Mobilna Android aplikacija za prepoznavanje egzotičnog voća i povrća klasifikacijskim postupcima strojnog učenja

Šabanović, Dina

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

Sveučilišni preddiplomski studij

MOBILNA ANDROID APLIKACIJA ZA PREPOZNAVANJE EGZOTIČNOG VOĆA I POVRĆA KLASIFIKACIJSKIM POSTUPCIMA STROJNOG UČENJA

Završni rad

Dina Šabanović

Osijek, 2022.

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

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 13.09.2022.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Dina Šabanović
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R 4428, 22.07.2019.
OIB Pristupnika:	59160918940
Mentor:	Prof.dr.sc. Goran Martinović
Sumentor:	,
Sumentor iz tvrtke:	
Naslov završnog rada:	Mobilna Android aplikacija za prepoznavanje egzotičnog voća i povrća klasifikacijskim postupcima strojnog učenja
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rad:	U teorijskom dijelu rada treba objasniti probleme prepoznavanja objekata sa slika i postupke strojnog učenja pogodne za rješavanje ove vrste klasifikacijskih problema na slikama egzotičnog voća i povrća. Također, treba pronaći prikladne skupove podataka, odnosno slika egzotičnog voća i povrća, definirati značajke, izabrati algoritme strojnog učenja za prepoznavanje navedenog voća i povrća i ostvariti model strojnog učenja.
Prijedlog ocjene završnog 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: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	13.09.2022.
Datum potvrde ocjene od strane Odbora:	21.09.2022.

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, 21.09.2022.

Ime i prezime studenta:	Dina Šabanović
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R 4428, 22.07.2019.
Turnitin podudaranje [%]:	12

Ovom izjavom izjavljujem da je rad pod nazivom**: Mobilna Android aplikacija za prepoznavanje egzotičnog voća i povrća 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

U današnjem društvu, gdje su digitalne slike dio svakodnevice, potreba za otkrivanjem, prepoznavanjem i klasifikacijom slika sve je veća, a iako se klasifikacija slika već koristi u mnogim područjima, poput medicine ili prometa, svakodnevno se razvijaju nove primjene. Klasifikacijom se na osnovu obilježja ulaznog uzorka predviđa klasa kojoj uzorak pripada.

Cilj ovog rada je napraviti model strojnog učenja koji će, koristeći neuronsku mrežu, biti u mogućnosti prepoznavati o kojoj se vrsti voća ili povrća radi. Model je zatim potrebno implementirati u mobilnu aplikaciju, kako bi korisnici mogli pomoću kamere ili slike iz galerije otkriti o kojem se voću ili povrću radi.

U drugom poglavlju analizirat će se metode prepoznavanja objekata sa slike i ukratko će biti opisane već postojeće aplikacije za prepoznavanje voća i povrća. U trećem poglavlju bit će objašnjeni funkcionalni i nefunkcionalni zahtjevi aplikacije, te će biti objašnjena građa aplikacije. Četvrto poglavlje objasnit će kako i kojim tehnologijama su razvijeni model i mobilna aplikacija, dok će peto poglavlje dati uvid u način korištenja i rezultate aplikacije.

1.1. Zadatak rada

U teorijskom dijelu rada treba objasniti probleme prepoznavanja objekata sa slika i postupke strojnog učenja pogodne za rješavanje ove vrste klasifikacijskih problema na slikama egzotičnog voća i povrća. Također, treba pronaći prikladne skupove podataka, odnosno slika egzotičnog voća i povrća, definirati značajke, izabrati algoritme strojnog učenja za prepoznavanje navedenog voća i povrća i ostvariti model strojnog učenja. Nadalje, treba izabrati potrebne programske tehnologije, alate i programske okvire za pripremu postupka strojnog učenja i za razvoj mobilne Android aplikacije. Uz to, treba razraditi arhitekturu Android mobilne aplikacije s bazom podataka, te je programski ostvariti. Aplikacija treba na temelju implementiranog postupka strojnog učenja omogućiti prepoznavanje egzotičnog voća i povrća, te prikaz značajnih podataka o prepoznatim vrstama. Mobilnu aplikaciju i predloženi postupak strojnog učenja treba ispitati i analizirati na odgovarajućem skupu ulaznih podataka.

2. PREGLED STANJA I IZAZOVI U PODRUČJU PREPOZNAVANJA OBJEKATA SA SLIKE

Prema izvoru [1], klasifikacija ima dva različita značenja. Ukoliko je skup podataka dodijeljen s ciljem utvrđivanja postojanja klasa u podacima, riječ je o nenadziranom učenju. Ako je poznato da postoji određeni broj klasa, a skup podataka dodijeljen je s ciljem uspostavljanja pravila po kojem će se podaci svrstavati u jednu od postojećih klasa, radi se o nadziranom učenju.

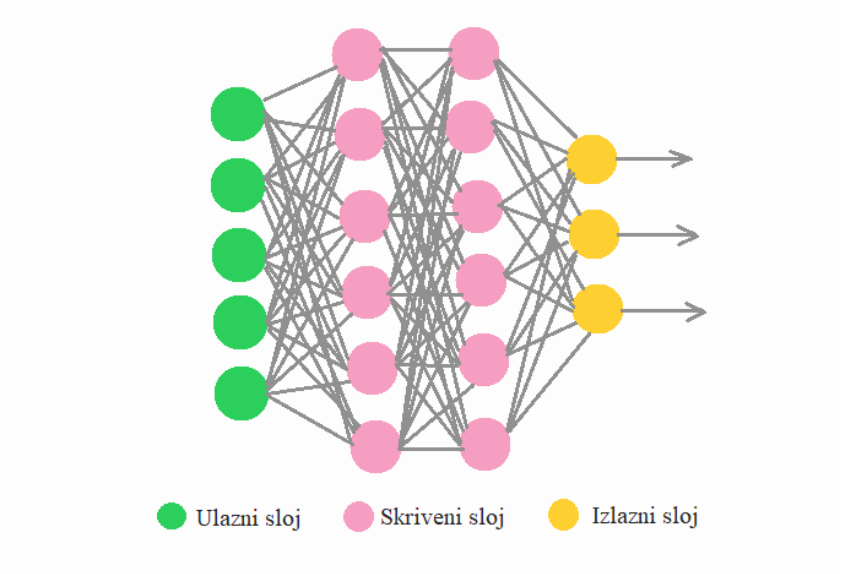
Prvo će biti objašnjene postojeće tehnike za klasifikaciju slika, a zatim će biti opisane već postojeće aplikacije koje rješavaju problem klasifikacije voća i povrća.

2.1. Tehnike prepoznavanja objekata sa slike

Kako je navedeno u izvoru [2], tehnike korištene za klasifikaciju slika su neuronske mreže, stablo odluke i strojevi potpornih vektora (*engl. Support Vector Machine*).

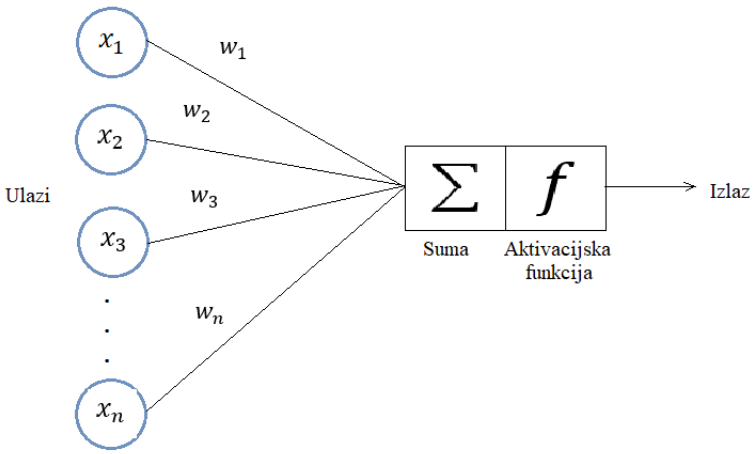
2.1.1 Neuronske mreže

Umjetne neuronske mreže (*engl. Artificial Neural Network, ANN*) opisane su u izvorima [3], [4] i [5]. Pojam "umjetna neuronska mreža" odnosi se na biološki inspirirano potpodručje umjetne inteligencije po uzoru na mozak inteligentnih bića, ideja datira još od pedesetih godina prošlog stoljeća, ali su tek posljednjih godina postale područje računarstva koje se iznimno istražuje. Umjetna neuronska mreža obično je računalna mreža temeljena na biološkim neuronskim mrežama koje grade strukturu ljudskog mozga. Ljudski mozak sastoji se od međusobno povezanih neurona, dok se umjetne neuronske sastoje od čvorova koji su međusobno povezani u različitim slojevima mreže. Ulazni sloj prihvaća unose u nekoliko različitih formata koje je osigurao programer. Skriveni sloj nalazi se između ulaznog i izlaznog sloja, on izvodi sve izračune kako bi se pronašle skrivene značajke i uzorci. Nakon što je ulaz prošao kroz niz transformacija pomoću skrivenog sloja, rezultat se prenosi pomoću izlaznog sloja. Arhitektura neuronske mreže prikazana je na slici 2.1.



Slika 2.1. Arhitektura neuronske mreže

Neuroni koji grade ljudski mozak su međusobno spojeni pomoću aksona i dendrita. Dendriti iz biološke neuronske mreže predstavljaju ulaze u umjetnim neuronskim mrežama, stanična jezgra predstavlja čvorove, sinapsa predstavlja težinu, a akson predstavlja izlaz iz umjetne neuronske mreže. Umjetni neuron nalazi se na slici 2.2.



Slika 2.2. Umjetni neuron

Nakon što se sumiraju ulazi u neuron, nad sumom je potrebno izvršiti funkciju aktivacije. Aktivacijska funkcija odnosi se na skup prijenosnih funkcija koje se koriste za postizanje željenog učinka. Postoje različite vrste aktivacijskih funkcija, a koriste se kako bi neuronska mreža bila nelinearna. Kada bi postojao samo linearan izlaz iz svakog sloja neuronske mreže, sve slojeve bi se moglo zamijeniti jednim slojem zbog nedostatka nelinearnosti, te ne bi bilo moguće napraviti kompleksniji model klasifikacije. Aktivacijske funkcije razlikuju se po matematičkim

formulama, odnosno načinu na koji djeluju nad ulaznim podacima. Izuzetno su bitne prilikom konstruiranja neuronske mreže jer o odabiru aktivacijske funkcije ovisi brzina učenja, a često i preciznost i performanse same neuronske mreže.

Mrežu je moguće gledati kao kompleksni računski graf u kojem je jedinica računanja neuron. Neuronske mreže je moguće klasificirati na nekoliko različitih modela, ali sve rade na sličnom principu, imaju ulaze iz kojih se kroz nekoliko slojeva treba kreirati smisleni izlaz koji su naučile postupkom učenja. Slojevi se sastoje od neurona koji su simulirani pomoću aktivacijskih funkcija. Svaki neuron posjeduje vlastitu težinu (w) koja se propagira na sljedeći sloj kroz aktivacijsku funkciju i na taj se način dobivaju izlazi. Za učenje neuronske mreže potrebno se odlučiti za jedan od nekoliko optimizacijskih algoritama kao što su *Momentum*, *Adagrad*, *Adadelta*, *Adam*, *Nadam*, *RMSprop*, *AdaMax* i drugi.

Prilikom učenja neuronske mreže, prvo je potrebno poznavati ulazne podatke i očekivane izlazne podatke, a zatim je potrebno poznavati i dio matematike koji se bavi statističkim pogreškama. Funkcije koje računaju pogrešku nazivaju se funkcije gubitka, a opisane su u izvoru [6]. Funkcije gubitka jedan su od najvažnijih aspekata neuronskih mreža jer su one i optimizacijske funkcije izravno odgovorne za prilagođavanje modela zadanim podacima za treniranje. Funkcija gubitka uspoređuje ciljane i predviđene izlazne vrijednosti, točnije, mjeri koliko dobro neuronska mreža modelira podatke o treningu. Prilikom učenja mreže, cilj je minimizirati ovaj gubitak između predviđenih i ciljnih rezultata. Nakon što je izračunata funkcija gubitka, potrebno je izračunati i gradijent. U strojnom učenju, gradijent je derivat funkcije koja ima više od jedne ulazne varijable. Poznat kao nagib funkcije u matematičkim terminima, gradijent jednostavno mjeri promjenu u svim težinama s obzirom na promjenu pogreške [7].

