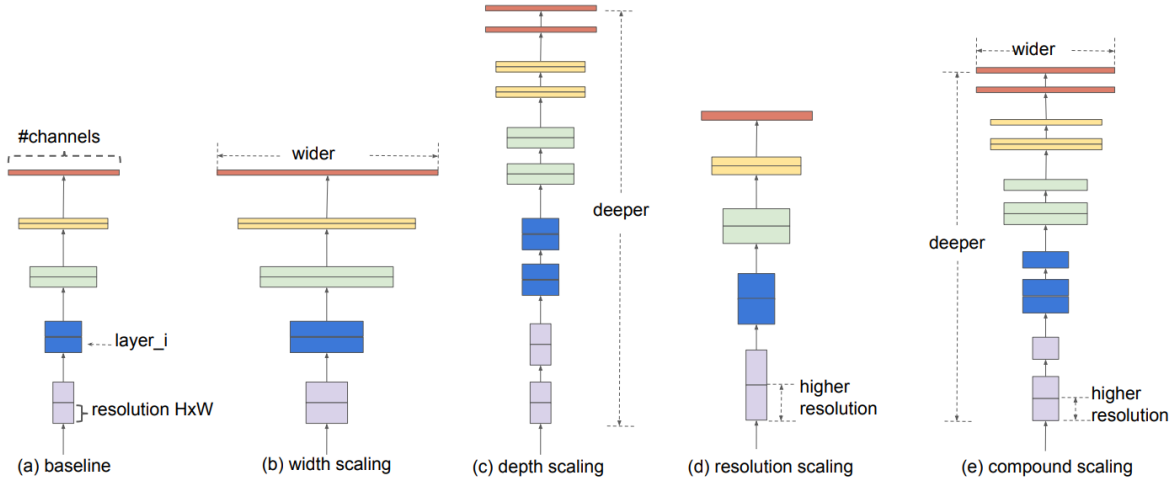



**Slika 31** Arhitektura EfficientDet-a [16]

U sljedećem poglavlju će biti objašnjena konvolucijska mreža EfficientNet.

### 5.2.1. EfficientNet

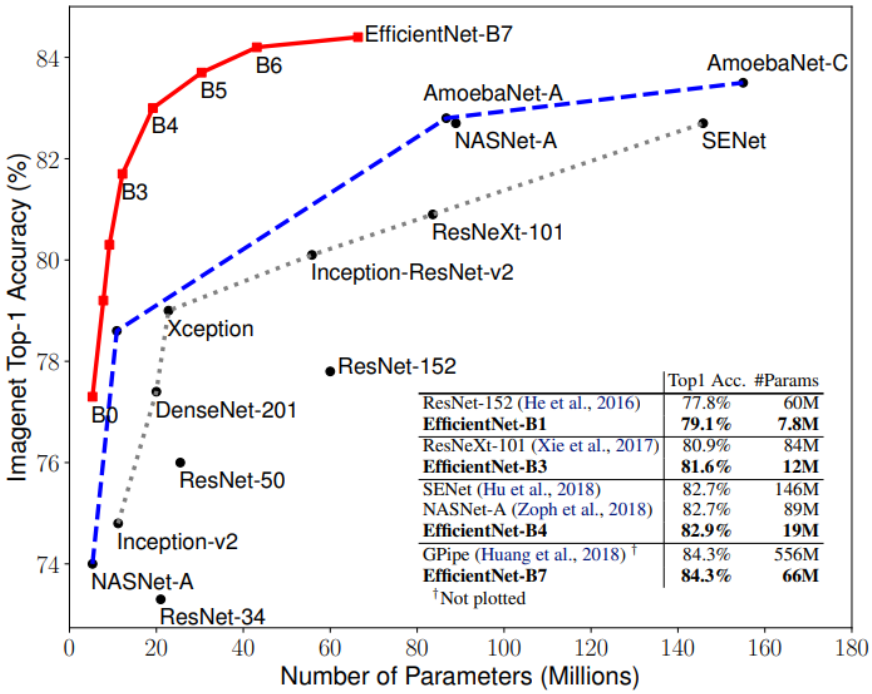
EfficientNet je konvolucijska neuronska mreža nastala od strane Google-ovih znanstvenika s ciljem poboljšanja performansi. Uobičajeni načini poboljšanja performansi konvolucijskih mreža je povećanje jednog od sljedećih parametara: širina mreže, dubina mreže i razlučivost slike. Slika 32 prikazuje različite metode skaliranja.




**Slika 32** Različite metode skaliranja konvolucijske neuronske mreže [18]

Ono što su znanstvenici iz Google-ovog tima otkrili je da je važno uravnotežiti sve dimenzije mreže (širina, dubina i razlučivost slike). Način na koji su to postigli je skaliranjem svake dimenzije konstantnim omjerom te su tu metodu nazvali složenim skaliranjem (engl. *Compound scaling*). Kako slike koje se koriste za treniranje konvolucijskih mreža postaju veće, tako složeno skaliranje počinje imati smisla jer veće slike trebaju dublje mreže kako bi povećale receptivno polje (engl. *Receptive field*) i više kanala za hvatanje manjih detalja u većoj slici.

S obzirom na to kako se učinkovitost skaliranja modela oslanja na osnovnu mrežu (engl. *Baseline network*), kako bi poboljšali performanse, autori složenog skaliranja razvili su novu osnovnu mrežu koristeći algoritam pretraživanja neuronske arhitekture (engl. *Neural architecture search*, skraćeno NAS). Korištenjem NAS algoritma optimizira se i točnost i učinkovitost (FLOPS) te je dobivena nova arhitektura EfficientNet. Slika 33 prikazuje rezultate arhitekture EfficientNet u odnosu na ostale mreže [18] [19].

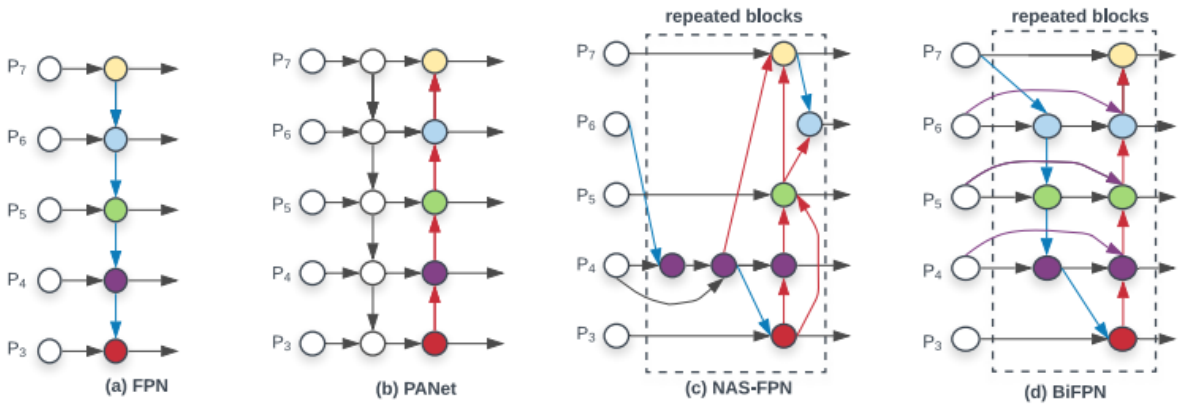



**Slika 33** EfficientNet u usporedbi s drugim konvolucijskim neuronskim mrežama [18]

### 5.2.2. BiFPN

Druga je optimizacija poboljšanje učinkovitosti mreža značajki. Većina detektora koristi piramidalnu mrežu od vrha prema dolje (engl. *Pyramidal Feature Hierarchy*, skraćeno FPN), ali takav pristup je ograničen jednosmjernim protokom informacija. FPN-ovi, kao što je PANet, dodaju dodatni protok odozdo prema gore uz nedostatak većeg računanja. Također, postoji NAS-FPN arhitektura koja je, iako učinkovita, također nepravilna i visoko optimizirana za određeni zadatak, čime je otežana prilagodba za druge zadatke.

Da bi riješili ove probleme, Google-ov tim znanstvenika je predložio novu dvosmjernu mrežu značajki, BiFPN, koja uključuje ideju višerazinskog spajanja značajki od FPN/PANet/NAS-FPN-a i omogućuje protok informacija u smjeru odozgo prema dolje i odozdo prema gore, uz korištenje redovitih i učinkovitih veza. Usporedbu mreža značajki je moguće vidjeti na slici 34.




**Slika 34** Usporedba različitih mreža značajki [16]

Dodatno, Google-ov tim je predložio brzu normaliziranu tehniku spajanja za povećanje učinkovitosti. Kod tradicionalnog pristupa sve značajke koje se unose u FPN se tretiraju jednako, čak i one s različitim rezolucijama, iako ulazne značajke na različitim rezolucijama često imaju nejednak doprinos izlaznim značajkama. Stoga se u novom pristupu dodaje dodatna težina za svaku ulaznu značajku i time omogućuje mreži da nauči važnost svake od njih.

Također, sve redovite konvolucije su zamijenjene s odvojivim dubinskim konvolucijama (engl. *Depth-wise separable convolutions*).

Uz sve navedene optimizacije BiFPN dodatno poboljšava točnost za 4%, istovremeno smanjujući troškove izračuna za 50% [16] [17].

### 5.2.3. Metoda složenog skaliranja za detektore

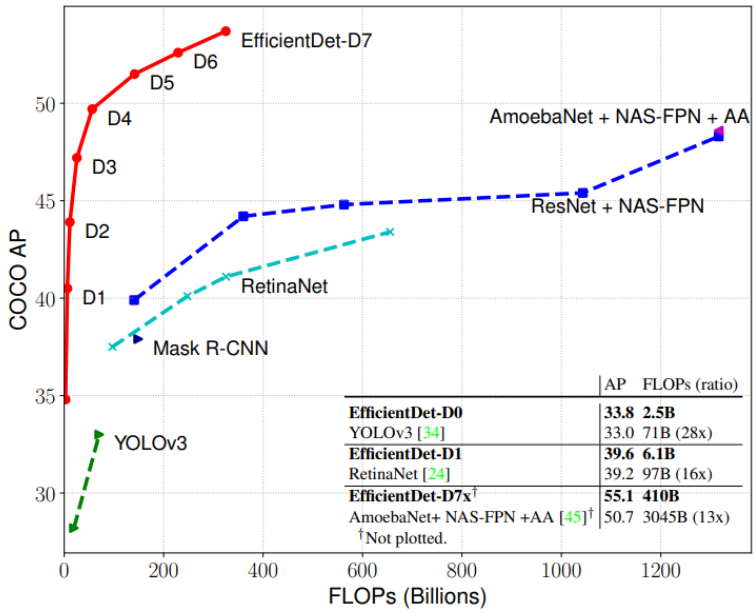
Treća optimizacija uključuje postizanje veće preciznosti i učinkovitosti kod ograničenja resursa. Kod EfficientNet-a se pokazalo da zajedničko skaliranje dubine, širine i razlučivosti mreže može značajno poboljšati učinkovitost prepoznavanja slika. Inspirirani ovom idejom, Google-ov tim znanstvenika je predložio novu metodu složenog skaliranja za detektore objekata koja zajednički povećava razlučivost, dubinu i širinu. Svaka mrežna komponenta (okosnica, značajka i mreža predviđanja okvira i klase) će imati jedan složeni faktor skaliranja koji kontrolira sve dimenzije skaliranja koristeći pravila zasnovana na heurističkim osnovama. Ovaj pristup omogućuje jednostavno određivanje načina skaliranja modela izračunavanjem faktora skaliranja za zadana ograničenja ciljanih resursa [16] [17].

### 5.2.4. EfficientDet modeli

EfficientDet se sastoji od više modela koji se kreću od EfficientDet-D0 do EfficientDet-D7 (D0, D1, D2, …, D7). Svaki naredni model ima veće računske troškove, pokrivajući širok raspon resursa od 3 milijarde FLOP-ova do 300 milijardi FLOP-ova, pružajući sve veću točnost [16] [17]. U navedenom radu korišten je D1 model s veličinom slike 640x640 piksela.

### 5.2.5. Usporedba EfficientDet-a s drugim modelima

EfficientDet je evaluiran na COCO skupu podataka. Njegov najveći model, D7, ostvario je 52.2 mAP, nadmašivši time najmoderniji model za 1.5 bodova, istovremeno koristeći 4 puta manje parametara i 9.4 puta manje računanja. Slika 35 prikazuje usporedbu EfficientDet-a s drugim modelima.




**Slika 35** EfficientDet rezultat na COCO skupu podataka s obzirom na mAP u odnosu na ostale najmodernije modele [16]

Također, Google-ov tim znanstvenika je usporedio veličinu parametara i kašnjenje kod CPU-a ili GPU-a između EfficientDet-a i ostalih modela. Pod sličnim uvjetima, modeli EfficientDet-a su 2 do 4 puta brži koristeći GPU i 5 do 11 puta brži koristeći CPU u odnosu na ostale modele.

EfficientDet, iako je namijenjen i dizajniran isključivo za objektnu detekciju, je također uspoređen s ostalim modelima i u odnosu na semantičku segmentaciju. Kako bi EfficientDet model funkcionirao nad zadacima segmentacije, Google-ov tim znanstvenika je modificirao

EfficientDet-D4 zamjenom glave za detekciju (engl. *Detection Head*) i funkcije gubitka za detekciju sa segmentacijskom glavom i funkcijom gubitka, zadržavajući istu okosnicu (EfficientNet) i BiFPN.

