SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6581

Detekcija objekata na slikama

Tomislav Kurtović

Zagreb, lipanj 2020.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod

1.1. Računalni vid

Računalni vid grana je umjetne inteligencije u kojoj se računalo "uči" interpretirati slike. Danas se primjenjuje u:

- Optičkom prepoznavanju znakova
- Sigurnosnim sustavima
- Medicini

Započinje početkom 1970-tih, međutim tada ljudi nisu znali niti što bi trebali učiniti kako bi razvili ovakve sustave niti koliko bi takav razvoj trajao. Tadašnji vrsni znanstvenici i profesori smatrali su da će problem moći riješiti "unutar jednog ljeta, tako da priključe kameru na računalo". (Szeliski, 2010)

1.2. Detekcija objekata

Detekcija objekata podgrana je računalnog vida te označava postupak kojim računalo nastoji odrediti razred i položaj objekta na slici. Razred ili klasa (engl. *Class*) označava vrstu kojoj objekt na slici pripada. Razredi mogu biti generalizirani ili specijalizirani. Na primjer, "pas" može biti jedan razred, ali isto tako može biti i "bulldog", "pekinezer", "bigl" itd. Ovakav postupak zapravo objedinjuje postupak klasifikacije objekata i lokalizacije objekata. Klasifikacija je postupak određivanja razreda, a lokalizacija je postupak određivanja pozicije objekta. Složenost ovog postupka nalazi se u činjenici da je za određivanje razreda kojem objekt pripada potrebna golema količina podataka iz stvarnoga svijeta. Detekcija objekata može se odvijati u stvarnome vremenu (engl. *real-time*) ili naknadnom analizom na slici.

1.3. Cilj

Cilj ovoga rada je proučiti dostupne metode detekcije objekata na slikama koristeći duboko učenje i implementacija jedne metode unutar mobilnog okruženja.

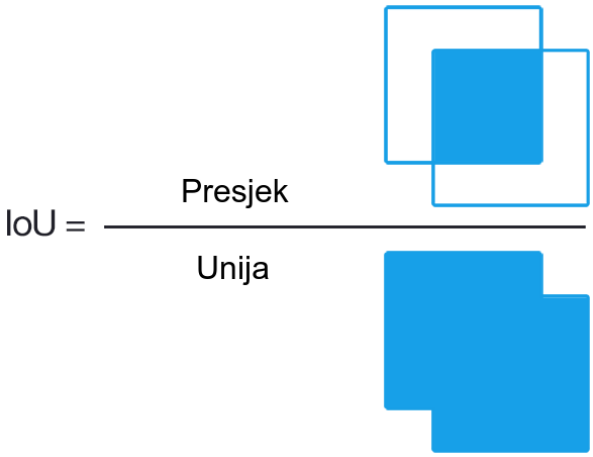
2. mAP vrijednost

Prije opisa postojećih metoda detekcije objekata potrebno je objasniti mAP vrijednost (engl. *Mean Average Precission*), jer se ona često koristi prilikom evaluacije kvalitete treniranog modela.

Ova vrijednost predstavlja srednju vrijednost srednjih preciznosti svih razreda nad svim vrijednostima IoU iz skupa podataka i koristi se prilikom evaluiranja točnosti treniranog modela. Kako bi se mogla objasniti vrijednost i značenje mAPa prilikom validacije nekog modela, prvo se moraju objasniti pojmovi preciznosti (engl. *precission*) i opoziva (engl. *recall*). Preciznost je omjer uspješno klasificiranih objekata i svih klasificiranih objekata na slici. Opoziv je omjer broja uspješno klasificiranih objekata i stvarnog broja objekata traženog razreda na slici. Srednja preciznost (engl. *Average precision*) je srednja vrijednost maksimalne preciznosti za vrijednosti opoziva između 0 i 1 s korakom od 0.1. U nastavku će srednja preciznost biti označena oznakom SP. Formulom $SP = \frac{1}{11} \sum_{n=0}^{10} P(R(n))$ računa se ova vrijednost. R(n) označava n-tu vrijednosti opoziva iz polja [0, 0.1, 0.2, ... , 1.0], a P(R(n)) je vrijednost maksimalne preciznosti za R(n). (Everingham et al., 2010)

Ove vrijednosti pokazuju točnost modela prilikom klasificiranja objekata na slikama. Uz ove vrijednosti potrebno je odrediti i uspješnost modela prilikom lokalizacije, tj. određivanja pozicije na slici gdje se objekt nalazi. Za ovu vrijednost potrebno je izračunati IoU (engl. *Intersect over union*) ili omjer presjeka i unije okvira temeljne istine i generiranog okvira. Okvir temeljne istine je pravokutnik koji omeđuje objekt na slici i može biti zapisan u različitim formatima. Na primjer, može biti u formatu [x_min, y_min, x_max, y_max] ili [x_središte, y_središte, širina, visina]. Okviri temeljne istine unose se ručno za svaku fotografiju u skupu. Vizualni prikaz računanja vrijednosti IoU nalazi se na slici 2.1

Za računanje mAPa može se uzeti jedna vrijednost IoU koja predstavlja granicu. Rezultati detektiranja kojima je IoU manji od ove vrijednosti se odbacuju. Ostalim rezultatima računa se SP. mAP je rezultat izračuna srednje vrijednosti SPa svih razreda. Može se također uzeti skup vrijednosti IoU koje predstavljaju granicu. Skup



Slika 2.1: Omjer presjeka i unije

vrijednosti može biti proizvoljan, ali se na primjer za COCO natjecanja uzima skup vrijednosti između 0.5 i 0.95 uz korak 0.05. [1] Što je vrijednost mAPa veća to je veća sigurnost da će model ispravno detektirati lokaciju i razred objekta.

[1] `http://cocodataset.org/#detection-eval`

3. Postojeće metode detekcije objekata

3.1. R-CNN modeli

3.1.1. R-CNN

R-CNN ili "Region-Based Convolutional Neural Network" smatra se jednim od prvih većih uspjeha u području detekcije objekata pomoću konvolucijskih neuronskih mreža.(Brownlee, 2019)
Ovaj model sastavljen je od 3 modula:

Prijedlog regija: generiranje regija, tj. okvira oko mogućih objekata

Ekstraktor značajki: konvolucijska neuronska mreža koja ekstrahira značajke iz svake predložene regije

Klasifikator: klasificira značajke u neki od poznatih razreda

Model iz ulazne slike generira metodom selektivnog pretraživanja (engl. *Selective search*) regije koje bi mogle biti neki objekt i šalje ih konvolucijskoj neuronskoj mreži da ekstrahira značajke. Za ovaj postupak uzeta je mreža "AlexNet deep CNN", koja šalje vektor značajki klasifikatoru. Ovaj je postupak jednostavan i fleksibilan, ali je veoma spor. Razlog tomu je što se za svaku regiju (može ih biti i do 2000) ponavlja gore navedeni postupak.(Girshick et al., 2013)
Slika 3.1 prikazuje ovaj postupak.

Slika 3.1: Koncept R-CNN modela. Slika preuzeta iz (Girshick et al., 2013)

.

