SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5697

Klasifikacija slika dubokim konvolucijskim modelima s rezidualnim vezama

Antonio Borac

Zagreb, srpanj 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 16. ožujka 2018.

ZAVRŠNI ZADATAK br. 5697

Pristupnik: **Antonio Borac (0036492829)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Klasifikacija slika dubokim konvolucijskim modelima s rezidualnim vezama**

Opis zadatka:

Klasifikacija slika prirodnih scena je neriješen problem računalnog vida s mnogim zanimljivim primjenama. U posljednje vrijeme najbolji rezultati u tom području postižu se pristupima utemeljenima na dubokim konvolucijskim modelima. Za ovaj rad posebno su zanimljivi nadzirani pristupi gdje je svaka slika skupa za učenje označena semantičkim razredima objekata koje slika sadrži.
U okviru rada, potrebno je istražiti postojeće postupke za klasifikaciju slika. Proučiti dokumentacije programskih okvira Tensorflow i PyTorch te biblioteke programskog jezika Python za rukovanje matricama i slikama. Izraditi izvedbu programskog sustava za učenje i primjenu klasifikacijskog modela. Evaluirati utjecaj rezidualnih veza na točnost modela. Prikazati i ocijeniti ostvarene rezultate.
Radu priložiti izvorni i izvršni kod razvijenih postupaka, ispitne slijedove i rezultate, uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 16. ožujka 2018.
Rok za predaju rada: 15. lipnja 2018.

Mentor:
Prof. dr. sc. Siniša Šegvić

Predsjednik odbora za završni rad modula:
Prof. dr. sc. Siniša Srbljić

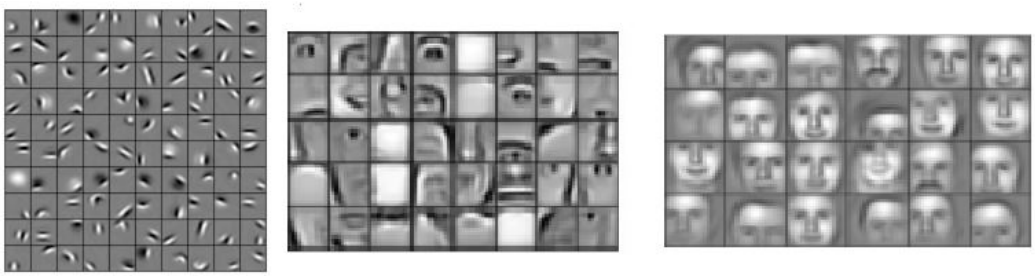
Dielovođa:
Doc. dr. sc. Tomislav Hrkać

Zahvaljujem se svom mentoru prof.dr.sc Siniši Šegviću na velikoj pomoći pri izradi ovog rada i svojim roditeljima za podršku tijekom cijelog školovanja.

SADRŽAJ

1. Uvod

Klasifikacija slika je jedan od najzanimljivih problema računalnog vida. Pokazuje se kao čest problem u svakodnevnoj primjeni kojeg bi željeli rješavati automatski. Razlika u svjetlini, položaju objekata unutar slike kao i orijentacija objekata unutar slike predstavljaju velike izazove u klasifikaciji koji otežavaju razvoj robusnih modela. Trenutno najzanimljiviji modeli koji veoma uspješno rješavaju probleme klasifikacije slika na ogromnim skupovima podataka poput Imagenet-a [6] su duboki konvolucijski modeli. Konvolucijski modeli su specifični po tome što samostalno uče prepoznavati značajke ulaznih podataka karakteristične za pojedine razrede koje omogućavaju preciznu klasifikaciju. Duboki konvolucijski modeli pretpostavljaju da se podaci mogu hijerarhijski razložiti na niz značajki - od temeljnih značajki poput rubova i linija, do kompliciranijih poput šarenice oka, boje kose ili u konačnici cijelog lica. Takav niz značajki u dubokom učenju učimo raspoznavati sa nizom slojeva - modeli koje razvijamo predstavljaju srž područja dubokog učenja. Za potrebe učenja i testiranja takvih modela



Slika 1.1: Niz hijerarhijskih značajki iz slike. Slika je preuzeta iz [2]

potrebni su veliki skupovi slika među kojima se ističu primjerice [6] koji sadrži preko `10` tisuća različitih klasa slika i danas predstavlja jedan od najkvalitetnijih skupova za učenje i testiranje modela, te [12], jednostavniji skup za ispitivanje modela razvijenih za probleme klasifikacije i regresije.

Za potrebe izgradnje modela za strojno učenje su razvijene popularne biblioteke poput Tensorflow-a [4] i PyTorcha koje znatno olakšavaju i ubrzavaju proces razvoja i treniranja modela.

Današnji uspješni modeli se baziraju na rezidualnim blokovima koji se oslanjaju na pojam rezidualnih veza. Rezidualna veza omogućava direktan prolaz signala i gradijenata kroz blok dubokog modela bez utjecaja nelinearnih funkcija, što se pokazuje iznimno korisnim za učenje dubokih modela. Pokazuje se da ovakvom organizacijom možemo graditi dublje modele koji će moći dati bolje rezultate na problemima u okviru dubokog učenja. Cilj ovog rada je razviti duboke konvolucijske modele sa rezidualnim vezama za klasifikaciju slika na skupu CIFAR-10 i ustanoviti utjecaj rezidualnih veza na točnost klasifikacije podataka. Rezultate je potrebno usporediti sa jednostavnijim modelima koje ćemo za tu svrhu rekonstruirati. U radu je dan kraći pregled popularne Tensorflow biblioteke za strojno učenje. Detaljan opis arhitekture i rezultata, kao i dijelovi programske implementacije i tehnika korištenih u razvoju modela su prikazani u nastavku rada.

2. Duboko učenje

Duboko učenje kao grana strojnog učenja na popularnosti je dobila tek nedavno, razvojem novih tehnologija modeliranja i učenja modela. Znatno poboljšavanje procesne moći računala[1] kao i razvoj velikih skupova od nekoliko stotina tisuća podataka omogućavaju izgradnju modela koji danas ostvaruju impresivne rezultate. Duboko učenje se bazira na dubokim modelima, odnosno arhitekturama koje sadrže slijed naučenih nelinearnih transformacija za ekstrakciju značajki iz podataka.

2.1. Konvolucijski modeli

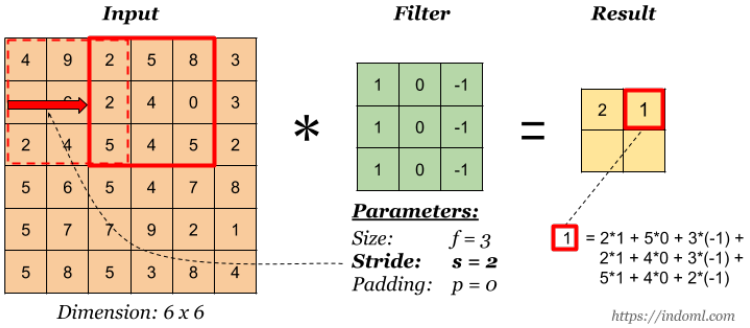
U praksi, najčešći modeli koje danas koristimo u okviru računalnog vida su konvolucijski modeli. Takve modele karakteriziraju konvolucijski slojevi koji omogućavaju različite aktivacije na pojedinim dijelovima ulaznih podataka.

