SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1931

Strojno očitanje natpisa na prometnim znakovima

Ivan Krešo

Zagreb, srpanj 2011.

Sadržaj

1. Uvod **1**

2. Korišteni algoritmi, metode i izvorne slike **3**

2.1. Binariziranje slika 3

2.2. Analiza povezanih komponenata 5

2.3. Filtriranje lažnih pozitivnih detekcija 6

2.4. Obrada segmentiranih slika 8

2.5. Neuronska mreža 8

2.5.1. Algoritam s propagiranjem greške unatrag 12

3. Programska izvedba **15**

3.1. Radni okvir Qt 15

3.2. OpenCV 15

3.3. Generiranje skupa segmentiranih slova 16

3.4. Implementacija neuronske mreže 26

3.4.1. Biblioteka FANN 26

3.4.2. Učenje neuronske mreže i klasifikacija 29

4. Eksperimentalni rezultati **32**

4.1. Detekcija i segmentacija teksta 32

4.2. Algoritmi učenja i parametri mreže 32

4.2.1. Distibucija slika i uspješnosti po znakovima 35

4.2.2. Matrica zabune 36

4.2.3. Primjeri pogrešnih klasifikacija 38

5. Zaključak **39**

Literatura **40**

1. Uvod

Računalni vid je područje umjetne inteligencije koje se bavi izvlačenjem informacija iz slika i video sekvenci kako bi obradom tih informacija računala mogla riješiti neki problem ili možda *razumjeti* scenu u širem ili ograničenom smislu. Računalni vid je, na neki način, inverzija područja računalne grafike. Dok je cilj računalne grafike stvoriti sliku iz 3D modela, računalni vid će htjeti stvoriti 3D model iz slike. Česti problemi koje ovo područje obuhvaća su: rekonstrukcija slike, detekcija i klasifikacija objekata i događaja, praćenje pokreta u sceni, strojno učenje, procjena kretanja, restauracija slike. Detekcija teksta i klasifikacija strojnim učenjem su problemi koji će ovaj rad obuhvatiti. Pritom je važno reći da se detekcija i klasifikacija bitno razlikuju. Zadaća detekcije je samo otkriti da li je objekt uopće prisutan na slici i gdje. Nakon toga dolazi klasifikacija sa dovoljnim znanjem o domeni da prepozna i kaže nam o kojem se objektu radi.

Dok je čovjekovo osjetilo vida vrlo razvijeno i sve probleme detekcije i klasifikacije rješava tako *intuitivno* da nam se niti ne čini da obavljamo neki posao, računalima koja su vrlo osjetljiva na nepreciznost i nepotpunost informacija *gledanje* još uvijek predstavlja velik problem. Razlog tomu djelomično leži u činjenici da je računalni vid relativno novo područje istraživanja koje je započelo pojavom bržih računala 70-ih godina. Zbog širokog raspona problema koje obuhvaća, istraživanje u računalnom vidu često je prožeto interdisciplinarnošću. Iz istog razloga ne postoji ni standardna formulacija koja bi obuhvatila sve probleme računalnog vida. Umjesto toga postoji mnoštvo metoda za rješavanje dobro definiranih problema.

Svjesni smo da je promet u kojem se svakodnevno krećemo još uvijek poprilično opasno okruženje. Automatiziranjem radnji u prometu možemo postupno smanjivati mogućnost ljudske pogreške i time povećati sigurnost za sve sudionike. U sklopu ovog rada obuhvaćen je razvoj postupka za strojno očitanje natpisa na prometnim znakovima za koje pretpostavljamo da su prethodno detektirani nekom drugom metodom. Rješenje problema je važno u sklopu šireg područja rada

raspoznavanja prometnih znakova iz perspektive vozača.

Odabrani put do rješenja problema može se podijeliti u nekoliko koraka. Najprije je potrebno razviti postupke pretprocesiranja slike koji će ju učiniti pogodnom za idući korak detekcije teksta. Najvažniji korak u pretprocesiranju je binarizacija slike. Detekcija teksta riješena je korištenjem algoritma *Connected component labeling* koji radi sa prethodno binariziranom slikom. U radu sa slikama bit će korištena biblioteka *OpenCV* koja osim što pruža sučelje za osnovne operacije poput dohvata i postavljanja vrijednosti piksela, sadrži i velik broj algoritama često korištenih u području računalnog vida. Idući korak nakon detekcije slova je njihova klasifikacija. Klasifikacija će biti ostvarena u okviru strojnog učenja izgradnjom neuronske mreže. Za izgradnju mreže koristit će se biblioteka FANN (engl. *Fast Artificial Neural Network*) čija implementacija u programskom jeziku C pruža neke napredne mogućnosti i odlične performanse. Čitav postupak će biti enkapsuliran u dva programa napisana u jeziku C++ s grafičkim sučeljima izgrađenima pomoću radnog okvira *Qt*. Prvi program obuhvatit će korake pretprocesiranja i detekcije u svrhu prikupljanja uzoraka segmentiranih slova. Drugi program će implementirati korištenje neuronske mreže koja će prethodno biti naučena u zasebnim komponentama bez grafičkog sučelja. Na samom kraju pogledat ćemo dobivene eksperimentalne rezultate.

2. Korišteni algoritmi, metode i izvorne slike

2.1. Binariziranje slika

Kako bi se iz njih mogao izvući tekst, slike je najprije potrebno binarizirati. Postupak binarizacije slika sastoji se od dva koraka. U prvom koraku potrebno je dobiti sliku u sivim tonovima (engl. *grayscale*). Vrijednost sive komponente izračuna se iz vrijednosti triju RGB komponenti s time da se uzimaju različiti udjeli komponenti kako bi se osvjetljenje što bolje očuvalo [2.1]. Dobivena vrijednost predstavlja nijansu sive boje i nalazi se u istom rasponu kao i komponente triju boja. U drugom koraku se određuje granična vrijednost (engl. *threshold*) koja mora biti u istom rasponu. Napokon, sve nijanse sive boje čija vrijednost je ispod granične poprimaju crnu boju, dok se onima iznad granične vrijednosti dodjeljuje bijela boja.

$$v_{siva} = 0.3 * v_{crvena} + 0.59 * v_{zelena} + 0.11 * v_{plava} \tag{2.1}$$

Iako je postupak binarizacije vrlo jednostavan, nepoznata vrijednost graničnog praga ujedno predstavlja i glavni problem. Vrlo je poželjno da se automatizira određivanje ove vrijednosti. Bitno je uočiti da će granična vrijednost izravno ovisiti o svjetlini slike. Što je slika svjetlija bit će potreban veći prag kako bi se njime obuhvatili željeni svijetli pikseli. Vrijedi obrnuto za suprotni slučaj pa će optimalna vrijednost praga padati za tamnije slike.

Postoji više različitih metoda za određivanje praga binarizacije i općenito se dijele u dvije skupine. Prvu skupinu čine metode globalne binarizacije (engl. *Global thresholding*) i zajedničko im je to da sve računaju jedan globalni prag za cijelu sliku te zatim binariziraju sliku koristeći izračunatu vrijednost. Druga skupina sastoji se od metoda koje koriste lokalnu ili adaptivnu binarizaciju (engl. *Adaptive thresholding*). Razlog pojave ovih metoda leži u činjenici da vrlo često slike nisu

jednoliko osvjetljenje, već su neki dijelovi tamniji, a neki svjetliji. Budući da je vrlo moguća pojava prometnog znaka koji je samo djelomično izravno osvijetljen, dok mu je drugi dio u sjeni što ga čini problematičnim za postupak binarizacije, potrebno je obratiti pozornost i na ovaj problem.

Za ostvarivanje globalne binarizacije korišten je iterativni algoritam k-srednjih vrijednosti [2] koji dijeli vrijednosti piksela u dvije grupe te računa novu vrijednost na temelju srednjih vrijednosti. Koraci algoritma dani su u nastavku.

Algoritam 1 Određivanja praga metodom k-srednjih vrijednosti

1. Određuje se početna vrijednost praga T, nasumično ili pomoću neke druge metode.

2. Slika se segmentira u dvije grupe, jedna predstavlja vrijednosti piksela objekta, a druga se odnosi na pozadinske piksele:

$$G_1 = \{f(m,n) : f(m,n) > T\} \tag{2.2}$$

$$G_2 = \{f(m,n) : f(m,n) \leq T\} \tag{2.3}$$

   gdje $f(m,n)$ sadrži vrijednost piksela u $m-tom$ stupcu i $n-tom$ retku.

3. Izračunava se srednja vrijednost za obje grupe:

$$m_1 = \frac{\sum_{k=1}^{n} x_k}{n}, \quad x_k \in G_1 \tag{2.4}$$

$$m_2 = \frac{\sum_{k=1}^{n} x_k}{n}, \quad x_k \in G_2 \tag{2.5}$$

4. Nova vrijednost praga je srednja vrijednost od m_1 i m_2.

$$T_{novi} = \frac{m_1 + m_2}{2} \tag{2.6}$$

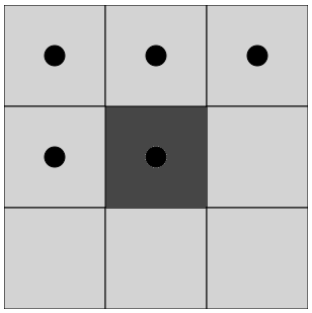
5. Zamijeni staru vrijednost praga novom i vrati se na drugi korak. Algoritam se ponavlja sve dok nova izračunata vrijednost ne bude jednaka staroj tj. dok ne dođe do konvergencije.

Adaptivna binarizacija ostvarena je pozivanjem funkcije implementirane unutar biblioteke *opencv*[1].

[1] *Open Source Computer Vision* (OpenCV) je biblioteka programskih funkcija namijenjenih prvenstveno području računalnog vida te obradi slika i videa u stvarnom vremenu. Više riječi

2.2. Analiza povezanih komponenata

Analiza povezanih komponenata (engl. *connected component labeling*) je algoritam čija se ideja zasniva na teoriji grafova. Ako skup svih povezanih komponenata na slici promatramo kao graf, a svaki piksel kao jedan čvor tog grafa, tada se svaka pojedina komponenta može promatrati kao podgraf promatranog grafa. Pojam komponenata u nazivu zapravo označava *otoke* povezanih piksela iste boje. Svaki piksel može biti povezan sa najviše osam piksela koji ga okružuju. Budući da se u radu koriste binarizirane slike sa dvije vrijednosti boja, to znači da će jednu komponentu činiti skupina međusobno povezanih crnih piksela.



Slika 2.1: Promatrano susjedstvo za trenutni piksel (u centru)

Algoritam se sastoji od dva prolaska kroz raster. U prvom prolasku se na temelju povezanosti postavljaju oznake pripadnosti svakom pikselu, dok se u drugom rješavaju prepoznate ekvivalencije između piksela.

---

o ovoj biblioteci bit će u poglavlju posvećenom programskoj izvedbi.