3.1.2. Brzi R-CNN

R-CNN, iako je bio veliki uspjeh, imao je svojih nedostataka. Tako su u radu Girshick (2015) navedeni neki nedostaci R-CNN-a :

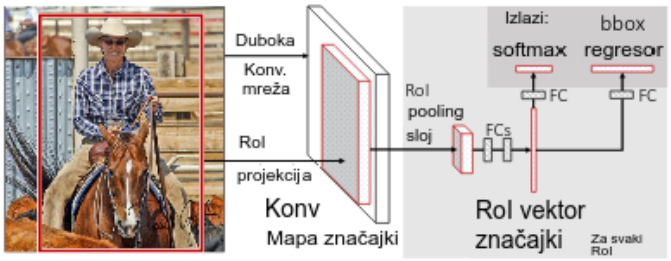
Velike vremenske i prostorne složenosti: treniranje na velikom broju regija

Detekcija objekata je spora: detektiranje objekata je složeno zbog broja regija

Kao nadogradnju na R-CNN bio je predložen SPPnet (engl. *Spatial Pyramid Pooling network*), koji smanjuje računanje generirajući mapu značajki za cijelu sliku.
Pomoću ove mape model može klasificirati svaki prijedlog objekta koristeći vektor značajki ekstrahiran iz same mape. Ovaj model bio je značajno brži od R-CNN-a (između 10 i 100 puta za vrijeme testiranja). Ovaj model je međutim morao spremati značajke u memoriju.

Rješenje ovih problema je Brzi R-CNN sa sljedećim značajkama i prednostima:

1. Veći mAP
2. Treniranje se odvija u jednom prolasku kroz mrežu koristeći MTL (engl. *Multi task loss*)
3. Treniranje ažurira sve slojeve mreže
4. Nema spremanja značajki u memoriju



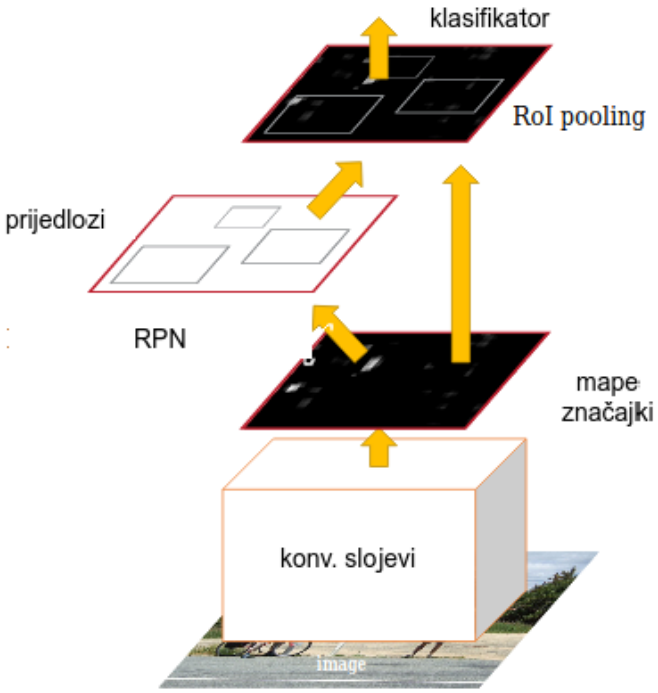
Slika 3.2: Koncept Brzog R-CNN modela. Slika preuzeta iz (Girshick, 2015)

Vjerojatno glavna značajka ovoga modela je regija interesa (engl. *Region of interest*), koja ubrzava detekciju samih objekata na ulaznoj slici. Značajke koje generira ovaj sloj šalju se dalje u potpuno povezane slojeve (engl. *Fully connected layer*) koji se na kraju račvaju u dva izlaza. Jedan izlaz daje vjerojatnost pojave pojedinog objekta u regiji, dok drugi kao rezultat daje dimenzije i koordinate okvira koji obavija sam

objekt. Na slici 3.2 može se vidjeti pojednostavljeni prikaz rada ovog modela. Girshick (2015)

3.1.3. Brži R-CNN

Uspjeh prethodnih modela potaknuo je Girshicka i njegov tim da dalje razvijaju brže i preciznije modele. Tako su razvili Brži R-CNN. Ovaj se model sastoji od dva modula. Prvi modul konvolucijska je neuronska mreža koja na temelju ulazne slike generira predložene regije za pojedine objekte, kao i vrijednost koja određuje pripada li ta regija nekom razredu iz skupa razreda ili pozadini.



Slika 3.3: Koncept Bržeg R-CNN modela. Slika preuzeta iz (Ren et al., 2015)

Taj modul naziva se RPN (engl. *Region Proposal Network*). Važnost ovog modula je što on predlaže drugom modulu gdje da pretražuje. Vrijednost koja označava pripadnost razredu ili pozadini omogućuje drugom modulu da brže klasificira i odredi poziciju objekta. Drugi modul zapravo je Brzi R-CNN. On iz predloženih regija ekstrahira značajke i na temelju njih daje kao izlaz oznaku razreda kojemu objekt pripada kao i dimenzije i poziciju okvira koji omeđuje objekt.
Značajke ovakve arhitekture su znatno smanjena količina generiranih regija i ubrzanje u samoj detekciji. Slika 3.3 grafički prikazuje ova dva modula (Ren et al., 2015)

3.2. YOLO modeli

RCNN modeli imaju svojih prednosti i mana. Kroz godine razvijanja modeli su značajno napredovali po pitanju preciznosti. U današnje je vrijeme međutim sve veća potreba za brzinom pojedinih modela pa RCNN modeli nisu najpraktičnija skupina modela za zadovoljavanje ovih potreba.
YOLO (engl. *You Only Look Once*) modeli takva su vrsta modela. Nisu nužno precizniji od RCNN modela, ali su neusporedivo brži i najčešće se koriste za detekciju u stvarnome vremenu (engl. *Real-time object detection*).

3.2.1. YOLO

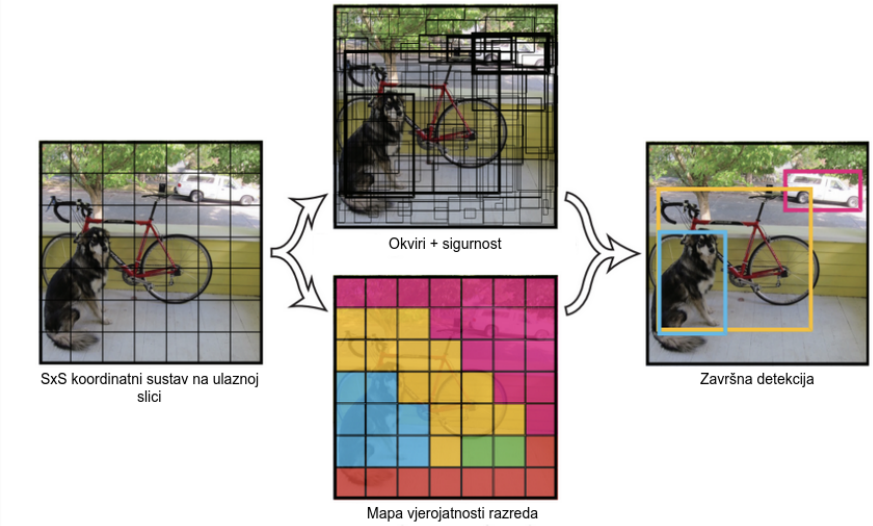
U radu (Redmon et al., 2015) Joseph Redmon prvi je opisao ovu vrstu modela.
U razvoju ovog modela sudjelovao je i sam Ross Girshick. Razlog zašto je ovakav model brži i bolji za uporabu u detektiranju objekata u stvarnome vremenu je korištenje konvolucijske neuronske mreže koja u isto vrijeme pretpostavlja višestruke regije i razrede za iste. Glavna prednost YOLO modela je veoma velika brzina. Objekte na videozapisu može detektirati sa samo 25 milisekundi kašnjenja.
Model radi na sljedeći način (Redmon et al., 2015):

1. Ulaznu sliku dijeli na SxS kvadratnu mrežu, a S je proizvoljno odabran prirodni broj i predstavlja broj kvadrata u retku/stupcu mreže
2. Ako se sredina objekta nađe unutar jednog kvadrata mreže, taj kvadrat je zadužen za detektiranje tog objekta
3. Svaki kvadrat pretpostavlja
   - B okvira oko objekta, a B je proizvoljan prirodni broj okvira čija se težišnica nalazi unutar kvadrata. Svaki od B okvira sadrži strukturu podataka u kojoj su zapisani podaci o x i y koordinatama centra okvira u odnosu na rubove kvadrata. Isto tako su zapisani podaci o širini i visini okvira u odnosu na cijelu sliku i vrijednost IoU između generiranog okvira i okvira temeljne istine za taj objekt
   - C vjerojatnosti pojave razreda iz skupa razreda, a C je broj razreda u skupu. Ove se vjerojatnosti računaju samo ako je središte nekog objekta unutar kvadrata

4. Svaki okvir koji ima IoU manji od proizvoljne granice (na primjer 0.25 ili 0.5) se odbacuje

5. Za preostale okvire se računa postotak koji označava sigurnost modela da se unutar okvira nalazi objekt iz razrednog skupa (na primjer "čovjek"). Računa se kao umnožak IoU i vjerojatnosti pojave određenog razreda unutar kvadrata koji je zadužen za detekciju tog objekta.

