UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Gregor Jeraj

Prepoznavanje prometnih znakov v slikah

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Uroš Lotrič

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01726/2011
Datum: 15.02.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **GREGOR JERAJ**

Naslov: **PREPOZNAVANJE PROMETNIH ZNAKOV V SLIKAH**
TRAFFIC SIGN RECOGNITION SYSTEM

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Med voznikovimi elektronskimi pripomočki se vedno pogosteje pojavljajo sistemi za razpoznavanje prometnih znakov kot pomoč pri varni vožnji. V nalogi zasnujte programsko opremo za lasten sistem, ki vključuje lociranje znakov v zajetih slikah, njihovo kategorizacijo in prepoznavanje. Obnašanje sistema preverite na realnih podatkih.

Mentor:

izr. prof. dr. Uroš Lotrič

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Gregor Jeraj,

z vpisno številko 63040059,

sem avtor diplomskega dela z naslovom:

Prepoznavanje prometnih znakov.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  prof. dr. Uroša Lotriča
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 11.3.2011 Podpis avtorja/-ice:

Zahvala

Na prvem mestu bi se rad zahvalil mentorju, prof. dr. Urošu Lotriču, ki mi je pomagal kadarkoli sem pri diplomi potreboval pomoč, prof. dr. Bojanu Orlu za pomoč pri diferencialnih enačbah, ter vsem, ki so kakorkoli pripomogli k uspešni izvedbi diplomskega dela.

Kazalo

Povzetek

Vse več voznikov se zanaša na različne pripomočke za pomoč pri vožnji. Eden od teh je tudi sistem prepoznavanja prometnih znakov, ki voznikom, glede na prometne znake na cesti, prikazuje informacije in ga obvešča o bližajočih nevarnostih in prepovedih, bodisi z govorom, bodisi z sporočilom na armaturni plošči. Samodejno prepoznavanje znakov voznika pravilno obvešča o bližajočih znakih in s tem preprečuje morebitne nesreče, ki so posledica nepozornosti ali nepoznavanja prometnih znakov. Sistem prepoznavanja prometnih znakov je obvezni del avtonomnih vozil, saj se ti pri pravilni vožnji v veliki meri zanašajo na prometne znake.
V diplomskem delu je predstavljen sistem prepoznavanja prometnih znakov v slikah, ki je sestavljen iz dveh glavnih delov: modul za lociranje znakov ter modul za klasifikacijo znakov. Modul za lociranje znakov sliko barvno segmentira, nato glede na določen tip znaka poišče ključne točke, ki so potrebne za določitev oblike znakov. Modul za klasifikacijo pa s pomočjo nevronskih mrež notranjost znaka klasificira v razrede.

Ključne besede:

nevronske mreže, prometni znaki, konvolucija, maske, klasifikacija znakov

Abstract

More and more devices are used in assisting drivers on the road. One of them is traffic sign recognition system. It uses a front mounted video camera and informs the driver of incoming dangers and restrictions on the road, mainly by speech or special dashboard display. Traffic sign recognition system is used to avoid accidents, caused by drivers inability to properly recognize traffic sign. They are also a major part of autonomous vehicle systems since they are responsible for providing safe driving instructions.
We proposed an intelligent traffic sign recognition system, which is based on two main parts: traffic sign location and classification module. First one performs color segmentation and locates traffic sign key points, which are later used to form traffic sign shape. Second module then uses neural network approach to classify shapes into traffic signs.

Key words:

neural networks, traffic signs, convolution, masks, traffic sign classification

Poglavje 1

Uvod

Prepoznavanje prometnih znakov predstavlja del problema avtonomnih vozil. Ta vozila se pri sprejemanju odločitev v prometu zanašajo izključno na sistem prepoznavanja objektov s pomočjo računalniškega vida. Sistem mora učinkovito in v realnem času dobavljati potrebne podatke za nadzorovanje volana, pedala za plin, zavore... Pri tem mora:

- prepoznavati vozne pasove, po katerih vozilo lahko potuje,
- prepoznati ovire na cesti, da se jim sistem lahko izogne, ter
- prepoznati in interpretirati prometne znake, ki zagotavljajo varno vožnjo v prometu.

