SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3948

Raspoznavanje objekata konvolucijskim neuronskim mrežama

Dario Smolčić

Zagreb, lipanj 2015.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod

Računalni vid je područje koje uključuje metode za dohvaćanje, obrađivanje i shvaćanje slika i općenito podataka velikih dimenzija te je zanimljivo područje računalne znanosti zbog mogućnosti široke primjene u današnjem svijetu. Jedna od podgrana ovog područja je raspoznavanje objekata.

Ljudi su sposobni prepoznati mnoštvo različitih objekata sa jako malo truda no za računala je to složen proces koji ima brojna ograničenja koja ljudi nemaju. Uzmimo u obzir da se slika u računalu reprezentira kao višedimenzionalni niz jačina svjetlosti. Promjene u prikazu objekta poput različite orijentacije, skaliranja, i osvijetljenja objekta su u digitalnim slikama prestavljene sa različitim podatcima. Objekt također može biti i zaklonjen. Dobar model raspoznavanja mora biti otporan na ove varijacije te je zato problem raspoznavanja objekata još uvijek neriješen i u zadnjih nekoliko desetljeća su razvijene brojne metode kojima se pokušava riješiti ovaj problem. Za razliku od pisanja klasičnih algoritama poput sortiranja brojeva za problem klasifikacije objekata nije očito kako bi se mogao napisati takav algoritam gdje su sve varijacije ulaza posebno obrađene u kodu. Zato se za klasifikaciju objekata koristi pristup usmjeren na podatke (engl. *data-driven approach*). Programu se da veliki broj ulaza sa velikom količinom primjera za svaku klasu te se razvije algoritam učenja koji učitava date primjere te uči o vizualnom prikazu svake klase. Takve programe nazivamo klasifikatorima.

U zadnjih nekoliko desetljeća su razvijeni različiti klasifikatori za što točnije prepoznavanje objekata. Među tim klasifikatorima su i umjetne neuronske mreže. Ispostavilo se da se sa dubokim neuronskim mrežama trenutno dobivaju najbolji rezultati za problem klasifikacije. Najkorišteniji oblik dubokih neuronskih mreža u račnunalnom vidu su konvolucijske neuronske mreže.

Cilj ovog rada je razviti implementaciju konvolucijske neuronske mreže za primjenu na osobnim računalima, optimirati hiperparametre mreže te vrednovati učinak naučene mreže. Razvijena mreža će se testirati na skupu MNIST rukom pisanih znamenki.

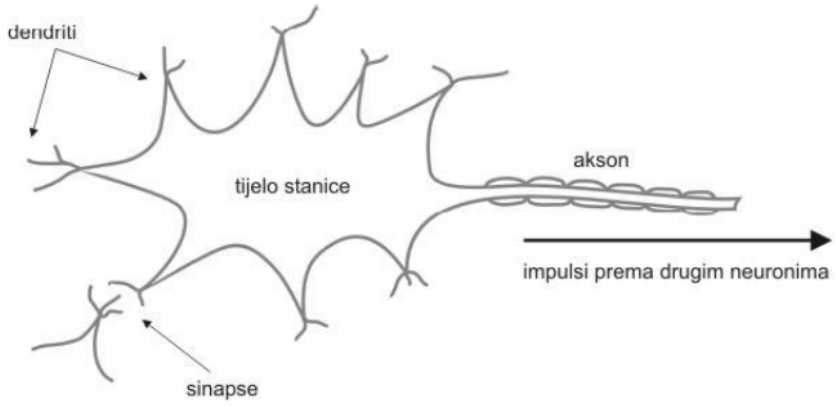
2. Neuronske mreže

Područje umjetnih neuronskih mreža (engl. *Artificial Neural Networks - ANN*) je prvotno bilo inspirirano sa modeliranjem biološkog živčanog sustava, a tek kasnije se počelo koristiti u sklopu strojnog učenja.

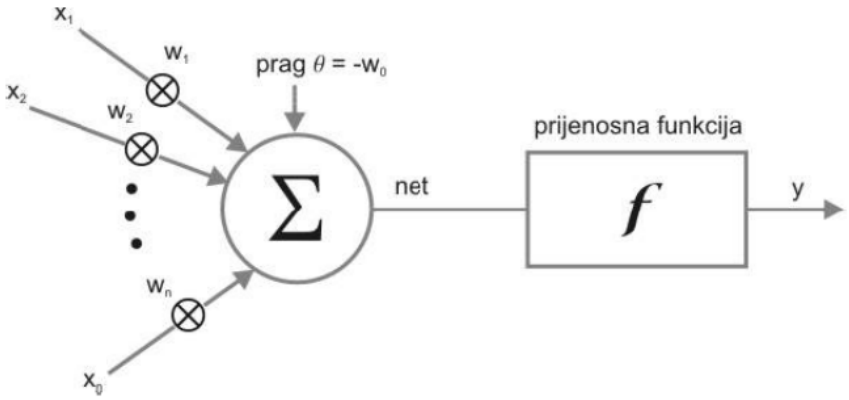
2.1. Neuron

Radi razumijevanja neuronske mreže potrebno je prvo razumijeti funkcioniranje jednog neurona. Ljudski živčani sustav se sastoji od otprilike 86 bilijona neurona koji su povezani sa 10^{14} do 10^{15} sinapsi. Svaki neuron dobiva svoje ulazne signale kroz dendrite i šalje izlazni signal kroz akson. Akson je sa sinapsama spojen sa dendritima drugih neurona. Na slici 2.1 možemo vidjeti izgled biološkog neurona.

U modelu umjetnog neurona signali koji putuju aksonom (npr. x_0) se množe sa sinaptičkim snagama dendrita(težinama) drugih neurona (npr. w_0). Ideja je da se sinaptičke snage mogu mijenjati sa učenjem te određuju utjecaj jednog neurona na drugi. Svaki neuron ima aktivacijsku funkciju koja uzima sumu umnoška ulaza neurona sa pripadnim težinama i praga (θ) te ih preslikava na izlaz neurona koji modelira signal



Slika 2.1: Biološki neuron



Slika 2.2: Umjetni neuron

na aksonu(y). Na slici 2.2 možemo vidjeti model umjetnog neurona.

Označimo ulaze sa $x_1, x_2, ..., x_n$ te njihove pripadne težine sa $w_1, w_2, ..., w_n$, i prag sa θ. Onda možemo izlaz neurona zapisati kao:

$$y = f(\sum_{i=1}^{n} x_i w_i + \theta) \tag{2.1}$$

Radi pojednostavljenja se često uzima oznaka w_0 umjesto θ te se dodaje jedan ulaz x_0 koji je stalno jednak 1. Sa ovom modifikacijom izlaz neurona se može izraziti kao:

$$y = f(\sum_{i=0}^{n} x_i w_i) \tag{2.2}$$

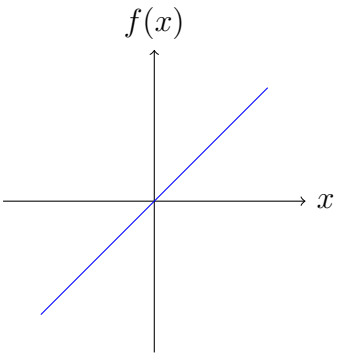
2.1.1. Aktivacijske funkcije

Postoje veliki izbor aktivacijskih funkcija no u praksi se koriste samo neke koje su se pokazale korisnima. Spomenuti ćemo četiri različite aktivacijske funkcije (slika 2.3) te njihove karakteristike.

