SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 696

Raspoznavanje objekata dubokim neuronskim mrežama

Vedran Vukotić

Zagreb, srpanj 2014.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 12. ožujka 2014.

DIPLOMSKI ZADATAK br. 696

Pristupnik: **Vedran Vukotić (0112019329)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Raspoznavanje objekata dubokim neuronskim mrežama**

Opis zadatka:

Područje rada je raspoznavanje lokaliziranih objekata u slikama. Razmatraju se metode raspoznavanja koje imaju strukturu duboke neuronske mreže, pod pretpostavkom da je dostupan potpuno označen skup za učenje.
U okviru rada, potrebno je iz literature proučiti različite oblike dubokih neuronskih mreža. Posebnu pažnju posvetiti konvolucijskim mrežama. Razviti postupak učenja konvolucijske mreže korištenjem skupa uzoraka za učenje. Hiperparametre mreže optimirati unakrsnom validacijom. Vrednovati konvergenciju i učinak naučene mreže na ispitnim skupovima MNIST i FER-MASTIF-TS2010. Prikazati i ocijeniti ostvarene rezultate.
Radu priložiti izvorni i izvršni kod razvijenih postupaka, ispitne slijedove i rezultate, uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 14. ožujka 2014.
Rok za predaju rada: 30. lipnja 2014.

Mentor:

Izv.prof.dr.sc. Siniša Šegvić

Djelovođa:

Doc.dr.sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:

Prof.dr.sc. Siniša Srbljić

Zahvala

Zahvaljujem svom mentoru, Siniši Šegviću, na savjetima, strpljenju i stručnoj pomoći. Posebice mu zahvaljujem na podršci mojih mnogobrojnih stručnih praksi, još većem broju prijava te hrpi pitanja o pronalaženju vlastitog puta u stručnom i akademskom usavršavanju.

Josipu Krapcu zahvaljujem na prijedlogu teme ovog rada, mnogobrojnim savjetima oko treniranja konvolucijskih neuronskih mreža te u pruženoj podršci prilikom nastavka mog usavršavanja u Francuskoj.

Zahvaljujem majci i baki na motivaciji i financijskoj pomoći u završavanju ne jednog, već dva fakulteta. Nešto što nikako ne bih uspio bez njih.

Na kraju, želja mi je zahvaliti kolegicama Meliti Kokot i Petri Vučković na nesebičnoj pomoći u ispunjavanju administracijskih obveza na Fakultetu za vrijeme mojih dugotrajnih odsustava tokom međunarodnih stručnih praksi. Naravno, zahvaljujem im i na svim razgovorima i uzajamnoj pomoći tijekom učenja i svladavanja zajedničkih predmeta.

SADRŽAJ

1. Uvod

Standardni se pristup raspoznavanju objekata u računalnom vidu sastoji od uobičajenih koraka: izračun značajki (primjerice SIFT [15]), njihovu agregaciju (primjerice BOW [5]) te učenje nekim klasifikatorom (primjerice SVM [4]).

Problem kod takvih pristupa ima više. Izrada takvog sustava je vremenski zahtjevna. Značajke su općenite, nisu prilagođene problemu a nisu prilagođene ni međuklasnim modalnostima. Sve to utječe na suboptimalnost takvog pristupa.

Duboke neuronske mreže (engl. *Deep Neural Networks - DNN*) predstavljaju pristup u kojemu se sve (od značajki do klasifikacije), u potpunosti, uči automatski na temelju skupa uzoraka. Pozitivne strane takvog pristupa uključuju prilagođenost naučenih značajki konkretnom problemu i njegovom skupu uzoraka, dijeljenje značajki između više klasa te učenje različitih značajki za različite modalnosti pojedinih klasa. S obzirom da su značajke optimalno naučene za određeni problem, takvi sustavi postižu trenutno najbolje rezultate (engl. *state of the art*) u različitim klasifikacijskim problemima.

Cilj ovog rada je izrada duboke neuronske mreže za više-klasnu klasifikaciju. Sustav će se najprije testirati na skupu MNIST [12] rukom pisanih znamenki, koji je uobičajen test za takve arhitekture. Nakon toga će se izgrađeni klasifikator prilagoditi za prepoznavanje prometnih znakova Republike Hrvatske na temelju skupa FER-MASTIF TS2010 [19] [20]. S obzirom na prirodu problema - veliki broj različitih klasa i velike međuklasne razlike sa određenim dijeljenim značajkama (npr. oblik prometnog znaka), očekuje se da će pristup dubokim neuronskim mrežama [3] postići bolje rezultate od ranijih sustava za klasifikaciju.

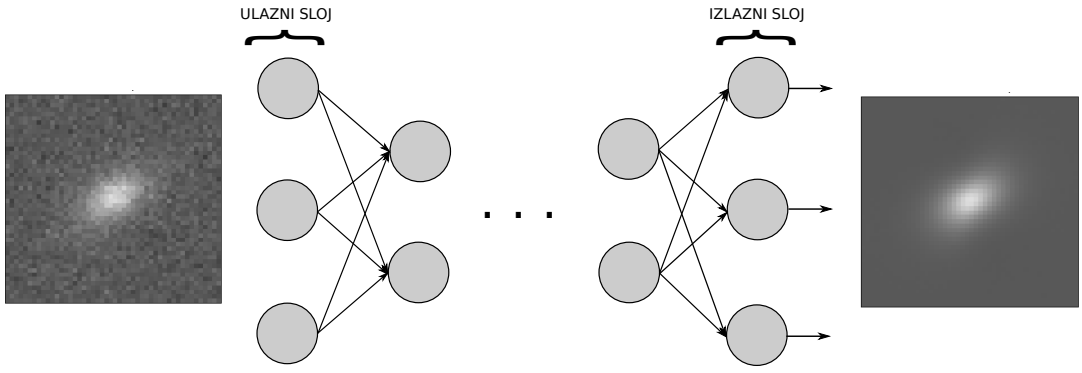
Trenutno najčešće korišten oblik dubokih neuronskih mreža u računalnom vidu su konvolucijske neuronske mreže (engl. *Convolutional Neural Networks - CNN*) [11].

2. Duboke arhitekture učenja

Postoje više različitih dubokih arhitektura, koje se mogu podijeliti u ovisnosti o načinu rada i funkciji. Po funkciji, duboke arhitekture možemo podjeliti na klasifikatore i autoenkodere.

Autoenkoderi imaju jednaku dimenzionalnost ulaza i izlaza, dok su slojevi u sredini uglavnom manje dimezijonalnosti. Svrha autoenkodera je pronalaženje interne reprezentacije uzoraka, odnosno njihovo optimalno kodiranje koje će na izlazu dati uzorak što je moguće sličniji ulazu. Autoenkoderi se često koriste u smanjivanju dimezijonalnosti uzoraka (umjesto metode PCA) ili kao pomoć učenju dubokih klasifikatora.

Klasifikatori imaju dimenzionalnost ulaza jednaku dimenzionalnosti uzoraka, dok je izlaz jednak broju različitih klasi u sustavu. Oni pronalaze značajke koje su relevantne za pojedine klase te im nadalje smanjuju dimenzionalnost i uče njihovu pripadnost pojedinim klasama.



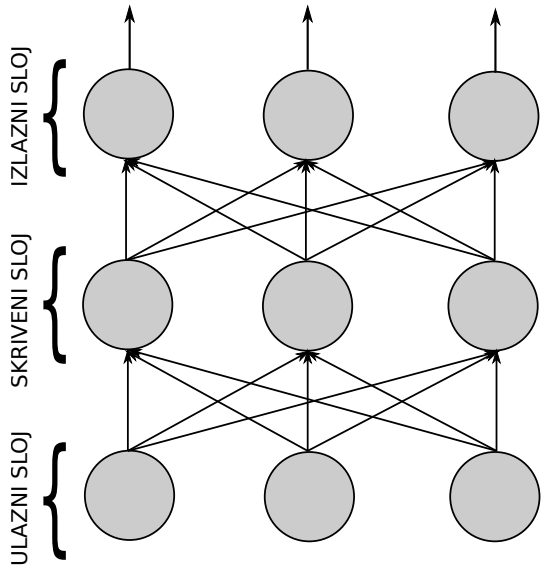
Slika 2.1: Prikaz rada autoenkodera

Po načinu rada postoji veliki broj različitih arhitektura: klasične duboke neuronske mreže, ograničeni Boltzmannovi strojevi (engl. *Restricted Boltzmann Machines*), neuronske mreže s povratnim vezama, konvolucijske neuronske mreže i mnoge druge.

Za postupke vizualne klasifikacije najčešće su korištene upravo konvolucijske neuronske mreže [3] [11] te će isključivo one biti biti promatrane u sklopu ovog rada.

3. Konvolucijske neuronske mreže

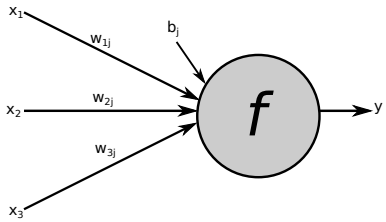
Konvolucijske neuronske mreže mogu se prikazati kao proširenje klasičnih višeslojnih unaprijednih neuronskih mreža. Na slici 3.1 prikazana je jednostavna višeslojna unaprijedna neuronska mreža s jednim ulaznim slojem, jednim izlaznim slojem i jednim skrivenim slojem. Za potrebe klasifikacije broj neurona na ulazu je uvijek jednak dimenzionalnosti uzoraka $|x|$, broj neurona na izlazu jednak broju klasa $|C|$, dok je broj skrivenih slojeva i broj neurona u svakome od njih proizvoljan (uz napomenu da struktura skrivenih slojeva utječe na ekspresivnost same neuronske mreže te je optimalan odabir parametara strukture mreže važan za postizanje dobrih rezultata klasifikacije).



Slika 3.1: Primjer unaprijedne višeslojne neuronske mreže

Unutar svakog neurona nalazi se aktivacijska funkcija f koja uzima sumu umnoška ulaza neurona s pripadnim težinama i praga te ih preslikava na izlaz neurona tako modelirajući ponašanje neurona. Postoje više različitih tipova aktivacijskih funkciji o čemu će biti riječi kasnije. U skladu s time, jedan neuron se može zapisati kao:

$$y_i = \sum_j f(w_{ij}x_j + b_j) \tag{3.1}$$



Slika 3.2: Prikaz jednog neurona

Rad jednog neurona prikazan je na slici 3.2. Trenutni sloj će uvijek biti označen sa j dok će prethodni sloj uvijek nositi oznaku i. Prag (engl. *bias*) b je ekvivalentan pomicanju aktivacijske funkcije po apscisi. Dodavanjem slobode pomicanja po apscisi proširuje se područje djelovanja neurona, čime se mogu lakše modelirati ulazi neurona čije težište nije u ishodištu (uz pretpostavku aktivacijske funkcije koja je simetrična oko nule). Zbog jednostavnosti se često uvodi supstitucija $b_j = 1 * w_{0j}$ kojom se prag može predstaviti kao dodatna sinapsa (težina) koja je uvijek spojena na jedinicu. Tada se funkcija neurona može zapisati kao:

$$y_i = \sum_j f(w_{ij} x_j) \tag{3.2}$$

gdje je $x_0 = 1$. Takvim se zapisom postiže lagana vektorizacija funkcije jednog neurona:

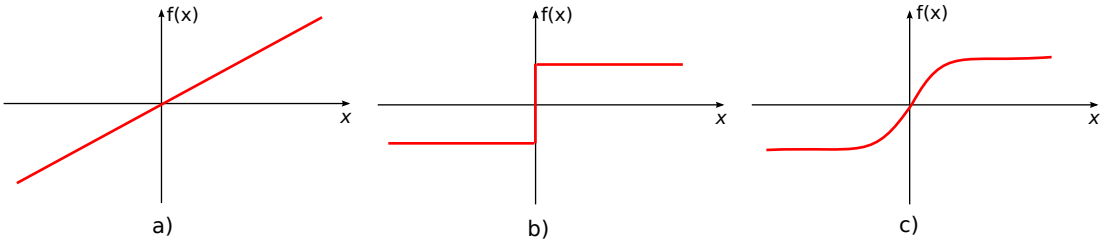
$$y_i = f(\mathbf{w}^T \mathbf{x}) \tag{3.3}$$

Kako svaki sloj tipično sadrži više od jednog neurona, korisno je vektorizirati proračun za čitav sloj. Uvodi se matrica težina $\mathbf{W}$ u kojoj je svaki stupac vektor težina jednog neurona (uključujući težinu za prag) $\mathbf{w}_j$:

$$\mathbf{W} = \begin{bmatrix} w_{01} & \dots & w_{0j} & \dots & w_{0n} \\ \vdots & \dots & \vdots & \dots & \vdots \\ w_{i1} & \dots & w_{ij} & \dots & w_{in} \\ \vdots & \dots & \vdots & \dots & \vdots \\ w_{m1} & \dots & w_{mj} & \dots & w_{mn} \end{bmatrix} \tag{3.4}$$

Uz korištenje proširenog ulaza $\mathbf{x} = [1, x_1, \dots, x_j, \dots, x_m]$ može se operacija čitavog jednog sloja jednostavno zapisati u matričnom obliku:

$$\vec{y} = f(\mathbf{W} \cdot \mathbf{x}) \tag{3.5}$$



Slika 3.3: Najčešće vrste aktivacijskih funkcija: a) linearne funkcije b) funkcije skoka c) sigmoidalne funkcije

Ovakva je formulacija posebno praktična u izvornom kodu, pisanom u višim programskim jezicima, koji inherentno koristi vektorske instrukcije i paralelizaciju (npr. (*Matlab, Python* + *NumPy*, itd.), čime se postižu veće brzine izvođenja.

Postoji poveći izbor aktivacijskih funkcija neurona. Na slici 3.3 prikazani su najčešći primjeri aktivacijskih funkcija: a) linearne aktivacijske funkcije, b) funkcije skoka te c) sigmoidalne funkcije. Linearne funkcije se najčešće koriste u izlaznim slojevima kada je potreban neograničeni izlaz. Neograničen izlaz je često potreban kod regresijskih problema, no u klasifikacijskim problemima (poput onih prezentiranih u ovom radu) neograničen izlaz samo bi otežavao učenje. Kod klasifikacijskih problema smislenije je ograničavati izlaze neurona na male veličine (klasifikacija se vrši odabirom najvećeg izlaza). Funkcije skoka i sigmoidalne funkcije puno su bolji odabir za neuronske mreže koje vrše klasifikaciju. Međutim za učenje neuronskih mreža algoritmom *backpropagation* (unazadnom propagacijom greške), o kojemu će biti riječ u idućem poglavlju, potrebno je raspolagati derivabilnom funkcijom (postoje i aproksimacije algoritma sa po dijelovima derivabilnih funkcija). Zbog toga se najčešće kao aktivacijske funkcije koriste funkcije iz skupine sigmoidalnih funkcija. Najčešće korištene sigmoidalne funkcije su logistička funkcija i sigmoidalna funkcija hiperbolnim tangensom.

Jednadžba logističke aktivacijske funkcije dana je u izrazu (3.6) dok je njena derivacija dana u izrazu (3.7). Dodatni parametar β opisuje "strminu" krivulje u točki infleksije.

$$f(x) = \frac{2}{1 + e^{-\beta x}} - 1 \tag{3.6}$$

$$f'(x) = \frac{2\beta x e}{(1 + e)^2} \tag{3.7}$$

U ovom radu taj hiperparametar mreže nije dodatno optimiran s obzirom da su se konvolucijske mreže s hiperbolnim tangensom pokazale boljima [13].

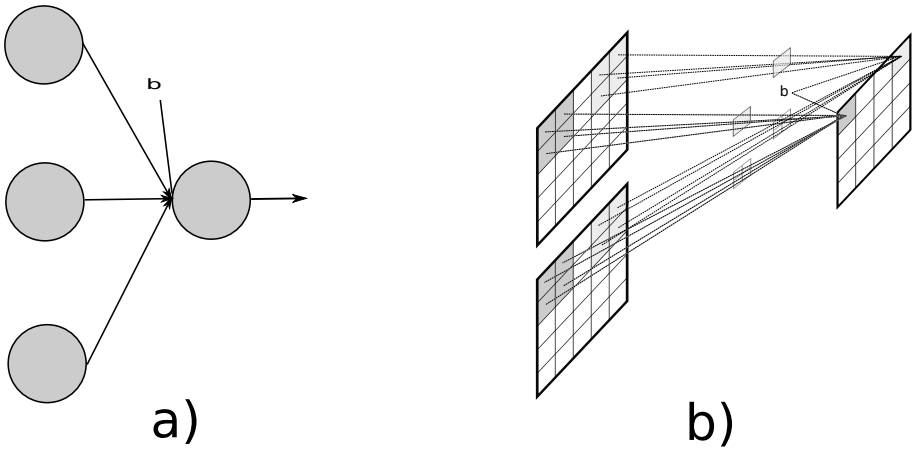
U konvolucijskim neuronskim mrežama se najčešće koristi skalirana sigmoidalna funkcija hiperbolnog tangensa [13] čiji je opis dan u izrazu 3.8 a njena derivacija u izrazu 3.9.

$$f(x) = 1.7159 \tanh\left(\frac{2}{3}x\right) \tag{3.8}$$

$$f'(x) = 1.444\left(1 - \tanh^2\left(\frac{2}{3}x\right)\right) \tag{3.9}$$

Vrijedi napomenuti da o odabiru logističke funkcije ovisi i raspon uzorkovanja vrijednosti za slučajnu inicijalizaciju težina neuronske mreže. O tome će biti riječi u poglavlju 4.2.

U do sada opisanim neuronskim mrežama izlaz svakog neurona je bio skalar. U konvolucijskim neuronskim mrežama uvodi se proširenje na $\mathbf{R}^2$. Izlazi takvih elemenata su dvodimenzionalni te ih nazivamo mapama značajki (engl. *feature maps*). Ulaz je također dvodimenzionalan a umjesto težina koriste se jezgre (engl. *kernels*).