Algoritam 2 Analiza povezanih komponenata

`Prvi prolazak:`

- Iteriraj kroz sve piksele po stupcima, pa zatim po redcima (rastersko skeniranje)
- Ako piksel nije boje pozadine (obično se za pozadinu uzima bijela boja)
    1. Dohvati susjedne crne piksele koji su odmah lijevo i iznad trenutnog piksela jer su oni do sada obrađeni (1)
    2. Ako nema susjeda tada trenutni piksel označi novom oznakom i prijeći na idući
    3. Inače, pronađi susjeda s najmanjom oznakom i dodijeli tu oznaku trenutnom elementu
    4. Spremi izraze ekvivalencije između susjednih oznaka

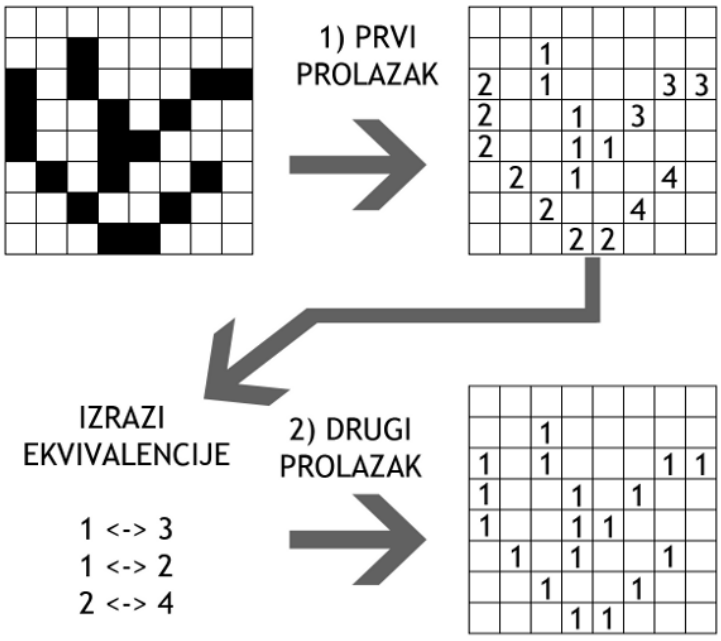
`Drugi prolazak:`

- Iteriraj kroz sve piksele po stupcima, pa zatim po redcima (rastersko skeniranje)
- Ako piksel nije boje pozadine (obicno se za pozadinu uzima bijela boja)
    1. Promjeni oznaku piksela u najmanju ekvivalentnu oznaku

2.3. Filtriranje lažnih pozitivnih detekcija

Nakon analize povezanih komponenata na slici su prepoznate skupine povezanih crnih piksela. Budući da su za segmentaciju potrebni okviri oko slova, jedine informacije koje treba sačuvati su minimumi x i y osi te visina i širina okvira. Zasigurno, u mnogo slučajeva pronađeni okviri neće okruživati slova već će sadržavati i ostale simbole te rubne površine. Takvi okviri pripadaju skupini lažnih pozitivnih[2] detekcija. U ovom poglavlju bit će riječi o različitim načinima prepoznavanja i izbacivanja neželjenih okvira kako bi na kraju ostala samo slova. Većina ovih algoritama temeljit će se na nekim eksperimentalno utvrđenim svojstvima teksta unutar prometnih znakova te teksta kao takvog općenito.

[2]Lažna pozitivna (engl. *false positive*) detekcija je slučaj kada algoritam detektira slovo tamo gdje ga nema. Suprotno tomu je lažni negativni (engl. *false negative*) slučaj kada algoritam previdi pojavljivanje slova.



Slika 2.2: Algoritam analize povezanih komponenata

Algoritam 3 Filtriranje lažnih pozitiva

– Iteriraj po svim pronađenim okvirima

1. Ako trenutni okvir u sebi sadrži barem jedan drugi okvir ili neki njegov dio izbaci ga iz liste okvira
2. Ako trenutni okvir ne zadovoljava uvjet izbacuje se iz liste

$$0.3 < \frac{visina}{širina} < 15.0 \tag{2.7}$$

3. Usporedi s prosječnom visinom

– Filtriranje po visinskoj razlici

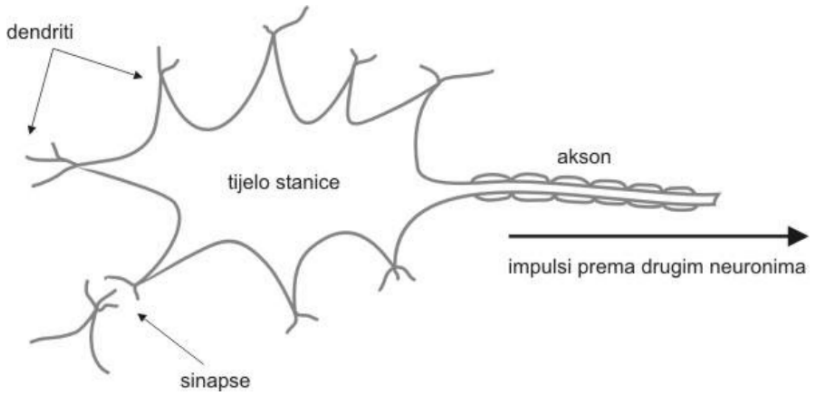
1. Sortiraj okvire prema visinama donjeg brida od najvišeg do najnižeg
2. Iteriraj po preostalim okvirima
3. Ako su okviri lijevo i desno od trenutnog previše odmaknuti na y osi izbaci trenutni okvir[3]

2.4. Obrada segmentiranih slika

Sada kada su eliminirani lažni okviri pristupa se obradi onih koji predstavljaju slova. Kako će u postupku klasifikacije biti korištena neuronska mreža, ove okvire je potrebno spremiti u prikladnom formatu. Odabrat ćemo da slike budu jednake visine i širine, dok će rezolucija biti dovoljno velika kako bi se kasnije mogla smanjiti ovisno o performansama neuronske mreže.

2.5. Neuronska mreža

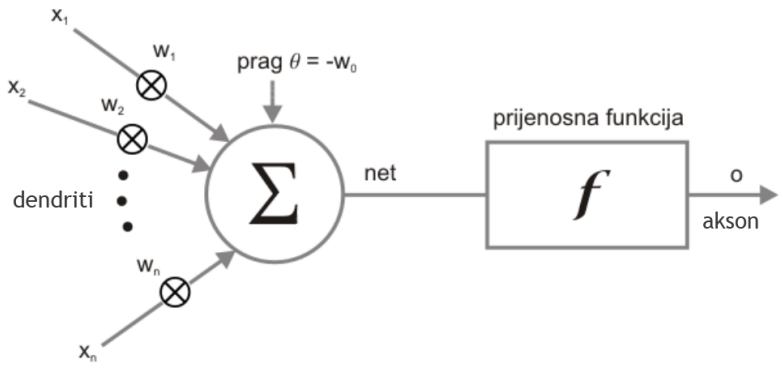
Neuronska mreža, punog naziva umjetna neuronska mreža, je matematički model inspiriran strukturom i načinom rada bioloških neuronskih mreža. Neuroni su osnovna funkcijska jedinica u živčanim sustavima živih bića što uključuje i mozak koji sam sadrži oko 10^{11} neurona. Svaki neuron sastavljen je od tijela stanice ili some iz kojeg su razgranati kraći produžetci zvani dendriti i jedan duži produžetak ili akson. Tijelo stanice sadrži informaciju predstavljenu električkim potencijalom između unutrašnjeg i vanjskog dijela stanice (oko –70 mV u neutralnom stanju). Dendriti imaju ulogu dovođenja neurotransmitera koji povećaju (engl. *excite*) ili smanjuju(engl. *inhibit*) ukupni potencijal some [5]. Ako vrijednost potencijala tijela neurona prijeđe određenu vrijednost praga, neuron emitira električni impuls kroz akson. Akson je na svojem kraju sadrži mnoštvo sinapsa pomoću kojih je povezan sa dendritima drugih neurona pa će svaki poslani impuls kroz akson uzrokovati promjenu potencijala u tim neuronima što za posljedicu ima propagaciju impulsa kroz mrežu. Svaki je neuron preko aksona izravno povezan sa prosječno $10^3 - 10^4$ drugih neurona.



Slika 2.3: Građa neurona

Nakon kraćeg osvrta na rad biološkog neurona, vrijeme je da pogledamo kako je modelirana umjetna neuronska mreža. Neuronska mreža sastoji se od grupe međusobno povezanih umjetnih neurona čija se funkcionalnost temelji na biološkom neuronu i koji služe distribuiranoj paralelnoj obradi podataka te koristi konektivistički[4] pristup razvoju inteligentnih sustava. Ona je prilagodljiv sustav robustan na pogreške u podacima koji mijenja svoju strukturu u odnosu na informacije koje propuštamo kroz nju za vrijeme faze učenja. Najčešće se koristi kako bi modelirala kompleksne odnose između ulaza i izlaza s ciljem pronalaska uzoraka u podacima i to posebno dobro radi s nejasnim ili manjkavim podacima tipičnim za podatke iz različitih senzora, poput kamera i mikrofona.

Kao i u slučaju bioloških neurona, svaki neuron ima ulazne i izlazne veze te graničnu vrijednost okidanja. Vrijednost *potencijala* u tijelu stanice modelirana je sumom ulaznih vrijednosti. Svakoj vezi između dva neurona pridijeljena je vrijednost težine w_i. Upravo težine veza su mjesto gdje se obično odvija učenje mreže pa kažemo da je znanje o izlazu kao funkciji ulaza pohranjeno implicitno u tim težinama. Funkcionalnost aksona ostvarena je prijenosnom (aktivacijskom) funkcijom. Na slici [2.4] prikazan je matematički model neurona. Možemo primijetiti da su električni signali predstavljeni numeričkim vrijednostima, jakost sinapse opisana je težinskim faktorom w_i, tijelo stanice čini zbrajalo, a akson je prijenosna funkcija f. Bitno je shvatiti da je propagacija signala kroz umjetnu neuronsku mrežu ovim modelom bitno pojednostavljena u usporedbi s kompleksnim elektrokemijskim reakcijama koje se odvijaju u biološkim neuronima.



Slika 2.4: Model neurona [6]

Prijenosna funkcija f preslikava izlaznu vrijednost zbrajala u vrijednost koja

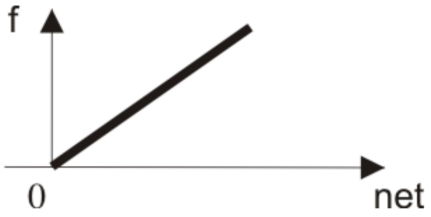
[4]Konektivistički pristup temelji se na izgradnji sustava arhitekture slične arhitekturi mozga koji, umjesto da ga se programira, uči samostalno na temelju iskustva

je obično između -1 i 1, ovisno koja funkcija se koristi. Izlazna vrijednost zbrajala *net* dana je formulom [2.8], dakle to je ukupna vrijednost zbroja kojoj se oduzima vrijednost praga okidanja θ. Prag se oduzima ukupnom zbroju kako bi se njegova vrijednost svela na nulu jer je to vrlo pogodno za različite prijenosne funkcije. Analogno ovome bi bilo da smo biološki neuron natjerali da *puca* samo ako je potencijal pozitivan.