Kod usporedbe s do tada najboljim modelom za segmentaciju s obzirom na Pascal VOC 2012 pod nazivom DeepLabV3+ (Xception) dobio je rezultate koje prikazuje slika 36.

| Model | mIOU | Params | FLOPs |
|---|---|---|---|
| DeepLabV3+ (ResNet-101) [6] | 79.35% | - | 298B |
| DeepLabV3+ (Xception) [6] | 80.02% | - | 177B |
| **Our EfficientDet†** | **81.74%** | **17M** | **18B** |

†Modificirana verzija EfficientDet-D4

**Slika 36** EfficientDet rezultati s obzirom na segmentaciju u odnosu na ostale modele [16].

Moguće je vidjeti da je EfficientDet ostvario bolje rezultate od DeepLabV3+ modela s 9.8 manje proračuna (18B FLOP-ova u odnosu na 177B) u istim uvjetima [16] [17].

### 5.2.6. Konfiguracija EfficientDet-a

EfficientDet model je testiran u konfiguraciji u kojoj ima postavljenu veličinu slike na 640x640 i veličinu grupe na 8.

Na lokaciji ../training_demo/models/my_efficient_det nalazi se datoteka pipeline.config na kojoj je potrebno promijeniti linije kako bi se dobio prilagođen model koji radi na navedenom skupu podataka od 1700 slika.

Stavke koje su promijenjene (ili dodane) su iste kao i kod konfiguracije SSD ResNet-50 modela te se mogu vidjeti u poglavlju „5.1.3. Konfiguracija SSD ResNet-50 modela“.

Dio koda iz pipeline.config dokumenta na kojem su vidljive promjene je prikazan na slici 37.

```
model {
  ssd {
    num_classes: 2
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 640
        max_dimension: 640
        pad_to_max_dimension: true
      }
    }
    ...
}
train_config {
  batch_size: 8
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      ...
    }
  }
  data_augmentation_options {
    random_rgb_to_gray {
      probability: 0.3
    }
  }
  data_augmentation_options {
    random_scale_crop_and_pad_to_square {
      output_size: 640
      scale_min: 0.10000000149011612
      scale_max: 2.0
    }
  }
  ...
  fine_tune_checkpoint: "pre-trained-models/
  efficientdet_d1_coco17_tpu-32/checkpoint/ckpt-0"
  num_steps: 300000
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "detection"
  use_bfloat16: false
  fine_tune_checkpoint_version: V2
}
train_input_reader: {
  label_map_path: "annotations/label_map.pbtxt"
  tf_record_input_reader {
    input_path: "annotations/train.record"
  }
}

eval_config: {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
  batch_size: 1;
}

eval_input_reader: {
  label_map_path: "annotations/label_map.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "annotations/validation.record"
  }
}
```

**Slika 37** EfficientDet pipeline.config

# 6. RAD NA GOOGLE COLAB NOTEBOOK-U

Za korištenje Google Colab platforme potreban je *.ipynb notebook* koji se nalazi unutar TensorFlow mape u *notebook* podmapi pod nazivom Programski_kod.ipynb.

Nakon otvaranja, u Google Colab-u na Google Disk-u će se stvoriti mapa žute boje Colab *Notebooks* koja će sadržavati nedavno otvorene Colab *notebook*-ove.

## 6.1. Postavljanje izvršavanja

Na novootvorenom Google Colab-u potrebno je postaviti izvršavanje putem grafičke kartice (skraćeno GPU, engl. *Graphical Processing Unit*). Na alatnoj traci na vrhu ispod naziva dokumenta je potrebno kliknuti na *Runtime* pa *Change Runtime Type* i postaviti *Hardware accelerator* na GPU.

Potrebno je postaviti hardversko ubrzanje na GPU radi bržeg treniranja modela. GPU ima vlastitu memoriju kojom može istovremeno detektirati više slika te također sadrži potrebne CUDA jezgre optimizirane za strojno učenje. Nakon promjene moguće je provjeriti GPU status pomoću koda prikazanog na slici 38.

```
# Provjera GPU statusa (ako piše pronađen gpu onda je uspješno)

# potrebno prvo uvesti tensorflow
import tensorflow as tf
print(tf.__version__)

# provjera
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
   raise SystemError('GPU nije pronađen')
print('Pronađen GPU: {}'.format(device_name))
```

**Slika 38** Provjera dostupnog GPU-a

## 6.2. Dodavanje Google Disk-a

Sljedeće je potrebno dodati Google Disk. Google Disk se dodaje kodom prikazanim na slici 39.

```
# Nakon pokretanja ćelije dobit će se link.
# Kliknuti na link i kopirati kod te ga zalijepiti u formu.
from google.colab import drive
drive.mount('/content/gdrive')
```

**Slika 39** Spajanje Google Disk-a s Google Colab-om

### 6.3. Preuzimanje *TensorFlow Model Garden-a*

Zatim je potrebno preuzeti *TensorFlow Model Garden*. *TensorFlow Model Garden* je službeni GitHub repozitorij od tvrtke Google koji sadrži sve predefinirane modele koji se mogu koristiti i alate potrebne za rad samog API-a.

Navedeni opis (preveden na hrvatski) moguće je vidjeti na njihovoj stranici: "*TensorFlow Model Garden* je spremište s nizom različitih implementacija najsuvremenijih (engl. *State of the* Art, skraćeno SOTA) modela i rješenja za modeliranje za korisnike TensorFlow-a. Cilj nam je pokazati najbolje prakse za modeliranje kako bi korisnici TensorFlow-a mogli u potpunosti iskoristiti TensorFlow za svoja istraživanja i razvoj proizvoda" [20].

*TensorFlow Model Garden* se sastoji od sljedećih mapa: ***official***, ***research***, ***community*** i ***orbit***.

*Official* mapa se sastoji od zbirke primjera implementacija za SOTA (engl. *State of the ART*) modele koji koriste najnoviji TensorFlow 2 API. Službeno održavana, podržana i ažurirana s najnovijim implementacijama TensorFlow 2 API-a. Razumno je optimizirana za brze performanse, a da ju je i dalje lako pročitati.

*Research* mapa se sastoji od zbirka provedbe istraživačkih modela u programu TensorFlow (TF1 ili TF2) od istraživača. Održavaju je sami istraživači.

*Community* mapa je popis GitHub spremišta s modelima strojnog učenja i implementacijama koje pokreće TensorFlow 2.

I za kraj, *orbit* mapa je fleksibilna i lagana biblioteka koju korisnici mogu lako koristiti ili rastaviti prilikom pisanja prilagođenog koda petlje treninga u TensorFlow 2.x. Neprimjetno se integrira s tf.distribute i podržava izvođenje na različitim vrstama uređaja (CPU, GPU i TPU) [20].

*TensorFlow Model Garden* preuzima se kodom prikazanim na slici 40.

```
# prvo je potrebno prebaciti se u TensorFlow direktorij na Google Drive-u korištenjem cd komande.
%cd '/content/gdrive/My Drive/TensorFlow'
```

```
/content/gdrive/My Drive/TensorFlow
```

```
# nakon toga s git clone naredbom kopirati s githuba Model Garden u Tensorflow direktorij.
!git clone https://github.com/tensorflow/models.git
```

**Slika 40** *TensorFlow Model Garden*

### 6.4. Instalacija potrebnih alata

Za sam rad i instalaciju TensorFlow API-a za objektnu detekciju potrebni su sljedeći alati: Protobuf. python-lxml, python-pill, cython, Pandas, TF-Slim, lvis.

Protobuf (puni naziv *Protocol Buffers*, hrv. međuspremnici protokola) je jezično i platformno neutralan, proširivi mehanizam za serijalizaciju strukturiranih podataka kojeg je napravila tvrtka Google. Sličan je kao XML format, ali manji, brži i jednostavniji. Korisnik definira kako želi da podaci budu jednom strukturirani, a zatim koristi posebno generirani izvorni kod za jednostavno pisanje i čitanje tih strukturiranih podataka iz različitih tokova podataka koristeći razne jezike [21].

Lxml je biblioteka s najviše značajki jednostavne upotrebe za obradu XML-a i HTML-a preko Python programskog jezika. Pomoću alata vezuje se za C knjižnice libxml2 i libxslt. Jedinstvena je po tome što kombinira brzinu i cjelovitost XML značajki ovih knjižnica s jednostavnošću izvornog Python API-ja [22].

PIL (puni naziv *Python Imaging Library*) biblioteka dodaje mogućnosti obrade slika Python interpreteru. Ova biblioteka pruža opsežnu podršku formata raznih datoteka, učinkovitu internu reprezentaciju i prilično jake mogućnosti obrade slika. Osnovna biblioteka slika dizajnirana je za brzi pristup podacima pohranjenim u nekoliko osnovnih pikselskih formata. Pruža čvrst temelj općenitom alatu za obradu slika [23].

Cython je superset jezika Python koji dodatno podržava pozivanje C funkcija i deklariranje C vrsta na varijablama i atributima klase. To omogućuje kompajleru da generira vrlo učinkovit C kod iz Cython koda. C kod se generira jednom, a zatim kompajlira sa svim glavnim C / C ++ kompajlerima u CPythonu 2.6, 2.7 (2.4+ sa Cython 0.20.x), kao i 3.3 i svim kasnijim verzijama.

Sve ovo čini Cython idealnim jezikom za umotavanje vanjskih C knjižnica, ugradnju CPythona u postojeće programe i za brze C module koji ubrzavaju izvršavanje Python koda [24].

Pandas je brz, jak, fleksibilan i jednostavan za korištenje alat za analizu i manipulaciju podacima. Otvorenog je koda te je izgrađen u programskom jeziku Python [25].

TF-Slim je lagana knjižnica za definiranje, obuku i procjenu složenih modela u TensorFlow-u. Komponente TF-Slim mogu se slobodno miješati s izvornim TensorFlow-om, kao i s drugim okvirima. Neke značajke TF-Slim-a koje se mogu navesti su da omogućuje korisniku da kompaktno definira modele, dostupnost raznih složenih modela koji se mogu koristiti (poput modela računalnog vida), olakšava proširenje složenih modela i nastavak treniranja modela s postojećim kontrolnim točkama modela [26].

LVIS je novi skup podataka za segmentaciju instanci. Ima visokokvalitetne maske za segmentaciju instanci za preko 1000 početnih kategorija objekata na 164 000 slika. Zbog Zipfianove distribucije kategorija u prirodnim slikama, LVIS prirodno ima dugačak rep kategorija s malo uzoraka treninga. S obzirom na to da se najmodernije metode dubokog učenja za otkrivanje objekata loše izvode u režimu s malim uzorkom, ovaj skup podataka predstavlja važan i uzbudljiv novi znanstveni izazov [27].

Kod za instalaciju navedenih biblioteka prikazuje slika 41.

```
# navedene biblioteke dodajemo u Colab runtime pokretajući sljedeće dvije naredbe
!apt-get install protobuf-compiler python-lxml python-pil
!pip install Cython pandas tf-slim lvis
```

**Slika 41** Instaliranje biblioteka potrebnih za rad TensorFlow 2 API-a za objektnu detekciju

### 6.5. Pokretanje Protobuf-a

Prema zadanim postavkama, TensorFlow API za objektnu detekciju koristi prethodno instalirani Protobuf za konfiguriranje modela i parametara treninga te je potrebna ova biblioteka za dalje. Biblioteku pokrećemo s kodom prikazanim na slici 42.

```
# cd u 'TensorFlow/models/research'
%cd '/content/gdrive/My Drive/TensorFlow/models/research/'

# protobuf pokretanje
!protoc object_detection/protos/*.proto --python_out=.
```

**Slika 42** Pokretanje Google-ovog Protobuf-a

### 6.6. Dodavanje puteva mape u okruženje

Potrebno je dodati potrebne alate *TensorFlow 2 API*-a za objektnu detekciju, koji su instalirani prilikom instalacije *TensorFlow Model Garden*-a. Naredbe kojima se to odrađuje prikazuje slika 43.

```
# potrebno uvesti sys i os biblioteke kako bi se mogao dodati Python put koji će se koristiti
import os
import sys

# Dodavanje u path sljedećih direktorija: models i models/research
os.environ['PYTHONPATH']+=":/content/gdrive/My Drive/TensorFlow/models"
sys.path.append("/content/gdrive/My Drive/TensorFlow/models/research")
```

**Slika 43** Postavljanje okruženja

### 6.7. Instalacija TensorFlow 2 API-a za objektnu detekciju

Nakon postavljanja okruženja potrebno je instalirati TensorFlow API za objektnu detekciju. TensorFlow API za objektnu detekciju se instalira pokretanjem setup.py datoteke koja se nalazi unutar TensorFlow/models/research mape koja je u prethodnom koraku dodana u put radi lakšeg pristupa (bez potrebe prebacivanja na mjesto mape prije pokretanja stavki koje se nalaze u njoj).

