SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2080

SOFTVERSKI SUSTAV ZA AUTOMATSKO OTVARANJE USISNOG OTVORA NA SPECIJALIZIRANIM VOZILIMA

Dino Perić

Zagreb, veljača 2020

SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2080

SOFTVERSKI SUSTAV ZA AUTOMATSKO OTVARANJE USISNOG OTVORA NA SPECIJALIZIRANIM VOZILIMA

Dino Perić

Zagreb, veljača 2020

Sadržaj

1. UVOD

U današnje vrijeme bilježi se sve veći napredak i razvoj tehnologije koja se primjenjuje u raznim aspektima ljudskog života – od medicine, kućanstva, do primjerice prijevoza.

Isto vrijedi i za današnja vozila koja postaju sve opremljenija naprednim tehnološkim rješenjima i funkcionalnostima. Naravno, preduvjet za takav napredak je razvoj hardverske i softverske opreme.

I dok su u prošlosti ljudi upravljali vozilima isključivo mehaničkim polugama ili pedalama, a naprednijim upravljanjem smatralo se korištenje korisničkog sučelja kao posrednika između ljudske naredbe i samog vozila, danas je razvoj tehnologije omogućio dodavanje sustava koji mogu sami upravljati vozilima i obavljati složenije funkcionalnosti.

Jedna od prekretnica u automatizaciji sustava koji su dotad bili isključivo pod direktnim ljudskim upravljanjem je razvoj umjetne inteligencije. Do razvoja umjetne inteligencije bilo je teško ili nemoguće postići takvu automatizaciju jer su zadaci bili previše složeni za softver. No, strojno učenje je omogućilo modeliranje ljudskog ponašanja u nekim specifičnim zadacima.

Pogotovo je značajan razvoj neuronskih mreža za analizu vizualnih podataka jer ljudi za rješavanje mnogih problema koriste svoj vid. Duboke neuronske konvolucijske mreže su se pokazale kao vodeće u učenju modela strojnog učenja iz vizualnih podataka. Algoritmi strojnog učenja zahtijevaju ogromne količine podataka da bi mogle iz njih naučiti kvalitetne modele koji mogu zamijeniti ljudskog operatora u specifičnom zadatku. Zahtjev za količinom podataka je riješen postavljanjem mnogih dodatnih senzora na vozila. Senzori omogućuju vozilima da percipiraju svoj okoliš i onda korištenjem umjetne inteligencije reagiraju na te podražaje bez ljudske intervencije. Druga svrha senzora je da služe kao generatori kontinuiranih podataka potrebnih za učenje modela. Korištenjem strojnog učenja moguće je analizirati i iskoristiti generirane podatke kako bi softver mogao izvršavati pametne operacije te osloboditi ljudskog operatora da se bavi sa složenijim zadacima.

Vozilo za koje će testirati princip automatizacije je električna čistilica firme *Rasco* prikazana na slici 1.

Slika 1 *Rasco Lynx* električna čistilica

Čistilica pored četki ima usisnu pumpu za smeće koju ljudski operator mora ručno otvarati i zatvarati ovisno o tome što se nalazi ispod nje. Glavni senzor u ovom slučaju je kamera koja se nalazi iznad četki. Kamere generiraju slijedne slikovne podatke iz kojih se mogu izvući mnoge korisne informacije. Svrha ovog rada je simulacijski testirati koncepte koji će, korištenjem strojnog učenja, automatski temeljeno na videu s kamere otvarati i zatvarati usisni otvor.

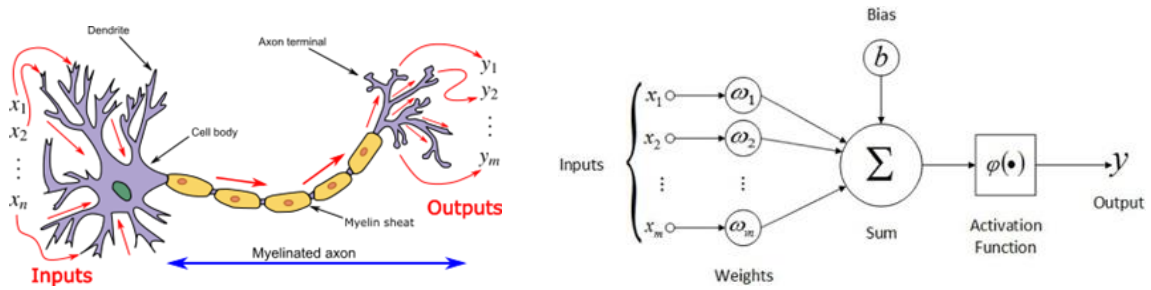
Zbog nemogućnosti pristupa snimkama s kamere same čistilice koristit će se zamjenski skup podataka te će na njemu biti razvijen model koji će imati arhitekturu koja se može primijeniti na stvarne slike s čistilice. Dodatno, zbog nemogućnosti fizičkog pristupa samoj čistilici i instalacije softvera na nju model će biti razvijen i postavljen na Android uređaj. Umjesto kamere od čistilice bit će iskorištena kamera mobilnog uređaja, a komunikacija između aplikacije i čistilice bit će odrađena preko CAN Bus protokola. Umjesto spajanja na CAN Bus čistilice koristit će se CAN Wireless Gateway uređaj kao simulacijsko okružje.

2. DUBOKE NEURONSKE MREŽE

2.1 Općenito

Ljudi i životinje obrađuju podatke neuronskim mrežama koje se sastoje od milijardi neuronskih stanica (neurona) koje su povezane putem sinapsi kojima razmjenjuju električne signale (aktivacije). Računalni algoritmi koji imitiraju ovakve biološke strukture nazivaju se umjetne neuronske mreže. Cilj ovakvih algoritama izvući je korisne značajke iz skupa podataka te na temelju tih značajki proizvesti željeni izlaz**.**

Osnovna gradivna jedinica neuronskih mreža je neuron. Iz biološke perspektive, neuron prima signale na svojim ulaznim sinapsama, obavlja neku transformaciju tih ulaznih podataka i stvara neki izlaz. Matematički, neuron se često modelira kao linearna kombinacija $\sum_i w_i * x_i$ ulaznih signala x_i na koju je dodan neki pomak b te se onda taj zbroj „provlači“ kroz neku nelinearnu aktivacijsku funkciju. Analogija između biološkog i umjetnog neurona prikazana je na slici 2.



Slika 2 Biološki neuron (lijevo) (Chauhan, 2019) i umjetni neuron(desno) (Nielsen, 2019)

Najjednostavnija vrsta umjetnog neurona zove se perceptron. Perceptroni se u praksi više ne koriste, ali su pogodni za početno razumijevanje načina funkcioniranja neuronskih mreža. Slika 2 (desno) prikazuje perceptron s n ulaza (x_1, x_2,..., x_n). Veličina ulaza u neuron naziva se broj značajki. Svaki perceptron u sebi sadrži težine (eng. *weights*) koje određuju važnost određenog ulaza u odnosu na izlaz. Težine su realni brojevi kojima se množi vrijednost ulaza (Nielsen, 2019). Izlaz neurona se računa po sljedećem izrazu:

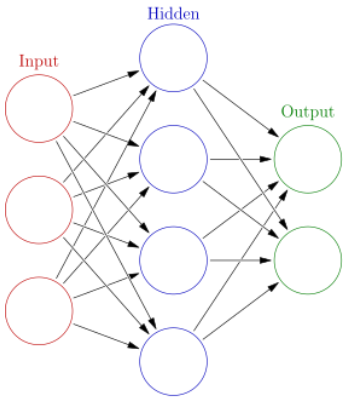
$$izlaz = \begin{cases} 0\ ako\ \sum_j w_j x_j \leq prag \\ 1\ ako\ \sum_j w_j x_j > prag \end{cases} \qquad (1)$$

w_j je težina kojom se množi x_j ulaz, a prag je neka vrijednost koju rezultat zbrajanja umnožaka težina s ulazima mora zadovoljiti. Mijenjanjem težina i praga, neuroni se različito ponašaju. Aktivirani biološki neuron bio bi ekvivalentan izlazu 1, a neaktivirani izlazu 0.

Pojednostavljeni zapis perceptrona ostvaruje se dvjema izmjenama. Prva izmjena je zamjena zbroja umnožaka težina ulazima u skalarni produkt između njih, a druga je prebacivanje praga na lijevu stranu nejednakosti. Pristranost (eng. *bias*) postaje negativni prag te se perceptron može zapisati na sljedeći način:

$$\text{izlaz} = \begin{cases} 0 \text{ ako } w_j x_j + b \leq 0 \\ 1 \text{ ako } w_j x_j + b > 0 \end{cases} \quad (2)$$

Pomak b postaje mjera toga koliko je „lako“ perceptronu dobiti izlaz 1 (Nielsen, 2019). Aktivacijska funkcija neurona je funkcija kojoj se šalje izračunato unutarnje stanje neurona te ona primjenjuje neku transformaciju na to stanje koje onda postaje izlaz neurona. U slučaju perceptrona funkcija *izlaz* je aktivacijska funkcija koja je u ovom slučaju linearna. Kombiniranjem više perceptrona (neurona) stvara se jednostavna neuronska mreža.

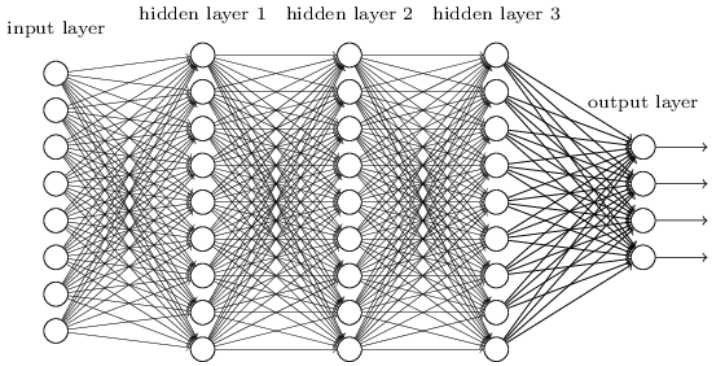


Slika 3 Neuronska mreža (Chauhan, 2019)

Na slici 3 prikazana je jednostavna unaprijedna mreža ili višeslojni perceptron. Prikazana se mreža sastoji od 3 sloja veličine 3, 4, i 1. Prvi se sloj naziva ulazni, zadnji sloj izlazni, a svaki sloj između naziva se skriveni. U ulazni sloj dolaze podaci koje zatim skriveni sloj transformira te šalje u izlazni sloj koji čini izlaz modela. Perceptroni se u ovom slučaju nalaze samo u središnjem sloju. Svaki neuron zapravo je funkcija koja preslikava vektor u skalar, odnosno prima sve vrijednosti iz prijašnjeg sloja (vektor) te ih množi s težinama i zbraja u jednu vrijednost (skalar). Broj skrivenih slojeva u neuronskim mrežama je proizvoljan i povećavanjem veličine i broja skrivenih slojeva povećava se kompleksnost mreže.

Duboke unaprijedne mreže najčešći su modeli u području dubokog učenja. Naziv „unaprijedne“ dolazi od činjenice da se informacije uvijek kreću prema sljedećem sloju i u mreži ne postoje nikakve povratne veze. Mreže se nazivaju dubokima jer se sastoje od više

skrivenih slojeva neurona, a broj tih slojeva određuje dubinu mreže (Ian Goodfellow, 2016). Mreža prikazana na slici 3 naziva se plitkom, a slika 4 prikazuje duboku mrežu.



Slika 4 Duboka neuronska mreža (Nielsen, 2019)

Duboke mreže imaju puno veću sposobnost učenja od plitkih jer im njihova slojevitost omogućuje epistemsku redukciju ulaznih podataka (Anderson, 2018). Epistemska redukcija je redukcija podataka na takav način da se nebitni dijelovi podataka odbacuju, a zadržavaju se samo oni koji će, u ovom slučaju, najviše utjecati na izlaz neurona, odnosno pamte se najvažnije značajke u podacima. Jednoslojne mreže ovo ne mogu napraviti jer njihov jedan skriveni sloj samo jednom transformira podatke i odmah ih šalje u izlazni sloj.

Još jedna prednost dubokih mreža je to što s manjim brojem parametara mogu postići bolje rezultate od plitkih mreža s većim bojem parametara (Mhaskar H. Liao Q., 2017). Cilj dubokih unaprijednih mreža je aproksimirati neku funkciju $f^*(x)$. Primjerice klasični klasifikator $y = f^*(x)$ pridružuje kategoriju y nekom ulazu $\boldsymbol{x}$, a unaprijedna mreža bi definirala pridruživanje $\acute{y} = f(\boldsymbol{x}, \boldsymbol{\theta})$ i učenjem naučila vrijednosti parametara $\boldsymbol{\theta}$ koji bi napravili najbolju aproksimaciju funkcije y. Funkcija y prikazuje stvarni odnos između ulaza x i izlaza y, a $\acute{y}$ je samo njena aproksimacija (Ian Goodfellow, 2016). Mreža se može predstaviti kompozicijom jednostavnih funkcija gdje svaka funkcija predstavlja jedan sloj: $f(x) = o(f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x}))))$. Model odgovara usmjerenom acikličkom grafu koji prikazuje kako su funkcije povezane. U ovom slučaju $f^{(1)}$ predstavlja prvi sloj, $f^{(2)}$ drugi i tako dalje. Broj ovakvih funkcija je dubina modela. Zadnji sloj unaprijednih mreža naziva se izlaz. Model se može prikazati i preko pomoćnih varijabli $\boldsymbol{h}_L, \boldsymbol{h}_{L-1}, \cdots \boldsymbol{h}_1$ kao:

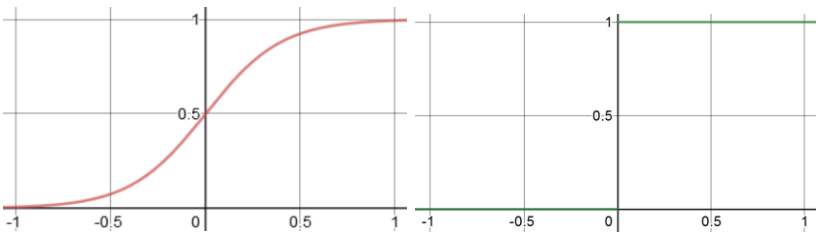
$$\begin{aligned} h_1 &= f_1(x,\, \theta_1) \\ &\vdots \\ h_{L-1} &= f_{L-1}(h_{L-2},\, \theta_{L-1}) \\ h_L &= f_L(h_{L-1},\, \theta_L) \\ f(x,\theta) &= o(h_L) \end{aligned} \qquad (3)$$

Gdje $\boldsymbol{\theta_i}$ predstavlja parametre i-tog sloja, $\boldsymbol{h_i}$ značajke u skrivenim slojevima, a $\boldsymbol{o}$ predstavlja izlazni sloj (Ian Goodfellow, 2016). Željeno ponašanje mreže je da mala promjena ulaza x uzrokuje malu promjenu izlaza y. Kad bi mreža imala ovo svojstvo, njeno učenje bi bilo omogućeno jer bi se težine i pomak mogli relativno jednostavno promijeniti da se dobiju željeni izlazi. Perceptroni nemaju ovo svojstvo jer mala promjena u ulazu može skroz promijeniti izlaz iz 0 u 1 ili obrnuto, što dosta otežava učenje mreže. Zbog ovakvog ponašanja perceptrona oni se ne koriste u svrhe dubokog učenja (Nielsen, 2019).

Jedno od rješenja problema nalazi se u novom tipu neurona – sigmoidalni neuron. Sigmoidalni neuroni vrlo su slični perceptronu, ali imaju dodatnu aktivacijsku funkciju. Korištenjem aktivacijske funkcije izlaz neurona postaje bilo koji broj između 0 i 1 (ne više samo 0 i 1 kao kod perceptrona). Aktivacijska funkcija kod sigmoidnog neurona je sigmoida definirana kao:

$$\sigma(z) = \frac{1}{1+ e^{-z}} \qquad (4)$$

Gdje je $z = w_j x_j + b$ vektorski produkt kao i kod perceptrona. Sigmoida se ponaša tako da umnožak težina s ulazima pridružuje rasponu <0, 1>. Ako je umnožak jako velik, onda cijeli izraz teži prema 1, a jako negativan izlaz teži prema 0. Slika 5 prikazuje izlaz sigmoidalne funkcije (lijevo) i izlaz perceptrona (desno) u koordinatnom sustavu. Sigmoidalni neuron zapravo je izglađen perceptron, a tim izglađivanjem je postignuto svojstvo da male promjene u ulazu uzrokuju male promjene u izlazu. Kod učenja mreže ovakav neuron nailazi na mnogobrojne druge probleme pa se u modernim mrežama koriste druge aktivacijske funkcije poput zglobnice (RelU, eng. *rectified linear unit*). Aktivacijske funkcije i njihov utjecaj na učenje mreže bit će objašnjene u kasnijim poglavljima.



Slika 5 Sigmoida (lijevo), Stepenasta funkcija (desno)

2.2 Učenje mreže

2.2.1 Općenito

U strojnom učenju postoje dva glavna načina za učenje modela: nadzirani i nenadzirani.

