Mobilna Android aplikacija za prepoznavanje objekata primjenom postupaka strojnog učenja

Rekić, Tomislav

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

Sveučilišni preddiplomski studij računarstva

MOBILNA ANDROID APLIKACIJA ZA PREPOZNAVANJE OBJEKATA PRIMJENOM POSTUPAKA STROJNOG UČENJA

Završni rad

Tomislav Rekić

Osijek, 2019.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

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

Osijek, 17.09.2019.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada

Ime i prezime studenta:	Tomislav Rekić
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	4173b, 19.09.2018.
OIB studenta:	06929739237
Mentor:	Prof.dr.sc. Goran Martinović
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Mobilna Android aplikacija za prepoznavanje objekata primjenom postupaka strojnog učenja
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Predložena ocjena 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: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	17.09.2019.
Datum potvrde ocjene Odbora:	25.09.2019.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 26.09.2019.

Ime i prezime studenta:	Tomislav Rekić
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	4173b, 19.09.2018.
Ephorus podudaranje [%]:	10

Ovom izjavom izjavljujem da je rad pod nazivom**: Mobilna Android aplikacija za prepoznavanje objekata primjenom postupaka 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

Strojno učenje je nedavno doživjelo veliki skok u popularnosti i njeno područje primjene se znatno proširilo. Zbog velikog potencijala strojnog učenja svakodnevno se pojavljuju i nove primjene. Robotski vid i klasifikacija slika neka su od područja strojnog učenja te je vrlo važno implementirati te tehnologije na razne platforme kako bi se ubrzao njen razvoj i zainteresiranost među populacijom.

Cilj ovog završnog rada je izraditi mobilnu aplikaciju koja primjenom ugrađenog postupka strojnog učenja omogućuje prepoznavanje vrste životinje snimljene kamerom mobilnog uređaja . Ostvarenje ovog rada može se prikazati kroz dva dijela aktivnosti.

Prvi dio bavi se pripremom postupka strojnog učenja, uključujući izgradnju modela strojnim učenjem, a drugi dio prikazuje sučelje prema korisniku aplikacije koje kroz mobilnu aplikaciju omogućuje korištenje ugrađenog algoritma strojnog učenja. Odabrani algoritam strojnog učenja je konvolucijska neuronska mreža. Strojno učenje je omogućeno s pomoću biblioteke TensorFlow, a algoritam je implementiran u programskom jeziku Python. Mobilna aplikacija je napravljena za operacijski sustav Android u razvojnoj okolini Android Studio, pri čemu su njene funkcionalnosti napisane u programskom jeziku Java, dok je izgled opisan koristeći opisni jezik XML.

U poglavlju 2 dan je kratki pregled u računalni vid i u algoritme strojnog učenja. U poglavlju 3 opisani su korišteni alati i programski jezici koji omogućuju implementiranje postupka strojnog učenja i razvoj mobilne aplikacije. Poglavlje 4 opisuje programsko rješenje mobilne aplikacije za prepoznavanje snimljenih slika primjenom konvolucijske neuronske mreže, dok je u poglavlju 5 prikazan način korištenja mobilne aplikacije i analizirani rezultati prepoznavanje slika.

1.1. Zadatak završnog rada

U teorijskom dijelu rada treba objasniti postupke strojnog učenja pogodne za prepoznavanje objekata iz slika iz dostupnih skupova podataka (npr. životinje), analizirati i izabrati potrebne programske tehnologije, alate i programske okvire za razvoj mobilne aplikacije i postupaka strojnog učenja. Nadalje, treba razraditi model i programski ostvariti Android mobilnu aplikaciju koja treba omogućiti implementaciju izabranog postupka strojnog učenja, dohvat i pripremu podataka, analizu, te prikaz i pohranu rezultata to s ciljem promidžbe zaštite životinja. Mobilnu aplikaciju i korišteni postupak strojnog učenja treba ispitati na odgovarajućem skupu ulaznih podataka i prikladno analizirati.

2. KLASIFICIRANJE SLIKA

U ovom poglavlju bit će objašnjena ideja, napredak i izazovi računalnog vida, postupci računalnog vida zasnovani na strojnom učenju, te postojeća slična rješenja i ideja vlastitog rješenja.

2.1. Ideja računalnog vida

Često je podcijenjeno koliko je ljudski mozak napredan i moćan. Svakodnevna stvar poput prepoznavanje lica i stvari, kretnja kroz svijet i oko prepreka čine se jednostavnim na prvi pogled. Prema [1], znanstvenici su u području umjetne inteligencije 60-ih godina 20. stoljeća mislili da je računalni vid po težini na razini studentskog projekta. Tek četrdeset godina kasnije računalni vid počinje konkurirati ljudskom, ali samo u klasificiranju objekta na slici.

Ključ ljudskog, a time i računalnog vida je izdvajanje i prepoznavanje karakteristika slike, a ne slike kao cijelu. Ako model može uspješno prepoznat da neki oblik odgovara nosu, očima, mačjoj šapi ili slično, onda može znati koji se objekt, osoba ili životinja nalazi na slici na temelju pronađenih karakteristika.

2.2. Napredak računalnog vida kroz godine

Kao temelj računalnog vida, kao što članak [2] govori, može se smatrati istraživanje dva neurofiziologa, David Hubela i Torsten Wiesela, 1959. godine. Dalje, prema članku, pokušavali su stimulirati neurone mačke raznim slikama. Otkrili su da postoje jednostavni i složeni neuroni. Procesiranje slike u mozgu počinje u jednostavnim neuronima koji se aktiviraju na jednostavne oblike, poput rubova i linija. Složeniji neuroni onda slušaju jednostavne neurone i aktiviraju se kada su prisutne složenije karakteristike. Većina algoritama računalnog vida pokušava replicirati ovakvu strukturu. Prepoznavanje kreće od jednostavnih karakteristika, pa do sve složenijih. Sljedeći korak je bio 1959. godine, kada je Russel Kirsch izumio digitalni čitač slika, koji je omogućio prikaz slike u obliku binarnog koda. Nakon toga, 1963. godine, Lawrence Roberts je opisao postupak kojim je uspio dobiti 3D informacije o jednostavnim geometrijskim oblicima na 2D fotografiji te ih nakon toga prikazati u 3D prostoru. 1982. godine, Kunihiko Fukushima stvorio je neuronsku mrežu koja sadrži jednostavne i složene neurone, inspiriran radom Hubela i Wielsa. Mreža se zvala Neocognitron te je sadržavala nekoliko konvolucijskih slojeva. Ta mreža je možda prva neuronska mreža koja se može nazvati dubokom. Malo kasnije, 1989. godine, Yann LeCun je primijenio algoritam *backpropagation* nad arhitekturu neuronske mreže Neocognitron. Tu mrežu je nazvao LeNet-5, prva moderna konvolucijska neuronska mreža. Računalni vid je polako prestajao pokušavati rekreirati 3D model slike, te su se sve više istraživanja usredotočila na prepoznavanje slike putem karakteristika koje sadrži.

Započela je utrka u unapređivanju računalnog vida, osmišljene su baze podataka na kojim će se modeli uspoređivati. Prvo je to bio Pascal VOC, a nakon toga, te i dan danas, ImageNet, koji sadrži milijune slika podijeljene u tisuću klasa. [2]

2.3. Izazovi računalnog vida

Iako računalni vid i strojno učenje općenito znatno olakšava određene zadatke te omogućava veliku razinu automatizacije pojedinih postupaka, primijeniti metode u rješavanju svakodnevnih problema nije lagano. Glavni razlog tome je potreba za velikom količinom podataka kako bi stroj naučio raditi sa sličnim, ali novim i neviđenim podacima.

Kao primjer, MNIST [3] sadrži bazu podataka slika jednoznamenkastih brojeva dimenzija 28x28. Spomenuta baza podataka sadrži 60000 slika za učenje i 10000 slika za testiranje uspješnosti algoritma strojnog učenja. Najčešće se koristi za učenje klasifikacije jednoznamenkastih brojeva, što se smatra jednim od najosnovnijih zadataka za strojno učenje.

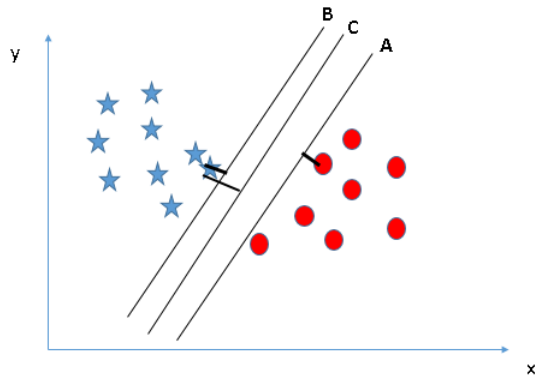
Brzina obradbe podataka je također važan faktor i izazov. Slabija računala često nisu ni u mogućnosti trenirati veće modele. Postupak treniranja je vrlo skup za RAM i CPU/GPU, pa tako treniranje naprednih modela može trajati tjednima. Brzina obradbe podataka je bitna i kod transformacije baze podataka, to jest grupe podataka prije nego ih model koristi. Takve transformacije mogu povećati preciznost modela, ili možda i ubrzati rad.

2.4. Postupci strojnog učenja u računalnom vidu

2.4.1 SVM

SVM (*eng. support vector machines)* se koristi za rješavanje problema koji imaju dva moguća rješenja. Prema [4], takva klasifikacija se još zove binarna klasifikacija.

Postoji dvodimenzionalni graf na kojemu se vrijednosti na X i Y osi kreću od 0 do 1. Na tom grafu se nalaze podaci koji pripadaju klasi A ili B. Idealnom primjerku klase A odgovara vrijednost (X=0, Y=1), a idealnom primjerku klase B vrijednost (X=1, Y=0). U ovom scenariju, a često i u stvarnom životu, podatci ne odgovaraju idealnim vrijednostima njihovih klasa. Tako se dođe do problema kako odvojiti ova dva podatka kada njihove vrijednosti mogu odstupati od idealne vrijednosti. Intuitivno bi bilo da se graf odvoji između dvije klase s pravcem $y=x$, ali je takav pristup pogrešan i često netočan. Ovakav graf je prikazan na slici 2.1.



Slika 2.1 Primjer idealnih pravaca. Pravac C ima najveću udaljenost do najbližeg podatka [5]

Dalje, prema [4], SVM upravo služi za pronalazak te idealne linije, a s time i prostor koji odvaja podatke iz te dvije klase. Idealna linija je ona koja odvaja podatke, ali je i najviše udaljena od najbližih točaka svake klase. To rezultira najvećoj točnosti klasifikacije. Nakon pronalaska idealne linije podatci se mogu klasificirati tako da ih se ucrta u graf i provjeri s koje strane linije se nalaze.

S obzirom na to da se kod SVM i ostalih metoda često koriste višedimenzionalni podatci, pronalazak idealne linije nije uvijek jednostavno kao što je na dvodimenzionalnom grafu. Iako je takav višedimenzionalan prostor nemoguće vizualizirati, princip pronalaženja te idealne linije ostaje isti. U slučaju da se podatci ne mogu odvojiti pravcem u prostoru u kojem se nalaze, postoje matematičke transformacije [4] koje mogu transformirati prostor na odgovarajući način, nakon čega ih se konačno može odvojiti pravcem.

Na kraju, prema [4], prednost ove metode je što ne zahtijeva puno podataka za pronalazak idealne linije, nakon čega je klasificiranje moguće. Potrebno je nekoliko graničnih vrijednosti klasa s pomoću kojih se spomenuta idealna linije pronađe. Jedan nedostatak je to što SVM samo govori kojoj klasi podatak pripada, a ne i vjerojatnost da pripada toj klasi.

2.4.2 Linearna Regresija

Linearna regresija je jedan od jednostavnijih algoritama strojnog učenja, zbog čega se često smatra početničkim algoritmom, od kojeg se kreće na kompliciranije algoritme. Kao što ime govori, radi se o modelu koji pokušava pronaći linearnu zavisnost između podataka. S obzirom na to da se radi o linearnom modelu, jednostavna jednadžba pravca (formula 2-1) dobro aproksimira ovaj algoritam.

$$y(x) = x_0 + kx \qquad (2-1)$$

S gore navedenom formulom 2-1 može se opisati pravac u n-dimenzionalnom prostoru ako se njeni elementi shvate kao matrice dimenzija [1, n]. Varijabla x je ulaz modela, dok je y(x) izlaz koji je linearno ovisan o x. Uz to se nalaze dvije konstante koje definiraju pravac. Konstanta x_0 opisuje translaciju pravca u koordinatnom sustavu, a konstanta k opisuje koliko promjena vrijednosti varijable x utječe na promjenu vrijednosti varijable y(x). Vrijednosti konstanta k i x_0 moraju se izračunati da se dobije pravac koji najviše odgovara podacima, takav pravac se još zove *best-fit* pravac. Postoje jednostavne metode u fizici i u analizi podataka koje služe za definiranje *best-fit* pravca u niže-dimenzionalnim prostorima, posebice dvodimenzionalnim i trodimenzionalnim prostorima.

Kako je ovo algoritam strojnog učenja, potrebno je ostvariti okolinu u kojem će računalo samo doći do točnih vrijednosti za konstante k i x_0. Za to je potreban način za opisivanje koliko je trenutna vrijednost izlaza modela udaljena od idealne vrijednosti izlaza modela, kao i način za opisivanje ovisnosti vrijednosti konstanti modela o vrijednosti izlaza modela. Jednostavnijim riječima, potreban je pokazatelj uspješnosti modela i potrebno je znati koliko promjena svake konstante doprinosi promjeni izlaznoj vrijednosti, nakon čega se izlazna vrijednost može promijeniti s određenim promjenama vrijednosti konstanta. Za opisivanje spomenute udaljenosti koristi se težinska funkcija (*eng. cost function*) prikazana na formuli 2-2 [6]. Težinska funkcija se najčešće računa kao zbroj kvadrata razlike vrijednosti dobivenog i očekivanog rezultata modela.

$$L(y_{pred},\ y) = (y_{pred} - y)^2 \qquad (2-2)$$

L – težinska funkcija

y_{pred} – matrica dobivenih vrijednosti na izlaznom sloju modela

y – matrica očekivanih vrijednosti

Za učenje modela koristi se algoritam postepenog silaska (*eng. gradient descent*). Algoritam postepenog silaska je zadužen za traženje minimuma težinske funkcije. Kada je težinska funkcija na minimumu model je u pravilu najprecizniji. Za algoritam postepenog silaska nužno je pronaći negativnu vrijednost gradijenta težinske funkcije, koji se može shvatiti kao smjer po kojem se treba kretati po težinskoj funkciji kako bi se vrijednost funkcije najviše smanjila. Za pronalazak gradijenta, tj. vektora ovisnosti izlazne vrijednosti o konstantama, koriste se parcijalne derivacije i lančano pravilo (*eng. chain rule*) prikazano na formuli 2-3 [7]. Pomoću lančanog pravila može se odrediti derivacija izlazne vrijednosti po svakoj konstanti modela.

$$\frac{dz}{dx} = \frac{dz}{dy} * \frac{dy}{dx} \qquad (2-3)$$

Lančano pravilo govori da je parcijalna derivacija varijable z po varijabli x, jednaka umnošku parcijalne derivacije varijable z po varijabli y i parcijalne derivacije varijable y po varijabli x.

Elementi izračunatog gradijenta poslije se samo zbroje s odgovarajućim konstantama modela, te se taj postupak iterativno ponavlja. Ako se gradijenti zbroje međusobno nakon svake iteracije, *n* broj puta te ih se nakon *n* broja puta zbroji s konstantama modela, dobije se algoritam stohastičkog postepenog silaska (*eng. stochastic gradient descent*), koji znatno smanjuje potrebni broj izvršenih operacija. Parametar *n* predstavlja veličinu grupe (*eng. batch size*) Nakon završetka postupka učenja, model bi trebao sa zadovoljavajućom sigurnošću točno odrediti linearnu zavisnost izlazne vrijednosti o ulaznoj vrijednosti.

2.4.3 Logistička regresija

Logistička regresija je algoritam strojnog učenja vrlo sličan algoritmu linearne regresije. Kao što članak [8] govori, razlika između dvoje je uporaba različitih funkcija kod izračuna očekivane vrijednosti izlaza te težinske funkcije. Logistička regresija se koristi kada se vrijednosti izlaza nalaze između nula i jedan, što se često koristi kod binarne klasifikacije, ali je moguće koristiti i kod više-klasne klasifikacije. Sam naziv funkcije, a i razlog zašto daje izlaz od nula do jedan, leži u logističkoj krivulji, još zvana sigmoid funkcija, prikazana na formuli 2-4 [9].

$$S(x) = \frac{1}{1 + e^{-x}} \qquad (2-4)$$

S – sigmoid ili logistička funkcija

x – ulaz funkcije

Logistička funkcija sažme sve vrijednosti u raspon od nula do jedan. Velikim negativnim vrijednostima se dodjeljuju vrijednosti blizu nule, a velikim pozitivnim vrijednostima vrijednosti blizu jedan. Područje prijelaza funkcije se otprilike nalazi između -4 i 4, te je poželjno da vrijednosti budu u tom rasponu.

Dakle, razlika kod izračuna očekivane vrijednosti algoritma logističke regresije leži u tome da se funkcija pravca koja se koristi u algoritmu linearne regresije dalje provede kroz logističku funkciju. Težinska funkcija se onda također mijenja te se njezin iznos dobije tako da se težinska funkcija (formula 2-2) opisana u linearnoj regresiji provede kroz inverz logističke funkcije (formula 2-4).

2.4.4 Naivni Bayes

Naivni Bayes algoritam je strojnog učenja temeljen na Bayesovom teoremu, prikazanom na formuli 2-5 [10]. Bayesov teorem govori kako izračunati vjerojatnost događaja A ako se dogodio događaj B.

$$P(A \mid B) = \frac{P(B \mid A) * P(A)}{P(B)} \qquad (2-5)$$

P(A | B) je uvjetna vjerojatnost, vjerojatnost da će se dogoditi događaj A ako se dogodio događaj B, i suprotno za P(B | A). P(A) je vjerojatnost događaja A, a P(B) je vjerojatnost događaja B.