Poleg uporabe v avtonomnih vozilih, se sistem za prepoznavanje prometnih znakov lahko uporablja tudi na drugih področjih. Lahko so samo v pomoč vozniku. V tem primeru sistem voznika obvešča o približujočih se prometnih znakih in mu svetuje nadaljnje ravnanje na cesti.
Prometni znaki obveščajo voznike o pravilnem načinu vožnje in mogočih nevarnostih na cesti. Njihova oblika in značilne barve omogočajo voznikom, da jih lahko hitro prepoznajo in identificirajo tudi ob slabših vidnih pogojih.
Obstaja več sistemov za prepoznavanje prometnih znakov. Enega od teh so razvili v podjetju Daimler-Benz [1] v okviru sistema za preprečevanje trkov. Ta sestoji iz treh glavnih delov:

- barvna segmentacija, ki glede na barvne lastnosti znakov najde možne kandidate za znake,
- prepoznavanje oblike znakov, ki loči kandidate glede na njihovo obliko, in

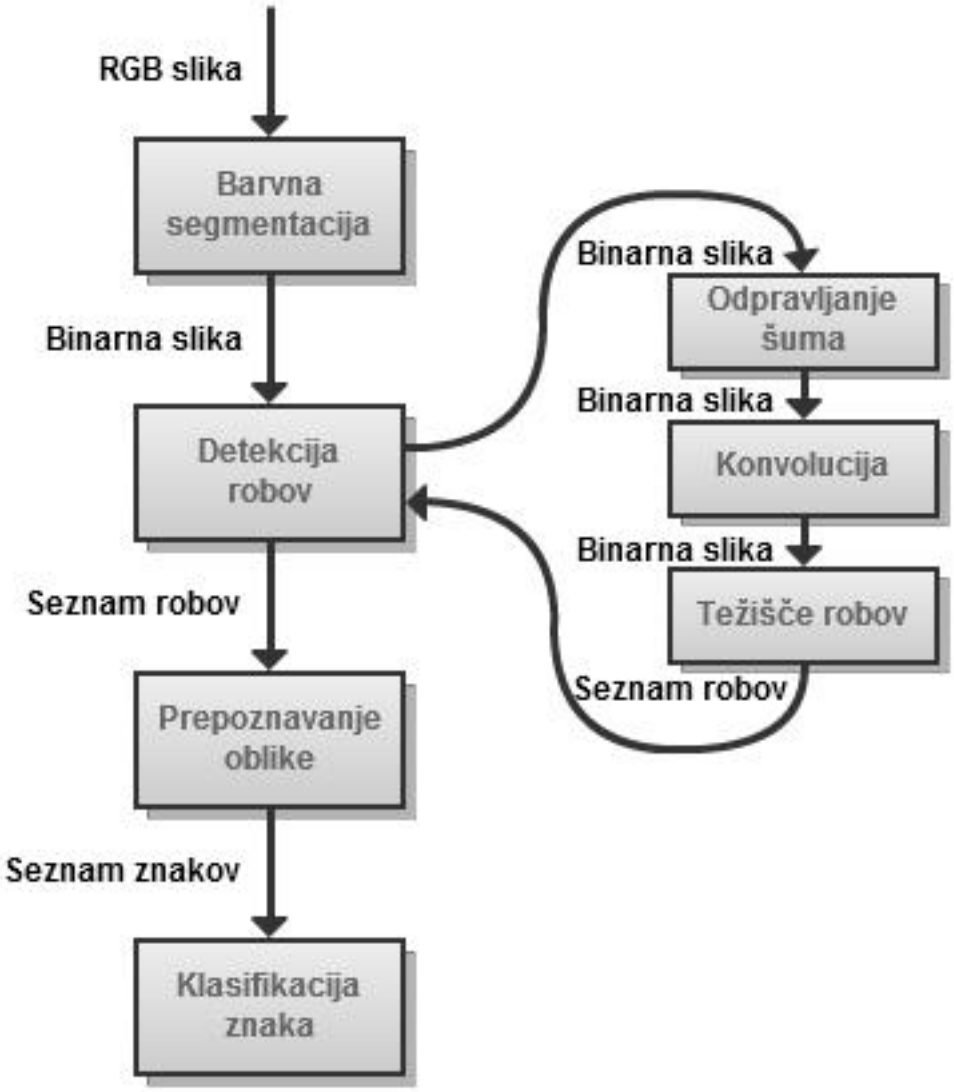
- prepoznavanje tipa znaka, ki primerja kandidate z bazo znakov in določi tip znaka.

