SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 18

UČENJE KONVOLUCIJSKIH MODELA U MIJEŠANOJ TOČNOSTI

Dario Oreč

Zagreb, lipanj 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 18

UČENJE KONVOLUCIJSKIH MODELA U MIJEŠANOJ TOČNOSTI

Dario Oreč

Zagreb, lipanj 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 12. ožujka 2021.

ZAVRŠNI ZADATAK br. 18

Pristupnik:	**Dario Oreč (0036514122)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Siniša Šegvić

Zadatak: **Učenje konvolucijskih modela u miješanoj točnosti**

Opis zadatka:

Raspoznavanje slika važan je zadatak računalnog vida s mnogim zanimljivim primjenama. Trenutno stanje tehnike zasniva se na dubokim modelima koji se uče s kraja na kraj. Nažalost, učenje tih modela zahtijeva velike računske resurse što isključuje neke zanimljive primjene. Taj problem možemo ublažiti izvedbom nekih računskih operacija u 16-bitnoj decimalnoj točnosti. U okviru rada, potrebno je odabrati okvir za automatsku diferencijaciju te upoznati biblioteke za rukovanje matricama i slikama. Proučiti i ukratko opisati postojeće pristupe za klasifikaciju slike. Odabrati slobodno dostupni skup slika te oblikovati podskupove za učenje, validaciju i testiranje. Uhodati postupke učenja modela u standardnoj točnosti i validirati hiperparametre. Predložiti prikladnu metodologiju za izvedbu dijelova postupka u 16-bitnoj točnosti. Procijeniti postignuto smanjenje računskih zahtjeva, vrednovati naučene modele te prikazati i ocijeniti postignutu točnost. Radu priložiti izvorni i izvršni kod razvijenih postupaka, ispitne slijedove i rezultate, uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 11. lipnja 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 18

Učenje konvolucijskih modela u miješanoj točnosti

Dario Oreč

Zagreb, srpanj 2021.

Zahvaljujem mentoru prof. dr. sc. Siniši Šegviću na ukazanom povjerenju i stručnim savjetima. Zahvaljujem svojoj obitelji i prijateljima na bezuvjetnoj podršci.

SADRŽAJ

1. Uvod

Klasifikacija slika je jedan od problema računalnog vida. Zadatak klasifikacije slika je dodijeliti ulaznoj slici jednu oznaku iz fiksnog skupa kategorija. Iako jednostavan, ovaj problem ima široku praktičnu primjenu, jer se mnogi problemi računalnog vida (kao otkrivanje objekata, semantička segmentacija i drugi) mogu svesti na klasifikaciju slika.

Postoje razna rješenja problema klasifikacije slika, no u novije vrijeme se najčešće koriste duboke konvolucijske mreže. Mnoge današnje arhitekture konvolucijskih mreža se sastoje od velikog broja konvolucijskih slojeva, pa zahtijevaju mnogo radne memorije. Postoje mnoga rješenja kako smanjiti memorijski otisak modela bez gubljenja informacije i točnosti. Jedno od tih rješenja, što je i dio ovog rada, jest računanje aktivacija u miješanoj preciznosti. (engl. *mixed precision*).

Tradicionalan način treniranja dubokih neuronskih mreža se oslanja na IEEE format s jednostrukom preciznošću FP32 (engl. *single-precision format*). Kod miješane preciznosti može se trenirati s pola preciznosti FP16 (engl. *half-precision*). Zbog korištenja FP16 miješana preciznost zahtjeva manje memorije za pohranjivanje latentnih aktivacija i samim time može "uklopiti" veće modele u memoriju. Uz to, računanje u FP16 preciznosti ubrzava izvođenje matematički zahtjevnih operacija, poput matričnog množenja i konvoluiranja, što dovodi do ubrzanja procesa treniranja. Nedostatak je što znatan broj grafičkih kartica ne podržava rad s FP16. Miješana preciznost je metoda koja koristi FP16 i FP32 tijekom učenja kako bi ubrzala učenje, te smanjila memorijsko zauzeće modela.

Cilj ovog rada je pobliže pojasniti učenje u miješanoj preciznosti, te ga usporediti s tradicionalnim učenjem u FP32 preciznosti. U drugom poglavlju detaljnije su objašnjeni pojmovi dubokih neuronskih mreža, arhitektura umjetne neuronske mreže, te algoritmi koji se koriste za vrijeme treniranja. Treće poglavlje detaljnije opisuje rad miješane preciznosti, dok se četvrto poglavlje bavi detaljima implementacije i načinom korištenja miješane preciznosti. U zadnjem su opisani i uspoređeni rezultati tradicionalnog i miješanog načina treniranja.

2. Umjetne neuronske mreže

2.1. Motivacija

Umjetne neuronske mreže izražavamo u okviru teorije strojnog učenja. Svaki algoritam strojnog učenja sastoji se od tri glavne komponente. Te komponente su: model, funkcija gubitka i optimizacijski postupak.

Model predstavlja skup hipoteza, gdje su hipoteze funkcije koje primjerima iz X daje oznaku iz Y. Dakle model je skup funkcija parametriziranih s θ, gdje θ označava vektor preko kojeg možemo dobiti točno jednu hipotezu iz skupa hipoteza. Umjetne neuronske mreže su vrsta algoritma strojnog učenja gdje je model proizvoljna nelinearna funkcija s velikim brojem parametara. Ta proizvoljna nelinearna funkcija treba biti jednom diferencijabilna.

Funkcija gubitka je vrijednost pogreške načinjene na jednom primjeru. Drugim riječima, funkcija gubtika govori koliko je model izgubio na točnosti na tom primjeru. Što je funkcija gubitka manja to model manje gubi na točnosti. Ako je gubitak nula tada model ispravno klasificira primjer. Za nadzirano učenje (engl. *supervised learning*) klasifikacijskih neuronskih mreža tipično koristimo funkciju gubitka unakrsne entropije (engl. *cross-entropy loss*).

Optimizacijski postupak je postupak, odnosno funkcija, koja nalazi parametre koji minimiziraju funkciju gubitka. Optimizacijski postupak kod umjetni neuronskih mreža gotovo je uvijek neka varijanta SGD-a (gradijentni spust).

2.2. Struktura

Umjetne neuronske mreže se često grade pomoću kompozicije slojeva. Slojeve često izražavamo parametriziranim linearnim preslikavanjem, poput matričnog množenja i konvolucija. Nakon slojeva postavljaju se nelinearne prijenosne funkcije bez parametara. Ulaz u sloj modeliran je varijablama x_1, x_2,...,x_n koje predstavljaju prijenos podataka iz prethodnog sloja. Kako bi modelirali utjecaj ulaza na sloj, za svaki x_i postoji varijabla w_i koju zovemo težina (engl. *weight*). Umnožak $w_i \cdot x_i$ predstavlja utjecaj varijable x_i. Također se definira i pomak *b* (engl. *bias*) (još označavan s w_0). Pomak se koristi kao dodatni parametar mreže kako bi pomogao da bolje odgovara danim podacima. Sve navedene vrijednosti sumiramo prema izrazu:

$$net = \sum_{i=1}^{n} w_i \cdot x_i + b. \tag{2.1}$$

Jedan izlaz sloja dobivamo tako što sumu *net* provučemo kroz prijenosnu funkciju *f* danu izrazom:

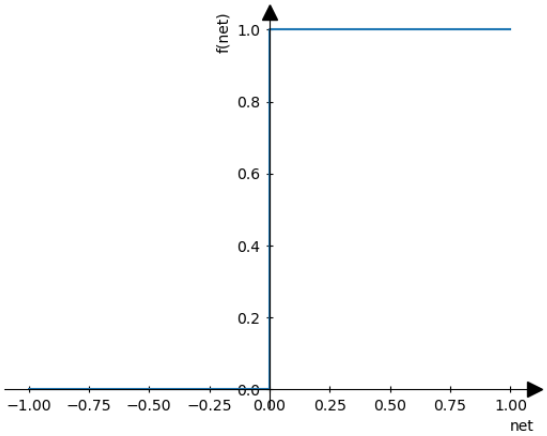
$$y = f(net). \tag{2.2}$$

2.2.1. Prijenosne funkcije

Prijenosne funkcije ili aktivacijske funkcije odlučuju treba li neuron biti "aktiviran", te uvode nelinearnost u model. Najjednostavniji oblik prijenosne funkcije je takozvana step funkcija ili funkcija skoka. Definira se izrazom:

$$f(net) = \begin{cases} 0, & \textit{net} < 0 \\ 1, & \textit{net} \geq 0 \end{cases} \tag{2.3}$$

Problem step funkcije je što ima prekid u točki 0, te ju nije moguće koristiti za postupke učenja temeljene na derivacijama. Step funkcija prikazana je na slici 2.1

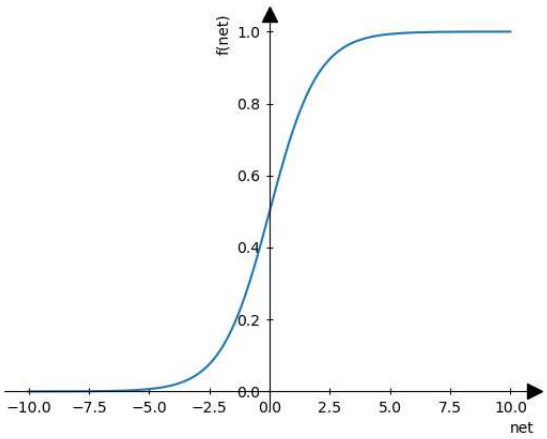


Slika 2.1: Prijenosna funkcija skoka. Ova funkcija poprima vrijednosti 0 ili 1.

