Primjena računalnog vida za detekciju korova na plantaži jagoda

Makarun, Josip

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Josip Makarun

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentori:

Izv. prof. dr. sc. Tomislav Stipančić, dipl. ing.

Student:

Josip Makarun

Zagreb, 2024.

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

Zahvaljujem se mentoru izv. prof.dr.sc. Tomislavu Stipančiću na pomoći i savjetima tokom izrade ovog rada.

Josip Makarun

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
Proizvodno inženjerstvo, inženjerstvo materijala, industrijsko inženjerstvo i menadžment, mehatronika i robotika, autonomni sustavi i računalna inteligencija

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 - 04 / 24 - 06 / 1**	
Ur.broj: **15 - 24 -**	

DIPLOMSKI ZADATAK

Student: **Josip Makarun** JMBAG: 0035202660

Naslov rada na hrvatskom jeziku: **Primjena računalnog vida za detekciju korova na plantaži jagoda**

Naslov rada na engleskom jeziku: **Application of computer vision for weed detection on a strawberry plantation**

Opis zadatka:

Računalni vid se koristi u različite svrhe, uključujući rješavanje zadataka koji uključuju detekciju, klasifikaciju, prepoznavanje i slično. Pritom se često koriste različite metode umjetne inteligencije koje prilikom treninga modela koriste prikladne podatke.

Cilj ovog rada je razviti sustav za prepoznavanje i detekciju korova na plantaži jagoda koristeći model duboke neuronske mreže (npr. YOLOv8 model).

U radu je potrebno:

- upoznati se s osnovnim principima dubokog učenja te njegove primjene u kontekstu računalnog vida
- upoznati se s arhitekturom konvolucijskih neuronskih mreža
- pripremiti skup podataka koji sadrži slike plantaže jagoda s različitim vrstama korova za obuku i testiranje modela
- trenirati model koristeći pripremljeni skup podataka
- evaluirati razvijeni model u sklopu Laboratorija za projektiranje izradbenih i montažnih sustava te dati kritički osvrt.

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

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
9. svibnja 2024.	11. srpnja 2024.	15. – 19. srpnja 2024.

Zadatak zadao: Predsjednik Povjerenstva:

Izv. prof. dr. sc. Tomislav Stipančić Prof. dr. sc. Ivica Garašić

SADRŽAJ

POPIS SLIKA

SAŽETAK

Razvojem umjetne inteligencije pojavile su se nove tehnologije koje koriste istu. Jedna od njih je i računalni vid koji ima istu ulogu za računala kao što oči imaju kod čovjeka. Uz pomoć računalnog vida mnogi problemi koje je čovjek teško rješavao postaju rješivi od strane računala u veoma kratkom vremenu i to s točnošću kakvom niti jedno ljudsko oko ne može riješiti. U ovome radu uz pomoć YOLOv8 algoritma, trenirat će se model koji će moći detektirati 4 vrste korova na plantaži jagoda. U radu će biti opisan postupak izrade samog modela te bit će dana rješenja za moguća poboljšanja modela u budućnosti.

Ključne riječi: umjetna inteligencija, duboko učenje, konvolucijske neuronske mreže, računalni vid, prepoznavanje objekata, YOLOv8

SUMMARY

With the development of artificial intelligence, new technologies have appeared that use it. One of them is computer vision, which has the same role for computers as the eyes have for humans. With the help of computer vision, many problems that were difficult for humans to solve became solvable by computers in a very short time and with accuracy that no human eye can solve. In this work, using the YOLOv8 algorithm, a model will be trained to detect 4 types of weed on a strawberry plantation. The process of creating a model will be explained and solutions for possible improvements in the future will be provided.

Key words: artificial intelligence, deep learning, convolutional neural networks, computer vision, object recognition, YOLOv8

1. UVOD

Rapidnim razvojem grafičkih kartica u posljednjih par desetljeća omogućen je nagli razvoj umjetne inteligencije. Umjetna inteligencija je u ovom trenutku toliko razvijena da neki današnji modeli imaju sličnu brojku neurona kao što to ima ljudski mozak. U današnjem društvu umjetna inteligencija se primjenjuje u gotovo svim područjima ljudskih aktivnosti od industrije i gospodarstva, financija pa do medicine gdje se uz pomoć umjetne inteligencije prikupljaju razne vrste podataka od slika, signala, nalaza i sličnog. Umjetna inteligencija ima veliku primjenu i u edukaciji gdje su se pojavili mnogi veliki jezični modeli koji su u stanju odgovarat na različita pitanja. Međutim trenutno jedna od najaktualnijih područja umjetne inteligencije su autonomna vozila, koja moraju sadržavati računalo koje uz pomoć kamera mora prepoznati sve što se nalazi u okolini vozila baš kao što bi to učinio i čovjek. Upravo u toj primjenu veliku ulogu ima računalni vid koji je zadužen za detekciju i praćenje objekata.

Glavni zadatak ovog rada bit će upravo korištenjem algoritma YOLOv8 računalnog vida, trenirati model koji će služiti za detekciju korova u plantaži jagoda. Za potrebe stvaranja odnosno treniranja modela bit će potrebno skupiti dovoljnu količinu kvalitetnih slika koje će služiti za treniranje modela. Također bitno je obaviti dovoljno dugačko treniranje kako bi model pokazao dovoljno dobru preciznost i pouzdanost. U nastavku rada prvo ću se bazirati na teorijskoj osnovi umjetne inteligencije kao i njezinim pod skupovima koji su bitni za praktični dio ovog rada, dok ću u drugom dijelu opisati postupak kojim sam stigao do konačnog modela te ću navesti dobre i loše strane, te moguća buduća poboljšanja modela.

2. UMJETNA INTELIGENCIJA

Umjetna inteligencija UI (eng. *AI – artificial inteligence*) je simulacija ljudske inteligencije koju koriste strojevi, posebice računalni sustavi. Sustavi umjetne inteligencije za obuku koriste velike količine podataka kako bi se pronašli uzorci i korelacije te uz pomoć tih uzoraka pokušavaju predvidjeti buduća stanja. Sistemi UI se baziraju na sljedećim vještinama:

- **Učenje**: uključuje prikupljanje podataka te postavljanje pravila, takozvanih algoritama koji daju računalu upute za izvršavanje određenih zadataka.
- **Obrazlaganje**: uključuje izbor ispravnog algoritma za postizanje željenih rezultata.
- **Samo ispravljanje**: uključuje algoritme koji se konstantno uče i prilagođavaju kako bi dobili što je moguće točnije rezultate.
- **Kreativnost**: uključuje korištenje neuronskih mreža, statističkih metoda te drugih tehnika UI. [1]

Neke od prednosti umjetne inteligencije su učinkovitost u poslovima s velikim bazama podataka, ušteda vremena, povećanje produktivnosti kao i dosljednost u. Dok kao nedostatke možemo navesti visoke troškove, složenost razvoja i rada sustava UI, na određenim radnim mjestima moguća zamjena ljudi sa strojevima koji koriste UI te moguće zlouporabe podataka. [2]

Slika 1. Veza između umjetne inteligencije, strojnog učenja i dubokog učenja

2.1. Strojno učenje

Strojno učenje je kao što je moguće vidjeti na slici 1. podskup umjetne inteligencije u kome se računala uče iz skupova podataka te donose odluke bez da su programirani za to. Algoritmi strojnog učenja mogu se svrstati u sljedeće tri kategorije:

- **Nadzirano strojno učenje**: algoritmi rade s poznatim skupom ulaznih i izlaznih podataka, omogućujući im da točno prepoznaju obrasce i predviđaju ishode.
- **Nenadzirano strojno učenje**: treniranje modela da sortira neoznačene skupove podataka kako bi pronašli temeljne odnose.
- **Polu-nadzirano strojno učenje:** nešto između prva dva tipa učenja. Tijekom obuke koristi se manji označeni skup podataka za klasifikaciju i izdvajanje značajki iz većeg ne označenog skupa podataka. [3]

2.2. Duboko učenje

Duboko učenje je podskup strojnog učenja koji koristi višeslojne neuronske mreže za simuliranje složene moći ljudskog mozga u donošenju odluka. Duboke neuronske mreže sastoje se od više slojeva međusobno povezanih čvorova, od kojih je izlaz jednog ulaz drugo čvora. Duboko učenje radi na većim skupovima podataka u odnosu na strojno učenje, a mehanizmi predviđanja su upravljani od strane samih strojeva. [4]

Modeli dubokog učenja mogu automatski učiti iz podataka, zbog čega su veoma prikladni za zadatke kao što su prepoznavanje slika i govora. Najčešće korištene arhitekture u dubokom učenju su:

- **Neuronska mreža širenja unaprijed (eng. *FNN feedfoward neural networks*)**: najjednostavniji tip umjetnih neuronskih mreža s linearnim protokom informacija kroz mrežu.
- **Konvolucijske neuronske mreže (eng. *CNN convolutional neural networks*)**: zbog svoje prednosti da mogu automatski naučiti značajke iz slika, primjenjuju se za zadatke kao što su klasifikacija i segmentacija slika te detekcija objekata na slikama i videima. Upravo zbog toga ću ovu skupinu umjetnih neuronskih mreža detaljnije upisati u nastavku rad.

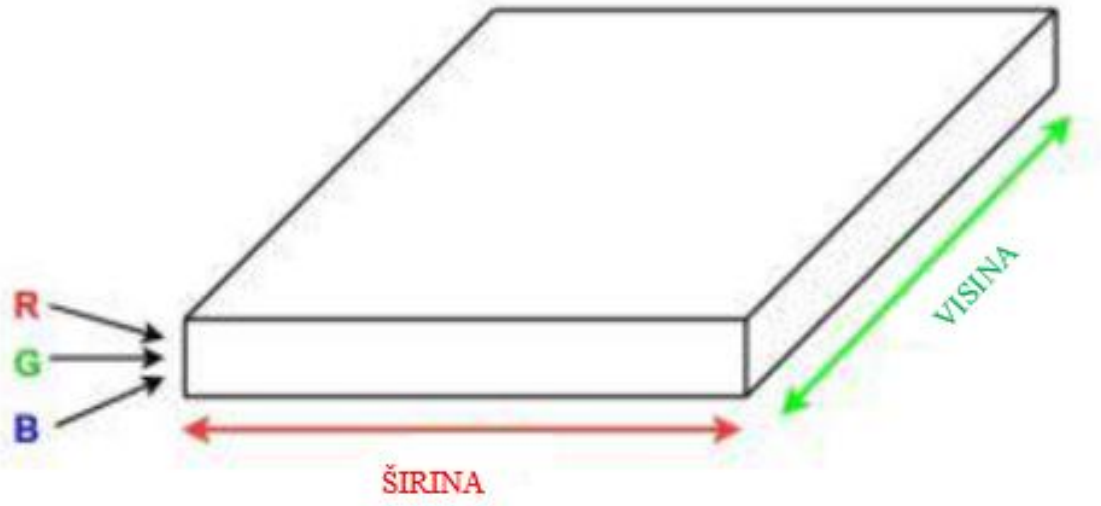
- **Ponavljajuće neuronske mreže (eng. *RNN recurrent neural networks*)**: skupina umjetnih neuronskih mreža koja se koristi za prepoznavanje govora i prijevod jezika.[5]

2.2.1. Konvolucijske neuronske mreže

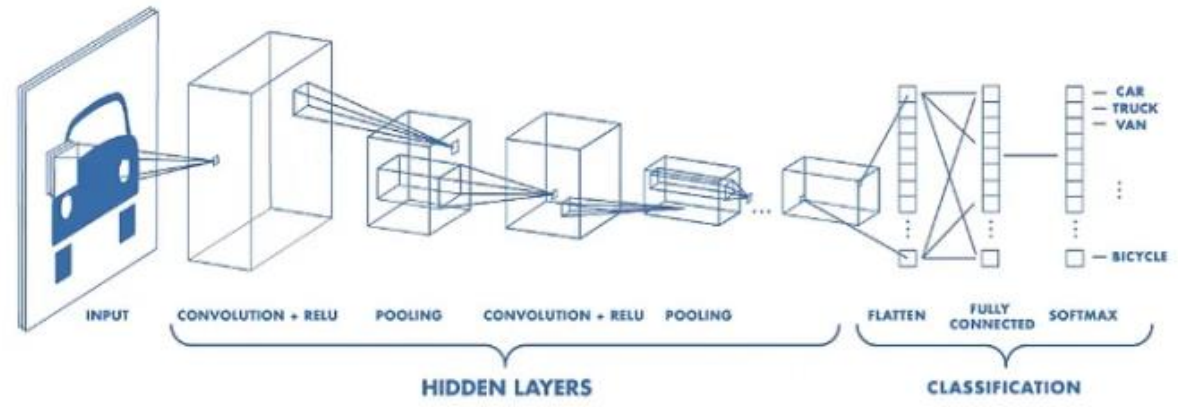
Konvolucijska neuronska mreža je vrsta arhitekture neuronske mreže dubokog učenja koja se najčešće koristi u računalnom vidu. Nazivane su po matematičkoj operaciji zvanoj konvolucija zbog toga što primjenjuju operaciju konvolucije na minimalno jednom sloju. Konvolucija je matematička funkcija koja je nastala integriranjem umnoška dviju funkcija po intervalu njihove definicije gdje su te funkcije ravnopravne. [6]

2.2.1.1. Arhitektura

Kako se konvolucijske neuronske mreže najčešće koriste za obradu slikovnih podataka, na ulazu se koristi slika koja je prikazana u tri dimenzije: širina, visina i dubina. Dubina označuje boje na slici te se najčešće označava s tri vrijednosti za RGB sustav ili sa samo jednom vrijednošću ako je slika crno bijela. Širina i visina označavaju broj piksela po širini odnosno visini.



Slika 2. Dimenzije ulaznih slikovnih podataka



Slika 3. Arhitektura konvolucijske neuronske mreže [7]

Osnovne komponente konvolucijske neuronske mreže:

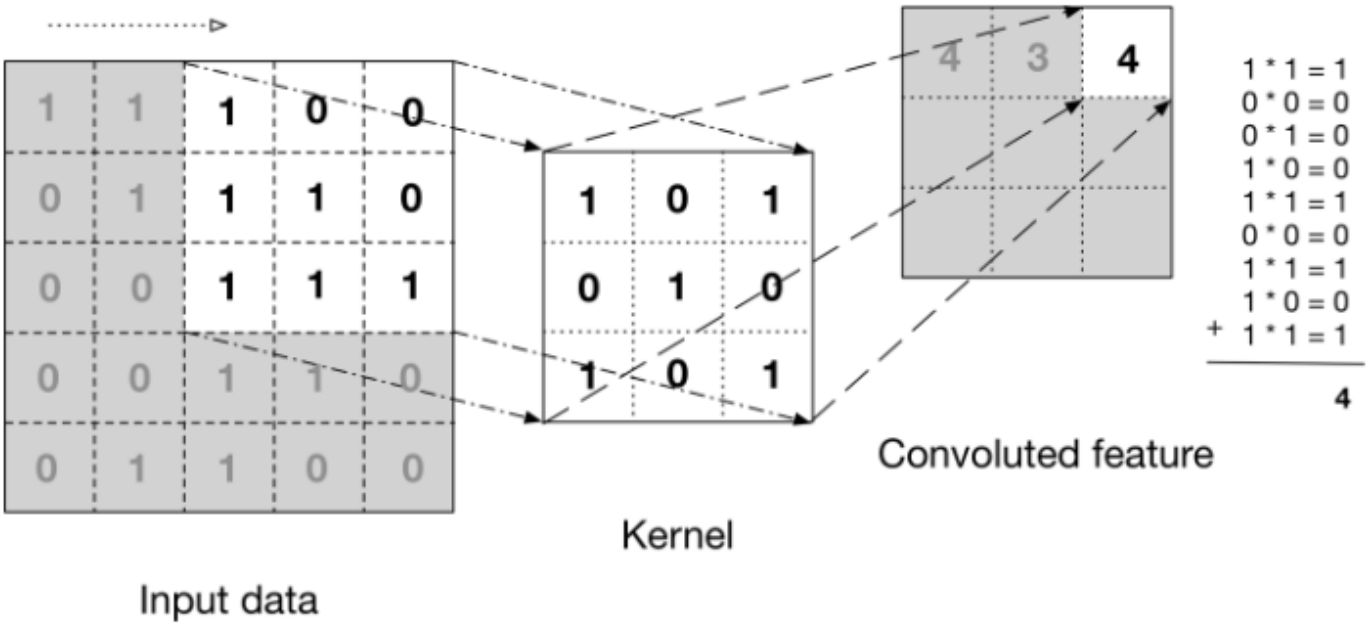
- Ulazni sloj;
- Konvolucijski sloj;
- Aktivacijska funkcija;
- Sloj sažimanja;
- Potpuno povezani sloj;
- Izlazni sloj.

2.2.1.2. Ulazni sloj

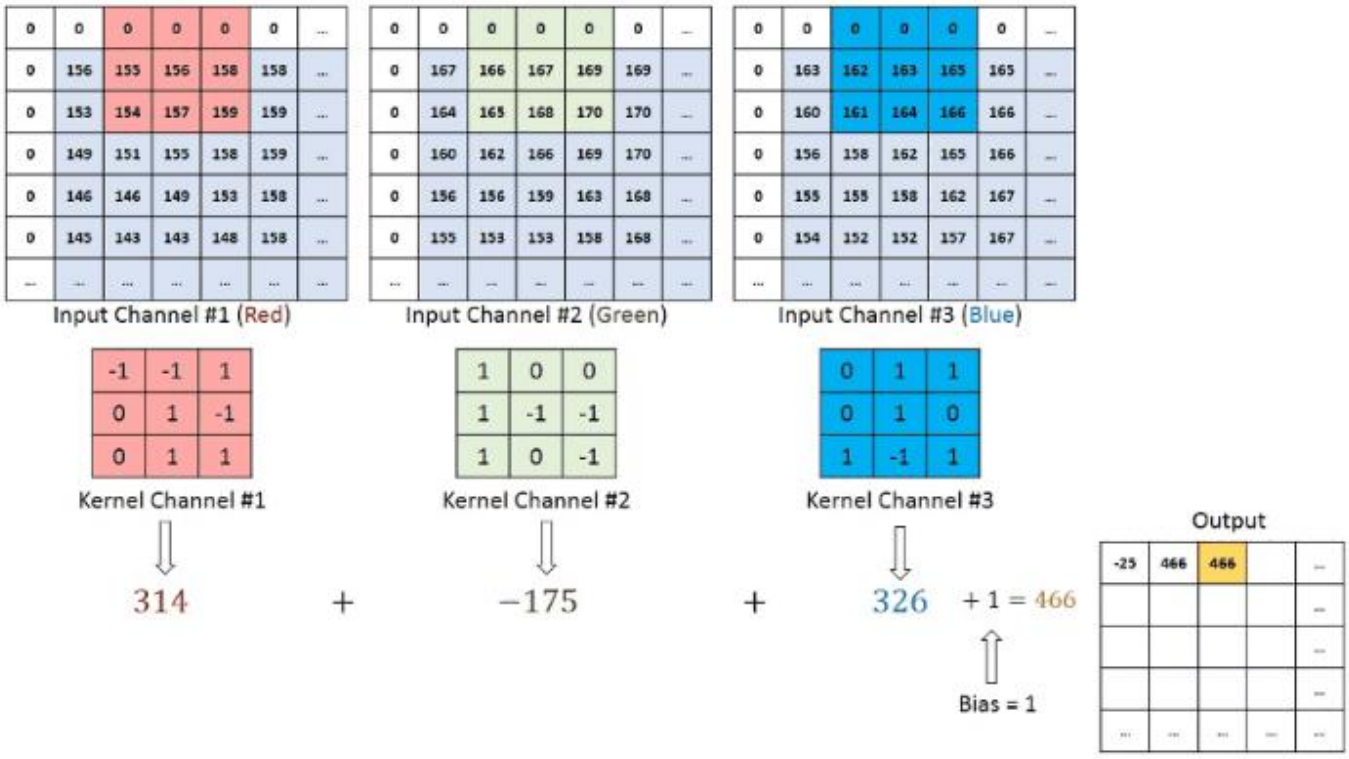
U ovaj sloj dajemo ulaz našem modelu što je uobičajeno slika ili niz slika. On sadrži neobrađeni ulaz slike s visinom i širinom 32 piksela i dubinom 3 piksela.

2.2.1.3. Konvolucijski sloj

Konvolucijski sloj je osnovni građevni blok konvolucijskih neuronskih mreža. U ovom sloju se primjenjuju specijalizirani filteri takozvani kerneli koji sadrže težine. Ulazni podaci i filteri su prikazani preko matrica. Ulazni podatak ima matricu veličine broja piksela po visini puta broj piksela po širini, dok se veličina matrice filtera odabire proizvoljno ali je poželjno uzeti veličinu matrice koja je neprana kako bi piksel na kome vršimo operaciju bio u centru. Za konvoluciju se ne koristi cijela matrica ulaznih podataka već se uzima dio po dio.

