

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

ZAVRŠNI RAD

**Detekcija jama u zemlji primjenom dubokih
neuralnih mreža**

Tomislav Bratić

Split, rujan 2021.



SVEUČILIŠTE U SPLITU
FAKULTET ELEKTROTEHNIKE, STROJARSTVA I
BRODOGRADNJE



Preddiplomski stručni studij: **Računarstvo**

Oznaka programa: 550

Akademska godina: 2020./2021.

Ime i prezime: **TOMISLAV BRATIĆ**

Broj indeksa: 837-2018

ZADATAK ZAVRŠNOG RADA

Naslov: **DETEKCIJA JAMA U ZEMLJI PRIMJENOM DUBOKIH NEURALNIH MREŽA**

Zadatak: U završnom radu potrebno je izraditi aplikaciju koja će temeljem analize slike snimljene kamerom s visine od 50-150 cm detektirati eventualne jame u zemlji. U okviru rada potrebno je dati pregled umjetnih neuralnih mreža i njihovih mogućnosti. Potrebno je izraditi bazu slika s pozitivnim primjerima kao i s negativnim primjerima. Izraditi duboku neuralnu mrežu za detekciju. Koristiti Matlab softverski paket. Provesti postupak učenja neuralne mreže. Dobivene rezultate evaluirati i komentirati. Radu priložiti cjeloviti kod i bazu slika.

Datum obrane:

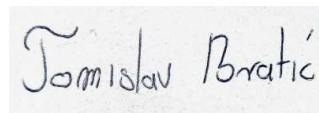
Mentor:

prof. dr. sc. Vladan Papić

IZJAVA

Ovom izjavom potvrđujem da sam završni rad s naslovom DETEKCIJA JAMA U ZEMLJI PRIMJENOM DUBOKIH NEURALNIH MREŽA pod mentorstvom prof. dr. sc. Vladan Papić pisao samostalno, primijenivši znanja i vještine stečene tijekom studiranja na Fakultetu elektrotehnike, strojarstva i brodogradnje, kao i metodologiju znanstveno-istraživačkog rada, te uz korištenje literature koja je navedena u radu. Spoznaje, stavove, zaključke, teorije i zakonitosti drugih autora koje sam izravno ili parafrazirajući naveo u završnom radu citirao sam i povezao s korištenim bibliografskim jedinicama.

Student

A rectangular box containing a handwritten signature in dark ink. The signature is written in a cursive style and reads "Tomislav Bratic".

Sadržaj

1	UVOD.....	1
1.1	Neuralna mreža	1
1.2	Uloga neuralne mreže.....	1
1.3	<i>Deep learning</i>	1
1.3.1	Konvolucijska mreža	2
2	IZRADA NEURALNE MREŽE KORIŠTENJEM GOOGLNET-A	4
2.1	Priprema baze podataka	4
2.2	Definiranje opcija prije treniranja	5
2.3	Proces treniranja mreže	6
2.3.1	Aktivacijske funkcije	9
2.3.2	Komponente metode odlaska ili <i>Backpropagation-a</i>	10
2.4	Izrada mreže u <i>Matlab-u</i>	15
2.5	Graf.....	17
3	IZRADA NEURALNE MREŽE KORIŠTENJEM SQUEEZENET-A	20
3.1	Tablice mjerenja	22
3.2	Slike i grafovi	25
3.3	Rezultati	30
	ZAKLJUČAK	34
	LITERATURA	35
	POPIS OZNAKA I KRATICE	36
	SADRŽAJ	37
	SUMMARY	38

1 UVOD

1.1 Neuralna mreža

Neuralna mreža (koja se također zove i umjetna neuralna mreža) je adaptivni sustav koji uči pomoću međusobno povezanih neurona ili slojeva u slojevitoj strukturi koja nalikuje ljudskom mozgu. Neuralna mreža može učiti iz podataka tako da se može osposobiti za razvrstavanje podataka, prepoznavanje uzoraka i predviđanje budućih događaja. Neuralna mreža razdvaja ulaz u više slojeva apstrakcije. Može se trenirati koristeći različite primjere za prepoznavanje uzoraka u govoru ili slikama, baš kao i ljudski mozak. Takvo ponašanje definirano je načinom na koji su njegovi pojedinačni elementi povezani “težinom” tih veza. Te se težine automatski prilagođavaju tijekom treninga prema određenom pravilu učenja sve dok umjetna neuralna mreža ne završi određeni zadatak.

1.2 Uloga neuralne mreže

Neuralne mreže se često koriste za prepoznavanje uzoraka kako bi identificirale i klasificirale objekte ili signale u slikama, zvuku, vida ili pokreta. Neki primjeri su: vidno prepoznavanje određenog objekta u stvarnom vremenu pomoću analiziranja velikog broja sličnih objekata iz baze podataka, prepoznavanje glasovnih komandi u govoru, prevođenje mandarinskog pisma ili japanskog pisma u neki drugi odabrani jezik pa čak i prepoznavanje tumora u ljudskom organizmu pomoću analize oblika stanica ili grupacije velikog broja stanica.

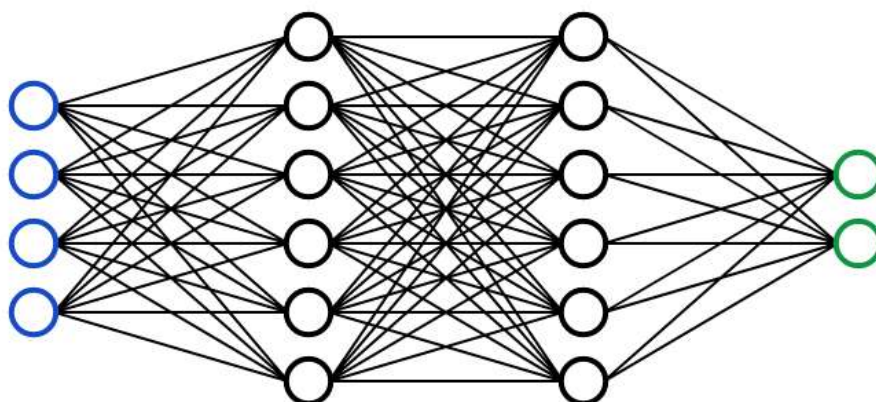
1.3 *Deep learning*

Deep learning spada u granu strojnog učenja koji koristi više slojeva neuralne mreže za izdvajanje obilježja iz određenog unosa. Postoje različite arhitekture dubokog učenja kao što su mreže dubokog uvjerenja, učenje dubokog pojačanja i konvolucijske mreže .

1.3.1 Konvolucijska mreža

Konvolucijska mreža (*ConvNet* ili *CNN*) je *deep learning* arhitekturna mreža koja direktno uči iz podataka, eliminirajući potrebu za ručno izvlačenje obilježja nekog objekta ili signala. Uglavnom se koriste na slikama za prepoznavanje objekata lica ili događaja. Također se mogu koristiti za prepoznavanje zvuka ili informacija u signalu. Najveće prednosti CNN-a je pružanje jako preciznih rezultata u prepoznavanju, mogu se vrlo lako prilagoditi za treniranje drugih oblika prepoznavanja koristeći već istrenirane mreže.

Konvolucijska mreža može imati desetke do stotine slojeva mreže koja svaka uči različite karakteristike slike. Filteri su dodijeljeni svakoj slici koja se trenira, a vrijednosti izlaza jednog sloja postaju ulazi na sljedećem sloju. Filteri mogu biti neka jednostavna obilježja kao naprimjer rezolucija ili svjetlina. Može se postepeno povećavati kompleksnost obilježja.



Slika 1.1 Neuralna mreža. Plava boja predstavlja ulazni sloj, zelena boja izlazni sloj i crna boja sakriveni sloj

CNN se sastoji od ulaznog sloja, izlaznog sloja i velikog broja sakrivenih slojeva između njih.

Ti slojevi izvode operacije koje mijenjaju podatke s namjerom učenja značajki specifičnih za podatke. Tri najčešća sloja su: konvolucija, aktivacija ili ReLU i udruživanje.

Konvolucija stavlja ulazne slike kroz skup konvolucijskih filtara, od kojih svaki aktivira određene značajke sa slike.

Rectified linear unit (ReLU) omogućuje učinkovitiji i brži trening korištenjem mapiranja negativnih vrijednosti na nulu i održavanjem pozitivnih vrijednosti. Često se naziva i aktivacija, jer se samo aktivirana obilježja prenose naprijed u sljedeći sloj.

Objedinjavanje(pooling) pojednostavljuje izlaz izvođenjem nelinearnog smanjivanja broja piksela, smanjujući broj parametara koje mreža treba naučiti.

2 IZRADA NEURALNE MREŽE KORIŠTENJEM GOOGLNET-A

Za razvoj ovog projekta koristiti će se već izrađena mreža *GoogleNet*. To je konvolucijska mreža razvijena od strane *Google*-a koja sadrži 22 sloja dubine. Jako je popularna, može klasificirati preko 1000 objekata i u treniranju je korišteno tisuće slika.

Da bi se uspješno mogla koristiti mreža za detekciju odabranog objekta koristi se koncept *Transfer learning* koji zamjenjuje zadnji sloj mreže *GoogleNet*-a sa definiranim slojem. U ovom slučaju taj sloj će se prilagoditi za detekciju jama u zemlji. Izrada će se sastojati u 3 koraka: priprema baze podataka, treniranje mreže i testiranje mreže.

