Određivanje smetnji na slici metodama strojnog učenja

Jambriško, Petra

Određivanje smetnji na slici metodama strojnog učenja

Jambriško, Petra

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Petra Jambriško

ODREĐIVANJE SMETNJI NA SLICI METODAMA STROJNOG UČENJA

Diplomski rad

Voditelj rada:
prof. dr. sc. Luka Grubišić

Zagreb, rujan, 2020.

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

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

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Poglavlje 1

Uvod

Posljednjih je godina tehnologija doživjela nagli skok i pružila korisnicima pregršt mogućnosti za uživanje i lagodan život. Više nije moguće zamisliti život bez pametnih telefona s odličnom kamerom, uređajima sa senzorima i prepoznavačima objekata. No, bilo u memoriji telefona, ili u detekciji objekata, zamućene slike nikako nisu poželjne. Kako nastaju zamućene slike? Odgovor je poprilično očit. Ljudi su danas često u pokretu i u toj užurbanosti žele fotografirati neki objekt, koji je također ponekad u pokretu, a događa se i da je leća objektiva zamagljena pa slika ne ispada dobro.

U svakom slučaju, javlja se potreba za određivanjem činjenice je li slika zamućena ili nije. U problemu sa zamućenim slikama, ono čemu bi dobra klasifikacija mogla pridonijeti jest odvajanje ili čak možda i automatsko brisanje takvih „loših“ slika. Kod senzora, koji možda detektiraju neke objekte na slici, nakon što bi slika bila shvaćena kao zamućena, mogao bi se pokrenuti postupak izoštravanja slike (en. *deblur*), kako bi se približno dobila izvorna slika i neki bi objekti lakše bili detektirani. Problem klasifikacije je li slika mutna ili nije cilj je ovog rada. Ovom bi se problemu moglo pristupiti matematički i računalno.

Shvati li se sliku kao matricu dimenzija koje predstavljaju njezinu širinu i visinu te da svaki element matrice predstavlja intenzitet određene boje u točki, tada se sliku lako može predstaviti realnom diskretnom funkcijom dvije varijable (x i y predstavljaju širinu i visinu) kojoj je vrijednost funkcije jednaka upravo intenzitetu boje na slici. Interpretiranu ovako, sliku se na razne načine može transformirati primjenom različitih operatora koji dijeluju na prostorima tako definiranih funkcija. Da bi se otkrilo je li slika zamućena ili nije, metodom koja koristi derivacije drugog reda prvo će se pronaći rubovi, tj. područja na kojima se intenzitet naglo mijenja, a zatim, ovisno o tome je li rubova puno ili malo, dat će se odgovor na postavljeno pitanje.

Kako je cilj rada ipak više na računalnim metodama i mogućnostima rješavanja klasifikacijskih problema, rezultati gore spomenute matematičke metode koristit će se za usporedbu i dodatno mjerilo uspješnosti modela. U radu će biti predstavljena tri modela

prepoznavanja zamućenja slike. Krenut će se od neuronskih mreža, koje postaju sve popularniji alati u rješavanju problema strojnog učenja upravo iz razloga što su građene tako da imitiraju ponašanje ljudskog mozga i njegove obrade informacija i učenja nad informacijama.

Ideja je pomoću već viđenih podataka, koji rješavaju zadani problem, stvoriti znanje i naučiti algoritam kako riješiti taj isti problem na novim podacima, na onome što algoritmu još nije poznato. Baš kao i mozak, nakon što nauči prepoznati ili razlikovati objekte na temelju njihovih značajki, tako i algoritam mora biti sposoban uspješno rješavati problem na nekom novom skupu podataka. To je cilj svakog modela strojnog učenja pa tako i modela neuronskih mreža, od kojih će ovdje, uz obične višeslojne neuronske mreže, kao druga metoda biti obrađene i testirane konvolucijske neuronske mreže, konstruirane isključivo za rad s višedimenzionalnim ulaznim podacima pa time i sa slikama.

Na kraju će, kao treća metoda, biti opisana novija metoda učenja dubokih neuronskih mreža, prijenosno učenje, gdje se za učenje koriste unaprijed trenirani modeli na velikom skupu podataka što uvelike može ubrzati, ali i povećati uspješnost korištenja modela.

Kroz rad se, dakle, detaljno opisuje svaka od metoda, uči i testira na skupu podataka (slika) iz javno dostupne mape *CERTH_ImageBlurDetection* koja se sastoji od dvije podmape *TrainingSet* i *EvaluationSet*. U obje se datoteke nalazi određeni broj čistih i mutnih slika.

1.1 Digitalna slika i obrada slike

Digitalna slika je numerički prikaz dvodimenzionalne slike zadan funkcijom koja položaju (x, y) pridružuje piksel čija vrijednost označava intenzitet svjetlosti na elementu slike predstavljenom tim pikselom. Ovisno o vrijednosti piksela, mogu se razlikovati crno-bijele slike, slike u sivim tonovima i slike u boji. Svaki piksel crno-bijele slike ima jednu od dvije moguće vrijednosti, obično 0 ili 1. Vrijednosti piksela slike u sivim tonovima su vrijednosti između 0 i 255, gdje 0 označava crnu, a 255 bijelu boju. Piksel digitalne slike u boji sadrži tri vrijednosti, intenziteta crvene, zelene i plave boje, a taj je prikaz poznat kao RGB prikaz slike. Svaka je vrijednost ponovo između 0 i 255 pa vrijednost $(0, 255, 255)$ označava da je na tom mjestu intenzitet crvene boje minimalan, dok su zeleni i plavi maksimalni, što rezultira žutom nijajsom. Da bi se RGB slika u boji pretvorila u sliku sivih tonova, često se koriste sljedeće formule za pretvorbu:

$$S = 0,3086R + 0,6094 \cdot G + 0,0820 \cdot B$$
$$S = 0,299 \cdot R + 0,587 \cdot G + 0,114 \cdot B,$$

gdje je sa S označen dobiveni intenzitet sive boje, a s R, G i B su redom označeni intenziteti crvene, zelene i plave boje. Može se primijetiti da je intenzitet sive boje zbroj različitih

težina intenziteta crvene, zelene i plave. Ako se koristi ista težina, na primjer $(R+G+B)/3$, tada čisto crvena, čisto zelena i čisto plava rezultiraju istom razinom intenziteta sive boje. A drugi razlog za korištenje različitih težina je taj što je ljudsko oko osjetljivije na zelene i crvene komponente nego na plave. Svi dodatni detalji mogu se pronaći u [27].

U pogledu računalnih znanosti, digitalna obrada slike upotreba je digitalnog računala za obradu digitalnih slika putem odabranih algoritama. Budući da su slike definirane u dvije dimenzije (moguće i više), digitalna obrada slike može se modelirati u obliku višedimenzionalnih sustava. Na stvaranje i razvoj digitalne obrade slike uglavnom utječu tri čimbenika: razvoj računala, razvoj matematike (posebno stvaranje i poboljšanje diskretne teorije matematike) i povećanje potražnje za širokim rasponom primjena u okolišu, poljoprivredi, vojsci, industriji, medicini, i slično. Dakle, obrada slike metoda je izvođenja određenih operacija na slici kako bi se dobila poboljšana slika ili iz nje izvukli neki korisni podaci. Obradom slike, izlaz može također biti slika, ali i karakteristike ili značajke povezane s tom slikom. Obrada slike u osnovi uključuje tri koraka, a to su uvoz slike pomoću alata za prikupljanje slika, obrada, analiza i manipulacija slikom te izlaz u kojem rezultat može biti izmijenjena slika ili izvještaj koji se temelji na analizi slike.

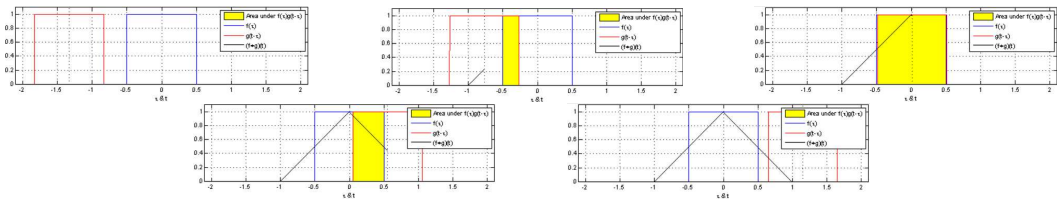
Jedan od načina obrade digitalne slike jest filtriranje korištenjem konvolucijskih transformacija jezgrinim funkcijama. U obradi slike, jezgra (en. *kernel*), konvolucijska matrica ili maska ustvari je matrica manjih dimenzija. Koristi se za zamagljivanje, izoštravanje, otkrivanje rubova i još mnogo toga. To se postiže primjenom diskretne konvolucije između kernela i slike. Ta se operacija primijenjuje da se izlaz poboljša ili transformira na željeni način.

1.2 Konvolucija

Konvolucija je matematička operacija koja prima dvije funkcije (f i g), a kao rezultat daje treću funkciju ($f * g$) izražavajući način na koji se oblik jedne mijenja pod utjecajem druge funkcije. Sam izraz konvolucije odnosi se i na rezultantnu funkciju, ali i na proces računanja navedene operacije. Definirana je kao integral:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau. \tag{1.1}$$

Iako se koristi simbol t, on ne mora predstavljati vremensku domenu. No, u tom se kontekstu konvolucija može opisati kao težinski prosjek funkcije $f(\tau)$ u trenutku t gdje je težina izražena s $g(-\tau)$, pomaknuta za vrijednost t. Kako se t mijenja, funkcija g naglašava različite dijelove ulazne funkcije.



Slika 1.1: Funkcija $g(\tau)$, tj. označena crvenim "impulsom", parna je funkcija ($g(-\tau) = g(\tau)$). Na slikama je prikaz funkcija $g(t-\tau)$ i $f(\tau)$ za neku vrijednost parametra t, koja se proizvoljno definira kao udaljenost od osi $\tau = 0$ do središta crvenog "impulsa". Žutom bojom prkazana je površina $f(\tau)\cdot g(t-\tau)$, dobivena formulom (1.1). Rezultat (prikazan crnom bojom) je funkcija po t, ali je prikazana na istoj osi kao i τ, za praktičnost i usporedbu.[2]

Definicija iz (1.1) proširuje se u višedimenzionalne prostore. Budući da se slika promatra kao funkcija dvije varijable, ovdje se formula za konvoluciju proširuje samo na dvodimenzionalni prostor. Za funkcije dvije varijable, f i g, konvolucija $f * g$ dana je izrazom

$$(f * g)(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\chi, \tau) g(x - \chi, y - \tau) d\chi d\tau. \tag{1.2}$$

Za realne funkcije f i g, definirane na skupu $\mathbb{Z} \times \mathbb{Z}$, definira se diskretna konvolucija funkcija dviju varijabli

$$f(x, y) * g(x, y) = \sum_{i=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} f(i, j) g(x - i, y - j). \tag{1.3}$$

Propozicija 1.2.1. *Za neprekidne (ili diskretne) funkcije dviju varijabli f_1, f_2 i f_3 vrijede sljedeća svojstva:*

- *komutativnost*
$$f_1 * f_2 = f_2 * f_1$$
- *asocijativnost*
$$f_1 * (f_2 * f_3) = (f_1 * f_2) * f_3$$
- *distributivnost prema zbrajanju*
$$f_1 * (f_2 + f_3) = (f_1 * f_2) + (f_1 * f_3)$$

Dokaz. Prikazan je dokaz navedenih svojstava samo za diskretne funkcije jer je takvom funkcijom prikazana slika nad kojom se operacija provodi.

Prvo se dokazuje komutativnost.

$$\begin{aligned} f_1 * f_2 &= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f_1(i,j) f_2(x-i, y-j) \\ &= (\text{supstitucija: } k = x-i, l = y-j) \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f_1(x-k, y-l) f_2(x,y) \\ &= f_2 * f_1. \end{aligned}$$

Dalje se dokazuje asocijativnost.

$$\begin{aligned} f_1 * (f_2 * f_3) &= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f_1(i,j)(f_2 * f_3)(x-i, y-j) \\ &= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f_1(i,j) \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f_2(k,l) f_3((x-i)-k, (y-j)-l) \end{aligned}$$

Ono što se želi dobiti jest:

$$\begin{aligned} (f_1 * f_2) * f_3 &= \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} (f_1 * f_2)(m,n) f_3(x-m, y-n) \\ &= \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f_1(i,j) f_2(m-i, n-j) f_3(x-m, y-n). \end{aligned}$$

Uvede li se u gornji raspis supstitucija $m = i + k$ i $n = j + l$, dolazi se do

$$f_1 * (f_2 * f_3) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f_1(i,j) f_2(m-i, n-j) f_3(x-m, y-n).$$

Time se pokazalo da asocijativnost vrijedi. □

Na kraju, dokazuje se i distributivnost.

$$\begin{aligned} f_1 * (f_2 * f_3) &= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f_1(i,j)(f_2 + f_3)(x-i, y-j) \\ &= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f_1(i,j) f_2(x-i, y-j) + \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f_1(i,j) f_3(x-i, y-j) \\ &= f_1 * f_2 + f_1 * f_3. \end{aligned}$$

U pozadini ove operacije stoji teorija linearnih mehaničkih i optičkih sustava u sklopu digitalne obrade signala, gdje je signal fizikalna veličina koja nosi neku informaciju i koju je moguće izmjeriti, manipulirati njome, prenijeti ju ili obraditi nekim fizikalnim procesom, tj. sustavom.

Definicija 1.2.2. *Za domenu $D \subseteq \mathbb{R}$ i kodomenu $K \subseteq \mathbb{R}$* signal *je definiran kao preslikavanje $x : D \mapsto K$. Ukoliko je D neprekinut i neprebrojiv skup, riječ je o* koninuiranom signalu*, a ukoliko je $D \subseteq \mathbb{Z}$, tj. domena je diskretna, riječ je o* diskretnom signalu*. Dodatno, ukoliko je domena višedimenzionalna, tada je i signal* višedimenzionalan*, a ukoliko je kodomena višedimenzionalna, tada je signal* vektorski*.*

Definicija 1.2.3. Sustav *je veza ulaznog i izlaznog signala $\{(x, y) \,|x \in X, y \in Y\}$, gdje je X prostor ulaznih signala, a Y prostor izlaznih signala. Preslikavanje je određeno operatorom $L : X \mapsto Y$, a kaže se da je sustav* linearan *ukoliko je preslikavanje L homogeno i aditivno, tj. vrijedi*

$$L(\alpha \cdot x) = \alpha \cdot Lx, \text{za svaki } \alpha \in \mathbb{R} \text{ i svaki } x \in X$$
$$L(x_1 + x_2) = Lx_1 + Lx_2, \text{ za sve } x_1, x_2 \in X.$$

Za sustav se kaže da je vremenski stalan ako pomak u ulaznom signalu prolaskom kroz sustav rezultira istim pomakom u izlaznom signalu, tj. ako je $Lx(n) = y(n)$, za svaki $n \in \mathbb{Z}$, tada je $Lx(n - n_0) = y(n - n_0)$ za neki $n_0 \in \mathbb{Z}$.

Definicija 1.2.4. *Neka je s δ označen jedinični impuls na domeni $\mathbb{Z}$, tj. za svaki $n \in \mathbb{Z}$ vrijedi*

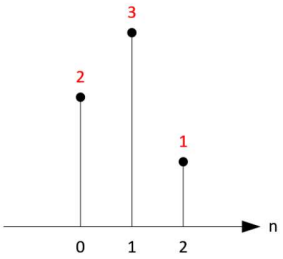
$$\delta(n) = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}.$$

Neka je s h označen odziv (izlazni signal) linearnog sustava na jedinični impuls δ, tj. $f(\delta) = h$. Tada se h naziva impulsni odziv sustava.

U ovim terminima, operacijom konvolucije se za proizvoljni ulazni signal izlazni signal sustava može dobiti poznavajući njegov impulsni odziv i koristeći svojstvo dekompozicije signala. Signal je moguće dekomponirati na jednostavne aditivne komponente i prikazati ga kao aditivnu sumu bazičnih signala. Izlazni se signal dobiva sumiranjem izlaza bazičnih signala koji prolaze kroz sustav.

Primjer 1.2.5. *Neka je s x označen signal sa slike (1.2). Koristeći jedinični impuls δ, taj je signal moguće rastaviti na sljedeći način:*

$$\begin{aligned} x(0) &= x(0) \cdot \delta(n) = 2 \cdot \delta(n) \\ x(1) &= x(1) \cdot \delta(n-1) = 3 \cdot \delta(n-1) \\ x(2) &= x(2) \cdot \delta(n-2) = 1 \cdot \delta(n-2) \\ x(n) &= x(0) \cdot \delta(n) + x(1) \cdot \delta(n-1) + x(2) \cdot \delta(n-2). \end{aligned}$$



Slika 1.2: Jednostavni primjer signala

Općenito, signal se može zapisati kao zbroj skaliranih i pomaknutih jediničnih impulsa:

$$x(n) = \sum_{k=-\infty}^{+\infty} x(k)\,\delta(n-k).$$

Ukoliko je sustav kroz koji signal prolazi linearan i vremenski stalan (uz operator preslikavanja L), za ulazni signal x, izlazni će signal biti jednak

$$\begin{aligned} y(n) &= L\left(\sum_{k=-\infty}^{+\infty} x(k)\,\delta(n-k)\right) \\ &= \sum_{k=-\infty}^{+\infty} x(k) \cdot L\delta(n-k) \\ &= \sum_{k=-\infty}^{+\infty} x(k) \cdot h(n-k) \\ &= x(n) * h(n). \end{aligned}$$

Ovime je prikazana konvolucija na jednodimenzionalnim signalima. Na dvodimenzionalnim diskretnim signalima, što su i slike, jedinični se impuls proširuje na sljedeći način:

$$\delta(m,n) = \begin{cases} 1,\ m = 0, n = 0 \\ 0 \text{ inače} \end{cases}.$$

Ponovno, signal može biti prikazan sumom skaliranih i pomaknutih jediničnih impulsa:

$$x(m,n) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} x(i,j)\,\delta(m-i, n-j).$$

Zato je izlaz linearnog, vremenski stalnog sustava moguće zapisati konvolucijom ulaznog signala i odziva sustava:

$$y(m,n) = x(m,n) * h(m,n) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} x(i,j)h(m-i, n-j).$$

U dvodimenzionalnim sustavima i obradi slike odziv se još naziva jezgrom ili filterom. Glavna je ideja da se operacija primijenjuje lokalno, što je ostvareno time da se manja matrica, tj. filter, konačne veličine i oblika "provuče" kroz sliku i provede operacija konvolucije kojom se aproksimira primjena željenog operatora na sliku. Vrijednost izlaznog piksela bit će težinska suma ulaznih piksela kroz filter gdje su težine vrijednosti filtera.

Ako je dan filter h dimenzija $J \times K$, kooordinate $(j = 0, k = 0)$ označavat će centar matrice H. Taj je centar dobro definiran ako su dimenzije matrice neparne, a u slučaju da su one parne, koristit će se aproksimacije $(\frac{J}{2}, \frac{K}{2})$ za centar filtera, tj. konvolucijske matrice.

$$H = \begin{bmatrix} h(-\frac{J-1}{2}, -\frac{K-1}{2}) & \dots & \dots & h(0, -\frac{K-1}{2}) & \dots & \dots & h(\frac{J-1}{2}, -\frac{K-1}{2}) \\ \vdots & \ddots & \vdots & \vdots & \vdots & \iddots & \vdots \\ \vdots & \dots & h(-1,-1) & h(0,-1) & h(1,-1) & \dots & \vdots \\ h(-\frac{J-1}{2}, 0) & \dots & h(-1,0) & h(0,0) & h(1,0) & \dots & h(\frac{J-1}{2}, 0) \\ \vdots & \dots & h(-1,1) & h(0,1) & h(1,1) & \dots & \vdots \\ \vdots & \iddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ h(-\frac{J-1}{2}, \frac{K-1}{2}) & \dots & \dots & h(0, \frac{K-1}{2}) & \dots & \dots & h(\frac{J-1}{2}, \frac{K-1}{2}) \end{bmatrix}$$

Ako se vrijednosti filtera h postave na nulu izvan matrice $j \in \{0, 1, \dots, J-1\}$, $k \in \{0, 1, ..., K-1\}$, kovolucija ulazne slike s danim filterom svodi se na:

$$f(x,y) * h(x,y) = \sum_{i=0}^{J-1} \sum_{j=0}^{K-1} f(i,j)h(x-i, y-j). \tag{1.4}$$

Promijene li se granice tako da budu u skladu s time da postoji "centar" kovolucijske matrice, pokazuje se da se mogu zahtijevati vrijednosti $f(j,k)$ koje su izvan granica slike:

$$(f * h)(x,y) = \sum_{i=-J_0}^{J_0} \sum_{j=-K_0}^{K_0} f(i,j)h(x-i, y-j), \quad J_0 = \frac{J-1}{2},\ K_0 = \frac{K-1}{2},$$

ili, zbog svojstva komutativnosti,

$$(f * h)(x, y) = \sum_{i=-J_0}^{J_0} \sum_{j=-K_0}^{K_0} h(i, j) f(x - i, y - j). \tag{1.5}$$

Pitanje je na koje vrijednosti treba postaviti vrijednosti funkcije kojom je slika reprezentirana za $m < 0$, $m \geq M$, $n < 0, n \geq N$? Na ovo pitanje ne postoji pravi odgovor. Neke od opcija su proširiti slike konstantnom (npr. nulom) vrijednošću intenziteta, periodično ili zrcalno po njenim granicama proširiti vrijednosti slike ili postaviti vrijednosti na granicama na neograničeno, tj. beskonačno. Ovdje se slika proširuje zrcaljenjem po rubovima, i to bez udvostručenja rubnog elementa.

Primjerom koji slijedi prikazana je konvolucija dvije matrice, gdje će jedna matrica predstavljati sliku, a druga će matrica predstavljati filter.

Primjer 1.2.6. *Potrebno je izračunati matricu Y koja se dobije konvolucijom matrica X i H koje su zadane kao*

$$X = \begin{bmatrix} 1 & 1 & 4 & 5 & 6 \\ 1 & 2 & 4 & 3 & 8 \\ 1 & 3 & 1 & 4 & 5 \\ 2 & 3 & 4 & 8 & 5 \\ 3 & 5 & 6 & 1 & 1 \end{bmatrix}, \qquad H = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

Da bi se mogli izračunati rubni elementi matrice Y, matricu X potrebno je proširiti zrcaljenjem po rubovima. Dobivaju se sljedeće rubne vrijednosti prikazane vektorima:

$$x_{\cdot,-1} = \begin{bmatrix} 2 & 1 & 2 & 3 & 2 & 5 & 2 \end{bmatrix}^t, \qquad x_{\cdot,5} = \begin{bmatrix} 3 & 5 & 3 & 4 & 8 & 1 & 8 \end{bmatrix}^t,$$
$$x_{-1,\cdot} = \begin{bmatrix} 2 & 1 & 2 & 4 & 3 & 8 & 3 \end{bmatrix}, \qquad x_{5,\cdot} = \begin{bmatrix} 2 & 2 & 3 & 4 & 8 & 5 & 8 \end{bmatrix}.$$

Uvrštavajući dimenzije matrice H u jednadžbu (1.5), dolazi se do formule za vrijednosti elemenata od Y

$$\begin{aligned} y(i, j) = {} & h(-1, -1) \cdot x(i + 1, j + 1) + h(-1, 0) \cdot x(i + 1, j) + h(-1, 1) \cdot x(i + 1, j - 1) \\ & + h(0, -1) \cdot x(i, j + 1) + h(0, 0) \cdot x(i, j) + h(0, 1) \cdot x(i, j - 1) \\ & + h(1, -1) \cdot x(i - 1, j + 1) + h(1, 0) \cdot x(i - 1, j) + h(1, 1) \cdot x(i - 1, j - 1). \end{aligned}$$

Uvrštavanjem vrijednosti za pojedini element, dolazi se do konačnog rješenja Y.

$$Y = X * H = \begin{bmatrix} 0 & 5 & -2 & -4 & 2 \\ 2 & 1 & -6 & 9 & -15 \\ 5 & -5 & 11 & 1 & 1 \\ 1 & 2 & 2 & -18 & 2 \\ 2 & -5 & -10 & 19 & 8 \end{bmatrix}.$$

Poglavlje 2

Detekcija zamućenja korištenjem Laplaceovog operatora i OpenCV

2.1 Detekcija rubova na slici

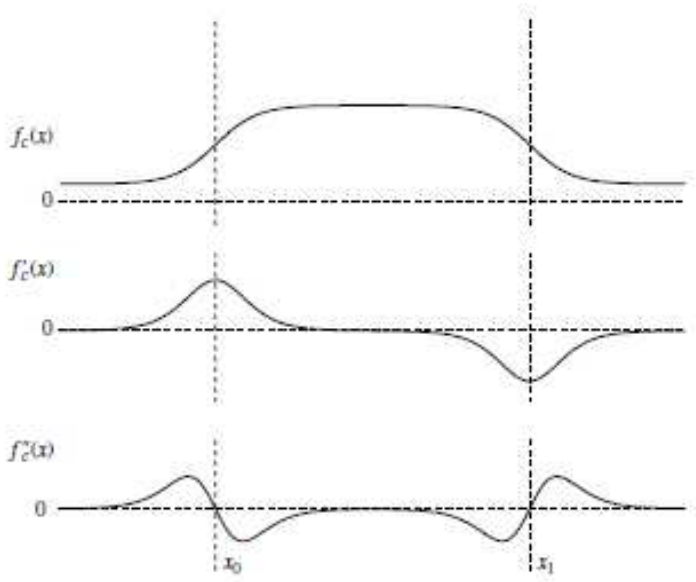
Kako bi se odredilo je li slika zamućena ili nije koristi se jedna od metoda detekcije rubova na slici kojoj je cilj identificiranje točaka u digitalnoj slici kod kojih se svjetlina slike naglo mijenja. Točke u kojima se intenzitet svjetline slike naglo mijenja, obično su organizirane u skup zakrivljenih segmenata linija nazvanih rubovima. Otkrivanje rubova temeljno je sredstvo u obradi slike, strojnom i računalnom vidu, posebno u područjima otkrivanja i ekstrakcije značajki.

Postoje mnoge metode za otkrivanje rubova, ali većina ih se može grupirati u dvije kategorije, one temeljene na pretraživanju i one koje traže prijelaze nule (en. *zero crossing*). Prve otkrivaju rubove izračunavanjem mjere jačine ruba, koristeći derivacije prvog reda. Smjer prve derivacije pokazuje prema ekstremima, a iznos određuje strmost lokalnog nagiba u određenoj točki. Metode temeljene na prijelazima nule, ovdje je to Laplaceova metoda, za detekciju rubova koriste derivacije drugog reda koje pokazuju zakrivljenost funkcije, odnosno informacije koje služe za točke u kojoj dolazi do promjene smjera funkcije i u kojoj će biti minimum ili maksimum funkcije. Kao korak prije otkrivanja rubova, gotovo se uvijek izvodi faza izglađivanja, najčešće Gaussovo izglađivanje (en. *Gaussian smoothing*) koje svojim djelovanjem uklanja šum i smjetnje sa slike.