Kod nadziranog učenja model se uči korištenjem podataka koji su označeni (eng. *labeled*). Nadzirano učenje se može usporediti s učenjem koje se događa u stvarnosti kad je u prisutnosti učenika učitelj koji mu govori koji su točni odgovori. Nadzirani model uči iz označenih podatka, ali njegov cilj je stvarati dobre rezultate za podatke koje nikad nije vidio.

Nenadzirani način učenja je izvan okvira ovog diplomskog rada te će se svako daljnje spominjanje učenja odnositi na nadzirano.

Generalizacijska pogreška odnosi se na sposobnost modela da radi dobra predviđanja na podacima koje nije prije vidio. Cilj učenja mreža je napraviti model koji dobro generalizira podatke. Kod učenja modela podaci se često razdvajaju u tri skupa:

1) Prvi skup je skup za učenje u kojemu se nalazi svi podaci koje će model pogledati i iskoristiti za učenje.
2) Drugi skup je validacijski skup koji se koristi u određenim točkama učenja modela da bi pokazao kako se model ponaša na podacima koje nije vidio te da pruži objektivan uvid u generalizacijsku sposobnost modela. Mnoge mreže koriste validacijski skup da bi usmjerili učenje mreže u određenim smjerovima.
3) Zadnji skup podataka je testni. Koristi se nakon učenja i validiranja modela da bi se izmjerile njegove performanse. Testni skup, kao i validacijski, sadrži primjere koje model nikad nije vidio.

Razlog zbog kojega je razdvajanje podataka potrebna praksa je taj što se ne može realno analizirati generalizacijska sposobnost modela ako ju analiziramo na podacima na kojima je model učen. Veličina podatka se u strojnom učenju naziva broj značajki (eng. *features*). Primjerice, broj značajki slike u boji bi bio visina x širina x broj kanala gdje su širina i visina rezolucija slike, a broj kanala se odnosi na način na koji se slike spremaju u računalo, a to je najčešće RGB ili *grayscale*.

Glavne vrste modela kod nadziranog učenja su klasifikacijski i regresijski. Razlika između te dvije vrste je što klasifikacijski model ima kategorijski izlaz, a regresijski numerički.

Najčešće klasifikacijski model vraća vjerojatnosti pripadanja određenoj klasi, a regresijski neki relevantni broj. U kontekstu ovog rada sve spomenute mreže biti će klasifikacijske.

Za nadzirano učenje, označen podatak podrazumijeva da se za njega zna točnu klasu kojoj pripada. Primjerice, ako je cilj naučiti mrežu da određuje koja se znamenka nalazi na slici, mreži je potrebna slika (x) i njena stvarna vrijednost $y(x)$. Točne se oznake podataka mreži često daju u „*one hot encoded*" (OHE) vektor obliku. Kad bi u prijašnjem primjeru (sa znamenkama) imali sliku trojke onda bi njena oznaka u OHE obliku bila $y = [0,0,1,0,0,0,0,0,0,0]^{\boldsymbol{T}}$. Vektor je duljine broja klasa, a jedinica se nalazi na tom mjestu u vektoru gdje primjer pripada. Izlaz modela bio bi u istom obliku samo bi umjesto isključivo nula i jedinica izlaz imao vjerojatnosti pripadanja svakoj klasi. Takvim načinom razmišljanja može se reći da podatak sa svojom točnom oznakom ima 100% šanse pripadanja svojoj klasi, a 0% šanse pripadanja svim ostalima. Svaki primjer u skupu za učenje mora biti označen na ovaj način.

Cilj učenja mreže, u matematičkom smislu, je pronaći težine (w) i pomake (b) za sve neurone takve da izlaz mreže aproksimira željenu funkciju $y = f^*(\boldsymbol{x})$ na skupu podataka za učenje (x). Izlaz iz mreže za primjer x označava se s $\acute{y}(x)$. Potrebno je definirati funkciju koja će dati metriku koliko dobro se aproksimira željena funkcija. U neuronskim mrežama ta funkcija zove se funkcija cilja (eng. *cost function*) ili funkcija gubitka (eng. *loss*).

$$C(y, \acute{y}) = \frac{1}{n}\sum_{i=1}^{n} 1(\acute{y}_i \neq y_i) \qquad (5)$$

Ovakva funkcija gubitka zove se 0-1 gubitak gdje je n broj primjera, $\acute{y}_i$ točna oznaka primjera i, a y_i izlaz modela za primjer i. 0-1 gubitak je zapravo broj primjera koji su krivo klasificirani na skupu za učenje (Ian Goodfellow, 2016). Ovako se zapisan gubitak može promatrati kao „nepreciznost" modela odnosno broj netočno klasificiranih primjera podijeljen s ukupnim brojem primjera. Izraz pod sumom dodaje +1 sumi ako se izlaz modela za primjer x razlikuje od stvarne oznake za taj primjer. Model postiže najveći gubitak ako krivo klasificira sve primjere. U tom slučaju suma će biti jednaka 1, a s tim će i cijeli gubitak biti jednak 1. Sukladno ovome model postiže najmanji gubitak ako za sve primjere napravi dobru klasifikaciju. Gubitak je tada 0 jer se u sumu nije dodalo ništa zbog toga što je $\acute{y}_i$ bio jednak y_i za svaki primjer.

Ovaj gubitak je vrlo jednostavno izračunati, ali ima mnoga svojstva koja su nepogodna za učenje mreže. Prvenstveno 0-1 gubitak nije glatka funkcija koja ovisi o parametrima (w, b).

Ova činjenica znači da male promjene u težinama i pomacima vrlo vjerojatno neće promijeniti broj dobro klasificiranih primjera što jako otežava pronalaženje dobrih težina i pomaka te čini funkciju teškom za optimizirati. Zamjenom 0-1 gubitka sa surogatnom funkcijom koja je glatka u ovisnosti o parametrima (w, b) omogućuje se lakše optimiziranje jer čak i ako broj dobro klasificiranih primjera ostane isti može se vidjeti je li se gubitak smanjio ili povećao. U ovom slučaju 0-1 gubitak je zamijenjen s funkcijom $C(w, b)$ koja se naziva kvadratna funkcija cilja (eng. *mean squared error*).

$$C(w, b) = \frac{1}{2n} \sum_{x} \|y(x) - \acute{y}(x)\|^2 \qquad (6)$$

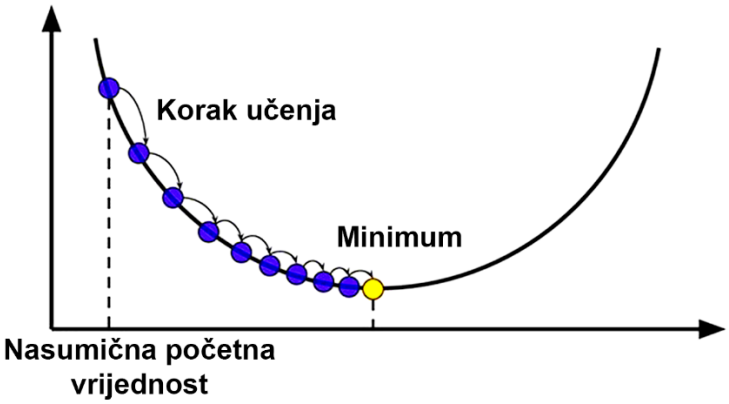
U formuli w predstavlja sve težine u mreži, b sve pomake, n je ukupni broj primjera za učenje, $\acute{y}(x)$ je vektor izlaza iz mreže kad je x ulaz u mrežu, a suma ide preko svih primjera za učenje x. Rezultat funkcije ovisi o primjerima x, ali on nije stavljen kao argument funkcije jer njega nije potrebno učiti. Iz formule se vidi da je funkcija strogo ne negativna zbog toga što će svi izrazi u sumi biti ne negativni. Gubitak postaje malen $C(w, b) \approx 0$ točno kad $\acute{y}(x)$ dobro aproksimira $y(x)$ jer izraz pod operatorom $\| \ \|$ postaje blizak nuli odnosno kad algoritam pronađe dobre vrijednosti za w i b.

U suprotnom slučaju $C(w, b)$ je velik kad algoritam daje izlaze koji su drugačiji od točnog za velik broj primjera za učenje. Cilj učenja mreže postaje minimizirati gubitak $C(w, b)$ odnosno pronaći vrijednosti za w i b koje daju minimalnu vrijednost gubitka. Ovdje je bitno razumjeti da se kod učenja minimizira grešku na primjerima za učenje, ali cilj strojnog učenja je minimizirati pogrešku na primjerima koje mreža nikad nije vidjela (greška generalizacije). Optimiziranje funkcije gubitka radi se algoritmom zvanim gradijentni spust.

2.2.2 Gradijentni spust

Gradijent je vektor koji je tangenta funkcije u nekoj točki i koji ima smjer najvećeg rasta te funkcije. U matematici se gradijent definira kao parcijalna derivacija funkcije po svim njenim varijablama (Ian Goodfellow, 2016). Primjerice ako postoji funkcija $C(w, b)$ ovisna o parametrima (w, b) njen gradijent bi bio definiran kao $\nabla C(w, b) = \left(\frac{\partial C}{\partial w}, \frac{\partial C}{\partial b}\right)$ gdje je $\frac{\partial C}{\partial w}$ parcijalna derivacija od C po w, a $\frac{\partial C}{\partial b}$ parcijalna derivacija od C po b. Ako je gradijent smjer najvećeg rasta onda je negativni gradijent smjer najvećeg pada funkcije.

Gradijentni spust je iterativna metoda koja minimizira funkciju tako da se kreće u smjeru njenog negativnog gradijenta u malim koracima. Gradijentni spust se lako može predočiti ako se funkcija zamisli kao dolina. Na početku algoritma se točka nalazi negdje na strani doline, a cilj spusta je doći do dna s malim koracima nizbrdo. Slika 6 prikazuje proces gradijentnog spusta na jednostavnoj funkciji.



Slika 6 Gradijentni spust (Bhattarai, 2019)

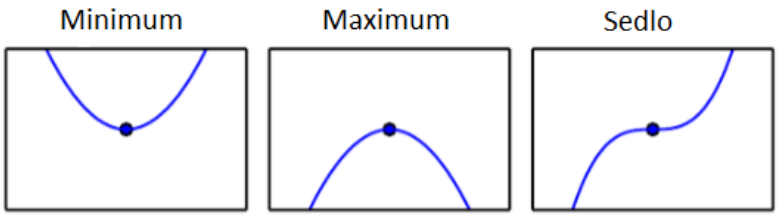
Radi jednostavnosti prikaza uzeta je funkcija $C(w, b)$ (formula 6) i njeni parametri su zamijenjeni s jednim sveobuhvatnim parametrom θ. Ovo dopušta da se gradijentni spust zapiše na sljedeći način:

$$\theta \rightarrow \theta' = \theta - \epsilon \nabla C \quad (7)$$

Gdje je ϵ veličina koraka ili stopa učenja (eng. *learning rate/step*), a ∇C iznos gradijenta s parametrima θ za primjer (x). Cilj je pronaći najbolje vrijednosti parametra θ koje će dati najmanju vrijednost funkcije C. Kreće se od neke početne vrijednosti θ i u svakom koraku se tu vrijednost promijeni za iznos negativnog gradijenta pomnožen sa stopom učenja. Iterativnom se metodom dolazi do parametara koji smanjuju gubitak. Ovaj postupak se ponavlja dok se ne dostigne kriterij zaustavljanja ili dok gradijent ne postane 0 (Ian Goodfellow, 2016).

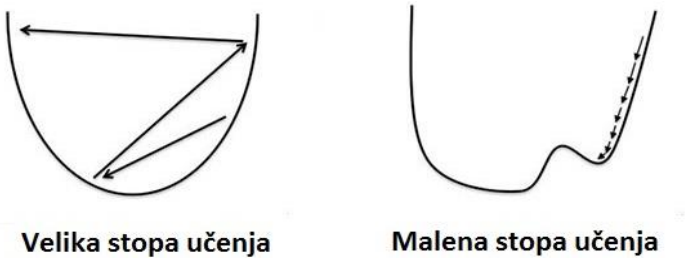
Globalni minimum je točka u funkciji za koju ta funkcija prima najnižu moguću vrijednost. Funkcija može imati jedan ili više globalnih minimuma, ali je i moguće da sadrži mnogo lokalnih minimuma koji nisu dovoljno dobri za svrhe modela. U kontekstu dubokog učenja funkcije su uglavnom takve da postoje razna mjesta gdje gradijentni spust može zapeti i davati loša rješenja. Gradijentni spust zapinje na mjestima gdje je gradijent jednak nuli. Takva mjesta se zovu kritične točke funkcije te tim u točkama algoritam staje jer ne može

odrediti u kojem smjeru se treba pomaknuti. Slika 7 prikazuje 3 vrste kritičnih točaka: minimumi, maksimumi i sedla (eng. *saddle*) (Ian Goodfellow, 2016).



Slika 7 Kritične točke

Procesom gradijentnog spusta će se uvijek doći do globalnog optimuma u funkcijama koje nemaju lokalnih optimuma. Funkcije gubitka koje se koriste za učenje neuronskiih mreža imaju mnoštvo lokalnih kritičnih točaka, a najčešće su lokalni minimumi, no oni u većini slučajeva daju dobre rezultate (Ian Goodfellow, 2016). Vrlo važan parametar kod gradijentnog spusta je stopa učenja ϵ. Kod velike stope učenja moguće je da će korak biti toliko velik da će u potpunosti preskočiti minimum te će s daljnjim učenjem mreže gubitak sve više rasti, a ako je stopa postavljena na premalu vrijednost onda će algoritmu trebati previše vremena da konvergira ili će zapeti u nekom lokalnom minimumu. Važno je odabrati dobru vrijednost stope učenja. Slika 8 prikazuje utjecaje prevelike i premalene stope učenja.



Slika 8 Utjecaji stope učenja (Bhattarai, 2019)

Postoje dvije vrste prolaza kroz mrežu tijekom učenja: unaprijedni i unatražni. Unaprijedni prolaz se odnosi na izračunavanje izlaza iz mreže temeljeno na nekom ulazu. Druga vrsta je prolazak unatrag koji se koristi tijekom izračunavanja gradijenta (Ian Goodfellow, 2016).

Potrebno je izračunati iznos gradijenta od kraja do početka mreže da bi se saznalo kako ažurirati parametre slojeva. Ažuriranje parametara tijekom unatražnog prolaza kroz mrežu je zapravo postupak koji uči mrežu. Metoda koja se koristi za izračun naziva se propagacija unatrag (eng. *backpropagation*). Propagacija unatrag često se krivo spominje kao algoritam učenja neuronske mreže, ali ona je samo način za dobivanje iznosa gradijenta, a algoritam gradijentnog spusta iskorištava izračunati gradijent da bi naučio mrežu (Nielsen, 2019).

Epoha je jedan unaprijedni i unatražni prolaz kroz mrežu za sve podatke za učenje. Neuronske mreže obično sadrže milijune parametara koji su međuovisni tako da su potrebne ogromne količine podataka za učenje da bi mreža uopće došla blizu optimalnog rješenja. Potreba za količinom podataka je još i veća ako se uzme u obzir da je gradijentni spust iterativna metoda koja se relativno sporo kreće prema optimalnom rješenju. Podaci koji se koriste za učenje mogu sadržavati korisne informacije za generalizaciju, ali gradijentnom spustu treba više prolaza kroz podatke da ih nauči. Broj dostupnih označenih podataka obično nije dovoljan da bi se jednim prolazom kroz mrežu došlo do zadovoljavajućih rezultata. Ovi problemi se rješavaju s korištenjem više epoha za učenje.

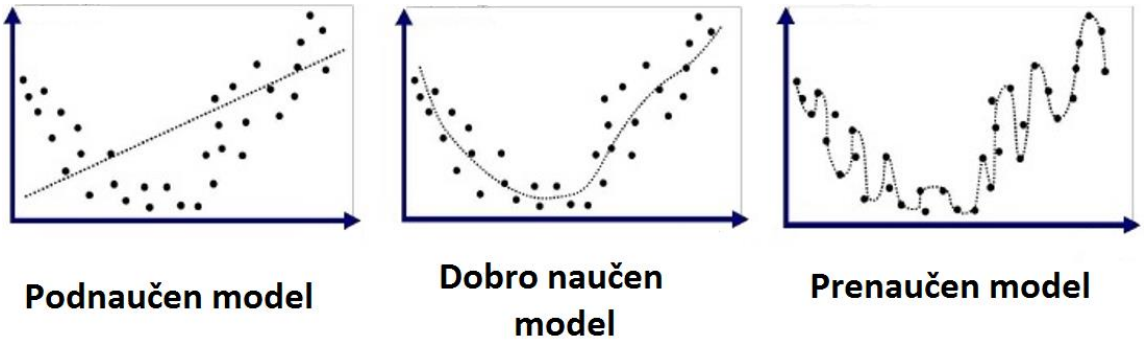
2.2.3 Podnaučenost i prenaučenost

Pojmovi koji su usko vezani za sposobnost generalizacije modela su prenaučenost (eng. *overfitting*) i podnaučenost (eng. *underfitting*).

