Detekcija geografskih karakteristika Venere korištenjem algoritama strojnog učenja

Đuranović, Daniel

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka, Faculty of Engineering / Sveučilište u Rijeci, Tehnički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:190:009642

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij elektrotehnike

Diplomski rad

DETEKCIJA GEOGRAFSKIH KARAKTERISTIKA VENERE KORIŠTENJEM ALGORITAMA STROJNOG UČENJA

Rijeka, rujan 2022.

Daniel Đuranović

0069077256

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij elektrotehnike

Diplomski rad

DETEKCIJA GEOGRAFSKIH KARAKTERISTIKA VENERE KORIŠTENJEM ALGORITAMA STROJNOG UČENJA

Mentor: Prof. dr. sc. Zlatan Car

Rijeka, rujan 2022.

Daniel Đuranović

0069077256

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA DIPLOMSKE ISPITE

Rijeka, 21. ožujka 2022.

Zavod: **Zavod za automatiku i elektroniku**
Predmet: **Primjena umjetne inteligencije**
Grana: **2.03.06 automatizacija i robotika**

ZADATAK ZA DIPLOMSKI RAD

Pristupnik: **Daniel Đuranović (0069077256)**
Studij: Diplomski sveučilišni studij elektrotehnike
Modul: Automatika

Zadatak: **Detekcija geografskih karakteristika Venere korištenjem algoritama strojnog učenja/Detection of Venus geographic characteristics using machine learning algorithms**

Opis zadatka:

Izvršiti pregled literature iz područja primjene umjetne inteligencije u astronomiji, posebice detekciji geografskih karakteristika planetoida. Korištenjem javno dostupnog seta podataka snimaka satelita planetarne površine Venere i algoritama strojnog učenja razviti sustav za određivanje prisutnosti vulkana. Za potrebe označavanja snimaka razviti sustav konverzije iz LXYR formata u format primjeren za učitavanje u modele strojnog učenja (YOLO, CSV ili slično). Varirati hiperparametre modela, te komentirati utjecaj hiperparametara na performanse razvijenih modela.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 21. ožujka 2022.

Mentor:

Prof. dr. sc. Zlatan Car

Predsjednik povjerenstva za diplomski ispit:

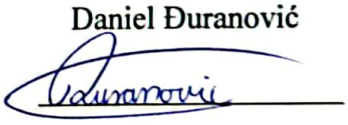
Prof. dr. sc. Viktor Sučić

IZJAVA

Sukladno članku 9. Pravilnika o diplomskom radu, diplomskom ispitu i završetku diplomskih sveučilišnih studija, izjavljujem da sam ovaj rad samostalno izradio koristeći stečena znanja tijekom studiranja.

Rijeka, rujan 2022.

Daniel Đuranović



ZAHVALA

Zahvaljujem se prof. dr. sc. Zlatanu Caru, dipl. ing. stroj. za pruženu priliku izrade diplomskoga rada pod njegovim mentorstvom. Naknadno se zahvaljujem asistentu Sandiu Baressiu Šegoti, mag. ing. comp. na pomoći, razumijevanju i strpljenju prilikom izrade istog.

.

Od Srca se zahvaljujem svojim roditeljima, obitelji te svim bližnjima što su me podržali, pružali mi potporu i trpili me tijekom mojeg školovanja.

.

Na kraju se zahvaljujem svim prijateljima i kolegama s Tehničkog Fakulteta Rijeka što su ovo iskustvo studiranja učinili zanimljivim i nezaboravnim.

Daniel Đuranović

Sadržaj

1. UVOD

Razvojem Interneta i hardverskih resursa u posljednjih dvadesetak godina olakšan je pristup velikom broju podataka. Generirani podaci posljednjih godina šire se u raznolikosti i složenosti te raste i brzina njihovog sakupljanja. Takav eksponencijalni rast količine podataka zahtjeva jednostavne, brze i točne alate za njihovu obradu. Kao rješenje tu uskače umjetna inteligencija (eng. *Artifical Intelligence - AI*). AI je grana računalne znanosti koja se bavi razvojem sposobnosti strojeva da obavljaju zadatke za koje je potreban nekakav oblik inteligencije [1]. Svoju upotrebu nalazi u računalnim igrama, simulacijama, obradi jezika i govora, računalnom vidu (raspoznavanje predmeta), upravljanju automatskim procesima, obradi velike količine podataka i još mnogo drugih područja. Računalni vid jedan je od najatraktivnijih istraživačkih područja današnjice. Znanstvenici u ovome području postigli su golemi napredak posljednjih godina, od razvoja algoritma za upravljanje autonomnih vozila [2], pa sve do otkrivanja karcinoma koristeći samo informacije sa slika [3].

U prošlosti, prikupljanje podataka u astronomiji bio je manualni i vremenski zahtjevan posao za čovjeka. No napretkom tehnologije i automatizacijom, danas je količina prikupljenih podataka promatrajući nebo gotovo nezamisliva. Tim povodom AI, pogotovo računalni vid, pronalazi svoju uporabu u ovoj grani znanosti. Od klasifikacije galaksija na noćnome nebu, pa sve do mapiranja površina planeta u znanstvene svrhe ili povodom planiranja budućih istraživačkih misija.

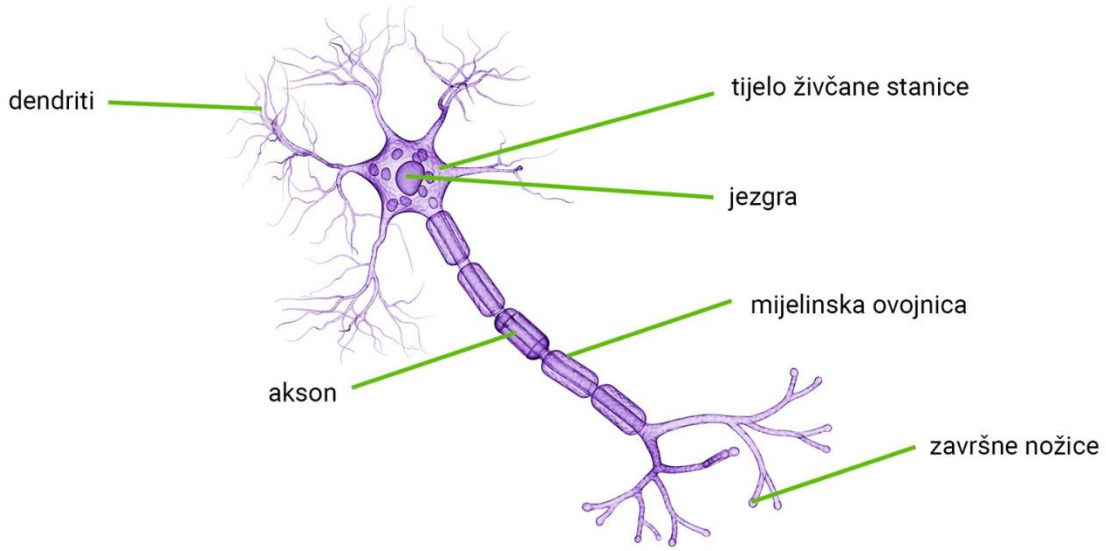
Tema ovog rada je razviti sustav umjetne inteligencije za prepoznavanje vulkana sa satelitskih snimaka površine planeta Venere, dok je cilj objasniti primjenu umjetne inteligencije u području računalnog vida i detekcije objekata, proces pripreme ulaznih podataka iz Magellan seta podataka za treniranje modela te dobiveni model primijeniti na testnim uzorcima i ocijeniti performanse istog. Također diskutirat će se problem učenja i validacije modela s relativnom malim setom podataka, te promotriti rješenja s kojima se došlo do zadovoljavajućih rezultata. Kao detekcijski algoritam odabran je YOLOv5 algoritam koji je trenutno jedan od najpreciznijih i najbržih algoritama za detekciju objekata. Na kraju rada analizirat će se dobiveni rezultati te će se navesti smjernice za buduća istraživanja.

2. UMJETNE NEURONSKE MREŽE

Umjetna neuronska mreža je računalni model koji oponaša rad živčanih stanica u ljudskome mozgu [4]. Oponašanje ljudskoga mozga računalnim programima ili sustavima omogućuje raspoznavanje obrasca i rješavanje uobičajenih problema u područjima umjetne inteligencije ili strojnog učenja. Svaka određena misao ili reakcija rezultira aktivacijom međusobno povezanih neurona u ljudskome mozgu. Tijekom svojeg života, ljudi svakodnevnim učenjem, pamćenjem, razmišljanjem stvaraju nove i jačaju stare veze između neurona. Slično tome, umjetne neuronske mreže također stječu znanje kroz nekakav proces učenja, te isto znanje pohranjuju u vezama između neurona u obliku težinskih faktora. Kako bi se mogao razumjeti princip rada neuronske mreže, prvo će se opisati implementacija jednog biološkog neurona na računalu.

2.1. Biološki i umjetni neuron

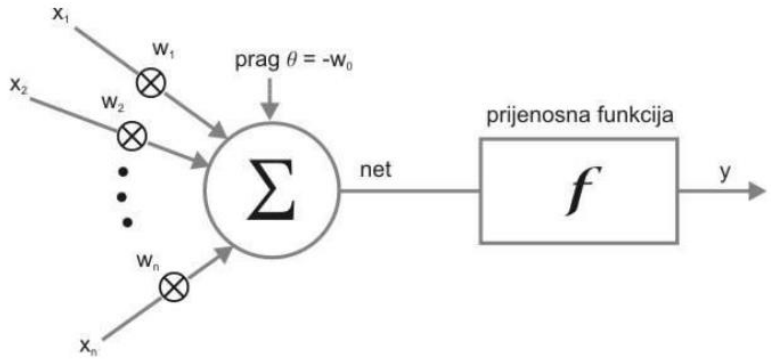
Sastavni dio ljudskog mozga koji omogućuje njegovo djelovanje je neuron (živčana stanica). Ljudski mozak sastoji se od 100 milijardi neurona koji su raspoređeni shodno svojoj svrsi, te oni čine 80% njegove mase [5]. Među neuronima u ljudskome mozgu postoji 10^{15} konekcija [6]. Svaki biološki neuron sastoji se od tri osnovna dijela: tijela, dendrita i aksona (Slika 2.1.) [7].



Slika 2.1. Građa biološkog neurona [8]

„Tijelo stanice sadrži informaciju predstavljenu električkim potencijalom između unutrašnjeg i vanjskog dijela stanice (oko *-70 mV* u neutralnom stanju)“ [9]. Informacije između neurona prenose se preko sinapsa koje su prekrivene dendritima. Neuroni međusobno prenose informacije u obliku uzbudljivog post-sinaptičkog potencijala koji depolarizira membranu i pomiče potencijal bliže pragu da se generira akcijski potencijal [10]. Kako neuron na sebe ima više vezanih ulaza, tj. povezan je sa više neurona odjednom (do nekoliko tisuća njih), u tijelu stanice se sumiraju svi ulazni post-sinaptički potencijali, te ukoliko ukupni napon na tijelu pređe određeni prag dolazi do generiranja akcijskog potencijala. Generiranjem akcijskog potencijala, informacija se prenosi na završetke neurona i odlazi do daljnje povezanih neurona, te se cijeli postupak ponovno ponavlja za svaki neuron u ljudskome mozgu.

Prvi korak prema računalnom modelu neurona koji danas koristimo kao temelj neuronskih mreža, predstavili su 1943. Godine McCulloch i Pitts, oponašajući funkcionalnost biološkog neurona [11]. Računalni model umjetnog neurona prikazan je na Slici 2.2. Model funkcionira na slijedeći način: ulazni signali u neuron opisani su numeričkim iznosom ($x_1, x_2, \ldots, x_n$) te se množe težinskim faktorima (eng. *weights*) koji predstavljaju jakost sinapse ($w_1, w_2, \ldots, w_n$). Zatim se svi ulazni signali koji su pomnoženi s težinskim faktorima sumiraju, što je analogno sumiranju post-sinaptičkog potencijala kod biološkog neurona. Ukoliko dobiveni iznos sume pređe određeni iznos praga, neuron daje signal na svojem izlazu. Takav model umjetnog neurona naziva se *Treshold Logic Unit (TLU).* U općenitijim slučajevima upotrebe, umjetni neuron umjesto funkcije praga koristi takozvanu aktivacijsku (prijenosnu) funkciju, tj. rezultat sume zbrojen za nekakvom numeričkom vrijednosti praga (eng. *bias*), prolazi kroz jednu od mnogih mogućih matematičkih funkcija koja na kraju daje nekakvu izlaznu vrijednost [9].



Slika 2.2. Model umjetnog neurona [9]

Gore opisani neuron sa aktivacijskom funkcijom predstavljen je sa jednadžbom 2.1.:

$$y = f((\sum_{i=1}^{n} x_i w_i) + \theta) \qquad (2.1.)$$

gdje je:

y izlazna vrijednost iz neurona,

f aktivacijska funkcija,

x_i vrijednost ulaznog signala,

w_i vrijednost težinskog faktora i

θ vrijednost praga [9].

2.2. Aktivacijske funkcije

„Aktivacijska funkcija predstavlja „matematička vrata“ između ulaza koji napaja trenutni neuron i njegovog izlaza koji prelazi u slijedeći sloj, analogno akcijskom potencijalu u biološkom neuronu“ (Slika 2.3.) [12].

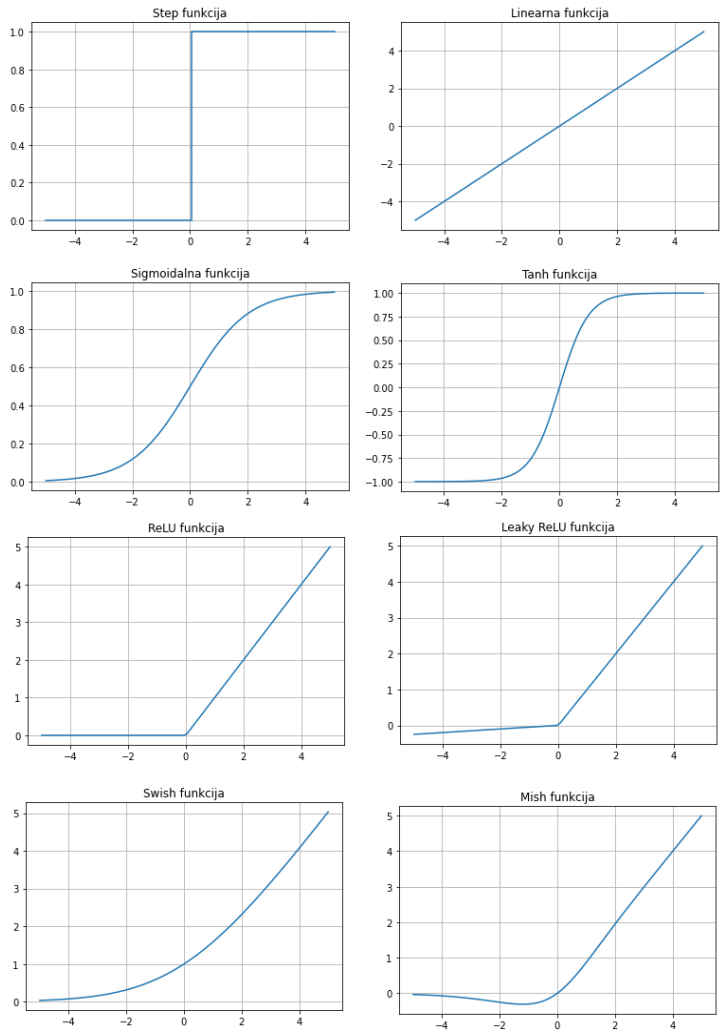
Slika 2.3. Aktivacijska funkcija [12]

Aktivacijske funkcije uvode dodatnu složenost u umjetnim neuronskim mrežama. Ukoliko neuronska mreža ne sadrži aktivacijske funkcije, u tome slučaju svaki neuron izvodi samo linearnu transformaciju nad ulazima koristeći samo težinske faktore i funkcije praga. Iako linearne transformacije čine neuronsku mrežu jednostavnijom, ta bi mreža bila manje sposobna moći naučiti složene obrasce iz podataka. „Neuronska mreža bez aktivacijske funkcije u biti je samo model linearne regresije“ [13]. Stoga se koristi nelinearna transformacija ulaza neurona,

a ova nelinearnost u mreži uvedena je aktivacijskom funkcijom. Neke od popularnijih aktivacijskih funkcija koje se danas koriste su:

- step funkcija,
- linearna funkcija,
- Sigmoidalna funkcija,
- Tanh funkcija,
- ReLU funkcija,
- Leaky ReLu funkcija,
- Swish funkcija i
- Mish funkcija [13].

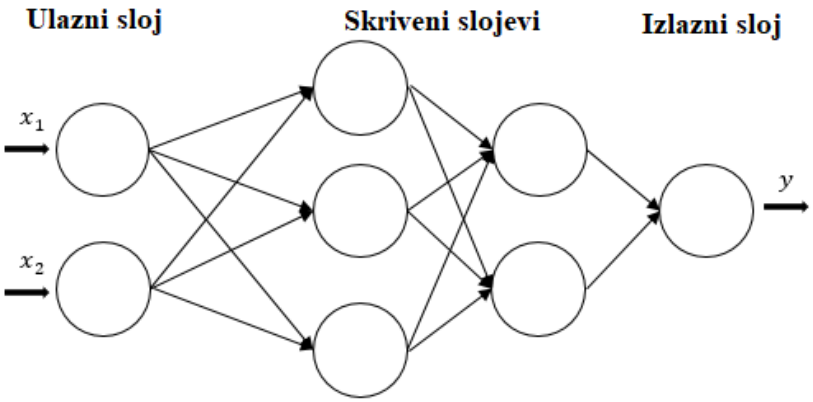
Gore navedene aktivacijske funkcije prikazane su na Slici 2.4.



Slika 2.4. Aktivacijske funkcije [autor]

2.3. Način djelovanja neuronske mreže

Umjetna neuronska mreža sačinjena je od međusobno povezanih čvorova, tj. neuroni su međusobno povezani na način tako da izlaz iz jednog neurona predstavlja ulaz u jedan ili više susjednih neurona. Klasična neuronska mreža sastoji se od ulaznog sloja, skrivenog sloja i izlaznog sloja (Slika 2.5.). U ulaznom sloju neuronske mreže dovode se podaci u neuronsku mrežu, dok u skrivenom sloju se isti obrađuju te na izlaznom sloju se reprezentiraju rezultati za dane ulazne podatke.



Slika 2.5. Primjer strukture neuronske mreže sa potpuno povezanim skrivenim slojevima [14]

Najjednostavniji primjer neuronske mreže bila bi mreža koja sadrži samo jedan ulazni i izlazni sloj. Povećanjem broja skrivenih slojeva neuronske mreže, povećava se njena kompleksnost te se mreže sa velikim brojem skrivenih slojeva nazivaju duboke neuronske mreže.

Učenje neuronske mreže postiže se podešavanjem težinskih koeficijenata i vrijednosti praga neurona. Parametri neuronske mreže podešavaju se na taj način da nakon dobivenih rezultata sa izlaznog sloja, mjeri se odstupanje izlaznog rezultat od stvarne vrijednosti te se na temelju tog rezultata unazadnom propagacijom prilagođavaju težinski koeficijenti neurona. Takav proces učenja neuronske mreže naziva se učenje s učiteljem te je jedan od metoda učenja neuronskih mreža- Sve metode učenja neuronskih mreža biti će detaljnije opisane u daljnjem poglavlju.

2.3.1. Funkcije gubitaka

Funkcija gubitka u neuronskoj mreži mjeri odstupanje izlaznog rezultata iz mreže od stvarne vrijednost pridružene tom rezultatu. Iznos funkcije gubitaka je veći samo ako je objekt potpuno pogrešno klasificiran. Ulazni podaci u neuronsku mrežu su fiksni i nepromjenjivi, no parametri mreže kao što su težinski koeficijenti i pomak su varijabilni, te utjecajem na njih je moguće minimizirati iznos funkcije gubitaka. Neke od korištenijih funkcija gubitaka su:

- srednja kvadratna pogreška (eng. *Mean Squared Error*) i
- unakrsna entropija (eng. *Cross entropy*) [15].

Srednja kvadratna pogreška izračunava se kao prosjek kvadrata razlika između predviđenih i stvarnih izlaznih vrijednosti, te je definirana izrazom 2.2.:

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \qquad (2.2.)$$

gdje je:

L – iznos funkcije gubitka,

N - ukupni broj uzorka,

y_i - stvarna vrijednost izlaza i

$\hat{y}_i$ - dobivena izlazna vrijednost [15].

Srednja kvadratna pogreška koristi se u slučajevima gdje su izlazne vrijednosti realni brojevi. Unakrsna entropija prikladna je u slučajevima binarne klasifikacije gdje bi rezultat izlaza bio jedan od dva moguća potencijalna ishoda, te je definirana izrazom 2.3.:

$$L = -\frac{1}{N}\sum_{i=0}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \qquad (2.3.)$$

gdje je:

L – iznos funkcije gubitka,

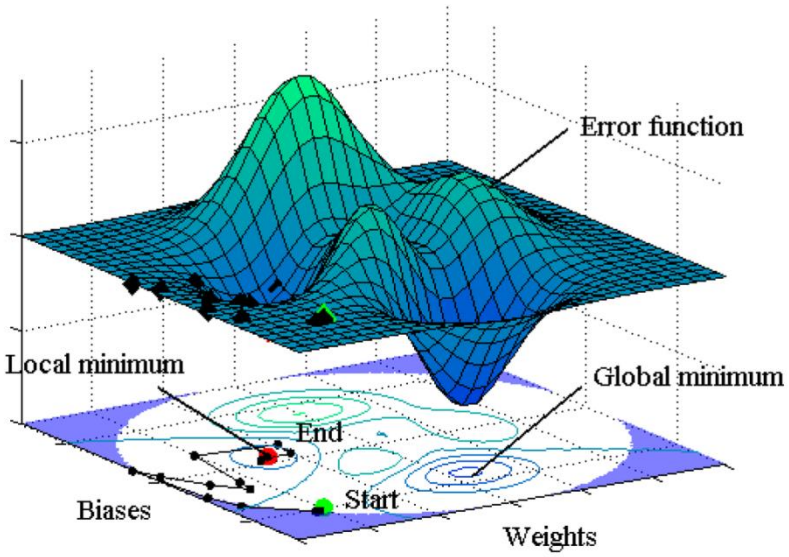
N - ukupni broj uzorka,

y_i - stvarna vrijednost izlaza i

$\hat{y}_i$ - dobivena izlazna vrijednost [15].

2.3.2. Optimizacijski algoritmi

Optimizacijski algoritmi su algoritmi ili metode kojima se mijenjaju parametri neuronske mreže kako bi se smanjio iznos funkcije gubitaka i povećala preciznost modela [16]. Proces nalaženja minimalne vrijednosti funkcije gubitaka svodi se na postupak traženja minimuma derivacije funkcije u prostoru rješenja (Slika 2.6.). Stoga aktivacijske funkcije u neuronskim mrežama moraju biti derivabilne. Ukoliko funkcija sadrži više varijabli, ulogu derivacije preuzima gradijent. Korištenjem gradijenta određuje se smjer pogreške i time je moguće utjecati na točno određene vrijednosti težinskih koeficijenata kako bi se dobilo zadovoljavajuće unaprjeđenje rezultata.



Slika 2.6. Primjer prostora rješenja [17]

Neki od češće korištenih optimizacijskih algoritama su:

- Gradijent spust,
- Stohastički gradijent spust i
- Adam algoritam [16].

2.4. Metode učenja umjetne neuronske mreže

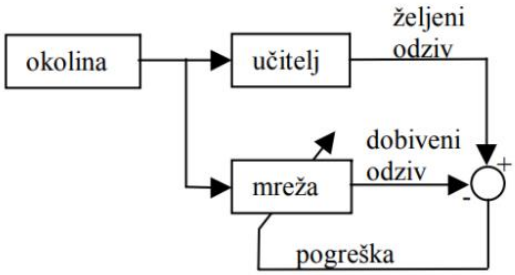
Kako je već navedeno u potpoglavlju 2.3., učenje neuronske mreže postiže se podešavanjem težinskih koeficijenata i vrijednosti praga neurona. Prilikom učenja neuronske mreže potrebno

je dobro poznavati ono o čemu se uči, tj. potrebno je poznavati ulazne podatke i očekivane izlaze podatke [17]. Postoje tri načina učenja neuronskih mreža, a oni su:

- učenje s učiteljem (eng. *supervised learning*),
- učenje bez učitelja (eng. *unsupervised learning*) i
- ojačano učenje (eng. *reinforcement learning*).

2.4.1. Učenje s učiteljem

Učenje s učiteljem definirano je korištenjem prethodno označenih skupova podataka za treniranje algoritama koji klasificiraju podatke ili točno predviđaju ishode [16]. Kod takvog načina učenja neuronske mreže, koristi se podjela seta podataka na set za učenje i set za testiranje. Set podataka za učenje koristi se kako bi se model naučio da daje željeni ishod tj. da model daje točne izlazne vrijednosti za ulazne vrijednosti koji prethodno nisu poznate. Težinski koeficijenti podešavaju se na temelju pogreške između željenog i dobivenog izlaza na neki ulazni vektor, gdje je pogreška razlika između željenog i dobivenog odziva. Zatim se proces iterativno ponavlja sve dok mreža ne nauči imitirati set podataka za učenje („učitelja“) [19]. Shematski prikaz učenja s učiteljem prikazan je na Slici 2.7.

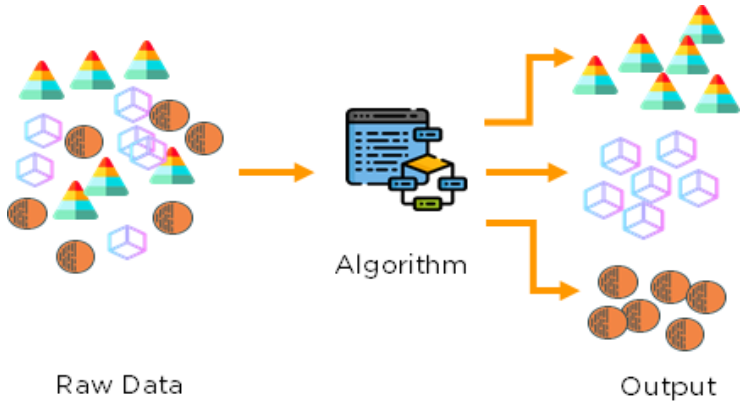


Slika 2.7. Shematski prikaz metode učenja s učiteljem [12]

2.4.2. Učenje bez učitelja

Učenje bez učitelja odnosi se na upotrebu algoritama umjetne inteligencije za prepoznavanje uzoraka u skupovima podataka koji sadrže nekakve podatke koji nisu prethodno klasificirani ili označeni. Algoritmima je stoga dopušteno klasificirati, označavati i/ili grupirati podatke unutar skupova bez ikakvih vanjskih smjernica (kao što je kod učenja s učiteljem) u obavljanju tog zadatka. Drugim riječima učenje bez učitelja omogućuje sustavu da sam identificira obrasce unutar skupova podataka [20]. Kod ovog tipa učenja sustav umjetne inteligencije će grupirati

nerazvrstane podatke prema sličnostima i razlikama iako nema ponuđenih kategorija. Algoritmi za učenje bez učitelja mogu obavljati složenije zadatke nego sustavi koji su trenirani sa metodom učenja s učiteljem. Osim toga podvrgavanje sustava učenju bez učitelja jedan je od načina testiranja umjetne inteligencije. Shematski prikaz metode učenja bez učitelja prikazan je na Slici 2.8.



Slika 2.8. Shematski prikaz metode učenja bez učitelja [21]

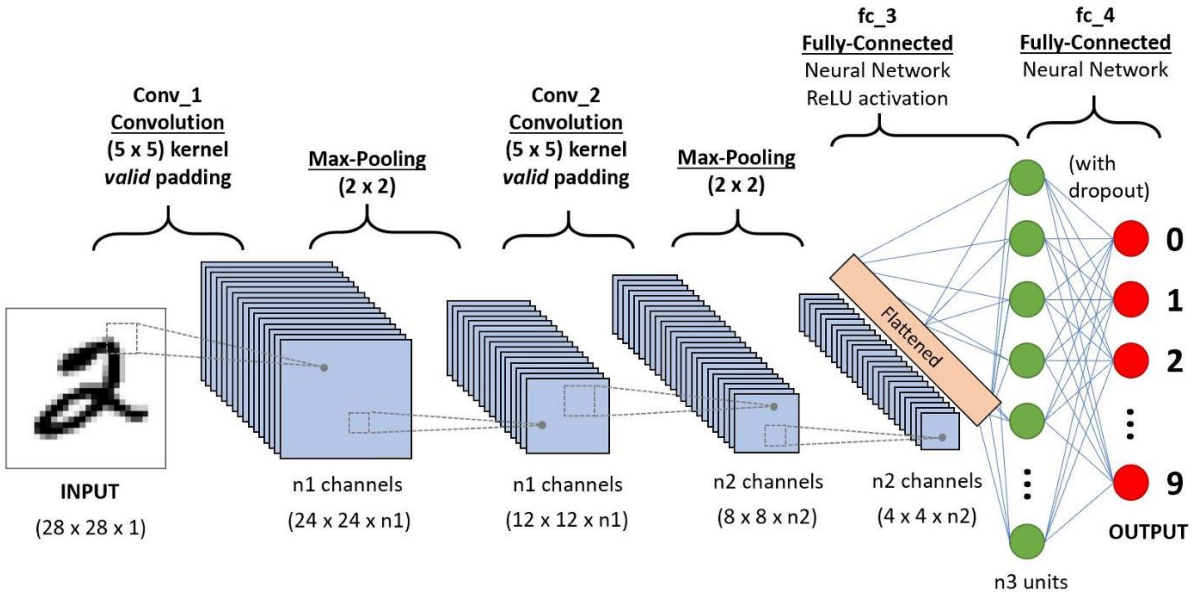
2.4.3. Ojačano učenje

Ojačano učenje je metoda učenja modela strojnog učenja za donošenje niza odluka. Kod ovog tipa učenja, umjetna inteligencija suočava se sa situacijom nalik na igru. Računalo (još nazivan i Agent) koristi metode pokušaja i pogreške kako bi došlo do rješenja problema. Kako bi se agenta natjeralo da radi ono što programer želi, agent dobiva nagrade ili kazne za radnje koje izvodi (nagrada je skalarna vrijednost). Cilj mu je maksimizirati ukupnu nagradu. Iako programer postavlja pravila same igre, on modelu ne daje savjete niti prijedloge kako uspješno doći do nagrade. Na agentu je da sam smisli kako izvršiti zadatak, tj. maksimizira nagradu, počevši od potpuno nasumičnih pokušaja pa sve do sofisticiranih taktika [22].

