UNIVERZA V MARIBORU

FAKULTETA ZA ELEKTROTEHNIKO,

RAČUNALNIŠTVO IN INFORMATIKO

Grega Vrbančič

UPORABA GLOBOKEGA UČENJA S KNJIŽNICO DEEPLEARNING4J NA PRIMERU PREPOZNAVE OBRAZOV

Magistrsko delo

Maribor, avgust 2017

UPORABA GLOBOKEGA UČENJA S KNJIŽNICO DEEPLEARNING4J NA PRIMERU PREPOZNAVE OBRAZOV

Magistrsko delo

Študent:	Grega Vrbančič
Študijski program:	Študijski program 2. stopnje
	Informatika in tehnologije komuniciranja
Mentor:	red. prof. dr. Vili Podgorelec, univ. dipl. inž. rač. in inf.
Lektorica:	Sanja Berend, mag. prof. slov.

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Koroška cesta 46
2000 Maribor, Slovenija

Številka: E5027017
Datum in kraj: 18. 08. 2017, Maribor

Na osnovi 330. člena Statuta Univerze v Mariboru (Statut UM – UPB 12, Ur. l. RS, št. 29/2017) izdajam:

SKLEP O ZAKLJUČNEM DELU

1. **Gregi Vrbančiču**, študentu študijskega programa 2. stopnje MAG INFORMATIKA IN TEHNOLOGIJE KOMUNICIRANJA, se dovoljuje izdelati zaključno delo.

2. Tema zaključnega dela je pretežno s področja Inštituta za informatiko.

MENTOR: red. prof. dr. Vili Podgorelec

3. Naslov zaključnega dela:
UPORABA GLOBOKEGA UČENJA S KNJIŽNICO DEEPLEARNING4J NA PRIMERU PREPOZNAVE OBRAZOV

4. Naslov zaključnega dela v angleškem jeziku:
THE USE OF DEEP LEARNING WITH DEEPLEARNING4J ON THE CASE OF FACIAL RECOGNITION

5. Rok za izdelavo in oddajo zaključnega dela je 18. 08. 2018. Zaključno delo je potrebno izdelati skladno z "Navodili za izdelavo zaključnega dela" in ga v treh izvodih (dva trdo vezana izvoda in en v spiralo vezan izvod) oddati v pristojnem referatu članice. Hkrati se odda tudi izjava mentor-ja/-ice (in morebitnega somentor-ja/-ice) o ustreznosti zaključnega dela.

Pravni pouk: Zoper ta sklep je možna pritožba na Senat članice v roku 10 delovnih dni od dneva prejema sklepa.

Dekan:
red. prof. dr. Borut Žalik

UNIVERZA V MARIBORU FAKULTETA ZA ELEKTROTEHNIKO RAČUNALNIŠTVO IN INFORMATIKO

Obvestiti:
- kandidata,
- mentor-ja/-ico,
- odložiti v arhiv.

Zahvala

Mentorju red. prof. dr. Viliju Podgorelcu se zahvaljujem za vodenje in strokovno pomoč pri izdelavi magistrskega dela.

Najlepša hvala tudi staršem, ki so mi omogočili študij.

Posebna zahvala gre mojemu dekletu, ki me je med študijem podpiralo in mi stalo ob strani.

Uporaba globokega učenja s knjižnico Deeplearning4j na primeru prepoznave obrazov

Ključne besede: strojno učenje, globoko učenje, Deeplearning4j, prepoznava obraza

UDK: 004.932.72'1(043.2)

Povzetek

V magistrskem delu smo se dotaknili področja globokega učenja, spoznali smo pristope in arhitekture algoritmov globokega učenja ter jih kategorizirali v tri skupine. V nadaljevanju smo podrobneje analizirali knjižnico Deeplearning4j, predstavili osnovne funkcionalnosti ter raziskali njene možnosti za uporabo na področju globokega učenja. V praktičnem delu smo uporabo globokega učenja s knjižnico Deeplearning4j aplicirali na primeru prepoznave obrazov. Implementirali smo dve različici konvolucijskih nevronskih mrež ter dva načina učenja – lokalno ter porazdeljeno učenje.

The use of deep learning with Deeplearning4j on the case of facial recognition

Key words: machine learning, deep learning, Deeplearning4j, facial recognition

UDK: 004.932.72'1(043.2)

Abstract

In the master's thesis, we briefly introduced the area of deep learning. We explored and described the approaches and architectures of deep learning algorithms and categorize them into three groups. In the following, we analyzed Deeplearning4j library, presented main features and studied possibilities of its use in the field of deep learning. In the empirical part, the use of deep learning with Deeplearning4j was applied on the case of facial recognition. We implemented two different versions of convolutional neural networks and two types of learning – local and distributed learning.

KAZALO

KAZALO SLIK

KAZALO TABEL

KAZALO PROGRAMSKIH KOD

1 UVOD

Strojno učenje je eno izmed najhitreje razvijajočih se in rastočih področij v računalniški znanosti v zadnjih dveh desetletjih. Ključno vlogo igra v širokem spektru aplikacij, kot so podatkovno rudarjenje, obdelava naravnega jezika, prepoznava slik, ekspertni sistemi in podobno.

Zaradi vedno večje količine podatkov, ki so na voljo, obstaja dober razlog za domnevo, da bo pametna analiza podatkov postala še bolj razširjena in prisotna kot ključna sestavina za tehnološki napredek. [1]

Enega izmed najvidnejših razvojev znotraj področja strojnega učenja v zadnjem desetletju doživlja globoko učenje, ki je z novimi pristopi z uporabo umetnih nevronskih mrež pomembno pripomoglo predvsem na področju računalniškega vida (prepoznava objektov, obrazov itd.). Spletna velikana, kot sta Google in Facebook, imata za namene prepoznave obrazov in objektov s fotografij razvite kompleksne algoritme globokega učenja, na trgu pa obstaja že množica odprtokodnih knjižnic, kot so Tensorflow[1], Caffe[2], Theano[3], Deeplearning4j[4] idr., ki omogočajo razvoj lastnih algoritmov oziroma napovednih modelov. V magistrskem delu se bomo posvetili prav tej knjižnici, jo analizirali in raziskali možnosti njene uporabe ter kot praktični primer uporabe s pomočjo globokega učenja razvili model za prepoznavo obrazov s fotografij.

V magistrskem delu se ukvarjamo s področjem globokega učenja oziroma natančneje z uporabo globokega učenja s knjižnico Deeplearning4j na primeru prepoznave obrazov. Vsebino smo razdelili na štiri logične vsebinske dele. V prvem delu smo raziskali in opisali globoko učenje, njegovo zgodovino, določili terminologijo ter kategorizirali arhitekture oziroma tehnike globokega učenja. V drugem delu smo se posvetili sami knjižnici

[1] Uradna spletna stran knjižnice Tensorflow: https://www.tensorflow.org/
[2] Uradna spletna stran knjižnice Caffe: http://caffe.berkeleyvision.org/
[3] Uradna spletna stran knjižnice Theano: http://www.deeplearning.net/software/theano/
[4] Uradna spletna stran knjižnice Deeplearning4j: https://deeplearning4j.org/

Deeplearning4j; opisali smo arhitekturno zasnovo knjižnice, predelali proces čiščenja, obdelave in transformacije podatkov ter obdelali grajenje nevronskih mrež, porazdeljeno učenje modelov ter vizualizacijo stanja in napredka učenja. V tretjem delu smo predstavili orodja in ogrodja, ki smo jih uporabili za izgradnjo primera globokega učenja ter opisali vzpostavitev lokalnega in porazdeljenega razvojnega okolja. Četrto poglavje vsebuje implementacijo praktičnega primera prepoznave obrazov z uporabo globokega učenja. V tem poglavju sta predstavljeni tako implementacija aplikacije za lokalno učenje kot tudi implementacija aplikacije za porazdeljeno učenje modela. Implementacija vključuje proces pridobivanja, obdelave in transformacije podatkov, opredelitev različnih nevronskih mrež ter evalvacijo rezultatov.

Ob koncu so podani še predlogi za izboljšanje učenja nevronskih mrež in predlogi za nadaljnje delo.

2 GLOBOKO UČENJE

Globoko učenje je podmnožica splošnejšega področja umetne inteligence, ki mu pravimo strojno učenje in temelji na ideji učenja na podlagi primera. Namesto da bi računalnik naučili seznam pravil za reševanje določenega problema, mu pri strojnem učenju podamo model, ki ocenjuje primere in manjši nabor ukazov za prilagajanje modela, kadar se ta zmoti. Pri primerno oblikovanem modelu pričakujemo, da bo bil čez čas sposoben reševati zastavljen problem z izredno natančnostjo. [2]

Do nedavnega je večina strojnega učenja in tehnik procesiranja signalov izkoriščala plitvo strukturirane arhitekture, ki so tipično sestavljene iz enega oziroma največ dveh nivojev nelinearnih transformacij atributov. Primeri takšnih plitvih arhitektur so na primer linearni oz. nelinearni dinamični sistemi, modeli z maksimalno entropijo (angl. Maximum entropy models – MaxEnt), metoda podpornih vektorjev (angl. Support Vector machine – SVM), logistična regresija ipd. Plitve arhitekture so se pokazale kot učinkovite pri reševanju mnogih lažjih ali strogo omejenih problemov. Njihovo omejeno modeliranje in predstavitvena moč pa lahko povzročita težave pri reševanju zapletenejših problemov iz realnega sveta, ki vključujejo naravne signale, kot so človeški govor, naravni zvok in slika ter vizualni prizori. [3]

Človeška mehanizma za procesiranje informacij, kot sta vid in sluh, kažeta na potrebo po uporabi globokih arhitektur za ekstrahiranje kompleksnih struktur in grajenje notranje reprezentacije iz bogatih senzoričnih vhodov. Tako sta recimo pri ljudeh sistema za produkcijo in zaznavo govora opremljena z jasno nivojsko hierarhično strukturo vse od nivoja valovanja impulzov do lingvističnega nivoja. Podobno je tudi pri vizualnem sistemu ljudi, ki je enako kot slušen po naravi hierarhičen. Od tod nam je torej naravno verjeti, da lahko pri reševanju zapletenih problemov iz realnega sveta napredujemo, če se razvijejo učinkoviti globoki učni algoritmi. [3]

2.1 Terminologija globokega učenja

Globoka nevronska mreža (angl. Deep Neural Network – DNN): opredelimo jo lahko kot večplastni perceptron[5] z več skritimi nivoji. Vse uteži nivojev so polno povezane ena z drugo in prejemajo povezave od prejšnjega nivoja. Uteži so inicializirane bodisi z nadzorovanim bodisi nenadzorovanim učenjem. [4]

Mreža globokega prepričanja (angl. Deep Belief Network – DBN): te tipe mrež lahko opredelimo kot verjetnostne generativne modele z vidnimi nivoji in več plastmi skritih latentnih spremenljivk. Vrhnja nivoja imata neusmerjeno, simetrično povezavo, spodnji nivoji pa prejemajo od zgoraj navzdol usmerjene povezave od nivoja nad njimi. [3], [4]

Boltzmanov stroj (angl. Boltzman Machine – BM): mreža simetrično povezanih, nevronom podobnih enot, ki sprejemajo stohastične odločitve, ali bodo v vklopljenem ali izklopljenem stanju. [4]

Omejeni Boltzmanov stroj (angl. Restricted Boltzman Machine – RBM): poseben tip Boltzmanovega stroja, ki je sestavljen iz nivoja vidnih enot in nivoja skritih enot, brez povezav vidnih enot med seboj in tudi brez vidnih povezav skritih enot med seboj. [4]

Globok avtokoder (angl. Deep Autoencoder): je tip avtokoderja, ki ima več skritih nivojev. Tak tip mreže lahko predhodno učimo (angl. pretrain) kot sklad enoplastnih avtokoderjev. Proces učenja je običajno težaven, saj moramo najprej prvi nivo naučiti, da prestrukturira vhodne podatke, ki so nato uporabljeni za učenje naslednjega skritega nivoja, da prestrukturira stanja predhodnega skritega nivoja in tako naprej. [4]

[5] Perceptron: umetni nevron pri nevronskih mrežah

Nevronska mreža s povratno zanko (angl. Recurrent Neural Network – RNN): je tip mreže globokega učenja, ki je posebej uporabna v učenju iz sekvenčnih oz. časovnih podatkov, kot sta govor in video. Primarni koncept RNN je, da opazovanja iz prejšnjega stanja ohranimo za naslednje stanje. [4]

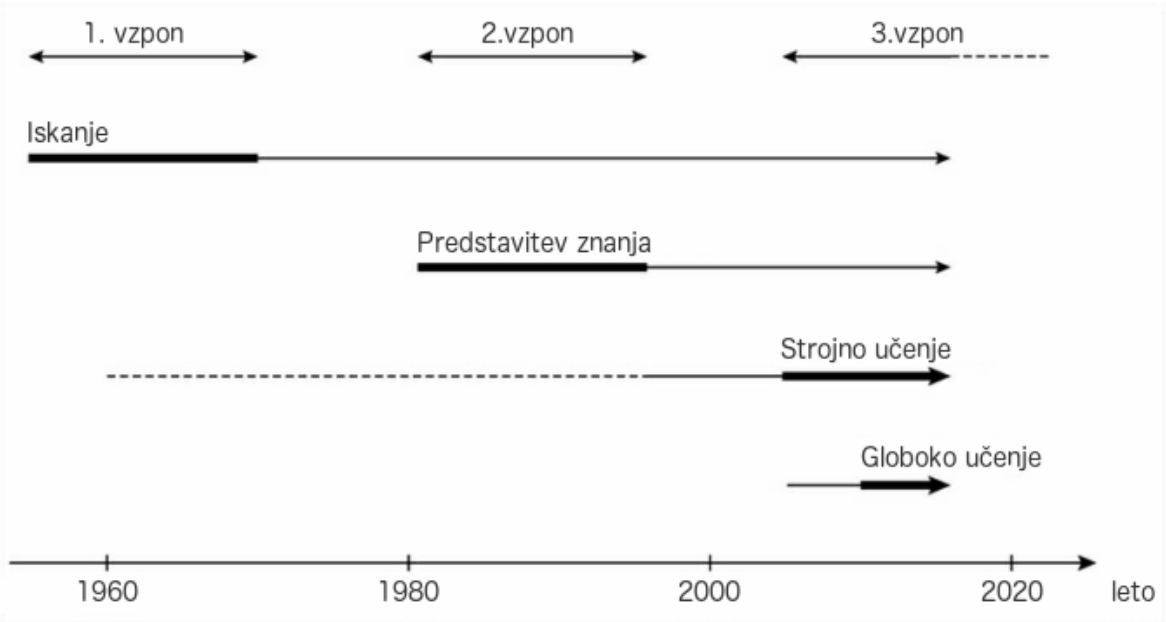
Konvolucijska nevronska mreža (angl. Convolutional Neural Network – CNN): so mreže, katerih nivoji so redko povezani med seboj. Vsak nevron sledečega nivoja je odgovoren samo za del vhoda. Globoke konvolucijske nevronske mreže so dosegle visoke zmogljivosti na področju prepoznave lokacije, klasifikacije slik, prepoznave obrazov ipd. [4]

Porazdeljena zastopanost (angl. Distributed representation): vse globoke mreže večinoma temeljijo na konceptu porazdeljene zastopanosti, ki je jedro teoretične prednosti, ki stoji za uspehom globokih učnih algoritmov. V kontekstu globokega učenja je porazdeljena zastopanost večrazsežnostna zastopanost in je tesno povezana z razsežnostnim modeliranjem teoretične kemije in fizike. Osnovna ideja porazdeljene zastopanosti je, da je zaznana lastnost rezultat več dejavnikov, ki delujejo kot kombinacija za doseganje želenih rezultatov. Primer iz vsakdanjega življenja bi lahko bili človeški možgani, ki uporabljajo porazdeljeno zastopanost za prikrivanje predmetov v okolici. Umetna nevronska mreža te vrste bo zgrajena tako, da bo imela številne značilnosti in nivoje, potrebne za celovito zastopanost našega modela. Model bo opisal podatke, kot so govor, video zapis ali slika, z več med seboj odvisnimi nivoji, kjer bo vsak nivo odgovoren za opis podatkov na drugačni ravni. Na ta način bo predstavitev razdeljena na več nivojev, zato takšno zastopanost označujemo kot porazdeljeno zastopanost. [4]

2.2 Zgodovina

Izraz »globoko učenje« (angl. deep learning) je prvič omenjen leta 2006 v članku profesorjev Hintona in Osindera iz Univerze v Torontu, ki sta v njem predstavila metodo, imenovano *mreža globokega prepričanja* (angl. deep belief network – DBN), ki je razširitev metode nevronske mreže, ki je bila takrat že znana metoda strojnega učenja. [5]

Besedna zveza »globoko učenje« je v današnjih časih pogosto uporabljena kot sopomenka za umetno inteligenco (angl. artificial intelegence – AI). Slednja postaja vedno bolj vroča in priljubljena tema raziskav. To pa ni prvi takšen razcvet umetne inteligence; v zgodovini se raziskave na področju umetne inteligence izvajajo že več desetletij. Skozi desetletja pa je področje prešlo kar nekaj ciklov vzponov in padcev (Slika 2.1).



Slika 2.1: Vzponi umetne inteligence

Strojno učenje je zaznamovalo tretji vzpon umetne inteligence in je kot metoda za podatkovno rudarjenje zelo koristno ter zmogljivo, vendar se je tako kot s predhodnimi pristopi tudi s pristopom strojnega učenja izkazalo, da ne bo dosegel želenega nivoja umetne inteligence. Vse je kazalo, da bo tretji vzpon začel kaj kmalu toniti, vendar presenetljivo – ne le da ni potonil, v bistvu se ni še niti končal in je še dodatno zrasel. Sprožitelj za to rast pa je prav globoko učenje. [5]

Z izumom globokega učenja je postal stroj sposoben vsaj na področju prepoznave slik in govora sam brez pomoči človeka razpoznati, kateri so pomembni atributi v vhodnih podatkih, na katere mora biti pozoren oziroma kateri imajo neko vrednost. Stroj, ki je bil do sedaj zmožen obdelave simbola kot zapisa simbola, je tako postal sposoben pridobivanja konceptov. [5]

2.3 Kategorizacija

Globoko učenje se nanaša na uporabo širokega nabora tehnik strojnega učenja z značilnostjo uporabe mnogih nivojev nelinearnega procesiranja informacij, ki so v naravi hierarhične. Glede na namen uporabe (sinteza oziroma generiranje ali prepoznava oziroma klasifikacija) arhitektur in tehnik je mogoče večino dela na tem področju na splošno razvrstiti v tri glavne razrede: [3]

1. **Globoke mreže za nenadzorovano ali generativno učenje:** njihov namen je zajem korelacij med opazovanimi ali vidnimi podatki za analizo vzorcev ali za namene sinteze, ko niso na voljo nobene informacije o oznakah ciljnega razreda. Kadar se uporabljajo v generativnem načinu, so lahko namenjene tudi opredelitvi skupnih statističnih porazdelitev vidnih podatkov in pripadajočih razredov.

2. **Globoke mreže za nadzorovano učenje:** namenjene so neposredni zagotovitvi moči razlikovanja za klasifikacije vzorcev. Podatki o ciljnih oznakah so za potrebe nadzorovanega učenja vedno na voljo v neposredni in posredni obliki. Pravimo jim tudi diskriminacijske globoke mreže (angl. discriminative deep networks – DDN).

3. **Hibridne globoke mreže:** njihov cilj je razlikovanje, ki je podprto z izhodi generativnih oziroma nenadzorovanih globokih mrež. To lahko dosežemo z boljšo optimizacijo in/ali regularizacijo globokih mrež za nadzorovano učenje. Cilj lahko prav tako dosežemo, kadar razlikovalni kriterij za nadzorovano učenje uporabimo za oceno parametrov pri kateri koli globoki mreži za generativno ali nenadzorovano učenje.