V idealnih pogojih prepoznavanje prometnih znakov ni posebej težko delo, saj imajo vsi znaki prepoznavne barve in oblike. V neidealnih pogojih sistem otežujejo naslednji problemi:

- veliko število znakov z različnimi oblikami, barvami in simboli,
- spremenljivi pogoji na cesti (sonce, sence, zastrtost, rotacija...) in
- procesiranje v realnem času.

Tako kot veliko ostalih problemov prepoznavanja objektov, tudi tega lahko razdelimo na lociranje in razpoznavanje v sliki. Opisan sistem je podoben kot sistem razvit v podjetju Daimler-Benz. Razdelimo ga lahko na štiri korake:

- v koraku **barvne segmentacije**, sistem filtrira določen razpon barv na robu znaka (v našem primeru nas zanimajo samo rdeče barve). Rezultat tega koraka je slika, sestavljena iz belih (rdeča barva) in črnih (ostale barve) točk,
- pri **detekciji robov** s pomočjo konvolucije z maskami v sliki najdemo koordinate robov znaka,
- v koraku **prepoznavanja oblik**, glede na geometrijske značilnosti, robove iz prejšnjega koraka združimo v znake in
- v koraku **razvrščanja** z uporabo nevronskih mrež in barvne informacije prepoznamo tip znaka.



Slika 1.1: Koraki algoritma.

Slika 1.1 prikazuje korake sistema. Ker je število različnih prometnih znakov zelo veliko, se v tej nalogi omejimo na stop znake, znake za križišče s prednostno cesto, znake za nevarnost in znake za prepoved. Stop znaki so osemkotniki, znaki za križišče s prednostno cesto in znaki za nevarnost so trikotni, medtem, ko so znaki za prepoved okrogli. Sistem se lahko razširi na vse preostale znake, saj za njihovo prepoznavanje lahko uporabimo podoben algoritem, le z drugačnimi operatorji. Na trenutni stopnji razvoja ima naš sistem precej

omejitev:

- število različnih znakov je omejeno na znake za nevarnost in prepoved,
- sistem ne podpira prevelike rotacije znakov,
- sistem ne zazna znakov, ki so preveč stisnjeni (omejen je vidni kot),
- sistem ne zazna znakov, ki so delno zakriti (vsi robovi znaka morajo biti vidni) in
- kljub temu, da je sistem namenjen za procesiranje v realnem času, to, zaradi kompleksnosti izvedbe, ni prioriteta.

Poglavje 2

Barvna segmentacija

2.1 Konvolucija

Konvolucija je matematični operator med dvema funkcijama v zveznem ali diskretnem prostoru. Tukaj se bomo osredotočili na diskretni prostor, saj slika predstavlja skupek diskretnih točk v dvodimenzionalnem prostoru. Diskretno konvolucijo podaja enačba

$$G(x,y) = g(x,y) \star f(x,y) = \sum_{s=-M/2}^{M/2} \sum_{t=-M/2}^{M/2} g(s,t) f(x+s, y+t), \qquad (2.1)$$

kjer je

- g konvolucijska maska, ki je dvodimenzionalna matrika,
- M velikost konvolucijske maske ter
- f slika nad katero izvajamo operacijo.

