Izrada sustava za prepoznavanje bolesti pčelinjaka prisustvom nametnika Varoa

Ćurić, Tomislav

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

IZRADA SUSTAVA ZA PREPOZNAVANJE BOLESTI PČELINJAKA PRISUSTVOM NAMETNIKA VAROA

Diplomski rad

Tomislav Ćurić

Osijek, 2023.

Sadržaj

1. UVOD

Svako živo biće na planetu Zemlji ima određenu ulogu u svome životnom vijeku, neovisno o veličini i obliku života. Proučavanjem hranidbenog lanca jasno je prikazano da u prirodi organizmi uvijek ovise jedan o drugom. Jedan segment hranidbenog lanca je u iznimnoj opasnosti, a ujedno je i jedan od najvažnijih čimbenika istog. Radi se o kukcima, prvenstveno o medonosnim pčelama i njihovom globalnom izumiranju. Grinja varoa je trenutno jedan od glavnih razloga izumiranja pčela. Iako se istoimena grinja prirodno nalazi u većini košnica, ukoliko se ne kontrolira brojnost i stanje grinje unutar košnice, može doći do izumiranja cijele kolonije. Praćenje stanja kolonija obavlja se jednostavnim prebrojavanjem grinja no problem se pojavljuje kada pčelar u određenim godišnjim dobima mora nekoliko puta obavljati pregled u većem broju košnica što iziskuje puno vremena.

U ovom radu predstavljen je sustav za detekciju grinje varoa pomoću modela dubokog učenja. Nakon opsežnog istraživanja implementiran je sustav prepoznavanja grinje te nakon implementacije prikazano je testiranje i ocjena rezultata. Ovaj rad uključuje prikupljanje podataka iz slika podnica košnice na kojima se nalaze uginule grinje zajedno s raznim ostalim organskim ostacima koje u određenim primjerima sadrže slične značajke objekata od interesa. Nakon toga slijedi priprema podataka što podrazumijeva izmjenu odnosno prilagodbu podatka prema potrebama modela te označavanje kao posljednja priprema podataka za treniranje duboke neuronske mreže, odabir algoritma za zadani slučaj. Nakon prethodno izvršenih koraka potrebno je procijeniti kvalitetu modela kako bi se pravilno odredila valjanost te je u nastavku opisna implementacija sustava i testiranje istog.

Rad se sastoji od četiri cjeline. U drugom poglavlju rada opisana je morfologija grinje varoa i ukratko predstavljena ideja ovog rada. Nadalje spomenuta je problematika detektiranja sitnih objekata u području strojnog učenja te su napomenute jednostavne tehnike s ciljem poboljšanja otkrivanja grinje. U trećem poglavlju opisana je teorija rada, vrste strojnog učenja i način rada te su navedene metode vrednovanja modela otkrivanja objekata. U četvrtom poglavlju opisani su alati potrebni za implementaciju sustava za otkrivanje grinje te je objašnjen postupak implementacije modela, Internet i Android aplikacija uz ilustrativan prikaz implementacije sustava. U petom poglavlju prikazani su rezultati procjene modela, te ukratko komentirani. Navedeni su problemi implementiranih aplikacija te moguća poboljšanja svih koraka u izradi ovog sustava.

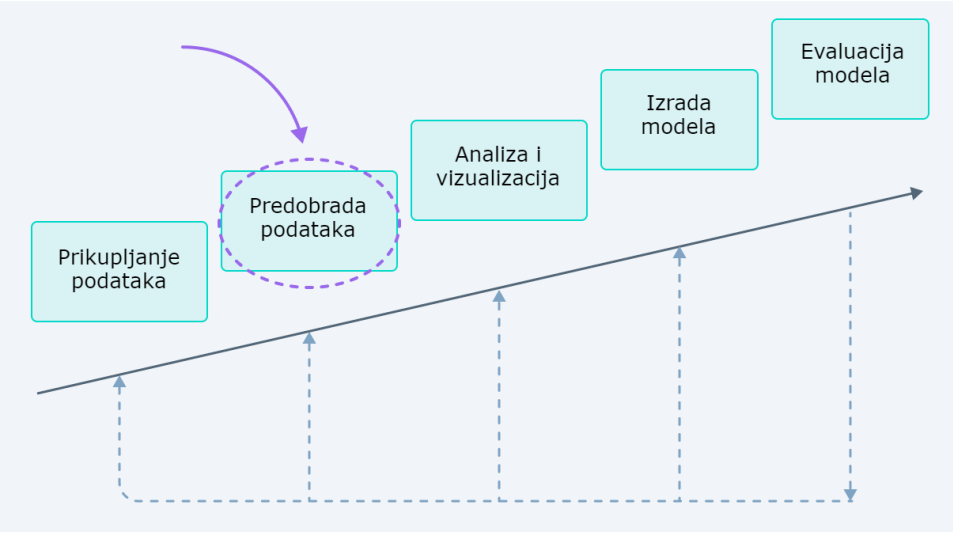
2. PROBLEMATIKA DETEKCIJE SIĆUŠNIH OBJEKATA

U ovom poglavlju predstavljen je problem otkrivanja malih objekata na slikama te su navedene i objašnjene pojedine tehnike s kojima je moguće pomoći modelu da što bolje i preciznije detektira željene objekte. Kako bi se pobliže prikazala problematika zadatka, potrebno je najprije razumjeti kako je veličina objekta u stvarnom svijetu zapravo relativan pojam u računalnom vidu. Naime, kada se u računalnom vidu govori o veličini objekta, govori se o reprezentaciji tog objekta kroz broj piksela unutar slike, odnosno što je broj piksela koji definiraju jedan objekt veća, to su detalji i značajke (engl. *features*) koje je moguće upotrijebiti za klasifikaciju i detekciju veće.

Tehnike koje su navedene u ovom poglavlju pomažu algoritmima detekcije i klasifikacije i nisu nužno unaprjeđenje samih algoritama. Iako slične implementacije ovih tehnika već postoje u raznim neuronskim mrežama i dubokom učenju (engl. *deep learning*) koji na određeni način manipuliraju ulaznim podacima odnosno slikama, postoje i druge tehnike koje se obično koriste u pred obradi (engl. *preprocessing*) podatkovnog skupa (engl. *dataset*) te imaju za cilj:

- povećati početni skup podataka ukoliko je malen
- ukoliko su podaci početnog skupa jako slični, dopunjavanjem unijeti raznolikosti
- normalizacija podataka

Navedeni koraci izvršavaju se prije izrade modela prikazano sljedećom slikom 2.1.



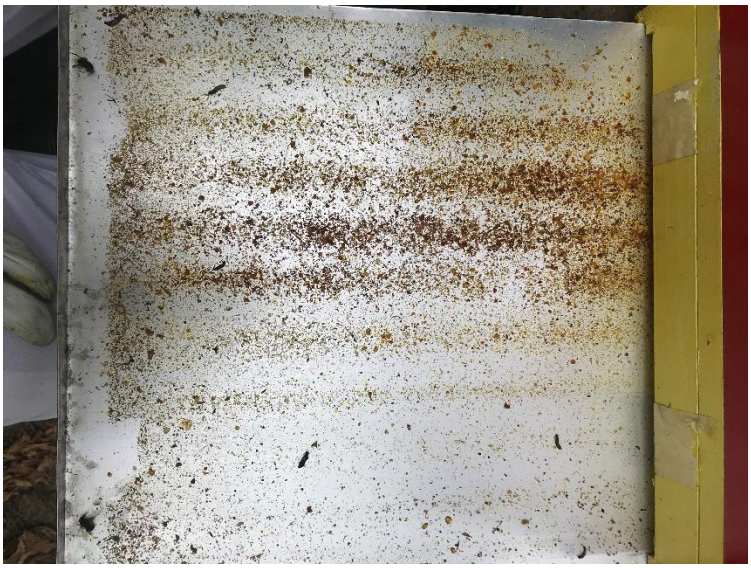
Slika 2.1 *Najvažniji koraci strojnog učenja*

2.1. Grinja Varoa

Prema [8], grinja varoa (*lat. varroa destructor*) su sitni, crveno-smeđi vanjski paraziti medonosnih pčela (Sl. 2.2). Iako se grinja varoa hrani i živi na odraslim pčelama, uglavnom se hrane i razmnožavaju na ličinkama i kukuljicama u leglu koji su u stadiju razvoja. Također uzrokuju malformacije i slabljenje kod pčela te prenose brojne viruse. Prema [8], kolonije sa slabom zaraznošću ovom grinjom općenito pokazuju vrlo malo simptoma, no kako se populacija grinja povećava, simptomi postaju očitiji. Jake infekcije grinjama varoa mogu se povećati za tri do četiri godine i uzrokovati razasuto leglo, osakaćene i gmižuće pčele, oslabljenu sposobnost leta, nižu stopu povratka u zajednicu nakon ishrane, skraćeni životni vijek i značajno smanjenu težinu pčela radilica. Simptomi kolonija koji se obično nazivaju sindrom parazitskih grinja uključuju nenormalan uzorak legla, udubljene i sažvakane poklopce i ličinke spuštene na dno ili sa strane ćelije što u konačnici uzrokuje smanjenje populacije medonosnih pčela, zamjenu pčelinjih matica i eventualni raspad i smrt zajednice.

Slika 2.2 *Primjeri pčela zaražene grinjom [8]*

Kako bi se pratila populacija grinja unutar košnice koriste se ljepljivi umetci. Pčelar takve umetke koristi na način da ih stavi u dno same košnice u nadi da će grinje koje se nalaze na pčelama i okolo njih, ispasti na ljepljivi umetak (Sl. 2.3).

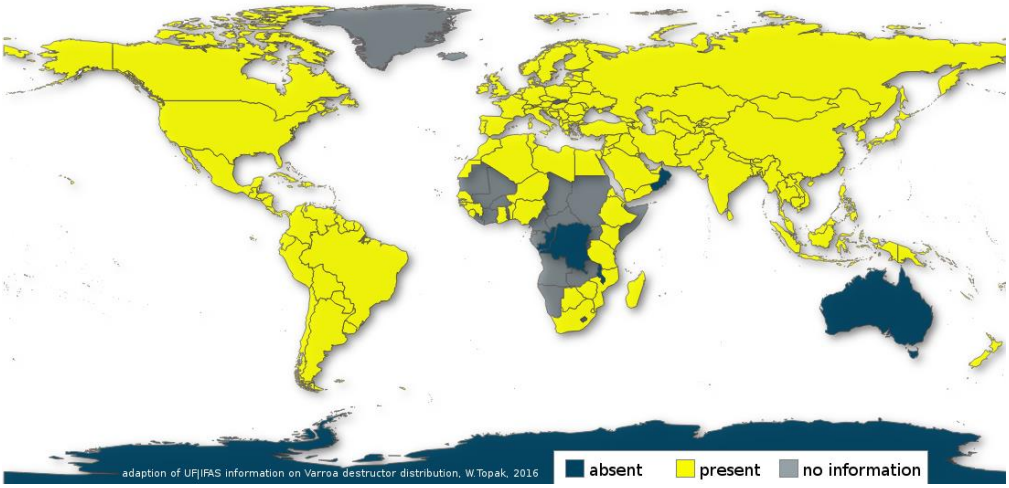


Slika 2.3 *Prikaz umetka na dnu košnice*

Skup slika koje izgledaju kao prethodno navedena slika 2.3 temelj su ovog rada te su iskorištene za učenje modela strojnog učenja s ciljem otkrivanja grinje varoa. Podatkovni skup je osigurao sumentor, a izvor su pčelinjaci i pčelinje zajednice profesora i nastavnika Zavoda za pčelarstvo i zoologiju Poljoprivrednog fakulteta u Osijeku.

2.2. Idejno rješenje rada

Kako je navedeno u uvodu ovog rada, grinja varoa jedna je od glavnih uzročnika izumiranja medonosnih pčela. Zadatak pčelara je kontrolirati populaciju varoa grinja unutar pčelinjaka dok je idejni zadatak ovog rada olakšati pčelaru kontroliranje istih sa bržim i jednostavnijim alatom. Kako bi se ozbiljnost ovog problema dodatno naglasila potrebno je obratiti pažnju na globalnu distribuciju grinje varoa vidljive na slici 2.4. Žuta boja na slici predstavlja prisutnost grinje u navedenim državama, plava boja predstavlja odsustvo grinje dok siva boja označava manjak informacija o prisutnosti grinja.



Slika 2.4 *Prikaz raširenosti grinje varoa* [10]

2.3. Načela i tehnike poboljšanja prikupljenih podataka

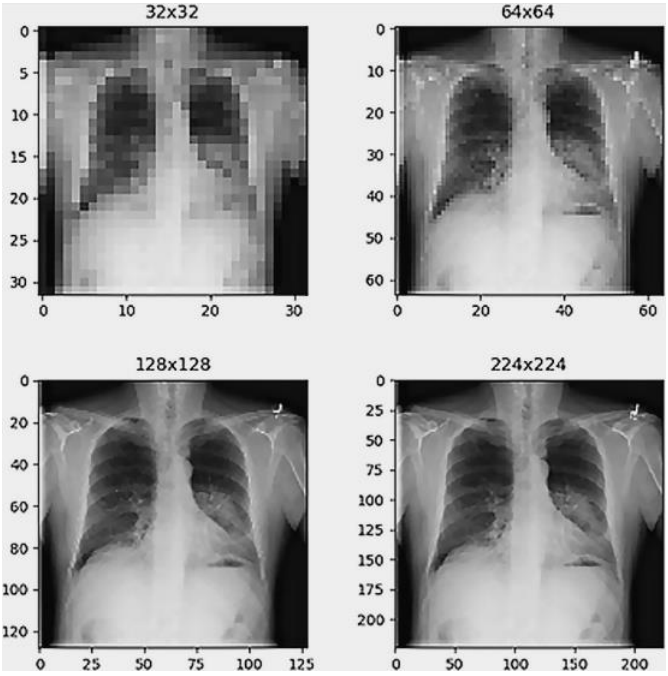
U ovom potpoglavlju nabrojane su i ukratko objašnjene neke od strategija koje su korištene u rješavanju problema detektiranja malih objekata na slici. Strojno učenje ulazi u sve više dijelova svakodnevnog života. Od ponuđenih oglasa koji se razlikuju od korisnika do korisnika, filmske preporuke na temelju prošlih pretraživanja, autonomni automobili i slično. Gotovo svi moderni automatizirani strojevi „vide“ svijet, no za razliku od ljudi koji putem različitih osjetila stvaraju sliku svijeta, strojevi koriste brojeve. Njihova je zadaća zasebno detektirati i klasificirati svaki predmet kako bi ga „vidjeli i prepoznali“ kao i ljudi. Iako većina modernih modela detekcije prilično dobro izvršava zadatak detekcije relativno velikih objekata kao što su primjerice ljudi, automobili, životinje, kuće i slično, manji objekti im i dalje predstavljaju veliki problem. Primjer takvih objekata su zračne slike u kojima dron ili avion fotografira određeno područje interesa, fotografije mikroskopskih snimki u medicini, kamere koje prate kakvoću na pokretnim trakama u raznim pogonskim postrojenjima. Prema [4], tablica 2.1 prikazuje velika odstupanja prosječne preciznosti (engl. *mAP-mean average precision*) algoritma nad detekcijom malih i velikih objekata. Mjera *mAP* jedna je od najvažnijih kriterija mjere valjanosti modela u području detekcije objekata. Usporedbom stupaca $AP_{S\text{-}M\text{-}L}$ primjećuje se velika razlika preciznosti detekcije malih objekata (AP_S) nad srednjim i velikim ($AP_{M\text{-}L}$) objektima. Također je moguće primijetit kako određeni algoritmi bolje detektiraju male objekte od drugih, pa tako *YOLO* algoritam sa *Darknet* arhitekturom ima $AP_S = 5.0$ dok *SOD-MTGAN* sa arhitekturom *ResNet-101* ima $AP_S = 24.7$, to je gotovo pet puta veća razlika što samo potvrđuje problematiku detekcije malih objekata.

