RAD PREPOZNAVANJE SLIKA POMOĆU KONVOLUCIJSKE NEURONSKE MREŽE

Dilberović, Ivan

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Algebra University College / Visoko učilište Algebra**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:225:271332

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

ALGEBRA VISOKO UČILIŠTE

Repository / Repozitorij:

Algebra Univerity College - Repository of Algebra Univerity College

zir.nsk.hr

dabar DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

PREPOZNAVANJE SLIKA POMOĆU KONVOLUCIJSKE NEURONSKE MREŽE

Ivan Dilberović

Zagreb, siječanj 2020.

Student vlastoručno potpisuje Završni rad na prvoj stranici ispred Predgovora s datumom i oznakom mjesta završetka rada te naznakom:

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada“.

U Zagrebu, 17.01.2020.

Ivan Dilberović

Predgovor

Ovim putem želio bih zahvaliti supruzi Ani na velikoj potpori i podršci tijekom pisanja ovog rada.

A Ilica 242, HR-10000 Zagreb
T (01) 2222 182
E student@algebra.hr
www.algebra.hr

ALGEBRA

Temeljem članka 8. Pravilnika o završnom radu i završnom ispitu na preddiplomskom studiju Visokog učilišta Algebra sačinjena je ova

Potvrda o dodjeli završnog rada

kojom se potvrđuje da student Ivan Dilberović, JMBAG 0195017407, OIB 63649882496 u šk. godini 2018./2019., studij: Primjenjeno računarstvo - Preddiplomski studij, smjer: Programsko inženjerstvo, od strane povjerenstva za provedbu završnog ispita, dana 05.03.2019. godine, ima odobrenu izradu završnog rada

s temom: **Prepoznavanje slika pomoću konvolucijske neuronske mreže**

i sažetkom rada: Završni rad prikazuje jedan od popularnih okvira za strojno učenje (Keras) te mu je cilj pokazati kako se završni proizvod, istrenirani model, može koristiti u produkcijskom okruženju.
U sklopu rada pripremljeno je aplikacijsko rješenje koje će podržavati treniranje i testiranje konvolucijske neuronske mreže za prepoznavanje rukom napisanih brojeva.
Rješenje je napisano koristeći Python programski jezik s odgovarajućim programskim okvirima (Keras, TensorFlow).
Korisničko sučelje je web aplikacija napisana u React programskom okviru koja omogućuje prostoručni unos brojeva.
Unesena slika se šalje na prepoznavanje u konvolucijsku mrežu putem web servis napisanog u Pythonu koristeći Flask programski okvir.
Cjelokupno rješenje je isporučeno i javno dostupno putem GitHub Pages servisa te Heroku platforme.

Mentor je: Marko Velić.

Odobrenjem završnog rada studentu je omogućen upis kolegija "Izrada završnog projekta/Praksa" te je sukladno članku 8. Pravilnika o završnom radu i završnom ispitu dužan najkasnije do početka nastave ljetnog semestra u sljedećoj školskoj godini, uspješno obraniti završni rad uspješnim polaganjem završnog ispita.

U protivnom student može zatražiti novog mentora/icu i temu te ponovo upisati kolegij "Izrada završnog projekta/Praksa" budući da rad koji nije predan i obranjen na završnom ispitu u roku određenom Pravilnikom završnom radu i završnom ispitu prestaje vrijediti. Izrada novog završnog rada se izvodi sukladno rokovima određenima za školsku godinu u kojoj je studentu određen novi mentor/ica i dodijeljen novi završni rad.

Potpis studenta:	Potpis mentora:	Potpis predsjednika povjerenstva:

Ova potvrda izdaje se u 4 (četiri) primjerka od kojih 3 (tri) idu kao prilog završnom radu.

Sažetak

Strojno učenje, jedno od područja računalne znanosti, postaje sve popularnije proteklih nekoliko godina. Koristeći resurse grafičkih kartica, algoritmi su u mogućnosti obraditi sve više podataka u što kraćem vremenskom periodu te tako naučiti i dati preciznije odgovore na postavljena pitanja i probleme. Motivacija ovog rada je istraživanje jednog od popularnih okvira za strojno učenje te demonstracija kako se završni proizvod, istrenirani model, može koristiti u produkcijskom okruženju.

Ključne riječi: strojno učenje, prepoznavanje slika, MNIST set podataka, Keras, TensorFlow, Python

Summary

Machine learning as one of the fields of computer science has become increasingly popular over the past few years. Using graphic card resources, algorithms can process more and more data in as little time as possible and they can learn and provide more accurate answers to given questions and problems. The motivation behind this paper was to explore one of the popular frameworks for machine learning and to show how the final product, a trained model, can be used in a production environment.

Keywords: machine learning, image recognition, MNIST data set, Keras, TensorFlow, Python

Sadržaj

1. Uvod 3

2. Računalni vid 5

2.1. Vezana područja 5

2.2. Obrada slika naspram računalnog vida – zadaci računalnog vida 5

2.3. Značajke i klasifikatori 6

3. Strojno učenje 12

3.1. Duboko učenje 13

4. Neuronske mreže 15

4.1. Višeslojni perceptron 16

4.2. Ponavljajuće neuronske mreže 17

4.2.1. Arhitektura 17

4.2.2. Učenje parametara 18

5. Konvolucijske neuronske mreže 20

5.1. Slojevi mreže 20

5.2. Funkcije gubitka 23

5.3. Inicijalizacija težinskih vrijednosti 25

5.4. Regularizacijske metode 27

5.5. Optimizacijske metode 29

5.6. Vizualizacija 30

6. Izrada vlastitog programskog rješenja 33

6.1. Programski jezik Python 33

6.2. Keras i Tensorflow okvir 34

6.3. Izrada modela neuronske mreže 36

6.4. Treniranje mreže i pohranjivanje modela 42
6.5. Testiranje modela 44
7. Izrada web servisa 46
7.1. Python virtualna okolina 46
7.2. Flask okvir 47
7.3. Inicijalizacija modela prilikom pokretanja web servisa 48
7.4. Priprema ulaznih podataka 48
7.5. Metode web servisa 50
8. Izrada korisničkog sučelja 51
8.1. Visual Studio Code 51
8.2. JavaScript i React 52
8.3. Struktura web aplikacije 53
8.4. Komponente web aplikacije 55
8.5. Priprema slike za web servis 56
9. Implementacija web servisa i korisničkog sučelja 58
9.1. Heroku Paas (*Platform as a service*) 58
9.2. Implementacija web servisa putem Heroku platforme 58
9.3. GitHub (Sustav za verzioniranje koda) 61
9.4. Podizanje web aplikacije putem GitHub Pages servisa 62
Zaključak 64
Popis slika 66
Popis formula 68
Literatura 69

1. Uvod

U IT industriji danas strojno učenje i umjetna inteligencija privlače sve više pažnje. Velike kompanije žele saznati što je više moguće informacija o svojim korisnicima prateći njihove svakodnevne rutine i postupke što im omogućavaju velike količine podataka koje korisnici generiraju na dnevnoj bazi. Pri tome im uvelike pomažu algoritmi strojnog učenja koji prepoznaju razne uzorke u velikim količinama podataka te na taj način pokušavaju predvidjeti ponašanje korisnika. Ovakve operacije su skupe i zahtijevaju puno resursa koji nisu bili uvijek dostupni. Iz tog razloga strojno učenje, koje se spominjalo već šezdesetih godina prošlog stoljeća, tek unazad desetak godina postaje popularno i pridaje mu se sve više značaja. Razlog tome je veliko poboljšanje računalnih komponenti kao što su procesori i grafičke kartice koje se sve više koriste prilikom izračuna matematičkih operacija, a koje koriste algoritmi strojnog učenja. Ovo nam je omogućilo da se treniranje neuronskih mreža za neki lakši zadatak svede na svega nekoliko minuta rada na prosječnom laptopu ili stolnom računalu. Takvo poboljšanje privuklo je sve više inženjera koji su počeli isprobavati i proučavati ovo zanimljivo područje, što je rezultiralo mnogobrojnim novim bibliotekama i okvirima za razne programske jezike koji uvelike ubrzavaju i olakšavaju samo korištenje algoritama strojnog učenja. U ovom sam radu htio isprobati jedan takav programski okvir i pokušati napraviti aplikaciju koja bi u konačnici koristila jedan istrenirani model u produkcijskom okruženju. Većina tutoriala na internetu pokazuje načine korištenja raznih modela na lokalnom računalu gdje prilikom testiranja koriste podatke iz istog seta, a koje nisu koristili prilikom treniranja. Cilj je bio napraviti i testirati model koji će nakon treniranja dobivati ulazni parametar od strane korisnika. Na taj način mogu vidjeti ponašanje modela u stvarnom okruženju u kojem sam siguran da model takav ulazni parametar nije imao priliku vidjeti. Aplikacija se sastoji od neuronske mreže napisane u programskom jeziku Python, koristeći okvire strojnog učenja Keras i Tensorflow. Nakon treniranja model se sprema i daje na korištenje *web API-ju* koji je također napisan u Pythonu koristeći Flask okvir. *Web API* je implementiran i javno dostupan putem Heroku platforme. Korisnik će moći putem web aplikacije isprobavati točnost i predviđanja modela vlastitim unosom. Aplikacija je napisana u JavaScript programskom jeziku koristeći React okvir. Također, web aplikacija je javno dostupna i implementirana putem GitHub Pages servisa. Što se tiče modela i skupa podataka, odlučio sam se za „MNIST“ (engl. *Modified National Institute of Standards and Technology*

database). To je set slika koji prikazuje rukom napisane brojeve od 0 do 9. Slike su obrađene i pripremljene unaprijed, tako da je to idealan početnički set za proučavanje algoritama strojnog učenja. Set se sastoji od 60 000 slika koje se koriste u treniranju i 10 000 slika koje na kraju koristimo za evaluaciju i testiranje treniranog modela prije spremanja. Nakon što je model neuronske mreže istreniran na spomenutom skupu podataka, bit će isporučen putem *web API-ja*. Nakon predanog unosa korisnika odgovorit će koji je broj je korisnik nacrtao, koliki je postotak sigurnosti te će isporučiti set slika iz prvih par slojeva mreže kako bi korisnik imao uvid u put slike kroz slojeve do krajnjeg rezultata.

2. Računalni vid

Računalni vid je znanost koja nastoji kopirati, odnosno funkcionalno nadmašiti ljudski vidni sustav – omogućiti računalima vizualno razumijevanje okoline. Istraživači ovo nastoje realizirati razvojem matematičkih postupaka koji iz dvodimenzionalnih prikaza konstruiraju tri dimenzije. Bez obzira na postignuti napredak u ovoj grani računalne znanosti, približavanje računalne razine percepcije onoj ljudskoj i dalje je puno izazova.[1]

2.1. Vezana područja

Ostvareni napredak na području računalnog vida manifestira se u njegovoj sve široj primjeni, od interakcije čovjeka i računala preko robotike do multimedije. Prepoznavanjem pokreta na snimkama video kamera postaje moguća komunikacija korisnika i uređaja, dok vješto kombiniranje tehnologija računalnog vida i naprednog hardvera čini osnovu nove generacije robota prikladnih za raznovrsne zadaće koje čovjek obavlja refleksno. Zahvaljujući razvoju algoritama procesuiranja, analize i interpretacije multimedijskih podataka, računalni vid ima ključnu ulogu u razvoju multimedijalnih aplikacija omogućujući, primjerice, pronalazak više milijuna srodnih videa na temelju kratkog video isječka.[2]

2.2. Obrada slika naspram računalnog vida – zadaci računalnog vida

Obrada slika je korak koji prethodi računalnom vidu. Preciznije, cilj obrade slika je ekstrakcija njenih osnovnih elemenata poput rubova, uglova, filtera i slično. Prije procesa semantičke segmentacije potrebno je provesti filtraciju slike. Glavna razlika između obrade slika i računalnog vida jest povratna informacija koja je sastavni dio računalnog vida, a izostaje kod obrade.[3] Prema razini apstrakcije izlaznih podataka, zadaci računalnog vida dijele se u tri kategorije: nižu, srednju i visoku. Zadaci niže razine računalnog vida

[1] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 1.
[2] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 3.
[3] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 3.

korištenjem izvučenih elemenata slike mogu se primjenjivati na slikovnom i video materijalu. Uspoređivanje slika (automatska identifikacija odgovarajućih točaka slika istog prikaza iz različitih kutova gledanja) primjer je zadatka niže razine računalnog vida. Istoj kategoriji pripada računanje optičkog protoka i analiza pokreta. Optički protok je dvodimenzionalno vektorsko polje u kojem svaki vektor odgovara vektoru pomaka pokazujući pomak točaka između kadrova.[4] Geometrijski vid kategorizira se kao zadatak srednje razine računalnog vida koja podrazumijeva višu razinu apstrakcije. Geometrijski vid uključuje geometriju višestrukog prikaza, stereo i višestruku sliku iz pokreta (*SfM*) te na taj način dolazi do trodimenzionalnih podataka iz dvodimenzionalnih prikaza čime postaje moguća 3D rekonstrukcija.[5] Istoj kategoriji pripada vizualno praćenje i snimanje pokreta kojim se procjenjuju dvodimenzionalne i trodimenzionalne kretnje. Koristeći rezultate niže i srednje razine, računalni vid visoke razine zaokružuje interpretaciju slike - utvrđuje prisutnost i korelaciju objekata. Prepoznavanje objekata i razumijevanje prikaza te razumijevanje slika i videa zadatci su visoke razine računalnog vida.[6]

2.3. Značajke i klasifikatori

Dvije ključne faze računalnog vida su ekstrakcija i klasifikacija značajki. Kvaliteta značajki i klasifikatora utječe na preciznost i učinkovitost cjelokupnog vizualnog sustava. Značajka je svaka svojstvena karakteristika koja se upotrebljava za rješavanje računalnog zadatka određene aplikacije. Kombinaciju *n* značajki predstavlja *n*-dimenzionalno vektorsko polje naziva „vektor značajki“ (engl. *feature vector*). Kvaliteta vektora značajki ovisi o njegovoj mogućnosti razlikovanja uzoraka slika različitih razreda. Naime, uzorci slika istog razreda trebaju imati istu vrijednost značajke. Dobra značajka je informativna neovisno o potencijalnim transformacijama.[7]

Zadatak klasifikatora je vektorom značajki svrstati sliku ili „područje interesa“ (engl. *region of interest RoI*) odgovarajućoj kategoriji, dok je težina klasifikacije ovisna o značajkama slika iste kategorije, relativna u odnosu na razliku značajki slika različitih kategorija. Zbog prisutnosti sjena, prepreka, problematičnih perspektiva, elemenata koji ne

[4] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 3-4.
[5] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 4.
[6] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 4.
[7] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 11-12.

pripadaju i drugih nejasnoća, savršena klasifikacija nije moguća te i dalje predstavlja izazov.[8]

Metode prve faze računalnog vida, ekstrakcije, dijele se u dvije kategorije: globalnu i lokalnu. Zadatak metoda prve kategorije jest definirati opće značajke koje opisuju sliku u globalu, zanemarujući detalje. Metode druge kategorije prikladnije su za rad s prikazima djelomično zaklonjenih predmeta jer ekstrakciju vrše lokalno, oko ključnih točaka.[9] Lokalne značajke/deskriptori poput HOG-a, SIFT-a ili SURF-a koriste se u većini aplikacija računalnog vida.

„Histogram usmjerenih gradijenata“ (HOG) je deskriptor slike kojim se utvrđuje prisutnost objekta na slici – kodira distribuciju smjerova gradijenata u pojedinim dijelovima slike. Gradijent slike predstavlja promjenu intenziteta piksela slike ili jednog dijela iste. Iz rasporeda piksela vidljivo je kako se mijenja vrijednost intenziteta osvijetljenosti u bilo kojem smjeru te je moguće izračunati histogram. Histogram usmjerenih gradijenata zasnovan je na tezi kako se pojavnost objekta i oblika na slici može opisati histogramom smjerova rubova, a izvodi se u četiri koraka: izračunavanje gradijenta, histograma ćelije, blokova deskriptora i normalizacija blokova deskriptora.[10]

Prvi korak je primjena „centrirane diskretne izvedbene maske“ (engl. *discrete derivative mask*) u vodoravnim i okomitim smjerovima. Ovo zahtijeva filtriranje crno-bijele slike:

$$f_x = [-10+1] \tag{1}$$

$$f_y = [-10+1]^T \tag{2}$$

U slučaju slike i konvolucijske operacije generiraju derivacije slike I u x i y smjerovima :

$$I_x = I * f_x \tag{3}$$

$$I_y = I * f_y \tag{4}$$

Sukladno navedenom, orijentacija θ i veličina $|g|$ gradijenta računaju se prema sljedećoj formuli:

[8] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 13.
[9] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 13.
[10] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 14.

$$|\theta = |\arctan\frac{I_y}{I_x} \tag{5}$$

$$|g| = \sqrt[2]{I_x^2 + I_y^2} \tag{6}$$

Izračunavanje histograma ćelije je drugi korak. Slika se podijeli u manje ćelije, najčešće 8 X 8 piksela. Pritom svaka ćelija ima fiksni broj „gradijentnih orijentacijskih polja“ (engl. *gradient orientation bins*) ravnomjerno raspoređenih između 0˚ i 180˚ ili 0˚ i 360˚, ovisno o tome je li gradijent pripisan ili ne. Svaki piksel unutar ćelije ima različiti utjecaj na gradijent orijentacijskih polja sukladno veličini gradijenta predmetnog piksela. Kako bi se uvrstile promjene u osvjetljenju i kontrastu, u trećem se koraku gradijenti normaliziraju povezivanjem ćelija u veće blokove pa je HOG deskriptor vektor komponenata normaliziranih histograma ćelija svih blokova.[11] Neka je u završnom koraku v nenormalizirani vektor koji sadrži sve histograme pojedinog bloka, $||v||_k$ njegova k norma za k=1,2, a ε konstanta. Normalizacijski faktor može biti:[12]

$$L2 - norm: v = \frac{v}{\sqrt{||v||_2^2 + e^2}} \tag{7}$$

ili

$$L1 - norm: v = \frac{v}{||v||_1 + \epsilon} \tag{8}$$

ili

$$L1 - sqrt: v = \sqrt{\frac{v}{||v||_1 + \epsilon}} \tag{9}$$

Krajnja slika formira se povezivanjem svih normaliziranih blokova deskriptora. Dosadašnja istraživanja pokazala su kako normalizacijske metode postižu bolje rezultate od nenormalizacijskog pristupa.[13]

[11] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 15.
[12] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 15.
[13] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 15.

Scale-invariant feature transform (SIFT) pruža set značajki neovisnih o promjeni dimenzija i rotaciji objekta. SIFT algoritam također se sastoji od četiri koraka. Prvi korak cilja na utvrđivanje mogućih ključnih točaka nepromjenjivih spram orijentacije i skale. Kako bi se utvrdile ključne točke, SIFT koristi „razliku među distribucijama Gaussa" (engl. *Difference of Gaussian DoG*), zamućivanje slike prema dvije različite skale za različite oktave slike u Gaussovoj piramidi. Slijedi pretraživanje lokalnih ekstrema u svim skalama i lokacijama pa se piksel uspoređuje sa susjednima unutar iste slike kao i onima u skalama iznad i ispod. Ako se pojedina točka ponavlja kao minimum ili maksimum, riječ je o potencijalnoj ključnoj točki.[14] U sljedećem se koraku Taylorovim izračunom za serijsko širenje prostora razmjera dobiva precizniji položaj ekstrema. Ako je intenzitet u svakom ekstremu manji od vrijednosti praga, ključna točka se odbacuje.[15] *DoG* funkcija ima snažne reakcije na bridovima što rezultira velikom primarnom zakrivljenošću na bridu, ali malom u okomitom smjeru *DoG* funkcije. Kako bi se uklonile ključne točke smještene na rubovima, primarna zakrivljenost ključne točke računa se u 2X2 Hessianovoj matrici na poziciji i skali ključne točke. Ako je omjer prve i druge svojstvene vrijednosti matrice veći od rubne vrijednosti, ključna točka se eliminira.[16] Kako bi se postigla postojanost uslijed rotacije slike, svakoj se ključnoj točki na temelju lokalnih svojstava dodjeljuje stalna orijentacija. Algoritam utvrđivanja orijentacije sastoji se od više etapa: odabira Gaussove zamućene slike najbliže skali ključne točke, računanja veličine i orijentacije svakog piksela predmetne skale, izrade orijentacijskog histograma koji sadrži trideset i šest polja raspona 360˚ od gradijentnih usmjerenja piksela koji okružuju ključnu točku – najviši vrh lokalnog orijentacijskog histograma odgovara prevladavajućem smjeru lokalnih gradijenata. Dominantno usmjerenje lokalnih gradijenata koristi se u kreiranju deskriptora ključnih točaka. Smjernice gradijenata rotiraju se ovisno o orijentaciji ključne točke, a zatim ponderiraju normalnom raspodjelom s varijacijom 1,5 skale ključne točke. Šesnaest puta šesnaest okruženje dijeli se u šesnaest podblokova 4X4 veličine. Za svaki podblok oblikuje se orijentacijski histogram od osam polja što rezultira vektorom vrijednosti značajki, SIFT deskriptorom, koji se sastoji od sto dvadeset i osam elemenata.[17]

---

[14] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 17.
[15] Srivastava, Goyal, An Efficient Image Identification Algorithm using Scale Invariant Feature Detection, 2007, 2.
[16] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 17.
[17] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 18.