Prvo je potrebno izgraditi (engl. *build*) sam Python program, a nakon toga ga i pokrenuti kako bi se instalirao navedeni API. Kod za instalaciju TensorFlow API-a za objektnu detekciju prikazan je na slici 44.

```
# prvo je potrebno odraditi build
!python setup.py build

# instalacija (može trajati i do 5 minuta)
!python setup.py install
```

**Slika 44** Instalacija TensorFlow API-a za objektnu detekciju

Za testiranje instalacije TensorFlow API-a za objektnu detekciju pokreće se sljedeća naredba prikazana na slici 45.

```
#cd u 'TensorFlow/models/research/object_detection/builders/'
%cd '/content/gdrive/My Drive/TensorFlow/models/research/object_detection/builders/'

# pokretanje python programa model_builder_tf2_test.py
!python model_builder_tf2_test.py

# uvoz potrebnih sučelja
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils

# ispis da je sve u redu
print('Testiranje uspješno')
```

**Slika 45** Testiranje instalacije TensorFlow 2 API-a za objektnu detekciju

Nakon pokretanja prethodnog koda dobit će se rezultat sličan slici 46 ako je instalacija bila uspješna.

```
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_invalid_first_stage_nms_iou_threshold): 0.0s
I0307 10:59:47.764241 140024335927168 test_util.py:2076] time(__main__.ModelBuilderTF2Test.test_invalid_first_stage_nms_iou_threshold): 0.0s
[       OK ] ModelBuilderTF2Test.test_invalid_first_stage_nms_iou_threshold
[ RUN      ] ModelBuilderTF2Test.test_invalid_model_config_proto
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_invalid_model_config_proto): 0.0s
I0307 10:59:47.764758 140024335927168 test_util.py:2076] time(__main__.ModelBuilderTF2Test.test_invalid_model_config_proto): 0.0s
[       OK ] ModelBuilderTF2Test.test_invalid_model_config_proto
[ RUN      ] ModelBuilderTF2Test.test_invalid_second_stage_batch_size
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_invalid_second_stage_batch_size): 0.0s
I0307 10:59:47.766239 140024335927168 test_util.py:2076] time(__main__.ModelBuilderTF2Test.test_invalid_second_stage_batch_size): 0.0s
[       OK ] ModelBuilderTF2Test.test_invalid_second_stage_batch_size
[ RUN      ] ModelBuilderTF2Test.test_session
[  SKIPPED ] ModelBuilderTF2Test.test_session
[ RUN      ] ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor): 0.0s
I0307 10:59:47.767598 140024335927168 test_util.py:2076] time(__main__.ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor): 0.0s
[       OK ] ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor
[ RUN      ] ModelBuilderTF2Test.test_unknown_meta_architecture
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_unknown_meta_architecture): 0.0s
I0307 10:59:47.768076 140024335927168 test_util.py:2076] time(__main__.ModelBuilderTF2Test.test_unknown_meta_architecture): 0.0s
[       OK ] ModelBuilderTF2Test.test_unknown_meta_architecture
[ RUN      ] ModelBuilderTF2Test.test_unknown_ssd_feature_extractor
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_unknown_ssd_feature_extractor): 0.0s
I0307 10:59:47.769044 140024335927168 test_util.py:2076] time(__main__.ModelBuilderTF2Test.test_unknown_ssd_feature_extractor): 0.0s
[       OK ] ModelBuilderTF2Test.test_unknown_ssd_feature_extractor
----------------------------------------------------------------------
Ran 20 tests in 24.667s

OK (skipped=1)
Done
```

**Slika 46** Rezultat testiranja instalacije TF2 API-a za objektnu detekciju

### 6.8. Pretvaranje XML dokumenata u .tfrecord format

Nakon instalacije TensorFlow 2 API-a za objektnu detekciju potrebno je prije same izrade vlastitog modela i treniranja pretvoriti oznake slika u XML formatu u .tfrecord format. Za pretvaranje će se koristiti Python program generate_tfrecord.py koji se nalazi u *preprocessing* mapi unutar TensorFlow mape. Kod potreban za pretvaranje prikazan je na slici 47.

```
# Potrebno je cd u preprocessing direktorij radi pristupa python programu generate_tfrecord.py
%cd '/content/gdrive/My Drive/TensorFlow/scripts/preprocessing'

# Stvaranje train.record
!python generate_tfrecord.py -x '/content/gdrive/My Drive/TensorFlow/workspace/training_demo/images/train' \
-l '/content/gdrive/My Drive/TensorFlow/workspace/training_demo/annotations/label_map.pbtxt' \
-o '/content/gdrive/My Drive/TensorFlow/workspace/training_demo/annotations/train.record'

# Stvaranje test.record
!python generate_tfrecord.py -x '/content/gdrive/My Drive/TensorFlow/workspace/training_demo/images/test' \
-l '/content/gdrive/My Drive/TensorFlow/workspace/training_demo/annotations/label_map.pbtxt' \
-o '/content/gdrive/My Drive/TensorFlow/workspace/training_demo/annotations/test.record'

# Stvaranje validation.record
!python generate_tfrecord.py -x '/content/gdrive/My Drive/TensorFlow/workspace/training_demo/images/validation' \
-l '/content/gdrive/My Drive/TensorFlow/workspace/training_demo/annotations/label_map.pbtxt' \
-o '/content/gdrive/My Drive/TensorFlow/workspace/training_demo/annotations/validation.record'
```

**Slika 47** Pretvaranje oznaka slika iz XML u .tfrecord format

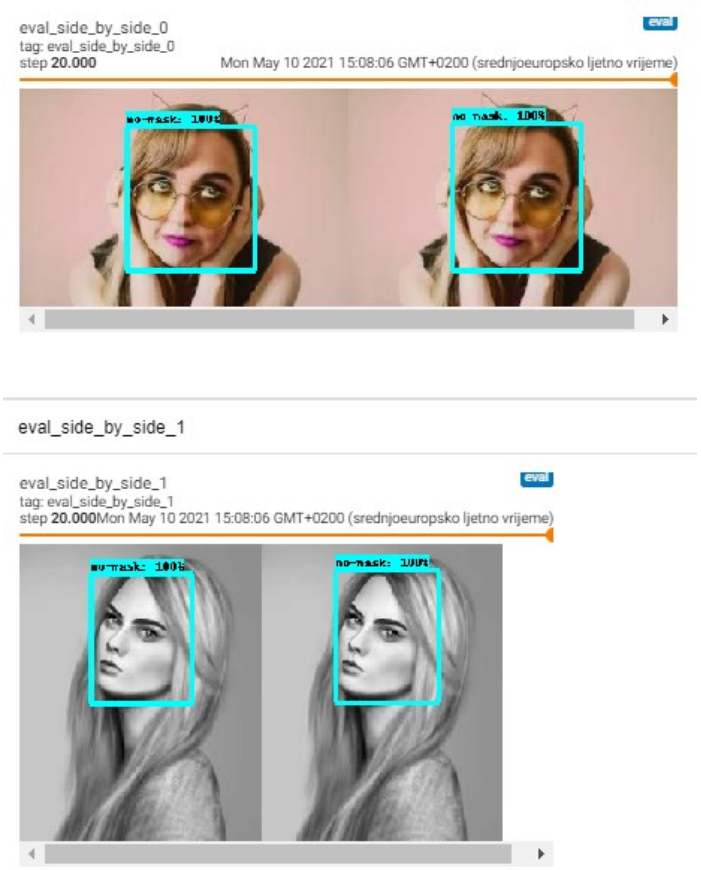
### 6.9. Treniranje modela

Nakon svega navedenog (instalacija TF2 API-a za objektnu detekciju, označavanje slika, pravljenje modela) potrebno je trenirati model nad vlastitim skupom podataka.

Za praćenje napretka koristi se TensorBoard koji služi za praćenje grafova tijekom treniranja modela. Moguće je pratiti grafove, usporedbu evaluacije s početnom predikcijom i slično.

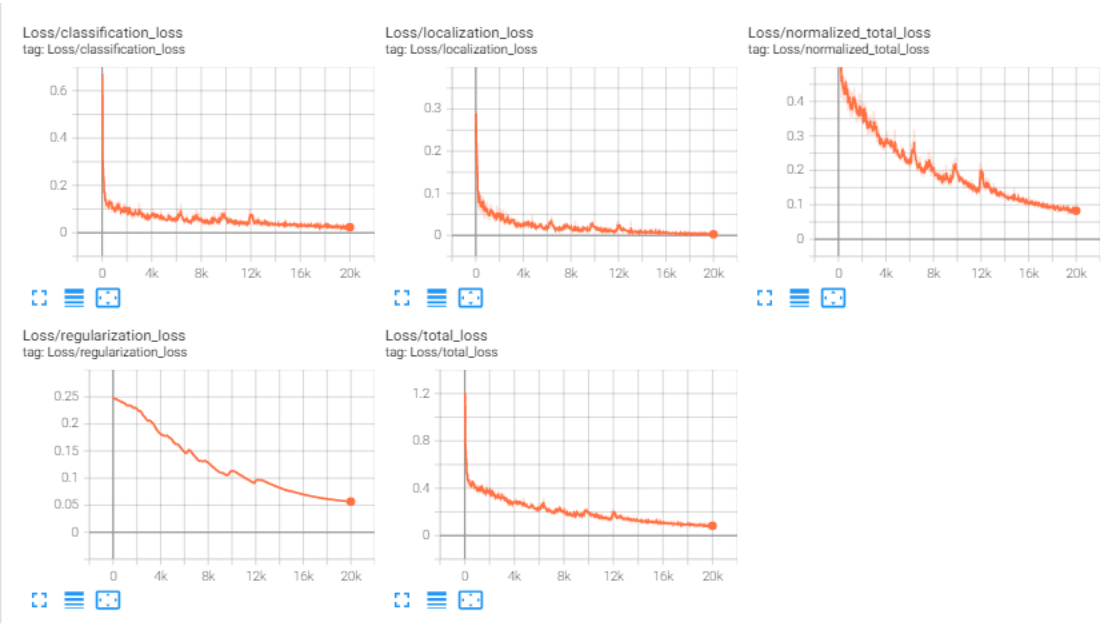
Neki primjeri TensorBoard-a dobiveni na navedenom skupu podataka i modelu napravljenom na njegovim karakteristikama vidljivi su na sljedećim slikama.

Slika 48 prikazuje evaluaciju modela nad testnim slikama (lijevo predikcija modela, desno očekivana predikcija).




**Slika 48** Usporedba evaluacije slike (lijevo) i već označene predikcije (desno)

Slika 49 prikazuje grafove funkcija gubitka. U pravilu, što više trenira model to bi funkcija gubitka trebala padati što je vidljivo na sljedećim grafovima.




**Slika 49** Grafovi funkcija gubitka

Treniranje modela se pokreće kodom prikazanim na slici 50.

```
%cd '/content/gdrive/My Drive/TensorFlow/workspace/training_demo'
#Treniranje modela
broj_koraka = 20000

# Linija za pokretanje treniranja na modelu my_ssd_resnet50_v1_fpn
!python model_main_tf2.py \
--model_dir=models/my_ssd_resnet50_v1_fpn \
--pipeline_config_path=models/my_ssd_resnet50_v1_fpn/pipeline.config \
--num_train_steps={broj_koraka} \
--alsologtostderr
```

**Slika 50** Kod za pokretanje treniranja modela

Prilikom treniranja prikazuju se vrijednosti funkcije gubitka. Radi testiranja modela svaki model je postavljen na 20 000 koraka.

### 6.10. Evaluacija modela

Po završetku treniranja je potrebno odraditi evaluaciju modela. Kod evaluacije testira se model u odnosu na slike koje se nalaze na ../img/validation. Kod koji pokreće evaluaciju je prikazan na slici 51.

```
!python model_main_tf2.py \
    --pipeline_config_path=models/my_ssd_resnet50_v1_fpn/pipeline.config \
    --model_dir=models/my_ssd_resnet50_v1_fpn \
    --checkpoint_dir=models/my_ssd_resnet50_v1_fpn
```

**Slika 51** Kod za pokretanje evaluacije modela

Ako je evaluacija uspješna dobit će se podaci za promatrani model i skup podataka prikazani na slici 52.

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.680
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.944
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.827
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.449
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.690
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.751
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.537
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.716
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.738
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.572
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.743
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.801
INFO:tensorflow:Eval metrics at step 20000
I0519 15:14:18.448198 140588880177024 model_lib_v2.py:853] Eval metrics at step 20000
INFO:tensorflow:        + DetectionBoxes_Precision/mAP: 0.679531
I0519 15:14:18.454787 140588880177024 model_lib_v2.py:856]      + DetectionBoxes_Precision/mAP: 0.679531
INFO:tensorflow:        + DetectionBoxes_Precision/mAP@.50IOU: 0.943544
I0519 15:14:18.456215 140588880177024 model_lib_v2.py:856]      + DetectionBoxes_Precision/mAP@.50IOU: 0.943544
INFO:tensorflow:        + DetectionBoxes_Precision/mAP@.75IOU: 0.826752
I0519 15:14:18.457675 140588880177024 model_lib_v2.py:856]      + DetectionBoxes_Precision/mAP@.75IOU: 0.826752
INFO:tensorflow:        + DetectionBoxes_Precision/mAP (small): 0.449468
I0519 15:14:18.459097 140588880177024 model_lib_v2.py:856]      + DetectionBoxes_Precision/mAP (small): 0.449468
INFO:tensorflow:        + DetectionBoxes_Precision/mAP (medium): 0.689843
I0519 15:14:18.460620 140588880177024 model_lib_v2.py:856]      + DetectionBoxes_Precision/mAP (medium): 0.689843
INFO:tensorflow:        + DetectionBoxes_Precision/mAP (large): 0.750982
I0519 15:14:18.462059 140588880177024 model_lib_v2.py:856]      + DetectionBoxes_Precision/mAP (large): 0.750982
INFO:tensorflow:        + DetectionBoxes_Recall/AR@1: 0.536646
I0519 15:14:18.463528 140588880177024 model_lib_v2.py:856]      + DetectionBoxes_Recall/AR@1: 0.536646
INFO:tensorflow:        + DetectionBoxes_Recall/AR@10: 0.716497
I0519 15:14:18.464964 140588880177024 model_lib_v2.py:856]      + DetectionBoxes_Recall/AR@10: 0.716497
INFO:tensorflow:        + DetectionBoxes_Recall/AR@100: 0.737681
I0519 15:14:18.466408 140588880177024 model_lib_v2.py:856]      + DetectionBoxes_Recall/AR@100: 0.737681
INFO:tensorflow:        + DetectionBoxes_Recall/AR@100 (small): 0.572449
I0519 15:14:18.467929 140588880177024 model_lib_v2.py:856]      + DetectionBoxes_Recall/AR@100 (small): 0.572449
INFO:tensorflow:        + DetectionBoxes_Recall/AR@100 (medium): 0.742568
I0519 15:14:18.469326 140588880177024 model_lib_v2.py:856]      + DetectionBoxes_Recall/AR@100 (medium): 0.742568
INFO:tensorflow:        + DetectionBoxes_Recall/AR@100 (large): 0.801251
I0519 15:14:18.470885 140588880177024 model_lib_v2.py:856]      + DetectionBoxes_Recall/AR@100 (large): 0.801251
INFO:tensorflow:        + Loss/localization_loss: 0.073586
I0519 15:14:18.472042 140588880177024 model_lib_v2.py:856]      + Loss/localization_loss: 0.073586
INFO:tensorflow:        + Loss/classification_loss: 0.323108
I0519 15:14:18.473272 140588880177024 model_lib_v2.py:856]      + Loss/classification_loss: 0.323108
INFO:tensorflow:        + Loss/regularization_loss: 0.056590
I0519 15:14:18.474476 140588880177024 model_lib_v2.py:856]      + Loss/regularization_loss: 0.056590
INFO:tensorflow:        + Loss/total_loss: 0.453284
I0519 15:14:18.475763 140588880177024 model_lib_v2.py:856]      + Loss/total_loss: 0.453284
```

**Slika 52** Rezultat evaluacije modela

### 6.11. Izvoz modela

Kad je faza treniranja i evaluacije modela dovršena potrebno je izvesti (engl. *export*) dobiveni model. Model se može izvesti naredbom prikazanom na slici 53.