2.1 Priprema baze podataka

Za bazu podataka prikupilo se 286 slika jama u tlu. Otprilike polovica slika je pronađeno na internetu dok je ostatak prikupljeno u prirodi fotografirano sa *smartphone*-om. Kreira se direktorij koja će sadržavati kod i bazu slika. Baza slika će se razdvojiti na dva dijela: *hole* i *nothole* i biti će u zasebnom direktoriju. Primjeri koji spadaju u *nothole* su livade, ceste, brežuljci ili put. Nakon što se pripremi baza slika pokreće se Matlab. Korištenje naredbe *ImageDatastore()* dohvaćaju se sve slike u bazi slika a označavaju se pomoću imena datoteka. Cijela se baza spremi u određenu varijablu koja će podijeliti na dio za treniranje i za validaciju. Omjer podijele je 70% za trening, 30% za validaciju. Potom su potrebna 2 sloja arhitekture koje će se modificirati. Prvi je *loss3-classifier layer* ili sloj za učenje obilježja. Također je i potpuno povezani sloj jer se nalazi iza zadnjeg procesa objedinjavanja. Drugi je izlazni sloj koji će omogućiti pravilnu klasifikaciju izlaznih podataka. Definira se vlastiti sloj za učenje obilježja tako da se dodijeli ime, broj klasa, faktor učenja težine i faktor učenja pristranosti.

Kod podešavanja vrijednosti težine i pristranosti koriste se vrijednosti od 1 pa naviše. Ti brojevi označavaju faktor učenja istih u odnosu na globalno učenje mreže. Naprimjer ako se postavi vrijednost učenja na 2 onda će se težina i pristranost istrenirati 2 puta brže nego globalno učenje mreže. Potom se zamjeni *Googlenet*-ovi slojevi sa definiranim slojevima. S ovim korakom završava se priprema baze podataka i nastavlja se na definiranje opcija za treniranje mreže.

2.2 Definiranje opcija prije treniranja

Nakon uspješno izvršenog prvog koraka slijedi postavljanje programa za treniranje mreže. Prvo dolazi augmentacija slike. Ona povećava broj podataka za treniranje mreže na način da se izmjeni svojstva ili položaj slike. Također može pomoći da rezultati budu precizniji s obzirom na moguće distorzije u slikama. Naprimjer, slikama se može dodijeliti drukčija boja, rotacija slike ili smanjiti veličina slike. *imageDataAugmenter* je objekt u kojem će se opisati sve postavke za augmentaciju. Pozivanjem *augmentedImageDatastore* stvara se objekt koji će spremi postavke augmentacije. *AugmentedImageDatastore* -u je potrebno još dodijeliti i veličinu slike, bazu slike i ime. Prilikom treniranja *augmentedImageDatastore* vrši augmentaciju i mijenja veličinu slika s tim da ne sprema promijenjene slike u memoriju. Naredba *trainNetwork* će kasnije ažurirati parametre i odbaciti augmentirane slike.

U *imageDataAugmenter* upisuje se *RandxReflection* koja stvara refleksiju slike po x osi. Kada se postavi na *true* postoji 50% posto vjerojatnosti na svim slikama da će se refleksija desiti horizontalno. Moguće je pozvati i *RandYReflection* za refleksiju slike po y osi. *RandXScale* i *RandYScale* omogućuju skaliranje slike po x i y osi, a zahtijevaju unos vektora sa 2 vrijednosti s tim da druga vrijednost elementa mora biti jednaka ili veća od prvog elementa. U procesu se odabire nasumična vrijednost skaliranja između vrijednosti vektora. Drugi način unos vrijednosti je preko *handle* funkcije u kojoj se deklarira faktor skaliranja. *RandYTranslation* i *RandXTranslation* omogućavaju translaciju slike po x i y osi koje također zahtjeva vektor s dva elementa ili *handle* funkciju. Translacija se određuje preko udaljenosti piksela a ona uzima određeni dio slike iz jedne domene i pretvara je tako da će ista imati karakteristike slike druge domene. Inače se ovaj proces popularizirao zadnjih godina radi povećanja broja aplikacija koje se koriste u *computer vision* problemima.

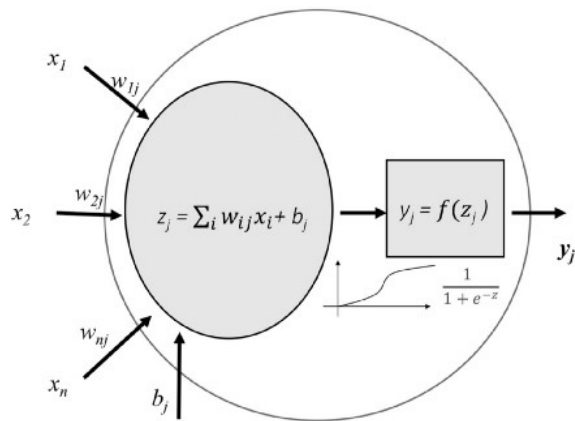
Prije pokretanja treniranja moraju se podesiti opcije za treniranje neuralne mreže. Postavlja se veličina paketa ili *batch*-eva koje služe da se podaci za treniranje prenose mrežom. Njihova veličina ovisi o dosta faktora a jedan od glavnih jest memorija računala koja je potrebna za izračun. Potom postavlja se broj epoha ili krugova koje će neuralna mreža proći. Povećanjem broja epoha povećavamo preciznost našeg rezultata ali i dužinu procesa. Nekađ je dobro podesiti na više epoha u slučaju preklapanja rezultata, a to se dogodi kada preciznost kod validacije počinje opadati dok se preciznost istreniranih podataka povećava. Potrebni su i validacijski

podaci. To su podaci koji će se koristiti za provjeru valjanosti tijekom vježbanja, navedeni kao spremište podataka, tablica ili polje ćelije koje sadrži prediktore za provjeru valjanosti i rezultate. Kada se mreža trenira izračunava se preciznost validacije i gubitak validacije. Da bi se podesilo učestalost validacije koristi se argument *Validation Frequency*. To je broj iteracija između procjena mjernih podataka validacije. *Shuffle* naredba se javlja da se podaci za trening i validacijski podaci zasebno ispremijeshaju. Po standardu je vrijednost 1, moguće je staviti 0 ili svaka epoha. Kada se postavi za svaku epohu, tada se podaci za treniranje prije svake epohe za trening ispremijeshaju i također validacijski podaci prije svake epohe za validaciju. *Verbose* opcija omogućava prikaz informacija o napretku vježbanja u naredbenom prozoru. Informacije koje prikazuje su: broj epohe, broj iteracije, proteklo vrijeme, preciznost mini paketa, gubitak na mini paketima, preciznost validacije, gubitak na validaciji i osnovna stopa učenja.

Zadnje opcije koje se dodaju su: *Stochastic Gradient Descent Momentum* ili SGDM i *Initial Learning Rate* ili stopa učenja. Ukratko SGDM je tip optimizatora koje će izračunati gubitke svakog sakrivenog sloja mreže. Pokazuje smjer kretanje funkcije gubitaka tako da se neuralna mreža može prilagoditi da ostvari veću preciznost mreže. *Initial Learning Rate* je hiperparametar koji izračunava koliko se model mijenja s obzirom na greške koje nastaju nakon svakog ažuriranja težina. U sljedećem poglavlju promotriti će se povezanost ovih dvaju pojmova. Zadnja naredba koja pokreće testiranje se naziva *trainNetwork*, a ona prihvata augmentirane podatke, modificiranu mrežu korištenjem metode *Transfer Learning* opcije treniranja.

2.3 Proces treniranja mreže

Kao što je spomenuto na početku, neuralna mreža je skup neurona povezanih jedno s drugim. U isto vrijeme, svaka veza neuralne mreže povezana je s težinom koja diktira važnost ovog odnosa u neuronu kada se pomnoži s ulaznom vrijednošću. Svaki neuron ima aktivacijsku funkciju koja definira izlaz neurona.

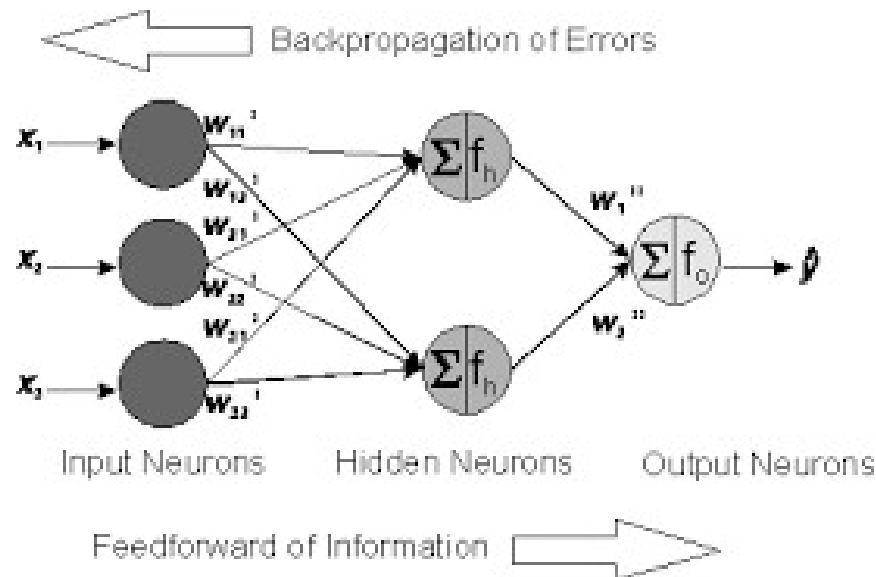


Slika 2.1 Prosljeđivanje vrijednosti u neuralnoj mreži

Treniranje neuralne mreže, odnosno učenje vrijednosti parametara (težine w_{ij} i b_j pristranosti) je ono što čini neuralnu mrežu posebnom i originalnom. Proces učenja se može vidjeti u neuralnoj mreži kao iterativni proces odlaska i povratka slojeva neurona. Odlazak je prosljeđivanje informacija, a povratak je pozadinska propagacija informacija.

