Tehnike dubinskog učenja za klasifikaciju peludi

Poljarević, Petar

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni preddiplomski studij matematike i računarstva

Petar Poljarević

Tehnike dubinskog učenja za klasifikaciju peludi

Završni rad

Osijek, 2021.

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni preddiplomski studij matematike i računarstva

Petar Poljarević

Tehnike dubinskog učenja za klasifikaciju peludi

Završni rad

Mentor: doc. dr. sc. Slobodan Jelić

Osijek, 2021.

Sažetak

U ovome radu bavit ćemo se konvolucijskim neuronskim mrežama te njihovom potencijalnom primjenom u klasifikaciji peludnih čestica. Budući da ljudi mogu biti alergični na neke specifične vrste peludi, važno je biti u mogućnosti prepoznati vrstu koja se nalazi u zraku. Stoga posežemo za tehnikama klasifikacije dubinskog učenja - preciznije, konvolucijskim neuronskim mrežama.

Prvi dio rada definira slojeve koji izgrađuju konvolucijske neuronske mreže, demonstrira pojedine slojeve na manjem primjeru (ukoliko je to potrebno) te objašnjava njihovu važnost u praksi.

Drugi dio rada demonstrira primjenu jedne konvolucijske neuronske mreže na nekom stvarnom primjeru, odnosno na skupu podataka dobivenom laserskim skeniranjem peludnih čestica te opisuje rezultate klasifikacije.

Ključne riječi

Konvolucijska neuronska mreža, dubinsko učenje, klasifikacija, pelud, višeklasna klasifikacija, binarna klasifikacija.

Abstract

In this paper, we will observe convolutional neural networks and their potential appliance in pollen grain classification. Since people can be allergic to some specific pollen species, it is extremely important to be able to recognize the species of airborne pollen grains. This is why we reach out for deep learning techniques - more precisely, convolutional neural networks.

The first part of this paper defines the layers used for building convolutional neural networks, demonstrates the work of each layer on a small example (if needed) and explains their practical importance.

The second part of this paper demonstrates how a convolutional neural network works on a real-life example, that is, on a data set obtained by laser-scanning airborne pollen grains, and then describes the classification results.

Key words

Convolutional neural network, deep learning, classification, pollen, multiclass classification, binary classification.

Sadržaj

Uvod

Peludne alergije su jedan od najčešćih i najrasprostranjenijih oblika alergija na svijetu. Simptomi variraju od manje ozbiljnih kao što su kihanje i curenje nosa pa sve do nesvjestice i problema s disanjem (astmom). Svaka biljna vrsta proizvodi drugačiju vrstu peludi, a ljudi mogu biti alergični na neke specifične vrste, dok im druge uopće ne škode - upravo zato je važno biti u mogućnosti prepoznati o kojoj vrsti peludi se radi dok je pelud još u zraku.

Kako bismo postigli ovo, potrebne su nam dvije stvari: stroj za "mjerenje" peludi, tj. svojstava peludnih čestica te računalni algoritam koji će na osnovu tih svojstava moći zaključiti o kojoj vrsti peludi se radi. Što se tiče stroja za mjerenje, jedan od poznatijih je Rapid-E uređaj[1], koji pomoću lasera mjeri raspršenost svjetla i signale fluorescencije peludnih čestica. Jednom kada smo prikupili potrebne podatke i ručno ih klasificirali, pozivamo u pomoć algoritme iz strojnog učenja, koji će na osnovu prikupljenih podataka pokušati naučiti prepoznavati različite vrste peludi te primijeniti naučene informacije pri klasificiranju novih, neviđenih podataka.

Budući da su podaci na kojima naš algoritam uči, u neku ruku, slike, moramo biti u mogućnosti sačuvati informacije o okolini svakog piksela na slici, tzv. *spacijalne* informacije, jer one u velikoj mjeri pridonose razumijevanju "konteksta" svakog piksela. U ovom zadatku nam jednostavni algoritmi iz strojnog učenja (npr. Perceptron[2] ili logistička regresija[3]), pa čak ni jednostavnije tehnike dubinskog učenja (eng. *deep learning*) kao što su obične neuronske mreže, neće biti od prevelike koristi jer se kod njih spacijalne informacije izgube već na ulazu. Zato za rješavanje ovog problema idemo korak dalje te odabiremo konvolucijske neuronske mreže - one se i inače koriste za klasificiranje slika otkako je 2012. godine poznata konvolucijska neuronska mreža *AlexNet* postigla najbolji rezultat na natjecanju *ImageNet Large Scale Visual Recognition Challenge* (vidi [1] i [2]).

