FlagAFlag - razvoj aplikacije za identifikaciju zastava

Zatezalo, Ivan

SVEUČILIŠTE U ZAGREBU
FILOZOFSKI FAKULTET
ODSJEK ZA INFORMACIJSKE I KOMUNIKACIJSKE ZNANOSTI
Ak. god. 2020./2021.

Ivan Zatezalo

FlagAFlag – razvoj aplikacije za identifikaciju zastava

Završni rad

Mentor: dr. sc. Ivan Dunđer, doc.

Zagreb, svibanj 2021.

Izjava o akademskoj čestitosti

Izjavljujem i svojim potpisom potvrđujem da je ovaj rad rezultat mog vlastitog rada koji se temelji na istraživanjima te objavljenoj i citiranoj literaturi. Izjavljujem da nijedan dio rada nije napisan na nedozvoljen način, odnosno da je prepisan iz necitiranog rada, te da nijedan dio rada ne krši bilo čija autorska prava. Također izjavljujem da nijedan dio rada nije korišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

_____

(potpis)

Sadržaj

1. Uvod

Ljudi su oduvijek bili opčinjeni načinom kojim druga bića razmišljaju i promatraju svijet. Ova opčinjenost je ujedno stvorila mnoga umjetna „bića“ koja iz dana u dan postaju sve sličnija ljudima. Jedan od takvih nedavnih izuma pod nazivom GPT-3 je znatno pomaknuo granice umjetne inteligencije. Sam pojam umjetne inteligencije veže se uz znanstvenika po imenu Alan Turing. Unatoč njegovoj briljantnosti i genijalnom umu, zbog tehničkih ograničenosti računala u 1950-im njegove ideje o umjetnoj inteligenciji se tek nešto kasnije počinju ostvarivati (Anyoha, 2017). Jedan od načina izgradnje umjetne inteligencije je korištenje istraživačkog područja strojnog učenja. Pojam strojnog učenja može se razumjeti kao podvrsta umjetne inteligencije koje funkcionira uz pomoću algoritama (Raschka, n.d.). Strojno učenje danas ima veliku primjenu, primjerice u prepoznavanju afektivnih stavova u govoru (Lugović et al., 2016) i tekstu (Dunđer & Pavlovski, 2019a), vektorskoj analizi korpusa (Dunđer & Pavlovski, 2019b) i sl. U kontekstu hrvatskog jezika te alata i resursa za hrvatski jezik, strojno učenje istraženo je s posebnim fokusom na strojno prevođenje (Dunđer, 2020; Dunđer et al., 2020) te klasifikaciju dokumenata (Dunđer et al., 2015).

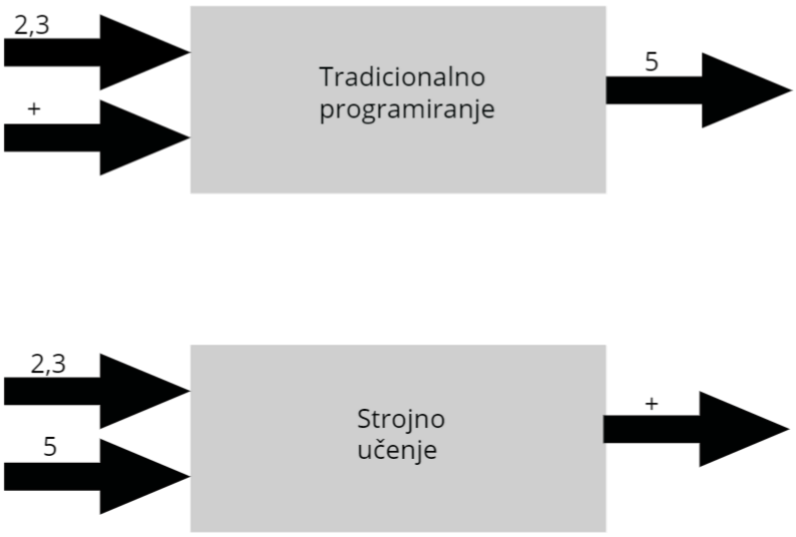
Glavna motivacija za pisanje ovog završnog rada je interes za umjetnu inteligenciju te neprestan razvoj strojnog učenja. Stoga je autor odlučio razviti vlastitu mobilnu aplikaciju koja će upravo uz pomoć raznih algoritama moći točno klasificirati svjetske zastave. Kako bi razvoj aplikacije bio moguć, prvo je bilo potrebno razumjeti pojmove vezane uz strojno učenje te razviti vlastitu inteligenciju koja će moći izvršiti dani zadatak.

1.1. Pristup strojnom učenju

Općenito govoreći, strojno učenje je grana umjetne inteligencije čiji je cilj izgradnja aplikacija na temelju podataka danih programu te poboljšanje rezultata kroz određeno vrijeme (IBM Cloud Education, 2020). Pristup strojnome učenju znatno je drukčiji od onoga u tradicionalnom programiranju. Za uspješno napisan tradicionalni program potrebno je dati podatke te određena pravila kako bi program dao izvjestan rezultat.

U strojnome učenju rezultat se koristi u skupu s podacima kako bi se dobila pravila na temelju kojih program funkcionira (Parthasarathy, 2019). Izrečeno je najjednostavnije ilustrirati

zbrajanjem dvaju brojeva. U tradicionalnome programiranju programer mora samostalno propisati pravila kako bi neki podaci dali željeni rezultat. Primjerice ako se uzmu brojevi 3 i 2 te im se pridruži operacija zbrajanja, rezultat programa bit će 5. U strojnome učenju, pak, programu će se primjerice dati podaci 2, 3 i 5 te na temelju danih podataka će program morati dokučiti kako doći do pravila koja daju rezultat 5. Grafički prikaz navedenoga nalazi se na slici 1.



Slika 1. Pristup strojnom učenju

U tradicionalnom programiranju problemi se rješavaju matematički i logično. Pristup strojnom učenju temelji se na statistici i eksperimentiranju. Također, važno je napomenuti da u strojnome učenju računalo znatno drukčije interpretira podatka i rezultate nego čovjek. Primjerice riječ „pas“ računalo će interpretirati kao [0.78, 0.3, 0.9], a neku drugu riječ poput „voda“ kao [0.1, 0.33, 0.03, 0.5]. Iz navedenoga moguće je uočiti zašto je ponekad teško razumjeti kako to točno računalo shvaća i razumije navedene podatke, međutim, nije nemoguće razumjeti i procijeniti dobivene rezultate (Google Developers, n.d.)

U strojnom učenju srž svakoga programa čini model. Model je zapravo dio programskog koda koji na temelju podataka može ostvariti pretpostavke nad različitim uzorcima. Proces

učenja nad podacima naziva se treniranje, a tijekom kojeg se moraju odabrati različiti algoritmi kako bi se model što bolje optimizirao i prilagodio na vrstu podataka. Nakon što je proces treniranja završen, modelu se daju novi podaci kako bi mogao pretpostaviti vrstu novih podatka (Cowley & Radich, 2019). Primjerice, u slučaju treniranja modela s fotografijama zastava, model bi trebao imati mogućnost točno klasificirati nove, tj. za vrijeme treniranja neviđene, fotografije zastava.

1.2. Strojno učenje kao eksperiment

Jedan od načina razumijevanja strojnoga učenja jest da se cijeli proces može shvatiti kao eksperiment u kojemu se mnoge varijable mijenjaju nakon svakog testiranja kako bi se model što bolje prilagodio podacima te izvršio svoj zadatak. Poput bilo kojeg eksperimenta proces je zanimljiv, uzbudljiv, ali ponekad i frustrirajuć. No, kada u konačnici model postigne mogućnost točnog klasificiranja fotografije, konačni rezultat postaje vrijedan uloženog truda i vremena (Google Developers, n.d.).

Izrada vlastite aplikacije uvjetovala je definiranje hodograma. Cijeli proces može se raspodijeliti na nekoliko koraka od kojih svaki ima svoje zasebne probleme i izazove. Koraci u izradi ove aplikacije navedeni u tablici 1.

Tablica 1. Hodogram istraživanja i izrade aplikacije

Korak	**Postupak**
Cilj istraživanja	Razvoj aplikacije za identifikaciju svjetskih zastava koja će s visokom točnošću moći procijeniti o kojoj se državi radi.
Hipoteza	Položaj zastave utječe na točnost modela.
Prikupljanje i obrada podataka	Prikupljanje slika zastava te njihova obrada.
Testiranje hipoteze	Razvoj modela primjenom slika prikupljenih u prethodnom koraku.

Analiza dobivenih rezultata	Je li model uspješno klasificirao zastave?
Zaključak na temelju dobivenih rezultata	Ako model nije dovoljno precizan, na koji način se može poboljšati te je li potrebno promijeniti način na koji su podaci obrađeni i prikupljeni?
Ponovna procjena modela	Na temelju dobivenih rezultata je li model potrebno ponovno trenirati ili je preciznost zadovoljavajuća?
Izgradnja aplikacije	Izgradnja aplikacija na temelju *widgeta* potrebnih za identifikaciju zastava.
Implementacija modela unutar aplikacije	Problemi vezani uz samu implementaciju modela unutar aplikacije.
Testiranje i procjena aplikacije.	Je li putem aplikacije moguće uspješno klasificirati zastavu?

S postavljenom hipotezom i ciljem istraživanja sljedeći korak je proces prikupljanja i obrade podataka, što će upravo biti u fokusu sljedeće cjeline ovoga rada.

2. Prikupljanje podataka za potrebe strojnog učenja

Jedan od mnogih izazova u strojnom učenju predstavlja prikupljanje podataka. Sam izazov proizlazi iz potrebe da se podaci moraju pripremiti, analizirati, obraditi te prilagoditi modelu za odabrano područje. Ovo često može predstavljati velik problem jer se strojno učenje temelji na podacima i njihovoj kvaliteti te u slučaju da je područje istraživanja relativno novo jednostavno će doći do manjka kvalitetnih podataka (Whang et al., 2019). Nešto poznatiji modeli strojnog učenja poput onoga za detekciju objekata ili prepoznavanja ljudskoga lica temelje se na podacima prikupljenih preko nekoliko desetljeća te su oni zbog toga i vrlo uspješni. Manjak kvalitetnih resursa je upravo zbog toga i najveća prepreka u strojnome učenju te je jedan od razloga zašto projekti vezani uz strojno učenju nisu izvedivi ili jednostavno propadnu (MV, 2020). Stoga postoji velika potreba za javno dostupnim, kvalitetnim podacima i metodama za njihovu izradu, pogotovo sada u eri „Big Data“ (Whang et al., 2019).

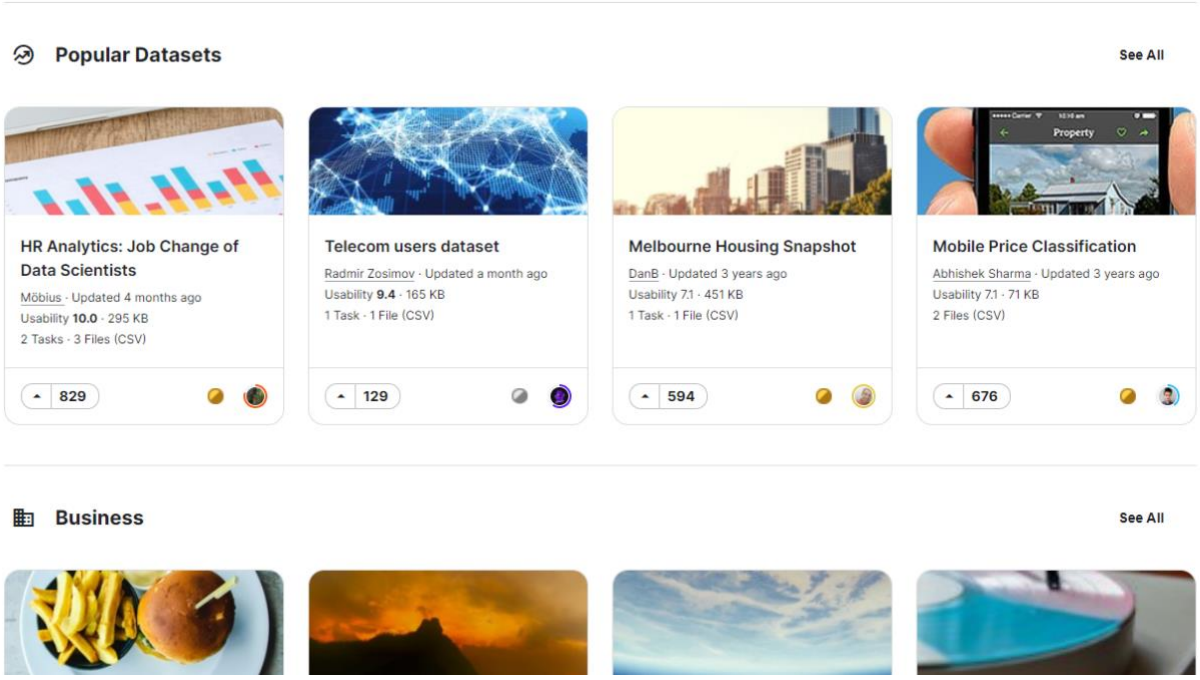
2.1. Načini prikupljanja podataka

Prikupljanje podataka može se odviti na tri različita načina. Prvi način prikupljanja podataka odnosi se na otkrivanje već gotovih i okupljenih podataka (Whang et al., 2019). Naravno, ti podaci moraju biti dostupni te indeksirani kako bi ih korisnik mogao koristiti. Neke od kompanija poput *Kaggle*[1] upravo to omogućuju svojim korisnicima. Kaggle omogućuje korisnicima da lako pronalaze, dijele i pretražuju podatke te tako grade kvalitetne i uspješne modele. Javna dostupnost ovako velike količine podataka omogućuje bilo kojem pojedincu aktivno sudjelovanje te interakciju sa stručnjacima u polju strojnoga učenja i podatkovne znanosti. Odabir vrste podatka ovisi o potrebama modela te samog istraživanja. Za prepoznavanje određenih uzoraka na fotografiji ili ljudskoga lica koriste se slike ili video zapisi, tj. vrsta podataka je vizualna.

S obzirom na to da se ovaj rad temelji na modelu za prepoznavanje zastava, autor rada fokusirat će se na prikupljanje podataka vizualnoga oblika. U slučaju primjene već gotovih skupova podataka, važno je razumjeti je li određen skup podataka podoban za određeni model te

[1] Kaggle: Your Machine Learning and Data Science Community (:https://www.kaggle.com/)

je li sam skup podataka označen. Neki od primjera skupova podataka prisutni su na slici 2 (Kaggle, n.d.).



Slika 2. Primjer zbirke podataka na web stranici Kaggle

Drugi način prikupljanja podataka proširuje prvi pristup tako da se originalni skup podataka dodatno razvija i prilagođava kako bi se podaci što bolje prilagodili vrsti istraživanja (Whang et al., 2019).

U slučaju usmjeravanja na manju domenu poput tematike zastava, vrlo je velika vjerojatnost da neće postojati javno dostupna zbirka podataka već će korisnik sam morati okupiti ili razvijati vlastite podatke. U ovom radu koristi se treći i zadnji pristup, u kojemu se razvija vlastita zbirka podataka (Whang et al., 2019). Naravno, dodatna prepreka u odabiru vrste podataka jest njegova primjena u stvarnom svijetu. Pri koraku prikupljanja podataka vrlo je važno razumjeti i razvijati zbirku podataka iz perspektive korisnika te uzeti u obzir razne slučajnosti i prepreke na koje će korisnik potencijalno naići. S obzirom na to da je cilj ovog završnog rada razviti aplikaciju koja će pomoći korisniku klasificirati zastavu, važno je prepoznati kakve će upite korisnik prosljeđivati samoj aplikaciji.

Slika 3. Primjer kvalitetne fotografije za razvijanje modela

Slika 4. Primjer fotografije koja nije idealna za model

Fotografije poput slike 3 (Williams-Ellis, n.d.) su bolji primjer vrste podataka nego slika 4 (Depositphotos, n.d.) zbog toga što je realističniji primjer u kojemu će se aplikacija koristiti. Naravno, dobro razvijen model moći će prepoznati i fotografiju sa desne strane zbog samih sličnosti s podacima unutar modela. Iz perspektive korisnika, važno je naglasiti da fotografije neće biti savršene te će možda sadržavati različite objekte, različitu razinu svjetlosti, rotaciju i slično. Dobar model je onaj koji u svojim testnim podacima sadrži podatke s raznim karakteristikama jer je uračunao nepredvidljivost budućih unosa korisnika. To je vrlo važno jer je cilj strojnog učenja razvoj robusnog modela koji će moći u budućnosti točno procesirati zahtjeve korisnika te dati točan rezultat na temelju podataka koje potencijalno mogu sadržavati i puno šuma.

Jedan od mogućih pristup prikupljanja fotografija zastava je vlastito fotografiranje s različitom svjetlinom, rotacijom i slično. Problem ovoga pristupa jest što je dugotrajan i neefikasan. Potrebno bi bilo fotografirati preko tisuće slika zastava s različitim kamerama jer se mora uračunati u obzir i kvaliteta kamere kao moguća prepreka kod korisnika aplikacije u budućnosti. Način na koji će se prikupljati podaci za potrebe ovog istraživanja obrađen je u sljedećem dijelu ovoga rada.

2.2. Prikupljanje vlastitih podataka za strojno učenje

Jedan od načina razvijanja vlastite zbirke podataka jest preuzimanje fotografija s interneta. Dodatan problem je način na koji prikupiti veliku količinu podataka prema vlastitim potrebama. Poput fotografiranja, preuzimanje svake fotografije zasebno dugotrajan je proces te je stoga izrađena Python skripta, tj. program koji će cijeli proces automatizirati te tako smanjiti potrebnu količinu vremena za preuzimanje podataka s interneta. Skripta je skup naredbi u datoteci koje se mogu pokrenuti kao program pomoću naredbenog retka (engl. command line) ili interpretera (Ramos, n.d.). S obzirom na to da Google sadrži veliku količinu dobavljivih fotografija odlučeno je napraviti skriptu koja će moći preuzeti fotografije različitih zastava sa Google slika. Prvo je bilo potrebno odlučiti koje će se države koristiti te koja će države moći biti identificirane pomoću aplikacije. Za razvoj prve inačice ove aplikacije uzet je popis 48 država koje se nalaze na Europskome području. Naravno, idealno bi bilo napraviti aplikaciju koja bi sadržavala sve države svijeta, međutim, veličina i sadržaj samoga projekta bi bili preveliki za potrebe ovoga rad. Popis država izrađen je u Microsoft Excelu te prikazan na slici 5.