2.5. Konvolucijske neuronske mreže

Obrada slika jedno je od područja u kojem se metodama dubokog učenja ostvario poprilično velik napredak. Jednostavnije strukture neuronskih mreža vrlo su ograničene kad se dolazi do pitanja klasifikacije slika. Dodavanjem slojeva u jednostavnije strukture neuronskih mreža u načelu rješava taj problem, no zbog toga broj neurona postaje prevelik te usporedno sa time

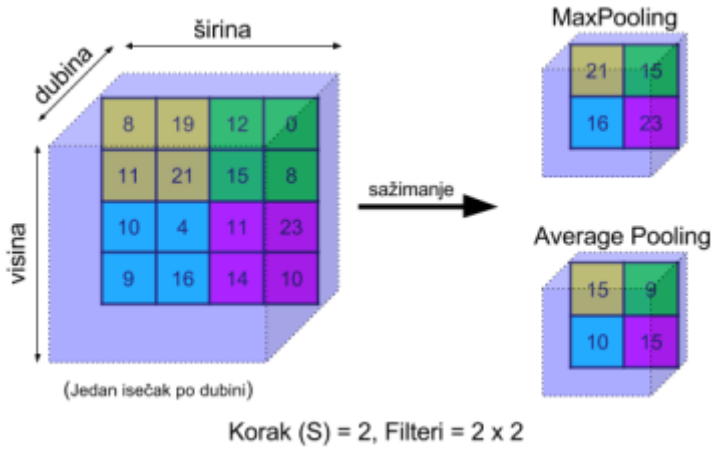
raste i potreba za sve većom količinom podataka za učenje kako bi se postigla zadovoljavajuća preciznost modela. 1998. godine predstavljena je nova struktura umjetnih neuronskih mreža koja je postala jedan od temelja primjene neuronskih mreža u problemu obrade slika [23]. Takve strukture neuronskih mreža danas se nazivaju konvolucijske neuronske mreže (eng. *Convolutional Neural Networks - CNN*). CNN sačinjava jedan ili više konvolucijskih slojeva i opcionalno jedan ili više potpuno povezanih slojeva. CNN su projektirane na način da prednost postižu u radu sa 2D strukturama, kao što su slike ili ulazi poput govornog signala [24]. Na Slici 2.9. prikazan je primjer strukture konvolucijske neuronske mreže.



Slika 2.9. Struktura konvolucijske neuronske mreže [25]

CNN se sastoji od ulaznog sloja na čiji se ulaz dolazi nekakva slika ili signal govora. Nakon ulaznog sloja dolazi jedan ili više uzastopnih konvolucijskih slojeva. U konvolucijskom sloju se na ulazne podatke primjenjuje matematička operacija konvolucije sa željenim filtrom (eng. *kernel*) te se time stvara takozvana mapa značajki. Filteri su predstavljeni sa dvodimenzionalnom matricom malih dimenzija i sastoje se od realnih vrijednosti [24]. Postoje različite vrste filtera koje se koriste ovisno o željenim značajkama koje se žele izlučiti (npr. detekcija rubova, uklanjanje šuma i sl.). Nakon konvolucijskog sloja dolazi sloj sažimanja (eng. *pooling layer*). Uobičajeno je da se u CNN periodično, između sukcesivnih konvolucijskih slojeva, umetne sloj sažimanja. Primjenom sloja sažimanja smanjuje se broj parametara, paralelno sa time i broj izračuna unutar same mreže. Najčešće korištene operacije sažimanja su

sažimanje maksimumom (eng. *max pooling*) i sažimanje sa srednjom vrijednošću (eng. *average pooling*). Primjeri gore navedenih dviju operacija prikazani su na Slici 2.10.

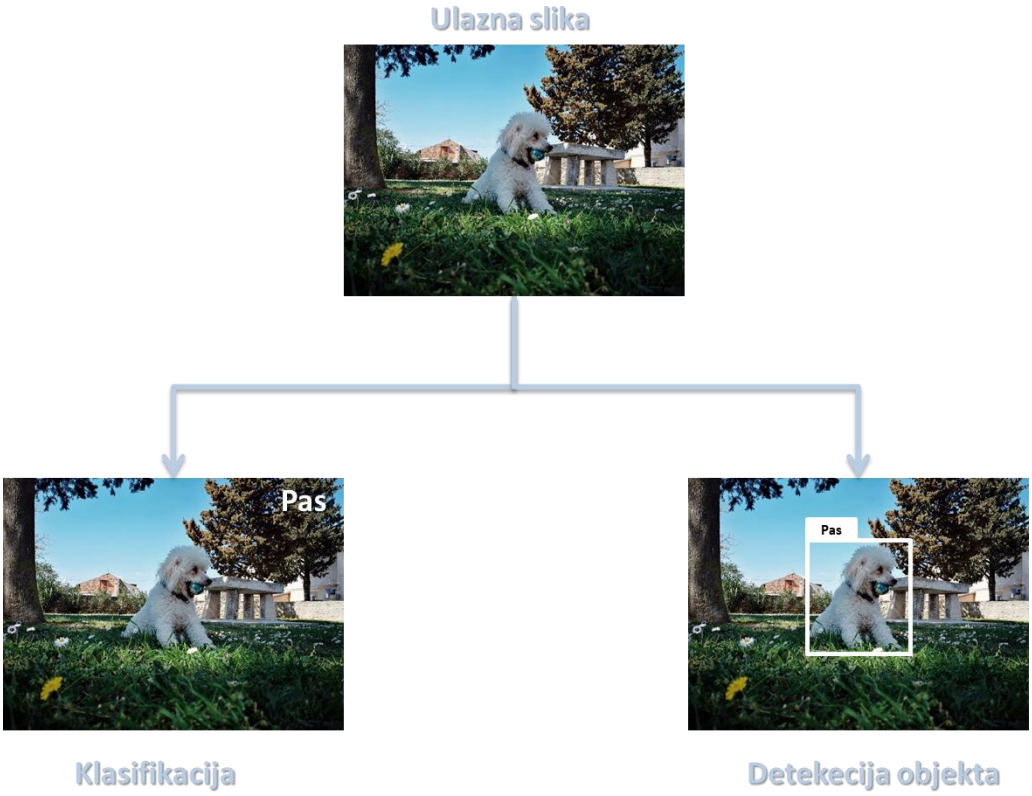


Slika 2.10. Max i Average pooling [24]

Nakon više uzastopnih konvolucijskih slojeva i slojeva sažimanja dolaze potpuno povezani slojevi, gdje je svaki od neurona povezan sa svim neuronima iz prethodna sloja (eng. *Multy Layer Percepteron MLP*). Ovaj zadnji sloj je ustvari klasična struktura umjetnih neuronskih mreža čiji je primjer prikazan na Slici 2.5. Na izlazu iz potpuno povezanog sloja dolaze rezultati klasifikacije.

3. DETEKCIJA OBJEKATA METODAMA DUBOKOG UČENJA

Detekcija objekata je metoda koja koristi računalni vid i kojom se nastoji prepoznati položaj nekakvog objekta u nekoj slici ili videu te zatim taj objekt se nastoji označiti i klasificirati. Kod ljudi taj proces je prirodan, odnosno čovjek će taj proces izvršiti bez puno razmišljanja, dok kod računala je to već podosta izazovno. Razvojem metoda dubokog učenja u zadnjem desetljeću dolazi do velikog napretka u ovom području. Detekcija objekata svoju primjenu nalazi u robotici, anotaciji slika, nadzornim sustavima, autonomnim vozilima i mnogim drugim područjima. Razlika između klasične klasifikacije i detekcije objekta prikazana je na Slici 3.1.

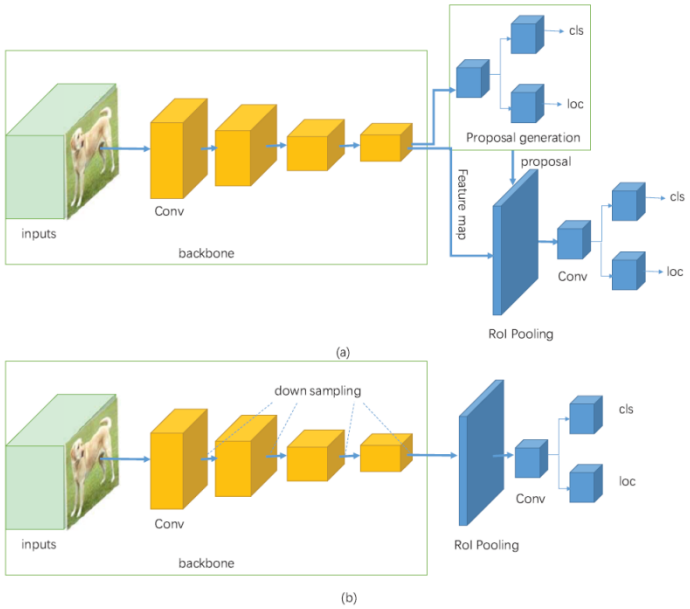


Slika 3.1. Razlika između klasifikacije i detekcije objekta sa slike [autor]

3.1.1. Jednostupanjske i dvostupanjske mreže

Metode koje se najčešće koriste za detekciju objekata su metode zasnovane na CNN-u te se mogu podijeliti u dvije klase: jednostupanjske (eng, *one-stage*) i dvostupanjske (eng. *twostage*) metode. U osnovi jednostupanjske metode istovremeno pronalaze lokaciju objekta i klasificiraju ga, dok dvostupanjske metode proces detekcije izvršavaju kroz više faza. Mreži se daje slika za koju se prvo izdvajaju prijedlozi mogućih objekata, a nakon toga se ti prijedlozi

klasificiraju u specifične kategorije objekata uz pomoć istreniranog klasifikatora. Dvostupanjske metode poznatije su po boljoj preciznosti nego jednostupanjske metode, te neki od najpoznatiji primjeri ovih struktura su : RCNN, SPPnet, Fast RCNN i Faster RCN. Jednostupanjske poznatije su po većoj brzini detekcije te jedan od poznatih primjera je YOLO model. Shematski prikaz arhitektura jednostupanjskih i dvostupanjskih mreža prikazan je na Slici 3.2.



Slika 3.2. Arhitektura dvostupanjske a) i jednostupanjske b) mreže [26]

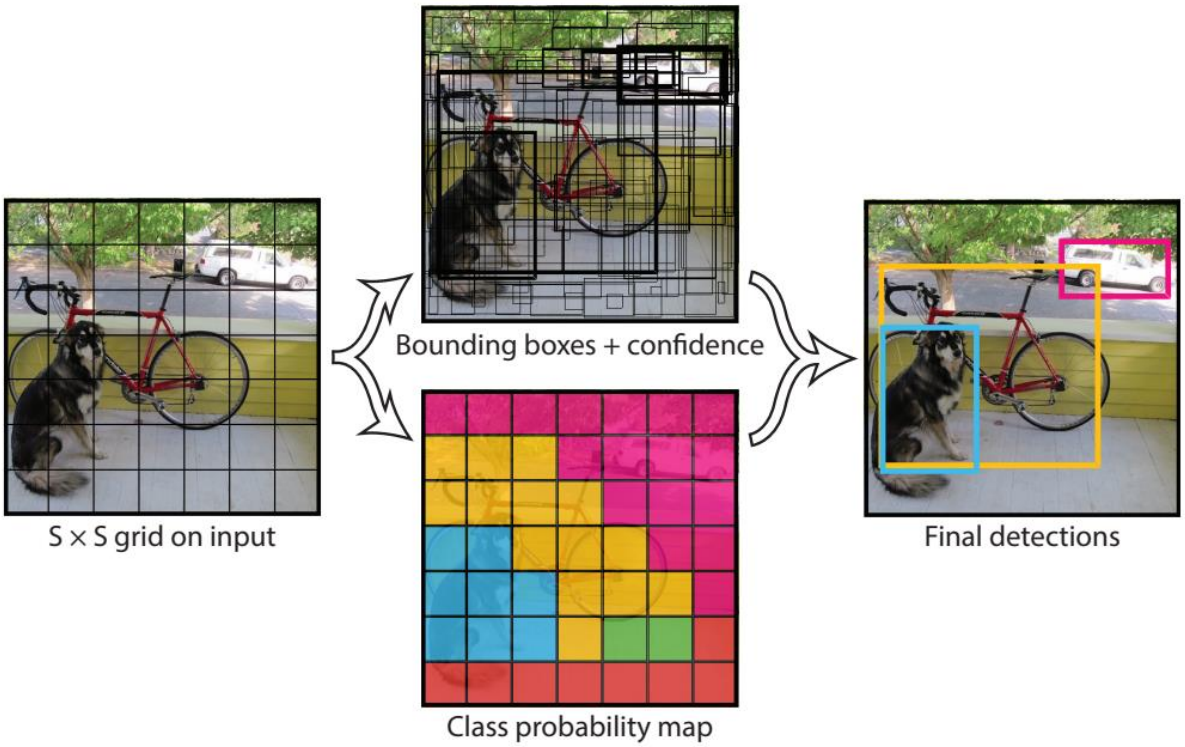
3.2. YOLOv5 (You Only Look Once) model

YOLOv5 (eng. *You Only Look Once*) je familija jednostupanjskih arhitektura namijenjena detekciji objekata u stvarnome vremenu razvijena od strane Ultralytics-a [27]. Ova verzija algoritma temeljena je na izvornom YOLO algoritmu razvijenom od strane Joseph Redmona [28], te je trenutno i jedna od najpoznatijih i najkorištenijih arhitektura.

3.2.1. Princip rada YOLO algoritma

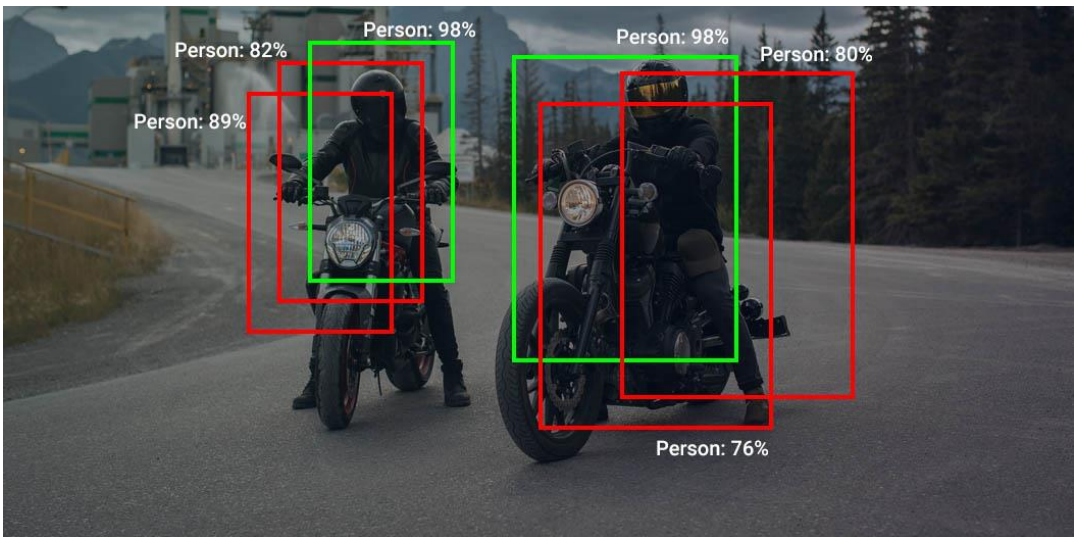
YOLO algoritam radi na principu tako da ulaznu sliku dijelu u matricu sa N ćelija jednakih dimenzija. Svaka od tih N ćelija odgovorna je za detekciju i lokalizaciju objekta kojeg ona sadrži. Sukladno tome, uz samu lokalizaciju objekta, te ćelije također vrše i klasifikaciju objekta unutar njih te samim time kao izlazni rezultat dobije se položaj objekta unutar čelije i

njegovu predviđenu klasu sa vjerojatnošću. Na Slici 3.3. prikazan je princip rada YOLO algoritma.



Slika 3.3. Princip rada YOLO algoritma [28]

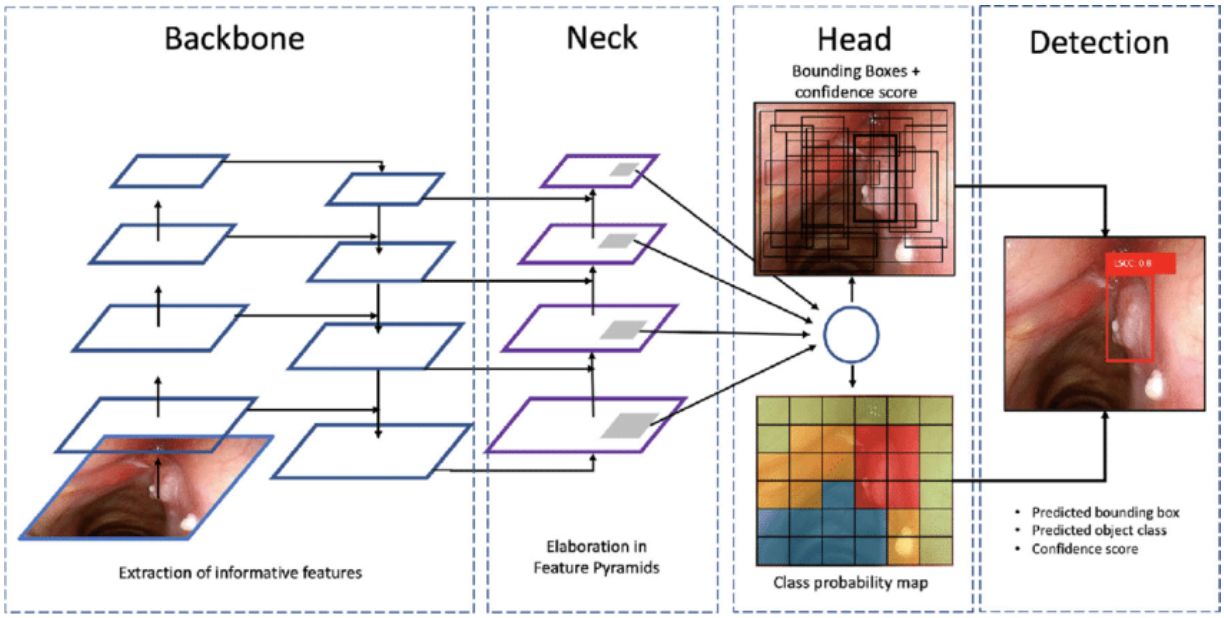
Ovaj proces uvelike smanjuje proračun budući da i detekcijom i klasifikacijom objekta upravljaju same ćelije zasebno, ali donosi mnogo dvostrukih predviđanja radi većeg broja ćelija koje predviđaju isti objekt sa različitim predviđanjima graničnog okvira. Kako bi se riješio dvostrukih okvira, YOLO koristi algoritam poznat pod nazivom Non Maximal Supression (NMS) Koristeći NMS, YOLO potiskuje sve granične okvire koji imaju niže rezultate vjerojatnosti. To postiže ako što prvo gleda na rezultate vjerojatnosti povezane sa svakom detekcijom i uzima granični okvir koji ima najveću vjerojatnost. Nakon toga, on potiskuje granične okvire koji imaju najveći presjek s trenutnim graničnim okvirom visoke vjerojatnosti. Ovaj korak se ponavlja dok se ne dobiju konačni granični okviri. Na Slici 3.4. prikazan je primjer djelovanja NMS algoritma te se može primijetiti kako su odabrani granični okviri sa najvišom vjerojatnošću (zeleni okviri), dok okviri koji sječu te okvire i imaju nižu vjerojatnost su potisnuti (crveni okviri).



Slika 3.4. Primjer rada Non Maximum Supression algoritma [29]

3.2.2. Arhitektura YOLOv5 algoritma

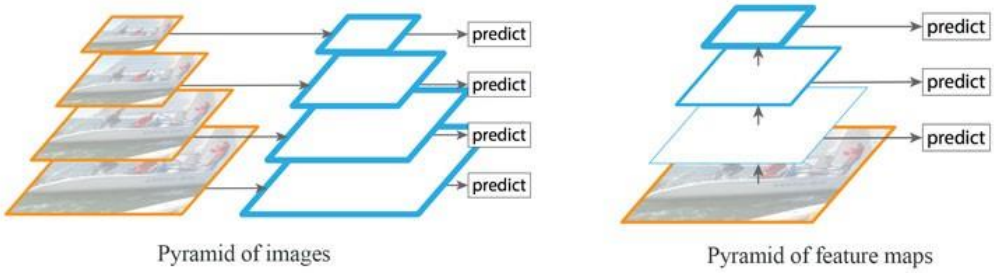
Arhitektura YOLO algoritma sastoji se od tri glavna dijela: Backbone-a. Head-a i Neck-a. Arhitektura YOLOv5 modela prikazana je na Slici 3.5.



Slika 3.5. Arhitektura YOLOv5 algoritma [30]

Backbone sloj se uglavnom koristi za izdvajanje važnih značajki iz dane ulazne slike. U YOLOv5 kao Backbone sloj koristi se Cross Stage Partial Network (CSPNet) [31]. CSPNet pokazao je značajno poboljšanje brzine vremena obrade sa deeplearning modelima.

Neck sloj služi za izvlačenje mapi značajki, točnije u ovom slučaju piramida značajki (eng. *Feature pyramids Networks (FPN)*). FPN pomažu modelima da dobro generaliziraju objekte pri skaliranju podataka, tj. pomažu identificirati isti objekt s različitim veličinama i omjerima. Detekcija objekata u raznim omjerima je dosta izazovan zadatak, pogotovo za male objekte. Jedno od rješenja je da se koristi piramidu ulazne slike u različitim veličinama (rezolucijama) za otkrivanje objekata (Slika 3.6. lijevo). Međutim, obrada slika u više rezolucija oduzima puno memorijskog resursa i vremena što nije dobro za treniranje modela. Alternativno tome može se stvoriti FPN (Slika 3.6. desno) i koristi se za detekciju objekata [32].



Slika 3.6. Piramida ulaznih slika raznih veličina (lijevo) i FPN (desno) [32]

FPN generira višestruke slojeve karte značajki (u više veličina) s informacijama koje su bolje kvalitete nego uobičajene piramide značajki za otkrivanje objekata. U YOLOv5 algoritmu PANet [33] je korišten kao Neck sloj za generiranje FPN-ova.

Head sloj se uglavnom koristi za izvođenje konačnog dijela detekcije. Kao izlaz se generiraju konačni izlazni vektori s vjerojatnosti klasa i graničnim okvirima. Head slojevi mogu biti jednostupanjski ili dvostupanjski.

YOLOv5 arhitektura sadrži pet modela, počevši od YOLOv5 nano (najmanji i najbrži model) pa sve do YOLOv5 extra-large (najveći model). Svih pet modela najprije se razlikuju po svojoj kompleksnosti, broju parametara, preciznosti i brzini. Pregled YOLOv5 modela dan je u Tablici 3.1.

Tablica 3.1. Pregled YOLOv5 modela [34]

Ime modela	Broj parametara ($x10^6$)	Preciznost (mAP@0.5)	Brzina izvođenja na procesoru (ms)	Brzina izvođenja na grafičkoj kartici (ms)
YOLOv5n	1.9	45.8	45	6.3
YOLOv5s	7.2	56.8	98	6.4
YOLOv5m	21.2	64.1	224	8.2
YOLOv5l	46.5	67.3	430	10.1
YOLOv5x	86.7	68.9	766	12.1

3.3. Mjere kvalitete dobivenog rješenja

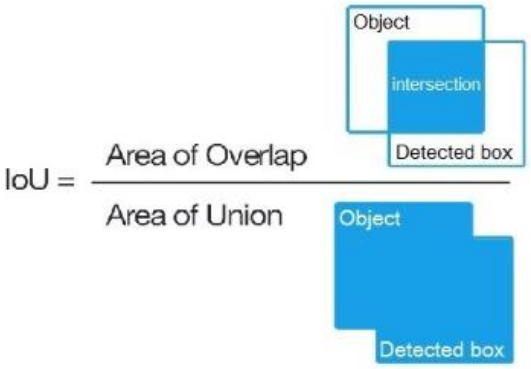
Nakon odrađenog treninga potrebno je ocijeniti model, te će metrike kojima se koristimo biti opisane u ovom dijelu rada. Na nižoj razini, procjena performansi detektora objekata svodi se na određivanje je li detekcija točna ili ne. Prije nego se počnu objašnjavati metrike točnosti potrebno je definirat slijedeće termine:

- **TP** – (eng. T*rue Positive*) – ishod gdje model ispravno predviđa pozitivnu klasu,
- **FP** - (eng. *False Positive*) – ishod gdje model netočno predviđa pozitivnu klasu,
- **FN** – (eng. *False Negative*) – ishod gdje model netočno predviđa negativnu klasu i
- **TN** – (eng. *True Negative*) – ishod gdje model ispravno predviđa negativnu klasu [35].

Pod pozitivnom klasom smatra se ona klasa koja je od interesa, npr. ukoliko treniramo model za detekciju pješaka, pješaci su onda pozitivne klase dok su sve ostale klase negativne (npr. automobili, prometni znakovi i dr.). Slijedeća mjera koju je potrebno definirati je Interesction over Union (IoU). IoU procjenjuje stupanj preklapanja između predviđenog graničnog okvira i graničnog okvira koji se smatra točnom predikcijom, te je definirana slijedećim izrazom:

$$IoU = \frac{područje\ preklapanja}{područje\ unije} \qquad (3.1.)$$

Primjer izračuna IoU prikazan je na Slici 3.7.



Slika 3.7. Računanje IoU [36]

IoU poprima vrijednosti između nula i jedan, gdje nula predstavlja nikakvo preklapanje, a predstavlja potpuno preklapanje između dva granična okvira Ova mjera je korisna kroz postavljanje praga (eng. *treshold)*, npr. potreban nam je prag (α. recimo) i korištenjem tog praga možemo odlučiti je li detekcija koju je model napravio ispravna ili ne. Za IoU vrijednost na vrijednosti praga α, TP je detekcija za koju vrijedi:

$$IoU\left(bbx_{stvarni}, bbx_{predviđemi}\right) \geq \alpha \qquad (3.2.)$$

gdje je:

$bbx_{stvarni}$ – granični okvir koji se smatra točnom predikcijom,

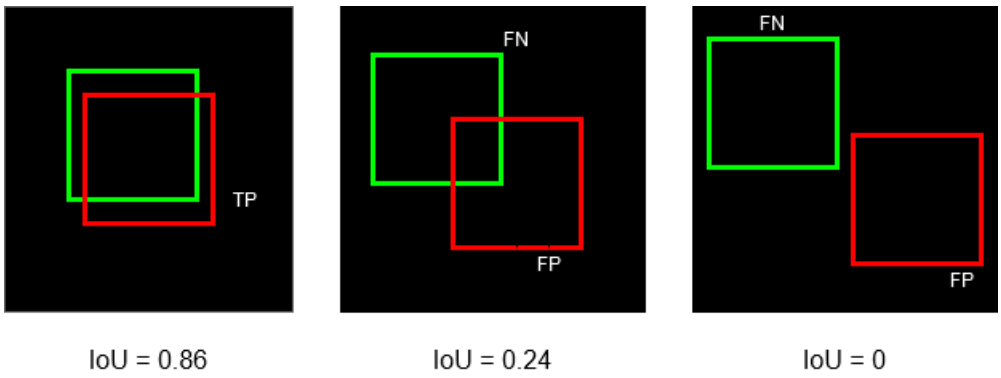
$bbx_{predviđeni}$ – predviđeni granični okvir i

α – vrijednost praga [35].

Dok bi za FP predikciju vrijedilo:

$$IoU\left(bbx_{stvarni}, bbx_{predviđemi}\right) < \alpha \qquad (3.3.)$$

Parametar α poznat je još kao i *„IoU treshold“.* Na Slici 3.8. prikazani su primjeri za slučajeve TP, FP i FN predikciju uzastopno za parametar α = 0.5.



Slika 3.8. Primjeri TP, FP i FN detekciju za α = 0.5 [35]

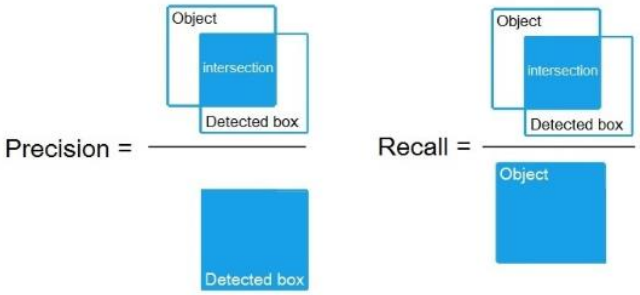
Slijedeće dvije mjere koje će biti definirane su Precision i Recall. Precision je stupanj točnosti modela i detektiranju samo relevantnih objekata, tj. to je omjer TP predikcija u svim detekcijama koje model napravio. Precision je definirana slijedećim izrazom:

$$Precision = \frac{TP}{TP+FP} \qquad (3.4.)$$

Recall odgovara na slijedeće pitanje: „Koji udio stvarnih TP predikcija je točno identificiran?“, te je definiran slijedećim izrazom:

$$Recall = \frac{TP}{TP+FN} \qquad (3.5.)$$

Za model se kaže da je dobar ako ima visoke Precission i Recall metrike. Idealni model ima nula FN predikcija i nula FP predikcija, što znači da vrijedji: Precission = 1 i Recall = 1. [23] Grafički prikaz računanja Precision i Recalla prikazan je na Slici 3.9.