Sljedeća važna prijenosna funkcija je sigmoidalna prijenosna funkcija dana izrazom:

$$\sigma(net) = \frac{1}{1 + e^{-net}} \tag{2.4}$$

Ova funkcija je poopćenje funkcije skoka. Kodomena ove funkcije je [0,1]. Prednost je što je derivabilna, pa time omogućava učenje s postupcima temeljenim na gradijentom spustu. Nedostatak je takozvani problem nestajanja gradijenta (engl. *vanishing gradient problem*) kod kojeg gradijenti postaju previše mali da bi utjecali na daljnje učenje mreže, što možemo primijetiti i na slici 2.2.



Slika 2.2: Sigmoidalna prijenosna funkcija. Ova funkcija je derivabilna, pa se može koristiti uz varijante optimizacijskog postupka SGD.

Još jedno lijepo svojstvo ove funkcije je njezina derivacija koju možemo izraziti pomoću same funkcije:

$$\frac{d\sigma(net)}{dnet} = \sigma(net) \cdot (1 - \sigma(net)) \tag{2.5}$$

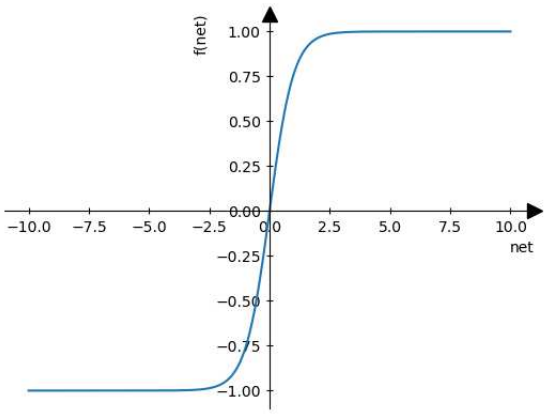
Prijenosnu funkciju tangens hiperbolni možemo dobiti pomoću sigmoidalne funkcije, kako je prikazano u izrazu:

$$tanh(net) = 2 \cdot \sigma(2 \cdot net) - 1 \tag{2.6}$$

Zbog ovoga i nju možemo smatrati poopćenjem funkcije skoka. Ova funkcija ima kodomenu [-1,1].
Kao i kod sigmoide, prednost je što je derivabilna i derivacija se može izraziti pomoću same funkcije prema izrazu:

$$\frac{dtanh(net)}{dnet} = 1 - tanh^2(net) \tag{2.7}$$

Nedostatak, kao i kod sigmoide, je problem nestajanja gradijenta. Ova funkcija je prikazana na slici 2.3.

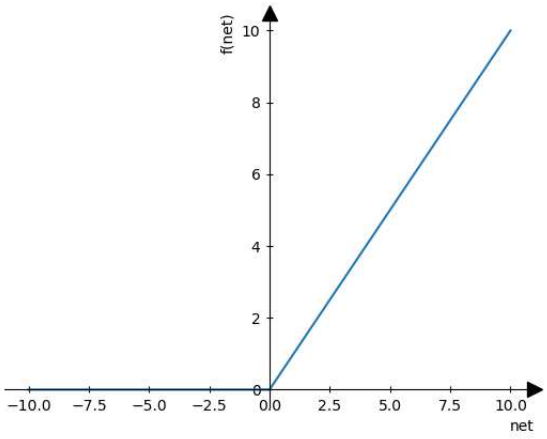


Slika 2.3: Tangens hiperbolni. Kao i sigmoida, tangens hiperbolni je derivabilan, pa se koristi uz varijante SGD-a.

Još jedna prijenosna funkcija koja se koristi za aktiviranje latentnih reprezentacija naziva se zglobnica (engl. *ReLU - rectified linear unit*). Latentne aktivacije su aktivacije modela na koje gubitak ne djeluje izravno. Karakteristika ove funkcije je da sve pozitivne vrijednosti propušta bez prigušenja, a negativne vrijednosti ne propušta. Zglobnica je dana izrazom:

$$relu(net) = max(0, net) \tag{2.8}$$

Ova funkcija je prikazana na slici 2.4.

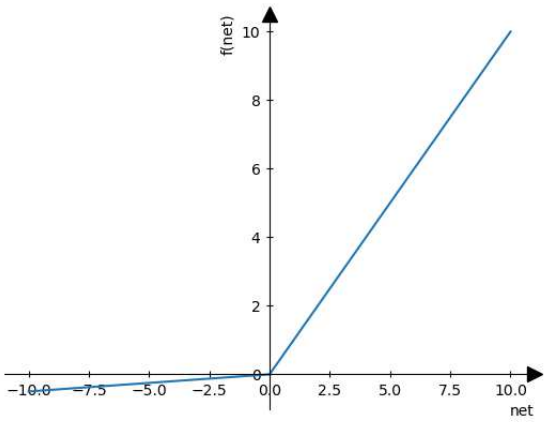


Slika 2.4: Zglobnica. Ova prijenosna funkcija se koristi za aktiviranje latentnih reprezentacija.

Derivacija ove funkcije daje funkciju skoka. Nedostatak je što za sve negativne vrijednosti funkcija daje 0. Napravljena je i modifikacija zglobnice poznata kao propusna zglobnica (engl. *LReLU - leaky rectified linear unit*) dana izrazom:

$$lrelu(net) = \begin{cases} net, & net > 0 \\ \alpha \cdot net, & net \leq 0 \end{cases} \tag{2.9}$$

Gdje α predstavlja parametar funkcije (mali pozitivan broj). Karakteristika ove funkcije je da sve pozitivne vrijednosti propušta bez prigušenja, a negativne vrijednosti propušta uz prigušenje, što se vidi iz slike 2.5. Propusna zglobnica se koristi u modelima i slojevima koji trebaju biti invertibilni.



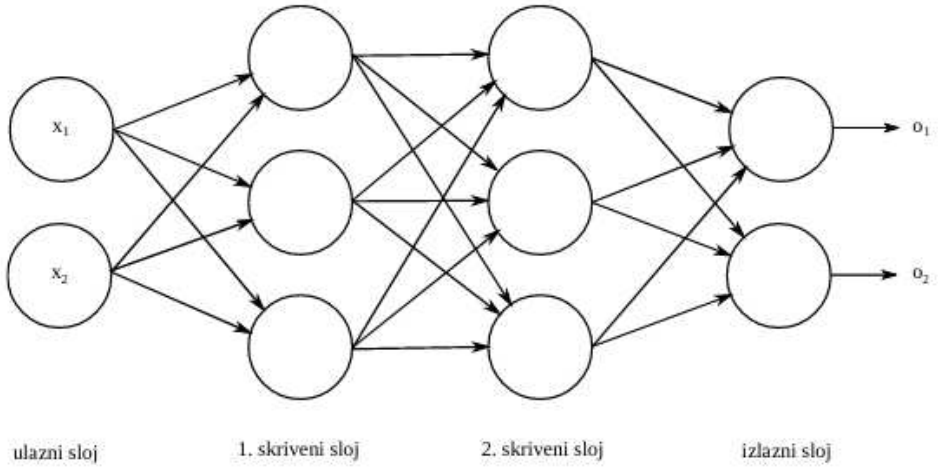
Slika 2.5: Propusna zglobnica. Koristi se u modelima i slojevima koji trebaju biti invertibilni.

Zbog ovog lijepog svojstva derivacija je posvuda različita od nule:

$$\frac{dlrelu(net)}{dnet} = \begin{cases} 1, & net > 0 \\ \alpha, & net \leq 0 \end{cases} \tag{2.10}$$

2.3. Arhitektura umjetne neuronske mreže

Neuronske mreže su složene strukture građene od velikog broja slojeva koje mogu za više ulaza proizvesti jedan izlaz. Obično se sastoji od ulaznog i izlaznog sloja među kojima se nalazi jedan ili više skrivenih slojeva (slika 2.6).



Slika 2.6: Arhitektura neuronske mreže s dva skrivena sloja. Slika preuzeta iz [9].