2.4 Globoke mreže za nenadzorovano učenje ali generativno učenje

Pri algoritmih za nenadzorovano učenje ne obstaja želen izhodni podatek v dani vhodni množici podatkov. Sistem se uči prepoznati pomembne lastnosti in vzorce iz njegovih izkušenj, pridobljenih skozi analizo množice podatkov. Namen nenadzorovanega učenja je razumeti strukturo podatkov. Vključuje lahko proces, imenovan gručenje, ki razvršča konstelacijo podatkov v vsaki skupini, ki ima skupno lastnost ali pa proces pridobivanja korelacijskega pravila. [4], [5]

Nenadzorovano učenje torej pomeni, da v procesu učenja niso uporabljene informacije o nadzorovanju posameznih nalog, kot so na primer oznake ciljnega razreda. Mnogo globokih mrež v tej kategoriji se lahko uporabi za smiselno generiranje vzorcev z vzorčenjem iz mrež, kot so na primer omejeni Boltzmanov stroj (angl. restricted Boltzman machine – RBM), mreža globokega prepričanja in globok Boltzmanov stroj (angl. deep Boltzman machine – DBM), ki spadajo v skupino t. i. generativnih modelov. [3]

Med različnimi podrazredi generativnih ali nenadzorovanih globokih mrež so med najpogostejšimi t. i. globoki modeli na osnovi energije (angl. energy-based deep models). Izvirna oblika globokih avtokoderjev je tipični primer te nenadzorovane kategorije modelov. Večina ostalih oblik globokih avtokoderjev je prav tako nenadzorovane narave, ampak s precej drugačnimi lastnostmi in implementacijami. [3]

Še eden izmed vidnejših tipov nenadzorovanih modelov z generativno sposobnostjo je globoki Boltzmanov stroj (DBM), ki je sestavljen iz mnogih slojev skritih spremenljivk, ki so na nivoju posameznega sloja med seboj nepovezane. To je posebna oblika splošnega Boltzmanovega stroja (angl. general Boltzman machine – BM), ki je mreža simetrično povezanih enot, ki spreminjajo stanje (vklop – izklop) na podlagi stohastičnega mehanizma. Ne glede na to, da imajo preprost učni algoritem, so splošni BM zelo kompleksni za razumevanje in se zelo počasi učijo. Če pa število skritih plasti zmanjšamo na eno plast, smo ustvarili t. i. omejeni Boltzmanov stroj (RBM). Enako kot pri globokem Boltzmanovem stroju tudi pri omejenem ni medsebojnih povezav niti med vidnimi niti med nevidnimi enotami. Ključna lastnost RBM se odraža v združevanju številnih RBM. Z oblikovanjem mnogih latentnih nivojev delujejo aktivacijske funkcije kot vhodni podatki za učenje naslednjega nivoja. Takšna arhitektura ustvari drugačno vrsto mrež, ki jim pravimo mreže globokega prepričanja (angl. Deep belief network – DBN). [3], [4]

Naslednja v vrsti prepoznavnejših predstavnikov generativnih mrež, ki jih lahko uporabimo za nenadzorovano (kot tudi nadzorovano) učenje, je t. i. »Sum-Product« mreža (angl. Sum-Product Network – SPN). SPN je globoka mreža, na katero lahko gledamo kot usmerjen aciklični graf, kjer so listi grafa opazovane spremenljivke in notranja vozlišča operatorji vsote in produkta. Vozlišča operatorja vsote predstavljajo mešane modele, vozlišča operatorja produkta pa hierarhijo atributov. Mrežo učimo z uporabo algoritma pričakovanja in maksimizacije (angl. expectation-maximization algorithm – EM) skupaj z uporabo vzvratnega razširjanja napake (angl. backpropagation). [4]

Nevronska mreža s povratno zanko (angl. Recurrent neural network – RNN) je še eden izmed predstavnikov globokih mrež za nenadzorovano učenje (kot tudi nadzorovano), katerih posebnost je, da je njihova globina lahko enaka širini zaporedja vhodnih podatkov. V primeru, da RNN uporabljamo v načinu nenadzorovanega učenja, je RNN navadno na podlagi predhodnih vzorcev podatkov uporabljena za napoved podatkovnega zaporedja v prihodnosti [3]. Razlika med standardnimi nevronskimi mrežami in RNN je, da pri RNN povezave med skritimi plastmi upoštevajo časovno komponento. Vhodni čas t je aktiviran in shranjen na skritem nivoju ob času t in posredovan skritemu nivoju ob času $t+1$ z vhodom ob času $t+1$. To omogoča mreži, da ohranja in odraža stanja preteklih podatkov. Teoretično mora omrežje tako upoštevati celotno zaporedje preteklih časovnih korakov, v praksi pa se uporabljajo časovna okna z določeno dolžino z namenom poenostavitve izračunov. RNN-mreže so bile prilagojene naravni obdelavi jezika (angl. Natural language processing – NLP) in so eden izmed najuspešnejših modelov na tem področju. [5]

2.5 Globoke mreže za nadzorovano učenje

V nasprotju z nenadzorovanim učenjem pri nadzorovanem učenju uporabljamo označene podatke, torej kombinacijo vhodnih in izhodnih podatkov in omembo, kateri vzorec naj bo prepoznan oziroma klasificiran kot posamezni tip podatka. Kadar stroju podamo nepoznane in neoznačene podatke, bo skušal ugotoviti, kateri vzorec lahko aplicira na posamezni podatek in bo podatke razvrstil glede na predhodno označene oz. podane pravilne odgovore. [5]

Tak tip učenja je pogosto uporabljen za reševanje klasifikacijskih problemov. V praksi to pomeni uporabo za zaznavo obrazov, prepoznavo obrazov in podobno. [4]

Mnogo tehnik razlikovanja oz. klasifikacij za nadzorovano učenje imajo plitve arhitekture, podobno kot velja za skriti Markov model (angl. Hidden Markov model – HMM) in CRF (angl. Conditional Random Fields). CRF je sama po sebi plitva klasifikacijska arhitektura, za

katero je značilno linearno razmerje med vhodnimi in prehodnimi atributi. Globoko strukturirani CRF so razviti z zlaganjem izhoda vsakega nižjega sloja skupaj z izvornimi vhodnimi podatki na njegov višji sloj. Mnoge različne verzije globoko strukturiranih CRF so bile uspešno uporabljene za reševanje problemov, kot sta identifikacija govorjenega jezika, obdelava naravnega jezika ipd. [3]

Kot je bilo že omenjeno v prejšnjem poglavju, so lahko tudi RNN-mreže uporabljene za nadzorovano učenje in lahko rešujejo probleme klasifikacijske narave, kjer je izhod povezan z zaporedjem označenih podatkov. Takšni klasifikacijski RNN-modeli so bili aplicirani na problemu govora že v preteklosti, vendar z zelo omejeno uspešnostjo. Vrstili so se tudi poizkusi, kjer so izhod RNN-klasifikacije poskušali mutirati s HMM, vendar nikoli niso povsem uspeli izkoristiti potenciala RNN-mrež. Nedavno je bilo razvitih nekaj novih metod in modelov za RNN-mreže, kjer je osnovna zamisel temeljila na tem, da se izhod RNN upošteva kot nekaj pogojnih porazdelitev, ki se porazdelijo po vseh možnih vhodnih zaporedjih. Te metode so omogočile opravljanje sekvenčne klasifikacije skupaj z vgradnjo LSTM (angl. Long short-term memory – LSTM) arhitekture v model. Glavna korist vgradnje LSTM-arhitekture se odraža v tem, da ni več potrebe po predhodni segmentaciji učnega nabora, prav tako več ni treba izvajati naknadne obdelave izhodnih podatkov. Učinkovitost algoritma te vrste se pogosto uporablja za problem prepoznave rokopisa. [3], [4]

Še eden izmed popularnih tipov arhitektur razlikovanja oz. klasifikacije je konvolucijska nevronska mreža (angl. Convolutional Neural Network – CNN). Pri slednji je vsak modul sestavljen iz enega konvolucijskega sloja in enega združitvenega sloja (angl. pooling layer). Za oblikovanje globokega modela so moduli v splošnem zloženi eden nad drugim. Konvolucijski sloj pripomore k deljenju mnogih uteži, združitveni sloj pa ločuje izhod konvolucijskega in s tem zmanjšuje količino podatkov prejšnjega sloja. CNN-mreže so priznane kot visoko učinkoviti modeli, še posebej pri reševanju problemov prepoznave slik računalniškega vida in podobno. [3], [4]

2.6 Hibridne globoke mreže

Termin »hibridne« za to tretjo kategorijo globokih mrež se nanaša na globoko arhitekturo, ki vsebuje ali uporablja generativne komponente modela in tudi komponente modela za razlikovanje. V obstoječih hibridnih arhitekturah, objavljenih v literaturi, je generativna komponenta najpogosteje izkoriščena za pomoč pri razlikovanju, kar je tudi končni cilj hibridne arhitekture. Kako in zakaj pa lahko generativno modeliranje pomaga pri razlikovanju, je mogoče preučiti z dveh stališč: [3]

1. s stališča optimizacije, kjer lahko generativni modeli, ki so naučeni s tehniko nenadzorovanega učenja, zagotavljajo odlične inicializacijske točke pri zelo nelinearnih problemih ocenjevanja parametrov (predučenje), in

2. s stališča regularizacije, kjer lahko modeli brez nadzorovanega učenja učinkovito zagotovijo nabor lastnosti, ki jih model predstavlja.

Na primer DBN-mrežo, omenjeno v prejšnjem podpoglavju, lahko spremenimo in uporabimo kot začetni model globoke nevronske mreže za nadzorovano učenje z enako strukturo, ki je nadalje učena z uporabo označenih podatkov. Kadar je DBN-mreža uporabljena na tak način, tak model imenujemo DBN-DNN model in predstavlja hibridno globoko mrežo, kjer model, ki je naučen z uporabo nenadzorovanih podatkov, prispeva k učinkovitosti klasifikacijskega modela za nadzorovano učenje. [3]

3 KNJIŽNICA DEEPLEARNING4J

Povsod po svetu so razvili mnogo knjižnic, ki nam lajšajo razvoj modelov nevronskih mrež s pristopom globokega učenja. Če pobližje pogledamo knjižnice oziroma programske jezike, v katerih so bile razvite, opazimo, da je bila večina knjižnic razvita v programskem jeziku Python ali v programskem jeziku C++, na višjem abstraktnem nivoju pa nam ponujajo Pythonov API. Slednje sicer ne preseneča, saj je primarni programski jezik mnogih podatkovnih znanstvenikov in inženirjev prav Python.

Po drugi strani je v gospodarstvu tako v velikih kot tudi manjših podjetjih v veliki meri prisoten programski jezik Java, ki glede na izvedeno anketo na portalu Stack Overflow po priljubljenosti med programskimi jeziki s 39,75 % zaseda tretje mesto [6]. Ker pa je strojno učenje na splošno v sedanjem času aplicirano v obliki mnogih vseprisotnih rešitev, je z namenom lažje integracije v obstoječa okolja podjetij kot ena izmed možnih alternativ na področju knjižnic s podporo strojnemu učenju prav knjižnica Deeplearning4j (v nadaljevanju DL4J).

DL4J je prva komercialna odprtokodna knjižnica za globoko učenje, napisana v programskem jeziku Java in kompatibilna s katerim koli programskim jezikom, ki teče v Javinem navideznem stroju (Scala, Clojure, Kotlin), ob integraciji z orodjem Keras[6] pa omogoča tudi uporabo s programskim jezikom Python. Integracija z orodjem Keras prav tako omogoča uvoz modelov nevronskih mrež iz večine večjih ogrodij, vključno z ogrodji TensorFlow, Caffe, Torch[7] in Theano, ter na ta način zapolnjuje vrzel med ekosistemom Python ter navideznim strojem Java z navzkrižno skupino orodij za podatkovne znanstvenike in inženirje. Knjižnica izkorišča tudi najnovejša porazdeljena računska ogrodja, med drugimi tudi Apache Hadoop[8] (krajše Hadoop) in Apache Spark[9] (krajše

[6] Uradna spletna stran knjižnice Keras: https://keras.io/
[7] Uradna spletna stran knjižnice Torch: http://torch.ch/
[8] Uradna spletna stran orodja Apache Hadoop: http://hadoop.apache.org/
[9] Uradna spletna stran orodja Apache Spark: https://spark.apache.org/

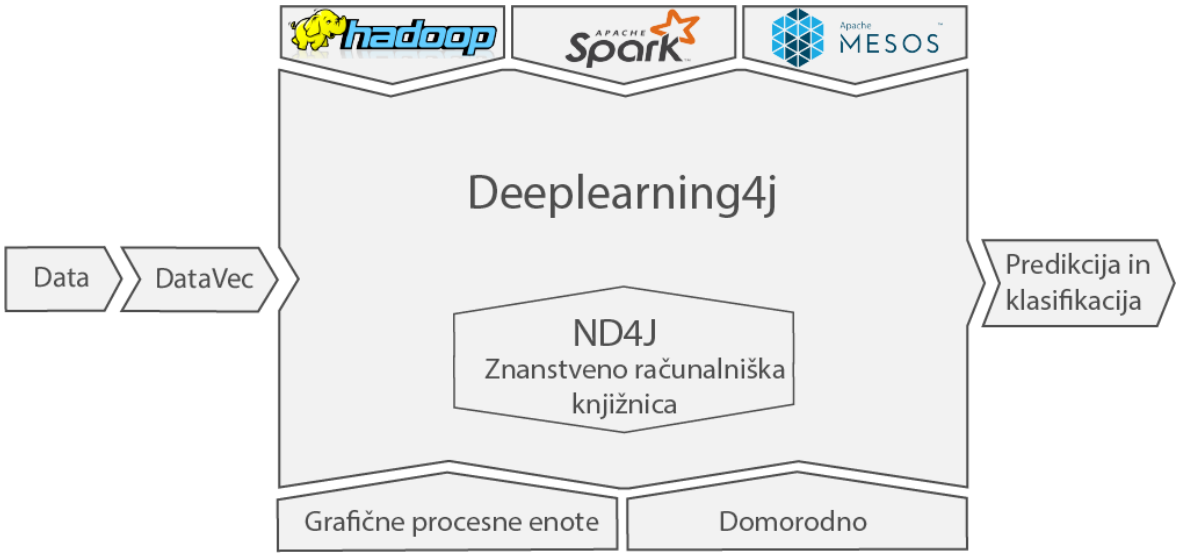
Spark) z namenom pospešitve učenja modelov ter omogoča porazdeljeno učenje tako na centralnih procesnih enotah (angl. Cental Processing Unit – CPU) kot tudi na grafičnih procesnih enotah (angl. Graphics Processing Unit – GPU). [7]

3.1 Zasnova

Knjižnica DL4J je zasnovana z mislijo na porazdeljeno učenje, predvsem integracijo z ogrodjema Hadoop in Spark. Z uporabo omenjenih ogrodij lahko enostavno porazdelimo model ter velike zbirke podatkov in opravljamo operacije vzporedno na več grafično procesnih enotah kot tudi na centralno procesnih enotah. DL4J je v glavnem pokazala znaten uspeh pri prepoznavi vzorcev iz slik, zvoka, besedila itd., uspešno pa rešuje tudi praktične probleme, kot so prepoznava obrazov, zaznavanje prevar, poslovna analitika, priporočilni sistemi, iskanje slik in oglasov ter napovedno vzdrževanje s podatki iz senzorjev. [4]

Splošni visokonivojski arhitekturni model knjižnice (Slika 3.1) prikazuje osnoven tok podatkov ob uporabi DL4J in ključne funkcionalnosti knjižnice. Knjižnica za vektorizacijo podatkov različnih formatov, tipov oz. struktur na vhodu uporablja knjižnico DataVec[10]. Kot pogon za matematične izračune je uporabljena knjižnica N-Dimensional Arrays for Java – ND4J[11]. Omogočeno je vzporedno učenje modelov z uporabo Hadoop YARN ali Spark, prav tako pa knjižnica integrira Cuda kernel za izvajanje operacij tako na porazdeljenih grafično procesnih enotah kot tudi na domorodnih. [4], [8]

[10] Uradna spletna stran knjižnice DataVec: https://github.com/deeplearning4j/DataVec
[11] Uradna spletna stran knjižnice ND4J: https://nd4j.org/



Slika 3.1: Visokonivojski arhitekturni model knjižnice DL4J

DL4J predstavlja knjižnico, ki je bila načrtovana tako za delovanje na več platformah kot tudi za izvajanje v paralelnem oz. vzporednem načinu. V nasprotju s knjižnicami, ki so bile načrtovane v prejšnjem desetletju, ne trpi zaradi problemov paralelizacije. [9]

Razvijalci knjižnice so s samo zasnovo in lastnostmi oz. funkcionalnostmi knjižnice poskrbeli, da se DL4J popolnoma razlikuje od ostalih obstoječih knjižnic, kot so Theano, Torch in podobne. Ključne lastnosti, ki jih je treba izpostaviti in veljajo za jedro knjižnice, so: [4]

1. distribuirana arhitektura;
2. podatkovna vzporednost;
3. sposobnost znanstvene računalniške obdelave za JVM in
4. vektorizacijsko orodje za strojno učenje.

Distribuirana arhitektura

Učenje modelov z DL4J je možno izvajati na dva načina: s porazdeljenim, večnitnim globokim učenjem ali pa s tradicionalnimi enonitnimi tehnikami globokega učenja. Učenje

se izvaja v gručah vozlišč, zato lahko DL4J hitro obdeluje velike količine podatkov. Nevronske mreže so učene vzporedno z uporabo iterativne zmanjševalne metode, ki deluje na Hadoop YARN in Spark. [4]

Operacije knjižnice DL4J lahko poganjamo na Hadoop YARN ali Spark ogrodju kot YARN oz. Spark opravilo (angl. YARN/Spark job), kjer delavska vozlišča (angl. worker nodes) simultano vzporedno obdelujejo podatke. Ko se obdelava zaključi, spremenjene parametre pošljejo nazaj glavnemu vozlišču (angl. master node), kjer se preračuna povprečje parametrov in se nato posodobi model vsakega delavskega vozlišča. [4], [10]

Podatkovna vzporednost

Nevronske mreže lahko učimo na dva načina, in sicer s paralelizacijo podatkov ali paralelizacijo modela, DL4J uporablja prvi način. V podatkovni vzporednosti lahko velike zbirke podatkov razdelimo na manjše zbirke in te porazdelimo modelom, ki tečejo vzporedno na več strežnikih z namenom vzporednega učenja. [4]

Sposobnost znanstvene računalniške obdelave za JVM

Za potrebe znanstvene računalniške obdelave v programskih jezikih, ki tečejo znotraj navideznega stroja Java, knjižnica vključuje oz. uporablja razred N-dimenzionalnega polja knjižnice N-Dimensional Arrays for Java – ND4J. ND4J je knjižnica za znanstveno računalniško obdelavo. Izpostavlja n-dimezionalna polja, ki jih lahko uporabimo v linearni algebri in pri manipulacijah matrik, ki so velikih dimenzij. Predstavlja čisto numerično fasado, ki uporabnikom omogoča menjavo zalednega dela med domorodnimi in grafičnimi procesnimi enotami, ne da bi bilo treba spremeniti implementacijo programske kode [9]. Funkcionalnosti ND4J so mnogo hitrejše kot na primer pri knjižnici Numpy[12] za programski jezik Python, saj so po večini spisane v programskem jeziku C++. Večina rutin ND4J je zasnovana tako, da se izvajajo hitro z minimalnimi zahtevami po sistemskem pomnilniku RAM. [4]

[12] Uradna spletna stran knjižnice Numpy: http://www.numpy.org/

Vektorizacijsko orodje za strojno učenje

Nevronske mreže lahko učimo zgolj na vektorjih, zato je vektorizacija podatkov nujna v fazi predprocesiranja podatkov. DL4J vključuje knjižnico DataVec, ki nam omogoča hitro in enostavno kreiranje vektoriziranih podatkov, primernih za modeliranje oz. uporabo s knjižnico DL4J. [9]

V tabeli (Tabela 3.1) je opredeljena struktura oz. programska zasnova knjižnice po paketih. [5]

Tabela 3.1: Programska zasnova knjižnice po paketih

Paket programske kode:	Opis:
org.deeplearning4j.base	vsebuje razrede za inicializacijo oziroma zagon
org.deeplearning4j.berkeley	vsebuje uporabne podporne matematične metode
org.deeplearning4j.clustering	vsebuje implementacijo postopka k-tih povprečij (angl. k-means clustering)
org.deeplearning4j.datasets	vsebuje razrede in metode za manipulacijo s podatkovnimi zbirkami, vključno z uvozom, kreiranjem, iteriranjem ...
org.deeplearning4j.distributions	vsebuje uporabne podporne metode za porazdelitve
org.deeplearning4j.eval	vsebuje evalvacijske razrede, vključno z matriko zmede (angl. confusion matrix)
org.deeplearning4j.exceptions	vsebuje razrede, ki implementirajo krmilnike izjem (angl. exceptions handlers)
org.deeplearning4j.models	vsebuje algoritme za nadzorovano učenje, vključno z mrežami globokega prepričanja, skladi avtokoderjev, omejenim Boltzmanovim strojem itd.
org.deeplearning4j.nn	vsebuje implementacije komponent in algoritmov, zasnovanih na nevronskih mrežah
org.deeplearning4j.optimize	vsebuje optimizacijske algoritme za nevronske mreže vključno z večslojno optimizacijo, optimizacijo izhodnega sloja …
org.deeplearning4j.plot	vsebuje različne metode za izris podatkov
org.deeplearning4j.rng	generator naključnih podatkov
org.deeplearning4j.util	vsebuje razne podporne metode

3.2 Podatki in ETL-proces

Eden največjih problemov, ki jih je treba rešiti pri strojnem učenju in pravzaprav nima kaj dosti opraviti z nevronskimi mrežami, je problem pridobivanja pravilnih podatkov v pravilni obliki. Globoko učenje oz. na splošno strojno učenje potrebuje za pravilno delovanje dobre učne množice. Pri delu s strojnim učenjem in tudi pri podatkovni znanosti kot taki moramo analizirati vse tipe podatkov. Ključna lastnost je zmožnost predstavitve posameznega podatka v obliki numeričnega vektorja ali v nekaterih primerih kot večdimenzionalnega polja števil. Nevronske mreže še vedno potrebujejo podatke, predstavljene v omenjeni obliki – kot vektorje in matrike, saj ne morejo neposredno delati z besedili, grafi in ostalimi nevektorskimi oz. nematričnimi predstavitvami podatkov. [11],[12]

