SADRŽAJ

1. UVOD

Od pojave prvog automobila 1886. godine, kompanije koje proizvode automobile streme da ih načine što bezbednijim, ali i mehanički i tehnološki naprednijim, kako bi se njihov proizvod istakao na tržištu i što bolje prodavao. Jedan od najznačajnijih pomaka u sigurnosti vozača u saobraćaju učinila je kompanija Volvo, zaslužne za izum bezbednosnog pojasa koji je postao deo obavezne opreme u automobilima širom sveta i spasio mnogobrojne živote.

U današnjim kolima su prisutni mnogobrojni sistemi za aktivnu i pasivnu pomoć vozaču. Većina automobila na putevima je opremljeno sistemima kao što su ABS(Anti-lock Braking system), ESC(Electronic Stability Control) i mnogi drugi. Svi elektronski sistemi za asistenciju vozača spadaju u grupu sistema zvanu ADAS(Advanced Driver Assistence Systems) i njihov cilj je pružanje podrške vozaču u vidu informacija o spoljašnjem okruženju, asistenciji prilikom izvođenja određenog manevra, pa sve do potpune autonomije u vožnji. Ovi sistemi se baziraju na najmodernijim tehnologijama i mogu se naći u velikom broju novih automobila kao deo nekog od viših paketa opreme.

Jedan od sistema iz ADAS grupe je sistem za detekciju saobraćajnih znakova (TSR – Traffic Sign Recognition), čija je svrha detekcija i prepoznavanje saobraćajnih znakova na koje automobil nailazi, kako bi vozaču prikazali ono što se na tim znakovima nalazilo i time vozača drži obaveštenim o tome šta ga očekuje na putu, kako bi mogao da se pravovremeno pripremi na predstojeću situaciju ili reguliše brzinu vozila prema brzini naznačenoj znakom ograničenja.

U ovom master radu biće predstavljen proces kreiranja modela mašinskog učenja za potrebe detekcije saobraćajnih znakova i rezultat implementacije istog u aplikaciju, sa ciljem da se korisnicima iste omogući korišćenje sistema za detekciju saobraćajnih znakova bez potrebe za posedovanjem jednog takvog sistema kao deo prateće opreme njihovog automobila.

U nastavku ovog poglavlja biće definisan cilj rada i opisani razlozi za izradu istog. Takođe, biće opisani predmet rada gde su opisane tehnologije korišćene u izradi rada, metode i struktura rada, gde će biti nabtojana poglavlja rada sa opisom svakog od njih.

1.1. CILJ RADA

Predmet izučavanja master rada je detekcija znakova u realnom vremenu primenom modela mašinskog učenja.

Upravljanje motornim vozilom je kompleksan zadatak koji od vozača zahteva da ostane fokusiran na put i dešavanja na njemu dug vremenski period. U slučaju da vozač ne primeti i pravovremeno odreaguje na znak postavljen kraj kolovoza, može dovesti u opasnost sebe i druge učesnike u saobraćaju. Korišćenjem sistema za detekciju saobraćajnih znakova, vozač ima mogućnost da sadržaj nekog znaka koji nije pravovremeno video pogleda kasnije, kao i da bude obavešten o prepreci na koju nailazi.

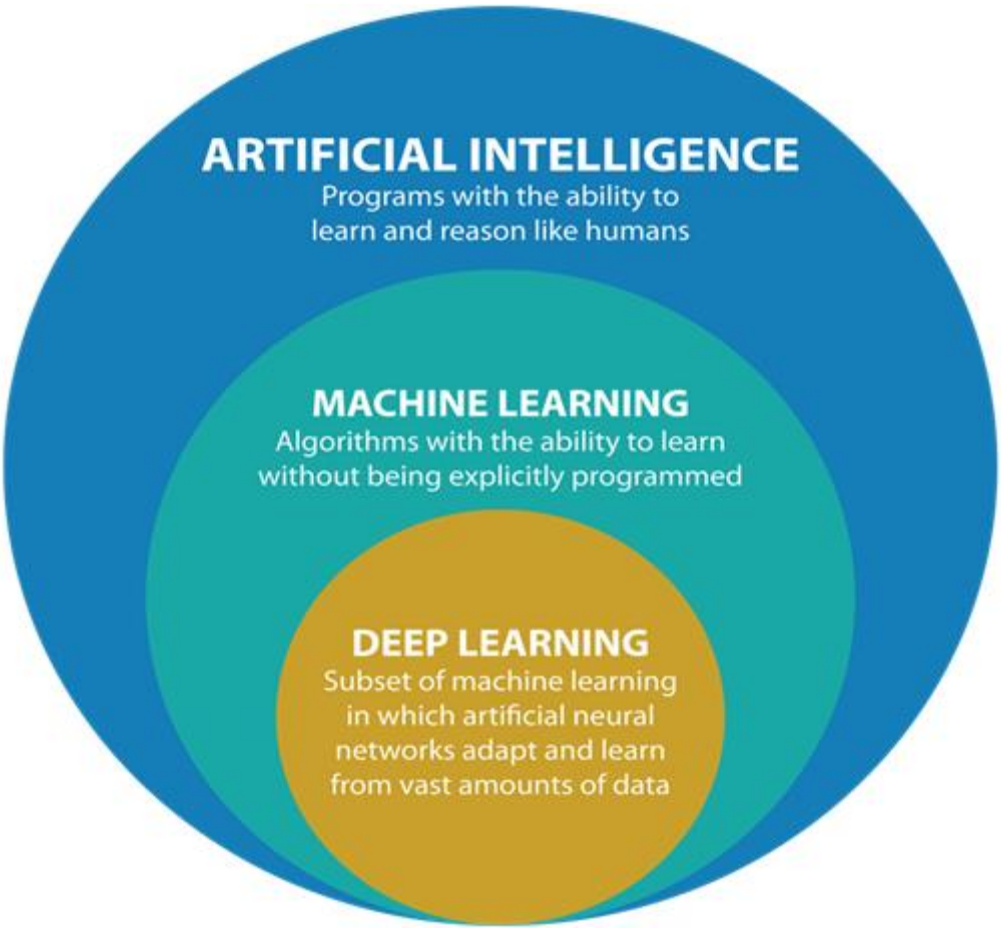
1.2. PREDMET RADA

Predmet istraživanja master rada je razvoj i implementacija sistema za detekciju saobraćajnih znakova na putu zasnovanog na konvolutivnoj neuronskoj mreži i njegova dalja implementacija u android mobilnu aplikaciju. Biće opisane metode i tehnike za najpre prikupljanje trening podataka, zatim predstavljeno treniranje modela pomoću tih podataka, a na kraju i izrada android aplikacije koja taj model koristi.

2. MAŠINSKO UČENJE

Mašinsko učenje se definiše kao izučavanje kompjuterskih programa koji koriste algoritme i statističke modele kako bi učili kroz inferencu i obrazce bez eksplicitnog programiranja za rešavanje tog zadatka. U toku protekle decenije, ovo polje se značajno razvijalo. [11]

Mašinsko učenje spada u disciplinu u okviru veštačke inteligencije (slika 1) i između te dve discipline postoje bitne razlike, od kojih je najveća to da veštačka inteligencija teži oponašanju ljudske inteligencije na jako širokom spektru problema i samo jedan njen deo je mašinsko učenje, dok se mašinsko učenje fokusira na rešavanje specijalizovanih problema učenjem iz trening podataka, koji su često usko vezani za određenu oblast (konkretno u ovom radu, vezani su za detekciju i klasifikaciju saobraćajnih znakova).



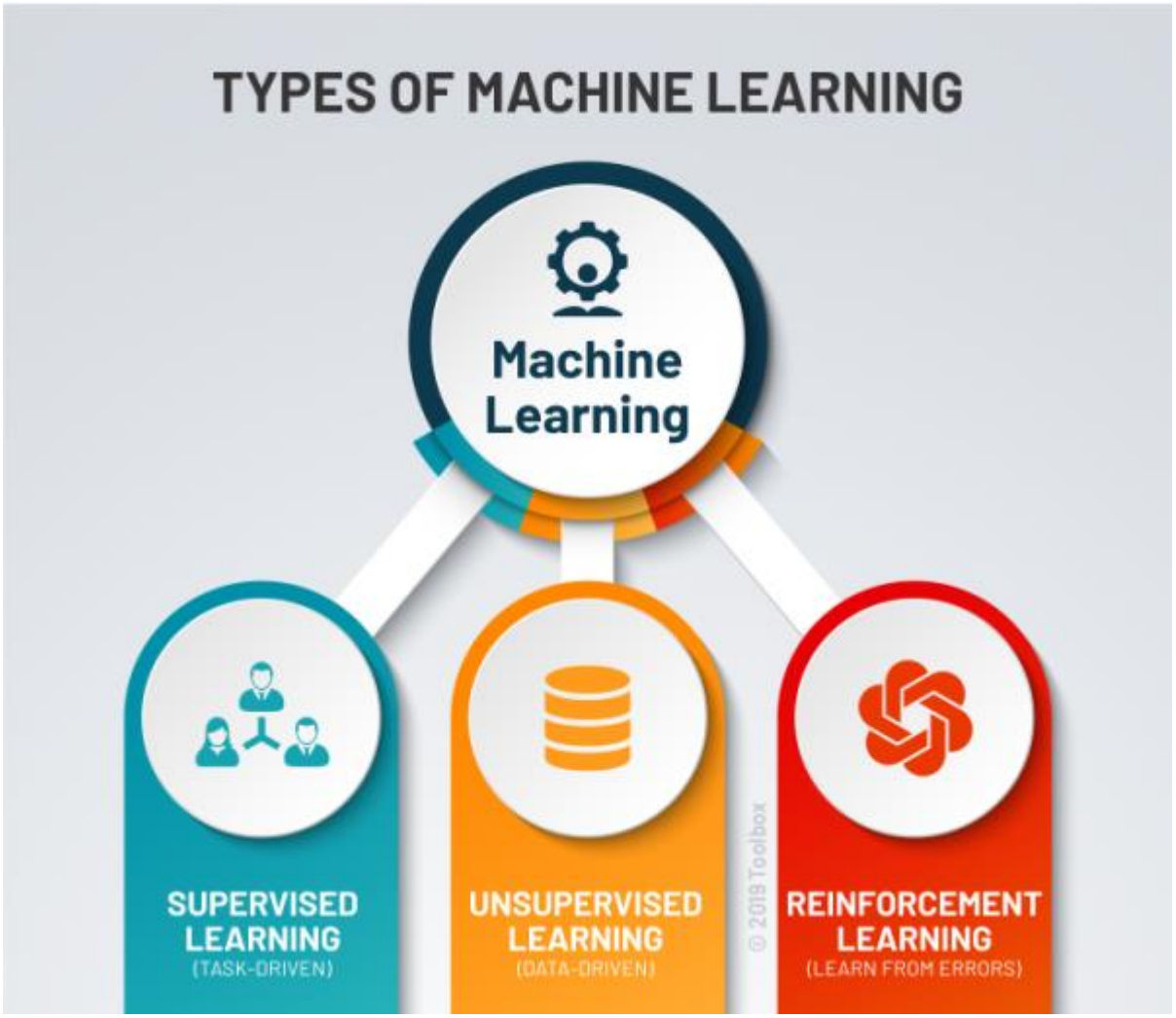
Slika 1: Odnos veštačke inteligencije, mašinskog učenja i dubokog učenja [9]

Mašinskim učenjem se može rešavati širok spektar problema, kao što su: detekcija objekata, klasifikacija objekata, prepoznavanje glasa, automatizacija korisničke podrške, prepoznavanje slova sa slike. Naravno, realan broj problema u čijem rešavanju se možemo pomoći mašinskim učenjem je izuzetno veliki i bilo bi skoro nemoguće sve primere navesti.

2.1 TIPOVI MAŠINSKOG UČENJA

Postoje 3 glavna tipa mašinskog učenja (slika 2):

- Nadgledano učenje
- Nenadgledano učenje
- Podsticajno učenje



Slika 2: Tipovi mašinskog učenja [11]

1.1.1 NADGLEDANO UČENJE:

Nadgledano učenje zahteva trening set koji se sastoji od ulaznih vektora i željenih izlaznih vektora, svaki od kojih je asociran sa određenim ulaznim vektorom. Model koristi željeni izlazni vektor radi utvrdjivanja uspešnosti učenja i kako bi podesio parametre neophodne za smanjenje sveukupne greške. [8, str. 27]

Dve osnovne vrste problema nadgledanog učenja su regresija i klasifikacija. Regresija je predviđanje neprekidne promenljive, na primer, predviđanje cene kriptovalute u odnosu na trendove uspona i padova iste u zadnjih nekoliko nedelja. Klasifikacija je predviđanje kategoričke celine promenljive. Kategoričke celine su celine među kojima nema međusobnog uređenja niti

na bilo koji način zavise jedna od druge. Primer klasifikacije je algoritam koji će se koristiti dalje u radu i koji se bavi problemom prepoznavanja saobraćajnih znakova.

Slika 3: Primer podataka iz seta za učenje modela za nadgledano učenje.

Koraci za rešavanje problema nadgledanog učenja:

1. Definisati tip podataka koji ćemo koristiti za treniranje modela. U konkretnom primeru iz ovog rada, podaci će biti slike, na kojima su obeleženi saobraćajni znakovi.
2. Prikupljanje podataka za treniranje. Podaci moraju predstavljati podatke nad kojima se očekuje da naš model kasnije vrši predikcije
3. Paziti na reprezentaciju atributa u ulaznim podacima. Preciznost pogađanja određenog objekta je usko vezana za to kako je objekat predstavljen u trening setu. Objekat se pretvara u vektor atributa, koji sadrži određeni broj atributa koji opisuju očekivani objekat. Broj atributa ne treba biti preveliki, već sadržati dovoljno podataka kako bi se predikcija uspešno obavila u realnim uslovima.

4. Odabrati strukturu algoritma. U zavisnosti od problema, neki algoritmi daju bolje rezultate od drugih.
5. Treniranje modela mašinskog učenja na setu podataka za učenje koji je prikupljen (slika 3).
6. Ocenjivanje preciznosti modela. Nakon završenog treninga, preciznost modela se mora oceniti na test setu podataka, koji sadrži podatke koji se ne nalaze u setu za trening, kako bi se ocenila sposobnost modela da vrši predikcije nad novim podacima.

Postoje mnogi algoritmi za nadgledano učenje i svi imaju svoje snage i slabosti. Ne postoji ni jedan algoritam koji je bolji od svih ostalih u rešavanju problema nadgledanog učenja.

Minimizacija empirijskog rizika

Minimizacija empirijskog rizika predstavlja jedan od glavnih problema u nadgledanom učenju. Ukoliko se ua primer uzmu X i Y i cilj je da algoritam nauči funkciju h:X→Y tako da za svaki ulaz x∈X postoji izlaz y∈Y. Za kreiranje te funkcije, algoritam na raspolaganju ima skup od n podataka (x1,y1),(x2,y2),...(xn,yn) gde y predstavlja željeni izlaz iz funkcije za dati ulaz x. Ukoliko nam je poznata raspodela P(x,y), Stvarni rizik možemo sračunati koristeći formulu:

$$R(f) = E[L(y, f(x))] = \int L(y, f(x))p(x, y)dxdy$$

L predstavlja funkciju gubitaka, koja meri odstupanje vrednosti f(x) od željene vrednosti y. p(x,y) predstavlja gustinu raspodele.

U slučaju mašinskog učenja, stvarni rizik se ne može izračunati jer je gustina raspodele p(x,y) nepoznata. Umesto stvarnog rizika, može se koristiti empirijski rizik, gde se umesto gustine raspodele računa srednja vrednost funkcije gubitaka L na nekom setu podataka veličine N:

$$E(f) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, f(x_i))$$

U slučaju klasifikacije, princip minimizacije empirijskog rizika se svodi na rešavanje problema:

$$\min_w \frac{1}{N}\sum_{i=1}^{N} L(y_i, f_w(x_i))$$

Problem sa empirijskom minimizacijom greške je taj, da ukoliko ima puno mogućih kompleksnih funkcija za funkciju *f* ili je skup trening podataka premali, samo empirijskom minimizacijom rizika lako se može dovesti do visoke varijanse i loše generalizacije, takozvanog preprilagođavanja.

Minimizacija strukturalnog rizika

Cilj minimizacije strukturalnog rizika je da se uz pomoć regularizacionog izraza izbegne preprilagođavanje do kog bi došlo korišćenjem samo minimizacije empirijskog rizika

Minimizaciju strukturalnog rizika može se predstaviti formulom:

$$S(f) = E(f) + \lambda C(f)$$

E(f) predstavlja izraz za minimizaciju empirijskog rizika, dok $\lambda C(f)$ predstavlja regularizacioni parametar pomoću koga se može kontrolisati odnos sistemskog odstupanja i varijanse. Sa λ=0, sistemsko odstupanje i empirijski rizik su jako mali, dok je varijansa velika, što se λ više povećava, to je veća sistemska greška i rizik, ali se varijansa smanjuje i time se dobija model koji dobro generalizuje.

Regularizacija

Regularizacija predstavlja tehniku za smanjenje verovatnoće preprilagođavanja. Kada bi model težio isključivo smanjenju srednje greške, jako brzo bi došlo do preprilagođavanja i predikcije takvog modela bi bile loše. Takođe, ako bi se modelu prilagođavanje potpuno onemogućilo, nikada ne bi bilo preprilagođavanja tog modela, ali takav model ništa ne bi naučio. Zbog toga se uvodi funkcija regularizacije, koja će u određenoj meri otežavati prilagođavanje, ali ne toliko da ga time potpuno onemogući. Koja težina se pridaje minimizaciji srednje greške, a koja regularizacionom izrazu kontroliše se *regularizacionim parametrom* (λ). Sledeća formula predstavlja implementaciju L2 regularizacije:

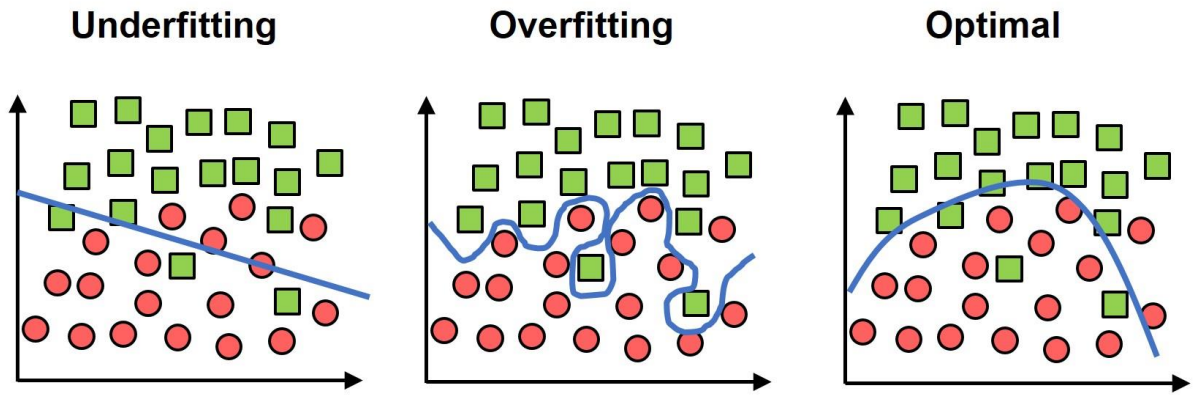
$$\min_w \frac{1}{N}\sum_{i=0}^{N} L(y_i, f_w(x_i)) + \lambda\|w\|_2^2$$

Izraz $\|w\|_2^2$ predstavlja regularizacioni izraz, koji je često kvadrat L2 norme vektora koeficijenata, a regularizacioni parametar λ reguliše uticaj regularizacionog izraza na celokupnu funkciju gubitaka.

Preprilagođavanje

U radu [1, str. 1] se preprilagođavanje objašnjava kao „korišćenje modela ili procedure koji narušavaju princip štedljivosti – tačnije, koji sadrže više promenljivih nego što je neophodno ili koji koriste pristupe koji su složeniji nego što je neophodno."

Problem nastaje često zbog premalog seta podataka za treniranje, gde model pored poželjnih atributa koji opisuju određeni objekat takođe krene da posmatra i podatke koji nisu nužno deo tog objekta (u slučaju saobraćajnog znaka to može biti plavo nebo ili drvo koje je deo pozadine), opisujući objekat sa više parametara nego što je potrebno, što dovodi do smanjenja sredje greške, ali jako niske generalizacije (slika 4), gde je pogađanje nečega van skupa za treniranje skoro nemoguće.



Slika 4: podprilagođen, preprilagođen i optimalan model [10]

Preprilagođen model ima malu srednju grešku na skupu podataka za treniranje, ali veliku srednju grešku na skupu za testiranje.

Podprilagođavanje

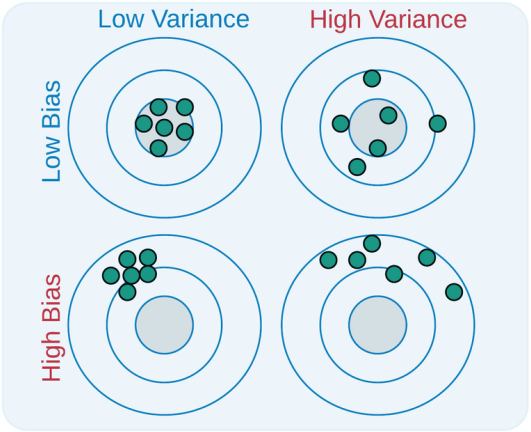
Potpunu suprotnost od preprilagođavanja čini podprilagođavanje. U ovom slučaju, model nije u mogućnosti da podatke opiše sa dovoljnim brojem parametara, ne uzimajući time neke bitne atribute tih podataka u obzir, što rezultuje lošim predikcijama tog modela.

Odnos sistemskog odstupanja i varijanse (bias-variance tradeoff)

Sistemsko odstupanje predstavlja razliku između predikcije modela i stvarne vrednosti. Model sa velikim sistemskim odstupanjem jako malo pažnje obraća na specifičnosti u podacima nad kojima vrši predikciju i preuprošćava ih. Sa druge strane, varijansa predstavlja mogućnost modela da generalizuje podatke kako bi se lakše prilagodio novim podacima. Model sa velikom varijansom loše generalizuje na nove podatke.

Idealno, model obraća pažnju na sve sitne detalje u podacima, ali i dobro generalizuje na nove podatke. Naravno, u praksi ovo nije mogće i zbog toga se mora naći sredina gde su sistemsko odstupanje i varijansa u balansu tako da model obraća pažnju na dovoljno detalja u podacima, ali ne toliko da mu to ne dozvoli da generalizuje naučeno.

Na slici 5 je prikazano delovanje sistemskog odstupanja i varijanse na raspodelu podataka.