Naivni Bayes zove se naivni zbog njegovog načina rada. Prema [11], Naivni Bayes pretpostavlja da su sve pretpostavke neovisne jedna o drugima, što znači da vjerojatnost jedne pretpostavke neće utjecati na vjerojatnost druge pretpostavke. Iako se ovaj pristup s razlogom zove naivnim jer u stvarnom životu ta pretpostavka skoro nikad nije točna, rezultati koje daju su usporedivi s puno kompliciranijim algoritmima. Ta jednostavnost također rezultira puno bržim i efikasnijim radom, gdje se broj operacija aproksimira s O(n), za razliku od $O(2^n)$ za normalni Bayesov algoritam.

Algoritam naivnog Bayesa ne uči se iterativno, kao mnogi ostali algoritmi. Računalo samo mora iz odgovarajućih podataka izračunati vjerojatnosti svake pretpostavke. Nakon toga se vjerojatnost izlazne pretpostavke računa ovisno o izračunatim vjerojatnostima ulaznih pretpostavka.

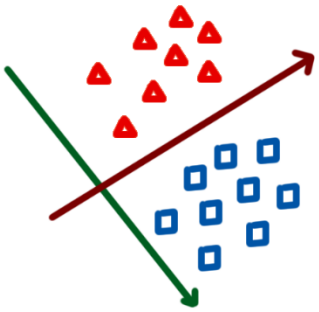
Postoje tri tipa algoritma naivni Bayes, s različitim oblikom rezultata, kao što članak [12] govori. Prvo, postoji Polinomski (*eng. multinomial*) koji daje klasu kao rezultat i govori kojoj klasi ulazne vrijednosti pripadaju. Drugo, postoji Bernoullijev koji kao ulaz i izlaz koristi *boolean* varijable. Zadnje, postoji Gaussov koji prima i daje neprekidne vrijednosti na Gaussovoj krivulji.

Naivni Bayes se uglavnom koristi kod analize dokumenata i filtera neželjenih poruka [12]. Zbog jednostavnosti je brz i lagan za implementaciju, ali se mora obratiti pozornost gdje se koristi jer nije svaka situacija prikladna za njegovu uporabu.

2.4.5 Linearna analiza diskriminante

Prema [13], linearna analiza diskriminante algoritam je koji se često koristi za smanjivanje dimenzija podataka. Smanjivanje dimenzija ima svrhu znatnog smanjivanja količine podataka s kojim se radi, što rezultira efikasnijim i bržim radom. Velik broj potrebnih operacija je uvijek problem u strojnom učenju što čini ovaj algoritam vrlo korisnim. Smanjivanje dimenzija radi se tako da se pronađe nova, bolja os s kojom se podaci mogu podijeliti. Nakon podjele, svi podaci

na novoj osi unutar jedne klase trebali bi se nalaziti blizu ostalim članovima iste klase. Podjela je prikazana na slici 2.2.



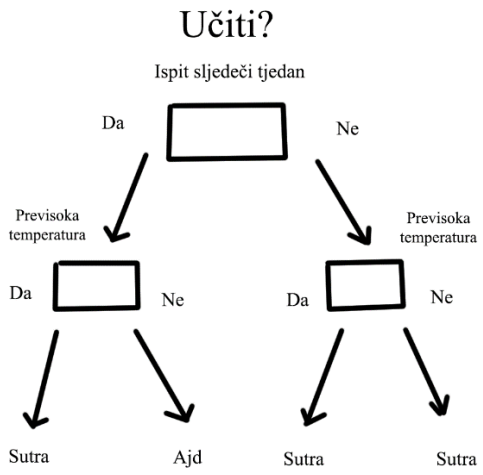
Slika 2.2 Podjela dvodimenzionalnih podataka na jednodimenzionalnoj osi. Crvena os je primjer loše podjele podataka, dok je zelena os primjer dobre podjele

Za pronalazak nove osi prvo se izračuna vrijednost srednje udaljenosti između klasa. Nakon toga se izračuna udaljenost između vrijednosti srednje udaljenosti klase i pojedinog podatka unutar klase. Zadnji korak je konstruiranje osi koja maksimizira udaljenost među klasama i minimizira udaljenost podataka unutar klase. Zbog jednostavnosti algoritma linearne analize diskriminante postoji mnogo sličnih algoritama koji služe kao proširenja [13] , kao što su kvadratna analiza diskriminante i generalizirana analiza diskriminante.

2.4.6 Stabla odlučivanja

Prema [14], stablo odlučivanja je algoritam korišten u strojnom učenju kao vrlo sposobna metoda za klasificiranje i odlučivanje. Sama struktura stabla odlučivanja je intuitivna i može ju interpretirati i osoba koja nije upoznata sa samim algoritmom i strojnim učenjem općenito. Razlog tome je njena binarna struktura koja je bliska načinu na kojem sami ljudi donose odluke.

Binarno stablo se sastoji od korijena na vrhu, te se razgranjuje na dvije odluke koje se najčešće suprotstavljaju. Donošenjem odluka i odgovaranjem na upite koje stablo postavi kreće se niz stablo sve do njegovog kraja, gdje se nalaze listovi. Listovi su krajnji dijelovi stabla te se na njima krije informacija o klasama i odlukama stabla. Primjer stabla je prikazan na slici 2.3.



Slika 2.3 Primjer binarnog stabla odlučivanja

Stablo se konstruira tako da se uzme značajka koja ima najmanju težinu, te se ona odabere kao korijen stabla. Dalje se pronalaze značajke u skupu unesenih podataka koje su povezani s korijenom, ili roditeljskim čvorom, te se stvori novi čvor razinu niže. Taj postupak se ponavlja rekurzivno sve dok su svi krajnji čvorovi listovi. Postupak kreiranja stabla zove se indukcija [14] (*eng. induction*)

Veliki problem kod stabla odlučivanja je pretreniranje stabla, što znači da se ono previše prilagodilo skupu unesenih podataka. Kad se to dogodi onda stablo sadrži čvorove koji sadrže upite koji su vrlo specifični, ili su beskorisni za donošenje konačne odluke. Pretreniranje se može izbjeći tako da se postavi donja granica za broj potrebnih podatka s istom značajkom prije nego se čvor kreira. Tako se izbjegnu upiti o značajkama koje su rijetke. Drugi način je da se ograniči broj razina stabla. Kad stablo dosegne određenu razinu čvorovi se više neće širiti. S ovim parametrima može se konstruirati stablo koje uspješno generalizira podatke umjesto da se njima prilagodi. U slučaju da je stablo pretrenirano nakon njegova stvaranja, postoji algoritam koji prolazi kroz stablo i čisti nepotrebne čvorove. Taj algoritam se zove obrezivanje stabla [14] (*eng. pruning*). Jednostavna metoda za obrezivanje stabla se izvršava prolaženjem kroz stablo i brisanjem određenog čvora. Ako se težinska funkcija ne promjeni, može se zaključiti da je obrisani čvor nepotreban.

2.4.7 Algoritam k-najbliži susjed

Algoritam k-najbliži susjed (*eng. K-nearest neighbour*) prvi je nenadgledani (*eng. unsupervised*) algoritam strojnog učenja koji se spominje u ovom radu. Svi ostali spomenuti algoritmi su nadgledani (*eng. supervised*). Nadgledani algoritmi se ispravljaju za vrijeme učenja te ih se

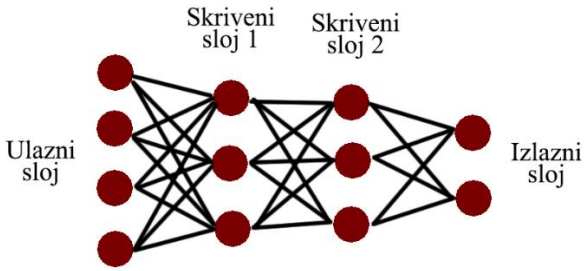
usmjeruje prema željenom rezultatu. Takvi algoritmi se koriste kod klasifikacije ili regresije. S druge strane, nenadgledani algoritmi se ne nadziru tijekom učenja. Nenadgledani algoritmi mogu naučiti samo strukturu podataka. Zbog toga oni ne mogu direktno naučiti s kakvim podatkom rade, ali mogu saznati o kakvom se podatku radi ako ga usporede sa sličnim podatkom za kojeg se zna kojoj klasi pripada.

Dakle, algoritam k-najbliži susjed može otkriti strukturu podataka. Također može pronaći podatke u bazi podataka koje su slične unesenom podatku. Zbog tih razloga često ga se koristi u sklopu internet stranica [15] [16] u svrhu preporuka sadržaja korisniku. Algoritam sličan k-najbližem susjedu se koristi kada internet stranica preporučuje sadržaj poput igara, filmova, pjesmi i ostalih sadržaja koji su označeni oznakama koje govore o njihovim žanrovima.

Algoritam k-najbliži susjed se ne koristi kod velikih baza podataka jer mu učinkovitost i brzina znatno opada s povećanjem baze podataka na kojoj se primjenjuje. Za takve slučajeve koristi se modifikacija tog algoritma, ali je princip sličan. Jedan od načina na koji se može znatno ubrzati pretraga baze podataka je napredno indeksiranje redova.

2.4.8 Neuronske mreže

Neuronske mreže, punim imenom umjetne neuronske mreže, algoritmi su strojnog učenja modelirani po biološkim mrežama neurona koje se nalaze u ljudskom mozgu, po čemu je algoritam nazvan. Arhitektura jednostavne neuronske mreže se može vidjeti na slici 2.4.



Slika 2.4 Shema jednostavne neuronske mreže

Neuronske mreže su među novijim algoritmima strojnog učenja. Iako su postojale i osamdesetih godina prošlog stoljeća, njihov napredak bio je spor zbog sporih i slabih računala. Neuronske mreže su zaživjele s napretkom sklopovlja računala te su danas prisutne u skoro svakoj grani industrije. Algoritmi modelirani na neuronskim mrežama daleko su najprecizniji algoritmi u obitelji algoritama strojnog učenja, pa zbog tog danas postoji računalna klasifikacija slika koja je

uspješnija od ljudi, postoje auti koji se sami voze i mnogi ostali sustavi koji svaki dan nadmašuju ljude.

Kad se spominju neuronske mreže danas, uglavnom se misli na duboke neuronske mreže koje po definiciji imaju dva ili više sakrivenih slojeva u modelu neuronske mreže. Većina neuronskih mreži danas imaju puno više slojeva tako da se većina mogu smatrati dubokim neuronskim mrežama. Trenutno je jedna od najproširenijih vrsta neuronskih mreža konvolucijska neuronska mreža, ali ima i puno drugih vrsta koje pronalaze svoju primjenu. Konvolucijske neuronske mreže se nalaze svuda gdje se radi sa slikama. To su spomenute klasifikacije slika i auti koji se sami voze, pa do detekcije raka i ostalih bolesti.

Konvolucija je rješenje za učinkovito smanjivanje dimenzije slike uz održavanje bitnih značajki slike. Sam algoritam nije kompliciran. Ukratko, konvolucija koristi kvadratne filtere koji prolaze preko slike te zbrajaju vrijednosti ispod. S time se slika sažima bez eliminiranja temeljnih značajka slike. Broj parametara u konvolucijskom sloju linearno je proporcionalan veličini filtera i broju filtera, dok je broj parametara između dva gusta sloja umnožak broja njihovih neurona. Zbog tog razloga učenje konvolucijskog sloja traje daleko kraće od učenje gustog sloja neurona. Dodatni razlog zašto su neuronske mreže danas toliko popularne među inženjerima i onima koji to žele postati je njena jednostavnost. Sam temeljni algoritam neuronskih mreža je jednostavan za shvatiti i matematički ga je lagano implementirati bez dodatnih biblioteka. Jednostavna se neuronska mreža može implementirati u većini programskih jezika koristeći for petlje [17]. Uz to, unaprijed naučeni modeli [18] dostupni su svima. Tako se otklanja možda najveći problem neuronskih mreža; ogromna količina računalnih resursa potrebnih za učenje. Tehnika dodavanja vlastitih slojeva na vrh ranije istreniranih modela zove se prijenos učenja (*eng. transfer learning*), te se svodi na to da se od unaprijed naučene neuronske mreže preuzme dio odgovoran za prepoznavanje značajka ulaznih podataka, ili konvolucijski dio, pa se na vrh konvolucijskog dijela doda nekoliko gustih slojeva neurona (*eng. dense layer*) koji uparuju značajke s odgovarajućim neuronima na izlazu.

Najčešće korištene aktivacijske funkcije neuronskih mreža su logistička funkcija i ispravljena linearna funkcija (*eng. rectified linear unit, ReLU*). Težinska funkcija koja se često koristi je jednostavna funkcija spomenuta kod linearne regresije(formula 2-2). Kao funkcija za optimizaciju često se koristi algoritam postepenog silaska koji je također spomenut, ali se koriste i učinkovitiji algoritmi kao Adam [19] koji prilagođuje brzinu učenja, tj. usporava faktor učenja kako se težina mreže približava minimumu težinske funkcije.

2.5. Postojeća rješenja računalnog vida zasnovana na postupcima strojnog učenja

2.5.1 LeNet

LeNet [20] konvolucijsku neuronsku mrežu napravili su Yann LeCun, Leon Bottou, Yoshua Bengio i Patrick Heffner te se može smatrati prvom konvolucijskom neuronskom mrežom. Prije konvolucijske neuronske mreže LeNet i prije konvolucijskih slojeva, posao klasificiranja slika većinom su odrađivali gusti slojevi. Efektivnost takvih neuronskih mreža je bila zadovoljavajuća za jednostavne zadatke poput klasificiranja izoliranih znamenki na slici dimenzija 28x28, ali za kompliciranije slike efektivnost je takvih neuronskih mreža drastično opadala. Tako je konvolucijska neuronska mreža LeNet dobila puno bolje rezultate, koristeći nekoliko slojeva konvolucije i prosječnog udruživanja [20] (*eng. Average pooling*). Nakon tih slojeva dodan je sloj za ravnanje rezultata konvolucije te je nakon toga dodano nekoliko gustih slojeva. LeNet-5, jedna inačica konvolucijske neuronske mreže LeNet, koristila je dva para konvolucijskih slojeva i dva gusta sloja.

2.5.2 AlexNet

AlexNet [21] napravili su Alex Krizhevsky, Ilya Sutskever te Geoffrey E. Hinton na sveučilištu u Torontu. Kao što piše u izvornom AlexNet radu [21], AlexNet je konvolucijska neuronska mreža namijenjena klasificiranju slika iz baze podataka ImageNet. Jedna je od najutjecajnijih konvolucijskih neuronskih mreža zbog ogromnog uspjeha na ILSVRC 2012 (*eng. ImageNet Large Scale Visual Recognition Competetion*), s mjerom pogreške od 15%, što je ogroman uspjeh u odnosu na drugoplasiranog, koji je postigao mjeru pogreške od 25%. Iz tog uspjeha došlo se do zaključka o izrazitoj važnosti dubine konvolucijske neuronske mreže za njenu preciznost, s obzirom na to da je AlexNet s pet konvolucijskih i tri gusta sloja najdublja konvolucijska neuronska mreža do tad. Proširivanje i dodavanje slojeva znatno povećava broj parametara mreže, što onda povećava potrebno vrijeme za učenje mreže. Zbog toga se povećavanje dubine neuronske mreže prije nije vidjelo kao dobro rješenje za povećanje njene preciznosti. Tim iza konvolucijske neuronske mreže AlexNet riješio je taj problem korištenjem grafičke kartice (dvije) za učenje neuronske mreže umjesto procesora koji se do tad većinom koristio. Uz par optimizacija kod računanja, grafička kartica mogla je puno brže odrađivati računske operacije te je znatno ubrzala učenje neuronske mreže. Jedna od optimizacija je korištenje ReLU aktivacijske funkcije umjesto dotad često korištenih tangens ili logističkih aktivacijskih funkcija. Prema [21], korištenje ReLU aktivacijske funkcije na četveroslojnoj konvolucijskoj neuronskoj mreži rezultira šest puta bržem

dostizanju mjere pogreške od 25%, u odnosu na ekvivalentnu mrežu s tangens aktivacijskom funkcijom

Dalje, iz izvornog rada [21], zbog svoje veličine i broja parametara, ova konvolucijska neuronska mreža sama po sebi je sklona pretreniranju. Za taj problem predstavljena su dva glavna rješenja. Prvo, baze podataka slika je proširena (*eng. data augmentation*) s blagim promjenama na slikama. Prije nego su slike predstavljene prvom konvolucijskom sloju, one se nasumično odrežu na određenu dimenziju (slika od 256x256 se odreže na 224x224). Uz to, RGB vrijednosti slike se blago promjene. Drugo, nad gustim slojevima mreže uvedeno je ispadanje neurona (*eng. dropout*). To ispadanje radi tako da svaki neuron ima vjerojatnost od 50% da će biti ugašen tijekom jedne iteracije učenja. Neuroni neuronske mreže naučeni ovom metodom ne mogu naučiti da ovise o susjednim neuronima, te se zbog toga pretreniranje znatno smanji. Vrijeme konvergencije (ili učenja) neuronske mreže zbog faktora ispadanja neurona od 0.5 traje dvaput duže.

2.5.3 ZFNet

ZFNet [22] napravili su Dr. Rob Fergus i Dr. Mathhew D. Zeiler 2013 godine. Iako nije pobjednik na ILSVRC 2012, donio je veliku inovaciju za područje računalnog vida te dodatna poboljšanja nad konvolucijskoj neuronskoj mreži AlexNet.

Koristeći dekonvoluciju, Zeiler i Fergus uspjeli su dobit podatke iz konvolucijskih filtera te su s time stvorili vizualizaciju kako prikazuje kako računalo i konvolucijski slojevi vide ulazne slike. Saznali su da konvolucijski sloj procesira sliku slično ljudskom mozgu. Kao i ljudski mozak, saznao je da konvolucijski slojevi prvo prepoznaju rubove, pa onda prepoznaju jednostavnije oblike, nakon toga prepoznaju uzorke, onda prepoznaju manje objekte (poput automobilskih guma, lica, ruka i slično), te na kraju prepoznaju cijele objekte (poput auta, životinja, aviona i slično).

