UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Andrej Damjan Mlinar Groznik

Izvajanje naučenih globokih nevronskih mrež v vgrajenih sistemih

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM
PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: akad. prof. dr. Ivan Bratko

Ljubljana, 2018

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Naloga je razviti metodo in postopke za implementacijo izvajanja nevronskih mrež v vgrajenih sistemih. Metoda naj temelji na uporabi že naučenih nevronskih mrež, dobljenih z uporabo obstoječih odprto kodnih orodij za globoko učenje, kot je na primer sistem Caffe. Taka implementacija naučenih nevronskih mrež v vgrajenih sistemih naj izvaja klasifikacijo novih primerov in pri tem daje enake rezultate, kot če bi jih izvajali na originalnem sistemu. V svojem delu verificirajte svojo metodo s poskusi globokega učenja v izbranih zahtevnejših učnih domenah, kot je razpoznavanja vrst sadja na slikah, in ocenite rezultate glede na učinkovitost izvajanja na vgrajenem sistemu.

Zahvaljujem se mentorju akad. prof. dr. Ivanu Bratku za pomoč in vodenje pri opravljanju diplomskega dela.

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani Andrej Damjan Mlinar Groznik, z vpisno številko 63990356, sem avtor diplomskega dela z naslovom:

Izvajanje naučenih globokih nevronskih mrež v vgrajenih sistemih.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom akad. prof. dr. Ivana Bratko.
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 1. Februar 2017 Podpis avtorja: _____

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
GPU	Graphical Processing Unit	Grafična procesna enota
NN	Neural Network	Nevronska mreža
DNN	Deep Neural Network	Globoka nevronska mreža
CNN	Convolutional Neural Network	Konvolucijska nevronska mreža
DSE	Design Space Exploration	Raziskovanje prostora možnih arhitektur
BP	Back Propagation	Vzvratno razširjenje

Povzetek

Naslov: Izvajanje naučenih globokih nevronskih mrež v vgrajenih sistemih

Današnja razpoložljivost velikih računalniških zmogljivosti v obliki relativno poceni GPUjev in javno dostopni računalniški centri v oblaku omogočajo učenje velikih nevronskih mrež. Tudi vgrajeni sistemi so računalniško močnejši in cenejši, kar omogoča izvajanje večjih nevronskih mrež v le-teh. V okviru te diplomske naloge sem izbral in prikazal sestavne dele in metode za implementacijo globokih nevronskih mrež za razvrščanje slik, naučenih v oblaku in izvedenih v vgrajenih sistemih. V ta namen je bilo uporabljenih več vgrajenih sistemov z različno arhitekturo nevronskih mrež, pri tem pa sem primerjal njihove sposobnosti, zmogljivosti, uporabo virov, ceno in praktičnost uporabe. To služi kot vodilo za implementacijo sistemov za klasifikacijo slik na lahko dostopnih in nizkocenovnih vgrajenih sistemih.

Ključne besede: globoke nevronske mreže, vgrajeni sistemi, umetna inteligenca, klasifikacija objektov

Abstract

Title: Running trained deep neural networks on embedded systems

Today's availability of enormous amounts of computational power in the form of relatively cheap GPUs and publicly accessible cloud computing facilities makes the training of large deep neural networks practical. Also embedded systems have been gaining in computational power and reducing their prices, making deployment of bigger neural networks on embedded systems feasible. In the scope of this diploma thesis the necessary components and methods for the implementation of deep neural networks for image classification trained on cloud computers and deployed on embedded systems are brought together and shown working. Several embedded systems were used with different neural network architectures and their capabilities, performance, resource usage, price and practicality compared. This serves as a guide to implement state of the art image classification on easily available and low cost embedded systems.

Keywords: Deep neural networks, embedded systems, artificial intelligence, object classification

Poglavje 1 Uvod

V uvodnem poglavju te naloge podajam nekaj osnov in zgodovinsko ozadje snovi. V drugem, tretjem, četrtem in petem poglavju pa predstavljam potrebno znanje v ožjem smislu. V šestem poglavju opisujem poskusno implementacijo vgrajenih sistemov za prepoznavanje objektov z uporabo globokih nevronskih mrež. V sedmem poglavju so prikazane razne meritve opravljene na poskusnem sistemu. V osmem poglavju podajam sklepne ugotovitve.

1.1 Klasifikacija objektov na slikah

Ena težjih nalog za računalniške programe je prepoznavanje objektov na slikah [1]. Kar je za človeka zelo enostavno je bilo zelo dolgo za računalniške programe nerešljiva naloga. Različni pogledi na isti objekt, razlike v osvetlitvi, skriti deli objekta, razlike v ozadju in več vrst šuma so le nekateri izmed dejavnikov, ki računalniškim programom otežijo najti rešitev.

Matematično gledano je prostor stanj tako obsežen, da izčrpno iskanje in podobni pristopi ne pridejo v poštev.

Za reševanje tega problema so se v preteklosti uporabljale metode, ki so bile predvsem statistične narave, na primer: linearni klasifikatorji, metoda podpornih vektorjev (support vector machines) in k-najbližjih sosedov (k-nearest neighbors).

Najsodobnejše pristope uporabljajo nevronske mreže. Metoda nevronskih mrež se je izkazala za izjemno uspešno in v zadnjih letih je celotno področje naredilo velik skok naprej. Na ožjih področjih so se namreč globoke nevronske mreže izkazale celo za boljše od človeških ekspertov [2].

1.2 Nevronske mreže

Nevronske mreže so se prvič pojavile v računalništvu okoli leta 1950, najprej s Turingovimi stroji tipa B [3], in malo za tem z implementacijo Hebbove mreže in prve simulacije možganskih celic [4]. Leta 1958 je bil izumljen algoritem perceptron [5], ki predstavlja eno prvih pravih nevronskih mrež, ki je izvajala nadzorovano učenje.

Leta 1969 sta Minsky in Papert izpostavila dve ključni težavi perceptronov [6]: nezmožnost izračunati XOR (izključni ali), in učenje je bilo računsko tako zahtevno, da je postalo neizvedljivo.

Rešitev se je pojavila leta 1975 z algoritmom vzvratnega razširjanja (backpropagation) [7].Ta algoritem omogoča učenje nevronskih mrež s poljubnim številom plasti. Take mreže so tudi zmožne izračunati XOR, kakor tudi poljubne funkcije.

V 1980ih letih so bile nevronske mreže potisnjene v ozadje, saj je prišlo do pomankanja vidnega napredka, kakor tudi zaradi odkrivanja in izboljšav drugih algoritmov.

Nevronske mreže so doživele preporod v 1990ih letih. Prvi razlog je dostopnost zmogljivih računalnikov, najprej v porazdeljenih sistemih in nato še z uporabo grafičnih procesnih enot (Grafical Processing Units). Drugi razlog je eksplozivna rast podatkov, ki so na voljo za učenje, in sicer zaradi pojave interneta in popularizacije digitalnih kamer.

1.3 Vgrajeni sistemi

Računalnik, ki ima specifično funkcijo in druge električne in mehanične komponente ter je hkrati vgrajen v napravo, ki ni računalnik, imenujemo vgrajeni sistem.

Mikroprocesorji vključujejo vse glavne funkcije računalniške procesne enote v enem samem čipu ali pa v majhnem številu le-teh. Prvič so se pojavili okoli leta 1970 [8], omogočali pa so manjše in cenejše sisteme, ki so tudi porabljali manj energije za delovanje.

Skoraj istočasno so se pojavili prvi mikrokrmilniki [9]. Ti, poleg glavnih funkcij, vsebujejo tudi pomnilnik in periferne enote, vse skupaj v enem čipu. Ker so taki sistemi še manjši in v povprečju tudi cenejši, je njihova uporabnost temu primerno večja. Posebnost mikrokrmilnikov je njihova specifičnost, ki je obenem prednost in pomanjkljivost.

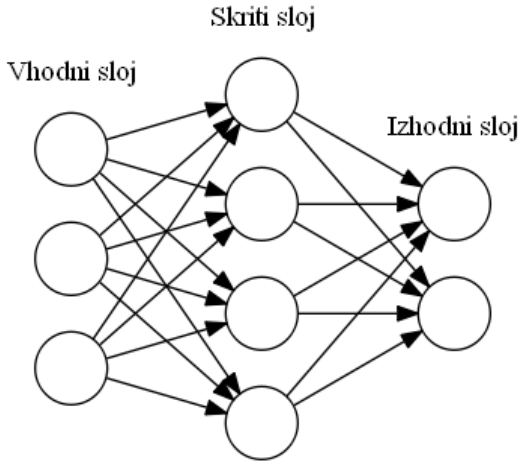
Danes so vgrajeni sistemi zelo razširjeni. Skoraj vsak zemljan ima mobilni telefon [10], ki v sebi nosi več mikroprocesorjev. Vsak avtomobil ima vsaj nekaj deset mikroprocessorjev, in avtomobili višjega cenovnega razreda preko sto [11]. Letno se izdela in proda več deset milijard mikroprocessorjev in industrija vgrajenih sistemov predstavlja znaten del svetovne ekonomije [12].

Poglavje 2 Globoke nevronske mreže

Strojno učenje je področje umetne inteligence, ki se ukvarja z algoritmi za učenje iz podatkov. Je zelo široka disciplina, ki je v zadnjih letih doživela zelo hiter razvoj. Del nje so nevronske mreže. V tem poglavju predstavljam osnovne pojme potrebne za razumevanje naloge. Obstaja več pogledov in načinov obravnave te snovi, osredotočil se bom na ožji del.

2.1 Nevronske mreže

Umetne nevronske mreže (artificial neural networks) ali pogosto kar nevronske mreže (neural networks) so računski model ali matematična funkcija, zasnovana na ideji bioloških nevronskih mrež. Sestavljene so iz umetnih nevronov, povezav med njimih in parametrov v povezavah.



Slika 1: Poenostavljena shema nevronske mreže s tremi sloji: vhodni, izhodni in skriti sloj.

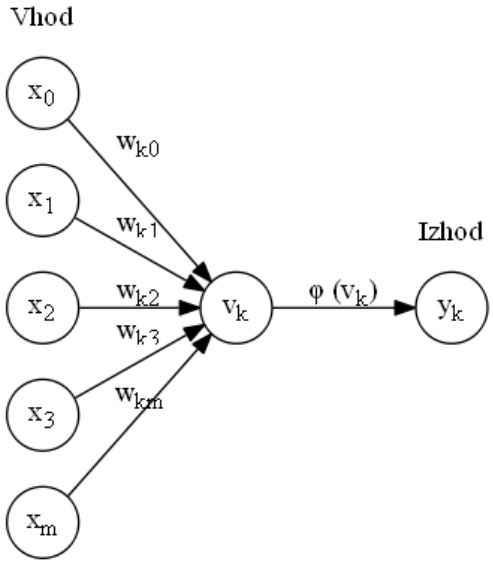
Pogosto nevronske mreže organiziramo v med seboj povezane sloje umetnih nevronov. Sloje oštevilčimo iz vhoda proti izhodu. Vsak sloj ima enega ali več nevronov. Vhod prvega sloja je povezan na vhodne podatke, izhod pa na naslednji sloj. Sloji se povezujejo tako, da se nevroni v sloju L povežejo na vhod nevronov v sloju L+1. Zadnji sloj je izhodni. Način povezave slojev je lahko zelo različen. Povezave med sloji so lahko polne (vsi izhodi so povezani z vsemi vhodi) ali samo delno zapolnjene. Poznamo tudi nevronske mreže kjer se sloji povežejo z drugimi,

oddaljenimi sloji, ki se ne nahajajo v njihovi neposredni bližini, ampak lahko preskočijo nekaj slojev, ali pa se celo povežejo nazaj, tako da ustvarijo cikel (Recurrent Neural Network, Povratna Nevronska Mreža).

Kompleksnost nevronske mreže je podana ne samo s številom nevronov in slojev ampak tudi, in skoraj bolj pomembno, s številom povezav med nevroni. Danes se uporabljajo v praksi nevronske mreže, ki lahko vsebujejo preko tisoč slojev (ResNet [13]), nekaj milijonov nevronov (VGG-16 [14]) in nekaj milijard povezav (VGG-19 [14]). Nevronske mreže s podobnimi učinki, vendar manjše velikosti in enostavnejše strukture, so z vidika funkcionalnosti bolj zanimive, zato je mogoče opaziti veliko vložka v raziskovanje manjših arhitektur nevronskih mrež (SqeezeNet [15]).

2.1.1 Umetni nevron

Umetni nevron je osnovni gradnik nevronskih mrež. Vsak nevron ima več vhodov in en izhod. Lahko ga tudi razumemo kot matematično funkcijo več spremenljivk. Izhod nevrona je seštevek uteženih vhodov skozi prenosno funkcijo.

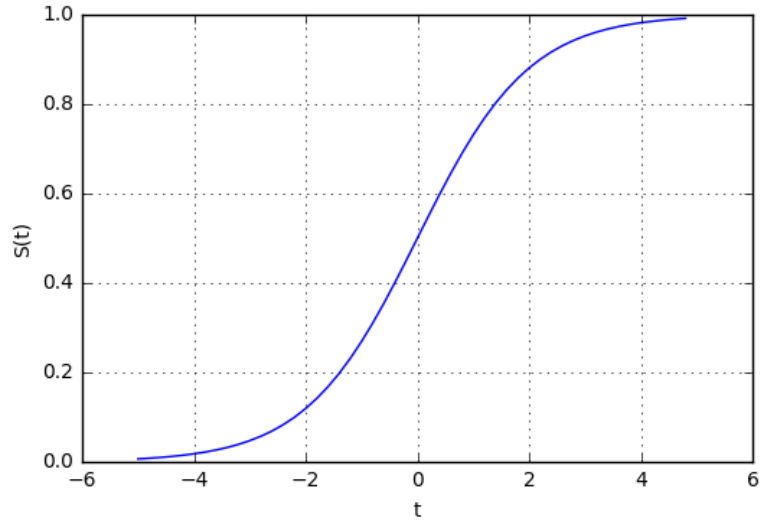


Slika 2: Umetni nevron

Prenosna funkcija je potrebna za doseg nelinearnega odziva nevrona, ki se zna tako prilagoditi bolj kompleksnim funkcijam. To velja predvsem, ko povežemo več nevronov zaporedno.

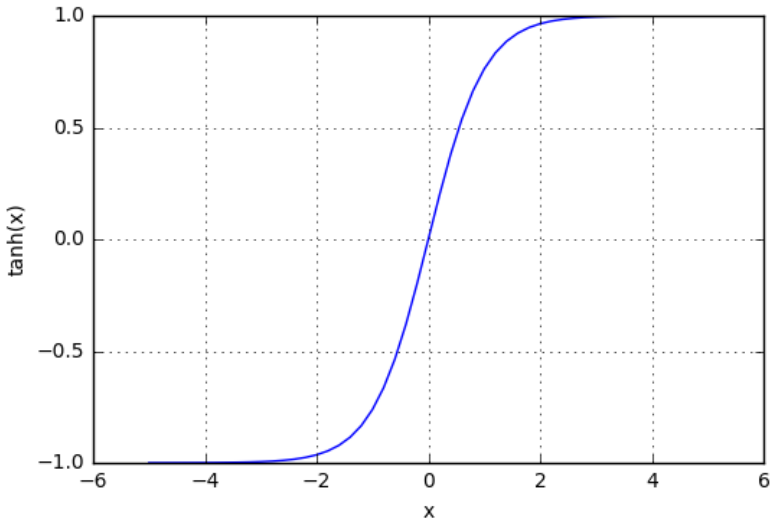
Pogosto uporabljene prenosne funkcije so sigmoidna (2.1) funkcija in hiperbolični tangens (2.2). Te funkcije imajo poleg nelinearnosti še druge uporabne lastnosti: omejijo izhod na znani interval (normalizacija) in so odvedljive.

$$S(t) = \frac{1}{1 + e^{-t}} \tag{2.1}$$



Slika 3: Sigmoidna funkcija

$$\tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2.2}$$



Slika 4: Hiperbolični tangens

2.1.2 Matematična predstavitev

Na nevronske mreže lahko gledamo matematično. Posamični nevron lahko podamo s formulo:

$$y_k = \varphi\left(\sum_{j=0}^{m} w_{kj} x_j\right) \tag{2.3}$$

Kjer je y_k izhod, φ prenosna funkcija, w_{kj} so uteži in x_j so vhodi.

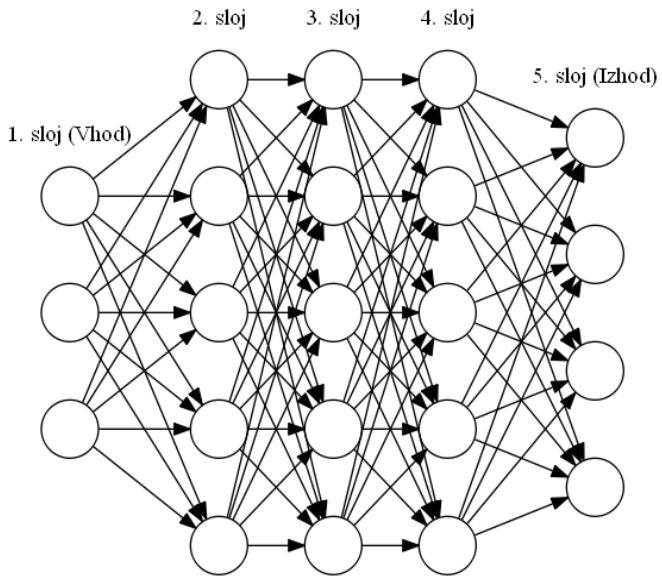
Posledično en sloj nevronske mreže predstavimo kot matrično operacijo:

$$y = \varphi(W\,X) \tag{2.4}$$

Kjer je y_k vektor izhodov, φ prenosna funkcija, W matrika uteži na povezavah, X vektor vhodov.

2.2 Globoke nevronske mreže

Globoke nevronske mreže imajo več skritih plasti med vhodom in izhodom. Te skrite plasti omogočajo, da se mreža organizira hierarhično in se je tako sposobna naučiti bolj zapletenih nelinearnih funkcij.



Slika 5 Večslojna (globoka) nevronska mreža

2.3 Učenje

Privlačnost nevronskih mrež je v možnosti učenja iz podatkov. Učenje pomeni iskanje optimalne postavitve parametrov (uteži v povezavah) v dani arhitekturi nevronske mreže za reševanje določenega problema.

Učenje nevronskih mrež je optimizacijski problem, pri čemer se s pomočjo hevristične funkcije napake približamo optimalni rešitvi.

Nenadzorovano učenje se uporablja za porazdelitev podatkov v kategorije, ki jih algoritem odkriva sam, na podlagi vhodnih podatkov.

Nadzorovano učenje poteka tako, da dobi algoritem učenja učne podatke označene, to je, za vsak podani vhod dobi še zaželen izhod. Ti pari so v naprej določeni, običajno s pomočjo človeka. Algoritem učenja pa na podlagi teh parov prilagaja parametre mreže. Cilj je, da tako naučena nevronska mreža poda pravilen odgovor za vsak možni vhod.

Algoritmov za iskanje optimalne arhitekture nevronske mreže še ne poznamo. Gradnja nevronskih mrež je danes umetnost, ki sloni na izkušnjah. To je področje intenzivnega raziskovanja v zadnjih letih in stalnega objavljanja novih rezultatov.

2.3.1 Vzvratno razširjanje

Algoritem vzvratnega razširjenja (back propagation) [7] je najbolj uporabljena metoda za učenje nevronskih mrež. Cilj algoritma je najti primerne uteži v nevronski mreži, tako da bo dala pravilne izhodne vrednosti na podanem vhodu. Delovanje algoritma je razdeljeno v tri koraki: razširjanje naprej vhodnega podatka skozi nevronske mreže, računanja napake na izhodu, ter popravljanje uteži nazaj. Algoritem se ponavlja veliko krat in se z vsako ponovitvijo približuje optimalni rešitvi. Algoritem razširjenja nazaj potrebuje znani pričakovani izhod za vsak vhodni podatek učne množice in je zato način nadzorovanega učenja.

V prvem koraku se uporablja vhodni podatek iz učne množice. Ta podatek se razširja skozi nevronske mreže naprej, to je iz vhoda proti izhoda, da dobimo izhodne vrednosti učnega primera trenutne nevronske mreže.

V drugem koraku se na izhodu uporablja funkcija napake (cost function ali error function) za določitev vrednosti napake. Ta vrednost je realno število, ki predstavlja razdaljo med dobljenim in pričakovanem izhodom.

V tretjem koraku se s pomočjo optimizacijske metode posodobijo vrednosti uteži v nevronski mreži, iz izhoda proti vhodu. Za vsak izhodni nevron računana napaka se uporablja za posodobitev uteži vhodov nevrona. Prispevek vsake uteži se računa kot parcialni odvod prenosne funkcije nevrona, glede na tisti vhod. Ta odvod se množi s trenutno vrednostjo napake in z parametrom alfa (razmerje učenja, learning rate), da se dobi vrednost popravka uteži. Sorazmerna vrednost napake se potem uporablja pri določitvi napake vhodnega nevrona, in celoten postopek se ponavlja za vsaki nevron proti vhodu.