Slika 5: Dijagram meta koji prikazuje uticaj sistemskog odstupanja i varijanse na raspodelu podataka. [12]

Matematički dokaz za povezanost sistemskog odstupanja i varijanse:

Varijansu promenljive θ možemo predstaviti pomoću formule:

$$Var(\theta) = E_\theta\left[\left(\hat{\theta} - E_\theta[\hat{\theta}]\right)^2\right]$$

***Sistemsko* odstupanje** promenljive $\hat{\theta}$ od vrednosti θ možemo predstaviti pomoću formule:

$$\mathrm{Bias}_\theta(\theta, \hat{\theta}) = \mathrm{E}_\theta[\hat{\theta}] - \theta$$

Srednju grešku za rezultat predikcije $\hat{\theta}$ u odnosu na nepoznatu vrednost θ računamo formulom

$$\mathrm{MSE}(\hat{\theta}) = \mathrm{E}_{\theta}[(\hat{\theta} - \theta)^2]$$

Odnos sistemskog odstupanja i varijanse može se dokazati na sledeći način:

$$\begin{aligned}
\mathrm{MSE}(\hat{\theta}) &= \mathrm{E}_\theta\left[(\hat{\theta} - \theta)^2\right] \\
&= \mathrm{E}_\theta\left[\left(\hat{\theta} - \mathrm{E}_\theta[\hat{\theta}] + \mathrm{E}_\theta[\hat{\theta}] - \theta\right)^2\right] \\
&= \mathrm{E}_\theta\left[\left(\hat{\theta} - \mathrm{E}_\theta[\hat{\theta}]\right)^2 + 2\left(\hat{\theta} - \mathrm{E}_\theta[\hat{\theta}]\right)\left(\mathrm{E}_\theta[\hat{\theta}] - \theta\right) + \left(\mathrm{E}_\theta[\hat{\theta}] - \theta\right)^2\right] \\
&= \mathrm{E}_\theta\left[\left(\hat{\theta} - \mathrm{E}_\theta[\hat{\theta}]\right)^2\right] + \mathrm{E}_\theta\left[2\left(\hat{\theta} - \mathrm{E}_\theta[\hat{\theta}]\right)\left(\mathrm{E}_\theta[\hat{\theta}] - \theta\right)\right] + \mathrm{E}_\theta\left[\left(\mathrm{E}_\theta[\hat{\theta}] - \theta\right)^2\right] \\
&= \mathrm{E}_\theta\left[\left(\hat{\theta} - \mathrm{E}_\theta[\hat{\theta}]\right)^2\right] + 2\left(\mathrm{E}_\theta[\hat{\theta}] - \theta\right)\mathrm{E}_\theta\left[\hat{\theta} - \mathrm{E}_\theta[\hat{\theta}]\right] + \left(\mathrm{E}_\theta[\hat{\theta}] - \theta\right)^2 && \mathrm{E}_\theta[\hat{\theta}] - \theta = \text{const.} \\
&= \mathrm{E}_\theta\left[\left(\hat{\theta} - \mathrm{E}_\theta[\hat{\theta}]\right)^2\right] + 2\left(\mathrm{E}_\theta[\hat{\theta}] - \theta\right)\left(\mathrm{E}_\theta[\hat{\theta}] - \mathrm{E}_\theta[\hat{\theta}]\right) + \left(\mathrm{E}_\theta[\hat{\theta}] - \theta\right)^2 && \mathrm{E}_\theta[\hat{\theta}] = \text{const.} \\
&= \mathrm{E}_\theta\left[\left(\hat{\theta} - \mathrm{E}_\theta[\hat{\theta}]\right)^2\right] + \left(\mathrm{E}_\theta[\hat{\theta}] - \theta\right)^2 \\
&= \mathrm{Var}_\theta(\hat{\theta}) + \mathrm{Bias}_\theta(\hat{\theta}, \theta)^2
\end{aligned}$$

Slika 1.7 Dokaz o odnosu sistemskog odstupanja i varijanse

Najpopularniji algoritmi nadgledanog učenja su:

- Veštačke neuralne mreže
- Linearna i logistička regresija
- Naivni bajes
- Drvo odlučivanja
- K-najbližih suseda

Neki od primera korišćenja nadgledanog učenja su:

- Detekcija i klasifikacija objekata
- Prediktivna analiza
- Detekcija spam mejl poruka
- Automatizacija korisničke podrške

2.1.2 NENADGLEDANO UČENJE

Za razliku od nadgledanog učenja, gde su podaci za trening obeleženi, kako bi algoritam prilikom trening znao koliko njegove predikcije odstupaju od željenog izlaza, kod nenadgledanog učenja, podaci u skupu za trening nisu obeleženi i prvi zadatak algoritma je ustanovi koji su to šabloni (eng. *pattern*) prisutni u podacima.

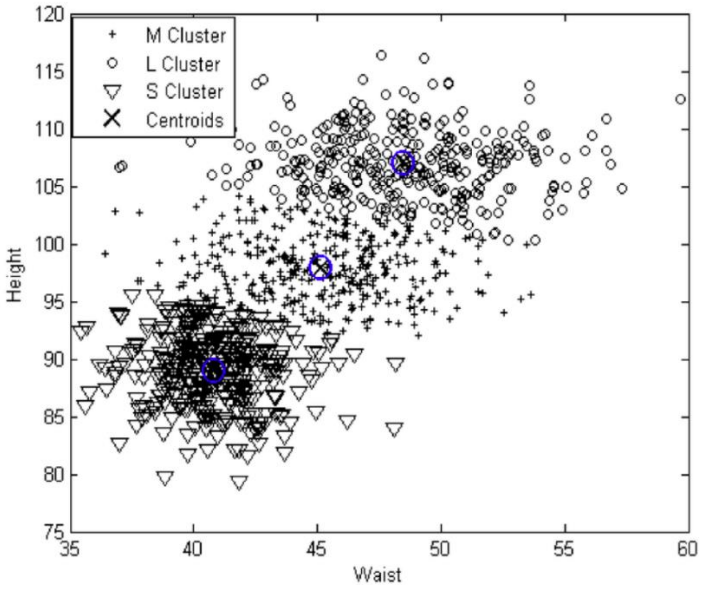
Veliku prednost ovih algoritama predstavlja to što podaci ne moraju biti ručno obeleženi. Čime se znatno umanjuje napor uložen za pribavljanje i uređivanje skupa podataka. Takođe, eliminiše se mogućnost ljudske greške, koja se može dogoditi usled umora, repetativnih poslova, kao što je obeležavanje podataka, ili nekih drugih neočekivanih okolnosti i ukoliko je dovoljno učestala, može značajno uticati na podobnost algoritma nakon treninga obavljenog nad tim podacima.

Sa druge strane, za treniranje algoritama nenadgledanog učenja, potreban je veliki skup trening podataka, mnogo veći nego što bi algoritam nadgledanog učenja zahtevao. Takođe, pošto se algoritmu ne navodi eksplicitno željeni rezultat predikcije, moguće je da iz podataka algoritam nauči neke neočekivane i lažne činjenice o podacima iz nekih naizgled nebitnih artefakta.

Klasterovanje

Klasterovanje predstavlja pronalaženje grupa u okviru skupa podataka. „Pojam klasterovanja nije jednoznačno definisan. U jednom skupu se može identifikovati više različitih grupisanja, često različite granularnosti. Pritom, takvi slučajevi nisu posledica nedovoljnog promišljanja definicije klasterovanja, već raznovrsnosti konteksta u kojima se grupisanje može vršiti i ciljeva koji se pomoću klasterovanja žele postići.“ [7 str. 199] Uglavnom algoritmi klasterovanja omogućavaju odabir željenog broj klastera u koji raspodeliti podatke iz skupa. Takođe, postoji više načina da se neki skup podataka(tačaka) raspodeli u klastere.

Osnovna i najčešće korišćena vrsta klastera je poznata kao *centrični* klaster, u kojima su svi elementi smešteni u unutrašnjost nekog kruga ili elipse, koji predstavlja granice tog klastera(slika 6). Klasteri se mogu kreirati i u odnosu na gustinu tačaka u nekom regionu i oni se nazivaju *gustinski* klasteri. Ukoliko imamo potrebu za „hijerarhijom“ klastera, gde jedan klaster može sadržati nekoliko odvojenih klastera, nazivamo *hijerarhijsko klasterovanje*.



Slika 6: Preporuka veličine odeće pomoću klasterovanja [13]

Algoritam K sredina

Algoritam K sredina je jedan od najpoznatijih algoritama za klasterovanje. On u podacima nalazi K klastera. Primenljiv je samo na podatke koji se mogu uprošćavati, kao na primer vektori, jer se svaki od K klastera nalazi pomoću centroida tog klastera, koji se dobija uprošćavanjem elemenata tog klastera.

Kako algoritam funkcioniše se može predstaviti pomoću 3 koraka koji se ponavljaju sve dok rezultat u 2 uzastopne iteracije nije isti. Pre svega, potrebno je odabrati gde staviti centroide. Ovo se može obaviti odabirom nasumične vrednosti ili postavljanjem centroide na mesto nekog od elemenata. Nakon postavljanja centroida, ponavljaju se sledeća 2 koraka:

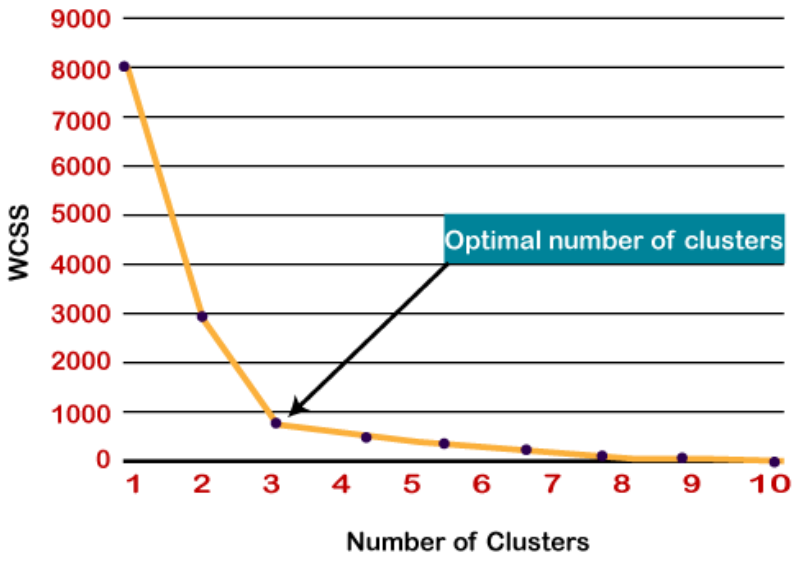
1. Raspoređivanje svih tačaka u klastere tako što se tačka pridruži najbližoj centroidi
2. Računanje nove centroide računanjem prosečne vrednosti instanci koje se nalaze u njenom klasteru

Može se reći da ovaj algoritam rešava problem

$$\min_{c_i} \sum_{j=1}^{K} \sum_{i=1}^{n} d(x, c_i)^2$$

Gde K predstavlja broj klastera, n broj klasa, a d euklidsko rastojanje, koje trebamo smanjiti po c_i

U praksi je često teško odabrati K tako da se podaci podele na smislen broj klastera. Jedan od načina je takozvano „pravilo lakta", gde se za različite vrednosti K iscrtava grafik zavisnosti sume kvadrata rastojanja u odnosu na K i za broj klastera se uzima ona tačka na kojoj suma kvadrata rastojanja naglo prestaje da opada (slika 7), što znači da se daljim povećavanjem K neće značajno doprineti daljem opadanju sume kvadrata rastojanja.



Slika 7: odabir broja klastera pravilom lakta [14]

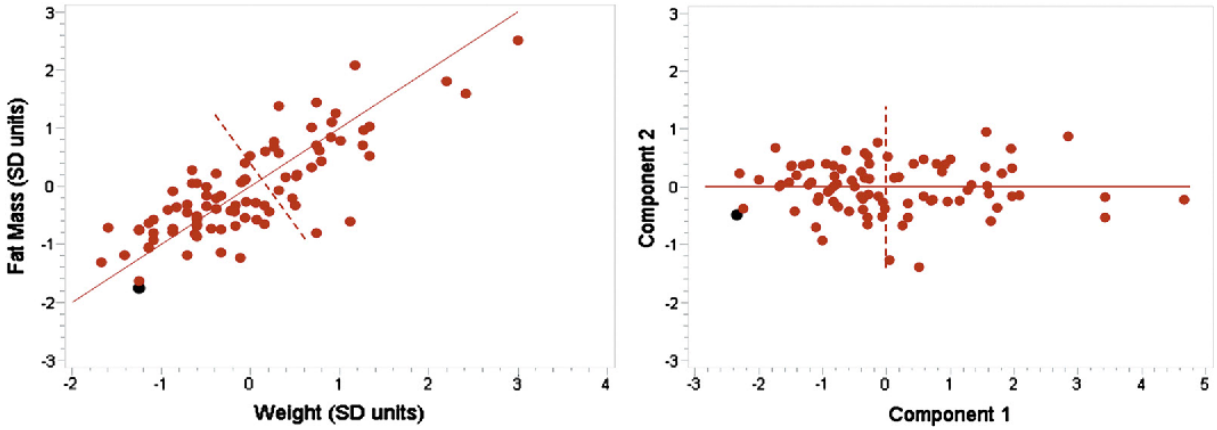
Smanjenje dimenzionalnosti

Uglavnom, veća količina podataka pruža bolje rezultate, ona može i uticati na performanse algoritma mašinskog učenja (na primer, preprilagođavanjem) i može otežati vizualizaciju seta podataka. Smanjenje dimenzionalnosti je tehnika koja se koristi kada je količina atributa ili dimenzija u setu podataka prevelika. Pomoću nje se količina ulaznih podataka smanjuje na korisniju veličinu, dok se integritet seta podataka čuva koliko je to moguće. [28]

Metoda analize glavnih komponenti

Metoda analize glavnih komponenti je metoda za smanjenje dimenzionalnosti seta podataka, koja za cilj ima da smanji broj atributa prisutnih u tom setu, a da pri tome minimizira gubitak podataka. Analizom glavnih komponenti može se ustanoviti u kom „pravcu" u setu ima najviše varijanse, u kom ima najmanje i koliko se podataka gubi ukoliko se dimenzija najsiromašnija podacima izostavi iz finalnog seta podataka. Analiza glavnih komponenti se može izvršiti u 5 koraka:

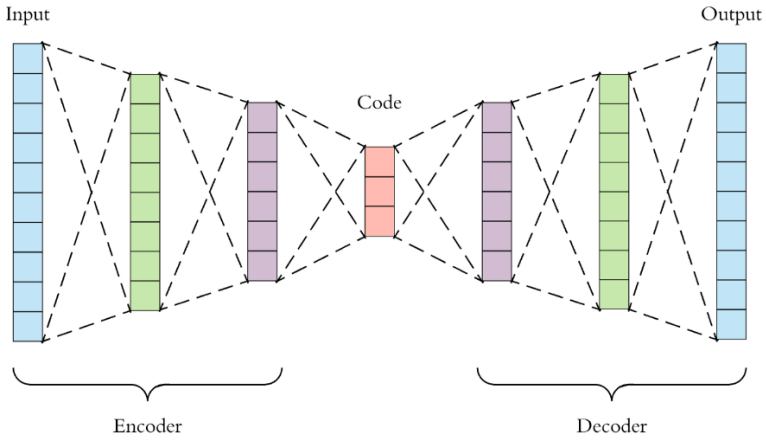
1. **Standardizacija**: Pošto ne moraju svi atributi da prate isktu skalu vrednosti (na primer, od 0 do 100), kao prvi korak je neophodno uraditi standardizaciju, gde skale svih atributa menjamo tako da su međusobno iste. To možemo matematički uraditi tako što od vrednosti atributa oduzmemo srednju vrednost i potom razliku podelimo standardnom devijacijom.
2. **Matrica kovarijanse**: Cilj izračunavanja matrice kovarijanse je da ustanovimo kako atributi odstupaju od srednje vrednosti u odnosu na vrednosti drugih atributa i ustanovimo da li su i kako njihova odstupanja povezana.
3. **Izračunavanje sopstvenih vektora i sopstvenih vrednosti**: Bitno je prvo napomenuti da sopstveni vektori dolaze u paru sa sopstvenim vrednostima i broj parova je uvek jednak broju dimenzija podataka. Zapravo, sopstveni vektor predstavlja pravac na kome u setu postoji najviše varijanse, dok sopstvene vrednosti predstavljaju količinu varijanse na tom pravcu. Bitno je napomenuti da prvi vektor uvek sadrži najveću količinu informacija, drugi manje, treći još manje i tako dalje.
4. **Kreiranje vektora atributa**: Radi smanjenja dimenzionalnost seta podataka, kreira se vektor artributa, koji nije ništa drugo nego vektor koji sadrži svih N sopstvenih vektora. On se može formirati od svih N sopstvenih vektora. Ukoliko je potrebno smanjiti dimenzionalnost podataka, idealno se odabira onaj vektor koji nosi najmanje podataka, jer se time gubi cela dimenzija, a gubitak podataka ostaje minimalan.
5. **Rotiranje podataka u odnosu na ose glavnih komponenti**: Pomoću vektora atributa dobijenog u 4. koraku, množi se transponovani prvobitni set podataka sa transponovanim vektorom atributa, čime se podaci pomeraju sa početnih osa na ose opisane glavnim komponentama (slika 8).



Slika 8: Metoda analize glavnih komponenti [15]

Autoenkoderi

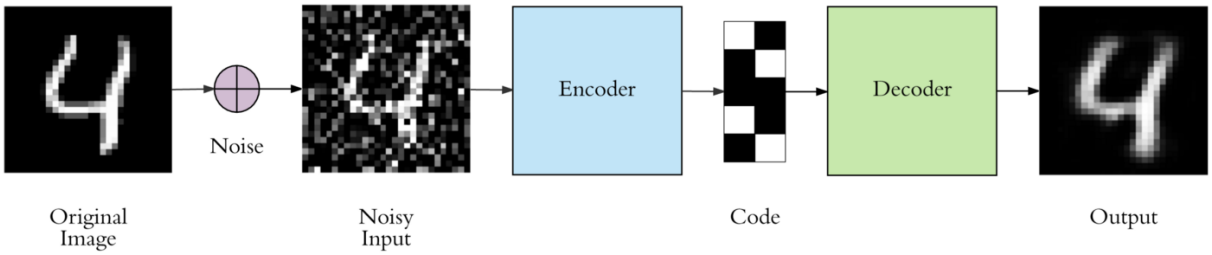
Za razliku od neuronskih mreža korišćenih za klasifikaciju, koje za svaki ulazni parametar imaju zaseban neuron koji taj parametar prima, a na izlazu imaju znatno manje neurona, gde svaki neuron predstavlja neku od klasa koje klasifikator treba da prepozna, autoenkoderi imaju isti broj ulaznih i izlaznih neurona. Tako da ako je ulazni podatak slika, i izlazni podatak će takodje biti slika.



Slika 9: šema autoenkodera [16]

Kao što se može videti na slici 9, broj ulaznih i izlaznih parametara je jednak, ali je broj parametara u srednjem skrivenom sloju znatno manji, kako bi stvorio efekat uskog grla i morao da „odbaci" reprezentacije podataka koje su najmanje relevantne, dok će se glavne komponente sačuvati i biti prepoznatljive na izlazu. Reprezentacija koju daje srednji sloj naziva se još i *„latentni prostor"* i predstavlja modifikaciju ulaznih podataka u skladu sa modifikacijama koje su naučene na setu za trening.

Jedan od primera korišćenja autoenkodera predstavlja autoenkoder za uklanjanje šuma (slika 10), koji se može kreirati tako što se kao ulazni parametar autoenkodera bira sliku sa šumom i izlaz se ocenjuje u odnosu na to koliko je taj šum uspešno otklonjen u odnosu na originalnu sliku. U praksi, ovakva metoda otklanjanja šuma je samo u određenoj meri korisna jer šum koji dodajemo trening podacima mora približno predstavljati šum prisutan na test podacima, ali ukoliko se taj uslov ispuni, može biti efektivna.



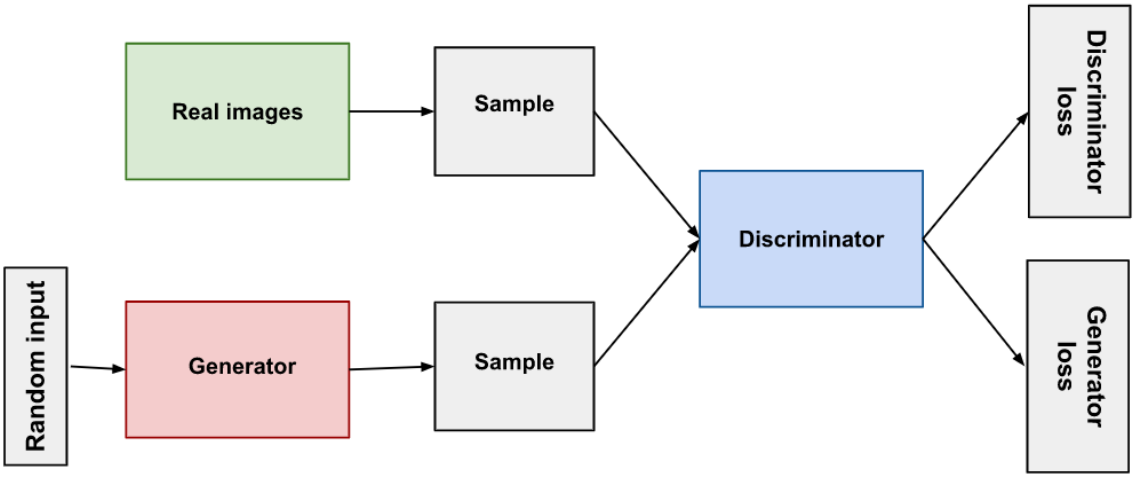
Slika 10: uklanjanje šuma autoenkoderom [16]

Generativni modeli

Generativni modeli su klasa statističkih modela koji za cilj imaju generisanje novih podataka. Ovi modeli su suprotnost od diskriminativnih modela, koji za cilj imaju da već postojeće podatke podele na određeni način. Možemo reći da generativni modeli modeluju naučenu raspodelu p(x), dok diskriminativni modeli modeluju uslovnu raspodelu p(Y|X).

Generativne suparničke mreže

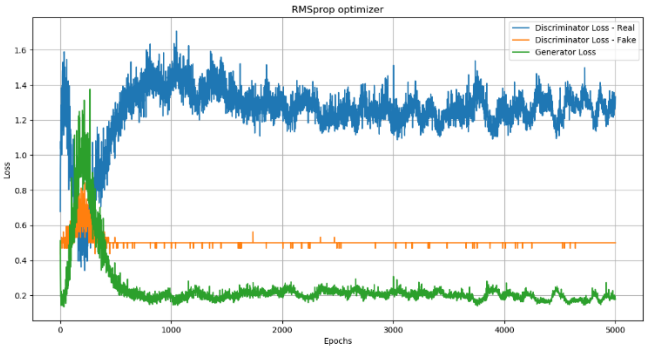
Generativne suparničke mreže predstavljaju skup od 2 modela, od kojih je jedan generator i jedan diskriminator (slika 11). Cilj obučavanja diskriminatora je da što bolje razlikuje podatke koje je generisao generator od pravih, dok je cilj obučavanja generatora da kreira podatke što bliže pravima. Obučavanje se vrši naizmenično, kako bi oba ova modela napredovala i medjusobno se usavršavala. [7 str 217]



Slika 11: Arhitektura generativne suparničke mreže [17]

Treniranje generativne suparničke mreže predstavlja poseban problem, jer kako proces treniranja odmiče, podaci koje generiše generator postaju sve sličniji stvarnim podacima i diskriminatoru je sve teže da razdvoji prave podatke od onih koje je kreirao generator. Trening se odvija tako što se generator trenira nekoliko epoha, pa se zatim diskriminator trenira nekoliko epoha (slika 12). Problem predstavlja to što kako trening odmiče i generator postaje sve bolji u generisanju novih podataka, povratne informacije diskriminatora postaju sve beznačajnije. Ukoliko se trening nastavi nakon trenutka kada diskriminator krene da daje nasumične povratne informacije, može se desiti da generator krene da trenira na lošim podacima i kao rezultat toga

generisani podaci izgube kvalitet.[2] Savršen generator će osigurati da je preciznost diskriminatora 50%.