#	Država
1	Albania
2	Andora
3	Armenia
4	Austria
5	Azerbaijan
6	Belarus
7	Belgium
8	Bosnia and Herzegovina
9	Bulgaria
10	Croatia
11	Cyprus
12	Czech Republic
13	Denmark
14	Estonia
15	Finland
16	France
17	Georgia
18	Germany
19	Greece
20	Holy See
21	Hungary
22	Iceland
23	Ireland
24	Italy
25	Kazakhstan
26	Kosovo
27	Latvia
28	Liechtenstein
29	Lithuania
30	Luxembourg
31	Macedonia
32	Moldova
33	Monaco
34	Montenegro
35	Netherlands
36	Poland
37	Portugal
38	Russia
39	San Marino
40	Serbia
41	Slovakia
42	Slovenia
43	Spain
44	Sweden
45	Switzerland
46	United Kingdom
47	Ukraine

Slika 5. Tablica s državama prisutnim u modelu

2.2.1. Razvoj skripte za prikupljanje podataka

Prvi korak u razvoju skripte je pretvaranje Microsoft Excel tablice u Pythonu čitljiv oblik. Pretvaranjem Excel tablice u listu omogućuje se Pythonu čitanje vrijednosti unutar tablice. Lista omogućuje nizanje različitih vrijednost koje su odvojene zarezom. Pretvorba Excel tablice omogućena je uz pomoć paketa *pandas*. Nakon što je napravljena lista, njen prikaz se može vidjeti uz pomoć funkcije *print()*. U sljedećem koraku implementirana je for petlja koja se izvršava sve dok se ne prođu vrijednosti unutar liste. Ova petlja će omogućiti da se za svaku zastavu napravi zasebna mapa te se u istoj spremaju fotografije. U slučaju da se dogodi neka pogreška Python će ispisati „Pogreška!“. Programski kod za navedeno prisutan je na slici 6.

```
14  import pandas as pd
15
16  tablica = pd.read_csv("testiranje.csv")
17  lista = tablica["Država"].tolist()
18
19  print(lista)
20
21  for x in lista:
22      drzava=x
23      try:
24          os.mkdir(drzava)
25      except:
26          print("Pogreška!")
```

Slika 6. Programski kod vezan uz pretvorbu Microsoft Excel tablice

Sljedeći korak unutar for petlje izvršen je uz pomoć paketa *Selenium* i *Beautiful Soup*. Selenium se koristi za kontrolu web preglednika i izvođenje automatizacije preglednika (GeeksforGeeks, 2020). Uz pomoću Beautiful Soup paketa korisnik može dohvatiti podatke iz HTML i XML datoteka (Richardson, 2020). Kombinacijom ova dva paketa moguće je razviti program koji će automatski dohvatiti vrijednosti Google web preglednika. Nakon implementacije

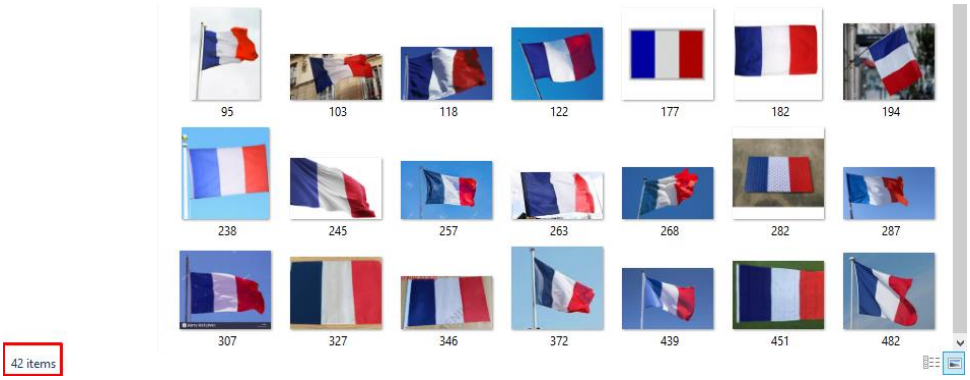
navedenih paketa za svaku je vrijednost iz prethodno napravljene liste automatiziran postupak unosa u Google. Nakon što for petlja dohvati vrijednost iz liste, Selenium otvara web preglednik, točnije stranicu Google slike, te u polje za pretraživanje automatski upisuje vrijednost iz liste. Nakon dohvaćenih podataka za svaku vrijednost, preuzimanje fotografija se izvršava pomoću paketa *urllib*. Završetkom petlje i samog programa dobivene su mape pripadajućim imenom države. S obzirom na temu zastava prosječna količina fotografija u svakoj mapi iznosi oko 350. Rezultat ovoga postupka je zbirka od preko 16000 fotografija. Djelić zbirke fotografija za zastavu Njemačke prisutan je na slici 7.

Slika 7. Primjer dobavljene količine fotografija

2.3. Odabir podataka

Jedan od najvažnijih procesa razvijanja vlastite zbirke podataka je selekcija podataka dobivenih putem Python skripte. S obzirom na to da je preuzeto otprilike 350 slika za svaku državu velika je vjerojatnost da neke fotografije neće moći biti korištene u razvoju modela zbog neodgovarajuće kvalitete i/ili sadržaja informacija. Kao što je već spomenuto, vrlo je važno napraviti realističnu bazu podataka kako bi sam model imao što veću preciznost prilikom primjene u stvarnom svijetu.

Rezultat odabira podataka od preko 16000 fotografija bio je iznenađujuć. Prosječni broj fotografija po zastavi je s 350 pao na svega 40. Većina fotografija je bila vrlo niske kvalitete ili pak sama fotografija uopće nije sadržavala traženu zastavu. Primjer odabrane zbirke vidljiv je na slici 8.



Slika 8. Količina podatka nakon procesa odabira podataka

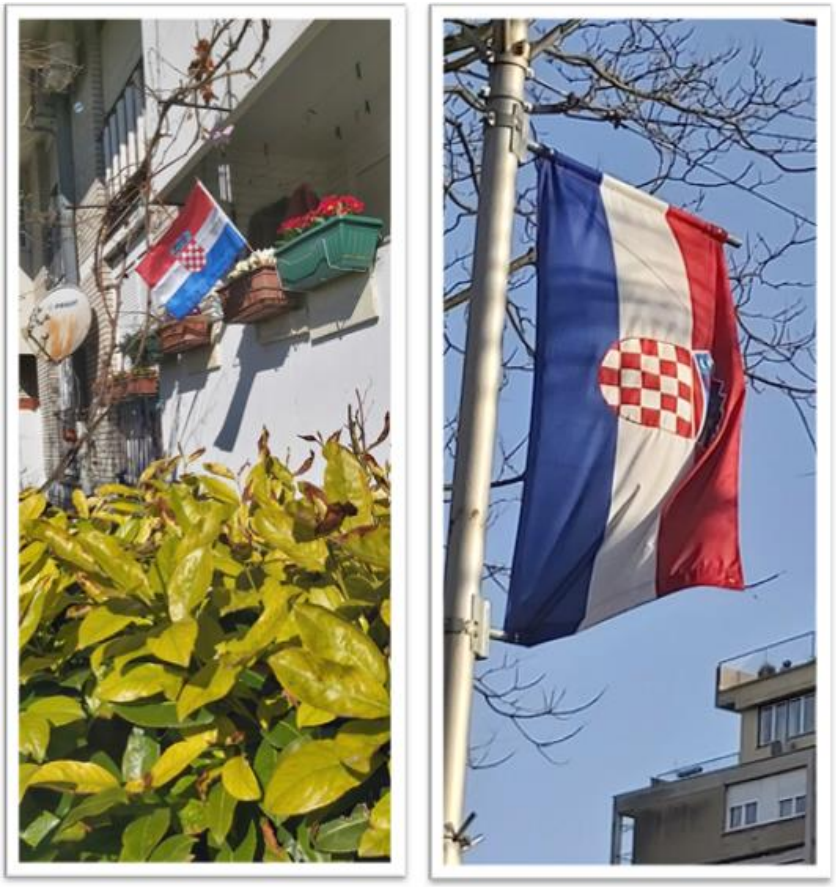
Općenito, u strojnom učenju kroz mnoga istraživanja i eksperimentiranje dokazano je kako modeli s većim brojem podataka u pravilu pokazuju znatno bolje rezultate, međutim, jednako je važno da su podaci kvalitetni i slični onima u stvarnome svijetu. Za što bolji model potrebno je imati dobar omjer kvalitete i količine podataka. Primjerice, ako se strojno učenje izvršava na temelju niske kvalitete podataka, model će naučiti krive osobine te će tako njegova efikasnost u stvarnome svijetu biti vrlo niska (Appen, 2018).

S obzirom na to da aplikacija izrađena za potrebe ovog završnog rada „FlagAFlag“ sadrži sveukupno 48 različitih klasa, tj. 48 grupa zastava, 40 fotografija po državi (zastavi) je vrlo mali broj za efikasan model. Sam broj podataka pri razvijanju vlastitog modela nije definiran te je često potrebno eksperimentiranje kako bi se utvrdila dostatna brojka potrebnih podataka za robustan model, međutim, postoji nepisano pravilo od 1000 slika po klasi u slučaju razvijanja vlastitog modela. Sam broj 1000 proizlazi iz izvornog izazova za klasifikaciju slika. Naime, ova brojka je bila zadovoljavajuća u izradi ranih klasifikatora poput AlexNeta te se na temelju toga i preporučuje brojka 1000 (Warden, 2017) . Iz navedenog razloga odlučeno je proširiti zbirku na nekoliko stotina slika po klasi. Iako je taj broj i dalje malen postoje daljnji načini na koje se zbirka može dodatno proširiti, no više tome bit će rečeno u poglavlju 3.4.

2.4. Obrada podataka

Prema postavljenoj hipotezi, položaj zastava ima utjecaj na samu točnost i preciznost modela. Istraživanjem utvrđeno je da se zastave učestalo pojavljuju u okomitom položaju ili se pak zrcale zbog utjecaja vjetra. Zbog toga su izrađene dodatne dvije skripte koje će generirati kopije zastava u okomitom i zrcalnom položaju. Zbog izrade dodatnih kopija slika prosječni broj fotografija po

klasi zastave popeo se na 120. Ova brojka nije blizu preporučene od 1000, međutim, dodavanje dodatnih fotografija u različitim položajima će definitivno poboljšati sam model te ga prilagoditi za stvarnu primjenu. Na slici 9 moguće je vidjeti neke od mogućih položaja zastava u stvarnom svijetu.



Slika 9. Položaji zastave u stvarnom svijetu

2.4.1. Razvoj skripte za rotaciju i zrcaljenje fotografija

Skripta za rotaciju i zrcaljenje fotografija je razvijena u Pythonu uz pomoć paketa *Pillow*. Paket Pillow dodaje Python interpreteru mogućnost obrade slike (Clark, 2021). Iako je sam proces zrcaljenja i promjene položaja moguće obaviti i ručno, na ovaj način dolazi do znatne uštede vremena. Primjerice, skripta za rotaciju će proći svaku zastavu u mapi te je zarotirati za 90 stupnjeva kako bi je zarotirala u okomit položaj. Nakon rotacije fotografija se sprema pod novim

imenom kako bi originalna fotografija ostala sačuvana, tj. nepromijenjena. Rezultat dobiven putem skripte prikazan je na slici 10. Ovaj postupak se ponavlja i u skripti za zrcaljenje te je nakon ova dva postupka dobivena zbirka od otprilike 6000 fotografija.

Slika *10.* Lijevo*:* originalna fotografija (The Flag Shop, n.d.a), u sredini*:* fotografija nakon okomite rotacije, desno*:* fotografija nakon zrcaljenja

2.5. Problem sličnosti podataka

Gledajući mnoge zastave svijeta vidljivo je da su korišteni razni simboli te boje, međutim, jednako lako je i uočljivo da postoje neke sličnosti među njima. Ove sličnosti su češće kod država koje se nalaze na istom geografskom području. Neke sličnosti u dizajnu često predstavljaju istu kulturnu, političku ili religijsku baštinu, primjerice, slavenske države poput Hrvatske, Slovenije, Srbije, Češke i Slovačke (Lee, 2017). Ove sličnosti mogu predstavljati znatan problem u slučaju vrlo male zbirke podataka te će zbog toga model netočno klasificirati zastavu. Sličnosti između slavenskih zastava moguće je uočiti na slici 11.

Slika 11. Sličnosti između zastava Slovenije (Dreamstime, n.d.) i Slovačke (Tokopedia, n.d.)

Najveći problem pri razvijanju ove aplikacije predstavljale su države poput Francuske, Luksemburga i Nizozemske. Rotirajući zastavu Nizozemske postaje sve teže uočiti razlike između nje i Francuske. Moguće rješenje ovoga problema je povećanje same zbirke fotografija ili znatna readaptacija modela, međutim, čak i s daljnjim modifikacijama modela identifikacija ovih dviju država ponekad ostaje problematična. Jedan od takvih primjera nalazi se na slici 12. Daljnjim istraživanjem utvrđeno je da postoji vrlo velik broj sličnih zastava te kao što je već spomenuto potrebna je što veća zbirka podataka, ali za manje područje istraživanja poput identifikacije zastava ovo ostaje problem jer manjak kvalitetnih podataka znatno narušava preciznost modela.

Slika 12. Zastava Nizozemske u okomitom položaju (The Flag Shop, n.d.b)

3. Razvoj modela za strojno učenje

Nakon izrade vlastite zbirke podataka, sljedeći korak u razvijanju ove aplikacije je razvoj modela. Prije samog razvoja modela vrlo je važno razumjeti način na koji računala procesiraju te razumiju fotografije u području strojnoga učenja. Sam proces identifikacije za ljude nije problematičan. Hodajući ulicom lako je razumjeti okolinu te obližnje objekte u istome trenutku, međutim, za računalo ovo je znatan problem. Kao što je prije spomenuto tijekom obrade podataka, ako se zastava rotira za svega 90 stupnjeva, računalo znatno teže prepoznaje te shvaća navedeni objekt. Računalo lako može razumjeti zarotiranu zastavu kao novi objekt te tu leži sam problem (Escrivá et al., 2018). Zbog toga je vrlo važno imati što veću količinu podataka kako bi računalo moglo predvidjeti što više scenarija iz stvarnoga svijeta.

Računalo vidi svaku fotografiju kao skup nula i jedinica. Ove nule i jedinice prikazane su u obliku piksela, a to su najmanje jedinice svake fotografije. Pri fotografiranju, slika se sprema kao kombinacija različitih piksela. Kombinacija može biti crno-bijela ili u boji. Pri identifikaciji zastava potrebna je fotografija u boji koja sadrži tri kanala, tj. crveni, zeleni i plavi kanal. Prikaz vrijednosti jednog piksela prisutan je na slici 13. Svaki piksel u fotografiji sadrži vrijednosti između 0 i 255, a te se vrijednosti moraju pretvoriti u biranu kako bi računalo moglo razumjeti fotografiju. Međutim, važno je napomenuti da samo pretvaranje fotografije u binarnu vrijednost nije dovoljno kako bi računalo razumjelo što navedena fotografija sadrži. Uz pomoć strojnog učenja ovaj proces razumijevanja sadržaja postaje moguć. Poput ljudi, računala je moguće naučiti određene vještine te prilagoditi ih za razne aktivnosti u stvarnom svijetu.

Proces učenja moguć je uz pomoć modela kojega se može razumjeti kao vještina za odrađivanje neke aktivnosti. Model će kroz proces učenja moći prepoznati karakteristike objekta te u slučaju budućeg korištenja, na temelju snimljenih karakteristika, prepoznati novu fotografiju objekta (Pokhrel, 2019). Način na koji računalo prepoznaje fotografiju zastave dodatno je razrađen u poglavlju 3.3. Dakle, na temelju nula i jedinica unutar same fotografije te treniranja obavljenim nad modelom, model će raditi statističke pretpostavke te tako klasificirati fotografiju.

Slika 13. Lijevo: slika zastave Estonije (Vemu, n.d.), desno: vrijednosti vezane uz piksel na fotografiji

3.1. Odabir razvojnog okvira za izradu modela

Strojno učenje temelji se na algoritmima čije razumijevanje zahtijeva znanje matematike i statistike, međutim, kako bi se strojno učenje prilagodilo u drugim poljima poput medicine potrebno je pojednostaviti primjenu algoritama kako bi se što više ljudi moglo uključiti u razvijanje vlastite umjetne inteligencije. Iz tog razloga izrađeni su brojni razvojni okviri koji pojednostavljuju algoritme te omogućuju uporabu strojnog učenja u drugim znanstvenim poljima. Postoje razni razvojni okviri strojnoga učenja čija korisnost ovisi u domeni, no većina ih zbog jednostavne sintakse, velike snage i pristupačnosti za razvijanje modela koristi programski jezik Python (Karczewski, 2020).

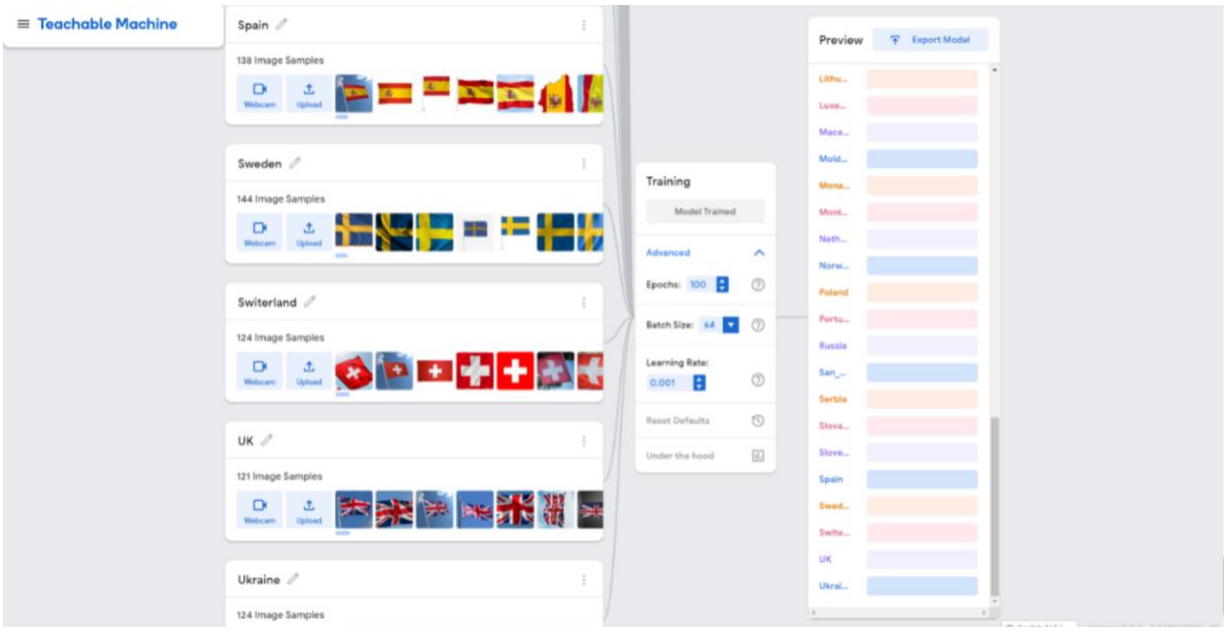
Uzimajući u obzir krajnji cilj ovoga rada, tj. implementaciju modela za mobilne uređaje, odabran je razvojni okvir *TensorFlow*. TensorFlow je javno dostupan projekt razvijen od strane Googlea koji se koristi kao alat za istraživanje i proizvodnju u strojnome učenju (Yegulalp, 2019). Također, zbog velike količine dostupne literature te resursa jedan je od najpristupačnijih razvojnih okvira za strojno učenje. Pojam razvojnog okvira može se promatrati kao „kutija za alat“ (engl. toolbox) unutar koje se nalaze različite funkcije i klase koje znatno olakšavaju uporabu te znatno skraćuju razvojno vrijeme aplikacije (Singh, 2020). Umjesto izrade nove platforme potrebno je detaljno istražiti i proučiti dostupne alate jer su neki alati poput TensorFlowa već optimizirani te spremni za primjenu. Temelj izgradnje modela u verziji

TensorFlow 2.0 čini podvrsta strojnog učenja koju se naziva duboko ili dubinsko učenje (engl. deep learning), a čiji rad omogućuje API pod nazivom *Keras*.