SIFT pokušava standardizirati sve slike - ako se ključna točka može utvrditi u skali σ, potrebna je kσ dimenzija kako bi se zahvatila ista ključna točka uvećane slike. Matematička pozadina SIFT-a je, međutim, poprilično komplicirana i time izazovna za istraživače.[18]

Speeded-up robust features (SURF) je zapravo ubrzana verzija SIFT-a koja aproksimira LoG „filterom okvira" (engl. *box filter*). Konvulacija s filterom okvira može se izračunati pomoću integralnih slika i to paralelno za različite skale. Na samom se početku kao osnova za lociranje ključnih točaka koristi Hessianova matrica. Determinantom Hessianove matrice selekcionira se pozicija i skala potencijalnih ključnih točaka. Konkretno, za sliku I i danu točku p=(x,y), Hessianova matrica H(p,σ) za točku p i skalu σ jest:

$$H(p,\sigma) = \begin{bmatrix} L_{xx}(p,\sigma) & L_{xy}(p,\sigma) \\ L_{xy}(p,\sigma) & L_{yy}(p,\sigma) \end{bmatrix} \tag{10}$$

gdje je Lxx (p,σ) Gaussova konvulacija derivacije drugog reda normalne razdiobe, kod slike I i točke p.[19] SURF koristi aproksimativne Gaussove derivacije drugog reda koje se mogu izračunati pomoću integralnih slika. Za razliku od SIFT-a, SURF ne zahtijeva uzastopnu primjenu istog filtera na izlaze prethodno filtriranih slojeva te se analiza vektorskog polja provodi zadržavanjem iste slike i promjenom veličine filtera. Najviše vrijednosti determinante Hessianove matrice interpoliraju se u skalu i polje slike metodom istraživača Brown i Lowe.[20] Slijedi izračun „Haar valićnih transformacija" (engl. *Haar wavelet transform*) u x horizontalnih i y vertikalnih smjerova u okruženju ključne točke radijusa 6s. Iste se potom ponderiraju i iskazuju kao točke u dvodimenzionalnom prostoru. Dominantni smjer ključne točke procjenjuje se na temelju sume svih reakcija unutar kliznog orijentacijskog prozora pod kutom od 60. Horizontalne i vertikalne reakcije se zbrajaju te se dva zbroja smatraju lokalnim vektorom. Najdulji vektor u svim prozorima određuje orijentaciju ključne točke.[21] Kako bi se opisao prostor oko svake ključne točke p, izvlači se kvadratni 20sX20s prostor oko točke p te usmjerava prema orijentaciji točke p. Normalizirani orijentacijski prostor oko točke p dijeli se u manje kvadratne dijelove (4X4). Za svaki se izdvajaju Haar valne reakcije u paralelnim i okomitim smjerovima koje se potom zbrajaju za svaki od manjih dijelova te kao suma ulaze u vektor značajki. Vektoru

[18] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 19.
[19] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 20.
[20] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 20.
[21] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 20.

se također dodaju apsolutne vrijednosti reakcija. Budući da svaki manji dio ima 4D vektor značajke, povezivanje svih 4X4 dijelova rezultira 64D deskiptorom.[22]

HOG, SIFT, SURF i slični algoritmi donedavno su bili osnova razvoja računalnog vida. Međutim, kako zahtijevaju previše stručnog znanja i vremena te su manjkavi u prikupljanju informacija iz slika, potisnuti su kombiniranjem računalnog vida i strojnog učenja, odnosno, nastankom fleksibilnih i otpornih algoritama računalnog vida.[23]

[22] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 21.
[23] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 4-5.

3. Strojno učenje

Strojno učenje je vrsta umjetne inteligencije koja računalima omogućuje učenje iz podataka bez izričitog programiranja.[24] Za razliku od klasičnog programiranja gdje programer razvija program i unosi podatke za obradu, kod strojnog učenja unose se podaci kao i očekivani ishodi. Tako se „uče“ pravila za buduću obradu.[25] Konkretnije, cilj strojnog učenja je razvoj metoda automatskog učenja promatranjem. Model strojnog učenja preobražava ulazne podatke u smislene izlazne. To je proces naučen kroz izloženost poznatim primjerima ulaznih i izlaznih podataka.[26] Središnji problem strojnog učenja postaje smislena preobrazba podataka – učenje korisnih reprezentacija ulaznih podataka.[27]

Predložen je niz algoritama namijenjenih različitim tipovima podataka i problema. Iako razlikujemo tri različita pristupa kod metoda učenja, nadzirani, polunadzirani i nenadzirani pristup, zahvaljujući boljim rezultatima većina aktualnih metoda strojnog učenja pripada prvoj skupini. Kod nadziranih metoda „set podataka za treniranje“ (engl. *training set*) poprima formu parova (x,y) s ciljem predviđanja odgovora y* na upit x. Funkcija f(x) predviđa izlaznu varijablu y za svaki x, pri čemu ulazni x može biti vektor značajki ili pak složeniji tip podataka poput slika, grafova i dokumenata. Raspon izlaznog y kreće se od binarne oznake do multidimezionalnih sekvenci predviđanja.[28] Neke od postojećih funkcija su binarna stabla, „slučajne kombinacije stabala“ (engl. *Random decision trees* RDF), „metoda potpornih vektora“ (engl. *Support Vector Machines* SVM), Bayesovi klasifikatori i neuronske mreže (NN).[29]

Za razliku od nadziranih metoda učenja, nenadzirane metode uključuju samo ulazne podatke x bez korespondirajućih izlaznih varijabli. One su namijenjene modeliranju osnovne strukture, odnosno distribucije podataka radi detekcije zanimljivih obrazaca. Grupiranje je najčešća nenadzirana metoda učenja. Postoje hijerarhijsko grupiranje, „klasterizacija metodom k-srednjih vrijednosti“ (engl. *k-means clustering*), *Gaussian*

[24] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 5.
[25] Chollet, Deep Learning with Python, 2018, 5.
[26] Chollet, Deep Learning with Python, 2018, 6.
[27] Chollet, Deep Learning with Python, 2018, 6.
[28] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 5.
[29] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 5.

Mixture Models (GMMs), samoorganizirajuće mape (SOMs), *Hidden Markov Models* (HMMs).[30] Polunadzirane metode nalaze se između nadziranih i nenadziranih, a koriste se u situacijama kada je dostupna velika količina djelomično označenih ulaznih podataka.[31]

3.1. Duboko učenje

Duboko učenje grana je strojnog učenja koja se zasniva na predstavljanju podataka složenim reprezentacijama na visokom stupnju apstrakcije. Do njih se dolazi putem naučenih nelinearnih transformacija. Metode dubokog učenja primjenjuju se u različitim područjima: automobilskoj industriji, medicinskoj dijagnostici, računalnom vidu, obradi prirodnog jezika, razumijevanju govora i zvučnih signala.[32]

Počevši od 2010. godine, izuzetni rezultati dubokog učenja u rješavanju perceptivnih problema doprinijeli su revoluciji cjelokupnog znanstvenog područja. Računalna klasifikacija prikaza, prepoznavanje govora, rukopis ili vožnja približeni su ljudskoj.[33] Duboko učenje automatizira ključni korak strojnog učenja, „inženjering značajki“ (engl. *feature engineering*). Potiskivanjem intervencije inženjera, odnosno omogućavanjem istovremenog zajedničkog učenja slojeva reprezentacija, bitno se pojednostavljuje i pospješuje radni proces strojnog učenja.[34] Sve navedeno bilo je moguće zahvaljujući napretku hardvera, podataka, referentnih programa i algoritama.

Intenzivna ulaganja industrije videoigara u razvoj grafičke kartice postupno su omogućila potiskivanje paralelne uporabe više procesora. Velike kompanije su počele uvježbavati modele dubokog učenja na skupinama više grafičkih kartica specijaliziranih za potrebe dubokog učenja, dok moderna industrija investira u čipove namijenjene dubokim neuronskim mrežama koji su deset puta brži i učinkovitiji od vrhunske grafičke kartice.[35] Pored velikog unapređenja skladišnog hardvera, kao ključni faktor pokazat će se uzlet Interneta koji je omogućio prikupljanje i distribuciju velikih setova podataka potrebnih za strojno učenje. Danas su velikim kompanijama na raspolaganju baze video materijala i slikovnih i jezičnih podataka koje su nastale zahvaljujući Internetu. Posebne zasluge

[30] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 5.
[31] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 6.
[32] Ciaburro, Venkateswaren, Neural Networks with R, 2017,137.
[33] Chollet, Deep Learning with Python, 2018, 11-12.
[34] Chollet, Deep Learning with Python, 2018, 17-18.
[35] Chollet, Deep Learning with Python, 2018, 20-21.

razvoju dubokog učenja pridaju se bazi ImageNet koja sadrži otprilike milijun i četiristo tisuća slika organiziranih u tisuću kategorija.[36]

Napredak strojnog, odnosno dubokog učenja, sve do unazad dvadeset godina bio je usporen manjkom pouzdanog načina učenja dubokih neuronskih mreža. Kao ključan problem nametnula se propagacija gradijenata kroz mnogo slojeva. Naime, povratni signal presudan za učenje neuronskih mreža gubio se proporcionalno povećanju broja slojeva. Prije desetak godina dolazi do nekoliko algoritamskih poboljšanja koja će pospješiti propagaciju gradijenata: bolje aktivacijske funkcije za neuronske slojeve, optimizacijske sheme i sheme inicijalizacije težinskih vrijednosti. Time su postali mogući modeli dubokog učenja od deset i više slojeva. Odnedavno su „normalizacija serija podataka“ (engl. *batch normalization*), „rezidualne veze“ (engl. *residual connections*) i „konvolucije djeljive po dubini“ (engl. *depthwise separable convolutions*) stvorile uvjete za učenje modela od više tisuća slojeva.[37]

Zahvaljujući pobrojanim poboljšanjima, duboko učenje nudi nekoliko presudnih prednosti. Duboke mreže sadrže jednostavne, lako podesive građevne blokove iskoristive za niz srodnih zadataka.[38]

[36] Chollet, Deep Learning with Python, 2018, 21.
[37] Chollet, Deep Learning with Python, 2018, 21-22.
[38] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 6.

4. Neuronske mreže

Dubina pojedinog modela dubokog učenja ovisi o brojnosti slojeva reprezentacije podataka koji mogu varirati od nekoliko desetaka do nekoliko stotina uzastopnih slojeva.[39] Gotovo se uvijek uče putem takozvanih neuronskih mreža. Neuronske mreže inspirirane su načinom na koji funkcionira ljudski mozak tijekom procesa učenja. Prvi model neuronske mreže su 1943. godine predstavili Warren McCulloch i Walter Pitts dokazavši kako jednostavne neuronske mreže mogu izračunati svaku aritmetičku ili logičku funkciju. Njihovim radom započela je era neuroračunalstva temeljenog na umjetnim neuronskim mrežama sadržanim od neurona i veza između njih. Neuron je najmanja procesorska jedinica koja prima unos da bi ga potom pomnožila s težinom veze kojom se ostvario unos. Zatim računa aktivaciju neurona jednom od aktivacijskih funkcija i u konačnici šalje sljedećem neuronu u mreži. Neuroni se pretežno grupiraju u slojeve prema danom zadatku i arhitekturi same mreže, a povezani su vezama čija se težina mijenja tijekom faze učenja.[40] Upravo je u snazi veza između neurona pohranjena moć obrade same mreže.[41]

Razlikujemo tri dijela/sloja neuronskih mreža: ulazni, skriveni i izlazni sloj. Prvi je zadužen za prihvat ulaza - informacija, značajki ili mjera iz vanjskog okruženja čijom se normalizacijom postiže bolja preciznost matematičkih funkcija u mreži. Drugi je sastavljen od neurona sa zadaćom ekstrakcije uzoraka povezanih s procesom koji se analizira. Treći sloj sadrži neurone zadužene za prezentaciju izlaza mreže. Prema tome, osnovna arhitektura neuronskih mreža prema poziciji i povezanosti neurona dijeli se na jednoslojne i višeslojne unaprijedne mreže te povratne mreže.[42]

Neuronske mreže su dobre za procjenu nelinearnih odnosa u uzorcima podataka, otporne na fragmentirane ili oštećene podatke, a svoje veze mogu stvarati i između podataka kada iste nisu izričito zadane. Primjenjive su kod višebrojnih unosa te prilagodljive okolini.[43]

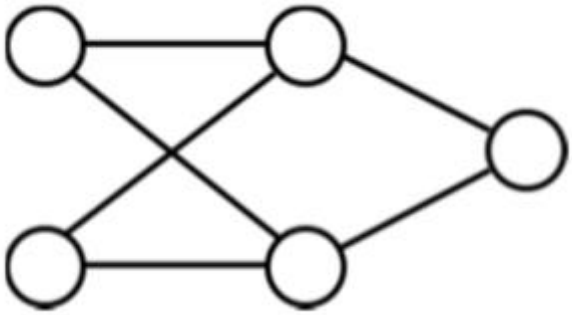
---

[39] Chollet, Deep Learning with Python, 2018, 8.
[40] Karačić, Korištenje web tehnologija za izradu i prikaz višeslojnih neuronskih mreža, 2016, 2.
[41] Božić-Štulić, Semantička segmentacija slika metodama dubokog učenja, 2017, 5.
[42] Božić-Štulić, Semantička segmentacija slika metodama dubokog učenja, 2017, 6.
[43] Karačić, Korištenje web tehnologija za izradu i prikaz višeslojnih neuronskih mreža, 2016, 2.

4.1. Višeslojni perceptron

Dok je jednoslojni perceptron najjednostavniji oblik neuronske mreže, višeslojni perceptron jedna je od popularnijih struktura neuronskih mreža koja ujedno nadilazi ograničenja jednoslojnog perceptrona. Između ulaznog i izlaznog sloja višeslojnog perceptrona nalazi se barem jedan skriveni sloj neurona čiji izlaz postaje ulaz izlaznog ili sljedećeg skrivenog sloja. Zahvaljujući takvoj strukturi, mreža ima sposobnost obrade nelinearno razdvojivih problema što otvara mogućnost učenja bilo koje matematičke i logičke operacije.

Mreža će biti N-slojna kada je sadržana iz N slojeva i N ne-ulaznih slojeva neurona. Prema tome, jednostavna dvoslojna mreža imat će dva ulazna, dva skrivena i jedan izlazni neuron.[44]



Slika 4.1. Višeslojni perceptron s dva ulazna, dva skrivena i jednim izlaznim neuronom

Optimalan broj skrivenih slojeva nalazi se između broja ulaznih i izlaznih. Ako ima premalo skrivenih slojeva, mreža neće biti u mogućnosti donositi složene odluke. U suprotnom slučaju, nastat će problem s generalizacijom. Odluka o broju skrivenih slojeva ovisit će o broju ulaznih i izlaznih neurona, algoritmu za učenje, složenosti same klasifikacije, ali i šumu u podacima.[45]

Postoji niz matematičkih funkcija koje se koriste kao aktivacijske funkcije neurona. Daleko najpopularniji algoritam za učenje višeslojnih perceptrona je algoritam „propagacije unatrag“ (engl. *Backpropagation – BP*). Sastoji se od dvije faze kroz koje optimizira težine veza u mreži kako bi ona mogla što kvalitetnije preslikavati ulazne u poželjne izlazne vrijednosti. U fazi propagacije unaprijed ulazne se vrijednosti do kraja prenose mrežom, da

[44] Karačić, Korištenje web tehnologija za izradu i prikaz višeslojnih neuronskih mreža, 2016, 8.
[45] Karačić, Korištenje web tehnologija za izradu i prikaz višeslojnih neuronskih mreža, 2016, 9.

bi se u fazi propagacije unatrag izračunavala razlika između željene i dobivene izlazne vrijednosti, a zatim propagirala natrag kroz mrežu gdje se prilagođava težina veze radi minoriziranja ukupne greške.[46]

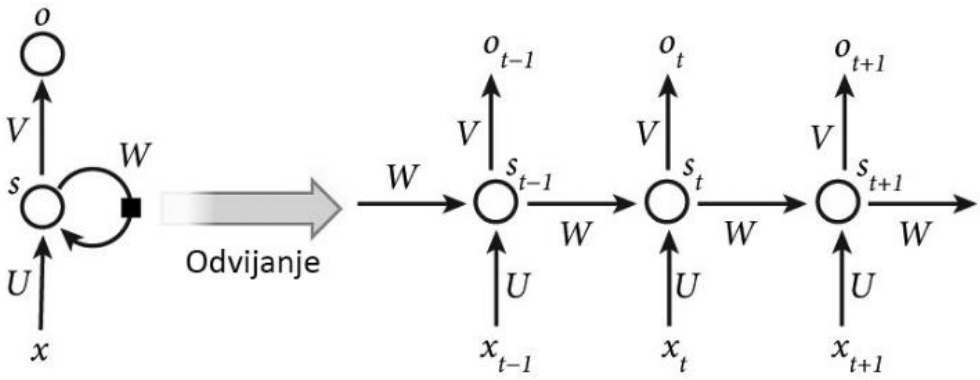
Prije algoritma unazadne propagacije najpopularnija aktivacijska funkcija bila je sigmoid funkcija koja uneseni realni broj smješta između 0 i 1. Pritom veliki negativni brojevi postaju 0, a veliki pozitivni 1. Glavni nedostatak sigmoida jest mogućnost zasićenja na nekoj od krajnjih vrijednosti čime se poništava gradijent. Ovo zahtijeva oprez s inicijaliziranim vrijednostima. Isti problem zasićenja ima i funkcija tangens koja uneseni realni broj smješta između -1 i 1. Zahvaljujući određenim prednostima u odnosu na sigmoid i tagens, jedna od korištenijih aktivacijskih funkcija jest ReLU (engl. *Rectified Linear Unit*). ReLU ubrzava konvergenciju algoritma stohastičkog gradijenta pada, a istovremeno nema operacija poput kvadriranja. Međutim, kod velike stope učenja i velikog gradijenta postoji opcija gradijenta 0 kod nekih jedinica.[47]

4.2. Ponavljajuće neuronske mreže

4.2.1. Arhitektura

„Ponavljajuće neuronske mreže“ (engl. *recurrent neural networks – RNN*) izvode isti zadatak za svaki element u slijedu, pri čemu izlaz svake operacije ovisi o izlazu prijašnje operacije.[48] Na slici (Slika 4.2.) prikazana je RNN arhitektura koja sadrži povratnu petlju. Njeno djelovanje prikazuje se odvijanjem ponavljajuće neuronske mreže kroz vrijeme. RNN funkcionira kao jednostavna višeslojna mreža gdje se protok informacija odvija tijekom vremena, dok različiti slojevi predstavljaju računske izlaze različitih vremenskih koraka.[49]

[46] Karačić, Korištenje web tehnologija za izradu i prikaz višeslojnih neuronskih mreža, 2016, 10-11.
[47] Karačić, Korištenje web tehnologija za izradu i prikaz višeslojnih neuronskih mreža, 2016, 9-11.
[48] Karačić, Korištenje web tehnologija za izradu i prikaz višeslojnih neuronskih mreža, 2016, 18.
[49] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 37.