2.1.1. Konvolucijski slojevi

Konvolucijski slojevi su najznačajniji slojevi specifični za konvolucijske neuronske mreže. Specifični su po operaciji konvolucije koju provode nad ulaznim podacima. Svaki konvolucijski sloj obilježavaju filtri koji se koriste u operaciji konvolucije, njihova dimenzija koja određuje dimenzije mape izlaznih značajki sloja, te pomak kojim se filtar pomiče po mapi ulaznih značajki u procesu generiranja mape izlaznih značajki. Proces konvolucije možemo najbolje prikazati na jednom primjeru. Ulaz u konvolucijski sloj nam prestavlja jedna slika dimenzija `32x32x3`. Konvoluciju vršimo sa `16` filtera dimenzija `3x3`. Filtar pomičemo za `1` u horizontalnom i vertikalnom smjeru. Bitno je napomenuti da filtar konvoluciju radi na cijeloj dubini ulaza, tako je zapravo filtar u našem slučaju dimenzija `3x3x3`. Izlazni sloj značajki će imati dubinu onoliku koliko imamo filtera a to je u našem slučaju `16`. Širinu izlaznog sloja računamo na sljedeći način $W_{out} = (W_{in} - F + 2P)/S + 1$, gdje je W_{in} širina ulazne matrice, F

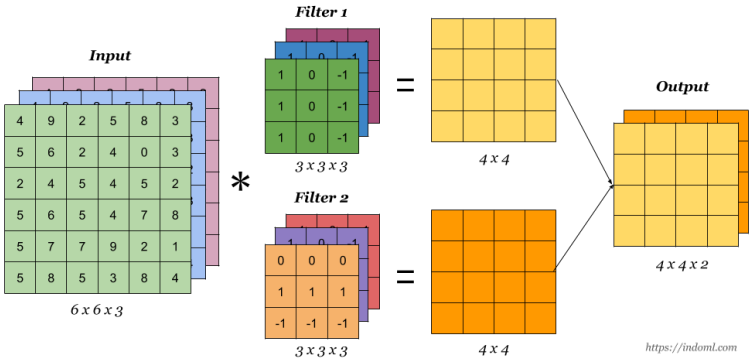
[1] Nastanak tehnologije cuda, biblioteke cuDNN kao i ogroman napredak današnjih grafičkih kartica

je širina filtra, P označava broj nula koje nadodajemo po širini mape sa svake strane za kontrolu širine izlazne mape značajki, S označava pomak po širini a W_{out} konačnu širinu izlazne mape značajki. Visinu izlazne mape značajki računamo analogno. Tako u našem primjeru konačne dimenzije izlazne mape značajki su `32x32x16`. Sam operator konvolucije pozicijski množi svaku težinu filtra sa svakom vrijednosti perceptivnog polja filtra te sve dobivene vrijednosti zbraja u jedan skalar koji predstavlja jedan element izlazne mape značajki. Dobar prikaz kako operacija konvolucije funkcionira možemo vidjeti na slici 2.1. Na slici 2.2 vidimo kako se slojevi nižu jedan na drugi i



Slika 2.1: Prikaz kako funkcionira operator konvolucije. Slika je preuzeta sa [1]

tako tvore konačnu izlaznu mapu značajki za neki konvolucijski sloj.



Slika 2.2: Slika prikazuje kako se formira konačni izlaz konvolucijskog sloja. Slika je preuzeta sa [1]

2.1.2. Potpuno povezani slojevi

U potpuno povezanom sloju svaki neuron je povezan sa svim izlazima prethodnog sloja. Potpuno povezni slojevi su korisni jer mogu naučiti nelinearne kombinacije značajki prethodnog sloja. Ako prethodni sloj u modelu ima ukupno `n` izlaza a potpuni sloj ima `m` neurona, između ta dva sloja ćemo imati `m x n` težina uz jednu dodatnu

težinu `b` koju nazivamo prag. Potpuno povezane slojeve u konvolucijskim neuronskim mrežama obično koristimo među zadnjim slojevima, jer kombiniraju značajke iz različitih dijelova mape ulaznih značajki i omogućavaju prepoznavanje izlazne klase iz informacija sadržanih u svim dijelovima slike, za razliku od konvolucijskih slojeva koje obilježava svojstvo lokalnosti. Značajke prije dovođenja na ulaz popunog sloja transformiramo u jednodimenzionalan vektor - matricu dimenzija `32x32x3` transformiramo u vektor sa `3072` člana. U okviru rada koristimo potpuno povezan sloj sa `10` neurona na kraju mreže, tako da svaki neuron daje izlaz za određen semantički razred. U dubokim konvolucijskim modelima za estrakciju značajki koristimo konvolucijske slojeve zbog njihovog svojstva lokalnosti i manjeg broja parametara što u konačnici doprinosi smanjenju vjerojatnosti pretreniranja mreže.

2.1.3. Sloj sažimanja

U ranijim modelima slojevi sažimanja su se masovno koristili za smanjivanje dimenzionalnosti ulazne mape značajki kako bi se smanjio broj parametara i nastojalo smanjiti pretreniranost. Sloj sažimanja provodi operacije nad svakim dijelom dubine ulazne mape značajki neovisno. Definiramo prozor dimenzija F, obavljamo definiranu operaciju i pomičemo prozor za S, slično kao kod konvolucijskih slojeva. Tako je širina izlazne mape značajki $W_{out} = (W_{in} - F)/S + 1$ gdje je W_{in} širina ulazne mape značajki, F je širina prozora, S označava pomak po širini a W_{out} širinu izlazne mape značajki. Analogno računamo i za visinu izlazne mape značajki. Bitno je primjetiti da dubina izlazne mape značajki nakon sloja sažimanja ostaje jednaka dubini ulazne mape značajki. Operacije koje obavljamo nad prozorom su obično operacije traženja maksimuma, traženje prosječne vrijednosti i L^2-sažimanje. Danas se konvolucijski slojevi sve rjeđe koriste jer primjerice u okviru rezidualnih modela za smanjenje dimenzionalnosti koristimo konvolucijske slojeve sa pomakom po širini i visini većim od `1`. U izgradnji našeg rezidualnog modela koristimo samo jedan sloj sažimanja sa operacijom traženja prosjeka sa pomakom po dužini i širini iznosa `1` nakon posljednjeg rezidualnog bloka.

2.2. Duboki konvolucijski modeli sa rezidualnim vezama

Duboki modeli su zaslužni za uspješno rješavanje mnogih problema poput klasifikacije slika, prevođenja jezika i analize teksta. Razdvajaju problem na učenje niza hijerarhijskih značajki čime omogućuju raspoznavanje složenijih objekata. Primjerice lice

čovjeka je sastavljeno od očiju, usta, nosa, oči su sastavljene od šarenice, bjeloočnice i zjenice i tako dalje. U sklopu rada istražujemo duboke modele sa rezidualnim vezama. Ideja dubokog učenja je razviti modele koji će podatake modelirati sa više razina apstrakcije. Doseg dubokih modela bez rezidualnih veza je primjerice model VGG19[14] sa `19` težinskih slojeva i velikim brojem parametara. U dubljim mrežama se počinje javljati problem povećane pogreške na skupu za treniranje koji nije uzrokovan pretreniranjem, što ćemo u ovom radu nastojati pokazati. Takve modele je teško trenirati a produbljivanjem modela se ne postiže veća točnost. Pokazalo se da samo daljnje produbljivanje modela neće moći riješiti kompliciranije probleme na području računalnog vida. Postojao je problem umirućih gradijenata koji je djelomično riješen sa normalizacijom ulaznih podataka i korištenjem BN slojeva koji su omogućili mrežama sa desecima slojeva ponovnu konvergenciju. Taj problem je potaknuo i korištenje aktivacijske funkcije zglobnice umjesto dotad popularne sigmoidalne aktivacijske funkcije. U okviru rada proučavamo predložen [10] rezidualni model koji uspješno rješava ove probleme. Konvolucijski slojevi se ne vežu direktno jedan na drugi već se uvodi veza prema ulaznom sloju bloka na način da od ulaza do izlaza imamo put gradijenata bez operacije konvolucije.

Rad pretpostavlja da je takvo mapiranje lakše optimizirati nego originalno mapiranje. Uvode se rezidualne veze koje ulaz na bloku konvolucijskih slojeva propagiraju bez izmjene na izlaz gdje se kombiniraju sa izlazom niza konvolucijskih slojeva. Takve veze ne dodaju dodatne parametre u model a izvedba nije računski zahtjevna - provodi se jednostavno zbrajanje ulaza sa izlazom mape značajki konvolucijskih slojeva. Takvu organizaciju možemo i matematički formulirati. Ako pretpostavimo da slojevi mogu aproksimirati neku složenu funkciju $\mathscr{H}(x)$, također možemo pretpostaviti da slojevi mogu aproksimirati funkciju $\mathscr{H}(x)-x$ [10]. Postavljamo da niz konvolucijskih slojeva aproksimira funkciju $\mathscr{F}(x) := \mathscr{H}(x) - x$, pa tako izlaz iz bloka predstavlja funkcija $\mathscr{F}(x) + x$.