Sam pojam duboko ili dubinsko učenje može se razumjeti kao evolucija strojnoga učenja. Vrlo važna razlika jest da duboko učenje koristi sustav umjetnih neuronskih mreža (engl. artificial neural network, ANN), sličnim onima u ljudskome mozgu, kako bi omogućile računalima donošenje vlastite odluke bez pomoći ljudi. U strojnome učenju – iako modeli progresivno postaju sve bolji i bolji – oni i dalje trebaju ljudsku pomoć kako bi prepravili neispravne pretpostavke modela. Cilj dubokog učenja jest razvoj algoritama koji mogu dokučiti točnost pretpostavke kroz svoju neuronsku mrežu. Sama inspiracija umjetnih neuronskih mreža proizlazi iz biološke neuronske mreže ljudskoga mozga te upravo ovo omogućuje novi proces učenja čije su mogućnosti daleko šire od standardnih modela strojnoga učenja (Grossfeld, 2020). Više o načinu rada neuronskih mreža bit će u sljedećem poglavlju ovoga rada.

Uz razvojna okruženja vrlo je važno spomenuti i web servis *Teachable Machine*[2]. Izrada vlastitih modela za strojno učenje omogućena je putem ovog servisa bez znanja programiranja i strojnoga učenja. Sve što je potrebno za izradu ovakvog modela su podaci. Primjerice na slici 14 (Teachable Machine, n.d.) prikazan je unos fotografija unutar sučelja Teachable Machine-a. Vrsta podataka kod strojnoga učenja može biti raznolika. Uz unos podataka moguće i promijeniti parametre vezane uz strojno učenje. Naravno kao i većina odmah spremnih usluga postoje ograničenja te će rezultati prilagođenih modela postajati sve bolji, međutim, ovakva pristupačnost uistinu svakome pruža priliku da se upozna sa svijetom strojnog učenja.

[2] Teachable Machine (https://teachablemachine.withgoogle.com/)

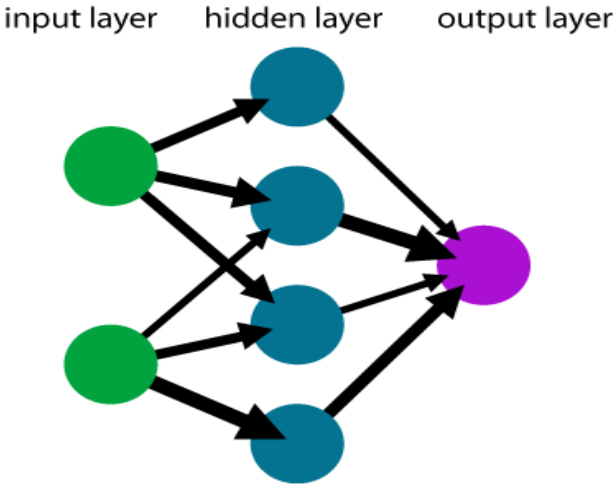


Slika 14. Postupak unosa podataka unutar web servisa Teachable Machine

3.2. Princip rada umjetnih neuronskih mreža

S obzirom na to da će se u ovome radu model razvijati uz pomoću umjetne neuronske mreže, potrebno je ovdje prvo razjasniti terminologiju. Poput bioloških neuronskih mreža princip funkcioniranja su neuroni. U umjetnoj neuronskoj mreži mogu se pronaći tri sloja neurona. Prvi sloj je ulazni sloj koji uzima vrijednosti poput piksela neke fotografije. Vrijednost tih piksela povezuje se sa središnjim dijelom koji se naziva skriveni sloj. Skriveni sloj nije obvezan element te je moguće imati različitu količinu skrivenih slojeva. Upravo zbog veće količine skrivenih slojeva koristi se naziv duboka neuronska mreža (engl. deep neural network) te se stoga sam proces i naziva duboko učenje (Hsu, 2020). Operacije unutar skrivenoga sloja ovise o vrste neuronskih mreža, primjerice, u ovome radu koristit će se funkcije za izradu konvolucijske neuronske mreže čija je svrha procesiranje fotografija. Kroz skrivene slojeve model će se fokusirati na različite aspekte fotografije te im dodavati različite vrijednosti. Optimalna količina skrivenih slojeva nije poznata, međutim, kroz eksperimentiranje treba otkriti kako je moguće što bolje optimizirati model ne bi li preciznije klasificirao fotografije. Posljednji sloj je izlazni te on na temelju različitih rezultata dobivenih u skrivenome sloju daje rezultat (Hsu, 2020). Rezultat će

u ovome modelu biti jedna od zapisanih zastava. Vizualni prikaz jednostavne umjetne neuronske mreže prikazan je na slici 15.



Slika 15. Sustav neuronskih mreža (Wikipedia, n.d.a)

3.3. Razvoj modela za klasifikaciju zastava

Prije same izrade modela potrebno je instalirati i importirati paket TensorFlow u Python. Proces instalacije razvojnog okvira TensorFlow identičan je procesu instalacije ostalih paketa u Pythonu. Pri instalaciji TensorFlowa automatski se instalira i Keras API koji omogućuje izradu, treniranje i ispitivanje modela (Brownlee, 2019b). Prvi korak izrade modela jest imenovanje varijable unutar koje će se nalaziti funkcije za izradu modela. Implementacija Keras API funkcija omogućena je na sljedeći način: prvo je potrebno uvesti paket TensorFlow za koji se koristi alternativni naziv (engl. alias), tj. skraćenica *tf* , a zatim se elementima Keras API-a pristupa tako da se uz skraćenicu *tf* pridruži *keras* te objekt kojemu se želi pristupiti (Brownlee, 2019b). Zatim je potrebno odrediti vrstu modela. Primjerice, implementacija modela na slici 16 predstavlja sekvencijski model (engl. sequential model). Ovaj model je ujedno i najjednostavnija vrsta modela za izradu neuronskih mreža te se sekvencijskih model upravo i koristi u ovome radu. Način na koji ovaj model radi moguće je dokučiti iz samoga naziva, tj. operacije se odvijaju na temelju sekvencija. U takvom pristupu slojevi se nižu jedan za drugim ili liniju po liniju (Brownlee, 2019b).

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(224, 224, 3)),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(2, activation='softmax')
])
```

Slika 16. Primjer sekvencijskog modela

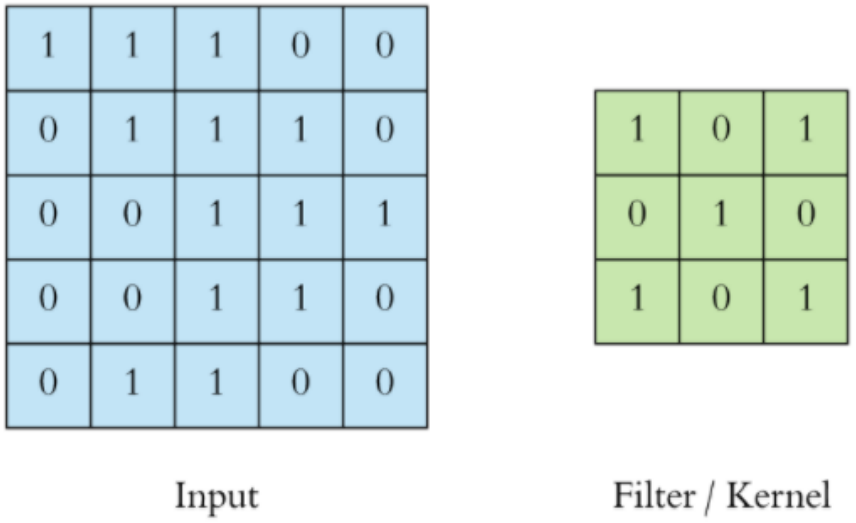
U prijašnjem poglavlju spomenuta je konvolucijska neuronska mreža kao mreža koja procesira fotografije te će se upravo ona i koristiti za izradu modela u ovom radu. Konvolucijska neuronska mreža (engl. convolutional neural network, CNN), vrsta je duboke neuronske mreže osmišljena za obradu strukturiranih vrsta podataka poput slika (Wood, 2020). Osim izrazite korisnosti u procesiranju fotografija, koristi se primjerice i za potrebe prepoznavanja glasa. Naziv dolazi od matematičke operacije „konvolucija“ (engl. convolution) koja se primjenjuje nad matricom (Albawi et al., 2017) i predstavlja matematičku kombinaciju dviju funkcija radi generiranja treće funkcije te time spaja dva skupa informacija

Arhitektura CNN se razlikuje u odnosu na klasičnu neuronsku mrežu. Prvi sloj naziva se konvolucijski sloj te on sadrži parametre poput ulaska podataka, filtera i slično. Također, prisutni su slojevi udruživanja (engl. pooling layers) te slojevi koji su prisutni i kod klasične neuronske mreže, tj. skriveni i izlazni sloj (Albawi et al., 2017). Na kraju neuronske mreže rezultat će biti sveden na jedan vektor vjerojatnost koji na temelju prisutnih klasa daje rezultat (Cornelisse, 2018). Kako bi se fotografija unijela unutar modela potrebno je odabrati ulazne vrijednosti poput visine, širine te vrste boje. S obzirom na to da su zastave prikazane u boji odabrana je vrijednost 3 koja se odnosi na 3 kanala boje. Druge dvije vrijednosti predstavljaju širinu i visinu fotografije koja se unosi u model. Prikaz ove vrijednosti unutar programskog koda prisutan je na slici 17.

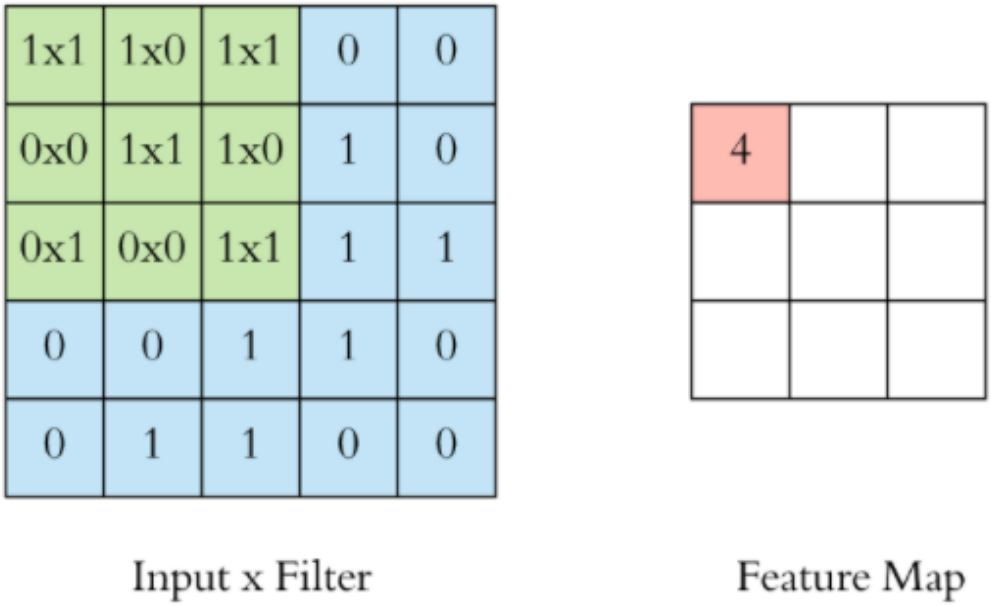
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(224, 224, 3)),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dropout(0.25),
```

Slika 17. Ulazne vrijednosti fotografije

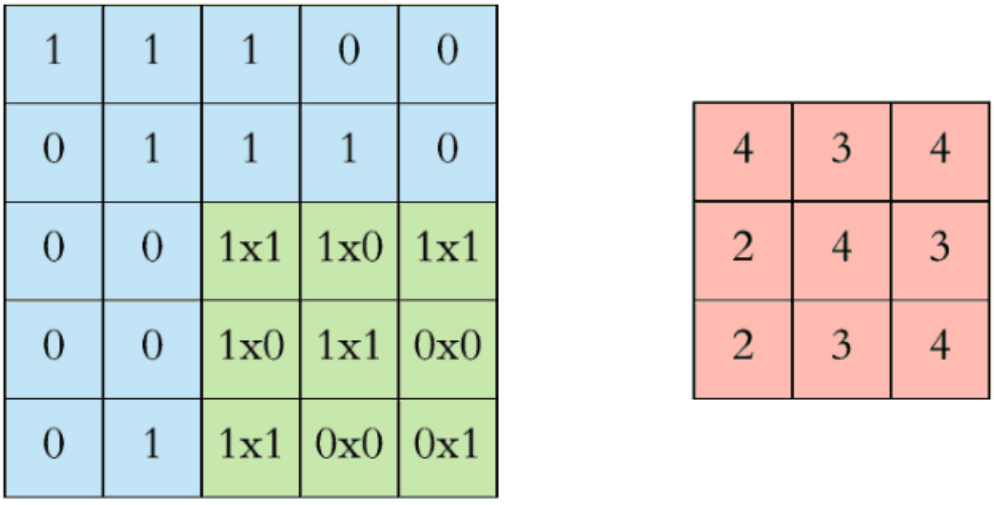
Kako bi model ispravno radio potrebno je prepoznati karakteristike svake zastave. Neke od pojedinosti su grb, boje, vertikalne ili okomite crte, rubovi i slično. Prepoznavanje karakteristika se vrši uz pomoću operacija konvolucija i udruživanja. U CNN, proces konvolucija odnosi se na primjenu, tj. prijelaz filtera (poznatim i pod nazivom „kernel“) preko ulaznih vrijednosti čime se dobiva vrijednost s nazivom mapa značajki (engl. feature map). Mapa značajki je zapravo rezultat množenja matrica ulaznih vrijednosti i filtera (Cornelisse, 2018). Vizualni prikaz nastanka mapa značajka prisutan je na slikama 18, 19 i 20 (Derat, 2017).



Slika 18. Lijevo: unesene vrijednost, desno: filter

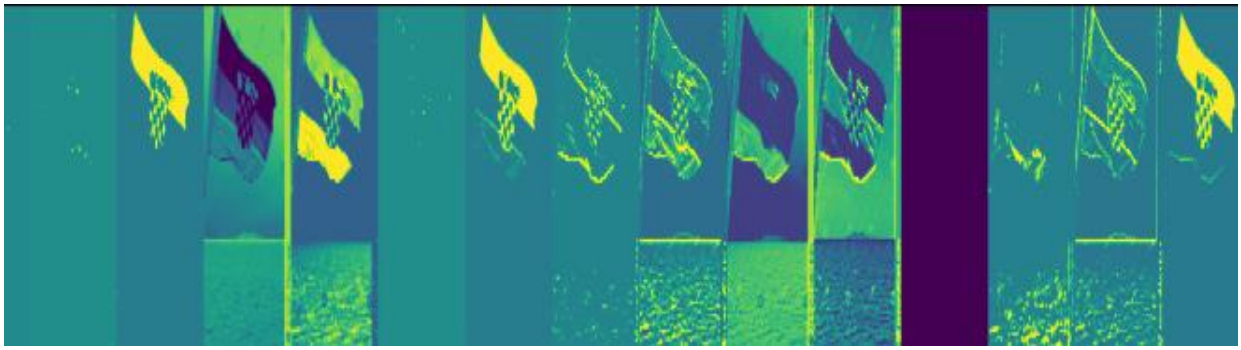


Slika 19. Lijevo: prelazak filtera preko unesenih vrijednost, desno: rezultat množenja sprema se u mapu značajki



Slika 20. Konačan rezultat mapa značajki

Nad ulaznom vrijednosti primjenjuje se velik broj konvolucija te se izrađuje veliki broj mapa značajki. Nakon završne konvolucije sve se mape značajki spajaju što daje završni prikaz jednoga sloja. Prikaz izrade mapa značajki za hrvatsku zastavu dan je na slici 21.



Slika 21. Prikaz izrade mapa značajki za hrvatsku zastavu

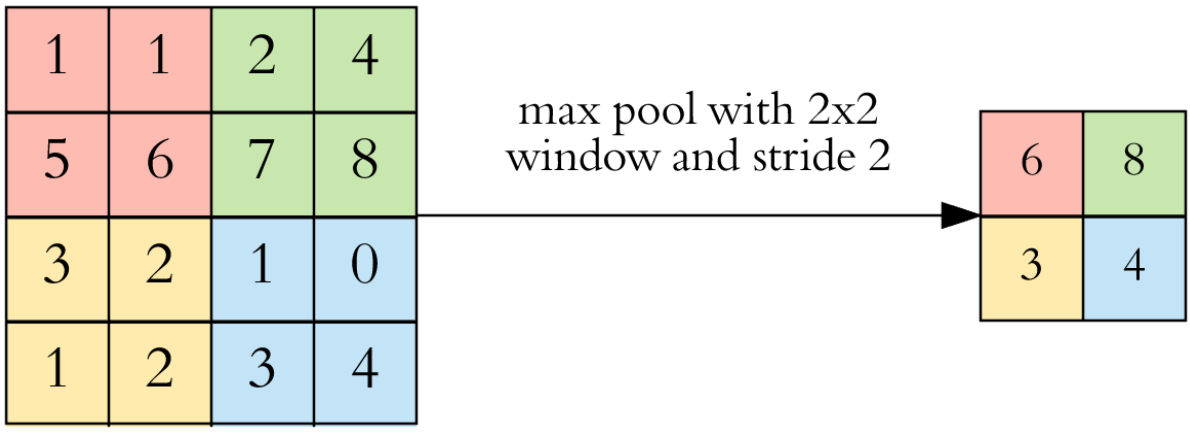
Tipični pomak filtera je jedan piksel ili jedan korak (engl. stride). Korak je veličina pomaka kojim se filter kreće nad vrijednostima. S povećanjem broja koraka, filter se pomiče s većim intervalom preko ulaznih vrijednosti (Cornelisse, 2018). Optimalan broj filtera i pomaka ovisi o vrsti podataka i složenosti detalja unutar fotografije. Kroz eksperimentiranje i proučavanje zbirke podataka najbolji rezultat modela dobiven je uz pomoću 32 filtera i pomaka od (3, 3). Ovo je prikazano unutar programskog koda na slici 22.

```
22  model = tf.keras.models.Sequential([
23      tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(224, 224, 3)),
24      tf.keras.layers.MaxPooling2D(),
```

Slika 22. Veličina koraka unutar modela za klasifikaciju zastava

Nakon svakog konvolucijskog sloja često se dodaje već spomenuti sloj udruživanja. Svrha ove operacije je smanjenje dimenzionalnosti kako bi se smanjila količina parametara te tako smanjila potrebna računalna snaga za treniranje neuronske mreže. Ovaj proces smanjuje potrebno vrijeme treniranje što je ujedno i jedan od načina za kontrolu prenaučenosti, tj. pretreniranosti modela (engl. overfitting). Više o pojmu prenaučenosti bit će dano u sljedećem poglavlju. U ovome radu koristi se operacija *Max_pooling* koja uzima najveću vrijednost svake

matrice te tako smanjuje veličinu mape značajki, ali zadržava sve potrebne informacije (Cornelisse, 2018). Primjer rada ove operacije prikazan je na slici 23 (Cornelisse, 2018).