Slika 4.2. RNN mreža i odvijena RNN mreža[50]

Ponavljajuće neuronske mreže mogu raditi s ulaznim podacima promjenjivih duljina, primjerice video materijalom varijabilnih duljina kadrova. Pritom duljina odvijene RNN strukture ovisi o duljini ulazne sekvence pa će tako za rečenicu od deset riječi biti ukupno deset slojeva odvijene RNN. Ona u skrivenom stanju sprema internu memoriju ranijeg računalnog slijeda što je zapravo ulaz od prethodnog sloja u odvijenoj RNN strukturi. Ponavljajuće neuronske mreže imaju sposobnost generiranja izlaznih podataka varijabilnih duljina. Primjerice, prijevod rečenice s jednog jezika na drugi gdje se duljina izlaznog slijeda razlikuje od ulaznog. Ovo je moguće jer uzimaju u obzir skriveno stanje koje na temelju prethodno procesuiranog slijeda predviđa nove izlaze.[51]

4.2.2. Učenje parametara

Svi parametri ponavljajuće neuronske mreže dijele se u svakom vremenskom koraku. Za njihovo učenje koristi se jedna vrsta algoritma unazadne propagacije, „algoritam unazadne propagacije kroz vrijeme“ (engl. *backpropagation through time – BPTT*). On se od prvog razlikuje po gradijentu koji ovisi o izračunima svih vremenskih koraka učenja. Kada rade s velikim brojem sljedova, ponavljajuće neuronske mreže mogu naići na probleme u učenju pomoću BPTT algoritma. Osim toga, u mrežama dubokog učenja gdje se slojevi povezuju množenjem, gradijent pomnožen već i malim brojem može eksplodirati ili pak nestati ako se pomnoži brojem manjim od nule. Prvi problem je korištenjem matematičkih operacija poput sigmoida ili tangensa lako rješiv, dok je drugi koji podrazumijeva vrijednosti težina

[50] Karačić, Korištenje web tehnologija za izradu i prikaz višeslojnih neuronskih mreža, 2016, 19.
[51] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 37-38.

premalih za računala, izazovniji. Navedene poteškoće su ujedno glavni razlog zašto se izvorne ponavljajuće neuronske mreže danas rjeđe koriste.[52]

[52] Karačić, Korištenje web tehnologija za izradu i prikaz višeslojnih neuronskih mreža, 2016, 19-20.

5. Konvolucijske neuronske mreže

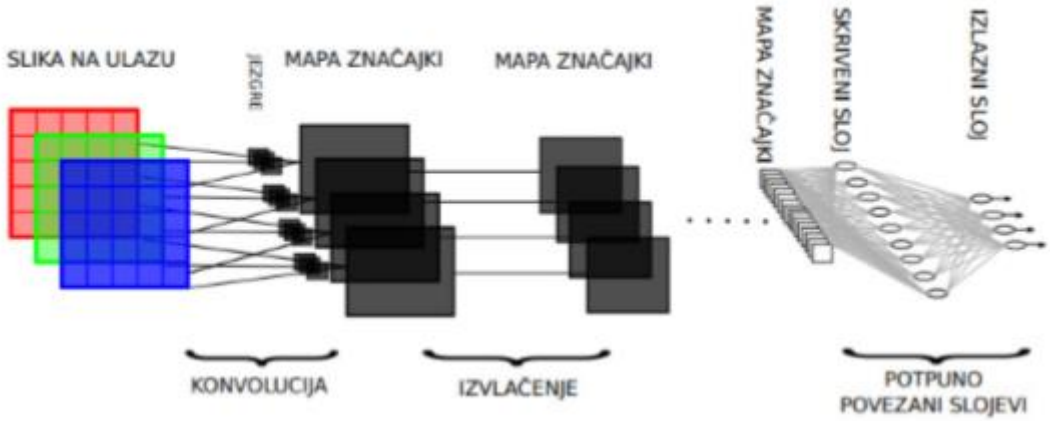
Neuronske mreže topologije rešetke specijalizirane za obradu podataka nazivaju se „konvolucijske neuronske mreže“ (engl. *convolutional neural network – CNN*).[53] Predstavljaju jednu od popularnijih kategorija neuronskih mreža, posebice za rad s višedimenzionalnim podacima. Funkcioniraju poput standardnih neuronskih mreža, uz razliku da je svaka jedinica pojedinog sloja konvolucijske mreže (minimalno) dvodimenzionalni filter zavijen s ulazom sloja. Ovo je presudno za prepoznavanje uzoraka kod ulaznog slikovnog ili video materijala. Sami filteri imaju sličan, ali manji prostorni oblik u odnosu na ulazni medij te koriste zajedničke parametre radi smanjenja broja varijabli za učenje.[54]

Konvolucijske neuronske mreže su korisne za nadzirano i nenadzirano učenje. Kod nadziranog učenja poznati su ulazni i izlazni podaci, a model uči preslikavanje ulaznih u izlazne. Mehanizam nenadziranog učenja ne zna izlazne podatke za pojedini skup ulaznih, već nastoji procijeniti osnovnu distribuciju ulaznih podataka. CNN uči pridruživanje pojedinog prikaza odgovarajućoj kategoriji izdvajanjem reprezentacija značajki koje se potom unutar mreže koriste za predviđanje ispravne kategorije izlaznog prikaza.[55]

5.1. Slojevi mreže

Konvolucijska neuronska mreža sadrži nekoliko građevnih elemenata koji se nazivaju konvolucijski slojevi. Na ulazu CNN nalazi se monokromatska slika ili slika u boji (Slika 5.1.), nakon čega se izmjenjuju konvolucijski slojevi i slojevi sažimanja. Zatim slijedi nekoliko potpuno povezanih jednodimenzionalnih slojeva (klasični perceptron) te izlazni sloj.[56]

[53] Božić-Štulić, Semantička segmentacija slika metodama dubokog učenja, 2017, 12.
[54] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 43.
[55] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 43.
[56] Božić-Štulić, Semantička segmentacija slika metodama dubokog učenja, 2017, 12.

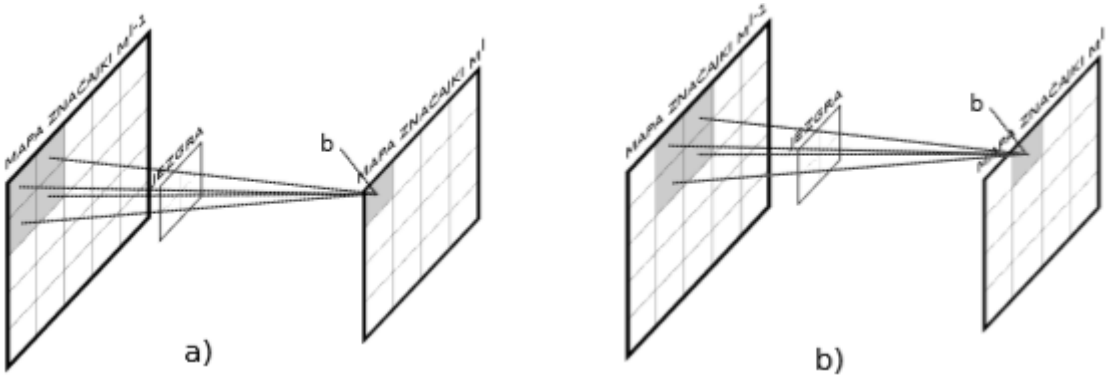


Slika 5.1. Struktura konvolucijskih neuronskih mreža[57]

Prosječna konvolucijska neuronska mreža ima desetak slojeva. Konvolucijski slojevi i slojevi sažimanja imaju dvodimenzionalne „neurone“ koje nazivamo „mapama značajki“. One sa svakim slojem poprimaju manje dimenzije. Posljednji takav sloj, dimenzija 1x1, predstavlja vezu na perceptron u zadnjim slojevima konvolucijske neuronske mreže. Dok su konvolucijski slojevi uvijek u jedan na više vezi s prethodnim slojem i uče težine u jezgrama s kojima vrše konvoluciju, slojevi sažimanja su s prethodnim slojem u jedan na jedan vezi te ne uče nikakve vrijednosti.[58]

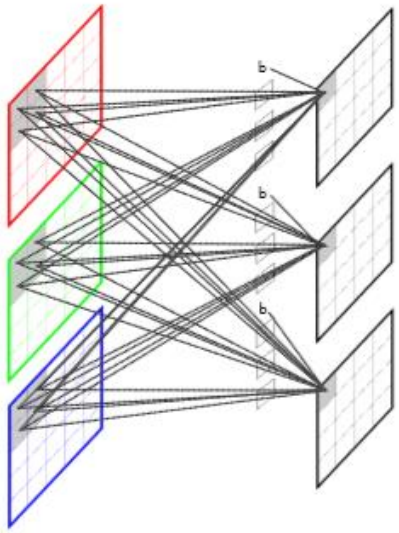
Mape značajki nastaju tako da konvolucijski slojevi uzimaju mapu na ulazu sloja te izvode dvodimenzionalnu konvoluciju s jezgrom. Konvolucija nastaje prolazom kroz ulaznu mapu prozorom veličine istim kao i jezgra, umnoškom ulaza i pripadajućih vrijednosti jezgre, zbrajanjem dobivenih vrijednosti s pragom, izračunom aktivacijske funkcije i u konačnici pohranom u pripadnu lokaciju izlazne mape. Prozor se potom pomiče pa se postupak ponavlja za sljedeći neuron u izlaznoj mapi. Svaka veza sloja s prethodnim predstavljena je jezgrom. Prema tome, svaka mapa mora imati onoliko jezgri koliko je mapa u prethodnom sloju na koji je spojena.[59]

[57] Božić-Štulić, Semantička segmentacija slika metodama dubokog učenja, 2017, 12.
[58] Vukotić, Raspoznavanje objekta dubokim neuronskim mrežama, 2014, 7.
[59] Vukotić, Raspoznavanje objekta dubokim neuronskim mrežama, 2014, 8.



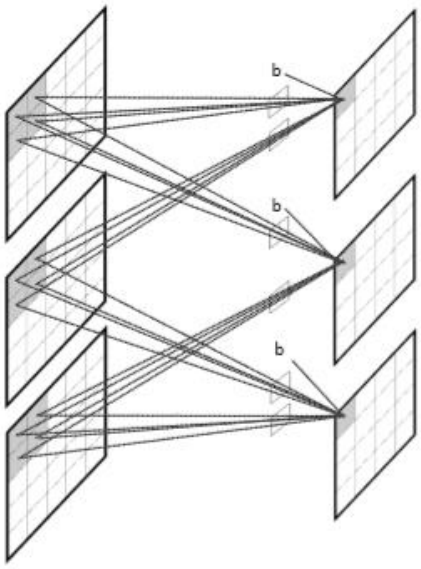
Slika 5.2. Konvolucija: a) prvi korak; b) korak nakon pomaka receptivnog polja[60]

Većina autora koristi potpuno povezane konvolucijske slojeve u kojima je svaka mapa značajki pojedinog sloja povezana sa svim mapama značajki prethodnog sloja (Slika 5.3.), no koristi se i raspršena povezanost u kojoj nisu sve mape značajki sloja povezane sa svim ostalim mapama značajki prethodnog sloja (Slika 5.4.).[61]



Slika 5.3. Prvi korak konvolucija kod potpuno povezanih slojeva[62]

[60] Vukotić, Raspoznavanje objekta dubokim neuronskim mrežama, 2014, 8.
[61] Vukotić, Raspoznavanje objekta dubokim neuronskim mrežama, 2014, 9.
[62] Vukotić, Raspoznavanje objekta dubokim neuronskim mrežama, 2014, 9.



Slika 5.4. Primjer raspršene povezanosti[63]

Slojevi sažimanja služe smanjivanju dimenzija značajki i osjetljivost neuronske mreže na manje pomake značajki u uzorku. U slojevima sažimanja postoje okviri kojima se prelazi preko mape značajki. Ona se sažima tako da se okvir prezentira samo jednom vrijednošću. Tako će se okvir dimenzija 2x2 predstaviti jednom vrijednošću koja proizlazi iz četiri vrijednosti čime se mapa značajki smanjuje četiri puta.[64] Raspoložive su dvije opcije sažimanja, sažimanje usrednjavanjem i sažimanje maksimalnom vrijednošću. Prva uzima aritmetičku sredinu vrijednosti koje se nalaze unutar okvira sažimanja, a druga uzima maksimalnu vrijednost.[65]

5.2. Funkcije gubitka

Posljednji sloj konvolucijske neuronske mreže koristi se isključivo tijekom procesa učenja kada koristi funkciju gubitka radi procjene kvalitete predviđanja same mreže. Funkcija gubitka kvantificira razliku između procijenjenog i ispravnog izlaza modela. Tip funkcije gubitka koja se koristi u CNN modelu ovisit će o krajnjem problemu, dolazi li iz kategorije binarne klasifikacije, verifikacije identiteta, višeklasne klasifikacije ili regresije.[66]

[63] Vukotić, Raspoznavanje objekta dubokim neuronskim mrežama, 2014, 10.
[64] Božić-Štulić, Semantička segmentacija slika metodama dubokog učenja, 2017, 13.
[65] Božić-Štulić, Semantička segmentacija slika metodama dubokog učenja, 2017, 13.
[66] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 65.

„Gubitak unakrsne entropije“ (engl. *cross-entropy loss*) definira se kao:[67]

$$L(p, y) = -\sum_{n} y_n \, log(p_n), \qquad n \in [1, N] \tag{11}$$

gdje y predstavlja željeni izlaz, a p vjerojatnost za svaku kategoriju izlaza. Budući da je N broj neurona u izlaznom sloju, slijedi $p, y \in \mathbb{R}^N$. Vjerojatnost svakog razreda može se izračunati korištenjem *Softmax* funkcije:

$$p_n = \frac{\exp(\hat{p}_n)}{\sum_k \exp(\hat{p}_k)} \tag{12}$$

Pritom je $\hat{p}_n$ izlazni rezultat prethodnog sloja koji nije normaliziran.[68]

SVM hinge loss motiviran je funkcijom pogreške koja se obično koristi tijekom učenja SVM klasifikatora. *Hinge loss* maksimizira margine između pozitivnih i negativnih uzoraka razreda, a definira se kao:[69]

$$L(p, y) = \sum_{n} \max(0, m - (2y_n - 1)p_n) \tag{13}$$

m je margina koja se postavlja kao stalna vrijednost 1. Kvadrirani *hinge loss* ipak se pokazao osjetljivijim spram narušavanja margine.

Euklidski gubitak označuje se kao kvadrirana greška između predviđanja i „istinitih oznaka“ (engl. *ground truth labels*):[70]

$$L(p, y) = \frac{1}{2N} \sum_{n} (p_n - y_n)^2 , \qquad n \in [1, N] \tag{14}$$

l^1 gubitak pokazao je, međutim, bolje rezultate kod regresijskih problema od euklidskog, a definira se kao:[71]

$$L(p, y) = \frac{1}{N} \sum_{n} |p_n - \, y_n| , \qquad n \in [1, N] \tag{15}$$

---

[67] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 66.
[68] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 66.
[69] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 66.
[70] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 67.
[71] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 67.

Kontrastni gubitak koristi se radi preslikavanja sličnih ulaza u bliske točke, a različitih u udaljene točke izlaznog prostora. Funkcionira samo na parovima sličnih ili različitih ulaza. Ovako se definira:[72]

$$L(p,y) = \frac{1}{2N}\sum_{n} yd^2 + (1-y)\max(0, m-d)^2\,, \qquad n \in [1,N] \tag{16}$$

„Gubitak očekivanja“ (engl. *expectation loss*):[73]

$$L(p,y) = \sum_{n}\left|y_n - \frac{\exp(p_n)}{\sum_k \exp(p_k)}\right|, \qquad n \in [1,N] \tag{17}$$

minimizira očekivanu vjerojatnost krive klasifikacije, no rijetko se koristi u dubokim neuronskim mrežama jer nije niti konveksna niti konkavna funkcija u odnosu na težine prethodnog sloja.[74]

5.3. Inicijalizacija težinskih vrijednosti

Ispravna inicijalizacija težinskih vrijednosti ključna je kod učenja vrlo dubokih neuronskih mreža te bi manjkava inicijalizacija mogla rezultirati problemom „nestajućeg“ ili pak gradijenta koji „eksplodira“. Postoji nekoliko različitih pristupa inicijalizaciji, a svaki sadrži određene komparativne prednosti i mane.

Gaussova nasumična inicijalizacija poprilično je čest pristup. Njime se konvolucijski i potpuno povezani slojevi inicijaliziraju korištenjem nasumičnih matrica čiji su elementi uzrokovani iz Gaussove distribucije s očekivanom vrijednošću nula i malom standardnom devijacijom.[75]

Za razliku od Gaussova nasumične inicijalizacije, uniformna nasumična inicijalizacija pokreće konvolucijske i potpuno povezane slojeve primjenom nasumičnih matrica čiji su elementi semplirani iz uniformne, a ne normalne distribucije. Normalne i uniformne nasumične inicijalizacije obično su jednako učinkovite, osim kod učenja vrlo dubokih neuronskih mreža gdje postoje određene poteškoće.[76]

---

[72] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 67.
[73] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 68.
[74] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 68.
[75] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 69.
[76] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 70.

Dobre rezultate u dubokim neuronskim mrežama pokazuje i pravokutna nasumična inicijalizacija za koju se raščlanjuje slučajna težina matrice, a potom se pravokutna matrica koristi za inicijalizaciju težinskih vrijednosti konvolucijskih neuronskih slojeva.[77] Kako bi se izbjegli problemi s nestajanjem i eksplozijom gradijenata, pristupa se nenadziranom pred-učenju slojeva koji može biti popraćen fazom nadziranog podešavanja. Zbog velike količine vremena koju ovaj pristup zahtijeva, ali i dostupnosti novih parametara i boljih tehnika inicijalizacije, pred-učenje slojeva rijetko se koristi za učenje vrlo dubokih konvolucijskih mreža.[78]

Kako bi se ublažio problem proporcionalnosti varijance izlaza i ulaznih veza, pristupa se inicijalizaciji težinskih vrijednosti varijancom ovisnom o broju ulaznih i izlaznih veza neurona gdje su *w* težinske vrijednosti mreže:[79]

$$Var(\omega) = \frac{2}{n_{f-in} + n_{f-out}} \tag{18}$$

Ovdje je riječ o „Xavier inicijalizaciji" koja pokazuje dobre rezultate u praksi te vodi boljim stopama konvergencije. Međutim, budući da se bazira na nizu pretpostavki poput linearne veze između ulaza i izlaza neurona, navedena inicijalizacija je statistički manje točna.[80] Zahvaljujući činjenici da neuroni ReLU nelinearnosti ne slijede pretpostavke Xavier inicijalizacije, ReLU *aware scaled* inicijalizacija, gdje je varijanca distribucije:

$$Var(\omega) = \frac{2}{n_{f-in}} \tag{19}$$

bolje funkcionira kod arhitekturi temeljenih na ReLU nelinearnosti.[81]

„Slojno sekvencijalna varijanca jedinice" (engl. *Layer-sequential unit variance LSUV*) je zapravo nastavak ortogonalne i normalizirane inicijalizacije težinskih vrijednosti u slojevima duboke neuronske mreže. Kombinira prednosti „normalizacije serije podataka" (engl. *batch normalization*) i ortogonalne i normalizirane inicijalizacije težinskih vrijednosti kako bi se postiglo učinkovito učenje vrlo dubokih mreža. Sastoji se od dva

[77] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 70.
[78] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 70.
[79] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 70.
[80] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 71.
[81] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 71.

koraka. U prvom, pravokutnoj inicijalizaciji, sve težinske vrijednosti konvolucijskih i potpuno povezanih slojeva inicijaliziraju se pravokutnom matricom. Zatim slijedi drugi korak, normalizacija varijanci, kada je varijanca svakog sloja normalizirana na jedan.