Slika 3.9. Izračuni Precision i Recall parametra [36]

Prosječna preciznost (eng. „*Average Precision - AP*“) ili AP@α je površina ispod Precision-Recall (PR) krivulje evaluirana na vrijednosti praga α. Matematički se definira slijedećim izrazom:

$$AP@\alpha = \int_0^1 p(r)dr \qquad (3.6.)$$

gdje je:

AP@α – prosječna preciznost,

α – vrijednost praga i

p(r) – funkcija P-R krivulje [35].

AP@α znači da se prosječna preciznost procjenjuje na vrijednosti praga (IoU treshold) jednakom α. Ukoliko se nalaze mjere zapisane kao *AP50 ili AP75*, one samo znači da se AP računa za vrijednosti *IoU = 0.5* odnosno *IoU = 0,75* [35] AP se računa pojedinačno za svaku klasu. To znači da postoji onoliko vrijednosti *AP-a* koliko je i klasa. Kako bi se dobila nekakva procjena preciznosti modela sveukupno za sve klase, računa se srednja vrijednosti *AP*-ova svih klasa i ta metrika se naziva srednja prosječna preciznost (eng. „*Mean Average Precision - mAP*“). *mAP* definirana je slijedećim izrazom i jedna je od glavnih metrika kojom se ocjenjuju performanse modela:

$$mAP@\alpha = \frac{1}{n} \sum_{i=1}^{n} AP_i \qquad (3.7.)$$

gdje je:

mAP – srednja prosječna preciznost,

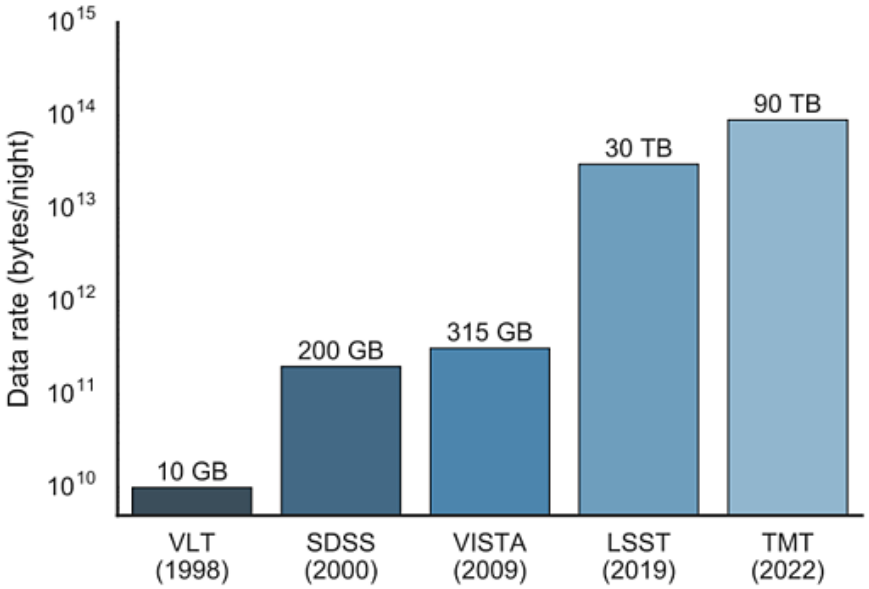
n – broj klasa i

AP_i – prosječna preciznost *i*-te klase [35].

4. UMJETNA INTELIGENCIJA U ASTRONIMIJI

4.1. Problem velike količine podataka

Astronomija se posljednjih godina stabilno razvija, a generirani podaci također se šire kako u raznolikosti tako i u složenosti. Od poboljšanih metoda sakupljanja podataka, do metoda zasnovanim na strojnom učenju, astronomi imaju više pristupa podacima nego ikad prije. „Prikupljanje astronomskih podataka nekada je bilo heterogeno, oslanjajući se na pojedine astronome koji su zahtijevali teleskope i snimali dijelove neba posebno za svoje istraživanje. Gledajući unazad 20 godina, pokušaji kao što su Sloan Digital Sky Survey (SDSS) [37], Pan-STARRS [38] i Large Synatopic Survey Telescope (LSST) [39] preusmjerili su se s individualiziranog prikupljanja podataka na proučavanje većih dijelova neba i širih raspona valnih svjetlosti, tj. prikupljanje podataka o više događaja, objekata i pokrivanje veće površine neba kroz veće površine i bolje detektore svjetla“ [40]. SDSS jedan je od najvećih astronomskih istraživanja današnjice. Svake noći SDSS teleskop prikupi 200 GB podatka, dok postoje i teleskopi koji prikupljaju do 90 terabajta podataka svake noći (Slika 4.1.) [41].



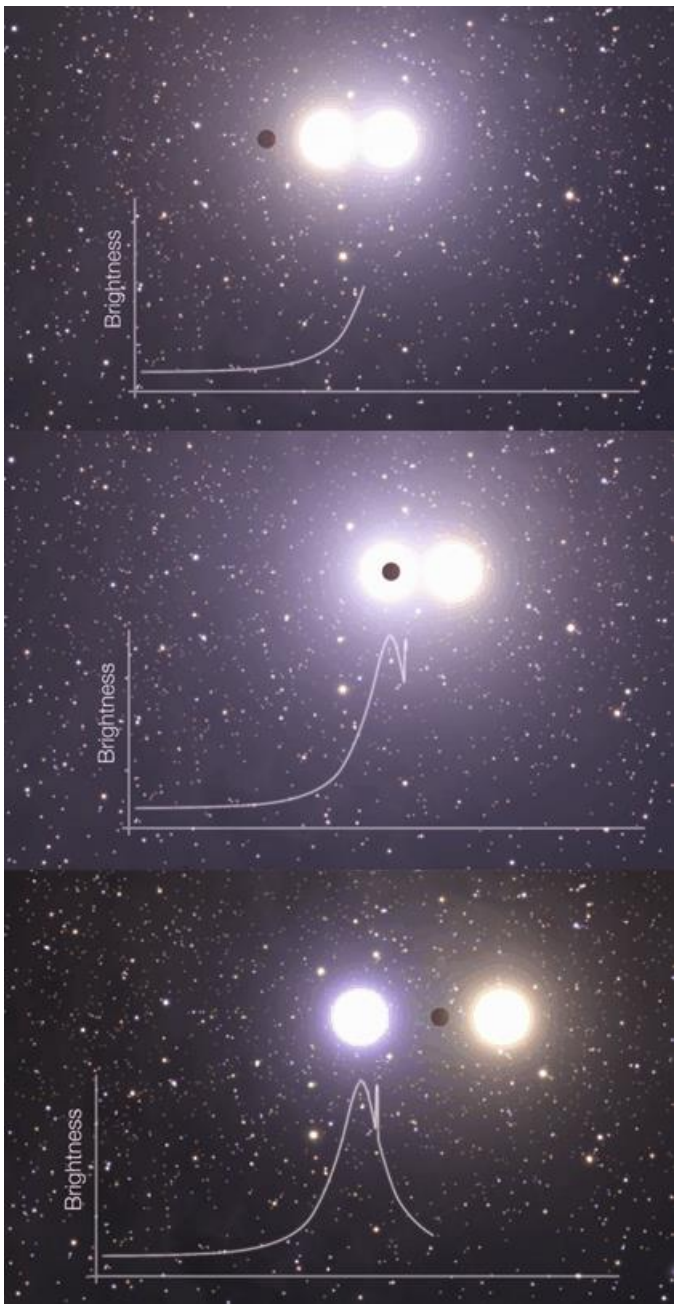
Slika 4.1. Povećanje količine podataka postojećih i nadolazećih teleskopa [41]

Radi enormne količine prikupljenih podataka, astronomi se bore s više od 100 do 200 petabajta podataka godišnje, što zahtjeva golemu infrastrukturu za pohranu. Stoga je ključno obraditi

ove ogromne prikupljene podatke. Takav eksponencijalan porast podatka pruža idealnu priliku za primjenu umjetne inteligencije i strojnog učenja kao pomoć u obradi svih ovih podataka.

4.2. Primjena umjetne inteligencije u astronomiji

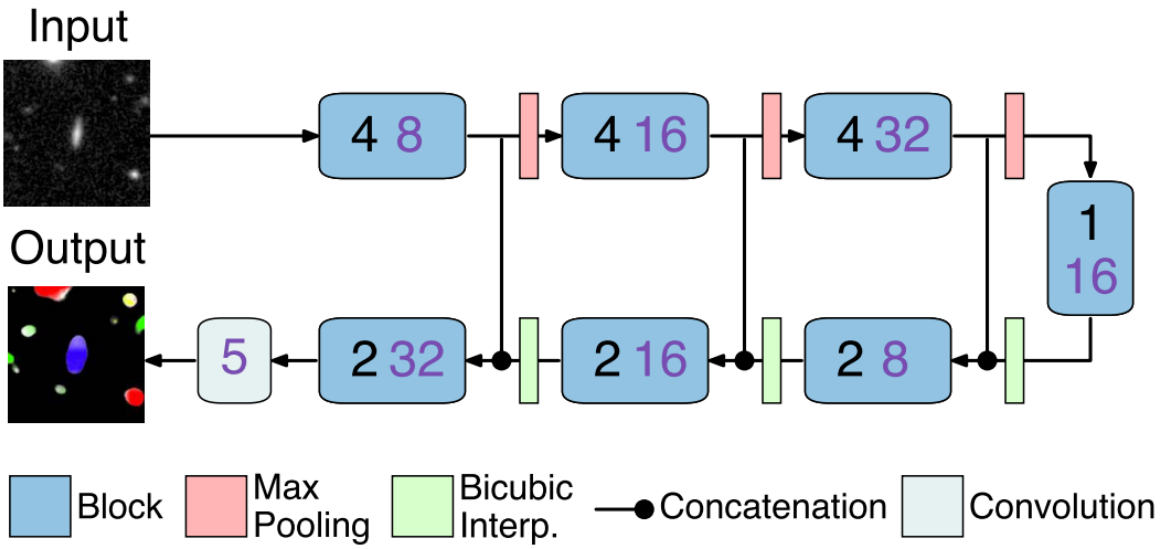
S razvojem računalnih sposobnosti, metode strojnog učenja dobile su veliki napredak u izdvajanju značajki, kompresiji podatka i prostornom preslikavanju. Postoji mnogo kategorija metoda strojnog učenja, uključujući neuronske mreže, neizrazite (eng. *fuzzy*) logičke aproksimacije, genetske algoritme i dr. U usporedbi s tradicionalnim metodama, strojno učenje ima slijedeće prednosti: kao prvo to je metoda vođena podacima koja je prikladna za obradu velikih količina podataka u astronomskim istraživanjima; kao drugo, može dobro izvući značajke samih podataka s malo ručnih intervencija; kao treće, ima dobru sposobnost prijenosa znanja. Jedno područje u kojem je umjetna inteligencija značajno utjecala je potraga za egzoplanetima. Postoji mnogo načina za traženje njihovih signala, ali najproduktivnije metode s trenutnom tehnologijom obično uključuju proučavanje varijacija sjaja zvijezde tijekom nekog vremenskog perioda. Ako krivulja svijetlosti zvijezde pokazuje karakteristično zatamnjenje, to bi mogao biti siguran znak planeta ispred te zvijezde. Suprotno tome, fenomen koji se naziva gravitacijska mikroleća može uzrokovati veliki skok u svjetlini same zvijezde, kada gravitacija planeta egzoplaneta djeluje kao leća koja povećava udaljeniju zvijezdu duž vidne linije. Detektiranje ovih padova i skokova znači pretraživati milijune krivulja svjetlosti, pažljivo prikupljenih svemirskim teleskopima poput NASA-inog Keplera i TESS-a [42]. Na Slici 4.2. prikazan je primjer podataka koji je gore opisan. Egzoplanet koji se ponaša kao mikroleća, uzrokuje skok u svjetlini zvijezde, što mogu detektirati ljudi ili algoritmi. Koristeći ogromne količine podataka promatranih krivulja svjetlosti, astronomi su uspjeli razviti modele temeljene na strojnom učenju koji mogu nadmašiti ljude u potrazi za egzoplanetima.



Slika 4.2. Skok svjetlosti zvijezde kod efekta mikroleće [42]

Proučavanje vrsta i svojstava galaksija važno je jer pruža važne tragove o podrijetlu i nastanku svemira. Klasifikacija samih galaksija važna je uloga u proučavanju nastanka galaksija i procjene svemira u kojem se nalazimo. Morfološka klasifikacija je sustav koji se koristi za podjelu galaksija u skupine na temelju njihovog vizualnog izgleda. Povijesno, klasifikacija galaksija vršila se vizualnom inspekcijom dvodimenzionalnih slika galaksija te ručnim klasificiranjem istih. No enormnim porastom količine podataka u današnjici, gotovo je nemoguće ručno klasificirati sve galaksije. Time razvijeni su sustavi kao što je *Morpheus* [43], koji služi za klasifikaciju galaksija i zvijezda sa velikom točnošću u velikim skupovima

podataka. *Morpheus* je model temeljen na dubokom učenju (eng. *deep-learning*) koji uključuje razne tehnologije umjetne inteligencije razvijene za aplikacije kao što su prepoznavanje slike i govora. Prethodna istraživanja obično su uključivala prilagodbu postojećih algoritama za prepoznavanje slika, a istraživači su algoritmima dodavali slike galaksija koje treba klasificirati. *Morpheus* model kao ulaz također koristi podatke izvornih astronomskih slika u standardnom formatu (kao što su *jpeg* i sl.). No za razliku od klasičnih algoritama, klasifikacija u navedenom modelu se odvija „piksel po piksel“ (eng. *Pixel-level classification*), tj. kod obrade slike područja neba, generira se novi skup slika tog dijela neba u kojem su svi objekti označeni bojama na temelju njihove morfologije, odvajajući astronomske objekte od pozadine i identificirajući točkaste izvore (zvijezde) i različite vrste galaksija. Izlaz uključuje razinu pouzdanosti za svaku klasifikaciju (Slika 4.3.) [44].



Slika 4.3. Arhiktektura Morpheus modela [43]

Gore navedeni primjeri samo su od nekih primjera primjene umjetne inteligencije u astronomiji. Umjetna inteligencija svoju primjenu nalazi još u mnogim slučajevima, neki od slučajeva su: upravljanje roverima za obilaženje prepreka, asistiranje astronautima u svakodnevnim zadacima na Internacionalnoj svemirskoj stanici (eng *International Space Station (ISS)*), detekcija zakopanih arheoloških ostataka na satelitskim snimkama i još mnogo drugih primjena [45].

4.3. Budućnost primjene umjetne inteligencije astronomiji

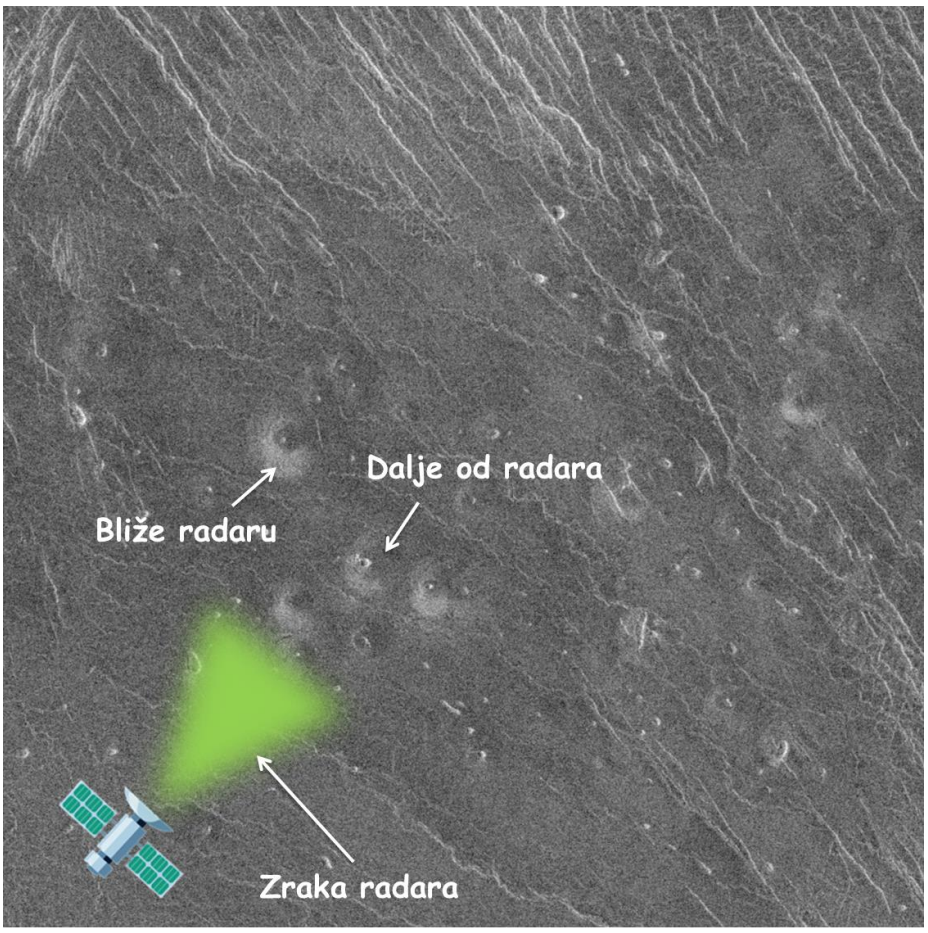
„U bližoj budućnosti, analiza slike i sustavi strojnog učenja koji mogu obraditi terabajte podataka u gotovo stvarnom vremenu s visokom točnošću biti će nužni. Postoje velike mogućnosti za nova otkrića, čak i u skupovima podataka koji su dostupni već desetljećima“ [41]. Dan danas je pregledan samo djelić slika sa istraživanja kao što su *SDSS* od strane ljudi i bez sumnje tavi podaci još uvijek kriju mnogo budućih otkrića za nadolazeća istraživanja. Tolike količine podataka neće biti moguće ručno pregledati te su zato napredne analize slika i algoritama strojnog učenja od velike važnosti. Takvi sustavi mogu biti korišteni za odgovaranje na pitanja poput: koliko vrsta galaksije postoje?, što razlikuje različite klase?, je li trenutni klasifikacijski način dovoljno dobar? i postoje li važne potklase ili neotkrivene klase? Ova pitanja zahtijevaju znanost o podacima, radije nego astrofizička znanja, no otkrića će i dalje biti od uvelike pomoći u samoj grani astrofizike. U ovoj novoj eri bogatoj podacima, astronomija i informatika mogu imati uvelike koristi jedno od drugog. Postoje novi problemi s kojima se treba pozabaviti, nova otkrića do kojih treba doći i nova znanja koja treba steći u oba polja. Stoga kako bi se postigao daljnji napredak, potreban je zajednički napredak obje grane znanosti.

5. MAGELLAN SET PODATAKA I PRIPREMA

Eksperimenti s tehnologijom letenja jedan su od načina na koji *NASA* teži novim sposobnostima koje su joj potrebne da pomakne granice istraživanja svemira. Ove usmjerene aktivnosti dovode do napretka sposobnosti svemirskih letjelica, istražujući njihov potencijal i vrijednost. Jedan od takvih primjera bila je svemirska letjelica Magellan, lansirana je 1989. godine, a pristigla je u orbitu Venere 1990. Tokom svojih četiri godine u orbiti oko sestrinskog planeta Zemlje, Magellan je pomoću sintetičkog radara (eng. *Synthetic Aperture Radar - SAR*) zabilježio 98% površine Venere te prikupio gravitacijske podatke samog planeta u visokoj rezoluciji [46]. Tokom Magellanovog aktivnog djelovanja prikupljeno je oko 30000 slika površine Venere na kojoj se ujedno nalaze i brojni vulkani, no ručno označavanje tih slika je vremenski iscrpno i dosad je učinjeno samo za 134 slike od strane geoloških stručnjaka [47]. Smatra se da na površini Venere se nalaze više od 1600 velikih vulkana čije su značajke dosad poznate, ali njihov broj može biti preko sto tisuća ili čak preko milijun [48]. Upravo radi toga pristupilo se izradi alata pomoću algoritama strojnog učenja koji bi služio kao pomoć pri označavanju tolike količine podataka.

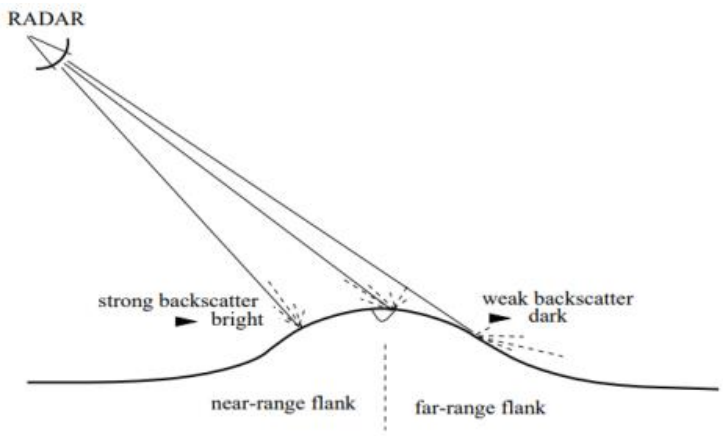
5.1. Opis seta podataka

Magellan set podataka [49] sastoji se od 134 slike dimenzija 1024x1024 piksela koje predstavljaju površinu planeta Venere. Slika 5.1. prikazuje 30km x 30km površine Venere snimljene od strane svemirske letjelice Magellan. Vulkani na površini planeta mogu se prepoznati po svojem takozvanom „radarskom potpisu“ koji je produkt SAR radara, tj. strana vulkana koja je okrenuta prema radaru je svjetlija, dok je suprotna strana vulkana tamnija.



Slika 5.1. Slika površine Venere sa vidljivim vulkanima i primjerom radarskog potpisa [autor]

Razlog tome je taj što je bliža strana objekta (u ovom slučaju vulkana), koja je okrenuta u smjeru radara, odbija veću količinu energije nazad prema senzoru radara, dok udaljenija (tamnija) strana koja je suprotno okrenuta raspršuje zraku u okolni prostor. Svjetlina svakog piksela proporcionalna je logaritmu odbijene energije nazad prema senzoru. Prema tome tipični „radarski potpis“ vulkana pojavljuje se kao „svjetlo-tamni“ u obliku kružnog obrisa sa svijetlim pikselima u središtu [47]. Ta opisana pojava ilustrirana je na Slici 5.2. Svjetliji pikseli u samome središtu vulkana obično se pojavljuju radi toga što se zraka sa SAR-a raspršuje unutar samog kratera. No ukoliko je krater manji naspram rezolucije slike, moguće da se ta značajka neće ni pojaviti ili se pojavljuje samo kao svijetla točka. Gore opisane pojave samo su neke od značajki prema kojima su stručnjaci klasificirali vulkane na površini planeta



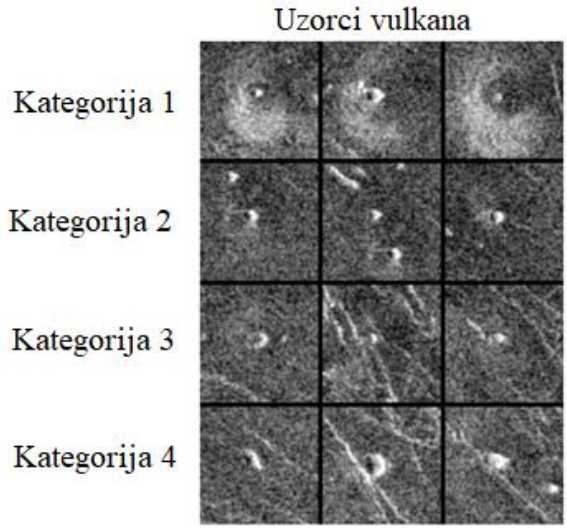
Slika 5.2. Odbijanje radarske zrake od vulkana [47]

Zbog same prirode šumovitosti slika SAR-a, čak ni stručnjaci ne mogu sa 100%-tnom sigurnošću odrediti ukoliko je jedan od objekata na slici zapravo vulkan ili ne. Nažalost ovaj problem javlja se zbog raznih utjecajnih faktora kao što su npr. rezolucija slike ili odnos signal šum (eng. *Signal to Noise Ratio - SNR*). Različiti stručnjaci za istu sliku će istim vulkanima pridodijeliti različite klase. U zajedničkom dogovoru stručnjaci su razvili pet klasa vulkana koje se koriste za strojno učenje u ovom radu, a one su:

- **Klasa 1 –** „*gotovo sigurno vulkan* " ($p \approx 0.98$); slika jasno prikazuje „svjetlo-tamni par" i kružni obris u središtu,
- **Klasa 2 –** „*najvjerojatnije vulkan*" ($p \approx 0.80$); slika prikazuje dvije od tri značajki,
- **Klasa 3 – „***moguće da je vulkan***"** ($p \approx 0.60$); slika prikazuje „svijetlo-tamni par", no kružni obris nije vidljiv,
- **Klasa 4 –** „*krater*" ($p \approx 0.5$); slika prikazuje vidljiv krater, no nema ostalih naznaka i
- **Klasa 5 –** *„nije vulkan"* ($p \approx 0$) [47].

Gdje slovo *p* označava vjerojatnost pojave vulkana.

Gore navedene klase prikazane su na Slici 5.3.



Slika 5.3. Definirane klase vulkana u setu podataka [47]

5.2. Priprema seta podataka

Sve slike iz seta podataka dolaze u dvije datoteke različitih formata (*.sdt* i *.spr*) koje opisuju sliku. *.spr* datoteka je takozvani *header file* koji sadrži značajke koje opisuju sliku (npr. dimenzije slike), dok *.sdt* datoteka sadrži binarni zapis same slike. Primjer jedne *.spr* datoteke prikazan je na Slici 5.4.

```
1  2              #2D matrica
2  1024           #Sirina slike
3  0.000000
4  1.000000
5  1024           #Visina slike
6  0.000000
7  1.000000
8  0              #tip podatka
```

Slika 5.4. Primjer .spr datoteke [autor]

Radi toga dolazi do potrebe prebacivanja slika u format koji je prigodan za učenje YOLO algoritma (npr. *.jpeg* ili *.png* format). Tim povodom razvijena je *vread.py* skripta (Dodatak A) čija je osnovna funkcija u Python kodu prikazana dole. Njezina svrha je učitavanje slika iz danih dviju datoteka u Python okruženje, te vraćanje istih u *.png* formatu koji je pogodan za učenje algoritma.

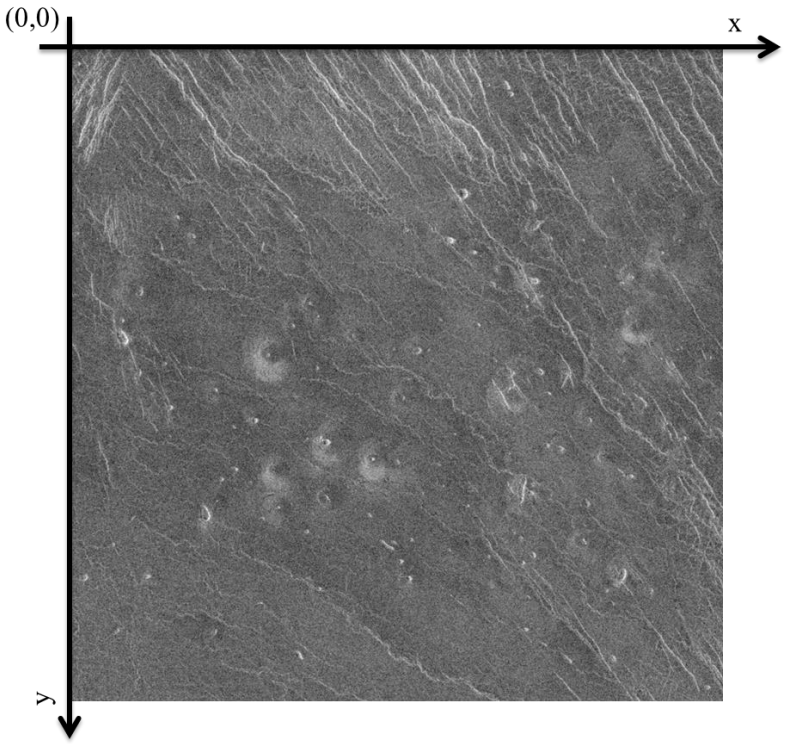
```
def vread(filename):

    pfile = filename + '.spr' #header datoteka - sadrzi znacajke slike
    dfile = filename + '.sdt' #sadrzi binarne podatke slike

    #ucitava podatke iz header (.spr) datoteke
    spr_data = []
    with open(pfile) as fobj:
        for line in fobj:
            row = line.split()
            spr_data.append(row[-1])

    #Izvlaci znacajke slike iz header datoteke
    ndim = int(spr_data[0])#1D ili 2D ili 3D...
    nc = int(spr_data[1]) #broj stupaca
    nr = int(spr_data[4]) #broj redaka
    n_type = int(spr_data[7])#tip podataka

    if (ndim != 2):
        print('Can only read two dimensional data!')
        return None

    #ovisno o tipu podataka formira sliku iz .sdt datoteke
    #usigned char
    if (n_type == 0):
        with open(dfile, 'rb') as fid:
            data_array = np.fromfile(fid, np.ubyte).reshape((nr,nc))
    #integer
    elif (n_type == 2):
        with open(dfile, 'rb') as fid:
            data_array = np.fromfile(fid, np.intc).reshape((nr,nc))
    #float
    elif (n_type == 3):
        with open(dfile, 'rb') as fid:
            data_array = np.fromfile(fid, np.single).reshape((nr,nc))
    #double
    elif (n_type == 5):
        with open(dfile, 'rb') as fid:
            data_array = np.fromfile(fid, np.double).reshape((nr,nc))

    else:
      print('Unrecognized data type')
      data_array = 0

    return data_array
```

Svaka slika iz skupa podataka uza sebe sadrži i istoimenu takozvanu *ground-truth*“datoteku koja označava mjesto vulkana na slici i njegovu pripadnost nekoj klasi. *Ground-truth* datoteke nazivaju se također i *labeli*. U Magellan setu podataka, *ground-truth* dolaze u *.lxyr* formatu čija je struktura prikazana na Slici 5.5. . Prva znamenka označava klasu vulkana, druga znamenka označava x-koordinatu ishodišta (u pikselima) kružnice, treća označava y-koordinatu ishodišta kružnice, te posljednja znamenka označava radijus kružnice.

```
1 293.000000 794.000000 21.930000
```

Slika 5.5. Anotacije u Magellan formatu [autor]

Prilikom označavanja slika, koordinatni sustav se nalazi u gornjem lijevom kutu same slike, prikaz primjera koordinatnog sustava slike prikazan je na Slici 5.6.