Prva faza odlaska ili *ForwardPropagation*-a događa se kada je mreža izložena podacima o treningu i oni prelaze cijelu neuralnu mrežu kako bi se izračunala njihova predviđanja (oznake). To jest, prosljeđivanje ulaznih podataka kroz mrežu na takav način da svi neuroni primjenjuju svoju transformaciju na informacije koje primaju od neurona prethodnog sloja i šalju ih neuronima sljedećeg sloja. Kada podaci prijeđu sve slojeve, a svi njegovi neuroni su napravili svoje izračune, konačni sloj će se postići rezultatom predviđanja oznaka za te ulazne primjere. Zatim se koristi funkcija gubitka za procjenu gubitka (ili pogreške) i za usporedbu i mjerenje koliko je dobar ili loš rezultat predviđanja bio u odnosu na točan rezultat. U idealnom slučaju, želi se da greška bude nula, odnosno bez razlike između procijenjene i očekivane vrijednosti. Stoga, kako se model trenira, težine neurona postupno će se prilagođavati dok se ne dobiju dobra predviđanja. Nakon izračuna gubitka te se informacije prenose unatrag. Taj se proces naziva povratak ili *BackPropagation*. Polazeći od izlaznog sloja, te informacije o gubitku šire se na sve neurone u skrivenom sloju. Međutim, neuroni skrivenog sloja primaju samo dio ukupnog signala gubitka, na temelju relativnog doprinosa koji je svaki neuron pridonio izvornom izlazu. Taj se proces

ponavlja, sloj po sloj, sve dok svi neuroni u mreži ne prime signal gubitka koji opisuje njihov relativni doprinos ukupnom gubitku.



Slika 2.2 metoda odlaska i povratka

Nakon što su sve informacije raširile metodom odlaska može se prilagoditi težina veza između neurona. Cilj je ostvariti što manji gubitak, odnosno da vrijednost gubitka bude jednaka nuli. Zbog toga će se koristiti metoda koja se zove gradijentni spust ili *Gradient descent*. Ova metoda mijenja težine u malim inkrementima uz pomoć izračuna derivacije funkcije gubitka, što omogućuje da se vidi u kojem smjeru će vrijednost padati prema globalnom minimumu[1]. To se općenito radi u serijama podataka u uzastopnim iteracijama svih skupova podataka koje se prosljeđuju mreži u svakoj iteraciji. U opcijama postavilo se broj epoha na 6. Inače se može promijeniti broj na više epoha ako se zahtijevaju precizniji rezultati.

Da se zaključi, koraci koje definiraju algoritam učenja su:

1. Unose se vrijednosti (često nasumične) za mrežne parametre (w_{ij} težine i b_j pristranosti)
2. Priprema se skup primjera ulaznih podataka i proslijedi se ih kroz mrežu kako bi dobili njihovo predviđanje.(Cijeli taj proces je već opisan u poglavlju 2.1)

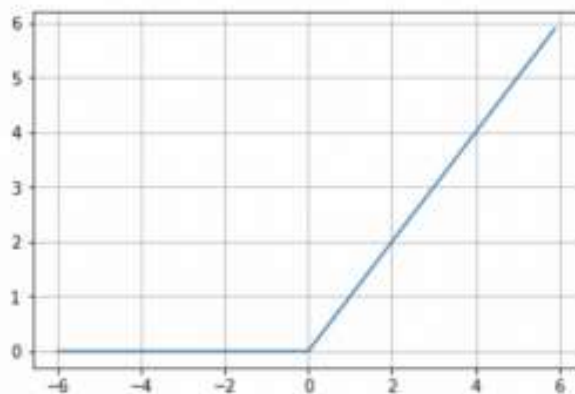
3. Uspoređuje se predviđanja dobivena s vrijednostima očekivanih oznaka i izračunava se gubitak
4. Metoda odlaska vrši pozadinsku propagaciju kako bi se proširio gubitak na svaki od parametara koji čini model neuronske mreže
5. Propagirana informacija će poslužiti za ažuriranje parametara neuronske mreže s *Gradient descent* metodom na način da se ukupni gubitak smanji i dobije bolji model
6. Iteracija koristi sve prethodne korake dok se ne dobije zadovoljavajući model

2.3.1 Aktivacijske funkcije

Aktivacijske funkcije se koriste za širenje izlaza neurona prema naprijed. Ovaj izlaz primaju neuroni sljedećeg sloja na koji je ovaj neuron povezan. Aktivacijske funkcije služe za uvođenje nelinearnosti u mogućnosti modeliranja mreže. Postoje ih više, a navest će se dvije osnovne funkcije:

Softmax: pretvara izlaz prethodnog sloja u vektor vjerojatnosti. Obično se koristi za višeklasnu klasifikaciju. Često se može pronaći na izlaznom sloju neuralne mreže. Arhitekture koje koriste su: *AlexNet*, *ResNet* i *GoogleNet*.

ReLU ili *Rectified linear unit*: vrlo je zanimljiva transformacija koja aktivira čvor ako je ulazna vrijednost iznad određenog praga. Uobičajenije ponašanje je da, sve dok ulaz ima vrijednost ispod nule, izlaz će biti nula, ali, kada se ulaz podigne iznad, izlaz stvara odnos s ulaznom varijablom primjera $f(x)=x$. ReLU aktivacijska funkcija se pokazala da djeluje u mnogim različitim situacijama i trenutno se široko koristi.



Slika 2.3 Rectified linear unit (ReLU)

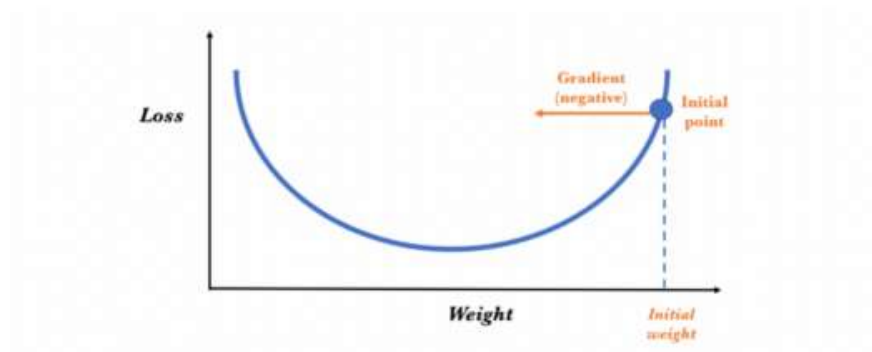
2.3.2 Komponente metode odlaska ili *Backpropagation*-a

Ukratko, *Backpropagation* metodu se može smatrati metodom za promjenu parametara (težina i pristranosti) neuronske mreže u idealnom smjeru. Prvo počinje izračunavanjem termina gubitka, a zatim se parametri neuronske mreže prilagođavaju obrnutim redoslijedom pomoću algoritma optimizacije uzimajući u obzir ovaj izračunati gubitak. Prva komponenta je funkcija gubitka koja je jedan od parametara koje će poslužiti za približavanje neuralne mreže idealnoj težini. Optimizator je još jedan od argumenata koji se koristio u opcijama treniranja a postoje različiti optimizatori: SGD, RMSprop, Adagrad, Adadelata. Iako već spomenut, opisati će se jedan od optimizatora tako da se bolje razumije cjelokupni rad optimizatora. Stupnjevito spuštanje ili *Gradient descent* je osnova mnogih optimizatora i jedan od najčešćih algoritama optimizacije u strojnom učenju i dubokom učenju.

Stupnjevito spuštanje koristi prvu derivaciju funkcije gubitka prilikom ažuriranja parametara. Bitno je spomenuti da derivacija daje nagib funkcije u tom trenutku. Proces se sastoji u povezanim derivacijama gubitka svakog skrivenog sloja od derivacija gubitka njegovog gornjeg sloja, ugrađujući njegovu funkciju aktivacije u izračun (zato funkcije aktivacije moraju biti derivabilne). U svakoj od iteracija, nakon što svi neuroni imaju vrijednost derivacije funkcije gubitka koja im odgovara, vrijednosti parametara ažuriraju se u suprotnom smjeru od onih označenih vrijednosti derivacije. Vrijednost uvijek pokazuje smjeru u kojem se vrijednost

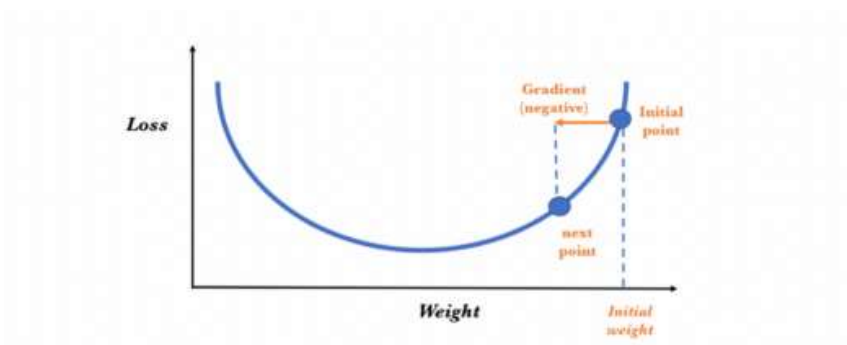
funkcije gubitka povećava. Stoga, ako se koristi negativ gradijenta, može se dobiti smjer u kojem se nastoji smanjiti funkcija gubitka.