Prednost takve organizacije modela je dodatno smanjivanje utjecaja umirućih gradijenata na model. U slučaju da blok konvolucijskih slojeva aktivacijom producira konstante izlaze uslijed aktivacije funkcija, odnosno da dio mreže odumre, model će i dalje moći optimizirati ostale parametre tijekom učenja propagacijom unazad. Dodatno, u rezidualnim modelima se intenzivno koriste BN slojevi kako bi se riješili navedeni problemi. Prednost rezidualnih veza je što omogućuju direktnu propagaciju informacije kroz slojeve. Problem sa klasičnim konvolucijskim mrežama je što izlaz sloja ovisi samo o izlazima prethodnog sloja. Rezidualne veze omogućavaju učenje konvolucijskih slojeva koji direktno ovise i o "plićim" dijelovima mreže.

3. Ispitni skupovi

Kako bismo mogli trenirati i testirati razvijene modele dubokog učenja potrebni su nam skupovi podataka. Skup podataka dijelimo na skup za treniranje i skup za testiranje. Iz skupa za treniranje izdvajamo mali dio koji koristimo kao validacijski skup. Skup podataka obično dijelimo tako da skup za treniranje sadrži 70% originalnih podataka, skup za validaciju i testiranje po 15% podataka[5]. Tijekom treniranja na ulaz modela dovodimo ulazne podatke, u našem slučaju slike, te za svaki ulazni podatak pripadnu oznaku semantičkog razreda kojemu određena slika pripada. Podatke propuštamo kroz naš model i izlaz modela uspoređujemo sa danim oznakama razreda na temelju kojih prilagođavamo parametre unutar modela algoritmom unazadne propagacije. Za potrebe treniranja i ispitivanja naših modela koristimo poznate ispitne skupove CIFAR10 i MNIST.

3.1. Predobrada podataka

Ulazni podatci često sadrže dosta šuma ili nisu prikladni za učenje, što nas motivira da ulazne podatke izmijenimo na način na koji bi naši modeli lakše učili - podatke želimo predprocesirati. Vršimo centriranje i normalizaciju podataka.

Centriranje se vrši oduzimanjem aritmetičke sredine cijeloga skupa od svakog člana skupa. Nakon centriranja aritmetička sredina skupa je 0. Kako bi normalizirali podatke, sve podatke podijelimo sa standardnom devijacijom. Ovaj postupak vršimo jer je moguće da naši podaci potencijalno odsupaju za nekoliko redova veličine što znatno otežava učenje modela. Normalizacijom u praksi ubrzavamo i olakšavamo učenje modela.

3.2. MNIST

MNIST ispitni skup je skup ručno pisanih znamenki. Skup za treniranje se sastoji od 60000 podataka a skup za testiranje od 10000 podataka. Svaka slika je veličine

28x28x1 piksela. Svaki piksel predstavlja razinu sive boje skalom od 0 do 255 (engl. *grayscale*), odnosno vrijednosti od 0 do 1, gdje 0 predstavlja crnu boju a 1 ili 255 bijelu boju. Danas čak i jednostavni konvolucijski modeli ostvaruju veoma dobre rezultate na MNIST skupu.

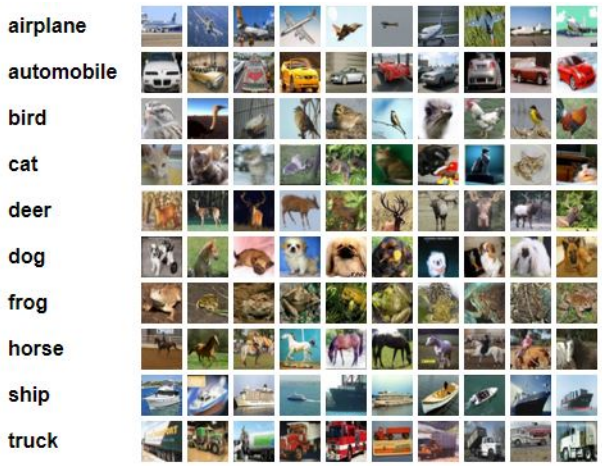
Slika 3.1: Primjer podataka iz skupa MNIST. Redom predstavljaju znamenke 6 7 7.

3.3. CIFAR10

Duboke konvolucijske modele sa rezidualnim vezama za potrebe ovog rada ćemo ispitivati na CIFAR10 skupu. Skup se sastoji od 60000 slika koje su podijeljene na 10 različitih semantičkih razreda koji predstavljaju različite vrste životinja i prijevoznih sredstava. Od svakog razreda u skupu imamo po 6000 slika. Skup za testiranje iz ovog skupa će sadržavati 10000 slika, dok će skup za treniranje sadržavati 50000 slika. Za potrebe validacije modela ćemo iz skupa za treniranje izdvojiti 5000 slika. Svaka slika je matrica dimenzija 32x32x3 u kojoj svaka jedinica matrice predstavlja količinu jedne od tri RGB komponente boje za određeni piksel slike, odnosno vrijednosti između 0 i 255 za svaku komponentu boje.

Podatke za potrebe treniranja ovog modela smo predprocesirali kako je opisano u poglavlju 3.1.

Slika 3.2: Primjer jednog podatka kojeg dovodimo na ulaz mreže.



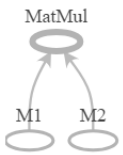
Slika 3.3: Primjer slika sadržanih u CIFAR10 skupu. Slika je preuzeta sa [12]

4. O Tensorflowu

Tensorflow je jedna od najpoularnijih biblioteka otvorenog koda za strojno učenje. Razvijena je pod okriljem Google-a koji ju je u početku koristio za internu upotrebu. Organizirana i jednostavna temeljna arhitektura, opširna dokumentacija i efikasnost izvođenja koda su glavne prednosti Tensorflow-a koje ga čine konkurentnim. Osim funkcionalnosti koje nudi za strojno učenje, koristi se i kao simbolička matematička biblioteka. Tensorflow se bazira na modelu toka podataka. Svaki Tensorflow program se dijeli u dvije faze:

- izgradnja statičkog grafa,
- izvođenje operacija.

Graf predstavlja reprezentaciju izračuna koji želimo obaviti u smislu ovisnosti između pojedinih operacija. U grafu toka podataka čvorovi predstavljaju jedinice izračuna, odnosno operacije, dok listove predstavljaju ulazni podaci koje graf konzumira i izlazni podaci koje graf proizvodi. Primjerice definirajmo jednostavan problem - želimo pomnožiti dvije matrice. Naš Tensorflow graf će se sastojati od tri čvora - ulazna matrica, izlazna matrica i čvora operacije množenja matrica.