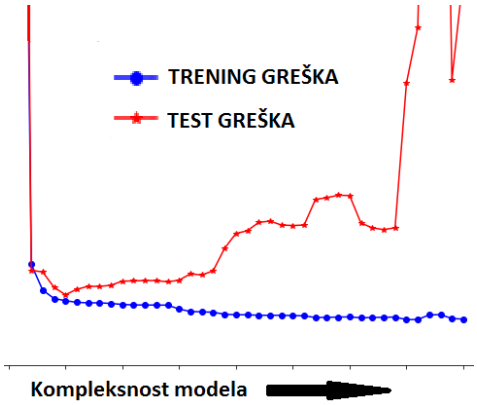
Prenaučenost se događa kad se model prilagođava podacima za učenje do takve mjere da postiže jako loše rezultate na podatcima koje nije vidio. Prenaučenost modela se lako može uočiti ako model ima odlične (skoro savršene) rezultate tijekom učenja, ali jako loše tijekom validiranja i testiranja. Prenaučenost se uglavnom događa zbog prevelike kompleksnosti modela koji onda ima sposobnost da napamet nauči podatke i tako smanji gubitak. Vjerojatnost prenaučenosti raste također s korištenjem prevelikog broja epoha za učenje. Ovo je nepoželjno ponašanje jer model s tim procesom nije naučio nikakve korisne značajke iz podataka i zbog toga ima loše rezultate na podacima koje nije vidio.

Podnaučenost se prvenstveno događa zbog prejednostavnog modela koji ili nema dovoljan broj neurona i slojeva ili koristi prejednostavne aktivacijske funkcije. U ovom slučaju model ne može ništa korisno izvući iz podataka jer nema gdje spremiti korisne informacije. Modeli koji su podnaučeni imaju veliku grešku na svim skupovima podataka. Podnaučenost također može biti posljedica nedostatka podataka za učenje ili nedovoljnog broja epoha za učenje.

Iz slike 10 se vidi da povećavanjem složenosti modela greška na skupu na učenje konstantno pada, a greška na skupu za testiranje se smanjuje do određene točke i najmanja je kad je složenosti modela najbliža stvarnoj složenosti podataka. Nakon te točke model postaje prenaučen i greška na skupu za testiranje kontinuirano raste jer je model počeo pamtiti podatke. Slika 9 prikazuje prednaučen, dobro naučen i prenaučen model.



Slika 9 Podnaučen, Dobro naučen i prenaučen model (Bhande, 2018)



Slika 10 Greška kod učenja i testiranja

2.2.4 Vrste gradijentnog spusta

Važno je razumjeti da su parametri (w, b) u kontekstu neuronskih mreža mnogobrojni te da ih se može brojati u milijunima. Primjerice, ako neki skup za učenje sadrži 10 000 primjera koji imaju 10 značajki. Količina računanja da se dobije vrijednost gradijenta za sve primjere bi bila: 10 000 * 10 = 100 000 puta za jednu epohu. Učenje mreže vrši se u više epoha te ako se uzme broj epoha 1000 onda je broj računanja samo za računanje gradijenta jednak 1000 * 100 000 = 100 000 000. Računanje derivacija je računski skupo te zbog toga postoji više načina kako da se smanji računska zahtjevnost izračuna gradijenta.

Gradijentni spust opisan u ovom poglavlju spada pod grupnu (eng. *batch*) kategoriju optimizacijskih postupaka jer je koristio cijeli skup primjera za učenje odnosno radi ažuriranje parametara kad prođe kroz sve primjere za učenje. Grupnim načinom računanja se dobije najtočnija vrijednost gradijenta na cijelom skupu, ali ovaj način je vrlo spor u usporedbi s drugima i potrebne su ogromne količine radne memorije da bi se izvršio.

U praksi se pokazalo da za dobro ponašanje modela nije potrebno izračunati potpuno točnu aproksimaciju gradijenta (Dabbura, 2017). Gradijenti spust koji koristi jedan po jedan primjer za učenje naziva se pojedinačni ili stohastički gradijentni spust (eng. *stochastic*

gradient descent). On nakon svakog primjera radi ažuriranje parametara. Često ažuriranje parametara uzrokuje bržu konvergenciju modela i manje je računski zahtjevno, ali su kod ovog načina jako česte oscilacije gradijenta jer se u obzir uzima gradijent samo jednog primjera koji ne mora nužno biti reprezentativan nad podacima.

Zadnja vrsta gradijentnog spusta u odnosu na količinu primjera odjednom iskorištenih za računanje gradijenta nalazi se na sredini između grupnog i stohastičkog i naziva se spust s mini grupama (eng. *minibatch*) gdje je broj uzoraka potreban za ažuriranje parametara određen hiperparametrom (engl. *batch size*). Mini grupe omogućuju računanje preciznijeg gradijenta bez velikih oscilacija stohastičkog spusta i nisu računski zahtjevne kao grupni algoritam (Dabbura, 2017). Slika 11 prikazuje kako se 3 vrste gradijentnog spusta ponašaju na putu prema minimumu (crni plus).

Slika 11 Gradijentni spust (Rizwan, 2018)

Razlog zašto algoritmi s mini grupama rade bolje od grupnih je što podaci za učenje vrlo često sadrže redundancije odnosno podatke koji su vrlo slični jedni drugima. To znači da nije potreban cijeli skup za učenje da bi se dobio dovoljno točan iznos gradijenta. Veličina grupe ne bi smjela biti prevelika jer velike grupe smanjuju generalizacijsku sposobnost modela (Nitish Shirish Keskar, 2016). Gradijentni spust s mini grupama u svakom koraku računanja gradijenta koristi samo mali broj podataka što znači da je količina izračuna u svakom koraku neovisna o veličini skupa za učenje. Ova činjenica omogućuje konvergenciju algoritma čak i kod velikih skupova, a ako je skup jako velik postoji mogućnost konvergencije prema konačnoj pogrešci na skupu za validaciju i prije nego što algoritam prođe kroz cijeli skup za učenje.

Jedno učenje mreže vrši se u više epoha, a svaka je epoha podijeljena u iteracije. Iteracija je broj mini grupa potreban da se završi jedna epoha. Primjerice, ako je veličina grupe 10, a broj sveukupnih podataka 150 potrebno je 15 iteracija za jednu epohu.

Uzorci za grupu bi trebali biti odabrani slučajnim uzorkovanjem tako da se dobije nepristrana procjena gradijenta. Nadalje, svaka uzastopna procjena gradijenta bi također trebala biti

nezavisna te bi se za svaku trebalo ponovno raditi uzorkovanje podataka. Stalno permutiranje podataka se pokazalo nepotrebnim i rješenje za nezavisni gradijent je početno permutiranje podataka u skupu za učenje i onda korištenje tog permutiranog redoslijeda za izgradnju mini grupa. Osnovni algoritam gradijentnog spusta opisan je sljedećim postupkom (Ian Goodfellow, 2016):

Zadano: *stopa učenja* ϵ
Zadano: *Inicijalni parametri* $\boldsymbol{\theta}$
dok *nije ispunjen uvjet zaustavljanja*:
  Uzmi minigrupu veličine m primjera iz skupa za treniranje $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$
    sa pripadajućim oznakama $\boldsymbol{y}^{(i)}$
  Izračunaj procjenu gradijenta: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L\left(f\left(x^{(i)}; \boldsymbol{\theta}\right), \boldsymbol{y}^{(i)}\right)$
  Ažuriraj parametre: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \epsilon \hat{\boldsymbol{g}}$

Norma gradijenta kod grupnog gradijentnog spusta smanjuje se napretkom optimizacije jer u lokalnom ili globalnom minimumu pravi gradijent postaje nula te algoritam može koristiti fiksnu stopu učenja. Kod ostalih algoritama u svakom koraku se samo procjenjuje prava vrijednost gradijenta te je s tom činjenicom neizbježno postojanje šuma koji postaje veći obrnuto proporcionalno veličini mini grupe. Posljedica šuma je ta što gradijent neće postati nula u lokalnom ili globalnom minimumu i moguće je da algoritam ne konvergira. Ovaj problem se rješava korištenjem promjenjive stope učenja. Stopa učenja se smanjuje kroz vrijeme tako da se smanji utjecaj šuma u procjeni gradijenta. Postoje 2 uvjeta za smanjivanje stope učenja koja su dovoljna da bi se osigurala konvergencija algoritma:

$$\sum_{k=1}^{\infty} \epsilon_k = \infty, \sum_{k=1}^{\infty} \epsilon_k^2 < \infty$$

Jednostavno rješenje za smanjivanje stope je njeno linearno smanjivanje od ϵ_0 u koraku 0 do ϵ_t gdje je t korak nakon kojeg smanjivanje prestaje. Za k > t stopa ostaje na zadnjoj vrijednosti i nakon toga se više ne smanjuje, a za $k \leq t$ vrijedi: $\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_t$, gdje je $\alpha = \frac{k}{t}$. Kod odabira ϵ_0 mora se paziti da vrijednost ne bude prevelika jer može uzrokovati osciliranje, nestabilnost ili divergenciju postupka. Mora se uzeti i u obzir da vrijednost ne bude premala jer u tom slučaju algoritam može napredovati vrlo sporo ili zaglaviti rano u vrlo lošem lokalnom minimumu.

Jedna od metoda ubrzavanja učenja s gradijentnim spustom je učenje s momentom. Koristi se za ubrzavanje kad su vrijednosti gradijenta relativno malene, ali konzistentne ili kad procjena gradijenta sadrži puno šuma. Postupak učenja s momentom akumulira eksponencijalno umanjujući prosjek prethodnih gradijenata te nastavlja učenje u tom smjeru. Učenje s momentom opisano je sa sljedeće dvije formule.

$$v \leftarrow \alpha \cdot v - \epsilon \cdot \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)}; \theta, y^{(i)} \right) \quad (8)$$

$$\theta \leftarrow \theta + v \quad (9)$$

Varijabla v predstavlja eksponencijalno umanjujući prosjek prethodnih gradijenata, a α je parametar koji govori koliko je relevantan prethodno izračunati prosjek odnosno određuje brzinu umanjivanja utjecaja prethodnih gradijenata. Vrijednost v označava moment (eng. *velocity*). Vrijednost parametra α je između 0 i 1, a što je on veći u odnosu na stopu učenja ϵ to je veći utjecaj prethodnih gradijenata na smjer ažuriranja. Korekcija gradijenta u koraku k će biti najveća kada su gradijenti poravnati u uzastopnim koracima. Algoritam gradijentnog spusta s momentom opisan je sljedećim postupkom (Ian Goodfellow, 2016):

Zadano: *stopa učenja* ϵ, *parametar momentuma* α
  Zadano: *Inicijalni parametri* $\boldsymbol{\theta}$, *inicijalna brzina* $\boldsymbol{v}$
  dok *nije ispunjen uvjet zaustavljanja*:
    Uzmi minigrupu veličine m primjera iz skupa za učenje $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$
      sa pripadajućim oznakama $\boldsymbol{y}^{(i)}$
    Izračunaj procjenu gradijenta: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L\left(f\left(x^{(i)}; \boldsymbol{\theta}\right), \boldsymbol{y}^{(i)}\right)$
    Izračunaj procjenu brzine $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
    Ažuriraj parametre: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

Postoje algoritmi gradijentnog spusta koji umjesto predodređenog načina promjene stope koriste adaptivno upravljanje njenom vrijednosti. Jedan od najpoznatijih i najpopularnijih adaptivnih algoritama se zove ADAM (eng. *Adaptive Moments*). Adam algoritam prati uprosječeno kretanje gradijenta i kvadrata gradijenta uz dodatno eksponencijalno zaboravljanje. Adam u sebi implicitno sadrži moment gradijenta te se korekcija radi prema uprosječenom gradijentu, a ne izračunatom. Adam postupak (Ian Goodfellow, 2016) opisan je na sljedeći način:

$\boldsymbol{Zadano}$: *stopa učenja* ϵ

$\boldsymbol{Zadano}$: *stope eksponencijalnog smanjivanja procjene momenta,* ρ_1 *i* ρ_2 *u rasponu* $[0, 1)$

$\boldsymbol{Zadano}$: *mala konstanta* $\boldsymbol{\delta}$ *koja se koristi za numeričku stabilizaciju*

$\boldsymbol{Zadano}$: *Inicijalni parametri* $\boldsymbol{\theta}$

Inicijaliziraj prvu i drugu varijablu momenta: $\boldsymbol{s} = \boldsymbol{0}, \boldsymbol{r} = \boldsymbol{0}$

Inicijaliziraj vremenski korak $t = 0$

$\boldsymbol{dok}$ *nije ispunjen uvjet zaustavljanja:*

Uzmi minigrupu veličine m primjera iz skupa za treniranje $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ *sa pripadajućim oznakama* $\boldsymbol{y}^{(i)}$

Izračunaj gradijent: $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L\left(f\left(x^{(i)}; \boldsymbol{\theta}\right), \boldsymbol{y}^{(i)}\right)$

$t \leftarrow t + 1$

Ažuriraj pristranu procjenu prvog momenta: $\boldsymbol{s} \leftarrow \rho_1 s + (1 - \rho_1)\boldsymbol{g}$

Ažuriraj pristranu procjenu drugog momenta: $\boldsymbol{r} \leftarrow \rho_1 \boldsymbol{r} + (1 - \rho_1)\boldsymbol{g} \odot \boldsymbol{g}$

Ispravi pristranost u prvom momentu: $\hat{s} \leftarrow \frac{s}{1 - \rho_1^t}$

Ispravi pristranost u drugom momentu: $\hat{r} \leftarrow \frac{r}{1 - \rho_2^t}$

Izračunaj iznos gradijenta: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$

Ažuriraj parametre: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

2.2.5 Aktivacijske funkcije

Aktivacijske funkcije su matematičke funkcije koje određuju izlaz neuronskih mreža. Funkcija je pridružena svakom neuronu u mreži i odlučuje hoće li se neuron aktivirati ili ne temeljeno na tome jesu li ulazne vrijednosti tog neurona važne ili ne za predviđanje cijelog modela.

Aktivacijske se funkcije primjenjuju na rezultat izračunat s množenjem ulaza s težinama te dodavanjem pristranosti. Jedno korisno svojstvo aktivacijskih funkcija je to što pomažu prilikom normalizacije izlaza tako da izlaz ograničavaju na vrijednosti između 0 i 1 ili -1 i 1. Dodatna svrha aktivacijskih funkcija je dodavanje nelinearnosti izlazu neurona, ali najvažnije svojstvo koje se nadovezuje na nelinearnost je to što omogućavaju učenje mreže korištenjem propagacije unatrag.

Propagacija unatrag omogućena je činjenicom da su sve aktivacijske funkcije derivabilne (barem djelomično), a za propagaciju su potrebne derivacije prvog reda. Bez aktivacijskih funkcija mreža bi bila samo linearni model i ne bi mogla naučiti kompleksne veze nad podacima. Svojstvo koje aktivacijske funkcije moraju zadovoljiti je to da moraju biti računski učinkovite jer će se računati milijune puta za svaki podatak tim više ako se koristi propagacija unatrag.

Postoje 3 tipa aktivacijskih funkcija.

Binarna stepenasta funkcija je tip funkcije koji daje izlaz 1 ako je izračun unutar neurona zadovoljio neki uvjet, a inače daje izlaz 0. Ovakvu funkciju koristi perceptron opisan u prijašnjem poglavlju. Korištenjem ovakve funkcije nije moguće dobiti model koji radi klasifikaciju s više od dvije klase.

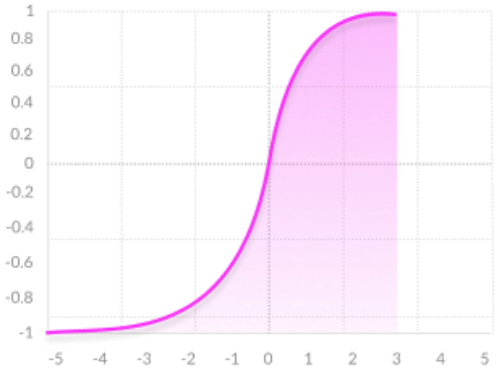
Druge vrsta aktivacijskih funkcija je linearna. Linearne aktivacijske funkcije koriste rezultat množenja težina s ulazom i dodavanjem pristranosti kao izlaz iz neurona. Ovakve funkcije su bolje od binarnih jer dopuštaju multi klasifikaciju, ali se ne koriste jer imaju dodatne probleme. Prvi je što se za učenje mreže ne može koristiti propagacija unatrag jer je derivacija funkcije uvijek konstanta i ne mijenja se u odnosu na ulaz u neuron što znači da se ne mogu izvući korisne informacije iz gradijenta.

Drugi problem je što se svi slojevi korištenjem ovakve funkcije urušavaju u jedan. Zadnji sloj mreže će biti linearna funkcija od prvog sloja jer je linearna kombinacija linearnih funkcija još uvijek linearna funkcija tako da cijela mreža postane jedan sloj. Neuronska mreža s linearnom aktivacijskom funkcijom je jednostavni linearni regresijski model koji ima vrlo ograničenu sposobnost učenja. Aktivacijske funkcije koje se danas koriste su uglavnom nelinearne jer dopuštaju modelu da nauči kompleksne odnose između ulaza i izlaza mreže na nelinearnim podacima (slike, video, audio...).

