Android mobilna aplikacija za prepoznavanje pasmina pasa postupcima strojnog učenja

Mihelčić, Dario

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

Sveučilišni preddiplomski studij

ANDROID MOBILNA APLIKACIJA ZA PREPOZNAVANJE PASMINA PASA POSTUPCIMA STROJNOG UČENJA

Završni rad

Dario Mihelčić

Osijek, 2019.

SADRŽAJ

1. UVOD

Problem kod programiranih alata je što, jednom kad se napišu i instaliraju, ostaju nepromijenjeni, a dani problem rješavat će na dani način bez mogućnosti samostalne prilagodbe. No, mnogi se problemi mijenjaju ovisno o vremenu i okolini. Alati bazirani na strojnom učenju daju rješenje takvih problema. Ovi alati mogu naučiti uzorke i pravila iz danog skupa podataka te to znanje primijeniti kako bi se samostalno prilagodili na nove skupove podataka. Jedna grana primjene strojnog učenja je računalni vid koja se bavi razvojem alata koji bi računalima dali sposobnost interpretacije vizualnog svijeta. U ovom završnom radu obrađena je tehnika strojnog učenja, odnosno neuronskih mreža kao rješenje problema računalnog prepoznavanja slika pasa.

Cilj rada je izraditi programski alat kojeg će se naučiti prepoznati pasminu pasa s danih slika. Programski alat će naučiti prepoznati pasmine pasa na temelju skupa podataka te će moći samostalno prepoznati pasmine pasa na novim slikama. Takvo programsko rješenje implementirat će se u Android aplikaciju kako bi se izradio prijenosni, samostalni alat koji će korisnicima uvijek biti pri ruci. Aplikacija će korisniku dati ime pasmine, sliku te opis prepoznate pasmine.

U drugom poglavlju dan je osnovi uvod u problematiku rada te opis slučajeva koji se rješavaju primjenom odabranih postupaka, neuronskih mreža. U trećem poglavlju opisani su alati i tehnike korištene pri izgradnji rješenja problema. U četvrtom poglavlju obuhvaćeno je programsko rješenje alata za prepoznavanje i aplikacije. Opisani su postupci obrade podataka na kojima program uči prepoznavati pasmine, proces treniranja programa te implementacija programa za prepoznavanje u Android aplikaciju. Također, opisani su i glavni dijelovi koda aplikacije. U petom poglavlju opisan je postupak korištenja aplikacije, analiza rada i testiranje aplikacije na novim ulaznim podatcima.

1.1. Zadatak završnog rada

U teorijskom dijelu rada treba objasniti postupke strojnog učenja pogodne za prepoznavanje pasmina pasa sa slika te analizirati programske tehnologije, alate, okvire i programsku arhitekturu potrebnu za razvoj mobilne aplikacije i implementiranje postupaka strojnog učenja. Također, treba razraditi model i programski ostvariti Android mobilnu aplikaciju koja uključuje izabrani postupak strojnog učenja, dohvaćanje i pripremu podataka, analizu, prikaz i pohranu rezultata, te omogućuje poboljšanje postupka učenja na temelju mogućih dodatnih ulaza. Mobilnu aplikaciju i postupak strojnog učenja treba ispitati i analizirati za odgovarajući skup ulaznih podataka.

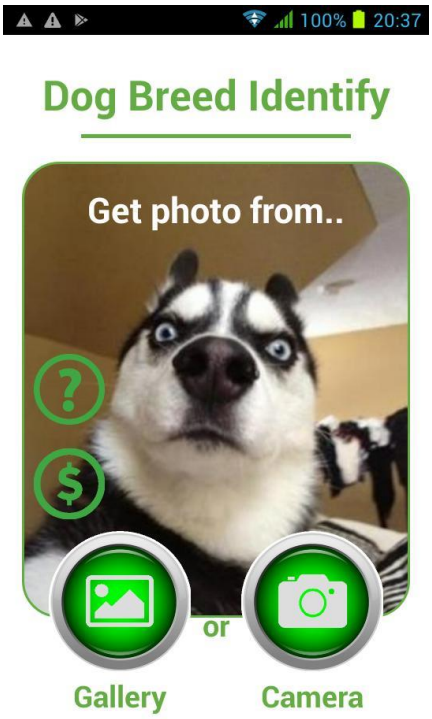
2. PREPOZNAVANJE SLIKA

Kako je napisano u [1], alati strojnog učenja mijenjaju ponašanje u skladu s promjenom okoline s kojom su povezane. Najčešće primjene strojnog učenja uključuju programe koji dekodiraju rukom pisani tekst različitih korisnika, programe koji prepoznaju neželjenu e-poštu (engl. *spam*) i programe koji prepoznaju govor. Jedna od primjena strojnog učenja rješava problem prepoznavanja uzoraka ili predmeta sa slika. Ova tehnika naziva se klasifikacija (engl. *classification).* Na primjer, detekcija *spam* poruka može se smatrati problemom klasifikacije. Klasifikator će na temelju podataka treninga naučiti kako se dana ulazna varijabla povezuje sa svakom od izlaznih klasa. U ovom slučaju, potrebna je baza podataka koja sadrži poznate *spam* ili poruke koje nisu *spam*. Ova tehnika strojnog učenja spada u nadgledano učenje, jer se uz ulazne podatke daje i odgovarajuća izlazna klasa.

U ovom poglavlju bit će opisani postupci rješavanja problema klasifikacije psećih pasmina sa slika. U prvom poglavlju dani su primjeri postojećih rješenja, zatim drugo poglavlje daje uvid u moguće postupke koje rješavaju problem klasifikacije slika te odabrani postupak koji se koristi u ovom radu. U zadnjem poglavlju su opisani različiti slučajevi u kojima se koristi odabrani postupak.

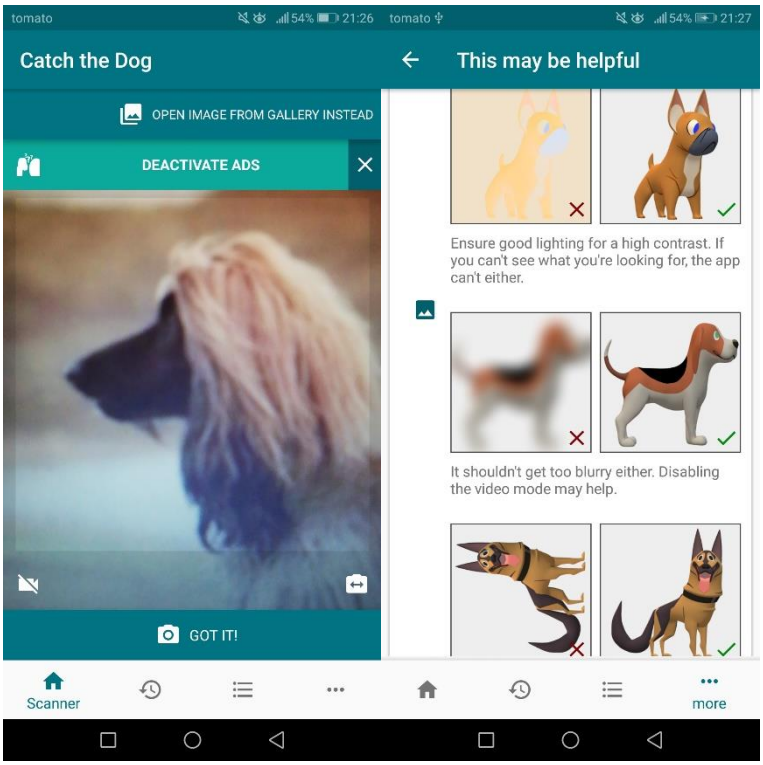
2.1. Postojeća rješenja

Platforma *Google Play* nudi nekoliko aplikacija koje se mogu koristiti za prepoznavanje pasmina pasa. *Dog Breed Auto Identify Photo* aplikacija (slika 2.1) prepoznaje pasminu pasa sa slike kamere ili slike iz galerije uređaja. Aplikacija koristi neuronske mreže kako bi pretpostavila pasminu psa na danoj slici. Ova aplikacija sposobna je prepoznati 48 pasmina pasa s postotkom točnosti od 84%. Aplikaciji je potreban pristup Internetu kako bi izvelo prepoznavanje. Korisnici aplikacije su je ocijenili ocjenom 3.2. Brojni korisnici tvrde da aplikacija ne funkcionira na obećan način te da brojne reklame uzrokuju loše iskustvo za korisnike.



Slika 2.1. Izgled aplikacije *Dog Breed Auto Identify Photo.*

Dog Scanner - #1 Dog Breed Identification je još jedna aplikacija za prepoznavanje pasmina pasa sa slike (slika 2.2). Aplikaciju je razvila tvrtka *Siwalu Software* koja je globalni voditelj u području prepoznavanja pasmina pasa i mačaka. Ova tvrtka razvija programe koji koriste umjetnu inteligenciju i strojno učenje. Aplikacija prepoznaje sve pasmine pasa s visokim postotkom točnosti te daje dodatne informacije o svakoj pasmini. Aplikacija prepoznaje pasmine sa slika uhvaćenih kamerom ili sa slika spremljenih na mobilni uređaj. U ovoj se aplikaciji također koriste neuronske mreže za prepoznavanje pasmina. Aplikacija ima ocjenu 4.3 na *Google Play* platformi. Nedostatak ove aplikacije je sporo generiranje rezultata i potreba za internetskom vezom.

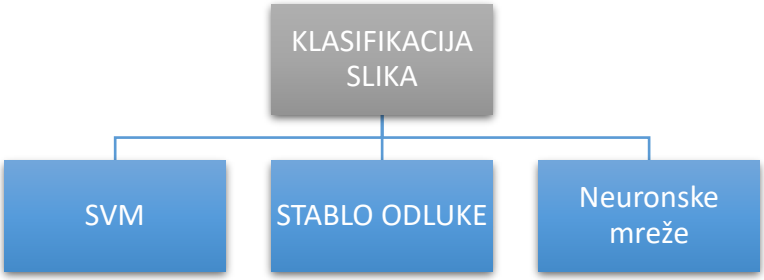


Slika 2.2. Izgled aplikacije *Dog Scanner*.

Aplikacije su dostupne za preuzimanje na *Play Store-u* no njihov izvorni kod nije dostupan te nisu poznati točni postupci korišteni u realizaciji programa za prepoznavanje. Prva aplikacija je potpuno besplatna no prepuna je reklama a prepoznavanje nije precizno. Druga aplikacija je uglavnom besplatna, no za dodatne opcije u aplikaciji je potrebno izdvojiti određeni iznos novca. Iako je točnost visoka i zastupljen je veliki broj pasmina, prepoznavanje je relativno sporo u odnosu na današnje sposobnosti mobilnih uređaja. U nastavku rada bit će istraženi postupci za prepoznavanje objekata sa slika, načini na koji se ti postupci realiziraju, te mogućnosti unaprjeđenja prepoznavanja. Također, bit će opisan proces izrade aplikacije koja će u stvarnom vremenu prikazivati rezultate prepoznavanja bez upotrebe Interneta. Ovakav oblik aplikacije će korisniku omogućiti brži i jednostavniji način dolazaka do rezultata, bez dugotrajnog čekanja za rezultate i bez smetnji reklama.

2.2. Pregled postupaka prepoznavanja slika

Glavna ideja iza računalnog prepoznavanja slika je rastaviti sliku na polja od interesa i tim poljima odrediti klasu. Ovaj se postupak još zove klasifikacija slika. Prema izvoru [2], najvažnije metode klasifikacije slika su strojevi potpornih vektora (engl. *Support Vector Machine,* SVM*),* stablo odluke i neuronske mreže. Na slici 2.3 može se vidjeti podjela metoda klasifikacije. U ovom radu razmatraju se nadzirane metode učenja.



Slika 2.3. Podjela tehnika strojnog učenja za klasifikaciju slika.

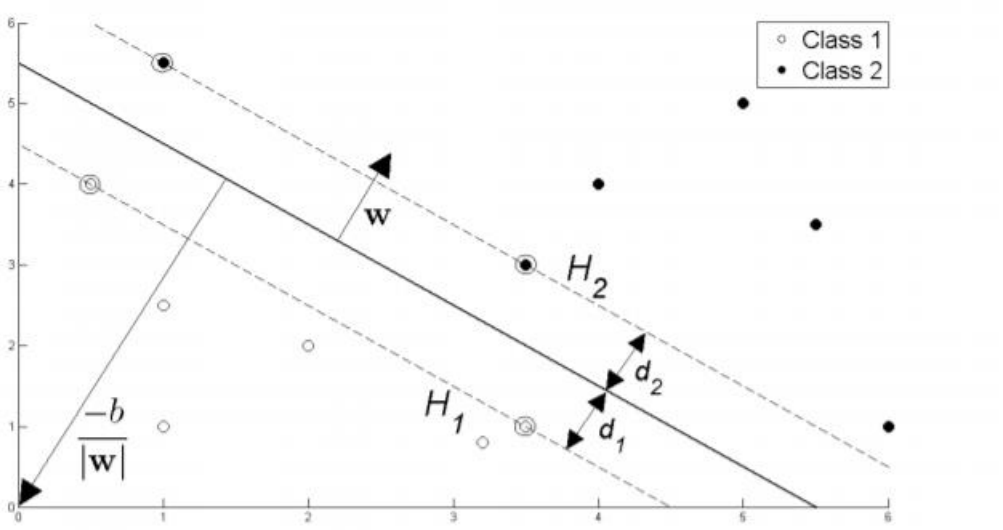
2.2.1. SVM

U radu [3] opisane su osnove SVM metode. Metoda je objašnjena na primjeru binarne klasifikacije koja se može linearno podijeliti. U primjeru se nalazi L točaka trening podataka (podatci mapirani u dvodimenzionalnom prostoru). Svaki ulazni podatak x_i ima D broj atributa (odnosno, ima dimenzionalnst D) i u jednoj je od dvije klase $y_i = -1\ or\ +1$, odnosno trening podatci definirani su kao:

$$\{\boldsymbol{x}_i,\ y_i\} \quad takvi\ da \quad i = 1 \dots L, y_i \in \{-1, 1\}, \boldsymbol{x} = \mathcal{R}^D \tag{2-1}$$

Pretpostavlja se da se podatci mogu linearno podijeliti, odnosno da se može nacrtati linija na grafu x_1, x_2 takvu da razdjeljuje dvije klase kada je D = 2 i da se može nacrtati hiper-ravnina na grafu $x_1, x_2 \dots x_D$ za D > 2.