Hitrost učenja je odvisna od parametra alfa. Pri večji vrednosti parametra alfa se bo učenje hitreje pomikalo proti rešitvi, ampak lahko rešitev tudi preskoči. Manjši parameter alfa je bolj točen, ampak upočasni proces. Pogosto uporabljena rešitev je prilaganje hitrosti učenja, z večjem koraku na začetku in postopno zmanjšanje.

Učenje se ponavlja na učnih podatkih tako dolgo, dokler nimamo zadovoljivih rezultatov, to je, da je funkcija napaka na izhodu dovolj majhna.

To je kratek opis algoritma. Razumevanje algoritma je ključnega pomena. Za bolj podrobno obravnavo priporočam ogled virov [16], [17], [18] in [19].

2.4 Nekateri problemi z nevronskimi mrežami

Pogost problem strojnega učenja nasploh je pretirano prilagajanje učnim podatkom. V globokih nevronskih mrežah je to še posebej pomembno, ker so lahko možnosti parametrizacije določene nevronske mreže veliko večje od razpoložljivih učnih podatkov. V tretjem poglavju predstavljam nekatere metode za obvladanje tega problema.

Če funkcija napake ni konveksna obstaja možnost stagnacije v lokalnih minimumih, doseganje globalnega minimuma ni zagotovljeno. V enostavnih primerih je lahko določiti funkcijo napake, ki je konveksna, ampak v realnih primerih, kjer je dimenzija problema zelo velika in množica možnih vhodov praktično neomejena, je to zelo težka naloga. Zato se po navadi uporabljajo funkcije napake, ki veljajo samo za manjšo delovno podmnožico učnih primerov (training batch) in se ta funkcija spreminja vsakokrat, ko vzamemo novo delovno učno podmnožico. Ta proces ponavljamo velikokrat in vsebino množic spremenimo. Izkazuje se, da v praksi to ni velik problem [20].

Izginjajoč gradient (vanishing gradient) [21] je problem, ki ga srečamo posebej pri učenju globokih nevronski mrež. Pojavi se, ker je posodobitev uteži v povezavah nevronske mreže odvisna od prispevka tiste povezave, ki je tudi odvisna od parcialnega odvoda prenosne funkcije. Pogosto uporabljene prenosne funkcije sigmoid in hiperbolični tangens imajo

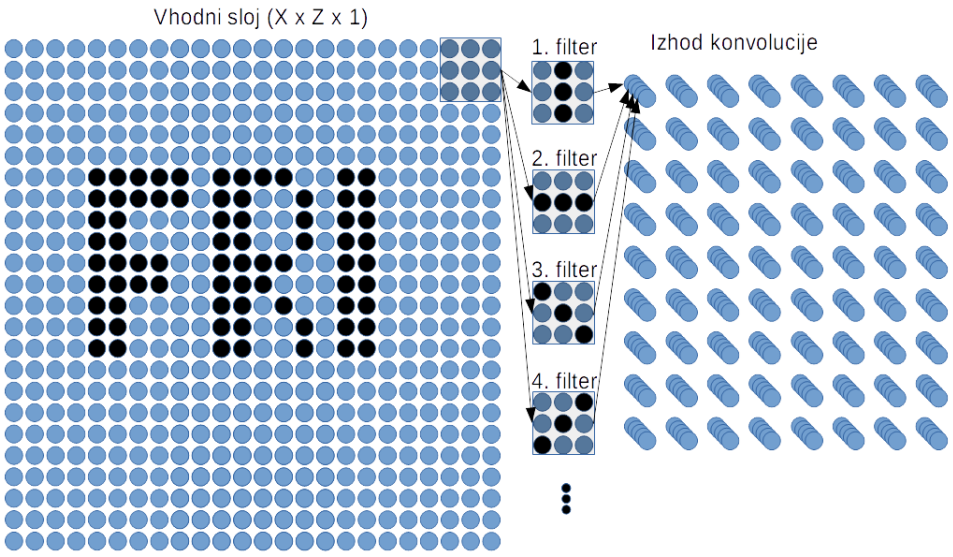
vrednost odvoda blizu ničle, ko prenesemo ta odvod verižno skozi sloje mreže, je vrednost čedalje manjša. To pomeni, da bolj kot je sloj oddaljen od izhoda, bolj počasi ga bo možno učiti. Zelo počasno učenje začetnih slojev je velik problem, za katerega so se v zadnjem času pojavile rešitve; nekatere bodo predstavljene v tretjem poglavju.

2.5 Konvolucijske nevronske mreže

Konvolucijske nevronske mreže delujejo podobno kot primarni vidni korteks in vidni asociacijski predel v možganih.

Ključni del konvolucijske nevronske mreže je konvolucijski sloj. Ta sloj je sestavljen iz več filtrov z možnostjo učenja (Slika 6). Vsak filter ima majhno polje zaznavanja in se ga premika po celem vhodnem polju ter tako računa novo polje korelacije med filtrom in vhodnim slojem. V vsakem sloju je več filtrov in rezultat vsakega filtra se nalaga v tri dimenzionalno izhodno polje. Filtri se lahko premikajo z različnim zamikom in lahko pokrivajo že predelane dele vhoda ali ne.

Konvolucijski filtri imajo pogosto majhno širino in dolžino ter isto globino kot vhodni sloj. Izhod konvolucijskega sloja se lahko vpelje v navadno nevronsko mrežo, ali pa se ga ponovno obravnava z drugim slojem konvolucije.

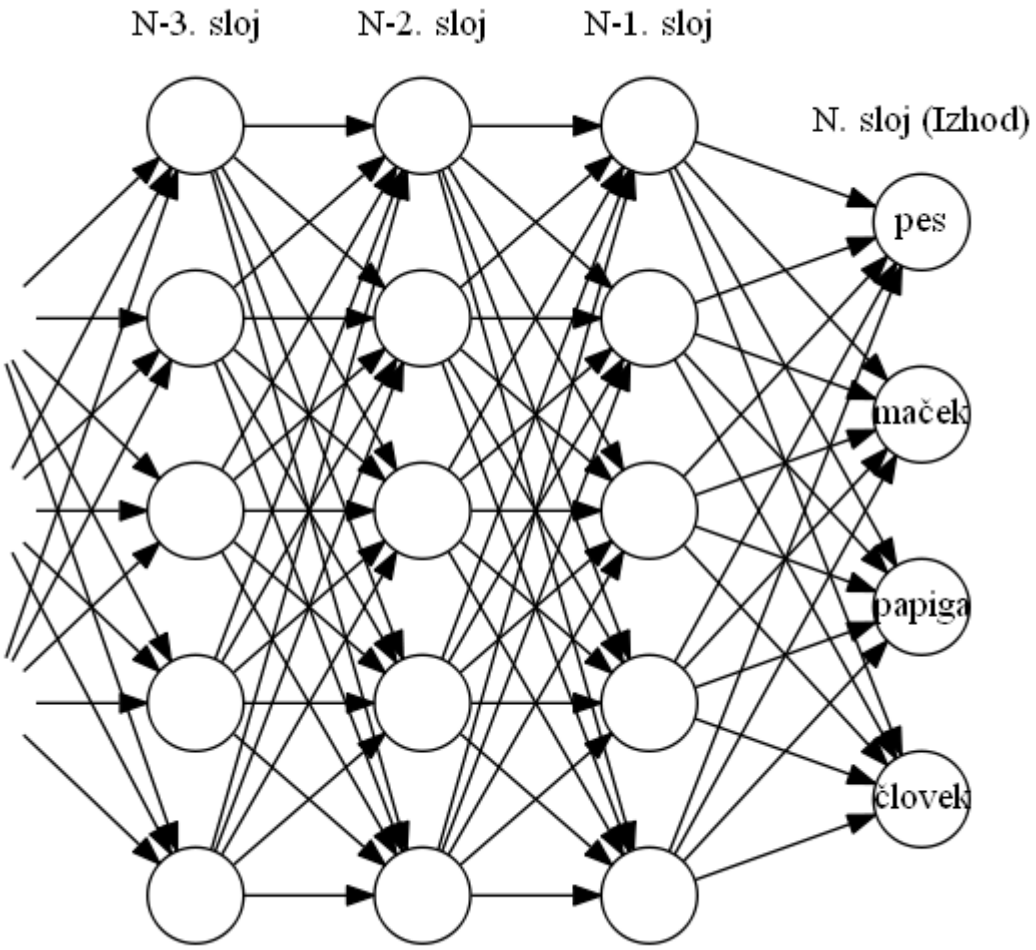


Slika 6: Konvolucijski sloj nevronske mreže:
Vhodni sloj ima dimenzije X x Y x 1 (monokromatska slika);
vsak filter je velikosti 3 x 3 x 1, in teh je N;
izhodni sloj ima torej velikosti (X+1-3) x (Y+1-3) x N

2.6 Klasifikacija objektov z uporabo nevronskih mrež

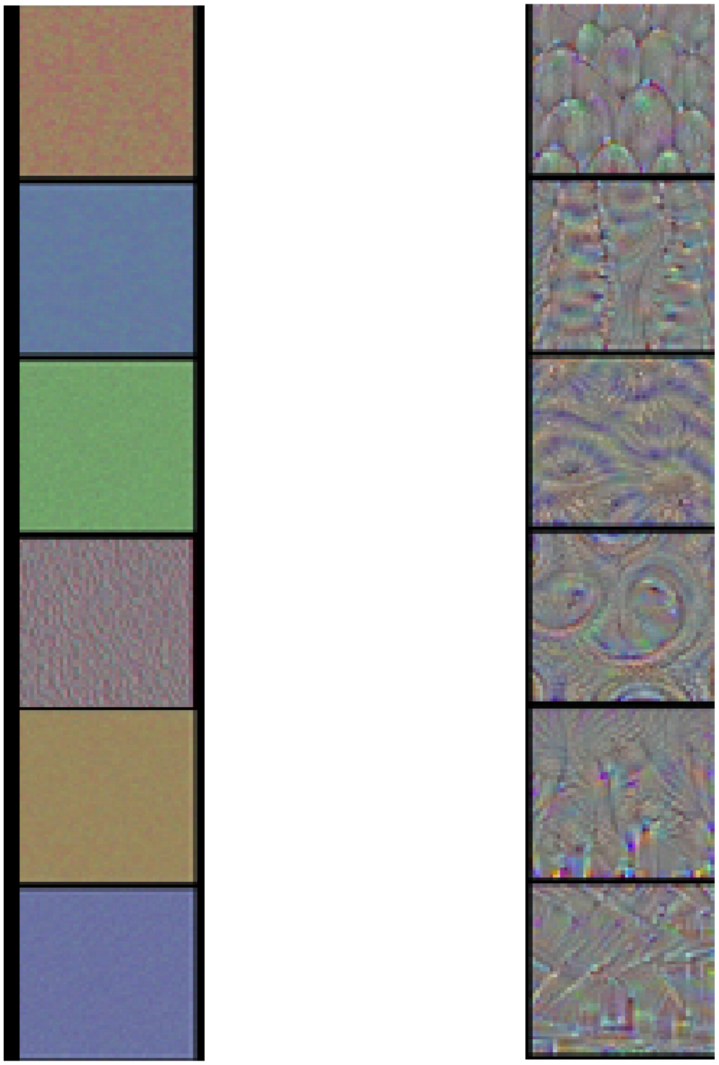
Konvolucijske nevronske mreže so se v praksi izkazale za zelo uspešne pri prepoznavanju objektov na slikah. Začetni sloji so konvolucijski, zadnji sloji pa navadni s polnimi povezanimi plastmi. Izhodni sloj ima po en nevron za vsak tip objekta, ki ga želimo prepoznati.

Pri učenju takšne nevronske mreže je funkcija napake torej takšna, da »ve«, da mora biti največ samo en izhodni nevron aktiven. Ker je izhodna vrednost vsakega nevrona številka, s primerno normalizacijo, jo lahko vzamemo kot verjetnost za napoved tistega razreda, ki ga izhodni nevron predstavlja.



Vizualizacija že naučene konvolucijske nevronske mreže dobro prikazuje, kako se informacije organizirajo hierarhično. V nižjih slojih vidimo osnovne geometrične vzorce in v višjih slojih sestavljene elemente. Za tako vizualizacijo lahko uporabimo dekonvolucijsko mrežo [22], ki pretvori aktivacije nevronov nazaj v slike.

Na Slika 7 vidimo primer vizualizacije nekaj filtrov konvolucijske arhitekture nevronske mreže VGG16, ki sem jo sam učil. V tem primeru 1. sloj prikazuje prepoznavanje bolj enostavnih vzorcev in barv, 5. sloj prepozna bolj kompleksne vzorce, torej kombinacija filtrov v spodnjih nivojih.



Slika 7: Vizualizacija nekaj filtrov že naučene konvolucijske nevronske mreže, 1. sloj (levo) in 5. sloj (desno)

Poglavje 3 Arhitektura nevronskih mrež

V tem poglavju bom opisal nekaj najbolj znanih arhitektur nevronskih mrež, njihove značilne lastnosti in nove ideje, ki so jih vpeljali na področje nevronskih mrež.

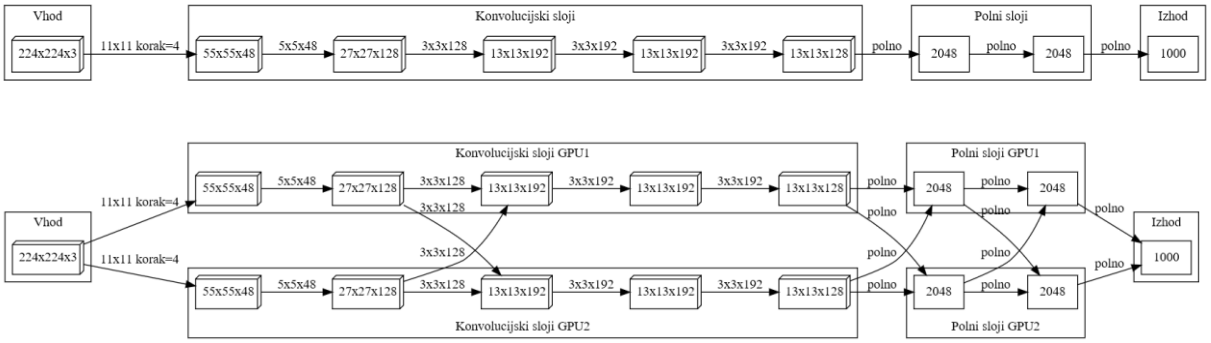
3.1 AlexNet (2010, 2012)

Alex Krizhevsky, Ilya Sutskever in Geoffrey Hinton so v letu 2012 objavili članek [23], kjer opisujejo zmagovalni algoritem na ILSVRC 2012 (ImageNet Large-Scale Visual Recognition Challenge).

AlexNet je dosegel izjemen rezultat in je naredil preskok naprej na področju prepoznavanja slik. Na tekmovanju ILSVRC je bila tisto leto podana učna množica slik s tisoč razredi in približno tisoč slikami za vsak razred, kar znaša v celoti približno 1,2 milijona slik.

Učenje nevronske mreže je potekalo na dveh grafičnih procesnih enotah GTX 580 s 3GB pomnilnika. Učenje je trajalo od pet do šest dni. Grafične procesne enote vsebujejo strojno opremo, specifično za hitro množenje matrik, kar omogoča znatno pohitritev postopka učenja. Faktor pohitritve z modernimi GPUji znaša, v primerjavi z uporabo splošnih centralnih procesorjev, nekaj deset krat [24].

Arhitektura nevronske mreže ima pet konvolucijskih slojev, nekaj vmesnih izpadnih slojev, nekaj slojev za združevanje in polno povezane sloje pri izhodu. Mreža je porazdeljena v dveh delih, s čimer se je mrežo dalo učinkovito učiti v dveh grafičnih procesnih enotah paralelno. Danes se ta arhitektura uporablja v enem delu, ker so grafične procesne enote dovolj zmogljive, da naložijo in učijo celotno nevronsko mrežo. Ker računanje poteka v eni sami enoti, je to hitreje, ker vmesnih rezultatov za sinhronizacijo ni potrebno prenašati med enotama.

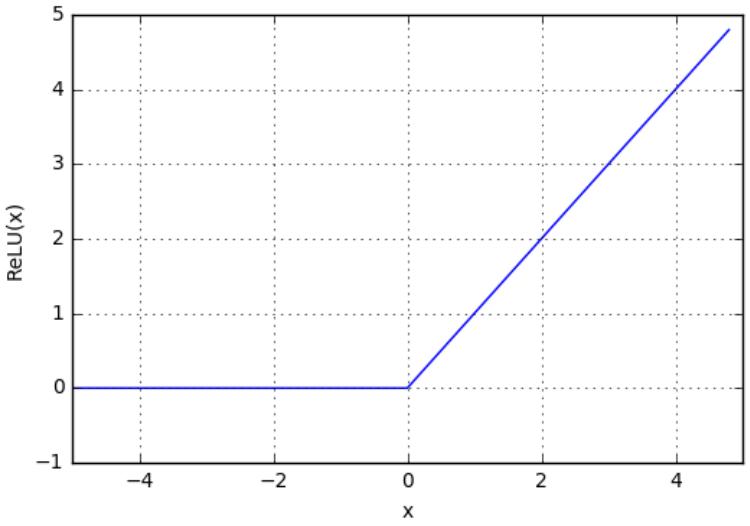


Slika 8: Arhitektura AlexNeta, v enem in v dveh tokovih

3.1.1 ReLU funkcija

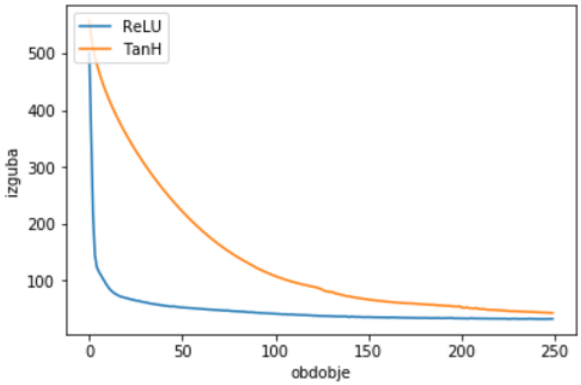
Ena od inovaciji, ki jih je uvedel AlexNet, je uporaba ReLU (Rectified Linear Unit) (3.1) (Slika 9) [25] kot prenosne funkcije nevronov.

$$f(x) = \max(0, x) \tag{3.1}$$



Slika 9: Funkcija ReLU

V meritvah, ki so jih naredili avtorji članka, navajajo šestkratno pohitritev pri učenju. Na sliki (Slika 10) prikazujem primer iz mojih poskusov na standardni množici podatkov (Boston house prices) kako nevronska mreža, ki uporablja ReLU funkcijo (modra črta) hitreje konvergira proti rešitvi, v primerjavi z nevronsko mrežo, ki uporablja hiperbolični tangens (oranžna črta).



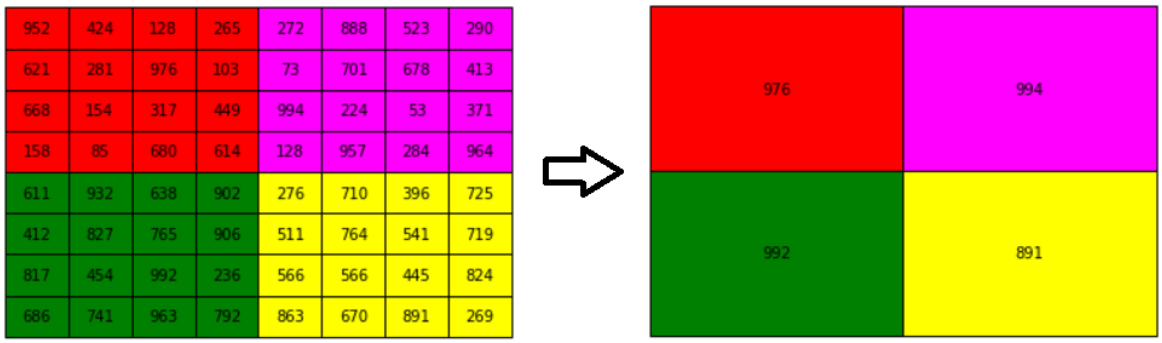
Slika 10: Prikaz pohitritve pri učenju nevronske mreže pri uporabi ReLU

Pohitritev ima več razlogov. Razpršena aktivacija nevronov je verjetno glavni faktor, saj nevroni, ki imajo izhodno vrednost nič, ne sodelujejo v naslednjem nivoju. Groba ocena je, da je v vsakem trenutku aktivnih, to je njen izhod je nad ničlo, samo približno polovica nevronov. Kot drugi dejavnik lahko sklepamo, da je razširjenje gradientov bolj učinkovito, ker ti niso pod vplivom odvodov z zelo nizko vrednostjo. Manjši prispevek je tudi ta, da računanje te funkcije vzame manj časa, kot računanje drugih prenosnih funkciji.

3.1.2 Združevanje s prekrivanjem

V nevronskih mrežah se uporabljajo tudi sloji, kjer se združujejo rezultati sosednjih nevronov. Pogosto se to naredi tako, da se filter premika skozi ravnino in iz polja, ki ga pokriva, vzame najvišjo vrednost. Slika (Slika 11) prikazuje filter velikosti 2x2 s korakom 2, nad poljem velikosti 4x4.

Krizhevsky et al. pa so na podlagi meritev odkrili, da uporaba takih filtrov, kjer je korak manjši od velikosti filtra, torej, da prihaja do prekrivanja, pripelje do boljših rezultatov. Z uporabo te tehnike je njihova nevronska mreža bolje klasificirala slike in se tudi težje pretirano prilegala učnim podatkom.