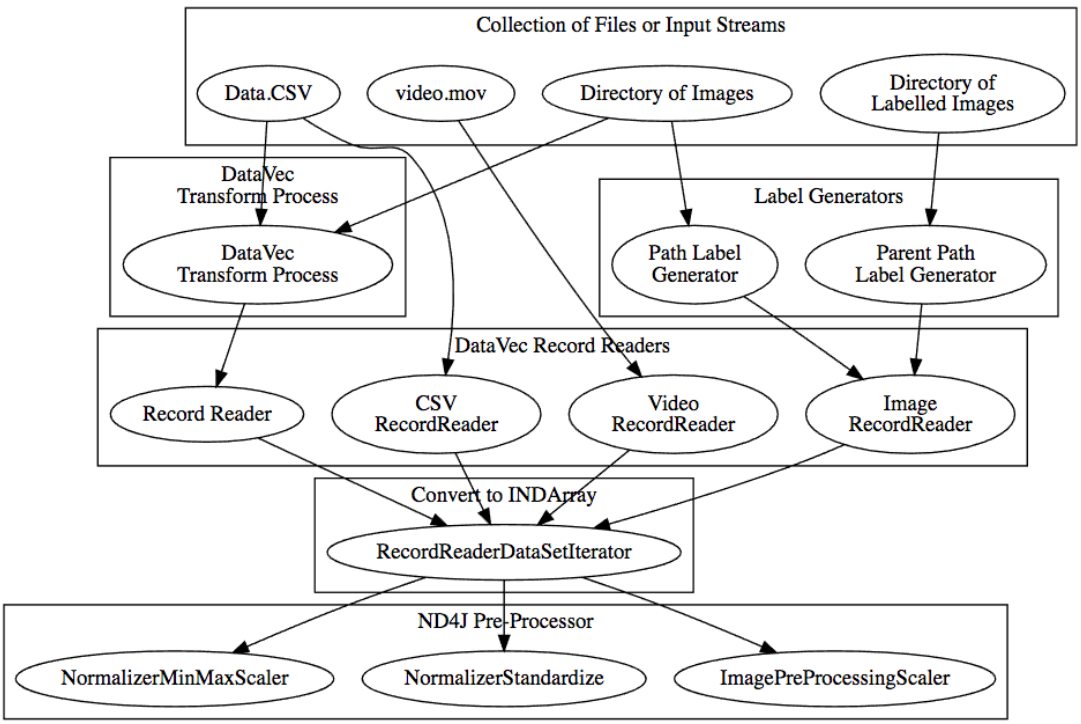
Za rešitev tega problema knjižnica DL4J uporablja DataVec knjižnico, ki omogoča vektorizacijo in predpripravo podatkov – razširitev, transformacijo in nalaganje (angl. Extract, Transform and Load – ETL) podatkov za strojno učenje. DataVec tako omogoča transformacijo neobdelanih podatkov v uporaben vektorski format, primeren za uporabo v različnih orodjih za strojno učenje. Omogoča funkcionalnosti za nalaganje tekstovnih datotek (CVS in podobno), slik in časovnih podatkovnih nizov (angl. time series data sets) za strojno učenje na enem stroju kot tudi za porazdeljeno učenje. [9],[13]

Uporaba knjižnice, kot je DataVec za predprocesiranje podatkov v format, ki ga DL4J lahko brez težav prebere in samodejno vektorizira, je eden izmed primerov dobrih praks za reševanje omenjenega problema. Ročno vektoriziranje neobdelanih podatkov je časovno potratna in zahtevna naloga, kateri se je treba – če je le mogoče – izogniti. [9]

V diagramu (Slika 3.2) so predstavljene možne kombinacije oz. poti pri procesu razširitve, transformacije in nalaganja podatkov. Knjižnica na vhodu dobi podatke v različnih formatih (CSV, avdio, video, direktorij slik itd.), katere lahko nato s pomočjo njenih transformacijskih procesov preoblikujemo.

Pri izdelavi klasifikatorja so oznake izhodne vrednosti, ki jih skušamo napovedovati. V primeru, da je zbirka podatkov zapisana v formatu CSV, so lahko oznake dela samega zapisa shranjene v isti vrstici posameznega vnosa. Če pa je treba oznake generirati, na primer ob branju zbirke direktorijev slik, pa DataVec nudi funkcionalnost generiranja oznak na podlagi poti datoteke z razredoma »PathLabelGenerator« in »ParentPathLabelGenerator«. [14]

S pomočjo vmesnika »RecordReader« podatke pretvorimo v zapis oz. predstavitev, katero lahko preberemo z »RecordReaderDataSetIterator« razredom knjižnice DL4J. Od vmesnika »RecordReader« podatki tipično potujejo k iteratorju po podatkovni zbirki, ki podatke pripravi za posredovanje nevronski mreži. Ko so podatki pripravljeni za uporabo, so že v obliki INDArray in ne več v obliki iteratorja. Funkcionalnosti transformacij in skaliranja podatkov so omogočene tudi na tej stopnji. Ker pa so podatki na tej stopnji že v obliki INDArray, so funkcionalnosti za transformacije in skaliranje omogočene v sklopu knjižnice ND4J (glej Slika 3.2 – ND4J Pre-processor). [14]



Slika 3.2: Diagram možnih poti pri procesu ETL [14]

DataVec tako v grobem omogoča dve kategoriji funkcionalnosti: [9]

- funkcionalnosti za nalaganje podatkov različnih formatov in
- funkcionalnosti za izvajanje splošnih operacij transformacije podatkov.

3.2.1 Nalaganje podatkov

V tem podpoglavju bomo predstavili dva najpogostejša primera branja oz. nalaganja podatkov.

Nalaganje CSV-podatkov

Za branje CSV-podatkov uporabimo razred »CSVRecordReader«, ki implementira že v prejšnjem podpoglavju omenjen vmesnik »RecordReader«. Samo branje (oz. uporaba razreda) je povsem preprosto in zahteva vse skupaj le nekaj vrstic programske kode. Spodnji primer (Programska koda 3.1) prikazuje, kako preberemo datoteko in ustvarimo iterator po prebrani podatkovni zbirki, pripravljeni za učenje nevronske mreže.

Programska koda 3.1: Primer branja CSV-datoteke

```
File file = new File("/path/to/my/file.csv");
int numLinesToSkip = 0;

String delimiter = ",";

RecordReader reader = new CSVRecordReader(numLinesToSkip, delimiter);
InputSplit inputSplit = new FileSplit(file);
reader.initialize(inputSplit);

int minibatchSize = 10;
DataSetIterator dsi = new RecordReaderDataSetIterator(reader, minibatchSize);
```

Če konstruktor razreda »CSVRecordReader« kličemo brez parametrov, je vejica privzeto ločilo (angl. delimiter) med posameznimi podatki v vrstici. Število vrstic, ki jih preskočimo, pa je postavljeno na nič. Pri konstruktorju iteratorja »DataSetIterator« je velikost mini serije (angl. mini batch) poljubno izbrana.

Nalaganje slik

Za potrebe branja oz. nalaganja slik ima knjižnica DataVec implementiran razred »ImageRecordReader«, ki podpira mnogo različnih formatov slik, vključno z najpogostejšimi jpg, gif, png, tiff in bmp. Učinkovito nalaganje in izvajanje operacij nad slikami je omogočeno z uporabo JavaCV[13] in OpenCV[14]. [9]

Implementacija omenjenega razreda omogoča visoko fleksibilnost glede tega, kako želimo, da se podatki naložijo. Spodnji primer (Programska koda 3.2) prikazuje preprosto uporabo razreda za nalaganje in branje slik. Predpostavljamo, da so v tem primeru slikovne datoteke shranjene v direktorijih, kjer ime direktorija predstavlja oznako slik.

Programska koda 3.2: Primer branja slik iz direktorijev

```
int inputNumChannels = 3, outputHeight = 32, outputWidth = 32;
File rootDir = new File("/path/to/my/images_directory/");
String[] allowedExtensions = BaseImageLoader.ALLOWED_FORMATS;
FileSplit inputSplit = new FileSplit(rootDir, allowedExtensions, new
Random());

ParentPathLabelGenerator labelMaker = new ParentPathLabelGenerator();
ImageRecordReader imageReader = new ImageRecordReader(
   outputHeight,
   outputWidth,
   inputNumChannels,
   labelMaker
);
imageReader.initialize(inputSplit);

int minibatchSize = 10;
DataSetIterator dsi = new RecordReaderDataSetIterator(
   imageReader,
   minibatchSize
);
```

[13] Uradna spletna stran knjižnice JavaCV: https://github.com/bytedeco/javacv
[14] Uradna spletna stran knjižnice OpenCV: http://opencv.org/

3.2.2 Transformacije podatkov

Podatki so pogosto namenjeni strojnemu učenju v obliki, ki zahteva določeno mero predprocesiranja, preden jih lahko uporabimo. Predprocesiranje je lahko čisto enostavno, kot je recimo odstranitev nepotrebnih stolpcev iz zbirke podatkov, ali pa zelo kompleksno, kot sta na primer združevanje in čiščenje podatkov iz več neodvisnih podatkovnih virov. DataVec tudi na tej stopnji ponuja funkcionalnosti tako za standardne kot tudi za sekvenčne podatke. [9], [14]

Tipični proces predprocesiranja podatkov z DataVec je sestavljen iz naslednjih korakov:

1. opredelitev sheme za neobdelane vhodne podatke;
2. opredelitev nabora potrebnih operacij za izvajanje (odstranjevanje stolpcev, obvladovanje manjkajočih vrednosti …);
3. nalaganje podatkov;
4. izvajanje operacij in
5. hranjenje procesiranih podatkov.

V kolikor podatki potrebujejo kompleksno predprocesiranje, je priporočljivo proces ločiti od procesa učenja. [9]

»Schema« je v DataVec knjižnici razred, ki opredeljuje tri lastnosti naših podatkov:

- ime posameznega stolpca;
- tip posameznega stolpca (numeričen, kategorični, niz itd.) in
- omejitve (če obstajajo) nad dovoljenimi vrednostmi.

Za sekvenčne podatke je na voljo razred »SequenceSchema«, ki vsebuje enake informacije za posamezen stolpec kot prej omenjeni razred »Schema«.

»TransformProcess« razred opredeljuje zaporedje operacij, ki jih je treba izvesti nad podatki. Sestavljen je z opredelitvijo dveh stvari:

- s shemo izvornih vhodnih podatkov in
- z naborom operacij, ki jih želimo izvesti.

DataVec ponuja mnogo tipov operacij, ki so izvršljive nad podatki. Povzetek teh operacij je predstavljen v spodnji tabeli (Tabela 3.2). [9]

Tabela 3.2: DataVec tipi operacij

Ime:	**Opis:**	**Primer uporabe:**
Transformacija	splošna operacija za posamezen primerek ali sekvenco	odstranjevanje stolpcev, matematične operacije, razčlenjevanje podatkov
Filter	odstranitev primerkov, ki ustrezajo pogoju	filtriranje manjkajočih ali nepravilnih vrednosti
Zmanjšanje	grupiranje primerkov po ključu in zmanjšanje	minimum, maksimum ali vsota vrednosti za vsak posamezen identifikator stranke
Pretvorba v sekvenco	grupiranje posameznih primerkov v sekvenco z enim ali več ključnimi stolpci	dnevniški podatki: grupiranje posameznih zapisov v sekvenco za vsak IP
Pretvorba iz sekvence	razdelitev vsakega časovnega koraka v sekvenčnih podatkih v posamezno (nesekvenčno) obliko	razdelitev sekvenc transakcij po meri v ločene zapise

3.3 Grajenje nevronskih mrež

Knjižnica DL4J ima za grajenje in učenje nevronskih mrež implementirana dva razreda:

1. **MultiLayerNetwork:** nevronska mreža, zgrajena iz več slojev in navadno z enim izhodnim slojem. Učimo jo s pristopom vzvratnega razširjanja napake (angl. backpropagation) z opcijskim predučenjem – odvisno od tipa nivojev, ki jih mreža vsebuje. [15]
2. **ComputationGraph:** namenjen je nevronskim mrežam s kompleksnejšo arhitekturo povezav. Dovoljuje poljubno usmerjeno aciklično vezno strukturo med posameznimi nivoji ter ima lahko poljubno število vhodov in izhodov. [15]

Za grajenje nevronske mreže s katerim koli izmed prej navedenih razredov je treba naprej začeti s konfiguracijskim razredom. Za grajenje z razredom »MultiLayerNetwork« moramo uporabiti konfiguracijski razred »MultiLayerConfiguration«, za grajenje z razredom »Computation-Graph« pa »ComputationGraphConfiguration«. S tema konfiguracijskima razredoma opredelimo globino nevronske mreže, lastnosti posameznih slojev, aktivacijske funkcije, uteži, izhode …

Naslednji primer (Programska koda 3.3) prikazuje, kako opredelimo konfiguracijo nevronske mreže z dvema skritima nivojema z uporabo razreda »MultiLayerConfiguration«.

Pri obeh tipih navadno fiksno nastavimo naključno seme (angl. random seed), s čimer omogočimo reprodukcijo rezultatov. Naključno seme pa je uporabljeno za inicializacijo uteži nevronske mreže. [15]

Programska koda 3.3: Primer konfiguracije nevronske mreže

```
MultiLayerConfiguration configuration = new NeuralNetConfiguration.Builder()
   .seed(seed)
   .iterations(iterations)
   .activation(Activation.RELU)
   .weightInit(WeightInit.XAVIER)
   .learningRate(0.1)
   .regularization(true).l2(1e-4)
   .list()
   .layer(0, new DenseLayer.Builder().nIn(numInputs).nOut(3).build())
   .layer(1, new DenseLayer.Builder().nIn(3).nOut(3).build())
   .layer(2, new OutputLayer.Builder
                (LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
                .activation(Activation.SOFTMAX)
                .nIn(3).nOut(numOutput).build())
   .backprop(true).pretrain(false)
   .build();
```

Z iteracijami oz. ponovitvami opredelimo število posodobitev parametrov modela nevronske mreže. Ena ponovitev predstavlja eno posodobitev parametrov. Ponovitve ne smemo zamenjati z epoho (angl. epoch), ki predstavlja en popoln prehod skozi podatkovni niz. Do ponovitve epohe se lahko pojavijo številne ponovitve. Epoha in ponovitev sta samo sinonima v primeru, če za vsak prehod skozi celoten niz podatkov samo enkrat posodobimo parametre. Če pa posodabljamo z uporabo mini serij (angl. mini batch), pa pomenita nekaj drugega. Poglejmo ob primeru, če so naši podatki razdeljeni na dve mini seriji A in B. Če nastavimo število ponovitev na 3, se bo učenje izvajalo na način AAABBB medtem ko bi se učenje izvajalo na način ABABAB v primeru, da bi število epoh nastavili na 3. [15]

Z aktivacijsko funkcijo določamo, kakšen izhod bo generiralo vozlišče glede na njegov vhod. Pri DL4J opredelimo aktivacijsko funkcijo za celoten sloj in uporabi se na vseh nevronih tega sloja. Aktivacijska funkcija izhodnega sloja določa izhod nevronske mreže.

Tako recimo za generiranje binarnih izhodov uporabljamo sigmoidno aktivacijsko funkcijo, za klasifikacijo uporabljamo softmax funkcijo, ki ji nastavimo število klasifikacijskih razredov itd. Poleg vgrajenih aktivacijskih funkcij DL4J omogoča tudi opredelitev lastnih funkcij. [15]

Na začetku so uteži nevronske mreže naključno inicializirane. Naključnost je pomembna, saj se dva nevrona z identično utežjo nikoli ne bosta sposobna razlikovati in naučiti različnih lastnosti. Tipično sta metodi »UNIFORM« in »XAVIER« dobri začetni točki za inicializacijo uteži. [15], [16]

Stopnja učenja (angl. Learning rate) določa, kako velik naj bo posamezen korak na poti proti dosegu čim manjše napake ob posodobitvi uteži. Če je stopnja učenja nastavljena na preveliko vrednost, lahko napaka močno niha ali se ne približa, medtem ko se premajhna vrednost stopnje učenja odraža v tem, da mreža za učenje po nepotrebnem porabi veliko časa. Za prilagajanje hitrosti učenja se uporabljajo prilagodljive posodobitvene funkcije, kot sta »ADAM« ali »AdaGrad«. [15]

Z nastavitvijo parametra regularizacije se borimo proti težnji nevronskih mrež, da se prenatrenirajo (angl. overfit) z učnimi podatki. Do prenatreniranosti pride, ko se mreža izkaže za zelo natančno pri uporabi učnih podatkov, obenem pa se ne uspe dovolj posplošiti oz. generalizirati in se tako posledično slabo odreže pri testnih podatkih. [15]

Po končani konfiguraciji nevronske mreže, kreiramo novo instanco razreda »MultiLayerNetwork«, kateremu kot parameter konstruktorja podamo prej opredeljeno konfiguracijo mreže. Novokreirani instanci je mogoče opredeliti tudi poslušalce, ki zbirajo statistiko ali poročila o napredku učenja. V primeru (Programska koda 3.4) je prikazano, kako uporabimo »ScoreIterationListener« poslušalca, ki ob poljubno nastavljeni frekvenci v konzolo izpiše trenutno natančnost, medtem ko se mreža uči. Na posameznem modelu lahko uporabimo seznam več različnih poslušalcev. Med drugimi lahko uporabimo tudi

poslušalca, ki posreduje podatke o učenju oz. napredku grafičnemu vmesniku Deeplearning4j UI (glej poglavje 3.5).

Programska koda 3.4: Primer uporabe poslušalca za izpis natančnosti učenja

```
MultiLayerNetwork network = new MultiLayerNetwork(configuration);
network.setListeners(new ScoreIterationListener(frequency));

network.fit(dsi);

ModelSerializer.writeModel(network, new File(System.getProperty(USER_HOME),
"model.zip"), true);

Evaluation evaluation = network.evaluate(testDsi);
```

S klicem metode »fit« razreda »MultiLayerNetwork«, kateri kot parameter podamo učno množico podatkov, pričnemo z učenjem nevronske mreže. Po koncu učenja imamo možnost shraniti mrežo s pomočjo podpornega razreda »ModelSerializer«, uspešnost mreže pa preverimo z »evaluate« metodo, kateri podamo testno podatkovno množico.

3.4 Porazdeljeno učenje modelov

DL4J podpira porazdeljeno učenje nevronskih mrež na gruči Spark strežnikov s ciljem pospešitve učenja mreže. Podobno kot pri razredih »MultiLayerNetwork« in »ComputationGraph« ima ogrodje opredeljena razreda za učenje nevronskih mrež na Sparku:

- **SparkDl4jMultiLayer** razred, ki je ovojnica okrog razreda »MultiLayerNetwork«;
- **SparkComputationGraph** razred, ki je ovojnica okrog razreda »ComputationGraph«. [10]

Ker sta omenjena razreda samo ovojnici okrog standardnih razredov za učenje na enem računalniku, je proces konfiguracije z razredoma »MultiLayerConfiguration« in »ComputationGraphConfiguration« povsem enak tudi pri porazdeljenem učenju. Slednje se na Sparku razlikuje od lokalnega učenja v dveh pogledih, in sicer po načinu, na katerega so podatki naloženi, in kako pripravimo učenje – potrebno je nekaj dodatne konfiguracije, specifične za porazdeljeno učenje na gruči.