Ako svaki neuron prethodnog sloja utječe na neuron trenutnog sloja, tada govorimo

o potpuno povezanim slojevima (slika 2.6).
Svrha ulaznog sloja je da prima podatke iz vanjskog izvora, te na temelju tih podataka svaki neuron ovog sloja generira sumu *net* te je šalje sljedećem sloju mreže.
Skriveni slojevi su slojevi koji su nevidljivi (skriveni) vanjskom svijetu. Glavni dio izračuna neuronske mreže se odvija u ovim slojevima. Skriveni slojevi uzimaju informaciju iz ulaznog sloja, te izvode potreban broj izračuna za generiranje rezultata. Na kraju se taj rezultat šalje izlaznom sloju koji omogućava pregled rezultata izračuna.

2.4. Konvolucijske neuronske mreže

Konvolucijska neuronska mreža (engl. *Convolutional Neural Network*) je vrsta modela dubokog učenja koja obrađuje podatke rešetkaste strukture. Rešetkasta struktura može biti 1D (jezik, bioinformatika), 2D (slike) ili 3D (rekonstrukcija dubine, CT, ili MRI). Neuronska mreža s 2D konvolucijama prilagođava strukturu modela činjenici da se na ulaz modela dovodi slika.

Kod potpuno povezanih mreža svaki skriveni sloj sadrži skup neurona gdje je svaki neuron povezan sa neuronom iz prijašnjeg sloja. Ako se na ulaz ovakve arhitekture dovede slika dimenzije *H* x *W* x *D*, gdje *H* predstavlja visinu slike, *W* širinu slike i *D* dubinu slike, tada će svaki neuron prvog skrivenog sloja imati $H \cdot W \cdot D$ težina. Ovime potpuna povezanost brzo raste i zahtjeva veliki broj parametara.

Za razliku od potpuno povezanih neuronskih mreža, neuroni kod konvolucijske neuronske mreže bit će povezani s malim brojem neurona prethodnog sloja. Zbog ovoga konvolucijske mreže su prorijeđene i zahtijevaju manji broj parametara. Za izgradnju konvolucijskih mreža koriste se četiri glavna sloja: **konvolucijski sloj**, **sloj sažimanja**, **potpuno povezani sloj** i **batchnorm**.

Batchnorm je tehnika koja se koristi pri treniranju dubokih modela. Cilj ove tehnike je normalizirati ulaze u slojeve mreže za svaku grupu učenja (engl. *mini-batch*). Dodatno batchnorm omogućava korištenje veće stope učenja, te pruža veću slobodu prilikom inicijalizacije težina.

2.4.1. Konvolucijski sloj

Konvolucijski sloj je temeljni građevni blok konvolucijske mreže koji obavlja većinu izračuna. Parametri ovog sloja se nazivaju filtri ili konvolucijske jezgre koji se tijekom treniranja uče. Filtri su prostorno dosta mali. Tijekom unaprijednog prolaza svaki filtar provlačimo po dužini i širini ulaza i računamo skalarni umnožak između filtra i ulaza

na svakom položaju. Rezultat ovih izračuna naziva se aktivacijska mapa (engl. *activation*). Na slici 2.7 ulaz predstavlja matrica dimenzija 3x3, dok je filtar dimenzije 2x2. Neka je korak filtra 1, tada se filtar provlači preko matrice ulaza, za svako pomicanje filtra računa se skalarni umnožak, te se na izlazu dobije aktivacijska mapa dimenzija 2x2.

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} * \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} = \begin{bmatrix} x_{11}w_{11} + x_{12}w_{12} & x_{12}w_{11} + x_{13}w_{12} \\ + x_{21}w_{21} + x_{22}w_{22} & + x_{22}w_{21} + x_{23}w_{22} \\ x_{21}w_{11} + x_{22}w_{12} & x_{22}w_{11} + x_{23}w_{12} \\ + x_{31}w_{21} + x_{32}w_{22} & + x_{32}w_{21} + x_{33}w_{22} \end{bmatrix}$$

Slika 2.7: Prikaz dvodimenzionalne konvolucije ulaza X s filtrom W bez korištenja nadopune (engl. *padding*).

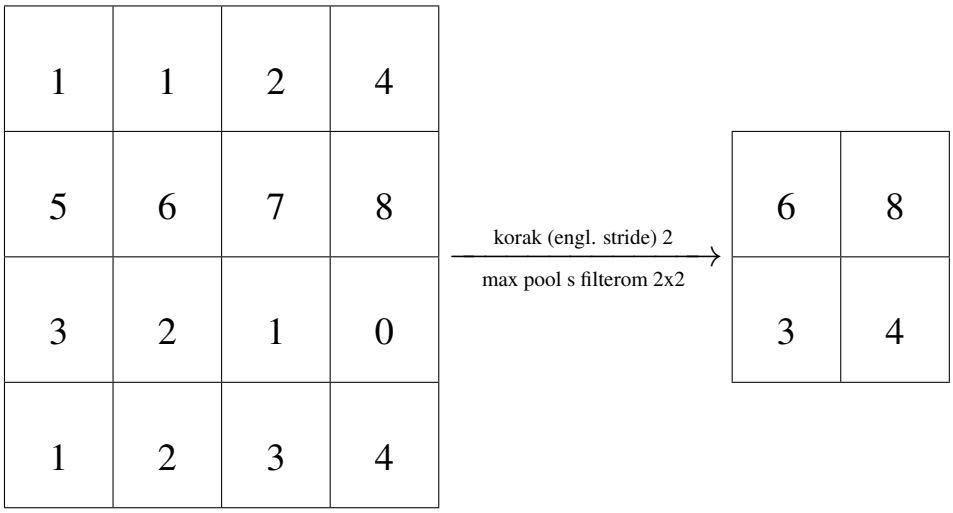
Pošto se u konvolucijskom sloju koristi više filtara, svaki od njih će proizvoditi različitu dvodimenzionalnu aktivacijsku mapu. Slaganjem ovako dobivenih dvodimenzionalnih aktivacijskih mapa po dimenziji dubine dobivamo izlaz konvolucijskog sloja.

Veličina izlaza konvolucijskog sloja ovisi o: dubini (engl. *depth*), koraku (engl. *stride*) i nadopuni (engl. *padding*). Dubina predstavlja broj različitih filtara koje želimo koristiti u sloju. Korak pokazuje za koliko se piksela filtar pomiče po ulazu. Nadopuna služi kako bi se sačuvale ulazne dimenzije nakon primjene konvolucije. Najčešće korištena nadopuna je nadopuna nulama (engl. *zero-padding*).

2.4.2. Sloj sažimanja

Sloj sažimanja (engl. *pooling layer*) je sloj koji se najčešće nalazi između uzastopnih konvolucijskih slojeva. Zadaća sloja sažimanja je smanjiti prostornu veličinu ulaza kako bi se smanjio broj parametara i izračuna u mreži i povećalo receptivno polje aktivacija. Važno je spomenuti da ovaj sloj ne uvodi dodatne parametre.

Postoji više vrsta sloja sažimanja, od kojih se često koristi sloj sažimanja najvećom vrijednošću (engl. *max-pooling*). Max pooling sloj uzima maksimalnu vrijednost aktivacijske mape na području koje je prekriveno filtrom (slika 2.8) .



Slika 2.8: Primjer sažimanja maksimalnom vrijednošću s jezgrom veličine 2x2 i korakom 2.

2.5. Učenje umjetne neuronske mreže

Učenje umjetne neuronske mreže predstavlja postupak u kojem se mreži predočavaju podatci iz skupa podataka za učenje. Na temelju tih podataka mreža ažurira svoje težine kako bi što bolje aproksimirala izlaznu funkciju na podacima koje nije još vidjela. Kako bismo utvrdili koliko dobro mreža aproksimira uvodimo pojam funkcije gubitka. Minimiziranjem funkcije gubitka mreža bolje aproksimira i samim time uči. Ako mreža na podacima za učenje ima minimalan gubitak, a na podacima koje nije vidjela gubitak je još izražen, tada kažemo da je mreža prenaučena tj. izgubila je sposobnost generalizacije.

Postoji više načina učenja neuronske mreže, a najčešći je nadzirano učenje (engl. *supervised learning*). Kod ovog učenja mreži se predaju parovi oblika *(ulaz, željeni izlaz)*. Cilj ovog učenja je postići da izlazi mreže što bolje odgovaraju željenim izlazima.

2.5.1. Funkcija gubitka

Funkcija gubitka (engl. *loss function*) je način na koji se može utvrditi koliko dobro neuronska mreža računa rezultate. Što mreža lošije predviđa funkcija gubitka je veća i obratno. Kako bi pomoću funkcije gubitka mogli ostvariti učenje modela, definiramo je tako da ovisi o težinama **W** i pomacima **b**.