Hiper-ravnina se može opisati jednadžbom $\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$ gdje je w normala hiper-ravnine, a $\frac{b}{||\boldsymbol{w}||}$ je okomita udaljenost od hiper-ravnine do ishodišta. Cilj SVM metode je orijentirati spomenute hiper-ravnine tako da se nalazi najdalje moguće od najbližih pripadnika obje klase. Primjer hiper-ravnine koja razdvaja podatke u dvodimenzionalnom prostoru prikazana je na slici 2.4.



Slika 1.4. Hiper-ravnina kroz dvije linerno razdvojive klase (izvor [4]).

Prema slici 2.4, korištenje SVM metode svodi se na računanje varijabli w i b takvi da se trening podatci mogu opisati kao:

$$\boldsymbol{x}_i \cdot \boldsymbol{w} + b \geq +1 \quad za\ y_i = +1 \tag{2-2}$$

$$\boldsymbol{x}_i \cdot \boldsymbol{w} + b \geq -1 \quad za\ y_i = -1 \tag{2-3}$$

Ove jednadžbe mogu se svesti na jednu, te se konačno dobiva:

$$y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 \geq 0\ \forall\ i \tag{2-4}$$

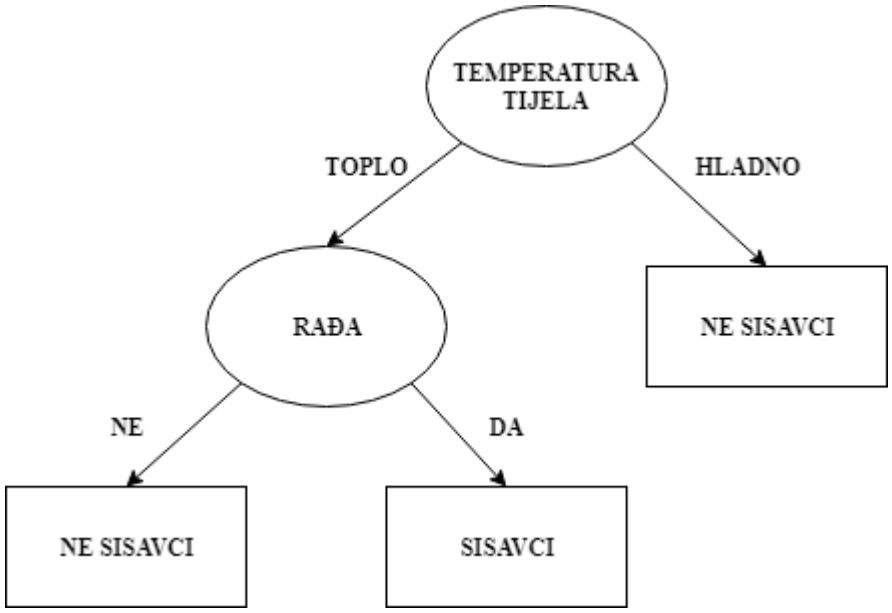
U slučajevima kada je broj svojstava podataka veći od dva, SVM metoda pokušava pronaći najbolju hiper-ravninu u višedimenzionalnom prostoru u kojem su smješteni svi podatci koja što bolje razdjeljuje podatke. Dimenzije prostora i hiper-ravnine određene su dimenzijama značajki podataka. SVM metoda ima odličnu sposobnost generalizacije i ne pokazuje znakove pretjeranog učenja na podatcima, no struktura ovog postupka je složena i time teška za razumjeti [5].

2.2.2. Stablo odluke

Izvor [4] daje uvid u rad metode stabla odluke. Dan je primjer klasificiranja hipotetske novo otkrivene vrste kralježnjaka te se postavlja pitanje kako odrediti je li nova vrsta sisavac ili ne. Jedna metoda je postavljanje uzastopnih pitanja o karakteristikama vrste. Prvo pitanje može pitati je li vrsta hladnokrvna ili toplokrvna. Ako je hladnokrvna, sa sigurnošću se može reći da nije sisavac. U suprotnom može biti ili ptica ili sisavac. U posljednjem slučaju potrebno je pitati dodatno pitanje, rađaju li ženke vrste potomke? Ako rađaju, sigurno su sisavci.

Ovaj primjer prikazuje kako niz pažljivo postavljenih pitanja o karakteristikama testiranog skupa rješava problem klasifikacije. Svaki puta kada se dobije odgovor, postavlja se novo pitanje

dok se ne dođe do konačne klase. Ovaj niz pitanja može se organizirati u strukturu stabla, hijerarhijsku strukturu koja se sastoji od čvorova i listova. Primjer stabla prikazan je na slici 2.5.



Slika 2.5. Stablo odluke za problem klasifikacije sisavaca (izvor [5]).

Stablo odluke računa pripadajuće klase ulaznih podataka tako da razgrađuje skup podataka u sve manje i manje podskupove dok u isto vrijeme gradi stablo odluke. Konačni rezultat je stablo sa čvorovima odluke i listovima odluke. Čvor odluke ima jednu ili više grana. Listovi odluke predstavljaju klasu. Prednost ove metode je to što nisu potrebne pretpostavke o distribuciji podataka i brzo se računaju. Primjer korištenja metode stabla odluka za klasifikaciju može se vidjeti u primjeru [6].

2.2.3. Neuronske mreže

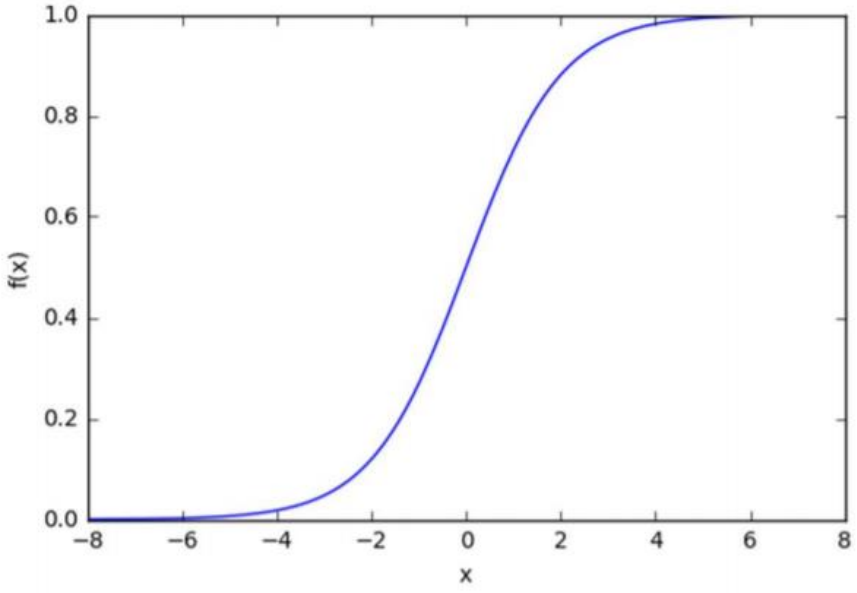
Osnovni opis neuronskih mreža prikazan je u [7]. Umjetne neuronske mreže su tip umjetne inteligencije koje imitiraju funkciju čovjekovog mozga. Neuronska mreža u čovjekovom mozgu je složen sustav međusobno povezanih neurona, gdje izlaz jednog neurona može biti ulaz u tisuće drugih neurona. Učenje se odvija konstantnim ponavljanjem aktivacije određenih neuronskih veza koje jača vezu tih neurona. Učenje uključuje i povratnu vezu, kada se postigne željeni rezultat, jača se neuronska veza koja ga je proizvela. Umjetne neuronske mreže (engl. *artificial neural network, odnosno skraćeno ANN)* pokušavaju pojednostaviti i imitirati taj proces.

Rad biološkog neurona je simuliran u ANN korištenjem aktivacijskih funkcija. U problemima klasifikacije aktivacijska funkcija mora imati karakteristike prekidača. Drugim riječima, kada vrijednost ulaza dosegne određenu vrijednost, izlaz bi se trebao promijeniti na primjer iz 0 u 1, od

-1 do 1 ili od 0 do >0. Ovo simulira aktivaciju biološkog neurona. Jedna od aktivacijskih funkcija koje se često koriste je sigmoid funkcija:

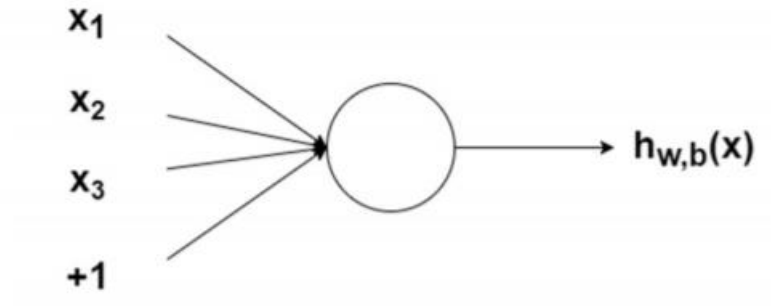
$$f(x) = \frac{1}{1+e^{-x}} \tag{2-5}$$

Kao što se vidi na slici 2.6, funkcija se aktivira ako se pomakne iz 0 u 1 kada je ulaz *x* veći od određene vrijednosti. Funkcija ima postupan prijelaz iz 0 u 1 što znači da izlaz nema skok iz jedne vrijednosti u drugu. Zbog toga postoji derivacija funkcije što je važno za algoritam treniranja ANN.



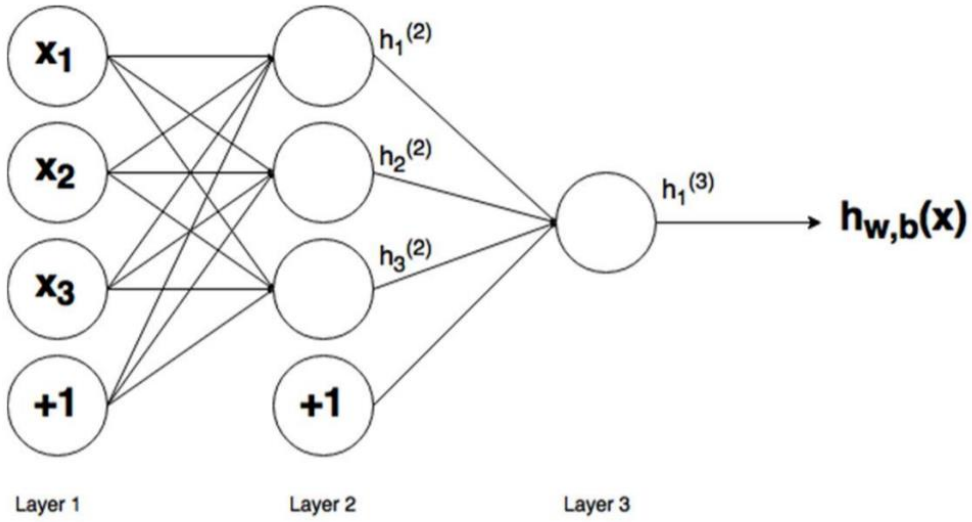
Slika 2.6. Sigmoid funkcija (izvor: [7]).

ANN se može prikazati kao mreža sastavljena od povezanih čvorova. Svaki čvor uzima težinski ulaz (ulaz pomnožen težinskim faktorom), primjenjuje aktivacijsku funkciju sumi svih ulaza te time računa izlaz. Vizualni prikaz neurona prikazan je na slici 2.7. Krug u sredini predstavlja čvor. Čvor je mjesto gdje se nalazi aktivacijska funkcija. Čvor uzima ulazne vrijednosti x_i, množi ih sa težinskim faktorom *w*, zbraja ih te na rezultat primjenjuje aktivacijsku funkciju. Rezultat aktivacije prikazanje kao slovo *h* na slici. Ovakav čvor mreže naziva se još i perceptronom. Vrijednost b, označena kao +1 na ulazu, naziva se *bias* element koji daje dodatnu fleksibilnost čvoru, određuje snagu odnosa promjene ulaz-izlaz.



Slika 2.7. Čvor sa ulazima (izvor: [7]).

Povezivanjem ovakvih čvorova stvara se neuronska mreža. Najjednostavniji oblik neuronske mreže je mreža od tri sloja: prvi ili ulazni sloj, skriveni sloj i sloj s izlaznim vrijednostima ili izlazni sloj. Primjer neuronske mreže prikazan je na slici 2.8.



Slika 2.8. Primjer građe jednostavne neuronske mreže.

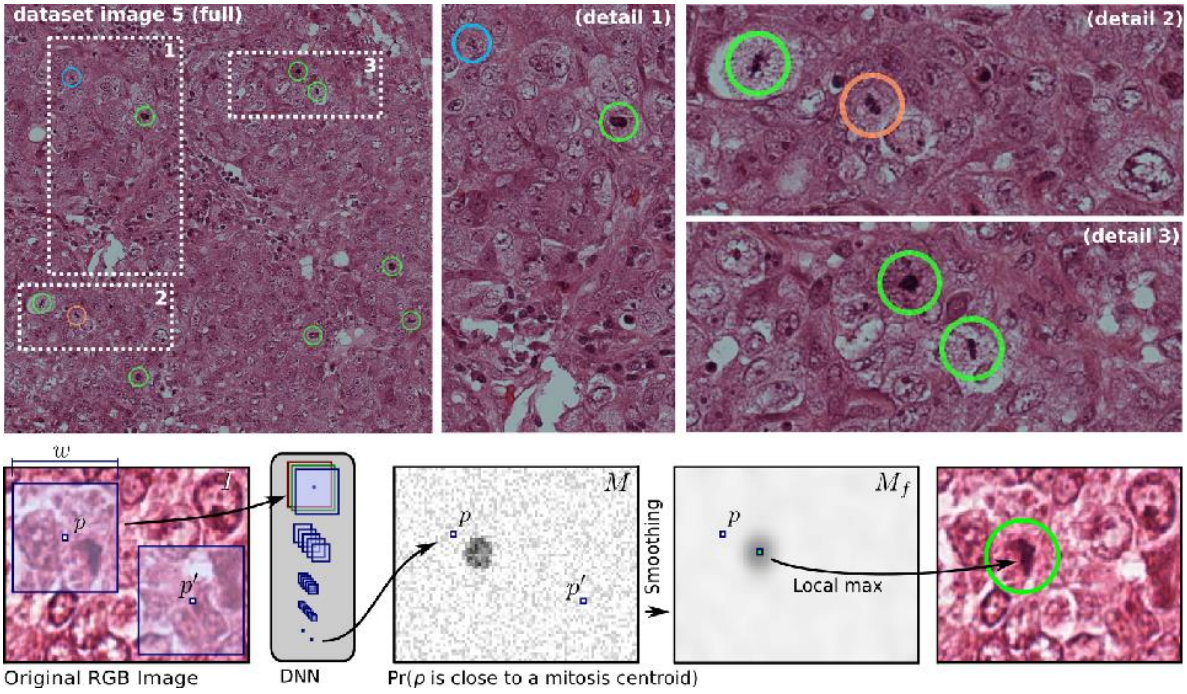
U ovom radu opisano je korištenje neuronskih mreža kao rješenje problema klasifikacije. Ova metoda odabrana je zbog sposobnosti samostalnog odabira značajki i zbog brojnih primjera u kojima se tehnika neuronskih mreža pokazala kao najbolje rješenje. Spomenuti primjeri navedeni su u idućem poglavlju. Odnosno, za razliku od drugih metoda gdje je potrebno ručno odabrati i izračunati značajke slike na kojima se obavlja klasifikacija, neuronske mreže samostalno odabiru i računaju značajke slike. Ova tehnika se pokazala kao najbolje rješenje za prepoznavanje predmeta sa slika u mnogim slučajevima. Samostalno odabiranje značajki slika u neuronskim mrežama postiže se korištenjem posebnih slojeva koji će biti opisani u poglavlju 3.5.