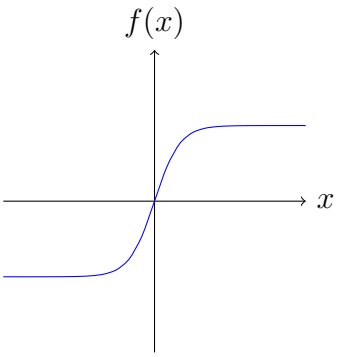
Najobičnija aktivacijska funkcija je linearna aktivacijska funkcija koja je preslikava svoj ulaz pomnožen sa nekom konstantom na izlaz. Ovakav tip aktivacijske funkcije ne koristimo u dubokim neuronskim mrežama zato što onemogućava učenje mreže.

Step funkcije u neuronima funkcioniraju kao prekidači. Izlaz funkcije može poprimiti samo dvije različite vrijednosti ovisno o tome da li je ulaz manji ili veći od nekog praga. Primjer jedne ovakve funkcije je:

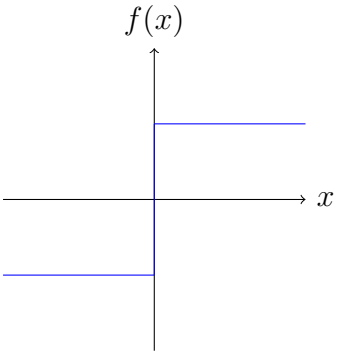
$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \tag{2.3}$$



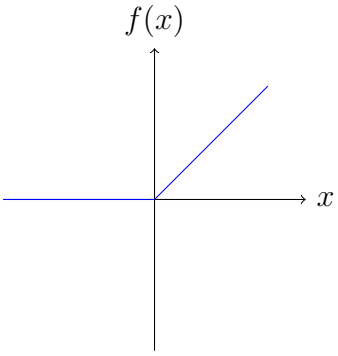
(a) Linearna funkcija



(b) Sigmoidalna funkcija



(c) Funkcija skoka



(d) Linearna rektifikacijska funkcija

Slika 2.3: Različite aktivacijske funkcije

Ovakva funkcija je korisna za binarne klasifikatore ali se ne koristi u dubokim neuronskim mrežama. Jedan od razloga je to što je za algoritam unazadne propagacije (kasnije ojašnjen) potrebna derivabilna ili po dijelovima derivabilna funkcija. Također zbog same definicije funkcije mala promjena ulaza može dovesti do potpuno suprotne aktivacije neurona čak iako su ulazi jako slični što je nepoželjno svojstvo za našu primjenu.

Sigmoidalne aktivacijske funkcije se najčešće koriste u praksi kod dubokih neuronskih mreža. Ovakve funkcije su derivabilne na cijeloj domeni i ograničene su što su dobra svojstva za algoritam unazadne propagacije i učenje mreže. Dvije najčešće korištene sigmoidalne funkcije su logistička funkcija i funkcija hiperbolnog tangensa. Primjer logističke funkcije dan je u izrazu 2.4.

$$f(x) = \frac{1}{1 - e^{-kx}} \tag{2.4}$$

U ovom radu nećemo koristiti logističku funkcije već funkciju hiperbolnog tangensa koja se pokazala boljom u praksi [**?**]. Po uzoru na [**?**] koristiti ćemo skaliranu funkciju hiperbolnog tangensa prema izrazu 2.5 čija je derivacija dana sa 2.6

$$f(x) = 1.7159 \tanh\left(\frac{2}{3}x\right) \tag{2.5}$$

$$\frac{f(x)}{dx} = 1.444\left(1 - \tanh^2\left(\frac{2}{3}x\right)\right) \tag{2.6}$$

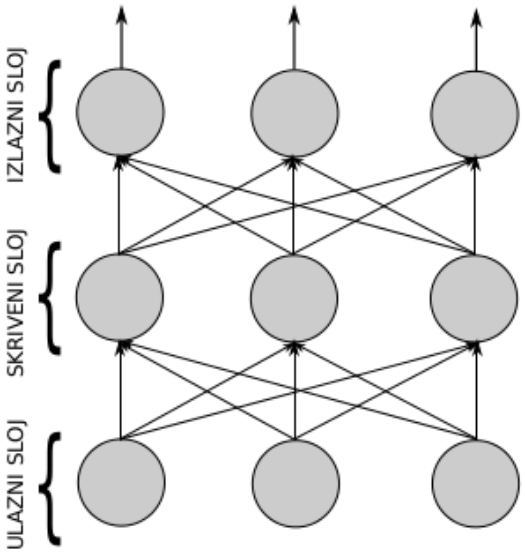
Još jedna aktivacijska funkcija koja je u zadnje vrijeme davala jako dobre rezultate je linearna rektifikacijska funkcija definirana kao:

$$f(x) = max(0, x) \tag{2.7}$$

Prema [**?**] korištenjem ove aktivacijske funkcije je postignuta čak 6 puta brža konvergencija mreže. Funkcija je po dijelovima derivabilna i nije linearna te je također vrlo jeftina za izračunat (dovoljan je jedan if uvjet u kodu).

2.2. Arhitektura neuronske mreže

Povezivanjem velikog broja neurona nastaju neuronske mreže. Neuronske mreže su modelirane kao kolekcije neurona koje su povezane acikličkim grafom. Neuroni u neuronskim mrežama su najčešće organizirani po slojevima (slika 2.4). Razlikujemo ulazni, izlazni i skriveni sloj. U ulaznom sloju na ulaze neurona dovodimo podatke koje je potrebno klasificirati. Na primjer, za rukom pisane znamenke bi pojedini ulaz



Slika 2.4: Potpuno povezana neuronska mreža sa jednim skrivenim slojem

bio pisana znamenka. Izlazi neurona ulaznog sloja su spojeni sa ulazima neurona skrivenog sloja. Skrivenih slojeva može biti više pa su zato izlazi neurona skrivenih slojeva povezani sa ulazima neurona idućih skrivenih slojeva ili sa ulazima neurona izlaznog sloja. Izlaz iz neurona izlaznog sloja se interpretira kao klasa koju je mreža klasificirala. Na primjer, ako klasificiramo rukom pisane znamenke onda postoji 10 različitih klasa i deset neurona u izlaznom sloju. Na temelju tog izlaza (najčešće u obliku brojeva od 0 do 1) inerpretiramo rezultat klasifikacije neuronske mreže. Slojevi su najčešće potpuno povezani poput primjera na slici 2.4. To znači da su svi neuroni trenutnog sloja povezani sa svim neuronima sljedećeg sloja.

Dubokim neuronskim mrežama nazivamo mreže koje imaju dva ili više skrivenih slojeva. Ispostavilo se da su duboke neuronske mreže pogodnije za kompleksnije probleme klasifikacije i da ostvaruju dobre rezultate. Možemo reći da svaki sloj mreže obrađuje podatke na drugoj razini apstrakcije i na temelju tih podataka donosi neku odluku, odnosno daje neki izlaz. Kretanjem od ulaznog sloja prema izlaznom razina apstrakcije se povećava te se grade kompleksniji i apstraktiniji koncepti odlučivanja. Ulazni sloj obrađuje podatke na razinama piksela dok izlazni sloj radi na najapstraktnijoj razini i daje nam rezultat klasifikacije. Intuitivno bismo mogli reći da sa većim brojem slojeva možemo preciznije dekompozirati apstraktni problem klasifikacije na niz jednostavnih odluka koje se mogu donesti na razinama piksela. Sa većim brojem slojeva je ta dekompozicija finija i preciznija.

Ono što u stvarnosti duboke neuronske mreže rade je simulirajnje nelinearne funkcije sa velikim brojem parametara. Kada je mreža "naučena", funkcija koju ona simu-

lira je točno ta funkcija koja nam za dane ulaze daje takve izlaze koje interpretiramo kao točne rezultate klasifikacije. Intitivno je jasno da za probleme klasifikacije često trebamo složene funkcije koje nisu jednostavne. Veći broj slojeva duboke neuronske mreže povećava tu složenost i omogućuje pronalazak takvih funkcija.