Prema izvoru [8], prednosti umjetnih neuronskih mreža su:

- Pohranjivanje informacija na cijeloj mreži – Pohranjivanjem informacija na cijeloj mreži, a ne u bazi podataka, omogućava se da nestanak nekoliko informacija na jednom mjestu ne sprječava funkcioniranje mreže.
- Sposobnost rada s nepotpunim znanjem – Nakon što je istrenirana, neuronska mreža, može proizvesti rezultate čak i s nepotpunim informacijama.
- Tolerancija na pogreške – Oštećenje jedne ili više ćelija neuronske mreže ne sprječava generiranje izlaza.
- Postupno oštećenje – Mreža se s vremenom usporava i podvrgava relativnoj degradaciji. Problem s mrežom ne nastaje odmah.

- Mogućnost paralelne obrade – Umjetne neuronske mreže imaju brojčanu snagu koja može obavljati više od jednog posla u isto vrijeme.

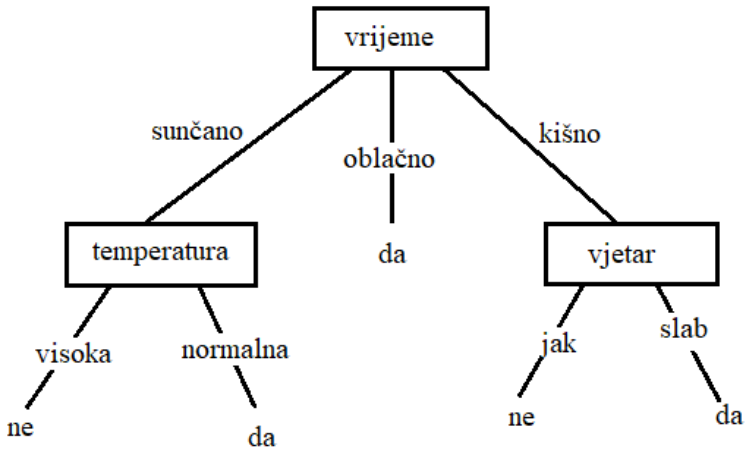
Izvor [8] objašnjava i nedostatke umjetnih neuronskih mreža:

- Ovisnost o hardveru – Umjetne neuronske mreže zahtijevaju procesore s paralelnom procesorskom snagom, u skladu s njihovom strukturom.
- Neobjašnjivo ponašanje mreže – Kada umjetna mreža proizvede rješenje za ispitivanje, ne daje naznaku zašto i kako. To smanjuje povjerenje u mrežu.
- Određivanje pravilne strukture mreže – Ne postoji posebno pravilo za određivanje strukture umjetnih neuronskih mreža, odgovarajuća struktura mreže postiže se iskustvom.
- Trajanje mreže je nepoznato – Kada je mreža smanjena na određenu vrijednost pogreške, znači da je učenje završeno. Ova nam vrijednost ne daje optimalne rezultate.

Umjetne neuronske mreže, iako su kroz godine napredovale, još uvijek ne mogu premašiti ljudsku inteligenciju zbog nedostatka računalne snage i kompleksne arhitekture mozga koju još uvijek nije moguće postići.

2.1.2. Stablo odluka

U izvoru [9] opisana su stabla odluka. Jedan od najintuitivnijih alata za klasifikaciju podataka su upravo stabla odlučivanja koja su cijenjena zbog lakog tumačenja i korištenja. Stablo odluka je usmjereno stablo slično dijagramu toka, sastoji se od čvorova, grana i listova. Svaki čvor predstavlja neku značajku podatka kojeg se želi klasificirati, a grane koje idu iz čvorova predstavljaju vrijednosti koje ta značajka može poprimiti. Kada je neki podatak potrebno klasificirati, granama se putuje kroz stablo dok se ne dođe do nekog od listova koji predstavljaju klase u koje je moguće klasificirati podatak. Stabla odlučivanja mogu se koristiti i s brojčanim i kategoričkim atributima. Postoje metode koje se bave rješavanjem problema nesigurnih vrijednosti ili slučajeva kada pojedine vrijednosti nedostaju. Na slici 2.3. prikazano je jednostavno stablo odlučivanja koje rješava problem odluke je li dan pogodan za igranje tenisa ili ne.

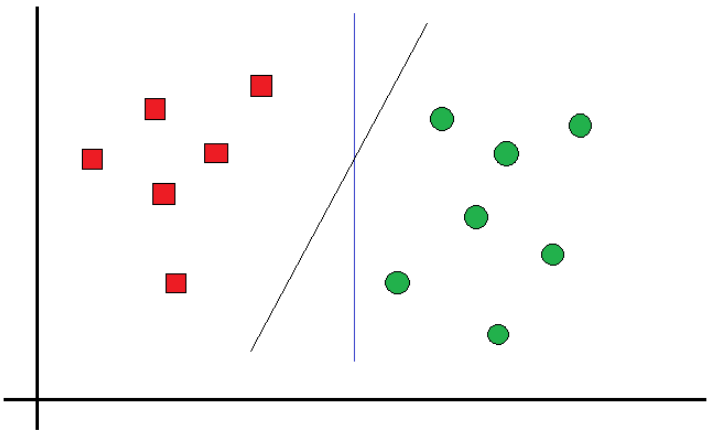


Slika 2.3. Jednostavno stablo odluka

Stablo odluka počinje od korijena, ako je vrijeme sunčano, spušta se na idući čvor, te se odgovara na pitanje kakva je temperatura, ako je temperatura normalna, a ne visoka, dan je pogodan za igranje tenisa, u suprotnom dan nije pogodan za igranje. Stabla odluka imaju razne primjene, primjerice u medicini, ali i u financijama.

2.1.3. Support Vector Machine (SVM)

Osnove rada SVM-a opisane su u izvoru [10]. SVM je vrsta algoritma nadziranog strojnog učenja koji pruža analizu podataka za klasifikaciju i regresiju. Iako ih je moguće koristiti i za regresiju, SVM-om se uglavnom koristi za klasifikaciju. Osnovni princip iza rada ovog algoritma je jednostavan, potrebno je napraviti hiper-ravninu koja razdvaja skup podataka u klase. SVM se može objasniti na jednostavnom primjeru, pretpostavimo da je za dati skup podataka potrebno klasificirati crvene pravokutnike od zelenih krugova. Cilj je stvoriti liniju koja klasificira podatke u dvije klase stvarajući razliku između crvenih pravokutnika i zelenih krugova.



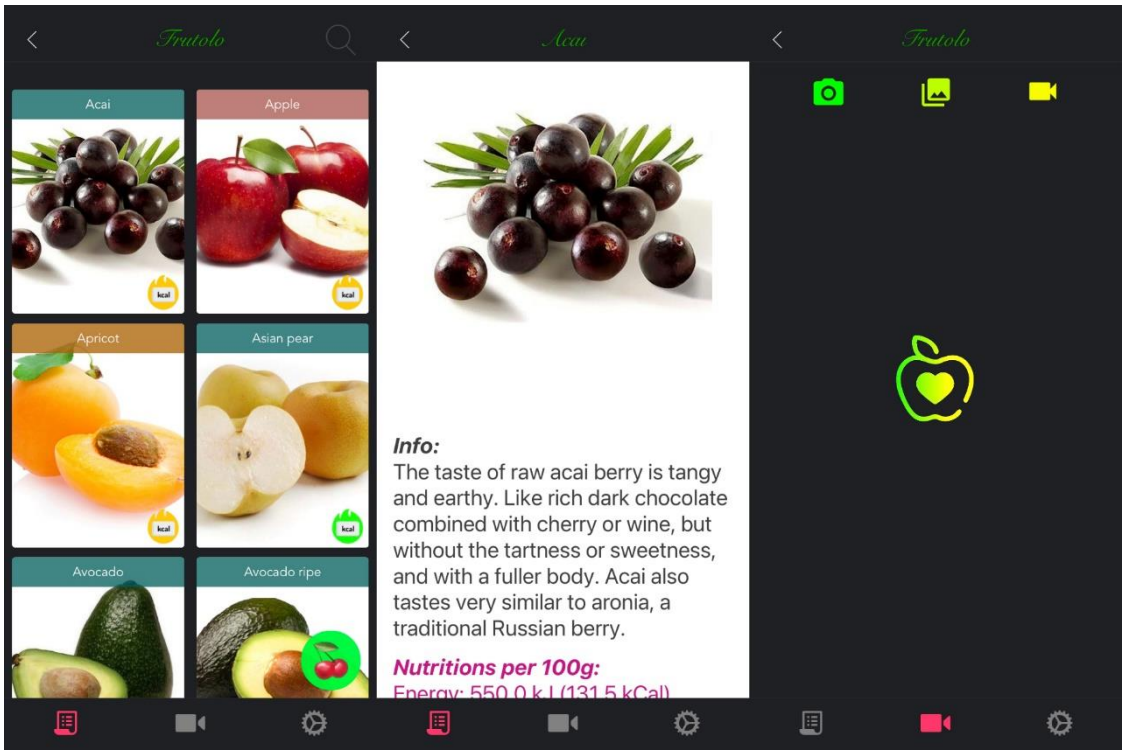
Slika 2.4. Problem koji je potrebno riješiti pomoću SVM algoritma

Sa slike 2.4. vidi se da je moguće pronaći više linija koje dijele podatke u dvije klase. Prema SVM-u, potrebno je pronaći točke koje leže najbliže objema klasama, te točke su poznate kao potporni vektori. U sljedećem koraku pronalazi se blizina razdjelne ravnine i vektora potpore. Udaljenost između točaka i razdjelne crte poznata je kao margina. Cilj SVM algoritma je maksimizirati marginu. Kada margina dosegne svoj maksimum, hiper-ravnina postaje optimalna.

Ako su podaci koje treba podijeliti u 2 dimenzije, hiper-ravnina će biti u jednoj dimenziji. Za veće dimenzije, recimo za n-dimenzionalni prostor, postojat će (n-1)-dimenzionalni podskup koji dijeli prostor na dvije komponente.