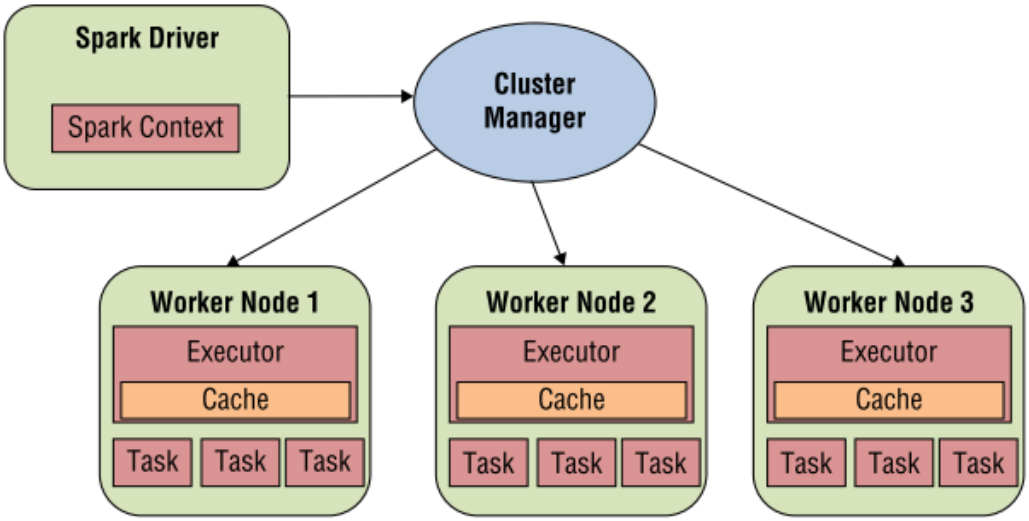
Tipičen proces za učenje mrež na gruči Spark poteka tako:

1. **Kreiranje učnega razreda mreže**. Na splošno ta korak zajema:
   - opredelitev konfiguracije mreže z enim izmed konfiguracijskih razredov (podpoglavje 3.3) na enak način kot pri lokalnem učenju;
   - kreiranje instance razreda TrainingMaster, ki opredeljuje, kako bo porazdeljeno učenje izvedeno v praksi (razlaga v nadaljevanju);
   - kreiranje instance enega izmed prej omenjenih razredov ovojnic z uporabo konfiguracijskih razredov;
   - nalaganje podatkov;
   - klicanje primerne »fit« metode;
   - hramba oziroma uporaba naučenega modela.
2. **Kreiranje jar datoteke**, pripravljene za uporabo na gruči Spark.
3. **Izvedba ukaza za zagon na gruči Spark.**

3.4.1 Arhitektura gruče Spark

V diagramu (Slika 3.3) so prikazane glavne komponente, ki izvršujejo zadane naloge z orodjem za upravljanje z gručo Spark, v samem ekosistemu Spark pa je več entitet, ki sodelujejo pri izvajanju aplikacije. Na visokem nivoju se vse začne z gonilnikom Spark (angl. Spark driver), ki je v bistvu nadzornik, ki vzdržuje povezave z drugimi entitetami v gruči in gruči delavskih vozlišč pošilja naloge v izvedbo (angl. Worker nodes). Delavska

vozlišča po drugi strani izvršujejo izvajalske procese, ki imajo eno ali več nalog in ki dejansko izvršujejo programsko kodo, celota katere sestavlja opravilo Spark. [17]



Slika 3.3: Arhitektura gruče Spark [17]

Gonilnik

Gonilnik je proces, odgovoren za zagon in upravljanje aplikacij Spark. Kot glavni proces izpolnjuje vrsto odgovornosti, v prvi vrsti pa vzdržuje oz. ohranja Spark kontekst – stanje programa, ki mu omogoča dodeljevanje nalog izvršiteljem (angl. Executors) ter tudi ohranjanje notranjih konstruktov. Ta kontekst je tisto, kar sledi aplikacijskim nastavitvam in razpoložljivim virom. Gonilnik prav tako upravlja komunikacijo z upravljavcem gruče in zahteva sredstva za izvedbo aplikacije. Ko so potrebna sredstva na voljo, gonilnik ustvari izvedbeni načrt, ki temelji na logični kodi aplikacije Spark, in ga pošlje dodeljenim delavskim vozliščem v obdelavo. Izvedbeni načrt je usmerjeni aciklični graf (angl. Directed Acyclical Graph – DAG) ukazov in transformacij. Spark optimizira graf z namenom zmanjšanja premikanja podatkov, nato pa ga notranji konstrukt »DAG Scheduler«[15] razdeli na individualne stopnje oz. faze in nato še v posamezne naloge. Stopnja oz. faza je niz transformacij, ki jih je treba izvršiti nad podatki, ki jih vsebuje RDD[16]. [17]

[15] »DAGScheduler« je razred, ki skrbi za razdelitev usmerjenega acikličnega grafa na faze oz. naloge ter urnike njihovega izvajanja.
[16] RDD je nespremenljiva porazdeljena zbirka objektov in osnovna podatkovna struktura Sparka. [37]

Podatki so razčlenjeni na particije, kar pomeni, da čeprav obstaja ena sama koherentna podatkovna množica, Spark ne deluje nad celotno množico naenkrat. Ko je posamezna faza razdeljena na naloge »DAG Scheduler«, za vsako nalogo ustvari RDD-particijo, tako da vsaka posamezna naloga izvaja isto operacijo, vendar na drugem delu podatkov. Rezultat vsake posamezne naloge je pozneje združen v en sam RDD. [17]

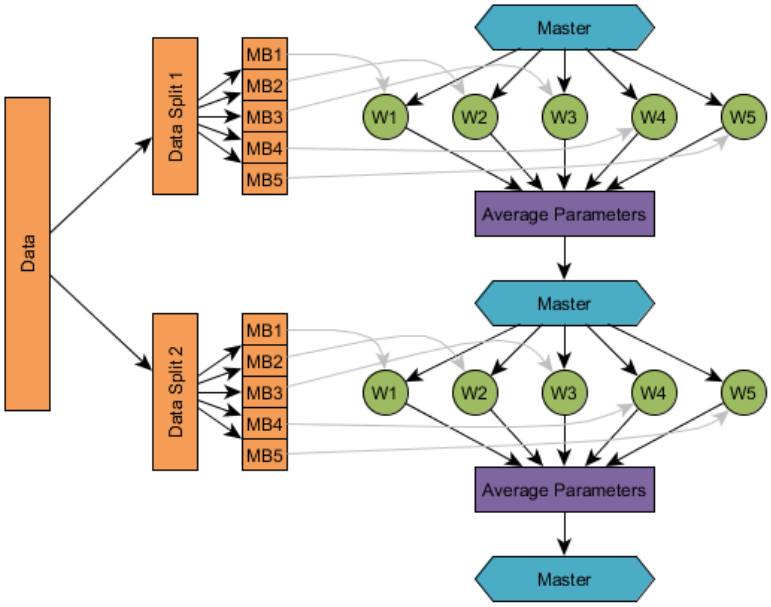
Delavci in izvrševalci

V gruči Spark so delavska vozlišča fizični stroji, ki poganjajo izvrševalce in naloge. Vsako delavsko vozlišče ima določen nabor virov, ki so na voljo, in ti viri so izrecno dodeljeni upravitelju gruče. Vsako delavsko vozlišče v gruči Spark lahko poganja še enega izvrševalca. Izvrševalci so abstrakcija, ki omogoča nastavljivo izvedbo aplikacije Spark. Vsak izvrševalec poganja samostojen navidezni stroj Java, ki ima določen nabor virov. Viri, kot so procesorska jedra in količina pomnilnika ter tudi število izvrševalcev aplikacije Spark, so nastavljivi parametri in imajo pomemben vpliv na izvedbo. Spark lahko paralelizira le v tolikšni meri, kot mu to dopušča konfiguracija. Če imamo na primer dodeljenega samo enega izvrševalca in zanj dve procesorski jedri, lahko Spark na tem izvršitelju sočasno izvaja samo dva procesa. [17]

3.4.2 Izvedba učenja

Knjižnica DL4J uporablja proces »povprečenja« parametrov (angl. parameter averaging) za učenje nevronskih mrež. Proces izvedbe učenja mreže na gruči Spark pa je tak, da gonilnik prične z inicializacijo konfiguracije mreže in parametrov. Zatem je podatkovna množica razdeljena na več podmnožic, skozi katere iteriramo. Za vsako podmnožico v iteraciji se nato porazdelijo konfiguracija in parametri iz gonilnika na vsa delavska vozlišča. Vsako delavsko vozlišče nato izvede »fit« metodo nad njegovo podmnožico podatkov. Nato se »povprečijo« parametri (če je določeno, se posodobi tudi stanje) in povprečje se vrne nazaj gonilniku. Učenje je zatem zaključeno, gonilnik pa ima kopijo naučene nevronske mreže. [10]

Spodnji primer (Slika 3.4) prikazuje proces s petimi delavci, kjer je tako kot pri lokalnem učenju učna podatkovna množica razdeljena na številne podmnožice – mini serije. Učenje se izvaja nad vsako podmnožico, kjer ima vsak delavec svojo podmnožico za učenje. [10]



Slika 3.4: Izvedba porazdeljenega učenja na gruči Spark [10]

Če bi torej želeli preoblikovati prejšnji primer lokalnega učenja, bi to storili na naslednji način (Programska koda 3.5). »TrainingMaster« je v DL4J vmesnik, ki omogoča uporabo različnih učnih implementacij z razredoma »SparkDl4jMultiLayer« in »SparkComputationGraph«. Trenutno je implementirana samo »ParameterAverageTrainingMaster«, to je implementacija prej opisanega procesa učenja.

Programska koda 3.5: Primer porazdeljenega učenja na gruči Spark

```
SparkConf sparkConf = new SparkConf();
JavaSparkContext javaSparkContext = new JavaSparkContext(sparkConf);

JavaRDD<DataSet> trainDataSetRDD = javaSparkContext.parallelize(trainData);

TrainingMaster trainingMaster = new
ParameterAveragingTrainingMaster.Builder(numWorkers, batchSize)
   .averagingFrequency(5)
   .workerPrefetchNumBatches(2)
   .batchSizePerWorker(batchSize)
   .storageLevel(StorageLevel.MEMORY_AND_DISK())
   .build();

SparkDl4jMultiLayer sparkDl4jMultiLayer = new
SparkDl4jMultiLayer(javaSparkContext, network, trainingMaster);
sparkDl4jMultiLayer.setListeners(new ScoreIterationListener(frequency));

sparkDl4jMultiLayer.fit(trainDataSetRDD);

ModelSerializer.writeModel(network, new File(System.getProperty(USER_HOME),
"model.zip"), true);

Evaluation evaluation = sparkDl4jMultiLayer.evaluate(testDataSetRDD);
```

Kot je razvidno iz primera, smo nastavili frekvenco »povprečenja« parametrov, opredelili, koliko mini serij podatkov si vsak delavec lahko vnaprej pridobi, opredelili tudi velikost posamezne mini serije ter nastavili nivo hrambe podatkov.

Poleg uporabljenih nastavitev učenja nam razred omogoča številne druge možnosti konfiguracije, s katerimi nadzorujemo, kako naj bo učenje izvedeno. V tabeli (Tabela 3.3) je prikaz pogosto uporabljanih možnosti pri upravljanju porazdeljenega učenja.

Tabela 3.3: Seznam metod razreda ParameterAveragingTrainingMaster

Metoda	Opis:
batchSizePerWorker	Nadzoruje velikost mini serije za posameznega delavca.
averagingFrequency	Frekvenca, kako pogosto so parametri povprečeni in redistribuirani,
workerPrefetchNumBatches	Število asinhrono prednaloženih mini serij v izogib čakanju, da se naslednja mini serija naloži.
saveUpdater	Hranjenje stanja učnih metod, kot so »momentum«, »RMSProp« in »AdaGrad«.
rddTrainingApproach	Nastavitev pristopa učenja nad RDD-množico. Na voljo sta neposreden pristop, ki hrani vse podatke v pomnilniku, in pristop z izvozom, kjer so podatki razčlenjeni v podmnožice in nato serializirani na disku. Posamezne podmnožice se nato nalagajo iz diska.
exportDirectory	V primeru uporabe pristopa učenja z izvozom podatkov omogoča opredelitev lokacije, kje naj bodo podatki shranjeni.
storageLevel	Določa nivo hrambe podatkov (pomnilnik, pomnilnik in trdi disk, trdi disk).

3.4.3 Zagon učenja na gruči Spark

Za zagon učenja implementirane nevronske mreže je treba celotno implementacijo z vsemi uporabljenimi knjižnicami zapakirati v jar datoteko. Navadno je jar datoteko najenostavneje ustvariti z uporabo orodja Maven in vtičnikom za izgradnjo. To jar datoteko je treba potem prenesti na vsa delavska vozlišča oz. jo narediti dostopno vsem vozliščem. [18]

Ogrodje Spark se namesti s pripadajočimi izvedbenimi skriptami, ki se nahajajo v mapi »bin« namestitvenega direktorija ogrodja. Za zagon učenja na gruči Spark se najpogosteje uporablja skripta »spark-submit«. Skripta poskrbi za pravilno nastavitev pravilne poti do razredov (angl. classpath) in podpira zagon učenja na vseh upravljavcih, ki jih podpira ogrodje Spark. [19]

Spodnji izvršilni ukaz »spark-submit« skripte (Programska koda 3.6), ki ga izvedemo v terminalu, prikazuje izvedbo aplikacije na gruči Spark. Opredeljena je možnost »--classpath«, s katero določimo pot do razreda, ki ima implementirano vstopno »main« metodo. Z opcijo »--master« opredelimo naslov glavnega strežnika Spark, z »--deploy-mode« pa povemo, ali želimo, da se aplikacija izvede na gruči ali na lokalnem računalniku. Zatem opredelimo pot do jar datoteke in morebitne zagonske argumente aplikacije.

Programska koda 3.6: Primer zagona učenja na gruči Spark

```
./bin/spark-submit \
  --class si.um.vrbancic.Primer \
  --master spark://192.168.1.1:6060 \
  --deploy-mode cluster \
  /spark/jars/primer.jar \
  -useSparkLocal false
```

3.5 Vizualizacija

Knjižnica DL4J v obliki modula – Deeplearning4j UI prek uporabniškega vmesnika v brskalniku omogoča vizualizacijo trenutnega stanja nevronske mreže in napredka učenja. Uporabniški vmesnik se navadno uporablja za pomoč pri finih nastavitvah učenja mrež z namenom doseči čim boljšo zmogljivost mreže.

Če želimo v aplikaciji omogočiti Deeplearning UI modul za vizualizacijo napredka učenja nevronske mreže, je treba najprej vključiti odvisnostno knjižnico »deeplearning4j-ui«. V primeru (Programska koda 3.7) je prikazan način, kako se vključi oz. integrira vizualizacija napredka učenja nevronske mreže. Inicializirati je treba zaledni sistem uporabniškega vmesnika za vizualizacijo, saj deluje ta kot samostojna spletna aplikacija.

Nastaviti je treba, kam bomo hranili podatke o učenju mreže. Hranimo jih lahko v pomnilniku za sproten pregled ali pa v datoteko, katero naložimo pozneje. Hrambo podatkov je treba nato povezati z instanco zalednega sistema uporabniškega vmesnika ter konfigurirani nevronski mreži dodati poslušalca, ki bo uporabniškemu vmesniku pošiljal podatke o napredku učenja.

Programska koda 3.7: Primer vključitve vizualizacije

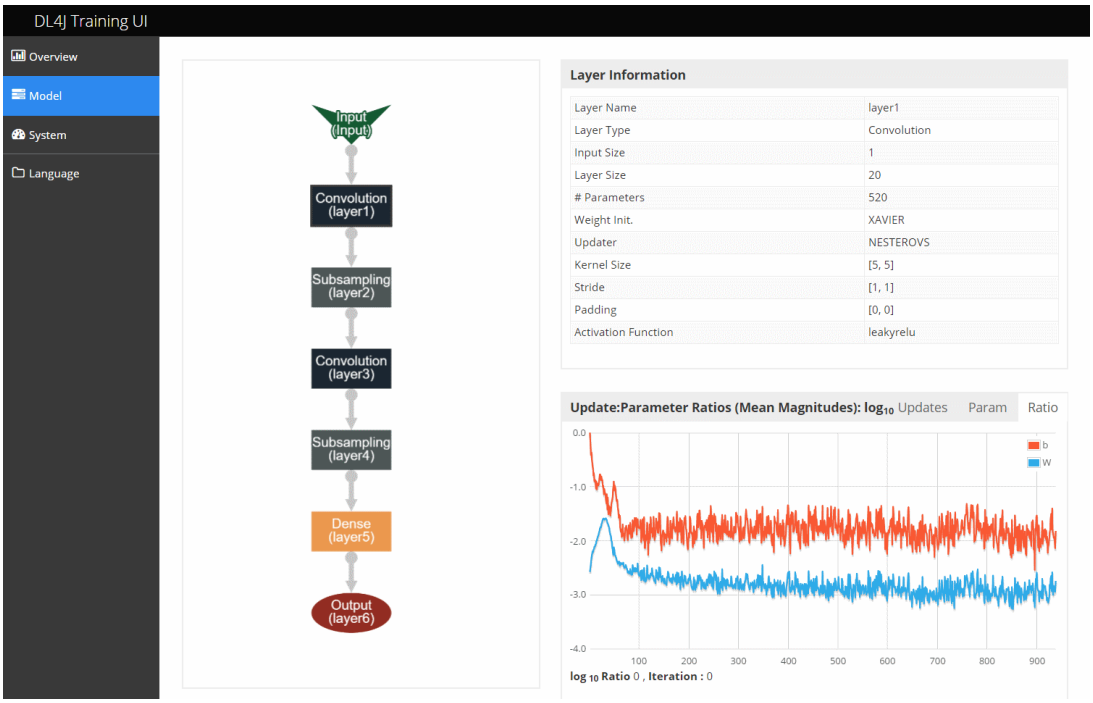
```
UIServer uiServer = UIServer.getInstance();

StatsStorage statsStorage = new InMemoryStatsStorage();

uiServer.attach(statsStorage);

network.setListeners(new StatsListener(statsStorage));
network.fit(dataset);
```

Uporabniški vmesnik je po zagonu aplikacije prek brskalnika dostopen na lokalnem naslovu računalnika na vratih 9000.

Uporabniški vmesnik je sestavljen iz treh pogledov: iz splošnega pogleda (Slika 3.5), pogleda modela (Slika 3.6) ter sistemskega pogleda (Slika 3.7). V okvirčku zgoraj levo je graf, ki prikazuje razmerje med oceno in iteracijami – vrednost funkcije napake za trenutno mini serijo. Zgoraj desno so prikazane informacije o modelu in o učenju, v spodnjem levem okvirčku je prikazano razmerje parametrov za posodobitve (po nivojih) za vse uteži mreže glede na iteracijo, v desnem okvirčku pa je graf standardnega odklona glede na aktivacije, gradiente in posodobitve.

Spodnja dva grafa imata vrednosti predstavljene v obliki logaritma z osnovo 10. Torej na primer vrednost –3 na grafu spodaj levo ustreza razmerju 10^{-3} = 0,001. Razmerje med parametri in posodobitvami je specifično razmerje med srednjimi vrednostmi teh vrednosti. [20]

Slika 3.5: Uporabniški vmesnik modula Deeplearning4j UI – splošen pregled [20]

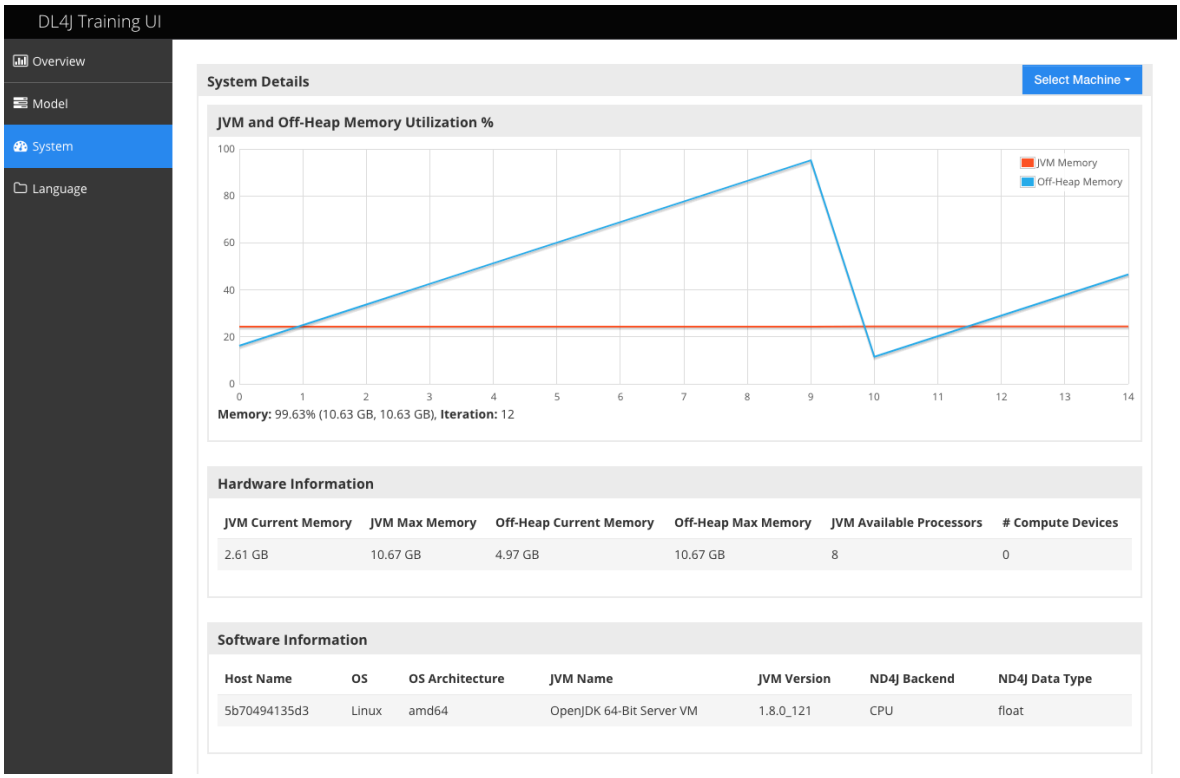


Slika 3.6: Uporabniški vmesnik modula Deeplearning4j UI – pregled modela [20]

Pogled s podrobnostmi o modelu (Slika 3.6) vsebuje diagram nivojev nevronske mreže, ki delujejo kot mehanizem za izbiro. Za pregled podrobnejših informacij o posameznem nivoju je treba klikniti na želen nivo v diagramu. Na desni strani je zgoraj izpisana tabela z informacijami o nivoju, pod njo pa je vsebnik s tremi grafi. Na prvem je izrisano razmerje posodobitev glede na parametre za vse uteži za izbrani nivo, na drugih dveh pa sta dostopna grafa razmerij parametrov in posodobitev srednjih vrednosti.

Na grafih so parametri označeni z *W* za uteži in z *b* za dodano korekcijo (angl. bias). Pri RNN-mrežah pa *W* predstavlja uteži, ki povezujejo nivo z nivojem pod njim in *RW* predstavlja ponavljajoče se uteži (angl. recurrent weights). [20]

Pogled s podrobnostmi o sistemu (Slika 3.7) vsebuje graf, ki prikazuje porabo pomnilniških virov navideznega stroja Java ter tabelo z informacijami o strojnih in programskih pomnilniških virih.



Slika 3.7: Uporabniški vmesnik modula Deeplearning4j UI – pregled sistema

3.5.1 Vizualizacija in Spark

Modul Deeplearning4j UI lahko uporabimo tudi pri porazdeljenem učenju na gruči Spark, vendar pa je zaradi konfliktov v odvisnostnih knjižnicah poganjanje modula za vizualizacijo na istem navideznem stroju Java lahko težavno. Za rešitev tega problema sta na voljo dve rešitvi, in sicer lahko zbiramo in shranimo podatke o učenju ter shranjene podatke vizualiziramo pozneje, lahko pa modul z uporabniškim vmesnikom poganjamo ločeno z uporabo funkcionalnosti oddaljenega uporabniškega vmesnika, kjer se podatki s strežnika, ki opravlja proces učenja modela, pošiljajo strežniku, ki poganja uporabniški vmesnik. [20]

Primer (Programska koda 3.8) prikazuje shranjevanje informacij o učenju v datoteko, katero po končanem procesu učenja naložimo v instanco strežnika za vizualizacijo podatkov ter tako pogledamo podrobnosti o učenju nevronske mreže za nazaj.