Slika 23. Smanjenje dimenzionalnosti uz operaciju udruživanja

Sve neuronske mreže također imaju još jednu važnu funkciju pod imenom aktivacija (engl. activation function). Odabir aktivacijske funkcije (ili funkcije aktiviranja/aktivacije) u skrivenom sloju kontrolirat će koliko je model sposoban naučiti podatke, dok odabir aktivacijske funkcije u izlaznom sloju određuje kakve vrste pretpostavki (hipoteza) model može uspostavljati. Odabir aktivacijske funkcije unutar skrivenog sloja ovisi o arhitekturi mreže te se za moderne mreže s arhitekturama poput konvolucijske često odabire aktivacija pod nazivom *ReLU - Rectified Linear Unit* (Brownlee, 2021). U neuronskoj mreži aktivacijska funkcija odgovorna je za transformiranje zbrojenog ponderiranog ulaza od čvora do aktivacije čvora ili izlaza za taj ulaz. ReLU je linearna funkcija koja će izravno vratiti ulaz ako je on pozitivan, u suprotnom, vraća se nula. Ta funkcija postala je standardna, tj. uobičajena aktivacijska funkcija za mnoge vrste neuronskih mreža jer je model koji koristi ReLU lakše trenirati i često postiže bolje performanse (Brownlee, 2020). ReLU funkcija jednostavna je za implementaciju i vrlo je učinkovita u prevladavanju ograničenja drugih aktivacijskih funkcija poput *Sigmoida*-a i *Tanh-a* (Brownlee, 2021). Posebno je manje osjetljiva na problem nestajućeg gradijenta (engl. vanishing gradient) koji onemogućuje treniranje pouzdanog modela. No, s druge strane ReLU može proizvesti zasićene (engl. saturated) ili „mrtve“ (engl. dead) jedinice. Kroz eksperimentiranje nad podacima, funkcija ReLU je dala najbolje rezultate u ovome modelu te je stoga ova funkcija i

prisutna u konačnoj inačici. Odabir aktivacijske funkcije unutar programskog koda prikazan je na slici 24.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(224, 224, 3)),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Conv2D(32, (3,3), activation="relu"),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.MaxPooling2D(),
```

Slika 24. Aktivacijska funkcija unutar konvolucijske mreže

Nakon konvolucijskih slojeva i slojeva udruživanja, klasifikacijski dio sastoji se od nekoliko potpuno spojenih slojeva. Ovi slojevi jedino mogu prihvaćati jednodimenzionalnu vrstu podataka. Stoga je potrebno pretvoriti trodimenzionalnu vrstu podataka u jednodimenzionalnu te je to moguće postići uz pomoć funkcije *izravnavanja/ravnanja* (engl. flatten) (Cornelisse, 2018). Implementacija ove funkcije vidljiva je na slici 25.

```
38      tf.keras.layers.MaxPooling2D(),
39      tf.keras.layers.Dropout(0.4),
40      tf.keras.layers.Flatten(),
41      tf.keras.layers.Dense(512, activation="relu"),
42      tf.keras.layers.Dropout(0.35),
43      tf.keras.layers.Dense(48, activation='softmax')
```

Slika 25. Funkcija ravnanja unutar modela za klasifikaciju zastava

Posljednji slojevi unutar konvolucijske mreže čine gusti slojevi (engl. dense layers). Ovo su potpuno povezani dijelovi neuronske mreže koji se nadovezuju na prethodne neurone (Cornelisse, 2018). U posljednjem sloju kao broj neurona potrebno je navesti broj kategorija unutar podataka. Zbog 48 vrsta zastava odabran je broj 48 na temelju kojega će biti odabran rezultat klasifikacije. S obzirom na to da se u ovome radu izvršava klasifikacija nad više vrsta zastava potrebno je odabrati aktivacijsku funkciju *Softmax*. Ova funkcija pretvara vektore u

vrijednost koju je moguće interpretirati kao vjerojatnost (Brownlee, 2021). Da bi se u konačnici prikazala arhitektura modela te broj parametara koristi se metoda *sažetak* (engl. summary). Prikaz arhitekture modela prisutan je na slici 26.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 222, 222, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 111, 111, 32)      0
_________________________________________________________________
dropout (Dropout)            (None, 111, 111, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 109, 109, 32)      9248
_________________________________________________________________
dropout_1 (Dropout)          (None, 109, 109, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 54, 54, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 52, 52, 32)        9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 26, 26, 32)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 26, 26, 32)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 24, 24, 32)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 10, 32)        9248
_________________________________________________________________
dropout_4 (Dropout)          (None, 10, 10, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 32)          9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 32)          0
_________________________________________________________________
dropout_5 (Dropout)          (None, 4, 4, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 512)               262656
=================================================================
Total params: 334,416
Trainable params: 334,416
Non-trainable params: 0
```

Slika 26. Sažetak te broj parametara unutar modela za klasifikaciju zastava

3.4. Povezivanje i daljnja obrada podataka

Nakon izrade modela i agregacije podataka potrebno je povezati model s mapom koja sadrži zbirku podataka. Povezivanje podataka s modelom je vrlo jednostavno; potrebno je pohraniti relativnu ili apsolutnu adresu mape sa zbirke podataka kao varijablu. Povezivanje zbirke podataka i modela prikazano je na slici 27.

```
52
53   training_dir="data/flags/training"
54
```

Slika 27. Pohrana relativne adrese podataka

Nakon izrade varijable s relativnom adresom uvedena je klasa iz paketa Keras po imenu *ImageDataGenerato*r. Ova klasa omogućava povećanje zbirke podataka, tako da se fotografije mogu dodatno mijenjati na razne načine. Neke od mogućih tehnika su: normalizacija, rotacija, zrcaljenje, promjena svjetline i slično. Navedene tehnike koriste se kako bi se dodatno proširio skup podataka te se tako poboljšala točnost modela u stvarnom svijetu. Na ovaj će način model dobiti različite varijacije podatka bez da ih se nadodaje na originalnu zbirku podataka (Bhandari, 2020). Demonstracija korisnosti ove klase vizualizirana je na slici 28.

Slika 28. Lijevo: originalna fotografija (Dennis, 2019), desno: nakon prolaska kroz ImageDataGenerator

Jedan od razloga primjene klase poput ImageDataGenerator je potencijalno izbjegavanja prenaučenosti. Do prenaučenosti dolazi kada model nauči detalje, ali i šum koji se nalazi u podacima za treniranje te se previše na njih oslanja tijekom validacije, tj. ispitivanja (Brownlee, 2019a). Zbog navedenoga, model će imati problema s klasifikacijom budućih podataka te je stoga potrebno, pogotovo u slučaju jako male zbirke podataka, dodatno modificirati zbirku. Dodatan način za rješavanje ovoga problema prenaučenosti je dodavanje slojeva ispuštanja (engl. dropout layers) kod razvoja konvolucijskih slojeva i slojeva neuronskih mreža. Slojevi ispuštanja omogućuju odbacivanje određenog broja izlaza unutar odabranog sloja. Izlazi su odabrani nasumično s ciljem stvaranja šuma unutar procesa treniranja. Zbog ispuštanja, tj. oblika šuma, slojevi unutar modela moraju se prilagoditi i popraviti pogreške prijašnjih slojeva te na taj način model postaje više robustan (Brownlee, 2018a), što je ujedno i cilj stvaranja šuma. Korištenje ovakvih slojeva unutar programskog koda prikazano je na slici 29.

```
30      tf.keras.layers.Conv2D(32, (3,3), activation="relu"),
31      tf.keras.layers.Dropout(0.25),
32      tf.keras.layers.MaxPooling2D(),
33      tf.keras.layers.Conv2D(32, (3,3), activation="relu"),
34      tf.keras.layers.MaxPooling2D(),
35      tf.keras.layers.Dropout(0.25),
36      tf.keras.layers.Conv2D(32, (3,3), activation="relu"),
37      tf.keras.layers.Dropout(0.25),
```

Slika 29. Primjena slojeva ispuštanja

Uz prenaučenost također je moguća pojava podnaučenosti, poznate i pod nazivima nedovoljna naučenost, podtreniranje ili podbačaj (engl. underfitting). Podnaučenost se pojavljuje kada model ne može prepoznati i točno klasificirati podatke tijekom treniranja. Zbog toga postotak točnosti opada ili je vrlo nizak. Do ovoga dolazi zbog uporabe neprimjerenih aktivacijskih funkcija na model ili slično. Rješenje je uporaba drugih algoritama za strojno učenje (Brownlee, 2019a).

Idealan model za strojno učenje nalazi se između prenaučenosti i podnaučenosti te se postiže kroz temeljito eksperimentiranje i poznavanje vrste podataka. Jedan on načina prepoznavanja je

li model približno jednak optimalnom je upotreba validacijskog skupa podataka (Brownlee, 2019a). U ovome radu validacijski podatkovni skup (engl. validation data set) izrađen je na temelju podjele izvornog skupa podataka u omjeru 80:20. Pomoću 80% podataka izvršeno je treniranje modela, a pomoću preostalih 20% podataka izvršena je validacija, tj. ispitivanje učinkovitosti modela.

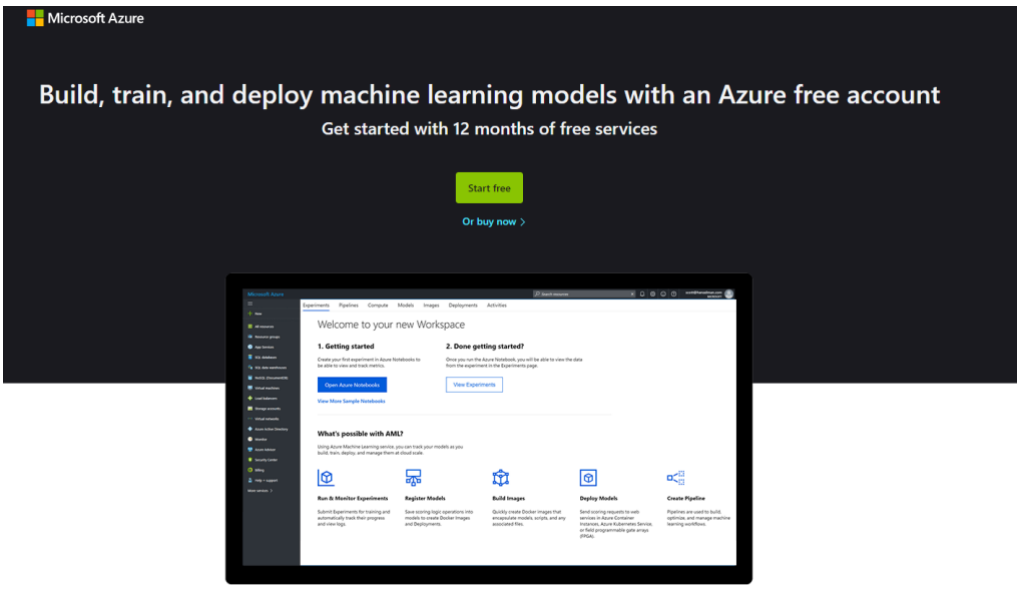
```
55  datagen = ImageDataGenerator(validation_split=0.20, rescale=1./255,
56  rotation_range=30, brightness_range=[0.2,1.8],
57  width_shift_range=0.2, height_shift_range=0.2)
```

Slika 30. Vrijednosti unutar klase ImageDataGenerator

Kako bi varijabla *datagen* bila povezana s mapama koje sadrže fotografije, koristi se metoda *flow_from_directory()*, što je prikazano i na slici 31. Prva varijabla odnosi se na direktorij ili lokaciju mapa s fotografijama. Relativna adresa prethodno je spremljena u varijablu *training_dir* te se stoga ona ovdje upisuje kao prvi argument. S obzirom na podjelu podataka na one za treniranje i validaciju, potrebno je navedeno istaknuti varijablom *subset*. Kako bi se fotografije mogle koristiti putem modela potrebno je promijeniti veličinu, što je i učinjeno varijablom *target_size*. S obzirom da će prikupljene fotografije biti različitih veličina potrebno je odabrati jednu (temeljnu) vrijednost za sve odabrane fotografije. Još jedna vrlo važna varijabla je *class_mode*. Uzimajući u obzir da skup podataka sadrži 48 zastava potrebno je navesti da je vrsta podataka kategorička. Posljednja važna stavka je varijabla po imenu *batch_size*. U ovome modelu batch_size može se razumjeti kao broj fotografija koje model treba proučiti prije ažuriranja svojih vrijednosti. Uobičajeno za svaku neuronsku mrežu postoji optimalan batch_size te će različite zbirke podataka i pripadajući modeli imati različit batch_size. Količina podataka koju računalo može prihvatiti u jednome koraku ovisi o memorijskom kapacitetu računala (Haleva, 2020). Nakon temeljitih testiranja ove zbirke podataka te vlastitih računalnih performansi odabran je batch_size u iznosu od 64, međutim, postoje online servisi koji omogućavaju treniranje modela putem cloud usluge. Jedan od takvih servisa je Microsoft Azure; navedeni servis prikazan je na slici 32 (Microsoft, n.d.).

```
train_generator = datagen.flow_from_directory(
    training_dir,
    subset='training',
    target_size=(224,224),
    class_mode="categorical",
    shuffle=True,
    batch_size=64
)
```

Slika 31. Povezivanje podataka s modelom



Slika 32. Primjer servisa za treniranje modela

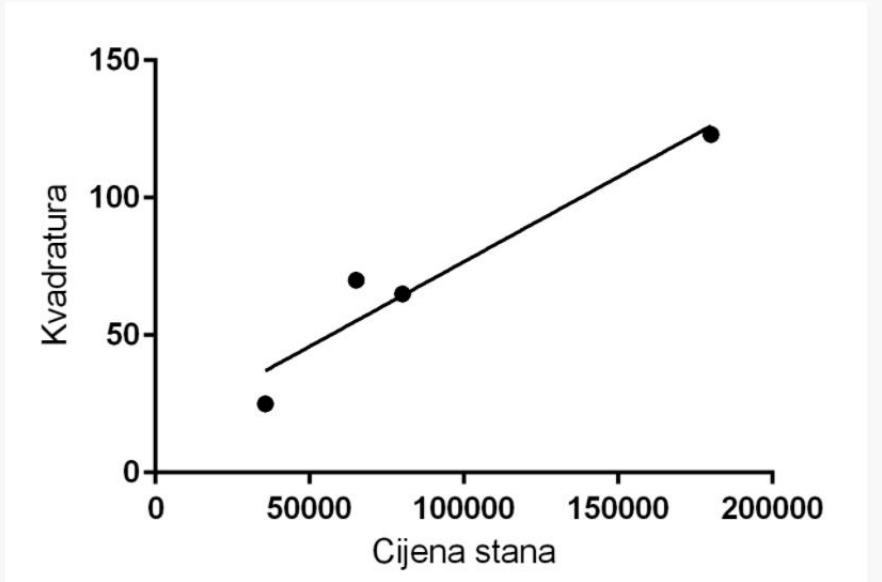
3.5. Pokretanje modela

Nakon što je model izrađen moguće ga je podešavati uz pomoću metode *compile()*. Prikaz programskog koda vezan uz metodu compile() prikazan je na slici 33. Unutar ove metode nalazi se funkcija gubitka (engl. loss function) i optimizator (engl. optimizer), a moguće je dodati i dodatnu metriku kako bi se pokazala primjerice točnost modela tokom treniranja (Abadi et al., 2015a). Općenito, izrada vlastitih funkcija gubitka i optimizatora nije potrebna već je velik broj dostupan kroz Keras API.

```
80    model.compile(optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"])
```

Slika 33. Metoda compile() unutar modela za klasifikaciju zastava

Iako sama arhitektura neuronskih mreža igra veliku uloga pri ekstrakciji informacija iz podataka, većina ih se optimizira kroz konstantno ažurirajuća pravila koja se temelje na gradijentu funkcije gubitka (Descamps, 2017). Funkcija gubitka koristi se za izračun pogreške između vrijednosti danih od strane modela te vrijednosti prisutnih u podacima. Funkciju gubitka moguće je ilustrirati na primjeru linearne regresije (Mahendru, 2019). U tom primjeru prikazan je problem linearne regresije uz pomoću podataka vezanih uz cijenu stana te kvadrature. Ovdje se cijena stana povećava paralelno u odnosu na kvadraturu te je na temelju ove pretpostavke napravljen grafikon 1.



Grafikon 1. Odnos cijene stana i kvadrature

Crne točke na grafikonu predstavljaju cijenu stana s obzirom na kvadraturu, a dijagonalni pravac prikazuje predviđanja koje su pomoću strojnog učenja ostvarene na temelju ulaznih podataka. Kao što je vidljivo, sve vrijednosti se ne nalaze točno na liniji regresije već se poneke

nalaze van pravca. Funkcija gubitka se upravo odnosi na razliku između originalne vrijednosti te vrijednosti koje je model odredio. Cilj optimizatora je pronaći poziciju pravca u kojemu je funkcija gubitka što je moguće manja. U ovome primjeru linearne regresije koristi se funkcija gubitka pod nazivom *mean squared error* ili *MSE*. Formula za izračun MSE dana je na slici 34 (Wikipedia, n.d.b).

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

Slika 34. Formula za računanje MSE

S obzirom na problem identifikacije zastava potrebno je pronaći funkciju gubitka koja se koristi za identifikaciju više različitih klasa. Na temelju preduvjeta odabrana je funkcija gubitka pod nazivom *categorical_crossentropy()*. Ova funkcija gubitka omogućuje modelu pridruživanje vrijednosti za vjerojatnost svakoj novoj fotografiji pri klasifikaciji. Primjerice, ako je pridružena fotografija najsličnija zastavi Njemačke ona će poprimiti vrijednost blizu 1 za ovu klasu (zastavu), a za sve druge neku vrijednost bliže nuli. Ova se funkcija gubitka često uparuje s aktivacijskom funkcijom Softmax te se navedena kombinacija naziva *Softmax loss.* (Gómez, 2018).

Kao što je već ranije spomenuto, svrha optimizatora je što je moguće više smanjiti funkciju gubitka. Optimizatori izvršavaju navedeno tako da ažuriraju model nakon rezultata primjene funkcije gubitka te kroz narednu iteraciju pokušaju novu vrijednost približiti originalnim podacima. U ovome radu korišten je optimizator imena *Adam* ili „Adaptive Learning Rate Optimization“ algoritam. Ovaj algoritam primjenjuje metodu prilagodljive stope učenja/prilagođujućih stopa učenja (engl. adaptive learning rate method) kako bi se pronašla pojedina, tj. indvidualna vrijednost stope učenja za svaki parametar modela. Parametri modela se odnose na svojstva koja su naučena iz zbirke podataka tijekom treiniranja, primjerice određena svojstva hrvatskog grba poput šahovnice. Implementacija ovoga algoritma je vrlo jednostavna te zahtijeva malenu količinu memorije. Vrlo važna osobnost ovoga algoritma je sposobnost rješavanja problema velikih količina podataka i parametara te je zbog toga ovaj algoritam vrlo pogodan i za daljnje razvijanje ove aplikacije (Kingma & Ba, 2017). Posljednji parametar pod

imenom *metrics* u metodi compile() omogućuje izračun točnosti podatkovnog skupa za treniranje i validaciju.