[1] `http://www.plair.ch/Rapid-E.html`
[2] `https://en.wikipedia.org/wiki/Perceptron`
[3] `https://en.wikipedia.org/wiki/Logistic_regression`

1 Konvolucijska neuronska mreža

Konvolucijske neuronske mreže su u velikoj mjeri inspirirane načinom na koji ljudski mozak prepoznaje različite objekte: kada primimo ulaz pomoću osjetila vida, prvo tražimo određene značajke za koje znamo da ih neka određena klasa objekata mora imati, zatim sažimamo informacije o tome ima li objekt pred nama određene značajke ili nema te na kraju na temelju toga svrstavamo objekt u onu klasu koja nam se čini najvjerojatnijom.

1.1 Konvolucijski sloj

Glavna razlika između običnih i konvolucijskih neuronskih mreža nalazi se u konvolucijskim slojevima. Upravo oni su odgovorni za očuvanje spacijalnih informacija kod konvolucijskih neuronskih mreža. Njihov je zadatak primijeniti određeni broj tzv. filtera na ulaznu sliku, odnosno primijeniti operaciju unakrsne korelacije (eng. *cross-correlation*) na ulaznoj slici.

Konvolucijski slojevi su ime dobili po matematičkoj operaciji konvolucije, no operacija koja se u tim slojevima izvršava je malo drugačija od konvolucije u matematici. Unatoč tome, kada govorimo o konvoluciji u kontekstu neuronskih mreža, mislimo upravo na unakrsnu korelaciju. Slijede definicije obiju operacija.

Definicija 1.1 (Konvolucija neprekidnih funkcija)
Neka su $f, g : \mathbb{R} \to \mathbb{R}$ neprekidne. Konvoluciju funkcija f i g definiramo kao

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

Definicija 1.2 (Konvolucija diskretnih funkcija)
Neka su $x, w : \mathbb{Z} \to \mathbb{R}$. Konvoluciju funkcija x i w definiramo kao

$$(x * w)(n) := \sum_{m=-\infty}^{\infty} x(m) w(n - m).$$

Definicija 1.3 (Unakrsna korelacija diskretnih funkcija)
Neka su $x, w : \mathbb{Z} \to \mathbb{R}$. Unakrsnu korelaciju funkcija x i w definiramo kao

$$(x * w)(n) := \sum_{m=-\infty}^{\infty} x(m) w(n + m).$$

Primijetimo da konvolucija i unakrsna korelacija imaju istu oznaku. Da bi se izbjegle zabune, obično se naglasi o kojoj operaciji je riječ, a i koriste se u različitim kontekstima (vidi [3] i [4]).

Definicija 1.4 (Unakrsna korelacija realnih matrica)
Neka su $K \in \mathbb{R}^{M \times N}$ i $I \in \mathbb{R}^{P \times Q}$, $M \leq P$, $N \leq Q$. Unakrsnu korelaciju matrica I i K definiramo kao

$$R(i, j) = (I * K)(i, j) := \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i + m, j + n) K(m + 1, n + 1), \tag{1}$$

pri čemu

$$i = 1, ..., P - M + 1, \qquad j = 1, ..., Q - N + 1.$$

Matrice I i K iz prethodne definicije zvat ćemo **ulaz** (eng. *input*), odnosno **jezgra** (njem. *kernel*) ili **filter**. Matricu $R \in \mathbb{R}^{(P-M+1)\times(Q-N+1)}$ zvat ćemo rezultatom konvolucije. Kao što je već ranije spomenuto, u kontekstu neuronskih mreža, unakrsnu korelaciju zovemo konvolucijom. Nadalje u tekstu ćemo koristiti termin *konvolucija* za unakrsnu korelaciju.

Sad kada znamo što je konvolucija, preostaje nam na primjeru pokazati kako ona radi s ulaznim podacima, tj. slikama.

Zamislimo za početak crno-bijelu sliku (ne *grayscale*) širine i visine 5 piksela. To je zapravo matrica dimenzija 5×5 kojoj su elementi 0 i 1 - ta slika će biti naš ulaz I. Sada zamislimo realnu matricu dimenzija 3×3 - ona će biti naš filter K. Na sljedeće dvije slike nalazit će se naš ulaz i naš filter.