Postupak se može vidjeti na slici Slika 2.4 pod pretpostavkom samo jedne dimenzije. Pretpostavi se da ova linija predstavlja vrijednosti koje funkcija gubitka uzima za svaku moguću vrijednost parametra i da je negativ gradijenta prikazan strelicom u početnoj točki:

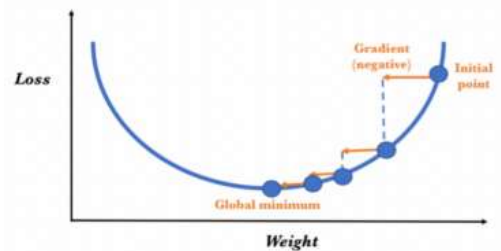


Slika 2.4 Graf funkcije gubitka

Da bi se odredila sljedeća vrijednost parametra, ova metoda će se ponašati tako da će usmjeriti težinu u suprotnom smjeru od kretanja linije. U početnoj točki desna strana pokazuje rast pogreške, ali greška treba biti što manja. Ovaj će algoritam ponavljati proces sve dok ne dođe do minimuma vrijednosti.



Slika 2.5 Vrijednost težine nakon procesa



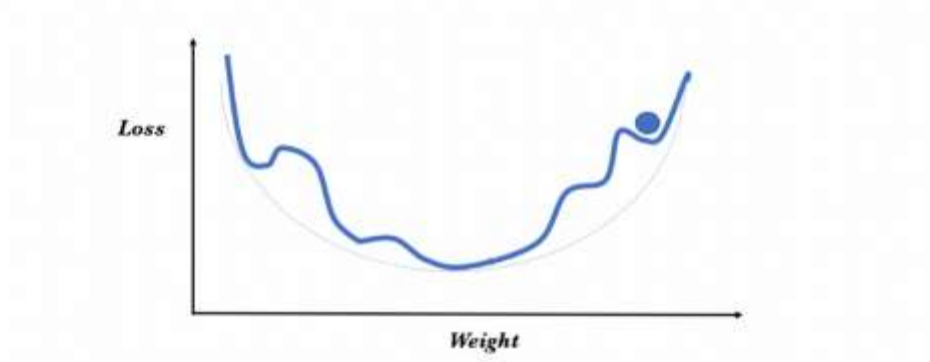
Slika 2.6 Pronalazak minimuma

U prethodnim odjeljcima vidjelo se kako se vrijednosti parametara prilagođavaju, ali ne u kojoj frekvenciji. Javlja se pojam *Stochastic Gradient descent* ili SGD koji je sličan GD-u ali ima bitne razlike.

U GD-u i u SGD -u ažurira se skup parametara na iterativni način kako bi se minimizirala funkcija pogreške. Kad se koristi GD, mora se proći kroz sve uzorke u skupu za trening kako bi se izvršilo jedno ažuriranje za parametar u određenoj iteraciji, u SGD-u, s druge strane, koristi se samo jedan ili podskup uzorka (paketa) vježbanja iz skupa za vježbanje kako bi se ažurirao parametar u određenoj iteraciji. Dakle, ako je broj uzoraka vježbanja velik, zapravo vrlo velik, tada korištenje GD-a može potrajati predugo jer u svakoj iteraciji kada se ažurira vrijednosti parametara prolazi se kroz kompletan skup treninga. S druge strane, korištenje SGD-a bit će brže jer se koristi samo jedan uzorak, koji se nasumično odabire i on se počinje poboljšavati odmah od prvog uzorka. SGD se često konvergira mnogo brže u usporedbi s GD-om, ali funkcija pogreške nije tako dobro minimizirana kao u slučaju GD-a. Često je u većini slučajeva dovoljna bliska aproksimacija koja se dobiva u SGD-u za vrijednosti parametara jer dosežu optimalne vrijednosti.

Međutim, fali zadnji pojam a on se naziva zamah ili *momentum*. Problem kod SGD-a jest da on može zapeti na vrijednosti lokalnog minimuma, dok se zapravo traži vrijednost globalnog minimuma pa zbog toga daje nedovoljno optimalne rezultate. Problem nastaje kada se zapne na lokalnom minimumu gradijent postaje 0 i tako se više ne može pomaknuti. Jedan od načina je da se sve resetira i krene od drugog početnog položaja što će povećati mogućnost dosega globalnog

minimuma. Drugo rješenje je upravo zamah koji će uzeti vrijednost težina prošlih koraka tako da uspije pomaknuti gradijent koji je zapeo. Može nastati problematika kod određivanja vrijednosti zamaha zato što će koristiti težine više koraka osim težine samo posljednjeg koraka. Zamah se pokušava postaviti u vrijednosti između 0 i 1. U praksi se pokazalo da zamah daje bolje rezultate naspram ponovnog pokretanja mreže.



Slika 2.7 Lokalni minimum u odnosu na globalni minimum

Vektor gradijenta ima smjer i magnitudu. GD algoritmi množe veličinu gradijenta skalarom poznatim kao *Learning rate* koja se ponekad naziva i veličina koraka kako bi se odredila sljedeća točka.

Na formalniji način algoritam odlaska izračunava kako se pogreška mijenja u odnosu na svaku težinu:

$$\frac{dError}{dw_{ij}} \quad (2.1)$$

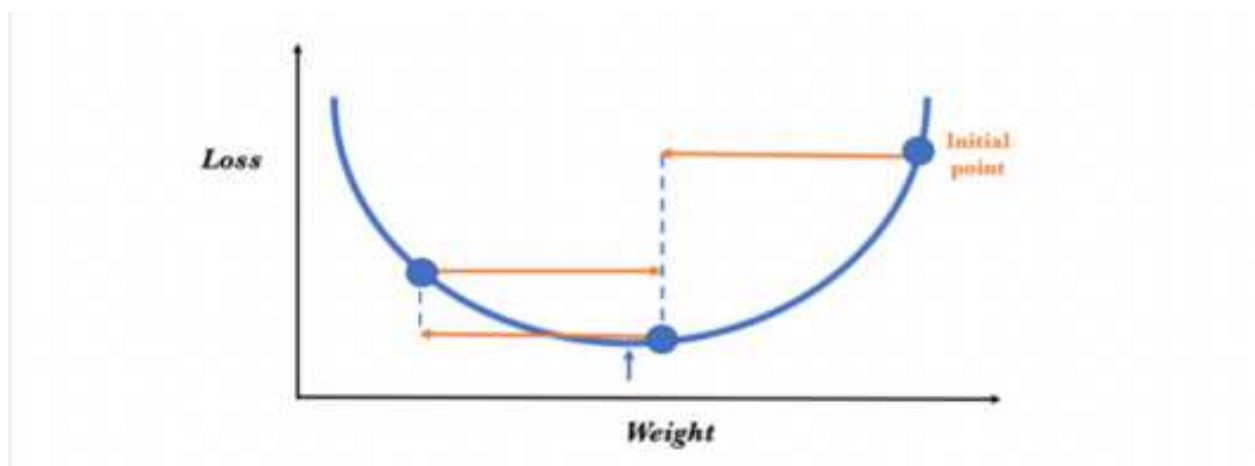
Jednostavno pravilo ažuriranja:

$$w_{ij} = w_{ij} - \alpha \frac{dError}{dw_{ij}} \quad (2.2)$$

Gdje je α stopa učenja.

Naprimjer, ako je veličina gradijenta 1,5, a stopa učenja 0,01, algoritam spuštanja gradijenta odabrat će sljedeću točku u 0,015 od prethodne točke.

Pravilna vrijednost ovog hiperparametra vrlo ovisi o određenom problemu, ali općenito, ako je vrijednost prevelika, algoritam izvodi ogromne korake, što bi moglo otežati pronalaženje zaustavne točke ili minimuma, jer će proces vjerojatno preskočiti minimum. Kada se traži sljedeća točka, vrijednost bi se mogla odbijati između dna vrijednosti. Može se vidjeti na ovoj slici Slika 2.8 učinak koji se može pojaviti, gdje se minimalna vrijednost nikada ne dosegne.



Slika 2.8 Greška stope učenja

Suprotno tome, ako je stopa učenja mala, ostvarit će se mali napredak, s boljim izgledima za postizanje lokalnog minimuma, ali to može uzrokovati da proces učenja bude vrlo spor. Općenito, dobro pravilo je smanjiti stopu učenja ako model učenja ne funkcionira. Ako je poznato da je gradijent funkcije gubitka mali, onda je potrebno testirati da kompenzira gradijent sa stopom učenja.

Zato je stopu učenja najbolje odrediti tako da opada što se više približava rješenju. Takva stopa učenja se još zove i *Learning rate decay*. Ona će omogućiti da će se mreža istrenirati brže na početku, dok nakon svake epohe će se smanjivati. Manja stopa učenja na kraju će odrediti bolje rezultate i greška učenja će biti manja.