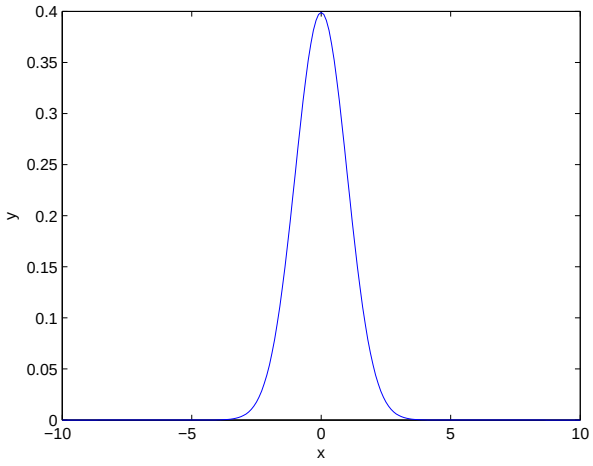
Operacijo si najlažje predstavljamo kot drsenje maske po sliki. V vsaki točki slike naredi linearno kombinacijo sosednjih točk, ki jih maska prekriva in shrani vsoto v novo sliko $G(x,y)$.

2.2 Glajenje slike

Slike v neidealnih pogojih vsebujejo nezaželen šum. Te anomalije lahko zmanjšamo z Gaussovim glajenjem slike. Ta s pomočjo konvolucije sliko zamegli in s tem zmanjša šum. V enodimenzionalnem prostoru ima Gaussova porazdelitev obliko

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}, \tag{2.2}$$

kjer σ predstavlja standardno odstopanje od povprečne vrednosti.



Slika 2.1: Gaussova porazdelitev z $\sigma = 1$.

Vhodne slike so dvodimenzionalne, zato potrebujemo dvodimenzionalno Gaussovo porazdelitev

$$f(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}. \tag{2.3}$$

Tehnično gledano gre pri glajenju za uporabo dvodimenzionalne maske in konvolucije. Ker je slika sestavljena iz diskretnih točk, moramo primerno aproksimirati tudi 2D Gaussovo porazdelitev. V teoriji je porazdelitev vedno različna od nič, kar bi pomenilo izjemno veliko konvolucijsko jedro. Če vrednosti, ki so dovolj blizu 0 zanemarimo, nam ostane maska velikosti 5x5. Primer maske prikazuje tabela 2.2.

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

Tabela 2.1: Diskretna Gaussova porazdelitev z $\sigma = 1$.

Dobljeno masko nato le konvoluiramo s sliko in dobimo zglajeno sliko z manj šuma.

2.3 Barvna segmentacija

Barvna informacija v sliki nam je v veliko pomoč pri prepoznavanju znakov, saj je neodvisna od rotacije in velikosti znakov. Izbrana skupina prometnih znakov ima rob v živo rdeči barvi. To lahko izkoristimo tudi pri lociranju znaka v sliki. V sliki pobarvamo belo tiste točke, ki imajo delež rdeče barve dovolj velik, medtem, ko ostale točke pobarvamo črno. To močno zmanjša kompleksnost slike za nadaljnjo procesiranje. Opisano segmentacijo v prostoru RGB podaja funkcija:

$$S(x,y) = \begin{cases} 1 & \text{če } R_{min} \geq f_r(x,y) \geq R_{max} \\ 0 & \text{drugod} \end{cases}, \tag{2.4}$$

kjer so

- $S(x,z)$ izhodna vrednost segmentacije v točki s koordinatama x, y,
- $f_r(x,y)$, $f_g(x,y)$, $f_b(x,y)$ vhodna vrednost rdeče zelene in modre komponente prostora RGB in
- R_{min}, R_{max} mejni vrednosti rdeče komponente.

Glavni problem prostora RGB je velika občutljivost na spremembo svetlosti (npr. pri oblačnem vremenu, soncu...). Že majhna sprememba v svetlosti bi lahko v enačbi 2.4 povzročila veliko spremembo rdeče komponente, kar bi privedlo do napačne segmentacije.
Mnogo bolj primerna je uporaba prostora HSI. Ta prostor loči tri komponente: barvo (H), nasičenost z belo svetlobo (S) in svetlost (I). Funkcija segmentacije je podobna kot v prejšnjem primeru:

$$S(x,y) = \begin{cases} 1 & \text{če } H_{min} \geq f_h(x,y) \geq H_{max} \\ 0 & \text{drugod} \end{cases} , \tag{2.5}$$

pri čemer so

- $S(x,y)$ izhodna vrednost segmentacije v točki s koordinatami x, y,
- $f_h(x,y)$ vhodna vrednost komponente prostora HSI, ki predstavlja barvo, ter
- H_{min}, H_{max} mejni vrednosti barvne komponente.