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Slika 1: Primjer ulaznog podatka

1	0	1
0	1	0
1	0	1

Slika 2: Primjer konvolucijskog filtera

Naš rezultat će očigledno biti matrica dimenzija 3×3. Izračunajmo nekoliko njegovih elemenata koristeći formulu (1), npr. elemente na pozicijama $(1,1)$, $(1,2)$ i $(2,1)$.

$$R(1,1) = (I * K)(1,1) = \sum_{m=0}^{3-1} \sum_{n=0}^{3-1} I(m+1, n+1)K(m+1, n+1) = 4$$

$$R(1,2) = (I * K)(1,2) = \sum_{m=0}^{3-1} \sum_{n=0}^{3-1} I(m+1, n+2)K(m+1, n+1) = 3$$

$$R(2,1) = (I * K)(2,1) = \sum_{m=0}^{3-1} \sum_{n=0}^{3-1} I(m+2, n+1)K(m+1, n+1) = 2$$

Kada napravimo ovo sa svim elementima od $(1,1)$ do $(3,3)$, dobivamo sljedeći rezultat:

4	3	4
2	4	3
2	3	4

Slika 3: Rezultat konvolucije gornjih matrica

Što ako radimo sa slikama u boji, a ne sa crno-bijelima? U tom slučaju naš ulaz nije jedna matrica, već uređena trojka (I_1, I_2, I_3) matrica jednakih dimenzija - svaka matrica odgovara jednoj boji, tj. jednom RGB kanalu[4] te poprima cjelobrojne vrijednosti od 0 do 255 (ukoliko nismo prethodno skalirali ulazne podatke). Slično, naš filter postaje uređena trojka (K_1, K_2, K_3) matrica jednakih dimenzija te kažemo da naš filter ima tri **ulazna kanala**.

Podrezultate konvolucije R_1, R_2 i R_3 tada dobivamo na sličan način kao i u gornjem primjeru: uzmemo matrice I_1 i K_1, na njima provedemo konvoluciju i time dobijemo matricu R_1. Zatim ponovimo proces za matrice I_2 i K_2, odnosno I_3 i K_3 te time dobijemo rezultat R_2, odnosno R_3. Krajnji rezultat konvolucije R dobijemo tako što jednostavno zbrojimo matrice R_1, R_2 i R_3.

Kako bismo do kraja izgradili jedan konvolucijski sloj, preostaje nam povećati broj filtera. Broj filtera u konvolucijskim slojevima često odgovara nekoj potenciji broja 2 (npr. 64, 128 ili 256), a svaki od filtera "traži" neku određenu značajku na ulaznim podacima. Npr. ako želimo da naša mreža može prepoznati je li objekt na slici neka vrsta automobila, tada će jedan filter tražiti kotače, drugi će tražiti svjetla, treći će tražiti vjetrobransko staklo, četvrti će tražiti ulazna vrata itd. Svaki od ovih filtera imat će zaseban rezultat, što znači da ako trenutni konvolucijski sloj ima 16 filtera, svaki od filtera će se primijeniti na ulaznom podatku i proizvesti zaseban rezultat - time naš konvolucijski sloj kao rezultat daje 16 matrica ili kažemo da sloj ima 16 **izlaznih kanala**.

Dakle, ako naš konvolucijski sloj (nazovimo ga `Conv1`) na ulazu prima sliku u boji, imat će tri ulazna kanala, što znači da će svaki od filtera biti uređena trojka matrica jednakih dimenzija. Nadalje, recimo da isti sloj ima 16 filtera - dakle 16 uređenih trojki matrica jednakih dimenzija - tada će sloj `Conv1` kao rezultat dati 16 matrica jednakih dimenzija, tj. imat će 16 izlaznih kanala.
Ako bismo odmah nakon ovog sloja dodali još jedan konvolucijski sloj `Conv2` sa 64 filtera, tada bi taj sloj morao imati 16 ulaznih kanala, tj. svaki od 64 filtera bio bi uređena 16-orka matrica jednakih dimenzija i taj bi sloj kao rezultat dao 64 matrice jednakih dimenzija.
Ako bismo sada dodali još jedan konvolucijski sloj `Conv3`, taj bi sloj morao imati 64 ulazna kanala itd.

Dodatne informacije o konvolucijskim slojevima kao što su *stride* i *padding* mogu se pronaći na [5] i [6].