2.4 Izrada mreže u *Matlab-u*

Da bi se uspješno implementirala mreža za korištenje, potrebno je korištenje okruženja koji omogućava podršku izrade mreže. Koristi se *Matlab*, koji je ujedno i programski jezik i računalno okruženje koje je razvila kompanija *MathWorks*. Prvi korak je pokretanje *Matlab-a* i instalacija dodatnih alata koji će biti potrebne za implementaciju mreže. Klikom na gumb *Add-ons* otvara se prozor na kojem su prikazani različiti alati i njihova mogućnost instalacije. U tom prozoru potrebno je instalirati sljedeće alate: *Deep Learning ToolBox*, *GPU Coder* i *Image Processing Toolbox*. Pošto će se koristiti *GoogleNet* arhitektura potrebno je još instalirati i *Deep Learning Toolbox For GoogleNet Network*. Ako *Matlab* nije prije toga već instaliran na računalu i mora se instalirati, njegove opcije instalacije omogućuju biranje ovih alata koje će se dodatno instalirati skupa s *Matlab-om*. Ako se želi koristiti grafička kartica za pokretanje procesa treniranja, prvo se mora provjeriti kompatibilnost grafičke kartice sa *CUDA* alatom koja se nalazi na stranici kompanije *Nvidia*. Također grafička kartica mora biti marke *Nvidia*. Nakon toga se slijede upute instalacije na njihovoj stranici. Odabire se datoteka u kojoj se želi da se kod nalazi i pripremi se dodatna datoteka koja će biti baza slika i dodatna datoteka za testiranje slika. Odabir datoteke se može obaviti korištenjem trake za navigaciju. Pritom se desnim klikom u području prikaza datoteke kreira dokument koji će služiti za treniranje mreže.

Za pisanje koda *Matlab* koristi vlastiti programski jezik. Inicijalizira se varijabla *Dataset* kojoj se pridružuje funkcija *ImageDatastore*, a funkcija sadrži vrijednosti: ime datoteke u kojoj se nalazi baza slika, uključivanje *subfolder-a* postavljeno na *true* i *LabelSource* postavljeno na *foldernames*. Potom se podijeli *Dataset* na *TrainingSet* i *ValidationSet* u omjeru 70:30. Inicijalizira se varijabla *net* koja će prihvatiti arhitekturu *GoogleNet* i nakon toga varijabla *Input_Layer_Size* prihvaća ulaznu vrijednost do koje se pristupa upisom *net.Layers(1).Inputsize*. Na isti način se izvlače *loss3-classifier layer* koje je na 142. sloju i izlazni sloj koji je na 144. sloju. Za bolje razumijevanje, Korištenjem funkcije *LayerGraph* se može iscrtati cijela

arhitektura *GoogleNet*-a i vidjeti pozicije slojeva. Odredi se broj klasa brojanjem *label*-a. Kreira se *New_Feature_Learner* koji prihvaća ime, broj klasa, *WeightLearnRateFactor* i *BiasLearnRateFactor* (Objašnjenje zadnjih dvaju pojmova se nalazi u poglavlju 2.1). Varijabla *New_Classifier_Layer* stvara novi sloj klasifikacije korištenjem naredbe *classificationLayer* koja prihvaća ime sloja. Potom se biraju opcije augmentacije (Opisane u poglavlju 2.2), definira se *Pixel_range*=[-30 30], *Scale_range*=[0.9 1.1] koje će se koristiti kao vrijednosti augmentacije.

```
Image_Augmenter = imageDataAugmenter(...
    'RandXReflection', true, ...
    'RandXTranslation', Pixel_Range, ...
    'RandYTranslation', Pixel_Range,...
    'RandXScale', Scale_Range, ...
    'RandYScale', Scale_Range);|
```

Slika 2.9 Opcije augmentacije

U *Layer_Graph* varijabli se potom zamjene slojevi tog istog sa *New_Classifier_Layer* i *New_Feature_Learner* korištenjem naredbe *replace*.

Potom se deklariraju *Augmented_Training_Image* i *Augmented_Validation_Image* koje će koristiti funkciju *augmentedImageDatastore* (Funkcija je objašnjena u poglavlju 2.2), a prihvaćaju veličinu ulaza, *Training_Dataset* ili *Validation_Dataset* i *ImageAugmenter*. Nakon toga se odrede opcije treniranja ili *Training_Options* (Opisane u poglavlju 2.2) i varijabla *net* poziva naredbu *trainNetwork* koja sadrži *Augmented_Training_Image*, *Training_Options* i *Layer_Graph*.

```

Training_Options = trainingOptions('sgdm',...
    'MiniBatchSize', Size_of_Minibatch, ...
    'MaxEpochs', 6,...
    'InitialLearnRate', 4e-4,...
    'Shuffle', 'every-epoch', ...
    'ValidationData', Augmented_Validation_Image, .
    'ValidationFrequency', 3, ...
    'ExecutionEnvironment','auto',...
    'Verbose', false, ...
    'Plots', 'training-progress');

```

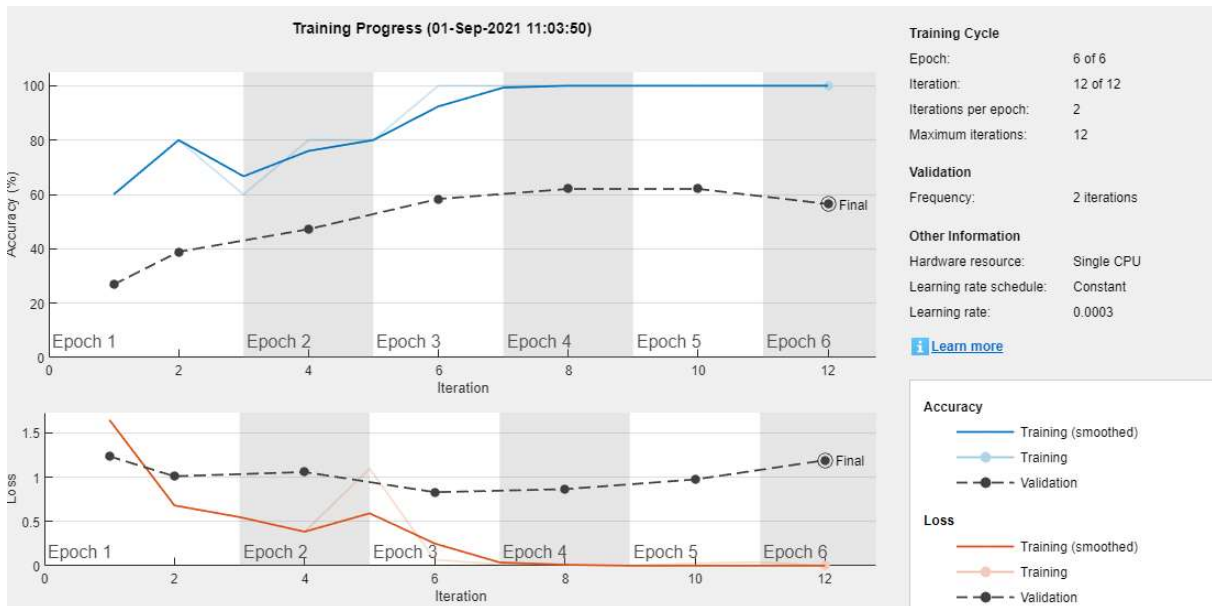
Slika 2.10 Opcije treniranja

2.5 Graf

Kada se treniraju mreže za duboko učenje jako je korisno nadzirati proces treniranja. Iscrtavanjem različitih mjernih podataka tijekom vježbanja može se saznati kako trening napreduje. Naprimjer, može se uvidjeti poboljšava li se točnost mreže i koliko brzo i je li dolazi do preklapanja podataka prilikom treniranja. Na slici Slika 2.11 se nalazi graf koji prikazuje preciznost treniranja (plava boja), preciznost validacije (crna boja), gubitci (pogreške) treniranja, i gubitci (pogreške) validacije. Preciznost treniranja prikazuje klasifikacijsku preciznost svakog pojedinačnog paketa. Validacijska preciznost prikazuje klasifikacijsku preciznost cijelog validacijskog seta. Gubitci treniranja prikazuje gubitke svakog paketa, a gubitci validacije prikazuje gubitke na validacijskom setu. Kada treniranje završi, rezultati prikazuju konačnu validacijsku preciznost. Konačne validacijske metrike su nazvane *final* u grafu. Graf također ispisuje broj iteracija koje određena arhitektura stvara i proteklo vrijeme treniranja koje je bilo potrebno arhitekturi da uspješno istrenira mrežu.

Slika se stvara tako da se ažurira mjerni podaci nakon svake iteracije. Pošto se zadalo u opcijama da se izvrši treniranje kroz 6 epoha izvršit će se 12 iteracija odnosno 2 iteracije po epohi. Preciznost treniranja teži u 1 ili u stopostotnu preciznost a gubitci teže u 0 što je zadovoljavajuće dok je problem nastao kod treninga validacije jer je preciznost manja od 60 posto, a gubitci su iznad 1. Da bi rezultati bili valjani, očekuje se da vrijednosti preciznosti treninga i preciznosti

validacije se nalaze između 80% i 100 %, a vrijednosti gubitaka između 0 i 1. Ova procjena rezultata valjanosti ovisi o korisnikovom izboru i nije nužno precizna.