2.3. Učenje neuronskih mreža

Pošto neuronske mreže imaju milijune parametara (težina) koje je potrebno odrediti kako bi mreža radila dobru klasifikaciju trebamo znati kako odrediti te parametre. Dva algoritma su ključna za rad neuronske mreže i za njezino učenje a to su: algoritam *feedforward* i algoritam sa širenjem pogreške unatrag(engl. *backpropagation*).

2.3.1. Algoritam feedforward

Algoritam feedforward omogućava rad neuronske mreže. Algoritam je vrlo jednostavan. Za svaki sloj računamo njegov izlaz krećući od ulaznog. Ulaz ulaznog sloja su podatci za klasifikaciju dok je njegov izlaz ulaz sljedećeg sloja. Jedino na što treba paziti je povezanost slojeva koja za svaki neuron određuje koji dio ulaza utječe na njegov izlaz. Algoritam je opisan sa pseudokodom 1.

Pseudokod 1 Feedforward

Ulaz: x
za svaki sloj od ulaznog do izlaznog **radi**
  Izračunaj izlaz sloja za ulaz x
  $x \leftarrow$ izlaz trenutnog sloja
kraj za

2.3.2. Algoritam backpropagation

Neuronsku mrežu možemo shvatiti kao funkciju više varijabli. Varijable su težine na prijelazima neurona a izlaz iz funkcije je pogreška mreže. U našem promatranju smatramo da je ulaz konstantan i da nije varijabla. Naš cilj je minimizirati pogrešku što se svodi na pretraživanje n-dimenzionalnog prostora gdje je n ukupan broj težina u mreži. Pogreška u takvom prostoru se može vizualizirati kao hiper-površina sa više lokalnih minimuma.

Ideja algoritma backpropagation je određivanje greške i gradijenata u svakom sloju te ažuriranje težina na temelju gradijenata tako smanjujući grešku neuronske mreže

(gradijentni spust). Prvo se pomoću algoritma feedforward dobije odziv mreže za neki ulaz. Zatim se izračunaju greške izlaznog sloja (greške se računaju na svakom neuronu). Zatim se za prethodni sloj određuje utjecaj neurona na greške u idućem sloju te se izračuna greška prethodnog sloja. Zatim se izračuna gradijent greške po težinama koje povezuju te slojeve te se težine ažuriraju. Ovaj postupak se ponavlja za svaki ulaz i određen broj puta.

U svim oznakama koje slijede vrijedi konvencija označavanja trenutnog sloja sa *j* te prethodnog sloja sa *i*, izlaza neurona sa *y* te ukupan ulaz neurona sa z. Stoga y_i označava izlaz *i*-tog neurona prethodnog sloja a y_j izlaz *j*-tog neurona trenutnog sloja, z_j ulaz *j*-tog neurona trenutnog sloja, b_j prag j-tog neurona trenutnog sloja te w_{ij} težinu koja spaja i-ti neuron prethodnog sloja sa j-tim neuronom trenutnog sloja.

Da bismo odredili grešku izlaznog sloja moramo prvo odrediti funkciju pogreške. Najčešće se koristi srednja kvadratna pogreška:

$$E = \frac{1}{2}\sum_j (t_j - y_j)^2 \tag{2.8}$$

Parametar t_j predstavlja očekivani izlaz j-tog neurona. Grešku trenutnog sloja definiramo kao:

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j} \tag{2.9}$$

Parcijalnu derivaciju pogreške po izlazu neurona y_j za srednju kvadratnu pogrešku možemo raspisati kao:

$$\frac{\partial E}{\partial y_j} = \frac{1}{2}\frac{\partial}{\partial y_j}(t_j - y_j)^2 = -(t_j - y_j) \tag{2.10}$$

Druga parcijalna derivacija u izrazu 2.9 je jednaka derivaciji aktivacijske funkcije. Derivacija aktivacijske funkcije skaliranog hiperbolnog tangensa je već dana u izrazu 2.6.

Nakon računanja greške trenutnog sloja računa se greška prethodnog sloja koja je dana sa izrazom:

$$\frac{\partial E}{\partial z_i} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial z_i} \tag{2.11}$$

Druga parcijalna derivacije je ponovno jednaka derivaciji aktivacijske funkcije a parcijalna derivaciju pogreške po izlazu neurona prethodnog sloja dobijemo sumiranjem utjecaja neurona na sve neurone trenutnog sloja:

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial y_i} \tag{2.12}$$

Raspišimo z_j kao:

$$z_j = \sum_i w_{ij} y_i + b_j \tag{2.13}$$

Uvrštavanjem 2.13 u 2.12 dobivamo:

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial z_j} \frac{\partial z_j}{\partial y_i} = \sum_j w_{ij} \frac{\partial E}{\partial z_j} \tag{2.14}$$

Na kraju se određuju parcijalne derivacije po težinama i pragovima:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}} = \frac{\partial E}{\partial z_j} y_i \tag{2.15}$$

$$\frac{\partial E}{\partial b_j} = \frac{\partial E}{\partial z_j} \frac{\partial z_j}{\partial b_j} = \frac{\partial E}{\partial z_j} * 1 \tag{2.16}$$

Nakon čega se težine i pragovi ažuriraju u ovisnosti o stopi učenja η:

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}} = w_{ij} - \eta * y_i \frac{\partial E}{\partial z_j} \tag{2.17}$$

$$b_j \leftarrow b_j - \eta \frac{\partial E}{\partial b_j} = b_j - \eta \frac{\partial E}{\partial z_j} \tag{2.18}$$

Stopa učenja η je mali pozitivni broj koji nam govori koliko brzo ćemo se kretati u smijeru negativnog gradijenta. Gradijent pokazuje u smijeru rasta funkcije pa je zato kod ažuriranja težina i pragova potrebno dodati negativan predznak jer pokušavamo minimizirati funkciju.

Algoritam backpropagation je opisan sa pseudokodom 3. Uvjet zaustavljanja algoritma je najčešće unaprijed zadan broj iteracija. Svaku iteraciju algoritma nazivamo epohom. Uvjet nemora nužno biti zadan brojem epoha, također je moguće da se kao uvjet postavi minimalna pogreška izlaza tj, da algoritam staje kad je pogreška dovoljno mala.

Prethodno opisani algoritam koristi stohastički gradijentni spust što znači da se težine ažuriraju nakon svakog ulaza. To znači da nije nužno da se uvijek krećemo u smijeru negativnog gradijenta na razini cijelog skupa za učenje. Ovakva varijanta gradijentnog spusta više oscilira te je upravo zbog tog svojstva otpornija na zapinjanje u lokalnim minimumima. Standardna varijanta gradijentnog spusta ažurira težine ili nakon nekog određenog broja ulaza (engl. batch) ili nakon svake epohe. U obzir se uzima prosjek gradijenata na svim obrađenim ulazima te je zato ova varijanta stabilnija i ima manje oscilacije ali zato ima veće šanse zapinjanja u lokalnim minimumima te je puno sporija. Mi ćemo koristiti navedeni stohastički gradijentni spust pošto se u praksi pokazao veoma efikasnim a ujedno je računski puno manje zahtjevan od standardnog.

Pseudokod 2 Backpropagation

Ulaz: D (skup za učenje), η (stopa učenja)
Inicijaliziraj težine na male slučajno generirane vrijednosti
dok nije ispunjen uvjet zaustavljanja **radi**
  za svaki (x, t) iz D **radi**
    Izračunaj izlaz svakog sloja mreže za ulaz x
    Izračunaj pogrešku izlaznog sloja prema formulama 2.9 i 2.10
    za svaki sloj od izlaznog do ulaznog **radi**
      Izračunaj pogrešku prethodnog sloja prema formulama 2.11 i 2.14
      Izračunaj parcijalne derivacije pogreške po težinama i pragovima prema formulama 2.15 i 2.16
      Ažuriraj težine i pragove prema formulama 2.17 i 2.18
    kraj za
  kraj za
kraj dok

3. Konvolucijske neuronske mreže

Konvolucijske neuronske mreže možemo smatrati proširenjima standardnih neuronskih mreža koje su se pokazale učinkovitijima prilikom klasifikacija slika. Neuroni u konvolucijskim neuronskim mrežama su dvodimenzionalni i nazivamo ih mapama značajki (engl. *feature maps*). Ulaz je također dvodimenzionalan a umjesto težina se koriste jezgre (engl. *kernels*).