Promatra li se slika kao funkcija gdje vrijednost izlaza predstavlja intenzitet sive boje, tj. promatra li se slika u sivim tonovima, tada se rub definira kao nagla promjena razine sive boje. Pomoću [34], [14], [24], [5], [1] i [3] opisat će se na koji se način rub detektira, ali se najprije slikom (2.1) prikazuje ideja detekcije ruba svijetlog područja na tamnoj pozadini zadanog funkcijom jedne varijable.

Lijevi dio funkcije količine sive boje $f_c(x)$ pokazuje glatki prijelaz iz tamnog u svijetli

kako se x povećava. Neka je točka x_0 u kojoj se događa iz područja s niskom amplitudom s lijeve strane na susjedno područje s visokom amplitudom u sredini. Gradijentni pristup otkriva rub u točki x_0 u kojoj f postiže lokalni maksimum ili, ekvivalentno, $f_c'(x)$ postiže lokalni ekstrem, kao što je prikazano na drugom grafu slike (2.1). Drugom derivacijom, ili Laplaceovim pristupom, pronalazi se rub u točki x_0 gdje dolazi do prijelaza nule, tj. gdje je $f_c''(x) = 0$, što je prikazano na trećem grafu slike (2.1). Desna strana sa slike ilustrira slučaj za padajući rub smješten na x_1.



Slika 2.1: Prikaz metode pronalaženja rubova pomoću jednodimenzionalnog signala

Da bi se koristio pristup gradijentom ili Laplaceov pristup kao osnova za detekciju ruba slike, proces se mora proširiti na dvije dimenzije te se prilagoditi diskretnom slučaju.

Budući da je norma gradijenta mjera promjene, a slika se može smatrati nizom vrijednosti neke neprekidne funkcije intenziteta slike, analogno tome, značajne promjene sivih vrijednosti na slici mogu se otkriti pomoću diskretne aproksimacije gradijenta. Gradijent se definira kao vektor

$$\nabla f(x,y) = \begin{bmatrix} f_x(x,y) \\ f_y(x,y) \end{bmatrix} = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix}.$$

Vektor $\nabla f(x,y)$ pokazuje u smjeru maksimalne brzine rasta funkcije $f(x,y)$ [11], a

apsolutna vrijednost gradijenta, dana jednadžbom

$$|\bigtriangledown f(x,y)| = \sqrt{f_y(x,y)^2 + f_y(x,y)^2}$$

jednaka je maksimalnoj brzini rasta funkcije $f(x,y)$ po jedinici udaljenosti u smjeru $\bigtriangledown f(x,y)$. Međutim, apsolutna se vrijednost gradijenta obično aproksimira sljedećim izrazom:

$$\bigtriangledown f(x,y) \approx |f_x(x,y)| + |f_y(x,y)|$$

ili

$$\bigtriangledown f(x,y) \approx \max(|f_x(x,y)|, |f_y(x,y)|).$$

Dodatno, smjer gradijenta definiran je sljedećim izrazom:

$$\alpha(x,y) = \arctan\left(\frac{f_x(x,y)}{f_y(x,y)}\right),$$

gdje se kut α mjeri od osi x.

Kako je po definiciji derivacija funkcije f u smjeru vektora v u proizvoljnoj točki c jednaka

$$\frac{\partial f}{\partial v}(c) = \lim_{h\to 0} \frac{f(c+hv) - f(c)}{h},$$

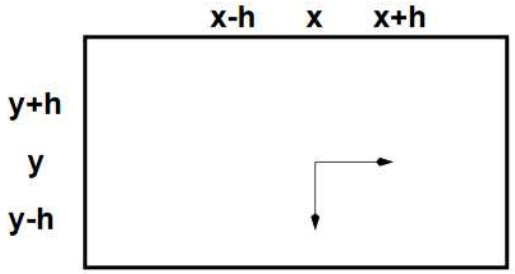
tj. posebno za smjer x i smjer y vrijedi:

$$\frac{\partial f}{\partial x}(x,y) = \lim_{h\to 0} \frac{f(x+h,y) - f(x,y)}{h}$$
$$\frac{\partial f}{\partial y}(x,y) = \lim_{h\to 0} \frac{f(x,y+h) - f(x,y)}{h},$$

tada je najjednostavnija aproksimacija gradijenta jednaka:

$$\frac{\partial f}{\partial x}(x,y) \approx \frac{f(x+h_x,y) - f(x,y)}{h_x} = f(x+1,y) - f(x,y), \text{ (za } h_x = 1) \tag{2.1}$$
$$\frac{\partial f}{\partial y}(x,y) \approx \frac{f(x,y+h_y) - f(x,y)}{h_y} = f(x,y+1) - f(x,y), \text{ (za } h_y = 1). \tag{2.2}$$

Ono što treba pripaziti jest da ako se slika promatra kao skup piksela, gdje proizvoljni piksel ima koordinate (i,j), tada j odgovara smjeru x, a i odgovara negativnom y smjeru. Prikaz toga je na slici (2.2).



Slika 2.2: Prikaz koordinata slike

Za sliku reprezentiranu funkcijom f vrijedi sljedeće:

$$\frac{\partial f}{\partial x}(i,j) = f(i,j+1) - f(i,j), \tag{2.3}$$

$$\frac{\partial f}{\partial y}(i,j) = f(i-1,j) - f(i,j) \text{ ili } \frac{\partial f}{\partial y}(i,j) = f(i,j) - f(i+1,j). \tag{2.4}$$

To može biti implementirano jednostavnim konvolucijskim filterima:

$$\frac{\partial f}{\partial x} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}^t \qquad \text{i} \qquad \frac{\partial f}{\partial y} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}. \tag{2.5}$$

Neka je dio slike dan matricom

$$\begin{bmatrix} \ddots & & & \iddots \\ & f(i-1,j-1) & f(i-1,j) & f(i-1,j+1) & \\ & f(i,j-1) & f(i,j) & f(i,j+1) & \\ & f(i+1,j-1) & f(i+1,j) & f(i+1,j+1) & \\ \iddots & & & & \ddots \end{bmatrix}.$$

Tada je

$$\begin{aligned} f * \begin{bmatrix} 1 & -1 \end{bmatrix}(i,j) &= 1 \cdot f(i-0, j-(-1)) - 1 \cdot f(i-0, j-0) \\ &= f(i,j+1) - f(i,j) \\ &= \frac{\partial f}{\partial x}(i,j). \end{aligned}$$

Slično vrijedi i za parcijalnu derivaciju u smjeru y:

$$\begin{aligned} f * \begin{bmatrix} -1 \\ 1 \end{bmatrix}(i,j) &= -1 \cdot f(i-(-1), j-0) + 1 \cdot f(i-0, j-0) \\ &= f(i,j) - f(i+1) \\ &= \frac{\partial f}{\partial y}(i,j). \end{aligned}$$

Na sličan se način mogu odrediti aproksimacije za derivacije slike drugog reda, tj. definira se Laplaceov operator i njegova aproksimacija u diskretnom prostoru. Dodatno se odredi konvolucijski filter koji mu odgovara.

Vrijednost druge derivacije slike iznosit će 0 na mjestu ruba. Laplaceov operator je dvodimenzionalni ekvivalent druge derivacije. Za proizvoljnu funkciju f, definira se kao:

$$\nabla^2 f = \frac{\partial^2 f}{\partial^2 x} + \frac{\partial^2 f}{\partial^2 y} \tag{2.6}$$

Aproksimacija parcijalne derivacije drugog reda funkcije f je tada

$$\begin{aligned} \frac{\partial^2 f}{\partial^2 x}(x,y) &= \lim_{h \to 0} \frac{\frac{\partial f}{\partial x}(x+h, y) - \frac{\partial f}{\partial x}(x,y)}{h} \approx \frac{\partial f}{\partial x}(x+1, y) - \frac{\partial f}{\partial x}(x,y) \\ &= f(x+2, y) - 2f(x+1, y) + f(x,y), \text{ uzevši } h = 1. \end{aligned}$$

Gornja je aproksimacija centrirana oko $x + 1$. Zamijeni li se $x + 1$ sa x, dobije se

$$\frac{\partial^2 f}{\partial^2 x}(x,y) \approx f(x+1, y) - 2f(x,y) + f(x-1, y).$$

Analogno, isto se dobije i za smjer y:

$$\frac{\partial^2 f}{\partial^2 y}(x,y) \approx f(x, y+1) - 2f(x,y) + f(x, y-1).$$

Ponovno, kako za piksel slike reprezentirane funkcijom f, s koordinatama (i, j), j odgovara smjeru x, a i odgovara negativnom y smjeru, konačno se dolazi do

$$\frac{\partial^2 f}{\partial^2 x}(i,j) \approx f(i, j+1) - 2f(i,j) + f(i, j-1) \tag{2.7}$$

$$\frac{\partial^2 f}{\partial^2 y}(i,j) \approx f(i+1, j) - 2f(i,j) + f(i-1, j). \tag{2.8}$$

Uvrste li se dobiveni izrazi u jednadžbu (2.6), dobije se

$$\nabla^2 f(i,j) = -4f(i,j) + f(i, j+1) + f(i, j-1) + f(i+1, j) + f(i-1, j). \tag{2.9}$$

Laplaceov operator može biti implementiran konvolucijskim filterom ili jezgrom

$$L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$

što se može provjeriti za neki element slike $f(i, j)$.

$$\begin{aligned} f(i,j) * \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} &= \sum_{k=-1}^{1} \sum_{l=-1}^{1} h(k,l) \cdot f(i-k, j-l) \\ &= 0 + 1 \cdot f(i+1, j) + 0 + 1 \cdot f(i, j+1) - 4 \cdot f(i,j) + 1 \cdot f(i, j-1) \\ &\quad + 0 + 1 \cdot f(i-1, j) + 0 \\ &= \nabla^2 f(i,j) \end{aligned}$$

Laplaceov operator primijenjen na sliku ističe područja slike u sivim tonovima koja sadrže brže promjene intenziteta, pa je rezultat ove operacije slika sa sivkastim rubnim linijama i ostalim diskontinuitetima na tamnoj pozadini.

Primjer 2.1.1. *Ukoliko se promatra rezultat konvolucije matrica X i H iz primjera (1.2.6), gdje je konvolucijska matrica H upravo bila matrica koja implementira Laplaceov operator, rezultat je bio*

$$Y = X * H = \begin{bmatrix} 0 & 5 & -2 & -4 & 2 \\ 2 & 1 & -6 & 9 & -15 \\ 5 & -5 & 11 & 1 & 1 \\ 1 & 2 & 2 & -18 & 2 \\ 2 & -5 & -10 & 19 & 8 \end{bmatrix}.$$

Kada bi matrica $\boldsymbol{x}$ *predstavljala sliku u sivim tonovima, moglo bi se zaključiti da se rubovi nalaze tamo gdje dolazi do promjene predznaka između dva susjedna elementa matrice* $\boldsymbol{y}$*. Jedno takvo mjesto bilo bi između drugog i trećeg piksela u prvom retku.*

Sada je potrebno odrediti je li slika zadana funkcijom intenziteta sivih tonova f dimenzija $M \times N$ zamućena ili nije. Za to će se koristiti varijanca Laplasijana, tj. varijanca rezultata dobivenog primjenom Laplaceovog operatora na promatranu sliku. Može se zaključiti da zamućene slike jedva imaju rubove, dok ih one fokusirane imaju puno više. Kada bi se izračunala varijanca, uz pretpostavku da bijela boja ima intenzitet 255, a crna 0, ako je velik dio slike crne boje, tada je varijanca mala, što je u slučaju mutnih slika, jer su mala odstupanja u intenzitetima sive boje. Suprotno tome, ako je bijelih i crnih područja otprilike podjednako, što bi značilo da je slika fokusirana i čista, varijanca će biti visoka. Dakle, varijanca može poslužiti kao dobra mjera za klasifikaciju, tj. odvajanje mutnih od čistih/fokusiranih slika.

Varijanca se računa tako da se prvo primijeni Laplaceov operator na zadanu sliku (konvolucija slike s maskom operatora). Neka je s $L_*(i, j)$ označena vrijednost koja se dobije konvolucijom dane slike f s jezgrom L u točki (i, j). Ako se ta vrijednost izračuna za svaku točku slike i uzme njena apsolutna vrijednost, dobije se

$$\mathrm{LAP}(f) = \sum_{i=1}^{M} \sum_{j=1}^{N} |L_*(i, j)|. \tag{2.10}$$

Varijanca realne funkcije je kvadratno odstupanje od srednje vrijednosti, pa je

$$\mathrm{LAP\_VAR}(f) = \sum_{i=1}^{M} \sum_{j=1}^{N} \left[|L_*(i, j)| - \overline{L_*} \right]^2, \tag{2.11}$$

gdje je $\overline{L_*}$ srednja vrijednost

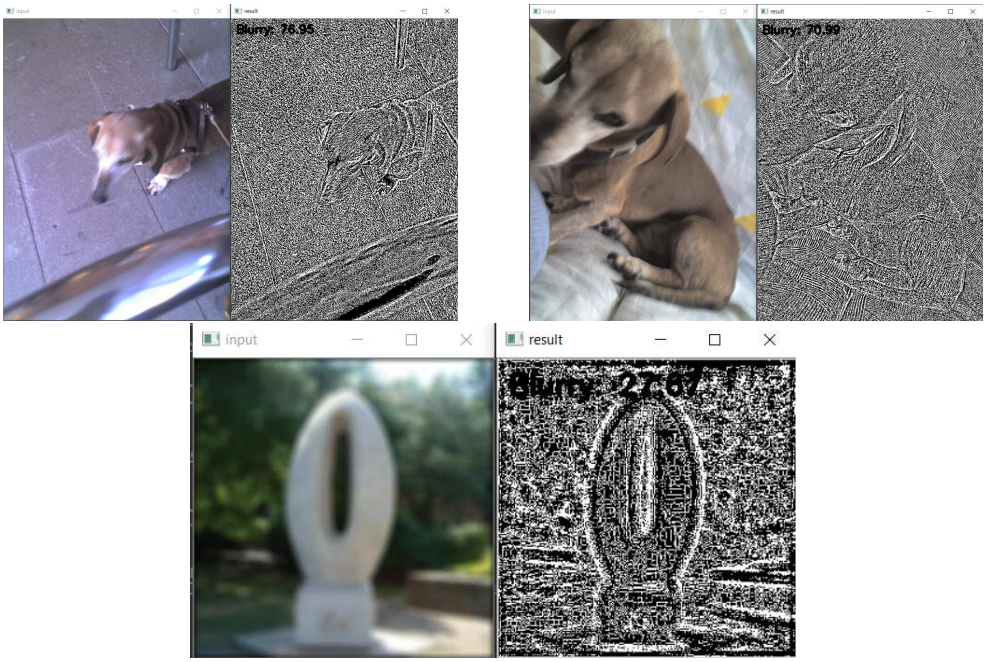
$$\overline{L_*} = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} |L_*(i, j)| \tag{2.12}$$

Ono što ostaje za odrediti jest prag iznad kojeg će se slika smatrati čistom, a ispod kojeg će se slika smatrati mutnom. Postavljanje ispravnog praga može ovisiti o domeni. Ako je prag previsok, tada će slike biti pogrešno označene mutnima kada one to nisu. Prenizak prag će slike koje jesu mutne označiti da to nisu. Primjer slike koja nam može zadati probleme jest slika neba. Čak i ako je fokusirana, možda će imati malu varijancu, jer i nema baš dobro definirane rubove. Ova metoda najbolje funkcionira u okruženjima u kojima je moguće izračunati prihvatljiv raspon mjerenja fokusa i zatim lako odrediti stršeće vrijednosti. Konačno, viša varijanca znači da je slika manje zamućena i slično, manja razina varijance znači da je razina zamućenja veća. Rezultati su prikazani u poglavlju ispod.

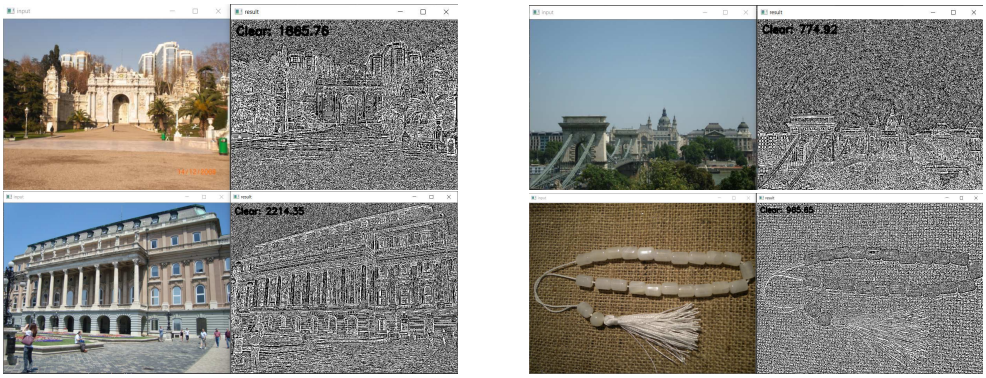
2.2 Testiranje

Slike koje se koriste učitane su naredbom *load_img* iz biblioteke *keras.preprocessing.image*, a njihove su matrične reprezentacije spremljene i kasnije čitane iz *.pkl* datoteka naredbama iz biblioteke *pickle*. Treniranja i testiranja vrše na slikama dimenzija 96×96 i 224×224.

Prvo se provjerava kako izgleda slika nakon što se na nju primijeni Laplaceov operator te izračuna varijanca dobivenog rezultata.

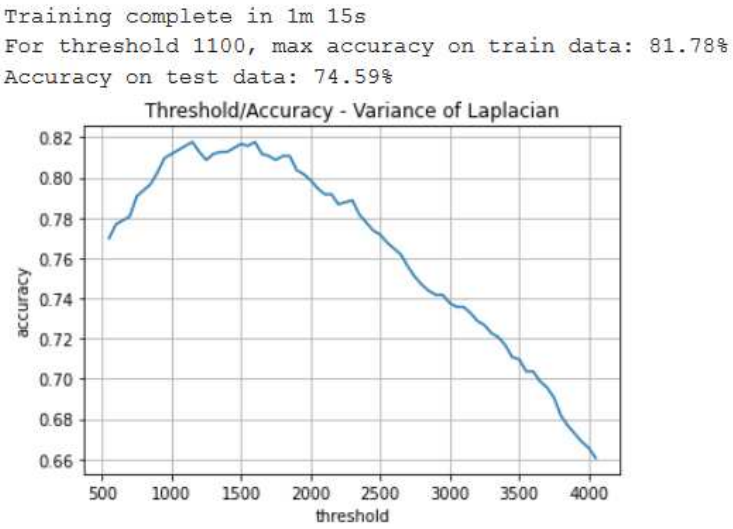


Slika 2.3: Primjer zamućenih slika s niskom varijancom Laplasijana



Slika 2.4: Primjer čistih slika s visokom varijancom Laplasijana

Uz pomoć [26], primjenom Laplacovog operatora na sliku i računanjem varijance dobivenog Laplasijana slike u sivim tonovima, dimenzija 224×224, iz mape *TrainingSet* dobivaju se rezultati sa slike (2.5)

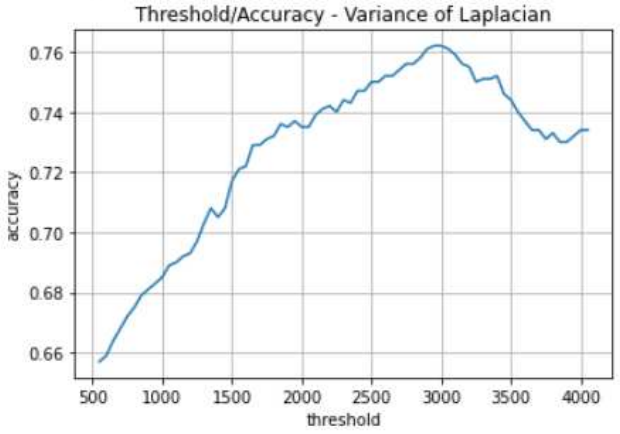


Slika 2.5: Odnos točnosti i granične vrijednosti varijance Laplasijana slika iz *TrainingSet*-a dimenzija 224×224

Dakle, maksimiziranjem točnosti na skupu za treniranje, dobiva se najveća točnost od 81.78% za graničnu vrijednost varijance 1100. Uz tako dobivenu varijancu, točnost na skupu za testiranje, što čine slike istih dimenzija i u sivim tonovima iz mape *EvaluationSet*, iznosi 74.59%. Ovo se može smatrati dobrom razinom točnosti za tako jednostavnu metodu klasifikacije. Dalje se provjerava postoji li model strojnog učenja koji ima veću točnost od ovoga ili mogu li neke poznate metode strojnog učenja na barem jednako dobar način prepoznati je li slika zamućena ili nije. Odabiru se metode nadziranog strojnog učenja gdje se za svaku sliku koja se koristi za treniranje i testiranje točno zna je li ona mutna ili nije.

Prethodna se dimenzija koristila jer su na skup slika takvih dimenzija primijenjene metode prijenosnog učenja o kojem će biti riječi kasnije. Kako bi za takve dimenzije slika modeli neuronskih mreža koje su objašnjene i testiranje u nastavku bili prespori, slike se smanjuju na dimenzije 96×96. Na skupu za treniranje se dobiva maksimalna točnost od 76.20% za graničnu vrijednost varijance 2900, a koristeći tu vrijednost varijance, na skupu za testiranje se dobiva točnost od 76.28%.

```
Training complete in 0m 14s
For threshold 2900, max accuracy on train data: 76.20%
Accuracy on test data: 76.28%
```



Slika 2.6: Odnos točnosti i granične vrijednosti varijance Laplasijana slika iz *TrainingSet*-a dimenzija 96×96

Poglavlje 3

Strojno učenje i neuronske mreže

3.1 Strojno učenje

Strojno učenje jedna je od najutjecajnijih i najmoćnijih tehnologija u današnjem svijetu kojom se podaci pretvaraju u znanje. Kako je u posljednjih nekoliko desetljeća došlo do prikupljanja mnoštva podataka, a ti su podaci beskorisni ako ih se ne analizira i tako pronađu korisni uzorci među njima, potrebne su tehnike strojnog učenja koje se koriste za automatsko pronalaženje vrijednih osnovnih obrazaca unutar složenih podataka. Ti se obrasci zajedno sa znanjem o problemu mogu koristiti za predviđanje budućih događaja i obavljanje svih vrsta složenih odluka. Razlog zašto je strojno učenje zanimljivo jest taj što je korak dalje od svih prethodnih sustava temeljenih na pravilima "if $x = y$: do z". Tradicionalno su se kombinirala pravila kako bi se dobio odgovor na neki problem. Umjesto toga, strojno učenje koristi podatke i odgovore kako bi otkrio pravila koja stoje iza problema.

Da bi naučili pravila koja rješavaju zadani problem, strojevi moraju proći kroz proces učenja, isprobavajući različita pravila i učiti na temelju svoje uspješnosti. Teorem poznat po imenu *No Free Lunch*, "Nema besplatnog ručka", navodi da ne postoji niti jedan algoritam koji će raditi dobro za sve zadatke. Svaki zadatak koji treba riješiti ima svoje vlastite posebnosti. Stoga postoji mnogo algoritama i pristupa koji odgovaraju svakom pojedinom problemu. Stoga će se još puno više stilova strojnog učenja i umjetne inteligencije tek uvoditi kako bi se pronašli oni koji najbolje odgovaraju različitim problemima. Danas su najpoznatiji pristupi provođenja strojnog učenja nadzirano, nenadzirano, polunadzirano i pojačano učenje. Prije samog objašnjenja navedenih oblika strojnog učenja, uvodi se potrebna terminologija.

- Skup podataka: Skup primjera podataka koji sadrže značajke važne za rješavanje problema.
- Značajke: Važni dijelovi podataka koji pomažu da problem bude razumljiv. Oni se

uvode u algoritam strojnog učenja kako bi mu se pomoglo u učenju.

- Model: reprezentacija onoga što je algoritam strojnog učenja naučio iz podataka za treniranje. Dakle, model je rezultat koji se dobije nakon treniranja algoritma.

Sam postupak učenja algoritma strojnog učenja kreće od prikupljanja i pripremanja podataka, zatim slijedi treniranje i evaluacija, kojom se model testira i provjerava njegova uspješnost. Zaključi li se da model uspješno rješava problem, tj. da pokazuje dovoljno dobre rezultate na podacima za testiranje, on je spreman na korištenje i rješavanje problema u području za koje je namijenjen.

Kako se kroz rad obrađuju i grade modeli nadziranog učenja, u nastavku se opisuju upravo nadzirano i nenadzirano učenje, kako bi se opisale osnovne ideje, ali i razlike između navedene dvije metode ([10], [23]).

Nadzirano i nenadzirano učenje

Cilj nadziranog učenja je naučiti preslikavanje (pravila) između skupa ulaza i izlaza. Takvim se problemom bavi i ovaj rad, jer koristeći skup podataka za koji se zna točna oznaka, cilj je naučiti preslikavanje koje određuje što je bitno da ulazna slika bude klasificirana kao zamućena, a što da bude klasificirana kao čista.

Na temelju parova ulaza i izlaza tijekom procesa učenja model uči kako se treba ponašati, a algoritam kasnije mora biti u mogućnosti dati dobru odluku za nove ulazne podatke, tj. slike. Pritom nadzirani model treba što bliže definirati stvarni odnos između ulaznih podataka i izlaza. Ako je model pretjerano uvježban, dolazi do njegove prenaučenosti, što rezultira dobrim ponašanjem na primjerima koji se koriste za učenje, ali se model neće moći prilagoditi novim, prije neviđenim ulazima.

Nuspojava do koje dolazi u nadziranom učenju jest da nadzor koji se pruža unosi pristranost u učenje. Model može samo imitirati točno ono što je prikazano, stoga je vrlo važno pokazati ga pouzdanim, nepristranim primjerima. Također, nadzirano učenje obično zahtijeva puno podataka prije nego što nauči. Dobivanje dovoljno pouzdano označenih podataka često je najteži i najskuplji dio korištenja nadziranog učenja.

Kao u problemu odvajanja mutnih i čistih slika, rezultat nadziranog modela strojnog učenja zapravo je jedna kategorija iz konačnog skupa. Kada je to slučaj, odlučuje se kako klasificirati ulazni podatak, pa je takav algoritam poznat kao klasifikacija. Alternativno, izlaz može biti realan broj iz nekog intervala. Kada je to slučaj, model je poznat kao regresija.

Za razliku od nadziranog, u nenadziranom učenju u primjerima su navedeni samo ulazni podaci, tj. ne postoje označeni primjeri rezultata za odgovarajuće ulazne podatke. Unatoč tome, još je uvijek moguće pronaći zanimljive i složene obrasce skrivene u podacima bez ikakvih oznaka i nekako ih grupirati. Ovakvo učenje može biti teže od nadzi-

ranog, jer uklanjanje nadzora znači da je problem postao manje definiran. Algoritam ima manje jasnu ideju o tome koje obrasce treba tražiti.

Da bi se pronašle zanimljive strukture u neobilježenim podacima, koristi se procjena gustoće. Najčešći oblik je grupiranje ili klasteriranje. Klasteriranje je postupak stvaranja grupa s različitim karakteristikama i ono pokušava pronaći različite podskupine unutar skupa podataka. Kako je ovo nenadzirano učenje, ne postoji ograničenje ni na koji skup oznaka i postoji sloboda koliko klastera stvoriti.