Slika 12: Funkcija gubitaka pri treniranju generativne suparničke mreže. Gubici generatora opadaju, dok se gubici diskriminatora povećavaju.

2.1.3 PODSTICAJNO UČENJE

Podsticajno učenje se izučava u mnogim naučnim disciplinama, kao što su teorja informacija, optimizacija zasnovana na simulaciji, sistemi sa više agenata, kolektivna inteligencija i mnoge druge.

Cilj podsticajnog učenja je da se putem funkcije nagrade za obavljanje željene akcije algoritam trenira tako da što više uveća nagradu koju dobija. Ovaj sistem donekle pokušava da oponaša sistem nagrade i kazne koji je urođen svim živim bićima, koja ukoliko rade nešto što ih ugrožava osećaju glad, žeđ, bol, a ako rade nešto što doprinosi održanju njihovog zdravlja i vrste, osećaju se srećno. Jedan od čestih primera podsticajnog učenja je dresura pasa. Pas se nagrađivanjem za ponašanje koje je poželjno i kažnjavanjem nepoželjnog ponašanja uči određenim šablonima ponašanja koji ga čine podobnim za život sa ljudima.

Na isti način algoritam „agent" podsticajno učenje uči kako da se „ponaša" u nepoznatom okruženju i koji šabloni ponašanja će biti najviše nagrađivani.

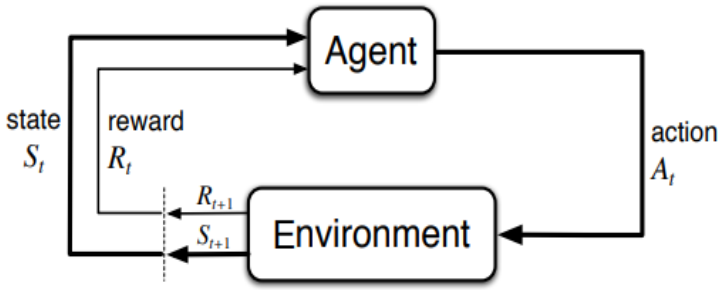
Markovljevi procesi odlučivanja

Ovo je osnovni model podstocajnog učenja i sadrži 4 celine:

1. Moguće postavke sveta i agenata ***S***
2. Moguće akcije koje agenti mogu preduzeti ***A***
3. Funkcija za određivanje nagrade ***R(S,A)***
4. Opis efekta svake akcije na svako stanje ***T***

Tok procesa je: Agent u okruženju S_i preduzima akciju A_i i kao rezultat dobija nagradu izračunatu po formuli $R(S_i,A_i)$ i novo okruženje S_{i+1} (slika 13).

Preduzete akcije ***A*** u određenoj postavci ***S*** u obzir ne uzimaju prethodne postavke, već se odnose isključivo na trenutnu.



Slika 13: Markovljev proces odlučivanja [18]

2.2 NEURONSKE MREŽE I DUBOKO UČENJE

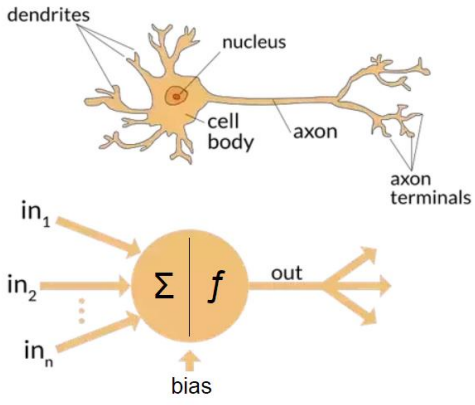
Neuronske mreže su jedna od najšire rasprostranjenih i najprimenjenijih metoda mašinskog učenja. Domen njihove korisnosti je izuzetno veliki i mogu se koristiti za probleme od kategorizacije teksta i prepoznavanja objekata sa slika, do medicinske dijagnoze i autonomne vožnje automobila.

U radu [7, str. 135] se navodi da "U svetlu njihovih izvanrednih uspeha i velike popularnosti, u laičkim krugovima postoji tendencija poistovećavanja mašinskog učenja, pa čak i veštačke inteligencije sa neuronskim mrežama. Ovakav pogled je prosto pogrešan. Takođe, postoji tendencija da se neuronska mreža razmatra kao prvi izbor metoda mašinskog učenja nevezano o kakvom se problemu radi."

Ovakva težnja neuronskim mrežama može biti pogubna bez predhodnog razmatranja problema koji želimo da rešimo, kao i njihovih prednosti i mana, koje, kao i svi ostali modeli mašinskog učenja imaju. Jedna od najvećih prednosti neuronskih mreža je njihova sposobnost da same konstruišu nove atribute kojima opisuju podatke na osnovu seta podataka za treniranje. Ukoliko je skup podataka za treniranje mali, velika je verovatnoća preprilagođavanja i najbolje bi bilo koristiti neki drugi model mašinskog učenja, dok su za velike setove podataka neuronske mreže ili neka njihova varijacija često dobar izbor.

Postoje različite arhitekture neuronskih mreža. Osnovnu predstavljaju *potpuno povezane neuronske mreže*, za zadatke kao što su obrada slika , teksta i sličnih podataka, koriste se *konvolutivne neuronske mreže*, za obradu podataka koji su predstavljeni nizovima koriste se *rekurentne neuronske mreže*, za podatke koji se mogu predstaviti pomoću stabla koriste *rekurzivne neuronske mreže***,** dok se za podatke koji se mogu predstaviti grafom koriste se *grafovske neuronske mreže*. U nastavku rada bliže će biti objašnjene **konvolutivne neuronske mreže** i **rekurentne neuronske mreže**

Neuronske mreže se sastoje od **neurona** i **veza** između neurona. Neuroni su delimična reprezentacija koncepata na kojima se zasniva funkcionisanje bioloških neurona unutar mozga. Svaki neuron može imati jednu ili više ulaznih vrednosti, jednu izlaznu vrednost koja je povezana na ulaz jednog ili više neurona (slika 14). Takođe, svaki neuron ima određenu funkciju aktivacije, koja na osnovu ulaznih vrednosti u neuron definiše izlaznu vrednost. Veze predstavljaju veze između izlaza neurona jednog sloja sa ulazom neurona nekog drugog sloja. Od atributa, svaka veza ima određenu težinu, koja diktira koliki uticaj će taj neuron čiji je izlaz povezan tom vezom imati na neuron do čijeg ulaza ta veza vodi.



Slika 14: Razlika između biološkog neurona i veštačkog neurona [19]

2.2.1 AKTIVACIONE FUNKCIJE

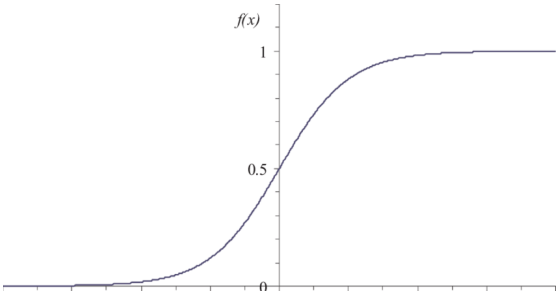
Zadatak aktivacione funkcije je da u odnosu na ulazne vrednosti neurona odredi njegovu izlaznu vrednost, na osnovu ulaznog signala i sklonosti neurona ka aktivaciji. *Ukupan ulazni signal* računamo kao sumu proizvoda izlaznih vrednosti neurona čiji je izlaz vezan na ulaz neurona za koji se računa vrednost signala i težina koje su vezane za te veze ($net = \sum_{i=1}^{I} y_i v_i$). Sklonost neurona ka aktivaciji predstavlja koliko je ta funkcija pomerena od centra koordinatnog sistema.

Aktivacionu funkciju možemo predstaviti kao $f_a = \sum_{i=1}^{I} y_i v, - \theta$

Moguće je koristiti veliki broj različitih funkcija, ali u praksi su najčešće korišćene 3 funkcije:

Sigmoidna

Široko korišćena u neuronskim mrežama. Veliki nedostatak predstavlja to što je svuda osim u okolini nule skoro konstanta, te može doći do anuliranja gradijenata i otežavanja ili onemogućavanja učenja. Grafik ove funkcije može se videti na slici 15.



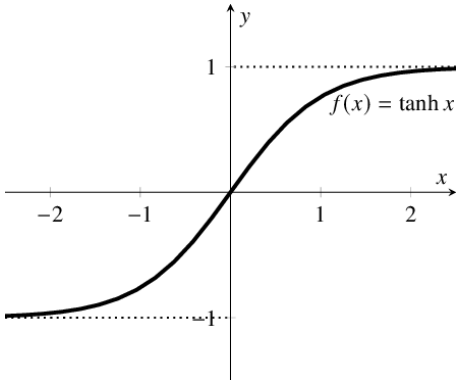
Slika 15: Sigmoidna aktivaciona funkcija

Matematički se može predstaviti formulom $\sigma(x) = \frac{1}{1+e^{-x}}$

Tangens-hiperbolička

Jako slična sigmoidnoj funkciji i od nje se u radu [5] pokazala kao podobnija za brže konvergovanje ka idealnom rešenju, u zavisnosti od podataka koji su korišćeni za treniranje. Ona je u okolini nule linearnija od sigmoidne, što olakšava optimiozaciju.

Grafik ove funkcije može se videti na slici 16.



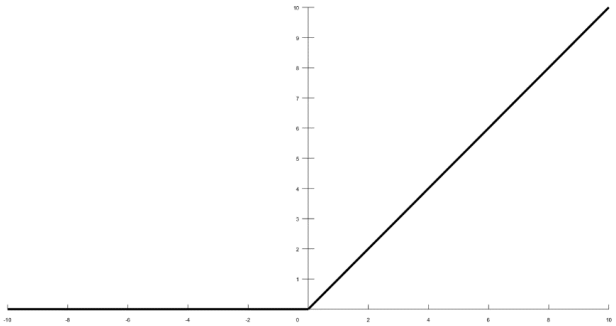
Slika 16: Tangens-hiperbolička aktivaciona funkcija

Matematilčki se predstavlja formulom $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$

Ispravljačka linearna jedinica

Trenutno najkorišćenija funkcija aktivacije. Uprkos nediferencijabilnosti, ova funkcija nudi najbolje performanse pri optimizaciji. Problem predstavlja što sve vrednosti koje su levo od nule neće imati uticaj na treniranje modela.

Grafik ove funkcije može se videti na slici 17.

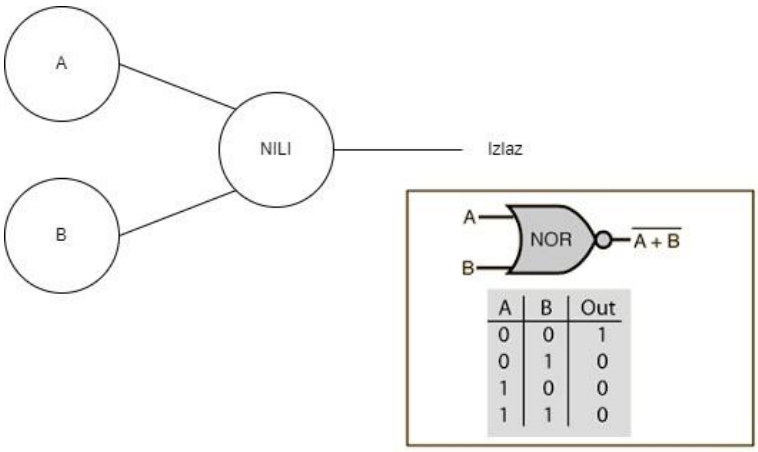


Slika 17: Ispravljačka linearna jedinica

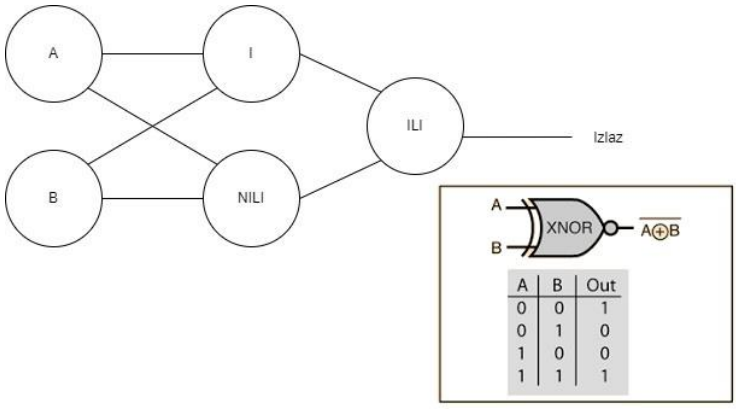
Matematički se izražava formulom rlu(x) = max(0,x)

2.2.2 UČENJE I OPTIMIZACIJA

Jedan neuron samostalno može rešavati problem čije rešenje predstavlja linearna funkcija. To znači da neuron treba da pogodi da li se data vrednost nalzi iznad ili ispod neke linearne funkcije. Tako, na primer, pomoću jednog neurona možemo realizovati **I, ILI, NI, NILI** i **NE** logičke funkcije (slika 18), jer su rešenja tih funkcija linearna, dok za rešavanje logičkih funkcija **XI** i **XNILI**, koje nisu linearne, moramo koristiti više neurona kako bi smo postigli 100% tačnosti (slika 19).



Slika 18: Predstavljanje NILI logičke funkcije putem neuronske mreže



Slika 19: Predstavljanje XNILI logičke funkcije pomoću neuronske mreže

Treniranje neurona zaprevo predstavlja podešavanje parametara v, i θ tako da se zadovoljava određeni kriterijum. Do rešenja se skoro nikada ne dolazi od jednom već u velikom broju koraka, sa ciljem da se pri svakom koraku sistem malo približi idealnom rešenju problema.

2.2.3 HIPERPARAMETRI

Vrednosti koje se podešavaju pre treniranja kako bi se optimizovao proces treninga i osigurala njegova uspešnost nazivaju se hiperparametrima.

Broj skrivenih slojeva: Definiše broj skrivenih potpuno povezanih slojeva u neuronskoj mreži.

Dropaut: Regularizaciona tehnika pomoću koje se izbegava preprilagođavanje. Određeni broj nasumično odabranih neurona se u svakoj iteraciji treniranja isključuje iz procesa optimizacije. Ovime se povećava generalizacija neuronske mreže. U radu [6] se navodi da je dobra početna vrednost za dropaut je oko 20%. Premala vrednost ovog hiperparametra neće imati skoro nikakav uticaj, dok će prevelika vrednost u velikoj meri sprečiti normalan tok učenja. Bolje performanse ove tehnike se pokazuju na većim mrežama.

Inicijalizacija težina veza: Najčešće se sve težine inicijalizuju sa istom vrednošću.

Funkcija aktivacije: Definiše koja će se funkcija aktivacije koristiti na kom sloju.

Brzina učenja: Predstavlja koliko će se veliki pomeraj niz gradijent vršiti u svakoj iteraciji treninga. Mala brzina učenja dovešće do jako spore optimizacije mreže, dok će prevelika skoro potpuno onemogućiti treniranje. Fiksna brzina učenja daje dobre rezultate na samom početku, ali potom ubrzo onemogućava dalje učenje, pa se uglavnon umesto statične vrednosti koristi funkcija koja tokom procesa učenja konstantno opada.

Momentum: Prilikom vožnje automobila, pri velikoj brzini je nemoguće to obaviti brzo, jer momentum onemogućava brzu promenu smera kretanja. Isto tako, ako se gradijentni spust određeni broj koraka odvija u jednom pravcu, momentumom se određuje koliko će u narednoj iteraciji maksimalno smeti da se promeni smer, u odnosu na dotadašnji smer kretanja. Ova metoda pomaže da algoritam ne krene da se kreće kružno po gradijentu, i zbog toga spreči dalje treniranje.

Broj epoha: Jedna epoha predstavlja period za koji model koji se trenira vidi ceo trening set jednom. Odabir vrednosti ovog parametra se najčešće obavlja tako što se broj epoha povećava dok model ne počne da se preprilagođava.

Veličina serije: Broj podataka koje će mreža videti pre optimizacije mreže. Velike serije podataka povećavaju generalizaciju.

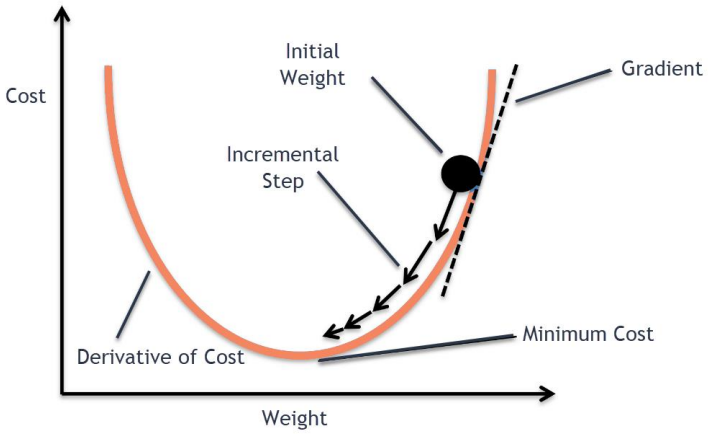
2.2.4 OPTIMIZACIJA

Gradijentni spust

U radu [8 str. 38] se tvrdi da je optimizacija gradijentnim spustom dovela do razvijanja jednog on najpopularnijig algoritama optimizacije, nazvanog **algoritam propagacije unazad.** Jedna iteracija ovog algoritma sastoji se od 2 faze:

1. **Feedforward prolaz**: Kalkuliše izlazne vrednosti neuronske mreže za svaki obrazac treniranja.
2. **Propagacija unazad:** Propagira signal greške nazad sa izlaznog sloja ka ulaznom sloju. Težine veza se podešavaju kao funkcija signala greške.

Jedan od najkorišćenijih metoda optimizacije je **gradijentni spust** (slika 20). On zahteva definisanu grešku ili funkciju pomoću koje će moći da odredi koliku je grešku neuron napravio pri pogađanju.Često se koristi zbir kvadratnih grešaka $\varepsilon = \sum_{P=1}^{P_T}(t_p - O_p)^2$, gde t_p predstavlja željenu vrednost, a O_p izlaznu vrednost neurona za p-ti par ulazna vrednost, željena vrednost (šablon) od mogućih P_T šablona. Cilj kriterijuma gradijentnog spuštanja je minimizacija vrednosti ε. Gradijent se izračunava **algoritmom propagacije unazad**, koji računa gradijent funkcije gubitaka u odnosu na težine veza, omogućavajući time treniranje višeslojnih modela. Glavni cilj je da se u većem broju koraka ka negativnoj vrednosti gradijenta dodje do najniže tačke, koja je najbliža idealnom rešenju.

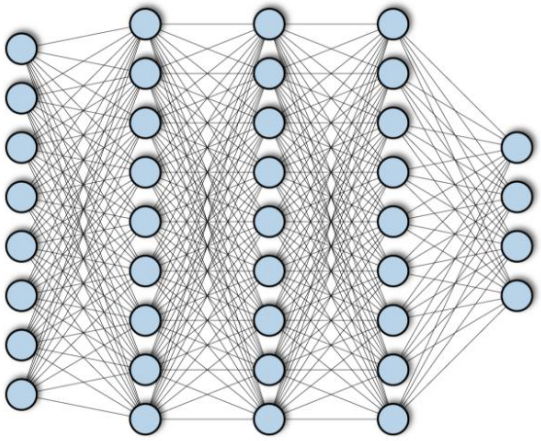


Slika 20: Gradijentno smanjenje gubitaka [20]

2.2.5 POTPUNO POVEZANE NEURONSKE MREŽE

Potpuno povezane neuronske mreže predstavljaju osnovni tip neuronske mreže, gde je izlaz svakog neurona u sloju N povezan sa ulazom svakog neurona u sloju N+1 (slika 21). Njihova prednost je širok spektar aplikacija, jer se ne moraju uzimati predpostavke iz seta podataka za treniranje koje će diktirati samu arhitekturu mreže, često daju lošije rezultate od mreže koja je napravljena za rešavanje nekog specifičnog problema.

Iako se za rešavanje nekog problema retko koristi samo potpuno povezana neuronska mreža, često je možemo naći kao deo neke druge arhitekture mreže, u vidu potpuno povezanih slojeva.



Slika 21: Potpuno povezana neuronska mreža [21]

2.2.6 KONVOLUTIVNE NEURONSKE MREŽE

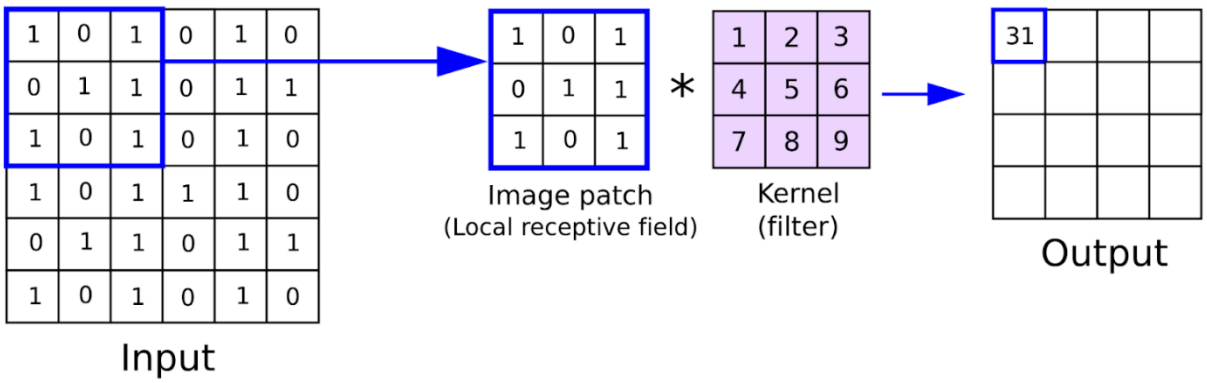
Primarno korišćene za obradu signala poput slika ili teksta. Najveća prednost ovih mreža je baš specifična za neuronske mreže, a to je da nove atribute može da generiše iz podataka i obrazaca u njima. Ovo generisanje atributa se ne odnosi samo na potpuno povezane slojeve, već i na slojeve konvolucije i agregacije, gde će mreža sama pronaći odgovarajuće filtere za date podatke, bez potrebe za manualno kreiranje istih.

