SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1013

Određivanje smjera gledanja konvolucijskim neuronskim mrežama

Mirko Jurić-Kavelj

Zagreb, veljača 2015.

Želim se zahvaliti svom mentoru izv. prof. dr. sc. Siniši Šegviću i kolegi mr. sc. Josipu Krapcu na pomoći i savjetima prilikom izrade diplomskog rada.

Također bih se htio zahvaliti svom bratu Srećku koji mi je bio potpora tokom cijelog studija i uvijek mi je pomagao kada mi je to trebalo.

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

1. Uvod

Osjetilo vida je najbitnije čovjekovo osjetilo koje nam omogućuje snalaženje u prostoru kao i prikupljanje informacija. Štoviše, većinu informacija primamo upravo vizualnim putem, bilo da čitamo članke u novinama, na Internet portalima ili gledamo televiziju. Određivanje smjera pogleda i nalaženje predmeta koje osoba gleda u prostoru predstavlja jedno zanimljivo područje istraživanja, područje koje dugo vremena nije bilo moguće istražiti zbog nedovoljno razvijene tehnologije. Podatci dobiveni tim istraživanjima mogu poslužiti kao podloga za objašnjavanje načina na koji ljudi promatraju svijet oko sebe.

Postoje razne metode za praćenje pogleda, no sve se mogu svrstati u sljedeće dvije skupine s obzirom na fizički utjecaj na korisnika: **nametljive** i **nenametljive**. Glavna razlika je u tome što nametljive metode zahtijevaju opremu koja je u fizičkom kontaktu s korisnikom, dok se kod nenametljivih oko snima kamerom. Nametljive metode su se pokazale preciznijim od nenametljivih. No dok su nametljive pogodne samo za upotrebu u laboratorijskim uvjetima, nenametljive su pogodnije za istraživanja u svakodnevnim okruženjima, pošto nikakva dodatna oprema nije potrebna, te subjekt ne mora niti biti svjestan sustava za praćenje.

Neuronske mreže su matematički modeli, inspirirani biološkim modelom mozga i neurona, koji se koriste za rješavanje problema koje je teško ili nemoguće riješiti analitičkim metodama. Njihova glavna karakteristika je prilagodljivost problemu za koji se koriste, pošto se značajke ne modeliraju ručno već je zadatak mreže da automatski nauči takve značajke iz datog skupa za učenje. Jedan tip neruonskih mreža su tzv. *konvolucijske* neuronske mreže.

U ovom radu bit će opisana jedna nenametljiva metoda koja se bazira na konvolucijskim neuronskim mrežama za određivanje smjera pogleda. Rad počinje s opisom teorijskih podloga i objašnjenja vezanih uz neuronske mreže i konvolucijske neuronske mreže respektivno, te će ukratko biti opisane motivacija i način na koji su se razvili spomenuti modeli. Potom će biti opisan programski sustav za učenje konvolucijskih neuronskih mreža *Caffe* [1], te će biti objašnjeno kako se taj sustav koristi. Biti će opi-

sani dodatni programi za prikupljanje podataka za učenje i skripte za obradu rezultata (implementirani u sklopu ovog rada), kao i ispitni skupovi nad kojima su učene mreže. Na kraju će biti diskutirani dobiveni rezultati.

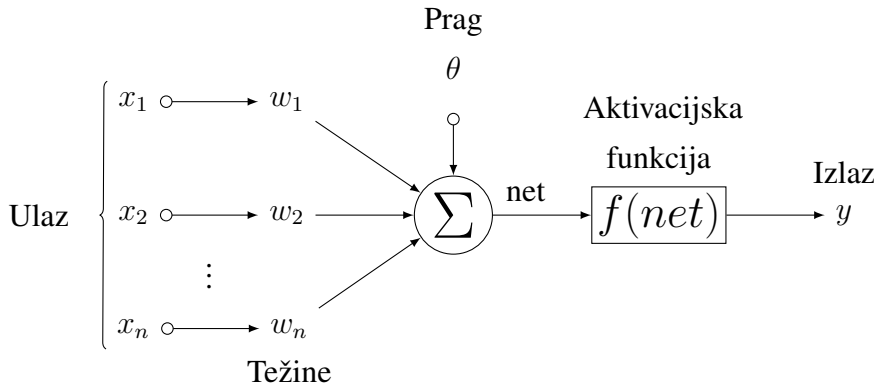
2. Umjetne neuronske mreže

Razvoj neuroračunarstva započeo je sredinom 20. stoljeća spojem niza bioloških spoznaja i njihovim modeliranjem prikladnim za računalnu obradu. Još davne 1943. Warren McCulloch i Walter Pitts u radu *A Logical Calculus of Ideas Immanent in Nervous Activity* su defnirali model umjetnog neurona (tzv. TLU perceptron) i pokazali da se njime mogu računati logičke funkcije I, ILI i NE. Uzevši u obzir da se uporabom tih triju funkcija može izgraditi proizvoljna Booleova funkcija, slijedi da se izgradnjom (tj. prikladnom konstrukcijom) umjetne neuronske mreže sastavljene od TLU neurona može dobiti proizvoljna Booleova funkcija. Međutim, problem koji ovdje valja istaknuti jest da se radi o rješavanju problema konstrukcijom, a ne učenjem.

Nekoliko godina kasnije, britanski biolog Donald Hebb u svojoj knjizi *The Organization of Behavior* iz 1949. godine objavljuje svoje spoznaje o radu i interakciji bioloških neurona. Konkretno, njegovo je zapažanje da, ako dva neurona često zajedno pale, tada dolazi do metaboličkih promjena kojima efikasnost kojom jedan neuron pobuđuje drugoga s vremenom raste. Tom spoznajom stvoreni su svi preduvjeti za definiranje algoritma učenja TLU perceptrona, i to je upravo kroz izvještaj *The Perceptron: A Perceiving Principles and Recognizing Automaton* iz 1957. i knjigu *Principles of Neurodynamics* iz 1962. napravio Frank Rosenblatt defnirajući algoritam učenja TLU perceptrona, iskoristivši Hebbovu ideju da *učiti znači mijenjati jakost veza između neurona*. [7]

U strojnom učenju, umjetne neuronske mreže su skupina statističkih algoritama učenja inspiriranih biološkim neuronskim mrežama koje se koriste za estimiranje i aproksimaciju funkcija koje ovise o velikom broju ulaza i općenito su nepoznate. Prikazuju se kao sustavi povezanih *neurona*. Kao i drugi postupci strojnog učenja (sustavi koji uče iz skupa podataka), umjetne neuronske mreže su korištene za rješavanje širokog spektra različitih problema koje je teško riješiti koristeći obično programiranje bazirano na pravilima, uključujući računalni vid i raspoznavanje govora.

S obzirom na niz dobrih svojstava neuronskih mreža, poput njihove robusnosti na pogreške u podatcima, otpornosti na šum, masovno paralelnu obradu podataka te mo-



Slika 2.1: Prikaz modela neurona

gućnost učenja, ova je paradigma i danas u fokusu istraživanja i primjene.

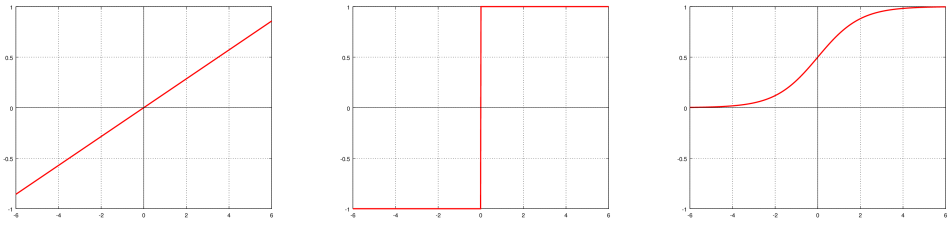
2.1. Model umjetnog neurona

Osnovna građevna jedinica neuronske mreže jest **neuron**. Umjetni neuron je osmišljen po uzoru na biološki neuron koji se sastoji od **dendrita**, **aksona** i **tijela** neurona. Dendriti predstavljaju ulaze preko kojih neuron prikuplja informacije. Tijelo stanice ih obrađuje i na kraju generira rezultat koji se prenosi kroz akson na izlaz neurona. Prema tom biološkom modelu ustrojen je umjetni neuron. Model se sastoji od ulaza x_1 do x_n (*dendriti*), težina w_1 do w_2 koje određuju u kojoj mjeri svaki od ulaza pobuđuje neuron, tijela neurona koje računa ukupnu pobudu (označenu kao *net*) te aktivacijske funkcije $f(net)$ (*akson* neurona) koja ukupnu pobudu obrađuje i prosljeđuje na izlaz. Struktura jednog neurona prikazana je na slici 2.1.

Ukupna pobuda *net* računa se prema izrazu:

$$net = w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_n \cdot x_n - \theta \tag{2.1}$$

pri čemu θ predstavlja prag paljenja neurona. Prag 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). [4] Zbog jednostavnosti se često uvodi supstitucija $\theta = 1 \cdot w_0$ (gdje je jedinica fiktivni ulaz x_0) kojom se prag može predstaviti kao dodatna težina koja je uvijek spojena na



(a) Linearna funckija **(b)** Funkcija praga **(c)** Sigmoidalna funkcija

Slika 2.2: Aktivacijske funkcije

jedinicu. Tada se funkcija ukupne pobude može zapisati kao:

$$net = w_0 \cdot x_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_n \cdot x_n \tag{2.2}$$

$$= \sum_{i=0}^{n} w_i \cdot x_i \tag{2.3}$$

$$= \vec{w}^T * \vec{x} \tag{2.4}$$

pri čemu je $\vec{x}^T = (1, x_1, x_2, \ldots, x_n)$ $(n+1)$-dimenzijski vektor koji predstavlja ulaz a $\vec{w}^T = (1, w_1, w_2, \ldots, w_n)$ (n+1)-dimenzijski vektor težina. [7]

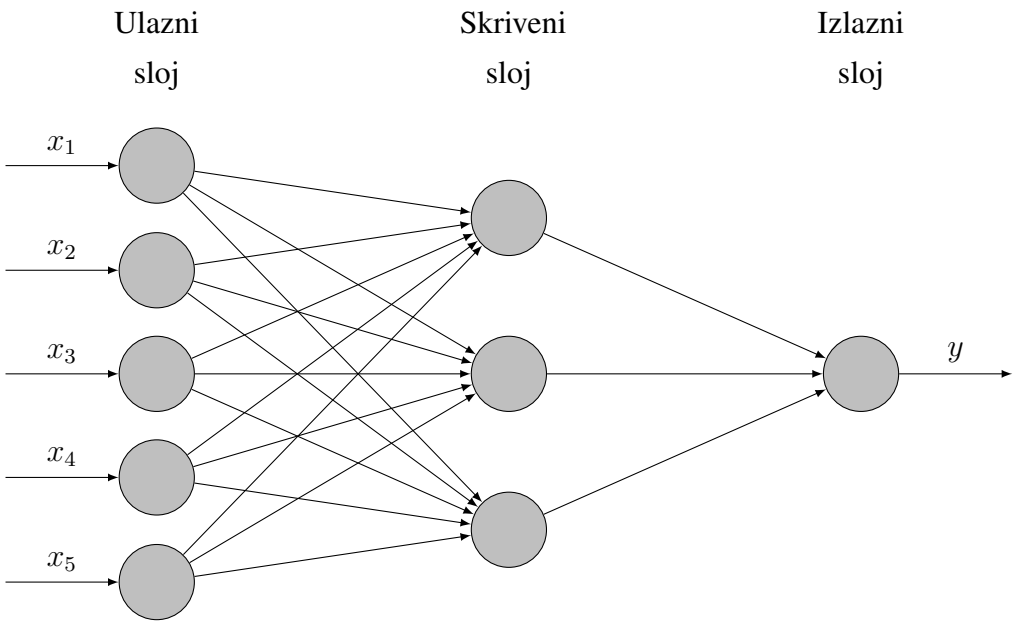
Aktivacijska funkcija modelira ponašanje aksona. U praksi se koristi nekoliko tipičnih aktivacijskih funkcija koje su prikazane na slici 2.2.