2.5.4 Inception

Inception [23] je konvolucijska neuronska mreža tvrtke Google koja je pobijedila ILSVRC 2014 s mjerom pogreške od 6.67%. Prema [23], glavna inovacija konvolucijske neuronske mreže Inception su takozvani Inception blokovi. Inception blokovi se temelje na saznanju prijašnjih radova koja tvrde da prorijeđene (*eng. sparse*) arhitekture konvolucijskih neuronskih mreža rezultiraju većom preciznošću u usporedbi s preciznošću gustih arhitektura. Prorijeđene arhitekture bile su korištene i prije, ali su se sve više počele koristiti guste arhitekture zbog sve više optimiziranih biblioteka koje vrše računanja nad gustim matricama. Ideja Inception bloka je spojiti prednosti prorijeđene i guste arhitekture. Ulaz Inception bloka se dijeli na četiri paralelna sloja čiji se rezultati kasnije udružuju u jedan vektor te se predaje izlazu Inception bloka. Vrste

paralelnih slojeva su 1x1 konvolucija, 3x3 konvolucija, 5x5 konvolucija te 3x3 maksimalno udruživanje (*eng. max pooling*). Slojevima osim 1x1 konvolucije je dodana dodatna 1x1 konvolucija koja služi smanjivanju dimenzija podatka.

Glavni fokus konvolucijske mreže Inception je optimizacija konvolucijske neuronske mreže smanjivanjem parametara i broja matematičkih operacija, zato što je procesorska snaga ograničena. Prema [23], optimizacija algoritama konvolucijske neuronske mreže je rješenje za njihovo unaprjeđivanje, a ne razvoj boljih računala koja mogu raditi s većim konvolucijskim neuronskim mrežama.

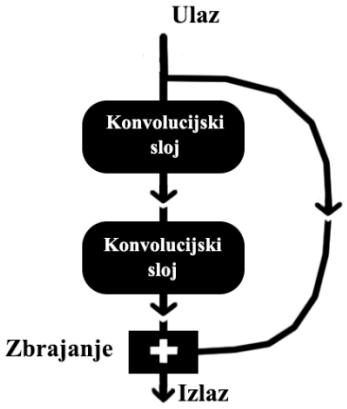
2.5.5 VGG

VGG [24] je konvolucijska neuronska mreža koju su razvili Karen Simonyan i Andrew Zisserman na sveučilištu u Oxfordu. Njihova glavna ideja je da konvolucijske mreže trebaju biti puno dublje nego su dotad bile, što je ideja slična ideji tima iza konvolucijske mreže Inception. Za razliku od konvolucijske neuronske mreže Inception, VGG je sličniji tada klasičnom obliku mreže. Druga ideja vezana za arhitekturu mreže je isključivo korištenje konvolucijskih slojeva s 3x3 veličinom filtera. Originalni rad iza VGG-a [24] objašnjava da je 3x3 veličina filtera minimalna veličina filtera koja može „vidjeti" horizontalne i vertikalne crte. Također, došli su do zaključka da je bolje koristiti više slojeva s 3x3 filterom nego jedan 5x5 ili 7x7 filterom. Iako dva 3x3 konvolucijska sloja jednako smanjuju dimenziju podatka kao jedan 5x5, te tri 3x3 kao jedan 7x7, pokazalo se da više 3x3 slojeva rezultiraju višom preciznošću, kao i manjim brojem parametara konvolucijske neuronske mreže. Zadnje, kod učenja mreže, nasumično se mijenja veličina ulazne slike od 256 do 512 piksela prije nego li se izreže da se prilagodi 224x224 ulazu, što više prilagođuje mrežu raznim slikama. Sve u svemu, ova arhitektura se pokazala obećavajućom te je VGG pobijedio na ILSVRC 2014 u kategoriji klasifikacije slike.

2.5.6 ResNet

Kaiming He, Xiangyu Zhang, Shaoqing Ren i Jian Sun članovi su Microsoftovog tima koji je razvio konvolucijsku neuronsku mrežu ResNet [25], koja je bila dominantna na ILSVRC 2015. Dolaze sa sličnom idejom kao tim iza konvolucijske neuronske mreže VGG, pobjednici ILSVRC 2014. Prema [25], vjeruju da je ključ boljih konvolucijskih neuronskih mreža u njihovoj dubini, ali pokazalo se da povećavanje dubine u uobičajenim konvolucijskim neuronskim mrežama poslije određenog broja slojeva vodi do veće pogreške. Taj su problem uspjeli riješiti na jednostavan način, ali s impresivnim rezultatima. Gradivni blokovi konvolucijske neuronske mreže ResNet

sastoje se od serijskog spoja nekoliko (uglavnom tri) konvolucijskih slojeva, ali se rezultat konvolucije poslije zbraja s ulazom u blok, te čini izlaz bloka, kao što se vidi na slici 2.5.



Slika 2.5 Izgled ResNet bloka [25]

Takva jednostavna arhitektura rješava problem opadanja preciznosti prisutan u uobičajenim konvolucijskim neuronskim mrežama. Tako je omogućena izvedba konvolucijske neuronske mreže ResNet sa 152 sloja, zvana ResNet-152. Konvolucijska neuronska mreža ResNet-152, kao pobjednik ILSVRC 2015, postigao je mjeru greške od 4.49% u top-5 kategoriji, a koristeći više konvolucijskih neuronskih mreža ResNet zajedno dostižu rezultat od 3.57%. Zanimljivo, ResNet-152 (11.3 milijarde FLOPs) je i dalje jednostavniji od VGG-16/19 (15.3/19.6 milijarde FLOPs). [25]

2.5.7 MobileNetV2

MobileNetV2 [26] je moderna konvolucijska neuronska mreža zamišljena za rad na mobilnim uređajima. Rad na mobilnim uređajima podrazumijeva rad s vrlo ograničenom pohranom i snagom procesora. Zbog toga je MobileNetV2 primarno usmjeren na povećavanje učinkovitosti konvolucijske neuronske mreže smanjivanjem broja parametara i potrebnih kalkulacija. Te mjere smanjivanja broja parametara i kalkulacija uzrokuju bržem odazivu mreže.

MobileNetV2 [26] temelji svoj dizajn i arhitekturu na inovacijama konvolucijskih neuronskih mreža koje su nastale prije, ali donosi i nekoliko svojih inovacija. Po uzoru na konvolucijsku neuronsku mrežu ResNet (poglavlje 2.5.6) ulaz gradivnih blokova MobileNetV2 mreže spojen je na njihov izlaz, što dokazano povećava preciznost neuronske mreže. Inovacija koju donosi su *depthwise* konvolucije, koje vrše konvoluciju po svakoj dimenziji ulaznog podatka zasebno. [27]

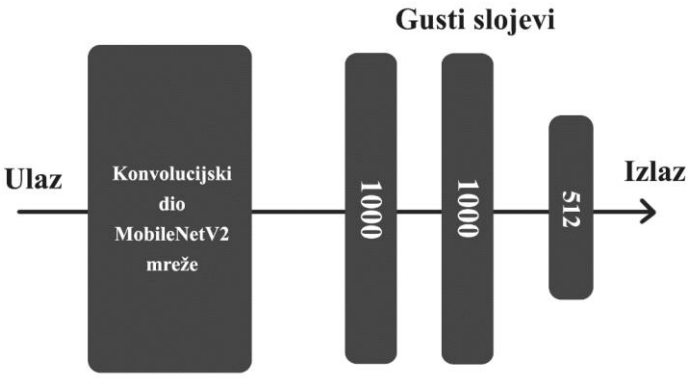
2.6. Ideja vlastitog rješenja

Zadatak ovog rada je izraditi aplikaciju koja mora zadovoljavati sljedeće kriterije. Prvo, aplikacija mora sadržavati ugrađenu funkcionalnost za upravljanje kamerom. Ugrađena kamera mora omogućiti slike rezolucije veće ili jednake veličini ulaza neuronske mreže, kako bi se umanjio negativni utjecaj slika niske kvalitete na učinkovitost neuronske mreže. Drugo, aplikacija mora uspješno klasificirati životinje prikazane na slikama koje su dane neuronskoj mreži putem kamere. Željena preciznost je 90% i više. Neuronska mreža mora dati odgovor najkasnije u 5 sekundi. Treće, aplikacija mora sadržavati bazu podataka koja sadržava podatke o životinjama, kao što je ime životinje, njezina slika, kratki opis i najveći postotak točnosti (ili aktivacija neuronske mreže). Slike životinja se uzimaju od slika koje korisnik snimi kamerom, te se koristi slika s najvećim postotkom točnosti. Opis slika se mora dohvatiti s Internet stranice, kao što je internet stranica Wikipedia [28].

Neuronske mreže su se pokazale kao najbolji algoritam strojnog učenja za zadovoljavajuće rješavanje ovoga zadatka. Konvolucija se pokazala izrazito dobra kod klasifikacije slika, što je točno ono što je potrebno za klasifikaciju 51 klase životinja.

Nakon brojnih eksperimenata pokazalo se da je metoda prijenosa učenja (*eng. transfer learning*) najjednostavnije rješenje. Većim djelom jer je veliki problem nabaviti računalo potrebno za učenje. Jedna mogućnost je da se postupak učenja odvija na oblaku, ali takve usluge nisu besplatne, pa je metoda prijenosa učenja idealno rješenje. Kod odgovoran za učenje takve mreže je objašnjen u poglavlju 4.1.

Za temelj mreže uzet je MobileNetV2, koja je kratko objašnjena u poglavlju 2.5.7. Kao što samo ime govori, zamišljen je za rad na mobilnim uređajima, gdje se moraju uzeti kompromisi zbog slabijeg procesora i grafičke kartice. MobileNetV2 se pokazao najbržim za Android [29]. Na vrh konvolucijskih slojeva MobileNetV2 mreže dodani su gusti slojevi. Arhitektura korištene konvolucijske neuronske mreže može se vidjeti na slici 2.6.



Slika 2.6 Arhitektura korištene neuronske mreže

Kao bazu podataka slika na kojima će se konvolucijska neuronska mreža trenirati uzela se baza podataka Caltech-256 slika [30]. Kategorije koje se ne odnose na životinje su obrisane, a dodatne kategorije se mogu dodati ako se želi proširiti uporaba modela.

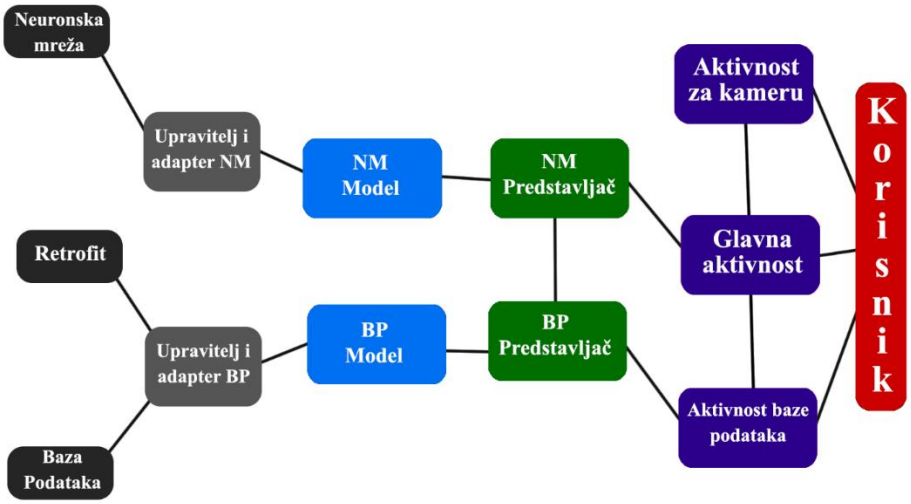
Za implementaciju baze podataka koristi se biblioteka SQLite koji je objašnjen u poglavlju 3.2.5. Baza podataka je oblikovano slično tablici 2.1.

Ime	Slika	Opis	Datum kad je viđen	Postotak točnosti	Koliko puta viđen
Životinja A	SlikaA	Životinja A	01/01/1000	0.0	0
Životinja B	SlikaB	Životinja B	01/01/1000	0.0	0

Tablica 2.1 Oblik baze podataka

Biblioteka Retrofit korištena je za dohvaćanje opisa životinje. Opisi su dohvaćeni s članaka odgovarajućih životinja na internet stranici Wikipedia [28] koji odgovaraju određenoj životinji. Više informacija o biblioteci Retrofit se nalazi u poglavlju 3.2.6.

Aplikacija je zasnovana na arhitekturnom predlošku MVP (model-pogled-predstavljač, *eng. model-view-presenter*) [31]. Arhitekturni predložak MVP poznat je u razvoju Android aplikacija jer je kod koji poštuje pravila takve arhitekture čitljiviji i lakši za nadograđivanje. Dodatno, svaka cjelina takvog koda može se zasebno testirati, što je vrlo značajno kod složenih aplikacija. Blok dijagram arhitekture aplikacije je prikazan na slici 2.7.



Slika 2.7 Blok dijagram arhitekture aplikacije

Kratica **NM** na slici 2.6 predstavlja neuronsku mrežu, a kratica **BP** predstavlja bazu podataka. Dalje, blokovi **upravitelj i adapter NM/BP** označuju sve klase i funkcije koje upravljaju ili su sučelje za neuronsku mreži ili bazu podataka. Za sučelje neuronske mreže se koristi biblioteka TFLite koja je objašnjena u poglavlju 3.3.1, a za sučelje baze podataka korištena je spomenuta biblioteka SQLite.

Sve aktivnosti se mogu smatrati pogledima, s obzirom na to da je aplikacija zasnovana na arhitekturnom predlošku MVP. Pogledi su odgovorni za predstavljanje sadržaja koje korisnik vidi. Modeli su odgovorni za dohvat podataka i rukovanju s njima. Predstavljači upravljaju tijekom aplikacije, te vrše komunikaciju između pogleda i modela.

3. KORIŠTENE TEHNOLOGIJE I ALATI

U ovom poglavlju bit će dan opis korištenih alata, kao i objašnjenje njihove uporabe u kontekstu ovog završnog rada.

3.1. Korištene tehnologije, alati i programski jezici u klasificiranju slika

3.1.1 Python

Python [32] je objektno orijentirani, interpretirani, opće-namjenski programski jezik visoke razine. Koristi dinamičnu semantiku (*eng. dynamic semantics*), dinamičnu provjeru tipa podatka (*eng. dynamic typing*), te dinamično povezivanje koda (*eng. dynamic binding*). Zajedno s ugrađenim podatkovnim strukturama visoke razine, čine ovaj jezik pogodnim za brzo razvijanje aplikacije (*eng. rapid application development*).

Programski jezik Python dobio je ogromnu popularnost nedavno, kao što se vidi na tablici 3.1. Predstavljen je kao programski jezik koji je jednostavan za naučiti te jednostavan za pisati, tj. kodirati. Samim time je idealan programski jezik za početnike, pa svako tko danas želi naučiti programirati često kreće od programskog jezika Python. Učenje ovog programskog jezika je dodatno olakšano zbog brojnih tečajeva dostupnih na internetu.

Rang	Programski jezik	Udio korištenja [%]	Promjena u odnosu na prošlu godinu [%]
1	Python	29.21	+ 4.6
2	Java	19.9	- 2.2
3	JavaScript	8.39	+ 0.0
4	C#	7.23	- 0.6
5	PHP	6.69	- 1.0
6	C/C++	5.8	- 0.4
7	R	3.91	- 0.2
8	Objective-C	2.63	- 0.7
9	Swift	2.46	- 0.3
10	Matlab	1.82	- 0.2

Tablica 3.1 Popularnost programskih jezika temeljena na broju pretraga na Google [33]

Razlog velikom skoku u popularnosti jednim je dijelom ogromni rast popularnosti strojnog učenja među znanstvenicima, inženjerima, tvrtkama, a i studentima i ostalim ljudima koje je ova znanstvena disciplina privukla.

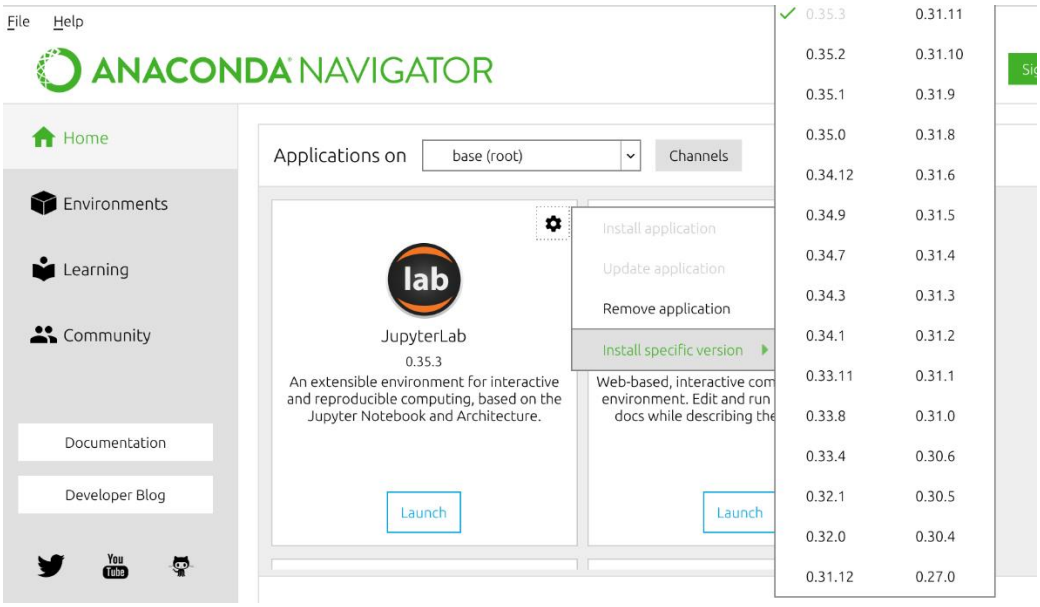
Za programski jezik Python karakteristična je velika preglednost koda i brojne gotove funkcije koje olakšavaju rad. Vjerojatno je najkontroverznija promjena u programskom jeziku Python, u odnosu na ostale programske jezike, nedostatak vitičastih zagrada. To znači da broj razmaka,

odnosno uvlačenje teksta postaje dio sintakse. Iako ovo ima smisla vizualno, upitno je, je li zapravo bolje. Ono što je sigurno da mnogima nedostaju vitičaste zagrade.