Programska koda 3.8: Primer shranjevanja in poznejšega prikaza informacij o učenju

```
// v aplikaciji za učenje shranimo podatke
SparkDl4jMultiLayer sparkDl4jMultiLayer = new
SparkDl4jMultiLayer(javaSparkContext, network, trainingMaster);

StatsStorage ss = new FileStatsStorage(new File("podatki_o_ucenju.dl4j"));
sparkNet.setListeners(ss, Collections.singletonList(new StatsListener(null)));

// po končanem učenju uporabimo podatke o učenju za vizualizacijo
StatsStorage statsStorage = new FileStatsStorage(new
File("podatki_o_ucenju.dl4j"));
UIServer uiServer = UIServer.getInstance();
uiServer.attach(statsStorage);
```

Spodnji primer (Programska koda 3.9) pa prikazuje uporabo funkcionalnosti oddaljenega uporabniškega vmesnika, kjer kreiramo ločeno aplikacijo, ki bo tekla na svojem strežniku, in v njej omogočimo instanci zalednega sistema uporabniškega vmesnika uporabo oddaljenega poslušalca. V aplikaciji za učenje nevronske mreže pa opredelimo usmerjevalnik oddaljene hrambe informacij o učenju ter opredeljeni mreži dodamo oddaljenega poslušalca.

Programska koda 3.9: Primer uporabe oddaljenega poslušalca za vizualizacijo informacij o učenju

```
// v ločeni aplikaciji omogočimo oddaljene poslušalce
UIServer uiServer = UIServer.getInstance();
uiServer.enableRemoteListener();

// v aplikaciji za učenje
SparkDl4jMultiLayer sparkDl4jMultiLayer = new
SparkDl4jMultiLayer(javaSparkContext, network, trainingMaster);
StatsStorageRouter remoteUIRouter = new
RemoteUIStatsStorageRouter("http://UI_SERVER_IP:9000");
sparkDl4jMultiLayer.setListeners(remoteUIRouter,
Collections.singletonList(new StatsListener(null)));
```

4 VZPOSTAVITEV RAZVOJNEGA OKOLJA

Za potrebe razvoja primera uporabe globokega učenja s knjižnico Deeplearning4j smo vzpostavili lokalno razvojno okolje in porazdeljeno okolje, ki smo ga uporabili za porazdeljeno učenje modelov. V nadaljevanju so najprej predstavljena izbrana orodja in ogrodja, čemur sledi v ločenih sklopih podrobnejši opis vzpostavitve tako lokalnega razvojnega okolja kot tudi porazdeljenega okolja.

4.1 Uporabljena orodja

Java

Java je visokonivojski programski jezik in programska platforma. Teče na več kot 50 milijonih osebnih računalnikov in milijardah mobilnih naprav po svetu. Platforma Java je sestavljena iz dveh glavnih komponent, in sicer Java API (angl. application programming interface, API), ki je knjižnica Java ukazov, ter Java navideznega stroja JVM (angl. Java Virtual Machine, JVM) ki interpretira Java kodo v strojni jezik. [21]

Apache Maven

Apache Maven[17] je orodje za upravljanje projektov programske kode. Temelji na podlagi koncepta POM (angl. project object model) in upravlja z gradnjo, poročanjem in dokumentiranjem projekta. [22]

Docker

Docker[18] je orodje, oblikovano z namenom olajšati kreiranje, nameščanje in poganjanje aplikacije z uporabo t. i. zabojnikov (angl. containers). Zabojniki razvijalcu omogočajo, da zapakira aplikacijo z vsemi potrebnimi knjižnicami, orodji in sistemskimi nastavitvami v en celovit paket – zabojnik [23]. Za razliko od navideznih strojev (angl. Virtual Machine – VM) zabojniki ne združujejo celotnega operacijskega sistema, ampak samo knjižnice, orodja in

[17] Uradna spletna stran orodja Apache Maven: https://maven.apache.org/
[18] Uradna spletna stran orodja Docker: https://www.docker.com/

nastavitve, ki so potrebne za pravilno delovanje aplikacije. To omogoča učinkovite, lahke, neodvisne sisteme in razvijalcu zagotavlja, da bo programska oprema vedno delovala enako, ne glede na to, na kakšnem sistemu bo nameščena [24]. Orodje Docker je na voljo za operacijske sisteme Windows, Mac OS ter linux distribucije CentOS, Debian, Fedora in Ubuntu, namestiti pa ga je mogoče tudi na Amazonov oblak AWS ter Microsoftov Azure.

Docker Compose

Docker Compose[19] je orodje za opredelitev in poganjanje Docker aplikacij z več zabojniki, pri čemer povezave med posameznimi zabojniki kot tudi konfiguracijo aplikacij opredelimo s kreiranjem konfiguracijske datoteke. Opredeljene aplikacije in storitve je nato moč ustvariti in zagnati z izvedbo enega terminalskega ukaza. Glavne lastnosti orodja so izolirana okolja na enem gostiteljskem sistemu, ohranitev podatkov ob kreiranju zabojnika ter ponovno kreiranje le tistih aplikacij oz. storitev, katerih konfiguracija je bila spremenjena. Orodje je najpogosteje uporabljano v razvojnih okoljih, avtomatiziranih testnih okoljih in pri namestitvah na en gostiteljski sistem. [25]

Apache Hadoop

Apache Hadoop je ogrodje, ki omogoča distribuirano procesiranje velikih podatkovnih zbirk prek gruč računalnikov z uporabo enostavnih programskih modelov. Zasnovan je na način, da omogoča skaliranje z enega strežnika na več tisoč strežnikov, od katerih vsak ponuja lokalno računanje in shrambo podatkov. Z mehanizmi zaznavanja in upravljanja napak na aplikacijskem nivoju zagotavlja visoko razpoložljivost storitev na vrhu gruče strežnikov, od katerih je vsak izmed njih nagnjen k okvaram. [26]

[19] Uradna spletna stran orodja Docker Compose: https://docs.docker.com/compose/

Projekt je zasnovan modularno iz naslednjih modulov: [26]

- Hadoop Common: splošne funkcionalnosti za podporo ostalim modulom;
- Hadoop Distributed File System – HDFS: porazdeljen datotečni sistem, ki zagotavlja visoko propustno dostopnost do aplikacijskih podatkov;
- Hadoop YARN: ogrodje za razporeditev nalog in upravljanje z viri gruče;
- Hadoop MapReduce: sistem, zasnovan na YARN ogrodju za potrebe paralelnega procesiranja velikih podatkovnih zbirk.

Apache Spark

Apache Spark je hiter, splošno namenski računski pogon, ki deluje nad Hadoop ogrodjem. Omogoča enostaven in izrazen programski model, ki podpira širok in bogat nabor aplikacij, vključno z ETL-procesom, SQL-poizvedbami in procesiranjem strukturiranih podatkov (Spark SQL), strojnim učenjem (MLlib), procesiranjem podatkovnih tokov (Spark Streaming) in izračunavanjem grafov (GraphX). Omogočen je visokonivojski aplikacijski programski vmesnik za programske jezike Java, Scala, Python in R. [27], [28]

Glavno vodilo pri izgradnji pogona je bila hitrost; Apache Spark je tako pri procesiranju velikih količin podatkov v primerjavi z Apache Hadoop do stokrat hitrejši predvsem na račun izkoriščanja pomnilniškega računanja in drugih optimizacij. [28]

Apache Spark lahko teče nad ogrodji Hadoop ali Mesos[20], samostojno ali pa v oblaku. Dostopa lahko do različnih virov podatkov, vključno z HDFS, Cassandra, HBase, S3, Hive, Tachyon in katerim koli podatkovnim virom Hadoop. [29]

[20] Uradna spletna stran ogrodja Mesos: http://mesos.apache.org/

4.2 Lokalno razvojno okolje

Za potrebe razvoja potrebujemo na lokalnem sistemu nameščen razvojni programski paket za Javo, orodje Apache Maven za pomoč z upravljanjem gradnje projektov ter upravljanje knjižnic, vključenih v projekt, orodje Docker, s pomočjo katerega bomo vzpostavili grafično nadzorno okolje Deeplearning4 UIj za spremljanje napredka učenja modelov, ter integrirano razvojno okolje za razvoj v programskem jeziku Java, v našem primeru je to okolje IntelliJ IDEA. V nadaljevanju je opisana vzpostavitev lokalnega razvojnega okolja na operacijskem sistemu Mac OS.

Za namestitev razvojnega paketa 8. verzija za Javo je treba najprej z uradne spletne strani (http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html) prenesti namestitveni program na lokalni računalnik. Po uspešnem prenosu je treba preneseni program namestiti, pri čemer nam je v pomoč namestitveni vodič, ki nas vodi skozi celoten potek namestitve.

V naslednjem koraku preverimo, ali je bil razvojni paket za Javo uspešno nameščen na računalnik. To storimo z izvedbo ukaza »java –version« znotraj terminala (Slika 4.1).

```
λ grega [~] → java -version
java version "1.8.0_112"
Java(TM) SE Runtime Environment (build 1.8.0_112-b16)
Java HotSpot(TM) 64-Bit Server VM (build 25.112-b16, mixed mode)
```

Slika 4.1: Izvedba ukaza za prikaz verzije nameščene Jave

Za namestitev Apache Maven je treba najprej s spleta (https://maven.apache.org/download.cgi) prenesti kompresirano datoteko, ki vsebuje potrebne Apache Maven programske ukaze. Pred samo namestitvijo je treba poskrbeti, da je na računalniku že nameščen in delujoč razvojni paket za Javo. Zatem preneseno kompresirano datoteko razširimo z izvedbo ukaza »tar xzfv apache-maven-3.5.0-bin.tar.gz« znotraj terminala ali za to uporabimo katero izmed nameščenih programskih orodij znotraj grafičnega vmesnika operacijskega sistema. Znotraj direktorija, v katerega

smo razširili kompresirano mapo, se nahaja direktorij »bin«, katerega je treba dodati v sistemsko spremenljivko PATH. Na operacijskih sistemih, temelječih na Unixu, to storimo z naslednjim terminalskim ukazom (Slika 4.2). [30]

```
λ grega [~] → export PATH=/opt/apache-maven-3.5.0/bin:$PATH
```

Slika 4.2: Ukaz za dodajanje Apache Maven izvedbenih datotek v sistemsko spremenljivko PATH

Da bi preverili, ali je Apache Maven orodje pravilno nameščeno v terminal, vpišemo sledeč ukaz, ki v primeru pravilne namestitve izpiše verzijo nameščenega orodja (Slika 4.3).

```
λ grega [~] → mvn -v
Apache Maven 3.5.0 (ff8f5e7444045639af65f6095c62210b5713f426; 2017-04-03T21:39:06+02:00)
Maven home: /usr/local/Cellar/maven/3.5.0/libexec
Java version: 1.8.0_112, vendor: Oracle Corporation
Java home: /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "mac os x", version: "10.12.5", arch: "x86_64", family: "mac"
```

Slika 4.3: Ukaz za izpis verzije nameščenega orodja Apache Maven

Za namestitev Docker orodja si najprej s spleta (https://www.docker.com/community-edition#/download) prenesemo namestitveni paket ter orodje namestimo s pomočjo čarovnika za namestitev.

Po končanem postopku namestitve orodja preverimo, ali je bilo to uspešno nameščeno ter ali pravilno deluje. Na operacijskih sistemih, ki temeljijo na Unixu, to storimo z naslednjim terminalskim ukazom (Slika 4.4). Če orodje deluje pravilno, se v terminalu izpiše nameščena verzija orodja.

```
λ grega [~] → docker -v
Docker version 17.06.0-ce, build 02c1d87
```

Slika 4.4: Ukaz za izpis verzije nameščenega orodja Docker

Sledi vzpostavitev oz. zagon strežnika grafičnega vmesnika modula Deeplearning4j UI. S tem namenom smo ustvarili zabojnik Docker (celotna opredelitev zabojnika je v prilogi A) z nameščenim razvojnim okoljem za Javo ter nameščenim orodjem Maven, ki po vzoru primera (Programska koda 3.9) vsebuje aplikacijo, kjer je kreirana nova instanca zalednega sistema modula Deeplearning4j UI. Sliko ustvarjenega zabojnika Docker smo naložili na javni register Docker slik zaradi enostavnejše ponovne uporabnosti – zabojnik bomo potrebovali tudi pri vzpostavitvi porazdeljenega okolja.

Ustvarjen zabojnik Docker nato zaženemo s terminalskim ukazom (Slika 4.5). Po končani izvedbi podanega ukaza je uporabniški vmesnik dostopen prek brskalnika na naslovu »http://localhost:9000«. Zatem moramo naložiti še samo izbrano integrirano razvojno okolje ali urejevalnik programske kode in s tem je vzpostavitev lokalnega okolja končana.

```
λ grega [~] → docker run --name deeplearning4j-ui -d -p 9000:9000 gregsi/docker-deeplearning4j-ui
```

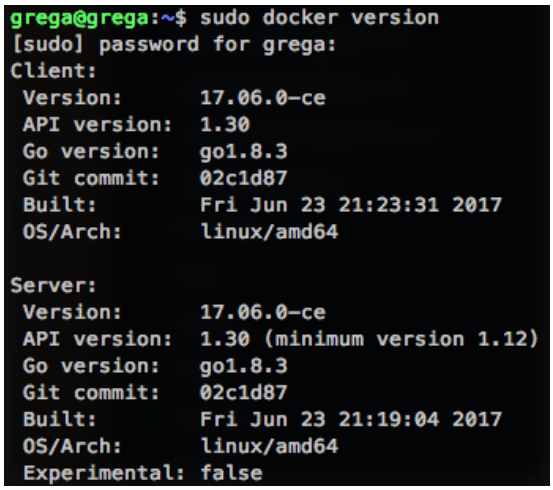
Slika 4.5: Ukaz za zagon Docker zabojnika za aplikacijo modula Deeplearning4j UI

4.3 Vzpostavitev okolja za porazdeljeno učenje

Za namene porazdeljenega učenja smo vzpostavili okolje na virtualnem zasebnem strežniku, na katerem teče operacijski sistem Ubuntu 16.04. Vzpostavitev okolja zajema namestitev orodij Docker in Docker Compose, ki služita kot osnova za vzpostavitev gruče Spark z enim vozliščem ter z enim Hadoop imenskim in podatkovnim vozliščem. Dodatno bo na strežniku prav tako v obliki zabojnika Docker tekla aplikacija za vizualizacijo stanja in napredka učenja Deeplearning4J UI.

Preden se lotimo vzpostavitve okolja za porazdeljeno učenje nevronskih mrež, moramo na sistem namestiti orodji Docker in Docker Compose, saj bomo z njuno pomočjo vzpostavili želeno okolje.

Za potrebe namestitve orodja Docker na operacijski sistem Ubuntu 16.04 smo spisali lupinsko skripto (Priloga B), ki avtomatizira postopek namestitve omenjenega orodja na sistem. Skripto izvedemo s terminalskim ukazom »sudo ./namestitev-docker.sh«, po končani izvedbi pa z ukazom »sudo docker version« (Slika 4.6) preverimo, ali je bila namestitev uspešna. V primeru pravilne namestitve orodja se nam izpišejo informacije o verzijah Docker odjemalca in strežnika.

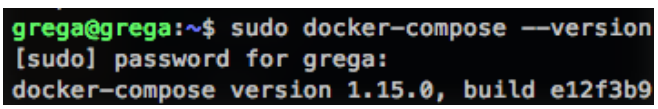


Slika 4.6: Ukaz za izpis verzije Docker odjemalca in strežnika

Sledi namestitev orodja Docker Compose, kar storimo s terminalskim ukazom:

```
curl -L https://github.com/docker/compose/releases/download/1.15.0/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose
```

Po končani izvedbi prejšnjega ukaza v terminal vpišemo ukaz »sudo docker-compose --version« (Slika 4.7) za izpis verzije nameščenega orodja Docker Compose in s tem preverimo, ali je orodje uspešno nameščeno na sistem.



Slika 4.7: Ukaz za izpis verzije Docker Compose orodja

Z namestitvijo tega orodja smo zaključili s samim nameščanjem orodij na strežnik. Sledita zagon zabojnika za aplikacijo modula Deeplearning4j UI za spremljanje stanja in napredka učenja nevronske mreže ter vzpostavitev Spark in Hadoop vozlišč s pomočjo orodja Docker Compose.

Po zgledu iz prejšnjega poglavja z ukazom (Slika 4.8) prenesemo in zaženemo zabojnik z nameščeno aplikacijo za spremljanje poteka učenja nevronskih mrež. Dodatno ukazu za zagon zabojnika podamo parameter »-v«, s katerim določimo, da se želena mapa znotraj zabojnika preslika v želeno mapo na gostiteljskem računalniku, kar nam omogoča olajšan dostop do shranjenih podatkov aplikacije. Po končani izvedbi ukaza lahko prek brskalnika preverimo, ali je aplikacija uspešno zagnana. Kot privzeta nastavitev aplikacija teče na vratih 9000.

```
grega@grega:~$ sudo docker run --name deeplearning4j-ui -d \
> -p 9000:9000 \
> -v ~/dl4j-ui-data:/root \
> gregsi/docker-deeplearning4j-ui
c303891c807fd6618f375fb085528f0d8a0f1dec57ff2b2667356253b0770954
```

Slika 4.8: Zagon zabojnika na strežniku za aplikacijo modula Deeplearning4j

Vzpostavitve gruče Spark z enim delavskim vozliščem in enim Hadoop imenskim in podatkovnim vozliščem smo se lotili z uporabo obstoječih zabojnikov Docker, katere smo s pomočjo orodja Docker Compose konfigurirali in povezali glede na naše potrebe oz. zahteve. S tem namenom smo ustvarili eno konfiguracijsko datoteko (Slika 4.9) z imenom hadoop.env, v kateri smo opredelili vrednosti spremenljivk za lokacijo privzetega datotečnega sistema, privzetega Hadoop uporabnika in omogočili spleten dostop do HDFS datotečnega sistema ter onemogočili dovoljenja za dostop do datotečnega sistema.

```
CORE_CONF_fs_defaultFS=hdfs://namenode:8020
CORE_CONF_hadoop_http_staticuser_user=root

HDFS_CONF_dfs_webhdfs_enabled=true
HDFS_CONF_dfs_permissions_enabled=false
```

Slika 4.9: Konfiguracijska datoteka hadoop.env

Zatem smo ustvarili datoteko »docker-compose.yml«, v kateri opredelimo aplikacije oz. storitve ter njihove povezave. V našem primeru smo tako opredelili Hadoop imenski in podatkovni strežnik ter glavno vozlišče Spark ter eno delavsko vozlišče. V spodnjem izseku (Programska koda 4.1) datoteke (celotna je v prilogi C) je del, kjer sta opredeljena Spark vozlišča.

Programska koda 4.1: Izsek docker-compose.yml datoteke

```
spark-master:
  image: bde2020/spark-master:2.1.0-hadoop2.8-hive-java8
  container_name: spark-master
  ports:
    - 8080:8080
    - 7077:7077
  env_file:
    - ./hadoop.env
spark-worker:
  image: bde2020/spark-worker:2.1.0-hadoop2.8-hive-java8
  depends_on:
    - spark-master
  environment:
    - SPARK_MASTER=spark://spark-master:7077
  ports:
    - 8081:8081
  env_file:
    - ./hadoop.env
```

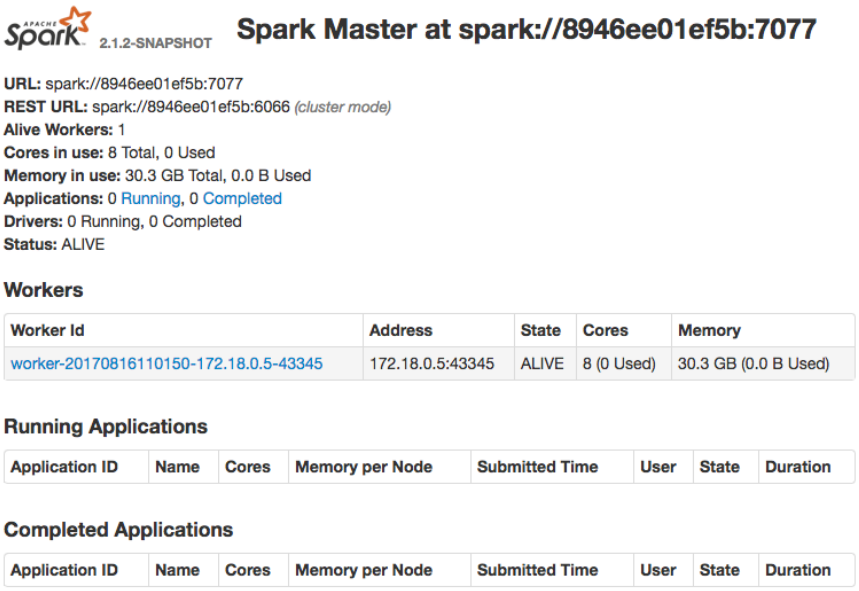
Glavno spark vozlišče, imenovano »spark-master«, ima opredeljene vhode in pa lokacijo prej ustvarjene konfiguracijske datoteke. Zatem pa je delavsko vozlišče Spark z imenom »spark-worker«, ki ima prav tako nastavljene vhode, odvisnost na glavno Spark vozlišče ter Spark pot do tega in prav tako lokacijo do konfiguracijske datoteke.