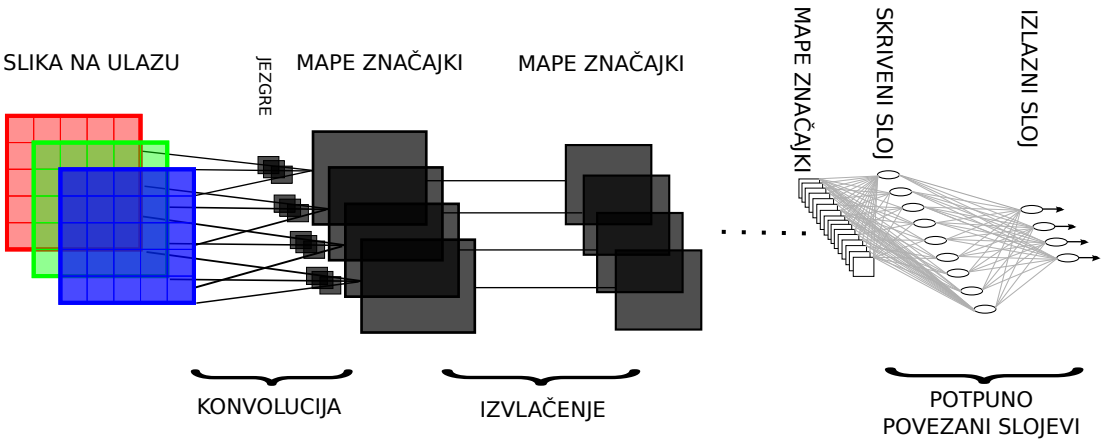


Slika 3.4: Elementi dubokih neuronskih mreža: a) neuron i njegova povezanost b) mapa značajke i primjer njezine povezanosti (jezgre iste boje predstavljaju istu jezgru)

3.1. Struktura konvolucijskih neuronskih mreža

Na slici 3.5 prikazana je opća struktura konvolucijskih neuronskih mreža. Na ulazu može biti jedna monokromatska slika ili višekanalna slika u boji. Zatim slijede naizmjenice konvolucijski slojevi i slojevi sažimanja (engl. *pooling*). Na samom kraju se nalazi nekoliko potpuno povezanih slojeva (klasični perceptron) koji su jednodimenzionalni, uključujući i izlazni sloj. Tipični primjeri konvolucijskih neuronskih mreža imaju oko desetak slojeva (čime jasno opravdavaju svoje mjesto u kategoriji dubokih

neuronskih mreža). Konvolucijski slojevi i slojevi sažimanja imaju dvodimenzionalne "neurone" koji se nazivaju mapama značajki (engl. *feature maps*) koji u svakom sloju postaju sve manjih dimenzija. Zadnji takav sloj je uvijek dimenzija 1×1 (iako je i dalje formalno dvodimenzionalan) te predstavlja vezu na perceptron koji se nalazi u zadnjim slojevima konvolucijske neuronske mreže. Konvolucijski slojevi uvijek imaju vezu jedan-na-više sa prethodnim slojem te uče težine u jezgrama (engl. *kernels*) s kojima obavljaju konvoluciju, dok slojevi sažimanja su uvijek u jedan-na-jedan vezi sa prethodnim slojem te ne uče nikakve vrijednosti (eventualno neke varijante pamte način propagacije greške). Detaljni opisi svakog postupka navedeni su u nastavku. Vrijedi napometuti da konvolucijske neuronske mreže ne moraju nužno imati višeslojni perceptron u krajnjim slojevima. Tako primjerice [13] koristi mrežu sa radijalnim baznim funkcijama (engl. *Radial Base Function Network - RBF Network*) na kraju svoje arhitekture. (Takve mreže najčešće imaju skriveni sloj sa Gaussovim funkcijama na određenim koordinatama te izlazni sumacijski sloj. Što je uzorak dalje od pojedinog centra predstavljenog jednom Gaussovom funkcijom to taj centar ima manje utjecaja na izlaz.) Zbog njihove jednostavnosti i brzine, međutim, sve je popularniji odabir višeslojnih unaprijednih neuronskih mreža sa sigmoidalnim aktivacijskim funkcijama u konačnim slojevima konvolucijskih neuronskih mreža (primjerice [3]).



Slika 3.5: Opća struktura konvolucijskih neuronskih mreža

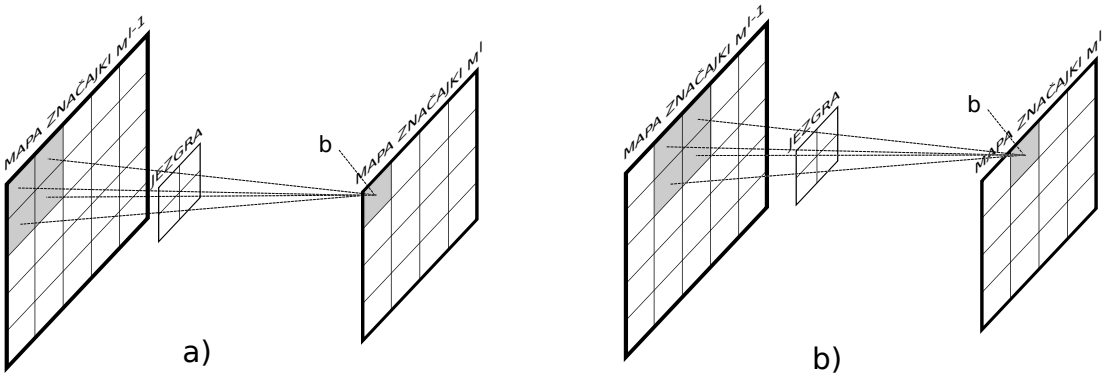
3.2. Konvolucijski slojevi

Konvolucijski slojevi stvaraju mape značajki tako da uzimaju mapu na ulazu sloja (ili uzorak/sliku ukoliko se radi o prvom sloju) te rade dvodimenzionalnu konvoluciju sa jezgrom. Neka je M^l mapa l-tog (trenutnog sloja) a M^{l-1} mapa prethodnog sloja.

Nadalje, neka je M_w širina mape, M_h visina mape, K_w širina jezgre, K_h visina jezgre (obično se kao jezgre biraju kvadratne matrice), S_w korak pomaka jezgre po širini prilikom konvolucije te S_h pomak jezgre po visini prilikom konvolucije. Veličine mape značajki u nekom sloju su tada date izrazima 3.10 i 3.10.

$$M_w^l = \frac{M_w^{l-1} - K_w}{S_w} + 1 \tag{3.10}$$

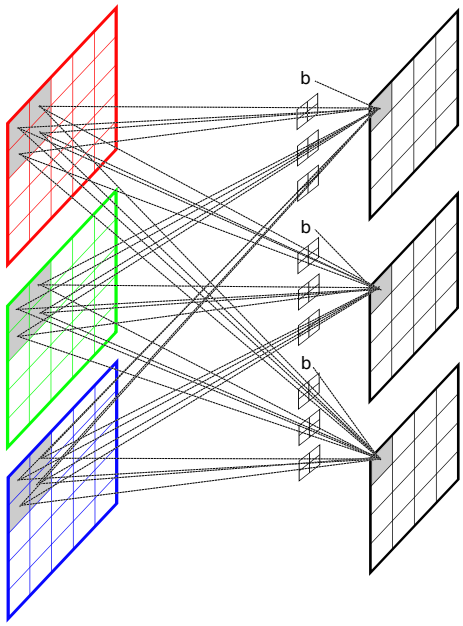
$$M_h^l = \frac{M_h^{l-1} - K_h}{S_h} + 1 \tag{3.11}$$



Slika 3.6: Konvolucija a) Prvi korak b) Idući korak nakon pomaka receptivnog polja za S_w

Konvolucija se tvori "prolazom" kroz ulaznu mapu sa prozorom veličine iste kao i jezgra, umnoškom ulaza sa pripadnim vrijednostima jezgre, sumiranjem dobivenih vrijednosti s pragom, izračunom vrijednosti aktivacijske funkcije te spremanjem u pripadnu lokaciju izlazne mape. Takav prozor definiran u ulaznoj mapi, naziva se vizualnim ili receptivnim poljem određenog neurona izlazne mape. Zatim se prozor pomiče u koracima S_w (odnosno S_h nakon dolaska do brida ulazne mape) te se postupak ponavlja za idući neuron u izlaznoj mapi. Ilustracija takvog postupka dana je na slici 3.6. Ovakva konvolucija opisuje samo relaciju između jedne ulazne mape značajki sa jednom izlaznom mapom dok konvolucijski slojevi često imaju jedan-na-više vezu sa prethodnim slojevima (osim ukoliko se radi o prvom konvolucijskom sloju koji se veže na monokromatski ulaz koji je predstavljen isključivo jednom mapom). Svaka veza sloja sa prethodnim slojem predstavljena je jezgrom. Svaka mapa stoga mora imati onoliko jezgara koliko je mapa u prethodnom sloju na kojem je spojena.

Neka je k jedna mapa značajki sloja M^l ($k \in M^l$). Sa $x_k^l(i,j)$ je označen element na poziciji (i,j) mape k sloja M^l. Svaka mapa značajki ima jedan zajednički prag (engl. *bias*) b_k^l te onoliko jezgri koliko ima mapa u prethodnom sloju. Veličine jezgre je označena sa $K_w^k x K_h^k$. Neka jezgra mape k sloja l na lokaciji (x,y) ima vrijednost



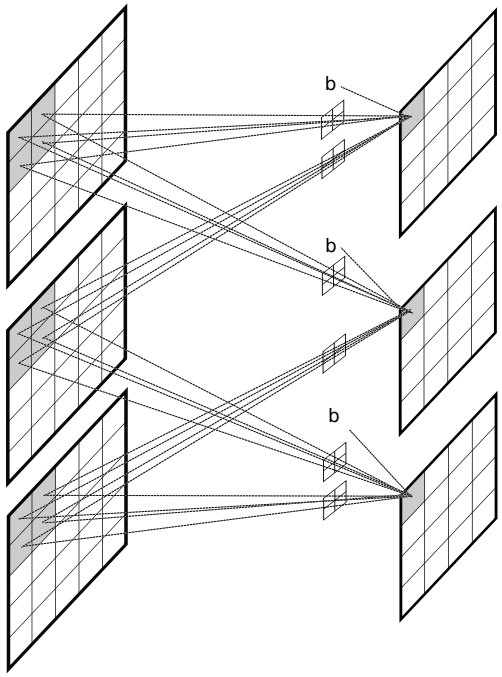
Slika 3.7: Prvi korak konvolucija (za svaku mapu značajki u sloju) kod potpuno povezanih slojeva

$w_k^l(x, y)$. Vrijednost mape k' sloja l na lokaciji (i, j) je tada dana u izrazu 3.12.

$$x_{k'}^l(i,j) = f\left(\sum_{k \in M^{l-1}} \sum_{\substack{0<m\leq K_w^k \\ 0<n\leq K_h^k}} x_k^{l-1}(i+m, j+n) w_{k'}^l(m,n) + b_k^l\right) \tag{3.12}$$

Većina autora, poput primjerice [18] i [3] koriste potpuno povezane konvolucijske slojeve u kojima je svaka mapa značajki trenutnog sloja povezana sa svim mapama značajki prethodnog sloja. U sloju l se stoga nalazi $M^l M^{l-1}$ jezgara koje definiraju dijeljenje (između neurona unutar pojedine mape) težine kojima se radi konvolucija između trenutnog i prethodnog sloja. Pojednostavljena ilustracija takve mreže bila je dana na slici 3.7. Za razliku od potpuno povezanih konvolucijskih slojeva, [13] koristi raspršenu (engl. *sparse*) povezanost u kojoj nisu sve mape značajki trenutnog sloja povezane sa svim ostalim mapama značajki prethodnog sloja. Primjer takve raspršene povezanosti ilustriran je na slici 3.8 dok je pripadna matrica povezanosti dana u izrazu 3.13. U slučaju korištenja raspršenog povezivanja potrebno je definirati matricu povezanosti $\mathbf{C}$ veličine $M^l \times M^{l-1}$ koja pokazuje koje su mape značajki trenutnog sloja povezane s mapama značajki prethodnog sloja.

$$\mathbf{C} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \tag{3.13}$$



Slika 3.8: Primjer raspršene povezanosti u kojoj je svaka mapa povezana samo sa dvije mape iz prethodnog sloja

Za slučaj raspršenog povezivanja izraz 3.12 postaje jednak izrazu 3.14.

$$x_{k'}^{l}(i,j) = f\left(\sum_{\substack{k \in M^{l-1} \\ \mathbf{C}(k,k')=1}} \sum_{\substack{0<m\leq K_w^k \\ 0<n\leq K_h^k}} x_k^{l-1}(i+m, j+n) w_{k'}^{l}(m,n) + b_k^l\right) \tag{3.14}$$

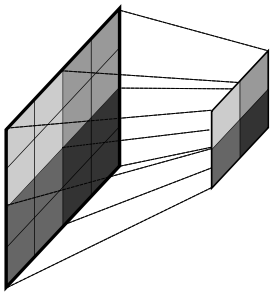
Nakon izračuna vrijednosti za element (i, j) izlazne mape značajki, prelazi se na idući element $(i + S_w, j)$ odnosno $(1, j + S_h)$ ukoliko se došlo do kraja retka. Korak (engl. *step, skipping factor*) služi za ubrzavanje rada mreže te djeluje na neki način kao i slojevi sažimanja značajki (o kojima će biti riječi u nastavku) samo bez ikakve ugrađene heuristike o tome što zanemariti. Najčešće implementacije [3] [11] jednostavno koriste $S_w = S_h = 1$ čime tvore običnu konvoluciju. U nastavku je prikazan postupak određivanja vrijednosti mapi značajka (propagacija unaprijed) za konvolucijsku vezu između dva sloja sa matricom povezanosti $\mathbf{C}$ te koracima $S_w = S_h = 1$. Radi se zapravo o implementaciji izraza (3.14).

Pseudokod 3.1: Unaprijedna propagacija konvolucijskog sloja

```
za svaku mapu k iz trenutnog sloja M
  za svaku mapu kp iz prethodnog sloja Mp
    ako je C(kp, k) == 1
      za i = 0 do visine mape k
        za j = 0 do sirine mape k
          suma = 0
          za y = 0 do visine jezgre K(kp, k)
            za x = 0 do sirine jezgre K(kp,k)
              suma += Mp(k, i + y, j + x) * K(kp, k, x, y)
          suma += bias(k)   # dodavanje praga pripadne mape
          M(k, i, j) = f(suma) # aktivacijska funkcija
```

3.3. Slojevi sažimanja

Postupak sažimanja (engl. *pooling*) smanjuje rezoluciju mapi značajki te povećava prostornu invanrijantnost (neosjetljivost na manje pomake značajki u uzorku/slici) neuronske mreže. Postupak se provodi tako da se $S_w \times S_h$ podataka grupira zajedno te ih predstavi jednom vrijednošću. Prvi način sažimanja se sastojao od predstavljanja grupe njezinom aritmetičkom sredinom [13]. Kasnije je pokazano [16] da se predstavljanje maksimalnom vrijednošću regije ponaša puno bolje (na razini čitavog klasifikatora), osim što je i brže za računanje te je većina današnjih konvolucijskih mreža sa takvom metodom sažimanja značajki. Važno je napomenuti da broj mapa ostaje isti prilikom postupaka sažimanja. U nastavku su opisane pojedine metode sažimanja značajki.



Slika 3.9: Operacija sažimanja značajki (engl. *pooling operation*)

3.3.1. Sažimanje usrednjavanjem

Sažimanje usrednjavanjem (engl. *Mean-pooling*) je postupak koji, kako mu i ime kaže, uzima vrijednosti iz pojedinih regija te ih predstavlja aritmetičkom sredinom svih ele-

menata grupe. Primjerice, neka je početna mapa $\mathbf{M} = \begin{bmatrix} 3 & 4 & 2 & 7 \\ 2 & 3 & 2 & 5 \\ 1 & 9 & 6 & 4 \\ 4 & 2 & 3 & 7 \end{bmatrix}$ te neka su faktori skaliranja $S_h = S_w = 2$. Tada je mapa na izlazu jednaka $\mathbf{M}' = \begin{bmatrix} 3 & 4 \\ 4 & 5 \end{bmatrix}$ Zbog algoritma unazadne propagacije (engl. *Backpropagation algorithm*), o kojemu će biti riječ u idućem poglavlju, potrebno je moći obaviti suprotnu operaciju: sa trenutnog sloja propagirati greške unazad u prethodni sloj. Takva je operacija u ovom slučaju trivijalna jer svaki element u grupi jednako sudjeluje u stvaranje reprezentacije u idućem sloju. (Kod drugih metoda sažimanja značajki to nije slučaj te je potrebno "zapamtiti" kako se izvršila unaprijedna propagacija da bi se kasnije moglo provesti širenje greške unazad).