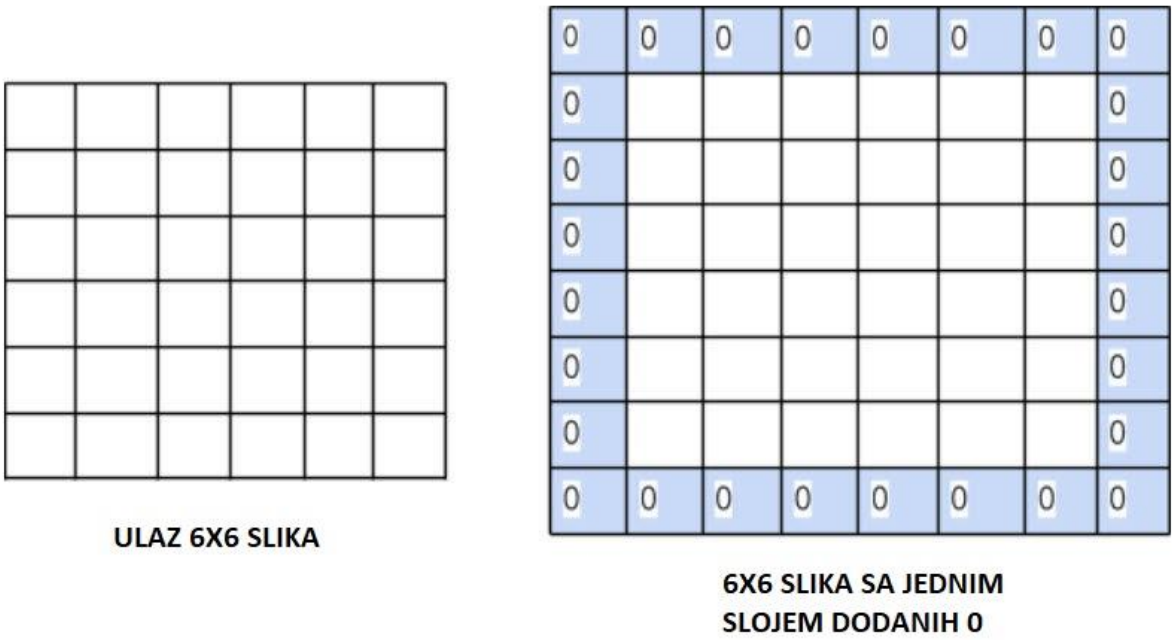


Slika 4. Prikaz konvolucije matrice ulaza i filtera za crno bijelu sliku na ulazu. [8]



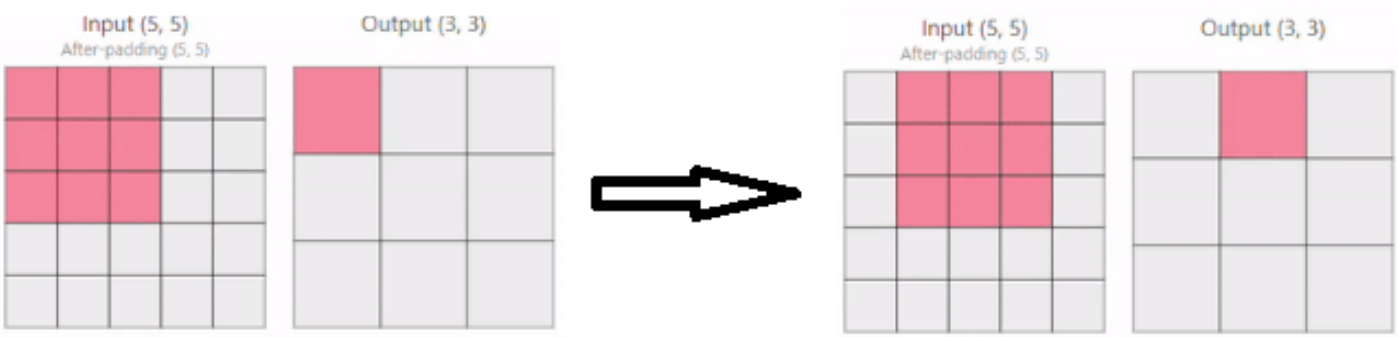
Slika 5. Prikaz konvolucije matrice ulaza i filtera za sliku u boji na ulazu. [8]

Nakon što se izvrši konvolucija na zadanom pikselu, pomičemo se na susjedni piksel te ponovo vršimo konvoluciju i tako dok se konvolucija ne izvrši nad svim pikselima u ulaznoj matrici. Da bi mogli provesti konvoluciju nad rubnim pikselima koristima eng. *zero padding*, odnosno dodajemo nule oko rubnih piksela kako bi se filter mogao pozicionirati uz rubne piksele a da ne dolazi do utjecaja na rezultat.

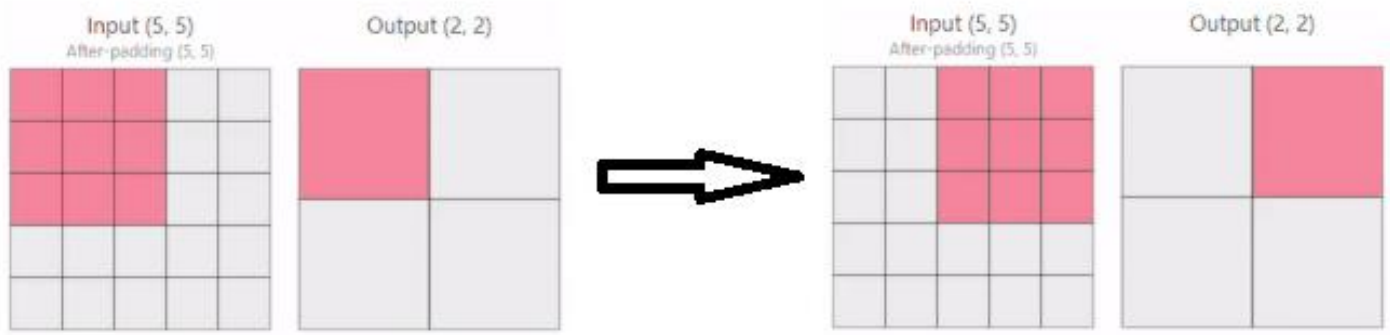


Slika 6 eng. *zero padding* [9]

Filter se kroz piksele može pomicati na različite načine, odnosno različitim korakom. Na slici 6 je prikazana konvolucija s korakom 1, dok je na slici 7 prikazana konvolucija s korakom 2. Kao što možemo vidjeti na slikama 6 i 7, korak nam određuje veličinu matrice izlaza te iz toga možemo zaključiti da korištenjem većeg koraka omogućujemo filteru da prođe veću površinu. Ako nas zanimaju više globalne značajke koristit ćemo veći korak dok u suprotnom za pronalaženje više lokalnih značajka koristimo manje korake.



Slika 7. Prikaz konvolucije s korakom 1 [10]

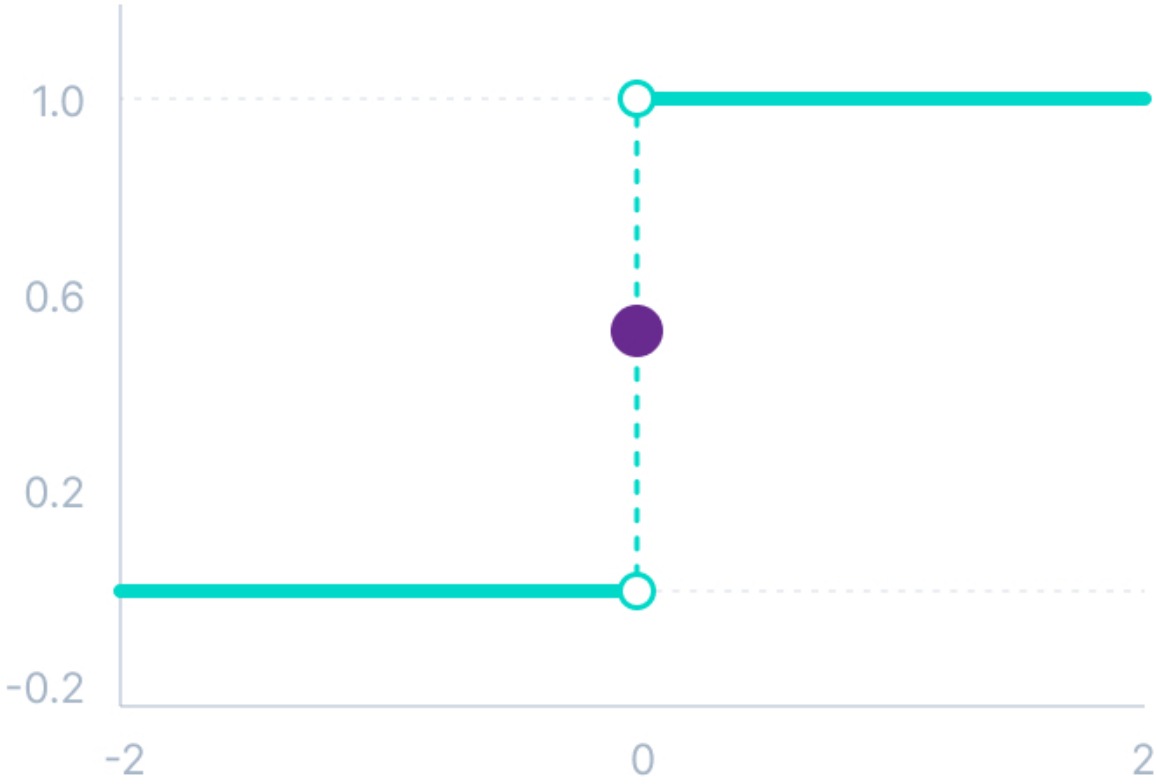


Slika 8. Prikaz konvolucije s korakom 2 [10]

2.2.1.4. Aktivacijska funkcija

Aktivacijska funkcija odlučuje je li ulaz u neuron važan ili ne, te u odnosu na taj podatak odlučuje hoće li se neuron aktivirati ili neće. Još jedna bitna svrha aktivacijske funkcije je ta da neuronskoj mreži koja je sama po sebi linearna dodamo nelinearnost. Tri su tipa aktivacijskih funkcija neuronskih mreža a to su:.

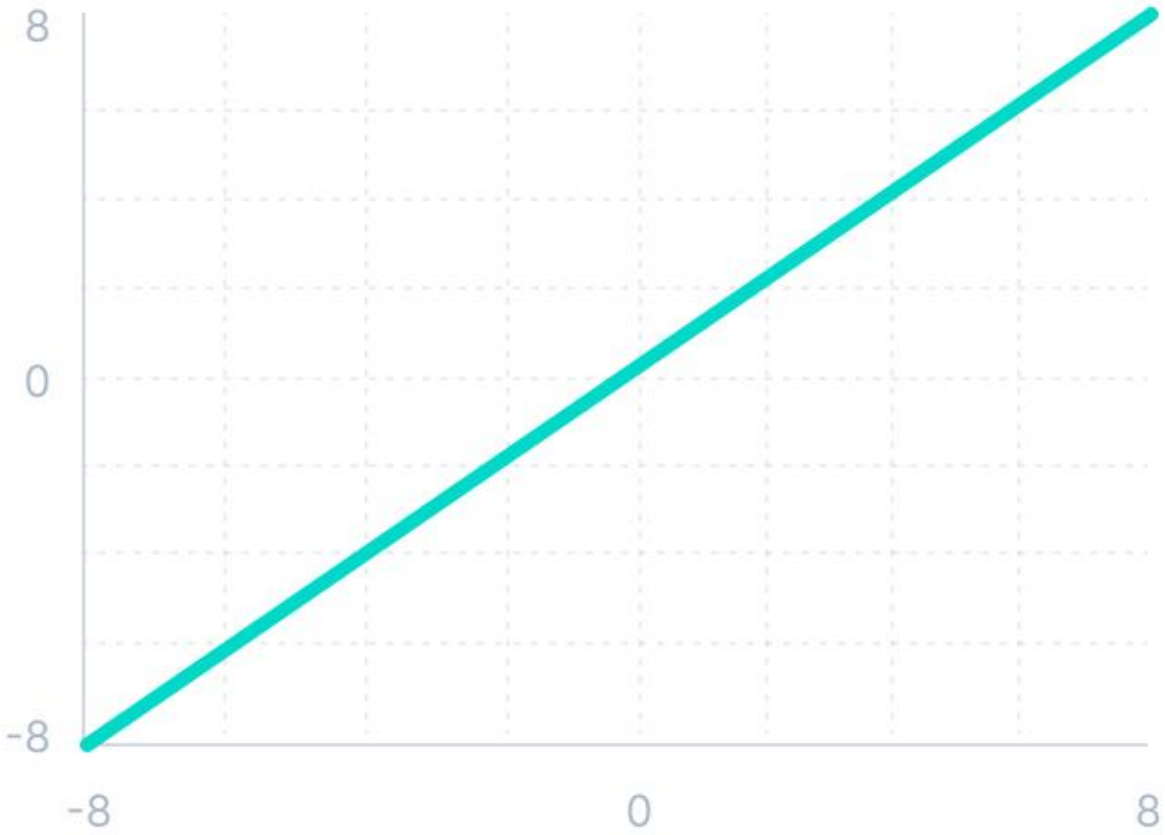
- **Binarna koračna funkcija**: ulaz doveden u funkciju se uspoređuje s određenim pragom, ako je ulaz veći od praga tada se neuron aktivira, dok u obrnutom slučaju neuron ostaje deaktiviran. Binarna koračna funkcija je prikazana slikom 9 i jednadžbom (1).



Slika 9. Binarna koračna funkcija [11]

$$f(x) = \begin{cases} 0 \; za \; x < 0 \\ 1 \; za \; x \geq 0 \end{cases} \quad \textbf{(1)}$$

- **Linearna aktivacijska funkcija**: funkcija u kojoj nemamo binarnu aktivaciju. Funkcija ne radi ništa s težinskom sumom na izlazu, već na izlazu samo izbaci vrijednost koju je primila. Veliki problem kod linearne aktivacijske funkcije je taj što će bez obzira na broj slojeva u neuronskoj mreži, posljednji sloj će i dalje biti linearna funkcija prvog sloja.

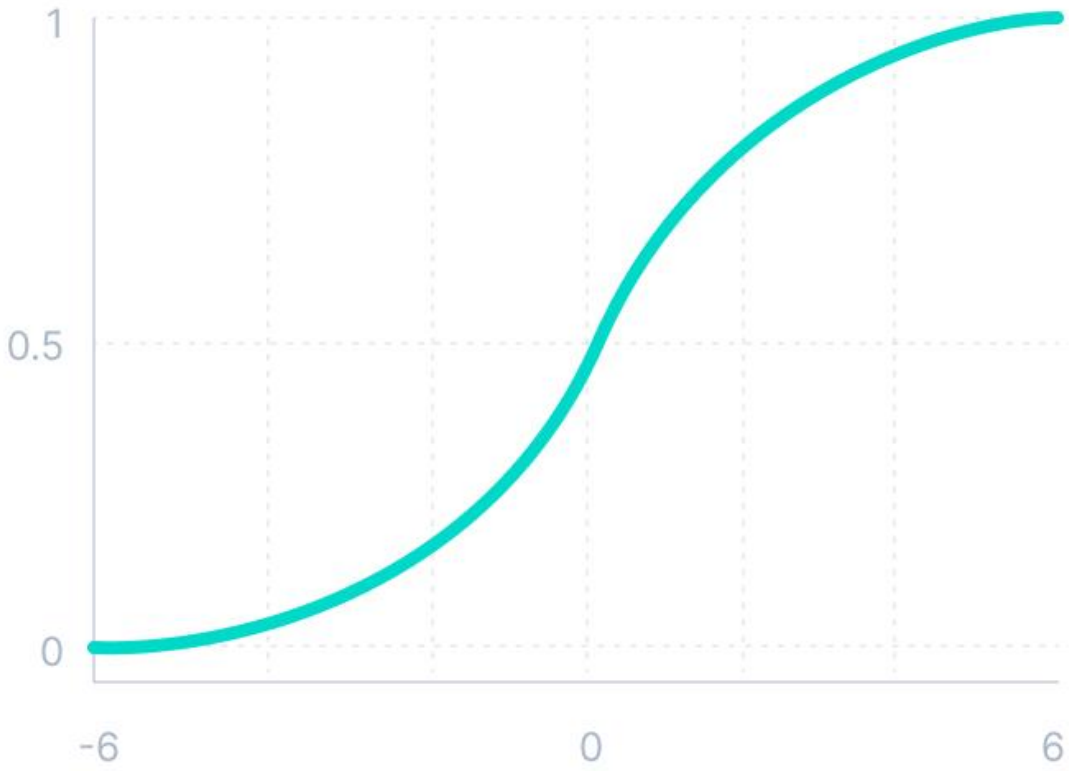


Slika 10. Linearna aktivacijska funkcija [11]

$$f(x) = x \tag{2}$$

- **Nelinearne aktivacijske funkcije** : najčešće su korištene aktivacijske funkcije jer rješavaju ograničenja i nedostatke linearnih aktivacijskih funkcija. Iz tog razloga ću u nastavku rada detaljnije opisati nekoliko najčešće korištenih nelinearnih aktivacijskih funkcija. Neke od bitnijih nelinearnih aktivacijskih funkcija su:

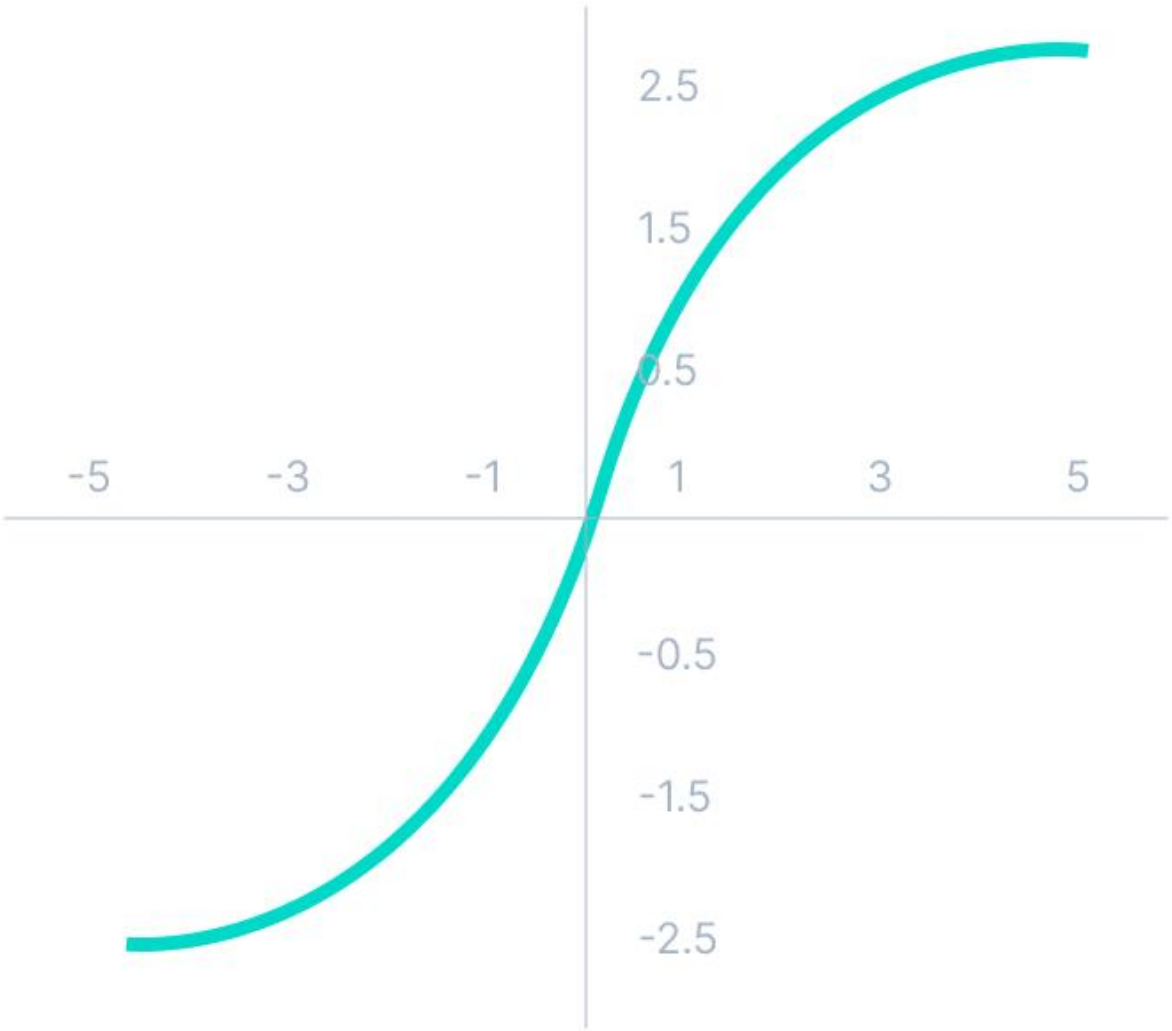
    - **Sigmoid ili logička aktivacijska funkcija**: kao ulaz uzima bilo koju stvarnu vrijednost i daje izlaznu vrijednost u rasponu od 0 do 1. Što je ulaz veći to će izlaz biti bliže 1, isto tako što je ulaz manji to će izlaz biti bliži 0. Funkcija je diferencijabilna što znači da možemo u bilo koje dvije točke pronaći nagib sigmoidne krivulje. Sigmoidna ili logička aktivacijska funkcija je prikazana na slici 11, te je opisana jednadžbom (3).



Slika 11. Sigmoid ili logička aktivacijska funkcija [11]

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

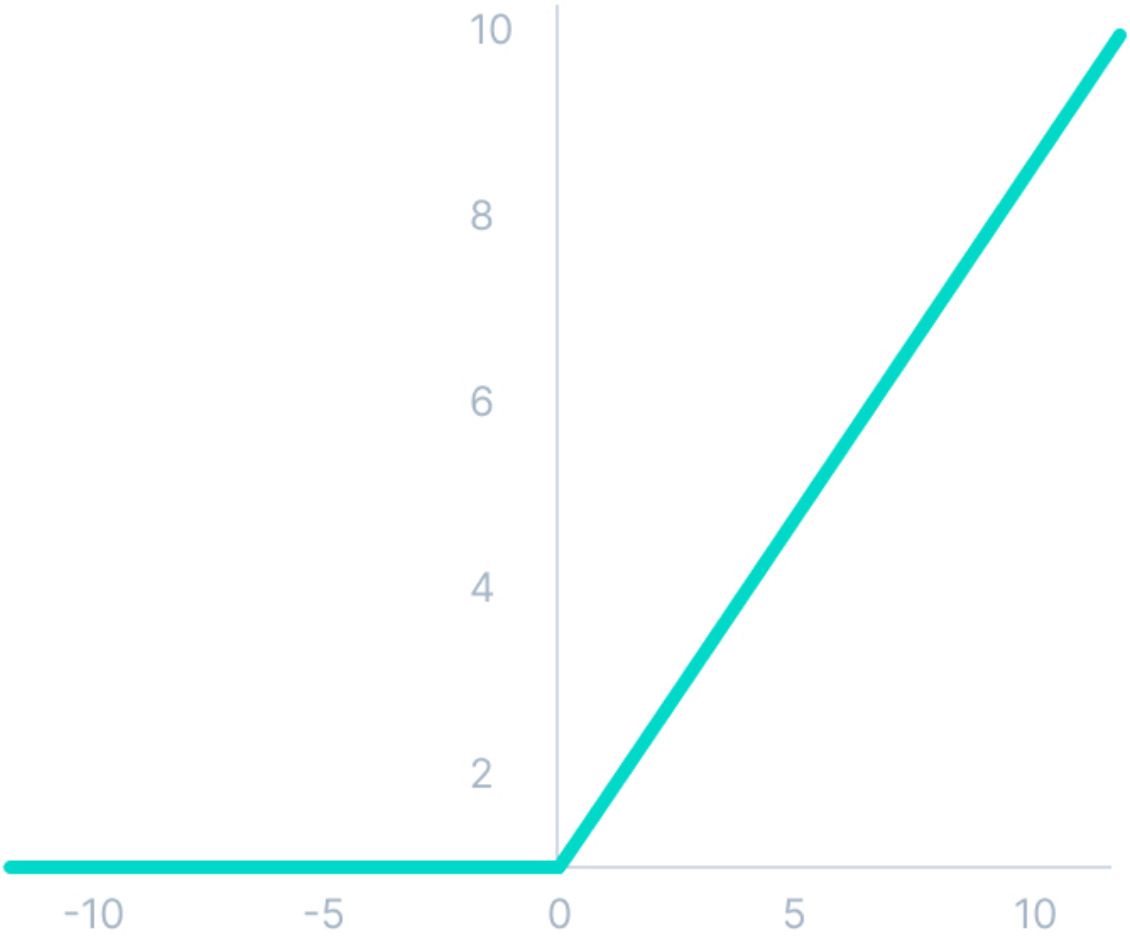
- **Tanh ili hiperbolička tangentna aktivacijska funkcija**: vrlo je slična sigmoidnoj, čak imaju i sličan S-oblik. Jedina razlika je što Tanh funkcija ima raspon od 1 do -1. Tanh aktivacijska funkcija je prikazana na slici 12, te je opisana jednadžbom (4).