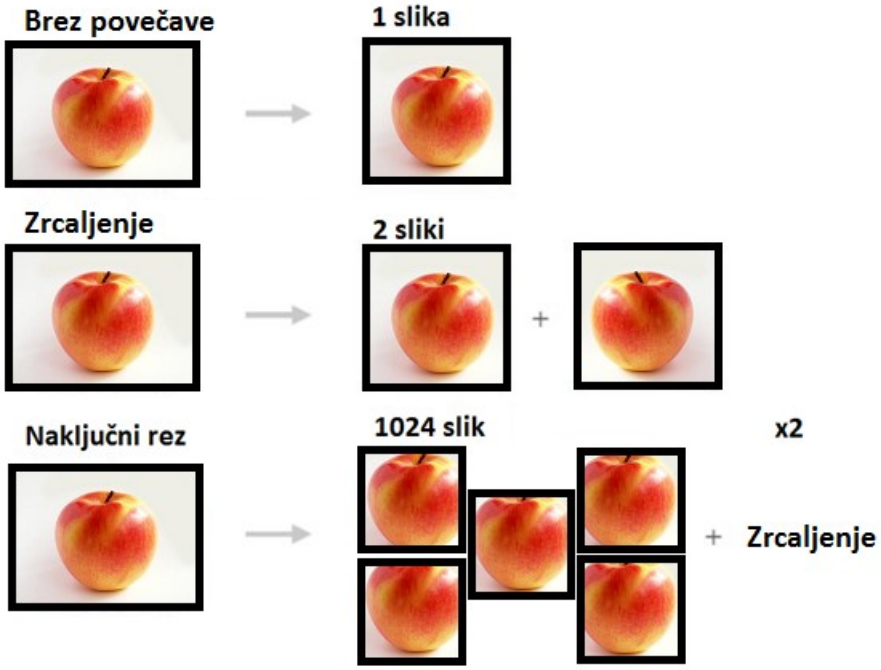
Slika 11: Združevanje maksimumov (max pooling)

3.1.3 Bogatenje podatkov

Ker njihova nevronska mreža vsebuje šestdeset milijonov parametrov, količina učnih podatkov ni bila zadostna za učenje brez pretiranega prilagajanja.

Da bi ustvarili več učnih slik so posegli po zelo enostavnih trikih. Za vsako sliko iz učne množice so vzeli večje število naključnih rezov skoraj celotne vhodne slike in njihova horizontalna zrcaljenja (Slika 12). Na primer, iz učne slike velikosti 256x256, so rezali več vhodnih slik velikosti 224x224. Na ta način so povečali število učnih slik za faktor 2048. V nadaljevanju so rahlo spremenili barve in osvetlitev z metodo analize glavnih komponent (principal component analysis, PCA).

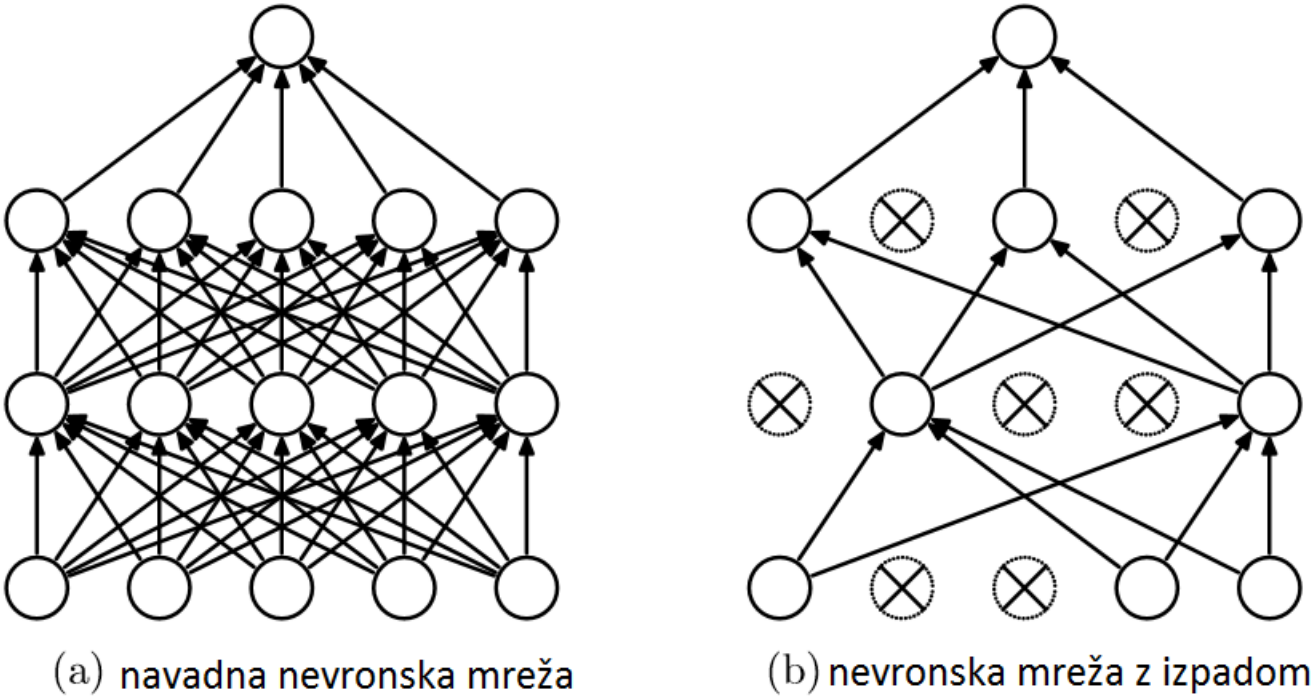
Takšno procesiranje je relativno poceni in ga je bilo mogoče izvesti sproti na glavnem procesorju učnega računalnika, ne da bi pri tem vplivali na algoritem učenja, ki je tekel na GPUju. Ker tako pridobljenih slik ni bilo potrebno shranjevati, je bilo prihranjeno veliko prostora.



Slika 12: Bogatenje učnih podatkov

3.1.4 Izpad

Združevanje napovedi večjih modelov je zelo učinkovit način za zmanjševanje testnih napak [26] [27] [28], vendar ne najbolj praktično, saj traja učenja enega samega modela veliko časa. Zelo učinkovita verzija te tehnike je izpad (drop-out) (Slika 13). Izpad temelji na naključnem izpadu nevronov z verjetnostjo 0.5 v fazi učenja. Nevroni, ki izpadejo v fazi učenja ne prispevajo k rezultatu in se tudi ne vključujejo v razširjenja nazaj. To zmanjša verjetnost skupne prilagoditve več nevronov, saj posamezen nevron ne more računati na prisotnost drugih nevronov. Tako se je torej nevron prisiljen naučiti se bolj robustnih vzorcev, ki morajo delovati ne glede na naključni izpad drugih nevronov.



Slika 13: Primer tehnike izpada pri majhni nevronski mreži

3.2 GoogLeNet

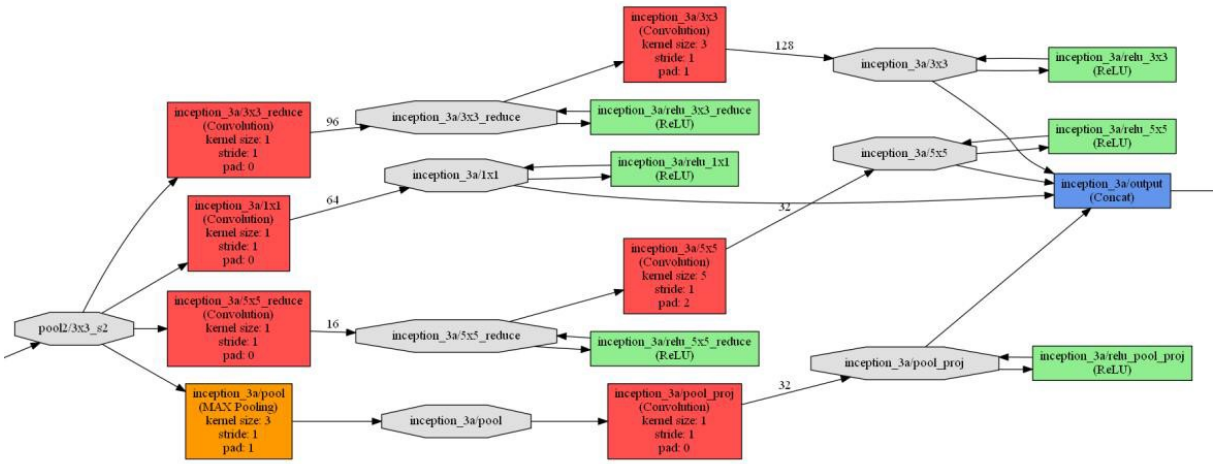
GoogLeNet [29] je bila zmagovalka tekmovanja ILSVRC 2014, z zelo majhno napako pri napovedi (top-5 napaka 6.7%, pri top-5 napaki se šteje odgovor mreže kot pravilen, če je pravilni razred med petimi najvišje ocenjenimi razredi). Predvsem presenetljivo je, da je GoogLeNet mreža tako točno klasifikacijo dosegla pri uporabi 12 krat manj parametrov kot AlexNet dve leti prej.

Ta nevronska mreža uporabi tako imenovane Inception module, zložene drug na drugega, namesto zaporedja enostavnih konvolucijskin in izpadnih slojev (Slika 14).

Zaradi velikosti nevronske mreže jo je težko v celoti prikazati v eni strani. Tok podatkov je iz zgornje proti spodnji strani. Za boljšo vizualizacijo je vsak sloj obarvan glede na njegov tip: konvolucijski sloji rdeče, združevanje s prikrivanjem oranžno, ReLU zeleno, konkatenacija oz. združevanje svetlo modro, notranji produkt vijolično.

Slika 14: Arhitektura nevronske mreže GooLeNet, prvi modul inception v zelenem pravokotniku, model uporablja več le teh. Namen te slike je le ilustracija strukture, medtem ko besedilo v posameznih elementih mreže ni pomembno.

Vsak modul (Slika 15) uporablja vhodne podatke na več paralelnih konvolucijskih podmrežah in rezultate potem združuje. To omogoča, da namesto, da bi se moral odločati za eno določeno velikost konvolucije na vsakem sloju, lahko naredi več konvolucij zapored in uporabi tako najboljši rezultat.



Slika 15: Inception modul (angl. Inception module), slika generirana iz modela z uporabo ogrodja Caffe

3.3 SQUEEZENET (2015)

SqueezeNet [15] je arhitektura nevronske mreže, ki dosega podobno točnost napovedovanja kot AlexNet, pri tem pa uporablja 50 krat manj parametrov.

Zaradi rasti kompleksnosti načrtovanja novih arhitektur iz osnovnih gradnikov so se začeli pojavljati v literaturi kompleksni moduli, ki se uporabljajo za gradnjo nevronskih mrež. Eden izmed teh primerov je, kot smo že videli, inception modul uporabljen v GoogLeNet [29]. SqueezeNet razvije svoj modul z imenom Fire modul, katerega glavni cilj je doseganje zadovoljivih rezultatov z uporabo majhnega števila parametrov.

V članku se vpeljeta pojma mikroarhitekture in makroarhitekture nevronskih mrež. Mikroarhitektura je pogled na zgradbo modulov, njihovo notranjo organizacijo in izbiro dimenzij filtrov. Makroarhitektura pa je pogled na celoten sistem, kako izbirati module, povezave med njimi, število slojev in druge pomožne gradnike.

Tudi ta nevronska mreža je dovolj velika, da jo je težko prikazati v eni sami strani. Zato so tudi sloji obarvani, tako ko v sliki GoogLeNet.

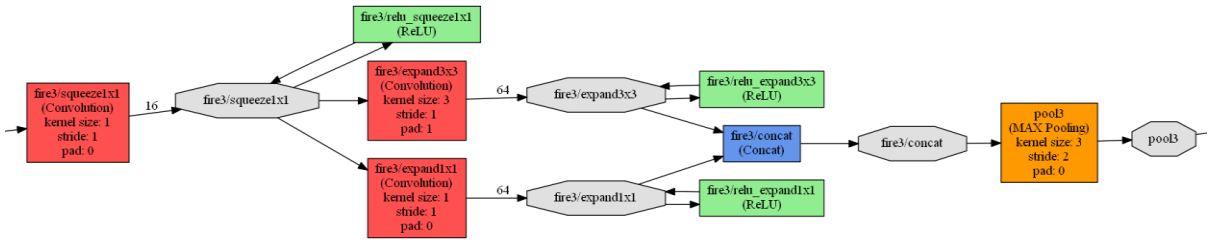
Slika 16: Arhitektura nevronske mreže SqueezeNet, prvi modul Fire v zelenem pravokotniku, model uporablja več le teh

Kot metodo iskanja najbolj učinkovitih arhitektur je v članku predlaganih nekaj idej za raziskovanje prostora možnih arhitektur (Design Space Exploration, DSE). V članku je opravljen specifičen pregled na mikroarhitekturnem nivoju, kako izbira meta parametrov (število filtrov, velikost filtrov) vpliva na velikost in točnost modulov. Na makroarhitektrunem nivoju pa je opravljena primerjava velikosti in točnosti različnih mrež, kjer dodajajo obvozne povezave (bypass connections), to je, povezave modulov, ki preskočijo druge module v hierarhiji.

SquezeNet uresničuje svoj cilj zmanjšanja števila parametrov s tremi strategijami, predstavljenimi v nadaljevanju.

Prva strategija: zamenjava nekaterih 3x3 filtrov s filtri velikosti 1x1. To zmanjšuje število parametrov v teh filtrih za devet krat. Pričakovali bi, da bi se pri tem tudi zmanjšala točnost napovedanja nevronske mreže, vendar empirične meritve kažejo, da temu ni tako.

Druga strategija (fire modul): zmanjšanje števila vhodov na filtre velikosti 3x3. Pri tem se uporablja sloje za stiskanje (squeeze layers), ki s filtri velikosti 1x1 zmanjšujejo število parametrov. Izhod tega sloja pa se napelje na sloj za razširitev (expand layer), kjer je mešanica 1x1 in 3x3 filtrov (Slika 17).



Slika 17: Modul Fire, stiskanje in razširitev

Tretja strategija: zmanjšanje velikosti aktivacijskega polja čim kasneje. Ta strategije prikazuje učinkovit način povečevanja točnosti. Pogosto se v nevronskih mrežah uporablja pri konvolucijskih filtrih korak, ki je večji od ena. To zmanjšuje polje, kjer je nevronska mreža aktivna in posledično zmanjšuje njeno moč, saj to izključuje nevrone v višjih slojih.

Poglavje 4 Caffe

V tem poglavju bom na kratko opisal odprto kodno programsko opremo za globoko učenje Caffe [30]. To ogrodje bom uporabil v svojih poskusih. Caffe je ogrodje za strojno učenje. Razvit je bil v Centru za računalniški vid in učenje univerze v Berkeleyju (Berkeley Vision and Learning Center, BVLC). Objavljen je pod BSD licenco [31].

Njegovi glavni cilji so:

- Ekspresivnost: modeli in optimizacije se opisujejo v tekstovnih datotekah in ne s programsko kodo.
- Hitrost: ključnega pomena za raziskave in tudi industrijo. Caffe lahko procesira slike v centralni procesni enoti (CPU) ali v grafičnih procesnih enotah (GPU).
- Modularnost: prilagodljivost novim nalogam.
- Odprtost: za čim hitrejši razvoj znanosti in za uporabnost rezultatov je potrebna prosto dostopna izvorna koda, referenčni modeli in možnost ponavljanja poskusov.
- Skupnost: akademsko raziskovanje, prototipi v startup podjetjih in industrijske aplikacije si delijo moč in prednosti z razvojem v projektu pod BSD licenco.

Caffe je napisan v programskem jeziku C++ in ima vmesnike za uporabo v jezikih C++, Python in Matlab.

4.1 Arhitektura modela Caffe

Osnovni gradniki modela v ogrodju Caffe so blob, sloj (layer) in mreža (net).

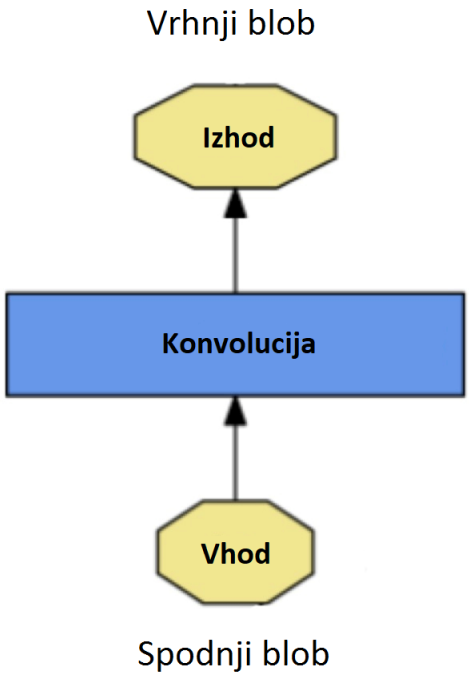
4.1.1 Blob

Blob je ovoj okoli podatkov, ki se uporabljajo v Caffeju. Matematično gledano je blob matrika N dimenzij. Razred blob skrije kompleksnosti komunikacije in sinhronizacije med glavnim procesorjem in grafično procesno enoto ter prenos podatkov med njima.

Na primer, skupina slik za obdelavo (image batch) je matrika štirih dimenziji (n, k, h, w), kjer je n število slik, k število kanalov na sliki (tipično tri, za rdečo, zeleno in modro), h višina slike, in w širina slike. Kot drugi primer, konvolucijski sloj s 96 filtri, vsak z vhodno dimenzijo 11x11 in s 3 vhodnimi kanali (en za vsako primarno barvo), bo matrika velikosti 96 x 3 x 11 x 11.

4.1.2 Sloj

Sloj je bistvo vsake nevronske mreže in enota računanja. Sloji se uporabljajo za konvolucijo filtrov, za računanje nelinearnih transformaciji, za normalizacijo vrednosti, za nalaganje vhodnih podatkov, za računanje napak, za računanje notranjih produktov. Vsak sloj zna izračunati prehod podatkov naprej in razširjenje nazaj. Vsak sloj vzame vhod iz spodnjega bloba in proizvaja izhod kot vrhnji blob (Slika 18).



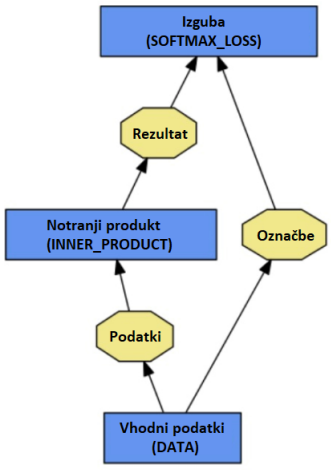
Slika 18: Pogled na en sloj v Caffeju z vhodnim (spodnjem) blobom in izhodim (zgornjim) blobom

Caffe ponuja obširno zbirko že razvitih slojev. Razvoj novih slojev je dokaj enostaven. Pri tem je potrebno implementirati začetno postavitev, izvesti izračune prehoda naprej ter izračune razširjenja nazaj.

4.1.3 Mreža

Mreža v Caffeju je množica slojev povezanih v usmerjen graf. Caffe poskrbi za vso potrebno infrastrukturo v ozadju ter za pravilno izvajanje izračunov prehoda naprej in razširjenja nazaj.

Mreža se definira kot množica slojev in njenih povezav v tekstovni datoteki z uporabo posebnega modelskega jezika, definiranega za ta namen. Na spodnji sliki (Slika 19) vidimo primer enostavnega klasifikatorja, naslednja Slika 20 pa podaja njegovo definicijo.



Slika 19: Graf enostavnega klasifikatorja

```
name: "LogReg"
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  data_param {
    source: "input_leveldb"
    batch_size: 64
  }
}
layer {
  name: "ip"
  type: "InnerProduct"
  bottom: "data"
  top: "ip"
  inner_product_param {
    num_output: 2
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip"
  bottom: "label"
  top: "loss"
}
```

Slika 20: Definicija klasifikatorja iz (Slika 19) v jeziku za modeliranje mrež

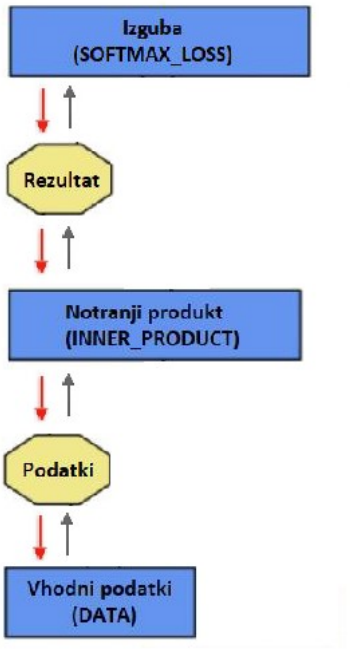
Definicijo nevronske mreže se shrani v tekstovni datoteki s končnico .prototx. Naučene nevronske mreže se shranijo v binarni datoteki s končnico .caffemodel.

4.2 Osnovno računanje v večslojnih sestavljenih modelih: naprej in nazaj

Bistvena naloga ogrodja Caffe je izvajanje izračunov naprej in nazaj v nevronski mreži.

