SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 478

Kvantiziranje dubokih konvolucijskih modela

Tomislav Ćosić

Zagreb, srpanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

ZAVRŠNI ZADATAK br. 478

Pristupnik:	**Tomislav Ćosić (0036526583)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Siniša Šegvić

Zadatak: **Kvantiziranje dubokih konvolucijskih modela**

Opis zadatka:

Duboki konvolucijski modeli danas su metoda izbora za mnoge zadatke računalnog vida. Međutim, njihova računska složenost isključuje mnoge zanimljive primjene u robotici i inteligentnom prometu. Taj problem možemo ublažiti zamjenom decimalnih računskih operacija odgovarajućim cjelobrojnim operacijama. U okviru rada, potrebno je upoznati se s konvolucijskim modelom ResNet-18. Istražiti postojeće pristupe za kvantiziranje težina dubokih modela. Odabrati slobodno dostupni skup slika te oblikovati podskupove za učenje, validaciju i testiranje. Oblikovati algoritme, validirati hiperparametre te uhodati učenje, validiranje i zaključivanje. Modificirati postupak učenja tako da na izlazu dobijemo kvantizirani model. Vrednovati naučene modele, prikazati postignutu točnost te provesti usporedbu s rezultatima iz literature. Komentirati učinkovitost učenja i zaključivanja. Predložiti pravce budućeg razvoja. Radu priložiti izvorni kod razvijenih postupaka uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 10. lipnja 2022.

Zahvaljujem se prof. dr. sc. Šegviću na pruženom znanju i korisnim savjetima. Zahvaljujem se obitelji i svim dragim osobama koje su mi dosad bile bezuvjetna potpora kroz školovanje.

SADRŽAJ

1. Uvod

Konvolucijske neuronske mreže vrsta su neuronskih mreža najčešće korištenih za rješavanje problema iz područja računalnog vida. Razvoj umjetnih neuronskih mreža započeo je sredinom prošlog stoljeća te se često temeljio na analizi mehanizama funkcioniranja mozga životinja ili ljudi. Ipak, treba naglasiti da cilj konvolucijskih modela, nije stvaranje *umjetnog mozga*, već rješavanje problema koji su takve prirode da ih ljudi često mogu lako riješiti, ali su to izuzetno teški problemi za rješavanje klasičnim algoritmima. To su problemi koji se u načelu svode na univerzalnu aproksimaciju funkcija. Cilj programskih rješenja u području računalnog vida je izvođenje zaključaka iz informacija u obliku fotografije ili videa, bila to klasifikacija, predikcija budućnosti ili neki drugi zadatak. Radi se o području koje izuzetno brzo napreduje. O potencijalu dovoljno govori činjenica da se predviđeni rast industrije računalnog vida trenutno kreće oko 7.3% godišnje za područje od 2021. do 2028. godine [4].

Ipak, izazov u uporabi u stvarnom svijetu, posebno u realnom vremenu, čini složenost računskih operacija tijekom obrade ulaza u konvolucijskom modelu. Naime, radi se o matričnom množenju i sličnim *računalno skupim* operacijama zbog čega mogućnost izvođenja u stvarnom vremenu ovisno o podatcima i dostupnoj računalnoj opremi postaje upitna. Značajna ubrzanja (ali uz cijenu gubitka točnosti) mogu se postići kvantizacijom modela, to jest modifikacijom određenih tipova podataka parametara modela kako bi računske operacije postale cjelobrojne umjesto decimalnih jer se računske operacije s cjelobrojnim brojevima na računalu izvode brže od onih s decimalnim brojevima.

2. Umjetne neuronske mreže

2.1. Motivacija i osnovni principi

Kao što je spomenuto u uvodu, umjetne neuronske mreže omogućavaju nam računalno rješavanje problema koji su inače izuzetno teški za računalno rješavanje. Uzmimo fotografiju psa kao primjer. Ako želimo napisati klasični algoritam koji bi trebao prepoznati psa ukoliko je on na fotografiji, zapravo moramo definirati što psa čini psom. To znači da trebamo jasno definirati ne samo kako izgleda pas, nego i paziti da ta definicija bude takva da vrijedi isključivo za pse (znači da na primjer "četiri noge, njuška i rep" nije dobra definicija jer recimo vrijedi i za mnoge druge životinje, ili čak možda ne vrijedi ni za sve pse jer neki nemaju sve noge ili nemaju rep). Kao što vidimo, radi se o izuzetno teškoj zadaći, a još nismo ni uzeli u obzir mnoge dodatne probleme, kao što su položaj psa, različite pasmine... Ipak, gotovo svi ljudi, uključujući čak i malu djecu, rutinski i bez poteškoća rješavaju ovaj i zaista jako velik broj sličnih problema u svakodnevnom životu.

Pošto klasični pristup nije realno primjenjiv na ovakve probleme, umjetna neuronska mreža povijesno je nastala kao ideja da se koristi drukčiji pristup, inspiriran prirodom, to jest mozgom živih bića.

2.1.1. Vrste problema

S obzirom na vrstu problema koji pokušavamo riješiti, zadatak će nam često biti klasifikacija ulaznih podataka ili regresija.

Klasifikacija Klasifikacija je pristup koji odabiremo kada želimo ulazni podatak svrstati (klasificirati) u jednu od unaprijed određenih klasa, odnosno želimo odrediti kojoj klasi podatak pripada. Primjer klasifikacije bio bi ranije spomenut primjer određivanja nalazi li se na fotografiji pas. Ovdje je važno primijetiti dva obilježja klasifikacije:

– pošto model zapravo donosi procjenu/odluku kojoj klasi dani primjer pripada,

broj mogućih izlaza modela je zapravo jednak broju mogućih klasa

– kod klasifikacije moguća su dva ishoda: model je točno klasificirao dani primjer ili nije. U našem primjeru sa psom bitno je je li model odlučio da je na fotografiji pas ili je pogriješio. Konkretan odabir klase ukoliko se radi o grešci nije naročito važan, to jest svejedno je je li model za psa procijenio da se radi o mački, zidu ili nečemu trećem. Sve greške se smatraju *jednako pogrešnima*.

Regresija Regresija je pristup u kojem radimo procjenu neke vrijednosti, no to koliko je procjena modela blizu stvarnoj vrijednosti igra ulogu. Primjer iz područja računalnog vida bi bila procjena dobi osobe pomoću fotografije osobe. Ukoliko je procjena modela 55 godina, a osoba zapravo ima 57 godina, to je puno bolja procjena nego da je model procijenio da osoba ima 20 godina.

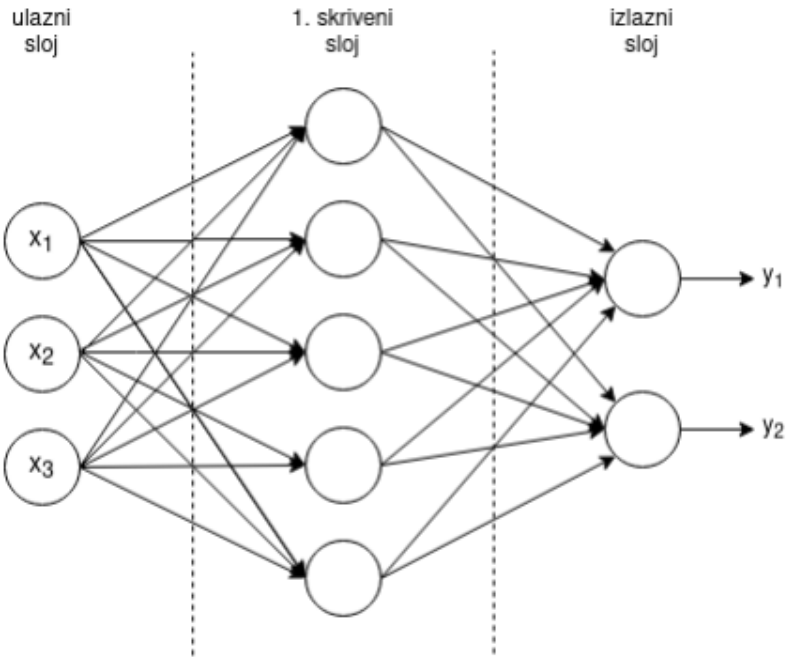
Ovaj rad će se baviti problemom klasifikacije.

2.1.2. Osnovna arhitektura neuronske mreže

Kako bismo shvatili kako umjetna neuronska mreža funkcionira, bitno je razumjeti osnovnu arhitekturu jedne neuronske mreže (slika 2.1.). Prvi sloj mreže čini ulazni podatak. U slučaju računalnog vida to je fotografija, to jest numeričke vrijednosti piksela. Zatim slijedi skriveni sloj. On se sastoji od određenog broja umjetnih neurona (detaljnije o njima u sljedećem potpoglavlju). Kao što vidimo, izlazi ulaznog sloja su zapravo ulazi za skriveni sloj. Isto vrijedi i u idućem koraku, gdje su izlazi skrivenog sloja zapravo ulazi izlaznog sloja. Izlazni sloj kod problema klasifikacije je vektor kod kojeg je broj elemenata jednak broju mogućih klasa (u konkretnom primjeru odabir se provodi između dvije klase).