Slika 4.1: Primjer Tensorflow grafa

Tensorflow graf interno sve **tenzore** pohranjuje u kolekcije do kojih je moguće doći unutar definicije grafa funkcijom dohvata `tf.get_collection` preko ključa imena varijable. Tenzori predstavljaju generalizaciju vektora i matrica. Interno se pohranjuju kao n-dimenzionalna polja podataka određenog tipa. Svaki tenzor definira dimenzija i tip podataka. U fazi izvođenja operacija definiramo Tensorflow sesiju

za pokretanje pojedinih dijelova grafa. Sesiji kao argument možemo predati Python imena iz definicije grafa koje nas zanimaju poput rezultata operacije množenja u našem slučaju. Također, preko rječnika `feed_dict` možemo predati vrijednosti ulaznih varijabli koje su nužne da bi se izračun mogao izvesti. Navedeni primjer u Tensorflowu bi mogao izgledati ovako:

```
import tensorflow as tf
import numpy as np

M1 = tf.placeholder(dtype=tf.float32,name="M1")
M2 = tf.placeholder(dtype=tf.float32,name="M2")
result = tf.placeholder(dtype=tf.float32,name="result")

result = tf.matmul(M1,M2)

with tf.Session() as sess:
    mat1 = np.array([[1,2,3],[4,5,6],[7,8,9]])
    mat2 = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])

    mul_result=sess.run([result],feed_dict={
                            M1: mat1,
                            M2: mat2,
                        })
    print ("Result of multiplying matrices mat1 and mat2.\n",
    mul_result)#numpy array
```

U sklopu rada sastavljanje grafa se svodi na definiciju ulaznih parametara, dubokog modela, funkcije gubitka, optimizatora koji se koristi u postupku treniranja i stope učenja. U fazi izvođenja operacija obično iterativno pozivamo sesiju kako bi naučila parametre na temelju podataka iz trenutne iteracije nad epohom iz trening skupa. Pozivamo je i sa podacima iz skupa za validaciju kako bi pratili moć generalizacije našeg modela. Na kraju treniranja je pozivamo sa skupom za učenje kako bi ispitali konačne performanse modela. Sami model neuronske mreže možemo jednostavno izgraditi korištenjem tehnologije poput `tf.layers` koji ukalupljuje sve varijable poput težina koje se optimiziraju i operacije potrebne za funkcioniranje određenog sloja neuronske mreže. Izgradnja modela postaje jednostavno nizanje zamišljenih slojeva jedan na drugi:

```
batch_norm = tf.layers.batch_normalization(conv1,training=isTrain)
relu = tf.nn.relu(batch_norm)
conv1 = tf.layers.Conv2D(filters=filters,
                         kernel_size=(3, 3),
```

```
5                                   strides=init_strides,
6                                   padding="same",)(relu)
```

Druga mogućnost je korištenje tehnologije `tf.nn` koja nam omogućuje pisanje koda na nešto nižoj razini. Kako se obje tehnologije baziraju na Tensorflow tenzorima, moguće ih je po volji kombinirati. Za potrebe izvođenja programske izvedbe u ovom radu ćemo koristiti Tensorflow sa podrškom za izvođenje na grafičkoj kartici. Tensorflow nam dopušta velik stupanj slobode pri definiciji svih dijelova modela. Biblioteku koristi široka zajednica ljudi koja konstantno doprinosi daljnjem razvoju. Za svaki dio biblioteke postoje primjeri koji olakšavaju primjenu napisanog koda. Tensorflow nudi i aplikacijsko korisničko sučelje visoke razine koje omogućava lakše i brže pisanje programa. Primjer je Tensorflow Estimator koji omogućava enkapsulaciju cijelog procesa učenja modela kojeg u okviru rada nećemo razmatrati.

5. Detalji implementacije

5.1. Arhitektura jednostavnog modela za klasifikaciju znamenki

Gradimo jednostvnu plitku neuronsku mrežu za klasifikaciju MNIST znamenki na sljedeći način:

- Ulazni sloj - podaci veličine `28x28x1` piksela, podijeljeni u mini grupe od po `32` slike
- Prvi konvolucijski sloj - dimezija filtra `5x5`, pomak po širini i visini iznosa `1`, `1` ulazni kanal i `16` izlaznih kanala. Izlaz iz sloja prosljeđujemo na ulaz od aktivacijske funkcije zglobnice.
- Prvi sloj sažimanja - jezgra dimenzija `2x2`, pomak po širini i visini iznosi `2`. Smanjuje dimenzije izlaznih značajki.
- Drugi konvolucijski sloj - dimenzija filtra `5x5`, `16` ulaznih i `32` izlazna kanala sa pomakom po visini i širini `1`.
- Drugi sloj sažimanja - jezgra dimenzija `2x2`, pomak po visini i širini iznosi `2`. Koristimo za smanjivanje dimenzija značajki.
- Potpuno povezani sloj sa `512` neurona.
- Izlazni sloj u kojem svaki neuron predstavlja izlaz za jednu klasu.

5.2. Arhitektura plitkog modela za skup CIFAR10

Za potrebe usporedbe plitkih i dubokih konvolucijskih modela, gradimo jednostavnu plitku neuronsku mrežu sastavljenu od sljedećih slojeva:

- Ulazni sloj predstavlja jednu mini grupu podataka iz skupa CIFAR10, svaki podatak je veličine `32x32x3`.

– Prvi konvolucijski sloj - dimenzija filtra `5x5`, `3` ulazna i `16` izlaznih kanala, pomak po visini i širini iznosi `1`. Izlaz sloja dovodimo na ulaz aktivacijske funkcije zglobnice.

– Prvi sloj sažimanja - jezgra dimenzija `3x3`, pomak po širini i visini iznosa `2`.

– Drugi konvolucijski sloj - dimenzija filtra `5x5`, `16` ulaznih kanala, `32` izlazna kanala. Pomak po visini i širini iznosa `1`. Izlaz sloja dovodimo na ulaz aktivacijske funkcije zglobnice.

– Drugi sloj sažimanja - jezgra dimenzija `3x3`, pomak po visini i širini iznosa `2`.

– Prvi potpuno povezani sloj - `256` neurona, izlaz sloja dovodimo na ulaz aktivacijske funkcije zglobnica.

– Drugi potpuno povezani sloj - `128` neurona, izlaz sloja dovodimo na ulaz aktivacijske funkcije zglobnica.

– Izlazni sloj - svaki od `10` izlaznih neurona predstavlja izlaz za svaku klasu iz skupa CIFAR10.

5.3. Arhitektura dubokog rezidualnog modela za skup CIFAR10

Model koji razvijamo bazira se na Resnet[10] arhitekturi za skup CIFAR10. Gradimo duboku neuronsku mrežu od sljedećih slojeva:

– Ulazni sloj predstavlja jednu grupu podataka iz skupa kojoj je svaki član slika iz skupa CIFAR10 `32x32x3`,

– Prvi konvolucijski sloj: dimenzija filtra - `3x3` , jedan ulazni i `16` izlaznih kanala. Pomak po dužini i širini iznosi `1`. Na izlaz konvolucijskog sloja stavljamo sloj normalizacije grupe(engl. *Batch Normalization layer*) (kasnije u tekstu BN). Izlaz BN sloja dovodimo na ulaz aktivacijske funkcije zglobnice.

– Blok konvolucijskih slojeva sa filtrom sa `16` izlaznih kanala - blok ponavljamo `n` puta

  - Konvolucijski sloj: dimenzija filtra - `3x3` , `16` ulaznih i `16` izlaznih kanala. Pomak po dužini i širini iznosi `1`. Nad izlazom konvolucijskog sloja radimo normalizaciju grupe i dovodimo na ulaz aktivacijske funkcije zglobnice.

    - Konvolucijski sloj: dimenzija filtra - `3x3` , `16` ulaznih i `16` izlaznih kanala. Pomak po dužini i širini iznosi `1`. Nad izlazom konvolucijskog sloja radimo normalizaciju grupe(BN sloj). Izlaz BN sloja zbrajamo sa zapamćenim ulazom na početku bloka i propuštamo kroz aktivacijsku funkciju zglobnicu.

– Blok konvolucijskih slojeva sa filtrom sa `32` izlazna kanala - blok ponavljamo `n` puta
    - Konvolucijski sloj: dimenzija filtra - `3x3` , `32` ulazna i `32` izlazna kanala. Pomak po dužini i širini iznosi `1`. Nad izlazom konvolucijskog sloja radimo normalizaciju grupe i dovodimo na ulaz aktivacijske funkcije zglobnice.
    - Konvolucijski sloj: dimenzija filtra - `3x3` , `32` ulazna i `32` izlazna kanala. Pomak po dužini i širini iznosi `1`. Nad izlazom konvolucijskog sloja radimo normalizaciju grupe(BN sloj). Izlaz BN sloja zbrajamo sa zapamćenim ulazom na početku bloka i propuštamo kroz aktivacijsku funkciju zglobnicu.