S obzirom na to da se većina posla odrađuje za vrijeme izvođenja, programski jezik Python nikada neće biti brz i optimiziran kao programski jezici Java ili C. Jednostavnost korištenja i općenito kraći kod čini ovaj programski jezik idealan za prototipiranje određenih funkcija, koje se kasnije prenesu u programski jezik C++, gdje se puno brže izvode. Zbog toga se često koristi kao jezik koji spaja funkcije napisane u bržim programskim jezicima. Jedan primjer je biblioteka TensorFlow koja će biti dodatno objašnjena u poglavlju 3.1.4. TensorFlow je biblioteka za programski jezik Python namijenjena za implementaciju algoritama strojnog učenja. Sve njegove funkcije se pozivaju u programskom jeziku Python, ali su sve te funkcije zapravo napisane u programskom jeziku C++.

3.1.2 Anaconda

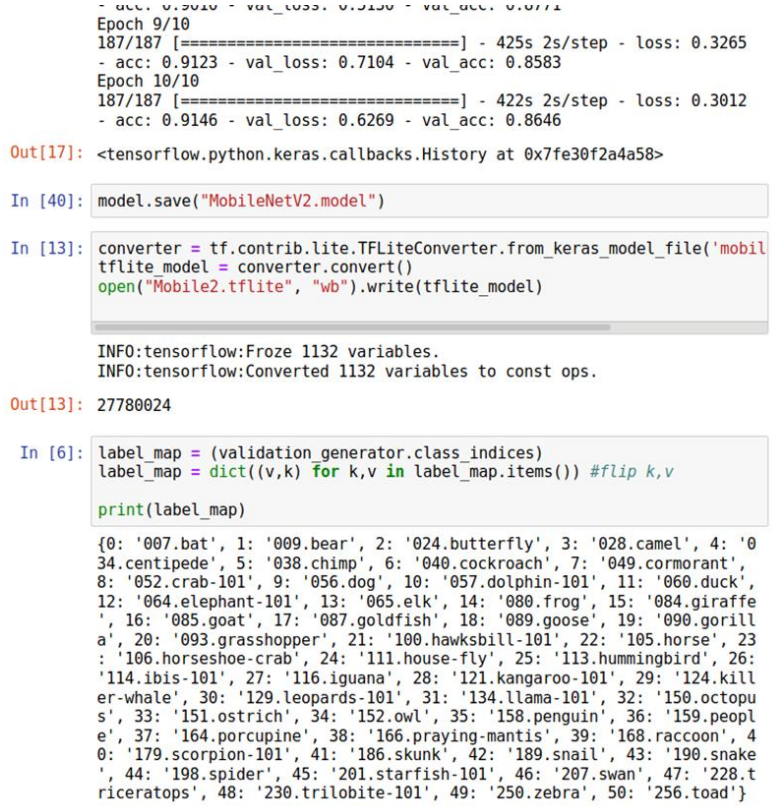
Anaconda je upravitelj paketa, upravitelj okoline, Python/R distribucija podatkovnih znanosti te skup od preko 1500 paketa otvorenog koda [34]. Anaconda se može pokrenuti kao **conda** unutar Anaconda Prompt za operacijski sustav Windows ili Terminal za operacijske sustave Linux i MacOS. Također postoji i grafičko korisničko sučelje zvano Anaconda Navigator, koji znatno olakšava rad korisniku koji nije upoznat s radom u konzolama. Zbog velikog broja paketa, zbog jednostavnosti korištenja uz Anaconda Navigator, Anaconda je idealno okružje za svakoga koji želi raditi s podatkovnim znanostima kao što je strojno učenje. Integrirana razvojna okruženja Jupyter Notebook i JupyterLab dolaze već instalirana s distribucijom Anaconda, te ih se koristi u ovom završnome radu. Dodatno, Anaconda se koristi kako bi se olakšao rad s virtualnim okružjima. Primjer instaliranja različitih inačica programa je prikazan na slici 3.2.



Slika 3.1 Primjer instaliranja različitih inačica programa uz Navigator

3.1.3 Jupyter Notebook

Jupyter Notebook [35] je relativno novije integrirano razvojno okruženje često korišteno u području podatkovnih znanosti. Ima vrlo drugačiji pristup organiziranju koda od ostalih razvojnih okruženja. Integrirano razvojno okruženje Jupyter Notebook je dostupno u preko 40 programskih jezika kao što su Python, R, Julia i Scala. Programski jezik Python je doduše nužan za njegovu instalaciju. Kao što je spomenuto, najvažniji dio ovog razvojnog okruženja je drugačiji pristup organiziranju koda. Umjesto da se napiše cijeli kod u datoteku ili više datoteka, te se izvede odjednom, razvojno okruženje Jupyter Notebook svoj kod izdvaja u ćelije. Svaka ćelija se izvršava zasebno, ali podaci, tj. varijable se spremaju u globalno spremište te su dostupni svakoj ćeliji. Izgled ćelija je prikazan na slici 3.3.

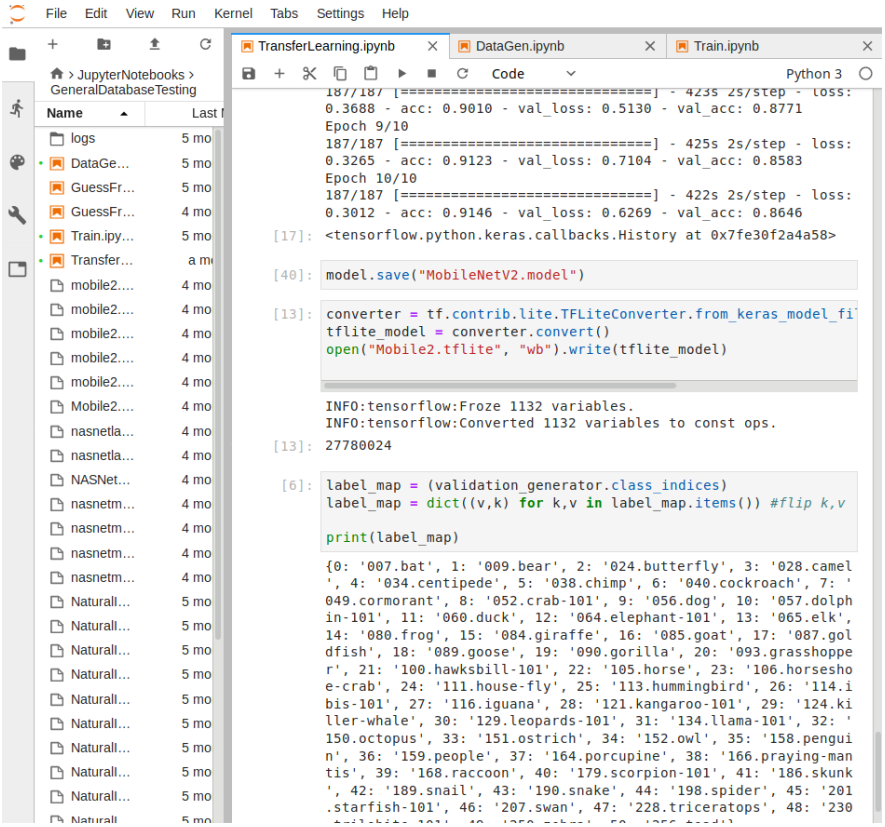


Slika 3.2 Izgled ćelija u Jupyter Notebook

Mogućnost da se svaki dio programskog koda zasebno izvede omogućava vrlo lako eksperimentiranje s programskim kodom i istraživanje promjena nastalih promjenom nekog izraza ili vrijednosti. Puno je brže s obzirom na to da se pokreće samo željeni, promijenjeni dio programskog koda.

Iako razvojno okruženje Jupyter Notebook nema vrlo razvijen sustav za automatsko dopunjavanje koda, kao što imaju razvojna okruženja Android Studio ili Visual Studio. Mogućnost zasebnog rada s ćelijama pokazala se izuzetno vrijedna kod rada u području podatkovnih znanosti, ali i svih ostalih znanosti gdje je potrebno brzo, ali efektivno prototipiranje programskog koda.

Razvojno okruženje JupyterLab je nadogradnja nad razvojnim okruženjem Jupyter Notebook. Sadrži mnoge nove mogućnosti koje znatno olakšavaju rad u ovom razvojnom okruženju. Neke od najbitnijih novih mogućnosti su izbornik datoteka koji daje brži pristup datotekama, mogućnost da jedan prozor sadrži više otvorenih datoteka, dodatne funkcije, postavke i posebni padajući izbornik na desnom kliku miša, koji je prije odgovarao običnom izborniku od internet preglednika. Izgled ćelija u razvojnom okruženju JupyterLab prikazan je na slici 3.4. Zbog svih tih pogodnosti integrirano razvojno okruženje JupyterLab odabrano je za razvoj neuronske mreže ovog završnog rada.



Slika 3.3 Izgled ćelija u razvojnom okruženju Jupyter Lab

3.1.4 TensorFlow

TensorFlow [36] je biblioteka programske podrške otvorenog koda namijenjena za numeričko računanje. Stvorio ju je Google za unutrašnje potrebe, ali se ubrzo pokazala vrlo korisna za široki spektar poslova. Nakon čega je Google učinio biblioteku TensorFlow dostupnim javnosti.

Biblioteka TensorFlow namijenjena je radu s tenzorima, no kako je rad s tenzorima najčešće potreban kod implementiranja algoritama strojnog učenja, može se reći da je TensorFlow biblioteka programske podrške namijenjena za istraživanje i rad u području strojnog učenja.

Strukturno, biblioteka TensorFlow koristi programski jezik Python kao sučelje između korisnika i kalkulacija u pozadini. Programski jezik Python idealan je zbog svoje jednostavnosti, pristupačnosti te popularnosti. Služi kao apstrakcija visoke razine za funkcije napisane u programskom jeziku C++ koje su zadužene za kalkulacije jer je C++ puno brži i optimiziraniji od programskog jezika Python. S obzirom na to da se radi u apstrakciji visoke razine, rad u biblioteci TensorFlow puno je lakši nego očekivano kad se spomene rad s neuronskim mrežama. Već u 20 linija koda može se implementirati jednostavna neuronska mreža koja može prepoznavati znamenke pomoću baze podataka MNIST [3] slika znamenaka. Za takvu jednostavnost korištenja zaslužna je biblioteka programske podrške Keras. Korisnik ne mora znati detaljno o algoritmima

ni arhitekturi neuronske mreže. Pomoću biblioteke TensorFlow korisnik slaže slojeve neuronske mreže te ju kasnije prevede. Učenje izgrađene neuronske mreže se čini s jednom jednostavnom funkcijom koja se nalazi na programskom kodu 4.6.

Biblioteka TensorFlow dodatno sadrži u sebi TensorBoard, skup alata za vizualizaciju rezultata neuronskih mreža koja olakšava korisniku da ispravi pogreške u kodu, optimizira kod, ili pak da shvati postupak učenja neuronskih mreža te da se upozna s pojmovima poput pretreniranja.

3.1.5 Keras

Keras [37] je biblioteka programske podrške visoke razine sagrađena na vrh biblioteke TensorFlow, ali ima mogućnost da radi s drugim bibliotekama programske podrške kao što su CNTK, Theano, MXNet, i PlaidML. Temeljna načela biblioteke Keras su modularnost, lakoća proširivanja i lakoća za korištenje. Visoka razina ove biblioteke ispunjava te zahtjeve te se složene operacije apstrahiraju jednostavnijim funkcijama. Visoka razina ove biblioteke znači da je manje prilagodljiva od većine ostalih biblioteka. Ako je tražena neuronska mreža vrlo specifična, biblioteke niže razine su bolji alat za njihovu implementaciju. Biblioteka Keras je sasvim dovoljna za implementaciju većinu neuronskih mreža te ga njegova jednostavnost čini odličnim za brzo prototipiranje neuronskih mreža.

U ovome radu se koristi biblioteka Keras uključena u biblioteku TensorFlow za izradu neuronske mreže, koja se zbog spomenutih prednosti pokazala kao jednostavniji način za izradu korištene neuronske mreže.

3.2. Korištene tehnologije i alati u izradi mobilne aplikacije

3.2.1 Android

Android je operacijski sustav tvrtke Google zasnovan na Linux jezgri prvenstveno namijenjen za pametne telefone. Na njegovom početku, bio je korišten na nekoliko crno-bijelih mobilnih uređaja. Sada, više od deset godina kasnije, najprošireniji je operacijski sustav na svijetu. U industriji pametnih telefona je uvjerljivo dominantan, a trenutni trendovi pokazuju da će ta dominacija sve više rasti.

Osim u industriji pametnih telefona, Google je razvio inačice operacijskog sustava Android koje danas upravljaju autima (Android Auto), pametnim satovima (Wear OS), modernim televizorima (Android TV) te se mogu pronaći na ostalim prenosivim uređajima.

Ključ njegovog uspjeha i rasprostranjenosti je Android projekt otvorenog koda (*eng. Android Open Source Project*). Cijeli programski kod nalazi se na Git-u te ga svako može preuzeti, testirati

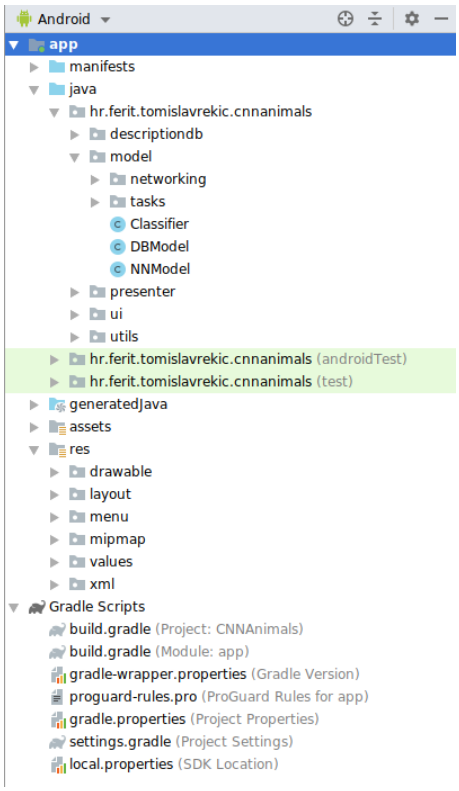
te unaprijediti. To tvrtki Google daje nadmoć nad tvrtkama koje čuvaju svoj kod. Te je i razlog zašto se operacijski sustav Android koristi u ovome radu.

3.2.2 Android Studio

Android Studio je službeno integrirano razvojno okruženje (*eng. integrated development enviroment, IDE*) namijenjeno razvijanju Android aplikacija. Zasnovan je na IntelliJ IDEA, jednog od najnaprednijih razvojnih okruženja za programski jezik Java. Tvrtka iza IntelliJ IDEA je također stvorila i razvojno okruženje PyCharm, možda najbolje razvojno okruženje za programski jezik Python, samim time čine odličan temelj za razvojno okruženje Android Studio.

Android Studio koristi Gradle koji je odgovoran za građenje aplikacije. Omogućava veliku fleksibilnost promjene projekta bez da se mijenjaju izvorne datoteke projekta. Gradle datoteke se nazivaju **build.gradle** te se u njima nalaze sve biblioteke koje se koriste u projektu, tj. biblioteke o kojem projekt ovisi (*eng. dependency*).

Struktura projekta u razvojnom okruženju Android Studio jednostavna je i intuitivna te se može vidjeti na slici 3.7. Na slici 3.7 nalazi se struktura Android projekta ovoga rada. U mapi pod imenom **java** nalazi se sav Java kod aplikacije. Kod se dijeli na kod aplikacije i kod za testiranje aplikacije, iako je kod za testiranje aplikacije briga za naprednije aplikacije i programere. Struktura **java** mape je proizvoljna te se može organizirati po volji, ali postoje naputci kojih se većina programera pridržava te ih je se dobro pridržavati kod razvoja većih projekta.



Slika 3.4 Struktura Android projekta

S obzirom na to da se kod razvijanja Android aplikacija često koristi arhitekturni predložak model-pogled-predstavljač (*eng. model-view-presenter, MVP*), tako se **java** mapa često dijeli na podmape **ui** za klase odgovorne za „pogled“ dio arhitekture, **presenter** za „predstavljač“ dio te **model** za „model“. Također postoji **util** za raznovrsne pomoćne klase, ali se mogu dodatno pronaći mape u koje se stavljaju klase odgovorne za bazu podataka i slično. Osim **java** mape postoji i **res** mapa u kojoj se nalaze sve XML (*eng. Extensible Markup Language*) datoteke, s kojim se opisuje izgled pogleda, te se definiraju korišteni nizovi znakova, ikone i ostali resursi.

Bitno je spomenuti da razvojno okruženje Android Studio dolazi s naprednim IntelliJ sustavom za automatsko dopunjavanje riječi, s kojim se vrijeme razvijanja aplikacije znatno smanjuje.

3.2.3 Java

Java [38] je objektno orijentirani programski jezik opće namjene. Razvio ju je James Gosling dok je radio u Sun Microsystems 1995. Sintaksa ovog programskog jezika je inspirirana programskim jezikom C++, a klasni sistem je sličan programskom jeziku Objective-C. Zbog toga, programeri koji imaju iskustva s programskim jezicima C ne bi trebali imati problema sa stvaranjem aplikacija u programskom jeziku Java.

Znatna promjena nad programskim jezicima C je automatsko sakupljanje smeća, što znači da korisnik ne mora ručno alocirati i dealocirati memoriju. Poziv za sakupljanje smeća se većinom događa kada je aplikacija neaktivna, iako će se dogoditi uvijek kada ponestaje pohrane memorije koja je aplikaciji dodijeljena. Zbog toga automatsko sakupljanje smeća ne bi trebalo imati veliki utjecaj na brzinu izvedbe aplikacije. Kvalitetno odrađeno ručno upravljanje memorijom je uvijek brže, ali treba više vremena da se kod za takvu aplikaciju napiše.