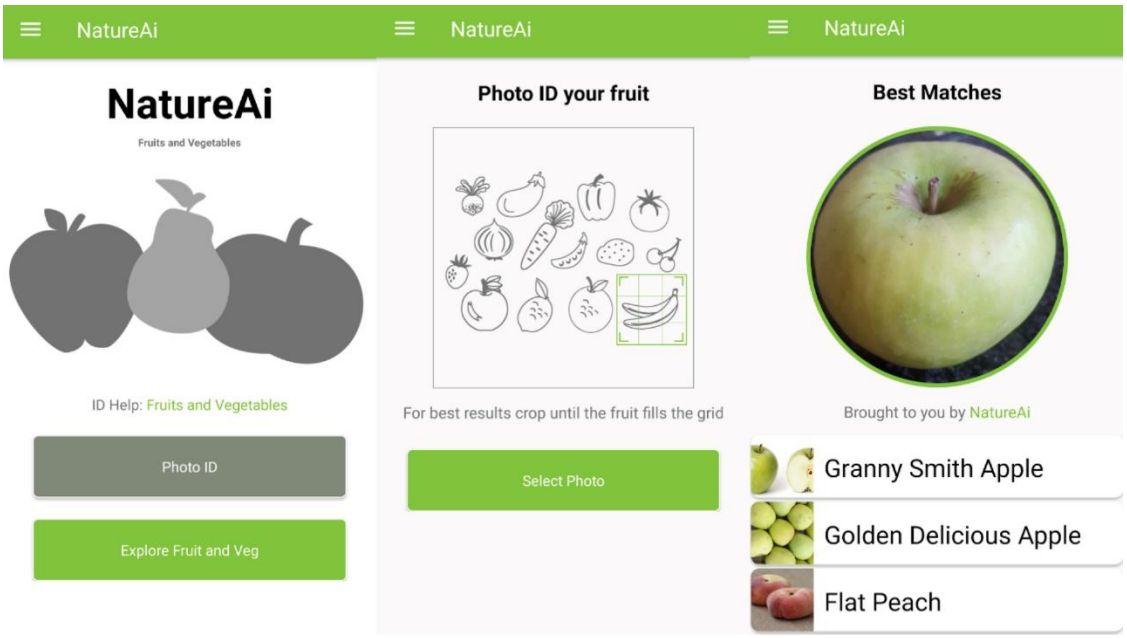
2.2. Analiza postojećih rješenja

Google Play i *AppStore* nude nekoliko aplikacija koje se mogu koristiti za prepoznavanje voća i povrća. *Frutolo* (Slika 2.5.) je prva aplikacija temeljena na strojnom učenju koja omogućuje izradu vlastite zdrave prehrane. Aplikacijom je moguće prepoznavati voće i povrće, te pronaći sve informacije o tome voću i povrću, poput količine vitamina i minerala i nutritivnih vrijednosti. Sliku namirnice koju želimo identificirati moguće je učitati iz galerije ili slikati ili snimiti video kamerom pametnog telefona. Aplikacija može prepoznati 50 vrsta voća i povrća, a sadrži listu s informacijama o njih 80 [11]. Aplikacija je besplatna, na platformi *Google Play* još nije ocijenjena, dok je na *AppStore-u* ocijenjena s 4.5 od 5, ali neki korisnici su se žalili da je aplikacija dobra samo za prepoznavanje jednostavnog voća i povrća.



Slika 2.5. Izgled *Frutolo* aplikacije

Na *Google Play-u* je moguće pronaći besplatnu aplikaciju *Fruits and Vegetables Identification & Scanner* [12]. U opisu aplikacije navedeno je da aplikacije može prepoznavati voće i povrće koje viđamo svaki dan i imamo kod kuće, ali i ono egzotično. Aplikacija nakon identifikacije voća vodi na stranicu Wikipedije kako bi saznali više o onome što je identificirano. *Fruits and Vegetables Identification & Scanner* ima više od deset tisuća preuzimanja, ali aplikacije je ocijenjena sa samo 1.8 od 5.



Slika 2.6. Izgled aplikacije *Fruits and Vegetables Identification & Scanner*

3. MODEL I GRAĐA MOBILNE APLIKACIJE

U ovom poglavlju navedeni su funkcionalni i nefunkcionalni zahtjevi aplikacije, te je prikazan i objašnjen dijagram toka aplikacije.

3.1. Analiza zahtjeva na mobilnu aplikaciju

3.1.1. Funkcionalni zahtjevi

Funkcionalni zahtjevi objašnjeni su u izvoru [13]. Funkcionalni zahtjevi su oni koji definiraju što bi aplikacija trebala raditi. To su ključne funkcije bez kojih aplikacija ne bi funkcionirala ili radila ono što bi trebala raditi, ti zahtjevi moraju se ispuniti i bez njih se ne može.

Iz definicije može se zaključiti da su funkcionalni zahtjev ove aplikacije:

- Prepoznavanje vrste voća ili povrća s uslikane fotografije
- Prepoznavanje vrste voća ili povrća sa slike učitane iz galerije
- Prikaz informacija o voću i povrću

3.1.2. Nefunkcionalni zahtjevi

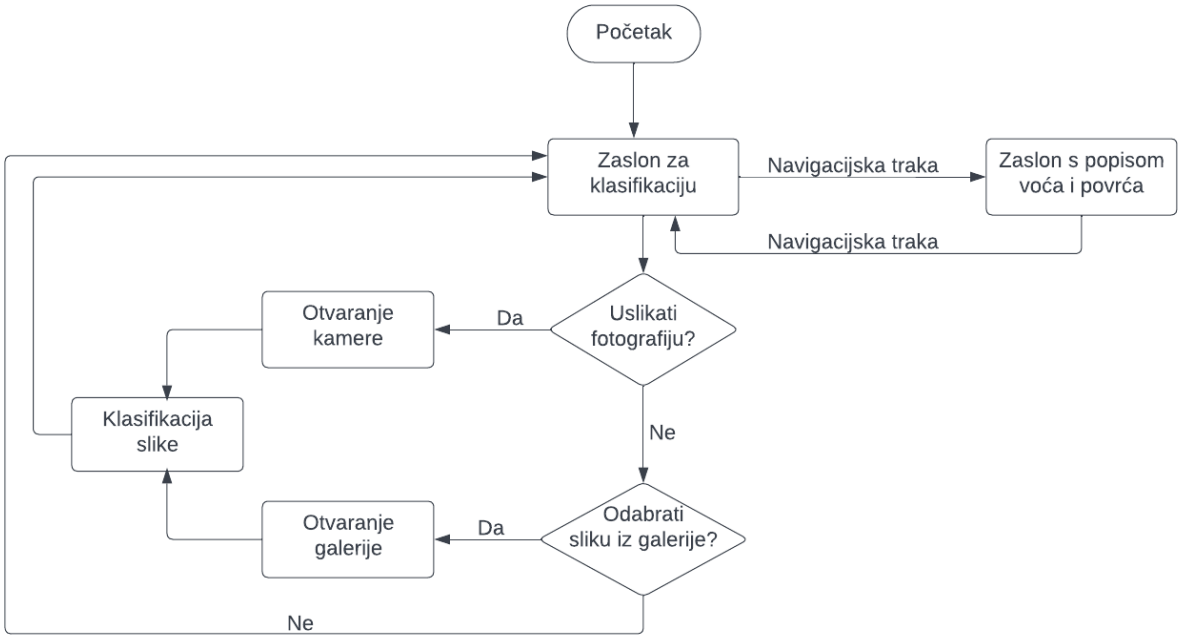
Nefunkcionalni zahtjevi opisani su u izvoru [14]. To su zahtjevi koji definiraju kako bi aplikacija trebala obavljati određenu funkciju, oni su atributi kvalitete aplikacije koji definiraju korisničko iskustvo. Trebaju se implementirati prema njihovom prioritetu u odnosu na funkciju aplikacije što ih čini fleksibilnima u određenoj mjeri. Moguće ih je preskočiti nekoliko u slučaju vremenskih, proračunskih ili tehnoloških ograničenja, ali ukoliko bi aplikacija bila ostvarena samo s funkcionalnim zahtjevima, bila bi nepouzdana i nepotpuna.

U nastavku će biti ukratko opisani nefunkcionalni zahtjevi ove aplikacije:

- Izvođenje – Učinkovitost aplikacije može se odrediti prema vremenu odziva, vremenu potrebnom za dovršavanje danog zadatka. U ovoj aplikaciji rezultat klasifikacije voća trebao bi biti prikazan unutar 1-2 sekunde.
- Upotrebljivost – Korisnici bi trebali moći koristiti aplikaciju bez ikakvih smjernica ili pomoći.
- Adaptacija zaslona – Mobilni uređaji dolaze s različitim veličinama zaslona i izgledom, a aplikacija bi trebala moći prikazati svoj izgled na različitim veličinama zaslona.

3.2. Dijagram toka aplikacije

Kada korisnik otvori aplikaciju, otvara se zaslon na kojemu korisnik može odabrati želi li uslikati ili odabrati iz galerije sliku koju želi klasificirati. Nakon što je odabran način na koji će se učitati slika, ona se prosljeđuje modelu koji ju klasificira, a na početnom zaslonu se prikazuje slika koju smo klasificirali i rezultat klasifikacije. Korisnik zatim ponovno može odabrati novu sliku koju želi klasificirati ili se u bilo kojem trenutku može pomoću navigacijske trake na dnu aplikacije prebaciti na zaslon s popisom voća i povrća. Također, ukoliko se korisnik nalazi na zaslonu s popisom voća i povrća, pomoću navigacijske trake se lako može vratiti u zaslon za klasifikaciju. Dijagram toka aplikacije prikazan je na slici 3.1.



Slika 3.1. Dijagram toka aplikacije

3.3. Građa aplikacije

Aplikacija se sastoji od dva zaslona, s jednoga na drugi moguće je prijeći koristeći navigaciju koja se nalazi na dnu oba zaslona. Ukoliko se korisnik nalazi na prvom zaslonu i odabere *Take Photo*, otvorit će se kamera i korisnik može uslikati fotografiju. Nakon što je fotografija snimljena, ona se prosljeđuje modelu. Kada je klasifikacija završena, slika i rezultat klasifikacije prikazuju se na zaslonu. Ako korisnik na prvom zaslonu odabere *Choose from Gallery*, otvara se galerija, korisnik iz galerije odabire sliku i postupak je isti kao i sa snimljenom fotografijom.

Ukoliko se korisnik nalazi na drugom zaslonu, može pročitati podatke o voću i povrću ili se vratiti na zaslon za klasifikaciju. Podaci i slike voća i povrća na drugom zaslonu dohvaćeni su iz baze podataka i smješteni u listu u aplikaciji.

4. PROGRAMSKO RJEŠENJE MOBILNE APLIKACIJE ZA PREPOZNAVANJE EGZOTIČNOG VOĆA I POVRĆA

U ovome poglavlju su opisane tehnologije korištene za kreiranje modela strojnog učenja korištenoga za klasifikaciju slika voća i povrća, zatim su objašnjeni alati korišteni za razvoj aplikacije. U trećem dijelu opisani su podaci korišteni za razvoj modela strojnog učenja. Na kraju poglavlja prikazano je programsko rješenje za razvoj modela i mobilne aplikacije.

4.1. Alati i tehnologije za klasifikaciju slika

4.1.1. Python

Python [15] je interpretirani programski jezik visoke razine i opće namjene. Filozofija njegovog dizajna naglašava čitljivost koda uz korištenje značajnog uvlačenja. Python podržava više paradigmi programiranja, od strukturiranog, a posebno proceduralnog programiranja, preko objektno orijentiranog do funkcionalnog programiranja. Često se opisuje kao jezik "uključene baterije" zbog svoje opsežne standardne biblioteke. Mana Pythona je njegova sporost, programi napisani u njemu sporije se izvode od programa napisanih u programskim jezicima C ili C++. Najveća razlika između Pythona i ostalih programskih jezika je u tome što Python kao metodu razlikovanja koristi uvlačenje, a ne vitičaste zagrade ili ključne riječi kao većina programskih jezika. Povećanje uvlačenja znači da dolazi novi, ugniježđeni blok, dok smanjenje označava kraj trenutnog bloka. Python je jezik jednostavan za učenje i pisanje što ga čini odličnim jezikom za početnike.

4.1.2. Juptyter Notebook

Jupyter Notebook [16] je interaktivno računalno okruženje koji se pokreće u internetskom pregledniku. Jupyter Notebook dokument sadrži uređeni popis ulaznih i izlaznih ćelija koje mogu sadržavati programski kod, tekst, grafikone i slične medije i obično završava s nastavkom *.ipynb.* Može se povezati s raznim jezgrama kako bi omogućio programiranje u različitim programskim jezicima. Jupyiter jezgra je program odgovoran za rukovanje različitim zahtjevima, a s ostalim komponentama komunicira koristeći *ZeroMQ*, stoga komponente mogu biti na istim ili udaljenim strojevima. Kompatibilan je s mnogim programskim jezicima kao što su Python, R, Julia i Haskell. Jupyter Notebook dokumente moguće je preuzeti u brojnim standardnim formatima (HTML, PDF, LaTeX, Python). Posebna prednost Jupyter Notebooka je mogućnost pokretanja samo dijelova programskog koda koji se želi testirati bez pokretanja cijelog programa.