U slučaju kada nemamo mnogo raspoloživih označenih podataka, neuronske mreže je preporučljivo učiti na srodnom, ali drugačijem problemu za koji postoji više dostupnih podataka. Model se potom može adaptirati novom zadatku inicijalizacijom težinskih vrijednosti već naučenih na većoj bazi podataka. Opisani postupak, poznat kao „podešavanje“, predstavlja jednostavan i učinkovit način prijenosa učenja s jednog zadatka na drugi.[82]

5.4. Regularizacijske metode

Postoji mogućnost da model dubokih neuronskih mreža pokazuje daleko bolje rezultate na podacima tijekom procesa učenja nego s novim podacima. Kako bi se zaobišao navedeni problem, primijenjuju se takozvane regularizacijske metode koje se mogu kategorizirati u nekoliko razreda. Postoje pristupi koji:

- reguliraju mrežu korištenjem tehnika razine podataka;
- unose stohastičko ponašanje u neuronske aktivacije;
- normaliziraju statistiku skupa u aktivaciji značajki;
- koriste “spajanje sličnih ili različitih elemenata u zajednicu” (engl. *decision level fusion)* radi izbjegavanja prekomjerne specijalizacije modela;
- unose ograničenja težinskih vrijednosti mreže;
- se vode validacijskim setom radi zaustavljanja procesa učenja.[83]

Najlakši te iznimno učinkovit način poboljšanja mogućnosti generalizacije modela konvolucijskih neuronskih mreža jest augmentacija skupa za učenje. Navedena metoda se posebice preporuča u slučajevima gdje postoji malo primjera za učenje pa se na ovaj način povećava baza podataka koja omogućava otpornije učenje modela većih razmjera. Izvodi se izradom nekoliko kopija pojedinog prikaza primjenom rotacije, preklapanja, skaliranja i translacije. Ostali pristupi povećanju podataka uključuju nijansiranje boje i uporabu kombinacije sintetičkih i pravih podataka.[84]

---

[82] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 71-72.
[83] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 73.
[84] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 74.

Još jedna popularna regularizacijska metoda je tehnika „slučajnog izostavljanja neurona“ (*engl. Dropout*). Tijekom učenja mreže svaki se neuron aktivira fiksiranom vjerojatnošću što cijelu mrežu uključuje u stvaranje predviđanja te samim time poprilično poboljšava izvedbu na neviđenim podacima u testnoj fazi.

Slično funkcionira „drop-connect“ tehnika u kojoj se nasumično deaktiviraju težinske vrijednosti mreže, odnosno veze između neurona, umjesto da se aktivacije neurona nasumično svedu na nulu.[85]

„Normalizacija serije podataka“ (engl. *batch normalization*) predstavlja metodu koja se pokazala vrlo uspješnom u učenju duboke mreže. Ona smanjuje pomak unutarnje kovarijance aktivacija slojeva koja se odnosi na promjenu distribucije aktivacija svakog sloja kako se parametri ažuriraju tijekom učenja. Ako se distribucija koju skriveni sloj konvolucijske neuronske mreže modelira nastavlja mijenjati, proces učenja će se usporiti, a vrijeme konvergencije povećati. Normalizacija distribucije omogućuje konzistentnost distribucije aktivacija prilikom učenja što povećava konvergenciju, a smanjuje potencijal nestabilnosti mreže. Normalizacija serije podataka obično se primijenjuje neposredno prije nelinearne aktivacijske funkcije i podrazumijeva niz koristi poput poboljšanja stope konvergencije ili smanjene osjetljivosti učenja mreže na odabir hiper parametara i loših inicijalizacija težinskih vrijednosti. Osim toga, model mreže postaje manje ovisan o drugim regularizacijskim metodama poput tehnika slučajnog izostavljanja neurona.[86]

„Usrednjavanje ansambla modela“ (engl. *ensemble model averaging*) je jednostavna i učinkovita regularizacijska metoda u sklopu koje se uči više modela različitih parametara. Njihovi se izlazi potom kombiniraju i oni generiraju krajnji rezultat predviđanja. Finalno predviđanje kao usrednjavanje izlaza ili težinski prosjek svih predviđanja je preciznije od bilo kojeg modela u ansamblu.[87]

Regulacija l^2 penalizira velike vrijednosti parametara prilikom učenja što se postiže dodavanjem l^2 norme vrijednosti parametra otežane hiper parametrom koji odlučuje o snazi penalizacije. Na sličan način funkcionira l^1 regularizacijska metoda, uz razliku što se ovdje koristi l^1 umjesto l^2 norme.

[85] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 75.
[86] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 76-77.
[87] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 77.

Metode l^2 i l^1 kombinira „metoda elastične mreže“ koja dodaje $\lambda_1|w| + \lambda_2 w^2$, gdje je w parametar, a Λ hiper parametar. Raspršenost težinskih vrijednosti i općenito bolji rezultati nego pojedinačno korištenje l^2 i l^1 metoda glavna su obilježja regularizacijske metode elastične mreže.[88]

„Ograničenje maksimalne norme“ je metoda regularizacije koja definira gornju granicu norme ulaznih vrijednosti svakog neurona u sloju neuronske mreže. Prema tome, vektor težinske vrijednosti w mora pratiti ograničenje $\|w\|_2 < h$, gdje je hiper parametar čija se vrijednost bazira na izvedbi mreže tijekom provjere. Glavna prednost ove metode leži u garanciji parametara razumnog numeričkog raspona koja rezultira većom stabilnošću.[89]

„Rano zaustavljanje“ je regularizacijska metoda koja se koristi radi izbjegavanja situacije kada model pokazuje dobre rezultate prilikom učenja, no loše tijekom obrade neviđenih rezultata kod algoritama iterativnih gradijenata. Ovo se postiže evaluacijom izvedbe nad setom za provjeru tijekom različitih ponavljanja. Algoritam učenja može se poboljšavati nad setom za učenje sve dok se izvedba nad setom za provjeru također poboljšava. Onog trenutka kad mogućnost generalizacije naučenog modela krene opadati, proces učenja se može usporiti ili zaustaviti.[90]

5.5. Optimizacijske metode

U središtu svake neuronske mreže je optimizacija ili učenje. Postoji više potencijalnih problema koje je nužno zaobići tijekom procesa učenja poput nestajućeg i eksplozivnog gradijenta ili zaustavljanja u lokalnom minimumu. Sukladno tome, razvijeno je nekoliko metoda kojima se navedeni problemi rješavaju.[91]

„Momentum optimizacija“ osigurava poboljšanu verziju stohastičkog gradijenta pada (SGD) s boljim svojstvima konvergencije. Momentum predstavlja faktor koji ubrzava gradijent ako ide u istom smjeru ili usporava ako ide u različitim smjerovima.[92] „Nesterov momentum“ pak računa gradijent kod sljedeće aproksimativne, a ne trenutne točke tijekom

[88] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 78.
[89] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 79.
[90] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 79.
[91] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 81.
[92] Karačić, Korištenje web tehnologija za izradu i prikaz višeslojnih neuronskih mreža, 2016,18.

ažuriranja parametara, što algoritmu daje mogućnost gledanja unaprijed kod svake iteracije i time planiranja skoka. Tako se izbjegavaju izazovniji koraci prilikom procesa učenja.[93]

U slučajevima kada je potrebno svaki parametar ažurirati sukladno njegovoj frekvenciji u skupu za učenje, koristi se AdaGrad algoritam koji prilagođava stopu učenja za svaki pojedinačni parametar u svakoj iteraciji. AdaGrad algoritam posebno dobro funkcionira kod „raspršenih" (engl. *sparse*) gradijenata.[94]

Iako AdaGrad uklanja potrebu manualnog podešavanja stope učenja za različite epohe, sadrži problem (pre)niskih stopa učenja pa se parametri neće mijenjati u kasnijim iteracijama. Osim toga, zahtijeva postavljanje početne stope učenja. Spomenute manjkavosti AdaGrad algoritma rješava AdaDelta algoritam.[95] Na sličnom principu kao AdaDelta radi RMSprop algoritam kojim se također nastoji riješiti problematika niskih stopa učenja kod AdaGrad algoritma.[96]

„ADAM" (engl. *ADAptive Moment Estimation*) procjenjuje stopu učenja za svaki parametar te kombinira prednosti AdaGrad i RMSprop algoritama. Procjena ažuriranja na temelju prvog i drugog momenta gradijenta predstavlja glavnu razliku između ADAM-a i navedenih algoritama. Budući da ADAM dobro funkcionira kod problema većih razmjera te pokazuje dobra svojstva konvergencije, predstavlja uobičajeni izbor za mnoge aplikacije računalnog vida temeljene na dubokom učenju.[97]

5.6. Vizualizacija

Kako bi se vizualiziralo što je konvolucijska neuronska mreža naučila tijekom i po završetku procesa učenja, primijenjuje se više pristupa vizualizaciji značajki i aktivacija. Prema kriterijima težinskih vrijednosti, aktivacija i gradijenata razlikujemo nekoliko tipova metoda vizualizacije.

Jedan od najjednostavnijih pristupa vizualizaciji onog što je konvolucijska neuronska mreža naučila jest provjera konvolucijskih filtera koji ilustriraju tip obrasca što ga pojedini konvolucijski sloj traži u ulaznim podacima.[98] Korisne informacije o kvaliteti naučenih

[93] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 82.
[94] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 84.
[95] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 84.
[96] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 85.
[97] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 85-87.
[98] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 92-93.

reprezentacija pružaju i aktivacije značajki prijelaznih CNN slojeva. Vizualizacija aktivacija izlaza pruža uvid u obrasce izlaznih prikaza koji su izvučeni kao korisne značajke klasifikacije.[99]

Primjenjuje se i pristup dobivanja reprezentacije značajki pretposljednjeg sloja naučene konvolucijske mreže te vizualizacije svih prikaza za učenje u bazi podataka u obliku 2D dijagrama. Ova vizualizacija pruža cjeloviti pregled i sugerira kvalitetu naučenih reprezentacija značajki za različite razrede. Pored vizualizacije dvodimenzionalnog umetanja višedimenzionalnih vektora značajki, mogu se vizualizirati ulazni slikovni prikazi povezani sa svakim vektorom značajke.[100]

Mapa pozornosti može pak pružiti uvid u područja kojima je pridan veći značaj prilikom donošenja odluke o klasifikaciji. Dakle, ona može vizualizirati područja prikaza koja su najviše pridonijela ispravnoj kategorizaciji. Ovo se može realizirati dobivanjem rezultata predviđanja za pojedinačne preklapajuće mrlje kako bi se potom generirala krajnja predikcija. Drugi pristup uključuje blokiranje kvadratne mrlje unutar ulaznog prikaza i stvaranje toplinske mape koja ukazuje na dijelove ulazne slike najodgovornije za ispravan izlazni odgovor mreže.[101]

Jedna od metoda vizualizacije zasniva se na pretpostavci kako vizualizacija gradijenata pruža korisne informacije o konvergenciji dubokih neuronskih mreža. Gradijenti se mogu podesiti tako da vizualiziraju obrasce koje je neuron naučio tražiti u ulaznim podacima. Prvi takav pristup zasnivao se na dekonvoluciji – inverziji reprezentacija značajki radi identifikacije povezanih obrazaca među ulaznim prikazima. Drugi autori uključuju stvaranje sintetičkih slikovnih prikaza odabirom neurona u sloju konvolucijske mreže, provođenje ulazne slike nasumičnih vrijednosti boja mrežom, izračunavanje korespondirajuće aktivacijske vrijednosti tog neurona i određivanje gradijenta te aktivacije. Gradijent u principu označuje kako se pikseli mogu mijenjati da bi se neuron maksimalno aktivirao. Koristeći tu informaciju, ulaz se uzastopno mijenja kako bi se dobio prikaz veće aktivacije neurona.[102]

Postoji i pristup povrata ulaznih slikovnih prikaza koji korespondiraju identičnim, višedimenzionalnim reprezentacijama značajke u pretposljednjem sloju konvolucijske

[99] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 93.
[100] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 94.
[101] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 94.
[102] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 97.

neuronske mreže. Predmeti na tim rekonstruiranim prikazima razlikuju se u veličini, poziciji i deformacijama, no osnovne karakteristike ostaju iste. To dokazuje da je mreža naučila značajne karakteristike ulaznih slikovnih prikaza.[103]

[103] Khan, Rahmani et al., A Guide to Convolutional Neural Networks for Computer Vision, 2018, 99.

6. Izrada vlastitog programskog rješenja

6.1. Programski jezik Python

Python je interpreterski, interaktivni, objektno orjentirani programski jezik kojeg je 1990. godine razvio Guido Van Rossum. Već do konca 1998. godine Python je imao bazu od 300 000 korisnika, a od 2000. su ga prihvatile ustanove kao MIT, NASA, IBM, Google, Yahoo i druge. Python ne donosi neke nove revolucionarne značajke u programiranju, već na optimalan način ujedinjuje sve najbolje ideje i načela rada drugih programskih jezika. On je jednostavan i snažan istodobno. Python je besplatan, *open source* softver s izuzetno dobrom potporom, literaturom i dokumentacijom.[104] Ima filozofiju dizajna koja naglašava čitljivost koda korištenjem „značajnog razmaka“ (engl. *significant whitespace*). Podržava višestruke paradigme programiranja, uključujući objektno orijentirane, imperativne, funkcionalne i proceduralne te ima veliku i sveobuhvatnu knjižnicu modula i dodataka. Pythonova „knjižnica“ (engl. *library*), koja uključuje standardnu instalaciju, uključuje preko 200 modula. To pokriva sve od funkcija operacijskog sustava do struktura podataka potrebnih za gradnju web-servera. Glavna Pythonova web stranica (www.python.org) nudi sažeti indeks mnogih Python projekata i različitih drugih knjižnica. Prevoditelji za Python dostupni su za mnoge operativne sustave. CPython, referentna implementacija Pythona, softver je otvorenog koda i ima razvojni model temeljen na zajednici, kao i gotovo sve Pythonove druge implementacije. Njima upravlja neprofitna zaklada Python Software Foundation. Python 3.0 izdan je 3. prosinca 2008. godine. To je bila velika revizija jezika koji nije u potpunosti kompatibilan s verzijom 2. Python nudi sve značajke očekivane u modernom programskom jeziku: objektu orijentirano programiranje s višestrukim nasljeđivanjem, dohvaćanje „izuzetaka ili iznimki“ (engl. *exception*), redefiniranje standardnih operatora, pretpostavljene argumente, „prostore imena“ (engl. *namespaces*), module i pakete. Pisan je u modularnoj C arhitekturi i zato se lako može proširivati novim značajkama ili „API-jima“. (engl. *application programming interface*). Velika prednost Pythona je i velika zajednica korisnika koja se svake godine udvostručuje.[105]

---

[104] Mario Essert, Digitalni udžbenik Python, 2007, 7
[105] https://en.wikipedia.org/wiki/Python_(programming_language), travanj. 2019.

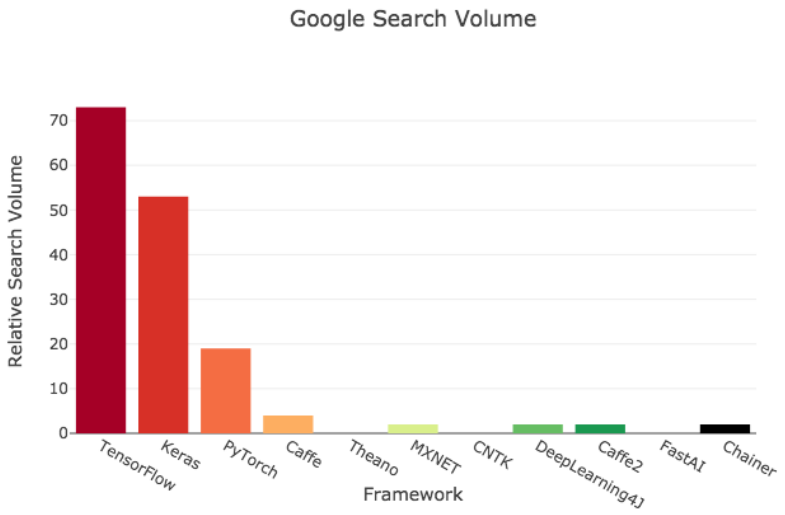
6.2. Keras i Tensorflow okvir

Keras je „okvir“ (engl. *framework*) dubokog učenja za Python koji pruža vrlo jednostavan način definiranja i treniranja gotovo bilo kojeg modela dubokog učenja. Keras početno razvijen za istraživače i znanstvenike s ciljem omogućavanja brzog eksperimentiranja. Ima sljedeće ključne značajke:

- omogućuje besprijekorno pokretanje istog koda na CPU-u ili GPU-u
- on je korisničko prilagođen API koji olakšava brzo prototipiranje modela dubokog učenja
- ima ugrađenu podršku za konvolucijske mreže (za računalni vid), ponavljajuće mreže (za obradu sekvenci) i bilo koju kombinaciju oba
- podržava proizvoljne mrežne arhitekture: *multi-input* ili *multi-output* modele, dijeljenje sloja, dijeljenje modela itd.

To znači da je Keras prikladan za izgradnju bilo kojeg modela dubokog učenja. Distribuira se pod dozvolom MIT licence, što znači da može biti slobodno korišten u bilo kojem komercijalnom projektu. Kompatibilan je s bilo kojom verzijom Pythona od 2.7 do 3.6 (od sredine 2017.). Keras ima više od 200 000 korisnika, od akademskih istraživača i inženjera koji rade u manjim i velikim tvrtkama do postdiplomaca i hobista. Koristi se u kompanijama kao što su Google, Netflix, Uber, CERN, Yelp, Square i u stotinama ostalih razvojnih komapnija koje rade na širokom rasponu različitih problema. Također je popularan okvir na Kaggleu (web-mjesto za održavanje natjecanja u strojnom učenju), gdje je gotovo svako nedavno natjecanje u dubokom učenju osvojeno pomoću nekog modela izgrađenog pomoću Keras okvir.[106] Kao što je vidljivo na slici (Slika 6.1.), Keras je drugi po redu najtraženiji okvir za duboko učenje prema statistikama tražilice Google.

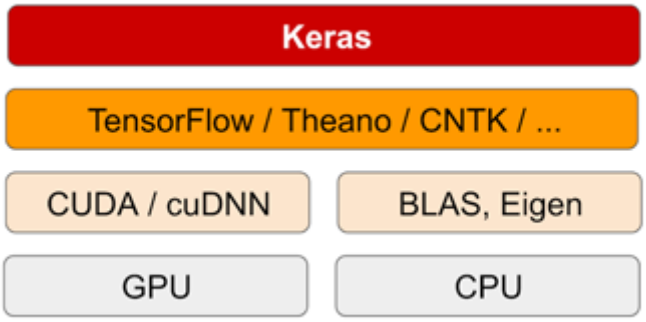
[106] François Chollet, Deep Learning with Python, 2018, 61



Slika 6.1. Prikaz pretraga po programskim okvirima

On je *model-level* biblioteka koja pruža visoku razinu izgradnje modela dubinskog učenja. Jedna od glavnih prednosti Keras okvira je specijalizirana te dobro optimizirana tenzor biblioteka koja omogućava modularni način funkcioniranja pozadinskog sustava. Ona omogućava promjenu različitih implementacija pozadinskog sustava bez ikakve promjene u samom kodu aplikacije koju pišemo. To znači da jednom kada napišemo kod za izradu modela, taj isti kod možemo pokrenuti s bilo kojim podržanim sustavom. Kao što je vidljivo na slici (Slika 6.2.), trenutno postoje tri postojeće pozadinske implementacije: TensorFlow, Theano i Microsoft Cognitive Toolkit (CNTK). TensorFlow (www.tensorflow.org) je razvio Google, Theano (http://deeplearning.net/software/theano) je razvio MILA laboratorij na Sveučilištu Montréal i CNTK (https://github.com/Microsoft/CNTK) je razvio Microsoft. U bilo kojem trenutku se možemo prebaciti između navedenih pozadinskih sustava ako je, primjerice, jedan od sustava brži za odrađivanje nekog specifičnog zadatka od drugog. Koristeći bilo koji od navedenih pozadinskih sustava, Keras je sposoban besprijekorno raditi koristeći CPU (engl. *Central processing unit*) ili GPU (engl. *Graphic processing unit*). Trenutno se preporuča korištenje TensorFlow pozadinskog sustava jer je to najšire prihvaćen, skalabilan i najstabilniji sustav spreman za produkciju.[107]

[107] François Chollet, Deep Learning with Python, 2018, 62



Slika 6.2. Prikaz slojeva nad kojima se nalazi Keras

TensorFlow je biblioteka otvorenog koda za numeričko računanje pomoću grafova protoka podataka. Čvorovi grafova predstavljaju matematičke operacije, dok rubovi grafova predstavljaju višedimenzionalne podatkovne nizove (tenzore) koji se kreću između njih. Ova fleksibilna arhitektura omogućuje implementaciju računanja na jednom ili više procesora ili grafičkih procesora na radnoj površini, poslužitelju ili mobilnom uređaju bez ponovnog pisanja koda. TensorFlow također uključuje TensorBoard, alat za vizualizaciju podataka.