3.3.2. Sažimanje maksimalnom vrijednošću

Sažimanje maksimalnom vrijednošću (engl. *Max-pooling*) je metoda sažimanja značajki koja odabire maksimalnu vrijednost unutar grupe te ju propagira dalje u idući sloj. U [16] je pokazano kako se takva metoda ponaša puno bolje od metode sažimanja usrednjavanjem. Unaprijedna propagacija je jednostavna te bi za prethodni primjer mape $\mathbf{M} = \begin{bmatrix} 3 & 4 & 2 & 7 \\ 2 & 3 & 2 & 5 \\ 1 & 9 & 6 & 4 \\ 4 & 2 & 3 & 7 \end{bmatrix}$ i faktora skaliranja $S_h = S_w = 2$ nastala nova mapa $\mathbf{M}' = \begin{bmatrix} 4 & 7 \\ 9 & 7 \end{bmatrix}$ Postupak širenja greške unazad je složeniji te je potrebno za svaku grupu označiti gdje se nalazio element koji je odabran kao maksimum i propagiran dalje da bi se moglo izvršiti širenje greške unazad. Lokacije se mogu zapisivati relativno (s obzirom na prozor) ili apsolutno (s obzirom na čitavu ulaznu mapu). Relativni zapis odabranih lokacija za prethodni primjer glasi: $\mathbf{L} = \begin{bmatrix} (0,1) & (0,1) \\ (0,1) & (1,1) \end{bmatrix}$ dok bi apsolutni zapis bio jednak $\mathbf{L} = \begin{bmatrix} (0,1) & (0,3) \\ (2,1) & (3,3) \end{bmatrix}$. U programskoj se implementaciji ovog rada koristi apsolutni zapis jer se tako izbjegava potreba za preračunavanjem lokacije u izvornoj (prethodnoj) mapi. Primjerice, za okno veličine 3×3 može se koristiti Ga-

ussova jezgra $\mathbf{G} = \frac{1}{16} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ gdje se jasno vidi kako u težinskoj sumi najveću vrijednost poprima element na sredini.

3.3.3. Sažimanje Gaussovim usrednjavanjem

Ova tehnika sažimanja je vrlo slična sažimanju usrednjavanjem aritmetičke sredine. Sredina se izračunava pomoću Gaussove jezgre G te se veće težine pridodaju elementima na sredini a manje elementima koji su dalje u oknu.

3.3.4. Sažimanje metrikom Lp

Sažimanje metrikom Lp predstavlja generalizaciju prethodnih dviju metoda. Takva metoda je korištena primjerice u [17] dok je teoretska osnova dana u [2]. Ova se metoda sažimanja (engl. *pooling*) provodi konvolucijom ulazne mape $\mathbf{M}$ sa Gaussovom jezgrom $\mathbf{G}$ i hiperparametrom P. Metoda sažimanja metrikom Lp je opisana izrazom (3.15).

$$\mathbf{M}' = \left(\sum \sum M(i,j)^P G(i,j) \right)^{\frac{1}{P}} \tag{3.15}$$

Za $P = 1$ Lp-pooling postaje Gaussovo usrednjavanje dok za $P \to \infty$ Lp-pooling postaje jednak metodi sažimanja maksimumom. Optimiranjem hiperparametra P moguće je postići bolje rezultate nego metodom sažimanja maksimumom, međutim u ovom radu to neće biti učinjeno te se svi slojevi sažimanja značajki temelje na metodi sažimanja maksimumom kao dobar kompromis između jednostavnosti i kvalitete.

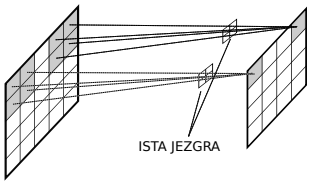
3.4. Svojstva konvolucijskih neuronskih mreža

Konvolucijske neuronske mreže imaju nekoliko svojstava koja im omogućavaju dobru generalizaciju prilikom višeklasne klasifikacije. To su redom:

Dijeljenje težina

U konvolucijskim slojevima se za svaku konvoluciju jedne izvorne mape sa jednom izlaznom mapom koristi jedna jezgra (engl. *kernel*). Ukoliko se promatraju pojedini neuroni unutar mape, jasno je da svi ti neuroni dijele iste $K_w \times K_h$ težine. Takvo dijeljenje težina omogućava da mreža nauči relevantne i diskriminativne značajke. Jezgre se specijaliziraju za određenu funkciju (primjerice

detekcija horizontalnih i vertikalnih bridova, odziv na različite uzorke i sl.) te postaju slične npr. Haarovim i drugim značajkama. Bez dijeljenja težina dijelovi neuronske mreže bi pretrenirali na određeni detalj podataka. S dijeljenjem, na istu jezgru dolaze različiti podatci što povećava općenitost naučene značajke i poboljšava generalizacijske sposobnosti mreže. Detaljan prikaz naučenih vrijednosti za različite klasifikacijske probleme (skupove učenja) naveden je na kraju rada.



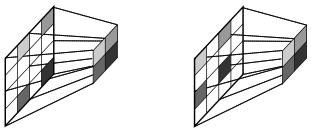
Slika 3.10: Dijeljenje težina (jezgara) u konvolucijskim slojevima

Raspršena povezanost

Na slici 3.8 prikazana je raspršena povezanost. Korištenje raspršene povezanosti može uvelike pomoći u učenju različitih značajki ukoliko je skup za učenje takav da mreža ima težnju konvergiranju istom manjem broju značajki. Bez raspršene povezanosti (potpuna povezanost) sve mape primaju sve vrijednosti iz prethodnih mapa. U tom slučaju je moguće da dvije ili više mape konvergiraju ka istoj vrijednosti. Uvođenjem raspršene povezanosti mape dobivaju različite ulaze (samo neke mape prethodnog sloja) čime se osigurava konvergencija ka različitim vrijednostima. Takav pristup korišten je u [13].

Translacijska invarijantnost

Translacijska invarijantnost omogućava konvolucijskoj neuronskoj mreži da bude otporna na male varijacije položaja značajki. Primarni mehanizam kojim se to postiže su slojevi sažimanja značajki koji smanjuju rezoluciju (odnosno dimenzionalnost) mapi značajki. S obzirom da se to postiže postepeno kroz više takvih slojeva, mreža i dalje uči međusobni položaj značajki (npr. očiju, nosa i ustiju kod detekcije lica) ali postaje otporna na manje varijacije u položaju.



Slika 3.11: Translacijska invarijatnost: Obije ulazne mape rezultiraju jednakom izlaznom mapom, usprkos manjim translacijama značajki

4. Učenje konvolucijskih neuronskih mreža

Najčešće korišten algoritam u učenju neuronskih mreža je algoritam backpropagation (algoritam širenja greške unatrag). Takav je algoritam konceptualno jednostavan (temelji se na gradijentnom spustu), računski isplativ te se kroz praksu pokazao kao veoma pouzdan algoritam. Algoritam backpropagation je moguće jednostavno proširiti na konvolucijske neuronske mreže uz obraćanje pažnje na nekoliko specifičnosti u konvolucijskim slojevima i slojevima sažimanja značajki. U nastavku je naprije opisan klasični algoritam backpropagation, a zatim se uvode proširenja na konvolucijske neuronske mreže.

4.1. Slučajna inicijalizacija težina

Dobra slučajna inicijalizacija težina poboljšava učenje neuronskih mreža postavljanjem težina u takvom rasponu aktivacijske funkcije da učenjem mogu krenuti u oba smjera. Jasno je stoga da raspon uzorkovanja ovisi o aktivacijskoj funkciji koja se koristi. Raspon iz kojeg se uzorkuju slučajne težine iz uniformne distribucije je za logističku funkciju [6] :

$$\left[\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}\right] \tag{4.1}$$

Za skaliranu aktivacijsku funkciju hiperbolnog tangensa se težine uzorkuju iz uniformne distribucije intervala [6]:

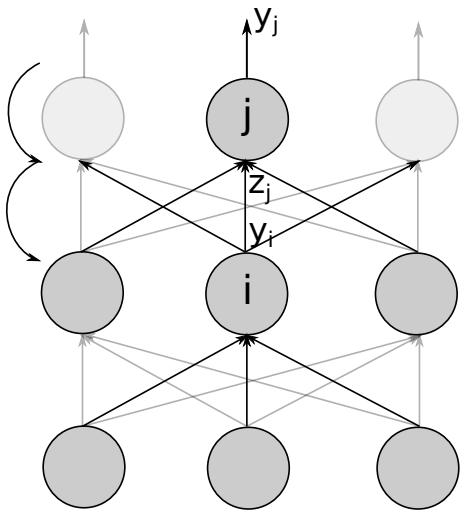
$$\left[-4\sqrt{\frac{6}{n_{in}+n_{out}}}, 4\sqrt{\frac{6}{n_{in}+n_{out}}}\right] \tag{4.2}$$

Uz prikladnu inicijalizaciju težina, potrebno je obaviti i predobradu ulaznih podataka centriranjem u ishodište te normiranjem varijacije na jedan. O tome će biti riječi

u poglavlju 6.1.3.

4.2. Algoritam Backpropagation

Učenje neuronske mreže se provodi u dva koraka. U prvom se koraku dovodi poznati uzorak na ulaz, računa se odziv mreže te se računa pogreška (razlika odziva mreže i očekivanog izlaza za dati poznati uzorak). U drugom se koraku greška širi unazad te se mreža ažuirira kako bi se greška smanjila. Ideja algoritma backpropagation je određivanje greške i gradijenata u svakom sloju te ažuriranje težina na temelju gradijenata tako smanjujući grešku neuronske mreže (gradijentni spust). Najprije se izračunavaju greške izlaznog sloja. Zatim se za prethodni sloj određuje koliko je svaki neuron utjecao na greške u idućem sloju te se računaju njihove greške. Na kraju se određuje gradijent greške po pojedinim težinama koju povezuju te slojeve te se one ažuriraju. Prikaz jednog koraka algoritma backpropagation ilustriran je na slici 4.1.



Slika 4.1: Algoritam Backpropagation

U svim oznakama vrijedi konvencija označavanja trenutnog sloja sa j te prethodnog sloja sa i. Stoga y_j označava izlaz j-tog neurona trenutnog sloja, z_j ukupan ulaz j-tog neurona trenutnog sloja, y_i izlaz i-tog neurona prethodnog sloja te w_{ij} težina koja spaja i-ti neuron prethodnog sloja sa j-tim neuronom trenutnog sloja.

Najprije se računa greška trenutnog sloja:

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j} \tag{4.3}$$

Prvi dio desne strane izraza (4.3) predstavlja parcijalnu derivaciju greške po iz-

lazu neurona $\left(\frac{\partial E}{\partial y_j}\right)$ dok drugi dio predstavlja parcijalnu derivaciju izlaza neurona po ulazu neurona $\left(\frac{\partial y_j}{\partial z_j}\right)$. Greška izlaza neurona se obično računa srednjom kvadratnom pogreškom:

$$E = \frac{1}{2}\sum_j (t_j - y_j)^2 \tag{4.4}$$

U prethodnom izrazu t_j predstavlja očekivanu vrijednost na izlazu neurona j dok y_j i dalje predstavlja izlaz izlaz j-tog neurona. Parcijalna derivacija greške po izlazu neurona je tada:

$$\frac{\partial E}{\partial y_j} = \frac{1}{2}\frac{\partial}{\partial y_j}(t_j - y_j)^2 = \frac{1}{2}\frac{\partial}{\partial y_j} t_j^2 - 2t_j y_j + y_j^2 = y_j - t_j = -(t_j - y_j) \tag{4.5}$$

Uzme li se, primjerice, logistička sigmoidalna funkcija $y_j = f(z_j) = \frac{1}{1+e^{-z}}$ desni dio izraza (4.3) postaje:

$$\begin{aligned}\frac{\partial y_j}{\partial z_j} = \frac{\partial}{\partial z_j}\left(\frac{1}{1+e^{-z_j}}\right) = \frac{\partial}{\partial z_j}\left(1+e^{-z_j}\right)^{-1} = \\ = \frac{-1(-e^{-z_j})}{(1+e^{-z_j})^2} = \frac{1}{1+e^{-z_j}}\frac{e^{-z_j}}{1+e^{-z_j}} = y_j\frac{e^{-z_j}}{1+e^{-z_j}} = \\ = y_j\frac{(1+e^{-z_j})-1}{1+e^{z_j}} = y_j\frac{1+e^{-z_j}}{1+e^{-z_j}}\frac{-1}{1+e^{-z_j}} = y_j(1-y_j)\end{aligned} \tag{4.6}$$

Nakon računanja greške trenutnog sloja izrazom (4.3), greška se propagira na prethodni sloj sumiranjem utjecaja neurona i na sve neurone trenutnog sloja j:

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial y_i} = \sum_j w_{ij}\frac{\partial E}{\partial z_j} \tag{4.7}$$

Na samom kraju, određuju se parcijalne derivacije po težinama:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}} = \frac{\partial E}{\partial z_j} y_i \tag{4.8}$$

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

$$w_{ij} \leftarrow w_{ij} - \eta\frac{\partial E}{\partial w_{ij}} \tag{4.9}$$

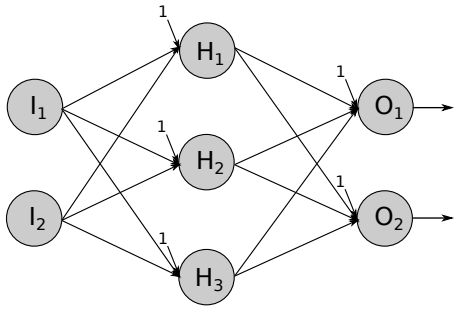
U nastavku je algoritam Backpropagation, koji je u prethodnom dijelu formalno definiran, opisan pseudokodom:

Pseudokod 4.1: Algoritam backpropagation

```
# izracunaj greske izlaznog sloja
za svaki neuron trenutnog sloja j
  outErr(j) = (expected_out(j) - out(j)) * F_deriv(out(j))

# izracunaj greske prethodnog sloja
za svaki neuron prethodnog sloja i
    prevErr(i) = 0
  za svaki neuron trenutnog sloja j
    prevErr(i) += out(j) * w(i,j) * F_deriv(prevOut(i))

# izracunaj promjene tezina
za svaki neuron prethodnog sloja i
  za svaki neuron trenutnog sloja j
    deltaW(i,j) = prevOut(i) * outErr(j)

# azuriraj tezine
nakon sto su greske i promjene tezina izracunate za sve
    slojeve
  za svaki neuron prethodnog sloja i
    za svaki neuron trenutnog sloja j
      W(i,j) -= eta * deltaW(i,j)

# backpropagation ostalih slojeva
ponavljaj za sve ostale slojeve
  prethodni sloj postaje trenutni sloj
  # greske trenutnog sloja su izracunate u prethodnom koraku

  # izracunaj greske prethodnog sloja
  za svaki neuron prethodnog sloja i
     prevErr(i) = 0
    za svaki neuron trenutnog sloja j
      prevErr(i) += out(j) * w(i,j) * F_deriv(prevOut(i))

  # izracunaj promjene tezina
  za svaki neuron prethodnog sloja i
    za svaki neuron trenutnog sloja j
      deltaW(i,j) = prevOut(i) * outErr(j)

  # azuriraj tezine
  nakon sto su greske i promjene tezina izracunate za sve
     slojeve
    za svaki neuron prethodnog sloja i
      za svaki neuron trenutnog sloja j
        W(i,j) -= eta * deltaW(i,j)
```

4.2.1. Numerički primjer algoritma Backpropagation

U nastavku je prikazan rad algoritma Backpropagation na primjeru jednostavne neuronske mreže arhitekture $2 \times 3 \times 2$ (dva ulazna neurona, tri neurona u skrivenom sloju i dva izlazna neurona). Na slici 4.2 je prikazana takva arhitektura.



Slika 4.2: Arhitektura neuronske mreže na kojoj je ilustriran rad algoritma Backpropagation

Težine su označene koristeći notaciju oblika: w_{ij}^L. Takva težina povezuje i-ti neuron $L-1$-og sloja sa j-tim neuronom L-tog sloja. Radi jednostavnosti sve su težine jednog neurona zapisane vektorski, sa zadnjim elementom koji označava težinu prema pragu (engl. *bias*): $\mathbf{w}_j^L = \left[w_{1j}^L w_{2j}^L, ..., w_{nj}^L, w_b^L\right]$. Neka su težine skrivenog sloja sljedeće: $\mathbf{w}_1^H = [0.1, 0.2, -0.1]$, $\mathbf{w}_2^H = [-0.3, 0.1, 0.2]$ te $\mathbf{w}_3^H = [0.2, -0.1, 0.1]$. Težine izlaznog sloja neka budu: $\mathbf{w}_1^O = [0.1, 0.2, 0.1, -0.2]$ i $\mathbf{w}_2^O = [0.3, 0.2, 0.1, -0.3]$. Neka je na ulazu uzorak $\mathbf{x} = [1, 0]$, neka se na izlazu očekuje $\mathbf{t} = [1, 0]$, neka je aktivacijska funkcija logistička te neka je stopa učenja $\eta = 0.1$.

Prvi korak predstavlja računanje odziva neuronske mreže. Računanje se provodi krećući s ulaza prema izlazu. Najprije izračunavamo vrijednosti skrivenog sloja: Izlaz prethodnog sloja se proširuje za 1 zbog težine praga te postaje $\mathbf{x} = [101]$.

Vrijednost prvog neurona: $y_1^H = f(z) = f(\mathbf{w_1^H}^T\mathbf{x}) = f(0.1 \cdot 1 + 0.2 \cdot 0 - 0.1 \cdot 1) = f(0) = \frac{1}{1+e^{-0}} = 0.5$

Vrijednost drugog neurona: $y_2^H = f(z) = f(\mathbf{w_2^H}^T\mathbf{x}) = f(-0.3 \cdot 1 + 0.1 \cdot 0 - 0.2 \cdot 1) = f(-0.1) = \frac{1}{1+e^{0.1}} = 0.48$

Vrijednost trećeg neurona: $y_3^H = f(z) = f(\mathbf{w_3^H}^T\mathbf{x}) = f(0.2 \cdot 1 + -0.1 \cdot 0 + 0.1 \cdot 1) = f(0.3) = \frac{1}{1+e^{-0.3}} = 0.57$

Nakon što su izračunate vrijednosti prvog skrivenog sloja, kreće se na računanje vrijednosti idućeg sloja (u ovom slučaju se to radi o izlaznom sloju). Ulaz u izlazni sloj se proširuje za 1: $x^O = [0.5, 0.48, 0.57, 1]$

Vrijednost prvog izlaza: $y_1^O = f(z) = f(\mathbf{w}1^H)^T\mathbf{x}) = f(0.1 \cdot 0.5 + 0.2 \cdot 0.48 + 0.1 \cdot 0.57 - 0.2 \cdot 1) = f(0.003) = \frac{1}{1+e^{-0.003}} = 0.50$

Vrijednost drugog izlaza: $y_2^O = f(z) = f(\mathbf{w}1^H)^T\mathbf{x}) = f(0.3 \cdot 0.5 + 0.2 \cdot 0.48 + 0.1 \cdot 0.57 - 0.3 \cdot 1) = f(0.003) = \frac{1}{1+e^{-0.003}} = 0.50$

Time je gotovo računanje odziva mreže. Idući korak predstavlja određivanje greške, širenje greške unazad te učenje same mreže (ažuriranje težina).