Prehod naprej računa izhod danega vhoda. V tem načinu delovanja Caffe sestavlja rezultate vsakega sloja, da izračuna funkcijo predstavljeno z modelom.

Pri vzvratnem prehodu, Caffe računa gradient glede na funkcijo napake in avtomatsko širi gradient skozi celotno mrežo, to je razširjenje nazaj (Slika 21).



Slika 21: Razširjenje nazaj v Caffeju [30]

4.3 Učenje: reševalnik, funkcija napake

Reševalnik vodi optimizacijo modela s koordiniranjem računanj naprej ter razširjenja nazaj, da dobi gradient vseh parametrov in jih postopoma posodobi, s čimer zmanjša funkcijo napake.

Caffe pozna več reševalnikov: stohastični gradientni spust, adaptivni delta, adaptivni gradient, adam, Nesterov pospešeni gradient, RMSProp. Omenjeni reševalniki so podrobneje opisani v [32].

Reševalnik postavi vso potrebno infrastrukturo in ustvari nevronsko mrežo za učenje in morebitne testne nevronske mreže za testiranje. Poleg tega tudi iterativno optimizira mrežo z računanjem naprej ter razširjanjem nazaj, periodično ocenjuje testne mreže ter omogoča posneti trenutno stanje modela in reševalnika skozi celotni proces.

V vsaki iteraciji reševalnik izračuna prenosno funkcijo vsakega sloja, izračuna obratno razširjanje v vsakem sloju, vključi posodobitev parametrov glede na izbrano metodo reševanja in posodobi stanje reševalnika (hitrost učenja in drugi meta parametri).

4.4 Katalog slojev

Caffe pozna zelo veliko tipov slojev in vsak tip sloja ima tudi več implementaciji. Tipi slojev so sledeči: **Podatkovni sloji** (za branje in pisanje vhodnih podatkov in parametrov ter izpisovanja informaciji o poteku), **vidni sloji** (konvolucijski, združevalni, dekonvolucijski, rezalni), **povratni sloji**, **standardni sloji** (notranji produkt, izpadni), **normalizacijski sloji**, **aktivacijski sloji** (za implementacijo nelinearnih in drugih prenosnih funkcij), **pomožni sloji** in **sloji za računanje napak** (vodenje učenja in optimizacije). Podroben opis vseh slojev lahko najdemo v [33].

4.5 Vmesniki

Caffe je napisan v programskem jeziku C++ in se lahko uporablja kot knjižnica. Za vsakdanjo uporabo pa sistema Caffe ni potrebno posebej prevajati, ampak ga lahko neposredno vodimo preko enega izmed sledečih vmesnikov, ki so že na voljo: ukazna vrstica, python in matlab. V tem podpoglavju bom opisal glavne lastnosti in uporabo prvih dveh, ker sta pomembna za razumevanje implementacije testnega sistema.

4.5.1 Ukazna vrstica

Vmesnik Caffe za ukazno vrstico omogoča hitro in enostavno učenje, testiranje, izvajanje in merjenje časa, pri tem pa ni potrebno programirati. Na Slika 22 lahko vidimo informacijo o uporabi, ki jo ponuja sam program.

```
caffe: command line brew
usage: caffe <command> <args>

commands:
  train           train or finetune a model
  test            score a model
  device_query    show GPU diagnostic information
  time            benchmark model execution time

  Flags from /home/f/caffe-master/tools/caffe.cpp:
    -gpu (Run in GPU mode on given device ID.) type: int32 default: -1
    -iterations (The number of iterations to run.) type: int32 default: 50
    -model (The model definition protocol buffer text file..) type: string
      default: ""
    -snapshot (Optional; the snapshot solver state to resume training.)
      type: string default: ""
    -solver (The solver definition protocol buffer text file.) type: string
      default: ""
    -weights (Optional; the pretrained weights to initialize finetuning. Cannot
      be set simultaneously with snapshot.) type: string default: ""
```

Slika 22: Informacija o uporabi vmesnika za ukazno vrstico

4.5.2 Python

Python [34] je visokonivojski programski jezik za splošno uporabo. Caffe vmesnik za Python omogoča dostop do vseh funkcionalnosti ogrodja in tudi notranjih podatkov.

Nekateri izmed najbolj uporabnih modulov so:

- caffe.Net: glavni vmesnik nalaganja, konfiguriranja in izvajanje modelov
- caffe.Classifier in caffe.Detector: priročna vmesnika za pogoste operacije klasifikacije in detekcije
- caffe.SGDSolver: vmesnik do reševalnika
- caffe.io: vmesnik vhod in izhod podatkov in parametrov
- caffe.draw: vmesnik za vizualizacijo arhitektur nevronskih mrež

Še ena prednost programskega jezika Python je, da se ga lahko uporabi znotraj beležnic Jupyter (Jupyter notebooks) [35]. Caffe ponuja več že pripravljenih beležnic z najbolj uporabnimi funkcijami tu [36].

Poglavje 5 Vgrajeni sistemi

V tem poglavju bom opisal vgrajene sisteme uporabljene v okviru naloge. Predstavil bom tudi nekaj osnovnih pojmov. Osredotočil se bom na praktičnost.

Svetovni trg vgrajenih sistemov je bil v letu 2015 velik kar skoraj 160 milijard ameriških dolarjev in pričakovati je, da bo zrastel za četrtino v naslednjih petih letih [37]. Vodilo te industrije je inovacija in ponujanje stalno boljših rešitev za nižjo ceno.

Razvoj novih arhitektur procesorjev je postal zelo drag in konkurenca za vse nižje cene je potisnila industrijo v proces konsolidacije. Danes najbolj razširjena arhitektura [38] po številu proizvedenih čipov je ARM [39]. ARM je podjetje, ki razvija intelektualno lastnino (IP) mikroprocesorjev in jo licencira proizvajalcem, da okoli nje gradijo svoje čipe za specifične potrebe.

Okoli arhitekture ARM je zrasel obsežen ekosistem orodij, informacij in pomožnih virov. Da sem se odločil za ARM arhitekturo, je vplivala njena razširjenost, nizka cena ter prosta dostopnost tako prevajalnika, kot tudi operacijskega sistema.

ARM je razvil skozi leta veliko različnih verzij svoje arhitekture. Trenutno najbolj razširjena je družina Cortex procesorskih jeder. Družina Cortex je razdeljena na sledeče profile: A (Application) za aplikacijske procesorje z veliko pomnilnika in periferije (na primer za mobilne telefone, tablične računalnike in druge potrošniške elektronike), M (Microcontroller) za mikrokrmilnike in globoko vgrajene sisteme, R (real-time) za sisteme v realnem času v industriji in druge varnostno kritične sisteme.

Za izvedbo te naloge sem se odločil za uporabo procesorjev Cortex-A, saj je za nalaganje nekaterih nevronskih mrež potrebna velika kapaciteta pomnilnika. Tudi zato, ker v tej družini procesorjev je vmesnik za kamero in ostalo potrebno periferijo.

Ker je ponudba računalnikov zelo velika, ustvarjanje računalnika iz nič pa v tem primeru ni smiselno, sem iskal med trenutno razpoložljivimi računalniki v eni sami plošči (single board computers) [40]. Družina Raspberry Pi [41] je najbolj primerna, ker izpolnjuje zahteve po velikosti pomnilnika, ima na voljo priključek za kamero, njegovi procesorji so relativno hitri, je enostavno dosegljiva v Evropi, obstaja že prilagojena verzija Linuxa in je cenovno ugodna.

Poleg tega je na voljo tudi več različic, ki so med sabo relativno kompatibilne in mi omogočajo enostavno razvijanje in primerjanje več prototipov.

5.1 Raspberry Pi

Raspberry Pi je serija majhnih računalnikov na eni sami plošči razvita v fundaciji Raspberry Pi v Združenem Kraljestvu za promocijo učenja osnov računalniških znanosti v šolah in v državah v razvoju [42].

Prvi model se je leta 2012 prodal v večjem številu kot je bilo pričakovano, ker se ga je uporabljalo tudi v druge namene, kot na primer v robotih. Prodanih je že več kot deset milijonov [43].

Na voljo je posebno prilagojena distribucija operacijskega sistema Linux, imenovana Raspbian. Ta distribucija je osnovana na družini distribucije Debian; je v isti družini kot Ubuntu, kar olajša prenos skript med namiznim računalnikom in Raspberry Pi. Raspberry Pi fundacija uradno ponuja Raspbian in je priporočen operacijski sistem.

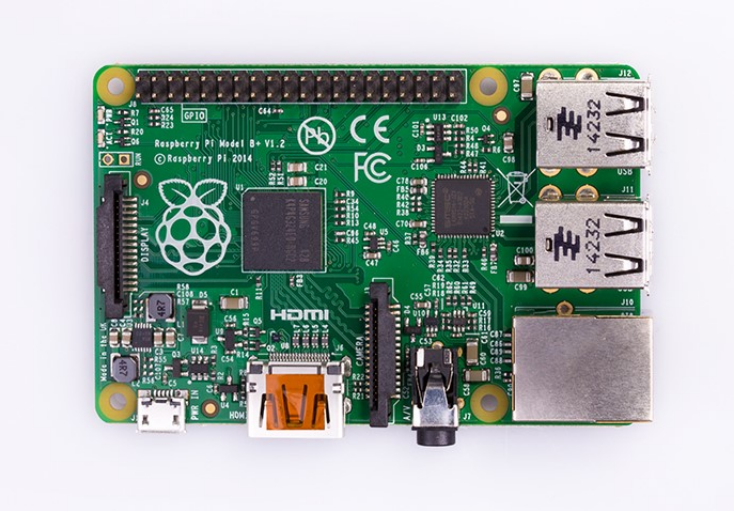
Obstaja več različnih modelov. Sledi opis modelov, uporabljenih v nalogi.

5.1.1 Raspberry Pi Model B+

Ta različica Raspberry Pi računalnika (Slika 23) je izšla Julija 2014. Vsebuje čip BMC2835 firme Broadcom, ki je zgrajen okoli jedra ARM1176JZF-S, ki deluje tovarniško na 700 MHz. Ima 512 MB glavnega pomnilnika. Nazivna moč je 3W.

To je starejše procesorsko jedro, ki je bilo izbrano zaradi nizke cene v prvih verzijah tega računalnika. Ima omejene procesorske zmogljivosti, bo pa zanimiv za primerjavo z novejšimi sistemi.

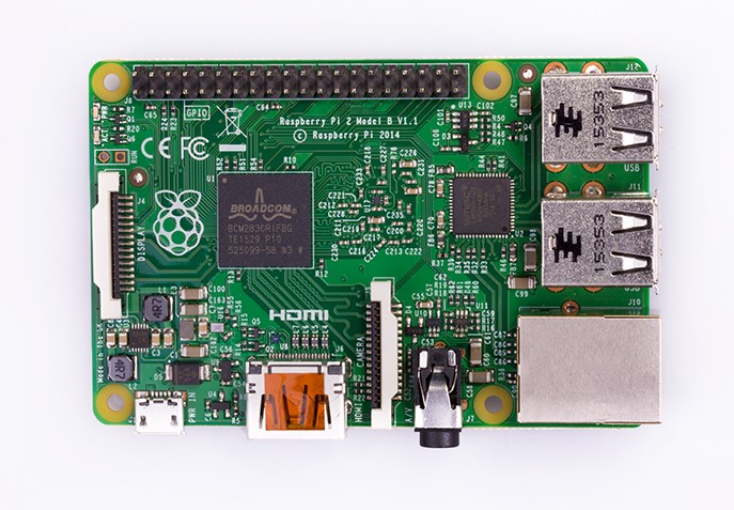
Čip ima tudi grafično procesno enoto VideoCore IV, ki deluje na frekvenci 250 MHz. Teoretična hitrost računanja je 24 GFLOPS, kar je zelo zanimivo za pohitritev matričnih operaciji, potrebnih za izvajanje nevronskih mrež. Velika težava enote je v nedostopnosti dokumentacije (tudi danes uradne povezave do nje ne delujejo), prav tako trenutno še ne obstajajo potrebna orodja za uporabo enote (prevajalnik, razhroščevalnik).



Slika 23: Raspberry Pi Model B+

5.1.2 Raspberry Pi 2 Model B

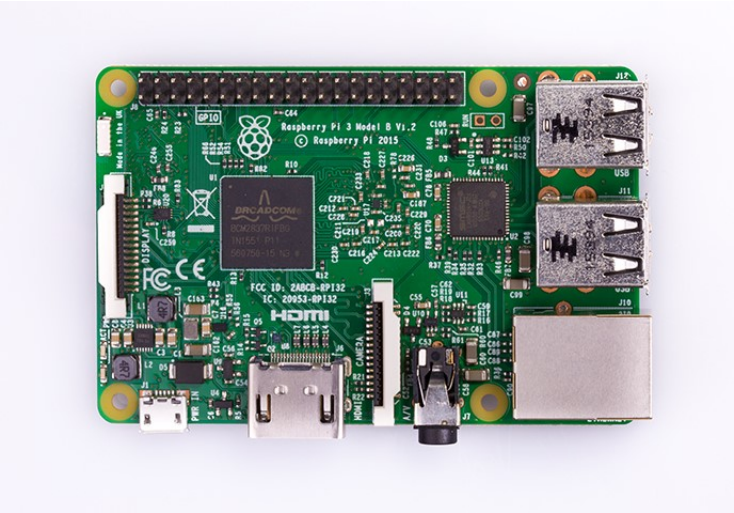
To različico (Slika 24) so zgradili okoli čipa BMC2826 firme Broadcom. Jedro v tem čipu je ARM Cortex A7, ki deluje na frekvenci 900MHz. Povečanje frekvence ni tako veliko, je pa velik preskok v hitrosti procesiranja, ta arhitektura namreč izvaja vsako posamezno instrukcijo hitreje in ima štiri jedra. Na voljo je tudi več pomnilnika, in sicer 1 GB.



Slika 24: Raspberry Pi 2 Model B

5.1.3 Raspberry Pi 3 Model B

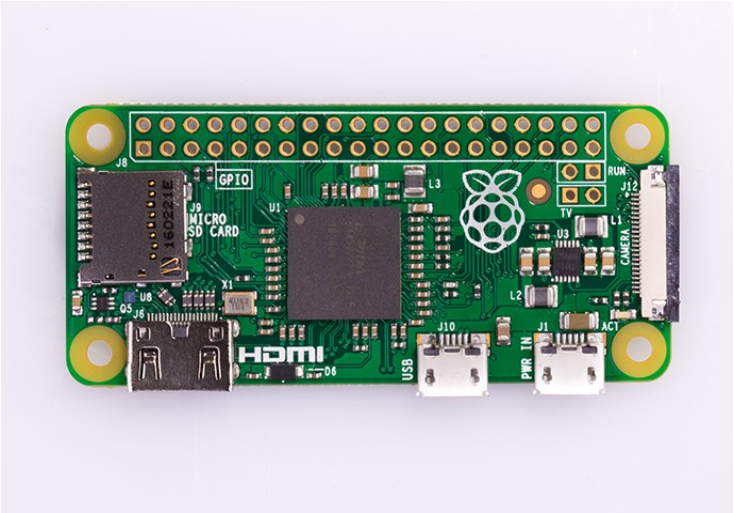
Zadnja različica (Slika 25) je zelo podobna prejšnji. Velikost pomnilnika ostaja enaka in večina perifernih enot ni spremenjenih. Je pa nov čip z novim jedrom ARM Cortex A53, ki ima v sebi štiri 64 bitne procesorje, ki delujejo na 1.2 GHz.



Slika 25: Raspberry Pi 3 Model B

5.1.4 Raspberry Pi Zero

Raspberry Pi zero (Slika 26) je verzija z zelo nizko ceno (predlagana cena je samo pet ameriških dolarjev) in majhno velikost. Uporablja isti čip kot prva različica, ampak s povečano frekvenco in sicer 1GHz. Ima 512 MB pomnilnika.



Slika 26: Raspberry Pi Zero

Poglavje 6 Implementacija

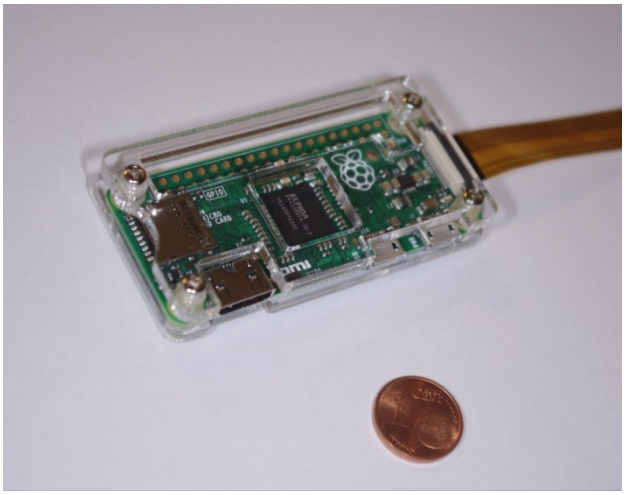
Glavni cilj te naloge je implementacija vgrajenega sistema za klasifikacijo slik. Sistem zajame sliko s kamero, izvaja že naučeno nevronsko mrežo in prikazuje rezultat klasifikacije. Vgrajeni sistem je lahko tudi prenosen, je majhne velikosti, ne potrebuje zunanjih komponent in lahko deluje na bateriji. Učenje nevronske mreže poteka predhodno v strežniku s hitrimi grafičnimi procesnimi enotami.

Moje delo je bilo: raziskava trga in izbira potrebne strojne opreme, dobava le te, sestava vseh komponent, nameščanje sistemske programske opreme, prilagoditev programske opreme za potrebe ogrodja Caffe, nameščanje vseh potrebnih knjižnic in podprogramov, prevajanje ogrodja Caffe in drugih knjižnic in programov v vgrajenem sistemu, pisanje pomožnih skriptnih programov, dobava in prilagoditev nevronskih mrež, generiranja dodatnih učnih podatkov (slik), dodatno učenje nevronskih mrež in sestava vseh delov v delujoči sistem.

6.1 Pridobivanja in sestavljanje strojne opreme

Po raziskavi trga in odločitvi za določeno strojno opremo, je bil moj prvi korak je bil nabava. Kot najbolj enostavno in cenovno ugodno opcijo sem uporabil naročilo preko spleta v eni izmed svetovnih spletnih trgovin (Amazon, eBay), kjer je na voljo širok nabor, med katerim so na voljo tudi vsi prej omenjeni modeli.

Kot že omenjeno, za potrebe primerjave imam na voljo Raspbery Pi 1 Model B+, Raspberry Pi 2 Model B, Raspberry Pi 3 Model B in Raspberry Pi Zero (Slika 27).



Slika 27: Raspberry Pi Zero v ohišju in s priključnim kablom za kamero.

Zaradi enostavnejših nastavitev in kasneje enostavnejše rabe sem za vsak sistem pridobil tudi zaslone (Slika 28).

Slika 28: Raspberry Pi 3 z velikim LCD ekranom v ohišju in kamera druge generacije.

Obstaja zelo velik nabor kamer različnih lastnosti, ki so kompatibilne z Rasberry Pi. V osnovi sta v uradni distribuciji operacijskega sistem podprta dva različna senzorja, mogoča pa je kombinacija kamer s fiksno lečo in fiksno goriščno razdaljo in kamer, ki omogočajo uporabo standardnih leč z navojem M12 (Slika 29).

Slika 29: Tiskano vezje s CCDjem in nastavkom za lečo, ter dve leči.

Da bi bila oprema čim bolj zavarovana in da bi se izognil nevšečnostim zaradi prahu, vlage in pregorele elektronike zaradi statične razelektritev, sem namestil vsak testni sistem v svoje ohišje.

6.2 Pridobivanje in nameščanje programske opreme v vgrajenem sistemu

6.2.1 Operacijski sistem

Operacijski sistem Raspbian je na voljo na uradni strani Raspberry Pi [44]. Po prenosu sem datoteko razširil in prenesel na pomnilniško kartico SD. V delovnem računalniku z operacijskim sistemom Microsoft Windows sem uporabil za pravilen zapis operacijskega sistema Raspbian na kartico pomožni program Win32DiskImager, ki je prosto dostopen tu [45]. Za zapisovanje operacijskega sistema na kartico na operacijskem sistemu Linux ali MacOS pa sem uporabil programe, ko so že dostopni na ukaznih vrsticah le-teh.

Med delom sem opazil, da ker sistem uporabi kartico tudi kot svoj »disk«, je priročno, da je ta čim večja, ker sem potreboval veliko prostora za prevajanje ogrodja Caffe, razne knjižnice, nameščanje dodatnih paketov programske opreme, potrebnih za delovanje, nalaganje več nevronskih mrež in za delovni prostor.

Ko je operacijski sistem v pomnilniški kartici SD, jo namestim na reži v sistemu Raspberry Pi, ga priključim na HDMI ekran, Ethernet, tipkovnico in miško ter nazadnje napajam. Sistem sem napajal s standardnim mikro-USB 5 Voltnim napajalnikom. Med delom sem pa večkrat naletel

na težave, ki so se pojavile naključno. S časom in poskusi sem ugotovil, da je za pravilno delovanje potrebno, da napajalnik daje vsaj dva ampera toka.

Po prvem zagonu sistema operacijski sistem samodejno poveča datotečni sistem, da bo zavzel celotni prostor pomnilniške kartice. Slednje je potrebno, saj sem med presnemavanjem operacijskega sistema na kartico uporabil standardno binarno sliko, ki je prilagojena manjšim karticam.