TensorFlow su izvorno razvili istraživači i inženjeri koji rade na Google Brain timu unutar Googleove organizacije za istraživanje inteligencije stroja. Njena je svrha provođenje strojnog učenja i istraživanje dubokih neuronskih mreža. Sustav je dovoljno općenit da se može primijeniti i u raznim drugim domenama. TensorFlow pruža stabilne Python i C API-je, kao i nezajamčeno stabilne API-je za C ++, Go, Java, JavaScript i Swift.[108]

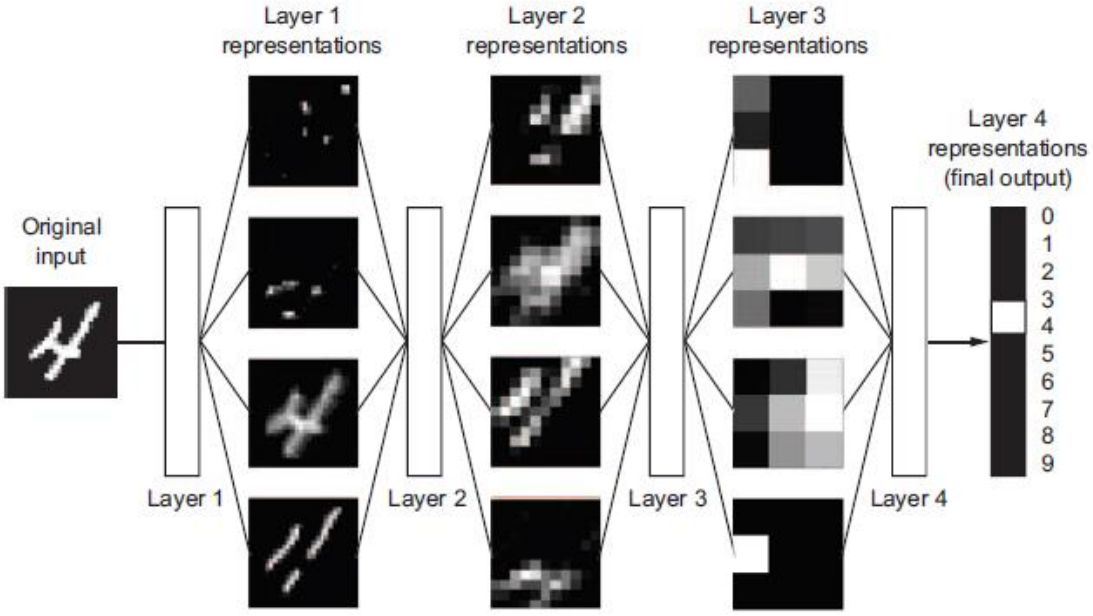
6.3. Izrada modela neuronske mreže

Problem koji pokušavamo riješiti je klasificirati rukom pisane brojeve prikazane slikom u sivim tonovima veličine 28×28 piksela u svojih 10 kategorija (0 do 9). Za skup podataka koristit ćemo MNIST skup podataka, svojevrsni klasik u zajednici za strojno učenje. MNIST je skup od 60 000 slika koje koristimo za treniranje, uz dodatnih 10 000 slika koje koristimo za testiranje, sastavljenih od strane Nacionalnog instituta za standarde i tehnologije 1980-ih. Prvo ćemo u mrežu pustiti trening slike zajedno s njihovim „oznakama“ (engl. *label*). Mreža će tada naučiti povezivati slike s oznakama.

[108] https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html, travanj. 2019.

Zatim ćemo od mreže zatražiti da proizvede predviđanja za skup testnih slika, a mi ćemo provjeriti odgovaraju li ta predviđanja odgovarajućim oznakama.

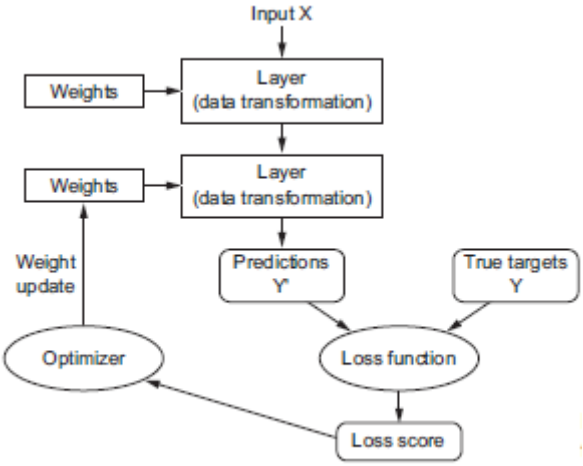
Jezgra neuralnih mreža je „sloj“ (engl. *layer*), modul za obradu podataka kojeg možemo zamisliti kao filter za podatke. Određeni skup podataka ulazi u sloj, biva filtriran prema zadanim pravilima te se naposljetku pojavljuje na izlazu u mnogo korisnijem obliku (Slika 6.3). Većina modela dubokog učenja sastoji se od ulančavanja jednostavnih slojeva koji će implementirati oblik progresivnog filtera podataka. Model dubokog učenja je poput sita za obradu podataka, napravljen od niza sve rafiniranijih filtera podataka - slojeva.



Slika 6.3. Prikaz slojeva unutar modela

Da bi mreža bila spremna za treniranje, moramo odabrati nekoliko parametara koji će nam uvelike pomoći u postizanju boljih rezultata. Prva stvar koja nam je potrebna je odabir „funkcije gubitka“ (engl. *loss function*). Funkcija gubitka služi za mjerenje izvođenja same mreže na testnim podacima te samostalne korekcije mreže u pravom smjeru. Da bismo nešto mogli kontrolirati, prvo moramo biti u mogućnosti to promatrati. Kako bismo mogli kontrolirati izlazne parametre mreže, prvo moramo biti u mogućnosti izmjeriti kolika je udaljenost između naših izlaznih parametara naspram onoga što smo očekivali. To je zadatak funkcije gubitka. U ovom slučaju koristio sam logaritamski gubitak, tj. *categorical crossentropy* funkciju unutar Keras okvira (Slika 6.4). Logaritamski gubitak mjeri učinkovitost klasifikacijskog modela gdje je vrijednost predviđanja vrijednost vjerojatnosti između 0 i 1. Cilj modela je minimizirati ovu vrijednost. Savršeni model imao bi logaritamski gubitak od 0. Logaritamski se gubitak povećava kako se predviđaju

odstupanja od stvarne oznake[109]. Funkcija uzima predikciju naše mreže i uspoređuje taj rezultat sa stvarnom vrijednošću koju očekujemo. Nakon toga izračunava udaljenost između rezultata te nam govori koliko je mreža dobro ili loše odradila posao na specifičnom primjeru. Dobiveni rezultat tada se koristi kao povratni signal za podešavanje težinskih vrijednosti koje koristimo kako bismo minimizirali funkciju gubitka. To podešavanje je posao optimizatora, još jednog ključnog parametra unutar naše mreže.



Slika 6.4. Prikaz funkcije gubitka

Optimizator je mehanizam pomoću kojeg će mreža ažurirati težinske vrijednosti. Svrha je smanjenje vrijednosti funkcije gubitka. Dokle god se vrijednost funkcije gubitka smanjuje, možemo reći da se mreža kreće u dobrom smjeru. Optimizator kojeg sam koristio je dio Keras okvira i zove se Adadelta. Adadelta prilagođava težinske vrijednosti bez da akumulira sve prethodne gradijente, što znači da nastavlja s učenjem čak i nakon višestrukih ažuriranja. Još jedan parametar kojeg bi trebalo spomenuti je sama preciznost mreže. Nju koristimo kako bismo u svakom prolasku znali koliko smo slika točno, a koliko krivo klasificirali.

Za izradu modela neuronske mreže koristio sam Keras okvir. Jedna od mnogih prednosti biblioteke je gotova metoda za učitavanje MNIST skupa podataka koji se nakon inicijalnog učitavanja lokalno pohranjuje na disk. Osim MNIST skupa podataka, biblioteka nudi mogućnost učitavanja još nekoliko poznatih skupova podataka koji se često koriste u kreiranju modela dubokog učenja (CIFAR10, CIFAR100, IMDB Movie reviews, Reuters newswire i drugi). Nakon učitavanja podataka, potrebno ih je obraditi i prilagoditi za

[109] http://wiki.fast.ai/index.php/Log_Loss, travanj. 2019.

treniranje i ubacivanje u mrežu. Kod korištenja Keras okvira moramo eksplicitno deklarirati dimenziju za dubinu ulazne slike. Primjerice, slika u punoj boji sa sva 3 RGB kanala imat će dubinu 3. Naše MNIST slike imaju dubinu 1, ali to moramo izričito naglasiti koristeći *reshape()* funkciju. Drugim riječima, želimo transformirati naš skup podataka iz jednog oblika (n, širina, visina) u drugi oblik (n, dubina, širina, visina) kojeg naš model razumije. Možemo i smanjiti naše memorijske zahtjeve prisiljavajući preciznost vrijednosti piksela na 32 bita, što je ionako zadana preciznost Keras okvira. Vrijednosti piksela u sivim tonovima su između 0 i 255. Gotovo je uvijek dobro napraviti skaliranje ulaznih vrijednosti kada se koriste modeli neuronske mreže. U našem slučaju normaliziramo vrijednosti piksela do raspona 0 i 1 dijeleći svaku vrijednost za maksimalno 255 (Slika 6.5.). Izlazna vrijednost mreže je cijeli broj od 0 do 9. To je klasifikacijski problem i u tom slučaju je dobro koristiti *one-hot* vektor kodiranja vrijednosti klasa, pretvarajući vektor u binarnu matricu. To možemo jednostavno učiniti pomoću ugrađene *np_utils.to_categorical()* pomoćne funkcije u Kerasu.

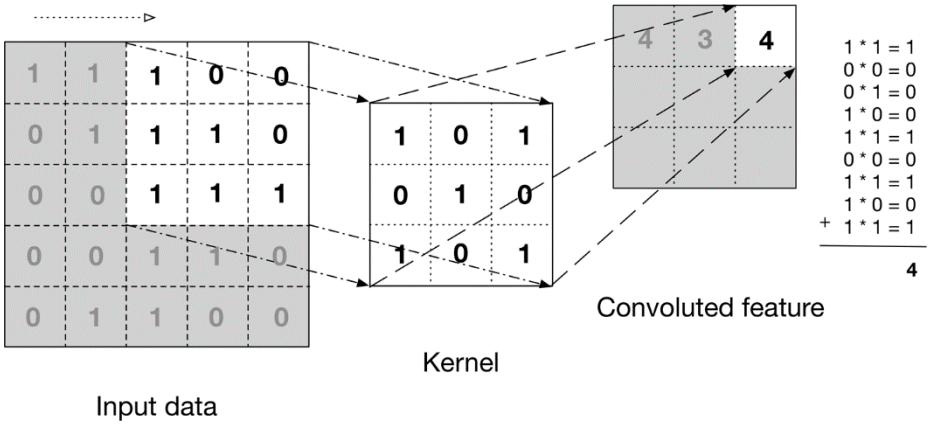
```
35  # Load MNIST dataset
36  (X_train, y_train), (X_test, y_test) = mnist.load_data()
37
38  # Reshape, convert and normalize
39  X_train = X_train.reshape(X_train.shape[0], 1, img_rows, img_cols)
40  X_test = X_test.reshape(X_test.shape[0], 1, img_rows, img_cols)
41
42  X_train = X_train.astype('float32')
43  X_test = X_test.astype('float32')
44
45  X_train /= 255
46  X_test /= 255
```

Slika 6.5. Prikaz učitavanja i normaliziranja MNIST seta podataka

Sljedeći zadatak je definirati model neuronske mreže. Za potrebu rada odabrao sam model konvolucijske neuronske mreže. Konvolucijska neuronska mreža sastoji se od nekoliko slojeva: „sloj konvolucije“ (engl. *convolution*), „sloj udruživanja“ (engl. *pooling*), „sloj izbacivanja“ (engl. *dropout*), „spljošteni sloj“ (engl. *flatten*), „potpuno povezani sloj“ (engl. *fully connected*) te „izlazni sloj“ (engl. *output*). Između slojeva koristimo aktivacijsku funkciju ReLU (engl. *rectified linear unit*) koja je najčešće korištena aktivacijska funkcija u neuronskim mrežama. Funkcija brzo konvergira, linearna je za sve pozitivne vrijednosti, a nula za sve negativne vrijednosti. To znači da izračuni ne opterećuju pretjerano resurse jer nema komplicirane matematike. Posljednji sloj je sloj aktivacije koji koristi *Softmax* funkciju. Ona nam omogućuje da izračunamo izlaz na

temelju vjerojatnosti. Zbroj svih vjerojatnosti je jednak 1, a klasa koja je obilježena najvećom vjerojatnošću je klasa za koju mreža smatra da je točna.

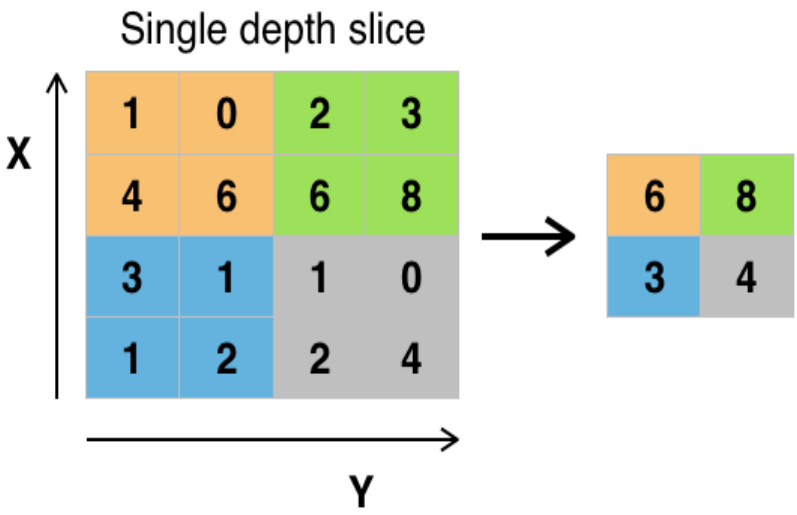
U sloju konvolucije izvodi se konvolucijska operacija. To je funkcija izvedena iz dviju zadanih funkcija integracijom koja izražava kako je oblik jedne modificiran drugom. U neuronskoj mreži izvršit ćemo rad konvolucije na matrici ulazne slike kako bismo smanjili njezinu veličinu i izvukli najbitnije značajke slike. To radimo tako da definiramo filter veličine 3x3 koji se pomiče po ulaznoj slici, piksel po piksel, dok ne prođe cijelu sliku. Svakim pomakom filtera dolazi do množenja elemenata između dvije matrice. Rezultat množenja formira jedan element u novoj izlaznoj matrici. Cijeli prikaz operacije vidljiv je na slici (Slika 6.6.).



Slika 6.6. Prikaz konvolucijske operacije[110]

Sljedeći sloj koji koristimo unutar mreže je sloj udruživanja (engl. *pooling*). Njega koristimo kako bismo smanjili veličinu matrice slike, ubrzali računanje te kako bismo određene značajke slike učinili robusnijima. Postoje dvije vrste sloja udruživanja: „maksimalno udruživanje“ (engl. *max pooling*) i „prosječno udruživanje“ (engl. *average pooling*). U ovom modelu mreže koristio sam sloj maksimalnog udruživanja. Prvo kreiramo filter (2x2) koji se pomiče po matrici slike i prilikom svakog pomaka vrati piksel s najvećim brojem. Princip rada je vidljiv na slici (Slika 6.7).

[110] Deep Learning, Adam Gibson, Josh Patterson, slika 4.12



Slika 6.7. Prikaz maksimalnog udruživanja

„Sloj izbacivanja“ (engl. *dropout*) je sloj koji se koristi tehnikama regularizacije kako bi se spriječila „prekomjerna specijalizacija modela“ (engl. *overfitting*). Prekomjerna specijalizacija modela odnosi se na slučajeve kada se pri modeliranju daje prevelika težina slučajnim varijacijama vrijednosti podataka. Takvi modeli imaju veliku prediktivnu točnost na skupu primjera za učenje, a značajno nižu na novim, nepoznatim primjerima podataka ili testnom skupu podataka. Kako bismo to izbjegli, kroz svaki se sloj mreže ignoriraju slučajno odabrani neuroni. To znači da je njihov doprinos privremeno uklonjen te se sva ažuriranja težinskih vrijednosti ne primijenjuju na njih. Mreža postaje manje osjetljiva na specifične težinske vrijednosti neurona što rezultira mrežom koja je sposobna bolje generalizirati i manje je vjerojatno da će doći do prekomjerne specijalizacije modela.

„Spljošteni sloj“ (engl. *flatten*) je sloj koji se koristi nakon sloja udruživanja. Njega koristimo kako bismo matricu pretvorili u vektor koji onda dalje šaljemo kao ulazni parametar u „potpuno povezani sloj“ (engl. *fully connected layer*).

Potpuno povezani sloj je zapravo već spomenuti višeslojni perceptron koji koristi funkciju aktivacije *Softmax* u izlaznom sloju. Izraz „potpuno povezani sloj“ podrazumijeva da je svaki neuron u prethodnom sloju povezan sa svakim neuronom u sljedećem. Njegova zadaća je klasificirati ulazne parametre, tj. značajke, koje je dobio od konvolucijskog i sloja udruživanja u razne klase. Osim klasifikacije, dodavanje potpuno spojenog sloja također je dobar način učenja nelinearnih kombinacija dobivenih značajki. Većina značajki iz konvolucijskog i sloja udruživanja može biti dobra za zadatak klasifikacije, ali kombinacije tih značajki mogu biti još bolje. Zbroj izlaznih vjerojatnosti iz potpuno

povezanog sloja je 1. To postižemo korištenjem *Softmaxa* kao aktivacijske funkcije u „izlaznom sloju“ (engl. *output*).

Model se u Keras okviru instancira putem *Sequential()* objekta. On predstavlja linearni stog različitih slojeva koje u model možemo dodati pozivajući funkciju *add()*. Model koji je napravljen prati LeNet arhitekturu i sastoji se od dva konvolucijska sloja s filterima veličine 3x3, slojem udruživanja sa filterom veličine 2x2, slojem izbacivanja, spljoštenim slojem, potpuno povezanim slojem i završava izlaznim slojem s aktivacijskom funkcijom *Softmax*. Kreiranje modela koristeći Keras okvir prikazano je na slici (Slika 6.8.).

```
# model
model = Sequential()
model.add(Convolution2D(32, 3, 3, border_mode='valid',input_shape = (1, img_rows, img_cols)))
model.add(Activation('relu'))

model.add(Convolution2D(32, 3, 3))
model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128))

model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

Slika 6.8. Prikaz modela

6.4. Treniranje mreže i pohranjivanje modela

Nakon postavljanja modela, potrebno ga je sastaviti, tj. kompilirati. Koristimo metodu *compile()* kojoj predajemo tri parametra: *loss, optimizer i metrics()*. Sva tri parametra objašnjena su u prethodnom poglavlju.

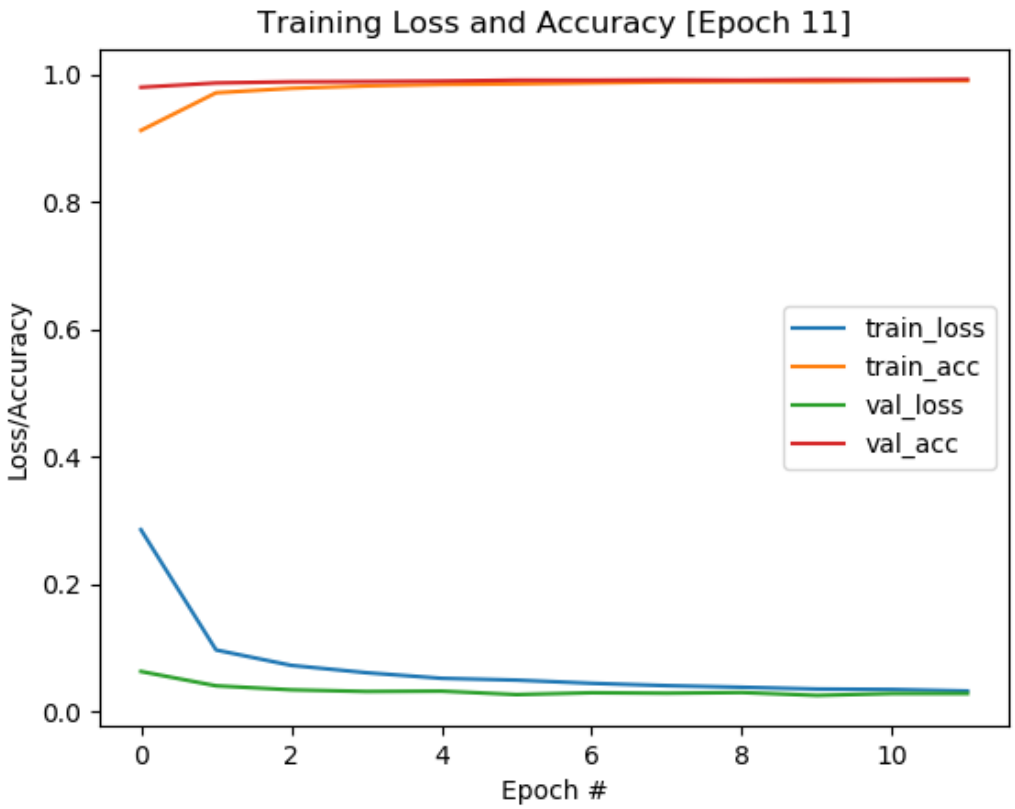
Zatim podešavamo model i počinjemo treniranje. Treniranje započinje pozivom metode *fit()* kojoj predajemo testne i trening slike, testne i trening *labele*, veličinu serije koja nam govori nakon koliko slika će se raditi ažuriranje parametara te broj epoha koji pokazuje koliko puta ćemo proći kroz sve trening slike. Za ovaj model broj epoha je postavljen na 12, s veličinom serije od 128. Na kraju svake epohe model se testira putem seta od 10 000 testnih slika te se ispisuje broj točnih pogodaka, kao i trenutna vrijednost funkcije gubitka i preciznost samog modela.