Tablica 2.1 *Performanse modela nad objektima* [4]

Algoritmi	Arhitektura	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
SSD512	VGG16	26.8	46.5	27.8	9.0	28.9	41.9
YOLO9000	Darknet -19	21.6	44.0	19.2	5.0	22.4	35.5
Faster RCNN	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
FPN	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
G-RMI	Inception-ResNet-v2	34.7	55.5	36.7	13.5	38.1	52.0
TDM	Inception-ResNet-v2-TDM	36.8	57.7	39.7	16.2	39.8	52.1
Mask RCNN	ResNeXt-101-FPN	39.8	62.3	43.4	22.1	43.2	51.2
SOD-MTGAN	ResNet-101	41.4	63.2	45.4	24.7	44.2	52.6

2.3.1. Povećanje razlučivosti zapisivanja slika

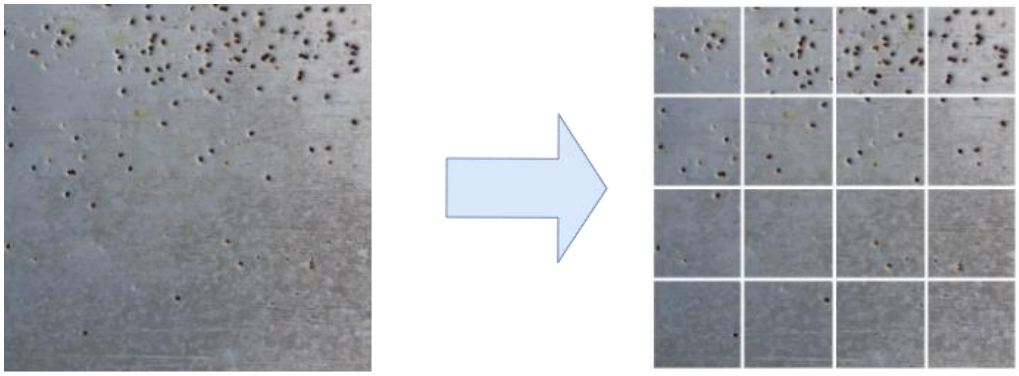
Jedno od rješenja odnosno unaprjeđenja detekcije manjih objekata je uporaba visoko razlučivih fotografija. Vrlo mali objekti mogu se nalaziti u svega nekoliko piksela unutar graničnog okvira (engl. *bounding box*) što znači da je važno povećati razlučivost fotografija kako bi se povećao broj piksela koji opisuju taj objekt da algoritam neuronske mreže lakše i efikasnije stvori mapu značajki. Ovakve metode pomažu neuronskoj mreži da dobije više značajki iz slike te da se u konačnici model bolje nauči. Prema slici 2.5 koja prikazuje usporedbu radiografije prsnog koša pacijenta pri različitim razlučivostima slike primjećuje se crna masa u svim fotografijama dok fotografije veće razlučivosti (donji red) imaju vidljivo poboljšanu jasnoću.



Slika 2.5 *Prikaz važnosti veće razlučivosti slike* [5]

2.3.2. Segmentiranje fotografije

U prethodnom potpoglavlju spomenuta je uporaba fotografija visoke razlučivosti, no treniranje modela sa ulaznim podacima visoke razlučivosti iziskuje ogromne količine resursa stoga je potrebno obaviti dodatnu obradu nad podacima. Takva operacija zove se segmentiranje fotografije (engl. *image tiling*) u kojoj se izvorna fotografija dijeli na više manjih fotografija (Sl. 2.6). Ovom tehnikom osim što se smanjuje opterećenje na resurse sustava, također se poboljšava i brzina treniranja, poboljšavaju se rezultati treniranja i kasnijeg detektiranja iz razloga što se u ovom koraku povećava podatkovni skup (više manjih inačica originalne fotografije).



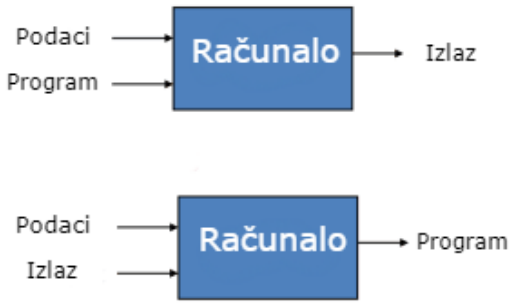
Slika 2.6 *Prikaz podjele slike na manje dijelove*

Razlog zašto se ovim korakom povećava preciznost predikcije modela je način rada konvolucijske mreže, a to je da se kroz duboke slojeve (engl. *deep layers*) ulazna slika konstantno mijenja, odnosno veličina ulazne slike se umanjuje te se iz nje izvlače značajke. Ukoliko je slika velike razlučivosti i na njoj se nalaze manji objekti od značaja, velika je vjerojatnost da će se značajke manjih objekata izgubiti propagacijom kroz konvolucijsku mrežu. Ukoliko su manji objekti na relativno maloj slici, iako se slika umanjuje, neće doći do većeg gubitka značajki objekta. Primjerice, ukoliko je slika veličine 1024*1024 piksela i u njoj se nalazi objekt od interesa veličine svega 20*30 piksela, taj objekt je gotovo neprimjetan pošto zauzima svega 0.0286% ukupne površine slike, dok primjerice u slici veličine 256*256 isti taj objekta zauzima 0,45% što je značajna razlika kada se govori o detekciji malih objekata. Također, na slici 2.6 vidljivo je da se u prvom redu koja sadrži četiri manje sličice nalazi puno manjih objekata, u prosijeku oko 20, što znači da je ukupna pokrivenost objekata u ovom slučaju oko 10% što je značajnija razlika u odnosu na početnih 0.0286%. Nakon segmentacije slike obično se izvodi operacija filtriranja objekata (engl. *filter null)* kojom se uklanja slika iz podatkovnog skupa na kojima se ne nalaze označeni objekti.

3. UMJETNA INTELIGENCIJA I STROJNO UČENJE

3.1. Osvrt na strojno učenje

Umjetna inteligencija (engl. *artificial intelligence*) i strojno učenje (engl. *machine learning*) dio su računalne znanosti koji su međusobno povezani. Definira se kao uporaba računala ili strojeva koji oponašaju sposobnosti ljudskog uma poput rješavanja problema i donošenje odluka, dok se strojno učenje svrstava kao podskup umjetne inteligencije koje se usredotočuje na uporabu samoučećih (engl. *self-learning*) algoritama koji izvlače značajke iz podataka kako bi predvidjeli ishode ili donijeli odluku. Ti podaci mogu biti dobiveni iz stvarnog svijeta ili umjetno stvoreni. Prema [3], cilj korištenja strojnog učenja je omogućiti računalima da nauče samostalno. Algoritam strojnog učenja omogućuje prepoznavanje uzoraka u promatranim podacima, izgradnju modela koji objašnjavaju svijet i otkrivaju stvari bez upotrebe unaprijed programiranih pravila i modela kao što je moguće vidjeti na slici 3.1. izrađenoj prema izvoru [6], gdje gornji dijagram prikazuje klasično programiranje u kojem se podaci obrađuju unutar programa kako bi se dobio izlaz, dok se kod strojnog učenja program dobiva putem izlaza i podataka nad kojim se obavljaju razne operacije. Takav program može se kasnije upotrijebiti i u klasičnom programiranju.



Slika 3.1 *Usporedba klasičnog programiranja i strojnog učenja [6]*

Razlog zašto su ove tehnologije sve učestalije i popularnije nije samo pristupačnija računalna obradbena snaga, već želja čovjeka za automatizacijom sve više problema ili poslova s kojima se ljudi susreću. Većina takvih poslova uključuje ponavljajući rad kojeg je moguće automatizirani kao npr. roboti koji sortiraju poštu, robotske ruke kojima se izrađuju automobili na pokretnim trakama i slično.

Također je važna i kontinuirana nadogradnja grafičkih procesnih jedinica (engl. *GPU - graphics processing unit*) koje su izrazito bitne u ovoj grani računalne znanosti. Prednost grafičkih kartica

nad procesorskim jedinicama (engl. *CPU - central processing unit*) je razlika u načinu obrade podataka i broju jezgri koje *GPU* ima na pretek. *GPU* je dizajniran za brze izračune i rad sa matricama za prikaz slike. Upravo te karakteristike moguće je iskoristiti za rad u strojnom učenju. Zbog raznih zadataka koje je moguće riješiti pomoću strojnog učenja, osmišljeno je nekoliko pristupa toj problematici.

3.2. Vrste strojnog učenja

Postoji nekoliko pristupa učenja model te svaki ima svoje prednosti i nedostatke. Glavna razlika je označavanje podataka (engl. *labeling*) gdje je u jednom potrebno označiti sve podatke ulazne i izlazne, dok u drugom ne. U ovom radu obrađene su dvije glave tehnike, a to su nadzirano učenje (engl. *supervised learning*) i nenadzirano učenje (engl. *unsupervised learning*).

3.2.1. Nadzirano učenje

Prema [2], kao prva grana strojnog učenja, nadzirano učenje usredotočeno je na obrasce učenja kroz povezivanje odnosa između varijabli i poznatih ishoda te rad s označenim skupovima podataka. Nadzirano učenje radi na način da stroju daje uzorke podataka s različitim značajkama predstavljenim kao „x“ i ispravnim izlaznim vrijednostima podataka predstavljenim kao „y“. Algoritam zatim interpretira obrasce koji postoje u podacima i stvara model koji može reproducirati ista temeljna pravila s novim podacima. Ovakav način učenja modela je najučestaliji jer obično daje najbolje rezultate, a nedostaci su kompleksnost algoritma i vrijeme trajanja učenja modela.

Postoje dvije najčešće primjene nadziranog učenja:

- klasifikacija – predviđanje diskretnih vrijednosti na izlazu
- regresija – predviđanje neprekidnih vrijednosti koje nije moguće klasificirati, vjerojatnost

Primjeri važnijih algoritama:

- *K-near neighbors (kNN)*
- *Linear regression*
- *Neural networks*
- *Support vector machines*
- *Logistic regression*
- *Decision trees*
- *Random forests*

3.2.2. Nenadzirano učenje

Prema [2], u slučaju učenja bez nadzora, nisu označene sve varijable i obrasci podataka. Umjesto toga, stroj mora samostalno otkriti skrivene uzorke i stvoriti oznake korištenjem algoritama učenja bez nadzora. Algoritam grupiranja podataka *k-means* popularan je primjer učenja bez nadzora. Ovaj jednostavan algoritam grupira podatkovne točke za koje se utvrdi da imaju slične značajke.

Primjene nenadziranog učenja:

- grupiranje (engl. *clustering)* – pronalazak sličnosti unutar neoznačenog skupa i svrstavanje podataka u grupe sličnih značajki
- otkrivanje anomalija
- rudarenje asocijacija
- smanjenje dimenzija

Nenadzirano učenje je manje kompleksno i izvodi se u stvarnom vremenu, no značajno je nepreciznije od nadziranog učenja.

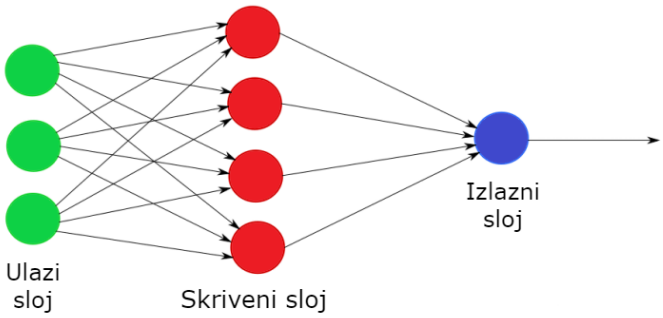
3.3. Neuronske mreže

Neuronske mreže formiraju temelje dubokog učenja, podskup su strojnog učenja u kojem su algoritmi inspirirani strukturom ljudskog mozga. Neuronske mreže kao ulaz primaju podatke, iterativno se uče da prepoznaju uzorke unutar podataka te na kraju rade predviđanje nad sličnim podacima. Neuronska mreža sastoji se od dvije glavne podjele, a to su slojevi i neuroni.

3.3.1. Slojevi neuronske mreže

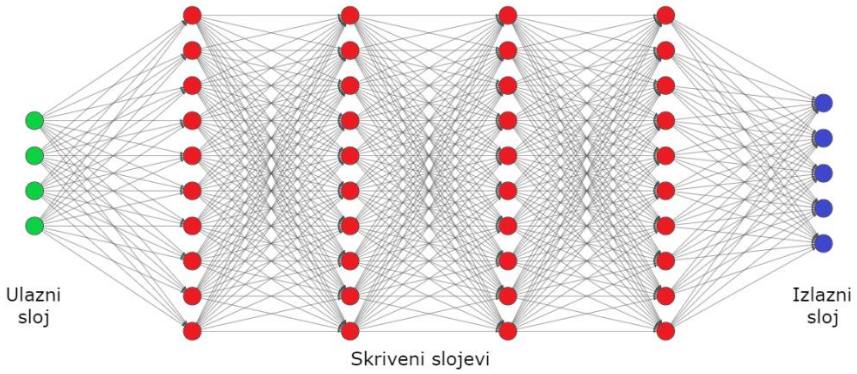
Neuronska mreža sastoji se od vertikalno postavljenih komponenti nazvanih slojevi kao što je vidljivo na slici 3.2. Postoje tri tipa slojeva:

- ulazni sloj – prvi sloj na ulasku u mrežu, a uloga je prihvaćanje podataka i propuštanje istih kroz nastavak mreže
- skriveni sloj – drugi sloj u kojem se može nalaziti jedna ili više skrivenih slojeva. Ovaj sloj je zaslužan za performanse i složenost neuronskih mreža. Obavljaju više funkcija u isto vrijeme kao npr. transformacija podataka, automatsko stvaranje značajki i sl.
- izlazni sloj – zadnji sloj u mreži, u njemu je sadržano rješenje problema



Slika 3.2 *Struktura slojeva neuronske mreže*

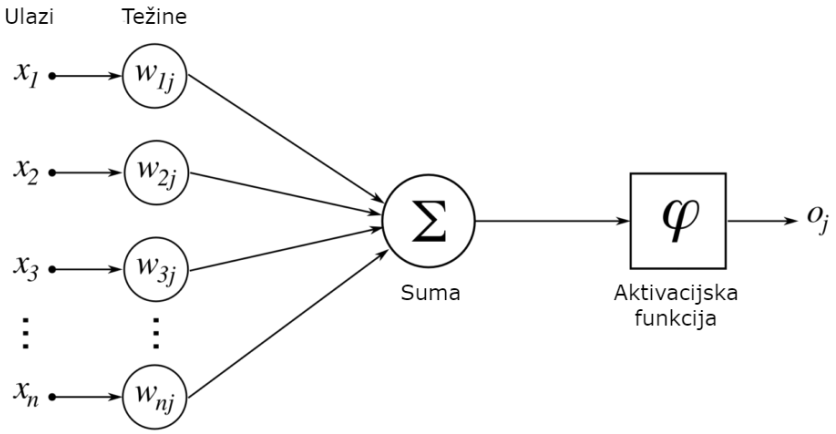
Osim jednostavnih neuronskih mreža koje se sastoje od samo jednog skrivenog sloja, postoje i kompliciranije koje se nazivaju duboke neuronske mreže (engl. *DNN - deep neural network*) i sastoje se od minimalno dva ili više skrivenih slojeva (Sl. 3.3).



Slika 3.3 *Struktura duboke neuronske mreže*

3.3.2. Neuron

Prema [2], umjetne neuronske mreže formiraju međusobno povezani neuroni, koji se nazivaju i čvorovi koji međusobno djeluju putem aksona, koji se nazivaju rubovi. U neuronskoj mreži, čvorovi su složeni u slojeve i obično počinju sa širokom bazom. Prvi sloj sastoji se od neobrađenih podataka kao što su numeričke vrijednosti, tekst, slike ili zvuk, koji su podijeljeni u čvorove. Svaki čvor zatim šalje informacije sljedećem sloju čvorova kroz rubove mreže (Sl. 3.4).