2.3. Primjene neuronskih mreža

U istraživačkom radu [8] izrađen je pregled upotrebe neuronskih mreža, odnosno područja kojem pripadaju neuronske mreže zvano dubinsko učenje. U radu se navodi sedam primjera upotrebe dubinskog učenja: automatsko prepoznavanje glasa, prepoznavanje slika, obrada prirodnog jezika, pronalaženje opojnih sredstava i toksikologija, upravljanje odnosa sa kupcima, sustavi preporuka i bioinformatika.

Za primjer automatskog prepoznavanja govora dan je Google-ov algoritam za prepoznavanje govora [9]. Google je 2012. godine napravio novi korak prema poboljšanju točnosti prepoznavanja govora korištenjem dubinskih neuronskih mreža. Dubinske neuronske mreže zamijenile su dotadašnje Gaussove multivarijabilne razdiobe koja su se koristile unazad 30 godina. Duboke neuronske mreže pokazale su se kao bolji način mjerenja zvuka koji govornik želi proizvesti u svakom trenutku što znatno pospješuje točnost rada algoritma za prepoznavanje govora.

Neuronska mreža korištena je i za prepoznavanje mitoze u histogramskim slikama dojke [10]. Primjer slika koje su se koristile za prepoznavanje prikazane su na slici 2.9. Prepoznavanje mitoze je zahtjevan zadatak. Duboke neuronske mreže u ovakvom slučaju služe kao snažni klasifikatori piksela, obrađuju direktno vrijednosti piksela i nije im potrebno ručno davati ulazne podatke. Duboke neuronske mreže tako uče skup vizualnih uzoraka iz skupa podataka za trening. Duboke neuronske mreže su testirane na javnim bazama podataka gdje su pokazivali znatno bolje rezultate u usporedbi s ostalim testiranim tehnikama, a i njihovo izvođenje nije zahtjevno te mogu proizvesti rezultate na standardnim prijenosnim računalima u roku minuta.



Slika 2.9. Primjer slika za prepoznavanje mitoze (izvor: [10]).

Danas sve popularnija vozila sa sposobnošću autonomne vožnje također zaslugu daju neuronskim mrežama. U radu [11] opisan je proces stvaranja potpunog modela za upravljanje automobilima. U radu se trenirala konvolucijska neuronska mreža za mapiranje piksela kamera direktno na upravljačke naredbe volana vozila. S minimalnim unosom trening podataka sustav je naučio upravljati vozilo u prometu na lokalnim cestama sa ili na cestama bez linija. Sustav je sposoban voziti i u područjima sa slabim vizualnim smjernicama kao što su parkinzi ili neasfaltirane ceste. U ovom radu konvolucijska neuronska mreža nadmašuje standardno prepoznavanje uzoraka i uči potpuni proces upravljanja automobila. Slike se unose u konvolucijsku neuronsku mrežu koja zatim računa moguću naredbu za upravljanje volana. Moguća naredba se uspoređuje sa željnom naredbom te se utezi mreže podešavaju kako bi se izlaz približio željenom. Podešavanje utega je postignuto propagacijom unazad korištenjem *Torch 7* programerskog alata za strojno učenje.

3. ODABRANI ALATI I TEHNIKE

U ovom poglavlju su opisani alati i tehnike koji su korišteni u razvoju modela za prepoznavanje pasmina pasa i Android aplikacije. Prvo su opisani programski jezici, alati i paketi koji su korišteni za realiziranje rješenja. Zatim je opisan korišteni skup podataka te predviđeni problemi koje može prouzrokovati. U nastavku su opisane tehnike koje će se koristiti kako bi se riješio problem prepoznavanja sa slika.

3.1. Alati za strojno učenje

3.1.1. Python i Jupyter notebook

Python je programski jezik koji nudi bogati sadržaj alata za obradu podataka i strojno učenje. Python stavlja naglasak na čitljivost koda, daje funkcije na visokoj razini koje upravljaju varijablama i mogućnost automatskog uklanjanja nepotrebnih podataka što omogućuje korisnicima fokus na logiku rješavanja problema, a ne kodiranje rješenja. Najveća snaga Pythona je veličina standardne biblioteke koja omogućuje obavljanje mnogih zadataka. Podržava brojne internetske formate i protokole, sadrži module za izradu grafičkih sučelja, upravljanjem bazama podataka, aritmetičke funkcije, prevoditelje za pokretanje u drugim jezicima i brojne druge dodatke. Od ožujka 2018. godine *Python Package Index* (*PyPI*), službeni repozitorij za neslužbene programe, sadrži preko 130,000 paketa sa širokim opsegom funkcionalnosti [12]. Još jedan od razloga popularnosti Python jezika danas je široka, aktivna zajednica programera koja konstantno proširuje opseg dodatnih biblioteka i doprinosi unaprjeđenju starih.

Jupyter notebook je jedan od razvojnih okolina koji podržava Python. Ova interaktivna okolina se pokreće u prozoru internetskog pretraživača. Pokreće se s poslužitelja ili lokalno s računala. Kod napisan u Jupyter notebook-u sprema se kao JSON (.ipynb format) dokument. Sastoji se od predane liste ulazno/izlaznih ćelija koje se mogu izvoditi zasebno, a mogu sadržavati kod, tekst, matematičke operacije, grafove i drugo. Planirano korištenje ove strukture pruža jednostavnost kreiranja koda. Moguće je pokretanje dijelova koda koji se žele ponoviti/testirati/izmijeniti bez potrebe pokretanja kompletnog programa. Ovo svojstvo čini Jupyter notebook prikladnim za procese obrade podataka. Datoteke Jupyter notebook-a može se sačuvati i u različitim drugim formatima poput HTML, PDF, Python dokumenta.

3.1.2. TensorFlow

TensorFlow je platforma otvorenog koda za strojno učenje. Kako je navedeno u [13], Tensorflow ima razumljiv, fleksibilni eko-sustav alata, biblioteka i javnih resursa što omogućuje konstantne napretke u strojnom učenju i daje programerima mogućnost jednostavne proizvodnje aplikacija koje koriste strojno učenje. TensorFlow programski paket je razvio tim istraživača i

inženjera koji je radio na *Google Brain* projektu zajedno sa Googleovom *Machine Intelligence Research* organizacijom u svrhu provođenja istraživanja vezanog za strojno učenje i duboke neuronske mreže. Ovaj sustav ima široku primjenu i u drugim granama znanosti. TensorFlow se može izvoditi korištenjem Python ili C++ programerskog jezika. Ime dolazi od operacije koju duboke neuronske mreže izvode na višedimenzionalnim nizovima, koji se nazivaju tenzori (engl. *tensors*). TensorFlow se može pokretati na više CPU i GPU procesora. TensorFlow je omogućen na 64-bitnim Linux, macOS, Windows operacijskim sustavim kao i na mobilnim platformama kao Android i iOS. Na slici 3.1 prikazan je TensorFlow logo.

Slika 3.1. TensorFlow logo (izvor: [13]).

Godine 2017. , Google je objavio programski paket TensorFlow Lite koji proizveden posebno za razvoj mobilnih aplikacija. Ovaj paket omogućuje sučelje za strojno učenje na mobilnim uređajima s brzom izvedbom i malom memorijskom vrijednosti. TensorFlow Lite daje alate za jednostavni prijenos modela iz drugih oblika u TensorFlow Lite oblik.

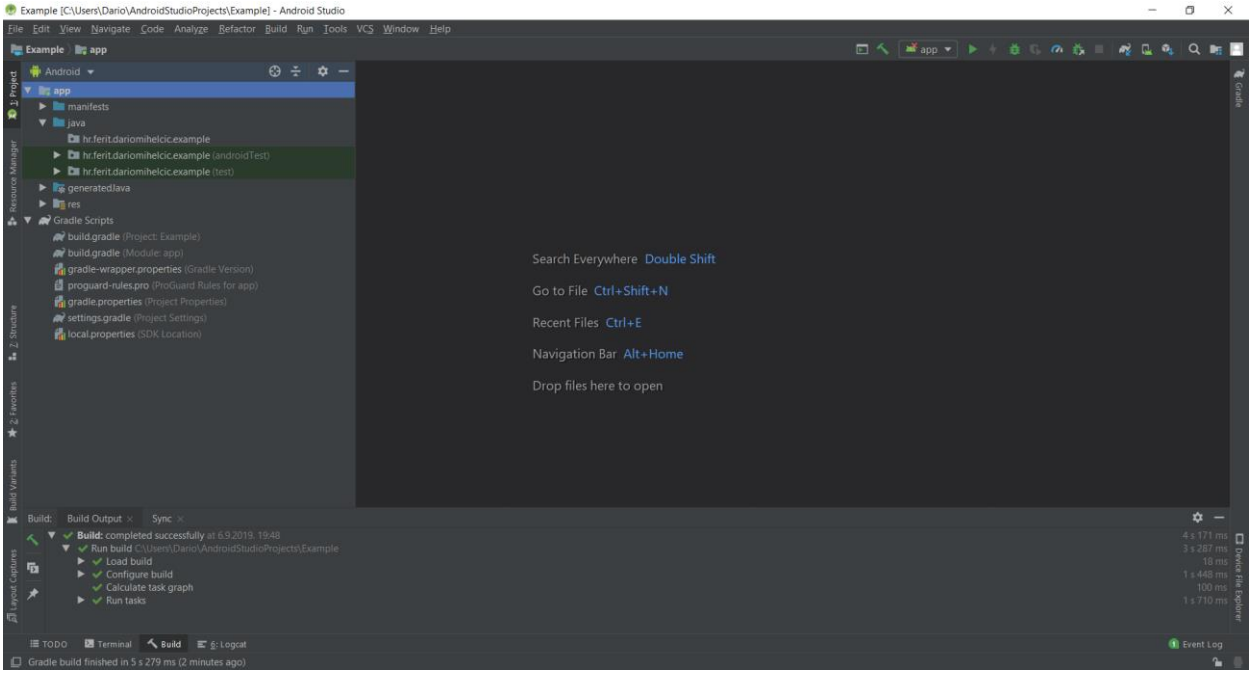
3.1.3. Keras

Keras je aplikacijsko programsko sučelje (engl. *Application Programming Interface, API*) visoke razine za razvoj neuronskih mreža. Keras je napisan u Pythonu i može se pokretati iznad Tensorflow, CNTK ili Theano programskih paketa. Razvijen je s ciljem brzog eksperimentiranja. Keras API prednost daje iskustvu korisnika, koristi najbolje tehnike za smanjivanje tereta na programerima. Daje stabilne i jednostavne API-je, smanjuje broj koraka potrebnih za izvođenje popularnih tehnika i daje jasne i smislene povratne poruke u slučaju korisničkih pogrešaka. Modeli su definirani kao niz ili graf samostalnih modela potpune promjenjivosti koji se mogu koristiti sa što manje mogućih ograničenja. Neuronski slojevi, vrijednosne funkcije, pospješitelji, sheme inicijalizacije, aktivacijske funkcije i regulacijske sheme su samostojeći modeli koji se mogu kombinirati pri izgradnji novih modela. Modeli su opisani u Python kodu, što pruža kompaktnost, jednostavnost ispravljanja grešaka i daje korisniku mogućnost unapređenja.

3.1.4. Android studio

Android Studio je razvojna okolina koja je postala standardom za razvoj Android aplikacija. Bazira se na IntelliJ IDEA-u. Uz moćni IntelliJ uređivač koda i razvojnih alata, Android Studio nudi mnoge usluge koje povećavaju produktivnost pri razvoju Android aplikacija. Android Studio nudi fleksibilan sustav datoteka, brz i bogat emulator, testiranja promjena koda bez ponovnog pokretanja aplikacije, vezu sa GitHub repozitorijem za jednostavniji prijenos dodataka i dijelova koda, raznolike alate za testiranje i mnogo drugih mogućnosti.

U Android Studio-u, projekt obično sadrži nekoliko modula sa datotekama izvornog koda i datotekama resursa (slika 3.2). Moduli mogu biti Android aplikacije, moduli biblioteka, moduli Google App Engine-a. Sve datoteke za izgradnju aplikacije su vidljivi sa najvišeg stupnja unutar Gradle Scripts direktorija i svaka aplikacija sadrži manifest(gdje se nalazi AndroidManifet.xml datoteka), java (direktorij sa svim izvornim kodovima) i "res" (gdje se nalaze svi ostali resursi poput XML datoteka, UI niza i *bitmap*-a slika).



Slika 3.2. Izgled Android Studio sučelja.