Ideja programskog jezika Java je pronaći način da se jednom prevedene aplikacije ne moraju ponovno prevoditi na drugom, različitom uređaju. To ostvaruje pomoću Java Virtualnog Stroja (*eng. Java Virtual Machine, JVM*). Programski jezik Java zapravo svoj kod prevodi u Java Bajt-kod (*eng. Java bytecode*), za razliku od ostalih u vrijeme nastanka programskog jezika Java koji su svoj kod prevodili izravno u strojni kod. Zbog toga, kod napisan u ovom programskom jeziku može se izvesti na svakom uređaju na kojem se nalazi JVM, za razliku od strojnog koda koji je specifičan za uređaj, iako se zbog toga brže izvodi.

Java je među najpoznatijim i najproširenijim programskih jezika danas. Nalazi se na internet stranicama u obliku malih aplikacija, pa sve do velikih igara. Od malih uređaja pa do poslužitelja, prilagodljiva je što pokazuje njezin broj korisnika. Popularnost programskog jezika Kotlin kao alternativa programskog jezika Java u kontekstu razvoja Android aplikacija raste te je upitno koji je programski jezik bolji. Međutim, Java je zbog svoje popularnosti, sličnosti programskim jezicima C, stabilnosti te intuitivne sintakse odabrana kao programski jezik za razvoj Android aplikacije.

3.2.4 XML

XML (*eng. Extensible Markup Language*) je jezik za označavanje podataka, ili opisni jezik. Sam po sebi ne radi ništa, samo sadrži podatke. Drugi programi moraju sami interpretirati XML format i raditi željene radnje s tim podacima. Razvio ga je W3C (*eng. World Wide Web Consortium*) s ciljem da stvori opisni jezik koji je jednostavan, proširiv i univerzalan.

XML, za razliku od HTML (*eng. Hypertext Markup Language*), nema strogo definirane oznake (*eng. tag*). Korisnik treba definirati svoje oznake koje misli koristiti, tako da nije ni definirano područje uporabe XML-a. U kontekstu razvoja Android aplikacija, služi za opisivanje resursa koje se koriste u aplikaciji. Od opisivanja pogleda do sadržavanja nizova znakova koje se u aplikaciji koriste. Primjer XML koda se može vidjeti u programskom kodu 3.1.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="@color/colorPrimaryDark">
    <TextureView
        android:id="@+id/texVCameraPreview"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="0.8" />
    <Button
        android:id="@+id/btnTakePicture"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="0.2"
        android:text="Take Picture"
        android:textColor="@color/colorAccent"
        android:background="@color/colorPrimary"
        android:textSize="@dimen/_20ssp"/>
</LinearLayout>
```

Programski kod 3.1 XML kod koji opisuje aktivnost za kameru

Oznaka **LinearLayout** na programskom kodu 3.1 označuje da su elementi koji se nalazi unutar poredani linearno, odozgo prema dolje. Takav raspored je jednostavan i često dovoljan, ali postoje složeniji načini za raspoređivanje elemenata po zaslonu. Oznaka **TextureView** je pogled ili okvir koji predstavlja rezervirano mjesto na kojemu će se prikazivati podatci dobiveni od kamere. Oznaka **Button** predstavlja gumb koji korisnik može stisnuti te se njezinim pritiskom spremi slika i pokrene glavna aktivnost, kao što je prikazano u programskom kodu 4.10. Parametri kao **layout_height** i **layout_width** definiraju visinu i širinu pogleda, redom. Parametar **layout_weight** definira koliki postotak zaslona pogled zauzima, ako je visina pogleda postavljena na 0. Rezultat programskog koda 3.1 se vidi na slici 5.2.

3.2.5 SQLite

SQLite [39] je biblioteka za stvaranje relacijskih baza podataka napisana u programskom jeziku C. Nalazi se u javnoj domeni te je besplatna za korištenje. Uz to je jedna od najkorištenijih upravitelja bazi podataka na svijetu. Za razliku od većine ostalih bazi podataka, baze podataka biblioteke SQLite ne rade na klijent-poslužitelj principu rada. Svi podaci su zapisani u pohrani uređaja na kojem se SQLite nalazi te su vezani za aplikaciju koja ga koristi. Baze podataka biblioteke SQLite su transakcijske baze podataka te se pridržavaju ACID načelima, tj. atomičnost (*eng. atomicity*), konzistentnost (*eng. consistency*), izoliranost, (*eng. isolation*) te izdržljivost (*eng. durability*).

Ukratko, atomičnost garantira da se svaka transakcija smatra kao zasebna jedinica. Transakcija mora ili apsolutno uspjeti ili apsolutno ne uspjeti. Ako se bilo koji dio transakcije zbog bilo kojeg razloga ne uspije izvršiti, baza podataka mora ostati nepromijenjena. Konzistentnost osigurava da uneseni podatak bude ispravan, da zadovoljava sva pravila i ograničenja koja u bazi podataka postoje za podatak koji se unosi. S obzirom na to da se transakcije često izvršavaju paralelno, tj. više transakcija u isto vrijeme, izoliranost osigurava da je rezultat paralelnog i sekvencijalnog čitanja ili pisanja podataka jednak. Drugim riječima, osigurava da jedna transakcija neće poremetiti rad druge. I posljednje, izdržljivost garantira da će jednom izvršena transakcija ostati zapamćena, čak i u slučaju neočekivanog prestanka rada uređaja.

Biblioteka SQLite dolazi instalirana na mnogim operacijskim sustavima, među kojima je i Android. U kontekstu razvoja Android aplikacija, biblioteka SQLite daje mogućnost sigurne i kvalitetne uporabe baze podataka u aplikaciji. Postoje biblioteke koje su više apstraktne od biblioteke SQLite te su time lakše za korištenje, ali biblioteka SQLite ima puno više mogućnosti i brža je, kao i svaki program koji koristi niže razine apstrakcije. To ovu biblioteku čini idealnom za programera koji zna barem osnove strukturnog upitnog jezika SQL. Primjer SQL koda za stvaranje baze podataka nalazi se u programskom kodu 3.2.

```
public final class DescriptionDatabaseSQLCommands {
    public static final String SQL_CREATE_ENTRIES =
            "CREATE TABLE " + DescriptionEntry.TABLE_NAME + " (" +
            DescriptionEntry._ID + " INTEGER PRIMARY KEY," +
            DescriptionEntry.COLUMN_NAME_NAME + " VARCHAR(25), " +
            DescriptionEntry.COLUMN_NAME_INFO + " TEXT, " +
            DescriptionEntry.COLUMN_NAME_PICTURE + " VARCHAR(50)," +
            DescriptionEntry.COLUMN_NAME_GUESS + " FLOAT, " +
            DescriptionEntry.COLUMN_NAME_GUESS_COUNT + " INTEGER, " +
            DescriptionEntry.COLUMN_NAME_LAST_SEEN + " DATE)";
    public static final String SQL_DELETE_ENTRIES =
            "DROP TABLE IF EXISTS " + DescriptionEntry.TABLE_NAME;
}
```

Programski kod 3.2 SQL kod za stvaranje baze podataka

Varijable **SQL_CREATE_ENTRIES** i **SQL_DELETE_ENTRIES** u programskom kodu 3.2 su statične te sadrže gotove SQL upite koji se mogu pozvati putem SQLite biblioteke.

3.2.6 Retrofit

Retrofit je REST (*eng. representational state* transfer) klijent za Javu i Android [40]. Retrofit služi za dohvaćanje podataka s internet stranica, te to čini s HTTP zahtjevima pomoću biblioteke OkHttp. Podatci na internet stranicama su najčešće u JSON formatu, ali Retrofit podržava i nekoliko ostalih formata.

Implementacija biblioteke Retrofit u Android aplikaciju jednostavno je, te se apstrahira iza nekoliko funkcija koje se odvijaju u pozadini. Sama jednostavnost ga čini idealnim za ovu aplikaciju te se koristi za dohvaćanje informacija o životinjama s internet stranice Wikipedia.

3.3. Povezivanje rezultata strojnog učenja s mobilnom aplikacijom

3.3.1 TFLite

TFLite, ili TensorFlow Lite je biblioteka namijenjena za korištenje istreniranih modela na mobilnim uređajima, poput pametnih telefona. Još je u razvoju pa sve mogućnosti nisu trenutno na raspolaganju, ali je za jednostavne zadatke poput klasificiranja slika odličan.

Napisan je u programskom jeziku C++, kao i biblioteka TensorFlow, te ima Java API kojeg mogu koristiti Android aplikacije. S obzirom na to da je namijenjen uporabi na mobilnim uređajima, najvažniji zadatak koji ima je optimizirati korištenje vrlo ograničenih resursa kojim većina mobilnih uređaja raspolaže.

Biblioteka TFLite se trenutno isključivo koristi za korištenje istreniranih modela, te se ne mogu na njemu modeli trenirati. Ideja je da se modeli prvo istreniraju na računalu, ili nekom drugom jačem uređaju.

4. PROGRAMSKO RJEŠENJE MOBILNE APLIKACIJE S KLASIFICIRANJEM SLIKA

U ovome poglavlju će se opisati i objasniti kod koji čini temelj funkcionalnosti aplikacije i neuronske mreže. Kod neuronske mreže je kraći pa će se objasniti u potpunosti, dok je kod android aplikacije predugačak za detaljni opis, pa će fokus biti na temeljnoj funkcionalnosti.

4.1. Programsko rješenje klasificiranja slika

Ovdje će se opisati dio koda odgovoran za opisivanje i učenje konvolucijske neuronske mreže ovog završnog rada. Kod je napisan u Pythonu koristeći biblioteku TensorFlow, kao što je prikazano na programskom kodu 4.1.

```
import tensorflow as tf
from tensorflow.keras.layers import Dense,GlobalAveragePooling2D,Dropout
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input,MobileNetV2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Model
```

Programski kod 4.1 Biblioteke korištene za izvedbu strojnog učenja

Počinje se od uvoza biblioteka i određenih klasa iz tih biblioteka (programski kod 4.1). Glavno je uvoženje biblioteke **tensorflow**, iz koje se onda bira potrebne funkcije. Točnije, uzima se iz inačice biblioteke Keras sadržane u biblioteci TensorFlow, koja je puno jednostavnija za korištenje, kao što je spomenuto. Klase vezane za arhitekturu neuronske mreže su gusti sloj, globalno prosječno udruživanje i sloj za opadanje.

S obzirom na to da se koristi MobileNetV2 kao temelj mreže koja će se koristiti i trenirati koristeći prijenos učenja, potrebno je koristiti i funkciju koja priprema podatke za rad s MobileNetV2 mrežom. Ta funkcija se zove **preprocess_input**. Inicijalizacija modela je prikazana na programskom kodu 4.2.

Dodatno, koristi se generator slika koji tijekom rada mreže predaje slike na ulaz mreže. Te slike mogu biti promijenjene nasumično, određeno parametrima koje se generatoru predaju. Funkcija za generator je **ImageDataGenerator**. Zadnje, uvozi se klasa **Adam** koja predstavlja algoritam za optimiziranje neuronske mreže.

```
base_model=MobileNetV2(weights='imagenet',include_top=False,
         input_shape=(224,224,3))

x=base_model.output
x=GlobalAveragePooling2D()(x)
x=Dense(1024,activation='relu')(x)
x=Dropout(0.3)(x)
x=Dense(1024,activation='relu')(x)
x=Dropout(0.3)(x)
x=Dense(512,activation='relu')(x)
x=Dropout(0.5)(x)
preds=Dense(51,activation='softmax')(x)

model=Model(inputs=base_model.input,outputs=preds)
```

Programski kod 4.2 Definiranje modela

Poslije uvoza biblioteka potrebno je definirati model neuronske mreže koji će se koristiti (programski kod 4.2). Prvo se u varijablu **base_model** stavi MobileNetV2 model već istreniran na bazi podataka ImageNet slika. Parametar **include_top** govori želi li se uključiti zadnje, guste slojeve mreže, što u ovom slučaju nije poželjno. Zadnji parametar **input_shape** definira dimenzije ulaza mreže.

U varijablu **x** se počne slagati struktura zamišljenog modela. Prvo se počinje s **base_model**, a poslije se redom slažu slojevi koji su potrebni. U ovom slučaju se koristi 3 gusta sloja s ReLU aktivacijskom funkcijom i zadnji **softmax** sloj. Dimenzije slojeva redom su 1024, 1024, 512 te zadnji od 51, što odgovara broju klasa. Koristi se sloj za ispadanje (*eng. dropout*) za svaki gusti sloj, dva od 0.3 i jedan 0.5. U varijablu model pohrani se krajnji model koji će se koristiti. Nakon definiranja modela slijedi određivanje koji slojevi se uče, prikazano na programskom kodu 4.3.

```
for i,layer in enumerate(model.layers):
  print(i,layer.name)


for layer in model.layers[:156]:
    layer.trainable=False
for layer in model.layers[156:]:
    layer.trainable=True
```

Programski kod 4.3 Ispis slojeva i onemogućavanje učenja određenih slojeva

Prva *for* petlja s programskog koda 4.3 ispisuje sve slojeve u korištenom modelu. S obzirom na to da model ima 162 slojeva ispis je dugačak, no iz ispisa se dobije korisna informacija koja se kasnije koristi kad se odlučuje koje se slojeve treba zamrznut tijekom treniranja. Druge i treća *for* petlja zajedno zamrznu, tj. onemoguće učenje prvih 155 slojeva, što odgovara broju slojeva modela bez gustih slojeva koji su naknadno dodani.

Nakon toga se moraju odrediti slike na kojim se neuronska mreža trenira. Slike će dostavljati generator slika te se njegova inicijalizacija nalazi na programskom kodu 4.4.

```
batch_size = 32
IMG_SIZE = 224
TRAINDIR = "direktorij gdje se nalaze slike za učenje"
TESTDIR = direktorij gdje se nalaze slike za testiranje"

train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    horizontal_flip=True,
    rotation_range=40,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.2,
    zoom_range=0.2,
    fill_mode ='nearest')

test_datagen = ImageDataGenerator(
        preprocessing_function=preprocess_input)

train_generator = train_datagen.flow_from_directory(
        TRAINDIR,
        target_size=(IMG_SIZE, IMG_SIZE),
        color_mode='rgb',
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=True)

validation_generator = test_datagen.flow_from_directory(
        TESTDIR,
        target_size=(IMG_SIZE,IMG_SIZE),
        color_mode='rgb',
        batch_size=batch_size,
        class_mode='categorical',
        shuffle = True)
```

Programski kod 4.4 Definiranje generatora slika

U varijablu **batch_size** s programskog koda 4.4 upiše se veličina grupe u kojoj se slike stavljaju i zajedno pošalju na procesiranje. U **IMG_SIZE** se upiše dimenzija ulazne slike.

Kod u programskom kodu 4.4 definira generatore koji će pružati slike modelu. Prvo se definira općeniti generator slika koji prilagođuje slike modelu putem **preprocess_input**. Parametrom **horizontal_flip** se omogućuje nasumično zrcaljenje slike, te se dodatno može odrediti kolika se rotacije slike želi, u parametru **rotation_range**. Također nalaze se **width_shift_range** i **height_shift_range** koji definiraju translaciju slike. Parametar **shear_range** definira iskrivljenje slike, a **zoom_range** približavanje ili udaljavanje slike. Parametar **fill_mode** definira način kako se popune prazna mjesta na slici koja mogu nastati nakon navedenih transformacija.

Nakon toga se definira generator koji će spojiti generator slika i ulaz modela. **TRAINDIR/TESTDIR** sadrže put do mape u kojoj se slike nalaze. Nakon toga se definira generator za treniranje i testiranje modela. Sadrži razne parametre s kojima se može prilagoditi raznim potrebama. Parametar **target_size** određuje veličinu slike koje generator dostavlja Parametar **color_mode** određuje najčešće koriste li se crno-bijele slike ili u boji. Parametar **class_mode** određuje način klasificiranja slike. Parametar **shuffle** određuje dostavljaju se slike iz

klasa zajedno ili nasumično iz svih klasa. Prije nego se može započeti učenje neuronske mreže određuju se dodatne mogućnosti s programskog koda 4.5.

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


filepath = "mobile2.{epoch:02d}-{val_acc:.2f}.hdf5"

checkpoint = tf.keras.callbacks.ModelCheckpoint(
        filepath,
        monitor='val_loss',
        save_best_only=True,
        mode='min',
        period=1)

callBacks = [checkpoint]
```

Programski kod 4.5 Prevođenje modela i automatsko spremanje

Prethodni programski kod 4.5 zadužen je za prevođenje modela, gdje se moraju opisati dodatni parametri, kao što je **loss** koji govori kakva se težinska funkcija želi, ili **metrics** gdje se bira po kojem parametru se mreža prati.

Također, u varijablu **callBacks** se spremaju dodatne mogućnosti koje se izvršavaju tijekom treniranja modela. Ovdje je dodano spremanje modela nakon svake epohe, s time da se model s najmanjom težinom ostavlja. Poslije definiranja povratnih funkcija učenje funkcije može započeti te je kod odgovoran za to prikazan na programskom kodu 4.6.

```
model.fit_generator(
        train_generator,
        steps_per_epoch=5000 // batch_size,
        epochs=30,
        validation_data=validation_generator,
        validation_steps=500 // batch_size, callbacks = callBacks)

model.save("MobileNetV2.model")
```

Programski kod 4.6 Učenje i spremanje modela

Konačno, programski kod 4.6 započinje postupak učenja modela. Funkciji se predaju definirani generatori, broj koraka treniranja, broj koraka za potvrdu, broj epoha te prethodno opisana povratna funkcija zadužena za spremanje najboljeg modela. Nakon toga model se može spremiti u datoteku za naknadno korištenje.

4.2. Prijenos naučenog modela na mobilnu aplikaciju

Povezivanje istreniranog modela je jednostavno zahvaljujući biblioteci TensorFlow Lite. Sve što je potrebno je pozvati par funkcija nad datotekom koja sadrži model i dobit će se datoteka koja se može koristiti u Android projektu. Funkcije su prikazane na programskom kodu 4.7.

```
converter = tf.contrib.lite.
        TFLiteConverter.from_keras_model_file('mobile2.04-0.33.hdf5')
tflite_model = converter.convert()
open("Mobile2.tflite", "wb").write(tflite_model)
```

Programski kod 4.7 Prijenos modela na novi format datoteke

Programski kod 4.7, poziva funkciju **from_keras_model_file** klase **TFLiteConverter** nad datotekom modela te dobivene podatke zapiše u novu varijablu **tflite_model**. Sljedeće, otvori se datoteka **Mobile2.tflite** te se u nju zapišu podaci varijable **tflite_model**. Parametar **wb** označuje da se podaci zapisuju u binarnom obliku (*eng. write binary*).