– Blok konvolucijskih slojeva sa filtrom sa `64` izlazna kanala - blok ponavljamo `n` puta
    - Konvolucijski sloj: dimenzija filtra - `3x3` , `64` ulazna i `64` izlazna kanala. Pomak po dužini i širini iznosi `1`. Nad izlazom konvolucijskog sloja radimo normalizaciju grupe i dovodimo na ulaz aktivacijske funkcije zglobnice.
    - Konvolucijski sloj: dimenzija filtra - `3x3` , `64` ulazna i `64` izlazna kanala. Pomak po dužini i širini iznosi `1`. Nad izlazom konvolucijskog sloja radimo normalizaciju grupe(BN sloj). Izlaz BN sloja zbrajamo sa zapamćenim ulazom na početku bloka i propuštamo kroz aktivacijsku funkciju zglobnicu.

– Sloj sažimanja sa operatorom prosjeka - pomak po dužini i širini iznosa `1`. Dimenzija jezgre ovisi o konkretnoj mreži.

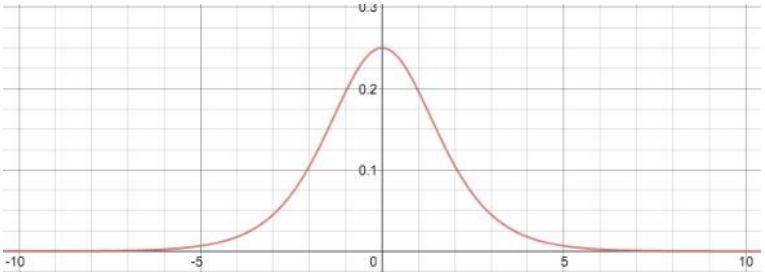
– Sloj izravnjavanja - ulaz izravnamo u jedan vektor kako bi podatke pripremili za potpuno povezani sloj.

– Potpuno povezani sloj - svaka jedinica je spojena sa svakom jedinicom prethodnog sloja. U ovom sloju imamo `10` jedinica od kojih svaka na izlazu daje nenormalizirane aktivacije za svaku izlaznu klasu.

Primjerice, za `n=3` ukupno težinskih slojeva imamo $3*2*3+2 = 20$. Tako konstruiranu mrežu nazivamo Resnet20. Bitno je napomenuti da u prvom konvolucijskom sloju u blokovima sa `32` i `64` sloja koristimo pomak `2` po visini i širini kako bi prikladno smanjili veličinu mape značajki - do ovih vrijednosti smo došli računajući potrebne veličine kako je opisano u 2.1.1.

U našim modelima nakon svakog konvolucijskog sloja koristimo BN sloj[1]. Problem sa značajkama je kao i sa ulaznim podacima - ako u dijelu mreže imamo značajke iz različitih distribucija, model će sporo učiti. Ako podaci is skupa za treniranje i podaci iz skupa za testiranje dolaze iz različitih distribucija, model će se loše ponašati na skupu za testiranje - pogreška će se propagirati kroz mrežu većom amplitudom. Bitno je naglasiti da se BN sloj različito ponaša prilikom treniranja i testiranja mreže. Prilikom treniranja se aritmetička sredina i standardna devijacija težina računaju nad cijelom mini grupom. Prilikom testiranja se aritmetička sredina i standarna devijacija računaju naučenim parametrima koji se uče prilikom treniranja mreže.

Nakon svakog BN sloja koristimo aktivacijsku funkciju zglobnicu (engl. *ReLU - Rectified Linear Unit*). ReLU je aktivacijska funkcija koja je omogućila duboke modele kakve danas poznajemo. Glavna prednosta ReLU nad primjerice često korištenom sigmoidalnom funkcijom je manja vjerojatnost problema umirućih gradijenata. Problem umirućih gradijenata se manifestira kod aktivacijskih funkcija koje imaju zasićenje za jako negativne i jako pozitivne vrijednosti mape značajki. Sigmoidalna funkcija



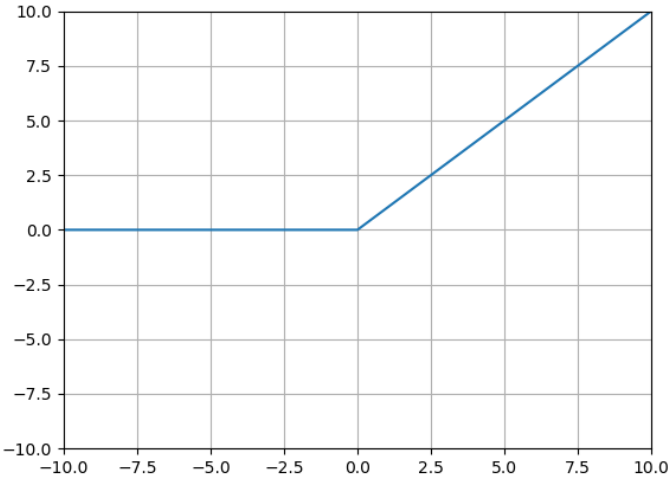
Slika 5.1: Prikaz gradijenata sigmoidalne aktivacijske funkcije. Slika je preuzeta sa [3]

za negativne vrijednosti producira vrijednosti blizu `0`, dok za pozitivne vrijednosti producira vrijednosti blizu `1`. U takvim situacijama je gradijent funkcije veoma blizu `0` što znači da se prilikom algoritma unazadne propagacije težine praktički ne ažuriraju. Prednost aktivacijske funkcije zglobnice je što je njezina kodomena znatno šira - ulaz manji od `0` priteže na `0`, dok ulaz veći od `0` propušta. Gradijent za ulaze veće od `0` je

[1]U okviru rada [11] se predlaže nešto drugačija arhitektura pojedinog bloka koju bi bilo korisno razmotriti

različit od `0`.

$$f(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases} \tag{5.1}$$



Slika 5.2: Prikaz ReLU aktivacijske funkcije.

Navedenim postupkom se postiže nelinearnost koja je nužna u dubokim modelima[8]. Glavni nedostaci zglobnice kao aktivacijske funkcije:

- Gradijent nije definiran za `x = 0` - problem rješavamo postavljanjem na vrijednost `0` ili `1`
- Za `x < 0`, odnosno u neaktivnom stanju, ne propušta značajke unaprijed i gradijente unatrag - ako koristimo normalizaciju grupe koja implicira srednju vrijednost `0` i standardnu devijaciju `1`, vjerojatnost neaktivnog stanja je `0.5`, što je prihvatljivo

Potencijalan problem sa aktivacijskom funkcijom zglobnicom je slučaj kad se težine ažuriraju na način da većina težina postane negativna - odmak se nauči tako da aktivacijska funkcija propušta konstantnu vrijednost `0` za bilo koji ulazni podatak - problem umiruće ReLU funkcije. Problemu možemo doskočiti tako da za vrijednosti manje od `0` koristimo $\alpha * x$, gdje je pri tome α mali pozitivan broj a x ulaz funkcije, što omogućuje funkciji da se polako oporavi - nova aktivacijska funkcija se naziva propusna zglobnica(engl. *LeakyReLU*).[2] U nastavku prikazujemo implementaciju rezidualnih blokova. Funkcija `shortcut` obavlja zbrajanje ulaza sa blokom niza konvolu-

[2]U ovom radu je nismo razmatrali, u daljnem radu bi bilo dobro isprobati utjecaj ove aktivacijske funkcije na učenje modela.

cija. Funkcija `residual_block` predstavlja ključan dio implementacije. Ponavlja blok sa pojedinim filtrom n puta. Funkcija `basic_block` gradi svaki pojedini blok.

```
def shortcut(input, residual):
    """Adds a shortcut between input and block of convolutions and merges them with "sum"
    """
    input_shape = input.get_shape().as_list()
    residual_shape = residual.get_shape().as_list()
    stride_width = int(round(input_shape[ROW_AXIS] / residual_shape[ROW_AXIS]))
    stride_height = int(round(input_shape[COL_AXIS] / residual_shape[COL_AXIS]))
    equal_channels = input_shape[CHANNEL_AXIS] == residual_shape[CHANNEL_AXIS]

    shortcut = input

    # 1 X 1 conv if shape is different. Else identity.
    if stride_width > 1 or stride_height > 1 or not equal_channels:
    #
        initializer=tf.keras.initializers.he_normal()
        regularizer = tf.contrib.layers.l2_regularizer(scale=WEIGHT_DECAY)
        shortcut = tf.layers.Conv2D(filters=residual_shape[CHANNEL_AXIS],
                                    kernel_size=(1, 1),
                                    strides=(stride_width, stride_height),
                                    padding="SAME",
                                    kernel_regularizer=regularizer,
                                    kernel_initializer=initializer)(input)

    return tf.add(shortcut,residual)

def residual_block(block_function, filters, repetitions, is_first_layer=False):
    """Builds a residual block repeating block with same filter n times.
    """
    def f(input):
        for i in range(n):
            init_strides = 1
            if i == 0 and is_first_layer:
```

```
                init_strides = 1
            input = block_function(filters=filters, init_strides=init_strides,
                                   is_first_block_of_layer=(not is_first_layer and i == 0))(input)
        return input

    return f

def basic_block(filters, init_strides=(1, 1), is_first_block_of_layer=False):
    """Basic 3 X 3 convolution blocks for use on resnets.
    """
    def f(input):
        special_strides=init_strides
        if is_first_block_of_layer:
            special_strides=(2,2)
        regularizer = tf.contrib.layers.l2_regularizer(scale=WEIGHT_DECAY)
        initializer=tf.keras.initializers.he_normal()
        conv1 = tf.layers.Conv2D(filters=filters, kernel_size=(3, 3),
                                 strides=special_strides,
                                 padding="same",
                                 kernel_regularizer=regularizer,
                                 kernel_initializer=initializer)(input)
        conv1 = tf.layers.batch_normalization(conv1,training=isTrain)
        conv1 = tf.nn.relu(conv1)
        regularizer = tf.contrib.layers.l2_regularizer(scale=WEIGHT_DECAY)
        initializer=tf.keras.initializers.he_normal()
        conv1 = tf.layers.Conv2D(filters=filters, kernel_size=(3, 3),
                                 strides=init_strides,
                                 padding="same",
                                 kernel_regularizer=regularizer,
                                 kernel_initializer=initializer)(conv1)
        conv1 = tf.layers.batch_normalization(conv1,training=isTrain)

        convolutions = conv1
        return tf.nn.relu(shortcut(input, convolutions))
```

```
66      return f
```

Sljedeći odsječak prikazuje način na koji nižemo blokove sa filtrima `16`, `32` i `64` jedan na drugi.

```
filters = 16
for i, r in enumerate(repetitions):
    block = residual_block(block_fn, filters=filters, repetitions=r,
    is_first_layer=(i == 0))(block)
    filters *= 2
```

5.4. Augmentacija podataka

Za učenje naših modela je potrebno mnogo podataka kako bi mogli što bolje klasificirati slike. Tehnika koja se često koristi kako bi se povećao skup podataka za treniranje je augmentacija podataka. Nad ulaznim podacima vršimo određene transformacije poput rotacije slike oko horizontalne i vertikalne osi, uvećanje i smanjivanje slike i nasumično izrezivanje slike iz transformirane slike kako bi dobili što robusnije modele. U okviru rada koristit ćemo prilično jednostavnu augmentaciju podataka koja je predložena u [10], koja uz manje izmjene pokazuje dobre rezultate u našoj implementaciji modela. Nad svakom mini-grupom podataka iz originalnog skupa podataka vršimo transformaciju na sljedeći način: svaku sliku iz mini-grupe uz neku fiksiranu vjerojatnost horizontalno zakrećemo. Na zakrenutu sliku sa svake strane dodajemo po `4` piksela. Iz dobivene slike nasumično izrezujemo sliku originalnih dimenzija `32x32x3`. Pokazuje se da na konkretnom problemu ostvarujemo bolje rezultate opisanom, tkzv. online augmentacijom nego offline augmentacijom gdje isti postupak provedemo samo jedanput prije samog treniranja. Online augmentacijom će model svaku sliku vidjeti u nekoliko različitih verzija koje će pomoći da naš model bude robusniji.

5.5. Postupak učenja

Ulazne podatke na početku naše programske izvedbe normaliziramo. Normalizirane podatke dovodimo na ulaz neuronske mreže i propagiramo značajke dobivene aktivacijom kroz mrežu algoritmom unaprijedne propagacije. Na kraju mreže računamo pogrešku koju propagiramo unatrag do prvih slojeva u mreži algoritmom unazadne propagacije i ovisno o pogreškama podešavamo parametre.

U okviru ovog rada nećemo detaljno razmatrati algoritam unazadne propagacije, već ćemo dati kratak pregled kako ažuriranje parametara funkcionira. Često korišten

algoritam za ažuriranje težina je algoritam stohastičkog gradijentnog spusta. Taj algoritam funkcionira na sljedeći način. Izračunamo pogreške svih slojeva u našem modelu. U skladu sa dobivenim pogreškama ažuriramo težine prema sljedećoj formuli:

$$\theta = \theta - \alpha \nabla_\theta J(\theta), \tag{5.2}$$

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i}^{N} \nabla_\theta L(\theta; x(i), y(i)) \tag{5.3}$$

gdje je θ težina koju ažuriramo, α stopa učenja, J iznos pogreške za težinu koju ažuriramo.

U našim modelima koristimo mini grupe, zbog kojih ažuriranje težina obavljamo tek nakon prolaza kroz sve podatke u mini grupi. Takav postupak je bolji od ažuriranja težina nakon svakog podatka jer brže konvergira i brže se izvodi.

U okviru rada koristimo algoritam stohstičkog gradijentnog spusta sa momentumom (engl. *SGD with momentum*). Ovaj algoritam unosi izmjene u odnosu na algoritam SGD, pa tako težine ažuriramo na sljedeći način:

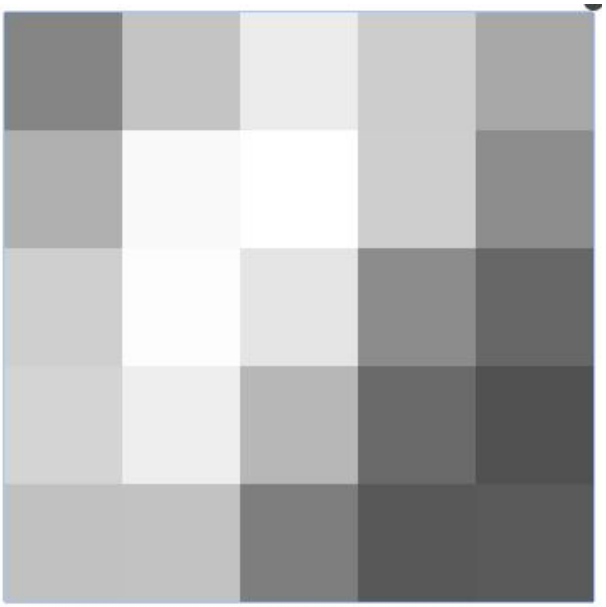
$$v = \gamma v + \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)}) \tag{5.4}$$

$$\theta = \theta - v \tag{5.5}$$

gdje je `v` brzina a γ vrijednost između `0` i `1` i predstavlja faktor kojim reguliramo doprinos brzine a nazivamo ga momentum. Momentum u našem radu postavljamo na `0.9`. Potrebno je pažljivo izabrati momentum jer premala i prevelika vrijednost utječu negativno na brzinu konvergencije. Ovaj algoritam vuče analogiju sa momentumom kakvim ga poznajemo u fizici. Prednost takvog algoritma nad običnim SGD je što postupak učenja ima manju vjerojatnost zapadanja u lokalne optimume funkcije gubitka. Za uspješno treniranje modela sa algoritmom SGD sa momentumom[15] potrebno je pažljivo podesiti hiperparametre i koristiti smislenu inicijalizaciju težina. U ovom radu korsitimo He normal inicijalizaciju težina kako je opisano u [9]. U slučaju nekvalitetne inicijalizacije i neadekvatnih parametara, model vjerojatno neće moći učiti. U okviru rada smo spoznali da je to zaista tako. Često nam je gubitak ispadao prevelik pri samom početku treniranja što je onemogućavalo konvergenciju - program smo morali mnogo puta terminirati i podesiti parametre te ponovno pokušati trenirati model.

