

**SVEUČILIŠTE U SPLITU
FAKULTET ELEKTROTEHNIKE, STROJARSTVA I
BRODOGRADNJE**

DIPLOMSKI RAD

**PROTOTIP AUTONOMNOG VOZILA
IZRAĐENOG POMOĆU METODE
OPONAŠANJA**

Marko Rašetina

Split, rujan 2019.



Diplomski studij: **Komunikacijska i informacijska tehnologija**

Smjer/Usmjerenje: **Telekomunikacije i informatika**

Oznaka programa: 242

Akademска godina: 2018./2019.

Ime i prezime: **Marko Rašetina**

Broj indeksa: 759-2017

ZADATAK DIPLOMSKOG RADA

Naslov: **Prototip autonomnog vozila izrađenog pomoću metode oponašanja**

Zadatak: Opisati kako su digitalni slikovni i video signal predstavljeni na računalu. Objasniti osnovne pojmove umjetne inteligencije i strojnog učenja te opisati koncept neuronskih mreža s naglaskom na konvolucijsku neuronsku mrežu. Korištenjem programskog jezika Python i odgovarajućih modula za Python izraditi konvolucijsku neuronsku mrežu i prethodnu obradu signala. Neuronska mreža mora moći predviđati kut upravljanja. Auto se mora moći kretati samo desnom trakom i s konstantnom brzinom (15/20/25 mph). Za ostvariti konstantu brzinu poslužiti se PID kontrolerom. Za stazu i auto iskoristiti Udacity simulator.

Rad predan:

Predsjednik
Odbora za diplomski rad:

prof. dr. sc. Dinko Begušić

Mentor:

prof. dr. sc. Zoran Blažević

IZJAVA

Ovom izjavom potvrđujem da sam diplomski rad s naslovom „Prototip autonomnog vozila izrađenog pomoću metode oponašanja“ pod mentorstvom prof. dr. sc. Zoran Blažević pisao samostalno, primjenivši znanja i vještine stečene tijekom studiranja na Fakultetu elektrotehnike, strojarstva i brodogradnje, kao i metodologiju znanstveno-istraživačkog rada, te uz korištenje literature koja je navedena u radu. Spoznaje, stavove, zaključke, teorije i zakonitosti drugih autora koje sam izravno ili parafrazirajući naveo u diplomskom radu citirao sam i povezao s korištenim bibliografskim jedinicama.

Student

Marko Rašetina

Sadržaj

1. UVOD	1
2. MULTIMEDIJSKI SUSTAVI.....	3
2.1. Uvod u multimediju	3
2.2. Slika	4
2.3. Video signal	7
3. UMETNINA INTELIGENCIJA	9
3.1. Inteligencija.....	9
3.2. Povijest i razvoj umjetne inteligencije.....	10
3.3. Problematika i primjena umjetne inteligencije	13
3.4. Strojno učenje	15
3.4.1. Učenje s obzirom na ljudski nadzor.....	16
3.4.2. Izvan mrežno učenje i učenje na mreži.....	19
3.4.3. Učenje na temelju primjera i učenje na temelju modela.....	21
4. NEURONSKE MREŽE.....	24
4.1. Umjetna neuronska mreža.....	24
4.2. Perceptron i višeslojni perceptron.....	26
4.3. Aktivacijska funkcija	31
4.4. Treniranje i optimizacija	35
4.4.1. Funkcija cijene	36
4.4.2. Algoritam optimizacije	38
4.5. Pretreniranje i podtreniranje	40
4.6. Reguliranje.....	42
4.7. Arhitektura neuronske mreže	46
4.7.1. Konvolucijska neuronska mreža	47
5. IZRADA AUTONOMNOG VOZILA	53
5.1. Korišteni programski jezik i simulator	53
5.1.1. Python	53
5.1.2. Udacity simulator i generiranje podataka	55
5.2. Predobrada i vizualizacija podataka za treniranje.....	58
5.3. Model umjetne neuronske mreže i treniranje.....	71
5.4. Rezultat treniranja i testiranje	79
6. ZAKLJUČAK	88
LITERATURA	89

POPIS SLIKA.....	92
POPIS OZNAKA I KRATICA.....	94
SAŽETAK	96
KLJUČNE RIJEČI.....	96
TITLE	97
SUMMARY.....	97
KEYWORDS.....	97
Dodatak A	98
Dodatak B	100

1. UVOD

Automatiziranim vozilom se smatra svako vozilo u kojem se barem neki aspekti sigurnosno-kritičnih upravljačkih funkcija kao što je upravljanje, gas ili kočenje događaju bez izravnog upletanja vozača. Vozila koja sugeriraju vozača na potencijalne opasnosti upozorenjima kao što je upozorenje o sudaru naprijed, ali i ne izvršavaju upravljačku funkciju se u ovom kontekstu ne smatraju automatiziranim iako takova upozorenja mogu igrati značajnu ulogu u sigurnosti. Automatiziranim vozilima su potrebni podaci koji se zaprimaju od različitih senzora (kamere, GPS, ultrazvučni senzor, LiDAR, žiroskop, altimetar, itd.) i telekomunikacije kako bi mogli donijeti vlastite prosudbe o sigurnosno kritičnim situacijama i djelovali na odgovarajući način izvršavajući kontrolu na nekoj razini. S obzirom na upravljačke funkcije koje vozilo obavlja organizacija NHTSA je kategorizirala vozila u pet nivoa:

Nivo 0 – nema automatizacije što znači da vozač isključivo sam upravlja komandama vozila i to cijelo vrijeme.

Nivo 1 – automatizacija određenih funkcija uključuje jednu ili više određenih kontrolnih funkcija, a u slučaju automatizacije više funkcija svaka funkcija djeluje neovisno o drugoj.

Nivo 2 – automatizacija kombinirane funkcije označava da je vozilo u stanju upravljati najmanje dvjema primarnim upravljačkim funkcijama dizajnirane tako da rade u skladu s vozačem.

Nivo 3 – ograničena automatizacija u automatskoj vožnji to jest pod određenim prometnim ili okolišnim uvjetima zahtijeva se od vozača da preuzme upravljanje vozilom.

Nivo 4 – potpuna automatizacija odnosno vozilo obavlja sve funkcije vožnje i nadgleda stanje na cesti tijekom čitavog putovanja. [1]

Tema diplomskog rada je izgraditi i istrenirati konvolucijsku neuronsku mrežu pomoću metode oponašanja kako bi bila u stanju predviđati kut upravljanja na osnovu slike dobivene iz kamere koja je postavljena na sredini vozila. Diplomski rad je podijeljen u dva dijela. Prvi dio se sastoji od teorije koja je potrebno poznavati kako bi se mogao što bolje obaviti praktični dio. U drugom

dijelu rada opisan je praktični dio rada pri čemu su prikazani isječci iskorištenog koda kojim se realizirao rad.

U drugom poglavlju se susrećemo s multimedijskim sadržajem pri čemu se naglasak stavlja na slikovni i video signal s obzirom da umjetna neuronska mreža upravo obrađuje i donosi odluku o kutu upravljanja na osnovu dobivene slike. U trećem poglavlju će se razjasniti definicija inteligencije potom opisati povijesni razvoj umjetne inteligencije. Isto tako će se u trećem poglavlju dotaknuti jednog jako rasprostranjenog pod područja umjetne inteligencije, a to je strojno učenje. U četvrtom poglavlju će se definirati umjetna neuronska mreža te će se razjasniti svi potrebni blokovi i dijelovi potrebni za kreiranje jedne umjetne neuronske mreže. Na samom kraju će se spomenuti arhitekture i dati će se detaljniji opis konvolucijske neuronske mreže jer će upravo ta arhitektura neuronske mreže biti upotrijebljena da na osnovu slike predvidi vrijednost kuta zakreta. Peto poglavlje započinje opisom upotrijebljenih alata počevši od programskog jezika Python i njegovih najznačajnijih modula te upotrijebljenog simulatora. Nakon upoznavanja s alatom u nastavku je opisana primijenjena predobrada podataka popraćena grafovima. Također je opisana i prikazana sloj po sloj iskorištena arhitektura konvolucijske neuronske mreže te na samom kraju su prikazani dobiveni rezultati treniranja i kako se vrši testiranje.

Cilj je korištenjem navedenog alata postići da se automobil samostalno kreće unutar desne trake na način da konvolucijska neuronska mreža predviđa kut upravljanja iz slike dobivene od kamere postavljene na automobilu, a PID kontroler treba nastojati zadržati brzinu vozila konstantom upotrebljavajući gas odnosno kočnicu.

2. MULTIMEDIJSKI SUSTAVI

2.1. Uvod u multimediju

Riječ multimedija je sastavljena od dvije riječi: multi i medij. Obje riječi dolaze iz latinskog jezika pri čemu riječ multi dolazi od riječi multus (brojan), a riječ medij dolazi od riječi medium (sredina). Multimedija predstavlja integraciju više oblika medija u jednu cjelinu. Primjer multimedija je internetska stranica s tekstom i slikama. U računalnoj znanosti multimedija znači da se računalni podatci mogu predstaviti putem zvuka, videa te animacije uz tradicionalne medije kao što su tekst, slika i sl.

Multimedijijski računalni sustav ima visoku sposobnost integriranja različitih medija pri čemu nam multimedijijski računalni sustavi uz odgovarajući softver omogućava predstavljanje, pohranu, obradu i manipuliranje multimedijijskim sadržajem. Glavne komponente multimedijijskog računalnog sustava su:

- Tekst – sadrži alfanumeričke i neke druge posebne znakove,
- Grafika – tehnologija koja generira, manipulira, obrađuje predstavlja i prikazuje slike,
- Animacija – pomaže u stvaranju, razvoju, sekvenciranju i prikazivanju skupa slika,
- Audio – tehnologija koja snima, sintetizira i reproducira zvuk,
- Video – tehnologija koja bilježi, sintetizira i prikazuje slike (okvire) u sekvencama fiksne brzine pri čemu se stvara iluzija pokreta.

Područja u kojima se primjenjuje multimedija:

- Audio/video konferencija,
- E-knjige,
- E-učenje,
- Web,
- Video igre,
- Animirano filmovi,

- Virtualna stvarnost,
- Kupovanje putem interneta.

Razvitak multimedijskih aplikacija omogućila je:

- Digitalizacija skoro svih medija i uređaja,
- Razvitak podatkovnih i komunikacijskih mreža,
- Veliki kapaciteti uređaja za pohranjivanje te brzi i specijalizirano procesori,
- Unaprijeđeni softver (operacijski sustavi, koder/dekoder). [2]

2.2. Slika

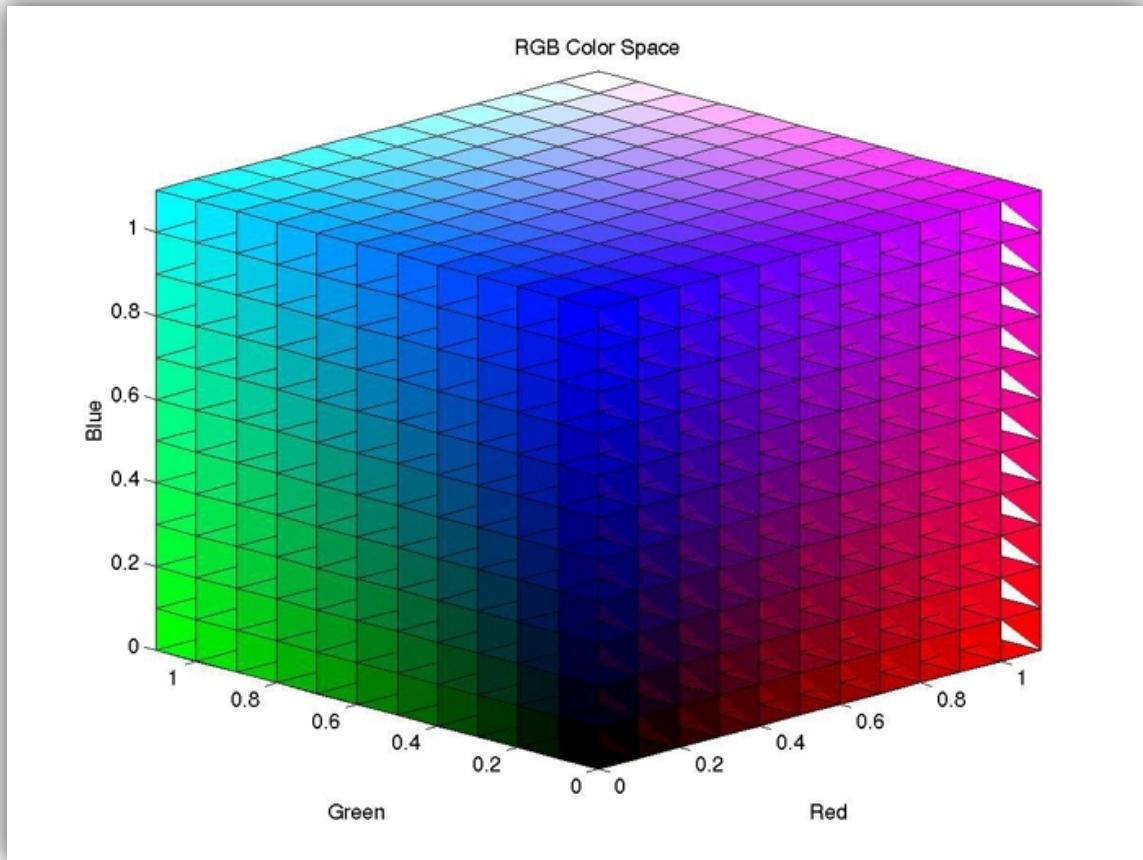
Slika je vizualna reprezentacija nečega što se našlo unutar vidnog polja kamere. Isto tako bi sliku mogli definirati kao grupu obojenih točaka na ravnoj površini koja izgleda isto kao i nešto drugo. Softverske aplikacije slike dijele se u dva tipa formata grafike:

- vektorska grafika (bazirana je na vektorima te koristi točke, linije, krivulje i oblike kako bi kreirala prikaz, a pohranjuje se kao matematička formula koja opisuje korištene linije, krivlje, točke i oblike),
- rasterska grafika (koristi pravokutnu rešetku koja se sastoji od ćelija, piksela, jednake veličine i svaka ćelija ima svoju boju).

U ovom radu se koriste slike koje pripadaju rasterskom formatu grafike.

Umjetnici od davnina još znaju da se miješanjem triju ili četiriju boja može dobiti bilo koja druga boja. Te se boje zovu primarne boje i postoji više kombinacija primarnih boja, a koju ćemo od kombinacija primarnih boja odabrati ovisi o njihovoj primjeni. Za lakše razumijevanje kako se iz primarnih boja može dobiti bilo koja druga boja definira se prostor boja. Prostor boja ili kako se to još zove model boja je apstraktni matematički model koji opisuje raspon boja kao brojeve te različitim kombinacijama dobivamo različite boje to jest svaka boja predstavlja jednu točku u prostoru. Postoji pet glavnih modela pri čemu se svaki model sastoji od više pod

modela. Primjer jednog takvog prostora boja je RGB (R = Red, G = Green, B = Blue) prostor boja (slika 2.1.).

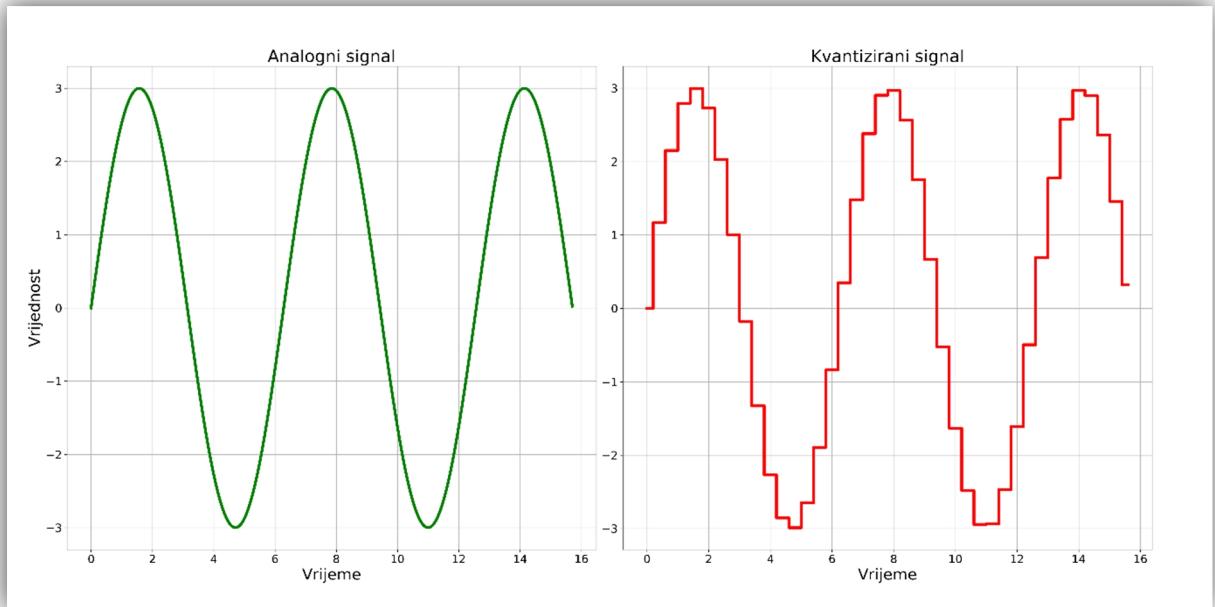


Slika 2.1. RGB prostor boja [3]

Osim RGB modela koji se najčešće koristi u elektroničkim sustavim imamo još CIE, luma plus chroma/chrominance, cilindrične transformacije i CMYK. U ovom radu se koriste slike RGB modela boja.

Svijet oko nas je jedan veliki analogni sustav. Bilo koji ulaz koji možemo uočiti je analogan. Na primjer zvuk je analogan signal. Tako naše uši slušaju analogni sustav i mi govorom proizvodimo analogni signal. Ista logika vrijedi za slike i video gdje su njihovi izvori analognog tipa, a naše oči analogni senzori. Analogni signal se definira kao signal s kontinuiran područjem vrijednosti definiran u kontinuiranom vremenskom području (slika 2.2).

S druge strane računalni sustavi (osobno računalo, raspberry Pi, mobitel itd.) su digitalni sustavi te se u svom radu koriste digitalnim signalima. Digitalni signal se definira kao signal s diskretnim područjem vrijednosti definiran u diskretnom vremenskom području (slika 2.2.). Postupak pretvorbe analognog signala u digitalni signal (A/D pretvorba) se postiže uzorkovanjem i/ili kvantizacijom. [4]



Slika 2.2. Analogni signal (lijevo) i kvantizirani signal (desno)

Kao što je već prije spomenuto u ovome radu će se koristi rasterski format slike. Da bi u potpunosti razumjeli rasterski format slike definirati ćemo što je to rezolucija i dubina slike. Rezolucija slike je mjera kojom se opisuje oštrina i jasnoća slike pri čemu se definira koliko piksela slika ima po širini i visini. Primjer nekolicine popularnih rezolucija su 640x480, 1280x720, 1920x1080 i tako dalje. Dubina boje, također poznat kao dubina bita, je broj bitova koji se koristi za označavanje boje jednog piksela. Slike se s obzirom na dubinu slike mogu razvrstati na:

- kolor slike,
- slike s paletom boja,
- sive slike,

- crno/bijele slike.

Raspon bita koji se koristi za boje pojedinog piksela varira od 1-bit, 4-bit, 8-bit, 15/16 bita (visoke boje), 24-bit (prave boje) pa do 30/36/48-bit (duboke boje). Kod RGB modela boje s dubinom slike od 24-bit svaki piksel predstavljen je s 3 bajta, a svaki bajt definira jednu komponentu iz RGB modela boja (1 bajt = 8 bita => 3 bajta = 24 bita). U sivim slikama se najčešće koristi 1 bajt za definiranje različitih vrijednosti nijansi sive boje. [2]

2.3. Video signal

Video predstavlja sekvencu slika koji se prikazuju u određenom vremenu periodu. Video ima za cilj stvoriti iluziju pokreta. Da bi bolje razumjeli video signal opisati ćemo nekoliko karakteristika koje se vežu uz video signal. Prva karakteristika koju ćemo opisati je broj slika u sekundi (FPS) ili brzina kadrova. Brzina kadrova je frekvencija iliti brzina kojom se na zaslon pojavljuju statične slike zvane okviri, a mjerna jedinica je herc (Hz). Za stvoriti iluziju pokreta dovoljno je da se slike izmjenjuju brzinom većom od 10 slika/sekundi. Danas velika većina digitalnih video kamera snima s minimalnom brzinom od 30 slika/sekundi što se ujedno smatra i full-motion video. Drugi parametar je omjer slike koji nam govori u kojem su odnosu širina i visina slike. Neki od poznatijih omjera su 1:1, 4:3, 16:9 i tako dalje. I zadnji parametar koji ćemo spomenuti je stereoskopski odnosno monoskopski video. Stereoskopija je tehnika za staranje iluzije dubine u slici. Postiže se tako što se pomoću dvije kamere snima isti objekt iz dvije različite perspektive, za svako oko jedna perspektiva. Monoskopski video snima objekt iz samo jedne perspektive. U ovom radu se koristi digitalni video signal s 30 slika u sekundi, omjera slike 2:1 te se koristi samo jedna kamera što znači da imamo monoskopski video.

Izračunajmo potrebnu brzinu koja bi nam bila potrebna za prijenos video signala. Uzeti ćemo da je rezolucija VGA drugim riječima imamo 640x480 piksela. Neka je video signal sastavljen od slika RGB modela pri čemu je svaka primarna boja definirana s 8 bita (1 bajt) po jednom pikselu te neka je broj slika u sekundi jednak 30. Za prijenos ovakvog video trebala bi nam sljedeća brzina:

$$640 * 480 * 8 \text{ bit} * 3 * 30 \text{ s}^{-1} = 221,184,000 \text{ bit/s}$$

Kad bi se ova brzina pretvorila u Mb/s to bilo malo više od 221 Mb/s. Uočavamo da se radi o velikoj propusnosti koja nam je potrebna pri čemu je i rezolucija i broj slika u sekundi relativno malen. Za usporedbu danas prosječni mobitel može snimati video u FHD rezoluciji (1920x1080) pri 60 FPS-a. Tu nam od velike pomoći može doći komprimiranje video signala. Kompresijskih metoda za video signala ima dosta, a neki od njih su MPEG-1, MPEG-2, MPEG-4, H.261, H.263 i tako dalje. [2]

Svi kompresijski formati pa tako i MJPEG ovise o otkrivanju uzoraka i predstavljanje tih uzoraka kraćim kodovima iliti porukama. Što je uzorak složeniji ili slučajniji, manja je vjerojatnost da se uzorak komprimira to jest razlika između originalne veličine i komprimirane će biti manja. MJPEG je video kompresijski format u kojem se okviri videozapisa komprimiraju pojedinačno kao JPEG slike (kompresija unutar okvira) za razliku od na primjer H.264 formata koji komprimira među slikovno (vremenska kompresija). Zbog toga omjer kompresije MJPEG formata je manji 1:20 nego što je to kod formata koji koriste vremensku kompresiju (1:50), ali MJPEG je manje računski intenzivan i traži manje memorijске zahtjeve na hardverskim uređajima. MJPEG standard se koristi za snimanje i uređivanje videozapisa, na igraćim konzolama, kod digitalnih kamera, IP kamera, MJPEG preko HTTP-a i tako dalje. [5]

3. UMJETNA INTELIGENCIJA

3.1. Inteligencija

Da bismo mogli opisati i razumjeti što je to umjetna inteligencija prvo ćemo opisati što je to inteligencija. I dan danas ne postoji jedna jedinstvena i jedno značajna definicija što je to inteligencija. Kroz povijest definicija inteligencije se nadograđivala kako je čovjek dolazio sve više do spoznaja o sebi i okolini koja ga okružuje. Trenutna zadnja formalna definicija inteligencije potpisana je od skupine akademskih istraživača njih pedeset dvoje koji su radili u području povezanim s testiranjem inteligencije. Definicija je dana u javnoj izjavi „Mainstream Science on Intelligence“ koja je javno objavljena u Wall Street Journal 13. prosinca 1994. godine. Njihova definicija inteligencije glasi:

„Inteligencija je vrlo općenita mentalna sposobnost, koja između ostalog, uključuje sposobnost rasuđivanja, planiranja, rješavanja problema, apstraktno mišljenje, shvaćanje kompleksnih ideja, brzo učiti i učiti iz iskustva. Ona ne obuhvaća samo učenje iz knjiga, usku akademsku vještina ili elegantno rješavanje testova. Prije toga ona reflektira širu i dublju sposobnost za dokučivanje našeg okruženja – opažanja, shvaćanja smisla u stvarima ili spoznati što napraviti.“ Mainstream Science on Intelligence, 1994. [6]

S obzirom da trenutno ne postoji jedna jedinstvena definicija i tumačenje inteligencije opće prihvaćen je pristup nazvan višestruka inteligencija. Američki razvojni psiholog Howard Gardner opisao je devet vrsta inteligencije, a one su:

- Prirodoslovna inteligencija,
- Glazbena (ritmička) inteligencija,
- Logičko – matematička inteligencija,
- Egzistencijalna inteligencija,
- Inter personalna inteligencija,
- Tjelesno – kineziološka inteligencija,
- Jezična inteligencija,
- Intra – osobna inteligencija,
- Prostorna inteligencija.

S druge strane imamo Steinbergovu teoriju višestruke inteligencije pri čemu njegova definicija višestruke inteligencije se sastoji od tri osnovna tipa:

- Analitička inteligencija – potrebna nam je kada trebamo analizirati nešto ili riješiti problem. Ovaj tip inteligencije se mjeri IQ testovima.
- Kreativna inteligencija – sposobnost korištenja znanja i vještina koje već imamo kako bismo riješili nove i neobične probleme ili situacije.
- Praktična inteligencija – uključuje sposobnost rješavanja svakodnevnih zadataka u stvarnom svijetu.

S gledišta umjetne inteligencije posebno je zanimljiva Steinbergova analitička inteligencija.

Umjetna inteligencija (strojna inteligencija) je relativno mlada znanost gledajući iz perspektive teme ovoga rada, a to je iz područja računalne znanosti. Pojam umjetna inteligencija se koristi za opisivanje strojeva koji oponašaju kognitivne funkcije koje ljudi povezuju s ljudskim i životinjskim umovima, kao što su učenje i rješavanje problema.

3.2. Povijest i razvoj umjetne inteligencije

Umjetna inteligencija kao znanstvena i zasebna disciplina je veoma mlada, ali je naslijedila mnoge zamisli, pristupe i tehnike iz drugih disciplina. Područje umjetne inteligencije je složena i opširna multidisciplinarna znanost jer ujedinjuje saznanja iz područja tehničkih, društvenih i bioloških znanosti. Najveći dio saznanja, ali i novih postignuća u umjetnoj inteligenciji dolazi kako napreduju discipline kao što su psihologija, logika odnosno matematika, računanje, psihologija to jest kognitivna znanost, biologija, neuroznanost te evolucija. Kroz primjenu umjetne inteligencije sve ove znanosti pa i mnoge druge doživljavaju svoj napredak.

Još davne 400 godine prije naše ere Sokrat (grčki filozof) je tražio algoritam za razlikovanje pobožnosti od ne pobožnosti. Aristotel je još jedan u nizu od filozofa, a formulirao je stil deduktivnog zaključivanja. Od matematičara možemo spomenuti Charles Stanhope i njegov uređaj koji je mogao riješiti mehaničke silogizme, numeričke probleme u logičkom obliku i elementarne probleme iz vjerojatnosti. George Boole je predstavio formalni jezik za logičko

zaključivanje poznat pod nazivom Booleova algebra. S područja računanja možemo istaknuti Williama Jovenona koji je napravio logički stoj koji je mogao obrađivati Booleovu algebru i Vennove dijagrame, a bio je u stanju rješavati logičke probleme brže od ljudi. I za kraj bi mogli spomenuti Alana Turinga koji je sa svojom vizijom imao najveći utjecaj. On je predstavio Turingov test, strojno učenje, genetičke algoritme i poboljšano učenje. Povrh toga svega predložio je ideju dječji program („Child Programme“) kojim je želio istaknuti da umjesto da pokušamo simulirati um odrasle osobe da radije pokušamo reproducirati onaj sličan djetetovom.

Iako umjetna inteligencija svoje korijene nailazi u drugim disciplinama pri čemu su neke veoma stare. Početak iliti rađanje umjetne inteligencije možemo smatrati razdoblje od 1943. do 1955. godine. Warren McCulloch i Walter Pitts su 1943. godine predložili model skupa umjetnih neurona pri čemu je svaki neuron mogao biti u stanju „uključen“ ili „isključen“. Pokazali su da se bilo koja računalna funkcija može izračunati pomoću neke mreže povezanih neurona. Donald Hebb 1949. godine pokazao je jednostavno pravilo ažuriranja kojim se može modificirati veza između neurona. To se pravilo prozvalo po njemu (Hebbiano pravilo) i ostalo je do dana danas kao jedno od najutjecajnijih modela. 1950. godine Marvin Minsky i Dean Edmonds su izgradila prvo računalo s neuronskom mrežom (SNARC). John McCarthy je 1956. na radionici u Dartmouthu prvi put službeno upotrijebio termin umjetna inteligencija pri čemu je ta ista radionica dala jasniju naznaku da umjetna inteligencija mora postati zasebno polje, a ne biti dio nekog drugog polja (matematike, teorija kontrole, teorija odlučivanja i slično).

U prvom desetljeću od kada je umjetna inteligencija postala zasebna grana stvorila su se velika očekivanja, a tome su pridonijeli prvi uspjesi koji su se tada ostvarili. S obzirom da su krajem 50-ih godina 20. stoljeća računala uglavnom radili aritmetičke operacije sve više od toga se činilo naprednim. James Slagleov je napravio program SAINT koji je mogao rješavati integracijske probleme zatvorenog oblika. Mreže dobivaju prva imena tako Bernard Widrow joj daje ime ADALINE, a Frank Rosenblatt mreži daje ime perceptron koje se do dana danas proteglo kada govorimo o jednoslojnim neuronskim mrežama. Ovaj uzlet i velika očekivanja, dijelom izazvan od strane tvoraca mreža sa svojim izjavama su stvorile dojam da će se industrija brzo promijeniti i napredovati, ali je ipak naišla na zid koji se zove stvarnost. Rani

sustavi su podbacili u rješavanju mnogih problema, a ti se problemi mogu svrstati u tri kategorije:

- 1) Rani programi nisu znali ništa o materiji koju su trebali rješavati.
- 2) Probleme koje su rješavali, rješavali su tako što su probavali različite kombinacije sve dok ne bi našli rješenje (brute force). Ovo je funkcioniralo samo kod ograničenih i manjih problema kakvi su se iz početka rješavali, no kada se prešlo na veće i složenije probleme pronalazak rješenja na takav način postao je gotovo nemoguć, a to se kasnije kroz teoriju računske složenosti i dokazalo.
- 3) Naišli su na fundamentalne granice nad osnovnim strukturama koje su se koristile za izradu inteligentnog ponašanja.