Slika 3.4 prikazuje gore navedeni postupak. Kako bi se izbjeglo računanje rezultata u slučaju da je više kvadrata detektiralo isti objekt koristi se non-max potiskivanje. Non-max potiskivanje postupak je u kojem se uklanjaju višestruka pojavljivanja pretpostavljenog okvira za isti objekt.



Slika 3.4: Rad YOLO modela. Najprije je potrebno podijeliti sliku na SxS kvadratnu mrežu. Na ovoj slici S je 7 što znači 49 kvadrata. Zatim svaki od tih 49 kvadrata generira pretpostavljene okvire. U ovom slučaju generiraju se 2 okvira oko svakog kvadrata, što je prikazano na gornjoj sličici u sredini. Radi preglednosti su uklonjeni kvadrati. Na donjoj slici u sredini su vjerojatnosti pojave razreda grupirane po bojama. Izbacuju se okviri s neodgovarajućom vrijednosti IoU. Zatim se množe vrijednost IoU iz preostalih okvira i vjerojatnost pojave razreda iz kvadrata oko kojih ti okviri jesu kako bi se dobili konačni okviri prikazani na posljednoj sličici. Slika preuzeta iz (Redmon et al., 2015)

Ograničenja su vidljiva prilikom detekcije manjih objekata u grupi, jer prilikom pretpostavljanja okvira i određivanja razreda svaki kvadrat na kvadratnoj mreži može imati zapisane informacije o samo dva okvira i jednom razredu. YOLO, za razliku od RCNN-a, u obzir uzima cijelu sliku prilikom generiranja pretpostavki. Time smanjuje

stopu pogrešaka, jer ne obrađuje pozadine koje ne predstavljaju niti jedan razred. Ovaj model neće podbaciti kada se na ulazu nađe novi, dosad neviđeni objekt, jer ima veliku sposobnost generalizacije.(Redmon et al., 2015)

Jedna od glavnih značajki YOLO modela jest povezanost više zasebnih komponenti u jednu neuronsku mrežu. Simultano pretpostavljanje okvira oko objekata i razreda čini ovaj model sofisticiranijim u odnosu na druge modele detekcije objekata.

3.2.2. YOLOv2

Nadogradnja prethodne inačice modela, YOLOv2, uvodi nove značajke koje joj omogućuju još kvalitetniju detekciju objekata.
Neki od novih dodataka su:

Grupna normalizacija (engl. *Batch normalization*): podigla je mAP za 2%

Klasifikator visoke rezolucije: omogućen je unos slika visoke rezolucije, što rezultira povećanjem mAP-a od 4%

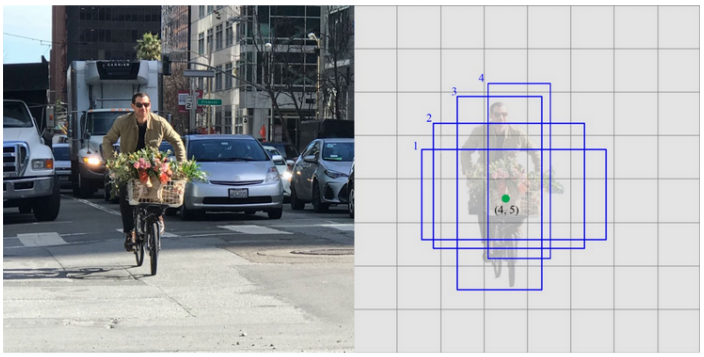
Usidreni okviri (engl. *Anchor Boxes*): omogućuju lakše treniranje mreže

Zanimljivost ovog modela je implementacija nasumične promjene rezolucije ulazne slike svakih 10 grupa. Autori modela su kao najmanju dimenziju odabrali 320x320, a kao najveću 608x608. Sve dimenzije između višekratnici su broja 32.
Ovakav način treniranja omogućuje modelu prilagođavanje na različite rezolucije slika prilikom testiranja.
Također, ovaj model je paralelno bio treniran na dva skupa podataka, COCO i ImageNet te može detektirati 9000 vrsta objekata u stvarnom vremenu. (Redmon i Farhadi, 2016)

3.3. SSD

SSD ili "Single Shot Detector" nastoji dodatno smanjiti kašnjenja i povećati preciznost prilikom detektiranja. Ovaj model koristi VGG16 kako bi ekstrahirao značajke iz ulazne slike. VGG16 predstavlja unaprijed treniranu konvolucijsku neuronsku mrežu koja služi za postavljanje parametara modela. Prvi sloj nakon VGG16 mreže služi za detekciju manjih objekata na slici i dimenzija je 38x38. Ova dimenzija predstavlja broj kvadrata na koje model dijeli sliku. Dimenzije naknadnih slojeva sve su manje (na primjer 8x8 i 4x4) i one služe detektiranju većih objekata na slici. Svaki kvadrat

generira četiri pretpostavke za mogući objekt. Svaka pretpostavka sastoji se od okvira oko objekta i C + 1 vjerojatnosti pojavljivanja razreda unutar okvira, a C je broj razreda u razrednom skupu. Broj 1 predstavlja dodatan razred kada ne postoji objekt unutar generiranog okvira i označen je brojem 0. Razred s najvećom vjerojatnošću uzima se kao pretpostavljeni razred za taj okvir. Na slici 3.5 prikazan je primjer generiranja okvira. Slika je preuzeta s [1]



Slika 3.5: Lijevo na slici nalazi se originalna fotografija. Na desnoj slici vidi se generiranje četiri okvira oko pretpostavljenog objekta za kvadrat koji se nalazi u 5 retku i 4 stupcu 8x8 mreže. Treba napomenuti da bi se ovaj postupak trebao izvodi u sloju koji je dimenzija 38x38, a da je zbog jednostavnijeg prikaza taj sloj na slici prikazan kao kvadratna mreža dimenzija 8x8. U slojevima s manjim dimenzijama (8x8 ili 4x4) generira se šest pretpostavki

Uz generirane okvire, svaki kvadrat generira i unaprijed definirane okvire. Ovi okviri ručno su definirani kako bi mogli pokriti širok spektar objekata iz stvarnog svijeta. U početnim slojevima s većim dimenzijama generiraju se četiri takva okvira, a u slojevima s manjim dimenzijama šest, isto kao što vrijedi za generiranje pretpostavki. Generiranje ovih okvira smanjuje količinu računanja na početku treniranja. Unaprijed definirani okviri koji imaju IoU s okvirom temeljne istine većim od 0.5 označuju se kao pozitivni, dok oni koji imaju IoU manji od te vrijednosti se označuju kao negativni. Lokalizacijski gubitak računa se kao suma smooth L1 vrijednosti odstupanja pretpostavljenih okvira od okvira temeljne istine za sve pozitivno označene unaprijed definirane okvire nad svim razredima. Smooth L1 funkcija definira se kao

$$smoothL1(x) = \left\{ \begin{array}{ll} 0.5x^2, & \text{ako } |x| < 1 \\ |x| - 0.5, & \text{inače} \end{array} \right\}$$

i objašnjenje zašto se koristi nije u sklopu ovog rada. (Liu et al., 2015)

[1] `https://cutt.ly/0yB2Odq`

4. Zahtjevi aplikacije i korištene tehnologije

U ovome poglavlju opisani su zahtjevi koje aplikacija mora zadovoljiti i tehnologije koje su bile korištene u njenoj izradi.