W Pretvorba v prostor HSI zahteva uporabo nelinearnih in trigonometričnih funkcij, ki so časovno zelo zahtevne operacije. Želenemu rezultatu se dovolj približamo, če namesto samih komponent prostora RGB delamo z deleži teh komponent proti vsoti vseh komponent:

$$S(x,y) = \begin{cases} 1 & \text{če } P_{min} \geq \frac{f_r(x,y)}{f_r(x,y)+f_g(x,y)+f_b(x,y)} \geq P_{max} \\ 0 & \text{drugod} \end{cases} , \tag{2.6}$$

pri čemer so

- $S(x,y)$ izhodna vrednost segmentacije v točki s koordinatami x, y,
- $f_r(x,y)$, $f_g(x,y)$ in $f_b(x,y)$ vhodna vrednost rdeče, zelene in modre komponente ter
- P_{min}, P_{max} mejni vrednosti deleža rdeče barve.

Primer segmentacije prikazuje slika 2.2. Na levi strani slike je nesegmentirana, na desni pa segmentirana slika, na kateri se lepo vidi rdeč rob znaka.

Slika 2.2: Vhodna (levo) ter segmentirana slika (desno).

Poglavje 3

Detekcija robov

Sedaj imamo na voljo črno-belo sliko. V tej sliki bele točke predstavljajo rdeč rob znakov, medtem ko črne točke predstavljajo vse ostale barve. Naša naloga je najti koordinate kotov znaka v sliki. Ti koti nam kasneje omogočajo prepoznavanje oblike znakov.