Slika 12. Tanh ili hiperbolička tangentna aktivacijska funkcija [11]

$$f(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})} \tag{4}$$

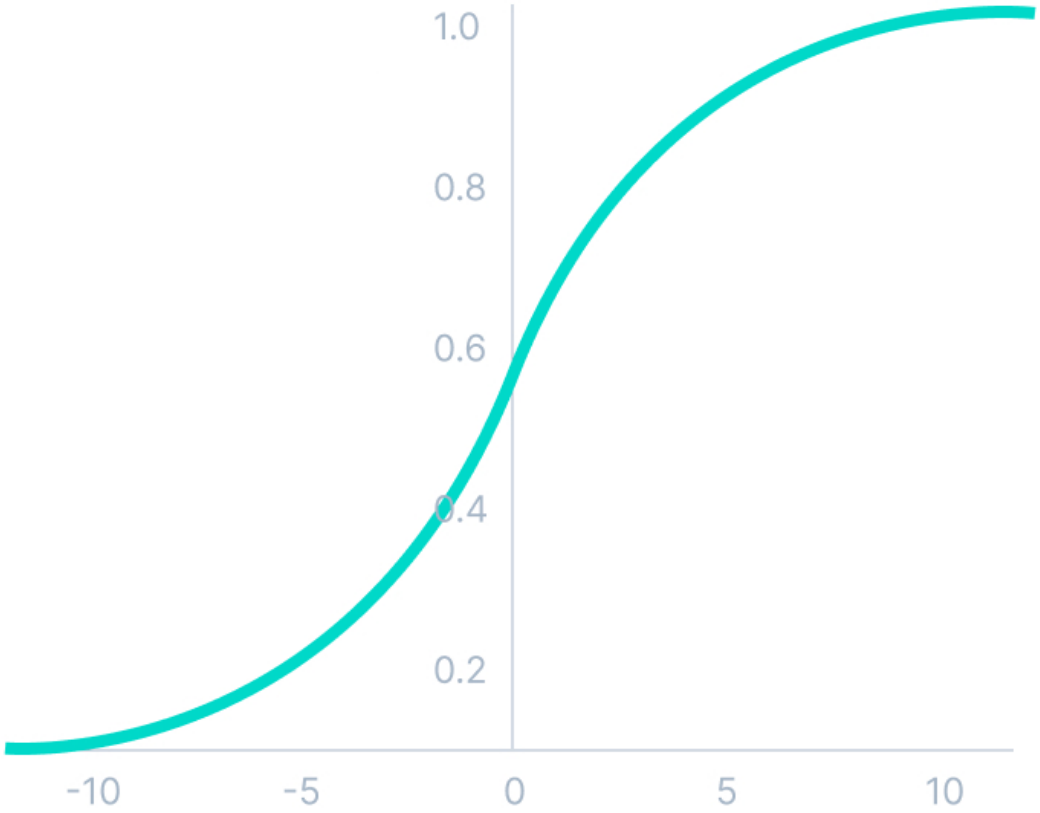
- **ReLU aktivacijska funkcija**: ako je ulaz pozitivan tada na izlazu prosljeđuje ulaz, a ako je ulaz negativan tada na izlazu imamo 0. Najveća prednost ReLU funkcije je ta što nisu svi neuroni aktivni u isto vrijeme. Zbog toga ReLu funkcija je brža od Sigmoid i Tanh funkcija te se zbog tog toga najčešće koristi. ReLU aktivacijska funkcija je prikazana na slici 13, te je opisana jednadžbom (5).



Slika 13. ReLU aktivacijska funkcija

$$f(x) = max\ (0, x) \quad \textbf{(5)}$$

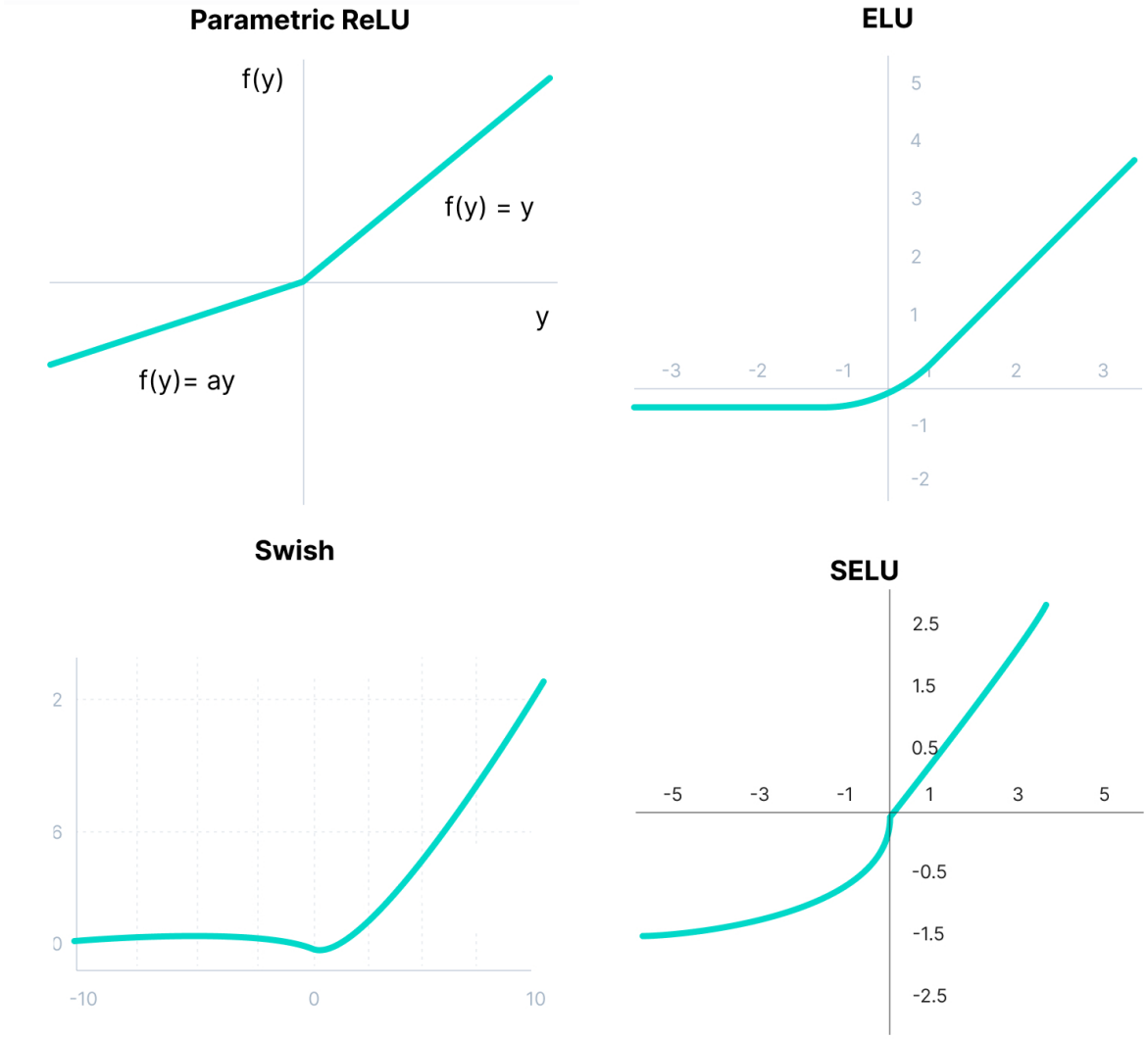
- **Softmax aktivacijska funkcija**: je vrsta sigmoidne funkcije s izlazom koji poprima iznose od 0 do 1. Najčešće se koristi kao aktivacijska funkcija za posljednji sloj neuronske mreže za slučajeve s više klasa. Softmax aktivacijska funkcija je prikazana na slici 14, te je opisana jednadžbom (6).



Slika 14. Softmax aktivacijska funkcija [11]

$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)} \quad \textbf{(6)}$$

- Postoje i druge aktivacijske funkcije koje su prikazane na slici 15, kao što su ELU, Swish, SELU i parametarski ReLU. Zbog toga što se ne primjenjuju u slučajima koje ću razraditi u ovome radu, neću ih detaljnije opisivati.



Slika 15. Neke od ostalih nelinearnih aktivacijskih funkcija [11]

- **Upute za odabir korištenja aktivacijskih funkcija**: započnemo s korištenjem ReLU aktivacijske funkcije, ako ReLU ne daje optimalne rezultate možemo prijeći na korištenje druge aktivacijske funkcije. ReLU aktivacijske funkcije se koriste samo u skrivenim slojevima dok sigmoidne i tanh funkcije nije poželjno koristiti u skrivenim slojevima jer čine model osjetljivim na probleme tijekom obuke. Ako imamo neuronske mreže koje posjeduju dubinu veću od 40 slojeva tada ćemo koristiti Swish aktivacijsku funkciju. Ovisno o vrsti problema predviđanja, odabiremo sljedeću aktivacijsku funkciju:
  - **Regresija** - Linearna aktivacijska funkcija;
  - **Binarna klasifikacija** – Sigmoidna aktivacijska funkcija;
  - **Višerazredna klasifikacija** – Softmax;

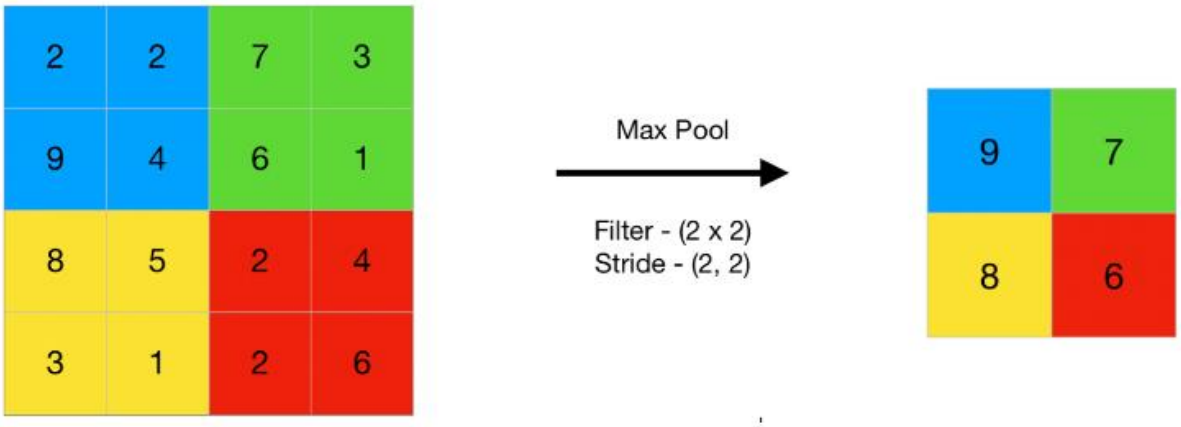
- **Klasifikacija s više oznaka** – Sigmoid.

- **Odabir aktivacijske funkcije na temelju vrste arhitekture neuronske mreže**:
  - Konvolucijske neuronske mreže (eng. *CNN convolutional neural networks*): – ReLu aktivacijska funkcija;
  - Ponavljajuće neuronske mreže (eng. *RNN recurrent neural networks*) – tanh i/ili sigmoidna aktivacijska funkcija. [11]

2.2.1.5. Sloj sažimanja

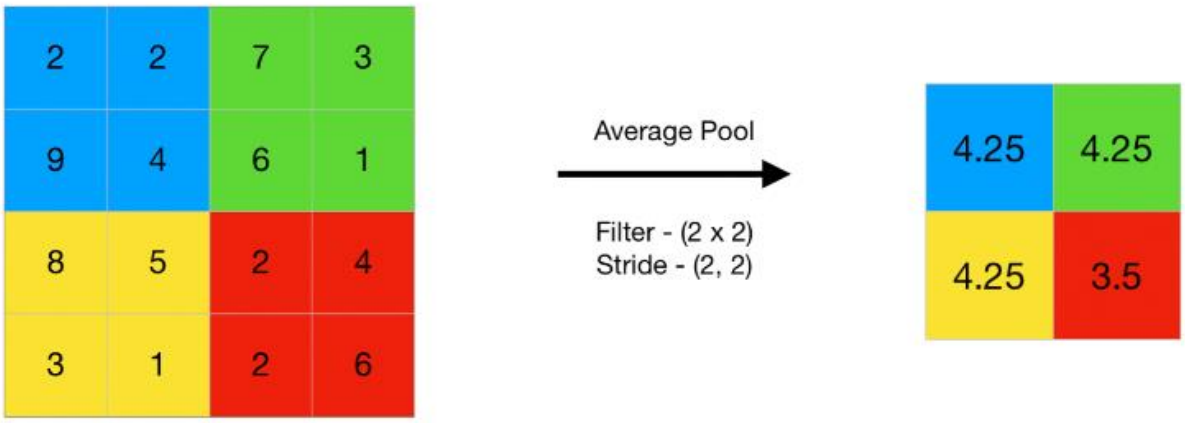
Sažimanje (eng. *pooling*) predstavlja operaciju prelaza dvodimenzionalnog filtera preko svakog kanala mape značajki te ujedno sažimanje prostornih dimenzija podataka. Pri sažimanju smanjuju se dimenzije širine i visine dok dubina ostaje nepromijenjena. Sažimanje se radi kako bi se smanjila količina podataka a samim time bila bi potrebna manja snaga računala za obradu tih podataka. Česta pojava kod arhitektura konvolucijskih neuronskih mreža je da imamo više konvolucijskih slojeva te nakon svakog od njih dolazi jedan sloj sažimanja. Parametri kod sloja sažimanja su veličina filtera i korak kojim se on pomiče. Postoje dvije vrste slojeva sažimanja:

- **Maksimalno sažimanje**: operacija koja odabire najveći element iz regije značajke na kojoj je primijenjen filter;



Slika 16. Primjer maksimalnog sažimanja [12]

- **Srednje sažimanje**: računa se prosjek elemenata prisutnih u području značajke koji su pokriveni filterom.

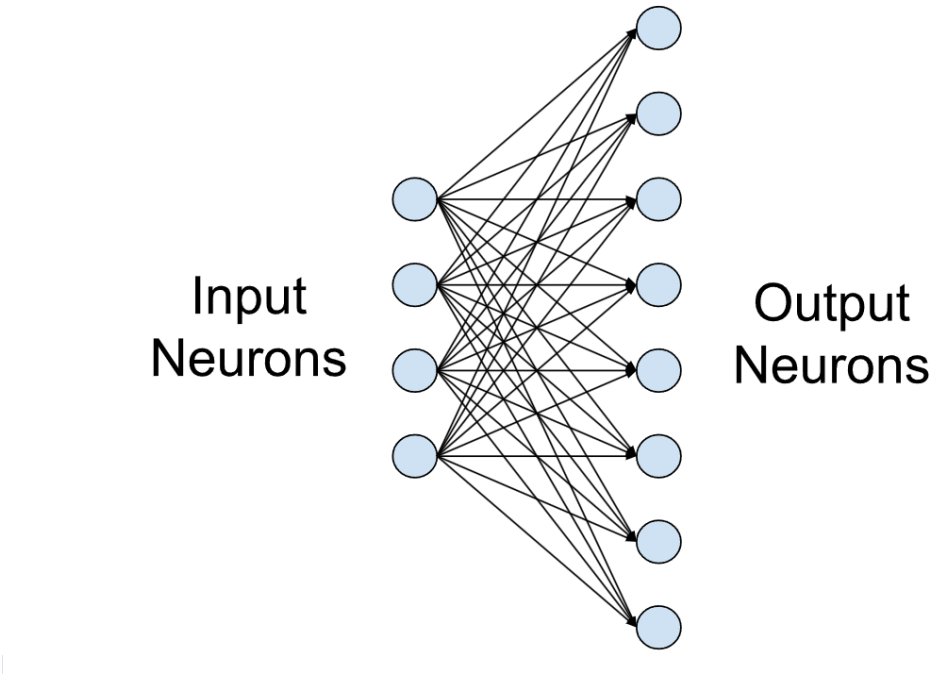


Slika 17. Primjer srednjeg sažimanja [12]

Srednje sažimanje izvodi smanjenje dimenzija kao mehanizam za suzbijanje šuma, zbog toga možemo reći da maksimalno sažimanje ima puno bolje rezultate te se zbog toga koristi u konvolucijskim neuronskim mrežama.

2.2.1.6. Potpuno povezani slojevi

Potpuno povezani sloj je vrsta sloja koji se koristi u konvolucijskim neuronskim mrežama gdje je svaki neuron iz prethodnog sloja povezan sa svakim neuronom trenutnog sloja. Potpuno povezani slojevi se nalaze na začelju arhitekture konvolucijskih neuronskih mreža te su odgovorni za konačna izlazna predviđanja. [13] Dok se u konvolucijskom sloju obično koristi ReLU aktivacijska funkcija, u potpuno povezanom sloju se koristi softmax aktivacijska funkcija za odgovarajuću klasifikaciju ulaza. [14]



Slika 18. Prikaz potpuno povezanog sloja [15]

2.2.1.7. Izlazni sloj

Izlaz dobiven iz potpuno povezanih slojeva se ubacuje u logičku funkciju za zadavanje klasifikacija kao na primjer sigmoid ili softmax koje pretvaraju izlaz svake klase u ocjenu vjerojatnosti. [16]

3. RAČUNALNI VID

Računalni vid je polje umjetne inteligencije koje koristi strojno učenje i neuronske mreže kako bi naučilo računalne sustave da izvuku značajne informacije iz slika, videa ili nekih drugih vizualnih ulaza, te da daju preporuke ili poduzmu radnje kada vide nedostatke i probleme. [18] Ako smatramo da je umjetna inteligencija alat koji omogućuje računalima da razmišljaju, tada možemo računalni vid smatrati alatom koji omogućuje računalima da vide i promatraju. Sam princip rada računalnog vida je dosta sličan onom ljudskom kojeg ljudi treniraju od svog rođenja. Računalni vid uz odgovarajući trening može veoma brzo nadmašiti ljudski, te upravo zbog toga je dobio primjenu na mnogim mjestima gdje može uočiti i analizirati ljudskom oku neprimjetne nedostatke i probleme.

3.1. Povijest

Dugo u povijest sežu pokušaji raznih znanstvenika koji su pokušavali razviti načine kako bi strojevi mogli vidjeti i razumjeti vizualne podatke. Prvo eksperimentiranje je započelo 1959. godine kada su neurofiziolozi pokazivali mački niz slika, te su otkrili da je prvo reagirala na oštre rubove, a znanstveno to znači da obrada slike počinje jednostavnim oblicima kao što su ravni rubovi. Razvoj tehnologije skeniranja slika 1960-ih godina, omogućeno je računalima dobivanje digitalnih slika. Razvojem tehnologije optičkog prepoznavanja znakova (eng. *OCR optical character recognition*) moguće je bilo prepoznati bilo kakav tekst isprintan bilo kojim fontom. Izumom inteligentnog prepoznavanja znakova (eng. *ICR intelligent character recognition*) postalo je moguće prepoznati tekst koji je pisan rukom. Te tehnologije se koriste još uvijek u nekim primjerima kao što su prepoznavanje tablica vozila ili obradu dokumenta.

Neuroznanstvenik David Marr je 1982. godine uveo algoritme za strojeve kojima se mogu otkriti rubovi, kutovi i ostali slični osnovni oblici. Istodobno znanstvenik Kunihiko Fukushima je razvio neuronsku mrežu nazvanu Neocognitron koja je mogla prepoznati obrasce te je sadržavala konvolucijske slojeve.

Prve aplikacije za prepoznavanje lica koje rade u stvarnom vremenu su se pojavile tek 2001. godine te su se oko 2010. godine počele pojavljivati baze podataka koje sadržavaju mnoštvo označenih slika koje su postale temelj za razvoj konvolucijskih neuronskih mreža. [18]

3.2. Primjena računalnog vida

Računalni vid je danas u velikoj potražnji te se koristi u mnogim različitim područjima kao što su zdravstvo, proizvodnja, robotika i slično. Međutim najčešći zadaci za čije se rješavanje primjenjuje računalni vid su:

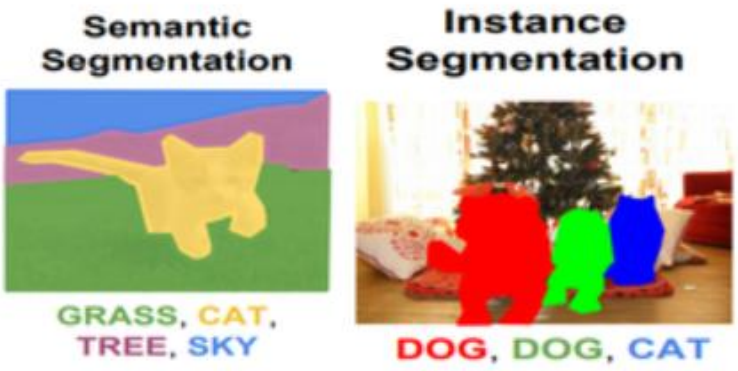
- **Klasifikacija slika**: temeljni zadatak u prepoznavanju, cilj je razumijevanje i kategoriziranje slike kao cjeline pod određenom oznakom. Klasifikacija slika odnosi se na slike na kojima se pojavljuje i analizira samo jedan objekt. [19]
- **Detekcija objekta**: tehnika za identificiranje objekata prisutnih na slikama i video zapisima. Jedna je od najvažnijih primjena strojnog i dubokog učenja a glavna zadaća je naučiti strojeve da razumiju što je na slici. Na slici 19. prikazana je razlika između klasifikacije slike i detekcije objekta.