```
# Linija za izvoz na lokaciju ./exported-models/my_model
!python exporter_main_v2.py \
--input_type image_tensor \
--pipeline_config_path ./models/my_ssd_resnet50_v1_fpn/pipeline.config \
--trained_checkpoint_dir ./models/my_ssd_resnet50_v1_fpn/ \
--output_directory ./exported-models/my_ssd_resnet_640x640
```

**Slika 53** Kod za izvoz modela

Nakon izvoza modela, model će biti spremljen na lokaciji TensorFlow/workspace/training_demo/exported_models/my_model.

### 6.12. Preuzimanje treniranog modela

Ako je potrebno preuzeti model direktno iz Google Colab-a to se može učiniti pokretanjem naredbi prikazanima na slici 54. Prva naredba obavlja komprimiranje mape modela dok se drugom naredbom preuzima model.

```
!zip -r /content/my_ssd_resnet50_v1_fpn_640x640.zip ./exported-models/my_ssd_resnet50_v1_fpn_640x640
```

```
from google.colab import files
files.download("/content/my_efficientdet.zip")
```

**Slika 54** Naredbe za preuzimanje treniranog modela

### 6.13. Testiranje modela

Također, moguće je testiranje modela na slikama na Google Colab-u.

Prvo je potrebno učitati model. Odsječak koda koji obavlja učitavanje modela prikazan je na slici 55.

```
# Potrebne biblioteke
import tensorflow as tf
import time
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils

# mjesto spremljenog modela.
lokacija_modela="/content/gdrive/My Drive/TensorFlow/workspace/training_demo/exported-models/my_model/saved_model"

print('Učitavanje modela ... ', end='')

# učitavanje modela
detect_fn=tf.saved_model.load(lokacija_modela)

print('Učitavanje završeno!')
```

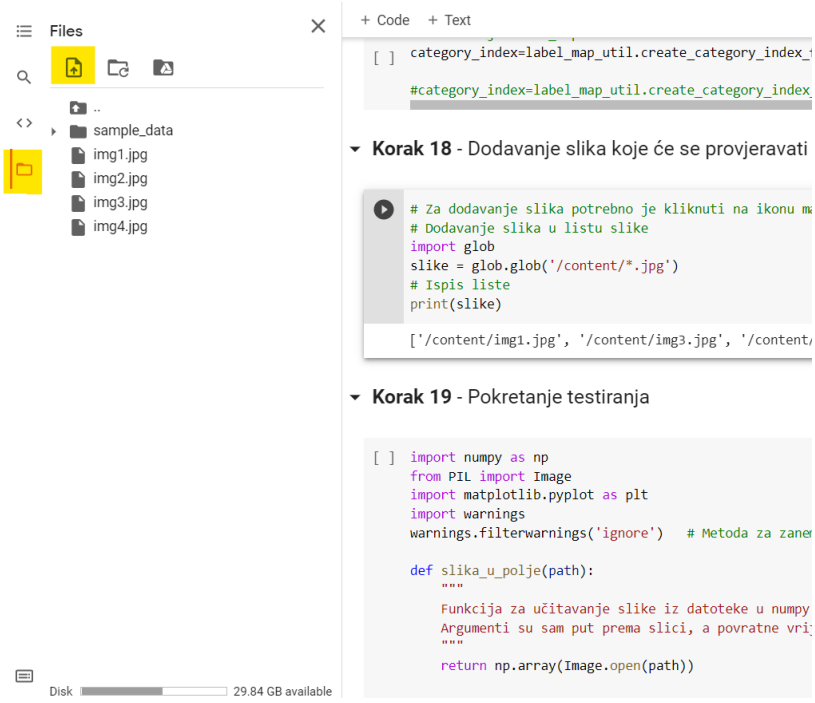
**Slika 55** Učitavanje modela

Nakon učitavanja modela slijedi učitavanje mape oznaka koja sadrži nazive klasa za promatrani skup podataka (with-mask, no-mask). Kod za učitavanje mape oznaka prikazan je na slici 56.

```
# Potrebne biblioteke
import tensorflow as tf
import time
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils

# mjesto spremljenog modela.
lokacija_modela="/content/gdrive/My Drive/TensorFlow/workspace/training_demo/exported-models/my_model/saved_model"

print('Učitavanje modela ... ', end='')

# učitavanje modela
detect_fn=tf.saved_model.load(lokacija_modela)

print('Učitavanje završeno!')
```

**Slika 56** Učitavanje mape oznaka

Zatim je potrebno dodati slike na kojima će se model testirati. Slike u Google Colab-u moguće je dodati klikom na ikonu mape koja se nalazi na lijevoj alatnoj traci te zatim pritiskom na tipku za dodavanje slika. Slike moraju biti u .jpg formatu. Slika 57 prikazuje navedeni postupak.




**Slika 57** Dodavanje slika za testiranje u Google Colab

Slike se zatim dodaju u listu potrebnu za testiranje korištenjem naredbe prikazane na slici 58.

```
# Za dodavanje slika potrebno je kliknuti na ikonu mape lijevo na google Colabu i dodati dvije ili više slika.
# Dodavanje slika u listu slike
import glob
slike = glob.glob('/content/*.jpg')
# Ispis liste
print(slike)
```

**Slika 58** Dodavanje slika u listu za testiranje

Nakon dodavanja slika moguće je testiranje pokretanjem odjeljka koda koji prikazuje slika 59.

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')   # Metoda za zanemarivanje upozorenja Matplotliba

def slika_u_polje(path):
    """
    Funkcija za učitavanje slike iz datoteke u numpy polje kako bi se mogla prikazivati u tensorflow grafu.
    Argumenti su sam put prema slici, a povratne vrijednosti numpy polje,
    """
    return np.array(Image.open(path))

for slika in slike:

    print('Detekcija za sliku {} ... '.format(slika), end='')
    slika_np = slika_u_polje(slika)

    # Ulaz mora biti tensor te ga je potrebno promijeniti s metodom `tf.convert_to_tensor`.
    input_tensor=tf.convert_to_tensor(slika_np)

    # Model očekuje više slika pa je potrebno dodati os sa metodom `tf.newaxis`
    input_tensor=input_tensor[tf.newaxis, ...]
    detections=detect_fn(input_tensor)

    # Izlazi se sastoje od n broja detekcija (u obliku batch tensora)
    # Potrebno promijeniti u numpy polje i maknuti index [0] da se makne dimenzija
    # Potrebno nam je samo prvih n detekcija
    num_detections=int(detections.pop('num_detections'))
    detections={key:value[0,:num_detections].numpy()

    # Pretvorba klasa u prirodne brojeve (int)
    detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

    slika_np_sa_detekcijama = slika_np.copy()

    viz_utils.visualize_boxes_and_labels_on_image_array(
          slika_np_sa_detekcijama,
          detections['detection_boxes'],
          detections['detection_classes'],
          detections['detection_scores'],
          category_index,
          use_normalized_coordinates=True,
          max_boxes_to_draw = 100,     # maksimalni broj kvadrata u slici
          min_score_thresh = .5,      # minimalna potrebna razina predikcije da bi se stavila na sliku
          agnostic_mode=False)
    %matplotlib inline
    plt.figure()
    plt.imshow(slika_np_sa_detekcijama)
    print('Detekcija završena')
    plt.show()
```

**Slika 59** Odjeljak koda za testiranje modela na slikama

Nakon pokretanja testiranja nad dodanim slikama, dobiveni su sljedeći rezultati u obliku Matplotlib grafova koji su prikazani na slici 60.


**Slika 60** Rezultati testiranja modela nad dodanim slikama (plavi okvir = bez maske, zeleni okvir = s maskom)

# 7. USPOREDBA REZULTATA

Nakon objašnjenja tematike i prikaza svih koraka potrebnih za treniranje modela, od skupa podataka do potrebnog programskog koda, potrebno je prikazati rezultate. Svaki model je treniran na 20 000 koraka. Modeli su trenirani korištenjem Google Colab grafičkih kartica. Svaka osoba na Google Colab-u dijeli grafičke kartice te im se dodjeljuje određeni broj resursa grafičke kartice kao i RAM-a (engl. *Random Access Memory*) te zbog nekonzistentnosti samih dijeljenih grafičkih kartica nije mjereno vrijeme potrebno da se treniranje završi. Odrađeno je treniranje i evaluacija modela te su dobiveni rezultati prikazani u sljedećim potpoglavljima. Korišteni rezultati uključuju mjerenje metrika nad funkcijom gubitka, srednjom prosječnom preciznosti (engl. *mean average precision,* skraćeno mAP) i srednjeg prosječnog odziva (engl. *mean* a*verage recall,* skraćeno mAR). Navedene metrike su objašnjene u poglavljima 2.3. „Funkcija gubitka (engl. Loss function)“ i 2.4. „Srednja prosječna preciznost (engl. mean average precision, skraćeno mAP) i srednji prosječni odziv (engl. mean average recall, skraćeno mAR)“.

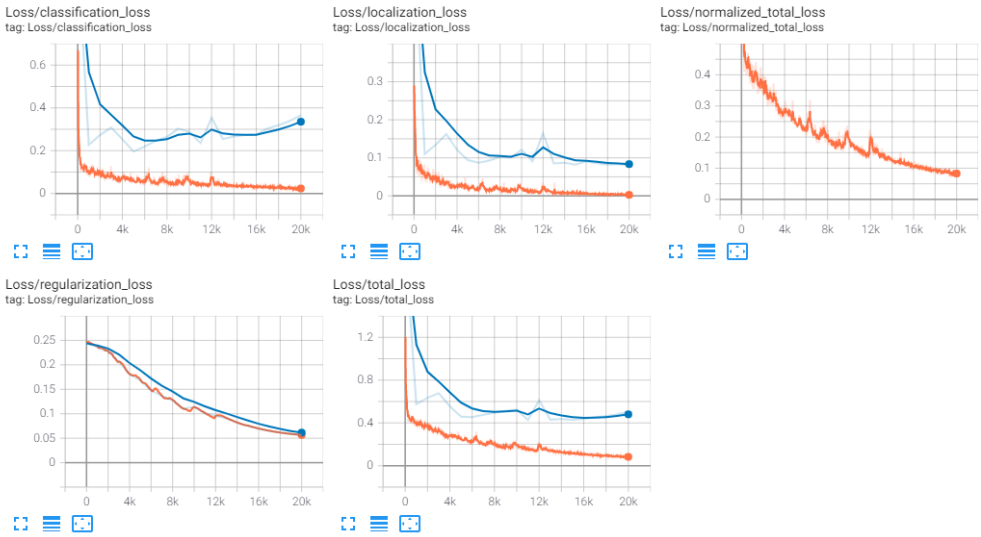
## 7.1. SSD ResNet 50

SSD ResNet 50 model je treniran u dvije konfiguracije kao što je navedeno u radu. Radi jednostavnosti, navedene modele će se nazivati SSD ResNet 50 320x320 i SSD ResNet 50 640x640.

### 7.1.1. SSD ResNet 50 320x320

Grafovi na sljedećim slikama prikazuju rezultate dobivene za SSD ResNet 50 320x320 model tijekom treniranja. Model je konfiguriran s veličinom slike dimenzija 320x320 piksela i veličinom skupa koja iznosi 32.

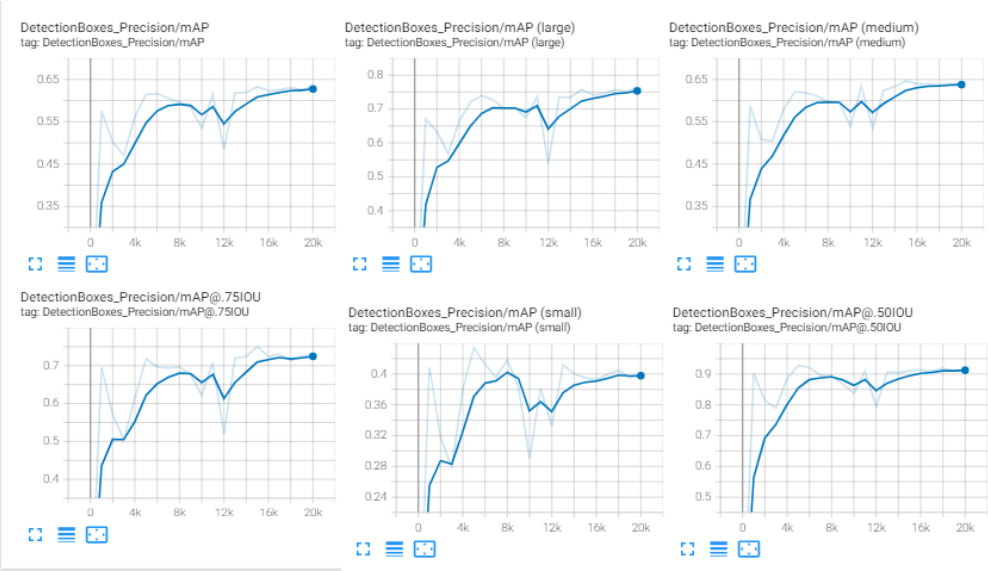
Slika 61 prikazuje grafove funkcije gubitka (engl. *loss function*) kod skupa podataka za treniranje i evaluaciju. Funkcija gubitka je metoda koja procjenjuje koliko dobro specifični algoritam modelira dane podatke, tj. koliko odstupa od danih rezultata.