4.3. Programsko rješenje mobilne aplikacije

4.3.1 Rad s neuronskom mrežom

Kad se aplikacija učita, korisniku predstavi glavni zaslon s tipkom da uključi kameru i snimi sliku koju želi klasificirati.

Pritiskom na gumb **OPEN CAMERA** (slika 5.1), pozove se funkcija **dispatchTakePictureIntent** koja se nalazi u programskom kodu 4.8 te koja funkcijom **startActivityIfNeeded** poziva aktivnost iz varijable **intentCamera** odgovornu za rukovanje s kamerom (slika 5.2).

```
private void dispatchTakePictureIntent(){
    try {
        btnSwitch.setEnabled(false);
        startActivityIfNeeded(intentCamera,0);
    }
    catch (Exception e){
        e.printStackTrace();
    }
}
```

Programski kod 4.8 Funkcija za pozivanje kamere

Prije nego što se pokrene aktivnost s kamerom, mora se u funkciji **configureReceiver** u programskom kodu 4.9 inicijalizirati prijemnik koji će primati signal koji obavještava glavnu aktivnost da je slika u aktivnosti s kamerom spremljena. Prvo se u varijablu **filter** spremi instanca klase **IntentFilter** koja sadrži ključ **Constants.BROADCAST_KEY1** koji opisuje signal poslan iz aktivnosti s kamerom. Nakon toga se u varijablu **receiver** spremi instancu klase **BroadcastReceiver** koji služi kao prijemnik. Pošto je klasa **BroadcastReceiver** apstraktna, mora se opisati funkcionalnost funkcije **onReceive**, koja se poziva kad instanca iste klase **receiver** primi odgovarajući signal. U funkciji **onReceive** se poziva funkcija predstavljača **runThroughNN** i predaje joj se ključ **Constants.TEMP_IMAGE_KEY** koji odgovara slici snimljene kamerom mobilnog uređaja.

```
private void configureReceiver() {
    filter = new IntentFilter();
    filter.addAction(Constants.BROADCAST_KEY1);
    receiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            presenter.runThroughNN(Constants.TEMP_IMG_KEY);
            showLoadingText();
        }
    };
    receiverReg();
}
```

Programski kod 4.9 Inicijalizacija prijemnika

U aktivnosti u kojoj se rukuje kamerom(Slika 5.2), postoji gumb **TAKE PICTURE** s kojim se snimi slika i pošalje signal prethodnoj aktivnosti da je slika snimljena (programski kod 4.10).

```
private void initListeners() {
    takePicture.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            try {
                takePicture.setEnabled(false);
                try {
                    CreateFileFromBitmap.createFileFromBitmap(
                            cameraController.takePicture(),
                            CameraActivity.this,
                            Constants.TEMP_IMG_KEY);
                }
                catch (Exception e){
                    e.printStackTrace();
                }
                finally {
                    broadcastIntent();
                    startActivityIfNeeded(switchIntent,0);
                }
            }
            catch (Exception e){
                e.printStackTrace();
            }
        }
    });
}
```

Programski kod 4.10 Spremanje slike, slanje signala te vraćanje u glavnu aktivnost

U programskom kodu 4.10 u funkciji **initListeners** na varijabli **takePicture** klase **Button** pozvana je funkcija **setOnClickListener** kojoj se predaje novostvorena instanca klase **View.OnClickListener**. Ta klasa je apstraktna, pa je nužno opisati funkcionalnost njene funkcije **onClick** koja se poziva na pritisak gumba. Nakon što je gumb pritisnut, pozove se funkcija **createFileFromBitmap** klase **CreateFileFromBitmap** kojoj se predaje slika, kontekst aktivnosti s kamerom i ime spremljene slike **Constants.TEMP_IMG_KEY**. Slika se predaje u obliku

povratne vrijednosti funkcije **takePicture** nad varijablu **cameraController**, koja je instanca klase **CameraController**, koja predstavlja omotač za sve potrebne funkcionalnosti kamere. Funkcija **createFileFromBitmap** stvara datoteku koja sadrži sliku snimljenu kamerom. Nakon što je datoteka zapisana šalje se signal putem funkcije **broadcastIntent** te se funkcijom **startActivityIfNeeded** prebacuje nazad na glavnu aktivnost. S obzirom na to da je poslan signal pokreće se funkcija u primatelju te se poziva funkcija predstavljača **runThroughNN** prikazana na programskom kodu 4.11.

```
@Override
public void runThroughNN(final String imageKey) {
    mView.updatePicture(imageKey);
    mModel.runThroughNN(imageKey, new ClassifierCallback() {
        @Override
        public void processFinished(String label, int labelIndex, float activation)
{
            mView.updateText(label, activation);
            mDBPresenter.updateRow(labelIndex, activation, imageKey);
        }
    });
}
```

Programski kod 4.11 Predaja slike neuronskoj mreži

Funkcija **runThroughNN** s programskog koda 4.11 prvo predaje ključ snimljene slike sadržan u parametru **imageKey** funkciji **updatePicture** pogleda **mView**, koja tu sliku otvori i prikaže na zaslonu. Drugo, poziva na modelu **mModel** istoimenu funkciju **runThroughNN** kojoj predaje parametar **imageKey** i novu instancu apstraktne klase **ClassifierCallback** čija se funkcija **processFinished** izvršava kad se postupak klasificiranja slike završi. U funkciji **processFinished** pogledu **mView** predaju se rezultati klasifikacije funkcijom **updateText**, kao i predstavljaču **mDBPresenter** funkcijom **updateRow** koja po potrebi ažurira sadržaj baze podataka rezultatima klasifikacije. Model **mModel** pozivom funkcije **runThroughNN** prosljeđuje podatke klasi **Classifier** koja sadrži i radi s modelom neuronske mreže (programski kod 4.12).

```
public void classify(ClassifierCallback callback, String imageKey){
    mCallback = callback;
    new FetchImageTask(mContext, new FetchImageCallback() {
        @Override
        public void processFinish(ByteBuffer output) {
            new RunNeuralNetworkTask().execute(output);
        }
    }).execute(imageKey);
}
```

Programski kod 4.12 Funkcija koja predstavlja adapter za neuronsku mrežu

Klasa **Classifier** odgovorna za rukovanje s istreniranom neuronskom mrežom sadrži funkciju **classify** (programski kod 4.12), koja stvara privremenu instancu klase **FetchImageTask** koja

proširuje klasu **AsyncTask**, te se na njoj poziva funkcija **execute** koja započinje asinkrono izvršavanje klase kojoj pripada. Klasa **FetchImageTask** ima zadaću učitati sliku snimljenu kamerom mobilnog uređaja. Zatim pozove na toj slici pretvarače iz formata **Bitmap** u format **ByteBuffer**, te zatim sliku proslijedi privremenoj instanci klase **RunNeuralNetworkTask** (programski kod 4.13) koja također proširuje klasu **AsyncTask** te se izvršava asinkrono, na drugoj niti.

```
class RunNeuralNetworkTask extends AsyncTask<ByteBuffer, Void,float[][]> {
    @Override
    protected float[][] doInBackground(ByteBuffer... params) {
        ByteBuffer imgData = params[0];
        float[][] output = new float[1][labels.size()];
        if(imgData == null) return output;
        tflite.run(imgData, output);
        return output;
    }
    @Override
    protected void onPostExecute(float[][] floats) {
        super.onPostExecute(floats);
        int labelIndex = processLabelProb(floats);
        float activation = floats[0][labelIndex];
        mCallback.processFinished(labels.get(labelIndex), labelIndex, activation);
    }
}
```

Programski kod 4.13 Asinkrono pozivanje neuronske mreže

Klasa **RunNeuralNetwork task** s programskog koda 4.13 opisuje funkcije **doInBackground** i **onPostExecute**. Funkcija **doInBackground** sadrži operacije koje se izvršavaju na drugoj niti, dok se operacije funkcije **onPostExecute** izvršavaju na glavnoj niti nakon izvršavanja funkcije **doInBackground**. Glavna funkcija ovdje je funkcija **run** varijable **tflite**, koja sadrži instancu klase **Interpreter**. Funkcija **run** kao parametar prima sliku **imgData** te u drugi parametar **output** sprema izlaz neuronske mreže. Poslije izvršavanja klasifikacije poziva se povratna funkcija **mCallback.processfinished** koja je opisana u predstavljaču (programski kod 4.11). Inicijalizacija **tflite** varijable se radi **Interpreter** klasom kojoj se preda varijabla tipa **File** koja sadrži datoteku modela (programski kod 4.14).

```
private void initModel(String modelFileName) {
    try {
        tflite = new Interpreter(getModelFile(modelFileName));
    }
    catch (IOException e){
        e.printStackTrace();
    }
}
```

Programski kod 4.14 Inicijalizacija neuronske mreže u tflite varijablu

4.3.2 Rad s bazom podataka

Rad s bazom podataka kreće od njezine inicijalizacije, koja se pokreće s pokretanjem aplikacije. Inicijalizacija baze podataka izvršava se pozivanjem funkcije s programskog koda 4.15.

```
private void initDb(){
     DescriptionDbController tempController = new DescriptionDbController(mContext);
     InputStream stream = null;
     AssetManager assetManager = mContext.getAssets();
     try{
          stream = assetManager.open(Constants.QUESTION_ICON);
     }
     catch (IOException e){
          Log.e(TAG, "initDbData: " + e.toString());
     }
     Bitmap scaled = null;
     if(stream != null){
          Bitmap temp = BitmapFactory.decodeStream(stream);
          scaled = Bitmap.createScaledBitmap(
                    temp,
                    Constants.DB_IMG_DIM_X,
                    Constants.DB_IMG_DIM_Y,
                    true);
     }
     CreateFileFromBitmap.createFileFromBitmap(scaled, mContext,
Constants.NO_IMG_KEY);
     for(int i=0; i<labels.size();i++){
          DescriptionDbSingleUnit tempUnit = new DescriptionDbSingleUnit(
                    labels.get(i),
                    "",
                    Constants.NO_IMG_KEY,
                    0.0f,
                    0,
                    "never" );
          tempController.insertRow(tempUnit);
     }
     initDesc();
}
```

Programski kod 4.15 Funkcija za inicijalizaciju baze podataka

U funkciji **initDb** u programskom kodu 4.15 se opisuju početne vrijednosti redova unutar baze podataka. Prvo se s funkcijom **assetManager.open** otvori datoteka koja sadrži početnu sliku te se njena dimenzija promjeni na željenu vrijednost, nakon čega se spremi u privremenu datoteku pomoću funkcije klase **CreateFileFromBitmap** zvanom **createFileFromBitmap**. Nakon toga se za svaku klasu životinja stvori novi **DescriptionDbSingleUnit** objekt kojemu se pridružuje ime, opis životinje, privremenu sliku te početne vrijednosti postotka točnosti, broja susreta te datum zadnjeg susreta. Opis životinje se dohvaća funkcijom **initDesc** u programskom kodu 4.19. Kad se unos podataka izvrši pozove se povratna funkcija koja omogućava nastavak rada s aplikacijom. Varijabla **tempController** predstavlja instancu klase **DescriptionDbController** koja formira SQL upite te služi kao omotač klasi **SQLiteDatabase** za funkcije s programskog koda 4.16.

```
long newRowId = db.insert(
        DescriptionContract.DescriptionEntry.TABLE_NAME,
        null,
        InputToContentVal(input, Mode.UPDATE_FULL));
```

```
Cursor cursor = db.query(
        DescriptionContract.DescriptionEntry.TABLE_NAME,
        projection,
        selection,
        selectionArgs,
        null,
        null,
        sortOrder);
```

```
db.update(
    DescriptionContract.DescriptionEntry.TABLE_NAME,
    values,
    selection,
    selectionArgs);
```

Programski kod 4.16 Temeljne funkcije klase SQLiteDatabase

Varijabla **db** s programskog koda 4.16 predstavlja instancu klase **SQLiteDatabase**. Tri koda prikazana na slici se nalaze u odvojenim funkcijama klase **DescriptionDbController**, ali su temelj tih funkcija koje su odgovorne za umetanje, pretragu odnosno ažuriranje redova. Funkciji **db.insert** se predaje ime tablice te redom podaci koji trebaju biti uneseni, a vraća indeks unesenog reda. Funkcija **db.query** prima ime tablice, ime stupaca koji će biti prikazani koji su pohranjeni u varijabli **projection**, oblik pretrage u **selection**, argument pretrage u **selectionArgs**, **null** za argumente vezane za grupiranje podataka te **sortOrder** za način sortiranja. Zadnje, funkciji **db.update** predaje se ime tablice, nove vrijednosti određenog retka te argumenti za pretragu slični argumentima kod **db.query**. Sintaksa ovih funkcija je vrlo slična tradicionalnim SQL upitima, ali se riječi upita formiraju iz varijabli koje sadrže nizove znakova. Funkcija za ažuriranje **DescriptionDbControllera** koja poziva **db.update** se poziva iz funkcije **updateRow** klase **DescriptionDbUpdateManager**, prikazane na programskom kodu 4.17.

```
public void updateRow(DescriptionDbSingleUnit input){
    if(input.getGuess() < minGuessValue){
        return;
    }
    DescriptionDbSingleUnit tempUnit = controller.readDb(input.getName()).get(0);
    input.setGuessCount(1 + tempUnit.getGuessCount());
    if(input.getGuess() > tempUnit.getGuess()){
        File temp = new File(mContext.getFilesDir() , input.getPicture());
        temp.renameTo(new File(mContext.getFilesDir() , input.getName()));
        input.setPicture(input.getName());
        updatePic(input);
    }
    else {
        updateGuess(input);
    }
}
```

Programski kod 4.17 Funkcija koja odlučuje način ažuriranja tablice

Funkcija **updateRow** s programskog koda 4.17 odlučuje na koji način će se redak ažurirati te se poziva unutar funkcije **updateDb** klase **DescriptionDbUpdater** (programski kod 4.18). Prvo se pročita postotak točnosti podatka koji se unosi s **input.getGuess**, ako je postotak točnosti manji od minimalnog dozvoljenog postotka točnosti onda se unos podataka prekine. Nakon toga u bazi podataka se traži redak pod istim imenom koristeći **controller.readDb** te se taj redak pohranjuje u varijablu **tempUnit**. Broj koliko je puta životinja viđena uveća se za jedan te ako je postotak točnosti retka koji se unosi veći od istoimenog retka iz baze podataka nastavlja se s unosom slike i broja susreta. Ako je postotak točnosti manji, onda se mijenja samo broj susreta.

```
public void updateDb(int guessedLabelIndex, float guessedActivation, Bitmap
inputImage) {
    DescriptionDbUpdateManager manager = new DescriptionDbUpdateManager(mContext);
    Date c = Calendar.getInstance().getTime();
    SimpleDateFormat df = new SimpleDateFormat("dd/MM/yyyy");
    String formattedDate = df.format(c);
    DescriptionDbSingleUnit tempUnit = new DescriptionDbSingleUnit(
            mLabels.get(guessedLabelIndex),
            null,
            createImageFromBitmapDB(inputImage),
            guessedActivation,
            0,
            formattedDate);
    manager.UpdateRow(tempUnit);
}
```

Programski kod 4.18 funkcija koja konstruira redak koji se unosi u bazu podataka

Funkcija **updateDb** ima odgovornost konstruirati redak te ga predati klasi **DescriptionDbUpdateManager**. U varijablu **formattedDate** sprema se trenutni datum, koji služi kao datum zadnjeg susreta. Dalje, stvara se objekt koji sadrži sve podatke za jedan red baze podataka te ga prosljeđuje funkciji s programskog koda 4.17.

```
public void initDesc(){
     if(labels.size() == 0){
          initLabels();
     }
     WikiDescService service = new WikiDescService();
     for (int j=0; j<labels.size(); j++){
          service.getResponse(labels.get(j), new Callback<WikiDescResponse>() {
               @Override
               public void onResponse(Call<WikiDescResponse> call,
Response<WikiDescResponse> response) {
                    if(!response.isSuccessful()) return;
                    Page page =
(Page)response.body().getQuery().getPageMap().values().toArray()[0];
                    String animName = call.request().url().queryParameter("titles");
                    String desc = page.getExtract();
                    descs.put(animName, desc);
                    counter++;
                    if(counter == labels.size()){
                         updateDescs();
                    }
               }
               @Override
               public void onFailure(Call<WikiDescResponse> call, Throwable t) {
                    Log.d(TAG, "onFailure: " + t.toString());
                    String animName = call.request().url().queryParameter("titles");
                    String desc = "";
                    descs.put(animName, desc);
                    counter++;
                    if(counter == labels.size()){
                         updateDescs();
                    }
               }
          });
     }
}
```

Programski kod 4.19 Funkcija za dohvaćanje opisa životinja

Funkcija **initDesc** na programskom kodu 4.19 primjer je jednostavne uporabe Retrofita. Prvo se stvara nova instanca klase **WikiDescService**, čiji se kod nalazi na programskom kodu 4.20. Nakon toga se za svaku klasu životinja dohvaćaju podaci s odgovarajućeg Wikipedia [28] članka te se nakon dohvaćanja interpretira i spremi u listu opisa životinja **descs**. Ako je dohvaćanje podataka neuspješno za opis životinje se spremi prazan niz znakova. Funkcija **initDesc** poziva funkciju **updateDescs** s programskog koda 4.21 nakon što se svi podaci dohvate.

```
public class WikiDescService {
    private Call<WikiDescResponse> mCallAsync;
    private WikiDescAPI wikiDescAPI;
    public WikiDescService(){
        HttpLoggingInterceptor interceptor = new HttpLoggingInterceptor();
        interceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
        OkHttpClient.Builder httpClient = new OkHttpClient.Builder()
                .addInterceptor(interceptor);
        Retrofit retrofit = new Retrofit.Builder().
                baseUrl(Constants.WIKI_API_BASE).
                addConverterFactory(GsonConverterFactory.create()).
                client(httpClient.build()).
                build();
        wikiDescAPI = retrofit.create(WikiDescAPI.class);
    }
    public void getResponse(String title, Callback<WikiDescResponse> callback){
        mCallAsync = wikiDescAPI.getResponse(
                "json",
                "query",
                "extracts",
                "1",
                "1",
                title);
        mCallAsync.clone().enqueue(callback);
    }
}
```