3.2 Umjetna nauronska mreža

Jedan od danas najpoznatijih modela nadziranog učenja jest umjetna neuronska mreža (en. *artificial neural network* ili ANN), ili kraće neuronska mreža, nadahnuta biološkim neuronskim mrežama koje čine ljudski i životinjski mozak. Neuronska se mreža temelji na skupu povezanih jedinica zvanih umjetni neuroni koji na određeni način modeliraju neurone u biološkom mozgu. Svaka veza može prenijeti signal s jednog na druge neurone. Umjetni neuron koji prima signal tada ga obrađuje i može proslijediti dalje neuronima koji su na njega povezani. Signal na vezi je realan broj, a izlaz svakog neurona izračunava se nekom nelinearnom funkcijom zbroja njegovih ulaza. Neuroni i rubovi obično imaju težinu koja se prilagođava tijeku učenja, a ona povećava ili smanjuje jačinu signala pri vezi. Neuroni mogu imati prag takav da se signal šalje samo ako obrađeni signal prijeđe taj prag. Taj se prag modelira parametrom koji se naziva pristranost (en. *bias*). Tipično su neuroni podijeljeni u slojeve. Različiti slojevi mogu izvoditi različite transformacije na svojim ulazima. Signali putuju od prvog sloja (ulaznog sloja) do posljednjeg sloja (izlaznog sloja) koji predstavlja konačni i željeni rezultat.

Primjer 3.2.1. *U terminima strojnog učenja definira se perceptron, jedinica s ponderiranim ulazima koja daje binarni izlaz temeljen na određenom pragu. Dakle, to je funkcija koja predstavlja binarni klasifikator i ona preslikava ulaz x u izlaz $f(x)$ koji je jedna binarna vrijednost:*

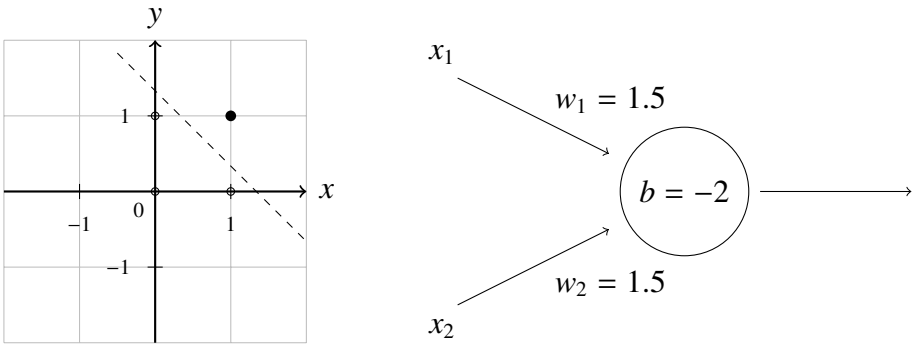
$$f(x) := \begin{cases} 1, & w \cdot x + b > 0, \\ 0, & \textit{inače}, \end{cases} \tag{3.1}$$

gdje je w vektor s realnim vrijednostima težina, $w \cdot x$ je skalarni produkt vektora w i x, a b je pristranost. Pristranost pomiče granicu odluke od nule i ne ovisi o ulaznoj vrijednosti. Ako je b negativan, tada $w \cdot x$ mora dati pozitivnu vrijednost veću od $|b|$ da bi se perceptorn "akrtivirao", tj. dao izlaznu vrijednost 1. *Prostorno, pristranost mijenja položaj (iako ne orijentaciju) granice odluke. Kako se radi o skalarnom produktu težina i ulaznih vrijednosti, algoritam učenja perceptrona donosi odluku ako i samo ako skup učenja nije linearno odvojiv. To znači da se na temelju skupa za učenje pronalazi hiperravnina koja*

će uspješno razdvojiti dvije klase primjera ovisno o granici b. Ako vektori nisu linearno odvojivi, učenje nikad neće doći do točke u kojoj su svi vektori klasificirani pravilno. Najpoznatiji primjer nemogućnosti perceptrona da riješi probleme s linearno nerazdvojnim vektorima je Boolov XOR problem ([33], [12]).

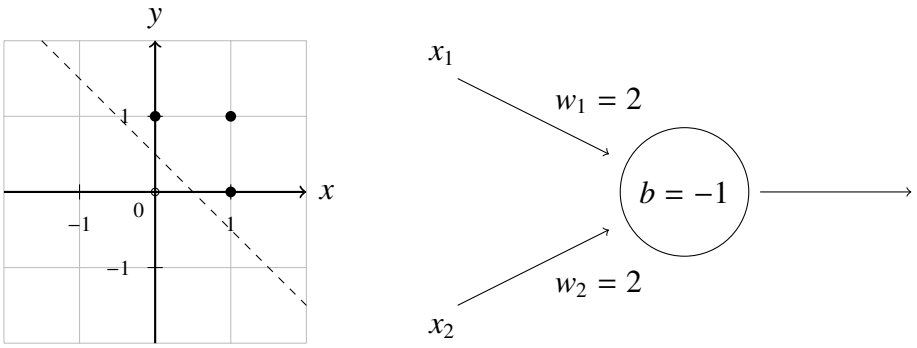
Sljedeće slike pokazuju kako se AND i OR operatori mogu prikazati pomoću jednog perceptrona, isto kao i NOT operator, što će također biti objašnjeno, dok za XOR to nije moguće.

x_1	x_2	**AND**	
0	0	0	$b < 0$
0	1	0	$b + w_2 < 0$
1	0	0	$b + w_1 < 0$
1	1	1	$b + w_1 + w_2 \geq 0$



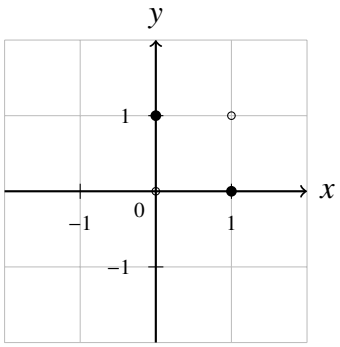
Slika 3.1: Promatra li se perceptron s dva ulaza x_1 i x_2 kojeg se želi naučiti da točno riješi problem AND, lako se dolazi do dobrih vrijednosti za težine w_1 i w_2 te pristranost b. To se postiže uvrštavajući ulazne vrijednosti u jednadžbu (3.1), gdje je funkcija f AND operator. Dobivaju se 4 nejednadžbe navedene u tablici, a jedno moguće rješenje je pravac $1.5x_1 + 1.5x_2 = 2$, tj. $w_1 = w_2 = 1.5$ i $b = -2$.

x_1	x_2	**OR**	
0	0	0	$b < 0$
0	1	1	$b + w_2 \geq 0$
1	0	1	$b + w_1 \geq 0$
1	1	1	$b + w_1 + w_2 \geq 0$



Slika 3.2: Slično kao i za AND problem, lako se dolazi do jednog mogućeg rješenja koje točno rješava problem OR, a to je $w_1 = w_2 = 2$ i $b = 1$.

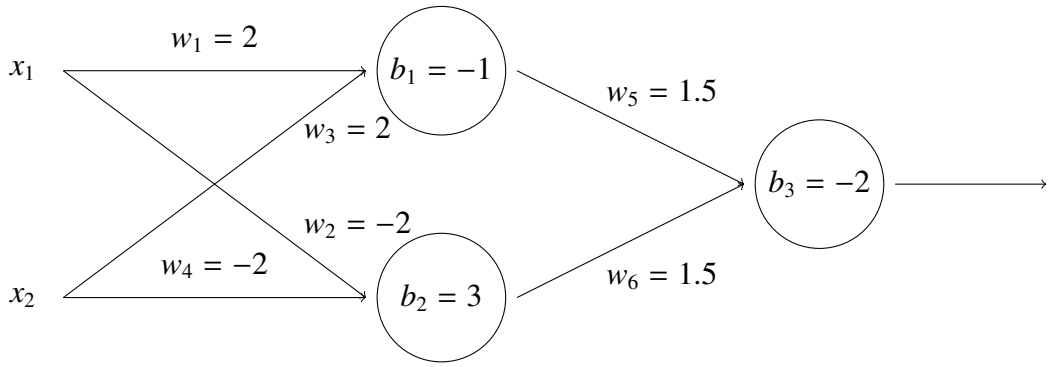
Za unarni operator NOT lako bi se pokazalo da su $w_1 = -1$ i $b = 0$ dobre vrijednosti parametara da perceptron daje točno rješenje. Do problema se dolazi kada se želi to isto napraviti za XOR problem. Kako je x_1 XOR $x_2 = 1$ ako i samo $x_1 \neq x_2$, u koordinatnom sustavu to izgleda ovako:



Slika 3.3: Koordinatni prikaz XOR operatora

Može se primijetiti da se za takvi problem ne može pronaći pravac koji točno razdvaja dvije klase primjera. No, problem XOR rješiv je ako se u mrežu doda još jedan sloj s dva perceptrona. Rješenje se dobije kombinirajući prethodna rješenja za AND, OR i NOT

operator, te činjenicu da je x_1 *XOR* x_2 $=$ (x_1 *OR* x_2) *AND* ($\overline{x_1}$ *OR* $\overline{x_2}$)*, gdje* $\overline{x}$ *označava negirani x.*



Slika 3.4: Perceptronska mreža za XOR problem - prvi neuron u skrivenom sloju daje rješenje za x_1 OR x_2, dok drugi skriveni neuron daje rješenje za $\overline{x_1}$ OR $\overline{x_2}$. Kako se težinama i pristranosti u izlaznom sloju opisuje AND operator, zaključuje se da je to dobro rješenje za x_1 XOR x_2. Potrebno je imati na umu da se nakon prvog skrivenog sloja primijenjuje granična funkcija, tj. izlazi su 0 ili 1.

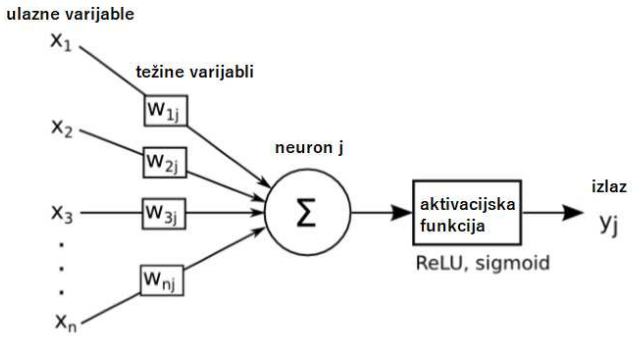
Ovo je glavna motivacija za definiranje višeslojne mreže neurona. Nadalje, moglo bi se pokazati da je proizvoljnu Booleovu funkciju moguće prikazati višeslojnom mrežom perceptrona. Postoji i slična tvrdnja koja je poznata kao univerzalni aproksimacijski teorem i kao rezultat daje činjenicu da je svaki problem modeliran određenom funkcijom moguće aproksimirati neuronskom mrežom s jednim skrivenim slojem.

Kako ljudski mozak ne funkcionira tako, tj. izlazi mu nisu 0 ili 1, za model neuronske mreže morali su se razmatrati realni brojevi kao ulazne vrijednosti, a za izlaze su potrebne realne predikcije. Upravo je to razlika između perceptrona i umjetnog neurona, perceptron kao izlaz daje 0 ili 1, dok neuron daje realnu vrijednost u intervalu $[0, 1]$. Dakle, neuronska mreža, tj. skup neurona podijeljen u slojeve, funkcija je koja danim ulazom, predviđa određeni izlaz.

Svaki je neuron realna funkcija

$$y_j = f_j = \Phi(\langle w_j, x \rangle + b_j),$$

gdje $x = (x_1, \ldots, x_d)$ predstavlja realni vektor ulaza, $w_j = (w_{j,1}, \ldots w_{j,d})$ realni vektor težina, b_j realnu pristranost. Pristranost je objašnjena kao granica nakon koje se neuron aktivira, tj. šalje svoj signal dalje sljedećim neuronima. Konačno, sa Φ definirana je aktivacijska funkcija.



Slika 3.5: Umjetni neuron gdje je $\Sigma = \langle w_j, x \rangle + b_j$

Neke od aplikacijskih funkcija koje se koriste u treniranju neuronske mreže jesu:

- sigmoidalna ili logistička funkcija

$$\Phi(x) = \frac{1}{1 + e^{-x}}$$

- hiperbolički tangens

$$\Phi(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

- obična granična funkcija

$$\Phi_\beta(x) = \mathbb{1}_{x \geq \beta}$$

- ReLU funkcija

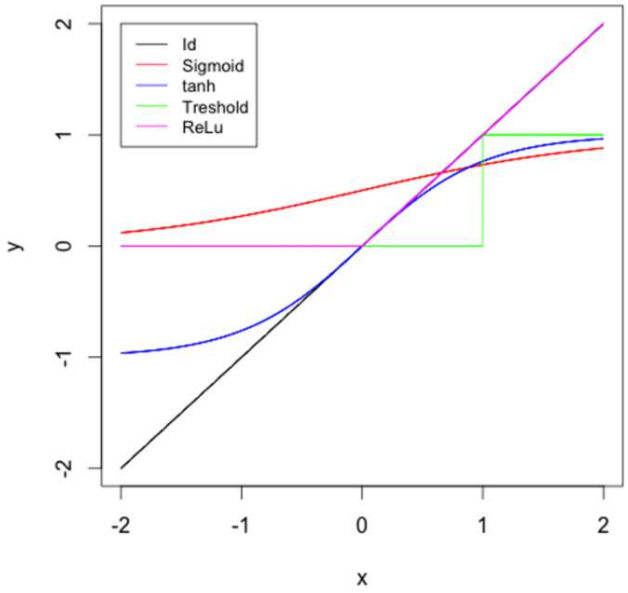
$$\Phi(x) = \max(0, x)$$

Povijesno gledano, sigmoid je bio uglavnom korištena aktivacijska funkcija jer je diferencijabilna i omogućava zadržavanje vrijednosti u intervalu $[0, 1]$. Ipak, ta je funkcija problematična jer je njen gradijent vrlo blizu 0 kada $|x|$ nije blizu 0. Kod neuronskih mreža s većim brojem slojeva, to izaziva probleme u algoritmu procjene parametara. Zbog toga je sigmoidna funkcija zamijenjena ReLU funkcijom koja nije diferencijabilna u 0, ali u praksi to i nije problem jer je vjerojatnost unosa jednakog 0 općenito nula. Upotreba ove funkcije omoguća brže treniranje mreže, a razlog je taj što se koristeći sigmoid ili tanh mogu dobiti gradijenti blizu nule, a to usporava optimizaciju metodom gradijentnog spusta. S druge strane, gradijent funkcije ReLU nije blizu nule za bilo koje pozitivne vrijednosti, što

pomaže optimizaciji da brže konvergira. Funkcija ReLU i njen gradijent jednaki su 0 za negativne vrijednosti, te se u ovom slučaju ne mogu dobiti nikakve informacije. Zato se savjetuje dodavanje male pozitivne pristranosti kako bi se osiguralo da je svaka jedinica aktivna. Na primjer,

$$\Phi(x) = \max(x, 0) + \alpha \min(x, 0),$$

gdje je α ili fiksirana mala pozitivna vrijednost, ili parametar za procijeniti.



Slika 3.6: Aktivacijske funkcije

Nadalje, višeslojna neuronska mreža je struktura sastavljena od nekoliko skrivenih slojeva neurona gdje izlazi jednog sloja neurona postaju ulaz neurona sljedećeg sloja. Na posljednjem sloju, koji se naziva izlazni sloj, najčešće se primijenjuje drugačija aktivacijska funkcija nego za skrivene slojeve, a ovisi o vrsti problema, koji može biti regresija ili klasifikacija. Za binarnu klasifikaciju izlaz predstavlja vjerojatnost $P(Y = 1|X)$, a s obzirom da je ta vrijednost u $[0, 1]$, općenito se razmatra sigmoidalna funkcija kao aktivacijska funkcija. Za klasifikaciju s više klasa, izlazni sloj sadrži jedan neuron po klasi i, koji daje vjerojatnost $P(Y = i|X)$. Zbroj svih ovih vrijednosti mora biti jednak 1. Općenito se koristi funkcija softmax

$$\gamma(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}.$$

Prije opisa algoritma učenja neuronske mreže, navodi se teorem koji pokazuje da se svaku ograničenu funkciju $\mathbb{R}^d \mapsto \mathbb{R}$ može aproksimirati za bilo koju preciznost neuronskom mrežom sa samo jednim skrivenim slojem.

Teorem 3.2.2. *Neka je Φ ograničena, neprekidna i neopadajuća (aktivacijska) funkcija. Neka je K_d neki kompaktni podskup na $\mathbb{R}^d$ i $C(K_d)$ skup neprekidnih funkcija na K_d. Neka je $f \in C(K_d)$. Tada za svaki $\epsilon > 0$ postoji $N \in \mathbb{N}$, realni brojevi v_i, b_i i vektori $w_i \in \mathbb{R}^d$ takvi da, ako se definira*

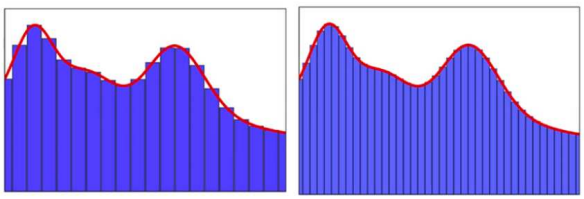
$$F(x) = \sum_{i=1}^{N} v_i \Phi(< w_i, x > + b_i),$$

tada je

$$\forall x \in K_d, |F(x) - f(x)| \leq \epsilon.$$

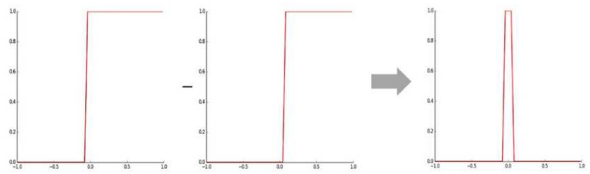
Ovaj teorem tvrdi da neuronska mreža sa samo jednim skrivenim slojem koji sadrži ograničeni broj neurona može aproksimirati neprekidne funkcije na kompaktnim podskupovima od $\mathbb{R}$. Iako teorijski zanimljiva činjenica, u praksi ne pokazuje dobre rezultate jer skriveni sloj može sadržavati jako veliki broj neurona. No, zbog ovog se teorema zna da je uvijek moguće pronaći i duboku neuronsku mrežu koja će približiti bilo koji složeni odnos ulaza i izlaza. Moć dubokog učenja (učenje neurnskim mrežama s više od jednog skrivenog sloja) leži upravo u dubini, tj. u većem broju skrivenih slojeva koji ubrzavaju i optimiziraju učenje.

Primjer 3.2.3. *Slijedi intuitivni dokaz univerzalnog aproksimacijskog teorema na primjeru realne funkcije s jednodimenzionalnim realnim ulazom x.[19] Neka je crvenom krivuljom prikazan stvarni odnos ulaza x i izlaza y. Da bi se aproksimirao prikazani odnos, jedna od ideja jest razbiti funkciju na više manjih dijelova i svaki od njih predstaviti nekom jednostavnijom funkcijom. Kombinirajući niz takvih jednostavnih funkcija, u ovom slučaju su to plavi stupovi na slici, može se aproksimirati odnos između x i y. Ključno je da ne treba uzimati komplicirane funkcije za aproksimaciju, već se uzmu one jednostavne te se one kombiniraju duž domene. Što je više manjih dijelova, to će aproksimacija biti bolja.*



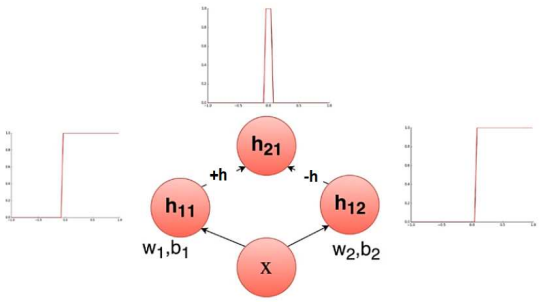
Slika 3.7: Crvenom bojom prikazan je stvarni odnos između ulaza i izlaza, a plavom su bojom označeni stupovi kojima se aproksimira stvarna funkcija.

Neka je sigmoid aktivacijska funkcija s dovoljno oštrim nagibom, što se postiže većim vrijednostima težina. Pomakne li se jednu funkciju ulijevo (mijenjanjem pristranosti b), a drugu udesno i oduzmu li se te dvije funkcije, dobiva se oblik tornja kakav se traži za aproksimaciju dane funckije.



Slika 3.8: Razlika dvije funckije sigmoid s različitim pomakom

Poslaže li se više takvih tornjeva u niz, sumirajući ih, može se aproksimirati bilo koja funkcija. Neuronska mreža koja opisuje konstrukciju takvih tornjeva dana je slikom (3.9). Tako se dobije jedan blok koji se sastoji od tri neurona, dva u skrivenom sloju i jedan u izlaznom sloju, i njime se aproksimira tražena funkcija. Definira li se više skrivenih slojeva kao u istoj slici i ukoliko ih se posumira, tj. poveže s izlaznim neuronom točno definiranim težinama koje predstavljaju visinu tornja, dobiva se aproksimacija tražene funkcije duž cijele domene.



Slika 3.9: Primijene li se na ulaz x dvije sigmoidalne funkcije s određenom težinom i pristranošću te ako se izlazi tako dobivena dva neurona kombiniraju težinama $+1$ i -1, to je isto kao da se ta dva izlaza oduzmu pa se dobije toranj visine 1. S ciljem da toranj bude neke druge visine, umjesto $+1$ i -1, uzmu se za težine pozitivna i negativna vrijednost te visine.

Ovim je primjerom ilustriran način kako se jednim skrivenim slojem može aproksimirati proizvoljna funkcija. (Formalni dokaz može se pronaći na [21].)

Dalje slijedi matematička formulacija višeslojne neuronske mreže s L skrivenih slojeva. Za $k = 1, \ldots, L$ (skriveni slojevi), neka je

$$a^{(k)}(x) = b^{(k)} + W^{(k)} h^{(k-1)}(x)$$
$$h^{(k)} = \Phi(a^{(k)}(x)).$$

Za $k = L + 1$ (izlazni sloj)

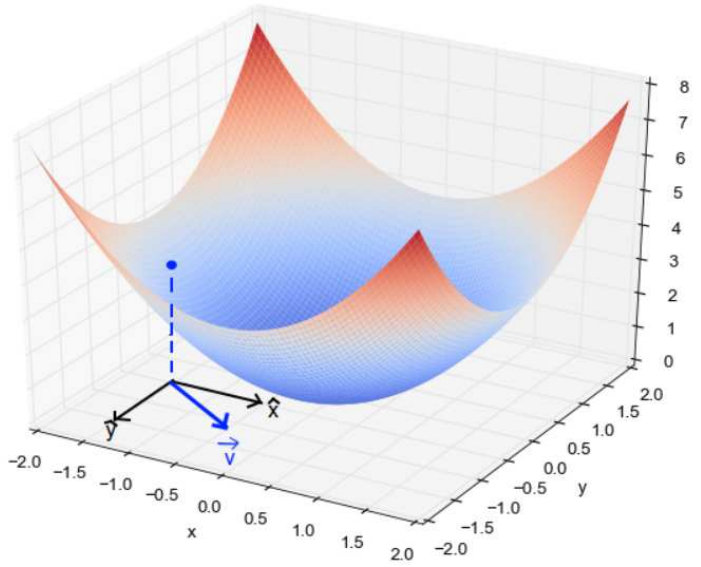
$$a^{(L+1)}(x) = b^{(L+1)} + W^{(L+1)} h^{(L)}(x)$$
$$h^{(L+1)} = \psi(a^{(L+1)}(x)) =: f(x, \theta),$$

gdje je Φ aktivacijska funkcija za skrivene slojeve, a ψ aktivacijska funkcija izlaznog sloja. U svakom koraku $W^{(k)}$ je matrica težina u kojoj je broj redaka broj neurona u k-tom sloju, a broj stupaca je broj neurona u $(k-1)$-om sloju. Ovakva arhitektura mreže naziva se *feed-forward* arhitekturom, ili arhitekturom širenja naprijed, što znači da je svaki sloj neurona povezan samo sa slojem koji je njegov neposredni prethodnik, tj. u mreži nema petlji.

Parametri mreže (težine i pristranosti) su numeričke vrijednosti koje treba prilagoditi učenjem mreže s već poznatim skupom podataka, kao u svakom drugom problemu nadziranog strojnog učenja. Konačni rezultat bit će model izgrađen iz tih podataka, sposoban predvidjeti buduće uzorke. Procjena parametara se postiže minimiziranjem funkcije troška algoritmom gradijentog spusta. Promatrajući diferencijabilnu funkciju, gradijent govori koliku promjenu u izlazu može uzrokovati mala promjena u ulazu ili u težinama. Funkcija troška koja će se ovdje promatrati bit će diferencijabilna pa će takav postupak biti moguć.

Računajući gradijent funkcije troška, dobiva se vektor koji daje smjer u kojem se funkcija troška najbrže povećava, tako da bi se algoritam trebao kretati u suprotnom smjeru ako funkciju troška pokušava svesti na minimum.

Primjer 3.2.4. *Ako se zamisli mreža s dva parametra, moglo bi se prikazati funkciju troška u tri dimenzije (XY ravnina za vrijednosti parametara, Z ravnina za vrijednost troška). Gradijent bi u ovom slučaju bio vektor s dvije komponente (jedna na X osi i jedna na osi Y) u ravnini XY u smjeru vrijednosti parametara kojima bi se funkcija troška najbrže povećavala. Njegova apsolutna vrijednost je veća što je strmiji nagib tangente u točki u kojoj je izračunat. Prema tome, što je algoritam bliže minimumu funkcije troška, to će biti manja apsolutna vrijednost gradijenta (u minimumu je nagib jednak* 0*). Dakle, ideja je pomaknuti se u suprotnom smjeru od gradijenta, pokušavajući dostići globalni minimum:*



Slika 3.10: Primjer

Funkcija troška u neuronskoj mreži puno je složenija pa izračun vektora gradijenta nije trivijalan. Obično ga je prilično komplicirano izračunati zbog velikog broja parametara i njihovog rasporeda u više slojeva.

Da bi se postupak olakšao, koristi se algoritam za izračunavanje gradijenta poznat kao širenje unatrag (en. *back propagation*). Ovaj algoritam započinje izračunavanjem parcijalnih derivacija funkcije troška obzirom na parametre izlaznog sloja, koji ne utječu na ostale parametre mreže. Ovo nije komplicirano izračunavanje zahvaljujući lančanom pravilu. Jednom kada se izračunaju parcijalne derivacije izlaznog sloja, prelazi se na prethodni sloj i izračunavaju se parcijalne derivacije funkcije troška, ali sada obzirom na parametre ovog sloja, ponovo uz pomoć lančanog pravila. Tako se postupak nastavlja unatrag, sve do početka mreže, tj. do prvog skrivenog sloja i parametara koji ga povezuju s ulaznim slojem.