Postoje različite funkcije gubitka, no kod problema klasifikacije slika najčešće se koristi unakrsna entropija (engl. *cross-entropy loss*) definirana izrazom:

$$L_{CE} = -\sum_{i=1}^{n} t_i \cdot log(p_i) \tag{2.11}$$

gdje t_i predstavlja željeni izlaz, p_i vjerojatnost i-te klase, a *n* sveukupni broj klasa.

2.5.2. Regularizacija

Brojne tehnike strojnog učenja su napravljene da bi smanjile grešku na testnom skupu podataka, uz posljedicu povećanja greške pri treniranju. Ove tehnike se još nazivaju i regularizacijom (engl. *regularization*). Drugim riječima, regularizacija je svaka izmjena na algoritmu učenja koja služi smanjenju pogreške na neviđenim podacima, ali ne nužno i smanjenju pogreške tijekom učenja.

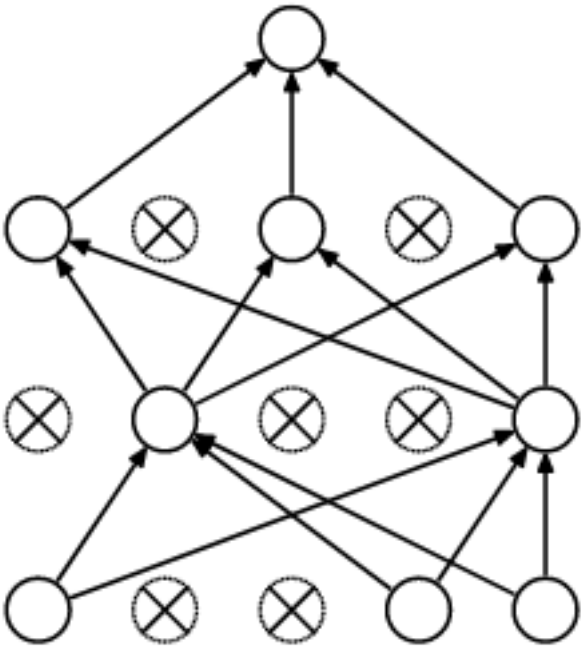
Jedna od metoda regularizacije je dodavanje regularizacijskog faktora funkciji gubitka. Najpoznatije su L^1 i L^2 regularizacija, koje kažnjavanju normu vektora težina. L^1 regularizacija definirana je izrazom 2.12, a L^2 izrazom 2.13.

$$L_R = L + \lambda \cdot \sum_i |w_i| \tag{2.12}$$

$$L_R = L + \lambda \cdot \sum_i w_i^2 \tag{2.13}$$

Jedan od najboljih načina da mreža bolje generalizira je da uči na velikim skupovima podataka. No u praksi količina podataka je ograničena. Jedan način da se izbjegne ovakav problem je da se umjetno poveća broj podataka. Ova metoda se naziva uvećanje skupa podataka (engl. *data augmentation*). Ovakav pristup se pokazao jako učinkovit za probleme klasifikacije. Povećanje podataka se može postići korištenjem raznih transformacija na početnom skupu podataka. Neke od transformacija su: zrcaljenje, rotacija, nasumično rezanje dijelova slike, dodavanje šuma i mnoge druge.

Još jedna metoda regularizacije koja se danas sve više koristi naziva se dropout. Droput tijekom treniranja "ugasi" neke neurone, tj. zanemari njihov utjecaj u mreži (slika 2.9). Prednosti droputa je što pomaže u tome da mreža ne postane previše ovisna o pojedinim neuronima.



Slika 2.9: Primjer korištenja dropouta. Prekriženi neuroni predstavljaju "ugašene" neurone čiji se utjecaj u mreži zanemaruje. Slika preuzeta iz [6].

2.5.3. Gradijentni spust

Svi algoritmi strojnog učenja koriste neku vrstu optimizacije, gdje optimizacija predstavlja pronalazak minimuma funkcije gubitka *f(**x**)*. Funkcija gubitka je funkcija koju želimo minimizirati. Jedna tehnika minimiziranja funkcije gubitka naziva se gradijentni spust (engl. *gradient descent*).

Neka imamo funkciju *y = f(x)*, gdje su *x* i *y* realni brojevi. Označimo derivaciju ove funkcije s $f'(x)$, tada derivacija $f'(x)$ daje nagib funkcije u točki *x*. Drugim riječima, ako napišemo:

$$f(x + \epsilon) \approx f(x) + \epsilon \cdot f'(x) \tag{2.14}$$

tada izraz 2.14 govori koliko mala promjena ulaza utječe na promjenu izlaza. Prema tome, derivacija je korisna za pronalazak minimuma jer govori koliko trebamo promijeniti *x* da bi dobili malu promjenu u *y*.

Ako u izraz 2.14 uvrstimo pomak u smjeru negativne derivacije dobijemo:

$$f(x - \epsilon \cdot sign(f(x))) \approx f(x) - f'(x) \cdot \epsilon \cdot sign(f'(x)) \tag{2.15}$$

Vidimo da je izraz *f (x - ϵ · sign($f'(x)$))* manji od *f(x)* za dovoljno mali ϵ. Prema tome možemo smanjivati *f(x)* tako što pomičemo *x* malim koracima u smjeru negativne

derivacije i time možemo doći do minimuma funkcije.

Neka je *L* funkcija gubitka koja ovisi o težinama **W** i pomacima **b**. Tada gradijente funkcije gubitka možemo napisati:

$$\nabla L_w = \left(\frac{\partial L}{\partial w_1}, ..., \frac{\partial L}{\partial w_n} \right), \tag{2.16}$$

$$\nabla L_b = \left(\frac{\partial L}{\partial b_1}, ..., \frac{\partial L}{\partial b_n} \right), \tag{2.17}$$

gdje *n* predstavlja broj parametara. Svakom novom iteracijom gradijentnog spusta parametri će se ažurirati prema izrazima:

$$W_{i+1} = W_i - \epsilon \cdot \nabla L_w(\mathbf{W}, \mathbf{b}), \tag{2.18}$$

$$b_{i+1} = b_i - \epsilon \cdot \nabla L_b(\mathbf{W}, \mathbf{b}) \tag{2.19}$$

gdje ϵ predstavlja mali pozitivan realni broj koji se naziva stopa učenja (engl. *learning rate*).

Postoje različite varijante gradijentnog spusta. Neke od poznatijih su stohastički gradijentni spust i učenje sa zaletom (ili učenje momentom gradijenta)[2].

Stohastički gradijentni spust je algoritam koji se ne primjenjuje na cijeli skup podataka nego samo na manju grupu podataka (engl. *batch*). Bitno je naglasiti da se manja grupa podataka za svaku iteraciju uzima nasumično. Važno svojstvo stohastičkog gradijentnog spusta je da vrijeme izračuna za svaku iteraciju ne raste, bez obzira na veličinu podataka za učenje.

Učenje sa zaletom je metoda kojoj je glavni cilj ubrzati učenje. Za razliku od stohastičkog gradijentog spusta, momentum nastoji smanjiti oscilacije u različitim iteracijama postupka. Učenje sa zaletom definira novi vektor koji predstavlja brzinu kretanja prema minimumu. Označimo taj vektor sa **v**. Vektor **v** ažuriramo prema sljedećem izrazu:

$$v_{i+1} = \alpha v_i - \epsilon \cdot \nabla L, \tag{2.20}$$

gdje je α mali pozitivan realan broj koji predstavlja prigušenje koje omogućuje zaustavljane u minimumu. Sada težine **W** možemo ažurirati na sljedeći način:

$$W_{i+1} = W_i + v_{i+1} \tag{2.21}$$

2.5.4. Algoritam propagacije pogreške unatrag

Propagacija pogreške unazad (engl. *back-propagation*) je metoda za izračun gradijenata, dok se algoritmi objašnjeni u dijelu 2.5.3 (poput stohastičkog gradijentnog

spusta) koriste za učenje parametara mreže koristeći izračunate gradijente. Računanje gradijenata se odvija rekurzivnom primjenom pravila ulančavanja. Kod učenja modela korištenjem gradijentnog spusta, algoritam propagacije pogreške unatrag računa parcijalne derivacije funkcije gubitka *L* s obzirom na težine **W** i pomake **b**.