Brojevne vrijednosti ulaze u mrežu kao komponente ulaznog sloja. U svakom sloju dolaze u jedan od neurona te promijenjena vrijednost izlazi iz neurona te ulazi u sljedeći sloj. Što se točno događa u neuronu, to jest kako se ta brojevna vrijednost mijenja bit će objašnjeno u poglavljima o umjetnom neuronu te o učenju.

Ova mreža je potpuno povezana. To znači da je izlaz neurona u jednom sloju povezan sa svim neuronima u idućem sloju. Općenito, to ne mora biti slučaj. Također, u mreži na slici 2.1 nema povratnih veza (veza koje su usmjerene iz kasnijeg sloja u raniji), kao ni preskočnih veza (veza koje *preskaču* sloj ili slojeve mreže).



Slika 2.1: Jednostavna umjetna neuronska mreža

2.1.3. Umjetni neuron

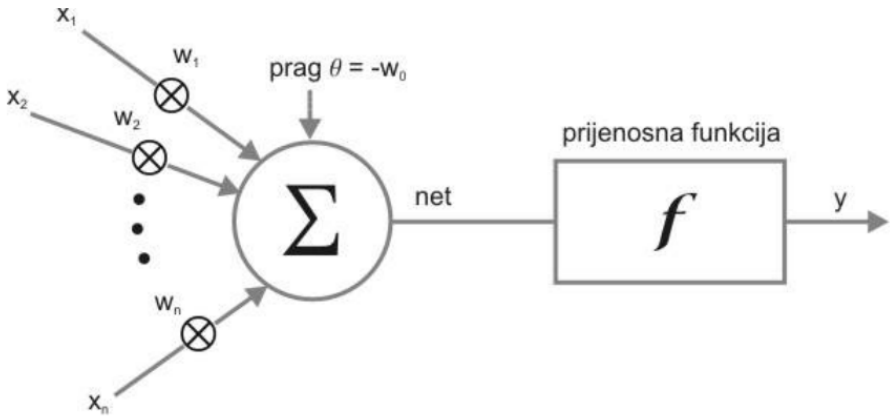
Temeljni gradivni element umjetne neuronske mreže je umjetni neuron (slika 2.2.). On ima niz ulaza, na slici označenih s x_i. Svaki ulaz se u neuronu prvo množi s vlastitom težinom (w_i). Zatim se dobivene vrijednosti zbrajaju, uz mogućnost da se pribroji i dodatna vrijednost; slobodni član ovdje označen s w_0 (slobodni jer se ne množi ni s jednom komponentom ulaza). w_0 nazivamo i pomak (*engl. bias*).

$$\sum_{i=0}^{n} w_i * x_i \tag{2.1}$$

U navedenom izrazu *n* je broj komponenti ulaznog tenzora. Zbog praktičnosti računa, podrazumijeva se da je $x_0 = 1$ iako x_0 zapravo ne postoji, no to je samo tehnički detalj jer se i inače izraz može zapisati kao

$$\sum_{i=1}^{n} w_i * x_i \ + w_0 \tag{2.2}$$

Dobiveni rezultat je argument prijenosne funkcije, čiji rezultat je izlaz neurona (na slici označen s *y*).



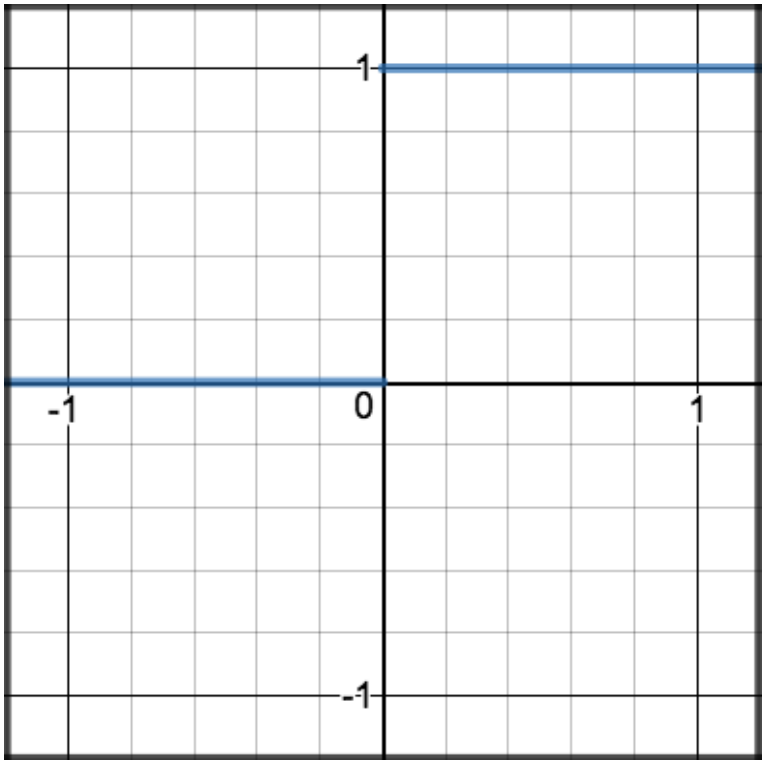
Slika 2.2: Model umjetnog neurona

2.1.4. Aktivacijska funkcija

Konačno, zadnji korak u obradi ulaza u neuronu je primjenjivanje aktivacijske funkcije (na slici 2.2. označena kao prijenosna funkcija) na vrijednost dobivenu zbrajanjem umnožaka ulaza i odgovarajućih težina. Postoji više različitih aktivacijskih funkcija. Aktivacije dubokih modela igraju veliku ulogu u ukupnim performansama i karakteristikama modela. Kako se područje razvijalo, tako se i mijenjao odabir aktivacijske funkcije pokušavajući riješiti nedostatke pojedinih funkcija.

***Step* funkcija** Na početku razvoja neuronskih mreža aktivacijska funkcija bila je *step* funkcija:

$$step(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases} \tag{2.3}$$



Slika 2.3: Graf *step* funkcije

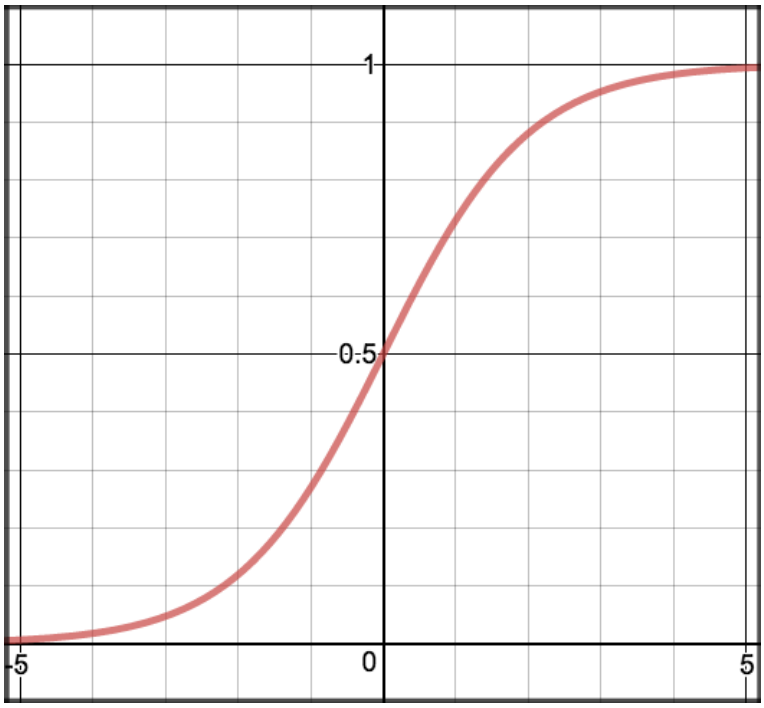
U kontekstu suvremenog gledišta, *step* funkcija ima nekoliko nepoželjnih karakteristika za korištenje u neuronskim mrežama. Prvi problem je što je tijekom procesa učenja neuronske mreže potrebno razmotriti derivaciju aktivacijske funkcije, a *step* funkcija nije derivabilna za *x = 0* (slika 2.3.). Druga nepoželjna karakteristika je što je vrijednost derivacije u svim točkama u kojima je *step* funkcija derivabilna jednaka nuli, što u kontekstu učenja neuronske mreže znači da ne znamo u kojem je smjeru potrebno prilagoditi pojedine parametre mreže. Još jedna nepoželjna karakteristika *step* funkcije je što za male promjene na ulazu može dati velike promjene na izlazu. Konkretno, u okolini ulaza 0 *step* funkcija će za negativan broj vrlo blizu nuli imati vrijednost 0, dok će za pozitivan broj koji je vrlo malo veći od nule poprimiti vrijednost 1.

$$step(0 - \epsilon_0) = 0 \tag{2.4}$$

$$step(0 + \epsilon_0) = 1 \tag{2.5}$$

Sigmoidalna funkcija Sigmoidalna funkcija (slika 2.4.) ispravlja mnoge nedostatke *step* funkcije. Nelinearna (kao i njezine kombinacije) što proširuje spektar problema koji je model sposoban riješiti, to jest funkcija koje je sposoban aproksimirati. Nadalje,