Slika 3.4 *Model neurona*

Prema [2], svaki rub ima numeričku težinu koja se može mijenjati i formulirati na temelju iskustva. Ako zbroj spojenih rubova zadovoljava postavljeni prag, poznat kao aktivacijska funkcija, aktivirat će se neuron na sljedećem sloju. Međutim, ako zbroj spojenih rubova ne dosegne postavljeni prag, aktivacija se neće dogoditi. Težine duž svakog ruba su jedinstvene kako bi se osiguralo da se čvorovi aktiviraju drugačije i da ne vraćaju isti rezultat.

3.4. Duboko učenje i konvolucijske mreže

Duboko učenje je grana strojnog učenja (Sl. 3.5) koja je u potpunosti temeljena na umjetnim neuronskim mrežama, gdje neuronske mreže oponašaju rad ljudskog mozga ili preciznije, duboko učenje je jedan od načina oponašanja rada ljudskog mozga. Također se definira kao posebna vrsta učenja koja postiže veliku moć i fleksibilnost učenjem predstavljanja svijeta kao ugniježđene hijerarhije koncepata, pri čemu je svaki koncept definiran u odnosu na jednostavnije koncepte, a apstraktniji prikazi izračunati u terminima koji su manje apstraktni. U dubokom učenju nije potrebno eksplicitno programirati svaki zasebni dio neuronske mreže.

Slika 3.5 *Hijerarhijski odnos umjetne inteligencije i pripadajućih podskupova*

Ogroman potencijal dubokog učenja u zadnjem desetljeću otkriven je kroz razna poboljšanja u računalnom svijetu. Jedan od bitnijih faktora je količina podataka koje se svakodnevno stvara. Prema [7], količina stvorenih podataka raste eksponencijalno te se procjenjuje da količina podataka na dnevnoj bazi doseže i do 2.5 milijuna terabajta ($2.5*10^6$TB) što savršeno odgovara algoritmima dubokog učenja jer takvi algoritmi zahtijevaju uporabu, ukoliko je moguće, što većeg podatkovnog skupa jer samostalno uče iz podataka kako je i ranije objašnjeno. Razna poboljšanja u proizvodnji računalne opreme poput novih arhitektura, procesnih jedinica ili integriranih krugova (engl. *integrated circuits*) poput *TPU* (engl. *tensor processing unit*) i služi prvenstveno kao zamjena za do sada korištene resurse poput *CPU* i *GPU* jer imaju sposobnost bržeg obavljanja operacija korištenjem manje resursa u odnosu na *GPU*. Također je potrebno napomenuti da su iznad nabrojani razlozi utjecali i na daljnji napredak u razvitku novih, kompleksnijih i boljih algoritama dubokog učenja.

Neki od primjera korištenja algoritama dubokog učenja su predviđanje vremenskih prilika i klimatskih promjena na temelju dosadašnjih podataka, virtualni asistenti poput *Cortana, Siri* i *Alexa*, autonomna vožnja, odnosno sposobnost automobila da se pomoću detekcije objekata kreću samostalno, prepoznavanje lica u području sigurnosti i autentifikacije, automatski prijevod s jednog jezika na drugi i slično. Prema [2], u tablici 3.1 prikazane su skupine problema i mogući odabir algoritma za rješavanje istog.

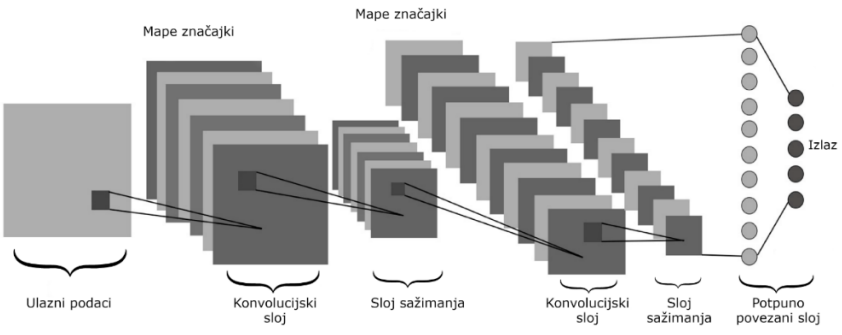
Tablica 3.1 *Zadatci i rješenja odgovarajućim algoritmima*

	Recurrent network	*Recursive neural tensor network*	*Deep belief network*	*Convolution network*	*MLP*
Obrada teksta	✓	✓		✓	
Raspoznavanje slika			✓	✓	
Prepoznavanje objekta		✓		✓	
Prepoznavanje govora	✓				
Analiza vremenskih nizova	✓				
Klasifikacija			✓	✓	✓

3.4.1. Konvolucijske neuronske mreže

Prema [1], konvolucijske mreže, poznatije kao konvolucijske neuronske mreže (engl. *CNN - convolutional neural network*) posebne su vrste neuronskih mreža za obradu podataka koje imaju poznatu topologiju nalik mreži. Primjeri uključuju podatke o vremenskoj seriji, koji se mogu zamisliti kao 1D mreža koja uzima uzorke u redovitim vremenskim intervalima, i slikovni podaci koji se mogu zamisliti kao 2D mreža piksela. Naziv konvolucijska mreža označava da mreža koristi matematičku operaciju konvolucije. Konvolucija je posebna vrsta linearne operacije. Konvolucijske mreže moguće je jednostavno opisati kao neuronske mreže koje koriste konvoluciju umjesto općeg množenja matrica u barem jednom od konvolucijskih slojeva.

Kao što je vidljivo na slici 3.6, konvolucijska mreža se sastoji od nekoliko dijelova, točnije dva glavna, a to su konvolucijski slojevi (engl. *convolution layers*) i slojevi sažimanja (engl. *pooling layers*) te na kraju od potpuno povezanog sloja (engl. *fully connected layer*).



Slika 3.6 *Prikaz strukture, slojeva i operacija CNN*

3.4.2. Konvolucijski sloj

Konvolucijski sloj je sloj konvolucijske mreže u kojem se nad ulaznim podacima obavlja operacija konvolucije. Konvoluciju je moguće definirati kao matematičku operaciju nad dvije funkcije koja za rezultat daje treću funkciju i opisuje kako se oblik jedne modificira drugom funkcijom. Matematička operacija konvolucije označava se simbolom $(*)$ i definirana je sljedećim izrazom (3-1).

$$h(t) = (f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{3-1}$$

gdje je f i g predstavljaju ulazne funkcije, a h novonastalu funkciju dobivenu konvolucijom funkcija f i g.

Pošto se u računalnom vidu koriste isključivo normalizirane i diskretne vrijednosti, konvoluciju moguće je prikazati sljedećim izrazom (3-2).

$$h(t) = (f * g)(t) = \sum_{n=-\infty}^{\infty} f(n)g(t-n) \tag{3-2}$$

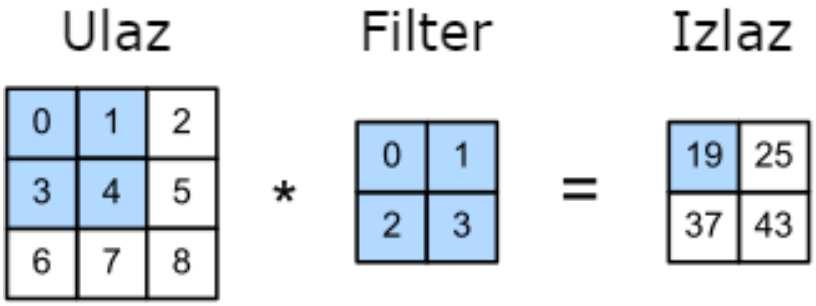
U strojnom učenju se radi sa višedimenzionalnim podacima poput slike stoga je izraz (3-2) potrebno je prilagoditi. Izraz (3-3) prikazuje konvoluciju između ulaza (slike) i filtra odnosno jezgre (engl. *kernel*).

$$H(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n) \tag{3-3}$$

gdje I predstavlja dvodimenzionalnu sliku kao ulaz dok K predstavlja dvodimenzionalni filtar.

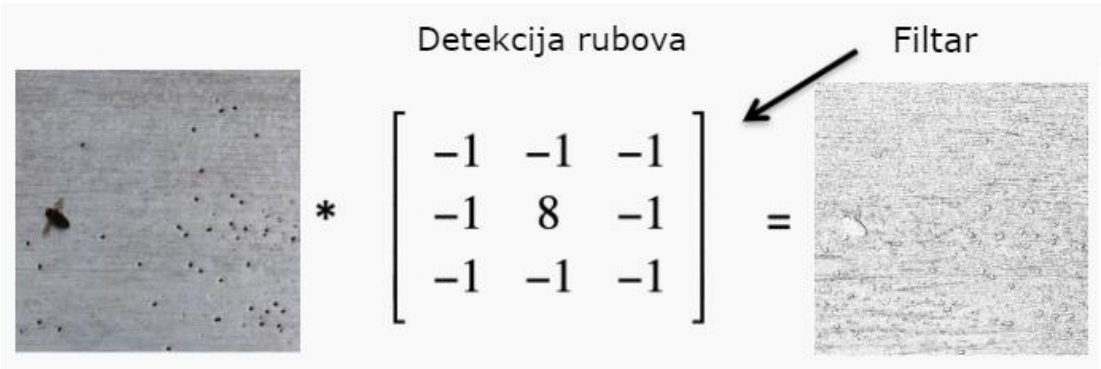
Prema [9], u kontekstu konvolucijske neuronske mreže, konvolucija je linearna operacija koja uključuje množenje skupa težina sa ulazom, slično poput klasične neuronske mreže. Obzirom da je tehnika dizajnirana za dvodimenzionalni unos, množenje se svodi između niza ulaznih podataka i dvodimenzionalnog niza težina (engl. *weights*). Na sljedećoj slici 3.7 prikazan je postupak izvršavanja matematičke operacije konvolucije nad ulaznim podatkom pomoću filtra u kojem se element ulaznog podatka množi sa elementom filtra identičnog indeksa, rezultati umnožaka se

zbrajaju i vrijednost se upisuje u novu matricu. Nakon toga, filtar se pomiče za jedan korak udesno ili u novi red ukoliko je dosegnut kraj trenutnog reda matrice i operacija se ponavlja. Operacija konvolucije se ponavlja sve dok filtar ne obradi svaki piksel ulaznog podatka.



Slika 3.7 *Operacija konvolucije*

Na slici 3.8 prikazan je primjer korištenja jednog od filtra (*laplacian*) u konvolucijskim mrežama. U ovom slučaju, prikazana je jezgra pomoću koje konvolucijska mreža detektira rubove unutar slike. Jezgre ovog tipa poput detekcije rubova, krugova, tekstura i sličnih jednostavnijih filtara, obično se nalaze u prvim slojevima konvolucijske mreže dok se u dubljim slojevima mreže koriste nešto kompleksniji sve u cilju otkrivanja uzoraka (engl. *pattern*). Izlazi filtra nazivaju se mape značajki (engl. *feature maps*) čija je uloga generalizirati uzorke.



Slika 3.8 *Primjer korištenja filtra unutar konvolucijskog sloja*

3.4.3. Aktivacijske funkcije

Sljedeći sloj konvolucijske mreže je sloj sažimanja (engl. *pooling layer*), no najprije je potrebno opisati operaciju koja prethodi sloju sažimanja, a to su **aktivacijske funkcije**. Uloga aktivacijskih funkcija je uvođenje nelinearnosti u neuronsku mrežu. Iako je neke probleme moguće riješiti pomoću obične linearne funkcije, većina kompleksnih problema zahtjeva korištenje nelinearnih funkcija. Ukoliko je problem jednostavan najčešće duboki slojevi nisu niti potrebni. Ovisno o vrsti

problema postoje i odgovarajuće aktivacijske funkcije koje je moguće primijeniti kako bi se neuronskoj mreži omogućilo preciznije i brže učenje iz početnog skupa podataka, te u konačnici bolje predviđa.

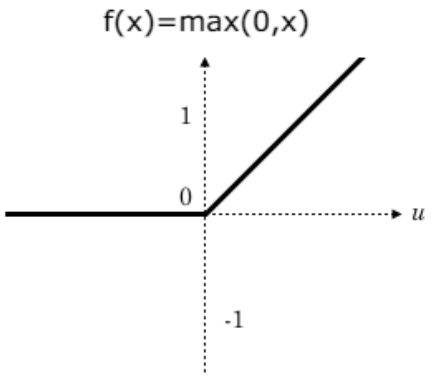
Osvrtom na sliku 3.4. vidljivo je da na svakom izlazu neurona je trenutna vrijednost linearna odnosno suma umnožaka ulaznih signala i težinskih vrijednosti (engl. *weights*). Nakon toga slijedi pretvaranje trenutne linearne funkcije u nelinearnu kako se bi problem bolje opisao. Ovaj korak izvodi se između svakog neurona trenutnog i slijedećeg sloja. Postoji nekoliko aktivacijskih funkcija koje se koriste u konvolucijskim neuronskim mrežama, a neke od poznatijih su:

- *ReLU*
- Sigmoidna funkcija (*sigmoid*)
- Hiperbolična tangentna funkcija (*tanh*)

ReLU aktivacijska funkcija je jedna od trenutno najčešće korištenih funkcija u konvolucijskim mrežama i definirana je izrazom (3-4)

$$f(x) = \max(0, x) \tag{3-4}$$

koja ukoliko je ulazna vrijednost funkcije negativna vraća vrijednost 0 dok u slučaju pozitivne vrijednosti na ulazu, ta vrijednost ostaje nepromijenjena i na izlazu, prikazano na slici 3.9.



Slika 3.9 *Prikaz grafa ReLU aktivacijske funkcije*

Jedan od važnijih razloga primjene ove funkcije nad ostalima je što se korištenjem ove funkcije rješava problema nestajućeg gradijenta (engl. *vanishing gradient problem*). Sigmoidna funkcija je nelinearna i ograničava vrijednost neurona u rasponu [0,1]. Kada je izlazna vrijednosti blizu 1,

neuron je aktivan i omogućuje protok informacija, dok vrijednosti blizu 0 odgovara neaktivnom neuronu. Izlaz sigmoidne aktivacijske funkcije može se interpretirati kao vjerojatnost obzirom da se vrijednosti nalaze u rasponu [0,1]. Zbog toga se koristi u izlaznim neuronima zadataka predviđanja. Izraz (3-5) opisuje sigmoidnu funkciju

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3-5}$$

gdje je x vrijednost ulaza neurona.

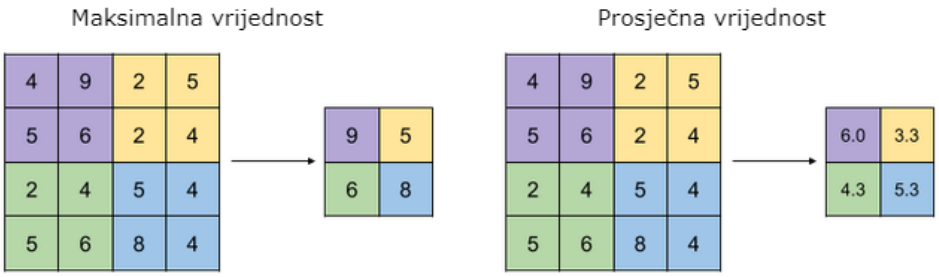
Sljedeća popularna funkcija aktivacije je tangentna hiperbolična funkcija koja se jednostavnije naziva *tanh* funkcija i dana je izrazom (3-6). *Tanh* je pomaknuta i rastegnuta inačica sigmoide.

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{3-6}$$

Raspon izlaza je [-1,1] i sličnog je rada u odnosu na sigmoidnu funkciju. Glavna razlika je činjenica da funkcija *tanh* stavlja ulazne vrijednosti na [-1,1] umjesto [0,1]. Ova funkcija također ima problema sa gubitkom gradijenta.