Linearne funkcije se koriste u izlaznim slojevima kada je potreban neograničen izlaz. Neograničen izlaz je često potreban kod regresijskih problema, no u klasifikacijskim problemima on bi samo otežavao učenje. Kod klasifikacijskih problema bolje je ograničavati izlaz na mali interval te vršiti klasifikaciju odabirom najvećeg izlaza. Stoga se funkcije skoka i sigmoidalna funkcija koriste u klasifikacijskim problemima. Međutim, od navedenih funkcija, za učenje neurona odgovara jedino sigmoidalna 2.5, pošto je ona derivabilna te se može koristi u algoritmu *Backpropagation*.

$$f(net) = \frac{1}{1 + e^{-net}} \tag{2.5}$$

Derivacija sigmoidalne funkcije je prikazana formulom 2.6

$$\frac{df(net)}{dnet} = f(net) \cdot (1 - f(net)) \tag{2.6}$$



Slika 2.3: Struktura poptuno povezane neuronske mreže

2.2. Potpuno povezana unaprijedna umjetna neuronska mreža

Jedan neuron može obavljati samo jednostavnu obradu podataka, stoga se za složenije probleme neuroni povezuju u različite strukture poznate pod nazivom *umjetne neuronske mreže*. Jedna od takvih struktura se naziva *potpuno povezana unaprijedna neuronska mreža* koju vidimo na slici 2.3.

Općenito se neuronska mreža sastoji od 3 tipa različitih slojeva.

- Ulazni sloj sastoji se od neurona koji ne obavljaju nikakvu funkciju već samo preslikavaju dobivene ulazne podatke i čine ih dostupnima ostatku mreže. To su neuroni čija aktivacijska funkcija je funkcija **identiteta**.
- Skriveni slojevi su svi slojevi koji nisu ili ulazni ili izlazni sloj. Drugim riječima to su slojevi koji nemaju nikakvog doticaja s okolinom, već kao ulaze dobivaju izlaze neurona iz prethodnih slojeva, a izlaze predaju neuronima u narednim slojevima.
- Izlazni sloj je zadnji sloj u mreži. Izlazi ovoga sloja su upravo rezultati koje cijela mreža producira.

Treba napomenuti kako broj neurona u ulaznom i izlaznom sloju, za razliku od skrivenih slojeva, nije proizvoljan već ovisi o dimenziji ulaznog vektora $\vec{x}$ i izlaznog vektora

$\vec{y}$.

Postupak računanja izlaza mreže je postepen i računa se sloj po sloj. Promotrimo postupak na primjeru jednog sloja. Uzmimo l-ti sloj u mreži i recimo da taj sloj kao ulaz prima $(n+1)$-dimenzijski vektor $\vec{x}$, te da taj sloj sadrži m neurona. Tada izraz za računanje izlaza izgleda ovako:

$$y_i^{(l)} = f(\vec{w}_i^{(l)T} * \vec{x}), \forall i \in \{1, \ldots, m\} \tag{2.7}$$

Uočimo da se ovaj izraz može pojednostaviti ako sve težine za svaki neuron u l-tom sloju napišemo u matrici $\vec{W}^{(l)}$:

$$W^{(l)} = \begin{bmatrix} w_{1,0} & \ldots & w_{1,j} & \ldots & w_{1,n} \\ w_{2,0} & \ldots & w_{2,j} & \ldots & w_{2,n} \\ \vdots & \ldots & \vdots & \ldots & \vdots \\ w_{m,0} & \ldots & w_{m,j} & \ldots & w_{m,n} \end{bmatrix} \tag{2.8}$$

Uz ovakav zapis težina, izlaz se može računati kao:

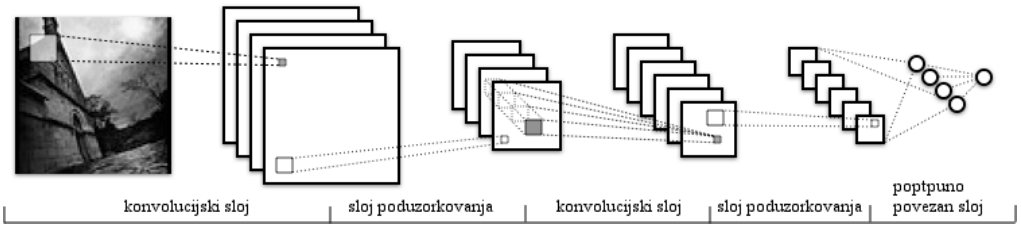
$$\vec{y}^{(l)} = f(W^{(l)} * \vec{x}) \tag{2.9}$$

3. Konvolucijske neuronske mreže

Konvolucijske mreže se mogu prikazati kao proširenje klasičnih višeslojnih unaprijednih neuronskih mreža. Do sada smo pričali o ulazima, težinama i izlazima mreže koji su bili skalari, no kod konvolucijksih mreža uvodi se proširenje na $\mathbb{R}^2$. To znači da su ulazi, težine i izlazi ustvari matrice, a ne skalari. Matrice koje imaju ulogu težina nazivaju se jezgre (engl. *kernel*), a izlaze nazivamo mapama značajki (engl. *feature maps*).

Struktura jedne takve mreže izgleda ovako. Na ulazu može biti jedna monokromatska slika ili višekanalna slika u boji. Zatim slijede naizmjenice konvolucijski slojevi i slojevi sažimanja (engl. *pooling*). Konvolucijski slojevi i slojevi sažimanja 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. Na samom kraju se nalazi nekoliko potpuno povezanih slojeva (klasični perceptron) koji su jednodimenzionalni, uključujući i izlazni sloj.

Konvolucijski slojevi uvijek imaju vezu jedan-na-više s prethodnim slojem te uče težine u jezgrama s kojima obavljaju konvoluciju, dok su slojevi sažimanja uvijek u jedan-na-jedan vezi s prethodnim slojem te ne uče nikakve vrijednosti (eventualno neke varijante pamte način propagacije greške). Tipični primjeri konvolucijskih neuronskih mreža imaju oko desetak slojeva (čime jasno opravdavaju svoje mjesto u kategoriji dubokih neuronskih mreža). Primjer jedne takve mreže vidimo na slici 3.1.



Slika 3.1: Primjer konvolucijske neuronske mreže.

3.1. 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 2D konvoluciju (detaljnije objašnjenoj u idućem podpoglavlju) s jezgrom. Neka je M^u ulaz trenutnog sloja (tj. izlaz prethodnog), a M^i izlazna mapa trenutnog 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 3.1 i 3.2.

$$M_w^i = \frac{M_w^u - K_w}{S_w} + 1 \tag{3.1}$$

$$M_h^i = \frac{M_h^u - K_h}{S_h} + 1 \tag{3.2}$$

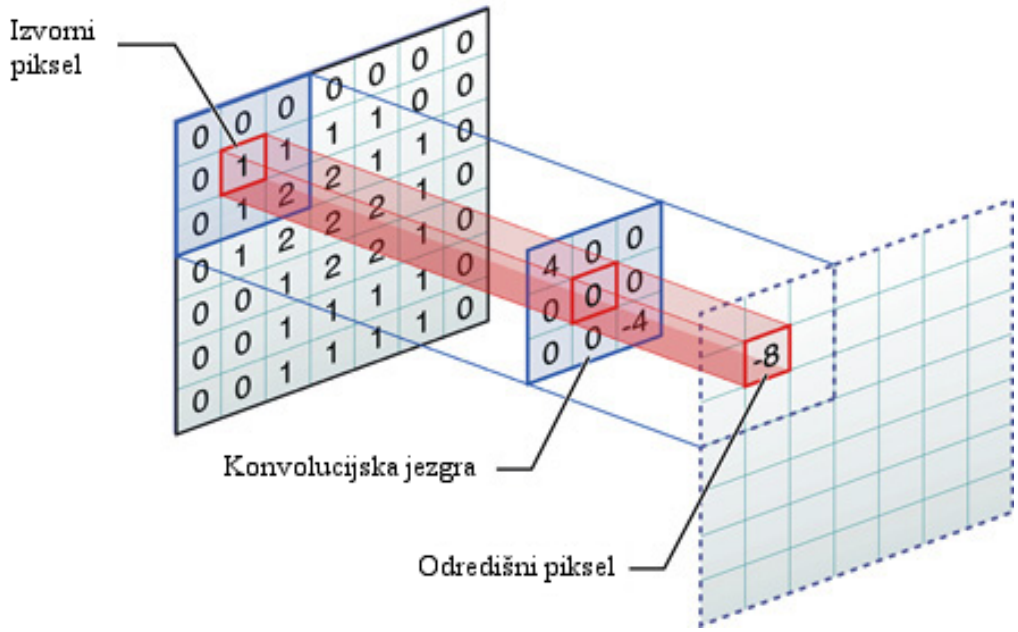
3.1.1. Konvolucija

U obradi slike, jezgra, konvolucijska matrica, ili maska je mala matrica koja se koristi za zamagljivanje, oštrenje, utiskivanje (graviranje), detekciju rubova, i još mnogo toga. To se postiže **konvolucijom** između jezgre i slike (prikazana na slici 3.2). 2D konvolucija koja je jedna od oblika matematičke konvolucije. Konvolucija se tvori "prolazom" kroz ulaznu matricu s prozorom veličine iste kao i jezgra. Jezgra se prvo pozicionira na gornji lijevi kraj ulazne matrice, potom se izvršava *element-wise* množenje (Hadamardov produkt, poznat i kao Schurov produkt) [6] jezgre i podmatrice ulazne matrice koju jezgra "pokriva". Dobivene vrijednosti se sumiraju[1], te se cijeli postupak ponavlja na sljedećoj poziciji. Jezgra "putuje" od lijeva prema desno do kraja retka, te tada prelazi u novi redak i počinje cijeli postupak ponovo od lijevog kraja sve dok ne dođe do donjeg desnog kraja ulazne matrice[2]

Normalizacija je proces kojim se svaki element izlazne matrice dijeli sa sumom apsolutnih vrijednosti u jezgri. Normalizacija osigurava da su vrijednosti piksela u izlaznoj slici iste relativne veličine kao i oni iz ulazne slike. Na slici 3.2 vidimo princip rada konvolucije.