Najprije se određuju greške izlaznog sloja:

$\frac{\partial E_1}{\partial z_1} = (t_1 - y_1^O) * f'(y_1^O) = (t_1 - y_1^O)y_1^O(1 - y_1^O) = (0 - 0.5)0.5(1 - 0.5) = -0.125$

te

$\frac{\partial E_2}{\partial z_2} = (t_2 - y_2^O) * f'(y_2^O) = (t_2 - y_2^O)y_2^O(1 - y_2^O) = (1 - 0.5)0.5(1 - 0.5) = 0.125$

Zatim se određuju parcijalne derivacije grešaka po težinama izlaznog sloja, na način prikazanim s početkom na 11. liniji pseudokoda:

$\frac{\partial E_1}{\partial w_{11}^O} = y_1^H \cdot \frac{\partial E_1}{\partial z_1} = -0.125 \cdot 0.5 = -0.060$
$\frac{\partial E_1}{\partial w_{21}^O} = y_1^H \cdot \frac{\partial E_1}{\partial z_2} = -0.125 \cdot 0.48 = -0.060$
$\frac{\partial E_1}{\partial w_{31}^O} = y_1^H \cdot \frac{\partial E_1}{\partial z_3} = -0.125 \cdot 0.57 = -0.071$
$\frac{\partial E_1}{\partial w_{b1}^O} = y_1^H \cdot \frac{\partial E_1}{\partial z_b} = -0.125 \cdot 1 = -0.125$
$\frac{\partial E_2}{\partial w_{12}^O} = y_2^H \cdot \frac{\partial E_2}{\partial z_1} = 0.125 \cdot 0.5 = 0.060$
$\frac{\partial E_2}{\partial w_{22}^O} = y_2^H \cdot \frac{\partial E_2}{\partial z_2} = 0.125 \cdot 0.48 = 0.060$
$\frac{\partial E_2}{\partial w_{32}^O} = y_2^H \cdot \frac{\partial E_2}{\partial z_3} = 0.125 \cdot 0.57 = 0.071$
$\frac{\partial E_1}{\partial w_{b1}^O} = y_1^H \cdot \frac{\partial E_1}{\partial z_b} = -0.125 \cdot 1 = 0.125$

Greška se širi unazad te se izračunavaju greške skrivenog sloja na način prikazan u bloku petog retka pseudokoda:

$\frac{\partial E}{\partial y_1^H} = 0.5 \cdot 0.1 \cdot f'(0.5) + 0.5 \cdot 0.3 \cdot f'(0.5) = 0.124$
$\frac{\partial E}{\partial y_2^H} = 0.5 \cdot 0.2 \cdot f'(0.48) + 0.5 \cdot 0.2 \cdot f'(0.48) = 0.124$
$\frac{\partial E}{\partial y_2^H} = 0.5 \cdot (-0.2) \cdot f'(0.57) + 0.5 \cdot (-0.3) \cdot f'(0.57) = -0.159$

Dalje bi se nastavilo na identičan način te bi se izračunale parcijalne greške po težinama skrivenog sloja. Na kraju, kada su sve greške i parcijalne derivacije po težinama izračunate, pristupa se ažuriranju težina na način prikazanim u pseudokodu s početkom na liniji 16:

$w_{11}^{O'} \leftarrow w_{11}^O - \eta\frac{\partial E}{\partial w_{11}^O} = 0.1 - 0.1 \cdot (-0.060) = 0.106$
$w_{21}^{O'} \leftarrow w_{21}^O - \eta\frac{\partial E}{\partial w_{21}^O} = 0.2 - 0.1 \cdot (-0.060) = 0.206$
$w_{31}^{O'} \leftarrow w_{21}^O - \eta\frac{\partial E}{\partial w_{31}^O} = 0.1 - 0.1 \cdot (-0.071) = 0.107$
$w_{b1}^{O'} \leftarrow w_{b1}^O - \eta\frac{\partial E}{\partial w_{b1}^O} = -0.2 - 0.1 \cdot (-0.125) = 0.188$
$w_{12}^{O'} \leftarrow w_{12}^O - \eta\frac{\partial E}{\partial w_{12}^O} = 0.3 - 0.1 \cdot 0.060 = 0.294$
$w_{22}^{O'} \leftarrow w_{22}^O - \eta\frac{\partial E}{\partial w_{22}^O} = 0.2 - 0.1 \cdot 0.060 = 0.194$
$w_{32}^{O'} \leftarrow w_{22}^O - \eta\frac{\partial E}{\partial w_{32}^O} = 0.1 - 0.1 \cdot 0.071 = 0.093$
$w_{b2}^{O'} \leftarrow w_{b2}^O - \eta\frac{\partial E}{\partial w_{b2}^O} = -0.3 - 0.1 \cdot 0.125 = -0.312$

Na isti se način ažuriraju težine ostalih slojeva. Na kraju se na ulaz dovodi drugi uzorak te se postupak ponavlja sve dok mreža ne postigne zadovoljavajuću grešku.

4.3. Specifičnosti konvolucijskih slojeva

Konvolucijske neuronske mreže, za razliku od klasičnih neuronskih mreža, imaju svojstvo dijeljenja težina. To se upravo dešava u konvolucijskim slojevima gdje konvolucijska jezgra (koja zapravo predstavlja težine w_{ijk}) utječe na sve neurone u izlaznoj mapi. Zbog toga je prilikom računanja unazadne propagacije potrebno izračunati vrijednosti grešaka, promjena težina po svim neuronima mapa te sumirati njihov utjecaj.

Pseudokod 4.2: Algoritam backpropagation za konvolucijske slojeve

```
1
2  # inicijalizacija greski svake mape na 0
3  prevErr = [
4  [ [0, ..., 0],
5    ...
6    [0, ..., 0]],
7
8   ...
9  [ [0, ..., 0],
10   ...
11   [0, ..., 0]]
12 ]
13
14 ]
15
16 # racunanje greske mapi
17 za i = 0 do broja mapi u prethodnom sloju
18   za j = 0 do broja mapi u trenutnom sloju
19     # jezgra K(i,j) povezuje i-tu mapu prethodnog sloja
20     # sa j-tom mapom trenutnog sloja te je velicine w x h
21     za y = 0 do visine mape j
22       za x = 0 do sirine mape j
23       za ky = 0 do h
24         za kx = 0 do w
25           prevErr(i, y, x) += out(i,y,x) * K(i,j,y,x) *
                F_deriv(prevOut(i, y+ky, x+kx))
26
27
28 # azuriraj tezine
29 za svaku mapu prethodnog sloja i
30   za svaku mapu trenutnog sloja j
31     K(i,j) -= eta * prevErr(i,j)
```

4.4. Specifičnosti slojeva sažimanja značajki

Kod unazadne propagacije slojeva sažimanja (engl. *pooling layers*) potrebno je propagirati grešku manje mape trenutnog sloja u veću mapu prethodnog sloja. Veza je jedan-na-jedan te nema učenja parametra. Slojevi sažimanja tokom unazadne propagacije isključivo propagiraju grešku unazad te ne čine ništa drugo. Način propagacije naravno ovisi o operaciji sažimanja koja se koristila prilikom unaprijedne propagacije.

4.4.1. Sažimanje maksimumom

Tokom unaprijedne propagacije Max-pooling operacije propagiraju samo maksimalne vrijednosti prethodne mape značajki a lokacije pojedinih maksimuma se zapisuju da bi se greška mogla propagirati unazad.

Pseudokod 4.3: Širenje greške unazad kod Max-pooling slojeva

```
1
2 # broj mapa susjednih slojeva je isti
3 # kod slojeva izvlacenja znacajki
4 za svaki par mapi
5   za y = 1 do visine mape prethodnog sloja
6     za x = 1 do sirine mape prethodnog sloja
7       prevErr(y, x) = currErr( L(x,y, 0), L(x,y,1) )
```

Za primjer iz poglavlja 3.3.2 gdje je matrica mapiranja $\mathbf{L} = \begin{bmatrix} (0,1) & (0,1) \\ (0,1) & (1,1) \end{bmatrix}$ neka je greška trenutnog sloja $\mathbf{E}' = \begin{bmatrix} -0.21 & 0.18 \\ 0.72 & -0.30 \end{bmatrix}$. Unazadnom propagacijom dobiva se greška prethodnog sloja koja iznosi: $\mathbf{E} = \begin{bmatrix} 0 & -0.21 & 0 & 0.18 \\ 0 & 0 & 0 & 0 \\ 0 & 0.72 & 0 & 0 \\ 0 & 0 & 0 & -0.30 \end{bmatrix}$

4.4.2. Sažimanje aritmetičkom sredinom

Kod ovakvih slojeva sažimanja svaki neuron mape sudjeluje u definiranju mape idućeg sloja. Iz tog razloga se i greška propagira unatrag svim slojevima. Svi neuroni unutar grupe primaju jednaku grešku. Iako se ovakvo širenje greške unazad može jednostavno programski implementirati petljama, zanimljivi je matrični zapis. Neka je greška trenutnog sloja i dalje $\mathbf{E}' = \begin{bmatrix} -0.21 & 0.18 \\ 0.72 & -0.30 \end{bmatrix}$ te neka je I jedinična matrica

veličine $S_w \times S_h$, gdje su S_w i S_h faktori skaliranja sloja. Tada je greška prethodnog sloja definirana Kroneckerovim produktom jedinične matrice i matrice trenutnog sloja:

$$\mathbf{E} = \mathbf{E}' \otimes \mathbf{I} \tag{4.10}$$

Za danu matricu greške trenutnog sloja $\mathbf{E}'$, i faktora skaliranja $S_h = S_w = 2$ je greška prethodnog sloja $\mathbf{E} = \begin{bmatrix} -0.21 & 0.18 \\ 0.72 & -0.30 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} -0.21 & -0.21 & 0.18 & 0.18 \\ -0.21 & -0.21 & 0.18 & 0.18 \\ 0.72 & 0.72 & -0.30 & -0.30 \\ 0.72 & 0.72 & -0.30 & -0.30 \end{bmatrix}$

4.5. Varijante gradijentnog spusta

Postoje dvije glavne varijante gradijentnog spusta u ovisnosti o trenutku ažuriranja težina:

- **Standardni (grupni) gradijentni spust** - (engl. *batch gradient descent*) predstavlja varijantu u kojoj se najprije izračunaju potrebne promjene težina Δw koje nastaju svim uzorcima u skupu za učenje ili grupi (engl. *batch*), nakon čega se vrši ažuriranje težina. Takav je pristup često brži u konvergenciji i stabilniji ali je podložniji zapinjanju u lokalnim optimumima te uz njega nije moguće koristiti metode drugog reda prilikom učenja konvolucijskih neuronskih mreža [14].
- **Stohastički gradijentni spust** - (engl. *online/stochastic gradient descent*) je varijanta gradijentnog spusta kod koje se nakon svakog uzorka ažuriraju težine. Ovakva varijanta više oscilira te je zbog toga otpornija na zapinjanje u lokalne optimume. Također, kod učenja konvolucijskih neuronskih mreža ovom varijantom, moguće je ubrzati postupak metodama drugog reda [14].

4.6. Poboljšavanje konvergencije učenja mreže

Algoritam backpropagation je iterativna metoda optimiranja funkcije cilja koja ovisno o složenosti modela (složenosti funkcije cilja) i složenosti podataka može konvergirati poprilično sporo. S obzirom da su konvolucijske neuronske mreže klasifikatori velike složenosti, njihovo je učenje poprilično sporo (s obzirom na jednostavnije ne-duboke arhitekture). Zbog toga se često u praksi koriste dodatne metode koje ubrzavaju konvergenciju, od kojih su dvije navedene u nastavku.

4.6.1. Dodavanje inercije

Učenje gradijentnim spustom je često sporo (pogotovo kod višemodalnih funkcija) te je podložno zapinjanju u lokalnim optimumima. Metoda dodavanja inercije (engl. *momentum*) uzima u obzir prethodnu brzinu (definiranu kao pomak u prethodnom koraku Δw) te ju dodaje trenutnoj brzini (trenutni pomak Δw gradijentnog spusta) smanjenu za amortizacijsku konstantu α. Ubrzanje gradijentnog spusta postiže se kroz dva svojstva: povećanje brzine i smanjivanje oscilacija kod veoma neravnih višemodalnih funkcija cilja.

$$\Delta w(t) = \alpha \Delta w(t-1) - \eta \frac{\partial E}{\partial w}(t) \tag{4.11}$$

Prilikom određivanja amortizacijske stope α potrebno je uzeti u obzir da su na početku gradijenti poprilično veliki dok su u kasnijim iteracijama oni sve manji. Kao okvirno pravilo, obično se na početku koristi amortizacijska stopa $\alpha \approx 0.5$ a kasnije $\alpha = 0.9$ do $\alpha = 0.99$. η i dalje predstavlja, kao i kod klasičnog gradijentnog spusta, stopu učenja (koja utječe na prve derivacije) te se odabire isprobavanjem različitih veličina (npr. 0.01, 0.01, 0.1, itd.) i traženjem vrijednosti kod kojih je učenje dovoljno brzo a da ne dolazi do oscilacija (i stoga nedostatka konvergencije).

4.6.2. Metoda drugog reda

Metodu drugog reda moguće je koristiti isključivo stohastičkim gradijentnim spustom [14]. Ideja iza te metode je dodatno skaliranje globalne stope učenja η za svaku pojedinu težinu i mape k (s obzirom na globalnu stopu učenja η na temelju informacija o zakrivljenosti funkcije cilja u trenutnoj točki). Svaka težina tada ima svoju vlastitu stopu učenja η_{ki}. Neka je $o_i = f(y_i)$ te neka je μ maleni broj (npr. 0.1) koji služi izbjegavanju da η_{ki} počne težiti u beskonačnost. Tada je metoda drugog reda definirana izrazom (4.12).

$$\eta_{ki} = \frac{\eta}{\left(\frac{\partial^2 E}{\partial w_{ki}^2}\right) + \mu} \tag{4.12}$$

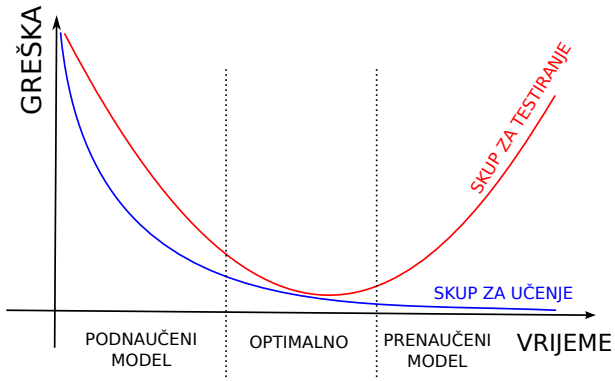
Druge se derivacije aproksimiranju Gauss-Newtonovom metodom [14] izrazima 4.13 i 4.14:

$$\frac{\partial^2 E}{\partial y_k^2} = \frac{\partial^2 E}{\partial o_k^2} \left(f'(y_k)\right)^2 \tag{4.13}$$

$$\frac{\partial^2 E}{\partial w_{ki}^2} = \sum_k \frac{\partial^2 E}{\partial y_k^2} x_i^2 \tag{4.14}$$

4.7. Poboljšavanje generalizacije

Učenje klasifikator sve više prilagođava skupu za učenje. Ukoliko je klasifikator dovoljno ekspresivan moguće je da će se s vremenom previše prilagoditi skupu za učenje pa će loše klasificirati uzorke koje nije vidio (loša generalizacija), što je pokazano na slici 4.3. Tada kažemo da je model prenaučen (engl. *overfitting*). Prenaučenost se može spriječiti na više načina: normalizacijama, povećanjem varijacije uzorka za učenje, modeliranjem složenosti klasifikatora, kombiniranjem više klasifikatora (engl. *averaging*) i drugih metoda.



Slika 4.3: Greške modela

4.7.1. Slučajne transformacije

Generalizaciju je općenito moguće poboljšati povećanjem skupa za učenjem (pa samim time i varijacije skupa). Sakupljanje dodatnih uzoraka (posebice ukoliko se radi o redu veličine više uzoraka) može biti skupo, kako vremenski tako i novčano. Moguće je koristiti slučajne transformacije (najčešće **translacija, rotacija i skaliranje**) nad postojećim skupovima kako bi se dodatno povećao skup te poboljšala generalizacija klasifikatora, bez dodatnih troškova [3]. U ovom radu korištene su sljedeće transformacije:

– **Slučajni pomak rubova i uvećanje** - uzorkovan iz $\mathcal{N}(0, 1)$ pomiče koordinate lijevog gornjeg vrha i donje-desnog vrha pravokutnika koji upisuje uzorak na slici, tako utječući i na skaliranje i translaciju izvornog uzorka.

– **Slučajna rotacija** - uzorkovana iz $\mathcal{N}(0, 5^\circ)$ odgovara približno kutu prometnih znakova koji su uvijek vertikalno postavljeni.