Programski kod 4.20 Klasa koja koristi Retrofit za komunikaciju s internet stranicom

Klasa **WikiDescService** s programskog koda 4.20 odgovorna je za kreiranje upita i postavljanje veze s internet stranicom. Klasa **HttpLoggingInterceptor** nije nužna za rad, ali daje korisne informacije vezane za stanje veze između aplikacije i internet stranice. U konstruktoru klase **WikiDescService** stvara se usluga koja je vezana ta osnovni URL stranice s koje se dohvaćaju podaci. U ovom slučaju usluga je vezana za osnovni URL internet stranice Wikipedia. Funkcijom **getResponse** se usluzi govori s kojeg se članka na internet stranici Wikipedia dohvaćaju podaci i u kojem obliku.

```
private void updateDescs(){
     DescriptionDbController controller = new DescriptionDbController(mContext);
     List<DescriptionDbSingleUnit> data = controller.readAll();
     for(int i = 0; i<labels.size(); i++){
          DescriptionDbSingleUnit input = data.get(i);
          input.setInfo(descs.get(input.getName()));
          controller.updateRow(input, DescriptionDbController.Mode.UPDATE_FULL);
     }
     mCallback.processFinished();
}
```

Programski kod 4.21 ažuriranje baze podataka s novim opisima životinja

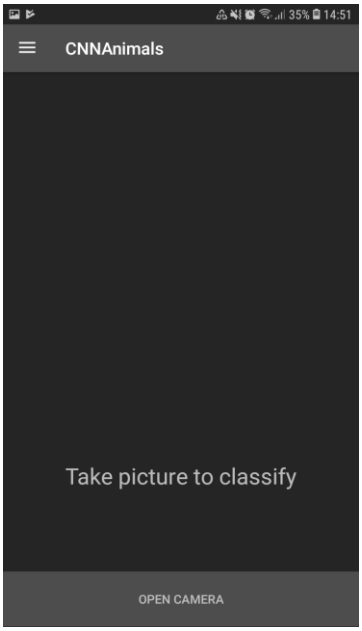
Programski kod 4.21 iščitava podatke iz baze podataka putem funkcije **controller.readAll** te ih pohranjuje u varijablu **data**. Zatim se svakom pročitanom podatku pridružuje opis životinje dohvaćene s internet stranice Wikipedia te ih sprema nazad u bazu podataka. Povratna funkcija **mCallback.processFinished** prosljeđuje informaciju da je posao završen.

5. KORIŠTENJE MOBILNE APLIKACIJE, REZULTATI I ANALIZA

U ovome poglavlju nalazi se kratki vodič kroz funkcionalnosti i mogućnosti aplikacije. Nakon čega se nalaze ispiti kvalitete rada aplikacije i neuronske mreže u pozadini.

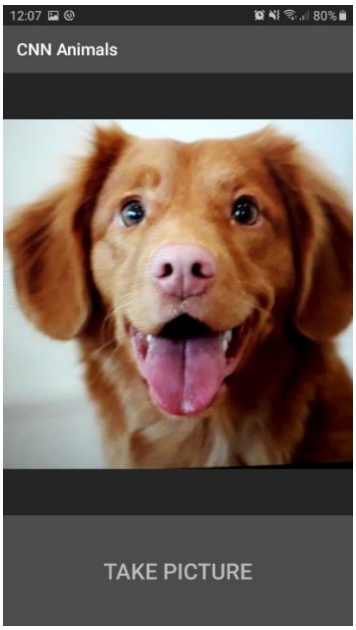
5.1. Korištenje mobilne aplikacije

Korištenje aplikacije je intuitivno i u svakom trenutku bi trebalo biti jasno kako se interakcija s aplikacijom treba odvijati. Na početnom zaslonu (Slika 5.1) nalaze se gumb **OPEN CAMERA** koji otvara kameru koja stvara sliku za ulaz u neuronsku mrežu. Dodatno, postoji navigacijska traka (slika 5.7) koja se može otvoriti tako da se prevuče prstom preko zaslona s lijeve strane na desnu.



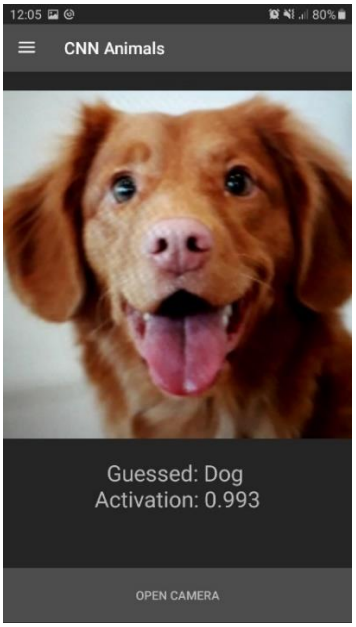
Slika 5.1 Početni zaslon aplikacije

Kada se stisne gumb **OPEN CAMERA** otvori se aktivnost u kojoj se može snimiti slika. Kada se životinja koja se želi klasificirati prikaže jasno u kadru treba se stisnuti **TAKE PICTURE** (Slika 5.2) kako bi sliku snimili.



Slika 5.2 Aktivnost s kamerom

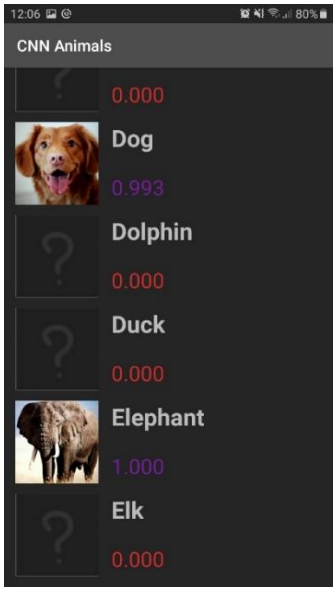
Aplikacije je vratila rezultat strojnog učenja zasnovanog na konvolucijskoj neuronskoj mreži koja radi u pozadini. Neuronska mreža je 99.3% sigurna da je na ovoj slici pas, što je dobar i točan rezultat (Slika 5.3). Ova se slika sada također nalazi u bazi podataka aplikacije jer je najkvalitetnija slika psa do sad (Slika 5.4).



Slika 5.3 Rezultat neuronske mreže

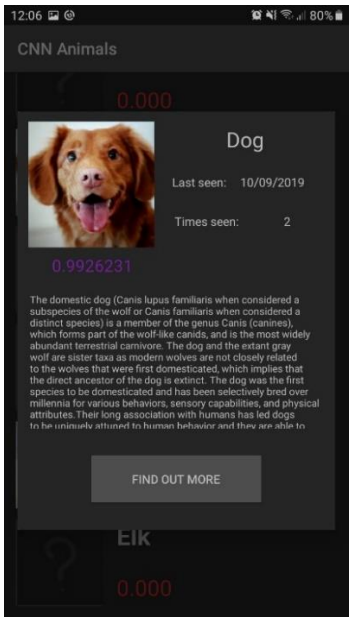
Snimljena slika psa vidi se među ostalim životinjama (Slika 5.4). Broj ispod svake životinje označuje aktivaciju ili sigurnost da životinja na slici pripada navedenoj klasi. Boja ovog teksta ovisi o vrijednosti koju prikazuje, zbog lakšeg vizualnog raspoznavanja rezultata. Kategorije su

sljedeće. od 0 do 0.4 je crvena, od 0.4 do 0.6 je narančasta, od 0.6 do 0.7 je žuta, od 0.7 do 0.8 je zelena, od 0.8 do 0.9 je plava, a od 0.9 do 1 je ljubičasta. Upitnik na slici 5.4 predstavlja početnu sliku svake klase životinja, koju će zamijeniti slike s kamere.



Slika 5.4 Lista životinja u bazi podataka

Kada se pritisne neki element iz liste sa slike 5.4, otvori se prozor koji prikazuje dodatne detalje o toj životinji (Slika 5.5).

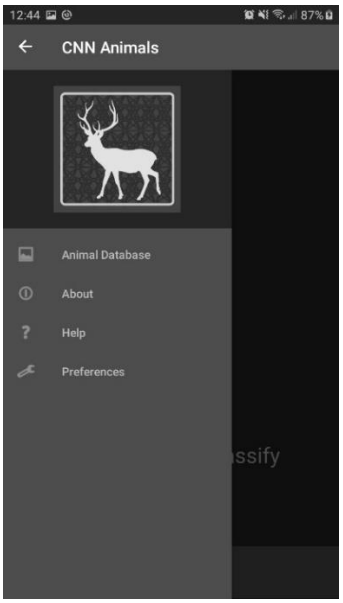


Slika 5.5 Detaljni prikaz informacija o životinji

Uz ime životinje i aktivaciju, na slici 5.5 piše datum kad je zadnji put životinja viđena, broj koliko je puta viđena te njezin kratki opis. Opis je povučen s internet stranice Wikipedia [28]. Pritisak na gumb **FIND OUT MORE** vodi na članak o toj životinji (Slike 5.6).

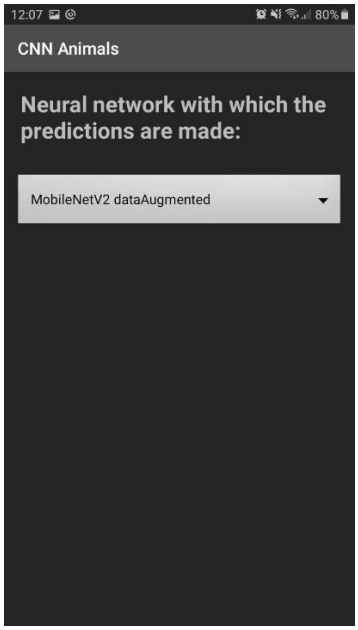
Slika 5.6 Wikipedia članak o životinji

Dodatno, na početnom zaslonu nalazi se navigacijski izbornik (slika 5.7) koji omogućava pristup bazi podataka, kratkim informacijama o aplikaciji, kratkim uputama za korištenje, te postavkama koje se nalaze na slici 5.8.



Slika 5.7 navigacijski izbornik

U postavkama na slici 5.8 može se odabrati koja se konvolucijska neuronska mreža koristi za klasificiranje slika. Ponuđene neuronske mreže su MobileNetV2 s proširenjem podataka, MobileNetV2 bez proširenja podataka te NASNetMobile s proširenjem podataka.



Slika 5.8 izbornik neuronskih mreža

5.2. Ispitivanje rada aplikacije na primjerima klasifikacije slika

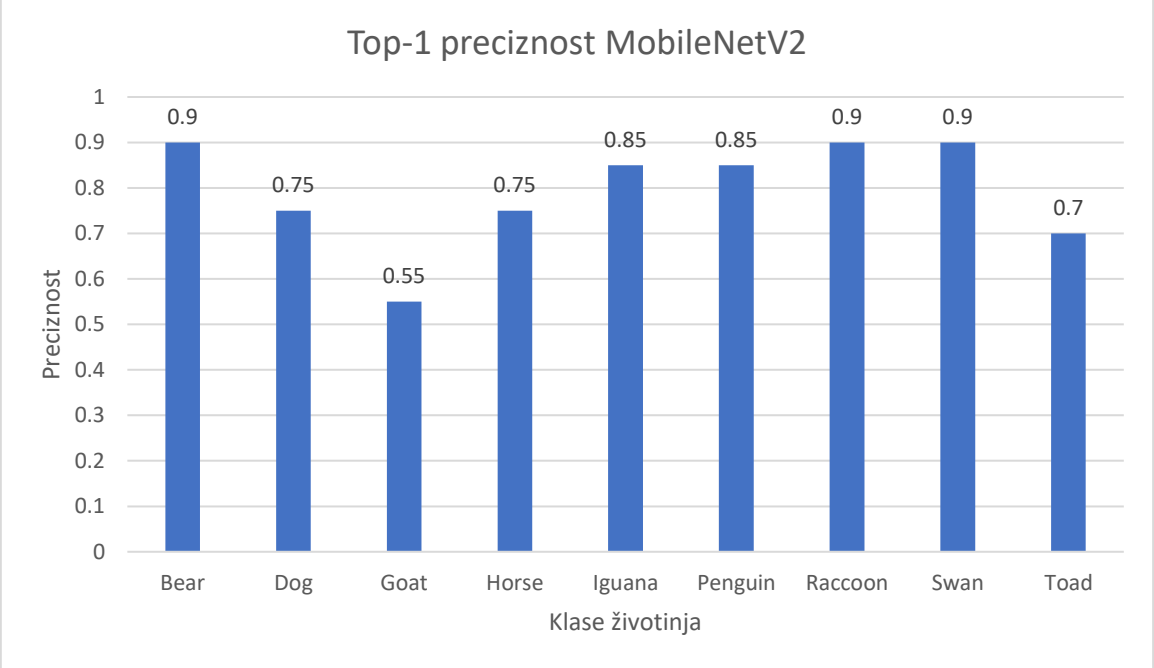
Android aplikacija ovog završnoga rada sadrži tri konvolucijske neuronske mreže naučene na slikama iz baze podataka Caltech-256. Klasifikacija slika snimljenih kamerom mobilnog uređaja se može obavljati sa sve tri konvolucijske neuronske mreže, a po potrebi se može u Android aplikaciju uključiti više neuronskih mreža. Primarna neuronska mreža je konvolucijska neuronska mreža MobileNetV2 s proširenom bazom podataka slika. Sekundarne i tercijarne neuronske mreže su konvolucijska neuronska mreža MobileNetV2 bez proširene baze podataka slika i konvolucijska neuronska mreža NASNetMobile s proširenom bazom podataka slika. Sve tri konvolucijske neuronske mreže imaju približno dobre rezultate klasifikacije slika. Njihovi rezultati se mogu vidjeti na tablici 5.1

	Preciznost	Težina
MobileNetV2 proširen	0.8604	0.5527
MobileNetV2 običan	0.8875	0.4202
NASNetMobile proširen	0.8697	0.4945

Tablica 5.1 rezultati neuronskih mreža

Rezultati iz tablice 5.1 upućuju na to da proširenje baze podataka slika uzrokuje manjom preciznošću i većom težinom konvolucijske neuronske mreže. Unatoč tome, proširenje baze podataka slika rezultira većom generalizacijom konvolucijske neuronske mreže, te u pravim uvjetima treba rezultirati neuronskom mrežom koja može precizno klasificirati širi spektar slika.

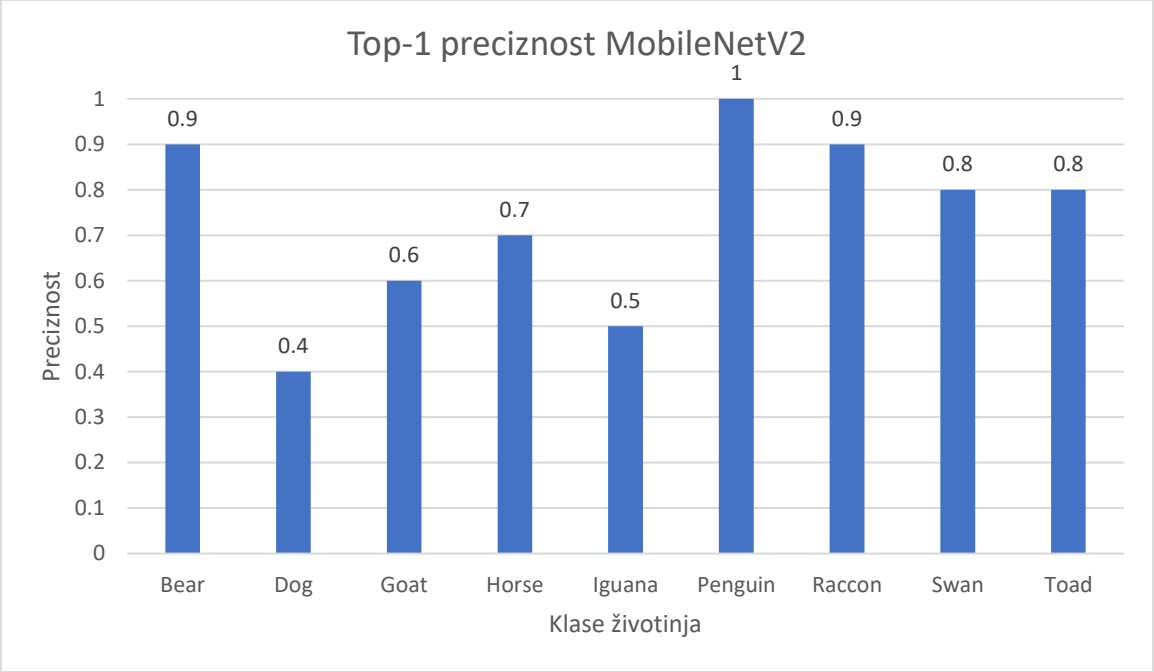
Iako klase u bazi podataka Caltech-256 slika nemaju jednak broj slika, njihov broj je približno 130 za svaku klasu. U svrhu testiranja uspješnosti neuronske mreže odvojeno je 20 slika za svaku klasu. Za mjerenje uspješnosti neuronske mreže koristi se *top-1* preciznost, s obzirom na to da je u ovom slučaju *top-1* i *top-5* preciznost jednaka vrijednost. Razlog tome je što se ovoj neuronskoj mreži ulazne slike ne mijenjaju, a ona daje jednak rezultat klasifikacije za istu sliku. *Top-1* preciznost označava vjerojatnost da se željeni rezultat pojavi prvi put kada neuronska mreža predviđa klasu određene slike. Na grafu 5.1 prikazani su rezultati testiranja neuronske mreže nad slikama nasumično odabranih devet klasa životinja iz baze podataka slika. Dodatno, slike su unesene direktno u neuronsku mrežu te se ona nikad nije učila na slikama na kojim se testira.