[1] U Caffe-u je moguće dodati i dodatni parametar prag koji se pridodaje ovoj sumi. Prag je parametar mreže koji se isto uči.

[2] U Caffe-u se mogu definirati horizontalni i vertikalni pomaci jezgre u slikama. Ovo će biti detaljnije opisano u poglavlju 4.4.3.



Slika 3.2: Princip rada konvolucije. Izlazni piksel nije *normaliziran*.

Konvolucija uobičajeno zahtijeva vrijednosti piksela koje se nalaze izvan granica slike. Postoji nekoliko različitih načina rješavanja ovog problema.

Proširivanje (engl. *extend*) Rubni pikseli slike se proširuju koliko je potrebno za konvoluciju. Kutni pikseli se proširuju s 90° stupnjevitim klinovima, tj. u površinu. Ostali pikseli se proširuju u linijama. Ova metoda ilustrirana je na slici 3.3.

Omatanje (engl. *wrap*) Slika je konceptualno "omotana", tj. potrebni pikseli se uzimaju sa suprotnog ruba ili kuta.

Slika 3.3: Proširivanje slike za konvoluciju. Retci i stupci se proširuju u linijama, a kutovi u površinu.

Podrezivanje (engl. *crop*) Svaki piksel kojemu je potrebna vrijednost izvan slike se zanemaruje. Ova metoda rezultira slikom koja je malo manja od izvorne, s izrezanim rubovima.

3.2. Slojevi sažimanja

Postupak sažimanja smanjuje rezoluciju mapi značajki te povećava prostornu invarijantnost neuronske mreže. Ovo je bitno kod klasifikacije objekata i detekcije. Postupak se provodi tako da se pravokutnik podataka iz mape veličine $S_w \times S_h$ grupira zajedno i postavi jednom vrijednošću. Broj mapa ostaje isti prilikom provođenja postupka sažimanja. Neke od metoda sažimanja su sljedeće:

3.2.1. MAX-POOLING

Sažimanje maksimalnom vrijednošću (engl. *max-pooling*) je metoda sažimanja značajki koja odabire maksimalnu vrijednost unutar grupe te ju propagira u idući sloj. Recimo da imamo matricu sažimanja veličine 2×2, tada imamo sažimanje funkcionira kao u sljedećem primjeru:

$$\left[\begin{array}{cc|cc} 1 & \mathbf{3} & \mathbf{9} & 8 \\ 2 & 2 & 5 & 6 \\ \hline 1 & \mathbf{7} & 2 & 4 \\ 2 & 5 & \mathbf{8} & 7 \end{array}\right] \rightarrow \begin{bmatrix} 3 & 9 \\ 7 & 8 \end{bmatrix}$$

3.2.2. MEAN-POOLING

Sažimanje usrednjavanjem (engl. *mean-pooling*) je postupak koji uzima vrijednosti iz pojedinih regija te ih predstavlja aritmetičkom sredinom svih elemenata grupe. Sljedeći primjer pokazuje kako funkcionira usrednjavanje:

$$\left[\begin{array}{cc|cc} 1 & 3 & 9 & 8 \\ 2 & 2 & 5 & 6 \\ \hline 1 & 7 & 1 & 4 \\ 3 & 5 & 8 & 7 \end{array}\right] \rightarrow \begin{bmatrix} 2 & 7 \\ 4 & 5 \end{bmatrix}$$

3.2.3. GAUSSOVO-USREDNJAVANJE

Ova tehnika sažimanja je vrlo slična sažimanju usrednjavanjem. Jedina razlika jest u činjenici da se kod običnog usrednjavanja računa jednostavna aritmetička sredina,

dok se kod Gaussovog usrednjavanja računa težinska sredina pomoću Gaussove jezgre koja veće težine pridodaje elementima na sredini, a manje elementima koji su dalje od sredine jezgre.

3.3. Učenje konvolucijske neuronske mreže

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.

3.3.1. 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.

U svim oznakama vrijedi konvencija prethodnog sloja sa i te označavanja trenutnog sloja sa j. Stoga y_j oznacava 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:

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j} \tag{3.3}$$

Prvi dio desne strane izraza predstavlja parcijalnu derivaciju greške po izlazu neurona ($\frac{\partial E}{\partial y_j}$) dok drugi dio predstavlja parcijalnu derivaciju izlaza po ulazu neurona ($\frac{\partial y_j}{\partial z_j}$). Greška izlaza neurona se obično računa srednjom kvadratnom pogreškom:

$$E = \frac{1}{2}\sum_j (t_j - y_j)^2 \tag{3.4}$$

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

$$\frac{\partial E}{\partial y_j} = \frac{1}{2}\frac{\partial}{\partial y_j}(t_j - y_j)^2 = \frac{1}{2}\frac{\partial}{\partial y_j}(t_j^2 - 2t_j y_j + y_j^2) = y_j - t_j = -(t_j - y_j) \tag{3.5}$$

Uzme li se, primjerice, logistička sigmoidalna funkcija $y_j = f(z_j) = \frac{1}{1+e^{-z}}$ desni dio izraza 3.3 postaje:

$$\begin{aligned}\frac{\partial y_j}{\partial z_j} &= \frac{\partial}{\partial z_j}\left(\frac{1}{1+e^{-z_j}}\right) = \frac{\partial}{\partial z_j}(1+e^{-z_j})^{-1} = \frac{-1(-e^{-z_j})}{(1+e^{-z_j})^2} \\ &= \frac{1}{1+e^{-z_j}}\frac{e^{-z_j}}{1+e^{-z_j}} = y_j\frac{e^{-z_j}}{1+e^{-z_j}} = y_j\frac{(1+e^{-z_j})-1}{1+e^{z_j}} = \\ &= y_j\frac{1+e^{-z_j}}{1+e^{-z_j}}\frac{-1}{1+e^{-z_j}} = y_j(1-y_j)\end{aligned} \tag{3.6}$$

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

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial y_i} = \sum_j w_{ij}\frac{\partial E}{\partial z_j} \tag{3.7}$$

Na samom kraju, određuju se parcijalne derivacije po težinama:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}} = \frac{\partial E}{\partial z_j}y_i \tag{3.8}$$

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

$$w_{ij} \leftarrow w_{ij} - \eta\frac{\partial E}{\partial w_{ij}} \tag{3.9}$$

3.3.2. Varijante gradijentnog spusta

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

Standardni (grupni) (engl. *batch gradient descent*) predstavlja varijantu u kojoj se najprije izračunaju potrebne promjene težina Δw koje nastaju tako da se prvo izračuna ukupna greška nad cijelim skupom za učenje, ili grupom (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 neuronskih mreža.

Stohastički (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 mreža ovom varijantom, moguće je ubrzati postupak metodama drugog reda.

3.3.3. Specifičnosti konvolucijskih mreža

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_{ij}) utječe na sve neurone u izlaznoj mapi. Zbog toga je prilikom računanja unazadne propagacije potrebno izračunati vrijednosti grešaka i promjena težina po svim neuronima mapa te sumirati njihov utjecaj.

Kod unazadne propagacije slojeva sažimanja 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.

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. Npr. ako imamo matricu mapiranja L:

$$L = \begin{bmatrix} (1,0) & (0,1) \\ (1,1) & (1,0) \end{bmatrix}$$

i neka je greška trenutnog sloja E':

$$E' = \begin{bmatrix} 0.54 & 0.39 \\ -0.34 & -0.88 \end{bmatrix}$$

Unazadnom propagacijom dobiva se greška prethodnog sloja koja iznosi:

$$E = \left[\begin{array}{cc|cc} 0 & 0 & 0 & 0.39 \\ 0.54 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ 0 & -0.34 & -0.88 & 0 \end{array}\right]$$

Kod slojeva sažimanja usrednjavanjem (MEAN-POOLING), 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. Neka je greška trenutnog sloja i dalje E':

$$E' = \begin{bmatrix} 0.54 & 0.39 \\ -0.34 & -0.88 \end{bmatrix}$$

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:

$$E = E' \otimes I \tag{3.10}$$

Za danu matricu E' konkretna matrica greške jest:

$$E = \begin{bmatrix} 0.54 & 0.39 \\ -0.34 & -0.88 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \left[\begin{array}{cc|cc} 0.54 & 0.54 & 0.39 & 0.39 \\ 0.54 & 0.54 & 0.39 & 0.39 \\ \hline -0.34 & -0.34 & -0.88 & -0.88 \\ -0.34 & -0.34 & -0.88 & -0.88 \end{array}\right]$$

4. Programska biblioteka Caffe

Caffe [1] je programska biblioteka i *framework* specijalizirana za duboke konvolucijske neuronske mreže. Napravio ju je Yangqing Jia tokom svog doktorskog studija na Berkeleyu. Izvorni kod je objavljen pod *BSD 2-Clause license* [2]. Glavne značajke ove bilioteke su definiranje strukture mreže kroz konfiguracijske datoteke te mogućnost treniranja mreža koristeći GPU.

Korištena platforma se sastoji od operacijskog sustava *Ubuntu 14.10* te prijenosnog računala *Dell Inspirion 15R* koje ima *NVIDIA GeForce GT 525M* grafičku karticu, te rezoluciju ekrana 1366×768.

4.1. Instalacija Caffe-a

U ovom poglavlju će biti opisan postupak instalacije Caffe-a za izvršavanje na procesoru računala (engl. *CPU*). Moguće je instalirati Caffe i za izvrsavanje na grafičkom procesoru (engl. *GPU*), ali je taj postupak složeniji te je nakon završenog postupka grafičko korisničko sučelje *Unity* (standardno sučelje operacijskog sustava Ubuntu) postalo neupotrebljivo te zbog navedenih razloga neće biti opisan ovaj postupak.