Konvolutivne neuronske mreže se sastoje od tri različite vrste slojeva, a to su: **sloj konvolucije, sloj agregacije i potpuno povezani sloj.**

SLOJ KONVOLUCIJE

Sloj konvolucije vrši apstrakovanje i umanjenje veličine ulaznih podataka kako bi uz minimalne gubitke detalja iz podataka bile postignute povećane performanse. Iz ulazne matrice podataka se uzimju manji delovi matrice, množe sa filterom i njihov proizvod se upisuje u izlaznu matricu zvanu **mapa atributa** (slika 22).

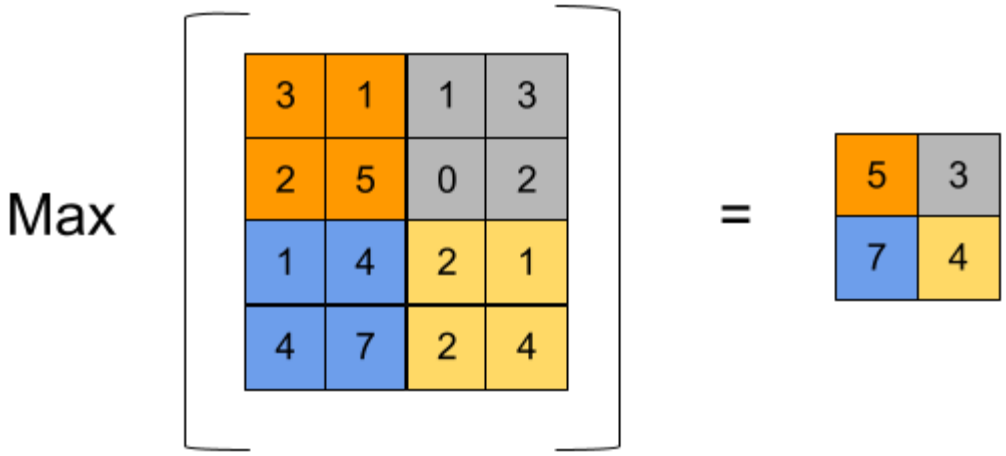
Veličina filtera je namerno manja od ulaznih podataka kako bi dopustila da se filter sa ulaznim podacima pomnoži veći broj puta na različitim delovima slike. Filter se na sliku primenjuje u inkrementima koji su veličine filtera, sa leva na desno i od vrha ka dnu. [22]



Slika 22: Princip funkcionisanja sloja konvolucije [22]

SLOJ AGREGACIJE

Primarna upotreba slojeva agregacije je takođe povećanje performansi modela kroz smanjenje kompleksnosti podataka. Sloj agregacije funkcioniše slično sloju konvolucije, sa tim da se u glavnom uzimaju manje matrice podataka po koraku (uglavnom matrice veličine 2x2) i umesto množenja filterom, rezultat agregacije se bira najčešće kao maksimalna vrednost iz matrice nad kojom se vrši agregacija (slika 23), ili kao srednja vrednost podataka u matrici.



Slika 23: Agregacija odabirom maksimalne vrednosti [23]

POTPUNNO POVEZANI SLOJ

U konvolutivnim neuronskim mrežama, potpuno povezani slojevi imaju ulogu klasifikatora i nalaze se uglavnom nakon slojeva konvolucije i agregacije. Strukturalno su identični potpuno povezanim neuronskim mrežama.

2.2.7 REKURENTNE NEURONSKE MREŽE

Ukoliko se problem koji rešavamo sastoji od nizovnih podataka, sa određenim tokom, kao što su na primer zvuk ili tekst, najbolje rešenje za obradu te vrste podataka u polju neuronskih mreža jesu rekurentne neuronske mreže.

Ove mreže su konstruisane sa idejom medjusobne zavisnosti elemenata nekog niza. Ako, pa primer, obrađujemo niz reči i trebamo da znamo pravo značenje neke reči. To je nemoguće pez poznavanja prethodnih reči, jer su neophodne kako bi trenutnoj dale kontekst. Rekurentne neuronske mreže ovaj problem prevazilaze tako što ulazne elemente obrađuju u koracima, akumulirajući pritom informacije o prethodno obađenim elementima i kreirajući predikcije u odnosu na sve te raspoložive podatke.

3. TENSORFLOW

Slika 24: Tensorflow logo [24]

Mašinsko učenje je vrlo široka i komplikovana oblast čija implementacija u svakodnevnom životu može doneti razne benefite i pogodnosti. Tensorflow je open source framework za mašinsko učenje koji korisnicima omogućava korišćenje velikog broja algoritama i modela kroz zajedničke apstrakcije korišćenjem Tensorflow API-ja (aplikativni programski interfejs). Tensorflow pruža API u Python, C++, JavaScript i Java programskim jezicima[3], sa planom da se lista programskih jezika konstantno proširuje i da se ohrabruju korisnici da sami razvijaju API za neki drugi programski jezik, uz pomoć ekstenzivne dokumentacije koju Tensorflow pruža.

Iako je poželjno, za korišćenje Tensorflow framoworka za kreiranje i treniranje različitih modela mašinskog učenja nije potrebno predznanje iz oblasti mašinskog učenja i data science. API koji nam Tensorflow pruža omogućava korisnicima interakciju sa frameworkom na nekoliko nivoa, tako da je na najvišem nivou proces dovoljno uprošćen da korisnik ne mora da ima prethodno znanje o mašinskom učenju, a na najnižem nivou i dalje omogućava dovoljno fleksibilnosti i podešavanja detalja modela i algoritama da je koristan i za kompanije i profesionalce.

Tensorflow je razvio Google brain tim. Inicijalna verzija Tensorflow 1.0 je izašla 2015. godine, a druga i trenutna verzija Tensorflow 2.0 je izašla 2019 godine.

3.1 METODE TRENIRANJA MODELA

Treniranje modela predstavlja smanjenje njegove funkcije gubitaka na određenom setu podataka za treniranje. Treniranje se odvija u koracima, gde se u svakom koraku modelu prosledjuje serija podataka određene veličine sve dok model nije video sve podatke, kada se završava jedna „epoha". Broj epoha za treniranje modela zavisi od mnogo faktora, ali je najbolji trenutak da se treniranje prekine obično onaj kada model krene da se preprilagodjava podacima.

Koristeći tensorflow model se moeže trenirati na 3 različita načina:

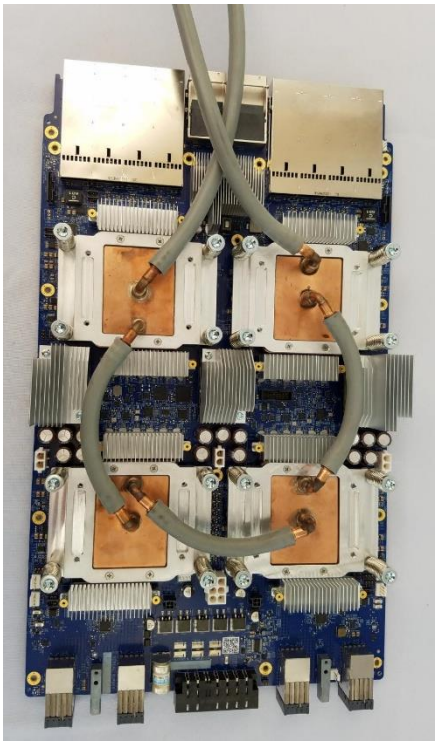
- Koristeći CPU

- Koristeći GPU
- Koristeći TPU

CPU je procesor računara. Radi na jako velikim frekvencijama i vrši serijsko procesiranje podataka. Pogodan je za treniranje manjih modela sa manjom serijom (batch) podataka. Dizajniran je tako da se ne specijalizuje ni za jedan zadatak, već je u većini zadataka podjednako dobar. Vrši desetine operacija po ciklusu.

GPU je grafička kartica računara i specijalizuje se u obradi 2D i 3D grafike. Znatno je sporiji od CPU, ali se specijalizuje u paralelnoj obradi podataka. Može brzo obraditi velike serije podataka i treniranje velikih modela je znatno brže na njemu nego na CPU. Model korišćen u ovom radu je treniran na GPU i u odnosu na CPU, trening se odvijao oko 10 puta brže. GPU može obavljati desetine hiljada operacija po ciklusu.

TPU (Tensor Processing Unit) Akselerator specijalne namene za mašinsko učenje (slika 25). On je procesorsko integrisano kolo koje je dizajnorao Google kako bi obavljao procesiranje neuronskih mreža koristeći TensorFlow.[25]



Slika 25: Cloud TPU V3 [25]

Tensorflow takođe podržava i distribuirano treniranje koje korisniku omogućava da po određenoj strategiji treniranja uz minimalne promene proces treninga raspodeli na više načina. Treniranje se može odvijati na više GPU na jednoj mašini, na više mašina unutar mreže ili u oblaku. Takože, trening može biti *asinhron* ili *sinhronizovan.* Kod sinhronizovanog treninga, ulazni podaci se dele na delove koji se daju različitim mašinama i u svakom koraku se rezultati treninga sa različitih mašina agregiraju. U slučaju asinhronog treniranja, sve mašine onoga trenutka kada završe sa korakom svoj rezultat treniranja obično upisuju u parametarski server.

Preneseno učenje (transfer learning) predstavlja poseban vid treniranja gde se za početni model ne uzima model sa nasumično dodeljenim parametrima, već se uzima model koji je već specijalizovan za rešavanje problema koji je sličan onome za koji naš model treba da se specijalizuje. Potom se treniranje vrši na samo zadnjih par slojeva modela, koji služe za

klasifikaciju, dok se ostali slojevi podešavaju u malim inkrementima (fine-tuning). Ovo omogućava da sa manjim skupom podataka i za mnogo manje vremena nego što bi inače bilo potrebno postignu rezultati treniranja komparativni onima dobijenim sa treniranjem celog modela od početka.[27]

Već trenirani modeli za preneseno učenje se mogu naći na tensorflow hub-u na adresi https://tfhub.dev/

3.2 EKSTENZIJE

Tensorflow.js – Biblioteka za mašinsko učenje u JavaScript programskom jeziku. Pruža sve potrebne API i pruža mogućnost pokretanja Tensorflow.js modela ili nekih drugih konvertovanih TensorFlow modela u okviru neke internet stranice.

TFX – TensorFlow Extended pored Usluga koje nam pruža TensorFlow API koje se tiču treniranja i validacije modela, nam pruža komponente koje pokrivaju ceo životni ciklus modela, od prikupljanja i transformacije ulaynih podataka, preko treniranja modela i optimizacije, pa sve do pokretanja modela u produkcijskom okruženju.

TFLite – TensorFlow Lite je biblioteka koja nam omogućava integraciju TensorFlow modela u mobilnu aplikaciju. Običan TensorFlow model se može konvertovati u TFLite, sa time da se pri konverziji neki aspekti modela uprošćavaju i žrtvuje se preciznost modela zarad mnogo boljih performansi.

3.3 KERAS

Keras je API za duboko učenje napisan u Python programskom jeziku, pokreće se nad platformom za mašinsko učenje TensorFlow. Razvijen je sa fokusam na omogućavanje brze eksperimentacije. *„Biti u mogućnosti da odeš od ideje do implementacije što brže je ključ dobrog istraživanja*"[26]

Keras (slika 26) je kreiran 2017. godine kao odvojen open source projekat koji je pružao API za komunikaciju sa mnoštvom frameworka (Tensorflow, Theano, PlaidML). Fokusirao se na brzo razvijanje modela dubokog učenja, sa minimalnim zalaženjem u detalje toga kako funkcioniše model koji se koristi. Od svoje verzije 2.4, Keras podržava isključivo Tensorflow framework.

Slika 26: Keras logo[26]

Od Tensorflow verzije 2.0, Keras dolazi kao standardni API za komunikaciju sa Tensorflow bibliotekom. Korisnici kojima treba veći nivo kontrole i dalje mogu pristupiti „starom" tensorflow API.Cilj Keras API je da korisnicima omogući brzo i lako treniranje modela dubokog učenja kroz brojne implementacije čestih pojmova i akcija, kao što su slojevi neuronske mreže, funkcije aktivacije, optimizatori, funkcije gubitaka, kao i mnoge druge funkcionalnosti, uz lako razumljivu i pristupačnu dokumentaciju i smernice za programere.

4. TRENIRANJE MODELA ZA DETEKCIJU SAOBRAĆAJNIH ZNAKOVA

Za treniranje modela korišćenog u ovom radu korišćen je Tensorflow, verzija 2.4.1

4.1 ODABIR MODELA

Kako bi se odabrao odgovarajući model, prvo je potrebno formulisati problem koji će taj model rešavati, kao i okruženja na kojima će se pokretati.

Prvo je bitno navesti da će se model koristiti na **android platformama** (telefonima i tabletima). Ono što ove platforme ograničava je procesorska moć, koja izuzetno varira od uređaja do uređaja. Sa obzirom da se radi o mobilnoj aplikaciji, model će morati da bude u **Tflite** formatu, jer on osigurava laku optimizaciju i korišćenje modela na mobilnom uređaju.

Pošto je detekcija objekata na slikama oblast koja se aktivno usavršava i na kojoj se konstantno radi, možemo koristiti već postojeći model, koji je samom svojom arhitekturom optimizovan za mobilne platforme i koristeći **preneseno učenje** istrenirati ga sa velikom uštedom vremena u odnosu na vreme koje bi bilo potrebno da se istrenira ceo model.

Pošto je za tip modela odabrali **Tflite** model, postoji nekoliko opcija:

- Obaviti trening standardnog Tensorflow modela i potom ga pretvoriti u Tflite model
- koristiti **Tflite model maker** biblioteku koja omogućava treniranje i konverziju modela sa značajno uprošćenim procesom treniranja.

4.1.1 TRENIRANJE TENSORFLOW MODELA I NAKNADNA KONVERZIJA U TFLITE

Tensorflow pruža veliki broj modela, u ovom slučaju specijalizovanih za detekciju objekata na slikama, koji su trenirani na COCO[1] datasetu. Za svaki model data je njegova srednja preciznost, izlazi koje daje i brzina izvršenja u milisekundama. Bitno je napomenuti da je izbor modela zbog kasnije konverzije u Tflite ograničen na samo SSD[2] modele, što izbor efektivno smanjuje na svega par modela. Bitno je napomenuti da ovom metodom takođe imamo pristup tensorboardu, pomoću koga je moguće prćenje grafovskog prikaza napretka modela usled treniranja, kao i da se prilikom treninranja kreiraju checkpointi, koji se, ukoliko se trening neočekivano zaustavi, mogu koristiti kako bi od te tačke nastavili treniranje.

4.1.2 TRENIRANJE MODELA KORIŠĆENJEM TFLITE MODEL MAKER BIBLIOTEKE

U oficijalnoj dokumentaciji [4] se Tflite model maker definiše kao „biblioteka koja koristeći *preneseno učenje* uprošćava proces treniranja TensorFlow Lite modela nad svojevrsnim setom podataka. Ponovnim treniranjem TensorFlow modela nad svojevrsnim setom podataka smanjujemo podatke i vreme potrebno za treniranje modela."

Ova biblioteka omogućava lakše učitavanje podataka, treniranje, konverziju, evaluaciju i snimanje modela nego prva metoda, sa žrtvovanjem mogućnosti za pristup nekim parametrima koje bi inače mogli da menjamo.

---

[1] COCO - (Common Objects in Context) je set za treniranje modela koji sadrži preko 330 000 slika, sa 15 000 000 instanci objekata podeljenih u 80 kategorija.

[2] SSD – (Single Shot Detector) model koji u samo jednom prolazu kroz sliku može detektovati više objekata na njoj

Izbor potencijalnih modela čine EfficientDet-Lite modeli koji se mogu koristiti za detekciju objekata na slikama. Kao i u prvom primeru, trenirani su na COCO datasetu i njihove performanse su zabeležene u tabeli.

4.1.3 ODABIR METODE TRENIRANJA

Iako nam obe metode nude modele trenirane na istom setu podataka*, tflite model maker* prikazuje performanse kvantizovanog modela koji se izvršava na telefonu[3], dok performanse modela vezane za modele iz *tensorflow model zoo* predstavljaju performanse nekvantizovanog modela, i nije specificirano na kom okruženju su merene, što znači da će realne performanse biti manje od onih prikazanih u tabeli, jer se kvantizacijom modela smanjuje kompleksnost, a samim tim i preciznost modela, zarad brzine.

Liste modela se mogu naći na sledećim linkovima:

Tensorflow:
https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

Tflite model maker:
https://www.tensorflow.org/lite/tutorials/model_maker_object_detection

U razmatranje su uzeti sledeći modeli: SSD MobileNet V2 FPN 640x640, SSD MobileNet V2 FPNLite 320x320, EfficientDet-Lite0, EfficientDet-Lite1 EfficientDet-Lite3, jer svojim karakteristikama predstavljaju mogući izbor za upotrebu u aplikaciji.

Modeli su, bez dodatnog treniranja, implementirani u probnu android aplikaciju (izvorni kod: https://github.com/tensorflow/examples) i kreirana su sledeća opažanja[4]:

Performanse modela rezolucije 320x320 su bile u proseku oko 100 milisekundi po slici. Povećavanjem rezolucije modela se preciznost pogađanja neznatno povećala, dok je vreme izvršenja 640x640 modela bilo čak 600ms, pa se manji model pokazao korisnije, jer iako ne može da primeti objekte na razdaljini na kojoj veći model može, samo ubrzanje od 6 puta čini da češće vidi objekat u pokretu.

Sva preostala 3 modela imaju skoro identičan kvalitet predikcija, sa time da je EfficientDet-Lite1 u proseku 80 milisekundi sporiji.

Nakon oprobavanja svih modela, za dalji rad odabran je **EfficientDet-Lite0,** jer pored preciznosti i performansi komparativnih sa SSD MobileNet V2 FPNLite 320x320 modelom, omogućava treniranje korišćenjem **Tflite model maker** biblioteke, koja će znatno olakšati proces treniranja.

[3] Performanse se mere na telefonu pixel 4, sa omogućene 4 niti
[4] Opažanja su vršena na Honor 8x mobilnom telefonu, sa 4 niti

4.2 PRIBAVLJANJE I PRIPREMA SETA PODATAKA ZA TRENIRANJE

Početni i najbitniji korak u mašinskom učenju predstavlja kreiranje upotrebljivog i korisnog seta podataka za treniranje i validaciju. Podaci moraju u što većoj meri biti slični realnim podacima nad kojima će model vršiti predikciju. Veliki broj faktora može uticati na kasnije performanse modela, kao što su ***osvetljenje, rezolucija, veličina objekta, orjentacija***

Pribavljanje seta podataka za treniranje i samo treniranje se u više navrata prepliću, kako bi se dokazala prednost određenog pristupa treniranju.

4.2.1 PRIBAVLJANJE SLIKA

Set podataka za treniranje modela korišćenog u ovom radu se sastoji od slika koje su nađene u različitim već postojećim skupovima podataka za treniranje, skinute sa interneta ili ručno slikane. Slike su različite rezolucije, uglavnom 720x720 piksela.

Ukupno, u setu se nalaze 4012 slike.

Slike su dobijene tako što su se snimali individualni kadrovi, koji su prvobitno bili deo snimka vožnje niz ulicu.

Slika 27: Primer slike iz seta podataka

4.2.2 OBELEŽAVANJE SLIKA

Pošto je reč o nadgledanom učenju, svi znaci na slikama moraju biti obeleženi.

Postoji puno različitih formata(notacija) obležavanja slika. U ovom primeru korišćen je Pascal VOC format. U ovom formatu, podaci o objektima na slikama su sadržani u XML fajlovima i za svaku sliku postoji odgovarajući XML fajl koji opisuje objekte na njoj.

```
<annotation>
        <folder>cropped</folder>
        <filename>28.jpg</filename>
        <path>C:\Users\iseguljev\Desktop\Master\Datasets\stop_signs\cropped\28.jpg</path>
        <source>
                <database>Unknown</database>
        </source>
        <size>
                <width>506</width>
                <height>507</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>stop</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>85</xmin>
                        <ymin>182</ymin>
                        <xmax>126</xmax>
                        <ymax>223</ymax>
                </bndbox>
        </object>
</annotation>
```

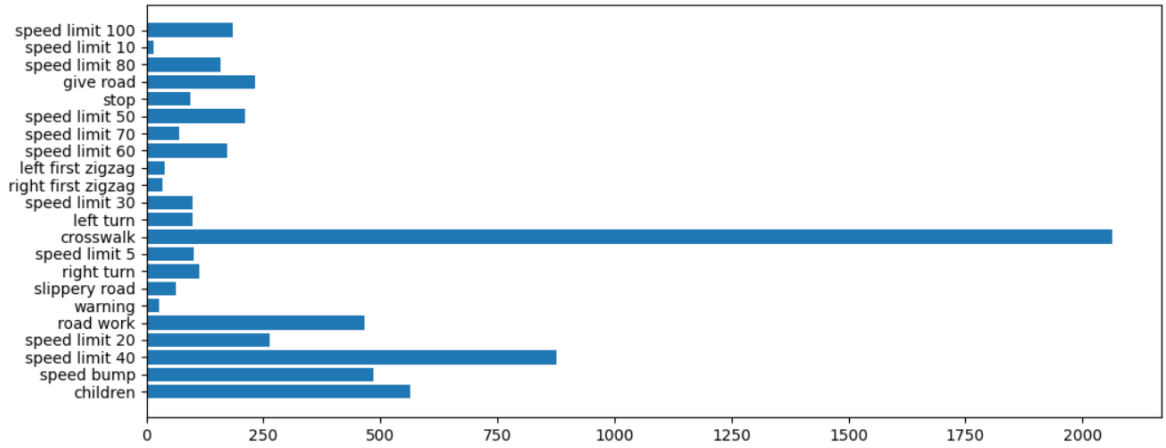
Slika 28: Primer Pascal VOC notacije za sliku 1

Na slikama su obeležavani svi znaci zabrane, ograničenja i opasnosti, kao i znakovi za pešački prelaz i prepreku za usporavanje saobraćaja.

Najpre su obeležene 22 klase znakova. Ovako dobijeni set je nebalansiran i broj instanci nekih znakova je isuviše mali da bi model naučio njegovu reprezentaciju. Iako **EfficientDet-lite0** model koristi funkciju ocenjivanja koja u odnosu na reprezentaciju klase u nekom setu podataka toj klasi pridaje veću ili manju vrednost[5], i time anulira efekte nebalansiranosti seta podataka, postoji mogućnost da će rezultati treniranja biti nezadovoljavajući.

Kako bi se potvrdila ova sumnja, najpre je nad ovim setom treniran EfficientDet-Lite0 model. Treniranje se odvijalo u 70 epoha.

[5] „Focal loss" funkcija gubitaka



Slika 29: Raspodela klasa u prvobitnom setu podataka

Prilikom validacije modela, srednja preciznost (Apm) je bila **0,175**, što znači da se u **17.5%** slučajeva vrednost koju je predvideo model podudarala sa očekivanom vrednošću. Pored nezadovoljavajuće prosečne preciznosti, preciznost pogađanja određenih klasa je, zbog nebalansiranosti seta podataka, bila ravna nuli.