Slučajne transformacije je moguće izvoditi na dva načina:

– **Tijekom svake iteracije** (engl. *online*) - izvodi se dodavanjem dodatnog sloja na ulazu koji transformira ulaz te ga dalje prosljeđuje na prvi sloj konvolucijske neuronske mreže. Ovakav pristup je pogodan iz činjenice što može generirati proizvoljno mnogo uzoraka (iako će se, ovisno o distribucijama parametara transformacija, ponekada pojavljivati isti uzorci ukoliko ih se generira u puno većem broju nego što postoji izvornih uzoraka) te je neovisan o skupu (dio je klasifikatora). Mogući problem kod ovakvog pristupa je potreba za spremanjem veće slike od veličine izvornog uzorka. To je potrebno da bi se prilikom rotacije i translacije i dalje sačuvala originalna pozadina (izrezivanje uzorka na način da se pojavljuju nedefinirani dijelovi bi jako loše djelovalo na klasifikator odnosno na njegovo generalizacijsko svojstvo). Ilustracija problema spremanja slike prilikom korištenja slučajnih transformacija tokom učenja dana je u slici 4.4.

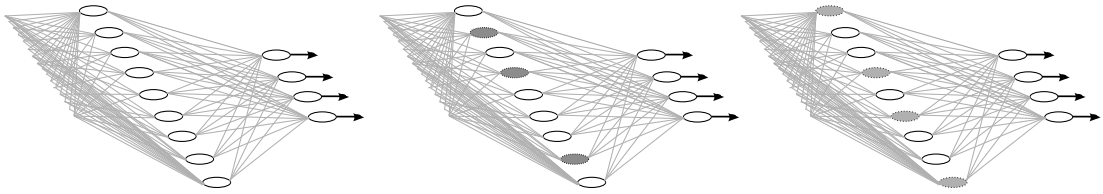
– **Preprocesiranjem uzoraka** - u pred-obradi uzoraka za učenje na svaki se uzorak apliciraju slučajne transformacije te se tako tvore drugi uzorci iste klase. Ovakvim pristupom nema usporavanja učenja (nema dodatnog procesiranja u svakoj iteraciji) te nema dodatne složenosti kod spremanja podataka (sprema se uvijek isti okvir u kojem se nalazi već transformirani uzorak). Ukoliko se znatno poveća broj uzoraka (s obzirom na početni broj uzoraka) ovom metodom, postupak teži identičnom ponašanju kao i "*online*" procesiranje. U ovom je radu korištena metoda preprocesiranja uzoraka koja povećava broj uzoraka za više od red veličine (detaljnije u poglavlju o ispitnim skupovima), na sličan način kao što je napravljeno u [1].

4.7.2. Slučajno izostavljanje neurona prilikom učenja

Poznati i često korišteni način poboljšavanja generalizacije je kombiniranje više modela (klasifikatora) te njihovo usrednjavanje. Kod konvolucijskih neuronskih mreža (kao i kod ostalih dubokih neuronskih mreža) treniranje više klasifikatora može biti veoma skupo po pitanju računalnih resursa i vremenu. Slučajno izostavljanje neurona prilikom učenja (engl. *dropout*) je metoda koja prividno stvara jako veliki broj klasifikatora, iako se i dalje radi samo sa jednim [7]. Ideja je jednostavna: u svakoj se iteraciji slučajno "isključuju" pojedini neuroni (ne uzimaju se u obzir prilikom propagacije

Slika 4.4: Slučajne transformacije: a) Spremanje većeg dijela originalne slike (ispravno) b) Isključivo spremanje područja detekcije (pogrešno)

i unazadne propagacije). Za jednostavan primjer perceptrona sa jednim skrivenim slojem od N neurona to stvara ukupno 2^N virtualnih arhitektura. Kod konvolucijskih neuronskih mreža sa nekoliko slojeva i po nekoliko desetaka mapi u svakom sloju je broj takvih virtualnih arhitektura višestruko veći. Međutim i dalje se u implementaciji trenira samo jedna arhitektura, zbog čega nema dodatnih zahtjeva po pitanju računalnih resursa. Naravno, koristeći metodu slučajnog izostavljanja neurona prilikom učenja mreža može značajno sporije konvergirati, ali zato puno bolje generalizirati. Metoda slučajnog izostavljanja neurona prilikom učenja se preporuča ukoliko mreža ima problema s prenaučenosti [7] te ju nije potrebno koristiti ukoliko zbog drugih metoda ili prirode skupa za učenja to nije slučaj.



Slika 4.5: Slučajno izostavljanje neurona prilikom učenja: različite arhitekture sa dijeljenim težinama i privremeno deaktiviranim neuronima (sivo)

4.8. Mogući problemi

Duboke neuronske mreže mogu biti teške za trenirati. One, za razliku od primjerice višeslojnih perceptrona, sadrže puno više hiperparametara koje treba uskladiti da bi se postigla konvergencija. Štoviše, učenje takvih mreža je sporo (zahtjevno po pitanju računalnih resursa) pa optimiranje hiperparametara može biti veoma vremenski zahtjevno za arhitekta mreže.

4.8.1. Odabir strukturalnih hiperparametara mreže

Kod odabira broja mapi u svakom sloju, treba uzeti u obzir da je računanje kod svake mape puno skuplje (kako tokom unaprijedne tako i tokom unazadne propagacije). Idealno treba naći kompromis između dovoljno složene arhitekture za navedeni problem te realnog vremena učenja klasifikatora. (Učestalo je da se *state-of-the-art* rezultati dobivaju složenijim arhitekturama koje se treniraju više od tjedan dana na grafičkim procesorima [3].) Odabir broja slojeva je sekundaran. Umjesto toga, preporuča se da se vrši odabir veličine jezgara (engl. *kernel*) i veličina prozora sažimanja značajki u pojedinim slojevima a da dubina arhitekture bude rezultat tog odabira.

Odabir veličine jezgara u pojedinim konvolucijskim slojevima je jedan od najbitnijih stavki. Veličina jezgre treba biti takva da naučene apstrakcije budu prikladne skale (engl. *scale*) s obzirom na prirodu skupa uzoraka koji se pokušava klasificirati. Primjerice, za skup MNIST odabrali smo jezgre veličine 5×5 u prvim slojevima (radi se o tankim crtama rukom pisanih brojki) dok smo za skup FER-MASTIF TS2010 odabrali veličinu jezgara u prvom sloju od 7×7. Ukoliko su jezgre premalih dimenzija, moguće je da će se mreža "fokusirati" na značajke premale granulacije koje nisu relevantne za klasifikaciju. S druge strane, ukoliko su jezgre prevelike, vjerojatno je da će kasniji slojevi tek izvući relevantne značajke te da bi mreža mogla raditi i sa slojem manje (što je brže, s obzirom na manju složenost). Dublji konvolucijski slojevi tipično sadrže jezgre malo manjih dimenzija od prvog konvolucijskog sloja. Ipak, treba pripaziti da ne smanjimo dimenzije pre naglo da ne bi došlo do prevelikog gubitka informacije.

Slojevi sažimanja značajki su puno jednostavniji te se uglavnom svode na smanjivanje dimenzionalnosti prozorom veličine 2×2 ili, za jako velike mape, prozorom veličine 4×4 .

4.8.2. Testiranje gradijenata i konvergencije

Konvolucijske neuronske mreže su dosta složene arhitekture te je prilikom njihove implementacije (osim ukoliko se ne koristi već provjerena biblioteka) lako moguće uvesti pogrešku u kodu. Stoga se preporuča da se kod svake male komponente (neuron višeslojnog perceptrona, konvolucijska mapa, pojedini slojevi i dr.) ručno testira na jednostavnim izgeneriranim primjerima. Potrebno je testirati unaprijednu propagaciju, unazadnu propagaciju, provijeriti da li gradijenti stvarno pokazuju u smjeru minimizacije pogreške te da li se kroz iteriranje greška smanjuje (da li jednostavni ispitni sustav konvergira).

4.8.3. Limitiranje vrijednosti

Limitiranje vrijednosti te njihovo stabilno držanje unutar područja rada aktivacijske funkcije ključno je za učenje složenih arhitektura poput konvolucijskih neuronskih mreža. Za to su bitne dvije stvari:

- **Ispravna slučajna inicijalizacija težina** - omogućava nepristranost u prvim iteracijama učenja te osigurava da se težine mogu pomicati u oba smjera aktivacijske funkcije, u ovisnosti o skupu za učenje. Loša inicijalizacija težina može usporavati učenje tako da uvodi prevelike udaljenosti od početne vrijednosti do trenutka kada ulazi u aktivacijske funkcije prijeđu na ispravnu stranu, za što može biti potrebno puno koraka gradijentnog spusta.
- **Nulta sredina i jedinična varijanca skupa za učenje** - (engl. *zero mean, unit variance*) osigurava da su podatci prikladni za što brže učenje te da se nalaze što je moguće bliže središtu aktivacijske funkcije (tako da nije potreban veliki broj koraka gradijentnog spusta dok se ulaz u aktivacijsku funkciju ne prenese na ispravnu stranu). Ovaj postupak je detaljnije opisan u poglavlju predobrade skupa za učenje FER-MASTIF TS2010.

4.8.4. Stopa učenja

Uobičajeno se prilikom odabira stope učenja najprije testiraju razni redovi veličina stope učenja poput $\eta = 0.001, 0.01, 0.1, 1, ...$ Ukoliko je stopa učenja premala, sustav će sporo konvergirati dok ukoliko je stopa učenja prevelika sustav će oscilirati (preskakati s obiju strana optimuma). Prilikom odabira stope učenja konvolucijskih neuronskih mreža, važno je imati na umu da se u konvolucijskim slojevima nalaze više neurona koji dijele iste težine, zbog čega je potrebna puno manja stopa učenja da dođe do neželjenih oscilacija (iako se provjerom predznaka višeslojnog perceptrona na vrhu konvolucijske neuronske mreže takva oscilacija ne bi uvidjela).

4.8.5. Lokalni optimumi

Ovisno o ulaznim podatcima te arhitekturi mreže, može se desiti da mreža zapinje za lokalne optimume te ne postigne minimalnu grešku koju bi mogla postići bez tog problema. To je moguće riješiti na nekoliko različitih načina:

- **Uvođenje raspršene povezanosti** - (engl. *sparse connectivity*) [11] forsira da svaka mapa nauči što različitije značajke čime se mogu povećati performanse klasifikatora (vidi poglavlje 3.4) .

– **Metoda slučajnog izostavljanja neurona prilikom učenja** (engl. *dropout*) - koja virtualnim povećanjem broja arhitektura olakšava prelazak preko lokalnih optimuma. Ukoliko jedna arhitektura zapinje za lokalni optimum, malo je vjerojatno da će i druge zapeti za taj isti optimum. S obzirom na veliki broj virtualnih arhitektura, kod metode slučajnog izostavljanja neurona prilikom učenja, gotovo je nemoguće da cjelokupni sustav klasifikatora zapne na lokalnom minimumu. (Vidi poglavlje 4.7.2)

– **Slučajne transformacije skupa za učenje** - metode prikazane u poglavlju 4.7.1 jako dobro sprečavaju zapinjanje u lokalnim optimumima te je preporučeno koristiti ih.

– **Slučajna perturbacija težina** - ukoliko sustav i dalje zapinje u lokalnim optimumima, moguće je u svakoj iteraciji slučajno odabrani skup težina perturbirati dodavanjem malog pomaka uzorkovanim iz Gaussove distribucije (npr. $\Delta w \leftarrow \Delta w + \mathcal{N}(0, 0.001)$). Takve slučajne perturbacije omogućavaju da mreža prođe preko lokalnog optimuma te nastavlja s optimizacijom.

5. Programska izvedba

Za programsku izvedbu odabran je programski jezik *Python* uz korištenje numeričke *NumPy*. Izbor se temeljio na želji za što lakšom i bržom izradom prototipova mreža, s fokusom na brzinu u drugom planu. Također, cilj je bio stvoriti programsku okolinu koja omogućava brzo stvaranje različitih tipova mreža, te je stoga većina toga implementirano kao biblioteka.

Uvedena su dva pojma:

- **sloj** - predstavlja jedan sloj u dubokoj neuronskoj mreži koji može biti:
  - dvodimenzionalni ili jednodimenzionalni ulazni sloj.
  - dvodimenzionalna mapa (koja nastaje konvolucijom ili operacijama sažimanja značajki). Vrijedi napomenuti da je takva mapa implementirana tako da ukoliko je ona veličine 1×1, tada se ona može nadovezati na skriveni višeslojnog perceptrona. Ukoliko je mapa veća od 1×1 nije ju moguće povezati na jednodimenzionalne slojeve višeslojnog perceptrona te je potrebno provesti neku operaciju (konvolucija ili sažimanje) kako bi se smanjila njezina dimenzionalnost.
  - jednodimenzionalni skriveni sloj.
  - jednodimenzionalni izlazni sloj.
- **veza** - predstavlja skup veza (i operacija nad njima) između dva sloja. Veze predstavljaju težine i jezgre, izračunavaju odziv te šire grešku vrijednosti unazad. Veza može biti:
  - konvolucijska veza.
  - veza operacije sažimanja.
  - veza klasičnog višeslojnog perceptrona, često pod nazivom (engl. *fully connected layer*) s obzirom da nema dijeljenja težina u takvim slojevima te je povezanost uvijek "svaki neuron sa svakim".

Kod definiranja slojeva definiraju se samo njihove veličine i broj mapi ili neurona unutar njih. Kod definiranja veza definiraju se svi ostali parametri poput broja jezgara,

matrice povezanosti, aktivacijskih funkcija i dr. Sustav pri inicijalizaciji mreže provjerava da li dani parametri mreže definiraju valjanu mrežu, npr. da li broj jezgara i broj slojeva odgovara danim dimenzijama matrice povezanosti.

Odabirom aktivacijske funkcije (ukoliko se ništa ne odabere sustav pretpostavlja sigmoidalnu funkciju hiperbolnog tangensa) sustav automatski definira pripadnu derivaciju te intervale slučajnog inicijaliziranja težina.

5.1. Struktura

Struktura programske podrške ovog rada podijeljena je u sljedeće direktorije:

- ***code*** - svi kodovi programa.
- ***data***- podatci o skupovima MNIST i FER-MASTIF TS2010 u formatu *pkl* te uzorci ispitnih skupova koji su krivo klasificirani u odgovarajućim poddirektorijima.
- ***models*** - naučeni parametri mreža za klasifikaciju MNIST i FER-MASTIF TS2010 skupova u pkl formatu.
- ***remapping*** - pomoćne datoteke koje su se koristile prilikom izrade novih oznaka FER-MASTIF TS2010 skupa [9].

Datoteke programske podrške (iz direktorija ***code*** su navedene u nastavku, grupirane po funkciji:

Biblioteke:

- ***data.py*** - funkcije za učitavanje i vizualizaciju skupova te funkcije za pretvorbu formata MNIST IDX formata u format *pkl*.
- ***utils.py*** - klase koje definiraju različite aktivacijske funkcije, njihove derivacije te pripadajuće intervale uzorkovanja slučajnih početnih težina.
- ***mlp.py*** - klase višestrukog perceptrona.
- ***featuremaps.py*** - klase mape značajki (koje se koriste kod konvolucijskih slojeva i slojeva sažimanja).
- ***conv.py*** - klase za slojeve konvolucije.
- ***pooling.py*** - klase za slojeve sažimanja.

Aplikacije za učenje:

- ***learnMNIST.py*** - uči konvolucijsku neuronsku mrežu na skupu za učenje skupa MNIST te sprema naučeni model u direktoriju models.

– ***learnMASTIF_TS2010.py*** - uči konvolucijsku neuronsku mrežu na skupu za učenje skupa FER-MASTIF TS2010 te sprema naučeni model u direktoriju *models*.

Aplikacije za testiranje:

– ***testMNIST.py*** - učitava model iz direktorija *models* te ga evaluira na ispitnom skupu skupa MNIST. Generira matricu zabune te zapisuje pogrešno klasificirane uzorke u direktoriji data/misclassifiedMNIST.

– ***testMASTIF_TS2010.py*** - učitava model iz direktorija models te ga evaluira na ispitnom skupu skupa FER-MASTIF TS2010. Generira matricu zabune te zapisuje pogrešno klasificirane uzorke u direktorij *data/misclassifiedMASTIF*.

Pomoćne aplikacije:

– ***MASTIFReLabeler.py*** - učitava izvorne anotacije i dodatne anotacije iz direktorija remapping te stvara novu (unificiranu) anotacijsku datoteku *../data/labels.txt*.

– ***processMASTIF.py*** - učitava izvorne slike skupa FER-MASTIF TS2010, novo kreiranu anotacijsku datoteku *../data/labels.txt,* generira statistiku skupa, izvlači znakove čije se klase pojavljuju dovoljno učestalo, generira i proširuje skup za učenje i skup za testiranje te ih zapisuje u direktoriju *data*.

– ***visualizeKernelsMNIST.py*** - generira vizualizaciju jezgara prvog i drugog konvolucijskog sloja konvolucijske neuronske mreže za skup MNIST.

– ***visualizeKernelsMASTIF.py*** - generira vizualizaciju jezgara prvog i drugog konvolucijskog sloja konvolucijske neuronske mreže za skup FER-MASTIF TS2010.

5.2. Primjer izrade duboke neuronske mreže

U nastavku je prikazana izrada konvolucijske neuronske mreže za klasifikaciju uzorka iz skupa MNIST korištenjem funkcija i biblioteka razvijenih u sklopu ovog rada. Dvodimenzionalni sloj (sloj koji sadrži mape značajki) definiran je klasom *LayerFM*, dok je jednodimenzionalni sloj neurona definiran klasom *Layer1D*. Prototipovi konstruktora dani su u nastavku:

Programski kod 5.1: Prototip konstruktora klase LayerFM

```
1 class layerFM:
2   def __init__(self, n, width, height, isInput = False,
      isOutput = False):
```

Programski kod 5.2: Prototip konstruktora klase Layer1D

```
1 class layer1D:
2   def __init__(self, n, isInput = False, isOutput = False,
      hasBias = None):
```

Svaki je sloj definiran sljedećim parametrima:

- ***n*** - broj neurona/mapi u sloju.
- ***isInput*** - oznaka da li se radi o ulaznom sloju.
- ***isOutput*** - oznaka da li se radi o izlaznom sloju.