Prvo treba instalirati potrebne programe i programske pakete bez kojih se ne može instalirati niti pokrenuti Caffe[3].

```
$ sudo apt-get install git
$ mkdir src; cd src/
$ git clone https://github.com/BVLC/caffe.git
$ sudo apt-get install libatlas-base-dev libprotobuf-dev \
  libleveldb-dev libsnappy-dev libopencv-dev \
  libboost-all-dev libhdf5-serial-dev
$ sudo apt-get install libgflags-dev libgoogle-glog-dev \
```

[1] http://caffe.berkeleyvision.org/

[2] https://github.com/BVLC/caffe/blob/master/LICENSE

[3] Grana *master*; *commit*: e3c895baac64c7c2da4f30ab7ae35a1c84e63733

```
liblmdb-dev protobuf-compiler
```

Ovo je dosta ako se želi samo instalirati Caffe za CPU bez sučelja za druge jezike. Sljedeće pakete je potrebno instalirati ako se želi koristiti sučelje za programski jezik *Python*.

```
$ sudo apt-get install python-dev python-numpy \
  python-skimage python-protobuf ipython \
  ipython-notebook ipython-qtconsole
```

Postupak instalacije se svodi na kompajliranje izvornog koda.

```
$ cd caffe
$ cp Makefile.config.example Makefile.config
$ sed -i 's/# CPU_ONLY/CPU_ONLY/' Makefile.config
$ make -j4 all
$ make -j4 runtest
$ make -j4 pycaffe
$ export PYTHONPATH=$CAFFE_ROOT/python
```

Gdje je *CAFFE_ROOT* putanja do direktorija gdje se nalazi izvorni kod Caffe-a. Naredba je potrebna jer omogućava korištenje Caffe-a unutar Python ljuske. Preporučuje se korištenje *IPython*[4] interaktivne ljuske.

4.1.1. Dodatni parametar is_color

Tokom provođenja eksperimenata kao ulazni sloj koristio se IMAGE_DATA sloj (detaljnije objašnenom u podpoglavlju 4.4.2). Da bi se sloj moglo ispravno koristii, bilo je potrebno napraviti izmjenu u izvornom kodu koja omogućava Caffe-u da ispravno učitava sive slike (monokromatske, s jednim kanalom), jer ih inače učitava kao slike u boji.

Datoteke koje je bilo potrebno izmijeniti su:

- src/caffe/proto/caffe.proto
- src/caffe/layers/image_data_layer.cpp

Doduše, ovo nije potrebno raditi jer je moguće pretvoriti ulazni skup u jednu od baza koju Caffe može pročitati, a to su *LMDB*[5] ili *LevelDB*[6]. Ovo se radi s Caffe-ovim

[4] http://ipython.org/
[5] http://symas.com/mdb/
[6] https://github.com/google/leveldb

alatom *convert_imageset*. Međutim, bilo je lakše koristiti IMAGE_DATA sloj zbog prirode eksperimentiranja. Provodilo se testove s različitim izlaznim podatcima za iste ulazne podatke. Da se nije koristio IMAGE_DATA bilo bi potrebno raditi konverziju podataka nakon svake promjene izlaza.

4.2. Konfiguracija mreža

U Caffeu se mreže i sve vezano za treniranje istih definira u konfiguracijskim datotekama. Ono što ga čini iznimno fleksibilnim jest činjenica da se struktura mreže može definirati u jednoj datoteci, a parametri vezani za sam algoritam treniranja u drugoj datoteci. Također je moguće sve parametre o mreži i algoritmu treniranja zapisati u jednoj datoteci, ali na taj način gubimo fleksibilnost koju daje odvajanje tih dviju konfiguracija. Npr. možemo korisititi jednu konfiguraciju algoritma za treniranje na više različitih mreža i obrnuto. Tipovi konfiguracijskih datoteka se mogu podijeliti na sljedeće kategorije:

solver Služi za definiranje svih potrebnih parametara algoritma učenja mreže.

train_test Služi za definiranje strukture mreže. Mreža se definira sloj po sloj te se za svaki sloj definira kojeg je tipa i koliko neurona sadrži. Definiraju se svi potrebni parametri za treniranje mreže, koji nisu vezani uz konkretni algoritam učenja (takvi parametri idu u *solver*) kao npr. inicijalizacija težina. Također se definira koji će se podatci koristiti za treniranje, te koji za testiranje mreže.

deploy Konfiguracijska datoteka ovog tipa također služi za definiranje strukture mreže, no za razliku od *train_test* tipa ne sadrži nikakve parametre koji su bitni za treniranje mreže. Možemo reći da je ovo čista struktura mreže. Ova datoteka se koristi prilikom učitavanja naučene mreže u programima, npr. u Python skriptama pomoću Caffe-ovog sučelja za Python.

4.3. solver

Solver upravlja procesom optimizacije modela mreže upravljajući procesima unaprijedne i unazadne propagacije, tj. rezultata i greške, respektivno, koje rezultiraju promjenama vrijednosti parametara u pokušaju dobivanja manje greške. Caffe podržava 3 različita algoritma optimizacije:

- Stohastički gradijentni spust (*SGD*)

– Adaptivni gradijent (*ADAGRAD*)

– Nesterov akcelerirani gradijent (*NAG*)

Neki od bitnijih parametara su:

net Putanja do train_test datoteke s konfiguracijom mreže koja će se trenirati.

test_interval Broj iteracija između 2 testiranja.

test_iter Broj iteracija testova prilikom svakog testiranja.

display Broj iteracija između ispisivanja informacija o napretku treniranja.

max_iter Maksimalan broj iteracija treniranja.

snapshot Broj iteracija nakon kojeg se stanje treniranja sprema u snimak (engl. *snapshot*).

snapshot_prefix Prefiks imena snimka.

solver_mode Definira hoće li se treniranje izvršavati na CPU ili na GPU. Podrazumijevani parametar je GPU.

solver_type Algoritam učenja koji se koristi. Podrazumijevani algoritam je SGD.

Ovo su parametri koji su neovisni o tipu algoritma koji se koristi.

U nastavku su prikazani parametri algoritma SGD. Napomena, ako se koriste parametri nekog drugog algoritma, Caffe će dojaviti grešku o neispravnom parametru.

base_lr Početni koeficijent učenja.

lr_policy Politika mijenjanja koeficijenta učenja. String koji može biti "inv", "step".

momentum Vrijednost momentuma.

weight_decay Opadanje težina.

gamma Parametar računanja nove vrijednosti koeficijenta učenja.

power Parametar računanja nove vrijednosti koeficijenta učenja.

4.3.1. Stohastički gradijent u Caffe-u

Za skup podataka D funkcija greške je prosjek nad $|D|$ brojem podataka.

$$E(W) = \frac{1}{|D|} \sum_{i}^{|D|} f_W(X^{(i)}) + \lambda r(W) \tag{4.1}$$

Pošto je ovdje riječ o stohastičkom gradijentu, ne uzima se cijeli skup podataka za računanje greške, već se uzme N podataka iz podataka za učenje gdje je $N << |D|$.

$$E(W) \approx \frac{1}{N} \sum_{i}^{N} f_W(X^{(i)}) + \lambda r(W) \tag{4.2}$$

Model računa f_W u unaprijednom prolazu i gradijent ∇f_W u prolazu unazad. Nove težine W se računaju linearnom kombinacijom negativnog gradijenta $\nabla E(W)$ i prethodne promjene V_t. Moment μ je udio prethodne promjene.

$$V_{t+1} = \mu V_t - \alpha \nabla E(W_t) \tag{4.3}$$

$$W_{t+1} = W_t + V_{t+1} \tag{4.4}$$

4.4. train_test i deploy

Ove dvije konfiguracije definiraju strukturu mreže i jako su slične. U principu je *deploy* ista kao i *train_test*, jedino što fali u *deploy* su ulazni i izlazni slojevi, te parametri slojeva koji su bitni samo za treniranje mreža. U nastavku će biti opisani slojevi te njihovi parametri.

4.4.1. Slojevi

Za definiranje slojeva i parametara koristi se *Google Protocol Buffer*[7]. Svaki sloj se definira kao u sljedećem primjeru:

```
layers {
  name: "proizvoljno_ime_sloja"
  bottom: "ulaz_sloja"
  top: "izlaz_sloja"
  type: <TIP_SLOJA>
  tip_sloja_params {
```

[7] https://code.google.com/p/protobuf/

```
7       ...
8     }
9     # include: {phase: TRAIN | TEST}
10  }
```

Svaki sloj neće imati i ulaz i izlaz, primjerice ulazni slojevi će imati samo izlaz. Bitno je napomenuti da svaki tip sloja ima sebi specifične parametre. <TIP_SLOJA> može biti npr. IMAGE_DATA_LAYER, CONVOLUTION, TANH, itd. Znak # se interpretira kao komentar. Parametar koji je napisan u komentaru se koristi za definiranje faze u kojoj se sloj koristi, a to može biti ili faza treniranja ili faza testiranja. Ovaj parametar se koristi kod ulaznih slojeva.

4.4.2. Ulazni slojevi

Zadaća ulaznih slojeva jest da učitaju podatke za učenje i proslijede ih ostalim slojevima koji izvode konkretnu obradu. Caffe je sposoban pročitati podatke koji dolaze iz različitih izvora, poput efikasnih baza podataka *LevelDB* i *LMDB*, direktno iz memorije ili iz, kada efikasnost nije bitna, datoteka na disku u HDF5 formatu ili standardnim formatima za slike. Za svaki tip podataka postoji konkretan sloj koji ga čita. Bitno je napomenuti da je moguće definirati više ulaznih slojeva iz kojeg se čitaju podatci, te da je moguće definirati koji se podatci koriste za treniranje te koji za testiranje. U nastavku će svi biti navdedeni i opisani, dok će se sloj koji je korišten u ovom radu detaljno opisati.

DATA Za čitanje podataka iz baza podataka LEVELDB ili LMDB.

MEMORY_DATA Za čitanje podataka iz radne memorije.

DUMMY_DATA Korišten za razvoj i *debuggiranje*.

IMAGE_DATA Služi za učitavanje slika spremljenih u uobičajenim slikovnim formatima.

Parametri IMAGE_DATA sloja su:

source Ime datoteke u kojoj svaka linija sadrži ime datoteke i labelu klase kojoj slika pripada.

batch_size Broj slika koje se stavljaju u jednu grupu (engl. *batch*).

shuffle Hoće li se prethodno ispermutirati ulazni podatci. Podrazumijevana vrijednost jest *false*, tj. NE.

new_height Promijeni visinu slika prije treniranja.

new_width Promijeni širinu slika prije treniranja.

is_color Naknadno dodan parametar! Govori jesu li ulazne slike u boji ili ne, tj. govori kako će se slike učitati. Podrazumijevana vrijednost jest *false*, tj. NE.

4.4.3. Skriveni slojevi

CONVOLUTION je sloj koji obavlja konvoluciju.

num_output Broj filtera.

weight_filler Definira način inicijalizacije težina na početku treniranja.

pad Padding oko slike, isti u širini i visini.

pad_h Padding oko slike u visinu.

pad_w Padding oko slike u širinu

kernel_size Veličina jezgre. I širina i visina.

kernel_h Visina jezgre.

kernel_w Širina jezgre.

stride Jednak pomak jezgre po širini i po visini.

stride_h Pomak jezgre po visini.

stride_w Pomak jezgre po širini.