Graf 5.1 top-1 preciznost, slike unesene u neuronsku mrežu direktno; slike iz baze podataka slika

Podaci s grafa 5.1 prikazuju raspon vrijednosti preciznosti neuronske mreže koji je blizu očekivane vrijednosti od 0.86 u prosjeku za sve klase. Pojedine klase mogu odstupati od prosjeka zbog težih karakteristika koje sadrže, kao i sličnosti s drugim životinjama koje također mogu smanjiti preciznost klasificiranja slika. Na grafu 5.2 je prikazana preciznost neuronske mreže kod

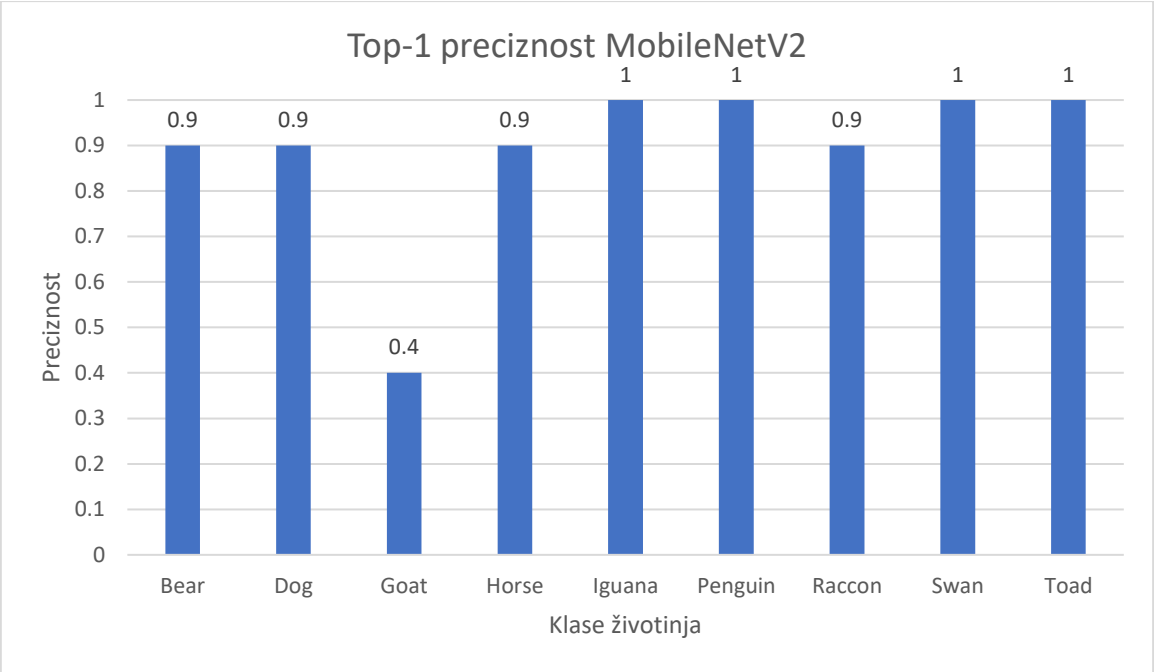
klasifikacije istih klasa životinja koje se nalaze na grafu 5.1. Korištene su slike iz iste baze podataka slika, ali ih se ovaj put učitava u neuronsku mrežu preko kamere mobilnog uređaja.



Graf 5.2 Top-1 preciznost, slike unesene u neuronsku mrežu preko kamere; slike iz baze podataka slika

Na grafu 5.2 se nalaze *top-1* preciznosti neuronske mreže. Uzeto je devet klasa životinja i deset uzoraka iz baze podataka slika po klasi životinje. Cilj ispita je utvrditi koliko kvaliteta kamere utječe na preciznost neuronske mreže.

Preciznosti neuronske mreže na grafu 5.2 su generalno lošije zbog učinka kvalitete kamere na kvalitetu slike. Taj negativni učinak je dodatno pogoršan jer su slike za testiranje aplikacije male rezolucije. S obzirom na to da se testiranje učinkovitosti neuronske mreže uključene u Android aplikaciju vrši tako da se kamerom snimi zaslon laptopa na kojemu se nalaze slike iz baze podataka Caltech-256 slika, mala rezolucija slika iz baze podataka slika znači da su slike prikazane na zaslonu laptopa niske kvalitete i malih dimenzija. Kamera dodatno pogorša već nisku kvalitetu slika iz baze podataka slika.. Dodatno, neke slike iz baze podataka slika loše prikazuju karakteristike klase kojoj pripadaju. Kada se snimaju slike veće rezolucije koje jasno prikazuju karakteristike životinje neuronska mreža ima veću preciznost, kao što se vidi na grafu 5.3.



Graf 5.3 Top-1 preciznost, slike unesene u neuronsku mrežu preko kamere, slike s interneta

Na grafu 5.3 nalaze se *top-1* preciznosti neuronske mreže kod klasifikacije ručno odabranih slika koje se nalaze na internetu, preciznije Google pretraživanje slika. Uzeto je devet klasa i deset uzoraka po klasi. Cilj ispita je utvrditi koliku je preciznost neuronske mreže kod klasifikacije životinja moguće ostvariti kada se aplikacija koristi na namijenjen način, tj. kada se uzimaju slike koje jasno prikazuju životinju i sve njene karakteristike. Većina klasi postiže postotak preciznosti od 90 ili više, što je zadovoljavajuće. Klasa Goat, tj. koza, odstupa znatno od ovog prosjeka jer neuronska mreža često klasificira slike smeđih koza kao klasu Elk, tj. wapiti ili kanadski jelen.

6. ZAKLJUČAK

U ovome završnome radu opisane su metode strojnog učenja, implementacije konvolucijskih neuronskih mreža, te je u sklopu rada izrađena Android aplikacija koja sadrži konvolucijsku neuronsku mrežu naučenu za klasificiranje 51 vrsti životinja. Učenje s prijenosom pokazala se kao idealna metoda za primjenjivanje naučenih konvolucijskih neuronskih mreža kao temelj vlastitih konvolucijskih neuronskih mreža uključenih u Android aplikaciju. Primarni razlog korištenja prijenosa učenja je nedostatak računalnih resursa za učenje vlastite konvolucijske neuronske mreže koja je dovoljno složena i sposobna za rješavanje problema klasifikacije životinja.

Konvolucijska neuronska mreža namijenjena klasifikaciji životinja naučena je pomoću biblioteke TensorFlow u okviru integriranog razvojnog okruženja JupyterLab koristeći programski jezik Python. Naučena konvolucijska neuronska mreža je dalje uključena u Android aplikaciju koristeći funkcije biblioteke TensorFlowLite, koja služi za pokretanje modela konvolucijskih neuronskih mreža naučenih u biblioteci TensorFlow na Android uređajima. Za razvoj Android aplikacije korišten je programski jezik Java u integriranom razvojnom okruženju Android Studio. Android aplikacija je izrađena po uzoru na arhitekturni predložak MVP koji znatno pomaže pri organizaciji i testiranju koda. Slike korištene kao ulazni podatci za konvolucijsku neuronsku mrežu dohvaćaju se koristeći kameru mobilnog uređaja čija je funkcionalnost integrirana u aplikaciju. Za implementaciju baze podataka koja sadrži podatke o klasama životinja koristi se biblioteka SQLite. Konačno. Za dohvaćanje opisa životinja s članaka pronađenih na internet stranici Wikipedia koristi se biblioteka Retrofit.

Stvorena aplikacija i učinkovitost uključenih konvolucijskih neuronskih mreža može se unaprijediti integracijom boljih konvolucijskih neuronskih mreža naučenih za klasifikaciju životinja. Dodatno, mobilna aplikacija može se poboljšati boljim dizajnom, ili pak novim mogućnostima koje mobilna aplikacije nudi. Mobilna aplikacija ne mora se nužno koristiti za klasificiranje životinja, već aplikacija predstavlja sučelje za konvolucijsku neuronsku mrežu te bi trebala raditi za rješavanje sličnih problema klasifikacije.

LITERATURA

[1] J. Brownlee, A Gentle Introduction to Computer Vision, Machine Learning Mastery, Ožujak 2019. , dostupno na: https://machinelearningmastery.com/what-is-computer-vision/ [Lipanj 2019].

[2] R. Demush, A Brief History of Computer Vision (and Convolutional Neural Networks), Hackernoon, Veljača 2019. , dostupno na: https://hackernoon.com/a-brief-history-of-computer-vision-and-convolutional-neural-networks-8fe8aacc79f3 [Lipanj 2019].

[3] Y. LeCun, C. Cortes i C. J. C. Burges, THE MNIST DATABASE of handwritten digits, dostupno na: http://yann.lecun.com/exdb/mnist/ [Rujan 2019].

[4] Augmented Startups, Support Vector Machine (SVM) - Fun and Easy Machine Learning, Youtube, Kolovoz 2017. , dostupno na: https://youtu.be/Y6RRHw9uN9o [Rujan 2019].

[5] S. Ray, Understanding Support Vector Machine algorithm from examples (along with code), Analytics Vidhya, Rujan 2017. , dostupno na: https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/ [Lipanj 2019].

[6] Wikimedia Foundation, Loss function, Rujan 2019. , dostupno na: https://en.wikipedia.org/wiki/Loss_function [Rujan 2019].

[7] Wikimedia Foundation, Chain Rule, dostupno na: https://en.wikipedia.org/wiki/Chain_rule [Rujan 2019].

[8] A. Pant, Introduction to Logistic Regression, Towards Data Science, dostupno na: https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148 [Rujan 2019].

[9] Wikimedia Foundation, Sigmoid function, dostupno na: https://en.wikipedia.org/wiki/Sigmoid_function [Rujan 2019].

[10] Wikimedia foundation, Bayes' theorem, dostupno na: https://en.wikipedia.org/wiki/Bayes%27_theorem [Rujan 2019].

[11] N. Kumar, Naive Bayes Classifiers, GeeksforGeeks, dostupno na: https://www.geeksforgeeks.org/naive-bayes-classifiers/ [Rujan 2019].

[12] R. Gandhi, Naive Bayes Classifier, Towards Data Science, Svibanj 2018. , dostupno na: https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c [Rujan 2019].

[13] S. Sawla, Linear Discriminant Analysis, Medium, Lipanj 2019. , dostupno na: https://medium.com/@srishtisawla/linear-discriminant-analysis-d38decf48105 [Rujan 2019].

[14] G. Seif, A Guide to Decision Trees for Machine Learning and Data Science, Towards Data Science, Studeni 2018. , dostupno na: https://towardsdatascience.com/a-guide-to-decision-trees-for-machine-learning-and-data-science-fe2607241956 [Rujan 2019].

[15] O. Harrison, A Guide to Decision Trees for Machine Learning and Data Science, Towards Data Science, Rujan 2018. , dostupno na: https://towardsdatascience.com/a-guide-to-decision-trees-for-machine-learning-and-data-science-fe2607241956 [Rujan 2019].

[16] D. A. Adeniyi, Z. Wei i Y. Yongquan, Automated web usage data mining and recommendation system using K-Nearest Neighbor (KNN) classification method, ScienceDirect, 2016. , dostupno na: https://www.sciencedirect.com/science/article/pii/S221083271400026X [Rujan 2019].

[17] T. Rekić, Unity Number Recognition, Studeni 2018. , dostupno na: https://github.com/tomislavrekic/Unity-Number-Recognition [Rujan 2019].

[18] Google, Transfer Learning Using Pretrained ConvNetsTensorFlow, dostupno na: https://www.tensorflow.org/beta/tutorials/images/transfer_learning [Rujan 2019].

[19] V. Bushaev, Adam — latest trends in deep learning optimization., Towards Data Science, Listopad 2018. , dostupno na: https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c [Rujan 2019].

[20] Y. LeCun, L. Bottou, Y. Bengio i P. Haffner, Gradient-Based Learning Applied to Document Recognition, Studeni 1998. , dostupno na: http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf [Rujan 2019].

[21] A. Krizhevsky, I. Sutskever i G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, 2012. , dostupno na: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks [Kolovoz 2019].

[22] M. D. Zeiler i R. Fergus, Visualizing and Understanding Convolutional Networks, Studeni 2013. , dostupno na: https://arxiv.org/pdf/1311.2901.pdf [Rujan 2019].

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vincent i A. Rabinovich, Going deeper with convolutions, Rujan 2014. , dostupno na: https://arxiv.org/pdf/1409.4842.pdf [Kolovoz 2019].

[24] K. Simonyan i A. Zisserman, Very deep convolutional networks for large-scale image recognition, Travanj 2015. , dostupno na: https://arxiv.org/pdf/1409.1556.pdf [Kolovoz 2019].

[25] K. He, X. Zhang, S. Ren i J. Sun, Deep Residual Learning for Image Recognition, Prosinac 2015. , dostupno na: https://arxiv.org/pdf/1512.03385.pdf [Kolovoz 2019].

[26] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov i L.-C. Chen, MobileNetV2: Inverted Residuals and Linear Bottlenecks, Ožujak 2019. , dostupno na: https://arxiv.org/pdf/1801.04381.pdf [Rujan 2019].

[27] S.-H. Tsang, Review: MobileNetV2 — Light Weight Model (Image Classification), Towards Data Science, Svibanj 2019. , dostupno na: https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c [Rujan 2019].

[28] Wikimedia Foundation, Wikipedia, dostupno na: https://www.wikipedia.org/ [Rujan 2019].

[29] Google, Performance benchmarks, 2019. , dostupno na: https://www.tensorflow.org/lite/performance/benchmarks [Kolovoz 2019].

[30] G. Griffin, A. Holub i P. Perona, Caltech 256, Studeni 2006. , dostupno na: http://www.vision.caltech.edu/Image_Datasets/Caltech256/ [Rujan 2019].

[31] A. Leiva, MVP for Android: how to organize the presentation layer, antonioleiva, dostupno na: https://antonioleiva.com/mvp-android/ [Rujan 2019].

[32] Python Software Foundation, What is Python? Executive Summary, 2019. , dostupno na: https://www.python.org/doc/essays/blurb/ [June 2019].

[33] P. Carbonnelle, PYPL PopularitY of Programming Language, Rujan 2019. , dostupno na: http://pypl.github.io/PYPL.html [Rujan 2019].

[34] Anaconda, Anaconda Documentation, 2019. , dostupno na: https://docs.anaconda.com/anaconda/ [Lipanj 2019].

[35] Project Jupyter, Jupyter, 2019. , dostupno na: https://jupyter.org/ [Lipanj 2019].

[36] Google, TensorFlow, dostupno na: https://www.tensorflow.org/ [Rujan 2019].

[37] Keras, Keras: The Python Deep Learning library, dostupno na: https://keras.io/ [Rujan 2019].

[38] Oracle, Java, 2019. , dostupno na: https://www.java.com/en/ [Rujan 2019].

[39] D. R. Hipp, About SQLite, dostupno na: https://www.sqlite.org/about.html [Rujan 2019].

[40] L. Vogel, S. Scholz i D. Weiser, Using Retrofit 2.x as REST client - Tutorial, vogella, Lipanj 2018. , dostupno na: https://www.vogella.com/tutorials/Retrofit/article.html [Rujan 2019].

ŽIVOTOPIS

Tomislav Rekić rođen je 01.06.1997. godine u Osijeku, Hrvatska. Pohađao je osnovnu školu Fran Krsto Frankopan u Osijeku, te poslije toga upisao je Elektrotehničku i prometnu školu Osijek, smjer elektrotehnika. Trenutno studira na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek na sveučilištu Josipa Jurja Strossmayera u Osijeku, gdje je upisao preddiplomski studij elektrotehnika te je nakon prve godine prešao na smjer računarstvo. Bavio se strojnim učenjem, posebice neuronskim mrežama; razvojem Android aplikacija te razvojem igara u Unity, za što je pohađao praksu u tvrtki ReRoot d.o.o. u trajanju od mjesec dana.

SAŽETAK

Cilj ovoga rada je predložiti algoritam strojnog učenja pogodan za prepoznavanje slika životinja, izraditi primjenjiv model strojnog učenja, te ga ugraditi u mobilnu aplikaciju koja će na temelju snimljene slike omogućiti prepoznavanje životinja i ažuriranje postojeće ugrađene baze podataka životinja rezultatima klasifikacije. Dodatno, cilj rada je opisati popularne metode strojnog učenja, opisati neke od implementacija odabranog algoritma strojnog učenja, dati pregled značajki mobilne aplikacije i ispitati uspješnost odabranog algoritma strojnog učenja. Mobilna aplikacija je razvijena u razvojnom okruženju Android Studio i napisana u programskom jeziku Java, dok je algoritam strojnog učenja implementiran u razvojnom okruženju JupyterLab koristeći biblioteku TensorFlow i napisan u programskom jeziku Python. Baza podataka životinja uključena u mobilnu aplikaciju implementirana koristeći biblioteku SQLite, a opisi klase životinja dohvaćeni koristeći biblioteku Retrofit.

Ključne riječi: Android, klasifikacija životinja, konvolucijska neuronska mreža, strojno učenje, TensorFlow

ABSTRACT

The goal of this thesis is to suggest a machine learning algorithm suitable for recognizing pictures of animals, create an applicable machine learning model and to embed it in the mobile application which will on the basis of a captured image allow recognition of animals and updating of an existing embedded database with the results of the classification. The thesis describes popular machine learning methods and some of the implementations of the chosen machine learning algorithm. Furthermore, the goal of this thesis is to describe the popular machine learning methods, describe some of the implementations of the chosen machine learning algorithm, give an insight into the characteristics of the mobile application and to test the success of the chosen machine learning algorithm. The mobile application is developed inside the Android Studio IDE and written in the Java programming language, while the machine learning algorithm is implemented in the JupyterLab IDE using the TensorFlow library and written in the Python programming language. The animal database included in the mobile application is implemented using the SQLite library and the descriptions of animal classes were retrieved using the Retrofit library.

Key words: Android, animal classification, convolutional neural network, machine learning, TensorFlow

PRILOZI

Prilog 1. Dokument i pdf završnog rada

Prilog 2. Programski kod projekta razvoja mobilne aplikacija i postupka strojnog učenja