Neka je w_{ij}^l težina za *j*-ti neuron sloja *l*-1 koja povezuje *i*-ti neuron *l*-tog sloja, a b_i^l označava pomak za *i*-ti neuron *l*-tog sloja. Također, definirajmo izlaz iz neurona prije i poslije primjene aktivacijske funkcije. Neka je a_i^l izlaz *i*-tog neurona *l*-tog sloja bez primjene aktivacijske funkcije, a o_i^k izlaz *i*-tog neurona *l*-tog sloja s primjenom aktivacijske funkcije. Kako bismo olakšali izračun, neka se pomak b_i^l uključi u težine kao w_{0i}^l sa fiksiranim izlazom $o_0^{l-1} = 1$. Prema tome izlaz neurona prije primjene aktivacijske funkcije možemo napisati prema sljedećem izrazu:

$$a_i^l = b_i^l + \sum_{k=1}^{r_{l-1}} w_{ki}^l \cdot o_j^{l-1} = \sum_{k=0}^{r_{l-1}} w_{ki}^l \cdot o_j^{l-1} \tag{2.22}$$

gdje r_{l-1} predstavlja broj neurona *l*-tog neurona.

Sada koristeći algoritam propagacije pogreške unatrag, možemo računati parcijalne derivacije funkcije gubitka primjenom pravila ulančavanja:

$$\frac{\partial L}{\partial w_{ij}^l} = \frac{\partial L}{\partial a_i^l} \cdot \frac{\partial a_i^l}{\partial w_{ij}^l} \tag{2.23}$$

Prvi dio umnoška izraza 2.23 se često naziva i greškom neurona:

$$\delta_i^l = \frac{\partial L}{\partial a_i^l}. \tag{2.24}$$

Drugi dio umnoška izraza 2.23 možemo napisati kao:

$$\frac{\partial a_j^l}{\partial w_{ij}^l} = \frac{\partial}{w_{ij}^l} \cdot \left(\sum_{k=0}^{r_{l-1}} w_{kj}^l \cdot o_k^{l-1} \right) = o_i^{l-1} \tag{2.25}$$

Koristeći 2.24 i 2.25 izraz 2.23 možemo čitljivije zapisati:

$$\frac{\partial L}{\partial w_{ij}^l} = \delta_j^l \cdot o_i^{l-1} \tag{2.26}$$

Nadalje, ako iskoristimo da je $b_i^l = w_{0i}^l$, te da vrijedi $o_0^{l-1} = 1$, tada uvrštavanjem ovih vrijednosti u izraz 2.26 dobivamo:

$$\frac{\partial L}{\partial b_i^l} = \delta_i^l \tag{2.27}$$

Izrazima 2.26 i 2.27 računa se ovisnost funkcije gubitka o svakom parametru, te se s njima ažuriraju parametri tijekom postupka gradijentnog spusta.

3. Miješana preciznost

3.1. Uvod

Treniranje dubokih neuronskih mreža se tradicionalno odvija predstavljanjem aktivacija u skladu s IEEE formatom s jednostrukom preciznošću FP32 (engl. *single-precision format*). Međutim, kako se povećanjem veličine neuronske mreže obično dobiva bolja točnost, tradicionalno treniranje postaje problematično, jer zahtijeva previše memorije. Stoga se kao jedno riješenje ovog problema razvila i miješana preciznost (engl. *mixed-precision*).

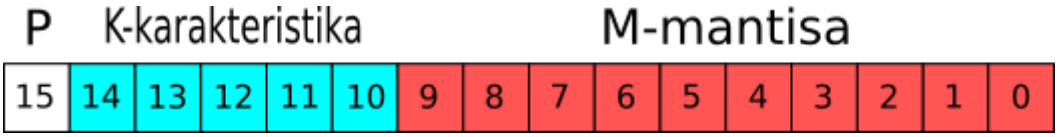
Kako se povećanjem veličine mreže povećava memorijski otisak i izračuni tijekom treniranja, miješana preciznost koristi format s pola preciznosti FP16 (engl. *half-precision format*) gdje god je to moguće. Prednost ove metode je znatno ubrzanje izračuna tijekom učenja provođenjem matematičkih operacija u FP16 formatu. Također, pokazuje se da učenje u miješanoj preciznosti često ne gubi točnost, niti zahtijeva modificiranje hiperparametara, te uvijek smanjuje memorijski otisak modela tijekom učenja.

Iako korištenje FP16 formata smanjuje memorijsko zauzeće, skraćuje vrijeme treniranja i zaključivanja, ipak ima problem. Naime, FP16 ima uži dinamički raspon od FP32, što onemogućava spremanje vrlo malih i vrlo velikih vrijednosti. Kako bi se spriječio ovaj nedostatak uvode se tri tehnike: održavanje glavne kopije težina u FP32, skaliranje gubitka (engl. *loss-scaling*) i FP16 računanje s akumulacijom u FP32[8].

3.2. FP16

IEEE 754 standard definira FP16 kao 16-bitni format s pomičnim zapisom: 1 bit predznaka, 5 bitova karakteristike (eksponenta) i 10 bitova mantise kao što je vidljivo na slici 3.1.

Većina brojeva u ovom formatu je "normalizirano". Ovo znači da je prvi bit man-



Slika 3.1: FP16 prikaz brojeva s posmičnim zarezom. P predstavlja bit predznaka.

tise jednak 1 i ne sprema se direktno (implicitni bit). Isto tako za broj kažemo da je normaliziran ako bitovi karakteristike nisu sve nule ili jedinice. Prema tome, normalizirani brojevi imaju raspon od približno 6.10^{-5} do 65504. Drugi način zapisa brojeva su takozvani subnormali (engl. *subnormals, denormal*). Ovo je poseban zapis gdje su sve vrijednosti karakteristike nula. Minimalna vrijednost ovog zapisa je približno jednaka 5.96^{-8}.

Kao što je već spomenuto, FP16 se koristi kod treniranja dubokih modela, koristeći metodu miješane preciznosti. Nedostatak ovog zapisa je što ga velik broj grafički kartica ne podržava. No u novije vrijeme postaje sve više prisutan, pa moderne grafičke kartice počinju podržavati FP16 (posebice NVIDIA grafiče kartice koje koriste Tensor Cores[7]).

3.3. Održavanje glavne kopije težina u FP32

Tijekom treniranja, težine, aktivacije i gradijenti se čuvaju u FP16 formatu. Kod unaprijednog i unazadnog prolaza koristi se FP16 kopija težina koja zahtjeva upola manje memorije u odnosu na FP32. Ipak, kako bi se zadržala točnost modela kao kod FP32 formata čuva se kopija težina u FP32. Čuvanje kopija težina u FP32 se pokazalo kao dobar pristup, te se u praksi i koristi.

Jedan od razloga zašto je ova tehnika potrebna je što tijekom ažuriranja težina, gradijent pomnožen sa stopom učenja može postati jako mali broj kojega FP16 ne može prikazati. Ove male vrijednosti bi u FP16 formatu postale nula i time narušile ažuriranje težina, a samim time i točnost modela. Čuvanje težina za ažuriranje u FP32 zaobilazi ovaj problem, pa se točnost ne mijenja.

Drugi razlog za kojim se javlja potreba ove tehnike proizlazi iz samog zapisa broja u FP16 (slika 3.1). Naime, omjer vrijednosti težine i vrijednosti za koju se težina treba ažurirati može biti jako velik. Iako se vrijednost ažuriranja može prikazati u FP16, kada se njegova vrijednost doda vrijednosti težine može se dogoditi da se vrijednost težine ne promijeni, ako je vrijednost težine 2048 puta veća od vrijednosti ažuriranja težine. Budući da FP16 ima 10 bitova mantise, implicitni bit se mora pomaknuti

udesno za barem 11 mjesta da bi se stvorila nula. Stoga, kada je omjer veći od 2048, implicitni bit se pomiče za 12 ili više mjesta i zbog toga se, iako nije, vrijednost ažuriranja težine ponaša kao nula. Kao i prije ovaj problem se zaobilazi korištenjem kopija težina u FP32.

Ovaj problem možemo vidjeti u sljedećem kratkom primjeru:

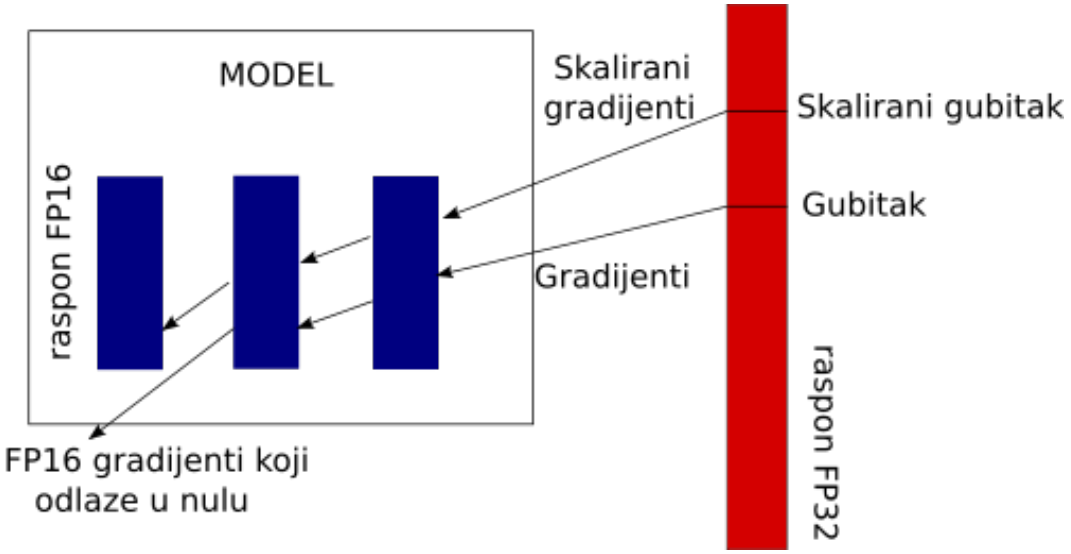
```
1  import torch
2
3  #tensor([1.], dtype=torch.float16)
4  param = torch.HalfTensor([1.0])
5
6  #tensor([0.0001], dtype=torch.float16)
7  update = torch.HalfTensor([.0001])
8
9  #tensor([1.], dtype=torch.float16)
10 print(param + update)
```

Programski kod 3.1: Problem koji se dogodi kod pribrajanja male vrijednost u FP16.