Dodatno, klasa *LayerFM* sadrži definiciju veličine mape značajki sloja:

- ***width*** - širina mape značajki sloja.
- ***height*** - visina mape značajki sloja.

Nadalje, svaki sloj implementira metodu *setFM(data)* kojom se postavlja vrijednost svih mapi značajki unutar sloja. Naravno, to je moguće isključivo za ulazni sloj, dok se kod ostalih slojeva to automatski izračunava prilikom propagacije unaprijed.

Definicija slojeva za predloženu arhitekturu konvolucijske neuronske mreže je tada:

Programski kod 5.3: Definicija slojeva konvolucijske neuronske mreže

```
1  inputLayer0 = layerFM(1, 32, 32, isInput = True)
2  convLayer1 = layerFM(6, 28, 28)
3  poolLayer2 = layerFM(6, 14, 14)
4  convLayer3 = layerFM(16, 10, 10)
5  poolLayer4 = layerFM(16, 5, 5)
6  convLayer5 = layerFM(100, 1, 1)
7  hiddenLayer6 = layer1D(80)
8  outputLayer7 = layer1D(10, isOutput = True)
```

Nakon definiranja slojeva, potrebno je definirati veze između njih. Za to su predviđene tri klase:

- ***convolutionalConnection*** - koja spaja dva (dvodimenzionalna) sloja operacijom konvolucije.
- ***poolingConnection*** - koja spaja dva (dvodimenzionalna) sloja operacijom sažimanja značajki.
- ***fullConnection*** - koja spaja jednodimenzionalne slojeve (klasični višeslojni perceptron).

Prototipovi konstruktora tih klasa dani su u nastavku:

Programski kod 5.4: Prototip konstruktora klase convolutionalConnection

```
1 class convolutionalConnection:
2   def __init__(self, prevLayer, currLayer, connectionMatrix,
      kernelWidth, kernelHeight, stepX, stepY, useLogistic =
      False):
```

Programski kod 5.5: Prototip konstruktora klase poolingConnection

```
1 class poolingConnection:
2   def __init__(self, prevLayer, currLayer, poolingStepX,
      poolingStepY):
```

Programski kod 5.6: Prototip konstruktora klase fullConnection

```
1 class fullConnection:
2   def __init__(self, prevLayer, currLayer, useLogistic =
      False):
```

Zajedničko svim konstruktorima su parametri koji ih povezuju sa slojevima:

- ***prevLayer*** - prethodni sloj. Iz njega se čitaju vrijednosti prilikom propagacije unaprijed, te se u njega upisuju vrijednosti grešaka prilikom propagacije unazad.
- ***currLayer*** - trenutni sloj. U njega se upisuju izračunate vrijednosti prilikom propagacije unaprijed, te se iz njega iščitavaju greške prilikom propagacije unazad. Iznimka je slučaj kada je trenutni sloj ujedno i izlazni sloj: tada se njegova greška računa na temelju trenutne vrijednosti (izlaza) i željene vrijednosti.

Konvolucijske veze imaju sljedeće dodatne parametre:

- ***connectionMatrix*** - matrica povezanosti koja označava koja je mapa značajki prethodnog sloja povezana s određenom mapom trenutnog sloja: 1 označava

povezanost a 0 označava da mape nisu povezane.

- ***kernelWidth*** - označava širinu jezgre kojom se provodi konvolucija.
- ***kernelHeight*** -označava visinu jezgre kojom se provodi konvolucija.
- ***stepX*** - označava horizontalni pomak jezgre prilikom provođenja konvolucije (obično 1).
- ***stepY*** - označava vertikalni pomak jezgre prilikom provođenja konvolucije (obično 1).
- ***useLogistic*** - označava da li se koristi logistička sigmoidalna funkcija ili sigmoidalna funkcija hiperbolnog tangensa (predefinirani odabir).

Nadalje, veze slojeva sažimanja sadrže dva dodatna parametra:

- ***poolingStepX*** - širina prozora unutar kojeg se sažimaju značajke i predstavljaju jednom značajkom u idućem sloju. Nakon toga se za toliku vrijednost prozor horizontalno pomiče te se određuje iduća značajka.
- ***poolingStepY*** - isto kao i *poolingStepX*, samo za vertikalnu komponentu.

Slojevi veza definiraju dvije metode:

- ***propagate()*** - obavlja propagaciju unaprijed između dva sloja.
- ***bprop(****eta, [label])* - obavlja propagaciju unazad između dva sloja uz stopu učenja *eta*. Nadalje, ukoliko je *currentLayer* izlazni sloj, potrebno je predati i željene izlaze za trenutni ulaz.

Definiranje veza između slojeva obavlja se stoga na sljedeći način:

Programski kod 5.7: Definiranje veza između slojeva

```
1  convolution01 = convolutionalConnection(inputLayer0,
       convLayer1, np.ones([1, 6]), 5, 5, 1, 1)
2
3  pooling12  = poolingConnection(convLayer1, poolLayer2, 2, 2)
4
5  convolution23 = convolutionalConnection(poolLayer2,
       convLayer3, np.ones([6, 16]), 5, 5, 1, 1)
6
7  pooling34  = poolingConnection(convLayer3, poolLayer4, 2, 2)
8
9  convolution45 = convolutionalConnection(poolLayer4,
       convLayer5, np.ones([16, 100]), 5, 5, 1, 1)
10
11 full56     = fullConnection(convLayer5, hiddenLayer6)
12
13 full67     = fullConnection(hiddenLayer6, outputLayer7)
```

Na kraju, učenje se provodi postavljanjem uzorka na ulaz te pozivanjem *propagate()* i *bprop()* metoda, za svaki pojedini sloj:

Programski kod 5.8: Učenje mreže

```
for it in range(maxIter):

  np.random.shuffle(seq) # slucajna permutacija skupa

  for i in range(len(images)):

    # postavljanje ulaza
    inputLayer0.set_FM(np.array([images[seq[i]]]))

    # propagacija unaprijed
    convolution01.propagate()
    pooling12.propagate()
    convolution23.propagate()
    pooling34.propagate()
    convolution45.propagate()
    full56.propagate()
    y = full67.propagate()

    # propagacija unazad (ucenje)
    full67.bprop(eta, labels[seq[i]])
    full56.bprop(eta)
    convolution45.bprop(eta)
    pooling34.bprop()
    convolution23.bprop(eta)
    pooling12.bprop()
    convolution01.bprop(eta)
```

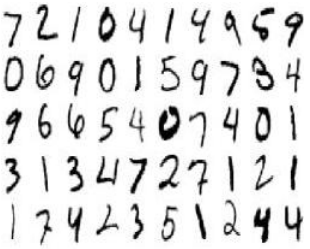
Na kraju, ukoliko se težine trebaju spremiti ili učitati, klase *convolutionalConnection*, *poolingConnection* i *fullConnection* nude pristup jezgrama (svojstvo k) i težinama (svojstvo w) kojim se to može učiniti.

6. Ispitni skupovi

6.1. MNIST

Skup MNIST (engl. *Mixed National Institute of Standards and Technology*) [12] sadrži 10 klasa rukom pisanih brojeva (od nule do devet). Nekoliko takvih znakova prikazano je na slici 6.1. 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 - koji sadrži 60 000 znakova te
- skup za ispitivanje - koji sadrži 10 000 znakova.



Slika 6.1: Nekoliko slučajno odabranih uzoraka iz skupa MNIST

6.1.1. Izvorni format zapisa skupa - IDX

Postoje dva formata datoteka za skup MNIST, oba pod oznakom IDX (engl. *Indexed*):

- **IDX3-UBYTE** format za zapis uzoraka oblika>
  - 32 bita: broj 2051 - oznaka formata.
  - 32 bita: broj uzoraka N.
  - 32 bita: broj redaka u svakom uzorku n_{rows}.
  - 32 bita: broj stupaca u svakom uzorku n_{cols}.

- $8Nn_{rows}n_{cols}$ bita: redom posloženi uzorci, redak po redak (jedna točka je 8 bita).

– **IDX1-UBYTE** format za zapis oznaka uzoraka (pripadnih klasa)

- 32 bita: broj 2049- oznaka formata.
- 32 bita: broj uzoraka N.
- $8N$ bita: redom posložene oznake (jedna oznaka je 8 bita).

6.1.2. Vlastiti format zapisa skupa

Radi praktičnosti i brzine učitavanja, izvorni je skup pretvoren u vlastiti format. Format sačinjavaju kompresirani (ZIP) Python nizovi. Prvi niz sadrži uzorke (gdje je svaki uzorak jedna matrica 28×28) dok drugi niz sadrži klasu pripadnosti. Pripadnost je označena nizom dužine 10, gdje je svaki element jednak nuli, osim onoga koji predstavlja ispravnu klasu pripadnosti. Primjerice, uzorak broja "3" bi bio označen kao $\{0, 0, 0, 1, 0, 0, 0, 0, 0, 0\}$. Učitavanje skupa se jednostavno obavlja ne sljedeći način:

Programski kod 6.1: Učitavanje skupa

```
1 def loadData(fName):
2   f = gzip.open(fName)
3   data = cPickle.load(f)
4   f.close()
5   return data
6
7 images, labels = loadData("../data/MNISTtrain.pkl")
```

6.1.3. Nulta sredina i jedinična varijanca

Za uspješno učenje mreže, važno je da je aritmetička sredina skupa približno nula (da bi učenje težina moglo "lagano" 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. Također, svakom uzorku je dodan rub širine 2 kako bi se konvolucija mogla normalno odvijati i za rubne značajke. Nakon toga su uzorci spremljeni u prije opisanom vlastitom formatu.

6.2. FER-MASTIF TS2010

Skup FER-MASTIF TS2010, izrađen u sklopu istoimenog projekta (engl. *Mapping and Assessing the State of Traffic InFrastructure*) [19] [20] sadrži 3889 slika koje su izvučene iz video snimki kamere montirane na automobilu uključenog u promet, a u svakoj slici su označeni prometni znakovi (hrvatskog prometnog sustava) uz ukupno 87 različitih klasa prometnih znakova. Jedna takva slika prikazana je na slici 6.2.

Slika 6.2: Slika iz FER-MASTIF TS2010 skupa

6.2.1. Izvorni anotacijski format

Uzorci su izvorno anotirani na sljedeći način:

- $files =$ Popis datoteka, razdvojenih točkom-zarez.
- $[datoteka] : znak1_{oznaka}@(x = x_{koordinata}, y = y_{koordinata}, w = sirina, h = visina)\&...\&znakN_{oznaka}@(x = x_{koordinata}, y = y_{koordinata}, w = sirina, h = visina)$.

Izvadak 6.2: Isječak izvorne anotacijske datoteke

```
files=_0000.bmp;...;A16_0009.bmp;A16_0010.bmp;A16_0011.bmp
[B32_0168.bmp]:B32@(x=479,y=259,w=21,h=23)&B31@(x=480,y=283,w=21,h=20)
[C44_0162.bmp]:A04@(x=554,y=231,w=43,h=39)&C44@(x=561,y=272,w=33,h=30)
[C80_0102.bmp]:C79@(x=572,y=218,w=59,h=32)&C80@(x=572,y=251,w=56,h=30)
[B48_0019.bmp]:C86@(x=659,y=182,w=58,h=62)&B48@(x=658,y=244,w=58,h=60)
[C80_0109.bmp]:C79@(x=568,y=223,w=52,h=29)&C80@(x=571,y=255,w=50,h=23)
[A33_0015.bmp]:A33@(x=623,y=171,w=95,h=92)
```

6.2.2. Predobrada i podjela skupa

Skup je najprije podijeljen na temelju oznake okvira video snimke (engl. *frame number*) kako se isti fizički znak (koji je najprije snimljen iz daljine, a zatim izbliza) ne bi više puta našao u isti skup (za učenje ili testiranje) [9].

Predložena je i stvorena nova anotacijska datoteka sljedećeg formata: Svaki uzorak je definiran u jednom redu, a sam zapis se sastoji od sljedećih kolona odvojenim tabulatorom:

- naziv datoteke u kojoj je spremljena čitava slika
- oznaka prometnog znaka
- Test/Train - oznaka pripadnosti skupu navedenog znaka
- x koordinata okvira koji označava položaj znaka unutar slike
- y koordinata okvira koji označava položaj znaka unutar slike
- širina okvira koji označava znak unutar slike
- visina okvira koji označava znak unutar slike

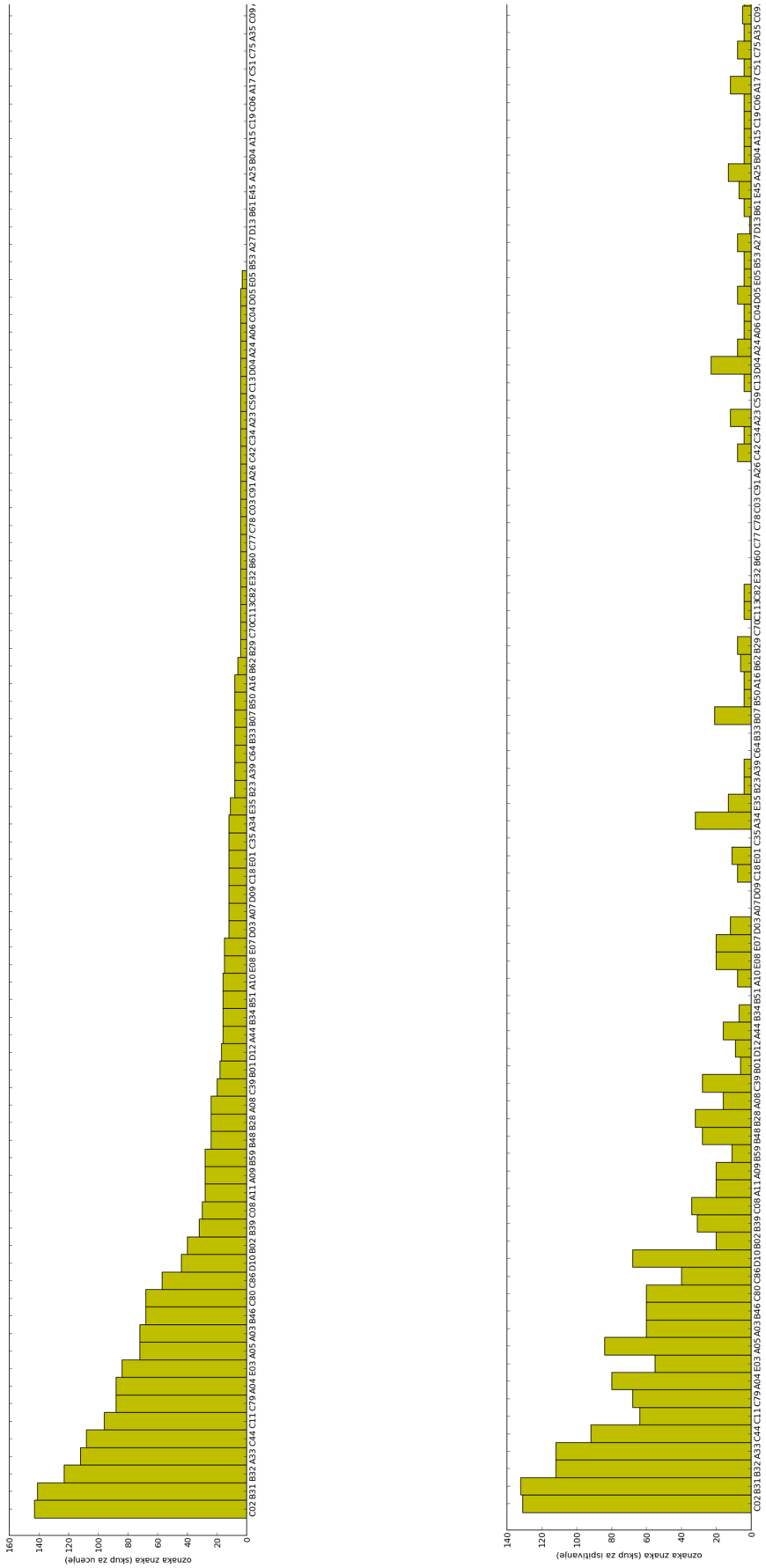
Izvadak 6.3: Isječak predložene anotacijske datoteke

```
1  C02_0110.bmp C02  Test  586   231   51    53
2  A05_0065.bmp A05  Test  486   244   34    32
3  D12_0007.bmp D10  Train 509   16    106   41
4  C02_0051.bmp C02  Test  664   225   41    47
5  C02_0011.bmp C02  Train 654   154   53    59
6  B02_0053.bmp B02  Train 606   222   17    48
7  C39_0001.bmp C39  Test  440   234   37    57
8  C44_0191.bmp C44  Test  651   229   51    57
9  A03_0120.bmp A03  Train 457   247   34    33
10 C02_0253.bmp C02  Test  551   235   35    36
```