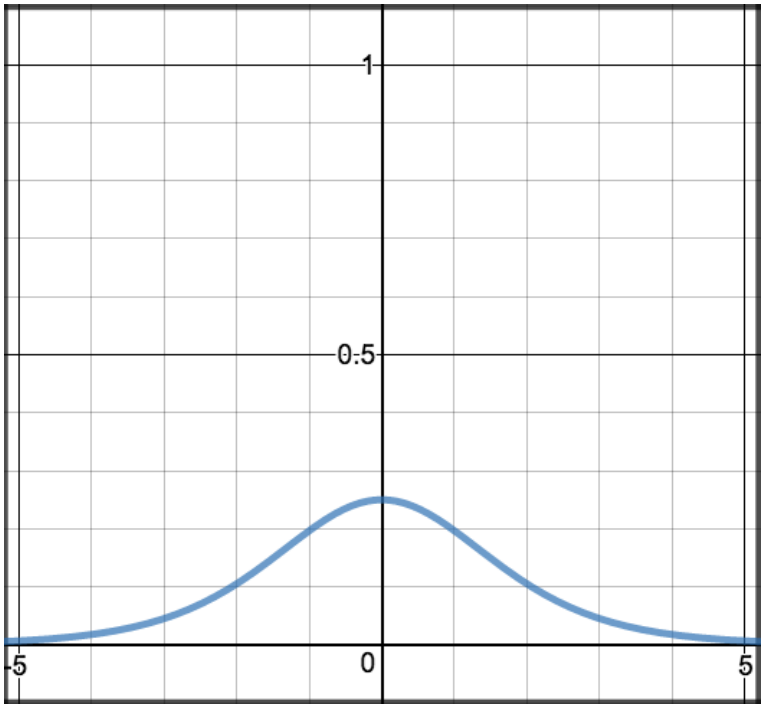
domena je skup realnih brojeva, a kodomena realni interval [0, 1]. Za razliku od *step* funkcije, daje analognu aktivaciju te je derivabilna na cijeloj svojoj domeni. Ipak, ni sigmoidalna funkcija nije bez nedostataka. Za vrlo velike ili vrlo male vrijednosti, velika promjena ulaza rezultira malom promjenom izlaza, odnosno gradijent funkcije u tim vrijednostima teži nuli, što nas vodi na isti problem kakav je opisan kod *step* funkcije. To nas vodi na problem nestajućeg gradijenta (*engl. vanishing gradient*). Za vrlo velike ili vrlo male vrijednosti iz domene, zbog vrlo malih vrijednosti gradijenta funkcije značajno je otežan proces učenja.



Slika 2.4: Graf sigmoidalne funkcije

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

Zanimljivo je promotriti derivaciju sigmoidalne funkcije:



Slika 2.5: Graf derivacije sigmoidalne funkcije

$$\frac{d\sigma(x)}{dx} = \frac{d\frac{1}{1+e^{-x}}}{dx} \tag{2.7}$$

$$= \frac{e^{-x}}{(1+e^{-x})^2} \tag{2.8}$$

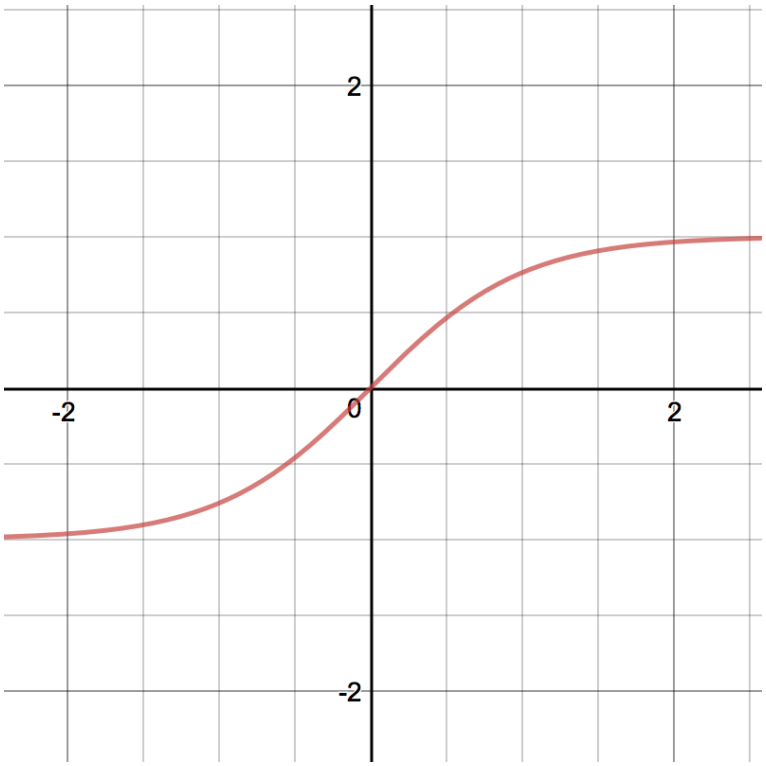
$$= \frac{1}{1+e^{-x}} * \frac{1-1+e^{-x}}{1+e^{-x}} \tag{2.9}$$

$$= \sigma(x) * (1-\sigma(x)) \tag{2.10}$$

Kao što vidimo iz izraza 2.9, vrijednost derivacije sigmoidalne funkcije u nekoj točki može se izračunati kao vrijednost same funkcije u toj istoj točki. To je vrlo praktično svojstvo koje se može praktično iskoristiti za brži izračun derivacije funkcije. Naime, ne samo da nema potrebe raditi s relativno kompliciranim izrazima, već se i vrijednost sigmoidalne funkcije u točki x može predmemorirati (*engl. caching*) te ju onda nije potrebno ponovno izračunavati za upotrebu u izrazu 2.9.

Tangens hiperbolni Tangens hiperbolni pokazuje slične karakteristike kao sigmoidalna funkcija, ali uz neka poboljšanja. Naime, tangens hiperbolni ima srednju vrijednost (*engl. mean*) jednaku nuli (*engl. zero-centered*). Također, gradijent tangensa hiperbolnog je nešto strmiji u odnosu na gradijent sigmoide, što je poželjna značajka

kod procesa učenja modela. Nažalost, problem nestajućeg gradijenta spomenut kod sigmoidalne funkcije ostaje prisutan i kod tangensa hiperbolnog.



Slika 2.6: Graf tangensa hiperbolnog.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.11}$$

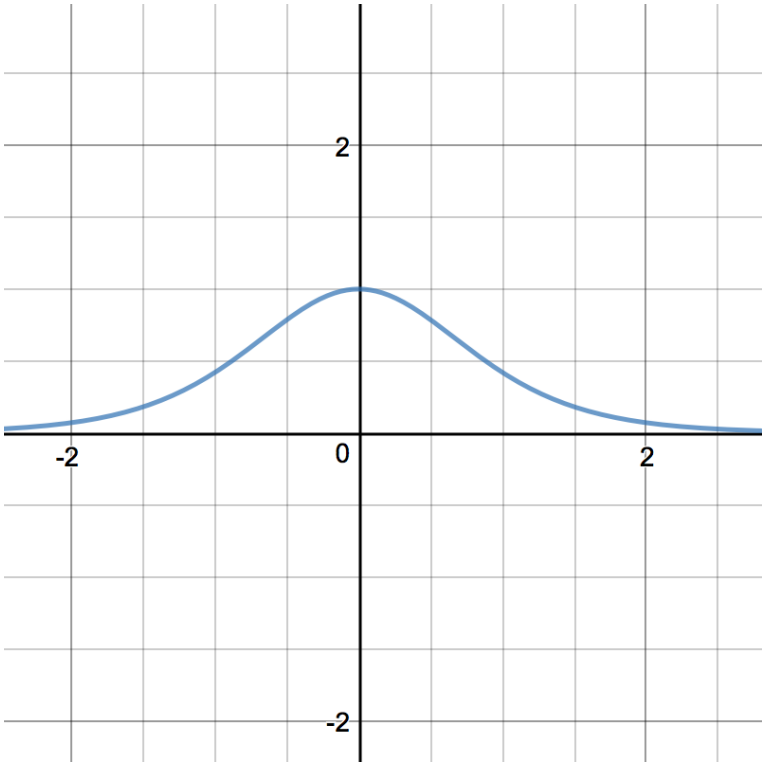
$$\frac{dtanh(x)}{dx} = \frac{d\frac{e^x - e^{-x}}{e^x + e^{-x}}}{dx} \tag{2.12}$$