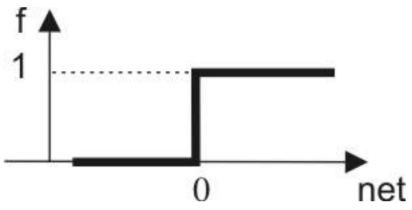
$$net = w_1x_1 + w_2x_2 + \ldots + w_nx_n - \theta = \sum_{k=1}^{n} w_ix_i - \theta \tag{2.8}$$

Na slikama su prikazane najpoznatije prijenosne funkcije. Funkcija ADALINE (engl. *Adaptive Linear Element*) [2.5] je najjednostavnija prijenosna funkcija koja jednostavno kao izlaz neurona daje težinsku sumu njegovih ulaza. Funkcija skoka [2.6] kao izlaz daje samo diskretne vrijednosti 1 ili 0 ovisno o tome da li je postignut aktivacijski potencijal ili ne. Neuron koji za prijenosnu funkciju koristi funkciju skoka se još naziva i TLU perceptron (engl. *Thresholding logical unit*). TLU perceptron jest linearni klasifikator i može raditi samo sa linearno razdvojivim problemima. Problem je linearno razdvojiv ako je moguće povući hiperravninu koja će razdvojiti različite izlaze perceptrona koji se dobiju za različite ulaze. Dakle potrebno je razdvojiti izlaze 1 od izlaza 0 gdje su položaji u hiperprostoru određeni ulaznim podacima. Ako uzmemo da imamo dva ulazna podatka x_1 i x_2, tada će izlazne vrijednosti biti razmještene u 2D prostoru na koordinatama (x_1, x_2). Jasno je sada da se svim točkama pridružuju pripadajuće izlazne vrijednosti i ako se može povući pravac koji će s jedne strane imati samo točke koje predstavljaju izlaz 1, a sa druge one kojima je pridružena 0, tada je taj problem linearno razdvojiv.

Posljednja je dana sigmoidalna funkcija [2.7] koja je ujedno i najčešće korištena, a koristit će se i u ovom radu. Prednost ove funkcije je u tome što je, za razliku od prve dvije, derivabilna. To je bitno svojstvo potrebno za implementaciju *backpropagation* algoritma za učenje koji će biti opisan u idućem poglavlju.



Slika 2.5: ADALINE funkcija



Slika 2.6: Funkcija skoka

– `ADALINE funkcija:`

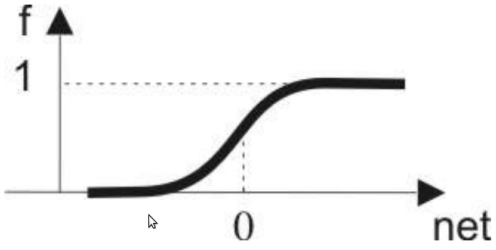
$$f(net) = net$$

– `Funkcija skoka:`

$$f(net) = \begin{cases} 0 & \text{za } net < 0 \\ 1 & \text{inace} \end{cases}$$

– `Sigmoidalna funkcija:`

$$f(net) = \frac{1}{1 + e^{-a \cdot net}}$$



Slika 2.7: Sigmoidalna funkcija

Neuronske mreže su, pored SVM-a[5], jedna od najčešćih metoda korištenih u strojnom očitanju teksta. U ovom radu koriste se za klasifikaciju slova. Rad s neuronskim mrežama može se podijeliti u dvije faze: fazu učenja (treniranja) i fazu obrade podataka ili iskorištavanja mreže. U fazi učenja mreži će biti predočeni uzorci slova iz skupa za učenje dobiveni prethodno opisanim postupcima segmentacije. Dakle skup segmentiranih slika u ovoj fazi potrebno je podijeliti na dva podskupa. Jedan će služiti za učenje, dok će drugi biti korišten prilikom testiranja mreže. Učenje mreže je nadzirano od strane učitelja što znači da će za svaki ulaz prilikom učenja mreža znati što bi trebala dati na izlazu. Kod svake slike, u prvom slovu naziva datoteke, označeno je o kojem je slovu riječ. Na ovaj način pojednostavljen je postupak generiranja ulaznih podataka za neuronsku mrežu.

Struktura neuronskih mreža može se podijeliti na tri sloja. Razlikujemo ulazni sloj, skriveni sloj i izlazni sloj. Ulazni sloj čine neuroni na koje dovodimo ulazne podatke. U ovom slučaju ulaz predstavljaju vrijednosti piksela binarizirane slike slova. To znači da će ako su slike rezolucije 24×24, na ulaz doći 576 vrijednosti

[5]Support vector machine - `http://en.wikipedia.org/wiki/Support_vector_machine`

piksela zbog kojih će ulazni sloj imati isti toliki broj neurona. Skriveni sloj čine neuroni čija je jedina uloga propagacija podataka kroz mrežu. Ovaj sloj se može sastojati od više podslojeva, no najčešće sadrži samo jedan jer se pokazalo da je jedan sloj sasvim dovoljan da veliku skupinu problema. Postavlja se pitanje kako odrediti broj čvorova skrivenog sloja. Svakako ne želimo da ovaj broj bude premalen Odgovor je da nema egzaktnog pravila za ovaj problem, već postoje razne smjernice. Neke od smjernica dane su u nastavku [3].

1. Broj skrivenih neurona trebao bi se nalaziti između broja neurona u ulaznom sloju i broja u izlaznom sloju.

2. Broj skrivenih neurona trebao bi biti jednak 2/3 veličine ulaznog sloja, plus veličina izlaznog sloja.

3. Broj skrivenih neurona ne bi smio prelaziti dvostruku veličinu ulaznog sloja.

No na kraju je najsigurnije ovaj broj odrediti eksperimentalno. Bilo da smo vođeni smjernicama, iskustvom ili pukim instinktom, dovoljno je testirati mrežu za različite veličine skrivenog sloja neurona i odabrati onu koja daje najbolje rezultate. Ipak treba paziti da se ne odabere premalen broj jer će prejednostavan model raditi mnogo grešaka budući da neće biti u stanju obuhvatiti kompleksnost podataka. Isto tako ako se uzme prevelik broj neurona, neki od njih ostat će nedovoljno trenirani jer skup za učenje neće sadržavati dovoljno podataka kako bi se treniranje dovršilo u potpunosti. Pored toga prevelika složenost mreže će bespotrebno usporiti proces učenja kao i samo korištenje.

2.5.1. Algoritam s propagiranjem greške unatrag

Do sada smo pričali o strukturi neuronske mreže i stekli uvid o tome kako ona radi. Spomenuli smo i to da je mrežu prije korištenja najprije potrebno naučiti, no nismo rekli ništa o tome kako se to učenje može ostvariti. Znamo da je naučeno znanje mreže spremljeno implicitno u težinama veza između neurona i to nam daje naslutiti da će biti potreban algoritam koji će mijenjati upravo vrijednosti tih težina.

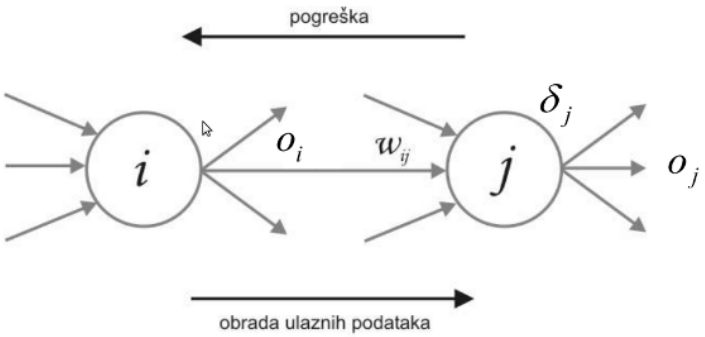
Algoritam s propagiranjem greške unatrag (engl. *backpropagation algorithm*) je učinkovit i popularan algoritam koji učenje mreže izvodi širenjem pogreške unatrag. *Backpropagation* radi samo sa derivabilnim prijenosnim funkcijama pa se uz njega najčešće koristi sigmoidalna prijenosna funkcija [2.7]. Ova kombinacija omogućuje neuronskoj mreži da predstavlja visoko nelinearne odnose između

ulaznih i izlaznih podataka što je bitna prednost naspram TLU perceptrona za kojeg smo vidjeli da može raditi samo sa linearno razdvojivim problemima.

Algoritam koristi metodu gradijentnog spusta kako bi minimizirao nastalu pogrešku $E(w)$ na izlazu mreže nad skupom primjera za učenje D definiranu kao

$$E(w) = \frac{1}{2} \sum_{d \in D} \sum_{k \in izlazi} (t_{kd} - o_{kd})^2 \tag{2.9}$$

pri čemu su t_{kd} i o_{kd} ciljna i stvarna izlazna vrijednost za k-ti neuron izlaznog sloja dobivene s primjerom za učenje d.



Slika 2.8: Povezani neuroni

Koraci algoritma prikazani su u nastavku [4] [6]. Korištena notacija: o_i i o_j su izlazi iz neurona i odnosno neurona j, w_{ij} je odgovarajuća težina, δ_n je pogreška izlaza jedinice n, η je stopa učenja. Ove su veličine prikazane na slici [2.8].

Mreži se predočavaju uzorci za učenje u obliku parova (x, t) gdje je x niz ulaznih vrijednosti a t niz ciljnih izlaznih vrijednosti. Algoritam najprije inicijalizira težinske faktore na slučajne vrijednosti, zatim ponavlja predstavljanje svih uzoraka za učenje mreži sve dok se ne ispuni uvjet zaustavljanja. Uvjet zaustavljanja može biti dozvoljeni iznos ukupne pogreške ili maksimalni broj epoha tijekom učenja. Nadalje, za svaki uzorak računaju se i spremaju izlazi svih neurona na način da se uzorak propagira unaprijed, od ulaza ka izlazu. Pogreška δ_k u izlaznom sloju računa se u odnosu na odstupanje stvarnog izlaza o_k od ciljnog izlaza t_k.

Nakon izlaznog sloja računaju se pogreške δ_h u skrivenom sloju gdje je bitno napomenuti da se, ukoliko je riječ o višeslojnoj mreži, one računaju počevši od sloja najbližeg izlaznom sloju da bi se zatim pogreška propagirala unatrag sve do sloja izravno povezanog s ulaznim slojem. Pogreška za h-ti neuron u skrivenom sloju dobije se u odnosu na zbroj pogrešaka δ_s svih neurona na koje izlaz h-tog neurona utječe pomnoženih težinskim faktorom w_{hs}. Kao što je vidljivo iz

notacije, faktor je zapravo težina spoja između neurona h i s. Bitno je uočiti da se svi neuroni s na koje utječe izlaz neurona h nalaze *nizvodno*. U prijevodu jedan njihov ulaz je izravno povezan sa izlazom neurona h, što znači da će njihove pogreške sigurno biti već izračunate u ranijim koracima.

Na samom kraju se provodi korekcija nad svim težinskim faktorima. Iznos korekcije Δw_{ij} težine w_{ij} proporcionalan je izračunatoj pogrešci pripadajućeg neurona δ_j, faktoru učenja η te ulaznoj vrijednosti o_i koja je dovedena na tu težinu.

Algoritam 4 *Backpropagation*

- **Inicijaliziraj** težinske faktore na slučajne vrijednosti
- **Dok** nije ispunjen uvjet zaustavljanja **čini**
  - Za svaki (x, t) iz D **čini**
    1. Izračunaj izlaz o_u za svaki neuron u za ulaz x
    2. Za svaki **izlazni** neuron k izračunaj pogrešku δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \tag{2.10}$$

    3. Za svaki **skriveni** neuron h izračunaj pogrešku δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{s \in Nizvodno} w_{hs}\delta_s \tag{2.11}$$

    4. Ugodi svaki težinski faktor w_{ij}

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij} \tag{2.12}$$

    gdje je

$$\Delta w_{ij} = \eta \delta_j o_i$$

3. Programska izvedba

Programska izvedba ostvarena je u jeziku C++. Grafičko sučelje napravljeno je u Qt radnom okviru, a kao pomoćna biblioteka korišten je OpenCV.