[4] `https://en.wikipedia.org/wiki/Channel_(digital_image)`

1.2 *Pooling* sloj

Budući da konvolucijske neuronske mreže rade s podacima iznimno velikih dimenzija, važno je biti u mogućnosti sažeti te podatke, a da pritom ne izgubimo važne informacije o istima. Iz tog razloga uvodimo *pooling* slojeve ili **slojeve sažimanja**. Njihov je zadatak uzeti vrlo male grupe parametara iz ulaznog podatka, tj. male grupe piksela iz slike, na njima primijeniti neku od funkcija za sažimanje i kao rezultat dobiti matricu složenu od tih malih grupa. Na taj smo način iz ulaznih podataka izbacili manje važne informacije, a uz to i smanjili broj parametara koje naš podatak, tj. slika, sadrži - a to značajno ubrzava rad mreže i sprječava prenaučenost modela.

Dvije najčešće funkcije sažimanja su sažimanje po maksimumu (eng. *Max Pooling*) i sažimanje po aritmetičkoj sredini (eng. *Average Pooling*). Definirajmo ih.

Definicija 1.5 (Sažimanje po maksimumu)
Neka je $I \in \mathbb{R}^{P\times Q}$ *i neka su* $M, N \in \mathbb{N}$, $M \leq P$, $N \leq Q$. *Sažimanje matrice* I *po maksimumu s jezgrom dimenzija* $M \times N$ *definiramo kao*

$$P_{max}(i,j) := \max\left\{I\left[M(i-1)+m,\ N(j-1)+n\right] : \quad m = 1,...,M, \quad n = 1,...,N\right\}, \tag{2}$$

pri čemu

$$i = 1,...,\left\lfloor \frac{P}{M} \right\rfloor, \qquad j = 1,...,\left\lfloor \frac{Q}{N} \right\rfloor.$$

Definicija 1.6 (Sažimanje po aritmetičkoj sredini)
Neka je $I \in \mathbb{R}^{P\times Q}$ *i neka su* $M, N \in \mathbb{N}$, $M \leq P$, $N \leq Q$. *Sažimanje matrice* I *po aritmetičkoj sredini s jezgrom dimenzija* $M \times N$ *definiramo kao*

$$P_{avg}(i,j) := \frac{1}{MN}\sum_{m=1}^{M}\sum_{n=1}^{N} I\left[M(i-1)+m,\ N(j-1)+n\right], \tag{3}$$

pri čemu

$$i = 1,...,\left\lfloor \frac{P}{M} \right\rfloor, \qquad j = 1,...,\left\lfloor \frac{Q}{N} \right\rfloor.$$

Pokažimo sada na primjeru kako navedene funkcije rade.

Kao ulaz ćemo za primjer uzeti sljedeću matricu.

12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

Slika 4: Primjer ulaznog podatka

Neka je zadana jezgra dimenzija 2×2. Izračunajmo rezultat sažimanja po maksimumu, odnosno sažimanja po aritmetičkoj sredini za element na poziciji $(1, 1)$ koristeći formulu (2), odnosno (3).

$$\begin{aligned} P_{max}(1,1) &= \max\{I\left[2\cdot(1-1)+m,\ 2\cdot(1-1)+n\right]: \quad m=1,2, \quad n=1,2\} \\ &= \max\{I[1,1],\ I[1,2],\ I[2,1],\ I[2,2]\} \\ &= \max\{12,\ 20,\ 8,\ 12\} \\ &= 20 \end{aligned}$$

$$\begin{aligned} P_{avg}(1,1) &= \frac{1}{2\cdot 2}\sum_{m=1}^{2}\sum_{n=1}^{2} I\left[2\cdot(1-1)+m,\ 2\cdot(1-1)+n\right] \\ &= \frac{1}{4}\sum_{m=1}^{2}\sum_{n=1}^{2} I\left[m,\ n\right] \\ &= \frac{1}{4}\left(12+20+8+12\right) \\ &= 13 \end{aligned}$$

Kada napravimo ovo na svim elementima od $(1, 1)$ do $(2, 2)$, dobivamo sljedeće rezultate:

20	30
112	37

Slika 5: Rezultat sažimanja po maksimumu

13	8
79	20

Slika 6: Rezultat sažimanja po aritmetičkoj sredini

Podaci koje šaljemo *pooling* slojevima gotovo uvijek sadrže više od jednog kanala, no za razliku od konvolucijskih, *pooling* slojevi sadrže jednak broj ulaznih i izlaznih kanala. Sve što trebamo je jednostavno primjeniti istu funkciju na svaki kanal pojedinačno i to proglasiti rezultatom sažimanja. Dodatne informacije o *pooling* slojevima kao što su *stride* i *padding* mogu se pronaći na [5] i [6].

