Mobilna Android aplikacija za potporu zaštiti nasada jabuka klasifikacijskim postupcima strojnog učenja

Rizner, Josip

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

Sveučilišni preddiplomski studij

MOBILNA ANDROID APLIKACIJA ZA POTPORU ZAŠTITI NASADA JABUKA KLASIFIKACIJSKIM POSTUPCIMA STROJNOG UČENJA

Diplomski rad

Josip Rizner

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 13.09.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Josip Rizner
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1243R, 08.10.2021.
OIB studenta:	79111943693
Mentor:	prof. dr. sc. Goran Martinović
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 1:	prof. dr. sc. Goran Martinović
Član Povjerenstva 2:	izv. prof. dr. sc. Ivan Aleksi
Naslov diplomskog rada:	Mobilna Android aplikacija za potporu zaštiti nasada jabuka klasifikacijskim postupcima strojnog učenja
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U teorijskom dijelu diplomskog rada treba analizirati probleme i izazove u zaštiti biljaka s naglaskom na voćarske kulture i bolesti vidljive na listu. Uzimajući u obzir postojeća slična programska rješenje, mogućnosti klasifikacijskih postupaka strojnog učenja u prepoznavanju slika, te načine liječenja i praćenje liječenja biljaka, potrebno je definirati funkcionalne i nefunkcionalne zahtjeve na mobilnu aplikaciju, predložiti model i arhitekturu aplikacije na strani korisnika i poslužitelja. Također, koristeći dostupne skupove podataka sa slikama
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	13.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 25.09.2023.

Ime i prezime studenta:	Josip Rizner
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1243R, 08.10.2021.
Turnitin podudaranje [%]:	15

Ovom izjavom izjavljujem da je rad pod nazivom**: Mobilna Android aplikacija za potporu zaštiti nasada jabuka klasifikacijskim postupcima strojnog učenja**

izrađen pod vodstvom mentora prof. dr. sc. Goran Martinović

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. UVOD

Voćarstvo, posebno uzgoj jabuka, vrlo su bitni u svjetskoj poljoprivredi, pružajući hranu i ekonomske resurse milijunima ljudi diljem svijeta. Međutim, nasadi se suočavaju s nizom ozbiljnih bolesti koje predstavljaju znatnu prijetnju njihovom zdravlju i produktivnosti. Bolesti u voćarstvu predstavljaju ozbiljnu problematiku koja zahtijeva pažljivu pozornost i brzu reakciju. Različite bolesti, kao što su krastavost jabuke, hrđa jabuke, pepelnica i mrljavost lista mogu uništiti cijele nasade jabuka ako se ne prepoznaju i ne kontroliraju na vrijeme. Posljedice takvih infekcija uključuju smanjenje prinosa, smanjenje kvalitete plodova te dugoročno oštećenje stabala jabuke. Iz tog razloga, prevencija i liječenje bolesti u voćarstvu iznimno su važni za očuvanje ekonomske održivosti voćnjaka i osiguranje kvalitetne ponude voća na tržištu.

Cilj ovog diplomskog rada je analizirati i opisati mogućnosti primjene strojnog učenja i mobilnih aplikacija za detekciju bolesti i zaštitu nasada jabuka. Isto tako u radu će se prikazati postojeća slična rješenja sustava za klasifikaciju bolesti lista jabuke. Dodatno, razvit će se model s parametrima značajnima za bolesti lista jabuke, razraditi postupak analize podataka i predložiti model mobilne aplikacije. Uz to, opisat će se korišteno razvojno okruženje za mobilne aplikacije kao i programske tehnologije. Osnovni zadatak je razviti mobilnu aplikaciju koja će omogućiti unos slike lista jabuke na osnovu čega će koristeći strojno učenje prepoznati o kojoj se bolesti radi te će se dati preporuke za liječenje. Programsko rješenje će se ispitati i analizirati za odgovarajuće ulazne podatke.

Drugo poglavlje prikazuje problematiku bolesti u voćarstvu i stanje u području. U trećem poglavlju predstavljen je konceptualni model mobilne Android aplikacije te klasifikacijski model za prepoznavanje bolesti. U četvrtom poglavlju detaljno će se objasniti programsko rješenje mobilne Android aplikacije, postupak treniranja klasifikacijskog modela te korišteni programski alati i okoline. Na kraju, u petom poglavlju ispitat će se ispravnost programskog rješenja i valjanost klasifikacijskih postupaka za prepoznavanje bolesti jabuka, te analizirati rezultati ispitivanja.

2. BOLESTI U VOĆARTVU I PRIKAZ STANJA U PODRUČJU

U ovom poglavlju opisat će se problematika bolesti u voćarstvu, opisat će se neke od najčešćih bolesti i prikazati će se preporuke za liječenje tih bolesti. Isto tako, prikazati će se neka od postojećih rješenja sa prepoznavanje bolesti u voćarstvu.

2.1. Važnost prevencije i liječenja bolesti

Prevencija i liječenje bolesti u voćarstvu izuzetno su važni iz nekoliko razloga. Prvi od njih je očuvanje prinosa. Kako je opisano u [1], sigurnost hrane jedna je od glavnih globalnih briga. Ukupna opskrba hranom sama po sebi ne ispunjava zahtjeve sigurnosti hrane jer ljudsko zdravlje i društvena stabilnost temelje se na hrani koja je sigurna, hranjiva, pristupačna i dostupna u različitim oblicima. Bolesti mogu ozbiljno utjecati na zdravlje voćnih stabala, smanjujući njihovu vitalnost, produktivnost i konačni prinos. Redovita prevencija i liječenje bolesti pomažu u održavanju zdravih stabala, što rezultira većim i kvalitetnijim prinosima voća. Drugi razlog je ekonomski faktor. Bolesti u voćarstvu mogu prouzročiti značajne gubitke za voćare. Inficirana stabla mogu propasti ili imati smanjen prinos, što dovodi do financijskih gubitaka. Prevencija i liječenje bolesti smanjuju rizik od gubitaka i osiguravaju stabilnost i profitabilnost voćarskih operacija. Nadalje, prevencija i liječenje bolesti bitno je zbog održavanja kvalitete voća. Bolesti mogu utjecati na kvalitetu voća, uključujući izgled, okus, teksturu i nutritivnu vrijednost. Inficirano voće može biti manje privlačno kupcima, smanjujući vrijednost i konkurentnost na tržištu. Prevencija bolesti osigurava da voće ostaje zdravo, privlačno i visokokvalitetno. Isto tako, jedan od vrlo bitnih razloga je zaštita okoliša. Nepravilno upravljanje bolestima može dovesti do prekomjerne upotrebe pesticida i drugih kemijskih tretmana, što može imati negativan utjecaj na okoliš. Pravilna prevencija i liječenje bolesti u voćarstvu mogu smanjiti potrebu za pesticidima, očuvati prirodnu ravnotežu i smanjiti negativne utjecaje na okoliš. Još jedan razlog koji valja istaknuti je sprječavanje širenja bolesti. Bolesti u voćarstvu mogu se brzo širiti među stablima, ali i na susjedne voćnjake. Ako se ne poduzmu odgovarajuće mjere prevencije i liječenja, bolesti se mogu proširiti na velika područja, uzrokujući epidemije koje je vrlo teško kontrolirati. Pravilno upravljanje bolestima sprječava širenje i štiti cjelokupnu voćarsku industriju.

Naravno, kod prevencije i liječenja bolesti nailazi se na razne prepreke. U [2] se navodi kako se, općenito u poljoprivredi, upravljanje biljnim bolestima suočava sa sve većim izazovima. Neki od razloga su:

- Sve veće potražnje za ukupnom, sigurnom i raznolikom hranom radi podrške rastućoj globalnoj populaciji i poboljšanju životnih standarda
- Smanjenja proizvodnog potencijala u poljoprivredi zbog konkurencije za zemljište u plodnim područjima i iscrpljenosti marginalnih obradivih površina
- Pogoršanja ekologije agroekosustava i iscrpljivanja prirodnih resursa
- Povećanog rizika od epidemija bolesti uzrokovanog intenziviranjem poljoprivrede i monokultura

Ukratko, prevencija i liječenje bolesti u voćarstvu su ključni za održavanje zdravlja stabala, visok prinos voća, ekonomsku profitabilnost, očuvanje okoliša i zaštitu cjelokupne voćarske industrije. Ranim otkrivanjem, praćenjem zdravlja voćaka i primjenom odgovarajućih mjera prevencije i kontrole, doprinosi se suzbijanju ovih bolesti i očuvanju zdravlja biljaka.

2.2. Različite bolesti voćaka i biljaka

Postoji puno različitih bolesti u voćarstvu koje variraju ovisno o vrsti voća i geografskom području. U ovom radu naglasak se uzima na nasade jabuka te najčešće bolesti koje se pojavljuju na istima. Neke od najčešćih bolesti su mrljavost lista jabuke, krastavost jabuke, hrđa jabuke te pepelnica jabuke.

2.2.1. Mrljavost lista jabuke

Kako je navedeno u [3], mrljavost lista jabuke (engl. *frogeye leaf spot*) je bolest koja se može pojaviti na listovima jabuka, a uzrokovana je gljivicom *Diplocarpon mali*. To je gljivična infekcija koja uzrokuje pojavu malih tamnih mrlja na listovima jabuke. Ove mrlje mogu biti smeđe ili crne s laganim svjetlijim središtem, što im daje izgled sličan oku, zbog čega je bolest dobila ime "*Frogeye*". Ova bolest može utjecati na sposobnost biljke da efikasno obavlja fotosintezu, što može smanjiti prinos jabuka. Kako se infekcija širi, mrlje se mogu spajati i uzrokovati veće oštećenje lišća. Pravilno upravljanje sadnjom, održavanjem i primjenom odgovarajućih fungicida može pomoći u suzbijanju širenja ove bolesti kako bi se očuvala zdravlje jabuka. Na slici 2.1 može se vidjeti kako izgleda mrljavost lista jabuke.

Slika 2.1. *Mrljavost lista jabuke*

2.2.2. Krastavost jabuke

Krastavost jabuke (engl. *scab*) jedna je od najpoznatijih gljivičnih bolesti jabuke. Ovu bolest uzrokuje gljiva *Venturia ineaqualis*. Kako je navedeno u [4], ova gljiva zimu provodi u otpalim listovima nakon čega, u proljeće, oslobađa veliku količinu spora. Stabla jabuke su obično najosjetljivija na ovu bolest kada tek prolistaju. U kontinentalnim područjima Hrvatske najpovoljniji uvjeti za zarazu su obično krajem travnja. Tada su količine spora najveće, ali još jedan razlog je kiša koja vlaži lišće. Još jedan faktor koji doprinosi opasnosti od zaraze su rose i magle. Krastavost jabuke vrlo je problematična jer pojava ove bolesti uzrokuje smanjenje kvalitete i količine uroda. U [5] je navedeno kako u povoljnim uvjetima, ova bolest može rezultirati gubitkom 70% ili više uroda. Razlog tome su mrlje, deformacije i šupljine koje jabuke čine neprikladnima za prodaju. Dodatno, ova bolest smanjuje rast i prinos biljke. To se najčešće događa kada krastavost uzrokuje ponovljeno opadanje lišća s drveća tijekom nekoliko sezona. Na slici 2.2 može se vidjeti kako izgleda krastavost ploda i lista jabuke. Zaštita od krastavosti primjenjuje se na različite načine u različitim nasadima. U određenim nasadima i područjima, zaštita se provodi prema prognozi koja se temelji na praćenju prisutnosti spora, vlažnosti biljnih organa, oborinama i temperaturi. Međutim, u drugim nasadima, proizvođači se orijentiraju prema fenofazama. Kao i kod ostalih gljivičnih bolesti, najbolje je djelovati preventivno i pokušati spriječiti zarazu. Ukoliko prognozni modeli nisu dostupni, stabla bi se trebala tretirati prije svake kiše ili neposredno nakon nje.

Slika 2.2. *Krastavost ploda i lista jabuke*

2.2.3. Hrđa jabuke

Hrđa jabuke (engl. Apple Rust), gljivična je bolest koja utječe na jabuke i druge biljke iz porodice *Rosaceae*, kao i na četinare. Ova bolest uzrokuje karakteristične hrđave mrlje na lišću, plodovima te ponekad granama domaćina. Gljive iz roda *Gymnosporangium* odgovorne su za

ovu bolest. Infekcija počinje kada spore gljivica prenesene vjetrom dospiju s četinara na lišće i plodove jabuke. Na jabukama se formiraju karakteristične hrđave mrlje koje oslobađaju nove spore. Ove spore mogu ponovno inficirati četinare, završavajući ciklus. U [6] je navedeno kako je visoka vlažnost i temperatura između 15-20°C optimalno vrijeme kako bi se bolest širila. Tada je potrebno šest do osam sati kako bi se spora razvila i zarazila domaćina. Simptomi hrđe jabuke uključuju:

- Hrđave mrlje: Na gornjoj strani lišća jabuke pojavljuju se hrđave mrlje koje mogu varirati u veličini i intenzitetu.
- Izrasline: Na donjoj strani lišća i ponekad na plodovima mogu se razviti izrasline koje oslobađaju spore u vlažnim uvjetima.
- Deformaciju lišću: Inficirano lišće može se deformirati i sušiti.
- Oštećene plodove: Plodovi također mogu razviti hrđave mrlje i deformirati se, što može smanjiti kvalitetu i tržišnu vrijednost.

Kontrola hrđe jabuke uključuje korištenje otpornih sorti jabuka, održavanje higijene voćnjaka, redovito uklanjanje zaraženih dijelova biljaka, primjenu fungicida te, ako je moguće, smanjenje prisustva četinara u blizini voćnjaka kako bi se prekinuo životni ciklus gljivica. Na slici 2.3 može se vidjeti kako izgleda hrđa jabuke.

Slika 2.3. *Hrđa jabuke*

2.2.4. Pepelnica jabuke

Kako se navodi u [7] i [8], pepelnica jabuke najučestalija je bolest jabuka nakon krastavosti jabuke. Ovu bolest uzrokuje gljiva *Podosphaera leucotricha*. Pepelnica se vrlo lako prepoznaje, površinski dijelovi su prekriveni pepelastobijelom prevlakom i lagana prašina se osipa nakon

protresanja lišća. Pojavi pepelnice jabuke pogoduju blage zime, suha i topla proljeća, intenzivna gnojidba, osjetljivost sorti te suvremene metode uzgoja. U usporedbi s krastavosti, pepelnica jabuke se češće javlja u toplijim, manje kišnim godinama te na sunčanijim, manje vlažnim i toplijim područjima. Ova bolest prezimljuje u zaraženim pupovima, najčešće vršnim pupovima lišća i cvijeta. Kada voćke proklijaju, pepelnica jabuke se aktivira i brzo prekriva najmlađe dijelove biljke, kako vegetativne tako i generativne organe. Zaraženi pupovi sprječavaju normalan razvoj mladica, što rezultira kraćim internodijima i slabije razvijenim, deformiranim lišćem koje ostaje u rozeti na vrhu biljke. Na slici 2.4 može se vidjeti kako izgleda list zaražen pepelnicom.

Slika 2.4. *Pepelnica jabuke*

Kako je opisano u izvoru [9], pepelnica jabuke je vrlo lako primjetna pa je važno odmah poduzeti mjere za suzbijanje ove bolesti, uključujući uklanjanje zaraženih mladica i listova putem rezidbe. Pepelnica se može pojaviti tijekom vegetacije bez obzira na količinu oborina, što znači da se može javiti i u sušnim razdobljima. Budući da nije moguće potpuno iskorijeniti pepelnicu samo jednim prskanjem, suzbijanje se obično kombinira s mjerama zaštite od drugih bolesti. Ključno je započeti s mjere suzbijanja pepelnice prije same cvatnje biljke. Program zaštite od pepelnice kemijskim sredstvima sastoji se od minimalnog broja prskanja, koja su usklađena s mjerama zaštite od krastavosti ili crvljivosti. U preventivne svrhe koriste se određeni pripravci, a kad se bolest već pojavi, primjenjuju se razni sustavski pripravci za liječenje.

2.3. Preporuke za liječenje bolesti u voćarstvu

U većini izvora preporuča se djelovati preventivno, odnosno spriječiti razvoj bilo koje bolesti. Neke od preventivnih mjera za zaštitu od svih bolesti, ali posebno od gljivičnih koje su fokus

ovog rada su: pravilna higijena voćnjaka, prozračivanje voćnjaka i pravilno razrjeđivanje plodova. Pod pravilnom higijenom voćnjaka smatra se redovito uklanjanje lišća, trulih plodova te drugih biljnih ostataka iz voćnjaka. To je izrazito bitno jer su takvi biljni ostatci stvaraju povoljne uvjete za preživljavanje i razvoj gljiva koje uzrokuju bolesti. Uklanjanjem biljnih ostataka smanjuje se izvor infekcije. Prozračivanjem voćnjaka osigurava se dobra cirkulacija zraka između stabala kako bi se smanjila vlažnost. Manja vlažnost ne odgovara gljivama pa se tako sprječava razvoj bolesti. Dodatno, pravilno raspoređivanje plodova također može pomoći pri prevenciji bolesti u voćarstvu. Gusta krošnja i pretrpani plodovi mogu stvarati povoljne uvjete za razvoj bolesti. Stoga, redovito razrjeđivanje plodova može pomoći u sprječavanju bolesti, a posebno pomaže kod sprečavanja bolesti poput krastavosti i mrljavosti. Nadalje, dodatne mjere zaštite podrazumijevaju pravilnu gnojidbu koja može osigurati da voćnjak dobije potrebne hranjive tvari kako bi biljke bile otpornije na bolesti. Odabir sorti je također vrlo bitan, treba razmotriti uzgoj sorti koje su otporne na specifične bolesti koje su problem u određenom području. Isto tako, vrlo je bitno redovito praćenje voćnjaka kako bi se bilo kakvi simptomi brzo primijetili te kako bi se na vrijeme poduzele mjere kontrole.

Nažalost, prevencija bolesti nije uvijek moguća. Kada je voćnjak već zaražen, potrebno je koristiti razna sredstva, odnosno fungicide u ovom slučaju, za suzbijanje bolesti. Pri korištenju fungicida, važno je strogo slijediti upute proizvođača o doziranju, razmacima između tretiranja i vremenskim uvjetima za primjenu. Također je važno razmotriti moguće negativne utjecaje na okoliš i pratiti zakonske regulacije i smjernice za upotrebu pesticida u vašem području.

Kako je već spomenuto, krastavost jabuke jedna je od najčešćih bolesti lista i ploda jabuke. U [4] je navedeno kako se za liječenje krastavosti jabuke najčešće koriste fungicidi na bazi propikonazola te fungicidi na bazi kaptana. Kao i kod mrljavosti, fungicidi na bazi bakra mogu pomoći pri suzbijanju ove bolesti. Neka od sredstava koja se koriste za liječenje krastavosti jabuke su Kastor, Argo, Bellis, Chromosul 80 i druga sredstva.

Za kontrolu mrljavosti lista jabuke najčešće se koriste fungicida na azoksistrobina i difenkonazola, ali fungicidi na bazi bakra također mogu pomoći pri suzbijanju mrljavosti. Neki od pripravaka koji se koriste za liječenje mrljavosti lista jabuke su Delan 700 WDG, Polyram DF, Captan 50, Folpan 50 WP i drugi.

Kako je opisano u [6], za liječenje hrđe jabuke koriste se fungicida na bazi bakra i mankozeba. Najučinkovitija sredstva koja se koriste za liječenje ove bolesti su Strobe, Topaz, Vectra, Tsineba i Poliram.

Kako je navedeno u [9], pepelnicu je nemoguće iskorijeniti samo jednim prskanjem i vrlo je važno početi sa suzbijanjem prije same cvatnje. Ako je moguće, uklanjanje zaraženih mladica i

listova putem rezidbe uvelike pomaže. Za suzbijanje pepelnice jabuke najčešće se koristi sumpor. Postoji različite formulacije sumpora, uključujući mokri i suhi sumpor. Primjeri sredstva koji se koriste za liječenje pepelnice uključuju Chromosul 80, Argo, Bellis i druga sredstva.
Dodatno, vrijeme koje treba proći od prskanja fungicida do prvog kontakta s kišom može biti ključno za učinkovitost sredstava koja se koriste za kontrolu bolesti u voćnjaku. Ovo se odnosi na tzv. razdoblje sušenja nakon primjene fungicida. Trajanje tog razdoblja može varirati ovisno o vrsti fungicida i uvjetima u voćnjaku. Dobra praksa je da primijenite fungicid u razdoblju kada ne očekujete kišu u narednih nekoliko sati ili dana. Tako ćete povećati šanse da fungicid ima dovoljno vremena da se osuši i fiksira na biljkama prije nego što dođe do kiše, a to će povećati njegovu učinkovitost.

2.4. Prikaz postojećih sličnih rješenja

Iako u trgovini *Google Play* nema aplikacije koja prepoznaje bolesti na lišću jabuke te daje preporuke za liječenje istih, mogu se naći aplikacije slične namjene koje prepoznaju bolesti različitih biljaka. Dodatno, može se naći nekoliko različitih znanstvenih radova koji se bave klasifikacijom bolesti lišća raznih biljaka.

2.4.1. Analiza stanja u području

Većina sustava za detekciju i klasifikaciju bolesti biljaka koristi konvolucijske neuronske mreže. Konvolucijske neuronske mreže su posebno dizajnirane za analizu vizualnih podataka, kao što su slike i videozapisi. Također, vrlo su učinkovite u izvlačenju značajki iz slika putem konvolucijskih slojeva pa ih to čini vrlo pogodnima za ovakve sustave. U [10] je opisano kako je identifikacija bolesti lišća jedno od izazovnih područja istraživanja u obradi slika, strojnom učenju i računalnom vidu jer se nailazi na mnogo problema. Na primjer: šum, defokusirane slike, udaljenost i kut uređaja i lista u trenutku slikanja, osvijetljene i još mnogi drugi.
U [11] su trenirana tri modela za prepoznavanje bolesti na lišću i uspoređeni su njihovi rezultati. Korišteni su algoritmi k-najbližih susjeda (engl. *K-Nearest Neighbors*), stroj potpornih vektora (engl. *support vector machines*) te konvolucijske neuronske mreže. Za k-najbližih susjeda dobivena je točnost od 64%, sa strojem potpornih vektora dobivena je točnost od 76% te sa konvolucijskim neuronskim mrežama dobivena je točnost od 96%. Zaključak znanstvenog rada je bio kako su konvolucijske neuronske mreže najbolji odabir prilikom detekcije bolesti na lišću.
Dodatno, u [12] predložen je model konvolucijske neuronske mreže temeljene na regijama s poboljšanom reprezentacijom (engl. *Representation-Enhanced Region-based Convolutional Neural Network (RE-RCNN)*) za ranu detekciju bolesti na lišću jabuke. U ovom znanstvenom radu uspoređen je predložen model s nekim poznatim modelima poput *YOLOv2, YOLOv3* i

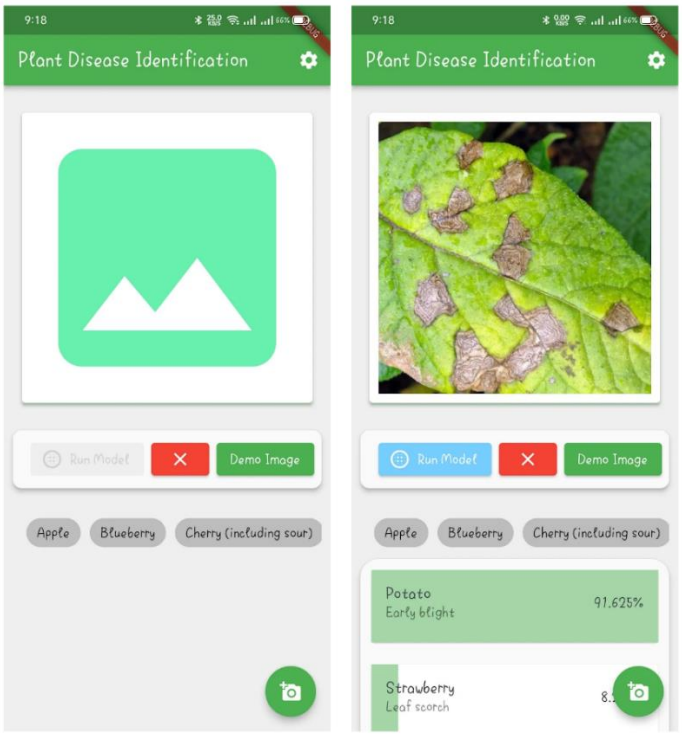
ResNet. U tablici 2.1 mogu se vidjeti rezultati različitih modela prilikom klasifikacije 5 različitih bolesti lista jabuke.

Tablica 2.1. *Usporedba točnosti RE-RCNN modela s ostalim modelima, preuzeto iz [11]*

Models	**backbone**	**Alternaria**	**Brown spot**	**Gray spot**	**Mosaic**	**Rust**
Faster RCNN	**VGGNet-16**	53.01	77.45	55.55	66.27	89.38
Faster RCNN	**ResNet50+FPN**	61.04	85.74	80.69	87.53	64.71
FPN	**VGGNet-16**	54.06	72.90	65.22	76.78	90.09
R-FCN	**ResNet-101**	64.40	77.99	73.47	69.50	90.28
SSD	**VGGNet-16**	57.31	75.55	57.60	77.61	88.26
SSD	**ResNet-101**	51.22	77.47	50.79	73.71	87.90
SSD	**ResNet-101**	59.85	82.00	58.97	52.06	88.67
DSSD	**ResNet-101**	59.85	82.00	58.97	52.06	88.67
YOLOv2	**Darknet-19**	44.88	74.59	50.53	73.68	**94.57**
YOLOv3	**DarkNet-53**	64.48	66.04	62.48	71.31	89.50
VMF-SSD	**VGGNet-16**	64.97	84.93	64.10	79.13	89.11
INSR	**VGG-Incep**	70.11	82.15	68.47	78.78	92.65
RE-RCNN	**ResNet50+FPN**	**70.53**	**86.16**	**83.82**	**90.76**	66.90

2.4.2. Sustav za klasifikaciju bolesti biljaka – Plant Disease Detector

Za primjer sustava koji prepoznaje bolesti lišća biljaka bit će prikazana aplikacija „*Plant Disease Detector*“ [13]. Aplikacija korisniku omogućuje učitavanje slike nakon čega se klasificira bolest lista sa slike. Pri pokretanju aplikacija otvara se početna kartica gdje se može vidjeti gumb za učitavanje fotografije. Pritiskom na gumb otvara se galerija iz koje se može učitati fotografija. Dodatno, na početnom zaslone može se vidjeti koje su sve biljke podržane. Na slici 2.5 može se vidjeti izgled početnog zaslona. Nakon učitavanja slike iz galerije, može se pritisnuti *Run Model* gumb nakon čega se slika predaje modelu te se prikazuju rezultati klasifikacije.