$$= \frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2} \tag{2.13}$$

$$= 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} \tag{2.14}$$

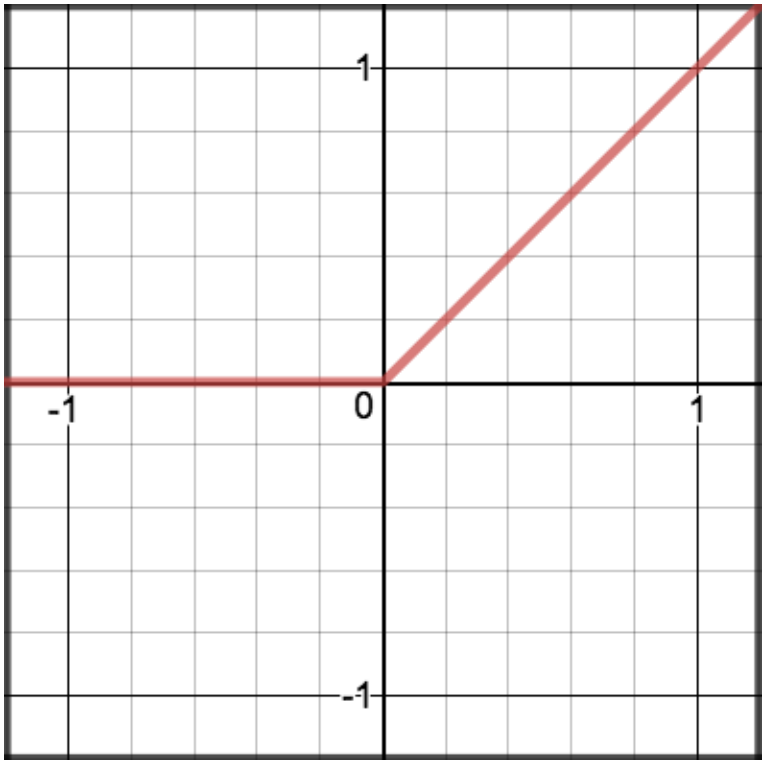
$$= 1 - tanh^2(x) \tag{2.15}$$

Kao što vidimo iz 2.14, i kod tangensa hiperbolnog vrijede prednosti predmemoriranja navedene kod sigmoidalne funkcije.



Slika 2.7: Graf derivacije tangensa hiperbolnog.

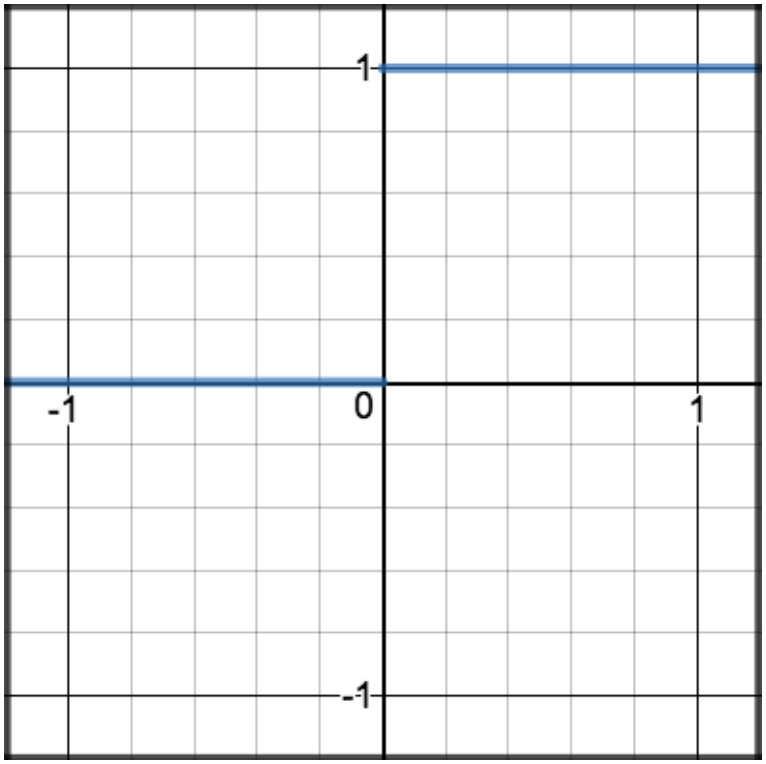
ReLU ReLU (kratica dolazi od engl. ***Re**ctified **L**inear **U**nit*, u hrvatskom jeziku se koristi i naziv zglobnica) je aktivacijska funkcija koja usprkos svom imenu nije linearna. U načelu, radi se o funkciji definiranoj na $x \in \mathbb{R}$ koja je na negativnom dijelu domene jednaka nuli, dok na pozitivnom dijelu poprima vrijednost x (slika 2.8).



Slika 2.8: Graf zglobnice.

$$ReLU(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \tag{2.16}$$

Ovakva konstrukcija funkcije u isto vrijeme omogućuje nelinearnost kako bi model bio sposoban riješiti širu klasu problema te rješava problem nestajućeg gradijenta jer kod učenja modela neće doći do uzastopnog množenja malih pozitivnih vrijednosti kao kod sigmoidalne funkcije. Ipak, ne dolazi bez svojih nedostataka. Prvi problem je pojava eksplodirajućeg gradijenta. Problem je što izlaz zglobnice nije odozgo ograničen pa je moguće da dođe do eksponencijalnog rasta umnoška gradijenata uzastopnim množenjem vrijednosti većih od 1. Drugi problem je što za negativne vrijednosti zglobnica daje vrijednost 0 i ima derivaciju 0. Takva aktivacija zapravo odgovara mrtvom neuronu i ukoliko se dogodi da su težine takvih vrijednosti da ulaz u zglobnicu bude negativan broj, to će rezultirati smanjenjem ukupnog kapaciteta modela, odnosno sposobnosti modela da se prilagodi proizvoljnoj funkciji svojim parametrima zato što težine neurona s *mrtvom* aktivacijom ne igraju nikakvu ulogu u obradi ulaza.



Slika 2.9: Graf derivacije zglobnice.

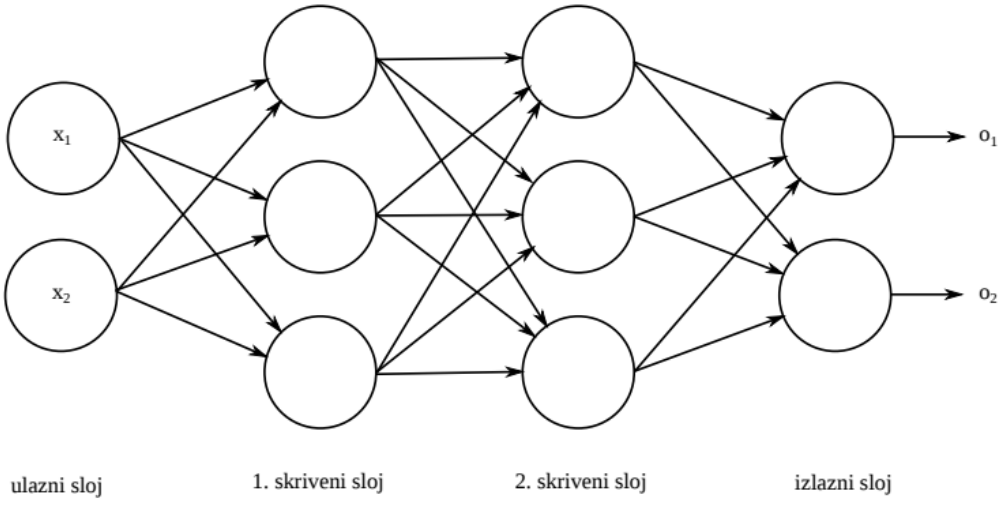
$$\frac{dReLU(x)}{dx} = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases} \tag{2.17}$$

2.2. Duboki modeli

Dosad opisani modeli dobro su rješenje za rješavanje jednostavnijih problema, ne samo u smislu kompleksnosti i suptilnosti zadatka nego i jednostavnijih u smislu dimenzionalnosti podataka. Kada se modeli s jednim skrivenim slojem suoče s podatkom velikih dimenzija, taj skriveni sloj se mora često značajno povećati kako bi model mogao uspješno zaključivati što, posebno uz pretpostavku potpune povezanosti, vodi na eksponencijalno povećanje broja parametara modela. Stoga je jasno kako modeli s jednim skrivenim slojem nisu dobar odabir za takve zadatke.

Drugi pristup su duboki modeli, koji svoje ime duguju činjenici da imaju više od jednog skrivenog sloja (slika 2.10). Razlog određene nepopularnosti dubokih modela u povijesnom razvoju područja između ostalog leži u činjenici da duboki modeli ne garantiraju konvergenciju učenja. Također, pošto model ima više skrivenih slojeva, to znači i da će tijekom učenja doći do uzastopnog množenja gradijenata, otvarajući mogućnost za ranije spomenute probleme nestajućeg i eksplodirajućeg gradijenta. Nadalje, učenje dubokih modela često zahtijeva veliki skup podataka, za koji obično zapravo nema prave alternative nego ručno označiti iznimno velik broj fotografija.

Ipak, ovim poteškoćama se doskočilo u dovoljnoj mjeri da duboki modeli nađu svoju praktičnu primjenu. Zbog nestajućeg gradijenta se u načelu izbjegava korištenje sigmoide za aktivacije. Također, iako nema formalne garancije konvergencije, postoje mnoge metode koje poboljšavaju konvergenciju učenja.



Slika 2.10: Duboki model.