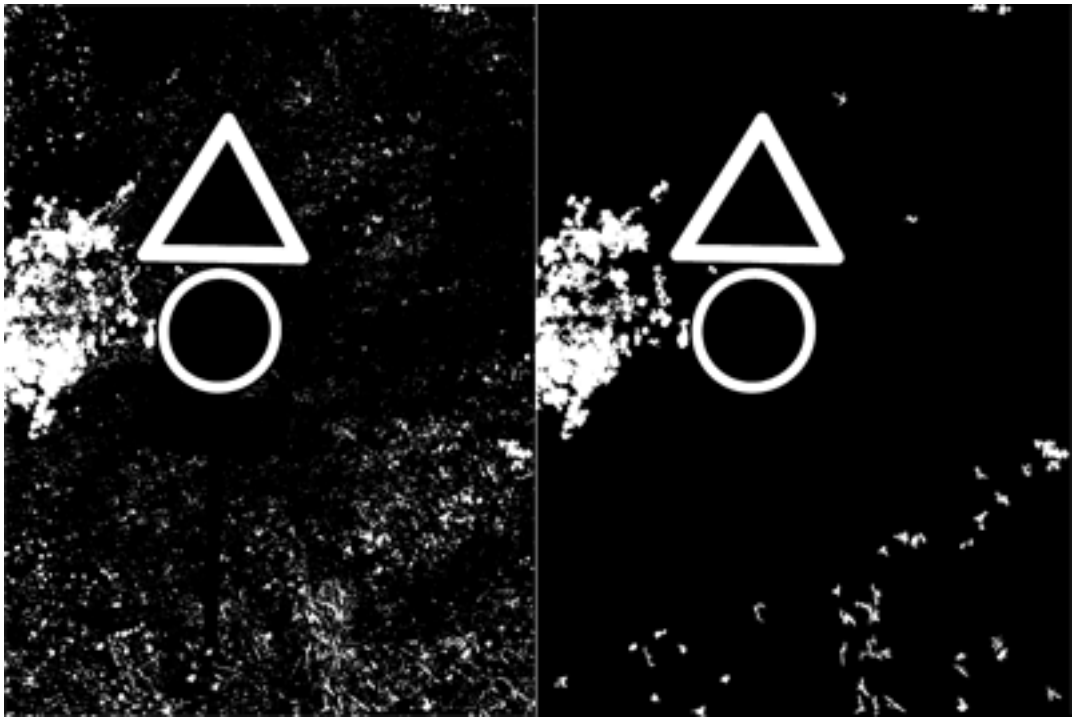
3.1 Odstranjevanje šuma

Preden se lotimo lociranja kotov znakov, moramo odstraniti šum. Ker smo predpostavili nezakritost in minimalno velikost znakov, mora skupek rdečih točk, ki določa rob znaka, obsegati vsaj določeno število točk. Evidentno lahko iz slike odstranimo vse skupke belih točk, ki so manjši od tega števila. Uporabimo lahko rekurzivni algoritem, ki ga ponazorjuje algoritem 3.1.

```
foreach tocka in slika
  if tocka.obiskana and tocka.bela then
    skupek = velikost( tocka, 0 )

    if skupek > prag then
      pobrisi skupek
    end
  end
end

function velikost( tocka, crne )
  s = 0

  if tocka.bela then
    crne = crne + 1

  if tocka ni rob and crne < 2 then
    tocka.obiskana = true
    s += velikost( slika[ tocka.x, tocka.y + 1 ], crne )
    s += velikost( slika[ tocka.x, tocka.y - 1 ], crne )
    s += velikost( slika[ tocka.x + 1, tocka.y ], crne )
    s += velikost( slika[ tocka.x - 1, tocka.y ], crne )
  end

  return s
end
```

Algoritem 3.1: Algoritem za odstranjevanje šuma

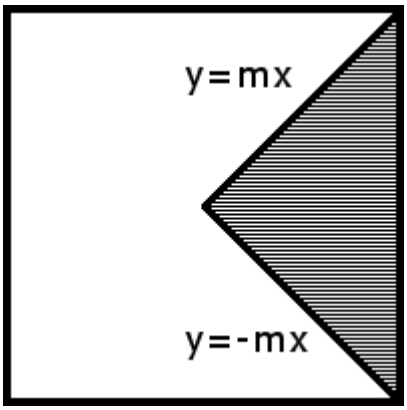
Algoritem gre skozi vse točke v sliki in išče bele pike. Za vsako belo piko, ki jo najde, rekurzivno preveri še njene sosede. Tako počasi povečuje velikost skupka. Rekurzija se prekine, če naleti na dva zaporedna črna soseda. Če je skupek pik premajhen, ga zbriše (pobarva vse bele pike v skupku črno). Primer segmentirane slike z odstranjenim šumom prikazuje slika 3.1.



Slika 3.1: S šumom (levo) in brez šuma (desno).

3.2 Optimalna detekcija robov

Znaki imajo določeno geometrijsko obliko. To lahko izkoristimo in za vsak kot v znaku, ustvarimo svojo konvolucijsko masko. Primer so znaki za nevarnost. Ti imajo tri kote, za kar potrebujemo tri maske. Torej imamo za vsak kot svojo masko, kar nam omogoča klasificiranje kotov in zmanjšuje kompleksnost algoritma. Rangarajan [2] predlaga naslednji način definicije optimalne konvolucijske maske.



Slika 3.2: Model kota.

Za modeliranje kota uporabimo notranji rob znaka, saj vemo, da je notranjost znaka vedno bela. S tem se izognemo vplivu okolice znaka na rob, saj bi v primeru uporabe zunanjega roba in rdečega ozadja, prehoda med barvami ne bi bilo, kar bi onemogočilo detekcijo robov. V enačbi modela 3.1 spremenljivka $I(x, y)$ predstavlja črno belo sliko, medtem ko m naklon kota.

$$I(x,y) = \begin{cases} 1, & \text{če} - mx < y < mx \\ 0, & \text{drugod} \end{cases} \tag{3.1}$$