'APm': 0.17537244
'AP_/crosswalk': 0.39831614,
'AP_/children': 0.24980427,
'AP_/warning': 0.0,
'AP_/left first zigzag': 0.0,
'AP_/right first zigzag': 0.0,
'AP_/left turn': 0.003379538,
'AP_/right turn': 0.000990099,
'AP_/slippery road': 0.0,
'AP_/road work': 0.18723463,
'AP_/give road': 0.4033809,
'AP_/stop': 0.0045655826,
'AP_/speed bump': 0.13780917,
'AP_/speed limit 5': 0.024752475,
'AP_/speed limit 10': 0.0,
'AP_/speed limit 20': 0.0052794926,
'AP_/speed limit 30': 0.0010714164,
'AP_/speed limit 40': 0.26305133,
'AP_/speed limit 50': 0.004819849,
'AP_/speed limit 60': 0.008414492,
'AP_/speed limit 70': 0.00050290744,
'AP_/speed limit 80': 0.0036105472,
'AP_/speed limit 100': 0.48914173

Tako, na primer, preciznost pri pogađanju znaka za pešački prelaz je 39.8%, dok je preciznost pri pogađanju, na primer, znaka za skretanje u levo svega 0.3%. Znak za pešački prelaz se u setu podataka pojavljuje preko 2000 puta, dok se znak za skretanje desno pojavljuje manje od 100 puta.

Dobijene performanse modela su nezadovoljavajuće i neophodno ih je poboljšati kako bi model bio koristan.

4.2.3 AGREGACIJA KLASA

Kao prvi korak u unapredjivanju seta podataka za treniranje vršiće se agregacija sličnih klasa.

Ovim korakom rešava se više problema koji su u inicijalnom setu podataka bili prisutni, a to su:

1. Zbog malog broja pojavljivanja nekih znakova u skupu podataka, model nije bio u mogućnosti da nauči reprezentaciju tih znakova. Agregacijom će se slični znakovi spojiti u jednu klasu, što će poboljšati balansiranost skupa podataka.
2. Zbog niske rezolucije modela, detalji koji su potrebni za razlikovanje sličnih znakova gube se u slojevima konvolucije i agregacije.

Tako da su u ovom koraku klase koje su postojale u prvobitnom setu spojene u sledeće klase, čiji se primeri mogu videti na slici 30:

1. warning_ahead – Trouglasti znakovi, koji upozoravaju na potencijalnu opasnost na predstojećoj deonici puta. Svi znakovi su trouglog oblika sa crvenim obodom i upozorenjem u sredini.

2. mandatory – Okrugli znakovi koji uglavnom izriču maksimalnu dozvoljenu brzinu, maksimalnu visinu ili širinu vozila ili zabranu skretanja. Okruglog oblika, sa crvenom ivicom i zabranom u sredini.

3. no_stopping_or_parking- Znak za zabranjeno zaustavljanje i parkiranje. Plave boje sa crvenim obodom i precrtan sa jednom ili 2 linije

4. warning_on_spot- Znak upozorenja na pešački prelaz ili prepreku za usporavanje saobraćaja, koji stoji na samom mestu prepreke. Kvadratnog oblika, plave boje, sa belim trouglom u sredini unutar koga se nalazi upozorenje

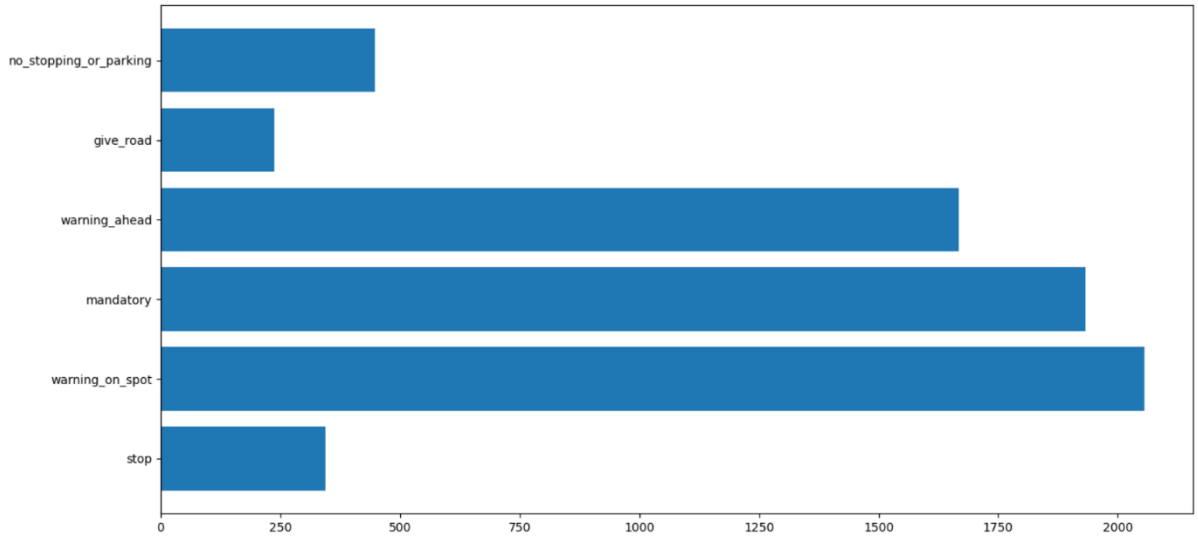
5. stop- Obavezno potpuno zaustavljanje vozila pre uključenja u put sa pravom prvenstva prolaza. Šestougaonog oblika, crvene boje sa belim STOP u sredini.

6. give_road- Upozorenje na uključivanje u kolovoz sa pravom prvenstva. Obrnuti trougao sa crvenim rubom.

Slika 30: Reprezentativni primeri klase znakova.

Nakon agregacije klasa znakovi različite klase se razlikuju po obliku ili boji, koji su dominantne i lako uočljive komponente, tako je očekivan značajan pomak u preciznosti modela..

Nakon agregacije raspodela klasa izgleda:



Slika 31: Raspodela klasa nakon agregacije

Set je bolje izbalansiran nego u početku i iako je broj instanci klasa *give_road, stop, no_stopping_or_patking* manji nego ostalih, razlika između klasa je veća i uz pomoć **focal loss** funkcije gubitaka, očekuje se da model uspešno nauči reprezentaciju svake klase.

'APm': 0.553289
'AP_/warning_ahead': 0.41741055,
'AP_/mandatory': 0.4638313,
'AP_/no_stopping_or_parking': 0.49355558,
'AP_/warning_on_spot': 0.45593295,
'AP_/stop': 0.23495442,
'AP_/give_road': 0.32368103

Metodom agregacije klasa, prosečna preciznost ja sa **17.5%** porasla na **55.3%,** što predstavlja izuzetan porast. Preciznost pogađanja klasa **stop** i **give_road** je i dalje nezadovoljavajuća, ali znatno bolja od preciznosti pogađanja bez agregacije klasa.

4.2.4 ISECANJE SLIKA

Daljim istraživanjem seta podataka primećeno je da je većina znakova premale veličine kako bi ih model detektovao na slici.

Većina slika je formata 720x720. Kako bi model koji kao ulazni podatak prima sliku rezolucije 320x320 mogao tu sliku da koristi, on prvo mora da je smanji na odgovarajuću veličinu. Slika formata 720x720 ima 5 puta više piksela nego slika formata 320x320(518 400 px i 102 400 px) i samim tim će se za 4 puta smanjiti količina informacija koju neki objekat na toj slici nosi.

Inicijalna pomisao je ta da će se isecanjem slika dobiti bolji rezultati jer će se set podataka više prilagoditi modelu, ali je bitno napomenuti **da ukoliko neki objekat nije prikazan sa dovoljno piksela, nakon konvolucije postaje nemoguće detektovati ga** i previše takvih objekata može pogubno uticati na sigurnost modela pri detekciji istih.

Isecanju slika je pristupljeno programski, pomoću python skripte koja isecanje radi u sledećim koracima:

1. Čitanje pozicija znakova na slici iz odgovarajućeg fajla

2. Učitavanje slike (slika 32)
3. Deljenje slike na 4 dela i rečunanje novih koordinata znakova u odnosu na deo slike na kome se nalaze (slika 33)
4. Deljenje znakova na više delova ukoliko se nalaze na granici između 2 ili više dela slike
5. Snimanje onih delova slike u kojima se nalazi barem jedan znak, kao i snimanje fajlova sa koordinatama znakova.(slike 34 i 35)

Pored očiglednog smanjenja rezolucije pojedinačnih slika, ovom metodom postignuto je i smanjenje seta podataka, jer su svi delovi slika na kojima se ne nalaze znakovi odbačeni. Inicijalna veličina trening seta podataka je **325MB**, a veličina nakon isecanja je **137MB**, što omogućuje kraće vreme treniranja.

Slika 32: Početna slika

Slika 33: Podela na 4 jednaka dela

Slika 34: Odbacivanje delova slike koji ne sadrže znak

Slika 35: Krajnji rezultat

Korišćenjem ove metode, dobijeni su sledeći rezultati treniranja:

'APm': 0.71674806,
'AP_/warning_ahead': 0.59631634,
'AP_/mandatory': 0.62884533,
'AP_/no_stopping_or_parking': 0.6265325,
'AP_/warning_on_spot': 0.538075,
'AP_/stop': 0.6941571,
'AP_/give_road': 0.5878895

Korišćenjem ovog seta podataka, prosečna preciznost je porasla na **71.6%** i prosečna preciznost predikcije svake klase je iznad **50%**, što je zadovoljavajuće i model ovih performansi se može koristiti pri izradi aplikacije.

4.3 TRENIRANJE MODELA

Treniranje modela predstavlja obavljanje procesa nadgledanog učenja nad konvolutivnom neuronskom mrežom. Kvalitet krajnjeg zavisi najviše od toga koliko je model mašinskog učenja podoban za rešavanje datog problema i kvalitet podataka za trening. Veliki faktor u treniranju modela mašinskog učenja čini i okruženje, jer proces zahteva izuzetno veliku brzinu paralelne obrade podataka i okruženje u kome se trenira model mora posedovati hardver adekvatnih performansi kako bi se treniranje odvijalo prihvatljivom brzinom.

4.3.1. ODABIR OKRUŽENJA ZA TRENIRANJE

Treniranje modela mašinskog učenja zahteva veliku računarsku moć kako bi se sa dovoljno velikim setom podataka uspešno trenirao model zadovoljavajućih performansi.

Kako bi se obavio odabir odgovarajućeg okruženja za treniranje, proces treniranja je pokretan na svakom od okruženja i rezultati su navedeni u nastavku.

Model je treniran u 70 epoha, ali nije uvek neophodno da se trening završi kako bi se zanlo očekivano vreme završetka.

1. **Treniranje lokalno na CPU**[6]: Vreme treniranja preko 48 sati. Procesor nije optimalno rešenje za obradu slika, jer je on specijalizovan za brzu sekvencijalnu obradu, a ne paralelnu. Još jedan problem je predstavljalo to što se treniranje odvijalo na laptopu, čije rashladno rešenje nije adekvatno za dugotrajno maksimalno opterećenje, performanse nakon nekog vremena degradiraju kao rezultat smanjenja radnog takta procesora u cilju smanjenja temperature.
2. **Google Colaboratory**: Cloud okruženje koje korisnicima pruža korišćenje Nvidia Tesla K80 GPU, kao i direktan pristup fajlovima na google drive. Korišćenje okruženja je besplatno, ali veliku prepreku predstavlja to što sesija može biti isključena u bilo kom trenutku zbog neaktivnosti ili jer predugo traje i prava na korišćenje GPU mogu biti oduzeta na neodređeni period, od 24 sata, do nekoliko nedelja.
3. **Paperspace gradient**: Cloud okruženje koje nam pruža korišćenje Quadro m4000 GPU besplatno, sa mogućnošću iznajmljivanja bržeg GPU uz doplatu. Performanse besplatnog okruženja, zbog zastarele grafičke kartice, koja je izašla 2015. godine, nisu dobre i maksimalno vreme sesije je 6 sati, što nije uvek dovoljno.
4. **Kaggle**: Od svih okruženja, kaggle je za ovaj slučaj korišćenja najpodobniji. Pruža moćne Nvidia Tesla P100 GPU i umesto ograničavanja sesije, kaggle ograničava koliko vremena mesečno korisnik može da koristi GPU, pa samim tim nema iznenadnog prekida sesija. Takođe, Kaggle omogućava lako korišćenje javnih setova podataka koje su postavili drugi korisnici, olakšavajući time proces kreiranja i traženja seta podataka.

Za potrebe treniranja modela korišćenog u ovom radu, korišćena je platforma **Kaggle** (slika 36)**,** jer je pored dobrih performansi okruženja pružala sesije koje će se, ukoliko je dostupno dovoljno vremena na GPU, sigurno izvršiti. Takođe, laka integracija i verzionisanje seta podataka znatno olakšava ceo proces treniranja.

[6] Intel I7 8550U laptop procesor

Slika 36: kaggle logo

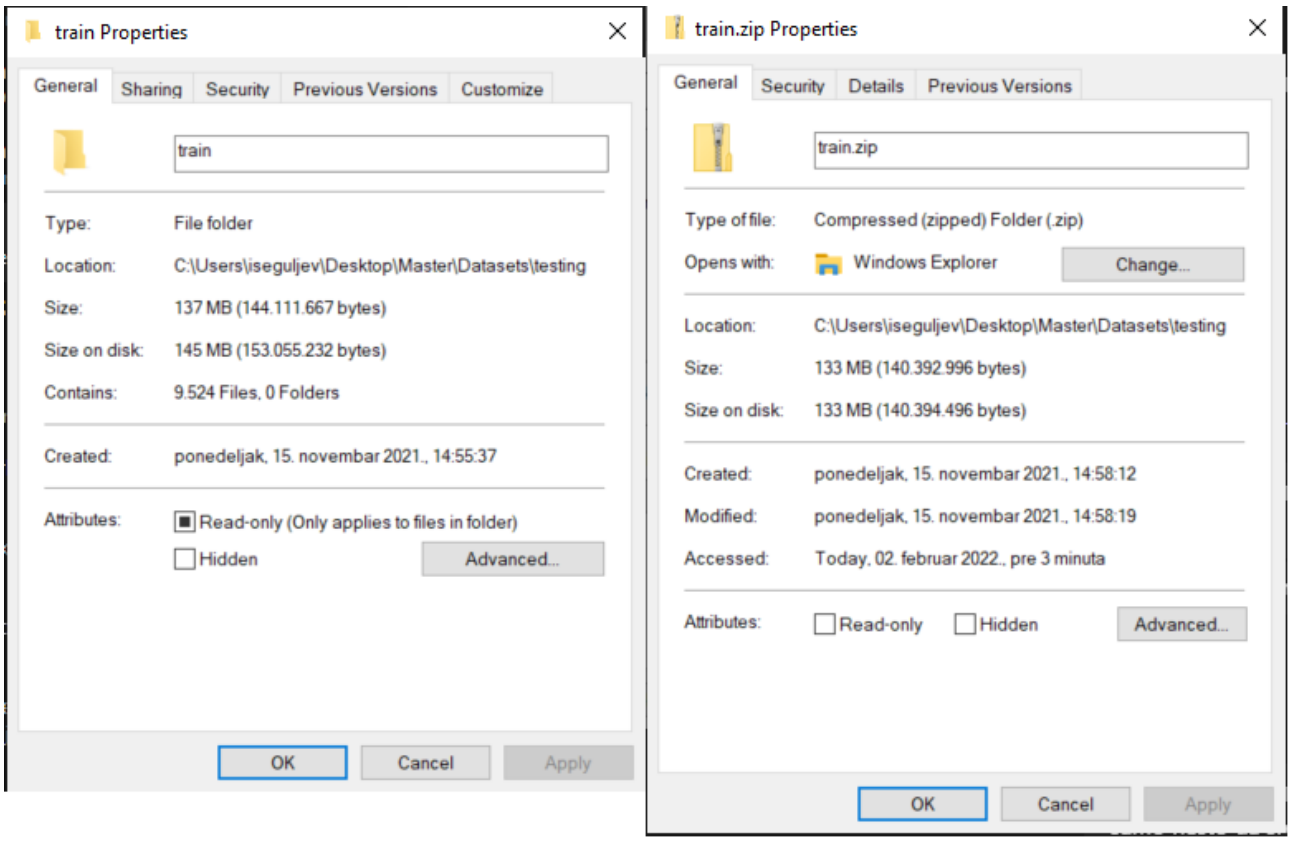
4.3.2. UPLOAD SKUPA PODATAKA ZA TRENING

Kao prvi korak ka treniranju modela mašinskog učenja neophodno je da na okruženju na kome će se treniranje obavljati bude dostupan set podataka za trening tog modela.

Kaggle korisnicima omogućuje upload i korišćenje svojih, kao i korišćenje tuđih skupova podataka, koji se na toj platformi nalaze.

Prvi korak pre uploada podataka predstavlja priprema skupa podataka za upload. Bitan korak u ovoj fazi predstavlja komprsija i podataka, kako bi se njihova ukupna veličina smanjila, ali i zbog činjenice da je jedan veliki fajl brže uploadovati nego puno manjih, ukoliko im je zbirna veličina ista.

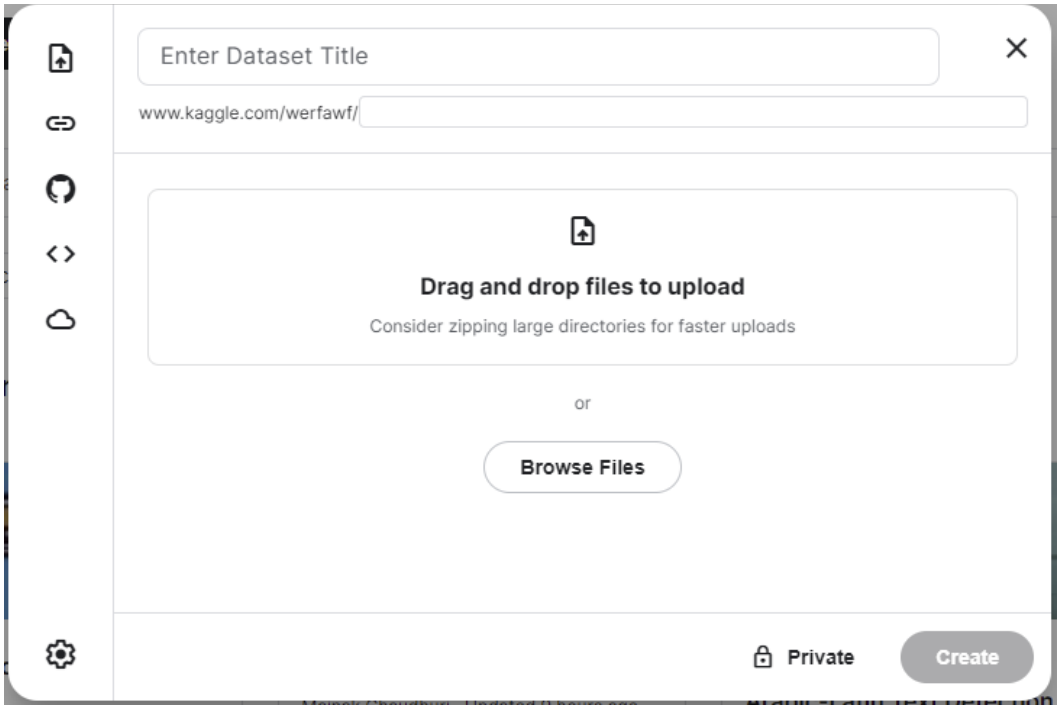
Kaggle podržava upload fajlova u .zip formatu i na slici 37 možemo videti da je kompresijom sa 137 megabajta, veličina smanjena na 133 megabajta, ali još bitnije, umesto 9524 fajlova koliko je sadržao trening skup, biće uploadovan samo jedan.



Slika 37: nekompresovani i kompresovani podaci

Nakon kompresije podataka, treba napraviti nalog na kaggle.com kako bi upload dataseta bio omogućen.

Ukoliko je nalog kreiran i korisnik ulogovan, odabirom opcije za kreiranje novog seta podataka otvara se dijalog za upload seta podataka, gde se učitavaju podaci i setu podataka dodeljuje ime (slika 38).



Slika 38: Dijalog za upload dataseta

U polje za naziv skupa podataka se unosi željeni naziv, a pretragom ili prevlačenja prethodno kompresovanih test i trening podataka pokreće se upload tih podataka na server.

Nakon što su fajlovi uspešno uploadovani na server, klikom na dugme „Create" (slika 38) podaci se ekstraktuju iz arhiva u koje su bili prvobitno zapakovani i kreira se skup podataka koji je moguće koristiti u svim projektima na kaggle platformi.

4.3.3. KREIRANJE SVESKE ZA TRENIRANJE MODELA

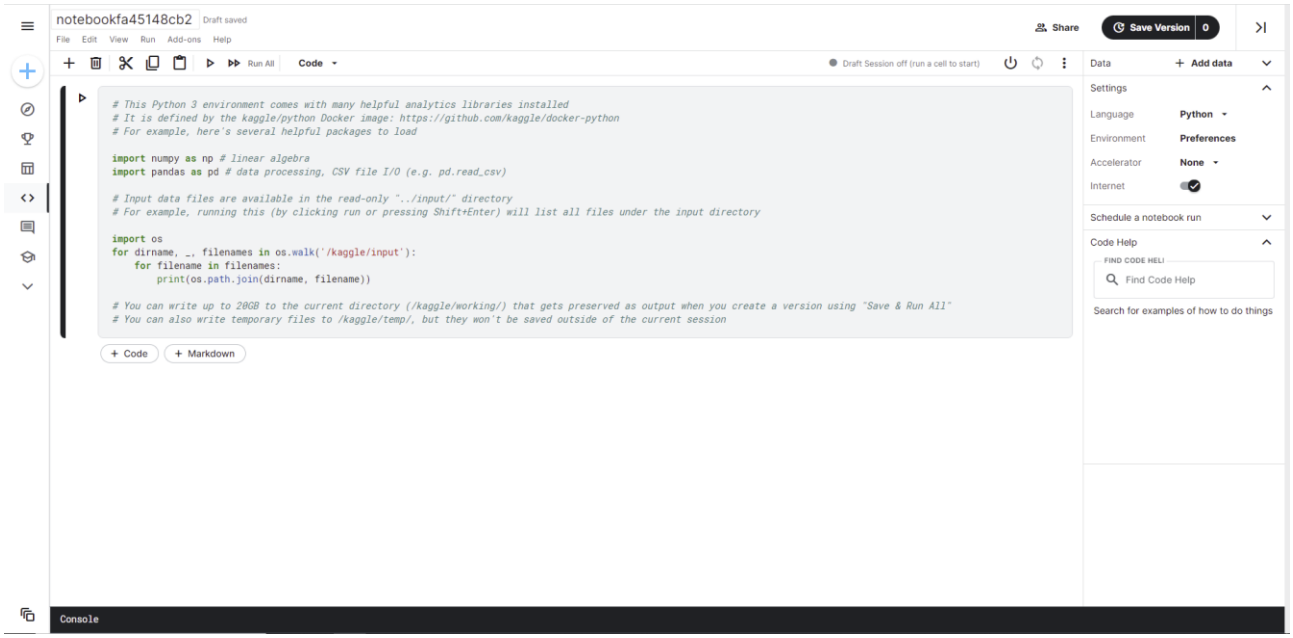
Kaggle sveske (notebooks) predstavljaju Jupyter sveske koje se sastoje od zasebnih blokova koda, koji se nezavisno jedan od drugog mogu pokrenuti. Nakon pokretanja nekog od blokova koda, izlaz koji bi se inače video u konzoli se iscrtava direktno ispod bloka, što značajno olakšava pronalaženje eventualne greške.