4.1.3. TensorFlow

Kako je objašnjeno u izvoru [17], strojno učenje je složena disciplina, ali implementacija modela strojnog učenja daleko je manje zahtjevna nego što je bila zahvaljujući platformama koje olakšavaju proces stjecanja podataka, modela obuke, predviđanja i usavršavanja budućih rezultata. TensorFlow objedinjuje niz modela i algoritama strojnog učenja i dubokog učenja i čini ih korisnim putem uobičajenih programskih metafora. Koristi Python ili JavaScript za pružanje prikladnog *front-end* API-ja za izgradnju aplikacija, a aplikacije izvršava u programskom jeziku C++. TensorFlow omogućava kreiranje grafove protoka podataka. Svaki čvor u grafu predstavlja matematičku operaciju, a svaka veza između čvorova je višedimenzionalni niz podataka ili tenzor. TensorFlow aplikacije mogu se pokretati na lokalnom računalu, u oblaku, iOS i Android uređajima, CPU-ovima ili GPU-ovima. Najveća prednost koju TensorFlow pruža za razvoj strojnog učenja je apstrakcija. Umjesto da se bavi sitnim detaljima implementacije algoritama ili pronalaženjem ispravnih načina povezivanja izlaza jedne funkcije s ulazom druge, programer se može usredotočiti na cjelokupnu logiku aplikacije.

4.1.4. Keras

Keras [18] je aplikacijsko programsko sučelje (engl. *Application Programming Interface*, API) napisano u Pythonu koje radi na platformi TensorFlow. Razvijen je s naglaskom na omogućavanje brzog eksperimentiranja. Keras sadrži brojne implementacije najčešće korištenih građevnih blokova neuronske mreže poput slojeva, ciljeva, aktivacijskih funkcija, optimizatora. Sadrži i niz alata koji olakšavaju rad sa slikovnim i tekstualnim podacima kako bi pisanje koda duboke neuronske mreže bilo što jednostavnije. Keras smanjuje kognitivno opterećenje programera kako bi se usredotočili na dijelove problema koji su stvarno važni. Keras pruža performanse i skalabilnost visoke industrije, koriste ga velike organizacije i tvrtke, uključujući NASA-u, YouTube i Waymo.

4.2. Alati i tehnologije za izradu Android aplikacije

4.2.1. Flutter

Flutter je razvojni okvir otvorenog koda za razvoj višeplatformskih aplikacija koji je razvio Google. Moguće je razvijati aplikacije za Android, iOS, Linux, macOS, Windows, Google Fuchsiu i web iz iste baze koda. Flutter koristi Dart programski jezik i mnoge njegove napredne mogućnosti [19]. Izvor [20] navodi neke od značajki koje čini Flutter odličnim alatom za razvoj kvalitetnih aplikacija. Jedna od tih značajki jest da Flutter ima ugrađeno ponovno pokretanje (*engl. Hot reload*) zbog kojeg je moguće odmah, bez ponovnog pokretanja aplikacije, vidjeti promjene napravljene u programskom kodu. Također, Flutter omogućava korištenje stotine unaprijed napravljenih ili prilagođenih *widgeta* i olakšava implementaciju prilagodljivog i responzivnog dizajna kako bi aplikacije izgledale dobro na svakom zaslonu. Flutter aplikacije moguće je izgraditi koristeći Visual Studio Code ili Android Studio s dodacima za poboljšanje tijeka rada i integriranim razvojnim alatima. Još jedna prednost Fluttera je bogata dokumentacija koju je moguće pronaći na službenoj stranici i drugim izvorima, uključujući mnogobrojne videozapise koji olakšavaju snalaženje apsolutnim početnicima, ali i onima koji već imaju iskustva u programiranju.

4.2.2. Dart

Dart [21] je programski jezik opće namjene otvorenog koda kojeg je razvio Google 2011., a kasnije ga je internacionalna neprofitna organizacija za standardizaciju informacijskih i komunikacijskih sustava, ECMA, odobrila kao standard. To je čisto objektno orijentirani jezik sa sintaksom koja je slična sintaksi Jave, C#-a i Kotlina, te ga je zbog toga lako naučiti ukoliko se poznaje neki od navedenih jezika. Dart je jezik optimiziran za klijenta za razvoj brzih aplikacija na bilo kojoj platformi. Dart podržava programske alate kao što su sučelje, zbirke, klase i dinamičko tipkanje. Koristi se za izradu *front-end* korisničkog sučelja za mobilne i web aplikacije. Dart koristi i statičku provjeru tipa i provjere vremena izvođenja kako bi potvrdio da vrijednost varijable uvijek odgovara statičkom tipu varijable. Koristan je programski jezik za skriptiranje, programiranje i jezik za označavanje, moguće ga je koristiti za različite aplikacije zbog mnogobrojnih biblioteka. Kao i za Flutter, za Dart je na službenoj stranici moguće pronaći bogatu dokumentaciju.

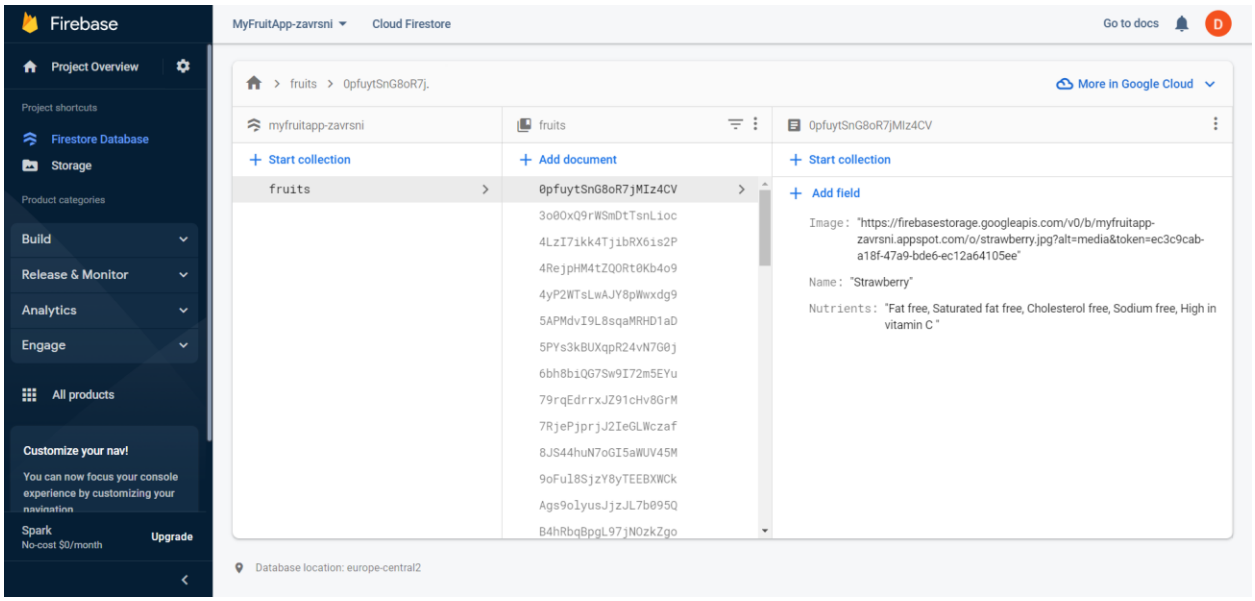
4.2.3. Visual Studio Code

Visual Studio Code je uređivač izvornog koda kojeg je razvio Microsoft i dostupan je za Windows, macOS i Linux. Dolazi s ugrađenom podrškom za JavaScript, TypeScript i Node.js, ali ima bogat sustav proširenja za druge jezike i okruženja. Visual Studio Code podržava označavanja sintakse i podudaranje zagrada, pametno dovršavanje, otklanjanja pogrešaka i refaktoriranje za različite programske jezike [22].

U anketi na internetskoj stranici *StackOverflow*, Visual Studio Code već je nekoliko godina rangiran kao najpopularnije integrirano razvojno okruženje (*engl. Integrated Development Environment,* IDE), tako je i 2022. zauzeo prvo mjesto sa 74.48% glasova, a glasovalo je 71 010 ispitanika [23].

4.2.4. Cloud Firestore

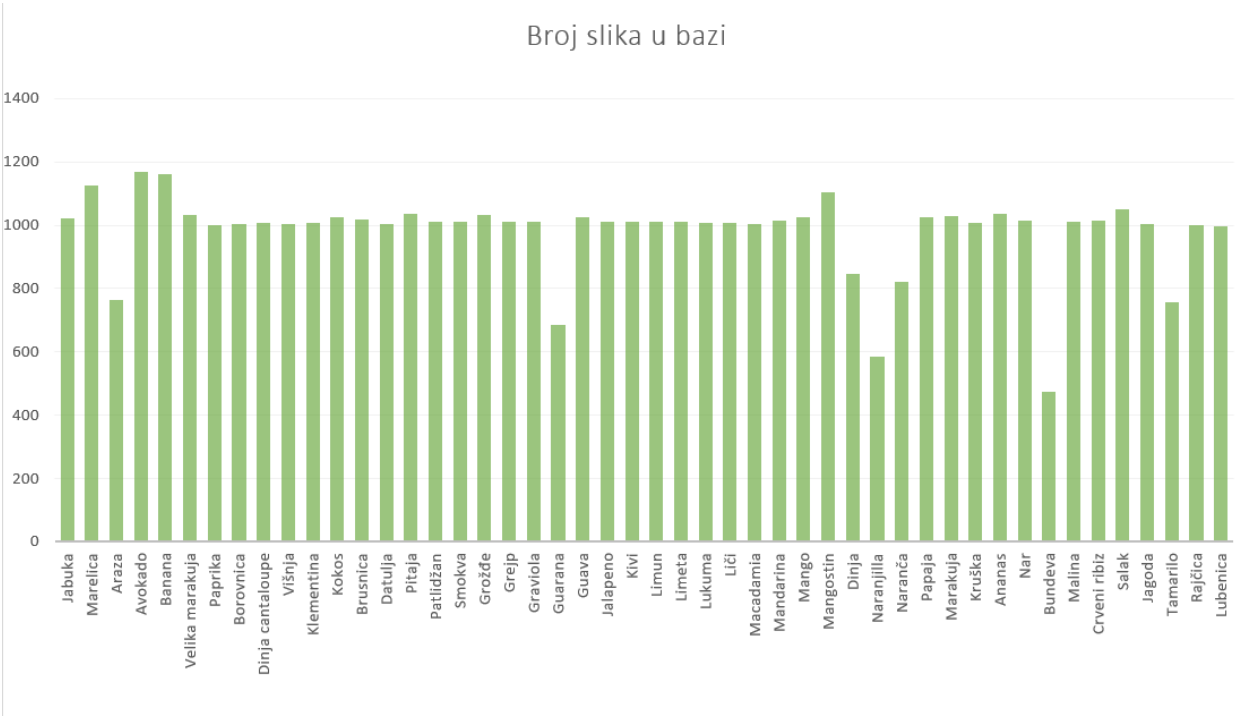
Na službenoj Firebase internetskoj stranici [24] moguće je pronaći informacije o svim njihovim proizvodima. Cloud Firestore je fleksibilna, skalabilna baza podataka za mobilni, web i razvoj poslužitelja razvijen od Firebase-a i Google Clouda. Održava sinkronizaciju podataka u klijentskim aplikacijama i nudi izvanmrežnu podršku za mobilne uređaje i web kako bi bilo moguće izraditi aplikacije koje rade bez obzira na internetsku povezanost. Cloud Firestore, također, nudi i odličnu integraciju s drugim Firebase i Google Cloud proizvodima. Slijedeći *NoSQL* podatkovni model, podatci su u Cloud Firestore-u pohranjeni u dokumente koji sadrže mapiranje polja u vrijednosti, a ti su dokumenti pohranjeni u zbirkama koje su spremnici za dokumente koji se mogu koristiti za organiziranje podataka i izradu upita. Dokumenti podržavaju mnoge različite vrste podataka, od jednostavnih nizova i brojeva do složenih, ugniježđenih objekata. Također, moguće je stvaranje podzbirki unutar dokumenata i izgradnja hijerarhijske strukture podataka.