```
model.compile(loss='categorical_crossentropy', optimizer='adadelta', metrics=["accuracy"])

# Learn
history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

Slika 6.9. Podešavanje modela i početak treniranja

Metoda vraća *history* rječnik koji u sebi sadrži metrike potrebne za praćenje kretanja vrijednosti *loss* funkcija i preciznosti vezanih uz treniranje modela i validaciju nakon svake epohe. Dobivene podatke koristimo kako bismo iscrtali graf treninga i validacijskih vrijednosti (Slika 6.10.). Graf nam u slučaju treniranja može pomoći u praćenju kretanja vrijednosti funkcije gubitka te na taj način možemo lakše vizualno zaključiti je li došlo do „prekomjerne specijalizacije modela“ (engl. *overfitting*) koju želimo izbjeći.



Slika 6.10. Graf funkcije gubitka i preciznosti

Prilikom završetka treniranja postavljamo si pitanje: kako ovaj model radi s testnim podacima? Testne podatke smo izdvojili tako da ih model još nije vidio. Ako na njima dobijemo dobre rezultate mjerenja, dokazat ćemo da naš model dobro generalizira. Ovo je i bio inicijalni cilj. Kako bismo testirali model, koristimo funkciju *evaluate()* kojoj predajemo set testnih slika i njihovih oznaka. Povratni parametri funkcije vrijednosti su koje nam pokazuju preciznost i rezultat funkcije gubitka za naš model.

Naposljetku, nakon treniranja i testiranja pohranjujemo arhitekturu modela za kasniju uporabu. Cijelu arhitekturu modela pohranit ćemo u jednu datoteku formata HDF5. To je jedinstveni format koji omogućuje upravljanje izuzetno velikim i složenim zbirkama podataka. Naš pohranjeni model sadržavat će cjelokupnu arhitekturu, težinske vrijednosti i kompletnu trening konfiguraciju sa svim vrijednostima funkcija gubitaka. Pohranjivanje modela vrlo je jednostavno i odrađuje se pozivom metode *save()* kojoj se predaje putanja na koju želimo spremiti datoteku (Slika 6.11). Ako želimo koristiti pohranjeni model, pozivamo funkciju *load_model()* kojoj predajemo putanju do njega. Nakon učitavanja, model će biti rekreiran i spreman za rad.

```
#Evalute model
score = model.evaluate(X_test, Y_test, verbose=0)

print('Test score:', score[0])
print('Test accuracy:', score[1])

print('Saving the model')
#Saving the model
save_model_name = "model.h5"
save_model_path = os.getcwd() + "/save/"

model.save(os.path.join(save_model_path,save_model_name), include_optimizer = True)

print('Model is saved and ready for testing.')
```

Slika 6.11 Evaluacija i pohranjivanje modela

6.5. Testiranje modela

Kako bih se uvjerio da je model ispravno spremljen i da je treniranje modela zaista uspjelo, kreirana je dodatna Python datoteka. Unutar nje će se učitati i inicijalizirati model s diska i predat će mu se dvije slučajno odabrane slike na evaluaciju. Model se učitava s diska već spomenutom metodom *load_model()*. Nakon učitavanja i inicijalizacije modela, potrebno je učitati testne slike i pripremiti ih za slanje u mrežu. Za učitavanje slika koristimo PILLOW biblioteku te metode *open()* i *convert()* kojima učitavamo sliku i radimo konverziju slike u nijansama sive. Obrađenu sliku prebacujemo u *tip float32* i normaliziramo u rang od 0 do 1. Model očekuje takav tip podataka jer je na takvom tipu podataka bio i treniran. Proces učitavanja i obrade vidljiv je na slici (Slika 6.12.)

```
23
24  #Load a test image from disk and prepare it for prediction
25  img_pred = PIL.Image.open("osam.jpg").convert("L")
26  img_array_pred = np.array(img_pred).reshape(1,1,28,28)
27  img_array_pred = img_array_pred.astype('float32')
28  img_array_pred /= 255
29
```

Slika 6.12. Obrada učitane slike

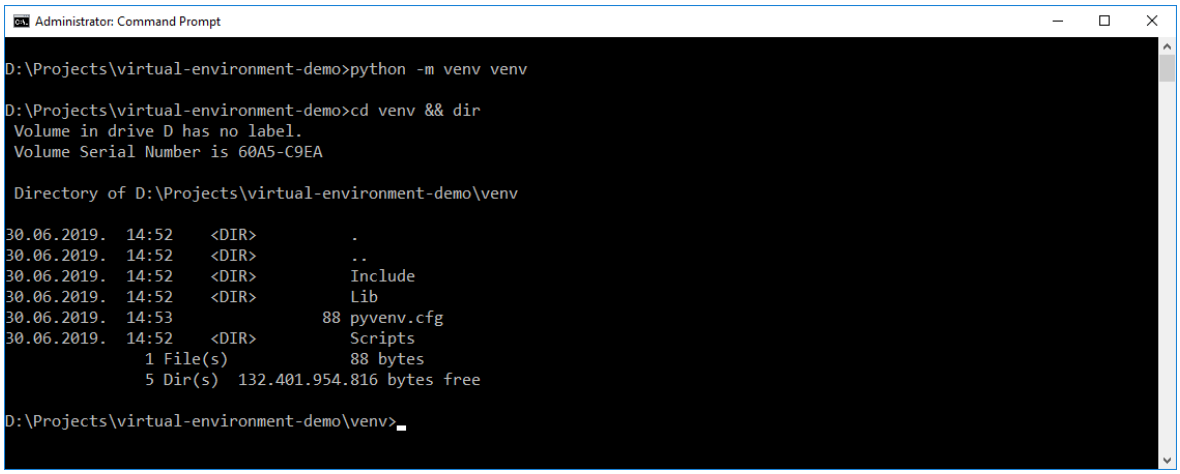
Nakon obrade, slika se predaje u *predict()* funkciju koja vraća listu rezultata predikcije. Rezultat predikcije je ključ-vrijednost rječnik gdje ključ predstavlja broj od 0 do 9. U našem slučaju on označava slike brojeva od 0 do 9 te vrijednost koja označava vjerojatnost mreže o kojem broju se radi. Uz dobivene rezultate predikcije, iz modela možemo dobiti i izgled slike unutar pojedinih slojeva. To nam omogućava praćenje promjena koje se događaju na slici poslije svakog sloja unutar modela. Koristeći metodu *get_layer()* unutar Keras okvira možemo dohvatiti sloj koji želimo promatrati. Željenom sloju možemo izvući izlazne parametre te ih nakon obrade prikazati. Isti princip vrijedi i za prikaz težinskih vrijednosti konvolucijskih slojeva modela.

7. Izrada web servisa

7.1. Python virtualna okolina

Za izradu web servisa korišten je programski jezik Python. Jedna od glavnih značajki Python programskog jezika je mnoštvo modula i paketa koje možemo odabrati kako bi nam rad s određenim problemima bio što lakši. Python moduli i paketi instaliraju se globalno na naše računalo, što bi u prijevodu značilo da može postojati samo jedna verzija nekog modula ili paketa za sve naše programe. Vrlo brzo ćemo naletjeti na konflikte s verzijama određenih modula ili paketa budući da ne moraju svi programi uvijek koristiti istu verziju. Problemu s različitim verzijama možemo doskočiti izolirajući Python okoline u tzv. „virtualne okoline“. Virtualna okolina nam omogućava da odvojimo Python zavisnosti po projektima i na taj način održavamo različite verzije modula i paketa za svaki projekt zasebno. Virtualna okolina je izolirana i živi u mapi projekta te sadržava sve potrebne module i pakete koje projekt treba za rad. Ovako možemo unaprijediti bilo koji modul ili paket bez straha da će promjena utjecati na ostale programe. Postavljanje virtualne okoline vrlo je jednostavno i može se odraditi u nekoliko koraka. Postoji nekoliko preduvjeta koje moramo ostvariti kako bi virtualna okolina radila. Osim samog Pythona, potrebno je instalirati i „pip“ (engl. *Python Package Installer*). To je glavni alat za instalaciju Python paketa i modula. On je sam po sebi uključen u novije verzije Python jezika, tako da ćemo instalacijom jezika dobiti instaliran i *pip*. Kako bismo postavili virtualnu okolinu, prvo se trebamo pozicionirati na željenu lokaciju te nakon toga kreirati mapu novog projekta. U ovom slučaju koristit ćemo testni projekt *virtual-enviroment-demo*.

Sljedeći korak je kreiranje virtualne okoline koristeći naredbu *>python -m venv venv*. Ovim ćemo kreirati virtualnu okolinu sa svim potrebnim modulima i paketima za rad. Modul kojeg Python koristi kako bi kreirao virtualnu okolinu zove se *venv*, a druga oznaka *venv* je generalna konvencija za naziv direktorija u kojeg će se instalirati sve što je potrebno za rad. Opisani koraci vidljivi su na slici (Slika 7.1.).



Slika 7.1. Kreiranje virtualnog okruženja

Zadnji korak je pokretanje aktivacijske skripte koja će „podignuti“ virtualnu okolinu. To radimo aktiviranjem skripte *activate*. Znat ćemo da je okolina podignuta kada se u komandnoj liniji prikaže ime direktorija u kojemu je okolina instalirana. U našem slučaju radi se o *(venv)* direktoriju. Sve što od sada radimo i instaliramo događa se u virtualnoj okolini i ne utječe na globalne module i pakete. To nam omogućuje „spuštanje“ ili „podizanje“ verzija bilo kojih paketa ili modula bez straha od narušavanja njihovih globalnih funkcionalnosti. Nakon podizanja virtualne okoline možemo krenuti u instalaciju svih potrebnih okvira, modula i paketa kao što su Flask, Keras, Tensorflow i drugi.[111]

7.2. Flask okvir

Flask je mikro web okvir pisan u Pythonu. „Mikro“ znači da Flask cilja na zadržavanje jezgre jednostavnom, ali jako proširivom. Klasificiran je kao „mikrokvir“ jer ne zahtijeva posebne alate ili knjižnice. Nema sloj apstrakcije baze podataka, provjeru valjanosti obrazaca ili bilo koje druge komponente gdje postojeće biblioteke trećih strana pružaju uobičajene funkcije. Međutim, Flask podržava proširenja koja mogu dodati značajke aplikacije kao da su implementirane u njemu samom. Postoje proširenja za objektno-relacijska mapiranja, provjeru valjanosti, rukovanje učitavanjem, razne otvorene tehnologije provjere autentičnosti i drugo. Flask se obično koristi s MongoDB bazom podataka. Programi koji koriste okvir Flaska uključuju Pinterest, LinkedIn i web stranicu zajednice za sam Flask. Stvorio ga je Armin Ronacher iz Pocooa, međunarodne skupine entuzijasta Pythona formiranih 2004. godine.

[111] https://realpython.com/python-virtual-environments-a-primer/, travanj. 2019.

Značajke Flaska:

- sadrži razvojni poslužitelj i program za ispravljanje pogrešaka
- integrirana podrška za testiranje jedinica
- RESTful zahtjev za otpremu
- koristi Jinja2 *templating*
- podrška za sigurne kolačiće (sesije na strani klijenta)
- 100% kompatibilan sa WSGI 1.0
- baziran na Unicode standardu
- Opsežna dokumentacija
- Google App Engine kompatibilnost
- dostupna proširenja za poboljšanje željenih značajki

7.3. Inicijalizacija modela prilikom pokretanja web servisa

Kako bi servis mogao brzo vraćati podatke o predikciji i slike iz različitih slojeva, potrebno je učitati model s diska i inicijalizirati ga samo jednom. To se događa prilikom inicijalnog pokretanja servisa kako bi se izbjeglo konstantno učitavanje modela za svaki „zahtjev“ (engl. *request*) koji dolazi. Tako dobijemo na brzini jer jednom kad je učitani model spreman za rad, vrijeme odaziva je puno brže za razliku od načina rada prilikom kojeg bi se skupa operacija učitavanja ponavljala stalno za svaki zahtjev. Tijekom pisanja ovog rada pojavio se problem s Tensorflow i Keras okvirom unutar Flask aplikacije zbog načina na koji Flask odrađuje dolazne zahtjeve. Za svaki novi zahtjev Flask kreira novu „dretvu“ (engl. *thread*) te se u tom trenutku kreira i nova Tensorflow „sesija“ (engl. *session*) koja je drugačija od sesije koja je kreirana prilikom inicijalnog učitavanja modela. Problem je riješen tako da je u globalnu varijablu spremljena inicijalno kreirana sesija koja se kasnije koristi prilikom poziva bilo koje metode na modelu.

7.4. Priprema ulaznih podataka

Nakon pokretanja servisa i inicijalnog učitavanja modela, naš web servis je spreman za rad. Budući da naš model zna raditi s točno određenim tipom podataka, slikom, unutar servisa je trebalo pripremiti metodu koja će osigurati da svaki ulazni podatak bude pripremljen po istom principu kao što su bili pripremljeni i trening podaci na kojima je

model izgrađen. U tu svrhu koristimo metodu *prepare_image()* koja prima jedan parametar, reprezentaciju slike u *base64* formatu, a vraća sliku pripremljenu za ulazak u mrežu (Slika 7.2.). *Base64* je binarna shema za kodiranje teksta koja se obično koristi za prijenos poruka temeljenih na sadržaju putem Interneta. Funkcionira tako da dijeli svaka tri bita binarnih podataka na šestbitne jedinice. Novonastali podaci prikazani su kao sedmobitni ASCII tekst. Budući da je svaki bit podijeljen u dva bita, pretvoreni podaci su 33 posto veći od izvornih podataka. Poput binarnih podataka, podaci kodirani u *base64* nisu čitljivi ljudima[112]. Slika sa sučelja šalje se *ajax* pozivom kao *base64* kodirana poruka koja se kao parametar prosljeđuje gore navedenoj metodi. Unutar metode poruka se dekodira, učitava u memoriju koristeći Pillow biblioteku te se konvertira u nijanse sive boje (engl. *grayscale*). Nakon konverzije, slika se smanji na veličinu 28x28 koju model očekuje. Za smanjivanje slike koristi se tehnika *antialiasing*. To je tehnika koja se koristi u digitalnoj obradi slika kako bi se smanjili vizualni nedostatci koji se pojavljuju kada su slike visoke razlučivosti prikazane u manjoj razlučivosti. *Antialiasing* se manifestira kao nazubljene ili stepenaste linije na rubovima i objektima koji bi inače trebali biti glatki. *Antialiasing* čini ove zaobljene ili nakošene linije opet glatkima dodajući laganu diskoloraciju na rubove linije ili predmeta, uzrokujući zamagljenost i istresanje nazubljenih rubova.[113] Preostalo je obrađenu sliku putem metode *reshape()* postaviti u odgovarajući oblik (n, dubina, širina, visina) te normalizirati sliku postavljajući piksele u vrijednosti od 0 do 1. Normalizacija će nam pomoći da uklonimo izobličenja uzrokovana svjetlima i sjenama na slici.

```
89
90  def prepare_image(imageData):
91      output = BytesIO(base64.b64decode(imageData))
92      output.seek(0)
93
94      image = Image.open(output).convert("L")  # need to convert because the image comes as ARGB
95      image.thumbnail(size=(28, 28), resample=Image.ANTIALIAS)
96
97      img_for_prediction = np.array(image).reshape(1, 1, 28, 28)
98      img_for_prediction = img_for_prediction.astype("float32")
99      img_for_prediction /= 255
100
101     return img_for_prediction
102
```

Slika 7.2. Priprema slike za učitavanje u mrežu

[112] https://www.techopedia.com/definition/27209/base64, travanj. 2019.
[113] https://www.techopedia.com/definition/1950/antialiasing, travanj. 2019.

7.5. Metode web servisa

Metoda predikcije *get_prediction()* najvažnija je metoda web servisa. Sa sučelja prima nacrtanu sliku koju obrađuje navedenim metodama te ju predaje modelu na predikciju. Rezultate iz modela dobivamo pozivajući metodu *predict()*. Te vrijednosti formatiramo i spremamo u ključ-vrijednost rječnik koji u JSON formatu vraćamo u odgovoru.

Sljedeća je metoda *get_layer_names()* koja nam služi dohvatu imena svih slojeva koji se nalaze u modelu. Iz tih slojeva dohvaćamo slike koje nam služe za prikaz vizualizacije na korisničkom sučelju. Valjalo bi napomenuti da unutar metode uzimamo samo slojeve čija je dimenzija veća od 2 jer samo iz tih slojeva možemo rekreirati i napraviti sliku za prikaz. Metoda u JSON formatu vraća listu naziva svih slojeva.

Metoda *get_layer_image()* je metoda koja služi za dohvat slike iz pojedinog sloja. Metoda kao parametar prima nacrtanu sliku sa sučelja te naziv sloja iz kojeg bismo željeli dohvatiti sliku za vizualizaciju. Nakon obrade slike putem naziva sloja iz modela, dohvaćaju se izlazni parametri tog sloja. Nakon kreiranja slike možemo vidjeti transformacije koje su se dogodile na originaloj slici nakon što je prošla kroz dotični sloj. Metoda u JSON formatu vraća listu s nazivom slike te *base64* kodiranu sliku za prikaz na sučelju. Vrlo slična metoda je i *get_weight_image()* uz jedinu razliku što ona radi dohvat težinskih vrijednosti prva dva konvolucijska sloja koje na taj način možemo vizualizirati i pokazati korisniku.

8. Izrada korisničkog sučelja

Cilj ovog rada, osim treniranja konvolucijske neuronske mreže, bio je i prikazati kako se istrenirani model može koristiti u stvarnom produkcijskom okruženju. Većina programskih okvira koji se bave strojnim učenjem kao demonstraciju koriste testiranje svojih modela na skupu podataka koji je odvojen i pripremljen specifično za tu uporabu. U našem slučaju radilo se o 10 000 slika koje smo ostavili za testiranje i koje model nije vidio tijekom treniranja. Ideja je bila prikazati jedan model koji „živi“ u produkcijskom okruženju i prima ulazne podatke isključivo od korisnika putem napravljene web aplikacije. Na taj način stvarno isprobavamo mogućnosti generalizacije našeg modela jer smo sigurni da takvu vrstu ulaznih podataka model još nije vidio i upoznao. Kako bismo to postigli, kreirana je web aplikacija u programskom jeziku JavaScript koristeći React okvir.