U kaggle ekosistemu platforma na kojoj se izvršavaju jupyter sveske se naziva „kaggle kernels". Velika prednost korišćenja kaggle kernela je to što „možemo poštedeti sebe problema podešavanja lokalnog okruženja, i imati Jupyter svesku i okruženje unutar pretraživača, bilo gde u svetu gde imamo internet konekciju"[30]

Svaka sveska se pokreće na oblaku i prilikom pokretanja sesije, svesci su dodeljeni resursi kojima raspolaže. Za svesku koja kao akselerator koristi GPU, uglavnom se dodeljuju:

- 1 procesorsko jezgro[7]
- 16 gigabajta RAM memorije
- Tesla p100 grafička kartica sa 16GB memorije

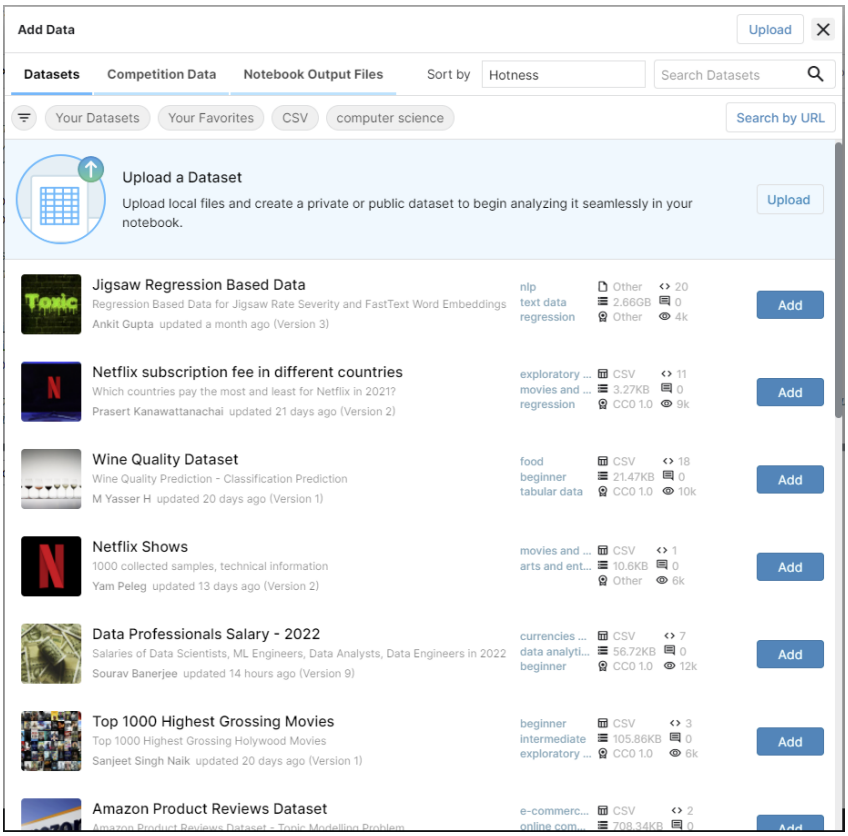
Odabirom opcije „new notebook", kreiramo novu svesku. Radno okruženje je prikazano na slici 39.



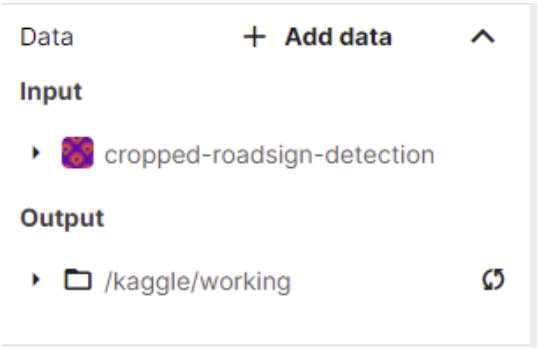
Slika 39: Kaggle radno okruženje

Najpre je potrebno povezati svesku sa setom podataka koji će koristiti. To je moguće uraditi klikom na opciju „add data" u „data" segmentu radnog okruženja, nakon čega se otvara dijalog prikazan na slici 40. Odabirom filtera „Your Datasets" prikazuju se skupovi podataka koje je kreirao trenutno prijavljeni korisnik. Biranjem opcije „Add" pored skupa podataka koji je potrebno povezati sa kernelom, taj skup se dodaje kao izvor podataka. U slučaju uspešnog dodavanja, skup podataka bi trebao biti prikazan u „data" segmentu kao na slici 41.

[7] intel xeon procesor radne frekvencije 2.00GHz.



Slika 40: Dijalog za povezivanje skupa podataka sa kaggle kernelom



Slika 41: Uspešno dodat skup podataka

Nakon povezivanja skupa podataka sa kaggle kernelom, sledeći korak je podešavanje okruženja. Kaggle kernel dolazi sa već podešenium python 3 okruženjem, kao i Tensorflow bibliotekom.

Najpre je neophodno instalirati sve neophodne pakete kako bi se treniranje modela moglo izvršiti. Ovi paketi su tensorflow lite model maker, koji će biti korišćen za treniranje modela i pycoco tools, koji je neophodna zavisnost pri evaluaciji modela pomoću tensorflow lite model makera. Pozivanjem komandi sa slike 42 osigurava se instalacija prthodno navedenih paketa.

```
!pip install tflite-model-maker==0.3.0
!pip install pycocotools
```

Slika 42: instalacija neophodnih paketa

Nakon instalacije paketa potrebno je importovati sve klase koje će biti korišćene u daljem radu (slika 43). U ovom koraku se obavlja i provera da li je Tensorflow okruženje verzija 2 ili novija, a ukoliko nije, izvrženje programa će se obustaviti.

```
from tflite_model_maker.config import ExportFormat
from tflite_model_maker import model_spec
from tflite_model_maker import object_detector
import os
import random
import shutil
import tensorflow as tf
assert tf.__version__.startswith('2')
```

Slika 43: Importovanje neophodnih klasa i provera verzije tensorflow okruženja

Sledeći korak (slika 44) predstavlja učitavanje klasa znakova koje će model biti u stanju da detektuje u vidu mape koja svakoj klasi dodeljuje odgovarajući broj.

```
labelmap={
    1: "warning_ahead",
    2: "mandatory",
    3: "no_stopping_or_parking",
    4: "warning_on_spot",
    5: "stop",
    6: "give_road",
}
```

Slika 44: Učitavanje klasa

U narednom bloku, definišu se promenljve koje opisuju model koji se trenira (slika 45). Promenljiva „spec" definiše specifikaciju modela koji će biti treniran, „model_name" definiše ime koje će se modelu dodeliti pri snimanju.

```
spec = model_spec.get('efficientdet_lite0')
model_name="trained_model.tflite"
```

Slika 45: promenljive neophodne za dalji proces treniranja

Naradni korak predstavlja učitavanje seta podataka. Tflite model maker biblioteka poseduje već kreirane funkcije za učitavanje podataka za treniranje i pošto su podaci već raspodeljeni u trening i test podatke pre uploadovanja, pozivom funkcije *object_detector.DataLoader.from_pascal_voc* se može obaviti učitavanje podataka. Kao parametre, ova funkcija prima:

1. *Putanja ka direktorijumu sa slikama*
2. *Putanja la direktorijumu sa anotacijama*
3. *Mapu klasa*

Podacma povezanim sa sveskom se može pristupiti tako što se iz direktorijuma pre onog u kome se nalazi sveska zalazi u direktorijum koji se zove isto kao skup podataka. Nakon toga, raspored fajlova i direktorijuma je isti kao u samom skupu podataka. Na slici 45 prikazan je poziv funkcije za učitavanje podataka.

```
train_data = object_detector.DataLoader.from_pascal_voc("../input/cropped-roadsign-detection/train", "../input/cropped-roadsign-detection/train", labelmap)
validation_data = object_detector.DataLoader.from_pascal_voc("../input/cropped-roadsign-detection/test", "../input/cropped-roadsign-detection/test", labelmap)
```

Slika 45: Učitavanje podataka u trening podatke i podatke za validaciju

Naredni korak predstavlja treniranje modela. Tflite model maker omogućuje treniranje modela samo pozivom funkcije object_detector.create (slika 46). Parametri ove funkcije su:

1. **train_data** – Promenljiva u koju su pomoću from_pascal_voc funkcije učitani podaci za trening (slika 45).
2. **validation_data** - Promenljiva u koju su pomoću from_pascal_voc funkcije učitani podaci za validaciju (slika 45).
3. **model_spec –** Specifikacija modela koji će biti treniran (slika 45)
4. **epochs –** Broj epoha koje će model biti treniran. U ovom primeru, ustanovljeno je da je broj epoha za koji se dobiju adekvatni rezultati 70.
5. **batch_size –** Veličina serije čiji rezultati detekcije će se uzimati u obzir pre procesa optimizacije. Odabrana serija veličine 8.
6. **do_train –** ukoliko se vrednost ove promenljive postavi na *False*, trening se neće obaviti. Ova funckija je izuzetno korisna ukoliko je potrebno testirati neku funkcionalnost koja se tiče, na primer, snimanja modela, jer nije potrebno čekati da se trening završi. Za potrebe rada, ova prmomenljiva se postavlja na *True* jer je cilj treniranje modela
7. **train_whole_model –** Ukoliko je postavljena na *False,* treniranje će se vršiti samo na sloju klasifikacije, dok će ostali slojevi ostati nepromenjeni. Tada će trening biti brži ali model kojim taj proces treniranja rezultuje će biti manje precizan. Opcija *True*, koja je u ovom radu i korišćena menja model tako da slojeve klasifikacije ponovno trenira, dok nad ostalim slojevima vrši „fine tuning".

```
model = object_detector.create(train_data=train_data, model_spec=spec, epochs=70, batch_size=8, do_train=True, train_whole_model=True,validation_data=validation_data)
```

Slika 46: treniranje modela

Kreira se zaseban direktorijum u koji će biti snimani model kao i svi ostali fajlovi koji su rezultat treniranja modela (slika 47).

```
if not os.path.exists("./model_out"):
  os.mkdir("./model_out")
```

Slika 47: Kreiranje direktorijuma u kome će biti snimljen model

Nakon obavljenog treniranja model je neophodno evaluirati, kako bi proverili njegove performase nakon treniranja (slika 48). Rezultati evaluacije se snimaju u isti direktorijum u kome je i model, kako bi ukoliko imamo više modela znali na koji se odnose koji rezultati evaluacije.

```
evaluation = model.evaluate(validation_data, batch_size=3)
with open("./model_out/" + model_name+"_evaluate.txt", "w") as f:
    f.write(str(evaluation))
```

Slika 48: Evaluacija modela i snimanje rezultata u fajl

Finalni korak u kreiranju sveske za treniranje modela predstavlja snimanje modela nakon treniranja (slika 49). Ova funkcionalnost se postiže pozivanjem funkcije *model.export*, kojoj se prosleđuju direktorijum u koji će snimljeni model biti sačuvan i formati u kojima će model biti snimljen. Za potrebe rada, odabrani formati su .TFLITE i .LABEL, tflite jer će model biti korišćen u android aplikaciji, a .LABEL opcija definiše da je potrebno snimiti i fajl sa mapom klasa, sličan mapi sa slike 44.

```
model.export(export_dir='./model_out', export_format=[ExportFormat.SAVED_MODEL, ExportFormat.LABEL, ExportFormat.TFLITE], tflite_filename=model_name)
```

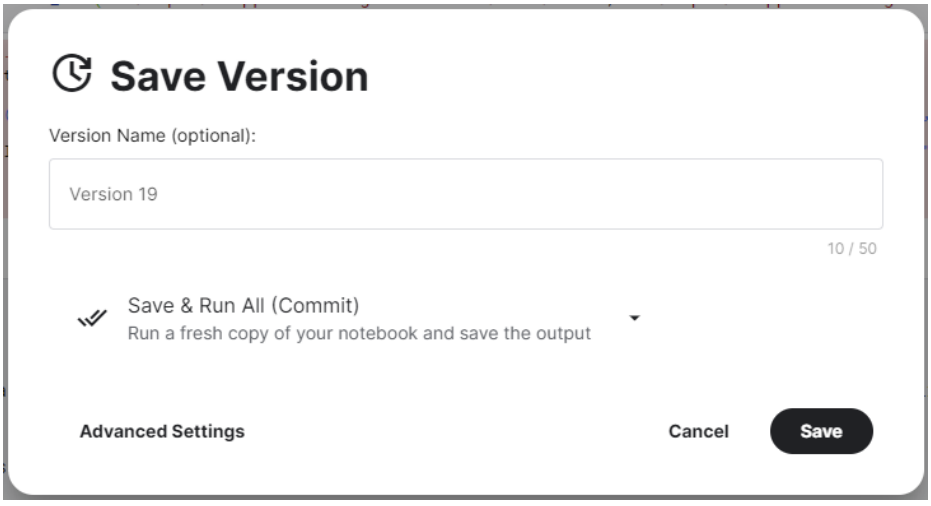
Slika 49: Snimanje treniranog modela

4.3.4 POKRETANJE KAGGLE KERNELA

Nakon kreiranja sveske, neophodno je proveriti da li su sve komande unete pravilno. To je najlakše uraditi postavljanjem *do_train* promenljive u koraku za treniranje modela na False, kako ne bi bilo neophodno čekati da se završi treniranje modela. Odabirom opcije „Run all" pokreću se svi blokovi koda redom, od gore na dole i u realnom vremenu se mogu videti rezultati izvršavanja svakog od njih.

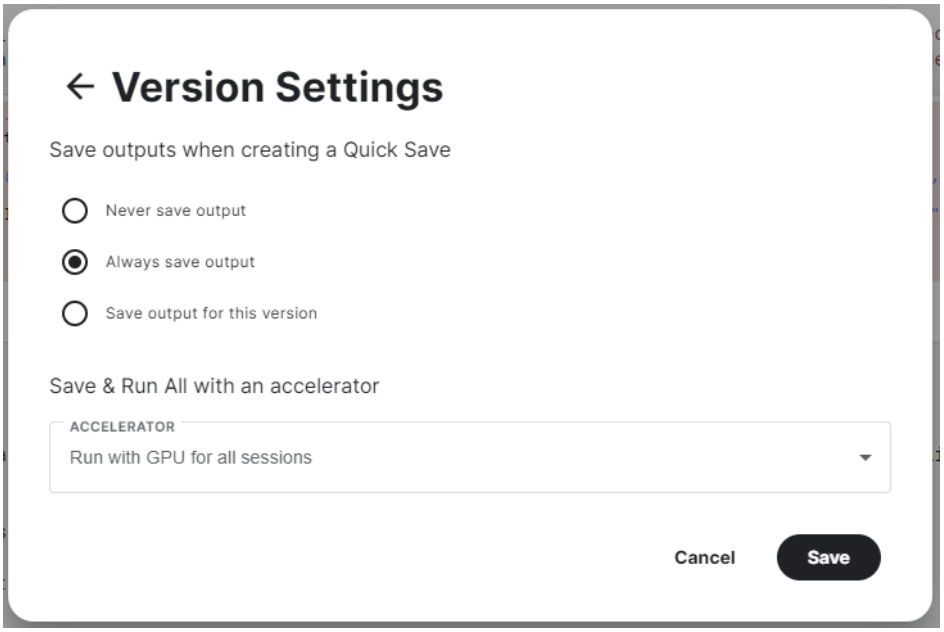
Ukoliko je izvršenje sveske uspešno, nastavlja se na sledeći korak, koji predstavlja pokretanje treniranja modela. promenljivu *do_train* treba postaviti na true.

Klikom na opciju „Save Version" se otvara dijalog za snimanje nove verzije sveske(slika 50). Pri snimanju verzije, izlazi svih ćelija, kao i snimljeni podaci bivaju sačuvani, a najveću prednost predstavlja to što nije potrebno da sveska bude otvorena u prozoru pretraživača, već se pokreće odvojeno od interaktivne sesije.



Slika 50: Dijalog za snimanje verzije

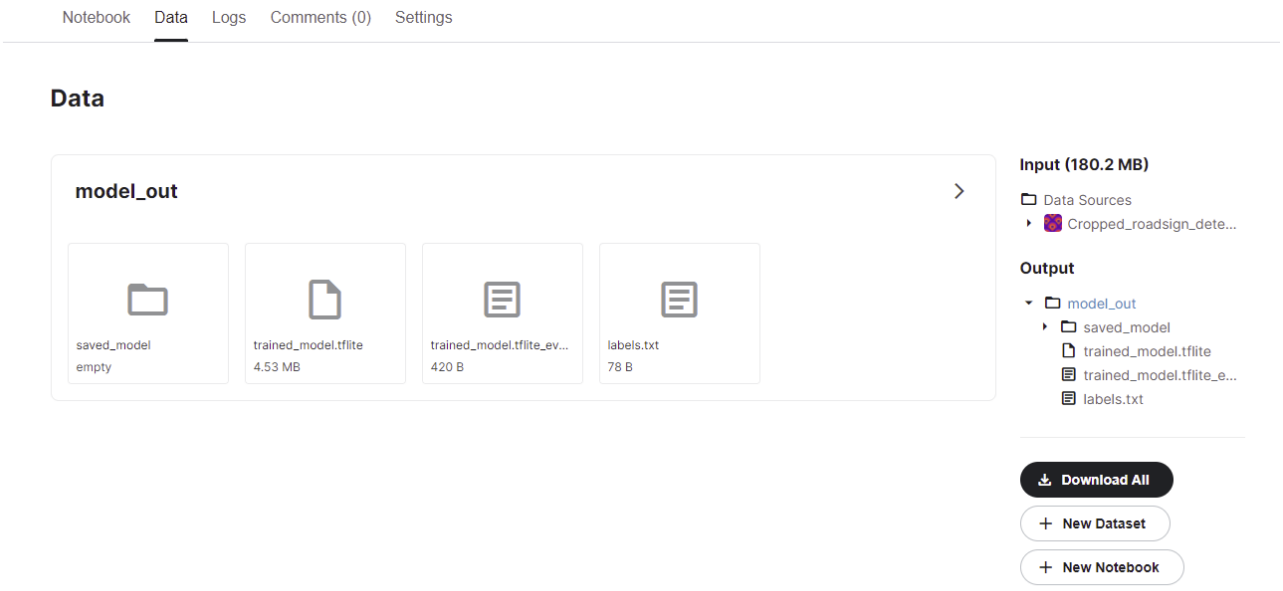
U dijalogu sa slike 50 najpre je neophodno odabrati opciju „Advanced Settings" i osigurati da je za opcije snimanja izlaza selektovana opcija „Always save output", a kao akselerator da se koristi GPU. Prikaz podešavanja se može videti na slici 51.



Slika 51: Podešavanje parametara za snimanje verzije

Ukoliko su svi parametri podešeni, odabirom opcije „Save" sveska se pokreće. U donjem levom uglu radnog okruženja (slika 39) mogu se videti trenutno pokrenut kaggle kerneli, kao i njihovi statusi.

Nakon završetka treniranja, odlaskom na snimljenu svesku mogu se pregledati rezultati izvršenja sveske pri snimanju, a odlaskom na sekciju „Data" (slika 52) je moguć pregled svih snimljenih podataka. Odabirom opcije „Download all" preuzimamo trenirani model mašinskog učenja, kao i sve ostale dokumente koje smo snimili prilikom pokretanja sveske.



Slika 52: Data sekcija nakon treniranja modela

5. IZRADA ANDROID APLIKACIJE

Samo treniran model mašinskog učenja, iako obavlja zadatak za koji je namenjen, korisnicima nije koristan bez aplikacije koja ceo taj model predstavlja na taj način da su njegovi izlazi upotrebljivi korisniku te aplikacije.

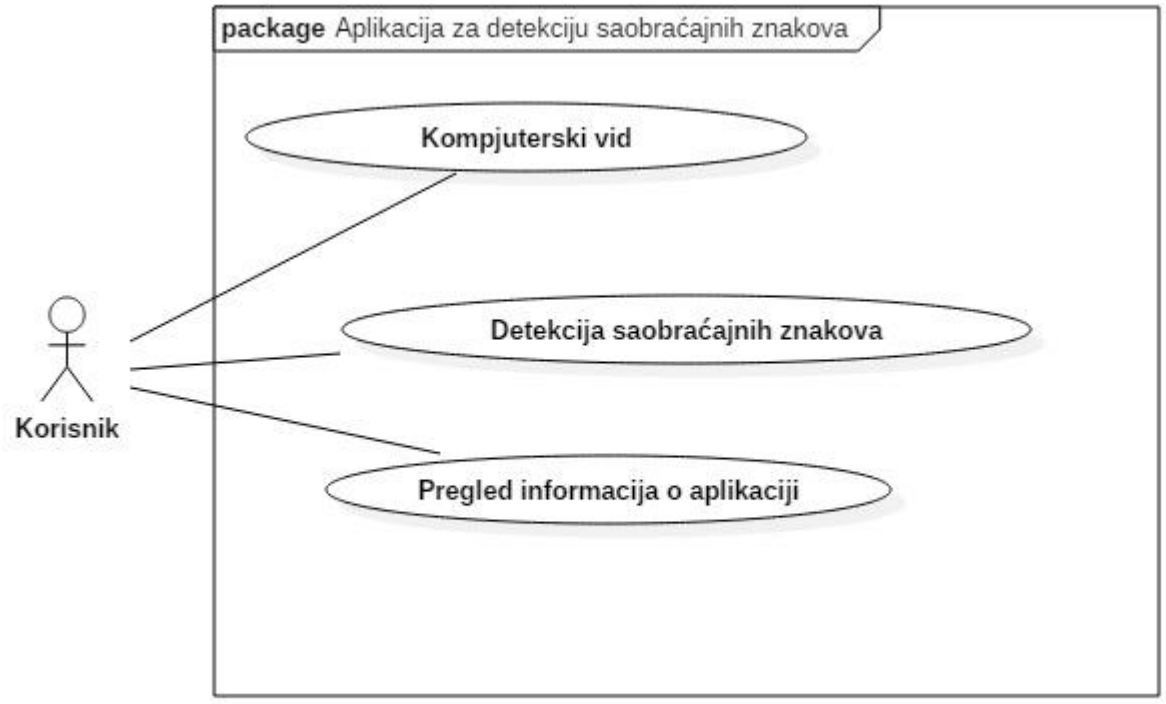
5.1. DEFINISANJE ZAHTEVA

Osnovna svrha aplikacije za detekciju saobraćajnih znakova je da u toku vožnje vozaču na koristan način prikazuje informacije o detektovanim saobraćajnim znakovima, kako bi pružila dodatni nivo sigurnosti u vožnji i omogućila da što manji broj znakova prodje neopažano od strane korisnika aplikacije.