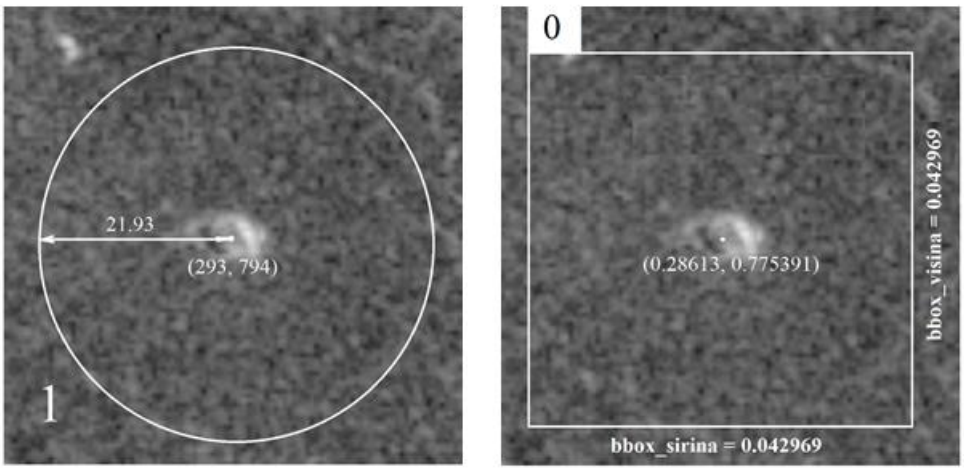
Slika 5.6. Primjer označavanja koordinatnog sustava kod slika [autor]

Primjer *label-a* za YOLO format prikazan je na Slici 5.7. gdje prva znamenka označava klasu vulkana, druga znamenka označava x-koordinatu ishodišta, treća označava y-koordinatu ishodišta, a posljednje dvije znamenke označavaju širinu i visinu graničnog okvira koji opisuje objekt (eng. *bounding box*) normalizirane na vrijednosti [0,1].

```
0 0.286133 0.775391 0.042969 0.042969
```

Slika 5.7. Anotacija u YOLO formatu [autor]

Primjeri razlike između gore dva navedena formata prikazani su na Slici 5.8. Iz slike se može primijetiti osnovna razlika između dva formata anotacije objekata. Magellan format anotacije svoje koordinate središta iskazuje u pikselima, dok kod YOLO formata su iste normalizirane. Druga vidljiva razlika je da kod Magellan formata se za označavanje objekta na slici koristi kružnica, dok kod YOLO formata se za označavanje koristi pravokutnik ili kvadrat. Zadnja ali bitna razlika u anotacijama je ta, da kod YOLO formata oznake pripadnosti klasi startaju od broja nula dok kod Magellan formata startaju od broja jedan.



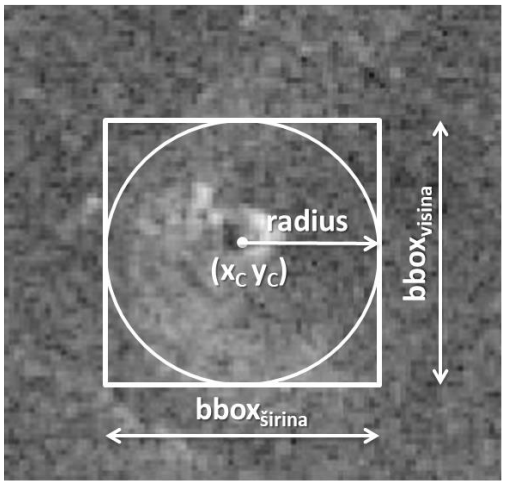
Slika 5.8. Razlika između Magellan (lijevo) i YOLO (desno) anotacija [autor]

Zbog same razlike u anotacijama, došlo je do potrebe da se razvije način na koji će se *labeli* prebaciti iz Magellan formata u *YOLO* format koji je prikladan za učenje. U tu svrhu napravljena je skripta *lxyrConvert2yolo.py* (Dodatak B) čija je glavna funkcija prikazana u slijedećem kodu:

```
def lxyrConvert2Yolo(filename, img_size):
    file = filename + '.lxyr'

    #ignorira "empty text file" upozorenje od np.loadtxt()
    with warnings.catch_warnings():
        warnings.simplefilter("ignore")
        lxyr_data = np.loadtxt(file)
    #vraca None ako nema mogucih vulkana u slici
    if(len(lxyr_data) == 0):
        return None
    else:
        #provjerava broj redova u .lxyr datoteci (broj redaka = broj mogucih
vulkana)

        try:
            nov = (lxyr_data.shape[0], lxyr_data.shape[1])
        except IndexError:
```

```
            nov = (1, lxyr_data.shape[0])

        #cita 1D vektore
        if(nov[0] == 1):
            v_class  = np.array(lxyr_data[0]) #klasa vulkana
            x_cord = np.array(lxyr_data[1]) #x koordinate
            y_cord = np.array(lxyr_data[2]) #y koordinate
            radius = np.array(lxyr_data[3]) #radijus kruga

        #cita 2D vektore
        else:
            v_class  = np.array(lxyr_data[:,0]) #klasa vulkana
            x_cord = np.array(lxyr_data[:,1]) #x koordinate
            y_cord = np.array(lxyr_data[:,2]) #y koordinate
            radius = np.array(lxyr_data[:,3]) #radius kruga

        #pretvara u YOLO format
        #[class, x_center, y_center, width, height] svi normalizirani [0,1]
        #u ovom slucaju -> width = height (bounding box je kvadrat)
        ground_truth = []
        ground_truth = np.append(ground_truth, v_class-1)#YOLO klase startaju od
nule

        x_center = x_cord/img_size[0]
        y_center = y_cord/img_size[1]
        ground_truth = np.c_[ground_truth, x_center]
        ground_truth = np.c_[ground_truth, y_center]

        width = np.empty([nov[0],1])
        height = np.empty([nov[0],1])

        if(nov[0] == 1):
            width[0] = np.round(radius)*2/img_size[0]
            height[0] = np.round(radius)*2/img_size[1]
        else:
            for i in range(0,nov[0]):
                width[i] = np.round(radius[i])*2/img_size[0]
                height[i] = np.round(radius[i])*2/img_size[1]

        ground_truth = np.c_[ground_truth, width]
        ground_truth = np.c_[ground_truth, height]

    return ground_truth
```

Ova funkcija učitava *.lxyr* format datoteke te ih prebacuje u YOLO format na način prikazan na Slici 5.9.

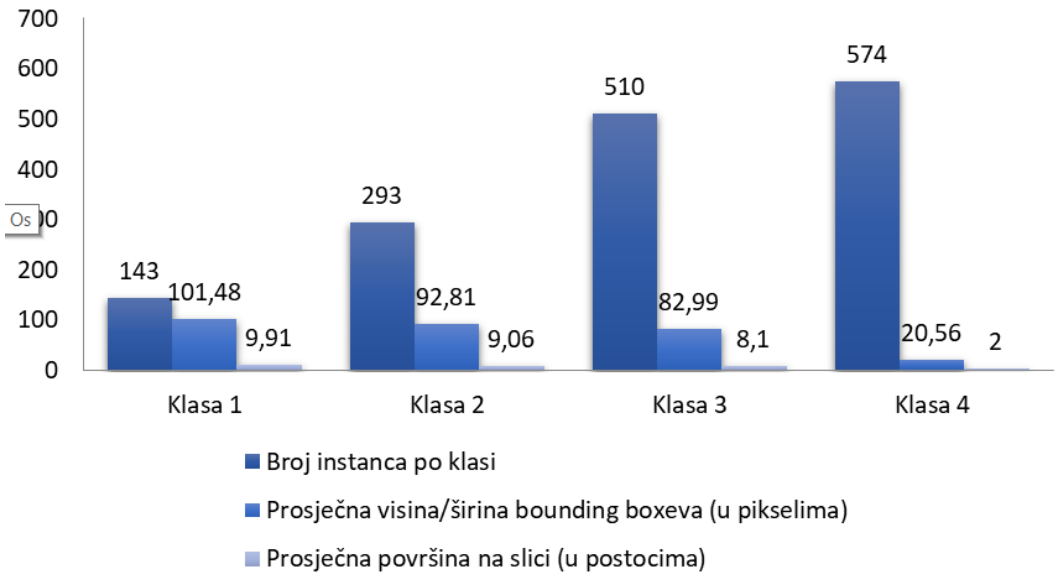


YOLO	⟷	.lxyr
klasa	=	klasa – 1
x_C	=	x_C /(širina slike)
y_C	=	y_C /(visina slike)
$bbox_{širina}$	=	2 × radius /(širina slike)
$bbox_{visina}$	=	2 × radius /(visina slike)

Slika 5.9. .Postupak prebacivanja oznaka iz Magellan formata u YOLO format [autor]

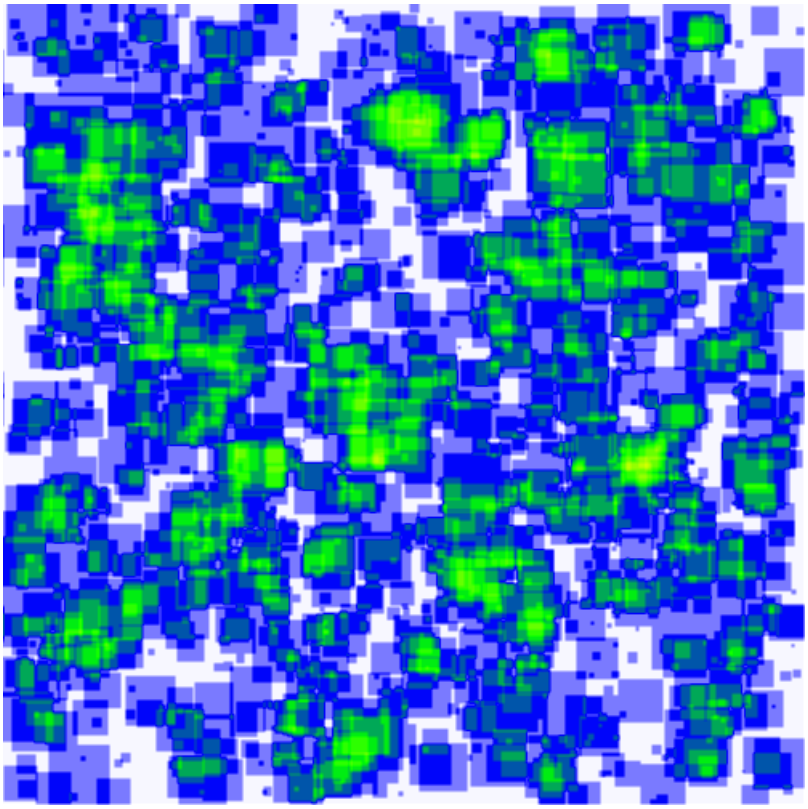
5.3. Analiza seta podataka

Kako je već napomenuto u uvodu ovog poglavlja, Magellan set podataka sastoji od ukupno 134 slike dimenzija 1024x1024 piksela. Set podataka sastoji od relativno malog broja slika, što predstavlja zasebne izazove prilikom uspješnog treniranja modela. Na Slici 5.10. prikazan je grafički prikaz analize seta podataka zasebno za svaku klasu. Prvi stupac prikazuje broj instanca po klasi, tj. broj objekata po klasi u setu podataka. Drugi stupac prikazuje prosječne dimenzije objekta te klase, dok treći stupac prikazuje koliku prosječnu površinu slike ti objekti zauzimaju.



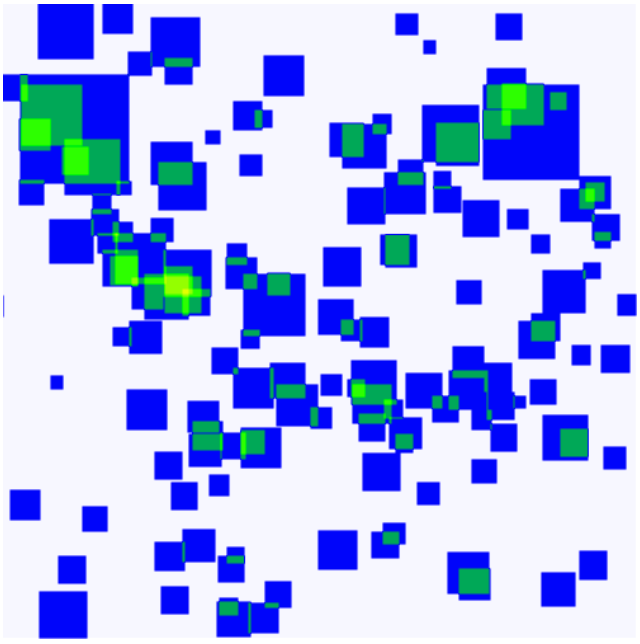
Slika 5.10. Grafikon analize Magellan seta podataka [autor]

Iz grafikona se može zaključiti da je set podataka podosta neizbalansiran, tj. pojedine klase objekta su prezastupljene, dok neke klase su podzastupljene. U Magellan setu podataka nalazi se sveukupno 1520 označenih vulkana, od kojih su 574 njih (37.76%) iz klase četiri te 510 njih (33.55%) iz klase tri, dok iz klase jedan i dva uzastopno ima 143 (9.41%) i 293 (19.28%) vulkana. U drugome slučaju, može se primijetiti kako objekti iz klase jedan su najveće veličine, dok objekti iz klase četiri su jako maleni (≈20x20 piksela) i zauzimaju prosječno 2% površine ukupne slike, što pruža podosta poteškoća u detekciji i klasifikaciji takvih objekta. Na Slici 5.11.prikazana je toplotna mapa (eng. *heat map*) anotacija svih objekata u Magellan setu podataka. Iz Slike 5.11. se može zaključiti kako postoji pokrivenost skoro svih pozicija na slikama iz seta podataka.



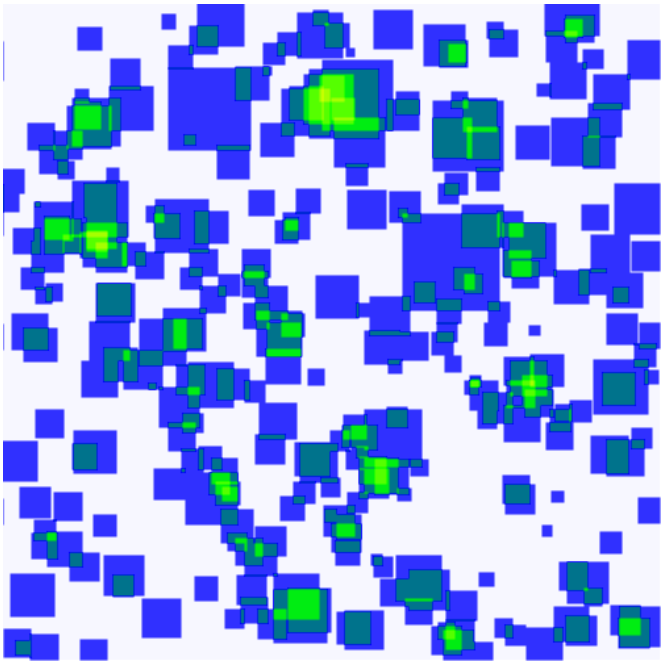
Slika 5.11. Toplotna mapa svih klasa objekata iz Magellan seta podataka [autor]

Slike 5.12 do 5.15 uzastopno. prikazuju toplotne mape svake klase objekta zasebno.



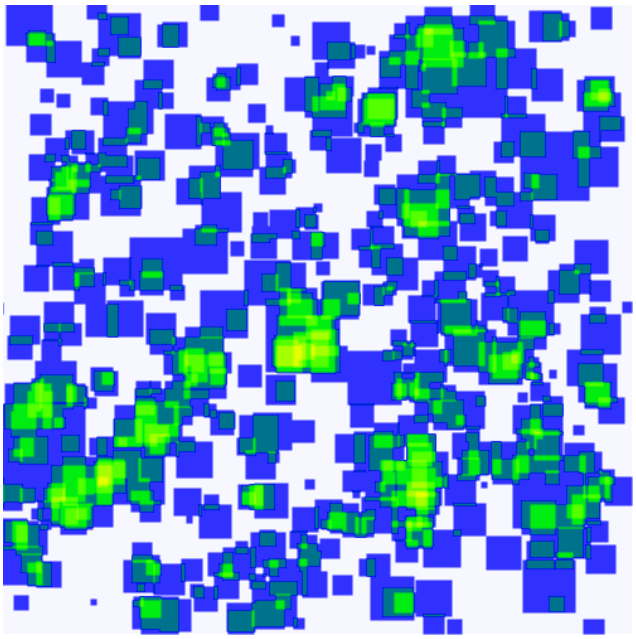
Slika 5.12. Toplotna mapa klase 1 iz Magellan seta podataka [autor]

Promatrajući Sliku 5.12., može se primijetiti kako su vulkani klase 1 većih dimenzija, no u manjem broju (nedovoljno zastupljeni), no njihova veća površina (≈10% ukupne slike) čini ih klasom čije je karakteristike lakše klasificirati i detektirati, te zato daju bolje rezultate nego ostale klase.



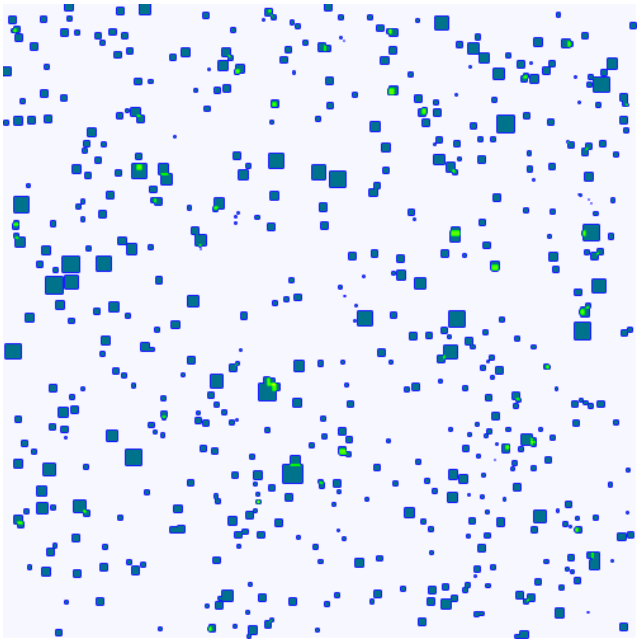
Slika 5.13. Toplotna mapa klase 2 iz Magellan seta podataka [autor]

Iz Slike 5.13., može se zaključiti kako vulkani klase dva imaju slične karakteristike, no veću rasprostranjenost nego klasa 1.



Slika 5.14. . Toplotna mapa klase 3 iz Magellan seta podataka [autor]

Promatrajući Sliku 5.14., usporedno sa Slikama 5.13. i 5.15., može se doći do zaključka da klasa 3 vulkana dijeli sličnosti u karakteristikama između klase 2 i 4 po veličini i rasprostranjenosti, te radi doga dolazi do pogrešnih klasifikacija ove klase.



Slika 5.15. Toplotna mapa klase 4 iz Magellan seta podataka [autor]

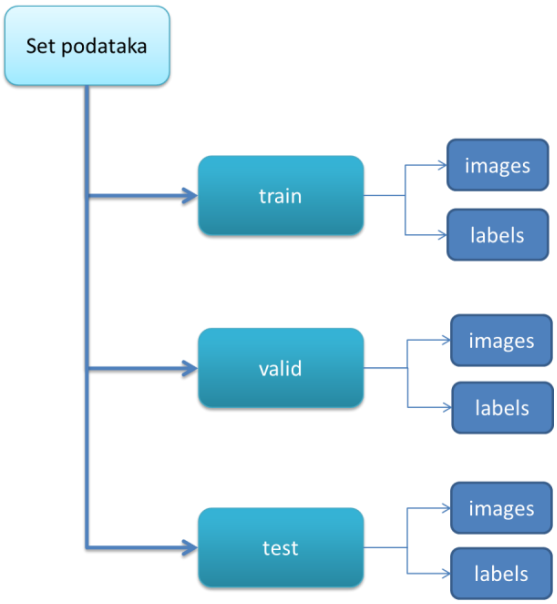
5.4. Podjela seta podataka

Prije početka treniranja modela, set podataka potrebno je nasumično podijeliti na tri skupine:

- Set za učenje,
- Set za validaciju i
- Set za testiranje.

Dio podataka odvojen za učenje je stvarni skup podataka koji koristimo za treniranje modela, tj. model vidi te podatke i uči iz njih. Set podataka za validaciju koristi se za evaluaciju prilikom treniranja određenog modela, tj, koristi se za fino podešavanje hiperparametara modela. Stoga model povremeno vidi ove podatke, ali nikada ne uči iz njih. Dakle set podataka za validaciju utječe na model, ali samo neizravno. Set za testiranje pruža zlatni standard za procjenu modela. Koristi se samo nakon što je model potpuno istreniran, te je testni skup općenito ono što se koristi za procjenu modela. Podjela uglavnom ovisi o dvije stvari. Prva stvar je ukupan zbroj uzoraka u cijelom setu podataka, a druga stvar je u ovisnosti o modelu koji se trenira. Neki modeli trebaju značajne podatke za učenje te se u tom slučaju setu za učenje daje veći broj podataka. Modele s vrlo malo hiperparametara lakše je validirati i podesiti, tako da se vjerojatno može smanjiti veličina seta za validaciju. No ako model ima veliki broj hiperparametara, potrebno je i imati veliki set podataka za validaciju. Neki od standardnih omjera podjele seta podataka su 90:10, 80:20 60:40 te čak 50:50 [50] za setove za učenje i testiranje respektivno. Zatim se nakon toga omjer seta podataka za učenje raspodjeli još na dva zasebna djela (set za učenje i validaciju),

YOLO model koristi strukturu podjele podataka na gore navedene setove koja je prikazana na Slici 5.16. Slike i istoimene anotacije (*labeli*) su razdvojene u tri različite mape: *train, valid* i *test*, dok unutar tih mapa su međusobno podijeljene u zasebne mape (*images* i *labels*).

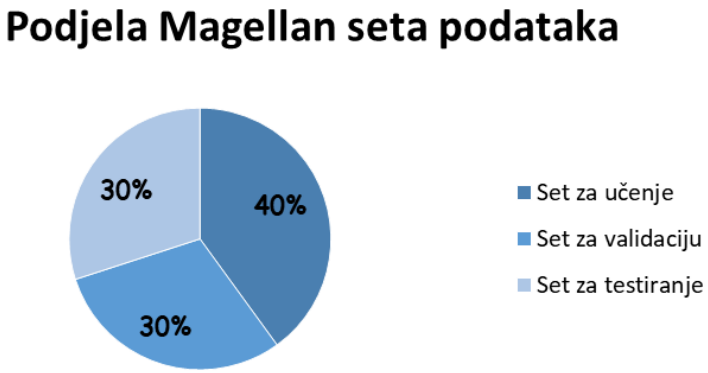


Slika 5.16. Struktura podjele seta podataka za YOLO algoritam [autor]

Za podjelu seta podataka korištena je Python skripta *createYoloDataset.py* dana u Dodatku C. Jedna od osnovnih funkcija korištenih za podjelu podataka u skripti je *test_train_split* funkcija iz Python knjižnice *sklearn.model_selection*.

```
train_images, test_images, train_labels, test_labels = train_test_split(images,
lables, test_size = 0.2)
```

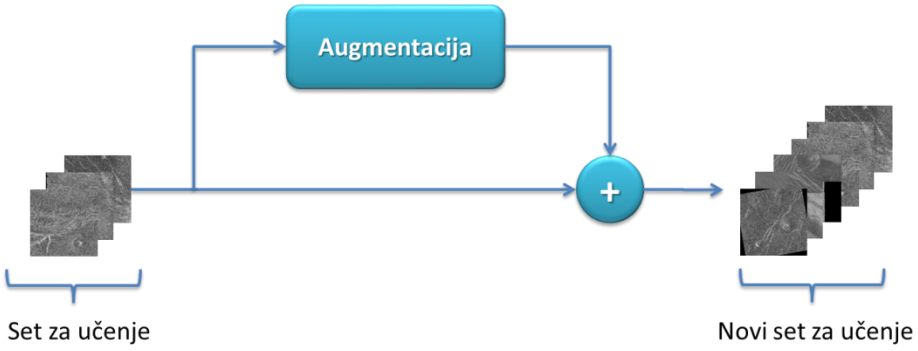
Na Slici 5.17. prikazan je omjer podjele Magellan seta podataka. Ovim omjerom podijele seta podataka nastojalo se dobiti što veći broj seta za validaciju pošto YOLOv5 modeli imaju veliki broj parametara.



Slika 5.17. Omjer podijele Magellan seta podataka [autor]

5.5. Augmentacija podataka

Zbog relativno malog broja slika u Magellan setu podataka nije se moglo rasporediti dovoljan broj slika u setu za učenje kako bi se učinkovito istrenirao model. Broj slika u setu za učenje može se umjetno povećati augmentacijskim tehnikama. Augmentacija je proces umjetnog povećanja količine novih podataka iz već postojećih podataka. To uključuje dodavanje manjih izmjena podacima ili korištenje modela strojnog učenja za generiranje novih podataka [51]. Shematski prikaz augmentacije podataka prikazan je na Slici 5.18.

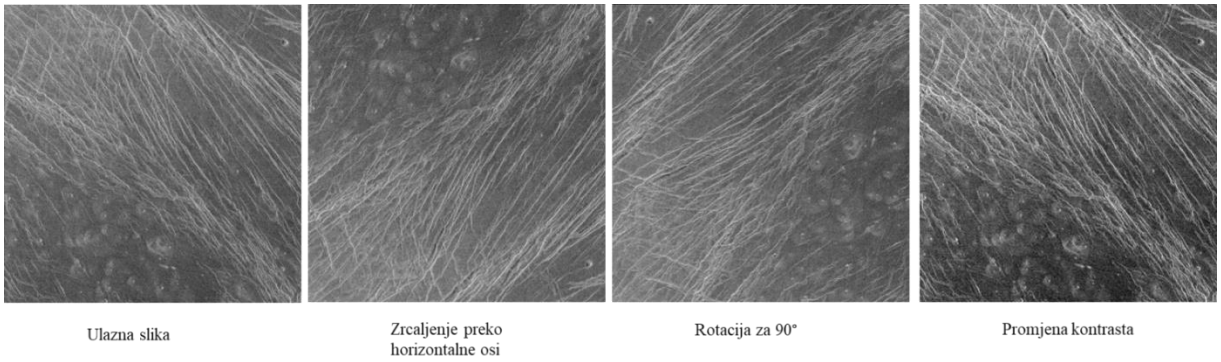


Slika 5.18. Shematski prikaz augmentacije podataka [autor]

Prilikom augmentacije seta podataka mora se pripaziti da ne dođe to takozvanog *data leakega*, odnosno bitan je redoslijed između podijele seta podataka i augmentacije. Ukoliko se augmentacija podataka učini prije podijele seta podataka dolazi do problema gdje se dvije verzije iste slike (koja sadrži iste značajke) nađu u setovima za treniranje i testiranje. Ukoliko se to desi rezultati takvog modela nisu valjani te tome treba pristupiti s oprezom. Prilikom augmentacije seta za treniranje pokušavano je raditi smislene augmentacije, tj. radila se manipulacija nad slikama na taj način da nove umjetno dobivene slike dovoljno dobro predstavljaju okoliš u kojem se radi detekcija vulkana. Prilikom obrade slika neki primjeri augmentacijskih tehnika su rotacija, zrcaljenje, promjena kontrasta, zumiranje, translacija, itd. Prilikom augmentacije seta podataka za učenje koristile su se slijedeće metode:

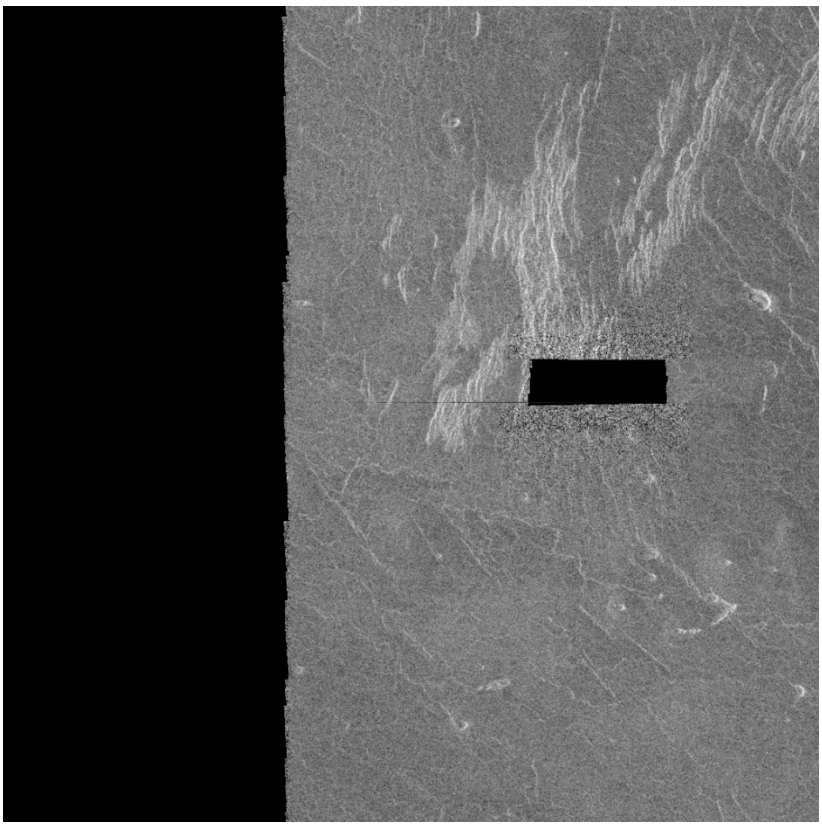
- Rotacije za 90,180 i 270°,
- zrcaljenja preko horiznotalne i vertikalne osi,
- promjena kontrasta, svjetline i
- *cutout* tehnika.