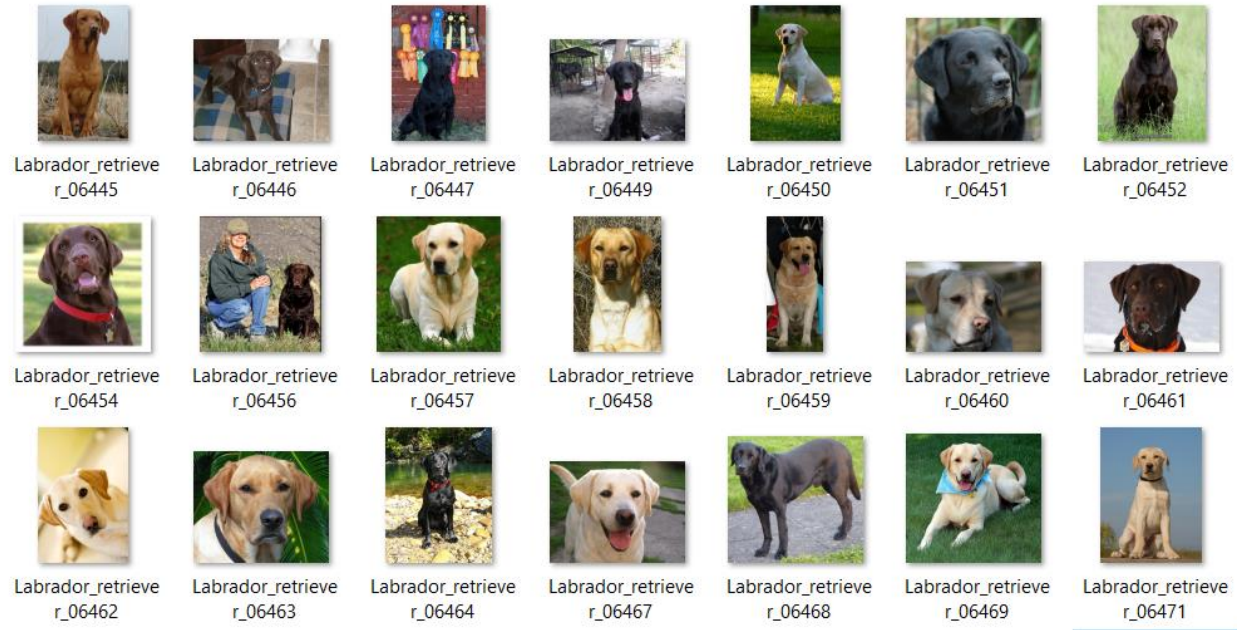
3.2. Skup podataka za strojno učenje

Odabrano je 133 pasmina pasa koje će se moći prepoznati korištenjem aplikacije. Skup podataka preuzet je iz *GitHub* repozitorija edukativne organizacije *Udacity* koja pruža brojne tečajeve. Za svaku pasminu pripremljeno je od 5 do 50 slika za trening i 1 do 10 slika za testiranje. Sve zajedno koristilo se 6680 slika. Kako bi se provjerila točnost modela potrebno je model testirati na slikama koje se ne nalaze u trening skupini. Na slici 3.3 prikazan je dio slika iz skupa

slika za graničarskog škotskog ovčara. Slike u originalnom obliku nisu iste veličine te će se prije procesa treniranja promijeniti u dimenzije 224x224.

Slika 3.3. Trening slike graničarskog škotskog ovčara.

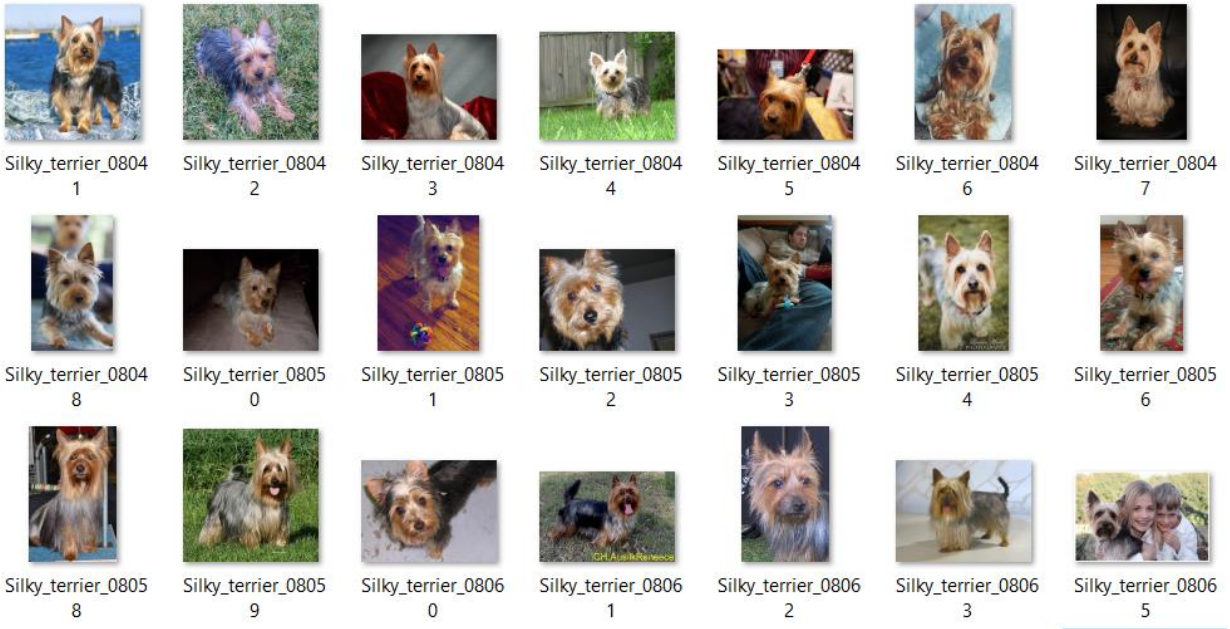
Postoji nekoliko mogućih problematičnih pasmina u ovom skupu. Pasmina retriver labradora (slika 3.4) može biti u žutoj, smeđoj ili crnoj boji. Ovakve razlike u izgledu pripadnika pasmine može dovesti do slabe točnosti klasifikatora pripadajuće klase. Ovaj problem se može riješiti povećanjem broja slika svake varijacije pasmine u skupu podataka te pasmine.



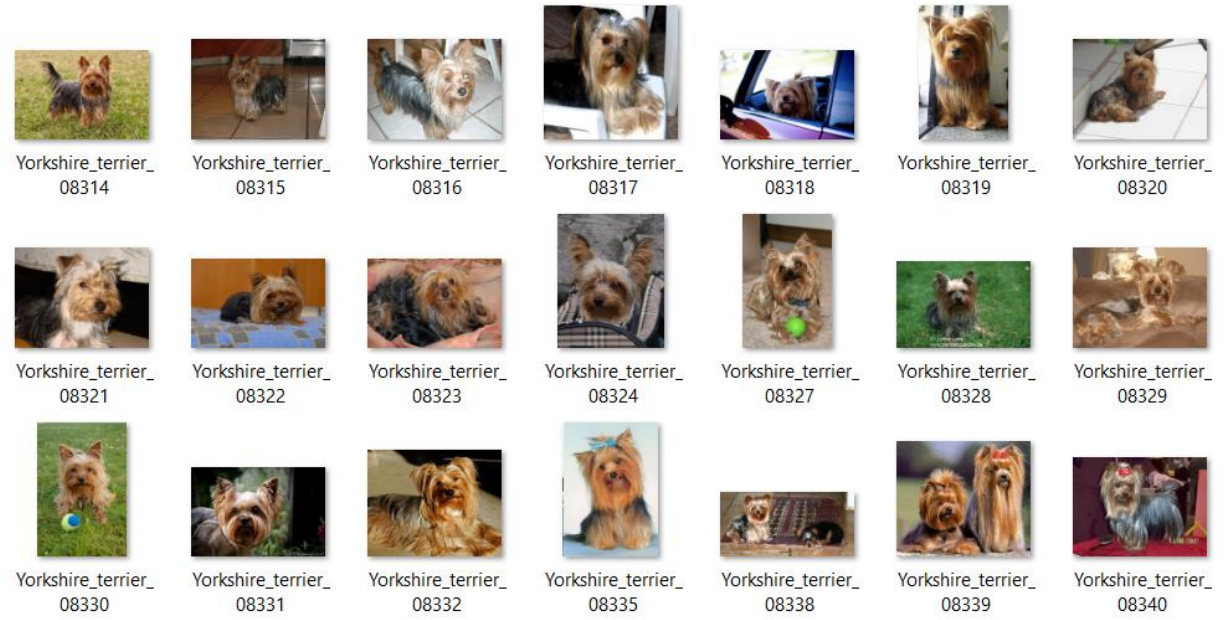
Slika 2.4. Trening slike retriver labradora.

Drugi problem koji smanjuje točnost klasifikacije je sličnost različitih klasa. Na slici 3.5 prikazan je svileni terijer, a na slici 3.6 jorkširski terijer. Ove dvije pasmine su vrlo sličnog izgleda

te se razlika može vidjeti samo u par detalja. Na primjer, svileni terijer ima nešto duže tijelo dok jorkširski terijer ima dužu dlaku. Ovakva suptilna razlika teška je i ljudima za prepoznati pa se može pretpostaviti da će i klasifikator u slučaju prepoznavanja ovih pasmina imati nisku točnost. Točnost bi se mogla povećati dodavanjem dodatnih slika u svaku od klasa no razlika bi bila vrlo mala.

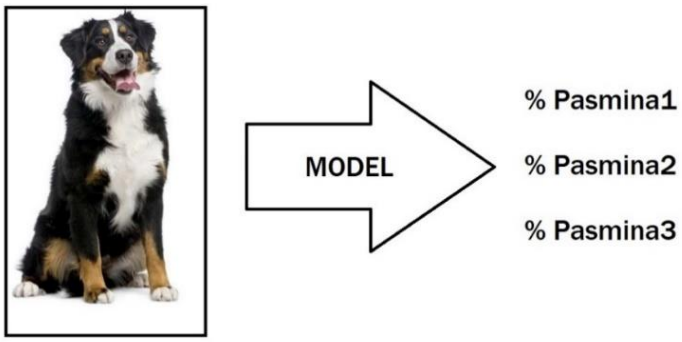


Slika 3.5. Svileni terijer.



Slika 3.6. Jorkširski terijer.

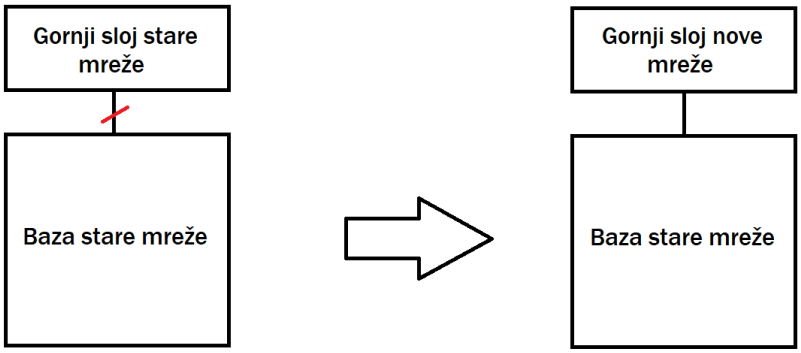
Model konvolucijske mreže će u konačnoj inačicu, kao što je prikazano na slici 3.7, na ulazu primiti sliku psa, a na izlazu će dati tri od 133 pasmine i postotke sigurnosti da se na slici nalazi određena pasmina. Kako bi se model mogao koristiti u mobilnoj aplikaciji potrebno je optimizirati točnost i brzinu modela. Model mora obaviti klasifikaciju u vrlo kratkom vremenu kako bi se rezultati klasifikacije mogli prikazivati na zaslonu u stvarnom vremenu, odnosno pokušava se smanjiti vrijeme između trenutka kada se kamera usmjeri na psa i kada se rezultati klasifikacije pojave na zaslonu.



Slika 3.7. Ulaz i izlaz modela.

3.3. Preneseno učenje

U slučaju nedovoljne količine trening podataka može se koristiti tehnika prenesenog učenja. Postoje brojni modeli za prepoznavanje otvorenog koda kao što su VGG-19, ResNet-50, Inception, Xception. Ovi modeli trenirani su na velikim bazama podataka. Konvolucijske mreže ovih sustava već su postavljene za prepoznavanje predmeta na slici. Ovakve konvolucijske neuronske mreže se mogu podijeliti na dva dijela. Prvi dio se sastoji od konvolucijskih i drugih slojeva za obradu slike. U ovom dijelu svaki sloj ima ulogu pronalaženja osobina slike. U nižim dijelovima mreže slojevi će tražiti linije, rubove, nakon toga slojevi će tražiti forme i oblike itd. Drugi dio mreže, dio potpuno spojenih neurona, ima ulogu klasifikacije značajki promatranog predmeta. Ovaj sloj na temelju značajki slike koje dobije iz prvog dijela daje odluku o predmetu koji se nalazi na slici. Kod prenesenog učenja, uzima se prvi dio istrenirane mreže, bez zadnjih slojeva koji su zaduženi za klasifikaciju (slika 3.8). Uzeti dio mreže služit će kao baza za izradu novog modela. Na bazu se stavljaju novi slojevi za klasifikaciju te se nova mreža trenira sa željenim podatcima. Pri treniranju, svi parametri i utezi bazne mreže se zamrzavaju tako da ih nije moguće mijenjati.



Slika 3.8. Osnovna ideja prenesenog učenja.

3.4. Bazni modeli

Za izgradnju mreže koristi se preneseno učenje sa različitim baznim modelima. Od svakog se modela oduzimaju zadnji slojevi zaslužni za klasifikaciju i stavljaju se novi slojevi koji će prepoznavati pasmine pasa. Prvi model korišten kao baza je ResNet-50 konvolucijska neuronska mreža [14] koja je trenirana na više od milijun slika iz ImageNet baze podataka. Ova mreža ima sposobnost klasificirati 1000 različitih kategorija predmeta. Zahvaljujući tome, ova mreža tvori dobru bazu za prepoznavanje pasmina jer ima bogatu sposobnost prepoznavanja obilježja slike. Drugi model korišten kao bazu je VGG-19 mreža [15], također trenirana na ImageNet slikama no u usporedbi s ResNet-50 ima manje slojeva. ResNet-50 mreža ima 50 slojeva, a VGG-19 ima 19 slojeva konvolucijske neuronske mreže. Sljedeći iskorišteni model je Inception model [16], treniran sa ImageNet bazom podataka i sastoji se od 48 slojeva. Ovaj model ima veće dimenzije ulazne slike u odnosu na prošle modele. Zadnji korišteni bazni model je Xception [17], ovaj model je sličan Inception modelu no ima ukupno 71 slojeva konvolucijske neuronske mreže. U programskoj realizaciji rješenja koristilo se prenesno učenje za stvaranje 4 nova modela za prepoznavanje. Modeli VGG-19, ResNet-50, Inception i Xception baze su trenirane istom bazom trening podataka psećih pasmina. Modeli sa različitim bazama dali su različita rješenja pa su se spomenuti modeli testirali slikama za testiranje kako bi se pronašao najbolji model.