3.1. Struktura mreže

Konvolucijske neuronske mreže su građene od tri različite vrste slojeva: konvolucijski slojevi, slojevi sažimanja i potpuno povezani slojevi. Na ulazu mreže se nalazi jedna monokromatska ili višekanalna slika u boji. Zatim slijede naizmjence konvolucijski slojevi i slojevi sažimanja. Mape značajki u tim slojevima u svakom sloju postaju sve manjih dimenzija krećući se od ulaznog sloja. Zadnji takav sloj je dimenzija 1 $\times$ 1. Na takav sloj se vežu potpuno povezani slojevi koji su jednodiemnzionalni te se ponašaju kao obične neuronske mreže opisane u prethodnom poglavlju. Primjer ovakve strukture vidimo na slici **??**.

3.1.1. Konvolucijski slojevi

Konvolucijski slojevi uzimaju mape na ulazu sloja te rade 2D kovoluciju sa jezgrama. Označimo sa M^j mape j-tog sloja, sa M dimenzije tih mapa te sa K^j jezgre koje povezuju mape prethnodnog sloja sa mapama trenutnog sloja i sa K dimenzije tih jezgri. Radi jednostavnosti ćemo koristiti kvadratne mape značajki i kvadratne jezgre te kad govorimo o dimenziji M misli se na $M \times M$ (ekvivalentno i sa dimenzijama jezgri). Također označimo sa S korak pomaka jezgre po širini i visini prilikom konvolucije. Veličine mapi značajki u nekom sloju dana je sa izrazom:

$$M = \frac{M - K}{S} + 1 \tag{3.1}$$

Konvolucija se tvori prolazom kroz ulaznu mapu sa prozorom jednake veličine kao i jezgra te se množe vrijednosti ulazne mape unutar prozora sa korespodentnim vrijednostima jezgre(možemo zamisliti koda preklopimo jezgru preko dijela ulazne mape i množimo vrijednosti koje su jedna na drugoj). Sumiramo te umnožke za sve ulazne mape značajki i dodamo prag te izračunamo izlaz aktivacijske funkcije koji zapisujemo u odgovarajući neuron izlazne mape značajki. Pod pojmom neuron u ovom kontekstu se misli na jednu jedinicu mape značajki. Dakle jedna mapa značajki dimenzije M ima $M \times M$ neurona. Nakon toga pomičemo okvir za S vodoravno, ili okomito ako smo došli do kraja reda te radimo proces isponova za idući neuron.

Vidimo da na jedan neuron izlazne mape značajki utječu samo dijelovi ulaznih mapi značajki koji su unutar okvira koji je potreban za taj neuron. To područje ulazne mape značajki "vidljivo" neuronu nazivamo vizualnim ili receptivnim poljem neurona. Ako se neuron u izlaznoj mapi nalazi na kordinatama (x, y) onda je njegovo vizualno polje definirano sa kvadratom dimenzija jednakih dimenzijama jezgre K, a kordinate gornjeg lijevog kuta vizualnog polja (x', y') u kordinatnom sustavu ulaznih mapi značajki su definirane kao:

$$x' = x * S \tag{3.2}$$

$$y' = y * S \tag{3.3}$$

Označimo sa M_k^j k-tu mapu j-tog sloja te sa w_{ik}^j jezgru koja povezuje k-tu mapu j-tog sloja sa i-tom mapom prethodnog sloja. Svaka mapa značajki ima po jedan prag b_k^j. Pošto su mape značajki i njihove jezgre dvodimenzionalne njihove elemente indeksiramo sa zagradama. Tako će vrijednost mape značajki na lokaciji (x, y) biti jednaka $M_k^j(x, y)$ a jezgre $w_{ik}^j(x, y)$. Uz ovaj sustav oznaka vrijednost mape k u sloju j na lokaciji (x, y) možemo prikazati sa sljedećim izrazom:

$$M_k^j(x, y) = f(\sum_i \sum_{x'=0}^{K-1} \sum_{y'=0}^{K-1} M_i^{j-1}(x' + x, y' + y) w_{ik}^j(x', y') + b_k^j) \tag{3.4}$$

Funkcija u jednadžbi je aktivacijska funkcija te je podrazumijevani pomak okvira S jednak 1. Naravno pričati ćemo samo o potpuno povezanim mrežama gdje je svaka mapa značajki trenutnog sloja povezana sa svim mapama značajki prethodnog sloja. Vrijednosti mapa značajki prilikom unaprijedne propagacije se računaju prema formuli 3.4.

3.1.2. Slojevi sažimanja

Slojevi sažimanja (engl. *pooling*) nemaju parametre koji se mogu učiti i služe za smanjenje dimenzija mapi značajki i uklanjanje varijance što znači da će se slični izlazi dobiti za male translacije ulaza. U ovim slojevima također imamo okvire sa kojima prolazimo po ulaznoj mapi značajki. Mapa se sažima na taj način da okvir predstavimo sa jednom vrijednošću. Na primjer okvir veličine 2×2 (najčešća veličina okvira koju ćemo i mi koristiti) se reprezentira sa jednom vrijednoću dobivenom iz 4 vrijednosti unutar okvira čime smanjujemo mapu 4 puta. Okvir se najčešće pomiče na način da se svaka vrijednost iz mape značajki koristi u samo jednom sažimanju. Pomak okvira bi za navedeni primjer bio jednak 2 u horizontalnom i vertikalnom smijeru.

Sažimanje usrednjavanjem

Sažimanje usrednjavanjem (engl. *mean pooling*) se ostvarije uzimanjem aritmetičke sredine vrijednosti unutar okvira sažimanja. Npr ako imamo mapu $M = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$ sažimanjem usrednjavanjem sa okvirom veličine 2×2 i pomakom 2 po horizontali i vertikali ćemo dobiti 4 puta manju mapu značajki $M' = \begin{bmatrix} 3.5 & 5.5 \\ 11.5 & 13.5 \end{bmatrix}$

Sažimanje maksimalnom vrijednošću

Sažimanje maksimalnom vrijednošću (engl. *max pooling*) se ostvaruje uzimanjem maksimalne vrijednosti unutar okvira sažimanja. Za istu mapu značajki M iz prethodnog primjera i za iste dimenzije sažimanja bismo dobili mapu značajki $M' = \begin{bmatrix} 6 & 8 \\ 14 & 16 \end{bmatrix}$

3.1.3. Backpropagation u konvolucijskim mrežama

Potrebno je definirati izmijenjeni algoritam backpropagation za primjenu u konvolucijskim slojevima i slojevima sažimanja. Pošto su zadnji slojevi mreže potpuno povezani slojevi kao u običnim dubokim neuronskim mrežama unazadna propagacija pogreške i ažuriranje težina se u tim slojevima obavlja na već opisani način iz prethodnog poglavlja. Također ćemo radi lakšeg razumijevanja i unazadne propagacije iz konvolucijskog sloja izdvojiti primjenu aktivacijskih funkcija u posebni sloj. Nazovimo ga aktivacijski sloj. To znači da se u konvolucijskom sloju rade samo sva potrebna sumiranja a u

aktivacijskom sloju se na svaku vrijednost prethodne mape značajki (konvolucijskog sloja) primjenjuje aktivacijska funkcija. Također držati ćemo se sljedeće konvencije imenovanja:

- $M_k^j(x, y)$ - izlaz neurona k-te mape j-tog sloja koji se nalazi na lokaciji (x, y)
- $w_{ik}^j(x, y)$ - vrijednost jezgre j-tog sloja između k-te mape značajki j-tog sloja i i-te mape značajki prethodnog sloja na lokaciji (x, y)
- b_k^j - prag k-te mape značajki j-tog sloja
- K^j - veličina jezgri između j-tog i prethodnog sloja
- M^j - veličina mape značajki trenutnog sloja
- $z_k^j(x, y)$ - ulaz neurona k-te mape značajki u sloju j koji se nalazi na lokaciji (x, y)
- $\dfrac{\partial E}{\partial M_k^j(x, y)}$ - pogreška izlaza k-te mape značajki u sloju j na lokaciji (x, y)
- $\dfrac{\partial E}{z_k^j(x, y)}$ - pogreška k-te mape značajki u sloju j na lokaciji (x, y)

Radi kompatibilnosti sa programskom implementacijom indeksiranje lokacija (x, y) započinje sa 0 a ne sa 1 što znači da bismo gornji lijevi kut indeksirali sa $(0, 0)$. Za svaki sloj posebno ćemo objasniti računanje greške izlaza prethodnog sloja pod uvjetom da imamo izračunatu grešku izlaza trenutnog sloja (primjetimo razliku u definiciji greške izlaza sloja i greške sloja).

Konvolucijski slojevi

U konvolucijskim slojevima z_k^j možemo napisati kao:

$$z_k^j(x, y) = \sum_i \sum_{x'=0}^{K-1} \sum_{y'=0}^{K-1} M_i^{j-1}(x' + x, y' + y) w_{ik}^j(x', y') + b_k^j \tag{3.5}$$

Pošto smo aktivacijsku funkciju izdvojili u poseban sloj onda vrijedi sljedeći izraz:

$$M_k^j(x, y) = z_k^j(x, y) \tag{3.6}$$

Poznata nam je greška izlaza za svaki neuron mape značajki trenutnog sloja (tu informaciju smo dobili od idućeg sloja). Prvo moramo dobiti grešku trenutnog sloja. Greška pojedinog neurona trenutnog sloja je jednaka:

$$\frac{\partial E}{z_k^j(x, y)} = \frac{\partial E}{\partial M_k^j(x, y)} \frac{\partial M_k^j(x, y)}{\partial z_k^j(x, y)} \tag{3.7}$$

Uvrštavanjem 3.6 dobivamo:

$$\frac{\partial E}{z_k^j(x,y)} = \frac{\partial E}{\partial M_k^j(x,y)} \frac{\partial z_k^j(x,y)}{\partial z_k^j(x,y)} = \frac{\partial E}{\partial M_k^j(x,y)} \tag{3.8}$$

Potrebno je izračunati grešku izlaza prethondog sloja tako da sumiramo utjecaj neurona prethodnog sloja na sve neurone trenutnog sloja. Sumiranje obavljamo po mapama značajki trenutnog sloja i po lokacijama u tim mapama na koje utječe izlaz neurona $M_k^{j-1}(x,y)$:

$$\begin{aligned} \frac{\partial E}{\partial M_k^{j-1}(x,y)} &= \sum_i \sum_{x'=0}^{K^j-1} \sum_{y'=0}^{K^j-1} \frac{\partial E}{\partial z_i^j(x-x', y-y')} \frac{\partial z_i^j(x-x', y-y')}{\partial M_k^{j-1}(x,y)} \\ &= \sum_i \sum_{x'=0}^{K^j-1} \sum_{y'=0}^{K^j-1} \frac{\partial E}{\partial z_i^j(x-x', y-y')} w_{ki}^j(x', y') \end{aligned} \tag{3.9}$$

Nakon računanja greške prethodnog sloja potrebno je izračunati parcijalne derivacije greške po težinama i pragovima:

$$\begin{aligned} \frac{\partial E}{\partial w_{ik}^j(x,y)} &= \sum_{x'=0}^{M^j-1} \sum_{y'=0}^{M^j-1} \frac{\partial E}{\partial z_k^j(x',y')} \frac{\partial z_k^j(x',y')}{\partial w_{ik}^j(x,y)} \\ &= \sum_{x'=0}^{M^j-1} \sum_{y'=0}^{M^j-1} \frac{\partial E}{\partial z_k^j(x',y')} M_i^{j-1}(x+x', y+y') \end{aligned} \tag{3.10}$$

$$\begin{aligned} \frac{\partial E}{\partial b_k^j} &= \sum_{x'=0}^{M^j-1} \sum_{y'=0}^{M^j-1} \frac{\partial E}{\partial z_k^j(x',y')} \frac{\partial z_k^j(x',y')}{\partial b_k^j} \\ &= \sum_{x'=0}^{M^j-1} \sum_{y'=0}^{M^j-1} \frac{\partial E}{\partial z_k^j(x',y')} \end{aligned} \tag{3.11}$$

Na kraju se ažuriraju težine i pragovi prema sljedećim izrazima:

$$w_{ik}^j(x,y) \leftarrow w_{ik}^j(x,y) - \eta \frac{\partial E}{\partial w_{ik}^j(x,y)} \tag{3.12}$$

$$b_k^j \leftarrow b_k^j - \eta \frac{\partial E}{\partial b_k^j} \tag{3.13}$$

Slojevi sažimanja

Pošto u slojevima sažimanja nemamo težine ni pragove koje treba ažurirati potrebno je samo odrediti pogrešku izlaza prethodnog sloja. Promatrajmo mapu veličine 2×2 za koju nam je poznata pogreška izlaza. Označimo mapu pogreške izlaza sloja j sa δ^j:

$$\delta^j = \begin{bmatrix} \delta_{00} & \delta_{01} \\ \delta_{10} & \delta_{11} \end{bmatrix} \tag{3.14}$$

Za promatrani primjer se podrazumijeva da je okvir sažimanja veličine 2×2 što znači da je mapa značajki prethodnog sloja dimenzija 4×4.

Za slojeve sažimanja možemo definirati funkcije koje za dani broj elemenata unutar okvira daju neki iznos. Za sažimanje maksimumom i sažimanje srednjom vrijednošću možemo definirati te funkcije kao:

$$f_{max}(x_{00}, x_{01}, x_{10}, x_{11}) = max(x_{00}, x_{01}, x_{10}, x_{11}) \tag{3.15}$$

$$f_{med}(x_{00}, x_{01}, x_{10}, x_{11}) = \frac{x_{00} + x_{01} + x_{10} + x_{11})}{4} \tag{3.16}$$

Definirajmo parcijalne derivacije funkcija sažimanja:

$$\frac{\partial f_{max}}{\partial x_{ij}} = \begin{cases} 0, & \text{ako } x_{ij} \text{ nije maksimalna vrijednost} \\ 1, & \text{ako } x_{ij} \text{ je maksimalna vrijednost} \end{cases} \tag{3.17}$$

$$\frac{\partial f_{med}}{\partial x_{ij}} = \frac{1}{4} \tag{3.18}$$

Sada pogrešku izlaza prethodnog sloja možemo dobiti sa umnoškom parcijalne derivacije funkcije sažimanja sa odgovarajućom pogreškom izlaza trenutnog sloja. Rezultati tih operacija za sloj sažimanja maksimumom i sloj sažimanja aritmetičkom sredinom su:

$$\delta_{max}^{j-1} = \begin{bmatrix} \delta_{00} & 0 & 0 & 0 \\ 0 & 0 & 0 & \delta_{01} \\ 0 & 0 & \delta_{11} & 0 \\ 0 & \delta_{10} & 0 & 0 \end{bmatrix} \tag{3.19}$$

$$\delta_{med}^{j-1} = \begin{bmatrix} \frac{\delta_{00}}{4} & \frac{\delta_{00}}{4} & \frac{\delta_{01}}{4} & \frac{\delta_{01}}{4} \\ \frac{\delta_{00}}{4} & \frac{\delta_{00}}{4} & \frac{\delta_{01}}{4} & \frac{\delta_{01}}{4} \\ \frac{\delta_{10}}{4} & \frac{\delta_{10}}{4} & \frac{\delta_{11}}{4} & \frac{\delta_{11}}{4} \\ \frac{\delta_{10}}{4} & \frac{\delta_{10}}{4} & \frac{\delta_{11}}{4} & \frac{\delta_{11}}{4} \end{bmatrix} \tag{3.20}$$