Funkcija troška

Kod klasifikacijskog problema, što je ovdje slučaj, parametri se najčešće procijenjuju maksimiziranjem izglednosti (*likelihood*), ili ekvivalentno, maksimiziranjem logaritma izglednosti. To odgovara minimiziaciji negativnog logaritma izglednosti. Idealno bi bilo optimizirati mrežu tako da reprezentira željenu funkciju na cijeloj domeni, u ovom slučaju za svaku danu sliku. Kako konkretna funkcija koja se aproksimira mrežom nije poznata, potrebno je prikupiti dovoljan broj primjera za treniranje i pomoću njih definirati funkciju

troška. Pomoću tog skupa aproksimira se cijela funkcija, tj. procijenjuju se parametri tako da konačna mreža za odgovarajuće primjere za treniranje daje točne izlaze.

Ako se radi o klasifikacijskom problemu s K klasa, tada je izlaz višeslojne neuronske mreže vektor

$$f(x) = \begin{pmatrix} P(y = 1|x) \\ \vdots \\ P(y = K|x) \end{pmatrix}.$$

Neka je s θ označen vektor parametara kojeg se želi procijeniti. Funkcija troška je tada jednaka:

$$\ell(f(x), y) = \sum_{k=1}^{K} \mathbb{1}_{y=k} \log P_\theta(y = k|X), \tag{3.2}$$

gjde je s $P_\theta(y = k|X)$ označena vjerojatnost da je $y = k$, za neku klasu k. Neka je

$$(f(x))_y = \sum_{k=1}^{K} \mathbb{1}_{y=k} (f(x))_k, \tag{3.3}$$

gdje je $(f(x))_k$ k-ta komponenta od $f(x)$, tj. $(f(x))_k = P(Y = k|x)$. Tada je

$$-\log(f(x))_y = -\sum_{k=1}^{K} \mathbb{1}_{y=k} \log(f(x))_k = \ell(f(x), y). \tag{3.4}$$

Ako postoji $n \in \mathbb{N}$ primjera za treniranje $(X_i, y_i)_{1 \le i \le n}$ i pomoću njih se računa funkciju troška, tada je ona jednaka:

$$L_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(X_i, \theta), y_i). \tag{3.5}$$

"Mini-batch" i regularizacija kao metoda optimizacije gradijentnog spusta

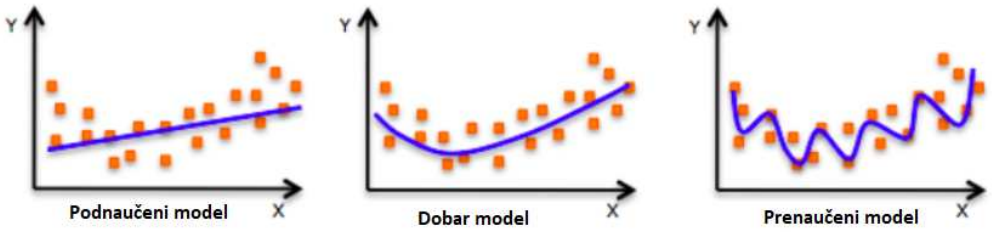
Kako bi se minimizirala funkcija troška $L_n(\theta)$, koristi se metoda "mini-batch" gradijentnog spusta širenjem unatrag. Postupak je sljedeći:

- Inicijalizacija: $\theta = (W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, \ldots, W^{(L+1)}, b^{(L+1)})$.
- Kroz N iteracija, za svaki primjer (X_i, y_i) iz proizvoljnog podskupa za treniranje B s $m \in \mathbb{N}$ primjera,

$$\theta = \theta - \epsilon \frac{1}{m} \sum_{i \in B} \nabla_\theta \ell(f(X_i, \theta), y_i). \tag{3.6}$$

Ono što bi se ovakvim postupkom moglo dogoditi jest da će težine θ postati "savršene" za model nad kojim su učene. To znači da ako se isti podaci nad kojima je model bio treniran ubace u model, točnost modela će biti vrlo visoka. Zbog toga može doći do problema pri klasifikaciji modela na još neviđenim podacima. Model će vjerojatno biti puno manje točnosti za ostale skupove podataka. Stoga je poželjno dobiti manje savršeni model kako bi dobio bolja predviđanja i za ostale podatke, one na kojima nije trenirao. Kada model uči, potrebno je imati dva skupa podataka, jedan za treniranje i jedan za testiranje. Cilj učenja modela neuronskih mreža je učiniti grešku na skupu za treniranje malom, ali i smanjiti razliku između greške na podacima za treniranje i greške na podacima za testiranje.

Da bi se generalizirao model, tj. osposobio se da dobro klasificira i na neviđenim podacima, moglo bi se provjeravati za koji je broj neurona u skrivenom sloju model prihvatljiv. Povećavanjem ili smanjivanjem broja neurona mijenja se kapacitet mreže koji određuje mogućnost prilagođavanja podacima. Ako je model manjeg kapaciteta, on će vjerojatno biti podnaučen, dok će model većeg kapaciteta vjerojatno biti prenaučen. Taj je problem vidljiv na donjim slikama.



Slika 3.11: Kapacitet ili kompleksnost modela uvelike utječe na njegovu točnost. Ukoliko je on premali, model će biti podnaučen, a ukoliko je prevelik, model će biti prenaučen.[15]

U slučaju prenaučenosti, što je češći problem od podnaučenosti, model se trudi naučiti i predobro klasificirati, pa uzima i šum kao relevantnu značajku, a kao rezultat daje loše ponašanje na novim podacima. Promjena kapaciteta mreže nije uvijek dobar način rješavanja problema prenaučenosti jer se ili gube podaci ili smanjuje efikasnost mreže.

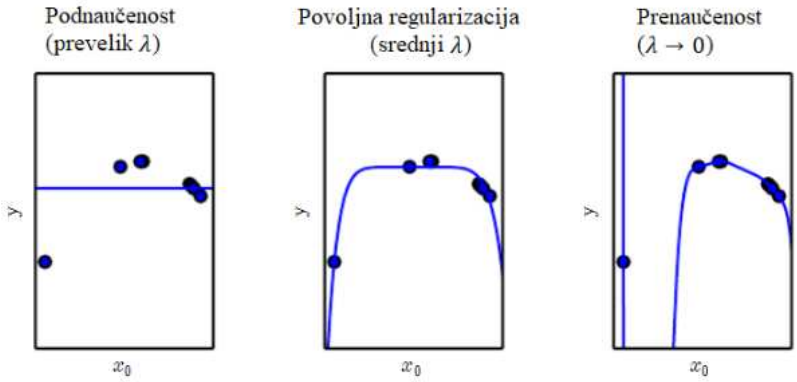
Umjesto toga, uvodi se regularizacija, tehnika koja modifikacijom algoritma za učenje bolje generalizira model. To ujedno pospješuje ponašanje modela na novim podacima. Ona penalizira vrijednosti parametara dodavanjem regularizacijskog člana funkciji troška:

$$L_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(X_i, \theta), y_i) + \frac{\lambda}{2n}\Omega(\theta). \tag{3.7}$$

Postoje dvije poznate regularizacije, L_2 i L_1 regularizacija. U L_2 regularizaciji, koja se češće koristi, Ω predstavlja funkciju

$$\Omega(\theta) = \sum_k \sum_i \sum_j \left(W_{i,j}^{(k)}\right)^2 = \sum_k \|W^{(k)}\|_F^2. \tag{3.8}$$

U gornjoj jednadžbi $\|W\|_F$ predstavlja Frobeniusevu normu matrice W, a u jednadžbi (3.7), λ predstavlja faktor regularizacije kojim se penaliziraju funkcije s visokim težinama te potiču algoritam da preferira funkcije s manjim težinama uz pretpostavku da funkcije s manjim težinama predstavljaju jednostavniji model.



Slika 3.12: Različite vrijednosti regularizacijskog faktora λ također utječu na ponašanje modela. Ukoliko je vrijednost λ prevelika, da bi se smanjila funkcija troška, težine će biti male, blizu nule. To će rezultirati podnaučenim modelom. Suprotno, ukoliko je vrijednost λ premala, to je kao da penalizacija nije ni dodana u model pa će model biti prenaučen.[9]

Ukoliko bi učenjem po postupku (3.6) došlo do prenaučenosti modela, tj. dobile bi se težine kojima model dobro klasificira podatke za treniranje, ali na podacima za testiranje radi sa smanjenom točnosti, uvođenjem regularizacijskog faktora λ postiže se pomak od tih težina koje uzrokuju prenaučenost.

Sada se dobije novi izraz za ažuriranje težina, uz regularizaciju, koji je jednak

$$\theta = \theta - \epsilon \frac{1}{m} \sum_{i \in B} \left[\nabla_\theta \ell(f(X_i, \theta), y_i) + \frac{\lambda}{2} \nabla_\theta \, \Omega(\theta) \right]. \tag{3.9}$$

Širenje unatrag (en. *backpropagation*)

Ključni korak u učenju modela neuronske mreže je promjena vrijednosti težina kroz određeni broj iteracija kako bi se postigla dobra predikcija. Jednadžbom

$$\theta = \theta - \epsilon \frac{1}{m} \sum_{i \in B} \left[\nabla_\theta \ell(f(X_i, \theta), y_i) + \frac{\lambda}{2} \nabla_\theta \, \Omega(\theta) \right].$$

dolazi se do optimalnih vrijednosti. Ovo poglavlje daje postupak dobivanja gradijenta iz gorje jednadžbe. Gradijent od $\Omega(\theta)$ iznosi

$$\nabla_{W^{(k)}} \Omega(\theta) = 2W^{(k)}, \qquad \nabla_{b^{(k)}} \Omega(\theta) = 0.$$

Za računanje $\nabla_\theta \ell(f(X_i, \theta), y_i)$, u svrhu izvoda i jednostavnosti uzima se (x, y), gdje je x neki proizvojni primjer za treniranje i y njegova oznaka.

Kao aktivacijska funkcija u izlaznom sloju uzima se *softmax* funkcija, s oznakom γ

$$\gamma(x_1, \ldots, x_K) = \frac{1}{\sum_{k=1}^{K}} (e^{x_1}, \ldots, e^{x_K}).$$

Za vektor $x = (x_1, \ldots, x_K)$, dimenzije K (uzima se dimenzija K jer se upravo izlazni sloj sastoji od K neurona, koji predstavljaju K klasa u problemu), računaju se parcijalne derivacije funkcije γ. Za $j \neq i$, vrijedi

$$\begin{aligned}
\frac{\partial \gamma(x)_i}{\partial x_j}(x_1, \ldots, x_K) &= \frac{\partial}{\partial x_j} \left(\frac{e^{x_i}}{\sum_k e^{x_k}} \right)(x_1, \ldots, x_K) \\
&= -\frac{e^{x_i}}{(\sum_k e^{x_k})^2} \cdot e^{x_j} \\
&= -\frac{e^{x_i}}{\sum_k e^{x_k}} \cdot \frac{e^{x_j}}{\sum_k e^{x_k}} \\
&= -\gamma(x)_i \cdot \gamma(x)_j.
\end{aligned}$$

Ukoliko je $i = j$, onda

$$\begin{aligned}
\frac{\partial \gamma(x)_i}{\partial x_i}(x_1, \ldots, x_K) &= \frac{\partial}{\partial x_i} \left(\frac{e^{x_i}}{\sum_k e^{x_k}} \right)(x_1, \ldots, x_K) \\
&= \frac{e^{x_i} \left(\sum_k e^{(x_k)} \right) - (e^{x_i})^2}{(\sum_k e^{x_k})^2} \\
&= \frac{e^{x_i}}{\sum_k e^{x_k}} \cdot \frac{\sum_k e^{x_k} - e^{x_i}}{\sum_k e^{x_k}} \\
&= \gamma(x)_i \cdot (1 - \gamma(x)_i).
\end{aligned}$$

U postupku dobivanja gradijenta funkcije troška, koristit će se

$$\frac{\partial \gamma(x)_i}{\partial x_j} = \begin{cases} \gamma(x)_i(1-\gamma(x)_i), & i = j \\ -\gamma(x)_i\gamma(x)_j, & i \neq j. \end{cases} \tag{3.10}$$

Ono što treba izračunati jesu gradijenti funkcije troška po izlaznim težinama i pristranošću te po skrivenim težinama i pristranostima, tj.

$$\frac{\partial \ell(f(x),y)}{\partial W_{i,j}^{(L+1)}}, \quad \frac{\partial \ell(f(x),y)}{\partial b_i^{(L+1)}}$$
$$\frac{\partial \ell(f(x),y)}{\partial W_{i,j}^{(h)}}, \quad \frac{\partial \ell(f(x),y)}{\partial b_i^{(h)}}, \text{za} 1 \leq h \leq L.$$

U postupku će se koristiti ulančano pravilo koje tvrdi: ako je $z(x) = F(a_1(x), \ldots, a_J(x))$, tada vrijedi

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial a_j} \cdot \frac{\partial a_j}{\partial x_i} = \langle \nabla F, \frac{\partial \mathbf{a}}{\partial x_i} \rangle. \tag{3.11}$$

Najprije se može zaključiti da je

$$\frac{\partial \ell(f(x),y)}{\partial W_{i,j}^{(L+1)}} = \sum_k \frac{\partial \ell(f(x),y)}{\partial (a^{(L+1)}(x))_k} \cdot \frac{\partial (a^{(L+1)}(x))_k}{\partial W_{i,j}^{(L+1)}}.$$

Prvo se odredi $\frac{\partial \ell(f(x),y)}{\partial (a^{(L+1)}(x))_k}$. Iz definicije (3.4) funkcije troška, očito je

$$\begin{aligned} \frac{\partial \ell(f(x),y)}{\partial f(x)_k} &= \begin{cases} \frac{-1}{(f(x))_k}, & y = k \\ 0, & y \neq k \end{cases} \\ &= -\frac{\mathbb{1}_{y=k}}{(f(x))_k} \end{aligned}$$

pa se to, zajedno s (3.10) iskoristi u sljedećem izvodu:

$$\begin{aligned} \frac{\partial \ell(f(x),y)}{\partial (a^{(L+1)}(x))_k} &= -\sum_j \frac{\mathbb{1}_{y=j}}{(f(x))_j} \frac{\partial (\gamma(a^{(L+1)}(x)))_j}{\partial (a^{(L+1)}(x))_k} \\ &= -\frac{1}{(f(x))_y} \frac{\partial (\gamma(a^{(L+1)}(x)))_y}{\partial (a^{(L+1)}(x))_k} \\ &= -\frac{1}{(f(x))_y} \gamma(a^{L+1}(x))_y (1-\gamma(a^{L+1}(x))_y) \cdot \mathbb{1}_{y=k} \\ &\quad + \frac{1}{(f(x))_y} \gamma(a^{L+1}(x))_k \gamma(a^{L+1}(x))_y \cdot \mathbb{1}_{y\neq k} \\ &= (-1 + (f(x))_y) \cdot \mathbb{1}_{y=k} + (f(x))_k \cdot \mathbb{1}_{y\neq k}. \end{aligned}$$

Iz toga se dobije, vektorski zapisano,

$$\nabla_{a^{(L+1)}(x)} \ell(f(x), y) = f(x) - e(y), \tag{3.12}$$

gdje je $e(y)$ vektor kojemu je i-ta komponenta jednaka 1 ako je $y = i$. Konačno, kako je $a^{(L+1)}(x) = b^{(L+1)} + W^{(L+1)}h^{(L)}(x)$, vrijedi

$$\frac{\partial(a^{(L+1)}(x))_k}{\partial W_{i,j}^{(L+1)}} = (h^{(L)})_j \cdot \mathbb{1}_{i=k},$$

pa je onda

$$\frac{\partial \ell(f(x), y)}{\partial W_{i,j}^{(L+1)}} = \sum_k \left((-1 + (f(x))_y) \cdot \mathbb{1}_{y=k} + (f(x))_k \cdot \mathbb{1}_{y\neq k}\right) \cdot (h^{(L)})_j \cdot \mathbb{1}_{i=k}.$$

Tako se dolazi do

$$\nabla_{W^{(L+1)}} \ell(f(x), y) = (f(x) - e(y))(h^{(L)})^t \tag{3.13}$$

Za $b^{(L+1)}$, zbog $\frac{(a^{(L+1)}(x))_k}{\partial b_j^{(L+1)}} = \mathbb{1}_{k=j}$, zaključuje se da je

$$\nabla_{b^{L+1}} \ell(f(x), y) = f(x) - e(y). \tag{3.14}$$

To isto treba dobiti i za svaki skriveni sloj. Neka je k broj nekog skrivenog sloja. Najprije se vidi da je

$$\frac{\partial \ell(f(x), y)}{\partial W_{i,j}^{(k)}} = \frac{\partial \ell(f(x), y)}{\partial a^{(k)}(x)_i} \cdot \frac{\partial a^{(k)}(x)_i}{\partial W_{i,j}^{(k)}},$$

što je, zbog jednakosti $a^{(k)}(x)_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} \cdot h_j^{(k-1)}(x)$, jednako

$$\frac{\partial \ell(f(x), y)}{\partial W_{i,j}^{(k)}} = \frac{\partial \ell(f(x), y)}{\partial a^{(k)}(x)_i} \cdot h_j^{(k-1)}(x). \tag{3.15}$$

Sada je potrebno odrediti $\frac{\partial \ell(f(x),y)}{\partial a^{(k)}(x)_i}$.

$$\frac{\partial \ell(f(x), y)}{\partial a^{(k)}(x)_i} = \frac{\partial \ell(f(x), y)}{\partial h^{(k)}(x)_i} \cdot \Phi'(a_j^{(k)}(x)), \tag{3.16}$$

za aktivacijsku funkciju Φ i definiranu $h^{(k)}(x) = \Phi\left(a^{(k)}(x)\right)$. Još je potrebno odrediti $\frac{\partial \ell(f(x),y)}{\partial h^{(k)}(x)_i}$.

$$\begin{aligned}\frac{\partial \ell(f(x), y)}{\partial h^{(k)}(x)_i} &= \sum_j \frac{\partial \ell(f(x), y)}{\partial a_j^{(k+1)}} \cdot \frac{\partial a_j^{(k+1)}}{\partial h^{(k)}(x)_i} \\ &= \sum_j \frac{\partial \ell(f(x), y)}{\partial a_j^{(k+1)}} \cdot W_{i,j}^{k+1}\end{aligned}$$

Kako algoritam računa tražene gradijente od izlaznog sloja prema prvom skrivenom sloju, u k-tom će koraku $\frac{\partial \ell(f(x),y)}{\partial a_j^{(k+1)}}$ biti već poznat.

Time se dolazi i do vektorskih zapisa dobivenih rezultata:

$$\nabla_{h^{(k)}(x)} \ell(f(x), y) = \left(W^{(k+1)}\right)^t \cdot \nabla_{a^{(k+1)}(x)} \ell(f(x), y) \tag{3.17}$$

$$\nabla_{a^{(k)}(x)} \ell(f(x), y) = \nabla_{h^{(k)}(x)} \ell(f(x), y) \odot \left(\Phi'(a_1^{(k)}(x)), \ldots, \Phi'(a_j^{(k)}(x)), \ldots\right)^t \tag{3.18}$$

$$\nabla_{W^{(k)}} \ell(f(x), y) = \nabla_{a^{(k)}(x)} \ell(f(x), y) \cdot (h^{(k-1)}(x))^t, \tag{3.19}$$

gdje je s $\odot$ označeno množenje po elementima. Dodatno, za pristranost, zbog $\frac{\partial \ell(f(x),y)}{\partial b_i^{(k)}} = \frac{\partial \ell(f(x),y)}{\partial a_i^{(k)}(x)}$, vrijedi

$$\nabla_{b^{(k)}(x)} \ell(f(x), y) = \nabla_{a^{(k)}(x)} \ell(f(x), y). \tag{3.20}$$

Time je, uz pomoć [4], izvedeno sve potrebno da se koraci algoritam kompletno definiraju.

- Inicijalizacija parametara: $\theta^{(0)} = (W^{(1,0)}, b^{(1,0)}, W^{(2,0)}, b^{(2,0)}, \ldots, W^{(L+1,0)}, b^{(L+1,0)})$.
- Za svaku od N iteracija, u r-toj iteraciji:
  - Prolaz prema naprijed: za svaki primjer $(X_i, y_i) \in B$, gdje je B neki podskup od cijelog skupa za treniranje, koriste se trenutni parametri

    $$\theta^{(r)} = (W^{(1,r)}, b^{(1,r)}, W^{(2,r)}, b^{(2,r)}, \ldots, W^{(L+1,r)}, b^{(L+1,r)})$$

    i izračunava se prediktivnu vrijednost $f(X_i, \theta^{(r)})$ sa svim međuvrijednostima

    $$\left(a^{(k)}(X_i), h^{(k)}(X_i)\right)_{1 \le k \le L+1}$$

    .
  - Širenje unatrag: za svaki primjer $(X_i, y_i) \in B$
    - računa se izlazni gradijent $\nabla_{a^{(L+1)}(x)} \ell(f(X_i), y_i) = f(X_i) - e(y_i)$
    - Za $k = L + 1$ do 1
      - računaju se gradijenti skrivenog sloja k

        $$\nabla_{W^{(k)}(X_i)} \ell(f(X_i), y_i) = \nabla_{a^{(k)}(X_i)} \ell(f(X_i), y_i) h^{(k-1)}(X_i)^t$$
        $$\nabla_{b^{(k)}(X_i)} \ell(f(X_i), y_i) = \nabla_{a^{(k)}(X_i)} \ell(f(X_i), y_i)$$

      - računaju se gradijenti prethodnog sloja

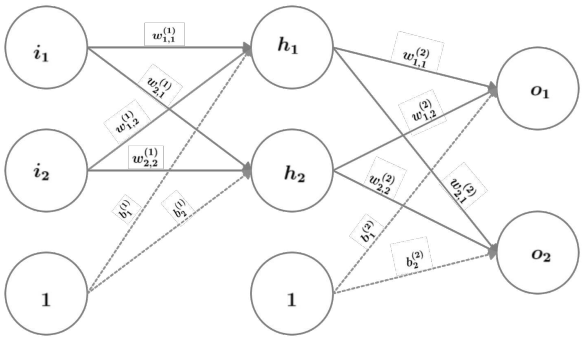
        $$\nabla_{h^{(k-1)}(X_i)} \ell(f(X_i), y_i) = \left(W^{(k)}\right)^t \nabla_{a^{(k)}(X_i)} \ell(f(X_i), y_i)$$

$$\nabla_{a^{(k-1)}(X_i)} \ell(f(X_i), y_i) = \nabla_{h^{(k-1)}(X_i)} \ell(f(X_i), y_i) \odot \left(\Phi'(a_1^{(k)}(x)), \ldots, \Phi'(a_j^{(k)}(x)), \ldots\right)^t.$$

– Na temelju dobivenih gradijenata, ažuriraju se vrijednosti parametara:

$$\theta = \theta - \epsilon \frac{1}{m} \sum_{i \in B} \left[\nabla_\theta \ell(f(X_i, \theta), y_i) + \frac{\lambda}{2} \nabla_\theta \Omega(\theta) \right].$$

Primjer 3.2.5 (Rad neuronske mreže)**.** *Neka je donjom slikom zadan primjer višeslojne neuronske mreže s jednim skrivenim slojem kod koje je ulaz dvodimenzionalni vektor, a izlaz je kao binarni klasifikator također dvodimenzionalan. Skriveni sloj, radi jednostavnosti, sadrži samo dva neurona. U ovom primjeru neće se primijeniti regularizacijski izraz u funkciji troška.*



Slika 3.13: Izgled neuronske mreže

Neka su za ulaz $i = (0.20, 0.30)$ *s izlazom* $y = 0$*, tj.* $\mathbf{y} = (1, 0)$*, početne vrijednosti parametara zadane sljedećom tablicom:*

$w_{1,1}^{(1)}$	0.15	$w_{1,1}^{(2)}$	0.25	$w_{1,2}^{(1)}$	0.20	$w_{1,2}^{(1)}$	0.15
$w_{2,1}^{(1)}$	0.25	$w_{2,1}^{(2)}$	0.30	$w_{2,2}^{(1)}$	0.30	$w_{2,2}^{(2)}$	0.45
$b_1^{(1)}$	0.30	$b_1^{(2)}$	0.55	$b_2^{(1)}$	0.55	$b_2^{(2)}$	0.65

Algoritam prvo, uz trenutne vrijednosti parametara, traži izlaznu vrijednost za zadani ulaz. U prvom se koraku dobije:

$$a_1^{(1)} = w_{1,1}^{(1)} \cdot i_1 + w_{1,2}^{(1)} \cdot i_2 = 0.47$$
$$a_2^{(1)} = w_{2,1}^{(1)} \cdot i_1 + w_{2,2}^{(1)} \cdot i_2 = 0.81.$$

Na ove se vrijednosti primijeni aktivacijska funkcija sigmoid $\Phi(x) = \frac{1}{1-e^{-x}}$ *pa su konačne vrijednosti skrivenog sloja jednake:*

$$h_1 = \Phi(a_1^{(1)}) = \frac{1}{1+e^{-0.47}} = 0.6154$$
$$h_2 = \Phi(a_2^{(1)}) = \frac{1}{1+e^{-0.81}} = 0.6921.$$

Isto se računa i za izlazni sloj, kojemu je ulaz upravo izlaz skrivenog sloja, te ispadne sljedeće:

$$a_1^{(2)} = w_{1,1}^{(2)} \cdot h_1 + w_{1,2}^{(2)} \cdot h_2 = 0.8077$$
$$a_2^{(2)} = w_{2,1}^{(2)} \cdot h_1 + w_{2,2}^{(2)} \cdot h_2 = 1.1461.$$

Da bi se dobio konačni izlaz, na dobivene se rezultate primijenjuje softmax funkcija $\gamma(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$ *i dobije se:*

$$f(i_1, i_2)_1 = o_1 = \frac{e^{a_1^{(2)}}}{e^{a_1^{(2)}} + e^{a_2^{(2)}}} = 0.0634$$
$$f(i_1, i_2)_2 = o_2 = \frac{e^{a_2^{(2)}}}{e^{a_1^{(2)}} + e^{a_2^{(2)}}} = 0.0889.$$

Sada se računa funkcija troška za dobiveni izlazni rezultat:

$$\ell(f(i_1, i_2), y) = -\mathbb{1}_{y=0} \log(o_1) - \mathbb{1}_{y=1} \log(o_2)$$
$$\ell(f(i_1, i_2), y) = -\log(o_1) = -\log(0.0634) = 2.7583.$$

Sljedeći je korak ažurirati parametre gradijentnim spustom algoritmom širenja unatrag. Kreće se od izlaznog sloja. Iz jednadžbi (3.13) i (3.14) lako se izračunaju gradijenti $\nabla_{W^{(2)}} \ell(f(i_1, i_2), y)$ *i* $\nabla_{b^{(2)}} \ell(f(i_1, i_2), y)$.

$$\begin{aligned}
\nabla_{W^{(2)}} \ell(f(i_1, i_2), y) &= (f(i_1, i_2) - e(y)) \cdot (h)^t \\
&= \begin{bmatrix} 0.0634 - 1 \\ 0.0889 - 0 \end{bmatrix} \cdot \begin{bmatrix} 0.6154 & 0.6921 \end{bmatrix} \\
&= \begin{bmatrix} -0.5764 & -0.6482 \\ 0.0547 & 0.0616 \end{bmatrix} \\
\nabla_{b^{(2)}} \ell(f(i_1, i_2), y) &= f(i_1, i_2) - e(y) \\
&= \begin{bmatrix} -0.9366 \\ 0.0889 \end{bmatrix}.
\end{aligned}$$