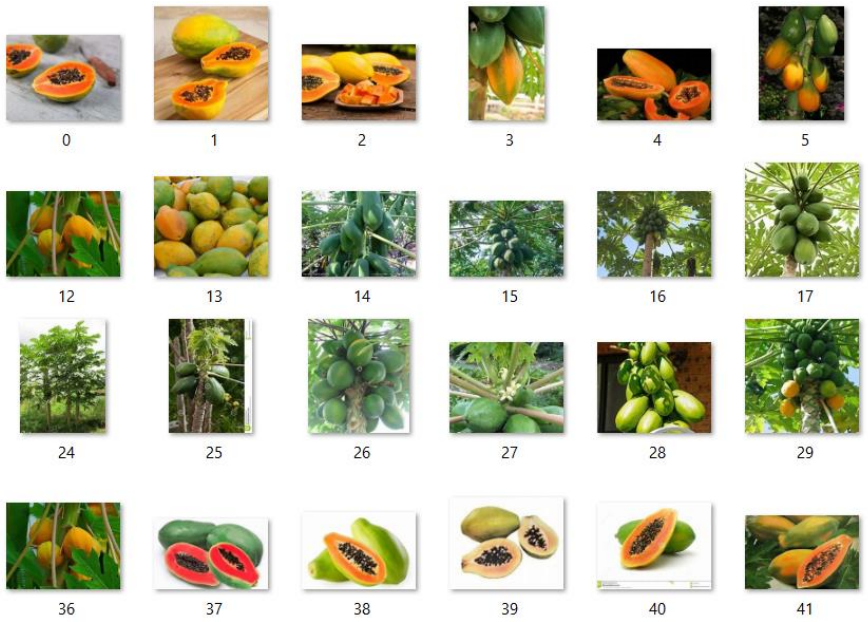
Slika 4.1. Cloud Firestore baza podataka korištena u aplikaciji

4.3. Podaci za strojno učenje

Kako bi se ostvario model strojnog učenja, potreban je skup podataka na kojemu će neuronska mreža biti istrenirana. Slike korištene za ovaj model preuzete su iz baze pod nazivom *Fruits-262* [25] koja sadrži 225 640 slika 262 različite vrste voća i povrća. Za potrebe ovog rada je, zbog veličine originalne baze, korišteno 47 046 slika podijeljenih u 48 klasa, koliko će vrsta voća i povrća ovaj model biti u mogućnosti prepoznati. Broj slika koje se nalaze u bazi za svaku vrstu varira, ali za većinu vrsta baza sadrži oko 1000 slika. Za neke vrste broj slika je ipak nešto manji, a najmanje slika u bazi postoji za bundevu, njih 475.



Slika 4.2. Broj slika u bazi za svaku vrstu voća



Slika 4.3. Slike papaje iz baze

4.4. Razvoj modela

Za razvoj modela strojnog učenja koristiti će se Tensorflow i Keras biblioteka, a program će biti napisan programskim jezikom Python u Jupyter Notebooku. Nakon dohvaćanja fotografija, pomoću *ImageDataGenerator* klase tenzori se skaliraju s vrijednosti između 0 i 255 na vrijednosti između 0 i 1, budući da neuronske mreže preferiraju rad s malim ulaznim vrijednostima [26]. Pomoću iste klase također se stvaraju generatori koji će predati slike modelu. Slike će biti podijeljene u dva dijela, *validation_split* govori da će se 20% slika koristiti kao podaci za provjeru valjanosti. Model će izdvojiti ovaj dio podataka o obučavanju, neće se na njemu uvježbavati i procijenit će gubitak modela na ovim podacima [27]. Metoda *flow_from_directory* učitava slike s diska i mijenja veličinu slika u potrebne dimenzije. Varijabla *batch_size* govori nam u kolikim grupama će slike biti procesuirane, ovdje je to 32.

```
IMAGE_SIZE = 224
BATCH_SIZE = 32

datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale = 1./255,
                                                          validation_split = 0.2)

train_generator = datagen.flow_from_directory(
base_dir,
target_size = (IMAGE_SIZE, IMAGE_SIZE),
batch_size = BATCH_SIZE,
    subset = 'training'
)
val_generator = datagen.flow_from_directory(
    base_dir,
    target_size = (IMAGE_SIZE, IMAGE_SIZE),
    batch_size = BATCH_SIZE,
    subset = 'validation')
```

Slika 4.4. Učitavanje i transformacija slika i generatori

4.4.1. Preneseno učenje

Preneseno učenje je tehnika strojnog učenja kojom se model istreniran i razvijen za jedan zadatak ponovno koristi za drugi, povezani zadatak. Odnosi se na situacije u kojoj se ono što je naučeno u jednom okruženju iskorištava za poboljšanje optimizacije u drugom [28]. Konvolucijski slojevi bliži ulaznom sloju modela uče značajke niske razine kao što su linije, a slojevi bliže izlazu interpretiraju izdvojena obilježja u kontekstu zadatka klasifikacije. Zbog toga se, u ovisnosti koliko se novi zadatak razlikuje od klasificiranih objekata, slojevi bliže izlazu mogu zamrznuti [29].

Ovdje je korištena MobileNetV2 arhitektura konvolucijske neuronske mreže koja nastoji dobro funkcionirati na mobilnim uređajima. Mreža je već istrenirana na *ImageNet* bazi koja sadrži više milijuna označenih slika.

```
IMG_SHAPE = (IMAGE_SIZE, IMAGE_SIZE, 3)
base_model = tf.keras.applications.MobileNetV2(
    input_shape = IMG_SHAPE,
    include_top = False,
    weights = 'imagenet'
)
```

Slika 4.5. Učitavanje baznog modela

Nakon učitavanja baznog modela, a prije treniranja modela, važno je zamrznuti konvolucijsku bazu. Zamrzavanje sprječava ažuriranje težina u danom sloju tijekom treniranja modela [30]. MobileNetV2 ima 53 sloja, postavljanjem *base_model.trainable = False* zamrznuti će se svi slojevi.

```
base_model.trainable = False
model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.Conv2D(32, (3,3), activation = 'relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(48, activation = 'softmax')
])
```

Slika 4.6. Dodavanje slojeva

Slojevi su sekvencijalno grupirani, prvo bazni model, zatim 2D konvolucijski sloj s 32 filtera, veličinom jezgre 3x3 i linearnom aktivacijskom funkcijom. Nakon toga dodan je sloj ispadanja sa stopom od 20% kako bi se spriječilo pretjerano podudaranje (*engl. overfitting*). Prije gustog sloja *GlobalAveragePooling2D* sažima sve značajki u jednu, udružujući sve relevantne informacije u jednu koju jedan gusti sloj može lako razumjeti. Na kraju se nalazi gusti sloj, 48 odgovara broju klasa, a *softmax* se koristi kao aktivacija za posljednji sloj mreže jer se onda rezultat može interpretirati kao distribucija vjerojatnosti.

```
model.compile(
     optimizer=tf.keras.optimizers.Adam(),
     loss='categorical_crossentropy',
     metrics=['accuracy']
)
```

Slika 4.7. Prevođenje modela

Kod prevođenja modela potrebno je odrediti neke parametre. U *optimizer* se sprema koji algoritam će se koristiti za prilagodbu atributa modela s ciljem da gubitci budu što manji. U ovome slučaju korišteni algoritam je Adam, *loss='categorical_crossentropy'* računa gubitak unakrsne entropije između predviđenih i pravih oznaka, a *metrics=['accuracy']* govori da se mreža prati prema tome koliko su često predviđanja jednaka pravim oznakama.

```
epochs = 10
history = model.fit(
          train_generator,
          epochs = epochs,
          validation_data = val_generator)
```

Slika 4.8. Učenje modela

```
keras_file = "MobileNetV2_Model.h5"
keras.models.save_model(model , keras_file)
```

Slika 4.9. Spremanje modela

Učenje modela provodit će se u 10 epoha, a funkciji se predaju ranije pripremljeni generatori i broj epoha. Model se zatim sprema u .h5 obliku.

4.4.2. Pretvorba modela u TFLite oblik

Za ugradnju modela strojnog učenja u mobilnu aplikaciju koristi se *tflite* biblioteka, stoga je potrebno model iz *.h5* oblika pretvorili u *.tflite* oblik.

```
My_TFLite_Model1 = tf.keras.models.load_model('MobileNetV2_Model.h5')
converter = tf.lite.TFLiteConverter.from_keras_model(My_TFLite_Model1)
tflite_model1 = converter.convert()
open("My_TFLite_Model1.tflite", "wb").write(tflite_model1)
```

Slika 4.10. Pretvorba modela iz .h5 u .tflite oblik

Spremljeni model učitava se u varijablu *My_TFLite_Model1* i nad njom se poziva funkcija *from_keras_model*. U varijablu *tflite_model1* sprema se pretvoreni model, a zatim ga se zapisuje u *My_TFlite_Model1.tflite* datoteku.

4.5. Razvoj mobilne aplikacije

U ovom dijelu rada će biti objašnjeno kako je implementiran model u aplikaciju, kako je implementirana baza podataka i kako je nastao izgled aplikacije koristeći Flutter.

4.5.1. Implementacija modela u aplikaciju

Kako bi bilo moguće koristiti sve zamišljene funkcionalnosti, prvo je u *pubspec.yaml* datoteku potrebno dodati željene pakete, te ih importirati u datotekama u kojima ćemo ih koristiti.

```
dependencies:
  flutter:
    sdk: flutter
  tflite: ^1.0.4
  image_picker: ^0.6.2+3
  firebase_core: ^1.21.1
  cloud_firestore: ^3.4.6
```

Slika 4.11. Paketi potrebni za razvoj aplikacije

Kada pokrenemo aplikaciju, prvi zaslon je zaslon klase *Home*. Funkcija *initState* se prva pokreće kada se ova klasa pozove, u programskom kodu 4.8. moguće je vidjeti da ona poziva funkciju *loadModel* i tako učitava model koji će biti korišten za klasifikaciju slika.

```
import 'dart:io';
import 'package:flutter/material.dart';
import 'package:tflite/tflite.dart';
import 'package:image_picker/image_picker.dart';

class Home extends StatefulWidget {
  const Home({Key? key}) : super(key: key);
  @override
  _HomeState createState() => _HomeState();
}

class _HomeState extends State<Home> {
  bool _loading = true;
  late File _image;
  late List _output;
  final picker = ImagePicker();

  @override
  void initState() {
    super.initState();
    loadModel().then((value) {
      setState(() {});
    });
  }
```

Slika 4.12. Klasa *Home* i importiranje potrebnih paketa

U mapu *assets* unutar projekta potrebno je dodati dokument *labels.txt* koji sadrži oznake voća i povrća kojeg je moguće prepoznati i model u *.tflite* obliku. Nakon toga, kako bi se model učitao, samo je potrebno pozvati *loadModel* funkciju kojoj se kao parametri predaju model i oznake.

```
loadModel() async {
  await Tflite.loadModel(
    model: 'assets/My_TFlite_Model1.tflite',
    labels: 'assets/labels.txt',
  );
}
```

Slika 4.13. Učitavanje modela

Kako bi učitani model bilo moguće koristiti, potrebno je dohvatiti sliku s kamere ili iz galerije, za to je potreban *image_picker* paket koji je, poput *tflite* paketa, ranije dodan u *pubspec.yaml* datoteku.

```
pickImage() async {
  var image = await ImagePicker.pickImage(source: ImageSource.camera);

  setState(() {
    _image = File(image.path);
  });
  classifyImage(_image);
}

pickGalleryImage() async {
  var image = await ImagePicker.pickImage(source: ImageSource.gallery);

  setState(() {
    _image = File(image.path);
  });
  classifyImage(_image);
}
```

Slika 4.13. Funkcije za dohvaćanje slika s kamere ili iz galerije

Na slici 4.13. moguće je vidjeti da se dohvaćena fotografija predaje funkciji *classifyImage*, ona pokreće model nad slikom koja joj je predana kako bi ju klasificirala. Funkcija *classifyImage* prima sliku s kamere ili iz galerije i poziva funkciju *runModelOnImage* kojoj se predaje putanja slike, broj klasa, *threshold* koji označava prag klasifikacije, to znači da bi slika bila klasificirana kao određena vrsta, vrijednost vjerojatnosti te vrste mora biti iznad 0.5, a *imageMean* i *imageStd* služe kako bi vrijednost piksela bila između 0 i 1.

```
classifyImage(File image) async {
  var output = await Tflite.runModelOnImage(
    path: image.path,
    numResults: 48,
    threshold: 0.5,
    imageMean: 0,
    imageStd: 255,
  );
  setState(() {
    _output = output!;
    _loading = false;
  });
}
```

Slika 4.14. Funkcija koja pokreće model nad slikom

4.5.2. Izgled aplikacije

Prvi ekran nakon pokretanja aplikacije je zaslon na kojemu se nalaze tipka za otvaranje kamere i tipka za učitavanje slike iz galerije. Na dnu zaslona nalazi se traka za navigaciju pomoću koje se moguće prebaciti na drugi zaslon koji će sadržavati listu 48 vrsta voća i povrća koje aplikacija može prepoznati, te kratki opis i sliku za svaku vrstu.