Nelinearne funkcije rješavaju probleme linearnih jer imaju derivaciju koja se mijenja u odnosu na ulaz u neuron i omogućuju višeslojnost mreža jer se slojevi više ne urušavaju u jedan kao kod linearnih aktivacijskih funkcija. Sigmoida je osnovni primjer nelinearne funkcije te je opisana u prijašnjem poglavlju. Sigmoida pruža glatke vrijednosti gradijenta te ih ograničava na vrijednosti između 0 i 1. Dodatna prednost sigmoide je što daje predviđanja s velikom sigurnosti jer svaka vrijednost iznad 2 ili ispod -2 se mapira vrlo blizu 1 ili 0 (eng. *clear prediction*). Najveći problem sigmoidne funkcije je nestajući gradijent koji

se događa za vrlo velike vrijednosti izračunate u neuronu. Vrlo velike vrijednosti se i dalje mapiraju na opseg od 0 do 1 što znači da nema promjene u predviđanju s takvim vrijednostima. Ovo može uzrokovati prestanak učenja ili presporo učenje. Dodatni problemi sigmoide su ti što izlazi nisu centrirani oko nule i što je dosta skupa za računanje. Hiperbolni tangens je aktivacijska funkcija koja rješava problem centriranja i mapira izlaz neurona od -1 do 1. Zbog centriranog izlaza optimizacija je lakša jer se lakše mapiraju jako negativne, pozitivne ili neutralne vrijednosti. Hiperbolni tangens računa se formulom (10), a graf mu je prikazan na slici 12:

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \qquad (10)$$



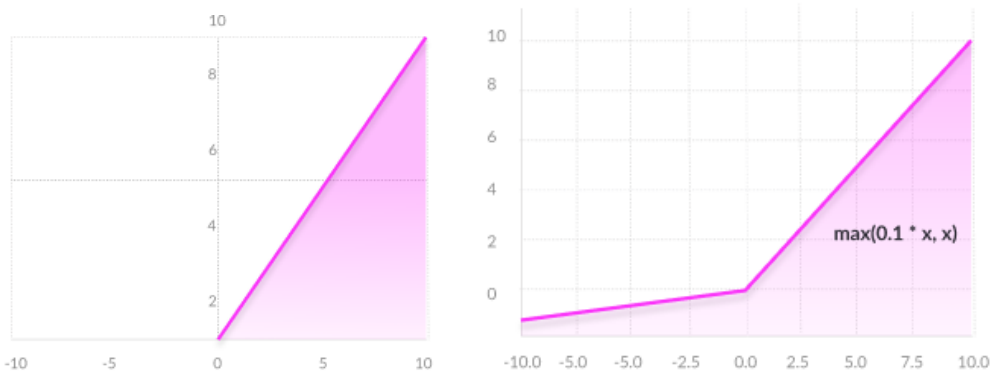
Slika 12 Tangens hiperbolni (MissingLink.ai, 2019)

RelU (eng. *rectified linear unit*) je aktivacijska funkcija koja izgleda linearno, ali zapravo nije te dopušta učenje propagiranjem unatrag. Vrlo je računski nezahtjevna i računa se formulom $R(x) = \max(0, x)$.

RelU funkcija rješava problem nestajućeg gradijenta kod velikih vrijednosti, ali se kod ove funkcije pojavljuje problem kad su ulazi u funkciju vrlo blizu nuli ili negativni. U ovom slučaju funkcija postaje 0 i taj neuron se nikada više neće aktivirati, a takav neuron se naziva mrtvim (eng. *dead neuron*). Graf RelU funkcije prikazan je na slici 13 iz koje se vidi da je za negativne vrijednosti iznos RelU funkcije uvijek 0.

Rješenje problema mrtvih neurona je korištenje modifikacije RelU funkcije tako da se napravi blaga padina na dijelu grafa ispod nule. Ovakav RelU je prikazan na slici 13 i naziva se RelU „koji curi“ (eng. *leaky relu*). Leaky RelU računa se formulom $R(x) = max(0.1 * x, x)$. Leaky RelU funkcija ne daje vrlo konzistentna predviđanja za negativne vrijednosti, ali se to nije pokazao kao prevelik problem u praksi. (MissingLink.ai, 2019) Zadnji problem svih RelU funkcija je taj što se ne mogu koristiti u svim slojevima

mreže već samo u skrivenim. Za izlaz mreže se najčešće koristi softmax funkcija. Softmax se koristi kad je cilj mreže klasifikacija više od dvije klase te on uzima sve izlaze iz neurona te ih skalira na vjerojatnosti pripadanja klasi.



Slika 13 RelU (lijevo) i Leaky RelU (desno) (MissingLink.ai, 2019)

2.2.6 Unakrsna entropija

Klasifikacijski problemi strojnog učenja u kojima postoje samo dvije vrste izlaza nazivaju se binarni klasifikacijski problemi, a oni koji imaju više oznaka nazivaju se kategorijski ili multi klasifikacijski. Unakrsna entropija (eng. *cross entropy*) se često koristi kao funkcija gubitka za optimizaciju klasifikacijskih modela, a zbog korištenja logaritma u izračunima naziva se i logaritamska funkcija gubitka (eng. *log loss*).

Za svaki primjer za učenje poznata je vjerojatnosna distribucija P koja se interpretira kao vjerojatnost pripadanja klasi. Svaki primjer za učenje ima točno određenu oznaku za klasu kojoj pripada s vjerojatnosti 1.0, a vjerojatnosti 0.0 za sve klase kojima ne pripada. Distribucija P je zapravo očekivana vjerojatnost ili točna oznaka klase (y). Vjerojatnosti od 1.0 i 0.0 su sigurni ili nemogući događaji što znači da ne sadrže nikakve korisne informacije odnosno da imaju nula entropije. Model procjenjuje vjerojatnost pripadanja primjera svakoj klasi. Ova distribucija se naziva Q i ona je predviđena vjerojatnost ($\acute{y}$). Unakrsna entropija se onda može iskoristiti da se izračuna različitost tih dviju distribucija. Formula za jedan primjer u multi klasifikacijskom problemu je sljedeća:

$$H(P,Q) = -\sum_{c=1}^{M} P(x_c) * \log\big(Q(x_c)\big) \qquad (11)$$

Gdje M označava broj klasa kojima primjer može pripadati, $P(x_c)$ je stvarna vjerojatnost pripadanja primjera klasi odnosno 1, za točnu klasu, a 0 za sve ostale, a $Q(x_c)$ je predviđena pripadnost klasi koju je model izračunao. Ako je model postigao potpuno točno predviđanje

za taj primjer izraz pod sumom će biti jednak 0 i samim tim će i entropija postati 0. U suprotnom slučaju, ako je model potpuno pogriješio i dao vjerojatnost 0 točnoj klasi (ili vrijednost vrlo blizu 0) onda će entropija biti velika jer će izraz pod logaritmom eksplodirati te će samim tim i cijeli gubitak biti velik. U binarnom slučaju klasifikacije unakrsna entropija se može napisati na sljedeći način:

$$H(P,Q) = -(P(x_0) * \log\big(Q(x_0)\big) + P(x_1) * \log\big(Q(x_1)\big) \qquad (12)$$

Gdje 0 i 1 predstavljaju dvije klase kojima primjer može pripadati. U binarnom slučaju vjerojatnost se modelira po Bernoullijevoj distribuciji za pozitivnu oznaku klase. Ovo znači da se pripadnost pozitivnoj klasi dobiva iz modela, a pripadnost drugoj se izračuna kao $1 - predviđena\ vjerojatnost$. U ovom slučaju to znači da je izlaz modela samo jedan broj. Kod korištenja unakrsne entropije kao gubitka bitno ju je minimizirati nad svim primjerima. Formula za gubitak unakrsne entropije za više primjera je sljedeća:

$$H(P,Q) = -\frac{1}{N}(y_i * \log(P(y_i)) + (1 - y_i) * \log(1 - P(y_i)) \qquad (13)$$

Gdje y_i označava točnu oznaku primjera, a $P(y_i)$ predviđenu vjerojatnost pripadanja primjera klasi.

2.3 Metrike modela

Postoje mnoge metrike kojima se mjeri „dobrota“ modela i koje mogu pomoći kod uočavanja problematičnih dijelova modela. Kod binarne klasifikacije izlaz modela je pripadnost modela kategoriji odnosno vjerojatnost da taj primjer pripada kategoriji. Za binarnu klasifikaciju izlazi iz modela se mogu rasporediti u četiri kategorije.

1. Stvarni pozitivni (eng. *true positive*, nadalje TP) primjeri su oni za koje je model predvidio da pripadaju nekoj kategoriji, a oni njoj stvarno pripadaju.
2. Stvarni negativni (eng. *true negative*, nadalje TN) primjeri su oni za koje je model predvidio da ne pripadaju kategoriji, a oni stvarno ne pripadaju kategoriji.
3. Lažni pozitivni (eng. *false positive, nadalje FP*) su primjeri za koje je model predvidio da pripadaju kategoriji, a oni toj kategoriji ne pripadaju.
4. Lažni negativni (eng. *false negative*, nadalje FN) su primjeri za koje je model previdio da ne pripadaju kategoriji, a oni njoj u stvarnosti pripadaju.

Ove četiri vrijednosti se slažu u matricu zbunjenosti (eng. confusion matrix) prikazanoj u Tablica 1.

Tablica 1 Matrica zbunjenosti

	1	**0**
1	TP	FN
0	FP	TN

Oznake u prvom stupcu označavaju stvarnu pripadnost primjera, a oznake u prvom retku predviđenu pripadnost primjera klasi. Matrica zbunjenosti je dobar način da se vide generalne performanse modela. Prikazana matrica na slici odnosi se na binarnu klasifikaciju, a proširuje se za veći slučaj s dodavanjem redaka i stupaca jednak broju klasa koji model mora predvidjeti. Točnost klasifikacije (eng. *accuracy*) je broj točnih izlaza modela podijeljen s brojem svih izlaza modela. Za binarnu klasifikaciju računa se formulom:

$$A = \frac{TP + TN}{TP + TN + FP + FN} \qquad (14)$$

Kod više klasnog izlaza modela podijeli se broj točnih klasifikacija s brojem svih klasifikacija. Poželjno je za model da ima što veću točnost tijekom testiranja. Prevelika točnost tijekom učenja je dobar znak prenaučenosti modela. Preciznost modela pokazuje kolika proporcija pozitivnih predviđanja su zapravo došla od pozitivnih primjera. Model koji ima nula lažno pozitivnih (FP) primjera ima preciznost 1. Preciznost (eng. *precision*) se računa kao udio pravih pozitivnih (TP) predviđanja sa svim pozitivnim predviđanjima (TP + FP) formulom:

$$P = \frac{TP}{TP + FP} \qquad (15)$$

Odziv (eng. *recall*) modela odgovara na pitanje koji je udio stvarno pozitivnih primjera bio točno označen. Modeli koji imaju nula lažno negativnih primjera imaju odziv 1. Matematički se definira formulom:

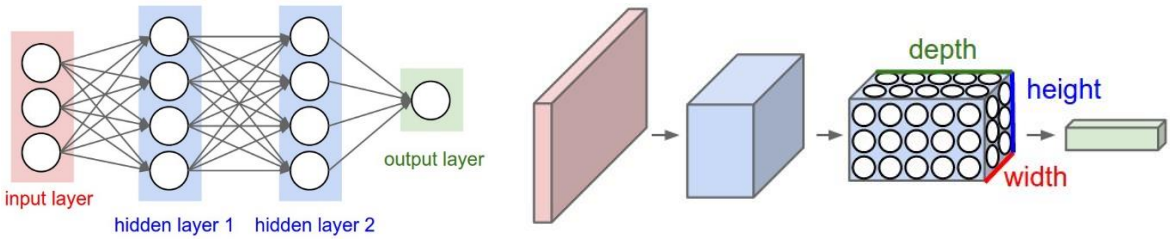
$$R = \frac{TP}{TN + FN} \qquad (16)$$

2.4 Slojevi

Neuronske se mreže sastoje od raznoraznih slojeva. U dosadašnjim primjerima svi su slojevi bili potpuno povezani (eng. *fully connected*, FC) što znači da je svaki neuron nekog sloja povezan sa svim neuronima iz prijašnjeg te sa svim neuronima sljedećeg sloja. Ne postoji lokalnost među neuronima nego svi utječu na sve u susjednim slojevima te ne postoji nikakvo dijeljenje parametara unutar istog sloja (svaki neuron ima svoje težine koje pripadaju samo njemu). Poseban je sloj zadnji, izlazni koji sadrži rezultate klasifikacije.

Neuronske mreže koje imaju samo FC slojeve nisu dobre za slikovne podatke. Primjerice, ako mreža prima ulaz slike veličine 200x200x3 (širina, visina, 3 kanala boje) prvi FC sloj bi imao 200x200x3 = 120,000 težina samo za jedan neuron. Mrežama je potrebno više neurona u sloju tako da bi broj parametrima jako brzo rastao i postojala bi vrlo velika vjerojatnost prenaučenosti. Još jedan razlog za loš performans FC mreža je taj što one ne mogu naučiti prostorne ovisnosti na slici (ovo ne mogu jer se slika pretvara u vektor veličine širina x visina x dubina) (Stanford University). Ove probleme rješavaju konvolucijske mreže.

Konvolucijske mreže uče pod pretpostavkom da su ulazni podaci slike i dizajnirane su da najbolje rade sa slikovnim podacima. Prva bitna promjena u odnosu na FC mreže je raspored neurona u 3 dimenzije (širina, visina i dubina[1]) u svakom sloju. Druga razlika je lokalna povezanost. Lokalna povezanost znači da je svaki neuron povezan samo s jednim malim dijelom prijašnjeg i sljedećeg sloja. Slika 14 prikazuje razliku između klasične i konvolucijske mreže. Svaki sloj konvolucijske mreže uzima 3D ulaz te ga pretvara u 3D izlaz. Na slici (desno) je crveno prikazan ulaz koji odgovara veličini slike. Mreža konvolucijom pretvara ulaz vektor duljine broja klasa.



Slika 14 FC mreža(lijevo), konvolucijska mreža (desno) (Stanford University)

U konvolucijskim mrežama obično se koriste tri vrste slojeva: konvolucijski, udružni (eng. *pooling*) i FC slojevi. Neki od ovih slojeva nemaju parametre koji se uče već uvijek

[1] Dubina se ovdje odnosi na treću dimenziju težina neurona, a ne na dubinu mreže koja se odnosi na broj slojeva

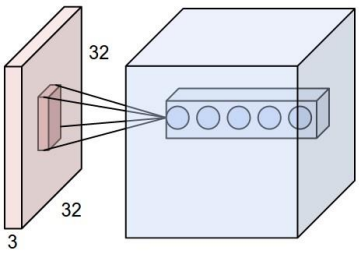
izvršavaju istu funkciju. Konvolucijske mreže se uče gradijentnim spustom koji uči i konvolucijske i FC slojeve.

Konvolucijski sloj sastoji se od filtera koji se moraju naučiti. Svaki filter je prostorno malen te gleda samo na manji dio slike po širini i visini, ali se proteže kroz njenu cijelu dubinu. Primjerice, filter veličine 5x5 gleda 5x5 piksela ulaza, ali uzima u obzir i dubinu od svakog od tih piksela. Tijekom unaprijednog prolaza po mreži filter se „pomiče“ po ulazu po njegovoj širini i visini te računa vektorski produkt između ulaznih vrijednosti na toj poziciji i svojih težina (pomicanje se naziva konvolucija, u ovom slučaju 2D konvolucija). Pomicanjem filtera po ulazu stvara se 2D izlaz koji odgovara izlazu filtera na svakoj od pozicija (Stanford University).

Obično prvi sloj naučenih filtera u konvolucijskoj mreži prepoznaje jednostavne elemente na slici (primjerice rubovi objekata), a što se ide dublje u mrežu to prepoznati elementi postaju sve kompleksniji. Svaki konvolucijski sloj ima više filtera od kojih svaki proizvede svoj 2D izlaz. Izlazi svih filtera se slažu jedan pored drugog te to tvori treću dimenziju izlaza iz sloja (svaki sloj gledano po dubini naziva se rez).

Svaki konvolucijski neuron povezan je s malom lokalnom regijom ulaza. Veličina regije je hiperparametar mreže koji se zove receptivno polje (eng. *receptive field*) koji je ujedno i veličina filtera.

Postoje 3 hiperparametra sloja koja određuju kakav će biti njegov izlaz: dubina, veličina koraka (eng. *stride*) i dopunjavanje nulama (eng. *zero padding*). Dubina je samo broj filtera koji se nalaze u sloju. Idealno svaki od filtera bi se trebao aktivirati kad vidi različiti element na slici, primjerice neki se aktiviraju kad vide određenu boju, a neki kad detektiraju horizontalni rub na slici. Broj neurona koji gleda na istu regiju slike zove se dubinski stupac (eng. *depth column*) prikazan na Slika 15.



Slika 15 Dubinski stupac veličine 5 (Stanford University)

Veličina koraka određuje koliko daleko će se filteri pomicati u svakom koraku. Kad je korak veličine jedan filteri se pomiču za jedan piksel. Povećavanjem veličine koraka smanjuje se izlazna veličina sloja (neki pikseli se preskaču s većim korakom). Ulazi se mogu dopunjavati s nulama, a to znači da se nule dodaju na rubove slike te ovo omogućuje da se veličina ulaza ne smanjuje prolazom kroz konvolucijski sloj.