Time su dobiveni gradijenti funkcije troška po parametrima izlaznog sloja. Prije samog ažuriranja, izračunaju se još gradijenti i za skriveni sloj koristeći jednadžbe (3.17) i (3.20).

$$\begin{aligned}
\nabla_h \ell(f(i_1, i_2), y) &= \left(W^{(2)}\right)^t \cdot \nabla_{a^{(2)}} \ell(f(i_1, i_2), y) \\
&= \left(W^{(2)}\right)^t \cdot (f(i_1, i_2) - e(y)) \\
&= \begin{bmatrix} 0.25 & 0.15 \\ 0.30 & 0.45 \end{bmatrix}^t \cdot \begin{bmatrix} -0.9366 \\ 0.0889 \end{bmatrix} \\
&= \begin{bmatrix} -0.2075 \\ -0.1005 \end{bmatrix} \\
\nabla_{a^{(1)}} \ell(f(i_1, i_2), y) &= \nabla_h \ell(f(i_1, i_2), y) \odot \begin{bmatrix} \Phi'(a_1^{(1)}) \\ \Phi'(a_2^{(1)}) \end{bmatrix} \\
&= \left(\textit{koristi se } \Phi'(x) = \Phi(x)\,(1 - \Phi(x)) \textit{ i } \Phi(a_j^{(1)}) = h_j, \textit{ za } j \in \{1, 2\}\right) \\
&= \nabla_h \ell(f(i_1, i_2), y) \odot \begin{bmatrix} h_1(1 - h_1) \\ h_2(1 - h_2) \end{bmatrix} \\
&= \begin{bmatrix} -0.2075 \cdot 0.6154 \cdot (1 - 0.6154) \\ -1.005 \cdot 0.6921 \cdot (1 - 0.6921) \end{bmatrix} \\
&= \begin{bmatrix} -0.0491 \\ -0.0214 \end{bmatrix}
\end{aligned}$$

Gradijenti funkcije troška po parametrima skrivenog sloja su:

$$\begin{aligned}
\nabla_{W^{(1)}} \ell(f(i_1, i_2), y) &= \nabla_{a^{(1)}} \ell(f(i_1, i_2), y) \cdot i^t \\
&= \begin{bmatrix} -0.0098 & -0.0344 \\ -0.0043 & -0.0149 \end{bmatrix} \\
\nabla_{b^{(1)}} \ell(f(i_1, i_2), y) &= \nabla_{a^{(1)}} \ell(f(i_1, i_2), y) \\
&= \begin{bmatrix} -0.0491 \\ -0.0214 \end{bmatrix}.
\end{aligned}$$

Gradijentnim spustom, uz faktor učenja $\eta = 0.5$ *ažuriraju se svi parametri:*

$$\begin{aligned} W^{(2)} &= W^{(2)} - \eta \cdot \nabla_{W^{(2)}} \ell(f(i_1, i_2), y) \\ &= \begin{bmatrix} 0.5382 & 0.4741 \\ 0.2726 & 0.4192 \end{bmatrix} \\ b^{(2)} &= b^{(2)} - \eta \cdot \nabla_{b^{(2)}} \ell(f(i_1, i_2), y) \\ &= \begin{bmatrix} 1.0183 \\ 0.6055 \end{bmatrix} \\ W^{(1)} &= W^{(1)} - \eta \cdot \nabla_{W^{(1)}} \ell(f(i_1, i_2), y) \\ &= \begin{bmatrix} 0.1549 & 0.2172 \\ 0.2521 & 0.3075 \end{bmatrix} \\ b^{(1)} &= b^{(1)} - \eta \cdot \nabla_{b^{(1)}} \ell(f(i_1, i_2), y) \\ &= \begin{bmatrix} 0.3246 \\ 0.5607 \end{bmatrix} \end{aligned}$$

Koristeći ažurirane vrijednosti parametara, izračuna se izlazna vrijednost za zadani ulaz. Kao rezultat u prvom koraku dobije se:

$$\begin{aligned} a_1^{(1)} &= w_{1,1}^{(1)} \cdot i_1 + w_{1,2}^{(1)} \cdot i_2 = 0.5075 \\ a_2^{(1)} &= w_{2,1}^{(1)} \cdot i_1 + w_{2,2}^{(1)} \cdot i_2 = 0.8264. \end{aligned}$$

Primjenom aktivacijske funkcije sigmoid $\Phi(x) = \frac{1}{1-e^{-x}}$ *na dobivene vrijednosti dobiju se konačne vrijednosti skrivenog sloja:*

$$\begin{aligned} h_1 &= \Phi(a_1^{(1)}) = \frac{1}{1 + e^{-0.5075}} = 0.6242 \\ h_2 &= \Phi(a_2^{(1)}) = \frac{1}{1 + e^{-0.8264}} = 0.6956. \end{aligned}$$

Isto se računa i za izlazni sloj:

$$\begin{aligned} a_1^{(2)} &= w_{1,1}^{(2)} \cdot h_1 + w_{1,2}^{(2)} \cdot h_2 = 1.6840 \\ a_2^{(2)} &= w_{2,1}^{(2)} \cdot h_1 + w_{2,2}^{(2)} \cdot h_2 = 1.0673. \end{aligned}$$

Konačni izlaz je:

$$\begin{aligned} f(i_1, i_2)_1 = o_1 &= \frac{e^{a_1^{(2)}}}{e^{a_1^{(2)}} + e^{a_2^{(2)}}} = 0.6495 \\ f(i_1, i_2)_2 = o_2 &= \frac{e^{a_2^{(2)}}}{e^{a_1^{(2)}} + e^{a_2^{(2)}}} = 0.3505. \end{aligned}$$

Vrijednost funkcije troška za novi izlazni rezultat sada je:

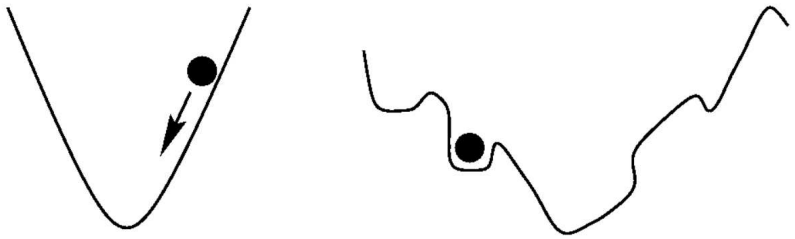
$$\ell(f(i_1, i_2), y) = -\mathbb{1}_{y=0} \log(o_1) - \mathbb{1}_{y=1} \log(o_2)$$
$$\ell(f(i_1, i_2), y) = -\log(o_1) = -\log(0.6495) = 0.4316.$$

Može se vidjeti da se vrijednost funkcije troška poprilično smanjila već nakon jednog koraka i samo jednim primjerom. Pravi algoritam ažurira parametre koristeći skup od zadanog broja n primjera, što se naziva "mini batch" metoda.

3.3 Dodatne metode optimizacije gradijentnog spusta

Momentum ili zamah

U neuronskim mrežama koristi se algoritam optimizacije gradijentnim spustom kako bi se minimizirala funkcija troška i kako bi se došlo do globalnih minimuma. U idealnom slučaju, funkcija troška izgledala bi kao na donjoj lijevoj slici.



Slika 3.14: Lijeva slika prikazuje idealni slučaj kada bi pronađeni minimum ujedno bio i globalni. Desna slika prikazuje izgledniju situaciju kada postoje lokalni minimumi u kojima algoritam može zaglaviti jer misli da je postigao globalni minimum.

U idealnom će slučaju biti pronađen globalni optimum jer ne postoji lokalni minimum u kojem bi optimizacija mogla zaglaviti. U stvarnosti je funkcija troška složenija, može se sastojati od nekoliko lokalnih minimuma i može izgledati kao na desnoj gornjoj slici.

Običnom metodom gradijentnog spusta ažuriraju se parametri samo služeći se gradijentima funkcije troška. Metoda zamaha vrši dodatnu obradu gradijenata kako bi konvergencija bila brža i točnija. Ideja za taj algoritam posuđena je iz fizike. Pusti li se kugla unutar zdjele bez trenja, ona se neće zaustaviti na dnu, već će ju stvoreni zamah odgurnuti naprijed pa će se lopta kotrljati naprijed-natrag. Pojam zamaha primijenjuje se na općeniti algoritam gradijentnog spusta tako da se gradijentu doda kretanje prethodnog koraka. Ma-

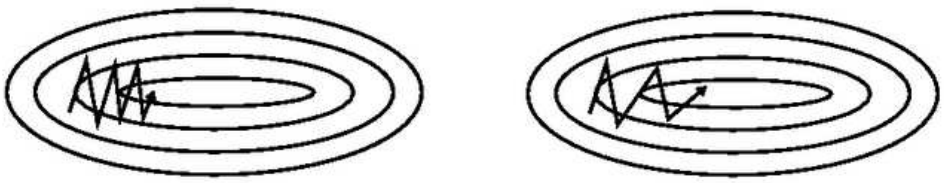
tematički je to dano izrazom

$$v_t = \gamma \cdot v_t + \eta \cdot \nabla_\theta L_n(\theta)$$
$$\theta = \theta - v_t$$

Da bi se izbjegla situacija s desne slike (3.14), u ciljnoj se funkciji koristi izraz zamaha, a to je vrijednost između 0 i 1 koja povećava veličinu koraka prema minimumu pokušavajući iskočiti iz lokalnih minimuma. Ako je vrijednost zamaha velika, stopa učenja treba biti manja. Velika vrijednost zamaha također znači da će do konvergencije brzo doći. Ali ako su i zamah i brzina učenja velikih vrijednosti, prevelikim će korakom minimum možda biti preskočen. Mala vrijednost zamaha ne može pouzdano izbjeći lokalne minimume, a također može usporiti učenje modela. Ako je vrijednost zamaha jednaka nuli, tada je dobiveni model isti kao i gradijentni spust. Ako je vrijednost zamaha jednaka jedan (i pod uvjetom da je stopa učenja mala), tada algoritam beskrajno skače naprijed-nazad poput analogije zdjele bez trenja. Obično se vrijednost zamaha postavlja na vrijednost u intervalu $[0.8, 0.9]$, tj. u primjeru, ona je poput površine s malo trenja, tako da se na kraju uspori i zaustavi.

Moguće je primijetiti dvije prednosti dodavanja zamaha u odnosu na obični gradijentni spust. Zamahom se algoritam brže kreće prema globalnom minimumu i najvjerojatnije neće zaglaviti u lokalnim minimumima (jer ga zamah može izbaciti iz lokalnog minimuma).

U osnovi, kada se koristi zamah, kugla se gura niz brdo. Ona akumulira zamah dok se kotrlja nizbrdo, postajući sve brža na svom putu (sve dok ne postigne svoju terminalnu brzinu, ukoliko postoji otpor zraka, tj. $\gamma < 1$). Ista se stvar događa s ažuriranjima parametara. Pojam zamaha povećava se za dimenzije čiji gradijenti pokazuju na iste smjerove i smanjuje ažuriranja za dimenzije čiji gradijenti mijenjaju smjerove. Kao rezultat toga, dobiva se brža konvergencija i smanjenje oscilacije. Više o temi može se pročitati u [28], [17] i [16].



Slika 3.15: Na lijevoj je slici prikazan gradijentni spust bez korištenja zamaha, a desno je zamah uključen u algoritam.

Adam optimizator

Zajedno s dodanim zamahom u algoritam gradijentnog spusta, često se u optimizaciju uključuje i Adam optimizator (više na [7]) koji je vjerojatno i optimizator u prosjeku s najboljim rezultatima. Adam optimizator koristi zamah i adaptivne stope učenja za bržu konvergenciju. Već je pokazano što predstavlja zamah pa se sada opisuju adaptivne stope učenja.

Dio intuicije za adaptivne stope učenja jest da se počne s velikim koracima i završi malim koracima, poput mini golfa. U početku se dopušta brže kretanje, a opadanjem stope učenja, koraci su sve manji, što omogućuje bržu konvergenciju.

Adam je prilagodljiva metoda brzine učenja, što znači da izračunava pojedinačne stope učenja za različite parametre. Ime je dobiveno po procjeni adaptivnog momenta, a razlog takog naziva zato što Adam koristi procjene prvog i drugog momenta gradijenta kako bi prilagodio brzinu učenja za svaku težinu neuronske mreže.

N-ti moment slučajne varijable definira se kao očekivana vrijednost te varijable na potenciju n. Formalnije, za slučajnu varijablu X:

$$m_n = E[X^n].$$

Gradijent funkcije troška neuronske mreže može se smatrati slučajnom varijablom, jer se obično procjenjuje na nekim malim slučajnim nizovima podataka. Prvi moment je sredina, a drugi moment je necentrirana varijanca (ne oduzima se sredina tijekom izračuna varijance). Da bi se procijenili momenti, u koraku t se koriste gradijenti procijenjeni na trenutnom skupu podataka $g_t = \nabla L_n(\theta_{t-1})$. Momenti su tada jednaki:

$$m_t = (1 - \beta_1)g_t + \beta_1 m_{t-1} \tag{3.21}$$

$$v_t = (1 - \beta_1)g_t^2 + \beta_2 v_{t-1}, \tag{3.22}$$

gdje su β_1 i β_2 hiperparametri algoritma koje zadaje korisnik. Obično su postavljeni na $\beta_1 = 0.9$ i $\beta_2 = 0.999$ i najčešće nitko ne postavlja te parametre drugačije. Dodatno, m_0 i v_0 inicijaliziraju se nulom. Budući da su m i v procjene prvog i drugog momenta, traži se sljedeće svojstvo:

$$E[m_t] = E[g_t]$$
$$E[v_t] = E[g_t^2].$$

Računanjem nekoliko vrijednosti od m može se vidjeti kako će se postići gornja jednakost.

$$m_0 = 0$$
$$m_1 = \beta_1 m_0 + (1 - \beta_1)g_1 = (1 - \beta_1)g$$
$$m_2 = \beta_1 m_1 + (1 - \beta_1)g_2 = \beta_1(1 - \beta_1)g_1 + (1 - \beta_1)g_2$$
$$m_3 = \beta_1 m_2 + (1 - \beta_1)g_3 = \beta_1^2(1 - \beta_1)g_1 + \beta_1(1 - \beta_1)g_2 + (1 - \beta_1)g_3.$$

"Što se više" razvija vrijednost m, manje prve vrijednosti gradijenata doprinose ukupnoj vrijednosti jer se množe s manjom vrijednosti β. Opća formula za m_t u koraku t može se zapisati kao

$$m_t = (1 - \beta_1) \sum_{i=1}^{t} \beta_1^{t-1} g_i.$$

Očekivana vrijednost od m u koraku t je tada

$$\begin{aligned} E[m_t] &= E\left[(1 - \beta_1) \sum_{i=1}^{t} \beta_1^{t-1} g_i\right] \\ &= E[g_t](1 - \beta_1) \sum_{i=1}^{t} \beta_1^{t-i} + \zeta \\ &= E[g_t](1 - \beta_1^t) + \zeta, \end{aligned}$$

gdje je ζ pristrana korekcija za procjenitelj prvog momenta. U gornjem se prvom redu koristi nova formula za razvoj od m. Zatim se svaki g_i aproksimira s g_t, zbog čega dolazi do pojave ζ.To se može izlučiti iz zbroja, jer više ne ovisi o i. U zadnjem retku koristi se poznata formula za zbroj konačnih geometrijskih nizova.

Mora se ispraviti procjenitelj, tako da je očekivana vrijednost ona koju se želi postići. Ovaj se korak obično naziva korekcija pristranosti. Konačne formule procjenitelja bit će sljedeće:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t}. \end{aligned}$$

Još preostaje koristiti te prosjeke za pojedinačno skaliranje brzine učenja za svaki parametar. Način na koji je to učinjeno u Adamu vrlo je jednostavno, za ažuriranje težina koristi se sljedeće:

$$\theta_{t+1} = \theta_t - \frac{\eta \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon},$$

gdje je η brzina učenja.

3.4 Testiranje obične neuronske mreže

Obična se neuronska mreža testira na dva načina, jedan je ručno implementiranom neuronskom mrežom, uz pomoć [30], s jednim skrivenim slojem koji se sastoji od 300 neurona.

Na prvi je sloj primijenjena aktivacijska funkcija sigmoid, dok je na izlazni sloj primijenjena funkcija γ. Učenje se odvija "mini-batch" gradijentnim spustom, gdje je u funkciju troška uključena L_2 regularizacija, i dodatno uz korištenje zamaha za ubrzanje konvergencije.

Podaci se združe i podijele na 80% podataka za treniranje i 20% podataka za validaciju, odnosno testiranje. Ono što se dobije takvom mrežom, primijenjenu na skup slika dimenzija 224×224 prolaskom u broj 25 epoha, tj. iteracija, te uz "batch" veličinu 20 jest točnost na skupu za treniranje od 80.34274193548387% i točnost na skupu za testiranje od 66.33064516129032%.

Dobiva se lošiji rezultat od primjene Laplacovog operatora, a sam algoritam se vrti duže od pola sata. Iako malo brže, na skupu slika dimenzija 96×96, također se dobiva lošiji rezultat. Točnost na skupu za treniranje je 75.35282258064517%, a na skupu za testiranje 66.73387096774194%.

Ova se metoda ne pokazuje praktičnom. Općenito nije praktično primjenjivati obične neuronske mreže na problem klasifikacije slika. Slike su, u ovom slučaju, reprezentirane dvodimenzionalnim matricama, a neuronska mreža kao ulaz prima vektor. Zato je sliku potrebno rastegnuti u vektor čime se vjerojatno gube neke prostorne značajke.

Osim ručno implementirane mreže, ovdje se još testira *MLPClassifier* iz biblioteke *sklearn.neural_network* na slikama dimenzija 96×96. On služi za usporedbu i dokaz kako obične neuronske mreže uistinu nisu dobar model za klasifikaciju slika.

```
model = sklearn.neural_network.MLPClassifier(hidden_layer_sizes=(300, ), activation='relu', solver='adam',
                                              alpha=0.0001, batch_size=10, learning_rate='constant', learning_rate_init=0.001, power_t=0.5,
                                              max_iter=1000, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9,
                                              nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08,
                                              n_iter_no_change=10)
```

Slika 3.16: MLPClassifier

Rezultati testiranja su sljedeći:

```
-- Training data --
Accuracy: 98.69
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      1001
           1       0.98      1.00      0.99       983

    accuracy                           0.99      1984
   macro avg       0.99      0.99      0.99      1984
weighted avg       0.99      0.99      0.99      1984

Confusion Matrix:
[[977  24]
 [  2 981]]
```

Slika 3.17: Rezultati na skupu za treniranje

```
---- Test data ----
Accuracy: 67.94
Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.57      0.64       248
           1       0.65      0.79      0.71       248

    accuracy                           0.68       496
   macro avg       0.69      0.68      0.68       496
weighted avg       0.69      0.68      0.68       496

Confusion Matrix:
[[141 107]
 [ 52 196]]
```

Slika 3.18: Rezultati na skupu za testiranje

Ovime se pokazalo da se običnom višeslojnom neuronskom mrežom primijenjenom na skup u vektor rastegnutih slika ne dobivaju dobri rezultati, a nikako bolji od varijance Laplasijana slike na slikama istih dimenzija. Zbog loših rezultata i zaključka da obična višeslojna neuronska mreža nije praktična za klasifikaciju slika, razvijena je konvolucijska neuronska mreža, konkretno za rješavanje problema klasifikacije slika, ili obrade slika općenito.

3.5 Konvolucijska neuronska mreža

Konvolucijska neuronska mreža jedan je od tipova dubokih neuronskih mreža koja se sastoji od ulaznog i izlaznog sloja te određenog, po volji, broja skrivenih slojeva. To je poseban slučaj dubokih neuronskih mreža za koji je pokazano da daje dobre i impresivne rezultate u prepoznavanju slika. Do boljih rezultata dovele su neke bitne promjene u arhitekturi poput lokalne povezanosti, prostorne invarijantnosti te hijerarhije značajki. Kako se slike sastoje od elemenata lociranih na različitim dijelovima slike, neuroni mreže ne moraju biti povezani sa svim jedinicama slike da bi pronašli značajne uzorke. Neuroni konvolucijske neuronske mreže povezani su samo s malim brojem jedinica u prostorno lokaliziranom području ulaza zvanom receptivno polje. To omogućuje neuronima da se usredotoče na lokalne značajke, a ne na globalne.

Mreža mora proizvesti slične izlazne vrijednosti iz sličnih obrazaca ulaza, bez obzira gdje se nalaze na slici. Konvolucijska neuronska mreža implementira ovo svojstvo dijeljenjem parametara između različitih neurona. Uzorci na slikama obično se mogu podijeliti u hijerarhiju značajki, s osobinama nižih razina koje se mogu grupirati radi stvaranja značajki na visokoj razini. Korištenjem više slojeva, konvolucijska neuronska mreža može automatski izdvojiti i naučiti ovu hijerarhiju značajki za prepoznavanje uzoraka. Na primjer, prvi konvolucijski sloj može naučiti osnovne elemente poput rubova, a drugi konvolucijski sloj može naučiti uzorke sastavljene od osnovnih elemenata naučenih u prethodnom sloju. I tako sve dok ne nauči vrlo složene obrasce. To omogućuje konvolucijskim neuronskim mrežama da učinkovito nauče sve složenije i apstraktnije vizualne koncepte.

Tradicionalne neuronske mreže koriste potpuno povezane slojeve, gdje je svaki neuron povezan sa svakim neuronom u prethodnom sloju. To može lako dovesti do umrežavanja s tisućama parametara koji se procjenjuju. Iako se one mogu koristiti za učenje značajki kao i za klasificiranje podataka, same po sebi nisu praktične za učenje nad slikama jer bi za to bio potreban veoma velik broj neurona, čak i u plitkoj arhitekturi, zbog vrlo velikih ulaznih veličina slika, gdje je svaki piksel relevantna varijabla. Na primjer, potpuno povezani sloj za (malu) sliku samo u sivim tonovima veličine 100×100 ima 10000 težina za svaki neuron u drugom sloju. Tim se brojem još može i raditi, ali ne dobro ako je cilj dodati više slojeva i neurona, što u dubokom učenju jest. Kako potpuno povezana mreža počinje rasti, ogroman broj parametara može brzo dovesti i do pretreniranja.

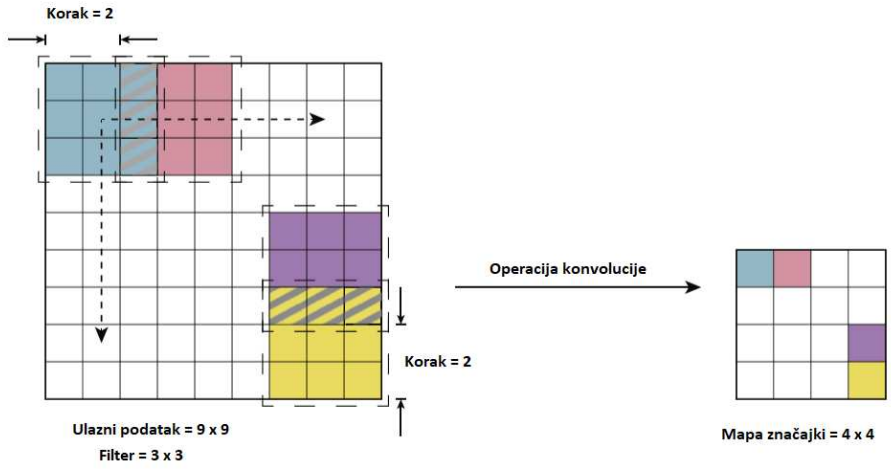
Konvolucija kao operacija donosi rješenje ovog problema jer smanjuje broj slobodnih parametara, omogućavajući mreži da bude dublja s manje parametara. Na primjer, bez obzira na veličinu slike, kovolucijske mreže rade s prozorima, npr. veličine 5×5, od kojih svaki ima iste zajedničke težine pa zahtijevaju samo 25 parametara koje treba naučiti da rezultiraju dobrim izlaznim vrijednostima. Korištenjem reguliranih težina preko manje parametara, izbjegavaju se problemi s nestabilnim gradijentima u tradicionalnim neuronskim mrežama. Korištenjem [32] opisuje se arhitektura i rad mreže.

Općenito, konvolucijski slojevi djeluju na trodimenzionalnim tenzorima, zvanim karakteristične mape, s dvije prostorne osi visine i širine te trećom osi, koja određuje broj kanala, a još se naziva i dubina. Za RGB sliku u boji, dimenzija osi dubine je 3, jer slika ima tri kanala: crveni, zeleni i plavi. Ulaz u konvolucijskoj neuronskoj mreži je tenzor oblika (broj slika) $\times$ (visina slike) $\times$ (širina slike) $\times$ (dubina slike). Konvolucijka mreža može biti sastavljena od nekoliko različitih slojeva. U sljedećim odjeljcima prikazuju se njih četiri: konvolucijski sloj, nelinearni sloj, sloj udruživanja i potpuno povezani sloj. Radi jednostavnosti, promatra se slučaj u kojem slojevi obrađuju dvodimenzionalne ulazne podatke (npr. slika s dubinom 1, tj. samo u sivim tonovima).

Konvolucijski sloj

U neuronskim mrežama svaki neuron prima ulaz s određenog broja mjesta u prethodnom sloju. U potpuno povezanom sloju obične neuronske mreže svaki neuron prima ulaz iz svakog elementa prethodnog sloja. U konvolucijskom sloju, neuroni primaju ulaz samo iz ograničenog dijela prethodnog sloja. Taj je dio obično kvadratnog oblika (npr. veličine 5×5) i naziva se njegovo receptivno polje. Zadatak konvolucijskog sloja, koji sadrži skup filtera, jest izvođenje operacije konvolucije između tih filtera i ulaznog sloja da bi se stvorile mape značajki. Filteri su obično kvadratne mreže diskretnih vrijednosti čiji parametri najčešće određuju neki uzorak. Operacija konvolucije smjestit će filter na lijevi gornji dio slike i rezultirat će zbrojem umnoška po elementitima između parametara filtera i odgovarajuće mreže ulaza, tj. receptivnog polja. Zatim će operacija konvolucije pomaknuti filter udesno i računati isto u novom položaju. Ovakva je operacija implementirana slijeva na desno, od vrha prema dnu preko ulaza i omogućuje primjenu filtera na svakoj poziciji slike.

Mapa značajki pohranjuje rezultat operacije konvolucije također u obliku mreže, tj. matrice što je bitno jer se operacije konvolucije sljedećih slojeva ponovo provode na isti način.



Slika 3.19: Operacija konvolucije u konvolucijskom sloju

Težine neurona su analogne parametrima filtera. Svaki neuron povezan s elementom u mapi značajki povezan je s uređenim područjem prethodnog sloja na takav način da je pokriven cijeli ulaz (”lokalna povezanost”). Unutar iste mape značajki, svaki neuron dijeli iste vrijednosti težina, na temelju pretpostavke da će se značajka pojaviti na svakoj ulaznoj poziciji (”prostorna invarijancija”). Kao rezultat ovog dizajna, cijela se slika prođe po istom obrascu. Nadalje, svaki se filter pohranjuje samo u jednom uzorku. Skeniranje slike samo po jednom uzorku vjerojatno bi rezultiralo ograničenom mrežom. Da bi se riješio ovog ograničenja, konvolucijski sloj mora imati nekoliko filtera, od kojih svaki stvara jednu dvodimenzionalnu mapu značajki. Nakon dobivanja različitih mapa značajki, sve se spajaju zajedno i to postaje konačni trodimenzionalni izlaz konvolucijskog sloja sa svim mapama značajki.