Primjena dubokih modela u praksi u velikoj mjeri koristi grafičke procesore (*engl.*

GPU) i specijaliziranu programsku podršku kako bi performanse dubokih modela bile dostatne za svakodnevnu primjenu u stvarnom svijetu, posebno u stvarnom vremenu. Stoga ne iznenađuje da je značajan iskorak u primjenjivosti dubokih modela došao s napretkom računalne opreme u aspektu procesne moći. Također, relativno nedavni razvoj tenzorskih procesora (*engl. TPU*) potencijalno je veliki napredak u procesnoj moći za duboke modele. Tenzorski procesori su svojim dizajnom optimizirani za rad s tenzorima, odnosno za matrični račun. To je račun koji je inače poprilično spor kada se računa "običnim" procesorima (posebno CPU-om) i predstavlja ograničavajući faktor u ukupnim performansama dobokog modela. Prema [6], tenzorski procesor je tijekom testiranja bio 15 do 30 puta brži od grafičkog i klasičnog procesora. Također, TPU-ovi su pokazali 30 do 80 puta veću vrijednost TOPS/W (TOPS je jedinica koja označava 10^{12} operacija u sekundi, engl. Trillion Operations per Second ili Tera Operations per Second, W je Watt). U načelu, TOPS/W je mjera energetske učinkovitosti, uspoređujući broj odrađenih rečunskih operaciju i u tu svrhu utrošenu energiju. Štoviše, autori [6] sugeriraju da bi korištenje GPU-ove GDDR5 memorije utrostručilo TOPS performanse tenzorskog procesora te dovelo TOPS/W na 70 puta veću vrijednost od one kod grafičkih procesora, odnosno 300 puta veću uspoređujući tenzorske s klasičnim procesorima (CPU).

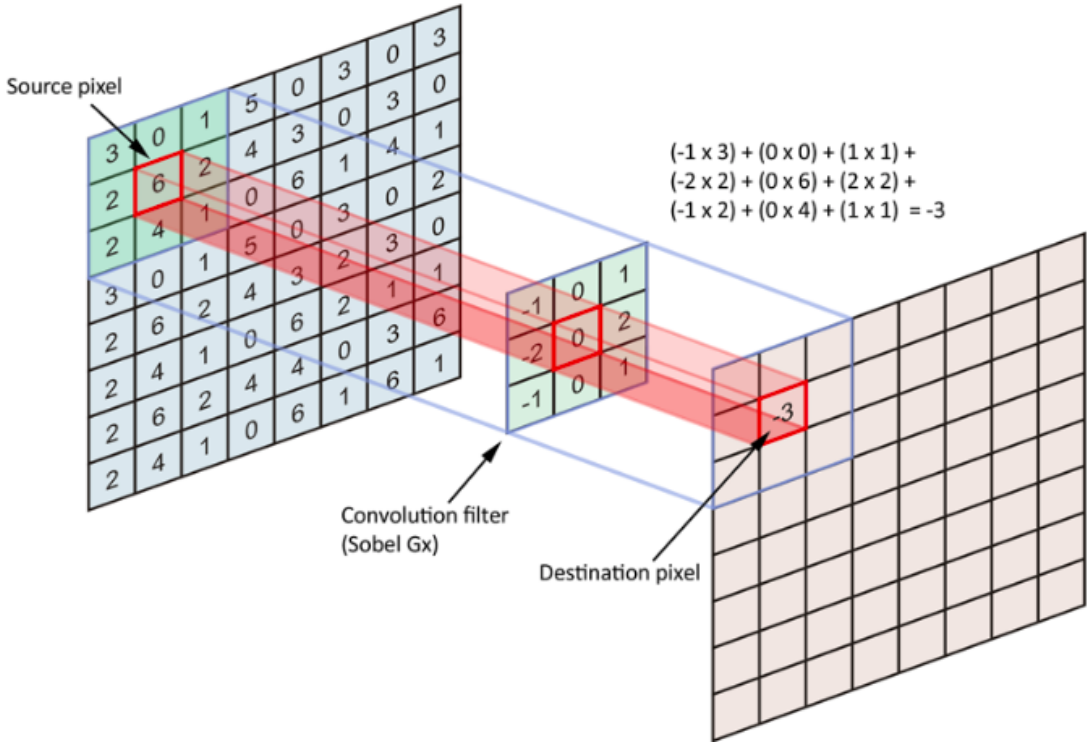
2.3. Duboki konvolucijski modeli

Motivacija za uvođenje konvolucijskih slojeva proizlazi iz činjenice da su veličine modela koji rješavaju zahtjevnije zadatke i dimenzionalnost ulaza takvi da ni metode koje ubrzavaju računanje opisane u prethodnim poglavljima nisu dovoljne. Uz porast veličine modela, duboki model s isključivo potpuno povezanim slojevima lako postaje prevelik, to jest računanje potrebnih vrijednosti za takav model postaje presporo. Rješenje u obliku konvolucijskih modela temelji se na činjenici da nema potrebe određeni piksel dovoditi u vezu sa svim drugim pikselima, nego ga možemo promatrati u kontekstu njegove lokalne okoline. Tako ne smanjujemo samo broj parametara i time poboljšavamo vremenske performanse modela, već i omogućavamo modelu da detektira lokalne značajke ulaznih podataka. To je izuzetno poželjno svojstvo. Kako bismo objasnili zašto, pogledajmo što je zapravo konvolucija. Na slici 2.11 vidimo tri komponente: lijevo se nalazi fotografija nad kojom će se primijeniti konvolucija, u sredini se nalazi konvolucijska matrica (poznata kao filter ili jezgra, engl. *kernel*), dok se desno nalazi grafički prikaz rezultata. Konvolucija je skalarni produkt matrice I (na slici lijevo) u općem slučaju dimenzije $[W \times H]$ i filtra (na slici u sredini) koji je obično

kvadratni, to jest dimenzije $[F \times F]$. Rezultat skalarnog produkta neka je matrica O. O će u našem primjeru biti dimenzije $[W-F+1 \times H-F+1]$, a vrijednost na poziciji $x, y : x \in [0, W-F], y \in [0, H-F]$ računa se kao:

$$O_{\text{x,y}} = \sum_{i=x}^{x+F-1} \sum_{j=y}^{y+F-1} I_{\text{i,j}} \cdot F_{\text{i-x, j-y}} \tag{2.18}$$

Intuitivno, gledajući sliku 2.11 možemo reći da je konvolucija dijela ulaznog podatka i filtera zapravo primjena filtera nad tim područjem ulaza te zapisivanje rezultata u izlaznu matricu (primijetite da to čini izlaznu matricu manje dimenzionalnosti od ulazne). Izlazna matrica naziva se mapa značajki (*engl. feature map*).



Slika 2.11: Konvolucija.

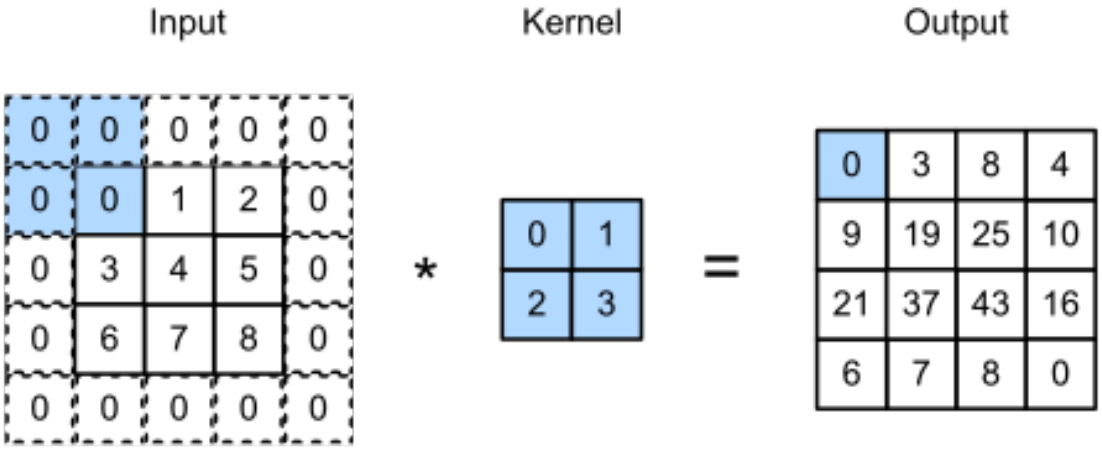
Skalarni produkt ulaza u filtera se obično provodi tako da se filter "šeta" preko ulazne matrice, obično slijeva nadesno i odozgo prema dolje. Ako se filter dizajnira tako da detektira neku značajku u ulazu, konvolucije čiji su rezultati visoke vrijednosti zapravo znače da je značajka detektirana na dijelu ulaza nad kojim je konvolucija primijenjena. Stoga su konvolucijski slojevi moćna oruđa u detekciji značajki. Nadalje, za razliku od potpuno povezanih modela koji jedan piksel dovode u vezu sa svim ostalima, konvolucijski modeli su zbog opisanog načina funkcioniranja invarijantni na translaciju. Naime, s jedne strane će uzorak aktivacija kod potpuno povezanog modela

biti potpuno drukčiji ovisno o tome gdje se u sceni nalazi tražena značajka. Efektivno, za potpuno povezani model koji traži psa na fotografiji su pas gore lijevo u sceni i pas dolje desno u sceni potpuno različite stvari koje se odvojeno uče i nisu ništa drukčiji nego razlika između prepoznavanja psa i prepoznavanja stola. S druge strane, konvolucijski filter se primjenjuje na lokalno područje scene kroz cijelu scenu te će pojava tražene značajke na bilo kojem lokalnom području rezultirati sličnom vrijednošću skalarnog produkta.