1.3 Potpuno povezani sloj

Potpuno povezani (eng. *fully connected*) slojevi su osnovne gradivne jedinice običnih neuronskih mreža te u konvolucijskim neuronskim mrežama funkcioniraju jednako kao i kod običnih, pa ovdje nećemo objašnjavati kako rade. O potpuno povezanim slojevima, odnosno običnim neuronskim mrežama može se više pročitati na [7]. Kao i kod običnih neuronskih mreža, broj neurona u posljednjem potpuno povezanom sloju odgovara broju klasa. No, za razliku od običnih, konvolucijske neuronske mreže u praksi imaju vrlo malen broj potpuno povezanih slojeva - najčešće jedan do tri.

1.3.1 Sloj za ravnanje

Budući da potpuno povezani slojevi kao ulaz primaju vektore, a ne uređene n-torke matrica, potrebno je dodati specijalan sloj odmah prije prvog potpuno povezanog sloja: sloj za ravnanje (eng. *flattening*). Zadatak ovog sloja je, kao što mu i ime govori, *izravnati* podatke koje je posljednji konvolucijski ili *pooling* sloj poslao, odnosno pretvoriti ih iz uređene n-torke matrica u vektor. U matematici se ova operacija zove **vektorizacija** (vidi [8]), a mnogi programski paketi specijalizirani za strojno učenje kao što su NumPy[5] i PyTorch[6] imaju ovu operaciju već implementiranu.

Napomena 1.1
Iako zbog ravnanja gubimo spacijalne informacije kod običnih neuronskih mreža, ovdje nemamo takav problem jer smo sve spacijalne informacije već izvukli iz ulaznog podatka korištenjem konvolucijskih i pooling slojeva.

1.4 Aktivacijska funkcija

Koliko god konvolucijskih i potpuno povezanih slojeva imali, ako ne koristimo neku od aktivacijskih funkcija (vidi [9] i [10]), naš model će na kraju biti samo polinom s velikim brojem varijabli, a to pomalo ruši poantu neuronskih mreža. Sličan učinak dobijemo ako kao aktivacijsku funkciju odaberemo funkciju identiteta. Kod običnih neuronskih mreža, najčešće se za aktivacijsku funkciju uzima funkcija pod nazivom Sigmoid (vidi [11]).

Kod konvolucijskih neuronskih mreža, najčešći odabir za aktivacijsku funkciju je funkcija pod nazivom **linearna ispravljačka jedinica** (eng. *Rectified Linear Unit*) ili kraće, **ReLU** (vidi [12]). Definirajmo ju.

Definicija 1.7 (Linearna ispravljačka jedinica)
Linearnu ispravljačku jedinicu definiramo kao

$$f : \mathbb{R} \to \mathbb{R}, \quad f(x) = \max(0, x) = \begin{cases} x, & x > 0, \\ 0, & inače. \end{cases}$$

[5] `https://numpy.org/`
[6] `https://pytorch.org/`

Ponekad ova funkcija može imati jedan vrlo nezgodan učinak, a to je da, zbog velikog broja jako negativnih parametara u modelu, velik broj gradijenata bude jednak nuli dokle god model uči - to zovemo **problem *umirućih* neurona**. U takvim se situacijama uvodi jedna vrlo slična funkcija, tzv. ***propuštajuća* ReLU funkcija** (eng. *Leaky ReLU*) (vidi [13]). Definirajmo ju.

Definicija 1.8 (Propuštajuća ReLU funkcija)
Neka je $a \in \mathbb{R}^+$. Propuštajuću ReLU funkciju definiramo kao

$$f : \mathbb{R} \to \mathbb{R}, \quad f(x) = \begin{cases} x, & x > 0, \\ 0.01x, & inače. \end{cases}$$

Ova funkcija vrlo učinkovito rješava problem *umirućih* neurona, no obično ju se koristi samo ukoliko je to neophodno jer nam manja količina *umirućih* neurona neće smetati, nego će samo blago ubrzati učenje modela.

1.5 Funkcija cilja

Funkcija cilja (eng. *loss function*) ili **funkcija troška** (eng. *cost function*) je posljednji sloj svake neuronske mreže. Nakon što smo obavili velike količine računanja na ulaznom podatku, moramo vidjeti koliko dobro naš model radi na trenutnom ulaznom podatku - upravo to je zadatak funkcije cilja (vidi [14]).