Po prvem zagonu sistem je potrebno ponovno zagnati, to naredim tako, da enostavno izključim napajalni kabel in ga ponovno priklopim; sistem nima gumbov za ugašanja ali ponovnega zagona. Ko je operacijski sistem odziven in je možno dostopati do ukazne vrstice, je bolje, da se v terminalnem oknu uporabi ukaz:

```
sudo reboot
```

V drugem zagonu sem konfiguriral sistem z ukazom:

```
sudo raspi-config
```

Uporabil sem naslednje nastavitve:

- Spremenim privzeto geslo »raspberry«. Privzeti uporabnik ima ime »pi«, kar bom potreboval pri naslednjem zagonu.
- Spremenim ime sistema (hostname) in sicer tako, da ne bo v konfliktu z drugimi uporabniki omrežja. Posebej pomembno, ker primerjam več sistemov.
- Spremenim nastavitev, da bo sistem po zagonu ostal v tekstovnem načinu. Grafični vmesnik je sicer res optimiziran za nizko porabo resursov, vseeno pa predstavlja neko obremenitev sistema, ki je ne potrebujem.
- Zaradi varnosti onemogočim samodejno prijavljanje (auto login).
- Omogočim vmesnik za kamero.
- Omogočim ssh. To mi da možnosti upravljanja sistema na daljavo, ne da bi ga priklopili na ekran in tipkovnico.
- Posodobim tudi časovni pas, na katerem se nahajamo.

Shranim vrednosti, zapustim program in ponovno zaženem sistem.

V naslednjem zagonu se mi pokaže tekstovni vmesnik. Prijavim se z uporabniškim geslom, ki sem ga prej spremenil.

Zdaj je priporočljivo, da se sistem posodobi z najnovejšo programsko opremo. V Raspbianu, kot v vsaki distribuciji družine Debian, to naredim tako, da v ukazni vrstici izvedem najprej:

```
sudo apt update
```

potem pa še:

```
sudo apt upgrade
```

To lahko traja dalj časa, lahko tudi ure. Ko je posodobitev končana, ponovno zaženem sistem.

Zdaj namestim še potrebno programsko opremo in morebitne gonilnike in module jedra (kernel modules) za uporabo posebnih ekranov, ker te niso na voljo v privzetem jedru. Navodila za nameščanja ekranov so zelo različna od ekrana do ekrana. Ko je programska oprema nameščena, sistem ugasnem in izklopim iz napajanja in iz druge periferije in sledim potrebnim varnostnim ukrepom za preprečevanje statične razelektritve, preden fizično namestim ekran v sistem. Po ponovnem zagonu preverim, da vse deluje pravilno.

V naslednjem koraku namestim še kamero. Navodila za priključitev in preverjanja delovanja kamere so tu [46].

Namestim še pomožno programsko opremo, ki mi olajša delo v tem majhnem sistemu. Iz izkušenj priporočam nameščanja in uporaba vmesnika midnight commander:

```
sudo apt-get install mc
```

6.2.2 Dodatni moduli

6.2.2.1 Knjižnica za linearno algebro

Privzeta knjižnica za linearno algebro, ki se uporablja, ko prevedemo ogrodje Caffe je ATLAS [47] [48]. Ta knjižnica se uporablja, ker je prosto dostopna, pogosto zelo hitra in hitro na voljo v novih arhitekturah.

Zamenjal sem privzeto knjižnico za linearno algebro s knjižnico OpenBLAS, ker mi ta omogoča multi-procesorsko računanje matričnih operaciji. V novejših Raspberry Pi sistemih z več procesorji to prinese znatne pohitritve.

Navodila za prenos in prevajanje zadnje verzije knjižnice OpenBLAS v sistemu Rasperry Pi so tu [49]. Da lahko uporabim to prednost sem nastavil kot spremenljivko okolja (environment variable):

```
NUMBER_OF_CORES=4
```

6.2.2.2 OpenCV

OpenCV je knjižnica za računalniški vid, napisana v programskem jeziku C++ z vmesnikom tudi za jezik Python. Je zelo priročna za obdelavo slik, pripravo vhodnih podatkov in drugih pomožnih nalog.

Navodila za prenos in prevajanje zadnje verzije knjižnice OpenCV v sistemu Rasperry Pi so tu [49]. To knjižnico sem prenesel in prevedel v vgrajenem sistemu. Namestim tudi vmesnik za jezik Python.

6.2.2.3 Python

Uradna verzija operacijskega sistema Raspbian že vsebuje programski paket Python, s katerim lahko delam. Potrebno je le še dodati nekaj paketov. To naredim tako, da izvedem sledeči ukaz:

```
sudo pip install lmdb numpy scipy scikit-image pydot scikit-learn
```

Zraven že nameščenega Pythona tudi uporabim posebno distribucijo imenovano Anaconda. Ima več prednosti, ena izmed bolj poglavitnih pa je bolj konsistentno upravljanje paketov različnih verziji. Navodila za nameščanje v sistemu Raspberry Pi so tu [50].

6.3 Prevajanja ogrodja Caffe za Raspberry Pi

Da lahko uporabim Caffe v sistemu Raspberry Pi, ga moram prevesti za ta sistem. To sem naredil najprej s prečnim prevajanjem (cross compiling), ampak s tem je bilo več težav zaradi nekompatibilnih knjižnic in podprogramov. Bolj enostavno je bilo prevesti kar neposredno v Raspberry Pi. Operacija traja več ur, se je pa izkazala za zanesljivejšo od prečnega prevajanja.

Najprej sem namestil orodje GIT, in sicer z ukazom:

```
sudo apt-get install git
```

nato pa v delovnem imeniku pridobim izvorno kodo ogrodja Caffe. To naredim z ukazom:

```
git clone https://github.com/benjibc/caffe-rpi
```

Pred samim prevajanjem je potrebno še namestiti nekaj pomožnih programskih paketov. To naredim z ukazi:

```
sudo apt-get update
sudo apt-get install -y libprotobuf-dev libleveldb-dev
sudo apt-get install –y libsnappy-dev libopencv-dev
sudo apt-get install -y libhdf5-serial-dev libboost-all-dev
sudo apt-get install -y libatlas-base-dev
sudo apt-get install -y python-dev python-pip
sudo apt-get install -y libgflags-dev libgoogle-glog-dev
sudo apt-get install -y liblmdb-dev protobuf-compiler
sudo apt-get install -y cmake unzip doxygen graphviz
sudo apt-get install -y libffi-dev python-dev build-essential
sudo apt-get install -y gfortran
sudo apt-get install -y python-numpy python-scipy python-nose
sudo apt-get install -y libopenblas-dev
```

Specifični seznam potrebnih knjižnic in pomožnih programov se lahko spremeni. V poskuzih sem opazil, da ta seznam ukazov deluje pravilno v vseh verzijah operacijskega sistema Raspbian. Ta korak traja dalj časa, odvisno je od hitrosti povezave v medmrežje in hitrosti SD pomnilniške kartice. Zame je to bilo približno eno uro.

Zdaj pa imam vse na voljo za prevajanje ogrodja Caffe. Ukazi za prevajanje so:

```
mkdir build
cd build
cmake ..
cd ..
make all
make pycaffe
make test
make runtest
make distribute
cd ..
```

Preden nadaljujem, sem še preveril, da je bilo izvajanje vseh testov uspešno. To potrjuje ne samo pravilno prevedeno ogrodje Caffe ampak tudi pravilno delovanje vgrajenega sistema. Imel sem nekajkrat težave s testi, če se je sistem segreval, ali če napajalnik ni imel dovolj moči.

6.4 Učenje

Za učenje nevronskih mrež je potrebno imeti model mreže, model reševalnika ter učne in testne podatke.

Že narejene arhitekture nevronskih mrež so na voljo v Caffe Model Zoo [51]. Tam sem našel veliko število nevronskih mrež različnih arhitektur. Za vsako nevronsko mrežo so na voljo: tekstovna datoteka z modelom mreže za učenje, datoteka z modelom reševalnika, in po navadi tudi datoteka z modelom mreže za uporabo ter binarna datoteka s parametri že naučene nevronske mreže.

6.4.1 Pridobivanje učnih podatkov

Da lahko sam naučim nevronske mreže ali da prilagodim že naučene nevronske mreže mojim potrebam, sem uporabil več metod za pridobivanje učnih podatkov.

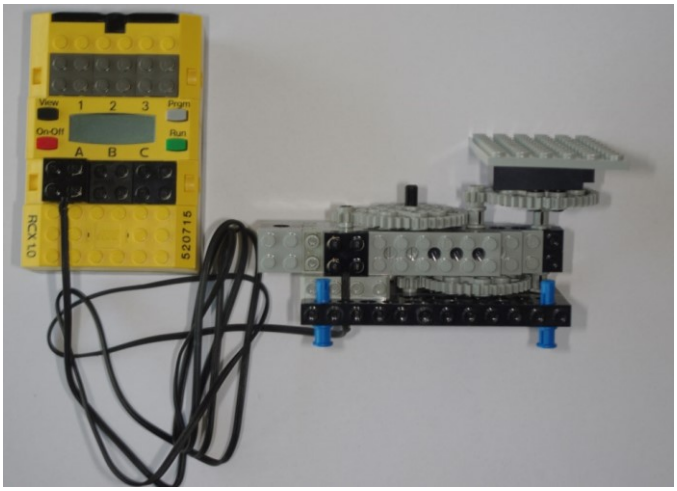
6.4.1.1 Javno dostopne baze

ImageNet [52] je velika baza slik, namenjena raziskovanju algoritmov za prepoznavanje objektov. Vsebuje več kot deset milijonov ročno klasificiranih slik. Letno ImageNet organizira tekmovanje ILSVRC, v katerem so se uspešno pokazale nekatere od nevronskih mrež, opisane v tej nalogi. Če je kategorija, ki me zanima, na voljo v ImageNet, sem prenesel slike iz te baze.

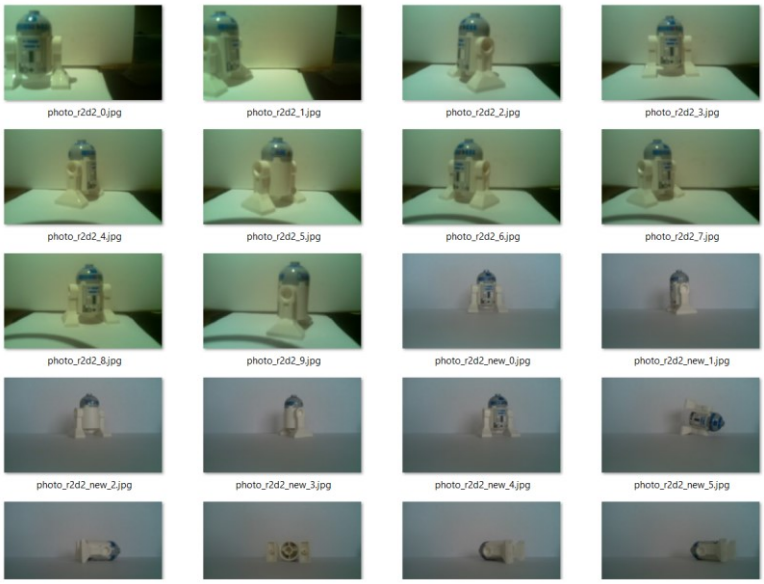
6.4.1.2 Zajemanje slik s kamero

Za pridobivanje učnih podatkov sem nekatere potrebne slike ročno naredil in izbral. Ker je potrebno število slik za učenje vsakega razreda zelo veliko, je to zelo zamudna naloga. Pogosto je potrebno vsaj tisoč različnih slik istega razreda, ki se jih potem še dodatno poveča z že opisanimi metodami, da se dobi uporabno učno množico. Koliko časa bi zato potrebovali, lahko hitro izračunamo, če vemo koliko časa potrebujemo za eno sliko.

Uvedel sem delno avtomatizacijo in je tako naloga postala bolj obvladljiva. Za potrebe te naloge, da bi naredil veliko število slik majhnih objektov, sem naredil z igralnimi kockami Lego rotirajočo platformo (Slika 30), kamor sem postavil objekt in ga potem lahko zelo hitro slikal večkrat iz različnih zornih kotov. S pomočjo te naprave in nekaj programskih skript napisanih v Pythonu sem lahko naredil 1000 slik enega objekta (Slika 31) v približno pol ure. Še vedno zelo potratno, ampak obvladljivo za majhne eksperimente.



Slika 30: Slika rotirajoče platforme za delno avtomatizacijo slikanja



Slika 31: Primer nekaterih slik, ki sem jih zajel z delno avtomatizacijo

6.4.1.3 Avtomatski prenos iz internetnih iskalnikov slik

En izmed načinov, ki sem jih uporabil za hitro pridobivanje velike količine učnih slik enega razreda, je, da jih poiščem z enim od internetnih iskalnikov slik ter jih potem prenesem. To sem naredil tako, da sem zato napisal skripto v Pythonu (Slika 32).

Slika 32: Primer zbirke slik, ki sem jih pridobil skriptno iz internetnega iskalnika slik

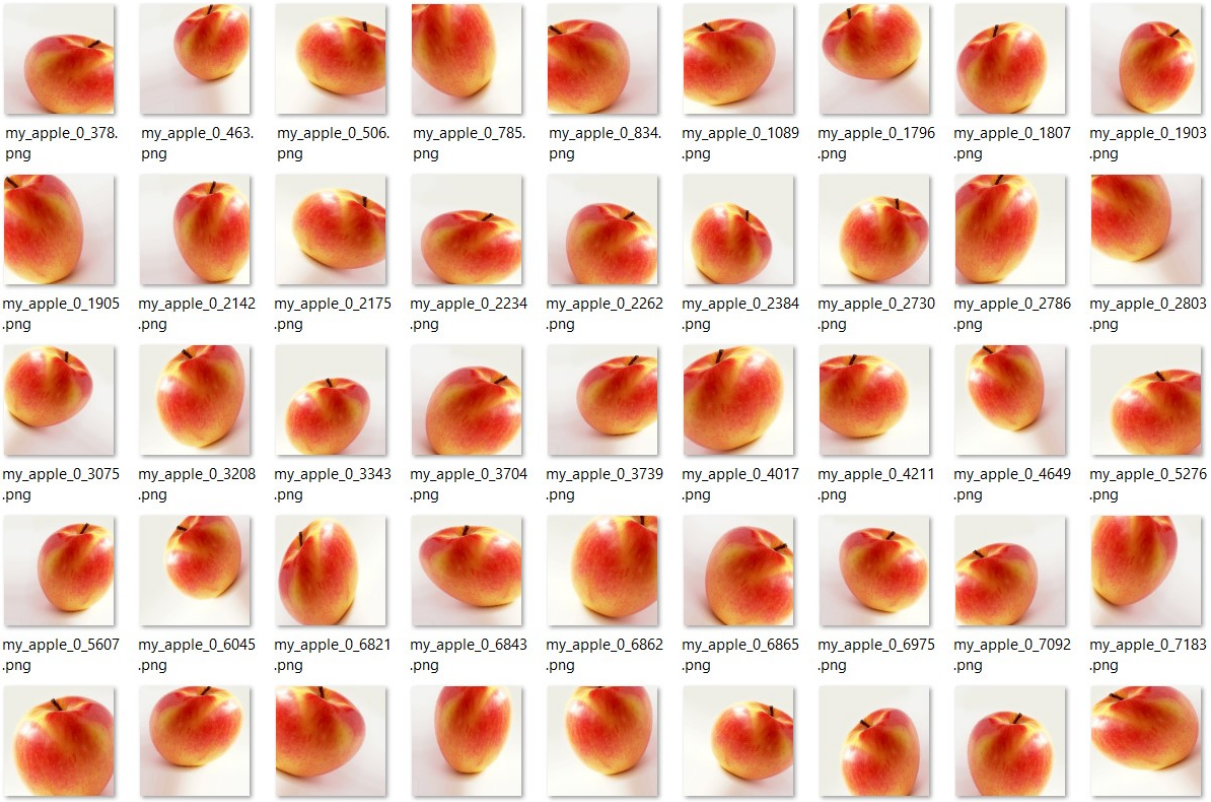
Kolikor sem lahko zasledil, še ni jasno, če je to zakonsko dovoljeno. Obstaja več pogledov glede uporabe intelektualne lastnine kot učni material za nevronske mreže. Nekatere zanimive diskusije na to temo so v [53] in [54].

Moramo tudi vedeti, da so slike, ki jih internetni iskalniki danes najdejo, klasificirane ne samo na podlagi teksta okoli njih, ampak predvsem z uporabo nevronskih mrež za klasifikacijo [55].

6.4.1.4 Umetno povečanje učnih podatkov

Učne podatke sem moral umetno povečati. Za današnje nevronske mreže z več milijoni parametrov sem potreboval več milijonov slik za učenje, da ne bi imel preveliko prilagoditev mreže učnim podatkom. Ker nisem imel toliko slik, je umetno povečanje nabora neizogibno.

Število učnih slik sem povečal z uporabo že omenjenih trikov, in sicer tako, da sem za vsako sliko iz učne množice generiral veliko dodatnih slik (Slika 33) z uporabo skripte napisane v Pytonu [56]. Knjižnica OpenCV ima vse potrebne funkcije za enostavno branje slik več formatov, spremembo le teh in hranjenja nazaj. Pomanjkljivost te metode je, da sem potreboval zelo veliko prostora za vse te slike. To je hitro postalo neobvladljivo.



Slika 33: Primer povečanja nabora učnih slik, ki sem jih pridobil skriptno iz ene same slike

Boljši način povečave števila slike sem dosegel z uporabo posebnega sloja za ogrodje Caffe, ki te dodatne učne slike generira med učenjem samim in jih ni potrebno shranjevati. Obstaja več že tako razvitih slojev. Sam sem uporabil caffe-augmentation, na voljo tu [57].

6.4.2 Učenje nevronskih mrež v oblaku

Učenje globoke nevronske mreže je računsko zelo zahtevna naloga. V mojih poskusih sem za učenje nekaterih nevronskih mrež potreboval več tednov učenja v zelo naprednih grafičnih procesnih enotah. Cena te strojne opreme je lahko zelo visoka: cena nVidia Tesla K80 je okoli 9.500 EUR (Januar 2017), cena nVidia Tesla je okoli 7.500 EUR (Januar 2017). Če nimamo stalne potrebe po takšni moči, je veliko bolj ugodno, če to opremo najamemo pri enem od internetnih ponudnikov strežnikov v oblaku.

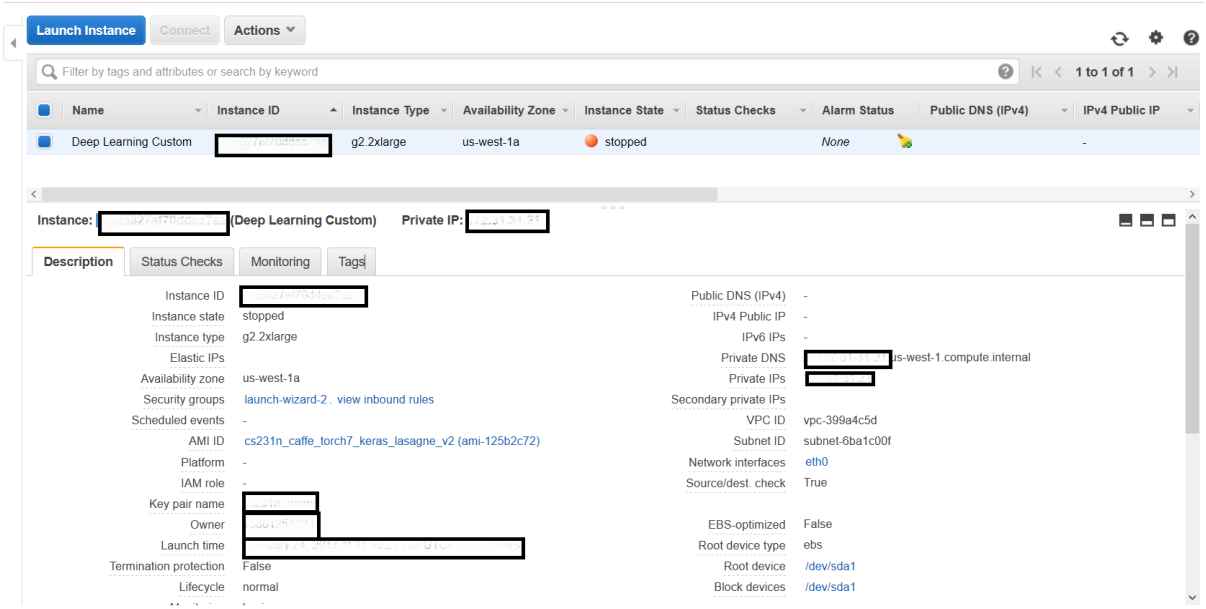
Sam sem za učenje nevronskih mrež v oblaku uporabil storitev AWS (Amazon Web Services). AWS ponuja različne storitve, razpršene na več geografskih lokacijah po celem svetu. Cenovno je relativno ugodno. Na primer, za najem strežnika g2.2xlarge (8 jeder, 15 GB pomnilnika, 60 GB SSD, 1x GPU s 1,536 CUDA jeder) je cena na uro 0,702 USD (Januar 2017). Za

enotedenski najem takšnega strežnika, ki zadostuje za učenje ene ali dveh globokih nevronskih mrež, bi plačali približno sto evrov. Cena takega strežnika je več tisoč evrov.