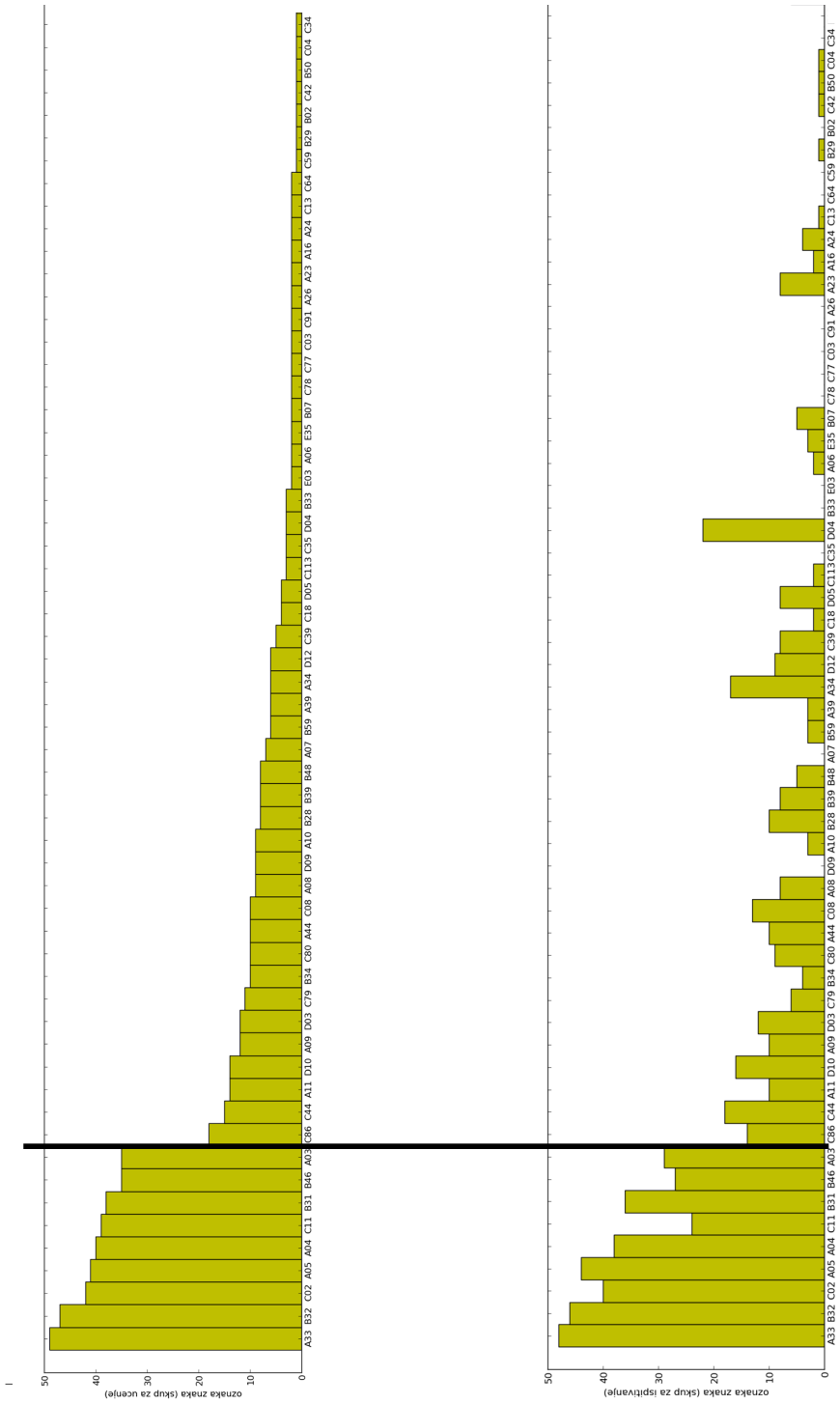
Na slici 6.4 prikazani su histogrami učestalosti pojave svakog pojedinog znaka u skupovima za učenje i ispitivanje. Iako se izvorno pojavljuje 87 različitih klasa prometnih znakova, velika većina ih nije pogodna za učenje klasifikatora. Odlučeno je trenirati klasifikator samo na klase znakova koje podliježu sljedećem uvjetu:

- Klasa sadrži barem 20 uzoraka (kako u ispitnom skupu, tako i u skupu za ispitivanje) koji su dimenzija većih ili jednakih od 44×44.

Znakova koji zadovoljavaju takav uvjet je samo 9. Zanimljivo je i napomenuti da svi takvi znaci zapravo imaju 35 ili više uzoraka po klasi (iako je izvorni uvjet bio samo 20). Klase takvih znakova su: C02, A04, B32, A33, C11, B31, A05, B46, A03 te su prikazani na slici 6.5.



Slika 6.3: Histogrami učestalosti pojave pojedinih klasa prometnih znakova u skupu za učenje i ispitnom skupu (svi znakovi).



Slika 6.4: Histogrami učestalosti pojave pojedinih klasa prometnih znakova u skupu za učenje i ispitnom skupu (znakovi veličine 44×44 ili veći). Crnom linijom je označena granica klasa koje su uzete u obzir u ovom radu (klase koje imaju barem 20 uzoraka većih ili jednakih 44×44).

Slika 6.5: Odabranih devet klasa prometnih znakova. Svaki znak se u skupu za učenje pojavljuje 500 puta, a u skupu za ispitivanje 100 puta.

Nadalje, kao i kod skupa MNIST, svakom je uzorku dodan rub širine 2 kako bi konvolucija mogla pravilno raditi i na rubnim značajkama. Kod skupa FER-MASTIF TS2010 se, za razliku od skupa MNIST, spremaju sve tri komponente boja (RGB), te se učenje provodi na takvom ulazu (detaljnije u poglavju "Experimentalni rezultati").

6.2.3. Proširenje skupa

S obzirom na maleni broj uzoraka u skupu (klasa s najmanje uzoraka ima tek 35 uzoraka unutar skupa), skup je proširen slučajnim transformacijama. Moguća su dva pristupa:

- Ugradnja dodatnog sloja (nakon ulaznog) koji obavlja slučajne transformacije prilikom svake iteracije. Takav je pristup korišten u [3] gdje se jako velikim brojem dodatnih uzoraka znatno poboljšava generalizacija sustava, čime je autor postigao tada najbolje rezultate (engl. *state of the art*). Međutim, takav pristup može dodatno usporavati učenje. U [3] se treniranje izvodilo tjedan dana na grafičkoj procesorskoj jedinici (GPU).
- Preprocesiranje skupa, odnosno generiranje dodatnog skupa uzoraka unaprijed. Manje je zahtjevno jer ne usporava učenje mreže, te za dovoljno veliki broj uzoraka postiže ponašanje jednako prvoj metodi. Također, ovaj problem pojednostavljuje spremanje podataka, kao što je već detaljnije opisano u poglavlju 4.7.1.

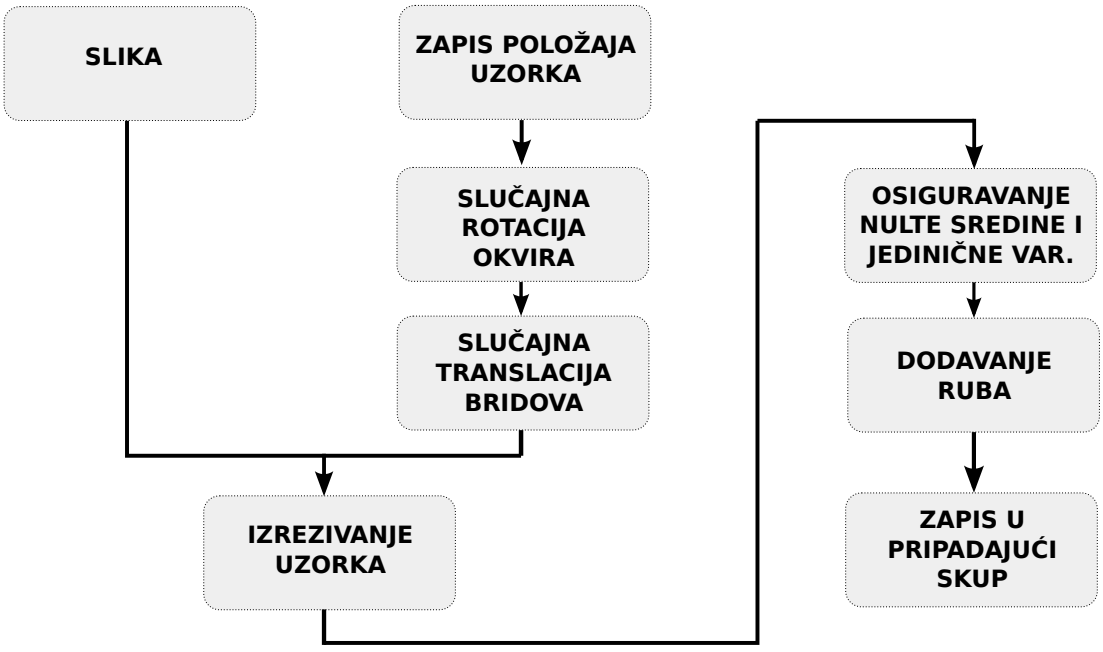
U ovom radu, kao i u [1] korišten je drugi pristup. Tako su za svaku klasu znakova generirani dodatni znakovi slučajnim transformacijama sve do ukupnog broja od 500 uzoraka po klasi za skup za učenje te 100 uzoraka po klasi za skup za ispitivanje. Slučajne transformacije su generirane na sljedeći način:

- **Slučajni pomak rubova** - uzorkovan iz $\mathcal{N}(0, 1)$.
- **Slučajna rotacija** - uzorkovana iz $\mathcal{N}(0, 5°)$.

Na kraju, kao i kod skupa MNIST, dodan je brid širine 2 koji služi poboljšavanju konvolucije kod rubnih značajki te je uzorak centriran tako da mu je aritmetička sredina jednaka nuli i normiran na jediničnu varijancu.

6.2.4. Spremanje skupa

Skup se sprema na sličan način kao i skup MNIST: kompresirani *Python* niz. Međutim, kako se kod skupa FER-MASTIF TS2010 spremaju sve tri komponente boja (RGB), svaki se uzorak sastoji od niza s tri elemenata. Svaki se element nadalje sastoji od dvodimenzionalnog niza veličine 48×48 u kojemu su spremljene vrijednosti pojedinih točaka date komponente.



Slika 6.6: Sustav predobrade uzoraka skupa FER-MASTIF TS2010

7. Eksperimentalni rezultati

7.1. Rezultati na skupu MNIST

7.1.1. Odabrana arhitektura mreže

Za skup MNIST odabrana je arhitektura slična arhitekturi *LeNet-5* [11] koja se sastoji od sljedećih slojeva:

- **Ulazni sloj** - jedna mapa veličine 32×32.
- **Prvi konvolucijski sloj** - 6 mapi veličine 28×28.
- **Prvi sloj sažimanja** - 6 mapi veličine 14×14.
- **Drugi konvolucijski sloj** - 16 mapi veličine 10×10.
- **Drugi sloj sažimanja** - 16 mapi veličine 5×5.
- **Treći konvolucijski sloj** - 100 mapi veličine 1×1.
- **Skriveni sloj** - 80 neurona.
- **Izlazni sloj** - 10 neurona.

Svi su slojevi potpuno povezani, metoda slučajnog izostavljanja neurona prilikom učenja (engl. *dropout*) nije korištena te se učenje provodi gradijentnim spustom sa konstantnom stopom učenja.

Iz opisane arhitekture je također jasno i jednoznačno definirano da:

- Prvi konvolucijski sloj sadrži 6 jezgri veličine 5×5.
- Drugi konvolucijski sloj sadrži 96 jezgri veličine 5×5.
- Treći konvolucijski sloj sadrži 1600 jezgri veličine 5×5.
- Svi slojevi sažimanja rade s prozorom veličine 2×2.

7.1.2. Rezultati

Odabrana je arhitektura trenirana sa deset prolaza kroz skup za učenje (dakle ukupno 600 000 iteracija) stohastičkim gradijentnim spustom. Tako trenirana mreža ima TP (engl. *True Positive*) (broj ispravno klasificiranih uzoraka) na ispitnom skupu od 98.67%. Detaljniji rezultati su dani matricom zabune u tablici 7.1.

		Predviđena klasa									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Stvarna klasa	**0**	973	0	1	0	0	0	3	1	2	0
	1	0	1127	4	1	0	0	1	0	2	0
	2	3	0	1020	0	0	0	0	4	5	0
	3	0	0	2	992	0	6	0	3	5	2
	4	1	0	1	0	963	0	4	0	2	11
	5	1	0	0	3	0	884	1	1	0	2
	6	10	2	0	0	1	2	943	0	0	0
	7	0	1	7	2	0	0	0	1014	1	3
	8	2	0	1	0	1	1	1	3	962	3
	9	3	2	0	3	1	3	1	4	3	989

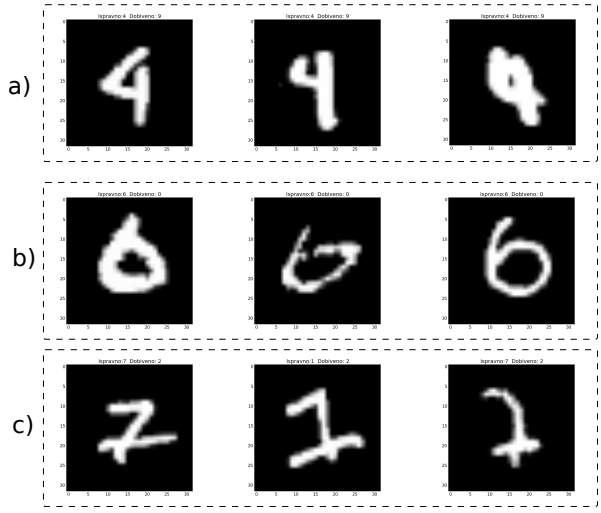
Tablica 7.1: Matrica zabune na ispitnom skupu MNIST

Tablica jasno pokazuje koje su greške najčešće. Skup za ispitivanje sadrži ukupno 10 000 uzoraka a tri najčešće greške su, redom:

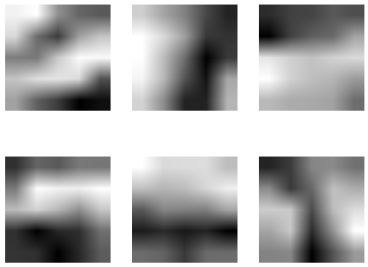
- Broj 4 klasificiran kao broj 9 (11 krivih klasifikacija).
- Broj 6 klasificiran kao broj 0 (10 krivih klasifikacija).
- Broj 7 klasificiran kao broj 2 (7 krivih klasifikacija).

Na slici 7.1 prikazano je nekoliko primjera krivo klasificiranih uzoraka iz ispitnog skupa MNIST. Vidljivo je da su i sa ljudskog aspekta takvi primjeri dosta slični predviđanjima mreže, iako je ljudskom oku jasna ispravna klasifikacija.

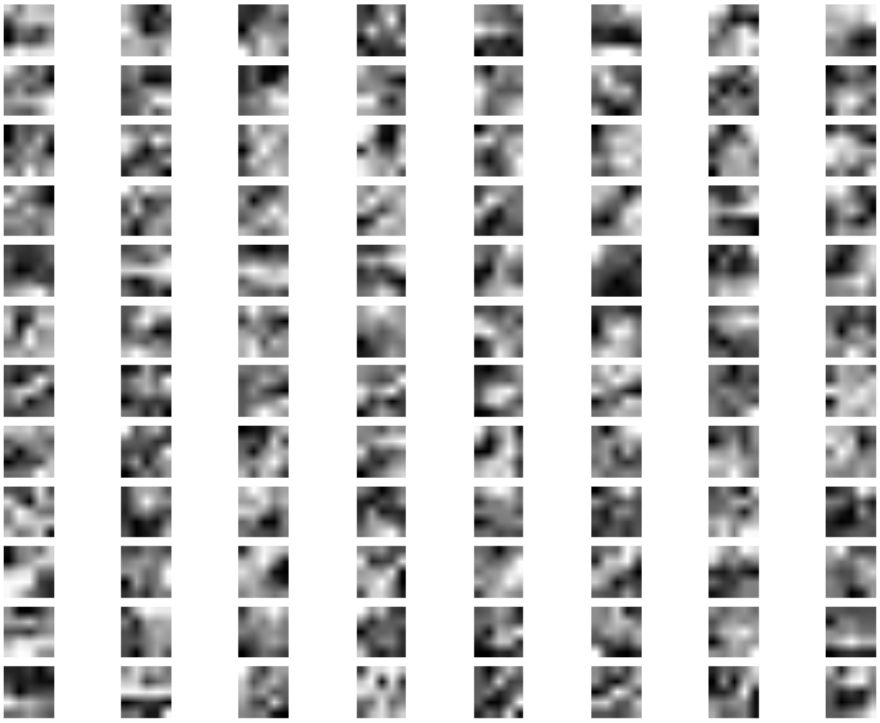
Na slikama 7.2 i 7.3 prikazane su naučene jezgre prvog i drugog konvolucijskog slojeva za skup MNIST.



Slika 7.1: Primjeri tri najučestalije krivo klasificiranih uzoraka skupa MNIST: a) Broj 4 klasificiran kao 9, b) Broj 6 klasificiran kao 0, c) broj 7 klasificiran kao 2



Slika 7.2: Prikaz naučenih jezgri prvog konvolucijskog sloja na skupu MNIST



Slika 7.3: Prikaz naučenih jezgri drugog konvolucijskog sloja na skupu MNIST

7.2. Rezultati na skupu FER-MASTIF TS2010

7.2.1. Odabrana arhitektura mreže

Za skup FER-MASTIF TS2010 odabrana je arhitektura slična arhitekturi *LeNet-5* [11] koja se sastoji od sljedećih slojeva:

- **Ulazni sloj** - tri mape veličine 48×48.
- **Prvi konvolucijski sloj** - 10 mapi veličine 42×42.
- **Prvi sloj sažimanja** - 10 mapi veličine 21×21.
- **Drugi konvolucijski sloj** - 15 mapi veličine 18×18.
- **Drugi sloj sažimanja** - 15 mapi veličine 9×9.
- **Treći konvolucijski sloj** - 20 mapi veličine 6×6.
- **Treći sloj sažimanja** - 10 mapi veličine 3×3.
- **Četvrti konvolucijski sloj** - 40 mapi veličine 1×1.
- **Skriveni sloj** - 80 neurona.
- **Izlazni sloj** - 9 neurona.

Kao i predložena arhitektura za skup MNIST, i ova je arhitektura potpuno povezana pa implicira sljedeća svojstva:

- Prvi konvolucijski sloj sadrži 30 jezgri veličine 7×7.
- Drugi konvolucijski sloj sadrži 150 jezgri veličine 4×4.
- Treći konvolucijski sloj sadrži 300 jezgri veličine 4×4.
- Četvrti konvolucijski sloj sadrži 800 jezgri veličine 3×3.
- Svi slojevi sažimanja rade s prozorom veličine 2×2.