Na Slici 5.19. prikazane su korištene neke od tehnika augmentacija (točke od 1-3).



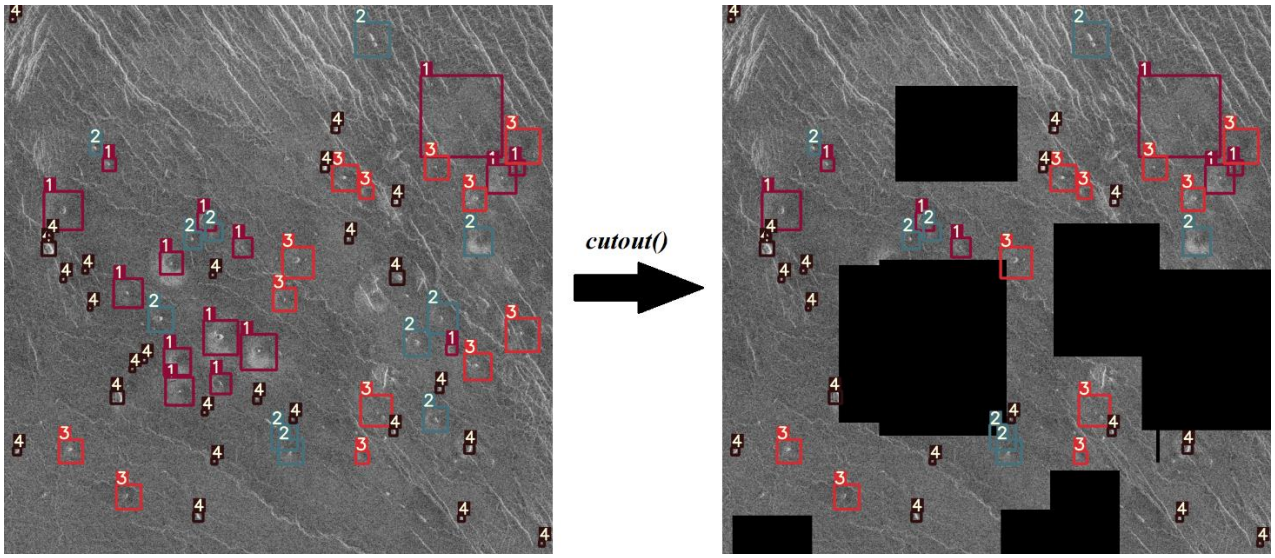
Slika 5.19. Primjeri korištenih tehinka augmentacija [autor]

Promotre li se neke od slika iz Magellan seta podataka može se primijetiti da neke slike imaju „crne zone“. One nastaju u slučaju kada radar nije uspio uhvatiti povratne zrake ili taj dio površine planeta nije bio u dometu. Slika 5.20. prikazuje primjer slike sa „crnom zonom“.



Slika 5.20. Primjer slike sa takozvanom „crnom zonom“ [autor]

Kako bi se simulirala ova pojava u Dodatku D razvijena je *cutout()* funkcija koja crta crne pravokutnike nasumičnih dimenzija i na nasumičnim lokacijama. Zatim ista ta funkcija provjerava po *label* datotekama (anotacijama) jeli prekrila više od 50% površine nekog vulkana. Ukoliko je crni pravokutnik prekrio više od 50% površine nekog vulkana briše ga iz liste anotacija. Primjer ove tehnike augmentacije prikazan je na Slici 5.21



Slika 5.21. . Primjer cutout() tehnike augmentacije [autor]

Implementacije gore navedenih tehnika u Pythonu dane su u Dodacima D i E (*AugmentFunctions.py* i *AugmentMain.py*). Prilikom augmentacije seta za učenjem koristeći ove skripte, u *config.txt* datoteci mogu se podešavati parametri augmentacije. Neki od parametara augmentacije su: *p* – vjerojatnost da se izvrši pojedina tehnika dokumentacije, maksimalni broj *cutout boxeva* i dr. Sadržaj *config.txt* datoteke dan je u slijedećem dijelu:

```
1.0 #flipVertical vjerojatnost [0.0-1.0]
1.0 #flipHoriznotal vjerojatnost [0.0-1.0]
1.0 #rotate(90) vjerojatnost [0.0-1.0]
1.0 #rotate(180) vjerojatnost [0.0-1.0]
1.0 #rotate(270) vjerojatnost [0.0-1.0]
1.0 #cutout vjerojatnost [0.0-1.0]
10 #broj cutout boxeva po slici
300 #maksimalna dimenzija cutout boxa u [pikselima]
1.0 #vjerojatnost da promjeni brightnes
1.0 #vjerojatnost da promjeni konrast
```

6. TRENIRANJE I OCJENA MODELA KLASIFIKACIJE

Za treniranje YOLOv5 modela korištena je Google Colab platforma. Google Colab besplatna platforma razvijena od strane Google Research-a koja omogućuje rad na oblaku (engl. Cloud computing). To je Juyter notebook usluga koja pruža besplatan pristup računalnim resursima uključujući grafičke kartice s NVIDIA CUDA podrškom. Temeljno je namijenjena za strojno učenje, analizu podataka i edukacije. U ovome djelu rada predstavljena je statistika podataka za treniranje, način treniranja modela te dobiveni rezultat.

6.1. Osnovni model

Prije početka treniranja i procjena rezultata modela, potrebno je odrediti bazni model na osnovu kojeg će se promjenom hiperparametara i tehnikama augmentacije podataka vršiti usporedba i procjena točnosti istreniranih modela. Skripta za učenje modela dana je u Dodatku G. Učenje modela počelo je na standardnom setu podataka bez korištenja augmentacijskih tehnika, koji je podijeljen u omjeru 40:30:30 na setove za učenje, validaciju i testiranje respektivno. Podaci osnovnog seta podataka dani su u Tablici 6.1.

Tablica 6.1. Podaci osnovnog seta podataka [autor]

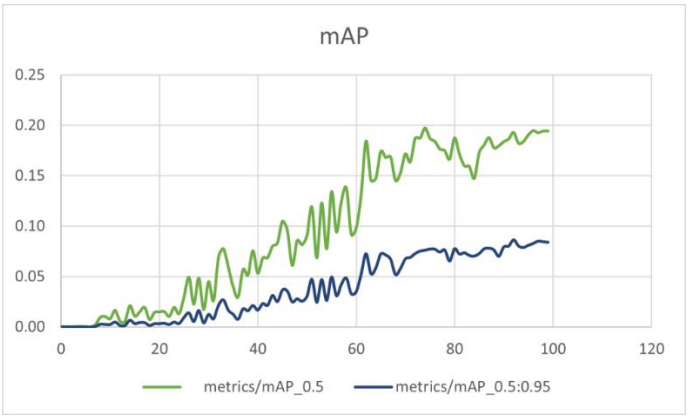
Osnovni set		**Augmentacija**	**Broj instanca po klasi**				
Set	Broj slika	Broj slika dobiven augmentacijom	K1	K2	K3	K4	Ukupni broj instanci po setu
Set za učenje	54	-	49	93	186	207	535
Set za validaciju	40	-	41	111	172	178	502
Set za testiranje	40	-	53	89	152	189	483
Ukupni broj slika		134	Ukupni broj instanci				1520

Kao osnovni model korišten je YOLOV5m6 model, te hiperparametri koji se koriste za treniranje svakog modela dani su u Dodatku H. Neki od osnovnih parametara dani su Tablici 6.2.

Tablica 6.2. Bitniji hiperparametri za treniranje modela [autor]

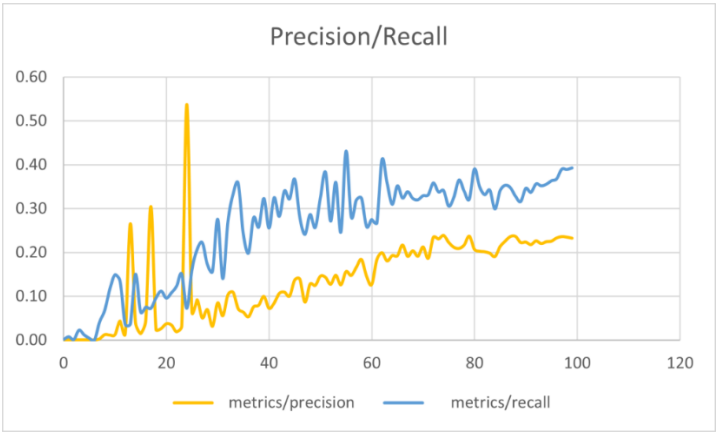
Hiperparametar	Vrijednost
Broj iteracija	100
Optimizacijski algoritam	SGD
Dimenzija ulazne slike	1024x1024
Stopa učenja	0.01
Batch size	16

Na Slici 6.1. grafički je prikazana srednja prosječna preciznost na setu za validaciju tokom procesa učenja.



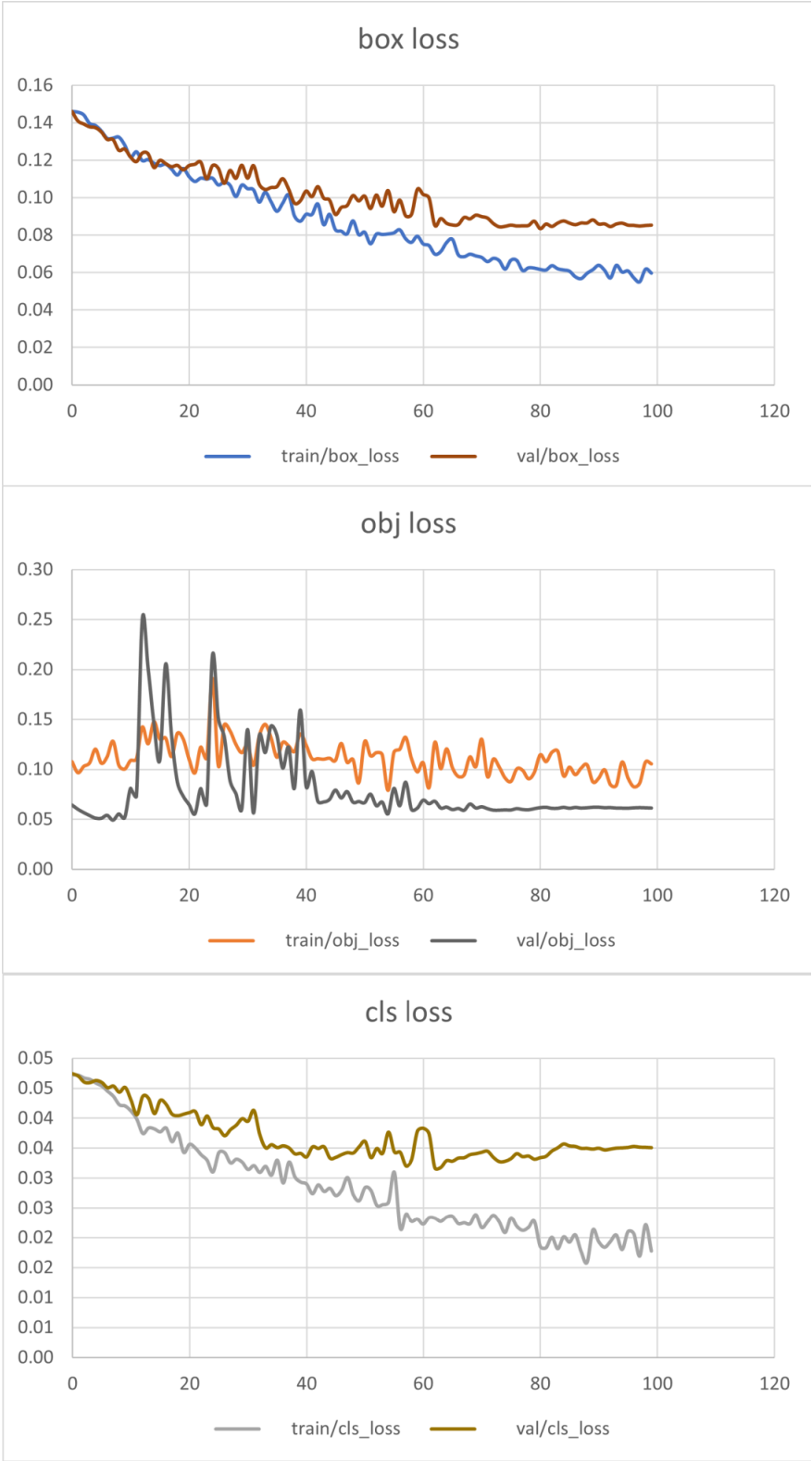
Slika 6.1. Srednja prosječna preciznost osnovnog modela [autor]

Na Slici 6.2. grafički su prikazani Precision i Recall na validacijskom setu podataka tokom procesa učenja.



Slika 6.2. Precision i Recall osnovnog modela [autor]

Na Slici 6.3. grafički su prikazane tri komponente funkcije gubitaka na validacijskom setu podataka tokom procesa učenja.



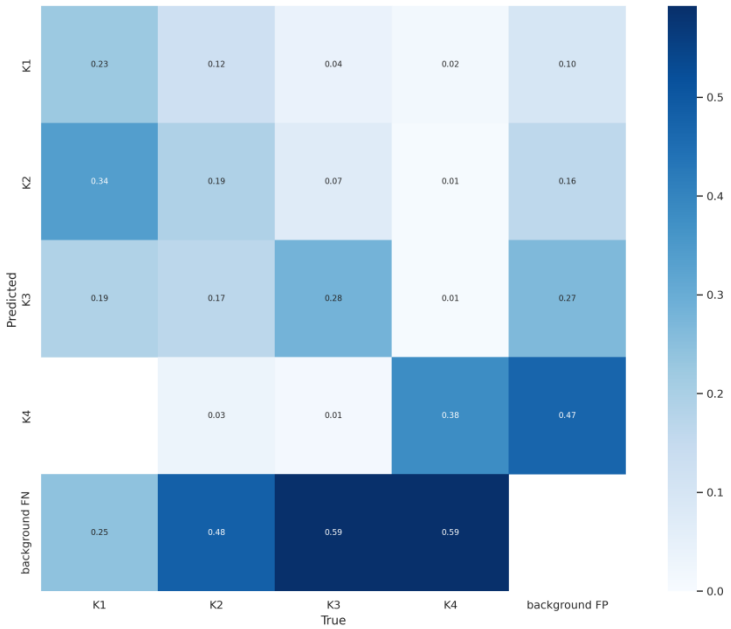
Slika 6.3. . Tri komponente funkcije gubitaka osnovnog modela [autor]

Rezultati osnovnog modela dani su u Tablici 6.3.

Tablica 6.3. Rezultati testiranja osnovnog modela [autor]

Mjera	Set za učenje	Set za validaciju	Set za testiranje
K1 - mAP@0.5	66.3%	34.8%	35.2%
K2 - mAP@0.5	44.6%	14.1%	14.4%
K3 - mAP@0.5	88.9%	10.3%	11.9%
K4 - mAP@0.5	67.9%	18.4%	17.1%
mAP@0.5 (Ukupni)	66.9%	18.4%	19.65%
F1-score	0.62	0.28	0.34

Iz Tablice 6.3. može se vidjeti kako je osnovni model nakon 100 iteracija učenja na setu za testiranje je imao iznos srednje prosječne preciznosti *mAP=19.65%*, što je mala točnost zbog ekstremno malog broja slika za treniranje modela. Zbog toga pristupljeno je korištenju tehnikama augmentacije na setu podataka za učenje. Na prethodnoj Slici 6.3. prikazane su tri komponente funkcije gubitaka, naime YOLO algoritam funkcije gubitaka računa u tri komponente te je konačna funkcija gubitaka zbroj te tri komponente. Matrica konfuzije prikazana je na Slici 6.4.



Slika 6.4. . Matrica konfuzije osnovnog modela [autor]

6.2. Osnovni model + klasična augmentacija

Radi relativno loše preciznosti modela postupilo se sa klasičnim metodama augmentacije, Parametri augmentacije dani su u Tablici 6.4.

Tablica 6.4. Parametri augmentacije [autor]

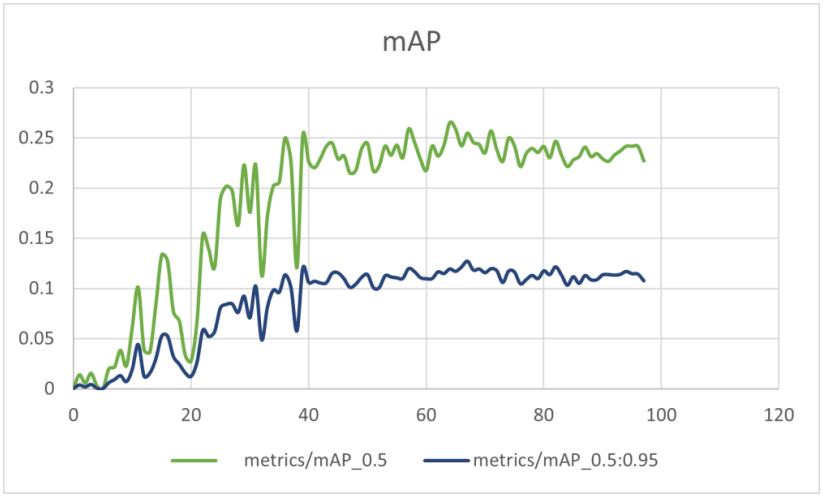
Tehnika augmentacije	vjerojatnost
Zrcaljenje preko vertikalne osi	100%
Zrcaljenje preko horizontalne osi	100%
Rotacija za 90 stupnjeva	100%
Rotacija za 180 stupnjeva	100%
Rotacija za 270 stupnjeva	100%
Cutout	100%
Vjerojatnost da se promjeni svjetlina	100%
Vjerojatnost da se promjeni kontrast	100%

Iako se parametri augmentacije mogu podešavati, radi toga da se dobije što veći broj slika za treniranje sve tehnike augmentacije su izvršene na svim slikama za učenje. Nakon augmentacije seta za učenje dobiven je novi set podataka dan u Tablici 6.5.

Tablica 6.5. Podaci augmentiranog seta podataka [autor]

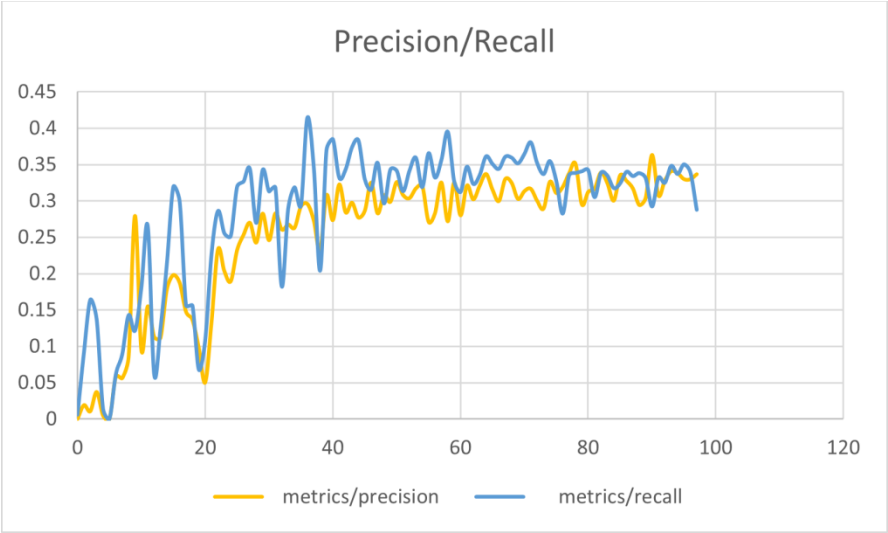
Osnovni set		Augmentacija	Broj instanca po klasi				
Set	Broj slika	Broj slika dobiven augmentacijom	K1	K2	K3	K4	Ukupni broj instanci po setu
Set za učenje	54	423	426	796	1628	1732	4582
Set za validaciju	40	-	41	111	172	178	502
Set za testiranje	40	-	53	89	152	189	483
Ukupni broj slika		557	Ukupni broj instanci				5522

Korištenjem klasičnih tehnika augmentacije iz Tablice 6.4., broj slika na setu u setu podataka za učenje je narastao na 477 slika, dok sveukupni broj objekata u setu za učenje je narastao na 4582 objekta. Na Slici 6.5. grafički je prikazana srednja prosječna preciznost na setu za validaciju tokom procesa učenja.



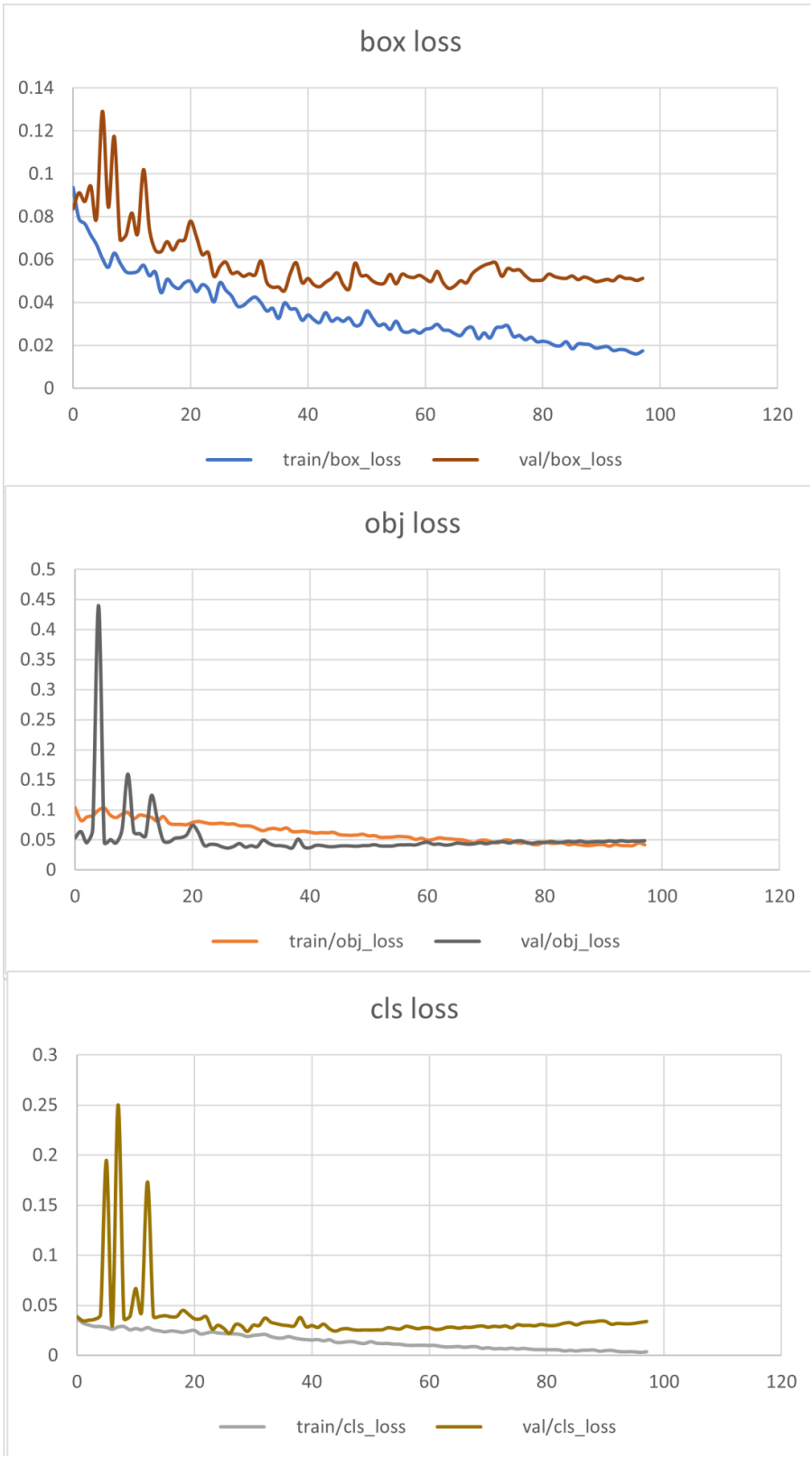
Slika 6.5. Srednja prosječna preciznost osnovnog modela sa klasičnim tehnikama augmentacije [autor]

Na Slici 6.6. grafički su prikazani Precision i Recall na validacijskom setu podataka tokom procesa učenja.



Slika 6.6. Precision i Recall osnovnog modela sa klasičnim metodama augmentacije [autor]

Na Slici 6.7. grafički su prikazane tri komponente funkcije gubitaka na validacijskom setu podataka tokom procesa učenja.



Slika 6.7. Tri komponente funkcije gubitaka osnovnog modela sa klasičnim metodama augmentacije [autor]

Rezultati osnovnog modela sa klasičnim metodama augmentacije dani su u Tablici 6.6.

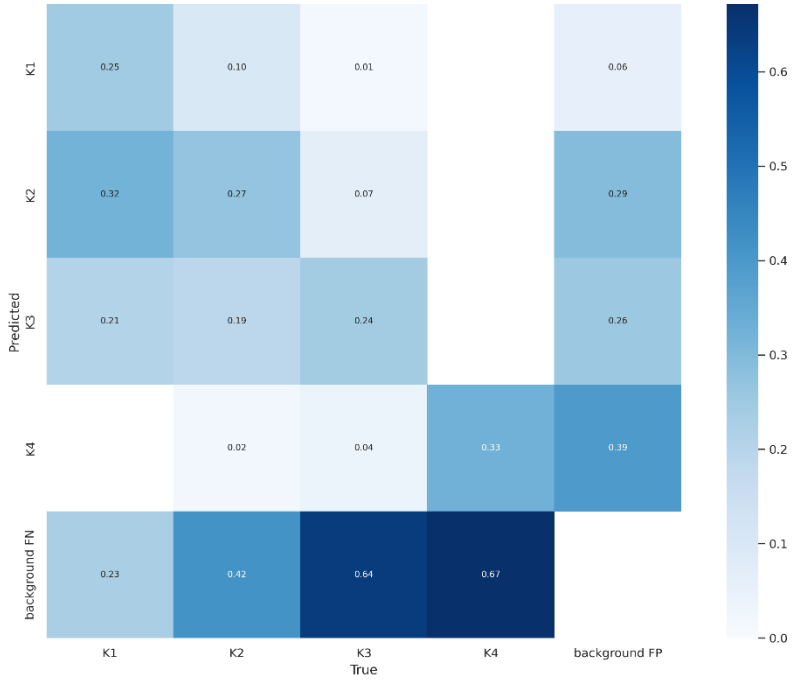
Tablica 6.6. Rezultati testiranja osnovnog modela sa klasičnim metodama augmentacije[autor]

Mjera	Set za učenje	Set za validaciju	Set za testiranje
K1 - mAP@0.5	85.9%	49.6%	39.2%
K2 - mAP@0.5	83.5%	24%	25%
K3 - mAP@0.5	89.1%	15.0%	17.4%
K4 - mAP@0.5	42.6%	13.2%	17.3%
mAP@0.5 (Ukupni)	75.3%	25.5%	24.7%
F1-score	0.72	0.34	0.34

Korištenjem klasičnih metoda augmentacije nad slikama u setu za učenje, dolazi do porasta preciznosti modela za ≈5% na testnom setu podataka. Nažalost ne dolazi do drastičnog porasta preciznosti i dobiveni rezultati i dalje su nezadovoljavajući. Razlog tome je i dalje relativno mali broj slika za učenje modela sa toliko velikim brojem parametara. Iako se broj slika u setu za učenje umjetno povećao za faktor 8 i dalje je nedovoljan broj za uspješno učenje modela. Prema [52] za uspješno učenje modela potrebno je barem više od 1500 slika po objektu te barem više od 10000 sveukupno označenih objekata po klasi. Pošto u ovom slučaju se na većini slika nalaze svi objekti, za uspješno treniranje dovoljno je da se broj slika umjetno poveća na otprilike tu brojku. Nažalost koristeći klasične tehnike augmentacije broj slika se može umjetno povećati samo do određene granice (može se povećati za faktor broja koliko se tehnika koristi). Još jedan problem proizlazi iz toga što Magellan set podataka ima ekstremno mali broj slika (134 sveukupno), te ovisno o podijeli seta podataka se može kvalitetno istrenirati model. Prilikom podijele seta podataka trebalo je paziti na to da se uzme dovoljan broj slika za validaciju i testiranje kako bi se pokrio veći broj slučajeva u stvarnom svijetu, uz to trebalo je pustiti dovoljno velik broj slika kako bi se model mogao učiti. Radi toga pristupilo se naprednijoj tehnici augmentacije podataka, te će ta tehnika biti opisana u slijedećem potpoglavlju.

Matrica konfuzije osnovnog modela sa klasičnim metodama augmentacije prikazan je na Slici 6.8. Iz matrice konfuzije može se zaključiti kako model često ima problem sa klasifikacijom četvrte klase vulkana (K4), jer ga često klasificira kao pozadinu. Promatrajući graf iz Slike 5.10., može se primijetiti kako vulkani iz K4 prosječno zauzimaju samo 2% površine slike.

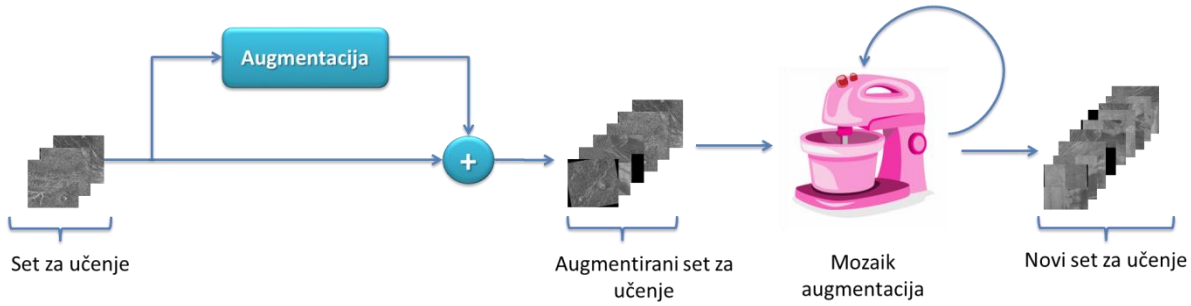
Tako mali objekti YOLO algoritmu zadaju probleme za detekciju i klasifikaciju, te je to jedna od mana tog algoritma. Točnost klasifikacije i detekcija objekata tako malih dimenzija može povećati na način da se YOLO model trenira na većoj ulaznoj rezoluciji od nativne, no povećanje rezolucije drastično povećava hardverske zahtjeve i vrijeme treniranja modela. Postoje i druge metode kojima se može povećati detekcija ovako malih objekata, te će njihov koncept i ideja biti objašnjeni u budućem poglavlju.