Slika 2.5 *Zaslon aplikacije Plant Disease Detector: Početna kartica te kartica s rezultatima klasifikacije*

3. MODEL I GRAĐA MOBILNE APLIKACIJE

Mobilna aplikacija za potporu zaštiti nasada jabuka klasifikacijskim postupcima strojnog učenja od korisnika prvo zahtjeva da se registrira. Za uspješnu registraciju treba unijeti ime, prezime, e-poštu i lozinku. Nakon što korisnik unese sve podatke, podatci se spremaju u bazu podataka. Baza podataka je realizira u oblaku računala korištenjem *Firebase-a*. Nakon registracije, korisnik mora aktivirati korisnički račun klikom na link koji će mu doći na e-poštu koju je unio prilikom registracije. U slučaju da je korisnik već registriran, prijavljuje se u aplikaciju pomoću e-pošte i lozinke. Nakon uspješne prijave, korisnik je preusmjeren na glavni zaslon aplikacije. Na dnu glavnog zaslona nalazi se izbornik iz kojeg korisnik može birati između četiri kartice. U prvoj kartici korisnik se može upoznati s bolestima nasada jabuka koje su podržane u aplikaciji. U drugoj kartici može učitati sliku koja će se predati unaprijed istreniranom modelu za klasifikaciju bolesti lista jabuke te će se na taj način prepoznati o kojoj se bolesti radi. Isto tako, iz ove kartice može se kreirati novi sustav preporuka za liječenje. Iz treće kartice se može pristupiti svim unaprijed kreiranim sustavima preporuka za liječenje. U četvrtoj kartici može se vidjeti trenutna vremenska prognoza te prognoza za narednih pet dana. Dodatno, vremenska prognoza se može dohvatiti za trenutnu ili za unesenu lokaciju.

3.1. Zahtjevi na mobilnu aplikaciju

3.1.1. Funkcionalni zahtjevi

Aplikacija treba korisniku omogućiti unos slike te analizu iste. Aplikacija treba što preciznije klasificirati bolest lista jabuke i dati prijedlog za liječenje klasificirane bolesti. Prilikom davanja preporuke, aplikacija treba uzeti u obzir vremensku prognozu za lokaciju gdje se nalazi nasad. Dodatno, aplikacija treba omogućiti ponovan unos slike kako bi se utvrdilo ako je tretman uspješan te dati nove smjernice na osnovu tih rezultata. Mobilna aplikacija treba imati registraciju, tj. prijavu korisnika kako bi svaki korisnik mogao pratiti bolesti lišća u svojim nasadima. Isto tako, aplikacija mora imati mogućnost praćenja stanja te davanje preporuka za više nasada. Korisnici također trebaju imati pristup svim zapisima i analizama do tog trenutka.

3.1.2. Nefunkcionalni zahtjevi

Mobilna aplikacija mora biti kompatibilna sa što više mobilnih uređaja koji koriste operacijski sustav Android te mora imati što kraće vrijeme odziva. Na primjer, prilikom pokretanja aplikacije ne bi trebalo proći više od tri sekunde za učitavanje početnog zaslona. Dodatno, vrlo je bitno da se aplikacija pobrinute o trajnosti i integritetu podataka. Također, bitno je napomenuti da svi podatci bilo kojeg korisnika trebaju ostati privatni i skriveni od ostalih korisnika.

3.2. Parametri bolesti i liječenja te preporuke za liječenje

Za prepoznavanje bolesti koristi se unaprijed istrenirana konvolucijska neuronska mrežna. Kada se modelu konvolucijske neuronske mreže preda slika lista jabuke, kao rezultat se dobije vjerojatnost za svaku od bolesti. Dobivene vjerojatnosti se koriste kako bi se odredile preporuke za liječenje. Dodatno, aplikacija omogućuje ponovno učitavanje slike te procjenu učinkovitosti liječenja. Kako bi se procijenila učinkovitost liječenja uspoređuju se vjerojatnosti zdravog lista prilikom prethodne i trenutne klasifikacije.

3.3. Klasifikacijski postupci strojnog učenja

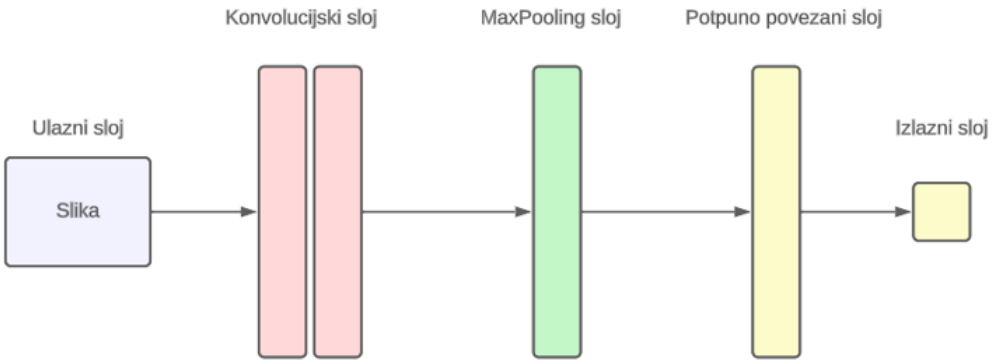
Jedan od najčešćih primjena strojnog učenja je klasifikacija. Zadaća klasifikacije je na temelju analize dovoljne količine podataka, koji su unaprijed podijeljeni na dvije ili više klase, izvesti aproksimaciju funkcije koja klasificirati instance podataka s kojim se prethodno nije susrela. Na primjer, primjenom strojnog učenja na skupu fotografija koje su unaprijed kategorizirane prema njihovom sadržaju, moguće je stvoriti funkciju koja s određenim stupnjem vjerojatnosti može klasificirati slike i identificirati što se nalazi na njima. Ovisno o algoritmu koji se koristi, rezultat ove klasifikacijske funkcije može biti numerička oznaka koja označava pripadnost određenoj kategoriji ili procijenjenu vjerojatnost da neka slika pripada određenoj kategoriji.

Za klasifikaciju se može koristiti mnogo različitih algoritama, neki od primjera su: Stroj potpornih vektora (engl. *support vector machine*), k najbližih susjeda (engl. *k-nearest neighbours*), stablo odlučivanja (engl. *decision trees*), slučajne šume (engl. *randon forest*), konvolucijske neuronske mreže (engl. *convolutional neural network*) i još mnogi drugi. Ne postoji najbolji ili najtočniji algoritam. Algoritam treba birati ovisno o skupu podataka s kojim se radi. Na primjer, kako je već opisano u potpoglavlju 2.4.1, u istraživanju [11] trenirana su tri modela za prepoznavanje bolesti na lišću soje i uspoređeni su njihovi rezultati. Korišteni su algoritmi k-najbližih susjeda, stroj potpornih vektora te konvolucijske neuronske mreže. Dobivene su točnosti od 64%, 76% te 96%. Konvolucijske neuronske mreže pokazale su se kao najbolji algoritam za taj slučaj. S druge strane, u [14] klasificirale su se hiperspektralne slike koristeći algoritme slučajnih šuma te dubokog učenja. U ovom slučaju algoritam slučajnih šuma pokazao se kao bolji izbor sa točnosti od 84.72% dok se koristeći algoritam dubokog učenja dobila točnost od 81.32%.

U [11] se rješavao problem klasifikacije bolesti lišća soje što je vrlo slično ovom radu. Zbog toga će se za klasifikaciju u ovom radu koristiti konvolucijske neuronske mreže.

3.3.1. Konvolucijske neuronske mreže

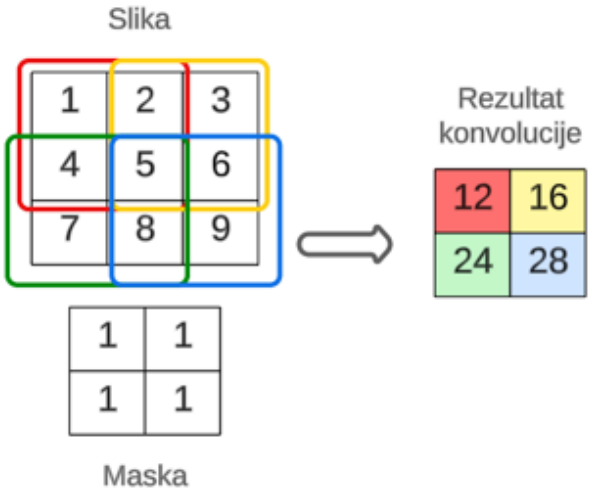
Kako je opisano u [15], cilj konvolucijske neuronske mreže je naučiti različite karakteristike iz slikovnih podataka koje postaju sve složenije kroz niz slojeva. Pomoću tih naučenih karakteristika, model treba prepoznati klasu nove slike na ulazu, čak i kada se prethodno nije susreo s tom slikom. U ovom radu, konvolucijska neuronska mreža se također koristi u tu svrhu. Jedna od prednosti konvolucijske neuronske mreže je da ne zahtijeva ravnanje slike u vektor brojeva, kao što je potrebno za potpuno povezane neuronske mreže, što bi izgubilo prostornu informaciju. Umjesto toga, konvolucijska neuronska mreža koristi konvolucijske operacije s maskama koje sadrže težine koje se uče. Filtrirana slika ili mapa značajki predstavlja novu sliku gdje svaki piksel nosi informaciju o prisutnosti karakteristike koju maska predstavlja na različitim dijelovima ulazne slike. Što je veće preklapanje između susjednih piksela na ulazu i maske, to će vrijednost piksela na mapi značajki biti veća nakon primjene konvolucije s odgovarajućom maskom. Budući da model ne može donijeti odluku samo na temelju jedne karakteristike slike, mreža koristi više maski koje proizvode različite mape značajki. Dimenzije slike se mogu smanjiti po potrebi, uz očuvanje prostorne informacije, korištenjem operacije *MaxPool*. Na kraju, potreban je potpuno povezani sloj koji donosi konačnu odluku o klasi kojoj slika pripada. Pojednostavljen prikaz konvolucijske neuronske mreže može se vidjeti na slici 3.1 po uzoru na [15].



Slika 3.1. *Pojednostavljena arhitektura konvolucijske neuronske mreže, po uzoru na [15]*

Konvolucijski sloj je ključna komponenta konvolucijskih neuronskih mreža te ima ključnu ulogu u izdvajanju značajki iz ulaznih slika. Njegova osnovna svrha je detekcija različitih karakteristika u ulaznim slikama. Kako ime govori, osnovna operacija konvolucijskog sloja je konvolucija. Kako je opisano u [16], konvolucija se provodi tako da se mala matrica poznata kao maska pomiče preko ulazne slike. Ova maska sadrži težine koje se uče tijekom treninga neuronske mreže. Svaki piksel u mapi značajki dobiva svoju vrijednost pomnoženu s težinom odgovarajućeg piksela na slici. Konvolucijski sloj koristi različite maske za izdvajanje različitih

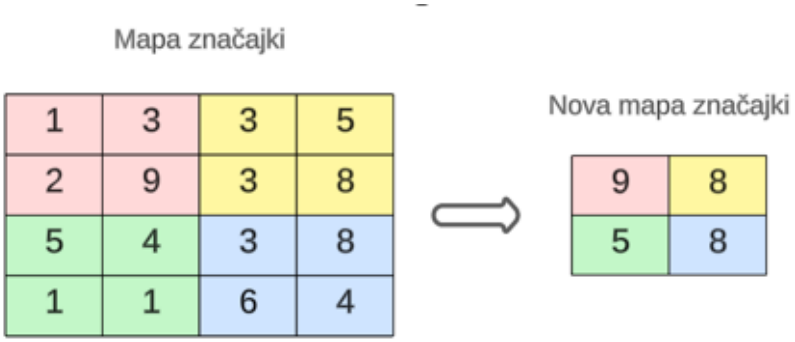
značajki iz slika. Svaka maska može biti odgovorna za prepoznavanje određenih oblika ili karakteristika, kao što su linije, bridovi, teksture itd. Kroz učenje prilikom treninga, mreža optimizira težine u maskama kako bi što bolje prepoznala značajke. Nakon što se maska pomakne preko cijele slike, rezultat je mapa značajki. Ova mapa sadrži informacije o prisutnosti različitih značajki na različitim dijelovima slike. Svaka mapa značajki odgovara jednoj masci i sadrži informacije o tome koliko je snažna prisutnost te značajke na različitim dijelovima slike. Nakon konvolucije, na svaki element mape značajki često se primjenjuje nelinearna funkcija kao što je *ReLU* (engl. *rectified linear unit*) kako bi se uvele nelinearnosti u modelu. To pomaže mreži da nauči složenije odnose između značajki. Ukratko, svrha konvolucijskog sloja je transformirati ulaznu sliku u reprezentaciju značajki koja je pogodna za daljnje analize i klasifikaciju. Ovo omogućava dubokim neuronskim mrežama da postanu izrazito efikasne u prepoznavanju oblika, objekata i uzoraka u slikama, što je ključno u mnogim aplikacijama kao što su prepoznavanje objekata, segmentacija slika, analiza medicinskih slika, autonomna vožnja i mnoge druge. Primjer konvolucije može se vidjeti na slici 3.2 po uzoru na [15].



Slika 3.2. *Primjer konvolucije koristeći 2D sliku i 2D masku, po uzoru na [15]*

MaxPooling sloj je još jedna važna komponenta konvolucijskih neuronskih mreža. Svrha *MaxPooling* sloja je smanjiti prostornu dimenziju mape značajki i tako smanjiti broj parametara u mreži. Ovaj sloj radi tako da dijeli mapu značajki (izlaz konvolucijskog sloja) na manje regije, obično kvadratne ili pravokutne. Svaka regija je obično veličine, na primjer, 2x2 piksela. Za svaku regiju, *MaxPooling* uzima najveću vrijednost iz te regije. To znači da se samo najjača značajka u svakoj regiji zadržava, dok se ostale značajke odbacuju. Ova operacija ima za posljedicu gubitak prostorne informacije, ali zadržava najvažnije značajke. Nakon primjene *MaxPooling* sloja, dimenzije mape značajki se smanjuju što pomaže smanjiti broj parametara u mreži jer se smanjuju dimenzije mapa značajki. To čini mrežu bržom za treniranje i manjom u

pogledu memorije. Primjer primjene *MaxPool* operatora može se vidjeti na slici 3.3 po uzoru na [15].



Slika 3.3. *Primjer primjene MaxPool operatora, po uzoru na [15]*

Potpuno povezani sloj često se koristi kao završni sloj u neuronskim mrežama. Ovaj sloj služi za donošenje konačnih odluka na temelju značajki iz prethodnih slojeva. Na primjer, ako je mreža korištena za klasifikaciju slika, potpuno povezani sloj donosi odluku o tome kojoj klasi slika pripada na temelju značajki koje su prepoznate u prethodnim slojevima. Ulaz u potpuno povezani sloj je vektor koji je proizašao iz prethodnih slojeva (konvolucijskih ili *MaxPooling* slojeva). Svaki neuron u potpuno povezanom sloju kombinira te značajke uz pomoć svojih težina i primjenjuje aktivacijsku funkciju. Izlaz potpuno povezanog sloja obično predstavlja rezultate klasifikacije. Na primjer, za klasifikaciju slika, svaki neuron u izlaznom sloju može predstavljati vjerojatnost pripadnosti određenoj klasi. Klasa s najvećom vjerojatnošću obično se smatra predikcijom mreže. Najčešće, izlazni sloj koristi *Softmax* aktivaciju kako bi pretvorio rezultate u vjerojatnosti čiji je zbroj jedan.

3.4. Slijed korištenja mobilne aplikacije

U ovom potpoglavlju objašnjeni su koraci kojima korisnik prolazi prilikom korištenja aplikacije. Na slici 3.1 može se vidjeti pojednostavljen prikaz slijeda korištenja aplikacije. Isto tako, može se vidjeti kako je prvo potrebna registracija. Nakon uspješne registracije korisnik je preusmjeren na glavni zaslon gdje se upoznaje s aplikacijom te podržanim bolestima. Nakon toga ima nekoliko izbora. Može samo učitati sliku i vidjeti o kojoj se bolesti radi ili može učitati sliku i pokrenuti sustav preporuka za liječenje. Isto tako, moguće je otvoriti već postojeći sustav preporuka za liječenje za jedan od nasada. Nakon što je otvoren jedan od sustava preporuka, moguće je zatražiti preporuke za liječenje. Dodatno, s glavnog zaslona korisnik još ima mogućnost dohvaćanja vremenske prognoze.

3.4.1. Registracija i prijava korisnika

Kao što je već spomenuto, korisnik se prvo mora registrirati kako bi mogao koristiti mobilnu aplikaciju. Prilikom registracije potrebno je unijeti ime, prezime, e-poštu i lozinku. Ukoliko je sve u redu, podatci se spremaju u bazu podataka. Dodatno, korisnički račun potrebno je aktivirati sljedeći upute koje su stigle na upisanu e-poštu prilikom registracije. U slučaju da je korisnik već registriran, treba se samo prijaviti.

3.4.2. Informativni dio mobilne aplikacije – upoznavanje korisnika s podržanim bolestima

Nakon uspješne prijave, korisnik se preusmjerava na početno zaslon aplikacije. Na početnom zaslonu, odnosno prvoj kartici, korisnik se može informirati o različitim bolestima koje su podržane te njihovoj problematici. Isto tako, korisnik može vidjeti kratki opis mobilne aplikacije.

3.4.3. Klasifikacija bolesti i kreiranje sustava preporuka za liječenje

S glavnog zaslon korisnik može odabrati karticu koja mu omogućuje učitavanje slike iz galerije ili slikanje koristeći kameru mobilnog uređaja. Kada se slika učita, predaje se unaprijed istreniranoj konvolucijskoj neuronskoj mreži koja prepoznaje ako je list zaražen nekom od bolesti ili je zdrav. Dodatno, nakon učitavanja slike može se kreirati sustav preporuka za liječenje bolesti. Prilikom kreiranja sustava preporuke korisnik treba unijeti ime sustava te lokaciju gdje se nasad nalazi. Kada je sustav kreiran, korisnik se preusmjerava na zaslon s kreiranim sustavom preporuka.

3.4.4. Pregled postojećih sustava preporuka za liječenje

Naravno, korisnik može otvoriti postojeće sustave preporuka. S glavnog zaslona treba odabrati karticu u kojoj su prikazani postojeći sustavi preporuka za liječenje. Nakon toga prikazuju se svi otvoreni sustavi preporuke prijavljenog korisnika. Klikom na jedan od njih, korisnik se preusmjerava na zaslon s tim sustavom za preporuke.

3.4.5. Pregled vremenske prognoze

Kao što je već spomenuto, vrijeme je vrlo bitno kod liječenja bolesti u voćarstvu. Iz tog razloga dodana je kartica gdje se može vidjeti vremenska prognoza. Na glavnom zaslonu treba odabrati karticu na kojoj će se prikazati vremenska prognoza za sljedećih pet dana u koracima od tri sata za trenutnu lokaciju. Dodatno, korisnik može ručno unijeti drugu lokaciju te dohvatiti vremensku prognozu za tu lokaciju.

3.4.6. Sustav preporuka za liječenje

Na zaslonu za sustav preporuka za liječenje korisnik može vidjeti informacije o sustavu i nasadu te cijelu povijest zahtjeva za preporuke kao i same preporuke. Dodatno, korisnik može zatražiti preporuku za liječenje, preporuku za vrijeme tretiranja nasada te provjeriti ako liječenje djeluje. Isto tako, korisnik ima mogućnost obrisati sustav preporuka.

3.4.7. Analiza podataka

Nakon učitavanja slike lista jabuke, slika se predaje unaprijed istreniranoj konvolucijskoj neuronskoj mreži. Mreža vraća vektor s vjerojatnostima da voćnjak boluje od pojedinih bolesti. Naravno, najveća vjerojatnost predstavlja predviđenu klasu, odnosno bolest. Te vjerojatnosti se skaliraju na način da klasa s najvećom vjerojatnošću ima vrijednost jedan, a klasa s najmanjom nula. Takve skalirane vrijednosti se spremaju u bazu podataka prilikom kreiranja novog sustava preporuka za liječenje kako bi se mogle koristiti prilikom generiranja budućih preporuka za liječenje. Sustav preporuka korisniku omogućuje provjeru ako liječenje djeluje ili ne. Tada korisnik ponovno učitava sliku, slika se predaje konvolucijskoj neuronskoj mreži koja vraća vektor s vjerojatnostima koje se skaliraju i spremaju u bazu podataka. Ovdje može doći do tri različita slučaja. Prvi slučaj je da je tretman djelovao i da je nasad sada izliječen. Tada se sustav preporuka prekida. U drugom slučaju, nasad je zaražen drugom bolešću ili mješavinom bolesti. Treći slučaj je da je nasad još uvijek boluje od iste bolesti i u tom slučaju se provjerava učinkovitost liječenja. Učinkovitost liječenja se određuje usporedbom vjerojatnošću da je nasad zdrav iz dva posljednja vektora s vjerojatnostima pronađenim u bazi podataka za taj sustav preporuka. Učinkovitost će biti izražena u postotcima te će taj broj biti pozitivan ako je liječenje pomoglo. U suprotnom, broj će biti negativan. Ovisno o postotku, ispisat će se primjerena poruka. Ako je izračunato da je učinkovitost veća ili jednaka od -5%, a manja ili jednaka od 5%, ispisat će se kako je stanje kao i kod prošle provjere te kako nema pomaka. Ako je učinkovitost veća od 5%, a manja ili jednaka od 50%, bit će ispisano kako liječenje djeluje. Nadalje, ako je učinkovitost veća od 50% bit će ispisano kako tretman djeluje te kako je nasad puno zdraviji. S druge strane, ako je učinkovitost manja od -5%, a veća ili jednaka od -50%, ispisat će se kako liječenje nije djelovalo kao što je bilo predviđeno. Na kraju, ako je učinkovitost manja od -50%, ispisat će se kako je nasad mnogo gore i da liječenje nije uopće djelovalo.

Korisnik u svakom trenutku može zatražiti preporuku za liječenje nasada jabuka. U tom slučaju pronalazi se posljednja klasificirana bolest u bazi podataka za taj sustav preporuka i ispisuje se primjerena poruka.

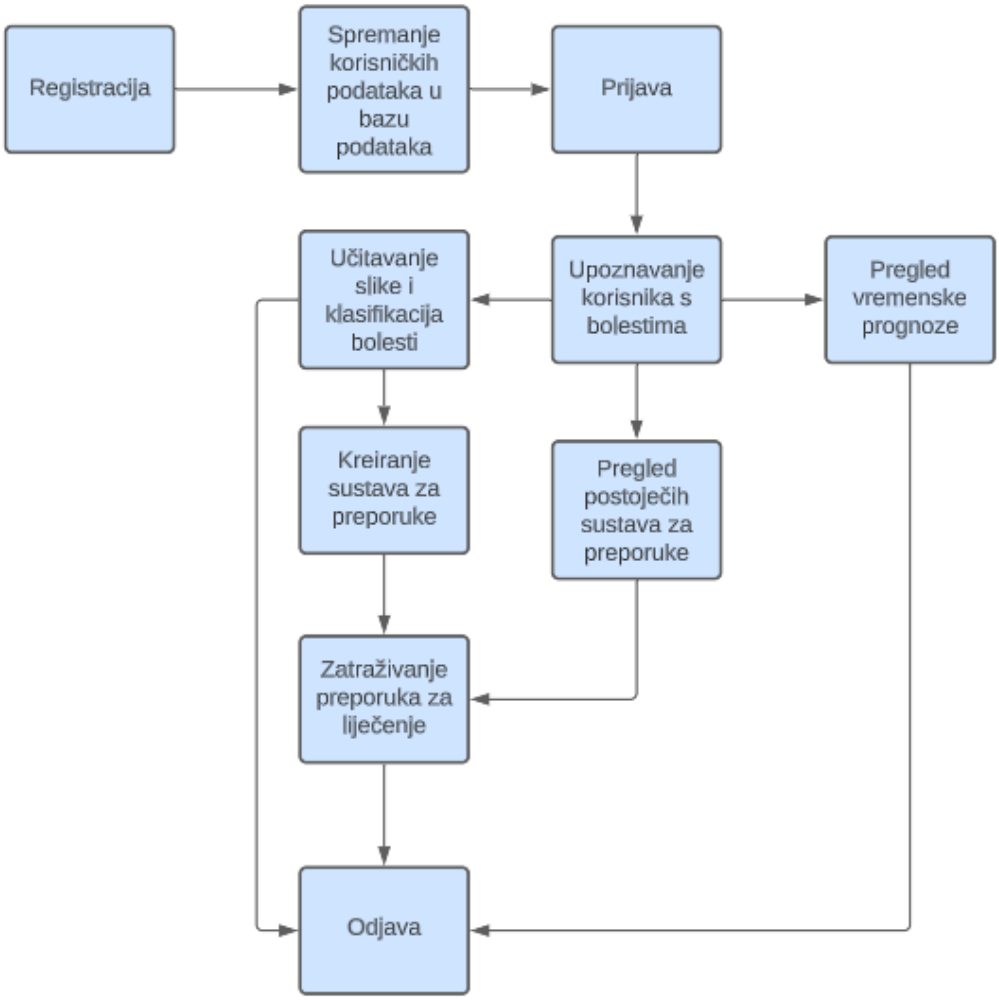
Isto tako, korisnik može zatražiti preporuku za vrijeme kada bi bilo najbolje tretirati nasad. Tada se dohvaća vremenska prognoza za lokaciju tog nasada i traži se vremenski prozor od dva dana gdje neće biti kiše, magle, leda ili bilo kakvih vremenskih nepogoda. Na kraju, ovisno o rezultatu, ispisuje se prikladna poruka.

3.4.8. Odjava korisnika

Korisnik ima mogućnost odjave. Korisnik kao i svi njegovi zapisi nalaze se u bazi podataka pa se korisnik može prijaviti s bilo kojeg uređaja koji ima instaliranu aplikaciju te pristupiti svojim sustavima preporuka za liječenje.