Glavni problem je bio u tome što su tražili kompletno rješenje i takav pristup se naziva slaba metoda. Alternativa je bila u domensko-specifičnom znanju za rješavanje uskih područja. Tako su nastali prvi ekspertni sustavi. Jedan od najpoznatijih primjera je primjer ekspertnog sustava u medicini. Sustav se zvao MYCIN i dijagnozirao je infekcije krvi, a ostvarivao je rezultate kao neki stručnjaci, a značajno bolje od mlađih doktora. 1980-ih umjetna inteligencija postaje industrija s velikim ulaganjima. Početkom 80-ih se ulagalo svega nekoliko milijuna da bi se do kraja 80-ih već milijarde ulagale te su bile uključene stotine firmi u izgradnji ekspertnih sustava. Prvi komercijalni ekspertni sustav, R1, pomagao je konfigurirati narudžbe za nove kompjuterske sustave i pritom je kompaniji štedio godišnje oko 40 milijuna dolara.

Osim što je 80-ih godina prošloga stoljeća umjetna inteligencija postala industrija dogodilo se još nekoliko važnih događaja koje su odredile samu budućnost te grane. Jedan od tih važnih događaja je bio unaprjeđenje i obnova metode učenja povratnim propagiranjem (back-propagation) koji je još otkriven 1969. godine nakon čega se algoritam počeo upotrijebljavati za rješavanje mnogih problema s učenjem. Uvelike se radilo na otkrivanju kako se neuronske mreže razlikuju od „tradicionalnih“ tehnika pri čemu su se počele uspoređivati s tehnikama iz statistike, prepoznavanje uzorka i strojnog učenja. Ovakav razvoj doveo je do razvoja nove industrije zvane rudarenje podatcima (data mining). Još jedan napredak je obilježio osamdesete, a to je nastanak Bayesovskog mrežnog formalizma. Ovaj pristup je nadišao mnoge probleme kod sustava vjerojatnosti za rasuđivanje iz 60-ih i 70-ih.

Bolje razumijevanje problema i njihovih svojstava složenosti u kombinaciji s povećanom matematičkom sofisticiranošću dovelo je do funkcionalnih istraživačkih programa i robusnih metoda pri čemu umjetna inteligencija dolazi pod znanstvenu metodu. To znači da se postavljena hipoteza mora biti podvrgnuta rigoroznom empirijskom iskustvu i rezultati se moraju statistički obraditi zbog njihove važnosti. Isto tako treba postojati mogućnost rekonstruiranja eksperimenta korištenjem podijeljenog repozitorija testnih podataka i koda. 90-ih godina 20. stoljeća zbog napretka umjetne inteligencije nad pod problemima počelo se opet gledati kako doći do cijelog agenta („whole agent“) to jest pojedini znanstvenici su se htjeli vratiti izvornim korijenima umjetne inteligencije, što je po Simon-ovima riječima, „stroj koji misli, uči i stvara“. Inače se takva umjetna inteligencija zove ljudska razina umjetne inteligencije (HLAI), a danas je dosta povezana s dijelom umjetne inteligencije koja se bavi generalnom umjetnom inteligencijom (AGI).

Tokom ovih zadnjih 70-tak godina razvoja umjetne inteligencije dosta veliki naglasak se stavlja na algoritme, ali zadnjih 20-tak godine se počinje prebacivati fokus s algoritama na podatke. Banko i Brill su 2001. godine su pokazali da se s povećanjem broja dostupnih podataka, u njihovom slučaju teksta i to s milijun riječi na milijardu, performanse povećaju do te razine da gotovo nema razlike koji smo algoritam izabrali. Jedan prosječni algoritam istreniran s 100 milijuna riječi će nadmašiti najbolji algoritam istreniran s 1 milijunom riječi. Drugi primjer koji pokazuje koliko na performanse algoritma utječe količina podataka je Haysov i Efrosov primjer iz 2007. godine gdje su za kreiranje maski na fotografiji s inicijalnih 10 tisuća povećali kolekciju na 2 milijuna fotografija. S rastom i raspršivanjem interneta problem količine podataka se smanjuje, ali i dalje se znanstvenici i inženjeri muče kako izraziti svo znanje koje sustav treba. [7]

3.3. Problematika i primjena umjetne inteligencije

Iako je u vrijeme nastanka ideje o umjetnoj inteligenciji bila osnovna ideja stvoriti stroj koji će se ponašati na intelligentan način, slično čovjeku, ubrzo se shvatilo da je problematika puno veća nego što se činilo na prvu ruku. Čak ni do dana danas, nekih 70-tak godina nakon nastanka umjetne inteligencije nismo uspjeli rekreirati u potpunosti neki intelligentni stroj koji će se

ponašati kao čovjek. U jednu ruku to je i razumno jer ako pogledamo koliko je trajala sama evolucija nas ljudi (govor je star oko 100,000 godina, prvi crteži oko 70,000 godina, moderno ljudsko ponašanje je nastalo prije 40,000 godina, a pismo je nastalo prije 5,000 godina) i koliko toga još ne znamo o ljudskom tijelu nije ni čudo što ne možemo napraviti stroj koji će se ponašati kao čovjek. Zato se krenulo s drukčijim pristupom. Umjesto da se radi jedan jedinstveni sustav, problem se podijelio na pod probleme i radi se na tome da se svaki pod problem riješi zasebno. Najviše pažnje su doatile idući aspekti:

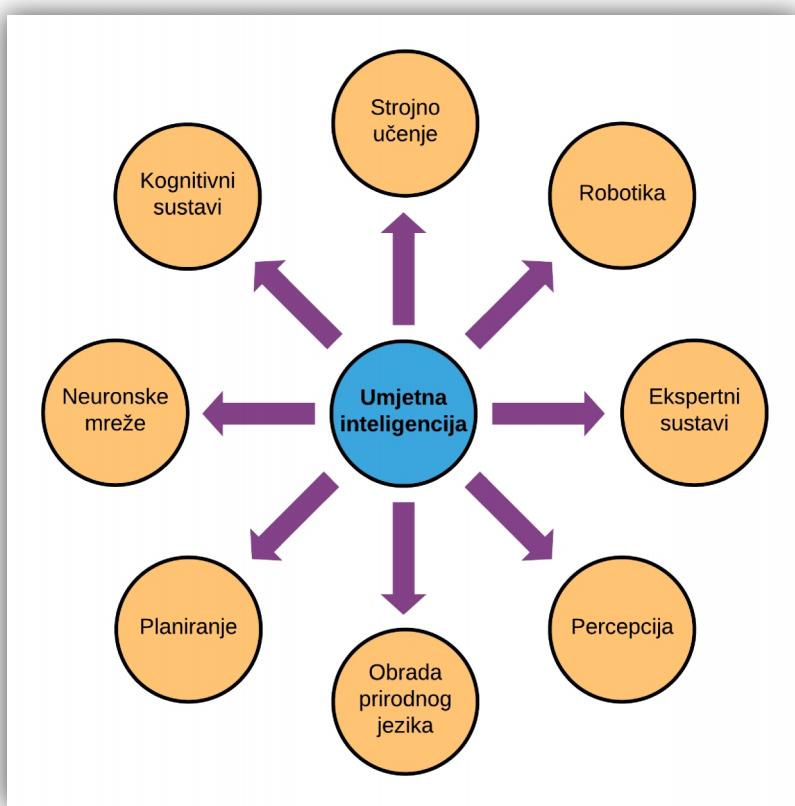
- Rasuđivanje i rješavanje problema,
- Predstavljanje znanja,
- Planiranje,
- Učenje,
- Obrada prirodnog jezika,
- Percepcija (kompjuterski vid),
- Pokret i manipulacija (robotske ruke i robotika),
- Društvena inteligencija,
- Generalna inteligencija.

Primjena umjetne inteligencije je raznovrsna i ulazi u sve aspekte ljudskih djelatnosti, a neke od najznačajnijih su:

- Zdravstvo (pomože liječnicima pri pronalasku pravog tretmana za rak, identificiranju raka kože, operiranju itd.),
- Financije i ekonomija (organiziranje poslova, održavanje knjigovodstva, ulagati u dionice, upravljati imovinom itd.),
- Vlada (alokacija resursa, predvidjeti scenarij, uvidi u velikim skupovima podataka, itd.),
- Video igre (bot za video igre kao što su šah, go, poker te u novije vrijeme borbeni AI u video igrarama kao Doom itd.),
- Vojska (vojne bespilotne letjelice),
- Oglašavanje (predvidjeti ili generalizirati ponašanje kupaca),
- Umjetnost (slika, video, zvuk, pjesma, animacija, itd.). [8]

3.4. Strojno učenje

Pod polja u umjetnoj inteligenciji je puno (slika 3.1.), a u ovom poglavlju ćemo se dotaknuti jednog od njih (strojno učenje) te kroz iduću cjelinu dati uvod u još jedno pod poglavlje (neuronske mreže) jer su nam ta dva pod poglavlja od velikog značenja jer se ovaj rad temelji na njima.



Slika 3.1. Pod polja umjetne inteligencije

Strojno učenje predstavlja granu u umjetnoj inteligenciji koja se bavi istraživanjem i primjenom modela koji rješavaju problem učenja na osnovu danih podataka koje će stroju ono predstavljati predznanje kojim će kasnije taj model rješavati nove, za stroj do tada neviđene, zadatke.

„Kažemo da je računalni program naučio iz iskustva E s obzirom na neki zadatak T i neke mjerne performanse P, ako se njegova performansa na zadatak T, mjerena kao performansa P, poboljšala s iskustvom E.“ Tom Mitchell, 1997. [8]

Kod strojnog učenja postoji mnogo različitih sustava, a njih bi mogli kategorizirati u neku od sljedećih kategorija na temelju sljedećih kriterija:

- Jesu li ili nisu li trenirani s obzirom na ljudski nadzor,
- Mogu li ili ne mogu li učiti inkrementalno u letu,
- Jeli radi tako što uspoređuju nove podatke sa starima ili umjesto toga pronađu uzorak u podatcima za treniranje (učenje) i onda izgrade prediktivni model.

3.4.1. Učenje s obzirom na ljudski nadzor

Učenje s obzirom na ljudski nadzor se odnosi koliko čovjek (programer) ima ulogu u navođenju što će i kako će algoritam naučiti. Algoritme u ovaj kategoriji dijelimo u četiri skupine:

- Nadzirano učenje,
- Ne nadzirano učenje,
- Polu nadzirano učenje,
- Poboljšano učenje.

Nadzirano strojno učenje za cilj ima mapirati funkciju na osnovu ulaznih varijabli to jest podataka x i izlaznih varijabli odnosno podataka Y .

$$Y = f(x) \quad (3.1.)$$

Time se postiže da za nove ulazne podatke x možemo predvidjeti izlaznu varijablu Y za taj podatak. Ovo se smatra nadziranim strojnim učenjem jer se proces učenja algoritma nadzire tako što su programeru (učitelju) dostupni i ulazni podatci x i izlazni podatci Y . Algoritam hranimo ulaznim podatcima x i on iterativno predviđa izlazne vrijednosti \hat{y} pri čemu ga programer usmjerava prema točnim izlaznim vrijednostima Y . Učenje se prekida kada algoritam postigne prihvatljivu razinu izvedbe. Nadzirani problemi učenja mogu se dalje grupirati u sljedeće probleme:

- Klasifikacija – slučaj kada ulazni podatak moramo grupirati u jednu od unaprijed definiranih klasa pri čemu mora postojati minimalno dvije klase, binarna klasifikacija, („pas“ ili „mačka“), a može i više, multinomna klasifikacija.
- Regresija – slučaj kada za dani ulazni podatak moramo predvidjeti stvarnu (kontinuiranu) vrijednost („dolari“, „visina“ itd.), matematički gledano vrijednost iz skupa realnih brojeva (\mathbb{R}).

Neki od popularnijih primjera algoritama u skupini za nadzirano učenje su:

- Linearna regresija,
- Logistička regresija,
- Stablo odlučivanje i slučajne šume,
- k-najbližih susjeda.

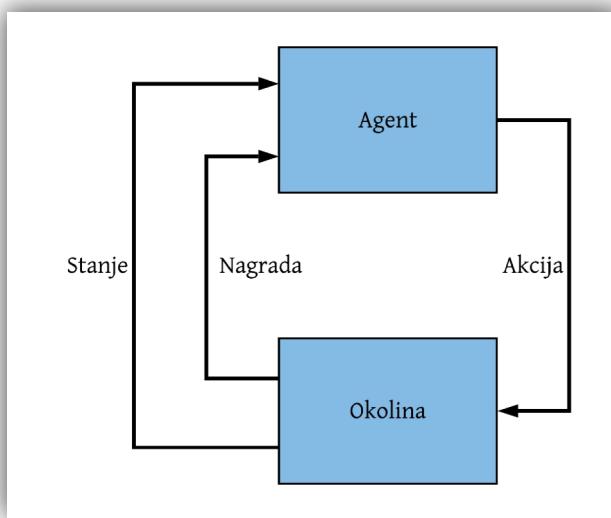
Kod ne nadziranog učenja cilj je modelirati temeljnu strukturu ili distribuciju u podatcima kako bi dobili bolji uvid u podatke s kojima radimo što znači da imamo samo ulazne podatke x bez odgovarajućih izlaznih varijabli (podataka). Za razliku od nadziranog učenja ovdje nema točnog odgovora tako da nam nije potreban učitelj da usmjerava algoritam već je algoritam prepušten samome sebi da otkrije i prikaže zanimljivu strukturu podataka. Primjena ne nadziranog učenja s odgovarajućim algoritmima je:

- I. Grupiranje – problem kod kojeg se želi otkriti inherentne grupacije u podatcima,
 - i. k-Means,
 - ii. Hjерархиjsка анализа кластера (HCA),
 - iii. Оčekivanje максимизације (EM).
- II. Vizualizacija i smanjenje dimenzionalnosti u podatcima,
 - i. Анализа главних компоненти (PCA),
 - ii. Kernel PCA,
 - iii. Локално линеарно уградње (LLE).
- III. Pravilo pridruživanja – kada u velikim skupovima podataka želimo otkriti veze među atributima.
 - i. Apriorno,
 - ii. Eclat.

Još jednu važnu ulogu ima ne nadzirano učenje, a to je detekcija anomalija. Detekcija anomalija se postiže tako što se algoritam trenira na takozvanim „normalnim“ podatcima, a zatim se počnu algoritmu davati nove instance (podatci) te algoritam može reći jeli nova instanca „normalna“ ili odskače od onoga na čemu se algoritam trenirao.

Polu nadzirano strojno učenje je vrsta učenja koja je nešto između nadziranog i ne nadziranog učenja. Sliči nadziranom strojnom učenju jer je samo jedan manji dio podataka označen Y , a veći dio podataka je neoznačen X što odgovara ne nadziranom strojnom učenju. Mnogi problemi iz stvarnog svijeta spadaju u ovu kategoriju, a razlog tomu je što može biti skupo i/ili dugotrajno za označavati podatke pri čemu nam možda čak trebaju i stručnjaci iz drugih domena. Neoznačeni podatci su jeftini i jednostavniji za prikupljanje i pohranjivanje. Što se tiče algoritama oni su najčešće kombinacija algoritama za nadzirano i ne nadzirano učenje, a jedan tak primjer je duboke mreže vjerovanja (DBMs).

Poboljšano učenje je jedno od najstarijih grana strojnoga učenja, a njezini počeci sežu još iz davnih 50-ih godina 20. stoljeća kada se upotrijebljavala u igrama (npr. TD-Gammon) i upravljanju strojevima. Poboljšano strojno učenje ima nešto drugačiji pristup nego preostale 3 metode učenja, a razlikuje se po tome što se sastoji od agenta koji se nalazi u okolini pri čemu agent daje ulaz u okolinu (akcija) za trenutno stanje (numerička reprezentacija onoga što agent promatra u određenome trenutku u okolini). Sama okolina vraća novo stanje i nagradu (kaznu) za radnju koju je agent napravio (slika 3.2.).



Slika 3.2. Metodologija poboljšanog učenja

Cilj poboljšanog strojnog učenja je za trenutno stanje odabrati optimalnu radnju (akciju) koja će maksimizirati dugoročnu očekivanu nagradu od strane okoline. Agent uči tako da optimalno interaktira u stvarnom vremenu s okolinom koristeći vremenski odgođene oznake (nagrada). Kroz interakciju s okolinom agent će naučiti politiku koja će vratiti akciju za dano stanje s najvišom nagradom. Neki od algoritama koji se koriste kod poboljšanog učenja su:

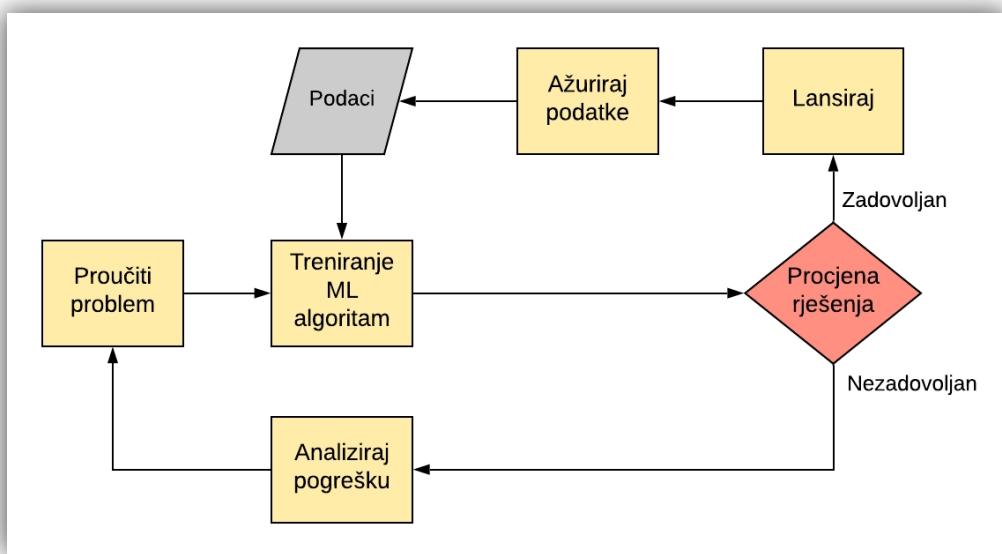
- Monte Carlo,
- Q-learning (state-action-reward-state),
- SARSA (state-action-reward-state-action),
- DQN (Deep Q Network),
- A3C (Asynchronous Advantage Actor-Critic).

3.4.2. Izvan mrežno učenje i učenje na mreži

Drugi kriterij po kojem se razlikuju algoritmi u strojnom učenju je da li sustav može ili ne može učiti inkrementalno iz toka nadolazećih podataka pa tako razlikujemo algoritme koji uče izvan mrežno i one algoritme koji uče na mreži. Svakako treba napomenuti da nazivi učenje na

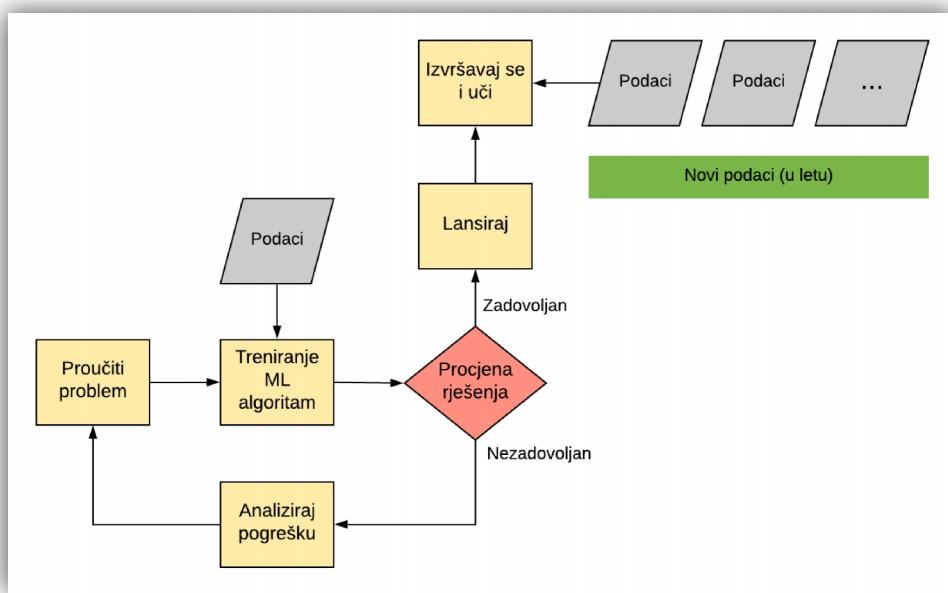
mrežni i izvan mrežno učenje nema veze s pristupom na internet nego kako je spomenuto u prvoj rečenici jeli algoritam sposoban učiti inkrementalno iz nadolazećeg toka podataka ili ne.

U izvan mrežnom načinu učenje algoritam je istreniran na cijelom skupu podataka i testiran prije nego je pušten u rad. Ovakav sustav u radu sada samo primjenjuje što je naučio. U slučaju da želimo da sustav nauči nešto novo to jest prilagodi se novim trendovima tada treba istrenirati potpuno novu verziju sustava od nule na cijelom skupu podataka (stari podatci plus novi dodani podatci). Kada je sustav istreniran i zadovoljni smo rezultatima koje postiže tada se stari sustav zaustavlja i zamjenjuje s novim (slika 3.3.). Izvan mrežni način učenja je jednostavan i često dobro radi, ali ima nekoliko nedostataka. Jedan od nedostataka je ako je za prvo treniranje trebao cijeli jedan dan onda će vam na novim podatcima trebati i više jer smo prvobitni skup podataka proširili novim i tako sa svakom novom inačicom koju želimo nadograditi će tražiti sve više i više vremena za treniranje. Skupovi podataka mogu biti jako veliki (veći od 1 TB) za što vam onda treba dosta računalnih resursa te ako svako malo trebate nanovo istrenirati sustav to može biti skupo i novčano i vremenski.



Slika 3.3. Izvan mrežno učenje

Za razliku od izvan mrežnog načina učenja, algoritmi s sposobnošću učenja na mreži uče inkrementalno davajućim sekvencijalno manje skupove podataka. Svaki novi korak treniranja je brz i jeftin i stoga algoritam može učiti o novim podacima (trendovima) u letu (slika 3.4.). Učenje na mreži je zgodan kada treba konstantno učiti i adaptirati se na nove trendove (cijene dionica, autonomna vozila), kada skup podataka ne stane u glavnu memoriju jednog stroja (ovakvo učenje se još zove i izvan temeljno učenje). Postoji jedan bitan parametar kod učenja na mreži, a to je stopa učenja. Stopa učenja nam govori koliko brzo će se algoritam prilagođavati novim podatcima. Ukoliko je stopa učenja veliko tada će se algoritam brzo prilagođavati novim podatcima, ali tada će isto tako brzo zaboravljati staro znanje.



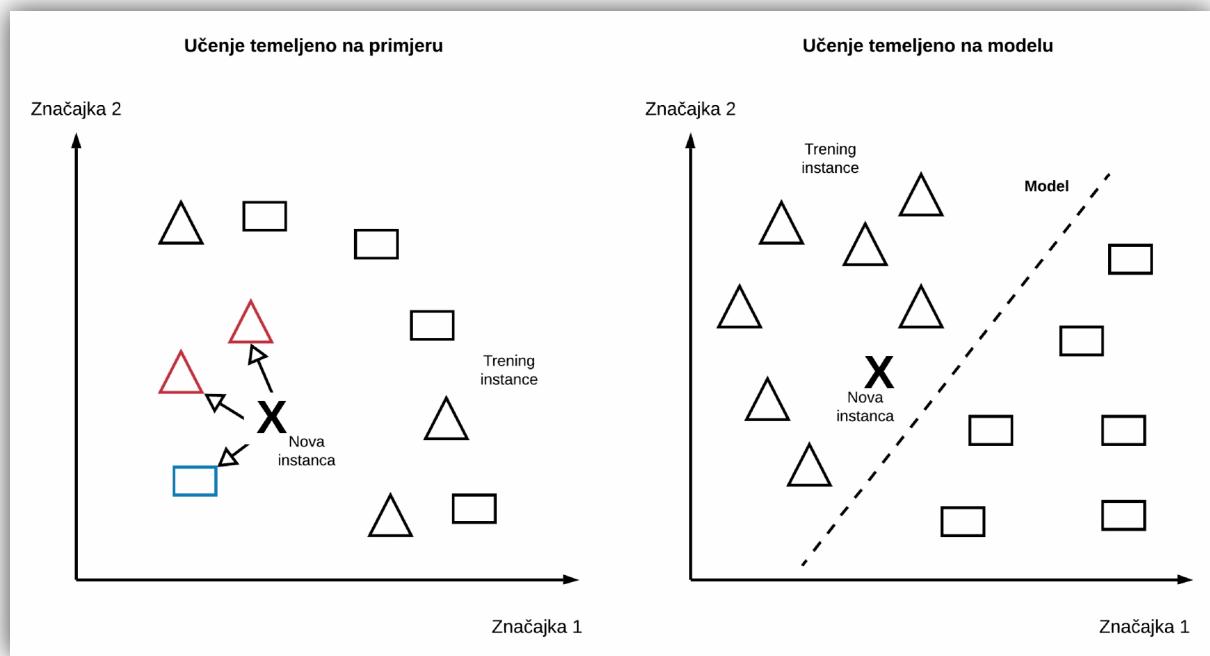
Slika 3.4. Učenje na mreži

3.4.3. Učenje na temelju primjera i učenje na temelju modela

Zadnja kategorija koja će biti opisana u ovom radu, a po kojoj se algoritmi mogu razvrstati je učenje na temelju primjera te učenje na temelju modela. Svakako se mora napomenuti da ovih kategorija ima još dosta, ali ove tri kategorije su najistaknutije i veliki broj sustava se može razvrstati u ove kategorije. Ova kategorija nam govori kako sustavi za strojno učenje

generaliziraju nove, dosad nikad viđene instance. Imati dobre performanse sustava nad podacima za treniranje je dobro, ali nije dovoljno stoga je pravi cilj postići dobre rezultate na novim podacima to jest instancama.

Učenje na temelju primjera ponekad se još naziva učenje temeljno na memoriji jer kategorija kod koje sustavi generaliziraju na način da spremi u memoriju primjere iz skupa za treniranje, a dolaskom novih instanci (primjera) uspoređuju ih s viđeniminstancama iz treninga koji je u ovom slučaju bio samo memoriranje trening primjera. Ovakvi sustavi ne nauče nikakvu eksplisitnu funkciju zbog toga se metode koje spadaju u ovu kategoriju ponekad nazivaju lijene metode učenja, a generaliziraju nove primjere korištenjem mjeri udaljenosti kao što je Minkowska udaljenost koja je samo generalizirani oblik Euklidske udaljenosti i Manhattan udaljenosti zatim kosinusova udaljenost, Mahalanobisova udaljenost i slično. Tijekom izrade modela definira se koja će se mjeri koristiti, a ovisno o samom modelu će ovisiti na koji će je način model iskoristiti. Bez obzira na koji način je bude koristila novi primjer (instanca) će pripasti onoj klasi koja s obzirom na druge instance bude imala najmanju udaljenost (slika 3.5.).



Učenje na temelju modela je drugi način na koji modeli mogu generalizirati nove instance. Ako model generalizira na osnovu modela tada stroj tijekom treninga treba izgraditi (naučiti) model to jest skup pretpostavki o problemskoj domeni, izražena preciznim matematičkim oblikom, koji će se kasnije koristiti za stvaranje rješenja (slika 3.5.). [9]

4. NEURONSKE MREŽE

4.1. Umjetna neuronska mreža

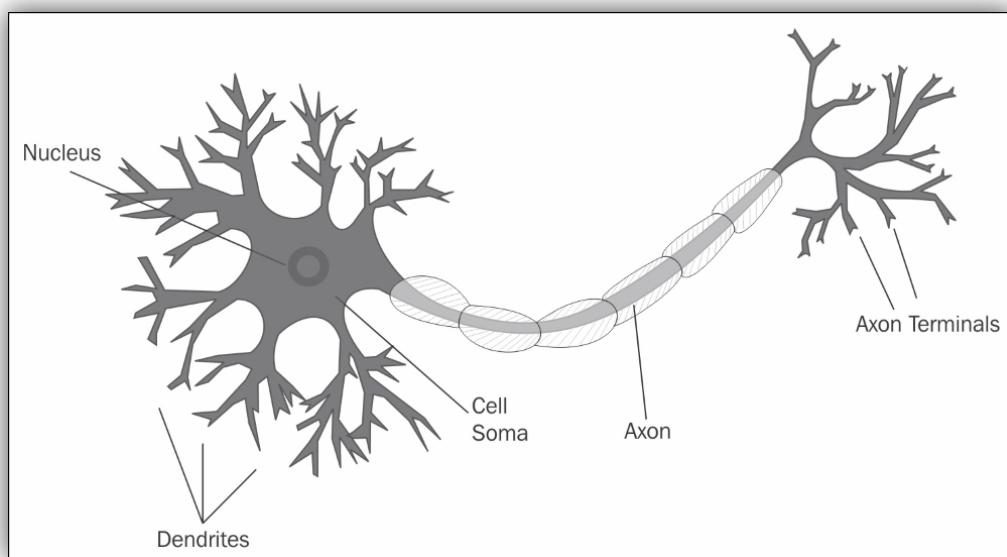
Umjetnom inteligencijom se pokušava oponašati kognitivne funkcije ljudi (životinja) dok strojno učenje imitirati sposobnost učenja, a same kognitivne funkcije se odvijaju u mozgu. Umjetne neuronske mreže su upravo nastale iz inspiracije da se u potpunosti simulira ljudski mozak što ćemo kasnije uočiti kada budemo usporedili umjetnu neuronsku mrežu s mrežom živčanih stanica (neurona) biološkog centralnog živčanog sustava. Izgradnjom umjetnih neuronskih mreža želi se nadići tradicionalni (digitalni ili analogni) računalni sustavi koji služe za zamjenu, poboljšanje ili ubrzanje procesorske moći ljudskog mozga. Osim same simulacije mozga postoji još nekoliko razloga zašto se danas dosta pažnje posvećuje umjetnim neuronskim mrežama. Umjetne neuronske mreže nam omogućavaju korištenje jednostavnih računalnih operacija kao što su zbrajanje, množenje i temeljni logički elementi za rješavanje složenih, matematički loše definirane probleme, nelinearne probleme ili stohastičke probleme dok konvencionalni algoritmi za rješavanje ovakvih problema trebaju koristiti skup složenih jednadžbi. Glavna prednost koja se vidi u umjetnim neuronskim mrežama je ta što omogućuje vrlo nisku razinu programiranja kako bi se riješili problemi koji imaju neke od sljedećih značajki:

- Ne analitički,
- Nelinearni,
- Ne stacionarni,
- Stohastički.

Ne samo da bi se rješavali problemi ovakvih značajki već bi se to radilo na način samoorganiziranja koji se odnosi na široki dijapazon problema bez ponovnog programiranja ili drugih poremećaja u programu.

Biološka neuronska mreža građena je od živčanih stanica takozvanih neurona (slika 4.1.). Sama stanica koja uključuje samu jezgru neurona je mjesto gdje se događa gotova sva radnja. Neuronska aktivnost započeta u jednom neuronu putuje prema drugome u obliku električnog

okidača preko aksona. Električni okidač je zapravo elektrokemijski proces naponske izmjene iona duž aksona i difuzije molekula neurotransmitera kroz membranu preko sinaptičkog raskoraka (živčane stanice nisu direktno vezane jedna s drugom). Svaki neuron može sadržavati nekoliko stotina dendrita koji su aktivni u recepciji impulsa od strane drugih neurona, a s druge strane taj isti neuron može sadržavati na stotine aksonskih terminala za prosljeđivanje impulsa prema drugim neuronima. Svakako je neophodno naglasiti da nisu sve međusobne veze jednakо ponderirane drugim riječima neke veze imaju veći prioritet (veću težinu) od drugih i obratno neke veze imaju manji prioritet (težinu) od drugih. Same umjetne neuronske mreže se temelje na ovakvoj logici povezivanja neurona i ponderiranje veza.