S obzirom da prolaz ulaznog podatka kroz konvolucijski sloj rezultira izlazom manje dimenzije, uz prikazani postupak konvolucije otvaramo mogućnost da izlaz (mapa značajki) bude manjih dimenzija od ulaznog podatka. To može biti u redu, no ipak bismo htjeli imati bolju kontrolu nad dimenzijama. To postižemo kroz dvije tehnike: nadopunjavanje (*engl. padding*) i korak (*engl. stride*).

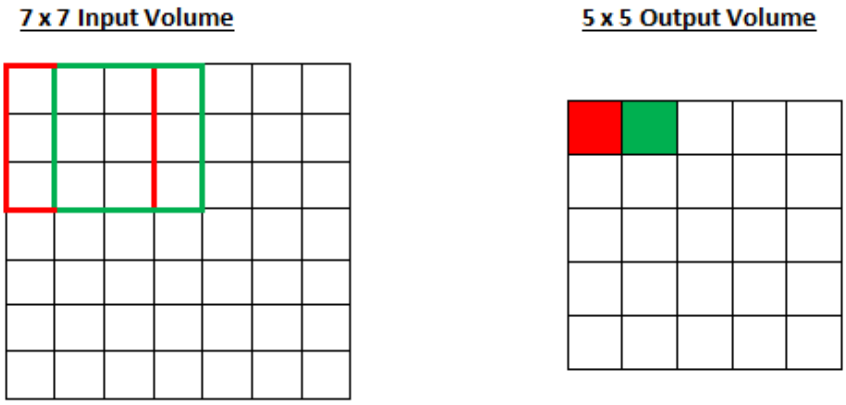
Nadopunjavanje Primjer mogućeg nadopunjavanja je prikazan na 2.12. Ovdje je sa svake strane ulaza dodan po jedan red, odnosno stupac ispunjen nulama. Kao što vidimo, to ulaz povećava s dimenzije $[3 \times 3]$ na $[5 \times 5]$, što izmnoženo s filtrom dimenzije $[2 \times 2]$ daje rezultat dimenzije $[4 \times 4]$. Vidimo da bi eventualnom promjenom broja dodanih redaka odnosno stupaca na izlazu konvolucijskog sloja dobili mapu značajki drukčijih dimenzija, što možemo iskoristiti za manipulaciju dimenzija.

Također, nadopunjavanje nam omogućuje da primjenimo punu konvoluciju (cijeli prolaz filtra preko piksela) čak i ako se radi o rubnom pikselu. To bez nadopunjavanja ne bismo mogli, a onda je doprinos krajnjem rezultatu piksela daljih od rubova veći od piksela bliže rubovima jer bez nadopunjavanja filter više puta prelazi preko piksela koji su bliže sredini od onih na rubovima, a posebno od onih u kutovima.

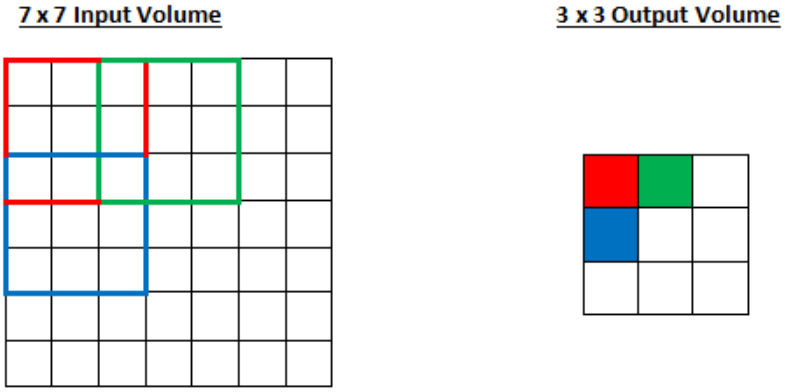


Slika 2.12: Primjer nadopunjavanja.

Korak Korak određuje za koliko će se mjesta filter pomaknuti u svakom koraku, bilo to horizontalno ili vertikalno. Na slici 2.13 vidimo primjer gdje je korak jednak 1 (takav je bio u prijašnjim primjerima). Početni položaj nad kojim se filter primjenjuje nalazi je na slici označen crvenim obrubom, a rezultat konvolucije se upisuje na polje izlazne matrice obojano crveno. Nakon računanja skalarnog produkta filtra i područja označenog crveno, iduće područje će biti ono na slici označeno zeleno. Analogno, rezultat odgovarajućeg skalarnog produkta upisat će se na zeleno polje.



Slika 2.13: Primjer konvolucije s korakom 1.



Slika 2.14: Primjer konvolucije s korakom 2.

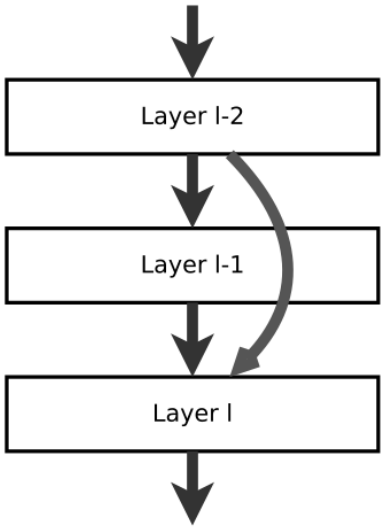
S druge strane, na slici 2.14 vidimo primjer u kojem korak sada iznosi 2. Prvo područje na ulaznom podatku koje se množi s filtrom je isto kao u slučaju kada je korak jednak 1, no kao što vidimo, zeleno područje nije jedno već dva mjesta udesno u odnosu na crveno. Analogno, i kad se radi o vertikalnom koraku, pomak iznosi 2 (pozicija plavog kvadrata u odnosu na crveni na lijevoj matrici). Povećanje koraka s 1 na 2 u ovom primjeru za posljedicu ima smanjivanje dimenzije izlaza s $[5 \times 5]$ na

$[3 \times 3]$.

Postoje dva primarna razloga za povećanje koraka u konvolucijskom sloju. Prvi je ubrzanje koje dolazi kao posljedica manjeg broja skalarnih produkata koje je potrebno izračunati, a drugi razlog je smanjivanje dimenzije izlaza (sažimanje ili *engl. downsampling*).

3. ResNet-18

U implementaciji se kao model koji se kvantizira koristio model ResNet-18 [5]. ResNet-18 je duboki konvolucijski model. Ova arhitektura sastoji se od 18 slojeva, no postoje i druge verzije (npr. ResNet-34 s 34 sloja). Što ResNet čini velikim postignućem je upotreba preskočnih veza u slojevima (*engl. skip connections*), čija je jednostavna shema prikazana na slici 3.1.



Slika 3.1: Osnovna shema preskočne veze.

Namjera ovakve arhitekture je suočavanje s problemom nestajućeg gradijenta u dubokim konvolucijskim modelima. Kao što smo diskutirali u ranijim poglavljima, u modelima s velikim brojem slojeva dolazi do uzastopnog množenja gradijenata kroz primjenu lančanog pravila što za posljedicu ima male rezultate koji praktično onemogućuju učenje modela, jer uz mali rezultat model ne može odrediti u kojem smjeru mijenjati parametre. Preskočne veze u načelu pružaju izravni "put" gradijentu u unatražnom prolazu (*engl. backpropagation*) tijekom učenja te tako značajno pospješuju učenje.

Intuitivno, ideja preskočnih veza vrlo je elegantna. Označimo ulaz u sloj I na slici

3.1 kao

$$h(x) = f(x) + x \tag{3.1}$$

gdje je x izlaz iz sloja $I - 2$ koji do sloja I dolazi kroz preskočnu vezu, a $f(x)$ je izlaz iz sloja $I - 1$.