3.4.9. Promjena lozinke

Dodatno, korisnik ima mogućnost promjene lozinke u slučaju da je zaboravi ili da je jednostavno želi zamijeniti. U zaslonu za promjenu lozinke, korisnik treba unijeti e-poštu korištenu kod registracije na koju za nekoliko trenutaka dolazi e-pošta s daljnjim uputama.



Slika 3.4. *Prikaz slijeda aktivnosti pri korištenja aplikacije*

4. PROGRAMSKO RJEŠENJE MOBILNE APLIKACIJE I MODELA

U ovom poglavlju opisat će se korištene programske okoline i jezici kod razvoja mobilne aplikacije i treniranja modela strojnog učenja. Isto tako, prikazat će se i objasniti programsko rješenja mobilne Android aplikacije i proces treniranja modela strojnog učenja.

4.1. Korištene programske okoline i jezici

4.1.1. Android Studio

Kao što je navedeno u literaturi [17] i [18], Android Studio je rezultat suradnje između *JetBrains-a* i *Google-a*. Ova razvojna platforma temelji se na moćnom *IntelliJ* okruženju, pružajući sve njegove funkcionalnosti. Gotovo sve što je moguće ostvariti u *IntelliJ-u* može se također izvesti u Android Studiju. Jedno od ključnih dostignuća Android Studija je pojednostavljenje procesa razvoja Android aplikacija, čime je razvoj postao pristupačniji nego ikada prije. Android Studio podržava programiranje Android aplikacija kako u Javi tako i u Kotlinu, pri čemu je Kotlin u porastu i postaje sve popularniji jezik u Android zajednici. Jedna od velikih prednosti Android Studija je u tome što pomaže programerima na više načina: detektira pogreške, pruža jednostavan način za ispravljanje istih, generira kod te nudi korisne sugestije tijekom pisanja koda. Osim toga, Android Studio omogućuje jednostavno pokretanje i testiranje Android aplikacija na virtualnom uređaju koji simulira karakteristike stvarnog uređaja. Naravno, aplikacije se mogu i dalje testirati na stvarnim uređajima.

4.1.2. Operacijski sustav Android

Prema izvoru [19], operacijski sustav Android se opisuje kao mobilni operacijski sustav. Prvotno je razvijen od strane Android Inc. i temelji se na modificiranoj jezgri Linux-a verzije 2.6. Android je otvorenog koda, omogućavajući svakome prilagodbu prema vlastitim potrebama. Zbog ovih prednosti, Android je postao najdominantniji operacijski sustav za mobilne uređaje i prisutan je u širokom rasponu uređaja, uključujući mobilne telefone, tablete, pametne satove, televizore, automobile i druge. Kako su izvor [20] i [21] istaknuli, arhitektura operacijskog sustava Android se može podijeliti na pet razina: Linux Kernel, komplet biblioteka, *Android Runtime*, aplikacijski sloj i aplikacije.

4.1.3. Programski jezik Java

Kako je opisano u izvoru [22], Sun Microsystems je započeo razvoj novog programskog jezika 1991. godine s namjerom da se usmjeri na male uređaje. Taj programski jezik kasnije je dobio ime Java. Glavni cilj razvoja bio je stvaranje jezika koji bi zauzimao malo memorije i istovremeno bio kompatibilan s različitim procesorima. Razvoj Java programskog jezika započeo

je s izmjenom UCSD Pascala. Generirani Java kod se izvršava na virtualnom stroju, dok stvarni stroj ima ugrađeni interpreter koji je odgovoran za interpretaciju takvog koda. Ovaj pristup omogućava da se Java programi pišu neovisno o konkretnoj arhitekturi procesora, što je jedan od ključnih faktora njegove popularnosti. Kao što su naveli izvori [22] i [23], Java je objektno orijentirani programski jezik i platforma koja je široko korištena na mobilnim uređajima, igraćim konzolama, prijenosnim računalima i mnogim drugim uređajima. Njegova sposobnost da radi na različitim vrstama uređaja i arhitekturama doprinosi njegovoj širokoj primjeni u raznim industrijama i sektorima.

4.1.4. Označni jezik XML

Kako je objašnjeno u izvoru [24], proširivi označni jezik ili XML (Engl. *Extensible Markup Language*) jezik je dizajniran za opisivanje strukturiranih tekstualnih podataka. Glavni cilj XML-a bio je razviti jezik koji bi bio lako čitljiv i interpretiran kako za ljude tako i za računala. Neka od ključnih obilježja XML- a su razumljivost, univerzalnost i fleksibilnost. Još je važno napomenuti da XML sam po sebi ne izvršava nikakve akcije. To znači da, iako omogućava strukturiranje podataka na način koji je razumljiv ljudima i računalima, potrebno je koristiti programski jezik ili alate za obradu XML podataka. Ovo omogućava fleksibilnost u tome kako se podaci obrađuju i interpretiraju, ovisno o potrebama konkretnih aplikacija.

4.1.5. Programski jezik Python

Python je programski jezik visoke razine koji je popularan među programerima zbog svoje jednostavnosti i čitljivosti. Razvio ga je Guido van Rossum i prvi put je objavljen 1991. godine. Python ima jednostavnu i intuitivnu sintaksu, što ga čini pogodnim za početnike u programiranju, ali istovremeno nudi i napredne mogućnosti koje zadovoljavaju potrebe iskusnih programera. Python podržava različite paradigme programiranja, uključujući proceduralno, objektno orijentirano i funkcionalno programiranje. Također, prenosiv je i može se izvoditi na različitim platformama, uključujući Windows, macOS, Linux i mnoge druge. Jedna od ključnih značajki Pythona je činjenica da je otvorenog koda (engl. *open source*), što znači da je besplatan za korištenje i da se može prilagoditi prema potrebama korisnika. Kako je opisano u [25], nakon usporedbe Python programskog jezika s ostalim popularnim programskim jezicima, Python se pokazao kao vrlo dobar izbor zbog svoje čitljivosti, lagane sintakse te pouzdanosti. Dodatno, Python se pokazao kao vrlo dobar izbor za početnike koji tek ulaze u svijet programiranja.

4.1.6. Tensorflow i Keras

Kako je opisano u izvoru [26], *TensorFlow* je otvorena platforma za strojno učenje razvijena od strane Google-a. Ona omogućava razvoj, treniranje i primjenu neuronskih mreža i drugih modela strojnog učenja. *TensorFlow* omogućava izradu složenih modela dubokog učenja koristeći grafički sustav za definiranje i izvođenje računalnih operacija na tensorima, koji su temeljne jedinice podataka u *TensorFlow-u*. *TensorFlow* pruža podršku za različite platforme, uključujući CPU, GPU i TPU, čime omogućava ubrzanje treniranja i izvođenja modela.

Također, u [26] se navodi kako je *Keras* visokorazinski API za izgradnju neuronskih mreža, koji je dizajniran da bude jednostavan za korištenje i intuitivan za programere. Početno je razvijen kao zaseban projekt, ali kasnije je postao sastavni dio *TensorFlow-a*. *Keras* omogućava brzo i jednostavno definiranje arhitekture modela dubokog učenja putem slojeva koji se povezuju zajedno. On također pruža bogat skup funkcionalnosti za upravljanje treniranjem, validacijom i evaluacijom modela.

4.1.7. Firebase

Kako je opisano u [27], *Firebase* je platforma za razvoj aplikacija koja je razvijena od strane Google-a. Ona pruža različite alate i usluge za olakšavanje razvoja mobilnih, web i desktop aplikacija. Neki od ključnih aspekata *Firebase* platforme su:

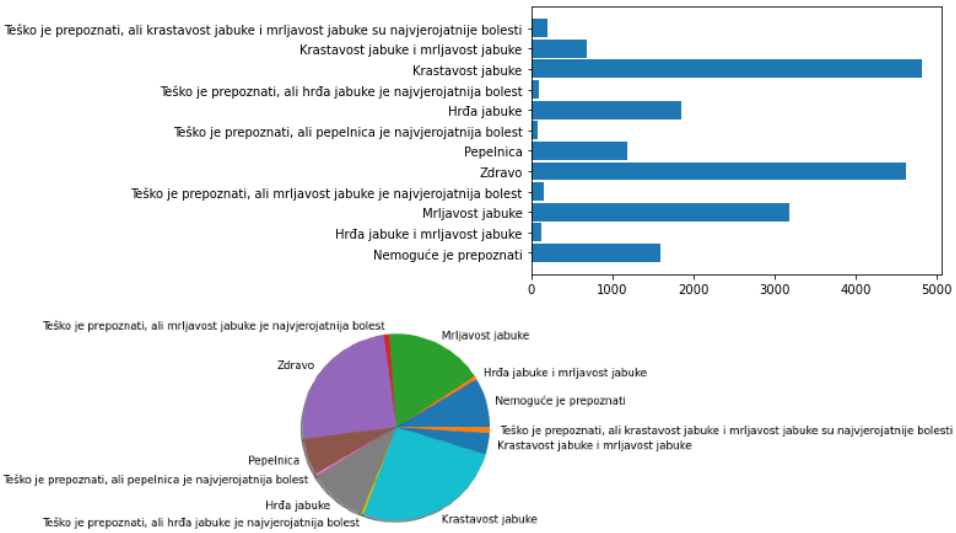
- Baza podataka u stvarnom vremenu: *Firebase Realtime Database* omogućava programerima da lako povežu svoje aplikacije s cloud bazom podataka u stvarnom vremenu. Ovo je posebno korisno za aplikacije koje zahtijevaju sinkronizaciju podataka između različitih uređaja i korisnika.
- Autentifikacija korisnika: *Firebase* omogućava jednostavno integriranje autentifikacije korisnika putem različitih metoda, kao što su e-mail i lozinka, Google, Facebook, Twitter i ostali.
- Hosting i skladištenje: *Firebase* Hosting omogućava brz i lak hosting web aplikacija i statičkih sadržaja. *Firebase* također pruža usluge za skladištenje datoteka i medija.
- Analitika: *Firebase* Analytics omogućava praćenje ponašanja korisnika u aplikaciji, omogućavajući programerima da bolje razumiju kako korisnici koriste aplikaciju i donose informirane odluke za poboljšanje.
- Testiranje i optimizacija: *Firebase* nudi alate za testiranje aplikacija kao što su A/B testiranje, praćenje performansi i pronalaženje grešaka.
- Dinamičke veze: Omogućava stvaranje dinamičkih dubokih veza koje korisnike mogu voditi do određenih dijelova aplikacije čak i kad je aplikacija instalirana naknadno.

- Notifikacije: *Firebase Cloud Messaging* omogućava programerima slanje obavijesti i poruka korisnicima na različitim platformama.
- Sigurnost i skalabilnost: *Firebase* pruža sigurno i skalabilno rješenje za aplikacije, omogućavajući programerima da se više usmjere na funkcionalnosti aplikacije umjesto na upravljanje infrastrukturom.

Dodatno, u [28] je uspoređen odziv *Firebase* baze u stvarnom vremenu te *MySQL* baze podataka. Zaključak je kako *Firebase* baze u stvarnom vremenu ima kraći odziv za CRUD (engl. *create, read, update and delete*) operacije što ju čini vrlo dobrim odabirom za aplikacije poput ovog rada.

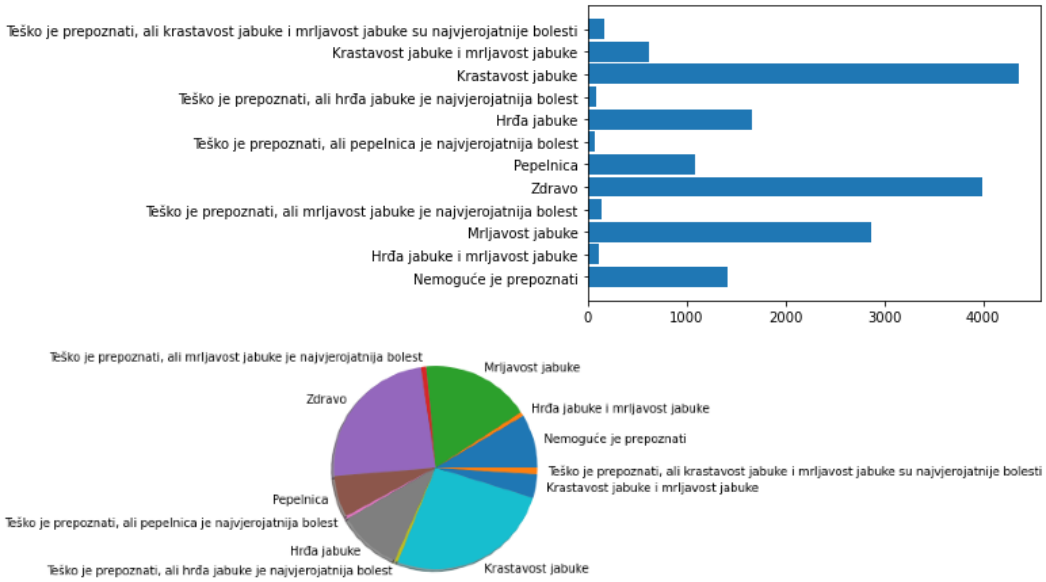
4.2. Analiza skupa podataka za treniranje modela strojnog učenja

Prilikom treniranja modela strojnog učenja koji se koristi u ovoj aplikaciji, korišten je skup podataka preuzet s [29]. Skup podataka sastoji se od 18632 slike dimenzija 512x512 piksela koje su unaprijed razvrstane u skupove za treniranje, validaciju te testiranje modela strojnog učenja. Skupovi podataka za testiranje i validaciju imaju svaki po tisuću slika dok je ostatak u skupu podataka za treniranje modela. Skup podatak se dijeli na dvanaest različitih klasa, a to su: zdrav list, mrljavost lista jabuke, pepelnica jabuke, krastavost jabuke, hrđa jabuke, mrljavost lista jabuke u kombinaciji s krastavosti jabuke, mrljavost lista jabuke u kombinaciji sa hrđom jabuke, mješavina više bolesti gdje prevladava mrljavost lista jabuke, mješavina više bolesti gdje prevladava pepelnica jabuke, mješavina više bolesti gdje prevladava hrđa jabuke, mješavina više bolesti gdje prevladavaju mrljavost lista jabuke i krastavost jabuke i mješavina više bolesti gdje je prekomplicirano odrediti bolest. Na slici 4.1 može se vidjeti približan broj i omjer slika po pojedinoj klasi za cijeli skup podataka.

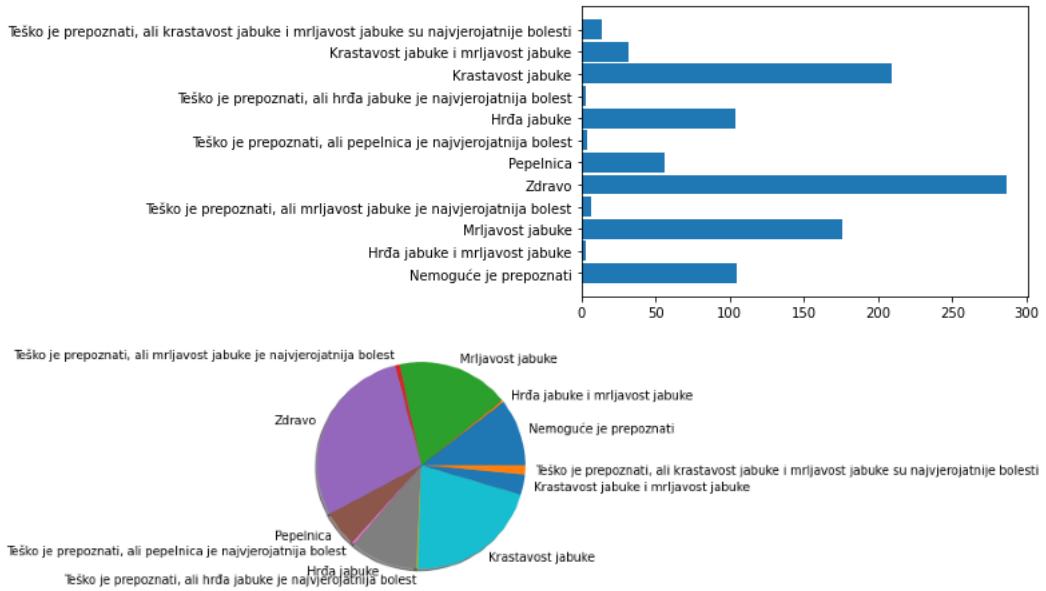


Slika 4.1. *Raspodjela slika po klasama za cijeli skup podataka*

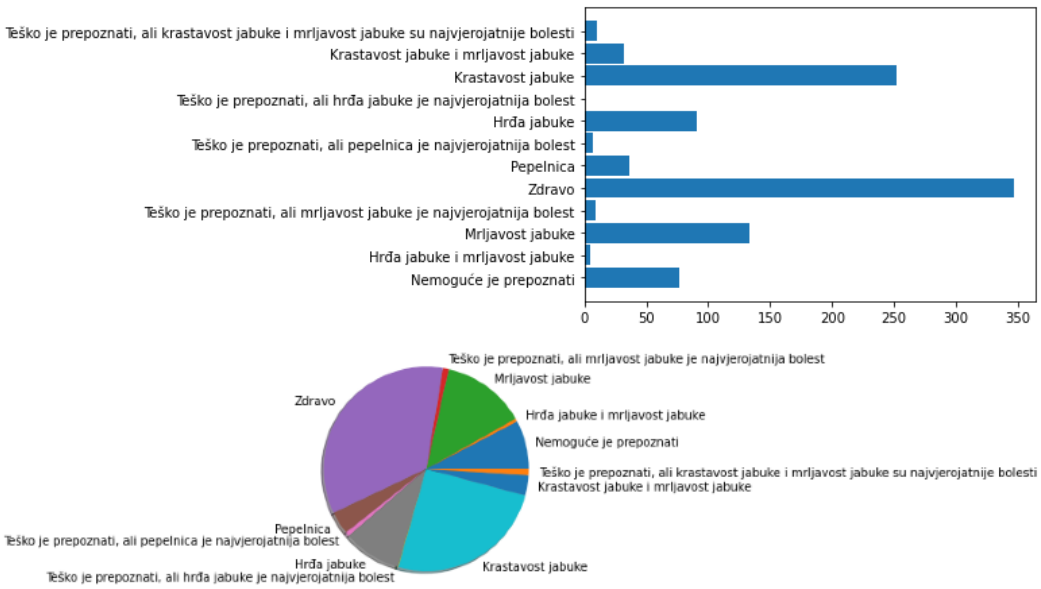
Sa slike 4.1 može se vidjeti kako najviše slika ima za slučajeve kada je prisutna jedna bolest, kada je list zdrav te kada je prekomplicirano odrediti bolest. Ostale klase kada je prisutno više bolesti su ne pojavljuju toliko teško, a to se može vidjeti i po broju takvih slika u skupu podataka. Kao što je već spomenuto, ovaj skup podataka se dijeli na tri podskupa, tj. skup za treniranje, validaciju i testiranje. Na slikama 4.2, 4.3 i 4.4 može se vidjeti približan broj te omjer slika po klasama.



Slika 4.2. *Raspodjela slika po klasama za skup podataka korišten za treniranje modela strojnog učenja*



Slika 4.3. *Raspodjela slika po klasama za skup podataka korišten za validaciju modela strojnog učenja*



Slika 4.4. *Raspodjela slika po klasama za skup podataka korišten za testiranje modela strojnog učenja*

Većina slika iz izvornog skupa podataka nalazi se u skupu podataka za treniranje modela strojnog učenja što se može vidjeti usporedbom slike 4.1 i slike 4.2. Skupovi podataka za validaciju i testiranje modela strojnog učenja su puno manji, ali na slikama 4.3 i 4.4 može se vidjeti kako su omjeri podataka po klasama otprilike jednaki kao i kod ostalih skupova podataka.

4.3. Razvoj, treniranje i ispitivanje modela strojnog učenja

Kao što je opisano u potpoglavlju 4.2, skup podataka unaprijed je razvrstan u skupove za treniranje, validaciju te testiranje modela strojnog učenja. Kada se skup podataka preuzme, sastoji se od datoteke u kojoj se nalaze sve fotografije, *txt* datoteke koja sadrži informacije o klasama te *csv* datoteke u kojoj je za svaku fotografiju navedeno ime te kojoj klasi i skupu podataka pripada. Isječak iz *csv* datoteke može se vidjeti na slici 4.5.

```
index, filepaths, labels, data_set
0,800113bb65efe69e.jpg,healthy,train
0,8002cb321f8bfcdf.jpg,scab frog_eye_leaf_spot complex,train
0,80070f7fb5e2ccaa.jpg,scab,train
0,80077517781fb94f.jpg,scab,train
0,800cbf0ff87721f8.jpg,complex,train
0,800edef467d27c15.jpg,healthy,train
0,800f85dc5f407aef.jpg,rust,train
0,801d6dcd96e48ebc.jpg,healthy,train
0,801f78399a44e7af.jpg,complex,train
```

Slika 4.5. *Isječak iz .csv datoteke koja sadrži informacije o fotografijama*

Kako bi se skupovi podataka za treniranje, validaciju i testiranje kasnije lako učitali koristeći metodu *image_dataset_from_directory* dostupnu u *Keras* biblioteci, potrebno ih je razvrstati u pripadajuće datoteke. Za razvrstavanje slika u pripadajuće datoteke korišten je programski kod koji se može vidjeti na slici 4.6.

```
import pandas as pd
import numpy as np
import shutil
import os

data = pd.read_csv("archive/data.csv")
data = pd.DataFrame(data)

data = data.iloc[:,[1,2,3]]
data = data.to_numpy()

labels = data[:, [1]]
unique_labels = np.unique(labels)
sets = np.unique(data[:, [2]])

#Sorting pictures to the classes
if not os.path.exists("sorted_data"):
    os.mkdir("sorted_data")

for subset in sets:
    path = os.path.join("sorted_data", subset)
    if not os.path.exists(path):
        os.mkdir(path)
    for label in unique_labels:
        path = os.path.join("sorted_data", subset, label)
        if not os.path.exists(path):
            os.mkdir(path)

#Copying images to the correct folders
source = os.path.join(os.getcwd(),"archive", "images")
count = 0

for picture in data:
    file = picture[0]
    file_class = picture[1]
    file_subset = picture[2]
    src = os.path.join(source, file)
    des = os.path.join(os.getcwd(), "sorted_data", file_subset, file_class, file)
    print("source -->" + src)
    print("destination -->" + des)
    shutil.copy(src, des)
    count += 1
```

Slika 4.6. *Programski kod za razvrstavanje fotografija u odgovarajuće datoteke*

Kao što se može vidjeti sa slike 4.6, prvo su učitani podatci iz *csv* datoteke. Nakon toga što su podatci učitani, traže se sve jedinstvene *labele*, odnosno klase te svi jedinstveni skupovi podataka. Nakon toga kreira se *sorted_data* datoteka i kreće se s razvrstavanjem podataka. Prolazi se redak po redak kroz *csv* datoteku te se fotografija kopira u odgovarajuću datoteku. Rezultat su tri nove datoteke unutar *sorted_data* koje predstavljaju skupove podatka za trening, validaciju i testiranje modela. Dodatno, svaki od skupova podataka sadrži još dvanaest datoteka od koje svaka predstavlja po jednu klasu. Nakon što su fotografije razvrstane može se krenuti s treniranjem modela konvolucijske neuronske mreže.

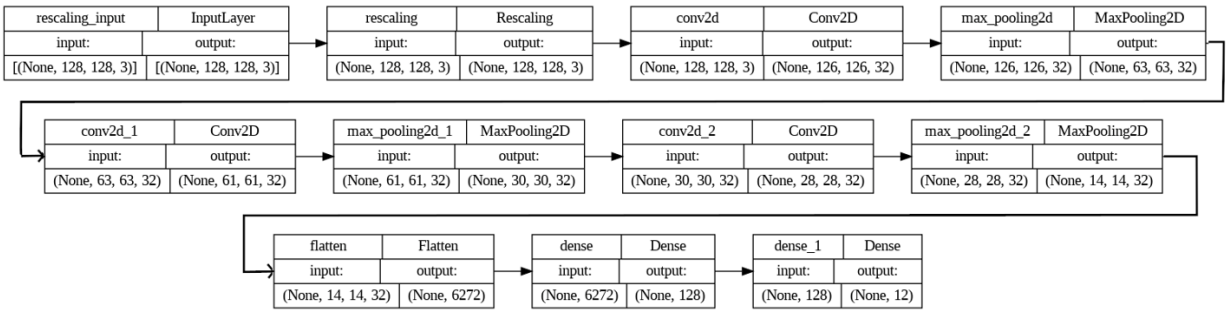
4.3.1. Prva inačica modela – jednostavna konvolucijska neuronska mreža

Prilikom treniranja prve inačice konvolucijske neuronske mreže ekperimentirano je s vrlo jednostavnim modelom. Na slici 4.7 može se vidjeti programski kod s kojim je dobiven najbolji rezultat nakon testiranja različitih parametara prilikom treniranja ovog modela.

```python
import tensorflow as tf

img_height, img_width = 128, 128
batch_size = 40

train_ds = tf.keras.utils.image_dataset_from_directory(
    "sorted_data/train",
    image_size = (img_height, img_width),
    batch_size = batch_size
)
val_ds = tf.keras.utils.image_dataset_from_directory(
    "sorted_data/valid",
    image_size = (img_height, img_width),
    batch_size = batch_size
)
test_ds = tf.keras.utils.image_dataset_from_directory(
    "sorted_data/test",
    image_size = (img_height, img_width),
    batch_size = batch_size
)

model = tf.keras.Sequential(
    [
     tf.keras.layers.Rescaling(1./255),
     tf.keras.layers.Conv2D(32, 3, activation="relu"),
     tf.keras.layers.MaxPooling2D(),
     tf.keras.layers.Conv2D(32, 3, activation="relu"),
     tf.keras.layers.MaxPooling2D(),
     tf.keras.layers.Conv2D(32, 3, activation="relu"),
     tf.keras.layers.MaxPooling2D(),
     tf.keras.layers.Flatten(),
     tf.keras.layers.Dense(128, activation="relu"),
     tf.keras.layers.Dense(12, activation='softmax')
    ]
)

model.compile(
    optimizer="adam",
    loss=tf.losses.SparseCategoricalCrossentropy(from_logits = True),
    metrics=['accuracy']
)

history = model.fit(
    train_ds,
    validation_data = val_ds,
    epochs = 10
)

model.evaluate(test_ds)
```

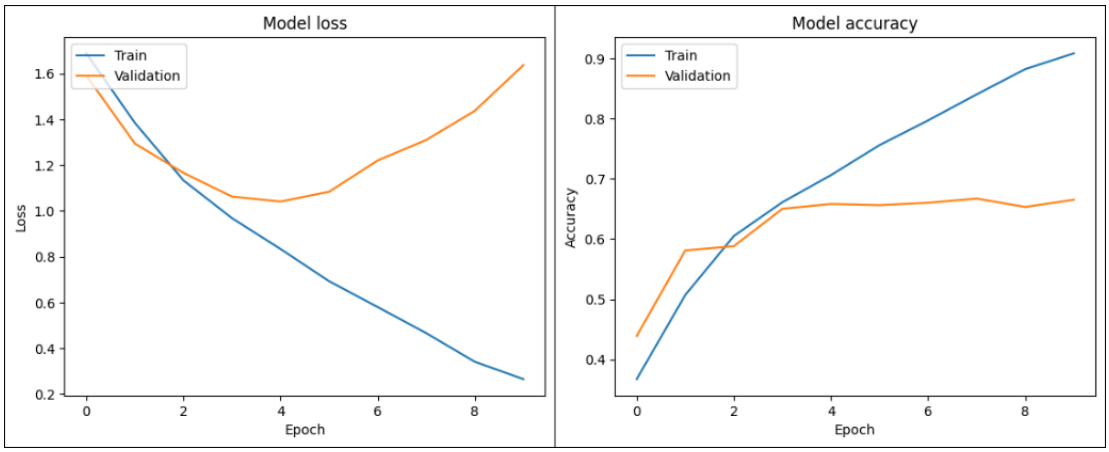
Slika 4.7. *Programski kod za treniranje prve inačice konvolucijske neuronske mreže*

Sa slike 4.7 može se vidjeti kako se na početku učitavaju skupovi podataka za treniranje, validaciju te testiranje. Učitane slike su dimenzija 128x128 piksela te je veličina grupe (engl. *batch size*) postavljena na četrdeset. Nakon toga model se definira sloj po sloj. Prvi sloj normalizira vrijednosti piksela na raspon od nula do jedan. Nakon toga slijede tri konvolucijska i *MaxPooling* sloja. Ovi slojevi zaduženi su za izlučivanje značajki iz slika. Slijedeći je *Flatten* sloj koji mape značajki pretvara u vektor koji se poslije toga predaje potpuno povezanom sloju koji ima sto dvadeset i osam neurona s *ReLu* aktivacijskom funkcijom. Na kraju se nalazi potpuno povezani sloj sa dvanaest neurona te *softmax* aktivacijskom funkcijom. Izlaz ovog sloja je vektor koji sadrži vjerojatnost pripadnosti svakoj od klasa. Arhitektura modela s dimenzijama ulaznih i izlaznih podataka može se vidjeti na slici 4.8.