Tako opredeljeno konfiguracijo aplikacij oz. storitev z orodjem Docker Compose zaženemo z naslednjim ukazom »sudo docker-compose up –d«. S parametrom »-d« orodju povemo, da želimo, da se storitve zaženejo kot prikriti procesi (angl. daemon). V primeru uspešnega zagona vseh storitev se nam pojavi podoben izpis kot na sliki (Slika 4.10).

```
grega@grega:~/docker-spark-cluster$ sudo docker-compose up -d
Recreating spark-master ...
Recreating spark-master
Recreating namenode ...
Recreating namenode ... done
Recreating dockersparkcluster_datanode_1 ...
Recreating spark-master ... done
Recreating dockersparkcluster_spark-worker_1 ...
Recreating dockersparkcluster_spark-worker_1 ... done
```

Slika 4.10: Zagon storitev s pomočjo orodja Docker Compose

Ob pravilni konfiguraciji storitev se nam ob obisku spletnega naslova strežnika na vhodu 8080 prikaže spletna stran glavnega vozlišča Spark (Slika 4.11) z osnovnimi informacijami, seznamom registriranih delavskih vozlišč, aplikacij v izvajanju ter zaključenih aplikacij.

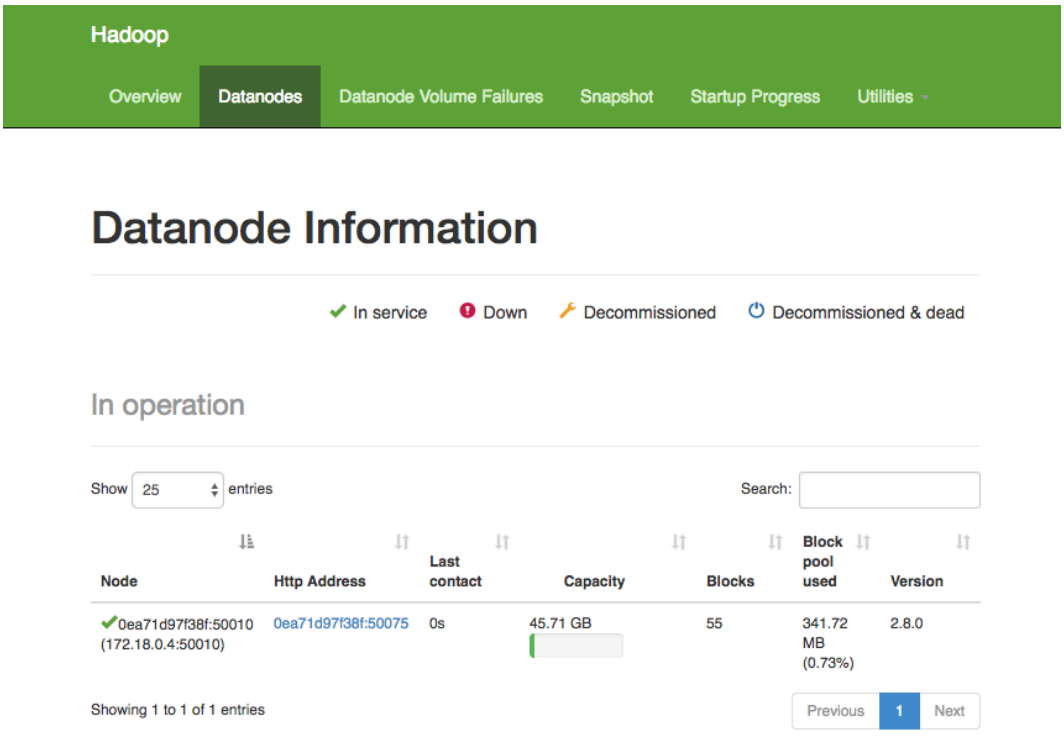


Slika 4.11: Stran glavnega vozlišča Spark

Ob obisku spletnega naslova strežnika se nam na vhodu 8081 prikaže spletna stran delavskega vozlišča (Slika 4.12) z osnovnimi informacijami ter tabelo izvrševalcev, ki so trenutno v teku.

Slika 4.12: Stran delavskega vozlišča Spark

Na vhodu 50070 spletnega naslova strežnika pa se nam prikaže Hadoop spletna stran (Slika 4.13), kjer se nahajajo vsi podatki o stanju imenskega in podatkovnega vozlišča, morebitne napake, spletni raziskovalec podatkovnega sistema itd.



Slika 4.13: Stran imenskega vozlišča Hadoop

5 IZDELAVA PRIMERA UPORABE

Kot primer uporabe globokega učenja s knjižnico DL4J smo implementirali rešitev za prepoznavo obrazov s slik. Z namenom prikaza čim več funkcionalnosti knjižnice smo si nalogo zastavili na način, da v prvem delu implementiramo aplikacijo, ki omogoča lokalno učenje na dveh različnih implementacijah konvolucijske nevronske mreže, in sicer »LeNet« ter »DeepFaceVariant«, v drugem delu pa obstoječ primer lokalnega učenja preoblikujemo za delovanje oz. porazdeljeno učenje na gruči Spark.

Implementacijo rešitve s konvolucijskimi nevronskimi mrežami smo izbrali zaradi tega, ker so v obstoječih raziskavah pogosto zastopane oz. predstavljene kot najučinkovitejše rešitve pri problemih, kot so prepoznava obrazov, zaznava obrazov na sliki in podobno. Kot smo spoznali v podpoglavju 2.5, spadajo konvolucijske mreže v skupino nevronskih mrež za nadzorovano učenje.

Primer smo implementirali v programskem jeziku Java v verziji 1.8 z uporabo knjižnice DL4J verzije 0.8.0 in podpornih knjižnic JCommander[21] verzije 1.27 ter Jackson[22] verzije 2.6.6. Za upravljanje odvisnostnih knjižnic, konfiguracije projekta ter grajenja projekta smo uporabili orodje Apache Maven verzije 3.5.

V nadaljevanju bomo podrobneje predstavili podatkovno množico, nad katero bomo izvajali učenje, princip delovanja konvolucijskih nevronskih mrež in tudi samo vzpostavitev projekta in implementacijo primera uporabe.

[21] Uradna spletna stran knjižnice Jcommander: http://jcommander.org/
[22] Uradna spletna stran knjižnice Jackson: https://github.com/FasterXML/jackson

5.1 Podatkovna množica

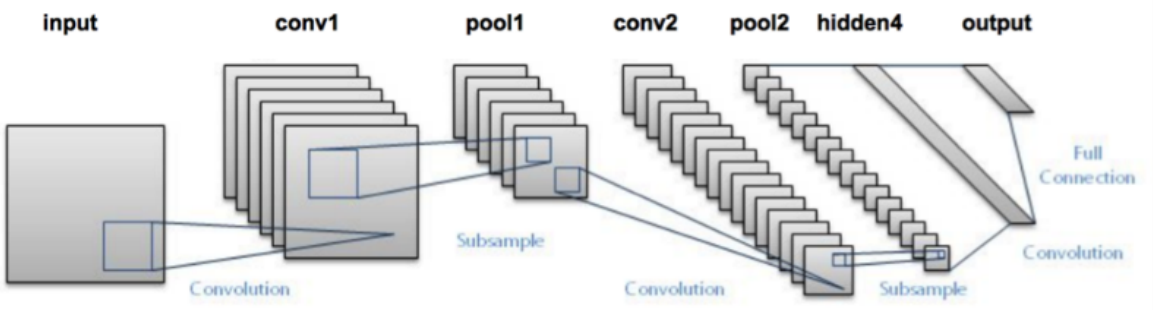
Za podatkovno množico, nad katero bomo izvajali učenje nevronske mreže s ciljem prepoznave obrazov, smo izbrali slikovno zbirko »Georgia Tech«, ki vsebuje fotografije petdesetih ljudi, fotografiranih na Georgia Institute of Technology. Vsaka oseba v zbirki je predstavljena s petnajstimi barvnimi slikami, shranjenimi v formatu JPEG ter posnetimi z ločljivostjo 640 x 480 slikovnih točk. Povprečna velikost obrazov na slikah je 150 x 150 slikovnih točk. Slike prikazujejo obraze, fotografirane s sprednjega profila in/ali obraze z različnimi nagibi ter različnimi obraznimi izrazi, zajetimi v različnih svetlobnih pogojih. [31]

Omenjeno zbirko smo izbrali zaradi velikosti zbirke kot take, saj s skupno 750 fotografijami omogoča enostavno in hitro obdelavo, po drugi strani pa vsebuje dovolj veliko število primerkov fotografij posamezne osebe, da je mogoče nad zbirko uspešno izvajati učenje nevronske mreže. Za namene hitrejšega učenja oz. zmanjšanja količine podatkov v procesu pridobivanja in obdelave podatkov pa sliko še dodatno obrežemo in pomanjšamo na velikost 120 x 140 slikovnih točk.

5.2 Konvolucijske nevronske mreže

Konvolucijske nevronske mreže so sestavljene iz enega ali več konvolucijskih nivojev (angl. Convolutional layer), mnogokrat tudi s korakom podvzorčenja (angl. Subsampling), nato pa sledi en ali več polno povezanih nivojev (angl. Fully connected layer) kot pri klasičnih večnivojskih nevronskih mrežah. Arhitektura CNN je sestavljena na način, da izkorišča dvodimenzionalno strukturo vhodnih podatkov (navadno slika ali avdio zapis). To doseže z lokalnimi povezavami in povezanimi utežmi, ki jim sledi neka oblika združevanja (angl. Pooling), kar ima za posledico invariantne parametre. Druga prednost CNN pa je lažja učljivost in manjše število parametrov kot polno povezane mreže pri istem številu skritih nivojev. [32]

Na sliki (Slika 5.1) je prikazan diagram primera konvolucijske nevronske mreže, sestavljene iz različnih konvolucijskih in združevalnih nivojev ter na koncu s polno povezanim nivojem. Kot vhod prvemu konvolucijskemu nivoju je podana slika. Filtri, ki so uporabljeni v konvolucijskem nivoju, izluščijo relevantne parametre s slike in jih podajo naprej. Vsak filter mora podati drugačen parameter za pravilno napovedovanje razreda. Če želimo ohraniti velikost slike, uporabimo enak (ničelni) odmik (angl. padding), v nasprotnem primeru uporabimo odmik, saj pripomore k zmanjšanju števila izluščenih parametrov. Zatem so dodani nivoji za združevanje, da dodatno zmanjšajo število parametrov. Pred samo predikcijo je dodanih več konvolucijskih in združevalnih nivojev. Globlje kot gremo v mrežo, več specifičnih parametrov je izluščenih v primerjavi s plitvimi mrežami, kjer so izluščeni parametri bolj splošni. Izhodni nivo je – kot smo že omenili –, polno povezan nivo, kjer so vhodi iz prejšnjih nivojev izravnani ter nato poslani izhodu. Na izhodnem nivoju se nato opravi transformacija vektorja realnih števil v vektor verjetnosti (nenegativna realna števila, katerih seštevek je 1). V tem kontekstu te verjetnosti ustrezajo možnosti, da je vhodna slika član določenega razreda. [33], [34]



Slika 5.1 Primer konvolucijske nevronske mreže [33]

5.3 Vzpostavitev projekta

Osnovni projekt smo vzpostavili s pomočjo integriranega razvojnega okolja IntelliJ IDEA in orodja Apache Maven. V ustvarjenem projektu smo v datoteki pom.xml opredelili konfiguracijo samega projekta, dodali odvisnosti ter opredelili postopek grajenja projekta v jar datoteko.

Spodnji izsek datoteke pom.xml (Programska koda 5.1) prikazuje opredelitev osnovnih podatkov o projektu, verzije knjižnic ter samo vključitev knjižnic v projekt.

Programska koda 5.1: Izsek datoteke pom.xml

```
<groupId>si.um.vrbancic</groupId>
<artifactId>dl4j</artifactId>
<version>1.0</version>
<packaging>jar</packaging>
<name>DeepLearning4J Example</name>
<description>Example of deep learning face recognition</description>
<!-- ostala konfiguracija -->
<dependencies>
    <dependency>
        <groupId>org.nd4j</groupId>
        <artifactId>${nd4j.backend}</artifactId>
        <version>${nd4j.version}</version>
    </dependency>
    <dependency>
        <groupId>org.deeplearning4j</groupId>
        <artifactId>dl4j-spark_${scala.binary.version}</artifactId>
        <version>${dl4j.spark.version}</version>
    </dependency>
    <dependency>
        <groupId>org.deeplearning4j</groupId>
        <artifactId>deeplearning4j-ui_${scala.binary.version}</artifactId>
        <version>${dl4j.version}</version>
    </dependency>
    <dependency>
        <groupId>com.beust</groupId>
        <artifactId>jcommander</artifactId>
        <version>${jcommander.version}</version>
    </dependency>
</dependencies>
```

5.4 Implementacija lokalnega učenja globoke mreže

Uporabljena zbirka slik je na spletu prosto dostopna v obliki kompresirane datoteke. Za hitrejši in enostavnejši nadaljnji razvoj smo najprej implementirali podporne metode za prenos zbirke na lokalni računalnik in njeno razširitev. Prav tako smo dodali metode, s katerimi preverjamo, ali je zbirka morda že prenesena in razširjena na lokalnem računalniku. S tem smo se izognili ponovnemu prenašanju in razširjanju podatkovne zbirke ob vsakem zagonu aplikacije.

Sledilo je inicializiranje potrebnih spremenljivk na privzete vrednosti. Z uporabo knjižnice JCommander smo implementirali podajanje in razčlenjevanje argumentov (Programska koda 5.2), ki so podani za vstopno točko aplikacije.

Programska koda 5.2: Implementacija razčlenjevanja argumentov s knjižnico JCommander

```
@Parameter(names = "-epochs", description = "epochs for the CNN")
private int epochs = 5;

@Parameter(names = "-saveNetwork", description = "save network to disk")
private boolean saveNetwork = true;

@Parameter(names = "-batchSize", description = "batch size")
private int batchSize = 25;

@Parameter(names = "-channels", description = "number of channels for images")
private int channels = 3;

@Parameter(names = "-totalImages", description = "number of images")
private int totalImages = 750;

@Parameter(names = "-type", description = "type of CNN")
private String type = "LeNet";

JCommander jCommander = new JCommander(this);
try {
    jCommander.parse(args);
} catch (ParameterException pe) {
    try {
        Thread.sleep(500);
    } catch (Exception e) {}
    throw pe;
}
```

S tem smo omogočili prilagajanje izvajanja aplikacije oz. spremembo učnih parametrov, ne da bi bilo treba spreminjati programsko kodo in ponovno zgraditi aplikacijo.

5.4.1 Branje in obdelava slik

Iz datotečnega sistema smo prebrali prej preneseno zbirko slik in množico naključno razdelili na dva dela, kjer je učenju nevronske mreže namenjenih 80 odstotkov slik, preostalih 20 odstotkov pa je namenjenih testiranju naučenega modela. Pri tem smo seme (angl. seed) za naključno razdelitev zbirke fiksno določili in s tem omogočili reprodukcijo rezultatov.

Sledil je proces branja slik iz učne množice in njihovo transformiranje. V procesu smo uporabili dve vgrajeni transformaciji, in sicer transformacijo za obrez fotografije, implementirane v razredu »CropImageTransform«, in transformacijo za spreminjanje velikosti slike, implementirane v razredu »ResizeImageTransform« (Programska koda 5.3). Z leve in desne strani smo slike obrezali za 200 slikovnih točk, zgoraj in spodaj pa za 100 slikovnih točk. Nato smo slike sorazmerno pomanjšali na velikost 120 x 140 slikovnih točk.

Programska koda 5.3: Branje in transformacija slik

```
ImageRecordReader imageRecordReader = new ImageRecordReader(height, width,
channels, new ParentPathLabelGenerator());
ImageTransform imageTransform = new MultiImageTransform(
    new CropImageTransform(100, 200, 100, 200),
    new ResizeImageTransform(120, 140));
imageRecordReader.initialize(trainData, imageTransform);
```

Rezultat transformacije slik je viden na sliki (Slika 5.2). Leva fotografija je sorazmerna izvorni velikosti, sredinska slika je prikaz rezultata obrezovanja slike, desna pa je rezultat pomanjšanja slike. S tem procesom smo zmanjšali porabo sistemskih virov, ki so potrebni za učenje nevronske mreže, ter pospešili proces učenja.

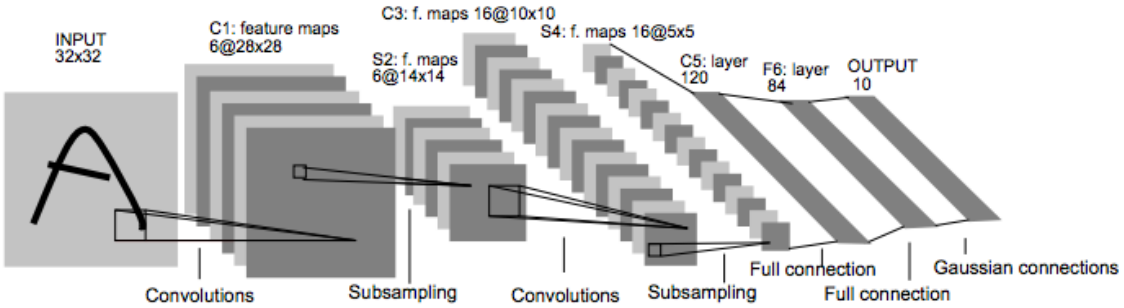
Slika 5.2: Rezultat transformacije slik

V nadaljevanju smo implementirali krmilno strukturo, s katero smo glede na vrednost spremenljivke »type« ustvarili ustrezen primerek razreda »MultiLayerNetwork«. Kot omenjeno smo opredelili dve različici konvolucijske nevronske mreže, in sicer »LeNet« ter »DeepFaceVariant«, katerih implementaciji sta podrobneje predstavljeni v naslednjem podpoglavju.

5.4.2 Implementacija nevronskih mrež

LeNet

Nevronska mreža LeNet izvira iz članka, objavljenega leta 1998, čigar avtorji so Yann LeCun, Leon Bottou, Yoshua Bengio in Patrick Haffner. Arhitektura mreže je sestavljena iz konvolucijskega nivoja z dvajsetimi filtri, kateremu sledi združevalni nivo, zatem je ponovno konvolucijski nivo, tokrat s petdesetimi filtri, ki je povezan s polno povezanim nivojem s petsto filtri. Slednjemu sledi še izhodni filter. Diagram (Slika 5.3) prikazuje izvorno arhitekturo mreže, ki je objavljena v članku. [16]



Slika 5.3: Arhitektura nevronske mreže LeNet [35]

To različico konvolucijske nevronske mreže smo implementirali po vzoru iz izvornega članka. Oba konvolucijska nivoja imata jedro (angl. kernel) v velikosti 5 x 5 točk, velikost koraka (angl. stride) 1 x 1 točk ter odmik (angl. padding), nastavljen na vrednost 0. V uporabi je aktivacijska funkcija usmerjene linearne enote (angl. Rectified Linear Unit – ReLU), za inicialicacijo uteži pa je uporabljena funkcija »Xavier«. Združevalna nivoja imata jedro v velikosti 2 x 2 točki in velikost koraka 1 x 1 točk. Uporabljena je združevalna funkcija maksimizacije. Polno povezan nivo za funkcijo izgube uporablja funkcijo negativne verjetnosti (angl. Negative likelihood), za normalizacijo podatkov pa funkcijo »softmax«. Za optimizacijski algoritem nevronske mreže je uporabljena funkcija stohastičnega gradientnega spusta (angl. Stochastic gradient descent). Spodnji izsek (Programska koda 5.4) prikazuje implementacijo prvega konvolucijskega in združevalnega nivoja mreže.

Programska koda 5.4: Izsek implementacije nevronske mreže LeNet

```
.layer(0, new ConvolutionLayer.Builder()
    .name("cnn1")
    .kernelSize(new int[] {5,5})
    .stride(new int[] {1,1})
    .padding(new int[] {0, 0})
    .nIn(channels)
    .biasInit(0)
    .nOut(20)
    .build())
.layer(1, new SubsamplingLayer.Builder(PoolingType.MAX)
    .name("maxpool1")
    .kernelSize(new int[] {2, 2})
    .stride(new int[] {2, 2})
    .build())
```

DeepFaceVariant

Različica nevronske mreže DeepFaceVariant temelji na arhitekturi nevronske mreže DeepFace, objavljene v članku, čigar avtorji so Yi Sun, Xiaogang Wang in Xiaoou Tang. Mreža je sestavljena iz štirih konvolucijskih nivojev z združevalnimi nivoji ter dvema polno povezanima nivojema. Na prvem konvolucijskem nivoju je dvajset filtrov, jedro je velikosti 4 x 4 točke z velikostjo koraka 1 x 1 točke. Na drugem nivoju je štirideset filtrov, jedro je velikosti 3 x 3 točke z velikostjo koraka 1 x 1 točke. Tretji korak ima šestdeset filtrov, velikost jedra in koraka pa je enaka kot na prejšnjem nivoju. Četrti konvolucijski nivo ima osemdeset filtrov z jedrom velikosti 2 x 2 točki in velikostjo koraka 1 x 1 točke. Trije združevalni nivoji imajo jedro velikosti 2 x 2 točki ter uporabljajo združevalno funkcijo maksimizacije. Polno povezan nivo ima sto šestdeset filtrov, kot funkcijo izgube uporablja funkcijo negativne, za normalizacijo podatkov pa funkcijo »softmax«. Za celotno nevronsko mrežo je v uporabi aktivacijska funkcija »ReLU«, za inicialicacijo uteži pa je uporabljena funkcija »Xavier«. Za optimizacijski algoritem nevronske mreže je uporabljena funkcija stohastičnega gradientnega spusta. Spodnji izsek programske kode (Programska koda 5.5) prikazuje implementacijo prvega konvolucijskega nivoja vključno z združevalnim nivojem.