Nakon pokretanja aplikacije i postavljanja uređaja na odgovarajuće mesto, aplikacija ne sme zahtevati od korisnika nikakvu dodatnu interakciju niti ga bespotrebno ometati pri vožnji.

5.2. DIJAGRAM SLUČAJEVA KORIŠĆENJA

Dijagram slučajeva korišćenja opisuje interakciju korisnika sa aplikacijom. Na slici 53 je prikazan dijagram slučajeva korišćenja koji opisuje interakciju korisnika sa aplikacijom za detekciju saobraćajnih znakova.



Slika 53: Dijagram slučajeva korišćenja

5.3. UČITAVANJE TENSORFLOW LITE MODELA

Kako bi pokretanje modela na mobilnom uređaju bilo moguće, najpre je potrebno u rad pod zavisnostima navesti implementaciju:

```
'org.tensorflow:tensorflow-lite-task-vision:0.3.0'
```

Ova zavisnost nam omogućuje da u okviru android aplikacije učitamo model u memoriju telefona i koristimo ga kako bi vršili detekcije nad podacima. Kako bi proces bio dodatno pojednostavljen, u aplikaciju je učitana biblioteka lib_task_api, koja sem što ispunjava sve potrebe za funkcionalnostima, takođe enkapsulira logiku za rad sa tensorflow lite modelom, što omogućava pisanje ostatka aplikacije nezavisno od implementacije tensorflow lite modela.

lib_task_api biblioteka se može naći na linku https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/android i prostom implementacijom ove biblioteke u rad osiguravamo osnovne funkcionalnosti za rad sa TensorflowLite modelom.

Samo učitavanje biblioteke u aplikaciju niije dovoljno kako bi se model koristio za detekciju, već je neophodno model pokrenuti onog trenutka kada se zna da će u daljem izvršavanju biti potreban. U ovom radu, taj trenutak je onda kada se odabere željena veličina pregleda sa kamere. Pozivom sledeće funkcije kreira se instanca modela i postaje dostupan za korišćenje:

```
public static Detector create(
    final Context context,
    final String modelFilename,
    final String labelFilename,
    final int inputSize,
    final boolean isQuantized)
    throws IOException {
  return new TFLiteObjectDetectionAPIModel(context, modelFilename);
}
```

Značenja parametara funkcije su:

1. **context** - Context trenutno aktivnog okruženja aplikacije
2. **modelFilename –** Ime tensorflow lite modela u direktorijumu sa resursima
3. **labelFilename –** Ime fajla sa mapom klasa u direktorijumu sa resursima
4. **inputSize –** veličina slike koju prima model, za model iz ovog rada, ta veličina je 320x320
5. **isQuantized –** Da li je model kvantizovan. Pošto tensorflow lite model maker vrši automatsku kvantizaciju nakon treniranja, vrednost ovog polja se postavlja na true.

Nakon što je detektor kreiran i referenca na njega sačuvana u promenljivoj, pozivanjem funkcije recognizeImage, kojoj prosledjujemo sliku u Bitmap formatu, slika se prosleđuje modelu i detekcije se vraćaju kao lista objekata klase *Detector.Recognition,* svaki od kojih sadrži sve informacije o detekciji (klasu koja je detektovana, sigurnost detekcije i lokaciju detekcije na forografiji). Daljom obradom ovih klasa korisniku je moguće prikazati detekcije na koristan i smislen način.

5.4 RAD SA KAMEROM

Rad sa kamerom mobilnog telefona predstavlja odabir i uspostavljanja sesije sa kamerom mobilnog telefona, čitanje podataka sa kamere i oslobađanje kamere nakon što više nije potrebna.

Aplikativni programski interfejs (API) korišćen za komunikaciju aplikacije sa kamerom je Camera2API.

Kako bi se kompleksnost izrade aplikacije smanjila, kao referenca za tok radnji pri otvaranju, čitanju i parsiranju slika sa kamere, korišćene su određene klase i delovi koda sa repozitorijuma koji se može naći na adresi https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/android i klase koje su korišćene su:

CammeraConnectionFragment.java – Klasa koja je zadužena za uspostavljanje sesije sa kamerom mobilnog telefona i prikazivanje prikaza sa kamere u realnom vremenu na korisničkom interfejsu.

CameraActivity – Abstraktna klasa koju nasleđuju klase kako bi koristile funkcionalnost komunikacije sa kamerom. Od bitnih funkcija, ova klasa sadrži funkciju za zahtevanje privilegija od korisnika, odabir kamere, postavljanje *CammeraConnectionFragment* fragmenta, kao i ostale funkvije koje su neophodne za rad sa kamerom. Kod funkcija je po potrebi menjan kako bi bio što korisniji za korišćenje u radu. Ovu klasu nasleđuju aktivnosti „kompjuterski vid" i „detekcija saobraćajnih znakova" kako bi radile sa kamerom

ViewKlase – Koriste se za olakšavanje iscrtavanja pregleda sa kamere i iscrtavanje detekcija u realnom vremenu.

ImageUtils – Pomoćna klasa koja sadrži funkcije za konverziju slika u neki drugi format (na primer YUV420 u ARGB8888).

Kako bi bilo koja aktivnost unutar aplikacije čitala podatke sa kamere, potrebno je samo da nasledi klasu CameraActivity i implementira abstraktne metode iz nje. Ovime je kompleksnost naknadnih dodavanja funkcionalnosti značajno smanjena.

5.3. KOMPJUTERSKI VID

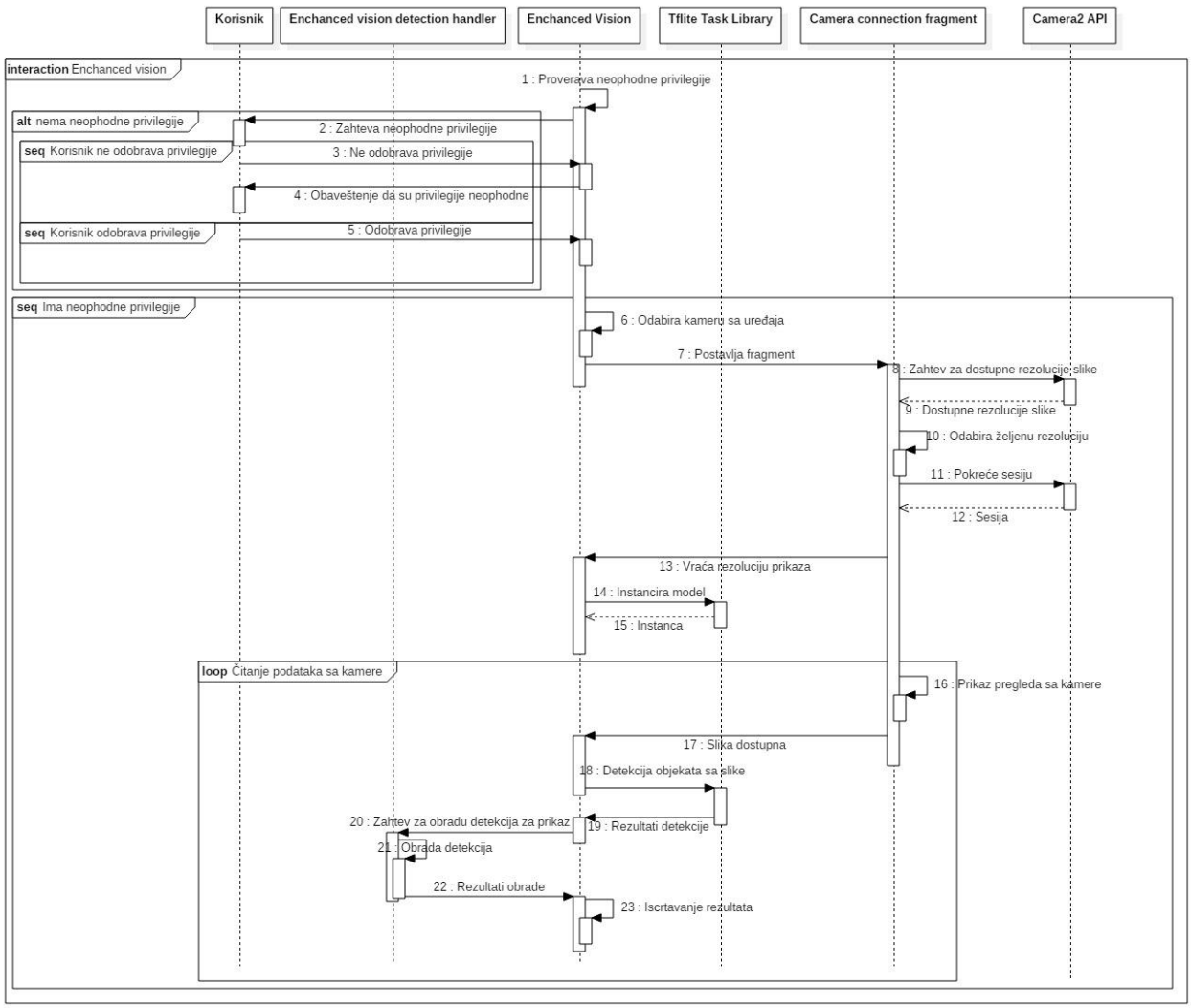
Kompjuterski vid predstavlja jedan od modova korišćenja aplikacije. Njegov cilj je da korisnicima približi princip rada aplikacije, jer nema mehanizam za kategoričko prikazivanje i pamćenje detekcija, tako da bi korisnik morao da aktivno posmatra telefon kako bi video da li je došlo do detekcije.

U ovom modu, detektovani znakovi se u realnom vremenu uokviruju određenom bojom, u zavisnosti od kategorije kojoj pripadaju, i iznad kvadrata koji uokviruje znak ispisjuje se ime detektovanog znaka i sigurnost modela mašinskog učenja pri detekciji.

Minimalna sigurnost algoritma u detekciju mora biti minimum 60% kako bi detekcija bila iscrtana na ekranu.

5.3.1. DIJAGRAM SEKVENCI

Na slici 54, prikazan je dijagram sekvenci u modu "kompjuterski vid".



Slika 54: Dijagram sekvenci za slučaj korišćenja aplikacije u modu "kompjuterski vid"

5.3.3 OBRADA DETEKCIJA

Obrada detekcija predstavlja ključnu funkcionalnost aplikacije. Samo pokretanje modela mašinskog učenja jeste dovoljno da bi se dobili podaci o detekciji, ali nije dovoljno da bi se korisniku aplikacije na valjan način prenela informacija o detektovanom znaku. U ovom modu detekcije, znakovi se uokviruju u realnom vremenu na ekranu uređaja.

DetectionInstance.java

Kreirana je klasa *DetectionInstance.java* koja nasleđuje klasu *Detector.Recognition* koja se nalazi u biblioteci lib_task_api i proširuje njene funkcionalnosti. *Detector.Recognition* klasa služi za prenošenje podataka o jednoj detekciji i polja koja se nalaze u ovoj klasi čuvaju podatke o nazivu klase koja je detektovana, jedinstvenom identifikatoru detektovane klase, sigurnosti detekcije i lokaciji na slici gde se detekcija nalazi.

Detector recognition ovu klasu proširuje tako što dodaje mogućnost da se za svaku detekciju umesto imena detekcije, koje je vezano za naziv klase pri treniranju modela mašinskog učenja, u odnosu na ime klase vrati ime koje je prilagođeno prikazu korisnicima. Tako na primer umesto da prikazano ime bude „warning_ahead", korisnku se prikazuje tekst „Upozorenje". Imena svih klasa se nalaze u strings.xml fajlu i ukoliko postoji potreba za promenom naziva neke klase, promenu je potrebno izvršiti na samo jednom mestu.

DetectionInstance.java takođe sadrži metodu koja za svaku detekciju vraća boju kvadrata koji će biti iscrtan oko nje, kako bi korisnici lakše uočili koja je klasa detektovana. Boje su:

- warning_ahead – žuta boja
- mandatory – bela boja
- no_stopping_or_parking – zelena boja
- stop – crvena boja
- give_road – siva boja
- warning_on_spot – plava boja

EnchancedVisionDetectionHandler.java

Klasa kreirana za potrebe obrade detekcija i njihovog iscrtavanja na ekranu.

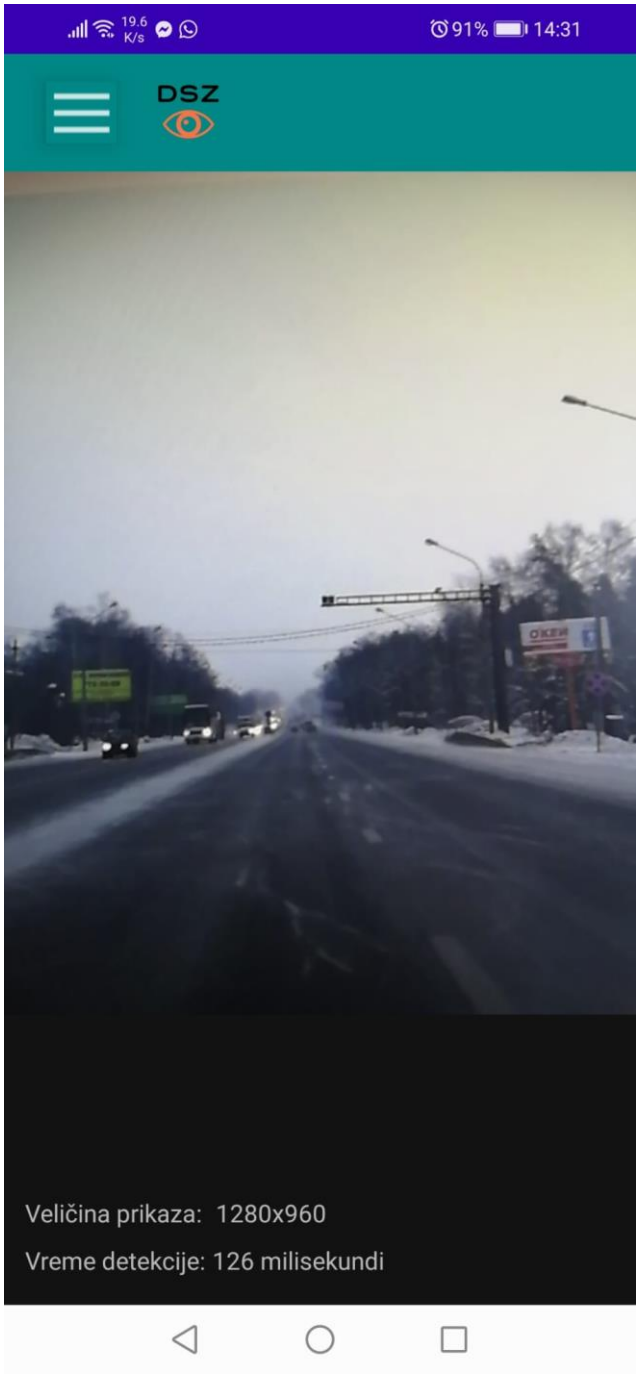
Sadrži funkciju process_results, koja služi za obradu detekcija. Ona kao ulazni parametar prima listu objekata klase *Detector.Recognition,* vrši validaciju detekcija, i podatke o detekcijama snima u listi objekata klase *DetectionInstance,* kako funkcija za iscrtavanje detekcija imala pristup ovim objektima.

Finkcija za iscrtavanje kao parametar prima objekat klase Canvas, koji je klasa koja se koristi za prikaz i modifikaciju slika u android okruženju. Na instanci ove klase funkcija za iscrtavanje iscrtava sve detekcije iz liste koju je kreirala funkcija za procesiranje detekcije.

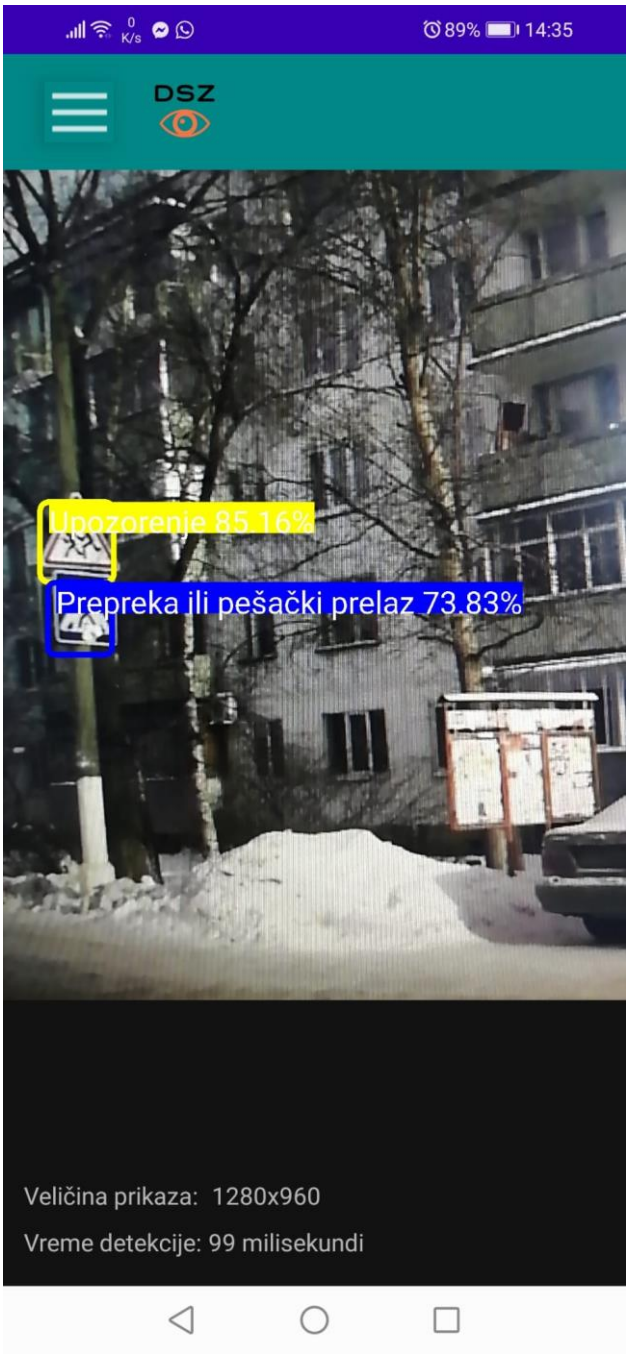
5.3.2. KORISNIČKI INTERFEJS

Na slici 55, vidi se korisnički interfejs aplikacije. U centralnom delu nalazi se prikaz sa kamere i preko njega se u realnom vremenu slikaju rezultati detekcije, što možemo videti na slici 56.

U donjem levom uglu nalaze se informacije o odabranoj veličini prikaza i vremenu potrebnom za detekciju znakova na slici.



Slika 55: Korisnički interfejs u modu kompjuterski vid



Slika 56: prikaz detekcija u realnom vremenu

5.4. DETEKCIJA SAOBRAĆAJNIH ZNAKOVA

U modu rada detekcija saobraćajnih znakova aplikacija korisniku omogućava struktuiran prikaz detektovanih znakova, kako bi korisnik za što manje vremena mogao biti obavešten o detektovanim znakovima na putu, bez potrebe da u trenutku detekcije gleda u aplikaciju.

Znakovi kružnog i trouglog oblika (ograničenja i upozorenja) na sebi nose određenu poruku, o ograničenju brzine vozila, predstojećoj prepreci na putu ili nečemu sličnom, na šta je bitno obratiti pažnju. Pošto su, zbog ograničenja modela u vidu ulazne rezolucije, kao i ograničenja uređaja na kojima će se model koristiti, znakovi u poglavlju 4 grupisani u 6 grupa, koje rezlikuju boja i oblik znaka, najbolji način za prikaz detekcije korisniku je isecanje znaka sa slike i prikazivanje istog uveličanog. Na ovaj način, aplikacija može da detektuje znakove po obliku, a korisnik samo kratkim pogledom u sliku može utvrditi koji se znak na njoj nalazi.

Znakovi koji su istog oblika koji uvek imaju isto značenje (npr. znak Stop) se mogu prikazati bez nepotrebnog zauzimanja mesta na ekranu, u vidu lampice upozorenja, jer uvek nose istu poruku i bitno je samo korisnika obavestiti o detekciji tog znaka, bez potrebe za daljim utvrđivanjem poruke na znaku.

Kako je detekcija bez ikakve obrade fotografije pre prosleđivanja iste modelu bila moguća na oko 5 metara od znaka, informacije o znaku dobijane su malo pre trenutka kada automobil prođe taj znak.Pošto se znakovi na putu nalaze sa desne strane puta, na visini od 2,2m do 2,4m u naseljenom mestu, odnosno 1,4m do 1,8m van naseljenog mesta[29], ukoliko je kamera uperena tako da posmatra nailazeći put, znakovi se mogu očekivati u gornjem desnom delu kamere.

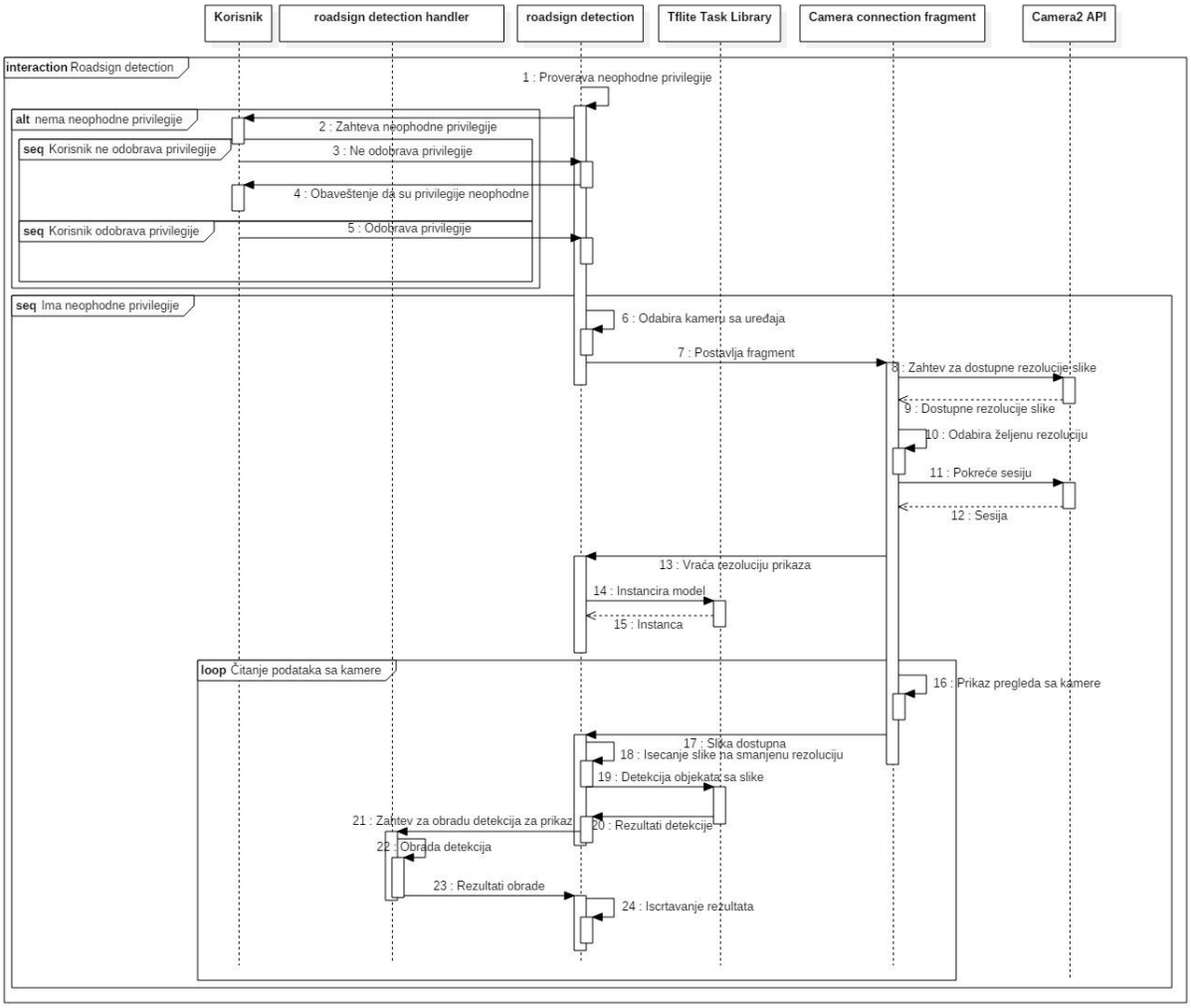
U cilju unapređenja daljine na kojoj se može detektovati slika, sa gornjeg desnog ugla slike iseca se slika čija je stranica jednaka 3/4 kraće stranice originalne fotografije i nad tom novom fotografijom se vrši detekcija. Ovom metodom, razdaljina na kojoj je detekcija moguća povećana je sa oko 5 na preko 15 metara, jer znak koji je veličine 100x100(10 000 piksela) na slici 1280x960 (1 228 800 piksela) zauzimao samo 0.81% ukupne fotografije, ali ukoliko se znak nalazi u delu veličine 720x720 koji je isečen sa slike veličine 1280x960, u tom slučaju znak zauzima 1,93% fotografije, sa time da u procesu nije izgubljen kvalitet, a detekcija znakova u daljini znatno poboljšana. Primer se može videti na slici 57, gde je žutim kvadratom obeležena zona u kojoj se vrši detekcija.