Slika 2.11 Primjer grafa



Slika 2.12 Primjer slike testiranja

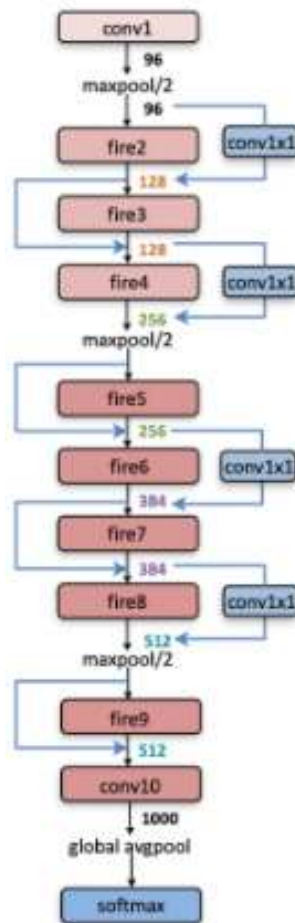


Slika 2.13 Primjer slike testiranja

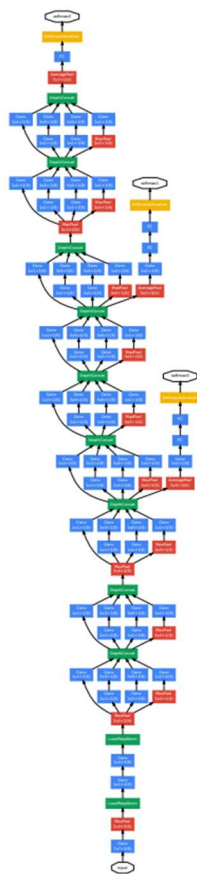
Slike iznad Slika 2.12 Slika 2.13 prikazuju konačni rezultat testiranja mreže. Ako je mreža dobro istrenirana uspjeh će prepoznati sliku jame, a također će prepoznati i sliku gdje se ne nalazi jama. Prilikom testiranja ispisat će poviše slike naziv kojem objektu slika pripada u ovom slučaju može biti *hole* ili *nothole*. Također će i ispisati preciznost prepoznavanja od 0 do 1. Ako rupa bude prepoznata s preciznošću od 0,8 do 1 smatra se zadovoljavajućom. Ovo je također procjena korisnika i nije nužno točna.

3 IZRADA NEURALNE MREŽE KORIŠTENJEM SQUEEZENET-A

Squeezenet je naziv za duboku neuralnu mrežu koja je nastala 2016 godine. Cilj stvaranja ovog projekta jest da se stvori manja neuralna mreža sa manje parametara koje se lakše mogu prilagoditi memoriji računala i lakše prenositi kroz mrežu. Sadrži 18 slojeva mreže, primjetno manji broj nego kod *GoogleNet* arhitekture koja sadrži 22 sloja. Ova mreža će se istrenirati da bi se moglo usporediti rezultate i odrediti koja je neuralna mreža optimalnija i bolja.



Slika 3.1 Arhitektura SqueezeNet-a



Slika 3.2 Arhitektura GoogleNet-a

Proces treniranja je relativno isti, i dalje se koristi metoda *Transfer Learning* samo što slojevi koji su zaduženi za treniranje mreže i potrebno ih je zamijeniti su *conv10* sloj i *ClassificationLayer_predictions* sloj. Naravno baza podataka mora biti pripremljena prije pokretanja treninga. Nije se koristila augmentacija slike kao kod *GoogleNet-a*. Jedina modifikacija jest promjena veličine slike.

3.1 Tablice mjerenja

Tablice mjerenja sadržavaju rezultate mjerenja mreže u konačnoj točki koja je na grafu nazvana *final* točka (objašnjenje svih elemenata tablice može se naći u poglavlju 2.5). Istrenirane su *GoogleNet* i *SqueezeNet* mreže sa zadanim opcijama:

- Epoha:6
- Veličina paketa:5
- Stopa učenja: $3 \cdot 10^{-4}$
- Validacijska frekvencija:3

Tablica 3.1 Rezultati mjerenja *SqueezeNet-a*

Naziv Arhitekture	Broj slika	Proteklo vrijeme	Broj iteracija	Preciznost treniranja (%)	Preciznost validacije (%)	Gubitci (pogreške) treniranja	Gubitci (pogreške) validacije
Squeezenet	50	4 min 16 sec	42	~100%	97,22%	~0	~0
	150	9 min 57 sec	102	~100%	97,75%	~0	~0
	286	21 min 13 sec	168	97%	100%	~0	~0

Tablica 3.2 Rezultati mjerenja GoogleNet-a

Naziv Arhitekture	Broj slika	Proteklo vrijeme (min, sec)	Broj iteracija	Preciznost treniranja (%)	Preciznost validacije (%)	Gubitci (pogreške) treniranja	Gubitci (pogreške) validacije
GoogleNet	50	43 sec	12	~100%	11,97%	~0	4,1
	150	1 min 38 sec	12	95,00%	25,00%	~0	3,2
	286	2 min 53 sec	12	~100%	32,95%	~0	3

Sljedeće mreže imaju definirane opcije:

- Epoha:10,
- Veličina paketa:100,
- Stopa učenja: $4 \cdot e^{-4}$
- Validacijska frekvencija: 3

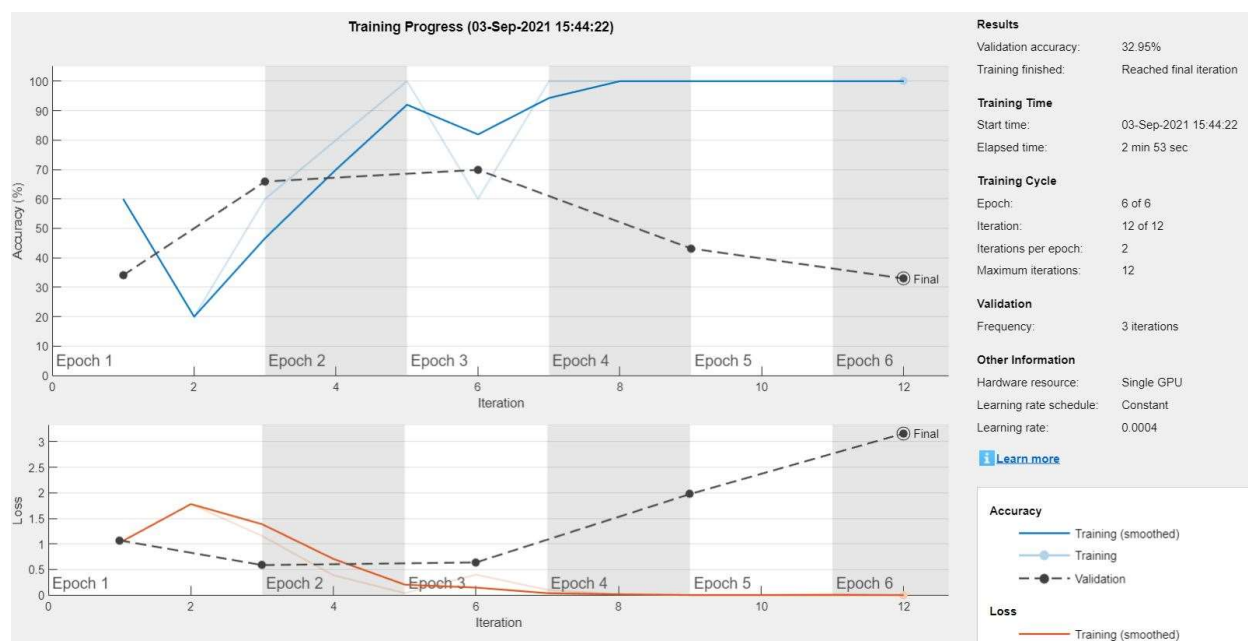
Tablica 3.3 Rezultati mjerenja SqueezeNet-a

Naziv Arhitekture	Broj slika	Proteklo vrijeme	Broj iteracija	Preciznost treniranja (%)	Preciznost validacije (%)	Gubitci (pogreške) treniranja	Gubitci (pogreške) validacije
SqueezeNet	50	2 min 33 sec	10	~76,3%	97,22%	0,2	0,2
	150	3 min 49 sec	10	84,27%	84,27%	0,5	0,4
	286	4 min 39 sec	10	90,14%	90,14%	0,2	0,2

Tablica 3.4 Rezultati mjerenja GoogleNet-a

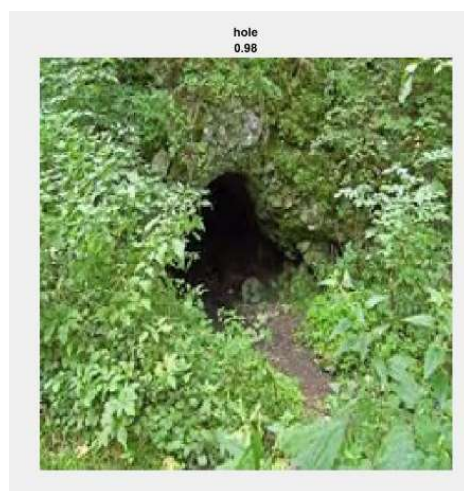
Naziv Arhitekture	Broj slika	Proteklo vrijeme	Broj iteracija	Preciznost treniranja (%)	Preciznost validacije (%)	Gubitci (pogreške) treniranja	Gubitci (pogreške) validacije
GoogleNet	50	53 sec	10	~100%	51,35%	~0	1,1
	150	1 min 41 sec	10	~100%	61,96%	~0	1
	286	2 min 25 sec	10	~100%	82,39%	~0	0,5