Veličina izlaza može se izračunati s formulom $\frac{W-F+2P}{S} + 1$, gdje je W veličina ulaza, F veličina filtera, S veličina koraka, a P broj dodanih nula na rub ulaza (Stanford University). Svi ovi parametri su međuovisni. Primjerice ako je W = 10, P = 0, F = 3 onda bi bilo nemoguće koristiti S = 2 jer bi rezultat formule bio 4.5 što nije cijeli broj što znači da se neuroni ne mogu uredno rasporediti po ulazu.

Konvolucijski slojevi često koriste dijeljenje parametara (eng. *parameter sharing*). Dijeljenje parametara koristi se da bi se smanjio sveukupni broj parametara, a može se koristiti pod pretpostavkom da, ako je korisno detektirati objekt na jednom dijelu slike, onda ga je korisno detektirati bilo gdje na slici. Drugim riječima, vrlo korisna značajka mreže bi bila da su filteri neovisni o translaciji jer su slike također neovisne o translaciji (ovo znači da bez obzira gdje se značajka nalazi na slici i dalje se može prepoznati kao ta ista značajka). Ovo se postiže tako da svi neuroni unutar istog dubinskog „reza“ koriste iste neurone (Stanford University).

Postoje i slučajevi gdje dijeljenje parametara nije poželjno. Primjerice u detekciji lica važno je na kojem mjestu se nalaze oči, usta i nos te takvi podaci više nisu neovisni o translaciji.

Svrha udružnog sloja je da smanji visinu i širinu ulaza tako da se smanji broj parametara u mreži te da se smanji vjerojatnost prenaučenosti. Obično se udružni slojevi stavljaju u mrežu između konvolucijskih. Najčešća vrsta udružnog sloja koristi filter veličine 2x2 s veličinom koraka 2. Koristi se operacija konvolucije, ali izlaz svakog filtera je samo jedna vrijednost.

Najčešća vrsta udruživanja je max udruživanje koje iz pročitanih ulaznih vrijednosti uzima najveću. U slučaju veličine filtera 2x2 i koraka veličine 2 ulazna veličina se smanjuje za 75%. Ako je ulazna veličina u sloj W1 x H1 x D1 onda je izlaz iz sloja jedan W2 x H2 x D1 gdje je $W2 = \frac{W1-F}{S} + 1$, $H2 = \frac{H1-F}{S} + 1$, S je veličina koraka, a F veličina filtera.

Udružni sloj ne sadrži nikakve parametre koji se uče. Glavna razlika između FC i konvolucijskih slojeva je lokalna povezanost s ulazom. Nakon konvolucijskih slojeva u

mrežu se dodaje nekoliko FC slojeva koji uzimaju izlaz konvolucije i pretvaraju ga u krajnji izlaz.

3. PRIJENOSNO UČENJE

U klasičnom nadziranom učenju, ako se model uči za neki zadatak i domenu tog zadatka A, mora se pretpostaviti da postoje označeni podaci za zadatak A iz te domene. Zadatak je funkcija modela koja se želi ostvariti (primjerice prepoznati znamenke na slici), a domena je izvor od kud podaci dolaze.

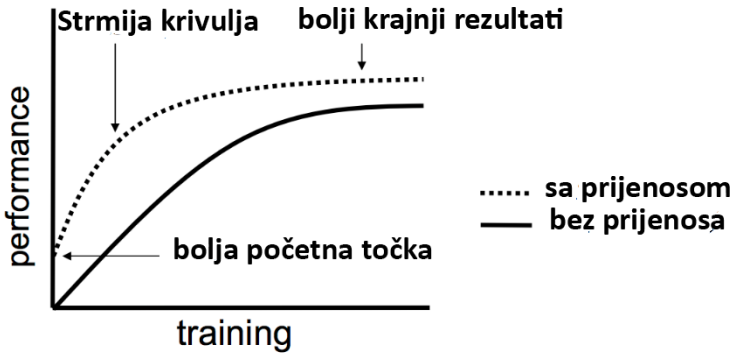
Učenjem modela A na skupu podataka A postiže se dobra sposobnost generalizacije za isti zadatak i domenu. U drugom slučaju za učenje modela koji ima zadatak i domenu B opet su potrebni označeni podaci iz te domene koji se mogu iskoristiti za učenje modela B i za taj model se može očekivati dobra sposobnost generalizacije za zadatak i domenu B.

Klasična se paradigma nadziranog učenja raspada ako ne postoji dovoljan broj označenih podataka za zadatak i domenu na kojoj se želi naučiti dobar model. Primjerice, ako je zadatak nekog modela detekcija pješaka po danu može se pronaći neki model koji je naučen za isti zadatak samo po noći. U praksi ovakvi postupci ne funkcioniraju jer je drugi model učen samo za slike po noći i model je postao previše pristran prema noćnim podacima. Dodatno, ako se zadatak modela promijeni u primjerice detekciju biciklista onda se drugi model ne može uopće iskoristiti jer su potrebne oznake podataka koje su drugačije od originalnih. Ovakva se ograničenja i problemi rješavaju prijenosnim učenjem (Gupta, 2017).

Prijenosno učenje (eng. *transfer learning*) je metoda gdje se model naučen na sličnom zadataku iskorištava kao početna točka za novi model. Prednaučeni model je model koji je već netko drugi naučio da riješi sličan zadatak. Za uspjeh se prijenosnog učenja podrazumijeva da je prednaučeni model dobro naučio značajke i da je izvukao korisne informacije iz podataka tijekom svog učenja. Ovaj način prijenosnog učenja u dubokom se učenju naziva induktivni prijenos (eng. *inductive transfer*).

Postoje tri glavne prednosti koje prijenosno učenje omogućuje. Prva je bolja početna točka u učenju modela (eng. *higher start*). Ovo znači da prednaučeni model prije bilo kakvih modifikacija daje bolje rezultate nego originalni, a samim tim put prema optimumu postaje kraći. Druga prednost je brža sposobnost učenja (eng. *higher slope*). Prednaučeni model može imati puno strmiju krivulju poboljšanja od modela kojeg se uči od nule. Zadnja

prednost prijenosnog učenja je da prednaučeni model može dati puno bolje krajnje rezultate. (eng. *higher asimptote*) (Gupta, 2017). Slika 16 prikazuje tri prednosti prijenosnog učenja.



Slika 16 Prednosti prijenosnog učenja (Gupta, 2017)

Prednaučeni modeli koji dobro generaliziraju na raznim podacima su danas dostupni na raznim platformama. Ti modeli su obično učeni na ogromnim količinama podataka kao što su primjerice Imagenet koji sadrži na desetke milijuna slika. Kod korištenja prednaučenih modela rade se razne promjene nad njima da bi se prilagodili specifičnom zadatku. Pretpostavka je da su oni dobro naučeni i zbog toga se njihovi parametri ne bi trebali previše i prebrzo mijenjati te se zbog toga često koriste stope učenja koje su manje od onih korištenih za inicijalni model. Mijenjanje ovakvih modela naziva se fino ugađanje.

Postoje tri najpopularnija načina korištenja prednaučenih modela

1. Iskorištavanje modela za izdvajanje značajki (eng. *feature extraction*)
2. Korištenje arhitekture modela
3. Smrzavanje nekih, a učenje drugih slojeva

Iskorištavanje modela za izdvajanje značajki podrazumijeva korištenje cijelog modela osim njegovog zadnjeg sloja – izlaza. Izbacivanjem zadnjeg sloja model vraća naučene značajke na novim podacima, a s tim podacima se mogu učiti dodatni FC slojevi.

Korištenjem arhitekture modela se cijeli model uči iz početka na novim podacima s nasumično inicijaliziranim vrijednostima težina.

Smrzavanje sloja znači da se njegovi parametri neće mijenjati tijekom učenja već će se koristiti neizmijenjene prednaučene vrijednosti. Najčešće se smrzavaju niži slojevi, a ponovno uče viši. Način korištenja prednaučenih slojeva ovisi o dva faktora: sličnost i broj podataka u novom skupu za učenje.

Ako je broj podataka malen, a sličnost velika onda se model ne mora ponovno učiti. U ovom slučaju potrebno je modificirati izlazne slojeve modela da se prilagode zadatku koji se rješava. Model se koristi za izdvajanje značajki. Primjer je korištenje modela učenog za klasifikaciju na imagenet setu podataka za predviđanje je li se na slici nalazi pas ili mačka. Slike koje bi se koristile su dosta slične slikama u imagenet setu, ali potrebne su samo dvije izlazne klase. Modificirali bi se zadnji FC slojevi i izlazni sloj da daje veličinu dva za dvije klase – pas i mačka.

Ako je za potrebni zadatak dostupno malo podataka i sličnost podataka je malena koristi se dijelomično učenje modela. Određeni broj slojeva (k) prednaučenog modela se smrzne, a ostali (n – k) se ponovno uče. Slojevi koji se smrzavaju su uvijek donji u modelu (najbliži ulazu). Ovim načinom se gornji slojevi modificiraju i prilagođavaju novim podacima. Zbog male sličnosti i broja podataka potrebno je ponovno učiti gornje slojeve tako da se dobro prilagode novim podacima. Mali broj podataka se kompenzira smrzavanjem i korištenjem donjih slojeva koji su već učeni na velikom skupu podataka.

Ako postoji puno podataka za novi zadatak, a koji su vrlo različiti od podataka nad kojima je mreža prednaučena potrebno je ponovno učiti cijelu mrežu. Iskorištava se samo arhitektura mreže bez naučenih težina, jer je razlika u podacima toliko velika da su izlazi prednaučene mreže beskorisni. Zadnji slučaj je velika sličnost i velika količina podataka za novi zadatak. Ovo je idealna situacija u kojoj bi prednaučeni model trebao postići najbolje rezultate i najviše olakšati rješavanje zadataka. Najbolji način o ovom slučaju je učiti model sa svim težinama, ali s manjom stopom učenja od one koja se koristila kod učenja prednaučene mreže (Gupta, 2017).

4. CAN BUS

Controller Area Network (CAN) sabirnica je standard za sabirnice u vozilima koji se koristi za komunikaciju između ECU-u (*Electronic Control Unit*). ECU-i su bilo koji dijelovi vozila s kojima se može programski upravljati, primjerice radio, zračni jastuci, senzori, aktuatori i slično. CAN protokol se temelji na porukama i napravljen je tako da ako više od jednog uređaja šalje poruku u isto vrijeme samo će se poslati poruka najvećeg prioriteta, a druge će čekati svoj red. Poruke se šalju u obliku podatkovnih okvira (eng. *data frame*).

PCAN-vrata (eng. *PCAN-Gateway*) omogućuju vezu između različitih CAN sabirnica preko IP mreža. Ovo se postiže slanjem CAN poruka preko TCP ili UDP protokola kroz mrežu od jednog uređaja do drugog. Format CAN poruke za mrežni prijenos definiran je u Tablica 2.

Tablica 2 Format CAN poruke

Duljina (bajtovi)	Ime polja	Opis	
2	Length	Fiksna vrijednost 0x24. Ovo je ekvivaletno dekadskom 36 i označava ukupnu vrijednost poruke u bajtovima (uključujući polje Length).	
2	Message type	Fiksna vrijednost 0x80. Označava CAN podatkovni okvir. (eng. *data frame*)	
8	Tag	Ne koristi se u trenutnoj verziji	
4	Timestamp Low	Vremenska oznaka CAN poruke u µs (mikrosekunde). Ova vrijednost nema utjecaja na prijenos podatkovnih okvira.	
4	Timestamp High		
1	Channel	Ne koristi se u trenutnoj verziji.	
1	DLC	Data Length Count (DLC) daje informaciju o veličini CAN Data polja.	
2	Flags	Ne koristi se u trenutnoj verziji.	
4	CAN ID	Bit 0 - 28	ID
		Bit 29	Fiksna vrijednost 0
		Bit 30	RTR
		Bit 31	1 za produženi okvir, 0 za standardni okvir.
8	CAN Data	Ovo polje je uvijek veličine 8x8 bitova, ali se koriste samo bitovi koji su obuhvaćeni s DLC poljem.	

Najvažnija polja su CAN ID i CAN Data. Zadatak je otvaranje i zatvaranje pumpe na električnoj čistilici, a za to je potreban CAN ID 0x18EFFF27. CAN Data ima 2 oblika ovisno o tome želi li se pumpa zatvoriti ili otvoriti. Poruka za otvaranje sadrži 0x041B01020127FFFF u svome Data polju, a poruka za zatvaranje 0x041B01060127FFFF. Obje poruke se šalju na isti CAN ID.

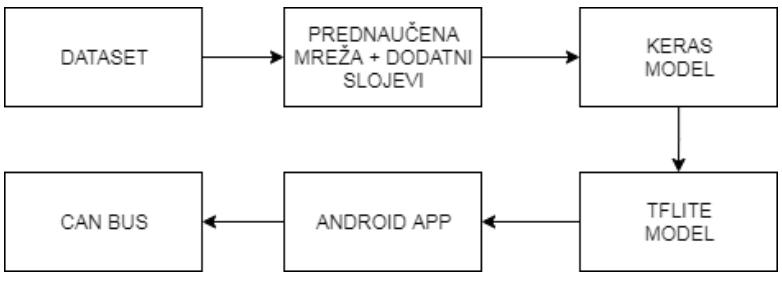
5. IMPLEMENTACIJA

U svrhe ovog rada uzeta je postojeća Android aplikacija koja uzima sliku s kamere mobilnog uređaja, šalje ju kroz naučeni model te vraća predviđanje modela za svaku sliku. Aplikacija je pisana u Javi te je razvijena s Android Studio alatom, dodatno je modificirana tako da radi s novim naučenim modelima. Nakon dohvaćanja predviđanja s modela aplikaciji je dodan element koji šalje poruke na CAN bus temeljene ne izlazu modela.

Modeli su izrađeni i naučeni u Keras okružju koje modele sprema u .h5 formatu. Keras je popularan Python paket za duboko učenje i služi kao omotač (eng. *wrapper*) za druge knjižnice dubokog učenja. U konkretnom slučaju koristi se Keras kao omotač oko Tensorflowa.

Format u kojem Keras sprema modele nije prikladan za mobilne uređaje jer oni imaju vrlo ograničenu računsku sposobnost i količinu memorije pa je bilo potrebno sve naučene modele pretvoriti u .tflite (tensorflow lite) oblik da bi se mogli pokretati u Tensorflow okružju na mobilnom uređaju. Tensorflow lite format smanjuje veličinu modela i ubrzava računanje njegovog izlaza bez prevelikog gubitka kvalitete modela.

Model uzima ulaznu sliku te na temelju nje zaključi nalazi li se na njoj objekt ili ne odnosno treba li otvoriti ili zatvoriti pumpu na električnoj čistilici. Izlaz modela je binarna klasifikacija odnosno broj koji govori kolika je vjerojatnost da se na slici nalazi objekt. Slika 17 prikazuje korake potrebne za ostvarivanje slanja relevantne poruke na CAN Bus krenuvši od podataka za učenje.



Slika 17 Tok podataka

5.1 Prednaučene mreže

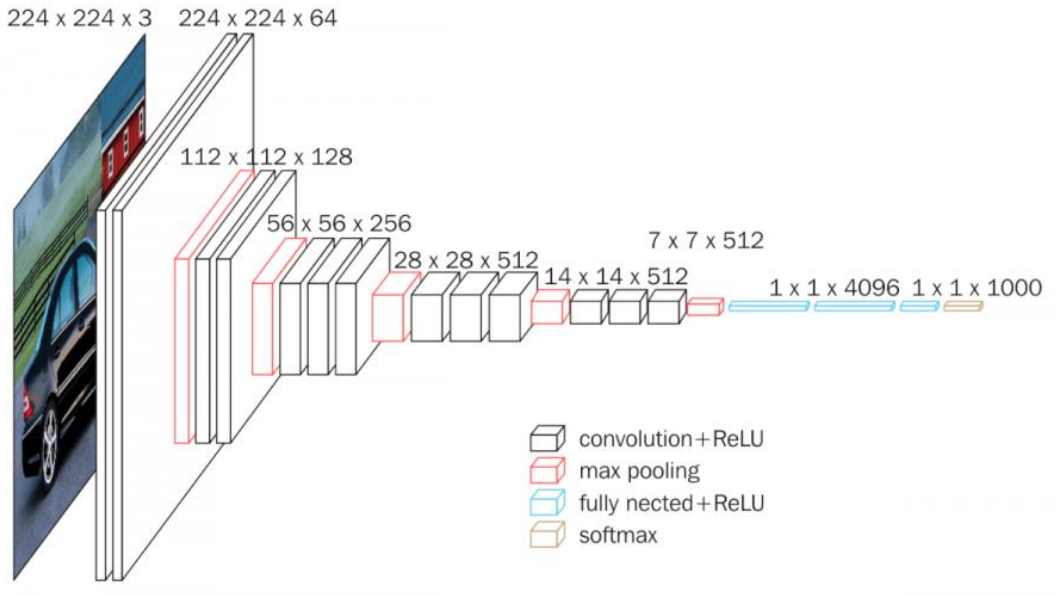
Odabrane su dvije prednaučene mreže: MobileNetV2 (Mark Sandler, 2018) i VGG19 (Karen Simonyan, 2015) koje će se iskoristiti za izdvajanje značajki iz ulaznih podataka.

MobileNetV2 mreža dizajnirana je za vizualno prepoznavanje objekata uključujući klasifikaciju, detekciju objekata i semantičku segmentaciju. Razvijena je od strane Google-a 2018. godine i s njom su uspjeli dobiti state-of-the-art performanse na mnogim natjecanjima. Napravljena je tako da ne zauzima previše memorije i računskih resursa i zbog toga je prikladna za postavljanje na mobilne uređaje. Arhitektura slojeva MobileNetV2 prikazana je u tablici 3.