3.4.4. Sloj sažimanja

Slično konvolucijskom sloju, sloj sažimanja (engl. *pooling layer*) smanjenje prostornu veličinu konvolucijskih značajki ili preciznije, mapi značajki. Smanjivanjem dimenzionalnosti smanjuje se i računalna snaga potrebna za obradu podataka te u konačnici ubrzavanje procesa izračuna. Nadalje, koristi se za izvlačenje dominantnih značajki koje su pozicijske i rotacijske invarijantne odnosno nepromjenjive, čime se održava proces učinkovite obuke modela. Postoje dvije vrste sažimanja, sažimanje po maksimalnoj vrijednosti (engl. *max pooling*) i sažimanje po prosječnoj vrijednosti (engl. *average pooling*). *Max pooling* vraća maksimalnu vrijednost iz dijela slike pokrivenog jezgrom (engl. *kernel*) dok *average pooling* vraća prosjek svih vrijednosti iz dijela slike pokrivene jezgrom (Sl. 3.10).



Slika 3.10 *Primjer sažimanja vrijednosti max* i *avg sažimanjem*

Funkcija sažimanja maksimalnom vrijednosti također služi i kao prigušivač buke (engl. *noise*) unutar podataka. Odbacuju se bučne aktivacije i izvodi se uklanjanje šuma zajedno sa smanjenjem dimenzionalnosti dok funkcija sažimanja prosječnom vrijednosti izvodi smanjenje dimenzionalnosti kao mehanizam za suzbijanje šuma. Stoga se može reći da *max pooling* ima puno bolje rezultate od *average pooling*. Konvolucijski sloj i sloj sažimanja zajedno čine n-ti sloj konvolucijske mreže gdje n označava sloj izvlačenja značajki odnosno uzoraka iz slike. Ovisno o složenosti slika, broj takvih slojeva može se povećati za poboljšano izvlačenje značajki niske razine po cijenu veće računalne snage.

3.4.5. Potpuno povezani sloj

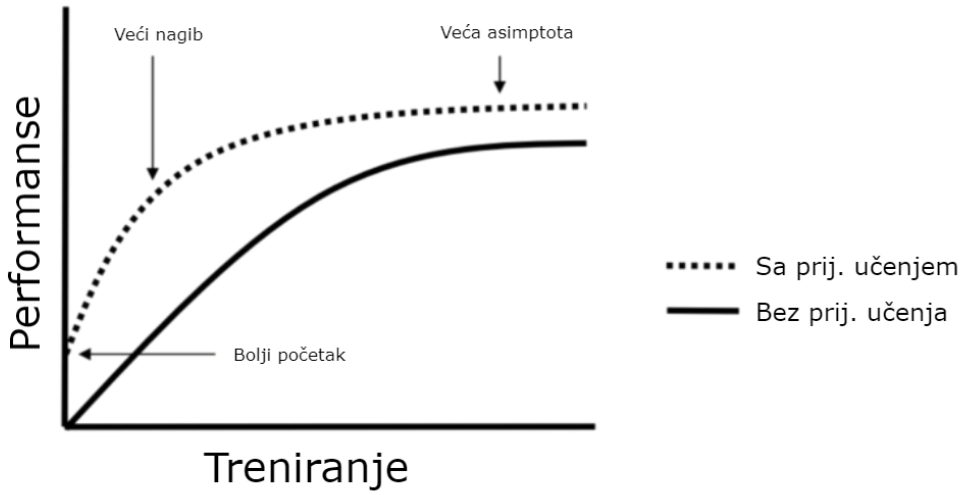
Nakon izvlačenja značajki iz ulaznih podataka potrebno je klasificirati podatke u različite klase, to se događa u posljednjem sloju konvolucijske neuronske mreže koji se naziva potpuno povezani sloj (engl. *fully connected layer*). Neuroni u potpuno povezanom sloju povezane su sa svim aktivacijama u prethodnom sloju (obično sloj sažimanja) kao što su i u neuronskim mrežama (Sl. 3.6). Između konvolucijskog dijela neuronske mreže i potpuno povezanog sloja obično se nalazi i operacija izravnavanja (engl. *flatten*) čija je uloga transformirati izlaz konvolucijskog dijela mreže na prihvatljiv oblik podatka kojeg potpuno povezani sloj može koristiti. Pošto se u konvolucijskom dijelu operacije vrše nad matricama, potrebno je matricu transformirati u vektor kojeg dalje koristi potpuno povezani sloj. Navedena operacija može se izvesti *softmax* funkcijom.

3.4.6. Prijenosno učenje

Osvrtom na prethodno definirani sloj konvolucijske mreže, objašnjeno je kako mreža izvlači i uči iz značajki. To je jedan od najzahtjevnijih i najsloženijih dijelova konvolucijske mreže koja za svaki novi problem odnosno svaki novokreirani model mora iznova učiti iz značajki. Kako bi se smanjilo vrijeme treniranja modela i bespotrebno ponavljanje zadataka poput otkrivanja rubova,

jednostavnih oblika, boja i slično, uvodi se prijenosno učenje (engl. *trasfer learning*). Ova metodologija proizlazi iz ideje da se značajke i težinske vrijednosti naučene u jednom modelu mogu koristiti za rješavanje drugog problema izradom novog modela.

Korištenjem metode prijenosnog strojnog učenja postiže se znatno smanjenje vremenskog dijela treniranja modela te su u konačnici rezultati precizniji i performanse modela bolje. Postoje modeli koji su naučeni na već postojećim podatkovnim skupovima koji u nekim slučajima sadržavaju i na desetke tisuća slika kojima su modeli učeni. Modeli naučeni na takvim podacima daju izvrsnu podlogu za izradu novih modela. Neki od poznatijih podatkovnih skupova su *ImageNet*, *COCO*, *Kaggle*, *MNIST*, *OpenImage* (*Google*-ov podatkovni skup s više od devet milijuna fotografija). Na slici 3.11. prikazana je prednost korištenja prijenosnog učenja nad klasičnim strojnim učenjem koji sve značajke uči iznova. Vidljivo je da model u početku treniranja već sadrži naučene značajke te je u značajnoj prednosti.



Slika 3.11 *Prikaz prednosti korištenja prijenosnog učenja*

Također je potrebno napomenuti da za model koji koristi prijenosno učenje nije potrebno imati prevelik početni skup podataka. Ovisno o sličnosti problema koji je riješen u prethodno naučenom modelu, potrebno je imati svega nekoliko stotina fotografija novog skupa kako bi novi model sa velikom preciznošću i u kratkom vremenskom periodu bio naučen za otkrivanje zadanog problema.

3.4.7. Mjere vrednovanja detektora objekata

Da bi se modelu pripisala ocjena valjanosti potrebno je definirati mjere ocjenjivanja.

Srednja vrijednost je matematički pojam definiran izrazom (3-7) koji opisuje prosjek uzorka. U statistici je definicija srednje vrijednosti slična prosjeku gdje je to zbroj svih danih vrijednosti podataka podijeljen s ukupnim brojem vrijednosti podataka danih u skupu

$$mAP = \frac{1}{n}\sum_{k=0}^{k=n} AP_k \tag{3-7}$$

gdje je AP prosječna preciznost klase k, n je ukupan broj kasa.

Preciznost mjeri postotak točnih predviđanja modela i definirana je izrazom (3-8)

$$P = \frac{TP}{TP + FP} \tag{3-8}$$

gdje su TP točno predviđeni objekti klase, FP netočno predviđeni.

Odziv mjeri postotak relevantnih podatkovnih točaka koje je model ispravno identificirao i definiran je izrazom (3-9)

$$R = \frac{TP}{TP + FN} \tag{3-9}$$

gdje je FN lažno prikazan nedostatak klase.

4. PROGRAMSKO RJEŠENJE ZA PRONALAZAK NAMETNIKA VAROA

U ovom poglavlju opisani su alati, programska okruženja i jezici te biblioteke potrebne za izradu ovog sustava.

4.1. Korištene programske tehnologije

Za realizaciju sustava koji detektira objekte potreban je priličan broj tehnologija. Najprije je potrebno pomoću algoritama učenja trenirati modele koji će kasnije biti u mogućnosti detektirati objekte na slikama i video zapisima.

4.1.1. Python

Python je objektno orijentirani programski jezik koji u današnje vrijeme pronalazi sve veću ulogu u području računarstva. Koristi se u strojnom učenju i podatkovnim znanostima, za izradu internet stranica i računalnih aplikacija, automatizaciju zadataka te između ostalog i kao jezik u kojem se pišu algoritmi za autonomnu vožnju vozila. Osim visoke primjene također je izrazito prenosiv što znači da se programski kod može izvršavati neovisno o platformi i okruženju. Python osim što je lak za korištenje, još bitnija značajka je brzina pisanja programa pa je tako pogodan i koristi se za pisanje skripti za obavljanje manjih poslova.

Prema [11], Python posjeduje izrazitu snagu s vrlo jasnom sintaksom. U sebi ima implementirane module, klase, dinamične tipove podataka vrlo visoke razine. Dodatna prednost korištenja ovog jezika je i veliki skup realiziranih modula i biblioteka koji potiču iz programske zajednice i dozvoljavaju brzu i jednostavnu implementaciju postojećih rješenja.

Za izradu ovog rada, korištene su 2.7 i 3.10 inačice programskog jezika zbog kompatibilnosti biblioteka i ostalih dijelova projekta.

4.1.2. C++

C++ je objektno orijentirani programski jezik koji je proizašao iz C programskog jezika i nadopunjen je alatima i značajkama. C++ jezik izrazito ističe koncepte poput nasljeđivanja, polimorfizma, enkapsulacije i slično, što čini programski kod fleksibilnim, modularnim i jednostavnim za održavanje. Efikasnost i performanse programskog koda jedni su od ključnih značajki. C++ posjeduje bogatu sintaksu i mogućnosti, omogućuje programerima da pišu kod

niske razine što je potrebno ukoliko izrada aplikacije, biblioteke, okruženja pa čak i operativnih sustava zahtjeva odlično upravljanje resursima poput memorije i procesorske snage što C++ jezik omogućuje. Koristi se u brojnim područjima računarstva što uključuje izradu programa za stolna računala, izrada računalnih igara, itd. Također jedna od bitnijih značajki ovog jezika je prenosivost što znači da se kod može izvršavati na velikom broju platformi i uređaja.

4.1.3. Javascript i Node.js

JavaScript jezik je sveobuhvatan i široko korišten programski jezik koji omogućuje interaktivnost i dinamički prikaz Internet stranica. U odnosu na tradicionalne programske jezike čija je primarna uloga u implementaciji pozadinskih sustava (engl. *backend*), JavaScript koristi se i u izradi aplikacije na klijentskoj strani. Omogućuje upravljanje sadržajem i ponašanjem stranice u stvarnom vremenu što za rezultat čini dinamičke stranice te pruža korisniku više interakcija sa stranicom i poboljšava korisničko iskustvo. JavaScript je jezik poznat po svojoj fleksibilnosti i jednostavnom korištenju. Tokom godina, jezik je značajno napredovao te su razne programske biblioteke i okruženja razvijena s ciljem još jednostavnijeg i bržeg procesa razvijanja aplikacija.

Biblioteke i alati poput jQuery, te okruženja i razvojni okviri poput React, Angular i Vue.js koriste se u širokom pojasu kako bi razvijali kompleksne i značajkama bogate Internet aplikacije. JavaScript više nije ograničen na izradu stranica samo na korisničkoj strani, pojavom Node.js JavaScript koristi se i na poslužiteljskoj strani. To je dovelo do izrade Internet aplikacija i stranica gdje je se gotovo u potpunosti koristi isključivo JavaScript programski jezik. JavaScript je jedan od temeljnih alata u izradi modernog i interaktivnog Interneta.

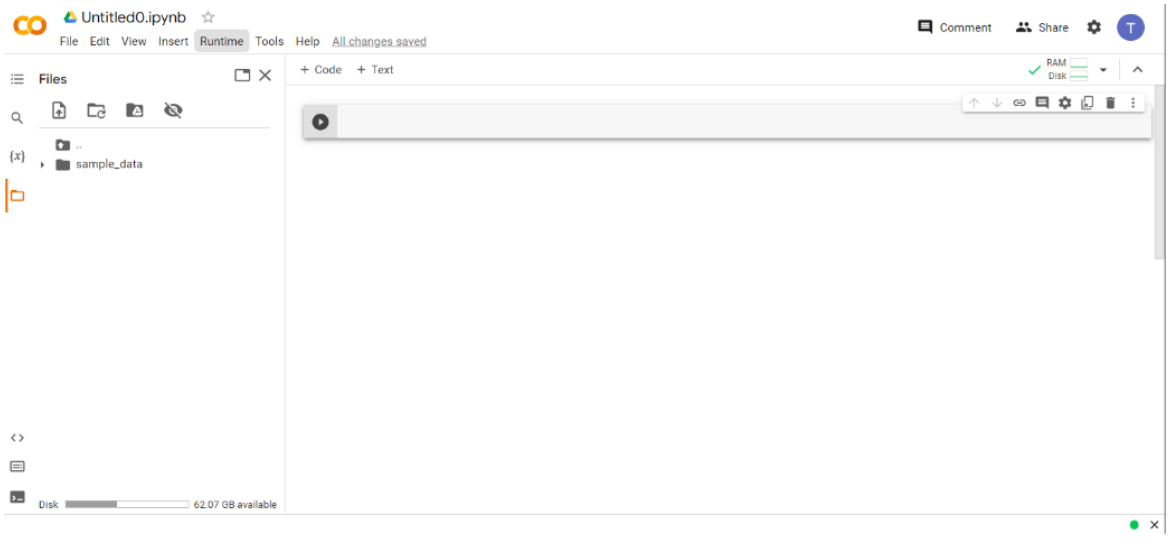
4.1.4. Jupyter Notebook i Google Colab

Jupyter Notebook je web aplikacija otvorenog koda koja se koristi za stvaranje i dijeljenje dokumenata koji sadrže programski kod, jednadžbe, vizualizacije i tekst. [12] Jupyter notebook je klijentska internet aplikacija odnosno okruženje koje omogućuje postepeno izvršavanje programskog koda te jednostavnu izmjenu istog. Jupyter Notebook intenzivno se koristi u svrhu istraživanja podataka. Koristi se i za dokumentiranje i dijeljenje programskog koda.

Google Colab je alat kojeg je izradio istraživački tim tvrtke Google. Colab omogućuje pisanje proizvoljnog python koda putem internet preglednika, a posebno je pogodan za strojno učenje, analizu podataka i edukaciju. Zbog lakšeg i jednostavnijeg pristupa strojnom učenju, Jupyter-ova

usluga poslužena je na udaljenom računalu koja ne zahtjeva postavljanje za rad iz razloga što je većina osnovnih biblioteka i alata unaprijed instalirana na virtualnim računalima prema [13].

Colab također pruža korisnicima besplatan pristup izrazito moćnim hardverskim resursima poput GPU, CPU, dovoljno radne memorije i zapremine na disku. Na sljedećoj slici 4.1 vidljiv je prikaz sučelja *Google Colab*-a na kojemu se nalazi uređivač teksta i prikaz datoteka unutar projekta.



Slika 4.1 *Prikaz sučelja Colab*

4.1.5. Tensorflow i Keras

TensorFlow je programski okvir otvorenog koda koji se koristi za implementaciju modela dubokog i strojnog učenja za potrebe klasifikacije, detekcije objekata, obrada prirodnog jezika i zvuka i sl. Najčešće se koristi uz programski jezik Python no moguća je uporaba ostalih jezika poput C++, JavaScript i Java. TensorFlow se može opisati i kao programska biblioteka koja koristi linearnu algebru i statistiku za izračune i prikaz podataka. Naziv je dobiven od riječi tenzor koji opisuje višestruku povezanost algebarskih objekata unutar vektorskog prostora odnosno u programiranju poznatiji kao višedimenzionalni niz.