Unutar mape *pages* nalaze se *home.dart* i *fruit_list.dart* datoteke između kojih će se moći navigirati, unutar *home.dart* datoteke nalazi se sav programski kod potreban za klasifikaciju slika i izgled zaslona za klasifikaciju, a unutar *fruit_list.dart* datoteke bit će implementirana baza podataka i izgled zaslona s listom voća.

U *main.dart* datoteci nalazi se programski kod potreban za stvaranje trake za navigaciju. U Flutteru, izgled aplikacije postiže se tako da se unutar *widgeta* slažu drugi *widgeti* kojima je onda moguće prilagođavati veličinu, boju i brojne druge atribute kako bi se postigao željeni izgled. Za početak se dodaje jedan *Scaffold*. *Scaffold* će se proširiti kako bi ispunio raspoloživi prostor, točnije, kako bi zauzeo cijeli zaslon uređaja, a služit će kao okvir unutar kojeg je onda moguće dodati druge *widgete* [31].

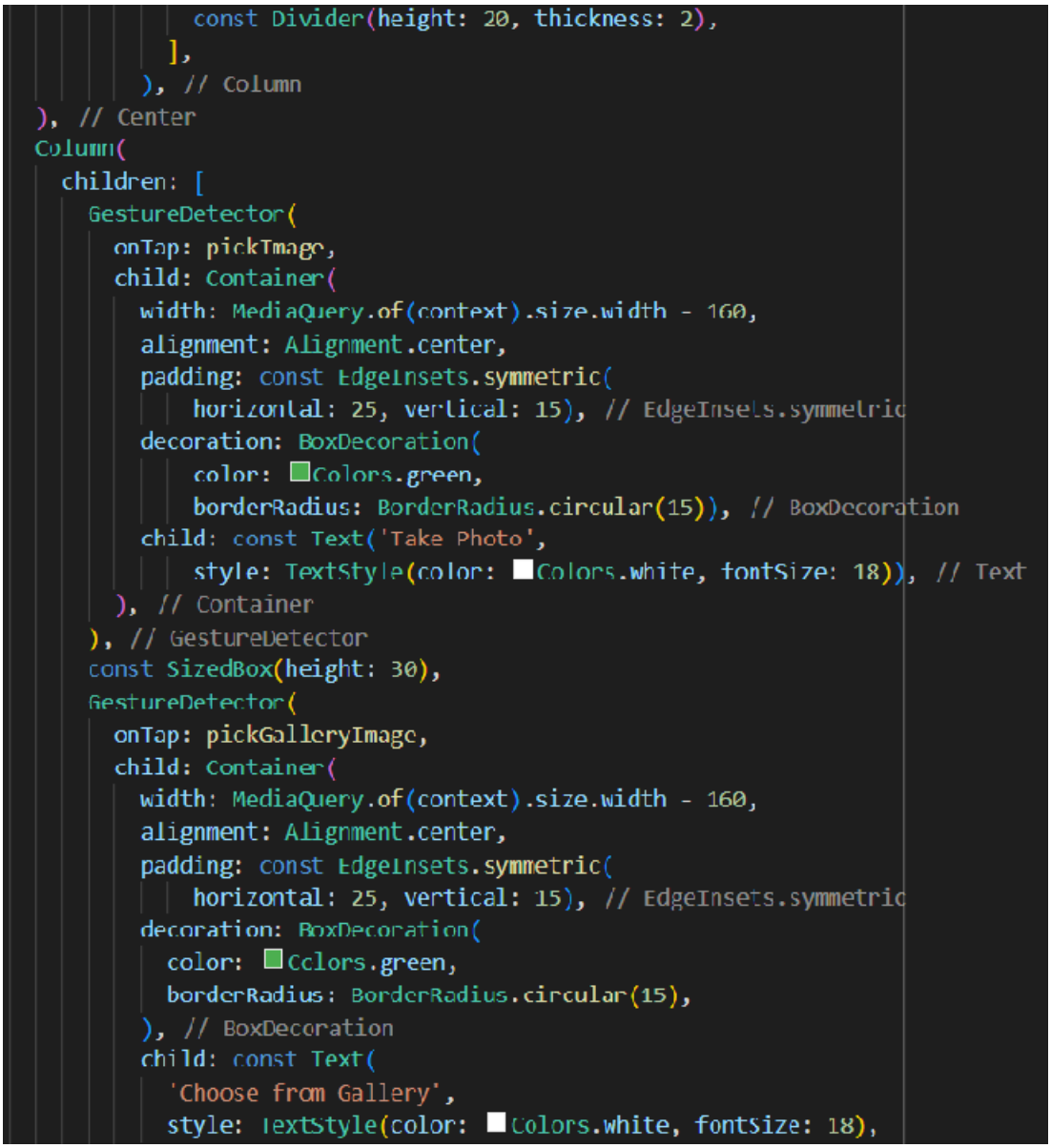
```
class _MyHomePageState extends State<MyHomePage> {
  int _selectedScreenIndex = 0;
  final List<Widget> _screens = [
    const Home(),
    const FruitList(),
  ];

  void _selectPage(int index) {
    setState(() {
      _selectedScreenIndex = index;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("FruitClassificationApp"),
        backgroundColor: const Color(0xFF2E7D32),
      ), // AppBar
      body: _screens[_selectedScreenIndex],
      bottomNavigationBar: BottomNavigationBar(
        backgroundColor: Colors.white,
        unselectedItemColor: Colors.black,
        selectedItemColor: Colors.green,
        currentIndex: _selectedScreenIndex,
        onTap: _selectPage,
        items: const [
          BottomNavigationBarItem(
              icon: Icon(Icons.image), label: 'Classification'), // BottomNavigationBarItem
          BottomNavigationBarItem(
              icon: Icon(Icons.list), label: 'List of fruits') // BottomNavigationBarItem
        ],
```

Slika 4.15. Programski kod za stvaranje izbornika i navigaciju između zaslona

Ovdje je unutar *Scaffolda* na vrhu dodan *appBar* koji će u ovom slučaju sadržavati naslov aplikacije, na dnu se nalazi već spomenuti *bottomNavigationBar* s dvije stavke, a izgled je lako moguće prilagoditi promjenom boje pozadine i ikona. Zasloni između kojih se navigira dodaju se u listu *widgeta*. *Body* je središnji dio *Scaffolda*, njemu se kao parametar postavlja već spomenuta lista zaslona, a trenutni zaslon mijenja se pomoću indeksa.

Na isti način nastaje i izgled zaslona za klasifikaciju. Unutar središnjeg dijela *Scaffolda* dodaje se spremnik, unutar spremnika ponovno je moguće kao djecu dodati nove spremnike, tekst, slike i druge *widgete*, neki od ovdje korištenih su, recimo, *SizedBox* i *Divider*.

```dart
class _HomeState extends State<Home> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        padding: const EdgeInsets.all(35),
        color: const Color(0xFFA5D6A7),
        child: Container(
          alignment: Alignment.center,
          padding: const EdgeInsets.all(30),
          decoration: BoxDecoration(
            color: Colors.green[300],
            borderRadius: BorderRadius.circular(30),
          ), // BoxDecoration
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Center(
                child: _loading == true
                    ? null
                    : Column(
                        children: [
                          SizedBox(
                            height: MediaQuery.of(context).size.width * 0.5,
                            width: MediaQuery.of(context).size.width * 0.5,
                            child: ClipRRect(
                              borderRadius: BorderRadius.circular(30),
                              child: Image.file(_image, fit: BoxFit.fill),
                            ), // ClipRRect
                          ), // SizedBox
                          const Divider(height: 20, thickness: 2),
                          Text(
                            '${_output[0]['label']}',
                            style: const TextStyle(
                                color: Colors.white,
                                fontSize: 20,
                                fontWeight: FontWeight.bold), // TextStyle
```



Slika 4.16. Programski kod kojim se postiže izgled zaslona za klasifikaciju

4.5.3. Implementacija baze podataka u aplikaciju

Osim što je u *pubsepc.yaml* datoteku potrebno dodati *cloud_firestore* i *firebase_core* pakete, te ih implementirati u datotekama u kojima će se koristiti, kako bi korištenje baze podataka bilo moguće, potrebno je napraviti još neke korake. Prvo, potrebno se pomoću Google računa prijaviti na Firebase službenoj stranici, nakon toga potrebno je stvoriti novi projekt i u njemu registrirati aplikaciju. Potrebno je još *.json* datoteku koju Firebase generira dodati u android mapu našeg projekta. Svi koraci koje je potrebno napraviti detaljno su objašnjeni u izvoru [32]. Nakon toga, može se popuniti baza podataka, svaka vrsta voća će biti prikazana u obliku jednog dokumenta. Izgled korištene baze moguće je vidjeti ranije na slici 4.1.

```
class _FruitListState extends State<FruitList> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: StreamBuilder(
          stream: FirebaseFirestore.instance.collection("fruits").snapshots(),
          builder: (context, AsyncSnapshot snapshot) {
            if (snapshot.connectionState == ConnectionState.waiting) {
              return const Center(
                child: CircularProgressIndicator(),
              ); // Center
            } else if (snapshot.hasError) {
              return const Center(
                child: Text("An error occurred."),
              ); // Center
            }
            return ListView.builder(
                itemCount: snapshot.data.docs.length,
                itemBuilder: (context, index) {
                  DocumentSnapshot fruit = snapshot.data.docs[index];
                  return ListTile(
                    leading: Image.network(fruit['Image']),
                    title: Text(fruit['Name']),
                    subtitle: Text(fruit['Nutrients']),
```

Slika 4.17. Uspostavljanje veze s bazom i prikaz dohvaćenih podataka

Ovdje je kao središnji dio *Scaffolda* postavljen *StreamBuilder,* to je *widget* koji se izgrađuje na temelju najnovije interakcije sa *Streamom* [33]. *Stream* pruža način primanja niza događaja. Svaki događaj je ili podatkovni događaj ili obavijest da nešto nije uspjelo [34]. *StreamBuilder* prima *Stream,* koji se dobije pomoću *snapshots* metode, i *builder*, koji vraća različiti *widget* u ovisnosti jesu li podaci dohvaćeni, veza se tek uspostavlja ili je došlo do pogreške.

Ako su podaci iz baze uspješno dohvaćeni, bit će prikazani u obliku liste, a to se postiže pomoću *ListViewa.* Broj elemenata jednak je broju dokumenata u bazi, dohvaćeni podaci spremaju se u varijablu *fruit* tipa *DocumentSnapshot,* te se svaki dokument iz baze prikazuje kao jedan redak unutar liste.