3.5. Struktura konvolucijske neuronske mreže

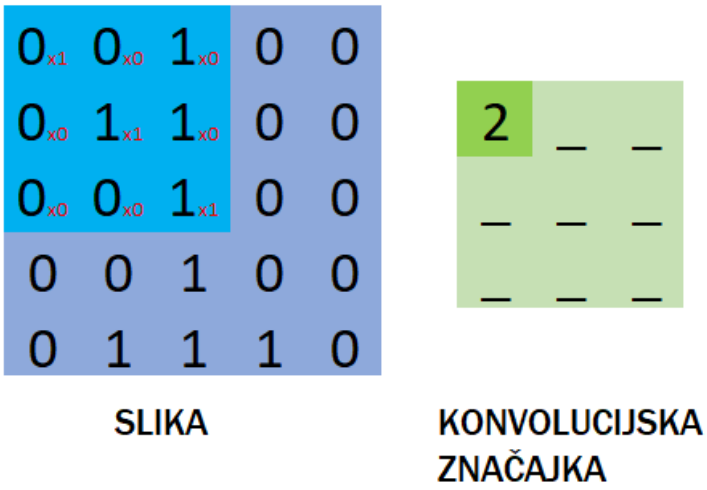
U radu se koriste konvolucijske neuronske mreže za rješavanje problema prepoznavanja pasmina pasa sa slika. Neuronska mreža obično se sastoji od nekoliko slojeva (engl. *layers*). Svaki sloj se sastoji od neurona koji su povezani s neuronima susjednih slojeva. Neuroni sadrže varijable koje se nazivaju težine (engl. *weights*). Da bi se neuronska mreža mogla koristiti, potrebno je namjestiti te utege na određene vrijednosti u procesu zvanom treniranje. Postoje različite forme

treniranja, u ovome radu razmatrana je metoda nadgledanog treniranja. U ovoj metodi prilikom treniranja mreže potrebno je na ulazu dati sliku objekta koji se želi prepoznati i tip objekta koji je na toj slici. Mreža se trenira na velikom broju takvih slika te se pri svakoj iteraciji podešavaju težine. Kod konvolucijskih neuronskih mreža koristi se nekoliko metoda za obradu slike prije nego što se mogu izvući informacije iz slike. U nastavku su opisani slojevi konvolucije, ReLU, maksimalno udruživanje.

3.5.1. Konvolucijski sloj

Prvi sloj koji se koristi u konvolucijskim neuronskim mrežama je konvolucijski sloj. Ovaj sloj na ulazu dobiva cijelu sliku a vraća manju verziju te slike koja se naziva aktivacijskom mapom (engl. *Activation map*). U ovom sloju koriste se posebni filteri koji, kada se primjene na sliku, izvlače željene informacije. Filterima se slijedno prelazi preko slike te se za svako polje piksela računa nova vrijednost koja ovisno o filteru ima izražena polja od interesa.

Na slici 3.9 je prikazana jednostavna konvolucija s filtrom 3x3 dimenzija. Preko početne slike (plavo) prolazi se filterom te se vrijednosti koje se poklapaju množe. Vrijednosti koje su nastale prelaskom filtera se zbrajaju i upisuju u novi nastali prikaz slike (zeleno).

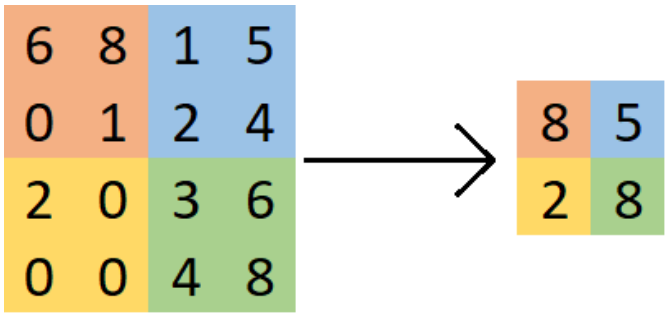


Slika 3.9. Konvolucija.

3.5.2. Maksimalno udruživanje

Kako bi se smanjila količina nepotrebne informacije u neuronskoj mreži koristi se maksimalno udruživanje. Ova metoda se koristi nakon konvolucijskog sloja, odnosno na ulazu dobiva izlaz iz konvolucijskog sloja. Aktivacijska mapa se dijeli na područja željene veličine te se iz tih područja određuje najveća vrijednost (veća vrijednost označava veću vjerojatnost da se u tom području aktivacijske mape nalazi željeni oblik). Od novo nastalih vrijednosti tvori se nova aktivacijska mapa koja je manjih dimenzija nego na ulazu ali sadrži kompaktniju količinu informacije.

Na slici 3.10 je prikazano jednostavno udruživanje gdje je aktivacijska mapa podijeljena u četiri područja, 2x2 dimenzija. Za svako područje odabire se maksimalna vrijednost koja se prenosi u izlaznu sliku. Ako se promatra crveno područje slike, vidljivo je da se područje sastoji od četiri vrijednosti: 6, 8, 0 i 2. Najveća vrijednost iz ovog područja je osam te se samo ta vrijednost prenosi na aktivacijsku mapu desno.



Slika 3.10. Maksimalno udruživanje.

3.5.3. ReLU

ReLU (engl. *rectified linear unit*) je tip aktivacijske funkcije koji se primjenjuje u neuronskim mrežama. ReLu funkcija uklanja negativne vrijednosti iz aktivacijske mape i postavlja ih na nulu a pozitivne vrijednosti prosljeđuje na izlaz. Funkcija se može napisati kao:

$$f(x)=max(0,x). \tag{3.1}$$

Ovime se povećavaju nelinearne sposobnosti mreže. Ova funkcija se češće koristi u odnosu na druge aktivacijske funkcije (sigmoid funkcija, hiperbolični tangens) zbog jednostavnosti funkcije i brže vrijeme računanja bez velike promjene u točnosti.

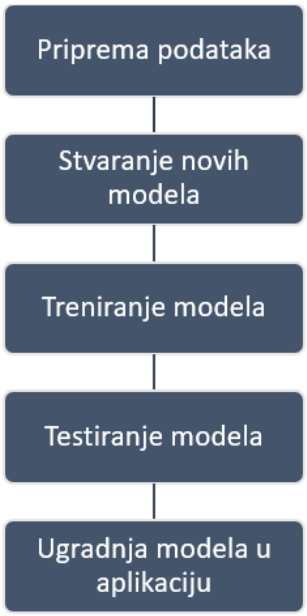
3.5.4. Potpuno povezani sloj

Nakon nekoliko uzastopno povezanih konvolucijskih slojeva i slojeva maksimalnog udruživanja moguće je izvući dovoljno informacija za odluku o klasifikaciji. Na kraju mreže se stavlja nekoliko potpuno povezanih slojeva. Ovi slojevi imaju sposobnost odluke, uzimaju do tada obrađene aktivacijske mape te na izlazu daju konkretni rezultat iz liste mogućih rezultata.

Mreže često dolaze u slučaj kada se previše prilagode slikama iz trening skupine. U ovom slučaju će mreža dobro prepoznavati slike iz trening skupine no pri susretom sa novim slikama koje prikazuju iste klase postotak točnosti je znatno manji. Kako bi se to izbjeglo koristi se sloj ispuštanja. U ovom sloju namjerno se prekidaju veze između neurona kako bi se smanjila povezanosti između neurona i izbjeglo učenje nepotrebnih informacija sa slika.

4. OSTVARENJE MODELA I APLIKACIJE

Za prepoznavanje psećih pasmina sa slike potrebno je prikupiti dovoljno slika za svaku pasminu. Za prepoznavanje pasmina koristi se preneseno učenje pomoću Keras biblioteke sa TensorFlow pozadinom. Program je napisan u Python programerskom jeziku. Model se pretvara u TensorFlow Lite verziju koja je pogodna za implementiranje u Android aplikaciju. Izgradnja same aplikacije napravljena je u Android Studio-u. Na slici 4.1 prikazan je proces stvaranja modela i implementacije modela u aplikaciju.



Slika 4.1. Proces stvaranja modela

4.1. Priprema podataka

Za izradu neuronske mreže i mobilne aplikacije potrebno je prikupiti informacije o psećim pasminama. Za opis psećih pasmina preuzeti su podatci sa Wikipedije [18] tehnikom *web scraping*-a (programski kod 4.1) za 520 pasmina. Korišten je API Wikipedije kako bi se dobio HTML sadržaj internetske stranice. HTML sadržaj se pretražuje kako bi se pronašli željeni odjeljci. Za pretraživanje kroz HTML sadržaj korišten je BeautifulSoup programski paket za Python. Iz početne tablice preuzeta su imena pasmina te poveznice na stranicu Wikipedije pojedine pasmine. Zatim se pomoću programskog koda preuzima opis sa svake stranice iz popisa. Svi tekstualni podatci su spremljeni u .json obliku kako bi se mogli jednostavno učitati u aplikaciju. U listi su uključene različite podskupine pasmina kao i izumrle pasmine.

```
html = requests.get('https://en.wikipedia.org/wiki/List_of_dog_breeds')

b = BeautifulSoup(html.text, 'html.parser')
b = b.body.table
table_rows = iter(b.find_all(name='tr'))
next(table_rows)

categorys = ['id', 'name', 'country', 'type', 'picture', 'description']

breeds = []

for b_id, row in enumerate(table_rows):

    b_name = ""
    b_country = ""
    b_type = ""
    b_picture = ""
    b_description = ""

    #get breed name
    col_name = row.find(name='a', title=True)
    b_name = col_name['title']

    #get breed country
    col_country = row.find_next('td').find_next('td')
    for a in col_country.find_all(name='a', title=True):
        b_country += a['title'] + ", "
    b_country = b_country[:-2]

    #get breed type
    col_type = row
    for j in range(0, 9):
        col_type = col_type.find_next(name='td')
    if col_type == None:
        b_type = "None"
    else:
        b_type = col_type.text
```

Programski kod 4.1. *Web scraping* kod.

Drugi dio obrade podataka uključuje proces prikupljanja i obrade slika koje će biti iskorištene za treniranje neuronske mreže. Skup slika preuzet je sa Udacity GitHub repozitorija. Set se sastoji od ukupno 8351 slika podijeljenih u 133 skupine (pasmine). Ovaj skup slika dalje je podijeljen na trening (6680 slika), validacijski (835 slika) i testni (836 slika) skup. Nakon učitavanja slika u program potrebno je iste pretvoriti u odgovarajući oblik. Ulaz za modele u Kerasu s Tensorflow pozadinom su posebna višedimenzionalna polja zvana tenzori (engl. *tensor*). ResNet50 bazni model na ulazu mora dobiti tenzor od 4 vrijednosti: (broj_uzoraka, 224, 224, 3). Broj uzoraka je broj slika (trening, validacijskog ili test) skupa. Sljedeća dva broja odnose se na dimenziju slike u matričnom obliku, a posljednji broj odnosi se na broj kanala slike (u ovom slučaju to su RGB kanali).

Funkcija *build_tensor* (programski kod 4.2) zaslužna je za pretvorbu slika. Funkcija prima listu putanji slika. Svaka slika se učitava, preoblikuje se u tražene dimenzije (224, 224) i pretvara se u polje vrijednosti koje se proširuje za dobivanje prikladnog tenzora. Važno je provjeriti izlaz iz baznog modela kako bi se mogao definirati ulaz za novi gornji sloj mreže.

```
def build_tensor(file_path):
    list_tensor = []
    for img_path in tqdm(file_path):
        img = image.load_img(img_path, target_size=(TARGET_SIZE, TARGET_SIZE))
        img_array = image.img_to_array(img)
        img_tensor = np.expand_dims(img_array, axis=0)
        list_tensor.append(img_tensor)
    return np.vstack(list_tensor)
```

Programski kod 4.2. Funkcija koja učitava ulazne podatke.

4.2. Stvaranje i treniranje modela

Prvo se u program učitava bazna mreža modela. To se jednostavno postiže pozivanjem konstruktora baznog modela. Kako bi se dobila mreža bez zadnjih slojeva zaduženih za klasifikaciju u funkciju se postavlja uvjet "include_top = False". Zatim se zamrzavaju utezi bazne mreže jer njih nije potrebno iznova trenirati.

Kako bi se jednostavno mogle isprobati različite bazne mreže definirane su funkcije koje čine proces kreiranja, treniranja i testiranja modela vrlo jednostavnim. Funkcija *create_new_model* (programski kod 4.3) prima ime željenog baznog modela, učitava utege mreže te na baznu mrežu postavlja dva nova sloja: ispuštanje (Dropout) i zgušnjivanja (Dense). Samo se novo dodani slojevi mijenjaju u procesu treniranja. Treniranje se obavlja pozivanjem funkcije *train_model* (programski kod 4.4). Treniranje je podijeljeno u 30 epoha s veličinom serije od 20. Broj epoha označuje koliko će se puta proći kroz sve uzorke iz skupa podataka, a veličina serije označava kroz koliko se slika iterira prije sljedećeg obnavljanja utega mreže. Na kraju svake epohe spremaju se novi parametri utega ukoliko se povećala točnost validacije.

```
def create_new_model(base_name):
    input_tensor = Input(shape=(TARGET_SIZE, TARGET_SIZE, 3))
    base_model = globals()[base_name](
                include_top=False,
                weights='imagenet',
                input_tensor=input_tensor,
                input_shape=(TARGET_SIZE, TARGET_SIZE, 3),
                pooling='avg')
    for layer in base_model.layers:
        layer.trainable = False

    fin_model = Sequential([
            base_model,
            Dropout(0.4),
            Dense(133, activation='softmax')
            ])

    return fin_model
```

Programski kod 4.3. Funkcija create_new_model za stvaranje novih modela.