Slika 19. Razlika između klasifikacije slike i detekcije objekta

- **Praćenje objekata**: tehnika računalnog vida koja se koristi u videozapisima. Cilj joj je prvo otkriti objekt te ga zatim pratiti. Ova tehnika je veoma bitna u autonomnim vozilima kod prepoznavanja ostalih sudionika u prometu te njihovo praćenje u stvarnom vremenu.
- **Segmentacija slika**: daljnje je proširenje detekcije objekta di se prisutnost objekta označava s maskom po pikselima generiranim za svaki objekt na slici. Ona je preciznija zbog toga što ne crta granični okvir nego određuje piksele koji čine taj objekt. Postoje dvije vrste segmentacije:

- **Semantička segmentacija:** svi objekti iste klase tvore jednu klasifikaciju te su obojeni istom bojom**;**
- **Segmentacija instance:** objekti iste klase se smatraju različitim, te su si objekti obojeni različitom bojom.

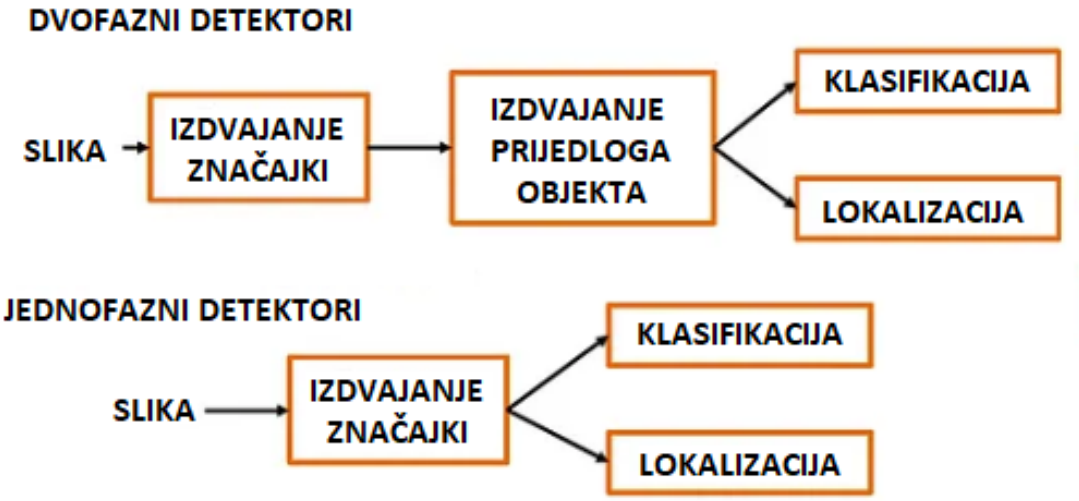
Razlika između semantičke segmentacije i segmentacije instance su prikazane slikom 20.



Slika 20. Razlika između semantičke segmentacije i segmentacije instance. [20]

3.3. Detektori objekta

Detektori objekta imaju jedan od najbitnijih izazova u računalnom vidu a to je detekcija objekta. Zadatak im je locirati i nacrtati granični okvir oko samog objekta. Algoritmi za otkrivanje objekata mogu se podijeliti u dvije kategorije a to su dvofazni i jednofazni detektori, koje ću detaljnije opisat u sljedećem dijelu rada.



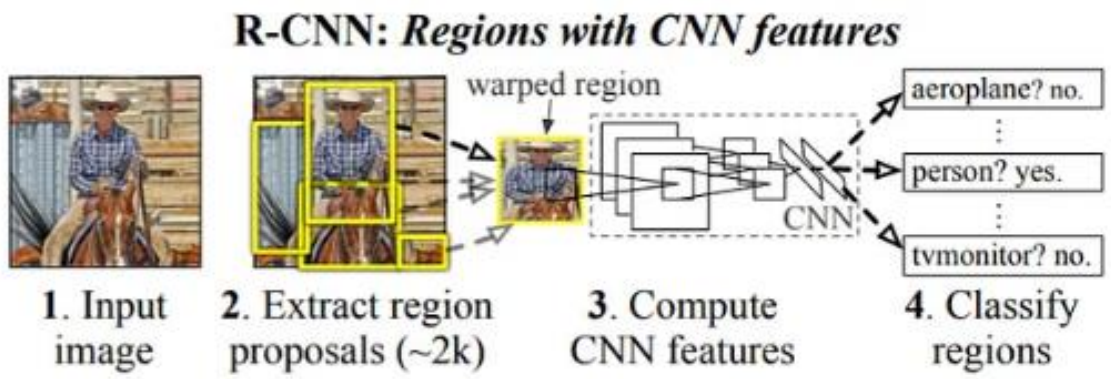
Slika 21. Pojednostavljeni prikaz dvofaznog i jednofaznog detektora

3.3.1. Dvofazni detektori

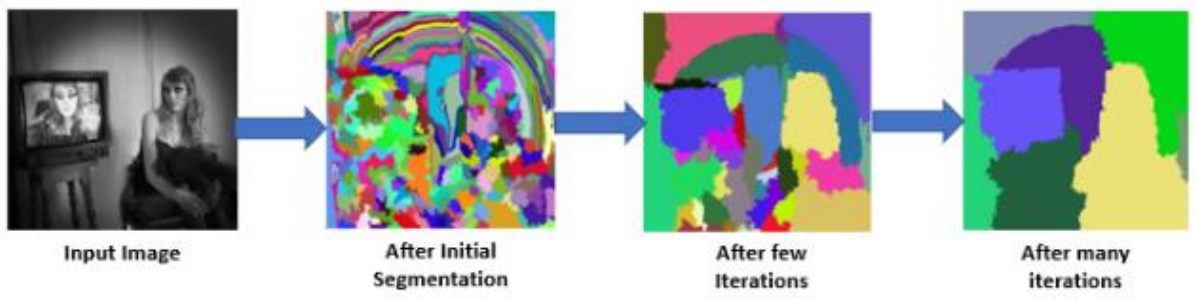
Dvofazni detektori se kao što im ime sugerira sastoje od dvije faze. Prva faza služi za izdvajanje regije dok se u drugoj fazi vrši klasificiranje i daljnja lokalizacija objekta. Dvofazni detektori su veoma moćni ali su ujedno i relativno spori. U nastavku rada bit će nabrojani neki od najčešćih dvofaznih detektora.

3.3.1.1. R-CNN

Konvolucijska neuronska mreža bazirana na regiji (eng. *R-CNN region-based convolutional neural network*) je jedna od prvo nastalih modela za detekciju objekta. R-CNN počinje s dijeljenjem ulazne slike u više malih regija koji se nazivaju kandidati za regije (eng. e*xtract region proposals*) koji se generiraju s pomoću vanjskih metoda kao što su Selective Search, što je prikazano na slici 23.. Nakon što su predložene regije generirane one se prosljeđuju kroz CNN za izdvajanje značajki gdje se detektiraju značajke te se klasificiraju objekti. Zbog toga što se klasificiraju objekti samo na predloženim regijama a ne na cijeloj slici, R-CNN je brži od klasične CNN.



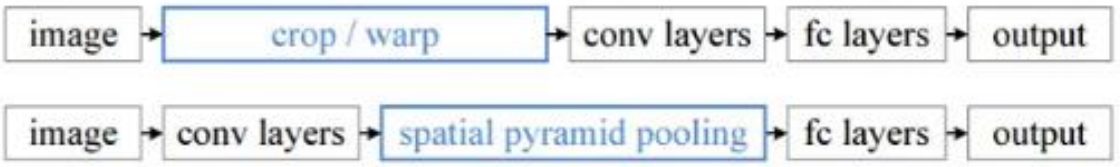
Slika 22. Algoritam R-CNN jednofaznog detektora [21]



Slika 23. Prikaz primjene Selective Search algoritma za stvaranje malih regija [21]

3.3.1.2. SPP-Net

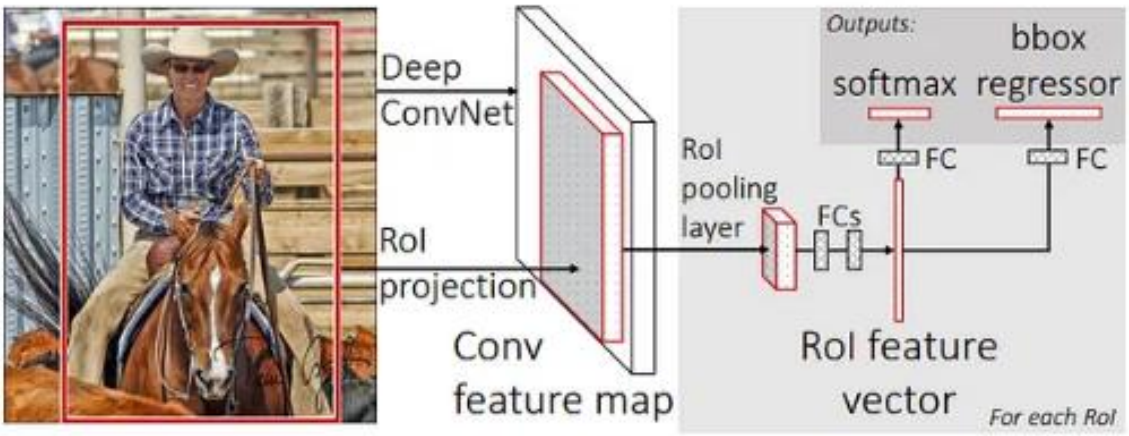
Prostorno piramidalno udruživanje (eng. *SPP spatial pyramid pooling*) služi za rad na bilo kojoj veličini slike bez potrebe za mijenjanje veličine na fiksnu, zbog čega neće doći do gubitka informacija i iskrivljenja slika. Ovo je brža metoda od R-CNN zato što se konvolucija provodi samo jednom. Razlika u arhitekturi između R-CNN i SPP-net mreže prikazana je na slici 24..



Slika 24. Razlika u arhitekturi R-CNN (gore) i SPP-net mreže (dolje) [21]

3.3.1.3. Brzi R-CNN

Princip je sličan kao i kod R-CNN, ali umjesto prijedloga regija u CNN šaljemo ulaznu sliku kako bismo generirali mapu konvolucijskih značajki. Nakon toga se identificiraju predložene regije poslije čega imamo RoI filter koji je ujedno i ulaz u potpuno povezani sloj gdje se korištenjem softmax funkcije vrši klasificiranje predloženih regija te se dodaje granični okvir.

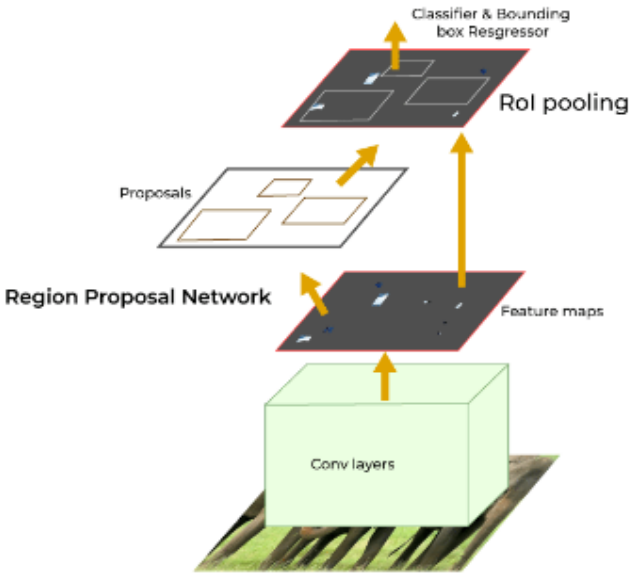


Slika 25. Prikaz arhitekture brze R-CNN mreže [21]

3.3.1.4. Brži R-CNN

Brži R-CNN je najsuvremenija arhitektura detekcije objekta obitelji R-CNN. Ova jednostavna arhitektura ne samo da detektira objekte unutar slike, već i locira objekte točno na slici. Brža R-CNN se sastoji od dvije komponente a to su mreža regionalnih prijedloga (eng. *RPN region proposal network*) i brzog R-CNN detektora. Slika se daje na ulaz u CNN, koja daje kartu

konvolucijskih značajki. Zatim se koristi zasebna mreža za predviđanje prijedloga regije, koji se nakon toga preoblikuju korištenjem RoI. Na kraju se nalazi potpuno povezani sloj koji klasificira objekti i stvara granični okvir.



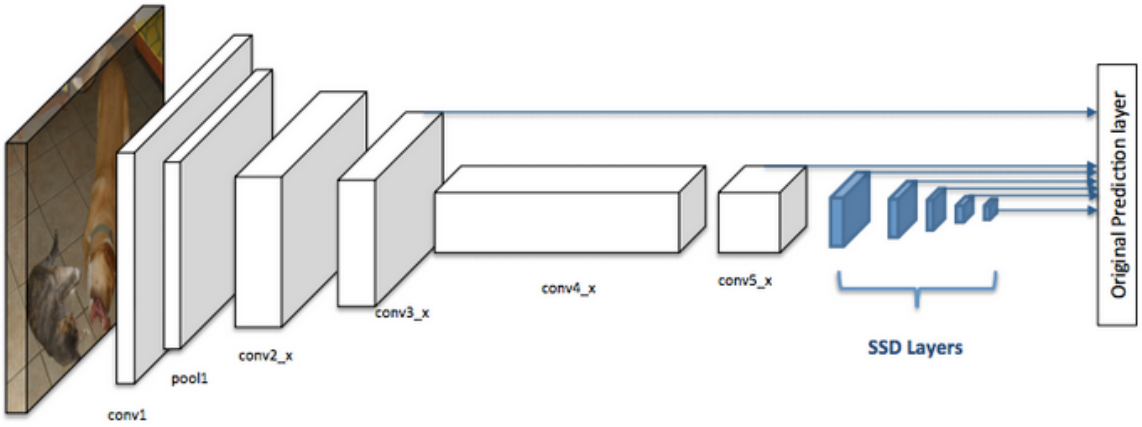
Slika 26. Arhitektura brže R-CNN [22]

3.3.2. Jednofazni detektori

Jednofazni detektori objekata u istoj fazi vrše klasifikaciju i generiranje graničnih okvira oko pronađenog objekta. Jednostavnije su arhitekture i brži su od dvofaznih detektora, zbog čega se koriste za zadatke koji se odvijaju u stvarnom vremenu. Što se točnosti tiče tu su i dalje dvofazni detektori nešto bolji ali razvojem novih algoritama jednofazni detektori su sve moćniji i snažniji. Predstavnici jednofaznih detektora su jednonamjenski SSD (eng. *single shot detection*) i YOLO (eng. *you only look once*) te ću ih detaljnije opisati u nastavku.

3.3.2.1. SSD detektor

Jednonamjenski detektor se sastoji od modela okosnice i SSD glave. Model okosnice je unaprijed obučena mreža za klasificiranje slika kao ekstraktor značajki. Glava SSD-a predstavlja jedan ili skup više konvolucijska sloja. Zbog učinkovitosti i brzine ovi modeli se mogu koristiti za primjenu u stvarnom vremenu.



Slika 27. Arhitektura konvolucijske neuronske mreže sa SSD detektorom [24]

3.3.2.2. YOLO detektor

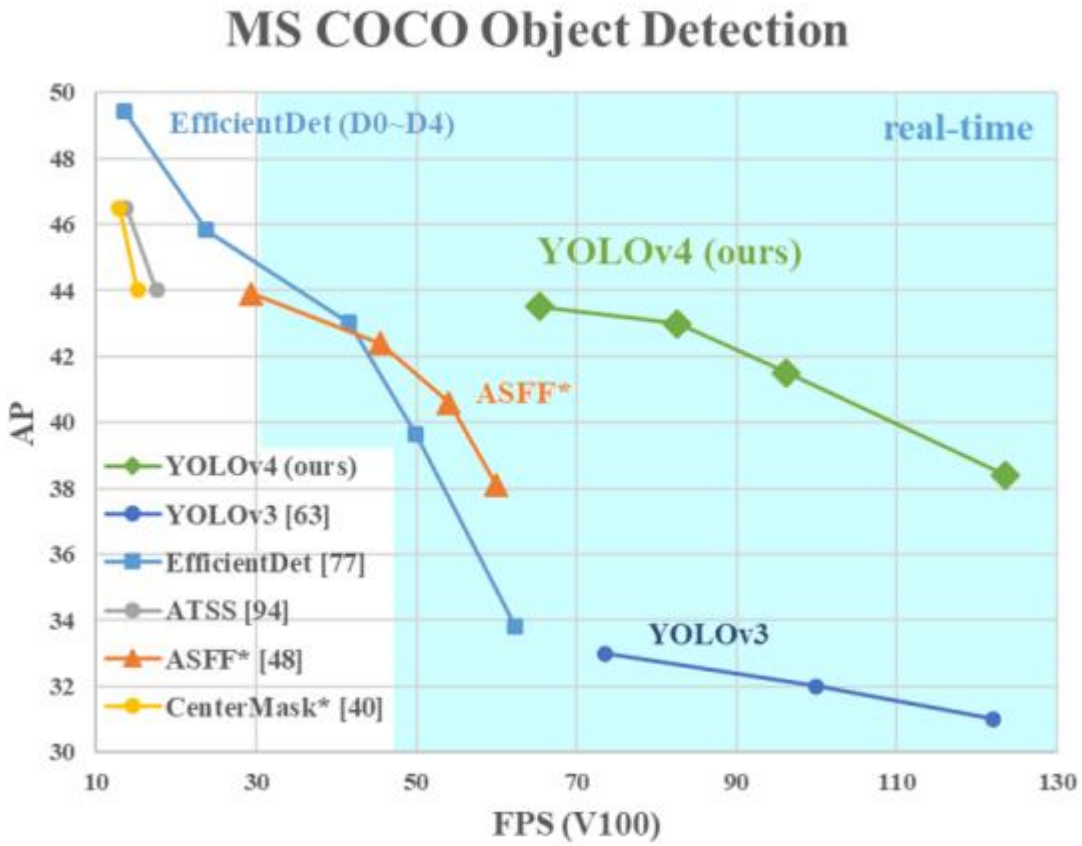
YOLO detektor razvijen od strane Joseph Redmon-a, neuronska je mreža koja daje predviđanja graničnih okvira i vjerojatnosti klasa od jednom. Zbog drugačijeg pristupa, YOLO je nadmašio druge algoritme za detekciju objekata u stvarnom vremenu. Za vrijeme testiranja gleda cijelu sliku tako da se njegova predviđanja temelje na globalnom kontekstu na slici. Zbog svoje brzine kao i SSD detektor, YOLO se primjenjuje u stvarnom vremenu. Princip rada YOLO algoritma možemo podijeliti na 6 sljedećih koraka:

1. Ulazna slika prolazi kroz konvolucijsku neuronsku mrežu kako bi se izdvojile značajke iz slike.
2. Značajke prolaze kroz niz potpuno povezanih slojeva koji predviđaju klase i koordinate graničnih okvira.
3. Slika se dijeli u mrežu ćelija, od čega je svaka ćelija odgovorna za predviđanje skupa graničnih okvira i vjerojatnosti klase.
4. Izlaz iz mreže je skup graničnih okvira i vjerojatnosti klase za svaku ćeliju.
5. Granični okviri se filtriraju uz pomoć algoritma za naknadnu obradu koji uklanja okvire koji se preklapaju i odabire okvir s najvećom vjerojatnošću.
6. Konačni izlaz je skup predviđenih graničnih okvira i oznaka klase za svaki objekt sa slike. [25]

Od svog predstavljanja 2015. godine razvijene su sljedeće verzije:

- **YOLOv1**

- **YOLOv2**: ili YOLO 9000 je napredna verzija prošlog modela. Jedna od najvećih razlika je korištenje takozvanih sidrenih okvira koje CNN predviđa. Još jedna razlika je to da se ulazna slika šalje kroz CNN u više razmjera, što omogućuje modelu otkrivanje objekata različitih veličina. Također koristi se i drugačija funkcija gubitka koja pomaže modelu da brže konvergira. [25]
- **YOLOv3**: treća verzija YOLO algoritma koja koristi tehniku nazvanu FPN (eng. *feature pyramid network*) za izdvajanje značajki sa slike u različitim razmjerima što omogućuje modelu detekciju objekata različitih veličina. [25]
- **YOLOv4**: ključna razlika u odnosu na prethodnike je ta da se koristi naprednija arhitektura neuronskih mreža SPP (eng. *spatial pyramid processing*) za izdvajanje značajki sa slike u različitim rezolucijama. Koristi se i tehnika CSP (eng. c*ross-stage partial connector*) kojim se postiže bolja točnost modela. [25]

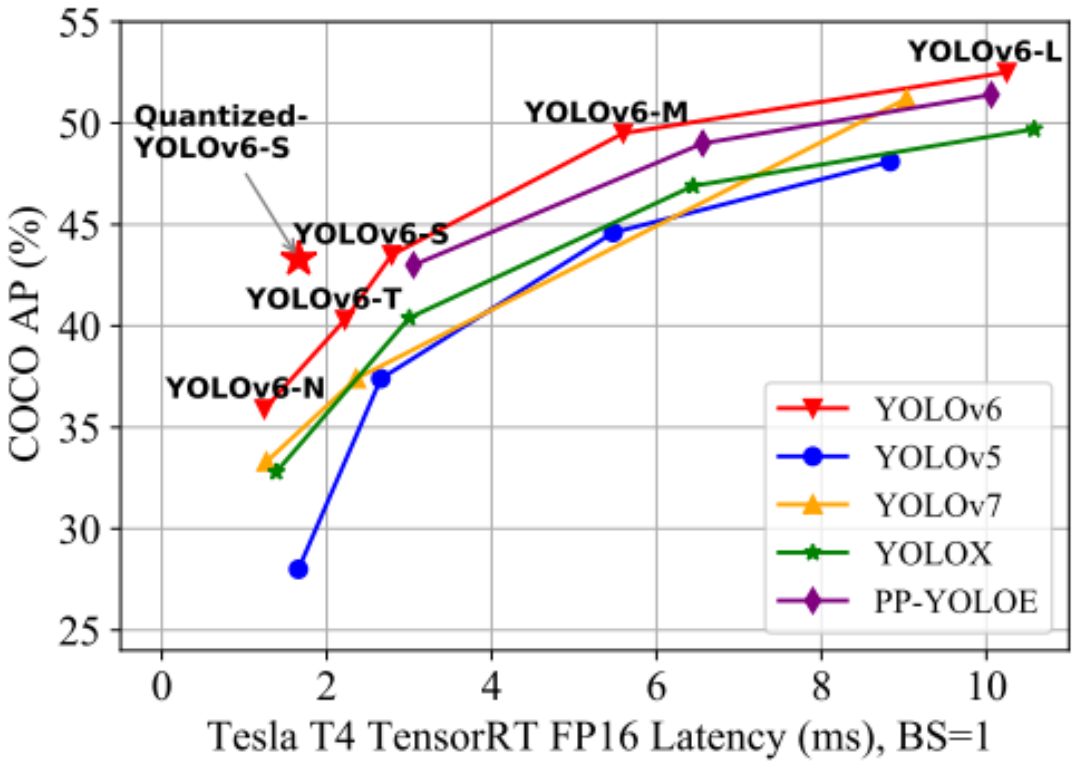


Slika 28. Usporedba performansa YOLOv4 s YOLOv3 modelom [26]