8.1. Visual Studio Code

Za pisanje web aplikacije korišten je Visual Studio Code editor. Njega sam odabrao jer nudi novi pristup razvoju aplikacija, osigurava odličnu kontrolu nad podešavanjima samog okruženja te daje veliku pomoć pri pisanju koda. Podržava mnoštvo programskih jezika, odličan *intellisense* te spajanje i integraciju sa sustavima za verzioniranje kao što su Git ili GitHub. Prvi je od alata unutar palete Visual Studia koji se može izvoditi na više platformi kao što su Linux, macOS i Windows. Također vrijedi spomenuti da Visual Studio Code spada u projekte „otvorenog koda“ (engl. *open-source*) što znači da je cijeli kod aplikacije dostupan i vidljiv svakome.[114] Vrlo poznata web stranica za programere, Stack Overflow, u svojoj je programerskoj anketi 2019. godine objavila kako je Visual Studio Code najpopularniji editor za pisanje koda. Od 87 317 ispitanika, 50.7% njih je odgovorilo da koristi upravo taj editor u svakodnevnom radu. Microsoft ga je najavio 29. travnja 2015. godine na jednoj od svojih godišnjih konferencija, dok je 18. studenog 2015. objavljen pod MIT licencom te postao javno dostupan putem GitHub sustava za verzioniranje.[115]

[114] https://almirvuk.blogspot.com/2016/03/visual-studio-code-nodejs-development.html, srpanj. 2019.
[115] https://en.wikipedia.org/wiki/Visual_Studio_Code, srpanj. 2019.

8.2. JavaScript i React

JavaScript je interpretiran, skriptni, objektno orijentirani jezik koji je najpoznatiji kao skriptni jezik za web stranice, iako se može koristit i van okruženja Internet preglednika. Popularan među programerima, vrlo je jednostavan što je veliki plus za svakog početnika budući da je zajednica ljudi koja ga koristi jako velika. JavaScript se pokreće na klijentskoj strani unutar preglednika, a koristi se za dizajniranje, programiranje i ponašanje web stranica. U osnovi je dinamičan skriptni jezik koji podržava prototipnu konstrukciju objekata, a osnovna sintaksa mu je namjerno slična ostalim popularnima jezicima kako bi se smanjio broj novih koncepata potrebnih za učenje. JavaScript može funkcionirati kao proceduralni i kao objektno orijentirani jezik. Ima razvijene „API-je“ (engl. *application programming interface*) za rad s tekstom, nizovima, datumima, regularnim izrazima i „DOM-om“ (engl. *document object model*), ali sam jezik zbog sigurnosti ne uključuje niti jednu metodu koja bi mogla direktno koristiti bilo kakav resurs van preglednika na kojem se nalazi.[116]

React (React.js ili ReactJS) je JavaScript biblioteka koja se koristi za kreiranje korisničkih sučelja. Kreirana je i održavana od strane Facebooka i zajednice individualnih programera i kompanija. Prvenstveno služi stvaranju dinamičkih komponenti koje se mogu ponovno koristiti unutar aplikacije. Jedna od važnijih značajki Reacta je da se može koristiti na klijentu, serveru, mobilnim aplikacijama pa čak i za aplikacije virtualne stvarnosti. U današnje vrijeme, kada web aplikacije postaju sve veće i moćnije s jako puno interaktivnog, promjenjivog sadržaja, počeo se primijećivati i dodatni efekt na same performanse izvođenja. Kako bi tome doskočili, inženjeri koji su razvijali React dosjetili su se da bi bilo efikasnije mijenjati samo dio stranice umjesto ponovnog učitavanja cijele stranice kod svake promjene. Iz tog razloga su unutar Reacta inkorporirali virtualni „DOM“ (engl. *document object model*). Promjene na virtualnom DOM-u događaju se u memoriji i primijenjuju se samo na dio stranice na kojemu se promjena dogodila. Na taj su način dobili vrlo dobru responzivnost aplikacije što uvelike utječe na korisničko iskustvo. React također potiče i korištenje najboljih praksi kao što su praćenje uzoraka, razdvajanje logike, podjela sučelja na više iskoristivih komponenti, komunikacija između komponenti s kretanjem podataka u jednom smjeru te pravilnim korištenjem stanja. Također treba

[116] https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript, kolovoz. 2019.

spomenuti i da je krivulja učenja Reacta jako kratka u usporedbi s ostalim JavaScript bibliotekama i okvirima kao što su Angular, Vue ili Backbone. To proizlazi iz činjenice da je cijeli React kod napisan s modernim JavaScriptom i nije potrebno poznavanje određenih uzoraka kao što su „injekcija ovisnosti“ (engl. *dependency injection*) ili „sistemi predložaka“ (engl. *template system*) kao kod Angulara. Neke od poznatijih kompanija koje koriste React su Microsoft, Netflix, Twitter, PayPal, Tesla, Uber i mnogi drugi.[117]

8.3. Struktura web aplikacije

Početni kostur i struktura web aplikacije napravljena je koristeći Node.js i „npm“ (engl. *Node.js package manager*). Node.js je platformsko JavaScript okruženje otvorenog koda koje je u mogućnosti izvršavati JavaScript kod izvan preglednika. *Npm* je upravitelj paketa za JavaScript programski jezik koji se sastoji od „naredbenog sučelja“ (engl. *command line interface*), također nazvanog *npm*, i online baze podataka koja sadrži javne i privatno plaćene pakete nazvane *npm registry*. Paketi se mogu pretraživati putem web sučelja i uključivati u projekte po potrebi. Kako bismo mogli koristiti *npm* na računalu, potrebno je instalirati Node.js i naredbeno sučelje *npm*. Nakon instalacije, sve potrebne naredbe i instalacije paketa mogu se izvršavati preko terminala. Jedna od naredbi koju je potrebno izvršiti za kreiranje početnog kostura i strukture React aplikacije je *npx create-react app <ime-aplikacije>* unutar direktorija u kojem želimo imati spremljenu aplikaciju (Slika 8.1.). *Npx* je dio *npm*-a i služi tome da bi se pojednostavili određeni procesi instaliranja paketa s *npm* registra. Navedena naredba dohvaća sve pakete iz registra koji su potrebni kako bi React aplikacija mogla raditi.

[117] Roldan, React Cookbook: Create dynamic web apps with React using Redux, Webpack, Node.js, and GraphQL, 2018, 6

```
D:\Projects>npx create-react-app my-app

Creating a new React app in D:\Projects\my-app.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts...

+ react@16.8.6
+ react-scripts@3.0.0
+ react-dom@16.8.6
added 1397 packages from 726 contributors and audited 878612 packages in 46.146s
found 0 vulnerabilities


Initialized a git repository.

Success! Created my-app at D:\Projects\my-app
```

Slika 8.1. Postavljanje React aplikacije

Nakon instaliranja svih paketa, početna struktura aplikacije je postavljena i spremna za korištenje. Izgled početne strukture vidljiv je na slici (Slika 8.2.).

```
my-app
├── build
├── node_modules
├── public
│   ├── favicon.ico
│   ├── index.html
│   └── manifest.json
├── src
│   ├── App.css
│   ├── App.js
│   ├── App.test.js
│   ├── index.css
│   ├── index.js
│   ├── logo.svg
│   └── serviceWorker.js
├── .gitignore
├── package.json
└── README.md
```

Slika 8.2. Struktura React aplikacije

Unutar glavnog direktorija aplikacije nalazi se nekoliko direktorija od kojih svaki ima svoje posebno značenje. *Public* direktorij sadrži početnu index.html stranicu koja služi kao glavni predložak i ne smije se micati ili brisati. To je prva stranica koja se učitava prilikom podizanja aplikacije. U *src* direktoriju nalazi se index.js datoteka koja služi kao JavaScript ulazna točka i također se ne smije micati ili brisati, u protivnom se projekt neće izgraditi i javit će grešku. On sadrži i App.js datoteku koja predstavlja App komponentu. To je glavna komponenta u Reactu i služi kao „kontejner“ za sve ostale komponente koje ćemo

dodavati u aplikaciju. Direktorij *node_modules* sadrži sve pakete koji su potrebni za funkcioniranje aplikacije i u njega se prebacuju svi naknadni paketi koji su potrebni tijekom faze razvoja. Za *build* direktorij valja spomenuti kako se u njega spremaju sve pretvorene datoteke tijekom izgradnje aplikacije. Drugim riječima, budući da preglednici ne razumiju JSX, sve datoteke moraju se prebaciti u čisti JavaScript kako bi ih preglednici znali razumjeti. Taj proces događa se tijekom „izgradnje“ aplikacije (engl. *build*). Treba još spomenuti datoteke s ekstenzijom .css koje sadrže stilove za uljepšavanje i uređivanje stranica te package.json datoteku unutar koje se nalazi popis svih paketa koje aplikacija koristi.

Tijekom razvoja aplikacije, *npm* nudi i mogućnost pokretanja i podizanja same aplikacije lokalno u razvojnom modu što omogućava automatsko osvježavanje prilikom bilo kakvih promjena u kodu same aplikacije. To uvelike ubrzava i olakšava razvoj jer su promjene i greške vidljive istog trena. Naredba za pokretanje koju je potrebno izvršiti kako bi se dobila ta funkcionalnost je *npm start*. Nakon podizanja aplikacije u pregledniku, bilo kakva promjena unutar koda će nakon spremanja pokrenuti ponovnu izgradnju i osvježavanje aplikacije te će promjene biti odmah vidljive.[118]

8.4. Komponente web aplikacije

Nakon što je početna struktura aplikacije postavljena, može se krenuti u proširivanje i daljnji razvoj. Izgled aplikacije je podijeljen u nekoliko elemenata ili komponenti koje su odvojene i postavljene svaka u svoj zasebni direktorij. Tri su glavne komponente koje valja spomenuti: Canvas, Table i ImageGallery. Svaki direktorij sadrži JavaScript datoteku, datoteku sa stilovima te sve ostale datoteke koje su potrebne kako bi ta komponenta mogla funkcionirati. Najveća komponenta, ujedno i najvažnija, je komponenta Canvas. Ona prilikom inicijalnog podizanja generira html element <*canvas* /> i kreira logiku koja dozvoljava prostoručno crtanje po navedenom elementu. Ovako smo omogućili korisniku aplikacije ručni unos jednog od brojeva kojeg kasnije koristimo u neuronskoj mreži prilikom prepoznavanja. Komponenta sadrži i logiku za izvlačenje nacrtane slike iz elementa, njezinu obradu u smislu svih potrebnih radnji kako bi se slika pripremila u sliku koju mreža očekuje te logiku za slanje slike u neuronsku mrežu. Komponenta služi i kao

[118] https://medium.freecodecamp.org/quick-guide-to-understanding-and-creating-reactjs-apps-8457ee8f7123, kolovoz. 2019.

početna stranica aplikacije te se prva učitava nakon pokretanja. Na temelju dobivenih odgovora iz neuronske mreže ona je odgovorna za renderiranje ostalih komponenti unutar aplikacije.

Ako je sve dobro prošlo s obradom i unosom od strane korisnika, neuronska mreža je vratila odgovor koji treba prikazati korisniku. U tu svrhu je kreirana Table komponenta koja omogućuje tablični prikaz podataka iz mreže. Podaci koji se prikazuju su broj za koji neuronska mreža misli da je nacrtan te postotak vjerojatnosti koji nam govori koliko je mreža sigurna u svoju predikciju. Tablica prikazuje prve četiri vjerojatnosti od mogućih deset. Komponenta će se renderirati ovisno o tome je li servis vratio podatke o predikciji.

Osim tabličnog prikaza rezultata, neuronska mreža šalje i slike koje prikazuju put ulazne slike kroz određene slojeve mreže. Radi lakšeg prikaza slika kreirana je komponenta koja predstavlja galeriju slika. Komponenta se zove ImageGalleryComponent. Njeno renderiranje također ovisi o povratnoj informaciji koja dolazi iz neuronske mreže i neće se prikazati ako mreža nije vratila nikakav podatak. Nakon renderiranja, komponenta prikazuje samo jednu sliku i to sliku prvog sloja. Za sve ostale slike naknadno se zove servis koji ih vraća jednu po jednu. Na taj smo način rasteretili servis i smanjili dužinu čekanja na prikaz korisniku. Nakon učitavanja svih slika više nije potrebno zvati servis već se slike učitavaju iz „predmemorije“ (engl. *cache*). Vrijeme prikaza je trenutno i vrlo brzo. Od ostalih komponenti treba spomenuti Loader komponentu koja služi kao obavijest korisniku prilikom dužih učitavanja podataka te Logo komponentu koja služi kao vizualni dodatak stranici i nema nikakvu drugu funkcionalnost.

8.5. Priprema slike za web servis

Kako bi poslana slika iz web aplikacije mogla što bolje biti prepoznata, potrebno ju je obraditi i pripremiti na način da bude što sličnija slici na kojoj je neuronska mreža bila istrenirana. U tu svrhu je u Canvas komponentu dodan skriveni html element *<canvas />* koji nije vidljiv korisniku aplikacije, ali služi tome da se nacrtana slika što bolje obradi i pripremi za slanje. Nakon što je korisnik stisnuo gumb za slanje svoje nacrtane slike u mrežu, sliku je potrebno dohvatiti iz prvog vidljivog canvas html elementa te proći kroz sve piksele i oduzeti maksimalnu vrijednost (255) od vrijednosti svake „RGB“ (engl. *red, green, blue*) komponente. Proces obrade slike unutar web aplikacije prikazan je na slici (Slika 8.3.).

```
//Put white background behind the image
context.globalCompositeOperation = "destination-over";
context.fillStyle = "white";
context.fillRect(0, 0, 150, 150);

//get destination canvas
const destCanvas = this.refs.canvasNetwork;
const destContext = destCanvas.getContext("2d");

let image_Data = context.getImageData(0, 0, 150, 150);
let data = image_Data.data;
//invert color of image
for (var i = 0; i < data.length; i += 4) {
  // red
  data[i] = 255 - data[i];
  // green
  data[i + 1] = 255 - data[i + 1];
  // blue
  data[i + 2] = 255 - data[i + 2];
}

// set original image on destination canvas and take the url from canvas
destContext.putImageData(image_Data, 0, 0);
const image = destCanvas.toDataURL("image/png");
```

Slika 8.3. Priprema slike unutar web aplikacije

Nakon što smo sliku pripremili i napravili inverziju boje, spremamo je u skriveni *html* element kako bismo ju lakše izvukli koristeći *toDataURL(„image/png“)* metodu. Metoda enkodira podatke u *base64* format koji možemo poslati putem web servisa u neuronsku mrežu.

9. Implementacija web servisa i korisničkog sučelja

9.1. Heroku Paas (*Platform as a service*)

Heroku je „cloud platforma kao usluga“ (engl. *cloud platform as a service*), što znači da svojim korisnicima omogućuje razvijanje, pokretanje i upravljanje aplikacijama bez kompleksnosti održavanja, projektiranja i održavanja bilo kakve infrastrukture koja bi inače bila vezana uz puštanje aplikacija u rad. Usluga pruža okruženje za stvaranje, *hosting* i implementaciju aplikacija. Tako se programeru olakšava postavljanje, konfiguriranje i upravljanje elementima kao što su serveri, dostupnost aplikacije, skalabilnost, spremišta podataka, operacijski sustavi i slično. Heroku je jedna od prvih cloud platformi, a u razvoju je od 2007. godine. U početku je podržavala samo programski jezik Ruby, no tijekom godina podrška za jezike se proširila na Javu, Node.js, Python, PHP i druge. Odabrao sam Heroku kao platformu zbog odlične integracije s GitHub sustavom, podrške za najnoviju verziju Python jezika kao i za sve Python biblioteke i okvira koje sam koristio te zbog jednostavnosti korištenja prilikom puštanja web servisa u rad.[119]

9.2. Implementacija web servisa putem Heroku platforme

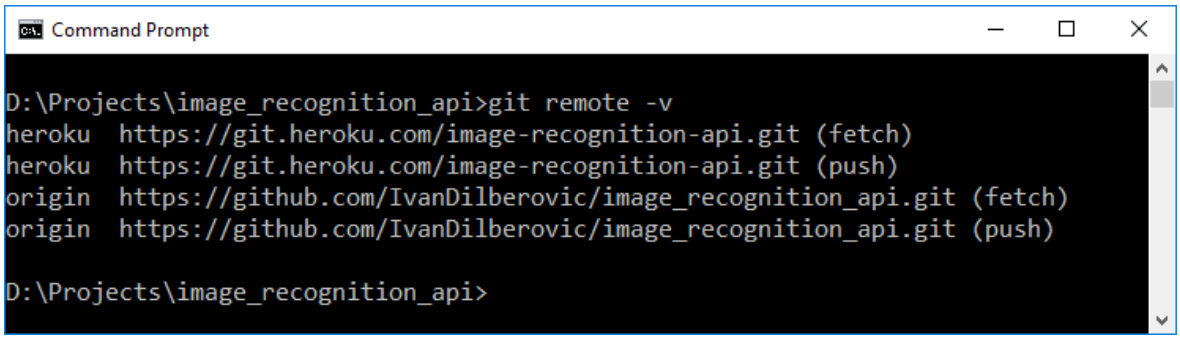
Prikaz rezultata unutar web aplikacije strogo ovisi o povratnoj informaciji dobivenoj od web servisa. Da bismo se maknuli od lokalnog načina rada i aplikaciju otvorili prema vanjskom svijetu i prema korisnicima, potrebno je web servis implementirati na neki od sustava za hosting. Izbor je pao na Heroku platformu. Kao što sam već naveo, Heroku nudi odličnu integraciju s GitHub sustavom, podržava jezik u kojem je web servis napisan te nudi besplatan korisnički račun koji je dovoljan za promet koji će aplikacija generirati. Postupak implementacije je vrlo jednostavan i odlično popraćen detaljnom dokumentacijom. Naravno, postoji nekoliko preduvjeta koji se moraju ispuniti: otvaranje besplatnog korisničkog računa, instalacija najnovije verzije Python programskog jezika te instalacija Heroku naredbenog retka, „Heroku CLI“ (engl. *Heroku Command Line*

[119] https://en.wikipedia.org/wiki/Heroku, kolovoz. 2019.

Interface). Nakon instalacije Heroku CLI unutar Windows naredbenog retka koristimo naredbu *heroku login* kako bismo se ulogirali u svoj korisnički račun. Naredba otvara preglednik i navodi nas na početnu stranicu Heroku platforme. Nakon uspješnog unosa preglednik se može zatvoriti, a putem naredbenog retka nam se ispisuje poruka o uspješnosti prijave.

Sljedeći korak odnosi se na kreiranje Heroku aplikacije. Ta aplikacija predstavlja temeljnu jedinicu organizacije na Heroku platformi. Svaka se aplikacija može povezati s vlastitim setom predviđenih dodataka, ako su potrebni. Prvo se trebamo pozicionirati u lokalni direktorij u kojem se nalazi naša aplikacija. Nakon toga pokrećemo naredbu za kreiranje Heroku aplikacije koja glasi *heroku create {naziv aplikacije} –buildpack heroku/python*. Buildpack su implementacijske skripte koje će se izvršiti za zadani programski jezik te naposljetku generirati sliku virtualnog stroja čija će instanca živjeti kao jedan proces unutar aplikacijske virtualne mašine. Za našu aplikacije izvršit će se Python skripte jer je cijeli API napisan u Pythonu koristeći Flask okvir. Sama Heroku aplikacija nije ništa drugo nego prazna aplikacija s odgovarajućim praznim *git* repozitorijem kojeg ćemo kasnije povezati s originalnim projektom koji se nalazi na GitHub-u. Aplikacija će biti dostupna nakon implementacije putem linka *https://{naziv aplikacije}.herokuapp.com*.

Nakon kreiranja Heroku aplikacije, moramo ju povezati s originalnim repozitorijem koji se nalazi na GitHub-u. Unutar lokalnog direktorija u kojem se nalazi naša aplikacija pokrećemo naredbu *heroku git:remote – a {naziv aplikacije}* te na taj način stvaramo referencu između našeg i Heroku repozitorija. Reference možemo vidjeti i provjeriti koristeći naredbu *git remote –v* što je vidljivo na slici (Slika 9.1.).