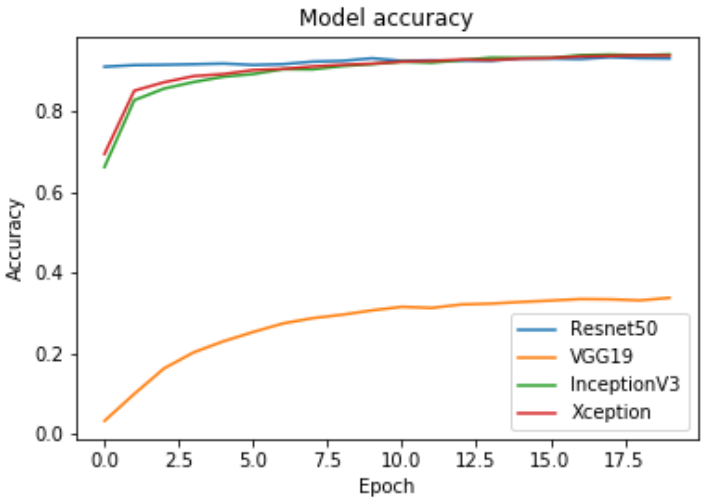
```
def train_model(model, model_name):
    file_path = 'saved_models/weights.best.'+model_name+'.hdf5'
    checkpointer = ModelCheckpoint(filepath=file_path,
                                   verbose=1, save_best_only=True)

    model.fit(train_tensors, train_targets,
          validation_data=(valid_tensors, valid_targets),
          epochs=20, batch_size=20, callbacks=[checkpointer], verbose=1)
```

Programski kod 4.4. Funkcija train_model za treniranje novih modela.

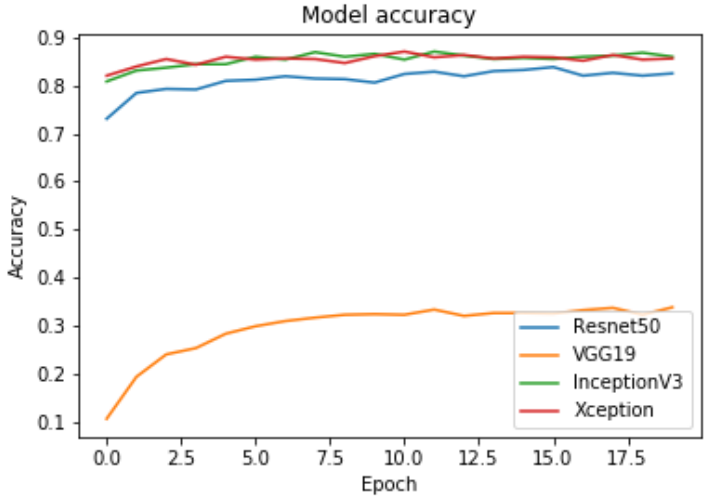
4.3. Testiranje modela

Nakon treniranja četiri različita modela uspoređene su točnosti. Program je tijekom treniranja za svaku sliku zapisivao vrijednosti točnosti treniranja i validacije. Te vrijednosti su kasnije iscrtane na grafove (slika 4.2 i 4.3).

Model sa VGG19 bazom ima najniže vrijednosti točnosti treniranja i validacije. Od početne vrijednosti točnosti 0, točnost VGG19 mreže sporo raste, kod pete epohe postiže svoju prosječnu maksimalnu vrijednost (oko 30%) koja se do kraja treniranja ne popravlja znatno. Resnet50 mreža imam znatno veću točnost u odnosu na VGG19 mrežu. Točnost treniranja konstantno je visoka, iznad 80%, no u usporedbi s točnosti validacije može se primijetiti pretjerano podudaranje s testnim podatcima (engl. *overfitting*). U ovakvom slučaju točnost treninga nije valjana reprezentacija točnosti mreže, odnosno točnost mreže će biti niža pri testiranju novih, neviđenih slika. InceptionV3 i Xception mreža imaju podjednak rast točnosti te su im krajnje vrijednosti točnosti gotovo jednake.



Slika 4.2. Usporedba točnosti treniranja.



Slika 4.3. Usporedba točnosti validacije.

Kako bi se testirao rad modela napisana je funkcija test_model (programski kod 4.5). Funkcija prima model i ime modela. Učitava se model i najbolji utezi odabranog modela koji su određeni procesom treniranja. Modelu se daju slike za trening, te se rezultati koje model daje uspoređuje sa stvarnim klasama slika. Točnost modela se računa tako da se suma točnih klasifikacija dijeli sa ukupnim brojem slika testne skupine.

```
def test_model(model, model_name):
    file_path = 'saved_models/weights.best.'+model_name+'.hdf5'
    model.load_weights(file_path)
    model_predictions = [np.argmax(model.predict(np.expand_dims(feature, axis=0))) for feature in test_tensors]

    test_accuracy = 10000*np.sum(np.array(model_predictions)==np.argmax(test_targets, axis=1))/len(model_predictions)
    print('Test accuracy: %.4f%%' % test_accuracy)
```

Programski kod 4.5. Funkcija test_model.

Na slici 4.4 prikazani su rezultati testiranja četiri novih modela za klasifikaciju psećih pasmina. Usporedbom točnosti testiranja Xception mreža pokazuje veću točnost od InceptionV3 mreže. Resnet50 mreža ima nešto manji postotak točnosti u odnosu na InceptionV3 mrežu. VGG19 daje očekivan postotak točnosti, znatno manji od ostalih mreža.

```
Resnet50 test accuracy: 80.9809%
VGG19 test accuracy: 33.4928%
InceptionV3 test accuracy: 81.5789%
Xception test accuracy: 85.5263%
```

Slika 4.4. Usporedba točnosti testiranja.

4.4. Ugradnja modela u aplikaciju

Kako bi se novi model mogao integrirati u mobilnu aplikaciju potrebno je model pretvoriti u prikladan oblik. Nova biblioteka TensorFlow Lite stvorena je sa tom namjerom. Ova nova biblioteka daje funkcije za pretvaranje Keras (.h5) modela u TensorFlow Lite model (.tflite). TensorFlow Lite je razvijen sa ciljem pokretanja na Androidu, daje funkcije sa kojima je jednostavno učitati i pokrenuti model unutar aplikacije. U programskom kodu 4.6 prikazan je postupak pretvaranja .hdf5 modela u .tflite model.

```
converter = tf.lite.TFLiteConverter.from_keras_model_file('saved_models/final_Xception_model_1.0.hdf5')
tflite_model = converter.convert()
open("model_3.1.tflite", "wb").write(tflite_model)
```

Programski kod 4.6. Pretvaranje .hdf5 u .tflite model.

Model je smješten u klasi *Classifier* (programski kod 4.7), u kojoj se nalazi i lista klasa (psećih pasmina). Ova klasa sadrži funkcije za učitavanje modela, obradu ulaznih podataka, pokretanje inferencije klasifikatora. Dvije važne funkcije koje se mogu pozivati izvan klase su funkcija konstruktora i *recognizeImage* funkcija (programski kod 4.8). Pri konstrukciji instance klase učitava se model i lista klasa iz *assets* datoteke, alocira se potrebna memorija za ulazne vrijednosti (sliku) i izlazne klase. Funkcija *recognizeImage* prima *Bitmap* oblik slike koju se želi klasificirati, pretvara ju u potrebni oblik te poziva proces inferencije na toj slici. Rezultati se spremaju u obliku polja instanci *Recognition* klase. *Recognition* klasa sadrži id, ime i postotak sigurnosti prepoznate klase. Rezultati su poredani u polju po postotku sigurnosti.

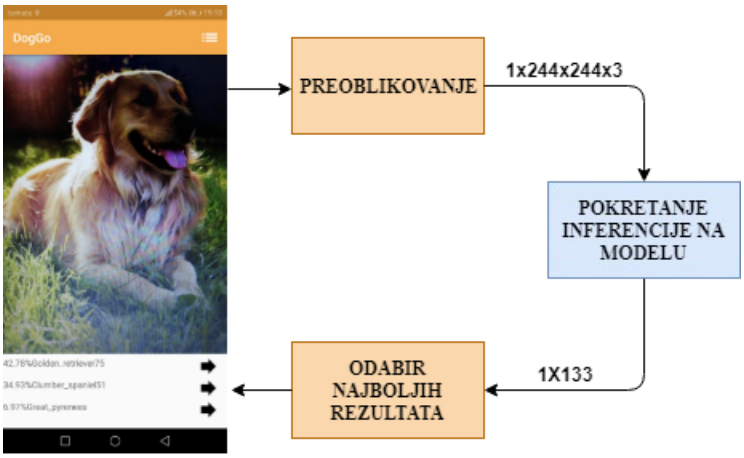
```
public class Classifier {

    private static final int MAX_RESULTS = 3;
    private static final int DIM_BATCH_SIZE = 1;
    private static final int DIM_PIXEL_SIZE = 3;
    private static final int IMAGE_SIZE_X = 224;
    private static final int IMAGE_SIZE_Y = 224;
    private static final int BYTES_PER_CHANNEL = 4;

    private static final String MODEL_PATH = "model_MBN_1.12.tflite";
    private static final String LABEL_PATH = "labels_dogs.txt";

    private final int[] intValues = new int[IMAGE_SIZE_X * IMAGE_SIZE_Y];
    private final Interpreter.Options tfliteOptions = new Interpreter.Opti-
ons();
    private float[][] labelProbArray = null;
    private MappedByteBuffer tfliteModel;
    private List<String> labels;

    private ByteBuffer imgData = null;
    private Interpreter tflite;
```

Programski kod 4.7. Klasa Classifier.

```
public List<Recognition> recognizeImage(final Bitmap bitmap) {
    convertBitmapToByteBuffer(bitmap);
    runInference();
    PriorityQueue<Recognition> pq =
            new PriorityQueue<Recognition>(
                    3,
                    new Comparator<Recognition>() {
                        @Override
                        public int compare(Recognition lhs, Recognition
rhs) {
                            return Float.compare(rhs.getConfidence(),
lhs.getConfidence());
                        }
                    });
    for (int i = 0; i < labels.size(); ++i) {
        pq.add(
                new Recognition(
                        "" + i,
                        labels.size() > i ? labels.get(i) : "unknown",
                        getNormalizedProbability(i)));
    }
    final ArrayList<Recognition> recognitions = new
ArrayList<Recognition>();
    int recognitionsSize = Math.min(pq.size(), MAX_RESULTS);
    for (int i = 0; i < recognitionsSize; ++i) {
        recognitions.add(pq.poll());
    }
    return recognitions;
}
```

Programski kod 4.8. Funkcija recognizeImage.

Na slici 4.5 prikazan je proces klasifikacije slike. Slika se preuzima sa kamere te se pretvara u matrični zapis, oblik (1x244x244x3) koji klasifikacijski model prima na ulazu. Zatim se sa slika prosljeđuje modelu koji vrši klasifikaciju nad slikom. Kao izlaz daje popis od 133 brojeva. Svaki broj predstavlja vjerojatnost da se jedna od klasa nalazi na slici. Ova lista vrijednosti filtrira se tako da se uzimaju samo tri rezultata sa najvećim vjerojatnostima. Odabrana tri rezultata se ispisuju u prozoru aplikacije te se svaki rezultat povezuje s opisom odgovarajuće pasmine kako bi se moglo pritiskom na strjelicu pokraj rezultata otvoriti opisni prozor pasmine.



Slika 4.5. Proces klasifikacije u aplikaciji.

4.5. Klase aplikacije za prikaz sadržaja

Za dobavljanje slika s kamere i provođenje procesa klasificiranja zaslužna je *CameraFragment* klasa (programski kod 4.9). Pri prvom pokretanju aplikacije, ova klasa zahtjeva pristup kameri. U fragmentu se nalazi instanca Classifier koja će se koristiti za klasificiranje podataka. U fragmentu je definiran proces koji uzima sliku iz kamere uređaja i dinamički je prikazuje u *TextureView* pogledu. Proces uključuje i konstantno obavljanje klasifikacije na preuzetim slikama. Rezultati klasifikacije obnavljaju se u stvarnom vremenu zajedeno s promjenom ulaza na kameri.

Uz svaki rezultat prepoznavanja stoji gumb za otvaranje opisnog prozora aplikacije. Ovaj prozor je definiran u DogDescriptionFragment klasi (programski kod 4.10). Svaki puta kada se pritisne gumb za opis pasmine stvara se novi opisni fragment. Potrebno je konstruktoru klase dati informacije o odabranoj pasmini. To se radi preko *bundle* varijable.

```
package hr.ferit.dariomihelcic.doggo.fragments;

import ...

public class CameraFragment extends Fragment {

    private static final String TAG = "CameraFragment";
    private static final int PERMISSION_REQUEST_CODE = 100;
    private static final int MAX_PREVIEW_WIDTH = 1920;
    private static final int MAX_PREVIEW_HEIGHT = 1080;

    private CameraCaptureSession captureSession;
    private Object lock = new Object();
    private boolean runClassifier = false;
    private boolean checkPermission = false;
    private Classifier imageClassifier;
    private DogListHolder holder = new DogListHolder();
    private String cameraId;
    private CameraDevice cameraDevice;
    private Size previewSize;

    private HandlerThread backgroundThread;
    private Handler backgroundHandler;
    private ImageReader imageReader;
    private CaptureRequest.Builder previewRequestBuilder;
    private CaptureRequest previewRequest;


    private Semaphore cameraOpenCloseLock = new Semaphore(1);
    private AutoFitTextureView textureView;
    private TextView tvResultOne;
    private TextView tvResultTwo;
    private TextView tvResultThree;
    private ImageButton btnDogOne;
    private ImageButton btnDogTwo;
    private ImageButton btnDogThree;
    private int[] result_id = new int[3];

    public CameraFragment() {}
```

Programski kod 4.9. CameraFragment klasa.