Slično kao kod običnih neuronskih mreža, parametri svakog filtera (težine zajedničkog neurona) uče u fazi učenja. Ovaj postupak učenja uključuje slučajnu inicijalizaciju parametara filtera na početku, koji se zatim mijenjaju kroz iteracije metodom gradijentnog spusta.

Da bi konvolucijski sloj uspješno riješio problem kojim se bavi, on mora kombinirati dokaze iz različitih mapa značajki s jednim filtrom za traženje prisutnosti svih elemenata potrebnih za odluku. Zato konvolucijski sloj ima različite hiperparametre u usporedbi sa slojevima obične neuronske mreže. To su:

- broj filtera - označava broj detektora značajki. Obično je postavljen na vrijednost koja je potencija broja 2, obično između 32 i 512. Korištenje više filtera rezultira

snažnijom neuronskom mrežom, ali povećava rizik od pretreniranja zbog većeg broja parametara kojeg treba procijeniti.

- veličina filtera - obično 3×3, 5×5 ili 7×7. Korištenjem filtera manjih dimenzija postiže se da se broj parametara koje treba naučiti znatno smanji te se osigurava da se o različitim uzorcima uči iz lokalnih regija.

- veličina koraka - označava za koliko se piksela pomiče prozor filtera. Vrijednost mu je obično 1, što znači da filter klizi piksel po piksel udesno i prema dolje. Međutim, veličina koraka se može povećati ukoliko je cilj da filter klizi s većim razmakom što rezultira manjim preklapanjem između receptivnih polja i smanjenjem rezultirajuće mape značajki.

- umetanje dodatnih vrijednosti uz rubove - sprječava da se mapa značajki smanji tijekom operacije konvolucije, jer se središnji piksel filtera slike može postaviti u rubni piksel ulazne slike. Na taj se način izbjegava kolaps dimenzija izlaznih značajki, omogućavajući dizajn dubljih mreža.

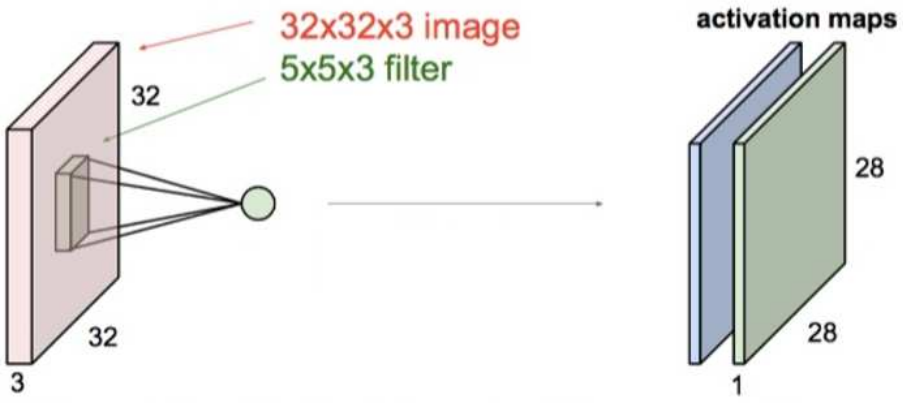
Primjer 3.5.1. *Ukoliko se na ulaznu sliku primijenjuje C_0 filtera, svaki veličine $k \times k$, i veličina ulazne slike je $W_i \times H_i \times C_i$, gdje W_i označava širinu, H_i visinu i C_i broj kanala (na ulazu je C_i obično jednak 3), volumen izlaza je $W_0 \times H_0 \times C_0$, gdje C_0 odgovara broju filtera koji se primijenjuje i još je*

$$W_0 = \frac{W_i - k + 2p}{s} + 1$$
$$H_0 = \frac{H_i - k + 2p}{s} + 1,$$

gdje je sa s označena veličina koraka, tj. koliko se piksela potrebno pomaknuti za računanje sljedeće konvolucije, a s p je označena veličina margina oko slike, dobivena ili zrcaljenjem ili nekom drugom metodom.

Ukoliko slika ima 3 kanala i s K_l, $l = 1, 2, \ldots, C_0$ je označen jedan filter veličine $5\times5\times3$, gdje 3 odgovara broju kanala ulazne slike, konvolucija slike I i filtera K_l tada zadovoljava formulu:

$$K_l * I(i, j) = \sum_{c=0}^{2} \sum_{m=0}^{4} \sum_{m=0}^{4} K_l(m, n, c) \cdot I(i + n - 2, j + m - 2, c).$$



Slika 3.20: Primjer konvolucijskog sloja

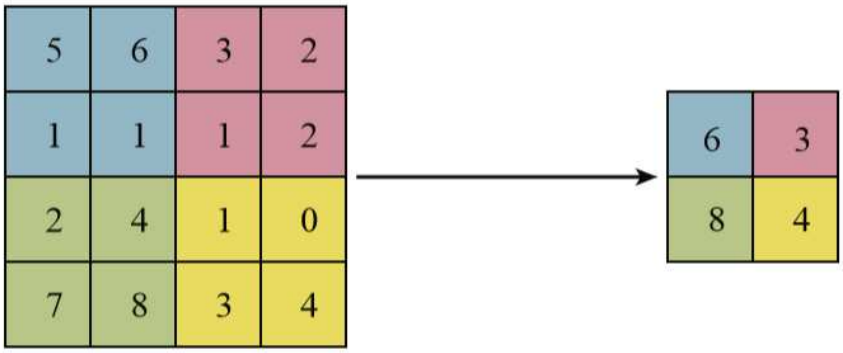
Za sliku s C^i kanala, oblik kompletne jezgre je (k, k, C^i, C^0), gdje C^0 predstavlja broj izlaznih kanala, ili ekvivalentno, broj filtera. Na slici (3.20) to je $(5, 5, 3, 2)$. Ukupan broj parametara koji se procijenjuju i povezuju s jezgrom oblika (k, k, C^i, C^0) jednak je $(k \cdot k \cdot C^i + 1) \cdot C^0$.

Nelinearni sloj

Konvolucijske neuronske mreže također uključuju upotrebu nelinearne aktivacijske funkcije nakon konvolucijskog sloja. Aktivacijska se funkcija najčešće definira unutar konvolucijskog sloja, primijenjena na dobivene mape značajki nakon operacije konvolucije. Ponekad se nelinearne transformacije provode kao neovisni sloj kako bi se omogućila veća preglednost mrežne arhitekture. Među mogućim aktivacijskim funkcijama najpopularnija je funkcija ReLU, a razlog je objašnjen u prijašnjem poglavlju.

Sloj udruživanja (en. *pooling layer*)

Svrha sloja udruživanja je smanjiti prostornu veličinu rezultata koji je dobiven konvolucijskim slojem. To uglavnom pojednostavljuje prikupljene podatke i stvara sažetu verziju istih podataka. Najčešći oblik udruživanja je maksimalno udruživanje koje pomiče prozor, najčešće veličine 2×2 preko svog ulaza i uzima maksimalnu vrijednost u prozoru, odbacujući sve ostale vrijednosti. Slično kao konvolucijski sloj, korisnik sam određuje veličinu prozora (analognu veličini filtera) i veličinu koraka/pomaka.



Slika 3.21: Maksimalno udruživanje

Potpuno povezani sloj

Obično su zadnji slojevi konvolucijske neuronske mreže potpuno povezani slojevi i to ne mora biti samo izlazni, klasifikacijski sloj. Ti su slojevi jednaki onima u običnim neuronskim mrežama i njihova je glavna uloga obavljanje klasifikacije obilježja koja su otkrivena i izdvojena nizom konvolucijskih slojeva i slojeva udruživanja. Da bi se unosile u potpuno povezane slojeve, mape značajki su prije transformirane u jedan jednodimenzionalni vektor.

Testiranje konvolucijske neuronske mreže

Slika se ostavlja u izvornom obliku, tj. ne treba ju prebaciti u sive tonove, dimenzija 96×96 i dubine 3, gdje dubina označava broj kanala slike, u ovom slučaju su to crveni, zeleni i plavi kanal. Ovakva se mreža puno brže trenira i dobivaju se bolji rezultata.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 96, 96, 32)	2432
activation_1 (Activation)	(None, 96, 96, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 48, 48, 32)	0
conv2d_2 (Conv2D)	(None, 48, 48, 64)	51264
activation_2 (Activation)	(None, 48, 48, 64)	0
max_pooling2d_2 (MaxPooling2	(None, 24, 24, 64)	0
flatten_1 (Flatten)	(None, 36864)	0
dense_1 (Dense)	(None, 1024)	37749760
activation_3 (Activation)	(None, 1024)	0
dropout_1 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 512)	524800
activation_4 (Activation)	(None, 512)	0
dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 2)	1026
activation_5 (Activation)	(None, 2)	0

Total params: 38,329,282
Trainable params: 38,329,282
Non-trainable params: 0

Slika 3.22: Građa konvolucijske neuronske mreže

U ovoj mreži, svaka je slika veličine 96×96 s dubinom 3 pa prvi konvolucijski sloj prima ulaze tih dimenzija, primijenjuje aktivacijsku funkciju ReLU te vraća mape značajki iste širine i visine, ali dubine 32, što je broj mapa značajki nakon prvog sloja. Nakon maksimalnog udruživanja, izlazni podatak je veličine $48 \times 48 \times 32$. Na to se primijenjuje još jedan kovolucijski sloj koji će zajedno s istom aktivacijskom funkcijom vratiti podatak veličine $48 \times 48 \times 64$, a maksimalnim udruživanjem dobije se izlazni podatak veličine

$24 \times 24 \times 64$. Nakon toga slijede dva potpuno povezana sloja, jedan s 1024 neurona, dok drugi ima 512. Konačno, nakon toga slijedi izlazni sloj koji ima dva moguća izlaza. Na taj je sloj primijenjena γ aktivacijska funkcija.

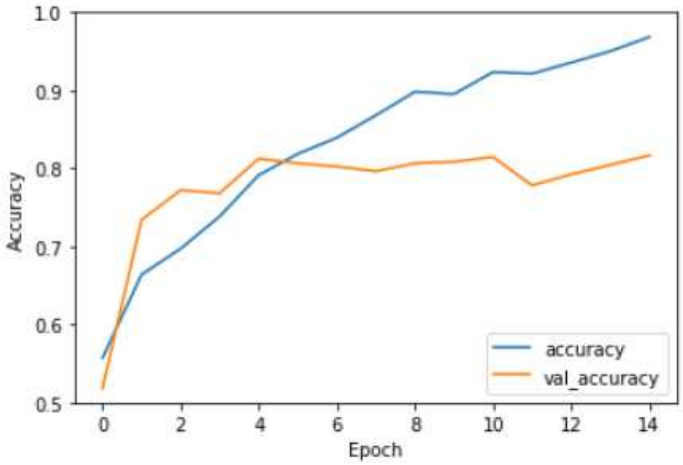
Kako skup za treniranje sadrži 1000 slika, a skup za validaciju i testiranje sadrži 1480 slika, ta se dva skupa združe i metodom *train_test_split* podijele na 1984 podataka za treniranje te 496 podataka za testiranje. Cijeli je model optimiziran Adam funkcijom, što je poboljšanje "mini-batch" gradijentnog spusta, s binarnom unakrsnom entropijom kao funkcijom troška. Kroz 15 epoha treniranje je izgledalo ovako:

```
Epoch 1/15
67/67 [==============================] - 72s 1s/step - loss: 0.8264 - categorical_accuracy: 0.5570 - val_loss: 0.6884 - val_categorical_accuracy: 0.5181
Epoch 2/15
67/67 [==============================] - 74s 1s/step - loss: 0.6401 - categorical_accuracy: 0.6638 - val_loss: 0.5879 - val_categorical_accuracy: 0.7339
Epoch 3/15
67/67 [==============================] - 72s 1s/step - loss: 0.5948 - categorical_accuracy: 0.6971 - val_loss: 0.5130 - val_categorical_accuracy: 0.7722
Epoch 4/15
67/67 [==============================] - 72s 1s/step - loss: 0.5410 - categorical_accuracy: 0.7384 - val_loss: 0.5013 - val_categorical_accuracy: 0.7681
Epoch 5/15
67/67 [==============================] - 72s 1s/step - loss: 0.4655 - categorical_accuracy: 0.7913 - val_loss: 0.4567 - val_categorical_accuracy: 0.8125
Epoch 6/15
67/67 [==============================] - 72s 1s/step - loss: 0.4061 - categorical_accuracy: 0.8185 - val_loss: 0.4679 - val_categorical_accuracy: 0.8065
Epoch 7/15
67/67 [==============================] - 72s 1s/step - loss: 0.3949 - categorical_accuracy: 0.8392 - val_loss: 0.4852 - val_categorical_accuracy: 0.8024
Epoch 8/15
67/67 [==============================] - 72s 1s/step - loss: 0.3413 - categorical_accuracy: 0.8679 - val_loss: 0.5355 - val_categorical_accuracy: 0.7964
Epoch 9/15
67/67 [==============================] - 72s 1s/step - loss: 0.2609 - categorical_accuracy: 0.8982 - val_loss: 0.5119 - val_categorical_accuracy: 0.8065
Epoch 10/15
67/67 [==============================] - 74s 1s/step - loss: 0.2632 - categorical_accuracy: 0.8952 - val_loss: 0.5290 - val_categorical_accuracy: 0.8085
Epoch 11/15
67/67 [==============================] - 72s 1s/step - loss: 0.1963 - categorical_accuracy: 0.9234 - val_loss: 0.4862 - val_categorical_accuracy: 0.8145
Epoch 12/15
67/67 [==============================] - 72s 1s/step - loss: 0.2084 - categorical_accuracy: 0.9214 - val_loss: 0.6004 - val_categorical_accuracy: 0.7782
Epoch 13/15
67/67 [==============================] - 72s 1s/step - loss: 0.1891 - categorical_accuracy: 0.9355 - val_loss: 0.8104 - val_categorical_accuracy: 0.7923
Epoch 14/15
67/67 [==============================] - 72s 1s/step - loss: 0.1551 - categorical_accuracy: 0.9501 - val_loss: 0.7224 - val_categorical_accuracy: 0.8044
Epoch 15/15
67/67 [==============================] - 72s 1s/step - loss: 0.0996 - categorical_accuracy: 0.9682 - val_loss: 0.8353 - val_categorical_accuracy: 0.8165
Model training complete...
```

Slika 3.23: Treniranje konvolucijske neuronske mreže

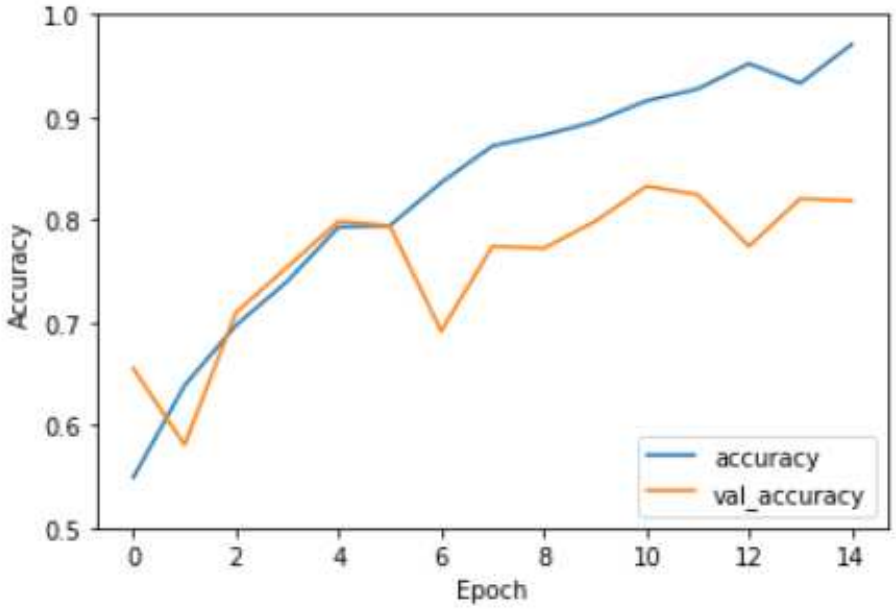
```
17/17 [==============================] - 4s 241ms/step - loss: 0.8353 - categorical_accuracy: 0.8165
accuracy: 81.65%
```

Slika 3.24: Točnost konvolucijske neuronske mreže na validation/test skupu je 81.65%



Slika 3.25: Graf treniranja konvolucijske neuronske mreže

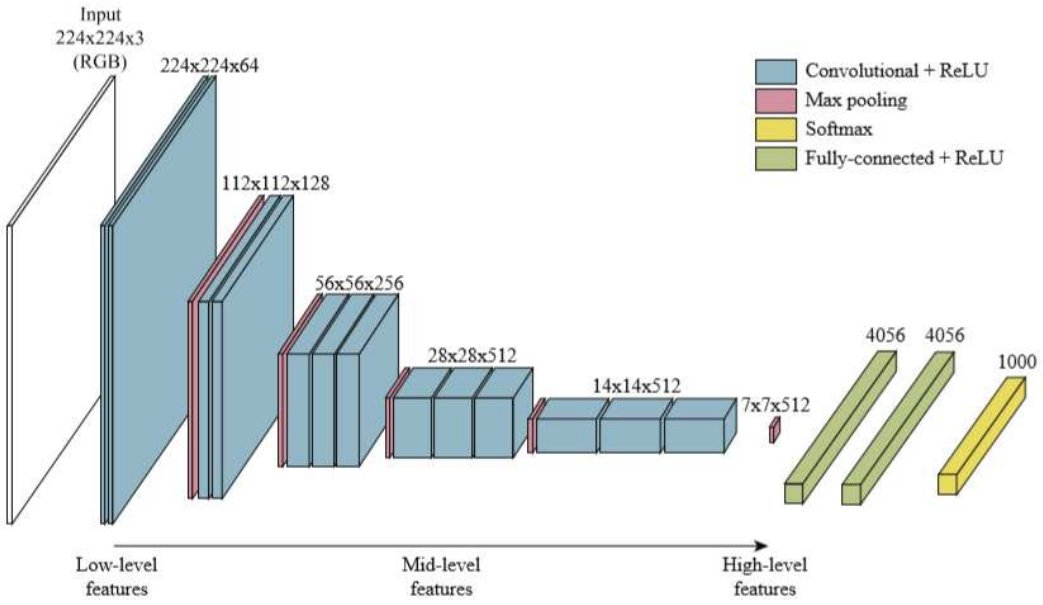
Da bi model bio usklađen s gore razrađenom teorijom, trenira se i testira s funkcijom troška "categorical_crossentropy", iako su obje funkcije troška na problemu s dvije klase jednake. Ponovnim treniranjem, ali s ovom funkcijom troška, dobiva se točnost od 81.85%.



Slika 3.26: Graf treniranja konvolucijske neuronske mreže, "cross_crossentropy" funkcija troška

VGG model konvolucijske neuronske mreže

U ovom se odjeljku navodi primjer poznate konvolucijske neuronske mreže: model *Visual Geometry Group* ili VGG model koja je bila početno rješenje za klasifikaciju ImageNet slika. Jedna njezina varijanta, koja će se kasnije testirati u sklopu prijenosnog učenja, ima 16 slojeva i zove se VGG16. VGG se može razdvojiti na dva dijela. Dio modela za ekstrakciju značajki sastoji se od parova konvolucijskih i ReLU slojeva, a zatim slijedi jedan sloj udruživanja. Ovaj se obrazac ponavlja više puta, smanjujući tako prostorne dimenzije podataka. U ovom nizu slojeva primjenjuje se nekoliko različitih filtera duž mreže, dobivajući značajke s različitim stupnjevima apstrakcije („hijerarhijska svojstva“). Klasifikacijski dio modela sadrži najsloženije značajke. Ovdje postoji prijelaz na potpuno povezane slojeve koji uče globalne obrasce u ukupnom ulaznom prostoru. Globalne informacije prisutne u posljednjem konvolucijskom sloju u potpunosti se koriste za obavljanje same klasifikacije.



Slika 3.27: VGG model konvolucijske neuronske mreže. Izlazni sloj s 1000 neurona pomoću γ aktivacijske funkcije izračunava vjerojatnost da ulazna slika pripada svakoj od 1000 kategorija uključenih u natjecanje ImageNet-a.

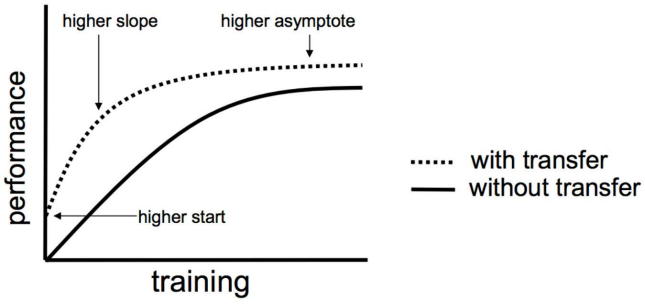
Poglavlje 4

Prijenosno učenje

Duboko učenje posljednjih je godina postiglo značajan napredak. To omogućuje rad sa složenijim problemima i dobivanje boljih rezultata. Vrijeme učenja i količina podataka koja je često potrebna za probleme dubokog učenja mnogo su veći od tradicionalnih problema.

Neuronske mreže se obično inicijaliziraju slučajnim težinama i one nakon određenog niza epoha dosežu vrijednosti koje omogućuju da se, u ovom slučaju, pravilno klasificiraju ulazne slike. Postoje različite mreže dubokog učenja s vrhunskim performansama (ponekad jednako dobrim ili čak boljim od ljudskih performansi) čiji se detalji u većini slučajeva dijele kako bi ih drugi mogli koristiti. Ove unaprijed osposobljene mreže čine osnovu prijenosnog učenja u kontekstu dubokog učenja ili takozvanog "duboko prijenosnog učenja". Tradicionalne tehnike strojnog učenja pokušavaju naučiti svaki zadatak ispočetka, dok tehnike prijenosnog učenja pokušavaju prenijeti znanje iz nekih prethodnih zadataka u ciljni zadatak kada potonji ima manje visokokvalitetnih podataka za treniranje. Time se težine u trenutrnom problemu inicijaliziraju na određene vrijednosti za koje se unaprijed zna da su dobre za klasificiranje određenog skupa podataka. Na taj se način postiže da trenutni skup podataka ne treba biti toliko velik kao kad se trenira mrežu ispočetka niti treba odabrati dobar broj epoha da bi težine bile postavljene na dovoljno dobre vrijednosti za klasifikaciju.

Neke od prednosti unaprijed obučenih modela su brzo treniranje, dobra izvedba i prilagodljivost postojećim arhitekturama. Treniranje ovako može trajati nekoliko minuta umjesto sati ili dana pa to omogućava brzo testiranje hipoteza i povećanje produktivnosti. Zahvaljujući prijenosnom učenju, mogu se graditi točni modeli na relativno malim skupovima podataka. Dodatna je prednost da ukoliko se potroše vrijeme i resursi gradeći infrastrukturu oko određenog okvira modeliranja, mogu se poboljšati performanse modela bez potpune promjene arhitekture. Neke od prednosti prijenosnog učenja vidljive su i na sljedećoj slici.



Slika 4.1: Prijenosnim učenjem model ima uspješnije početno i konačno ponašanje te strmiji nagib što znači da model brže uči, tj. uspješnost mu je svakim korakom veća od modela koji nije unaprijed treniran.[6]

Prijenosno učenje je ono učenje u kojem se model trenira na zadatku u jednoj domeni i aktivira se za drugu srodnu domenu i zadatak. Domena za trening naziva se izvorna domena, a druga domena naziva se ciljna domena. Slično se zadaci u različitim domenama nazivaju izvorni i ciljni zadatak.

Domena $\mathcal{D}$ definira se kao uređeni par $\mathcal{D} = \{\mathcal{X}, P(X)\}$, gdje je $\mathcal{X}$ prostor značajki, a $P(X)$ pridružena vjerojatnosna distribucija, gdje je $X = (x_1, x_2, \ldots, x_n) \in \mathcal{X}$ koji predstavlja skup primjera (instanci) za treniranje. Promatrajući problem klasifikacije slika određenih dimenzija, svaki piksel predstavlja jednu značajku, pa je $\mathcal{X}$ prostor matrica svih mogućih vrijednosti piksela, a x_i reprezentacija jedne slike. Za danu domenu $\mathcal{D} = \{\mathcal{X}, P(X)\}$, zadatak $\mathcal{T} = \{\mathcal{Y}, f\}$ se sastoji od dvije komponente: prostora obilježja (oznaka) $\mathcal{Y}$, što je u klasifikacijskom problemu konačan prostor kategorija, i klasifikatora f koji je zadan svojom vrijednošću na skupu za učenje. Funkcija f uči pomoću uređenih parova (x_i, y_i), $x_i \in X$ i $y_i \in \mathcal{Y}$, koji čine skup podataka za treniranje, kako bi mogla predvidjeti vrijednosti $y = f(x)$ za novi primjer x. Danom izvornom domenom $\mathcal{D}_S$ i zadatkom učenja $\mathcal{T}_S$ te ciljnom domenom $\mathcal{D}_T$ i zadatkom učenja $\mathcal{T}_T$, prijenosno učenje želi unaprijediti reprezentaciju klasifikatora f_T u $\mathcal{D}_T$ koristeći znanje iz $\mathcal{D}_S$ i $\mathcal{T}_S$ gdje $\mathcal{D}_S \neq \mathcal{D}_T$, ili $\mathcal{T}_S \neq \mathcal{T}_T$. Konkretno je ovdje cilj učenjem samo klasifikatora $f_{S,T}$ reprezentirati klasifikator $f_T(x) = f_{S,T}(f_S(x))$ do dovoljne razine točnosti.

U ovom je slučaju riječ o induktivnom prijenosnom učenju, učenju gdje $\mathcal{T}_S \neq \mathcal{T}_T$, tj. ciljni se zadatak razlikuje od izvornog zadataka, neovisno o odnosu domena. Objašnjeno na problemu određivanja je li slika zamućena ili nije, u izvornom se zadatku koristi javno dostupan ImageNet skup podataka za treniranje kojom se uči funkcija f_S definirana modelima dubokih neuronskih mreža koje će biti opisane kasnije. Funkcijom f_S se slikama pridružuje njihova kategorička oznaka. U ciljnom zadatku se koristi manji skup podataka za treniranje, podskup iz *EvaluationSet*-a za učenje funkcije f_T, definirane gotovo istim modelima dubokih neuronskih mreža, uz promjenu izlaznog sloja. Funkcija f_T svakoj slici

pridružuje oznaku 0 ili 1 u ovisnosti je li slika zamučena ili nije. Koristeći znanje izvornog zadatka, unaprijedit će se učenje od f_T dvjema strategijama prijenosnog učenja na modelima dubokih neuronskih mreža, a to su ekstrakcija značajki i fino ugađanje.