Za primerjavo, ocenjujem na podlagi nekaj meritev manjših primerov, za dosego podobnih rezultatov kot, jih dobim v strežniku v enem tednu s prenosnim računalnikom s štiri jedrskim i7 procesorjem, a brez splošne grafične procesne enote, bi potreboval približno tri mesece.

Za zagon nove instance najetega strežnika sem naredil uporabniški račun za uporabo AWS storitve. Za uporabo dražjih storitev, kot je najem strežnikov z grafičnimi procesnimi enotami, Amazon zahteva dodatno preverjanje identitete in potrditev, to traja nekaj dni. Pri konfiguraciji strežnika sem lahko izbral, kateri operacijski sistem bom uporabil, na katerega sem sam namestil vso ostalo programsko opremo. Možnost sem imel tudi vzeti eno od že obstoječih predhodno naloženih konfiguraciji. Naredil sem konfiguracijo, ki je zelo podobna mojemu lokalnemu strežniku, da lahko najprej naredim poskuse lokalno in ko je vse pripravljeno prenesem vse potrebno na strežnik, tako ne plačam uporabe strežnika med razvojem, ampak šele takrat, ko je vse pripravljeno.

Še eno konfiguracijo sem naredil na podlagi cs231n_caffe_torch7_keras_lasagne_v2 (Slika 34), ki ima že nameščenih več programskih paketov za strojno učenje, prevajalnike, knjižnice za obdelavo slik in druge pripomočke. V mojih poskusih je ta konfiguracija dosegla hitrejše delovanje v nekaterih primerih. Odvisno od količino potrebne prilagoditev mojih lastnih skriptnih programov za učenje sem uporabil eno ali drugo konfiguracijo.



Slika 34: EC2 Management Console, nekateri podatki so prekriti zaradi varnosti

Pri uporabi nove instance strežnika se ta dinamično dodeli enemu izmed fizičnih strežnikov, prenese binarno sliko operacijskega sistema, konfigurira dostop do interneta in prične z izvajanjem. Par minut po zagonu je strežnik pripravljen za uporabo. Za povezavo sem uporabil terminalni program s podporo protokola ssh. Nekaj navodil in priporočil za delo z AWS strežniki za potrebe učenja nevronskih mrež sem našel v [58].

Za učenje nevronske mreže sledimo enostavnim navodilom, ki so na voljo v [59].

6.4.3 Dodatno učenje na osnovi že naučenih nevronskih mrež

Učenje nove nevronske mreže lahko znatno pohitrimo z uporabo že naučene nevronske mreže. Konvolucijski filtri se najbolj počasi učijo v prvih slojih nevronske mreže, zelo pogosto pa lahko uporabimo uteži za spodnje sloje iste mreže naučene z drugo učno množico slik, če področja uporabe niso zelo drugačna. Na primer, če imamo nevronsko mrežo, ki zna prepoznati različne vrste sadja, lahko zelo verjetno uspešno uporabimo začetne sloje iste mreže za prepoznavanje zelenjave.

Caffe omogoča tudi, da vzamemo uteži enega sloja že naučene nevronske mreže v drugi nevronski mreži, če sta oba sloja enaka, mreži sta pa lahko različni [60].

Pri dodatnem učenju se lahko tudi odločimo, da želimo obdržati uteži določenih slojev med učenjem, to je, da se ti sloji ne spremenijo. Razlog za to je, da vemo, da je ta sloj že optimalen, dodatno učenje pa bi ta sloj preveč prilagodilo učnim podatkom. Navodila so v [61].

6.5 Primeri

Sledi nekaj poskusnih primerov, opravljenih na predstavljeni strojni in programski opremi, z uporabo tehnik in metod opisanih v nalogi.

6.5.1 Sistem za klasifikacijo sadja in zelenjave

V nadaljevanju predstavljam primer uporabe prototipa, ki zna prepoznavati različne vrste sadja in zelenjave. Odločil sem se za to kot prvo nalogo, ker sama narava sadja in zelenjave kot objekt klasifikacije dobro prikazuje prednosti tega pristopa: je vedno drugačno, po velikosti, barvi in obliki. Tudi, če je sistem odlično prilagojen nekemu naboru sadja en dan, je to sadje že naslednji dan ali vsaj v nekaj dneh drugačno, ker dozori ali gnije, torej sistem mora biti zmožen klasificirati vedno nove primerke. Sistem, ki je pretirano prilagojen neki množici, ne bo v deloval realnem svetu.

Za ta primer sem uporabil implementacijo mreže AlexNet, ki je že predhodno naučena in na voljo tu [62]. Vzel sem to arhitekturo nevronske mreže, ker se je dobro izkazala v ImageNet tekmovanju za klasifikacijo množice, ki je vsebovala nekaj podobnih razredov, zato lahko tudi predvidevam, da lahko uporabim tehniko dodatnega učenja. Spremenil sem zadnji sloj te nevronske mreže, ki je bila originalno naučena za 1000 razredov in jo prilagodil za samo 10. V mojem primero to so: paradižnik, hruška, jabolka, pomaranča, čebula, limona, grozdje, korenje, solata in banana.

Učno množico sem sestavil tako, da sem zajel za vsak razred 500 slik. Teh sem skriptno vzel iz video posnetkov, ki sem jih naredil z vgrajeno napravo. Takšen pristop ima več prednosti. Je hitro, ker iz posnetka, dolgega samo par minut lahko vzamem nekaj sto slik. Slike uporabljene za učenje in slike iz klasifikacije so zajeti z isto opremo. Sama narava zajemanja je ista kot med uporabo (osvetlitev, zorni kot, morebitni tresljaji ciljnega objekta ali naprave, itd).

Iz internetnega iskalnika slik sem pridobil dodatne slike za vsak razred. To sem naredil, da bi preprečil pretiranega prilagajanja nevronske mreže na ožji tip slik, ki sem jih sam naredil. Iskanje in prenos slik sem naredil skriptno, kjer sem za vsak razred prenesel 3000 slik in potem ročno izbral 500 najbolj primernih. To je nujno potrebno delo. En dober primer je razred »jabolko«, ker iskanja slik tega razreda pogosto daje tudi kot rezultat slike, ki niso sadje jabolko, ampak v povezavi z računalniško firmo Apple. Tudi v tem iskanju je pogosto videti slike moških teles z odvečno težo, to je z obliko jabolk, ker je bilo v času, ko sem te slike iskal, medijsko odmevnih nekaj novic o povečanju tveganja za srčno-žilnih bolezni pri moških s tako postavo.

V celoti sem imel 1000 slik za vsak razred ter 10 razredov, torej 10.000 slik. Teh sem razdelil s skripto naključno v učno in v testno množico v razmerju 70% / 30%.

Učil sem nevronsko mrežo v oblaku v AWS strežniku tipa G2 s K520 GPUjem.

Uporabil sem dodatno učenje (transfer learning), to je, sem vzel za začetno stanje že prej naučeno nevronsko mrežo na ILSVRC2012 vzorcu, s tem da sem zamenjal zadnji sloj mreže iz originalnega s 1000 razredi v sloj z 10 razredi. Poleg tega sem nadaljnji dve plasti polno povezanih povezav, ki služita kot klasifikator za začetne konvolucijske sloje postavil v naključno začetno stanje, ker sta že bili prilagojeni drugačnim razredom in bi to lahko preprečilo, da bi se nevronska mreža učila pravilno, ker bi lahko obtičala v napačnem lokalnem minimum.

Za parametre učenja sem vzel sledeče vrednosti:

- base_lr: 0.0001 – začetna hitrost učenja
- lr_policy "step" - koračno zmanjšanje hitrosti učenja, s formulo
  lr_policy = lr_policy * gamma
- gamma: 0.1
- stepsize: 2000 – število obdobij med spremembo koraka učenja
- test_iter: 500 – število ponovitev testov v vsakem testnem intervalu
- test_interval: 500 – število obdobij med testiranjem mreže
- display: 20 – kako pogosto (število obdobij) prikazati vmesne rezultate
- max_iter: 200000 – maksimalno število obdobij v enem izvajanju učenja
- momentum: 0.9 – faktor ohranjanje trenutne vrednosti uteži v povezavah, v novem obdobju učenja
- weight_decay: 0.0001 – faktor zmanjšanja velikih vrednosti uteži; deluje tudi kot normalizacija
- snapshot: 1000 – število obdobij med shranjenjem vmesnih rezultatov.
- average_loss: 40

Zgornje nastavitve parametrov sem izbral na podlagi primera podanega v ogrodju Caffe. Le te sem dodatno prilagodil z dolgotrajnim poskušanjem.

Dodatno učenje je teklo samo 7 ur. Delni rezultati in zgodovina učenja izpisana v terminalu so pokazali, da je nevronska mreža s točnostjo nad 80% bila na voljo že v treh urah.

Za primerjavo sem tudi izvajal učenje iz naključnega začetnega stanja, torej iz »prazne« nevronske mreže. Učenje v tem primeru je trajalo 36 ur, a je po 21 urah bil na voljo dober rezultat, katerega točnost je bila nad 80%. Njegova točnost se ni več znatno izboljšala.

Končna dosežena točnost sistema je bila v obeh primerov podobna in sicer približno 84%.

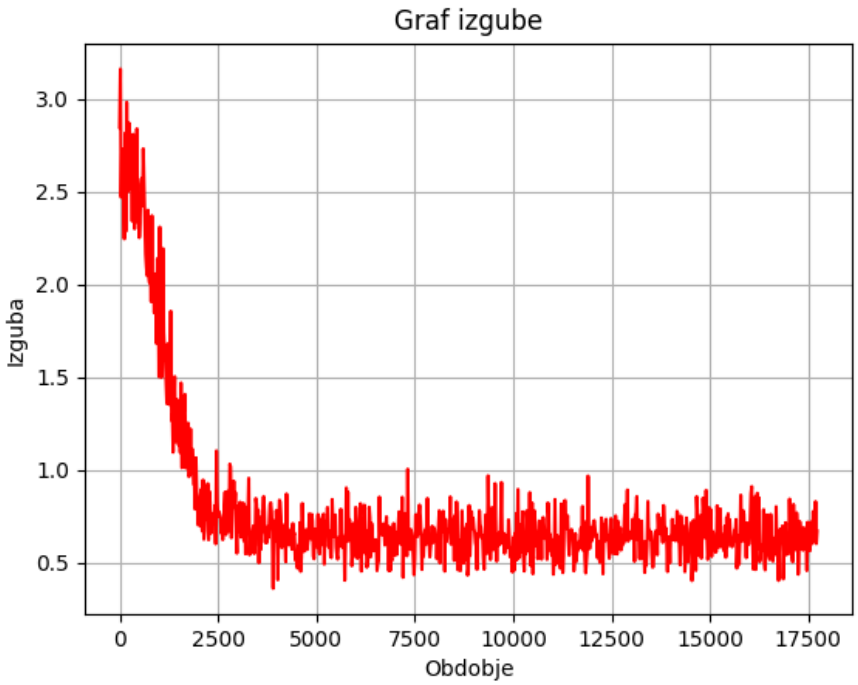
Učenje je veliko hitrejše od drugih primerov uporabe te arhitekture, ki sem jih videl omenjene v raznih internetnih forumih, zaradi zmanjšanega števila razredov in zato tudi manjšega števila povezav in hitrejše konvergence.

Zadnji sloj nevronske mreže je tipa »softmax«, to je normalizirana eksponenta funkcija, ki tako daje za vsak izhodni nevron oceno verjetnost pripadajočega razreda.

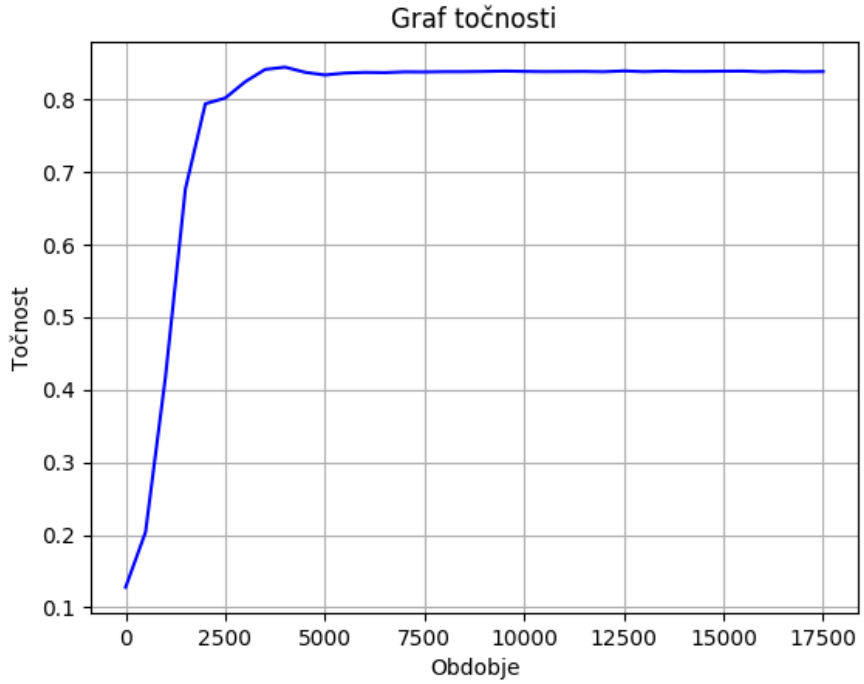
Med učenjem se izguba izračuna kot križna entropija (cross entropy) dobljene vrednosti v zadnjem sloju in pričakovani izhod za učni primer.

Ocena točnosti mreže med testiranjem je enostavno vrednost klasifikacije dobljena s klasifikacijo vseh primerov iz testne množice.

V grafih je na vodoravni osi časovna enota obdobje (angleško epoch). V enem obdobju gre učenje skozi vse slike učne množice enkrat. V tem primeru je to 1.000.000 slik: 10 razredov krat 1000 slik krat 100 kratno umetno povečanje.

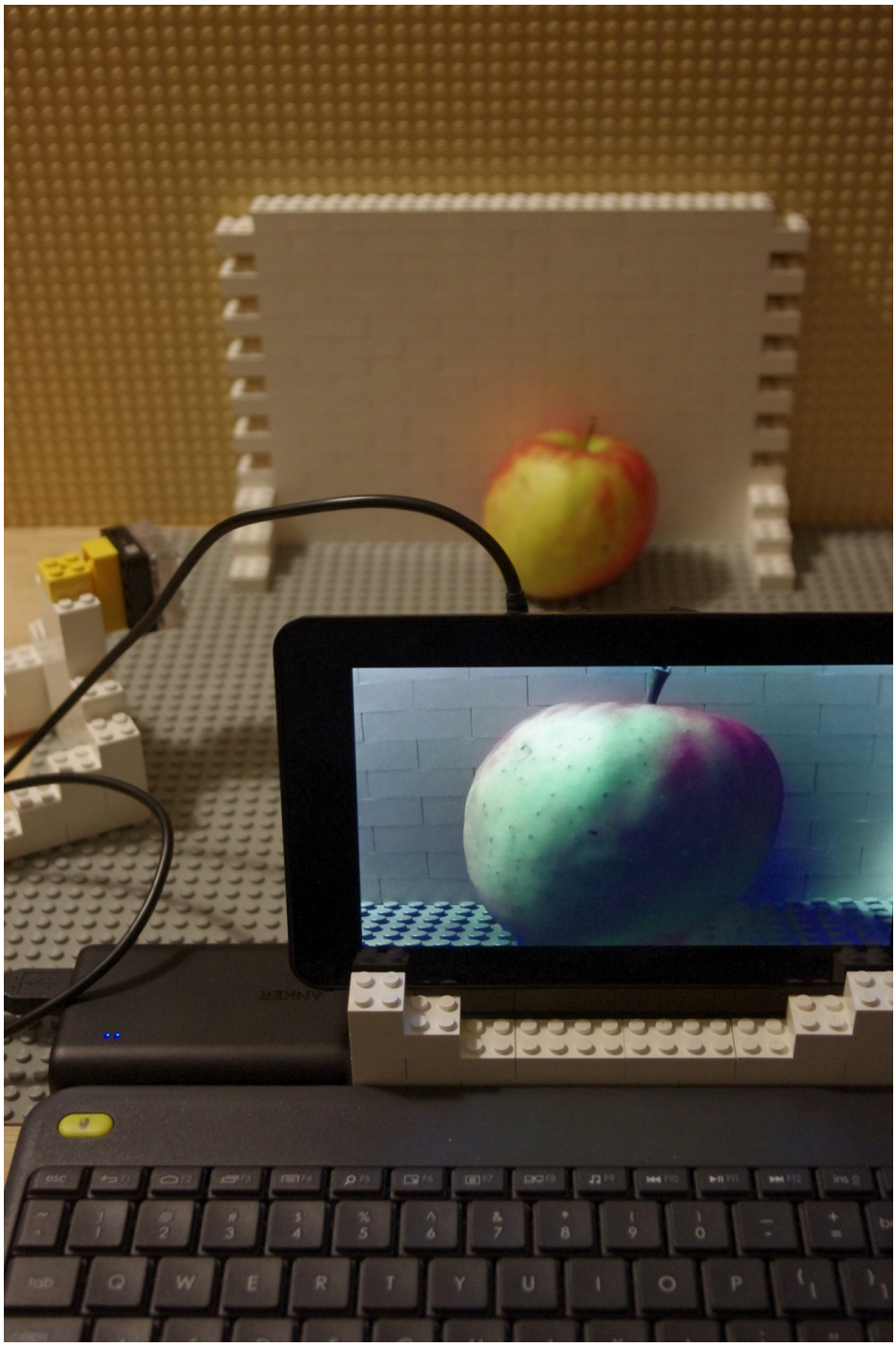


Slika 35: Graf izgube pri dodatnem učenju kaže na hitro konvergenco



Slika 36: Graf točnosti pri dodatnem učenju prikazuje hitro konvergenco in visoko točnostjo

Tako naučeno nevronsko mrežo sem uporabil v najhitrejšem vgrajenem sistemu in v testnem okolju. Ta mreža se je med testno uporabo dobro izkazala in klasificirala objekte pravilno skoraj vedno (Slika 37).



Slika 37: Testni vgrajeni sistem, med zajemanjem slike sadja in tekočo klasifikacijo

6.5.2 Sistem za klasifikacijo Lego figuric

V tem primeru sem učil nevronsko mrežo GoogLeNet za prepoznavanje LEGO figuric (Slika 38: Slika LEGO figuric uporabljenih). Ta primer je zanimiv za primerjavo, ker te figurice se,

za razliko od sadja in zelenjave, ne spreminjajo s časom. Figurica bo isto zgledala skozi mesece. Čeprav z leti lahko barve zbledijo in plastika porumeni, to ne bo vplivalo na moje delo.

Slika 38: Slika LEGO figuric uporabljenih za klasifikacijo

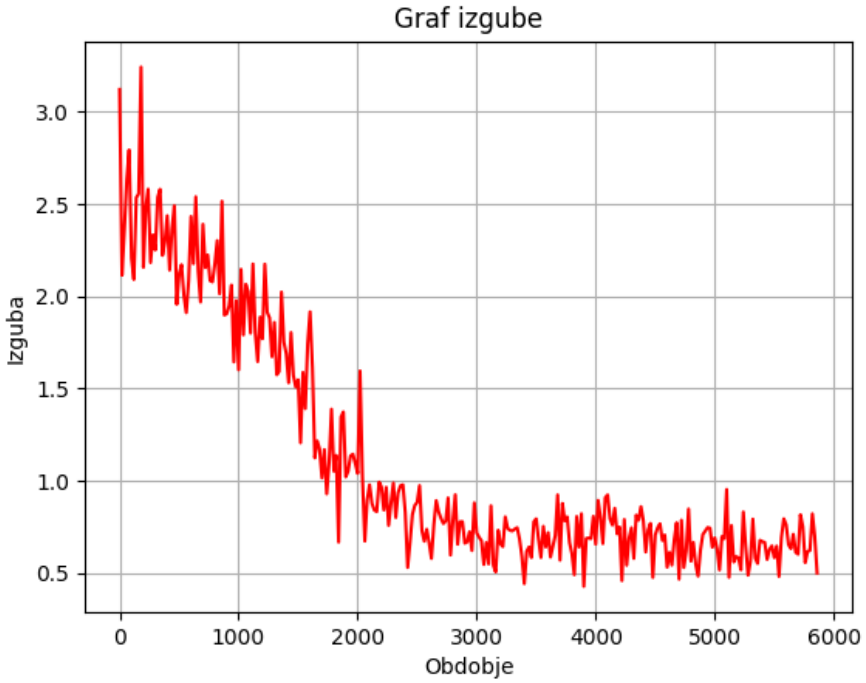
Učno množico slik sem naredil sam. Za približno polovico vseh slik sem uporabil avtomatizacijo (vrtljive platforme in avtomatsko slikanje). Za drugo polovico slik sem naredil video posnetek dolg nekaj minut in skriptno vzel iz njega sličice.

Učil sem za sedem različnih razredov in imel 10.000 slik za vsak razred. To je 70.000 slik v celoti, kar zadostuje za dobro učenje nevronske mreže, ki je lahko uspešna v klasifikaciji objektov pod realnimi pogoji.