U aplikaciji se još nalazi fragment koji će na zaslonu prikazati listu svih pasmina. Taj fragment definiran je u DogListFragment klasi (programski kod 4.11). Ovaj fragment koristi *RecyclerView* za ispis liste pasmina. Svaka pasmina reprezentirana je malom slikom i imenom pasmine. Svaka ćelija unutar *RecyclerView*-a može se pritisnuti kako bi se otvorio opisni fragment odabrane pasmine. Lista se može filtrirati upisom teksta u polje za pretraživanje što će smanjiti listu pasmina na one pasmine koje u imenu sadrže uneseni niz znakova.

```
package hr.ferit.dariomihelcic.doggo.fragments;

import ...

public class CameraFragment extends Fragment {

    private static final String TAG = "CameraFragment";
    private static final int PERMISSION_REQUEST_CODE = 100;
    private static final int MAX_PREVIEW_WIDTH = 1920;
    private static final int MAX_PREVIEW_HEIGHT = 1080;

    private CameraCaptureSession captureSession;
    private Object lock = new Object();
    private boolean runClassifier = false;
    private boolean checkPermission = false;
    private Classifier imageClassifier;
    private DogListHolder holder = new DogListHolder();
    private String cameraId;
    private CameraDevice cameraDevice;
    private Size previewSize;

    private HandlerThread backgroundThread;
    private Handler backgroundHandler;
    private ImageReader imageReader;
    private CaptureRequest.Builder previewRequestBuilder;
    private CaptureRequest previewRequest;


    private Semaphore cameraOpenCloseLock = new Semaphore(1);
    private AutoFitTextureView textureView;
    private TextView tvResultOne;
    private TextView tvResultTwo;
    private TextView tvResultThree;
    private ImageButton btnDogOne;
    private ImageButton btnDogTwo;
    private ImageButton btnDogThree;
    private int[] result_id = new int[3];

    public CameraFragment() {}
```

Programski kod 4.10. Klasa DogDescriptionFragment.

```
import ...

public class DogListFragment extends Fragment implements
SearchView.OnQueryTextListener, NameClickListener {

    private RecyclerView recycler;
    private RecyclerAdapter adapter;

    public static DogListFragment newInstance(){
        return new DogListFragment();
    }

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, @Nullable
ViewGroup container, @Nullable Bundle savedInstanceState) {
        setHasOptionsMenu(true);
        return inflater.inflate(R.layout.fragment_dog_list, container,
false);
    }

    @Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle
savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        setUpRecycler(view);
        setUpRecyclerData();
    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        super.onCreateOptionsMenu(menu, inflater);
        menu.clear();

        inflater.inflate(R.menu.main_menu, menu);
        MenuItem item = menu.findItem(R.id.dog_search);

item.setShowAsAction(MenuItem.SHOW_AS_ACTION_COLLAPSE_ACTION_VIEW|MenuItem.
SHOW_AS_ACTION_IF_ROOM);

        SearchView searchView = (SearchView) item.getActionView();
        searchView.setOnQueryTextListener(this);
    }
```

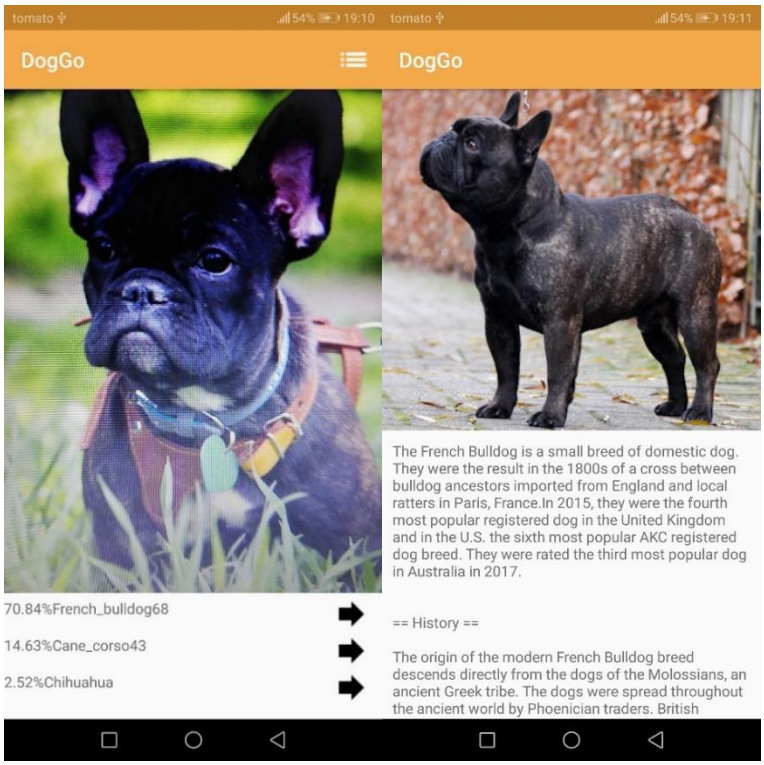
Programski kod 4.11. Klasa ListFragment

5. PRIKAZ RADA APLIKACIJE S ISPITIVANJEM I ANALIZOM

Aplikacija se može koristiti kada se želi saznati pasmina pasa. Vrlo jednostavno sučelje daje trenutne rezultate te dodatne informacije o pasmini. Izbornici i tekstovi u aplikaciji su na engleskom jeziku.

5.1. Izgled aplikacije

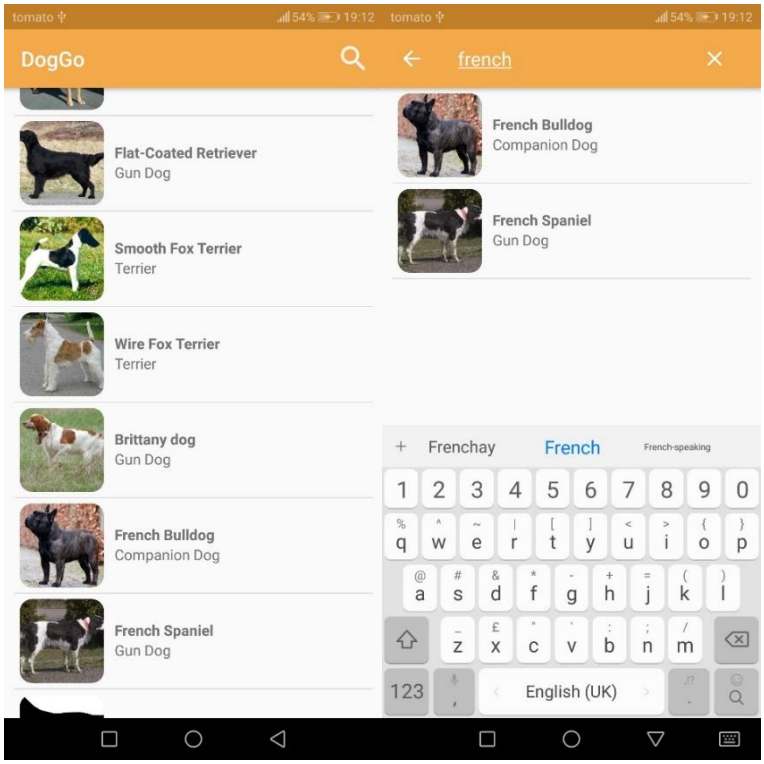
Pri prvom otvaranju, aplikacija će zatražiti od korisnika pristup kameri. U prvom prozoru otvara se polje u kojem se prikazuju kamere uređaja (slika 5.1). U gornjem desnom kutu prozora se nalazi ikona liste. Ikona liste vodi do prozora liste pasmina. Ispod polja kamere nalaze se tri reda teksta. Svaka linija teksta predstavlja jedno rješenje klasifikacije slike. U redu je prikazan postotak sigurnosti modela te naziv klase. Rezultati su poredani od najvećeg postotka sigurnosti do najmanjeg postotka sigurnosti. Uz svaku klasu nalazi se ikona strjelice koje vode do opisnog prozora. U opisnom prozoru prikazana je slika i opis odabrane pasmine (slika 5.1).



Slika 5.1. Prvi prozor aplikacije(lijevo), opisni prozor(desno).

U prozoru liste pasmina mogu se vidjeti svih 520 pasmina (slika 5.2). Svaka pasmina sadrži sliku pasmine i ime pasmine. Lista se može pretraživati slijedno listanjem prema dolje ili putem tekstualnog pretražitelja. U gornjem desnom kutu nalazi se ikona za pretraživanje liste.

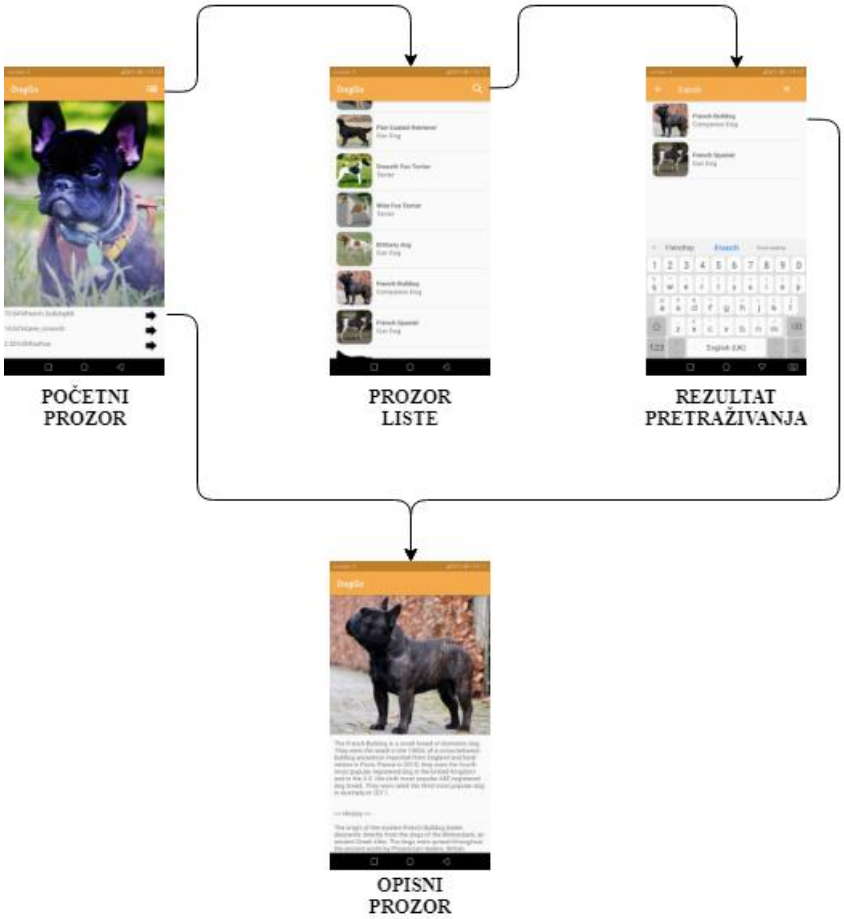
Pretraživanjem liste smanjuje se broj podataka u prozoru, a ispisuju se samo one pasmine koje sadrže traženi tekst u imenu.



Slika 5.2. Prozor liste (lijevo), pretraživanje liste (desno).

5.2. Korištenje aplikacije

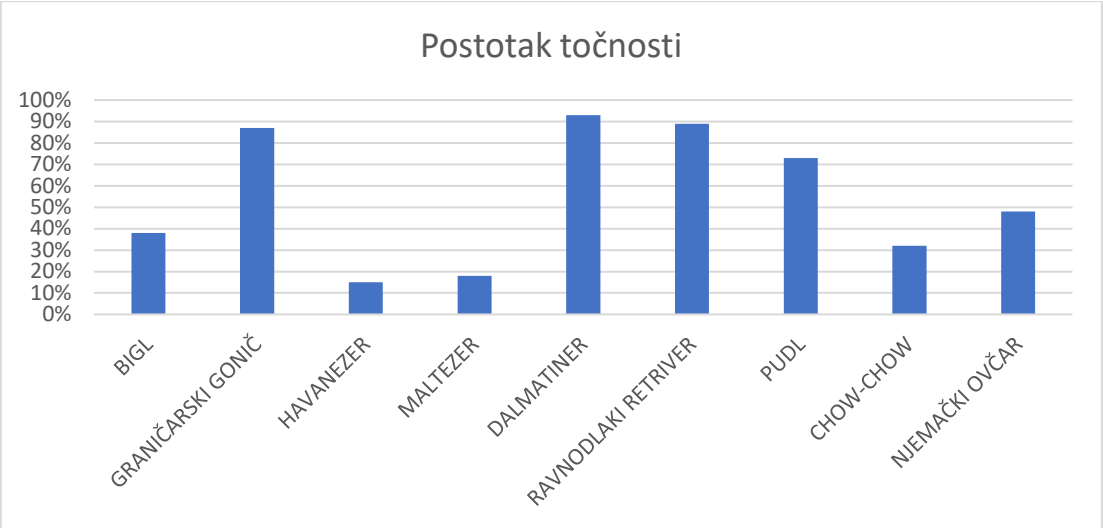
Aplikacija ima vrlo jednostavno sučelje koje omogućuje lako korištenje. Na slici 5.3 prikazan je graf korištenja aplikacije. Korisnik treba samo usmjeriti kameru u psa kojem želi otkriti pasminu. U prvom prozoru, u poljima za rješenje konstantno će se obnavljati rezultati klasifikacije. Kada korisnik vidi željeni postotak točnosti može pritisnuti ikonu strjelice pokraj željenog rezultata kako bi saznao više o pasmini. Pritiskom na ikonu otvara se prozor s opisom pasmine. Ako korisnik želi ručno pretražiti listu pasmina, može otvoriti prozor liste pasmina. Kada se otvori prozor liste pasmina može se listanjem pregledati sve pasmine ponuđene u aplikaciji. Klikom na pasminu u prozoru liste otvara se opisni prozor te pasmine. Ako korisnik želi pronaći pasminu preko imena može otvoriti opciju pretraživanja liste u prozoru liste. Upisivanjem imena dinamično se mijenja lista pasa u prozoru liste. Korisnik može u bilo kojem trenutku odabrati pasminu i otvoriti opisni prozor pasmine.



Slika 5.3. Graf korištenja aplikacije.

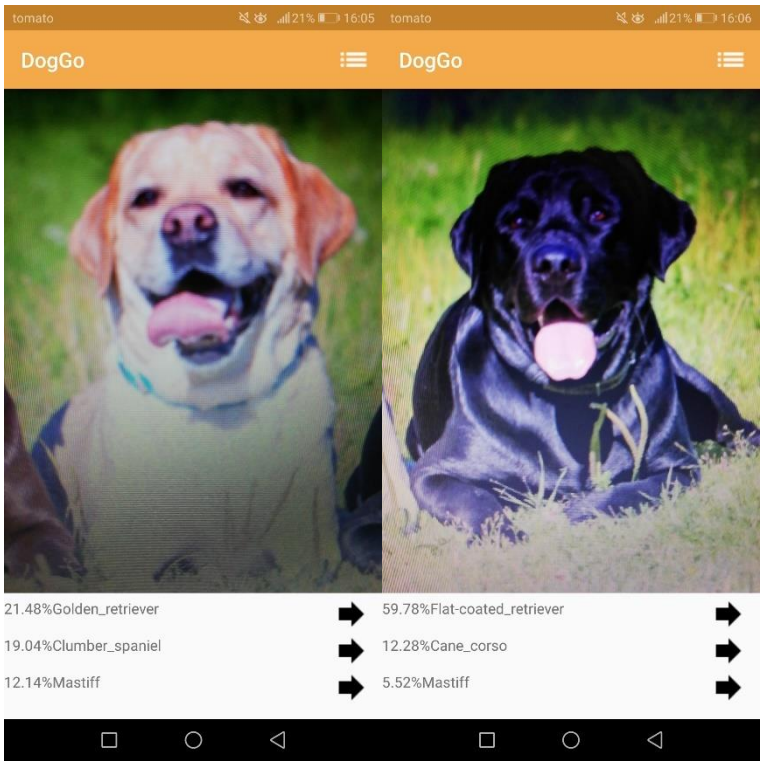
5.3. Testiranje rada aplikacije

Aplikacije su testirane na slikama koje se ne nalaze u trening skupini podataka. Odabrano je 10 pasmina od mogućih 133 pasmina. Za svaku pasminu uzete su dvije različite slike. Aplikacija se testirala tako da se kamera usmjeri u sliku pasmine te se sa prozora aplikacije zapisana dva rezultata za svaku sliku. Dobivena četiri postotka točnosti zbrajaju se i dijele s 4. Dobiveni rezultati po pasmini prikazani su na grafu na slici 5.4.