Na prvu se čini da korištenjem ove tehnike tijekom treniranja samo povećavamo memorijsko zauzeće mreže. Međutim, najveći dio memorije zauzimaju aktivacije čije su vrijednosti potrebne za vrijeme *back-propagation* algoritma. Pošto se aktivacije spremaju kao FP16 sveukupno zauzeće memorije se osjetno smanji.

3.4. Skaliranje gubitka

Kako je karakteristika FP16 za normalizirane vrijednosti u rasponu [-14, 15], često se dogodi da gradijenti za vrijeme treniranja budu manji od minimalne vrijednosti FP16 formata. Jedna mogućnost izbjegavanja ovog problema je pomicanje vrijednosti gradijenta u raspon FP16. To se postiže skaliranjem gubitka izračunatog u unaprijednom prolazu, prije početka algoritma unazadnog prolaza (slika 3.2). Zbog pravila ulančavanja tijekom unazadnog prolaza, osigurano je da se svi gradijenti skaliraju sa istim faktorom skaliranja.



Slika 3.2: Utjecaj skaliranja gradijenata tijekom učenja.

Gradijenti se prije ažuriranja moraju "odskalirati" (engl. *unscale*) kako bi se održale veličine ažuriranja kao i kod FP32. To je najbolje uraditi nakon unatražnog prolaza, ali prije bilo kakvog izračuna s gradijentom, da se ne bi morali prilagođavati hiperparametri.

3.4.1. Odabir faktora skaliranja

Da bi skaliranje gubitka imalo smisla, potrebno je odabrati smislen faktor skaliranja. Najjednostavniji pristup je odabrati neki fiksni faktor, takav da umnožak s najvećom vrijednošću gradijenta bude manja od 65,504 (maksimalna vrijednost u FP16). Odabir velikog faktora skaliranja nema posljedica, sve dok se ne događa "preljev" (engl. *overflow*) tijekom unatražnog prolaska. Ako se ipak dogodi "preljev", gradijenti će sadržavati vrijednost Inf (infinity) ili NaN (Not a Number), što će naškoditi učenju modela. Jednostavan način zaobilaska "preljeva" je preskočiti ažuriranje težina i nastaviti na sljedeću iteraciju[7].

Drugi način odabira faktora je dinamički. To znači da se započne sa nekim velikim faktorom skaliranja i nakon svake iteracije odluči treba li ga mijenjati. Odlučuje se na temelju "preljeva". Ako se "preljev" nije dogodio u N iteracija, gdje N predstavlja broj iteracija bez "preljeva", tada treba povećati faktor za neku fiksnu vrijednost c_i. Ako se "preljev" dogodio, preskoči ažuriranje težina i smanji faktor za fiksni c_d.

3.5. FP16 računanje s akumulacijom u FP32

Veliki dio izračuna neuronskih mreža otpada na skalarni umnožak vektora. Neke mreže, kako bi zadržale točnost kao i FP32 mreže, zahtijevaju da skalarni umnožak vektora akumulira djelomične umnoške u FP32, koji se prije pisanja u memoriju ponovo pretvori u FP16. Kako su prethodni modeli grafički kartica podžavali samo operacije unutar FP16 formata, NVIDIA uvodi takozvane Tensor Cores koji omogućavaju da se rezultat operacija množenja i zbrajanja u FP16 zapiše u FP16 ili FP32.

Tensor Cores su programibilne jedinice za množenje i zbrajanje matrica koje omogućavaju da se rezultati operacije u FP16 automatski zapišu u FP32 bez smanjenja točnosti modela.

3.6. Radni okvir Pytorch i miješana preciznost

Radni okvir Pytorch pruža podršku za rad s miješanom preciznosti koristeći paket torch.cuda.amp (*Automatic Mixed Precision*). Torch.cuda.amp pruža korištenje miješane preciznosti, gdje se neke operacije obavljaju u FP32, a neke u FP16. Amp pokušava koristiti odgovarajuće tipove podataka na mjestima gdje je njihova upotreba optimalna.

Ovaj paket omogućava korištenje torch.cuda.amp.autocast i torch.cuda.amp.GradScaler za treniranje u miješanoj preciznosti. Torch.cuda.amp.autocast služi kako bi predviđeni dio programa koristio miješanu preciznost. Torch.cuda.amp.GradScaler se koristi kao skaliranje gubitka objašnjeno u poglavlju 3.4. Ova dva alata se obično koriste zajedno, no mogu se koristiti i odvojeno.

Kratki isječak koda koji prikazuje primjenu ovih alata:

```
import torch

# Stvori model i optimizer
model = Net().cuda()
optimizer = optim.Adam(model.parameters(), ...)

# Stvaranje varijable scaler
scaler = torch.cuda.amp.GradScaler()

for input, target in data:
        optimizer.zero_grad()

        # Koristenje torch.cuda.amp.autocast
        with torch.cuda.amp.autocast():
                output = model(input)
                loss = loss_function(output, target)

        # Izlazak iz autocast
        # Koristenje varijable scaler
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()
```

Programski kod 3.2: Primer korištenja torch.cuda.amp paketa.

4. Implementacija

Ovo poglavlje obrađuje programsku izvedbu sustava koji koristi miješanu preciznost (amp), te tradicionalnog sustava bez miješane preciznosti (baseline ili osnovna).

4.1. Korišteni alati i tehnologije

Za izradu rada korišten je programski jezik Python[1], radno okruženje PyTorch[2], te biblioteka time[3]. Biblioteka time je korištena za izračun vremena treniranja amp i osnovne varijante.

Za programsku implementaciju korišten je radno okruženje PyTorch verzije 1.6.0. Pošto PyTorch nudi podatkovnu strukturu tenzor (engl. *tensor*), nije bilo potrebe za korištenje biblioteke Numpy[4] i Numpy polja koja ona pruža. Najveća prednost tenzora u odnosu na Numpy polje je što se može izvoditi na grafičkim karticama koje podržavaju CUDA[5], te podržavaju automatsku diferencijaciju. Također PyTorch pruža paket torch.cuda[6] koji daje podršku za CUDA tenzore koji koriste GPU za izračune. Uz podršku za izvođenje tenzora na GPU, torch.cuda nudi još puno korisnih funkcionalnosti od kojih je jedna i izračunavanje zauzeća GPU memorije od strane tenzora.

```
1 def start_timer():
2         global start_time
3         gc.collect()
4         torch.cuda.empty_cache()
5         torch.cuda.reset_peak_memory_stats()
6         torch.cuda.synchronize()
```

[1] https://www.python.org/
[2] https://pytorch.org/
[3] https://docs.python.org/3/library/time.html
[4] https://numpy.org/
[5] https://developer.nvidia.com/cuda-zone
[6] https://pytorch.org/docs/stable/cuda.html

```
7           start_time = time.time()
8
9   def end_timer(message):
10          torch.cuda.synchronize()
11          end_time = time.time()
12          print("\n" + message)
13          print("Total execution time = {:.3f} sec"
14                  .format(end_time - start_time))
15          print(Memory used = {}MB
16                  .format(torch.cuda.max_memory_allocated()
17                                              /(1024**2)))
```

Programski kod 4.1: Prikazuje metode za iračun vremena treniranja amp i baseline modela, kao i izračun zauzeća memorije modela na kraju treniranja.

4.2. Podatkovni skup CIFAR10

Oba načina (amp i baseline) trenirana su koristeći skup podataka *CIFAR10*[7]. Skup podataka *CIFAR10* se sastoji od 60000 slika dimenzija 32x32x3 iz 10 različitih kategorija (avion, automobil, ptica, mačka, jelen, pas, žaba, konj, brod i kamion). Svaka kategorija sadrži točno 6000 primjeraka. Dodatno skup podataka je unaprijed podijeljen na 50000 slika za treniranje, te 10000 slika za testiranje. Ispitni skup podataka se sastoji od 1000 slika svake kategorije[5].