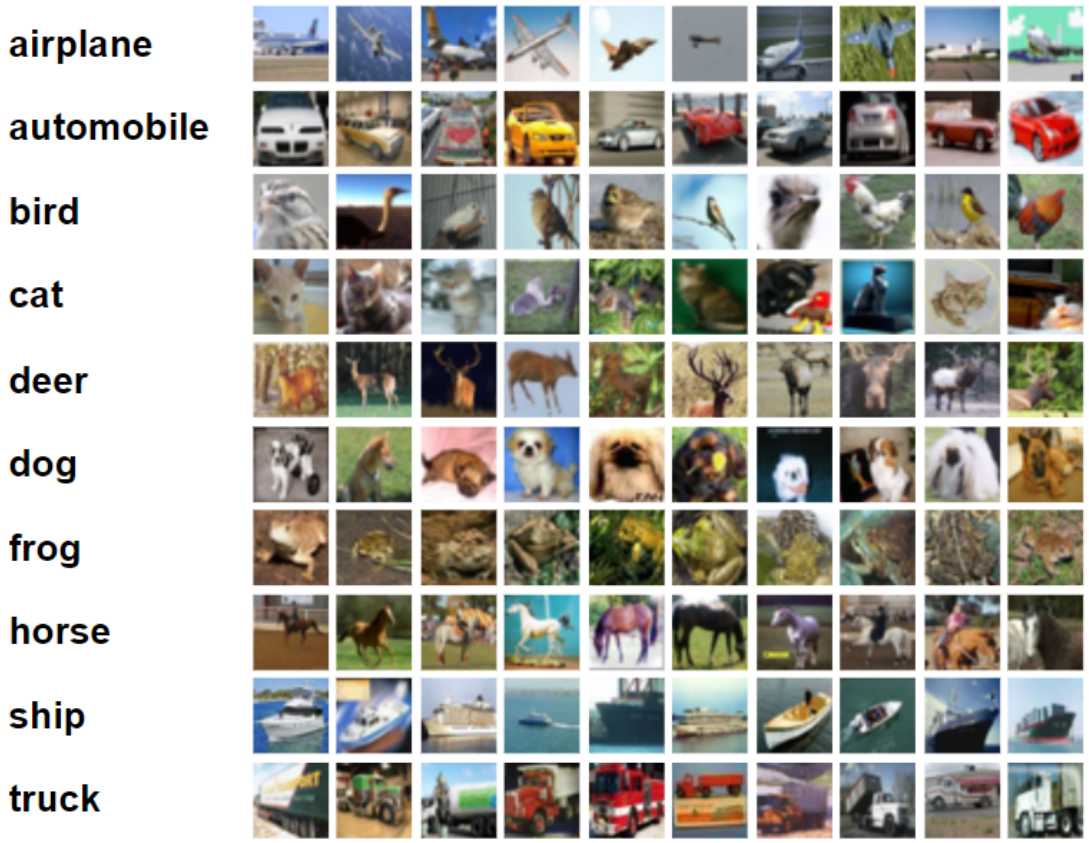
Osim ranije spomenute prednosti preskočnih veza, veliku ulogu igra da se, uz pretpostavku da je $f(x) = 0$ funkcija koja je modelu laka za naučiti, izraz 3.1 uvijek može svesti na

$$h(x) = x \tag{3.2}$$

To zapravo znači da će u najgorem slučaju I-ti sloj moći samo proslijediti izlaz sloja $I - 1$, tako onemogućujući degradaciju performansi modela zbog velike dubine, što je bila česta pojava u dubokim modelima s vrlo velikim brojem slojeva.

4. CIFAR-10

Skup podataka korišten u implementaciji ovog rada je CIFAR-10 [7]. Ovaj se skup podataka sastoji od 60000 fotografija iz 10 različitih klasa (4.1).



Slika 4.1: Primjer fotografija iz CIFAR-10 s odgovarajućim oznakama klasa.

Skup podataka je podijeljen na dva dijela: 50000 čini skup za učenje (po 5000 fotografija iz svake od 10 klasa), a preostalih 10000 čini skup za evaluaciju. Klase su disjunktne (nema fotografije koja pripada više od jednoj klasi, kao ni fotografije koja ne pripada nijednoj).

5. Kvantizacija

Modeli obično sadržavaju parametre u FP32 formatu, to jest 32-bitne decimalne brojeve. S obzirom na činjenicu da je računalno rješavanje matematičkih problema s brojevima u tom formatu bitno sporije nego u formatima manje preciznosti, postoji ideja promjene formata tih parametara. Mi ćemo dio parametara modela promijeniti iz FP32 u INT8, odnosno cijele brojeve. Kao rezultat očekujemo ubrzanje modela koje je posljedica bržih računskih operacija, no postoji i negativna posljedica: gubitak preciznosti. Naime, ovom promjenom oduzimamo modelu mogućnost da vrijednost parametra postavi na vrijednost između dva cijela broja te time unosimo grešku u odnosu na vrijednost parametra koju bi model idealno htio.

Ipak, ukoliko je povećanje brzine modela značajno, a gubitak preciznosti prihvatljiv, kvantizaciju ima smisla primijeniti jer otvara mogućnost da kompleksnije modele koji bi inače bili prespori za primjenu u nekom scenariju ubrza dovoljno da se mogu koristiti. Tu prije svega mislim na primjenu dubokih modela u sustavima koji rade u stvarnom vremenu.

Razlikujemo tri vrste kvantizacije: dinamičku, statičku i naučenu.

5.1. Dinamička kvantizacija

Dinamička kvantizacija postupak kvantizacije provodi nad težinama modela. S obzirom da su težine do kvantizacije bile pohranjene u 32-bitnom formatu, a zamjenjujemo ih 8-bitnim cijelim brojevima, očekujemo da će težine nakon dinamičke kvantizacije zauzimati četvrtinu memorijskog prostora kojeg su zauzimale dotad.

Važno je primijetiti da, iako su težine kvantizirane, aktivacije iz prethodnog sloja koje se množe s tim težinama nisu. Kako bi se te operacije mogle provoditi u INT8 aritmetici, potrebno je kvantizirati i aktivacije. S obzirom da vrijednosti aktivacija nisu unaprijed poznate, kod dinamičke kvantizacije aktivacije se kvantiziraju tijekom izvođenja. Za kvantizaciju aktivacija potrebno je naći najmanju i najveću vrijednost, što je postupak složenosti *O(n)*, izračunati potrebne parametre te provesti samu kvantizaciju

nad svakim od elemenata tenzora kojeg se kvantizira. To je proces koji ipak traje neko vrijeme pa se učinkovitost dinamičke kvantizacije zasniva na pretpostavci da će vrijeme uštađeno upotrebom brže INT8 aritmetike biti veće od vremena izgubljenog na kvantizaciju aktivacija. Istinitost te pretpostavke ovisi o konkretnoj arhitekturi na kojoj se model izvodi, što znači da na nekoj konkretnoj arhitekturi dinamička kvantizacija može značajno ubrzati model, no u općem slučaju se to ne može garantirati.

5.2. Statička kvantizacija

Za razliku od dinamičke, statička kvantizacija kvantizira i aktivacije prije izvođenja. Problem što nam vrijednosti aktivacija koje će se pojavljivati tijekom izvođenja nisu poznate rješavamo dodavanjem promatrača (*engl. observers*) u model koji će pratiti vrijednosti aktivacija koje se pojavljuju tijekom učenja. Nakon učenja, kroz model još jednom pustimo skup za treniranje kako bi promatrači pratili aktivacije i izračunali parametre potrebne za kvantizaciju. Primijetimo, ti parametri su se kod dinamičke aktivacije računali tijekom zaključivanja na temelju konkretnih aktivacija koje su se tad pojavile. Ovdje ih računamo jednom, prije zaključivanja, ali ih temeljimo na podatcima za učenje, pretpostavljajući da će podatci koji će se pojavljivati tijekom zaključivanja biti slični onima iz skupa za učenje. Još jedan nedostatak je potreba za intervencijom nad programskim kodom modela. Naime, moramo dodati dodatne slojeve za kvantizaciju i dekvantizaciju na ulazu, odnosno izlazu modela.

Ipak, prednost takvog postupka je značajno ubrzanje zaključivanja jer, iako samu kvantizaciju provodimo tijekom zaključivanja, svi parametri su spremni prije zaključivanja.

5.3. Naučena kvantizacija

Statička i dinamička kvantizacija pripadaju vrsti kvantizacija koje se događaju nakon faze učenja (*engl. post-training quantization*). S druge strane, naučena kvantizacija odvija se i tijekom učenja, gdje se u modelu u unaprijednom prolazu simuliraju kvantizirani parametri. Promatrač dodan u model ovaj šum umanjuje prilagodbom parametara modela. Tako model nakon kvantizacije ima minimalni gubitak preciznosti.

Konkretan način simulacije kvantizacije tijekom učenja zasniva se na uvođenju lažne kvantizacije svake aktivacije osim izlazne (*eng. fake quantization*). Lažna kvantizacija sastoji se od kvantiziranja aktivacije, obavještavanja promatrača o dobivenim

vrijednostima (kvantiziranim) te dekvantiziranja kvantizirane vrijednosti koja se prosljeđuje dalje kroz model. Ta dekvantizirana vrijednost zbog gubitka preciznosti u zapisu brojeva neće biti jednaka originalnoj aktivaciji. Ta se pojava naziva kvantizacijski šum. Promatrač dodan u model ovaj šum umanjuje prilagodbom parametara modela. S obzirom da bi model tako trebao postati prilagođen na postupak kvantizacije, nakon kvantiziranja ovakvog modela očekujemo minimalan gubitak preciznosti.

6. Eksperimentalni rezultati

Modeli su razvijani kroz PyTorch radni okvir (*engl. framework*). Zatim su pretvoreni u ONNX (*Open Neural Network Exchange*) format. ONNX je otvoren format, pogodan za prijenos modela iz jednog radnog okvira u drugi. Evaluacija modela je izvedena pomoću ONNX Runtime-a te TensorRT-a. Oba API-ja provode optimizaciju modela gdje pronađu mogućnost za ubrzanje izvođenja.