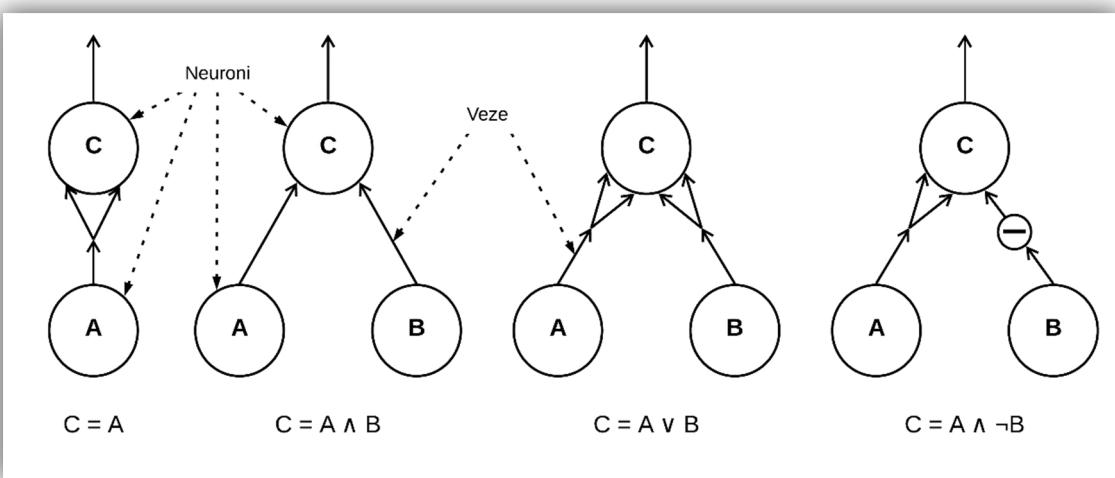
Slika 4.1. Biološka živčana stanica (neuron) [10]

Warren McCulloch i Walter Pitts su još 1943. godine dali osnovna načela umjetnih neuronskih mreža u obliku pet pretpostavki, a oni su:

- 1) Aktivnost neurona.
- 2) Određeni broj sinapsi veći od 1 mora biti pobuđen unutar zadanog intervala da bi neuron bio pobuđen.
- 3) Jedino značajno kašnjenje unutar neuronskog sustava trebalo bi biti sinaptičko kašnjenje.

- 4) Aktivnost bilo koje inhibitorne sinapse apsolutno sprječava pobuđivanje neurona u to vrijeme.
- 5) Struktura mreže međupovezivanja se s vremenom ne mijenja.

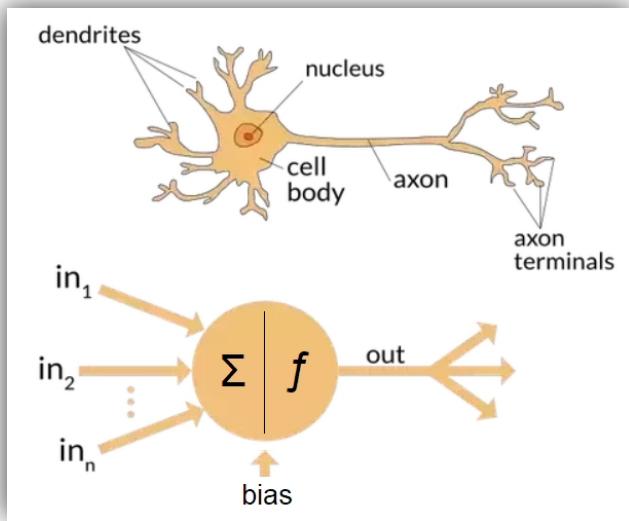
Ovo su jedni od najranijih postavljenih principa, a od tada pa do danas neuronske mreže su dosta napredovale pa stoga neki od ovih principa se ne odnose na najmodernije strukture mreža. Uz ovakve jednostavne principe moguće je graditi mrežu od umjetnih neurona koji mogu izračunati osnovne logičke prijedloge, a njihovom se kombinacijom mogu izračunati složeni logički izrazi (slika 4.2.).[11]



Slika 4.2. Arhitekture umjetnih neuronskih mreža koje obavljaju osnovne logičke operacije

4.2. Perceptron i višeslojni perceptron

Perceptron je neuralni računski model koji posjeduje temeljnu strukturu živčane stanice pri čemu sadrži nekoliko težinski različitih ulaza koji su povezani s prethodnim neuronskim izlazima te neuronskim izlazom koji se može povezati na druge stanice (slika 4.3.). Perceptron je jedan od najjednostavnijih arhitektura umjetnih neuronskih mreža osmišljena od strane Franka Rosenblatta 1958. godine te predstavlja jednostruku neuronsku mrežu za nadzirano učenje binarnih klasifikatora.



Slika 4.3. Biološki neuron naspram umjetnim neuronom (perceptron) [12]

Moderniji perceptron je baziran na umjetnom neuronu zvan logička jedinica praga (TLU) ili još ponekad i linearna jedinica praga (LTU) jer prvobitni perceptroni su se koristili za logičke (binarne) izraze i trebalo je uvesti neke promjene kako bi ulazne i izlazne vrijednosti mogli biti brojevi pri čemu je svakom ulazu bila pridijeljena određena težina.

Sami perceptron sastoji se od m ulaza ($x_1, x_2, x_3, \dots, x_m$) koji su vagani s pripadajućim težinama ($w_1, w_2, w_3, \dots, w_m$). Perceptron sumira umnožak j-tog ulaza s j-tom težinom. Ne smijemo nikako izostaviti bias-ov neuron čija se vrijednost prosljeđuje u svaki perceptron pri čemu se njegova vrijednost uglavnom postavlja na 1 (jednadžba 4.1.). Njega se shematski može na dva načina prikazati. Jedna je opcija nadodati nulti ulaz x_0 i predodrediti ga kao bias-ov, a drugi način je naznačiti i prikazati ga kao posebni ulaz (b_k) u perceptron (slika 4.4.).

$$v_k = w_{k1}x_1 + w_{k2}x_2 + \dots + w_{km}x_m + b_k = \sum_{j=1}^m w_{kj}x_j + b_k = \mathbf{w}^T \mathbf{x} + b_k \quad (4.1)$$

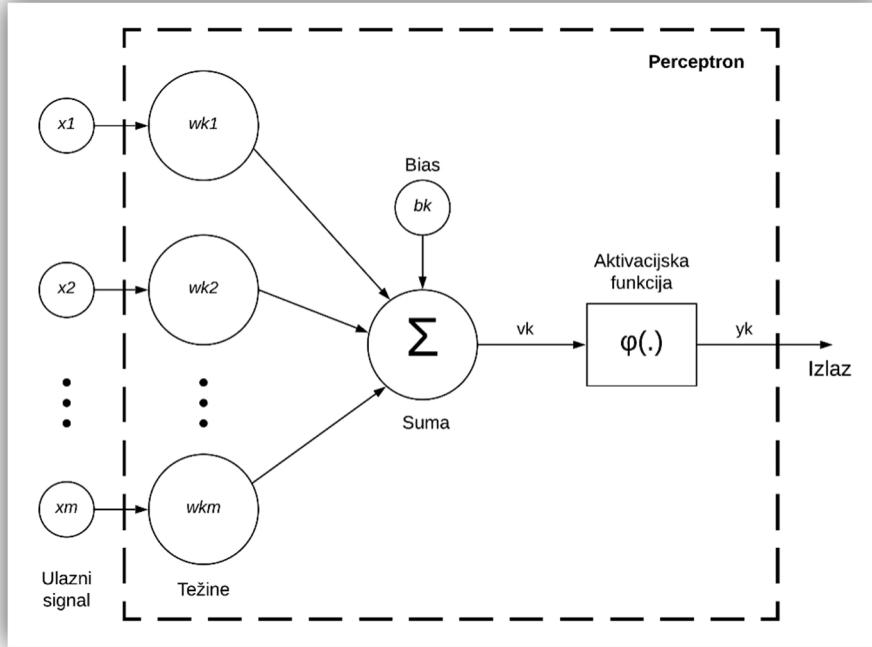
Gdje je:

v_k zbroj biasove vrijednosti sa sumom umnožaka ulaza s pripadajućim težinama

w_k težina j veze

x_j vrijednost j -tog ulaza

b_k Bias-ova vrijednost



Slika 4.4. Struktura perceptrona

Vrijednost izlaza iz perceptrona se dobije kada se u aktivacijsku funkciju unese vrijednost težinske sume koja se potom prenosi sljedećim umjetnim neuronima (jednadžba 4.2).

$$y_k = \varphi(v_k) \quad (4.2)$$

Gdje je:

y_k vrijednost izlaz iz perceptrona

$\varphi()$ aktivacijska funkcija

Rosenblattov neuron je inspiriran Hebbovim pravilom. Donald Hebb objašnjava da biološki jedan neuron često aktivira drugi neuron te da veza između ta dva neurona postaje sve jača što više puta jedan neuron aktivira drugi. Siegrid Löwel je ovu ideju sažeо u frazu: „Stanice koje

se aktiviraju zajedno, vežu se zajedno“. Ovo pravilo je kasnije postalo poznato kao Hebbovo pravilo ili kao Hebbovo učenje i do dana danas je ostalo kao jedno od najrelevantnijeg i najkorištenijeg pravila u umjetnim neuronskim mrežama. Naravno da nije dovoljno da jedan neuron aktivira drugi neuron i da njihova veza jača kada su u pitanju umjetne neuronske mreže zbog toga se mora uključiti i greška koju radi mreža. Stoga ako je mreža dala krivi izlaz tada neće doći do jačanja veza između neurona koji su se aktivirali da bi dali ovakav izlaz. Kada govorimo o umjetnim mrežama i jačanju veza među njima govorimo o ažuriranju težina w . Da bi se veze ažurirale moramo hraniti perceptrone jednom po jednom trening instancom (primjerom). Za svaki primjer koji se donese na ulaz neurona, on daje svoju prognozu. Za svaki izlaz za koji je neuron donio točnu prognozu pojačati će se težine veza od ulaznih podataka koje su pridonijele ispravom predviđanju (jednadžba 4.3).

$$w_{i,j}^{(s)} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i \quad (4.3)$$

Gdje je:

$w_{i,j}$ težinska veza između i -tog ulaznog neurona i j -tog izlaznog neurona

x_i i -ta ulazna vrijednost od trenutnog primjera

\hat{y}_j izlaz iz j -tog izlaznog neurona za trenutni primjer

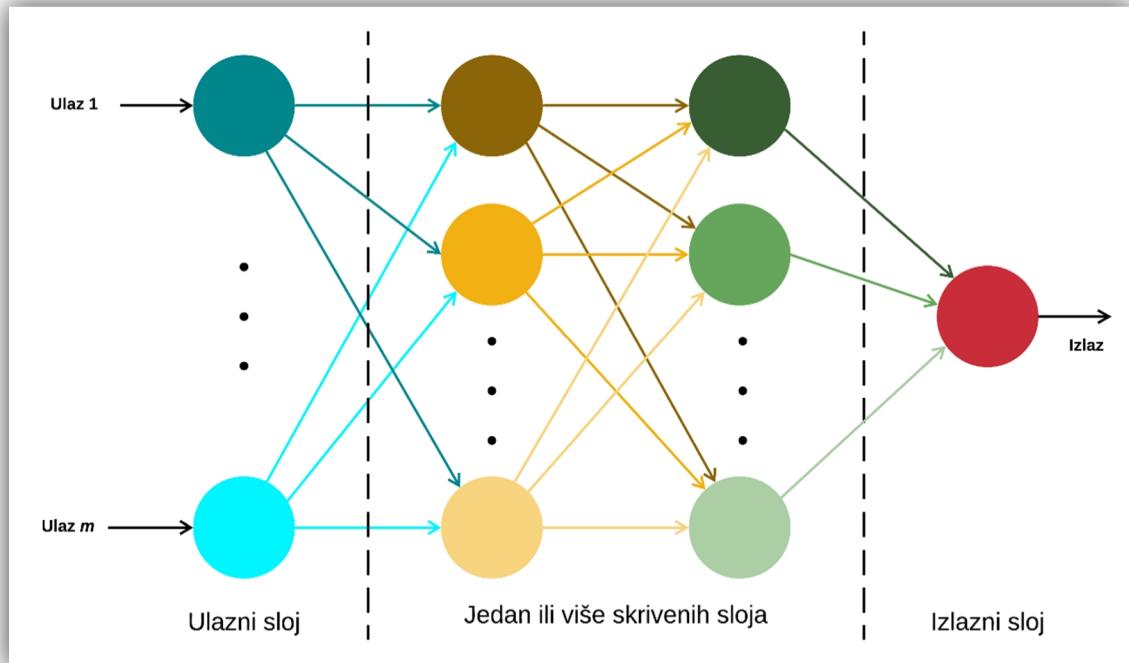
y_j ciljni izlaz iz j -tog izlaznog neurona za trenutni primjer

η stopa učenja

Granica odluke za svaki izlaz iz neurona je linearan stoga su perceptroni nemoćni kada su u pitanju učenje složenih uzoraka. Ukoliko su trening instance linearno odvojive Rosenblatt je demonstrirao da će njegov perceptron konvergirati prema rješenju drugim riječima perceptron će naučiti linearu granicu koja razdvaja primjere. Ovo se zove perceptronov teorem konvergencije. Marvin Minsky i Seymour Papert su 1969. godine ukazali na nekoliko nedostataka kod perceptrona, a ponajprije činjenica da perceptron nije sposoban riješiti banalni problem kao što je ekskluzivno ILI (XOR). Nadalje mnogi primjeri klasifikacije nisu linearne razdvojive i na takvim je problemima isto tako perceptron bio neuspješan i ono što se u početku mislilo da će biti veliki uspjeh ipak se pokazalo kao razočaranje. No ipak tu se nije stalo i nakon određenog perioda našlo se rješenje kako riješiti neke od manjkavosti perceptrona. Rješenje je

u slaganju više uzastopnih slojeva perceptrona pri čemu svaki sloj sadrži jedan ili više perceptrona. Ovakav skup neurona rezultira umjetnom neuronskom mrežom koja se zove višeslojni perceptron (MLP).

Višeslojni perceptron se sastoji od tri ili više slojeva pri čemu se prvi sloj u koji dolaze podaci se naziva ulazni sloj. Zadnji sloj koji nam ujedno daje i rezultat se naziva izlazni sloj, a svi ostali slojevi koji se nalaze između ulaznog i izlaznog sloja naziva se skriveni sloj i za razliku od ulaznog i izlaznog sloja može ih biti više od jednoga (slika 4.5.). Ako umjetna neuronska mreža sadrži dva ili više skrivenih slojeva tada je njezino ime duboka neuronska mreža (DNN).



Slika 4.5. Duboka neuronska mreža

Može se uočiti sa slike 4.5. da signal putuje samo u jednom smjeru i to od ulaza prema izlazu stoga se neuronske mreže s ovakvom arhitekturom nazivaju unaprijedna neuronska mreža (FNN). [9]

4.3. Aktivacijska funkcija

Do sada se nekoliko puta spomenula aktivacijska funkcija, a može se i vidjeti na slici 4.4. gdje se ona nalazi unutar same strukture perceptronra. Aktivacijska funkcija predstavlja matematičku jednadžbu koja određuje vrijednost izlaza iz neurona u ovisnosti o sumi v_k drugim riječima aktivacijsku funkciju možemo razumjeti jednostavno pitajući se koja je vrijednost na y za definiranu krivulju, aktivacijsku funkciju, za dani x .

Osnovne zadaće aktivacijske funkcije su:

- Određuje hoće li se neuron aktivirati („ispaliti“) ili ne na temelju vrijednosti ulaza u neuron i postavljenome pragu ako se radi o korak funkciji ili mapiranje ulaznog signala u izlazni signal za ostale funkcije.
- Normalizira izlaz svakog neurona u određenom rasponu koji ovisi o odabranoj aktivacijskoj funkciji.
- Moraju biti računski učinkoviti jer se izračunavaju za svaki primjerak podatka u svakom neuronu kojih može biti na desetke tisuća ili čak milijune unutar jedne umjetne neuronske mreže.
- Uvode nelinearna svojstva u umjetne neuronske mreže što dozvoljava rad s podatcima koji u sebi imaju prisutne nelinearne uzorke, a takvih skupova podataka je veliki.

Aktivacijske funkcije možemo svrstati u jedno od idućih triju kategorija:

1. Korak funkcija
2. Linearna funkcija
3. Nelinearna funkcija

Iz kategorije korak funkcije najčešće korištena funkcija je binarna korak funkcija. Koristi se za binarnu klasifikaciju na temelju praga pri čemu je sama funkcija vrlo jednostavna. Ukoliko vrijednost prelazi neki predefinirani prag neuron će aktivirati, a inače će ga ostaviti neaktivnog. Ova funkcija je više teorijska jer derivacija ove funkcije je nula stoga nije korisna u umjetnim

neuronskim mrežama koje koriste algoritam s unatražnim rasprostiranjem. Funkcije koje spadaju u kategoriju linearnih funkcija rješavaju neke od nedostataka koje imaju korak funkcije. Jedan od glavnih nedostataka je da derivacije funkcija nisu jednake nula što je onemogućavalo poboljšanje (učenje) umjetne neuronske mreže. Linearna funkcija je definira:

$$f(x) = a + b \quad (4.4)$$

Gdje su:

a – konstanta (nagib pravca)

b – konstanta ($f(x)$ presijecište za $x = 0$)

Derivacijom funkcije dobijemo konstantu a što opet ne nije dobro jer ta vrijednost ne ovisi o ulaznoj vrijednosti x što znači da bez obzira koliko slojeva imali u neuronskoj mreži posljedni sloj će biti linearna funkcija prvog sloja.

U novije vrijeme umjetne neuronske mreže koriste nelinearne aktivacijske funkcije. One omogućavaju mreži izračunati i naučiti bilo koju funkciju. Isto tako daju značaj algoritmu s unatražnim rasprostiranjem i kreiranje umjetnih mreža s više slojeva neurona. Sigmoidna aktivacijska funkcija je primjer jedne nelinearne funkcije (jednadžba 4.5).

$$f(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} \quad (4.5)$$

e predstavlja Eulerov broj (konstanta) čija vrijednost zaokružena na tri decimale iznosi $e = 2.718$. Funkcija je glatka i kontinuirano diferencirana što sprečava „skokove“ u izlaznim vrijednostima, a same izlazne vrijednosti se nalaze u rasponu od nula do jedan. Postoji nekoliko nedostatka, a najznačajniji su da izlazi nisu centrirani na nulu, računski zahtjevna te postoji problem nestajanja gradijenta (za vrlo visoke ili vrlo niske vrijednosti x praktički nema promjene u izlazu što usporava učenje i postizanju točnog predviđanja). Rješenje za centriranje na nulu može se postići ako se umjesto sigmoidne funkcije odabere funkcija hiperbolični tangens (jednadžba 4.6). Sve ostale prednosti i nedostatke tanh funkcija dijeli sa sigmoidnom funkcijom.

$$f(x) = t \quad h(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (4.6)$$

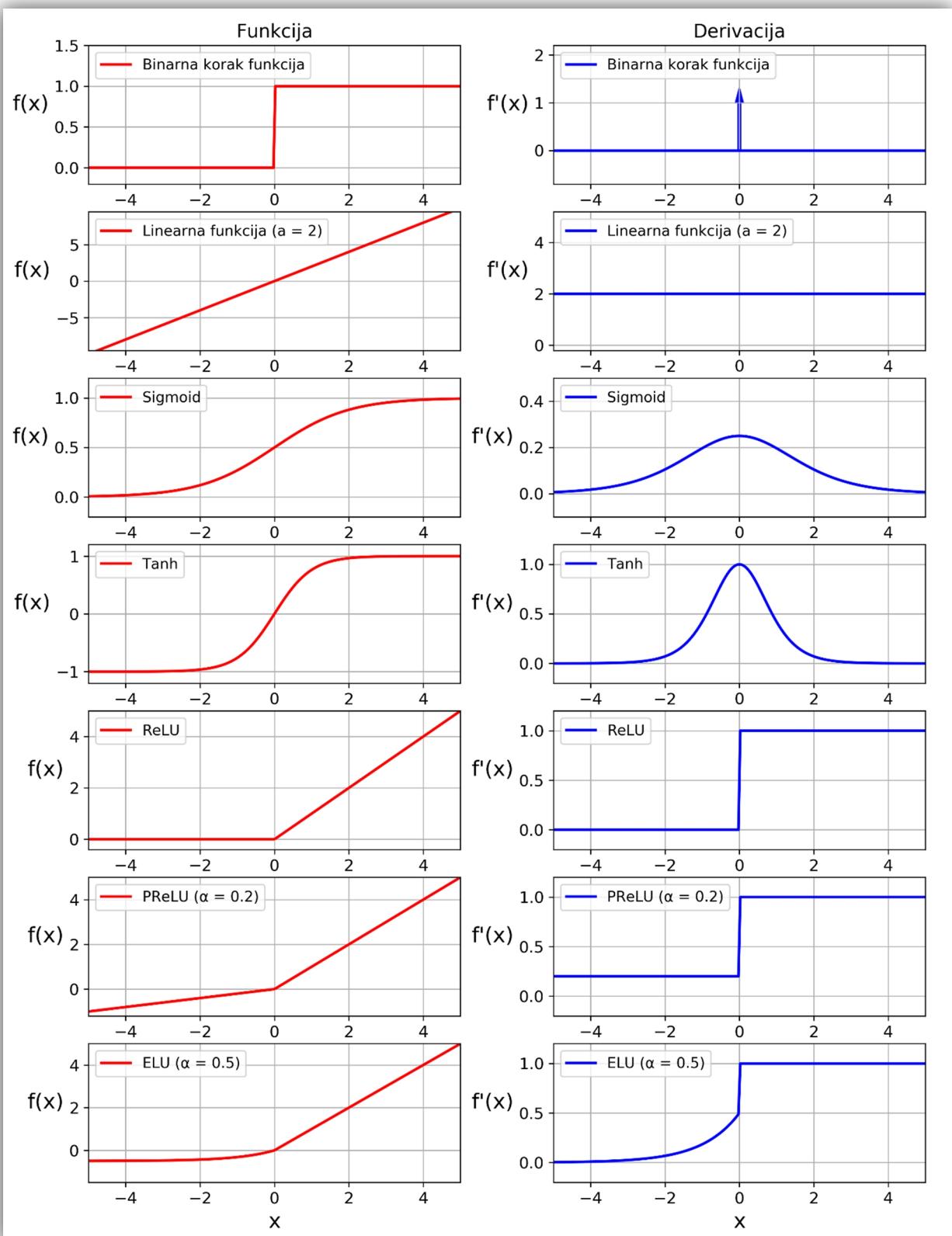
Iduća funkcija iako jednim dijelom izgleda kao linearna funkcija ipak je nelinearna funkcija, a funkcija je poznata pod imenom ispravljena linearna jedinica (ReLU) (jednadžba 4.7). ReLU je najčešće korištena aktivacijska funkcija i kada programer ne zna koju aktivacijsku funkciju odabrati preporuka je da se krene s ReLU funkcijom.

$$f(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases} \quad (4.7)$$

Funkcija je računski jako efikasna što mreži omogućuje brzo konvergiranje, jedini nedostatak funkcije umirući ReLU. Umirući ReLU znači da tijekom učenja neki neuroni mogu umrijeti drugim riječima njihov izlaz je uvijek nula. Jednom kada neuron počne na svom izlazi davati nulu jako mala vjerojatnost je da će ikad više dati išta drugo. Rješenje ovoga problema se nailazi u redizajniranim varijantama ReLU kao što su cureći ReLU (Leaky ReLU), nasumično cureći ReLU (RReLU) te parametarski ReLU (PReLU) (jednadžba 4.8) [9].

$$f(\alpha, x) = \begin{cases} \alpha, & x < 0 \\ x, & x \geq 0 \end{cases} \quad (4.8)$$

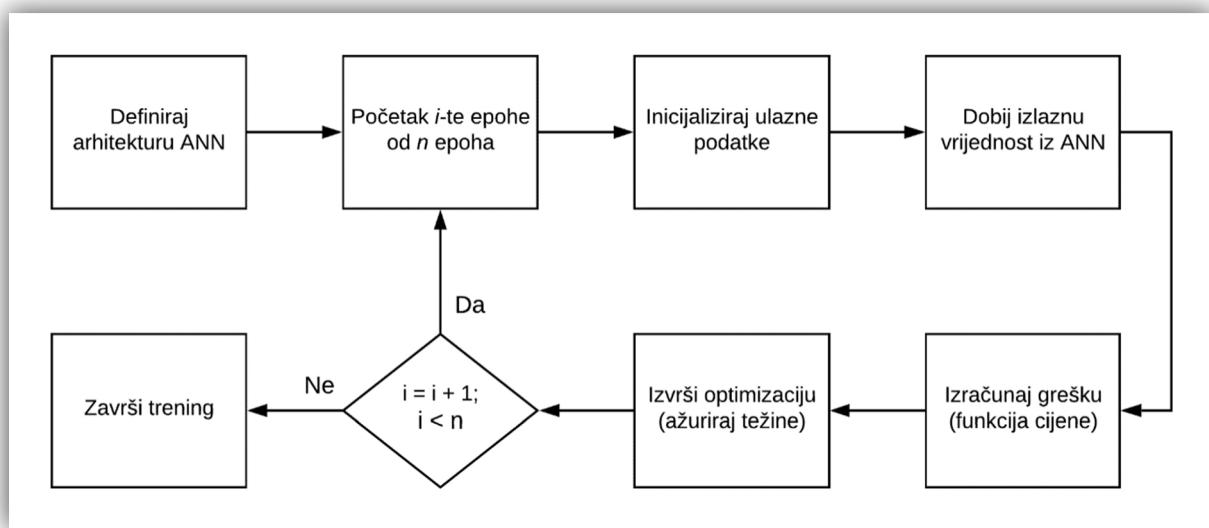
Na slici 4.6. se mogu vidjeti grafikoni krivulja većine funkcija koje su ovdje opisane te pripadajući derivacijske krivulje. [13]



Slika 4.6. Primjer nekolicine aktivacijskih funkcija i njihove derivacije

4.4. Treniranje i optimizacija

Umjetne neuronske mreže se treniraju tako što se koristi optimizacijski pristup drugim riječima postupno moramo ugađati (mijenjati) parametre modela koji se mogu naučiti (težine modela) tako da minimiziramo vrijednost funkcije cijene tijekom treninga. Ovaj proces ugadanja parametara dok mreža ne postane dovoljno dobra da rješava specifični (zadani) problem je iterativan što znači da napreduje korak po korak s malim nadogradnjama težina modela svakom novom iteracijom (slika 4.7.). Također treba osigurati da se mreža dobro generalizira kako bi bila bolja u predviđanju za podatke koje do sada nije vidjela. Funkcija cijene mjeri koliko je loša učinkovitost umjetne neuronske mreže drugim riječima funkcija cijene vraća grešku između predviđene vrijednosti u usporedbi sa stvarnom vrijednošću. Drugi način mjerjenje učinkovitosti je pomoću funkcije korisnosti ili funkcije sposobnosti koja mjeri koliko je dobra učinkovitost. U umjetnim neuronskim mrežama najčešće se koristi funkcija cijene. Kada se radi o umjetnim neuronskim mrežama često se može naići još i na funkciju gubitka. Funkcija gubitka mjeri pogrešku nad jednim uzorkom dok je funkcija cijene prosječna pogreška u broju uzoraka nad danim skupom podataka.



Slika 4.7. Dijagram tijeka treniranja neuronske mreže

Kada govorimo o arhitekturi neuronskih mreža, treniranju, optimizaciji i reguliranju često se mogu čuti pojmovi kao što su parametar i hiperparametar.

Parametar modela je konfiguracijska varijabla koja je unutarnja za model i čija se vrijednost može procijeniti iz podataka. Karakteristike parametara modela su:

- Oni su potrebni kada model radi predviđanje.
- Njihove vrijednosti definiraju performanse modela nad zadanim problemom.
- Oni se procjenjuju ili se uče iz podataka.
- Programer ih često ne postavlja ručno.
- Često se spremaju kao dio naučenog modela.

Primjeri parametara su težine u umjetnim neuronskim mrežama, vektori podrške u strojnom vektoru podrške (SVM), koeficijenti u linearnoj regresiji ili logističkoj regresiji i tako dalje.

Hiperparametri modela predstavljaju vanjsku konfiguraciju modela i njihova se vrijednost ne može procijeniti iz podataka. Karakteristike hiperparametra modela su:

- Često se koriste u procesima koji pomažu u procjeni parametara modela.
- Specificirani su od strane programera.
- Ponekad se mogu postaviti pomoću heuristike.
- Ne možemo unaprijed znati najbolju vrijednost hiperparametra za zadani problem.

Neki od primjera hiperparametara su stopa učenja, broj skrivenih slojeva, aktivacijska funkcija kod neuronskih mreža, C i sigma kod SVM-a, k kod k-najbližih susjeda i tako dalje.

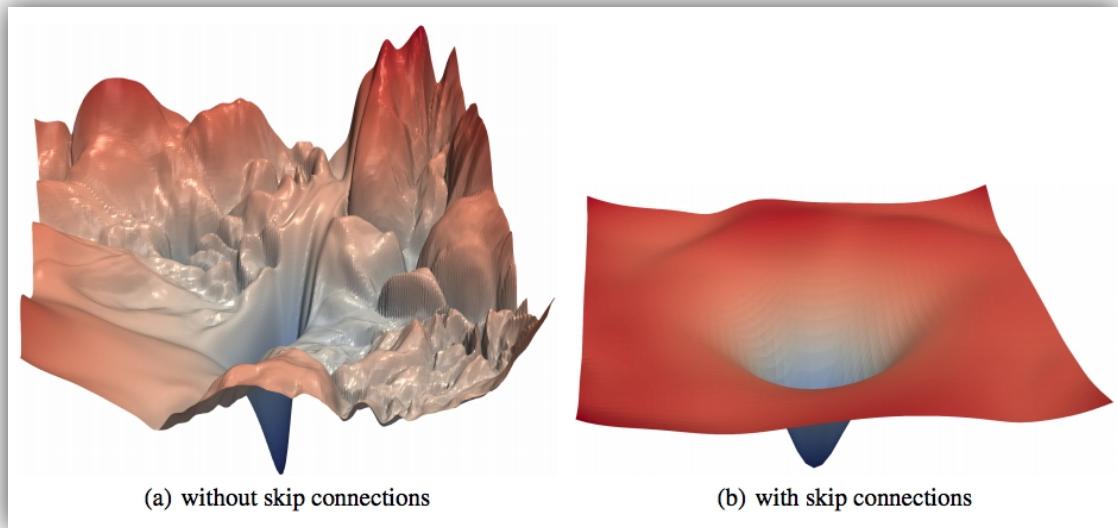
4.4.1. Funkcija cijene

Funkcija cijene mjeri matematičku udaljenost između predviđenih i stvarnih vrijednosti. Jednom kada se definira funkcija cijene potrebna su joj dva argumenta stvarne vrijednosti i previđene vrijednosti, a ona vraća skalarnu vrijednost za svaki par točka podataka (tablica 4.1.). Funkcija cijene je jako bitna jer nam ona definira površinu gubitka te za svaki problem se ne može odabrati bilo koja funkcija, a dimenzionalnost prostora parametara se povećava kako raste broj parametara modela.