Programska koda 5.5: Izsek implementacije nevronske mreže DeepFaceVariant

```
.layer(0, new ConvolutionLayer.Builder(4, 4)
    .name("cnn1")
    .nIn(channels)
    .stride(1, 1)
    .nOut(20)
    .build())
.layer(1, new SubsamplingLayer.Builder(SubsamplingLayer.PoolingType.MAX, new
int[]{2, 2})
    .name("pool1")
    .build())
```

5.4.3 Učenje mreže

Proces učenja nevronske mreže je zahteval uporabo razreda »DataSetIterator« za iteriranje po učni množici zbirk, katerega smo povezali s primerkom razreda »MultipleEpochsIterator«, s čimer smo omogočili večkratno ponavljanje učenja skozi celotno učno množico. V izseku programske kode (Programska koda 5.6) je prikazana implementacija učenja.

Programska koda 5.6: Izsek implementacije učenja nevronske mreže

```
DataSetIterator dataSetIterator = new
   RecordReaderDataSetIterator(imageRecordReader,batchSize,1, numSamples);
MultipleEpochsIterator multipleEpochsIterator = new
   MultipleEpochsIterator(epochs, dataSetIterator, numberOfCores);
startTime = System.currentTimeMillis();
multiLayerNetwork.fit(multipleEpochsIterator);
endTime = System.currentTimeMillis();
trainTime = endTime - startTime;
```

5.4.4 Evalvacija mreže

Za evalvacijo naučene nevronske mreže (Programska koda 5.7) smo uporabili že implementirano metodo »evaluate« razreda »MultiLayerNetwork«, ki ji kot vhodni parameter podamo testno množico podatkov. Metoda nam vrne objekt tipa »Evaluation«, ki med drugim vsebuje rezultate pravilnosti, natančnosti, občutljivosti in rezultat F1.

Programska koda 5.7: Izsek implementacije evalvacije nevronske mreže

```
imageRecordReader.initialize(testData);
dataSetIterator = new
   RecordReaderDataSetIterator(imageRecordReader, batchSize, 1, numSamples);
startTime = System.currentTimeMillis();
Evaluation evaluation = multiLayerNetwork.evaluate(dataSetIterator);
endTime = System.currentTimeMillis();
testTime = endTime - startTime;
totalTime = System.currentTimeMillis() - totalTime;
```

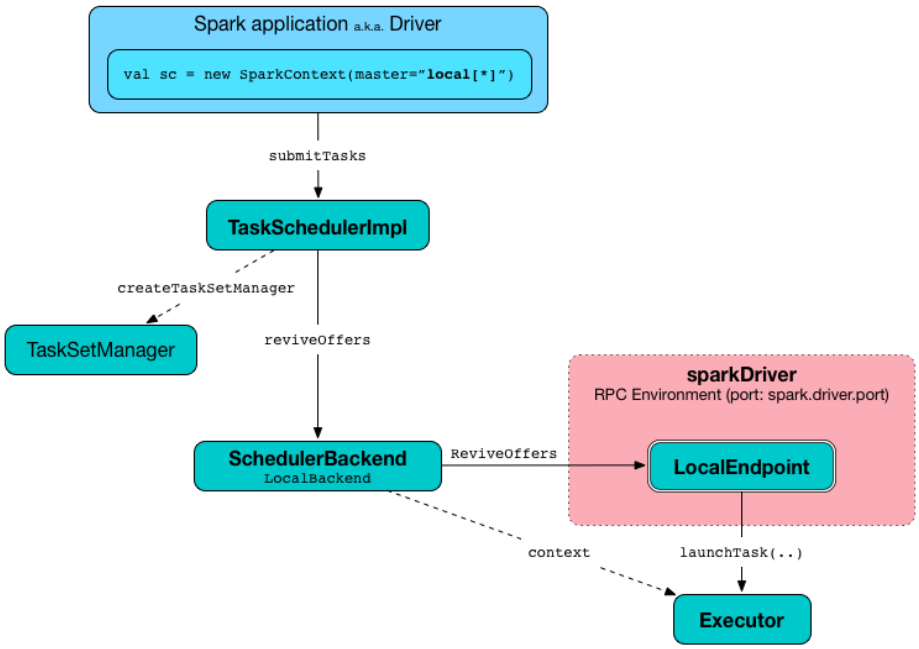
5.4.5 Serializacija modela

Po končani evalvaciji smo naučeni model serializirali na lokalni datotečni sistem, kar nam omogoča poznejšo ponovno uporabo modela bodisi za potrebe klasifikacije slik bodisi za nadaljnje učenje modela. Serializacijo smo izvedli z uporabo metode »writeModel« razreda »ModelSerializer«, ki smo ji kot parametre poslali celotno naučeno mrežo »multiLayerNetwork« in logično pot direktorija, kamor želimo serializirati model.

5.5 Primer porazdeljenega učenja globoke mreže

Knjižnica DL4J je bila, kot smo že omenili, razvita z mislijo na moderne porazdeljene sisteme in s tem tudi na porazdeljeno učenje še posebej z ogrodjema Hadoop in Spark. Knjižnica ima dobro zasnovan vmesnik za implementiranje porazdeljenega učenja na prej omenjenih ogrodjih. V nadaljevanju bomo predstavili postopek preoblikovanja ustvarjenega primera iz prejšnjega podpoglavja na način, da bomo omogočili delovanje oz. učenje na porazdeljenih sistemih, v našem primeru na gruči Spark.

Najprej je bilo treba v obstoječi primer vpeljati nov koncept konteksta Spark in konfiguracije Spark. S konfiguracijo Spark smo opredelili delovanje aplikacije v lokalnem in porazdeljenem načinu Spark. Namreč knjižnica omogoča delovanje oz. zagon aplikacije na lokalnem računalniku, na katerem ni nameščenih vozlišč Spark. Pri tem zažene znotraj enega samega neporazdeljenega gostiteljskega navideznega stroja Java vse izvedbene komponente ogrodja Spark, kot so gonilnik, izvršitelj itd. Paraleliziranje izvajanja pa poteka s pomočjo niti. Diagram (Slika 5.4) prikazuje potek lokalnega zagona oz. izvajanja aplikacije. Z nastavitvijo vrednosti atributa »master« konteksta Spark upravljamo s številom niti, ki bodo uporabljene za paralelizacijo izvajanja. [36]



Slika 5.4: Diagram lokalnega izvajanja aplikacije Spark [36]

Dodaten korak je potreben pri pripravi učne in testne množice podatkov, saj potrebujemo pri porazdeljenem učenju podatke v obliki JavaRDD<DataSet> podatkovne množice. Kot je razvidno iz izseka programske kode (Programska koda 5.8), smo z iteratorjem prešli skozi celotno množico učnih podatkov ter slike shranili v obliki seznama podatkovnih množic (List<DataSet>). V naslednjem koraku pa smo z uporabo metode »parallelize« razreda »JavaSparkContext« seznam spremenili v prej omenjeno strukturo, potrebno za porazdeljeno učenje. Enak postopek smo ponovili tudi za testno množico.

Programska koda 5.8: Preoblikovanje podatkovne strukture učne množice

```
imageRecordReader.initialize(trainData, imageTransform);
dataSetIterator = new RecordReaderDataSetIterator(
    imageRecordReader, batchSize, 1, numSamples);
List<DataSet> trainDataList = new ArrayList();
while(dataSetIterator.hasNext()) {
    trainDataList.add(dataSetIterator.next());
}

JavaRDD<DataSet> trainDataSetRDD = javaSparkContext.parallelize(trainDataList);
```

Ključna sprememba v programski kodi je vpeljava »TrainingMaster« vmesnika vključno z vgrajeno implementacijo učenja »ParameterAveragingTrainingMaster«. Ustvarimo primerek prej omenjenega razreda in mu nastavimo parametre učenja, kot je prikazano v poglavju 3.4.2 v primeru Programska koda 3.2.

Zadnja sprememba je pri hranjenju naučenega modela – ker poteka porazdeljeno učenje na gruči Spark, smo po serializaciji modela na lokalni datotečni sistem slednjega shranili tudi na datotečni sistem HDFS in s tem omogočili enostaven dostop do naučenega modela prek Hadoopovega spletnega vmesnika.

5.6 Izvedba učenja nevronskih mrež

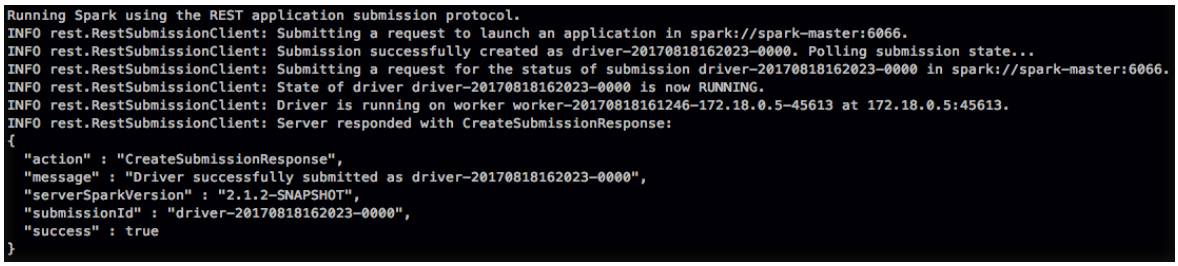
Kot smo že omenili v podpoglavju 3.4, je treba, če želimo izvajati porazdeljeno učenje na gruči Spark, najprej aplikacijo zapakirati v jar datoteko. Zatem moramo ustvarjeno jar datoteko narediti dostopno vsem vozliščem Spark ter na koncu zagnati proces učenja z ukazom »spark-submit«.

Mi smo si s tem namenom napisali enostavno lupinsko skripto (Programska koda 5.9), ki v prvem koraku z orodjem Maven zgradi jar datoteko, ki vključuje vse potrebe knjižnice za izvajanje aplikacije. V drugem koraku z ukazom »scp« prek povezave SSH na oddaljen virtualni zasebni strežnik pošlje ustvarjeno datoteko. Naslednji korak vsebuje ukaz za kopiranje prenesene datoteke v Docker zabojnika glavnega in delavskega vozlišča Spark, s čemer imata obe vozlišči dostop do jar datoteke. Zadnji korak pa je ukaz za zagon učenja nevronske mreže na gruči Spark.

Programska koda 5.9: Lupinska skripta za zagon porazdeljenega učenja

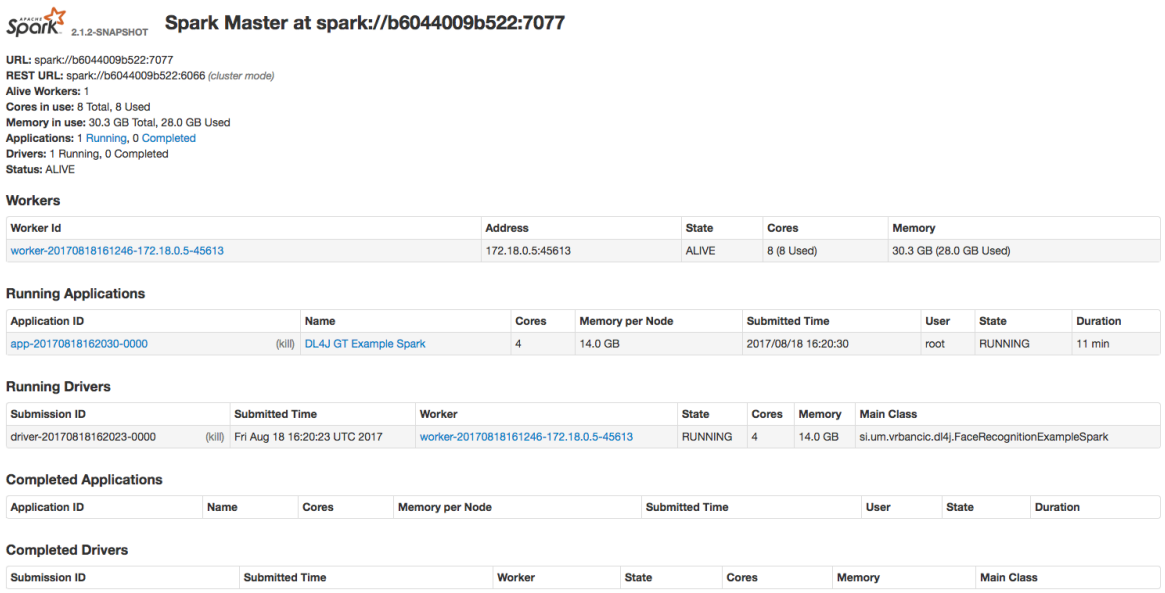
```
# build fat jar
mvn package -T 1C &&
# copy jar to remote VPS
scp target/dl4j-1.0-bin.jar root@grega.xyz:~/ &&
#copy jar from VPS to running docker containers
ssh root@grega.xyz 'docker cp dl4j-1.0-bin.jar spark-master:/dl4j-1.0-bin.jar
&& docker cp dl4j-1.0-bin.jar dockersparkcluster_spark-worker_1:/dl4j-1.0-
bin.jar';
# execute job on spark master container
ssh root@grega.xyz 'docker exec -i spark-master /spark/bin/spark-submit \
    --class si.um.vrbancic.dl4j.FaceRecognitionExampleSpark \
    --master spark://spark-master:6066 \
    --deploy-mode cluster \
    --executor-cores 4 \
    --driver-cores 4 \
    --executor-memory 14g \
    --driver-memory 14g \
    /dl4j-1.0-bin.jar -useSparkLocal false -epochs 10 -type LeNet'
```

Ukazu za zagon učenja »spark-submit« smo opredelili razred, ki predstavlja izvršljivo vstopno točko aplikacije, in podali naslov do glavnega vozlišča gruče Spark. Dodatno smo opredelili še način delovanja v gruči, nastavili število procesorskih jeder za gonilnik in izvršitelja kot tudi količino pomnilnika, ki ga lahko zavzameta. Sledijo nastavitev poti do jar datoteke ter parametri, podani naši aplikaciji. S parametrom »-useSparkLocal« smo opredelili, v kakšnem načinu želimo, da se aplikacija izvede. V konkretnem primeru se je aplikacija izvedla v porazdeljenem načinu. Parameter »-epochs« predstavlja število učnih ponovitev čez celotno učno množico, s parametrom »-type« pa določimo, katero nevronsko mrežo želimo učiti. Po izvršitvi lupinske skripte dobimo odgovor glavnega vozlišča gruče Spark o prejeti nalogi (Slika 5.5).



Slika 5.5: Odgovor glavnega vozlišča gruče Spark

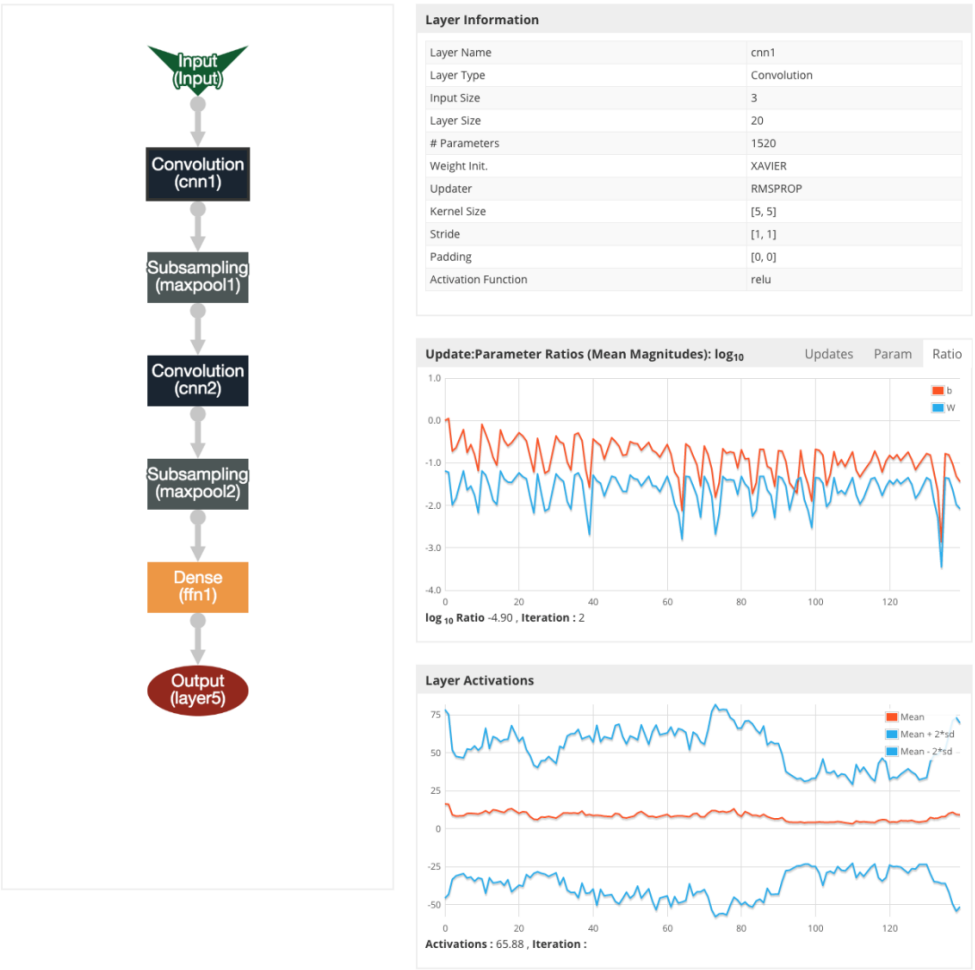
Na uporabniškem vmesniku glavnega vozlišča Spark (Slika 5.6) lahko vidimo stanje izvajanja aplikacije, napredek učenja pa lahko spremljamo prek uporabniškega vmesnika Deeplearning4j UI (Slika 5.7).



Slika 5.6: Pregled stanja izvajanja aplikacije

Slika 5.7: Uporabniški vmesnik za spremljanje napredka učenja

Kot smo že predstavili v podpoglavju 3.5, nam uporabniški vmesnik Deeplearnign4j UI omogoča spremljanje stanja in poteka učenja tudi po posameznih nivojih nevronske mreže. Na levi strani slike (Slika 5.8) je prikazan diagram nivojev nevronske mreže LeNet. Puščica na vrhu predstavlja vhod podatkov, elipsa na dnu pa predstavlja zadnji polno povezan nivo oz. izhod. Pravokotniki predstavljajo nivoje nevronske mreže. Ob kliku na posamezen nivo se v okvirju zraven diagrama prikažejo podrobnosti o izbranem nivoju, ki med drugim vključujejo tip nivoja, vhodno velikost, število filtrov, število parametrov, velikost jedra, velikost odmika itd. Pod okvirjem z osnovnimi informacijami imamo v obliki grafa predstavljeno razmerje posodobitev parametrov, aktivacij na nivoju, histogram parametrov nivoja, histogram posodobitev nivoja ipd. Orodje je predvsem uporabno za pomoč pri oblikovanju nevronske mreže, razhroščevanju učenja, in pri finih nastavitvah ter optimizacijah mrež.



Slika 5.8: Prikaz modela nevronske mreže LeNet

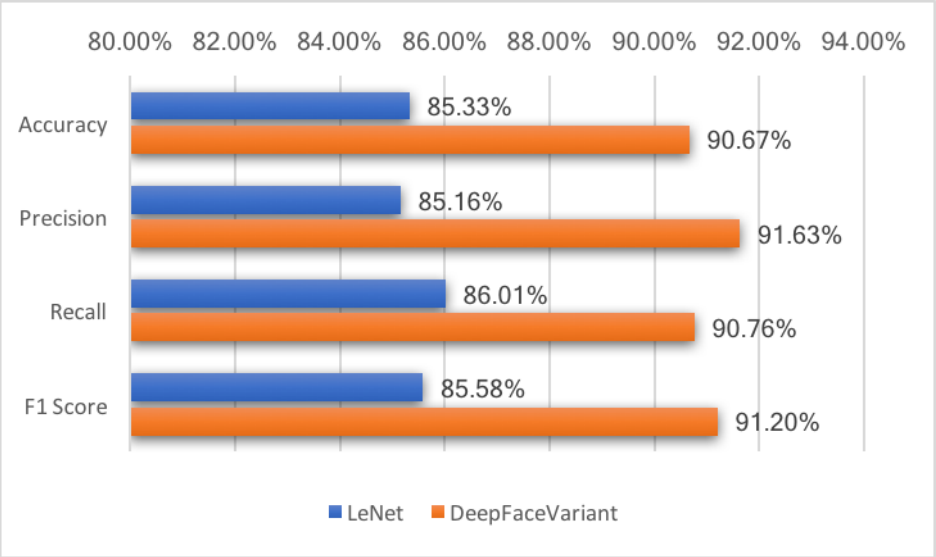
5.7 Analiza rezultatov učenja

Vsako različico konvolucijske nevronske mreže smo učili na isti strojni opremi, tj. v našem primeru na virtualnem zasebnem strežniku z osemjedrnim procesorjem, ki ima 32 GB pomnilnika in nameščen operacijski sistem Ubuntu 16.04. Uporabljeni zagonski parametri ukaza »spark-submit« in tudi parametri podani aplikaciji so predstavljeni v spodnji tabeli (Tabela 5.1). Učenje je potekalo na gruči Spark z enim glavnim vozliščem, enim delavskim vozliščem ter Hadoop imenskim in podatkovnim strežnikom.