- **YOLOv5**: predstavljen 2020. s razlikom da koristi učinkovitiju arhitekturu neuronske mreže nazvane EfficientNet, koja je dizajnirana da bude učinkovita u smislu računanja i uporabe memorije a da pritom ne gubi na točnosti. Razlika je i ta da se više ne koriste sidreni okviri umjesto kojih se koristi još jedan konvolucijski sloj za izravno

predviđanje koordinata graničnih okvira što omogućuje modelu da bude fleksibilniji i prilagodljiviji na različitim oblicima i veličinama objekta. YOLOv5 također koristi tehniku CmBN (eng. *cross mini-batch normalization*) koja služi za povećanje točnosti modela. [25]

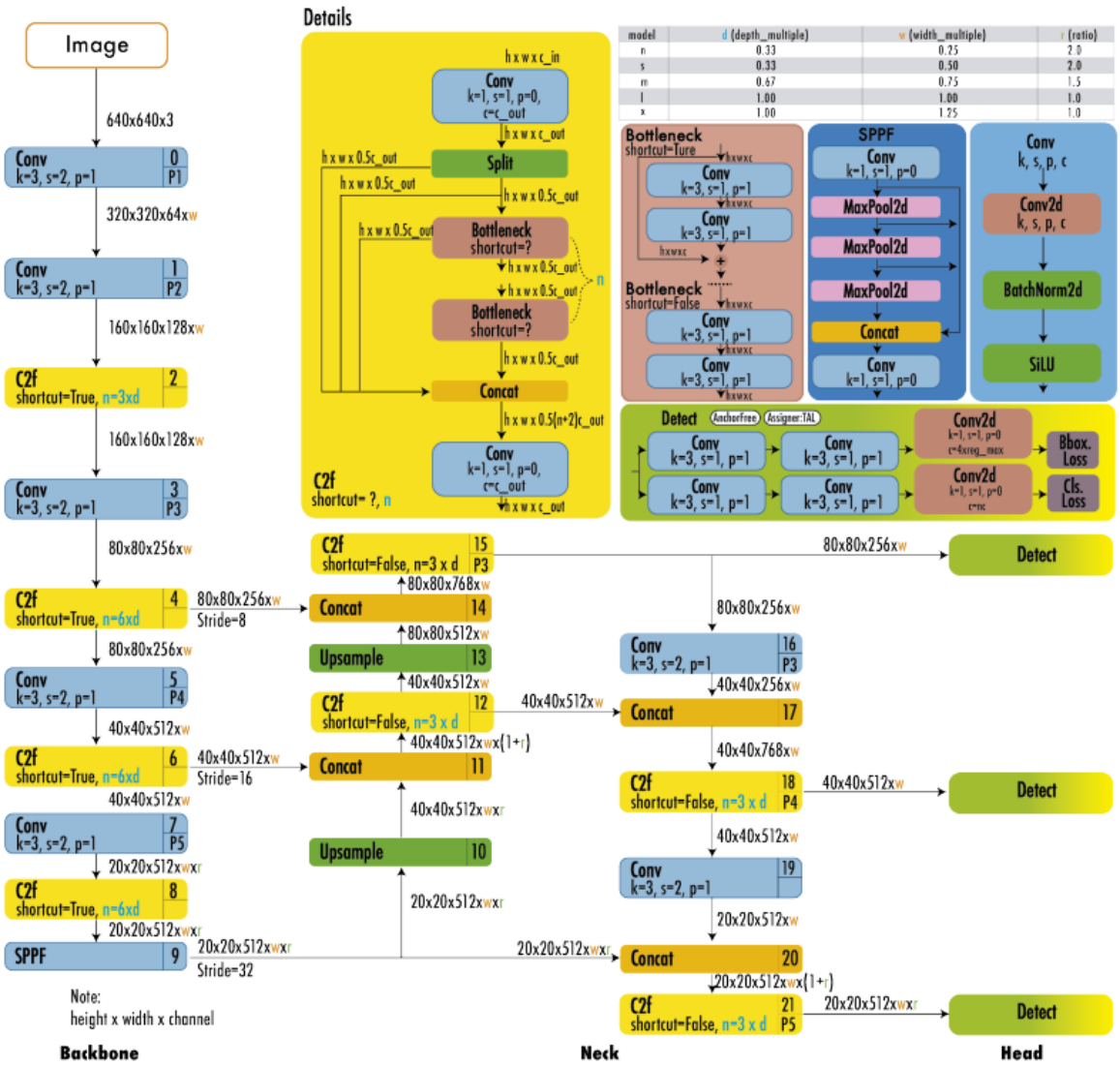
- **YOLOv6**: značajna razlika od prethodnika je ta da se koristi učinkovitija i laganija arhitektura neuronske mreže, zbog čega YOLOv6 radi brže i s manje računalne snage. [25]



Slika 29. Usporedba točnosti i brzine YOLOv6 s YOLOv5 [25]

- **YOLOv7**: implementirani su sidreni okviri čije korištenje dovodi do smanjenja broja lažno pozitivnih rezultata. Implementirana je nova funkcija gubitaka za poboljšanje performansi. Ujedno ona rješava i poteškoće u otkrivanju malih objekta prilagođavanjem težine gubitka na dobro klasificiranim primjerima i stavljanjem većeg naglaska na zahtjevne primjere za otkrivanje. [25]
- **YOLOv8**: posljednja verzija koja je izašla 2023. godine. Ima 5 verzija: YOLOv8n (nano), YOLOv8s (mala), YOLOv8m (srednja), YOLOv8l (velika) i YOLOv8x (jako velika). YOLOv8 ima sličnu okosnicu kao i YOLOv5 s nekim promjenama na CSPLayeru, koji se sad naziva C2f modul. C2f modul kombinira značajke visoke razine za poboljšavanje točnosti detekcije. YOLOv8 nudi model semantičke segmentacije pod

nazivom YOLOv8-Seg koji je postigao vrhunske rezultate u detekciji objekata i semantičkoj segmentaciji uz zadržavanje visoke brzine i učinkovitosti. Upravo ću ovu verziju koristiti za praktični dio ovog rada. [27]



Slika 30. YOLOv8 arhitektura [27]

4. PRAKTIČNI DIO – DETEKCIJA KOROVA NA PLANTAŽI JAGODA

Za praktični dio rada, cilj je bio napraviti YOLOv8 model za detekciju korova na plantaži jagoda OPG-a Makarun. OPG Makarun se već 29 godina bavi uzgojem jagoda sorti Clery i Alba na površini od oko 3000 metara kvadratnih.

4.1. Baza podataka

Baza podataka za ovaj rad je u potpunosti napravljena od slika s plantaža jagoda OPG Makarun. Iz razloga što je problematika ovog rada specifična i usko vezana uz ovu plantažu, nije bilo moguće preuzeti neku od gotovih baza podataka s kojom bi se vrlo jednostavno i brzo mogao početi trenirati model, te bi i rezultati bili sigurno bolji zbog veličine tih online baza podataka. Slike iz ove prilagođene baze podataka možemo podijeliti u 2 vrste a to su:

- Slike nastale na plantaži koje uključuju korov kao i pozadinu. Primjer ovakvih slika je slika 31.

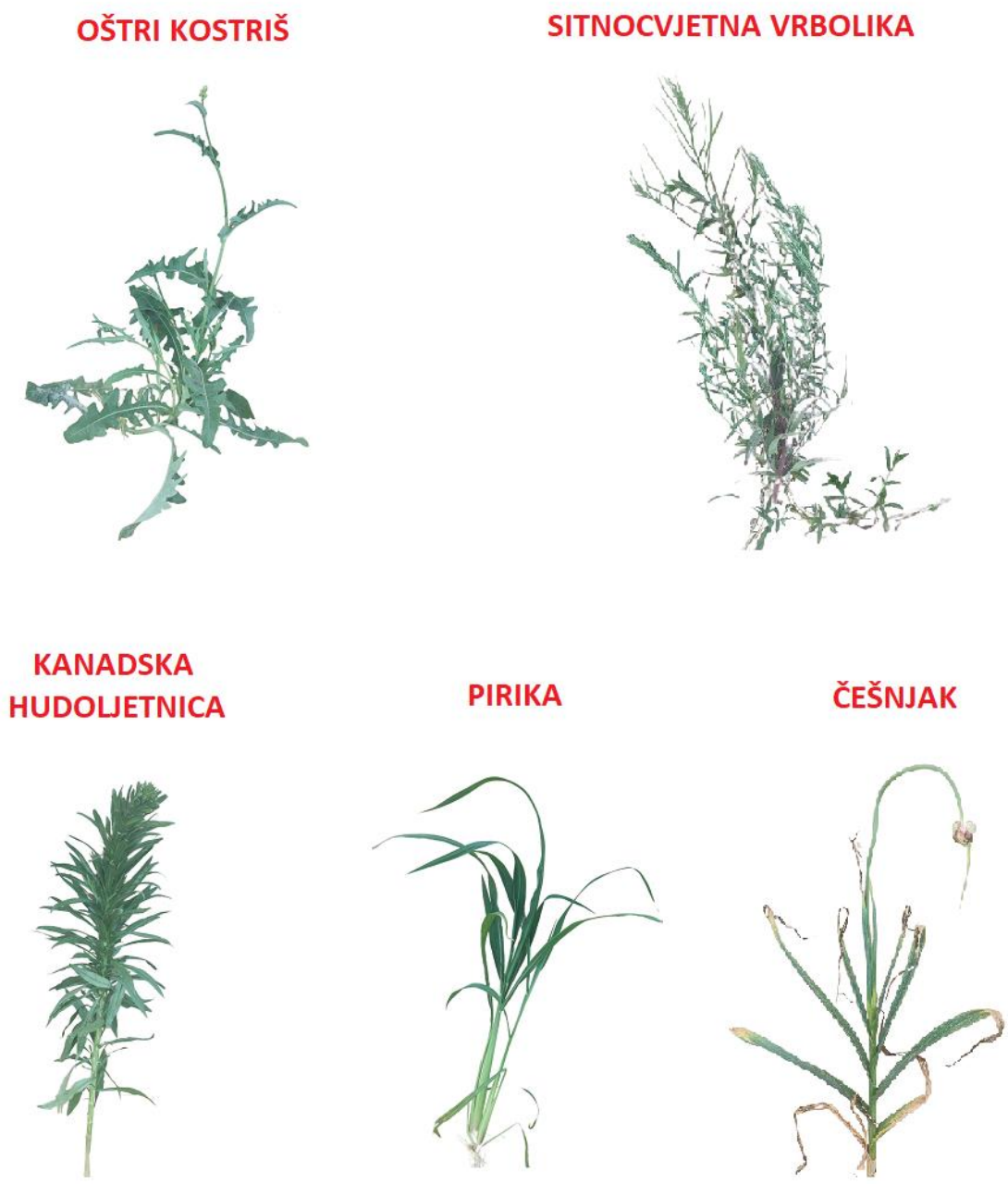
Slika 31. Slika korova na plantaži

- Slike iščupanog korova na kartonu, slika 32. Slike su fotografirane na taj način kako bi se jednostavnije mogla izbrisati pozadina te bi dobili samo sliku korova, kao što je to na slici 33. Pozadine sa slika su brisane s pomoću online dizajnerskog programa Canva, koji ima opciju prepoznavanja i brisanja pozadine.

Slika 32. Slika korova na kartonu

Slika 33. Slika korova s izbrisanom pozadinom

Prolaskom kroz plantažu, uz sadnice jagoda uočene su 4 vrste korova te češnjak koji je namjerno sađen svakih desetak metara kako bi odbijao voluharice koji se smatraju jednim od najvećih štetnika u plantaži jagoda. Od korova uočeni su: : oštri kostriš, pirika, sitnocvjetna vrbolika i kanadska hudoljetnica. Pronađeno je još nekoliko vrsta korova međutim oni su se pojavljivali zanemarivom broju primjeraka te ne bi imali dovoljno njihovih fotografija tako da su oni zanemareni. Za svaku od prethodno navedenih biljki, napravio sam 30 do 50 fotografija u plantaži, te sam za svaku biljku još dodao četrdesetak slika na kartonu, kod kojih će biti uklonjena pozadina. Što znači da za bazu podataka ukupno sam sakupio oko 250 slika.

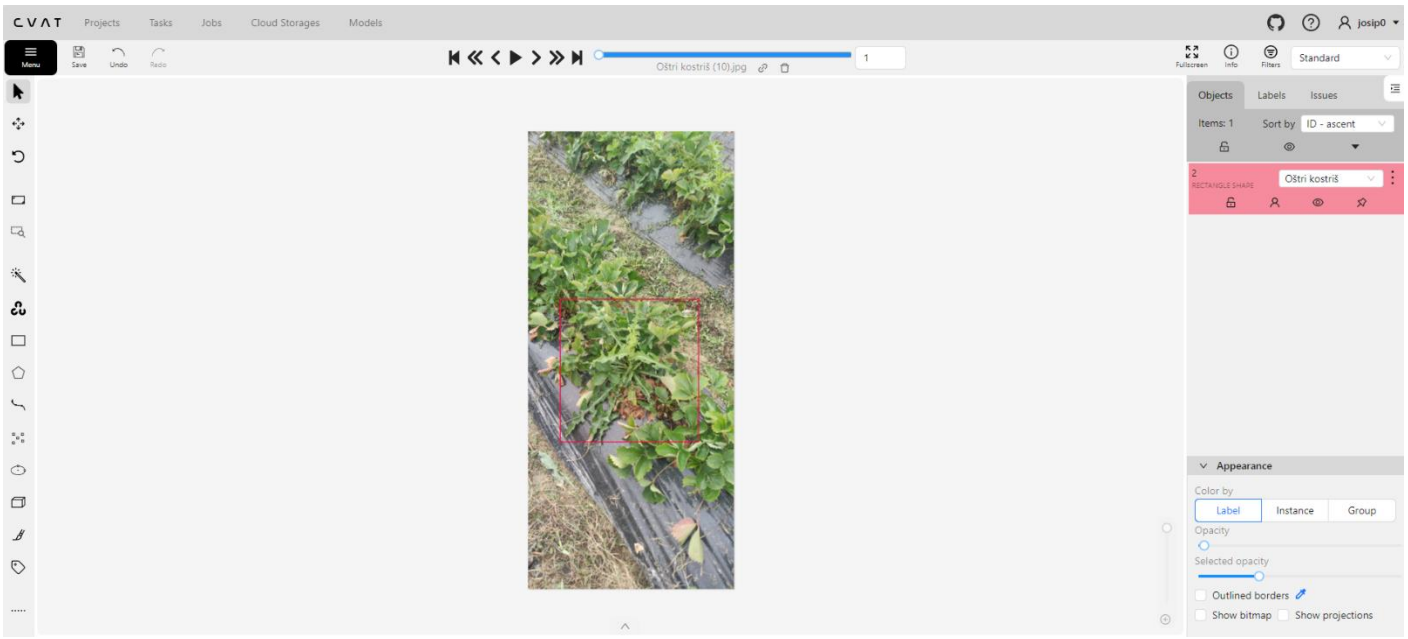


Slika 34. Prikaz biljka za koje ćemo trenirati model

Nakon što smo prikupili slike za bazu podataka, potrebno je krenuti s označavanjem biljaka na slikama. To sam radio u programu zvanom Cvat.AI. Kako bismo mogli krenuti s označavanjem prvo se je potrebno prijaviti te nakon toga otvoriti novi task. Zatim je u tasku potrebno dodati oznake za prethodno nabrojane biljke koje želimo da model detektira. Kada smo to napravili

preostaje nam još samo označiti fotografije na kojima ćemo vršiti označavanje te ih učitati u program.

Sam proces označavanja je poprilično monoton i dugotrajan. Potrebno je nacrtati okvir koji što bolje označava svaku biljku koju smo prepoznali na slici, te na desnoj strani prozora odabrati o kojoj biljci se radi. Proces označavanja je prikazan na slici 35.



Slika 35. Proces označavanja slika

Kada smo gotovi s označavanjem svih slika, prvo je potrebno spremiti rad, te preuzeti datoteke oznaka. Veoma je bitno da odaberemo ispravan format prilikom preuzimanja što je u ovom slučaju YOLO 1.1.

Preuzeta datoteka sadrži se od mnoštvo .txt datoteka u kojima se nalazi podatak o položaju okvira na slici. Prvi broj označava o kojoj biljci se radi, druga dva broja označavaju koordinate prve točke pravokutnog okvira dok zadnje dvije brojke označavaju drugu točku pravokutnog okvira kojim je biljka označena. Primjer zapisa položaja okvira je prikazan na slici 36.

```
0 0.529953 0.589960 0.771602 0.451853
3 0.547341 0.402460 0.481687 0.393518
```

Slika 36. Izgled zapisa podataka o položaju okvira

4.2. Treniranje modela

Treniranje modela ćemo raditi uz pomoć Google Drivea, u kome ćemo napraviti mapu nazvanu Yolov8. U toj mapi bit će potrebno napraviti još dvije mape naziva images i labels. U mapi images dodat ćemo mapu train te ćemo u nju dodati slike koje će služiti za treniranje modela. U mapu labels također dodamo mapu train, te u nju stavimo sve .txt datoteke koje smo preuzeli iz programa Cvat.AI. Za provođenje treninga potrebna nam je još jedna datoteka koju ćemo nazvati config.yaml. Kako bismo nju napravili, potrebno je otvoriti notepad, te u njega upisati kod sa slike 37.

```
path: '/content/gdrive/My Drive/Yolov8/'
train: images/train
val: images/train

# Classes
names:
  0: Oštri kostriš
  1: Pirika
  2: Sitnocvjetna vrbolika
  3: Češnjak
  4: Kanadska hudoljetnica
```

Slika 37. Kod koji se nalazi u datoteci config.yaml

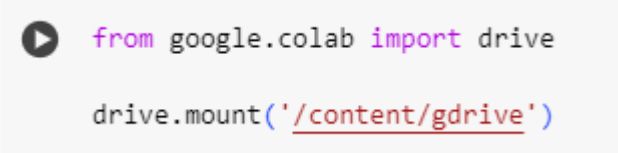
Pod path moramo navesti putanju na Google Driveu koja će se smatrati glavnom mapom, iz koje će kasnije kretati sve ostale naredbe. Pod train i val moramo navesti putanju iz prethodno navedene glavne lokacije do slika na kojima će se vršiti treniranje i provjera. Zadnji dio config.yaml datoteke su imena klasa koje moramo napisati te numerirati isto onako kako smo to učinili u programu Cvat.AI. Kao što možemo vidjeti na slici 36. 0 na početku prvog reda označava da se radi o oštrom kostrišu, dok 3 u drugom redu označava Češnjak. Kada smo gotovi, s unosom koda potrebno je datoteku spremiti pod nazivom config.yaml, te je potrebno u prozoru „Save as type:“ odabrati „All Files“. Nakon što smo spremili datoteku config.yaml potrebno je učitati u Google Drive na lokaciju koja je navedena ranije.

Za samo pokretanje koda treniranja koristit ćemo programski jezik Phyton u programu Google Colab za čiju uporabu nije potrebno nikakvo instaliranje te on pruža besplatan pristup računalnim resursima za provedbu treniranja. Jedino što je potrebno učiniti je dodati Google

Colab datoteku u glavnu mapu Google Drivea tako da pritisnemo desni klik miša, odaberemo Google Colab.

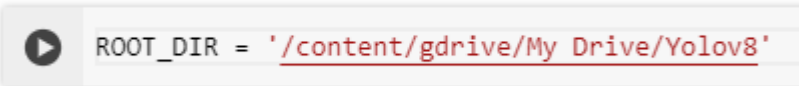
Za samo treniranje bit će potrebno u Google Colab-u izvršiti 5 jednostavnih naredbi koje će biti objašnjene u sljedećem dijelu poglavlja.

Prva naredba prikazana na slici 38. pomoću koje povezujemo Google Colab datoteku sa svojim Google Driveom na kome su podaci za treniranje. Kada upišemo naredbu potrebno je pritisnuti znak za pokretanje te je potrebno potvrditi dozvole za povezivanje Google Drivea na Google Colab.



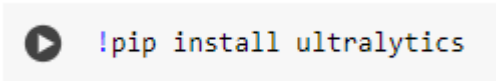
Slika 38. Prva naredba u Google Colab-u

Druga naredba koju moramo pokrenuti je prikazana na slici 39. te s njom postavljamo mapu Yolov8 kao glavnu datoteku, te više ne moramo prilikom pisanja naredba, pisati putanju od početka već nastavljamo samo od glavne datoteke.



Slika 39. Druga naredba u Google Colab-u

Treća naredba koje pokrećemo je prikazana na slici 40. s pomoću nje provodimo instalaciju ultralytics paketa koji sadrži YOLOv8 na trajno mjesto, u našem slučaju na Google Drive.