POOLING je sloj koji obavlja funkciju sažimanja.

pool Metoda sažimanja. Može biti MAX, AVE ili STOCHASTIC.

pad Padding oko slike, isti u širini i visini.

pad_h Padding oko slike u visinu.

pad_w Padding oko slike u širinu

kernel_size Veličina jezgre. I širina i visina.

kernel_h Visina jezgre.

kernel_w Širina jezgre.

stride Jednak pomak jezgre po širini i po visini.

stride_h Pomak jezgre po visini.

stride_w Pomak jezgre po širini.

INNER_PRODUCT je sloj koji predstavlja sloj poptuno povezanih običnih perceptrona s prethodnim slojem. Bitno je napomenuti da izlaz ovog sloja nije "provučen" kroz aktivacijsku funkciju.

num_output Broj neurona u ovom sloju.

weight_filler Definira način inicijalizacije težina na početku treniranja.

bias_filler Definira način inicijalizacije praga na početku treniranja.

Aktivacijski slojevi su slojevi koji imaju ulogu aktivacijskih funkcija. Imaju vezu jedan-na-jedan s prethodnim slojem i većina nema posebnih parametara. U nastavku su opisani samo neki od mogućih slojeva.

SIGMOID Računa sigmoidalnu funkciju ulaza.

TANH Računa tangens hiperbolni ulaza.

ABSVAL Računa apsolutnu vrijednost ulaza.

Ostali su RELU, POWER, BNLL.

4.4.4. Izlazni slojevi

Jedna posebnost izlaznih slojeva jest da primaju 2 ulaza. Jedan ulaz je izlaz prethodnog sloja, a drugi je traženi izlaz (engl. *ground truth*). Za klasifikacijske probleme se koriste SOFTMAX_LOSS i HINGE_LOSS slojevi, a za regresiju EUCLIDEAN_LOSS.

SOFTMAX_LOSS Računa multinomijalnu logističku funkciju od softmaxa ulaza.

HINGE_LOSS Računa jedan-protiv-svih *hinge* ili kvadratni *hinge* gubitak. Ovaj sloj ima svoj poseban parametar `norm` koji može biti L1 ili L2.

EUCLIDEAN_LOSS Ralčuna Euklidsku udaljenost između dobivenog i traženog ulaza.

5. Pomoćni alati

U ovom poglavlju će biti opisani programski alati napravljeni za potrebe ovog rada koji služe za prikupljanje i pripremu podataka.

5.1. Mapiranje slika s točkama na ekranu

Pošto na Internetu nije pronađena slobodno dostupna baza slika s mapiranim podatcima o smjeru pogleda, napravljen je ovaj program. Njegova funkcija je izgradnja baze slika lica s mapiranima podatcima o smjeru pogleda, konkretno o točci na ekranu gdje osoba gleda.

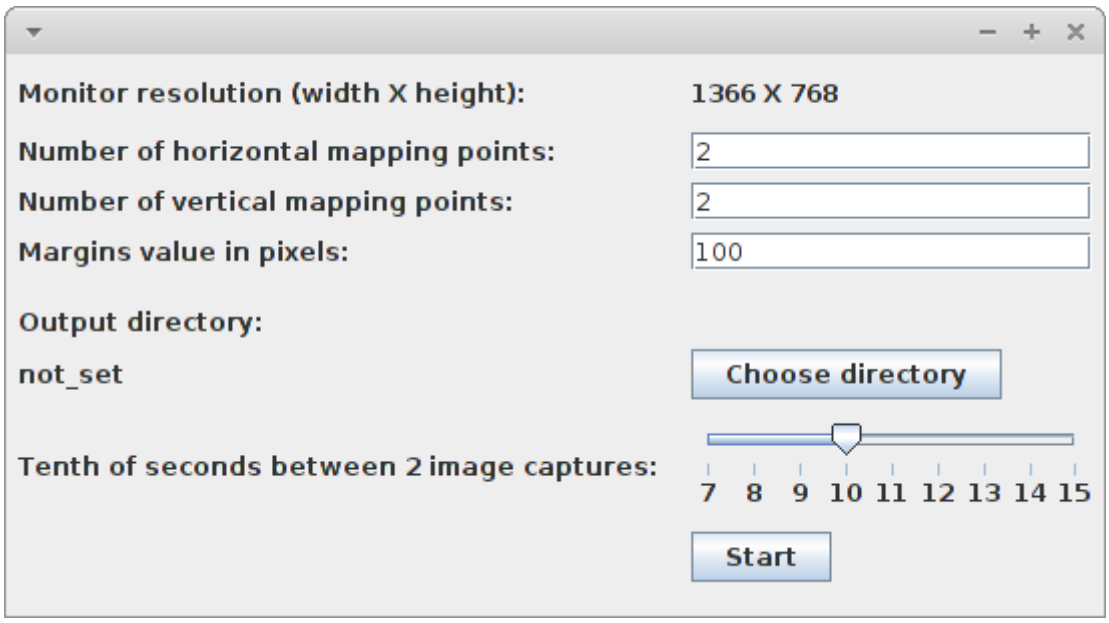
Radi na način da se prvo definira broj točaka po visini i širini u koje će korisnik gledati, kao i margina, odabere se direktorij u kojem će se stvoriti direktorij sesije u kojem će biti snimljene slike i datoteka s informacijama o smjerovima pogleda, te vrijeme koje je potrebno da se točka u koju korisnik gleda "preseli" s jednog mjesta na drugo. Smisao margine jest da se za isti broj vertikalnih i horiznotalnih točaka mapiraju drugi podatci, inače bi se uvijek uzimale slike za iste koordinate točaka na ekranu. Korisnik gleda u svaku točku, te se pritom snimi slika. Podatci o imenu slike i koordinatama točke se spreme u posebnu datoteku koja se koristi u dajlnjim fazama. Korištenje programa prikazano je na slikama 5.1 i 5.2.

Za snimanje slika korištena je biblioteka *webcam-capture*[1]. Slike su snimane u VGA rezoluciji 640×480 i spremane su u *PNG* formatu. Prikupljanje slika je odvijano u više sesija. Slike su prikupljane integriranom web-kamerom prijenosnog računala. Tokom prikupljanja slika nisu korištene nikakve dodatne naprave koje bi fiksirale položaj glave, ali se nastojalo da se tijelo i glava ne pomiču previše.

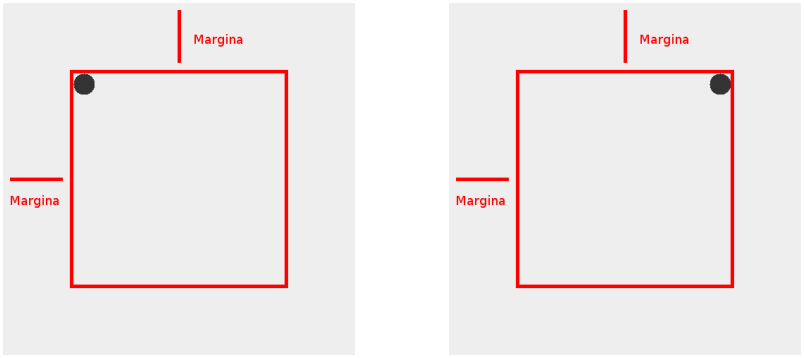
Program se koristi tako da se u konzoli pozicionira u direktroij koji sadrži datoteku *GazeMapper.jar*. Da bi se pokrenulo program upiše:

```
$ java -jar GazeMapper.jar
```

[1] http://webcam-capture.sarxos.pl/



Slika 5.1: Konfiguracijski prozor



(a) Inicijalna pozicija. **(b)** Pomaknuta točka gledanja.

Slika 5.2: Primjer rada *GazeMapper* programa. Slike su umanjena verzija cijelog ekrana. Crveni pravokutnici, linije i tekst su naknadno dodani i ne prikazuju se u programu. Ovdje se samo želi prikazati princip rada programa.

Upisivanjem ove naredbe prikazuje se konfiguracijski prozor kao na slici 5.1

Format zapisa podataka u datoteku je sljedeći. Svaki redak predstavlja jedan uređeni par (*ime_slike*, *koordinate_točke*). Prvih nekoliko redaka su samo informativne naravi, i započinju znakom "#", koji se zanemaruju (tj. interpretiraju se kao komentari).

```
### INFO
# Screen resolution:          1366 X 768
# Mapping points resolution: 2 X 2
# Dir: /path/to/output/dir
1422804065635.png 115 115
1422804066922.png 1251 115
1422804068100.png 115 653
1422804069231.png 1251 653
```

5.2. Stvaranje uzoraka za neuronsku mrežu

Ovaj program iz prethodno prikupljenih slika priprema uzorke za učenje. Nad svakom slikom dobivenom prethodnim programom se prvo vrši detekcija lica i unutar pronađenog lica se vrši detekcija očiju. Pretpostavka je da se na svakoj slici nalazi samo jedno lice s 2 oka, te se svaka slika na kojoj nije detektirano točno jedno lice s 2 oka zanemaruje.

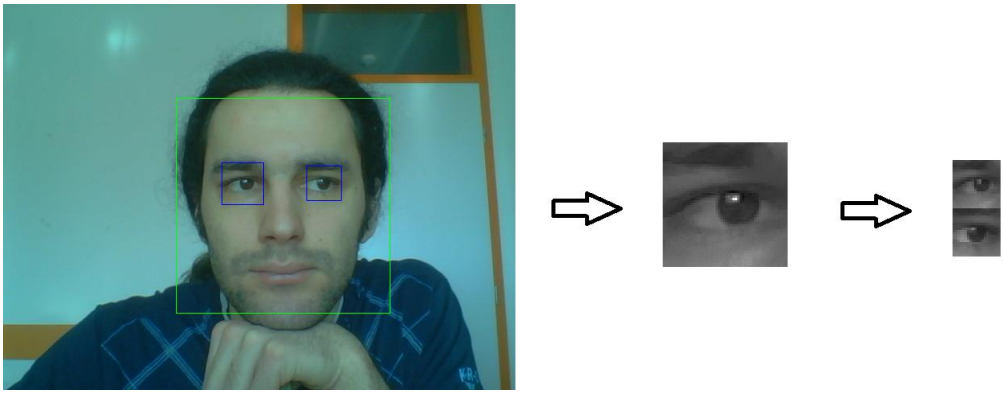
Za detekciju očiju korištena je gotova implementacija u biblioteci OpenCV[2]. Oči se detektiraju u 2 faze, prvo se detektira lice te potom unutar lica se detektiraju oči. Algoritam za detekciju se bazira na metodi Viole i Jonesa, tj. na kaskadi ojačanih Haarovih klasifikatora. OpenCV osim algoritama za detekciju sadrži i naučene kaskade za detekciju, između ostalih, lica i oka. Korištene su:

Za detekciju lica lbpcascade_frontalface.xml

Za detekciju oka haarcascade_eye_tree_eyeglasses.xml

Prvo se na svakoj slici segmentiraju oči, te se ti segmenti skaliraju na zadanu veličinu kvadrata i spajaju u jednu sliku dimenzija $velicina \times 2 \cdot velicina$. ***velicina*** je parametar koji se daje programu kroz naredbenu liniju. Podatci o imenu ovako segmentirane slike, koordinatama točke na ekranu, koordinatama pozicija gornje lijeve točke u kojima su detektirane oči te veličini i širini pojedinog detektiranog oka se zapisuju u datoteku. Ta datoteka se koristi za učenje. Također se podatci spremaju i u JSON

[2]http://opencv.org/. Korištena je verzija 2.4.9



Slika 5.3: Princip rada programa *CreateSamples*

(JavaScript Object Notation) datoteku koja omogućava jednostavnu serijalizaciju u nativne objetke jezika pomoću biblioteke za deserijalizaciju iz JSON-a. Ulazne slike su u boji dok se izrađeni uzorci konvertiraju u crno-bijele slike. Za ispravan rad programa potrebno je prethodno instalirati OpenCV, te OpenCV-jeve Java *bindinge*. Program se poziva iz konzole i to iz direktorija u kojem se nalazi datoteka *CreateSamples.jar*.