Kako bi se postupak treniranja izvodio, koristi se metoda *fit()*. Prikaz programskog koda vezan uz metodu fit() nalazi se na slici 35. Unutar ove metode upisuju se varijable koje sadrže podatke za treniranje i validaciju te se odabire broj na temelju kojeg će se izvoditi operacije nad modelom. Ova brojka naziva se epoha (engl. epoch). Nakon završetka jedne epohe model je imao priliku proučiti svaki podatak unutar zbirke jedanput te je isti podatak utjecao na vrijednosti unutar modela. Broj epoha je tradicionalno u stotinama ili tisućama jer ovako velika količina epoha omogućuje veće smanjenje funkcije gubitka (Brownlee, 2018b).

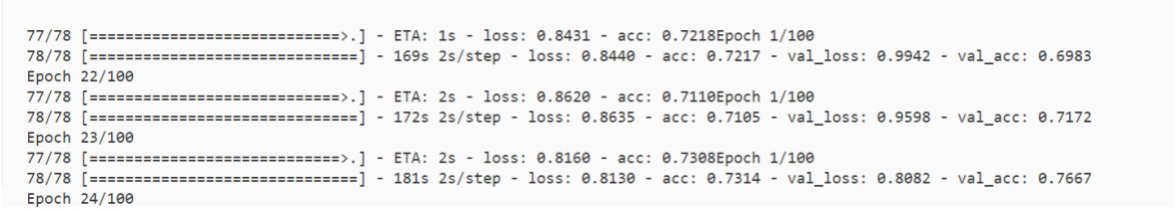
```
80  model.compile(optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"])
81  model.fit(train_generator, epochs=100, validation_data = validation_generator,
82   verbose = 1, callbacks=[callbacks],)
```

Slika 35. Metoda za treniranje modela

Unutar metode fit(), dodan je još jedan parametar pod nazivom *callbacks*. Taj parametar predstavlja objekt klase *myCallback* i on se uvodi s ciljem da se treniranje zaustavi u slučaju da se dosegne određena vrijednost. Prikaz programskog koda nalazi se na slici 36. Ovdje se navedena klasa poziva u slučaju da vrijednost validacijskog skupa prijeđe točnost od 94%. U slučaju da vrijednost validacijskog skupa ne prijeđe vrijednost od 94%, treniranje se zaustavlja nakon 100 epoha. Prikaz procesa treniranja je prisutan na slici 37.

```
13  class myCallback(tf.keras.callbacks.Callback):
14      def on_epoch_end(self, epoch, logs={}):
15          if (logs.get("val_acc")>0.94):
16              self.model.stop_training= True
17  callbacks=myCallback()
18
```

Slika 36. Metoda za zaustavljanje treniranja



Slika 37. Primjer treniranja modela

Nakon procesa treniranja model se sprema u obliku Keras datoteke te se nazivi svih mapa unutar mape za treniranje ispisuju unutar tekstualne datoteke. Spremanje modela omogućuje primjenu istog modela nad drukčijim, ali povezanim problemom. Primjerice, ako je razvijen klasifikator za identifikaciju ruksaka, moguće je iskoristi znanje koje je model stekao tijekom procesa treniranja kako bi se koristio za prepoznavanje drugih objekata poput sunčanih naočala ili slično. Proces ovakve vrste učenja naziva se prijenosno učenje (engl. transfer learning) (Donges, 2019). Uz prijenosno učenje model je moguće iskoristiti za daljnju primjenu na jednakom problemu bez potrebe ponovnog treniranja. Ovo omogućuje promjenu podatka ili varijabli unutar klase ImageDataGenerator kako bi se provodilo daljnje testiranje nad istraživačkom domenom. Implementacija spremljenog modela i tekstualne datoteke obavljena je u sljedećem dijelu ovoga rada.

4. Razvoj mobilne aplikacije za identifikaciju zastava

Sljedeći korak u hodogramu jest razvoj mobilne aplikacije nazvane „FlagAFlag“ kako bi se model mogao implementirati za mobilne uređaje. Razvoj aplikacije izvršen je u razvojnom okruženju *Flutter* te je korišten programski jezik Dart. Problem razvijanja mobilnih aplikacija leži u tome da pri izradi jedne aplikacije treba razviti jednu inačicu za iOS te drugu za Android platformu. Za ovaj pothvat neophodno je znanje više programskih jezika te operativnih sustava. Prednost Fluttera je što daje višeplatformsko rješenje te tako znatno skraćuje razvojno vrijeme i smanjuje potrebno znanje za uspješno razvijanje aplikacije. Flutter je vrlo mlada tehnologija iza koje, poput TensorFlowa, stoji Google (Thomas, 2019). Kroz nadogradnju 2.0 Flutter je postao rješenje i za web i desktop te tako proširio potencijalnu domenu razvoja aplikacija (Sells, 2021). Flutter je razvijen uz pomoću objektu orijentiranog jezika Dart, tj. programskog jezika optimiziranog za izradu korisničkih sučelja (Thomas, 2019).

Samu srž Fluttera predstavljaju widgeti. Widgeti opisuju kako bi točno sam prikaz aplikacije trebao izgledati na temelju danih stanja i naredbi. Widget može sadržavati funkcionalnost teksta, gumba, kamere, galerije i slično. Zbog toga se može reći da je sve unutar Fluttera zapravo widget (Flutter, n.d.). Još jedna značajna prednost Fluttera jest mogućnost tzv. *hot reload-a*. To omogućuje Flutter programerima jednostavno osvježavanje aplikacije te instantno uočavanje promjena bez ponovne reinstalacije aplikacije. Zbog toga je i dizajn same aplikacije vrlo „fluidan“ i pristupačan (Thomas, 2019). Poput Pythona, Flutter podržava primjenu raznih paketa ili pluginova koji znatno skraćuju vrijeme razvoja aplikacije. Primjerice, jedan od pluginova potrebnih za realizaciju ove aplikacije je plugin *camera* koji omogućuje prikaz korisnikove kamere unutar aplikacije.

4.1. Razvoj korisničkog sučelja

Pri izradi korisničkog sučelja glavni cilj je bio izraditi što jednostavnije sučelje za uporabu koje nije „pretrpano“ nepotrebnim dodatcima. S obzirom na to da je cilj omogućiti identifikaciju zastave vrlo je važno omogućiti korisniku fotografiranje željene zastave uz pomoću mobitela. Naravno, jednako toliko je važno i omogućiti korisniku da ima opciju odabira zastave iz vlastite galerije u slučaju da već ima fotografiju spremnu na mobitelu. To su ujedno bili glavni ciljevi za razvoj sučelja mobilne aplikacije.

Prvi korak pri implementaciji je bio uvoz samih pluginova unutar mape aplikacije. Za uvoz je potrebno pronaći datoteku pod nazivom *pubspec.yaml*. Za početak je bilo potrebno uvesti samo pluginove potrebne za rad galerije i kamere. Uvoz plugina prikazan je na slici 38.

```
23  dependencies:
24    flutter:
25      sdk: flutter
26    camera: ^0.7.0+2
27    path_provider:
28    path:
29    image_picker: ^0.6.7+2
30
```

Slika 38. Uvoz plugina unutar yaml datoteke

Nakon uvoza paketa potrebno ih je uvesti i u glavnu aplikaciju, tj. datoteku pod imenom *main.dart* što je prikazano na slici 39. Plugin *image_picker* koristit će se kao način za pristup korisnikovoj galeriji, *material* paket omogućuje pristup raznim widgetima koji se koriste za dizajn te spomenuti plugin camera koji se koristi za prikaz korisnikove kamere.

```
1  import 'package:flutter/material.dart';
2  import 'package:image_picker/image_picker.dart';
3  import 'package:camera/camera.dart';
4
```

Slika 39. Uvoz paketa unutar glavne datoteke

Pokretanje svake Flutter aplikacije započinje izradom funkcijom *main()* koja poziva imenovanu klasu. Klasa u ovome slučaju predstavlja glavni zaslon aplikacije te sadrži funkcije koje su potrebne za rad kamere. Prije poziva same klase potrebno je provjeriti jesu li kamere

dostupne na mobitelu korisnika. Pri pozivu klase uvodi se korištenje tamne teme. Programski kod za navedeno dostupan je na slici 40.

```
Run | Debug
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  final cameras = await availableCameras();
  final firstCamera = cameras.first;


  runApp(
    MaterialApp(
      debugShowCheckedModeBanner: false,
      darkTheme: ThemeData(
      brightness: Brightness.dark),   // ThemeData
      home: App(
        camera: firstCamera,
      ), // App


    ), // MaterialApp
  );
}

class App extends StatefulWidget {
  final CameraDescription camera;
  const App({Key key, @required this.camera,}):super(key:key);
  @override
  _AppState createState() => _AppState();
}
```

Slika 40. Programski kod za pozivanje main() funkcije

Budući da se htjelo izraditi minimalističko korisničko sučelje, odlučeno je da će se na glavnome zaslonu prikazati korisnikova kamera te gumbi čija će funkcionalnost biti fotografiranje, odnosno odabir već ranije pohranjene fotografije unutar galerije mobitela. Glavni zaslon trebao je biti što sličniji prikazu same kamere mobitela te tako korisnik ne bi trebao imati

problema s korištenjem aplikacije. Nakon više iteracija i eksperimentiranja dobiveno je sučelje prikazano na slici 41.

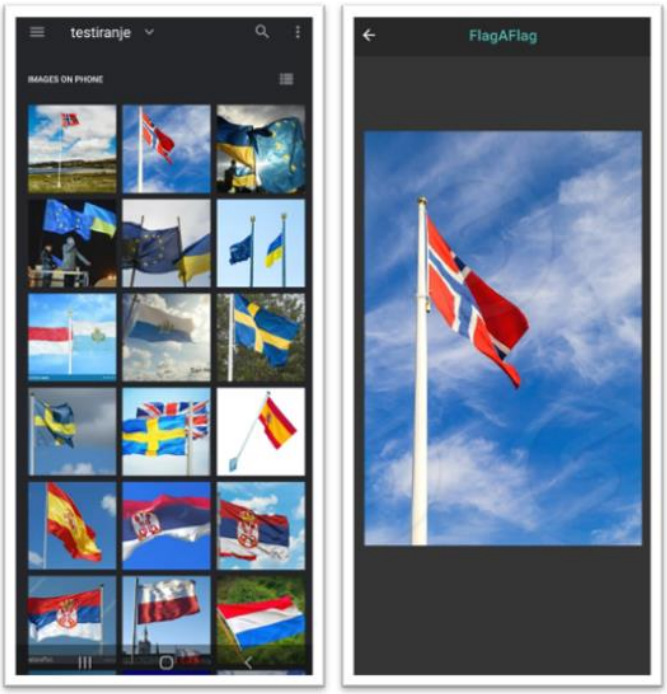
Slika 41. Korisničko sučelje unutar aplikacije

Kada korisnik pronađe položaj zastave pritiskom na gumb za fotografiranje, mobitel prikazuje fotografiju korisniku. Trenutna verzija ne sadrži mogućnost pohranjivanja fotografije, no to je svakako jedan od mogućih dodataka u daljnjoj razradi aplikacije.

Pri odabiru druge opcije korisnik dobiva mogućnost odabira pohranjene fotografije iz galerije. Odabir fotografije moguć je iz svih fotografskih mapa na mobitelu. Drugim riječima, moguće je

odabrati preuzete fotografije, fotografije uslikane putem mobitela i slično. Također, uz pomoću padajućeg izbornika moguće je i odabrati fotografiju iz Google Drive-a. Nakon pronalaska tražene fotografije korisnik odabire istu tako da pritisne na nju. Nakon odabira, fotografija je prikazana korisniku u odgovarajućoj rezoluciji. Odabir fotografije iz galerije te njen prikaz prisutan je na slici 42.

I time je uspješno implementirana funkcija kamere te mogućnost odabira slike. Sljedeći korak u izradi aplikacije je implementiranje modela za strojno učenje te mogućnost klasifikacije slike.



Slika 42. Lijevo: proces odabira fotografije, desno: prikaz odabrane fotografije

4.2. Klasifikacija fotografije na temelju modela

Kako bi se uspješno implementirao model unutar Fluttera potrebno je pretvoriti trenirani model u TensorFlow Lite model. TensorFlow Lite omogućuje strojno učenje na mobilnim uređajima te se tako sprječava potreba za dijeljenjem podataka između korisnika i poslužitelja na kojima bi se strojno učenje izvršavalo. Također, na ovaj način korisnici ne moraju imati izravan pristup internetu kako bi uspješno klasificirali vlastitu fotografiju. TensorFlow Lite također

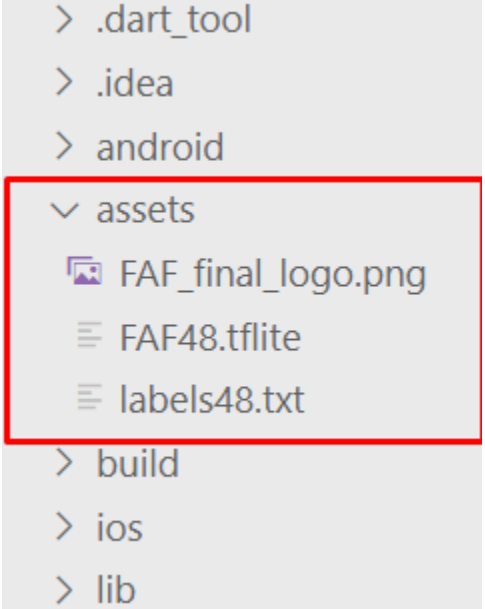
omogućuje rad i na drugim uređajima osim mobitela, primjerice malim kontrolerima ili Raspberry Pi uređajima. Ovo je izrazito korisno jer je moguće razviti vlastite sigurnosne uređaje koji će, primjerice, ako primijete ljudsko lice aktivirati funkciju alarma ili slično (Abadi et al., 2015b)

Kako bi se trenirani model pretvorio u TensorFlow Lite verziju potrebno je pokrenuti blok naredbi u Pythonu, a što je navedeno na slici 43. Pretvorba modela u TensorFlow Lite inačicu znatno smanjuje njihovu veličinu te uvodi dodatnu optimizaciju koja ne utječe na samu točnost modela. Naravno, postoje opcije koje omogućuju dodatno smanjenje modela kako bi model bio što brži, međutim, sama točnost modela onda opada. Nakon uspješne pretvorbe stvorena je nova datoteka koju je moguće implementirati unutar mobilne aplikacije (Abadi et al., 2015b)

```
1  import tensorflow as tf
2  model= tf.keras.models.load_model("Flag48.h5")
3  converter = tf.lite.TFLiteConverter.from_keras_model(model)
4  tflite_model = converter.convert()
5
6  with open("FAF48.tflite", 'wb') as f:
7      f.write(tflite_model)
```

Slika 43. Programski kod za pretvorbu modela u TensorFlow Lite inačicu

Sljedeći korak je premjestiti TensorFlow Lite model i datoteku s imenima zastava u mapu pod imenom *assets* koja se nalazi unutar mape u kojoj je kreirana Flutter aplikacija „FlagAFlag“. Prikaz modela i oznaka mapa dan je na slici 44.



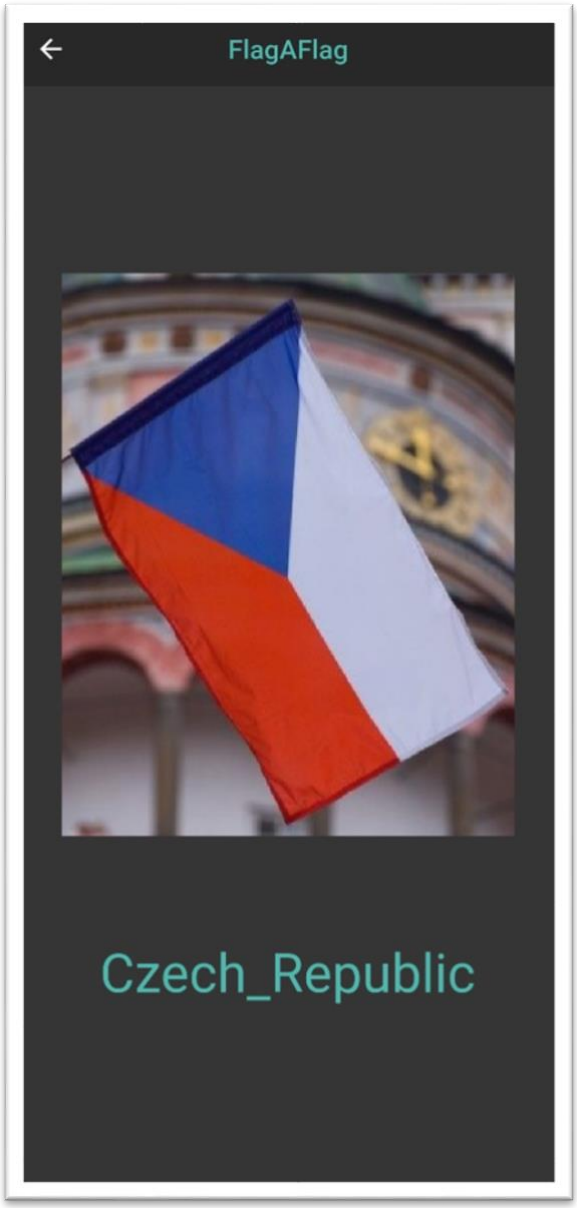
Slika 44. Lokacija te vrijednosti prisutne unutar mape assets

Kako bi sama klasifikacija slike bila omogućena uvodi se plugin *tflite* u već prije spomenutu yaml datoteku. Prikaz ažurirane yaml datoteke nalazi se na slici 45. Paket tflite omogućuje pokretanje TensorFlow Lite modela te klasifikaciju nad odabranom fotografijom. Osim klasifikacije fotografija moguće je provoditi i druge operacije strojnoga učenja, međutim, zbog toga što je u ovome projektu cilj identifikacija zastava, fokus rada ostaje na klasifikaciji fotografija. Poput i ranije navedenih paketa potrebno je tako i ovaj uvesti u glavnu, tj. main.dart datoteku.

```
23  dependencies:
24    flutter:
25      sdk: flutter
26    camera: ^0.7.0+2
27    async: ^2.4.2
28    path_provider:
29    path:
30    image_picker: ^0.6.7+2
31    tflite: ^1.1.1
```

Slika 45. Ažurirana verzija yaml datoteke

Nakon implementacije te dodatne dorade dizajna s ciljem jasnijeg i preglednijeg prikaza, dobiven je rezultat odabira fotografije iz galerije, što je i prikazano na slici 46. Aplikacija je uspješno klasificirala češku zastavu uz pomoć treniranog modela, međutim, što ako je fotografija lošije kvalitete ili je sama slika puna nepotrebnih detalja?