Slika 40. Treća naredba u Google Colab-u

Četvrta naredba služi učitavanje podatka potrebnih za trening. Također u ovom kodu odabiremo koju verziju YOLOv8 očitavamo, u ovom slučaju očitali smo yolov8n.yaml što znači da očitavamo nano verziju, koja je najmanja i najbrža ali s nešto manjom točnošću u odnosu na malu, srednju, veliku i extra veliku verziju. Zadnji red naredbe označava na koji način ćemo koristiti model, što je u ovo slučaju treniranje te se pozivamo na prije napravljenu config.yaml datoteku u kojoj je opisano na kojim mjestima se nalaze slike odnosno okviri. Zadnje što

zadajemo su epohe treniranja odnosno koliko dugo će se obavljati treniranje. Što stavimo veći broj epoha to možemo očekivati bolje rezultate, međutim u nekom trenutku model dosiže maksimalnu istreniranost za našu bazu podataka te model prestaje napredovati ili napreduje minimalno.

```
import os

from ultralytics import YOLO


model = YOLO("yolov8n.yaml")


results = model.train(data=os.path.join(ROOT_DIR, "config.yaml"), epochs=30)
```

Slika 41. Četvrta naredba u Google Colab-u

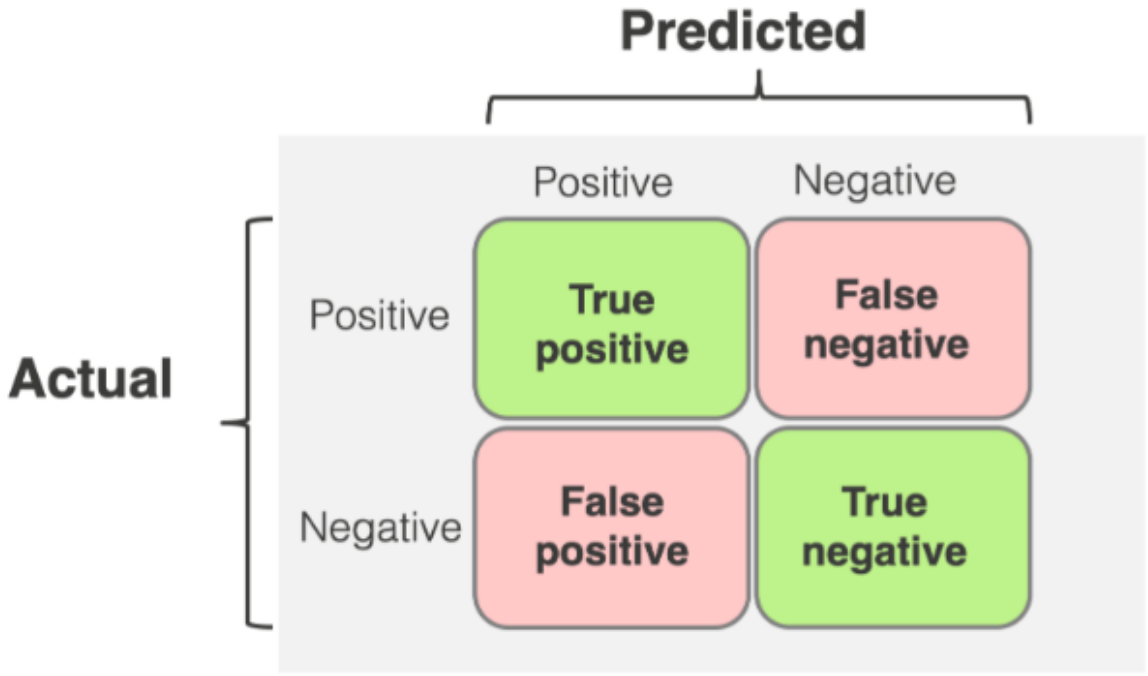
Posljednja naredba koju moramo izvršiti je prikazana na slici 42. s pomoću koje ispisujemo rezultate treniranja. U glavnoj mapi se stvara mapa pod nazivom runs u kojoj će se nalaziti mapa detect unutar koje će se nalaziti mapa train. U mapi train će se nalaziti rezultati našeg treniranja kao i sam model koga možemo kasnije dodatno trenirati.

```
!scp -r /content/runs '/content/gdrive/My Drive/Yolov8'
```

Slika 42. Peta naredba u Google Colab-u

Neke od datoteka koje možemo pronaći unutar mape train a važne su kako bi razumjeli rezultate su:

- **Weights**: sadrži 2 modela nazvana best.pt koji predstavlja najbolji model prilikom treniranja te last.pt koji predstavlja model nakon zadnje epohe treniranja.
- **Confusion_matrix**: je najpopularnija metoda vizualizacije kvalitete klasifikacijskih modela. Iz nje se može iščitati nekoliko bitnih podataka. Način na koji se mogu čitati podaci iz konfuzijske matrice su prikazani na slici 43.



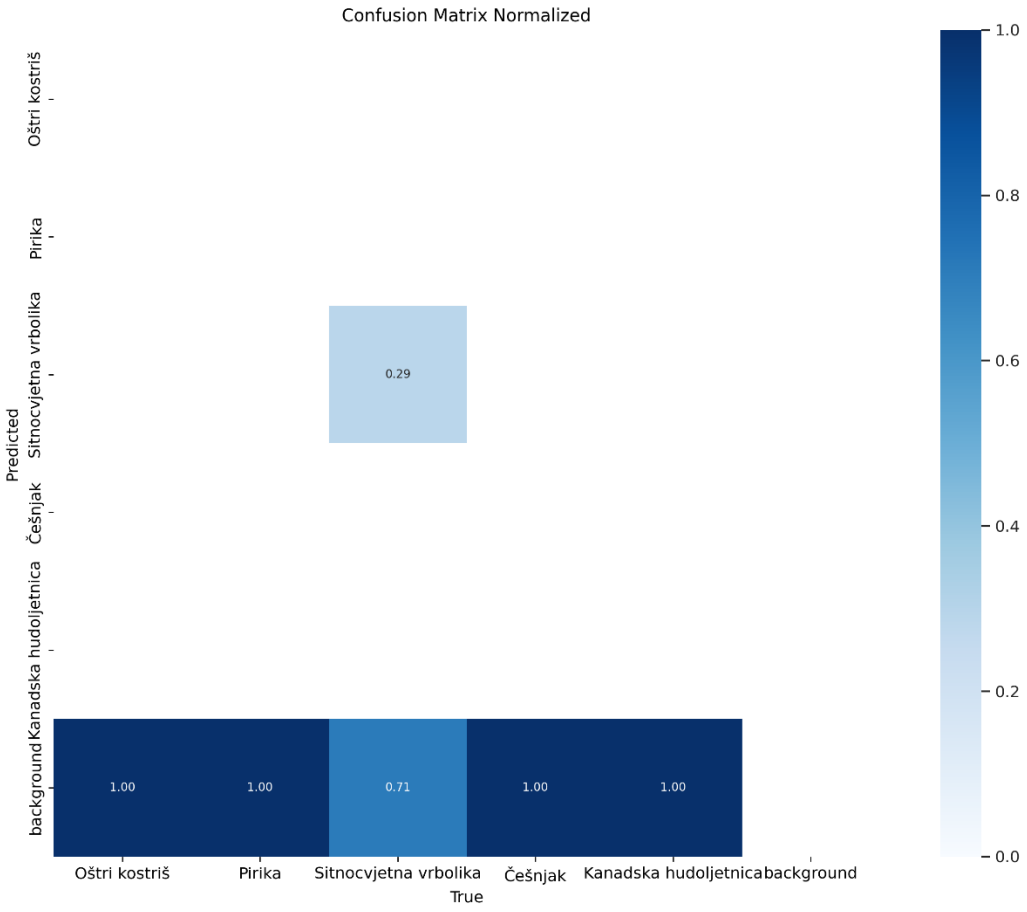
Slika 43. Značenje podatka u konfuzijskoj matrici [28]

- **Results**: je tablica u kojoj su prikazani svi numerički parametri dobiveni u svakoj epohi treniranja. Te iste parametre možemo iščitati u grafičkom obliku u datoteci results.png.
- **Train_batch**: prikazuju slike na kojima se vršio trening, s okvirima koje smo sami postavili. Služe za provjeru dali su slike i okviri točno pripremljeni te za usporedbu sa slikama na kojima je model postavio okvire.
- **Val_batch**: su iste slike na kojima je vršeno treniranje ali s okvirima postavljenim od strane modela. Na njima možemo vidjeti rezultate treniranja. Ako je trening bio uspješan okviri bi trebali biti pravilno pozicionirani te pored svakog okvira piše vrijednost s kojom sigurnošću je model prepoznao objekt sa slike. Što je veća vrijednost to je znak da model bolje i točnije radi.

Treniranje modela podijelit ćemo na 3 dijela, baš onako kako je tekao trening ovog modela. Dobiveni rezultati će biti opisani nakon svakog treniranja te će biti opisani koraci koji su bili poduzeti u svakom od ta 3 dijela, kako bi se dobili bolji rezultati.

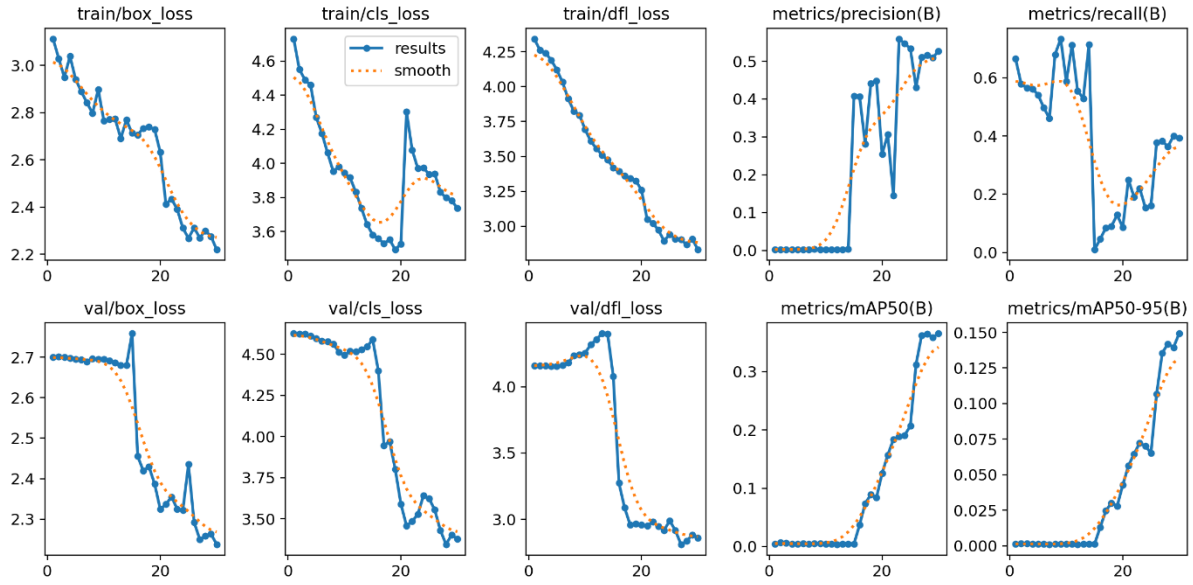
4.2.1. Trening 1

U prvom treniranju za bazu podataka uzete su 117 slike koje su sadržavale korov na plantaži te je treniranje obavljeno na 30 epoha. Od ovog treniranja nisam imao prevelika očekivanja zato što su slike korova na plantaži dosta kompleksne. Dolazi do dosta isprepletenosti korova i sadnica jagoda, ne vide se jasne konture korova te prilikom stavljanja okvira na korove, uočio sam da mnogo puta u okviru se uz korov nalaze i listovi jagoda. Rezultati prvog treniranja su prikazani na slikama 44., 45. i 46.



Slika 44. Konfuzijska matrica za 1. treniranje

Na konfuzijskoj matrici možemo vidjeti da kod 4 biljaka nije došlo niti do jedne detekcije, što je jasan znak da model ne ispunjava naša očekivanja što bi se moglo reći da je i očekivano.



Slika 45. Grafovi rezultata 1. treniranja

Iz grafova rezultata možemo vidjeti da su vrijednosti box_loss, cls_loss i dfl_loss previsoki. Međutim vidimo da i dalje imaju tendenciju pada, što bi moglo značiti da s nastavkom treniranja možemo očekivati bolje rezultate.

Slika 46. Usporedba slika train_batch i val_batch 1. treniranja

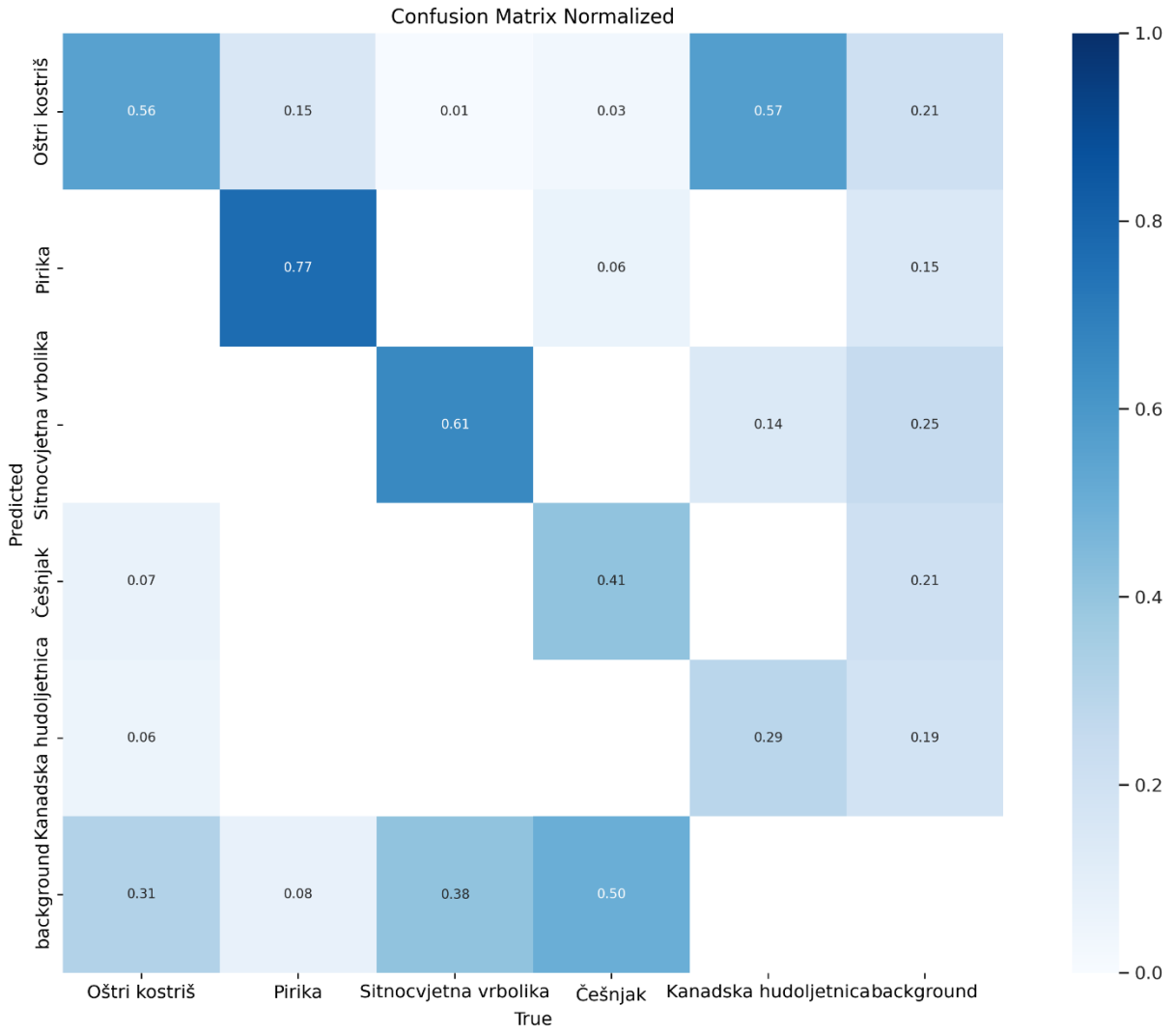
Na slikama val_batch kao što je prikazano na slici 46. vidimo da nema niti jedne detekcije u odnosu na slika s train_batch-a, što potvrđuje sumnje da model trenutno ne radi dobro.

4.2.2. Trening 2

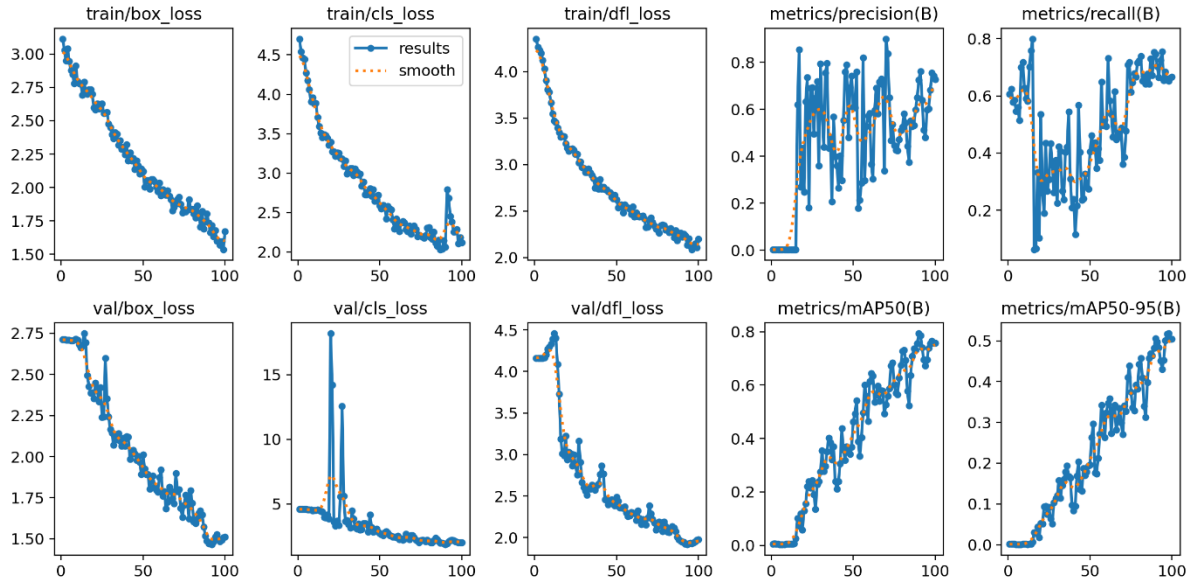
Kako bismo popravili performanse modela odlučio sam se za dodavanje dodatnih slika te povećanju broja epoha treninga. Umjesto da dodam još sličnih slika s plantaže odlučio sam se za opciju dodavanja slika na kojima će biti samo korovi, gdje neće biti interakcije korova s listovima jagoda. U bazu slika na Google Driveu su dodane 63 slike korova bez pozadine, broj

epoha je povećan na 100 te s istim kodom navedenim u prethodnom dijelu rad, pokrenuto je treniranje modela.

Nakon 4 sata treniranja rezultati su sljedeći. Kao što je vidljivo na konfuzijskoj matrici na slici 47. model je počeo prepoznavati biljke, međutim rezultati i dalje mogu biti bolji što možemo vidjeti iz grafova rezultata prikazanih na slici 48. gdje box_loss, cls_loss i dfl_loss i dalje imaju tendenciju pada.

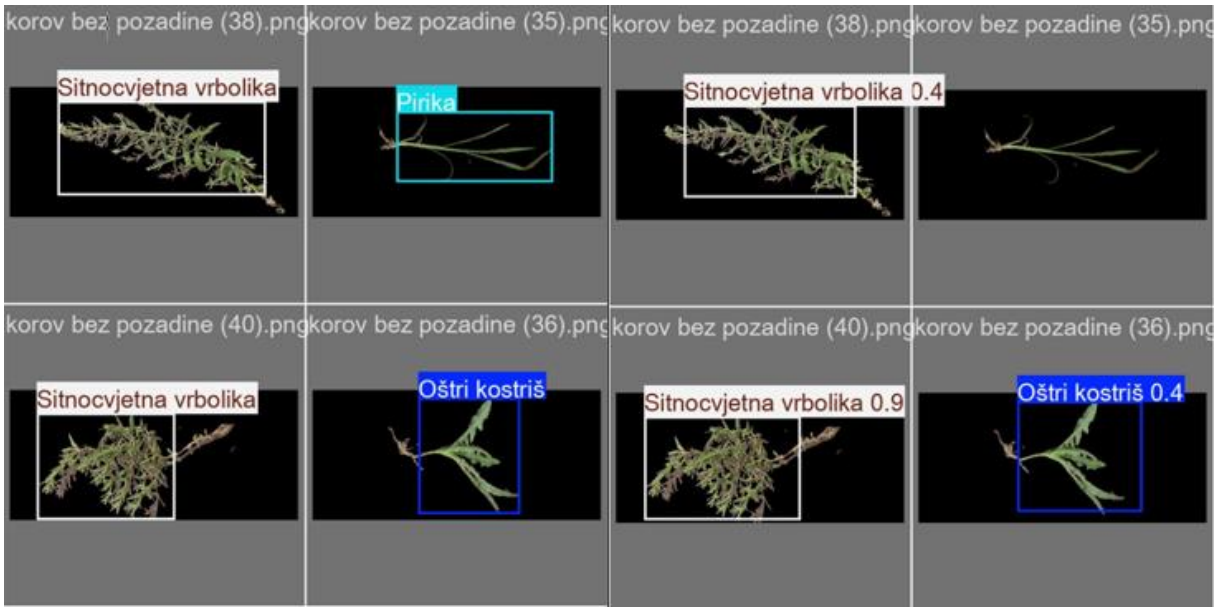


Slika 47. Konfuzijska matrica za 2. treniranje



Slika 48. Grafovi rezultata 2. treniranja

Na slikama 49. i 50. vidimo da model poprilično dobro prepoznaje korov na slikama bez pozadine, dok je kod slika na plantaži također počeo detektirati biljke ali s manjom sigurnošću te imamo više ne detekcija i lažnih detekcija.