Tablica 3 Arhitektura MobileNetV2 (Mark Sandler, 2018)

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

VGG19 je konvolucijska neuronska mreža koju su razvili K. Simonyan i A. Zisserman s Oxford sveučilišta. Mreža postiže 92.7% testnu točnost na ImageNet skupu podataka. Koristi male konvolucijske jezgre veličine 3x3 i njeno učenje je trajalo tjednima na Nvidia Titan Black grafičkim karticama. Arhitektura je prikazana na slici 18.



Slika 18 VGG19 Arhitektura (Frossard, 2017)

Ovi modeli su iskorišteni za prijenosno učenje tako da ih se koristi za izdvajanje značajki iz ulaznih slika. Odabrana su verzije modela učenih na Imagenet skupu podataka jer taj skup sadrži preko milijun slika te se od modela učenih na njemu mogu očekivati relativno dobre performanse čak i bez prevelike količine dodatnog učenja. Pošto se mreža koristi za izdvajanje značajki potrebno je izbaciti zadnji sloj veličina 1000x1 ili 1280x1 koji odgovara klasama na Imagenet skupu. Nakon izbacivanja zadnjeg sloja potrebno je sve ostale smrznuti odnosno spriječiti ažuriranje njihovih parametara. Keras u sebi sadrži gotovu implementaciju MobileNetV2 i VGG19 modela. Sljedeći kod prikazuje učitavanje MobileNetV2 mreže.

```
def get_mobile_net_model(include_top=False):
    base_model =  MobileNetV2(input_shape=(IMAGE_SIZE, IMAGE_SIZE, NUM_CHANNELS),
                              include_top=include_top,
                              weights="imagenet")
    for layer in base_model.layers:
        layer.trainable = False

    return base_model
```

Za obje mreže ulazna slika mora biti veličine 224x224x3 piksela te se sve ulazne slike smanjuju ili povećavaju da bi bile točno ove veličine. Svi slojevi se smrzavaju odnosno njihove težine se ne uče.

5.2 Učenje mreža i priprema podataka

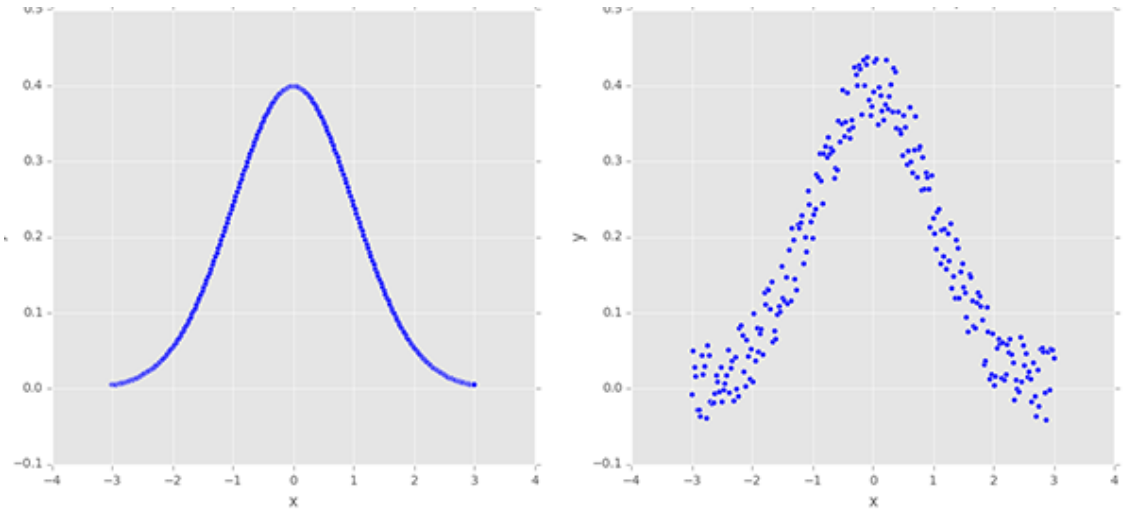
Prednaučene mreže će se koristiti za izdvajanje značajki iz ulaznih podataka. Ovo znači da je trenutno izlaz mreže zapravo izlaz iz predzadnjeg sloja mreže, a ne vjerojatnost pripadanja ulaznog podatka klasi jer je maknut posljednji sloj originalnih modela. Zbog ovoga potrebno je na kraj mreže nadodati nekoliko FC slojeva te izlazni sloj. Ovi dodatni slojevi će biti oni koji će se učiti. Broj slojeva u ovom slučaju je proizvoljan, ali važno je odabrati dobar broj slojeva i neurona unutar tih slojeva da bi se postigli kvalitetni rezultati. Potrebno je odabrati i funkciju gubitka te optimizator za model. U primjeru koda dodan je flatten sloj te 2 FC sloja veličine 256 na vrh MobileNetV2 modela. Pošto je u pitanju binarna klasifikacija zadnji izlazni sloj mora imati veličinu 1. Aktivacijske funkcije su RelU jer su one korištene i kod originalnih mreža.

```
def build_model():
    model = keras.models.Sequential()
    model.add(get_mobile_net_model())

    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(Dense(256, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss=LOSS,
                  optimizer=OPTIMIZER,
                  metrics=MODEL_METRICS)
    return model
```

Prije početka učenja modela potrebno je pripremiti podatke za učenje kako bi bili postignuti što moguće bolji rezultati. Priprema se podataka vrši tehnikama povećanja podataka (eng. data augmentation). Cilj povećavanja podataka je povećanje sposobnosti generaliziranja modela. Povećanje podataka je zapravo mijenjanje (augmentacija) podataka na razne načine kako bi se stvorio veći skup za učenje. Mreža može naučiti robustnije značajke ako uvijek viđa nove, malo promijenjene podatke umjesto stalno iste. Kod testiranja modela može se koristiti augmentacija podataka ali se u većini slučajeva model testira na originalnim neizmijenjenim podacima. Posljedica promjene podataka je najčešće gubitak točnosti kod učenja, ali povećanje točnosti kod testiranja.



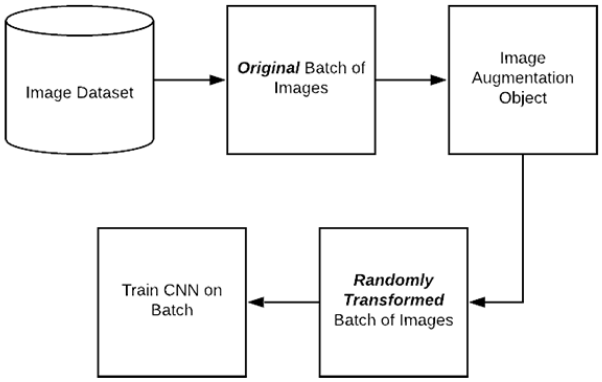
Slika 19 Normalna distribucija (lijevo) s dodanim šumom (desno) (Rosebrock, 2019)

Slika 19 (lijevo) prikazuje normalnu distribuciju podataka s varijancom i očekivanjem jednakim nula. Učenje modela na ovakvoj raspodjeli bi moglo rezultirati s modeliranjem savršeno točne normalne distribucije međutim u većini slučajeva podaci se nikad neće nalaziti na ovakvoj krivulji. Modificiranjem krivulje s malim odstupanjima (Slika 19)

postiže se aproksimacija normalne distribucije koja nije savršena. Svakoj točki u krivulji se doda neka mala vrijednost te učenje modela na ovakvoj distribuciji ima veće šanse da bolje generalizira nad podacima koji nisu bili u skupu za učenje. Ovim postupkom povećava se otpornost modela na šum.

Ulazni su podaci u slučaju ovog rada isključivo slike, a one se mogu augmentirati na više načina: translacijom, rotacijom, promjenom razmjera, rezanjem, vertikalnim ili horizontalnim zrcaljenjem i slično. Primjenom bilo koje od ovih metoda u malom iznosu na slici neće promijeniti njenu klasu, ali će malo promijeniti njen izgled. Augmentacija slikovnih podataka je zbog ovog lagana metoda za primjenu na zadatke računalnog vida. Postoje 3 glavna načina korištenja augmentacije podataka: generiranje podataka, mijenjanje podataka na mjestu te kombinacija prva 2 načina.

Generiranje podataka (proširenje postojećeg skupa podataka) je način koji se koristi ako je na raspolaganju jako malen broj podataka za učenje. Ne koristi se često jer se ovim načinom ne povećava sposobnost modela da generalizira jer je osnovni broj izvornih podataka još uvijek jako malen, a svi ostali su generirani u odnosu na njih.



Slika 20 Mijenjanje podataka na mjestu (Rosebrock, 2019)

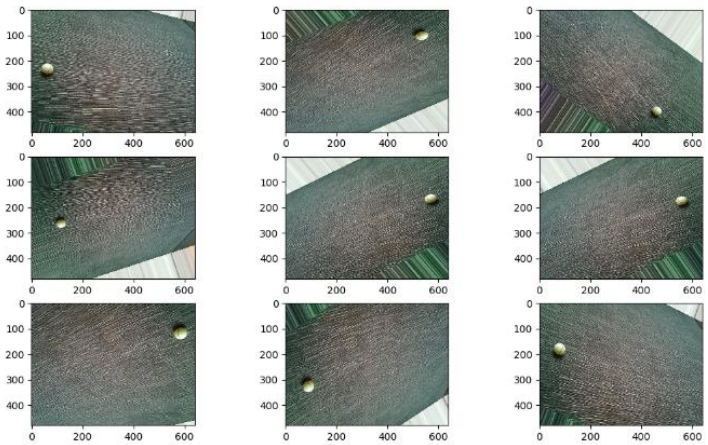
Mijenjanje podataka na mjestu koristi se kad je željeno ponašanje da mreža vidi drugačije podatke u svakoj epohi učenja. Slika 20 prikazuje proces mijenjanja podataka na mjestu tijekom učenja mreže. Prvo se uzimaju ulazni podaci iz skupa za učenje koji se zatim transformiraju na prijašnje opisane načine te se onda samo ti novi podaci šalju kroz mrežu. Razlog zašto se originalni podaci ne šalju već samo generirani je taj što se želi osigurati da mreža vidi potpuno nove podatke u svakoj epohi učenja kako bi mreža stalno viđala „nove“ podatke te kako bi se povećala generalizacijska sposobnost učenog modela. Kad bi se originalni podaci slali izmiješani s novima onda bi mreža više puta viđala te iste podatke i izgubio bi se smisao augmentacije.

Treća se vrsta augmentacije sastoji od kombiniranja generiranja podataka i mijenjanje podataka na mjestu. Često se koristi u kontekstu kloniranja ponašanja.

Predprocesiranje slikovnih podataka je odrađeno s Keras funkcijom ImageDataGenerator. Funkcija osim podataka prima više parametara koji na različite načine manipuliraju sa slikama. Rotation range parametar određuje koliki je maksimalni kut za koji se slika smije rotirati, a funkcija će onda nasumičnu rotaciju primijeniti na svaku od ulaznih slika. Width shift range i height shift range pomiču sliku u horizontalnom i vertikalnom smjeru za neku postotnu vrijednost. Shear range određuje smjer kuta u kojem će se slika izrezati po sredini, a zoom range određuje koliko će se u sliku zumirati. Rescale određuje operaciju koja će se izvesti na svim ulaznim podacima. U slučaju slika svaka vrijednost piksela se dijeli s 255 tako da sve vrijednosti budu na rasponu [0, 1]. Generiranje izmijenjenih podataka za učenje odrađeno je u funkciji get_train_data.

```
def get_train_data():
    train_datagen = ImageDataGenerator(
        preprocessing_function=preprocess_input,
        rotation_range=40, rescale=1./255.,
        horizontal_flip=True, vertical_flip=True,
        fill_mode='nearest')
    train_generator = train_datagen.flow_from_directory(
        TRAIN_DIR,
        target_size=(IMAGE_SIZE, IMAGE_SIZE),
        batch_size=BATCH_SIZE, class_mode=CLASS_MODE)
    return train_generator
```

Model se uči nad izmijenjenim podacima koji mu se šalju kroz uz pomoć ImageDataGenerator objekata. Slika 21 pokazuje izmijenjene podatke generirane iz jedne slike na kojoj se nalazi teniska loptica.



Slika 21 Izmijenjeni podaci

5.3 Spremanje i učitavanje modela

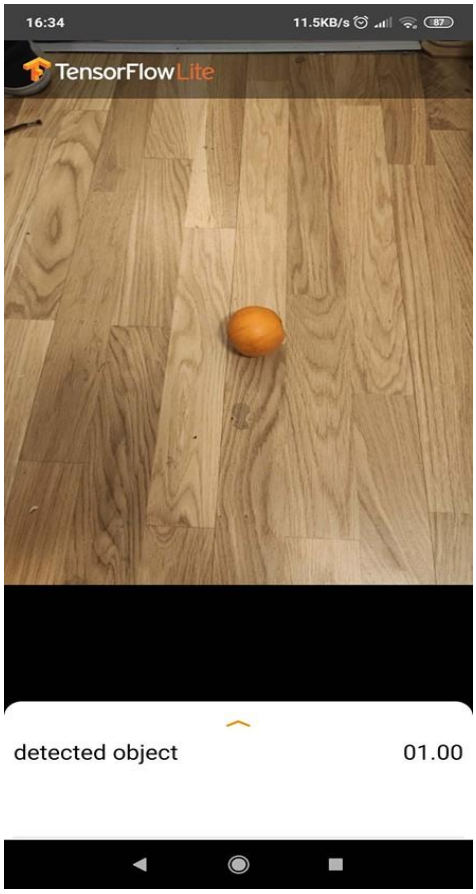
Nakon učenja modela on se sprema u .h5 obliku na disk.

Najpopularnije okružje za duboko učenje na Android uređajima je Tensorflow Lite. Tensorflow Lite ne podržava direktno .h5 modele pa je potrebno naučeni model pretvoriti u oblik koji je prikladniji ovom okružju. Korištenjem Tensorflowa .h5 model se može pretvoriti u .tflite model. Ovaj postupak smanjuje veličinu modela, a samim tim i smanjuje količinu računanja da bi se dobio rezultat iz modela odnosno povećava njegovu brzinu. Konverzija modela obično ima jako malen ili nikakav utjecaj na preciznost modela. (Juhyun Lee, 2019)

```
converter = tf.lite.TFLiteConverter.from_keras_model_file('keras_model.h5')
tfmodel = converter.convert()
file = open("model.tflite" , "wb").write(tfmodel)
file.write(model)
```

Unutar Android aplikacije nalazi se .tflite model koji se zajedno s aplikacijom instalira na uređaj. Za vrijeme rada aplikacije kamera uređaja je cijelo vrijeme upaljena te se dobivaju predviđanja za dolazeće slike u realnom vremenu.

Svaka slika se predprocesira tako da joj se mijenja veličina u onu odgovarajuću koju model prihvaća. Model ulaznu sliku učitava te radi predviđanje nad njom koje vraća u obliku liste veličine 1. Vraćena vrijednost je vjerojatnost pripadanja slike prvoj klasi ili nevjerojatnost pripadanja slike drugoj klasi. Ovo se događa više puta u sekundi, a točan broj ovisi o jačini uređaja na kojem je aplikacija instalirana. Iako je video niz slijednih podataka model nema mogućnost gledanja prijašnjih ili kasnijih slika već za svaku sliku neovisno radi novo predviđanje. Aplikacija je prikazana na slici 21.



Slika 22 Android aplikacija

5.4 CAN Bus komunikacija

Slanje poruka na CAN bus odrađeno je preko PCAN Wireless Gateway-a koji u ovom slučaju predstavlja električnu čistilicu. Prijenos poruka se radi uz pomoć TCP paketa formatiranih na prije opisan način. PCAN Gateway ima svoju IP adresu te je potrebno u njegovom administratorskom sučelju napraviti rutu s određenim vratima (eng. *port*) tako da uređaj zna na koja vrata može slati CAN poruke. Slanje poruka odrađeno je u MessageSender klasi koja se poziva nakon što se dobije predviđanje s modela.

```
Recognition recognition = results.get(0);
if (recognition != null) {
  Float confidence = recognition.getConfidence();
  if (confidence != null) {
    MessageSender messageSender = new MessageSender();
    messageSender.execute(confidence);
  }
}
```

MessageSender klasa prima broj koji predstavlja postotak pripadanja klasi te na temelju njega odlučuje koju će poruku poslati – OPEN ili CLOSE. Također uzima u obzir zadnju poslanu poruku te šalje nove poruke samo u slučaju da je drugačija od zadnje poslane. Ovo sprječava nepotrebno ponovno slanje istih poruka koje ne bi imale nikakvog utjecaja na pumpu. Ako poruka zadovoljava sve uvjete potrebne da bi se poslala onda se šalje preko TCP Mreže na PCAN Wireless Gateway u obliku Java byte-stream-a na predkonfiguriranu IP adresu i vrata. Mobilni uređaj je potrebno spojiti na bežičnu mrežu od PCAN Gateway-a da bi se poruke mogle slati te nije potrebna nikakva povezanost prema internetu.

```
protected Void doInBackground(Float... voids) {
    Float confidence = voids[0];
    MESSAGE_TYPE message_type = null;
    if (confidence != null) {
          if (confidence <= 0.5f) {
              if (last_message_sent != MESSAGE_TYPE.OPEN) {
                  message_type = MESSAGE_TYPE.OPEN;
                  last_message_sent = MESSAGE_TYPE.OPEN;
              }
          } else {
              if (last_message_sent != MESSAGE_TYPE.CLOSE) {
                  message_type = MESSAGE_TYPE.CLOSE;
                  last_message_sent = MESSAGE_TYPE.CLOSE;
              }
          }
    }
    if (message_type == null) {
      return null;
    }
    byte[] message = get_message_byte_array(message_type);
    if (message == null) {
      return null;
    }
    try {
      Socket s = new Socket("192.168.1.10", 7000);
      DataOutputStream dataOutputStream =
        new DataOutputStream(s.getOutputStream());
      dataOutputStream.write(message);
      dataOutputStream.flush();
      s.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
    return null;
}
```

6. REZULTATI

Simulacijski skup za učenje sastoji se od slika na kojima se nalazi ili ne nalazi teniska loptica. Skup predstavlja simulirano okružje detekcije objekata ispred kamere električne čistilice. Preuzet je sa Kaggle web stranice.[2] Teniska loptica na slici treba predstaviti objekt za koji bi se pumpa na električnoj čistilici trebala otvoriti, a ako na slici nema ničega pumpa bi se trebala zatvoriti. Slika 23 prikazuje pozitivni primjer sa lopticom i negativni primjer bez nje.