Tabela 5.1: Zagonski parametri učenja nevronskih mrež

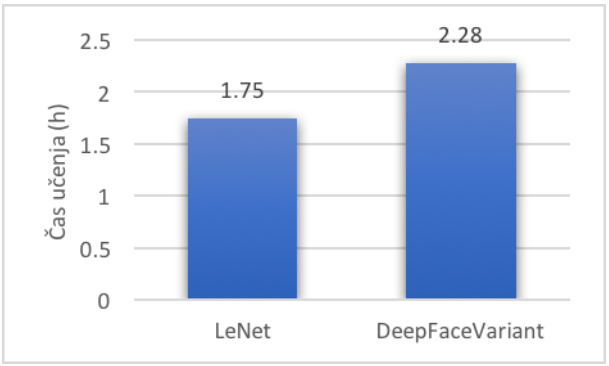
Parameter	Vrednost
Način izvajanja	Gruča
Število jeder izvršitelja	4
Število jeder gonilnika	4
Velikost pomnilnika izvršitelja	14 GB
Velikost pomnilnika gonilnika	14 GB
Način izvajanja aplikacije	Porazdeljeno
Število epoh	10

Na sliki (Slika 5.9) so grafično predstavljeni končni rezultati uspešnosti posamezne nevronske mreže. V splošnem je uspešnost obeh modelov visoka glede na to, da mrež nismo posebej prilagajali oz. optimizirali. Na tako pozitiven rezultat učenja mrež sta vsekakor vplivali zbirka slik z zadostnim številom fotografij posameznih oseb, kar je mrežam omogočilo uspešnejše učenje, kot tudi izbira konvolucijskih tipov nevronskih mrež, ki so se v mnogih raziskavah na področju prepoznave obrazov že izkazale kot uspešne.



Slika 5.9: Rezultati modelov nevronskih mrež

Razliko med rezultati posameznih modelov nevronskih mrež kot tudi času potrebnem za učenje (Slika 5.10) enega in drugega modela lahko pripišemo večji globini nevronske mreže DeepFaceVariant v primerjavi z LeNet.



Slika 5.10: Čas, ki je potreben za učenje nevronskih mrež

Za dodatno izboljšanje rezultatov učenja bi veljalo poizkusiti različne parametre učenja, kot so hitrost učenja (angl. learning rate), druga aktivacijska funkcija, druga funkcija za inicializacijo uteži ipd. Dodatno bi lahko v procesu priprave in preoblikovanja podatkov uporabili transformacijo, s katero bi naključno za nekaj stopinj rotirali sliko levo ali desno ter s tem pridobili bolj raznolike nagibe obrazov, kar bi pripomoglo pri boljši generalizaciji modela.

6 SKLEP

V magistrskem delu smo spoznali globoko učenje, ki predstavlja nov mejnik na področju strojnega učenja in umetne inteligence. Vedno večje količine nakopičenih podatkov, zmogljiva strojna oprema ter arhitekture in pristopi globokega učenja tvorijo kombinacije, ki iz dneva v dan prinašajo nove izzive, kar je razvidno tudi iz aktivnosti raziskovalcev na tem področju.

Knjižnica DL4J se je izkazala kot zmogljiva in učinkovita izbira pri reševanju problemov z uporabo globokega učenja. Kot eno izmed največjih prednosti pred konkurenčnimi knjižnicami bi izpostavili njeno usmerjenost oz. podporo za porazdeljeno učenje modelov, ki je ob vedno večjih količinah podatkov pravzaprav že nujna za vsako resnejšo naslavljanje problemov z uporabo globokega učenja. Kot slabost bi izpostavili pomanjkljivo dokumentacijo, zaradi katere je uporabnik mnogokrat primoran k raziskovanju izvorne kode knjižnice.

Za namene razvoja primera uporabe globokega učenja s knjižnico DL4J smo vzpostavili tako lokalno kot tudi porazdeljeno okolje. Kot primer uporabe smo implementirali aplikacijo za prepoznavo obrazov z uporabo lokalnega učenja, katerega smo v nadaljevanju preoblikovali s ciljem porazdeljenega učenja na gruči Spark. V sklopu aplikacije smo opredelili dve različici konvolucijskih nevronskih mrež »LeNet« in »DeepFaceVariant« ter z vsako posebej izvedli porazdeljeno učenje nad podatkovno zbirko »Georgia Tech«.

Menimo, da se je izbrana kombinacija ogrodja Spark in knjižnice DL4J izkazala za uspešno ter da ima uporabljen pristop k uporabi globokega učenja visok potencial še posebej pri reševanju realnih problemov v gospodarskem okolju. V nadaljnjih raziskavah bi bilo smiselno izvesti eksperiment z uporabo grafičnih procesnih enot ter preizkusiti izvajanje porazdeljenega učenja na ostalih podprtih porazdeljenih okoljih (YARN, Mesos).

LITERATURA

[1] A. Smola in S. V. N. Vishwanathan, *Introduction to Machine Learning*. Cambridge: Cambridge University Press, 2010.

[2] N. Buduma in N. Locascio, *Fundamentals of Deep Learning*, First Edit. O'Reilly Media, Inc., 2017.

[3] L. Deng in D. Yu, *Deep Learning: Methods and Applications*. 2014.

[4] D. Dev, *Deep Learning with Hadoop*. Packt Publishing, 2017.

[5] Y. Sugomori, B. Kaluža, F. M. Soares, in A. M. F. Souza, *Deep Learning - Practical Neural Networks with Java*. Packt Publishing, 2017.

[6] Stack Overflow, „Stack Overflow Developer Survey 2017", 20017. [Na spletu]. Dostopno: https://insights.stackoverflow.com/survey/2017#most-popular-technologies. [Dostopano: 26. julija 2017].

[7] Skymind, „Deeplearning4j: Open-source, Distributed Deep Learning for the JVM". [Na spletu]. Dostopno: https://deeplearning4j.org/index.html. [Dostopano: 08. avgusta 2017].

[8] Skymind, „Deeplearning4j With GPUs - Deeplearning4j: Open-source, Distributed Deep Learning for the JVM". [Na spletu]. Dostopno: https://deeplearning4j.org/gpu. [Dostopano: 10. avgusta 2017].

[9] A. Gibson in J. Patterson, *Deep learning: A practitioner's approach*. 2017.

[10] Skymind, „Deeplearning4j on Spark - Deeplearning4j: Open-source, Distributed Deep Learning for the JVM". [Na spletu]. Dostopno: https://deeplearning4j.org/spark. [Dostopano: 10. avgusta 2017].

[11] O. Mendelevitch, C. Stella, in D. Eadline, *Practical Data Science with Hadoop and Spark*. Pearson Education, Inc., 2017.

[12] Skymind, „Data sets and machine learning - Deeplearning4j: Open-source, Distributed Deep Learning for the JVM". [Na spletu]. Dostopno: https://deeplearning4j.org/data-sets-ml. [Dostopano: 11. avgusta 2017].

[13] Skymind, „Custom Datasets - Deeplearning4j: Open-source, Distributed Deep

Learning for the JVM“. [Na spletu]. Dostopno: https://deeplearning4j.org/customdatasets. [Dostopano: 11. avgusta 2017].

[14] Skymind, „DeepLearning4J - ETL User Guide - Deeplearning4j: Open-source, Distributed Deep Learning for the JVM“. [Na spletu]. Dostopno: https://deeplearning4j.org/etl-userguide. [Dostopano: 12. avgusta 2017].

[15] Skymind, „Building Neural Networks with DeepLearning4J - Deeplearning4j: Open-source, Distributed Deep Learning for the JVM“. [Na spletu]. Dostopno: https://deeplearning4j.org/building-neural-net-with-dl4j. [Dostopano: 13. avgusta 2017].

[16] A. Grigorev, *Mastering Java for Data Science*. Birmingham - Mumbai: Packt Publishing, 2017.

[17] I. Ganelin, E. Orhian, K. Sasaki, in B. York, *Spark: Big Data Cluster Computing in Production*. John Wiley & Sons, Inc., 2016.

[18] Apache Software Foundation, „Submitting Applications - Spark 2.2.0 Documentation“. [Na spletu]. Dostopno: http://spark.apache.org/docs/latest/submitting-applications.html. [Dostopano: 15. avgusta 2017].

[19] M. Asif Abbasi, *Learning Apache Spark 2*. Packt Publishing, 2017.

[20] Skymind, „How to Visualize, Monitor and Debug Neural Network Learning - Deeplearning4j: Open-source, Distributed Deep Learning for the JVM“. [Na spletu]. Dostopno: https://deeplearning4j.org/visualization. [Dostopano: 14. avgusta 2017].

[21] Oracle, „Java basics“. [Na spletu]. Dostopno: http://www.oracle.com/technetwork/topics/newtojava/downloads/index.html. [Dostopano: 26. julija 2017].

[22] Apache, „Maven – Welcome to Apache Maven“. [Na spletu]. Dostopno: https://maven.apache.org/. [Dostopano: 26. julija 2017].

[23] opensource.com, „What is Docker? | Opensource.com“. [Na spletu]. Dostopno: https://opensource.com/resources/what-docker. [Dostopano: 08. avgusta 2017].

[24] Docker Inc., „What is a Container | Docker“. [Na spletu]. Dostopno: https://www.docker.com/what-container. [Dostopano: 08. avgusta 2017].

[25] Docker Inc., „Overview of Docker Compose | Docker Documentation“. [Na spletu]. Dostopno: https://docs.docker.com/compose/overview/#only-recreate-containers-that-have-changed. [Dostopano: 12. avgusta 2017].

[26] The Apache Software Foundation, „Welcome to Apache™ Hadoop®!“ [Na spletu]. Dostopno: http://hadoop.apache.org/#What+Is+Apache+Hadoop%3F. [Dostopano: 12. avgusta 2017].

[27] Apache Software Foundation, „Overview - Spark 2.2.0 Documentation“. [Na spletu]. Dostopno: https://spark.apache.org/docs/latest/. [Dostopano: 13. avgusta 2017].

[28] Databricks, „What is Apache Spark?“ [Na spletu]. Dostopno: https://databricks.com/spark/about. [Dostopano: 13. avgusta 2017].

[29] Apache Software Foundation, „Apache Spark™ - Lightning-Fast Cluster Computing“. [Na spletu]. Dostopno: http://spark.apache.org/. [Dostopano: 13. avgusta 2017].

[30] Apache, „Maven – Installing Apache Maven“. [Na spletu]. Dostopno: https://maven.apache.org/install.html. [Dostopano: 26. julija 2017].

[31] A. V. Nefian, „Ara Nefian Face Recognition Page“. [Na spletu]. Dostopno: http://www.anefian.com/research/face_reco.htm. [Dostopano: 13. avgusta 2017].

[32] Stanford University, „Unsupervised Feature Learning and Deep Learning Tutorial“. [Na spletu]. Dostopno: http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/. [Dostopano: 15. avgusta 2017].

[33] J. Kunal, „Architecture of Convolutional Neural Networks (CNNs) demystified“. [Na spletu]. Dostopno: https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/. [Dostopano: 16. avgusta 2017].

[34] Cambridge Spark, „Deep learning for complete beginners: convolutional neural networks with keras“. [Na spletu]. Dostopno: https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html. [Dostopano: 15. avgusta 2017].

[35] Y. LeCun, L. Bottou, Y. Bengio, in P. Haffner, „Gradient-Based Learning Applied to Document Recognition“, 1998.

[36] J. Laskowski, „Spark local (pseudo-cluster) · Mastering Apache Spark 2 (Spark 2.2+)“. [Na spletu]. Dostopno: https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-local.html. [Dostopano: 15. avgusta 2017].

[37] Apache Software Foundation, „Spark Programming Guide - Spark 2.2.0 Documentation“. [Na spletu]. Dostopno: https://spark.apache.org/docs/latest/rdd-programming-guide.html. [Dostopano: 15. avgusta 2017].

Priloga A

Definicija Docker zabojnika za aplikacijo Deeplearning4j UI modula

```
FROM gregsi/docker-oracle-java8
MAINTAINER Grega Vrbančič <grega.vrbancic@gmail.com>

# get mvn
ENV MVN_VERSION 3.5.0
RUN wget -O /tmp/apache-maven-$MVN_VERSION.tar.gz
http://archive.apache.org/dist/maven/maven-3/$MVN_VERSION/binaries/apache-
maven-$MVN_VERSION-bin.tar.gz

# install mvn
RUN tar xzf /tmp/apache-maven-$MVN_VERSION.tar.gz -C /opt/
RUN ln -s /opt/apache-maven-$MVN_VERSION /opt/maven
RUN ln -s /opt/maven/bin/mvn /usr/local/bin
RUN rm -f /tmp/apache-maven-$MVN_VERSION.tar.gz
ENV MAVEN_HOME /opt/maven

WORKDIR /deeplearning4j-ui

# resolve mvn dependencies
ADD deeplearning4j-ui/pom.xml /deeplearning4j-ui/pom.xml
RUN ["mvn", "dependency:resolve"]
RUN ["mvn", "verify", "-Dmaven.test.skip=true"]

# package deeplearning4j-ui to jar
ADD deeplearning4j-ui /deeplearning4j-ui
RUN ["mvn", "package", "-Dmaven.test.skip=true"]

# clean
RUN apt-get clean
RUN rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*

# run
EXPOSE 9000
CMD ["java", "-cp", "target/deeplearning4j-ui-1.0-bin.jar",
"xyz.grega.deeplearning4jui.Server"]
```

Lupinska skripta za namestitev orodja Docker

```
#!/bin/bash
sudo true
sudo apt-get update
sudo apt-get install apt-transport-https ca-certificates curl software-
properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

sudo apt-get update
sudo apt-get install docker-ce
```

docker-compose.yml datoteka

```yaml
version: '2'
services:
  namenode:
    image: bde2020/hadoop-namenode:1.1.0-hadoop2.8-java8
    container_name: namenode
    volumes:
      - ./data/namenode:/hadoop/dfs/name
    environment:
      - CLUSTER_NAME=test
    env_file:
      - ./hadoop.env
    ports:
      - 50070:50070
  datanode:
    image: bde2020/hadoop-datanode:1.1.0-hadoop2.8-java8
    depends_on:
      - namenode
    volumes:
      - ./data/datanode:/hadoop/dfs/data
    env_file:
      - ./hadoop.env
    ports:
      - 50075:50075
  spark-master:
    image: bde2020/spark-master:2.1.0-hadoop2.8-hive-java8
    container_name: spark-master
    ports:
      - 8080:8080
      - 7077:7077
    env_file:
      - ./hadoop.env
  spark-worker:
    image: bde2020/spark-worker:2.1.0-hadoop2.8-hive-java8
    depends_on:
      - spark-master
    environment:
      - SPARK_MASTER=spark://spark-master:7077
    ports:
      - 8081:8081
    env_file:
      - ./hadoop.env
```

ZA ŠTUDENTE E, ITK, MK, RI-T in TK

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Koroška cesta 46
2000 Maribor, Slovenija

FERI

IZJAVA O USTREZNOSTI ZAKLJUČNEGA DELA

Podpisani mentor/-ica : red. prof. dr. VILI PODGORELEC
(ime in priimek mentor-ja/-ice)

in somentor/-ica (eden ali več, če obstajajo): _____
(ime in priimek somentor-ja/-ice)

Izjavlja-m/-va/-mo, da je študent/-ka

Ime in priimek: GREGA VRBANČIČ, *ID številka:* 10019945 23,

vpisna številka: E5027017, *na študijskem programu:*
INFORMATIKA IN TEHNOLOGIJE KOMUNICIRANJA MAG

izdelal/-a zaključno delo z naslovom:
UPORABA GLOBOKEGA UČENJA S KNJIŽNICO DEEPLEARNING4J
NA PRIMERU PREPOZNAVE OBRAZOV
(naslov zaključnega dela v slovenskem jeziku)

v skladu z odobreno temo zaključnega dela, navodili o pripravi zaključnih del in mojimi (najinimi/našimi) navodili.

Preveril/-a/-i in pregledal/-a/-i sem/sva/smo poročilo o preverjanju podobnosti vsebin z drugimi deli (priloga) in potrjujem/potrjujeva/potrjujemo, da je zaključno delo ustrezno.

Datum in kraj: MARIBOR, 24.08.2017

Podpis mentor-ja/-ice:

Datum in kraj:

Podpis somentor-ja/-ice (če obstaja):

Priloga:
- Poročilo o preverjanju podobnosti vsebin z drugimi deli.

ZA ŠTUDENTE E, ITK, MK, RI-T in TK

Univerza v Mariboru

FERI

Fakulteta za elektrotehniko, računalništvo in informatiko

IZJAVA O AVTORSTVU IN ISTOVETNOSTI TISKANE IN ELEKTRONSKE OBLIKE ZAKLJUČNEGA DELA

Ime in priimek študent-a/-ke: GREGA VRBANČIČ
Študijski program: INFORMATIKA IN TEHNOLOGIJE KOMUNICIRANJA MAG
Naslov zaključnega dela: UPORABA GLOBOKEGA UČENJA S KNJIŽNICO DEEPLEARNING4J NA PRIMERU PREPOZNAVE OBRAZOV

Mentor: red. prof. dr. VILI PODGORELEC
Somentor: _____

Podpisan-i/-a študent/-ka GREGA VRBANČIČ

- izjavljam, da je zaključno delo rezultat mojega samostojnega dela, ki sem ga izdelal/-a ob pomoči mentor-ja/-ice oz. somentor-ja/-ice;
- izjavljam, da sem pridobil/-a vsa potrebna soglasja za uporabo podatkov in avtorskih del v zaključnem delu in jih v zaključnem delu jasno in ustrezno označil/-a;
- na Univerzo v Mariboru neodplačno, neizključno, prostorsko in časovno neomejeno prenašam pravico shranitve avtorskega dela v elektronski obliki, pravico reproduciranja ter pravico ponuditi zaključno delo javnosti na svetovnem spletu preko DKUM; sem seznanjen/-a, da bodo dela deponirana/objavljena v DKUM dostopna široki javnosti pod pogoji licence Creative Commons BY-NC-ND, kar vključuje tudi avtomatizirano indeksiranje preko spleta in obdelavo besedil za potrebe tekstovnega in podatkovnega rudarjenja in ekstrakcije znanja iz vsebin; uporabnikom se dovoli reproduciranje brez predelave avtorskega dela, distribuiranje, dajanje v najem in priobčitev javnosti samega izvirnega avtorskega dela, in sicer pod pogojem, da navedejo avtorja in da ne gre za komercialno uporabo;
- dovoljujem objavo svojih osebnih podatkov, ki so navedeni v zaključnem delu in tej izjavi, skupaj z objavo zaključnega dela;
- izjavljam, da je tiskana oblika zaključnega dela istovetna elektronski obliki zaključnega dela, ki sem jo oddal/-a za objavo v DKUM.

Uveljavljam permisivnejšo obliko licence Creative Commons:_____(navedite obliko)

Začasna nedostopnost:

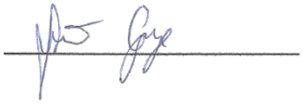
Zaključno delo zaradi zagotavljanja konkurenčne prednosti, zaščite poslovnih skrivnosti, varnosti ljudi in narave, varstva industrijske lastnine ali tajnosti podatkov naročnika:
_____(naziv in naslov naročnika/institucije) ne sme biti javno dostopno do _____(datum odloga javne objave ne sme biti daljši kot 3 leta od zagovora dela). To se nanaša na tiskano in elektronsko obliko zaključnega dela.

Temporary unavailability:

To ensure competition priority, protection of trade secrets, safety of people and nature, protection of industrial property or secrecy of customer's information, the thesis _____(institution/company name and address) must not be accessible to the public till_____(delay date of thesis availability to the public must not exceed the period of 3 years after thesis defense). This applies to printed and electronic thesis forms.

Datum in kraj: MARIBOR, 24.08.2017

Podpis študent-a/-ke:



Podpis mentor-ja/-ice: _____
(samo v primeru, če delo ne me biti javno dostopno)

Ime in priimek ter podpis odgovorne osebe naročnika in žig:

_____
(samo v primeru, če delo ne sme biti javno dostopno)

ZA ŠTUDENTE E, ITK, MK, RI-T in TK

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Koroška cesta 46
2000 Maribor, Slovenija

IZJAVA O OBJAVI OSEBNIH PODATKOV

Ime in priimek diplomant-a/ magistrant-/-ke: GREGA VRBANČIČ

ID številka: 1001994523

Študijski program: INFORMATIKA IN TEHNOLOGIJE KOMUNICIRANJA MAG

Naslov zaključnega dela: UPORABA GLOBOKEGA UČENJA S KNJIŽNICO DEEPLEARNING4J NA PRIMERU PREPOZNAVE OBRAZOV

Mentor/-ica: red. prof. dr. VILI PODGORELEC

Somentor/-ica: _____

Podpisan-i/-a izjavljam, da dovoljujem objavo osebnih podatkov, vezanih na zaključek študija (ime, priimek, leto zaključka študija, naslov zaključnega dela) na spletnih straneh Univerze v Mariboru in v publikacijah Univerze v Mariboru.

Datum in kraj: MARIBOR, 24.08.2017

Podpis diploman-ta/magistran-ta/-ke: