Konvolucijske neuronske mreže

Džomba, Kristina

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO–MATEMATIČKI FAKULTET
MATEMATIČKI ODSJEK

Kristina Džomba

KONVOLUCIJSKE NEURONSKE MREŽE

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Saša Singer

Zagreb, srpanj 2018.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Računalni vid je područje umjetne inteligencije koje je posljednjih godina postalo iznimno efikasno za rješavanje problema u robotici, zdravstvu i medicini, pametnom upravljanju i nadzoru, razvoju autonomnih vozila i sličnim disciplinama. Zadaće vizualnog raspoznavanja, kao što su klasifikacija slika, te lokalizacija i detekcija objekata, čine temelj pri rješavanju spomenutih problema. Nedavni razvoj konvolucijskih neuronskih mreža (CNN) doveo je do izvanrednih rezultata pri rješavanju ovih novih, vrlo popularnih izazova. Upravo iz tog razloga, konvolucijske neuronske mreže predstavljaju ključni dio algoritama dubokog učenja u području računalnog vida.

Radi boljeg razumijevanja modela konvolucijskih neuronskih mreža, u radu će najprije biti predstavljene klasične neuronske mreže. Vrlo kratko i slikovito bit će prikazani model neurona, arhitektura mreže te metoda kojom neuronska mreža uči. Cilj ovog poglavlja je uvođenje osnovnih pojmova dubokih modela, kao što su slojevi mreže, aktivacijska funkcija, funkcija gubitka i gradijentni spust.

Potom, u drugom poglavlju dajemo detaljan prikaz konvolucijskih neuronskih mreža. Kako bismo dobili jasnu sliku što zapravo predstavlja pojam konvolucije, prezentiramo matematički opis operacije konvolucije na jednodimenzionalnim i dvodimenzionalnim signalima. U nastavku, kroz detaljan opis svakog sloja mreže, dajemo motivaciju za upravo takvu građu modela i uvid u prednosti u odnosu na klasične neuronske mreže. Na kraju, predstavljamo algoritam učenja modela te dajemo metode optimizacije i regularizacije kojima poboljšavamo svojstva i učenje mreže. Od čitatelja se očekuje razumijevanje osnovnih pojmova linearne algebre, vjerojatnosti i statistike.

Za praktični dio rada odabran je zadatak neuronskog prijenosa stila pomoću konvolucijskih neuronskih mreža. Prijedlog da se konvolucijske neuronske mreže upotrijebe kao sredstvo rješavanja danog zadatka pojavio se 2015. godine u članku [4]. Vodeći se zadanim rješenjem iz članka, razvijen je model prijenosa stila, koristeći biblioteku otvorenog koda *Tensorflow*.

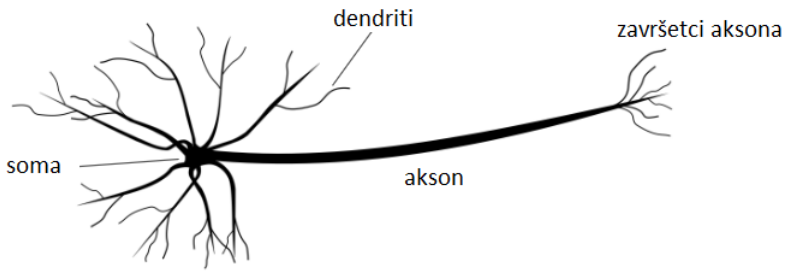
Poglavlje 1

Uvod u klasične neuronske mreže

1.1 Motivacija

Duboko učenje kao grana umjetne inteligencije razvilo se modeliranjem prvih umjetnih neuronskih mreža još 1940-ih. Umjetne neuronske mreže su povezane mreže sastavljene od gradivnih jedinica, raspoređenih u slojeve, koje nazivamo umjetnim neuronima. Ideja modela umjetne mreže je oponašanje rada bioloških neurona mozga. Neuroni mozga međusobno su povezani **sinapsama** i u interakciji su s tisućama drugih neurona koji ih okružuju. Pažljivim izborom arhitekture i hiperparametara umjetne neuronske mreže, cilj je replicirati biološko učenje, koje se temelji na prijenosu informacija između neurona.

Svaki biološki neuron od susjednih neurona prima ulazne signale pomoću **dendrita**. Izlazni modificirani signal zatim se šalje ostalim neuronima preko **aksona**, koji su svojim završetcima povezani s dendritima drugih neurona. Na slici 1.1 možemo vidjeti pojednostavljenu građu biološkog neurona s prikazom osnovnih dijelova.

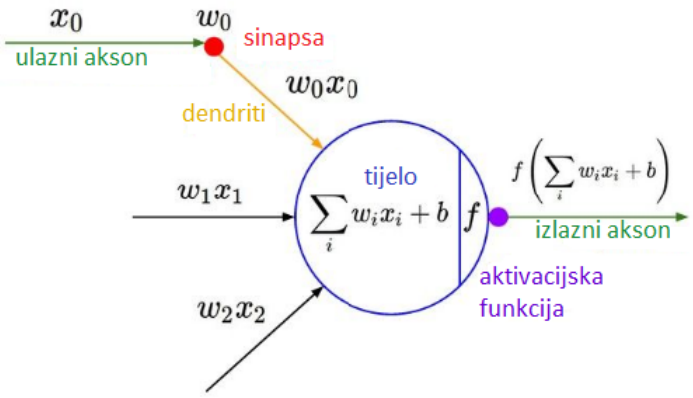


Slika 1.1: Građa biološkog neurona. Neuron prima signal pomoću dendrita te ga preko aksona šalje drugim neuronima.

Klasične neuronske mreže sadrže neurone koji su motivirani građom biološkog ne-

urona, ali s određenim modifikacijama. Iako je model mreže poznat već desetljećima, pravi zamah dobio je tek u posljednjih nekoliko desetljeća, kada se uvodi algoritam propagacije pogreške unatrag, te je time postao vrlo bitan dio umjetne inteligencije.

Slično kao i kod biološkog neurona, umjetni neuron prima ulazne signale od susjednih neurona, koje zatim zbraja, i na zbroj djeluje određenom transformacijom koju nazivamo **aktivacijska funkcija**.



Slika 1.2: Matematički model umjetnog neurona.

Na slici 1.2 možemo vidjeti da signali koji putuju aksonima (npr. $\boldsymbol{x_0}$) dolaze u interakciju s dendritima drugog neurona na način da se množe s **težinama** sinapse. Težine sinapsi $\boldsymbol{w_i}$ predstavljaju parametre modela koje mreža može sama naučiti. Njihova vrijednost eksplicitno utječe na doprinos ulaznog signala ukupnom izlaznom signalu. Svaki neuron računa sumu umnožaka svih ulaznih signala x_i s odgovarajućim težinama w_i, uzimajući u obzir pomak b. Potom na dobivenu sumu djeluje aktivacijskom funkcijom. U ovisnosi o izlaznom signalu, moguće je postaviti granicu odluke koja kazuje hoće li se određeni neuron "aktivirati" ili ne.

U nastavku dajemo egzaktan opis linearne transformacije ulaznih podataka neurona.

Linearna klasifikacija

Ukoliko se radi o linearnoj jedinici, izlazna vrijednost neurona linearno ovisi o njegovim ulaznim podacima. Drugim riječima, možemo reći da na neuronu djeluje linearna aktivacijska funkcija.

Linearna aktivacijska funkcija neurona, u ovisnosti o ulaznom podatku x, definirana je s:

$$f(x, W) = W^T x + b, \tag{1.1}$$

gdje W i b predstavljaju parametre modela.

Nakon računanja linearne aktivacijske funkcije, možemo odrediti uvjete pod kojima će se neuron "aktivirati", odabirom **granice odluke** $k \in \mathbb{R}$:

$$\begin{cases} y = 1, & \text{za } f(x, W) \geq k, \\ y = 0, & \text{za } f(x, W) < k. \end{cases}$$

Ulogu linearne jedinice možemo shvatiti i na drugi način. Neka je pred nama problem binarne klasifikacije, gdje ulazni objekt x želimo svrstati u jednu od dvije zadane klase objekata. Tada neuron ima ulogu binarnog klasifikatora, a njegova izlazna vrijednost $\widehat{y} = f(x, W)$ govori nam u koju klasu je svrstan objekt x, na temelju zadane granice odluke k.

Ipak, korištenje linearne jedinice za rješavanje problema klasifikacije ne daje uvijek najbolje rezultate. Naime, dobivena granica odluke, kojom se objekti svrstavaju u jednu od dvije klase, je linearna, stoga će mnogi objekti biti pogrešno klasificirani. Radi postizanja boljih rezultata, na linearnoj jedinici djeluje se s nekom nelinearnom aktivacijskom funkcijom. Najpoznatiji primjer aktivacijske funkcije za problem binarne klasifikacije je sigmoidna funkcija, koju ćemo opisati kasnije.

U ovisnosti o zadanom problemu, odabiru se različite vrste jedinica, s različitim aktivacijskim funkcijama. Neke od njih opisat ćemo u ovom poglavlju.

Kako bismo mogli kvantificirati nezadovoljstvo izlaznim rezultatom klasifikacije, u nastavku uvodimo pojam **funkcije gubitka**, odnosno, **cijene**.

1.2 Funkcija gubitka

Neka je s R^D označen skup objekata koje je potrebno klasificirati u jednu od K zadanih klasa. U nastavku ćemo takav skup podataka zvati **skupom za učenje** modela. Neka je s $\boldsymbol{x^{(i)}}$ označen i-ti primjer iz skupa za učenje. Svaki objekt $x^{(i)}$ prethodno je označen **labelom** $\boldsymbol{y^{(i)}} \in \{1, \ldots, K\}$, koja označava kojoj klasi objekt $x^{(i)}$ pripada.

Nakon što smo ulazni podatak $x^{(i)}$ prepustili djelovanju klasifikatora, zanima nas u kolikoj mjeri se dobiveni rezultat razlikuje od stvarne klase $y^{(i)}$ kojoj objekt pripada.

Ukoliko se radi o zadaći klasifikacije objekta u jednu od K klasa, izlazni rezultat jedinice je **vektor rezultata** $f(x^{(i)}, W)$ s K elemenata. Element s najvećom vrijednosti predstavlja pripadnu klasu (koja može biti pogrešna).

Funkcija gubitka mjeri odstupanje izlaznog rezultata $f(x^{(i)}, W)$ jedinice, od stvarne vrijednosti $y^{(i)}$ pridružene ulaznom objektu $x^{(i)}$. Gubitak, odnosno, greška bit će veća ako smo objekt potpuno pogrešno klasificirali.

Ulazni podaci su fiksni i nepromjenjivi, međutim, parametri W i b, koji predstavljaju težine i pomak, su varijabilni i na njih možemo utjecati kako bismo minimizirali funkciju gubitka.

U nastavku dajemo pregled najčešće korištenih funkcija gubitka. Primijetimo da izbor funkcije gubitka ovisi o vrsti jedinice koju smo upotrijebili za zadaću klasifikacije.

Unakrsna entropija

Unakrsna entropija (engl. *Cross entropy*) je funkcija gubitka koju primjenjujemo na vektoru rezultata čije su vrijednosti normalizirane na vjerojatnosnu razdiobu.

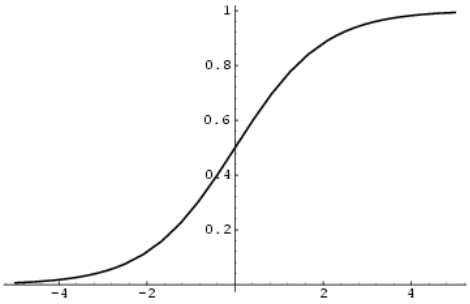
Mnoge zadaće zahtijevaju predviđanje binarne varijable y. Problem binarne klasifikacije može se definirati Bernoullijevom razdiobom nad y, uz uvjet x. Jedinica ima zadaću predviđanja vjerojatnosti $P(y = 1|x)$, stoga vrijednost izlaznog rezultata mora biti iz intervala $[0, 1]$.

Povijesno se za ovu zadaću koristila **sigmoidna aktivacijska funkcija** σ na izlaznom neuronu mreže, definirana s:

$$\widehat{y} = \sigma(z) = \frac{1}{1 - e^{-z}},$$

gdje je $z = f(x, W) = W^T x + b$ linearna transformacija prethodno definirana s (1.1).

Na slici 1.3 možemo primijetiti da se vrijednosti sigmoidne funkcije kreću između 0 i 1, što odgovara potrebama binarne klasifikacije.



Slika 1.3: Sigmoidna funkcija

Metoda rješavanja zadaće binarne klasifikacije vjerojatnosnom razdiobom definiranom sigmoidnom funkcijom još se naziva i **logistička regresija**.

Definiramo **unakrsnu entropiju logističke regresije** na ulaznom podatku x na način:

$$L(\widehat{y}, y) = -(y \log \widehat{y} + (1 - y) \log(1 - \widehat{y})),$$

gdje je $\widehat{y}$ dobiveni rezultat logističke regresije, a y stvarna klasa kojoj objekt x pripada.

U posljednje vrijeme sigmoidna funkcija sve je rjeđe u upotrebi i postaje zastarjela. Pokazano je kako se prilikom učenja neuronske mreže, informacije koje putuju mrežom počinju gubiti na jedinicama sa sigmoidnom funkcijom. Prilikom opisa algoritma učenja mreže, primijetit ćemo kako se prijenos informacija u mreži održava računajući gradijente funkcija. Međutim, gradijenti sigmoidne funkcije vrlo su mali kada se vrijednost funkcije približi nuli ili jedinici. Time se određene informacije gube, a učenje mreže je otežano.

Možemo primijetiti da izlazne vrijednosti sigmoidne funkcije nisu centrirane oko ishodišta, što se, također, pokazalo velikom manom pri učenju mreže.

Za slučaj kada bismo željeli dobiti vjerojatnosnu razdiobu diskretne varijable s $\boldsymbol{K}$ mogućih vrijednosti, koristili bismo **softmax aktivacijsku funkciju**. Zadaću klasifikacije objekta u jednu od K klasa shvaćamo kao generalizaciju binarne klasifikacije sa sigmoidnom aktivacijskom funkcijom.

Softmax jedinica najčešće se upotrebljava na izlazu neuronske mreže i ima ulogu klasifikatora. Vrijednosti vektora rezultata, također, imaju vjerojatnosnu razdiobu i u sumi daju 1. Izlazni rezultat je vektor vjerojatnosti takav da je i-ti element $\widehat{y}_i = P(y = i|x)$.

Softmax funkciju definiramo na sljedeći način:

$$\text{softmax}(z)_i = P(y_i = 1|x) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \tag{1.2}$$

gdje je $z = W^T x + b$, a K broj klasa.

Unakrsna entropija softmax klasifikatora definirana je s:

$$L = \sum_{i=1}^{K} -y_i \log P(y_i = 1|x),$$

gdje y_i predstavlja i-tu klasu. Primijetimo da $P(y_i = 1|x)$ predstavlja vjerojatnost da je softmax jedinica klasificirala objekt u klasu y_i, kao u (1.2).

Ukoliko želimo izračunati ukupni gubitak na svim primjerima iz skupa podataka za učenje, računat ćemo aritmetičku sredinu gubitaka po svim primjerima. Često dobivenoj vrijednosti gubitka dodajemo i gubitak regularizacije $\lambda R(W)$, koji ćemo pobliže objasniti u sljedećem poglavlju.

Definiramo **ukupni gubitak**, odnosno, **funkciju cijene** klasifikacije:

$$J = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x^{(i)}, W)) + \lambda R(W), \tag{1.3}$$

gdje je N kardinalnost skupa podataka za učenje.

1.3 Arhitektura duboke neuronske mreže

Duboke neuronske mreže predstavljaju kolekciju neurona organiziranih u slojeve. Možemo ih promatrati i kao acikličke grafove, jer izlazni rezultat jednog neurona predstavlja ulaz drugoga. Iz tog razloga nazivamo ih i **unaprijednim mrežama**, u kojima se tok informacija prenosi od ulaznog podatka, kroz sve slojeve mreže, sve do izlaznog sloja. Neuronske mreže u kojima informacije teku u oba smjera nazivamo rekurzivnim neuronskim mrežama.

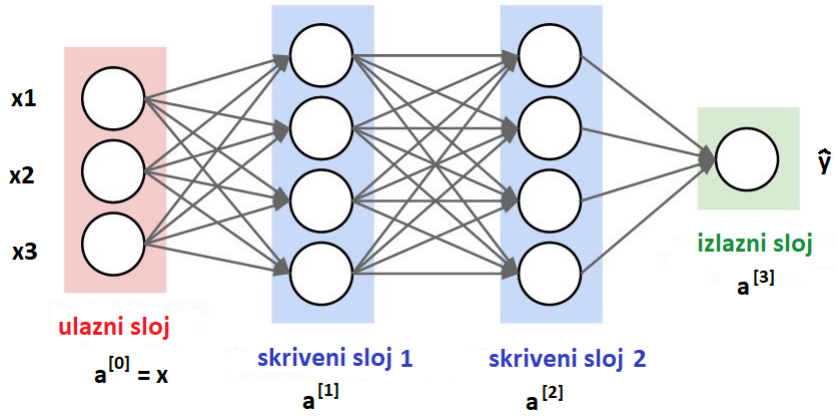
Prvi sloj mreže još se naziva **ulaznim slojem** i njegova uloga je "uvođenje" podatka $\boldsymbol{x^{(i)}}$ u mrežu. Parom $(x^{(i)}, y^{(i)})$ iz skupa podataka za učenje eksplicitno je određeno kojoj klasi izlazni vektor rezultata pridružuje ulazni objekt $x^{(i)}$.

Sve preostale slojeve, izuzev izlaznog sloja, nazivamo **skrivenim slojevima**, jer njihove izlazne vrijednosti još uvijek ne daju zadovoljavajuće rezultate. Svaki skriveni sloj možemo reprezentirati vektorom neurona, čija je zadaća transformacija ulaznih podataka dobivenih iz prethodnog sloja i slanje rezultata neuronima sljedećeg sloja.

Dubinu modela neuronske mreže određuje broj slojeva mreže.

Najčešće su u upotrebi **potpuno povezani slojevi** (tzv. FC slojevi, engl. *Fully connected layer*), u kojima su neuroni između dva susjedna sloja svi međusobno povezani. Na slici 1.4 vidimo prikaz jednostavne arhitekture neuronske mreže. Slojeve redom označavamo indeksima, počevši od nule. Izlazni sloj mreže može se sastojati od samo jednog izlaznog neurona, ili može predstavljati vektor rezultata klasifikacije.

U sljedećem odjeljku prikazat ćemo metodu učenja neuronske mreže pomoću propagacije pogreške unatrag i ažuriranja parametara modela metodom gradijentnog spusta.



Slika 1.4: Primjer duboke neuronske mreže s dva skrivena sloja i izlaznim slojem s jednim neuronom.

1.4 Učenje neuronske mreže metodom gradijentnog spusta

U većini dubokih modela, ideja učenja svodi se na računanje ukupnog gubitka i pokušaj minimizacije gubitka određenim metodama strojnog učenja. Prisjetimo se, parametri mreže na koje algoritam može utjecati su težine W i pomaci b, dok su ulazni podaci (x, y) nepromjenjivi.

Učenje mreže postiže se konačnim nizom iteracija, koje se sastoje od propagacije unaprijed i propagacije pogreške unatrag, s ažuriranjem parametara modela u svakoj iteraciji.

Radi jednostavnosti prikaza, pretpostavimo da model učimo na samo jednom primjeru iz skupa za treniranje.

Algoritam propagacije unaprijed

Algoritam propagacije unaprijed kroz mrežu s ulazom $\boldsymbol{x}$, najprije generira rezultat $\widehat{\boldsymbol{y}}$, koji ukazuje na klasu pripadnosti. Nakon toga, računa se funkcija cijene $\boldsymbol{J}$, čiju vrijednost interpretiramo kao odstupanje dobivenog rezultata od stvarne vrijednosti y.

Algoritam 1 pokazuje propagaciju unaprijed, koja parametrima modela pridružuje gubitak $\boldsymbol{L}(\widehat{\boldsymbol{y}}, \boldsymbol{y})$ vezan za ulazni podatak (x, y), gdje je $\widehat{y}$ izlaz neuronske mreže za dani ulazni podatak x. Uvodimo oznaku $\boldsymbol{\theta}$ za vektor koji sadrži vrijednosti svih parametara modela, odnosno, vrijednosti težina W i pomaka b.

Algoritam 1 Propagacija unaprijed. Funkcija gubitka $L(\widehat{y}, y)$ ovisi o dobivenom rezultatu $\widehat{y}$ i stvarnoj labeli y. Kako bismo izračunali ukupnu cijenu klasifikacije, funkciji gubitka dodajemo vrijednost $\lambda R(\theta)$ koja predstavlja regularizaciju.

Zahtjev: $\boldsymbol{l}$, dubina mreže,
Zahtjev: $\boldsymbol{W}^{[i]}$, $i \in \{1, \ldots, l\}$, matrice težina modela
Zahtjev: $\boldsymbol{b}^{[i]}$, $i \in \{1, \ldots, l\}$, pomaci
Zahtjev: $\boldsymbol{x}$, ulaz za obradu
Zahtjev: $\boldsymbol{y}$, stvarna vrijednost
$a^{[0]} = x$
for $k = 1, \ldots, l$ **do**
    $z^{[k]} = W^{[k]} a^{[k-1]} + b^{[k]}$
    $a^{[k]} = f(z^{[k]})$
$\widehat{y} = a^{[l]}$
$J = L(\widehat{y}, y) + \lambda R(\theta)$

Nakon propagacije unaprijed do posljednjeg sloja mreže, željeli bismo umanjiti vrijednost funkcije cijene $J(\theta)$, odnosno, modificirati parametre modela tako da greška bude minimalna:

$$\theta^* = \arg\min_{\theta} J(\theta).$$

Iz tog razloga, **algoritam propagacije pogreške unatrag** računa gradijent funkcije cijene $\nabla J(\theta)$, koji je korisno sredstvo za otkrivanje **točaka minimuma**.

Upravo u točkama minimuma, gradijent funkcije cijene $J(\theta)$ jednak je nuli. Minimum se može postići **metodom gradijentnog spusta**, kojom se u svakoj iteraciji parametri modela ažuriraju u smjeru "pada" funkcije cijene.

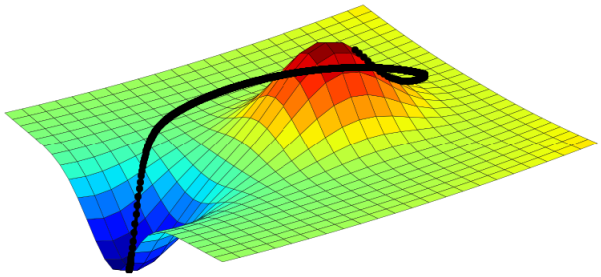
Metoda gradijentnog spusta ažurira parametre modela na sljedeći način:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla J(\theta^{(t)}),$$

gdje t predstavlja redni broj iteracije. Hiperparametar modela α predstavlja **stopu učenja** te označava brzinu učenja, odnosno, ažuriranja parametara u smjeru minimuma. Na slici 1.5 možemo vidjeti primjer djelovanja metode gradijentnog spusta, koja ažurira parametre modela u smjeru minimuma.

Kada u obzir uzmemo da Jacobijeva matrica $\nabla L(\theta)$ sadrži parcijalne derivacije funkcije cijene po svim težinama w_k, pravilo ažuriranja možemo pisati na način:

$$w_k^{(t+1)} = w_k^{(t)} - \alpha \frac{\partial J(w_k^{(t)})}{\partial w_k}, \quad \forall w_k \in \theta.$$



Slika 1.5: Trajektorija gradijentnog spusta s kretanjem prema lokalnom minimumu (*plavo*). Svaka točka trajektorije predstavlja jednu iteraciju učenja.

Algoritam propagacija pogreške unatrag

Propagaciju pogreške unatrag, od izlaznog sloja mreže, pa sve do ulaznog sloja, moguće je izračunati primjenom **lančanog pravila** deriviranja.

Definicija 1.4.1. *Neka su* $g : \mathbb{R}^m \to \mathbb{R}^n$ *i* $f : \mathbb{R}^n \to \mathbb{R}$ *realne funkcije. Ako* $y = g(x)$ *i* $z = f(y)$*, tada vrijedi lančano pravilo deriviranja:*

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}.$$

Primjernom lančanog pravila iz prethodne definicije na funkciju cijene J, gradijente u smjeru težina W i pomaka b sada je lako računati.

Algoritam 2 prikazuje propagiranje dobivene pogreške unatrag. Algoritam računa gradijente aktivacija $a^{[k]}$ u svakom sloju k, počevši od izlaznog sloja l, propagirajući dobivene vrijednosti gradijenata sve do prvog skrivenog sloja. Dobiveni gradijenti mogu se odmah koristiti za ažuriranje težina i pomaka, kao dio metode **stohastičkog gradijentnog spusta**, ili se mogu koristiti s drugim optimizacijskim metodama.

Algoritam 2 Propagacija pogreške unatrag kroz duboku neuronsku mrežu.

$g \leftarrow \nabla_{\widehat{y}} J = \nabla_{\widehat{y}} L(\widehat{y}, y)$
for $k = l, l-1, \ldots, 1$ **do**
  $g \leftarrow \nabla_{z^{[k]}} J = g \odot f'(z^{[k]})$
  $\nabla_{b^{[k]}} J = g + \lambda \nabla_{b^{[k]}} R(\theta)$
  $\nabla_{W^{[k]}} J = g a^{[k-1]T} + \lambda \nabla_{W^{[k]}} R(\theta)$
  $g \leftarrow \nabla_{a^{[k-1]}} J = W^{[k]T} g$

Iz algoritma vidimo da se, nakon propagacije unaprijed, najprije računa gradijent funkcije cijene na izlaznom sloju mreže. Nakon toga se, koristeći lančano pravilo, računa gradijent nelinearne aktivacijske funkcije f, gdje operator $\odot$ predstavlja množenje po točkama.

Posljednji korak je računanje gradijenata u smjeru težina i pomaka te propagacija u prethodni sloj mreže.

1.5 Aktivacijske funkcije

Prethodno smo definirali dvije nelinearne aktivacijske funkcije, sigmoidnu i softmax funkciju. Njihova uloga bila je klasifikacija objekta, stoga se većinom upotrebljavaju samo na izlaznom sloju mreže.

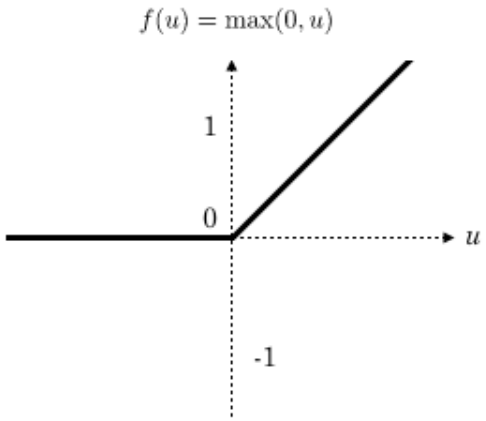
U nastavku navodimo dvije najčešće korištene aktivacijske funkcije koje se koriste na skrivenim slojevima mreže.

ReLU aktivacijska funkcija

ReLU (engl. *Rectified Linear Unit*) vrlo je slična linearnoj aktivaciji pa je njezina optimizacija jednostavna. ReLU aktivacijska funkcija definirana je s:

$$g(z) = \max\{0, z\}, \tag{1.4}$$

gdje je $z = W^T x + b$ afina transformacija.



Slika 1.6: ReLU aktivacijska funkcija

Na slici 1.6 vidimo da funkcija ReLU koristi upravo nulu kao granicu odluke. U praksi pokazuje puno bolje rezultate od sigmoidne aktivacijske funkcije, upravo zbog

svoje "skoro" linearne forme. Za razliku od drugih nelinearnih aktivacijskih funkcija, vrlo je nezahtjevna za računanje, stoga je algoritam učenja brži.

Nažalost, ReLU jedinice mogu biti neotporne na "odumiranje" tijekom algoritma učenja. Primjerice, ukoliko gradijenti velikih vrijednosti prolaze kroz ReLU jedinicu, moguće je da se težine ažuriraju na način da se neuron više nikada ne aktivira. Pametnim izborom stope učenja α moguće je u velikoj mjeri izbjeći takve probleme.

U praksi se sve češće koristi poboljšana verzija funkcije, zvana "Leaky ReLU".

Tanh aktivacijska funkcija

Tanh aktivacijska funkcija definirana je s:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$

gdje je $z = W^T x + b$ afina transformacija.

Funkcija tanh često se koristi kao alternativa sigmoidnoj funkciji zbog centriranosti podataka oko ishodišta.

Poglavlje 2

Konvolucijske neuronske mreže

2.1 Uvod

Konvolucijske neuronske mreže su specijalizirana vrsta neuronskih mreža za pretprocesiranje nestrukturiranih podataka, posebice vezanih za slike, tekst, zvuk i govor. Arhitektura mreže inspirirana je višeslojnim perceptronima u više dimenzija i pokazala se kao najbolji model za izvlačenje bitnih svojstava iz danih nestrukturiranih podataka.

Radom Hubela i Wiesela, nagrađenim Nobelovom nagradom 1981. godine, pokazano je djelovanje stanica vizualnog korteksa mozga mačke na različite podražaje. Naime, Hubel i Wiesel otkrili su da se neuroni korteksa koji su blizu jedan drugome aktiviraju u prisustvu linija pod jednim kutem, dok se druga skupina neurona aktivira kada kut promijenimo. Također, neke skupine neurona prepoznaju rubove, neovisno o kutu pod kojim se nalaze u receptivnom polju.

Njihov rad predstavljao je inspiraciju pri modeliranju konvolucijskih neuronskih mreža, što će biti vidljivo u nastavku.

Ime "konvolucijske neuronske mreže" ukazuje na primjenu matematičke operacije zvane **konvolucija**. Konvolucija je specijalna vrsta linearnog operatora koju konvolucijske neuronske mreže primjenjuju na barem jednom sloju.

U ovom poglavlju prvo ćemo opisati operaciju konvolucije te dati motivaciju za njezino korištenje u konvolucijskim neuronskim mrežama. Obično se definicija konvolucije u konvolucijskim neuronskim mrežama ponešto razlikuje od njezine definicije u drugim područjima, kao što su inžinjerstvo ili matematika. Nakon toga, dat ćemo detaljan prikaz arhitekture mreže, opisujući svaku vrstu sloja. Također, objasnit ćemo proces učenja mreže na ulaznim podacima. Nakon toga, opisat ćemo neke od najčešće korištenih tehnika optimizacije i regularizacije mreža, koje su se pokazale vrlo efikas-

nima. Na samom kraju poglavlja, prikazat ćemo arhitekturu nekih od najpoznatijih klasičnih konvolucijskih neuronskih mreža, redom kojim su se pojavljivale.

2.2 Konvolucija

Konvolucija je operacija na dvije funkcije s realnim domenama, koja daje modificiranu verziju jedne od dviju originalnih funkcija. U većini slučajeva za operaciju konvolucije koristi se oznaka $*$.

Kako bismo dali predodžbu o tome što je zapravo konvolucija, zamislimo da radimo sa signalima. Tada modificirani signal može imati bolja svojstva od originalnog signala za neku određenu zadaću. Primjerice, ako kao originalni signal koristimo monokromatsku, odnosno, "*crno-bijelu*" 2D sliku i na njoj primijenimo konvoluciju s drugim signalom, zvanim filter ili jezgra, izlazni signal može imati svojstvo sadržavanja svih rubova originalne slike. Rubovi na slici mogu predstavljati granice objekata, promjenu u osvjetljenju, promjene na svojstvu materijala, diskontinuitet u dubini i slično. Znanje o linearnim vremenski invarijantnim sustavima i linearnim sustavima invarijantnima na pomak pogoduje razumijevanju operacije konvolucije. U nastavku ćemo za takve sustave koristiti kratice LTI (*Linear time invariant system*) i LSI (*Linear shift invariant system*).

Neka je $x(t)$ ulazni signal, takav da sustav proizvodi izlazni signal $y(t)$ određenom transformacijom $f(t)$. Tada za $y(t)$ možemo pisati:

$$y(t) = f(x(t)).$$

Definicija 2.2.1. *Neka su $\alpha \in \mathbb{R}$ i $\beta \in \mathbb{R}$ skalari i f zadana transformacija. Kažemo da je neki sustav **linearan** ako su zadovoljena sljedeća dva svojstva:*

- *Skaliranje: $f(\alpha x(t)) = \alpha f(x(t))$,*
- *Superpozicija: $f(\alpha x_1(t) + \beta x_2(t)) = \alpha f(x_1(t)) + \beta f(x_2(t))$.*

Slično, kako bi sustav bio vremenski invarijantan, mora vrijediti:

$$f(x(t-\tau)) = y(t-\tau), \quad \forall \tau. \tag{2.1}$$

U nastavku uvodimo opis konvolucije na jednodimenzionalnim, a zatim i na dvodimenzionalnim signalima.

Konvolucija na jednodimenzionalnim signalima

Za neki sustav kažemo da je vremenski diskretan ako na njemu djeluju funkcije transformacije definirane na diskretnom skupu vremena $\mathbb{N} \cup \{0\}$. U nastavku ćemo takve funkcije nazivati **vremenski diskretnim funkcijama**.

Slično, za sustav kažemo da je vremenski neprekidan ako na njemu djeluju funkcije definirane na $\mathbb{R}$, ili na nekom "neprekidnom" podskupu od $\mathbb{R}$. Primjerice, za neke zadaće, funkcije transformacije mogu biti definirane samo na pozitivnim vrijednostima $\mathbb{R}^+$. Za takve funkcije kažemo da su **vremenski neprekidne funkcije**.

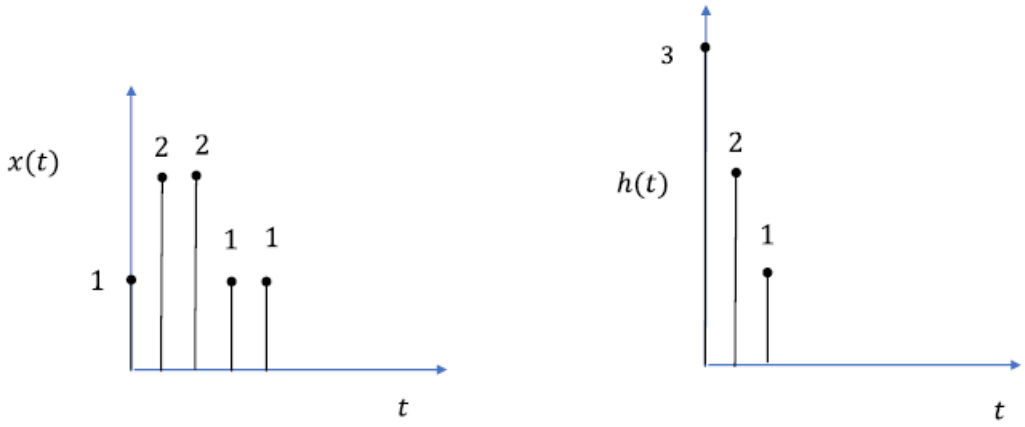
Operaciju konvolucije između dvije funkcije sustava možemo zamisliti i kao mjeru za stupanj preklapanja između jedne funkcije i translatirane i zrcalno okrenute druge funkcije.

Definicija 2.2.2. *Neka su x i h vremenski diskretne funkcije.* ***Konvoluciju*** *funkcija x i h definiramo na način:*

$$y(t) = \sum_{\tau=-\infty}^{\infty} x(\tau)\, h(t-\tau), \tag{2.2}$$

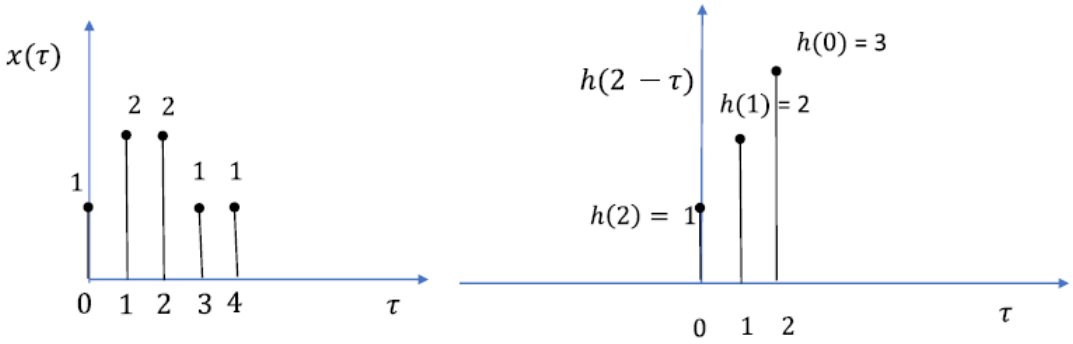
te ju kratko označavamo s $x(t) * h(t)$.

Na slici 2.1 prikazane su dvije vremenski diskretne funkcije x i h, u ovisnosti o vremenu t.

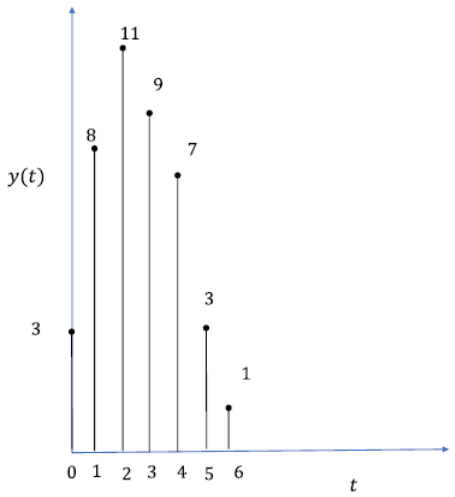


Slika 2.1: Ulazni diskretni signali $x(t)$ i $h(t)$ (izvor [13], str. 156).

Njihovo preklapanje prikazano je na slici 2.2, za fiksnu vrijednost t. Funkcije ulaze u konvoluciju te kreiraju novu diskretnu funkciju $y(t)$, prikazanu u nastavku (slika 2.3).



Slika 2.2: Preklapanje funkcija koje ulaze u konvoluciju, u točki $t = 2$ (izvor [13], str. 156).



Slika 2.3: Izlazni diskretni signal $y(t)$ kao rezultat konvolucije (izvor [13], str. 157).

Uočimo da vrijedi:

- Kada je $t = -1$, težine su dane s $h(-1 - \tau)$, ali nema preklapanja s funkcijom $x(\tau)$ pa je suma 0.
- Kada je $t = 0$, težine su dane s $h(-\tau)$ i jedini element funkcije $x(\tau)$ koji se preklapa s težinama je $x(\tau = 0)$, s preklapajućom težinom $h(0)$, pa je suma $x(0) * h(0) = 1 * 3 = 3$.
- Kada je $t = 1$, težine su dane s $h(1 - \tau)$. Elementi $x(0)$ i $x(1)$ preklapaju se s težinama $h(1)$ i $h(0)$, respektivno. Konvolucija je stoga $x(0)*h(1)+x(1)*h(0) = 1 * 2 + 2 * 3 = 8$.

Slično, ukoliko se radi o vremenski neprekidnim funkcijama, konvoluciju definiramo integralom.

Definicija 2.2.3. *Neka su $x(t)$ i $h(t)$ vremenski neprekidne funkcije. Tada konvoluciju funkcija x i h definiramo na način:*

$$y(t) = \int_{\tau=-\infty}^{\infty} x(\tau)\, h(t-\tau)\, d\tau,$$

te kratko označavamo s $x(t) * h(t)$.

U modelu konvolucijskih neuronskih mreža, prvi argument konvolucije (u našem primjeru, funkcija x) najčešće je **ulazni** podatak, dok je drugi argument (u našem primjeru, funkcija h) **filter**, odnosno **jezgra**. Izlaz takve konvolucije nazivamo **mapa značajki** ili **aktivacijska mapa**.

Također, pri razvoju konvolucijske neuronske mreže, svaki element ulaznog podatka i jezgre pohranjujemo zasebno, pa pretpostavljamo da su te funkcije nula svuda, osim u konačno mnogo točaka čije vrijednosti pohranjujemo. To znači da beskonačna suma iz jednadžbe (2.2) zapravo predstavlja konačnu sumu elemenata polja.

Konvolucija na 2D i 3D signalima

Svaku monokromatsku sliku dimenzije $M \times N$ možemo poistovjetiti s 2D signalom njezinih prostornih koordinata. Signal možemo prikazati kao funkciju

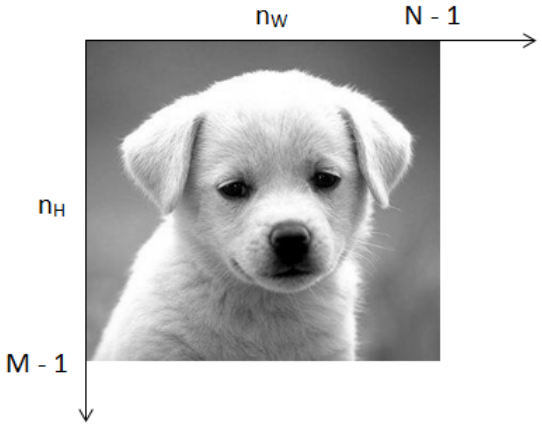
$$x(n_H, n_W), \quad 0 < n_H < M-1, \quad 0 < n_W < N-1,$$

gdje su n_H i n_W diskretne prostorne koordinate na horizontalnoj i vertikalnoj osi, a $x(n_H, n_W)$ predstavlja intenzitet piksela na prostornoj koordinati. Vrijednosti koje pikseli mogu poprimiti su između 0 i 255. Na slici 2.4 možemo vidjeti 2D signal u obliku monokromatske slike.

Obojena slika predstavlja vektor 2D signala. Stoga, za RGB sliku dimenzija $N \times M \times 3$, signal možemo prikazati kao vektor

$$x(n_H, n_W) = [x_R(n_H, n_W), x_G(n_H, n_W), x_B(n_H, n_W)],$$
$$0 < n_H < M-1, \quad 0 < n_W < N-1,$$

gdje x_R, x_G, x_B označavaju intenzitet piksela obzirom na crveni, zeleni i plavi kanal boja.



Slika 2.4: Monokromatska slika kao diskretni 2D signal.

Sada kada smo sliku prikazali kao 2D signal, željeli bismo ju obraditi kroz 2D konvoluciju. Kako bismo iz slike dobili različite željene značajke, možemo ju konvoluirati s tzv. 2D filterima sustava za procesiranje slika. Poželjne značajke koje bismo htjeli dobiti mogu se dobiti uklanjanjem vidljivih šumova iz signala kroz filtere za suzbijanje šumova. Primjerice, za suzbijanje "bijelih šumova" možemo koristiti *Gaussov filter*, koji ćemo opisati kasnije, dok je za detekciju rubova potreban filter koji izdvaja visokofrekventne komponente iz slike.

Dakle, filtere za obradu slika možemo interpretirati kao LSI sustave za obradu slika.

U nastavku ćemo pokazati kako se svaki 2D signal može prikazati pomoću dvodimenzionalne step-funkcije δ.

Heavisideova step funkcija $\delta(n_1, n_2)$ definirana je s:

$$\delta(n_1, n_2) = \begin{cases} 1, & n_1 = 0 \text{ i } n_2 = 0, \\ 0, & \text{inače.} \end{cases} \tag{2.3}$$

Definicija 2.2.4. *Svaki diskretni 2D signal $x(n_H, n_W)$ možemo izraziti kao težinsku sumu funkcije δ iz* (2.3) *po različitim koordinatama, na način:*

$$x(n_H, n_W) = \sum_{k=-\infty}^{+\infty} \sum_{l=-\infty}^{+\infty} x(k, l)\, \delta(n_H - k, n_W - l). \tag{2.4}$$

U općem slučaju, ukoliko LSI sustavu s funkcijom transformacije S proslijedimo signal $x(n_H, n_W)$, definiran pomoću funkcije δ kao u (2.4), dobivamo sljedeće preslikavanje:

$$x(n_H, n_W) \to S(x(n_H, n_W)),$$

gdje je

$$S(x(n_H, n_W)) = S\Big(\sum_{k=-\infty}^{+\infty}\sum_{l=-\infty}^{+\infty} x(k,l)\,\delta(n_H - k, n_W - l)\Big).$$

Primjenom svojstva superpozicije LSI sustava iz Definicije 2.2.1 dobivamo:

$$S(x(n_H, n_W)) = \sum_{k=-\infty}^{+\infty}\sum_{l=-\infty}^{+\infty} x(k,l)\,S(\delta(n_H - k, n_W - l)).$$

Funkciju $f(\delta(n_H, n_W))$ označimo s $h(n_H, n_W)$. Zbog invarijantnosti LSI sustava na pomak definiran s (2.1), vrijedi $S(\delta(n_H - k, n_W - l)) = h(n_H - k, n_W - l)$.

Definicija 2.2.5. ***Konvoluciju 2D slike*** $x(n_H, n_W)$*, dimenzije* $N \times M$*, s 2D filterom* $h(n_H, n_W)$ *definiramo na način:*

$$y(n_H, n_W) = (x * h)(n_H, n_W) = \sum_{k=0}^{M-1}\sum_{l=0}^{N-1} x(k,l)\,h(n_H - k, n_W - l),$$

gdje je $0 \leq n_H \leq M-1$ *i* $0 \leq n_W \leq N-1$.

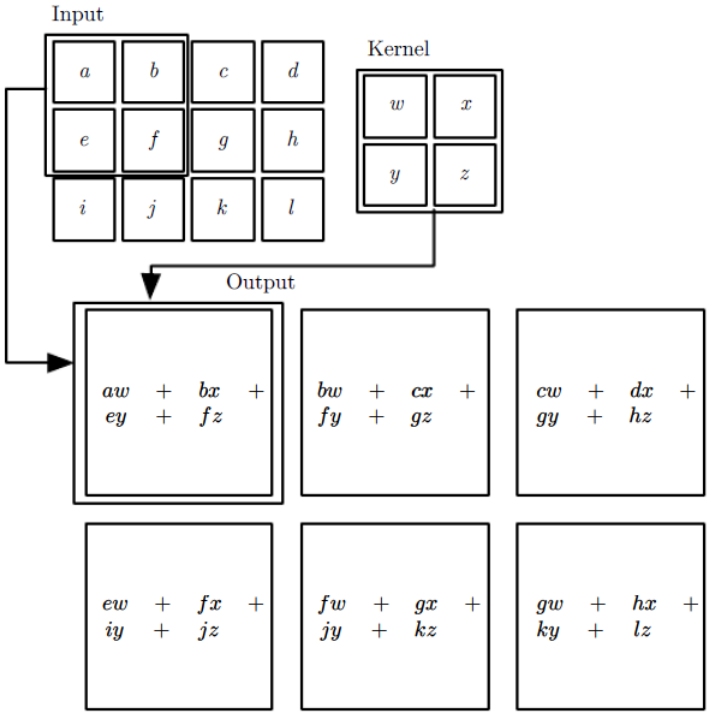
Nadalje, zbog komutativne prirode konvolucije, ekvivalentno je pisati:

$$y(n_H, n_W) = (x * h)(n_H, n_W) = \sum_{k=0}^{M-1}\sum_{l=0}^{N-1} x(n_H - k, n_W - l)\,h(k,l).$$

Komutativno svojstvo proizlazi iz činjenice da smo **okrenuli** filter relativno ulaznoj slici. Unatoč dobrom svojstvu komutativnosti, u primjeni se najčešće koristi verzija konvolucije bez okretanja filtera, zvana **unakrsna korelacija**:

$$y(n_H, n_W) = (x * h)(n_H, n_W) = \sum_{k=0}^{M-1}\sum_{l=0}^{N-1} x(n_H + k, n_W + l)\,h(n_H, n_W).$$

Na slici 2.5 vidljivo je djelovanje konvolucije na ulaznu sliku prikazanu matrično.



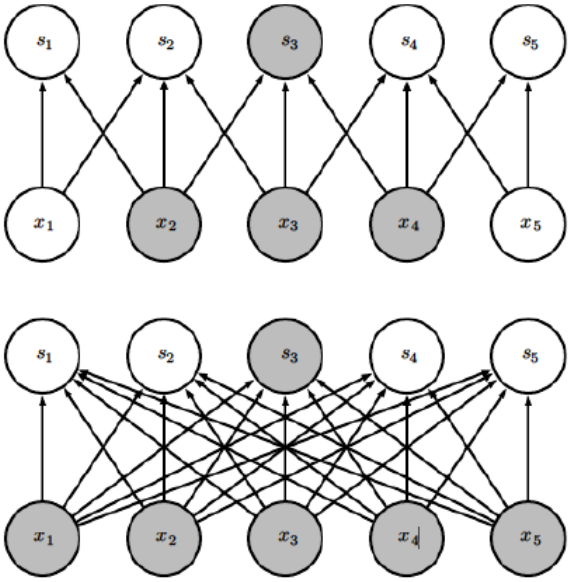
Slika 2.5: Primjer 2D konvolucije bez okretanja filtera. Izlaz konvolucije je 2D matrica (izvor [6], str. 334).

Prednosti u odnosu na klasične neuronske mreže

Konvolucija ima tri vrlo značajna svojstva koja uvelike mogu poboljšati sustav učenja: **raspršenu povezanost**, **dijeljenje parametara** i **translatornu ekvivarijantnost**.

Slojevi klasičnih neuronskih mreža koriste množenja po točkama matrica svih parametara, stoga svaka ulazna jedinica ima interakciju sa svakom izlaznom jedinicom. Dakle, jedan parametar sloja ovisan je o svim parametrima prethodnoga sloja. S druge strane, konvolucijske mreže imaju vrlo rijetku, odnosno, **raspršenu povezanost** težina. Svojstvo raspršenosti postiže se izborom filtera koji su manjih dimenzija od ulazne slike. Unatoč tome što ulazna slika može imati tisuće ili milijune piksela, možemo na njoj prepoznati male, nama bitne značajke, kao što su rubovi objekata, izborom filtera malih dimenzija. Na taj način imamo manje parametara za spremanje u memoriju, stoga će računanje zahtijevati manje operacija. U dubokim konvolucijskim mrežama, jedinice u dubokim slojevima mogu implicitno utjecati na veliki dio ulazne

slike, čime se omogućuje da mreža gradi komplicirane interakcije između objekata, blokovima koji su raspršeno i rijetko povezani.



Slika 2.6: *Gore*: Prikaz raspršene povezanosti u konvolucijskim mrežama. Sivom bojom označen je izlazni neuron s_3 i pripadni ulazni neuroni x_2, x_3 i x_4 koji su na njega imali utjecaj. *Dolje*: Kada je izlaz s_3 generiran množenjem matrica po točkama, svaki ulazni neuron je imao utjecaj na svaki izlazni neuron. Iz tog razloga klasične neuronske mreže nemaju raspršenu povezanost (izvor [6], str. 336).

Pojam **dijeljenja parametara** predstavlja korištenje istih parametara modela više puta. Dok se kod klasičnih neuronskih mreža svaki parametar, odnosno, težina matrice koristi točno jednom pri računanju izlaznog rezultata sloja, u konvolucijskim mrežama, svaka težina u filteru koristi se iznova na svakoj novoj poziciji ulazne aktivacijske mape. Time nam zadaća učenja mreže predstavlja samo učenje težina filtera.

Translatorna ekvivarijantnost je svojstvo konvolucijske mreže uzrokovano određenom vrstom dijeljenja parametara.

Definicija 2.2.6. *Kažemo da su funkcije $f(x)$ i $g(x)$ translatorno ekvivarijantne ako vrijedi:*

$$f(g(x)) = g(f(x)).$$

Definiciju 2.2.6 možemo protumačiti na sljedeći način: Ukoliko je funkcija $g(x)$ takva da translatira ulaznu sliku za određeni broj piksela, tada će i izlazni rezultat, nakon djelovanja funkcije $f(x)$ na translatiranu sliku, također biti translatiran za isti broj piksela. Operacija konvolucije kreira 2D mape značajki, gdje se određene značajke ulazne slike javljaju na cijeloj slici. Primjerice, rubovi objekata su značajka koja će se vjerojatno pojavljivati na cijeloj slici, stoga je korisno dijeliti parametre (filter za rubove) na cijelom području slike. Ukoliko je slika translatirana za jedan piksel, informacija o rubovima će i dalje ostati sačuvana nakon operacije konvolucije.

Konvolucija nije ekvivarijantna na neka druga svojstva, kao što su rotacija ili skaliranje.

Najčešće korišteni filteri za obradu slika

Filter srednje vrijednosti

Filter srednje vrijednosti najčešće je korišten pri računanju lokalnog prosjeka intenziteta piksela u svakoj točki. Filter možemo prikazati matricom

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}.$$

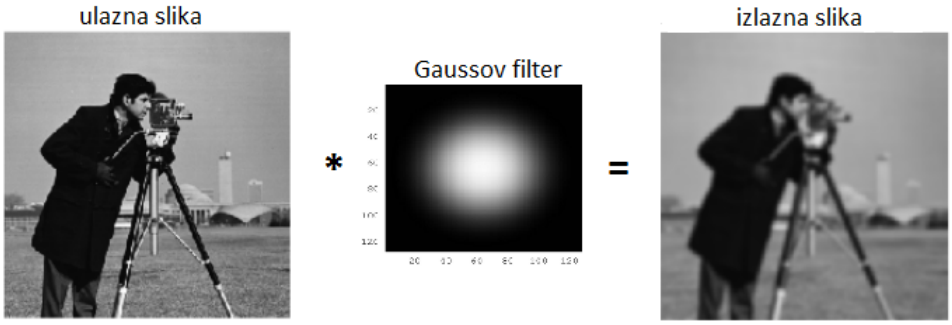
Primjećujemo da će konvolucija u svakoj zadanoj točki prikazivati prosjek svih vrijednosti piksela, odnosno, filter će svakom pikselu pridjeljivati jednaku važnost.

Median filter

Median filter zamjenjuje svaki piksel u okolini s vrijednosti medijana piksela u toj okolini, u ovisnosti o veličini filtera. Djelovanje median filtera pokazalo se vrlo uspješnim kod uklanjanja tzv. "papar-sol šumova", koji se javljaju u formi bijelih i crnih piksela.

Gaussov filter

Gaussov filter je modificirana verzija filtera srednje vrijednosti, gdje vrijednosti težina filtera imaju normalnu razdiobu oko središta. Težine poprimaju veću vrijednost u središtu filtera, a padaju prema rubovima. Svrha ovakve distribucije je eliminacija šumova slike smanjivanjem piksela visokih frekvencija. Kao rezultat primjene Gaussovog filtera na ulaznu 2D sliku dobivamo "zamagljenu" izlazni sliku (vidi sliku 2.7).



Slika 2.7: Konvolucija Gaussovog filtera s ulaznom slikom

Sobel filter za detekciju rubova

Sobel filter registrira horizontalne i vertikalne rubove objekata na ulaznoj slici. Neka su H_x i H_y matrice koje predstavljaju Sobelov filter za prepoznavanje rubova duž horizontalne i vertikalne osi, respektivno. Tada njihove matrične reprezentacije mogu biti:

$$H_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix},$$

$$H_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

Slika 2.8: Originalna slika i slika nakon primjene Sobel filtera za detekciju rubova

2.3 Arhitektura konvolucijske neuronske mreže

U nekoliko sljedećih odjeljaka bit će opisane osnovne sastavnice konvolucijskih neuronskih mreža i njihova uloga.

Na ulazu konvolucijske mreže može biti monokromatska slika ili slika u boji. Ukoliko se radi o slici u boji, prikazujemo ju u tri dimenzije: **visina**, **širina** i **dubina**, pri čemu dimenzija dubine odgovara kanalu boje (crvena, zelena, plava).

Konvolucijsku neuronsku mrežu možemo prikazati kao niz slojeva, pri čemu svaki sloj transformira rezultat prethodnog sloja pomoću određenih diferencijabilnih funkcija i proizvodi izlaz koji nazivamo **mapa značajki** (engl. *Feature maps*).

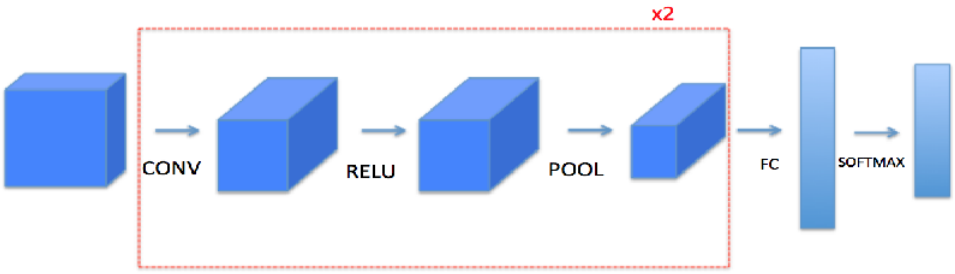
Navodimo osnovne komponente konvolucijske neuronske mreže:

- **Ulazni sloj** predstavlja ulaznu sliku i čuva vrijednosti piksela. Primjerice, slika s visinom 64, širinom 64 i dubinom 3, za crveni, zeleni i plavi kanal boja, ima dimenzije $64 \times 64 \times 3$.
- **Konvolucijski sloj** uzima sliku iz prethodnog sloja te ju konvoluira s određenim brojem filtera, kako bi kreirao izlaznu sliku zvanu **mapa značajki**.
- **Aktivacijska funkcija** je, uglavnom, ReLU funkcija, definirana u prethodnom poglavlju s (1.4). Dimenzija izlaznog volumena nakon primjene ReLU funkcije se ne mijenja. Njezina uloga je dodavanje nelinearnosti u skrivene slojeve mreže. Također, ReLU funkcija djeluje pozitivno na uklanjanje zasićenosti gradijenta za pozitivne ulazne vrijednosti.
- **Sloj sažimanja** djeluje na 2D aktivacijske mape tako da ih sažima te one poprimaju manju visinu i širinu, no dubina aktivacijske mape ostaje očuvana.
- **Potpuno povezani slojevi** sadrže neurone kao i klasična neuronska mreža. Svaki neuron povezan je sa svim neuronima iz prethodnog sloja. Izlazna jedinica je vektor rezultata. Ukoliko se radi o binarnoj klasifikaciji, izlazni sloj generira samo jednu vrijednost.

Na slici 2.9 prikazan je pojednostavljeni model konvolucijske neuronske mreže s ulaznom slikom na početku i nizom transformacija koje se vrše nad ulazom. Primjećujemo kako se veličina i oblik mape značajki mijenjaju kroz slojeve.

Napomena 2.3.1. *U nastavku ćemo za l-ti sloj mreže koristiti notaciju $[l]$. Primjerice, $f^{[l]}$ će označavati dimenziju filtera u l-tom sloju konvolucijske neuronske mreže.*

U nastavku dajemo detaljan opis svakog sloja te prikazujemo njegovu ulogu u konvolucijskoj neuronskoj mreži.



Slika 2.9: Pojednostavljeni model konvolucijske neuronske mreže.

Konvolucijski sloj

Konvolucijski sloj je osnovna građevna jedinica konvolucijskih neuronskih mreža te obavlja veliku većinu transformiranja podataka. U literaturi se vrlo često naziva i skraćeno, **CONV** sloj.

Najčešće se sastoji od dva dijela. U prvom dijelu, ulazna aktivacijska mapa, koja u prvom sloju mreže predstavlja sliku, ulazi u **konvoluciju** s određenim brojem filtera koji prepoznaju različite značajke. Nakon toga, u drugom dijelu, na dobiveni rezultat djeluje se nekom od **aktivacijskih funkcija**. Najučinkovitije djelovanje na raznim zadaćama pokazala je ReLU aktivacijska funkcija.

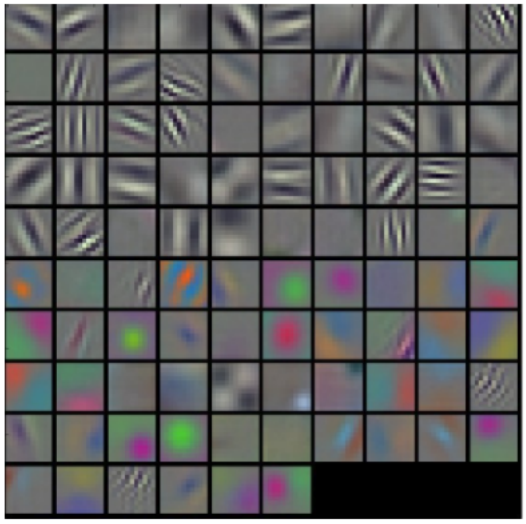
Svaki konvolucijski sloj sadrži unaprijed odabrani broj **filtera**, tzv. **jezgri** fiksne dimenzije, koji pohranjuju vrijednosti težina W.

Konvolucija filtera s mapom značajki odvija se samo na prostornim koordinatama širine i visine mape značajki te predstavlja **Hadamardov produkt** filtera i malog dijela ulazne slike, tzv. receptivnog polja. Ukoliko se radi o slici u boji, za svaki kanal boje uglavnom se primjenjuje isti filter, te se rezultati zbrajaju po kanalima mape značajki.

Konvolucijski slojevi koji su na početku mreže uče prepoznati detalje i strukture. Primjerice, prvi sloj mreže može naučiti prepoznati rubove objekata na slikama. Što se konvolucijski sloj nalazi dublje u neuronskoj mreži, on uči prepoznavati kompleksnije značajke. Tako drugi sloj mreže može prepoznavati jednostavne geometrijske oblike, dok posljednji sloj može prepoznavati lica, pse, automobile i slično.

Vrijednosti težina koje su pohranjene u filterima predstavljaju **parametre** modela te ih mreža može naučiti.

Na slici 2.10 nalazi se vizualizacija filtera prvog CONV sloja poznate naučene konvolucijske mreže AlexNet. Primijetimo kako svaki filter prepoznaje određenu značajku.



Slika 2.10: Vizualizacija vrijednosti težina filtera.

Hiperparametri modela koji se definiraju u konvolucijskom sloju su **broj filtera** $\boldsymbol{d}$, **veličina filtera** $\boldsymbol{f}$, **dopunjavanje** $\boldsymbol{p}$ i **korak** $\boldsymbol{s}$.

Veličina filtera

Veličina filtera definira se visinom i širinom matrice filtera. U većini slučajeva koriste se filteri malih kvadratnih dimenzija. Primjerice, filter dimenzije 3×3 bi imao 9 težina. U pravilu, težine svih filtera u konvolucijskom sloju inicijalizirane su na proizvoljne vrijednosti prije početka učenja mreže.

Neka je na ulazu konvolucijskog sloja slika dimenzija $n \times n$. Slika ulazi u konvoluciju s filterom dimenzije $f \times f$ te proivodi izlaznu aktivacijsku mapu dimenzija

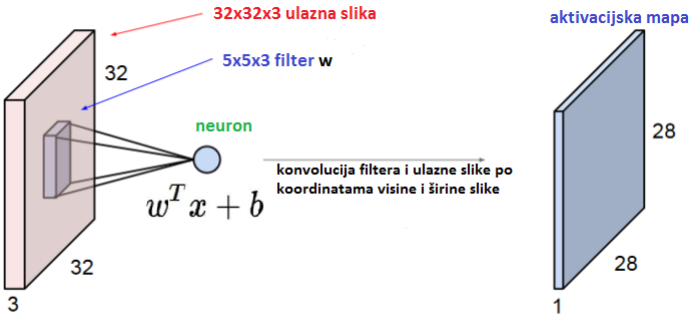
$$(n - f + 1) \times (n - f + 1).$$

Radi bolje predodžbe, rezultat u obliku aktivacijske mape možemo vidjeti na slici 2.11.

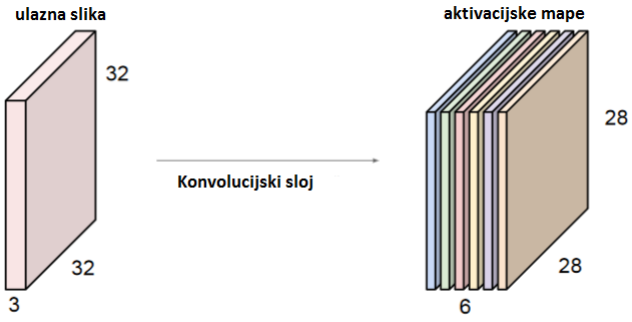
Ako sada svaki filter konvoluiramo s ulaznom slikom, te na rezultat djelujemo ReLU aktivacijskom funkcijom, dobivamo onoliko različitih aktivacijskih mapa koliko smo imali filtera (slika 2.12).

Krajnji rezultat konvolucijskog sloja je 3D volumen, kreiran slaganjem dobivenih aktivacijskih mapa jedne na drugu. Dakle, dimenzije dobivenog izlaznog aktivacijskog 3D volumena su

$$(n - f + 1) \times (n - f + 1) \times d.$$



Slika 2.11: Ulazna slika dimenzija $32 \times 32 \times 3$ ulazi u konvoluciju s filterom dimenzije $5 \times 5 \times 3$. Primijetimo, filter preuzima dimenziju dubine slike. Rezultat djelovanja filtera na receptivno polje slike je broj $w^T x + b$, koji je element izlazne aktivacijske mape (izvor [10]).



Slika 2.12: Djelovanje 6 filtera dimenzija 5×5 na ulaznu sliku dimenzija $32 \times 32 \times 3$ rezultira sa 6 različitih aktivacijskih mapa (izvor [10]).

U nastavku dajemo egzaktnu definiciju djelovanja konvolucijskog sloja na ulaznu mapu značajki.

Definicija 2.3.2. *Neka je s $A^{[l-1]}$ označena izlazna mapa značajki $(l-1)$-og sloja konvolucijske neuronske mreže s visinom $M^{[l-1]}$, širinom $N^{[l-1]}$ i dubinom $C^{[l-1]}$. Neka je djelovanje l-tog konvolucijskog sloja definirano s D filtera dimenzija $f^{[l]} \times f^{[l]} \times C^{[l-1]}$, koji sadrže vrijednosti težina $W^{[l]}$ i pomaka $b^{[l]}$, te neka na njemu djeluje aktivacijska funkcija $g^{[l]}$.*

Definiramo rezultat djelovanja l-tog konvolucijskog sloja na ulaznu mapu značajki $A^{[l-1]}$, pri čemu dobiveni rezultat predstavlja izlaznu mapu značajki $A^{[l]}$ definiranu s:

$$A^{[l]} = g^{[l]}(Z^{[l]}), \tag{2.5}$$

gdje je $Z^{[l]}$ *definirano s*

$$Z^{[l]}_{i,j,c} = \sum_{m=0}^{f^{[l]}-1} \sum_{n=0}^{f^{[l]}-1} \sum_{k=0}^{C^{[l-1]}-1} A^{[l-1]}_{i+m,j+n,k} W^{[l]}_{c,m,n,k} + b^{[l]}_{i,j,c},$$

za $i = 0, \ldots, M^{[l]} - 1,\ \ j = 0, \ldots, N^{[l]} - 1,\ \ c = 0, \ldots, D - 1.$

Lako je vidjeti da su dimenzije nove izlazne mape značajki $A^{[l]}$ iz (2.5) jednake $M^{[l]} \times N^{[l]} \times C^{[l]}$, gdje je

$$\begin{aligned} M^{[l]} &= M^{[l-1]} - f^{[l]} + 1, \\ N^{[l]} &= N^{[l-1]} - f^{[l]} + 1, \\ C^{[l]} &= D. \end{aligned}$$

Dopunjavanje

Dopunjavanje (engl. *Padding*) predstavlja nadopunjavanje originalne mape značajki nulama oko rubova. Time povećavamo prostorni volumen mape, no dobivamo korisna svojstva. Primjerice, elementi na rubovima mape sada više pridonose prijenosu informacija, jer više puta ulaze u konvoluciju s filterom. Također, dopunjavanjem možemo upravljati dimenzijama izlazne aktivacijske mape.

Dva najčešća dopunjavanja obzirom na dimenziju izlazne mape značajki su:

- **Valjano** (engl. *Valid padding*) — mapu značajki ne nadopunjujemo nulama pa se dimenzija izlazne aktivacijske mape mijenja.
- **Nepromijenjeno** (engl. *Same padding*) — mapu značajki nadopunjujemo nulama oko rubova na način da, nakon konvolucije, izlazna aktivacijska mapa ima očuvane dimenzije.

Korak

Korak (engl. *Stride*) predstavlja broj piksela za koji se pomičemo u horizontalnom i vertikalnom smjeru pri konvoluciji aktivacijske mape i filtera. Ukoliko ne želimo preskakati piksele u konvoluciji, korak će biti 1, ali ako želimo odabrati svaki drugi piksel, tada će korak biti 2.

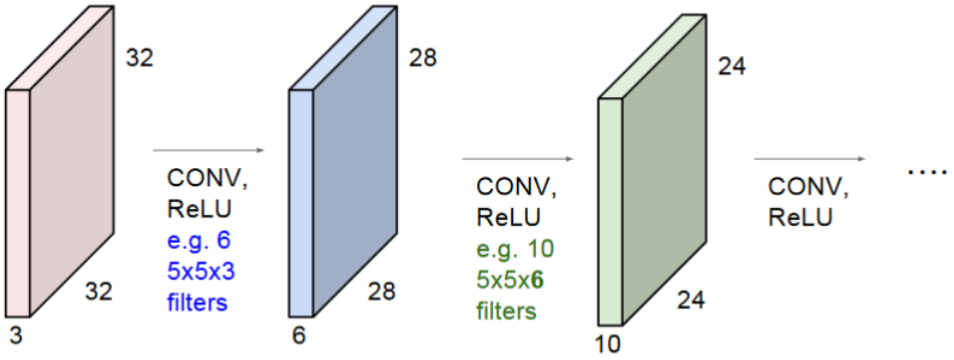
Definicija 2.3.3. *Neka je na ulazu l-tog konvolucijskog sloja 3D volumen dimenzija $M^{[l-1]} \times N^{[l-1]} \times C^{[l-1]}$, te neka su dani hiperparametri sloja: broj filtera D, veličina filtera $f^{[l]}$, dopunjavanje $p^{[l]}$ i korak $s^{[l]}$. Izlazni rezultat konvolucijskog sloja je 3D volumen dimenija $M^{[l]} \times N^{[l]} \times C^{[l]}$, takav da je*

$$M^{[l]} = \left\lfloor \frac{M^{[l-1]} - f^{[l]} + 2p^{[l]}}{s^{[l]}} \right\rfloor + 1,$$

$$N^{[l]} = \left\lfloor \frac{N^{[l-1]} - f^{[l]} + 2p^{[l]}}{s^{[l]}} \right\rfloor + 1,$$

$$C^{[l]} = D.$$

Pri modeliranju konvolucijske neuronske mreže, najčešće se uzastopno koristi više konvolucijskih slojeva, kao što prikazuje slika 2.13. Izlazni rezultat jednog konvolucijskog sloja predstavlja ulaz sljedećeg sloja.



Slika 2.13: Niz od nekoliko uzastopnih konvolucijskih slojeva (izvor [10]).

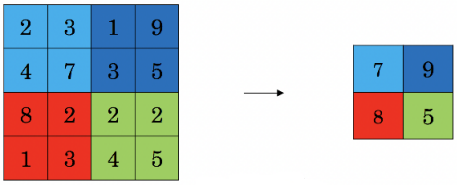
Sloj sažimanja

Najčešće se, pri modeliranju konvolucijske neuronske mreže, pojavljuju tri osnovne faze. U prvoj fazi odvijaju se konvolucije u paraleli, što rezultira aktivacijskim mapama. U drugoj fazi, svaka aktivacijska mapa provodi se kroz postupak nelinearizacije nekom od aktivacijskih funkcija. Ova faza ponekad se naziva i **faza otkrivanja**. U trećoj fazi koriste se **funkcije sažimanja**, kako bi se modificirao aktivacijski sloj.

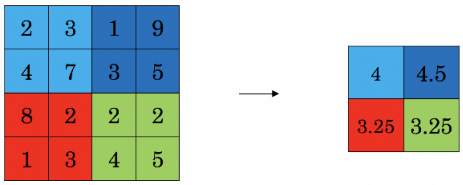
Sloj sažimanja (engl. *Pooling layer*) predstavlja primjenu funkcija sažimanja na aktivacijske mape iz prethodnog sloja, s ciljem umanjivanja njezinih dimenzija.

Operacija sažimanja na ulaznoj slici, uglavnom, primjenjuje sumirane statističke podatke na lokalnom području zadanom veličinom filtera.

Dva najčešća tipa sažimanja su **sažimanje izborom maksimalnog elementa** (engl. *max-pooling*) i **sažimanje prosječnom vrijednošću** (engl. *average-pooling*). U nastavku dajemo vizualizaciju spomenutih tipova sažimanja na slikama 2.14 i 2.15.



Slika 2.14: Sažimanje izborom maksimalnog elementa s filterom veličine 2×2 i korakom 2. Sažimanje se odvija uzimanjem lokalnog maksimuma vrijednosti piksela na receptivnom polju.

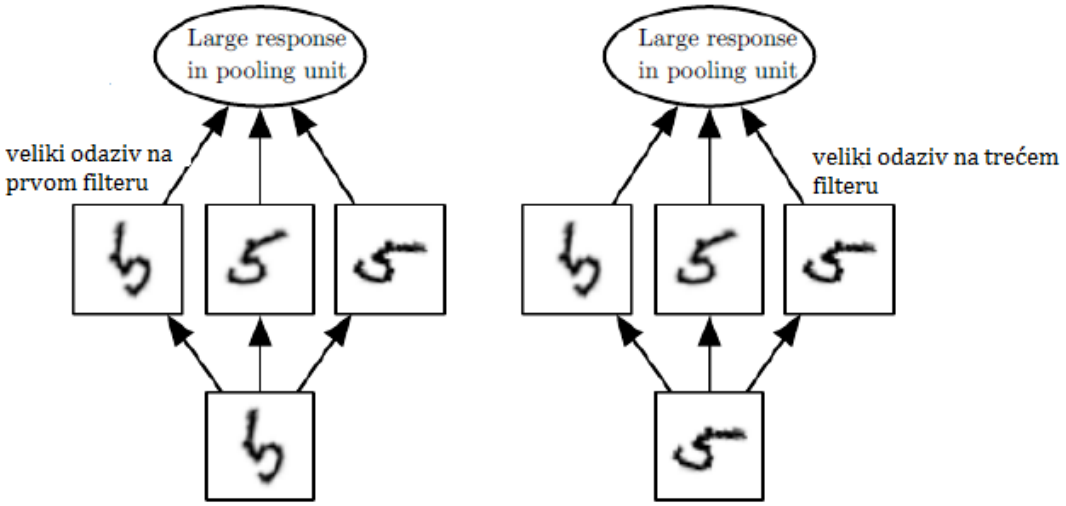


Slika 2.15: Sažimanje prosječnom vrijednošću s filterom veličine 2×2 i korakom 2. Sažimanje se odvija uzimanjem aritmetičke sredine vrijednosti svih elemenata receptivnog polja.

Sloj sažimanja vrlo je koristan pri očuvanju **invarijantnosti prema malim translacijama** piksela. Sama invarijantnost na translaciju znači da, ukoliko ulaznu sliku translatiramo za neki mali iznos, vrijednost izlaznog rezultata nakon sažimanja neće se značajno primijeniti (vidi sliku 2.16). Ovo svojstvo pokazalo se vrlo korisnim kada nam je bitno je li neka značajka na slici, ali nam nije toliko bitno gdje se ona nalazi.

Za mnoge zadaće, sažimanje ima veliku ulogu kod obrade ulaznih slika različitih dimenzija. Primjerice, posljednji sloj sažimanja mreže možemo definirati tako da daje četiri skupa sumiranih statističkih podataka, po jedan skup za svaki kvadrant slike, neovisno o početnoj veličini slike.

Sloj sažimanja može naučiti biti invarijantan na translacije.



Slika 2.16: Primjer naučene invarijantnosti na translacije. Prikazana su tri filtera i jedinica za sažimanje izborom maksimalnog elementa, koja može naučiti biti invarijantna na rotaciju. Filteri su naučeni s odvojenim parametrima kako prepoznati rukom pisanu znamenku 5, tako da svaki filter prepoznaje drugačiju rotaciju znamenke 5. U prvom slučaju, kada se ulazna slika 5 pojavi, aktivira se prvi filter, dok se u drugom slučaju aktivira treći filter. U svakom slučaju, sažimanjem izborom maksimalnog elementa uzet će u obzir najveću vrijednost, odnosno, bit će očuvana informacija o najvećem odazivu filtera (izvor [6], str. 344).

Potpuno povezani sloj

Potpuno povezani sloj nalazi se na samome kraju konvolucijske neuronske mreže. Sastavljen je od neurona u potpunosti povezanih s prethodnim 3D volumenom, kao što je to slučaj kod klasičnih neuronskih mreža. Skraćeno ga nazivamo **FC** sloj (engl. *Fully connected layer*).

Svaki neuron povezan je sa svakim elementom aktivacijske mape iz prethodnog sloja. U praksi se često koristi nekoliko FC slojeva na samom kraju neuronske mreže. Nakon potpuno povezanih slojeva slijedi rezultat, obično prikazan u vektorskom obliku.

Ukoliko se radi o zadaći klasifikacije u jednu od K klasa, tada će izlazni vektor rezultata sadržavati vrijednosti za svaku klasu, kao u poglavlju 1.

2.4 Učenje konvolucijske neuronske mreže

Algoritam propagacije pogreške unatrag

Propagiranje pogreške unatrag kroz konvolucijski sloj vrlo je slično propagaciji unatrag kroz duboku klasičnu neuronsku mrežu. Jedina razlika leži u povezanosti neurona. U konvolucijskoj neuronskoj mreži povezanost neurona je **raspršena** i **rijetka**, jer su iste vrijednosti težina dijeljene između različitih receptivnih polja na aktivacijskoj mapi.

U nastavku prikazujemo propagaciju pogreške unatrag kroz jedan sloj konvolucijske mreže.

Neka je $\boldsymbol{A^{[l-1]}}$ izlazna mapa značajki $(l-1)$-og konvolucijskog sloja. Nakon djelovanja l-tog konvolucijskog sloja s ulazom $\boldsymbol{A^{[l-1]}}$, dobivamo rezultat u obliku nove aktivacijske mape $\boldsymbol{A^{[l]}}$, kao u definiciji 2.3.3 iz prethodnog odjeljka.

U nastavku želimo izračunati vrijednosti **gradijenta** funkcije cijene J, u smjeru težina $W^{[l]}$ i pomaka $b^{[l]}$ u l-tom sloju mreže, kako bismo umanjili ukupni gubitak dobiven propagacijom unaprijed.

Neka je J funckija cijene zadana s (1.3) koju želimo minimizirati. Radi jednostavnosti, zanemarujemo regularizacijski utjecaj na funkciju gubitka. Tijekom propagacije unaprijed, svaki konvolucijski sloj računa aktivacije $A^{[l]}$, koje se zatim propagiraju kroz ostatak mreže, sve do posljednjeg sloja. Naposljetku, računa se vrijednost funkcije cijene J.

Radi preglednosti, označimo s $dW^{[l]}, dZ^{[l]}, dA^{[l]}$ parcijalne derivacije funkcije J po $W^{[l]}, Z^{[l]}$ i $A^{[l]}$, respektivno. Odnosno, neka je:

$$dW^{[l]} = \frac{\partial J}{\partial W^{[l]}},$$
$$dZ^{[l]} = \frac{\partial J}{\partial Z^{[l]}},$$
$$dA^{[l]} = \frac{\partial J}{\partial A^{[l]}}.$$

Tijekom propagacije pogreške unatrag računamo parcijalne derivacije funkcije cijene J po $W^{[l]}$ upotrebom lančanog pravila:

$$\boldsymbol{dW^{[l]}_{c,i,j,k}} = \frac{\partial J}{\partial W^{[l]}_{c,i,j,k}} = \frac{\partial J}{\partial Z^{[l]}_{m,n,c}} \frac{\partial Z^{[l]}_{m,n,c}}{\partial W^{[l]}_{c,i,j,k}}. \tag{2.6}$$

Sada iz (2.6), upotrebom nove notacije i deriviranjem $Z^{[l]}$ po $W^{[l]}$, dobivamo:

$$dW^{[l]}_{c,i,j,k} = \sum_{m=0}^{M^{[l]}-1} \sum_{n=0}^{N^{[l]}-1} dZ^{[l]}_{m,n,c} A^{[l-1]}_{i+m,j+n,k}. \tag{2.7}$$

Iz (2.7) potrebno je izračunati gradijente $dZ^{[l]}$. To činimo na način:

$$\begin{aligned} \boldsymbol{dZ^{[l]}_{i,j,k}} &= \frac{\partial J}{\partial Z^{[l]}_{i,j,k}} = \frac{\partial J}{\partial A^{[l]}_{i,j,k}} \frac{\partial A^{[l]}_{i,j,k}}{\partial Z^{[l]}_{i,j,k}} = \frac{\partial J}{\partial A^{[l]}_{i,j,k}} \frac{\partial}{\partial Z^{[l]}_{i,j,k}} g^{[l]}(Z^{[l]}_{i,j,k}) \\ &= dA^{[l]}_{i,j,k}\, g^{[l]\prime}(Z^{[l]}_{i,j,k}), \end{aligned}$$

gdje vrijednost $\boldsymbol{dA^{[l]}}$ dobivamo iz prethodnog sloja prilikom propagacije unatrag.

U nastavku je još preostalo definirati gradijente $\boldsymbol{dA^{[l-1]}}$ koji će se propagirati prethodnom sloju mreže. U općem slučaju dobivamo:

$$\boldsymbol{dA^{[l-i]}_{i,j,k}} = \frac{\partial J}{\partial A^{[l-i]}_{i,j,k}} = \sum_{\substack{p,m \\ p+m=i}} \sum_{\substack{r,n \\ r+n=j}} \sum_{c=0}^{C^{[l]}-1} W^{[l]}_{c,m,n,k} dZ^{[l]}_{p,r,c},$$

gdje $i = 0, \ldots, M^{[l-1]} - 1$, $j = 0, \ldots, N^{[l-1]} - 1$ i $k = 0, \ldots, C^{[l-1]} - 1$. Preostaje izračunati gradijente pomaka $db^{[l]}$. Dobivamo:

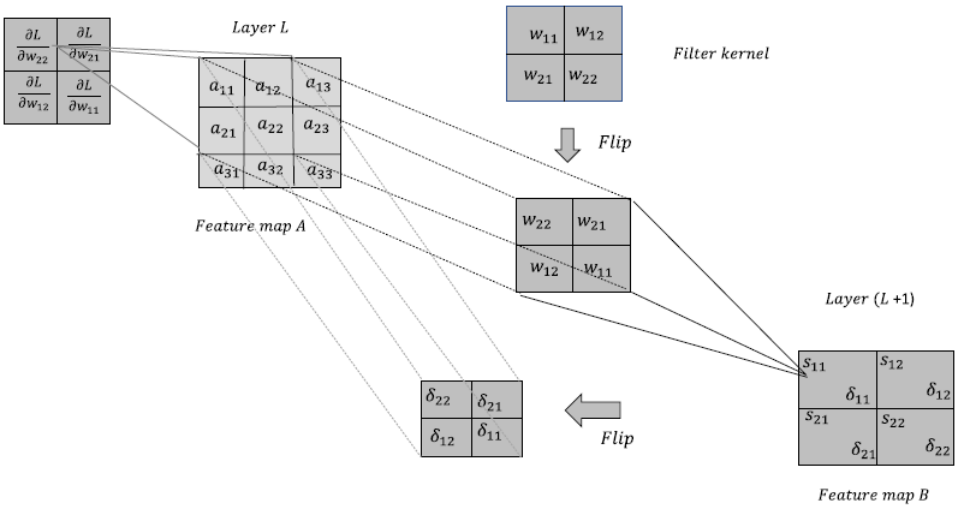
$$\boldsymbol{db^{[l]}_{i,j,k}} = \sum_{m=0}^{M^{[l]}-1} \sum_{n=0}^{N^{[l]}-1} \sum_{k=0}^{C^{[l]}-1} dZ^{[l]}_{m,n,k}.$$

Nakon što smo definirali računanje potrebnih gradijenata $dW^{[l]}$ i $db^{[l]}$, ažuriramo parametre modela $W^{[l]}$ i $b^{[l]}$ u ovisnosti o stopi učenja α:

$$\begin{aligned} W^{[l]}_{c,i,j,k} &\leftarrow W^{[l]}_{c,i,j,k} - \alpha \frac{\partial L}{\partial W^{[l]}_{c,i,j,k}}, \\ b^{[l]}_{i,j,k} &\leftarrow b^{[l]}_{i,j,k} - \alpha \frac{\partial L}{\partial b^{[l]}_{i,j,k}}. \end{aligned}$$

Na slici 2.17 možemo vidjeti djelovanje propagacije pogreške unatrag na jednom konvolucijskom sloju mreže.

Kako bismo model pripremili za učenje, potrebno je pripremiti skup podataka za učenje te inicijalizirati sve parametre mreže. U nastavku prikazujemo osnovne metode **pretprocesiranja** ulaznih podataka, kojim poboljšavamo svojstva modela.



Slika 2.17: Algoritam propagacije pogreške unatrag kroz konvolucijski sloj (izvor [13], str. 182).

Pretprocesiranje podataka

Ulazni podaci za učenje neuronskih mreža nisu uvijek u obliku kakvom bismo željeli da budu, stoga je ponekad neizbježno korištenje sofisticiranih metoda pretprocesiranja podataka.

Željeli bismo da vrijednosti ulaznih podataka budu **normalizirane**, primjerice da vrijednosti piksela budu u rasponu $[0, 1]$ ili $[-1, 1]$. Mnogi modeli dubokog učenja zahtijevaju ulazne podatke standardnih vrijednosti, stoga ih je potrebno na neki način skalirati.

Ostale vrste pretprocesiranja podataka imaju za cilj reprezentirati podatake u kanonskim formama, čime se umanjuje varijacija između pojedinih svojstava. Umanjivanjem varijacije podataka umanjujemo i gubitak modela te mu smanjujemo veličinu.

Ako za učenje mreže imamo veliku količinu raznolikih podataka, tada pretprocesiranje nije nužno, jer će mreža uspjeti naučiti na koja svojstva treba biti invarijantna.

U nastavku opisujemo nekoliko metoda predobrade podataka, koje mogu poboljšati učenje konvolucijske neuronske mreže.

Oduzimanje aritmetičke sredine

Oduzimanje aritmetičke sredine najčešći je oblik pretprocesiranja podataka. Uključuje oduzimanje aritmetičke sredine po svim značajkama na podacima. Geometrij-

ski, ovu metodu možemo opisati kao centriranje oblaka podataka oko fiksnog ishodišta po svim dimenzijama.

Neka je $X^{(i)} \in \mathbb{R}^{r \times c \times 3}$ i-ti primjer slike iz skupa za učenje. Prikažimo sliku kao vektor duljine $n = 3rc$. Tada je nova, normalizirana slika dobivena na način:

$$X^{(i)} = X^{(i)} - \frac{1}{n} \sum_{j=1}^{n} X_j^{(i)}.$$

Također, ponekad je dovoljno oduzimati aritmetičku sredinu piksela samo na crvenom, zelenom ili plavom kanalu boja.

Normalizacija kontrasta

U većini slučajeva, varijacija koju možemo ukloniti iz skupa podataka za učenje je veličina kontrasta. Kontrast predstavlja razliku između svijetlih i tamnih piksela na slici i najčešće ga interpretiramo kao standardnu devijaciju piksela na slici.

Neka je $X^{(i)} \in \mathbb{R}^{r \times c \times 3}$ i-ti primjer slike iz skupa za učenje. Prikažimo sliku kao vektor duljine $n = 3rc$. Definiramo **kontrast** slike $X^{(i)}$ na način:

$$\sigma = \sqrt{\frac{1}{n} \sum_{j=1}^{n} \left(X_j^{(i)} - \bar{X}^{(i)} \right)^2}, \tag{2.8}$$

gdje $\bar{X}^{(i)}$ predstavlja aritmetičku sredinu vrijednosti intenziteta na cijeloj slici:

$$\bar{X}^{(i)} = \frac{1}{n} \sum_{j=1}^{n} X_j^{(i)}.$$

Metodom **normalizacije globalnog kontrasta** (engl. *GCN*) od svake ulazne slike oduzimamo aritmetičku sredinu vrijednosti piksela cijele slike. Nakon toga ju ponovno skaliramo, tako da je standardna devijacija na vrijednostima piksela jednaka nekoj konstanti $\boldsymbol{s}$.

Dijeljenje sa standardnom devijacijom σ iz (2.8) rezultiralo bi samo šumovima ili supresijom određenih značajki, stoga uvodimo malu, pozitivnu vrijednost $\boldsymbol{\lambda}$, kako bismo procijenili željenu standardnu devijaciju s. Također, možemo zadati i granicu za nazivnik, kako ne bi došlo do dijeljenja s nulom. Slike koje u svakom pikselu imaju istu vrijednost su slike s nul-kontrastom, stoga je njihova standardna devijacija nula. Uvođenjem male vrijednosti $\boldsymbol{\epsilon}$ rješavamo taj problem.

Neka je $X^{(i)}$ ulazna slika. Tada metoda normalizacije globalnog kontrasta modificira ulaznu sliku na način:

$$X_j^{\prime(i)} = s \frac{X_j^{(i)} - \bar{X}^{(i)}}{\max\left\{\epsilon, \sqrt{\lambda + \frac{1}{n}\sum_{i=1}^{n}\left(X_j^{(n)} - \bar{X}^{(i)}\right)^2}\right\}}.$$

Ako se skup podataka sastoji od velikih slika koje su "rezane", s ciljem da je na njima neki objekt, tada je sigurno vrijednost λ staviti na nulu, a vrijednost ϵ staviti na jako mali broj, primjerice 10^{-8}. Razlog tome leži u intenzitetu piksela, koji gotovo nikada neće imati konstantnu vrijednost.

Ako u skupu podataka imamo male slike "rezane" nasumično, veća je vjerojatnost da će vrijednosti piksela biti konstantne. U takvom slučaju, možemo postaviti vrijednosti $\lambda = 10$ i $\epsilon = 0$.

Metoda osnovnih komponenata

PCA (engl. *Principal Components Analysis*), odnosno, metoda osnovnih komponenata je metoda **umanjivanja dimenzionalnosti** podataka s ciljem očuvanja informacija o podacima.

Obično se slike sastoje od piksela, čije su vrijednosti međusobno visoko korelirane s pikselima u njihovom bližem susjedstvu. To svojstvo je redundantno za učenje dubokih modela, stoga je poželjno lišiti ga se, odnosno, ukloniti korelaciju između takvih piksela, kako bi oni međusobno bili statistički neovisni.

Cilj ove metode je ukloniti korelaciju između elemenata i postići da varijanca bude jednaka na svim dimenzijama u novoj reprezentaciji podataka. Pritom, želimo podatke linearno projicirati, tako da greška bude minimalna.

Neka je $X^{(i)} \in \mathbb{R}^{r\times c\times 3}$ i-ti primjer slike iz skupa za učenje. $X^{(i)}$ prikažimo kao vektor duljine $n = 3rc$. Za svaku sliku iz skupa podataka za učenje, najprije uklanjamo aritmetičku sredinu vrijednosti piksela na slici i time skup podataka centriramo oko ishodišta. Dobivamo:

$$X^{(i)} = X^{(i)} - \frac{1}{n}\sum_{j=1}^{n} X_j^{(i)}, \quad i = 1, \ldots, m,$$

gdje je m veličina skupa podataka za učenje. Kada smo normalizirali podatke, računamo **matricu kovarijance** $\boldsymbol{C}$ na način:

$$C = \frac{1}{m}\sum_{i=1}^{m} X^{(i)} X^{(i)T}.$$

U nastavku uvodimo pojam **dekompozicije singularnih vrijednosti** matrice.

Neka je A realna matrica dimenzija $m \times n$. Može se pokazati da za matricu A postoji dekompozicija definirana na način $A = U\Sigma V^T$. Ortogonalna matrica U, dimenzija $m \times m$, sadrži stupce u_i, koje nazivamo lijevi singularni vektori, dok ortogonalna matrica V, dimenzija $n \times n$, sadrži desne singularne vektore v_i. Matrica Σ je dijagonalna matrica s nenegativnim vrijednostima σ_i na dijagonali. Za detaljnije objašnjenje pogledati materijale [11] o metodi osnovnih komponenti.

Za dobivenu matricu kovarijance C radimo dekompoziciju singularnih vrijednosti

$$C = UDV^T.$$

Jer je C pozitivno definitna simetrična matrica, zapravo je $U = V$. Tada su stupci matrice U svojstveni vektori matrice kovarijance. Na dijagonali matrice D nalaze se svojstvene vrijednosti matrice kovarijance, koje predstavljaju varijance podataka u smjeru svojstvenih vektora. Na svim ostalim mjestima u matrici D su nule, jer ona predstavlja matricu kovarijance za nekorelirane smjerove zadane ortogonalnim svojstvenim vektorima.

Korištenjem PCA metode projiciramo podatke u smjerovima svojstvenih vektora i njihovu projekciju dijelimo s korijenom svojstvene vrijednosti, odnosno, sa standardnom devijacijom u smjeru projiciranja podataka. Dobivamo:

$$T = D^{-\frac{1}{2}}U^T.$$

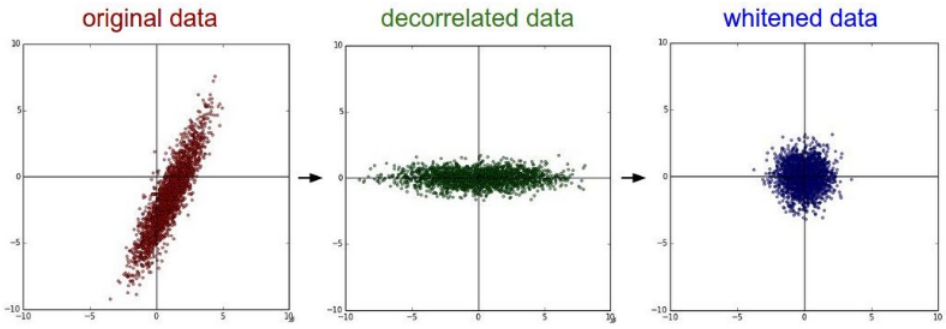
Kada smo na taj način transformirali podatke, dobivamo nekorelirani skup podataka i konstantnu varijancu (vidi sliku 2.18). Transformirani podaci su oblika:

$$X_{pw}^{(i)} = TX^{(i)}.$$

Dobivenom transformacijom T, zapravo, mijenjamo prostor podataka, odnosno, podaci su sada u transformiranom rotiranom prostoru, što nije poželjno svojstvo kada su u pitanju konvolucijske neuronske mreže. Naime, rotacijom gubimo korisne informacije o prostornoj orijentaciji slika pa koristimo modificiranu verziju transformacije T. Ukoliko uzmemo ortogonalnu matricu R i pomnožimo ju s T, ponovno proizvodimo transformaciju izbjeljivanja. Neka je $R = U$. Sada transformacija podatke ostavlja u istom prostoru, ali uklanja korelaciju između elemenata na način:

$$Z = UT = UD^{-\frac{1}{2}}U^T. \tag{2.9}$$

PCA izbjeljivanje se u modeliranju konvolucijskih mreža u praksi rijetko koristi, zbog zahtjevnog računanja dekompozicije singularnih vrijednosti. Metodu navodimo zbog potpunosti.



Slika 2.18: *Lijevo*: Originalni 2D ulazni podaci. *Sredina*: Podaci centrirani oduzimanjem aritmetičke sredine. *Desno*: Podaci skalirani metodom izbjeljivanja (izvor [10]).

Inicijalizacija težina

Vrijednosti težina predstavljaju parametre konvolucijske neuronske mreže. Korak nakon pretprocesiranja ulaznih podataka je inicijalizacija parametara modela, koji će se naposljetku ažurirati u svakoj iteraciji prilikom učenja. U nastavku navodimo neke vrste inicijalizacija težina koje se u praksi najčešće koriste.

Napomena 2.4.1. *Primijetimo, inicijalizacija svih težina nekom konstantnom vrijednošću ne bi bila dobar izbor. Ako svaki neuron računa isti izlaz, tada će i vrijednosti gradijenta biti jednake, čime ne uvodimo nikakvu asimetriju nad podacima.*

Inicijalizacija malim nasumičnim vrijednostima

Želimo da vrijednosti težina budu vrlo blizu nule, ali da ne budu nula. Time razbijamo simetriju, jer naposljetku očekujemo da naučene težine budu i pozitivne i negativne vrijednosti. Najčešće upotrebljavamo inicijalizaciju s vrijednostima iz Gaussove distribucije s očekivanjem 0 i standardnom devijacijom 1.

Problem koji se može pojaviti primjenom ove metode je što distribucija izlaznih vrijednosti neurona ima rastuću varijancu s povećanjem broja ulaznih elemenata. Kako bismo riješili taj problem, možemo normalizirati varijancu izlaza svakog neurona na vrijednost 1.

Neka je $y = \sum_{i=1}^{n} w_i x_i$ skalarni produkt težina w i ulaza x, koji predstavlja izlaznu vrijednost neurona prije primjene nelinearne aktivacijske funkcije.

Varijancu izlaza y možemo prikazati na sljedeći način:

$$\begin{aligned}\operatorname{Var}[y] &= \operatorname{Var}\left[\sum_{i=1}^{n} w_i x_i\right] \\ &= \sum_{i=1}^{n} \operatorname{Var}[w_i x_i] \\ &= \sum_{i=1}^{n} \Big((\mathrm{E}[w_i])^2 \operatorname{Var}[x_i] + (\mathrm{E}[x_i])^2 \operatorname{Var}[w_i] + \operatorname{Var}[x_i] \operatorname{Var}[w_i]\Big) \\ &= \sum_{i=1}^{n} \operatorname{Var}[x_i] \operatorname{Var}[w_i] \\ &= n \operatorname{Var}[w] \operatorname{Var}[x].\end{aligned}$$

U trećem koraku, pretpostavili smo da su očekivanja ulaza x i težina w jednaka nula, $\mathrm{E}[x_i] = \mathrm{E}[w_i] = 0$ (ne mora uvijek biti slučaj). Zbog pretpostavke o jednakoj distribuciji vrijednosti w_i i x_i, slijedi posljednji redak jednadžbe.

Može se pokazati da za slučajnu varijablu X i skalar a vrijedi jednakost:

$$\operatorname{Var}[aX] = a^2 \operatorname{Var}[X]. \tag{2.10}$$

Želimo li postići da je varijanca izlaznih vrijednosti y jednaka varijanci ulaznih vrijednosti x, potrebno je podijeliti inicijalizirane težine s $1/n$, gdje je n broj ulaznih podataka neurona. Sada iz (2.10) slijedi da skalar a mora imati vrijednost $\sqrt{1/n}$.

U članku [5], X. Glorot i Y. Bengio pokazali su da mreže koje koriste ReLU aktivacijsku funkciju bolje konvergiraju ako vrijednosti težina w inicijaliziramo iz uniformne razdiobe U:

$$w \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}\right],$$

gdje su n_{in} i n_{out} broj ulaznih i izlaznih jedinica neurona, respektivno.

Normalizacija malim serijama podataka

Metodom stohastičkog gradijentnog spusta distribucija ulaznih vrijednosti svakoga sloja mijenja se zbog ažuriranja vrijednosti težina u svakom prethodnom sloju. Ta činjenica otežava učenje vrlo dubokih neuronskih mreža, jer je svaki ulazni podatak sloja ovisan o vrijednostima parametara iz svih prethodnih slojeva mreže.

Ako za aktivacijsku funkciju koristimo *sigmoidnu* funkciju ili *tanh*, tada će vrijednosti gradijenata biti "dobre" za učenje mreže samo za određene ulazne vrijednosti. Na ostalim dijelovima, gradijenti će poprimati vrijednosti vrlo blizu nule, što će rezultirati vrlo sporim učenjem.

Jedno rješenje problema je korištenje druge aktivacijske funkcije, koja se bolje ponaša za razne distribucije ulaznih vrijednosti, odnosno, koja ima gradijente koji ne iščezavaju.

Drugo rješenje predstavili su S. Ioffe i C. Szegedy, 2015. godine, u članku [7], gdje uvode pojam **internog kovarijantnog pomaka**. Pojam označava mijenjanje distribucije ulaznih podataka.

Normalizacija malim serijama podataka (engl. *Mini-batch Normalization*) je metoda umanjivanja internog kovarijantnog pomaka. Ulazni podaci se, na početku učenja, normaliziraju na Gaussovu razdiobu.

U svakom sloju mreže, na maloj grupi podataka procjenjuju se očekivanje i varijanca. S druge strane, za predviđanje pri testiranju primjenjuje se cijeli skup podataka.

Neka je $x^{(k)} \in \mathbb{R}^{n\times 1}$ ulazni vektor aktivacija iz prethodnih slojeva. Tada, na svakoj maloj seriji podataka s m aktivacijskih vektora, normaliziramo vektor $x^{(k)}$ na način:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}] + \epsilon}}, \quad k = 1, \ldots, m, \tag{2.11}$$

gdje je

$$\mu_B = \frac{1}{m}\sum_{k=1}^{m} x^{(k)},$$

$$\sigma_B^2 = \frac{1}{m}\sum_{k=1}^{m} \left(x^{(k)} - \mu_B\right)^2.$$

Nakon prethodnog koraka normalizacije aktivacijskih vektora, slijedi njegova linearna transformacija parametrima $\boldsymbol{\gamma}$ i $\boldsymbol{\beta}$, prije same primjene aktivacijske funckije:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}. \tag{2.12}$$

Uvođenjem ovih parametara neuronska mreža može naučiti prepoznati koje vrijednosti aktivacijskih vektora joj više odgovaraju. Primjerice, ukoliko mreža nauči da su originalne vrijednosti aktivacijskih vektora bile bolje za učenje od transformiranih, može povratiti stare vrijednosti vektora mapiranjem.

Parametre μ_B, σ_B^2, γ i β mreža može naučiti algoritmom propagacije unatrag, kao što uči i ostale parametre mreže. Dakle, ukoliko mreža želi povratiti originalne parametre, primijenit će mapiranje pomoću naučenih vrijednosti $\gamma^{(k)} = \text{Var}[x^{(k)}]$, $\beta^{(k)} = \text{E}[x^{(k)}]$.

Obično operacije iz (2.11) i (2.12) objedinjujemo u novi sloj mreže, koji se nalazi nakon konvolucijskih slojeva, prije primjene aktivacijskih funkcija, ili nakon potpuno povezanog sloja.

2.5 Optimizacija

Optimizacija dubokih neuronskih mreža još uvijek je predmet istraživanja i proučavanja te predstavlja vrlo zahtjevnu zadaću. Primjenom određenih metoda strojnog učenja moguće je izbjeći određene poteškoće koje se javljaju pažljivim izborom arhitekture modela. Unatoč tome, neki problemi su neizbježni pa je neophodna upotreba optimizacijskih metoda.

U nastavku će biti opisane neke od najčešćih poteškoća pri učenju dubokih neuronskih mreža. Nakon toga, bit će prikazane neke od optimizacijskih metoda koje su se pokazale vrlo učinkovitima.

Važno je napomenuti da se spomenuti problemi i optimizacijske metode ne odnose samo na konvolucijske neuronske mreže, nego su karakteristični za sve vrste dubokih neuronskih mreža.

Problemi pri učenju dubokih neuronskih mreža

Problem lokalnog minimuma

Konveksne funkcije imaju vrlo bitno svojstvo, koje problem optimizacije jednostavno reducira na problem pronalaska lokalnog minimuma. Naime, svaki lokalni minimum ujedno predstavlja i globalni minimum, zbog svojstva konveksnosti.

S druge strane, ukoliko se radi o **nekonveksnim** funkcijama, kao što je funkcija gubitka neuronske mreže, moguća je velika "zakrivljenost", koja rezultira s više ili beskonačno mnogo lokalnih minimuma.

Problem pri učenju mreže s nekonveksnom funkcijom gubitka nastaje kada lokalni minimumi imaju velik gubitak u odnosu na globalni minimum. Ako mreža koristi stohastički gradijentni spust kako bi ažurirala parametre, moguće je da učenje mreže stane na nekom lokalnom minimumu koji nije optimalno rješenje. Svojstvo nekonveksnosti neuronskih mreža posljedica je nelinearnosti skrivenih slojeva mreže.

Ovaj problem i danas je predmet proučavanja, no poznato je da većina dubokih neuronskih mreža ima male vrijednosti funkcije gubitka u lokalnim minimumima,

stoga traženje globalnog minimuma nije od velike važnosti. Naime, ponekad je dovoljno pronaći točku u prostoru koja ima vrlo nisku cijenu, unatoč tome što nije globalni minimum.

Problem sedlaste točke

U odnosu na točke lokalnog minimuma, sedlaste točke javljaju se puno rjeđe. U takvim točkama, gradijenti funkcije cijene također su nula. Međutim, s porastom dimenzionalnosti, broj sedlastih točaka raste eksponencijalno. Sedlaste točke u svojoj okolini imaju točke s većom i manjom vrijednosti funkcije cijene.

Za neku točku kažemo da je sedlasta ako **Hesseova** matrica sadrži svojstvene vrijednosti koje su i pozitivne i negativne. Podsjetimo se prikaza Hesseove matrice na primjeru realne funkcije dvije varijable $f(x, y)$:

$$H_f = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x^2} & \dfrac{\partial^2 f}{\partial x \partial y} \\ \dfrac{\partial^2 f}{\partial x \partial y} & \dfrac{\partial^2 f}{\partial y^2} \end{bmatrix}.$$

Gradijenti funkcije gubitka u okolini takvih točaka često poprimaju vrlo male vrijednosti, što uvelike otežava učenje mreže. Naime, učenje SGD metodom najčešće "zapne" u okolini sedlaste točke radeći vrlo male korake, čime drastično usporava proces učenja.

Jedno rješenje prethodnih problema je optimizacijom stohastičkog gradijentnog spusta momentom.

Problem litice i ogromnih gradijenata

Duboke neuronske mreže s mnogo slojeva često imaju vrlo strma područja, koja nalikuju liticama. Vrlo često se pojavljuju kod rekurzivnih neuronskih mreža, kod kojih često dolazi do množenja velikih vrijednosti težina. Ukoliko dođe do ažuriranja parametara gradijentnim spustom u točki na rubu litice, moguće je da će ažurirani parametri biti jako udaljeni, "skačući" s litice u podnožje manjih vrijednosti. Time se gube do sada naučene informacije.

Problem iščezavanja gradijenata

Duboke unaprijedne neuronske mreže imaju duboke grafove računanja, pri čemu dolazi do ponavljanja iste operacije, primjerice operacije množenja nad istim parametrima. Ako W predstavlja matricu težina, a graf računanja mreže ima put koji

se sastoji od ponavljanja množenja s W, tada nakon t koraka dobivamo matricu W^t. Pretpostavimo da matrica W ima svojstvenu dekompoziciju

$$W = V \operatorname{diag}(\lambda) V^{-1}.$$

Lako je vidjeti da vrijedi:

$$W^t = (V \operatorname{diag}(\lambda) V^{-1})^t = V \operatorname{diag}(\lambda)^t V^{-1}.$$

Vrijednosti λ_i na dijagonali matrice $\operatorname{diag}(\lambda)$ predstavljaju svojstvene vrijednosti matrice težina. Ako su svojstvene vrijednosti manje od 1, gradijenti će nakon nekog vremena iščeznuti. S druge strane, ako su svojstvene vrijednosti veće od 1, gradijenti će se stalno povećavati.

Metode optimizacije

Metoda momenta

Metoda momenta pokazala se vrlo djelotvornom kod ubrzavanja učenja neuronskih mreža, posebice na funkcijama koje su "više zakrivljene" i koje imaju male konzistentne gradijente. Njezino djelovanje možemo opisati kao akumuliranje eksponencijalno padajućih aritmetičkih sredina kretanja prethodnih gradijenata.

Naziv metode ima analogiju s kretanjem mase u fizici, gdje predstavlja umnožak mase i brzine tijela. U primjenama učenja neuronskih mreža, pretpostavljamo da je masa jedinična pa moment zapravo predstavlja **brzinu gibanja**.

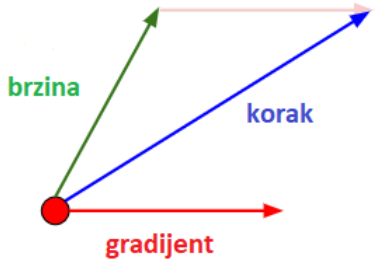
Neka je $\boldsymbol{v}$ varijabla koja predstavlja brzinu i smjer kojom se parametri kreću kroz prostor. Uvodimo hiperparametar $\rho \in [0, 1\rangle$, koji određuje koliko brzo prethodni gradijenti doprinose kretanju u smjeru minimuma.

Definiramo novo pravilo ažuriranja stohastičkog gradijentnog spusta pomoću momenta na način:

$$v_{ij}^{(t+1)} = \rho v_{ij}^{(t)} + \frac{\partial J}{\partial w_{ij}}(w_{ij}^{(t)}),$$
$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \alpha v_{ij}^{(t+1)},$$

gdje t predstavlja iteraciju.

Ažuriranjem parametara na prethodno definiran način, dobivamo bolju konvergenciju prema minimumu, ali i manje oscilacije oko točke minimuma. Pri korištenju klasičnog algoritma stohastičkog gradijentnog spusta bez momenta, ažuriranje parametara mreže stalo bi na pronađenoj točki lokalnog minimuma ili na sedlastoj točki te osciliralo u njihovoj okolini.



Slika 2.19: Smjer kretanja gradijentnog spusta primjenom momenta.

Upotrebom optimiziranog stohastičkog gradijentnog spusta s momentom, unaprijed je izračunata prosječna brzina i kretanje prethodnih gradijenata (vidi sliku 2.19). Na taj način, gradijenti se kreću u izračunatom smjeru i time izbjegavaju "loše" točke lokalnog minimuma i sedlaste točke, te se kreću prema globalnom minimumu ili prema zadovoljavajuće dobrom lokalnom minimumu.

Nesterovljev moment

Nesterovljev moment je varijanta metode momenta primijenjena na stohastičkom gradijentnom spustu. Inspirirana je metodom Nesterovljevog ubrzavajućeg gradijenta, detaljno opisanom u [12].

Metoda daje nova pravila ažuriranja težina definirana s:

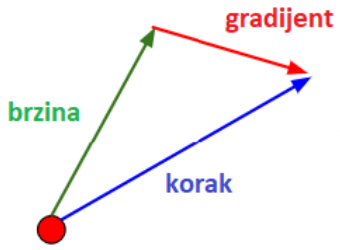
$$v_{ij}^{(t+1)} = \rho v_{ij}^{(t)} - \alpha \frac{\partial L^{(t)}}{\partial w_{ij}}(w_{ij}^{(t)} + \rho v_{ij}^{(t)}),$$
$$w_{ij}^{(t+1)} = w_{ij}^{(t)} + v_{ij}^{(t+1)},$$

gdje t predstavlja iteraciju, a parametri α i ρ imaju značenje kao u definiciji momenta.

Razlika u odnosu na klasičan moment je u točki računanja gradijenta. U Nesterovljevom algoritmu, gradijent se primjenjuje tek nakon što se izračunala trenutna brzina (vidi sliku 2.20). Tako Nesterovljev moment možemo shvatiti kao korekciju metode momenta, na način da gledamo jedan korak unaprijed, u kojem smjeru će se gradijentni spust kretati.

AdaGrad algoritam

AdaGrad pripada skupini algoritama s prilagodljivom stopom učenja i predstavlja modifikaciju tradicionalnog stohastičkog gradijentnog spusta. Umjesto globalne stope učenja α, stopa učenja se normalizira po svim dimenzijama o kojima funkcija



Slika 2.20: Smjer kretanja gradijentnog spusta primjenom Nesterovljevog momenta.

cijene ovisi. Nova stopa učenja sada je globalna stopa učenja α, podijeljena s l^2 normom prijašnjih gradijenata sve do trenutne iteracije.

Neka je J funkcija cijene. Tada je novo pravilo ažuriranja težina za w_{ij} definirano s:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \frac{\alpha}{\sqrt{\sum_{\tau=1}^{t}(w_{ij}^{\tau})^2 + \epsilon}} \frac{\partial J^{(t)}}{\partial w_{ij}},$$

gdje je α globalna stopa učenja, a w_{ij}^{t} i w_{ij}^{t+1} vrijednosti težina u iteracijama t i $t+1$, respektivno.

Za razliku od algoritma stohastičkog gradijentnog spusta, koji svakom parametru pridjeljuje jednaku važnost u svim iteracijama fiksnom stopom učenja, AdaGrad algoritam prilagođava stopu učenja iteraciji i parametru. Na taj način, svaki parametar dobiva na važnosti. Problem koji se događa sa SGD algoritmom jest nestajanje važnih informacija, nakon primjene na matricu težina koja je rijetka. AdaGrad algoritam pridaje više značajnosti rijetkim parametrima, jer će vrijednost $\sqrt{\sum_{\tau=1}^{t}(w_{ij}^{\tau})^2 + \epsilon}$ tada biti manja, stoga će ažuriranje biti veće, pa će informacije ostati očuvane.

U teoriji, AdaGrad algoritam ima vrlo poželjna svojstva, ali empirijski je pokazano da, pri učenju vrlo dubokih mreža, akumulirane kvadrirane vrijednosti s početka učenja mreže mogu dovesti do preranog usporenja učenja. Upravo zato, algoritam radi dobro za neke slučajeve, ali za druge ne.

RMSprop algoritam

RMSprop algoritam je modificirana verzija AdaGrad algoritma, koja optimizira učenje na nekonveksnim funkcijama.

Algoritam AdaGrad konvergira vrlo brzo na konveksnim funkcijama. Međutim, na nekonveksnim funkcijama, trajektorije učenja često mogu prijeći različite okolne strukture, prije nego stignu do ciljanog konveksnog udubljenja. Razlog tome leži u

činjenici da AdaGrad prilagođava stopu učenja u ovisnosti o cijeloj povijesti gradijenata i time drastično smanjuje stopu učenja prije dolaska do konveksne strukture.

RMSprop algoritam koristi eksponencijalno padajući niz prosječnih vrijednosti gradijenata, kako bi pridao više važnosti nedavnoj povijesti, a staru zanemario. Posljedica toga je da se gradijenti kreću vrlo brzo, čim algoritam pronađe ulaz u konveksno udubljenje.

Neka je $\boldsymbol{\beta}$ hiperparametar algoritma koji predstavlja stopu eksponencijalnog padanja niza sredina i neka je ϵ prevencija dijeljenja nulom. Ažuriranje parametara definiramo na sljedeći način:

$$v_{ij}^{(t)} = \beta v_{ij}^{(t-1)} + (1-\beta)\left(\frac{\partial J^{(t)}}{\partial w_{ij}}\right)^2,$$

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \frac{\alpha}{\sqrt{v_{ij}^{(t)} + \epsilon}} \frac{\partial J^{(t)}}{\partial w_{ij}}.$$

U praksi se ovaj algoritam često koristi i pokazuje dobre rezultate.

Adam algoritam

Adam algoritam još je jedan algoritam s prilagodljivom stopom učenja, za svaku težinu zasebno. Sam naziv nastao je od izraza "adaptirajući moment". Algoritam možemo shvatiti kao kombinaciju prethodno opisanog algoritma momenta i RMSprop algoritma, s dodatnom korekcijom.

Podaci koji se u svakoj iteraciji računaju, predstavljaju eksponencijalno padajući niz aritmetičkih sredina gradijenata, kao kod algoritma momenta i RSMprop algoritma. Zadaju se i hiperparametri $\boldsymbol{\beta_1}$ i $\boldsymbol{\beta_2}$, koji imaju ulogu **stope padanja**, te ϵ, kao prevencija dijeljenja s nulom.

Računamo padajući niz prosječnih vrijednosti prošlih gradijenata kao i njihovih kvadrata na način:

$$m_{ij}^{(t)} = \beta_1 m_{ij}^{(t-1)} + (1-\beta_1)\frac{\partial L^{(t)}}{\partial w_{ij}}, \tag{2.13}$$

$$v_{ij}^{(t)} = \beta_2 v_{ij}^{(t-1)} + (1-\beta_2)\left(\frac{\partial L^{(t)}}{\partial w_{ij}}\right)^2, \tag{2.14}$$

gdje (2.13) nazivamo **prvi moment**, a (2.14) **drugi moment**.

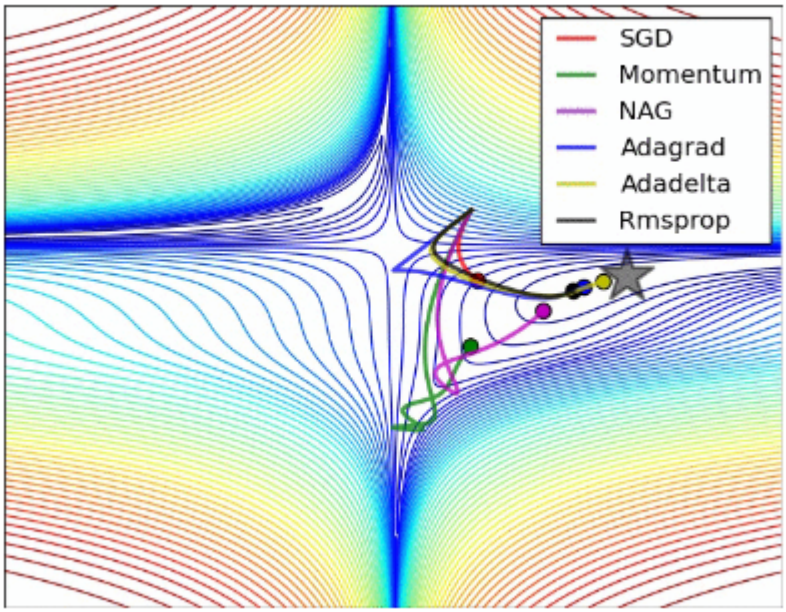
Sada računamo normalizirani prvi i drugi moment s:

$$\widehat{m}_{ij}^{(t)} = \frac{m_{ij}^{(t)}}{1-\beta_1^{(t)}}, \qquad \widehat{v}_{ij}^{(t)} = \frac{v_{ij}^{(t)}}{1-\beta_2^{(t)}}. \tag{2.15}$$

Pomoću (2.15) dobivamo novo pravilo ažuriranja:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \frac{\alpha}{\sqrt{\widehat{v}_{ij}^{(t)} + \epsilon}} \widehat{m}_{ij}^{(t)}. \tag{2.16}$$

Na slici 2.21 vidimo razliku u napretku učenja različitih algoritama u danom trenutku. Primjećujemo da stohastički gradijentni spust vrlo sporo u konvergira, unatoč tome što ima najkraću putanju. Ostale prikazane metode pokazale su se bržima u praksi, unatoč tome što imaju "skretanja" s puta i duže trajektorije.



Slika 2.21: Trajektorije različitih optimizacijskih algoritama koje konvergiraju prema lokalnom minimumu, označenom sa simbolom zvijezde (izvor [10]).

2.6 Regularizacija

Osnovna zadaća prilikom modeliranja i učenja neuronskih mreža jest da algoritam ima dobre performanse na skupu za učenje, ali i na potpuno novim, još neviđenim podacima. Ovaj problem rješava se raznim metodama regularizacije u strojnom učenju, modificiranjem postojećeg algoritma kako bi se umanjila opća pogreška.

U nastavku opisujemo nekoliko poznatih metoda regularizacije.

Kažnjavanje norme težina

Kod vrlo dubokih i kompleksnih neuronskih mreža moguće je pojavljivanje **prenaučenosti** (engl. *overfitting*). Model se može ponašati vrlo dobro na podacima za učenje, dok njegove performanse na skupu podataka za testiranje mogu biti loše. Zbog visoke varijance, model postaje vrlo osjetljiv na male promjene ulaznih podataka, jer su parametri mreže velikih magnituda.

Kao prevencija opisanog problema, često se funkcija cijene J modificira na način da joj se dodaje vrijednost $\boldsymbol{\lambda R(W)}$, koju nazivamo **penal**:

$$J = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i; W)) + \boldsymbol{\lambda R(W)},$$

gdje je λ hiperparametar, a R zadana norma težina. Dodana vrijednost služi za kažnjavanje funkcije gubitka u slučaju pojave parametara velikih magnituda. Najčešće R predstavlja l^1 ili l^2 normu. Tada se radi o $\boldsymbol{L^1}$ **regularizaciji**, odnosno, o $\boldsymbol{L^2}$ **regularizaciji**.

Hiperparametar $\lambda \in [0, +\infty\rangle$ predstavlja doprinos koji norma R ima na funkciju cijene. Veće vrijednosti λ označavaju veću regularizaciju.

U praksi, obično svaki sloj neuronske mreže koristi penal s drugačijim izborom hiperparametra λ.

L^2 regularizacija

L^2 regularizacija najjednostavniji je i najčešće korišten oblik kažnjavanja parametara normom. U praksi se može čuti i naziv "pad težina", što će biti jasno kada promotrimo na koji način se ažuriraju vrijednosti težina gradijentom.

Definicija 2.6.1. ***L^2 regularizacija*** *parametara modela W definirana je s:*

$$R(W) = \|W\|_2 = \sum_i \sum_j w_{ij}^2. \tag{2.17}$$

Neka je zadana funkcija gubitka $L(W; x, y)$. Ako u $L(W; x, y)$ uključimo penal zadan l^2 normom iz (2.17) i hiperparametrom λ, tada regularizirani oblik funkcije gubitka ima oblik:

$$\widehat{L}(W; x, y) = L(W; x, y) + \frac{\lambda}{2}\|W\|_2. \tag{2.18}$$

Gradijent modificirane funkcije gubitka iz (2.18) je tada:

$$\nabla_W \widehat{L}(W; x, y) = \nabla_W L(W; x, y) + \lambda W. \tag{2.19}$$

Kako bismo ažurirali težine pri propagaciji unatrag, uzimajući u obzir dobiveni gradijent (2.19), računamo:

$$W \leftarrow W - \alpha(\lambda W + \nabla_W L(W; x, y)), \tag{2.20}$$

odnosno, raspisujući (2.20), dobivamo:

$$W \leftarrow (1 - \alpha\lambda)W - \alpha\nabla_W L(W; x, y).$$

Odmah je jasno da pri ažuriranju težina dolazi do "pada" trenutne vrijednosti težina za konstantni multiplikativni faktor.

Isključivanje

Regularizacija težina u potpuno povezanom sloju konvolucijske neuronske mreže može se postići i **isključivanjem** nekih neurona. Prilikom učenja unaprijedne neuronske mreže na skupu podataka za učenje, moguće je s određenom vjerojatnošću u potpunosti isključiti određene neurone, zajedno s njihovim težinama. Time omogućujemo preostalim neuronima učenje bitnih značajki, bez ovisnosti o isključenim neuronima. Razlog upotrebljavanja ove metode leži u činjenici da velika ovisnost neurona jedan o drugome može dovesti do **prenaučenosti** modela, koji zatim ne pokazuje dobre rezultate na testnom skupu podataka.

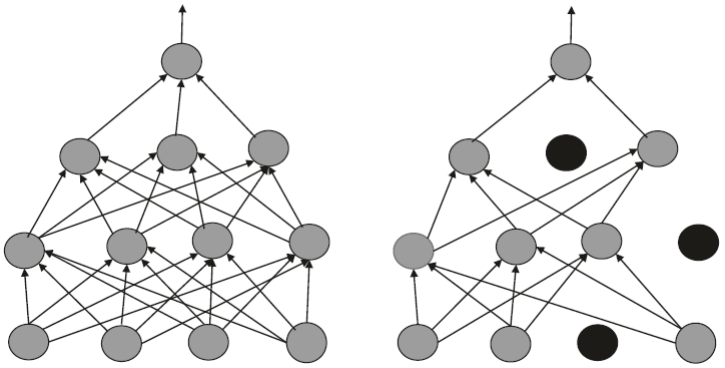
Ako se mreža sastoji od N neurona, tada postoji 2^N mogućih konfiguracija mreže. Za svaki primjer iz malog skupa za treniranje, generira se jedna konfiguracija s neuronima nasumično isključenim određenom vjerojatnošću. Dakle, učenje neuronske mreže isključivanjem jedinica jednako je učenju skupa različitih neuronskih mreža.

Nakon generiranja skupa neuronskih mreža, predviđanje se temelji na prosječnoj vrijednosti svih neuronskih mreža, čime se umanjuje varijanca i izbjegava prenaučenost. Na slici 2.22 prikazana je jedna konfiguracija mreže.

Nakon svakog učitavanja primjera iz skupa podataka za učenje, nasumično biramo novu **binarnu masku $\boldsymbol{\mu}$**, koja označava koje neurone iz ulaznog sloja i skrivenih slojeva isključujemo. Vjerojatnost pojavljivanja jedinice u maski je hiperparametar koji se zadaje na početku učenja. Obično se ulazni neuroni uključuju s vjerojatnošću 0.8, a neuroni skrivenih slojeva s 0.5.

Nakon koraka isključivanja, slijedi propagacija unaprijed, propagacija unatrag te ažuriranje težina.

Unatoč tome što različitih konfiguracija originalne mreže ima eksponencijalno mnogo, samo manji dio skupa konfiguracija ulazi u proces učenja. Svaka konfiguracija mreže inicijalno ima jednake sve parametre kao i ostale konfiguracije. Dijeljenje



Slika 2.22: *Lijevo*: Primjer potpuno povezane neuronske mreže. *Desno*: Primjer jedne konfiguracije originalne neuronske mreže s isključena tri neurona (izvor [13], str. 191).

parametara mreže na taj način, čini da svaka konfiguracija mreže naposljetku rezultira zadovoljavajućim parametrima.

Kako bi se postigli dobri rezultati, dovoljno je izračunati **geometrijsku sredinu** distribucije predviđanja skupa svih konfiguracija. Definiramo novu vjerojatnost, temeljenu na skupu konfiguracija s:

$$\widetilde{p}_{konf}(y|x) = \sqrt[2^d]{\prod_{\mu} p(y|x,\mu)}, \tag{2.21}$$

gdje je d broj jedinica koje mogu biti isključene.

Međutim, dobivena distribucija iz (2.21) nije vjerojatnosna pa ju je potrebno normalizirati. Normaliziramo ju na sljedeći način:

$$p_{konf}(y|x) = \frac{\widetilde{p}_{konf}(y|x)}{\sum_{y'} \widetilde{p}_{konf}(y'|x)}.$$

U praksi se zadovoljavajući rezultati postižu već za 10 do 20 konfiguracija mreže.

Testiranje podataka provodi se na mreži bez isključivanja, koju modificiramo na način da svakom neuronu prilagodimo težine. Ako je vjerojatnost pojavljivanja neurona u mreži tijekom učenja mreže bila p, tada će težine koje izlaze iz neurona biti skalirane množenjem s vjerojatnosti p.

Povećanje skupa podataka

Poznato je da duboke konvolucijske neuronske mreže daju bolje rezultate ako su učene na većem skupu podataka. Međutim, ponekad je skup podataka s kojim radimo

ograničen, stoga moramo pribjeći novim metodama kojima bismo ga povećali.

Primjerice, moguće je kreirati potpuno nove podatke, transformirajući podatke koje već imamo na određeni način. Kod zadatka klasifikacije, uspješnim su se pokazale translacija piksela u određenom smjeru te rotacija. Takve modifikacije možda neće biti dobre za neke druge zadaće, stoga je izbor transformacije vrlo bitan i ovisi o zadanom problemu kojeg rješavamo.

Tradicionalne transformacije su afine te za svaku ulaznu sliku proizvode njezin modificirani duplikat. Time smo povećali skup podataka za učenje.

S druge strane, kako bi se povećala robusnost neuronskih mreža, često se ulaznim slikama dodaju nasumični šumovi, čime se uvelike smanjuje generalna greška.

Ostale metode transformacije, koje su se u praksi pokazale djelotvornima, su:

- rotacija,
- rastezanje,
- horizontalni okret,
- nasumična rezanja slike i skaliranja,
- mijenjanje kontrasta i svjetline.

2.7 Primjeri klasičnih konvolucijskih mreža

U ovom odjeljku dajemo prikaz tri najpoznatije arhitekture konvolucijskih neuronskih mreža kronološkim redom. Ove arhitekture se i danas često koriste u nekom optimiziranom obliku. Problemi koji se njima rješavaju nisu ograničeni samo na klasifikaciju slika. Primjene idu vrlo daleko, od segmentacije, lokalizacije, detekcije više objekata, prepoznavanja lica, do prijenosa stila jedne slike na drugu sliku.

LeNet-5

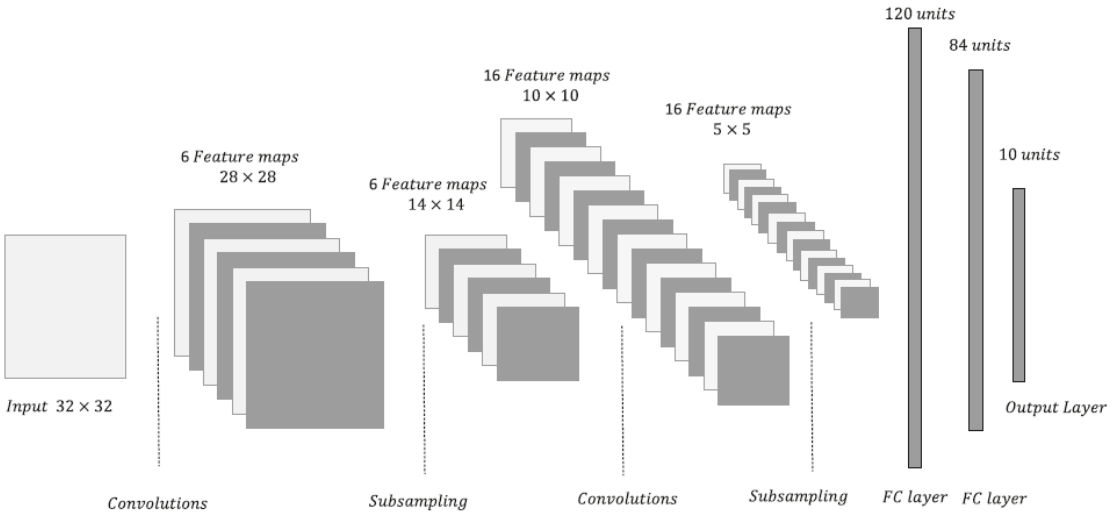
LeNet-5 arhitektura prvi puta se javlja 1998. godine, kada ju je modelirao Yann LeCun za prepoznavanje rukom pisanih znamenki. Ovo je ujedno i prva konvolucijska neuronska mreža.

Mreža je na ulazu primala skup podataka, koji se sastojao od monokromatskih slika dimenzija 32×32, koje su predstavljale rukom pisane znamenke. Nakon prvog konvolucijskog sloja dobiveno je 6 izlaznih aktivacijskih mapa dimenzija 28×28. Dobivene aktivacijske mape tada su prošle sloj sažimanja te su se dimenzije smanjile na 14×14. Sljedeći konvolucijski sloj generirao je 16 aktivacijskih mapa dimenzija 10×10, nakon čega je ponovno uslijedilo sažimanje na dimenzije 5×5.

Nakon toga, slijede dva potpuno povezana sloja, dimenzija 120 i 84, respektivno. Posljednji sloj mreže predstavljen je s 10 neurona koji predstavljaju klase znamenaka od 0 do 9.

Mreža je imala 60 000 parametara za učenje.

Na slici 2.23 prikazana je arhitektura slojeva mreže LeNet-5. Detaljna arhitektura mreže izložena je u članku [9].



Slika 2.23: Arhitektura LeNet-5 konvolucijske neuronske mreže, 1998. (izvor [13], str. 207)

AlexNet

Model AlexNet mreže razvili su Alex Krizhevsky, Ilya Sutskever i Geoffrey Hinton 2012. godine, kako bi se natjecali na ImageNet ILSVRC natjecanju, na kojemu su i osvojili prvo mjesto.

Mreža je postigla izvrsne rezultate sa stopom greške od samo 15.4% na prvih 5 predviđanja, dok je drugoplasirana mreža imala grešku od 26.2%.

Sastoji se od 5 CONV slojeva, MAX POOL slojeva i slojeva isključivanja, te 3 potpuno povezana sloja.

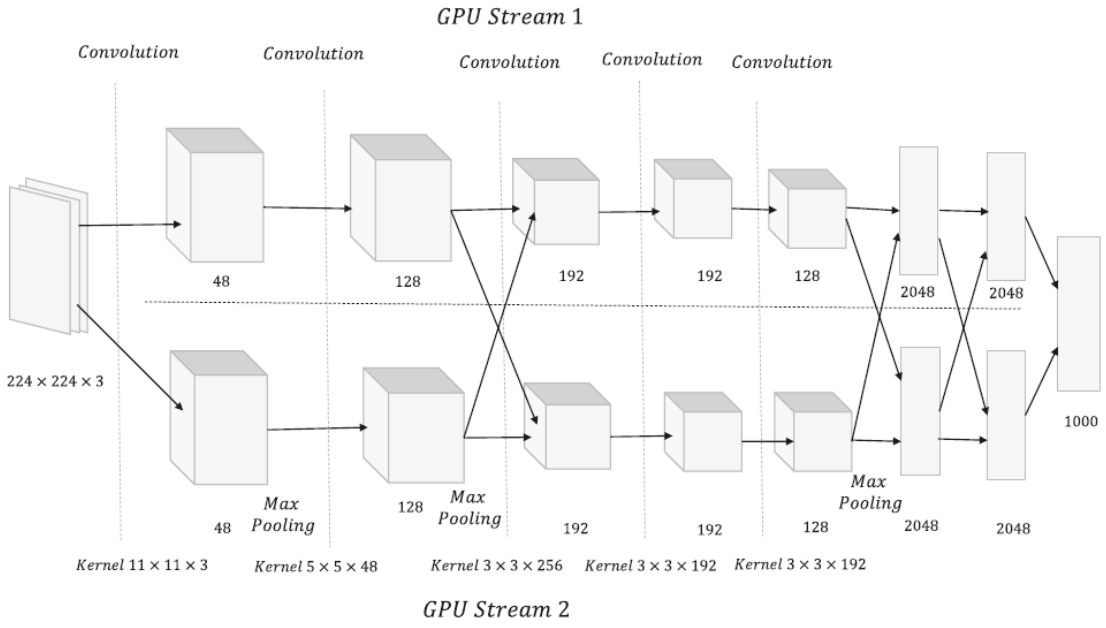
Izlazni sloj prepoznavao je 1000 klasa te je ovo prvi veliki skok u razvoju konvolucijskih neuronskih mreža.

Broj parametara drastično je narastao na čak 160 milijuna, što je bilo vrlo zahtjevno za računanje. Iz tog razloga mreža se učila na GPU jedinicama.

Bitne značajke koje su unaprijedile AlexNet mrežu su:

- Korištenje ReLU aktivacijske funkcije, koja je mnogo lakša za računanje u odnosu na sigmoidnu funkciju ili tanh.
- Korištenje isključivanja neurona s vjerojatnošću 0.5, kako bi se izbjegla prenaučenost mreže.
- Metoda SGD Momentuma 0.9.
- Korištena L2 regularizacija s padom težina za $5e^{-4}$.
- Korištenje preklapajućeg sažimanja.
- Učenje modela na GPU GTX 580 oko 5 dana.
- Povećani skup podataka metodama translacije, horizontalne refleksije, itd.

Više o rezultatima dobivenim učenjem mreže vidjeti u [1].



Slika 2.24: Arhitektura AlexNet konvolucisjke neuronske mreže, 2012. (izvor [13], str. 209)

VGG16

VGG je naziv tima koji je 2014. godine sudjelovao na ImageNet ILSVRC natjecanju te osvojio nagradu sa svojom arhitekturom mreže u 16 slojeva.

Novost u odnosu na prijašnje arhitekture je korištenje više manjih 3×3 filtera, čime se smanjuje broj parametara mreže.

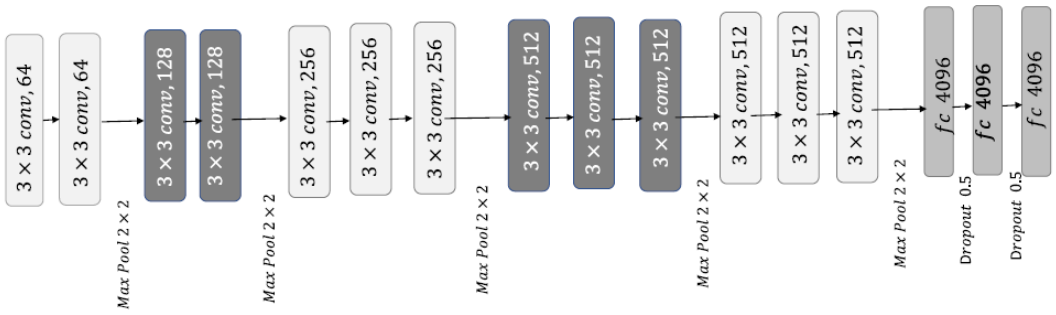
Prostorne dimenzije volumena smanjuju se prolaskom kroz dublje slojeve mreže, ali se zato dubina volumena povećava, zbog većeg broja filtera.

Korištenje tri CONV sloja s filterima dimenzija 3×3 ima jednako receptivno polje kao i jedan CONV sloj s filterima 7×7. Međutim, broj parametara se smanjio sa 7^2C^2 na $3 \cdot (3^2C^2)$, gdje je C broj kanala sloja.

Detalji:

- Arhitektura je osvojila 2. mjesto u klasifikaciji i 1. mjesto u lokalizaciji objekata na natjecanju ILSVRC 2014.
- Duboka mreža sa 138 milijuna parametara.
- Svi CONV slojevi koriste konvolucije s pomakom 1 i nadopunjavanjem 1.
- Danas postoji "bolja" verzija mreže, VGG19, koja koristi više memorije.
- Greška samo 7.3%.

Na slici 2.25 vidimo pojednostavljeni prikaz modela VGG16 konvolucijske neuronske mreže s ukupno 16 slojeva. Za više informacija o arhitekturi i rezultatima vidjeti članak [14].



Slika 2.25: Arhitektura VGG16 konvolucijske neuronske mreže, 2014. (izvor [13], str. 210).

U posljednjih nekoliko godina pojavila se prava eksplozija u razvijanju i modeliranju konvolucijskih neuronskih mreža. Razlog tome je i sve veća računalna snaga i

upotreba vrlo moćnih GPU kartica na kojima se proces učenja mreže znatno ubrzava, stoga mreže postaju sve dublje i kompleksnije.

Spomenimo još neke značajnije arhitekture razvijene u posljednjih nekoliko godina.

GoogLeNet je arhitektura mreže razvijena 2014. godine, s 22 sloja i samo 5 milijuna parametara za učenje, što je 12 puta manje od AlexNet arhitekture. Arhitektura je osvojila prvo mjesto na natjecanju ILSVRC 2014. godine za klasifikaciju objekata, s greškom od 6.7% za prvih 5 predviđanja.

ResNet je arhitektura razvijena 2015. godine, sa 152 sloja mreže. Ujedno je i pobjednica natjecanja ILSVRC 2015 za klasifikaciju objekata, s greškom od samo 3.57% za prvih 5 najboljih predviđanja.

Mreža se sastoji od tzv. rezidualnih blokova. Ideja modela je kopiranje već naučenih težina iz "plićih" slojeva mreže te dodavanje slojeva za mapiranje.

SENet arhitektura odnijela je pobjedu na ILSVRC 2017 natjecanju, s greškom od 2.251% na prvih 5 predviđanja i time postala za 25% bolja od pobjednika prethodne godine.

Poglavlje 3

Razvoj modela prijenosa stila pomoću konvolucijske neuronske mreže

3.1 Neuronski prijenos stila

Neuronski prijenos stila (engl. *Neural style transfer*) je metoda izgrađena na modelu konvolucijske neuronske mreže, koja koristi neuronsku reprezentaciju kako bi razdijelila sadržaj i stil proizvoljne slike. U članku [4] pokazano je kako se reprezentacija sadržaja i stila slike može razdvojiti, te se s njima može neovisno manipulirati. Slika sadržaja može biti neka fotografija, dok slika stila može biti neko umjetničko djelo. Rezultat algoritma je nova slika, koja je zadržala sadržaj jedne slike, a stil druge slike.

Za reprezentaciju **stila** ulazne slike koristi se prostor značajki koji promatra teksturu ulazne slike. Prostor značajki izgrađuje se nad rezultatima svakog sloja mreže. Teksturu slike možemo izračunati promatrajući korelaciju između različitih rezultata filtera u mapama značajki, i intuitivno ju možemo shvatiti kao očuvanje boje i lokalnih struktura slike. Kako bi se dobila stacionarna, više-skalirana reprezentacija ulazne slike, koja sadržava bitne informacije o teksturi, najčešće se promatra nekoliko slojeva mreže.

Pri generiranju nove slike, najčešće to nije slika koja savršeno primjenjuje i sadržaj jedne slike i stil druge. Međutim, koristeći funkciju gubitka, možemo istovremeno minimizirati udaljenost od oba svojstva. Također, različitim regularizacijskim parametrima možemo upravljati utjecajem stila, odnosno, sadržaja.

Kako bi rezultati bili što bolji, kao podlogu za računanje gubitaka koristimo već naučenu konvolucijsku neuronsku mrežu VGG19, koju smo detaljnije opisali u pret-

hodnom poglavlju. Model VGG19 prvenstveno služi za klasifikaciju objekata, no mi ćemo za prijenos stila upotrijebiti samo neke slojeve koji su nam od značaja.

Neka su p i x originalna slika i slika koja je generirana, respektivno. Neka su $P^{[l]}$ i $F^{[l]}$ njihove reprezentacije značajki u sloju l. Ako je $F^l \in \mathbb{R}^{N_l \times M_l}$, gdje je N^l broj aktivacijskih mapa, a M^l dimenzija aktivacijske mape, tada s F^l_{ij} označavamo vrijednost aktivacije i-tog filtera na poziciji j u sloju l.

Definiramo funkciju gubitka između reprezentacija $P^{[l]}$ i $F^{[l]}$ na način:

$$L_{content}(p, x, l) = \frac{1}{2} \sum_{i,j} (P^l_{ij} - F^l_{ij})^2. \tag{3.1}$$

Parcijalne derivacije funkcije gubitka iz (3.1) po aktivacijama iz sloja l tada su:

$$\frac{\partial L_{content}}{\partial F^l_{ij}} = \begin{cases} (F^l - P^l)_{ij}, & \text{za } F^l_{ij} > 0, \\ 0, & \text{za } F^l_{ij} < 0. \end{cases}$$

Algoritmom propagacije pogreške unatrag, mijenjamo vrijednosti piksela generirane slike x, koja je na početku inicijalizirana nasumičnim vrijednostima. Aktivacije, svakom iteracijom, postaju sve sličnije aktivacijama originalne slike sadržaja u odabranim slojevima. Time se generirana slika x sve više sadržajem približava slici sadržaja p.

Kako bismo prikazali stil ulazne slike stila, na svakom odabranom sloju konvolucijske mreže gradimo reprezentaciju koja računa korelaciju između rezultata različitih filtera u aktivacijskoj mapi. Pokazalo se kako je za reprezentaciju stila najbolje upotrijebiti Gramovu matricu $G^l \in \mathbb{R}^{N_l \times N_l}$, definiranu s:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}.$$

Za generiranje slike koja preuzima stil ulazne slike stila, minimiziramo funkciju gubitka stila. Funkcija gubitka stila mjeri udaljenost Gramove matrice originalne slike stila i Gramove matrice generirane slike.

Neka su a i x slika stila i generirana slika, i neka su A^l i G^l njihove reprezentacije stila Gramovim matricama u sloju l. Doprinos koji sloj l ima na ukupni gubitak stila definiran je na način:

$$E^l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G^l_{ij} - A^l_{ij})^2.$$

Tada ukupni gubitak definiramo zbrajanjem gubitaka po svim slojevima:

$$L_{style}(a, x) = \sum_{l=0}^{L} w_l E_l,$$

gdje su w_i težine koje određuju doprinos svakog sloja ukupnom gubitku.

Parcijalne derivacije funkcije E_l po aktivacijama iz sloja l mogu se izračunati na način:

$$\frac{\partial E_l}{\partial F_{ij}^l} = \begin{cases} \frac{1}{N_L^2 M_l^2} \left((F^l)^T (G^l - A^l)\right)_{ji}, & \text{za } F_{ij}^l > 0, \\ 0, & \text{za } F_{ij}^l < 0. \end{cases}$$

Kako bismo generirali sliku koja je mješavina sadržaja fotografije i stila neke umjetničke slike, minimiziramo udaljenost generirane slike od reprezentacije sadržaja fotografije u jednom sloju i reprezentacije stila umjetničke slike u nekoliko slojeva.

Ako s p označimo fotografiju, a s a umjetničku sliku, funkciju gubitka računamo na sljedeći način:

$$L_{total}(p, a, x) = \alpha L_{content}(p, x) + \beta L_{style}(a, x), \tag{3.2}$$

gdje su α i β težinski faktori za rekonstrukciju sadržaja i stila.

3.2 Opis zadatka

Cilj praktičnog dijela ovog rada bio je razviti model neuronskog prijenosa stila koji daje zadovoljavajuće rezultate na izlaznim generiranim slikama. Za ulazne slike stila korišteno je nekoliko umjetničkih slika slavnih slikara, a za ulazne slike sadržaja korištene su fotografije. Slike su proizvoljnih dimenzija te ih algoritam sam prilagođava traženim dimenzijama modela. Također, razvijeni model ima sposobnost primjene stila više umjetničkih slika na ulaznu fotografiju.

Model je razvijen u programskom jeziku Python 3.0, uz pomoć biblioteke otvorenog koda *TensorFlow* [3]. U nastavku opisujemo implementacijske detalje.

3.3 Korišteni alati i detalji modela

Model neuronskog prijenosa stila učen je na razvojnom oblaku Google Colaboratory, koji koristi Jupyter bilježnicu za rad. Google Colaboratory je okolina integrirana

s Google Drive oblakom, na kojemu su pohranjene sve bilježnice koje kreiramo. Prilikom rada, sve izmjene se istovremeno spremaju na naš Google Drive račun.

Razlog korištenja ove okoline leži u vrlo brzim grafičkim karticama Tesla K80 GPU na kojima se vršilo računanje. Time se znatno skratilo trajanje učenja modela.

Kako bi se implementirani model mogao koristiti, najprije je potrebno urediti potrebnu okolinu. Zbog integracije s Google Drive-om, najprije je potrebna autentifikacija korisnika pomoću njegovog Google računa. U radnom prostoru kreirani su direktoriji `pretrained-model`, `content_images`, `style_images` i `output_images`, u koje su zatim učitane potrebne datoteke. Datoteke uključuju slike stila, slike sadržaja, te naučeni VGG19 model. Za prijenos datoteka s Google Drive oblaka u radni prostor korištena je biblioteka `pydrive`.

Model konvolucijske neuronske mreže

Prijenos stila vrlo je zahtjevna zadaća, stoga je korištena već naučena mreža VGG19. Model je službeno javno dostupan za korištenje i preuzimanje na [2].

Korištena verzija modela je `"imagenet-vgg-verydeep-19.mat"`, sa 16 konvolucijskih slojeva i 5 slojeva sažimanja. Učenje se odvijalo na nekoliko središnjih slojeva mreže, jer se ispostavilo da oni daju najbolje rezultate.

Primjerice, za učenje sadržaja s fotografije korištena su dva konvolucijska sloja:

```
CONTENT_LAYERS=('conv4_1','conv5_1')
```

S druge strane, za učenje stila umjetničke fotografije korišteno je nekoliko konvolucijskih slojeva:

```
STYLE_LAYERS=('conv1_1','conv2_1','conv3_1','conv4_1','conv5_1')
```

Slojevi su zadani u konfiguracijskoj klasi `CONFIG` i mogu se vrlo lako izmijeniti po potrebi. Učitavanje modela VGG19 napravljeno je uz pomoć metode `load_vgg_model(path)`, koja kao argument prima putanju do modela. Metoda vraća graf koji predstavlja strukturu mreže. Struktura grafa čuva osnovne informacije računanja koje se evaluiraju tek kasnije, u kreiranoj sesiji sa stvarnim podacima. Pomoću imena sloja, sada lako možemo pristupiti bilo kojem sloju modela i dobiti njegove aktivacijske vrijednosti.

```
def load_vgg_model(path):
  vgg_model = scipy.io.loadmat(path)
  vgg_layers = vgg_model['layers']
```

```
def weights(layer, expected_layer_name):
    wb = vgg_layers[0][layer][0][0][2]
    W = wb[0][0]
    b = wb[0][1]
    return W, b

def relu_layer(conv2d_layer):
    return tf.nn.relu(conv2d_layer)

def conv2d_layer(prev_layer, layer, layer_name):
    W, b = weights(layer, layer_name)
    W = tf.constant(W)
    b = tf.constant(np.reshape(b, (b.size)))
    return tf.nn.conv2d(prev_layer,
      filter=W, strides=[1, 1, 1, 1],
      padding= CONFIG.PADDING) + b

def conv2d_relu_layer(prev_layer, layer, layer_name):
    return relu_layer(conv2d_layer(prev_layer, layer, layer_name))

def avgpool(prev_layer):
    return tf.nn.avg_pool(prev_layer,
      ksize=[1, 2, 2, 1],
      strides=[1, 2, 2, 1],
      padding= CONFIG.PADDING)

graph = {}
graph['input']  = tf.Variable(np.zeros((1, CONFIG.IMAGE_HEIGHT,
  CONFIG.IMAGE_WIDTH,
  CONFIG.COLOR_CHANNELS)),
  dtype = 'float32')

graph['conv1_1']  = conv2d_relu_layer(graph['input'], 0, 'conv1_1')
graph['conv1_2']  = conv2d_relu_layer(graph['conv1_1'], 2, 'conv1_2')
graph['avgpool1'] = avgpool(graph['conv1_2'])
graph['conv2_1']  = conv2d_relu_layer(graph['avgpool1'], 5, 'conv2_1')
graph['conv2_2']  = conv2d_relu_layer(graph['conv2_1'], 7, 'conv2_2')
graph['avgpool2'] = avgpool(graph['conv2_2'])
graph['conv3_1']  = conv2d_relu_layer(graph['avgpool2'], 10, 'conv3_1')
```

```
  graph['conv3_2']  = conv2d_relu_layer(graph['conv3_1'], 12, 'conv3_2')
  graph['conv3_3']  = conv2d_relu_layer(graph['conv3_2'], 14, 'conv3_3')
  graph['conv3_4']  = conv2d_relu_layer(graph['conv3_3'], 16, 'conv3_4')
  graph['avgpool3'] = avgpool(graph['conv3_4'])
  graph['conv4_1']  = conv2d_relu_layer(graph['avgpool3'], 19, 'conv4_1')
  graph['conv4_2']  = conv2d_relu_layer(graph['conv4_1'], 21, 'conv4_2')
  graph['conv4_3']  = conv2d_relu_layer(graph['conv4_2'], 23, 'conv4_3')
  graph['conv4_4']  = conv2d_relu_layer(graph['conv4_3'], 25, 'conv4_4')
  graph['avgpool4'] = avgpool(graph['conv4_4'])
  graph['conv5_1']  = conv2d_relu_layer(graph['avgpool4'], 28, 'conv5_1')
  graph['conv5_2']  = conv2d_relu_layer(graph['conv5_1'], 30, 'conv5_2')
  graph['conv5_3']  = conv2d_relu_layer(graph['conv5_2'], 32, 'conv5_3')
  graph['conv5_4']  = conv2d_relu_layer(graph['conv5_3'], 34, 'conv5_4')
  graph['avgpool5'] = avgpool(graph['conv5_4'])

  return graph
```

Funkcije gubitka

Prateći definiciju funkcija gubitka sadržaja i stila iz članka [4], definirane su metode `total_content_loss` i `total_style_loss`.

Metoda `total_content_loss` kao ulazne parametre prima trenutnu sesiju, model VGG19 mreže i sliku sadržaja, te računa ukupni gubitak na svim slojevima koji su definirani u konfiguraciji. Najčešće se radi o jednom ili dva središnja sloja.

Metoda `total_style_loss` kao ulaz prima sesiju, model mreže i ulaznu sliku stila. Ukupni gubitak stila računa se za svaku definiranu sliku stila (ukoliko želimo primijeniti više stilova) na svim definiranim slojevima stila.

Implementacijske detalje ostalih korištenih metoda ne prikazujemo, jer se iz njihova imena lako može iščitati značenje.

```
def total_content_loss(session, model, content_image):

  session.run(model['input'].assign(content_image))
  content_layers_weights = init_content_layers_weights()

  total_content_loss = 0

  for layer in CONTENT_LAYERS:
    out = model[layer]
    A_p = session.run(out)
```

```
    A_x = out
    total_content_loss += content_layer_loss(A_p, A_x)
                            * content_layers_weights[layer]

  return total_content_loss

def total_style_loss(session, model, style_images):

  style_layers_weights = init_style_layers_weights()
  style_blend_weights = init_style_blend_weights()

  if len(style_images) != len(style_blend_weights):
    raise ValueError('Style images count different from '
                     'style blend weights count!')
    return

  total_style_loss = 0.

  for image, key in zip(style_images, style_blend_weights):
    session.run(model['input'].assign(style_images[image]))
    style_loss = 0.
    for layer in STYLE_LAYERS:
      out = model[layer]
      A_a = session.run(out)
      A_x = out
      style_loss += style_layer_loss(A_a, A_x)
                     * style_layers_weights[layer]

    total_style_loss += style_loss*style_blend_weights[key]

  return total_style_loss
```

Učenje prijenosa stila

Učenje prijenosa stila odvija se u velikom broju iteracija. Za razliku od učenja modela, gdje je zadaća klasifikacija objekta i gdje se ažuriraju vrijednosti težina svakog sloja, ovdje se ažuriraju upravo vrijednosti piksela generirane slike.

Učitane slike stila i sadržaja mogu biti različitih dimenzija. Parametrima `IMAGE_WIDTH` i `IMAGE_HEIGHT` klase `CONFIG`, određene su krajnje dimenzije, kojima će model

prilagoditi sve ulazne slike. Prilikom učitavanja slika, one se pretprocesiraju kako bi odgovarale potrebama i dimenzijama modela VGG19. Njihove vrijednosti normaliziraju se oduzimanjem aritmetičke sredine po kanalima slike, zadane parametrom `MEANS`.

Prije početka učenja modela, generirana je slika s nasumičnim vrijednostima piksela iz uniformne razdiobe. U svakom koraku, ažurirane su vrijednosti piksela slike, kako bi se aktivacijske vrijednosti iz zadanih slojeva približile aktivacijskim vrijednostima slike sadržaja i slike stila. Za računanje gradijenata funkcije gubitka i samu minimizaciju gubitka, u implementacijskom rješenju moguće je koristiti jedan od dva zadana optimizatora koje pruža *TensorFlow* — Adam ili AdaGrad.

Model prijenosa stila definiran je u funkciji `stylize(iterations)`. U nastavku prikazujemo implementaciju metode. Metoda započinje kreiranjem nove sesije, koja predstavlja kontekst izvršavanja i sadrži sve potrebne fizičke resurse. Nakon naučenih vrijednosti piksela, metoda vraća generiranu sliku na kojoj je primijenjen stil jedne ili više slika stila.

Možemo primijetiti kako se parametri α i β, koji određuju utjecaj sadržaja i stila, čitaju iz konfiguracijske klase. Također, u funkciju gubitka dodana je regularizacija ulaza modela totalnom varijacijom `J_total_variation`, koja doprinosi glatkoći slike.

```
def stylize(iterations=1000):

  with tf.Session() as sess:

    model = load_vgg_model(CONFIG.VGG_MODEL_PATH)

    content_image = load_content_image(CONFIG.CONTENT_IMAGE_PATH
      + CONFIG.CONTENT_IMAGE)
    style_images = load_style_images(CONFIG.STYLE_IMAGES_PATH)
    generated_image = generate_noise_image(content_image)

    J_content = total_content_loss(sess, model, content_image)
    J_style = total_style_loss(sess, model, style_images)
    J_total_variation = tf.image.total_variation(model['input'])

    tf.summary.scalar('content_loss', J_content)
    tf.summary.scalar('style_loss', J_style)

    alpha = CONFIG.ALPHA
    beta = CONFIG.BETA
    regularization_weight = CONFIG.LAMBDA
```

```
J = alpha*J_content + beta*J_style
      + regularization_weight*J_total_variation
tf.summary.scalar('total_loss', J)


optimizer= get_optimizer(CONFIG.OPTIMIZER)
train_step = optimizer.minimize(J)

sess.run(tf.global_variables_initializer())
train_writer = tf.summary.FileWriter( './log', sess.graph)
sess.run(model['input'].assign(generated_image))

for i in range(iterations):

  sess.run(train_step)
  generated_image = sess.run(model['input'])
  merge = tf.summary.merge_all()

  if i % 50 == 0:
    summary, J_total, J_c, J_s = sess.run([merge, J, J_content,
                                           J_style])
    print("Iteration " + str(i) + " :")
    print("total cost = " + str(J_total))
    print("content cost = " + str(J_c))
    print("style cost = " + str(J_s))

    train_writer.add_summary(summary, i)
    save_image(CONFIG.OUTPUT_PATH + str(i) + ".png",
               generated_image)

save_image(CONFIG.OUTPUT_PATH + 'generated_image.jpg',
           generated_image)

return generated_image
```

Konfiguracija modela

Reguliranjem parametara modela možemo upravljati tokom učenja. Primjerice, pažljivim izborom stope učenja, ili težinskih faktora α i β funkcije gubitka, možemo dobiti bolje rezultate prijenosa stila.

Parametri modela, zajedno s pripadnim vrijednostima, definirani su u klasi `CONFIG` i moguće ih je mijenjati po potrebi. U nastavku prikazujemo kôd klase, sa stvarnim vrijednostima korištenima za učenje generirane slike. Uz pomoć službenog implementacijskog rješenja prijenosa stila iz [8], odabrani su parametri koji najbolje odgovaraju modelu. Naravno, za drugačije ulazne slike, moguće je da će neke druge vrijednosti dati bolje rezultate.

Osim standardnih parametara modela, uveli smo jedan dodatni parametar `STYLE_BLEND_WEIGHTS`, koji sadrži težine kojima reguliramo doprinos pojedine slike stila na ulaznu fotografiju.

```
class CONFIG:
    CONTENT_IMAGE = 'dog.jpg'
    STYLE_IMAGES = ('monet.jpg',)
    IMAGE_WIDTH = 600
    IMAGE_HEIGHT = 400
    COLOR_CHANNELS = 3
    NOISE_RATIO = 0.6
    MEANS = np.array([123.68, 116.779, 103.939]).reshape((1,1,1,3))
    VGG_MODEL_PATH = 'pretrained-model/imagenet-vgg-verydeep-19.mat'
    STYLE_IMAGES_PATH = 'style_images/'
    CONTENT_IMAGE_PATH = 'content_images/'
    OUTPUT_PATH = 'output_images/'
    STYLE_LAYERS_WEIGHT = 0.2
    CONTENT_LAYERS_WEIGHT = 1
    STYLE_BLEND_WEIGHTS = (4, 6)
    ALPHA = 5e0
    BETA = 1e2
    LAMBDA = 1e-03
    OPTIMIZER = 'Adam'
    LEARNING_RATE = 1e0
    ADAM_BETA1 = 0.9
    ADAM_BETA2 = 0.999
    ADAM_EPSILON = 1e-08
    ITERATIONS = 1500
    PADDING = 'SAME'
```

Poglavlje 4

Rezultati i analiza

Rezultati učenja implementiranog modela su nove slike, s naučenim vrijednostima piksela. Nove slike zadržale su oblike i sadržaj fotografija, a teksturu i boju su prilagodile zadanom umjetničkom djelu. Model je primijenjen na nekoliko ulaznih slika sadržaja, s pripadnim slikama stila. U nastavku prikazujemo nekoliko stvarnih rezultata koje je model generirao, s opisom parametara učenja.

Za vizualizaciju rezultata učenja korišten je alat Tensorboard. Uz pomoć Tensorboarda, za zadani TensorFlow graf, moguće je kreirati datoteku događaja, koja sadrži informacije o toku varijabli u jednoj sesiji učenja. Iz zadane datoteke događaja, moguće je u pregledniku rekonstruirati statističke podatke učenja u obliku grafova i histagrama.

U nastavku prikazujemo sumirane podatke učenja na jednom primjeru. Za ulaznu sliku sadržaja korištena je fotografija katedrale u Zagrebu, dok je za sliku stila korišteno poznato djelo slikara Vincenta Van Gogha, Starry Night. Vrijednosti nekih od parametara modela u izvršenoj sesiji bile su:

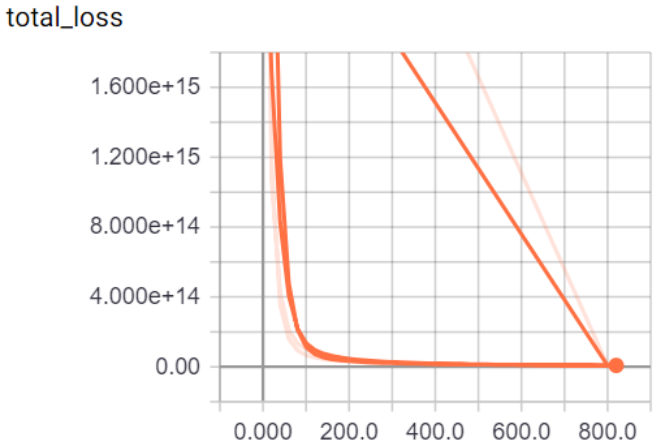
- broj iteracija = 1200
- optimizator = Adam
- stopa učenja = 1
- težina sadržaja = 10
- težina stila = 100

Na slici 4.1 prikazane su krivulje funkcije gubitka sadržaja i gubitka stila kroz iteracije učenja.

Slika 4.1: *Lijevo*: Prikaz vrijednosti funkcije gubitka sadržaja kroz 1200 iteracija učenja. *Desno*: Prikaz funkcije gubitka stila kroz 1200 iteracija učenja.

Možemo primijetiti kako funkcija gubitka stila pada mnogo brže od funkcije gubitka sadržaja. Jedan od razloga je, svakako, pridavanje većeg utjecaja gubitku stila pomoću odabranog parametra težine stila.

Na slici 4.2 prikazana je funkcija ukupnog gubitka.



Slika 4.2: Prikaz funkcije ukupnog gubitka sadržaja i stila kroz 1200 iteracija učenja.

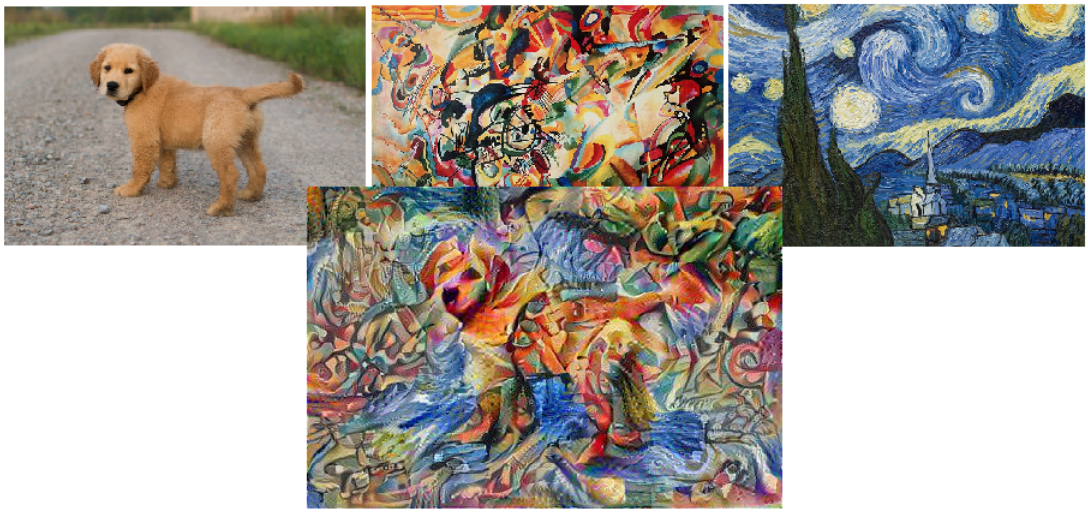
Naposljetku, ulazne slike sadržaja i stila te dobivena slika s prenesenim stilom prikazane su na slici 4.3.

Slika 4.3: *Gore lijevo*: Fotografija katedrale. *Gore desno*: Slika stila Starry Night, slikara V. Van Gogha. *Dolje*: Generirana slika s prenesenim stilom.

U nastavku prikazujemo rezultate učenja prijenosa stila s dviju ulaznih slika stila. Na fotografiji psa primijenjen je prijenos stila s poznatih djela umjetnosti, Kompozicije VII i Starry Night. Neki od parametara modela bili su:

- broj iteracija = 1000
- optimizator = Adam
- stopa učenja = 10
- težina sadržaja = 5
- težina stila = 100
- težine slika stila = $(6, 4)$

Primijetimo da smo u ovom primjeru povećali stopu učenja u odnosu na prethodni primjer, kako bi rezultati bili prije vidljivi. S vrijednostima parametra težine slika stila $(6, 4)$, odredili smo utjecaj svake od slika stila. Slici stila Kompozicija VII dali smo malu prednost u prijenosu stila, što možemo vidjeti i na slici 4.4.



Slika 4.4: *Gore lijevo*: Fotografija psa kao ulazna slika sadržaja. *Gore sredina*: Kompozicija VII, kao prva ulazna slika stila. *Gore desno*: Starry Night, kao druga ulazna slika stila. *Dolje*: Generirana slika s prenesenim stilom dviju ulaznih slika stila.

Iz dobivenih rezultata možemo zaključiti kako je model prijenosa stila vrlo osjetljiv na promjene parametara. Pažljivim odabirom, moguće je postići uvjerljiviji prijenos struktura i boja slike. Također, utjecaj na izgled slike ima i odabir slojeva mreže na kojima se model uči. Važno je napomenuti kako je vrlo teško procijeniti ishod modela, stoga je poželjno eksperimentirati s različitim parametrima, te vidjeti za koje vrijednosti se postižu zadovoljavajući rezultati.

Bibliografija

[1] *ImageNet Classification with Deep Convolutional Neural Networks*, Curran Associates, Inc., 2012, http://dl.acm.org/citation.cfm?id=2999134.2999257.

[2] *MatConvNet — Pretrained models*, http://www.vlfeat.org/matconvnet/pretrained/, posjećeno 20. 6. 2018.

[3] *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, https://www.tensorflow.org/, posjećeno 20. 6. 2018.

[4] L. A. Gatys, A. S. Ecker i M. Bethge, *A Neural Algorithm of Artistic Style*, CoRR **abs/1508.06576** (2015), http://arxiv.org/abs/1508.06576.

[5] X. Glorot i Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, Proceedings of Machine Learning Research **9** (2010), 249–256.

[6] I. Goodfellow, Y. Bengio i A. Courville, *Deep Learning*, MIT Press, 2016.

[7] S. Ioffe i C. Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, CoRR **abs/1502.03167** (2015).

[8] J. Johnson, *neural-style*, https://github.com/jcjohnson/neural-style, 2015.

[9] Y. LeCun, L. Bottou, Y. Bengio i P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE **86** (1998), 2278–2324.

[10] F. Li, J. Johnson i S. Yeoung, *Convolutional Neural Networks for Visual Recognition*, http://cs231n.stanford.edu/syllabus.html, posjećeno 17. 5. 2018.

[11] T. Šmuc, *Metode redukcije dimenzionalnosti*, https://web.math.pmf.unizg.hr/nastava/su/index.php/download_file/-/view/199/, posjećeno 10. 5. 2018.

[12] Y. Nesterov, *A Method for Solving the Convex Programming Problem with Convergence Rate* $O(1/k^2)$, Soviet Mathematics Doklady **27** (1983), 372–376.

[13] S. Pattanayak, *Pro Deep Learning with Tensorflow*, Apress, 2017.

[14] K. Simonyan i A. Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, CoRR **abs/1409.1556** (2014), `http://arxiv.org/abs/1409.1556`.

Sažetak

U ovom diplomskom radu detaljno je obrađen model konvolucijskih neuronskih mreža. U prvom dijelu rada dan je uvod u klasične neuronske mreže, radi boljeg razumijevanja arhitekture i osnovnih pojmova dubokih modela.

Drugi dio rada započinje opisom operacije konvolucije, koja se primjenjuje u konvolucijskim mrežama. U nastavku smo opisali svaki sloj konvolucijske mreže te dali matematički prikaz algoritma učenja modela. Nakon toga, naveli smo neke od najčešćih problema koji se javljaju prilikom učenja mreže te prikazali neke od optimizacijskih i regularizacijskih metoda, koje rješavaju dane probleme i poboljšavaju učenje modela.

Na samom kraju rada, opisali smo implementacijsko rješenje za problem neuronskog prijenosa stila, s umjetničke slike na odabranu fotografiju. Model je razvijen pomoću biblioteke TensorFlow za strojno učenje. Za učenje modela korištena je okolina razvojnog oblaka Google Colaboratory, koja pruža besplatnu upotrebu grafičkih kartica. Rezultati modela su slike s primijenjenim stilom drugih slika, koje su prikazane u posljednjem poglavlju rada.

Summary

In this thesis we gave a detailed view of convolutional neural network models. In the first part of the thesis, we presented a gentle introduction to classic neural networks for better understanding, with some basic concepts of deep learning models.

Second part of the thesis begins with the convolution operation description, which is the main carrier operation in convolutional neural networks. Furthermore, we described every CNN layer, and gave a mathematical background for the learning algorithm. After that, we presented some most common problems which can occur while training deep models. We described some optimization and regularization methods that can solve these problems and improve the model learning.

Lastly, we described an implementation for neural style transfer problem, from an artistic image to a chosen photography. The model is implemented using TensorFlow machine learning library. Model is learned in Google Colaboratory environment which offers free GPU support. Results of the model computations are images, with style transfered from other images, as shown in the last chapter of the thesis.

Životopis

Rođena sam 18.5.1992. godine u Trbovlju u Republici Sloveniji. Nakon selidbe u Hrvatsku 2000. godine, u mjestu Klinča Selo nastavila sam svoje osnovnoškolsko obrazovanje. Nakon završene osnovne škole, odlučila sam obrazovanje nastaviti u Zagrebu. 2007. godine upisujem Prvu gimnaziju, koju s odličnim uspjehom završavam 2011. godine. Inspiraciju za upis na Matematički odsjek PMF-a u Zagrebu dobila sam sudjelovanjem na brojnim natjecanjima iz matematike i fizike tijekom osnovnoškolskog i srednjoškolskog obrazovanja, dobivajući stalnu potporu profesora i obitelji. Nakon završenog preddiplomskog studija, upisujem diplomski studij "Računarstvo i matematika" te se zapošljavam kao junior razvojni inženjer u tvrtki *Corvus Info*, kao dio tima za interni razvoj grupacije. Osnovni zadaci s kojima sam se susrela bili su održavanje i razvoj internog sustava baziranog na Microsoftovim tehnologijama. Rad je uključivao razvoj novih aplikacija u .NET okviru, razvoj novih modela na bazi podataka, održavanje aplikacija vezanih za WMS (engl. *Warehouse management system*) i drugo. Nakon toga, u siječnju 2017. godine prelazim u tvrtku *M SAN ulaganja* unutar grupacije, s ciljem mapiranja postojećeg sustava na SAP rješenje. Učenjem od iskusnih SAP inženjera i ravijajući nekoliko aplikacija na SAP HANA XS platformi, dobivam uvid u funkcioniranje kompleksnih procesa grupacije.