Za sloj sažimanja maksimumom se pretpostavlja da se na mjestima različitima od 0 nalaze maksimalni izlazi unutar okvira. Za drugačije dimenzija okvira sažimanja primjenjuje se ista logika. Napišimo općenitu formulu za pogrešku neurona prethodnog sloja uz funkciju sažimanja f čije su varijable $x_{ij}, i, j \in (0..K-1)$ uz veličinu okvira sažimanja K.

$$\frac{\partial E}{M_k^{j-1}(x,y)} = \frac{\partial E}{\partial M_k^j(\lfloor \frac{x}{K} \rfloor, \lfloor \frac{y}{K} \rfloor)} \frac{\partial f}{\partial x_{ij}}, \qquad i = x \ mod \ K, \ j = y \ mod \ K \tag{3.21}$$

Aktivacijski slojevi

Kao i za slojeve sažimanja u aktivacijskim slojevima je isto samo potrebno odrediti pogrešku izlaza prethednog sloja. Označimo aktivacijsku funkciju sa $f(x)$. Sada pogrešku izlaza prethodnog sloja možemo pisati kao umnožak greške izlaza trenutnog sloja i derivacije aktivacijske funkcije:

$$\frac{\partial E}{M_k^{j-1}(x,y)} = \frac{\partial E}{\partial M_k^j(x,y)} \frac{\partial M_k^j(x,y)}{\partial M_k^{j-1}(x,y)} = \frac{\partial E}{\partial M_k^j(x,y)} f'(M_k^{j-1}(x,y)) \tag{3.22}$$

Pseudokod algoritma backpropagation

Sada možemo napisati pseudokod izmijenjenog algoritma backpropagation za konvolucijske neuronske mreže:

Pseudokod 3 Backpropagation

Ulaz: D(skup za učenje), η stopa učenja
Inicijaliziraj težine u konvolucijskim i potpuno povezanim slojevima na male slučajno generirane vrijednosti
Dok nije ispunjen uvjet zaustavljanja:
Za svaki (x, t) iz D:
Izračunaj izlaz svakog sloja mreže za ulaz x
Izračunaj pogrešku izlaznog sloja prema formulama 2.9 i 2.10
Za svaki sloj od izlaznog do ulaznog:
Ako je sloj potpuno povezani:
Izračunaj pogrešku izlaza prethodnog sloja prema formulama 2.11 i 2.14
Inače ako je sloj aktivacijski:
Izračunaj pogrešku izlaza prethodnog sloja prema formuli 3.22
Inače ako je sloj sloj sažimanja:
Izračunaj pogrešku izlaza prethodnog sloja prema formuli 3.21
Inače ako je sloj konvolucijski:
Izračunaj pogrešku izlaza prethodnog sloja prema formuli 3.9
Izračunaj parcijalne derivacije pogreške po težinama i pragovima prema formulama 3.10 i 3.11
Ažuriraj težine i pragove prema formulama 3.12 i 3.13

3.1.4. Hiperparametri mreže

Definiramo hiperparametar algoritma učenja kao varijablu koju je potrebno postaviti prije aplikacije algoritma na skupu podataka.[REFERENCA] Dakle to su svi parametri koje trebamo odabrati i odrediti prije nego što mreži damo podatke za učenje. Optimiziranje hiperparametara je postupak pronalaženja optimalnih (ili dovoljno dobrih) hiperparametara mreže. Hiperparametri u konvolucijskim neuronskim mrežama su:

- **Stopa učenja** η je jedan od najvažnijih parametara neuronske mreže. Konvergencija mreže ovisi o pronalasku dobre stope učenja. Obično je vrijednost ovog parametra između 10^{-6} i 1. Stopa učenja se određuje isprobava različite vrijednosti i prateći konvergenciju mreže. Ako nemamo vremena optimizirati više hiperparametara i ako se koristi stohastički gradijentni spust onda je definitivno najbolje optimizirati ovaj hiperparametar.
- **Broj epoha** je hiperparametar kojeg je lako optimizirati tehnikom ranog stajanja(engl. *early stop*). Prateći prosječnu grešku na validacijskom skupu nakon svake epohe može se odlučiti koliko dugo trenirati mrežu za proizvoljnu konfiguraciju ostalih hiperparametara.
- **Arhitektura mreže** kao hiperparametar je skup odluka na koji način oblikovati mrežu: broj slojeva u mreži i redoslijed tih slojeva, broj mapi značajki u svakom sloju, dimenzije mapi značajki, dimenzije jezgri u konvolucijskim slojevima i dimenzije okvira sažimanja u slojevima sažimanja. Kod izgradnje arhitekture mreže važno je odabrati mrežu koja je dovoljno velika. Obično veličine veće od optimalne generalno ne štete performansama mreže osim što ju usporavaju. Potrebno je modelirati mrežu na taj način da je dovoljno velika za klasifikacijski problem a da nije toliko velika da bude jako spora.
- **Aktivacijska funkcija** je najčešće ista za cijelu neuronsku mrežu. Aktivacijske funkcije smo već obradili u poglavlju 2.1.1.
- **Inicijalizacija težina**. Pragovi se mogu inicijalizirati na 0 ali se težine moraju inicijalizirati oprezno kako nebismo usporili konvergenciju mreže odmah na početku. Preporučena incijalizacija u [REFERENCA] je da se uzimaju vrijednosti iz uniformne distribucije u rasponu $(-r, r)$ gdje je $r = 4\sqrt{6/(n_{in} + n_{out})}$ za aktivacijsku funkciju hiperbolnog tangensa odnosno $r = \sqrt{6/(n_{in} + n_{out})}$ za logističku funkciju gdje n_{in} i n_{out} predstavlja broj ulaznih i izlaznih neurona za određeni sloj.
- **Predobrada ulaza** se koristi kako bi se obradili ulazi prije nego što se daju

kao ulaz neuronskoj mreži. Najčešći oblici predobrade ulaza su standardizacija i dodavanje okvira za konvolucijske neuronske mreže. Standardizacija se radi tako da od svakog piksela oduzmemo srednju vrijednost cijele slike te dobivenu vrijednost podijelimo sa standardnom devijacijom svih piksela u slici. Dodavanjem okvira slici se slika proširi za nekoliko piksela sa svake strane a ova predobrada se koristi kako konvolucijska mreža nebi zanemarila i rubne podatke slike (tj. da ih uzima sa jednakim značajem).

- **Slojevi sažimanja** su obrađeni u poglavlju 3.1.2.
- **Funkcija pogreške** je također bitan hiperparametar o kojem može ovisiti mogućnost i brzina konvergencije mreže. Spomenuli smo funkciju srednje kvadratne pogreške (formula 2.8). Još jedna funkcija pogreške koja se često koristi je *cross-entropy loss*.