4.1. Zahtjevi

Zahtjevi su podijeljeni na:

1. Korisničke
2. Funkcionalne

Korisnički zahtjevi predstavljaju zahtjeve samoga korisnika, tj. što aplikacija mora omogućiti korisniku. Funkcionalni zahtjevi služe identifikaciji glavnih funkcija sustava i omogućuju kvalitetniju izradu same aplikacije.

4.1.1. Korisnički zahtjevi

Omogućen odabir načina detektiranja objekata

Unutar aplikacije moguće je odabrati između statičkog i dinamičkog detektiranja objekata. Pojam dinamičkog detektiranja odnosi se na detekciju u stvarnom vremenu. Statičko detektiranje mora moći podržavati odabir fotografije iz galerije, ali i snimanje nove fotografije nad kojom se izvodi detektiranje. Za detekciju u stvarnom vremenu otvara se kamera čime počinje postupak detektiranja.

Prikaz rezultata na zaslonu

Aplikacija mora moći prikazati podatke dobivene iz ulazne slike ili videozapisa korisniku. Rezultat mora biti prikazan kao okvir oko detektiranog objekta, te ime razreda

kojem objekt pripada.

4.2. Funckionalni zahtjevi

Učitavanje sadržaja

Aplikacija mora podržavati učitavanje sadržaja iz različitih izvora. Prilikom učitavanja fotografija iz galerije ne postoje dodatni zahtjevi, dok se kod korištenja kamere mora prvo provjeriti postoji li na uređaju kamera.

Učitavanje modela

Model mora biti učitan lokalno. Aplikacija se ne smije spajati s udaljenim poslužiteljem kako bi se izbjegla ovisnost o internetskoj vezi.

4.3. Tehnologije

Za implementaciju modela detekcije objekata korištene su različite tehnologije bez kojih bi postupak bio značajno otežan i skloniji pogreškama.

4.3.1. Tensorflow

Tensorflow je Googleova biblioteka otvorenog koda. Služi kao sučelje za uporabu algoritama strojnog i dubokog učenja, te implementaciju istih. Korisnik se koncentrira na izgled modela i slojeve koji su zastupljeni u samom modelu, dok će Tensorflow brinuti za ostale postupke koji se događaju u pozadini. Širok je spektar proizvoda koji se mogu razvijati pomoću Tensorflowa, kao što su:

- Segmentacija
- Klasifikacija
- Detektiranje poze

Fleksibilna arhitektura Tensorflowa omogućuje korisniku razvoj i pokretanje modela na procesoru, grafičkoj kartici, pa čak i TPU (engl. *Tensor processing unit*). Mobilna inačica Tensorflowa naziva se Tensorflow Lite. Razlog razvoja ove inačice bila je sve veća količina podataka koju su mobilni uređaji mogli pohraniti, ali i sve veća procesorska snaga. Milijuni korisnika diljem svijeta u svakom trenutku koriste svoje mobilne

uređaje za neke određene zadatke, te nije praktično imati model na udaljenom poslužitelju kojemu se pristupa putem mobilnog uređaja. Tensorflow Lite značajan je za daljnji razvoj i implementaciju modela dubokog i strojnog učenja, jer eliminira potrebu za spajanjem na poslužitelj i ubrzava postupak dobivanja rezultata od modela.(Abadi et al., 2015)

4.3.2. Flutter

Razvoj mobilnih aplikacija u današnje je doba od velike važnosti zbog sve veće globalne potražnje. Dvije mobilne platforme koje imaju najveći udio na tržištu su Appleov IOS i Googleov Android. Vještine u izradi aplikacija i sustava za ove platforme izrazito su tražene. Problem je međutim naći programere koji istodobno rade na obje platforme, budući da se za razvoj aplikacija koriste različiti jezici i radni okviri. Radni okvir Flutter nastoji riješiti ovu poteškoću tako što nudi mogućnost prevođenja koda za IOS i Android, ali sve iz iste baze koda. Nastao je u Googleu 2017. godine, kada je mogao jedino prevoditi kod za Android operativni sustav. Programski jezik koji se koristi za razvoj unutar ovog okvira naziva se Dart. Ovaj jezik također je razvijen u Googleu. Glavna komponenta u Flutteru su "Widgeti". Služe za prikaz elemenata na ekranu. Dva skupa ovih komponenti koje postoje unutar radnog okvira su Material Design (Android) i Cupertino (IOS). U najnovijoj inačici Fluttera postoji mogućnost pokretanja aplikacija kao web aplikacije. Ova mogućnost još uvijek je u beta razvoju.

5. Implementacija i rezultati

U ovom poglavlju objašnjava se arhitektura aplikacije u kojoj je implementiran model detekcije objekata. Model detekcije objekata pomoću konvolucijskih neuronskih mreža bit će implementiran za operacijski sustav Android. Koristi se model u Tensorflow lite obliku, koji je izveden iz Tensorflow modela. Aplikacija je pisana u radnom okviru Flutter koristeći programski jezik Dart.
Postupak treniranja i implementacije modela značajno je pojednostavljen činjenicom da Google u ponudi ima širok spektar različitih modela koji su istrenirani do određene točke. Modeli se dalje mogu modificirati i istrenirati po potrebi čime se štede računalni i vremenski resursi.

5.1. Izgled skupa podataka i pohrana

Za početak je bilo potrebno odrediti razrede koje će model moći prepoznati i pronaći odgovarajući skup fotografija koje odgovaraju razredima. Za potrebe ove implementacije odabrane su različite pasmine pasa i mačaka. Preuzeti skup slika sadrži oko 7400 slika pasa i mačaka. Unutar skupa fotografija nalazi se 37 razreda od kojih je podjednak broj razreda za pse i mačke. Svaka slika unutar skupa ima odgovarajuću anotaciju koja daje informaciju o koordinatama okvira koji omeđuje životinju na slici.

Nakon što je skup slika preuzet, potrebno ga je pretvoriti u odgovarajući format (TFRecord) koji je prikladan za daljnju analizu. To je bilo moguće ostvariti pomoću skripti koje se mogu preuzeti sa Googleovog repozitorija na githubu. Generiranje TFRecorda daje kao rezultat dva skupa datoteka. Jedan koji se koristi za treniranje, a drugi za validaciju. Kako bi model mogao točno klasificirati objekte stvorena je datoteka s labelama. Slika 5.1 prikazuje primjer izgleda ove datoteke.

Kao pomoć prilikom pohranjivanja datoteka iskorištena je Googleova usluga u oblaku, koja omogućuje pohranjivanje i izvođenje različitih zadataka bez korištenja vlastitih resursa. Kako bi sve funkcioniralo, potrebno je najprije postaviti novi "bucket" u kojem se mogu pohranjivati sve datoteke i rezultati treniranja i validacije.