Kljub temu, da sem imel velik nabor učnih slik, sem posebej prevedel ogrodje Caffe z dodatnim slojem za povečanje učnih podatkov. Uporabil sem kar privete vrednosti, ki se dobro prilagodijo tipu objektov in slikam, s katerim delam v tem primeru. Za faktor povečave učnih podatkov sem vzel 100, kar potem predstavlja 700.000 slik v vsakem obdobju učenja.

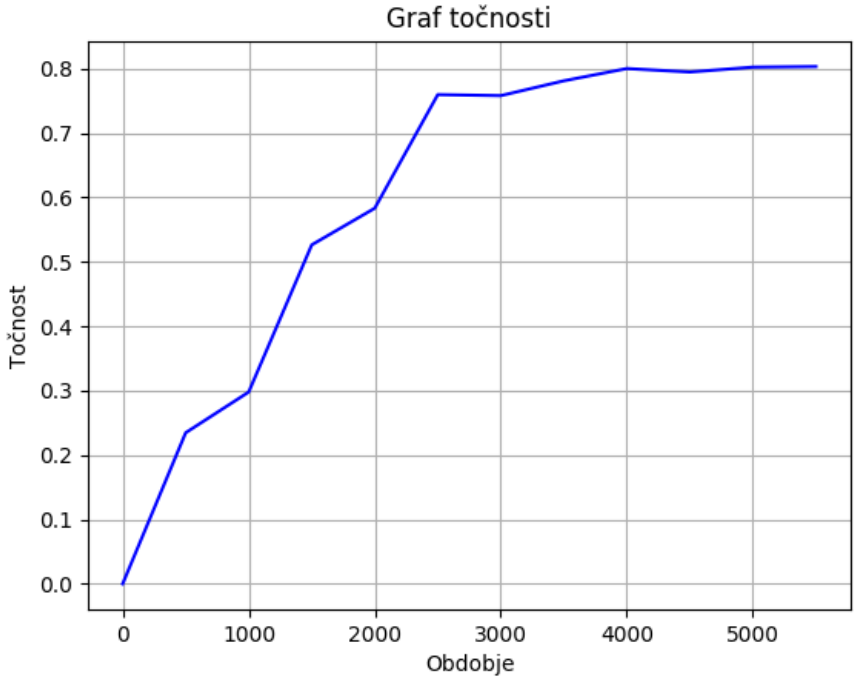
Učenje je potekalo na AWS EC2 strežniku tipa G2.2xlarge z nVidia K520 GPUjem.

Mrežo sem učil iz naključnega začetnega stanja, saj nisem imel na voljo primernih uteži, da bi izvajal dodatno učenje. Učenje sem omejil na 200.000 obdobji, vendar sem ga ustavil po 6.000 obdobjih, ali približno 70 urah, ker sem med spremljanjem dnevniške datoteke učenja opazil, da točnost in izguba nista več vidno napredovala in se je točnost ustalila pri vrednosti nad 80%.



Slika 39: Graf izgube pri učenju nevronske mreže za prepoznavanje LEGO figuric

Pridobljena mreža dosega visoko točnost klasifikacije, vendar ima težave razločevati figurice, ki so med sabo podobne ali pa figurice, ki niso v celoti na sliki. Verjetno je do tega prišlo, ker so učne slike med sabo zelo podobne. Za boljše obnašanja v teh robnih primerih bi moral razširiti učno množico slik s takimi, ki pokrivajo take situacije.



Slika 40: Graf točnosti pri učenju nevronske mreže za klasifikacijo LEGO figuric

Zanimivo pa je, da sistem zna pravilno klasificirati figurice tudi, ko so te postavljene v položajih (upognjeni, drugačno postavitev rok in nog, drugače obrnjena glava), ki niso bili predvideni med zajemanjem slik za učno množico.

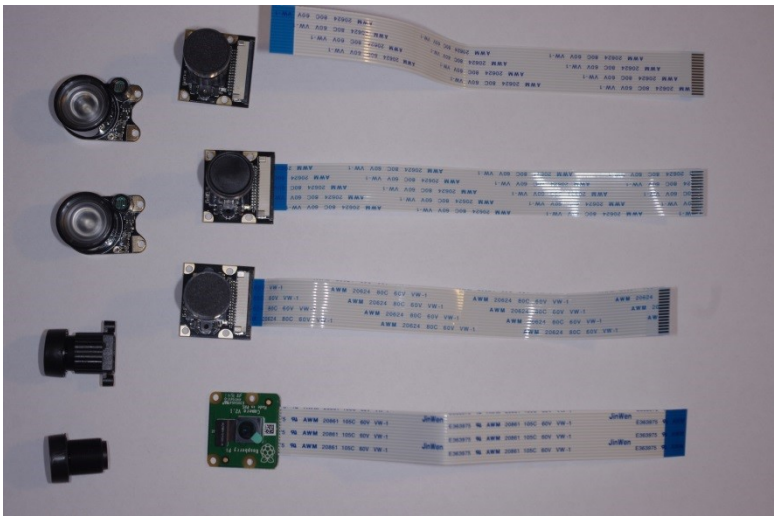
Tudi ko je osvetlitev drugačna od te, ki sem jo uporabil med slikanjem učne množice, se sistem dobro obnaša. Sklepam, da je to zato, ker taki so pogoji pokriti s variacijam, ki jih je vneslo v učenje umetno povečanje izvornih slik. To naredi sistem bolj odporen na šum in druge nepredvidene situacije.

6.5.3 Eksperimentalni sistem za diagnosticiranja kožnega raka

V tem primeru je moj cilj prikazati prve korake za izdelavo prototipa naprave, ki bo znala interaktivno klasificirati kožna znamenja kot benigna ali maligna. Zato sem prilagodil strojno opremo in naučil primerno nevronsko mrežo.

Učne podatke za to nalogo je bilo težje iskati, ni veliko javno dostopnih baz. Po nekaj iskanja sem dobil majhno množico označenih slik [63] pri projektu ADDI (Automatic computer-based Diagnostic system for Dermatoscopy Images). Ta nabor je premajhen za učinkovito učenje, ampak vseeno sem poskusil z vsemi pripomočki, ki jim imam na voljo (povečanje učnih podatkov, dodatno učenje).

Druga težava je bilo pridobivanje primerne kamere in osvetlitev za napravo. Nabor učnih slik je zajet z medicinskim dermatoskopom. Poizkusil sem več različnih kamer, leč in dodatnih osvetlitev, nekaj teh je vidnih na Slika 41, pri tem pa sem našel kombinacijo, ki daje za veliko nižjo ceno podobne rezultate (kamera in leča staneta manj kot 50 EUR): RPi Cam2 z nastavkom za leče z navojem M12, tele leča M12 in bele LED svetilke, ki jih lahko pritrdim z vijaki na modul kamere in napajam neposredno iz RPi.



Slika 41: Nekaj kamer, leč in osvetlitev

Nekaj ročnega dela mi je vzela zgradba nastavka za kamero. S pomočjo nekaj prozornih plastičnih kozarcev, škarij in lepilnega traku sem imel zadovoljiv prototip.

V učni množici sem imel na voljo samo 200 slik, 160 benignih tvorb in 40 malignih. Ta nabor slik ne vsebuje slik kože brez tvorb. Predvidevam, da bi bilo koristno, da naprava zna ločiti kožo s tvorbo in brez kot dodatno pomoč uporabniku, ki lahko preverja, da naprava tvorbo identificira pravilno.

Da bi povečal nabor učnih podatkov, sem naredil še nekaj slik lastne kože. Najprej sem opravil pregled pri zdravniku, ki je potrdil, da ne opazi na meni nobene maligne tvorbe. Nato sem naredil 600 slik različnih predelov, 400 slik kože in 200 slik znamenj. Med slikanjem sem opazil, da je prisotnost las in dlak ali ne v sliki lahko faktor, ki komplicira pravilno klasifikacijo znamenj, zato sem poskusil narediti nabor slik, ki bo najbolj raznolik glede tega: stopala in dlani brez dlak, lasišče največ, in roke in noge kjer je vmesna situacija.

To mi je dalo nekaj več realnih slih, ampak z veliko omejitvijo: to je samo ena oseba, z eno barvo kože, eno barvo in tipom las. Sam mislim, da lahko drugi faktorji vplivajo na točnost klasifikacije, kot so spol in starost, ker je zato lahko koža drugačna. Za realni sistem bi bilo

potrebno pridobiti slike več različnih oseb. Sem razmišljal o tem, da bi prosil druge, da bi zajel slike pri njih, ampak obstaja nevarnost, da naletim na maligna znamenja, ki jih sam ne znam identificirati in jih dam v napačni razred učnih podatkov. Zato sem se odločil, da se za potrebe tega poizkusa omejim na ta majhen nabor, ki ga imam do sedaj. To je vseeno zanimiv eksperiment, ker je izredno malo podatkov. S tem lahko poizkusim, kako daleč pridem s tem v praksi.

Za dodatno povečavo nabora slik, sem uporabil skriptni program, ki sem ga napisal v pythonu, in uporabil pomožno knjižnico keras. Ta knjižnica ima zelo fleksibilen generator naključnih slik na podlagi ene slike. Pri generaciji dodatnih slik sem moral razmišljati, katere transformacije naj uporabim, ker se pri nekaterih transformacijah razred ne ohrani. Na primer, če vzamem sliko benigne tvorbe in močno spremenim kontrast in mogoče tudi barve, lahko dobim sliko, ki je bolj podobna maligni tvorbi. Prebral sem nekaj splošnih navodil za laike, kako identificirati kožni rak pri pregledu samega sebe, da sem lahko pristopil bolj informirano.

Za generacijo dodatnih slik, sem se odločil, da uporabim sledeče transformacije:

- Rotacija (rotation): sliko lahko poljubno rotiram, ker orientacija tvorbe ne vpliva na njeno klasifikacijo. Tudi naprava lahko gleda na znamenja iz vsakega zornega kota, odvisno od tega, kje je in kako ga je bolj priročno slikati.

- Premiki v širino in višino (width shift, height shift): naprava ne bo vedno zajela znamenja v sredini slike. Zato je smiselno, da se jo uči s slikami, ki so tudi zamaknjene iz idealne sredine.

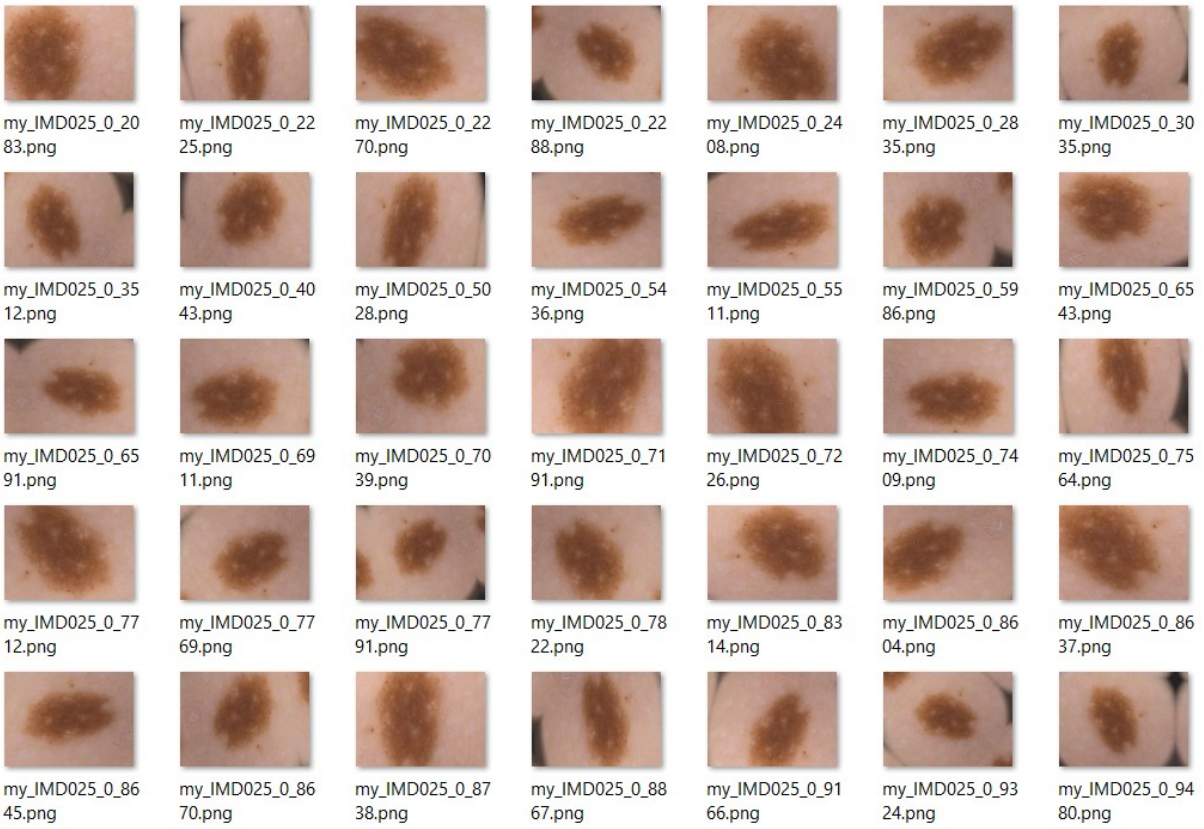
- Povečava (zoom): to ni enostaven kriterij, ker je velikost tvorbe lahko znak njene malignosti. Zato sem se odločil, da bom povečavo uporabil, ampak samo med -50% in +10%.

- Deformacija (shear): tudi tu je treba biti previden, ker je oblika znamenja pomemben dejavnik, zato sem vzel za to majhno vrednost, do 0.1, torej 10%.

- Zrcaljenje: to ne vpliva na klasifikacijo transformirane slike, zato je dovoljeno v celotnem naboru transformaciji.

Na podlagi teh transformaciji in iz začetnih 800 slik sem generiral približno sto krat več slik (Slika 42). Torej 40.000 slik kože brez znamenj, 36.000 slik benignih tvorb ter 4.000 slik malignih tvorb.

Te slike sem generiral in shranil, ker sem jih hotel pregledati, preden jih uporabim za učenje. Nad tem naborom slik bom še uporabil transformacije med samim učenjem, ampak z zelo majhnimi vrednosti parametrov: malo šuma, rotacije, majhni premiki, zelo majhno dodatno deformacijo.



Slika 42: Primer generiranih slik iz ene originalne slike benigne tvorbe

Odločil sem se, da bo klasifikacija v tri razredi: koža brez tvorb, benigna tvorba, maligna tvorba.

Za nevronsko mrežo sem vzel SqueezeNet, verzija 1.1. Ta nevronska mreža doseže dobro točnost klasifikacije slik v raznih domenah. Je tudi majhna in hitra, dobro za napravo, ki mora delovati interaktivno.

Ocenil sem, da noben primer prosto dostopne in že naučene mreže SqueezeNet, ki sem jo uspel najti, ni v podobni domeni uporabe. Zato sem se odločil, da učenje naredim iz »prazne« nevronske mreže, torej z naključnim začetnim stanjem.

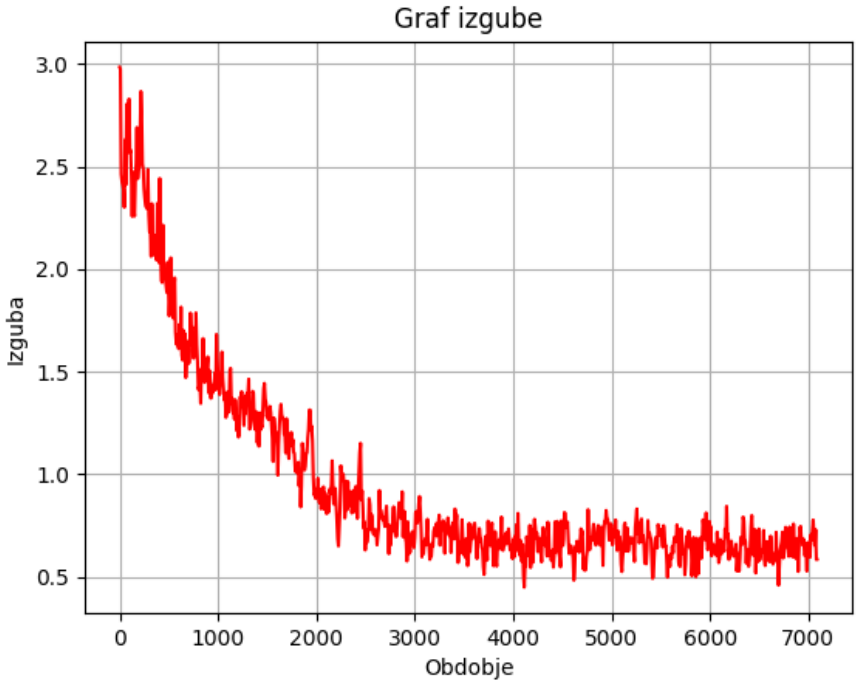
Podatke sem razdelil tako, da jih je 2/3 v učni množici in 1/3 v testni množici. Razdelitev sem naredil naključno s skripto. To mi omogoča, da celotno učenje nevronske mreže lahko enostavno ponovim z novo razdelitvijo, da primerjam točnost več mrež in preverjam, da ni

rezultat učenja zelo odvisno od začetne množice. Ocenim, da je lahko to težava, ker imam tako malo začetnih podatkov.

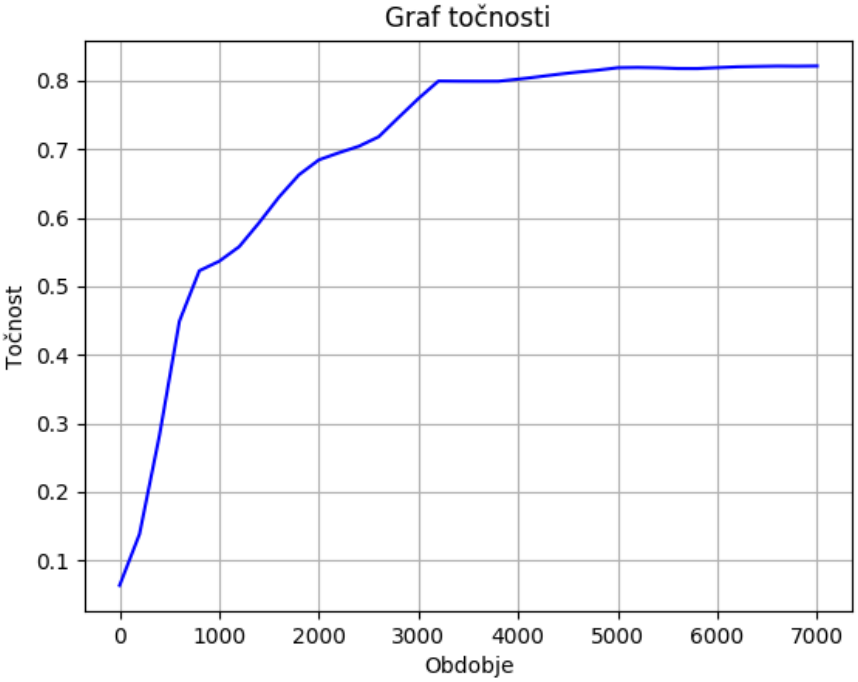
Za parametre učenja sem vzel sledeče vrednosti:

- lr_policy "poly": polinomsko padanje hitrosti učenja s formulo $\left(1 - \frac{iter}{maxiter}\right)^{power}$
- base_lr 0.01: začetna hitrost učenja
- power: 1.0
- max_iter 200000: število iteraciji učenja, predvideval sem, da bo mreža že prej konvergirala, ampak iz komentarjev razvijalcev v raznih internetnih forumih sem lahko sklepal, da bo učenje z veliko verjetnost konvergiralo v tem času.
- iter_size: 16, test_iter: 2000, test_interval: 1000:
- display 100: učenje izpisuje podatke na 100 iteraciji
- momentum: 0.9, weight_decay: 0.0002, snapshot: 1000, snapshot_prefix: "train",
- solver_mode: GPU,
- average_loss: 40

Učil sem nevronsko mrežo v AWS EC2 strežniku z nVidia K520 GPUjem. Učenje sem ustavil po približno 50 urah, ker sem videl iz dnevniške datoteke, da je učenje že konvergiralo.



Slika 43: Graf izgube pri učenju nevronske mreže za prepoznavanje kožnega raka



Slika 44: Graf točnosti pri učenju nevronske mreže za prepoznavanje kožnega raka

Učenje konvergira pri približno 3000 iteraciji, kjer opazim, da se točnost napovedi nevronske mreže ustali. Iz izkušenj sklepam, da učenje po tej iteracije bi samo dodatno prilagajalo nevronsko mrežo učni množici.

Čeprav sistem identificira kožna znamenja precej točno, se pri tem ne morem opredeliti o njeni točnost za identifikacijo malignih tvorb brez testiranja s pomočjo eksperta. Je bil pa ta primer poučen in lahko služi kot podlaga za nadaljnjo delo.