4. Programska izvedba

Za programsku izvedbu odabran je programski jezik C++. Izbor se temeljio na tome što je jezik pogodan za pisanje brzih programa (što nam je potrebno zbog dugotranog treniranja mreže), a također je pogodan za oblikovanje složenijih sustava i ovisnosti među elementima. Programski je mreža organizirana po slojevima kao što je navedeno u poglavlju o algoritmu backpropagation (aktivacijska funkcija je u posebnom sloju). Sljedeći programski kod pokazuje apstraktni razred Layer koji modelira jedan općeniti sloj konvolucijske neuronske mreže:

Programski kod 4.1: Razred Layer

```
1  typedef std::vector<float> vf;
2  typedef std::vector<vf> vvf;
3  typedef std::vector<vvf> vvvf;
4
5  class Layer
6  {
7  public:
8      Layer(int prevMapSize, int mapSize, int prevFM, int numFM);
9      virtual vvf& forwardPass(const vvf &input) = 0;
10     virtual vvf& backPropagate(const vvf &error) = 0;
11     vvf& getOutput() { return output; }
12     vvf& getPrevError() { return prevError; }
13     int getMapSize() { return mapSize; }
14
15 protected:
16     const int mapSize, prevMapSize;
17     // number of feature maps
18     const int prevFM, numFM;
19     vvf output, prevError;
20     const vvf *input;
21
22 };
```

Konstruktoru razreda Layer potrebno je specificirati sljedeće parametre:

- **prevMapSize** - veličina mapi značajki prethodnog sloja ili ulaza ako je ulazni sloj

- **mapSize** - veličina mapi značajki trenutnog sloja
- **prevFM** - broj mapi značajki prethodnog sloja
- **numFM** - broj mapi značajki trenutnog sloja

Također možemo vidjeti da je svaki sloj zadužen za stvaranje svog izlaza (**output**) kao i za stvaranje greške izlaza prethodnog sloja (**prevError**) dok za potrebe algoritma backpropagation čuva pokazivač na zadnji primljeni ulaz (**input**). Također je određeno da svaki sloj mora definirati dvije metode koje su različite za različite tipove slojeva a to su:

- **forwardPass** - metoda koja računa izlaz(output) sloja za zadani ulaz(input)
- **backPropagate** - metoda koja obavlja algortiam backpropagation na tom sloju(računa grešku prethodnog sloja i updatea težine ako one postoje)

4.0.5. Konvolucijski sloj

Razred ConvolutionLayer modelira konvolucijski sloj neuronske mreže te je prikazan sa sljedećim kodom:

Programski kod 4.2: Razred ConvolutionLayer

```
1  class ConvolutionLayer : public Layer
2  {
3  public:
4      ConvolutionLayer(int mapSize, int prevFM, int numFM, int kernelSize, Initializer &init, float learnin
5      virtual vvf& forwardPass(const vvf &input);
6      virtual vvf& backPropagate(const vvf &error);
7      vvvf& getKernel() { return kernelW; }
8      vf& getBias() { return bias; }
9      float getLearningRate() { return learningRate; }
10     void printKernel();
11     void writeKernel(std::string path);
12     void loadWeights(std::string file);
13
14 private:
15     const int kernelSize;
16     //kernelw[numFM][prevFM][i*kernelSize + j]
17     vvvf kernelW;
18     vf bias;
19     float learningRate;
20
21     void update(const vvf &error);
22     double convolve(int w, int h, const vvf &input, int numFM);
23 };
```

Konstruktoru predajemo sljedeće parametre:

- **mapSize** - veličina mapi značajki trunutnog sloja
- **prevFM** - broj mapi značajki prethodnog sloja
- **numFM** - broj mapi značajki trenutnog sloja
- **kernelSize** - veličina jezgri
- **init** - inicijalizator težina kojime specificiramo na koji način ćemo inicijalizirati težine prije učenja (pomoćna klasa Initializer)
- **learningRate** - stopa učenja η

Konvolucijski slojevi su dodatno zaduženi za stvaranje jezgri koje povezuju trenutni sloj sa prethodnim(**kernelW**) i pragova (**bias**). Dodatne metode koje je potrebno objasniti su:

- **update** - metoda koja ažurira težine i pragove, a poziva se iz metode backpropagate
- **convolve** - metoda koja obavlja konvolucija ulaza(input) i jezgri te sprema rezultat u mapu značajki numeriranu sa *numFM* na kordinatama (w, h), ova metoda se poziva iz metode forwardPass
- **printKernel** - ispisuje jezgre na standardni izlaz
- **writeKernel** - iscrtava jezgre u .jpg formatu u zadanu lokaciju(path)
- **loadWeights** - učitava težine i pragove iz datoteke (file), koristi se za učitavanje naučenih parametara

4.0.6. Potpuno povezani slojevi

Potpuno povezane slojeve možemo gledati kao specijalizacije konvolucijskih slojeva gdje su sve mape značajki i sve jezgre veličine 1×1. Sa obzirom na veličinu mreže mali postotak operacija se obavlja u potpuno povezanim slojevi (najveći postotak je u konvolucijskim slojevima) pa nije bilo potrebno posebno modelirati potpuno povezane slojeve već su oni napravljeni kao podtip konvolucijskih slojeva modelirani sa razredom FullyConnectedLayer:

Programski kod 4.3: Razred FullyConnectedLayer

```
1 class FullyConnectedLayer : public ConvolutionLayer
2 {
3 public:
4     FullyConnectedLayer (int prevFM, int numFM, Initializer &init, float learningRate) :
5                         ConvolutionLayer(1, prevFM, numFM, 1, init, learningRate) {}
6 };
```

Parametri u konstruktoru su istih naziva i značenja kao i kod konvolucijskih slojeva.

4.0.7. Aktivacijski slojevi

Aktivacijski slojevi su modelirani sa razredom ActivationLayer:

Programski kod 4.4: Razred ActivationLayer

```
1  class ActivationLayer : public Layer
2  {
3  public:
4      ActivationLayer(int numFM, int mapSize = 1);
5      virtual vvf& forwardPass(const vvf &input);
6      virtual vvf& backPropagate(const vvf &error);
7
8  protected:
9      virtual float activationFunction(float x) = 0;
10     virtual float activationFunctionDerivative(float x) = 0;
11 };
```

Konstruktor razreda prima dva parametra: broj mapi značajki (numFM) i veličinu mapi značajki(mapSize). Pošto su aktivacijske funkcije samo izdvojene u poseban sloj broj prethodnih mapi značajki i veličine prethodnih mapi značajki su jednake trenutnima. ActivationLayer je i dalje apstraktni razred jer ima dvije čiste virtualne funkcije koje zahtjevaju implementaciju od razreda koji ga nasljeđuju:

- **activationFunction** - aktivacijska funkcija u točki x
- **activationFunctionDerivative** - derivacija aktivacijske funkcije u točki x

Na ovaj način je lako dodavati različite aktivacijske funkcije. Potrebno je samo definirati ove dvije navedene metode.

4.0.8. Slojevi sažimanja

Slojevi sažimanja su modelirani sa razredom PoolLayer:

Programski kod 4.5: Razred PoolLayer

```
1  class PoolLayer : public Layer
2  {
3  public:
4      PoolLayer (int frameSize, int numFM, int prevMapSize);
5
6  protected:
7      const int frameSize;
8  };
```

Konstruktor razreda prima sljedeće parametre:

- **frameSize** - veličina okvira sažimanja
- **numFm** - broj mapi značajki
- **prevMapSize** - veličina mapi značakji prethodnog sloja (veličina mapi značajki trenutnog sloja se računa kao $prevMapSize/frameSize$

Razred je i dalje apstraktan a od razreda koji ga nasljeđuju se očekuje da implementiraju funkcije forwardPass i backPropagate.