Za postavljanje navedenog direktorija korišteno je gsutil sučelje kroz komandnu liniju. Primjer naredbe gsutil sučelja vidljiv je na slici 5.2. Korištenje ovog sučelja jednostavno je te omogućuje brzo postavljanje novog projekta i prostora za pohranu. Za potrebe ovog rada korištena je probna inačica. Unutar probne inačice nalaze se sve potrebne komponente za kvalitetno provođenje treniranja, validacije i pohrane podataka.

```
item {
  id: 1
  name: 'Abyssinian'
}

item {
  id: 2
  name: 'american_bulldog'
}

item {
  id: 3
  name: 'american_pit_bull_terrier'
}

item {
  id: 4
  name: 'basset_hound'
}
```

Slika 5.1: Odsječak iz datoteke s labelama korištenim prilikom treniranja

```
gsutil mb gs://objectdetectiontomislavbucket
```

Slika 5.2: Postavljanje novog repozitorija unutar projekta pomoću komandne linije

5.2. Sučelja i alati

Nakon konfiguracije projekta na oblaku potrebno je instalirati Tensorflow Object Detection API koji u sebi sadrži brojne skripte i konfiguracijske datoteke korištene prilikom postavljanja okruženja za treniranje i validaciju. Za lakšu instalaciju paketa potrebnih za rad sučelja preuzet je upravitelj paketima anaconda. Anaconda je veoma praktičan alat koji se koristi u računarskoj znanosti i služi za jednostavnije upravljanje i implementaciju paketa. Nakon instalacije alata kreirano je novo virtualno okruženje u kojem su bili naknadno preuzeti paketi. Tensorflow Object Detection API preuzet je sa službenog github repozitorija i konfiguriran pomoću skripte. Uz ovo sučelje instalirano je i sučelje COCO. COCO je veliki skup fotografija koji se koristi za razne

zadatke poput detekcije objekata, segmentacije i drugih.
Veoma je bitno na početku ispravno konfigurirati sve potrebne direktorije i sučelja. U protivnom kasnije može doći do neželjenih poteškoća prilikom implementacije.

5.3. SSD Mobilenet

Model koji se koristi za implementaciju metode detekcije objekata je SSD MobileNet. Razlog odabira ovog modela mogućnost je treniranja pomoću tranzitivnog učenja na Googleovom oblaku, što je detaljnije pojašnjeno u nastavku. SSD MobileNet inačica je SSD-a prilagođena za izvođenje na mobilnim uređajima.

Ograničenost vlastitih računalnih i vremenskih resursa zahtijeva preuzimanje unaprijed treniranog modela koji ima postavljene parametre i arhitekturu te treniranje modela korištenjem tehnike tranzitivnog učenja. Tranzitivno učenje tehnika je u kojem se treniranje ne započinje od početka, nego od određene kontrolne točke na unaprijed treniranom modelu. Kontrolna točka je datoteka u kojoj se nalaze informacije o definiranim parametrima prilikom treniranja modela. Na službenom github repozitoriju[1] postoje mnogi različiti modeli koji se mogu koristiti za tranzitivno učenje. Ovaj model [2] izabran je zbog kompatibilnosti izvođenja na TPU, što omogućuje brže treniranje. Nakon preuzimanja modela kontrolne točke bile su kopirane u isti direktorij gdje se nalaze i datoteke koje služe za treniranje.

Kako se koristi unaprijed trenirani model s konfiguriranom arhitekturom i parametrima potrebno je, osim postavljanja kontrolnih točaka, postaviti konfiguracijsku datoteku u kojoj se zapisuju podaci bitni za daljnje treniranje i validaciju istreniranog modela. U konfiguracijskoj datoteci postavljaju se vrijednosti poput broja razreda, broja kontrolnih primjeraka i funkcije za računanje stope učenja. Odsječak iz ove datoteke vidljiv je na slici 5.3.

5.4. Treniranje

Treniranje ovog modela zahtijeva prolazak kroz sve primjere za treniranje i uspoređivanje pretpostavljenih okvira sa stvarnim okvirima oko objekata. Budući da se treniranje

[1] `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md`

[2] `http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v1_0.75_depth_quantized_300x300_coco14_sync_2018_07_18.tar.gz`

```
train_config: {
  fine_tune_checkpoint: "gs://objectdetectiontomislavbucket/data/model.ckpt"
  fine_tune_checkpoint_type: "detection"
  load_all_detection_checkpoint_vars: true
  batch_size: 128
  sync_replicas: true
  startup_delay_steps: 0
  replicas_to_aggregate: 8
  num_steps: 60000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    ssd_random_crop {
    }
  }
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        cosine_decay_learning_rate {
          learning_rate_base: 0.2
          total_steps: 60000
          warmup_steps: 6000
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "gs://objectdetectiontomislavbucket/data/pet_faces_train.record*"
  }
```

Slika 5.3: Odsječak iz konfiguracijske datoteke u kojem se postavlja konfiguracija za treniranje

odvija na oblaku nije potrebno koristiti vlastite računalne resurse, što znatno olakšava razvoj. Gsutil sučelje putem komandne linije čini jednostavnim proces stvaranja novog postupka treniranja nad modelom. Potrebno je međutim prethodno konfigurirati sve datoteke kako bi cijeli postupak bio što učinkovitiji. Za potrebe pokretanja novog postupka treniranja napisana je vlastita pomoćna skripta koja je dostupna u repozitoriju ovoga rada. Skripta služi kako bi se jednostavnije mogle postaviti putanje do svih potrebnih datoteka i direktorija. Unutar skripte također su navedene potrebne zastavice, nužne kako bi se treniranje izvršilo.

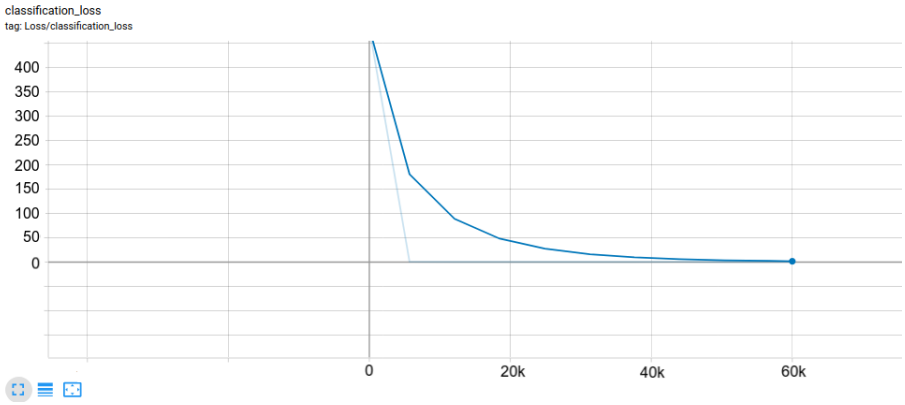
```
#!/usr/bin/bash
...
conda activate tensorflow_obj_detection
cd ~/Tomo/Faks/Zavrsni-rad/model/models/research

JOB= ...
gcloud ai-platform jobs submit training $JOB \
--job-dir=$JOB_DIR  \
--packages $PACKAGES --module-name $MODULE_NAME_TPU \
--runtime-version $RUNTIME_VERSION \
--scale-tier $SCALE_TIER_TPU \
```

```
$REGION $BLANK $TPU_ZONE \
--model_dir=$MODEL_DIR \
--pipeline_config_path=$PIPELINE_CONFIG_PATH
```

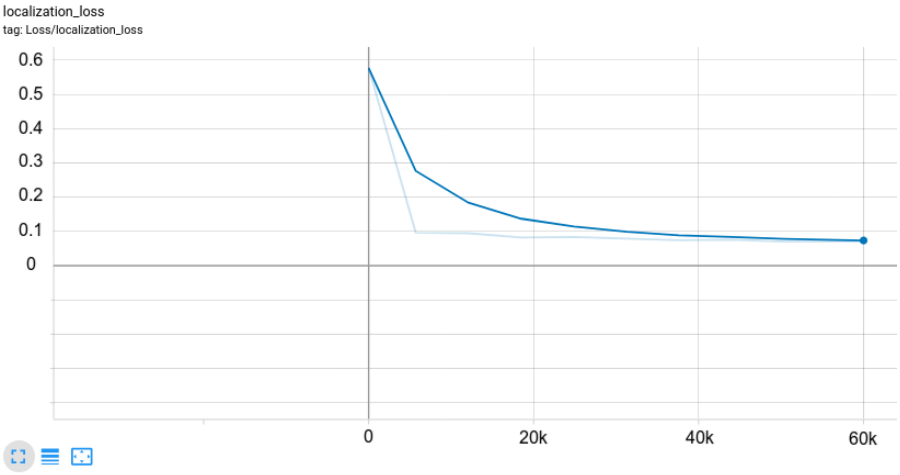
Kako bi sve moglo raditi, potrebno se prvo pozicionirati u "research" direktorij unutar Tensorflow Object Detection API direktorija koji je prethodno preuzet i konfiguriran, jer se u njemu nalaze potrebne skripte i paketi nužni za rad. Za pokretanje postupka treniranja potrebno je postaviti direktorij u koji se spremaju rezultati treniranja.
Isto tako, treniranje se ne bi moglo izvršiti bez određivanja paketa koji se koriste, kao niti inačice tensorflowa koja se koristi prilikom izvođenja računskih operacija. Prilikom treniranja ovog modela korištena je inačica 1.15.