Budući da se u strojnom učenju funkcija cilja obično pokušava minimizirati, možda bi zapravo bilo bolje reći da nam ona govori koliko *loše* naš model radi. Minimizacija se može vršiti raznim metodama, a najpoznatija od njih je **gradijentna metoda** ili **gradijentni spust** (eng. *gradient descent*) (vidi [15]).

Funkcijâ cilja postoji mnogo, a odabir iste u potpunosti ovisi o problemu s kojim radimo. Ukoliko naš model svrstava podatke u samo dvije klase (0 i 1), tj. radimo s binarnim klasifikatorom (vidi [16]), koristimo samo jednu funkciju cilja.

Ukoliko želimo svrstati podatke u više od dvije klase, tj. radimo s višeklasnim klasifikatorom (vidi [17]), onda na odabranu funkciju cilja na kraju još primijenimo funkciju po imenu **Softmax** (vidi [18]) kako bi naš rezultat predstavljao vjerojatnost da ulazni podatak pripada određenoj klasi. Definirajmo Softmax funkciju.

Definicija 1.9 (Softmax funkcija)
Neka je $K \in \mathbb{N}$ zadani broj klasa. Softmax funkciju $\sigma : \mathbb{R}^K \to [0,1]^K$ definiramo kao

$$\sigma(\mathbf{z})_i := \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}, \quad i = 1, ..., K, \quad \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K.$$

2 Klasificiranje peludnih čestica

Sad kada znamo kako konvolucijske neuronske mreže funkcioniraju, vrijeme je da pokažemo kako rade na nekom stvarnom primjeru. U tu svrhu napisan je projekt koristeći programski jezik Python - preciznije, programski paket PyTorch[7].

2.1 Ulazni podaci

Kao što je već spomenuto u uvodu, podatke koje treba klasificirati dobili smo koristeći Rapid-E uređaj koji je postavljen u Osijeku. Uređaj radi tako da usisa određenu količinu zraka u kojem se nalaze peludne čestice te zatim laserom "skenira" usisani zrak i mjeri nekoliko svojstava kao što su raspršenost svjetla, signali fluorescencije te životni vijek fluorescencije.

Dobiveni podaci su dodatno procesuirani spajanjem spomenutih svojstava te vektorizacijom i dopunjavanjem do zajedničke dimenzije nulama. Kao što je to obično situacija kod strojnog učenja, podaci su dodatno skalirani kako bismo radili s manje raširenim podacima. Konkretno u projektu je korišten `StandardScaler` iz programskog paketa `scikit-learn` (vidi [19] za dodatno objašnjenje).

Podaci su sadržavali osam vrsta peludi:

- Alnus Glutin
- Ambrosia Artemisifolia
- Artemisia Vulgaris
- Betula Pendula
- Cedrus
- Corylus Avelana
- Quercus
- Urtica Diocia

2.2 Arhitektura neuronske mreže

Tijekom izrade projekta korišteno je desetak različitih arhitektura konvolucijskih neuronskih mreža, no najbolje rezultate dala je arhitektura `CNN_Odin`, djelomično inspirirana poznatom arhitekturom *AlexNet* (vidi [1] i [2]).

Bitna razlika je u tome što ova arhitektura koristi jednodimenzionalne konvolucijske i *pooling* slojeve (jer su joj ulazni podaci vektori), dok *AlexNet* koristi dvodimenzionalne (jer su joj ulazni podaci slike).

[7] `https://pytorch.org/`

Na sljedećim dviju slikama prikazan je detaljan opis arhitekture `CNN_Odin`.

Sloj	Broj filtera	Veličina jezgre	Stride	Padding	Aktivacijska funkcija
Conv 1	96	1 × 5	1	2	ReLU
Max Pool 1	-	1 × 2	2	-	-
Conv 2	256	1 × 5	1	2	ReLU
Max Pool 2	-	1 × 2	2	-	-
Conv 3	384	1 × 5	1	2	ReLU
Max Pool 3	-	1 × 2	1	-	-
Conv 4	384	1 × 3	1	1	ReLU
Max Pool 4	-	1 × 2	2	-	-
Conv 5	256	1 × 3	1	1	ReLU
Max Pool 5	-	1 × 2	2	-	-
Ravnanje	-	-	-	-	-

Slika 7: Konvolucijski dio arhitekture `CNN_Odin`

Sloj	Broj neurona	Aktivacijska funkcija
FC 1	4096	ReLU
FC 2	2048	ReLU
FC 3	8	Softmax

Slika 8: Potpuno povezani dio arhitekture `CNN_Odin`

2.3 Rezultati

2.3.1 Višeklasni klasifikator

Pokušaj klasifikacije ulaznih podataka višeklasnim klasifikatorom dao je relativno ujednačene rezultate: sveukupna točnost na neviđenim podacima kretala se između 60% i 70%, ovisno o hiperparametrima.