**Slika 61** SSD ResNet 320x320, grafovi funkcija gubitka

Ono što je moguće vidjeti je porast funkcije gubitka za klasifikaciju na validacijskom skupu podataka kod evaluacije. Porast se pojavljuje nakon 16 000 koraka i predstavlja oblik *overfitting-a*. Razlog tome je mali skup podataka od 1700 slika s puno varijacija čovjeka kod kojeg su neke slike preteške za klasificirati (npr. obje ruke prekrivaju lice). Neoznačene slike će biti prikazane kod potpoglavlja „7.3 Neoznačene slike kod validacijskog skupa“.

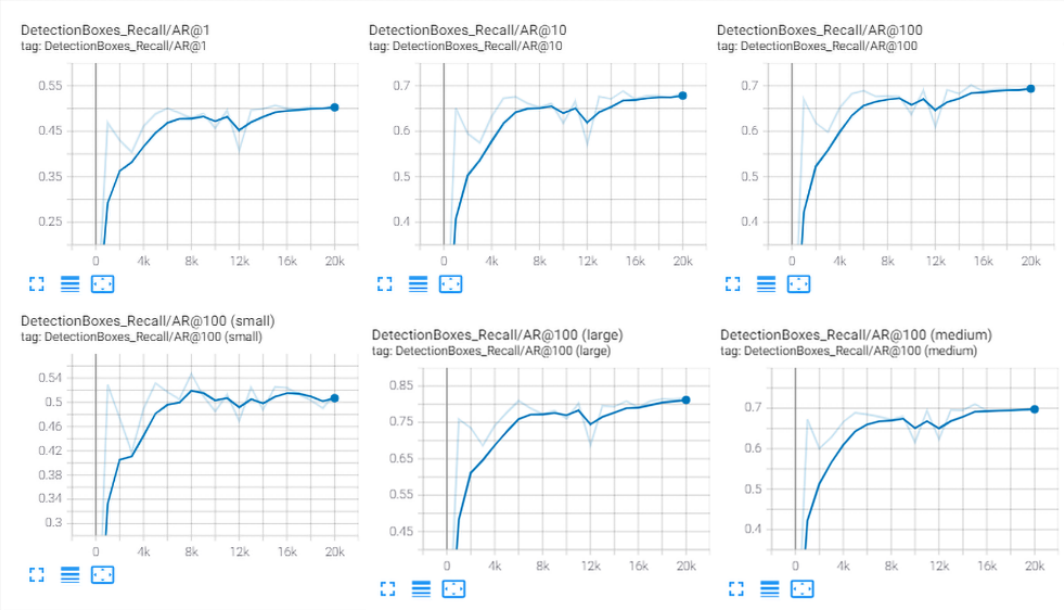
Slika 62 prikazuje grafove za metriku srednje prosječne preciznosti (skraćeno mAP). mAP predstavlja procjenu točnosti otkrivanja objekata (klasifikacija i lokalizacija). To bi predstavljalo podatak koji govori koliko je bilo točno detekcija koje je model učinio nad slikama na navedenom skupu podataka. Tako, na primjer, ako na slici model označi četiri objekta kao pozitivna, mAP vraća koliko je od detektiranih objekata, model točno detektirao. Kao prag točnosti metrika koristi IoU. Pri manjoj IoU vrijednosti model može vratiti više detekcija uz manji postotak sigurnosti. Zbog toga mAP uzima vrijednosti preciznosti nad više IoU vrijednosti i vraća srednju (engl. *mean*) vrijednost. Više o metrici mAP moguće je pročitati u poglavlju 2.4. „Srednja prosječna preciznost (engl. *mean average precision*, skraćeno mAP) i srednji prosječni odziv (engl. *mean average recall*)“.




**Slika 62** SSD ResNet 320x320, grafovi za mAP

Kod grafova za metriku mAP vidi se rast preciznosti za validacijski skup podataka. To bi u ovom slučaju predstavljalo da model sa ≈ 65% srednjom prosječnom preciznošću, gledajući prosjek svih prosječnih vrijednosti nad raznim IoU vrijednostima, lokalizacijski i klasifikacijski točno detektira objekte.

Sljedeća slika prikazuje grafove za metriku srednjeg prosječnog odziva (engl. *mean average recall*, skraćeno mAR). Iako na grafovima piše AR, riječ je ipak o mAR jer se računa prosjek nad više klasa (u ovom slučaju dvije). Metrika odziva (engl. *recall*) mjeri koliko je model dobar u pogađanju točne klase, tj. koliko je algoritam točno pogodio pozitivnih klasa označenih prije treniranja gdje se kod računanja postotka gleda i klasifikacija i lokalizacija. Više o metrikama odziva, AR i mAR moguće je pročitati u poglavlju 2.4. „Srednja prosječna preciznost (engl. *mean average precision*, skraćeno mAP) i srednji prosječni odziv (engl. *mean average recall,* skračeno mAR)“.



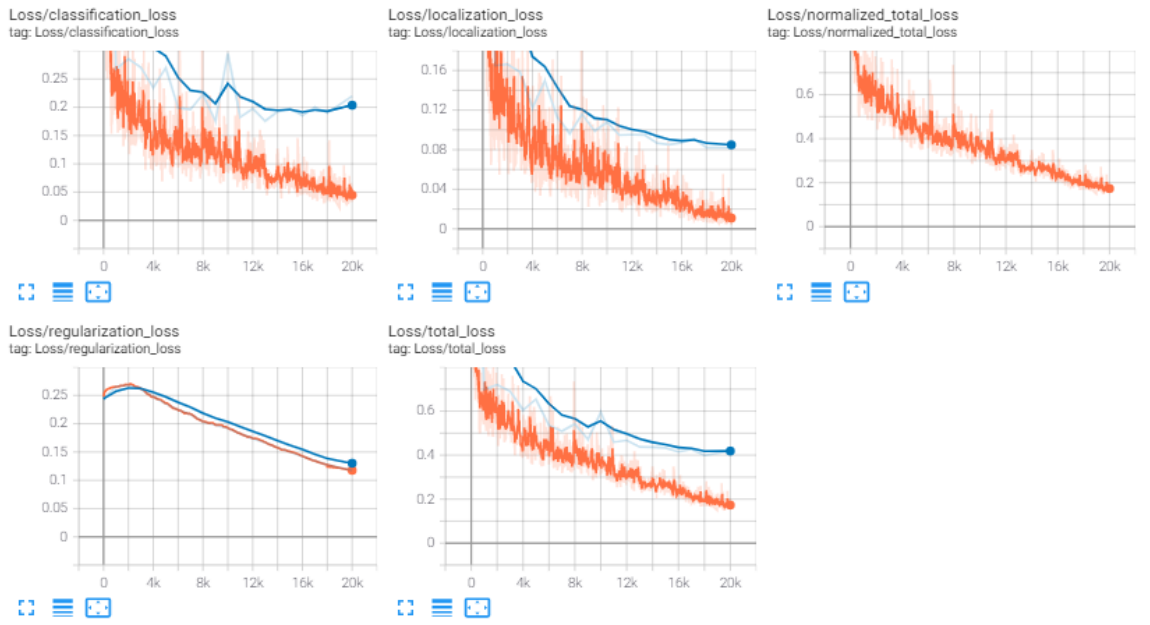

**Slika 63** SSD ResNet 320x320, grafovi za AR

Jednako kao i kod metrike mAP vidi se rast na promatranom validacijskom skupu podataka. U ovom slučaju bi za graf koji prikazuje podatke za AR1 predstavljalo da model sa ≈ 50% prosječnim odzivom lokalizacijski i klasifikacijski točno detektira objekte.

### 7.1.2. SSD ResNet 50 640x640

Grafovi na sljedećim slikama prikazuju rezultate dobivene za SSD ResNet 50 640x640 model tijekom treniranja. Model je konfiguriran s veličinom slike 640x640 i veličinom skupa koja iznosi 8.

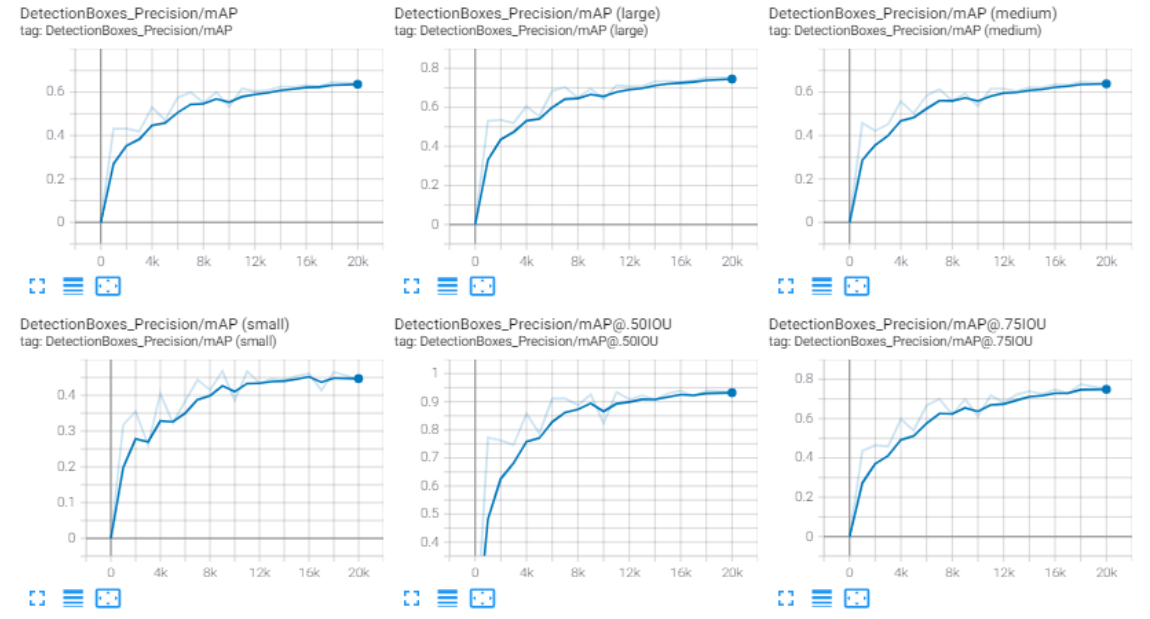
Slika 64 prikazuje graf funkcije gubitka (engl. *loss function*) za lokalizaciju kod skupa podataka za treniranje i evaluaciju.




**Slika 64** SSD ResNet 640x640, grafovi funkcija gubitka

Jednako kao i kod prethodnog modela (SSD ResNet 50 320x320), vidljiv je porast funkcije gubitka nakon 16 000 koraka na promatranom validacijskom skupu podataka.

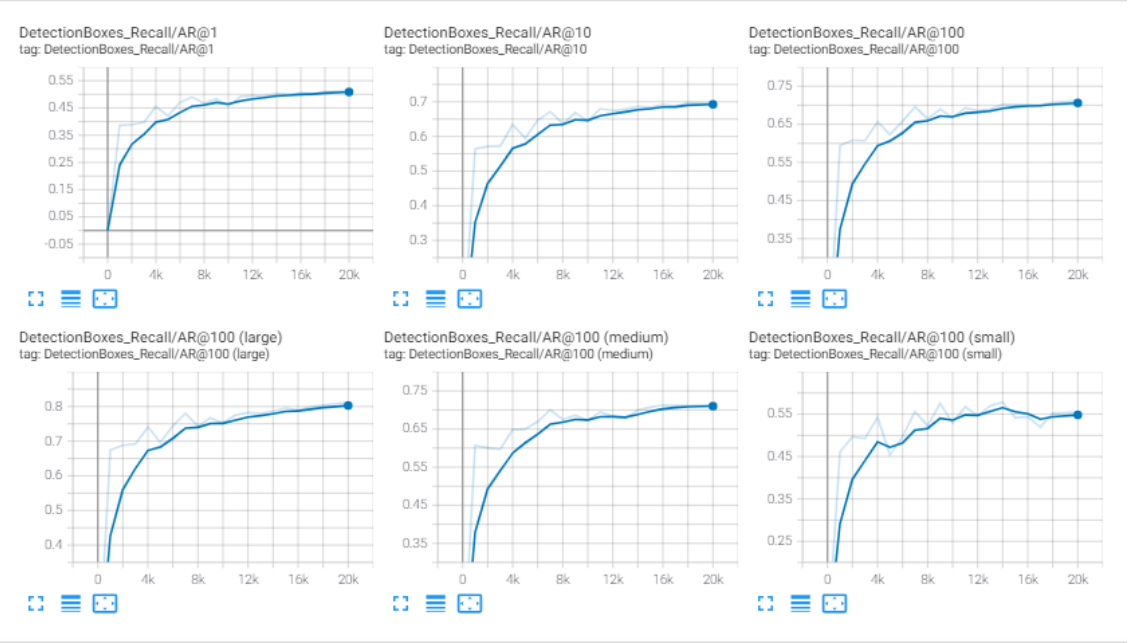
Sljedeća slika prikazuje grafove za metriku mAP.




**Slika 65** SSD ResNet 640x640, grafovi za metriku mAP

Vidljiv je rast navedene metrike za promatrani skup podataka.

Na sljedećoj slici moguće je vidjeti grafove za metriku AR.