Na izlazu našeg modela primjenjujemo softmaks funkciju. Funkcija na temelju izlaza posljedneg sloja računa vjerojatnosnu distribuciju po svim izlaznim klasama, za svaki podatak iz mini grupe. Za svaki podatak i za svaki razred računamo sljedeće:

$$p(y = j|x) = \frac{e^{(w_j^T x + b_j)}}{\sum_{k \in K} e^{(w_k^T x + b_k)}}, \tag{5.6}$$



Slika 5.3: Prikaz naučenog filtra za podatke iz skupa MNIST.

gdje je K skup mogućih razreda. Na temelju takve distribucije, gubitak predstavlja negativna log izglednost točnog razreda, odnosno:

$$L_i = -log(\frac{e^{(w_j^T x + b_j)}}{\sum_{k \in K} e^{(w_k^T x + b_k)}}). \tag{5.7}$$

Kako radimo sa mini grupama, kao konačan rezultat uzimamo sumu negativnih log izglednosti za svaki podatak iz minigrupe. Pa tako konačno za svaku težinu imamo:

$$J(\theta) = -\sum_i log * p(y_i|x_i, \theta). \tag{5.8}$$

6. Rezultati

Kako bi mogli usporediti performanse dubokih konvolucijskih modela sa rezidualnim vezama sa jednostavnim modelima, za početak smo implementirali jednostavan konvolucijski model za klasifikaciju slika iz skupa CIFAR10 i model za klasifikaciju slika iz skupa MNIST. Plitkim modelom nad skupom CIFAR10 ostvarujemo točnost od `73.08%`, dok nad skupom MNIST ostvarujemo rezultat od `98.22%`. Utjecaj rezidualnih dubokih modela smo ispitivali nad skupom CIFAR10.

Konstruirali smo modele sa `20` konvolucijskih slojeva, odnosno parameterom `n=3` i `14` konvolucijskih slojeva, odnosno parametrom `n=2`, kako je opisano u poglavlju 5.3. U okviru opisa rezultata običnim mrežama nazivamo duboke konvolucijske modele bez rezidualnih veza a rezidualnim modelima nazivamo duboke konvolucijske modele sa rezidualnim vezama.

Na rezidualnom modelu sa parametrom `n=2`, odnosno sa `14` konvolucijskih slojeva, ostvarujemo točnost od `87.36%` na testnom skupu. Analognim dubokim modelom bez rezidualnih veza ostvarujemo točnost od `85.04%`. U mreži sa `20` konvolucijskih slojeva ostvarujemo točnost `88.33%` na testnom skupu sa rezidualnom arhitekturom. Sa običnom konvolucijskom mrežom iste dubine ostvarujemo točnost od `86.12%`. Primjećujemo da pliće arhitekture brže konvergiraju ali na manju točnost. U dubljim arhitekturama bez rezidualnih veza uočavamo stagnaciju i degradaciju točnosti koja najvjerojatnije nije posljedica problema umirućih gradijenata - koristimo dosta BN slojeva koji uspješno otklanjaju problem i u unaprijednoj propagaciji značajki i u unazadnoj propagaciji pogrešaka. BN umanjuje problem jer sve značajke povlači na mali raspon oko `0`, odnosno svi ulazni podaci će biti iz distribucije $N(0, 1)$, to jest gradijenti će biti dovoljno veliki da se težine mogu ponovno ažurirati.

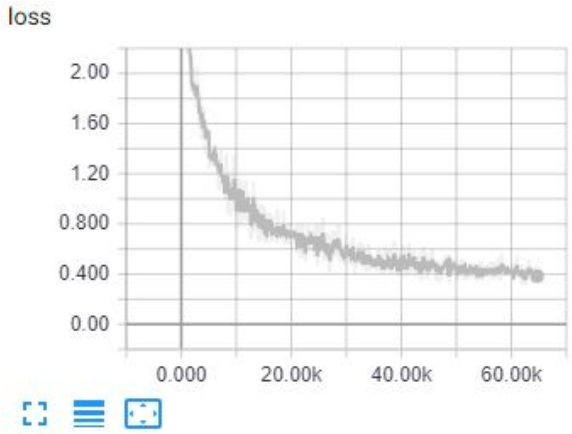
Detaljnije razmatramo ponašanje rezidualne arhitekture sa `20` slojeva. Primjećujemo da model stabilno konvergira prema konačnom gubitku `0.32`, na evaluacijskom skupu postižemo najvišu točnost od oko `89.38%`. Model smo naučili na način da smo pokrenuli učenje sa velikim brojem iteracija. Pratili smo točnost na validacijskom skupu i pamtili broj epoha koji je dao najbolji rezultat. Zatim smo spojili skup

Tablica 6.1: Prikaz postotaka pogreške na različitim modelima na testnom skupu CIFAR10

n	obični	rezidualni
2	14.96	12.64
3	13.88	11.67

za treniranje i validacijski skup i ponovno pokrenuli učenje sa brojem epoha koji je u prethodnom slučaju dao najvišu točnost.

Slika 6.1 prikazuje gubitak na trening skupu tijekom treniranja.



Slika 6.1: Gubitak na skupu za treniranje.

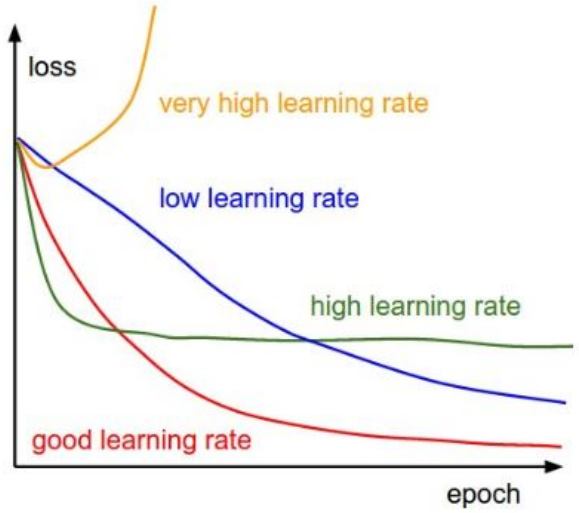
Prilikom učenja našeg modela smo koristili L^2 regularizaciju. Općenito, regularizaciju koristimo jer želimo limitirati kapacitet našeg modela. Naime, kako radimo sa dubokim modelima sa mnogo parametara, postoji velika opasnost da naši modeli nauče napamet klasificirati podatke iz trening skupa i da ne daju dobre rezultate na testnom skupu, odnosno da ne generaliziraju dobro. Regularizacija na neki način kažnjava model što smanjuje prenaučenost. Općenito, regularizacija utječe na gubitak na sljedeći način:

$$\widetilde{J}(\theta; X; y) = J(\theta; X; y) + \beta\Omega(\theta), \tag{6.1}$$

gdje je J gubitak prije regularizacije, $\widetilde{J}$ ukupni gubitak, β iznos parametra koji predstavlja doprinos regularizacije, Ω funkcija regularizacije. U slučaju L^2 regularizacije, na pogrešku dodajemo sumu L^2 gubitaka svih težina pomnoženu sa regularizacijskim faktorom.

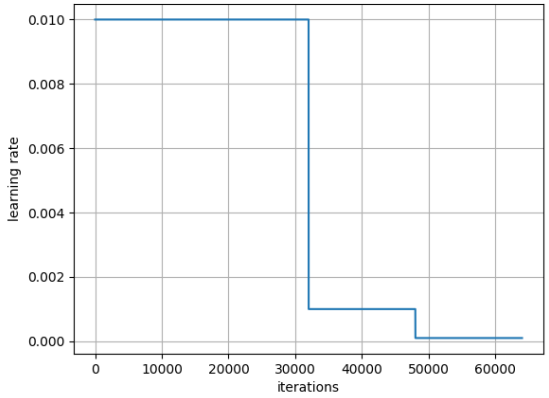
Bitan faktor u učenju modela predstavlja stopa učenja α. Prilikom ažuriranja težina ta stopa nam je bitna jer govori za koliko trebamo ažurirati težine u smjeru gradijenata.