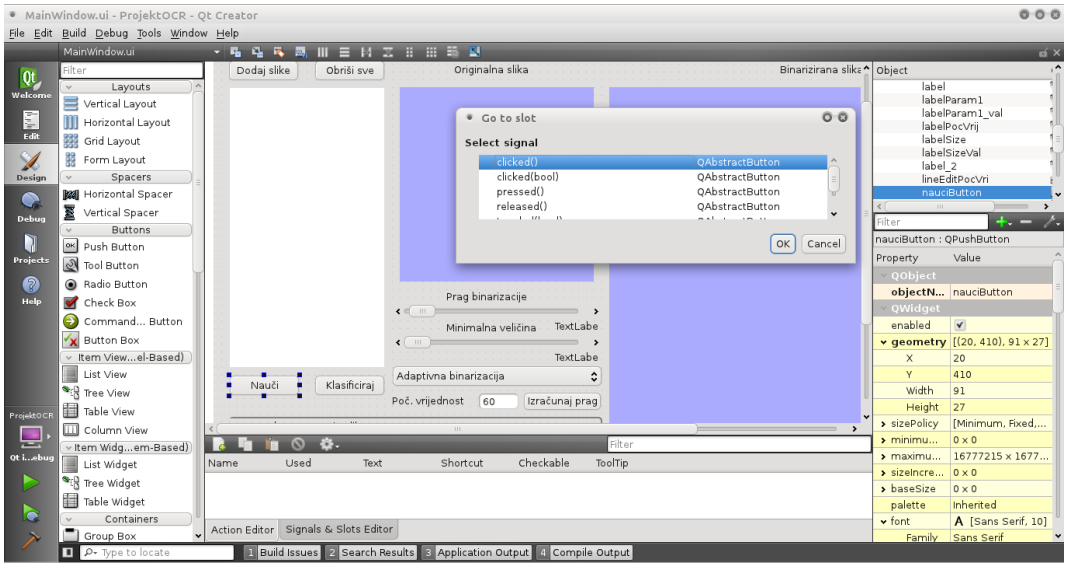
3.1. Radni okvir Qt

Qt (engl. */'kju:t/*) je multiplatformski (Linux, Windows, OS X, Symbian) radni okvir koji se široko koristi za razvoj aplikacija s grafičkim sučeljem u jeziku C++. U manjoj mjeri koristi se i za razvoj bez grafičkih sučelja budući da nudi razna proširenja za C++ jezik poput vlastitih standardnih tipova podataka, kao i podršku za rad s bazama podataka, XML parsiranje, paralelizam te mrežno programiranje. Dodatna proširenja postiže i koristeći posebni generator koda (engl. *Meta Object Compiler*) obogaćujući C++ dodatnom sintaksom.

Distribuiran pod *GNU Lesser General Public* licencom, Qt je besplatan i otvorenog koda (engl. *open source*). U ovom radu korišten je Qt verzije 4.7.1. Adresa službene stranice je `http://qt.nokia.com/`. Za početak rada dovoljno je skinuti i instalirati SDK za odgovarajući operacijski sustav. SDK uz sve biblioteke sadrži i Qt Creator IDE [3.1] u kojemu je programska izvedba ovog rada i ostvarena. IDE nudi sučelje jednostavnog i ugodnog dizajna te razne mogućnosti poput sučelja za dizajniranje prozora aplikacija.

3.2. OpenCV

OpenCV (*Open Computer Vision library*) je open source multiplatformska biblioteka pod BSD licencom što znači da je slobodna kako za akademsku, tako i za komercijalnu upotrebu. Sadrži preko 2000 optimiranih algoritama iz područja računalnog vida koji rade na obradi slika u stvarnom vremenu. Prva verzija napisana je u programskom jeziku C, a kasnije je prepisana u C++ i sav danji razvoj nastavljen je u tom jeziku. Uz C++ postoje programska sučelja prema



Slika 3.1: Qt Creator IDE

Pythonu, Rubyu, Matlabu, Javi i drugim jezicima. Sadrži module za rad sa matričnim strukturama podataka, rad sa video sadržajem iz datoteka ili kamera, procesiranje i transformaciju slika, izgradnju histograma i kontura, alate za praćenje pokretnih objekata, kalibriranje kamera, izgradnju projekcije i 3D pogleda, algoritme za strojno učenje, te modul za izgradnju grafičkog sučelja. U ovom je radu korištena verzija biblioteke 2.1.

– Upute za instalaciju i korištenje mogu se pronaći na službenoj stranici: `http://opencv.willowgarage.com/`

– Za Ubuntu postoje paketi u službenom repozitoriju dostupni preko apt-get naredbe. `https://help.ubuntu.com/community/OpenCV`

3.3. Generiranje skupa segmentiranih slova

Prva od dvije razvijene aplikacije s grafičkim sučeljem u Qt-u služi za generiranje skupa segmentiranih slova za kasnije učenje i testiranje neuronske mreže. Cilj aplikacije je da što je moguće više olakša učitavanje skupa slika, ostvari automatsku segmentaciju slova na slikama te ubrza unos slova od strane korisnika prilikom spremanja slika.

Za rad sa slikama iskorištene su neke od osnovnih funkcionalnosti koje nudi biblioteka OpenCV [1]. Jedna od češće korištenih metoda je `cvLoadImage` koja učitava sliku sa dane lokacije. Veliku prednost predstavlja mogućnost učitavanja slika spremljenih u bilo kojem od popularnih formata (JPG, PNG, BMP, PBM,

PGM, SR, TIFF ...).

```
IplImage* cvLoadImage(const char* filename, int iscolor)
```

`filename` - lokacija slike.
`iscolor` - određuje da li će slika biti učitana u boji ili *grayscale*. Nama treba slika u sivim tonovima pa će ovdje ići 0.
`returns IplImage*` - funkcija vraća pokazivač na sliku.

Funkcija `cvThreshold` korištena je za globalnu binarizaciju.

```
void cvThreshold(const CvArr* src, CvArr* dst,
                 double threshold, double max_value,
                 int threshold_type )
```

`src` - izvorna slika koja se binarizira.
`dst` - odredišna slika u koju će biti spremljena binarizirana slika.
`threshold` - vrijednost globalnog praga binarizacije.
`max_value` - maksimalna vrijednost do koje funkcija treba raditi s pikselima. Potrebno je da radi sa svim vrijednostima pa će se ova vrijednost postavljati na 255.
`threshold_type` - tip binarizacije. Može biti normalna pa se funkciji prosljeđuje makro `CV_THRESH_BINARY` ili inverzna kada se prosljeđuje `CV_THRESH_BINARY_INV`.

Adaptivna binarizacija riješena je pozivanjem `cvAdaptiveThreshold` funkcije.

```
void cvAdaptiveThreshold(const CvArr* src, CvArr* dst,
        double max_value, int adaptive_method,
        int threshold_type, int block_size=3,
        double param1=5)
```

`src, dst, max_value, threshold_type` - isto kao kod `cvThreshold`.
`adaptive_method` - koristi se algoritam koji računa vrijednost praga za svaki piksel zasebno i pritom ih binarizira s obzirom na dobivenu vrijednost. Prag se dobije tako da se izračuna srednja vrijednost susjedstva promatranog piksela. Veličinu obuhvaćenog susjedstva određuje parametar `blockSize` koji mora biti neparan jer određuje veličinu stranice kvadrata unutar kojeg promatrani piksel mora biti smješten točno u sredini. Iznos ovog parametra određen je eksperimentalno u odnosu na veličinu slike. Ovisno o dimenzijama slike promatrano susjedstvo iznosi 9×9, 17×17 ili 27×27 piksela. Konačna vrijednost praga dobije se oduzimanjem parametra `param1` od izračunate srednje vrijednosti.

(a) Nejednoliko osvijetljena siva slika

(b) Binarizacija adaptivnom metodom

(c) Binarizacija globalnom metodom

Slika 3.2: Usporedba adaptivne i globalne binarizacije

(a) Nejednoliko osvijetljena siva slika

(b) Binarizacija adaptivnom metodom

(c) Binarizacija globalnom metodom

Slika 3.3: Usporedba adaptivne i globalne binarizacije

`MainWindow` klasa je glavna klasa grafičkog sučelja aplikacije. U njoj su implementirane funkcionalnosti koje ono ostvaruje. Ova klasa koristi klasu `AbsTextSegment` koja za nju obavlja segmentaciju teksta iz slika.

`izlazni_folder, zadnji_folder` - prvi pamti odabrani direktorij za spremanje slika, a drugi pamti posljednji direktorij u kojemu su dodavane slike u listu.

`ui` - preko ovog pokazivača pristupa se svim elementima grafičkog sučelja.

`slika` - pokazivač na klasu koja obavlja posao segmentacije nad trenutno odabranom slikom.

`model` - ova klasa implementira funkcije za rad s listom koja sadrži popis dodanih slika.

`trenutni_index_` - pamti trenutno označeni element u modelu liste.

`obradiNovuSliku(QModelIndex index, int thr_val, int size_val)` - funkcija se poziva kada je označena nova slika za obradu. Prima indeks slike u listi, vrijednost praga globalne binarizacije te minimalnu dozvoljenu veličinu okvira slova.

Tablica 3.1: `MainWindow` razred

QMainWindow
MainWindow
- izlazni_folder: QString - model: FileListModel* - slika: TextSegment* - trenutni_index_: QModelIndex - ui: UI::MainWindow* - zadnji_folder: QString
+ izbaciBBox(QPoint): void + isSlikaNull(): bool - IplImage2QImage(IplImage*): QImage - iterativeThreshold(IplImage*, int): int - obradiNovuSliku(QModelIndex, int, int): void - obradiTrenutnuSliku(int, int): void - saveImages(int, int): void - setThrScrollVisible(bool): void - thrStrategyFactory(): AbsThresholdStrategy* # closeEvent(QCloseEvent): void # keyPresEvent(QKeyEvent*): void # on_scrollBarThrVal_valueChanged(int): void # on_...(): void # on_ucitajButton_clicked(): void

`obradiTrenutnuSliku(int thr_val, int size_val)` - funkcija se poziva kada je potrebno iznova obraditi trenutno odabranu sliku s promijenjenim parametrima.

`saveImages(int width, int height)` - funkcija se poziva kada korisnik pokrene postupak spremanja segmentiranih okvira. Prima veličinu rezolucije u kojoj će spremati slike.

`iterativeThreshold(IplImage* img, int init_val)` - funkcija iterativnom metodom računa te vraća prag za strategiju globalne binarizacije.

`IplImage2QImage(const IplImage *iplImage)` - radi konverziju iz opencv slike u tip slike koji koristi Qt. `setThrScrollVisible(bool val)` - postavlja vidljivost kontrola koje koristi globalna binarizacija ovisno o tome da li je ona odabrana kao strategija koja se koristi.