Slika 4.1: Skup podataka CIFAR10 sa 10 nasumično odabranih slika iz svake kategorije.

[7] https://www.cs.toronto.edu/~kriz/cifar.html

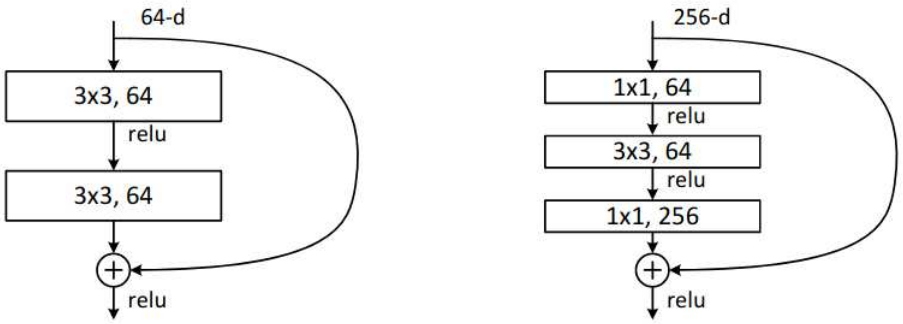
4.3. Korišteni modeli

Arhitektura neuronske mreže korištene za traniranje je ResNet (engl. *Residual Network*). Konkretno korišteni su ResNet-18, ResNet-34 i ResNet-50.

4.3.1. ResNet

Problem kojeg ResNet nastoji riješiti je problem degradacije (engl. *degradation problem*). Problem degradacije nastaje kada se povećava dubina modela, pa se time smanjuje točnost na skupu za učenje, ali i na ispitnom skupu. ResNet koristi rezidualne blokove za izgradnju modela koji služe za rješavanje navedenog problema.

Razlikuju se dva tipa konvolucijski jedinica: osnovna (engl. *basic*) i konvolucijska jedinica s uskim grlom (engl. *bottleneck*) prikazani na slici 4.2.



Slika 4.2: Prikaz konvolucijski jedinica ResNet-a. Lijevo je prikazana osnovna jedinica, dok je desno prikazana konvolucijska jedinica s uskim grlom. Slika preuzeta iz [4]

Kod osnovnih jedinica koriste se dvije konvolucije s filtrima dimenzija 3x3, a konvolucijska jedinica s uskim grlom koristi dvije konvolucije dimenzije 1x1 i jednu konvoluciju kao i osnovna jedinica (slika 4.2). Pošto konvolucijske jedinice zahtijevaju da je dimenzija izlaza iz konvolucijskih jedinica jednaka dimenziji prečice (engl. *skip-connection*) konvolucije 1x1 se postavljaju prije, odnosno poslije 3x3 konvolucije. Time prva 1x1 konvolucija smanjuje broj kanala, dok druga 1x1 konvolucija povećava na početni broj kanala. Korištenje rezidualnih konvolucijskih jedinica omogućava povećanje dubine modela, no povećava i broj parametara.

ResNet-18 i ResNet-34 su građeni samo od osnovnih konvolucijskih jedinica, te se sastoji od 18 odnosno 34 sloja, dok je ResNet-50 građen od konvolucijskih jedinica s uskim grlom i sastoji se od 50 slojeva.

4.4. Amp i baseline izvedba

Amp izvedba je provedena uz praćenje uputa službene PyTorch stranice. Korišteni su već navedeni torch.cuda.amp.autocast i torch.cuda.amp.GradScaler. Petlja za učenje je rađena na uzor programskog koda 3.2, uz korištenje dodatne vanjske *for* petlje za epohe.

Za razliku od amp izvedbe, baseline ne koristi torch.cuda.amp.autocast i torch.cuda.amp.GradScaler, pa prema tome petlja za učenje nema liniju 12, kao ni varijablu scaler. Ostatak programskog koda se ne razlikuje.

5. Rezultati

Obje varijante (amp i baseline) koristile su iste modele za učenje (ResNet-18, ResNet-34 i ResNet-50). Svi modeli su evaluirani na CIFAR10 skupu podataka. Kao točnost modela korišten je omjer točno klasificiranih slika kroz ukupan broj slika.

$$\text{točnost} = \frac{\text{točno klasificirane slike}}{\text{ukupan broj slika}} \tag{5.1}$$

Od ukupno 60000 slika, 10000 je korišteno za testiranje, 45000 za treniranje, te 5000 za validiranje. Skup podataka za validiranje je napravljen od 5000 nasumično odabranih slika iz skupa za učenje. Stopa učenja (engl. *learning rate*) postavljena je na 10^{-1}, te se tijekom učenja nakon svake dvadesete epohe ažurirala za parametar gama koji iznosi 0.5. Korišteni optimizator je stohastički gradijentni spust sa zaletom. Paramater zaleta (engl. *momentum*) smo postavili na vrijednost 0.9. Veličina grupe (engl. *batch-size*) je 128. Treniranje je provedeno sa 50 epoha na ResNet-18 i ResNet-34, dok je za ResNet-50 treniranje provedeno sa 60 epoha. Tijekom treniranja korištena je funkcija gubitka unakrsna entropija opisana jednadžbom 2.11.

Eksperimenti su provedeni koristeći Colab[1] i grafičku karticu Tesla T4 koja podržava rad s miješanom preciznosti. Cilj eksperimenata je izračunati i usporediti točnost, vrijeme treniranja i memorijski otisak modela koji koriste amp i modela koji ne koriste amp.

[1] `https://colab.research.google.com/notebooks/intro.ipynb?utm_source=scs-index`

	ResNet-18	ResNet-34	ResNet-50
točnost	0.8361	0.8313	0.7824
vrijeme treniranja	3593 s	5596 s	12815 s
memorijski otisak	707 MB	1221 MB	3387 MB

Tablica 5.1: Rezultati eksperimenta bez korištenja miješane preciznosti.

	ResNet-18	ResNet-34	ResNet-50
točnost	0.8367	0.8330	0.7886
vrijeme treniranja	1793 s	2802 s	5821 s
memorijski otisak	445 MB	788 MB	1883 MB

Tablica 5.2: Rezultati eksperimenta sa korištenjem miješane preciznosti.

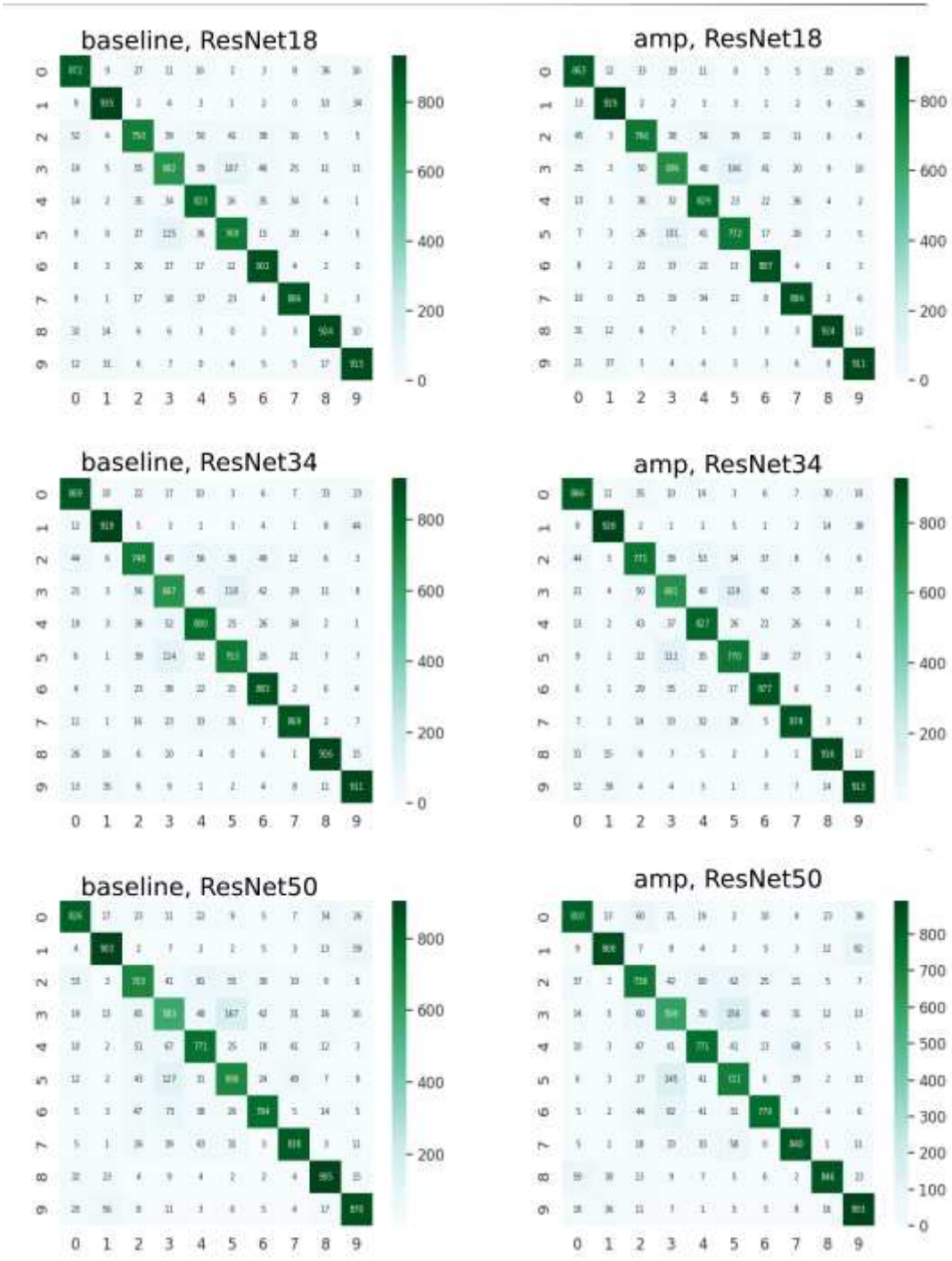
Vrijeme treniranja smo izračunali pomoću programskog isječka 4.1. Za izračun memorijskog otiska korišteni su dostupni alati koje nudi torch.cuda paket. Konkretno korištene su metode torch.cuda.memory_allocated() i torch.cuda.max_memory_allocated(). Prva metoda vraća trenutno zauzeće tenzora na GPU, a druga metoda vraća maksimalno zauzeće tenzora na GPU u bajtovima.