Kako bi se podaci iz baze dohvatili i prikazali u listi, potrebno je još u *main.dart* datoteci u glavnoj funkciji inicijalizirati bazu podataka.

```dart
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(const MyApp());
}
```

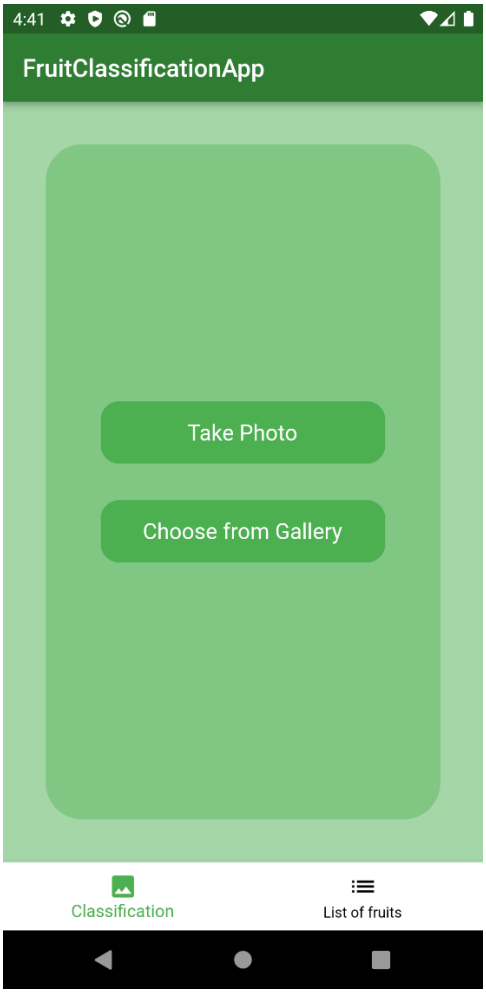
Slika 4.18. Inicijalizacija baze podataka

5. PRIKAZ RADA APLIKACIJE, REZULTATI I ANALIZA

U ovom poglavlju prikazano je korištenje aplikacije i njezine mogućnosti. Također, će biti prikazano testiranje aplikacije i pojašnjeni rezultati.

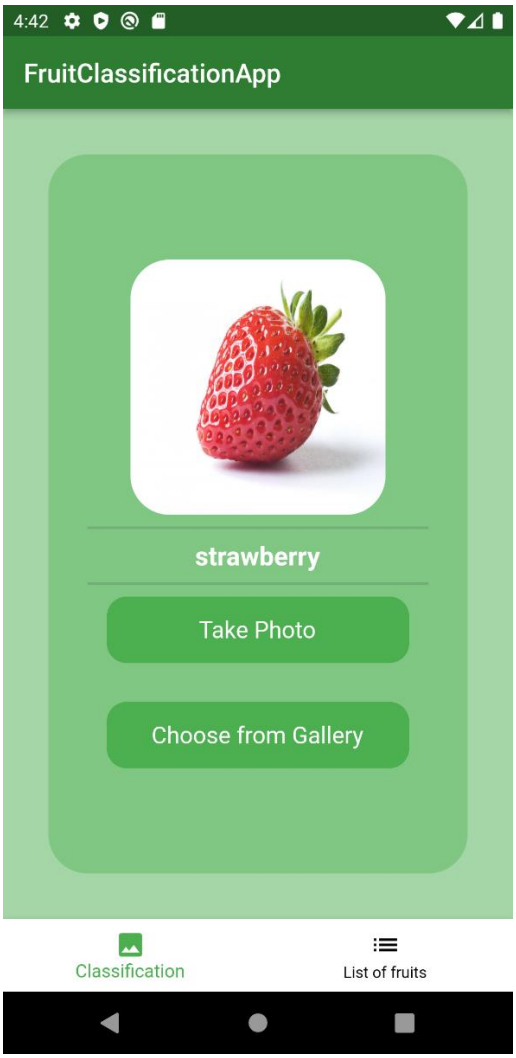
5.1. Prikaz korištenja aplikacije

Aplikacije je vrlo jednostavna. Na početnom zaslonu nalaze se tipka za otvaranje kamere, te tipka za odabir slike iz galerije. Na vrhu se nalazi ime aplikacije, a na dnu traka za navigaciju.



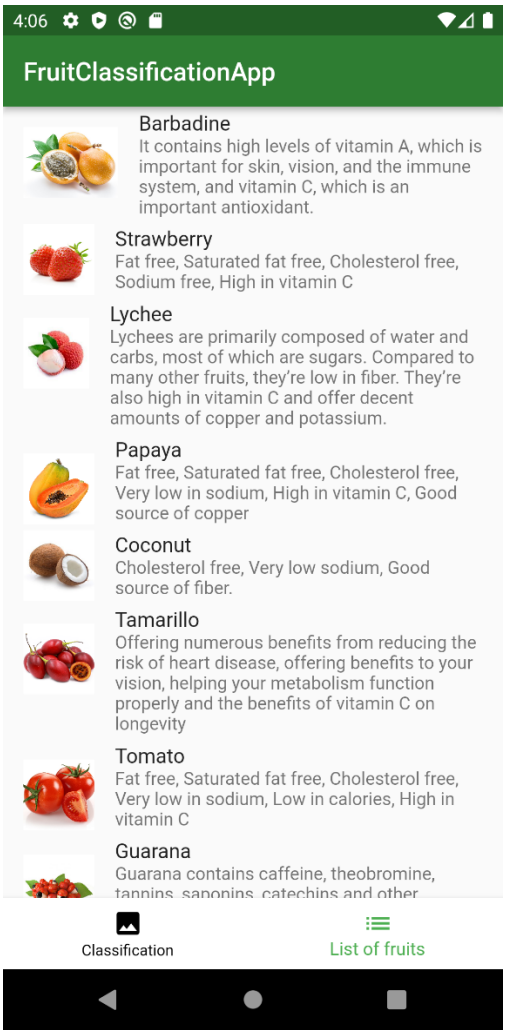
Slika 5.1. Početni zaslon

Ukoliko se koristi prvi put, aplikacija će tražiti dopuštenje za pristup medijima na uređaju. Nakon što je slika uslikana kamerom ili odabrana iz galerije, aplikacija bi trebala vratiti rezultat klasifikacije i prikazati sliku iznad *Take Photo* i *Choose from Gallery* tipki.



Slika 5.2. Rezultat klasifikacije

Na slici 5.2. može se vidjeti da je aplikacija u ovome slučaju vratila točan rezultat. Prilikom pritiska na *List of fruits* na navigacijskoj traci, aplikacija će iz zaslona za klasifikaciju otići na zaslon koji sadrži bazu svih vrsta voća i povrća koje bi aplikacija trebala biti u mogućnosti prepoznati i kratki opis, te sliku svake vrste.

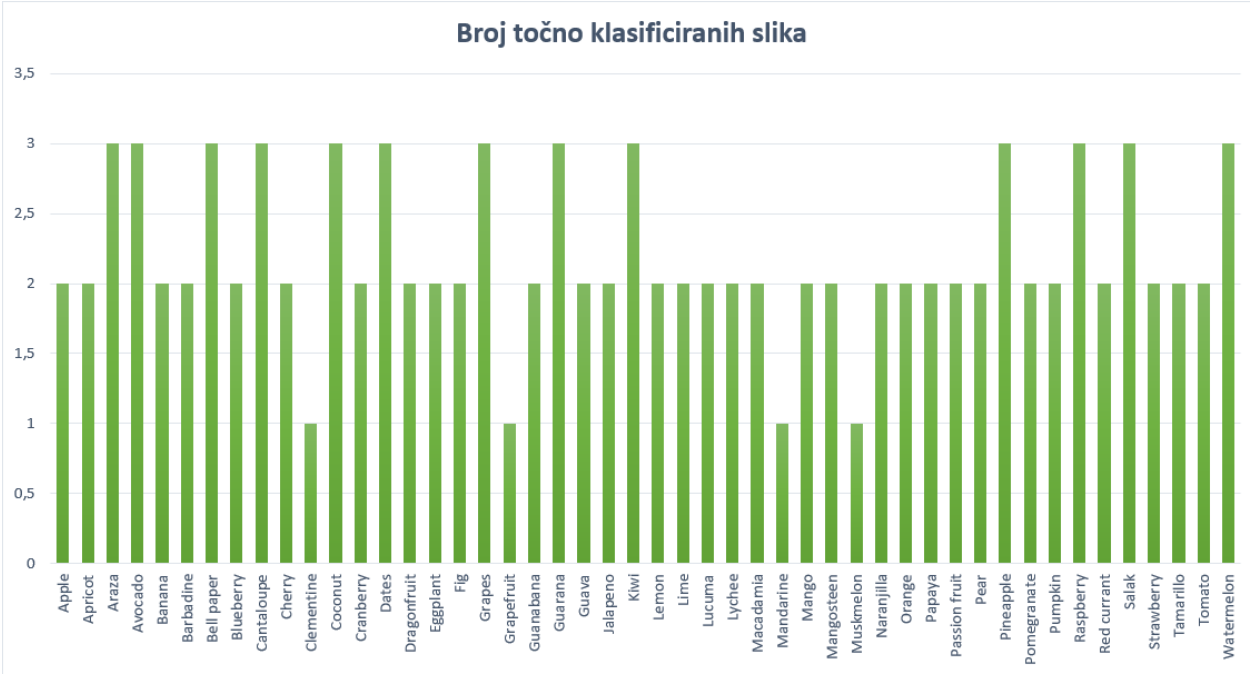


Slika 5.3. Zaslon s listom voća i povrća

5.2. Ispitivanje aplikacije i analiza rezultata

Kako bi se ispitala aplikacija i korišteni model strojnog učenja, za svaku vrstu odabrane su tri različite slike s interneta. Cilj je bio da su slike što različitije, da se razlikuju po pozadini, broju plodova ili kutu slikanja.

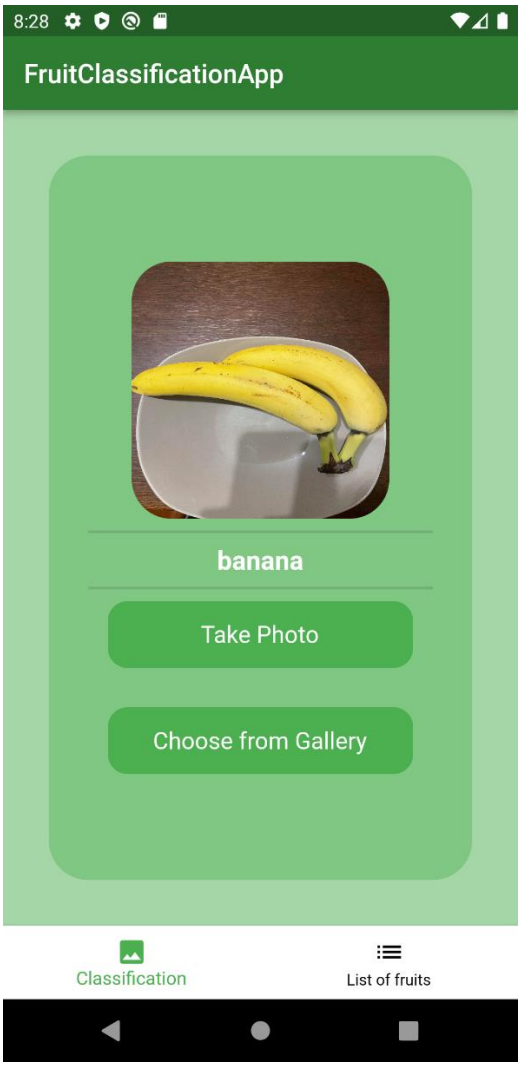
Za većinu vrsta, aplikacija je uspješno prepoznala o kojoj se vrsti radi dva od tri puta, a u postotku je to 66.67%. Nešto lošiji rezultat postiže se kod mandarine, grejpa i klementine, aplikacija je te vrste uspješno prepoznala samo jednom od tri puta, ali to je i očekivano jer su te vrste međusobno dosta slične. Aplikacija je za 13 od 48 vrsta uspješno prepoznala o čemu se radi sva tri puta.