Tablica 4.1. Funkcije cijena i njihove pripadajuće jednadžbe

Ime funkcije	Formula
Srednja kvadratna pogreška	$J = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$
Srednja kvadratna logaritamska pogreška	$J = \frac{1}{n} \sum_{i=1}^n (l\epsilon \cdot (y^{(i)} + 1) - l\epsilon \cdot (\hat{y}^{(i)} + 1))^2$
Srednja absolutna pogreška	$J = \frac{1}{n} \sum_{i=1}^n y^{(i)} - \hat{y}^{(i)} $
L1	$J = \sum_{i=1}^n y^{(i)} - \hat{y}^{(i)} $
L2	$J = \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$
Unakrsna entropija	$J = -\frac{1}{n} \sum_{i=1}^n [y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$
Kullback Leibler (KL) divergencija	$J = \underbrace{\frac{1}{n} \sum_{i=1}^n (y^{(i)} \log(y^{(i)}))}_{E} - \underbrace{\frac{1}{n} \sum_{i=1}^n (y^{(i)} \log(\hat{y}^{(i)}))}_{U - e}$
Hinge	$J = \frac{1}{n} \sum_{i=1}^n \max(0, 1 - y^{(i)} \hat{y}^{(i)})$
Kvadratna Hinge	$J = \frac{1}{n} \sum_{i=1}^n (\max(0, 1 - y^{(i)} \hat{y}^{(i)}))^2$
Poisson	$J = \frac{1}{n} \sum_{i=1}^n (\hat{y}^{(i)} - y^{(i)} \log(\hat{y}^{(i)}))$

Kako bi se smanjila dimenzionalnost i olakšao posao optimizacijskim algoritmima upotrebljava se metoda preskakanja veza (slika 4.8.). [14]

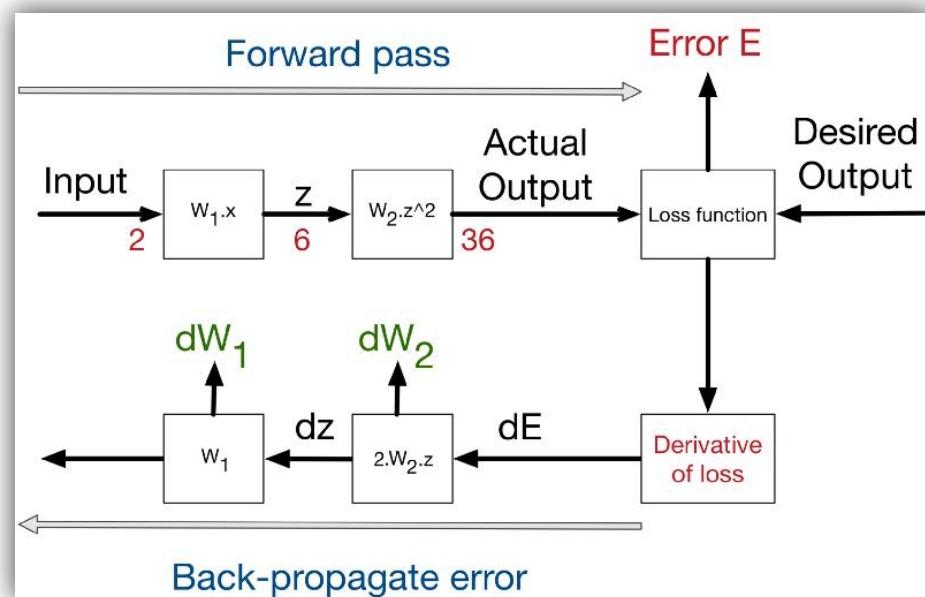


Slika 4.8. Površina gubitaka za neuronsku mrežu ResNet-56 bez i s preskakanjem veza [15]

4.4.2. Algoritam optimizacije

Parametri modela se optimiziraju pomoću optimizacijskih algoritama. Optimizacijski algoritmi su vrsta algoritama koja može na učinkoviti način pretraživati površinu gubitka pri čemu je cilj algoritma naći najnižu točku (globalni minimum) u što manjem broju koraka od početne pozicije. Početna pozicija je nasumična jer se značajkama to jest težinama početne vrijednosti dodjeljuju slučajnom inicijalizacijom. Iako postoje tehnike kao što su He inicijalizacija i Xavier inicijalizacija one se i dalje temelje na slučajnom odabiru, a njihova uloga je da se na nešto pametniji način nađu nasumično početne vrijednosti kako bi se smanjilo vrijeme treniranja osobito za vrlo velike modele te kako bi se izbjegao problem nestajanja odnosno eksplozija gradijenata.

Optimizacija se događa nakon prosljeđivanja prema naprijed i izračuna pogreške pomoću funkcije cijene. Polazeći od izlaznog sloja, informacije o gubitku se šire svim neuronima u skrivenom sloju koji izravno doprinose izlazu. Međutim, neuroni skrivenog sloja primaju samo dio ukupnog signala gubitka, na temelju relativnog doprinosa koji je svaki neuron pridonio izvornom izlazu. Ovaj proces se ponavlja, sloj po sloj, dok svi neuronи u mreži ne dobiju signal gubitka koji opisuje njihov relativni doprinos ukupnom gubitku. Na osnovu signala gubitka možemo prilagoditi sve težine veza između neurona. Ono što se ažuriranjem težina želi postići je da kada sljedeći put mrežu koristimo za predviđanje greška bude što bliže nuli. Algoritmi koji koriste gore opisano proces za ažuriranje težina koristeći parcijalnu derivaciju i pravilo lanca nazivaju se algoritmi s unatražnim rasprostiranjem (slika 4.9.).



Slika 4.9. Smjer prosljeđivanja unaprijed (predviđanje) i unatrag (ažuriranje težina) [16]

Za treniranje neuronskih mreža koristi se jedan od sljedećih optimizacijskih algoritma:

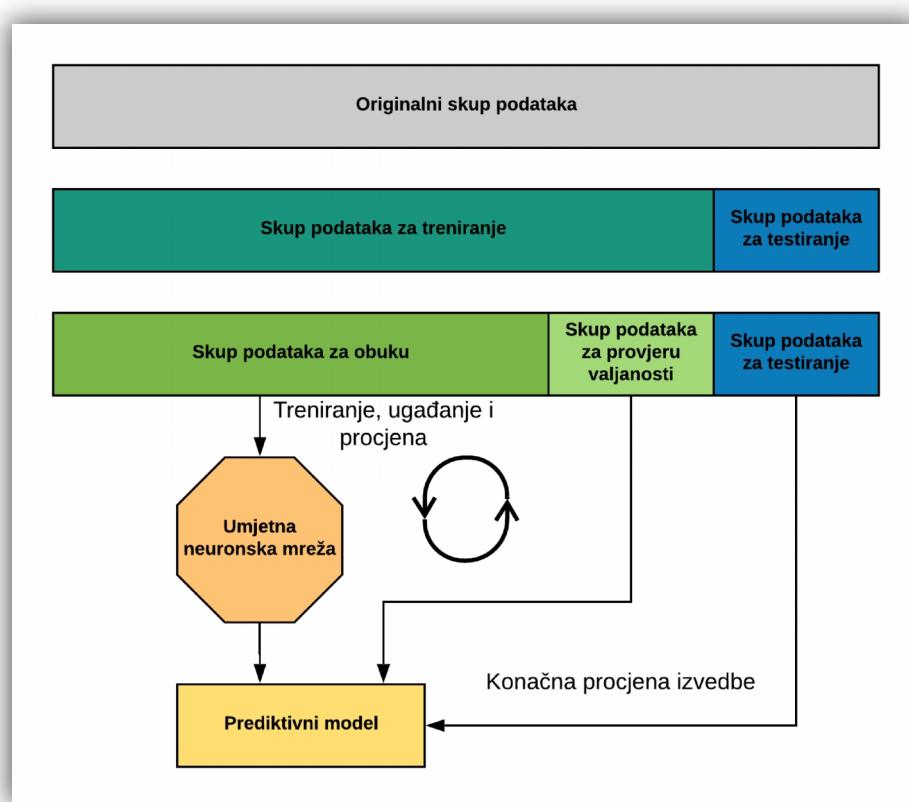
- Algoritam postupnog opadanja,
- Moment optimizacije,
- Nesterovo ubrzano opadanje (NAG),
- AdaGrad,

- RMSProp,
- Procjena adaptivnog momenta (Adam).

4.5. Pretreniranje i podtreniranje

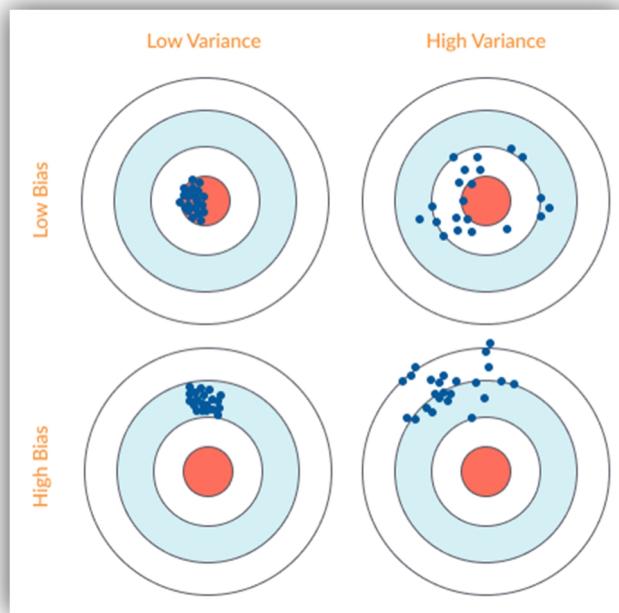
Skup podataka koji imamo na raspolaganju se najčešće dijeli na tri dijela (slika 4.10.) i to na:

- Skup podataka za obuku – koristi se za obuku modela (težina i odstupanja u slučaju neuronskih mreža).
- Skup podataka za provjeru valjanosti – omogućava nepristranu procjenu modela dok se ugađaju hiperparametri modela.
- Skup podataka za testiranje – služi za pružanje nepristrane procjene konačnog modela.



Slika 4.10. Podjela skupa podataka i njihova namjena

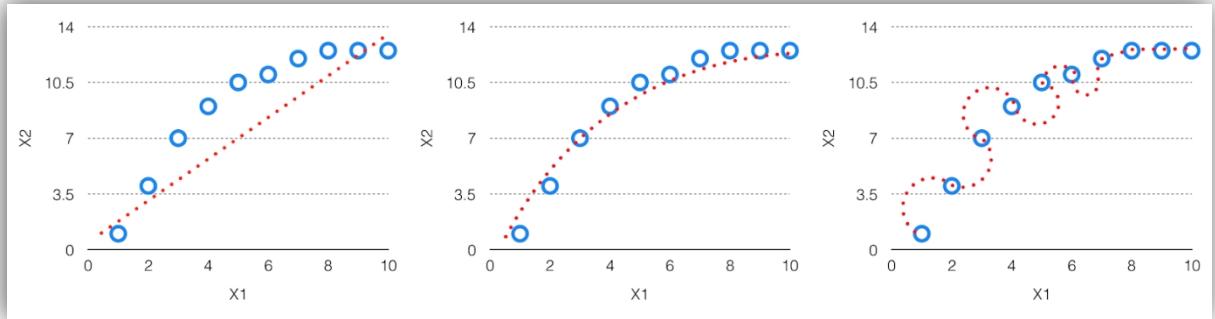
Kod procjenjivanja modela jako su nam bitne dvije karakteristike i to odstupanje i varijanca. Odstupanje nam ukazuje koliko su predviđene vrijednosti daleko od stvarnih vrijednosti. Model ima visoko odstupanje kada se sami model ne uklapa dobro u skup podataka za obuku, a to će se odraziti tako što će trening pogreška biti velika. Nisko odstupanje nam govori da se model dobro uklapa te će pogreška treninga biti mala. S druge strane varijanca nam ukazuje koliko je model osjetljiv na promjene u podacima za trening. Mala varijanca implicira da će model imati mala odskakanja za male promjene, a velika varijanca uzrokuje da mreža modelira slučajni šum u podacima umjesto namjeravanog (slika 4.11.).



Slika 4.11. Kompromis odstupanja i varijance [17]

Podtreniranje se javlja kada je model previše jednostavan da bi naučio temeljnu strukturu podataka. Rješenja za podtreniranje su odabir složenijeg modela, odabir boljih značajki i/ili smanjivanje ograničenja modela. S druge strane pretreniranje je kada model dobro funkcioniра nad skupom podataka za obuku, ali model nije dobro poopćen pa pravi velike greške na neviđenim podacima. Pretreniranje nastaje kad je model previše kompleksan u odnosu na količinu i šum podataka za trening. Rješenja koja se nameću za problem pretreniranja su pojednostavljenje modela s manjim brojem parametara, sakupljanje većeg broja instanci u

skupu podataka za obuku i/ili smanjivanje šuma u podacima za trening (slika 4.12.). Ograničavanje modela kako bi bio jednostavniji te smanjili rizik od pretreniranja zove se reguliranje.



Slika 4.12. Grafički prikaz podtreniranja, optimalnog treniranja i pretreniranja [18]

4.6. Reguliranje

Duboke neuronske mreže obično imaju na desetke tisuća parametara, a neke od njih imaju i na milijune parametara. S tolikim brojem parametara mreže su toliko fleksibilne da mogu naučiti i jako kompleksne uzorke iz skupova podataka. Koliko god je učenje kompleksnih uzoraka prednost ono sa sobom nosi i veliki rizik da dođe do pretreniranja. Tehnika reguliranja nam omogućava da i nešto složenije neuronske mreže s većim brojem parametara istreniramo nad jednostavnijim skupovima podataka. U tehnike reguliranja spadaju $L1$ i $L2$ reguliranje, rano zaustavljanje, ispuštanje, povećanje podataka i tako dalje.

$L1$ i $L2$ reguliranje je tehniku reguliranja koja se implementira u funkciju cijene, a čija joj je uloga ograničiti težine veza neuronske mreže. Pod ograničenjem težina veza se smatra smanjivanje njihovih vrijednosti jer možemo prepostaviti da neuronska mreža s manjim težinama vodi do jednostavnijeg modela te se na taj način može postići ublažavanje prekomjernog učenja. Nakon implementacije ograničavanja funkcija cijene ima sljedeći opći oblik:

$$\text{Funkcija gubitka} = \text{Greška}(y, \hat{y}) + \text{Regulacijski član}$$

Funkcija cijene s $L1$ regulatorom ima sljedeći oblik:

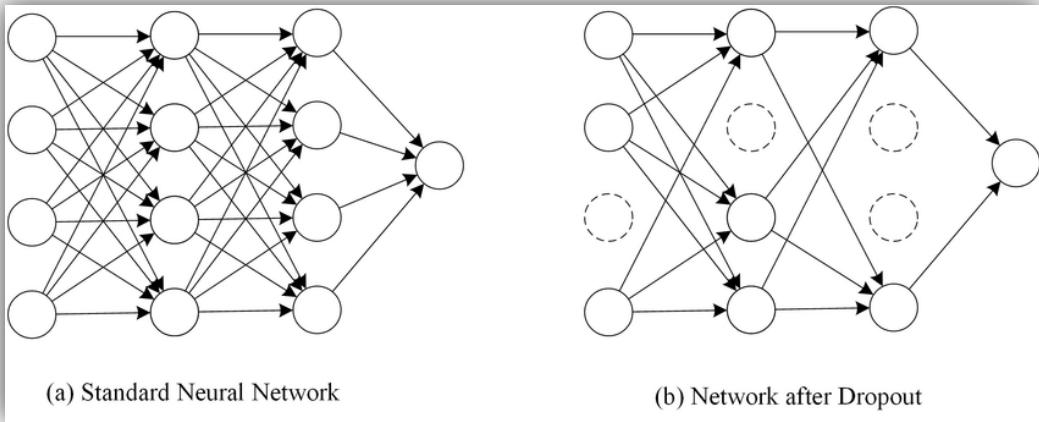
$$F_{L1} = G - šk(y, \hat{y}) + \lambda \sum_{i=1}^n |w_i| \quad (4.9.)$$

Funkcija cijene s $L2$ regulatorom ima sljedeći oblik:

$$F_{L2} = G - šk(y, \hat{y}) + \lambda \sum_{i=1}^n w_i^2 \quad (4.10.)$$

$L1$ regulator kažnjava absolutnu vrijednost težina te težine se mogu svesti na vrijednost nula što je vrlo korisno kada se pokušava sažeti model. $L2$ regulator prisiljava da težine veza propadaju prema nuli, ali ne točno na nulu. λ je parametar regulacije drugim riječima on spada među hiperparametre čija se vrijednost optimizira za bolje rezultate. Međutim treba se naglasiti ukoliko umjetna neuronska mreža ima puno slojeva ovaj pristup reguliranja nije baš prikladan.

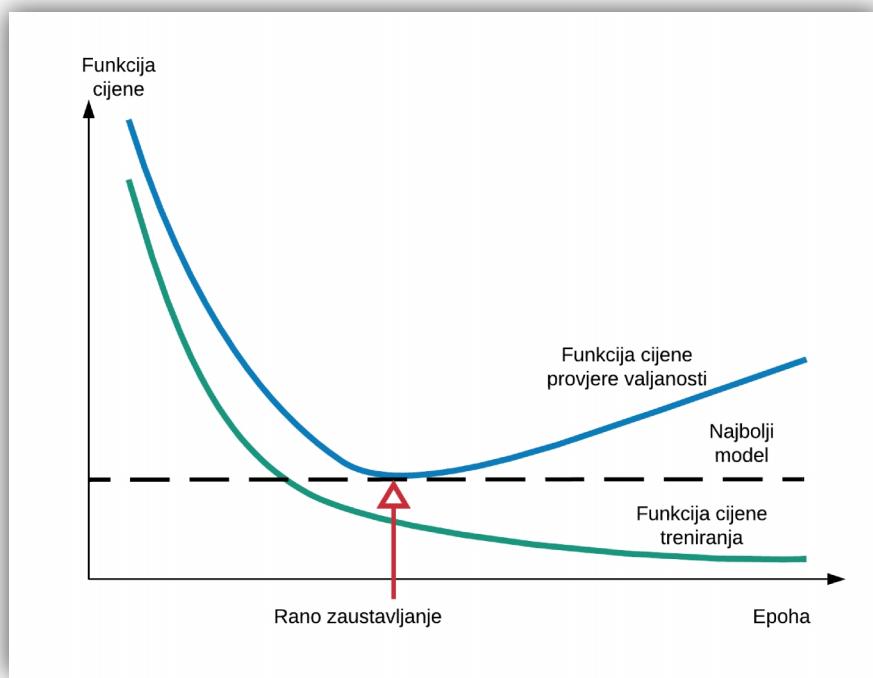
Ispuštanje je vrsta tehnike reguliranja koja za vrijeme svakog koraka u procesu treniranja mreže može isključiti svaki neuron s vjerojatnošću p . Ako je neuron isključen (ispušten) to znači da će za vrijeme trenutnog koraka treninga biti u potpunosti ignoriran kao da ne postoji (slika 4.13).



Slika 4.13. Standardna neuronska mreža i mreža nakon primjene ispuštanja neurona [19]

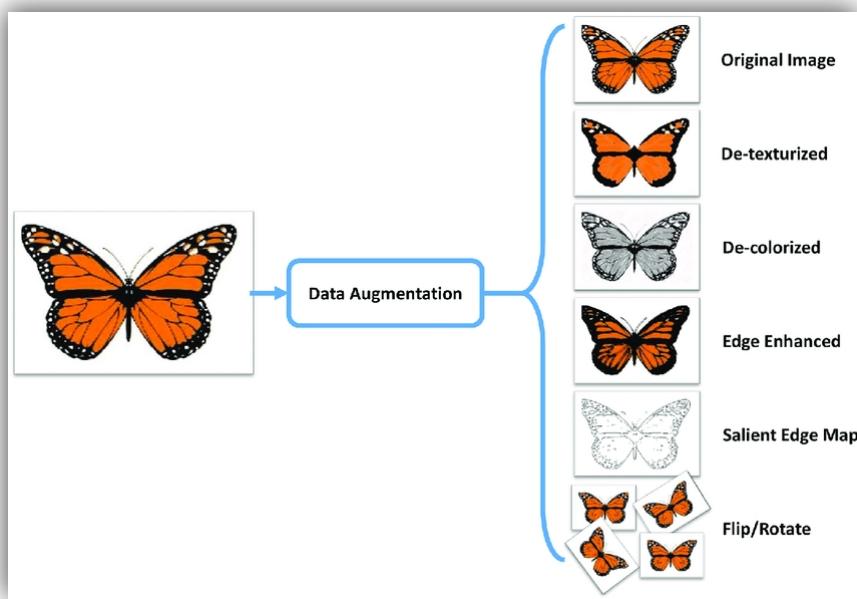
Ispuštanje ima učinak da proces učenja učini šumovit kako bi prisilio čvorove unutar sloja da preuzmu više ili manje odgovornosti za ulaze. Nuspojava ispuštanja simulira rijetku aktivaciju unutar danih slojeva što potiče mrežu da uči rijetku zastupljenost. Budući da ispuštanje izravno utječe na izlazni sloj ono može dovesti do pojave smanjenja kapaciteta ili stanjivanja mreže tijekom učenja stoga je preporučeno raditi šire mreže (veći broj neurona po sloju) kada se primjenjuje tehnika ispuštanja pogotovo kada je vjerojatnost ispuštanja p veća. Vjerojatnost ispuštanja spada u kategoriju hiperparametara. S obzirom na koje vrste slojeva se može primijeniti ispuštanje je skoro pa neograničeno pa se tako ispuštanje može primijeniti na potpuno povezane slojeve, konvolucijske slojeve, ponavljujuće slojeve kao i na dugi kratkoročni memorijski sloj. Ispuštanje se primjenjuje i na skrivene slojeve u mreži pri čemu se dozvoljava implementirati na bilo koji odnosno bilo koje slojeve pa čak na sve skrivene slojeve. Ispuštanje se čak može implementirati na ulazni sloj. Jedini sloj nad kojim se ne implementira ispuštanje je izlazni sloj.

Rano zaustavljanje je metoda reguliranja koja zaustavlja trening to jest učenje čim greška provjere valjanosti počne rasti. Kod treniranja velikih mreža tijekom treninga će se pojaviti točka kada će model prestati generalizirati i početi će učiti statistički šum u skupu podataka za obuku. To je trenutak kada počinje rasti vrijednost funkcije cijene provjere valjanosti te od tog trenutka dalnjim treniranje modela postaje sve manje koristan za predviđanje novih podataka. Izazov je obučavati mrežu dovoljno dugo da je sposobna naučiti ono što se od nje traži, ali da se ne prilagodi previše podacima za obuku. Iz tog razloga i postoji skup podataka za provjeru valjanosti inače bi se umjetna neuronska mreža mogla beskonačno dugo trenirati praktički dok greška odnosno cijena funkcije treniranja ne padne do nula. Jedan pristup kako riješiti koliko dugo trenirati mrežu je da broj epoha treninga tretiramo kao hiperparametar i da model treniramo više puta svaki put s drugom vrijednošću. Problem ovog pristupa je što zahtijeva da se više puta jedan te isti model nanovo trenira pri čemu će se mnogi odbaciti, a to sve skupa može biti računski neefikasno i dugotrajno. Drugi to jest alternativni pristup je trenirati model jednom pri čemu se postavi da je broj epoha velik. Za vrijeme treninga potrebno je vršiti vrednovanje modela preko funkcija cijena. Ako se učinak modela na skupu podataka za provjeru valjanosti počne smanjivati (gubitak počinje povećavati ili točnost počinje opadati) proces treninga se zaustavlja. (slika 4.14.)



Slika 4.14. Rano zaustavljanje s obzirom na vrijednosti funkcija cijena

Povećanje podataka je još jedna u nizu od tehnika kako regulirati to jest spriječiti pretreniranje. Ovom se tehnikom kako joj samo ime kaže povećava to jest generiraju nove instance za treniranje (učenje) iz postojećih. Ovim postižemo da na umjetan način povećamo veličinu skupa podataka za trening. Jedina je stvar u tome što se mora paziti da se kreiraju što realističniji trening instance pri čemu bi idealan slučaj bio takav da umjetno proizvedene instance čovjek ne može razlikovati od stvarnih. Povrh toga jednostavno dodavanje bijelog šuma nije rješenje jer varijacije koje se primjene morale bi se moći naučiti, a bijeli šum nije jedna od njih. Proširenje podataka se može izvršiti prije nego se podaci počnu davati modelu što je dobra tehnika ako imamo mali skup podataka i samo proširenje podataka neće značajno povećati njegovu veličinu. Druga opcija je da proširenje podataka bude jedan od koraka unutar faze učenja modela te da se prošire samo oni podaci koji će se koristiti u tom koraku učenja. Na slici 4.15. možemo vidjeti nekoliko primjera proširenja instance slike.



Slika 4.15. Primjer primjene proširenja podataka na instanci slike [20]

4.7. Arhitektura neuronske mreže

Arhitektura neuronskih mreža raste i mnoge od njih imaju svoje pod varijante, ali neke od najvažnijih i najpoznatijih su sljedeće:

- Konvolucijska neuronska mreža (CNN),
- Ponavljujuća neuronska mreža (RNN),
- Duga / kratkotrajna memorija (LSTM),
- Usmjerena ponavljujuća jedinica (GRU),
- Hopfield mreža (HN),
- Boltzmannov stroj (BM),
- Mreža dubokih uvjerenja (DBN),
- Autoenkoder,
- Generativna suparnička mreža (GAN) itd.

Od svih navedenih, a i ostalih umjetnih neuronskih mreža opisati će se samo konvolucijska neuronska mreža jer će se ona koristiti u ovome radu.

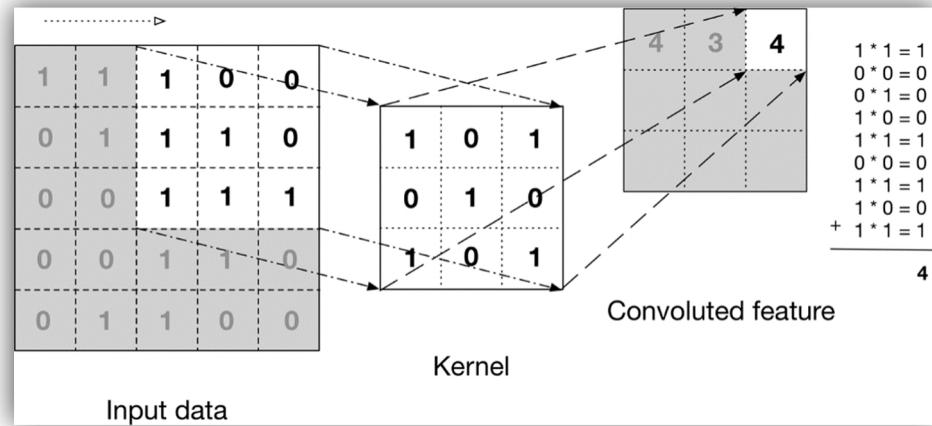
4.7.1. Konvolucijska neuronska mreža

Konvolucijske neuronske mreže su jedne od važnijih vrsta mreža, a to možemo uočiti iz njihove primjene koja uključuje kategoriziranje slika, prepoznavanje slika, detekcija objekata na slici, prepoznavanje lica, kod pretraživača slika, autonomnih vozila. Iako se uglavnom koriste za vizualnu percepciju njihovu primjenu možemo naći i u drugim granama kao što su prepoznavanje glasa i prirodna obrada jezika (NLP). Njihovo rasprostranjenosti pridonosi obilježje što mogu raditi s podacima koji imaju prostorni i/ili vremenski odnos koji je najčešće dvodimenzionalan, matrični, ali može biti promijenjen u jednodimenzionalan što će omogućiti razvoj unutarnjeg prikaza jednodimenzionalnog slijeda. U ovom radu su se koristile slike kao ulazni podatci, a oni su pravi primjer podataka s prostornim odnosom. Za više detalja kako su digitalne slike reprezentirane u samom računalu to jest kako računalo vidi slike pogledajte [dodatak A](#). Konvolucijske neuronske mreže se nadograđuju na arhitekturu potpuno povezanih mreža na način da dodaju dva nova tipa sloja prije standardnog ulaznog sloja te ujedno postaje i sami ulaz u neuronsku mrežu. Ta dva sloja su konvolucijski sloj i sloj ujedinjavanja. Ova dva sloja se mogu dodati u većem broju to jest nema nekakvog ograničenja koliko ih može biti, ali uglavnom će njihov broj rasti kako raste kompleksnost zadatka. Broj neurona u prvom sloju će uvijek biti manji od ukupnog broja piksela, a razlog tomu je što svaki piksel u ulaznoj slici nije povezan s jednim neuronom nego je svakom pikselu dodijeljeno receptivno polje što omogućava mreži da traži prostorni odnos među pikselima. Receptivno polje je dio slike na koje smo trenutno fokusirani drugim riječima na taj dio slike trenutno primjenjujemo operaciju konvolucije. Ovakva logika se nastavlja na većinu sljedećih konvolucijskih slojeva gdje se jednom neuronu dodjeljuje odgovarajuće receptivno polje što arhitekturi same mreže omogućava da se u prvim slojevima koncentrira na značajke niske razine. Kako se podaci propagiraju prema kraju tako značajke niže razine se formiraju u značajke više razine. Većina konvolucijskih slojeva će imati manji broj neurona nego što je piksela, ali ponekad se zna ubaciti i konvolucijski sloj u kojem broj neurona odgovara broju piksela.

Najvažniji sloj konvolucijske neuronske mreže je konvolucijski sloj i on je ujedno uvijek prvi sloj u ovoj mreži. Konvolucija je matematička operacija koja jednu funkciju (g) pomjera nad drugom (f) i mjeri integral njihovog umnoška (jednadžba 4.11.).

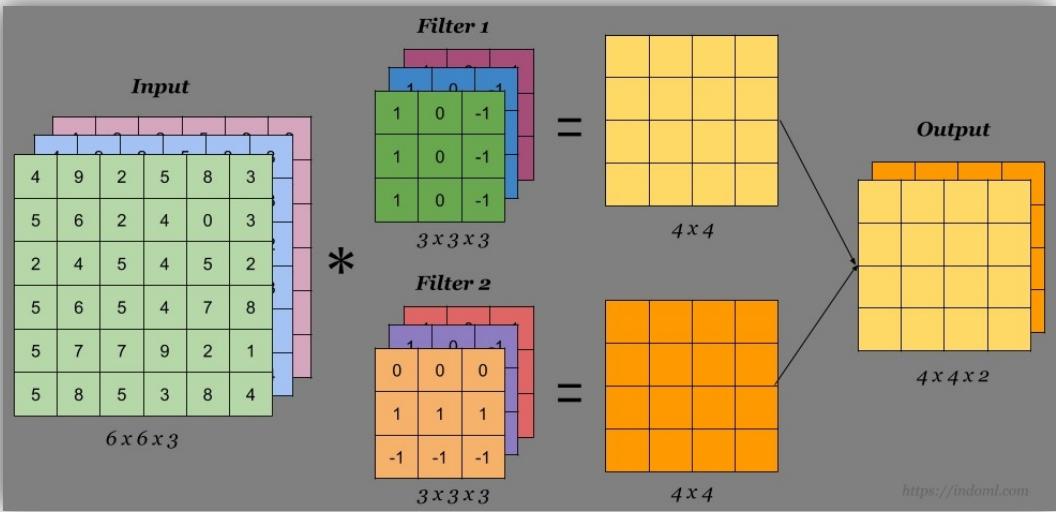
$$(f \cdot g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau \quad (4.11.)$$

Jednadžba 4.11. za našu primjenu je takav da će funkcija f biti ulazni podaci (slika), a funkcija g će biti filter ili kernel. Za nešto više detalja o filterima pogledate [dodatak B](#). Težine neurona se kod konvolucijskih mreža mogu predstaviti kao male slike veličine receptivnog polja. Te težine nazivamo filterima ili kernelima. Konvolucija će u našem slučaju biti skalarni umnožak ulaznih podataka i kernela, kao rezultat ove konvolucije biti će karta značajki koja naglašava one dijelove slike koju najviše sliče filteru (slika 4.16.).