Slika 4.8. *Arhitektura prve inačice modela gdje se koristi jednostavna konvolucijska neuronska mreža*

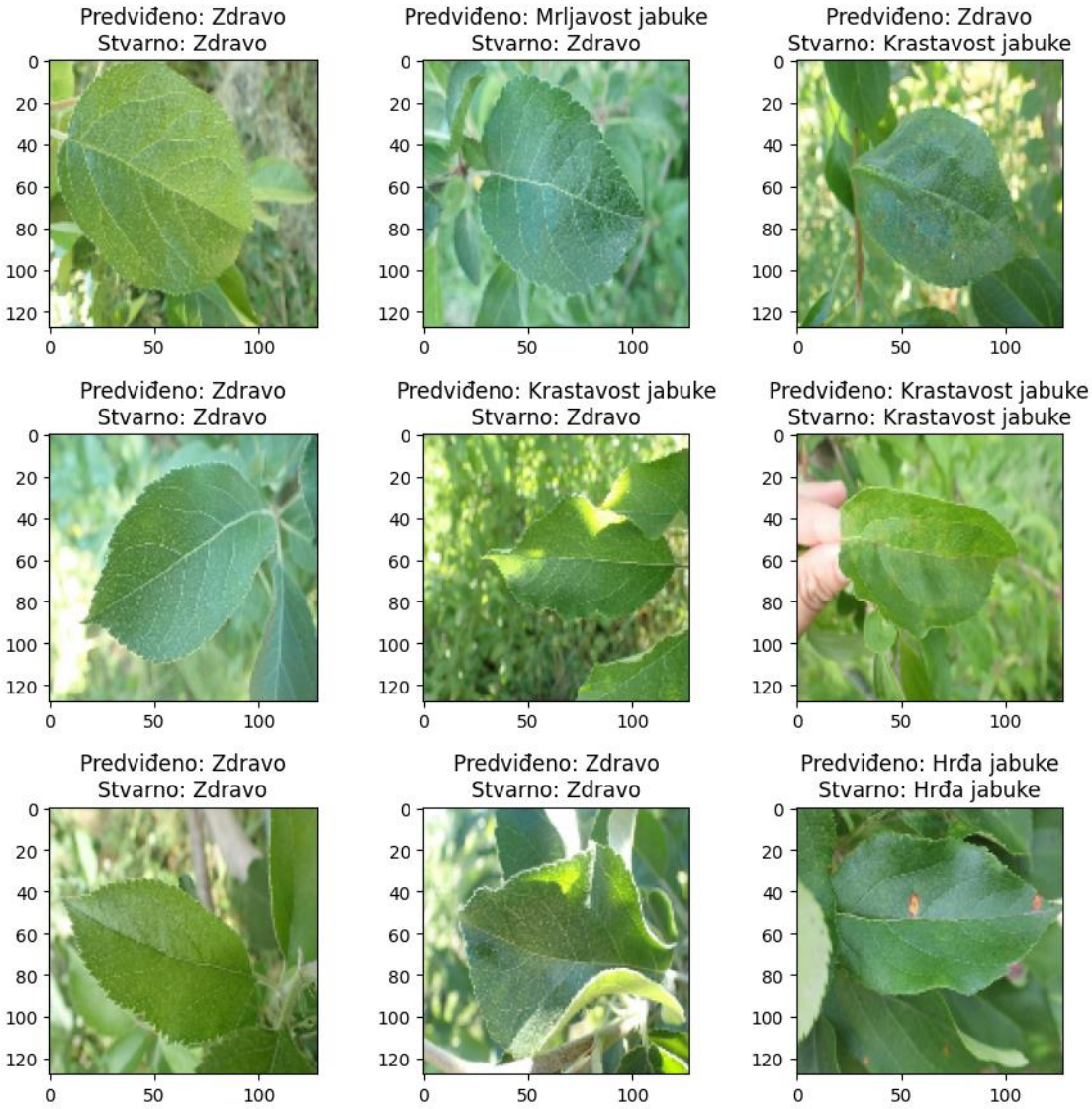
Model se nakon toga prevodi. Za *optimizer* se postavlja „Adam“. "Adam" je popularan *optimizer* u dubokom učenju, on adaptivno prilagođava stopu učenja (engl. *learning rate*) kako bi ubrzao konvergenciju i poboljšao efikasnost optimizacije. Za funkciju gubitka (engl. *loss function*) postavlja se *Sparse Categorical Crossentropy*, što je česta funkcija gubitka za zadatke klasifikacije s više klasa, posebno kada svaka instanca pripada samo jednoj klasi. Funkciju gubitka koristi se za mjerenje razlike između stvarnih izlaza modela i ciljnih vrijednosti tijekom treniranja. Točnost (engl. *accuracy*) postavlja se za metriku za praćenje performansi modela tijekom treniranja. Na kraju, modelu se predaju skupovi podataka za trening i validaciju te se model trenira kroz deset iteracija (engl. *epochs*). Na slici 4.9 može se vidjeti kretanje validacijskog gubitka (engl. *validation loss*) te gubitka na treniranju (engl. *training loss*), odnosno točnosti na validacijskom skupu te točnosti na skupu za treniranje kroz iteracije.



Slika 4.9. *Validacijski gubitak i gubitka na treniranju te točnost na validacijskom skupu i točnost na skupu za treniranje – prva inačica modela*

Sa slike 4.9 može se vidjeti kako poslije četvrte iteracije validacijski gubitak počinje rasti dok gubitak na treniranju nastavlja padati, to je snažan znak da je model previše prilagođen (engl. *overfitting*) trening podatcima što nije dobro te treba uzeti u obzir. Korištenjem ovog modela postignuta je najveća točnost od 69.4%. Veliki problem pri treniranju modela stvaraju i

neujednačeni podatci, tj. neke klase imaju puno više primjera od ostalih što uvelike smanjuje točnost kod određivanja tih klasa. Korištenjem ovog modela te klase će najčešće biti krivo klasificirane dok će klase za koje ima puno primjera većinom biti točno klasificirane. Na slici 4.10 može se vidjeti devet nasumičnih slika iz trening skupa te njihovu predviđenu i stvarnu klasu.



Slika 4.10. *Devet nasumičnih slika iz skupa za trening te njihove predviđene i stvarne klase – prva inačica modela*

4.3.2. Druga inačica model – složenija konvolucijska neuronska mreža

Nakon eksperimentiranja s vrlo jednostavnim modelom, testirali su se malo složeniji modeli konvolucijskih neuronskih mreža. Na slici 4.11 može se vidjeti programski kod s kojim je dobiven najbolji rezultat prilikom eksperimentiranja s malo složenijim modelima. Programski kod za učitavanje skupova podataka nije prikazan pošto je sličan istom kodu u prošlom

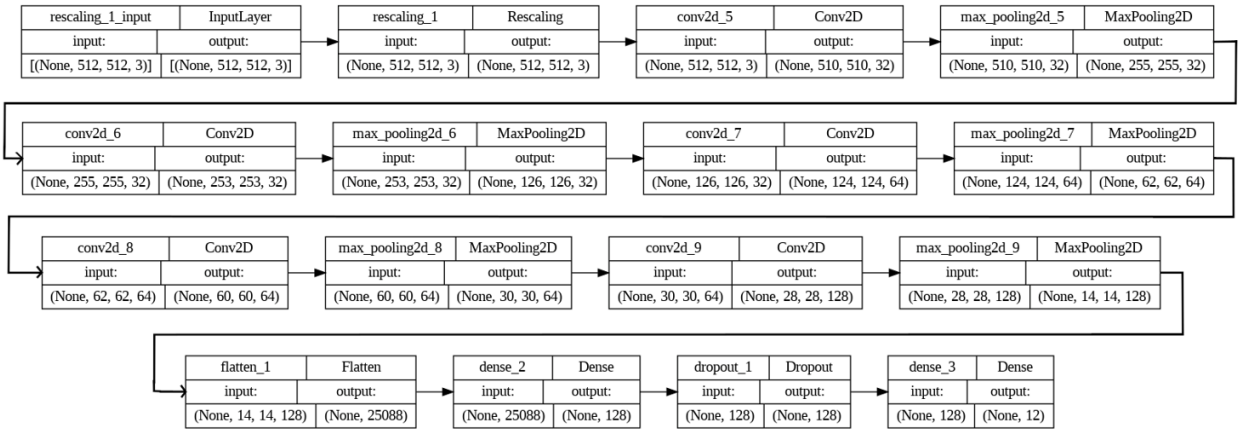
potpoglavlju. Razlika je jedino u dimenziji učitanih slika koja su sada 512x512 piksela te veličini grupe koja je sada trideset i pet.

```
model = tf.keras.Sequential(
    [
     tf.keras.layers.Rescaling(1./255),
     tf.keras.layers.Conv2D(32, 3, activation='relu', input_shape=(512, 512, 3)),
     tf.keras.layers.MaxPooling2D(),
     tf.keras.layers.Conv2D(32, 3, activation='relu'),
     tf.keras.layers.MaxPooling2D(),
     tf.keras.layers.Conv2D(64, 3, activation='relu'),
     tf.keras.layers.MaxPooling2D(),
     tf.keras.layers.Conv2D(64, 3, activation='relu'),
     tf.keras.layers.MaxPooling2D(),
     tf.keras.layers.Conv2D(128, 3, activation='relu'),
     tf.keras.layers.MaxPooling2D(),
     tf.keras.layers.Flatten(),
     tf.keras.layers.Dense(128, activation="relu"),
     tf.keras.layers.Dropout(0.5),
     tf.keras.layers.Dense(12, activation='softmax')
    ]
)

early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True
)

model.compile(
    optimizer="adam",
    loss=tf.losses.SparseCategoricalCrossentropy(from_logits = False),
    metrics=['accuracy']
)

history = model.fit(
    train_ds,
    validation_data = val_ds,
    epochs = 15,
    callbacks=[early_stopping]
)

model.evaluate(test_ds)
```

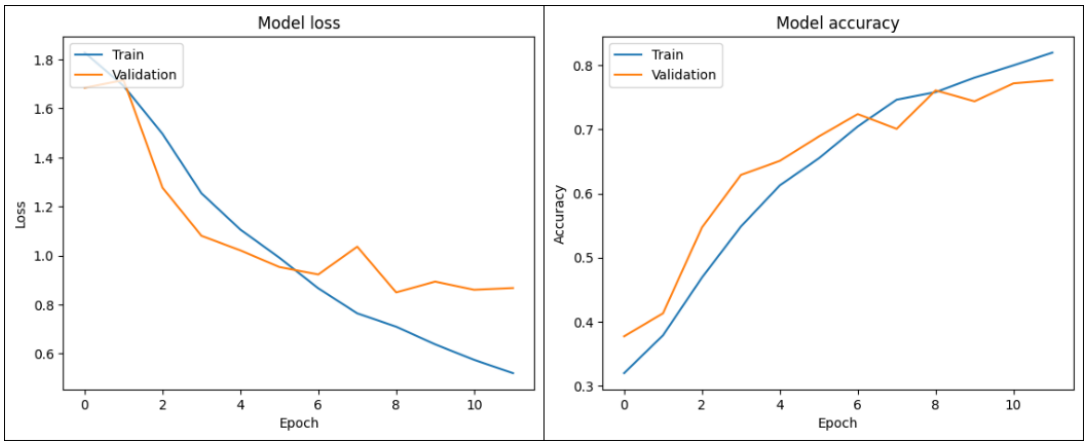
Slika 4.11. *Programski kod za treniranje druge inačice konvolucijske neuronske mreže*

Kao i kod prve inačice modela, ovaj model definira se sloj po sloj. Prvi sloj normalizira vrijednosti piksela na raspon od nula do jedan. Nakon toga slijede konvolucijski i *MaxPooling* slojevi. Ralika od prošlog modela je što sada ima više takvih slojeva te se postupno povećava broj filtera u konvolucijskim slojevima. Što omogućuje mreži da nauči različite složene značajke iz slika. Kao i kod prve inačice modela, slijedeći je *Flatten* sloj koji mape značajki pretvara u vektor koji se poslije toga predaje potpuno povezanom sloju koji ima sto dvadeset i osam neurona s *ReLu* aktivacijskom funkcijom. Tada je dodan *Dropout* sloj koji se koristi za regularizaciju kako bi se izbjeglo da se model previše prilagodi trening podatcima. Na kraju se nalazi potpuno povezani sloj sa dvanaest neurona te *softmax* aktivacijskom funkcijom. Arhitektura modela s dimenzijama ulaznih i izlaznih podataka može se vidjeti na slici 4.12. Dodatno, kod ovog modela implementira se rano zaustavljanje (engl. *early stopping*) kako bi se

treniranje automatski zaustavilo kada se funkcija gubitka na validacijskom skupu prestane poboljšavati nakon određenog broja iteracija. U ovom slučaju zaustavit će se nakon tri iteracije ako nema napretka. Rano zaustavljanje koristi se kako bi se model spriječio da se previše prilagodi trening podatcima. Isto tako, ako se treniranje zaustavi, vratit će se težine koje su dale najbolje rezultate na validacijskom skupu. Model se tada prevodi. „Adam“ se postavlja za *optimizer* te *Sparse Categorical Crossentropy* za funkciju gubitka. Isto tako, točnost se postavlja za metriku koja će se pratiti za provjeru performansi modela tijekom treniranja. Na kraju, modelu se predaju skupovi podataka za trening i validaciju te se počinje s treniranjem. Na slici 4.9 može se vidjeti kretanje validacijskog gubitka i gubitka na treniranju te točnosti na validacijskom skupu te točnosti na skupu za treniranje kroz iteracije.



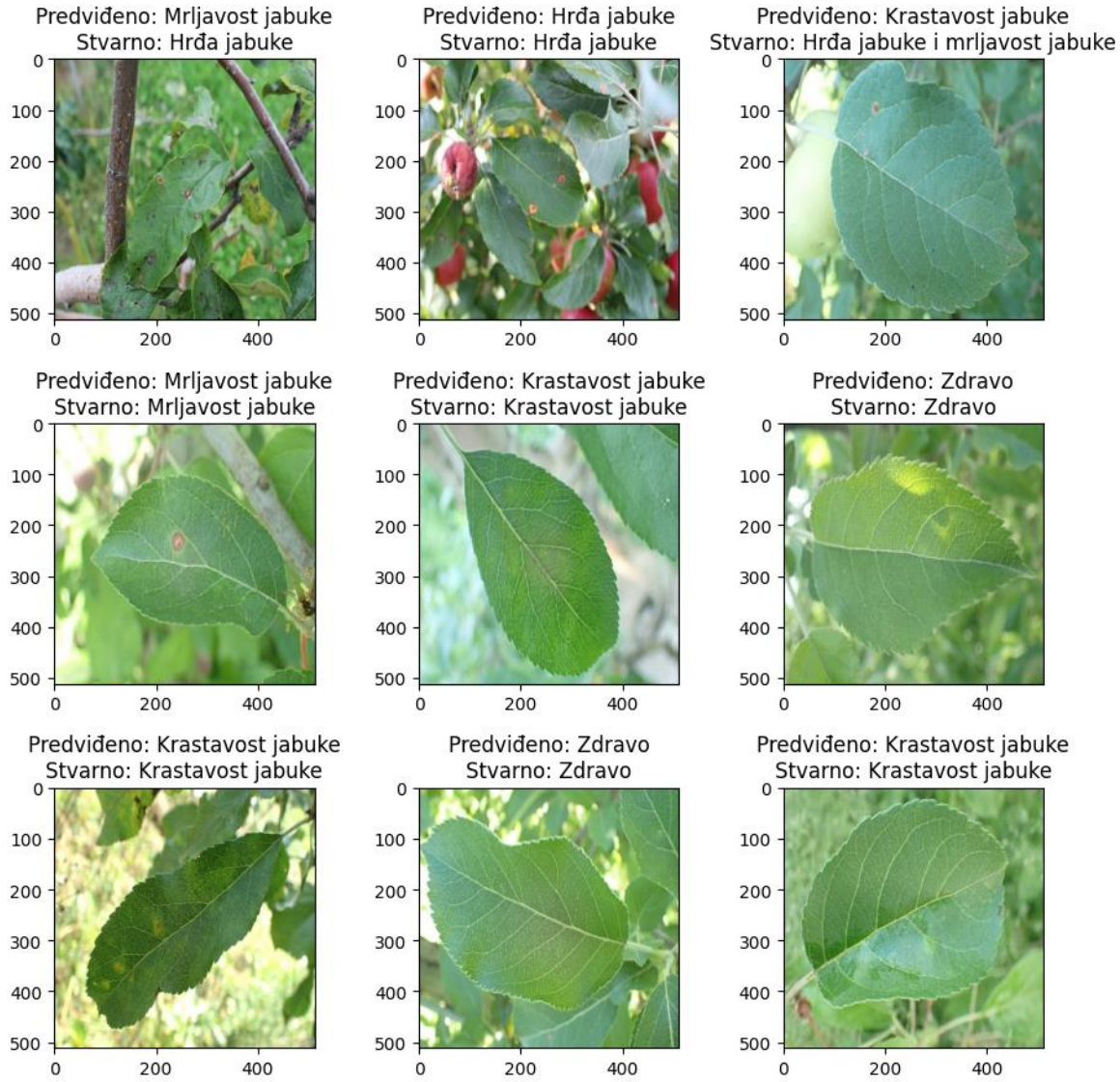
Slika 4.12. *Arhitektura druge inačice modela gdje se koristi složenija konvolucijska neuronska mreža*



Slika 4.13. *Validacijski gubitak i gubitka na treniranju te točnost na validacijskom skupu i točnost na skupu za treniranje – druga inačica modela*

Sa slike 4.13 može se vidjeti kako oko osme iteracije validacijski gubitak počinje usporava i prestaje padati kroz sljedeće iteracije. Treniranje je tada automatski prekinuto kako se model ne bi previše prilagodio trening podatcima. Korištenjem ovog modela postignuta je najveća točnost od 79.8%. Na slici 4.10 može se vidjeti devet nasumičnih slika iz trening skupa te njihovu

predviđenu i stvarnu klasu. Iako je situacija bolje nego kod prvog modela, neujednačeni podatci još uvijek stvaraju problem. To se može vidjeti kod trećeg slučaja sa slike 4.10. Isto tako, može se vidjeti kako još uvijek ima krivih klasifikacija i za slučajeve gdje ima mnogo primjera prilikom treniranja (prvi primjer sa slike 4.10), ali korištenjem ovog modela može se zapaziti značajan napredak u odnosu na prvi model.

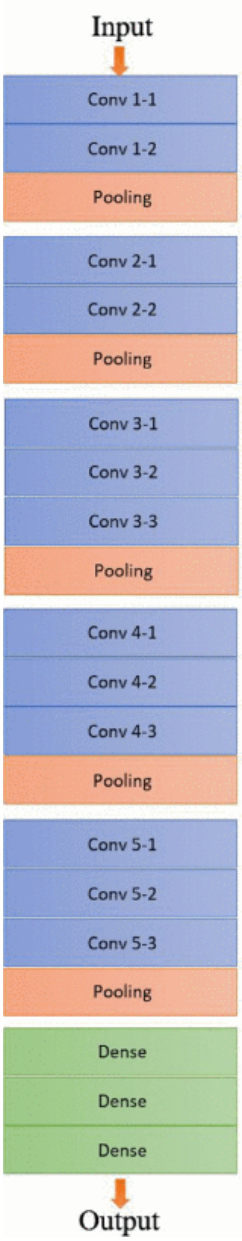


Slika 4.14. *Devet nasumičnih slika iz skupa za trening te njihove predviđene i stvarne klase – druga inačica modela*

4.3.3. Treća inačica modela - korištenje residualne neuronske mreže kao ekstraktora značajki

Nakon eksperimentiranja s vrlo jednostavnim i nešto složenijim modelom, testiran je model gdje se residualne neuronske mreže (engl. *residual networks*), točnije *ResNet101* arhitektura, koristi kao ekstraktor značajki. *ResNet101* jedna je od najkorištenijih konvolucijskih neuronskih mreža i daje vrlo dobre rezultate kod klasifikacije slika. Kako je opisano u [30], residualne neuronske

mreže istražuju rezidualne funkcije u odnosu na ulaze slojeva. *ResNet* grupira nekoliko identifikacijskih mapiranja (konvolucijskih slojeva bez akcije na početku), zanemaruje te razine i ponovno koristi aktivacije prethodnog sloja. *ResNet101* je verzija *ResNet-a* koja sadrži sto i jedan konvolucijski sloj te preko četrdeset i četiri milijuna parametara. *ResNet101* se uglavnom sastoji od pet vrsta konvolucijskih blokova. Pojednostavljena *ResNet101* arhitektura može se vidjeti na slici 4.15.