Slika 5.4. Rezultati testiranja aplikacije

Sveukupno su za testiranje korištene 144 slike, a točno su klasificirane 103 slike, to znači da je točnost aplikacije kod ovog testiranja, u postotku, iznosila 71.53%. Rezultat dosta ovisi o slikama koje se koriste, s drugim slikama rezultat bi vjerojatno bio nešto drugačiji, ali može se vidjeti da aplikacija u dovoljnoj mjeri uspješno prepoznaje voće i povrće sa fotografije.

Kako bi se ispitalo radi li aplikacija uspješno i sa vlastitim slikama, zbog korištenja emulatora, slika je uslikana pametnim telefonom, poslana i preuzeta na računalo, a zatim učitana iz galerije emulatora.



Slika 5.5. Rezultat klasifikacije vlastite fotografije

Na slici 5.5. može se vidjeti da je aplikacija uspješno prepoznala da se radi o banani.

5.3. Analiza zahtjeva

Funkcionalni, kao i nefunkcionalni, zahtjevi aplikacije navedeni su u poglavlju 3.1.

Klasifikacija se uspješno provodi i na slikama učitanima iz galerije, ali i na onima koje su snimljene kamerom, slika i rezultat klasifikacije prikazuju se na zaslonu. Hoće li slika biti ispravno klasificirana ovisi o implementiranom modelu, kao i o samoj fotografiji, ne o aplikaciji. Podaci o voću i povrću su, također, uspješno dohvaćeni i prikazani korisniku, stoga je moguće zaključiti da su funkcionalni zahtjevi aplikacije ispunjeni.

Kada je fotografija dohvaćena, aplikaciji je potrebna otprilike jedna sekunda da prikaže rezultat na zaslonu. Aplikacija je jako jednostavna za korištenje, nema previše funkcionalnosti i svakome bi trebalo odmah biti jasno kako ju koristiti. Aplikacija se prilagođava veličini zaslona, a iako je

nekim slikama potrebno oko 6 sekundi da se pojave u listi, to nije toliko dugačak period koji bi pokvario korisničko iskustvo. Dakle, nefunkcionalni zahtjevi aplikacije su također ispunjeni.

6. ZAKLJUČAK

Cilj rada bio je razviti mobilnu aplikaciju koja će prepoznavati egzotične vrste voća i povrća koristeći klasifikacijske postupke strojnog učenja. Prvo su, u teorijskom dijelu rada, opisane metode strojnog učenja kojima je moguće ostvariti klasifikaciju slika, te su prikazana dva već postojeća rješenja aplikacije za prepoznavanje voća i povrća sa slike. Nakon toga, opisane su željene funkcionalnosti aplikacije i prikazan je blok dijagram rada aplikacije.

Za razvoj modela strojnog učenja korištena je MobileNetV2 arhitektura neuronske mreže i preneseno učenje. Učenje neuronske mreže ostvareno je uz pomoć TensorFlow i Keras biblioteka u Jupyter Notebook-u, a korišteni programski jezik bio je Python. Mobilna aplikacija razvijena je koristeći Visual Studio Code i Flutter, a osim modela strojnog učenja, ima implementiranu i Cloud Firestore bazu podataka. Razvoj aplikacija u Flutteru olakšavaju brojni paketi i bogata dokumentacija. Sve tehnologije koje su korištene u razvoju aplikacije detaljno su opisane u radu.

Rezultati aplikacije su zadovoljavajući, ali ima prostora za napredak. Model je moguće unaprijediti, kako bi rezultati bili još bolji, koristeći neku drugu arhitekturu neuronske mreže ili neki drugi skup podataka koji će sadržavati više vrsta voća i povrća ili više slika za svaku vrstu. Također, moguće je dodati još funkcionalnosti, recimo prijavu korisnika, u aplikaciju.

LITERATURA

[1] D. Michie, D.J. Spiegelhalter, C.C. Taylor, *Machine Learning, Neural and Statistical Classification*, 1994.

[2] P. Kamavisdar, S. Saluja, S. Agrawal, *A Survey on Image Classification Approaches and Techniques*, International Journal of Advanced Research in Computer and Communication Engineering Vol. 2, str. 1005.-1009., Siječanj 2013

[3] R. Dastres, M. Soori, *Artificial Neural Network Systems*, International Journal of Imaging and Robotics, No.2, Vol. 21, str. 14.-25., Ožujak 2021.

[4] T. Kramberger, B. Nožica, I. Dodig, D. Cafuta, *Overview of Artificial Neural Network Technologies*, Politehnika i dizajn, Vol. 7, No. 1, str. 25.-32., 2019.

[5] *Artificial Neural Network Tutorial*, javatpoint.com, dostupno na: https://www.javatpoint.com/artificial-neural-network [Rujan 2022.]

[6] V. Yathish, *Loss Functions and Their Use In Neural Networks*, Towards Data Science, 2022., dostupno na: https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9 [Rujan 2022.]

[7] N. Donges, *Gradient Descent: An Introduction to 1 of Machine Learning's Most Popular Algorithms*, builtin.com, 2021., dostupno na: https://builtin.com/data-science/gradient-descent [Rujan 2022.]

[8] M.M.Mijwil, *Artificial Neural Networks Advantages and Disadvantages*, LinkedIn, 2018., dostupno na: https://www.linkedin.com/pulse/artificial-neural-networks-advantages-disadvantages-maad-m-mijwel [Rujan 2022.]

[9] V. E. Lee, L. Liu, R. Jin, *Decision Trees: Theory and Algorithms*, 2014.

[10] *Introduction to Support Vector Machines*, DataFlair, dostupno na: https://data-flair.training/blogs/svm-support-vector-machine-tutorial/ [Lipanj 2022.]

[11] *Frutolo*, dostupno na: https://www.frutolo.com/ [Lipanj 2022.]

[12] Google Play, *Fruits and Vegetables Identification & Scanner*, dostupno na: https://play.google.com/store/apps/details?id=e.fruit.natureai&hl=hr&gl=US [Lipanj 2022.]

[13] H. Atha, *Understanding Functional and Non.Functional Requirements in App Development*, moveoapps.com, 2021., dostupno na: https://www.moveoapps.com/blog/functional-and-non-functional-requirements-in-app-development/ [Rujan 2022.]

[14] V. Ng, *Non-Functional Requirement of the Mobile Development system*, Medium, 2019., dostupno na: https://medium.com/@vishwasng/non-functional-requirement-of-the-mobile-development-system-e0ed98f2a872 [11.09.2022.]

[15] Wikipedia, Python, dostupno na: https://en.wikipedia.org/wiki/Python_(programming_language) [Lipanj 2022.]

[16] Wikipedia, Jupyter Notebook, dostupno na: https://en.wikipedia.org/wiki/Project_Jupyter [Lipanj 2022.]

[17] S.Yegulalp, *What is TensorFlow? The machine learning library explained*, InfoWorld, 2022., dostupno na: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html [Lipanj 2022.]

[18] Keras, About Keras, dostupno na: https://keras.io/about/ [Lipanj 2022.]

[19] Wikipedia, Flutter, dostupno na: https://en.wikipedia.org/wiki/Flutter_(software) [Rujan 2022.]

[20] Flutter, dostupno na: https://flutter.dev [Rujan 2022.]

[21] U. Urathal Alias Sri Swathiga, P. Vinodhini, V. Sasikala, *An Interpretation of Dart Programming Language*, DRSR Journal, Vol. 11, No. 3, str. 144.-149., Listopad 2021.

[22] Microsoft, Visual Studio Code, dostupno na: https://code.visualstudio.com/ [Rujan 2022.]

[23] StackOverflow, Developer Survey, dostupno na: https://survey.stackoverflow.co/2022/#section-most-popular-technologies-integrated-development-environment [Rujan 2022.]

[24] Google, Firebase: Firestore, dostupno na: https://firebase.google.com/docs/firestore [Rujan 2022.]

[25] M. Minut, *Fruits-262 dataset: A dataset containing a vast majority of the popular and known fruits*, Kaggle, 2021., dostupno na: https://www.kaggle.com/datasets/aelchimminut/fruits262 [Rujan 2022.]

[26] A. Bhandari, *Image Augmentation on the fly using Keras ImageDataGenerator!*, Notebook Community, Analytics Vidhya, 2020., dostupno na: https://www.analyticsvidhya.com/blog/2020/08/image-augmentation-on-the-fly-using-keras-imagedatagenerator/ [Rujan 2022.]

[27] Keras, Model training APIs, dostupno na: https://keras.io/api/models/model_training_apis/ Rujan 2022.]

[28] Gao, Y., and Mosalam, K., *Deep Transfer Learning for Image-Based Structural Damage Recognition. Computer-Aided Civil and Infrastructure Engineering*, 2018.

[29] J. Brownlee, *Transfer Learning in Keras with Computer Vision Models*, Machine Learning Mastery, 2019., dostupno na: https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/ [Rujan 2022.]

[30] TensorFlow, Transfer learning, dostupno na:. https://www.tensorflow.org/tutorials/images/transfer_learning [Rujan 2022.]

[31] Flutter, Scaffold class, dostupno na: https://api.flutter.dev/flutter/material/Scaffold-class.html [Rujan 2022.]

[32] Google, Firebase, dostupno na: https://firebase.google.com/ [Rujan 2022.]

[33] Flutter, StreamBuilder class, dostupno na: https://api.flutter.dev/flutter/widgets/StreamBuilder-class.html [Rujan 2022.]

[34] Flutter, Stream class, dostupno na: https://api.flutter.dev/flutter/dart-async/Stream-class.html [Rujan 2022.]

SAŽETAK

U teorijskom dijelu rada opisana je problematika prepoznavanja objekata sa slike, te metode strojnog učenja kojima se klasifikacija slika može ostvariti. Prikazana su i neka već postojeća rješenja aplikacija za prepoznavanje voća i povrća. Za razvoj modela strojnog učenja korištena je MobileNetV2 neuronska mreža koja je, koristeći preneseno učenje, prilagođena odabranom skupu podataka. Za razvoj modela korišten je Jupyter Notebook s TensorFlow bibliotekom i programski jezik Python. Model je zatim implementiran u mobilnu aplikaciju koja je razvijena koristeći Visual Studio Code, Flutter i Dart programski jezik. Aplikacija ima implementiranu i bazu podataka koja sadrži sve vrste voća i povrća koje bi trebalo biti moguće prepoznati. Baza podataka stvorena je koristeći Firebase Cloud Firestore. Na kraju rada, prikazani su rezultati, te izgled i korištenje aplikacije.

Ključne riječi: Flutter, klasifikacija voća, neuronska mreža, strojno učenje, TensorFlow.

ABSTRACT

Title: ANDROID MOBILE APPLICATION FOR RECOGNITION OF EXOTIC FRUITS AND VEGETABLES USING MACHINE LEARNING

In the theoretical part of the thesis, the problem of recognizing objects from images is described, as well as machine learning methods that can be used to classify images. Some already existing solutions for fruit and vegetable recognition applications are also shown. MobileNetV2 neural network and transfer learning were used for developing machine learning model. Jupyter Notebook, TensorFlow library and Python programming language were used to develop the model. The model was then implemented into a mobile application that was developed using Visual Studio Code, Flutter and the Dart programming language. The application, also, has implemented database which contains all types of fruits and vegetables that application should be able to recognize. The database was created using Firebase Cloud Firestore. At the end of the work, the results are presented, as well as the appearance and usage of the application.

Key words: Flutter, fruit classification, neural network, machine learning, Tensorflow.

PRILOZI

Prilog 1. Dokument završnog rada

Prilog 2. Pdf završnog rada

Prilog 3. Programski kod strojnog učenja

Prilog 4. Projekt mobilne aplikacije