Slika 23 Primjeri iz skupa za učenje

U svrhe dobivanja raznovrsnih rezultata zbog boljeg uvida u podatke i sam zadatak učeno je više modela koji se razlikuju po raznim kriterijima. Uzete su u obzir dvije prednaučene mreže: MobileNetV2 te VGG19. Nakon toga su na njihov vrh dodane dvije različite arhitekture koje se sastoje od nekoliko FC slojeva: 256x256x1 te 256x256x256x256x1. Mreže su učene s različitim optimizatorima te različitim stopama za učenje.

Učenje svih mreža provedeno je u 20 epoha. Sve mreže imaju isti izlazni sloj jer sve rade binarnu klasifikaciju ulaznih podataka. Korištene su RelU aktivacijske funkcije za skrivene slojeve, a sigmoid za izlazni sloj. Mreže su učene s tri različite stope učenja: 0.1, 0.01 i 0.001.

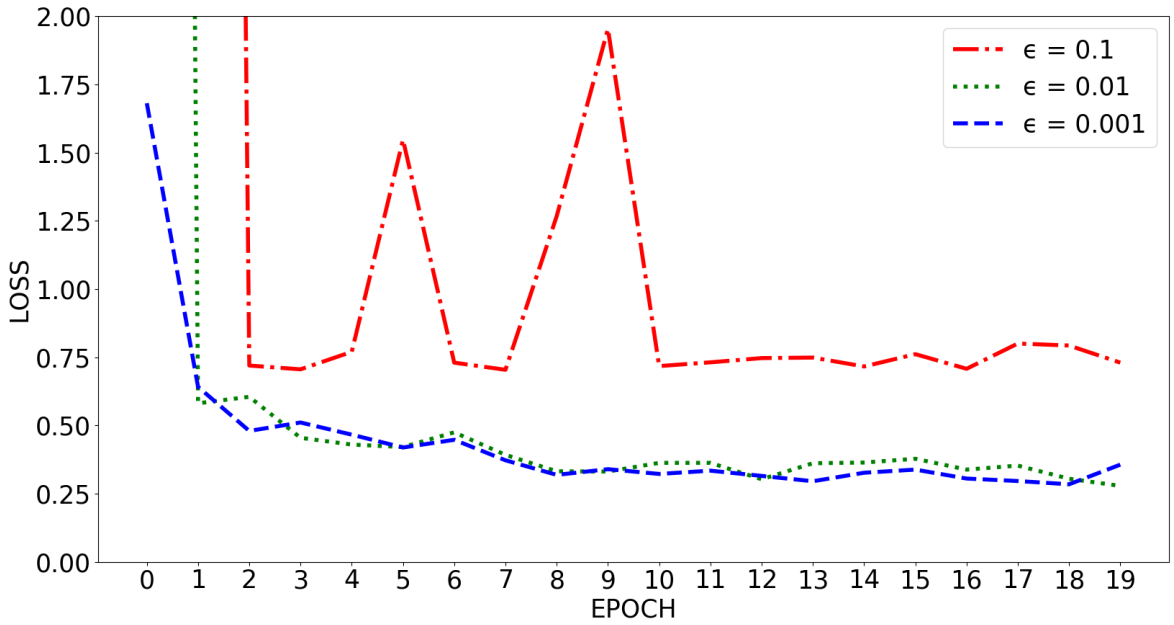
Prva arhitektura sastoji se od dodavanja 2 FC sloja veličina 256 na vrh prednaučenih mreža te od izlaznog sloja veličine 1. Tablica 4 prikazuje rezultate učenja i testiranja za ove modele. Dva korištena optimizatora su Adam i standardni gradijenti spust s momentom.

[2] https://www.kaggle.com/domhenjes/ballsemtpytt

Tablica 4 Rezultati arhitekture 256x256x1

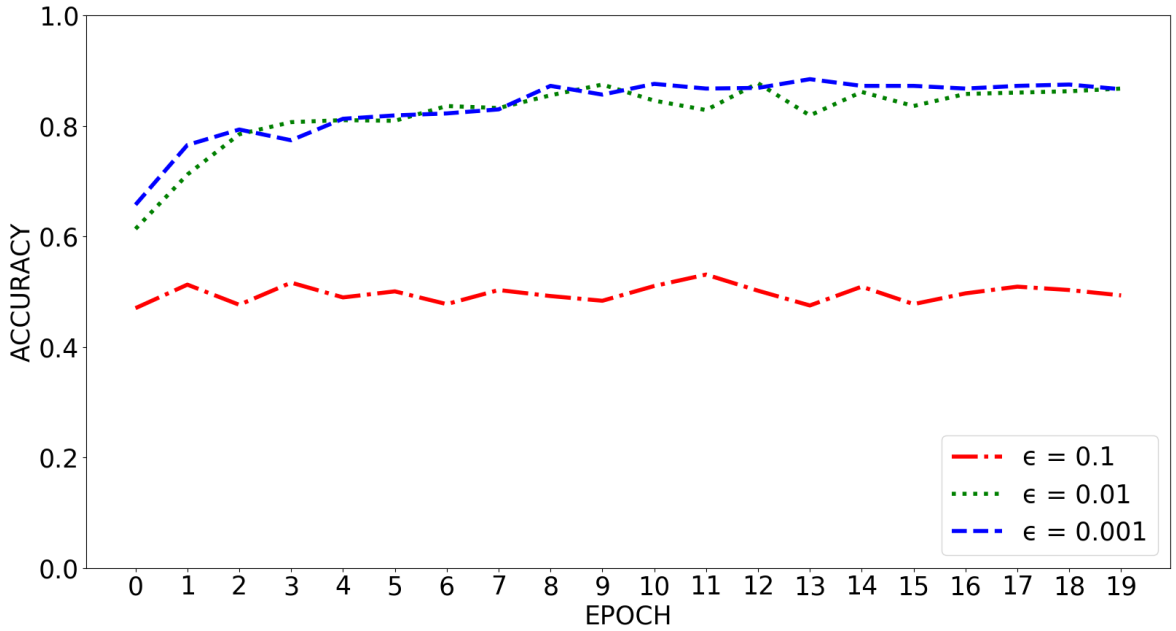
Model	ϵ	Optimizator (optimizer)	Gubitak (loss)		Točnost (accuracy)		Preciznost (precision)		Odziv (recall)	
			Train	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
MobileNetV2	0.1	**ADAM**	0.730	0.728	0.493	0.50	0.50	0.00	0.78	0.00
MobileNetV2	0.01		0.278	0.80	0.867	0.712	0.871	0.869	0.88	0.350
MobileNetV2	0.001		0.356	0.451	0.866	0.812	0.852	0.777	0.956	1.00
VGG19	0.1		0.697	0.687	0.501	0.5	0.507	0.5	0.678	0.00
VGG19	0.01		0.250	51.153	0.884	0.512	0.865	1.0	0.994	0.012
VGG19	0.001		0.26	50.302	0.894	0.512	0.880	1.0	0.98	0.012
			Train	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
MobileNetV2	0.1	**SGD**	0.695	0.696	0.496	0.5	0.503	0.0	0.685	0.00
MobileNetV2	0.01		0.280	0.520	0.890	0.762	0.862	0.838	0.94	0.530
MobileNetV2	0.001		0.279	0.303	0.879	0.824	0.876	0.824	0.92	0.825
VGG19	0.1		0.698	89.20	0.488	0.487	0.495	0.481	0.592	0.245
VGG19	0.01		0.693	0.881	0.506	0.524	0.506	0.519	0.126	0.964
VGG19	0.001		0.320	4.969	0.866	0.612	0.848	1.0	0.979	0.126

Iz tablice 4 vidi se da je najbolje rezultate s Adam optimizatorom postigao MobileNetV2 model sa stopom učenja od 0.001. Stopa učenja od 0.1 se pokazala kao prevelikom za sve vrste modela te je rezultirala s najgorim rezultatima poput preciznosti i točnosti od 50% što pokazuje da je model jednako dobar kao nasumičan odabir klase za ulazni primjer. U nekim slučajevima se dogodi je preciznost 1.0 kod testiranja modela međutim to samo govori da model nema niti jedan FP primjer odnosno da je za sve primjere predvidio da ne sadrže tenisku lopticu. VGG19 se pokazala kao neprikladna mreža ako se koristi Adam optimizator te s niti jednom stopom učenja nije uspjela dobiti zadovoljavajuću točnost i preciznost kod testiranja.



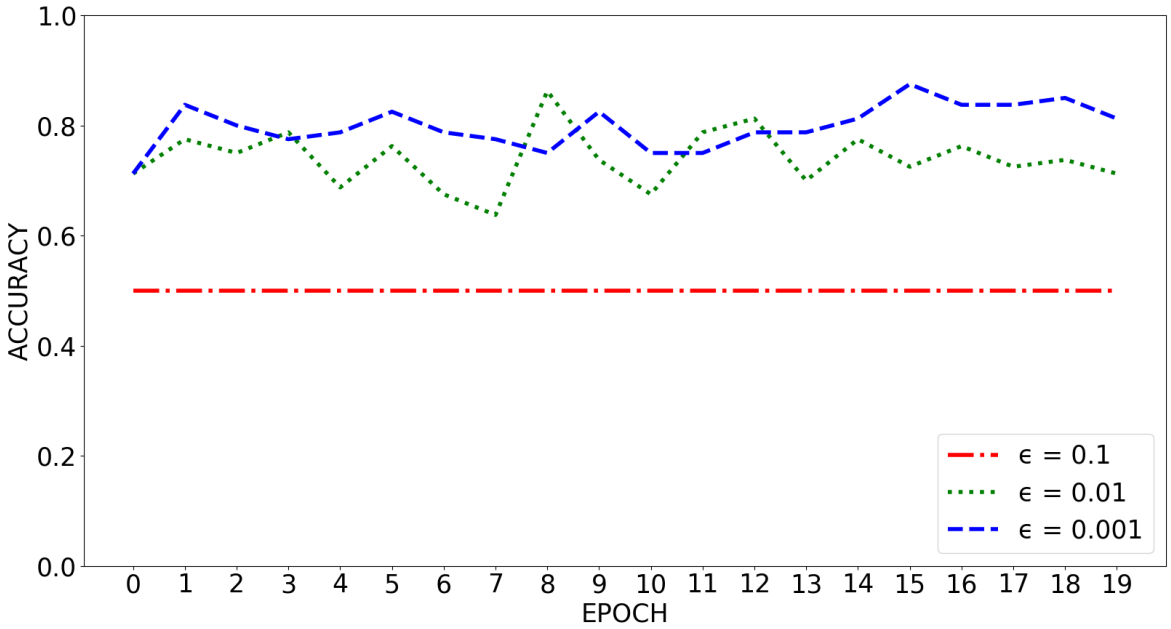
Slika 24 Gubitak učenja za model MobilenetV2 s Adam optimizatorom

Slika 24 pokazuje promjenu gubitka kod učenja MobileNetV2 mreže uz Adam optimizator. Iz slike se vrlo dobro vidi utjecaj prevelike stope učenja na gubitak mreže iz epohe u epohu. Gubitak je već nakon druge epohe došao na vrijednost sličnu onoj koju je imao nakon 20 epoha, a između se vide ogromni skokovi u gubitku koji ukazuju na veliku nestabilnost modela. Za manje stope učenja se vidi puno normalnija krivulja koja se polako smanjuje te se stabilizira oko devete epohe. Ova brza stabilizacija te daljnje neznatno smanjivanje gubitka je pokazatelj da model ili nema dovoljno podataka ili da je mreža prejednostavna. Modeli sa stopama od 0.01 i 0.001 su u ovom slučaju imali slične performanse te su oba završila učenje sa sličnim iznosom gubitka.



Slika 25 Točnost učenja za model MobiletnetV2 s Adam optimizatorom

Na slici 25 je prikazana točnost modela tijekom učenja. Model s prevelikom stopom učenja se tijekom 20 epoha nije pomaknuo sa početne točke odnosno uvijek se vrtio oko 50% što znači da model ništa nije naučio iz podataka te samo nasumično predviđa klasu. Slično kao i kod gubitka točnost se stabilizirala oko devete epohe za modele sa manjim stopama učenja.



Slika 26 Točnost validiranja za model MobiletnetV2 s Adam optimizatorom

Slika 26 prikazuje točnost modela kod validiranja. Jednako kao i u prijašnjim grafovima model s prevelikom stopom učenja se nije maknuo od 50% točnosti nakon 20 epoha. Model sa srednjom veličinom stope učenja je završio sa malo gorim rezultatima od onog s

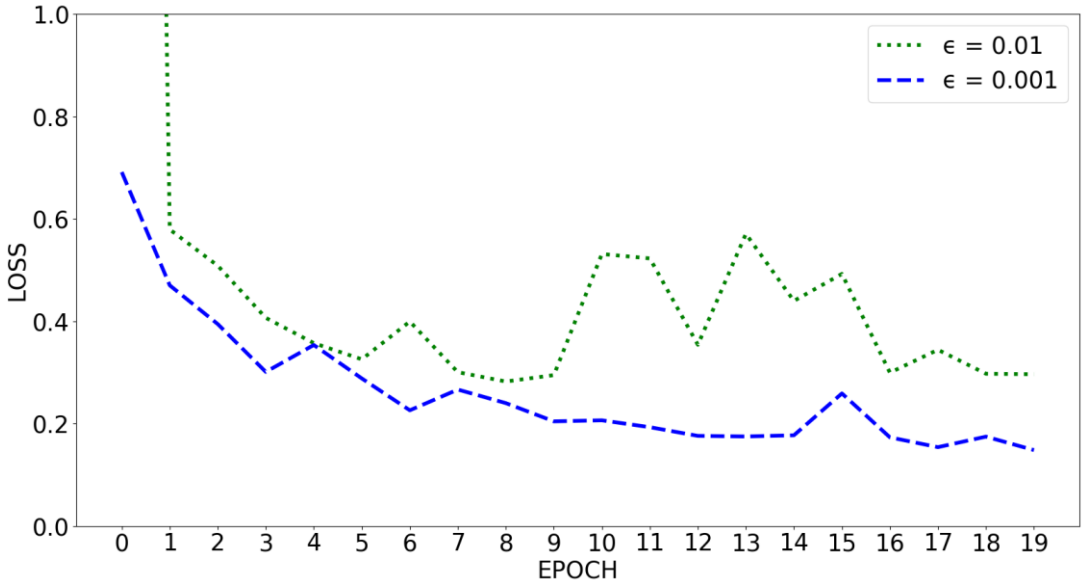
najmanjom, ali je u oba slučaja točnost modela ostala slična početnoj. Validirajući graf je još jedan pokazatelj da ili nije dovoljan broj podataka ili da model nije dovoljno kompleksan da nauči korisne značajke iz podataka.

Iz prve arhitekture postalo je očito da je stopa učenja od 0.1 prevelika te se za sljedeću arhitekturu testira samo za dvije preostale stope: 0.01 i 0.001. Druga arhitektura nad kojom je model učen se sastoji od 4 sloja veličine 256 odnosno vrh mreže je oblika 256x256x256x256x1. Ovim se povećala kompleksnost i dubina modela. Tablica 5 prikazuje rezultate nove arhitekture. Najbolje rezultate ovaj put je postigla VGG19 mreža koristeći Adam optimizator sa stopom učenja od 0.001 s točnosti od skoro 90%. Povećanje kompleksnosti modela je generalno poboljšalo rezultate u odnosu na manju arhitekturu. Adam optimizator se općenito pokazao kao bolji od standardnog gradijentnog spusta s momentom te je u oba slučaja najbolji model bio onaj koji je koristio Adam.