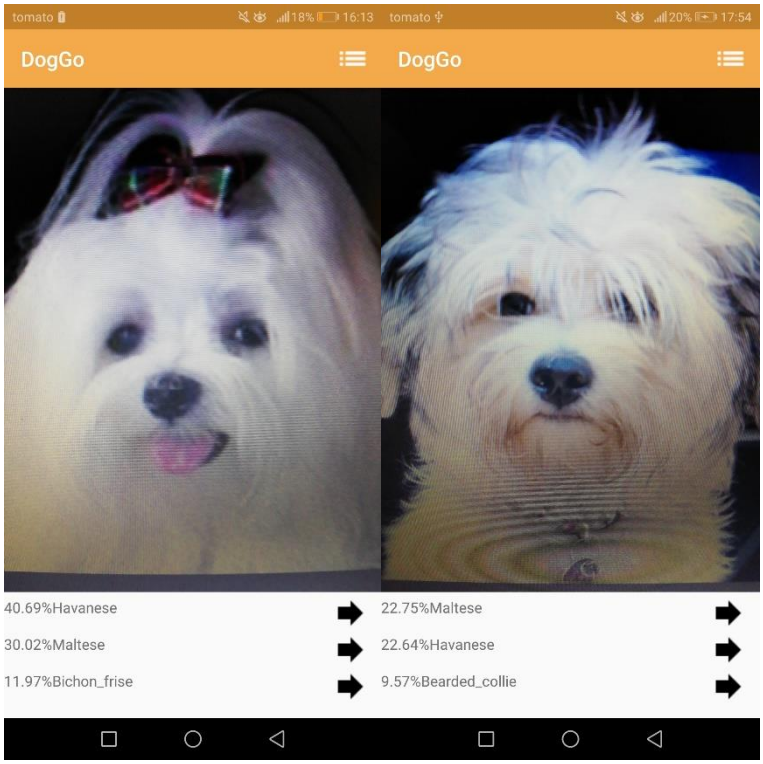
Slika 5.4. Rezultati testiranja aplikacije.

Primjer jedne od testiranih pasmine prikazan je na slici 5.5. Na slici se nalazi labrador retriver no ta pasmina se ne nalazi niti u jednom od rezultata prikazanih u prozoru aplikacije. Grešku proizvodi činjenica da ova pasmina dijeli mnogo karakteristika sa drugim pasminama a sama pasmina ima mnogo varijanti. Zlatni labrador retriver jako sliči zlatnom retriveru po boji i po obliku lica. Općenito, pasmine retrivera se prepoznaju sa niskim postotkom sigurnosti. U ovom slučaju klasifikator nikada neće postići visoki postotak sigurnosti.



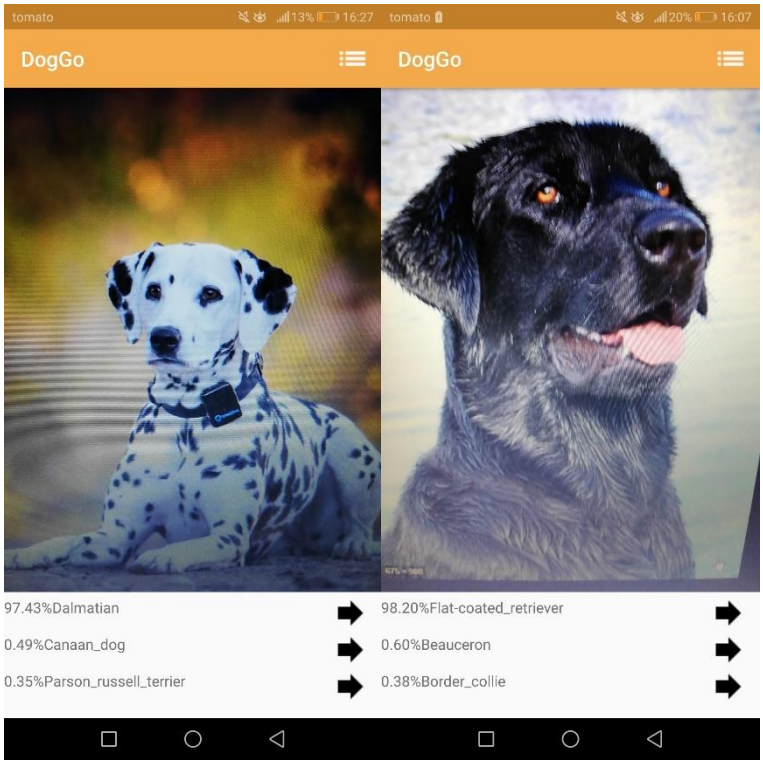
Slika 5.5. Primjeri prepoznavanja labrador retrivera.

Na slici 5.6 prikazani su rezultati prepoznavanja pasmine maltezer i havanezer. Pripadnici ove pasmine imaju konzistentan izgled, male građe sa bijelom dlakom. Problem kod ove pasmine je sličnost sa havanezer pasminom. Model u većini slučajeva ima podvojeno mišljenje kod prepoznavanja ovih pasmina pa je rezultat nizak postotak sigurnosti.



Slika 5.6. Primjer prepoznavanja maltezera (lijevo) i havanezera (desno).

Pasmine poput dalmatinera (slika 5.7) koje imaju specifičan izgled uvijek se prepoznaju sa visokim postotkom točnosti. Razlog tome je poseban uzorak na tijelu dalmatinera koji model lako prepoznaje. Pasmina ravnodlakog retrivera (slika 5.7) također se prepozna s visokim postotkom točnosti. Ova pasmina nema poseban uzorak na tijelu no zbog posebne kombinacije oblika lica, crne dlake i oblik ušiju model može točno klasificirat pasminu.



Slika 5.7. Primjer prepoznavanja dalmatinera (lijevo) i ravnodlakog retrivera (desno).

6. ZAKLJUČAK

Cilj rada bio je izrada aplikacije koja prepoznaje pasmine pasa. Na početku rada dana je teorijska podloga vezana za problematiku rada, postojeća rješenja i druge primjene. Istraživanjem mogućih postupaka prepoznavanja slika pronađena je tehnika neuronskih mreža kao najbolja u tom području. Dan je i pregled alata i postupaka koji su korišteni u realizaciji rješenja. U radu se koristio programski jezik Python te tehnika prenesenog učenja za izradu konvolucijske neuronske mreže. Izrađena su četiri modela za prepoznavanje koristeći četiri različita bazna modela neuronske mreže. Modeli su testirani na testnim podatcima kako bi se odredio najbolji. Model neuronske mreže pretvoren je u potreban oblik kako bi se model mogao implementirati u Android aplikaciju. Aplikacija sadrži i opise svih psećih pasmina. U aplikaciji je izrađen prozor koji automatski generira rezultate klasifikacije u stvarnom vremenu, bez vremena čekanja što je napredak u odnosu na postojeća rješenja. Testiranjem aplikacije na testnim slikama pokazuje varijacije u postotku točnosti ovisno o karakteristikama pasmine. Klase koje imaju članove s različitim karakteristikama te klase koje imaju karakteristike srodne drugim klasama pokazale su se kao problematične i proizvodile su nizak postotak točnosti. Klase čiji pripadnici imaju specifičan izgled i karakteristike su klasificirane sa visokom razinom točnosti.

Postoji mnogo načina za moguća poboljšanja klasifikacijskog modela. U prvom planu bilo bi korištenje veće baze slika za povećanje točnosti klasifikacije i proširenje mogućih izlaznih klasa. Mogući je porast točnosti primjenom drugačijih hiper-parametara mreže, augmentiranje baze podataka, te promjena strukture krajnjih slojeva mreže. Model bi mogao biti proširen s većim brojem klasa koje se mogu prepoznati, a opisi pasmina mogli bi se prevesti i na druge jezike.

LITERATURA

[1] S. Shalev-Shwartz i S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, 2014.

[2] P. Kamavisdar, S. Saluja i 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] T. Fletcher, *Support Vector Machines Explained*, 2008., dostupno na: https://cling.csd.uwo.ca/cs860/papers/SVM_Explained.pdf, [Pokušaj pristupa Rujan 2019].

[4] P. N. Tan, M. Steinbach, A. Karpatne i V. Kumar, *Introduction to Data Mining*, 2005.

[5] N. Thakur i D. Maheshwari, *A review of image classification techniques*, International Research Journal of Engineering and Technology (IRJET), Vol.4, str. 1588.-1591., Studeni 2017.

[6] R. Janković, *Classifying Cultural Heritage Images by Using Decision Tree Classifiers in WEKA*.

[7] Internetska stranica A. Thomas, *An introduction to neural networks for beginners*, dostupno: https://adventuresinmachinelearning.com/wp-content/uploads/2017/07/An-introduction-to-neural-networks-for-beginners.pdf, Adventures in Machine Learning [Pokušaj pristupa Rujan 2019].

[8] N. F. Hordri, S. S. Yuhaniz i S. M. Shamsuddin, *Deep Learning and Its Applications: A Review*, Postgraduate Annual Research on Informatics Seminar, Universiti Teknologi Malaysia, Kuala Lumpur 2016.

[9] H. Sak, A. Senior, K. Rao, F. Beaufays i J. Schalkwyk, *Fast and Accurate Recurrent Neural Network Acoustic Models for Speech Recognition*, Google, 2015.

[10] D. C. Ciresan, A. Giusti i J. Schmidhuber, *Mitosis Detection in Breast Cancer Histology Images with Deep Neural Networks*, IDSIA, Dalle Molle Institute for Artificial Intelligence, USI-SUPSI, Lugano, Switzerland 2013.

[11] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao i K. Zieba, *End to End Learning for Self-Driving Cars*, e-prints arXiv:1604.0731, 2016.

[12] Internetska stranica *Python*, https://en.wikipedia.org/wiki/Python_(programming_language) [Pokušaj pristupa Rujan 2019].

[13] Internetska stranica *TensorFlow*, https://github.com/tensorflow/tensorflow. [Pokušaj pristupa Rujan 2019].

[14] K. He, S. R. Xiangyu Zhang i J. Sun, *Deep residual learning for image recognition*, u Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778., 2016.

[15] K. Simonyan i Z. Andrew, *Very deep convolutional networks for large-scale image recognition*, arXiv preprint arXiv:1409.1556, 2014.

[16] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens i Z. Wojna., *Rethinking the inception architecture for computer vision*, u Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2818-2826., 2016..

[17] F. Chollet, *Xception: Deep Learning with Depthwise Separable Convolutions*, arXiv preprint, pp.1610-02357., 2017.

[18] Internetska stranica *List of dog breeds*, https://en.wikipedia.org/wiki/List_of_dog_breeds [Pokušaj pristupa Rujan 2019].

SAŽETAK

Cilj ovog rada bio je izraditi model i mobilnu aplikaciju koja može prepoznati pasmine pasa sa slike. U teorijskom dijelu rada objašnjeni su postupci strojnog učenja koji se mogu koristiti za prepoznavanje pasmina sa slika. Nakon danih teorijskih opisa postupaka te istraživanjem primjena postupaka pronađene su neuronske mreže kao najbolje rješenje. Nakon toga, dan je opis alata i algoritama koji se koriste u ostvarenju odabranog postupka. Odabrana je metoda prenesenog učenja za stvaranje modela za prepoznavanje. U programskom dijelu rada izrađene su funkcije za jednostavnu izgradnju modela za prepoznavanje odabranih klasa. Izrađena su četiri modela za klasificiranje 133 psećih pasmina od kojih je testiranjem izabran najbolji model. Izrađena je Android aplikacija u kojoj je model implementiran. Uz mogućnost prepoznavanja pasmina pasa, aplikacija daje korisniku i opise svih pasmina. Rezultati testiranja aplikacije pokazali su određene probleme koji mogu nastati kod klasifikacije objekata sa slika kao što je raznolikost karakteristika članova jedne klase ili zajedničke karakteristike više klasa.

Ključne riječi: Android, klasifikacija slika, neuronske mreže, preneseno učenje, strojno učenje.

ABSTRATCT

Title: ANDROID APPLICATION FOR DOG BREED CLASSIFICATION USING MACHINE LEARNING

The goal of this paper was to create a model and a mobile application that can recognize dog breeds in pictures. Methods of machine learning which can be used for recognizing dog breeds from pictures were explained in the theoretical part of the paper. After the theoretical description of the methods was given and usages of those methods were explored, neural networks were found to be the best solution to the problem. After that, a description of tools and algorithms used to create the solution to the problem were given. Transfer learning was chosen for creating the recognition model. In the programming part of the paper, functions for easy creation of recognition models capable of recognizing the chosen classes were made. Four models capable of recognizing 133 dog breeds were made from which the best one was chosen through testing. Apart from recognizing dog breeds, the application gives the user description of all dog breeds. While testing the applications, some problems that can occur in object classification were found, such as differences in characteristics of members of the same class or characteristics shared among different classes.

Keywords: Android, image classification, neural networks, transfer learning, machine learning.

ŽIVOTOPIS

Dario Mihelčić rođen je 4. prosinca 1997. godine u Slavonskom Brodu. Pohađanje osnovne škole započinje u Starim Perkovcima te kasnije završava osnovnu školu u Vrpolju. Upisao je srednju školu Gimnaziju „Matija Mesić“ u Slavonskom Brodu gdje je prošao sve četiri godine. Nakon srednje škole upisuje preddiplomski sveučilišni studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

_____

Dario Mihelčić

PRILOZI (na CD-u)

Prilog 1. Dokument završnog rada
Prilog 2. Pdf završnog rada
Prilog 3. Programski kod strojnog učenja
Prilog 4. APK Android aplikacije