Slika 4.16. Operacija konvolucije

Često se može susresti da ulazni podaci imaju i dubinu, ako nemaju steknu je nakon prve konvolucije. Primjer dubine kao ulaznog podatka je RGB slika u boji koja sadrži tri matrice (polja). Treba se naglasiti da filtera može biti više unutar jedne konvolucije, ali svi filteri moraju biti istih dimenzija pri čemu mu je samo dubina unaprijed određena i jednaka je dubini ulaznog podatka. Izlazna dubina podataka odgovara broju primijenjenih filtera (slika 4.17.). Konvolucija zahtijeva da jedna funkcija u našem slučaju filter pomjera po drugoj funkciji to jest ulaznoj matrici. Kod konvolucijskih neuronskih mreža konvolucijsko pomjeranje se definira kao broj piksela za koji se pomiče filter preko ulazne matrice.



Slika 4.17. Konvolucija s više filtera i s podacima koji sadrže dubinu [21]

Na osnovu ulazne matrice, veličine filtera i napredovanja (pomjeranje) može se dobiti dimenzije izlaza (jednadžbe 4.12. i 4.13.).

$$iz \quad \text{šir} = \frac{W - F_w + 2P}{S_w} + 1 \quad (4.12.)$$

$$iz \quad v = \frac{H - F_h + 2P}{S_h} + 1 \quad (4.13.)$$

Gdje je:

W – širina ulaza

F_w – širina filtera

S_w – horizontalno napredovanje

P – nadopunjavanje

F_h – visina filtera

S_h – vertikalno napredovanje

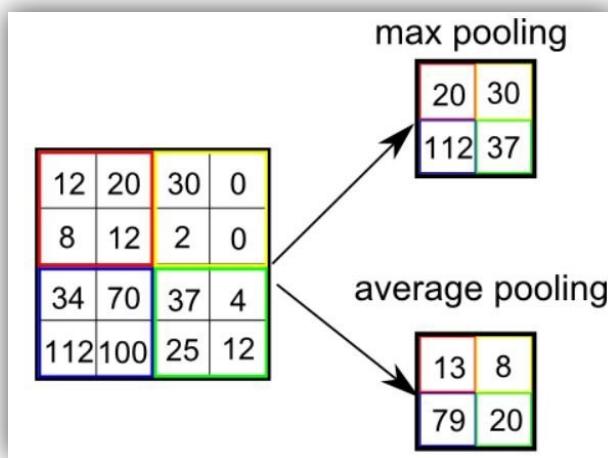
Ove jednadžbe će nam dati prostorne dimenzije, širinu te visinu, pri čemu se konačna dimenzija to jest dubina dobije tako što se dubina izjednači s brojem filtera koji se primijenio u konvoluciji.

Ponekad filter s obzirom na odabranu veličinu koja je uvijek neparne veličine (3×3 , 5×5 , 7×7 , itd.) i napredovanja (1, 2, 3, itd.) ne može savršeno preći preko ulazne matrice stoga imamo dvije opcije:

- Dodati okvir oko slike i nadopuniti ga vrijednostima (najčešće nulama) čime taj okvir sada postaje dijelom slike.
- Izbaciti dio slike na koji se filter nije uklopio.

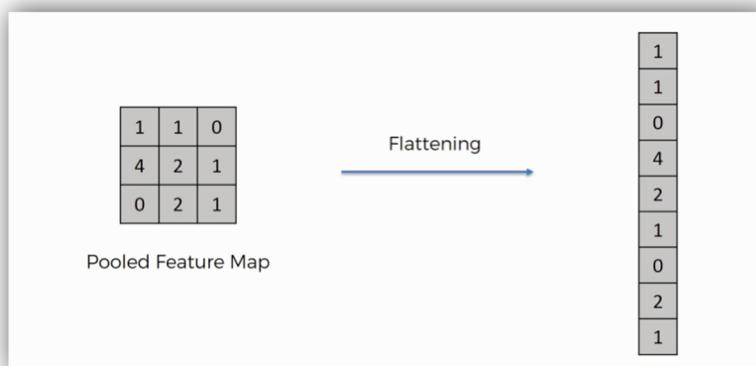
Još je jedan razlog zašto se radi nadopunjavanje osim da bi filter savršeno prešao preko slike. Drugi razlog iz kojeg se može vršiti nadopunjavanja je želja da podatak (slika) ostane iste prostorne dimenzije, rezolucije, po širini i visini. Za kraj konvolucijskog sloja imamo aktivacijsku funkciju kojoj je cilj uvesti nelinearnost u konvolucijske neuronske mreže. Najčešće korištena funkcija je ReLU funkcija. Konvolucijski sloj ima nekoliko hiperparametara među kojima su broj filtera, visina i širina filtra, kretanje (napredovanje) te nadopunjavanje.

Sloj ujedinjavanja služi kako bi uzeo poduzorke od ulazne matrice (slike) kako bi smanjio računalno opterećenje, upotrebu memorije i smanjio broj parametara te time ograničio rizik od pretreniranja. Drugim riječima se može reći da podaci prolaskom kroz sloj ujedinjavanja doživljavaju smanjenje dimenzionalnosti. Nadalje, korisno je za izvlačenje dominantnih značajki koje su rotacijski i pozicijski invarijantni. Kod sloja za ujedinjavanje potrebno je definirati veličinu bloka, korak i tip. Ujedinjavanje se obično vrši kroz cijelu dubinu (sve kanale) tako da je izlazna dubina jednaka ulaznoj. Postoji i alternativno ujedinjavanje koje se vrši preko dimenzije dubine, u tom slučaju prostorne dimenzije (visina i širina) ostaju nepromijenjene, ali se smanji broj kanala. Za ujedinjavanje se koriste najčešće maksimalni ujedinjavanje i prosječno ujedinjavanje. Kod maksimalnog ujedinjavanja se vraća maksimalna vrijednost iz dijela matrice (slike) pokrivene kernelom. S druge strane, kod prosječnog ujedinjavanja vraća se prosjek svih vrijednosti iz dijela matrice (slike) pokrivene kernelom (slika 4.18).



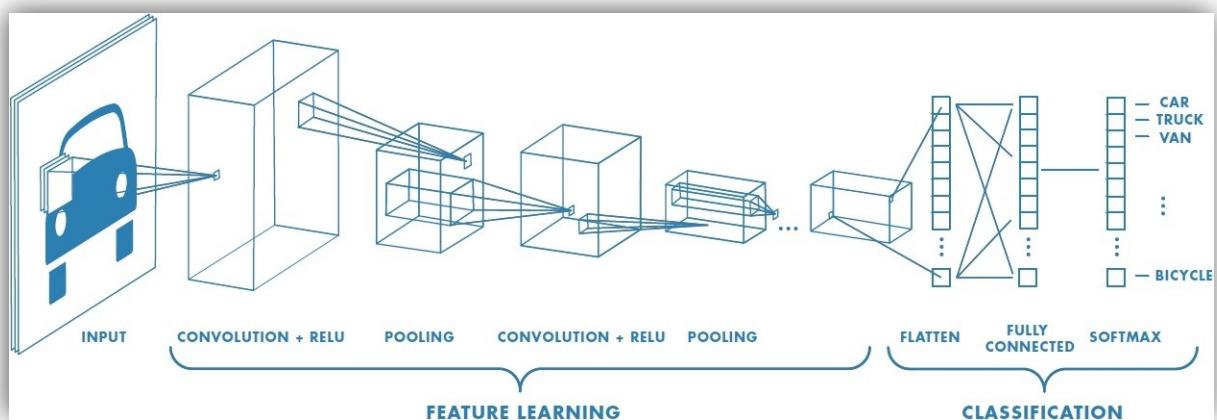
Slika 4.18. Primjer maksimalnog i prosječnog ujedinjavanja [22]

Konvolucijske neuronske mreže slažu nekoliko konvolucijskih slojeva pri čemu svaki sloj općenito praćen ReLU slojem, potom sloj ujedinjavanja, zatim drugi konvolucijski sloj plus ReLU, potom sloj ujedinjavanja i tako dalje. Ulazna slika postaje sve manja i manja kako napreduje kroz mrežu, ali iz nje se izvlači sve više i više značajki upravo zbog konvolucijskih slojeva zato se ovaj dio arhitekture konvolucijske neuronske mreže naziva učenje značajki. Na kraju ili vrhu same arhitekture nalazi se standardna unaprijedna neuronska mreža. S obzirom da konvolucijski slojevi i slojevi ujedinjavanja rade s prostornim podacima (dvodimenzionalnim), a unaprijedna mreža radi s vektorom stupca zbog čega se na početak unaprijedne neuronske mreže dodaje sloj za poravnavanje (slika 4.19.).



Slika 4.19. Primjer poravnavanja [23]

Unaprijedna neuronska mreža je sastavljena od popuno povezanih slojeva pri čemu se najčešće koristi ReLU kao aktivacijska funkcija i na samom kraju imamo izlazni sloj za predviđanje koji ovisi o samojoj upotrebi mreže (slika 4.20.).



Slika 4.20. Arhitektura konvolucijske neuronske mreže [24]

Ono što konvolucijska neuronska mreža uči pomoću algoritma s unutrašnjim rasprostiranjem nisu više samo težine u unaprijednoj neuronskoj mreži nego i filtere koje će primjenjivati u konvolucijskim slojevima. [9]

5. IZRADA AUTONOMNOG VOZILA

5.1. Korišteni programski jezik i simulator

5.1.1. Python

Za praktičnu izvedbu odabran je programski jezik Python koji je osmišljen u kasnim 80-tim godinama prošlog stoljeća od strane Guido van Rossum. Python je interpretirani, objektno orijentirani programski jezik visoke razine s dinamičkom semantikom. Visoka razina strukture podataka (string, lists, tuples, sets, dicts) u kombinaciji s dinamičkom semantikom programske jezike Python čine veoma dobrim za brzi razvoj aplikacija i skripti te jezikom za povezivanje postojećih komponenti. Vodeća načela kojima Python žudi su sažeta u 20 aforizama od kojih je samo napisano 19 (<https://www.python.org/dev/peps/pep-0020/>), a samo neka od njih su:

- Lijepo je bolje nego ružno,
- Jednostavno je bolje nego složeno,
- Ravno je bolje nego ugniježđeno,
- Iako praktičnost pobijeđuje čistoću itd.

Veliki uspjeh Python-a počiva na podržavanju modula i paketa što potiče modularnost programa i ponovnu upotrebu koda. Moduli i paketi ne dolaze samo od samog ekosustava programskog jezika Python već i od strane drugih programera. Od mnogobrojnih modula i paketa koji postoje za Python za potrebe praktične izvedbe korišteni su sljedeći (istaknuti će se samo najznačajniji):

- Keras,
- Pandas,
- NumPy,
- OpenCV,
- Matplotlib,
- Seaborn,
- Socketio,
- Os itd.

Keras je sučelje visoke razine za programiranje neuronskih mreža i napisano je u programskom jeziku Python. Keras se nadostavlja povrh TensorFlow-a, CNTK-a ili Theano-a, a omogućava jednostavno i brzu izgradnju modela neuronskih mreža za eksperimentiranje. Usredotočuje se na prilagođavanje korisnicima, modularnost i proširivost. Keras programeru omogućuje implementaciju kako najčešće korištene građevne blokove neuronskih mreža (slojeve, aktivacijske funkcije, optimiziranje, itd.) tako i složene neuronske mreža (konvolucijske i ponavljujuće neuronske mreže) te zahtjevne blokove i slojeve (odbacivanje, normalizacija hrpe, ujedinjavanje itd.). Isto tako Keras omogućuje programeru izradu dubokih modela za pametne telefone, za Internet te čak i za Java virtualni stroj. Kako bi treniranje neuronskih mreža bilo brže Keras omogućuje distribuciju modela za duboko učenje na klasterima jedinica za grafičku obradu (GPU) i Tensor procesnim jedinicama (TPU). [25]

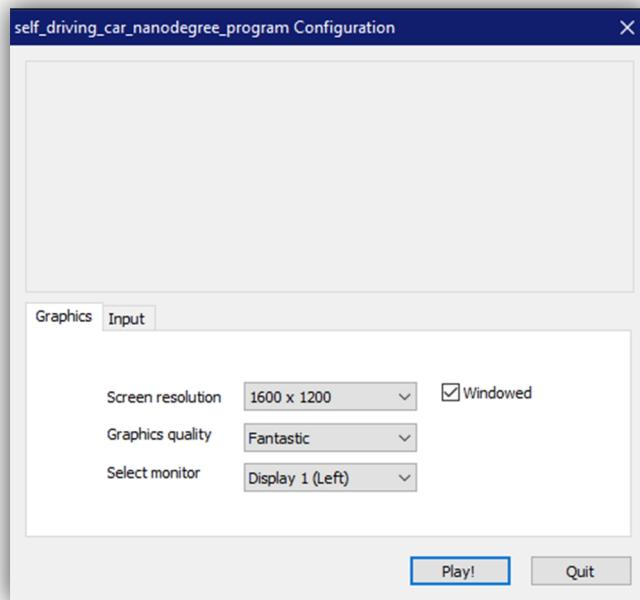
Pandas je modul s BDS licencom i otvorenim kodom, a pruža strukture podataka visokih performansi i jednostavnosti za korištenje. Također služi kao i alat za analizu podataka za programski jezik Python što dobro nadopunjuje Python mogućnosti za obradu i pripremu podataka. Neke od važnijih značajki Pandas modula su da posjeduje objekt DataFrame koji je brz i učinkovit za manipulaciju podatcima s integriranim indeksiranjem. Alat za čitanje i pisanje podataka između podataka u memoriji i različitim formata (CSV, Microsoft Excel, SQL baze podataka, HDF5 format itd.). Veoma je optimiziran za performanse pri čemu su kritični dijelovi koda pisani programskim jezikom Cython ili C. [26] Drugi važni modul koji se koristio uz Pandas je NumPy. NumPy predstavlja temeljni paket znanstvenog računanja s Python-om te isto tako kao i Pandas modul. NumPy je licenciran pod BDS licencom. Sadrži N-dimenzionalne matrice te pripadajuće sofisticirane funkcije za manipulaciju tim istim matricama. Često je korišten modul kada je potrebno primijeniti linearnu algebru, Fourierovu transformaciju i slučajne brojeve. [27]

Matplotlib je modul za prikaz 2D grafova u raznim formatima od onih koji se mogu tiskati pa do onih koji se mogu upotrijebiti u interaktivnim okruženjima. Mogu se generirati stupčasti, linijskim grafikoni, tortni i prstenasti grafikoni, raspršeni grafikoni, histogrami i tako dalje. Za jednostavniji prikaz tu je pyplot modul koji pruža sučelje nalik MATLAB-u. [28] U radu se

osim Matplotlib-a za vizualizaciju koristio još jedan alat, a to je TensorBoard. On nam služi kako bi lakše mogli razumjeti, ispraviti i optimizirati TensorFlow programe poput treniranja robustnih dubokih neuronskih mreža, a to se postiže tako što se uz pomoć TensorBoard-a mogu vizualizirati TensorFlow grafovi, ocrtavati kvantitativne metrike o izvršavanju grafikona te prikaz dodatnih podataka poput slika koje prolaze kroz njega kada je u potpunosti konfiguriran. [29]

5.1.2. Udacity simulator i generiranje podataka

Udacity simulator je simulator za naučiti automobil da se kreće cestom koristeći duboko učenje. Sam simulator je napravljen u Unity-ju čiji je kod u potpunosti dostupan na GitHub-u (<https://github.com/udacity/self-driving-car-sim>) na kojem su ujedno i upute kako koristiti simulator. Pokretanjem simulatora dolazi se do prozora za konfiguraciju grafike i ulaznih kontrola (slika 5.1.).



Slika 5.1. Konfiguracijski zaslon simulatora

Nakon što korisnik po želji konfigurira simulator pokreće ga klikom na botun „Play!“. Korisnika to dovodi do zaslona gdje potom ima opciju birati između dvije opcije, trening i autonomni, pri čemu trening omogućuje korisniku generiranje podataka, a odabirom autonomnog načina može testirati svoj model. Zatim može vidjeti, ali ne i mijenjati kontrole te ima opciju odabira jednu od dvije ponuđene staze (slika 5.2.).



Slika 5.2. Opcije koje nudi simulator

Za ovaj rad je odabrana džungla staza (desna od dvije ponuđene), a auto se upravlja pomoću tipkovnice koristeći zadane tipke („a“ za lijevo, „w“ za naprijed, „d“ za desno, „s“ kao kočnica). Karakteristika džungle staze je ta što se sastoji od lijeve i desne trake pri čemu su rubni dijelovi označeni punom bijelom trakom, a sredinom prolazi isprekidana bijela traka. Odabirom trening opcije i džungle staze otvara se staza, postavlja se auto na početnu poziciju, te u gornjem lijevom kutu se može vidjeti kut zakreta koji se kreće u rasponu od -25° do 25° . U gornjem desnom uglu se nalazi opcija za snimanje čijim se klikom otvara novi prozor koji nam nudi mogućnost odabira gdje će se podaci koji se budu generirali spremiti. U donjem

desnom uglu se nalazi brzinomjer koji nam pokazuje trenutnu brzinu izraženu u miljama na sat (slika 5.3.).



Slika 5.3. Džungla staza i trening opcija

Nakon odabira po želji gdje će se spremati podaci prije početka snimanja potrebno je još jednom pritisnuti na dugme za snimanje. Nakon njegovog ponovnog aktiviranja može se započeti s vožnjom prema želji. Cilj ovog diplomskog rada je da automobil se kreće samo desnom trakom prema tome i krajnji rezultat prije početka treniranja je trebao izgledati kao da se automobil kretao samo desnom trakom. Nakon što se odvozi željeni dio staze da bi se sami podaci generirali potrebno je ponovno pritisnuti gumb za snimanje. Njegovim pritiskom će se kreirati .csv datoteka imena „*driving_log*“ na odabranoj lokaciji u memoriji. Uz samu datoteku kreirati će se i folder imena *IMG* u kojemu će biti spremljene slike od triju kamera (lijeva, centralna i desna). Datoteka „*driving_log*“ sastoji se od 5 vrijednosti u jednom redu odvojeni zarezom pri čemu su vrijednosti od prve prema zadnjoj sljedeće, absolutni put slike centralne kamere, absolutni put slike lijeve kamere, absolutni put slike desne kamere, kut zakreta koji je

ujedno i normaliziran u rasponu od -1 do 1, gas i kočnica čiji je raspon vrijednosti kreće od 0 do 1 te brzina.

Odabirom opcije autonomnog moda može se testirati model da se vidi koliko dobro model može voziti s obzirom što je korisnik postavio kao cilj. U ovom modu simulator se ponaša kao server na koji će se korisnikov program spojiti te započeti primati slijed slika od centralne kamere te podatak o brzini. Cilj korisnikova programa je da iz skupa informacija izdvoji sliku kako bi njegova mreža mogla odraditi predviđanje kuta zakreta. Zajedno s informacijom o kutu zakreta potrebno je još upakirati informaciju o akceleraciji ili deceleraciji te preko ostvarene veze vratiti komande na server odnosno simulator.

Generirani podaci za treniranje koji su korišteni u ovom diplomskom radu podijeljeni su u tri kategorije. U prvu kategoriju spada vožnja desnom stranom ceste u oba smjera staze. Druga kategorija obuhvaća vožnju lijevom stranom ceste isto tako koristeći oba smjera. Za kraj se još dodatno generirala mala skupina podataka za oštra skretanja i kako bi se vozilo zadržalo podalje od traka ceste to jest kako bi ga pokušali zadržati u centru trake. Za jedan krug je u prosjeku generirano oko 3,000 vrijednosti, a vozila su se po četiri kruga u svakom smjeru za jedan smjer što u konačnici daje oko 24,000 ($4 \times 2 \times 3,000$) vrijednosti za jedan smjer. Isti postupak je ponovljen za vožnju u drugom stranom ceste što u konačnici generira oko 50,000 podataka. Zadnja skupina za oštra skretanja je generirano malo više od 5,000 podataka što na kraju stvara skup od oko 55,000 podataka i tri puta više slika jer za jedan podatak imamo tri slike jer dobivamo slike iz tri kamere (centralna, lijeva i desna).

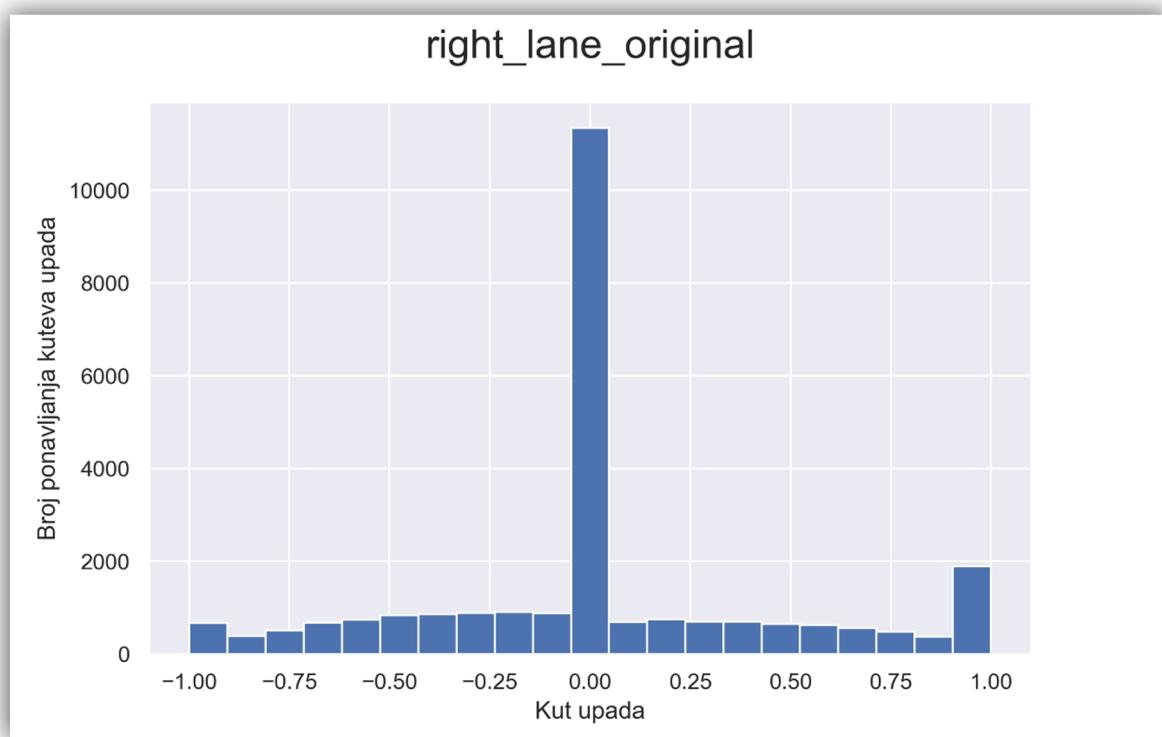
5.2. Predobrada i vizualizacija podataka za treniranje

Predobrada podataka sastavni je korak u primijenjenom strojnom učenju jer kvaliteta podataka i koja se daju modelu za vrijeme treninga izravno utječu na sposobnost i performanse modela stoga je izuzetno važno da se podaci unaprijed obrade prije početka treniranja. Uz predobradu usko je vezana i vizualizacija podataka jer predstavlja vizualnu reprezentaciju podataka i često

je lakše protumačiti podatke kroz vizualnu reprezentaciju. U koncept predobrade među ostalim upada:

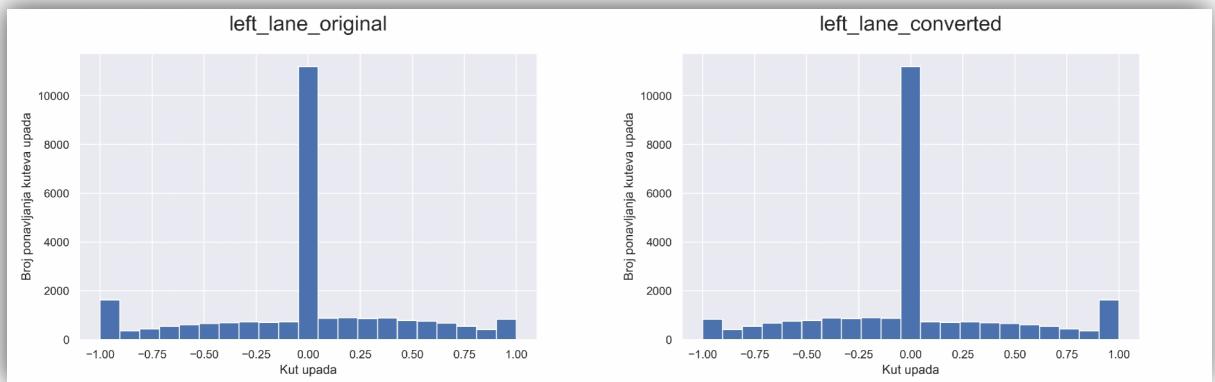
- Odabir podataka
- Formatiranje
- Baratanjem nultim vrijednostima
- Standardizacija
- Kodiranje kategorijskih značajki
- Uzorkovanje
- Objedinjavanje značajki

Za početak su generirane dvije vrste podataka, a to su vožnja desnom stranom, vožnja lijevom stranom. Distribuciju kutova zakreta možemo vidjeti na slici 5.4. za vožnju desnom stranom.



Slika 5.4. Dijagram kutova upada za vožnju desnom stranom

Na slici 5.5. možemo vidjeti na lijevoj strani originalnu distribuciju kutova zakreta, a na desnoj strani možemo vidjeti distribuciju kutova zakreta kada vožnju lijeve strane pretvorimo u vožnju desnom stranom. Razlog pretvorbe je taj što nam je cilj da se auto kreće samo desnom stranom.



Slika 5.5. Dijagram kutova upada za vožnju lijevom stranom

Pretvorba kutova da bi postigli efekt što je izvorno bila vožnja lijevom stranom, a sada izgleda kao vožnja desnom stranom postignuta je idućim kodom:

```

1. def correct_steering_angle(steering_angle_left):
2.     steering_angle_right = pandas.DataFrame()
3.
4.     steering_angle_right['steering'] = -1 * steering_angle_left
5.
6.     return steering_angle_right

```

Funkcija `correct_steering_angle()` će pretvoriti kutove iz vožnje lijevom stranom u vožnju desnom stranom tako što će pomnožiti sve vrijednosti kutova s -1. Idući korak je zrcaliti slike za 180° oko y-osi, a to se postiglo pomoću `flip_image()` funkcije.

```

1. def flip_image(images_path, path_to_save):
2.     for path in images_path:
3.         image = cv2.imread(path)
4.         left_flip = cv2.flip(image, flipCode=180)
5.         image_name = extract_img_name(path)
6.         cv2.imwrite(os.path.join(path_to_save, image_name), left_flip)

```

Funkcija učitava sliku u memoriju, zrcali te takovu spremi na novu lokaciju. Na slici 5.6. u prvo redu možemo vidjeti tri slike od triju kamera koje nam jasno daju indikaciju da se

automobil nalazi u lijevoj traci. Primjenom funkcije `flip_image()` te zamjenom slika lijeve i desne kamere dobivamo efekt da se automobil navodno nalazi u desnoj traci.



Slika 5.6. Zrcaljene slike kako bi postigli efekt vožnje u desnoj traci

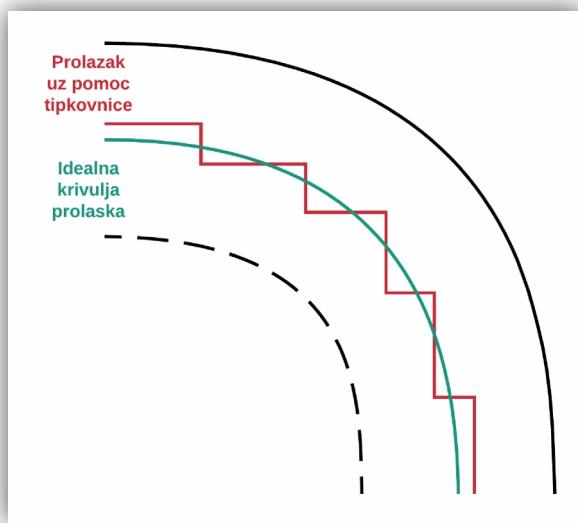
Za kraj je trebalo obratiti pozornost da ono što je prvotno bila lijeva slika sada postaje desna, a ono što je bila desna slika to je sada lijeva slika.

Iz grafova prikazanih na slikama 5.4. i 5.5. možemo uočiti da su podaci poprilično pristrani vrijednostima oko nule. Ako bi pokušali zamisliti kako staza izgleda na osnovu tih grafova zamislili bi jednu poprilično ravnu stazu s nekoliko krivina, ali stvarna staza je poprilično krivudava. Razlog ovakve distribucije kutova je to što se automobil vozio uz pomoć tipkovnice. Tipkovnice ne nude nikakav analogni signal već čisti digitalni signal i to 0 u slučaju ako tipka nije pritisnuta i 1 ako je tipka pritisnuta. Simulator funkcioniра tako da u slučaju pritiska tipke za gas, rikverc, lijevo ili desno od ravnotežnog stanja (0) naglo raste prema maksimalnoj vrijednosti (+1 ili -1 ovisno o parametru). To za posljedicu ima da automobilom kroz zavoje ne može proći glatkim zakrivljenim uzorkom koji odgovara uzorku krivine već se prolazi dijagonalnim zig-zag uzorkom (slika 5.7.).

0.00	0.15	0.35	0.50	0.70	0.85	0.00	0.00	0.00	0.05	0.25	0.40	0.55	0.75	0.95	0.00	0.00	0.15	0.30	0.50	0.70	0.85	1.00	1.00	0.00
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

Slika 5.7. Vrijednosti kuta zakreta za prolazak kroz krivinu

Na slici 5.8. možemo vidjeti vizualnu reprezentaciju prolaska kroz krivinu.



Slika 5.8. Prolazak kroz krivinu koristeći se tipkovnicom naspram uzorkom krivine

Postoji nekoliko načina na koji bi se mogao riješiti problem jednostranih podataka. Jedan od načina je da se izbacio dio podataka prvenstveno one koje doprinose pristranosti vrijednosti oko nule. U toj situaciji gubimo podatke, a neuronske mreže su poznate po tome što je za njihovo treniranje potrebno dosta podataka pogotovo kako raste broj parametara mreže. Druga opcija kojom bi se moglo pristupiti je nad i pod uzorkovanjem pri čemu bi se vrijednosti kojih ima puno poduzorkovale, a vrijednosti kojih ima malo naduzorkovale. S obzirom da su nam potrebni podaci izbacivanje nije opcija, a s različitim uzorkovanjem neki uzorci će više puta proći od drugih što bi moglo uzrokovati da za njih neuronska mreža naučiti „napamet“ vrijednosti i nije baš najbolja opcija.