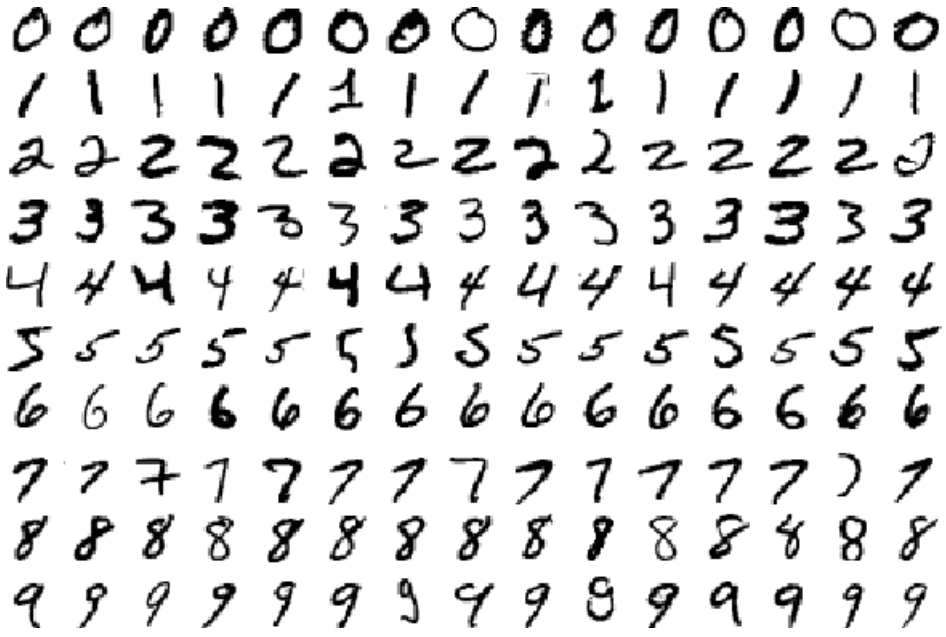
```
$ java -Djava.library.path=/usr/lib/jni/ -jar CreateSamples.jar \
  <path/to/input/dir/> <path/to/output/dir> <sample-size>
```

Valja napomenuti da su rađeni eksperimenti u kojima su korišteni svi podatci o pozicijama i veličini i širini pojedinog detektiranog oka, kao i eksperimenti u kojima su korišteni samo podatci o točci u koju subjekt gleda.

6. Ispitni skupovi

6.1. MNIST

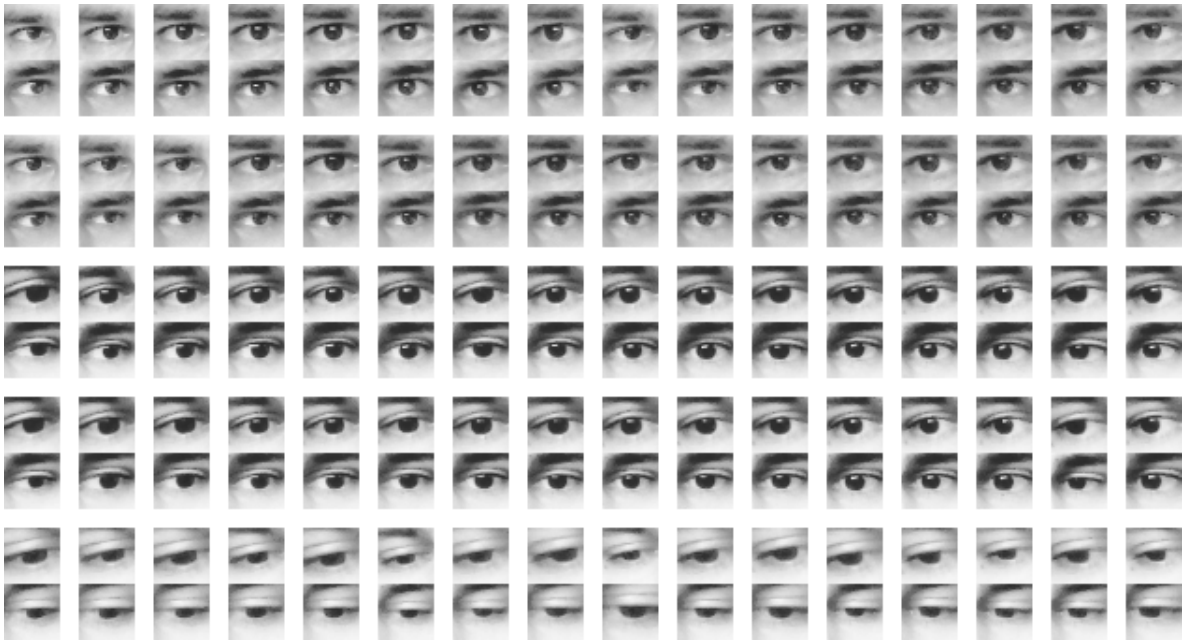
Skup MNIST (engl. *Mixed National Institute of Standards and Technology*) [3] sadrži 10 klasa rukom pisanih brojeva (od nule do devet). Nekoliko takvih znakova prikazano je na slici 6.1. Takav se skup najviše koristio za testiranje ranih sustava automatskog prepoznavanja rukom pisanih brojki kod bankovnih čekova. Slike su monokromatske (8 bitne) te veličine 28×28 točki. Skup je nadalje podijeljen u skup za treniranje koji sadrži 60 000 znakova i skup za testiranje koji sadrži 10 000 znakova.



Slika 6.1: Primjer znakova iz MNIST baze podataka.

6.2. Vlastiti skup podataka

Za potrebe ovog rada pripremljen je vlastiti skup podataka pošto nije pronađen nijedna slobodno dostupna baza slika s označenim smjerom gledanja. Slike su prikupljane



Slika 6.2: Primjer uzoraka iz vlastite baze podataka.

kako je opisano u poglavlju 5.1.

Prikupljeno je sve skupa 5421 slika koje su podijeljene u skup za treniranje od 4700 slika, te skup za testiranje od 721 slike. Slike su monokromatske, rezolucije 30×60. Nekoliko uzoraka je prikazano na slici 6.2

7. Eksperimentalni rezultati

7.1. Ispitni skup MNIST

Mreža je od svih primjera na skupu za treniranje točno klasificirala 99.51% primjera, a na skupu za testiranje taj postotak iznosi 99%. Vidimo matricu zabune za ovu mrežu dobivenu na skupu za testiranje 7.1.

Tablica 7.1: Matrica zabune za detekciju znamenaka

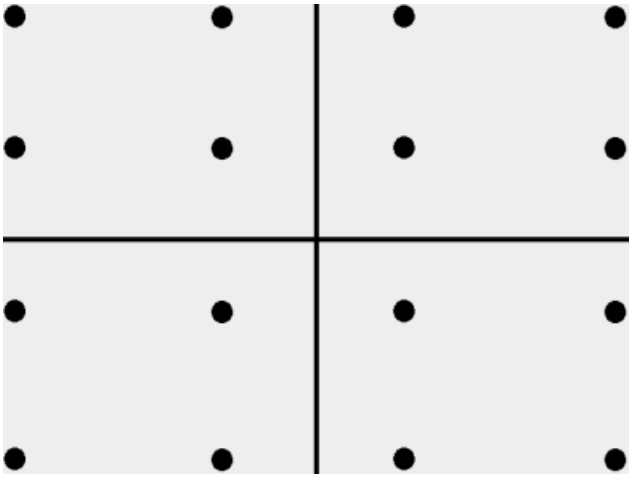
		Predviđena znamenka									
		0	1	2	3	4	5	6	7	8	9
Stvarna znamenka	0	**975**	0	1	0	0	0	2	1	1	0
	1	0	**1131**	1	1	0	1	1	0	0	0
	2	2	1	**1026**	0	0	0	0	2	1	0
	3	1	0	2	**999**	0	4	0	4	0	0
	4	0	0	2	0	**975**	0	1	0	0	4
	5	2	0	0	4	0	**883**	1	1	1	0
	6	5	2	2	0	2	6	**940**	0	1	0
	7	0	2	6	0	0	0	0	**1016**	1	3
	8	2	0	2	2	0	2	1	3	**958**	4
	9	0	0	0	0	5	4	0	2	1	**997**

7.2. Vlastiti ispitni skup

Pošto je intuitivno pretpostaviti da će mreža jako teško naučiti točne koordinate iz slika, odlučeno je problem pojednostaviti i provjeriti kako se mreža odnosi s lakšim problemima.

Naime, skup podataka je mapiran s točnim koordinatama točke na ekranu, no uz poznatu rezoluciju ekrana lako je remapirati te točke u neke druge oznake. Naprav-

ljeno je sljedeće. Postojeće točke bi se podijelile u *ćelije* ekrana $x \times y$ gdje je x broj redaka, a y broj stupaca. Ovaj postupkak ilustriran je na slici 7.1. Dakle, za svaku pos-



Slika 7.1: Podjela točaka u *ćelije*.

tojeću točku bi se odredila ćelija kojoj ta točka pripada i to bi postala nova koordinata točke. Ovom pretvorbom smo praktički dobili problem klasifikacije, gdje je ukupan broj klasa koje treba klasificirati $x \times y$. Za svaki problem su trenirane po 2 mreže, jedna koja uči horizontalnu koordinatu točke i jedna koja uči vertikalnu koordinatu točke. Napravljeni su testovi s točkama podijeljenim u ćelije rezolucije 2×2, 5×5, 10×10 te se pokušalo regresijom naučiti i točne koordinate.

Tokom eksperimentiranja korištene su različite arhitekture mreža na različitim problemima, no u nastavku će biti objašnjeni dobiveni rezultati s mrežom prikazanoj na slici 7.2. Mreza ima sljedece slojeve:

1. konvolucijski sloj "conv1" Ima 20 neurona s jezgrama veličine 5 piksela.