Dodatnu pogodnost čini to da je model koji se u radu koristi treniran na fotografijama čija je veličina uglavnom bila 720x720, tako da to predstavlja idealan slučaj korišćenja modela.

slika 57: isecanje dela fotografije u cilju unapredjenja daljine detekcije znakova

5.4.1. DIJAGRAM SEKVENCI

Na slici 58 prilazan je dijagram sekvence za slučaj korišćenja detekcija saobraćajnih znakova.



Slika 58: Dijagram sekvenci za slučaj korišćenja u modu "Detekcija saobraćajnih znakova"

5.4.2 OBRADA DETEKCIJA

U ovom modu, korisniku se detekcije prikazuju na taj način da tokom vožnje u bilo kom trenutku može pogledati u telefon i videti detekcije.

DetectionInstance.java

Kao klasa za prenošenje insformacija o detekcijama, ponovo se koristi klasa *DetectionInstance,* samo što su za potrebe ovog slučaja korišćenja klasi dodate metode za snimanje i čitanje vremenskih otisaka, kako bi se čuvala informacija o trenutku detekcije. Ova informacija je bitna kako bi bio implementiran period kada se korisniku ne prikazuje više instanci jedne klase. Ako je prosečno vreme detekcije 100 milisekundi, polje sa zadnjim znakom bi se prikazivalo 10 puta u sekundi, što bi rezultovalo bespotrebnom odvlačenju pažnje korisnika i zatrpavanjem nerelevantnim informacijama.

RoadsignDetectionHandler.java

U ovoj klasi se nalazi sva logika vezana za obradu i odabir detekcija koje će biti prikazane korisniku.

Pomoću 2 mape, koje beleže detekcije koje su na redu da budu prikazane i zadnji put kada je instanca detekcije prikazana, moguće je odrediti kada je vreme za prikazivanje neke detekcije. Pošto je algoritam ograničen na jednu detekciju po klasi, uvek pri učitavanju u mapu prednost ima detekcija koja je sa najvećom sigurnošću. Ukoliko se zadnja detekcija klase nekog znaka dogodila pre manje od 3 sekunde, neće biti prikazana.

Pošto samo 2 klase znakova na znaku nose podatke koji korisniku mogu biti od koristi, pri odabiru detekcije koja će korisniku biti prikazana kao zadnja, u slučaju da se u istom trenutku dogodilo više detekcija, odabira se ona detekcija za koju je model sigurniji da je ispravna.

Funkcija koja vraća detekcije koje je potrebno prikazati vraća objekat klase DetectionsToDisplay.java, koji sadrži isečeni bitmap zadnje detekcije, listu bitmapa koje je potrebno prikazati na listi zadnjih detektovanih znakova i polja koja definišu da li su detektovani neki od znakova stop, pešački prelaz ili zabranjeno parkiranje.

Nakon vraćanja detekcija za prikaz, zadnja detekcija se postavlja u odgovarajući objekat, listi zadnjih detekcija se dodaju potrebne detekcije, a detekcije znakova stop,pešački prelaz i zabranjeno parkiranje obrađuje klasa *FlashNotifUtil.*

FlashNotifUtil.java

Obrađuje detekcije znakova čiju sliku nije potrebno prikazivati, već je dovoljno samo pustiti animaciju koja upozorava korisnika da je znak detektovan. Animacija koja se prikazuje korisniku u slučaju detekcije je bojenje celog ekrana na sekundu, kako bi se korisniku privukla pažnja na činjenicu da je znak detektovan.

Za znak stop ili obrnuti trougao, ekran se popunjava crvenom bojom, za znak pešački prelaz plavom, a znak zabranjeno parkiranje magenta.

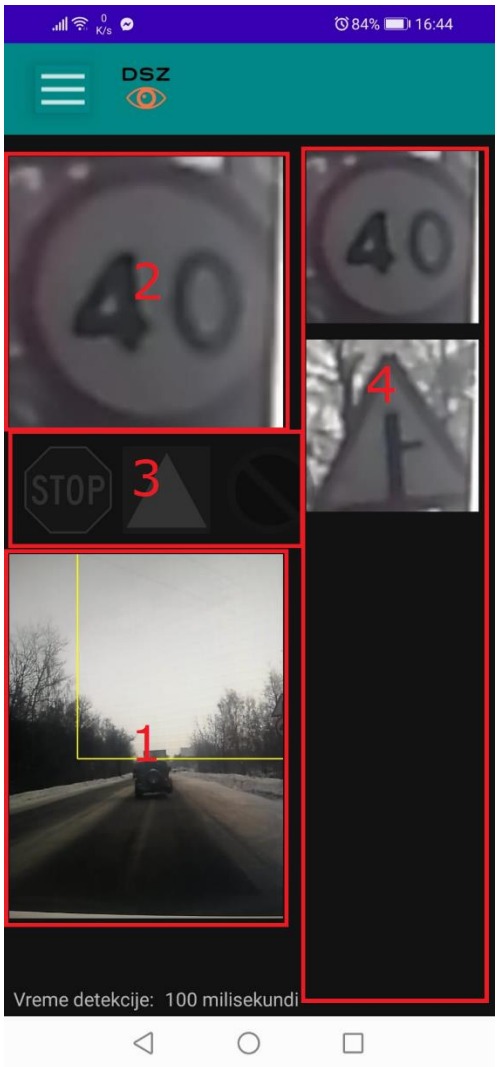
Pored animacije, na ekranu se osvetljava ikonica (slika 59) koja nastavlja da svetli sve dok nije prošlo 5 sekundi od zadnje detekcije te klase znaka. Dok se ikonica ne ugasi, ni animacija za tu klasu znaka se neće pokretati, već će samo ikonica nastaviti da svetli.

Slika 59: ugašene i upaljene ikonice

5.4.3. KORISNIČKI INTERFEJS

U modu detekcije saobraćajnih znakova, akcenat je na pravovremenoj detekciji znakova i obaveštavanju vozača o detekcijama. Korisnički interfejs je prikazan na slici 60 i sastoji se od nekoliko komponenti:

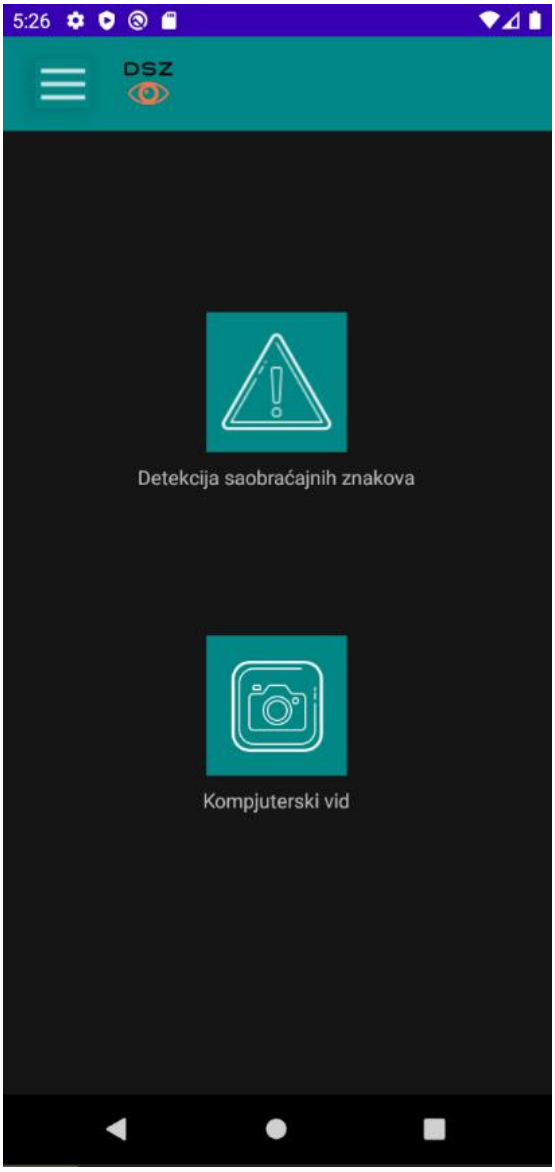
1) Prikaz sa kamere - omogućava korisniku da vidi da li je položaj kamere odgovarajući za detekciju
2) Prikaz zadnjeg detektovanog znaka - Zadnji detektovani znak sa slike se prikazuje uveličano kako bi korisnik, ukoliko ga nije ranije video, mogao lakše da pročita ono što je na njemu napisano.
3) Prikaz sa upozorenjuma – U slučaju detekcije nekog od znakova na mestu prepreke ili pešačkog prelaza, znaka stop ili obrnutog trougla ili znaka za zabranu parkiranja, ekran aplikacije zasvetli u određenoj boji (stop i obrnuti trougao – crveno, pešački prelaz ili prepreka za usporavanje saobraćaja – plavo i zabranjeno parkiranje - magenta) i upali se odgovarajući znak. Znak ostaje upaljen 5 sekundi nakon zadnje detekcije znaka te kategorije, kako bi se izbeglo nepotrebno višestruko upozoravanje na isti znak na putu.
4) Lista zadnjih vidjenih znakova – zadnjih 20 vidjenih znakova se čuva u listi



Slika 60: korisnički interfejs u modu detekcije saobraćajnih znakova

5.5. POČETNA STRANICA

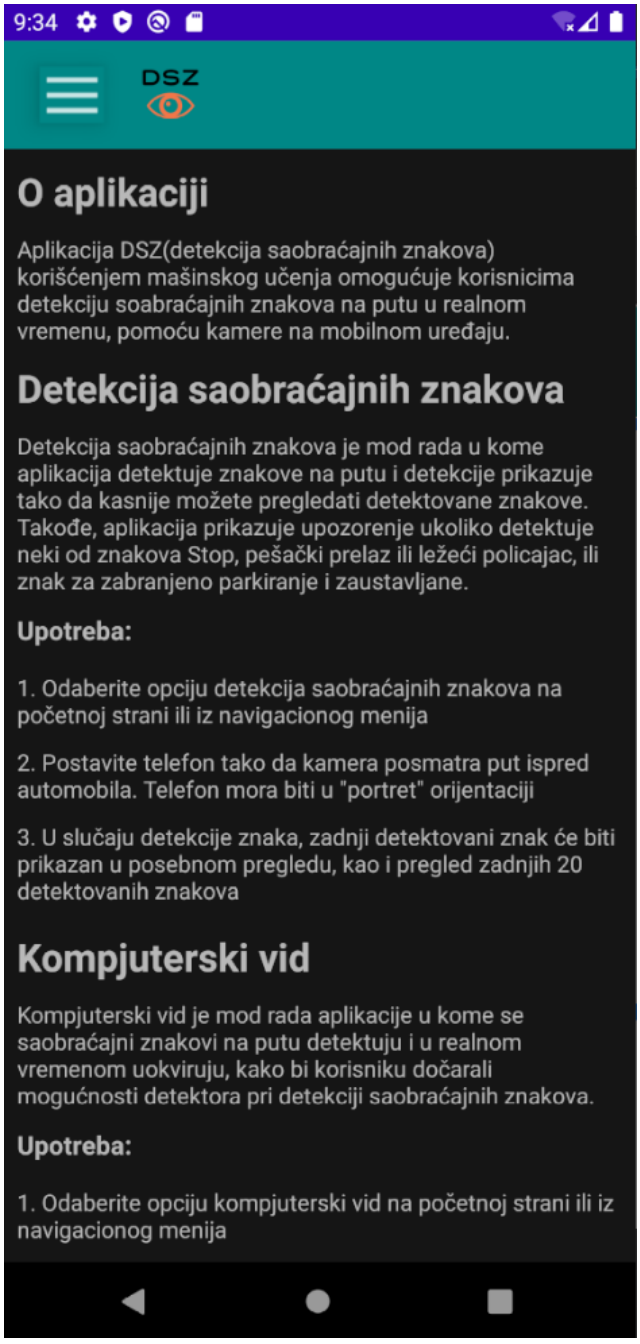
Početna stranica aplikacije (slika 61) je kreirana sa ciljem da korisnik što brže odabere mod koji želi da koristi bez nepotrebne navigacije kroz menije. Početni meni sačinjavaju 2 dugmeta, jedno koje je označeno glifikonom znaka upozorenja i vodi na aktivnost detekcija saobraćajnih znakova i drugo koje je označeno glifikonom kamere i koje vodi na aktivnost kompjuterski vid.



Slika 61: Početna stranica

5.6. O APLIKACIJI

Stranici o aplikaciji (slika 62) se može pristupiti putem navigacionog menija i na njoj se nalazi opis aplikacije, predviđenu namenu modova upotrebe, kao i uputstvo za korišćenje svakog od modova.

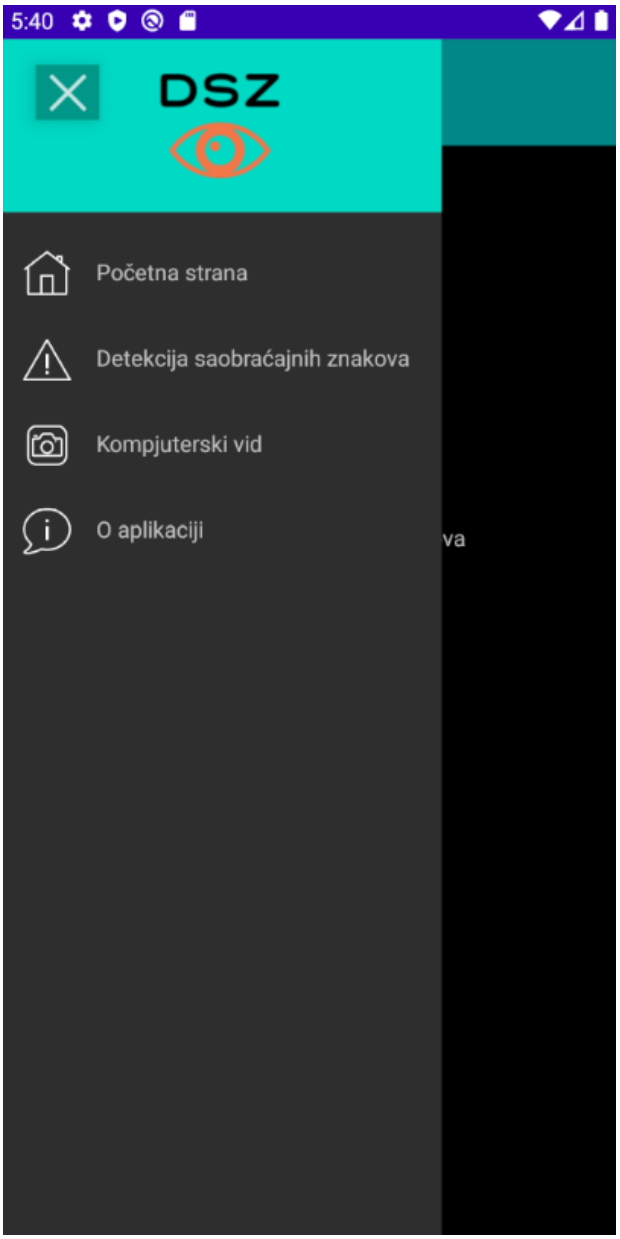


Slika 62: Stranica o aplikaciji

5.7. NAVIGACIONI MENI

Radi olakšane navigacije kroz aplikaciju, implementiran je navigacioni meni (slika 64), koji korisnicima omogućava da dospeju do bilo kog dela aplikacije u 2 klika. Meni se otvara klikom na dugme za otvaranje navigavionog menija u gornjem levom uglu ekrana (slika 63)

Slika 63: Navigacioni meni (zatvoren)



Slika 64: Navigacioni meni (Otvoren)

LITERATURA

[1] Hawkins D., Journal of Chemical Information and Computer Sciences, University of Minnesota, American Chemical Society, 2004.

[2] GAN Training https://developers.google.com/machine-learning/gan/training (posećeno 5.12.2021)

[3] Official Tensorflow API documentation https://www.tensorflow.org/api_docs (posećeno 11.12.2021.)

[4] Official TensorFlow Lite Model Maker documentation. https://www.tensorflow.org/lite/tutorials/model_maker_object_detection (posećeno 24.12.2021.)

[5] Chamant M., Activation Functions : Why "tanh" outperforms "logistic sigmoid"? https://medium.com/analytics-vidhya/activation-functions-why-tanh-outperforms-logistic-sigmoid-3f26469ac0d1, 2019. (posećeno 25.12.2021.)

[6] Radhakrishnan P., What are Hyperparameters? and How to tune the Hyperparameters in a Deep Neural Network? https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a, 2017. (posećeno 25.12.2021.)

[7] Nikolić M., Zečević A., Mašinsko Učenje, Beograd, Prirodno matematički fakultet, 2019.

[8] Engelbrecht A., Computational Intelligence An Introduction, University of Pretoria, John Wiley & Sons, 2007.

[9] The difference between Artificial Intelligence, Machine Learning and Deep Learning https://datacatchup.com/artificial-intelligence-machine-learning-and-deep-learning 2019. (posećeno 9.10.2021.)

[10] Živković S., PyTorch – Popular techniques to prevent the Overfitting in a Neural Networks https://datahacker.rs/018-pytorch-popular-techniques-to-prevent-the-overfitting-in-a-neural-networks/ 2021. (posećeno 13.11.2021.)

[11] What Is Machine Learning: Definition, Types, Applications And Examples https://www.potentiaco.com/what-is-machine-learning-definition-types-applications-and-examples (posećeno 9.10.2021.)

[12] Gudivada V., Apon A., Ding J., Data Quality Considerations for Big Data and Machine Learning: Going Beyond Data Cleaning and Transformations https://www.researchgate.net/publication/318432363_Data_Quality_Considerations_for_Big_Data_and_Machine_Learning_Going_Beyond_Data_Cleaning_and_Transformations 2017. (posećeno 8.1.2022.)

[13] Bagherzadeh R., Latifi M., Faramarzi A. mploying a Three-Stage Data Mining Procedure to Develop Sizing System https://www.researchgate.net/publication/230874681_Employing_a_Three-Stage_Data_Mining_Procedure_to_Develop_Sizing_System 2010. (posećeno 12.11.2021.)

[14] Tayal N., K-means Clustering Algorithm https://nancytayal.medium.com/k-means-clustering-algorithm-6b8958028fe4 2021. (posćeno 23.11.2021.)

[15] Sainani K., Introduction to Principal Components Analysis https://onlinelibrary.wiley.com/doi/full/10.1016/j.pmrj.2014.02.001 2014. (posećeno 21.11.2021.)

[16] Dertat A., Applied Deep Learning - Part 3: Autoencoders https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798 2017. (posećeno 8.1.2022.)

[17] Overview of GAN Structure https://developers.google.com/machine-learning/gan/gan_structure (posećeno 5.12.2021.)

[18] Reinforcement Learning : Markov-Decision Process https://towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision-process-44c533ebf8da 2019. (posećeno 4.1.2022.)

[19] Nagyfi R, The differences between Artificial and Biological Neural Networks https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7, 2018. (posećeno 23.12.2021.)

[20] Dawar H. Stochastic Gradient Descent https://medium.com/analytics-vidhya/stochastic-gradient-descent-1ab661fabf89 2020. (posećeno 17.12.2021.)

[21] Mahajan P., Fully Connected vs Convolutional Neural Networks https://medium.com/swlh/fully-connected-vs-convolutional-neural-networks-813ca7bc6ee5 2020. (posećeno 29.12.2021.)

[22] Lendave V. What Is A Convolutional Layer? https://analyticsindiamag.com/what-is-a-convolutional-layer/ 2021. (posećeno 17.12.2021.)

[23] Explain Pooling layers: Max Pooling, Average Pooling, Global Average Pooling, and Global Max pooling. https://androidkt.com/explain-pooling-layers-max-pooling-average-pooling-global-average-pooling-and-global-max-pooling/ 2021. (posećeno 21.12.2021.)

[24] Tensorflow https://www.tensorflow.org/ (posećeno 17.12.2021.)

[25] Tensor Processing Unit https://semiengineering.com/knowledge_centers/integrated-circuit/ic-types/processors/tensor-processing-unit-tpu/ (posećeno 11.12.2021.)

[26] Keras https://keras.io/about/ (posećeno 8.1.2022.)

[27] Transfer learning and fine-tuning https://www.tensorflow.org/guide/keras/transfer_learning (posećeno 8.1.2022.)

[28] IBM Unsupervised Learning https://www.ibm.com/cloud/learn/unsupervised-learning (posećeno 8.1.2022.)

[29] Pravilnik o postavljanju saobraćajne signalizacije https://www.paragraf.rs/propisi/pravilnik-o-saobracajnoj-signalizaciji.html (posećeno 30.1.2022.)

[30] Introduction to Kaggle Kernels Yufeng G. https://towardsdatascience.com/introduction-to-kaggle-kernels-2ad754ebf77#:~:text=Kaggle%20Kernels%20is%20a%20free,you%20have%20an%20internet%20connection. 2017. (posećemp 5.2.2022.)