S druge strane, točnost za pojedinačne klase davala je vrlo mješovite rezultate (na neviđenim podacima): npr. klase *Alnus Glutin* i *Corylus Avelana* uglavnom su imale točnost između 15% i 35%, dok su klase *Cedrus* i *Quercus* znale dosegnuti točnost od čak 85%. Sljedeća

tablica prikazuje udio svake klase u sveukupnom skupu podataka te prosječnu točnost klasifikacije za tu klasu.

Vrsta	Udio	Točnost
Alnus Glutin	3.38 %	27 %
Ambrosia Artemisifolia	3.92 %	45 %
Artemisia Vulgaris	9.56 %	60 %
Betula Pendula	6.37 %	36 %
Cedrus	56.58 %	80 %
Corylus Avelana	5.55 %	27 %
Quercus	7.79 %	80 %
Urtica Diocia	6.88 %	43 %

Tablica 1: Rezultati višeklasne klasifikacije

Zbog ovakvih rezultata postavlja se pitanje imaju li neke vrste izraženija svojstva od drugih, odnosno može li se neke određene vrste bolje klasificirati binarnim klasifikatorom.

2.3.2 Binarni klasifikatori

Blagim modificiranjem posljednjeg potpuno povezanog sloja naše arhitekture dobiven je binarni klasifikator koji za unaprijed zadanu vrstu (npr. *Quercus*) klasificira ulazne podatke u dvije klase: zadanu klasu i sve ostale.

Budući da ovi klasifikatori rade s vrlo nebalansiranim skupovima podataka, važno je, osim točnosti na testnom skupu, provjeriti i točnost za pojedine klase, odnosno *true positive* (TP) i *true negative* (TN) vrijednosti (vidi [20]). Najbolji rezultati odabrani su tako da je točnost na neviđenim podacima što veća, a razlika između TP i TN vrijednosti što manja. Znalo se dogoditi da točnost i funkcija cilja imaju odlične vrijednosti, no razlika između TP i TN vrijednosti bila je toliko velika da se ta točnost jednostavno nije mogla smatrati relevantnom.

Sljedeća tablica prikazuje najbolje rezultate binarne klasifikacije za pojedinu vrstu, njihove udjele u cjeloukupnom skupu podataka te stopu učenja (LR) i broj epoha za koje su isti rezultati postignuti. Regularizacija je fiksirana na 1.0E-4 zbog bržeg traženja optimalne stope učenja, a veličina hrpe (eng. *batch*) postavljena je na 64. Za sam proces učenja korišten je optimizacijski algoritam *Adam*[8], a optimalna stopa učenja pronađena je pretraživanjem po rešetki[9] (eng. *grid search*).

[8] `https://en.wikipedia.org/wiki/Stochastic_gradient_descent#Adam`
[9] `https://en.wikipedia.org/wiki/Hyperparameter_optimization#Grid_search`

Vrsta	Udio	Točnost	TP	TN	LR	epohe
Alnus Glutin	3.38 %	75 %	76 %	75 %	7.5E-5	10
Ambrosia Artemisifolia	3.92 %	76 %	75 %	76 %	1.0E-4	6
Artemisia Vulgaris	9.56 %	81 %	78 %	81 %	5.0E-5	6
Betula Pendula	6.37 %	73 %	72 %	73 %	1.0E-5	8
Cedrus	56.58 %	82 %	83 %	82 %	2.5E-5	13
Corylus Avelana	5.55 %	71 %	73 %	70 %	5.0E-5	13
Quercus	7.79 %	93 %	94 %	93 %	5.0E-5	9
Urtica Diocia	6.88 %	72 %	73 %	72 %	5.0E-5	9

Tablica 2: Rezultati binarne klasifikacije

Važno je napomenuti način na koji je broj epoha odabran: prilikom završetka svake epohe, trenutni parametri modela spremljeni su u zasebnu vanjsku datoteku te su po završetku učenja svi redom učitani i testirani. Zatim je pronađen model s najboljom točnošću na neviđenim podacima, a da ima što manju razliku između TP i TN vrijednosti te je broj epoha odabran prema rednom broju modela kojeg smo proglasili najboljim. Zbog nebalansiranosti skupa podataka, nismo mogli jednostavno pustiti model da iskonvergira, nego smo ga na ovaj način morali "zaustaviti" kada pronađemo dovoljno dobar rezultat.