1. sloj sazimanja "pool1" MAX-POOLING sloj s jezgrom veličine 2 piksela i korakom od 2 piksela.

2. konvolucijski sloj "conv2" Ima 50 neurona s jezgrama veličine 5 piksela.

2. sloj sazimanja "pool2" MAX-POOLING sloj s jezgrom veličine 2 piksela i korakom od 2 piksela.

1. poptuno povezani sloj "ip1" Sadrži 500 neurona.

Sigmoidalna aktivacijska funkcija "sig"

2. poptuno povezani sloj "ip2" Ima neurona ovisno o problemu (2, 5, 10 ili 1).

U ovim eksperimentima su korišteni samo podatci o slici i koordinatama točke u koju osoba gleda, a ne dodatne informacije o poziciji i veličini detektiranih očiju.

7.2.1. Rezolucija 2x2

Mreža koja određuje stupac u koji se gleda je od svih primjera na skupu za treniranje točno klasificirala 99.87% primjera, a na skupu za testiranje taj postotak iznosi 96.67%. Vidimo matricu zabune za ovu mrežu dobivenu na skupu za testiranje u tablici 7.2.

Tablica 7.2: Matrica zabune za 2X mrežu nad skupom za testiranje

		Predviđeni redak	
		0	1
Stvarni	0	**386**	15
	1	9	**311**

Mreža koja određuje redak u koji se gleda je od svih primjera na skupu za treniranje točno klasificirala 99.04% primjera, a na skupu za testiranje taj postotak iznosi 97.92%. Vidimo matricu zabune za ovu mrežu dobivenu na skupu za testiranje u tablici 7.3.

Tablica 7.3: Matrica zabune za 2Y mrežu nad skupom za testiranje

		Predviđeni stupac	
		0	1
Stvarni	0	**378**	5
	1	10	**328**

7.2.2. Rezolucija 5x5

Mreža koja određuje stupac u koji se gleda je od svih primjera na skupu za treniranje točno klasificirala 97.64% primjera, a na skupu za testiranje taj postotak iznosi 90.98%. Vidimo matricu zabune za ovu mrežu dobivenu na skupu za testiranje u tablici 7.4.

Mreža koja određuje redak u koji se gleda je od svih primjera na skupu za treniranje točno klasificirala 94.60% primjera, a na skupu za testiranje taj postotak iznosi 87.79%. Vidimo matricu zabune za ovu mrežu dobivenu na skupu za testiranje u tablici 7.5.

Tablica 7.4: Matrica zabune za 5X mrežu nad skupom za testiranje

		Predviđeni redak				
		0	1	2	3	4
Stvarni redak	0	**114**	8	0	0	2
	1	14	**173**	7	0	0
	2	0	14	**123**	2	0
	3	0	0	7	**138**	2
	4	0	0	0	9	**108**

Tablica 7.5: Matrica zabune za 5Y mrežu nad skupom za testiranje

		Predviđeni stupac				
		0	1	2	3	4
Stvarni stupac	0	**129**	11	0	0	0
	1	8	**133**	4	0	0
	2	0	18	**149**	6	1
	3	1	3	10	**125**	13
	4	0	0	0	13	**97**

7.2.3. Rezolucija 10x10

Mreža koja određuje stupac u koji se gleda je od svih primjera na skupu za treniranje točno klasificirala 85.17% primjera, a na skupu za testiranje taj postotak iznosi 74.06%. Vidimo matricu zabune za ovu mrežu dobivenu na skupu za testiranje u tablici 7.6.

Mreža koja određuje redak u koji se gleda je od svih primjera na skupu za treniranje točno klasificirala 84.60% primjera, a na skupu za testiranje taj postotak iznosi 75.31%. Vidimo matricu zabune za ovu mrežu dobivenu na skupu za testiranje u tablici 7.7.

7.2.4. Originalna rezolucija

Regresija u Caffe-u se pokazala jako teškim zadatkom. Za ovaj problem mreža nikako nije mogla naučiti točne koordinate, čak niti približno. Mreža bi otprilike naučila pravac, tj. za svaki ulaz bi dobili sličan izlaz iz jako malog intervala.

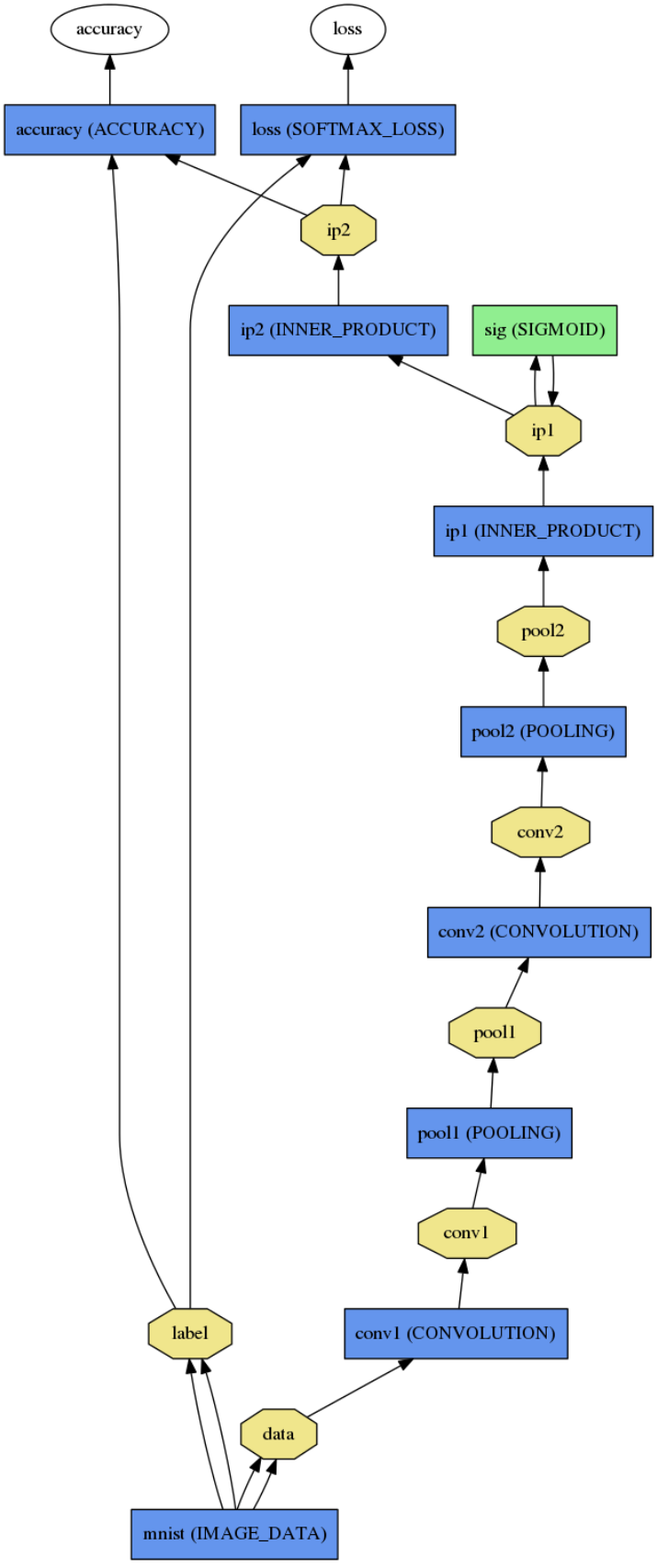
Na temelju dobivenih rezultata zaključujemo da se mreža može koristiti za grubu estimaciju točke u koju korisnik gleda. Naravno, to može biti dovoljno dobro ili jako loše uzimajući u obzir prirodu mogućih konkretnih primjena.

Tablica 7.6: Matrica zabune za 10X mrežu

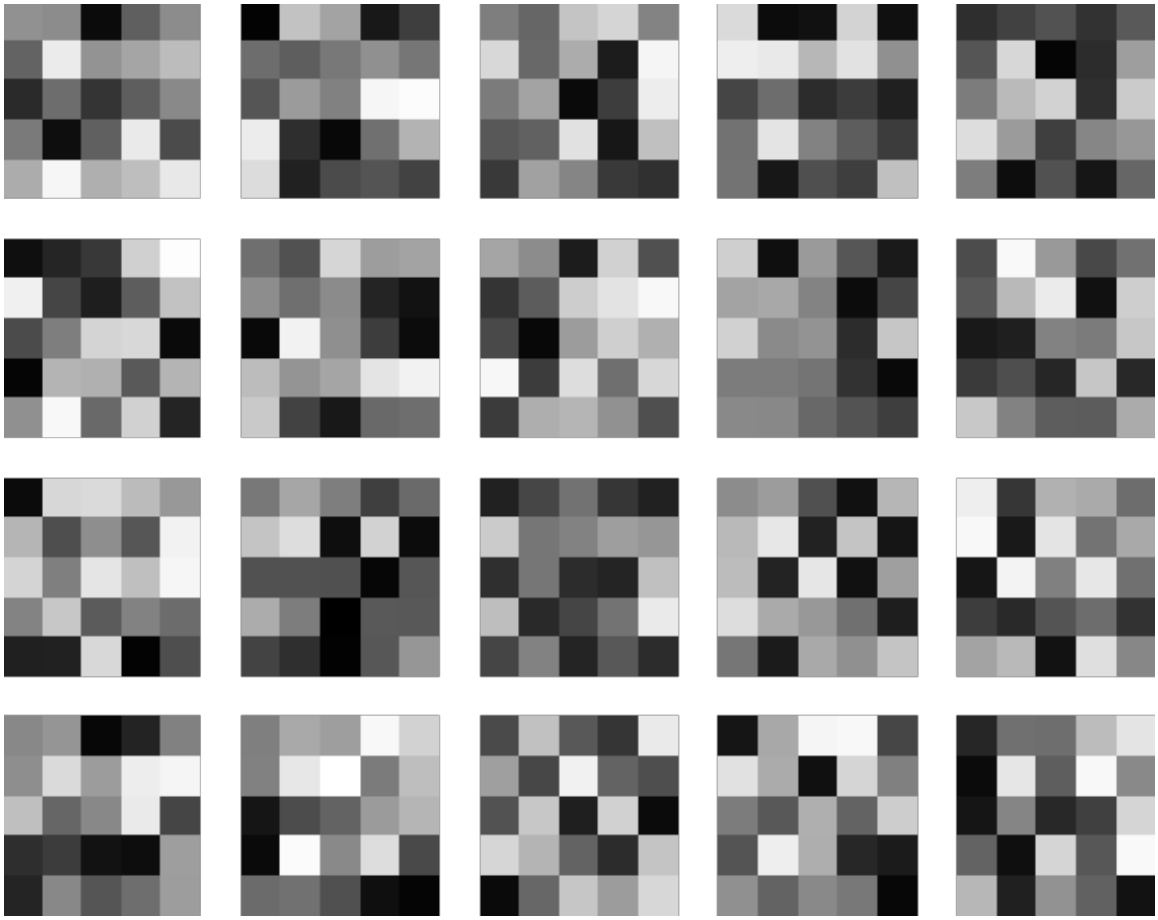
		Predviđeni redak									
		0	1	2	3	4	5	6	7	8	9
Stvarni redak	0	**51**	10	3	0	0	0	0	0	0	2
	1	12	**30**	16	0	0	0	0	0	0	0
	2	1	5	**74**	10	0	0	0	0	0	0
	3	0	0	13	**84**	7	0	0	0	0	0
	4	0	0	0	22	**55**	6	0	0	0	0
	5	0	0	0	0	11	**37**	5	0	0	0
	6	0	0	0	0	0	8	**57**	9	0	0
	7	0	0	0	0	0	0	9	**57**	9	1
	8	0	0	0	0	0	0	0	6	**33**	18
	9	0	0	0	0	0	0	0	1	3	**56**