Slika 9.1. Heroku – GitHub referenca

Na kraju je potrebno prebaciti kod iz repozitorija u novokreirani Heroku repozitorij. Trebamo pokrenuti naredbu *git push heroku master*. Ovo će prebaciti sve datoteke iz našeg repozitorija, pokrenuti sve implementacijske skripte koje su potrebne te na kraju izvršiti i

pokrenuti našu aplikaciju na Heroku serveru. Kako bismo bili sigurni da barem jedna instanca naše aplikacije radi, pokrećemo naredbu *heroku ps:scale web=1*. Ovime smo pokrenuli i podigli jedan proces, koliko nam i omogućava osnovni korisnički račun na Heroku platformi. Kada smo osigurali sve resurse za našu aplikaciju, jedino što nam preostaje je pokrenuti aplikaciju i vidjeti je li sve funkcionalno. Koristeći naredbu *heroku open* otvara se generirani link prema našoj aplikaciji u zadanom Internet pregledniku.[120]

Trebalo bi spomenuti i dodatne preduvjete i korake koje moramo odraditi unutar samog projekta kako bi implementacija prošla bez problema. Prvi preduvjet je instalacija Gunicorn servera. Njega možemo instalirati kao paket koristeći naredbu *pip install gunicorn*. Gunicorn (Green Unicorn) je Python HTTP server za WSGI aplikacije, pri čemu je WSGI kratica za *Web Server Gateway Interface*. WSGI nije ništa drugo nego jednostavna konvencija pozivanja za web servere koja objašnjava kako proslijediti zahtjeve prema web aplikacijama napisanim u Python jeziku. Pruža savršenu ravnotežu performansi, fleksibilnosti i jednostavnosti konfiguracije.[121]

Kada smo instalirali server, u projektu trebamo kreirati popis svih paketa i modula koje smo koristili kako bi se ti isti paketi i moduli mogli instalirati na Heroku platformu prilikom implementacije. Kreiramo tzv. requirements.txt datoteku koje će sadržavati popis svih stvari o kojima ovisi ispravnost naše aplikacije (Slika 9.2.). Datoteka se mora nalaziti u početnom direktoriju naše aplikacije jer u protivnom implementacija na Heroku platformu neće uspjeti. Naredba kojom kreiramo datoteku glasi *pip freeze > requirements.txt*. Kako bismo bili sigurni da će svaka implementacija na serveru proći u redu, najbolje je kreirati datoteku i popisati sve pakete i module koje koristimo s njihovim točnim verzijama.

Slika 9.2. Primjer sadržaja requirements.txt datoteke

[120] https://kenya-tech.com/2019/01/06/how-to-deploy-a-flask-application-on-heroku/, kolovoz. 2019.
[121] https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface, kolovoz. 2019.

Zadnja stvar koju moramo dodati je datoteka naziva Procfile. U njoj su definirane sve naredbe koje će se izvršiti prilikom pokretanja aplikacije. To je jednostavna tekstualna datoteka bez .txt ekstenzije koja se uvijek mora nalaziti u početnom direktoriju aplikacije. U protivnom se implementacije na Heroku platformi neće izvršiti. U nju upisujemo jedan redak koji glasi *web: gunicorn app:app*. „Web: gunicorn“ označava Heroku proces koji može primati HTTP promet s Heroku usmjernika. U našem slučaju, budući da radimo web aplikaciju, koristili smo Gunicorn. Prva oznaka app odnosi se na ime Python datoteke koja se pokreće, dok se druga oznaka app odnosi na ime koje smo definirali unutar datoteke kao ime aplikacije. Drugim riječima, u app.py datoteci pokreni *app.run()*.[122]

9.3. GitHub (Sustav za verzioniranje koda)

Git je sustav za verzioniranje koda kojeg je pokrenuo Linus Torvalds, autor Linux operativnog sustava. Pokrenuo ga je zbog nezadovoljstva sa sustavima za verzioniranje koda koje je koristio do tada. Vrlo često ga koriste programeri prilikom razvijanja neke aplikacije zbog učestalih promjena u programskom kodu. Korištenjem alata Git sve promjene mogu se sačuvati kao točno određena verzija programskog koda koja se sprema u centralni repozitorij. Sve verzije programskog koda vidljive su i dostupne te se programer može u bilo kojem trenutku vratiti na bilo koju verziju. Verzioniranje programskog koda omogućuje lakšu suradnju timova na istom projektu. Svaki programer može raditi na svom zadatku i kada je spreman, može „gurnuti“ (engl. *push*) svoje promjene u centralni repozitorij. Svi ostali u timu mogu te promjene „povući“ (engl. *pull*) i vidjeti razlike. Ako je više programera napravilo promjene na istim linijama koda, Git će to prepoznati i javiti konflikt koji se mora riješiti. Konflikti nam služe kako bi se izbjegle situacije u kojima bi jedan programer mogao prebrisati programski kod drugog programera. Da bi riješili konflikt, moraju se usuglasiti oko verzije koda koja će završiti u glavnom repozitoriju. Git radi lokalno, što znači da je to aplikacija koju instaliramo na računalo i obično je koristimo iz naredbenog retka. Nakon što je instaliramo, možemo započeti s verzioniranjem naših datoteka koje se nalaze u nekom direktoriju te iz njih kreirati takozvani „repozitorij“ (engl. *repository*). Repozitorij je cjelina koja sadrži sve datoteke koje smo uključili u verzioniranje i bit će spremljen samo na našem lokalnom računalu. Ako želimo dijeliti svoj repozitorij s drugima, moramo koristiti „poslužitelja“

[122] https://devcenter.heroku.com/articles/getting-started-with-python#define-a-procfile, rujan. 2019.

(engl. *server*) koji razumije Git. Jedan od najpopularnijih mrežnih servisa koji omogućuje besplatno spremanje Git repozitorija i njihovo dijeljenje je GitHub.com. On nudi odlične alate za spremanje, ažuriranje, dijeljenje i općenito praćenje Git repozitorija. Osim osnovnih alata koji za verzioniranje repozitorija, GitHub.com nudi i mogućnost posluživanja mrežnih stranica iz samih repozitorija. Servis koji to omogućuje je GitHub Pages. GitHub Pages omogućuje stvaranje neograničenog broja mrežnih stranica. Jedna mrežna stranica može biti vezana uz ime našeg korisničkog računa na GitHub.com servisu i uz neograničen broj repozitorija koje imamo spremljene na GitHub.com servisu.[123]

Rad s GitHubom je vrlo jednostavan. Potrebno je na računalo instalirati Git i otvoriti korisnički račun na GitHub stranici. Nakon toga sve radnje mogu se izvršavati putem komandnog retka iz samog Visual Studio Code editora.

Cjelokupni projekt ovog rada spremljen je na GitHubu i javno je dostupan.

9.4. Podizanje web aplikacije putem GitHub Pages servisa

Nakon završetka programiranja i lokalnog testiranja aplikacije, odlučio sam iskoristiti GitHub Pages servis kako bi aplikacija postala javno dostupna i spremna za korištenje. Sama implementacija zahtijeva nekoliko koraka koji se moraju odraditi kako bi sve prošlo u redu. Preduvjet je da imamo već kreirani repozitorij s učitanim kodom na samom GitHub servisu.

Prvi korak je postavljanje početne stranice aplikacije unutar package.json datoteke u samom projektu. Početna stranica nije ništa drugo nego besplatna domena na GitHub Pages servisu u obliku *https://korisnickoime.github.io/repozitorij*. Postavke datoteke su vidljive na slici (Slika 9.3.).

```
"name": "mnist_image_recognition",
"version": "0.1.0",
"private": true,
"proxy": "http://localhost:3000",
"homepage": "https://ivandilberovic.github.io/mnist_image_recognition",
```

Slika 9.3. Postavke unutar package.json datoteke

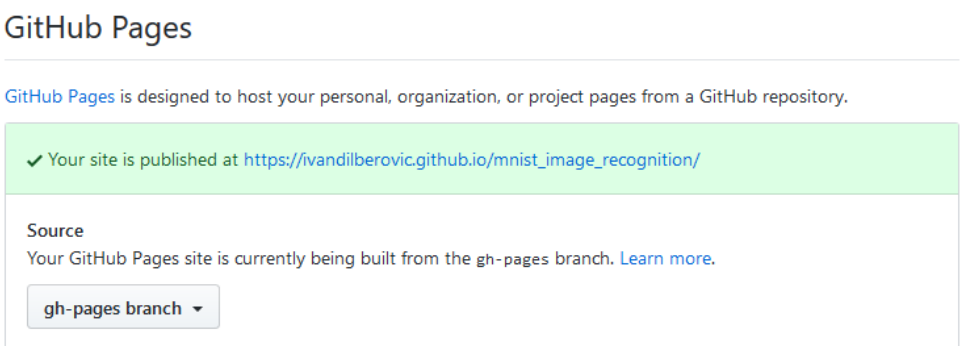
[123] https://sistemac.srce.hr/github-pages-108, rujan. 2019.

Zatim sam instalirao paket pod imenom gh-pages koristeći npm servis. On nam služi kako bismo što lakše klonirali i objavili svoj projekt u novom gh-pages repozitoriju koji će se automatski objaviti na GitHub Pages servisu. Korištenje je vrlo jednostavno. Trebamo instalirati paket putem naredbenog retka u direktorij u kojem se nalazi naš projekt. Naredba za instalaciju paketa glasi *npm install --save gh-pages*. Nakon uspješne instalacije paketa, potrebno je dodati implementacijske naredbe u package.json datoteku (Slika 9.4.). Naredba *predeploy* će zapakirati i pripremiti sve projektne datoteke koje su potrebne prilikom implementacije, dok će *deploy* naredba prebaciti generirane datoteke u novokreirani gh-pages repozitorij.[124]

```
"scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject",
    "predeploy": "npm run build",
    "deploy": "gh-pages -d build"
},
```

Slika 9.4. Implementacijske naredbe unutar package.json datoteke

Nakon ubacivanja implementacijskih skripti, potrebno je putem naredbenog retka pokrenuti naredbu za implementaciju *npm run deploy*. Ako je sve prošlo u redu, projekt bi trebao biti implementiran i dostupan putem gore navedenog linka. Kako bismo to provjerili, potrebno je prijaviti se na GitHub sučelje te se pozicionirati na karticu Postavke unutar koje je moguće provjeriti stanje GitHub Pages servisa. Kao što je vidljivo na slici (Slika 9.5.), web aplikacija je objavljena, javno je dostupna te spremna za korištenje.



Slika 9.5. GitHub Pages verifikacija

[124] https://reactgo.com/deploy-react-app-github-pages/, rujan. 2019.

Zaključak

Tijekom izrade ovog rada, od samih početaka pokušao sam koristiti nekoliko različitih tehnologija i programskih jezika kako bih uspio dobiti funkcionalan istrenirani model kojeg bih mogao koristiti u produkciji. Svi pokušaji, osim ovdje opisanog, završili su neuspješno. Neki modeli jednostavno nisu bili dobri i precizni, za neke je treniranje na osnovnom MNIST setu podataka trajalo danima, a neki jednostavno nisu bili upotrebljivi van lokalnog okruženja zbog mnogih internih problema i načina na koji rade. To me dovelo do programskih okvira kao što su Keras i Tensorflow koji su pokušali riješiti neke od čestih problema kao što su modularnost, laka proširivost, optimizacija koda, korištenje resursa grafičke kartice i slično. Iza navedenih okvira i ostalih koji nisu navedeni, stoje velika imena poput Google-a, Microsofta, Facebooka, IBM-a, Amazona. Njihova misao vodilja glasi:„Ideje dubokog učenja su jednostavne, zašto bi implementacija trebala biti bolna?“ Na taj način su pojednostavili korištenje i približili krajnjeg korisnika (programera) nekim novim područjima koja su prije uglavnom bila rezervirana za znanstvenike na raznim fakultetima i znanstvenim institucijama. Također, većina okvira pripada zajednici otvorenog koda. To omogućuje programerima aktivno sudjelovanje u samom razvoju i optimizaciji koda što je dovelo do izvanrednih rezultata. Zato je danas moguće vrlo jednostavno napisati program koji će u nekoliko minuta moći istrenirati model na velikom setu podataka i pokušati dati odgovore na tražena pitanja. Uz takav način rada moguće je puno naučiti jer uglavnom postoji i detaljna dokumentacija. Zbog svega navedenog odabrao sam jedan takav okvir za vlastiti rad. Tu dolazimo do važnog pitanja u poslovnom svijetu:„Koji je Vaš poslovni slučaj?“ Sve tehnologije koje su zaživjele i opstale su tehnologije koje su iza sebe imale poslovni slučaj koji je rješavao neki problem i donosio korist. Ako taj slučaj nije bio dobar ili nije donosio dovoljno koristi, vrlo brzo bi bio zaboravljen. Osobno smatram da je to još uvijek slučaj sa strojnim učenjem, dubokim učenjem te umjetnom inteligencijom. Iako korištenje tih tehnologija ubrzano raste, vrlo velik broj kompanija i dalje ne zna kako bi takvo što monetizirala, pogotovo kada se uzmu u obzir velika ulaganja potrebna kako bi se jedan takav veliki model postavio i koristio u produkciji. Činjenica je da to i dalje ostaje u rukama velikih igrača kao što su Google, Amazon, Microsoft, Facebook i drugi koji imaju dovoljno resursa za istraživanje te si mogu priuštiti povremene moguće gubitke na tom polju.

Gledajući iz vlastite perspektive, aplikacija koju sam napravio prepoznaje rukom napisane brojeve što je vrlo zanimljivo, ali ne vidim niti jedan poslovni slučaj u kojem bi to moglo opstati. Takve aplikacije mogle bi se koristiti kao primjer budućim studentima nekog *data science* smjera na fakultetu, no u tome nema nikakve financijske svrhe koja je ipak, nažalost, jedna od pokretača današnjeg svijeta. Za kraj, želio bih naglasiti kako su strojno učenje, duboko učenje te umjetna inteligencija vrlo zanimljivo područje u kojemu se svakodnevno događaju izvanredne stvari. Zajednica je velika i svakim danom postaje sve veća. Radeći na ovom radu naučio sam puno novih stvari i otkrio načine kako možemo upotrijebiti stroj da temeljem podataka donosi odluke i predikcije. Međutim, moje osobno mišljenje je da još uvijek ne postoji mnogo dobrih poslovnih slučajeva koji bi mogli rezultirati financijskom korišću.

Popis slika

Slika 4.1 Višeslojni perceptron s dva ulazna, dva skrivena i jednim izlaznim neuronom .. 16

Slika 4.2 RNN mreža i odvijena RNN mreža 18

Slika 5.1 Struktura konvolucijskih neuronskih mreža 21

Slika 5.2 Konvolucija: a) Prvi korak; b) Korak nakon pomaka receptivnog polja 22

Slika 5.3 Prvi korak konvolucija kod potpuno povezanih slojeva 22

Slika 5.4 Primjer raspršene povezanosti 23

Slika 6.1 Prikaz pretraga po programskim okvirima 35

Slika 6.2 Prikaz slojeva nad kojima se nalazi Keras 36

Slika 6.3 Prikaz slojeva unutar modela 37

Slika 6.4 Prikaz funkcije gubitka 38

Slika 6.5 Prikaz učitavanja i normaliziranja MNIST seta podataka 39

Slika 6.6 Prikaz konvolucijske operacije 40

Slika 6.7 Prikaz maksimalnog udruživanja 41

Slika 6.8 Prikaz modela 42

Slika 6.9 Podešavanje modela te početak treniranja 43

Slika 6.10 Graf funkcije gubitka i preciznosti 43

Slika 6.11 Evaluacija i spremanje modela 44

Slika 6.12 Obrada učitane slike 45

Slika 7.1 Kreiranje virtualnog okruženja 47

Slika 7.2 Priprema slike za učitavanje u mrežu 49

Slika 8.1 Postavljanje React aplikacije 54

Slika 8.2 Struktura React aplikacije 54

Slika 8.3 Priprema slike unutar web aplikacije 57

Slika 9.1 Heroku – GitHub referenca 59

Slika 9.2 Primjer sadržaja requirements.txt datoteke 60

Slika 9.3 Postavke unutar package.json datoteke 62

Slika 9.4 Implementacijske naredbe unutar package.json datoteke 63

Slika 9.5 GitHub Pages verifikacija 63

Popis formula

(1) Centrirana diskretna izvedbena maska (engl. *discrete derivative mask*) 7

(2) Centrirana diskretna izvedbena maska (engl. *discrete derivative mask*) 7

(3) Derivacije slike I u x i y smjerovima 7

(4) Derivacije slike I u x i y smjerovima 7

(5) Orijentacija θ i veličina │g│ gradijenta 8

(6) Orijentacija θ i veličina │g│ gradijenta 8

(7) Normalizacijski faktor 8

(8) Normalizacijski faktor 8

(9) Normalizacijski faktor 8

(10) Hessianova matrica H(p,σ) 10

(11) Gubitak unakrsne entropije (engl. *cross-entropy loss*) 24

(12) *Softmax* funkcija 24

(13) *Hinge loss* 24

(14) Euklidski gubitak 24

(15) l^1 gubitak 24

(16) Kontrastni gubitak 25

(17) Gubitak očekivanja 25

(18) Xavier inicijalizacija 26

(19) ReLU aware scaled inicijalizacija 26

Literatura

[1] A Guide to Convolutional Neural Networks for Computer Vision, Salman Khan, Hossein Rahmani, Syed Afaq Ali Shah, and Mohammed Bennamoun, ISBN: 9781681730226 ,Copyright © 2018 by Morgan & Claypool

[2] Vikram Srivastava, Prashant Goyal, An Efficient Image Identification Algorithm using Scale Invariant Feature Detection

[3] François Chollet, Deep Learning with Python, ISBN 9781617294433, ©2018 by Manning Publications Co.

[4] Giuseppe Ciaburro, Balaji Venkateswaran, Neural Networks with R, ISBN 978-1-78839-787-2, Copyright © 2017 Packt Publishing

[5] Karačić, V. (2016). *Korištenje web tehnologija za izradu i prikaz višeslojnih neuronskih mreža* (Diplomski rad). Preuzeto s https://urn.nsk.hr/urn:nbn:hr:166:455086

[6] Božić-Stulić D. Semantička segmentacija slika metodama dubokog učenja (Diplomski rad). Preuzeto sa: https://data.fesb.unist.hr/public/news/Dunja Božić-Štulić-a3b4859ad0.pdf

[7] Vukotić, V. (2014). *Raspoznavanje objekata dubokim neuronskim mrežama* (Diplomski rad). Preuzeto s https://urn.nsk.hr/urn:nbn:hr:168:749831

[8] Essert, Mario. Digitalni udžbenik Python - osnove - Odjel za matematiku Sveučilišta Josipa Jurja Strossmayera Osijek, 2007.

[9] Carlos Santana Roldan, React Cookbook: Create dynamic web apps with React using Redux, Webpack, Node.js, and GraphQL, ISBN 978-178398-072-7, Copyright © 2018 Packt Publishing

[10] Python, https://en.wikipedia.org/wiki/Python_(programming_language), travanj. 2019.

[11] Tensorflow, https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html, travanj. 2019.

[12] Logaritamski gubitak, http://wiki.fast.ai/index.php/Log_Loss, travanj. 2019.

[13] Python virtualno okruženje, https://realpython.com/python-virtual-environments-a-primer/, travanj. 2019.

[14] Base64, https://www.techopedia.com/definition/27209/base64, travanj. 2019.

[15] *Antialiasing*, https://www.techopedia.com/definition/1950/antialiasing, travanj. 2019.

[16] Visual Studio Code, https://almirvuk.blogspot.com/2016/03/visual-studio-code-nodejs-development.html, srpanj. 2019.

[17] Visual Studio Code, https://en.wikipedia.org/wiki/Visual_Studio_Code, srpanj. 2019.

[18] JavaScript, https://developer.mozilla.org/docs/Web/JavaScript/About_JavaScript, kolovoz. 2019.

[19] Postavljanje React aplikacije, https://medium.freecodecamp.org/quick-guide-to-understanding-and-creating-reactjs-apps-8457ee8f7123, kolovoz. 2019.

[20] Heroku, https://en.wikipedia.org/wiki/Heroku, kolovoz. 2019.

[21] Postavke Heroku servisa, https://kenya-tech.com/2019/01/06/how-to-deploy-a-flask-application-on-heroku/, kolovoz. 2019.

[22] Web Server Gateway Interface, https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface, kolovoz. 2019.

[23] Heroku ProcFile, https://devcenter.heroku.com/articles/getting-started-with-python#define-a-procfile, rujan. 2019.

[24] Git, https://sistemac.srce.hr/github-pages-108, rujan. 2019.

[25] GitHub Pages, https://reactgo.com/deploy-react-app-github-pages/, rujan.2019.

ALGEBRA
VISOKO
UČILIŠTE

PREPOZNAVANJE SLIKA POMOĆU KONVOLUCIJSKE NEURONSKE MREŽE

Pristupnik: Ivan Dilberović, 0195017407

Mentor: prof. dr. Marko Velić