Slika 6.8. Matrica konfuzije osnovnog modela sa klasičnim tehnikama augmentacije [autor]

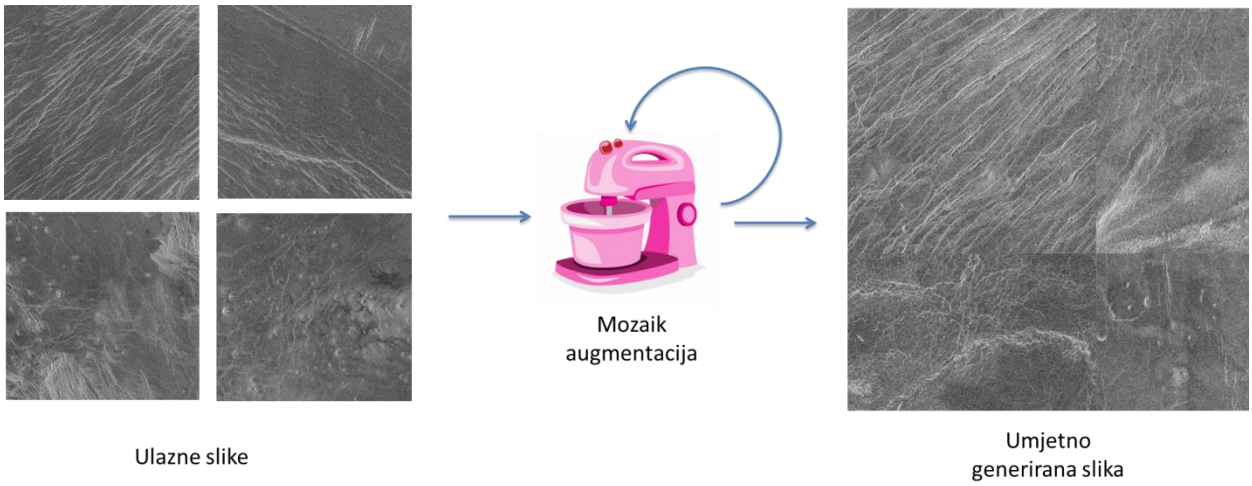
6.3. Osnovni model + mozaik augmentacija

Pošto klasičnim metodama augmentacije se nije uspio umjetno povećati broj slika na dovoljan broj, moralo se pristupiti naprednijoj tehnici augmentacije. Kako bi se dobio dovoljan broj slika i objekata za učenje modela, uz klasičnu tehniku augmentacije pristupilo se mozaik augmentaciji. Na Slici 6.9. prikazan je shematski prikaz implementacije augmentacijskog *pipeline-a* koristeći mozaik augmentaciju. Uz klasičnu augmentaciju seta podataka, serijski je dodana mozaik augmentacija kako bi se umjetno povećao broj slika. Razlog korištenja klasične augmentacije u ovoj seriji je da se maksimalno poveća broj slika prije ulaza u mozaik augmentaciju kako bi se dobio što raznovrsniji broj slika.



Slika 6.9. Shematski prikaz implementacije augmentacijskog pipeline-a [autor]

Mozaik tehnika augmentacije funkcionira na principu da od dvije ili više slika (najčešće 4) umjetno generira novu sliku koja je sastavljena od manjih dijelova prethodnih slika. Ova metoda augmentacije nije uvijek prikladna za upotrebu jer može generirati nove slike koje imaju neke nove značajke koje nisu reprezentativne za slučajeve okoline u kojem će model djelovati. No u ovom slučaju ova tehnika augmentacije je prikladna za upotrebu. Primjer mozaik augmentacije prikazan je na Slici 6.10.



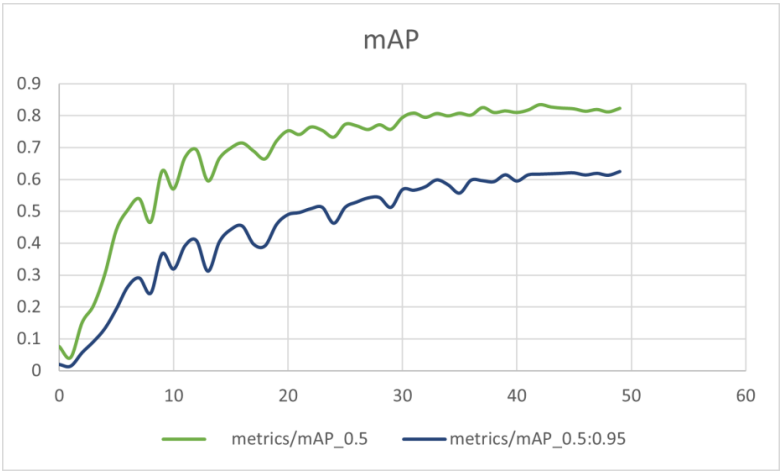
Slika 6.10. Primjer mozaik tehnike augmentacije [autor]

Koristeći gore navedeni augmentacijski pipeline dobiveni su podaci prikazani u Tablici 6.7.

Tablica 6.7. Podaci augmentiranog seta podataka 2 [autor]

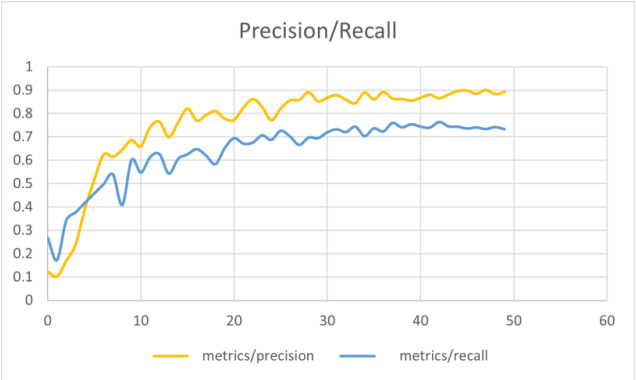
Osnovni set		Augmentacija	Broj instanca po klasi				
Set	Broj slika	Broj slika dobiven augmentacijom	K1	K2	K3	K4	Ukupni broj instanci po setu
Set za učenje	54	2808	3194	6740	13427	10044	33405
Set za validaciju	40	-	41	111	172	178	502
Set za testiranje	40	-	53	89	152	189	483
Ukupni broj slika		2942	Ukupni broj instanci				34390

Iz Tablice 6.7. može se primijetiti kako je broj slika seta za učenje umjetno porastao na 2808, slike dok ukupni broj objekata u testnom setu je narastao na 33405. Ovaj model treniran je na 50 iteracija učenja zbog velikog vremenskog razdoblja koji bi bio potreban da se model učio na 100 iteracija. Na Slici 6.11. grafički je prikazana srednja prosječna preciznost na setu za validaciju tokom procesa učenja.



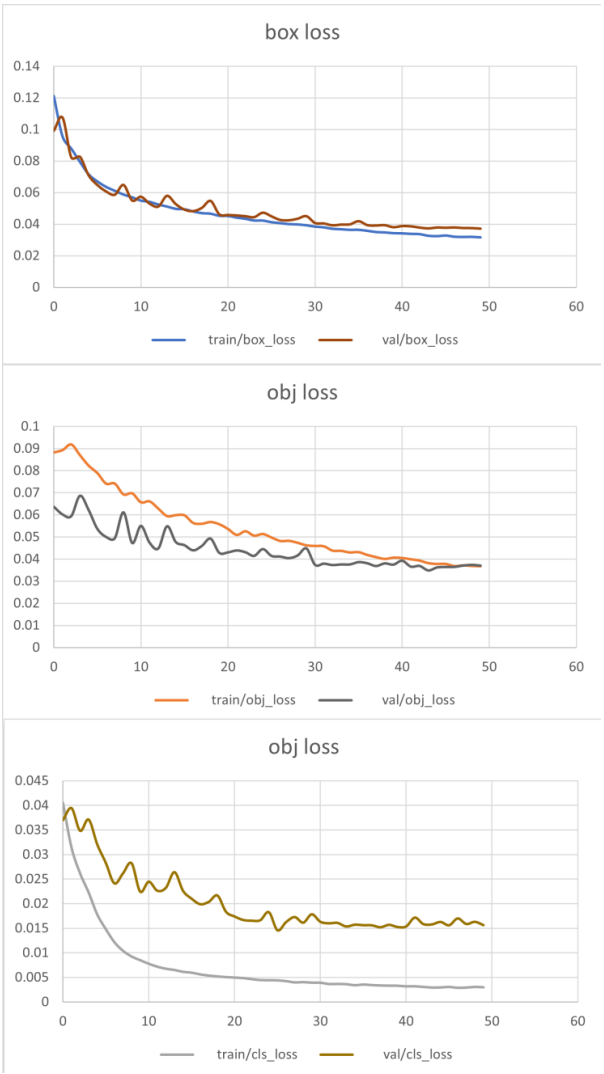
Slika 6.11. Srednja prosječna preciznost osnovnog modela sa napredniji, tehnikama augmentacije [autor]

Na Slici 6.12. grafički su prikazani Precision i Recall na validacijskom setu podataka tokom procesa učenja.



Slika 6.12. Precision i Recall osnovnog modela sa naprednijim tehnikama augmentacije [autor]

Na Slici 6.3. grafički su prikazane tri komponente funkcije gubitaka na validacijskom setu podataka tokom procesa učenja.



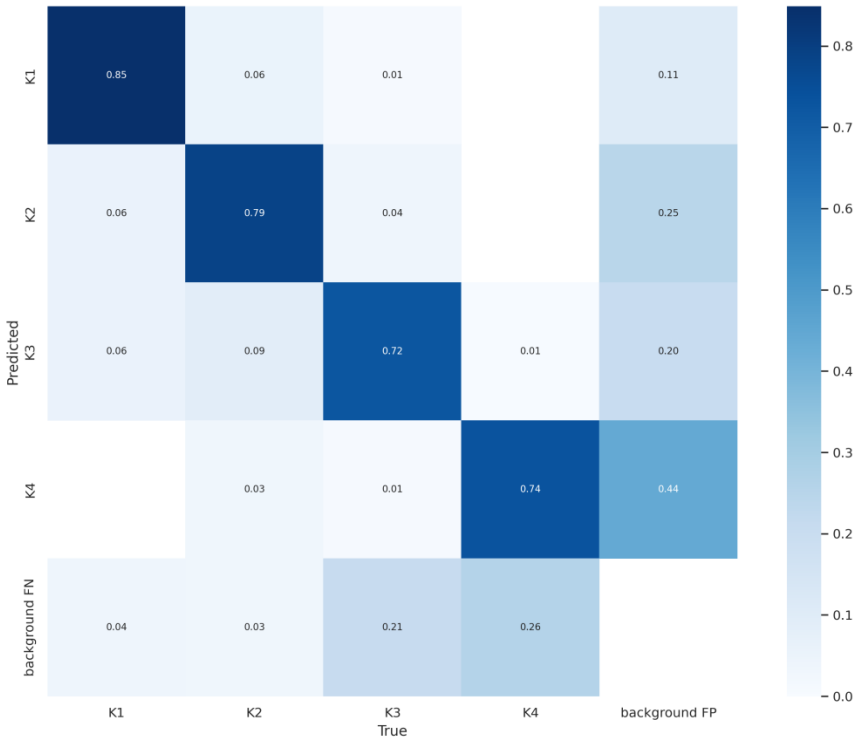
Slika 6.13. Tri komponente funkcije gubitaka osnovnog modela sa naprednijim tehnikama augmentacije [autor]

Rezultati osnovnog modela sa naprednijim metodama augmentacije dani su u Tablici 6.8.

Tablica 6.8. Rezultati testiranja osnovnog modela sa naprednijim metodama augmentacije[autor]

Mjera	Set za učenje	Set za validaciju	Set za testiranje
K1 - mAP@0.5	97.9%	90.8%	90.7%
K2 - mAP@0.5	99.0%	81.4%	89.2%
K3 - mAP@0.5	99.2%	84.1%	80.0%%
K4 - mAP@0.5	94.1%	74.6%	73.9%
mAP@0.5 (Ukupni)	97.9%	82.7%	83.5%
F1-score	0.97	0.81	0.83

Matrica konfuzije istreniranog modela prikazana je na Slici 6.14.



Slika 6.14. Matrica konfuzije osnovnog modela sa naprednijim tehnikama augmentacije [autor]

Promatranjem rezultata iz Tablice 6.8. može se primijetiti da je preciznost modela na testnom setu naspram osnovnog modela porasla za ≈65%, što je dokaz da je ovo uspješna metoda za treniranje modela. Iako model ima toliko veliki porast u svojoj točnosti i dalje ima veliki

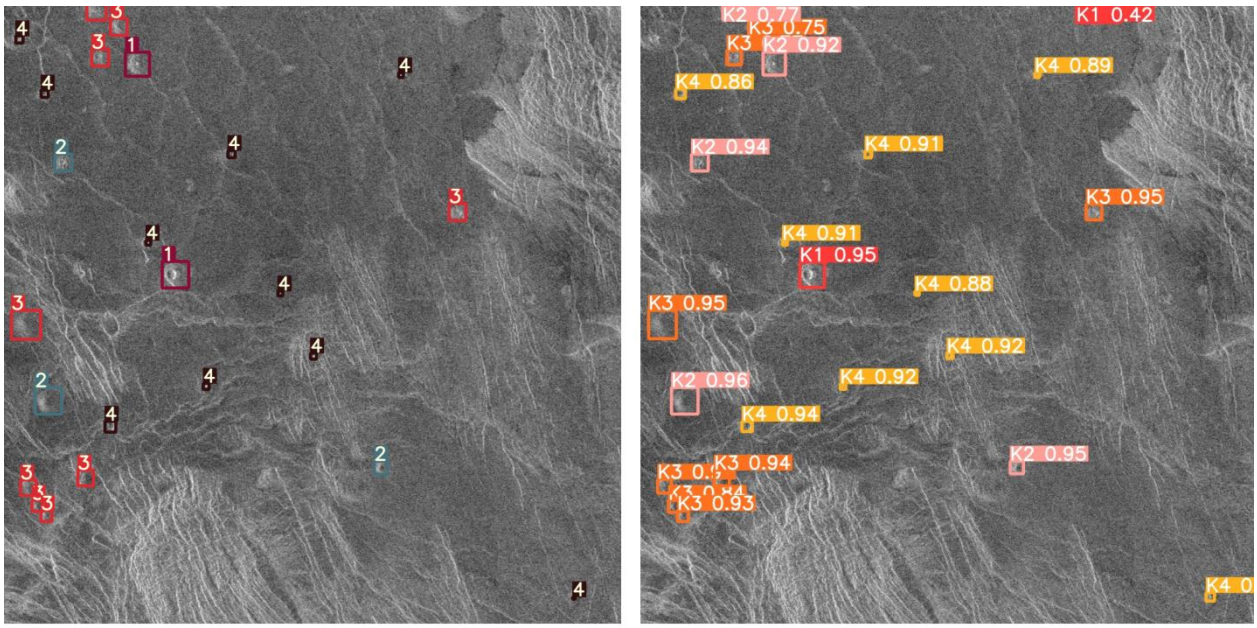
overfitting naspram seta podataka za učenje (≈13%). Jedan od razloga tolikog overfittinga je preciznost prilikom detekcije i klasifikacije klase 4 vulkana. Uz to, veća točnost modela može postići ručnim podešavanjem hiperparametara, ali to je vrlo zahtjevan proces radi velikog broja hiperparametara koje YOLO ima (Dodatak G) Drugi način podešavanja hiperparametara je pomoću evolucijskog algoritma prema [53], ali također to je hardverski i vremenski jako zahtjevan proces. Primjerice kako bi se podesili hiperparametri nekog modela pomoću evolucijskog algoritma, prema [53] je preporučeno barem 300 generacija. Svaka generacija stvara se jednom iteracijom učenja modela, te primjerice podešavanje hiperparametara ovog modela trajalo bi 1200 sati ili 50 dana neprestano (300 generacija x 4h vrijeme jednog učenja) . Treći način poboljšavanja preciznosti modela je učenje modela sa više parametara (npr. yolov5l), ali modeli sa prevelikim brojem parametara nastoje učiti na značajkama kao što je šum te time im pada preciznost. Još jedan od načina koji bi vrijedilo probati je učenje modela sa manjim brojem parametara nego osnovni model korišten u ovome radu, zato jer postoji mogućnost da osnovni model radi gore navedenu grešku. No usprkos svemu tome, model je dao zadovoljavajuće rezultate za detekciju i klasifikaciju vulkana na površini planeta Venere.

6.4. Praktični primjer detekcije

Nakon treniranja YOLOv5m6 modela metodom opisanom u potpoglavlju, preuzeti su težinski faktori koji se koriste prilikom detekcije objekata na setu podataka za testiranje. Detekcija na željenoj slici ili željenom skupu slika poziva se slijedećom naredbom u Google Colabu:

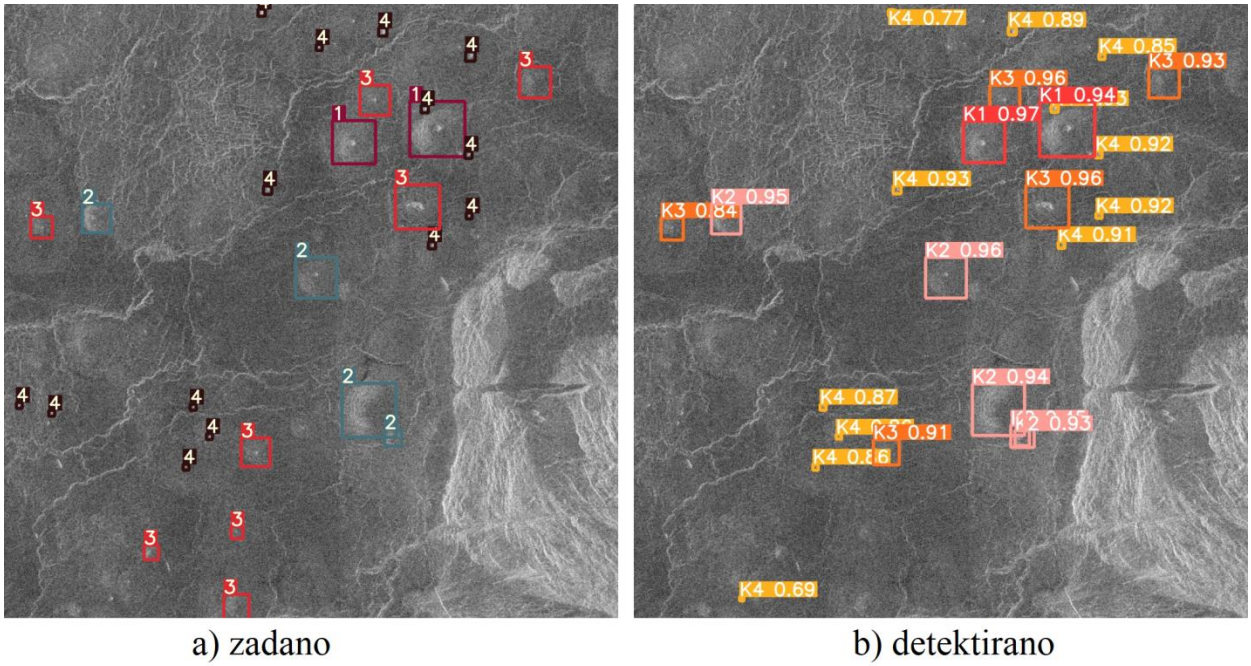
```
!python detect.py --weights {izvor tezinskih koef.} --img {rezolucija slike} --
conf {conf. treshold} --source {izvor slike}
```

Rezultati detekcije YOLOv5 modela prikazani su na Slikama 6.15. – 6.18. Iz Slike 6.15. može se primijetiti kako dolazi do ispravne detekcije klasifikacije većine objekata.



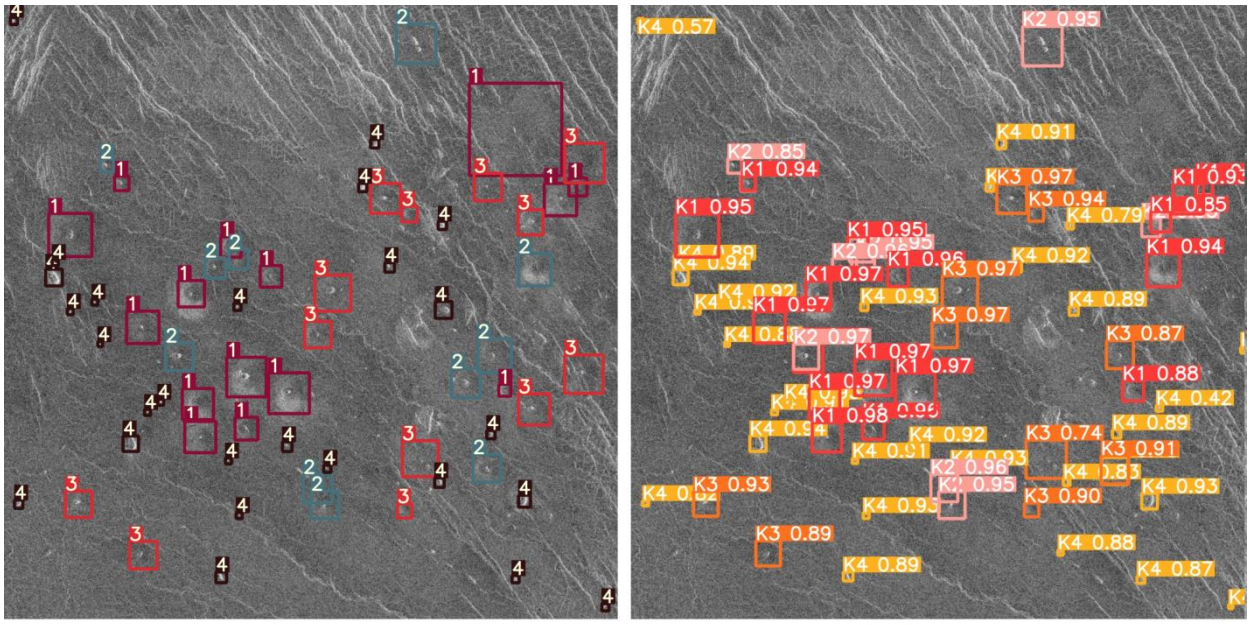
Slika 6.15. Detekcija modela 1 [autor]

Na Slici 6.16. prikazan je primjer gdje ne dolazi do potpune detekcije vulkana iz klase 4. razlog tome su, kako je već rečeno, premale dimenzije



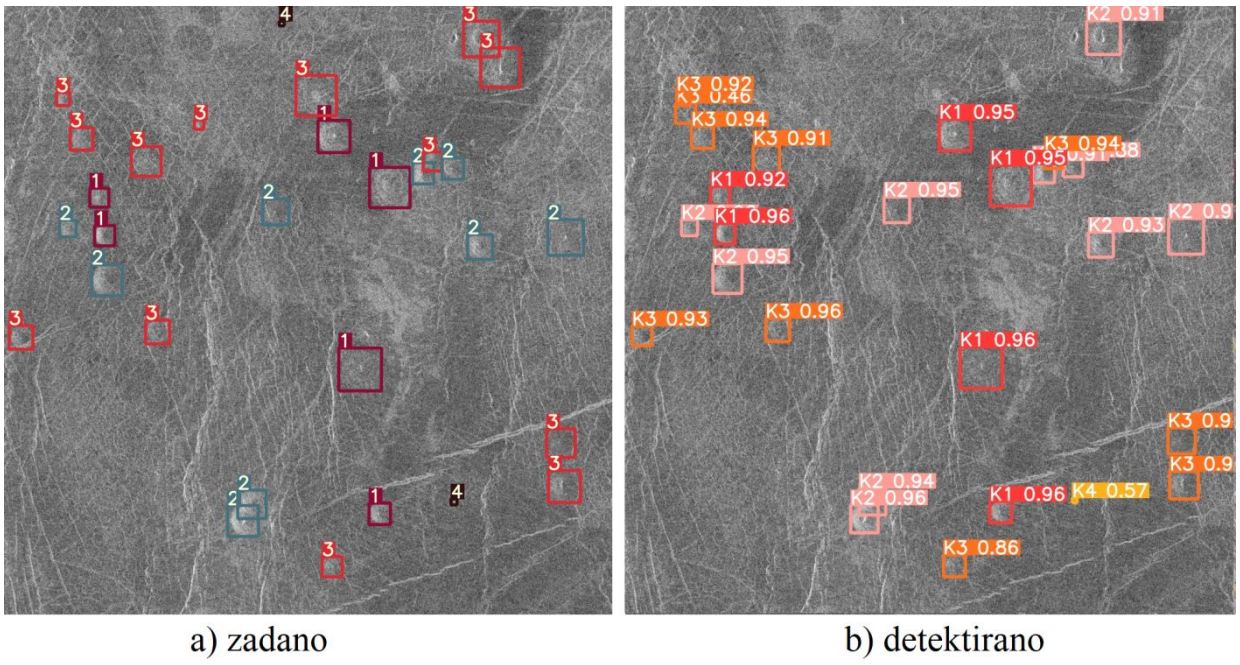
Slika 6.16. Detekcija modela 2 [autor]

Sa slike 6.17. može se primijetiti kako YOLO algoritam ima problem prilikom detekcije i klasifikacije objekata na mjestima gdje se nalazi puno instanca raznih klasa



Slika 6.17. Detekcija modela 3 [autor]

Na Slici 6.18. prikazan je jedan od slučajeva gdje su vulkani klase 3 krivo klasificirani (kao klasa 2) radi svojih sličnih karakteristika između klasa.

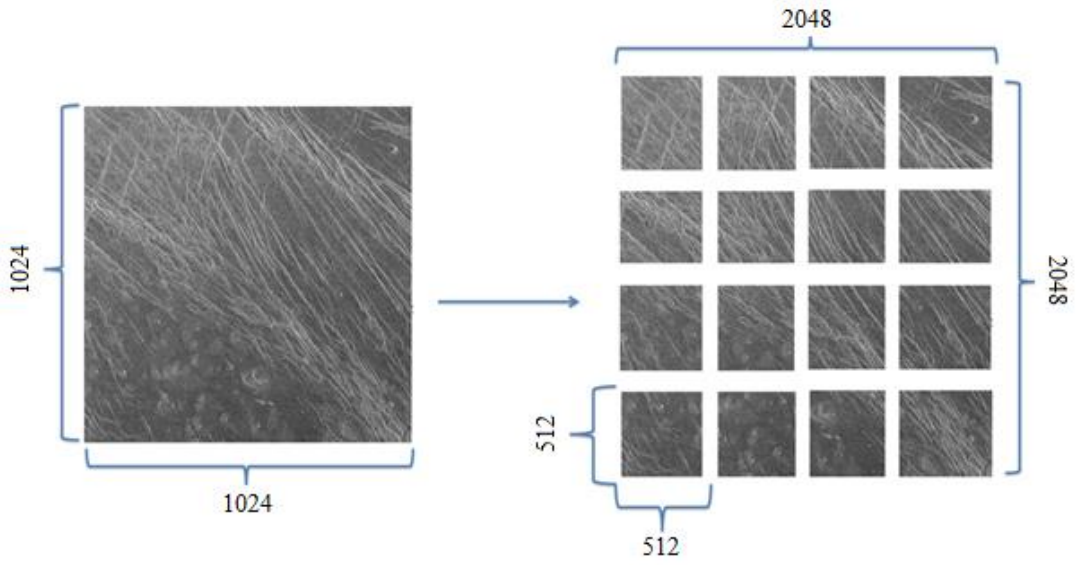


Slika 6.18. Detekcija modela 4 [autor]

Iz Slika 6.15. – 6.18. može primijetiti da je model uspješno detektirao i klasificirao većinu, razlog tome su, kako je već rečeno, premale dimenzije. Usprkos svemu, model je dao zadovoljavajuće rezultate detekcije i klasifikacije vulkana sa satelitskih slika površine planeta Venere, te se može zaključiti kako je uspješno izrađen sustav za detekciju vulkana. Naknadno točnost modela se može poboljšati ako se vrši treniranje mreže na barem duplo većim rezolucijama od nativne rezolucije ulaznih slika, te detekcija i klasifikacija objekata na isto. Također u slijedećem potpoglavlju iznijeti će se idejno rješenje kako bi se mogla poboljšati klasifikacija i detekcija manjih objekata.