Metoda koja će se iskoristiti za promjenu distribucije kutova upada u ovom radu je otežana aritmetička srednja vrijednost nepraznog konačnog skupa podataka $\{x_1, x_2, \dots, x_n\}$ s odgovarajućim ne-negativnim težinama $\{w_1, w_2, \dots, w_n\}$ (jednadžba 5.1.). Inače se metoda srednje vrijednosti zna često koristi za zamjenjivanje nedostajućih vrijednosti, a mi u jednu ruku možemo gledati na vrijednosti oko nule kao na nedostajuće vrijednosti, ali naravno ne smijemo sve vrijednosti oko nule zamijeniti većima zbog toga se koristi otežana srednja vrijednost.

$$x = \frac{\sum_{i=1}^n w_i x_i}{\sum_{i=1}^n w_i} \quad (5.1.)$$

Gdje je:

x – vrijednost otežane srednje aritmetičke sredine

w_i – i -ta težina

x_i – i -ti podatkovni element

Podatkovni elementi s većim težinama doprinose više ponderiranoj srednjoj vrijednosti dok elementi s manjim težinama doprinose manje. Težine ne mogu biti negativne dok pojedine težine mogu biti nula, ali ne sve jer dijeljenje s nulom nije dozvoljeno. Kako bi se jednadžba pojednostavnila težine će se normalizirati tako da njihova suma iznosi 1 jer kod dijeljenja s 1 količnik će biti isti kao i djeljenik. Ako su težine normalizirane tada otežana aritmetička sredina poprima oblik kao u jednadžbi 5.2.

$$x = \sum_{i=1}^n w'_i x_i \quad (5.2.)$$

Gdje je:

w'_i – i -ta normalizirana težina

Sada kada imamo odabranu metodu kako modificirati kutove upada treba vidjeti kako iskoristiti samu metodu te odrediti broj težina i njihove vrijednosti. Postoji nekoliko načina kako iskoristit metodu srednjih vrijednosti. Jedna opcija je uzeti m vrijednosti prije trenutne te n vrijednosti poslije trenutne vrijednosti kuta zakreta te nad njima izvršiti proračun otežane srednje vrijednosti. U ovoj opciji ima nekoliko varijanti, a one su:

1. $m > n > 0$,

2. $n > m > 0$,
3. $n = m > 0$,
4. $m > n = 0$,
5. $n > m = 0$.

U ovom radu se išlo s opcijom pod brojem 5, a to je situacija gdje su se gledale samo vrijednosti kutova upada nakon trenutne vrijednosti kuta upada. S obzirom da se s automobilom u simulatoru kretalo samo prema naprijed odlučilo se ići s metodom 5 iako se moglo pokušati i s metodom 2. Mogli bismo reći da je u našoj situaciji bitno što se nalazi ispred vozila, a ne što je bilo prije iako bi se moglo postići nešto bolje izglađivanje kutova ako bi u obzir uzeli i nekoliko vrijednosti prije trenutne vrijednosti. Sada kada smo izabrali na koji će način ugraditi vrijednosti sljedeće pitanje koje se postavlja je odabir broja težina i njihove vrijednosti. Kod odabira broja težina tu nema neke velike filozofije već je najbolje napraviti automatiziranu skriptu koja će za početak grafički prikazati kako se mijenja distribucija s obzirom na broj elemenata koje uzimamo za proračun. Za odabir vrijednosti težina se odlučilo ići u dvije varijante:

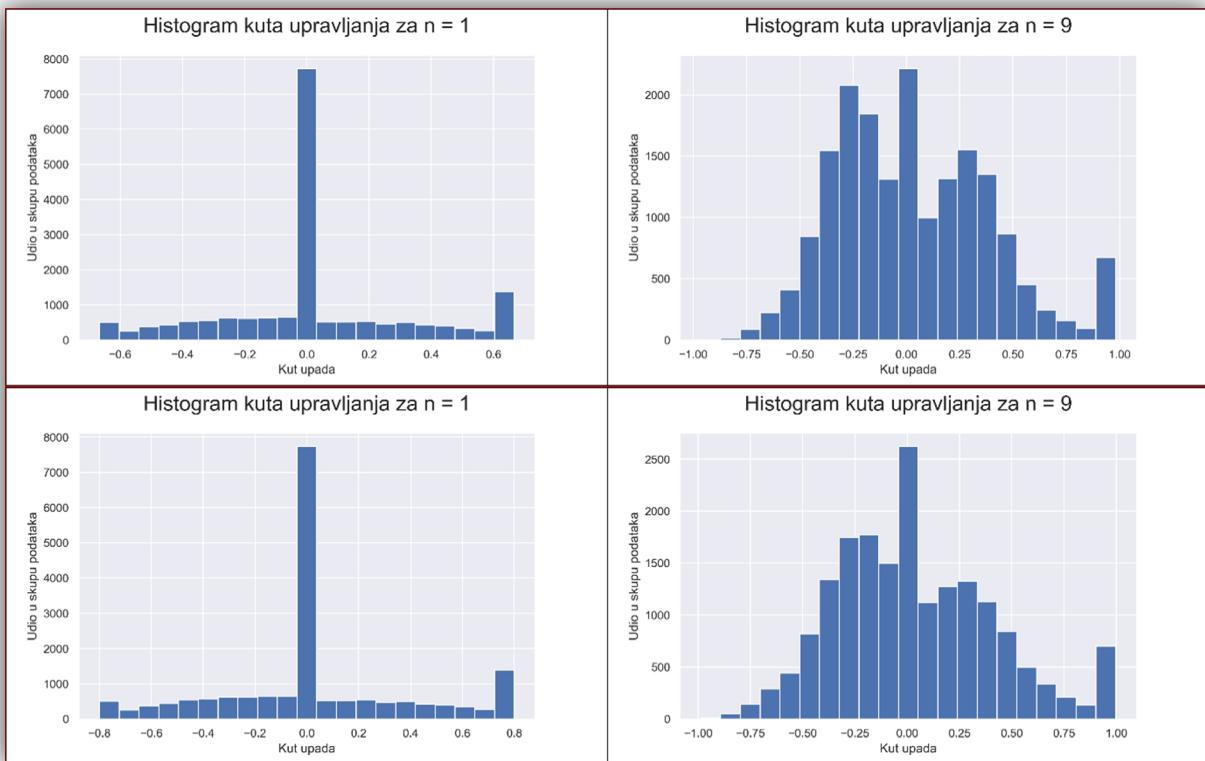
1. s linearnim vrijednostima,
2. s eksponencijalnim vrijednostima (drugog stupnja).

Za broj elemenata koje će se uzeti u obzir je odabran raspon od 1 do 15. Za prvu situaciju s linearnim vrijednostima odabrane su vrijednosti 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 i 16 dok za eksponencijalne vrijednosti težina su odabrane vrijednosti 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225 i 256. Na slici 5.9. možemo uočiti kako se mijenjaju vrijednosti kutova upada s mijenjanjem broja kutova koje uzimamo za proračun srednje aritmetičke vrijednosti.

```
Original data:
[0.00 0.15 0.35 0.50 0.70 0.85 0.00 0.00 0.00 0.00 0.05 0.25 0.40 0.55 0.75 0.95 0.00 0.00 0.15 0.30 0.50 0.70 0.85 1.00 1.00 0.00]
Data after mean calculation for n = 1
[0.07 0.25 0.42 0.60 0.77 0.42 0.00 0.00 0.00 0.03 0.15 0.33 0.48 0.65 0.85 0.47 0.00 0.07 0.22 0.40 0.60 0.77 0.93 1.00 0.50]
Data after mean calculation for n = 2
[0.17 0.33 0.52 0.68 0.52 0.28 0.00 0.00 0.00 0.02 0.10 0.23 0.40 0.57 0.75 0.57 0.32 0.05 0.15 0.32 0.50 0.68 0.85 0.95 0.67]
Data after mean calculation for n = 3
[0.25 0.42 0.60 0.51 0.39 0.21 0.00 0.00 0.01 0.07 0.17 0.31 0.49 0.66 0.56 0.42 0.27 0.11 0.24 0.41 0.59 0.76 0.89 0.71]
Data after mean calculation for n = 4
[0.34 0.51 0.48 0.41 0.31 0.17 0.00 0.01 0.06 0.14 0.25 0.40 0.58 0.53 0.45 0.37 0.28 0.19 0.33 0.50 0.67 0.81 0.71]
Data after mean calculation for n = 6
[0.36 0.36 0.34 0.29 0.22 0.13 0.04 0.10 0.18 0.29 0.42 0.42 0.41 0.40 0.39 0.38 0.37 0.36 0.50 0.64 0.62]
```

Slika 5.9. Promjena vrijednosti kutova s promjenom veličine „n“

Na slici 5.10. možemo u prvom redu grafikona vidjeti kako izgleda distribucija podataka upotrebom linearnih vrijednosti težina za $n = 1$ i $n = 9$ dok u drugom redu možemo vidjeti distribuciju kutova upada kada se za proračun kutova upotrijebe eksponencijalne vrijednosti težina. Osim što uspoređujemo kakva je razlika nastaje u zaglađivanju upotrebom različitih vrijednosti težina možemo vidjeti kako na zaglađivanje utječe broj vrijednosti koje smo uzeli u obzir pa tako u prvom stupcu grafikona se u obzir uzela samo još jedna vrijednost dok u drugom stupcu grafikona smo uz trenutnu vrijednost uzeli u obzir još 9 vrijednosti koje dolaze ispred trenutne vrijednosti.



Slika 5.10. Distribucija kutova za linearne i eksponencijalne vrijednosti težina

Računanje vrijednosti kuta upada se ostvaruje pomoću dvije funkcije. Prva funkcija je `process_driving_log_data()` i prima četiri argumenta od čega su nam najbitnija prva tri. Prvi argument je niz vrijednosti kutova upada. Drugi argument označava koliko ćemo još kutova upada uzeti u obzir osim trenutnog te treći argument je lista težina.

```

1. def process_driving_log_data(dataframe, n, weights, steering=''):
2.     post_processed_data = dataframe.iloc[:n, :3].copy()
3.     np_array_of_data = dataframe[steering].values
4.
5.     new_data = numpy.zeros((np_array_of_data.shape[0] - n), dtype=float)
6.     iteration = len(new_data)
7.
8.     for i in range(iteration):
9.         data = np_array_of_data[i:n + 1 + i]
10.        result = calc_weighted_mean_steering_angle(data, weights, n)
11.        new_data[i] = result
12.
13.    post_processed_data[steering] = new_data
14.    post_processed_data.reset_index(drop=True, inplace=True)
15.
16.    return post_processed_data

```

Funkcija `process_driving_log_data()` ne računa direktno nove vrijednosti težina već to radi funkcija `calc_weighted_mean_steering_angle()`. Funkcija `process_driving_log_data()` radi isječak iz cijelog niza vrijednosti kutova upada nad kojim će se provesti proračun te ga zajedno s vrijednostima težina i brojem koliko je dodatnih vrijednosti uzeto šalje u funkciju za proračun nove vrijednosti kuta upada.

```

1. def calc_weighted_mean_steering_angle(data, weights, n):
2.     result = 0
3.     weights_length = len(weights)
4.     norm_weights = numpy.zeros(weights_length)
5.     sum_of_weights = numpy.sum(weights)
6.
7.     for i in range(weights_length):
8.         norm_weights[i] = weights[i] / sum_of_weights
9.
10.    norm_weights = numpy.flip(norm_weights)
11.
12.    for i in range(n + 1):
13.        result += (data[i] * norm_weights[i])
14.
15.    return result

```

Funkcija prvo normalizira vrijednosti težina, a potom računa otežanu aritmetičku sredinu i rezultat proračuna vraća u prvu funkciju na spremanje vrijednosti.

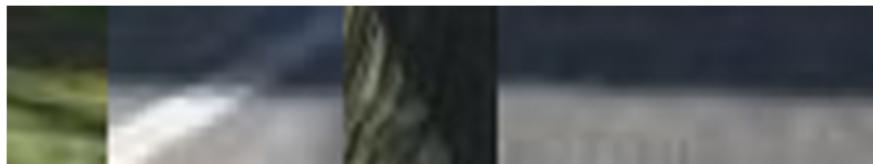
Dosadašnja priča o predobradi podataka je bazirana na obradi koja će se događa prije samog treninga. No jedan dio obrade podataka će se vršiti za vrijeme treninga, a takova obrada podataka ima za cilj na umjetan način proširiti postojeće podatke. Umjetno proširivanje podataka uključuje dodavanje sjene, prebacivanje iz RGB formata u BGR format te prilagođavanje kuta upada u slučaju odabira slike generirane od strane lijeve ili desne kamere s automobila.

Kod umjetnog proširivanja i predobrade podataka koje se događa za vrijeme treninga prvo se bira jedna od triju slika pomoću funkcije *choose_image()*.

```
1. def choose_image(data_dir, center, left, right, steering_angle):
2.     choice = numpy.random.choice(3)
3.
4.     adjust = 0.12
5.     if choice == 0:
6.         return load_image(data_dir, left), (steering_angle + adjust)
7.     elif choice == 1:
8.         return load_image(data_dir, right), (steering_angle - adjust)
9.     else:
10.        return load_image(data_dir, center), steering_angle
```

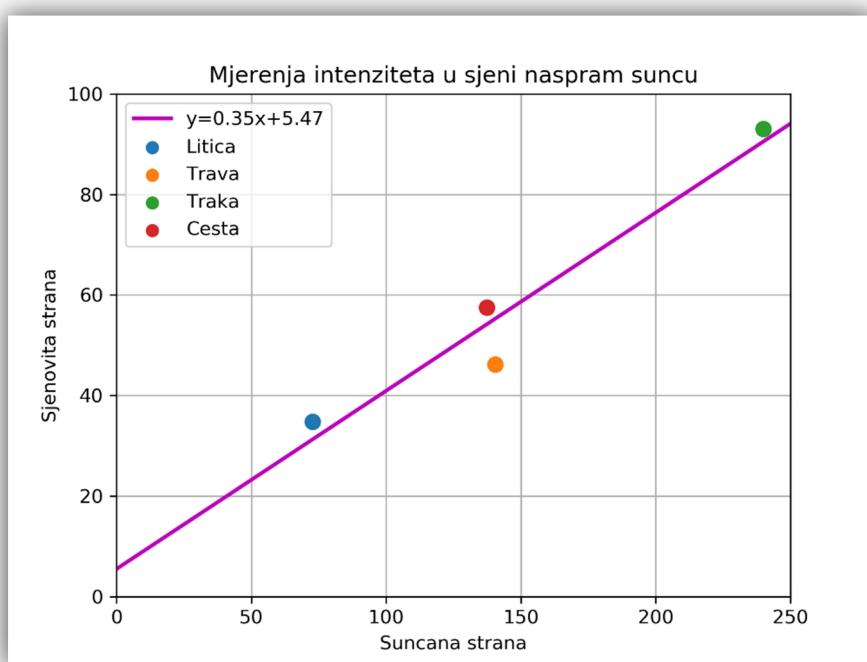
Nakon odabira slike provjerava se odabir slike jer u situaciji kada se odabere slika koja je generirana od strane lijeve ili desne kamere vrijednost kuta zakreta se treba podešiti. Razlog podešavanja je taj što za vrijeme testiranja odnosno vožnje automobila u autonomnom modu simulator (server) šalje samo slike od centralne kamere stoga moramo podešiti vrijednost kuta otklona za slike lijeve i desne kamere kako bi model mogao bolje naučiti se kretati unutar traka na stazi. Vrijednost varijable *adjust* nam služi da korigiramo vrijednost kuta otklona pri čemu se za lijevu sliku ta vrijednost dodaje na vrijednost kuta otklona dok za sliku dobivenu iz desne kamere vrijednost kuta se treba umanjiti za vrijednost na koju je postavljena varijabla *adjust*. Do vrijednosti varijable *adjust* došlo se eksperimentalnim putem odnosno drugim riječima trenirao se isti model nad istim skupom podataka, ali s drugom vrijednošću varijable *adjust*. U mojoj slučaju za moj model kao najbolja vrijednost se iskazala vrijednost od 0.12.

Ključna stavka za imati dobar model je umjetna dodavanje sjene u slici. Druga staza to jest džungla staza za razliku od prve staze uključuje sjenu što može značajno utjecati na performanse modela pogotovo na prijelaznim dijelovima između osunčane strane staze i staze u sjeni stoga je bitno istražiti kako rekreirati na umjetan način sjenu koja će biti najbliža stvarnoj sjeni koja je u simulatoru. Da bi rekreirali sjenu blisku sjeni kakva je prirodno u simulatoru trebamo naći poveznice između vrijednosti piksela na osunčanoj i sjenovitoj strani samih dijelova u simulatoru kao što su trava, litica, traka i pločnik (slika 5.11.).



Slika 5.11. Primjer trave, trake, litice i pločnika na osunčanoj i sjenovitoj strani

Jednu od boljih poveznica između vrijednosti piksela za navedene dijelove je pronađena u V komponenti HSV (Hue, Saturation, Value) formata slike gdje oznaka za V označava vrijednost. Ako bismo uzeli vrijednosti piksela pojedinih dijelova iz simulatora za osunčanu i sjenovitu stranu te prikazali vrijednosti piksela na grafu pri čemu bismo provukli najbolju moguću liniju za dobivene točke dobili bismo graf kao na slici 5.12.



Slika 5.12. Mjera intenziteta u sjeni naspram suncu za pojedine dijelove simulatora

Sa slike 5.12. možemo uočiti linearnu povezanost između vrijednosti piksela koji se nalaze na suncu i koji se nalaze u sjeni. Vrijednost komponente V za piksele u sjeni je manja za 0.35 puta od vrijednosti istog elementa i iste komponente koja se nalazi na sunčanoj strani. Implementacija i generiranje ovakve umjetne sjene ne bih trebao značajno utjecati na dugu

predobradu podataka. Implementacija sjene odvija se za vrijeme treninga kroz funkciju *random_shadow()*.

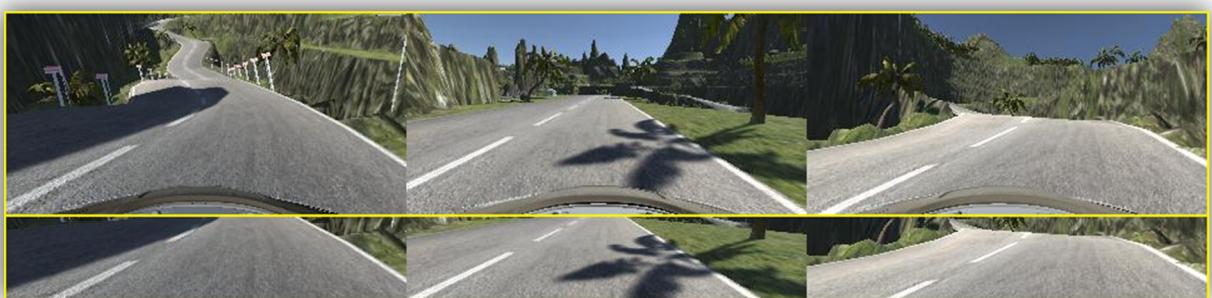
```
1. def random_shadow(image):
2.     global IMAGE_WIDTH
3.     global IMAGE_HEIGHT
4.
5.     x1, y1 = IMAGE_WIDTH * numpy.random.rand(), 0
6.     x2, y2 = IMAGE_WIDTH * numpy.random.rand(), IMAGE_HEIGHT
7.     x_m, y_m = numpy.mgrid[0:IMAGE_HEIGHT, 0:IMAGE_WIDTH]
8.
9.     mask = numpy.zeros_like(image[:, :, 2])
10.    mask[(y_m - y1) * (x2 - x1) - (y2 - y1) * (x_m - x1) > 0] = 1
11.
12.    cond = mask == numpy.random.randint(2)
13.    s_ratio = 0.35
14.
15.    hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
16.    hsv[:, :, 2][cond] = hsv[:, :, 2][cond] * s_ratio
17.
18.    image_shadow = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)
19.
20.    return image_shadow
```

Funkcija *random_shadow()* nasumično odabire dvije točke na gornjem i donjem rubu slike. Zatim se povlači linija između te dvije točke pri čemu linija dijeli sliku na lijevu i desnu stranu. Generira se nasumični broj koji će odlučiti hoće li se sjena dodati na lijevu ili desnu stranu. Prije ubacivanja umjetne sjene potrebno je sliku prebaciti iz RGB formata u HSV format. Potom se njezina treća V komponenta množi s faktorom 0.35. Za sami kraj slika se prebacuje u BGR format te u ovakvom obliku stiže na ulaz u model umjetne neuronske mreže. Na slici 5.13. u prvom redu slika možemo vidjeti originalnu sliku u RGB formatu zatim sliku u RGB formatu s dodanom sjenom te u drugom redu iste te slike samo u BGR formatu.



Slika 5.13. Primjer izvorne slike, slike s dodanom sjenom u RGB formatu i njihovi ekvivalenti u BGR formatu

U predobradu podataka upadaju još dvije stvari, a to su normalizacija vrijednosti piksela slike i rezanje slike. Normalizacijom se postiže da se vrijednosti piksela koje su originalno u rasponu od 0 do 255 pretvore u vrijednosti između -0.5 i 0.5. Ovime se želi postići da su vrijednosti centrirane oko nula s malom standardnom devijacijom. Rezanjem slike žele se ukloniti dijelovi (nebo, hauba, pozadina) koji nisu bitni u donošenju pravilne vrijednosti kuta upada. Ovim se ujedno smanjuje i broj parametara za treniranje što znači da ubrzava treniranje i kasnije kada model bude predviđao će brže i preciznije moći donijeti odluke. Vrijednosti rezanja slike doneseni su na osnovi eksperimentiranja pritom su u obzir bile uzete tri situacije. Pod situacije se smatra gdje se vozilo nalazi na stazi točnije rečeno jeli se vozilo nalazi na pravcu, nizbrdici ili uzbrdici. Kod rezanja donjeg dijela slike nije bilo većih problema jer hauba bez obzira na poziciju automobila uvijek zauzimala isti dio slike što je dovelo samo do nedoumice koliko odrezati od vrha slike s obzirom gdje se automobil našao. Nakon nekoliko eksperimenata odlučeno je ići s rezanjem prvih 70 redova slike koji će ukloniti nebo i drugu pozadinu te zadnjih 25 redova slike koji će odrezati haubu (slika 5.14.).



Slika 5.14. Originalne slike naspram izrezanim za nizbrdo, pravac i uzbrdo

Mora se naglasiti da su normalizacija vrijednosti piksela slike i rezanje slike implementirane kao prva dva sloja modela konvolucijske neuronske mreže.

5.3. Model umjetne neuronske mreže i treniranje

S obzirom da se radi s podacima koji imaju prostornu zavisnost, slika, odabir tipa umjetne neuronske mreže nije bio problem. Za takav tip podataka se odabire konvolucijska neuronska mreža. Sama srž konvolucijske neuronske mreže su konvolucijski sloj i sloj ujedinjavanja koji se dodaju prije potpuno povezanih slojeva.

Model korišten u ovom radu započinje sa slojem za normaliziranje vrijednosti piksela koji vrši normalizaciju nad cijelom slikom dimenzija 160x320x3. Zatim slijedi sloj za rezanje koji će odrezati nepotrebne dijelove prije nego se implementira prvi konvolucijski sloj. Dimenzije slike nakon rezanja iznose 65x320x3. Prvi konvolucijski sloj koristi filter veličine 3x3 pri čemu se pomiče za jedno polje. Sloj će kreirati 32 karte značajki. Nad slikom je dodan okvir kako bi ulazne dimenzije (visina i širina) slike ostale iste nakon primjene operacije konvolucije. Nakon konvolucijskog sloja slijedi aktivacijska funkcija. Kroz gotovo cijeli model se koristi ista aktivacijska funkcija i to Leaky ReLU s parametrom alfa u vrijednosti od 0.2. Jedini izuzetak je aktivacijska funkcija izlaznog sloja gdje se koristi hiperbolični tangens (tanh). Hiperbolični tangens izlazne vrijednosti ograničava u rasponu od -1 do 1 što su ujedno i odgovara maksimalnim normaliziranim vrijednostima kuta upada. Nakon prve aktivacijske funkcije

dolazi sloj ujedinjavanja. Koristi ste maksimalno ujedinjavanje s veličinom polja od 2x2. Ovim se slojem smanjuju dimenzije slike (visina i širina) za pola. U slučaju da je dimenzija neparan broj ona se smanjuje na prvu manju vrijednost nakon što se prvobitna vrijednost prepolovi. U ostaku modela se koriste veoma slični postavke konvolucijskog sloja kao u prvom. Ono što je isto za sve upotrijebljene konvolucijske slojeve je veličina filtra od 3x3, pomicanja za jedno polje i da visina i širina ostaju nepromijenjeni zbog dodavanja okvira oko slike. Isto tako iza svakog konvolucijskog sloja slijedi aktivacijska funkcija. Nakon prvog konvolucijskog sloja i sloja za ujedinjavanje implementirani su još šest konvolucijskih slojeva na način da su rađena po dva konvolucijska sloja u paru prije ponovnog ujedinjavanja piksela. Jedina razlika u pojedinim konvolucijskim slojevima je broj mapa značajki koji će proizvesti. Pa će tako implementirani treći konvolucijski sloj dati 48 mapa pa potom peti konvolucijski sloj će dati maksimalne 64 mape i s šestim slojem će se smanjiti na 32 mape značajki. Prva tri sloja ujedinjavanja će svaki put prepoloviti dimenzije širine i visine dok će posljednji sloj ujedinjavanja, četvrti, prepoloviti samo širinu slike. Nakon četvrtog sloja ujedinjavanja ide sloj za poravnavanje koji će transformirati ulazni vektor dimenzija 8x20x32 u vektor dimenzije 5120x1. Nakon sloja za poravnavanje dolazi potpuno povezana mreža neurona koja se sastoji od četiri sloja. Prvi sloj iliti ga ulazni sloj sastoji se od 196 neurona iza kojega slijedi Leaky ReLU aktivacijska funkcija. Zatim je implementirana regularizacija otpuštanja neurona čija je vrijednost postavljena na 0.2 i proteže se kroz sve slojeve osim izlaznog sloja. Drugi sloj se sastoji od 64 neurona dok treći sloj ima 16 neurona. Zadnji sloj to jest izlazni sloj sadrži samo 1 neuron koji daje vrijednost u rasponu od -1 do 1. Na slici 5.15. možemo vidjeti tipove slojeva zajedno s izlaznim dimenzijama iz svakog sloja te broj parametara po pojedinom sloju.

Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 160, 320, 3)	0
cropping2d_1 (Cropping2D)	(None, 65, 320, 3)	0
conv2d_1 (Conv2D)	(None, 65, 320, 32)	896
leaky_re_lu_1 (LeakyReLU)	(None, 65, 320, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 32, 160, 32)	0
conv2d_2 (Conv2D)	(None, 32, 160, 32)	9248
leaky_re_lu_2 (LeakyReLU)	(None, 32, 160, 32)	0
conv2d_3 (Conv2D)	(None, 32, 160, 48)	13872
leaky_re_lu_3 (LeakyReLU)	(None, 32, 160, 48)	0
max_pooling2d_2 (MaxPooling2D)	(None, 16, 80, 48)	0
conv2d_4 (Conv2D)	(None, 16, 80, 48)	20784
leaky_re_lu_4 (LeakyReLU)	(None, 16, 80, 48)	0
conv2d_5 (Conv2D)	(None, 16, 80, 64)	27712
leaky_re_lu_5 (LeakyReLU)	(None, 16, 80, 64)	0
max_pooling2d_3 (MaxPooling2D)	(None, 8, 40, 64)	0
conv2d_6 (Conv2D)	(None, 8, 40, 64)	36928
leaky_re_lu_6 (LeakyReLU)	(None, 8, 40, 64)	0
conv2d_7 (Conv2D)	(None, 8, 40, 32)	18464
leaky_re_lu_7 (LeakyReLU)	(None, 8, 40, 32)	0
max_pooling2d_4 (MaxPooling2D)	(None, 8, 20, 32)	0
flatten_1 (Flatten)	(None, 5120)	0
dense_1 (Dense)	(None, 196)	1003716
leaky_re_lu_8 (LeakyReLU)	(None, 196)	0
dropout_1 (Dropout)	(None, 196)	0
dense_2 (Dense)	(None, 64)	12608
leaky_re_lu_9 (LeakyReLU)	(None, 64)	0
dropout_2 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 16)	1040
leaky_re_lu_10 (LeakyReLU)	(None, 16)	0
dropout_3 (Dropout)	(None, 16)	0
dense_4 (Dense)	(None, 1)	17

Slika 5.15. Prikaz slojeva modela s vrijednostima izlaznog oblika i brojem parametara

Izračun broja parametara je vrlo jednostavan i u našem slučaju trebaju nam samo dvije formule. Jedna od njih će nam poslužiti za izračun parametara u konvolucijskom sloju dok će druga formula dati broj parametara za potpuno povezane slojeve. Jednadžba 5.3. nam daje broj parametara za konvolucijski sloj.

$$\text{Broj parametara} = (\text{veličina filtera} * \text{broj ulaznih značajki} + 1) * \text{broj izlaznih značajki} \quad (5.3.)$$

Primjena jednadžbe 5.3. na prvi konvolucijski sloj:

$$\text{Veličina filtera} = 3 \times 3$$

$$\text{Broj ulaznih značajki} = 3$$

$$\text{Broj izlaznih značajki} = 32$$

$$\text{Broj parametara} = (3 * 3 * 3 + 1) * 32 = 896$$

Primjena jednadžbe 5.3. na drugi konvolucijski sloj:

$$\text{Veličina filtera} = 3 \times 3$$

$$\text{Broj ulaznih značajki} = 32$$

$$\text{Broj izlaznih značajki} = 32$$

$$\text{Broj parametara} = (3 * 3 * 32 + 1) * 32 = 9,248$$

Jednadžba 5.4. nam daje broj parametara za potpuno povezani sloj.

$$\text{Broj parametara} = \text{neurona u prethodnom sloju} * \text{neurona u trenutnom sloju} + \text{neurona u trenutnom sloju} \quad (5.4.)$$

Primjena jednadžbe 5.4. na prvi (ulazni) sloj potpuno povezane mreže:

$$\text{Neurona u prethodnom sloju} = 5120$$

$$\text{Neurona u trenutnom sloju} = 196$$

$$\text{Broj parametara} = 5120 * 196 + 196 = 1,003,716$$

Primjena jednadžbe 5.4. na drugi sloj potpuno povezane mreže:

Neurona u prethodnom sloju = 196

Neuron u trenutnom sloju = 64

$$\text{Broj parametara} = 196 * 64 + 64 = 12,608$$

Nakon što se na ovakav način proračuna broj parametara za pojedini sloj potrebno ih je zbrojiti kako bi se dobio ukupan broj parametara modela. Kod modela korištenog u ovom radu ukupan broj parametara iznosi 1,145,285 od čega je 1,145,285 parametara za treniranje odnosno svi parametri koji postoje će se podešavati. Možemo isto tako uočiti da ukupnom broju parametara najviše doprinosi broj parametara koji se nalaze između sloja za poravnavanje i ulaznog sloja potpuno povezane mreže (~87%). U nastavku možemo vidjeti funkciju *build_model()* koja kreira prethodno opisani model umjetne neuronske mreže.

```
1. def build_model(name, input_shape=(160, 320, 3)):
2.     model = Sequential()
3.
4.     model.add(Lambda(lambda x: (x / 255.0) - 0.5, input_shape=input_shape))
5.
6.     model.add(Cropping2D(cropping=((70, 25), (0, 0))))
7.
8.     model.add(Conv2D(32, (3, 3), strides=(1, 1), padding='same'))
9.     model.add(LeakyReLU(alpha=0.2))
10.
11.    model.add(MaxPool2D(pool_size=(2, 2)))
12.
13.    model.add(Conv2D(32, (3, 3), strides=(1, 1), padding='same'))
14.    model.add(LeakyReLU(alpha=0.2))
15.    model.add(Conv2D(48, (3, 3), strides=(1, 1), padding='same'))
16.    model.add(LeakyReLU(alpha=0.2))
17.
18.    model.add(MaxPool2D(pool_size=(2, 2)))
19.
20.    model.add(Conv2D(48, (3, 3), strides=(1, 1), padding='same'))
21.    model.add(LeakyReLU(alpha=0.2))
22.    model.add(Conv2D(64, (3, 3), strides=(1, 1), padding='same'))
23.    model.add(LeakyReLU(alpha=0.2))
24.
25.    model.add(MaxPool2D(pool_size=(2, 2)))
26.
27.    model.add(Conv2D(64, (3, 3), strides=(1, 1), padding='same'))
28.    model.add(LeakyReLU(alpha=0.2))
29.    model.add(Conv2D(32, (3, 3), strides=(1, 1), padding='same'))
30.    model.add(LeakyReLU(alpha=0.2))
31.
32.    model.add(MaxPool2D(pool_size=(1, 2)))
33.
34.    model.add(Flatten())
35.
36.    model.add(Dense(196))
```

```

37.     model.add(LeakyReLU(alpha=0.2))
38.     model.add(Dropout(rate=0.2))
39.
40.     model.add(Dense(64))
41.     model.add(LeakyReLU(alpha=0.2))
42.     model.add(Dropout(rate=0.2))
43.
44.     model.add(Dense(16))
45.     model.add(LeakyReLU(alpha=0.2))
46.     model.add(Dropout(rate=0.2))
47.
48.     model.add(Dense(1, activation='tanh'))
49.
50.     model.summary()
51.
52.     tensorboard = TensorBoard(log_dir=f'.\logs\{name}', write_images=True,
53.                               write_graph=False)
54.
55.     return model, tensorboard

```

Na samom kraju funkcije je dodan TensorBoard koji će nam poslužiti za vizualizaciju funkcije gubitaka.