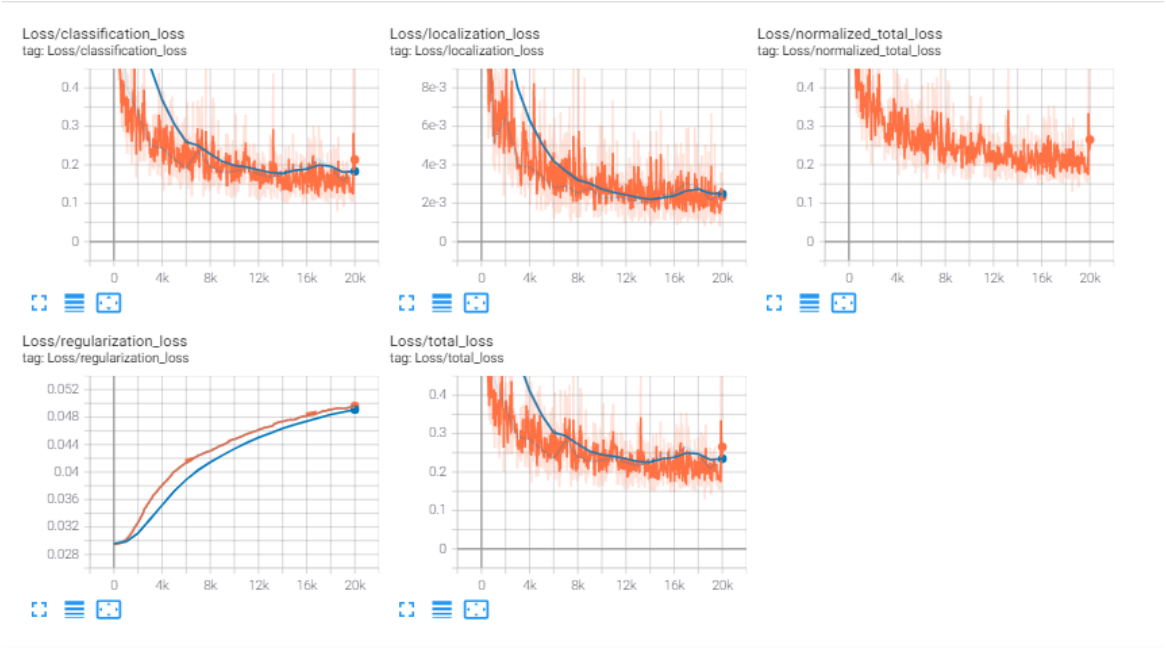

**Slika 66** SSD ResNet 640x640, grafovi za metriku AR

Jednako kao i kod metrike mAP, vidljiv je rast funkcija za metriku AR.

### 7.2. EfficientDet 640x640

Grafovi na sljedećim slikama prikazuju rezultate dobivene za EfficientDet model tijekom treniranja. Model je konfiguriran s veličinom slike 640x640 i veličinom skupa koja iznosi 8.

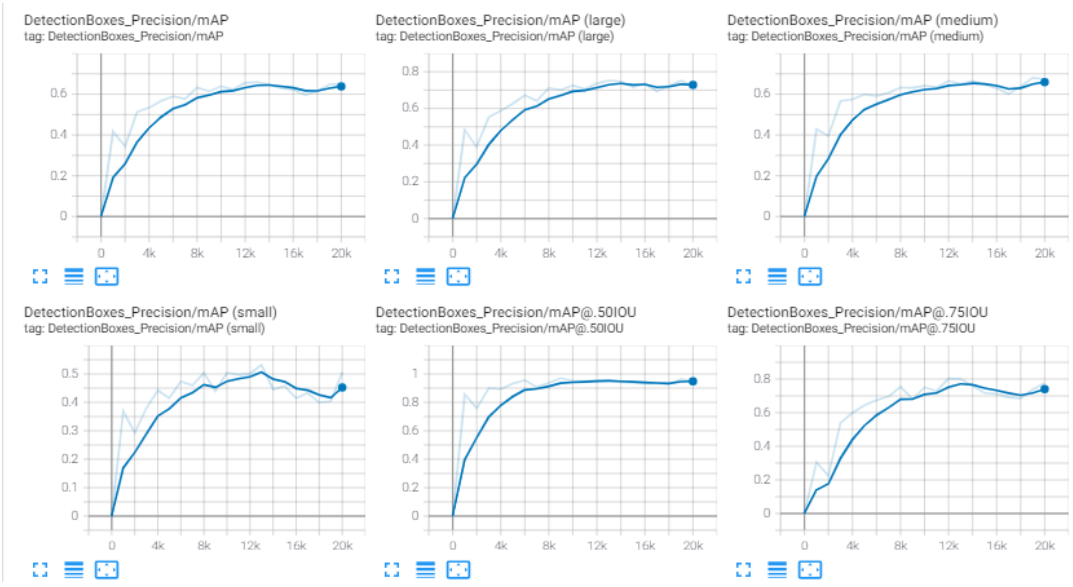
Slika 67 prikazuje grafove funkcija gubitka (engl. *loss function*) kod skupa podataka za treniranje i evaluaciju.




**Slika 67** EfficientDet 640x640, grafovi funkcija gubitka

Vidljiv je pad na većini funkcija gubitka osim na funkciji gubitka za regularizaciju koja ima rast na promatranim skupovima podataka za trening i validaciju. Međutim, riječ je o jako malim vrijednostima koje su manje u odnosu na prethodna dva modela.

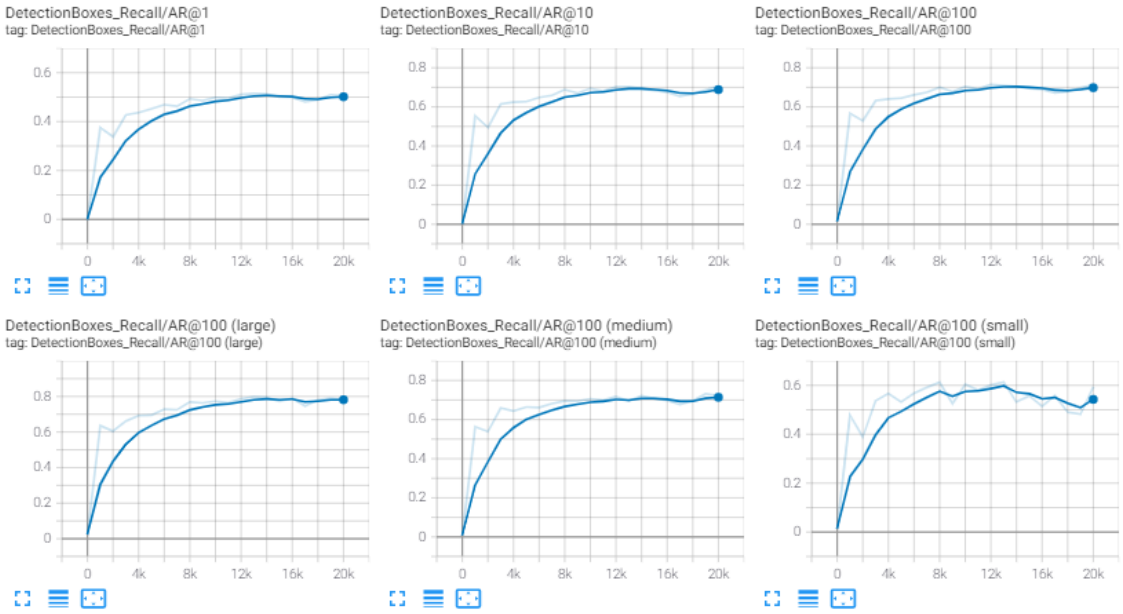
Sljedeća slika prikazuje grafove za metriku mAP kod skupa podataka za treniranje i evaluaciju.




**Slika 68** EfficientDet 640x640, grafovi za metriku mAP

Vidljiv je porast navedenih grafova uz određene oscilacije nakon 16 000 koraka što ukazuje na *overfitting*.

Slika 69 prikazuje grafove za metriku AR kod skupa podataka za treniranje i evaluaciju.




**Slika 69** EfficientDet 640x640, grafovi za metriku AR

Za navedene grafove vidljiv je rast do 14 000 koraka nakon kojeg slijedi stagniranje rasta.

Time se može zaključiti nakon svih parametara da bi za navedeni model posljednji optimalni korak treniranja bio 14 000, dok bi za prethodne modele optimalni posljednji korak treniranja bio do 16 000 koraka.

### 7.3. Neoznačene slike kod validacijskog skupa

Za provjeru slika koje stvaraju probleme detekcije korišteni su rezultati nad EfficientDet modelom.

Ono što je moguće vidjeti kod rezultata treniranja modela i prikaza evaluacije je to da je model imao probleme evaluacije kod određenih slika. Te slike su sadržavale osobe gdje bi im ruka prekrivala veći dio lica i osobe okrenute bočno. Neoznačene slike prikazuje sljedeća slika.

**Slika 70** Neoznačene slike kod validacijskog skupa

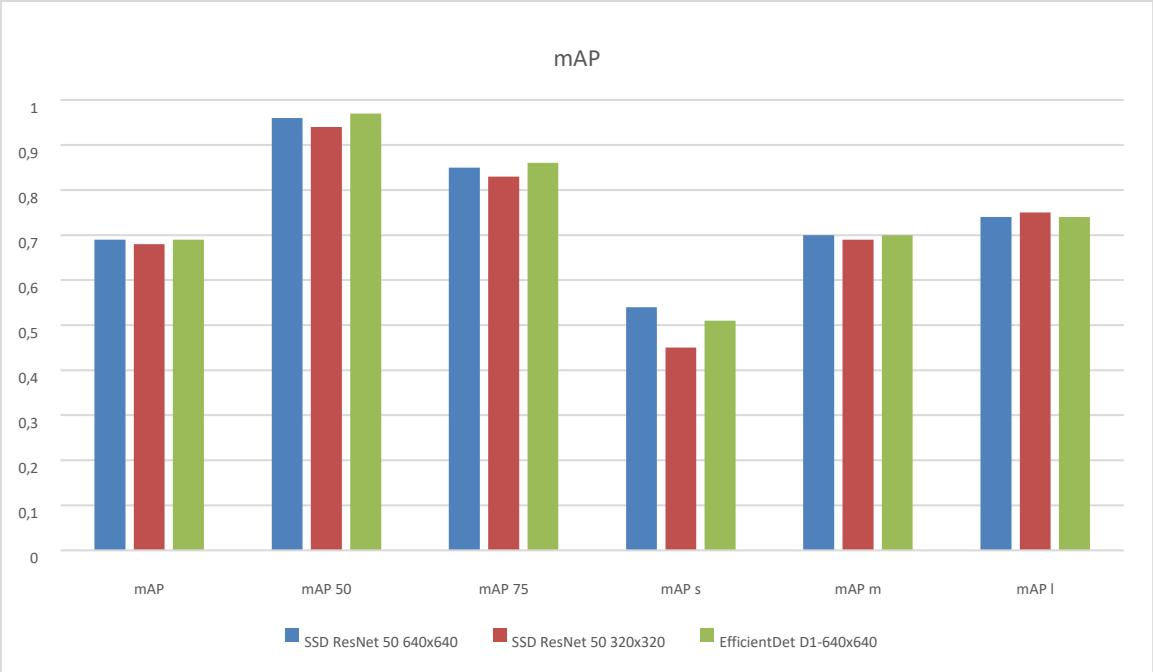
## 7.4.Usporedba evaluacije testnog seta između modela na 20 000 koraka

Sljedeća tablica prikazuje usporedbu tri navedena modela nad testnim setom za metriku mAP.

**Tablica 1** Rezultat evaluacije modela nad testnim skupom za metriku mAP

| **Model** | **mAP** | **$mAP_{50}$** | **$mAP_{75}$** | **$mAP_{small}$** | **$mAP_{medium}$** | **$mAP_{large}$** |
|---|---|---|---|---|---|---|
| SSD ResNet 50 640x640 | 0,69 | 0,96 | 0,85 | 0,54 | 0,7 | 0,74 |
| SSD ResNet 50 320x320 | 0,68 | 0,94 | 0,83 | 0,45 | 0,69 | 0,75 |
| EfficientDet D1-640x640 | 0,69 | 0,97 | 0,86 | 0,51 | 0,7 | 0,74 |

Rezultate vidljive iz tablice vizualno prikazuje sljedeći graf.




**Grafikon 1** Rezultat evaluacije za metriku mAP

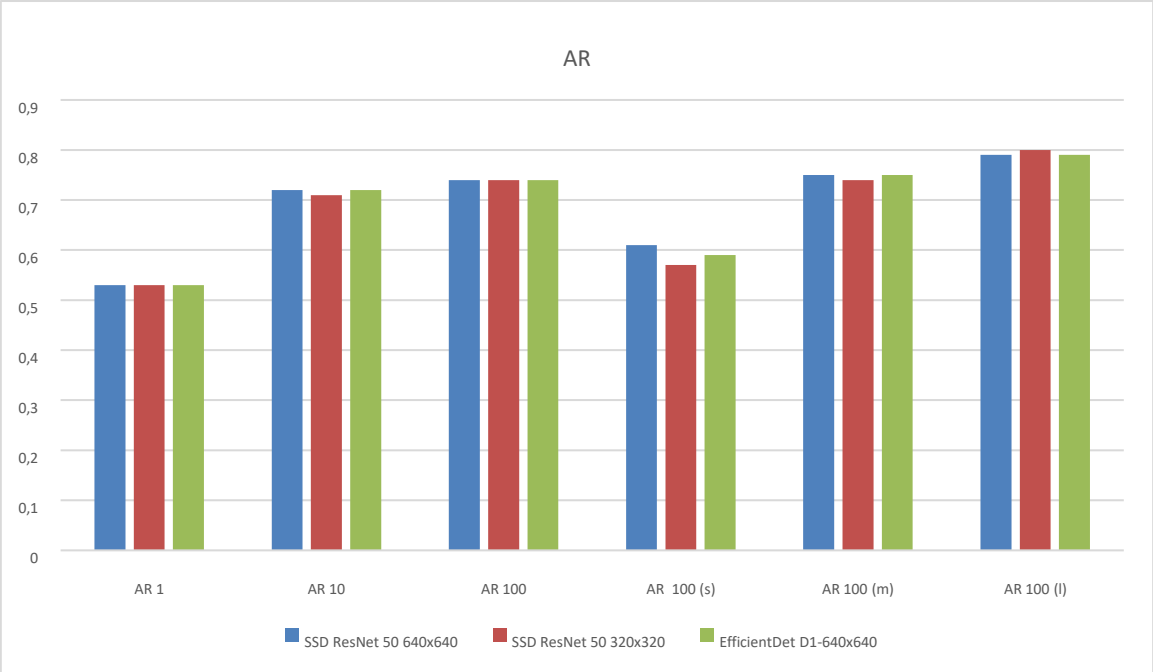
Ono što je moguće vidjeti da su svi rezultati u sličnom rasponu s malim razlikama. Što se time može zaključiti jest da je najveće ograničenje navedenim modelima bio mali skup podataka od 1700 slika s previše varijacija položaja čovjeka. Pod varijacijom položaja misli se na položaje glave (da li osoba gleda ravno, prema gore ili dolje), položaj ruke (da li se ruka nalazi na licu, je li obje ruke prekrivaju lice), položaj kose na licu i slično. Ono što se može pretpostaviti je da su svi modeli imali poteškoće nad određenim slikama koje su predstavljale preveliki izazov prilikom objektne detekcije.