7.2.2. Rezultati

Odabrana je arhitektura trenirana sa dvanaest prolaza kroz skup za učenje (dakle ukupno 54 000 iteracija) stohastičkim gradijentnim spustom. Tako trenirana mreža ima TP (engl. *True Positive*) (broj ispravno klasificiranih uzoraka) na ispitnom skupu od 98.22%. Detaljniji rezultati su dani matricom zabune u tablici 7.2.

Tablica jasno pokazuje koje su greške najčešće. Skup za ispitivanje sadrži ukupno 900 uzoraka, a tri najčešće greške su, redom:

- Znak A03 klasificiran kao znak A05 (7 krivih klasifikacija).

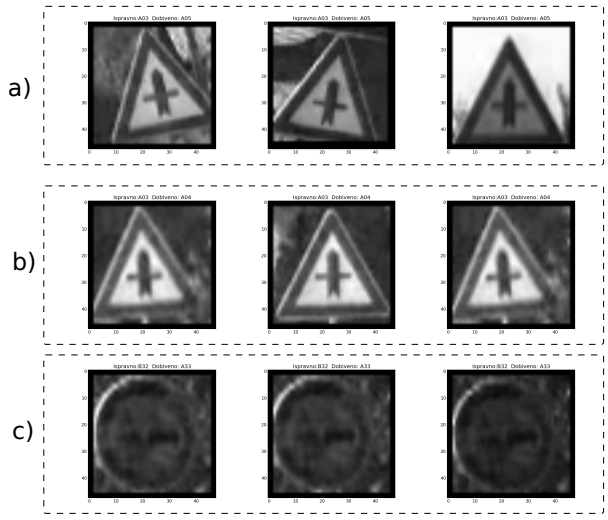
		Predviđena klasa								
										
Stvarna klasa	C02 	100	0	0	0	0	0	0	0	0
	A04 	0	99	0	0	0	0	1	0	0
	B32 	0	0	97	3	0	0	0	0	0
	A33 	0	0	0	100	0	0	0	0	0
	C11 	0	0	0	0	100	0	0	0	0
	B31 	0	0	0	0	0	100	0	0	0
	A05 	0	0	0	0	0	0	98	0	2
	B46 	0	0	0	0	0	0	0	100	0
	A03 	0	3	0	0	0	0	7	0	90

Tablica 7.2: Matrica zabune na ispitnom skupu FER-MASTIF TS2010

– Znak A03 klasificiran kao znak A04 (3 krivih klasifikacija).

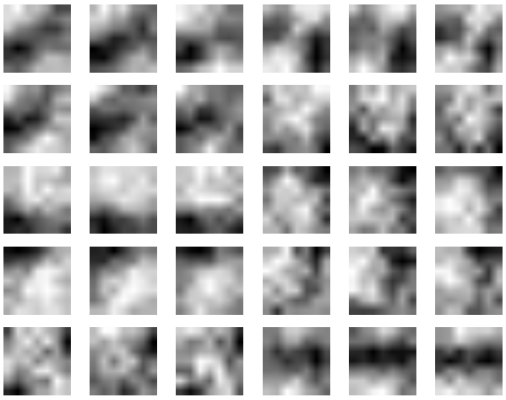
– Znak B32 klasificiran kao znak A33 (3 krivih klasifikacija).

Prve dvije najučestalije krive posve su logične, dok je treća (klasifikacija znaka B32 kao znak A33) pomalo neočekivana. Vrijedi primijetiti i da su se sve tri takve pojave desile kod tamnih uzoraka sa lošim kontrastom. Moguće je da bi takva greška nestala ukoliko bi se treniranje vršilo nad većim brojem klasa prometnih znakova a ne samo devet. U tom slučaju je preporučeno sakupiti više primjera za učenje.

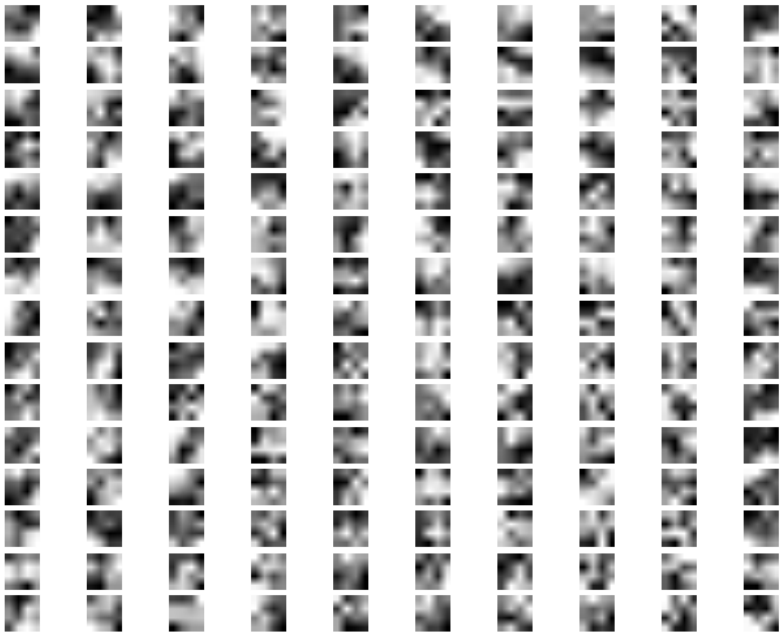


Slika 7.4: Primjeri tri najučestalije krivo klasificiranih uzoraka (veličine 44×44) skupa MASTIF: a) Znak A03 klasificiran kao znak A05, b) Znak A03 klasificiran kao znak A04, c) Znak B32 klasificiran kao znak A33

Na slikama 7.5 i 7.6 prikazane su naučene jezgre prvog i drugog konvolucijskog sloja za skup FER-MASTIF TS210.



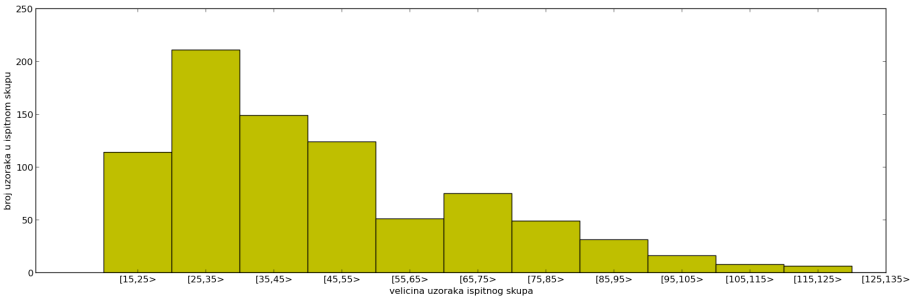
Slika 7.5: Prikaz naučenih jezgri prvog konvolucijskog sloja na skupu FER-MASTIF TS210



Slika 7.6: Prikaz naučenih jezgri drugog konvolucijskog sloja na skupu FER-MASTIF TS210

7.3. Utjecaj veličine uzoraka

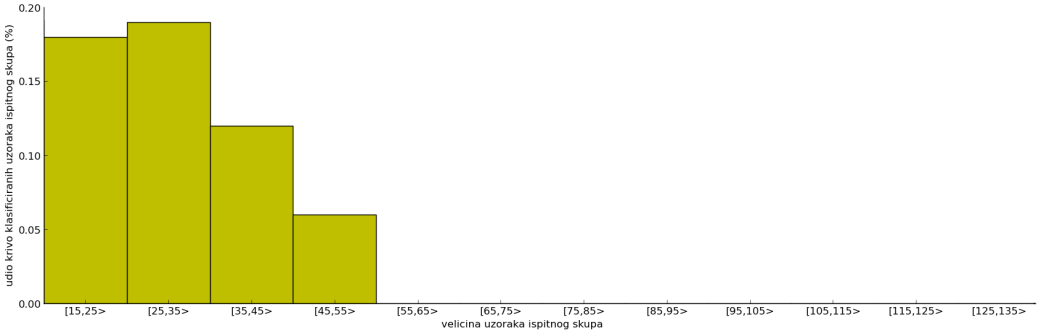
Uzorci u ispitnom skupu variraju u veličini od 15 točaka do 124 točaka (po najvećoj osi). Na slici 7.7 je prikazan broj uzoraka u ovisnosti o veličinama za odabranih 9 klasa uzoraka koji su trenirani i testirani.



Slika 7.7: Broj uzoraka u ovisnosti o veličini (odabranih 9 klasa)

U izvornim eksperimentima su odbačeni uzorci koji su bili manji od 44×44. Ovdje su međutim sačuvani svi uzorci odabranih klasa. Svi su uzorci skalirani na 44×44 (s obzirom da konvolucijske neuronske mreže rade sa ulazom fiksne veličine). Manji se uzorci interpoliraju na željenu veličinu. Skupovi su podjeljenji u grupe veličine 10 točaka (od 15 točaka do 25 točaka, od 25 točaka do 35, itd.) te su izračunate srednje vrijednosti grešaka u klasifikaciji. Na slici 7.8 su prikazani rezultati takve analize.

Kod manjih uzoraka (uzorci manji od 35×35 točaka) je greška oko 20% krivo



Slika 7.8: Utjecaj veličine uzoraka na klasifikaciju

klasificiranih. Povećanjem veličine uzoraka klasifikacija opada, prvo na 12% krivo klasificiranih za uzorke veličine između 35×35 i 45×45 točaka a zatim na 6% krivo klasificiranih uzoraka za veličine između 45×45 i 55×55. Za uzorke veće od 55×55 točaka krive klasifikacije nije bilo.

7.4. Usporedba izrađene biblioteke i biblioteke Caffe

Premda vlastita implementacija daje dobre rezultate (gotovo identične drugim implementacijama za skup MNIST bez dodatnih trikova za poboljšanje generalizacije) brzina je poprilično loša. U vlastitoj implementaciji klasifikacija uzorka veličine 48×48 traje gotovo 6 sekundi, što je za nekoliko redova veličina predugo za pretragu prozorom u snimkama u realnom vremenu.

Biblioteka *Caffe* [8] s druge strane klasificira sliku iste veličine u približno 20ms koristeći CPU jedinicu, što je realnije vrijeme za pretragu pomičnim prozorom. Iako u ovom radu to nije testirano, u opisu bliblioteke je prilikom korištenja jedinice GPU navedena oko 10 puta veća brzina što je sasvim prihvaljivo za detekciju i klasfikaciju u realnom vremenu.

Sporost vlastite implementacije ne ovisi toliko o jeziku (*Python*) koliko o količini koda koji nije niti vektoriziran niti paraleliziran. Prije eventualne implementacije dijelova koda u bržem jeziku (npr. C) bilo bi potrebno isprobati vektorizirane operacije konvolucije koje pruža *NumPy* umjesto trenutnih ugniježdenih petlji. Konvolucijske su neuronske mreže također prikladne za paralelizaciju mnogih dijelova (i pogodne za nestandardne načine paraleliziranog učenja [10]) te bi u eventualnom daljnjem razvijanju bilo preporučeno koristiti vektorizirane operatore konvolucije i paralelizaciju nezavisnih dijelova (primjerice, svaka se mapa značajki može nezavisno izračunati od drugih mapi u istom sloju).

8. Zaključak

Kroz ovaj su rad prikazani način rada i implementacija dubokih neuronskih mreža, primarno konvolucijske neuronske mreže. Sama implementacija nije ovisna o drugim bibliotekama te je iscrpno komentirana i čitljiva. Fokus implementacije nije bio na brzini izvođena već na jednostavnosti i čitljivosti.

Pojedini dijelovi mreže su najprije zasebno testirani. U svakom su građevnom elementu (jednodimenzionalni sloj, konvolucijski sloj i sloj sažimanja) ispitani gradijenti te je njihova konvergencija testirana. To se pokazalo veoma bitnim, s obzirom na veliku složenost dubokih neuronskih arhitektura naspram običnih višeslojnih perceptrona.

Nakon što je uspostavljeno da pojedini dijelovi uspješno konvergiraju, izgrađena je manja konvolucijska neuronska mreža za klasifikaciju rukom pisanih znamenki iz skupa MNIST [12]. Takva je mreža postigla rezultat koji je očekivan za konvolucijsku neuronsku mrežu te iznosi 98.67% ispravno klasificiranih uzoraka na skupu za ispitivanje. U sklopu ispitivanja, izrađena je i cjelokupna matrica zabune te su određeni najčešće krivo klasificirani primjeri. Uvođenjem slučajnih distorzija (kao što je to učinjeno na skupu FER-MASTIF TS2010, mogla bi se poboljšati mogućnost generalizacije za mogućih dodatnih 1%.

Drugi dio eksperimenta sastojao se u izradi veće konvolucijske neuronske mreže i njezinom treniranju na pogodnom podskupu hrvatskih prometnih znakova skupa FER-MASTIF TS2010. Odabrano je devet različitih klasa prometnih znakova, kako bi dovoljan broj uzoraka bio zadovoljavajuće veličine. Pokazano je kako je tako trenirana mreža postigla rezultat od 98.22% ispravno klasificiranih znakova. Ponovno je izrađena cjelokupna matrica zabune te su analizirani najčešće krivo klasificirani znakovi.

U eventualnom daljnjem radu u poboljšanju prepoznavanja prometnih znakova hrvatskih cesti i proširenju broja znakova koji je moguće prepoznati, preporuča se povećanje baze znakova za barem jedan red veličine. Također, preporučeno je ubrzati (npr. vektoriziranjem pojedinih djelova koda i uvođenjem paralelizacije) postojeću implementaciju ili koristiti neku već optimiziranu biblioteku za konvolucijske neuronske mreže.

LITERATURA

[1] Igor Bonači, Ivan Kusalić, Ivan Kovaček, Zoran Kalafatić, i Siniša Šegvić. Addressing false alarms and localization inaccuracy in traffic sign detection and recognition. 2011.

[2] Y Boureau, J Ponce, i Y LeCun. A theoretical analysis of feature pooling in vision algorithms. U *Proc. International Conference on Machine learning (ICML'10)*, 2010.

[3] Dan Cireşan, Ueli Meier, Jonathan Masci, i Jürgen Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338, 2012.

[4] Corinna Cortes i Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[5] Li Fei-Fei, Rob Fergus, i Antonio Torralba. Recognizing and learning object categories. *CVPR Short Course*, 2, 2007.

[6] Xavier Glorot i Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. U *International Conference on Artificial Intelligence and Statistics*, stranice 249–256, 2010.

[7] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, i Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[8] Yangqing Jia. Caffe: An open source convolutional architecture for fast feature embedding. `http://caffe.berkeleyvision.org/`, 2013.

[9] Josip Krapac i Siniša Šegvić. Weakly supervised object localization with large fisher vectors.

[10] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

[11] Yann LeCun i Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361, 1995.

[12] Yann Lecun i Corinna Cortes. The MNIST database of handwritten digits. URL `http://yann.lecun.com/exdb/mnist/`.

[13] Yann LeCun, Léon Bottou, Yoshua Bengio, i Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[14] Yann A LeCun, Léon Bottou, Genevieve B Orr, i Klaus-Robert Müller. Efficient backprop. U *Neural networks: Tricks of the trade*, stranice 9–48. Springer, 2012.

[15] David G Lowe. Object recognition from local scale-invariant features. U *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, svezak 2, stranice 1150–1157. Ieee, 1999.

[16] Dominik Scherer, Andreas Müller, i Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. U *Artificial Neural Networks–ICANN 2010*, stranice 92–101. Springer, 2010.

[17] Pierre Sermanet, Soumith Chintala, i Yann LeCun. Convolutional neural networks applied to house numbers digit classification. U *Pattern Recognition (ICPR), 2012 21st International Conference on*, stranice 3288–3291. IEEE, 2012.

[18] Patrice Simard, David Steinkraus, i John C Platt. Best practices for convolutional neural networks applied to visual document analysis. U *ICDAR*, svezak 3, stranice 958–962, 2003.

[19] Siniša Šegvić, Karla Brkić, Zoran Kalafatić, i Axel Pinz. Exploiting temporal and spatial constraints in traffic sign detection from a moving vehicle. *Machine Vision and Applications*.

[20] Siniša Šegvić, Karla Brkić, Zoran Kalafatić, Vladimir Stanisavljević, Marko Ševrović, Damir Budimir, i Ivan Dadić. A computer vision assisted geoinformation inventory for traffic infrastructure. *ITSC 2010, Madeira, Portugal*, 2010.

Raspoznavanje objekata dubokim neuronskim mrežama

Sažetak

U ovom su radu prikazani načini rada i implementacijski detalji dubokih neuronskih mreža sa fokusom na konvolucijske neuronske mreže. Napravljen je sustav za brzu izradu različitih konvolucijskih neuronskih mreža i njihovo testiranje. Testirane su dvije različite konvolucijske neuronske mreže: jedna za klasifikaciju podataka iz skupa rukom pisanih znamenki MNIST te druga za prepoznavanje hrvatskih prometnih znakova iz skupa FER-MASTIF TS2010. Na kraju su opisani dobiveni rezultati te su analizirani krivo prepoznati uzorci.

Ključne riječi: neuronske mreže, duboke neuronske mreže, konvolucijske neuronske mreže, višeklasna klasifikacija, raspoznavanje prometnih znakova, FER-MASTIF TS2010, MNIST, računalni vid, raspoznavanje uzoraka, duboko učenje

Deep neural networks in object recognition

Abstract

In this work, the specific mechanisms and implementational details of deep neural networks are shown with an emphasis on convolutional neural networks. A system for fast convolutional neural network prototyping is made and two different CNNs are tested: one for handwritten digits recognition, based on the MNIST dataset, and the second for Croatian traffic signs recognition, based on the FER-MASTIF TS2010 dataset. Following that, numerical results are expressed from the classifiers, their performance is evaluated and misclassified examples are briefly discussed.

Keywords: neural networks, deep neural networks, convolutional neural networks, multiclass classification, traffic signs recognition, FER-MASTIF TS2010, MNIST, computer vision, pattern recognition, deep learning