`thrStrategyFactory()` - metoda tvornica koja vraća instancu one strategije binarizacije koja je trenutno odabrana.

```
AbsThresholdStrategy* MainWindow::thrStrategyFactory() {
    int index = ui->comboBoxBinarizacija->currentIndex();
    switch(index) {
    case(0):
        return new AdaptiveThreshold;
    case(1):
        return new ClassicThreshold;
    }
}
```

`on_widgetName_action()` - metode ovakve sintakse pozivaju se kada je određena akcija *action* izvršena nad određenim elementom sučelja imena *widgetName*. Bitno je napomenuti da se ne mogu sve akcije pozvati nad svakim tipom *widget* elementa. Najlakše je dodati akciju u grafičkom dizajneru desnim klikom na element te odabirom `Go to slot...`.
`izbaciBBox(QPoint mpos)` - poziva se klikom miša unutar binarizirane slike. Metoda izbacuje okvir koji sadrži danu poziciju točke unutar sebe.
`keyPressEvent(QKeyEvent * event)` - redefinira (engl. *overrides*) baznu metodu za hvatanje događaja tipkovnice.
`closeEvent(QCloseEvent * event)` - redefinira baznu metodu koja se poziva zatvaranjem prozora.

`MouseClickGetter` klasa Slika [3.2] izvodi `QObject` klasu i redefinira njenu metodu `eventFilter` koja hvata sve događaje i filtrira ih prema vrsti. Objekt ove klase može se instalirati kao *event listener* nekom widgetu. U konkretnom slučaju nama je potreban kako bi hvatao klikove mišem na prostor binarizirane slike te izbacivao pripadajuće okvire.

```
MouseClickGetter *mouse_click = new MouseClickGetter(this);
ui->obradenaSlikaLabel->installEventFilter(mouse_click);
```

`KeyGetterQListView` klasa Slika [3.3] izvodi `QListView` klasu i redefinira njenu metodu `keyPressEvent` koja obrađuje događaje tipkovnice. Cilj je ovu metodu iznova implementirati tako da samo proširi baznu metodu na način da emitira signal klika miša ukoliko je došlo do promjene označenog elementa liste. Ovime je omogućeno da korisnik ne mora stalno kliknuti svaki put na iduću sliku tijekom postupka generiranja, već se može poslužiti strelicama za pomi-

Tablica 3.2: `MouseClickGetter` razred

QObject
MouseClickGetter
- parent_: MainWindow*
+ MouseClickGetter(QObject*) # eventFilter(QObject*, QEvent*): bool

canje po listi, a ova klasa će za njega odraditi i klik miša. Nakon definiranja klase dovoljno je podesiti da element sučelja koji sadrži listu slika bude tipa `KeyGetterQListView`.

Tablica 3.3: `KeyGetterQListView` razred

QListView
KeyGetterQListView
- parent_: MainWindow*
+ KeyGetterQListView(QWidget*) # keyPressEvent(QKeyEvent*): void

```
void keyPressEvent(QKeyEvent *event) {
    //zapamti stari index
    QModelIndex oldIdx = currentIndex();
    //zove baznu metodu da obavi pomicanje indeksa
    QListView::keyPressEvent(event);
    //dohvati novi index
    QModelIndex newIdx = currentIndex();
    //ako su razliciti emitira clicked na novom indeksu
    if(oldIdx.row() != newIdx.row()) {
        emit clicked(newIdx);
    }
}
```

`FileListModel` klasa Slika [3.4] izvodi `QAbstractListModel` klasu i proširuje njeno ponašanje. `rowCount(const QModelIndex &parent` - funkcija vraća broj elemenata u listi tj. broj slika.
`vratiLokaciju(int indeks)` - funkcija vraća element liste na traženom indeksu

Tablica 3.4: `FileListModel` razred

QListView
FileListModel
- fileCount: int
- fileList: QStringList
+ FileListModel(QObject*)
+ data(QModelIndex&, int): QVariant{query}
+ dodajImena(QStringList&): void
+ numberPopulated(int): void
+ ocisti(): void
+ rowCount(QModelIndex&): int{query}
+ vratiKlasu(int): QString
+ vratiLokaciju(int): QString

(lokaciju slike).
`vratiKlasu(int indeks)` - funkcija vraća koje slovo je na traženom indeksu (klasu slike).
`ocisti()` - funkcija briše sve elemente u listi ostavljajući je praznom.

`AbsThresholdStrategy` klasa Slika [3.5] je apstraktna klasa koja omogućuje da se iz nje izvedu različite strategije po kojima će se izvoditi binarizacija slika. U sklopu ovog rada izvedene su dvije strategije binarizacije.
Razred `ClassicThreshold` izvodi strategiju koja se postavlja ako je odabrana globalna binarizacija[1]. Razred `AdaptiveThreshold` izvodi strategiju koja se postavlja ako je odabrana adaptivna binarizacija[2].

Tablica 3.5: `AbsThresholdStrategy` razred

AbsThresholdStrategy
+ threshold(IplImage*, IplImage*, int): void
+ thresholdInverse(IplImage*, IplImage*, int): void

`AbsSaveStrategy` klasa Slika [3.6] je apstraktna klasa koja omogućuje da se iz nje izvedu različite strategije koje definiraju kako će slike biti spremane na disk. Potrebno je naglasiti da želimo slike spremati na načine koji su pogodni za korištenje s neuronskom mrežom. Dakle cilj je pokušati što bolje centrirati

[1] Koristi `cvThreshold` metodu.
[2] Koristi `cvAdaptiveThreshold` metodu.

slovo unutar kvadratne slike ili točnije, dovoljno je postići da unutar jedne klase slova dobijemo jednako pozicioniranje slova, ali i jednake proporcije između slova, unutar kvadratne slike. U sklopu ovog rada izvedene su tri strategije spremanja slika.

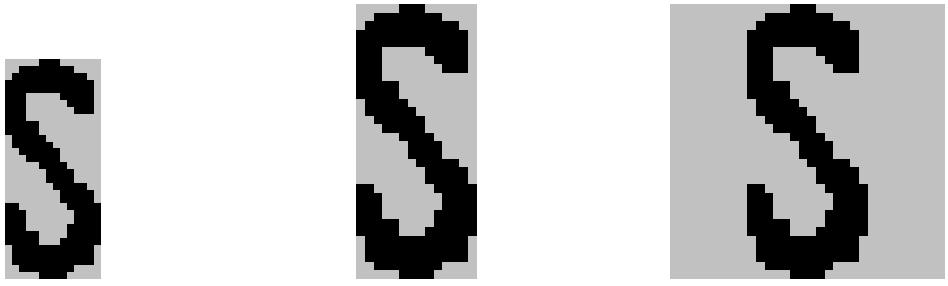
Tablica 3.6: `AbsSaveStrategy` razred

AbsSaveStrategy	
#	`izlaz_dir: std::string`
+	`saveBlobImage(IplImage*, CvRect&, int, int, char): void`

U samoj aplikaciji odabir strategije spremanja ostvaren je putem padajućeg izbornika (engl. *combo box*). Pored tri strategije moguće je odabrati i četvrtu opciju koja koristi sve tri strategije, odnosno sprema sliku na sva tri načina. Razred `FixedSaveStrategy` izvodi strategiju koja sprema sliku prema sljedećem algoritmu.

Algoritam 5 FixedSaveStrategy spremanje slika

1. Slika se najprije skalira po većoj dimenziji do željene rezolucije zadržavajući pritom omjer visine i širine jednakim. [4.6b]
2. Zatim se skalirana slika produžuje po manjoj rezoluciji do željene vrijednosti s obje strane jednako kako bi slika ostala centrirana. [4.6d]



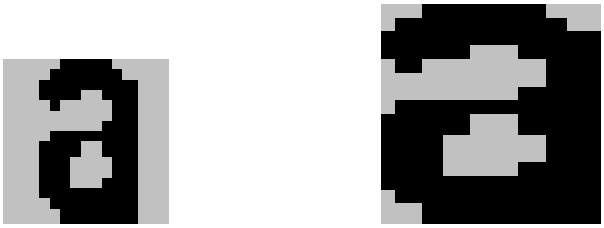
(a) Originalno segmentirano slovo

(b) Skaliranje veće dimenzije do tražene rezolucije

(c) Širenje manje dimenzije do veće (kvadrata)

Slika 3.4: `FixedSaveStrategy` strategija

Razred `ScaleSaveStrategy` izvodi apstraktnu klasu i implementira jednostavno skaliranje slike do željene rezolucije po obje dimenzije. [3.5b]



(a) Originalno segmentirano slovo

(b) Skaliranje obje dimenzije do željene rezolucije

Slika 3.5: `ScaleSaveStrategy` strategija

Razred `CenterSaveStrategy` izvodi klasu `AbsSaveStrategy` i implementira strategiju spremanja slika u odnosu na njihovo težište. Težište slike definirat ćemo kao koordinatu piksela koji ima svojsto da ako kroz njega povučemo vertikalnu, odnosno horizontalnu liniju, ona dijeli površinu crnih piksela na dva jednaka dijela. Konačno, cilj ovog algoritma je da podudara težište slike sa centrom nove slike. Primjetimo da se ovaj algoritam razlikuje od fiksnog spremanja samo u tome što je potonji podudarao centar originalne slike s centrom nove slike.

Algoritam 6 CenterSaveStrategy spremanje slika

1. Slika se najprije skalira po većoj dimenziji do željene rezolucije zadržavajući pritom omjer visine i širine jednakim.

2. U ovom koraku potrebno je izračunati težište skalirane slike.

3. Zatim se skalirana slika produžuje po manjoj rezoluciji do željene vrijednosti i to tako da težište izvorišne slike sada postane centar odredišne slike.

Razred `TextSegment` izvodi klasu `AbsTextSegment` koja pruža sučelje prema metodama za obradu slika te obavlja segmentaciju teksta oviseći pritom o opencv biblioteci. U nastavku je dan UML[3] zapis klase i opis bitnih metoda.
`addSaveStrategy` - dodaje strategiju spremanja u vektor strategija koje će biti pozivane prilikom spremanja slika.
`getDisplayImage` - vraća binariziranu sliku za prikaz u grafičkom sučelju.
`getGrayscaleImage` - vraća učitanu sliku u sivim tonovima.

[3] *Unified Modeling Language* je standardizirani jezik za slikovno predstavljanje i modeliranje objekata u objektno orijentiranoj paradigmi.

Slika 3.6: Generator slova

`saveBlobImage` - sprema okvir na zadanom indeksu. Funkcija najprije od korisnika traži da unese o kojem se slovu radi, a zatim zove sve strategije u vektoru da ga spreme.

`extractText` - metoda koju pozivamo kako bismo obavili segmentaciju teksta.
`sortText` - metoda sortira okvire odozgo prema dolje te s lijeva na desno.
`filterCharBlobs` - metoda poziva algoritme filtriranja lažnih detekcija određenim redoslijedom.
`filterBlobByPos` - izbacuje okvir koji sadrži dane koordinate. Ova metoda se poziva na klik mišem unutar prostora za prikaz binarizirane slike i izbacuje okvir koji sadrži točku pod tim koordinatama.
`filterByAltitude` - metoda ostvaruje filtriranje po visinskoj razlici [3].
`countBlackPixels` - broji crne piksele u slici.
`filterBlobBySize` - izbacuje sve okvire manje od dane vrijednosti površine.
`drawBox` - crta plavi okvir oko slova na slici za prikaz korisniku.
`filterByHeight` - filtrira okvire po njihovoj prosječnoj visini, ovisno o tome da li visina nekog okvira previše odstupa od prosjeka.