Za vrijeme treniranja klasifikacijski gubitak na slici 5.4 relativno se brzo smanjio, najvećim djelom zbog toga što se koristi unaprijed trenirani model. Nakon 20 tisuća koraka nema značajnijih promjena, tako da se može smanjiti broj koraka treniranja čime bi se uštedjeli računalni i vremenski resursi. Klasifikacijska pogreška bi međutim bila veća. Ovakvi kompromisi nužni su kako bi se moglo doći do što kvalitetnijeg proizvoda s najmanjim mogućim ulogom.



Slika 5.4: Pogreška prilikom klasifikacije tokom treniranja. X os predstavlja broj koraka ili iteracija. Y os predstavlja broj primjera iz skupa fotografija za koje je određivanje razreda bilo pogrešno.

Također se mora pratiti lokalizacijska pogreška jer kvalitetna detekcija objekata zahtijeva točnost kako klasifikacije tako i lokalizacije objekata. Lokalizacija objekata prilikom treniranja ima očekivan pad pogreške, isto kao što ima i klasifikacija. Kod lokalizacije, također nakon 20 tisuća koraka, nije bilo značajnijeg pada postotka pogreške. Postupak treniranja unutar Googleovog oblaka trajao je nešto manje od dva sata, prvenstveno zahvaljujući izvođenju treniranja na unaprijed treniranom modelu, ali i korištenju TPU. Promjena lokalizacijskog gubitka vidljiva je na slici 5.5



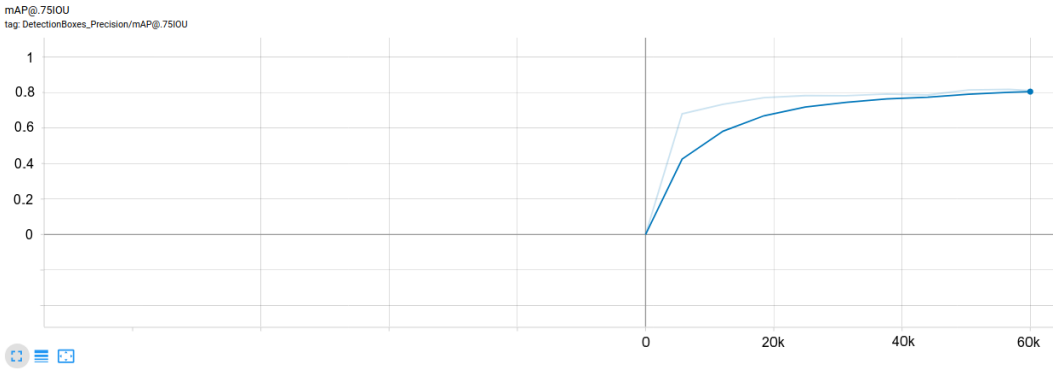
Slika 5.5: Lokalizacijski gubitak tokom treniranja. X os predstavlja broj koraka ili iteracija. Y os predstavlja vrijednost lokalizacijskog gubitka, koji se računa kao suma smooth L1 vrijednosti odstupanja generiranih okvira od okvira temeljne istine za sve razrede

5.5. Validacija

Kako bi se mogla utvrditi točnost istreniranog modela, potrebno je izvršiti validaciju nad kontrolnim primjercima. Kao i kod treniranja, za validaciju se koristi Googleov oblak, ali se računanje izvodi na GPU jer trenutno ne postoji mogućnost validacije na TPU. Kod koji pokreće potreban validacijski postupak na oblaku nalazi se u istoj skripti koja se koristi za konfiguraciju putanja do direktorija za treniranje.

```
#!/usr/bin/bash
...
JOB_EVAL= ...
gcloud ai-platform jobs submit training $JOB_EVAL \
--job-dir=$JOB_DIR --packages $PACKAGES \
--module-name $MODULE_NAME_GPU \
--runtime-version $RUNTIME_VERSION \
--scale-tier $SCALE_TIER_GPU $REGION $BLANK \
--model_dir=$MODEL_DIR \
--pipeline_config_path=$PIPELINE_CONFIG_PATH \
--checkpoint_dir=$CHECKPOINT_DIR
```

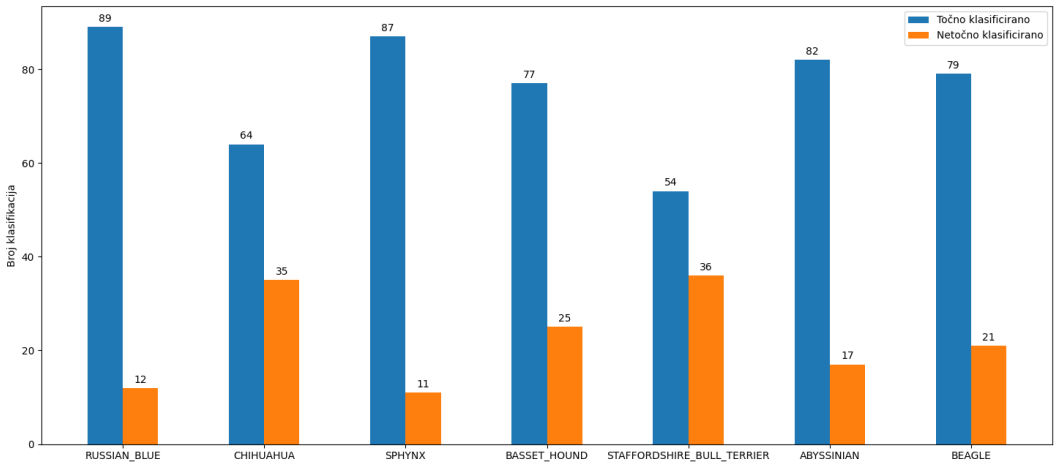
Za validaciju nije potrebno promijeniti sve zastavice, već samo one koje određuju da se koristi GPU, a ne TPU. Validacija se izvršava paralelno s treniranjem i postupak također traje dva sata. Prilikom validacije treniranog modela računa se mAP vrijednost za sve razrede iz skupa podataka i IoU koji su veći od 75%.