Tensorflow se kao programski okvir ističe zbog velike zajednice i kolekcije API-a za obradu podataka, vizualizaciju, evaluaciju modela i jednostavnu implementaciju što čini strojno učenje jednostavnijim. Iznimno je prenosiv i izvodi se na manjim uređajima poput pametnog telefona, mikrokontrolera poput Arduino i Raspberry uz pomoć Tensorflow Lite. Također, može se izvoditi i na internet pregledniku uz pomoć TensorflowJS, te ima svojstvo skalabilnosti glavne jezgre koja podržava izvođenje na više TPU i GPU odnosno CUDA jezgri. Koristi se u medicini za detekciju

objekata prilikom skeniranja magnetskom rezonancom i drugim uređajima, društvenim mrežama, multimedijskim (*streaming)* platformama u svrhu predlaganja glazbe, detekciji zlonamjernih radnji prilikom transakcija i sl.

Prema [14], Keras je aplikacijsko programsko sučelje (engl. *API* - application programming interface) visoke razine napisano u Python jeziku s ciljem da se ubrza i olakša rad sa programskim okvirima niske razine. Karakteristike koje opisuju Keras sučelje su brza implementacija i jednostavna izrada modela strojnog učenja u svega nekoliko linija programskog koda. Moguća je implementacija neuronskih mreža svih poznatih arhitektura i prenosivost istih. Od pojave 2.0 inačice Tensorflow-a, Keras je u potpunosti integriran u TensorFlow okruženje što omogućuje jednostavnu implementaciju neuronskih mreža neovisno o problematici.

4.1.6. Torch i PyTorch

PyTorch je programsko okruženje otvorenog koda za strojno učenje. Facebook inženjeri su implementirali ovo okruženje u programskoj jeziku Python, a naziv je dobio po Python-u i zastarjelom internet pregledniku Torch-u. PyTorch je također jedan od alata koji pojednostavljuju proces izrade neuronskih mreža. PyTorch je modularan i interoperabilan set temeljnih blokova za izradu mreže. Sadrži temeljni set značajki koji omogućuju razvoj, optimizaciju i produkciju. PyTorch je izrađen oko tenzor klase odnosno višedimenzionalnog niza i omogućuje izvršavanje zadataka korištenjem više procesnih jezgri CPU/GPU poput TensorFlow-a.

4.2. Priprema podatkovnog skupa

Kako bi se započeo proces učenja modela strojnog učenja najprije je potrebno stvoriti skup podataka i prilagoditi zahtjevima algoritma strojnog učenja. Koraci izrade podatkovnog skupa:

- Prikupljanje podataka
- Uklanjanje praznih slika (ne sadrže traženi objekt)
- Postavljanje specifične veličine slika ($širina \times dužina\ piksela$)
- Datotečni nastavak ulaznih podataka mora biti jedinstven kroz cijeli skup (.jpg, .png i dr.)
- Izrada prijenosne datoteke sa oznakama (.txt, .xml, .json, .csv i dr.)

Ukoliko postojeći skup nije zadovoljavajući u odnosu na složenost problema otkrivanja određenog objekta, skup se dodatno poboljšava raznim izmjenama koje su navedene u nastavku rada.

4.2.1. Odstranjivanje nepotrebnih podataka iz skupa

Početni skup korišten u ovom radu sastoji se od 50 upotrebljivih slika. Slike su visoke razlučivosti i kvalitete te ih je potrebno obraditi prije učenja modela. Analizom podatkovnog skupa može se primijetiti da poveći dio slika sadrži područja koja nisu od interesa odnosno ne sadrže grinju varoa. U ovom slučaju, podnica košnice na travnatoj površini (Sl. 4.2).

Slika 4.2 *Primjer slike početnog skupa*

Kako bi se izbjeglo nepotrebno korištenje računalnih resursa ovaj dio slike potrebno je ukloniti na svim slikama unutar skupa. Rezultat je prikazan sljedećom slikom 4.3.

Slika 4.3 *Izrezan nepotreban dio slike*

Nakon što su iz skupa izbačeni nepotrebni dijelovi, preostaje podijeliti skup na manje slike. Python skripta koja dijeli sliku na manje dijelove nalazi se u *split-image* Python paketu. Skripta se poziva putem naredbe terminala „*split-image image_dir x y*“ gdje *image_dir* predstavlja lokaciju slike ili mape slika, x i y predstavljaju broj horizontalnih i vertikalnih presjeka slike. Ova tehnika zove se segmentacija slike i objašnjena je u drugom poglavlju ovog rada. Nakon analize podataka dobivenih ovom tehnikom, stvorene su manje slike koje također ne sadržavaju grinju te ih je potrebno ukloniti. Dio slike iz podatkovnog skupa koji ne sadrži grinju (Sl. 4.4).

Slika 4.4 *Nepotreban dio podatkovnog skupa*

Ovim jednostavnim koracima predobrade podataka omogućuje se brži prolazak algoritma strojnog učenja kroz podatkovni skup i smanjuje opterećenje na komponente. Ovaj korak je u pravilu neophodan za rad algoritama jer učitavanje velikih slika u memoriju grafičke kartice obično nije moguć, posebice kada se koriste skupovi koji sadrže na desetke tisuća slika.

4.2.2. Povećavanje veličine skupa

Nakon ovih koraka, podatkovni skup se sastoji od 160 slika. Ovakav skup sadrži relativno malo podataka za treniranje pa je potrebno novonastali skup dodatno popuniti umjetno stvorenim podacima. Neke od tehnika stvaranja umjetnih podataka:

- Segmentacija slike
- Rotacija slike
- Rotacija označenih objekata
- Umetanje šuma

- Umetanje raznih pozadina
- Izmjena intenziteta osvijetljenosti (iluminacija)
- Zamučivanje/zamagljivanje slike

Nakon dodatno stvorenih umjetnih slika, skup se sastoji od 1044 slike identičnih dimenzija (640x640 piksela u boji).

4.2.3. Označavanje objekata unutar podatkovnog skupa

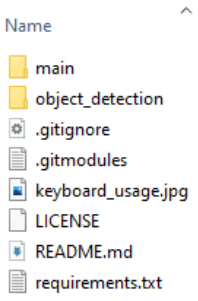
Nakon pripreme podatkovnog skupa, tražene objekte potrebno je označiti. Označavanje slike (engl. *image annotation*), vrsta je označavanja podataka koja se usredotočuje na prepoznavanje i označavanje specifičnih detalja ili područja na slici. U računalnom vidu označavanje podataka uključuje dodavanje oznaka podacima kao što su slike i videozapisi. Svaka oznaka predstavlja klasu koja je pridružena podacima. Modeli strojnog učenja koriste oznake kada uče identificirati klase objekata u neklasificiranim podacima.

Označavanje slike koristi se u području računalnog vida kako bi se stvorili skupovi podataka za izradu modela strojnog učenja. Takvi skupovi dodatno se dijele na:

- Skup za treniranje
- Skup za validaciju
- Testni skup

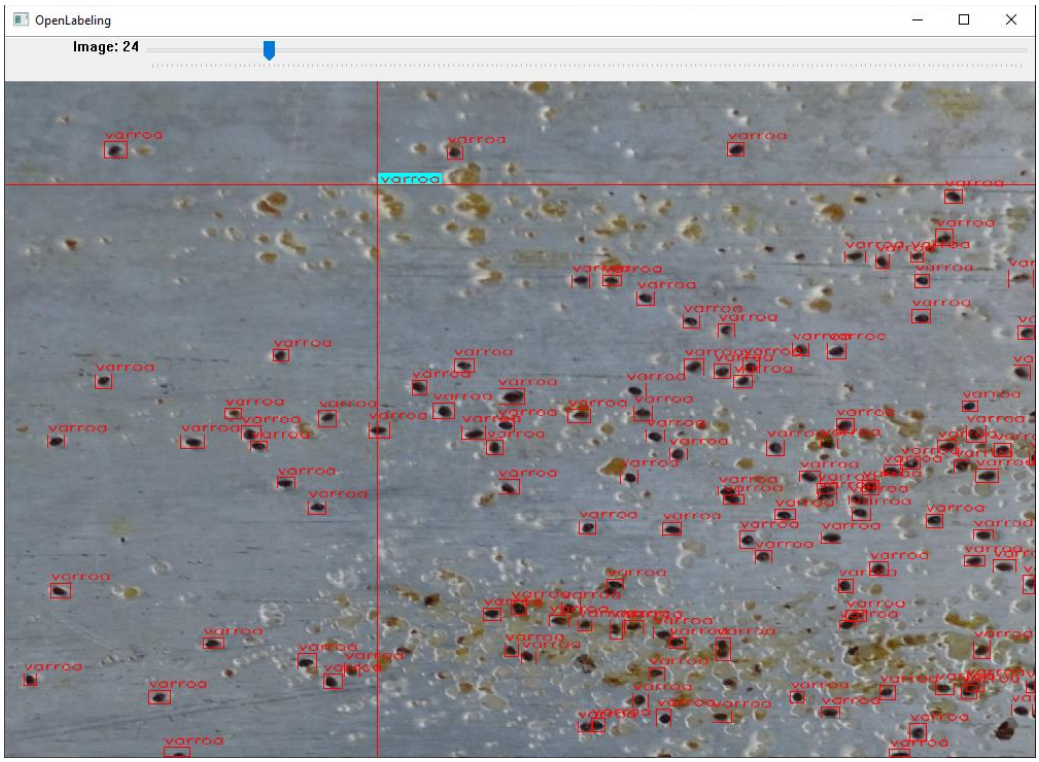
Skup za treniranje koristi se pri učenju modela. Nakon učenja potvrđuje se valjanost modela validacijskim skupom. Ukoliko rezultati nisu zadovoljavajući, slijedi izmjena parametara i učenje se nastavlja. Učenje se završava testnim skupom koji pokazuje rad modela u stvarnom svijetu nad podacima koji prethodno nisu viđeni.

Za označavanje podataka (engl. *data labeling*) u ovom radu korišten je alat za označavanje OpenLabeling. Mapa navedenog modula prikazana je sljedećom slikom (Sl. 4.5).



Slika 4.5 *Sadržaj OpenLabeling GIT repozitorija*

Unutar mape „*main*“ nalaze se dvije mape pod nazivom „*input*“ i „*output*“ i datoteka „*class_list.txt*“. Prije nego započne proces označavanja, potrebno je popuniti mapu *input* sa podacima koji će biti korišteni u procesu učenja dok je u .txt datoteku potrebno upisati sve klase koje su sadržane u podatkovnom skupu. Nakon pokretanja skripte naredbom „*python main.py*“ otvara se jednostavno sučelje i započinje proces označavanja (Sl. 4.6). Prilikom označavanja podataka, u mapi *output* se automatski stvaraju oznake objekata pripadajućih klasa u formate za prijenos podataka .txt, .json, .csv, .yaml, itd.



Slika 4.6 *Proces označavanja klase podatkovnog skupa*

4.3. Treniranje modela

Nakon što su ulazni podaci uspješno označeni potrebno je odabrati model na kojem će se podaci primijeniti. Prema izvorima [15], [16], [17] i [18] prethodno je proučeno koji algoritmi detekcije mogu odgovarati rješenju problema ovog rada. Za potrebe rada i istraživanja odabrano je nekoliko implementacija modela strojnog učenja vrhunskih performansi:

- SSD
- FasterRCNN
- YOLOv3
- YOLOv4

- YOLOv5
- YOLOv7
- YOLOv8
- YOLOS
- YOLOR

Prije učenja modela potrebno je postaviti određene parametre učenja kako bi se postigao uniforman pregled rezultata svih modela. Parametri korišteni pri treniranju modela prikazani su sljedećom tablicom 5.1.

Tablica 5.1 *Hiperparametri modela*

Naziv parametra	**Vrijednost/opis**
Broj epoha	20-50
Stopa učenja	0.001, 0.01
Broj slika po iteraciji	4, 8, 16
Optimizacijski algoritam	Adam
Weights datoteka	(algoritam) + nano/small/medium
Veličina slike	640*640 piksela

Navedeni parametri obično variraju od algoritma do algoritma, a na njihovu vrijednost utječu razni čimbenici poput načina na koji je algoritam implementiran, programsko okruženje, računalni resursi i slično. Broj epoha je vrijednost kojom je zadan broj iteracija kroz cijeli podatkovni skup, stopa učenja je vrijednost kojom je opisana brzina učenja zadanog modela, broj slika po iteraciji je vrijednost kojom je zadan broj ulaznih slika kroz algoritam unutar jedne iteracije kroz mrežu. Optimizacijski algoritam se kao funkcija u učenju mreža koristi za bolju prilagodbu težinskih vrijednosti s ciljem poboljšanja točnosti i vremena učenja mreže. *Weights* datoteka je datoteka koja se koristi za učenje modela prijenosnog učenja, a u njoj se nalaze težinske vrijednosti modela učenih na većim podatkovnim skupovima.

Za izradu navedenih modela korištena je usluga GoogleColab koja omogućuje učenje modela na udaljenim poslužiteljima i besplatnim korištenjem računalnih resursa.

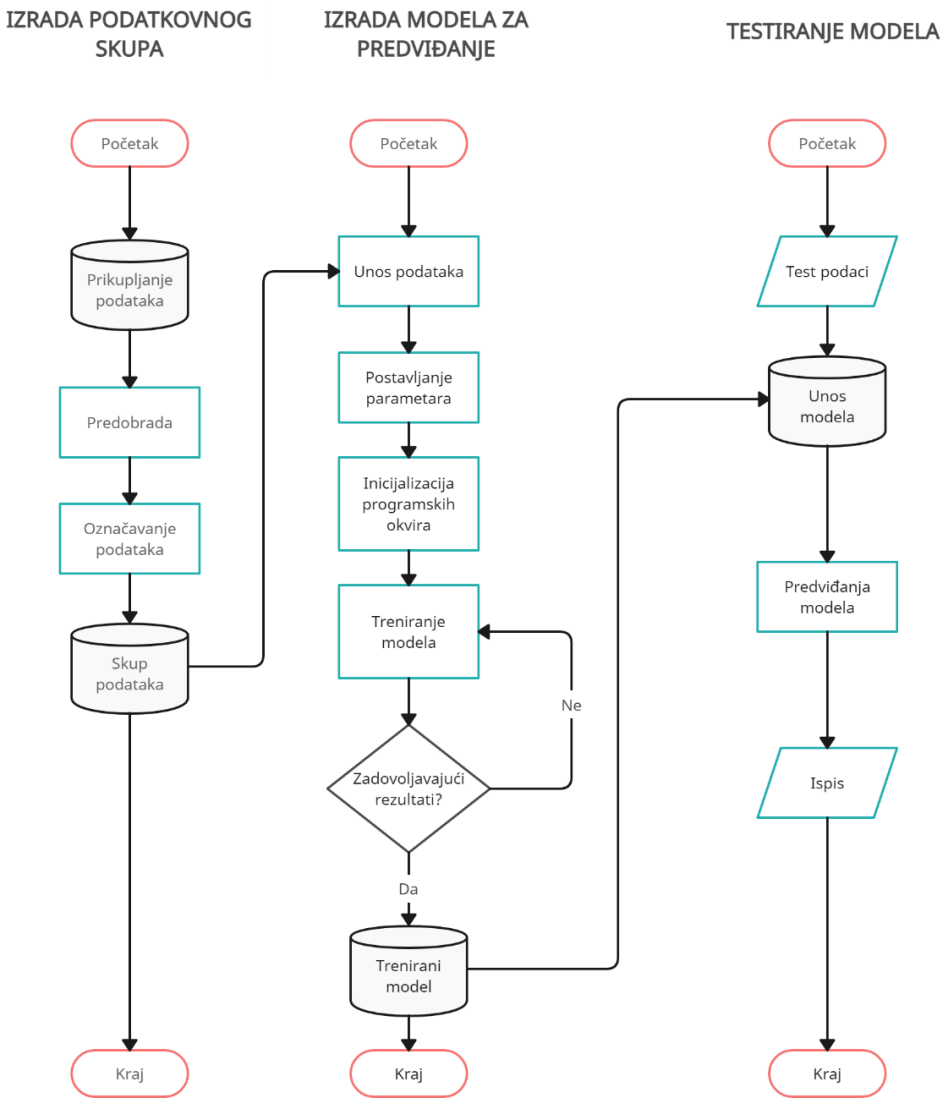
Podatkovni skup korišten za izradu modela sastoji se od sveukupno 1044 slike koji su za učenje raspoređeni prema omjeru 0.82:0.12:0.6 odnosno 856 slika koristi se za učenje, 128 slika za evaluaciju prilikom učenja i 60 testnih slika koje se koriste na kraju učenja. Unutar tih slika nalazi se 15800 zasebno označenih instanci grinje varoe. Kod na slici 4.7 prikazuje primjer naredbe za učenje modela.

```
[ ]  %cd {HOME}
     %%time

     !yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml epochs=40 imgsz=640 plots=True
```

Slika 4.7 *Naredba za početak treniranja modela*

Na sljedećoj slici 4.8 prikazani su koraci pri učenju svih modela, a ovisno o programskom okviru postupak izrade modela varira.