Treniranje modela obavlja funkcija *train_model*, a sastavljena je iz tri dijela.

```

1. def train_model(model, tensorboard, name, X_train, X_valid, y_train, y_valid):
2.     checkpoint = ModelCheckpoint(f'{name}-{epoch:03d}.h5',
3.                                   monitor='val_loss')
4.
5.     model.compile(loss='mae',
6.                   optimizer=Adam(lr=FLAGS.learning_rate),
7.                   metrics=['mse', 'mape', 'cosine'])
8.
9.     model.fit_generator(batch_generator(FLAGS.driving_log_dir, X_train,
10.                                         y_train, FLAGS.batch_size, True),
11.                         steps_per_epoch=FLAGS.steps_per_epoch,
12.                         epochs=FLAGS.epochs,
13.                         max_queue_size=1,
14.                         validation_data=batch_generator(
15.                             FLAGS.driving_log_dir, X_valid, y_valid,
16.                             FLAGS.batch_size, False),
17.                         nb_val_samples=len(X_valid),
18.                         callbacks=[checkpoint, tensorboard],
19.                         verbose=2)

```

Prvi dio predstavlja *ModelCheckpoint* koji nam služi za spremanje modela nakon svake epohe. Drugi dio predstavlja konfiguracija procesa učenja koja se mora definirati prije početka samog procesa učenja u ona je obavljena pomoću metode *model.compile()*. Funkcija *compile()* prima tri argumenta:

- Optimizator (Adam optimizator)
- Funkcija gubitaka (MAE odnosno srednja apsolutna pogreška)

- Popis mjernih podataka (MSE, MAPE, cosine)

Popis mjernih podataka je funkcija ili skup funkcija koje se mogu iskoristiti za prosudbu performansi modela što znači da je veoma slična funkciji cijene (gubitaka), osim što se rezultati ocjenjivanja metrike ne koriste prilikom obuke modela. Treći dio to jest zadnji dio funkcije *train_model* predstavlja funkcija *fit_generator()* koja služi za treniranje modela na podacima koji generiraju kao serija (hrpa) pomoću Python generatora. Generiranje serije podataka obavlja paralelno s modelom radi povećanja učinkovitosti. Ovim se omogućuje da u stvarnom vremenu se vrši povećanje podataka slika na procesoru dok u paraleli se vrši obuka modela na jedinici za obradu grafike (GPU).

Prvi parametar funkcije *fit_generator()* je sami generator, a u mojoj situaciji je napisana posebna funkcija za generator te se zove *batch_generator()* i njezin kod se može vidjeti u nastavku.

```

1. def batch_generator(data_dir, img_paths, steering_angles, batch_size,
2.                      is_training):
3.     images = numpy.empty([batch_size, FLAGS.height, FLAGS.width,
4.                          FLAGS.channels])
5.     steers = numpy.empty(batch_size)
6.
7.     while True:
8.         i = 0
9.         for index in numpy.random.permutation(img_paths.shape[0]):
10.             center, left, right = img_paths[index]
11.             steering_angle = steering_angles[index]
12.
13.             if is_training and numpy.random.rand() < 0.6:
14.                 image, steering_angle = augment_data(data_dir, center, left,
15.                                         right, steering_angle)
16.             else:
17.                 image = load_image(data_dir, center)
18.
19.             images[i] = image
20.             steers[i] = steering_angle
21.
22.             i += 1
23.             if i == batch_size:
24.                 break
25.
26.     yield images, steers

```

Funkcija *batch_generator()* posjeduje pet argumenata. Prvi argument upućuje funkciju na direktori u kojem se nalazi .csv datoteka podataka. Drugi argument je lista apsolutnih puteva slika, treći argument je lista vrijednosti kutova upada dok je četvrti argument veličina serije (skupa) podataka za jedan korak funkcije treninga i peti argument je boolean varijabla koja će

za vrijeme treninga poprimiti vrijednost *True* dok za vrijeme validacije vrijednost joj je *False*. Služi za umjetno povećanje podataka koje se vrši samo za vrijeme treninga i to na nasumično odabranim instancama. Funkcija kreira dva niza pri čemu će jedan sadržavati slike dok će drugi sadržavati pripadajuće vrijednosti kutova upada. Za kreiranje skupa (hrpe) podataka koristi se generator (*yield*) zbog svojih performansi.

Fit_generator() još kao parametre prima broj koraka po epohi, broj epoha, maksimalna veličina za red generatora, za validacijske podatke opet iskorištavamo funkciju *batch_generator()*, broj uzoraka u validacijskom skupu te potom definiramo listu povratnih uzoraka koji služe za uvid u unutarnja stanja i statistiku modela tijekom treninga. Za kraj se definirao *verbose* na vrijednost 2 što znači da se statistika modela prikazivala samo na kraju svake epohe.

Kada se dovršila planirana predobrada podataka i definirala predobrada i umjetno povećanje podataka koje će se vršiti za vrijeme treninga u praksi se prakticira jedna od iduće dvije opcije:

1. Obaviti nekoliko epoha treninga na cijelom skupu podataka.
2. Izvući manji skup podataka iz cijelog skupa podataka za treniranje i obaviti trening.

Prakticiranje jedne od ove dvije opcije nam služe kako bi mogli vidjeti jeli potrebna još kakova predobrada i umjetno povećanje podataka, ali isto tako jesmo li odabrali dobre vrijednosti hiperparametara za treniranje. Na ovaj način ne moramo čekati puno vrijeme treninga koje može potrajati satima pa čak i danima ili tjednima već u vrlo kratkom roku možemo vidjeti kako model napreduje. Ja sam se poslužio opcijom pod brojem 1 gdje sam pustio da se model trenira nekoliko epoha nad cijelim skupom podataka što je vremenski zahtjevalo oko 1.5 sat naspram potpunom treningu kojemu je trebalo oko 10 sati. Na ovaj način sam uspio doći do spoznaje oko odabira najbolje funkcije gubitaka, odabiru vrijednosti korekcije kuta upravljanja kada se odabire lijeva ili desna slika te gdje model najviše zapinje na stazi, što je za posljedicu imalo da sam generirao dodatne podatke za najproblematičnija mesta. U gore navedenim isjećcima kodova može se vidjeti dio konačno upotrijebljenih parametra dok ostatak parametara se može vidjeti u nastavku.

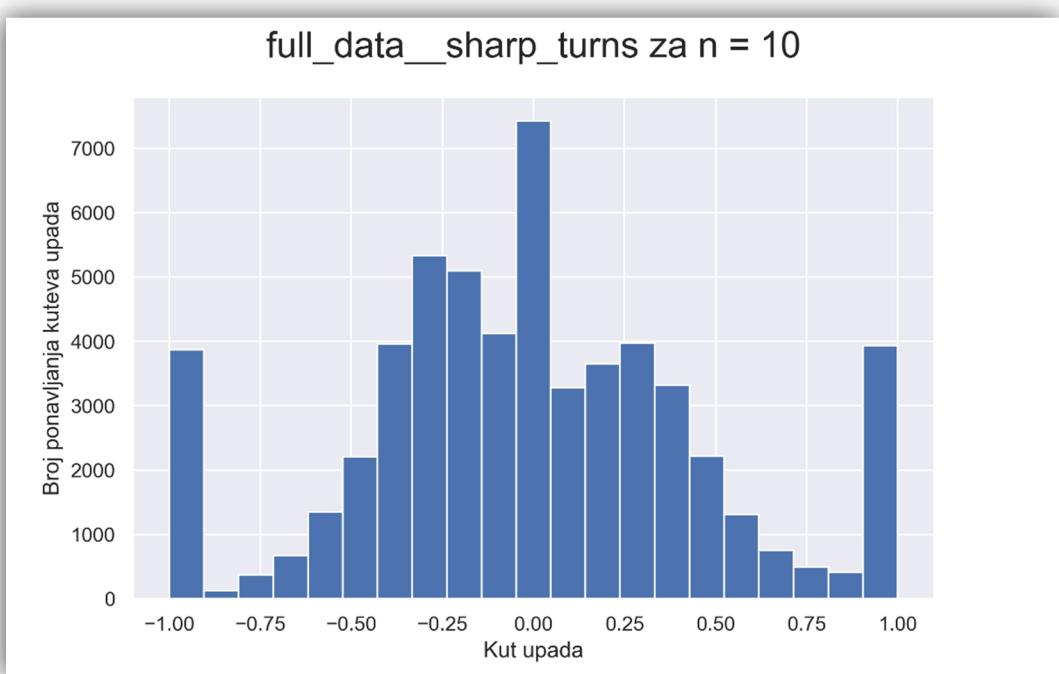
```
1. flags.DEFINE_integer('epochs', 36, 'Number of epochs to train')
2. flags.DEFINE_integer('batch_size', 64, 'Batch size')
3. flags.DEFINE_integer('train_size', 49208, 'Size of training data')
4. flags.DEFINE_integer('steps_per_epoch', (FLAGS.train_size //
```

```

5.                                     FLAGS.batch_size), 'Number of steps')
6. flags.DEFINE_float('test_size', 0.15, 'Size of test data')
7. flags.DEFINE_float('learning_rate', 0.0001, 'Learning rate of the model')
8. flags.DEFINE_integer('height', 160, 'Height of image')
9. flags.DEFINE_integer('width', 320, 'Width of image')
10. flags.DEFINE_integer('channels', 3, 'Number of channels of image')

```

Konačna distribucija podataka se može vidjeti na slici 5.16.

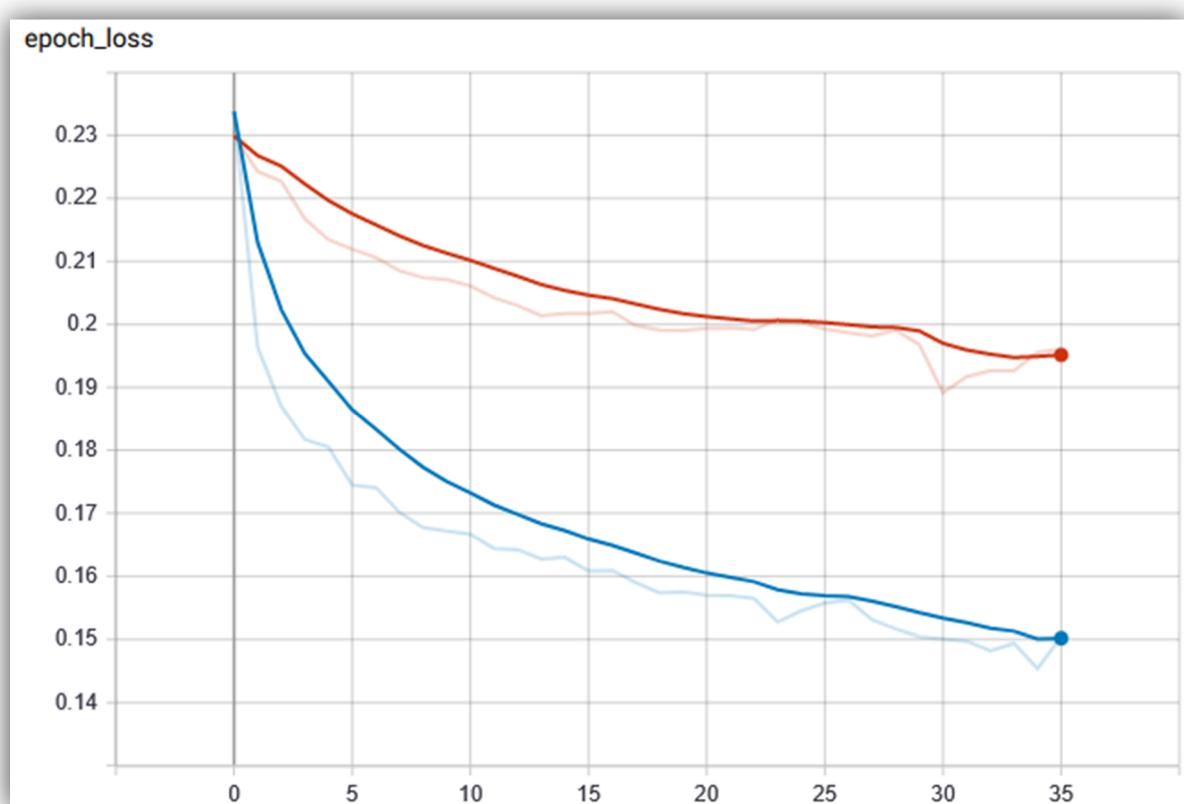


Slika 5.16. Distribucija kutova upada nad kojom je izvršen trening, a dobiven najbolji rezultat

5.4. Rezultat treniranja i testiranje

Nakon završetka treniranja prva stvar koja se čini je pregled grafa krivulje funkcije gubitaka, a potom ako su postavljene još neke funkcije metrike kao u mojoj slučaju onda se pregledavaju i grafovi tih funkcija. Uz dobro obrađene podatke i dobro odabranu funkciju cijene u najboljem slučaju graf bi trebao sličiti grafu kao na slici 4.14. Iz grafa sa slike 4.14. može se vidjeti gdje model ima najoptimalnije vrijednosti težina te bi trebalo ispitati model s težinama oko te vrijednosti.

Finalna upotrijebljena funkcija gubitaka je srednja apsolutna pogreška (MAE) jer nam daje praktično najbolje rezultate, ali njezin graf ne sliči grafu sa slike 4.14. Stoga se trebao svaki model testirati dok se ne nađe najbolji model po procjeni testera. Na slici 5.17. može se vidjeti graf cijene funkcije za opisani model zajedno s navedenim parametrima te prikazanom distribucijom podataka sa slike 5.16.



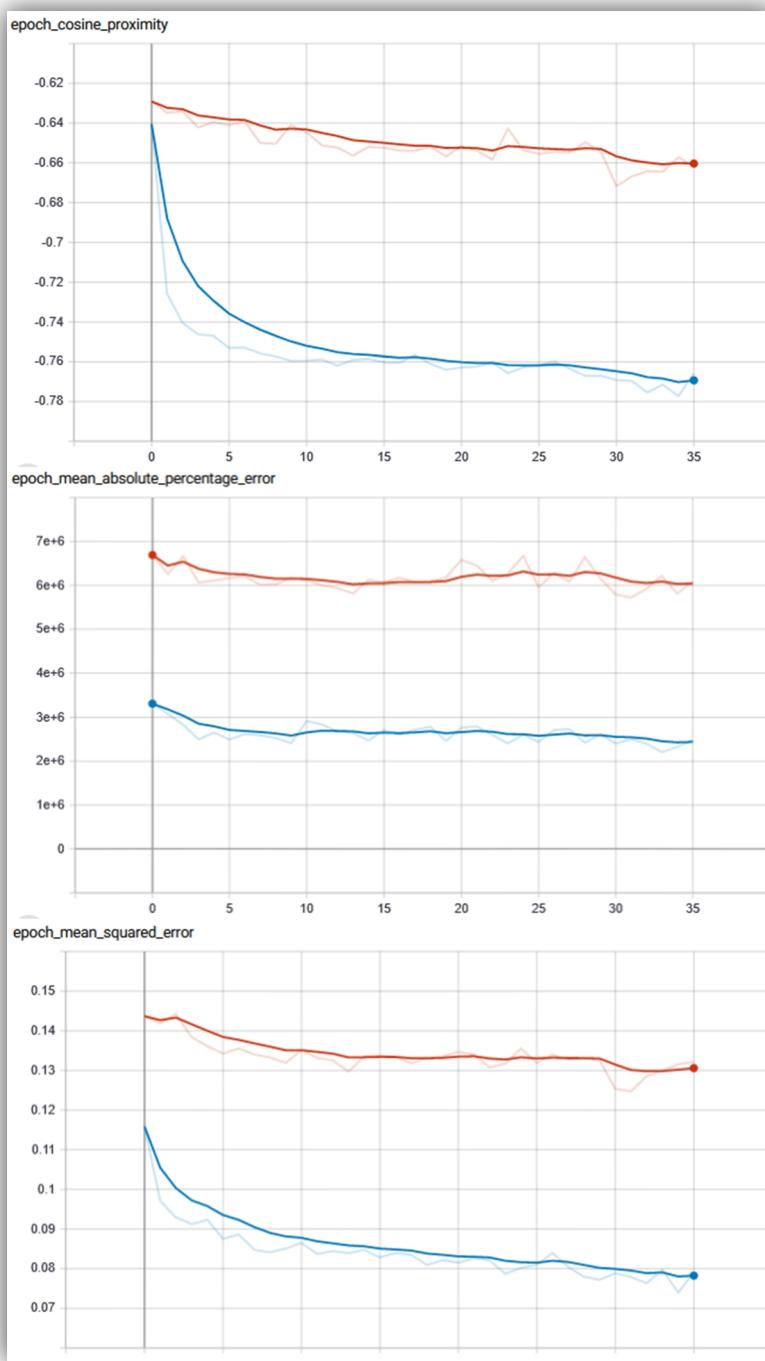
Slika 5.17. Cijena funkcije za opisani model i distribuciju podatka sa slike 5.16.

S grafa koji se nalazi na slici 5.17. plavom bojom je označena cijena gubitaka nad trening podacima dok rozom bojom označeni validacijski gubici. Naglašenom bojom vidimo krivulje koje su dobivene nakon zaglađivanja krivlje s faktorom 0.8, dok je izblijedenom bojom prikazana krivulja koja se dobije spajanjem ravnom linijom dvije susjedne vrijednosti gubitaka. Plava linija odgovara očekivanom rezultatu, a to je da gubici nad trening podacima konstanto opadaju, ali krivulja gubitaka nad validacijskim podacima (crvena) za vrijeme od 36 epoha

nigdje ne počinje rasti kako bismo mogli reći da od tog trenutka počinje pretreniranje. Može se zapitati onda jeli broj epoha previše nisko postavljen. Nakon testiranja svih težina dolazi se do zaključka da to nije situacija jer u načinu vožnje modeli nakon 30-e epohe pokazuju znakove pretreniranja. Najbolje težine se najčešće nalaze u rasponu od 15 do 30 epohe.

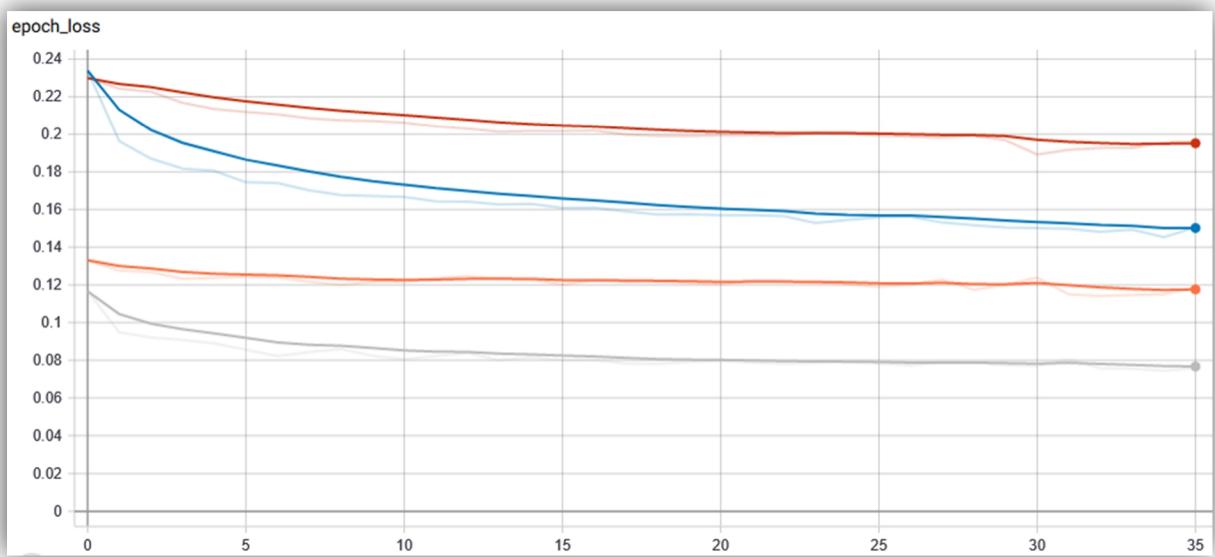
Nakon prve epohe vrijednosti gubitaka za trening iznosi 0.2338 dok validacijski gubici su 0.2295. Prelaskom na vrijednosti gubitaka druge epohe imamo najveći pad pa tako gubici nad trening podacima iznosi 0.1964 dok validacijski podaci iznose 0.2243. Prolaskom po treći put kroz podatke iznosi gubitaka su 0.187 i 0.2227. Sa svakom idućom epohom razlika između susjednih gubitaka je sve manja i manja dok se ne dođe do 15 epohe koja po prvu puta ima veće gubitke u trening podacima nego u epohi prije. Slična se situacija dogodila i s validacijskim gubicima samo je ovdje trebala jedna epoha više da gubici budu veći od prethodnih. Nakon 15 odnosno 16 epohe pa sve do 36 epohe događaju se blage oscilacije u vrijednostima gubitaka. Nekoliko epoha padaju pa ponovno rastu pa opet padaju i tako naizmjenice dokle ne dođe kraj treniranja. Završne vrijednosti gubitaka u 36-oj epohi iznose 0.1505 nad trening podacima i 0.1961 nad validacijskim podacima.

Na slici 5.18. možemo redom vidjeti grafove kosinusove funkcije, srednje apsolutne postotne pogreške i funkcija srednje kvadratne pogreške. Oznake krivulja iste kao i na grafu sa slike 5.17. Nakon pregleda grafova upotrijebljenih u polju *metrics* koji nam daje metriku drugih funkcija (MSE, MAPE, cosine) isto tako nam ne olakšavaju niti sužuje izbor najboljih težina što se može vidjeti sa slike u nastavku.



Slika 5.18. Funkcije metrike za odabrani finalni model i odabране finalne podatke

Na slici 5.19. možemo vidjeti usporedbu između cijena gubitaka pri čemu svijetlo plava i crvena boja pripadaju MAE funkciji gubitaka dok siva i narančasta pripadaju MSE funkciji gubitaka.

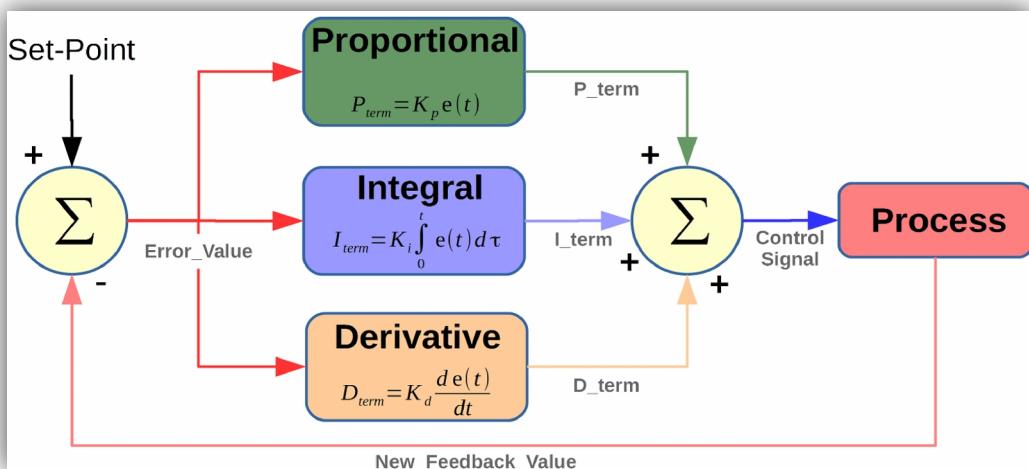


Slika 5.19. Usporedba cijene gubitaka MAE naspram MSE

Kada se prati funkcija gubitaka (cijene) tada je bolji model koji ima manje gubitke. Praktično to jest nakon testiranja najbolje performanse postignute za MAE kao funkciju cijene iako su očigledno vrijednosti gubitaka veće od vrijednosti gubitaka kada se za funkciju cijene koristila MSE funkcija. Jednim dijelom se može objasniti time što MSE preferira da distribucija ciljane varijable bude Gauss-ovog oblika i pravljenjem većih pogrešaka MSE više kažnjava nego radeći manje pogreške. S druge strane funkcija cijene MAE je prikladna u slučaju kada u skupu podataka postoji nešto više ekstremnih vrijednosti ili vrijednosti koje iskaču iz zadanih okvira, a u slučaju ovoga rada to bi bile vrijednosti -1 i 1.

Da bi se automobil uspješno kretao stazom potrebne se dvije komande. Prva komanda bi bio kut upravljanja, kojega predviđa model umjetne neuronske mreže, a druga komanda bi bila gas odnosno kočnica. Simulator je izведен tako da ako se šalje pozitivna vrijednost preko varijable *throttle* automobil će ubrzati, ako se šalje negativna vrijednost automobil će kočiti. Za postizanje i održavanje zadane brzine implementiran je proporcionalni-integralni-derivacijski (PID) regulator.

PID regulator je mehanizam upravljačke petlje koji koristi povratne informacije potrebne za kontinuirano modulirano upravljanje. Povratne informacije bi bili podaci od senzora brzinomjera koje simulator kontinuirano šalje zajedno sa slikom iz centralne kamere. Pomoću PID algoritma se izračunava vrijednost greške kao razlika između željene zadane vrijednosti i izmjerene procesne vrijednosti. Da bi se primijenila korekcija kako bi se došlo s izmjerene na zadatu vrijednost, računanjem proporcionalne, integralne i derivativne komponente te zbrajanjem tih triju komponenti dobiva se izlazna odnosno korekcijska vrijednost (slika 5.20.).



Slika 5.20. Blok dijagram PID regulatora [30]

PID regulator je izведен kao klasa *PIDController*, a kod klase se može vidjeti u nastavku.

```

1. class PIDController:
2.     def __init__(self, Kp, Ki, Kd):
3.         self.Kp = Kp
4.         self.Ki = Ki
5.         self.Kd = Kd
6.         self.speed = 0.
7.         self.errors = [0] * 29
8.
9.     def set_speed(self, speed):
10.        self.speed = speed
11.
12.    def update(self, measurement):
13.        self.errors.append(self.speed - measurement)
14.
15.        output = (self.Kp * self.errors[-1] +
16.                  self.Kd * self.errors[-2] +
17.                  self.Ki * sum(self.errors))
18.