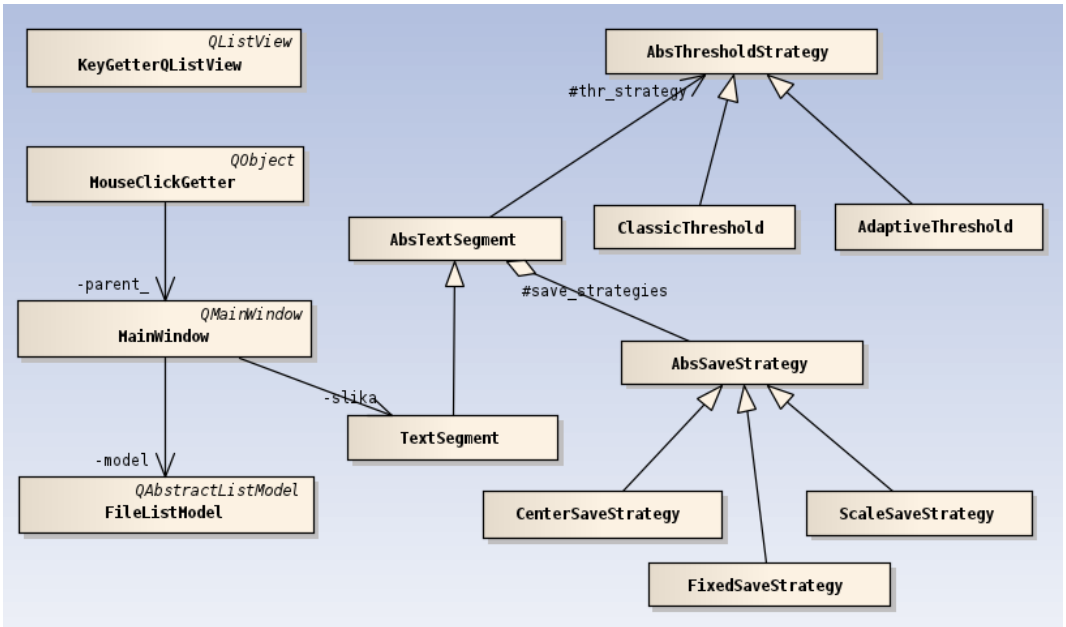
Tablica 3.7: `TextSegment` razred

AbsTextSegment
TextSegment
save_strategies: std::vector<AbsSaveStrategy*>
thr_strategy: AbsThresholdStrategy*
imageOrg: IplImage*
imageThr: IplImage*
displayedImage: IplImage*
bboxes_: std::vector<CvRect>
- small_bboxes_: std::vector<CvRect>
- last_size: int
+ addSaveStrategy(AbsSaveStrategy*): void
+ clearSaveStrategies(): void
+ setThrStrategy(AbsThresholdStrategy*): void
+ clearSaveStrategies(): void
+ getDisplayImage(): IplImage*
+ getGrayscaleImage(): IplImage*
+ saveBlobImage(int, int, int): void
+ getBlobCount(): int
+ extractText(int, int): void
+ setToExit(): void
+ sortText(): void
+ getBlobRect(int): CvRect*
+ filterBlobByPos(int, int): IplImage
- filterByAltitude(): void
- redrawBlobs(): void
- countBlackPixels(IplImage*): int
- filterBlobBySize(int): IplImage*
- drawBox(IplImage*, CvRect): void
- filterCharBlobs(CBlobResult): void
- filterByHeight(): void
- fixedImgShow(CvRect, int, int): IplImage*

3.4. Implementacija neuronske mreže

3.4.1. Biblioteka FANN

Fast Artificial Neural Network je besplatna open source biblioteka za rad s umjetnim neuronskim mrežama koja implementira višeslojne neuronske mreže u programskom jeziku C s podrškom za potpuno povezane i djelomično povezane mreže. Podržava multiplatformsko izvođenje u cjelobrojnom zapisu i zapisu s pomičnim zarezom te uključuje radni okvir za laganu manipulaciju nad podacima za učenje. Postoje omotači za više od 15 jezika, ali i nekoliko grafičkih sučelja za rad s bibliotekom.



Slika 3.7: Dijagram klasa

Adresa službene stranice: `http://leenissen.dk`
Dokumentacija: `http://leenissen.dk/fann/html/files/fann-h.html`

Pogledajmo sada kako izgleda rad s bibliotekom na jednostavnom primjeru aproksimiranja XOR logičke funkcije. Rad s neuronskom mrežom pod nadzorom učitelja može se podijeliti u tri osnovna koraka:

1. Generiranje skupa za učenje u obliku parova $ulaz \rightarrow izlaz$.
2. Podešavanje parametara mreže i treniranje mreže nad skupom za učenje.
3. Testiranje i iskorištavanje mreže.

U nastavku je prikazan primjer datoteke koja sadrži skup za učenje. Tri broja u prvom retku označavaju redom: broj parova ulaz-izlaz u datoteci, broj ulaznih podataka, broj izlaznih podataka. Ostatak datoteke predstavljaju podaci grupirani u parove ulaznih i izlaznih redaka. Zapisani podaci opisuju ponašanje XOR funkcije.

```
4 2 1
-1 -1
-1
-1 1
1
1 -1
1
1 1
-1
```

Pogledajmo sada programski kod koji najprije podešava parametre mreže, a zatim trenira istu s podacima za učenje.

```
const unsigned int br_ulaza = 2;
const unsigned int br_izlaza = 1;
const unsigned int br_slojeva = 3;
const unsigned int br_skrivenih_neurona = 3;
const float zeljena_pogreska = (const float) 0.001;
const unsigned int max_epochs = 500000;
//broj epoha koji mora proci prije svakog izvjestaja
const unsigned int epoha_izmedu_ispisa = 1000;
struct fann *ann = fann_create_standard(br_slojeva,
          br_ulaza, br_skrivenih_neurona, br_izlaza);
fann_set_activation_function_hidden(ann,
          FANN_SIGMOID_SYMMETRIC);
fann_set_activation_function_output(ann,
          FANN_SIGMOID_SYMMETRIC);
fann_train_on_file(ann, "xor.data", max_epochs,
          epoha_izmedu_ispisa, zeljena_pogreska);
fann_save(ann, "xor_float.net");
fann_destroy(ann);
```

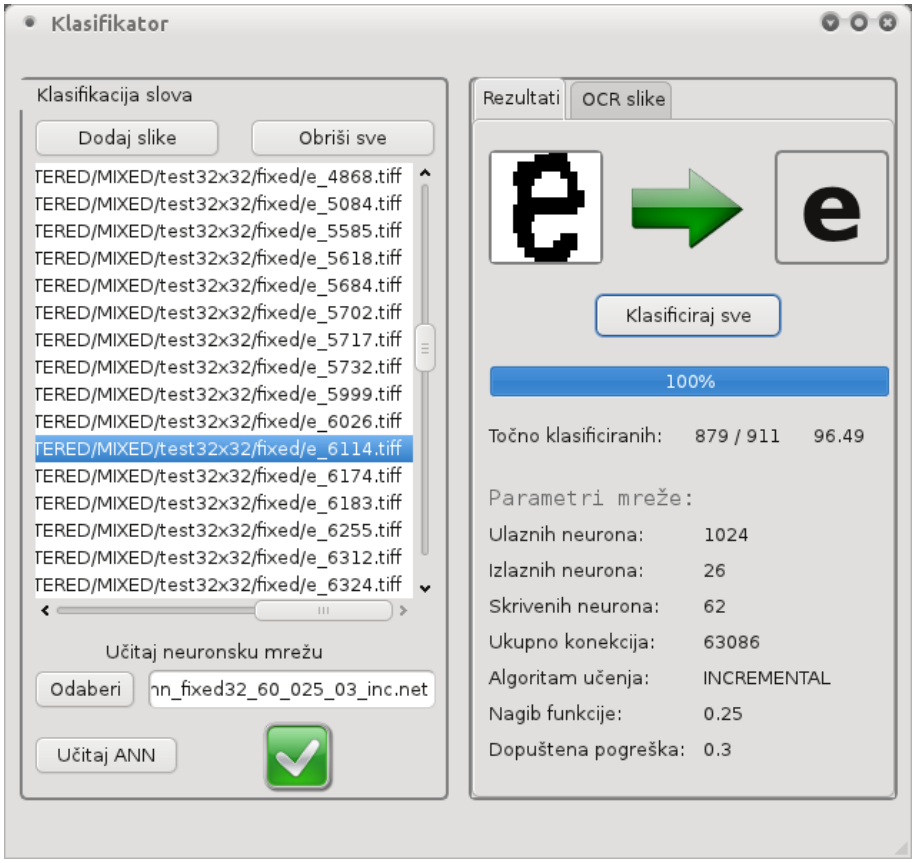
Napokon, nakon uspješnog treniranja, neuronska mreža je spremna za korištenje. Slijedeći kod prikazuje kako se učitava i koristi već naučena neuronska mreža.

```
fann_type *calc_out;   //izlaz mreze
fann_type input[2];    //ulazni podaci
//ucitava prethodno naucenu neuronsku mrezu
struct fann *ann = fann_create_from_file("xor_float.net");
input[0] = -1;
input[1] = 1;
//klasificira ulazne podatke i vraca izlaz
calc_out = fann_run(ann, input);
std::cout << "Izlaz je: " << calc_out[0] << std::endl;
...
fann_destroy(ann);
```

3.4.2. Učenje neuronske mreže i klasifikacija

Sada kada je savladan problem izgradnje i treniranja neuronske mreže, stečeno znanje potrebno je primijeniti u domeni problema kojeg obrađuje ovaj rad. Kako je prvi korak generiranje skupa za učenje, odlučeno je da ulaz mreže predstavljaju vrijednosti binariziranih piksela slike. Vidimo da će broj ulaza izravno ovisiti o rezoluciji slike pa se nameće pitanje koju rezoluciju odabrati. O odgovoru na ovo pitanje odlučit ćemo eksperimentalnom metodom. Prilikom generiranja slika, segmentirana slova spremana su u rezoluciji 32×32. Ovo je maksimalna rezolucija koja će biti testirana, a pored nje mreža će se trenirati i testirati i sa skaliranim slikama rezolucija 24×24 te 16×16 piksela. Izlazni sloj će imati 26 neurona, za svako slovo engleske abecede. Hrvatski dijakritički znakovi, velika slova kao i brojevi nisu smješteni u skup za učenje zbog ograničenih resursa. Naime, iako se skup slika sastojao od gotovo tisuću kamerom snimljenih prometnih znakova s tekstom, dosta ih sadržavalo iste znakove što je ograničilo raznolikost slova, pogotovo velikih koji obično dolaze samo kao prvo slovo riječi. Pored toga, jedan dio znakova je morao biti preskočen jer je, zbog premale rezolucije i prevelikog šuma uzrokovanog popratnim efektima snimanja, bio potpuno nečitljiv.

U svrhu pretvaranja slika iz slikovnog formata u format datoteke za učenje mreže opisane u prethodnom poglavlju, napisan je program generator.
`trainfile_generator.cpp` - ova programska komponenta generira datoteku za



Slika 3.8: Program klasifikator

učenje. Kao parametar prima datoteku s popisom slika čije vrijednosti piksela zatim čita i zapisuje kao ulaz. Budući da je prilikom spremanja slika u prvom slovu naziva naznačeno o kojem je slovu riječ, ovaj će podatak služiti za zapisivanje izlaza. Generator tako ide redom po slikama u listi generirajući pritom parove ulaza i izlaza.