Slika 4.8 *Dijagram izrade modela u radu*

Kako bi se modeli mogli koristiti na stvarnim podacima, potrebno je implementirati aplikacije koje koriste sve prednosti naučenih modela u stvarnom vremenu. Nakon treniranja svih modela, potrebno je procijeniti koji najbolje odgovara potrebama ovog rada. Neki od kriterija su: veličina

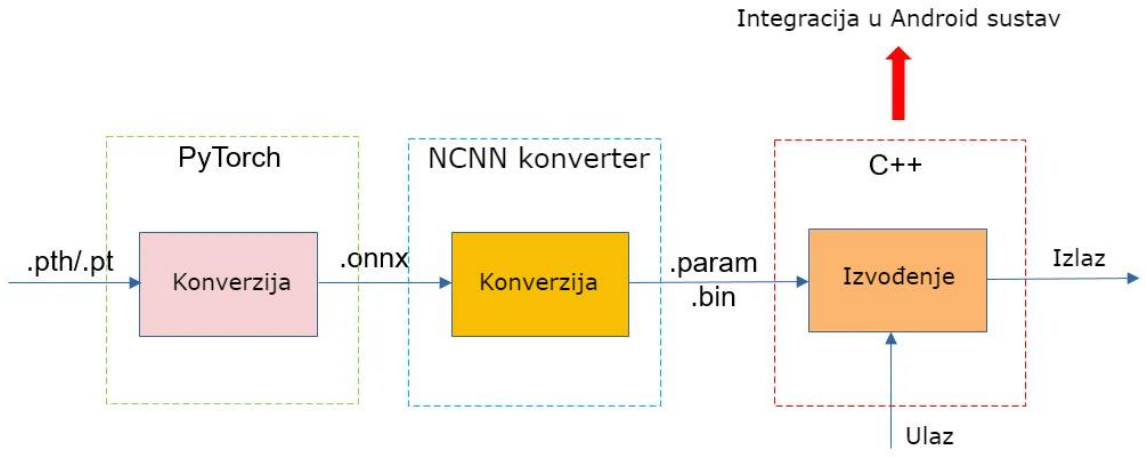
modela, brzina operacija nad podacima, jednostavnost implementiranja i slično. Prije implementiranja modela u aplikaciju, model je potrebno dodatno prilagoditi.

4.4. Prijenosni formati

Nakon što je model treniran u određenom okruženju obično nije kompatibilan s ostalim programskim okruženjima niti s pojedinim platformama. Razlog je prvenstveno drugačija implementacija algoritma odnosno funkcije i metode koje se koriste za učenje modela koje mogu dati iste ili slične rezultate u odnosu na druga okruženja što ih ne čini jednakima.

Kako bi se trenirani model koristio u drugim okruženjima i platformama, potrebno je funkcije prilagoditi odgovarajućim standardima ili ih pojednostaviti. ONNX *(*engl. *open neural network exchange)* je jedan od alata koji se koristi pri implementaciji programskih rješenja. ONNX može pojednostaviti gotovo svaki algoritam sa svakom funkcijom koja je implementirana izvan određene norme.

Iako se ONNX format koristi i za testiranje na uređajima, njegova glavna uloga je prenosivost modela. Postoje bolja rješenja za testiranje rada modela na uređajima od navedenog, a jedan od poznatijih je NCNN format gdje je naglasak primarno na resurse računala, podizanje performansi modela kroz efikasniju uporabu centralne procesorske jedinice i posebno optimizirane za rad na mobilnim uređajima. Također je bitno napomenuti da je NCNN okruženje neovisno o drugim okruženjima te za rad modela na uređajima ne zahtjeva instalaciju programskih paketa drugih okruženja. Dijagram izmjene formata vidljiv je na sljedećoj slici 4.9.



Slika 4.9 *Konverzija modela*

4.5. Android aplikacija

Aplikacija za pronalazak grinje varoa izrađena je u programskom okruženju Android sustava te je logika rada aplikacije napisana u C++ programskom jeziku koji se izvršava u Java okruženju. Zbog preciznosti algoritma i visokih rezultata prilikom evaluacije rada modela, odabran je model Yolov8.

Model Yolov8 radi na način da potpuno povezani slojevi najprije prekriju čitavu ulaznu sliku s matricom određene veličine. Broj slojeva je obično tri i svaki je specijaliziran za detekciju različitih veličina objekata. Parametar broja slojeva moguće je mijenjati prije samog učenja modela sve do maksimalnih šest slojeva ili se dozvoli modelu da sam korigira taj parametar u ovisnosti o podatkovnom skupu i rezultatima tijekom učenja. Svaki sloj generira elemente određene veličine. Obično su to veličine matrica 80x80, 40x40, 20x20. Važno je napomenuti da su elementi zaduženi za detekciju objekata unutar samog sebe do maksimalno jednog objekta tokom treniranja te za svaku klasu prilikom izvođenja modela. Prema tome, moguće je reći da je za potrebe ovog rada poželjno da elemenata bude što više. Na slici 4.10. prikazana je funkcija *generate_grids_and_stride* koja stvara elemente imaginarnih matrica preko cijelog ulaznog podatka. Funkciji se prosljeđuje visina i širina elementa te referenca na vektor pomaka i strukturu elementa.

```
static void generate_grids_and_stride(const int target_w, const int target_h, std::vector<int>& strides, std::vector<GridAndStride>& grid_strides)
{
    for (int i = 0; i < (int)strides.size(); i++)
    {
        int stride = strides[i];
        int num_grid_w = target_w / stride;
        int num_grid_h = target_h / stride;
        for (int g1 = 0; g1 < num_grid_h; g1++)
        {
            for (int g0 = 0; g0 < num_grid_w; g0++)
            {
                GridAndStride gs;
                gs.grid0 = g0;
                gs.grid1 = g1;
                gs.stride = stride;
                grid_strides.push_back(gs);
            }
        }
    }
}
```

Slika 4.10 *Kreiranje matrice nad ulaznim podatkom*

Sljedeće se poziva funkcija *generate_proposals* (Sl. 4.11), njoj se predaje niz objekata novonastalih elemenata matrice, prag vjerojatnosti da se klasa nalazi unutar elementa, te niz objekata koji sadrže naziv klase i vjerojatnost (engl. *probability*) predloženog objekta. Funkcija vraća koordinate središta predloženog objekta skupa sa visinom i širinom okvira kutije, ime klase

i vjerojatnost točnog predviđanja. Ukoliko je vjerojatnost da se objekt nalazi unutar elementa manja od zadanog praga, elementi se uklanjaju. Pozitivnim rezultatima pridružuje se nelinearnost i vrijednosti kutije spremaju se u strukturu podatka koja će biti proslijeđena.

```
static void generate_proposals(std::vector<GridAndStride> grid_strides, const ncnn::M
{
    const int num_points = grid_strides.size();
    const int num_class = 1;
    const int reg_max_1 = 16;

    for (int i = 0; i < num_points; i++)
    {
        const float* scores = pred.row(i) + 4 * reg_max_1;

        // find label with max score
        int label = -1;
        float score = -FLT_MAX;
        for (int k = 0; k < num_class; k++)
        {
            float confidence = scores[k];
            if (confidence > score)
            {
                label = k;
                score = confidence;
            }
        }
        float box_prob = sigmoid(score);
        if (box_prob >= prob_threshold)
        {
            ncnn::Mat bbox_pred(reg_max_1, 4, (void*)pred.row(i));
            {
                ncnn::Layer* softmax = ncnn::create_layer("Softmax");

                ncnn::ParamDict pd;
                pd.set(0, 1); // axis
                pd.set(1, 1);
                softmax->load_param(pd);

                ncnn::Option opt;
                opt.num_threads = 1;
                opt.use_packing_layout = false;

                softmax->create_pipeline(opt);

                softmax->forward_inplace(bbox_pred, opt);

                softmax->destroy_pipeline(opt);

                delete softmax;
            }

            float pred_ltrb[4];
            for (int k = 0; k < 4; k++)
            {
                float dis = 0.f;
                const float* dis_after_sm = bbox_pred.row(k);
                for (int l = 0; l < reg_max_1; l++)
                {
                    dis += l * dis_after_sm[l];
                }

                pred_ltrb[k] = dis * grid_strides[i].stride;
            }

            float pb_cx = (grid_strides[i].grid0 + 0.5f) * grid_strides[i].stride;
            float pb_cy = (grid_strides[i].grid1 + 0.5f) * grid_strides[i].stride;

            float x0 = pb_cx - pred_ltrb[0];
            float y0 = pb_cy - pred_ltrb[1];
            float x1 = pb_cx + pred_ltrb[2];
            float y1 = pb_cy + pred_ltrb[3];

            Object obj;
            obj.rect.x = x0;
            obj.rect.y = y0;
            obj.rect.width = x1 - x0;
            obj.rect.height = y1 - y0;
            obj.label = label;
            obj.prob = box_prob;

            objects.push_back(obj);
        }
    }
}
```

Slika 4.11 *Metoda generate_proposals*

Nakon dodavanja usidrenih kutija (engl. *anchor boxes*) potrebno je za svaki element dobiti izračune koji govore o tome koliko je model siguran da se unutar elementa nalazi traženi objekt te nakon toga dodatno izbaciti kutije koje ne zadovoljavaju prag preklapanja *IoU (*engl. *intersection over union)*. To se postiže pomoću funkcija *nonmax suppression* i *IoU* (Sl. 4.12) kojoj se predaje prag preklapanja i niz odabranih elemenata. *IoU* je moguće promatrati kao evaluaciju rada modela, tj. koliko dobro model predviđa okvire za navedene objekte. *IoU* traži 2 kuta presjeka te računa područje presjeka. *Nonmax suppression* funkcija odbacuje predložene kutije čija vrijednost postojanja objekta unutar kutije ne prelazi zadovoljavajući prag procijene od 0.4. Nakon toga odabire se kutija sa najvećom vrijednosti postojanja objekta i odbacuju se sve ostale čija vrijednost *IoU* praga (engl. *threshold*) ne prelazi 0.5. Ovim postupkom uklanjaju se preostale slične predložene kutije koje nisu dio stvarne kutije. Ukoliko se uklone kutije koje su jednake ili prelaze *IoU* prag, tada bi se uklonile i ostale kutije koji su zapravo dio prave kutije koja objedinjuje objekt. Pozitivnim rezultatima pridružuje se nelinearnost i vrijednosti kutije spremaju se u strukturu podatka koja će biti proslijeđena na ispis.

```
static void nms_sorted_bboxes(const std::vector<Object>& faceobjects, std::vector<int>& picked, float nms_threshold)
{
    picked.clear();

    const int n = faceobjects.size();

    std::vector<float> areas(n);
    for (int i = 0; i < n; i++)
    {
        areas[i] = faceobjects[i].rect.width * faceobjects[i].rect.height;
    }

    for (int i = 0; i < n; i++)
    {
        const Object& a = faceobjects[i];

        int keep = 1;
        for (int j = 0; j < (int)picked.size(); j++)
        {
            const Object& b = faceobjects[picked[j]];

            // intersection over union
            float inter_area = intersection_area(a, b);
            float union_area = areas[i] + areas[picked[j]] - inter_area;
            // float IoU = inter_area / union_area
            if (inter_area / union_area > nms_threshold)
                keep = 0;
        }

        if (keep)
            picked.push_back(i);
    }
}
```

Slika 4.12 *Nonmax suppresion i IoU funkcije*

Nakon što su dobivene koordinate zadovoljavajuće kutije, rezultati se prikazuju (Sl. 4.13).

```cpp
static Yolo* g_yolo = 0;
static ncnn::Mutex lock;

class MyNdkCamera : public NdkCameraWindow
{
public:
    virtual void on_image_render(cv::Mat& rgb) const;
};

void MyNdkCamera::on_image_render(cv::Mat& rgb) const
{
    // nanodet
    {
        ncnn::MutexLockGuard g(lock);

        if (g_yolo)
        {
            std::vector<Object> objects;
            g_yolo->detect(rgb, objects);
            g_yolo->draw(rgb, objects);
            // Display total count of detected objects
            char count_text[32];
            sprintf(count_text, "Total Objects: %zu", objects.size());

            int baseLine = 0;
            cv::Size label_size = cv::getTextSize(count_text, cv::FONT_HERSHEY_SIMPLEX, 0.5, 1, &baseLine);

            int y = 20;
            int x = 10;

            cv::rectangle(rgb, cv::Rect(cv::Point(x, y - label_size.height), cv::Size(label_size.width, label_size.height + baseLine)),
                          cv::Scalar(255, 255, 255), -1);

            cv::putText(rgb, count_text, cv::Point(x, y),
                        cv::FONT_HERSHEY_SIMPLEX, 0.5, cv::Scalar(0, 0, 0));
        }
        else
        {
            draw_unsupported(rgb);
        }
    }

    draw_fps(rgb);
}
```

Slika 4.13 *Ispis kutija, postotak predviđanja i fps*

Na slici 4.14 prikazani su rezultati izvođenja algoritma u Android okruženju.

Slika 4.14 *Rezultati detekcije mobilne aplikacije*

4.6. Internet aplikacija

Za potrebe ovog rada također je napravljena i Internet aplikacija, web Node.js usluga s podrškom *ONNX-runtime*-a. Aplikacija radi na način da se najprije klikne na gumb *Open local image*. Nakon klika na gumb otvara se prozor za učitavanje slike, odabirom slike i klikom na gumb *Open* slika se prilagođava postavkama modela identičnima kao i prilikom treniranja. Nakon prilagodbe veličine slike, sprema se u tip podatka zvan tenzor (Sl. 4.16).

```
const [modelWidth, modelHeight] = inputShape.slice(2);
const [input, xRatio, yRatio] = preprocessing(image, modelWidth, modelHeight);

const tensor = new Tensor("float32", input.data32F, inputShape); // to ort.Tensor
```

Slika 4.16 Prilagodba ulazne slike

Nadalje se obavljaju operacije kao i što je ranije spomenuto u izradi mobilne aplikacije. Slijedi prekrivanje slike imaginarnom matricom određene veličine, te nakon toga *Non-max suppression* funkcije i prolazak kroz preostali dio elemenata i spremanje koordinata predloženih kutija, broj klasa i vrijednost parametra predviđanja (Sl. 4.17).

```
  // looping through output
  for (let idx = 0; idx < selected.dims[1]; idx++) {
    const data = selected.data.slice(idx * selected.dims[2], (idx + 1) * selected.dims[2]); // get rows
    const box = data.slice(0, 4);
    const scores = data.slice(4); // classes probability scores
    const score = Math.max(...scores); // maximum probability scores
    const label = scores.indexOf(score); // class id of maximum probability scores

    const [x, y, w, h] = [
      (box[0] - 0.5 * box[2]) * xRatio, // upscale left
      (box[1] - 0.5 * box[3]) * yRatio, // upscale top
      box[2] * xRatio, // upscale width
      box[3] * yRatio, // upscale height
    ]; // keep boxes in maxSize range

    boxes.push({
      label: label,
      probability: score,
      bounding: [x, y, w, h], // upscale box
    }); // update boxes to draw later
  }

  renderBoxes(canvas, boxes); // Draw boxes
  input.delete(); // delete unused Mat
};
```

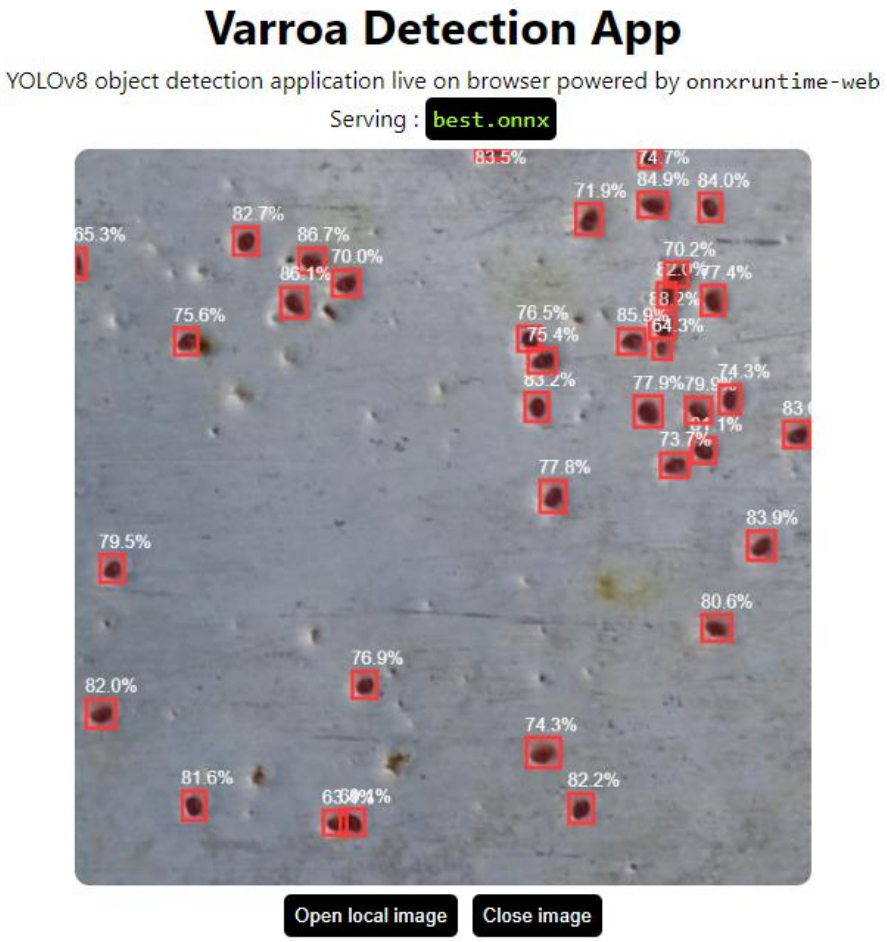
Slika 4.17 *Pretraga kutija*

Preostaje ispisati detektirane okvire i izbrisati preostale (Sl. 4.18).

```
renderBoxes(canvas, boxes); // Draw boxes
input.delete(); // delete unused Mat
```

Slika 4.18 *Ispis rezultata predviđanja na zaslon uređaja*

Rezultati izvođenja modela na Node.js platformi prikazani su na sljedećoj slici 4.19.



Slika 4.19 *Prikaz rezultata Internet aplikacije*

Da bi se na jednostavan način prikazao tijek izrade cjelokupnog sustava, predstavljen je dijagram i priložen je u prilogu P.4.1. U prilogu su slikovito prikazani svi prethodno objašnjeni koraci.

5. TESTIRANJE I EVALUACIJA SUSTAVA

Kako bi se utvrdio ispravan rad nekog sustava, potrebno ga je testirati.

5.2. Minimalni zahtjevi za izradu i rad sustava

Sljedeće tablice prikazuju konfiguraciju uređaja na kojim je implementiran ovaj sustav. Prva tablica 5.1 je konfiguracija virtualnog računala u oblaku koje posjeduje Google i na kojima su naučeni svi modeli u ovom radu. Operativni sustav je Linux, distribucija Ubuntu.

Tablica 5.1 *Preporučena minimalna konfiguracija za treniranje modela strojnog učenja*

CPU	Intel Xeon @ 2.00GHz
GPU	Tesla T4
GPU VRAM	15GB
RAM	8GB
HDD/SSD	20GB

Tablica 5.2 prikazuje konfiguraciju računala na kojem je implementiran kod ovog rada i na kojem se pokretala Internet aplikacija.

Tablica 5.2 *Preporučena minimalna konfiguracija za izradu Internet i Android aplikacije*

CPU	Intel Core i7-7700HQ @ 2.8Hz (8CPU)
GPU	Nvidia GeForce GTX 1050Ti
GPU VRAM	4GB
RAM	16GB
HDD/SSD	100GB

Tablica 5.3 prikazuje minimalnu konfiguraciju koja može pokrenuti Android aplikaciju. Pretpostavka je da ju mogu pokretati svi Android uređaji pod uvjetom da je inačica Android SDK veća ili jednaka 28.

Tablica 5.3 *Minimalna konfiguracija za testiranje Android aplikacije*

CPU	Octa-core (4x2.4 GHz Cortex-A73 & 4x1.8 GHz Cortex-A53)
GPU	Mali-G71 MP8
GPU VRAM	2GB
RAM	4GB
HDD/SSD	20GB
SDK inačica	28<=inačica

5.1. Testiranje modela

U tablici 5.4 prikazani su rezultati metoda mjerenja detektora objekata.

Tablica 5.4 *Minimalna konfiguracija za testiranje Android aplikacije*

Model	**mAP**	**P**	**R**
YOLOv3	0.975	0.966	0.945
YOLOv3-tiny	0.933	0.941	0.879
YOLOv3-SPP	0.975	0.961	0.943
Yolov5-small	0,924	0,91	0,868
YOLOv7	0,859	0,837	0,83
YOLOv8	0,972	0,964	0,955
YOLOv8-nano	0,965	0,95	0,931
YOLOR	0,862	0,696	0,933
YOLOS	0,718	0,264	0,407
Faster-RCNN-FPN	0,432	0,493	0,578
SSD	0,44	0,44	0,53

*FPN-Feature Piramidal Netrowk, *SPP-Spatial Piramidal Pooling, *tiny,nano, small – manje inačice modela

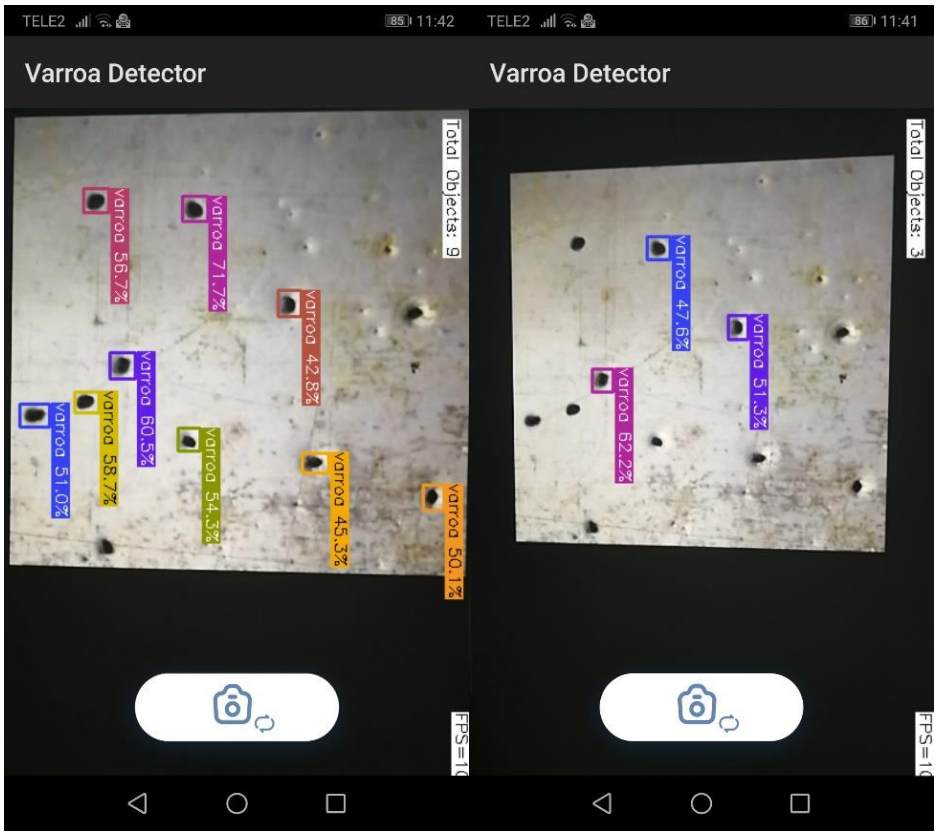
Iz rezultata prikazanih gornjom tablicom vidljivo je da je većina modela uspješno usavršilo otkrivanje klase varoa osim Faster-RCNN i SDD modela. Razlog tomu je isključivo rad algoritma. Oba algoritma u usporedbi sa YOLO modelima imaju loše rezultate iz razloga što je u YOLO algoritmima implementiran FPN i SPP. Ove dvije nadogradnje algoritma pomažu modelima da bolje detektiraju objekte raznih razlučivosti. FPN u konvolucijskom sloju tako što na ulazu uzima sliku u jednom mjerilu proizvoljne veličine i daje mape značajki proporcionalne veličine na više razina, na potpuno konvolucijski način. SPP je sloj sažimanja koji zamjenjuje zadnji sloj sažimanja koji se obično nalazi u konvolucijskom sloju neposredno prije potpuno povezanog sloja. Taj sloj uklanja ograničenje fiksne veličine mreže odnosno konvolucijska mreža nije ograničena samo jednom prostornom veličinom ulaznog podatka. Iz tog razloga moguće je zaključiti da SSD i Faster-RCNN ne pokazuju dobre rezultate jer koriste mape značajki koje su male razlučivosti pa značajke manjih objekata postanu premale da bi se objekt mogao detektirati. Svi modeli su u prosjeku naučeni u manje od 5 sati dok je učenje SSD modela trajalo gotovo 48 sati.

5.2. Ispitivanje aplikacija

5.2.1. Ispitivanje Android aplikacije

Ova aplikacija testirana je mobilnom uređaju Huawei P10 čije su specifikacije navedene u tablici 4.3. Testirana je na više od 30 različitih slika koje mreža nije koristila za učenje. Neke su stvorene

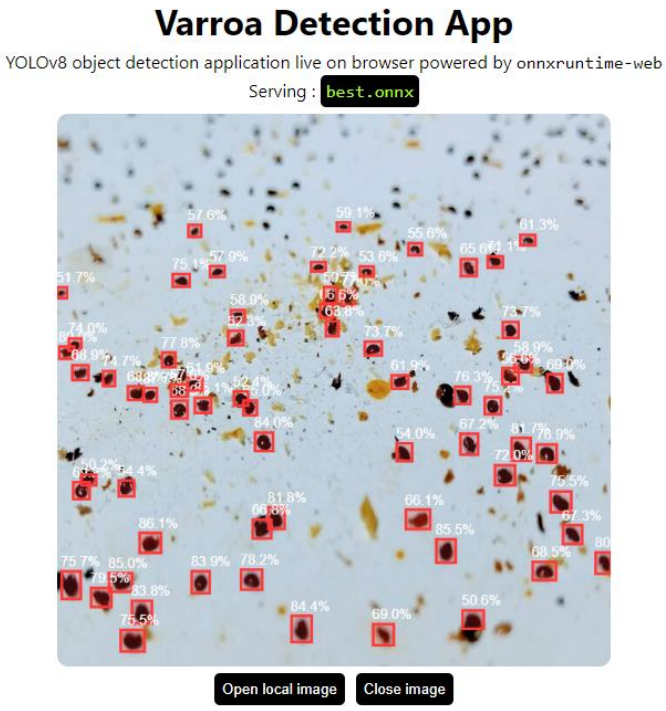
umjetno dok su neke preuzete s interneta. Aplikacija ima problema sa otkrivanjem prilično jednostavnih objekata u određenim uvjetima. Najveći uzrok greškama je udaljenost kamere od objekata kao što je vidljivo na slici 5.1. Prilikom korištenja aplikacije također je vidljivo da je osjetljiva i na najmanje pokrete. U idealnim uvjetima, aplikacija daje relativno očekivane rezultate.



Slika 5.1 *Rezultati detekcije različitih udaljenosti*

5.2.2. Ispitivanje Internet aplikacije

Internet aplikacija pokazuje izrazito dobre rezultate. Testirana na identičnom setu testnih podataka kao i mobilna aplikacija, uredno klasificira sve podatke te je odziv na klasu varoa iznad 95% što znači da greška prilikom otkrivanja objekta vrlo mala. Na slici 5.2 prikazan je rezultat aplikacije na slici koja nimalo ne nalikuje na podatkovni skup za treniranje i pokazuje zadovoljavajuće rezultate.



Slika 5.2 *Rezultati Internet aplikacije nad neviđenim podacima*

5.2.3. Usporedba dobivenih rezultata

Sljedeća tablica 5.5 prikazuje rezultate detekcije grinje varoe na pet različitih slika te rezultate dobivene korištenjem obje aplikacije. Prvi stupac prikazuje koliko je stvarnih objekata bilo na slikama na kojima modeli nisu bili učeni dok preostala dva stupca prikazuju rezultate detekcije aplikacija.

Tablica 5.5 *Usporedba performansi aplikacija*

Broj objekata	**Android aplikacija**	**Internet aplikacija**
5	4	5
7	5	7
12	8-9	11
8	6	8
14	9	14

Iz tablice je vidljivo da Internet aplikacija radi prilično dobro te da u usporedbi sa prvotno naučenim modelom nimalo ne zaostaje. Daljnjim testiranjem utvrđeno je da su rezultati Internet aplikacije i treniranog modela prilično jednaki. Obzirom da je Internet aplikacija prilagođena za rad na računalu ti su rezultati očekivani, no optimizacijom modela odnosno izmjenom prijenosnog

formata modelu su parametri smanjeni, te su težinske vrijednosti smanjene sa float 32 na float 16 broja s pomičnim zarezom što nije previše utjecalo na rad modela.

Iako su rezultati nešto lošiji na mobilnoj aplikaciji iz tablice je također vidljivo da Android aplikacija ima relativno dobre rezultate obzirom da je testirana na starijem uređaju. Na ove rezultate također dodatno utječu nekoliko faktora, primjerice model jako loše reagira na pomičnost uređaja. Time se unose smetnje poput zamućivanja slike, te obzirom na brzinu pomicanja mobilnog uređaja dolazi i do gubitka okvira slika.

5.3. Moguća poboljšanja sustava

Testiranjem rada sustava utvrđeno je da su u nekim slučajima izmjene moguće i poželjne dok je u nekim slučajima izmjena ili poboljšanje nužno. Mobilna aplikacija je u ovom radu imala najviše problema sa detekcijom objekata. Pošto je aplikacija implementirana na način da se računske operacije i analiza obavlja na samome uređaju tako što je integrirana unutar same aplikacije, nemoguće je kontrolirati svaki aspekt koji unosi šum u ulaznu sliku. Bolja kamera definitivno će unaprijediti rezultate detekcije objekata jer će pokrivenost objekta pikselima biti veća, što će algoritam modela iskoristiti bez dodatne implementacije. Također je od velike važnosti da se uređaj ne pomiče prilikom procesa otkrivanja objekta kako se ne bi unosile pogreške, čistina ulazne slike je ključna. Jedna od dodatnih izmjena koju je moguće napraviti unutar mobilne aplikacije je drugačija implementacija koja bi slala slike udaljenom aplikacijskom programskom sučelju (*API*) koji bi zadatak pripreme i obrade podataka izvršio na drugom računalnom uređaju i vratio samo pronađene objekte putem prijenosnog formata poput *JSON* kako bi se na aplikaciji prikazivali samo rezultati.

Iako je u rezultatima vidljivo da su neki modeli prilično dobro prihvatili ovaj skup podataka i pokazali odlične rezultate, određeni algoritmi su previše osjetljivi na jako male objekte. Novi skup podataka koji bolje opisuje morfologiju varoe bi svakako pridonio boljoj detekciji. Primjerice SSD model, iako je pokazao lošije rezultate, prednost korištenja ovog algoritma je brzo izvršavanje zadatka otkrivanja objekata koji bi se mogao iskoristiti ukoliko podatkovni skup bolje opisuje morfologiju grinje varoa. Ukoliko bi se novim podacima kreirao bolji i precizniji model, moguće je implementirati manji sustav čija je kamera postavljena na ulaz košnice te model u stvarnom vremenu pregledava pčele koje ulaze i izlaze i košnice. Prilikom detekcije, šalje se obavijest korisniku.

6. ZAKLJUČAK

Zadatak ovog diplomskog rada bila je implementacija sustava koja otkriva grinju varoa unutar dane slike što je i realizirano. Kroz detaljnu analizu znanstvenih članaka i istraživača unutar zajednice koja se bavi ovim područjem otkriveno je nekoliko načina unaprjeđenja detekcije sitnih objekata kroz poboljšanu implementaciju izvlačenja značajki poput FPN metode u konvolucijskom sloju te metode SPP u sloju sažimanja. Iako je u konvolucijskim mrežama poznat pojam crne kutije, odnosno problematično je točno odrediti na koji točno način naučena mreža detektira objekte, sa sigurnošću se može reći da metode FPN u konvolucijskom sloju te SPP u posljednjem sloju sažimanja uvelike pomažu otkrivanju značajki sitnih objekata jer tim metodama rješavaju problem osjetljivosti algoritma u nižim prostornim dimenzijama. Implementirano je i uspoređeno nekoliko algoritama pretrage objekata unutar slika. To su sedam algoritama iz YOLO obitelji te SSD i Faster-RCNN algoritam. Za realizaciju ovog sustava implementirane su mobilna i Internet aplikacija čiji rezultati imaju relativna odstupanja.

Usporedbom rezultata evaluacije detekcije objekata navedenih algoritama uočena je zamjetna razlika između algoritama YOLO obitelji sa preostala dva algoritma, SSD i Faster-RCNN, gdje je udio točno klasificiranih instanci u modelima SSD i Fast-RCNN ispod 45% dok je u YOLO slučaju rezultat veći od 90%. Inačice 3 i 5 YOLO algoritma pokazuju sličnosti sa rezultatima dobivenih testiranjem posljednje i najnovije inačice 8 što ih također čini izvrsnim kandidatima za nastavak implementacije sustava. Pošto je YOLOv8 inačica najnovija implementacija ovog algoritma te posjeduje navedene metode, odabrana je za realizaciju sustava ovog rada. Usporedbom rezultata detekcije objekata Android i Internet aplikacije nad testnim skupom stvarnih situacija primijećeno je da Android aplikacija ostvaruje lošije rezultate detekcije. Razlog je sklonost mobilne aplikacije na vanjske faktore poput prostorne dimenzije slike zabilježena kamerom uređaja, pomicanjem uređaja dolazi do povećanja grešaka, te udaljenost uređaja od samih objekata uvelike doprinosi povećanju grešaka modela pri otkrivanju objekata. Poboljšanje sustava može se postići realizacijom sustava koji je naučen sa znatno većim podatkovnim skupom, nepomični uređaj koji bilježi veće prostorne dimenzije slike. Ovaj rad rezultirao je uspješno realiziranim sustavom za detekciju grinje varoa te može poslužiti kao osnova za daljnji rad na poboljšanju rezultata i implementacijom na računalno jačim uređajima u svrhu bržeg i jednostavnijeg održavanja zdravlja pčelinje zajednice.

LITERATURA

[1] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.

[2] O. Theobald, Machine Learning for Absolute Beginners:A Plain English Introduction (Second Edition), nezavisno objavljeno 2017.

[3] V. Maini, S. Sabri, Machine Learning for Humans, 2017., dostupno na: https://everythingcomputerscience.com/books/Machine%20Learning%20for%20Humans.pdf [4.6.2023.]

[4] Y. Bai, Y. Zhang, M. Ding, B. Ghanem, SOD-MTGAN: Small Object Detection via Multi-Task Generative Adversarial Network, Proceedings of the European Conference on Computer Vision (ECCV), sv. 11217, str. 206-221, rujan 2018.

[5] C.F. Sabottke, B.M. Spieler, The Effect of Image Resolution on Deep Learning in Radiography, Radiology: Artificial Intelligence, br. 1, sv. 2, str. 190015-190022, siječanj 2020.

[6] Machine Learning Mastery, Basic Concepts in Machine Learning, 2015., dostupno na:

https://machinelearningmastery.com/basic-concepts-in-machine-learning/ [4.6.2022.]

[7] CloudTweaks, INFOGRAPHIC: HOW MUCH DATA IS PRODUCED EVERY DAY?,

2009., dostupno na: https://cloudtweaks.com/2015/03/how-much-data-is-produced-every-day [5.6.2022.]

[8] BeeAware, beeaware.org.au, dostupno na: https://beeaware.org.au/archive-pest/varroa-mites/ [7.6.2022.]

[9] Machine Learning Mastery, machinelearningmastery.com, dostupno na: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/ [7.6.2022.]

[10] Top Lab – VarroaCounter, varroacounter.com, dostupno na: https://www.varroacounter.com/ [10.6.2022.]

[11] The Python wiki, FrontPage , dostupno na: https://wiki.python.org/moin/FrontPage [10.6.2022.]

[12] RealPython, jupyter notebook, dostupno na: https://realpython.com/jupyter-notebook-introduction/ [10.6.2022.]

[13] Google Colaboratory, google research, dostupno na: https://research.google.com/colaboratory/faq.html [11.6.2022.]

[14] Keras, About Keras, dostupno na: https://keras.io/about/ [4.6.2022.]

[15] Liu, B.Y. Chen, H.X. Huang, Z. Liu, X. Yang, Y.Z. ZoomInNet: A Novel Small Object Detector in Drone Images with Cross-Scale Knowledge Distillation, Remote Sens., br. 2021, sv. 13, str. 1198-1219., 21. ožujak 2021.

[16] N.D. Nguyen, T. Do, T. Ngo, D. Le, An Evaluation of Deep Learning Methods for Small Object Detection, Journal of Electrical and Computer Engineering, sv. 2020, str. 3189691-3189709, 27. travanj 2020.

[17] X . Wang, D. Zhu, Y. Yan, Towards Efficient Detection for Small Objects via Attention-Guided Detection Network and Data Augmentation, Sensors, sv. 22, str. 7663-7675, 9. listopad 2022.

[18] Saeed, Faisal, Ahmed, Muhammad Jamal, Gul, M.J.. A robust approach for industrial small-object detection using an improved faster regional convolutional neural network. Scientific Reports, br. 23390, sv. 11, str. 23390-23403, prosinac 2021.

[19] R. Russell, Machine Learning:Step-By-Step Guide to Implement Machine Learning Algorithms with Python, CreateSpace Independent Publishing Platform, Scotts Valley, California, 2018.

[20] A. Burkov, The hundred-page machine learning book, nezavisno objavljeno, 2019. dostupno na:http://ema.cri-info.cm/wp-content/uploads/2019/07/2019BurkovTheHundred-pageMachineLearning.pdf [4.6.2023.]

[21] Michael Nielsen, Neural Networks and Deep Learning, Determination press San Francisco, CA, USA, 2015

[22] Keith D. Foote,Deep learning history, 2022, https://www.dataversity.net/brief-history-deep-learning/ [12.6.2022.]

[23] Q. Yang, Y. Zhang, W. Dai, S. J. Pan, Transfer Learning, Cambridge University Press, Cambridge, United Kingdom 2020

[24] Top Lab–VarroaCounter, varroacounter.com, dostupno na: https://www.varroacounter.com/ [14.6.2022.]

[25] Git repozitorij tensorflow modela, dostupno na: https://github.com/tensorflow/models [3.7.2022.]

[26] Git repozitorij YOLO modela, dostupno na: https://github.com/ultralytics [4.7.2022.]

[27] Git repositorij YOLO/NCNN implementacija modela, dostupno na: https://github.com/Tencent/ncnn [10.7.2022.]

SAŽETAK

Cilj ovog diplomskog rada je izrada sustava za detekciju grinje varoa koja može poslužiti pčelarima za davanje upozorenja u ranijim fazama pojave grinje u svrhu pravovremenog i odgovarajućeg tretiranja. U teorijskom dijelu ovog rada prikazana je morfologija grinje te je kao problem navedena detekcija sitnih objekata i moguća rješenja poput povećanja prostorne razlučivosti slike i segmentacija fotografije. Nadalje, objašnjen je način rada neuronskih mreža, objašnjena načela prijenosnog učenja te definirane mjere vrednovanja modela. U praktičnom dijelu rada ostvareno je programsko rješenje za detekciju grinje varoa. Programsko rješenje implementirano je na mobilnoj i Internet platformi te se za unos podataka na Android platformi koristi kamera mobitela, vrši se detekcija i prikazuje korisniku. Za Internet aplikaciju potrebno je imati pripremljene fotografije koje se učitaju i rezultat se ispisuje. U implementaciji aplikacija koristio se YOLOv8 model koji je postigao najbolje rezultate. Rezultati ispitivanja sustava ukazuju na dodatan rad mobilne aplikacije kako bi se sustav usavršio.

Ključne riječi: detekcija objekata, grinja varoa, konvolucijske neuronske mreže, prijenosno strojno učenje

ABSTRACT

The purpose of this paper is to develop a system that detects varroa mites which can be used by beekeepers as early warning in the earlier stages of appearance of the mite for the purpose of timely and appropriate treatment. In the theoretical part of this paper the morphology of the mite is presented, and as a problem, detection of small objects was stated and possible solutions such as increasing the spatial resolution of the image and image segmentation. Furthermore, the way how neural networks operate is explained, the principles of transfer learning are explained and the evaluation measures of the model are defined. In the practical part of the paper, a software solution for the detection of the varroa mite was implemented. The software solution is implemented on the mobile and internet platforms, and the mobile camera is used for data input on Android platform, afterwards detection is applied and shown to user. For the web application, it is necessary to have preprocessed images, so when images are uploaded the results can be shown. In the implementation of the application, YOLOv8 model was used, model that had achieved the best performance. The results of the system indicate that additional work of the mobile application is needed in order to improve the system

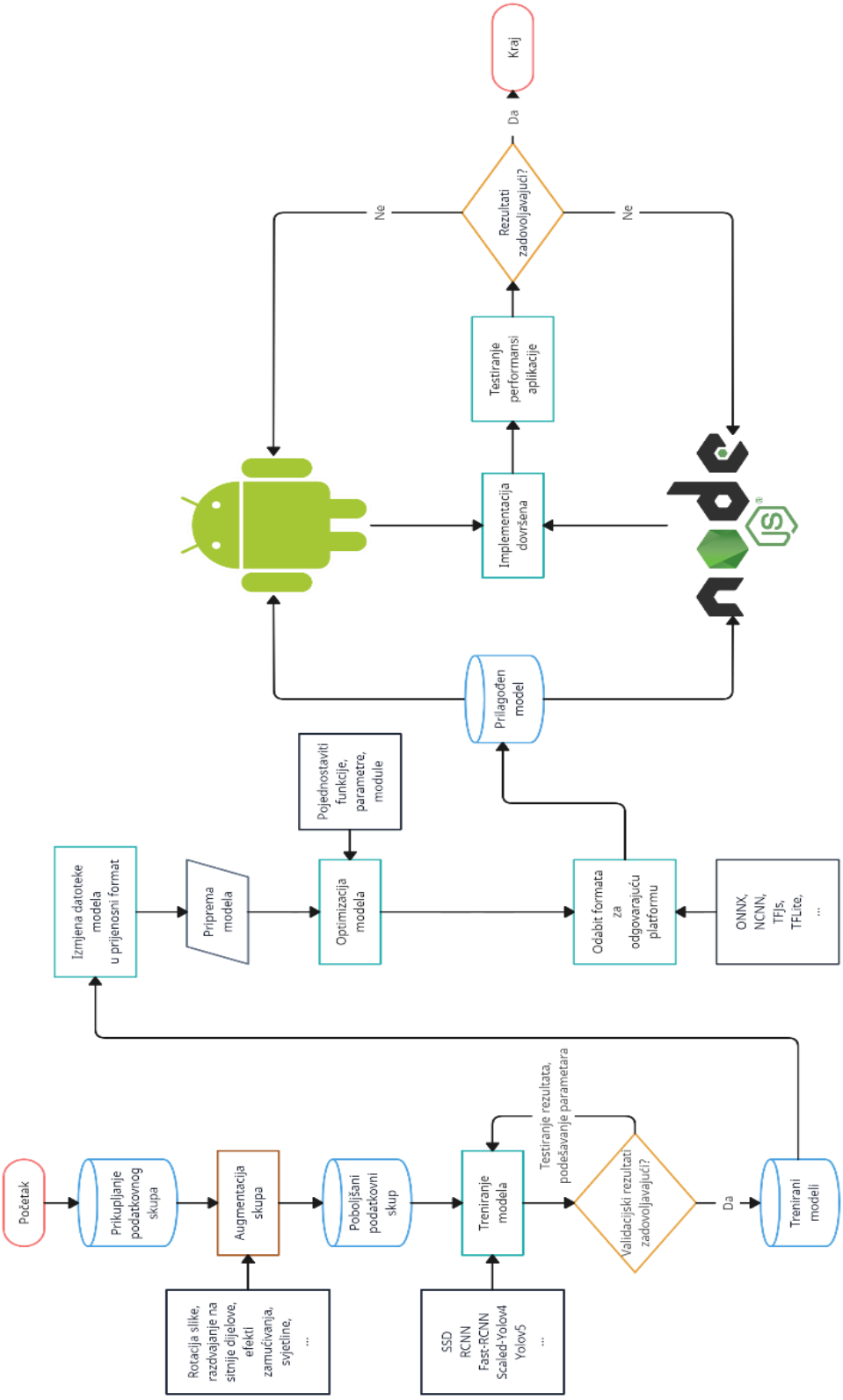
Keywords: object detection, varroa mite, convolution neural network, transfer machine learning

ŽIVOTOPIS

Tomislav Ćurić rođen je 05.02.1993. godine u Našicama. Osnovnu školu pohađa u Donjem Miholjcu, a nakon završene osnovne škole upisuje srednju Poljoprivrednu školu u Donjem Miholjcu. Srednjoškolsko obrazovanje završava 2011. godine te odlazi raditi u inozemstvo u području gastronomije. Upisuje stručni studij Informatika na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija. 2012. godine. Apsolventsku godinu upisuje 2015. godine te se zapošljava u tvrtki StakloDom gdje radi kao administrator Internet stranice. Nakon završenog stručnog studija i razlikovnih obveza 2017. godine upisuje diplomski studij smjer Informacijske i podatkovne znanosti. Od 2019. godine skrbi o bolesnoj majci i održava obiteljsko gospodarstvo.

PRILOZI

1. Dokument diplomskog rada u .docx formatu
2. Dokument diplomskog rada u .pdf formatu
3. Programski kodovi za treniranje modela unutar okruženja *Jupiter notebook*
4. Programski kod Android aplikacije
5. Programski kod web aplikacije
6. Dijagram tijeka implementacije sustava:



P.4.1 *Dijagram tijeka izrade sustava*