Slika 4.15. *Pojednostavljena arhitektura Resnet101 neuronske mreže, preuzeto iz [28]*

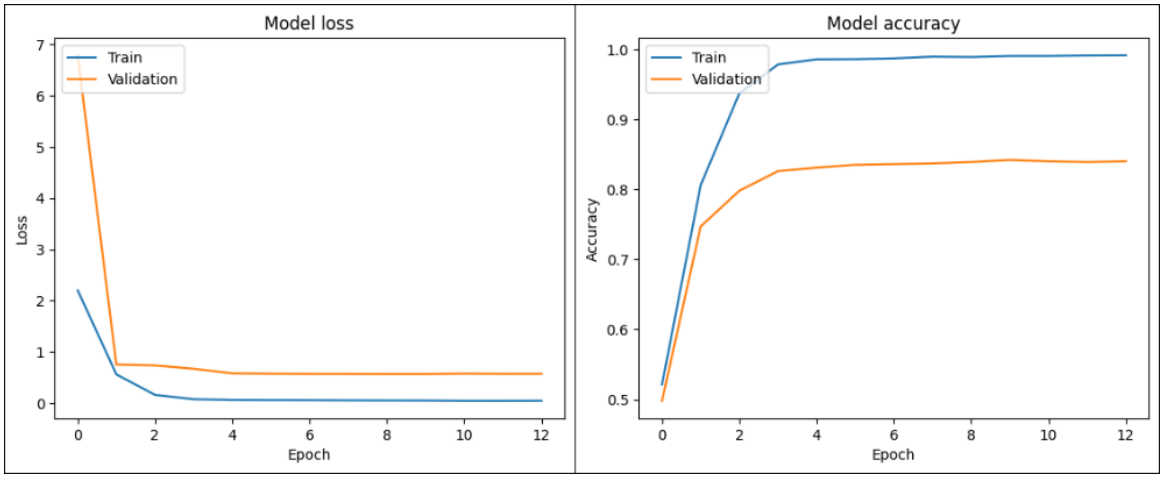
Prilikom učitavanja skupova podataka, dimenzija učitanih slika treba biti 224x224 piksela. Dodatno, veličina grupe je sada postavljena na dvadeset. Ostatak učitavanje podatak isti je kao u prethodnim primjerima pa neće biti prikazan. Na slici 4.16 može se vidjeti programski kod s kojim je dobiven najbolji rezultat prilikom eksperimentiranja s arhitekturom *Resnet101*.

```
resnet101 = ResNet101(weights='imagenet', include_top=False, input_shape=(img_height, img_width, 3))

flatten = layers.Flatten()(resnet101.output)
output = layers.Dense(12, activation='softmax')(flatten)
model = Model(inputs=resnet101.input, outputs=output)


model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

adapted_train_ds = train_ds.map(lambda x, y: (x, tf.one_hot(y, depth=12)))
adapted_val_ds = val_ds.map(lambda x, y: (x, tf.one_hot(y, depth=12)))
adapted_test_ds = test_ds.map(lambda x, y: (x, tf.one_hot(y, depth=12)))

def learning_rate_schedule(epoch):
    if epoch < 1:
        return 0.0001
    elif epoch < 4:
      return 0.00005
    else:
        return 0.0000001

lr_scheduler = LearningRateScheduler(learning_rate_schedule)


early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True
)


class_labels = np.concatenate(list(train_ds.map(lambda x, y: y).as_numpy_iterator())).astype(np.int32)
class_counts = np.bincount(class_labels)
total_samples = np.sum(class_counts)
class_weights = total_samples / (len(class_counts) * class_counts)


epochs = 20
history = model.fit(adapted_train_ds,
                    validation_data=adapted_val_ds,
                    epochs=epochs,
                    callbacks=[early_stopping, lr_scheduler],
                    class_weight=dict(enumerate(class_weights)))
```

Slika 4.16. *Programski kod za treniranje konvolucijske neuronske mreže koristeći ResNet101 arhitekturu za izvlačenje značajki*

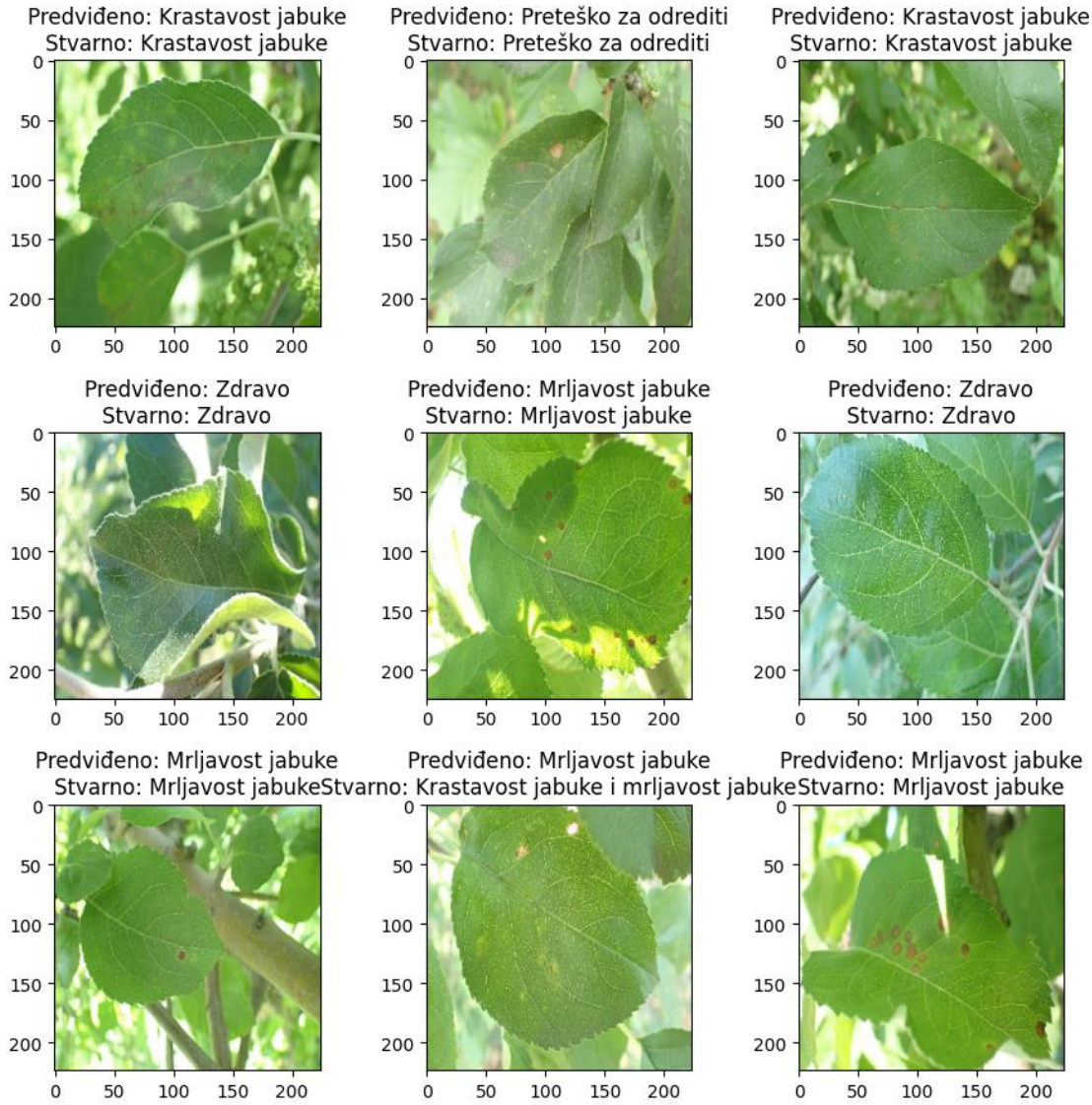
Kao što se može vidjeti sa slike 4.16, prvo se učitava prethodno obučen *ResNet101* model koji je dostupan putem *Kerasa*. Ovaj model se učitava sa svojim težinama treniranim na *ImageNet* skupu podataka. Argument *include_top=False* osigurava da se ukloni gornji sloj (klasifikacijski sloj). Gornji sloj je potrebno ukloniti jer se *ResNet101* koristi samo kao ekstraktor značajki, a na kraj se dodaje prilagođeni klasifikacijski sloj. Nakon toga, dodaju se dodatni slojevi na model. Prvo, izlazi iz *ResNet101* modela ravnaju se u jednodimenzionalni vektor pomoću *Flatten* sloja. Ovaj korak je potreban kako bi pripremili značajke za spomenuti prilagođeni klasifikacijski sloj. Posljednji sloj je potpuno povezani sloj s dvanaest izlaznih čvorova koji koristi *softmax* aktivacijsku funkciju. Ovaj sloj koristi se za klasifikaciju slika u jednu od dvanaest klasa. Model se tada prevodi s jednakim parametrima kao i kod prošla dva modela. Dodatno, kako bi se dobili bolji rezultati još se implementira rano zaustavljanje, raspored učenja (engl. *learning rate schedule*) te se računaju težine klasa. Kao što je već opisano u prošlom potpoglavlju, rano

zaustavljanje automatski zaustavlja treniranje kada se funkcija gubitka na validacijskom skupu prestane poboljšavati nakon definiranog broja iteracija. Raspored učenja određuje brzinu učenja u svakoj od iteracija. Kako prolaze iteracije, brzina učenja se smanjuje što pomaže u postizanju boljih rezultata tijekom treniranja. Proračunavanje težina klasa pomaže modelu da bolje tretira manje brojne klase i postigne bolje rezultate na svim klasama. Ovo je posebno pomaže kada su skupove podataka neuravnoteženi, tj. gdje neke klase imaju mnogo više primjera od drugih. Konačno, modelu se predaju skupovi podataka za trening i validaciju te se počinje s treniranjem. Na slici 4.17 može se vidjeti kretanje validacijskog gubitka i gubitka na treniranju te točnosti na validacijskom skupu te točnosti na skupu za treniranje kroz iteracije.



Slika 4.17. *Validacijski gubitak i gubitka na treniranju te točnost na validacijskom skupu i točnost na skupu za treniranje – model s ResNet101 arhitekturom kao ekstraktorom značajki*

Na slici 4.17 može se vidjeti kako poslije druge iteracije validacijski gubitak naglo usporava, ali smanjuje se stopa učenja te ipak ima napretka. Tek oko dvanaeste iteracije utvrđuje se kako više nema značajnog napretka i treniranje se zaustavlja kako se model ne bi previše prilagodio trening podatcima. Korištenjem ovog modela postignuta je najveća točnost od 85.2%. Na slici 4.18 može se vidjeti devet nasumičnih slika iz trening skupa te njihovu predviđenu i stvarnu klasu. Ovaj model pokazao je najbolje rezultate. Naravno, još postoji puno mjesta za napredak, posebno kod klasa koje imaju malo primjera kod treniranja. To se može vidjeti na osmom primjeru sa slike 4.18, model je predvidio da je list zaražen s mrljavošću jabuke, dok je stvarna bolest mrljavost jabuke i hrđa jabuke. Na kraju, pošto je ova inačica modela dala najbolje rezultate, koristit će se kao klasifikator u mobilnoj Android aplikaciji.



Slika 4.18. *Devet nasumičnih slika iz skupa za trening te njihove predviđene i stvarne klase – model s ResNet101 arhitekturom kao ekstraktorom značajki*

4.4. Rješenje dohvaćanja prognoze vremena za određenu geolokaciju

Točna vremenska prognoza izrazito je bitna kod tretiranja nasada raznim sredstvima. Treba proći određeno vrijeme kako bi se primijenjena sredstva osušila te kako bi ona počela djelovati. Za dohvaćanje vremenske prognoze korištena je *OpenWeather* API usluga dostupna na [31]. *OpenWeather* nudi detaljnu trenutnu prognozu, prognozu za sljedećih pet dana u koracima od 3 sata, vremenske mape i još mnogo toga za bilo koju lokaciju na planeti. Dodatno, podatke je moguće dohvatiti u nekoliko različitih oblika te u različitim mjernim jedinicama. Kako bi se usluga koristila potrebno je napraviti korisnički račun te generirati API ključ koji se kasnije koristi pri dohvaćanju podataka. Vremenska prognoza dohvaća se na temelju geografske širine (engl. *latitude*) i geografske visine (engl. *longitude*). U slučaju kada se vremenska prognoza dohvaća za trenutnu lokaciju, koriste se geografska širina i geografska visina mobilnog uređaja.

U slučaju kada korisnik dohvaća vremensku prognozu za neku drugu lokaciju, *OpenWeather* API usluga omogućuje vraćanje geografske širine i geografske visine na osnovu predanog imena grada. Na slici 4.19 mogu se vidjeti metode *createWeatherRequest()* i *createCityLocationRequest()* metode koje služe za stvaranje zahtjeva za vremensku prognozu, odnosno za lokaciju predanog grada.

```
private void createWeatherRequest(String latitude, String longitude){
    RequestParams params = new RequestParams();

    params.put("lat", latitude);
    params.put("lon", longitude);
    params.put("units", "metric");
    params.put("appid", API_KEY);

    fetchData(params, WEATHER_URL);
}

private void createCityLocationRequest(String cityName){
    RequestParams params = new RequestParams();
    params.put("q", cityName);
    params.put("limit", 1);
    params.put("appid", API_KEY);

    fetchData(params, CITY_LOCATION_URL);
}
```

Slika 4.19. *Metode createWeatherRequest() i createCityLocationRequest()*

Kao što se može vidjeti sa slike 4.19, *createWeatherRequest()* služi za stvaranje zahtjeva za vremenskim podacima na temelju geografske širine i geografske dužine. Prima dvije ulazne vrijednosti, *latitude* i *longitude*, koje predstavljaju koordinate željene lokacije. Prvo se stvara objekt *RequestParams* koji će se koristiti za postavljanje parametara HTTP zahtjeva. Ovaj objekt se koristi kako bi se definirali parametri koje će API prihvatiti. Zatim se koriste metode *put()* objekta *params* kako bi se postavili parametri za zahtjev. Konkretno, postavljaju se *lat*, *lon*, *units* i *appid* parametri. Prva dva parametra su geografska širina i dužina, *units* postavlja jedinice na metrički sustav, a *appid* je API ključ koji se koristi za identifikaciju i autorizaciju pristupa API-ju. Na kraju, poziva se metoda *fetchData()* koja će poslati HTTP zahtjev s postavljenim parametrima prema *WEATHER_URL*. Druga metoda, odnosno *createCityLocationRequest()*, služi za stvaranje zahtjeva za dohvaćanje lokacije (koordinata) grada na temelju imena grada. Metoda prima argument *cityName*, koji predstavlja ime grada čiju lokaciju želimo dobiti. Ponovno se stvara objekt *RequestParams* kako bi se postavili parametri HTTP zahtjeva. Zatim se koristi metoda *put()* kako bi se postavili parametri za zahtjev. Ovdje se postavljaju *q*, *limit*, i *appid* parametri. Parametar *q* je ime grada za koji želimo dobiti koordinate, *limit* ograničava broj rezultata na jedan, a *appid* je API ključ. Na kraju, poziva se metoda *fetchData()* koja će poslati

HTTP zahtjev s postavljenim parametrima prema *CITY_LOCATION_URL*. Na slici 4.20 može se vidjeti metoda *fetchData()*.

```
private void fetchData(RequestParams params, String URL){
    AsyncHttpClient client = new AsyncHttpClient();
    client.get(URL, params, new JsonHttpResponseHandler(){

        @Override
        public void onStart() {
            super.onStart();
            loadingDialog.showLoadingDialog();
        }
        @Override
        public void onSuccess(int statusCode, Header[] headers, JSONObject response) {
            super.onSuccess(statusCode, headers, response);
            Toast.makeText(getActivity(), text "Data fetched successfully ", Toast.LENGTH_SHORT).show();
            WeatherData weatherData = WeatherData.fromJson(response);
            currentlySetWeatherData = weatherData;
            updateUI(weatherData);
        }
        @Override
        public void onSuccess(int statusCode, Header[] headers, JSONArray response) {
            super.onSuccess(statusCode, headers, response);
            if(response.length() == 0){
                Toast.makeText(getActivity(), text "Please check if you spelled City name correctly", Toast.LENGTH_SHORT).show();
                return;
            }

            Toast.makeText(getActivity(), text "Data fetched successfully ", Toast.LENGTH_SHORT).show();
            CityInformation cityInformation = CityInformation.fromJson(response);
            String latitude = Double.toString(cityInformation.getLatitude());
            String longitude = Double.toString(cityInformation.getLongitude());
            createWeatherRequest(latitude, longitude);
        }
        @Override
        public void onFailure(int statusCode, Header[] headers, Throwable throwable, JSONObject errorResponse) {
            super.onFailure(statusCode, headers, throwable, errorResponse);
            Toast.makeText(getActivity(), text "Data NOT fetched", Toast.LENGTH_SHORT).show();

        }
        @Override
        public void onFailure(int statusCode, Header[] headers, Throwable throwable, JSONArray errorResponse) {
            super.onFailure(statusCode, headers, throwable, errorResponse);
            Toast.makeText(getActivity(), text "Data NOT fetched", Toast.LENGTH_SHORT).show();

        }
        @Override
        public void onFinish() {
            super.onFinish();
            loadingDialog.dismissLoadingDialog();
        }
    });
}
```

Slika 4.20. *Metoda fetchData()*

Kao što se može vidjeti sa slike 4.20, metoda *fetchData()* prima dva argumenta. Prvi je objekt tipa *RequestParams* koji sadrži parametre koji će se poslati u HTTP zahtjevu. Drugi parametar je tipa *String* i predstavlja *URL* prema kojem će se HTTP zahtjev izvršiti. Unutar metode, stvara se objekt *AsyncHttpClient*, koji je dio biblioteke za asinkronu obradu HTTP zahtjeva. Ovaj objekt će se koristiti za izvođenje HTTP zahtjeva prema zadatom *URL-u*. Nakon toga izvršava HTTP GET zahtjev. Prije nego što se zahtjev pošalje, poziva se metoda *OnStart()*. Unutar metode

OnStart() prikazuje se dijalog koji korisniku daje do znanja da se podatci dohvaćaju. Ako je HTTP zahtjev uspješan, poziva se metoda *OnSuccess()*. Ovdje se prikazuje kratka obavijest da su podatci uspješno dohvaćeni, a zatim se, ovisno o odgovoru, podaci iz JSON odgovora pretvaraju u odgovarajući objekt, tj. *WeatherData* ili *CityInformation*. U slučaju kada je odgovor pretvoren u objekt klase *CityInformation*, odmah se poziva metoda *createWeatherRequest()* koja će dohvatiti lokaciju za predanu geografsku širinu i geografsku visinu. S druge strane, kada je odgovor pretvoren u objekt klase *WeatherData*, poziva se metoda *updateUI()* kojoj se predaje kreirani objekt. Metoda *updateUI()* služi za prikazivanje vremenske prognoze i može se vidjeti na slici 4.21.

```
private void updateUI(WeatherData weatherData){
    tvTemperature.setText(weatherData.getCurrentTemperature());
    tvCity.setText(weatherData.getCity());
    tvWeatherState.setText(weatherData.getCurrentWeatherType());
    int resourceId = getResources().getIdentifier(weatherData.getCurrentIcon(), defType: "drawable",
            getActivity().getPackageName());
    ivWeatherState.setImageResource(android.R.color.transparent);
    ivWeatherState.setImageResource(resourceId);
    JSONObject[] weatherForecast = weatherData.getWeatherForecastAsJSONObjectList();

    //Add image resource id according to weather state id
    for(int i = 0; i < weatherForecast.length; i++){
        try {
            int weatherStateId = weatherForecast[i].getJSONArray( name: "weather").getJSONObject( index: 0).getInt( name: "id");
            int imageId = getResources().getIdentifier(WeatherData.updateWeatherIcon(weatherStateId), defType: "drawable",
                    getActivity().getPackageName());
            weatherForecast[i].getJSONArray( name: "weather").getJSONObject( index: 0).put( name: "image_id", imageId);

        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
    weatherAdapter = new WeatherAdapter((weatherForecast));
    rvWeatherForecast.setHasFixedSize(true);
    rvWeatherForecast.setLayoutManager(new LinearLayoutManager(getContext(), LinearLayoutManager.HORIZONTAL, reverseLayout: false));
    rvWeatherForecast.setAdapter(weatherAdapter);
}
```

Slika 4.21. *Metoda updateUI()*

Sa slike 4.21 može se vidjeti kako se prikazuje trenutno vrijeme. Nakon toga, identifikator stanja vremena mijenja se s identifikatorom resursa slike s ikonom vremena za svaki element u vremenskoj prognozi. Na kraju se vremenska prognoza predaje *WeatherAdapter-u* je zadužen za popunjavanje *RecyclerView-a*, tj. prikaz vremenske prognoze u *WeatherForecastFragment-u*.

4.5. Postupak stvaranja preporuka za liječenje

Već je spomenuto kako je potrebno kreirati sustav preporuka za liječenje kako bi korisnik mogao dobiti preporuke za liječenje svog nasada. Preporuke su implementirane kroz niz poruka,

odnosno zahtjeva korisnika i odgovora sustava. Korisnik može zatražiti preporuku čime liječiti nasad, kada liječiti nasad te provjeriti učinkovitost liječenja.

Kada korisnik zatraži preporuku čime liječiti nasad, poruka se sprema u bazu podataka i poziva se metoda *createTreatmentRecommendation().* Metoda *createTreatmentRecommendation()* može se vidjeti na slici 4.22.

```
private void createTreatmentRecommendation() {
    List<Float> lastDiseaseConfidences = getDiseaseConfidencesFromLastCheck();
    String className = DiseaseClassificator.getDiseaseName(getHighestConfidenceIndex(lastDiseaseConfidences));
    SimpleMessage message = new SimpleMessage( message: "Treatment recommendation:\n\n\t\t" +
            DiseaseClassificator.getDiseaseTreatmentRecommendation(className) +
            "\n\nWarning: read manufacturer's instructions regarding dosage and intervals between treatments", recommendationMessageType: 0);
    addRecommendationMessage(message);
}
```

Slika 4.22. *Metoda createTreatmentRecommendation()*

Naravno, potrebno je znati čime je nasad zaražen pa će se prvo pronaći posljednje spremljene vjerojatnosti pripadnosti različitim klasama. Vrijednosti pripadnosti klasama pronalaze se koristeći metodu *getDiseaseConfidencesFromLastCheck()* koja se može vidjeti na slici 4.23.

```
private List<Float> getDiseaseConfidencesFromLastCheck() {

    List<SimpleMessage> messages = recommendationSystem.getRecommendationMessages();

    for(int i = messages.size()-2; i >= 0; i--){
        if (messages.get(i).getRecommendationMessageType() == 30){
            return messages.get(i).getDiseaseConfidences();
        }
    }

    return recommendationSystem.getStartDiseaseConfidences();
}
```

Slika 4.23. *Metoda getDiseaseTreatmentRecommendation*

Metoda *getDiseaseConfidencesFromLastCheck()* iterirat će kroz sve poruke sustava preporuka kako bi pronašla posljednju put gdje korisnik provjerava učinkovitost liječenja. Ta poruka se traži jer se u tom slučaju ponovno učitava slika koja se predaje model za klasifikaciju te se dobivaju vjerojatnosti pripadnosti različitim klasama kako bi se provjerila učinkovitost liječenja. Te vjerojatnosti se spremaju u bazi podataka zajedno s porukom. Ako je takva poruka pronađena, vraćaju se vjerojatnosti pripadnosti različitim klasama iz te poruke. U suprotnom, vraćaju se vjerojatnosti pripadnosti klasama bolesti koje su dobivene prilikom prve klasifikacije kada je kreiran sustav preporuka. Nakon toga, dohvaća se ime klase s najvećom vjerojatnosti i predaje se metodi *getDiseaseTreatmentRecommendation()* koja vraća preporuke za liječenje. Metoda *getDiseaseTreatmentRecommendation()* može se vidjeti na slici 4.24.

```
public static String getDiseaseTreatmentRecommendation(String className){

    int classIndex = getClassIndex(className);

    final String[] diseaseTreatmentRecommendation = {"Consult with the expert since it is to complex to classify", //to complex to tell
            "Treat with Azoxystrobin and Difenconazole-Based Fungicides and Copper-Based Fungicides\n\nExamples:\n" +
                    "-Delan 700 WDG\n-Captan 50\n-Polyram DF", //frog eye leaf spots
            "It is hard to tell but treating with Azoxystrobin and Difenconazole-Based Fungicides and Copper-Based Fungicides should help" +
                    "\n\nExamples:\n-Chromosul 80", // complex frog eye leaf spots
            "", //healthy
            "Treat with Sulfur\n\nExamples:\n-Chromosul 80", //powdery mildew
            "It is hard to tell but treating with sulfur should help\n\nExamples:\n-Chromosul 80", //complex powdery mildew
            "Treat with Copper-Based and Mancozeb-Based Fungicides\n\nExamples:\n-Strobe\n-Topaz\n-Vectra\n-Tsineba", //rust
            "It is hard to tell but treating with Copper-Based and Mancozeb-Based Fungicides should help\n\nExamples:\n-Strobe" +
                    "\n-Topaz", //complex rust
            "Treat with Copper-Based Fungicides\n\nExamples:\n-Poliram", // rust frog eye leaf spots
            "Treat with Propiconazole-Based and Captan-Based Fungicides\n\nExamples:\n-Kastor\n-Argo\n-Bellis", //scab
            "Treat with the mix of Azoxystrobin and Difenconazole-Based and Propiconazole-Based Fungicides but consult with the" +
                    " expert for the best mix", //scab frog eye leaf spots
            "It is hard to tell but treating with mix of Azoxystrobin and Difenconazole-Based and Propiconazole-Based Fungicides" +
                    " should help. Consult with the expert for the best mix"}; //complex scab frog eye leaf spots

    return diseaseTreatmentRecommendation[classIndex];
}
```

Slika 4.24. *Metoda getDiseaseTreatmentRecommendation()*

Na osnovu imena klase, metoda vraća preporuku za liječenje. Na kraju se poruka, odnosno preporuka sprema se u bazu podataka koristeći metodu *addRecommendationMessage()*. Metoda *addRecommendationMessage()* će biti dodatno objašnjena u potpoglavlju 4.7.1.

Kada korisnik zatraži preporuku kada liječiti nasad, poruka se sprema u bazu podataka i dohvaća se vremenska prognoza za lokaciju nasada. Dohvaćanje vremenske prognoze objašnjeno je u prošlom potpoglavlju. Kada je vremenska prognoza uspješno dohvaćena, poziva se metoda *createTreatmentTimeRecommendationBasedOnWeather()* koja se može vidjeti na slici 2.25.

```
private void createTreatmentTimeRecommendationBasedOnWeather(WeatherData weatherData){
    List<Integer> weatherIds = weatherData.getWeatherIdsList();
    List<String> date_time = weatherData.getDateTimeList();
    //8 is 1 day since forecast is in steps of 3 hours
    List<String> treatmentWindow = findTreatmentWindow(weatherIds, date_time,  treatingWindowSize: 16);
    int mostOccurring;

    if(treatmentWindow.size() == 0){
        mostOccurring = weatherData.findMostOccurringBadWeatherId(weatherIds);
    }
    else {
        mostOccurring = weatherData.findMostOccurringWeatherId(weatherIds);
    }

    String url = weatherData.getWeatherImageUrl(mostOccurring);
    String forecastMessage = weatherData.getWeatherForecastMessage(mostOccurring);
    SimpleMessage message;

    if(treatmentWindow.size() == 0){
        message = new SimpleMessage( message: "Unfortunately there is no window where plantation can be treated. \nThere will be " + forecastMessage

    }
    else {
        message = new SimpleMessage( message: "There is window from: " + treatmentWindow.get(0) + " - " + treatmentWindow.get(treatmentWindow.size() - 1)
    }
    addRecommendationMessage(message);
}
```

Slika 4.25. *Metoda createTreatmentTimeRecommendationBasedOnWeather()*

Metoda prima objekt tipa *WeatherData* gdje je spremljena vremenska prognoza. Iz vremenske prognoze izdvajamo identifikatore stanja vremena i datume u zasebne liste koje predajemo

metodi *findTreatmentWindow()*. Osim listi, metoda prima i trajanje prozora povoljnog vremena. Traži se prozor od dva dana gdje neće biti kiše, magle ili bilo kojeg drugog nepovoljnog vremena. Metoda iterira kroz identifikatore stanja vremena i traži prozor s odgovarajućim vremenom. U slučaju da takav prozor postoji, vraća se lista s datumima tog prozora. Ukoliko je vrijeme povoljno duže od dva dana, lista će zadržavati te datume. S druge strane, kada takav prozor ne postoji, vraća se prazna lista. Metoda *findTreatmentWindow()* može se vidjeti na slici 4.26.

```
public List<String> findTreatmentWindow(List<Integer> weatherIds, List<String> date_times, int treatingWindowSize) {
    List<String> treatingWindow = new ArrayList<>();

    for (int i = 0; i < weatherIds.size(); i++) {
        int currentValue = weatherIds.get(i);
        String currentString = date_times.get(i);

        if (currentValue >= 800) {
            treatingWindow.add(currentString);
        } else {
            if(treatingWindow.size() < treatingWindowSize){
                treatingWindow.clear();
            }
            else{
                break;
            }
        }
    }

    if(treatingWindow.size() < treatingWindowSize){
        treatingWindow.clear();
    }

    return treatingWindow;
}
```

Slika 4.26. *Metoda findTreatmentWindow()*

Kada se zna postoji li prozor s povoljnim vremenom, kreiramo preporuku s odgovarajućom porukom. Dodatno, ova poruka sadrži sliku s najčešćim vremenom. Pronalazi se identifikator stanja vremena koje se najčešće pojavljuje te se u poruku dodaje identifikator prikladnog resursa, odnosno slike. Na kraju se poruka, odnosno preporuka sprema se u bazu podataka.