Slika 49. Usporedba slika train_batch i val_batch korova bez pozadine 2. treniranja

Slika 50. Usporedba slika train_batch i val_batch korova na plantaži 2. treniranja

4.2.3. Trening 3

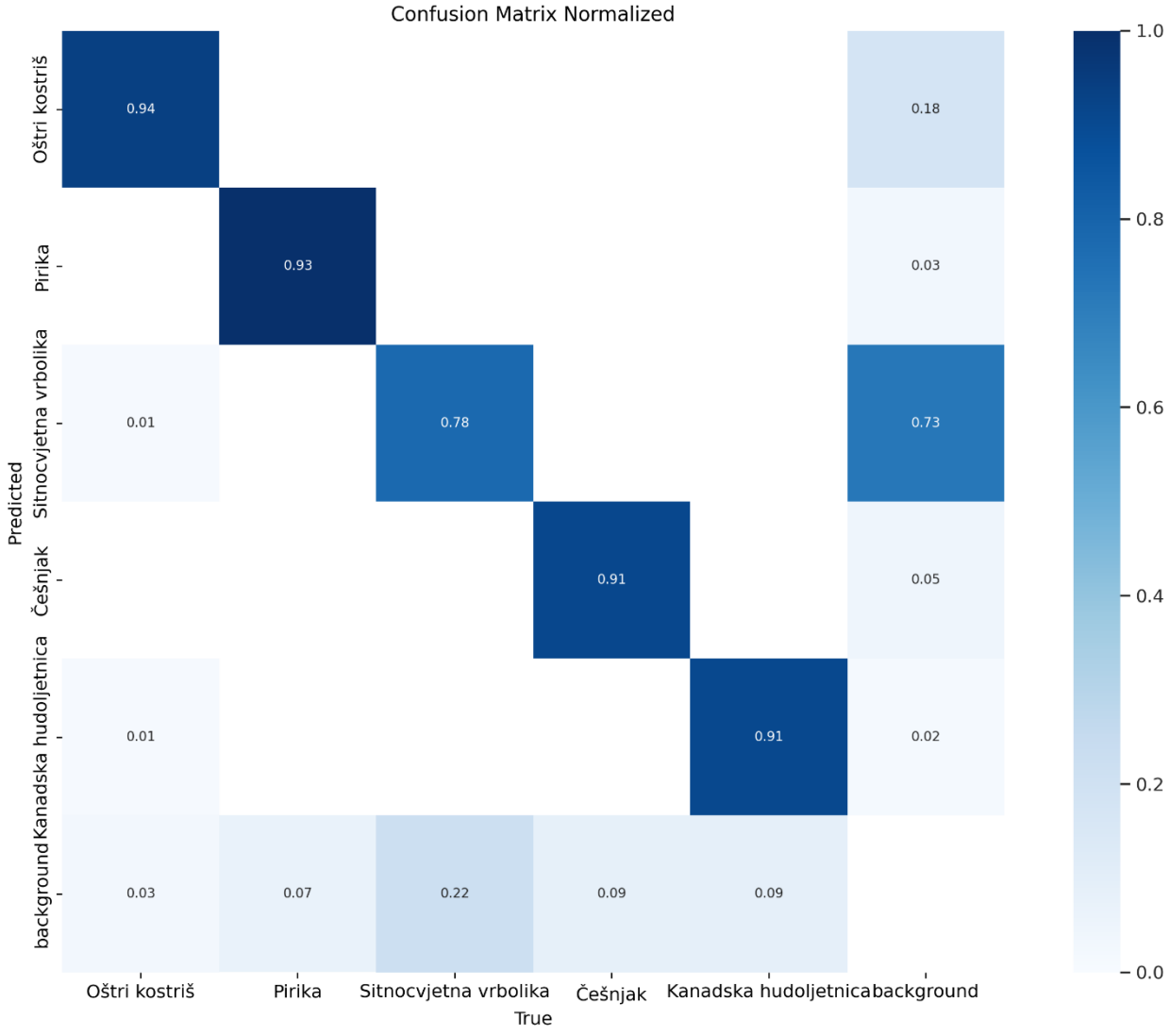
Za sljedeće poboljšanje pripremljeno je još 95 slika korova bez pozadine, što će povećati bazu slika na 275. Treniranje ćemo izvršiti na 40 epoha.

U ovom slučaju želimo nastaviti trenirati prethodni model, zbog čega moramo promijeniti četvrtu naredbu u Google Colab-u, te ona mora izgledati kao na slici 51.

```
from ultralytics import YOLO

model = YOLO('/content/gdrive/My Drive/Yolov8/runs/detect/train/weights/last.pt')

train_results = model.train(data='/content/gdrive/My Drive/Yolov8/config.yaml', epochs=40)
```

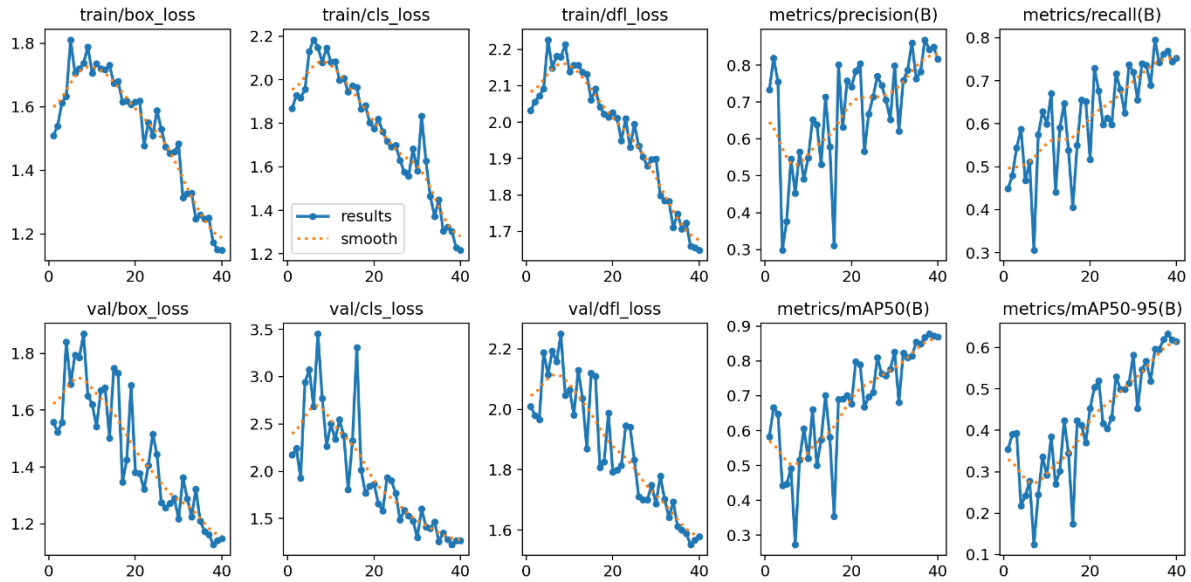
Slika 51. Nova četvrta naredba za nastavak treniranja postojećeg modela

Analizom rezultata nakon 3. testiranja možemo zaključiti da model za prepoznavanje pirike i sitnocvijetne vrbolike radi odlično što je vidljivo iz konfuzijske matrice na slici 52. Model detektira češnjak malo manje dobro međutim i dalje vrlo zadovoljavajuće, dok za detekciju oštrog kostriša i kanadske hudoljetnice bilo bi poželjno popraviti performanse modela.



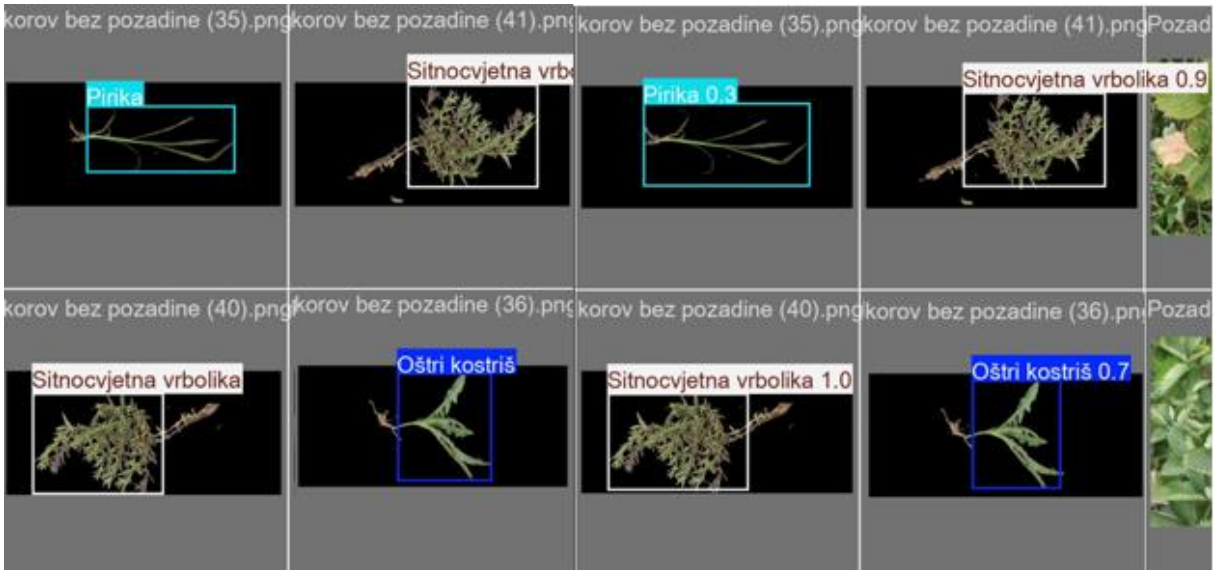
Slika 52. Konfuzijska matrica za 3. treniranje

Što se tiče mogućnosti poboljšanja modela, iz grafova rezultata sa slike 53. možemo vidjeti da i dalje postoji mogućnost napretka s nastavkom treniranja.

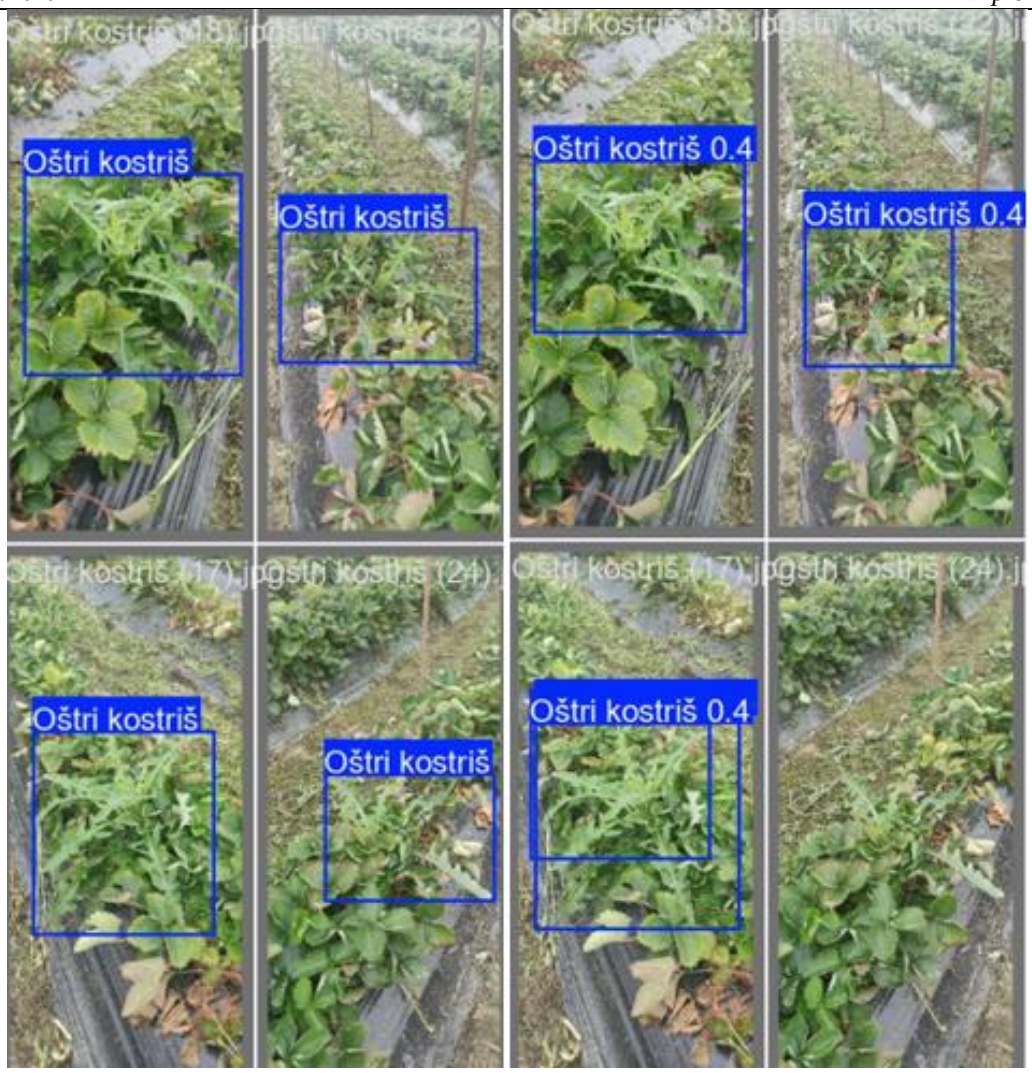


Slika 53. Grafovi rezultata 3. treniranja

Iz slika 54. i 55. možemo vidjeti da model prepoznaje biljke na slikama bez pozadine odlično, dok se kod slika s plantaže tu i tamo dogodi neprepoznavanje i neko lažno prepoznavanje, međutim možemo primijetiti da model negdje prepozna biljku točno tamo gdje ju niti mi nismo označili zato što je teško uočljiva zbog sličnosti sa sadnicama jagoda.



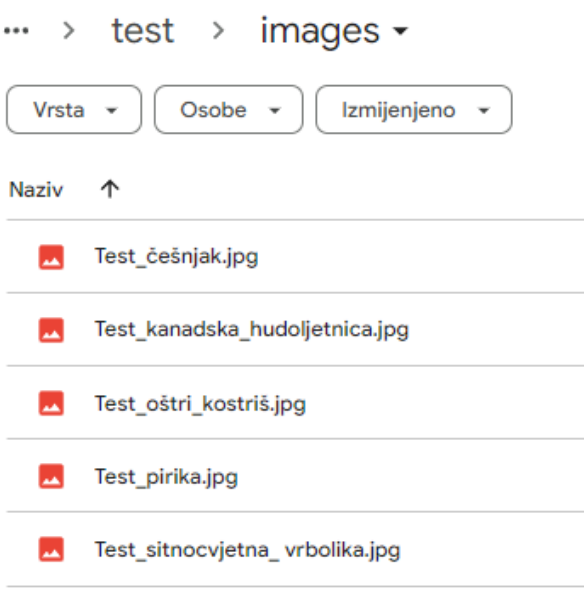
Slika 54. Usporedba slika train_batch i val_batch korova bez pozadine 3. treniranja



Slika 55. Usporedba slika train_batch i val_batch korova na plantaži 3. treniranja

4.3. Testiranje modela

Za potrebe testiranja posljednjeg modela bit će potrebno odabrati po jednu sliku svake biljke koja nije korištena u treniranju modela. U glavnoj mapi Google Drivea ćemo napraviti mapu test unutar koje ćemo staviti mapu images. U mapu images stavljamo slike koje će nam služiti za testiranje s nazivima kao na slici 56.



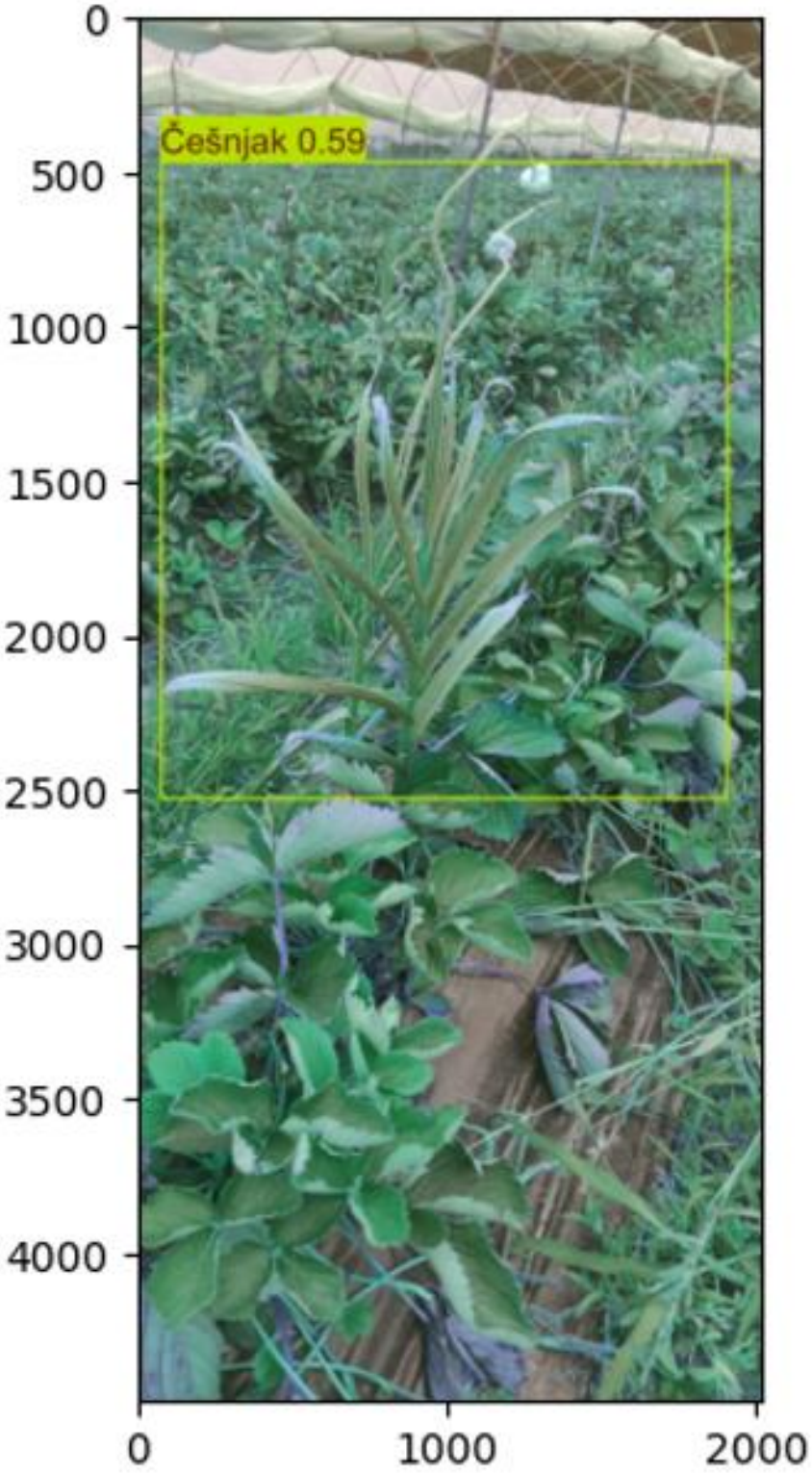
Slika 56. Nazivi slika za testiranje

Kada smo pripremili slike nad kojima ćemo izvršiti testiranja, u Google Colab upisujemo kod za provedbu testiranja, prikazan na slici 57.

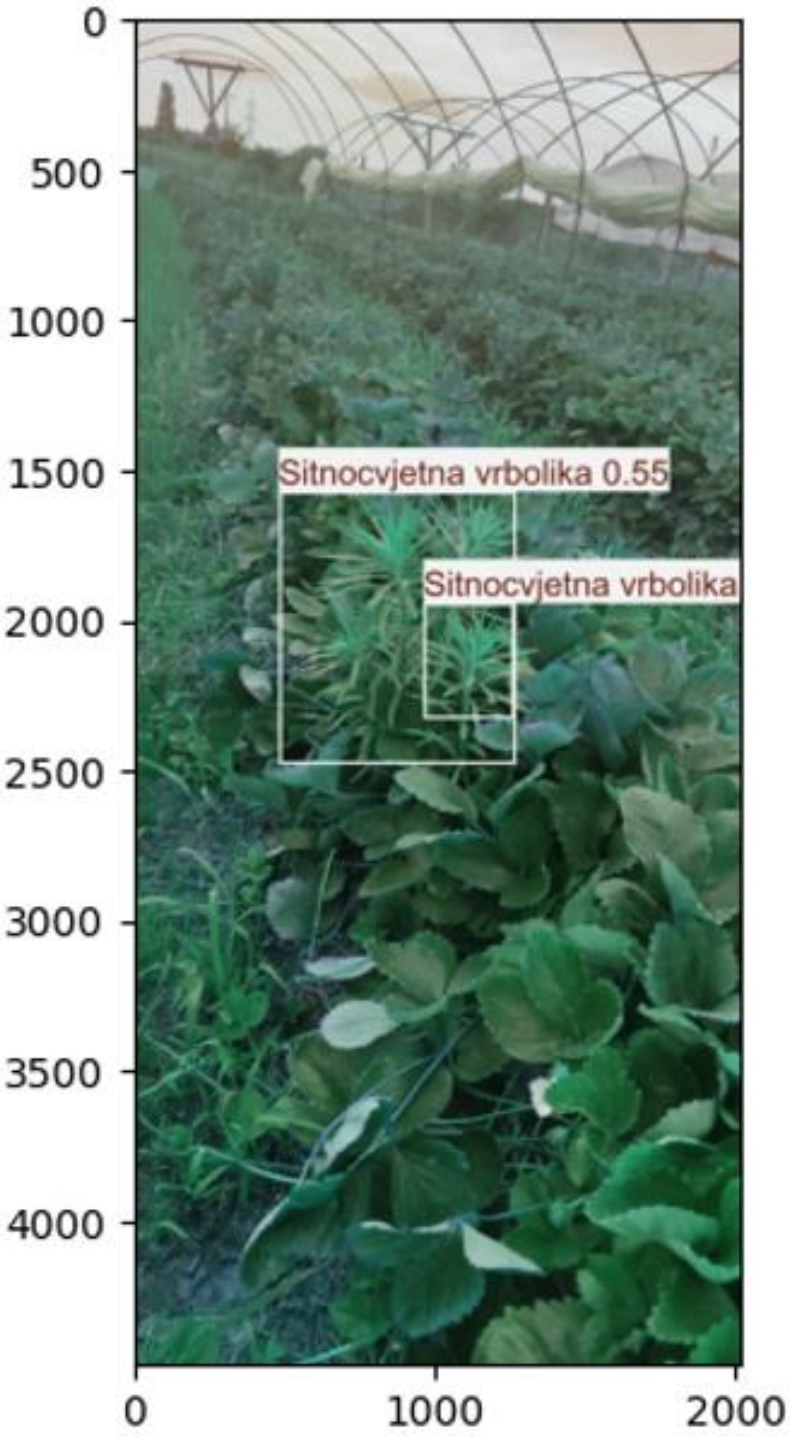
```
from ultralytics import YOLO
from matplotlib import pyplot as plt
from PIL import Image

model = YOLO("/content/gdrive/My Drive/Yolov8/runs/detect/train/weights/last.pt")

results = model.predict(source="/content/gdrive/My Drive/Yolov8/test/images/Test_češnjak.jpg", conf=0.4)

results_array = results[0].plot()
plt.figure(figsize=(6, 6))
plt.imshow(results_array)
```