6.5.3.1 Sorodni sistemi za diagnosticiranja kožnega raka

Objavljenih je bilo že mnogo člankov na to temo, kot na primer [64], [65], [66], [67]. Najbolj odmevni je članek s naslovom »Klasifikacija kožnega raka s točnostjo dermatologa z uporabo globokih nevronskih mrež« v originalu »Dermatologist-level classification of skin cancer with deep neural networks« v priznani reviji Nature [64]. V tem članku je opisan sistem klasifikacije kožnega raka na osnovi globokih nevronskih mrež, učen na množici skoraj 130.000 slik. Sistem so primerjali s točnostjo, ki jo doseže skupina 21 certificiranih dermatologov, ki je bila testirana za primerjavo. Avtorji članka so videli, da sistem deluje s točnostjo ekspertov. Omenjeno je tudi, da ta rezultat kaže na možnost široke uporabe takega sistema prilagojenega na mobilne naprave (pametni telefoni). To bi omogočalo zgodnjo diagnostiko kožnega raka zelo širokemu delu človeštva.

Poglavje 7 Primerjava nevronskih mrež in vgrajenih sistemov

V tem poglavju sem opravil primerjavo nevronskih mrež že opisanih in uporabljenih v primerih.

7.1 Čas izvajanja (klasifikacija ene slike)

Tu podajam empirično določene čase med živim zajemanjem slike. Klasifikacija vzame slike v realnem času iz video toka kamere (camera video stream). Čas ene klasifikacije se lahko spreminja odvisno od obremenitev celotnega sistema, neodvisno od same klasifikacije. Zato sem kot čas ene klasifikacije vzel povprečje vseh meritev v drugi minuti tekočega delovanja, to je med 60 in 120 sekund po zagonu.Za vsako nevronsko mrežo je tudi podano število operaciji, ki jih potrebuje za eno klasifikacijo. Enota je MFLOP (mega- FLoating point OPerations), to je, koliko milijonov operaciji v plavajoči vejici je potrebnih za eno klasifikacijo.

	SqueezeNet v1.1 360 MFLOP	AlexNet 727 MFLOP	GoogLeNet 2000 MFLOP
Raspberry Pi 1	4,11 s	5,64 s	11,42 s
Raspberry Pi 2	0,68 s	1,01 s	2,24 s
Raspberry Pi 3	0,43 s	0,77 s	1,83 s
Raspberry Pi Zero	2,98 s	4,14 s	9,17 s

7.2 Velikost nevronske mreže

Vse nevronske mreže tu podane imajo za parametre številke v plavajoči vejici, velikosti 32 bitov.

	SqueezeNet v1.1	AlexNet	GoogLeNet
Vsi sistemi	4,72 MB	232 MB	51 MB

7.3 Točnost klasifikacije

Točnost klasifikacije se standardno izmeri na testni množici slik, to je, na znanem vzorcu.

7.4 Točnost klasifikacije v ILSVRC2012 vzorcu

ILSVRC2012 je kratica za »ImageNet Large Scale Visual Recognition Challenge 2012«. To je tekmovanje za programske sisteme za klasifikacijo in segmentacijo slik, ki se izvaja letno. 2012 je bilo leto ko je zmagala ekipa, ki je uporabljala globoke nevronske mreže v svoji rešitvi. Zato je postal nabor podatkov iz te izdaje tega tekmovanja neuradni standard za primerjavo drugih nevronskih mrež.

Pri top-1 napaki se šteje odgovor mreže kot pravilen, če je pravilni razred najvišje ocenjen. Pri top-5 napaki pa, če je pravilni razred med petimi najvišje ocenjenimi razredi.

	SqueezeNet v1.1 (1000 razredov) [15]	AlexNet (1000 razredov) [68]	GoogLeNet (1000 razredov) [68]
Točnost (top-1)	58%	57%	69%
Točnost (top-5)	80%	80%	89%

7.5 Točnost klasifikacije v primerih

V primerih opisanih v tej nalogi je število razredov majhno, zato zmatram, da ni smiselno uporabljati top-5 kriterija točnosti. Podam samo podatek za top-1.

	SqueezeNet v1.1 Diagnostika kožnega raka (3 razredi)	AlexNet Klasifikacijo sadja in zelenjave (10 razredov)	GoogLeNet Klasifikacijo LEGO figuric (7 razredov)
Točnost (top-1)	86%	84%	82%

Poglavje 8 Sklepne ugotovitve

V sklopu diplomske naloge sem razvil tri različne sisteme za klasifikacijo slik z uporabo globokih nevronskih mrež.

Celoten sistem se lahko prilagodi za nova področja uporabe hitro in brez večjih stroškov. Vsi trije primeri so med sabo podobni. Ključna razlika med njimi je, kako sem učil nevronsko mrežo za vsako področje uporabe.

Učenje nevronske mreže je enostavno razumeti, ampak sama implementacija je lahko bolj zapletena od pričakovanj. Izbira primernih parametrov učenja je ključnega pomena in še ne obstaja preprosti način, kako zagotoviti njihovo optimalnost. Narobe izbrani parametri učenja lahko vodijo k procesu učenja, ki ne bo konvergiralo, ali ki bo obtičalo v lokalnem minimum, daleč od smiselne rešitve. Poskušanje z različnimi vrednostnimi teh parametrov pa je zelo zamudno, ker učenje ene nevronske mreže traja lahko več dni.

Ustvarjanje ali nabiranja primerne množice učnih in testnih slik zahteva tudi veliko dela in časa, ker je število potrebnih slik zelo veliko in končno obnašanje nevronske mreže je odvisno od njih. Nevronske mreže se lahko učijo delati stvari, ki so bile samo nekaj let nazaj težko predstavljive, ampak zato tudi velja nepisano pravilo »smeti na vhodu, smeti na izhodu« (angleško »trash in, trash out«), z drugimi besedami, nevronska mreža bo tako dobra kot podatki uporabljeni v učenju.

Hitrost klasifikacije je dobra za interaktivno uporabo. Je pa videti velika razlika med različnimi nevronskimi mrežami. Raziskovanje in razvoj sta se v zadnjih letih usmerjala v iskanje arhitektur nevronskih mrež, ki dajejo podobne rezultate, kot že uveljavljene, ampak z manj računanja. To se še naprej nadaljuje in je vsak teden videti v literaturi nove zanimive ideje.

Točnost klasifikacije je zelo visoka v vseh treh obravnavanih primerih in je primerljiva s točnostjo doseženo v literaturi za uporabljene arhitekture nevronskih mrež. Točnost je pravzaprav nekoliko višja, ker je število razredov v naših primerih majhno.

Poraba energije je dovolj nizka, da se sistemi lahko napajajo z baterijami in tako delujejo dalj časa. So tudi relativno majhni in lahki.

Zaradi vseh teh lastnosti je tako implementirani sistem zanimiv za široko uporabo. Primeri, predstavljeni v nalogi, lahko služijo kot vodilo za implementacijo drugih podobnih sistemov na drugih področjih.

Programska oprema, ki je bila uporabljena v nalogi, sloni na operacijskem sistemu za splošno rabo in na drugih splošnih modulih programske opreme, ki uporabljajo več pomnilnika in časa, kot je nujno potrebno. Kot nadaljnjo delo bi bilo to programsko opremo možno prilagoditi na operacijski sistem v realnem času za vgrajene sisteme in optimizirati nekaj modulov. To bi omogočalo delovanje na še cenejših sistemih, spekter uporabnosti pa bi bil tako še večji.

Poglavje 9 Literatura

[1] N. Pinto, D. D. Cox in J. J. DiCarlo, „Why is Real-World Visual Object Recognition Hard?,“ 25 Jan 2008. [Elektronski]. Available: https://dx.doi.org/10.1371%2Fjournal.pcbi.0040027. [Poskus dostopa 15 Jan 2017].

[2] R. C. Johnson, „Microsoft, Google Beat Humans at Image Recognition,“ Februar 2015. [Elektronski]. Available: http://www.eetimes.com/document.asp?doc_id=1325712.

[3] B. Farley in W. Clark, „Simulation of Self-Organizing Systems by Digital Computer,“ *IRE Transactions on Information Theory,* p. 76–84, 1954.

[4] N. Rochester, J. Holland, L. Habit in W. Duda, „Tests on a cell assembly theory of the action of the brain, using a large digital computer,“ *IRE Transactions on Information Theory,* p. 80–93, 1956.

[5] F. Rosenblatt, „The Perceptron: A Probabilistic Model for Information Storage and Organization In the Brain,“ *Psychological Review,* p. 386–408, 1958.

[6] M. Minsky in S. Papert, Perceptrons: An Introduction to Computational Geometry, MIT Press, 1969.

[7] P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.,* Cambridge, Massachusetts: Harvard University, 1975.

[8] Intel, „Intel's First Microprocessor,“ [Elektronski]. Available: http://www.intel.co.uk/content/www/uk/en/history/museum-story-of-intel-4004.html. [Poskus dostopa Januar 2017].

[9] K. Shirriff, „Ken Shirriff's blog,“ [Elektronski]. Available: http://www.righto.com/2015/05/the-texas-instruments-tmx-1795-first.html. [Poskus dostopa Januar 2017].

[10] Statista, „The Statistics Portal,“ [Elektronski]. Available: https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide/. [Poskus dostopa Januar 2017].

[11] J. Turley, „embedded.com,“ 11 August 2013. [Elektronski]. Available: http://www.embedded.com/electronics-blogs/significant-bits/4024611/Motoring-with-microprocessors. [Poskus dostopa Januar 2017].

[12] Semiconductor Industry Association, „Industry Statistics,“ [Elektronski]. Available: http://www.semiconductors.org/industry_statistics/industry_statistics/. [Poskus dostopa Januar 2017].

[13] K. He, X. Zhang, S. Ren in J. Sun, „Deep Residual Learning for Image Recognition,“ 2015.

[14] Z. A. Simonyan K., „Very Deep Convolutional Networks for Large-Scale Image Recognition,“ 2014.

[15] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally in K. Keutzer, „SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size,“ 2016.

[16] M. Nielsen, „Neural Networks and Deep Learning,“ Januar 2017. [Elektronski]. Available: http://neuralnetworksanddeeplearning.com/chap2.html. [Poskus dostopa Januar 2017].

[17] I. Goodfellow, Y. Bengio in A. Courville, Deep Learning, MIT Press, 2016.

[18] M. Mazur, „A Step by Step Backpropagation Example,“ 17 Marz 2015. [Elektronski]. Available: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/. [Poskus dostopa Januar 2017].

[19] Wikipedia, [Elektronski]. Available: https://en.wikipedia.org/wiki/Backpropagation. [Poskus dostopa Januar 2017].

[20] Y. LeCun, Y. Bengio in G. Hinton, „Deep learning,“ *Nature,* p. 436–444, 2016.

[21] S. Hochreiter, *Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis.,* Munich: Institut f. Informatik, Technische Univ. Munich, 1991.

[22] M. D. Zeiler in R. Fergus, „Visualizing and Understanding Convolutional Neural Networks,“ 2013.

[23] A. Krizhevsky, I. Sutskever in G. E. Hinton, „ImageNet Classification with Deep Convolutional Neural Networks,“ v *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, 2012.

[24] V. Vanhoucke, A. Senior in M. Mao, „Improving the speed of neural networks on CPUs,“ v *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop. Vol. 1*, 2011.

[25] V. Nair in G. E. Hinton, „Rectified linear units improve restricted boltzmann machines.,“ v *Proc. 27th International Conference on Machine Learning*, 2010.

[26] R. Belland in Y. Koren, „Lessons from the netflix prize challenge,“ *ACM SIGKDD Explorations Newsletter,* p. 75–79, 2007.

[27] A. Berg, J. Deng in L. Fei-Fei, „Large scale visual recognition challenge 2010,“ 2010. [Elektronski]. Available: www.imagenet.org/challenges. [Poskus dostopa Januar 2017].

[28] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever in R. Salakhutdinov, „Improving neural networks by preventing co-adaptation of feature detectors,“ arXiv preprint arXiv:1207.0580, 2012.

[29] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke in A. Rabinovich, „Going Deeper with Convolutions,“ *IEEE Xplore,* 2015.

[30] Jia, Y. a. Shelhamer, E. a. Donahue, J. a. Karayev, S. a. Long, J. a. Girshick, R. a. Guadarrama, S. a. Darrell in Trevor, „Caffe: Convolutional Architecture for Fast Feature Embedding,“ arXiv preprint arXiv:1408.5093, 2014.

[31] Open Source Initiative, „The 2-Clause BSD License,“ [Elektronski]. Available: https://opensource.org/licenses/BSD-2-Clause. [Poskus dostopa Januar 2017].

[32] „Caffe Solver,“ [Elektronski]. Available: http://caffe.berkeleyvision.org/tutorial/solver.html. [Poskus dostopa Januar 2017].

[33] „Caffe Layers,“ [Elektronski]. Available: http://caffe.berkeleyvision.org/tutorial/layers.html. [Poskus dostopa Januar 2017].

[34] Python Software Foundation, „Python,“ [Elektronski]. Available: https://www.python.org/. [Poskus dostopa Januar 2017].

[35] Project Jupyter, „Jupyter,“ [Elektronski]. Available: http://jupyter.org/. [Poskus dostopa Januar 2017].

[36] „Classification: Instant Recognition with Caffe,“ [Elektronski]. Available: http://nbviewer.jupyter.org/github/BVLC/caffe/blob/master/examples/00-classification.ipynb. [Poskus dostopa Januar 2017].

[37] Market Research Store, „Global Embedded Systems Market Set for Rapid Growth, to Reach around USD 225.34 billion by 2021,“ [Elektronski]. Available: http://www.marketresearchstore.com/news/global-embedded-systems-market-249. [Poskus dostopa Januar 2017].

[38] K. M. Balanza, „ARM from zero to billions in 25 short years,“ 11 September 2013. [Elektronski]. Available: https://community.arm.com/iot/b/blog/posts/arm-from-zero-to-billions-in-25-short-years. [Poskus dostopa Januar 2017].

[39] Wikipedia, „ARM architecture,“ [Elektronski]. Available: https://en.wikipedia.org/wiki/ARM_architecture. [Poskus dostopa Januar 2017].

[40] Wikipedia, „Single Board Computer,“ [Elektronski]. Available: https://en.wikipedia.org/wiki/Single-board_computer. [Poskus dostopa Januar 2017].

[41] Raspberry Pi Foundation, „Raspberry Pi,“ [Elektronski]. Available: https://www.raspberrypi.org/. [Poskus dostopa Januar 2017].

[42] Wikipedia, „Raspberry Pi,“ [Elektronski]. Available: https://en.wikipedia.org/wiki/Raspberry_Pi#Pi_Zero. [Poskus dostopa Januar 2017].

[43] E. Upton, „Ten millionth Raspberry Pi, and a new kit,“ [Elektronski]. Available: https://www.raspberrypi.org/blog/ten-millionth-raspberry-pi-new-kit/. [Poskus dostopa Januar 2017].

[44] Raspberry Pi Foundation, „Raspbian,“ [Elektronski]. Available: https://www.raspberrypi.org/downloads/raspbian/. [Poskus dostopa Januar 2017].

[45] SourceForge.net, „WIN32 Disk Imager,“ [Elektronski]. Available: https://sourceforge.net/projects/win32diskimager/.

[46] Raspberry Pi Foundation, „Camera Module,“ [Elektronski]. Available: https://www.raspberrypi.org/products/camera-module-v2/. [Poskus dostopa Januar 2017].

[47] ATLAS, „Automatically Tuned Linear Algebra Software,“ [Elektronski]. Available: http://math-atlas.sourceforge.net/.

[48] Wikipedia, „Automatically Tuned Linear Algebra Software,“ [Elektronski]. Available: https://en.wikipedia.org/wiki/Automatically_Tuned_Linear_Algebra_Software.

[49] A. Rosebrock, „Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3,“ [Elektronski]. Available: http://www.pyimagesearch.com/2016/04/18/install-guide-raspberry-pi-3-raspbian-jessie-opencv-3/. [Poskus dostopa Januar 2017].

[50] Continuum Analytics, „Anaconda on the Raspberry Pi,“ 22 Marz 2013. [Elektronski]. Available: https://www.continuum.io/blog/developer/anaconda-raspberry-pi. [Poskus dostopa Januar 2017].

[51] „Caffe Model Zoo,“ [Elektronski]. Available: https://github.com/BVLC/caffe/wiki/Model-Zoo. [Poskus dostopa Januar 2017].

[52] Stanford Vision Lab, Stanford University, Princeton University, „IMAGENET,“ [Elektronski]. Available: http://www.image-net.org/. [Poskus dostopa Januar 2017].

[53] Reddit, „Copyright laws and machine learning algorithms,“ [Elektronski]. Available: https://www.reddit.com/r/MachineLearning/comments/3a24wx/copyright_laws_and_machine_learning_algorithms/?st=j08mm7en&sh=997716fa. [Poskus dostopa Januar 2017].

[54] Reddit, „Is it legal to use copyright material as training data?,“ [Elektronski]. Available: https://www.reddit.com/r/MachineLearning/comments/4qrgh8/is_it_legal_to_use_copyright_material_as_training/?st=j08mm5t5&sh=e25169ac. [Poskus dostopa Januar 2017].

[55] C. Metz, „https://www.wired.com/2016/02/ai-is-changing-the-technology-behind-google-searches/,“ 2 April 2016. [Elektronski]. Available: https://www.wired.com/2016/02/ai-is-changing-the-technology-behind-google-searches/. [Poskus dostopa Januar 2017].

[56] A. Jung, „GitHub,“ [Elektronski]. Available: https://github.com/aleju/imgaug. [Poskus dostopa Januar 2017].

[57] K. Ke-Yun Lin, „GitHub,“ [Elektronski]. Available: https://github.com/kevinlin311tw/caffe-augmentation. [Poskus dostopa Januar 2017].

[58] „AWS EC2 GPU enabled Caffe AMI,“ [Elektronski]. Available: https://github.com/BVLC/caffe/wiki/AWS-EC2-GPU-enabled-Caffe-AMI. [Poskus dostopa Januar 2017].

[59] D. Curro, „Training and Resuming,“ Februar 2016. [Elektronski]. Available: https://github.com/BVLC/caffe/wiki/Training-and-Resuming. [Poskus dostopa Januar 2017].

[60] D. Curro, „Borrowing Weights from a Pretrained Network,“ [Elektronski]. Available: https://github.com/BVLC/caffe/wiki/Borrowing-Weights-from-a-Pretrained-Network. [Poskus dostopa Januar 2017].

[61] D. Curro, „Fine Tuning or Training Certain Layers Exclusively,“ [Elektronski]. Available: https://github.com/BVLC/caffe/wiki/Fine-Tuning-or-Training-Certain-Layers-Exclusively. [Poskus dostopa Januar 2017].

[62] E. Shelhamer, „BVLC AlexNet,“ [Elektronski]. Available: https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet. [Poskus dostopa Januar 2017].

[63] ADDI Project, „ADDI Project,“ [Elektronski]. Available: http://www.fc.up.pt/addi/ph2%20database.html. [Poskus dostopa Januar 2017].

[64] F. Ercal, A. Chawla, W. Stoecker, H.-C. Lee in R. Moss, „Neural network diagnosis of malignant melanoma from color images,“ *IEEE Transactions on Biomedical Engineering,* pp. 837 - 845, Setember 1994.

[65] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau in S. Thrun, „Dermatologist-level classification of skin cancer with deep neural networks,“ *Nature,* pp. 115-118, 2 Februar 2017.

[66] E. Nasr-Esfahani, S. Samavi, N. Karimi, S. Soroushmehr, M. Jafari, K. Ward in K. Najarian, „Melanoma detection by analysis of clinical images using convolutional neural network,“ *Conf Proc IEEE Eng Med Biol Soc.,* pp. 1373-1376, Aug 2016.

[67] M. Binder, A. Steiner, S. M., S. Knollmayer, K. Wolff in H. Pehamberger, „Application of an artificial neural network in epiluminescence microscopy

pattern analysis of pigmented skin lesions: a pilot study,“ *British Journal of Dermatology,* April 1994.

[68] Y. He, „Models accuracy on ImageNet 2012 val,“ 25 Avgust 2017. [Elektronski]. Available: https://github.com/BVLC/caffe/wiki/Models-accuracy-on-ImageNet-2012-val. [Poskus dostopa December 2017].

[69] Aphex34, „Input volume connected to a convolutional layer,“ 15 December 2015. [Elektronski]. Available: https://en.wikipedia.org/wiki/File:Conv_layer.png. [Poskus dostopa Januar 2017].

[70] Aphex34, „Convolutional layers,“ 16 December 2015. [Elektronski]. Available: https://commons.wikimedia.org/wiki/File:Conv_layers.png. [Poskus dostopa Januar 2017].

[71] R. Munroe, „xkcd.com,“ [Elektronski]. Available: https://xkcd.com/1425/.

[72] Aphex34, „Max pooling,“ 16 December 2015. [Elektronski]. Available: https://commons.wikimedia.org/wiki/File:Max_pooling.png. [Poskus dostopa Januar 2017].

[73] D. Aghamirzaie in A. L. Salomon, „Deep Learning for Perception,“ 15 September 2015. [Elektronski]. Available: http://slideplayer.com/slide/8370683/. [Poskus dostopa Januar 2017].

[74] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever in R. Salakhutdinov, „Dropout: A Simple Way to Prevent Neural Networks from Overfitting,“ *Journal of Machine Learning Research 15,* pp. 1929-1958, 6 2014.

[75] S. Han, H. Mao in W. J. Dally, „Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding,“ 2015.

[76] Wikipedia, „Huffman coding,“ [Elektronski]. Available: https://en.wikipedia.org/wiki/Huffman_coding. [Poskus dostopa Januar 2017].