Slika 46. Točna klasifikacija češke zastave

Na temelju novo odabrane fotografije prikazane na slici 47, aplikacija nije uspješno obavila klasifikaciju. Jedan od mogućih razloga jest manjak poklapanja s podacima za treniranje, tj. model nije prethodno vidio sličnu fotografiju. Iako je ljudskim okom vidljivo da se radi o fotografiji mađarske zastave, za točnu klasifikaciju potrebno je fotografiju učiniti što sličnijom onoj u podacima za treniranje. Kako bi se zastava točno klasificirala potrebno je uvesti funkciju odsijecanja jer se tako omogućuje smanjenje nepotrebnih detalja u fotografiji, a to onda omogućuje preciznu klasifikaciju fotografije, tj. zastave. Implementacija odsijecanja je upravo tema sljedećeg dijela ovoga rada.

Slika 47. Netočna klasifikacija mađarske zastave

4.3. Implementacija funkcije odsijecanja unutar aplikacije

Fotografije često dolaze u svakakvim oblicima te je potrebno predvidjeti da fotografije ponekad neće biti savršene te je stoga važno implementirati mogućnost odsijecanja odabrane fotografije.

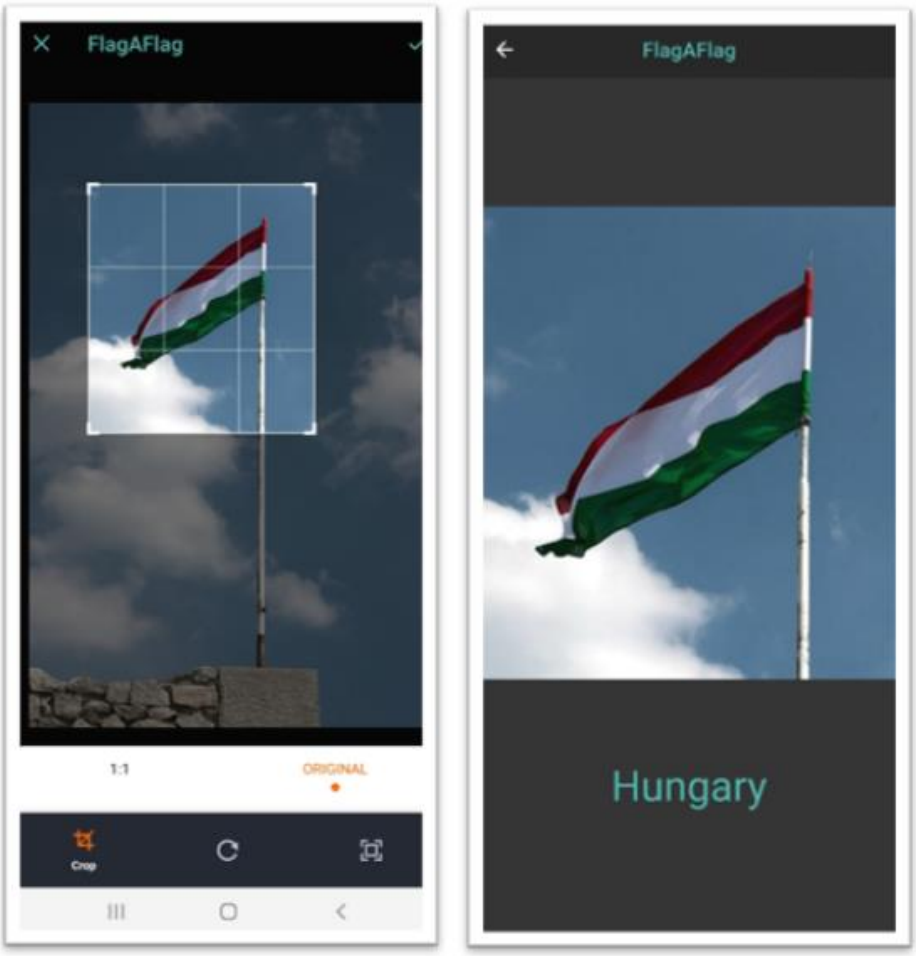
Iako je moguće odsjeći fotografiju unutar galerije, potrebno je isto omogućiti i unutar aplikacije kako bi korisnik mogao nesmetano obaviti identifikaciju zastave. Radi što točnije klasifikacije važno je imati što manje nepotrebnih detalja na odabranoj fotografiji. Ponekad se zastave nalaze jedna pokraj druge pa je stoga važno odsjeći jednu kako bi druga mogla biti u središtu same fotografije.

Implementacija je omogućena uz pomoću plugina *image_cropper*. Kao i kod prijašnjih pluginova i paketa, navedeni plugin je potrebno uvesti u yaml datoteku te je nakon potreban i uvoz u main datoteku. Implementacija funkcije odsijecanja prisutna je na slici 48. Nakon što korisnik završi s fotografiranjem funkcija dohvaća vrijednost samo u slučaju da vrijednost nije jednaka nuli, odnosno, varijabla *imageCam* mora dohvatiti vrijednost kamere mobitela. Vrijednost ove varijable vidljiva je na slici 48. Nad dohvaćenom fotografijom korisnik može vršiti odsijecanje. Jedna od opcija odsijecanja je u obliku kvadrata. Naravno, moguće je implementirati različite vrste odsijecanja, primjerice odsijecanje na temelju omjera slike, kruga i slično. Korisnik također ima mogućnost zaokrenuti fotografiju u slučaju da se fotografija nalazi u neobičnom ili neodgovarajućem položaju. Kako bi dizajn odsijecanja bio što sličniji onome na glavnome zaslonu odabrane su crna i tirkizna boja kako bi bile usklađene s tamnom temom na glavnom zaslonu. Nakon što su odabrane nove dimenzije fotografije, fotografija se klasificira te je rezultat predstavljen korisniku.

```
Future cameraImage() async{
  var imageCam= await _controller.takePicture();
  if (imageCam == null) return null;
  File croppedFile = await ImageCropper.cropImage(
    sourcePath: imageCam.path,
    aspectRatioPresets: [
      CropAspectRatioPreset.square,
      CropAspectRatioPreset.original,
    ],
    androidUiSettings: AndroidUiSettings(
        toolbarTitle: 'FlagAFlag',
        toolbarColor: Colors.black,
        toolbarWidgetColor: Colors.teal.shade300,
        initAspectRatio: CropAspectRatioPreset.original,
        lockAspectRatio: false),
    iosUiSettings: IOSUiSettings(
      minimumAspectRatio: 1.0,
    )
  );
  setState(() {
        _image=croppedFile;
      });
  await classifyImage(croppedFile);
}
```

Slika 48. Programski kod za funkciju odsijecanja

Nakon implementacije odsijecanja vidljiva je razlika u klasifikaciji na slici 49 te je očito da se smanjenjem samih detalja na fotografiji povećala i točnost klasifikacije, zbog toga što je sada samo zastava prisutna na fotografiji. S ovime je završen proces razvijanja mobilne aplikacije te se sljedeći dio rada bavi testiranjem i problematikom vezom uz model i aplikaciju „FlagAFlag“.



Slika 49. Uspješna klasifikacija mađarske zastave

5. Testiranje funkcionalnosti aplikacije i modela

Za ispitivanje funkcionalnosti aplikacije „FlagAFlag“ i modela odabrano je više od 100 fotografija. Brojka je odabrana iz razloga što se svaka zastava klasificira najmanje dva puta. Zastave nad kojima je provedeno ovo testiranje nisu prisutne u modelu te su neke uslikane putem kamere na mobitelu, a druge su preuzete putem online servisa.

5.1. Rezultati testiranja aplikacije i modela

Nakon testiranja rezultati su bili iznenađujući. Od 118 zastava 113 (95,6%) zastava točno je klasificirano. Ova brojka je vrlo slična brojci dobivenoj tijekom treniranja i validacije modela, međutim, potrebno je detaljnije istražiti koje su zastave pogrešno klasificirane i zašto.

Jedan od problema pri klasifikaciji predstavljale su zastave sličnih boja. Primjerice, razlika između Nizozemske i Luksemburga ovisi o svjetlini plave boje na zastavi. Usporedba ovih dviju zastava prikazana je na slici 50 (Domestic Protocol Office of the Federal Government, n.d.).

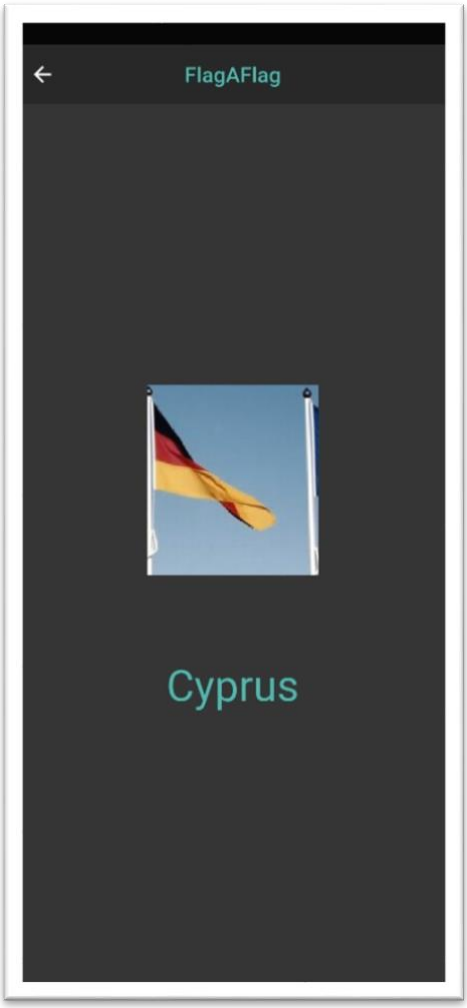
Slika 50. Usporedba nizozemske zastave i zastave Luksemburga

U gornjem primjeru, aplikacija je uspješno klasificirala zastavu lijevo kao zastavu Luksemburga, a zastavu desno kao zastavu Nizozemske. Do problema dolazi kod fotografija poput ove prikazane na slici 51. Ova slika prikazuje zastavu koja pripada Nizozemskoj, međutim, sama zastava sličnija je Luksemburškoj zbog plave boje. Ovo je jedan od problema ne samo aplikacije već i same fotografije jer zbog pozadine i načina fotografiranja čak je i ljudskim okom teško procijeniti o kojoj se točno zastavi radi. Daljnjim istraživanjem ove tematike tek treba otkriti na koji se način ovaj problem može riješiti, međutim, manje je vjerojatno da se pukim povećanjem zbirke podataka može riješiti ovaj problem. Naprotiv, možda je u ovom slučaju potrebno smanjiti zbirku podataka, jer je moguće da se samim povećanjem zbirke uvode fotografije poput ove što će negativno utjecati na točnost modela. Zbog toga je potrebno kontrolirati kvalitetu podataka koja se koristi za treniranje modela.

Slika 51. Netočna klasifikacija zastave Nizozemske

Još jedan od problema pri klasifikaciji predstavlja postotak vidljivosti zastave. Kao primjer se može uzeti ova pogrešna klasifikacija zastave Njemačke prikazane na slici 52. Zbog samo jednog vidljivog dijela zastave dolazi do pogrešne klasifikacije. Većina vidljivog dijela je

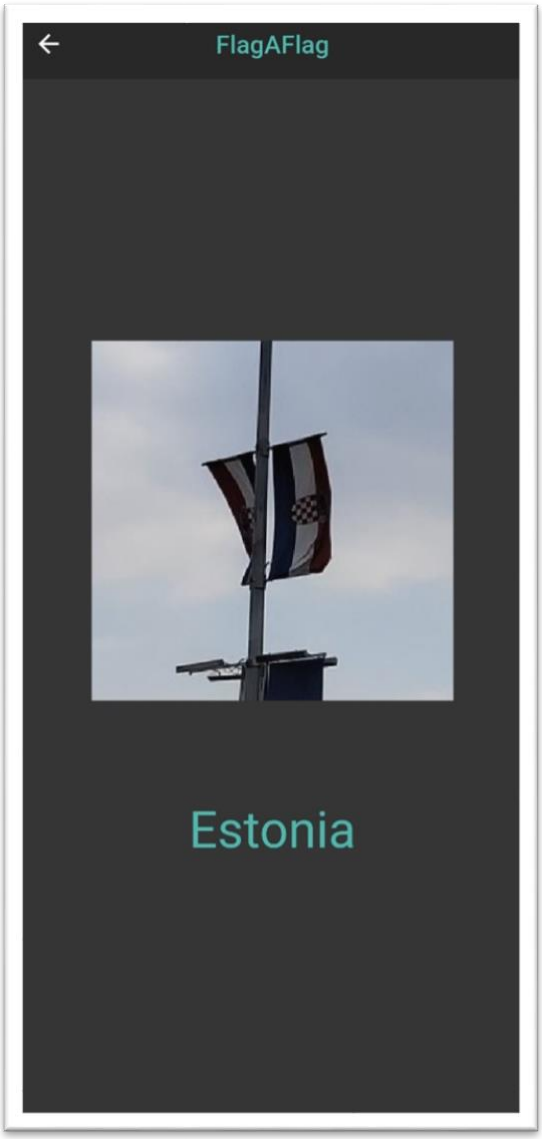
narančaste boje te je stoga fotografija klasificirana kao zastava Cipra. Ovaj problem moguće je riješiti povećanjem zbirke s točno ovakvim tipovima zastava, međutim, jednostavnije rješenje je ponovno fotografiranje zastave tako da se velika većine zastave smjesti u fokus kamere. Za ovakvo rješenje potrebno je dati upute korisniku o korištenju aplikacije te preporučenog načina fotografiranja.



Slika 52. Netočna klasifikacija zastave Njemačke

Posljednji problem u klasifikaciji je predstavljala udaljenost te neposredna okolina zastave. Zbog tamne pozadine te nemogućnosti daljnjeg odsijecanja, fotografija je klasificirana kao zastava Estonije, što je prikazano na slici 53. Ljudskim okom moguće je procijeniti da se na fotografiji nalazi hrvatska zastava, međutim, zbog manjka sličnih zastava dolazi do pogrešne

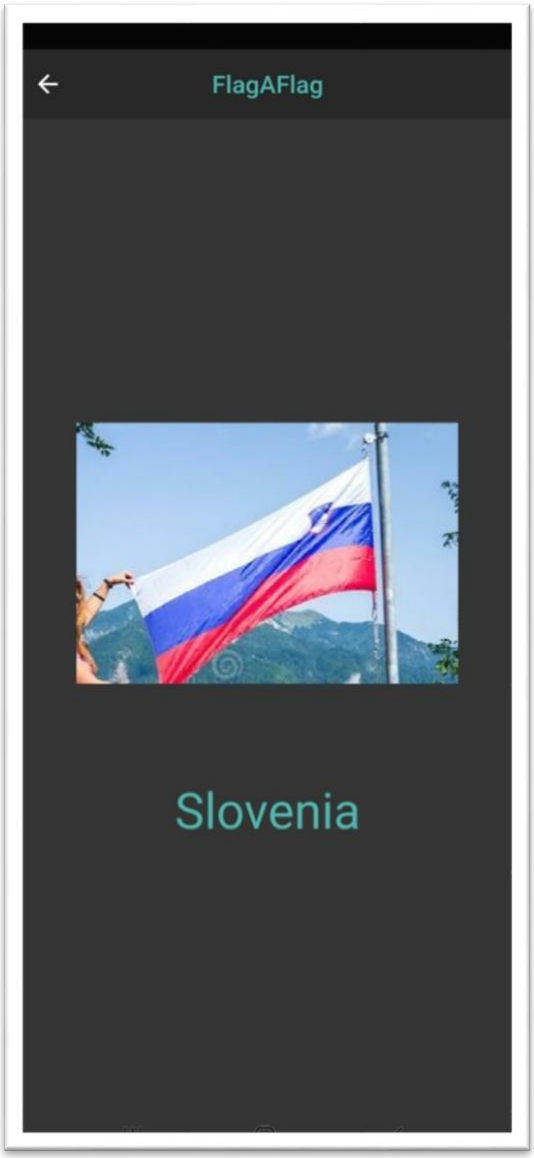
klasifikacije. Ovaj je problem je rješiv uz pomoću promjena unutar samoga modela, i to tako da se prilagode vrijednosti unutar klase ImageDataGenerator ili se sama zbirka podataka može proširiti.



Slika 53. Netočna klasifikacija hrvatske zastave

Uz nekoliko pogrešnih klasifikacija, bilo je dosta iznenađujućih identifikacija. Kao primjer može se uzeti zastava Slovenije sa slike 54. Iako je grb Slovenije vidljiv ljudskom oko, zbog svoje sličnosti sa slavenskim zastavama razumljivo je pretpostaviti kako će aplikacija ovu

fotografiju netočno klasificirati. Ipak, aplikacija je uspješno identificirala ovu zastavu te tako nadmašila očekivanja autora.

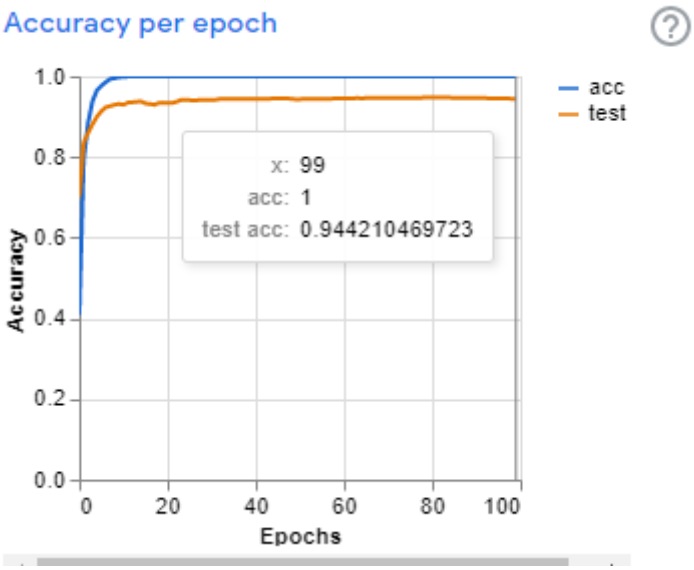


Slika 54. Točna klasifikacija slovenske zastave

5.2. Rezultati Teachable Machine modela

Radi usporedbe modela razvijenog putem TensorFlowa, obavljeno je treniranje pomoću web servisa Teachable Machine. Ova usporedba će dati više informacija o tome je li potrebno

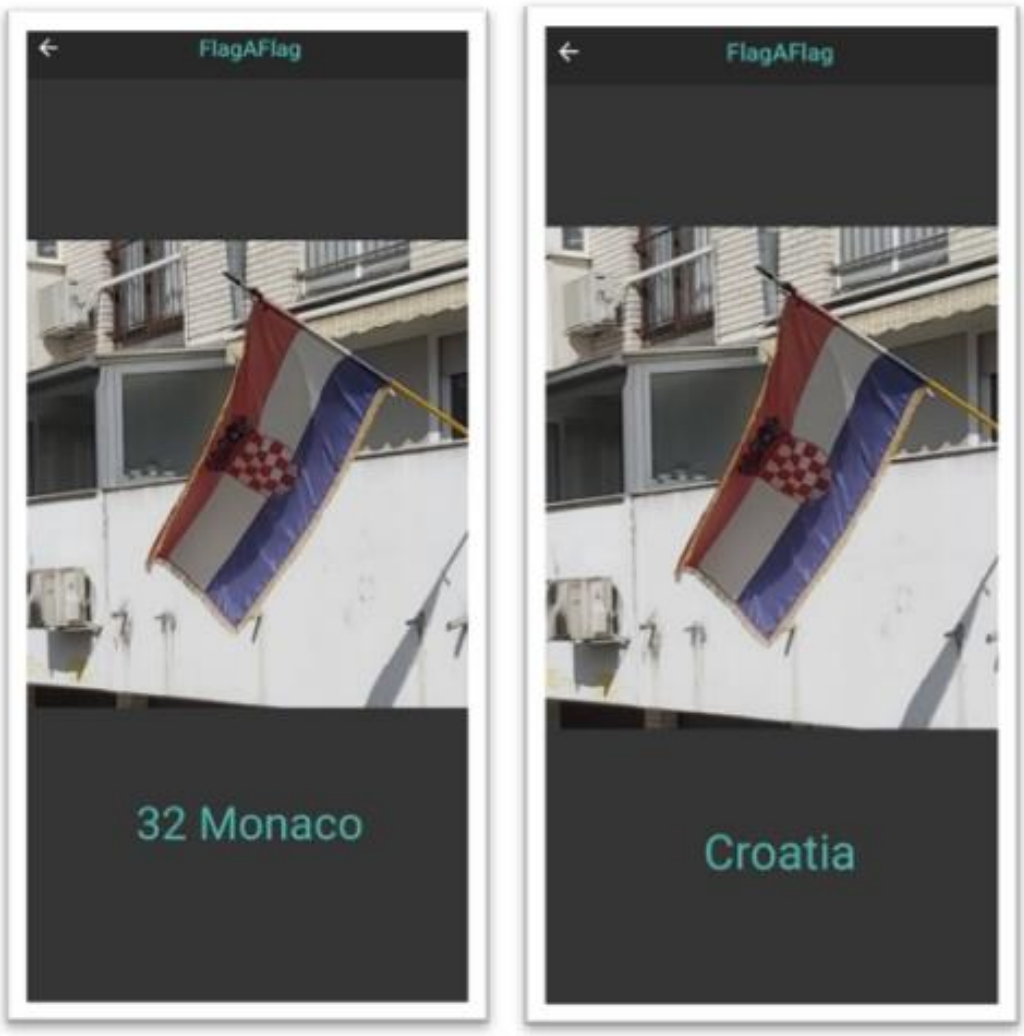
razvijati vlastite modele ili je korištenje online servisa dovoljno kako bi se razvila vlastita umjetna inteligencija. Treniranje je znatno jednostavnije s obzirom na to da je potrebno samo pridružiti zbirku fotografija. Proces treniranja putem ovog servisa je automatiziran, međutim, moguće je podesiti nekoliko parametara poput brzine učenja prije procesa treniranja. Odabirom od 100 epoha dobivena je točnost od 94%. Prikaz točnosti modela kroz epohe prikazan je na slici 55. Točnost Teachable Machine modela je poprilično jednaka točnosti modela razvijenog putem TensorFlowa, međutim, kolika je točnost ovog modela nad podatcima za testiranje?



Slika 55. Točnost Teachable Machine modela nad podacima za treniranje

Točnost Teachable Machine modela je bila iznenađujuće loša. Testiranjem je dobivena točnost od svega 30%. Primjer usporedbe vlastitog modela i onoga dobivenog uz pomoć web servisa Teachable Machine prisutan je na slici 56.

Očekivano, posebno treniran i podešen model nadmašio je model koji je razvijen putem web servisa, međutim, ovako velika razlika između dva modela nije bila očekivana. U slučaju razvijanja manjeg modela ili u edukacijske svrhe, navedeni Googleov servis se čini kao dobra opcija, ali ako je cilj razvijanje modela koji će se koristiti u komercijalne svrhe potrebno je samostalno razvijati i trenirati projektno specifične modele.



Slika 56. Lijevo: netočna klasifikacija pomoću Teachable Machine modela, desno: točna klasifikacija prema vlastitom modelu

5.3. Budućnost i daljnji razvoj aplikacije

Prema mišljenju autora, aplikacija „FlagAFlag“ je spremna za eksperimentalnu primjenu u stvarnom svijetu, no potrebno bi bilo još unaprijediti model i aplikaciju. Unatoč visokoj točnosti, potrebno je proširiti zbirku podataka i količinu zastava kako bi se dodatno unaprijedila identifikacija, tj. klasifikacija zastava. Idealno bi bilo prikupiti sve svjetske zastave te nad njima provesti treniranje i potom omogućiti klasifikaciju svih svjetskih zastava, no opseg posla i

količina podataka bi bili znatno veći. U slučaju dodatnog proširenja zbirke, potrebno bi bilo doraditi model te prilagoditi umjetnu neuronsku mrežu.

Unutar mobilne aplikacije moguće je dodati nekoliko dodatnih opcija poput spremanja fotografije, geolokacije ili slično. Jedna od mogućih nadogradnji bi bila i opcija podjele uslikanih i klasificiranih fotografija. Uslikane fotografije zastava bi se mogle dijeliti s drugim korisnicima ili se poslati na poslužitelj na kojemu bi se prikupljale sve fotografije uslikane putem aplikacije „FlagAFlag“. Na taj način bi se mogla dodatno povećavati zbirka fotografija, a time bi i svaki korisnik imao priliku doprinijeti razvijanju modela, što bi svakako kroz određeno razdoblje moglo unaprijediti model te osigurati veću točnost aplikacije u budućnosti. Naravno, fotografije bi se morale i dalje ručno provjeravati nakon klasifikacije kako bi se osigurala što veća kvaliteta podataka.

6. Zaključak

Cilj istraživanja u sklopu ovog rada bio je razviti mobilnu aplikaciju pod nazivom „FlagAFlag“ čija je svrha identifikacija zastava s visokom preciznošću. Kako bi navedeno bilo moguće, bilo je potrebno prikupiti zbirku podataka na temelju koje će model biti treniran. Postupak prikupljanja podataka proveden je pomoću, posebno za ovu svrhu izrađene, Python skripte koja je automatizirala preuzimanje fotografija sa interneta za zbirku podataka. Kvaliteta i veličina zbirke podataka korištena za treniranje modela imala je znatan utjecaj na točnost i preciznost modela. Eksperimentiranjem nad podacima i modelom potvrđena je hipoteza postavljena u istraživanju ovoga rada, tj. da položaj zastave utječe na točnost modela. Zbog toga je autor odlučio dodatno obraditi i proširiti zbirku podataka uz pomoć klase ImageDataGenerator dohvaćene pomoću Keras API-a i dvije dodatne skripte koje su omogućile zrcaljenje i rotaciju originalnih fotografija. Na temelju ovih podataka istreniran je model čiju arhitekturu čine umjetne neuronske mreže. S obzirom na to da je potrebno klasificirati fotografije zastava, odabrana je konvolucijska neuronska mreža. Kroz rad je opisana arhitektura ove mreža koja je razvijena uz pomoću razvojnog okruženja TensorFlow i Keras API-a. Uz izgradnju modela opisani su česti problemi pri izgradnji modela za strojno učenje, poput prenaučenosti te utjecaj ove pojave na točnost i preciznost modela. Ovaj problem je riješen implementacijom slojeva ispuštanja što je znatno poboljšalo točnost validacijskog podatkovnog skupa. Nakon ove implementacije i ponovne procjene, preciznost modela je dosegla 94% nakon otprilike 100 epoha. Ova brojka bila je zadovoljavajuća te je stoga treniranje završeno i započeo je proces izgradnje mobilne aplikacije unutar Flutter okruženja.

Kako bi mobilna aplikacija bila funkcionalna bilo je potrebno implementirati funkciju kamere i galerije te je stoga napravljeno korisničko sučelje koje je vrlo slično kameri standardnih mobitela. Problemi implementacije modela riješeni su pretvorbom Keras modela u Lite verziju što je omogućilo rad tflite plugina. Ovom implementacijom je mobilna aplikacija dobila sposobnost klasifikacije zastava. Unutar ovog dijela rada implementirana je i funkcija odsijecanja. Funkcija odsijecanja je imala znatan utjecaj na točnost modela jer je omogućila uklanjanje nepotrebnih detalja fotografije. Sa završenom implementacijom ove funkcije, mobilna aplikacija je bila spremna za testiranje u stvarnom svijetu.

Testiranje mobilne aplikacije „FlagAFlag“ obavljeno je na temelju 118 fotografija nad kojima model nije bio treniran. Rezultati su bili iznenađujući jer je čak 113 (95.6%) zastava točno klasificirano. Mobilna aplikacija je uspješno klasificirala veliki broj fotografija, međutim, zastave Nizozemske, Francuske i Luksemburga predstavljaju problem zbog vrlo slične strukture i nijansi boja. Također, obavljena je i usporedba s web servisom Teachable Machine te su prikazani rezultati usporedbe. Model razvijen u sklopu ovog završnog rada pokazao je znatno veću točnost zbog mogućih modifikacija i promjena u programskom kodu, kao i zbog korištenja validacijskog podatkovnog skupa. Na kraju rada prikazana su potencijalna rješenja te načini na koje bi se model daljnje mogao unapređivati kao i mogući pravci daljnjeg razvoja aplikacije „FlagAFlag“.

7. Literatura

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ,Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jozefowicz, R., Jia, Y., Kaiser, L., Kudlur, M., Levenberg, J…. Jia, Y. (2015a). *Training and evaluation with the built-in methods.* Dohvaćeno 12. travnja 2021., iz Tensorflow: https://www.tensorflow.org/guide/keras/train_and_evaluate

2. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jozefowicz, R., Jia, Y., Kaiser, L., Kudlur, M., Levenberg, J…. Jia, Y. (2015b). *TensorFlow Lite guide.* Dohvaćeno 12. travnja 2021., iz Tensorflow: https://www.tensorflow.org/lite/guide

3. Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). *Understanding of a convolutional neural network.* Dohvaćeno 12. travnja 2021., iz IEEE Xplore: https://ieeexplore.ieee.org/abstract/document/8308186

4. Anyoha, R. (2017). *The History of Artificial Intelligence*. Dohvaćeno 12. travnja 2021., iz https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/

5. Appen. (2018). *Great Machine Learning Data: It's Not About Quantity or Quality — It's About Both*. Dohvaćeno 12. travnja 2021., iz Appen: https://appen.com/blog/great-machine-learning-data-quantity-or-quality/

6. Bhandari, A. (2020). *Image Augmentation on the fly using Keras ImageDataGenerator!* Dohvaćeno 12. travnja 2021., iz Analytics Vidya: https://www.analyticsvidhya.com/blog/2020/08/image-augmentation-on-the-fly-using-keras-imagedatagenerator/

7. Brownlee, J. (2018a). *A Gentle Introduction to Dropout for Regularizing Deep Neural Networks*. Dohvaćeno 12. travnja 2021., iz Machine Learning Mastery: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

8. Brownlee, J. (2018b). *Difference Between a Batch and an Epoch in a Neural Network*. Dohvaćeno 12. travnja 2021., iz Machine Learning Mastery: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

9. Brownlee, J. (2019a). *Overfitting and Underfitting With Machine Learning Algorithms*. Dohvaćeno 12. travnja 2021., iz Machine Learning Mastery: https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/

10. Brownlee, J. (2019b). *TensorFlow 2 Tutorial: Get Started in Deep Learning With tf.keras*. Dohvaćeno 12. travnja 2021., iz Machine Learning Mastery: https://machinelearningmastery.com/tensorflow-tutorial-deep-learning-with-tf-keras/

11. Brownlee, J. (2020). *A Gentle Introduction to the Rectified Linear Unit (ReLU).* Dohvaćeno 21. travnja 2021., iz Machine Learning Mastery: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

12. Brownlee, J. (2021). *How to Choose an Activation Function for Deep Learning*. Dohvaćeno 12. travnja 2021., iz Machine Learning Mastery: https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/

13. Chardan, J. (2012). *An image based colorimetric technique for portable blood gas analysis.* Dohvaćeno 12. travnja 2021., iz Semantic Scholar: https://www.semanticscholar.org/paper/An-image-based-colorimetric-technique-for-portable-Chandran/1fd8ceb34e9feebfed41b5f7fd893704592aa1f5

14. Clark, A. (2021). *Pillow*. Dohvaćeno 12. travnja 2021., iz Pypi.org: https://pypi.org/project/Pillow/

15. Cornelisse, D. (2018). *An intuitive guide to Convolutional Neural Networks*. Dohvaćeno iz 12. travnja 2021., freeCodeCamp: https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/

16. Cowley, E., & Radich, Q. (2019). *What is a machine learning model?* Dohvaćeno 12. travnja 2021., iz Microsoft: https://docs.microsoft.com/en-us/windows/ai/windows-ml/what-is-a-machine-learning-model

17. Dennis. (2019). *Danish Flag.* Dohvaćeno 11. travnja 2021., iz Diary of Dennis: https://diaryofdennis.com/2019/09/25/danish-flag/

18. Depositphotos. (n.d.). *Ballot box with Croatian flag on background — Stock Image & Photo.* Dohvaćeno 11. travnja 2021., iz depositphotos: https://depositphotos.com/326246184/stock-photo-ballot-box-croatian-flag-background.html

19. Derat, A. (2017). *Applied Deep Learning - Part 4: Convolutional Neural Networks*. Dohvaćeno 12. travnja 2021., iz towards data science: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

20. Descamps, B. (2017). *Custom Optimizer in TensorFlow*. Dohvaćeno 12. travnja 2021., iz towards data science: https://towardsdatascience.com/custom-optimizer-in-tensorflow-d5b41f75644a

21. Domestic Protocol Office of the Federal Government. (n.d.). *Information to flag displays.* Dohvaćeno 11. travnja 2021., iz Domestic Protocol Office of the Federal Government: https://www.protokoll-inland.de/Webs/PI/EN/flag-displays/general-information/informations/informations-node.html

22. Donges, N. (2019). *WHAT IS TRANSFER LEARNING? EXPLORING THE POPULAR DEEP LEARNING APPROACH.* Dohvaćeno 12. travnja 2021., iz builtin: https://builtin.com/data-science/transfer-learning

23. Dreamstime. (n.d.). *Slovenian flag waving in wind.* Dohvaćeno 12. travnja 2021., iz dreamstime: https://www.dreamstime.com/slovenian-flag-waving-wind-national-republic-slovenia-high-pole-clear-blue-sky-sole-image184917135

24. Dunđer, I. (2020). *Machine Translation System for the Industry Domain and Croatian Language. Journal of Information and Organizational Sciences*, 44(1), pp. 33-50

25. Dunđer, I., & Pavlovski, M. (2019a). *Behind the Dystopian Sentiment: a Sentiment Analysis of George Orwell's 1984*. 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 577-582

26. Dunđer, I., & Pavlovski, M. (2019b). *Through the Limits of Newspeak: an Analysis of the Vector Representation of Words in George Orwell's 1984*. 42nd International Convention

on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 583-588

27. Dunđer, I., Seljan, S., & Pavlovski, M. (2020). *Automatic Machine Translation of Poetry and a Low-Resource Language Pair*. 43rd International Convention on Information, Communication and Electronic Technology (MIPRO), pp. 1034-1039
28. Dunđer, I., Seljan, S., & Stančić, H. (2015). *Koncept automatske klasifikacije registraturnoga i arhivskoga gradiva*. 48. savjetovanje hrvatskih arhivista (HAD), pp. 195-211
29. Escrivá, M. D., Mendonça, G. V., & Joshi, P. (2018). *Learn OpenCV 4 by Building Projects: Build real-world computer vision and image processing applications with OpenCV and C++*, 2nd Edition (2nd ed.). Packt Publishing.
30. Flutter. (n.d.). *Introduction to widgets*. Dohvaćeno 11. travnja 2021., iz Flutter: https://flutter.dev/docs/development/ui/widgets-intro
31. GeeksforGeeks. (2020). *Selenium Python Tutorial*. Dohvaćeno 12. travnja 2021., iz GeeksforGeeks: https://www.geeksforgeeks.org/selenium-python-tutorial/
32. Gómez, R. (2018). *Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names*. Dohvaćeno 12. travnja 2021., iz Raúl Gómez blog: https://gombru.github.io/2018/05/23/cross_entropy_loss/
33. Google Developers. (n.d.). *The ML Mindset | Introduction to Machine Learning Problem Framing*. Dohvaćeno 12. travnja 2021., iz Google Developers: https://developers.google.com/machine-learning/problem-framing/big-questions
34. Grossfeld, B. (2020). *Deep learning vs machine learning: a simple way to understand the difference*. Dohvaćeno 11. travnja 2021., iz The Library: https://www.zendesk.com/blog/machine-learning-and-deep-learning/
35. Haleva, R. (2020). *How to Break GPU Memory Boundaries Even with Large Batch Sizes*. Dohvaćeno 12. travnja 2021., iz towards data science: https://towardsdatascience.com/how-to-break-gpu-memory-boundaries-even-with-large-batch-sizes-7a9c27a400ce

36. Hsu, H. (2020). *HOW DO NEURAL NETWORK SYSTEMS WORK?* Dohvaćeno 12 travnja 2021., iz CHM blog: https://computerhistory.org/blog/how-do-neural-network-systems-work/

37. IBM Cloud Education. (2020). *Machine learning*. Dohvaćeno 12 travnja 2021., iz IBM: https://www.ibm.com/cloud/learn/machine-learning

38. Kaggle. (n.d.). *kaggle Datasets.* Dohvaćeno 11. travnja 2021., iz kaggle.com: https://www.kaggle.com/datasets

39. Karczewski, D. (2020). *What Is The Best Language For Machine Learning In 2021?* Dohvaćeno 12 travnja 2021., iz ideamotive: https://www.ideamotive.co/blog/what-is-the-best-language-for-machine-learning

40. Kingma, D. P., & Ba, J. (2017). *Adam: A Method for Stochastic Optimization*. Dohvaćeno 12. travnja 2021., iz arXiv.org: https://arxiv.org/abs/1412.6980

41. Lee, K. (2017). *Identifying Patterns: Why Do Some Flags Look Similar?* Dohvaćeno 12 travnja 2021., iz Pro Quest: https://blogs.proquest.com/culturegrams/identifying-patterns-why-do-some-flags-look-similar/

42. Lugović, S., Dunđer, I., & Horvat, M. (2016). *Techniques and applications of emotion recognition in speech*. 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1278-1283

43. Mahendru, K. (2019). *A Detailed Guide to 7 Loss Functions for Machine Learning Algorithms with Python Code*. Dohvaćeno 12. travnja 2021., iz Analytics Vidhya: https://www.analyticsvidhya.com/blog/2019/08/detailed-guide-7-loss-functions-machine-learning-python-code/

44. Microsoft. (n.d.). *Microsoft Azure.* Dohvaćeno 11. travnja 2021., iz Microsoft Azure: https://azure.microsoft.com/en-us/free/machine-learning/