Slika 5.6: X os predstavlja broj koraka ili iteracija, a Y os predstavlja mAP vrijednost u pojedinom koraku mAP za vrijednosti IoU veće od 0.75

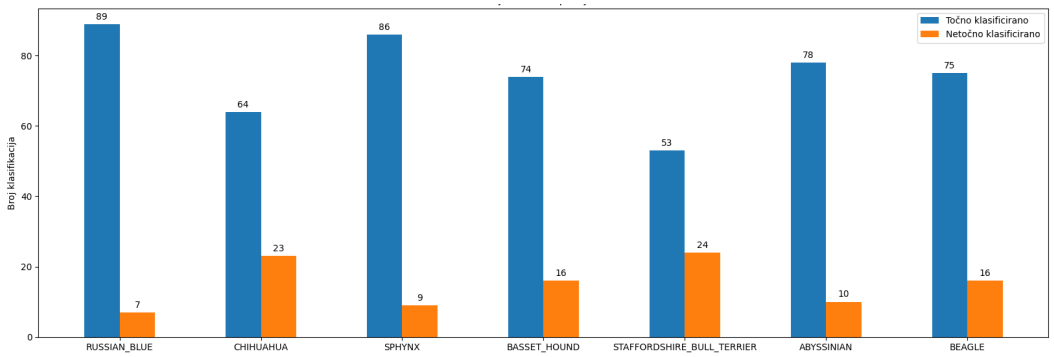
Iz grafa na slici 5.6 vidljivo je da je preciznost bila to veća što je model bio istreniraniji, što je u skladu s očekivanjima. Isto tako je potrebno uočiti kako se vrijednost mAPa nakon 40 tisuća koraka nije značajno promijenila i kreće stagnirati. Broj iteracija može se smanjiti kako bi se uštedjelo na računalnim resursima.

Uz validaciju koja je davala rezultate tokom treniranja, potrebno je predstaviti rezutate na primjerima za testiranje. Primjeri su uzeti iz istog skupa fotografija, ali ove fotografije model do sada nije analizirao. Testiranje se provodi automatski uz pomoć samostalno napisanih skripti koje su dostupne na guthub repozitoriju [3]. Skripte su napisane u Pythonu.



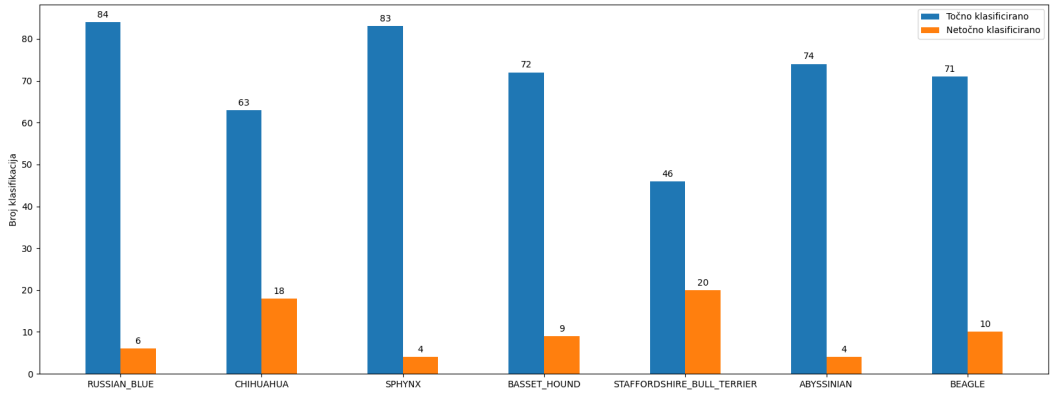
Slika 5.7: Prikaz rezultata klasificiranja nad 100 testnih primjera za postotak sigurnosti od barem 25%.

[3] `https://github.com/tkurtovic98/tflite_model_testing`



Slika 5.8: Prikaz rezultata klasificiranja nad 100 testnih primjera za postotak sigurnosti od barem 50%.

Slike su u novom direktoriju organizirane u direktorije koji su nazvani po imenu razreda. Za potrebe ovog rada, uzeta su tri nasumična razreda koji predstavljaju psa i tri nasumična razreda koji predstavljaju mačku. Nakon premještanja fotografija potrebno je pokrenuti skriptu koja će izvršiti testiranje klasifikacije. Postupak testiranja uspješnosti klasifikacije izvršavao se nad oko 100 fotografija za svaki razred. Da bi pretpostavka došla u obzir za određivanje uspješnosti klasifikacije morala je imati vrijednost postotka sigurnosti veću od 25, 50 ili 75 posto. Grafovi za ove vrijednosti prikazani su na slikama 5.7, 5.8 i 5.9, respektivno.



Slika 5.9: Prikaz rezultata klasificiranja nad 100 testnih primjera za postotak sigurnosti od barem 75%.

5.6. Mobilna aplikacija

Nakon što završi treniranje i validacija modela, potrebno je pretvoriti model u format prikladan za mobilne uređaje, drugim riječima u tensorflow lite format.

Kako bi se to ostvarilo potrebno je zamrznuti parametre trenutnog modela čime se dobiva zamrznuti graf (engl. *frozen graph*). Kao i do sada, pretvorba se odvija pomoću skripte u kojoj se navode potrebne putanje i poziva metoda tensorflow sučelja. Ovaj postupak odvija se unutar virtualnog okruženja, stvorenog u anacondi na samome početku, jer su u njemu konfigurirani potrebni paketi. Nakon pretvorbe u željeni format (.tflite umjesto .pb) potrebno je stvoriti novi flutter projekt. Kao razvojno okruženje koristi se Visual Studio Code, unutar kojega se instaliraju potrebne ekstenzije za Dart i Flutter.

Unutar samog projekta potrebno je dodati ovisnost (engl. *dependency*) o tflite paketu. Također je unutar assets direktorija potrebno dodati datoteku s labelama i sam model. Prvo je potrebno postaviti izgled zaslona prema korisničkim zahtjevima. Slika 5.10 prikazuje odsječak koda koji stvara glavnu stranicu.

```
Widget _appBar() => AppBar(
      title: Text('Object detector'),
      centerTitle: true,
    ); // AppBar

Widget _bottomNavigationBar() => BottomNavigationBar(
      items: const <BottomNavigationBarItem>[
        BottomNavigationBarItem(
            icon: Icon(Icons.camera_alt), title: Text('Static')), // BottomNavigationBarItem
        BottomNavigationBarItem(
            icon: Icon(Icons.videocam), title: Text('Dynamic')), // BottomNavigationBarItem
      ], // <BottomNavigationBarItem>[]
      currentIndex: _selectedIndex,
      selectedItemColor: Colors.tealAccent,
      onTap: _onItemTapped,
    ); // BottomNavigationBar
}
```

Slika 5.10: Odsječak koda za glavnu stranicu

Aplikacija je podijeljenja u dva područja: prvo služi za detekciju objekata na statičkim slikama, dok drugo služi za detekciju objekata unutar videozapisa. Kod za ta dva područja nalazi se unutar repozitorija na githubu [4].
Detektiranje objekata na statičkim slikama vrlo je precizno. U stvarnom vremenu vidljivo je nastupanje kašnjenja i preciznost ovisi o tome kako je objekt, u ovome slučaju pas ili mačka, postavljen tijekom snimanja (Osvjetljenje, kut snimanja itd.). U nastavku se mogu vidjeti primjeri statičkoga i dinamičkoga detektiranja. Na slici 5.11 vidljiv je odsječak koda koji unutar statičkog okruženja prikazuje rezultat na slici.