Sustavi i modeli dubokog učenja slojevite su arhitekture koje uče različite značajke na različitim slojevima (hijerarhijski prikaza slojevitih značajki). Ti se slojevi konačno spajaju sa zadnjim slojem (obično u potpunosti povezanim slojem, u slučaju nadziranog učenja) kako bi se dobio konačni rezultat. Ova slojevita arhitektura omogućava korištenje unaprijed obučene mreže bez završnog sloja kao ekstraktor fiksnih značajki za druge zadatke. Ključna ideja ovdje je samo iskoristiti ponderirane slojeve unaprijed treniranog modela kako bi se izdvojile značajke, ažurirati samo težine izlaznog sloja, a ne i ažurirati težine ostalih slojeva modela tijekom treninga s novim podacima za novi zadatak.

Kod finog ugađanja ne mijenja se samo završni sloj (za klasifikaciju ili regresiju), već se ažuriraju i neke od prethodnih slojeva. Pomoću finog ugađanja prvo se mijenja zadnji sloj da bi odgovarao klasama u ciljnom skupu podataka, kao što se to učini i kod ekstrakcije značajki. Ovdje se dodatno treniraju, tj. mijenjaju težine slojeva mreže po želji, a u ovom slučaju, to će biti cijela mreža.

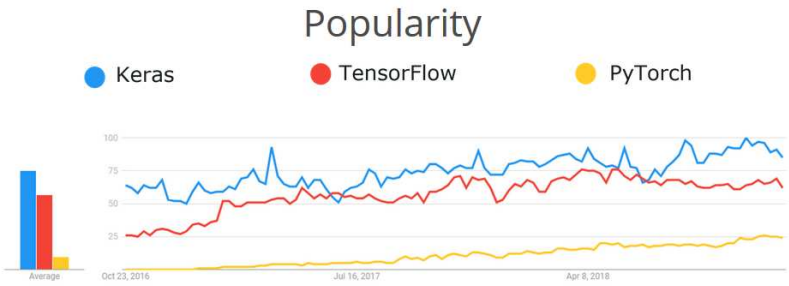
S finim ugađanjem ne postoji ograničenje samo na treniranje klasifikacijskog, tj. izlaznog ili potpuno povezanog sloja, već se treniraju i prethodni slojevi, a to su konvolucijski slojevi i slojevi udruživanja. Prvo se prenosi učenje, tj. inicijaliziraju se sve težine na vrijednosti koje su naučene na ImageNetu. To daje osnovu koja će se morati pospješiti za ciljni problem učenjem ne samo težina izlaznog sloja, već i svih ostalih slojeva. Puno više o metodi može se pročitati u [18], [29] i [22].

Da bi se testirala uspješnost ovog načina učenja, isprobavaju se razni pretrenirani modeli na istom skupu podataka kako bi se uočilo koji daje najbolje rezultate i jesu li oni bolji od prethodno isprobanih metoda. Osim testiranja pretreniranih modela, testira se i model iste arhitekture, ali s nasumičnom inicijalizacijom parametara.

Ako je riječ o finom ugađanju, ažurirat će se svi parametri. No, ako je riječ o metodi ekstrakcije značajki, ažurirat će se samo parametri koji potpuno povezuju zadnji skriveni sloj s izlaznim slojem koji klasificira slike. Prije samog treniranja i testiranja, opisuju se okviri u kojima je ono provedeno, a nakon toga slijedi kratki opis korištenih modela, o čemu se više može pročitati na [31] i [25].

4.1 Keras, TensofFlow, PyTorch

S povećanjem potražnje u području obrade podataka, u industriji je došlo do ogromnog rasta tehnologije dubokog učenja. Time su sva tri okvira, Keras, PyTorch i TensorFlow, stekla veliku popularnost. Keras je na vrhu liste, a slijede ga TensorFlow i PyTorch. Ogromnu popularnost Keras je stekao zbog svoje jednostavnosti u usporedbi s ostala dva.



Slika 4.2: Popularnost okvira koji se koriste u testiranju metoda, preuzeto sa [8]

Keras je *open source* biblioteka neuronskih mreža napisana u Pythonu. Može se pokretati preko TensorFlow-a. Dizajniran je da omogući brzo eksperimentiranje s dubokim neuronskim mrežama. Popularan je zbog sintaktičke jednostavnosti i korisničke naravi, čitljiviji je i sažetiji. Brzina Kerasa uglavnom je sporija u usporedbi s Tensorflowom i Pytorchom. Keras je usvojen i integriran u TensorFlow sredinom 2017. godine. Korisnici mu mogu pristupiti putem modula *tf.keras*. Međutim, Kerasova biblioteka i dalje može raditi odvojeno i neovisno.

TensorFlow je *open source* biblioteka softvera za programiranje protoka podataka kroz različite zadatke, a dodatno je kompatibilna s različitim jezicima, poput C, C++, Java... To je matematička biblioteka koja se koristi za modele strojnog učenja poput neuronskih mreža. Brzina Tensorflowa vrlo je velika i pruža visoke performanse. Koristi se za modele visokih performansi i velike skupove podataka koji zahtijevaju brzu izvedbu. Tensorflow, za razliku od Kerasa, nije toliko jednostavan za korištenje, ali pruža Kerasu okvir koji mu olakšava posao.

PyTorch je *open source* biblioteka strojnog učenja samo za Python koja se temelji na Torchu. Koristi se za modele poput prirodne obrade jezika, a razvila ga je istraživačka grupa AI u Facebooku. Brzina i performanse Pytorcha vrlo su slični Tensorflowu. Ovaj je okvir namijenjen izgradnji neuronskih mreža i za obradu prirodnog jezika. Koristi se za modele visokih performansi i velike skupove podataka koji zahtijevaju brzu izvedbu, arhitektura mu je složenija i čitljivost je slabija u usporedbi s Kerasom.

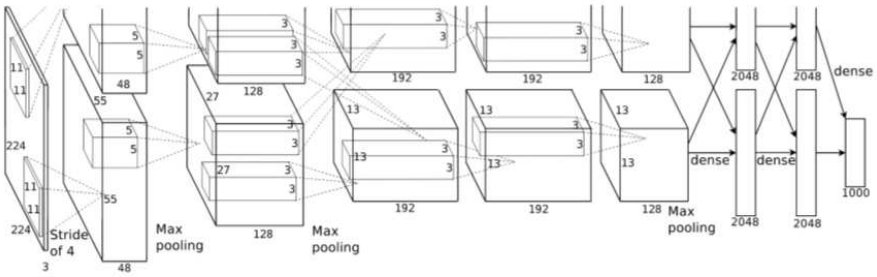
Prilikom biranja Kerasa ili PyTorcha, treba znati da su oba dobra ukoliko se tek počinje raditi s okvirima dubokog učenja. Matematičari i iskusni istraživači prije će odabrati PyTorch, a Keras je pogodniji za programere koji žele okvir pomoću kojeg mogu brzo graditi, trenirati i evaluirati svoje modele. Pytorch je brži od Kerasa i ima bolje mogućnosti uklanjanja pogrešaka. Obje platforme podjednako su popularne i obje nude puno resursa za učenje. Keras ima izvrstan pristup kodu i priručnicima, dok Pytorch ima izvanrednu podršku u zajednici i aktivni razvoj. Zato je Keras bolji kada se radi s malim skupovima podataka, brzim prototipiranjem i višestrukim back-end podrškama. On je ujedno i najpo-

pularniji okvir zahvaljujući svojoj jednostavnosti, a radi na Linuxu, MacOS-u i Windows-u.

4.2 Poznati i javno dostupni modeli

AlexNet

AlexNet je konvolucijska neuronska mreža koja se sastoji od osam slojeva; prvih pet su konvolucijski slojevi, gdje kod nekih slijede max-pooling slojevi, a posljednja tri su potpuno povezani slojevi. Mreža koristi nezasićenu aktivacijsku funkciju ReLU, što je pokazalo poboljšane performanse treninga od funkcija tanh i sigmoid. AlexNet se smatra jednim od najutjecajnijih radova objavljenih u računalnom vidu, jer je potaknuo još mnogo objavljenih radova koji koriste CNN i GPU kako bi se ubrzalo duboko učenje.



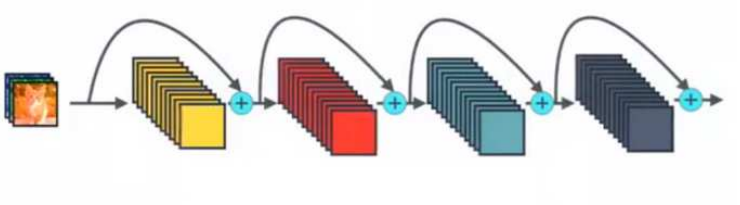
Slika 4.3: Arhitektura modela AlexNet

ResNet

ResNet modeli implementirani su s dvoslojnim ili troslojnim preskocima slojeva koji između njih provode ReLU i "mini-batch" normalizaciju. Može se koristiti i dodatna matrica težina za učenje preskakanja težina i ti su modeli poznati kao HighwayNets. Mogući su i modeli s nekoliko paralelnih preskoka koji se nazivaju DenseNet.

Jedna motivacija za preskakanje slojeva je izbjegavanje problema nestajućeg gradijenata (problem u kojem gradijentni spust daje male vrijednosti gradijenta pa ne dolazi do promjene težina) ponovnom uporabom aktivacija iz prethodnog sloja sve dok susjedni sloj ne nauči težinu. Zato preskakanje učinkovito pojednostavljuje mrežu, koristeći manje slojeve u početnim fazama treninga i ubrzava učenje smanjujući utjecaj nestajućih gradijenata, jer ima manje slojeva za širenje kroz njih. Mreža zatim postupno obnavlja preskočene slojeve kako uči prostor sa značajkama.

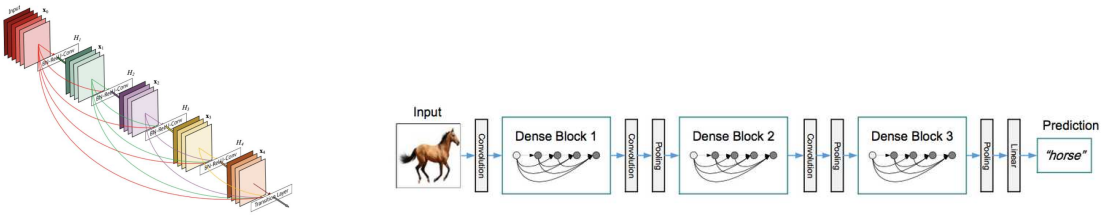
Prednosti ResNeta su ti da se povećava brzina treninga dubokih mreža, smanjuje problem nestajućeg gradijenta te postiže veća preciznost u radu mreže, posebno u klasifikacijama slika.



Slika 4.4: Arhitektura ResNeta

DenseNet

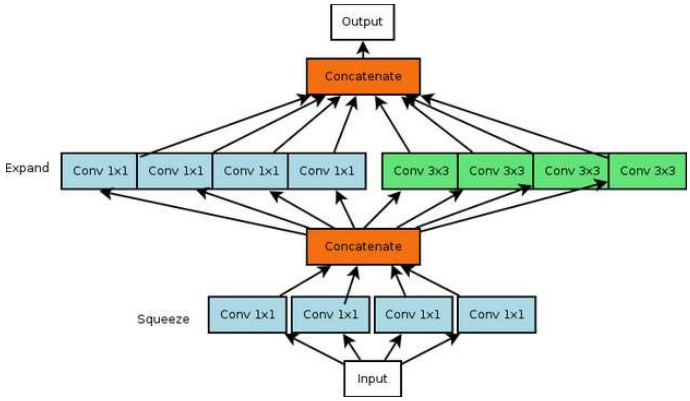
DenseNet je po arhitekturi sličan arhitekturi ResNeta, ali u DenseNetu svaki sloj dobiva dodatne ulaze iz svih prethodnih slojeva i prosljeđuje svoju mapu značajki na sve sljedeće slojeve. Svaki sloj prima "kolektivno znanje" od svih prethodnih slojeva. Cijela je arhitektura podijeljena na više tako povezanih blokova. Slojevi između su prijelazni konvolucijski i pooling slojevi. Neke od prednosti DensNeta su smanjenje problema s iščezavajućim gradijentom, poticanje ponovne upotrebe značajki i smanjenje broja parametara.



Slika 4.5: Arhitektura DenseNeta

SqueezeNet

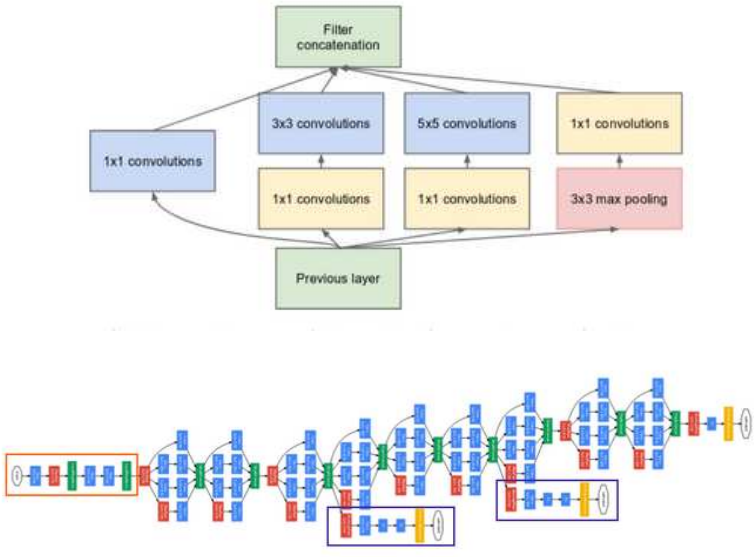
Pri dizajniranju SqueezeNet-a, cilj je bio stvoriti manju neuronsku mrežu s manje parametara koja bi se lakše uklopila u memoriju računala i lakše prenosila putem računalne mreže. Ovaj model sadrži relativno malu količinu parametara i može postići točnost razine AlexNeta na skupu podataka ImageNet s 50 puta manje parametara.



Slika 4.6: Arhitektura SqueezeNeta

GoogLeNet/Inception

GoogLeNet je model s drugačijom arhitekturom od ostalih spomenutih modela. Osnovni građevni blok ovog modela je takozvani početni modul. Početni modul radi višestruke konvolucije, različitih veličina filtera, i udruživanje u jedan sloj. Kao rezultat, umjesto da se odluči kada će se koristiti koji sloj za najbolji rezultat, mreža to automatski shvati nakon treninga. Koristi se 1×1 konvolucija koja radi kao selektor značajki.



Slika 4.7: Prva slika prikazuje osnovi blok mreže, a druga prikazuje arhitekturu cijele mreže

4.3 Testiranje

Keras

Kako se prijenosno učenje može implementirati i testirati koristeći i Keras i PyTorch, testirat će se metode pomoću oba obrasca, prvo s Kerasom. Na donjoj je slici prikaz dostupnih unaprijed treniranih modela. Od navedenih, ovdje su testirane VGG16, ResNet152V2 i Xception mreža.

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-
EfficientNetB0	29 MB	-	-	5,330,571	-
EfficientNetB1	31 MB	-	-	7,856,239	-
EfficientNetB2	36 MB	-	-	9,177,569	-
EfficientNetB3	48 MB	-	-	12,320,535	-
EfficientNetB4	75 MB	-	-	19,466,823	-
EfficientNetB5	118 MB	-	-	30,562,527	-
EfficientNetB6	166 MB	-	-	43,265,143	-
EfficientNetB7	256 MB	-	-	66,658,687	-

Slika 4.8: Dostupni modeli u spremljeni u "~/.keras/models/"

Tok učenja za Xception arhitekturu koja je nadogradnja Inception v3 arhitekture (više na [20]) izgleda ovako:

- Instanciranje osnovnog modela i učitavanje unaprijed treniranih parametara u model

  base_model = Xception(weights='imagenet', include_top=False,
  input_shape=input_shape)

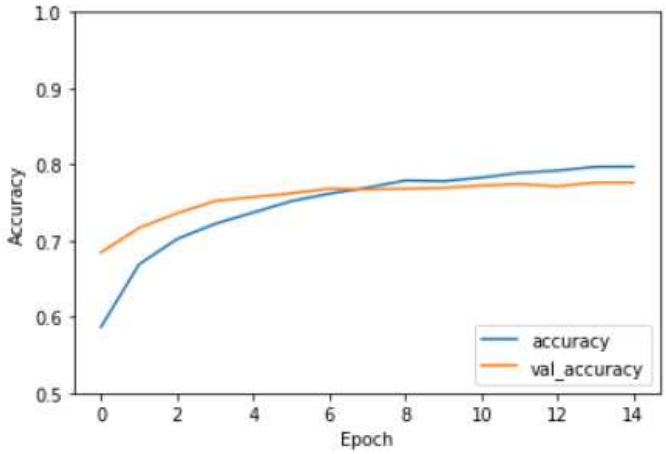
- Zamrzavanje svih slojeva u osnovnom modelu (trainable = False)

- Dodavanje jednog ili više potpuno povezanih slojeva, od kojih je jedan završni klasifikacijski sloj kojemu je izlaz predviđena oznaka za naš problem

$$\text{outputs} = \text{Dense}(2)(x)$$

- Treniranje novog modela

```
model.compile(optimizer=keras.optimizers.Adam(),
              loss=keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=[keras.metrics.BinaryAccuracy()])
history = model.fit(X_train_resized, Y_train,
          batch_size=30,
          validation_data=(X_test_resized, Y_test),
          epochs=15)
```

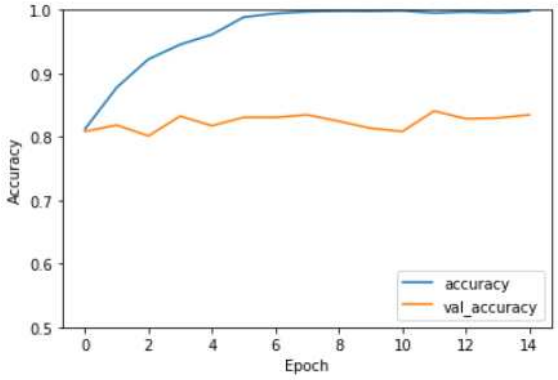


Slika 4.9: Kroz 15 epoha, gdje je veličina "mini-batch" skupa za treniranje jednaka 30, ekstrakcijom značajki, postiže se maksimalna točnost od 77.62% na skupu za validaciju i testiranje.

Jednom kada model završi treniranje, dodatno se mogu odmrznuti parametri osnovnog dijela i ponoviti treniranje cijelog modela s vrlo niskom stopom učenja. Ovo je neobavezni posljednji korak koji potencijalno može poboljšati uspješnost modela. To također može potencijalno dovesti do brze prenaučenosti.

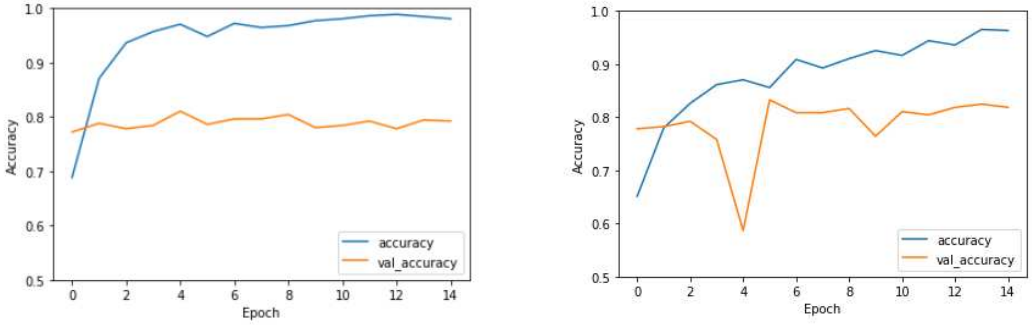
Ključno je napraviti ovaj korak tek nakon što se model sa smrznutim slojevima uspješno istrenira. Također je presudno koristiti vrlo nisku stopu učenja u ovoj fazi, jer se trenira puno veći model nego u prvoj fazi treninga, na skupu podataka koji je obično vrlo mali. Kao rezultat toga, postoji velika opasnost da se model prenauči.

Nakon implementacije i pokretanja ovakvog modela dolazi se do sljedećih rezultata:



Slika 4.10: Vidi se kako je model postao prenaučen, nakon šeste epohe na skupu za treniranje postigao je točnost od gotovo 100%, dok se točnost na skupu za validaciju/testiranje ne mijenja. Dovoljno bi bilo ovakav model pokrenuti kroz još 4 epohe finog ugađanja i dobiti malo bolju točnost na skupu za treniranje, koja bi u tom slučaju iznosila 83.27%.

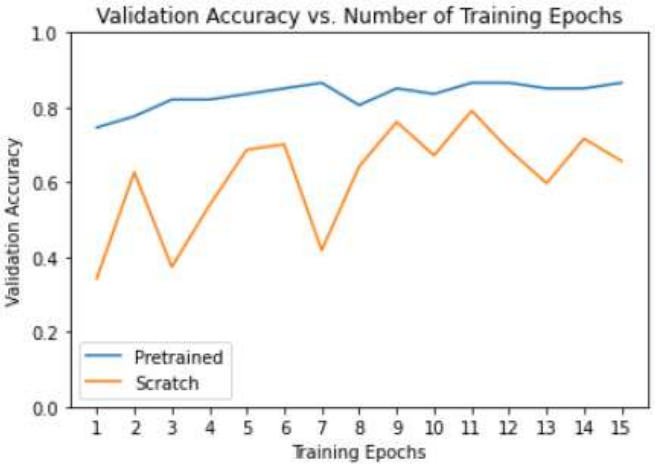
U Kerasu se implementira VGG16 i ResNet152V2, samo uz ekstrakciju značajki i dobivaju se sljedeći rezultati:



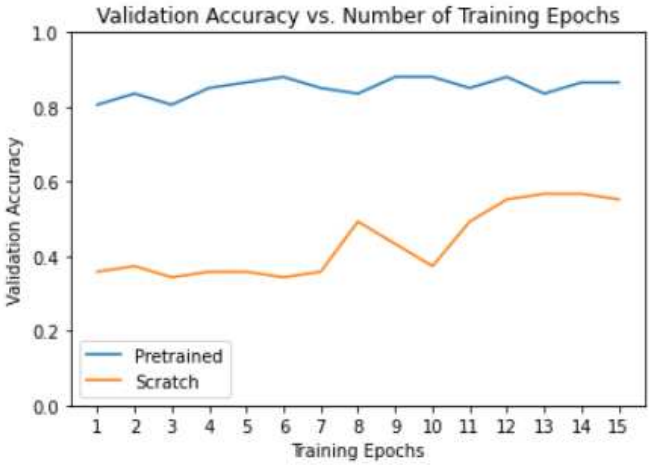
Slika 4.11: Na lijevoj su slici rezultati ResNet152V2 mreže, a na desnoj su slici rezultati VGG16 mreže, trenirane na slikama veličine $96 \times 96 \times 3$. U prvom modelu najveća je točnost od 83.27%, dok je na drugom najveća uspješnost 81.05% na skupu za validaciju i testiranje.

TensorFlow

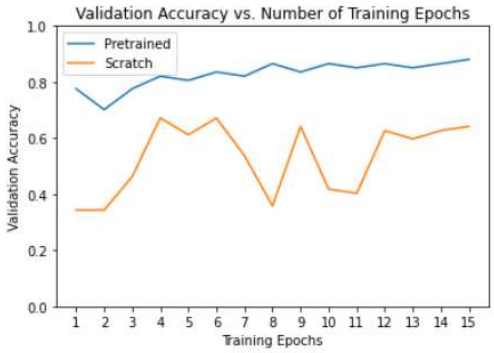
Nakon korištenja Kerasa, testiranje se provodi korištenjem TensorFlow-a. Uz pomoć [13], pokazalo se da je izvođenje podjednako brzo. Uz TensorFlow modele trenira se i testira na slikama većih dimenzija (224×224), ali se koristi puno manji skup podataka. Sljedećim slikama pokazani su rezultati testiranja svakog od prije navedenih modela i njihove uspješnosti. Prvo se prikazuju rezultati samo uz ekstrakciju značajki.



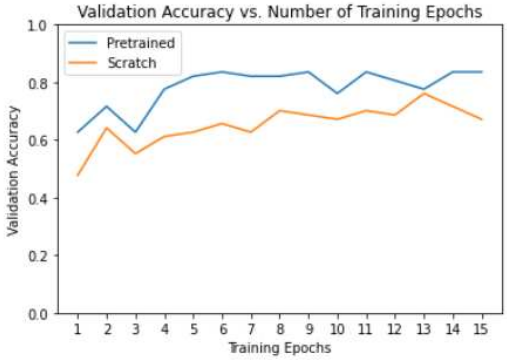
Slika 4.12: VGG



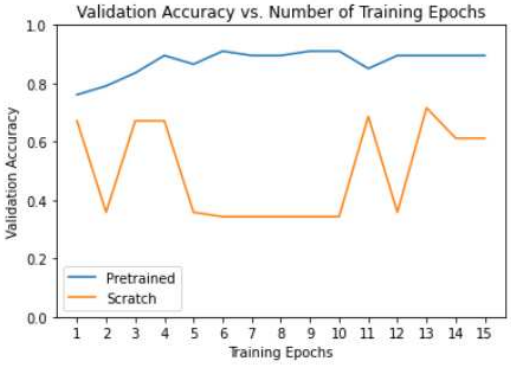
Slika 4.13: AlexNet



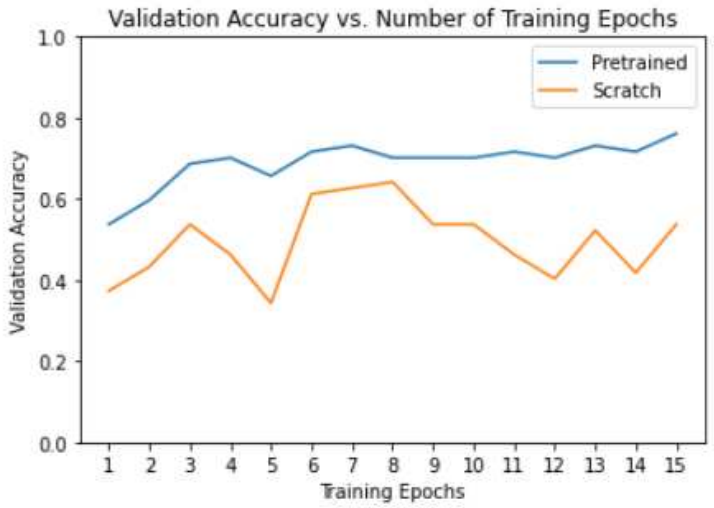
Slika 4.14: ResNet



Slika 4.15: DenseNet



Slika 4.16: SqueezeNet



Slika 4.17: Inception

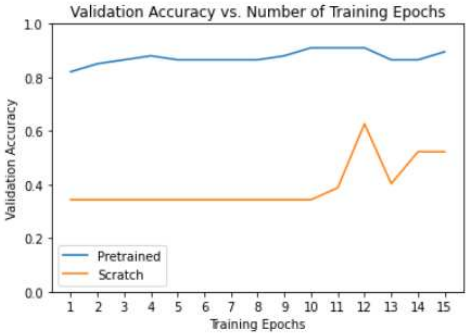
Svi se modeli treniraju istim skupom podataka koji se sastoji od 158 čistih slika i 155 zamućenih slika koje su nasumično odabrane iz prvotnog skupa za treniranje. Za testiranje se koristi 44 čistih slika i 23 zamućene slike. Modeli se, osim ekstrakcijom značajki, treniraju ispočetka, s nasumično inicijaliziranim parametrima. Cilj je pokazati da model s ekstrakcijom značajki daje puno bolje rezultate, od modela s istom arhitekturom, ali bez unaprijed postavljenih parametara.