Također, ONNX Runtime omogućuje i izvođenje uz JIT (just-in-time) kompilaciju, gdje prevoditelj tijekom izvođenja neprestano traži prilike za dodatna ubrzanja.

Model smo kvantizirali na sva tri ranije opisana načina te evaluirali prateći preciznost te brzinu izvođenja po ispitnom primjeru. Kod evaluacije na grafičkom procesoru, vremena izvršavanja dobivena kod prvih 10 primjera se zanemaruju zbog zagrijavanja GPU-a (*engl. warm-up*, koristi se i termin *cooking*). To je pojava kod koje će izvođenje kod prvih nekoliko primjera trajati duže, dok će se izvođenje ubrzati kada se *pokrenu* sve jezgre procesora jer se očekuje da će izvođenje kod prve aktivacije jezgre biti sporije nego u kasnijem radu.

Također, kako bi ONNX funkcije za primjenu kvantizacije mogle primijeniti kvantizaciju na model, za svaku vrstu kvantizacije je bilo potrebno unijeti manje promjene u osnovni model. Posljedica toga je da će svaka vrsta kvantizacije imati svoju referentnu vrijednost pa ćemo se usredotočiti na ubrzanje i pad preciznosti kvantiziranih modela u postotcima, a ne na apsolutne postignute vrijednosti. Svakako, točne postignute vrijednosti će se razlikovati kod svakog pokretanja modela, ali ne bi smjele previše varirati.

Modeli su razvijani i evaluirani pomoću Google Colaba. Colab je radno okruženje kod kojeg se kod svake sesije (*engl. session*) korisniku dodijeli (ne nužno uvijek isti) grafički procesor. Stoga se performanse postignute u dvije različite sesije mogu razlikovati. Kod sve tri vrste kvantizacije, performanse kvantiziranog modela i odgovarajućeg referentnog modela su dobivene tijekom iste sesije (odnosno, na istom grafičkom procesoru). Ipak, među dvije vrste kvantizacije se nije radilo o istoj sesiji pa je to još jedan razlog zašto rezultate među kvantizacijama ne treba uspoređivati u

kontekstu apsolutnih dobivenih vrijednosti. Nažalost, kvantiziranih modela na grafičkim procesorima nije bilo moguće uz korištenje *ONNX Runtime-a*.

6.1. Dinamička kvantizacija

Model	Točnost	Δ_{acc}	CPU	GPU
Referentni	81.5%		6.68 ms	4.06 ms
Dinamički kvantiziran	80.0%	-1.5 p.b.	5.72 ms	

Tablica 6.1: Usporedba rezultata eksperimenta (dinamička kvantizacija)

Kao što vidimo, kod dinamičke kvantizacije imamo mali pad točnosti modela. Postoji određeno ubrzanje koje je svakako primjetno i čini ovu metodu potencijalnom opcijom za stvarnu primjenu, no kao što ćemo vidjeti, to je ubrzanje manje nego kod drugih vrsta kvantiziranja. Razlog leži u potrebi da se kvantiziranje aktivacija provodi tek kod zaključivanja, što zahtijeva dodatno vrijeme prilikom izvođenja.

6.2. Statička kvantizacija

Model	Točnost	Δ_{acc}	CPU	GPU	JIT
Referentni	81.5%		6.68 ms	4.06 ms	6.05 ms
Statički kvantiziran	81.2%	-0.3 p.b.	4.05 ms		1.27 ms

Tablica 6.2: Usporedba rezultata eksperimenta (statička kvantizacija)

Kod statičke kvantizacije možemo primijetiti zanemarivo manju točnost kvantiziranog modela (-0.3 p.b.) u odnosu na referentni, no s druge strane, ubrzanje je značajno. Treba primijetiti i da je optimizacija provedena prilikom korištenja JIT kompilacije nad kvantiziranim modelom izuzetno učinkovita.

6.3. Naučena kvantizacija

Model	Točnost	Δ_{acc}	CPU	GPU	JIT
Referentni	81.5%		6.68 ms	4.06 ms	6.05 ms
QAT model	80.9%	-0.6 p.b.	4.89 ms		1.31 ms

Tablica 6.3: Usporedba rezultata eksperimenta (naučena kvantizacija)

Kod naučene kvantizacije možemo primijetiti slične rezultate onima dobivenima statičkim kvantiziranjem modela. Smanjenje točnosti je za većinu primjena prihvatljivo (-0.6 p.b.), dok je ubrzanje značajno: 26.79% na CPU-u. Kao i kod statičke kvantizacije, korištenje JIT kompilacije ima velik utjecaj na brzinu zaključivanja modela.

7. Zaključak

Kvantizacija dubokih modela zaista pokazuje karakteristike koje lako mogu učiniti ovu metodu poželjnom i primijenjivom u mnogim slučajevima. Valja primijetiti da je ubrzanje značajno, a pad točnosti sasvim prihvatljiv za veliki broj primjena.

U budućim radovima bilo bi zanimljivo promotriti učinkovitost dinamičke kvantizacije ovisno o broju ciljnih klasa, kao i primijeniti kvantizaciju u drukčijim dubokim modelima, na primjer segmentacijskim modelima. Također, rezultati navode na zaključak da vanjski optimizatori u kombinaciji s kvantizacijom mogu dovesti do iznimnog ubrzanja modela. Bilo bi zanimljivo istražiti koliki je učinak optimizatora kod nekvantiziranih modela te to usporediti s kvantiziranim modelima, pokušavajući utvrditi koliko kvantizacija, a koliko vanjski optimizatori utječu na poboljšanje brzine zaključivanja te kako kvantizacija utječe na rad vanjskih optimizatora.

LITERATURA

[1] What does the term convolution mean? URL https://mriquestions.com/what-is-convolution.html.

[2] ONNX Runtime developers. Onnx runtime. https://onnxruntime.ai/, 2021.

[3] Shuchen Du. Understanding deep self-attention mechanism in convolution neural networks. URL https://medium.com/ai-salon/understanding-deep-self-attention-mechanism-in-convolution-neural-

[4] Grand View Research. Computer vision market size, share trends analysis report by component, by product type, by application (quality assurance inspection, 3d visualization interactive 3d modeling), by vertical, by region, and segment forecasts, 2021 - 2028. 2021. URL https://www.grandviewresearch.com/industry-analysis/computer-vision-market.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

[6] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Nor-

rie, Mark Omernick, Narayana Penukonda, Andy Phelps, i Jonathan Ross. In-datacenter performance analysis of a tensor processing unit. 2017. URL https://arxiv.org/pdf/1704.04760.pdf.

[7] Alex Krizhevsky, Vinod Nair, i Geoffrey Hinton. Cifar-10 and cifar-100 datasets. *URl: https://www. cs. toronto. edu/kriz/cifar. html*, 6(1):1, 2009.

[8] Marko Minđek. Kvantizacija konvolucijskih modela za raspoznavanje slika, 2021. URL http://www.zemris.fer.hr/~ssegvic/project/pubs/mindek21bs.pdf.

[9] Aston Zhang. Padding and stride. URL https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html.

Kvantiziranje dubokih konvolucijskih modela

Sažetak

Duboki konvolucijski modeli danas su metoda izbora za mnoge zadatke računalnog vida. Međutim, njihova računska složenost isključuje mnoge zanimljive primjene u robotici i inteligentnom prometu. Taj problem možemo ublažiti zamjenom decimalnih računskih operacija odgovarajućim cjelobrojnim operacijama. U okviru rada, potrebno je upoznati se s konvolucijskim modelom ResNet-18. Istražiti postojeće pristupe za kvantiziranje težina dubokih modela. Odabrati slobodno dostupni skup slika te oblikovati podskupove za učenje, validaciju i testiranje. Oblikovati algoritme, validirati hiperparametre te uhodati učenje, validiranje i zaključivanje. Modificirati postupak učenja tako da na izlazu dobijemo kvantizirani model. Vrednovati naučene modele, prikazati postignutu točnost te provesti usporedbu s rezultatima iz literature. Komentirati učinkovitost učenja i zaključivanja. Predložiti pravce budućeg razvoja. Radu priložiti izvorni kod razvijenih postupaka uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Ključne riječi: računalni vid, duboko učenje, konvolucijski modeli, kvantizacija, klasifikacija, PyTorch, ResNet-18

Quantization of deep convolutional models

Abstract

Today, deep convolutional models are the method of choice for many tasks in computer vision. However, their computational complexity prevents many interesting applications in robotics and intelligent traffic. We can alleviate that problem by replacing floating-point operations with their corresponding integer counterparts. In the cope of this paper, it is necessary to introduce ourselves with the ResNet-18 convolutional model. Research existing approaches towards quantization of the weights in convolutional models. Select freely available dataset and form subsets for learning, validation and testing. Model the algorithm, validate hyperparameters and set up learning, validation and testing. Modify the learning procedure to get the quantized model as an output. Evaluate learned models, display acquired accuracy and compare the values with literature results. Comment on the efficiency of the learning and inference process. Suggest the directions of futher research. Attach the source code of developed procedure along with necessary explanations and documentation. Cite used literature and state received help.

Keywords: computer vision, deep learning, convolutional models, quantization, classification, PyTorch, ResNet-18