`fann_train.cpp` - programska komponenta implementira učenje mreže. Naučena mreža sprema sve svoje interne podatke o neuronima i težinama u izlaznu datoteku.

`fann_test.cpp` - učitava spremljenu neuronsku mrežu i pokreće testiranje mreže. Prilagođena je generiranju raznih statističkih podataka.

Druga aplikacija razvijena u Qt radnom okviru služi za testiranje rada naučene neuronske mreže. Središnja komponenta aplikacije koja enkapsulira rad neuronske mreže je klasa `NeuralNetwork` prikazana u tablici [3.8].

`NeuralNetwork(std::string location)` - konstruktor prima lokaciju datoteke koja u zapisu sadrži podatke neuronske mreže te učitava mrežu u strukturu varijable `ann`.

Tablica 3.8: `NeuralNetwork` razred

NeuralNetwork
- ann: struct fann*
- data: struct fann_train_data*
- calc_out: fann_type*
- br_slova: int
+ NeuralNetwork(std::string location)
+ classify(std::string, char&, char&) : void
- fann_create_train_from_array(float*, float*, int, int, int) : struct fann_train_data*
- loadImage(std::string, float*, float*, int, int) : void
- getImageName(std::string path) : std::string

`classify` - metoda klasificira sliku slova s dane lokacije i sprema ASCII vrijednosti očekivanog izlaza i dobivenog izlaza na izlazu iz mreže.
`fann_create_train_from_array` - metoda alocira prostor i puni `fann_train_data` strukturu podataka podacima iz polja ulaznih podataka i polja izlaznih podataka.
`loadImage` - učitava sliku s dane lokacije u polje ulaznih podataka i polje očekivanog izlaza.

4. Eksperimentalni rezultati

Ostvareno programsko rješenje testirano je na skupu slika prometnih znakova koji su prethodno bili detektirani te segmentirani iz video zapisa snimanog kamerom iz automobila tijekom vožnje. Skup za učenje sastoji se od 873 slika malih slova bez dijakritičkih znakova dok skup za testiranje čini 911 slika.

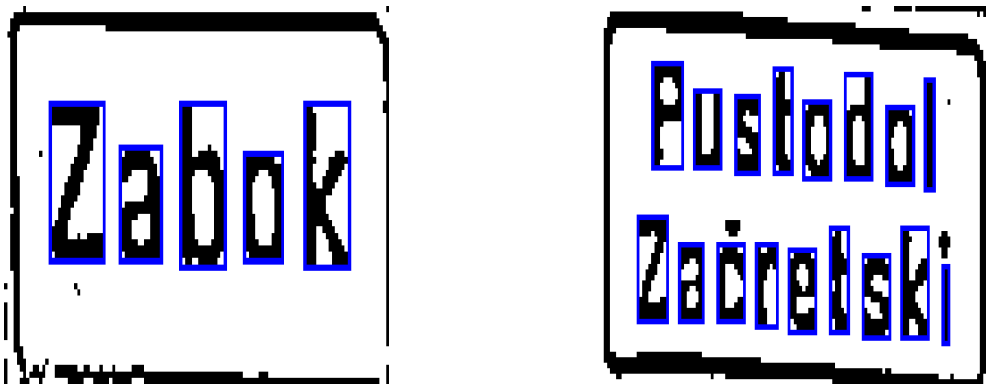
4.1. Detekcija i segmentacija teksta

Detekcija teksta pokazala je vrlo dobre rezultate na slikama prosječne kvalitete dok su uočeni problemi u radu sa zamućenim slikama malih rezolucija. Slika 4.1 prikazuje ispravne detekcije, na slikama 4.2a i 4.2b mogu se vidjeti primjeri pogrešnih detekcija dok je na slici 4.3 dan primjer loše binarizacije nad zamućenom slikom niske rezolucije. Zaključak je da bi se svakako dalo još poraditi na algoritmima filtriranja lažnih pozitivnih i negativnih detekcija u danjem radu. Treba primijetiti da jednom kada imamo naučenu neuronsku mrežu, možemo iskoristiti njeno znanje tijekom korištenja razvijenog sustava. U konkretnom primjeru, prilikom filtriranja lažno pozitivnih detekcija, raspon odstupanja visine slova jednog retka morao je biti znatno uvećan zbog slova *j* i *g*. Korištenjem neuronske mreže mogli bismo slučaj za ova slova izdvojiti zasebno, a za sva ostala bi se zatim moglo koristiti puno preciznije maksimalno odstupanje.

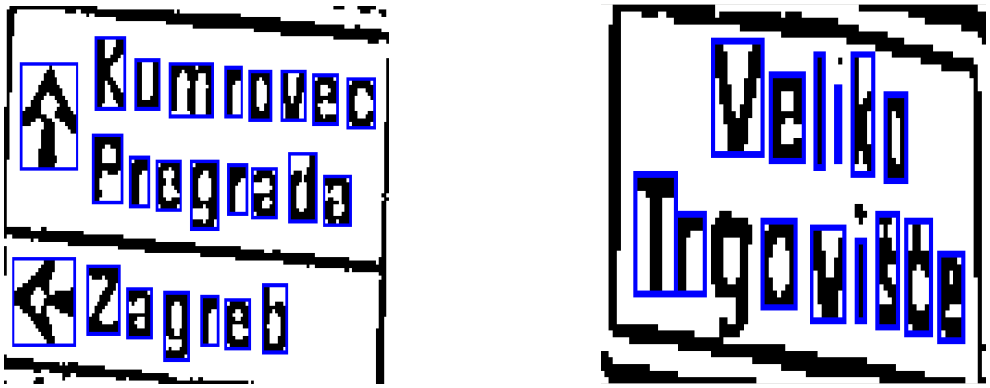
4.2. Algoritmi učenja i parametri mreže

Učenje je testirano s dvije različite varijante *backpropagation* algoritma:

`FANN_TRAIN_INCREMENTAL:` standardni *backpropagation* algoritam koji koristi pojedinačno učenje (engl. *on-line*), gdje se korekcija težina odvija nakon svakog predočenog primjera iz skupa za učenje. Nedostatak ovog algoritma je izrazita ovisnost brzine izvođenja o veličini skupa za učenje. Naime,



Slika 4.1: Primjeri ispravne detekcije



(a) *False positive* **(b)** *False negative*

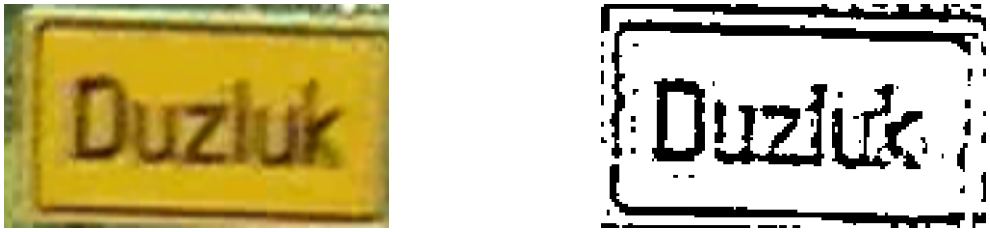
Slika 4.2: Primjeri neispravne detekcije

kako se korekcija odvija za svaki primjer u svakoj epohi učenja, algoritam će postajati sve sporiji što je veći skup za učenje.

`FANN_TRAIN_RPROP:` punog naziva *Resilient backPROPagation* je napredniji adaptivni algoritam koji, za razliku od prethodnog, koristi grupno učenje (engl. *batch*), kod kojeg se korekcija težina odvija tek na kraju svake epohe[1]. Na taj način će se unatrag propagirati suma svih pogrešaka na izlaznom neuronu za svaki takav neuron. Algoritam je adaptivan jer ne koristi parametar stope učenja, već ga sam računa i prilagođava. Više detalja o ovom algoritmu moguće je pronaći na: `http://en.wikipedia.org/wiki/Rprop`

Rezultati su prikazani u tablicama 4.1, 4.2, gdje svaka tablica prikazuje rezultate za jedan od dva navedena algoritma za učenje. Prvi stupac tablice sadrži broj neurona u skrivenom sloju testirane neuronske mreže, drugi stupac predstav-

[1]Epoha je jedno predočavanje svih uzoraka iz skupa za učenje.



Slika 4.3: Loša binarizacija zamućene slike

lja parametar dozvoljene pogreške, u trećem stupcu se nalazi parametar nagiba prijenosne funkcije i konačno, u četvrtom stupcu su navedeni postoci uspješnosti klasifikacije. Prije no što pogledamo rezultate objasnimo navedene parametre.

Parametar dozvoljene pogreške opisan je kao najveća dopuštena razlika između željenog izlaza i stvarnog izlaza na svakom izlaznom neuronu prilikom učenja. Svaki izlazni neuron čija je pogreška veća od dozvoljene se broji kao pogrešan *bit*. Treniranje se provodi sve dok je ukupan broj pogrešaka na *bitovima* cjelokupnog skupa za učenje veći od nule.

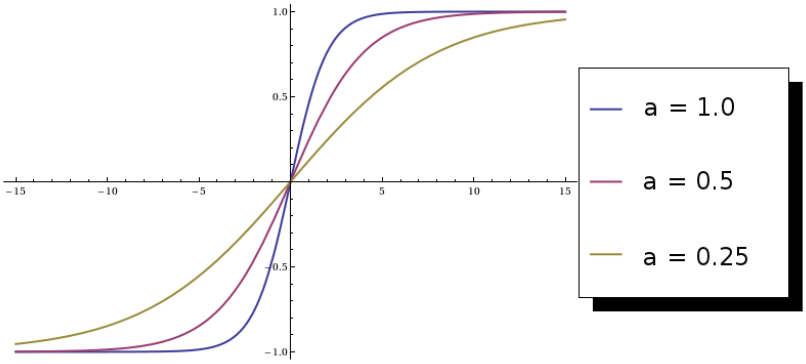
Kako bismo razumjeli parametar nagiba sigmoidalne prijenosne funkcije, prisjetimo se kako je ona matematički definirana.

$$f(net) = \frac{1}{1 + e^{-a \cdot net}}$$

Jedini parametar u funkciji je a i to je upravo parametar nagiba. Sada je stvar već mnogo jasnija jer je iz funkcije vidljivo da ovaj parametar izravno određuje brzinu rasta funkcije. Kako se uzima negativna vrijednost parametra, smanjenjem parametra će funkcija sporije rasti iz -1 u 1, dok će njegovo povećanje uzrokovati brži rast [4.4]. Podešavanjem ovog parametra pokušat ćemo ugoditi robusnost mreže i time donekle ublažiti posljedice nedovoljno velikog skupa za učenje koji zbog manjka raznolikosti može uzrokovati negativan efekt "štrebanja" uzoraka. Ako prijenosna funkcija sporije raste tada je manje osjetljiva na sitnije promjene ulaznih podataka što može učiniti mrežu otpornijom na šum. Naravno, treba paziti da se ne uzme premali nagib koji će izlaze početi grupirati oko srednje vrijednosti 0.