Na kraju vidimo da binarni klasifikatori rade puno bolje, no to automatski podrazumijeva korištenje osam neuronskih mreža umjesto jedne, a za svaku od njih još je potrebno pronaći i optimalne hiperparametre, što je zahtjevno i vremenski i memorijski. Iako smo za sve vrste dobili točnost od barem 70%, možemo primijetiti kako su neke vrste imale bolji rezultat - i to upravo one koje su imale najbolje rezultate prilikom višeklasne klasifikacije.

Zaključujemo da se neke vrste peludnih čestica mogu klasificirati prilično dobro, dok se preostale mogu klasificirati s manjom točnošću - razlog tome najvjerojatnije leži u premalenom broju dobrih reprezentanata određene vrste, a moguće je i da te vrste jednostavno nemaju dovoljno izražena svojstva da bismo ih uopće mogli klasificirati.

Literatura

[1] Review: AlexNet, CaffeNet — Winner of ILSVRC 2012 (Image Classification), Medium. Dostupno na: https://medium.com/coinmonks/paper-review-of-alexnet-caffenet-winner-in-ilsvrc-2012-image-classification-b93598314160 (14. rujan 2021.)

[2] AlexNet, Wikipedia. Dostupno na: https://en.wikipedia.org/wiki/AlexNet (14. rujan 2021.)

[3] Convolution, Wikipedia. Dostupno na: https://en.wikipedia.org/wiki/Convolution (15. rujan 2021.)

[4] Cross-correlation, Wikipedia. Dostupno na: https://en.wikipedia.org/wiki/Cross-correlation (15. rujan 2021.)

[5] Convolutional Networks, DeepLearningBook. Dostupno na: https://www.deeplearningbook.org/contents/convnets.html (15. rujan 2021.)

[6] A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way, Towards Data Science. Dostupno na: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (16. rujan 2021.)

[7] Artificial neural network, Wikipedia. Dostupno na: https://en.wikipedia.org/wiki/Artificial_neural_network (17. rujan 2021.)

[8] Vectorization (mathematics), Wikipedia. Dostupno na: https://en.wikipedia.org/wiki/Vectorization_(mathematics) (17. rujan 2021.)

[9] Activation function, Wikipedia. Dostupno na: https://en.wikipedia.org/wiki/Activation_function (17. rujan 2021.)

[10] Activation Functions in Neural Networks, Towards Data Science. Dostupno na: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 (17. rujan 2021.)

[11] Sigmoid function, Wikipedia. Dostupno na: https://en.wikipedia.org/wiki/Sigmoid_function (17. rujan 2021.)

[12] Rectifier (neural networks), Wikipedia. Dostupno na: https://en.wikipedia.org/wiki/Rectifier_(neural_networks) (17. rujan 2021.)

[13] Rectifier (neural networks), paragraph Leaky ReLU, Wikipedia. Dostupno na: https://en.wikipedia.org/wiki/Rectifier_(neural_networks)#Leaky_ReLU (17. rujan 2021.)

[14] Loss function, Wikipedia. Dostupno na: https://en.wikipedia.org/wiki/Loss_function (17. rujan 2021.)

[15] Gradient descent, Wikipedia. Dostupno na: https://en.wikipedia.org/wiki/Gradient_descent (17. rujan 2021.)

[16] Binary classification, Wikipedia. Dostupno na: https://en.wikipedia.org/wiki/Binary_classification (17. rujan 2021.)

[17] Multiclass classification, Wikipedia. Dostupno na: https://en.wikipedia.org/wiki/Multiclass_classification (17. rujan 2021.)

[18] Softmax function, Wikipedia. Dostupno na: https://en.wikipedia.org/wiki/Softmax_function (17. rujan 2021.)

[19] StandardScaler class, scikit-learn. Dostupno na: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (23. rujan 2021.)

[20] Confusion matrix, Wikipedia. Dostupno na: https://en.wikipedia.org/wiki/Confusion_matrix (24. rujan 2021.)