Slika 57. Naredba za testiranje modela na slikama koje nisu bile u bazi slika za treniranje

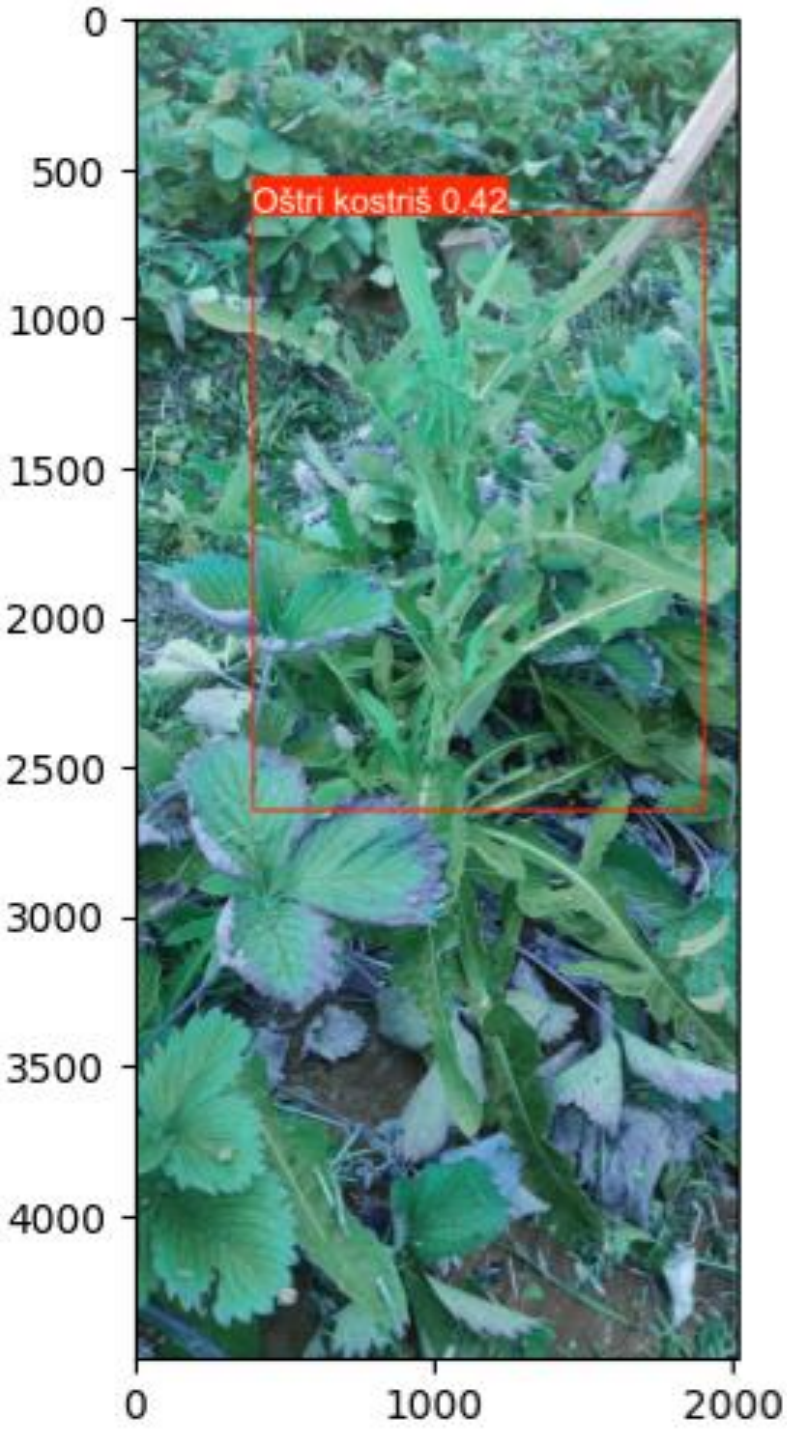
Prva tri reda koda služe za učitavanje biblioteka potrebnih za testiranje i ispisivanje rezultata testiranja. U četvrtom redu koda navodimo lokaciju modela dok u petom redu navodimo putanju do slike nad kojom ćemo vršiti testiranje, te upisujemo s kojom minimalnom sigurnošću će model prikazati detektirani objekt. Zadnja tri reda koda nam služe za ispis rezultata odnosno u ovom slučaju slike s okvirom. Nakon što upišemo kod i pokrenemo ga, ispod koda će nam se ispisati rezultati te slika testiranja. Bit će potrebno provesti naredbu 5 puta, za svaku sliku posebno.



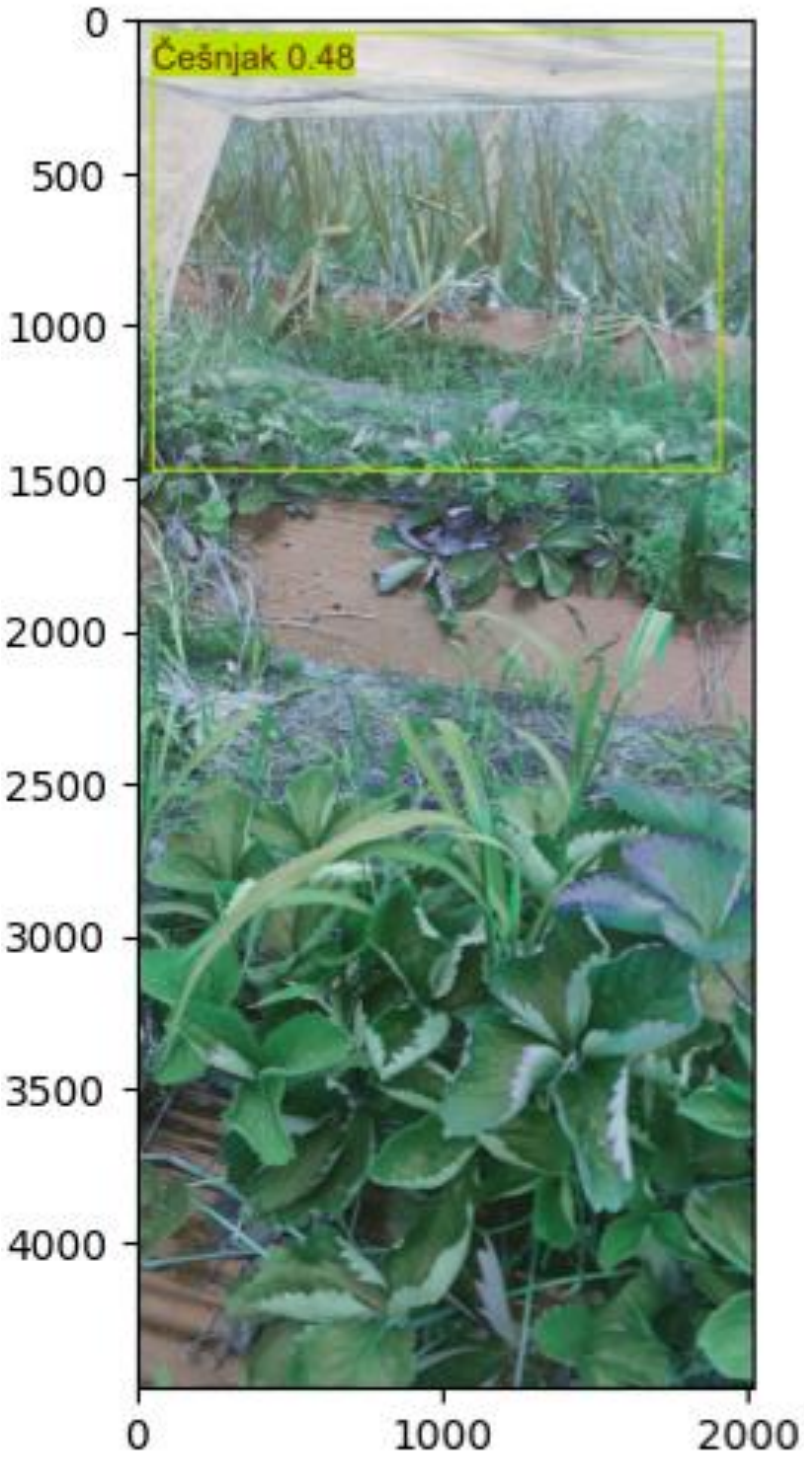
Slika 58. Testiranje modela na slici Test_češnjak.jpg



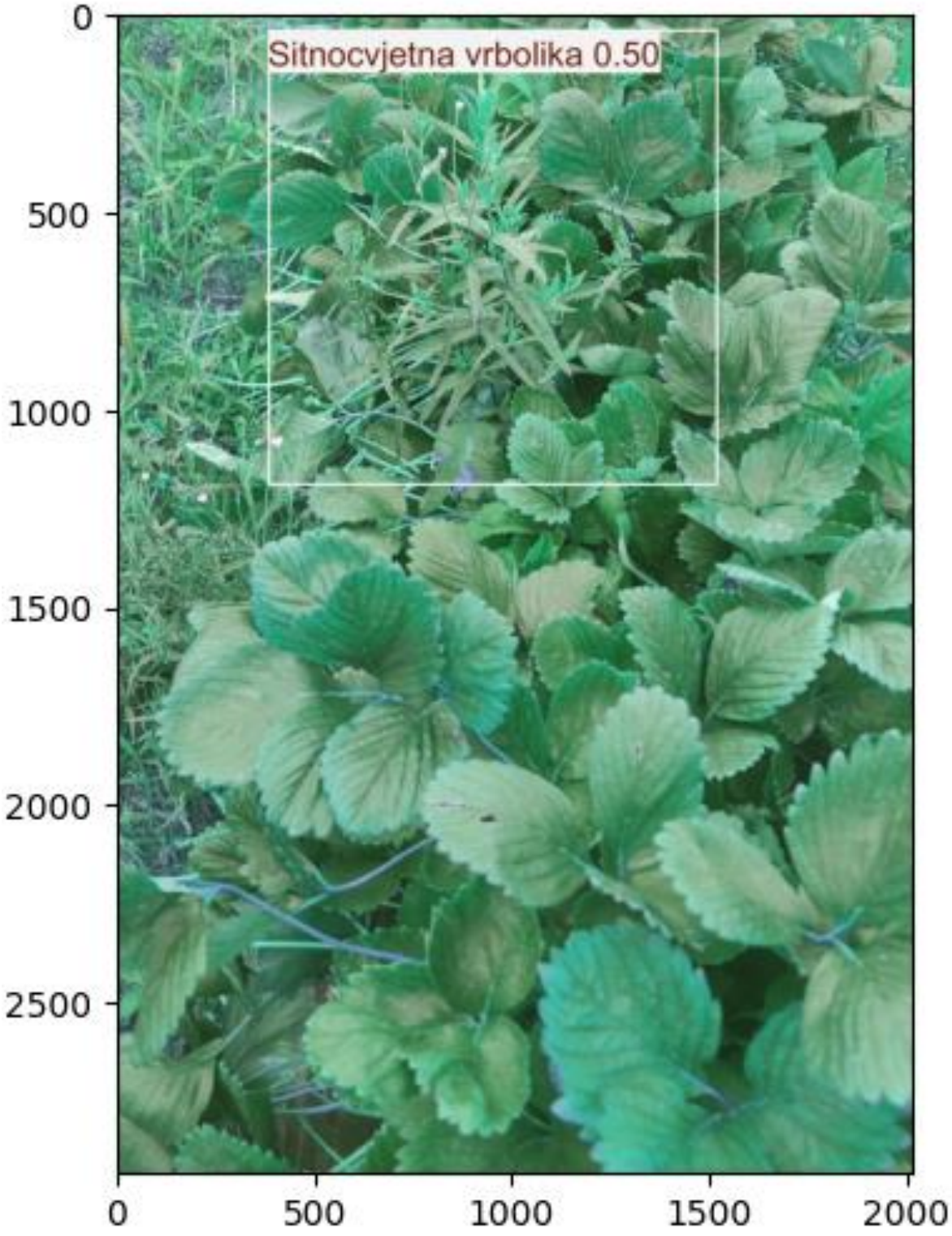
Slika 59. Testiranje modela na slici Test_kanadska_hudoljetnica.jpg



Slika 60. Testiranje modela na slici Test_oštri_kostriš.jpg



Slika 61. Testiranje modela na slici Test_pirika.jpg



Slika 62. Testiranje modela na slici Test_sitnocvjetna_vrbolika.jpg

4.4. Evaluacija modela i kritički osvrt

Kao što je vidljivo u rezultatima testiranja model ima mješovite rezultate koji su donekle i očekivani ako uzmemo u obzir kompleksnost slika. Od 5 mogućih biljka za detekciju model zadovoljavajuće detektira češnjak, oštri kostriš i sitnocvjetnu vrboliku. Dok piriku i kanadsku hudoljetnicu detektira s manjom točnošću. Jedan od glavnih razloga je i taj što smo za piriku i kanadsku hudoljetnicu imali manju bazu podataka odnosno imali smo manje slike nego za ostale biljke jer su se puno rjeđe pojavljivale.

U rezultatima testiranja možemo vidjeti kako je model na slici 59. krivo detektirao sitnocvjetnu vrboliku umjesto kanadske hudoljetnice, dok je na slici 61. točno detektirao češnjak koji se nalazio u pozadini i koji nije niti bio razlog fotografiranja već je razlog bila pirika koju model nije uspio detektirati.

Ako u budućnosti želimo poboljšati performanse modela, trebat ćemo povećati bazu slika. Trenutni broj slika od 275 je dovoljan za neko osnovno prepoznavanje biljaka, međutim ako tražimo visoku točnost i pouzdanost taj broj bi trebao biti puno veći. Druga mogućnost poboljšanja modela bi bio nastavak treniranja modela s istom bazom podataka. U tom slučaju mogli bismo očekivati poboljšanje ali teško da bi bilo značajno bolje od trenutnog modela. Postoji još jedna mogućnost poboljšanja koja je malo kompliciranija od prethodna dva slučaja. A to je da malo povećamo bazu slika bez pozadine, te napravimo novu bazu slika gdje bi bile samo pozadine. Tada bi uz pomoć Python koda mogli generirati slike koje bi bile slučajne kombinacije jedne ili više slike korova sa slikom pozadine. Neke od tih nasumično generiranih slika bi na oko izgledale umjetne i nerealne, ali bi za potrebe treninga ispunile ulogu. S ovim načinom bi s nekih dvjestotinjak slika korova i pedesetak slika pozadina, mogli generirati bazu podataka od desetak tisuća slika što bi trebalo biti dovoljno za treniranje modela veoma zadovoljavajućih performansa.

5. ZAKLJUČAK

Primjena umjetne inteligencije je raznolika, ona obuhvaća različite sektore i iz temelja mijenja način na koji pristupamo složenim problemima. Razvoj računalnog vida omogućio je strojevima interakciju sa svijetom na načine na koji su prije bili ne zamislivi. Uporabom različitih algoritama i tehnikama dubinskog učenja, sustavi za otkrivanje objekata mogu detektirati objekte na slikama i videima s nevjerojatnom preciznošću i brzinom.

U ovome radu je detaljno opisan teorijski dio koji je usko vezan uz računalni vid, te je prikazan proces stvaranja YOLOv8 modela za detekciju korova na plantaži jagoda. Opisano je sve što je potrebno za dobivanje modela iz baze slika prilagođenih za ovaj specifičan primjer. Za postizanje što boljih rezultata modela, najbitnije je skupiti dovoljno kvalitetnih slika nad kojim će se vršiti treniranje. To često zna biti dugačak i naporan posao međutim kvalitetna baza podataka je jedina garancija za dobar rad modela.

Pisanjem ovog rada stekao sam mnoga znanja vezana uz umjetnu inteligenciju, duboko učenje i računalni vid, a kako su ta područje sve više i više utkana u našu svakodnevicu, ne sumnjam da ću ta novo stečena znanja imati gdje u budućnosti primijeniti.

LITERATURA

[1] What is artificial intelligence (AI)? Everything you need to know, https://www.techtarget.com/searchenterpriseai/definition/AI-Artificial-Intelligence, pristupljeno: 27.6.2024.

[2] What's the Difference Between Artificial Intelligence, Machine Learning and Deep Learning?, https://blogs.nvidia.com/blog/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/, pristupljeno: 27.6.2024.

[3] What is machine learning (ML)?, https://www.ibm.com/topics/machine-learning, pristupljeno: 27.6.2024.

[4] Difference Between Artificial Intelligence vs Machine Learning vs Deep Learning, https://www.geeksforgeeks.org/difference-between-artificial-intelligence-vs-machine-learning-vs-deep-learning/, pristupljeno: 27.6.2024.

[5] Introduction to Deep Learning, https://www.geeksforgeeks.org/introduction-deep-learning/, pristupljeno: 27.6.2024.

[6] Konvolucija, http://struna.ihjj.hr/naziv/konvolucija/19228/, pristupljeno: 27.6.2024.

[7] Convolutional Neural Networks, Explained, https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939, pristupljeno: 27.6.2024.

[8] Convolutional Neural Networks (CNNs) and Layer Types, https://pyimagesearch.com/2021/05/14/convolutional-neural-networks-cnns-and-layer-types/, pristupljeno: 28.6.2024.

[9] series zero padding, https://rostore.onlinefactory2024outlet.com/category?name=series%20zero%20padding, pristupljeno: 28.6.2024.

[10] Convolutional Neural Networks: A Comprehensive Guide , https://medium.com/thedeephub/convolutional-neural-networks-a-comprehensive-guide-5cc0b5eae175, pristupljeno: 28.6.2024.

[11] Activation Functions in Neural Networks [12 Types & Use Cases], https://www.v7labs.com/blog/neural-networks-activation-functions, pristupljeno: 28.6.2024.

[12] CNN | Introduction to Pooling Layer, https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/, pristupljeno: 28.6.2024.

[13] Fully Connected Layer vs Convolutional Layer, https://www.geeksforgeeks.org/fully-connected-layer-vs-convolutional-layer/, pristupljeno: 28.6.2024.

[14] What are convolutional neural networks?, https://www.ibm.com/topics/convolutional-neural-networks, pristupljeno: 28.6.2024.

[15] Linear/Fully-Connected Layers User's Guide, https://docs.nvidia.com/deeplearning/performance/dl-performance-fully-connected/index.html, pristupljeno: 28.6.2024.

[16] Introduction to Convolution Neural Network, https://www.geeksforgeeks.org/introduction-convolution-neural-network/, pristupljeno: 28.6.2024.

[17] Deep Learning Algorithms, https://www.javatpoint.com/deep-learning-algorithms, pristupljeno: 28.6.2024.

[18] Computer Vision Tutorial, https://www.geeksforgeeks.org/computer-vision/, pristupljeno: 28.6.2024.

[19] Metamorphic Testing for Object Detection Systems, https://arxiv.org/abs/1912.12162, pristupljeno: 30.6.2024.

[20] Object Detection vs Object Recognition vs Image Segmentation, https://www.geeksforgeeks.org/object-detection-vs-object-recognition-vs-image-segmentation/?ref=header_search, pristupljeno: 30.6.2024.

[21] A guide to Two-stage Object Detection: R-CNN, FPN, Mask R-CNN, https://medium.com/codex/a-guide-to-two-stage-object-detection-r-cnn-fpn-mask-r-cnn-and-more-54c2e168438c, pristupljeno: 30.6.2024.

[22] Faster R-CNN | ML, https://www.geeksforgeeks.org/faster-r-cnn-ml/, pristupljeno: 30.6.2024.

[23] SSD: Single Shot MultiBox Detector, Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg, 2015.

[24] How single-shot detector (SSD) works?, https://developers.arcgis.com/python/guide/how-ssd-works/, pristupljeno: 30.6.2024.

[25] YOLO Algorithm: Real-Time Object Detection from A to Z, https://kili-technology.com/data-labeling/machine-learning/yolo-algorithm-real-time-object-detection-from-a-to-z, pristupljeno: 30.6.2024.

[26] YOLOv4: Optimal Speed and Accuracy of Object Detection, Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao, 2020.

[27] A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS, Juan Terven, Diana Cordova-Esparza, 2023.

[28] How to interpret a confusion matrix for a machine learning model, https://www.evidentlyai.com/classification-metrics/confusion-matrix, pristupljeno: 2.7.2024.

PRILOZI

I. Kod za 1. trening

```
from google.colab import drive

drive.mount('/content/gdrive')



ROOT_DIR = '/content/gdrive/My Drive/Yolov8'



!pip install ultralytics



import os

from ultralytics import YOLO

model = YOLO("yolov8n.yaml")

results = model.train(data=os.path.join(ROOT_DIR, "config.yaml"),
epochs=30)


!scp -r /content/runs '/content/gdrive/My Drive/Yolov8'
```

II. Kod za 2. trening

```
from google.colab import drive

drive.mount('/content/gdrive')



ROOT_DIR = '/content/gdrive/My Drive/Yolov8'



!pip install ultralytics
```

```
import os

from ultralytics import YOLO

model = YOLO("yolov8n.yaml")

results = model.train(data=os.path.join(ROOT_DIR, "config.yaml"),
epochs=100)


!scp -r /content/runs '/content/gdrive/My Drive/Yolov8'
```

III. Kod za 3. trening

```
from google.colab import drive

drive.mount('/content/gdrive')



ROOT_DIR = '/content/gdrive/My Drive/Yolov8'



!pip install ultralytics



from ultralytics import YOLO

model = YOLO('/content/gdrive/My
Drive/Yolov8/runs/detect/train/weights/last.pt')

train_results = model.train(data='/content/gdrive/My
Drive/Yolov8/config.yaml', epochs=40)



!scp -r /content/runs '/content/gdrive/My Drive/Yolov8'
```

IV. Kod za testiranje

```
from ultralytics import YOLO
from matplotlib import pyplot as plt
```

```
from PIL import Image

model = YOLO("/content/gdrive/My
Drive/Yolov8/runs/detect/train/weights/best.pt")

results = model.predict(source="/content/gdrive/My
Drive/Yolov8/test/images/Test_češnjak.jpg", conf=0.4)

results_array = results[0].plot()
plt.figure(figsize=(6, 6))
plt.imshow(results_array)
```