Iz tablica 5.1 i 5.2 vidimo da korištenjem miješane preciznosti ne gubimo na točnosti. Također vrijeme treniranja se upola smanjuje na svakom testiranom modelu, što je i očekivano. Zanimljivo je da se memorijski otisak kod ResNet-18 i ResNet-34 nije previše smanjio, kao što bi očekivali. Mogući razlog ovog ponašanja je da broj izračuna u FP16 formatu nije dovoljno velik da se nadomjesti memorijsko zauzeće čuvanja kopija težina u FP32 formatu. Kod ResNet-50 vidimo da se memorijski otisak smanjio za otprilike 40%. Moguće objašnjenje smanjenja zauzeća memorija od 40% kod ResNet-50 je što ResNet-50 ima puno više parametara u odnosu na ResNet-18 i ResNet-34, pa samim time i puno više izračuna u FP16.

Još jedan od načina kako računati i prikazati točnost modela je matrica zabune (engl. *confusion matrix*). Matrica zabune predstavlja tablicu u kojoj se prikazuju svi rezultati klasifikacije na testnom skupu podataka. Drugim riječima, matrica zabune govori koliko puta je model točno klasificirao neku klasu, te koliko puta je za neku klasu odredio neku drugu klasu (slika 5.1). Iz matrice zabune možemo očitati i točnost modela, no korisna je i da vidimo koje klase model "miješa", tj. vidimo za koju klasu model često klasificira neku drugu određenu klasu.

Slika 5.1 prikazuje šest matrica zabune, po jednu za svaki model (ResNet-18, ResNet-34 ili ResNet-50) i način izvedbe (amp ili baseline). Redci, odnosno stupci matrica su označeni brojevima 0-9. Svaki broj predstavlja jednu klasu. Konkretno 0 predstavlja avion, 1 predstavlja automobil itd. (nastavak se može pročitati sa slike 4.1). Redci matrice zabune predstavljaju točne klase, dok stupci predstavljaju predikcije.

Ako pogledamo matrice zabune baseline i amp izvedbe, vidi se da su amp matrice zabune dosta slične baseline matricama, pa se tim dodatno pokazuje da miješana preciznost ne narušava točnost modela. Dodatno, amp kao i baseline "griješi" na sličnim klasama. Primjerice vidimo da obje izvedbe pogrešno klasificiraju klasu automobil u klasu kamion i obratno.



Slika 5.1: Prikaz i usporedba matrica zabune za testirane modele u amp i baseline izvedbi.

6. Zaključak

U ovom radu su detaljnije opisani pojmovi vezan za neuronske mreže i njihov rad. Opisani su algoritmi gradijentnog spusta i propagacije pogreške unazad koji se koriste za učenje. Također je opisana arhitektura neuronskih mreža, a posebice rad i karakteristike konvolucijskih neuronskih mreža.

Pobliže je pojašnjen FP16 zapis i problemi koji se mogu stvoriti korištenjem istog. Opisana su i rješenja koja nastaju korištenjem FP16 formata. Detaljnije je obrađen pojam miješane preciznosti, njene prednosti u odnosu na tradicionalan pristup, implementacija i korištenje, te su prikazane osnovne smjernice za izražavanje učenja u miješanoj preciznosti.

Korišteni su rezidualni modeli, konkretno ResNet-18, ResNet-34 i ResNet-50. Svi modeli su testirani na skupu podataka CIFAR10. Upotrebom miješane preciznosti se ne gubi točnost, no vrijeme treniranja se na svim modelima prepolovilo.

U budućnosti bi se mogla usporediti miješana preciznost na dubljim modelima i drugim skupovima podataka. Također bi bilo zanimljivo isprobati na drugim problemima računalnog vida poput semantičke segmentacije.

LITERATURA

[1] Oreč F. Konvolucijski modeli za jedooku predikciju dubine scene. Završni rad, Fakultet elektrotehnike i računarstva, 2019.

[2] Goodfellow I., Bengio Y., i Courville A. *Deep Learning*. MIT Press, 2016. URL https://www.deeplearningbook.org/.

[3] Grubišić I. Semantička segmentacija slika dubokim konvolucijskim mrežama. Završni rad, Fakultet elektrotehnike i računarstva, 2016.

[4] He K., Zhang X., Ren S., i Sun J. Deep residual learning for image recognition. *CoRR, abs.-1512.003385*, 2015. URL https://arxiv.org/abs/1512.03385.

[5] Alex Krizhevsky. *The CIFAR-10 dataset*, 4. lipanj 2021. URL https://www.cs.toronto.edu/~kriz/cifar.html.

[6] Srivastava N., Hinton G., Krizhevsky A., Sutskever I., i Salakhutdinov R. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15*, 2015. URL https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf.

[7] NVIDIA. Training with mixed precision. zadnje ažurirano 4. rujan 2019. URL https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html.

[8] Narang S., Diamos G., Elsen E., Micikevicius P., Alben J., Garcia D., Ginsburg B., Houston M., Kuchaiev O., Venkatesh G., i Wu H. Mixed precision training. U *International Conference on Learning Representations ICLR*, 2018. URL https://arxiv.org/pdf/1710.03740.pdf.

[9] Čupić M. *Umjetna inteligencija: Umjetne neuronske mreže*. URL http://java.zemris.fer.hr/nastava/ui/ann/ann-20180604.pdf.

Učenje konvolucijskih modela u miješanoj točnosti

Sažetak

U ovom radu su opisane umjetne neuronske mreže, a posebice konvolucijske neuronske mreže. Opisane su metode učenja pomoću gradijentnog spusta i propagacije pogreške unazad, kao i neke tehnike regularizacije. Također je pobliže obrađen rad i korištenje miješane preciznosti, njene prednosti u odnosu na tradicionalni pristup i implementacija u radnom okviru PyTorch. Implementirani su i evaluirani modeli koji koriste miješanu preciznost i oni koji ju ne koriste. Modeli su implementirani pomoću rezidualnih neuronskih mreža, ResNet-18, ResNet-34 i ResNet-50. Na kraju rada su prikazani i uspoređeni rezultati tradicionalnog pristupa i pristupa koji koristi miješanu preciznost.

Ključne riječi: neuronske mreže, konvolucijske neuronske mreže, miješana preciznost, amp, ResNet, baseline

Mixed-precision training of convolutional models

Abstract

Paper describes artificial neural networks, and in particular convolutional neural networks. Gradient descent learning methods and back-propagation technique are described, as well as some regularization techniques. The work and use of mixed precision, its advantages over traditional approach, and implementation in the PyTorch framework are also discussed in more detail. Models that use mixed precision and those that do not use it have been implemented and evaluated. The models were implemented using residual neural networks, ResNet-18, ResNet-34 i ResNet-50. At the end of the paper, the results of the traditional approach and the approach that uses mixed precision are presented and compared.

Keywords: neural networks, convolutional neural networks, mixed precision, amp, ResNet, baseline