Kada korisnik želi provjeriti učinkovitost liječenja, prvo mora učitati novu sliku lista jabuke kako bi se odredilo trenutno stanje nasada. Slika se učitava, predaje klasifikacijskom modelu i dobivaju se vjerojatnosti pripadnosti različitim klasama, odnosno bolestima. Nakon toga poruka, slika i vjerojatnosti pripadnosti klasama spremaju se u bazu podataka i poziva se metoda *createTreatmentEffectivenessAnswer()*. Metoda *createTreatmentEffectivenessAnswer()* može se vidjeti na slici 4.27.

```
private void createTreatmentEffectivenessAnswer(List<Float> currentDiseaseConfidences){
    List<Float> pastDiseaseConfidences = getDiseaseConfidencesFromLastCheck();
    String currentDisease = DiseaseClassificator.getDiseaseName(getHighestConfidenceIndex(currentDiseaseConfidences));
    String lastClassifiedDisease = DiseaseClassificator.getDiseaseName(getHighestConfidenceIndex(pastDiseaseConfidences));

    if (currentDisease == "healthy"){
        addRecommendationMessage(new SimpleMessage( message: "Great news! Treatment worked and your plantation is now healthy",
                recommendationMessageType: 1));
    } else if(currentDisease != lastClassifiedDisease){
        String currentDiseaseDescription = DiseaseClassificator.getDiseaseClassDescription(currentDisease);
        String lastClassifiedDiseaseDescription = DiseaseClassificator.getDiseaseClassDescription(lastClassifiedDisease);
        SimpleMessage message = new SimpleMessage( message: "Looks like your plantation is now infected with:\n\n" +
                currentDiseaseDescription +
                "\n\nPreviously it was infected with: " + lastClassifiedDiseaseDescription + "\n\nTreatment didn't work",
                recommendationMessageType: 2);
        addRecommendationMessage(message);
    }
    else if(currentDisease == lastClassifiedDisease){
        float treatmentEffectivenessInPercentage = getTreatmentEffectiveness(
                pastDiseaseConfidences.get(DiseaseClassificator.getClassIndex( className: "healthy")),
                currentDiseaseConfidences.get(DiseaseClassificator.getClassIndex( className: "healthy")));
        SimpleMessage message = new SimpleMessage( message: "Your plantation is: \n\n" +
                getTreatmentEffectivenessMessage(treatmentEffectivenessInPercentage),
                getTreatmentEffectivenessImageUrl(treatmentEffectivenessInPercentage), recommendationMessageType: 40);
        addRecommendationMessage(message);
    }
}
```

Slika 4.27. *Metoda createTreatmentEffectivenessAnswer()*

Kao što se može vidjeti sa slike 4.27, metoda prima vjerojatnosti da nasad boluje od pojedine bolesti. Nakon toga pronalaze se vjerojatnosti od posljednje provjere pozivom metode *getDiseaseConfidencesFromLastCheck()*. Nakon toga, može doći do tri slučaja. Prvi slučaj je da je nasad sada zdrav. Tada se prikladna poruka prikazuje i sprema u bazu podataka. Drugi slučaj je kada se bolest sada i bolest prilikom posljednje provjere razlikuju. U tom slučaju korisnika se obavještava kako je nasad sada zaražen drugom bolešću. Naravno, poruka se sprema u bazu. Posljednji slučaj je kada nasad kod prošle provjere i sada boluje od iste bolesti. Tada se poziva metoda *getTreatmentEffectiveness()* koja računa učinkovitost liječenje. Ovisno o učinkovitosti, tj. rezultatu koji vrati metoda, bit će prikazana odgovarajuća poruka. Kao i kod prethodnih slučajeva, poruka se sprema u bazu podataka. Metoda *getTreatmentEffectiveness()* dodatno je opisana u potpoglavlju 4.7.4.

4.6. Programsko rješenje na strani korisnika

4.6.1. Prikaz postupka registriranja i prijave korisnika i promjene lozinke

Prilikom pokretanja aplikacije prikazuje se zaslon gdje korisnik može birati između registracije, prijave ili promjene lozinke. Ukoliko korisnik nema korisnički račun, treba se registrirati. Kod pritiska gumba *Signup* provjerava se ako su sva polja popunjena te ako formati podataka u svim

poljima prikladni. Na primjer, ako polje za e-poštu stvarno sadrži e-poštu. Ako je sve u redu, poziva se metoda *signupUser()* koja je prikazana na slici 4.28.

```
private void signupUser(String firstName, String lastName, String email, String password){
    mAuth.createUserWithEmailAndPassword(email, password)
            .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    progressBar.setVisibility(View.GONE);

                    if (task.isSuccessful()) {
                        // Sign in success, update UI with the signed-in user's information
                        FirebaseUser user = mAuth.getCurrentUser();
                        UserProfile userProfile = new UserProfile(firstName, lastName, email);
                        DatabaseReference dbReference = FirebaseDatabase.getInstance().getReference( path: "User");

                        dbReference.child(user.getUid()).setValue(userProfile).addOnCompleteListener(new OnCompleteListener<Void>() {
                            @Override
                            public void onComplete(@NonNull Task<Void> task) {

                                if (task.isSuccessful()){
                                    Toast.makeText( context: SignupActivity.this, text: "Account successfully created",
                                            Toast.LENGTH_SHORT).show();

                                    Intent intent = new Intent(getApplicationContext(), LoginActivity.class);
                                    startActivity(intent);
                                    finish();
                                }
                                else{
                                    Toast.makeText( context: SignupActivity.this, text: "Registration failed, something went wrong",
                                            Toast.LENGTH_SHORT).show();
                                }
                            }
                        });
                    } else {
                        // If sign in fails, display a message to the user.
                        Toast.makeText( context: SignupActivity.this, text: "Authentication failed.",
                                Toast.LENGTH_SHORT).show();
                    }
                }
            });
}
```

Slika 4.28. *Metoda signupUser()*

Metoda signupUser() prima podatke iz forme za registraciju. Nakon toga, poziva se metoda *createUserWithEmailAndPassword()* iz *Firebase* objekta kako bi se pokušalo stvoriti novi korisnički račun s unesenom e-poštom i lozinkom. Metodi se predaju e-pošta i lozinka. Dodatno, dodaje se *OnCompleteListener* koji osluškuje završetak operacije stvaranja korisničkog računa. Ovaj slušatelj će biti pozvan kada operacija bude dovršena, bez obzira je li uspjela ili nije. Kada je operacija dovršena, provjerava se ako je uspješna. Ako je uspješna, korisnički podatci spremaju se u bazu podataka i korisnik se preusmjerava na zaslon za prijavu. U suprotnom, ispisuje se prikladna *Toast* poruka.

U slučaju kada korisnik već ima korisnički račun, treba otići na zaslon s prijavom te upisati e-poštu i lozinku te pritisnuti gumb *Login*. Nakon toga provjerava se ako su sva polja popunjena te ako su formati podataka u redu. Ako su svi podatci ispravno uneseni, poziva se metoda

loginUser(). Metoda prima dva parametra, odnosno e-poštu i lozinku. Na početku se poziva metoda *signInWithEmailAndPassword*() iz *Firebase* objekta kako bi se pokušala prijava korisnika s unesenom email adresom i lozinkom. Ovdje se također dodaje *OnCompleteListener* koji osluškuje završetak operacije prijave. Ako je operacija uspješna provjerava se da li je korisnički račun aktiviran. Ako je korisnički račun aktiviran, korisnik se preusmjerava na glavni zaslon. S druge strane, kada korisnički račun nije aktiviran, šalje se mail za aktivaciju i ispisuje se prikladna poruka. Metoda *loginUser()* može se vidjeti na slici 4.29.

```
private void loginUser(String email, String password){
    mAuth.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    progressBar.setVisibility(View.GONE);
                    if (task.isSuccessful()) {
                        FirebaseUser user = mAuth.getCurrentUser();

                        if (user.isEmailVerified()){
                            Toast.makeText( context: LoginActivity.this,  text: "Login Successful", Toast.LENGTH_SHORT).show();

                            Intent intent = new Intent(getApplicationContext(), MainActivity.class);
                            intent.putExtra( name: "tab",  value: 1);
                            startActivity(intent);
                            finish();
                        }
                        else{
                            user.sendEmailVerification();
                            Toast.makeText( context: LoginActivity.this,  text: "Please check your email to verify your account",
                                    Toast.LENGTH_LONG).show();
                        }

                    } else {
                        // If sign in fails, display a message to the user.
                        Toast.makeText( context: LoginActivity.this,  text: "Authentication failed.",
                                Toast.LENGTH_SHORT).show();
                    }
                }
            });
}
```

Slika 4.29. *Metoda loginUser()*

Dodatno, prilikom pokretanja zaslona za prijavu poziva se metoda *OnStart()*. Unutar te metode dohvaća se trenutni korisnik. Ako je korisnik nedavno prijavljen u aplikaciju i ima aktiviran korisnički račun, automatski će biti preusmjeren na glavni zaslon. Ako je korisnik obavio registraciju, a nije aktivirao korisnički račun, ispisat će se prikladna *Toast* poruka. Metoda *OnStart()* može se vidjeti na slici 4.30.

```java
@Override
public void onStart() {
    super.onStart();
    // Check if user is signed in (non-null) and update UI accordingly.
    FirebaseUser currentUser = mAuth.getCurrentUser();
    if(currentUser != null && currentUser.isEmailVerified()){
        Intent intent = new Intent(getApplicationContext(), MainActivity.class);
        intent.putExtra( name: "tab", value: 1);
        startActivity(intent);
        finish();
    }
    else if (currentUser != null && !currentUser.isEmailVerified()){
        Toast.makeText( context: LoginActivity.this, text: "Please check your email to verify your account", Toast.LENGTH_LONG).show();
    }
}
```

Slika 4.30. *Metoda OnStart()*

Također, korisnik ima mogućnost promjene lozinke. Na zaslonu za promjenu lozinke treba upisati e-poštu s kojim se registrirao i pritisnuti *Send email* gumb. Pritiskom na gumb poziva se metoda *resetPassword()* prikazana na slici 4.31. U metodi se prvo provjera ako je polje za e-poštu popunjeno te da je stvarno upisana validna e-pošta. Nakon toga, poziva se *sendPasswordResetEmail()* iz *Firebase* objekta i dodaje se *OnCompleteListener* na operaciju. Na kraju se prikazuje prikladna *Toast* poruka ovisno o uspjehu operacije.

```java
private void resetPassword(){
    String email = etEmail.getText().toString().trim();

    if (TextUtils.isEmpty(email)) {
        etEmail.setError("Email is required");
        etEmail.requestFocus();
        return;
    }

    if (!Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
        etEmail.setError("This is not valid Email format");
        etEmail.requestFocus();
        return;
    }

    progressBar.setVisibility(View.VISIBLE);
    auth.sendPasswordResetEmail(email).addOnCompleteListener(new OnCompleteListener<Void>() {
        @Override
        public void onComplete(@NonNull Task<Void> task) {
            progressBar.setVisibility(View.GONE);
            if(task.isSuccessful()){
                Toast.makeText( context: ResetPasswordActivity.this, text: "Check your email to reset password",
                        Toast.LENGTH_LONG).show();
            }
            else{
                Toast.makeText( context: ResetPasswordActivity.this, text: "Something went wrong, please try again",
                        Toast.LENGTH_LONG).show();
            }
        }
    });
}
```

Slika 4.31. *Metoda resetPassword*

4.6.2. Prikaz glavnog zaslona aplikacije

Kod pokretanja *MainActivity-a* poziva se metoda *onCreate()* koja se može vidjeti na slici 4.32.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    auth = FirebaseAuth.getInstance();
    user = auth.getCurrentUser();
    btnLogout = findViewById(R.id.btnLogout);
    tvUser = findViewById(R.id.test);

    Bundle bundle = getIntent().getExtras();
    int tab = bundle.getInt( key: "tab");
    bottomNavigationView = findViewById(R.id.bnvBottomNavigation);
    bottomNavigationView.setOnItemSelectedListener(this);

    if(tab == 1){
        bottomNavigationView.setSelectedItemId(R.id.tab1);
    }
    else if(tab == 2){
        bottomNavigationView.setSelectedItemId(R.id.tab2);
    }
    else if(tab == 3){
        bottomNavigationView.setSelectedItemId(R.id.tab3);
    }
    else{
        bottomNavigationView.setSelectedItemId(R.id.tab4);
    }

    if (user == null){
        Intent intent = new Intent(getApplicationContext(), LoginActivity.class);
        startActivity(intent);
        finish();
    }
    else{
        DatabaseReference databaseReference = FirebaseDatabase.getInstance().getReference( path: "User").child(user.getUid());

        databaseReference.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                UserProfile user = snapshot.getValue(UserProfile.class);
                tvUser.setText("Welcome back " + user.firstName + "!");
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                Toast.makeText( context: MainActivity.this, error.toString(), Toast.LENGTH_SHORT).show();
            }
        });
    }

    btnLogout.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            FirebaseAuth.getInstance().signOut();
            Intent intent = new Intent(getApplicationContext(), LoginActivity.class);
            startActivity(intent);
            finish();
        }
    });
}
```

Slika 4.32. *Metoda onCreate*

Sa slike 4.32 može se vidjeti kako se u metodi *onCreate()* dohvaća trenutni korisnik, definiraju elementi te se postavlja navigacijska traka. Isto tako, dohvaćaju se podatci koji su preneseni u aktivnost putem *Intent* objekta. Točnijem, dohvaća se vrijednost *integer-a* pod imenom *tab* koji određuje koji fragment, odnosno koja kartica će biti prikazana. Na primjer, ako je korisnik preusmjeren s prijave, bit će prikazan *InfoFragment*. Dodatno, prilikom pokretanja *MainActivity-a* provjerava se ako je korisnik prijavljen. U slučaju kada je prijavljen dohvaćaju se njegovi podatci iz baze podataka. S druge strane, kada je trenutni korisnik *null*, korisnik se preusmjerava na zaslon za prijavu. Korištenjem navigacijske trake može se kretati između četiri zaslona, odnosno četiri fragmenta. Već spomenuti *InfoFragment* koji služi kao zaslon za informiranje korisnika o problematici podržanih bolesti i upoznavanje mobilnom aplikacijom. *InfoFragment* može se vidjeti na slici 5.4. Pritiskom gumba *Classify* na navigacijsko traci, prikazuje se fragment *ClassifyDiseaseFragment* vidljiv na slici 5.5. Pomoću *ClassifyDiseaseFragment-a* korisniku je omogućeno učitavanje slike i prepoznavanje o kojoj se bolesti radi te kreiranje novog sustava preporuka za liječenje. Nadalje, pritiskom gumba *Recommendations* na navigacijsko traci, prikazuje se *RecommendationSystemsFragment* gdje se mogu vidjeti svi kreirani sustavi preporuka za liječenje. *RecommendationSystemsFragment* može se vidjeti na slici 5.8. Na kraju, Pritiskom gumba *Forecast* na navigacijsko traci, prikazuje se *WeatherForecastFragment* koji se može vidjeti na slici 5.8. *WeatherForecastFragment* korisniku omogućuje pregled vremenske prognoze.

4.6.3. Prikaz zaslona sustava preporuka za liječenje

Prilikom kreiranja novog sustava preporuka za liječenje ili otvaranja postojećeg, poziva se metoda *onCreate()* za *RecommendationsActivity*. Prilikom pokretanja aktivnosti, moraju se dodati dodatni podatci *Intent* objektu. Dodaje se *recommendation_system_id* koji predstavlja identifikator sustava preporuka kako bi se znalo koji sustav preporuka treba prikazati. Na primjer, na slici 4.33 može se vidjeti *openCreateRecommendationSystem()* metoda koja se poziva nakon uspješnog kreiranja sustava preporuka za liječenje. U *onCreate()* metodi dohvaćaju se i prikazuju podatci za predani sustav preporuka. Dodatno, definiraju se svi elementi i postavljaju se *OnClickListener-i* na sve gumbe.

```
private void openCreatedRecommendationSystem(String recommendationSystemId){
    Intent intent = new Intent(getContext(), RecommendationsActivity.class);
    intent.putExtra( name: "recommendation_system_id", recommendationSystemId);
    startActivity(intent);
}
```

Slika 4.33. *Metoda openCreateRecommendationSystem*

4.6.4. Prikaz postupka odjave korisnika

U zaglavlju glavnog zaslona nalazi se gumb za odjavu korisnika. U metodi *onCreate*() vidljivoj no slici 4.32 na gumb se postavlja *OnClickListener*. Pritiskom na gumb poziva se *signOut()* metoda objekta *FirebaseAuth* koja odjavljuje korisnika. Nakon odjave, korisnik se preusmjerava na zaslon za prijavu.

4.7. Programsko rješenje na strani poslužitelja

4.7.1. Zapisivanje podataka u bazu podataka

Podatci se spremanju u bazu u nekoliko slučajeva, prilikom kreiranja korisnika, prilikom kreiranja sustava preporuka za liječenje te prilikom kreiranja zahtjeva ili preporuka u sustavu preporuka. Na slici 4.28 može se vidjeti kako se spremaju korisnički podatci. Može se vidjeti kako se prvo kreira *DatabaseReference* objekt. Nakon toga se dohvaća instanca *Firebase* baze podataka pozivom *getInstance()* metode i poziva se *getReference()* metoda koja stvara referencu na određenu putanju unutar *Firebase Realtime* baze podataka. U ovom slučaju, putanja se zove *User*. Kada imamo referencu na putanju gdje se nalaze korisnički podatci, pozivamo metodu *child()* kojoj predajemo identifikator korisnika koji se sprema u bazu podataka. Metoda kreira podreferencu unutar glavne reference. Na kraju, poziva se metoda *setValue()* metoda koja postavlja vrijednost na odabrani čvor u bazi podataka. Metodi se predaje objekt klase *UserProfile* koji sadržava korisničke podatke.

Prilikom kreiranja sustava preporuka za liječenje, podatci o sustavu spremaju se u bazu podataka. Prvo se poziva *saveImage()* metoda prikazana na slici 4.34. Metoda služi za spremanje slike u *Firebase Storage*. Može se vidjeti kako se prvo slika pretvara u odgovarajući format. Nakon toga dohvaća se instanca *Firebase Storage-a* i stvara se referenca na putanju koristeći *getInstance()*, *getReference()* i *child()* metode. Slika se sprema u čvor s korisničkim identifikatorom koji se nalazi unutar *images* datoteke. Ako je došlo do nekakvog problema, ispisuje se prikladna poruka. U suprotnom, kada je slika uspješno spremljena poziva se *saveRecommendationSystemInfo()* metoda koja se može vidjeti na slici 4.35. Metoda *saveRecommendationSystemInfo()* sprema sve ostale podatke o sustavu preporuka za liječenje. Može se vidjeti se stvara na referenca na putanju unutar *RecommendationSystems* čvora. Nakon toga pomoću *dbReference.push().getKey()* stvara se čvor s unikatnim identifikatorom za sustav preporuka. Tada se kreiraju početne poruke sustava i kreira se objekt sustava preporuka. Taj objekt se pomoću *setValue()* metode zapisuje u bazu podataka. Dakle, sustav preporuka će biti spremljen na sljedećoj putanji: *RecommentadionSystem* – identifikator korisnika – identifikator sustava preporuka. Ako su podatci uspješno spremljeni, otvara se aktivnost za prikaz sustava

preporuka za liječenje pozivom metode *openCreateRecommendationSystem()* prikazanoj na slici 4.33. U suprotnom, ispisuje se prikladna poruka.

```
private void saveImage(){
    Uri uriImage = getImageUri(image, Bitmap.CompressFormat.JPEG,  quality: 100);
    StorageReference storageReference = FirebaseStorage.getInstance().getReference( location: "images").child(user.getUid());

    if(uriImage != null){
        StorageReference fileReference = storageReference.child(System.currentTimeMillis() + "." + getFileExtension(uriImage));
        fileReference.putFile(uriImage).addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {
            @Override
            public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
                Toast.makeText(getActivity(),  text: "Image upload successful", Toast.LENGTH_SHORT).show();
                String name = etSystemRecommendationName.getText().toString().trim();

                fileReference.getDownloadUrl().addOnSuccessListener(new OnSuccessListener<Uri>() {
                    @Override
                    public void onSuccess(Uri uri) {
                        saveRecommendationSystemInfo(latitude, longitude, name, uri.toString());
                    }
                });

            }
        }).addOnFailureListener(new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                Toast.makeText(getActivity(), e.getMessage(), Toast.LENGTH_SHORT).show();
            }
        });
    }
    else {
        Toast.makeText(getActivity(),  text: "Couldn't get the image.", Toast.LENGTH_SHORT).show();
    }
}
```

Slika 4.34. *Metoda saveImage()*

```
private void saveRecommendationSystemInfo(String latitude, String longitude, String name, String image_reference){
    DatabaseReference dbReference = FirebaseDatabase.getInstance().getReference( path: "RecommendationSystems");

    String diseaseDescription = DiseaseClassificator.getDiseaseClassDescription(classifiedDisease);
    String recommendationSystemId = dbReference.push().getKey();
    List<SimpleMessage> recommendationMessages = new ArrayList<>();
    recommendationMessages.add(new SimpleMessage("This is your \"" + name + "\" plantation."));
    if (classifiedDisease == "healthy"){
        recommendationMessages.add(new SimpleMessage( message: "Your plantation is healthy.",  recommendationMessageType: 1));
    } else {
        recommendationMessages.add(new SimpleMessage("Recognized disease is: " + diseaseDescription));
    }
    RecommendationSystem recommendationSystem = new RecommendationSystem(recommendationSystemId ,latitude, longitude, name, image_reference,
            diseaseDescription, diseaseConfidences, recommendationMessages);

    dbReference.child(user.getUid()).child(recommendationSystemId).setValue(recommendationSystem).addOnCompleteListener(new OnCompleteListener<Void>() {
        @Override
        public void onComplete(@NonNull Task<Void> task) {

            if (task.isSuccessful()){
                Toast.makeText(getActivity(),  text: "Data saved successfully.", Toast.LENGTH_SHORT).show();
                loadingDialog.dismissLoadingDialog();
                openCreatedRecommendationSystem(recommendationSystemId);
            }
            else{
                Toast.makeText(getActivity(),  text: "Couldn't save data, please try later.", Toast.LENGTH_SHORT).show();
            }
        }
    });
}
```

Slika 4.35. *Metoda saveRecommendationSystemInfo()*

Posljednji slučaj spremanja podataka u bazu podataka je prilikom kreiranja zahtjeva ili preporuka u sustavu preporuka za liječenje. Tada se poziva *addRecommendationMessage()*

metoda prikazana na slici 4.36. Kao što se može vidjeti sa slike, poruka se dodaje na kraj liste poruka za taj sustav preporuka liječenja. Ovisno o uspjehu operacije, ispisuje se prikladna *Toast* poruka na zaslon.