3.2 Slike i grafovi



Slika 3.3 Graf rezultata Googlenet-a tablice

Tablica 3.2



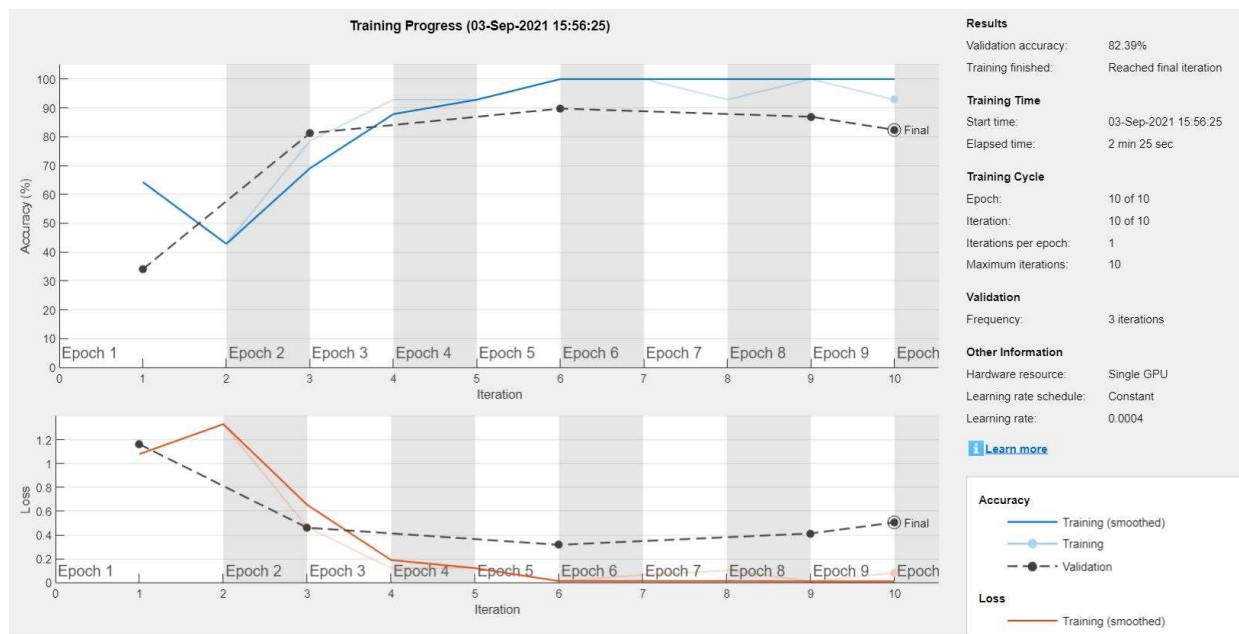
Slika 3.4 Slika rezultata GoogleNet-a tablice

Tablica 3.2

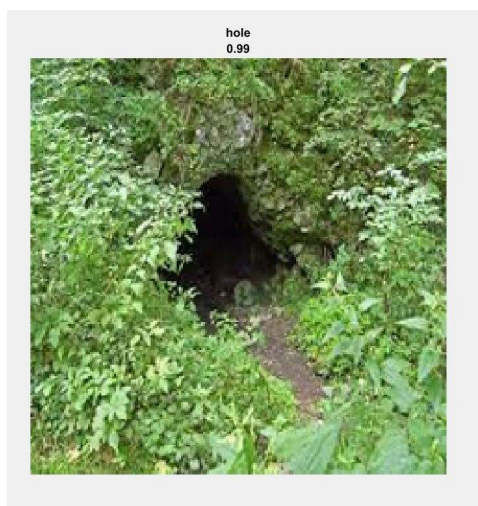


Slika 3.5 Slika rezultata GoogleNet-a tablice

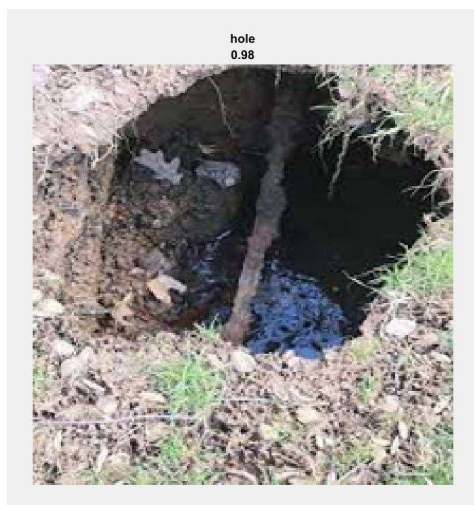
Tablica 3.2



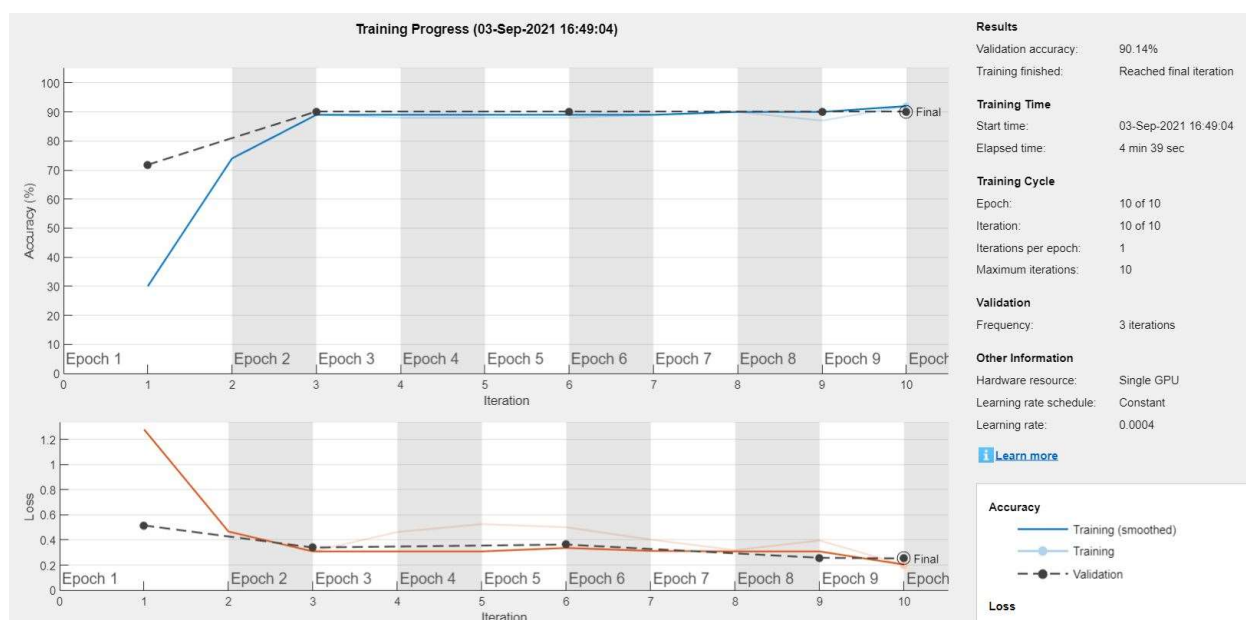
Slika 3.6 Graf rezultata Googlenet-a tablice Tablica 3.4



Slika 3.7 Slika rezultata GoogleNet-a tablice Tablica 3.4



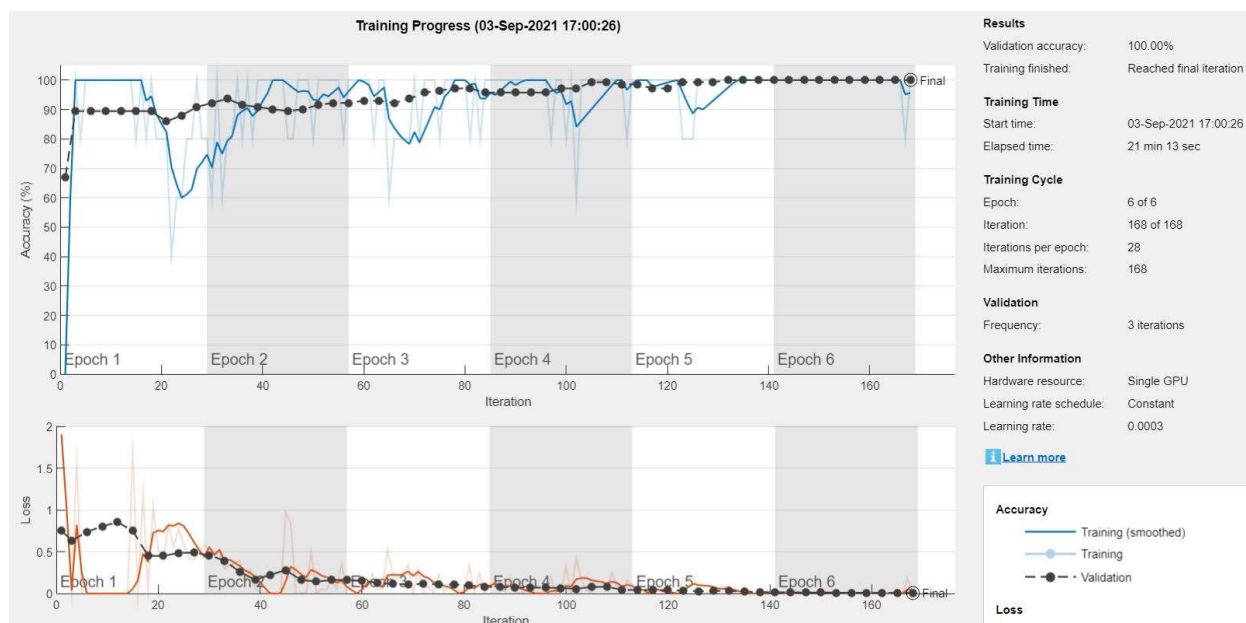
Slika 3.8 Slika rezultata GoogleNet-a tablice Tablica 3.4