Primijetimo da mreža koristi skaliranu i translatiranu izvornu sigmoidalnu funkciju. Razlog tomu je to što izvorna sigmoidalna funkcija daje samo pozitivne izlaze veće od 0 i manje od 1. Korištena sigmoidalna funkcija simetrična s obzirom na ishodište te područja kodomene $[-1, 1]$ definirana je kao:

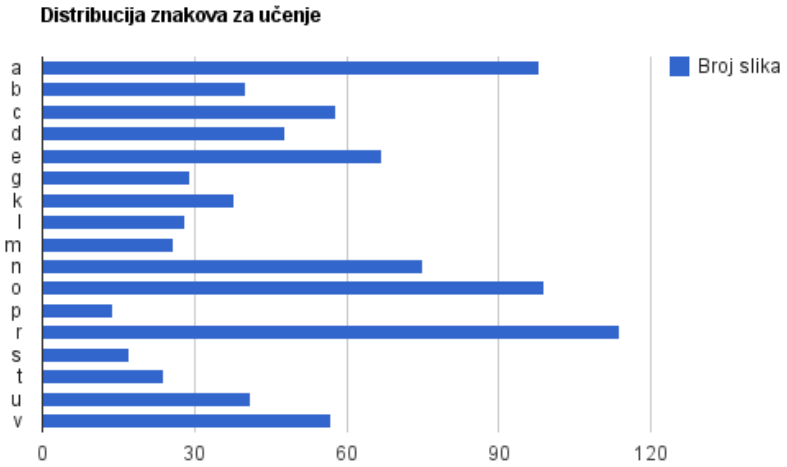
$$f(net) = \frac{2}{1 + e^{-a \cdot net}} - 1$$



Slika 4.4: Nagib sigmoidalne funkcije [4]

Testiranje je provedeno i nad slikama rezolucije 24×24 gdje je uspješnost u odnosu na rezultate prikazane u tablicama za slike veličine 32×32 bila u prosjeku lošija za 0.5% dok je za rezoluciju 16×16 zabilježen prosječan pad uspješnosti od oko 2%. Rezultati nam daju zaključiti da bi optimalne rezolucije bile 24×24 ili 16×16, ovisno o zahtjevima performansi. Dodatno je testirana i rezolucija 8×8 nad kojom je uspješnost klasifikacije pala za više od 15% te je treniranje mreže postalo vrlo problematično jer bi prečesto postupak zapeo u jednom od lokalnih minimuma gdje se nekoliko izlaznih vrijednosti za neke primjere slika jednostavno ne može svesti na dopušteno odstupanje (dopuštenu pogrešku).

4.2.1. Distibucija slika i uspješnosti po znakovima



Slika 4.5: Distribucija slika znakova za učenje

Tablica 4.1: Neuronska mreža sa 1024 ulaza (32×32) trenirana algoritmom FANN_TRAIN_RPROP

Skrivenih neurona	Dozvoljena pogreška	Nagib funkcije	Uspješnost
30	0.01	1.00	91.33 %
30	0.01	0.50	91.11 %
60	0.01	1.00	92.86 %
60	0.01	0.50	94.29 %
60	0.01	0.25	93.85 %
60	0.1	1.00	95.06 %
60	0.1	0.25	96.06 %
60	0.3	1.00	96.38 %
60	0.3	0.25	96.60 %
150	0.1	1.00	94.73 %
400	0.1	1.00	95.28 %

U ovom potpoglavlju prikazane su distribucije slika prema klasama znakova. Na slici 4.5 dana je distribucija slika znakova koji su korišteni za učenje mreže dok se u tablici

Tablica 4.3: Distribucija slika i uspješnosti znakova za testiranje

Znak	Količina	Uspješnost	Znak	Količina	Uspješnost
a	185	97.3 %	n	75	96 %
b	13	100 %	o	79	88.6 %
c	111	98.2 %	p	9	100 %
d	19	100 %	r	89	100 %
e	62	100 %	s	12	100 %
g	17	76.5 %	t	33	100 %
k	44	97.7 %	u	35	94.3 %
l	19	94.8 %	v	86	100 %
m	23	78.3 %			

4.2.2. Matrica zabune

Matrica zabune pokazuje distribuciju pogrešaka. U retcima su upisana slova koja se nalaze na ulazu dok su u stupcima smještena slova koja su rezultat kla-

Tablica 4.2: Neuronska mreža sa 1024 ulaza (32×32) trenirana algoritmom FANN_TRAIN_INCREMENTAL

Skrivenih neurona	Dozvoljena pogreška	Nagib funkcije	Uspješnost
30	0.01	1.00	94.84 %
30	0.01	0.50	93.11 %
60	0.01	1.00	95.17 %
60	0.01	0.25	96.16 %
60	0.1	1.00	94.56 %
60	0.1	0.25	95.84 %
60	0.3	0.25	96.79 %
60	0.3	0.25	96.60 %
150	0.1	1.00	94.51 %
400	0.1	1.00	95.28 %

sifikacije. Vrijednost elementa matrice govori koliko je puta slovo u tom retku pogrešno klasificirano kao slovo u tom stupcu. Matrica je dana u tablici 4.4. Uspoređujući matricu zabune i tablicu uspješnosti klasifikacije 4.3 vidljivo je da najveću pogrešku mreža radi za slova *m* i *g*. Boljim promatranjem uzoraka za učenje i testiranje otkriveno je da ta dva slova imaju velika odstupanja omjera visine i širine. Za takav problem korištena strategija spremanja sa smještanjem slova u centar slike uz zadržavanje omjera nije dobro rješenje. Mnogo bolje bi u ovom slučaju koristiti jednostavno skaliranje slike do tražene rezolucije bez očuvanja omjera (Slika 3.5). Slika 4.6 prikazuje usporedbu lošijeg i boljeg načina.

(a) Primjer m1 uz očuvanje omjera: široko slovo m

(b) Primjer m2 uz očuvanje omjera: usko slovo m

(c) Primjer m1 uz skaliranje bez očuvanja omjera

(d) Primjer m2 uz skaliranje bez očuvanja omjera

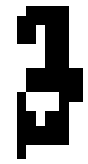
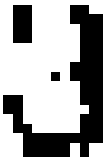
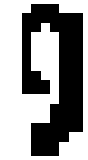
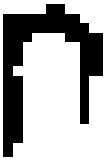
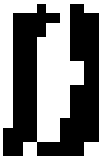
Slika 4.6: Skaliranje slika bez zadržavanja omjera

Tablica 4.4: Matrica zabune

	a	c	e	g	k	l	o	p	r	u	v
a			2		1		1				1
c							2				
g							1			2	1
k								1			
l									1		
m	1					2			2		
n			1						2		
o	1	3		1						4	
u							1				

4.2.3. Primjeri pogrešnih klasifikacija

Tablica 4.5: Primjeri pogrešnih klasfikacija.

Slika slova	Točno	Prepoznato
	a	e
	a	v
	g	o
	n	r
	o	u

5. Zaključak

U okviru ovog rada ostvaren je postupak detekcije i segmentacije teksta upotrebom postupka analize povezanih komponenata koji izravno ovisi o implementiranim algoritmima binarizacije slike. Drugi dio rada obuhvatio je klasifikaciju dobivenih slova te je ona ostvarena pomoću umjetne neuronske mreže. Eksperimentalni rezultati pokazali su dobru uspješnost klasifikacije za različite veličine slika. Osim toga, pokazali su da sama uspješnost u manjoj mjeri ovisi o parametrima mreže ako se oni nalaze u određenim granicama. Rezultati detekcije su ukazali da ona uvelike ovisi o prethodnom pretprocesiranju slike, stoga bi bilo dobro uvesti dodatno uklanjanje šuma kako bi se eliminirala mogućnost spajanja više slova u jedno kao i razdvajanja jednog slova na više njih.

U budućem radu uspješnost klasifikacije bi se mogla dodatno poboljšati dodavanjem kontekstne analize. U korist ovog prijedloga ide i činjenica da Hrvatska uistinu nema mnogo gradova i naselja što uvelike olakšava prikupljanje podataka za kontekstnu analizu. Pored naziva gradova i naselja, u analizu se mogu uvrstiti i ostale riječi jezika koje se pojavljuju na prometnim znakovima. Još jedan pravac za budući rad može biti korištenje znanja neuronske mreže kako bi se poboljšao postupak filtriranja lažnih rezultata u postupku detekcije slova.

Literatura

[1] *OpenCV documentation.* URL `http://opencv.itseez.com/modules/refman.html`.

[2] *Thresholding.* URL `http://en.wikipedia.org/wiki/Thresholding_(image_processing)`.

[3] Jeff Heaton. *Introduction to Neural Networks in Java.* Heaton Research, 2008.

[4] Steffen Nissen. *Neural Networks Made Simple*, 2005. URL `http://fann.sourceforge.net/fann_en.pdf`.

[5] Peter Norvig Stuart Russell. *Artificial Intelligence: A modern approach*, stranice 563–596. Prentice Hall, 1995.

[6] Šnajder J. Čupić M., Dalbelo Bašić B. *Umjetne neuronske mreže, Službeni materijali s predavanja iz predmeta Umjetna inteligencija.* Fakultet elektrotehnike i računarstva, Zagreb, 2008. URL `http://www.fer.hr/_download/repository/UmjetneNeuronskeMreze.pdf`.

Strojno očitanje natpisa na prometnim znakovima

Sažetak

Rad obuhvaća razvoj postupka za strojno očitanje natpisa na prometnim znakovima za koje pretpostavljamo da su prethodno detektirani nekom drugom metodom iz skupa slika pribavljenih vozilom u pokretu. Nad slikama se najprije provodi pretprocesiranje čiji je glavni zadatak binariziranu sliku predati algoritmima za detekciju slova. Kod detekcije je bitno istaknuti korištenje algoritma analize povezanih komponenata, a naknadno se provodi i filtriranje lažnih detekcija. Detektirana slova u slikama se segmentiraju i spremaju u slikovnom zapisu. Nakon prikupljanja dovoljne količine slika započinje postupak klasifikacije koji je ostvaren implementacijom neuronske mreže. Skup slika se najprije dijeli na dva zasebna skupa: skup za učenje i skup za testiranje. Skup za učenje poslužit će kao skup primjera za treniranje mreže, a nakon toga će se trenirana mreža evaluirati pomoću skupa za testiranje. Prikazani su i komentirani eksperimentalni rezultati evaluacije mreža treniranih s različitim vrijednostima parametara.

Ključne riječi: neuronske mreže, strojno učenje, OpenCV, računalni vid, binarizacija, Qt C++, analiza povezanih komponenata, strojno očitanje teksta

Optical character recognition on traffic signs

Abstract

This work includes the development of optical character recognition tool for reading the inscription on the road signs. We assume that the road signs have been previously detected by another method in images obtained from a moving vehicle. In preprocessing phase images are first binarized and then sent to letter detection algorithms. Detection phase uses connected component analysis method after which it performs the filtering of false positives. Detected letters are individually stored in the image format. After collecting sufficient amounts of letter examples, the process of image classification is achieved by implementing an artificial neural network. Set of images is now divided into two separate subsets: a set for learning and set for testing. A set for learning will serve as a set of examples to train the network which will then be evaluated using the images from set for testing. Experimental results are presented and discussed at the end.

Keywords: neural networks, machine learning, computer vision, OpenCV, binarization, Qt C++, connected component analysis, OCR