Tablica 7.7: Matrica zabune za 10Y mrežu

		Predviđeni redak									
		0	1	2	3	4	5	6	7	8	9
Stvarni redak	0	**66**	3	2	0	0	0	0	0	0	0
	1	13	**44**	6	2	0	0	0	0	0	0
	2	2	6	**43**	12	0	0	0	0	0	0
	3	0	0	5	**75**	6	0	0	0	0	0
	4	0	0	0	11	**72**	12	0	0	0	0
	5	0	0	0	1	9	**62**	6	0	0	1
	6	0	0	0	1	1	5	**70**	1	1	0
	7	0	0	0	0	2	1	17	**36**	9	4
	8	0	0	0	0	0	0	1	13	**24**	10
	9	0	0	0	0	0	0	1	4	10	**51**



Slika 7.2: Struktura mreže koja je trenirana. Konfiguracija ove mreže je dana u dodatku A.



Slika 7.3: Naučeni filteri 1. konvolucijskog sloja mreže 5X.

8. Zaključak

Cilj ovog rada bio je istražiti mogućnost određivanja smjera pogleda koristeći konvolucijske neuronske mreže. Pritom je korištena programska biblioteka Caffe koja je specijalizirana za treniranje konvolucijskih mreža. Bilo je potrebno proučiti kako se koristi biblioteka, prikupiti i obraditi podatke za učenje te provesti postupak učenja mreža i na kraju opisati dobivene rezultate.

Za prikupljanje podataka bilo je potrebno razviti vlastite alate, jedan od kojih snima osobu koja prati gibajuću točku na ekranu i mapira snimljene slike s kooridnatom te točke, te alat koji iz tako dobivenih slika stvara odgovorajuće uzorke koji se koriste za treniranje mreža. Pošto su slike monokromatske (tj. crno-bijele) bilo je potrebno napraviti izmjenu u izvornom kodu Caffe-a da bi se takve slike moglo ispravno učitati u memoriju.

Biblioteka Caffe je prvo testirana na skupu rukom pisanih znamenki MNIST. Pritom je korištena dobro znana struktura mreže za ovaj tip problema, tzv. LeNet mreža. Mreža je ispravno klasificirala 99% uzoraka na skupu za testiranje. To je očekivan rezultat pošto konvolucijske mreže postižu dosad jedne od najboljih rezultat na ovom skupu podataka [3].

Nad vlastitim skupom podataka rađeni su eksperimenti s različitim rezolucijama izlaza da bi se utvrdilo za koliko kompleksne slučajeve je mreža sposobna naučiti izlaz, a za koje ne. Izlazi su bili podijeljeni u ćelije 2×2, 5×5, 10×10, te se pokušalo s regresijom naučiti stvarne koordinate. Kao što je bilo očekivano, što je veća rezolucija to točnost opada, a u slučaju regresije mreža bi dala približno jednak odziv bez obzira na ulaz. Također je primjećeno da mreže koje određuju stupac u koji se gleda postižu veću točnost od onih koje određuju redak. Razlog tomu vjerojatno leži u činjenici što postoje veće vidljive razlike na oku kada gleda lijevo-desno, nego kada gleda gore-dolje. Kod pomaka gore-dolje postoje suptilne razlike u položaju vjeđa, dok se kod pomaka lijevo-desno mijenja odnos udjela bjeloočnica sa svake strane oka.

U daljnjem radu bilo bi poželjno istražiti kako na točnost utječu dodatne informacije poput npr. pozicije detektiranog oka u slici te veličine detektiranog oka. Takve

informacije bi mogle povećati robusnost sustava.

Na temelju dobivenih rezultata zaključujemo da su konvolucijske mreže (konkretnije Caffe) jako pogodne za rješavanje problema klasifikacije dok za probleme regresije ne pokazuju dobre rezultate, bar ne u problemu određivanja smjera pogleda.

Određivanje smjera gledanja konvolucijskim neuronskim mrežama

Sažetak

U ovom radu napravljen je kratak pregled neuronskih mreža s naglaskom na konvolucijske neuronske mreže. Cilj rada bio je napraviti sustav koji iz datog skupa slika osoba koje gledaju u ekran nauči u koju točku na ekranu osoba gleda koristeći konvolucijske neuronske mreže. Korištena je gotova biblioteka za treniranje konvolucijskih neuronskih mreža Caffe, opisan je način njenog korištenja, te su implementirani vlastiti alati za prikupljanje vlastitog skupa za učenje. Na kraju su opisani dobiveni rezultati.

Ključne riječi: računalni vid, neuron, umjetne neuronske mreže, konvolucijske neuronske mreže, Caffe, smjer pogleda, strojno učenje, gradijentni spust, MNIST, klasifikacija

Determining gaze direction using convolutional neural networks

Abstract

This paper made a brief overview of neural networks with a focus on convolutional neural networks. The aim was to create a system that, from a given set of images of people who look at the screen, learns where on the screen people are looking at. The system is based on convolutional neural networks. Framework for training and deploying convolutional neural networks Caffe was used. This paper describes how it is used. In order to obtain training and testing data additional tools were implemented. Finally, the obtained results from experiments were discussed.

Keywords: computer vision, neuron, artificial neural network, convolutional neural network, Caffe, gaze direction, machine learning, gradient descent, MNIST, classification

LITERATURA

[1] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, i Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. URL `http://caffe.berkeleyvision.org/`.

[2] Mirko Jurić-Kavelj. Određivanje smjera pogleda pomoću neuronske mreže. Technical report, Fakultet Elektrotehnike i Računarstva, 2013.

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

[4] Vedran Vukotić. Raspoznavanje objekata dubokim neuronskim mrežama. Magistarski rad, 2014.

[5] Wikipedia. Kernel (image processing), 2015. URL `http://en.wikipedia.org/wiki/Kernel_(image_processing)#Convolution`.

[6] Wikipedia. Hadamard product (matrices), 2015. URL `http://en.wikipedia.org/wiki/Hadamard_product_(matrices)`.

[7] Marko Čupić, Bojana Dalbelo Bašić, i Marin Golub. *Neizrazito, evolucijsko i neuroračunarstvo.* 2013.

Dodatak A
Primjeri konfiguracijskih datoteka

Treniranje se pokreće sljedećom naredbom:

```
$ $CAFFE_ROOT/build/tools/caffe train -solver \
  nets/10x10-lenet-1D-output/proto-confs/lenet_solver_X.prototxt
```

Gdje je $CAFFE_ROOT putanja do direktorija gdje se nalazi izvorni kod Caffe-a. Program se pokreće iz direktorija u kojem se nalaze direktoriji *nets/* i *images/*.

SOLVER

```
net: "nets/10x10-lenet-1D-output/proto-confs/
   lenet_train_test_X.prototxt"

test_iter: 1
test_interval: 500

base_lr: 0.0001
momentum: 0.9
weight_decay: 0.0005
lr_policy: "inv"
gamma: 0.0001
power: 0.75

display: 100
max_iter: 10000
snapshot: 1000

```

```
snapshot_prefix: "nets/10x10-lenet-1D-output/snapshots/snap-X"
solver_mode: GPU
```

TRAIN_TEST

```
name: "LeNet"
layers {
  name: "input"
  type: IMAGE_DATA
  top: "data"
  top: "label"
  image_data_param {
    source: "images/train/train-10-X.txt"
    batch_size: 1000
    shuffle: true
  }
  include: { phase: TRAIN }
}
layers {
  name: "input"
  type: IMAGE_DATA
  top: "data"
  top: "label"
  image_data_param {
    source: "images/val/val-10-X.txt"
    batch_size: 25
    shuffle: true
  }
  include: { phase: TEST }
}

layers {
  name: "conv1"
  type: CONVOLUTION
  bottom: "data"
  top: "conv1"
```

```
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "pool1"
  type: POOLING
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layers {
  name: "conv2"
  type: CONVOLUTION
  bottom: "pool1"
  top: "conv2"
  convolution_param {
    num_output: 50
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
```

```
      type: "constant"
    }
  }
}
layers {
  name: "pool2"
  type: POOLING
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layers {
  name: "ip1"
  type: INNER_PRODUCT
  bottom: "pool2"
  top: "ip1"
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "sig"
  type: SIGMOID
  bottom: "ip1"
  top: "ip1"
}
```

```
layers {
  name: "ip2"
  type: INNER_PRODUCT
  bottom: "ip1"
  top: "ip2"
  inner_product_param {
    num_output: 10
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "accuracy"
  type: ACCURACY
  bottom: "ip2"
  bottom: "label"
  top: "accuracy"
  include: { phase: TEST }
}
layers {
  name: "loss"
  type: SOFTMAX_LOSS
  bottom: "ip2"
  bottom: "label"
  top: "loss"
}
```

DEPLOY

```
name: "10x10-LeNet-X"
input: "data"
input_dim: 1
input_dim: 1
```

```
input_dim: 40
input_dim: 20

layers {
  name: "conv1"
  type: CONVOLUTION
  bottom: "data"
  top: "conv1"
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
  }
}
layers {
  name: "pool1"
  type: POOLING
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layers {
  name: "conv2"
  type: CONVOLUTION
  bottom: "pool1"
  top: "conv2"
  convolution_param {
    num_output: 50
    kernel_size: 5
    stride: 1
  }
}
```

```
layers {
  name: "pool2"
  type: POOLING
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layers {
  name: "ip1"
  type: INNER_PRODUCT
  bottom: "pool2"
  top: "ip1"
  inner_product_param {
    num_output: 500
  }
}
layers {
  name: "sig"
  type: SIGMOID
  bottom: "ip1"
  top: "ip1"
}
layers {
  name: "ip2"
  type: INNER_PRODUCT
  bottom: "ip1"
  top: "ip2"
  inner_product_param {
    num_output: 10
  }
}
layers {
```

```
  name: "prob"
  type: SOFTMAX
  bottom: "ip2"
  top: "prob"
}
```