Ako se pogledaju rezultati za metriku AR dobiju se rezultati koje pokazuje sljedeća tablica.

**Tablica 2** Rezultat evaluacije modela nad testnim skupom za metriku AR

| **Model** | **$AR_1$** | **$AR_{10}$** | **$AR_{100}$** | **$AR_{100\ (s)}$** | **$mAP_{100(m)}$** | **$mAP_{100\ (l)}$** |
|---|---|---|---|---|---|---|
| SSD ResNet 50 640x640 | 0,53 | 0,72 | 0,74 | 0,61 | 0,75 | 0,79 |
| SSD ResNet 50 320x320 | 0,53 | 0,71 | 0,74 | 0,57 | 0,74 | 0,8 |
| EfficientDet D1-640x640 | 0,53 | 0,72 | 0,74 | 0,59 | 0,75 | 0,79 |

Rezultate vidljive iz tablice vizualno prikazuje sljedeći graf.




**Grafikon 2** Rezultat evaluacije za metriku AR

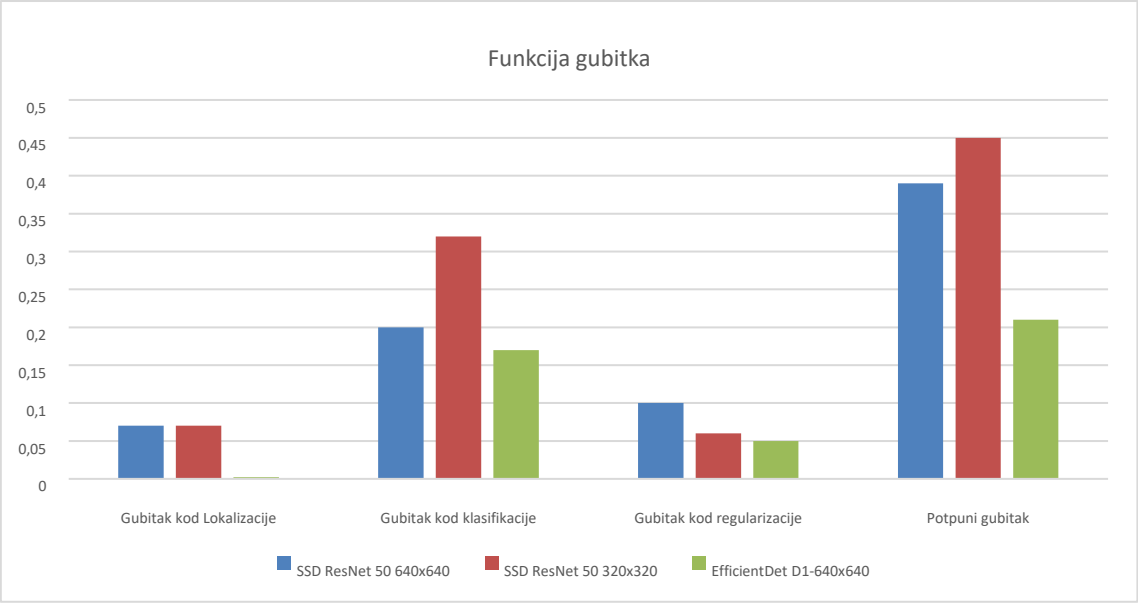
Moguće je vidjeti istu pretpostavku, kao i kod mAP rezultata, u kojem sva tri modela prikazuju relativno jednake rezultate za metriku AR.

I za kraj, ako se u tablici prikažu rezultati za metriku funkcije gubitka dobiju se sljedeći podaci.

**Tablica 3** Rezultat evaluacije modela nad testnim skupom za metriku funkcije gubitka

| Model | Gubitak kod Lokalizacije | Gubitak kod klasifikacije | Gubitak kod regularizacije | Potpuni gubitak |
|---|---|---|---|---|
| SSD ResNet 50 640x640 | 0,07 | 0,2 | 0,1 | 0,39 |
| SSD ResNet 50 320x320 | 0,07 | 0,32 | 0,06 | 0,45 |
| EfficientDet D1-640x640 | 0,002 | 0,17 | 0,05 | 0,21 |

Rezultate vidljive iz tablice vizualno prikazuje sljedeći graf.




**Grafikon 3** Rezultat evaluacije za metriku funkcije gubitka

Kod funkcije gubitka je moguće je vidjeti razlike između navedena tri modela u kojima je najbolje vrijednosti dobio EfficientDet D1 640x640 algoritam, dok je SSD ResNet-50 320x320 model naišao na probleme kod klasifikacije objekata i time imao najgori rezultat za funkciju gubitka za klasifikaciju i ukupni gubitak. Veliki gubitak kod klasifikacije za SSD ResNet-50 320x320 dogodio se i kod treniranja i razlog tome je *overfitting*.

### 7.5. Usporedba evaluacije testnog seta između modela kod optimalnog broja koraka

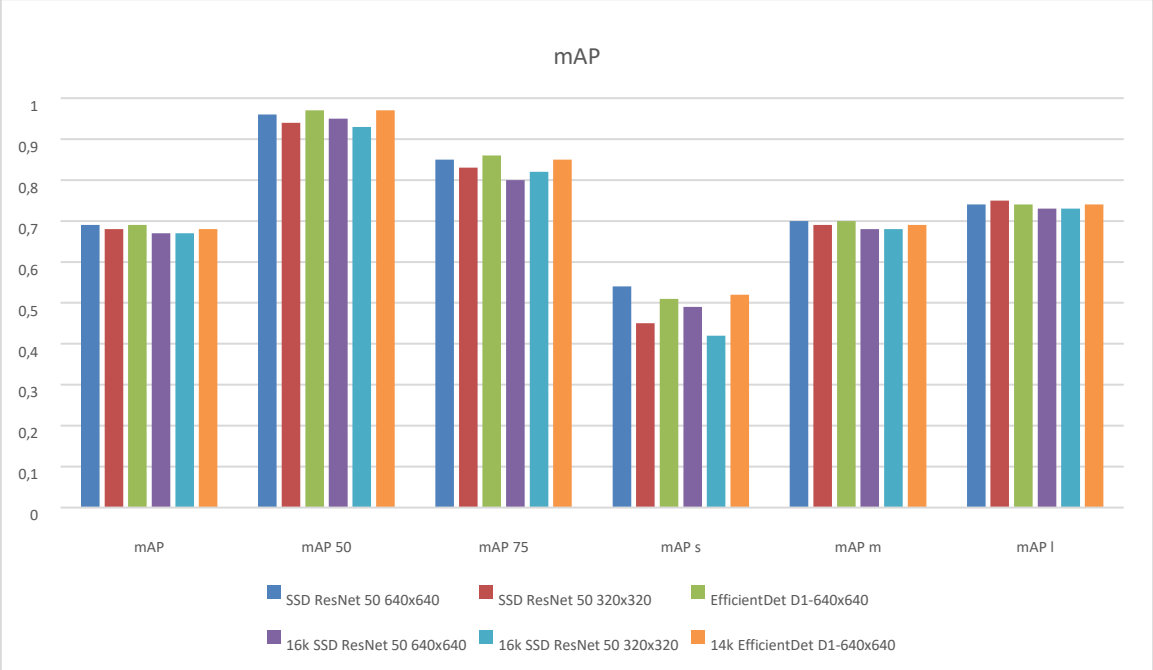
Ono što je vidljivo iz rezultata treniranja i usporedbe nad evaluacijskim skupom podataka je da su modeli zbog malog skupa podataka nakon određenog broja koraka počeli stagnirati ili padati. Optimalno vrijeme zaustavljanja treninga kod EfficientDet-a se bi bilo do 14 000 koraka, dok bi kod druga dva modela iznosilo 16 000 koraka. Sljedeće tablice prikazuju, uz prethodne rezultate do 20 000 koraka, rezultate za EfficientDet do 14 000 koraka i za SSD ResNet modele do 16 000 koraka.

Sljedeća tablica prikazuje usporedbu tri navedena modela (za 20 000 korake i optimalne korake) nad testnim setom za metriku mAP.

**Tablica 4** Rezultat evaluacije modela nad testnim skupom za metriku mAP

| Model | mAP | $mAP_{50}$ | $mAP_{75}$ | $mAP_{small}$ | $mAP_{medium}$ | $mAP_{large}$ |
|---|---|---|---|---|---|---|
| SSD ResNet 50 640x640 | 0,67 | 0,95 | 0,80 | 0,49 | 0,68 | 0,73 |
| SSD ResNet 50 320x320 | 0,67 | 0,93 | 0,82 | 0,42 | 0,68 | 0,73 |
| EfficientDet D1-640x640 | 0,64 | 0,94 | 0,79 | 0,52 | 0,64 | 0,70 |
| 16k SSD ResNet 50 640x640 | 0,67 | 0,95 | 0,8 | 0,49 | 0,68 | 0,73 |
| 16k SSD ResNet 50 320x320 | 0,67 | 0,93 | 0,82 | 0,42 | 0,68 | 0,73 |
| 14k EfficientDet D1-640x640 | 0,68 | 0,97 | 0,85 | 0,52 | 0,69 | 0,74 |

Rezultate vidljive iz tablice vizualno prikazuje sljedeći graf.




**Grafikon 4** Rezultat evaluacije za metriku funkcije gubitka

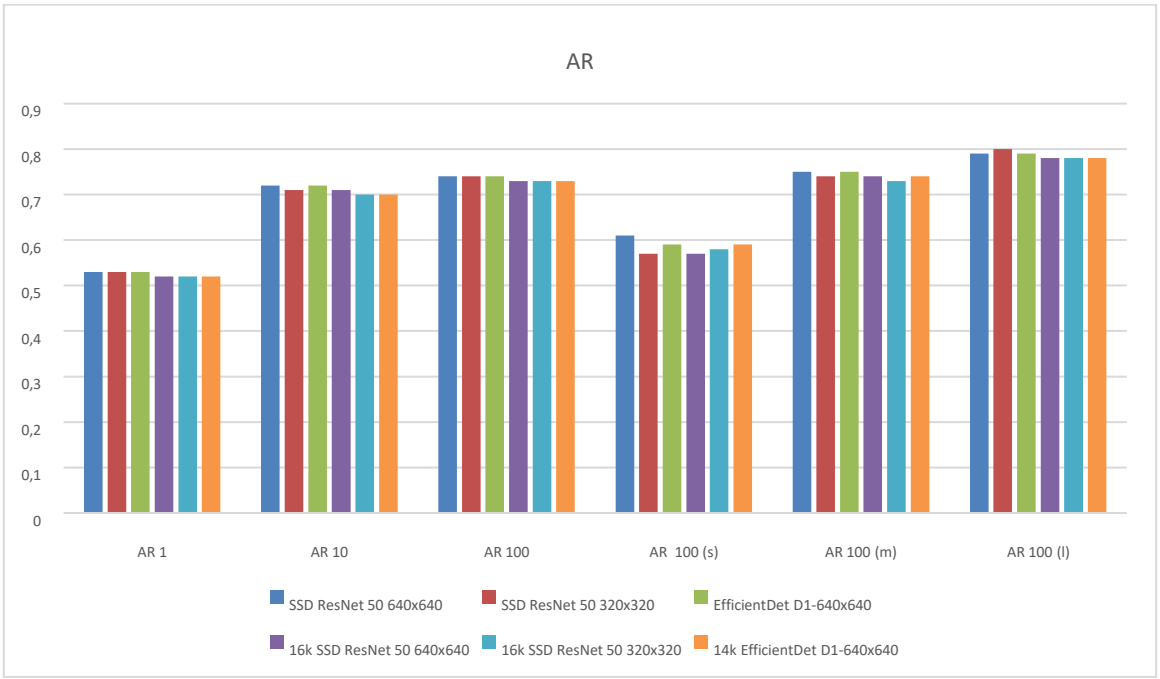
Vidljivo je da su rezultati nad mAP kod optimalnog vremena zaustavljanja blizu rezultata 20 000 koraka, čime navedene modele, nad relativno malim skupom podataka, nije potrebno trenirati do 20 000 koraka.

Sljedeća tablica prikazuje usporedbu tri navedena modela (nad 20 000 koraka i optimalnim brojem koraka) nad testnim setom za metriku AR.