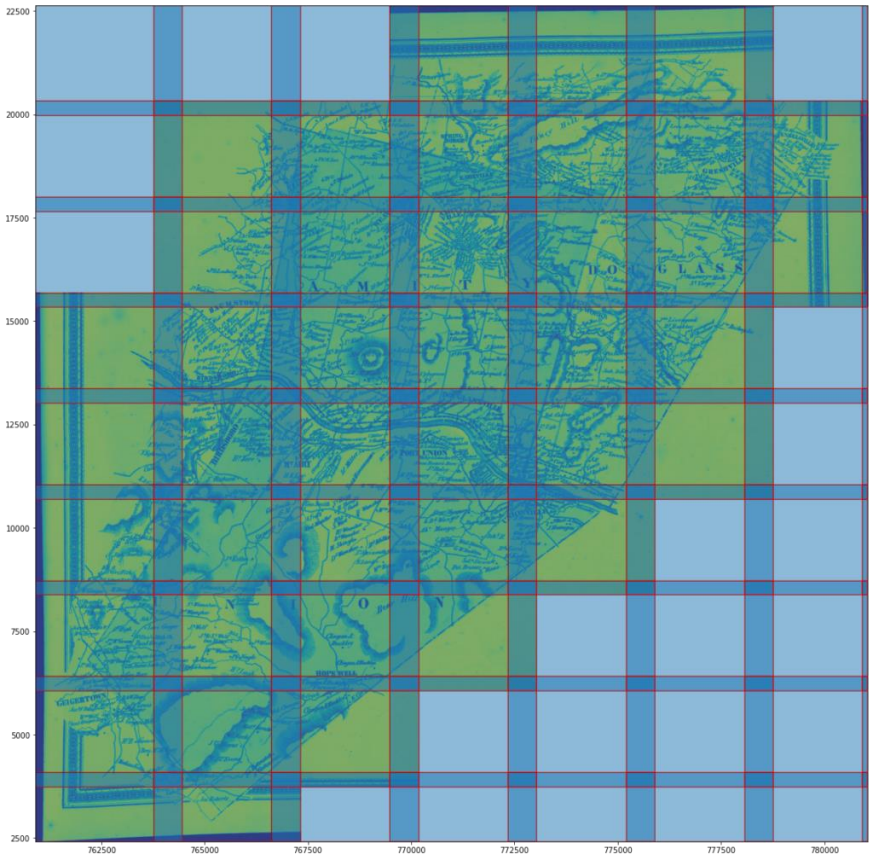
6.5. Daljnji smjer istraživanja

Jedno od najvećih problema, uz premali broj slika za učenje modela, su sitne dimenzije vulkana iz K4. Ukoliko se poveća rezolucija ulaznih slika za učenje mreže i detekciju, dolazi do eksponencijalnog rasta potrebnih hardverskih resursa i proteklog vremena za učenje mreže što nije prihvatljivo. Drugo način je da se poveća kompleksnost modela (veći broj parametara), no onda dolazi do problema da veći model uči iz nepotrebnih značajki kao što je šum, pa takav model loše generalizira. Jedno od predloženih rješenja je da se slika razdvoji u mrežu koja se sastoji od više manjih dijelova slike koji se skaliraju na određenu manju rezoluciju. Tako npr. ulazna slika dimenzija 1024 x 1024 piksela, može se odvojiti u 4 x 4 mrežu gdje svaka manja slika iz mreže ima dimenzije 512 x 512 piksela. Primjer odvajanja na mrežu prikazan je na Slici 6.19. Ta metoda se također naziva i tiling.



Slika 6.19. Primjer podjele slike na mrežu [autor]

Time se efektivno dobije ulazna slika duplo veće rezolucije na kojoj se vrše detekcija i klasifikacija. Prednost ovoga je što se mreža ući na slikama manje rezolucije, a suprotno tome su objekti na slici duplo veći. Podjelom slike na mrežu od više dijelova, povećava se efektivno njena rezolucija na kojoj se odvija detekcija i klasifikacija. Iskustveno veći broj slika manjih dimenzija ima puno manji utjecaj na hardverske zahtjeve i vrijeme treniranja, nego manje slika veće rezolucije. Jedan od problema koji se mogu javiti je da se prilikom podijele slike na mrežu, mogu odsjeći neki od objekata. Jedno od rješenja tome je da se mreža formira sa preklapanjem od 10-20% između susjednih slika (Slika 6.20), no onda se mora razviti algoritam koji nakon detekcije miče detektirane duplikate. Još jedan od zahtjeva ove metode je postojanje algoritma koji u djelu postporcesirnja rezultata mora imati mogućnost vračanje detekcija iz mreže nazad u originalnu sliku. Idejni kod ovog koncepta za razdvajanje slike u 2 x 2 mrežu priložen je u Dodatku H.



Slika 6.20. Koncept dijeljenja slike na mrežu sa međusobnim preklapanjem [54]

7. ZAKLJUČAK

U ovome radu razvijen je sustav za detekciju i klasifikaciju vulkana sa satelitskih snimaka površine planeta Venere koristeći YOLOv5 algoritam. Za uspješno treniranje modela potrebno je raspolagati što većim brojem slika i označenih objekata u setu podataka. No kod relativno malih setova podataka kao što je Magellan, moguće je umjetno povećati broj slika kako bi se pravilno istrenirao model. To se postiže zajedničkom uporabom klasičnih tehnika augmentacije podataka (npr. okretanje ili zrcaljenje slike) i naprednijih metoda (kao što je mozaik augmentacija) na način predstavljen u ovome radu. Također prilikom podijele seta podataka na setove za učenje, validaciju i testiranje, mora se voditi obzira prilikom odabira omjera podjele. Iz ovog rada zaključilo se da prilikom podijele relativnog malog seta podataka, ukoliko se ima model sa velikim brojem parametara, mora uzeti u obzir i veličina validacijskog ili testnog seta podataka kako bi se relativno dobro opisala stvarna okolina u kojem će model djelovati. Odabirom premalog omjera za setove podataka za validaciju i testiranje, postoji mogućnost da oni neće dovoljno dobro predstavljati okolinu te model neće imati dobre rezultate na testnim uzorcima. Prilikom izvođenja augmentacije podataka bitno je paziti kojim se metodama koristi, jer u suprotnom mogu se u set podataka za učenje uvesti neželjene značajke u podacima što rezultira gorim rezultatima modela. Testiranjem sustava na testnim slikama, model je dao zadovoljavajuće rezultate s obzirom na ograničenja koja pruža sam set podataka. Daljnji slijed istraživanja bio bi razviti algoritam koji dijeli sliku u mrežu opisan na način u poglavlju 6.5., kako bi se pokušala povećati preciznost modela kod detekcije i klasifikacije objekata veoma malih dimenzija (≈20x20 piksela). Zaključno, ovaj razvijen sustav svoju primjenu mogao bi pronaći kod mapiranja površina planeta u znanstvene svrhe ili kao alat koji pomaže prilikom planiranja mjesta slijetanja budućih istraživačkih misija.

LITERATURA

[1] „Umjetna inteligencija“, s Interneta, https://enciklopedija.hr/natuknica.aspx?ID=63150, pristupljeno 1. kolovoza 2022.

[2] Agarwal, Nakul & Chiang, Cheng-Wei & Sharma, Abhishek: „A Study on Computer Vision Techniques for Self-driving Cars“, 10.1007/978-981-13-3648-5_76., 2019. Godine

[3] Lorencin, Ivan. "An Intelligent System for Urinary Bladder Cancer Diagnostics : Doctoral dissertation." Disertacija, Sveučilište u Rijeci, Tehnički fakultet, 2022. https://urn.nsk.hr/urn:nbn:hr:190:025296

[4] „Artificial Neural Network (ANN)“, s Interneta, https://www.techopedia.com/definition/5967/artificial-neural-network-ann, pristupljeno: 1. kolovoza 2022.

[5] Herculano-Houzel, S.: „The Human Brain in Numbers: A linearly Scaled-up Primate Brain“, s Interneta, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2776484/, pristupljeno: 2. kolovoza 2022.

[6] Zimmer, C.: „100 Trillion Connections: New Efforts Probe and Map the Brain's Detailed Architecture“, s Inerneta, https://www.scientificamerican.com/article/100-trillion-connections/, pristupljeno: 2. kolovoza 2022.

[7] „Koji su dijelovi neurona?“, s Interneta, https://hr.sainte-anastasie.org/articles/neurociencias/cules-son-las-partes-de-la-neurona.html, pristupljeno: 2. kolovoza 2022.

[8] „Od živčane stanice do živčanog sustava“, s Interneta, https://edutorij.e-skole.hr/share/proxy/alfresco-noauth/edutorij/api/proxy-guest/3b8a4b4e-84b0-4580-aa6f-e38efe028ed9/biologija-8/m03/j01/index.html, pristupljeno: 5. kolovoza 2022.

[9] Dalbelo Bašić, B.; Čupić, M.; Šnajder, J.: „Umjetne neuronske mreže“, Umjetna inteligencija, Zavod za elektroniku, mikroelektroniku i inteligentne sustave, Fakultet elektrotehnike i računastva, Zagreb, svibanj 2008.

[10] „Sinaptički potencijal – Synaptic potential“, s Interneta, https://upwikihr.top/wiki/synaptic_potential, pristupljeno: 5. kolovoza 2022.

[11] Chandra, L. A.: „McCulloch-Pitts Neuron – Mankind's First Mathematical Model of A Biological Neuron“, s Interneta, https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1, pristupljeno: 5. kolovoza 2022.

[12] Car, Z.: „Umjetne neuronske mreže“, predavanja iz kolegija Primjena umjetne inteligencije, Tehnički fakultet Rijeka

[13] „Fundamentals of Deep Learning – Activation Functions and When to use Them?“, s Interneta, https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/, pristupljeno: 5. kolovoza 2022.

[14] „nuronova sit'“, s Interneta, http://po.licka.cz/neuronove_site_modely_neuronu_topologie_siti_metoda_backpropagation, pristupljeno 6. kolovoza 2022.

[15] Seb: „An Introduction to Neural Networks Loss Functions“, s Interneta, https://programmathically.com/an-introduction-to-neural-network-loss-functions/, pristupljeno: 6. kolovoza 2022.

[16] Gupta, A.: „A Comprenhensive Guide on Deep Learning Optimizers“, s Interneta, https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/, pristupljeno: 9. kolovoza 2022.

[17] Lee, A. i dr.: „Determination of Optimal Initial Weights of an Artificial Neural Network by Using the Harmony Search Algorithm: Application to Breakwater Armor Stones“, s Interenta, https://www.mdpi.com/2076-3417/6/6/164/htm, pristupljeno 17. kolovoza 2022.

[18] Kramberger, T. i dr: „PREGLED TEHNOLOGIJA U NEURONSKIM MREŽAMA“, *Polytechnic and design*, 7(1), str. 25-32. 2019. godine, https://doi.org/10.19279/TVZ.PD.2019-7-1-04

[19] IBM Cloud Education: „Supervised Learning“, s Interneta, https://www.ibm.com/cloud/learn/supervised-learning, pristupljeno: 17. kolovoza 2022.

[20] Pratt, M. K.: „unsupervised learing“, s Interneta, https://www.techtarget.com/searchenterpriseai/definition/unsupervised-learning, pristupljeno: 17. kolovoza 2022.

[21] Sharma, M.: „A Beginners Guide to Unsupervised Learning“, s Interneta, https://medium.com/analytics-vidhya/beginners-guide-to-unsupervised-learning-76a575c4e942, pristupljeno 17.kolovoza 2022.

[22] Osiński, B.; Budek, K.: „What is reinforcement learning? The complete guide“, s Interneta, https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/, pristupljeno: 20. kolovoza 2022.

[23] LeCunn, Y. i dr.: „Gradient-Based Learning Applied to Document Recognition“, Proc of the IEEE, November 1998.

[24] Dabović, Marko & Tartalja, Igor. (2017). Duboke konvolucijske neuronske mreže – koncepti i aktualna istraživanja.

[25] Saha, S.: „A Comprehensive Guide to Convolutional Neural Networks – the ELI5 way“, s Interneta, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, pristupljeno: 20. kolovoza 2022.

[26] „File: Object detector 1stage vs 2stage.png“, s Interneta, https://commons.wikimedia.org/wiki/File:Object_detector_1stage_vs_2_stage.png, pristupljeno 21. kolovoza 2022.

[27] „Ultralytics YOLOv5“, s Interneta, https://github.com/ultralytics/yolov5, pristupljeno 22. kolovoza 2022.

[28] Redmon, J. i dr.: „You Only Look Once: Unified, Real-Time Object Detection“, University of Washington, 2015. Godine

[29] Singh, A.: „Selecting the Right Bounding Box Using Non-Max Suppression (with implementation)“, s Interneta, https://www.analyticsvidhya.com/blog/2020/08/selecting-the-right-bounding-box-using-non-max-suppression-with-implementation/, pristupljeno 22. kolovoza 2022.

[30] Azam, M.:.“Deep Learning Applied to White Light and Narrow Band Imaging Videolaryngoscopy: Toward Real-Time Laryngeal Cancer Detection. The Laryngoscope“. 132. 10.1002/lary.29960., 2021.

[31] Wang C-Y.: „CSPNet: A new backbone that can enchance learning capability of CNN“, arXiv:1911.11929v1, 2019.

[32] Hui, J.: „Understanding Feature Pyramid Networks for object detection (FPN)“, s Interneta, https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c, pristupljeno 26.kolovoza 2022.

[33] Liu, S. i dr.: „Path Aggregation Network for Instance Segmentation“, arXiv:1803.01534v4, 2018.

[34] Rath, S. „Custom Object Detection Training using YOLOv5“, s Interneta, https://learnopencv.com/custom-object-detection-training-using-yolov5/, pristupljeno, 26. kolovoza 2022.

[35] Koech, K.E.: „Object Fetection Metrics Wiwth Worked Example“, s Interneta, https://towardsdatascience.com/on-object-detection-metrics-with-worked-example-216f173ed31e, pristupljeno 26. kolovoza 2022.

[36] „Bounding Box Interesection over Union (IoU) Measure“, s Interneta, http://www.gabormelli.com/RKB/Bounding_Box_Intersection_over_Union_(IoU)_Measure, pristupljeno 27.kolovoza 2022.

[37] „The Sloan Digital Sky Survey“, s Interneta, https://www.sdss.org/, pristupljeno: 27. kolovoza 2022.

[38] Institute for Astronomy, University of Hawai: „Pan-STARRS“, s Interneta, http://legacy.ifa.hawaii.edu/research/Pan-STARRS.shtml, pristupljeno: 27. kolovoza 2022.

[39] „About Rubin Observatory“, s Interneta, https://www.lsst.org/about, pristupljeno: 27. kolovoza 2022.

[40] Alonso A.: “The State of Data in Astronomy”, s Interneta, https://blog.dataiku.com/the-state-of-data-in-astronomy, pristupljeno: 27. kolovoza 2022.

[41] Kremer, J. i dr.: “Big Universe, Big Data: Machine Learning and Image Analysis for Astronomy”, Department of Computer Science, University of Copenhagen, Universitetsparken 5, 2100 Copenhagen Ø, Denmark, 18. travnja 2017.

[42] Spindler, A.: „How artificial intelligence is changing astronomy“, s Interneta, https://astronomy.com/news/2022/07/how-artificial-intelligence-is-changing-astronomy, pristupljeno: 27. kolovoza 2022.

[43] Hausen, R.; Robertson, B. E.: „Morpheus: A Deep Learning Framework for the Pixel-level Analysis of Astronomical Image Data“, The Astrophysical Journal Supplement Series, American Astronomical Society, svibanj 2020.

[44] Stephens T.: „Powerful new AI technique detects and classifies galaxies in astronomy image data“, s Interneta, https://phys.org/news/2020-05-powerful-ai-technique-galaxies-astronomy.html, pristupljeno: 29. kolovoza 2022.

[45] „Artificial intelligence in space“, s Interneta, https://www.esa.int/Enabling_Support/Preparing_for_the_Future/Discovery_and_Preparation/Artificial_intelligence_in_space, pristupljeno: 29. kolovoza 2022.

[46] S interneta: https://www2.jpl.nasa.gov/magellan/, pristupljeno: 29. kolovoza 2022.

[47] M.C. Burl, L. Asker, P. Smyth, U. Fayyad, P. Perona, L. Crumpler, and J. Aubele, "Learning to Recognize Volcanoes on Venus", Machine Learning, (ožujak 1998).

[48] S interneta: https://volcano.oregonstate.edu/venus, pristuoljeno: 29. kolovoza 2022.

[49] S interneta, https://www2.jpl.nasa.gov/magellan/http:/archive.ics.uci.edu/ml/datasets/volcanoes+on+venus+-+jartool+experiment, pristupljeno: 30. Kolovoza 2022.

[50] V. Roshan, J.: „Optimal ratio for data spliting“, s Interneta, https://onlinelibrary.wiley.com/doi/full/10.1002/sam.11583, pristupljeno 1. rujna 2022.

[51] Shah, D.: „The Essential Guide to Data Augmentation in Deep Learning“, s Interneta, https://www.v7labs.com/blog/data-augmentation-guide, pristupljeno 1. rujna 2022.

[52] Ultralytics: „Tips for Best Training Results“, s Interneta, https://docs.ultralytics.com/tutorials/training-tips-best-results/, pristupljeno 1.rujna 2022.

[53] Ultralytics: „Hyperparameter Evolution“, s Interneta, https://docs.ultralytics.com/tutorials/hyperparameter-evolution/, pristupljeno 1. rujna 2022.

[54] Blackadar, J.:“Tiling and Transcribing a Larger Map“. Digital History Learning Journal, siječanj 2021. godine

POPIS OZNAKA I KRATICA

AI - Artificial Intelligence

AP - Average Precision

CNN - Convolutional Neural Networks

FN - False Negative

FP - False Positive

FPN - Featurey pyramids Networks

IoU - Intersection over Union

ISS - International Space Station

K1 - Klasa 1 vulkana

K2 - Klasa 2 vulkana

K3 - Klasa 3 vulkana

K4 - Klasa 4 vulkana

LSST - Large Synatopic Survey Telescope

mAP - Mean Average Precision

MLP - Multy Layer Percepteron

NMS - Non Maximal Supression

SAR - Synthetic Aperture Radar

SDSS - Sloan Digital Sky Survey

TN - True Negative

TP - True Postive

YOLO - You Only Look Once

POPIS SLIKA

POPIS TABLICA

SAŽETAK I KLJUČNE RIJEČI

Ovim radom opisana je realizacija sustava za detekciju vulkana sa satelitskih snimka planetarne površine Venere. U prvome dijelu objašnjeni su osnovni princip rada i struktura *YOLO* algoritma koji je korišten kao metoda detekcije objekata. Zatim su prezentirane metode pripreme i obrade Magellan seta podataka u svrhu učenja neuronske mreže. Nakon toga su prezentirani problemi s kojima se suočilo zbog relativno malog seta podataka, te su objašnjena rješenja s kojima se uspjelo doći do zadovoljavajućih rezultata modela. Na samome kraju predstavljena su idejna rješenja za buduća istraživanja u svrhu poboljšanja preciznosti modela kod detekcije jako malih objekata.

Ključne riječi: Umjetna inteligencija, detekcija objekata, vulkani na Veneri, YOLO

ABSTRACT AND KEYWORDS

This work describes the implementation of a volcano detection system with a satellite image of the planetary surface of Venus. In the first part, the basic working principle and structure of the YOLO algorithm, which is used as an object detection method, are explained. Then the methods of preparation and processing of the Magellan dataset for the purpose of neural network learning are presented. After that, the problems faced due to a relatively small set of data were presented, and the solutions with which it was possible to achieve satisfactory model results were explained. At the very end, conceptual solutions for future research are presented in order to improve the precision of the model for detection of very small objects.

Key words: Artificial intelligence, object detection, volcanoes on Venus, YOLO

DODATAK A – *vread.py* skripta

```
"""
-vread('imgx')

-ucitava .spr i .sdt datoteke u Python i sprema ih u .png datoteku
zahtjeva da se te dvije vrste datoteka nalazet u istom direktoriju

22.3.2022.
@author: DĐ
"""

import numpy as np
import matplotlib as plt
import csv
import shutil
import os

def vread(filename):
    pfile = filename + '.spr' #header datoteka - sadrzi znacajke slike
    dfile = filename + '.sdt' #sadrzi binarne podatke slike

    #ucitava podatke iz header (.spr) datoteke
    spr_data = []
    with open(pfile) as fobj:
        for line in fobj:
            row = line.split()
            spr_data.append(row[-1])

    #Izvlaci znacajke slike iz header datoteke
    ndim = int(spr_data[0])#1D ili 2D ili 3D...
    nc = int(spr_data[1]) #broj stupaca
    nr = int(spr_data[4]) #broj redaka
    n_type = int(spr_data[7])#tip podataka

    if (ndim != 2):
        print('Can only read two dimensional data!')
        return None

    #ovisno o tipu podataka formira sliku iz .sdt datoteke
    #usigned char
    if (n_type == 0):
        with open(dfile, 'rb') as fid:
            data_array = np.fromfile(fid, np.ubyte).reshape((nr,nc))
    #integer
    elif (n_type == 2):
        with open(dfile, 'rb') as fid:
            data_array = np.fromfile(fid, np.intc).reshape((nr,nc))
    #float
    elif (n_type == 3):
        with open(dfile, 'rb') as fid:
            data_array = np.fromfile(fid, np.single).reshape((nr,nc))
    #double
    elif (n_type == 5):
        with open(dfile, 'rb') as fid:
            data_array = np.fromfile(fid, np.double).reshape((nr,nc))

    else:
      print('Unrecognized data type')
      data_array = 0
```

```
    return data_array

def main():
    #postavlja pathove
    parent_dir = os.getcwd()
    imgs_dir = os.path.join(parent_dir, 'imgs') #direktorij slika
    mag_dir = os.path.join(parent_dir, 'mag_data') #direktorij od Magellana

    os.chdir(mag_dir)

    files = [os.path.join(mag_dir, x) for x in os.listdir(mag_dir)]
    img = [0]*int(len(files)/2)
    i = 0
    for f in files:
        file_str = f.rpartition('\\')
        file_str = file_str[-1].rpartition('.')
        if(file_str[-1] == 'spr'):
            img[i] = vread(file_str[0])
            plt.pyplot.imsave(file_str[0] + '.png',img[i] , cmap=plt.cm.gray)
            print(file_str[0] + ' done!')
            i += 1
        else:
            continue

    #mice datoteke u "imgs" direktorij
    for filename in os.listdir(mag_dir):
        f = filename[-3:] #'png'
        if f == 'png':
            shutil.move(os.path.join(mag_dir, filename), imgs_dir)

    print('-----All done!-----')

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

DODATAK B - *lxyrConvert2yolo.py* skripta

```
"""
-lxyrConvert2Yolo('imgx', [1024,1024])
-ova skripta pretvara .lxyr ground truth datoteke iz Magellan dataseta
u YOLO format labele
-YOLO format: [class, x_center, y_center, width, height] normalizirane [0,1]

24.3.2022.
@author: DĐ
"""

import numpy as np
import warnings
import os
import shutil

def lxyrConvert2Yolo(filename, img_size):
    file = filename + '.lxyr'

    #ignorira "empty text file" upozorenje od np.loadtxt()
    with warnings.catch_warnings():
        warnings.simplefilter("ignore")
        lxyr_data = np.loadtxt(file)

    #vraca None ako nema mogucih vulkana u slici
    if(len(lxyr_data) == 0):
        return None
    else:
        #provjerava broj redova u .lxyr datoteci (broj redaka = broj mogucih
vulkana)
        try:
            nov = (lxyr_data.shape[0], lxyr_data.shape[1])
        except IndexError:
            nov = (1, lxyr_data.shape[0])

        #cita 1D vektore
        if(nov[0] == 1):
            v_class  = np.array(lxyr_data[0]) #klasa vulkana
            x_cord = np.array(lxyr_data[1]) #x koordinate
            y_cord = np.array(lxyr_data[2]) #y koordinate
            radius = np.array(lxyr_data[3]) #radijus kruga

        #cita 2D vektore
        else:
            v_class  = np.array(lxyr_data[:,0]) #klasa vulkana
            x_cord = np.array(lxyr_data[:,1]) #x koordinate
            y_cord = np.array(lxyr_data[:,2]) #y koordinate
            radius = np.array(lxyr_data[:,3]) #radijus kruga

        #pretvara u YOLO format
        #[class, x_center, y_center, width, height] svi normalizirani [0,1]
        #u ovom slucaju -> width = height (bounding box je kvadrat)
        ground_truth = []
        #privremeno = np.zeros([nov[0],1])
        ground_truth = np.append(ground_truth, v_class-1)#YOLO klase startaju od
nule
        #ground_truth = np.append(ground_truth, privremeno)

        x_center = x_cord/img_size[0]
```

```
        y_center = y_cord/img_size[1]
        ground_truth = np.c_[ground_truth, x_center]
        ground_truth = np.c_[ground_truth, y_center]

        width = np.empty([nov[0],1])
        height = np.empty([nov[0],1])


        if(nov[0] == 1):
            width[0] = np.round(radius)*2/img_size[0]
            height[0] = np.round(radius)*2/img_size[1]
        else:
            for i in range(0,nov[0]):
                width[i] = np.round(radius[i])*2/img_size[0]
                height[i] = np.round(radius[i])*2/img_size[1]

        ground_truth = np.c_[ground_truth, width]
        ground_truth = np.c_[ground_truth, height]

    return ground_truth

#petlja prolazi kroz sve datoteke u "lxyr_abels" direktoriju, prebacuje ih u
#YOLO format i mice ih u "yolo_labels" direktorij
def main():
    #postavlja pathove
    parent_dir = os.getcwd()
    lxyr_dir = os.path.join(parent_dir, 'lxyr_labels')
    yolo_dir = os.path.join(parent_dir, 'yolo_labels')

    os.chdir(lxyr_dir)

    files = [os.path.join(lxyr_dir, x) for x in os.listdir(lxyr_dir)]
    n = 0
    for f in files:
        if(f[-4:] == 'lxyr'):
            n += 1

    gt = [0]*n
    i = 0
    for f in files:
        file_str = f.rpartition('\\')
        file_str = file_str[-1].rpartition('.')
        if(file_str[-1] == 'lxyr'):
            gt[i] = lxyrConvert2Yolo(file_str[0], [1024,1024])
            if(gt[i] is None):
                gt[i] = []
                np.savetxt(file_str[0] + '.txt', gt[i],fmt = '%f')

            else:
                np.savetxt(file_str[0] + '.txt', gt[i], fmt = '%f')

            print(file_str[0] + ' done!')
            i += 1
        else:
            continue

    #mice datoteke u "imgs" direktorij
    for filename in os.listdir(lxyr_dir):
        f = filename[-3:] #'txt'
        if f == 'txt':
```

```
            shutil.move(os.path.join(lxyr_dir, filename), yolo_dir)

    print('-----All done!-----')

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

DODATAK C - *createYoloDataset.py* skripta

```
# -*- coding: utf-8 -*-
"""
-Ova skripta sadrži rutinu koja kreira dataset u YOLO strukturi i razdvaja
ga u "training",validation" i "testing"  djelove u definiranom omjeru
-slike se moraju nalaziti "images" direktoriju
-YOLO labeli se moraju nalaziti u "labels" direktoriju
-createYoloDataSet.py mora se nalaziti u istom direktoriju kao i "images" i
"labels" folderi

- YOLO  dataset struktura:

--datasets
      .
      .--ime_dataseta
              .
              .--train
              .     .
              .     .--images
              .     .--labels
              .
              .--valid
              .     .
              .     .--images
              .     .--labels
              .
              .--test
                    .
                    .--images
                    .--labels

24.3.2022.
@author: DĐ
"""
from pathlib import Path
import os
import shutil
from sklearn.model_selection import train_test_split

#ako ne postoji neki od ovih direktorija stvara istoimeni direktorij
def create_data_set(data_set_name):
    Path(f'datasets').mkdir(parents=True, exist_ok=True)
    Path(f"datasets/{data_set_name}/train/images").mkdir(parents=True,
exist_ok=True)
    Path(f"datasets/{data_set_name}/train/labels").mkdir(parents=True,
exist_ok=True)
    Path(f"datasets/{data_set_name}/test/images").mkdir(parents=True,
exist_ok=True)
    Path(f"datasets/{data_set_name}/test/labels").mkdir(parents=True,
exist_ok=True)
    Path(f"datasets/{data_set_name}/valid/images").mkdir(parents=True,
exist_ok=True)
    Path(f"datasets/{data_set_name}/valid/labels").mkdir(parents=True,
exist_ok=True)

def main():
    #Pitaj za ime dataseta
```

```
    data_set_name = input("Enter your data set name: ")

    #Pitaj za omjer podjele
    print('Enter your teaining/valid/test ratios (fe. test = 0.1)')
    split = [0]*3
    split[0] = float(input('Training: '))
    split[1] = float(input('Validation: '))
    split[2] = float(input('Testing: '))

    #Ucitaj slike i labele
    parent_dir = os.getcwd()
    img_dir = os.path.join(parent_dir, 'images')
    lb_dir = os.path.join(parent_dir, 'labels')
    images = [os.path.join(img_dir, x) for x in os.listdir(img_dir)]
    labels = [os.path.join(lb_dir, x) for x in os.listdir(lb_dir)]

    #razdvaja ih u training  i testing/validation (zajednicki) setove
    xtv = 1 - split[0]
    train_images, vt_images, train_labels, vt_labels = train_test_split(images,
labels, test_size = xtv)

    #razdvaja testing/validation zajednicki set
    #u testing i validation zasebne setove
    xt = split[2]/(1-split[0])
    val_images, test_images, val_labels, test_labels = train_test_split(vt_images,
vt_labels, test_size = xt)

    #kreira direktorije za novi dataset
    create_data_set(data_set_name)

    #Mice ih u njihove direktorije
    for filename in train_images:
        shutil.copy(filename, Path(f'datasets/{data_set_name}/train/images'))

    for filename in train_labels:
        shutil.copy(filename, Path(f'datasets/{data_set_name}/train/labels'))

    for filename in test_images:
        shutil.copy(filename, Path(f'datasets/{data_set_name}/test/images'))

    for filename in test_labels:
        shutil.copy(filename, Path(f'datasets/{data_set_name}/test/labels'))

    for filename in val_images:
        shutil.copy(filename, Path(f'datasets/{data_set_name}/valid/images'))

    for filename in val_labels:
        shutil.copy(filename, Path(f'datasets/{data_set_name}/valid/labels'))

    print('---All done!---')

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

DODATAK D – *AugmentFunctions.py* skripta

```
# -*- coding: utf-8 -*-
"""
-Ova skripta sadrzi sve funkcije potrebne za augmentaciju dataseta

7.4.2022.
@author: DĐ
"""
import random
import numpy as np
import cv2
import matplotlib.pyplot as plt
import sys
import os
import warnings

#provjerava broj redova u .lxyr datoteci (broj redaka = broj objekata na slici)
def check4NO(label):
    if(len(label)==0):
        no = 0
    else:
        try:
            no = (label.shape[0], label.shape[1])
            no = no[0]
        except IndexError:
            no = (1, label.shape[0])
            no = no[0]
    return no

#-------------------------flipVertical---------------------------------------
#Zrcali sliku vertikalno i racuna nove bounding box koordinate

def flipVertical(img,label,p):
    if(random.random() <= p):
        #zrcaljenje slike
        h = img.shape[0]
        f_img = cv2.flip(img, 0)