[4] `https://github.com/tkurtovic98/object-detector`

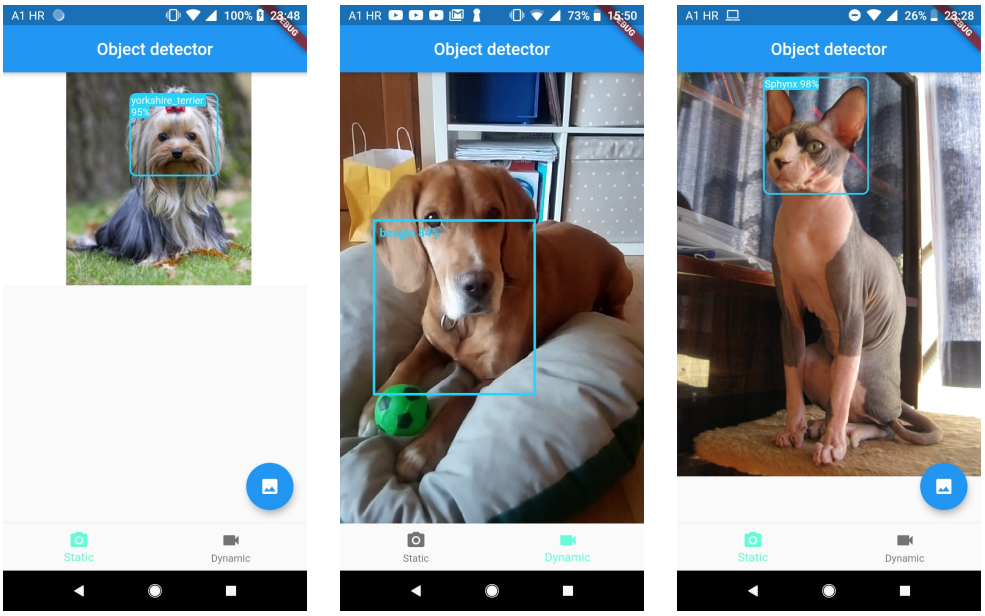
```
List<Widget> renderBoxes(Size screen) {
  if (_recognitions == null) return [];
  if (_imageHeight == null || _imageWidth == null) return [];

  double factorX = screen.width;
  double factorY = _imageHeight / _imageWidth * screen.width;
  Color blue = Color.fromRGBO(37, 213, 253, 1.0);
  return _recognitions.map((re) {
    return Positioned(
      left: re["rect"]["x"] * factorX,
      top: re["rect"]["y"] * factorY,
      width: re["rect"]["w"] * factorX,
      height: re["rect"]["h"] * factorY,
      child: Container(
        decoration: BoxDecoration(
          borderRadius: BorderRadius.all(Radius.circular(8.0)),
          border: Border.all(
            color: blue,
            width: 2,
          ), // Border.all
        ), // BoxDecoration
        child: Text(
          "${re["detectedClass"]} ${(re["confidenceInClass"] * 100).toStringAsFixed(0)}%",
          style: TextStyle(
            background: Paint()..color = blue,
            color: Colors.white,
            fontSize: 12.0,
          ), // TextStyle
        ), // Text
      ), // Container
    ); // Positioned
```

Slika 5.11: Kod koji za statičku detekciju ispisuje podatke na zaslon i crta okvir oko objekta

5.7. Testiranje aplikacije

U završnom djelu razvoja potrebno je testirati zadovoljava li razvijena aplikacija zahtjeve koji su izneseni na početku ovog rada. Aplikacija nudi odabir detektiranja na statičkim slikama ili videozapisu u stvarnom vremenu. Model se dohvaća lokalno i nema potrebe za internetskom vezom. Na slici 5.12 vidljive su snimke zaslona prilikom korištenja aplikacije. Prva i treća snimka zaslona prikazuje aplikaciju prilikom detektiranja statičkih fotografija preuzetih s interneta, dok druga prikazuje rad aplikacije pri detektiranju u stvarnom vremenu snimljeno vlastitim uređajem.



(a) Primjer detektiranja na statičkoj fotografiji **(b)** Primjer detektiranja u stvarnom vremenu **(c)** Primjer detektiranja na statičkoj fotografiji

Slika 5.12: Detekcija različitih kategorija pasa i mačaka

6. Zaključak

U ovome radu glavna ideja bila je objasniti i implementirati model detekcije objekata. Detektiranje objekata u stvarnome vremenu veoma je složen postupak koji zahtjeva mnogo računalne snage. Međutim, zadnjih godina istraživači su došli do modela koji brzo i precizno daju rezultate. Početak razvoja modela za detektiranje objekata može biti pripisan prvom RCNN-u, jer je on bio temelj za daljnje napretke u ovome području.
Premda se promatranjem i čitanjem tuđih radova može jako puno naučiti, za praktično znanje i razumijevanje bilo je potrebno pronaći neki model te ga implementirati unutar mobilne platforme. Budući da je SSD jedan od modela s najvećom kvalitetom, on je poslužio kako bi se na njemu izvelo treniranje. Korištenje unaprijed treniranog modela preporučena je metoda razvoja, jer znatno smanjuje vrijeme potrebno da bi se razvila aplikacija. Uz pomoć brojnih alata i sučelja te koristeći Googleovu platformu na oblaku, treniranje i validacija navedenog modela nisu bili toliko zahtjevni koliko je bila zahtjevna sama konfiguracija svih potrebnih datoteka.

Nakon implementacije navedenog modela uz pomoć Flutter radnog okvira, bilo je moguće testirati samu aplikaciju na stvarnim, još neviđenim fotografijama. Razvijanje uz pomoć Flutter radnog okvira bio je dobar odabir, jer se uz manje preinake izgled aplikacije može prilagoditi korisnicima IOS mobilne platforme. Ovakva vrsta aplikacije može imati široku primjenu, jer je postupak treniranja i implementiranja jednak za bilo koju vrstu objekata koju želimo moći detektirati. Daljnji razvoj ove aplikacije može, na primjer, uključiti detektiranje većeg broja pasa i mačaka, kako bi se mogla stvoriti baza podataka svih pasa i mačaka te njihovih vlasnika. Time bi se, u slučaju da se kućni ljubimac izgubi, omogučilo lakše pronalaženje kako vlasnika tako i informacija o izgubljenoj životinji.

LITERATURA

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, i Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Jason Brownlee. A Gentle Introduction to Object Recognition With Deep Learning. `https://machinelearningmastery.com/object-recognition-with-deep-learning/`, 2019. [Online; accessed 2-April-2020].

Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, i Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. URL `http://arxiv.org/abs/1504.08083`.

Ross B. Girshick, Jeff Donahue andTrevor Darrell, i Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013. URL `http://arxiv.org/abs/1311.2524`.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, i Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015. URL `http://arxiv.org/abs/1512.02325`.

Joseph Redmon i Ali Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.

Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, i Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. URL http://arxiv.org/abs/1506.02640.

Shaoqing Ren, Kaiming He, Ross B. Girshick, i Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. URL http://arxiv.org/abs/1506.01497.

R. Szeliski. *Computer Vision: Algorithms and Applications*. Texts in Computer Science. Springer London, 2010. ISBN 9781848829350. URL https://books.google.hr/books?id=bXzAlkODwa8C.

Detekcija objekata na slikama

Sažetak

Ovaj rad predstavlja neke arhitekture modela koji se koriste prilikom detektiranja objekata. Razvoj ove domene u zadnjih par godina bio je značajan, što potvrđuje složenost i preciznost dostupnih modela. RCNN-a bio je prvi značajan model u ovom području. Modeli koji su kasnije razvijeni su YOLO i SSD modeli i oni su pomaknuli granice i u brzini i u preciznosti detektiranja. Uključenost tvrtke Google u razvoj i distribuciju unaprijed treniranih modela također čini postupak treniranja jednostavnijim za korisnika, eliminirajući brigu o konfiguraciji arhitekture. Nakon treniranja modela slijedi implementacijski postupak u kojem se objašnjavaju svi važni koraci koji su omogućili pokretanje modela unutar mobilne platforme.

Ključne riječi: Android, Tensorflow, Flutter, SSD MobileNet

Object detection

Abstract

This paper presents some model architectures used in the field of object detection. The development of this domain in the last couple of years has been significant, which the complexity and precision of the available models confirm. RCNN was the first significant model in this area. After it other models such as the YOLO and SSD model were developed and have shifted boundaries both in speed and in detection accuracy. Also, Google's involvement in the development and distribution of pretrained models makes the training process easier for the user, eliminating the concern about architecture configuration. After training the model follows the implementation process in which all the important steps that enabled the launch of the model within the mobile platform are explained.

Keywords: Android, Tensorflow, Flutter, SSD MobileNet