**Tablica 5** Rezultat evaluacije modela nad testnim skupom za metriku AR

| Model | $AR_1$ | $AR_{10}$ | $AR_{100}$ | $AR_{100\ (s)}$ | $mAP_{100(m)}$ | $mAP_{100\ (l)}$ |
|---|---|---|---|---|---|---|
| SSD ResNet 50 640x640 | 0,52 | 0,71 | 0,73 | 0,57 | 0,74 | 0,78 |
| SSD ResNet 50 320x320 | 0,52 | 0,7 | 0,73 | 0,58 | 0,73 | 0,78 |
| EfficientDet D1-640x640 | 0,51 | 0,69 | 0,71 | 0,60 | 0,71 | 0,76 |
| 16k SSD ResNet 50 640x640 | 0,52 | 0,71 | 0,73 | 0,57 | 0,74 | 0,78 |
| 16k SSD ResNet 50 320x320 | 0,52 | 0,70 | 0,73 | 0,58 | 0,73 | 0,78 |
| 14k EfficientDet D1-640x640 | 0,52 | 0,70 | 0,73 | 0,59 | 0,74 | 0,78 |

Rezultate vidljive iz tablice vizualno prikazuje sljedeći graf.




**Grafikon 5** Rezultat evaluacije modela nad testnim skupom za metriku AR

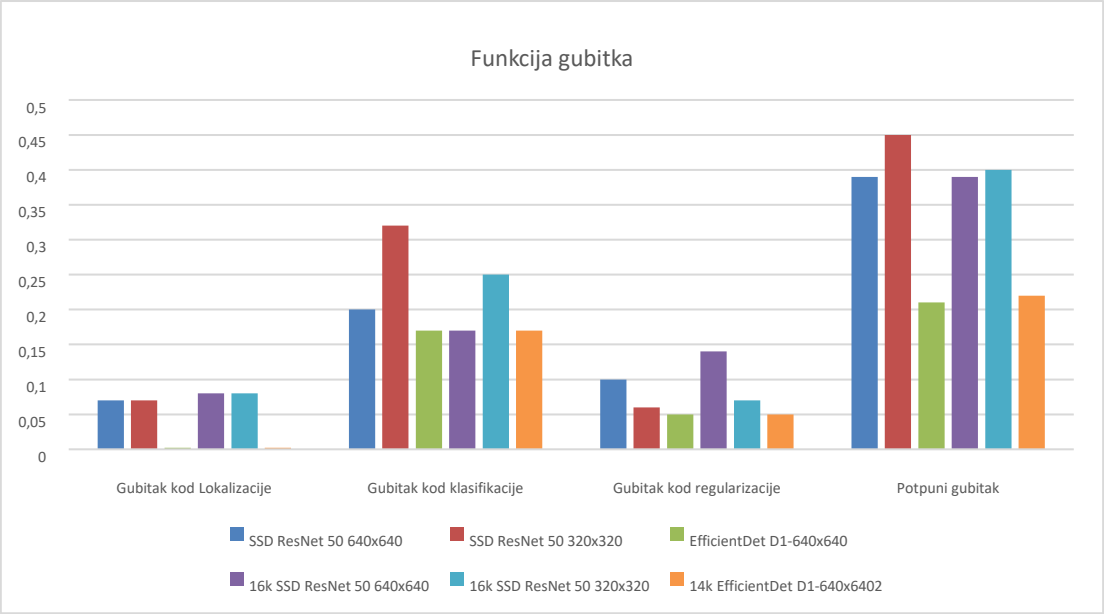
Moguće je vidjeti istu pretpostavku, kao i kod mAP rezultata, u kojem sva tri modela prikazuju relativno jednake rezultate za metriku AR bilo da je riječ o manjem broju koraka, ili 20 000 koraka.

Sljedeća tablica prikazuje usporedbu tri navedena modela (za 20 000 koraka i optimalni broj koraka) nad testnim setom za metriku funkcije gubitka.

**Tablica 6** Rezultat evaluacije modela nad testnim skupom za metriku funkcije gubitka

| **Model** | **Gubitak kod Lokalizacije** | **Gubitak kod klasifikacije** | **Gubitak kod regularizacije** | **Potpuni gubitak** |
|---|---|---|---|---|
| SSD ResNet 50 640x640 | 0,08 | 0,17 | 0,14 | 0,39 |
| SSD ResNet 50 320x320 | 0,08 | 0,25 | 0,07 | 0,4 |
| EfficientDet D1-640x640 | 0,002 | 0,22 | 0,05 | 0,27 |
| 16k SSD ResNet 50 640x640 | 0,08 | 0,17 | 0,14 | 0,39 |
| 16k SSD ResNet 50 320x320 | 0,08 | 0,25 | 0,07 | 0,4 |
| 14k EfficientDet D1-640x640 | 0,002 | 0,17 | 0,05 | 0,22 |

Rezultate vidljive iz tablice vizualno prikazuje sljedeći graf.




**Grafikon 6** Rezultat evaluacije modela nad testnim skupom za metriku funkcije gubitka

U ovom slučaju je vidljivo da su SSD ResNet modeli dobili bolje rezultate kod manjeg (optimalnog) broja koraka čime je pretpostavka da je došlo do *overfitting-a* kod 20 000 dokazana. EfficientDet je ostao jednak kod 14 000 koraka i 20 000 koraka.

### 7.6. Neoznačene slike kod testnog skupa

Za provjeru slika koje stvaraju probleme detekcije, korišteni su rezultati nad EfficientDet modelom.

Tijekom evaluacije testnog skupa, model je imao probleme detekcije kod slika koje sadrže osobe koje imaju vizir i masku istovremeno te bi za poboljšanje modela trebalo dodati još dodatnih slika s navedenim osobama kako bi model imao mogućnost boljeg učenja tijekom treniranja. Također, model je imao poteškoće kod prepoznavanja osoba sa zaštitnom maskom gdje osoba nosi zaštitnu masku i objekt preko glave, poput kapuljače. Osim navedenih oblika slika, model je imao poteškoće kod prepoznavanja osoba s maskom (ili bez maske) u slikama s manjim kontrastom. Sljedeća slika prikazuje slike od testnog skupa nad kojima je model imao poteškoće detekcije.


**Slika 71** Neoznačene slike kod testnog skupa

# 8. ZAKLJUČAK

Strojno učenje je znanost koja se tijekom godina i njenog sve većeg značaja implementirala u svaki dio ostalih znanosti te je postala neizostavan dio poslovanja velikih (pa i malih) tvrtki. U ovom radu je objašnjeno korištenje TensorFlow-a s TensorFlow API-om za objektnu detekciju te su pokazane prednosti korištenja navedenih platformi za pojedinca i tvrtku. Vidljivo je da je danas lakše nego ikad učiti vlastite modele uz brojne resurse koje nam TensorFlow daje. LabelImg alat olakšava pravljenje vlastitog skupa podataka uz jednostavan i intuitivan vizualni preglednik kojim se mogu označavati slike bilo da je riječ o malom ili velikom skupu podataka. Korištenjem TensorFlow API-a za objektnu detekciju moguće je napraviti vlastite modele koji daju izvrsne rezultate s obzirom na kompleksnost detekcije višestrukih objekata na slici ili videu. Također, uz TensorBoard biblioteku olakšano je praćenje raznih parametara tijekom treninga u vizualnom formatu.

I za kraj, pokazane su usporedbe kako se odabrani model ponaša ako mu se smanji veličina slika i poveća veličina grupe gdje je moguće vidjeti da, na vlastitom skupu podataka, oba modela pokazuju slične rezultate uz razliku u srednjoj prosječnoj preciznosti (skraćeno mAP) gdje model s većom veličinom grupe (a manjom veličinom slike) dobiva bolje rezultate. Međutim, model s većom veličinom slike (a manjom veličinom grupe) se pokazao otpornijim na *overfitting*. Također, odrađena je usporedba dva različita modela s istom veličinom slika i veličinom grupe sa svrhom usporedbe koji model daje bolje rezultate za promatrani skup podataka. Ono što je vidljivo je da oba modela, SSD ResNet i EfficientDet, daju dobre rezultate s obzirom na težinu i relativno malu veličinu i kompleksnost samog skupa podataka. Međutim, ipak se prednost daje EfficientDet-u s boljim rezultatima kod funkcije gubitka.

Ograničavajući faktor za navedeni projekt je bila veličina memorije grafičke kartice na Google Colab serverima čime se ograničila veličina skupa na 8 i 32 ovisno o dimenzijama slike. Osim veličine memorije grafičke kartice, ograničavajući faktor je bila i veličina skupa podataka koja je iznosila 1700 vlastito označenih slika što je nedovoljno za veći broj koraka i što preciznije treniranje što je vidljivo na prikazanim grafovima gdje nakon određenog broja koraka modeli imaju tendenciju da odlaze u *overfitting*. Time se zaključuje da je za daljnji napredak navedenog programskog rješenja preporučeno treniranje na većem skupu podataka i po mogućnosti jačim grafičkim karticama zbog povećanja veličine grupe i dimenzija slike.

# LITERATURA


[1] I. C. Education, »IBM,« [Mrežno]. Dostupno na: https://www.ibm.com/cloud/learn/what-is-artificial-intelligence. [Pokušaj pristupa 1. travnja 2021.]

[2] J. Frankenfield, »Investopedia,« 2021. [Mrežno]. Dostupno na: https://www.investopedia.com/terms/a/artificial-intelligence-ai.asp. [Pokušaj pristupa 1. travnja 2021.]

[3] »Artificial Intelligence (AI) Services & Solutions - KungFu,« [Mrežno]. Dostupno na: https://www.kungfu.ai/a-general-approach-for-using-2d-object-detection-for-facial-id/. [Pokušaj pristupa 10. travnja 2021.]

[4] R. V. Rodriguez, »Analytics India Magazine,« [Mrežno]. Dostupno na: https://analyticsindiamag.com/kinect-sensor-the-ai-tool-you-did-not-know-you-had/. [Pokušaj pristupa 10. travnja 2021.]

[5] »ML Glossary,« [Mrežno]. Dostupno na: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html. [Pokušaj pristupa 12. travnja 2021.]

[6] R. Parmar, »Towards Data Science,« 2018. [Mrežno]. Dostupno na: https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23. [Pokušaj pristupa 12. travnja 2021.]

[7] S. Yohanandan, »Towards Data Science,« 2020. [Mrežno]. Dostupno na: https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2. [Pokušaj pristupa 20. travnja 2021.]

[8] »NickZeng|曾广宇,« [Mrežno]. Dostupno na: https://blog.zenggyu.com/en/post/2018-12-16/an-introduction-to-evaluation-metrics-for-object-detection/. [Pokušaj pristupa 24. travnja 2021.]

[9] I. C. Education, »IBM,« 2020. [Mrežno]. Dostupno na: https://www.ibm.com/cloud/learn/neural-networks. [Pokušaj pristupa 2. svibnja 2021.]

[10] I. Goodfellow, Y. Bengio i A. Courville, Deep Learning, London: Massachusetts Institute of Technology, 2016.

[11] I. C. Education, »IBM,« [Mrežno]. Dostupno na: https://www.ibm.com/cloud/learn/convolutional-neural-networks. [Pokušaj pristupa 10. svibnja 2021.]

[12] »Tensorflow,« [Mrežno]. Dostupno na: https://www.tensorflow.org/guide/tensor. [Pokušaj pristupa 11. svibnja 2021.]

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed i C.-Y. Fu, »SSD: Single Shot MultiBox Detector,« 2016.

[14] »ArcGIS Developer,« [Mrežno]. Dostupno na: https://developers.arcgis.com/python/guide/how-ssd-works/. [Pokušaj pristupa 14. svibnja 2021.]

[15] K. He, X. Zhang, S. Ren i J. Sun, »Deep Residual Learning for Image Recognition,« 2015.

[16] M. Tan, R. Pang i Q. V. Le, »EfficientDet: Scalable and Efficient Object Detection,« 2020.

[17] M. Tan i A. Yu, »Google AI Blog,« 15. travnja 2020. [Mrežno]. Dostupno na: https://ai.googleblog.com/2020/04/efficientdet-towards-scalable-and.html. [Pokušaj pristupa 20. svibnja 2021.]

[18] M. Tan i Q. V. Le, »EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,« eprint arXiv:1905.11946, 2019.

[19] M. Tan i Q. V. Le, »Google AI Blog,« [Mrežno]. Dostupno na: https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html. [Pokušaj pristupa 20. svibnja 2021.]

[20] »GitHub,« [Mrežno]. Dostupno na: https://github.com/tensorflow/models. [Pokušaj pristupa 25. svibnja 2021.]

[21] »Google Developers,« [Mrežno]. Dostupno na: https://developers.google.com/protocol-buffers/. [Pokušaj pristupa 25. svibnja 2021.]

[22] Lxml, »LXML,« [Mrežno]. Dostupno na: https://lxml.de/. [Pokušaj pristupa 25. svibnja 2021.]

[23] F. Lundh i A. Clark, »Pillow,« [Mrežno]. Dostupno na: https://pillow.readthedocs.io/en/stable/. [Pokušaj pristupa 26. svibnja 2021.]

[24] »Cython,« [Mrežno]. Dostupno na: https://cython.org/. [Pokušaj pristupa 26. svibnja 2021.]

[25] »Pandas,« [Mrežno]. Dostupno na: https://pandas.pydata.org/. [Pokušaj pristupa 26. svibnja 2021.]

[26] »GitHub,« [Mrežno]. Dostupno na: https://github.com/google-research/tf-slim. [Pokušaj pristupa 26. svibnja 2021.]

[27] »LVIS,« [Mrežno]. Dostupno na: https://www.lvisdataset.org/. [Pokušaj pristupa 26. svibnja 2021.]

# PRILOZI

## Popis tablica



## Popis grafikona



## Popis slika

# IZJAVA

Izjavljujem pod punom moralnom odgovornošću da sam diplomski rad izradio samostalno, isključivo znanjem stečenim na studijima Sveučilišta u Dubrovniku, služeći se navedenim izvorima podataka i uz stručno vodstvo mentora izv.prof.dr.sc. Maria Miličevića, kome se još jednom srdačno zahvaljujem.

Jure Pirjać