U okviru našeg rada, predstavlja hiperparametar koji postavljamo na početku učenja. Ako stopu postavimo na premalu vrijednost, model će sporo konvergirati - trebat će mu mnogo epoha da nauči. Ako stopu postavimo na veliku vrijednost, naš model u nekom trenutku će prestati učiti jer se neće moći nastaviti približavati minimumu. Ako pretjeramo sa stopom učenja i postavimo je na preveliku vrijednost, model neće uopće učiti i gubitak će eksplodirati, što možemo vidjeti na 6.2.



Slika 6.2: Prikaz kako se gubitak ponaša ovisno o stopi učenja. Slika je preuzeta sa [13]

Stopu učenja smo dijelili sa `10` na `32000.` i `48000.` iteraciji kako bi pomogli modelu da brže konvergira kako je prikazano na 6.3. Takav način postavljanja stope učenja je dobar jer u početku relativno brzo dolazi blizu optimuma - kada dođe do stagnacije smanjimo stopu učenja i model ponovno konvergira prema optimumu.



Slika 6.3: Stopa učenja u rezidualnom modelu.

7. Zaključak

Usporedbom rezultata ostvarenih nad skupom CIFAR10 plitkim konvolucijskim modelom i dubokim konvolucijskim modelima, zaključujemo da gradnjom dubljih konvolucijskih modela uistinu postižemo bolje rezultate klasifikacije. Također, usporedbom običnih dubljih konvolucijskih modela i dubokih rezidualnih modela uočavamo probleme koji se pojavljuju pri konstrukciji običnih modela poput stagnacije točnosti na trening skupu i degradacije točnosti na ispitnom skupu. Modeli sa rezidualnim vezama uspješno rješavaju prepoznate probleme. Uz kombinaciju aktivacijske funkcije zglobnice i slojeva normalizacije grupe koji imaju regularizacijski efekt na model, rezidualna arhitekura ostvaruje znatno bolje rezultate u odnosu na obične konvolucijske mreže jednake dubine, kao i u odnosu na pliće konvolucijske mreže prikazane u uvodnom dijelu rezultata. Primjećujemo da razlika između obične mreže i rezidualne mreže sa `14` slojeva nije toliko istaknuta, rezidualne veze pokazuju svoju snagu na dubljim modelima. Smatramo da je cilj rada postignut - proučen je i argumentiran utjecaj rezidualnih veza na točnost dubokih modela. Uočeni su i obrazloženi problemi koji se pojavljuju u dubokim modelima, te su implementirani ili predloženi postupci kako takve probleme otkloniti. Rezultate koje smo ostvarili smo mogli poboljšati daljnjim poboljšavanjem hiperparametara i boljim tehnikama augmentacije podataka. Za razliku od slučajne inicijalizacije koju koristimo u okviru rada, u praksi se modeli obično predtreniraju na velikim skupovima poput Imagenet-a, čime se obično postižu bolji rezultati.

Daljnji nastavak ovog rada bi mogao teći u smjeru implementacije promjena u rezidualnoj arhitekturi prikazanoj u [11] u kojem je predložena nešto drugačija organizacija pojedinog rezidualnog bloka. Duboki konvolucijski modeli sa rezidualnim vezama predstavljaju današnji doseg u području računalnog vida, navedenim tehnikama se postižu nadljudski rezultati nad skupovima podataka koji su znatno veći problemi [6] od skupova korištenih u ovom radu. Tako na skupu Imagenet profesionalni ljudski označavač podatka ostvaruje top-5 pogrešku od `5.1%`, dok najbolji model ostvaruje najbolju top-5 pogrešku od `3.57%` [7] što je zaista impresivan rezultat i pokazatelj da

neke zadatke danas strojevi rješavaju dovoljno dobro da bi bili svakodnevno korišteni za rješavanje takvih problema. Osim problema računalnog vida, rezidualni modeli se pokazuju kao dobro rješenje i za probleme obrade prirodnog jezika.

LITERATURA

[1] Student notes: Convolutional neural networks (cnn) introduction – belajar pembelajaran mesin indonesia. URL https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/.

[2] Deep learning in a nutshell: Core concepts. URL https://devblogs.nvidia.com/deep-learning-nutshell-core-concepts/.

[3] Rohan: The vanishing gradient problem – a year of artificial intelligence. URL https://ayearofai.com/rohan-4-the-vanishing-gradient-problem-ec68f76ffb9b.

[4] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, i Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[5] Bojana Dalbelo Bašić, Marko Čupić, i Jan Šnajder. Umjetne neuronske mreže, predavanja iz kolegija umjetna inteligencija. URL https://www.fer.unizg.hr/_download/repository/UI_12_UmjetneNeuronskeMreze[1].pdf.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, i L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. U *CVPR09*, 2009.

[7] Samuel F. Dodge i Lina J. Karam. A study and comparison of human and deep learning recognition performance under visual distortions. *CoRR*, abs/1705.02498, 2017. URL http://arxiv.org/abs/1705.02498.

[8] Xavier Glorot, Antoine Bordes, i Yoshua Bengio. Deep sparse rectifier neural networks. U Geoffrey Gordon, David Dunson, i Miroslav Dudík, urednici, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, svezak 15 od *Proceedings of Machine Learning Research*, stranice 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL http://proceedings.mlr.press/v15/glorot11a.html.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. URL http://arxiv.org/abs/1502.01852.

[10] 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.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016. URL http://arxiv.org/abs/1603.05027.

[12] Alex Krizhevsky, Vinod Nair, i Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). URL http://www.cs.toronto.edu/~kriz/cifar.html.

[13] Fei-Fei Li, Andrej Karpathy, i Justin Johnson. Cs231n: Convolutional neural networks for visual recognition 2016. URL http://cs231n.stanford.edu/.

[14] Karen Simonyan i Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL http://arxiv.org/abs/1409.1556.

[15] Ilya Sutskever, James Martens, George Dahl, i Geoffrey Hinton. On the importance of initialization and momentum in deep learning. U *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, stranice III–1139–III–1147. JMLR.org, 2013. URL http://dl.acm.org/citation.cfm?id=3042817.3043064.

POPIS SLIKA

Klasifikacija slika dubokim konvolucijskim modelima s rezidualnim vezama

Sažetak

Klasifikacija slika prirodnih scena je neriješen problem računalnog vida s mnogim zanimljivim primjenama. U posljednje vrijeme najbolji rezultati u tom području postižu se pristupima utemeljenima na dubokim konvolucijskim modelima. Za ovaj rad posebno su zanimljivi nadzirani pristupi gdje je svaka slika skupa za učenje označena semantičkim razredima objekata koje slika sadrži. U okviru rada, bilo je potrebno proučiti dokumentacije programskih okvira Tensorflow i PyTorch te biblioteke programskog jezika Python za rukovanje matricama i slikama. Izrađena je izvedba programskog sustava za učenje i primjenu klasifikacijskog modela. Evaluiran je utjecaj rezidualnih veza na točnost modela. Detaljno su analizirane konvolucijske mreže u okviru dubokog učenja. Prikazani su i ocijenjeni ostvareni rezultati i predložene izmjene za poboljšanje rezultata.

Ključne riječi: strojno učenje, klasifikacija slika, duboki konvolucijski rezidualni modeli, neuronske mreže, CIFAR10, MNIST, PyTorch, Tensorflow

Image classification with deep convolutional models with residual connections

Abstract

Image classification of natural scenes is an unsolved problem of computer vision with many interesting applications. Recently, approaches with deep convolutional models achieve best results in that area. For this paper were specially interesting supervised approaches where every image from training set is labeled with semantic classes of objects which image contains. In this paper documentations of programming frameworks Tensorflow and PyTorch for matrix and image handling were studied. Implementation of programming system for learning and application of classification model has been implemented. Impact of residual connections on model accuracy has been evaluated. Convolutional networks were deeply analized in context of deep learning. Achieved results have been shown and evaluated and suggestions for classification accuracy improvement have been provided.

Keywords: machine learning, image classification, deep convolutional residual models, neural networks, CIFAR10, MNIST, PyTorch, Tensorflow