        #racunanje novih bounding box koordinata
        #width i height ostaju isti nakon ove augmentacije
        no = check4NO(label)
        f_label = label
        if(no == 0):
            f_label = None
        elif(no == 1):
            f_label[2] = 1 - label[2] - 1/h #f_yc
            #f_xc = xc
        else:
            for i in range(0,no):
                f_label[i,2] = 1 - label[i,2] - 1/h #f_yc
            #f_xc = xc
    else:
        f_img = None
        f_label = None

    return f_img, f_label
```

```
#-------------------------flipHorizontal-------------------------------------
#Zrcali sliku horizontalno i racuna nove bounding box koordinate

def flipHorizontal(img,label,p):
    if(random.random() <= p):
        #zrcaljenje slike
        w = img.shape[1]
        f_img = cv2.flip(img, 1)

        #racunanje novih bounding box koordinata
        #width i height ostaju isti nakon ove augmentacije
        no = check4NO(label)
        f_label = label
        if(no == 0):
            f_label = None
        elif(no == 1):
            f_label[1] = 1 - label[1] - 1/w #f_xc
        else:
            for i in range(0,no):
                f_label[i,1] = 1 - label[i,1] - 1/w #f_xc
                #f_yc = yc
    else:
        f_img = None
        f_label = None

    return f_img, f_label

#-------------------------rotateCoord----------------------------------------
#pomice koordinatni sustav u sredinu -> rotira ga -> vraca ga nazad u kut
#objasnjeno vise u "Augmentacija.pdf" datoteci

def rotateCoord(x,y,angle):
    a_rad = np.deg2rad(angle)
    x_rot = (x-0.5)*np.cos(a_rad) + (y-0.5)*np.sin(a_rad) + 0.5
    y_rot = -(x-0.5)*np.sin(a_rad) + (y-0.5)*np.cos(a_rad) + 0.5

    return x_rot, y_rot

#-----------------------------rotate-------------------------------------------
#Rotira sliku za odredjeni kut (u stupnjevima) i racuna nove koordinate
#bounding boxa, width i height mijenjaju dimenziju nakon ove augmentacije.
#Trenutno radi za rotacije od 90,180 i 270 stupnjeva, prilikom ostalih rotacija
#koordinate bounding boxa nisu tocne posto dolazi do promjene aspect ratia
#slike kako nebi bila cropana.
#objasnjeno vise u "Augmentacija.pdf" datoteci

def rotate(img, label, angle, p):
    if(random.random() <= p):
        #uzima dimenzije slike i odreduje koordinate centra
        (h, w) = img.shape[:2]
        (cX, cY) = (w // 2, h // 2)

        #racuna matricu rotacije
        M = cv2.getRotationMatrix2D((cX, cY), angle, 1.0)
        cos = np.abs(M[0, 0])
        sin = np.abs(M[0, 1])
        #izračunava nove granične dimenzije slike
        nW = int((h * sin) + (w * cos))
        nH = int((h * cos) + (w * sin))
        #prilagodjava matricu rotacije kako bi uzeo translaciju u obzir
```

```
        M[0, 2] += (nW / 2) - cX
        M[1, 2] += (nH / 2) - cY
        # perform the actual rotation and return the image
        r_img = cv2.warpAffine(img, M, (nW, nH))

        #racuna nove koordinate bounding boxa
        no = check4NO(label)
        r_label = label
        if(no == 0):
            f_label = None
        elif(no == 1):#ako se nalazi samo jedan objekt u slici
            #ucitava bounding box-coordinate
            x_min = label[1] - label[3]/2
            y_min = label[2] - label[4]/2

            x_max = label[1] + label[3]/2
            y_max = label[2] + label[4]/2

            #rotacija koordinata centra
            r_label[1], r_label[2] = rotateCoord(label[1],label[2],angle)

            #rotacija koordinata vrhova bounding boxa (kako bi se izracunali
            #novi width i height)
            x_min_rot, y_min_rot = rotateCoord(x_min,y_min,angle)
            x_max_rot, y_max_rot = rotateCoord(x_max,y_max,angle)

            #nove dimenzije bounding boxa (w i h)
            label[3] = abs(x_max_rot - x_min_rot)
            label[4] = abs(y_max_rot - y_min_rot)
        else:
            for i in range(0,no):#ako ima vise objekata na slici
                #ucitava bounding box-coordinate
                x_min = label[i,1] - label[i,3]/2
                y_min = label[i,2] - label[i,4]/2

                x_max = label[i,1] + label[i,3]/2
                y_max = label[i,2] + label[i,4]/2

                #rotacija koordinata centra
                r_label[i,1], r_label[i,2] =
rotateCoord(label[i,1],label[i,2],angle)

                #rotacija koordinata vrhova bounding boxa (kako bi se izracunali
                #novi width i height)
                x_min_rot, y_min_rot = rotateCoord(x_min,y_min,angle)
                x_max_rot, y_max_rot = rotateCoord(x_max,y_max,angle)

                #nove dimenzije bounding boxa (w i h)
                label[i,3] = abs(x_max_rot - x_min_rot)
                label[i,4] = abs(y_max_rot - y_min_rot)
    else:
        r_img = None
        r_label = None

    return r_img, r_label

#--------------------------calculateCoveredArea------------------------------
#Racuna koliko "cutout boxovi" prekrivaju bounding box u [%]
#objasnjeno vise u "Augmentacija.pdf" datoteci
```

```
def calculateCoveredArea(label,rect_map,numb,h,w):
   #ucitava bounding box-coordinate
    x_0 = round(round(label[1]*w) - round(label[3]*w)/2)
    y_0 = round(round(label[2]*h) - round(label[4]*h)/2)

    x_1 = round(round(label[1]*w) + round(label[3]*w)/2)
    y_1 = round(round(label[2]*h) + round(label[4]*h)/2)

    bbox_coords = [x_0,y_0,x_1,y_1]
    #print('bbox_coords: ', bbox_coords)

    #racuna povrsinu bounding boxa
    bbox_area = abs(y_1-y_0)*abs(x_1-x_0)
    #print('bbox_area: ', bbox_area)

    #mapira bounding box u matricu
    X=np.zeros((h,w))

X[int(bbox_coords[1]):int(bbox_coords[3]),int(bbox_coords[0]):int(bbox_coords[2])]
= 1
    rect_map[0] = X

    #racuna presjek svih cutout boxova i bounding boxa
    suma_bbox = np.zeros((h,w))
    for i in range(0,numb+1):
        suma_bbox += rect_map[i]

    IOU_bbox = np.sum(suma_bbox > 1)

    #racuna presjek svih cutout boxova bez bounding boxa
    suma_wo_bbox = np.zeros((h,w))
    for j in range(1,numb+1):
        suma_wo_bbox += rect_map[j]

    IOU_wo_bbox = np.sum(suma_wo_bbox > 1)

    #Racuna presjek svakog zasebnog cutout boxa sa bounding boxom
    IOU_values = [0]*(numb)
    for k in range(0,numb):
        IOU_values[k] = np.sum((rect_map[0] + rect_map[k+1]) > 1)

    #racuna vidljivu povrsinu bounding boxa
    area  = (IOU_bbox - IOU_wo_bbox + np.sum(IOU_values))/2
    c_area = area/bbox_area

    return c_area

#--------------------------cutout--------------------------------------------
#rutina koja crta odredjeni broj cutout boxova koji imaju maksimalnu dimenziju
#te prise bounding boxove cija je povrsina prekrivena vise od 50%
#objasnjeno vise u "Augmentacija.pdf" datoteci

def cutout(img, label, numb, max_dim, p):
    if(random.random() <= p):
        #definiranje osnovnih parametara
        color = (0,0,0) #crna boja
        thickness = -1 #ispunjuje cijelu oblik
        (h, w) = img.shape[:2] #dimenzije slike
        del_list = [] #lista za brisanje
        no = check4NO(label) #broj objekata
```

```
        c_label=label

        rect_coord = np.zeros((numb, 4)) #zapis koordinata svih cutout boxova
        for i in range(0,numb):
            #definira nasumicne koordinate cutout boxa
            x_min = random.randrange(0,w)
            y_min = random.randrange(0,h)
            x_max = x_min + random.randrange(1,max_dim)
            y_max = y_min + random.randrange(1,max_dim)

            #max koordinate ne mogu biti vece od dimenzije slika
            if(x_max > w):
                x_max = w
            if(y_max > h):
                y_max = h

            #crta cutoutbox na sliku
            start_point = (x_min,y_min)
            end_point = (x_max,y_max)
            c_img = cv2.rectangle(img, start_point, end_point, color, thickness)

            #sprema koordinate cutout boxa
            rect_coord[i] = [x_min,y_min,x_max,y_max]

        #print('rect_coord: ',rect_coord)
        #mapira sve cutout boxove u matrice
        rect_map = [0]*(numb+1)
        for j in range(0,numb):
            X=np.zeros((h,w))

X[int(rect_coord[j,1]):int(rect_coord[j,3]),int(rect_coord[j,0]):int(rect_coord[j,
2])] = 1
            rect_map[j+1] = X

        if(no == 0):#ako nema objekata
            c_label = None
        elif(no == 1):#ako je jedan objekt
            c_area = calculateCoveredArea(label, rect_map, numb, h, w)
            #print('c_area: ', c_area)
            #ako je prekrivena povrsina veca od 50% -> dodaj na listu za brisanje
            if(c_area > 0.5):
                del_list.append(0)
        else:#ako ima vise objekata
            for k in range(0,no):
                c_area = calculateCoveredArea(label[k], rect_map, numb, h, w)
                #print('c_area: ', c_area)
                #ako je prekrivena povrsina veca od 50% -> dodaj na listu za
brisanje
                if(c_area > 0.5):
                    del_list.append(k)

        #brise bounding boxove cija je povrsina prekrivena vise od 50%
        del_list = list(set(del_list))
        if(len(del_list) != 0):
            if(check4NO(label)==1 and del_list[0] == 0):
                c_label = None
            else:
                c_label = np.delete(c_label,del_list,axis = 0)
                if(check4NO(c_label)==0):
                    c_label = None
```

```
    else:
        c_label = None
        c_img = None

    return c_img, c_label

#----------------------applyBrightnesContrast---------------------------------
#brightnes = [-127,127]
#contrast = [-64,64]

def applyBrightnesContrast(input_img, label, brightness, contrast, p):
    if(random.random() <= p):
        if brightness != 0:
            if brightness > 0:
                shadow = brightness
                highlight = 255
            else:
                shadow = 0
                highlight = 255 + brightness
            alpha_b = (highlight - shadow)/255
            gamma_b = shadow

            buf = cv2.addWeighted(input_img, alpha_b, input_img, 0, gamma_b)
        else:
            buf = input_img.copy()

        if contrast != 0:
            f = 131*(contrast + 127)/(127*(131-contrast))
            alpha_c = f
            gamma_c = 127*(1-f)

            buf = cv2.addWeighted(buf, alpha_c, buf, 0, gamma_c)

        img_out = buf
        label_out = label
    else:
        img_out = None
        label_out = None


    return img_out, label_out

#-----------------------------zoom_center-------------------------------------
def zoom_center(img, zoom_factor=1.5):

    y_size = int(img.shape[0])
    x_size = int(img.shape[1])

    # define new boundaries
    x1 = round((0.5*x_size)*(1-1/zoom_factor))
    x2 = round(x_size-(0.5*x_size)*(1-1/zoom_factor))
    y1 = round((0.5*y_size)*(1-1/zoom_factor))
    y2 = round(y_size-(0.5*y_size)*(1-1/zoom_factor))

    # first crop image then scale
    img_cropped = img[y1:y2,x1:x2]
    return cv2.resize(img_cropped, None, fx=zoom_factor, fy=zoom_factor)

#---------------------------ZoomObject----------------------------------------
```

```
def zoomObject(img,label,zoom_scale = 1.5, p = 1):
    if(random.random() <= p):
        new_label = label

        (h,w) = img.shape[:2]
        tx = w/2 - (label[1]*w)
        ty = h/2 - (label[2]*h)

        M = np.float32([[1,0,tx],[0,1,ty]])
        dst = cv2.warpAffine(img,M,(h,w))
        zoomed = zoom_center(dst,zoom_scale)

        new_label[1] = 0.5
        new_label[2] = 0.5
        new_label[3] = label[3]*zoom_scale
        if(new_label[3] > 1):
            new_label[3] = 1
        new_label[4] = label[4]*zoom_scale
        if(new_label[4] > 1):
            new_label[4] = 1

    else:
        zoomed = None
        new_label = None

    return zoomed, new_label
```

DODATAK E – *AugmentMain.py* skripta

```
# -*- coding: utf-8 -*-
"""
-ova skripta sadrzi rutinu za augmentaciju dataseta
-zahtjeva da se slike nalaze u "images" direktoriju, a labeli u "labels"
direktoriju
-augmentirane slike i labeli pojavljuju se u "aug_images" i "aug_labels"
direktorijima
-config.txt datoteka sadrzi parametre augmentacije kao sto su:
    -p -> vjerojatnost da se ta augmentacija desi
    -numb -> broj cutout boxeva
    -max_dim -> maksimalna dimenzija cutout boxeva

9.5.2022.
@author: DĐ
"""
import AugmentFunctions as af
import random
import numpy as np
import cv2
import matplotlib.pyplot as plt
import sys
import os
import warnings

#postavlja pathove direktorija
parent_dir = os.getcwd()
img_dir = os.path.join(parent_dir, 'images')
lb_dir = os.path.join(parent_dir, 'labels')
augImg_dir = os.path.join(parent_dir, 'aug_images')
augLb_dir = os.path.join(parent_dir, 'aug_labels')
config_dir = os.path.join(parent_dir, 'config.txt')

#rutina jedne augmentacije
def performAugmentation(images, labels, img_ext, augmentation, numb, max_dim, p):
    img = [0]*len(images)
    label = [0]*len(images)

    new_img = [0]*len(images)
    new_label = [0]*len(images)

    for i in range(0,len(images)):
        img[i] = cv2.imread(images[i])

        #ignorira "empty text file" upozorenje od np.loadtxt()
        with warnings.catch_warnings():
            warnings.simplefilter("ignore")
            label[i] = np.loadtxt(labels[i])

        #ime slike
        img_str = images[i].rpartition('\\')
        img_str = img_str[-1].rpartition('.')

        #odradjuje augmentaciju
        if(augmentation == 0):
            new_img[i], new_label[i] = af.flipVertical(img[i], label[i], p)
        elif(augmentation == 1):
```

```
            new_img[i], new_label[i] = af.flipHorizontal(img[i], label[i], p)
        elif(augmentation == 2): #r90
            new_img[i], new_label[i] = af.rotate(img[i], label[i], 90, p)
        elif(augmentation == 3): #r180
            new_img[i], new_label[i] = af.rotate(img[i], label[i], 180, p)
        elif(augmentation == 4): #r270
            new_img[i], new_label[i] = af.rotate(img[i], label[i], 270, p)
        elif(augmentation == 5): #cutout
            new_img[i], new_label[i] = af.cutout(img[i], label[i], numb, max_dim,
p)
        elif(augmentation == 6): #brightnes
            brightness = random.randint(-127, 127)
            if((brightness > -50) and (brightness < 127)):
                brightness = -100
            new_img[i], new_label[i] = af.applyBrightnesContrast(img[i], label[i],
brightness,1, p)
        elif(augmentation == 7): #contrast
            contrast = random.randint(-64,64)
            if((contrast > -64) and (contrast < 25)):
               contrast = 25
            new_img[i], new_label[i] = af.applyBrightnesContrast(img[i], label[i],
1, contrast, p)
        elif(augmentation == 8): #zoom
            if(af.check4NO(label[i]) == 0):
                new_img[i] = None
                new_label[i] = None
            elif(random.random() > p):
                new_img[i] = None
                new_label[i] = None
            else:
                for j in range(0, len(label[i])):
                    l = label[i]
                    if(af.check4NO(l) != 1):
                        l = l[j]

                    if(l[0] == 0):
                        new_img[i], new_label[i] = af.zoomObject(img[i], l, 5,
0.35)
                    elif(l[0] == 1):
                        new_img[i], new_label[i] = af.zoomObject(img[i], l, 7,
0.35)
                    elif(l[0] == 2):
                        new_img[i], new_label[i] = af.zoomObject(img[i], l, 10,
0.35)
                    elif(l[0] == 3):
                        new_img[i], new_label[i] = af.zoomObject(img[i], l, 15,
0.35)
                    else:
                        new_img[i] = None
                        new_label[i] = None

                    if((new_img[i] is None) and (new_label[i] is None)):
                        continue
                    else:
                        #ime nove slike
                        new_img_name =  img_str[0] + img_ext + str(j+1)

                        if(new_label[i] is None):
                            open(augLb_dir+'\\'+new_img_name+'.txt', 'w').close()
                        elif(af.check4NO(new_label[i]) == 1):
```

```
                            np.savetxt(augLb_dir+'\\'+new_img_name+'.txt',
np.array(new_label[i]).reshape(1, np.array(new_label[i]).shape[0]), fmt = '%f')

                        cv2.imwrite(augImg_dir+'\\'+new_img_name+'.jpg',
new_img[i])
                        print(new_img_name + ' done!')


        if((new_img[i] is None) and (new_label[i] is None)):
            continue
        elif(augmentation == 8):
            continue
        else:
            #ime nove slike
            new_img_name =  img_str[0] + img_ext

            if(new_label[i] is None):
                open(augLb_dir+'\\'+new_img_name+'.txt', 'w').close()
            elif(af.check4NO(new_label[i]) == 1):
                if(augmentation == 5):
                    np.savetxt(augLb_dir+'\\'+new_img_name+'.txt',
np.array(new_label[i]).reshape(1, np.size(new_label[i])), fmt ='%f')
                else:
                    np.savetxt(augLb_dir+'\\'+new_img_name+'.txt',
np.array(new_label[i]).reshape(1, np.array(new_label[i]).shape[0]), fmt = '%f')
            else:
                np.savetxt(augLb_dir+'\\'+new_img_name+'.txt',new_label[i],fmt =
'%f')

            cv2.imwrite(augImg_dir+'\\'+new_img_name+'.jpg', new_img[i])
            print(new_img_name + ' done!')

def main():
    #ucitava slike i labele
    images = [os.path.join(img_dir, x) for x in os.listdir(img_dir)]
    labels = [os.path.join(lb_dir, x) for x in os.listdir(lb_dir)]

    #ucitava config datoteku
    config = np.loadtxt(config_dir)

    #flipVertical
    performAugmentation(images, labels, 'fv', 0, int(config[6]), int(config[7]),
config[0])
    #flipHorizontal
    performAugmentation(images, labels, 'fh', 1, int(config[6]), int(config[7]),
config[1])
    #rotate90
    performAugmentation(images, labels, 'r90', 2, int(config[6]), int(config[7]),
config[2])
    #rotate180
    performAugmentation(images, labels, 'r180', 3, int(config[6]), int(config[7]),
config[3])
    #rotate270
    performAugmentation(images, labels, 'r270', 4, int(config[6]), int(config[7]),
config[4])
    #Cutout
    performAugmentation(images, labels, 'c', 5, int(config[6]), int(config[7]),
config[5])
    #Brightnes
```

```
    performAugmentation(images, labels, 'br', 6, int(config[6]), int(config[7]),
config[8])
    #Contrast
    performAugmentation(images, labels, 'cont', 7, int(config[6]), int(config[7]),
config[9])
    #Zoom
    performAugmentation(images, labels, 'z', 8, int(config[6]), int(config[7]),
config[10])

    print('-----All done!-----')

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

DODATAK F – *Google Colab* skripta za treniranje *YOLOv5* modela

```
# -*- coding: utf-8 -*-
"""
#Treniranje YOLOv5 algoritma preko Jupyter Notebook-a
#Instaliranje potrebnih paketa
"""

# Commented out IPython magic to ensure Python compatibility.
# kloniranje yolov5 repozitorija
!git clone https://github.com/ultralytics/yolov5
# %cd yolov5
!git reset --hard fbe67e465375231474a2ad80a4389efc77ecff99

# install dependencies as necessary
!pip install -qr requirements.txt  # install dependencies (ignore errors)
import torch

from IPython.display import Image, clear_output  # to display images
from utils.downloads import attempt_download  # to download models/datasets

# clear_output()
print('Setup complete. Using torch %s %s' % (torch.__version__,
torch.cuda.get_device_properties(0) if torch.cuda.is_available() else 'CPU'))

"""# Uvoz dataseta sa Roboflowa ili preko Google Drivea

### **Roboflow**
"""

#follow the link below to get your download code from from Roboflow
!pip install -q roboflow
from roboflow import Roboflow
rf = Roboflow(model_format="yolov5", notebook="roboflow-yolov5")

# Commented out IPython magic to ensure Python compatibility.
#Potrebno uzeti APY_KEY sa roboflow stranice dok se eksporta generirani dataset,
#te zamnjeniti ovu liniju koda

# %cd /content/yolov5

rf = Roboflow(api_key="YOUR API KEY HERE")
project = rf.workspace().project("YOUR PROJECT")
dataset = project.version("YOUR VERSION").download("yolov5")

"""#**Google Drive**

*Mounta se Drive pod direktorijem: "/content/drive/MyDrive/..." te tamo pronaći
poveznicu na .yaml datoteku od dataseta.

*u .yaml datoteci definirati pathove za train, valid  i test foldere (images)

"""

from google.colab import drive
drive.mount('/content/drive')

"""#**Treniranje Modela**
```

```
*ukoliko se dataset nalazi na Google Driveu pod --data kopirati path do .yaml
filea od dataseta

"""

# Commented out IPython magic to ensure Python compatibility.
# %%time
# %cd /content/yolov5/
# !python train.py --img 1024 --batch 16 --epochs 100 --data
{dataset.location}/data.yaml --cfg /content/yolov5/models/yolov5m6.yaml --weights
yolov5m6.pt --name yolov5m6_results  --cache

"""#**Validacija modela**

"""

!python val.py --data {dataset.location}/data.yaml --weights
/content/yolov5/runs/train/yolov5m6_results/weights/best.pt --img 1024

"""#**Detekcija**"""

!python detect.py –-weights
/content/yolov5/runs/train/yolov5m6_results/weights/best.pt –-img 1024 –-conf 0.4
–-source {izvor slike}
```

DODATAK G – Hiperparametri osnovnog modela

Hiperparametar	Vrijednost
lr0	*0.01*
lrf	*0.01*
momentum	*0.937*
weight_decay	*0,0005*
warmup_epochs	*3.0*
warmup_momentum	*0.8*
warmup_bias_lr	*0.1*
box	*0.05*
cls	*0.5*
cls_pw	*1.0*
obj	*1.0*
obj_pw	*1.0*
iou_t	*0.2*
anchor_t	*4.0*
fl_gamma	*0.0*
hsv_h	*0.015*
hsv_s	*0.7*
hsv_v	*0.4*
degrees	*0.0*
translate	*0.1*
scale	*0.5*
shear	*0.0*
perspective	*0.0*
flipud	*0.0*
fliplr	*0.5*
mosaic	*1.0*
mixup	*0.0*
copy_paste	*0.0*

DODATAK H– Idejna implementacija tiling-a

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jun 15 21:12:42 2022

@author: DĐ
"""
import numpy as np
import cv2
import os
import warnings

#provjerava broj redova u .lxyr datoteci (broj redaka = broj objekata na slici)
def check4NO(label):
    if(len(label)==0):
        no = 0
    else:
        try:
            no = (label.shape[0], label.shape[1])
            no = no[0]
        except IndexError:
            no = (1, label.shape[0])
            no = no[0]
    return no

#Podijeli sliku i label na 4 dijela te vraca liste sa 4 slike i labela
def splitImage(img, label):
    zoom_factor = 2
    #Baratanje slikama
    M = img.shape[0]//2
    N = img.shape[1]//2
    tiles = [img[x:x+M,y:y+N] for x in range(0,img.shape[0],M) for y in
range(0,img.shape[1],N)]

    #svaki "tile" resize-a na originalnu dimenziju slike (2x se ustvari)
    #originalna slika povecala
    tiles_r = [0]*4
    tiles_r[1] = cv2.resize(tiles[1], [zoom_factor*M, zoom_factor*N])
    tiles_r[0] = cv2.resize(tiles[0], [zoom_factor*M, zoom_factor*N])
    tiles_r[2] = cv2.resize(tiles[2], [zoom_factor*M, zoom_factor*N])
    tiles_r[3] = cv2.resize(tiles[3], [zoom_factor*M, zoom_factor*N])

    #Baratanje labelima
    label_I = []
    label_II = []
    label_III = []
    label_IV = []
    shift  = np.array([[0.5,0],[0,0],[0,0.5],[0.5,0.5]]) #x_shift, y_shift

    if(check4NO(label) == 0):
        pass
    if(check4NO(label) == 1):
        label_buffer = label

        #Dimenzije bboxa se povecaju 2 puta (obrnuto proporcionalan)
        label_buffer[3] =  label_buffer[3]*zoom_factor
        label_buffer[4] =  label_buffer[4]*zoom_factor
```

```
    #Provjera kvadranta koordinata
    if(label[1] > 0.5 and label[2] <= 0.5): #I kvadrant
        label_buffer[1] = (label[1] - shift[0,0])*zoom_factor
        label_buffer[2] = (label[2] - shift[0,1])*zoom_factor
        label_I.append(label_buffer)
    elif(label[1] <= 0.5 and label[2] <= 0.5): #II kvadrant
        label_buffer[1] = (label[1] - shift[1,0])*zoom_factor
        label_buffer[2] = (label[2] - shift[1,1])*zoom_factor
        label_II.append(label_buffer)
    elif(label[1] <= 0.5 and label[2] > 0.5): #III kvadrant
        label_buffer[1] = (label[1] - shift[2,0])*zoom_factor
        label_buffer[2] = (label[2] - shift[2,1])*zoom_factor
        label_III.append(label_buffer)
    else: #IV kvadrant
        label_buffer[1] = (label[1] - shift[3,0])*zoom_factor
        label_buffer[2] = (label[2] - shift[3,1])*zoom_factor
        label_IV.append(label_buffer)

else:
    for i in range(0, len(label)):
        label_buffer = label[i]

        #Dimenzije bboxa se povecaju 2 puta (obrnuto proporcionalan)
        label_buffer[3] =  label_buffer[3]*zoom_factor
        label_buffer[4] =  label_buffer[4]*zoom_factor

        #Provjera kvadranta koordinata
        if(label[i,1] > 0.5 and label[i,2] <= 0.5): #I kvadrant
            label_buffer[1] = (label[i,1] - shift[0,0])*zoom_factor
            label_buffer[2] = (label[i,2] - shift[0,1])*zoom_factor
            label_I.append(label_buffer)
        elif(label[i,1] <= 0.5 and label[i,2] <= 0.5): #II kvadrant
            label_buffer[1] = (label[i,1] - shift[1,0])*zoom_factor
            label_buffer[2] = (label[i,2] - shift[1,1])*zoom_factor
            label_II.append(label_buffer)
        elif(label[i,1] <= 0.5 and label[i,2] > 0.5): #III kvadrant
            label_buffer[1] = (label[i,1] - shift[2,0])*zoom_factor
            label_buffer[2] = (label[i,2] - shift[2,1])*zoom_factor
            label_III.append(label_buffer)
        else: #IV kvadrant
            label_buffer[1] = (label[i,1] - shift[3,0])*zoom_factor
            label_buffer[2] = (label[i,2] - shift[3,1])*zoom_factor
            label_IV.append(label_buffer)

#pretvara listu nazad u np.array kako bi provjerio broj objekata
label_I = np.array(label_I)
label_II = np.array(label_II)
label_III = np.array(label_III)
label_IV = np.array(label_IV)

#provjera broja objekata
if(check4NO(label_I) == 0):
    label_I = None
if(check4NO(label_II) == 0):
    label_II = None
if(check4NO(label_III) == 0):
    label_III = None
if(check4NO(label_IV) == 0):
    label_IV = None
else:
```

```
        pass

    #slaze listu slika i labela
    new_labels = [label_I, label_II, label_III, label_IV]
    new_imgs = [tiles_r[1], tiles_r[0], tiles_r[2], tiles_r[3]]

    return new_imgs, new_labels

#Save label ovisno o tipu
def saveLabels(labels,new_name,path = os.getcwd()):
    new_name_ref = new_name

    for l in range(0, len(labels)):
        if(l == 0):
            new_name = new_name_ref + '-I'
        elif(l == 1):
            new_name = new_name_ref + '-II'
        elif(l == 2):
            new_name = new_name_ref + '-III'
        else:
            new_name = new_name_ref + '-IV'

        label = labels[l]
        if(label is None):
              open(path+'\\'+new_name+'.txt', 'w').close()
        elif(check4NO(label) == 1):
              np.savetxt(path+'\\'+new_name+'.txt', np.array(label).reshape(1,
np.array(label).shape[1]), fmt = '%f')
        else:
              np.savetxt(path+'\\'+new_name+'.txt',label,fmt = '%f')


#Save image - cisto reda radi
def saveImgs(imgs, new_name, path = os.getcwd()):
    new_name_ref = new_name
    for i in range(0,len(imgs)):
        if(i == 0):
            new_name = new_name_ref + '-I'
        elif(i == 1):
            new_name = new_name_ref + '-II'
        elif(i == 2):
            new_name = new_name_ref + '-III'
        else:
            new_name = new_name_ref + '-IV'

        cv2.imwrite(path+'\\'+new_name+'.png', imgs[i])

#----------------------------Main-------------------------------------------
#postavlja pathove direktorija
parent_dir = os.getcwd()
img_dir = os.path.join(parent_dir, 'images')
lb_dir = os.path.join(parent_dir, 'labels')
tiledImg_dir = os.path.join(parent_dir, 'tiled_images')
tiledLb_dir = os.path.join(parent_dir, 'tiled_labels')

def main():
    images = [os.path.join(img_dir, x) for x in os.listdir(img_dir)]
    labels = [os.path.join(lb_dir, x) for x in os.listdir(lb_dir)]

    for i in range(0,len(images)):
```

```
        img = cv2.imread(images[i])

        #ignorira "empty text file" upozorenje od np.loadtxt()
        with warnings.catch_warnings():
            warnings.simplefilter("ignore")
            label = np.loadtxt(labels[i])

        #ime slike
        img_str = images[i].rpartition('\\')
        img_str = img_str[-1].rpartition('.')

        new_imgs, new_labels = splitImage(img, label)
        saveImgs(new_imgs, img_str[0], tiledImg_dir)
        saveLabels(new_labels, img_str[0], tiledLb_dir)

        print(img_str[0] + ' done!')

    print('-----All done!-----')

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