Za sve mreže, osim Inceptiona, treniraju se i mijenjaju parametri zadnjeg sloja, a to su vrijednosti zapisane u ".weight" ".bias" argumentima. Kod VGG mreže, na slici (4.12) prikazano je testiranje oba modela kroz svaku epohu, i točnost modela s ekstrakcijom značajki je 86.57%, a točnost modela bez unaprijed treniranih značajki je 79.10%, što i nije tako loš rezultat, ali uz činjenicu da je skup podataka za testiranje dosta mali. Tako se odabralo jer bi se u suprotnom svaki od algoritama izvršavao više od sat vremena. Dalje, na slici (4.13), prikaz je točnosti na skupu za testiranje AlexNet modela. Pokazuje se da model kojem su težine nasumično postavljene uopće ne uči, dok se za model s ekstrakcijom značajki dobije točnost od 88.06%. Slika (4.14) pokazuje da se ekstrakcijom značajki također dobije točnost od 88.06%, ali za razliku od AlexNet-a, model uči i ako se parametri nasumično inicijaliziraju. Za DenseNet, prikazano na slici (4.15), dobije se najbolja točnost 83.58% uz ekstrakciju značajki, a za SqueezeNet, slika (4.16), točnost modela iznosi 91.04%. Time je ovo najbolje postignuti rezultat. Slika (4.17) pokazuje učenje Inception modela, gdje se, za razliku od ostalih modela, uče *AuxLogits.fc.weight*, *AuxLogits.fc.bias*, *fc.weight* i *fc.bias* parametri. Točnost koja se dobije je 76.12%, što je ujedno i najslabiji rezultat. Svi su rezultati prikazani tablicom.

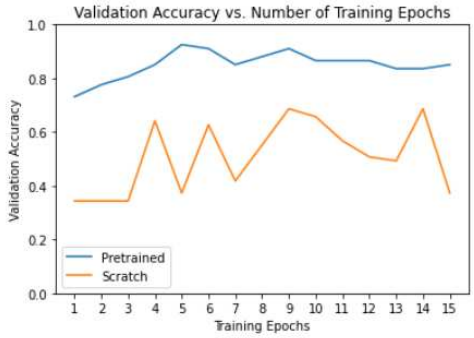
Model	Točnost: ekstrakcija značajki	Vrijeme izvršavanja	Točnost: nasumična inicijalizacija parametara	Vrijeme izvršavanja
AlexNet	88.06%	14m 22s	55.22%	20m 6s
ResNet	88.06%	19m 30s	67.16%	32m 9s
VGG	86.57%	38m 31s	79.10%	85m 32s
DenseNet	83.58%	35m 2s	76.12%	73m 26s
SqueezeNet	91.04%	17m 42s	71.64%	23m 60s
Inception	76.12%	40m 26s	64.18%	86m 32s

Tablica 4.1: Rezultati testiranja prijenosnog učenja ekstrakcijom značajki

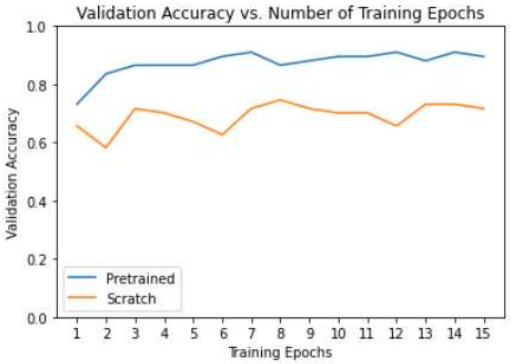
Rezultati dobiveni finim ugađanjem svakog modela, tj. treniranjem cijele mreže uz odabir unaprijed treniranih težina dani su sljedećim slikama:



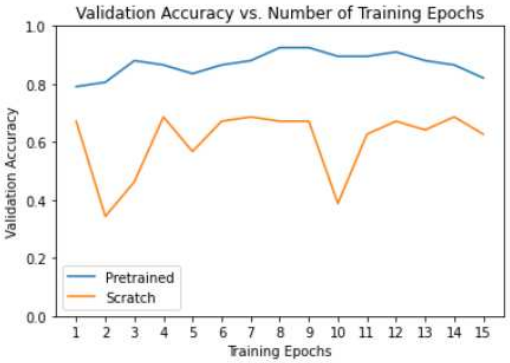
Slika 4.18: AlexNet



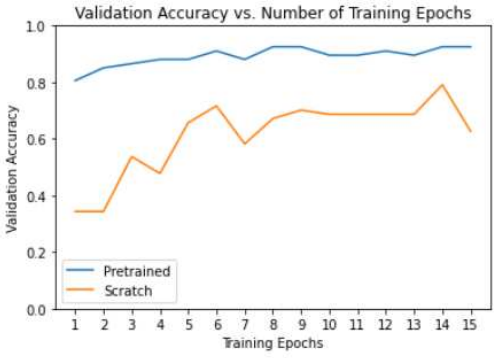
Slika 4.19: ResNet



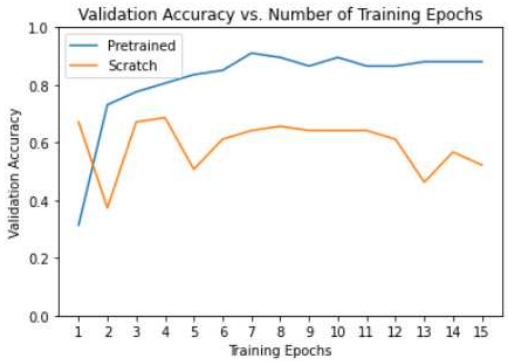
Slika 4.20: DenseNet



Slika 4.21: SqueezeNet



Slika 4.22: VGG



Slika 4.23: Inception

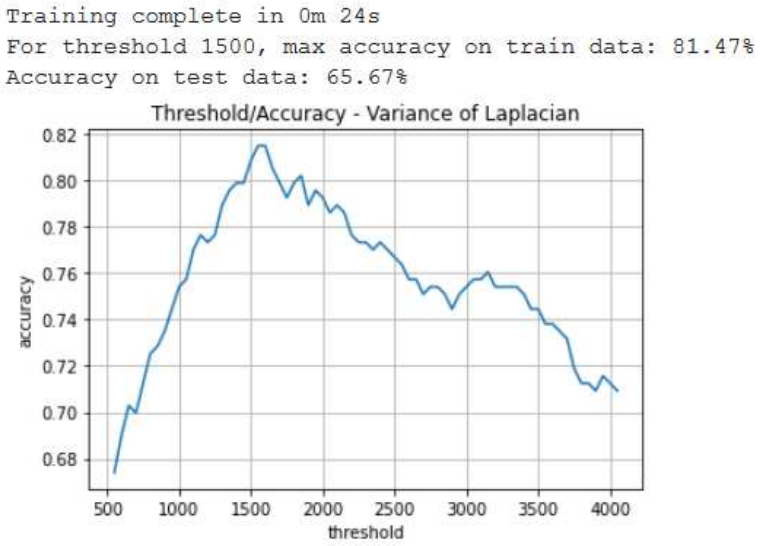
Finim ugađanjem dobivaju se dosta slični rezultati, a može ih se opisati tablicom:

Model	**Točnost: fino ugađanje**	**Vrijeme izvršavanja**	**Točnost: nasumična inicijalizacija parametara**	**Vrijeme izvršavanja**
AlexNet	91.04%	18m 58s	62.69%	19m 10s
ResNet	92.54%	32m 0s	68.67%	31m 58s
VGG	92.54%	86m 4s	$79.10%	86m 28s
DenseNet	91.04%	64m 37s	74.63%	64m 42s
SqueezeNet	92.54%	22m 1s	68.66%	22m 7s
Inception	91.04%	79m 39s	68.66%	78m 35s

Tablica 4.2: Rezultati testiranja prijenosnog učenja finim ugađanjem

Dobiveni rezultati pokazuju da je metoda prijenosnog učenja dobra i učinkovita. Odabirom bilo kojeg modela s unaprijed treniranim težinama moguće je dobiti točnost iznad 90%. Usporedi li se to s točnošću Laplaceovim pristupom, koji na ovom skupu za testiranje daje maksimalnu točnost od 65.67%, može se zaključiti da je prijenosno učenje metoda s daleko najboljim rezultatima. Vrijeme potrebno za izvršavanje bilo kojeg modela prijenosnog učenja puno je veće od izračunavanja varijance Laplasijana slike. Za najbržu metodu prijenosnog učenja, AlexNet s ekstrakcijom značajki, potrebno je bilo 14 minuta i 22 sekunde, a Laplaceov se pristup izvrši u 24 sekunde za 313 slika skupa za treniranje. Iako je vrijeme potrebno za Laplaceov pristup puno manje, rezultati koje daje na malom skupu podataka su lošiji u odnosu na prijenosno učenje koje se baš temelji na korištenju manjeg skupa podataka za treniranje. Također, treniranjem na početnom skupu podataka,

iz *TrainingSet*-a, Laplaceova metoda ne daje bolje rezultate na skupu za testiranje. Unatoč duljem izvršavanju, za problem ovog rada, zato bi najbolje bilo upotrijebiti neki od modela prijenosnog učenja, poput AlexNeta ili SqueezeNeta koji se izvršavaju najbrže, a daju i najbolje rezultate.



Slika 4.24: Odnos točnosti i granične vrijednosti varijance Laplasijana slika na istom skupu za treniranje koji se koristio i za treniranje modela prijenosnog učenja

Bibliografija

[1] *Edge Detection*, https://en.wikipedia.org/wiki/Edge_detection, (kolovoz 2020.).

[2] *Konvolucija*, https://en.wikipedia.org/wiki/Convolution, (kolovoz 2020.).

[3] *Laplacian Operator*, https://www.tutorialspoint.com/dip/laplacian_operator.htm, (kolovoz 2020.).

[4] *Neural Networks and Introduction to Deep Learning*, https://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-hdstat-rnn-deep-learning.pdf, (kolovoz 2020.).

[5] Saad J Bedros, *Edge Detection*, http://dept.me.umn.edu/courses/me5286/vision/VisionNotes/2017/ME5286-Lecture7-2017-EdgeDetection2.pdf, (kolovoz 2020.).

[6] Jason Brownlee, *A Gentle Introduction to Transfer Learning for Deep Learning*, https://machinelearningmastery.com/transfer-learning-for-deep-learning/, (kolovoz 2020.).

[7] Vitaly Bushaev, *Adam — latest trends in deep learning optimization*, https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c, (kolovoz 2020.).

[8] Francois Chollet, *Transfer learning and fine-tuning*, https://keras.io/guides/transfer_learning/, (kolovoz 2020.).

[9] Marin Doko, *Primjena strojnog učenja u održavanju*, Disertacija, University of Zagreb. Faculty of Mechanical Engineering and Naval Architecture., 2018.

[10] Gavin Edwards, *Machine Learning — An Introduction*, https://towardsdatascience.com/machine-learning-an-introduction-23b84d51e6d0, (kolovoz 2020.).

[11] Ilja Gogić, Pavle Pandžić i Josip Tambača, *Diferencijalni račun funkcija više varijabli*, 2019.

[12] Dr. Mark Humphrys, *Single-layer Neural Networks (Perceptrons)*, https://www.computing.dcu.ie/~humphrys/Notes/Neural/single.neural.html, (kolovoz 2020.).

[13] Nathan Inkawhich, *Finetuning Torchvision Models*, https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html, (kolovoz 2020.).

[14] Ramesh Jain, Rangachar Kasturi i Brian G Schunck, *Machine vision*, sv. 5, McGraw-hill New York, 1995.

[15] Shubham Jain, *An Overview of Regularization Techniques in Deep Learning (with Python code)*, https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/, (kolovoz 2020.).

[16] Lili Jiang, *A Visual Explanation of Gradient Descent Methods (Momentum, AdaGrad, RMSProp, Adam)*, https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c, (kolovoz 2020.).

[17] Ayoosh Kathuria, *Intro to optimization in deep learning: Momentum, RMSProp and Adam*, https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/, (kolovoz 2020.).

[18] Dr. Vaibhav Kumar, *Transfer Learning For Multi-Class Image Classification Using Deep Convolutional Neural Network*, https://analyticsindiamag.com/transfer-learning-for-multi-class-image-classification-using-deep-convolutional-neural-network/, (kolovoz 2020.).

[19] Niranjan Kumar, *Illustrative Proof of Universal Approximation Theorem*, https://medium.com/hackernoon/illustrative-proof-of-universal-approximation-theorem-5845c02822f6, (kolovoz 2020.).

[20] Sayantini Nag, *Keras vs TensorFlow vs PyTorch : Comparison of the Deep Learning Frameworks*, https://www.edureka.co/blog/keras-vs-tensorflow-vs-pytorch/#introduction, (kolovoz 2020.).

[21] Elisa Negrini, *Universal Approximation Theorem, G. Cybenko*, https://users.wpi.edu/~msarkis/BrownBag/Elisa1.pdf, (kolovoz 2020.).

[22] Sinno Jialin Pan i Qiang Yang, *A survey on transfer learning*, IEEE Transactions on knowledge and data engineering **22** (2009), br. 10, 1345–1359.

[23] Ayush Pant, *Introduction to Machine Learning for Beginners*, https://towardsdatascience.com/introduction-to-machine-learning-for-beginners-eed6024fdb08, (kolovoz 2020.).

[24] Maria MP Petrou i Costas Petrou, *Image processing: the fundamentals*, John Wiley & Sons, 2010.

[25] Bharath Raj, *A Simple Guide to the Versions of the Inception Network*, https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202/, (kolovoz 2020.).

[26] Adrian Rosebrock, *Blur detection with OpenCV*, https://www.pyimagesearch.com/2015/09/07/blur-detection-with-opencv/, (kolovoz 2020.).

[27] Lou Ross i John C Russ, *The image processing handbook*, Microscopy and Microanalysis **17** (2011), br. 5, 843.

[28] Sebastian Ruder, *An overview of gradient descent optimization algorithms*, https://ruder.io/optimizing-gradient-descent/, (kolovoz 2020.).

[29] Dipanjan (DJ) Sarkar, *A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning*, https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a, (kolovoz 2020.).

[30] Aditya Sharma, *Blur-and-Clear-Classification*, https://github.com/aditya9211/Blur-and-Clear-Classification, (kolovoz 2020.).

[31] Koustubh Sharma, *ResNet, AlexNet, VGGNet, Inception: Understanding various architectures of Convolutional Networks*, https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/, (kolovoz 2020.).

[32] Jordi Torres, *Convolutional Neural Networks for Beginners using Keras and TensorFlow 2*, https://towardsdatascience.com/convolutional-neural-networks-for-beginners-using-keras-and-tensorflow-2-c578f7b3bf25, (kolovoz 2020.).

[33] Manuela Veloso, *Perceptrons and Neural Networks*, https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15381-f01/www/handouts/110601.pdf, (kolovoz 2020.).

[34] Ian Young, Jan Gerbrands, Lucas Van Vliet, Cip data Bibliotheek, Den Haag, Young Theodore, Gerbrands Jacob, Van Vliet i Lucas Jozef, *Fundamentals Of Image Processing*, (2004).

Sažetak

Za čovjeka važnu ulogu u životu ima ono što vidi, kao i činjenica da to što vidi može zabilježiti slikom. Tehnologija je dovela do toga da digitalne slike postaju sve bolje i bolje, čime je došlo i do potrebe za različitim mogućnostima uređivanja koja se nad slikama mogu izvršavati. To je računalne stručnjake potaknulo da omoguće izvedbu svih traženih zahtjeva, od kojih su danas najpoznatije svakojake detekcije objekata, popravci slika i slično. Upravo se takvim problemom bavi ovaj rad. Da bi se lakše detektirao objekt na slici ili samo odvojila dobra slika od onih lošijih po pitanju kvalitete i čistoće, kroz rad se istražuje koja bi metoda bila najučinkovitija za takvo izdvajanje te za klasifikaciju je li slika zamućena ili nije.

Ako se sliku reprezentira kao funkciju koja prezentira intenzitet svjetla na danom mjestu, onda se računanjem diskretnog Laplasijana takve funkcije mogu detektirati ona mjesta na slici gdje dolazi do naglih promjena intenziteta svjetla i na tim se mjestima detektira rub nekog objekta na slici. Ukoliko je takvih rubova puno, moglo bi se zaključiti da je slika čista s jasno definiranim prijelazima intenziteta koji predsatvljaju „granice“ između objekata. U suprotnom, ukoliko je rubova malo, slika se smatra zamućenom. Konačno, kao mjera klasifikacije je li slika zamućena ili nije koristi se varijanca Laplasijana kojom se dobiva srednje odstupanje intenziteta sive boje. Ukoliko je varijanca mala, odstupanja u intenzitetima sive boje su mala, dakle, rubova je također malo i slika je mutna. U slučaju visoke varijance, slika je čista. Problem kod ove metode predstavlja prag kojeg treba postaviti i koji će prevagnuti je li slika mutna ili nije.

Maksimiziranjem točnosti, za slike dimenzija 224×224 točnost na skupu za treniranje iznosi 81.78% uz graničnu vrijednost varijance 1100. Na skupu za testiranje, uz tu graničinu vrijednost varijance, dobivena točnost iznosi 74.59%. Ukoliko se ista metoda testira na istim slikama, ali dimenzija 96×96, točnost koja se dobije na skupu za treniranje je 76.20%, uz graničnu varijancu 2900. Na skupu za testiranje, točnost iznosi 76.28% Na temelju točnosti na skupu za treniranje zaključuje se da, što je slika većih dimenzija, tj. što je bliža izvornoj veličini, to će točnost biti veća, a granična vrijednost za varijancu bit će manja. Najbolje bi zato bilo da se metoda primijeni na slike izvorne veličine. Bez obzira na veličine slika na koje je metoda primijenjena, ona se pokazuje dobrom, tj. uspješnom, a služi i kao usporedba za ostale metode.

Sljedeća metoda jesu obične neuronske mreže koje su u posljednje vrijeme postigle popularnost dajući odlične rezultate za razne probleme. Kako je slika dvodimenzionalna, a neuronska mreža kao ulaz prima vektore, matricu svake slike bilo je potrebno rastegnuti u vektor pa je na takvom skupu podataka učena neuronska mreža s jednim skrivenim slojem. Pokazalo se da se svaka funkcija može dovoljno dobro aproksimirati neuronskom mrežom sa samo jednim skrivenim slojem pa se koristi takva arhitektura mreže. Pretpostavka je bila da rezultati ove metode neće biti dobri, jer se sliku mora rastegnuti čime se gube značajne veze između susjednih piksela koji u vektoru više nisu susjedni. Provedena testiranja potvrđuju pretpostavku. Na skupu za testiranje ne dobiva se dobar rezultat. Točnost od približno 67% se smatra niskom, a puno je niža i od one koja se dobiva primjenom varijance Laplasijana.

Zbog takvih rezultata razvijen je poseban oblik neuronskih mreža, a to su konvolucijske neuronske mreže. Ove mreže kao ulaz primaju višedimenzionalne ulaze, pa tako primaju i cijelu sliku. Kao što im i ime govori, glavna je operacija konvolucija kojom se izdvajaju i detektiraju bitne značajke za rješavanje problema. Da daju bolje rezultate pokazalo je testiranje na istom skupu podataka s konačnom točnošću na skupu za testiranje od otprilike 82%. Kako su slike dimenzija 96×96, ovo se može smatrati visokom točnošću, jer se smanjivanjem dimenzija slike algoritam zakida za neke bitne prostorne značajke, kao i kada se u prethodnom modelu slika rastezala. Postignuta točnost je visoka, ali je li i maksimalna?

Zadnja obrađena metoda pokazala je moguće dobiti bolje rezultate. Prijenosnim se učenjem postiže više poboljšanja i pojednostavljenja poput smanjenja količine podataka potrebne za treniranje, ubrzanje treniranja i postizanje boljih konačnih rezultata. Razvili su se mnogi oblici konvolucijskih neuronskih mreža koje se koriste i čije se težine unaprijed postave na vrijednosti dobivene treniranjem na puno većem skupu podataka. Stoga se očekuje da je tako postavljenim težinama algoritam već nešto naučio pa će učenje biti brže i uspješnije. Treniranjem i testiranjem na manjem skupu podataka dobiju se dosta dobri rezultati. Pokazalo se da je moguće dobiti točnost i iznad 90% što nadmašuje sve prijašnje rezultate. Modeli ovdje ipak nisu testirani na istom skupu kao i prijašnji, ali oni svejedno pokazuju da je poboljšanje moguće.

Konačno, uspjelo se pokazati da postoje metode strojnog učenja koje uspješno rješavaju problem klasifikacije, ali ako se želi primijeniti najlakša, a dobra metoda, dovoljno je primijeniti Laplaceov pristup na sliku u sivim tonovima, dobro postaviti graničnu vrijednost i brzo klasificirati zamućenu sliku. Ukoliko se želi iskoristiti neka moderinija strojna metoda, dovoljno je naučiti običnu konvolucijsku neuronsku mrežu s nekoliko skrivenih slojeva za postizanje dobrih rezultata. Dodatno, želi li se koristiti napredna arhitektura, mogu se koristiti aktualno uspješne konvolucijske mreže, poput AlexNeta, ResNeta i slično, ali uz korištenje unaprijed treniranih težina koje su uvijek javno dostupne.

Summary

For a person, what he sees plays an important role in life, as well as the fact that what he sees can be recorded in an image. Technology has led to digital images getting better and better, leading to the need for different editing options that can be executed on images. This prompted computer experts to enable the performance of all the requirements, of which the most well-known today are all kinds of object detections, image corrections and the like. This is exactly the problem that this paper deals with. In order to more easily detect an object in an image or just separate a good image from those that are worse in terms of quality and purity, the paper investigates which method would be most effective for such separation and to classify whether the image is blurred or not.

If the image is represented as a function that presents the intensity of light in a given place, then by calculating the discrete Laplacian of such functions, there can be detected the places in the image where there are sudden changes in light intensity. Those places are detected ad the edges of an object in the image. If there are many such edges, it could be concluded that the image is clear with clearly defined intensity transitions that represent ”boundaries” between objects. Otherwise, if the edges are small, the image is considered blurred. Finally, as a measure of classification whether the image is blurred or not, the variance of Laplacian is used to obtain the mean deviation of the gray color intensity. If the variance is small, the deviations in gray intensities are small, therefore, the number of edges is also small and the image is blurred. In the case of high variance, the image is clear. he challenge in utilizing this method is to set the classification threshold to classify which images are blurry and which are not.

By maximizing accuracy, for 224× 224 images, the accuracy on the training set is 81.78% with a variance threshold of 1100. On the test set, with this variance threshold, the accuracy obtained is 74.59%. If the same method is tested on the same images, but with dimensions 96×96, the accuracy obtained on the training set is 76.20%, with variance value of 2900. At the test set, the accuracy is 76.28%. Based on the accuracy at the training set, it can be concluded that the larger the image, ie. the closer to the original size, the higher the accuracy, and the lower the limit value for variance. It would therefore be best to apply the method to the original sizes of images. Regardless of the image sizes to which the method is applied, it proves to be good and successful, and also serves as a comparison for other

methods.

The next method are neural networks which have recently gained popularity by giving excellent results for various problems. As the image is two-dimensional, and the neural network receives vectors as input, the matrix of each image had to be stretched into a vector, so a neural network with one hidden layer was learned on such a data set. It has been shown that any function can be approximated well enough by a neural network with only one hidden layer, so such a network architecture is used. The assumption was that the results of this method would not be good because the image had to be stretched thus losing significant links between adjacent pixels that are no longer adjacent in the vector. The tests performed confirm the assumption. A good result is not obtained at the test set. The accuracy of approximately 67% is considered low, and is much lower than that obtained by tresholding the variance of the Laplacian of an image.

Due to such results, a special form of neural networks has been developed, and these are convolutional neural networks. These networks receive multidimensional inputs as input, and thus receive the whole picture. As their name suggests, the main operation is convolution, which highlights and detects essential troubleshooting features. To give better results, testing on the same data set showed a final accuracy on the test set of approximately 82%. As the images are 96×96, this can be considered high accuracy, because by reducing the image dimensions, the algorithm is deprived of some important spatial features, as when the image was stretched in the previous model. The achieved accuracy is high, could it be improved?

The last processed method showed it is possible to get better results. Transfer learning achieves more improvements and simplifications such as reducing the amount of data needed to train, speeding up training, and achieving better end results. Many forms of convolutional neural networks have been developed that are used and whose weights are previously trained on a much larger data set. Therefore, it is expected that with the weights set in this way, the algorithm has already learned something, so learning will be faster and more successful. By training and testing on a smaller data set, quite good results are obtained. It turned out that it is possible to get accuracy above 90%, which surpasses all previous results. The models here have not been tested on the same set as the previous ones, but they still show that improvement is possible.

Finally, it has been shown that there are machine learning methods that successfully solve the classification problem. If the easiest, but rather good, method is to be applied, it is enough to apply Laplace method on images in gray scale, set the limit value well and quickly classify the blurred images. If one wants to use some modern machine method, it is enough to learn a simple convolutional neural network with several hidden layers to achieve good results. Additionally, if one wants to use an advanced architecture, it can use currently successful convolutional networks, such as AlexNet, ResNet and the like, but with the use of pre-trained weights that are always publicly available.

Životopis

Petra Jambriško rođena je 07.06.1997. godine u Varaždinu i svoje je djetinjstvo provela u malenom mjestu zvanom Donje Vratno. Osnovnu školu Vinica pohađala je u istoimenoj općini i od ranih je dana pokazivala svoje interese prema matematici i informatici sudjelovanjem na svim razinama natjecanja.

Potom je pohađala prirodoslovno-matematički smjer Prve gimnazije Varaždin gdje je, osim za matematiku i informatiku, interese pokazala i za hrvatski jezik te logiku sudjelovajući na državnim natjecanjima iz tih predmeta. Bila je član u ekipnom natjecanju Američke informatičke lige, a svake su se godine plasirali na finalno natjecanje koje se održavalo u SAD-u. Kako je bila član dramske družine Theatron u svojoj školi, u trećem je razredu sudjelovala na festivalu legendi u Portugalu.

Nakon svih srednjoškolskih avantura, 2015. godine upisala je preddiplomski studij matematike na Prirodoslovno-matematičkom fakultetu Sveučilišta u Zagrebu, a 2018. godine upisala je diplomski studij računarstva i matematike. Uz uspješno polaganje kolegija na fakultetu, sudjelovala je na dvjema razmjenama mladih, jednoj u Slovačkoj pod imenom *Building Bridges* i jednoj u Rumunjskoj pod imenom *Ready for the Next Age*.

Svoje ambicije planira širiti dalje u području računarstva te kroz rad koristiti znanja koja je stekla studirajući na Prirodoslovno-matematičkom fakultetu.