```
public void addRecommendationMessage(SimpleMessage message){

    int key = recommendationSystem.getRecommendationMessages().size();

    //reference is already set to the Recommendation system id in the OnCreate method
    databaseReference.child("recommendationMessages").child(String.valueOf(key)).setValue(message).addOnCompleteListener(new OnCompleteListener<Void>() {
        @Override
        public void onComplete(@NonNull Task<Void> task) {

            if (task.isSuccessful()){
                Toast.makeText( context: RecommendationsActivity.this,  text: "Data saved successfully.", Toast.LENGTH_SHORT).show();
            }
            else{
                Toast.makeText( context: RecommendationsActivity.this,  text: "Couldn't save data, please try later.", Toast.LENGTH_SHORT).show();
            }
        }
    });
}
```

Slika 4.36. *Metoda addRecommendationMessage()*

4.7.2. Dohvaćanje podataka iz baze podataka

Dohvaćanje podataka o korisniku izvršava se u *onCreate()* metoda za glavni zaslon prikazanoj na slici 4.32. Može se vidjeti kako se prvo kreira referenca na putanju gdje se nalaze podatci korisnika. Nakon toga, dodaje se *ValueEventListener* koji će osluškivati promjene podataka na toj putanji u *Firebase* bazi podataka. Ako dođe do promjene poziva se *onDataChange()* metoda. Unutar metode dobivaju se podatci o korisničkom profilu iz *DataSnapshot* objekta i pretvaraju se u objekt tipa *UserProfila*. Ako se dogodi greška prilikom čitanja podataka iz *Firebase* baze podataka, pozvat će se *onCancelled()* metoda gdje će se prikazati kratka *Toast* poruka koja će obavijestiti korisnika o grešci.

Podatci se vrlo slično dohvaćaju i za sustav preporuka za liječenje. Kreira se referenca na čvor s identifikatorom sustava preporuka za liječenje i postavlja se *ValueEventListener*. Kada dođe do bilo kakve promijene u podatcima na unutar tog čvora, poziva se *onDataChange()* metoda. Unutar metode dobivaju se podatci o sustavu preporuka za liječenje i pretvaraju se u objekt tipa *RecommentadionSystem*. Kada su podatci uspješno dohvaćeni, ažurira se korisničko sučelje. Dio programskog koda za dohvaćanje podatka o sustavu preporuka za liječenje i ažuriranje korisničkog sučelja može se vidjeti na slici 4.37.

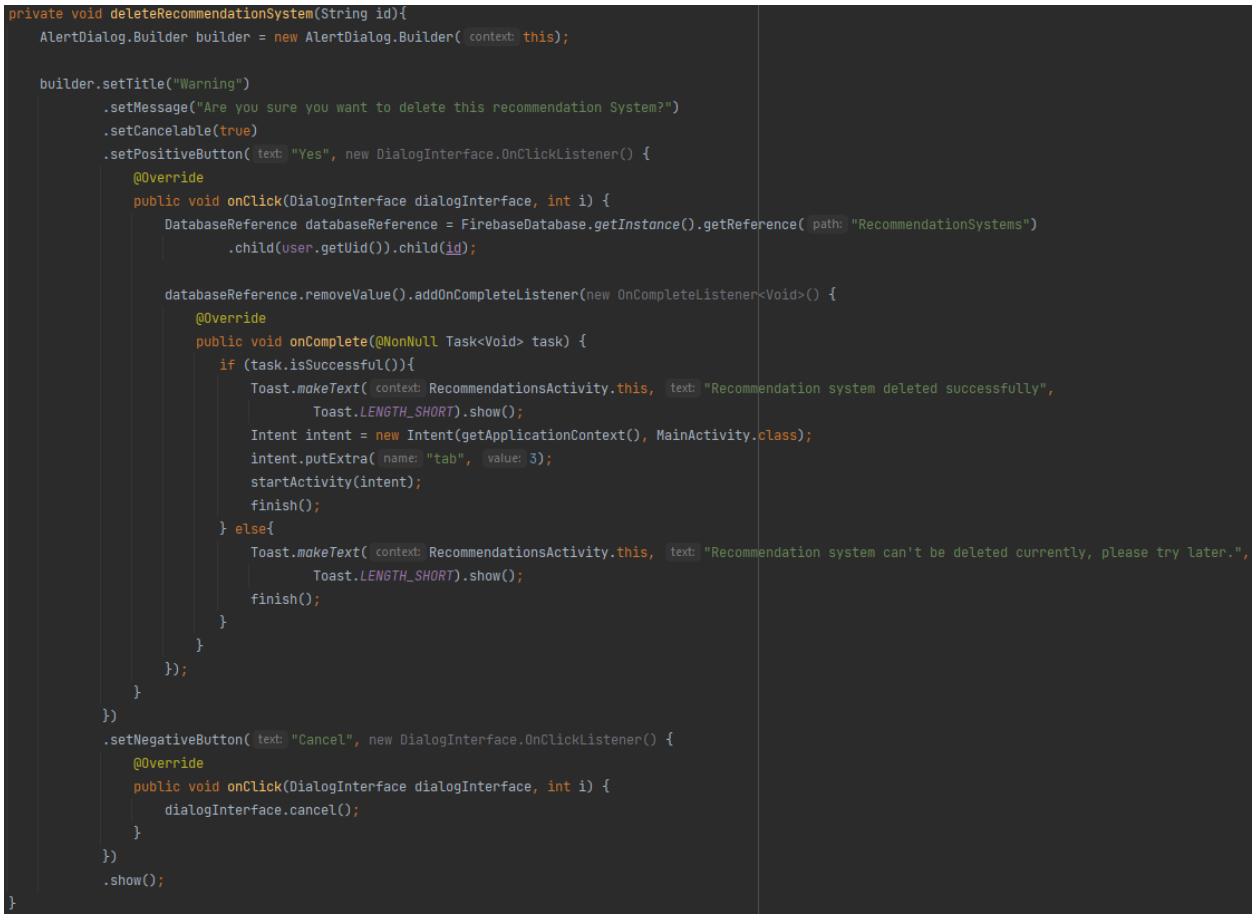
```
databaseReference.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        RecommendationSystem recommendationSystem = snapshot.getValue(RecommendationSystem.class);
        if (recommendationSystem == null){
            return;
        }
        setRecommendationSystem(recommendationSystem);
        updateRecommendationMessages(recommendationSystem);

        tvName.setText(recommendationSystem.getName());
        tvLatLong.setText(recommendationSystem.getLatitude() + " / " + recommendationSystem.getLongitude());
        tvDisease.setText(recommendationSystem.getClassifiedDisease());

        Picasso.with(getApplicationContext()) Picasso
                .load(recommendationSystem.getStartingImageReference()) RequestCreator
                .fit()
                .centerCrop()
                .into(ivStartingImage);
```

Slika 4.37. *Programski kod za dohvaćanje podataka o sustavu preporuka za liječenje i ažuriranje korisničkog sučelja*

Na isti način dohvaćaju se podatci i za sustave preporuka iz *RecommendationSystemsFragment-a*. Jedina je razlika što se tamo dohvaćaju svi sustavi preporuka za liječenje tog korisnika. Tada se iterira kroz svu djecu *DataSnapshot* objekta i ti podatci se pretvaraju u objekt tipa *RecommentadionSystem*. Na kraju se svi sustavi preporuka za liječenje prikazuju pomoću *RecyclerView-a.*

4.7.3. Brisanje podataka iz baze podataka

U zaglavlju zaslona za prikaz sustava preporuka za liječenje nalazi se gumb za brisanje tog sustava. Pritiskom na gumb poziva se metoda *deleteRecommendationSystem()* koja se može vidjeti na slici 4.38. Metoda prima identifikator sustava preporuka koji će se obrisati. Kao što se može vidjeti sa slike, kreira se te prikazu dijalog koji traži potvrdu da se želi obrisati sustav preporuka za liječenje. U slučaju potvrde kreira se referenca na čvor s identifikator to sustava preporuka i poziva se metoda *removeValue()*. Na operaciju se postavlja *OnCompleteListener* koji osluškuje operacije. Kada je operacija gotova poziva se metoda *onComplete()*. Ako je operacija uspješna, ispisuje se prikladna *Toast* poruka i korisnika se preusmjerava na glavni zaslon s prikazanim *RecommendationSystemsFragment-om.* U suprotnom, ispisuje se *Toast* poruka koja će obavijestiti korisnika kako operacija nije uspjela.



```
private void deleteRecommendationSystem(String id){
    AlertDialog.Builder builder = new AlertDialog.Builder( context: this);

    builder.setTitle("Warning")
            .setMessage("Are you sure you want to delete this recommendation System?")
            .setCancelable(true)
            .setPositiveButton( text: "Yes", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialogInterface, int i) {
                    DatabaseReference databaseReference = FirebaseDatabase.getInstance().getReference( path: "RecommendationSystems")
                            .child(user.getUid()).child(id);

                    databaseReference.removeValue().addOnCompleteListener(new OnCompleteListener<Void>() {
                        @Override
                        public void onComplete(@NonNull Task<Void> task) {
                            if (task.isSuccessful()){
                                Toast.makeText( context: RecommendationsActivity.this, text: "Recommendation system deleted successfully",
                                        Toast.LENGTH_SHORT).show();
                                Intent intent = new Intent(getApplicationContext(), MainActivity.class);
                                intent.putExtra( name: "tab", value: 3);
                                startActivity(intent);
                                finish();
                            } else{
                                Toast.makeText( context: RecommendationsActivity.this, text: "Recommendation system can't be deleted currently, please try later.",
                                        Toast.LENGTH_SHORT).show();
                                finish();
                            }
                        }
                    });
                }
            })
            .setNegativeButton( text: "Cancel", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialogInterface, int i) {
                    dialogInterface.cancel();
                }
            })
            .show();
}
```

Slika 4.38. *Metoda deleteRecommendationSystem()*

4.7.4. Analiza podataka

Učitavanjem slike s kartice *Classify* ili iz zaslona sa sustavom preporuka prilikom provjere učinkovitosti liječenja, poziva se metoda *classifyDisease()* iz klase *DiseaseClassificator*. Metoda *classifyDisease()* koristi se za klasifikaciju, odnosno prepoznavanje bolesti i može se vidjeti na slici 4.39.

```java
public void classifyDisease(Bitmap image){
    try {
        Model model = Model.newInstance(context);

        // Creates inputs for reference.
        TensorBuffer inputFeature0 = TensorBuffer.createFixedSize(new int[]{1, imageSize, imageSize, 3}, DataType.FLOAT32);
        ByteBuffer byteBuffer = ByteBuffer.allocateDirect(4 * imageSize * imageSize * 3);
        byteBuffer.order(ByteOrder.nativeOrder());

        int[] intValues = new int[imageSize * imageSize];
        int pixel = 0;
        //iterate over each pixel and extract R, G and B values. Add those values individually to the byte buffer.
        image.getPixels(intValues,  offset: 0, image.getWidth(),  x: 0,  y: 0, image.getWidth(), image.getHeight());
        for(int i = 0; i < imageSize; i++){
            for(int j = 0; j < imageSize; j++){
                int val = intValues[pixel++];
                byteBuffer.putFloat(((val >> 16) & 0xFF) * (1.f / 1));
                byteBuffer.putFloat(((val >> 8) & 0xFF) * (1.f / 1));
                byteBuffer.putFloat((val & 0xFF) * (1.f / 1));
            }
        }

        inputFeature0.loadBuffer(byteBuffer);

        // Runs model inference and gets result.
        Model.Outputs outputs = model.process(inputFeature0);
        TensorBuffer outputFeature0 = outputs.getOutputFeature0AsTensorBuffer();

        float[] confidence = outputFeature0.getFloatArray();
        float maxConfidence = 0;
        int maxPos = 0;
        for(int i = 0; i < confidence.length; i++){
            if(confidence[i] > maxConfidence){
                maxConfidence = confidence[i];
                maxPos = i;
            }
        }

        this.maxConfidencePosition = maxPos;
        this.classifiedClass = classes[maxPos];
        this.diseaseConfidences = rescaleConfidences(confidence);

        // Releases model resources if no longer used.
        model.close();


    } catch (IOException e) {
        // TODO Handle the exception
    }
}
```

Slika 4.39. *Metoda classifyDisease()*

Na početku se stvara novi objekt modela koji se koristi za klasifikaciju bolesti. Koristi se unaprijed istrenirana konvolucijska neuronska mreža uvezena u mobilnu aplikaciju. Nakon toga, stvara se *TensorBuffer* za ulazne karakteristike modela. Ovdje se očekuje da će ulaz biti slika u boji dimenzija *imageSize* x *imageSize* x 3 u formatu 32-bitnih brojeva s pomičnim zarezom. Varijabla *imageSize* unaprijed je definirana i predstavlja veličinu slike koju model očekuje na svom ulazu. Isto tako, stvara se *ByteBuffer* za pohranu piksela slike koji će sadržavati vrijednosti piksela slike kao niz 32-bitnih brojeva s pomičnim zarezom. Dodatno, stvara se niz *intValues* za

pohranu vrijednosti piksela slike kao 32-bitnih cijelih brojeva. Nakon toga, pikseli slike kopiraju se u niz *intValues* omogućava pristup RGB vrijednostima svakog piksela slike. Tada se iterira kroz svaki piksel slike te se RGB vrijednosti pretvaraju u 32-bitne brojeve s pomičnim zarezom te ih se pohranjuje u *ByteBuffer*. Nakon toga se sadržaj *ByteBuffer-a* učitava u *TensorBuffer* za ulazne karakteristike modela što priprema sliku za ulaz u model za klasifikaciju. Slika, odnosno ulazne karakteristike se provlače kroz konvolucijsku neuronsku mrežu i rezultat te obrade se pohranjuje u *outputs*. Tada se izlazna karakteristika modela dohvaća kao *TensorBuffer*. Ova karakteristika sadrži vjerojatnosti pripadnosti različitim klasama bolesti. Sljedeći dio koda prolazi kroz izlaznu karakteristiku i traži klasu s najvećom vjerojatnošću. Nakon što je pronađena, informacije o toj klasi se pohranjuju kako bi se kasnije koristile u aplikaciji. Dodatno, prije spremanja, vjerojatnosti pripadnosti različitim klasama se skaliraju na način da klasa s najvećom vrijednosti poprima vrijednost jedan, a klasa s najmanjom vjerojatnosti poprima vrijednost nula. Skaliranje se izvršava koristeći *rescaleConfidences()* metodu koja se može vidjeti na slici 4.40. Na kraju, metoda zatvara resurse modela pozivom *model.close()*. Ovo je važno kako bi se oslobodili resursi nakon što je klasifikacija završena.

```
public float[] rescaleConfidences(float[] inputArray) {
    float minValue = Float.MAX_VALUE;
    float maxValue = Float.MIN_VALUE;

    // Find the minimum and maximum values in the input array
    for (float value : inputArray) {
        if (value < minValue) {
            minValue = value;
        }
        if (value > maxValue) {
            maxValue = value;
        }
    }

    // Scale the values to the [0, 1] interval
    float scaleFactor = maxValue - minValue;
    float[] scaledArray = new float[inputArray.length];
    for (int i = 0; i < inputArray.length; i++) {
        scaledArray[i] = (inputArray[i] - minValue) / scaleFactor;
    }

    return scaledArray;
}
```

Slika 4.40. *Metoda rescaleConfidences()*

Dodatno, prilikom provjere učinkovitosti liječenja poziva se *getTreatmentEffectiveness()* metoda koja se može vidjeti na slici 4.41. Uspoređuje se vjerojatnost da je nasad zdrav iz posljednje dvije provjere. Metoda se koristi za procjenu i izračunavanje efikasnosti liječenja na temelju

promjene u vjerojatnostima da je nasad zdrav prije i nakon liječenja. Ovisno o tome je li se vjerojatnost povećala ili smanjila, metoda će vratiti odgovarajuću vrijednost postotka promjene.

```
private float getTreatmentEffectiveness(float confidenceBefore, float confidenceNow){
    if (confidenceBefore < confidenceNow){
        return ((1 - confidenceNow) / (1 - confidenceBefore)) *  100;
    } else if(confidenceBefore > confidenceNow) {
        return (1 - (confidenceNow / confidenceBefore)) * (-100);
    }