Slika 3.9 Graf rezultata SqueezeNet-a tablice Tablica 3.3



Slika 3.10 Slike rezultata SqueezeNet-a tablice Tablica 3.3



Slika 3.11 Graf rezultata SqueezeNet-a tablice Tablica 3.1



Slika 3.12 Slike rezultata SqueezeNet-a tablice Tablica 3.1

3.3 Rezultati

Da bi se provjerilo je li mreža uspješno detektira jame u tlu pripremit će se 20 slika za testiranje. Pomoću *Precision* i *Recall* metoda odrediti će se kvaliteta i preciznost mreže. *Precision* se u ovom slučaju još može nazvati i pozitivna prediktivna vrijednost, a *recall* se može nazvati pozitivna istinska stopa ili osjetljivost. Definirani su skupom dohvaćenih dokumenata, što je u ovom slučaju 20 slika jama u tlu i ostalih površina koje nisu jama. Da bi se moglo koristiti mora se prvo klasificirati dobiveni rezultati. Rezultati se podijele na 4 dijela: *true positive*, *true negative*, *false positive* i *false negative*. *True* i *negative* pojmovi se odnose na to je li odgovara predviđanje vanjskoj prosudbi dok *positive* i *negative* se odnose na očekivanje rezultata. Po ovoj definiciji formule bi trebale izgledati ovako:

$$\begin{aligned}\text{Precision} &= \frac{tp}{tp + fp} \\ \text{Recall} &= \frac{tp}{tp + fn}\end{aligned}\tag{3.1}$$

gdje je:

tp *true positive* vrijednost
 fp *false positive* vrijednost
 fn *false negative* vrijednost

Da bi se ove metode mogle implementirati za određivanje njihovih vrijednosti u prepoznavanju jama u tlu, deklarirati će se rezultati tako da je tp broj točno pronađenih jama pojedinim algoritmom, fp je broj pogrešno predloženih jama pojedinim algoritmom, tn je broj točno pronađenih slika koje nisu jama pojedinim algoritmom i fn je broj “promašenih” jama. Testirat će se 20 slika koje nisu bile dio učenja mreže i provjeriti u koju će skupinu rezultata pripasti dobivena rješenja.

Rezultate se navode u tablicama za *GoogleNet* i *SqueezeNet* mrežu i sa različitim opcijama.

Rezultati *GoogleNet* i *SqueezeNet* mreže sa zadanim opcijama:

- Epoha:6
- Veličina paketa:5
- Stopa učenja: $3 \cdot 10^{-4}$
- Validacijska frekvencija:3
- Broj slika testiranja:286

Tablica 3.5 Precision i Recall rezultati

Ime mreže	Tp vrijednost	Tn vrijednost	Fp vrijednost	Fn vrijednost	Precision	Recall
GoogleNet	11	7	1	1	0,917	0,917

Rezultati *GoogleNet* i *SqueezeNet* mreže sa zadanim opcijama:

- Epoha:10

Ime mreže	<i>Tp</i> vrijednost	<i>Tn</i> vrijednost	<i>Fp</i> vrijednost	<i>Fn</i> vrijednost	<i>Precision</i>	<i>Recall</i>
GoogleNet	10	4	4	2	0,71	0,833
SqueezeNet	12	6	2	0	0,857	1

- Veličina paketa:100
- Stopa učenja: $4 \cdot e^{-4}$
- Validacijska frekvencija:3
- Broj slika testiranja:286

Tablica 3.6 Precision i Recall Rezultati

Squeezenet	11	6	2	1	0,846	0,917
------------	----	---	---	---	-------	-------

Nakon provjere utvrđeno je da *GoogleNet* i *SqueezeNet* u većini slučajeva uspješno prepoznaju jame. *GoogleNet* ima *precision* vrijednost 0,71 i *recall* vrijednost 0,833 kada je zadano manji broj epoha i paketa (uzoraka), dok *SqueezeNet* ima *Precision* vrijednost 0,857 i *recall* vrijednost 1. *GoogleNet* ima *precision* vrijednost 0,917 i *recall* vrijednost 0,917 kada je zadano veći broj epoha i paketa (uzoraka), dok *SqueezeNet* ima *Precision* vrijednost 0,846 i *recall* vrijednost 0,917. *SqueezeNet* je dao bolje rezultate kod manjeg broja epoha i paketa, a *GoogleNet* daje bolje rezultate kod većeg broja epoha i paketa. Dobro se prilagođavaju kod različitih vrsta tla i nazočnosti svijetla. Također uspijevaju prepoznati jamu unatoč različitim objektima kojim je rupa okružena kao naprimjer kanta, motika, ruka. Isto tako dobro prepoznaju negativne slike kada su primjer livade, brežuljci i slično. *GoogleNet* je imao problema kod prepoznavanja s malim brojem paketa i epoha ostvarujući samo 33% validacijske preciznosti, ali promjenom postavki treniranja uspio je doći do solidnih 82% validacijske preciznosti. Kod slike Slika 3.5 nije uspio prepoznati jamu dok je kod slike Slika 3.8 izvršio jako dobro. Međutim *SqueezeNet*, s malim brojem epoha i veličinom paketa, uspio je ostvariti odličnu preciznost validacijskih podataka do skoro 100% ali je zato proces treniranja bio mnogo duži. Proces je trajao 21 minutu što je deset puta duže od *GoogleNet* mreže. Kod povećanja broja epoha i paketa počeo je opadati u preciznosti validacijskih podataka ali je svejedno održao jako dobrih 91% preciznosti. Na slici Slika 3.12 *SqueezeNet* je imao probleme kod prvih slika sa samo 77% i 82,5% preciznosti. Isti takav problem pokazuje i *GoogleNet*. Problem im znaju stvarati slike koje ne ostavljaju dojam za dimenziju dubine ili su fotografirane s prevelike udaljenosti. Uglavnom ih uspiju detektirati ali sa ne tako velikom preciznosti. Da se zaključi, mreža je uspjela ostvariti dobre rezultate i prepoznati jame u tlu.

ZAKLJUČAK

U procesu treniranja mreže kada se postavi mali broj paketa, manji broj epoha i veća stopa učenja *GoogleNet* mreža daje jako loše validacijske rezultate i visoke gubitke ali je vrijeme treniranja jako malo. *SqueezeNet* mreža daje jako precizne rezultate, a greške su približno nuli, međutim vrijeme treniranja je jako dugo. Dešava se razlika kod broja iteracija zato što *GoogleNet* vrši 2 iteracije po epohi, ukupno 12 iteracija, dok *SqueezeNet* stvara veliki broj iteracija i zbog toga proces traje puno duže ali daje preciznije rezultate.

Kada se koristi veliki broj epoha, velike pakete i manja stopa učenja, *GoogleNet* daje bolje rezultate, i stvara manje greške, iako ne daje sveukupno najbolje rezultate, ali je vrijeme treniranja jako malo. *SqueezeNet* ima veće probleme kod preciznosti validacije i validacije treniranja, stvara veće greške, a i vrijeme treniranja je puno duže. *GoogleNet* kod korištenja velikih paketa i više epoha daje bolje rezultate i zato je bolje prilagodljiv kod korištenja velike baze podataka. Također treniranje mreže traje puno manje što je veoma važno kod velikih baza podataka. Za razliku, *SqueezeNet* je manja mreža koja se nastoji prilagoditi računalima s manje snage i memorije pa zbog toga će bolje funkcionirati kod manjih baza podataka. *SqueezeNet* stvara previše iteracija koja bi, kod velikih baza podataka, mogla jako usporiti proces treniranja.

Da se zaključi, kada bi se odredilo koja je neuralna mreža bolja, odgovor daje veličina baze podataka. Također može ovisiti o performansama računala i postavljanju okruženja za izvršavanje treniranja mreže. Ako je u mogućnosti postavljanje jake grafičke kartice za proces, onda se može koristiti *GoogleNet* s velikom bazom podataka. Ako se posjeduje slabija grafička kartica ili se koristi CPU bolji izbor bi bio *SqueezeNet*. Naime, postavljanje opcija treniranja kao što su epohe i veličina paketa mogu zahtijevati ponajviše iskustvo korisnika koji trenira neuralnu mrežu. Isprobavanjem različitih veličina omogućiti će uvid kako neuralna mreža postaje bolja ili lošija i konačno naći optimalnu vrijednost.

LITERATURA

- [1] Torres, Jordi: “*Learning process of deep learning network*“, [Learning Process of a Deep Neural Network | by Jordi TORRES.AI | Towards Data Science](#) 25.Travnja 2021.

- [2] Das,Siddarth: *CNN Architectures: LeNet, AlexNet, VGG, GoogleNet, ResNet and more...*”, [CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more... | by Siddharth Das | Analytics Vidhya | Medium](#), 16.Studenog 2017.

POPIS OZNAKA I KRATICE

CNN konvolucijska neuralna mreža

GD Stupnjevito spuštanje gradijenta ili *Gradient descent*

SGD Stohastičko spuštanje gradijenta ili *Stochastic gradient descent*

SGDM Stohastičko spuštanje gradijenta sa zamahom ili *Stochastic descent with momentum*

SADRŽAJ

Ovaj projekt opisuje cijeli proces treniranja neuralne mreže za prepoznavanje jama u tlu. Polazi od pripreme baze podataka, treniranja mreže, proces treniranja mreže, testiranje mreže i rezultate mreže. Rezultati daju uvid koliko učinkovito, brzo i precizno se može istrenirati potrebna mreža za prepoznavanje objekta i kako se može poboljšati preciznost i kvaliteta prepoznavanja.

Ključne riječi: Neuralna mreža, GoogleNet, SqueezeNet, Detekcija, Jame

SUMMARY

This project describes the whole process of training the neural network to identify pits in the ground. It starts with database preparation, network training, network training process, network testing and network results. The results provide insight into how efficiently, quickly and accurately the required object recognition network can be trained and how the accuracy and quality of recognition can be improved.

Keywords: Neural network, GoogleNet, SqueezeNet, Detection, Pits