45. MV, P. (2020). *Top 10 Reasons Why 87% of Machine Learning Projects Fail*. Dohvaćeno 12. travnja 2021., iz Dzone: https://dzone.com/articles/top-10-reasons-why-87-of-the-machine-learning-proj

46. Parthasarathy, S. (2019). *Difference between Traditional programming versus Machine Learning from a PM perspective*. Dohvaćeno 12, travnja 2021., iz Product Coalition: https://productcoalition.com/difference-between-traditional-programming-versus-machine-learning-from-a-pm-perspective-3802b02bc7f6

47. Pokhrel, S. (2019). *How Does Computer Understand Images?* Dohvaćeno 12. travnja 2021., iz towards data science: https://towardsdatascience.com/how-does-computer-understand-images-c1566d4537bf

48. Ramos, L. P. (n.d.). *How to Run Your Python Scripts*. Dohvaćeno 12. travnja 2021., iz Real Python: https://realpython.com/run-python-scripts/#author

49. Raschka, S. (n.d.). *How are Artificial Intelligence and Machine Learning related?* Dohvaćeno 12. travnja 2021., iz sebastianraschka: https://sebastianraschka.com/faq/docs/ai-and-ml.html

50. Richardson, L. (2020). *beautifulsoup4*. Dohvaćeno 12. travnja 2021., iz Pypi.org: https://pypi.org/project/beautifulsoup4/

51. Sells, C. (2021). *What's New in Flutter 2*. Dohvaćeno 12. travnja 2021., iz Medium: https://medium.com/flutter/whats-new-in-flutter-2-0-fe8e95ecc65

52. Singh, V. (2020). *What is Frameworks? [Definition] Types of Frameworks*. Dohvaćeno 12. travnja 2021., iz hackr.io: https://hackr.io/blog/what-is-frameworks

53. Teachable Machine. (n.d.). *Teachable Machine.* Dohvaćeno 11. travnja 2021., iz Teachable Machine: https://teachablemachine.withgoogle.com/train/image

54. The Flag Shop. (n.d.a). *Dutch | Netherlands Flag 5Ft X 3Ft.* Dohvaćeno 11. travnja 2021., iz theflagshop: https://www.theflagshop.co.uk/dutch-netherlands-flag.html

55. The Flag Shop. (n.d.b). *Holland | Netherlands | Dutch Hand Waving Flag.* Dohvaćeno 11. travnja 2021., iz theflagshop: https://www.theflagshop.co.uk/flags/world-flags/holland-netherlands-flags/holland-netherlands-dutch-hand-waving-flag.html

56. Thomas, G. (2019). *What is Flutter and Why You Should Learn it in 2020*. Dohvaćeno 12. travnja 2021., iz freeCodeCamp: https://www.freecodecamp.org/news/what-is-flutter-and-why-you-should-learn-it-in-2020/

57. Tokopedia. (n.d.). *Bendera negara slovakia.* Dohvaćeno 11. travnja 2021., iz tokopedia: https://www.tokopedia.com/starcom77/bendera-negara-slovakia

58. Vemu. (n.d.)., *2016: 25th Anniversary of the Restoration of the Republic of Estonia!* Dohvaćeno 12. travnja 2021., iz VEMU: https://www.vemu.ca/index.php/en/events-events/1229-august-20-2016-25th-anniversary-of-the-restoration-of-independence-of-the-republic-of-estonia

59. Warden, P. (2017). *How many images do you need to train a neural network?* Dohvaćeno 12. travnja 2021., iz Pete Warden's blog: https://petewarden.com/2017/12/14/how-many-images-do-you-need-to-train-a-neural-network/

60. Whang, S. E., Roh, Y., & Heo, G. (2019). *A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective.* Dohvaćeno 12. travnja 2021., iz IEEE Xplore: https://ieeexplore.ieee.org/document/8862913

61. Wikipedia. (n.d.b). *Mean squared error*. Dohvaćeno 12. travnja 2021., iz Wikipedia: https://en.wikipedia.org/wiki/Mean_squared_error

62. Wikipedia. (n.d.a). *Neural network.* Dohvaćeno 12. travnja 2021., iz Wikipedia: https://en.wikipedia.org/wiki/Neural_network

63. Williams-Ellis, M. (n.d.). *Photo of the Croatian Flag in Dubrovnik, Croatia.* Dohvaćeno 12 travnja 2021., iz Matthew Williams-Ellis photography: https://matthewwilliamsellis.photoshelter.com/image/I0000yVQUyoX.DKI

64. Wood, T. (2020). *Convolutional Neural Network.* Dohvaćeno 21. travnja 2021., iz DeepAI: https://deepai.org/machine-learning-glossary-and-terms/convolutional-neural-network

65. Yegulalp, S. (2019). *What is TensorFlow? The machine learning library explained*. Dohvaćeno 12. travnja 2021., iz InfoWorld: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html

Popis slika

Popis grafikona

Popis tablica

Prilozi

Prilog 1 - Razvoj skripte za prikupljanje podataka

```
from selenium import webdriver
from bs4 import BeautifulSoup
import requests
import urllib.request
import time
import sys
import os
from selenium.webdriver.support import expected_conditions as EC
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait

import pandas as pd

tablica = pd.read_csv("lista_zastava.csv")
lista = tablica["Država"].tolist()

print(lista)

for x in lista:
    drzava=x
    try:
        os.mkdir(drzava)
    except:
        print("Pogreška!")



    chromedriver = "C:\chromedriver.exe"
    driver = webdriver.Chrome(executable_path=chromedriver)
    driver.get("https://www.google.com/imghp?hl=en")




    try:
        element = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.CSS_SELECTOR, "#sbtc > div > div.a4bIc > input"))
```

```
            )
        element.send_keys("German flag")
        element.send_keys(Keys.RETURN)

        x = 0

        while x<20:

            driver.execute_script("window.scrollBy(0,document.body.scrollHeight)")
            time.sleep(5)
            x+=1
            try:
                driver.find_element_by_xpath("/html/body/div[2]/c-
wiz/div[3]/div[1]/div/div/div/div/div[5]/input").click()
            except Exception as e:
                pass

        soup = BeautifulSoup(driver.page_source, 'html.parser')
        driver.close()

    except():
        driver.quit()



    imgs = soup.find_all("img", class_="rg_i")


    count = 0
    for x in imgs:

        try:

            urllib.request.urlretrieve(x["src"],"C:/Users/ivan/Desktop/Data_gathering/"+drzava+"/"+str(count)+".jpg")
            count+=1
            print("Broj preuzetih datoteka je : "+str(count),end='\r')
        except Exception as e:
            pass
```

Prilog 2 - Skripta za rotaciju i zrcaljenje fotografija

```
from PIL import Image
import os

def zrcalo():
    broj=2000
    maksimalan_broj=48
    for image in os.listdir(os.getcwd()):
        img = Image.open(image).convert("RGB")
        img = img.transpose(Image.FLIP_LEFT_RIGHT)
        img.save("flipped"+str(broj)+".jpg")
        broj+=1
        maksimalan_broj-=1
        if maxn ==0:
            exit()
        else:
            print("continue")

zrcalo()
```

```
def okreni(rotatedimg):
  broj=1000
  maksimalan_broj=40
  for Image in os.listdir(os.getcwd()):
    img = Image.open(Image).convert("RGB")
    img=img.rotate(rotatedimg, expand=True)
    img.save("rotated"+str(broj)+".jpg")
    broj+=1
    maksimalan_broj-=1
    if maksimalan_broj==0:
      exit()
    else:
      print("Okrećem!")

okreni(270)
```

Prilog 3 - Programski kod za treniranje i razvoj modela

```
import tensorflow as tf
import os
import matplotlib.pyplot as plt
import keras_preprocessing
from tensorflow.keras.preprocessing.image import ImageDataGenerator




class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if (logs.get("val_acc")>0.94):
            self.model.stop_training= True
callbacks=myCallback()









model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(224, 224, 3)),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Conv2D(32, (3,3), activation="relu"),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, (3,3), activation="relu"),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Conv2D(32, (3,3), activation="relu"),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, (3,3), activation="relu"),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Conv2D(32, (3,3), activation="relu"),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Flatten(),
```

```
    tf.keras.layers.Dense(512, activation="relu"),
    tf.keras.layers.Dropout(0.35),
    tf.keras.layers.Dense(48, activation='softmax')
 ])

model.summary()


training_dir="data/flags/training"

datagen = ImageDataGenerator(validation_split=0.20, rescale=1./255,
rotation_range=30, brightness_range=[0.2,1.8],
width_shift_range=0.2, height_shift_range=0.2)



train_generator = datagen.flow_from_directory(
    training_dir,
    subset='training',
    target_size=(224,224),
    class_mode="categorical",
    shuffle=True,
    batch_size=64
)

validation_generator = datagen.flow_from_directory(
    training_dir,
    subset='validation',
    target_size=(224,224),
    class_mode="categorical",
    shuffle=True,
    batch_size=64
)
#optimizer = keras.optimizers.Adam(lr=0.01)

model.compile(optimizer="Adam", loss="categorical_crossentropy", metrics=["accuracy"])
model.fit(train_generator, epochs=100, validation_data = validation_generator,
 verbose = 1, callbacks=[callbacks],)
model.save("Flag48.h5")


labels = '\n'.join(sorted(train_generator.class_indices.keys()))
with open('labels48TEST.txt', 'w') as f:
    f.write(labels)
```

Prilog 4 - Programski kod vezan uz pretvorbu TensorFlow modela u Lite verziju

```
import tensorflow as tf
model= tf.keras.models.load_model("Flag48.h5")
model.summary()
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

with open('FAF48.tflite', 'wb') as f:
  f.write(tflite_model)
```

Prilog 5 - Programski kod vezan uz main.dart datoteku

```
import 'dart:ui';
import 'package:image_cropper/image_cropper.dart';
import 'package:flutter/services.dart';
import 'package:flutter/material.dart';
import 'package:image_picker/image_picker.dart';
import 'package:tflite/tflite.dart';
import 'dart:async';
import 'dart:io';
import 'package:camera/camera.dart';


Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  final cameras = await availableCameras();
  final firstCamera = cameras.first;


  runApp(
    MaterialApp(
      debugShowCheckedModeBanner: false,
      darkTheme: ThemeData(
      brightness: Brightness.dark),
      home: App(

        camera: firstCamera,
      ),


    ),
  );
}

class App extends StatefulWidget {
  final CameraDescription camera;
  const App({Key key, @required this.camera,}):super(key:key);
  @override
  _AppState createState() => _AppState();
}

class _AppState extends State<App> {

  static List _results;
  static File _image;
```

```
CameraController _controller;
Future<void> _initializeControllerFuture;

File image;
@override
void initState() {
 super.initState();
 loadModel();
 classifyImage(image);

 _controller = CameraController(

 widget.camera,

 ResolutionPreset.veryHigh,
 );


 _initializeControllerFuture = _controller.initialize();

}

@override
Widget build(BuildContext context) {
 SystemChrome.setEnabledSystemUIOverlays([]);
 return Scaffold(
  resizeToAvoidBottomPadding: false,
  extendBodyBehindAppBar: true,
  appBar: AppBar(
   centerTitle: true,
   backgroundColor: Colors.transparent,
   //backgroundColor: Color(0x44000000),
   elevation: 0,

   ),
   body:  FutureBuilder<void>  (
   future: _initializeControllerFuture,
   builder: (context, snapshot) {
    final size = MediaQuery.of(context).size;

    final deviceRatio = size.width / size.height;

    if (snapshot.connectionState == ConnectionState.done) {

     return Transform.scale(
```

```
        scale: 1 ,
        child: Center(
        child: AspectRatio(
      aspectRatio: deviceRatio,
      child: CameraPreview(_controller)),
              ),
            );
    } else {

      return Center(child: CircularProgressIndicator());
    }
  },
),


floatingActionButtonLocation: FloatingActionButtonLocation.endFloat,

floatingActionButton: Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: <Widget>[
    SizedBox(width: 180,),
  FloatingActionButton (
  child: Icon(Icons.camera),
  heroTag: "Camera",

  onPressed: () async {

    try {

      await _initializeControllerFuture;


      await loadModel();

      await cameraImage();




      Navigator.of(
      context).push( MaterialPageRoute(builder:
      (BuildContext context)=>CameraScreen(testing: _image.path)));
```

```
      } catch (e) {

        print(e);
      }
    },
  ),
  SizedBox(height: 125, width:80),

  FloatingActionButton(
    heroTag: "Gallery",
    child: Icon(Icons.image),
    onPressed: () async{
      try{
        await loadModel();
        await pickImage();


        Navigator.of(
  context).push( MaterialPageRoute(builder:
  (BuildContext context)=>GalleryScreen()));


    }
    catch(e){
      print(e);
    }
    }
  ),

  ]
 ),

);
}
Future cameraImage() async{
 var imageCam= await _controller.takePicture();
 if (imageCam == null) return null;
 File croppedFile = await ImageCropper.cropImage(
  sourcePath: imageCam.path,
  aspectRatioPresets: [
    CropAspectRatioPreset.square,
    CropAspectRatioPreset.original,
  ],
```

```
    androidUiSettings: AndroidUiSettings(
        toolbarTitle: 'FlagAFlag',
        toolbarColor: Colors.black,
        toolbarWidgetColor: Colors.teal.shade300,
        initAspectRatio: CropAspectRatioPreset.original,
        lockAspectRatio: false),
    iosUiSettings: IOSUiSettings(
      minimumAspectRatio: 1.0,
    )
  );
  setState(() {
       _image=croppedFile;
      });
  await classifyImage(croppedFile);
}

Future pickImage() async {
  var image = await ImagePicker.pickImage(source: ImageSource.gallery);
  if (image == null) return null;
  File croppedFile = await ImageCropper.cropImage(
    sourcePath: image.path,
    aspectRatioPresets: [
      CropAspectRatioPreset.square,
      CropAspectRatioPreset.original,
    ],
    androidUiSettings: AndroidUiSettings(
        toolbarTitle: 'FlagAFlag',
        toolbarColor: Colors.black,
        toolbarWidgetColor: Colors.teal.shade300,
        initAspectRatio: CropAspectRatioPreset.original,
        lockAspectRatio: false),
    iosUiSettings: IOSUiSettings(
      minimumAspectRatio: 1.0,
    )
  );
  setState(() {
       _image=croppedFile;
      });

  await classifyImage(croppedFile);

}

Future classifyImage(File image) async {
    var output = await Tflite.runModelOnImage(
```

```
        path: image.path,
        numResults: 48,
        threshold: 0.2,
        imageMean: 1,
        imageStd:255);
    setState(() {
      _results = output;

    });
  }

  Future loadModel() async {
    await Tflite.loadModel(
      model: "assets/FAF48.tflite",
      labels: "assets/labels48.txt",
    );

  }

  @override
  void dispose() {
    Tflite.close();
    _controller.dispose();
    super.dispose();
  }
}
class CameraScreen extends StatefulWidget {
  final String testing;
  const CameraScreen({Key key, @required this.testing}) : super(key: key);

  _CameraScreenState createState() => _CameraScreenState();
}

class _CameraScreenState extends State<CameraScreen> {


  @override
  Widget build(BuildContext context) {


    return Scaffold(
      appBar: AppBar(
        centerTitle: true,

        backgroundColor: Color(0x44000000),
```

```
      elevation: 0,
      title: Text("FlagAFlag",
      style: TextStyle(color: Color(0xFF4DB6AC )),),),

    body:
    Container(
      width: MediaQuery.of(context).size.width,
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.center,
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Image.file(File(widget.testing)),
          SizedBox(
            height:80,),
            Text("${_AppState._results[0]["label"]}"??"",
            style: TextStyle(
              color: Colors.teal.shade300,
              fontSize: 40.0
                ),)
        ],
      )

   ));


  }
   @override
  void initState() {
   super.initState();
  }
  @override
  void dispose() {
   super.dispose();
  }
}


class GalleryScreen extends StatefulWidget {

  @override
  _GalleryScreenState createState() => _GalleryScreenState();
}

class _GalleryScreenState extends State<GalleryScreen> {
```

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(appBar: AppBar(
        centerTitle: true,
        //backgroundColor: Colors.transparent,
        backgroundColor: Color(0x44000000),
        elevation: 0,
        title: Text("FlagAFlag",
        style: TextStyle(color: Color(0xFF4DB6AC )),
        ),
        ),

        body: Container(
        width: MediaQuery.of(context).size.width,
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.center,
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Image.file(_AppState._image),
            SizedBox(
              height:80,),
              Text("${_AppState._results[0]["label"]}"??"",
              style: TextStyle(
                color: Colors.teal.shade300,
                fontSize: 40.0
              ),)
          ],
        )
      ));


  }
  @override
  void initState() {
    super.initState();
  }
  @override
  void dispose() {
    super.dispose();
}
}
```

Sažetak

FlagAFlag – razvoj aplikacije za identifikaciju zastava

Sažetak

U radu je opisan razvoj aplikacije koja je nazvana FlagAFlag. Aplikacija je napravljena u edukacijske svrhe, služi kako bi korisnik što lakše identificirao nepoznate zastave. Rad je podijeljen u četiri poglavlja. Prvi dio ovoga rada odnosi se na prikupljanje podataka te slika potrebnih za realizaciju ove aplikacije. Također ukazani su problemi vezani uz prikupljanje podataka te potencijalna rješenja istih kako bi se podaci mogli primijeniti u strojnome učenju. Srž ove aplikacije i završnoga rada je sama kreacija modela i treniranje provedeno na pažljivo selektiranim podacima dobivenih tijekom prikupljanja podataka. Uz pomoć okruženja TensorFlow, izgrađen je model za klasifikaciju zastava te su objašnjene metode i značajke korištene tijekom treniranja i validacije podataka. Nakon izgrađenoga modela, prikazana je implementacija istoga u razvojnom okruženju Flutter te je uz pomoć programskoga jezika Dart izrađena aplikacija s ciljem da korisnik može uz pomoć pametnoga mobitela na dohvat ruke identificirati zastavu. Na samome kraju prikazana je demonstracija korištenja aplikacije s primjerima uslikanima putem mobitela ili učitanima putem korisnikove galerije.

Ključne riječi: FlagAFlag, zastave, konvolucijske neuronske mreže, mobilna aplikacija, TensorFlow, Flutter, umjetna inteligencija

Summary

FlagAFlag – development of an application for flag identification

Summary

This thesis describes the development of an application called FlagAFlag. The application was created for educational purposes, and allows the user to identify unknown flags in an easier way. The paper is divided into four chapters. The first part of the thesis deals with the collection of data and images necessary to complete this application. This thesis also identifies problems in data collection and possible solutions so that the data can be used properly in machine learning. The core of this application and this bachelor thesis is the creation of a model and the training that is done on carefully obtained data during the process of data collection. Using the Tensorflow framework, a flag classification model is created, and the methods and features used during training and data validation are clarified. After creating the model, the implementation in the Flutter framework is shown, and by using the Dart programming language; an application is created that allows a user to identify a flag with the help of a smartphone within arm's reach. Finally, a real-world demonstration of the application is shown using examples taken with the phone camera or loaded from the user gallery.

Keywords: FlagAFlag, flags, convolutional neural networks, mobile application, TensorFlow, Flutter, artificial intelligence