    return 0f;
}
```

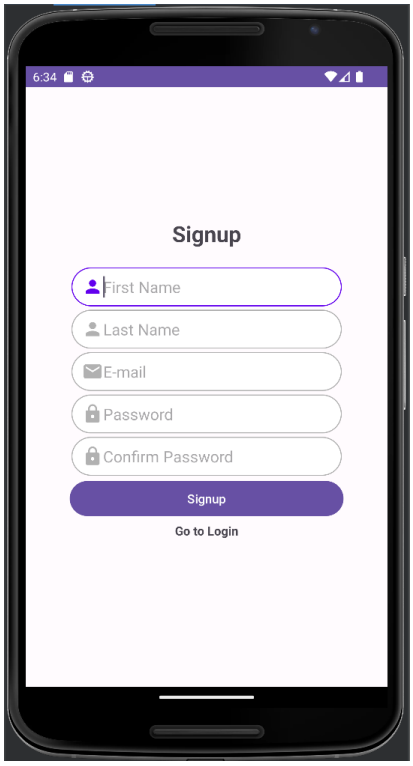
Slika 4.41. *Metoda getTreatmentEffectiveness()*

5. PRIKAZ KORIŠTENJA I ISPITIVANJE OSTVARENOG RJEŠENJA

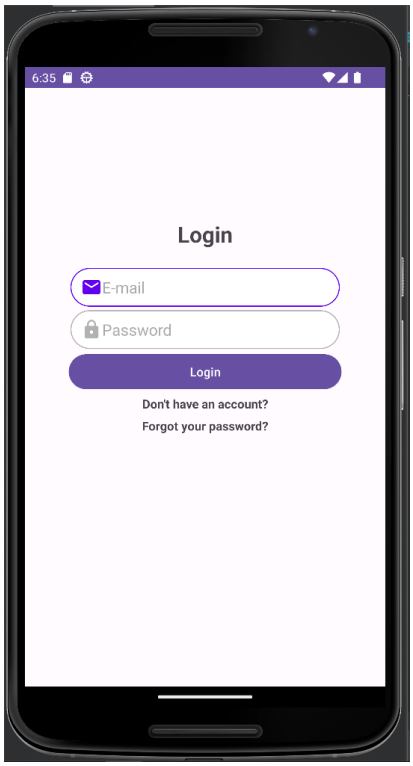
U ovom poglavlju prikazati će se mobilna aplikacija. Isto tako, rad mobilne aplikacije i postupka klasifikacije ispitat će se u nekoliko korisničkih slučajeva.

5.1. Prikaz korištenja mobilne aplikacije

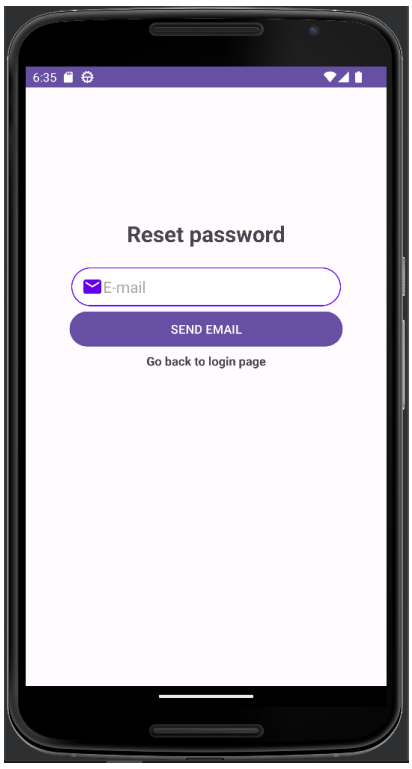
Kao što je već spomenuto, prilikom prvog pokretanja mobilne aplikacije od korisnika se traži registracija. Ukoliko korisnik već ima korisnički račun potrebna je samo prijava. Kada se korisnik registrira, dolazi mu e-mail za verifikaciju korisničkog računa. Korisnik aktivira korisnički račun i može početi koristiti aplikaciju. Na slikama 5.1 i 5.2 može se vidjeti zaslon za registraciju, odnosno zaslon za prijavu. Dodatno, korisnik ima mogućnost promijene lozinke ukoliko je zaboravio isti ili jednostavno želi promijeniti lozinku. Kada pošalje zahtjev za promjenu lozinke, dolazi mu e-mail koji mu omogućuje promjenu. Zaslon za promjenu lozinke prikazan je na slici 5.3.



Slika 5.1. *Prikaz zaslona za registraciju*



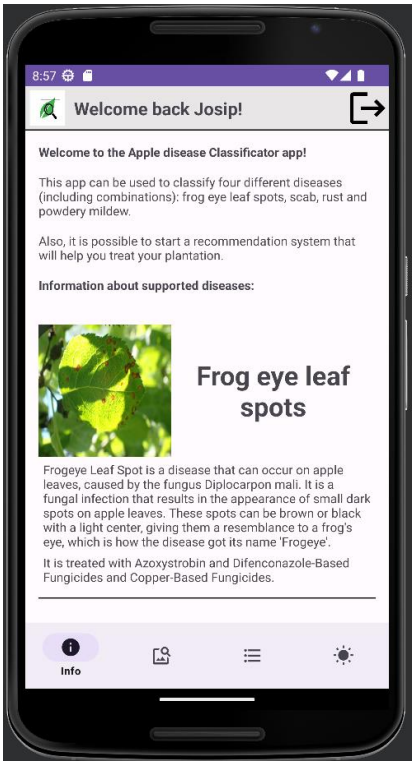
Slika 5.2. *Prikaz zaslona za prijavu*



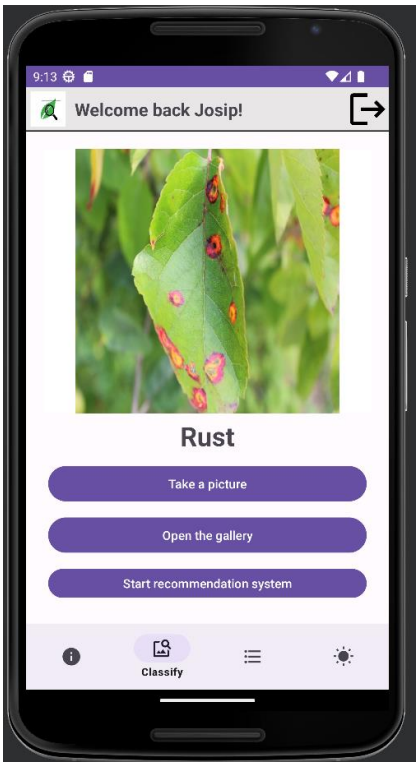
Slika 5.3. *Prikaz zaslona za promjenu lozinke*

Nakon uspješnog aktiviranja korisničkog računa i prijave, prikazuje se glavni zaslon aplikacije s *InfoFragment-om*. Glavni zaslon *InfoFragment-om* prikazan je na slici 5.4. Kao što je već opisano, *InfoFragment*, odnosno Info kartica, služi kako bi korisnika upoznala sa podržanim

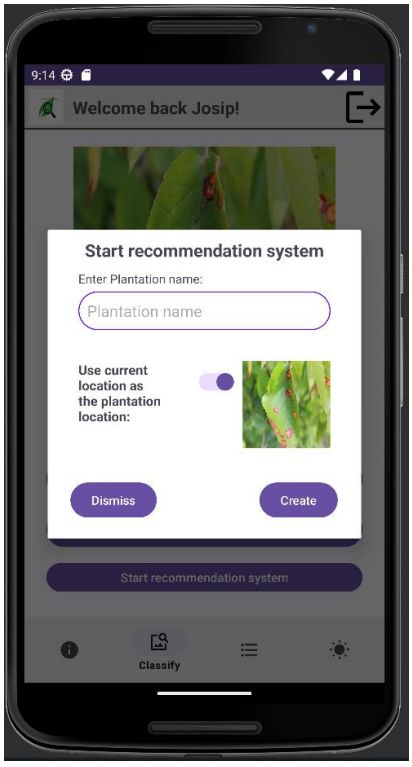
bolestima i mobilnom aplikacijom. Odabirom kartice *Classify* prikazuje se *ClassifyDiseaseFragment*. S ove kartice korisnik može učitati sliku i provjeriti ako je njegov nasad zaražen nekom od bolešću. *ClassifyDiseaseFragment* može se vidjeti na slici 5.5. Dodatno, iz ove kartice može se kreirati sustav preporuka za liječenje. Pritiskom na gumb *Start recommendation system* otvara se dijalog za kreiranje novog sustava preporuka. Dijalog se može vidjeti na slici 5.6.



Slika 5.4. *Prikaz InfoFragment-a na glavnom zaslonu*

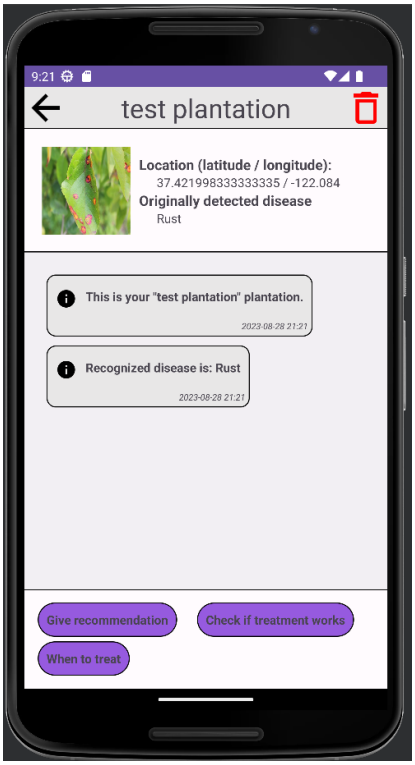


Slika 5.5. *Prikaz ClassifyDiseaseFragment-a na glavnom zaslonu*



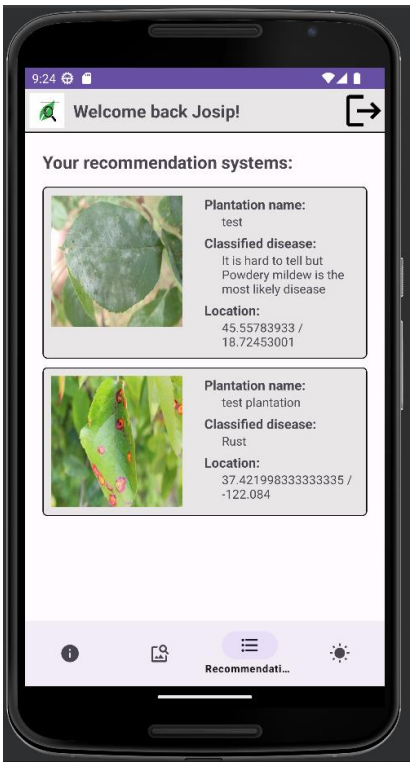
Slika 5.6. *Prikaz dijaloga za kreiranje sustava preporuka za liječenje*

Nakon uspješnog kreiranja sustava preporuka za liječenja, korisnik je preusmjeren na zaslon za sustav preporuka koje se može vidjeti na slici 5.7. Na ovom zaslonu vide se informacije o nasadu i sustavu preporuka, kao i povijest svih preporuka. Dodatno, iz ovog zaslona korisnik može zatražiti bilo koju od preporuka i obrisati otvoreni sustav preporuka. Pritiskom na strelicu nazad, korisnik se vraća na glavni zaslon s otvorenom *Recommendations* karticom, tj. prikazanim *RecommendationSystemsFragment-om*.

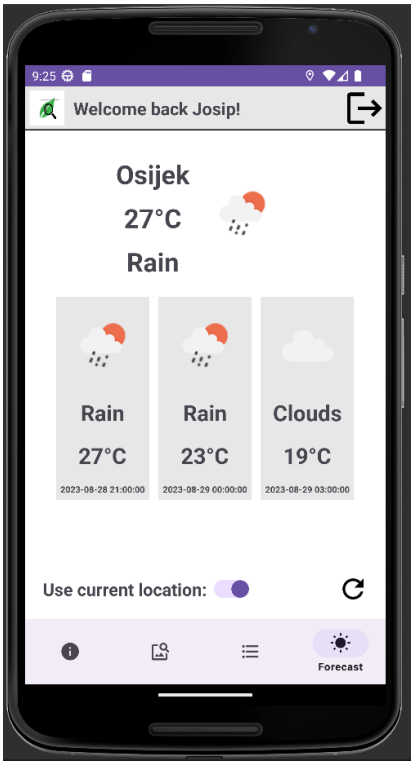


Slika 5.7. *Prikaz zaslona s sustavom preporuka za liječenje*

Na *Recommendations* kartici prikazani su svi korisnikovi sustavi preporuka za liječenje. Pritiskom na bilo koji od njih, prikazuje se zaslon s tim sustavom preporuka. *Recommendations* kartica i *RecommendationSystemsFragment* prikazani su na slici 5.8. Dodatno, korisnik može vidjeti vremensku prognozu za trenutnu ili bilo koju lokaciju. Otvaranje *Forecast* kartice prikazuje se *WeatherForecastFragment* i dohvaća se odgovarajuća vremenska prognoza. Na slici 5.9 može se vidjeti glavni zaslon s prikazanim *WeatherForecastFragment-om*.



Slika 5.8. *Prikaz RecommendationSystemsFragment-a na glavnom zaslonu*



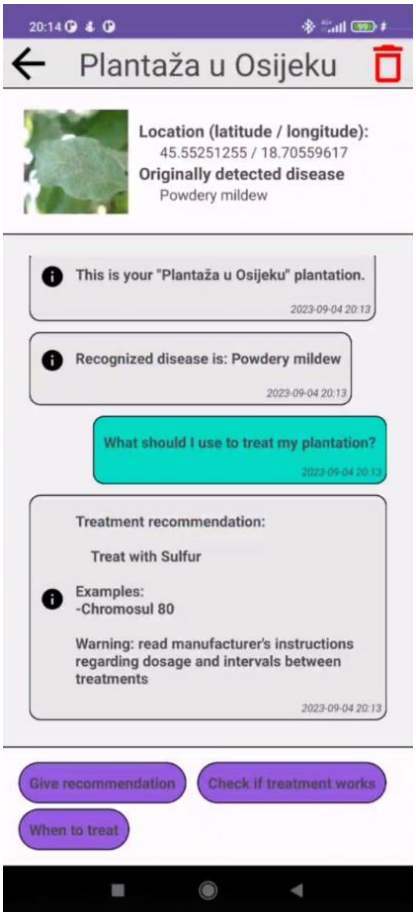
Slika 5.9. *Prikaz WeatherForecastFragment a na glavnom zaslonu*

5.2. Ispitivanje rada mobilne aplikacije i postupaka klasifikacije

Vrlo je važno ispitati rad mobilne aplikacije kako kod korištenja ne bi došlo do neočekivanih rezultata. U sljedećim poglavljima prikazani su rezultati različitih ispitnih slučajeva. Svaki od prikazanih ispitnih slučajeva opisuje različite slučajeve s različitim parametrima.

5.2.1. Korisnički slučaj 1: Preporuka za liječenje

U ovom ispitnom slučaju kreiran je sustav za preporuku liječenja gdje je prilikom kreiranja detektirano da nasad boluje od pepelnice. Nakon toga korisnik traži preporuku za liječenje. Na slici 5.10 prikazan je rezultat nakon kreiranja preporuke za liječenje.



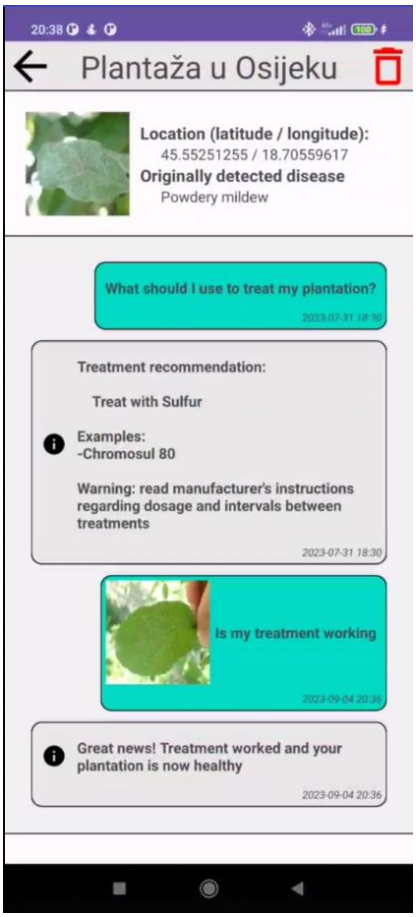
Slika 5.10. *Rezultat nakon kreiranja preporuke za liječenje*

Kao što se može vidjeti sa slike 5.10, sustav je stvorio preporuku gdje se preporuča liječenje sa sumporom. Isto tako predložen je primjer sredstva koji se koriste za liječenje pepelnice, odnosno Chromosul 80. Ovo je očekivani rezultat.

5.2.2. Korisnički slučaj 2: Nasad je izliječen nakon prethodno prepoznate bolesti

Nadovezujući se na prethodni primjer, korisnik je slijedio preporuku i tretirao nasad. Nakon toga odlučio je provjeriti ako je liječenje uspjelo i učitao je sliku lista koja se može vidjeti u korisničkom zahtjevu za provjeru učinkovitosti liječenja na slici 5.11. Nakon učitavanja slike i

provjere ako je liječenje uspjelo, sustav preporuka je kreirao poruku koja se također može vidjeti na slici 5.11.

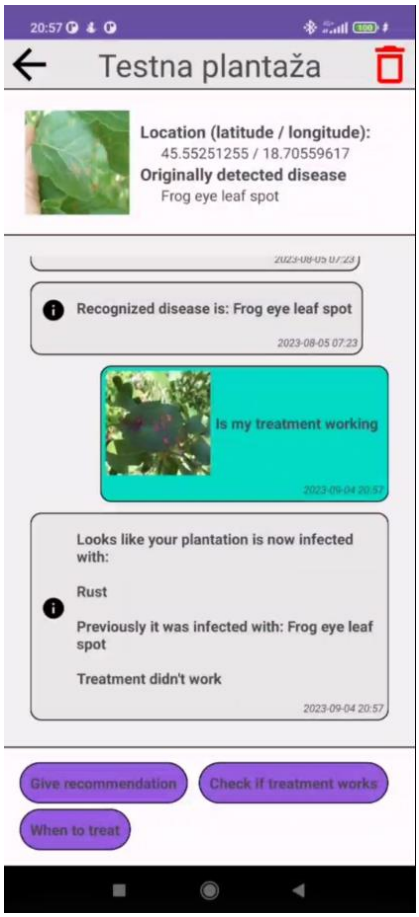


Slika 5.11. *Sustav preporuka za liječenje nakon provjere učinkovitosti liječenja - slučaj kada je nasad izliječen*

Kao što je i očekivano, sustav preporuka za liječenje prepoznao je da je nasad izliječen te je završio sustav preporuka.

5.2.3. Korisnički slučaj 3: Nasad je zaražen drugom bolešću

U ovom ispitnom slučaju, kreiran je sustav preporuka za liječenje gdje je izvorno detektirana mrljavost jabuke. Nakon toga, provjerava se učinkovitost liječenja. Učitava se slika lista jabuke i sa slike je moguće prepoznati kako je nasad sada zarađen hrđom jabuke. Učitana slika prilikom zahtjeva za provjeru učinkovitosti liječenja kao te poruka koju je vratio sustav preporuka mogu se vidjeti na slici 5.12.

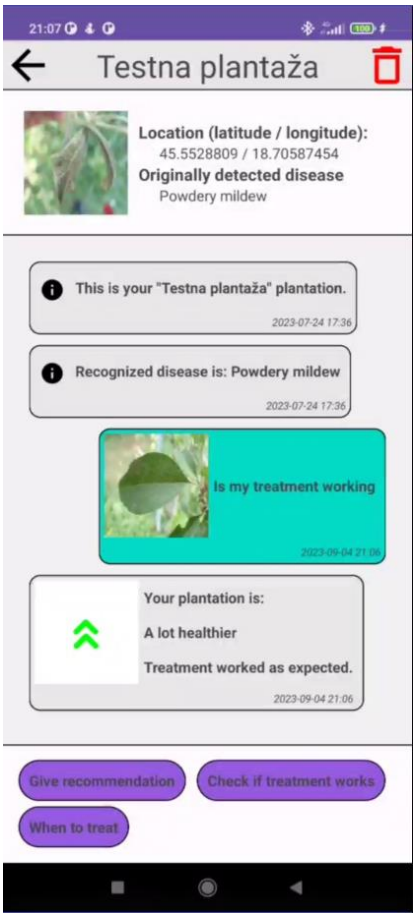


Slika 5.12. *Sustav preporuka za liječenje nakon provjere učinkovitosti liječenja - slučaj kada je nasad zaražen različitom bolesti od prethodno klasificirane*

Kako je i očekivano, sustav preporuka za liječenje je prepoznao da je nasad sada zaražen hrđom jabuke te je obavijestio korisnika prikladnom porukom.

5.2.4. Korisnički slučaj 4: Učinkovitost liječenja je pozitivna

U ovom ispitnom slučaju, kreiran je sustav preporuka za liječenje gdje je izvorno detektirana pepelnica. Nakon nekog vremena provjerava se stanje nasada. Učitava se nova slika lista jabuke i kreira se zahtjev za provjeru učinkovitosti liječenja. S učitane slike može se vidjeti da je nasad još uvijek zaražen pepelnicom, ali da je situacija puno bolja. Izvorna slika, učitana slika za provjeru učinkovitosti liječenja te poruka od sustava preporuka mogu se vidjeti na slici 5.13.

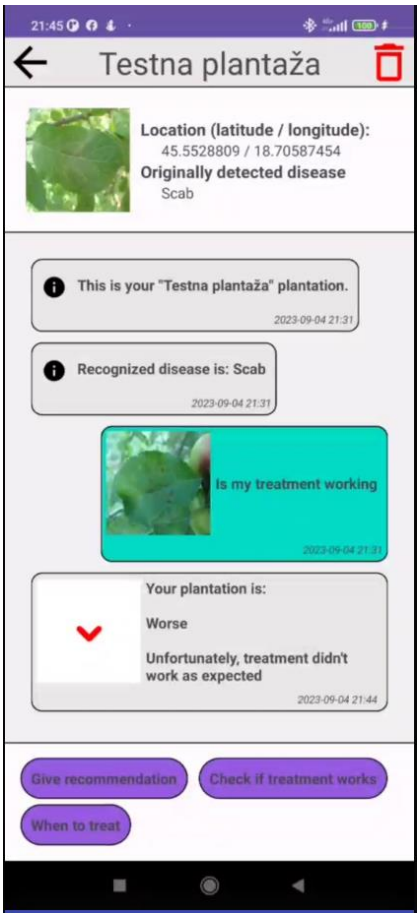


Slika 5.13. *Sustav preporuka za liječenje nakon provjere učinkovitosti liječenja - slučaj kada je učinkovitost liječenja pozitivna*

Kao što je bilo i za očekivati, sustav preporuka za liječenje je prepoznao kako da se na obje slike radi o pepelnici. Isto tako, nakon računanja učinkovitosti liječenja, sustav je prepoznao kako je situacija puno bolja te je ispisao prikladnu poruku.

5.2.5. Korisnički slučaj 5: Učinkovitost liječenja je negativna

Slično kao u prethodnom ispitnom slučaju, kreiran je sustav preporuka za liječenje gdje je izvorno detektirana krastavost jabuke. Nakon nekog vremena provjerava se stanje nasada. Učitava se nova slika lista jabuke i kreira se zahtjev za provjeru učinkovitosti liječenja. Različito nego u prošlom ispitnom slučaju, s učitane slike može se vidjeti da je nasad još uvijek zaražen krastavošću jabuke, ali je sada stanje nasada nešto gore. Izvorna slika, učitana slika za provjeru učinkovitosti liječenja te poruka od sustava preporuka mogu se vidjeti na slici 5.14.

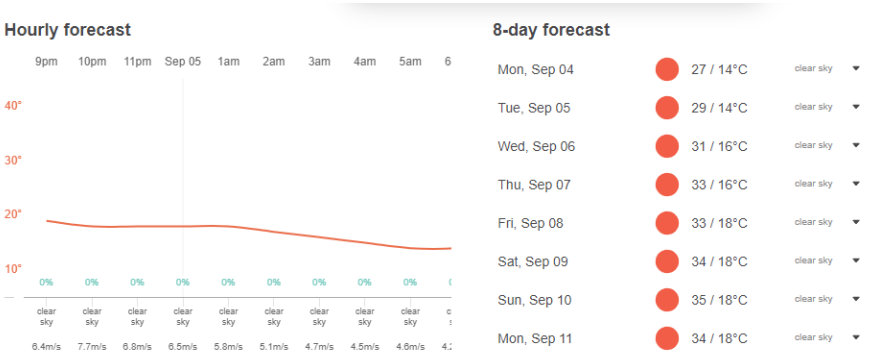


Slika 5.14. *Sustav preporuka za liječenje nakon provjere učinkovitosti liječenja - slučaj kada je učinkovitost liječenja negativna*

Sa slike 5.14 može se vidjeti kako je sustav preporuka za liječenje prepoznao da se na obje slike radi o krastavosti jabuke. Dodatno, nakon računanja učinkovitosti liječenja, sustav je prepoznao kako je situacija malo lošija nego kod prošle provjere. Isto tako, prikazana je primjerena poruka.

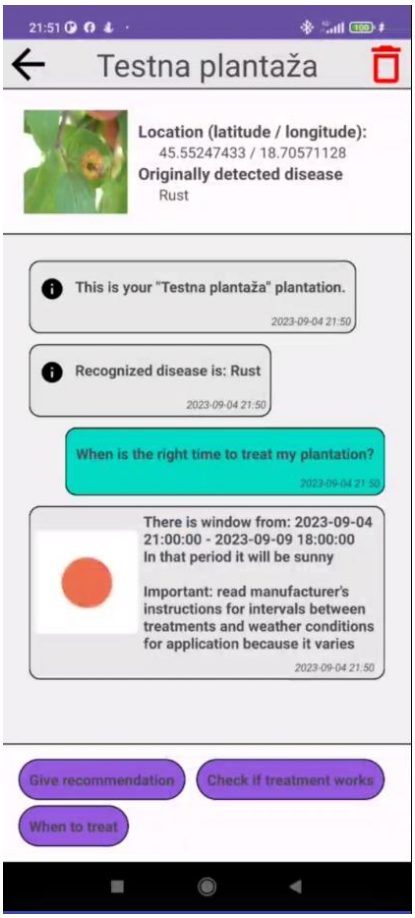
5.2.6. Korisnički slučaj 6: Pronalazak vremenskog prozora za liječenje nasada

U ovom ispitnom slučaju, kreiran je sustav preporuka za liječenje nakon čega je zatražena preporuka za vrijeme kada tretirati nasad. Nasad je prilikom kreiranja sustava preporuka za liječenje smješten u Osijek. Prognoza za Osijek prilikom ispitivanja aplikacije može se vidjeti 5.15. Vremenska prognoza prikazana na slici je uzeta s *OpenWeather* usluge dostupne na [29] s koje se dohvaća prognoza pri korištenju mobilne aplikacije. Promatrajući sliku može se vidjeti kako se ne očekuje kiša niti bilo kakvo drugo nepovoljno vrijeme u sljedećih osam dana.



Slika 5.15. *Vremenska prognoza za sljedećih osam dana u trenutku ispitivanja rada mobilne aplikacije*

Preporuka sustava preporuka za liječenje može se vidjeti na slici 5.16.



Slika 5.16. *Sustav preporuka za liječenje nakon traženja prozora za liječenje*

Kao što je očekivano, sustav je prepoznao da postoji prozor u trajanju od minimalno dva dana i ispisao u kojem razdoblju nastupa. Isto tako, može se primijetiti kako prozor traje samo pet dana što je također očekivano. Iako je na slici 5.15 prikazana vremenska prognoza za osam dana, mobilna aplikacija koristi detaljniju prognozu od pet dana u koracima od tri sata.

5.3. Analiza rezultata ispitivanja

Promatranjem ispitnih slučajeva može se vidjeti kako se uzimaju posebne situacije do kojih može doći prilikom korištenja sustava preporuka za liječenje. Naravno, sve preporuke koje sustav za liječenje daje ovise o točnosti prepoznavanja bolesti sa slike. U tablici 5.1 mogu se vidjeti dobivene točnosti s različitim iteracijama modela za klasifikaciju bolesti lista jabuke opisanima u potpoglavlju 4.3.

Tablica 5.1. *Dobivene točnosti kod različitih iteracija modela za klasifikaciju bolesti lista jabuke*

Prva iteracija – jednostavan model	**Druga iteracija – složeniji model**	**Treća iteracija – model s *ResNet101* kao ekstraktorom značajki**
69.4%	79.8%	85.2%

Iz tablice se može vidjeti kako su najbolji rezultati dobivene u trećoj iteraciji gdje se *ResNet101* koristi kao ekstraktor značajki. Model vrlo dobro klasificira u slučajevima kada je list zaražen s jednom bolešću. S druge strane, model nešto lošije klasificira kada je list zaražen sa više bolesti. Razlog tome je ponajviše količina primjera u skupu podataka za takve slučajeve. Dodatno, promatranjem korisničkih slučajeva, može se uočiti kako aplikacija nema neočekivanih izlaza za različite zahtjeve korisnika. Sustav za svaki zahtjev u obzir uzima ostale ulazne podatke te prikazuje odgovarajuću preporuka.

6. ZAKLJUČAK

Poznato je da bolesti u voćarstvu predstavljaju ozbiljan problem, a posljedice uključuju smanjenje prinosa, smanjenje kvalitete plodova te dugoročno oštećenje samih stabala jabuke. U ovom radu osmišljena je, modelirana i ostvarena Android aplikacija za potporu zaštiti nasada jabuka klasifikacijskim postupcima strojnog učenja koja korisnicima omogućuje prepoznavanje bolesti lista jabuke te podršku pri liječenju bolesti. Korisnik unosi slike pomoću kojih se procjenjuje o kojoj se bolesti rade te se daju preporuke za liječenje. Mobilna aplikacija izrađena je u razvojnoj okolini Android Studio pri čemu je korišten programski jezik Java i označni jezik XML. Dodatno, rad mobilne aplikacije ispitan je u nekoliko korisničkih slučajeva, pri čemu rezultati prikazuju da aplikacija nema neočekivanih izlaza za različite ulazne parametre. Klasifikacijski model strojnog učenja izrađen je koristeći Python te Tensorflow i Keras.

Može se vidjeti da mobilna aplikacija, odnosno klasifikacijski model strojnog učenja razmjerno dobro procjenjuje o kojoj se bolesti lista jabuke radi. Postignuta je točnost od 85.2% pri prepoznavanju bolesti lista jabuke. Isto tako, može se vidjeti da sustav preporuka za liječenje nema neočekivanih izlaza. Važno je napomenuti da se korištenjem razvijenog rješenja vjerojatno ne mogu dobiti rezultati koji bi zamijenili mišljenje stručnjaka. Uz to, model za klasifikaciju i mobilna aplikacija imaju mnogo prostora za poboljšanja, kao što je povećavanje točnosti pri prepoznavanju bolesti lista jabuke, davanje detaljnijih preporuka za liječenje ovisno o sorti jabuka i optimiranje rada mobilne aplikacije.

LITERATURA

[1] H. Dun-chun et al., "Triple Bottom-Line Consideration of Sustainable Plant Disease Management: From Economic, Sociological and Ecological Perspectives", Journal of Integrative Agriculture 2021, Vol. 20, Issue 10, pp. 2581–2591.

[2] H. Dun-chun, Z. Jia-sui, XIE Lian-hui, "Problems, Challenges and Future of Plant Disease Management: From an Ecological Point of View", Journal of Integrative Agriculture, 2016, Vol. 15, Issue 4, pp. 705–715.

[3] Mrljavost lista i krastavost ploda jabuke, Ministarstvo poljoprivrede RH [online], dostupno na: https://www.savjetodavna.hr/product/mrljavost-lista-i-krastavost-ploda-jabuke/ [26.06.2023.]

[4] Chromos Agro, Krastavost jabuke, fuzikladij (Venturia inaequalis), dostupno na: https://www.chromos-agro.hr/krastavost-jabuke-fuzikladij/ [26. 06. 2023.]

[5] M. Rocafort et al., „CRISPR-Cas9 Gene Editing And Rapid Detection of Gene-Edited Mutants Using High-Resolution Melting in the Apple Scab Fungus, Venturia Inaequalis“, Fungal Biology, January 2022, Vol. 126, Issue 1, pp. 35-46.

[6] S. Afzal et al., Grasp of Wheat Leaf Rust Through Plant Leaves Extract and Bioagent as an Eco-Friendly Measure, Journal of King Saud University - Science, February 2022, Vol. 35, Issue 2.

[7] Syngenta, Pepelnica jabuke, 2020., dostupno na: https://www.syngenta.hr/news/jabuka/pepelnica-jabuke-podosphaera-leucotricha [26. 06. 2023.]

[8] Agroklub, Suzbijanje pepelnice u nasadima jabuka, 2017., dostupno na: https://www.agroklub.com/vocarstvo/suzbijanje-pepelnice-u-nasadima-jabuke/33488 [26.06.2023.]

[9] Chromos Agro, Pepelnica jabuke, dostupno na: https://www.chromos-agro.hr/pepelnica-jabuke/ [26.06.2023.]

[10] C. Sarkar et al., Leaf Disease Detection Using Machine Learning and Deep Learning: Review and Challenges, Applied Soft Computing, 2023, Vol. 145, Volume 145, September 2023, 110534.

[11] B.V. Nikith, N.K.S. Keerthan, M.S. Praneeth, Dr. T Amrita, Leaf Disease Detection and Classification, Procedia Computer Science, 2023, Vol. 218, pp. 291-300.

[12] Bin Liu, Huakun Ren, Jiaxin Li, Nannan Duan, Aihong Yuan, and Haixi Zhang. 2023. RE-RCNN: A Novel Representation-Enhanced RCNN Model for Early Apple Leaf Disease Detection. ACM Trans. Sen. Netw. March 2023.

[13] Plant Disease Detector [online], Google Play, dostupno na: https://play.google.com/store/apps/details?id=com.faisalkabirgalib.plant_disease_detection [20.08.2023.]

[14] J. V. Rissati, P. C. Molina and C. S. Anjos, "Hyperspectral Image Classification Using Random Forest and Deep Learning Algorithms," 2020 IEEE Latin American GRSS & ISPRS Remote Sensing Conference (LAGIRS), Santiago, Chile, 2020, pp. 132-132.

[15] Introduction to Convolution Neural Network, Geeks for Geeks [online], dostupno na: https://www.geeksforgeeks.org/introduction-convolution-neural-network [19. 08. 2023.]

[16] J. Wu, Introduction to Convolutional Neural Networks, National Key Lab for Novel Software Technology. Nanjing University. China, svibanj 2017.

[17] A. Gerber, C. Craig , Learn Android Studio, Apress, 2014. 42

[18] Tech Target Contributor, Android Studio [online], TechTarget, 2018, dostupno na: https://searchmobilecomputing.techtarget.com/definition/Android-Studio [19. 08. 2023.]

[19] Ahamed Shibly, Android Operating System: Architecture, Security Challenges and Solutions, 2016.

[20] Stefan Brahler, Analysis of the Android Architecture, 2010.

[21] Developers, Platform Architecture [online], Google Devellopers, 2020., Dostupno na: https://developer.android.com/guide/platform [19. 08. 2023.]

[22] M. Topolnik, M. Kušek, Uvod u programski jezik Java, FER Zageb, 2008.

[23] IBM Cloud Education, What is Java? [online], IBM, 2019., dostupno na: https://www.ibm.com/cloud/learn/java-explained [19. 08. 2023.]

[24] E. T. Ray, Learning XML, O'Really, 2001.

[25] Z. Ahmed, F. J. Kinjol and I. J. Ananya, "Comparative Analysis of Six Programming Languages Based on Readability, Writability, and Reliability," *2021 24th International Conference on Computer and Information Technology*, Dhaka, Bangladesh, 2021, pp.1-6.

[26] TensorFlow, TensorFlow guidem, dostupno na: https://www.tensorflow.org/guide [17.8.2023]

[27] Katie Terell Hanna, Google Firebase, dostupno na: https://www.techtarget.com/searchmobilecomputing/definition/Google-Firebase [17.8.2023.]

[28] M. Ohyver et al., The Comparison Firebase Realtime Database and MySQL Database Performance using Wilcoxon Signed-Rank Test, Procedia Computer Science, 2019., Vol. 157, pp. 396-405.

[29] Gerry, Plant Pathology 2021-resized 512 x 512, Kaggle, dostupno na: https://www.kaggle.com/datasets/gpiosenka/plant-pathology-2021resized-512-x-512 [27.6.2023.]

[30] G. Surekha, P. S. Keerthana, N. J. Varma and T. S. Gopi, "Hybrid Image Classification Model using ResNet101 and VGG16," *2023 2nd International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, Salem, India, 2023, pp. 729-734.

[31] Open Weather, dostupno na: https://openweathermap.org/ [20.8.2023.]

POPIS SLIKA

POPIS TABLICA

SAŽETAK

U ovom diplomskom radu proučavaju se problemi kod bolesti nasada jabuka. Navedeni su i prikazani postojeći sustavi za prepoznavanje bolesti lista jabuke. Osmišljena je i programski ostvarena mobilna Android aplikacija za potporu zaštiti nasada jabuka klasifikacijskim postupcima strojnog učenja koji se zasnivaju na treniranju modela konvolucijskih neuronskih mreža. Korisniku se nakon registracije omogućuje unos slika lista jabuke, prepoznavanje o kojoj se bolesti radi te podrška pri liječenje nasada. Prikazane su sve funkcionalnosti i ispitan je rad mobilne aplikacije. Ispitivanjem rada aplikacije potvrđena je ispravnost ostvarenih funkcionalnost aplikacije te dostatna točnost primijenjenih klasifikacijskih postupaka strojnog učenja.

Ključne riječi: bolesti jabuka, klasifikacija, konvolucijska neuronska mreža, mobilna Android aplikacija, sustav preporuka za liječenje.

ABSTRACT

Mobile Android Application for Apple Orchard Protection Support Using Machine Learning Classification Methods

This master's thesis examines issues related to apple orchard diseases. Existing systems for apple leaf disease recognition are mentioned and presented. A mobile Android application has been conceived and programmatically implemented to support apple orchard protection using machine learning classification methods based on training convolutional neural network models. After user registration, the application allows the input of apple leaf images, recognizes the disease, and provides support for orchard treatment. All functionalities are demonstrated, and the mobile application's performance has been evaluated. The testing confirmed the correctness of the implemented application functionalities and the sufficient accuracy of the applied machine learning classification methods.

Keywords: apple diseases, classification, convolutional neural network, mobile Android application, treatment recommendation system.

ŽIVOTOPIS

Josip Rizner rođen je 13.04.1999. godine u Pakracu. Pohađao je Češku osnovnu školu Jana Amosa Komenskog Daruvar u Daruvaru. Nakon toga upisuje srednju Tehničku školu u Daruvaru, smjer računalni tehničar. Nakon završetka srednjoškolskog obrazovanja upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku na sveučilištu Josipa Jurja Strossmayera, smjer preddiplomski studij računarstva te ga završava 2021. godine. Nakon toga upisuje diplomski studij računarstva, smjer informacijske i podatkovne znanosti na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Krajem 2021. godine počinje raditi u kompaniji Ericsson Nikola Tesla d.d. kao inženjer osiguranja kvalitete.