4.0.9. Konvolucijska neuronska mreža

Cijela konvolucijska neuronska mreže enkapsulirana je sa razredom ConvolutionNeuralNetwork:

Programski kod 4.6: Razred ConvolutionNeuralNetwork

```
1  class ConvolutionNeuralNetwork
2  {
3  public:
4      ConvolutionNeuralNetwork (const std::vector<Layer*> &layers, CostFunction &costFunction, InputI
5      void feedForward(vvf &input);
6      void backPropagate(vvf &error);
7      void train(int numEpochs);
8      void registerSupervisor(TrainingSupervisor *s) { supervisers.push_back(s); }
9      void notifySupervisors(int epoch);
10     float getCost(vf &expectedOutput);
11     InputManager& getInputManager() { return inputManager; }
12
13 private:
14     std::vector<Layer*> layers;
15     CostFunction &costFunction;
16     InputManager &inputManager;
17     std::vector<TrainingSupervisor*> supervisers;
18 };
```

Konstruktor razreda prima sljedeće parametre:

- **layers** - vektor slojeva mreže, slojevi se kreiraju i organiziraju prije predavanja konstruktoru
- **costFunction** - funkcija pogreške (pomoćni razred CostFunction)
- **inputManager** - objekt zadužen za organiziranje i dohvaćanje ulaza mreže (pomoćni razred InputManager)

  Metode razreda su:

- **feedForward** - za određeni ulaz(input) računa izlaze svih slojeva
- **backPropagate** - propagira grešku unazad po svim slojevima
- **train** - trenira mrežu određen broj epoha(epoch), ulaze dohvaća pomoću inputManagera a pogrešku izlaza računa pomoću costFunctiona
- **registerSupervisor** - ova metoda omogućuje da se registriraju promatrači (izvedeni iz razreda TrainingSupervisor) prema oblikovnom obrascu promatrač koji nakon svake epohe obavljaju različite analize i na taj način omogućuju nadziranje procesa učenja mreže
- **notifySupervisors** - obaviještava sve promatrače da je kraj određene epohe (epoch)
- **getCost** - računa pogrešku za zadani očekivani izlaz, očekuje se da se prije poziva ove metode pozove metoda feedForward sa odgovarajućim ulazom

4.0.10. Pomoćni razredi

Imamo četiri osnovna pomoćna razreda iz kojih se izvode ostali. To su:

- **Initializer** određuje način na koji inicijalizira težine u konvolucijskim slojevima. Predaje se konstruktoru konvolucijskog sloja. Razredi koji nasljeđuju Initializer moraju implementirati jednu metodu init koja kao parametre prima težine(weights) te broj ulaznih(n_in) i broj izlaznih (n_out) neurona.
- **CostFunction** određuje funkciju pogreške izlaza neuronske mreže. Zadužena je za stvaranje i pohranjivanje greške izlaza neurona izlaznog sloja (prevError) kao i greške klasifikacije (error). Razredi koji nasljeđuju ovaj razred moraju implementirati metodu calculate koja kao parametre prima izlaz mreže (output) i očekivani izlaz (expectedOutput) te računa pogrešku izlaza izlaznog sloja i pogrešku klasifikacije.
- **InputManager** je zadužen za upravljanjem ulaznim podatcima za mrežu. Razredi koji ga nasljeđuju trebaju implementirati tri metode a to su getInput koja dohvaća ulaz na zadanom indeksu (i), getExpectedOutput koja dohvaća očekivani izlaz za ulaz označen sa zadanim indeksom (i), i funkcija reset() koja se poziva na kraju svake epohe. U funkciji reset se obično očekuje nasumično miješanje podataka radi veće generalizacije tijekom učenja ali nije nužno da se išta radi.

– **TrainingSupervisor** je osnovni razred zadužen za razrede koji nam pomažu u praćenju procesa učenja mreže. Razredi koji nasljeđuju ovaj razred trebaji implementirat funkciju monitor koja se poziva na kraju svake epohe. Primjeri konkretnih razreda koji nasljeđuju TrainingSupervisor su razredi WeightRecorder (sprema trenutne težine i pragove na disk), Validator (provjerava točnost klasifikacije na proizvoljnom skupu), ActivationVariance (računa varijancu aktivacija i sprema ju u datoteku na disku), GradientVariance (računa varijancu gradijenata i sprema ju u datoteku na disku).

Programski kod 4.7: Pomoćni razredi

```
1  class Initializer
2  {
3  public:
4      virtual void init(vf &weights, int n_in, int n_out) const = 0;
5  };
6
7  class CostFunction
8  {
9  public:
10     CostFunction(int numOutputs);
11     virtual vvf& calculate(const vvf &output, const vf& expectedOutput) = 0;
12     vvf& getPrevError() { return prevError; }
13     float getError() { return error; }
14 protected:
15     vvf prevError;
16     float error;
17     int numOutputs;
18 };
19
20 class InputManager
21 {
22 public:
23     InputManager (int n) : numOfInputs(n) {}
24     virtual vvf& getInput(int i) = 0;
25     virtual vf& getExpectedOutput(int i) = 0;
26     int getInputNum() { return numOfInputs; }
27     virtual void reset() = 0;
28 protected:
29     int numOfInputs;
30 };
31
32 class TrainingSupervisor
33 {
34 public:
35     TrainingSupervisor(std::string path) : path(path) {}
36     virtual void monitor(int epoch = 0) = 0;
```

```
37  protected:
38      std::string path;
39  };
```

5. Eksperimentalni rezultati

5.1. Ispitni skup MNIST

Skup MNIST (engl. Mixed National Institute of Standards and Technology) [9] sadrži 10 klasa rukom pisanih brojeva (od nule do devet). Nekoliko takvih znakova prikazano je na slici **??**. Takav se skup najviše koristio za testiranje ranih sustava automatskog prepoznavanja rukom pisanih brojki kod bankovnih čekova. Slike su monokromatske (8 bitne) te veličine 28×28 točki. Skup je nadalje podijeljen u:

- skup za treniranje - sadrži 60 000 znakova
- skup za ispitivanje - sadrži 10 000 znakova

Radi mogućnosti praćenja učenja te praćenja generalizacije mreže tijekom učenja potreban nam je jedan skup koji nije u skupu za učenje. Ispitni skup nemožemo uzeti u obzir zato što on služi za konačno ispitivanje kvalitet mreže i nesmije se koristiti tijekom učenja, čak ni za praćenje pogreške jer može utjecati na konačne rezultate, tj. moglo bi se desiti da podešavama hiperparametre mreže na taj način da pogodoju baš tom skupu a da ne mreža ne generalizira općenito. Zato ćemo podijeliti originalni skup za treniranje u dva skupa, jedan za treniranje a drugi za validaciju. Skup za validaciju se neće koristiti u treniranju ali će biti od koristi prilikom praćenja učenja i generalizacije mreže. Dakle naša podjela skupa je:

- **skup za treniranje** - sadrži prvih 50 000 znakova originalnog skupa za treniranje
- **skup za validaciju** - sadrži zadnjih 10 000 znakova originalnog skupa za treniranje
- **skup za ispitivanje** - sadrži 10 000 znakova

5.1.1. Predobrada ulaza

Za uspješno učenje mreže, važno je da je aritmetička sredina skupa približno nula (da bi učenje težina moglo krenuti u zahtjevanom smjeru) te da varijanca bude približno jednaka jedan (da bi se zadržao opseg u sredini aktivacijske funkcije). Za izvorni skup ne vrijedi ovo svojstvo. Stoga je svaki uzorak obrađen oduzimanjem aritmetičke sredine (samog uzorka) i normaliziranjem njegove varijance. Ujedno je i svakom uzorku dodan rub širine 2 te je time veličina ulaza proširena na dimenzije 32×32.

6. Zaključak

Zaključak.

LITERATURA

Raspoznavanje objekata konvolucijskim neuronskim mrežama

Sažetak

Sažetak na hrvatskom jeziku.

Ključne riječi: Ključne riječi, odvojene zarezima.

Title

Abstract

Abstract.

Keywords: Keywords.