```

```

19.     if (output > -0.3) and (output < 0.0):
20.         output = 0.0
21.
22.     self.errors.pop(0)
23.
24.     return output

```

Proporcionalna komponenta prilagođava brzinu dodavanjem vrijednosti pogreške (razlika zadane i stvarne brzine pomnožene s koeficijentom K_p). Derivacijska komponenta promatra koliko se brzo ili sporo mijenja pogreška. Pomaže u izglađivanju brzine promjene i sprečavanju nestalnih promjena u brzini. To se postiže uzimajući u obzir prethodnu grešku pomnoženu s koeficijentom K_d prilikom izračuna izlazne vrijednosti. Integralna komponenta pomaže u postizanju stabilnog stanja, a to čini množeći koeficijent K_i sa sumom svih grešaka. Sumom ovih vrijednosti dobiva se izlazna vrijednost (varijabla $output$). Do vrijednosti koeficijenata se dolazi eksperimentiranjem, a ja sam našao da mi najbolje odgovaraju sljedeće vrijednosti: [31]

- $K_p = 0.7$
- $K_i = 0.001$
- $Kd = 0.4$

Prvo što se radi je učitavanje modela zajedno s težinama. Model i težine su spremljene u datoteku tipa *.h5*, a Keras modul posjeduje metodu za čitanje takovih datoteka. Simulator se u autonomnom modu ponaša kao server što znači da treba napraviti skriptu koja će se ponašati kao aplikacija koja se povezuje na server. Skripta kreira tri instance, a to su instanca za povezivanje na server, Flask instancu te instancu PIDController. Uz pomoć funkcije *connect()* ostvaruje se povezivanje između simulatora i skripte. S obzirom da se simulator vrti na istom računalu kao i skripta tada je postavljanje vrijednosti kod metode *listen()* (zaprimaj što ti server šalje) pojednostavljen jer se vrijednost IP postavlja na prazan niz znakova (,,“) dok se za vrijednost porta po dokumentaciji postavlja na 4567.

```

1. sio = socketio.Server()
2. app = Flask(__name__)
3. controller = PIDController(0.7, 0.001, 0.4)      # Kp, Ki, Kd
4.
5. @sio.on('connect')
6. def connect(sid, environment):
7.     print('Connect ', sid)
8.     send_control(0, 0)

```

Isječak *main()* programa se može vidjeti u nastavku.

```

1. parser.add_argument('model',
2.                     type=str,
3.                     help='Path to model h5 file. Model should be on the '
4.                           'same path.')
5.
6. f = h5py.File(args.model, mode='r')
7.
8. app = socketio.Middleware(sio, app)
9. eventlet.wsgi.server(eventlet.listen(('', 4567)), app)

```

Server (simulator) nam šalje niz znakova zapakiranih u listu iz koje se potom podaci izvlače u zasebne varijable. Za potrebe ovoga rada su najbitniji podaci koji se nalaze u stupcima *image* koji nam prosljeđuje sliku i stupac *speed* koji nam prosljeđuje trenutnu vrijednost brzine automobila. Slika se zaprima kao niz znakova koji se prvo dekodira zatim pretvara u niz bitova i na kraju taj niz bitova postaje *Image* objekt. Objekt je moguće pretvoriti u *numpy* matricu koja se dalje prosljeđuje OpenCV modulu za pretvorbu iz RGB u BGR format. Matrica takove slike se predaje modelu kako bi izvršio predviđanje kuta upravljanja. Još je jedino ostalo dobiti korekcijsku vrijednost kako bi se automobil kretao postavljrenom brzinom. Dobivanjem korekcijske vrijednosti i imajući dobiveni novi kut upravljanja sve je spremno za povratak vrijednosti prema automobilu koji se obavlja funkcijom *send_control()*.

```

1. @sio.on('telemetry')
2. def telemetry(sid, data):
3.     if data:
4.         steering_angle = data['steering_angle']
5.         throttle = data['throttle']
6.         speed = data['speed']
7.         image_string = data['image']
8.
9.         image = Image.open(BytesIO(base64.b64decode(image_string)))
10.        image_array = numpy.asarray(image)
11.        image_array = cv2.cvtColor(image_array, cv2.COLOR_RGB2BGR)
12.
13.        steering_angle = float(model.predict(image_array[None, :, :, :], batch_size=1))
14.
15.
16.        throttle = controller.update(float(speed))
17.
18.        print(f'{steering_angle:.3f}\t{throttle:.2f}', flush=True)
19.
20.        sys.stdout.flush()
21.        send_control(steering_angle, throttle)
22.
23.    else:
24.        # NOTE: DON'T EDIT THIS.
25.        sio.emit('manual', data={}, skip_sid=True)

```

Funkcija *send_control()* za argumente ima dvije varijable koje treba proslijediti automobilu. Prije nego ih se upakiraju u rječnik (tip podatka u programskom jeziku Python) potrebno ih je pretvoriti u niz znakova.

```

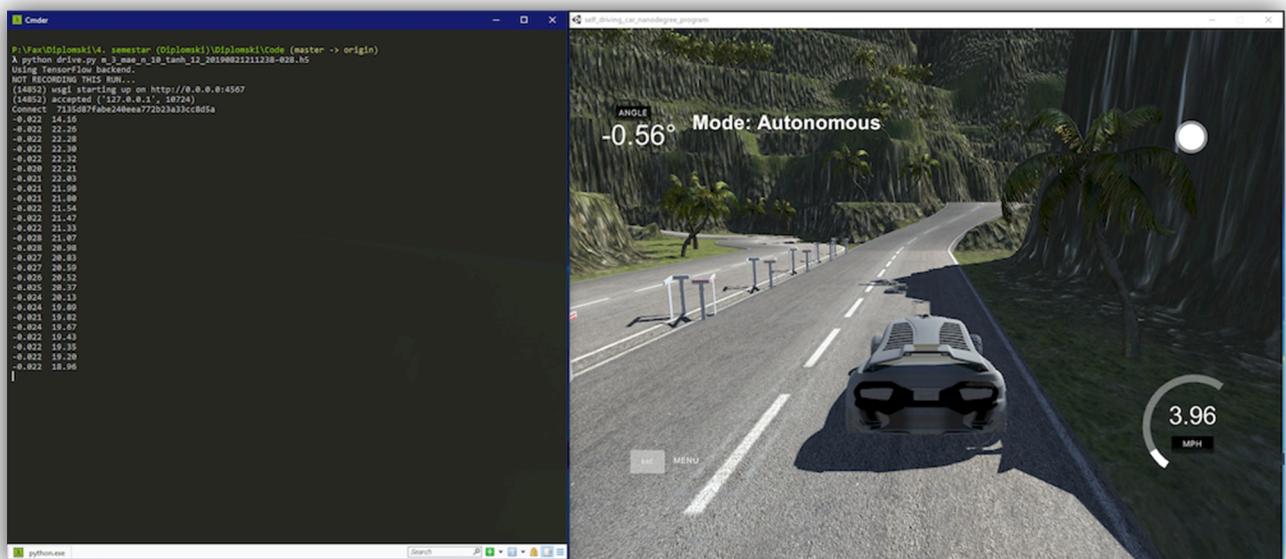
1. def send_control(steering_angle, throttle):
2.     sio.emit('steer',
3.             data={
4.                 'steering_angle': steering_angle.__str__(),
5.                 'throttle': throttle.__str__()
6.             },
7.             skip_sid=True)

```

Skripta se pokreće tako da otvorimo komadnu liniju (cmd) te dođemo do mape u kojoj se nalazi skripta za vožnju (drive.py). U istom direktoriju se treba nalaziti i model spremlijen u datoteci tipa .h5. Zatim se upisuje sljedeća komanda:

```
python ime_skripte.py ime_dateteke_modela_s_težinama.h5
```

U međuvremenu treba se pokrenuti simulator i odabratи autonomni način rada. Pričeka se dok se ostvari veza, inicijalizira potrebna memorija te učita i pokrene se model. Nakon što se svi koraci uspješno odrade automobil bi se trebao početi autonomno kretati, bez našega upletanja i to u najboljem slučaju onako kako se postavilo kao krajnji cilj. Na slici 5.21. možemo vidjeti kako izgleda jedan trenutak (okvir) autonomne vožnje.



Slika 5.21. Jedan okvir izgleda autonomne vožnje

6. ZAKLJUČAK

U današnje vrijeme vožnja automobila zahtjeva posvemašnu pozornost vozača kako bi se rizik od nezgode sveo na najmanju moguću mjeru. Ubrzani tempo i užurbani život ljudi ponekad zahtijevaju od ljudi da šalju SMS poruke ili obavljaju telefonske pozive među ostalim za vrijeme vožnje automobila što je jako nesigurno i opasno kako za same vozače tako i za ostale sudionike u prometu, a takove radnje za vrijeme vožnje u mnogim državama je protivno zakonu. Pozivanjem taksija da vas odvede do vašeg odredišta sigurnija je varijanta, ali s vremenom bi mogla biti skuplja. Razvoj umjetne inteligencije omogućio nam je osmišljavanje, koncipiranje i izgradnju vozila koji bi se kretali po cestama bez ljudskog uplitanja.

Timovi koje rade na razvoju i realiziranju autonomnih vozila moraju biti upoznati s mnogim aspektima moderne tehnologije (LiDAR, kamera, GPS, altimetar, žiroskop, itd.). Isto tako u timu moraju postojati osobe koje se razumiju u razvoju softvera što uključuje poznavanje programskih jezika poput Python, C++ potom znanje o umjetnoj inteligenciji što uključuje i strojno učenje i neuronske mreže.

Cilj je bio ostvariti autonomnu vožnju automobila pri brzinama od 15, 20 i 25 mph dok bi se vozilo kretalo desnom trakom. Nakon generiranja podataka krenulo se s analizom i vizualizacijom podataka koja nam je omogućila uvid u generirane podatke i dala osnovne spoznaje u kojem smjeru bi trebala teći predobrada podataka. Bez dobro obavljene predobrade podataka niti jedan model ma kako dobar bio neće funkcionirati. Nakon obavljene predobrade započeta je izgradnja konvolucijske neuronske mreže. Kako to biva u praksi teško je išta iz prve točno procijeniti pa tako i kada su u pitanju vrijednosti hiperparametara za izgrađeni model mreže. Ali nakon nekoliko kratkotrajnih treniranja modela mreže s različitim vrijednostima hiperparametara došlo se do optimalnih vrijednosti nad kojima se kasnije provelo puno treniranje. Iskorištavanjem PID kontrolera za postizanje zadanih brzina i nakon istrenirane konvolucijske neuronske mreže ostvario se zadani cilj to jest automobil se na džungla stazi Udacity simulatora samostalno može kretati desnom trakom pri brzinama od 15, 20 i 25 mph.

LITERATURA

- [1] National Highway Traffic Safety Administration: „Preliminary Statement of Policy Concerning Automated Vehicles“
- [2] Hrvoje Dujmić: „Multimedijijski sustavi“, 2012.
- [3] Slika RGB prostor boja:
<https://www.dynamsoft.com/blog/wp-content/uploads/2019/05/unnamed.jpg>, 27. kolovoza 2019.
- [4] Dinko Begušić: „Digitalna obrada signala“, 2017.
- [5] MJPEG: https://en.wikipedia.org/wiki/Motion_JPEG, 24. svibnja 2019.
- [6] Definicija inteligencije: Mainstream Science on Intelligence, 13. prosinca 1994.
- [7] Stuart Russell, Peter Norvig: „Artificial Intelligence A Modern Approach“, 3rd Edition 2010.
- [8] Problematika i primjena umjetne inteligencije:
https://en.wikipedia.org/wiki/Artificial_intelligence#Problems, 4. lipnja 2019.
- [9] Aurélien Géron: „Hands-On Machine Learning with Scikit-Learn & TensorFlow“, 2017.
- [10] Slika biološke živčane stanice:
https://static.packt-cdn.com/products/9781787121393/graphics/B06139_05_01.jpg, 27. kolovoza 2019.
- [11] Daniel Graupe: „Principles of Artificial Neural Networks“, 2nd Edition 2007.
- [12] Slika biološki neuron naspram umjetnim neuronom:
https://miro.medium.com/max/610/1*SJPacPhP4KDEB1AdhOFy_Q.png, 27. kolovoza 2019.
- [13] Aktivacijska funkcija:
<https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/>, 25. lipnja 2019.
- [14] Funkcija cijene: https://isaacchanghau.github.io/post/loss_functions/, 1. srpnja 2019.
- [15] Slika površina gubitaka:
https://cdn-images-1.medium.com/max/1000/1*UWLKK82On-GkwkfTBL-zTw.png, 27. kolovoza 2019.
- [16] Slika smjer prosljeđivanja informacije:

[https://www.datasciencecentral.com/profiles/blogs/a-simplified-explanation-for-understanding-the-mathematics-of?xg_source=activity](https://www.datasciencecentral.com/profiles/blogs/a-simplified-explanation-for-understanding-the-mathematics-of-xg_source=activity), 27. kolovoza 2019.

[17] Slika kompromis odstupanja i varijance:

https://miro.medium.com/max/544/1*Y-yJiR0FzMgchPA-Fm5c1Q.jpeg 27. kolovoza 2019.

[18] Slika mogući ishodi treniranja:

<https://blog.clearbrain.com/posts/4-reasons-your-machine-learning-model-is-wrong-and-how-to-fix-it>, 28. kolovoza 2019.

[19] Slika ispuštanja neurona:

https://www.researchgate.net/figure/Dropout-neural-network-model-a-is-a-standard-neural-network-b-is-the-same-network_fig3_309206911, 28. kolovoza 2019.

[20] Slika proširenje podataka:

<https://www.researchgate.net/publication/319413978/figure/fig2/AS:533727585333249@1504261980375/Data-augmentation-using-semantic-preserving-transformation-for-SBIR.png>, 27. kolovoza 2019.

[21] Slika više filtera konvolucije:

<https://indoml.files.wordpress.com/2018/03/convolution-with-multiple-filters2.png?w=979>, 27. kolovoza 2019

[22] Slika maksimalnog i prosječnog izvlačenja:

<https://qph.fs.quoracdn.net/main-qimg-cf2833a40f946faf04163bc28517959c>, 28. kolovoza 2019.

[23] Slika poravnavanja:

https://sds-platform-private.s3-us-east-2.amazonaws.com/uploads/73_blog_image_1.png, 28. kolovoza 2019.

[24] Slika arhitektura konvolucijske neuronske mreže:

https://miro.medium.com/max/1200/1*XbuW8WuRrAY5pC4t-9DZAQ.jpeg, 27. kolovoza 2019.

[25] Keras biblioteka: <https://en.wikipedia.org/wiki/Keras>, 12. kolovoza 2019.

[26] Pandas biblioteka: <https://pandas.pydata.org/>, 12 kolovoza 2019.

[27] NumPy biblioteka: <https://www.numpy.org/>, 12 kolovoza 2019.

[28] Matplotlib biblioteka: <https://matplotlib.org/>, 12 kolovoza 2019.

[29] TensorBoard biblioteka:

https://www.tensorflow.org/guide/summaries_and_tensorboard, 12. kolovoza 2019.

[30] Slika PID regulator:

<https://i0.wp.com/microcontrollerslab.com/wp-content/uploads/2019/02/2-PID-controller-Arduino.png?ssl=1>, 27. kolovoza 2019.

[31] PID regulator: <https://projects.raspberrypi.org/en/projects/robotPID>, 23. kolovoza 2019.

POPIS SLIKA

Slika 2.1. RGB prostor boja [3]	5
Slika 2.2. Analogni signal (lijevo) i kvantizirani signal (desno)	6
Slika 3.1. Pod polja umjetne inteligencije	15
Slika 3.2. Metodologija poboljšanog učenja.....	19
Slika 3.3. Izvan mrežno učenje	20
Slika 3.4. Učenje na mreži	21
Slika 3.5. Učenje temeljeno na primjeru (lijevo) i učenje temeljeno na modelu (desno).....	22
Slika 4.1. Biološka živčana stanica (neuron) [10]	25
Slika 4.2. Arhitekture umjetnih neuronskih mreža koje obavljaju osnovne logičke operacije	26
Slika 4.3. Biološki neuron naspram umjetnim neuronom (perceptron) [12]	27
Slika 4.4. Struktura perceptronu.....	28
Slika 4.5. Duboka neuronska mreža	30
Slika 4.6. Primjer nekolicine aktivacijskih funkcija i njihove derivacije	34
Slika 4.7. Dijagram tijeka treniranja neuronske mreže.....	35
Slika 4.8. Površina gubitaka za neuronsku mrežu ResNet-56 bez i s preskakanjem veza [15]	
.....	38
Slika 4.9. Smjer prosljeđivanja unaprijed (predviđanje) i unatrag (ažuriranje težina) [16]	39
Slika 4.10. Podjela skupa podataka i njihova namjena	40
Slika 4.11. Kompromis odstupanja i varijance [17]	41
Slika 4.12. Grafički prikaz podtreniranja, optimalnog treniranja i pretreniranja [18]	42
Slika 4.13. Standardna neuronska mreža i mreža nakon primjene ispuštanja neurona [19]....	43
Slika 4.14. Rano zaustavljanje s obzirom na vrijednosti funkcija cijena.....	45
Slika 4.15. Primjer primjene proširenja podataka na instanci slike [20]	46
Slika 4.16. Operacija konvolucije	48
Slika 4.17. Konvolucija s više filtera i s podacima koji sadrže dubinu [21].....	49
Slika 4.18. Primjer maksimalnog i prosječnog ujedinjavanja [22].....	51
Slika 4.19. Primjer poravnavanja [23]	51
Slika 4.20. Arhitektura konvolucijske neuronske mreže [24].....	52
Slika 5.1. Konfiguracijski zaslon simulatora	55
Slika 5.2. Opcije koje nudi simulator.....	56
Slika 5.3. Džungla staza i trening opcija.....	57

Slika 5.4. Dijagram kutova upada za vožnju desnom stranom	59
Slika 5.5. Dijagram kutova upada za vožnju lijevom stranom	60
Slika 5.6. Zrcaljene slike kako bi postigli efekt vožnje u desnoj traci.....	61
Slika 5.7. Vrijednosti kuta zakreta za prolazak kroz krivinu.....	62
Slika 5.8. Prolazak kroz krivinu koristeći se tipkovnicom naspram uzorkom krivine	62
Slika 5.9. Promjena vrijednosti kutova s promjenom veličine „n“	64
Slika 5.10. Distribucija kutova za linearne i eksponencijalne vrijednosti težina.....	65
Slika 5.11. Primjer trave, trake, litice i pločnika na osunčanoj i sjenovitoj strani.....	68
Slika 5.12. Mjera intenziteta u sjeni naspram suncu za pojedine dijelove simulatora.....	68
Slika 5.13. Primjer izvorne slike, slike s dodanom sjenom u RGB formatu i njihovi ekvivalenti u BGR formatu.....	70
Slika 5.14. Originalne slike naspram izrezanim za nizbrdo, pravac i uzbrdo.....	71
Slika 5.15. Prikaz slojeva modela s vrijednostima izlaznog oblika i brojem parametara.....	73
Slika 5.16. Distribucija kutova upada nad kojom je izvršen trening, a dobiven najbolji rezultat	79
Slika 5.17. Cijena funkcije za opisani model i distribuciju podatak sa slike 5.16.....	80
Slika 5.18. Funkcije metrike za odabrani finalni model i odabранe finalne podatke.....	82
Slika 5.19. Usporedba cijene gubitaka MAE naspram MSE	83
Slika 5.20. Blok dijagram PID regulatora [30]	84
Slika 5.21. Jedan okvir izgleda autonomne vožnje.....	87

POPIS OZNAKA I KRATICA

AGI (artificial general intelligence) – generalna umjetna inteligencija

AI (artificial intelligence) – umjetna inteligencija

ANN (artificial neural network) – umjetna neuronska mreža

BGR (blue, green, red) – plava, zelena, crvena

CIE - Commission internationale de l'éclairage

CMYK (cyan, magenta, yellow, black) – cijan, magenta, žuta, crna

CNN (convolution neural network) – konvolucijska neuronska mreža

DNN (deep neural network) – duboka neuronska mreža

FHD (full high definition) – puna visoka definicija

FNN (feedforward neural network) – unaprijedna neuronska mreža

FPS (frames per second) – slika u sekundi

HLAI (human-level AI) – ljudska razina AI

HSV (hue, saturation, value) – nijansa, zasićenje, vrijednost

IQ (intelligence quotient) – kvocijent inteligencije

LTU (linear threshold unit) – linearna jedinica praga

MAE (mean absolute error) – srednja apsolutna pogreška

MJPG (motion JPEG) – pokretni JPEG

MLP (multi-layer perceptron) - višeslojni perceptron

MSE (mean square error) – srednja kvadratna pogreška

PID (proportional–integral–derivative) – proporcionalni integralni derivacijski

ReLU (rectified linear unit) – ispravljena linearna jedinica

RGB (red, green, blue) – crvena, zelena, plava

SNARC (Stochastic Neural Analog Reinforcement Calculator) – stohastički kalkulator neuralnog analognog poboljšanja

tanh (hyperbolic tangent) – hiperbolični tangens

TLU (threshold logic unit) – logička jedinica praga

XOR (exclusive or) – ekskluzivno ili

SAŽETAK

Zamislite da se možete sigurno i povoljno voziti s jednog odredišta na drugo dok pregledavate dokumente, gledate televiziju ili čak spavate po potrebi. Takav koncept je moguć samo ako se na tržište pojavi autonomno vozilo četvrte razine. U ovom diplomskom radu pokušavamo riješiti jedan od ključnih elemenata koje će svaki autonomni automobil trebati da prevlada, a to je izračunavanje kuta upravljanja s obzirom na cestu ispred njega. Uz određivanje kuta upravljanja, automobil mora voziti u jednoj traci što bi u našem slučaju bilo desnom trakom. Uvodni dio rada sadrži osnovne definicije te su opisani potrebni pojmovi i koncepti potrebni za praktičnu izvedbu konvolucijske neuronske mreže koja će procijeniti kut upravljanja na osnovu slike dobivene od kamere postavljene na automobilu. Upotrebom simulatora kojega je napravio Udacity omogućio je većem broju programera da se posvete izradi arhitektura umjetnih mreža kojima će se moći testirati autonomna vožnja bez velikih ulaganja i rizika za sudionike prometa. Na kraju rada se može vidjeti upotrijebljena analiza podatka i predobrada podataka zajedno s relevantnim grafovima. Štoviše, predstavljena je arhitektura konvolucijske neuronske mreže koja uspješno procjenjuje kut upravljanja dok se vozilo kreće desnom trakom.

KLJUČNE RIJEČI

Autonomno vozilo, umjetna inteligencija, strojno učenje, konvolucijska neuronska mreža

TITLE

A prototype of an autonomous vehicle made using the behavioural cloning

SUMMARY

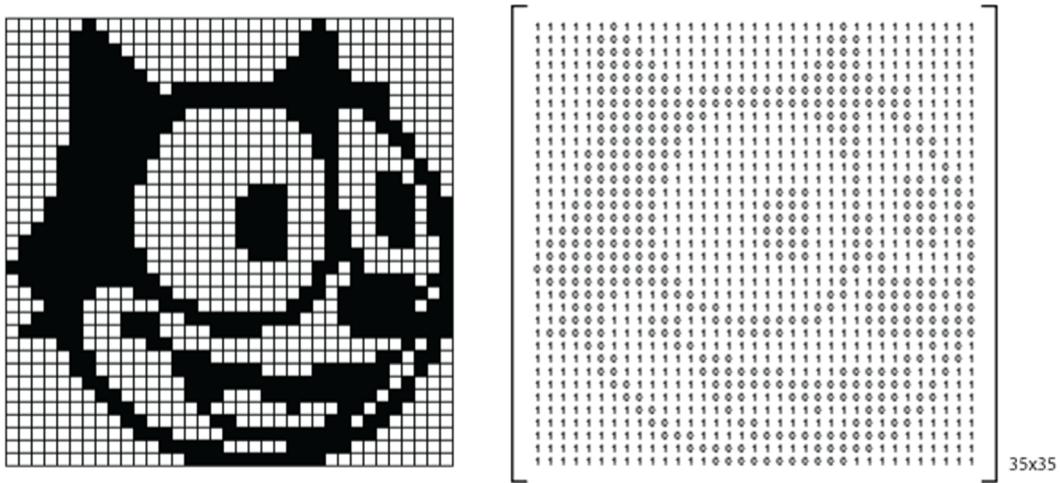
Imagine being able to safely and affordably drive from one destination to another while scrutinising documents, watching TV or even sleeping if it is required. Such a concept is only possible if a fourth level autonomous vehicle appears on the market. In this master thesis, we are trying to solve one of the key elements that every autonomous vehicle will need to overcome, which is to calculate the steering angle given the road ahead. In addition to determining the steering angle, a car must drive in one lane, which in our case would be the right lane. The introductory part of the paper provides basic definitions and describes the basic terms and concepts required for the practical implementation of a convolutional neural network that will estimate the steering angle based on the image received from a camera installed on a car. Utilizing a simulator created by Udacity has enabled more developers to devote themselves to creating artificial neural network architectures that can test autonomous driving without major investment and risk to traffic participants. At the end of the paper, we can see used data analysis and data pre-processing together with relevant graphs. Moreover, it is represented the architecture of a convolutional neural network which successfully estimating the steering angle while the vehicle is driving in the right lane.

KEYWORDS

Autonomous vehicle, artificial intelligence, machine learning, convolution neural network

Dodatak A

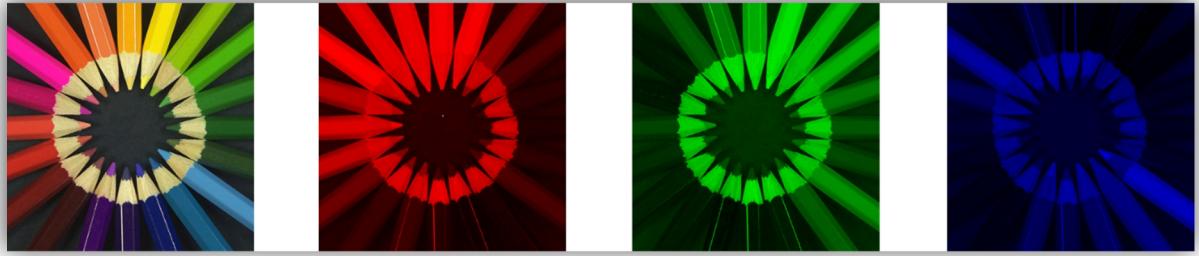
Način na koji ljudi vide slike se ne poklapa s načinom na koji digitalna računala vide. I dok ljudi vide i prepoznaju ili ne prepoznaju stvari, objekte, osobe i tako dalje na slici digitalna računala ništa od navedenog ne mogu vidjeti. Slike u rasterskoj grafici su dvodimenzionalne matrice pri čemu broj piksela po širini označava koliko slika ima stupaca dok broj piksela po visini označava koliko ta slika ima redaka. Ovisno o tipu slike (crno-bijela, sivih tonova, RGB, CMYK, itd.) će ovisiti veličina treće dimenzije i što predstavlja svaki kanal. Tako imamo priliku vidjeti na slici A.1. kako bi čovjek video crno bijelu sliku naspram kako je digitalno računalo vidi.



Slika A.1. Primjer crno bijele slike kada je čovjek vidi naspram kada je računalo vidi

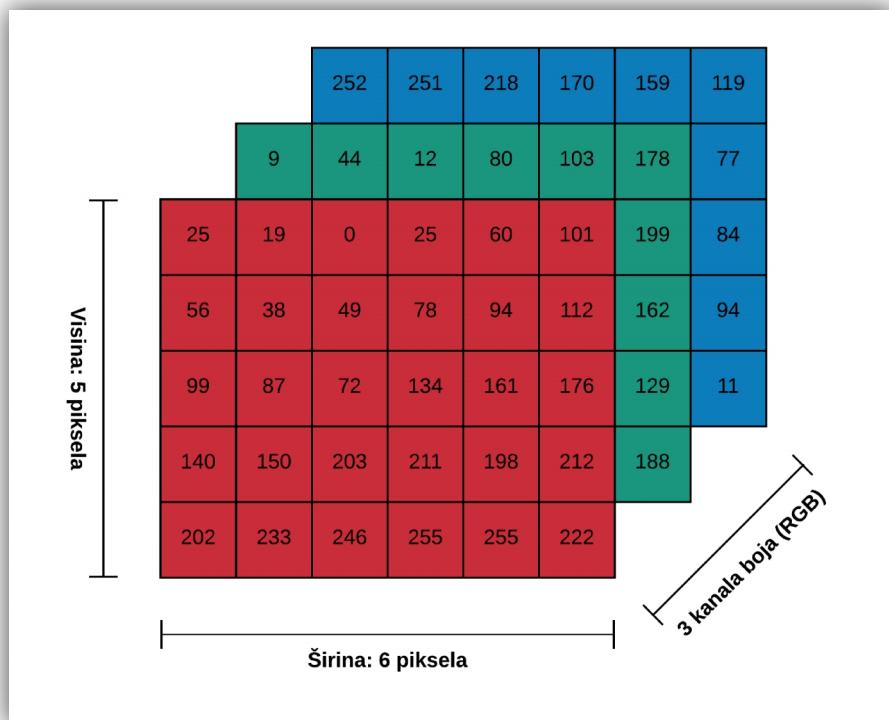
Standard za crno-bijele slike je da se bijela boja označi jedinicom, a crna boja nulom kao na primjeru slike A.1. Za ovakav tip slike nije potrebna treća dimenzija kao ni za slike sa sivim tonovima. Slike sa sivim tonovima imaju raspon vrijednosti od nula (0) do dvjesto pedeset i pet (255) pri čemu nula označava minimalni intenzitet (crna), a dvjesto pedeset pet maksimalni intenzitet (bijela). Na ovaj način se postiže ukupno $2^8 = 256$ drukčijih nivoa sive.

S druge strane za boje u slici potrebne su tri matice. Svaka matrica određuje količinu crvene, zelene i plave boje koja sačinjava sliku (slika A.2.).



Slika A.2. Originalna slika u boji s pripadajućim komponentama crvene, zelene i plave boje

Ovaj sustav je još poznat kao i RGB sustav i danas je veoma zastupljen. Osim njega imamo još CMYK sustav koji se upotrijebjava za printanje Y'IQ za TV analogni prijenos u NTSC itd. Elementi ovih matrica su cijeli brojevi između nula i dvjesto pedeset pet i oni određuju intenzitet svakog piksela s obzirom na boju matrice. Tako je u RGB sustavu moguće prikazati $256^3 = 2^2 = 16,777,216$ drukčijih boja (slika A.3.).



Slika A.3. Primjer kako računalo vidi RGB sliku u boji

Dodatak B

Filtriranje slika omogućuje primjenu različitih efekata na fotografije. Filtriranje slika je da imamo 2D matricu filtra i 2D sliku. Nova vrijednost piksela se dobije tako što se primjeni filter nad tim pikselom i njegovim susjedima. Ovaj se postupak ponavlja nad svim pikselima slike. Primjena filtera znači dobiti sumu umnožaka matrice filtera i dijela slike nad kojim se primjenjuje filter. Ovo možemo gledati i iz druge perspektive pa sliku i filter promatrati kao dvije funkcije. Tada primjena filtera nad slikom bi bila konvolucija ili korelacija. Razlika između konvolucije i korelacije je u tome što se kod konvolucije mora matrica filtra zrcaliti, ali većina filtera je simetrična tako da nema razlike. Filtri s konvolucijom relativno su jednostavnji. Postoje i složeni filtri koji mogu učiniti jako složene stvari, ali ovdje će se samo prikazati nekoliko osnovnih i jednostavnih filtera.

Postoji nekoliko pravila oko filtera, a to su:

- Veličina mora biti neparan broj kako bi imala centralni element (npr. 3x3, 5x5, 7x7, itd.)
- Zbroj svih elemenata filtra treba biti 1 ako želite da rezultirajuća slika ima istu svjetlinu kao i izvorna.
- Ako je zbroj elemenata veći od 1, rezultat će biti svjetlijia slika, a ako je manji od 1, tamnija slika.
- Ako je suma 0, rezultirajuća slika nije nužno potpuno crna, ali će biti vrlo mračna.

Slike su konačnih dimenzija te kada se filtriranje vrši nad rubnim pikselima filteri najednom ostaju bez vrijednosti za izvršavanje konvolucije. Jedna od metoda je je dodavanje potrebnog broja obruba (ovisi o veličini filtera) oko slike te u takvoj situaciji se dodanim obrubima dodjeljuje vrijednost nula. Druga opcija je „saviti“ filter do na drugu stranu slike. Nakon primjene filtera vrijednost može biti manja od nula ili veća od dvjesto pedeset i pet. U takvim se situacijama najčešće onda ta vrijednost zamjenjuje s minimalnom (0) odnosno maksimalnom (255) vrijednošću koja se može dodijeliti jednom pikselu. Na slici B.1. možemo vidjeti neke od osnovnih filtera s pripadajućim matričnim oblikom i kako filtriraju sliku.

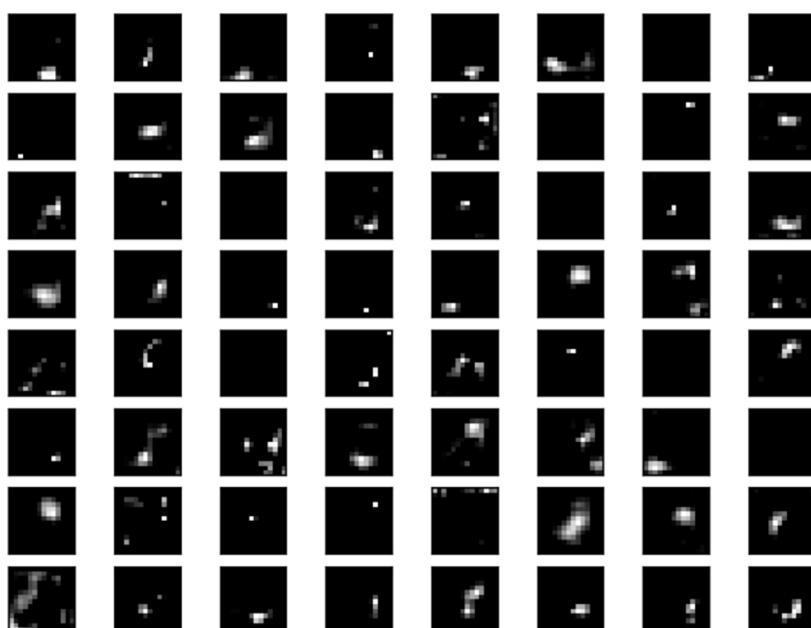
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Slika B.1. Filteri i njihova primjena na sliku

Na slici B.2. možemo vidjeti kako izgleda mape značajki za prvi konvolucijski sloj modela VGG16, dok na slici B.3. možemo vidjeti kako mape značajki izgledaju u petom bloku. Mape značajki se dobivaju tako što se primjenjuju različiti filteri na ulaznu matricu (sliku).



Slika B.2. Mapa značajki za prvi konvolucijski sloj VGG16 modela



Slika B.3. Mapa značajki za peti blok VGG16 modela