Tablica 5 Rezultati arhitekture 256x256x256x256x1

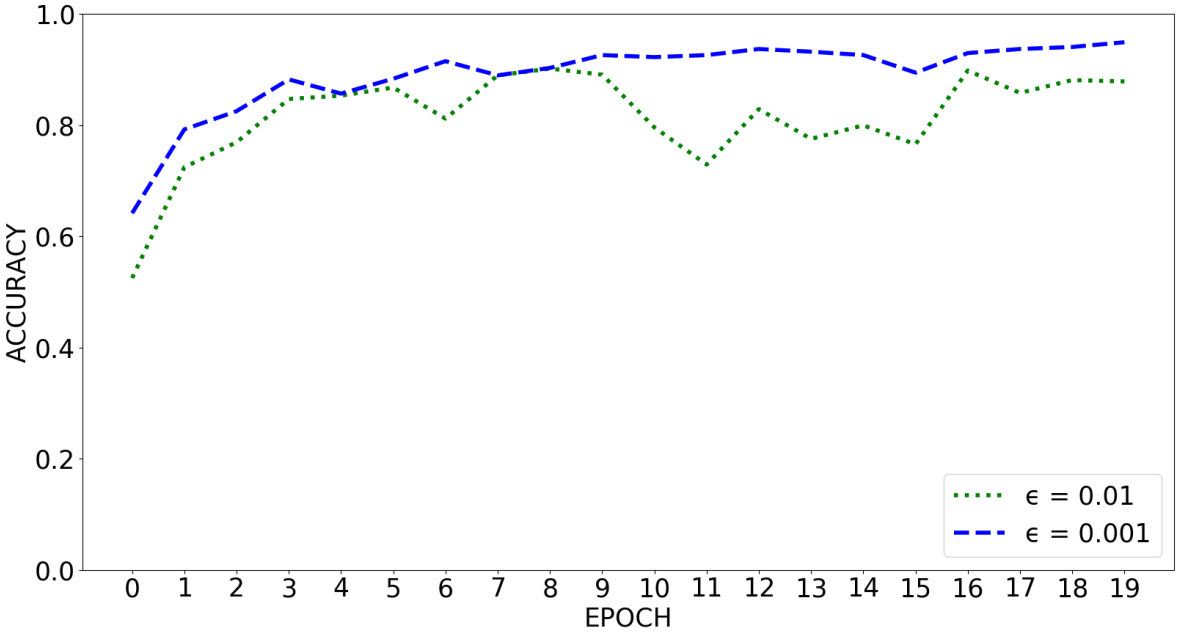
Model	ϵ	**Optimizator (optimizer)**	**Gubitak (loss)**		**Točnost (accuracy)**		**Preciznost (precision)**		**Recall (odziv)**	
			Train	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
MobileNetV2	0.01	**ADAM**	0.400	0.514	0.863	0.812	0.822	0.770	0.941	1.00
MobileNetV2	0.001		0.196	0.234	0.927	0.787	0.923	0.767	0.967	0.891
VGG19	0.01		0.296	0.089	0.878	0.887	0.845	0.829	0.945	0.914
VGG19	0.001		0.148	0.08	0.948	0.899	0.935	0.970	0.983	0.917
			Train	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
MobileNetV2	0.01	**SGD**	0.203	0.544	0.916	0.824	0.914	0.782	0.952	0.714
MobileNetV2	0.001		0.219	0.668	0.921	0.762	0.929	0.684	0.922	0.846
VGG19	0.01		0.694	0.694	0.479	0.5	0.489	0.0	0.909	0.00
VGG19	0.001		0.383	0.531	0.826	0.837	0.805	0.764	0.962	0.702

Slika 27 prikazuje gubitak kod učenja najbolje rezultirajućeg modela s dvije stope učenja. Iz slike se vidi da se gubitak cijelo vrijeme smanjuje za stopu od 0.001 i da više ne staje sa smanjivanjem oko devete epohe kako se dogodilo u prijašnjoj arhitekturi. Također se vidi da je manja stopa bolja i stabilnija od veće.



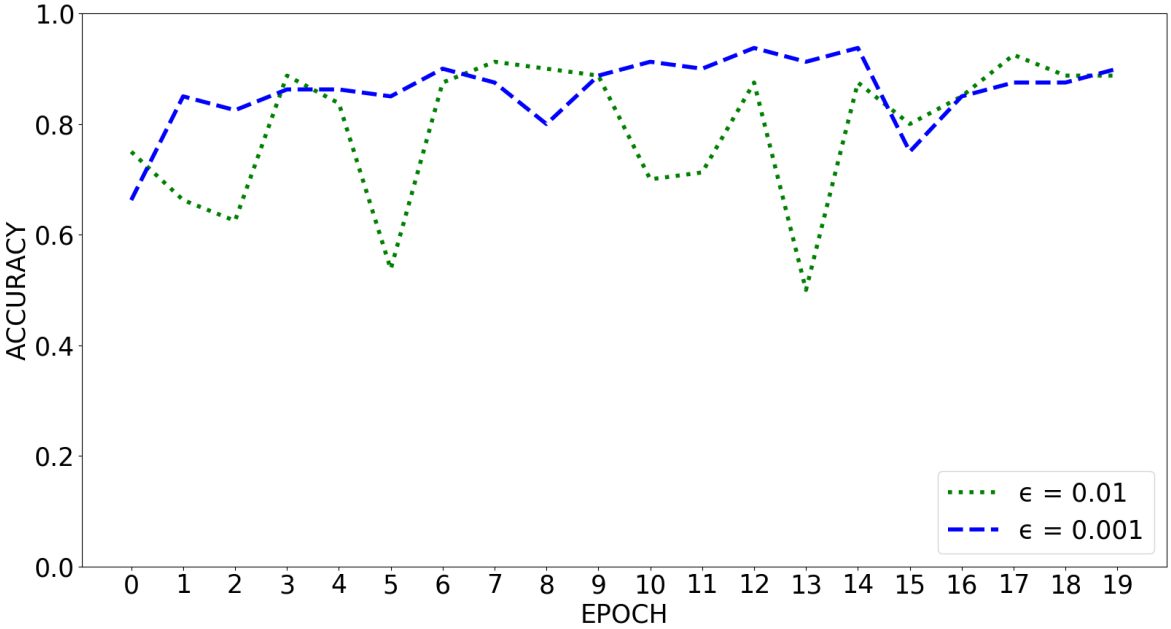
Slika 27 Gubitak VGG19 modela s Adam optimizatorom

Na slici 28 prikazana je točnost modela tijekom učenja mreže. Iz slike se vidi da je složenija arhitektura postigla da preciznost stabilno raste za manju stopu učenja.



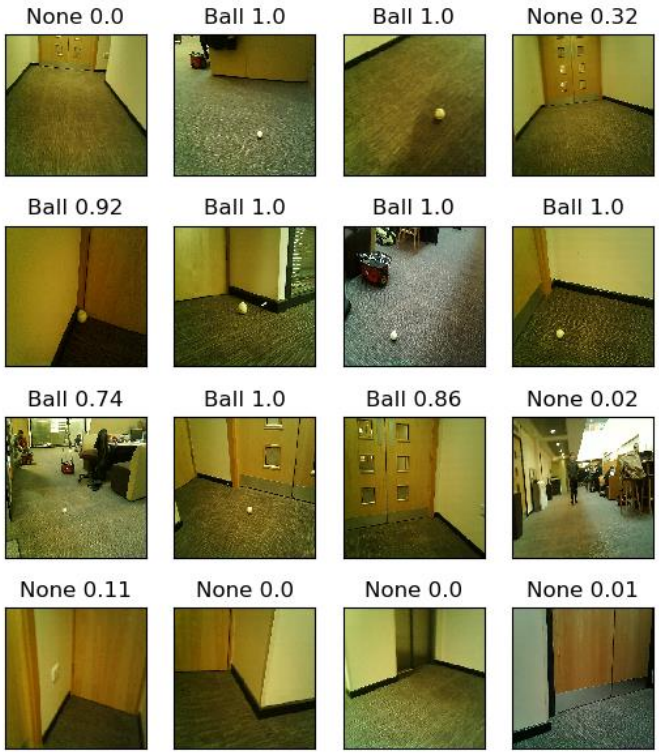
Slika 28 Točnost učenja VGG19 modela s Adam optimizatorom

Najveće poboljšanje u odnosu na manje složenu arhitekturu se očituje na slici 29 koja prikazuje točnost kod validiranja sa stabilnim rastom točnosti modela koja u niti jednoj epohi nije pala na vrijednost manje od početne u slučaju manje stope učenja.



Slika 29 Točnost validiranja VGG19 modela s Adam optimizatorom

Slika 30 prikazuje predviđanja modela nad skupom za testiranje za 16 slika. Brojke pored oznake označavaju vjerojatnost da se na slici nalazi teniska loptica. Rezultati prikazuju da je model uspješno naučio razlikovati slike sa teniskom lopticom i one bez nje.



Slika 30 Rezultati VGG19 modela

7. ZAKLJUČAK

Rezultati analiziranja performansi modela su pokazali da se može napraviti uspješna detekcija objekta uz korištenje prijenosnog učenja s relativno malenim skupom za učenje.

Ovako naučeni modeli mogu se iskoristiti za automatsko otvaranje i zatvaranje usisnog otvora na električnoj čistilici, ali da bi to bilo moguće potrebno je prikupiti velike količine vizualnih podataka s čistilice te ih točno označiti. U simulacijskom okružju bio je samo jedan objekt za detekciju (teniska loptica). Međutim, u stvarnosti bilo kakva stvar može dospjeti ispred kamere tako da je potrebno imati raznovrsne podatke kako bi model pravilno naučio razliku između pozadine i objekta kojeg treba usisati.

Dodatno, rezultati bi se mogli poboljšati korištenjem povratnih neuronskih mreža (eng. *recurrent neural networks*) jer se one pokazale kao dobre u učenju iz slijednih podataka. Trenutni naučeni modeli ne iskorištavaju slijednost snimki već rade predviđanje posebno za svaku sliku. Moguće je da bi povratne neuronske mreže mogle postići bolje rezultate od unaprijednih.

Kad bi se koristio pravi skup za učenje, a ne simulacijski skup, vjerojatno bi bilo potrebno napraviti klasifikaciju koja nije binarna jer u stvarnim uvjetima postoje stvari na cesti za koje se usisni otvor ne bi trebao otvarati (primjerice šahtovi, pješački prijelazi ili bilo koje druge oznake na cestama). Dodatno, trebalo bi uzeti u obzir i razne vremenske uvjete te bi trebalo prikupiti podatke iz raznih godišnjih doba.

LITERATURA

Anderson Monica Why Deep Learning Works [Mrežno] // Artificial Understanding. - 24. veljače 2018. - 18. siječnja 2020. - https://artificial-understanding.com/why-deep-learning-works-1b0184686af6.

Bhande Anup What is underfitting and overfitting in machine learning and how to deal with it. [Mrežno] // Medium. - 11. ožujka 2018. - 17. siječnja 2020. - https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76.

Bhattarai Saugat What is gradient descent in machine learning? [Mrežno] // Saugat Bhattarai A Tech Blog. - 22. lipnja 2019. - 17. siječnja 2020. - https://saugatbhattarai.com.np/what-is-gradient-descent-in-machine-learning/.

Chauhan Nagesh Singh Introduction to Artificial Neural Networks(ANN) [Mrežno] // TowardsDataScience. - 3. kolovoza 2019. - 17. siječnja 2019. - https://towardsdatascience.com/introduction-to-artificial-neural-networks-ann-1aea15775ef9.

Dabbura Imad Gradient Descent Algorithm and Its Variants [Mrežno] // Towards Data Science. - 21. prosinca 2017. - 17. siječnja 2020. - https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3.

Frossard Davi VGG in TensorFlow [Mrežno] // University of Toronto Computer Science. - 17. svibnja 2017. - 17. siječnja 2020. - https://www.cs.toronto.edu/~frossard/post/vgg16/.

Gupta Dishashree Transfer learning and the art of using Pre-trained Models in Deep Learning [Mrežno] // Machine Learning Mastery. - Analytics Vidhya, 1. srpnja 2017. - 17. siječnja 2020. - https://www.analyticsvidhya.com/blog/2017/06/transfer-learning-the-art-of-fine-tuning-a-pre-trained-model/.

Ian Goodfellow Yoshua Bengio, Aaron Courville Deep Learning [Knjiga]. - [s.l.] : MIT Press, 2016.

Juhyun Lee Nikolay Chirkov, Ekaterina Ignasheva, Yury Pisarchyk, Mogan Shieh, Fabio Riccardi, Raman Sarokin, Andrei Kulik, Matthias Grundmann On-Device Neural Net Inference with Mobile GPUs [Konferencija] // Computer Vision and Pattern Recognition Workshop. - 2019.

Karen Simonyan Andrew Zisserman Very Deep Convolutional Networks for Large-Scale Image Recognition [Konferencija]. - 2015.

Mark Sandler Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen MobileNetV2: Inverted Residuals and Linear Bottlenecks [Časopis] // The IEEE Conference on Computer Vision and Pattern Recognition (CVPR). - 2018.

Mhaskar H. Liao Q. Poggio T. When and Why Are Deep Networks Better than Shallow Ones? [Mrežno] // Semantic Scholar. - 2017. - 2019. siječnja 17. - https://pdfs.semanticscholar.org/f594/f693903e1507c33670b89612410f823012dd.pdf.

MissingLink.ai 7 Types of Neural Network Activation Functions: How to Choose? [Mrežno] // Missing Link AI. - 2019. - 17. siječnja 2019. - https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/.

Nielsen Michael Neural Networks and Deep Learning [Mrežno] // Neural Networks and Deep Learning. - prosinac 2019. - 2019. siječnja 17. - http://neuralnetworksanddeeplearning.com.

Nitish Shirish Keskar Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, Ping Tak Peter Tang On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima [Konferencija] // ICLR 2017. - 2016.

Rizwan Muhammad Mini-batch Gradient Descent for Deep Learning [Mrežno]. - 9. travnja 2018. - 18. siječnja 2020. - https://engmrk.com/mini-batch-gd/.

Rosebrock Adrian Keras ImageDataGenerator and Data Augmentation [Mrežno] // PyImageSearch. - 8. lipnja 2019. - 17. siječnja 2020. - https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/.

Stanford University Convolutional Neural Networks [Mrežno] // http://cs231n.github.io/. - 17. siječnja 2020. - http://cs231n.github.io/convolutional-networks/.

PRILOZI

Sav kod za učenje mreža napravljen je sa verzijom Pythona 3.7.4. Korišten je Keras 2.3.1 verzija sa tensorflow-gpu 2.0.0 backendom. Za vršenje učenja na grafičkoj kartici korištena je CUDA 10.0 dostupna na https://developer.nvidia.com/cuda-10.0-download-archive.

Skup za učenje sa teniskim lopticama preuzet je sa kaggle web stranice na slijedećem linku: https://www.kaggle.com/domhenjes/ballsemtpytt

Dokumentacija za PCAN-Wireless Gateway dostupna je na peak-system stranici: https://www.peak-system.com/PCAN-Wireless-Gateway.331.0.html . Dodatno korišten je softver PCAN-View za presretanje CAN Bus TCP paketa u Windows okruženju koji je dostupan na https://www.peak-system.com/PCAN-View.242.0.html.

Sav Python kod za potrebe rada nalazi se u sljedećem GitHub repozitoriju: https://github.com/dangerous999/Masters-Thesis-Object-Detection

POPIS OZNAKA I KRATICA

CAN – Controller Area Network
OHE – One Hot Encoded
FC – Fully Connected
FP – False Positive
FN – False Negative
TP – True Positive
TN – True Negative
σ – sigmoida
θ – parametri mreže
w – težine neurona ili mreže
b – pristranost neurona ili mreže
∇ – gradijent
ϵ – stopa učenja

POPIS SLIKA

POPIS TABLICA

SOFTVERSKI SUSTAV ZA AUTOMATSKO OTVARANJE USISNOG OTVORA NA SPECIJALIZIRANIM VOZILIMA

Sažetak

U ovom radu demonstrirano je učenje duboke konvolucijske neuronske mreže uz pomoć prijenosnog učenja za klasifikaciju vizualnih podataka u svrhu otvaranja i zatvaranja usisne pumpe na električnoj čistilici. Prikazana je detaljna analiza rezultata nekoliko mreža s različitim arhitekturama. Za potrebe simulacije mreže su naučene da mogu razlikovati slike s ili bez teniske loptice. Komunikacija između neuronske mreže i čistilice odrađena je preko protokola CAN Bus. Modeli su u svrhe simulacije ukomponirani u Android aplikaciju u kojoj analiziraju slike s kamere mobilnog uređaja.

Ključne riječi: duboko učenje, konvolucija, binarna klasifikacija, prijenosno učenje, CAN Bus, Android, Keras

SOFTWARE SYSTEM FOR AUTOMATIC SUCTION FLAP OPENING ON SPECIALIZED VEHICLES

Summary

This work demonstrates the training of a deep convolutional neural network using transfer learning for classification of visual data for the purposes of opening and closing a suction inlet on an electric cleaning vehicle. The work provides a detailed analysis for the results of several networks with different architectures. For simulation purposes the networks are trained to differentiate between images with and without a tennis ball. The neural networks and the vehicle communicate using a CAN Bus protocol. These networks are implemented into an Android application which uses them to make predictions on a video coming from the mobile device's camera.

Keywords: deep learning, convolution, binary classification, transfer learning, CAN bus, Android, Keras

---

Dino Perić, Zagreb veljača 2020.