Naj bo $g(x, y)$ maska, ki jo iščemo, in $n(x, y)$ bel Gaussov šum. Potem je funkcija slike s šumom definirana z

$$F(x,y) = I(x,y) + n(x,y). \tag{3.2}$$

Iščemo masko $g(x, y)$, s katero konvoluiramo sliko in s tem dobimo sliko možnih kotov znakov. Možnih mask $g(x, y)$, ki to lahko naredijo, je veliko, vendar niso vse optimalne. Zanimajo nas tiste maske, ki najbolje ustrezajo naslednjim kriterijem:

- maska ne sme biti občutljiva na šum (razmerje signal šum mora biti veliko),
- ne sme delocirati kota, kar pomeni, da ne premakne kota iz dejanske lokacije, kar izrazimo s matematičnim upanjem $E[x_0^2 + y_0^2]$,
- točka, v kateri je kot, mora biti tudi rob znaka in
- v vsaj dveh sosednjih točkah mora biti rob z drugačno orientiranostjo, kar pomeni, da mora biti kot presek dveh premic z različnima naklonoma.

Dani problem moramo kvantizirati. Canny [3] predlaga razmerje signal šum (SNR) za prvi, in $E[x_0^2 + y_0^2]$ za drugi kriterij (Λ).

$$SNR = \frac{A \int_0^\infty \int_{-mx}^{mx} g(x,y)dydx}{n_0 \sqrt{\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g^2(x,y)dydx}} \tag{3.3}$$

$$\Lambda = \frac{n_0^2 \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g_x^2(x,y)dydx}{(A \int_0^\infty \int_{-mx}^{mx} g_{xx}(x,y)dydx)^2} + \frac{n_0^2 \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g_y^2(x,y)dydx}{(A \int_0^\infty \int_{-mx}^{mx} g_{yy}(x,y)dydx)^2}, \tag{3.4}$$

kjer je n_0^2 varianca Gaussovega šuma $n(x,y)$, m velikost maske in g_x, g_y, g_{xx}, g_{yy} delni odvodi maske $g(x,y)$. V idealnem primeru bi bilo razmerje signal šum zelo visoko in delokalizacija (Λ) 0. Lahko zapišemo:

$$\sum = \frac{SNR}{\Lambda}. \tag{3.5}$$

Ta izraz moramo maksimizirati. To lahko storimo z variacijskim računom, Lagrangevimi multiplikatorji in Eulerjivim računom. Dobimo enačbo

$$g(x,y) = c_1 \sin\left(\frac{p\pi x}{W}\right)[-(e^{zW} + e^{-zW}) + e^{zy} + e^{-zy}], \tag{3.6}$$

ki velja za del maske s črnimi točkami (del maske, ki prekriva kot), in enačbo

$$g(x,y) = c_2 \sin\left(\frac{n_1\pi x}{W}\right) \sin\left(\frac{n_2\pi y}{W}\right), \tag{3.7}$$

ki velja za del maske z belimi točkami (del maske, ki ne prekriva kota). V enačbah spremenljivka W predstavlja velikost maske, medtem ko so c_1, c_2, p, n_1, n_2 in z konstante. Ti dve enačbi sta samo del rešitev, ki jih predstavlja neskončna Fourierjeva vrsta teh členov. Idealno mora del maske, ki prekriva kot, povečevati uteženo vsoto maske, medtem, ko jo ostali del zmanjšuje. Sama magnituda koeficientov ni tako pomembna, važno je le, da je konstantna. V najenostavnejšem primeru lahko vzamemo magnitudo 1 (območje kota) in -1 (območje brez kota). S tem povečamo hitrost algoritma, saj ni potrebno

množenje maske s točkami, le seštevanje in odštevanje. Toleranco maske na nepopolne kote pa spreminjamo s pragom utežene vsote koeficientov.

-1	-1	-1	-1	-1	-1	-1	-1	1
-1	-1	-1	-1	-1	-1	-1	1	1
-1	-1	-1	-1	-1	-1	1	1	1
-1	-1	-1	-1	-1	1	1	1	1
-1	-1	-1	-1	1	1	1	1	1
-1	-1	-1	-1	-1	1	1	1	1
-1	-1	-1	-1	-1	-1	1	1	1
-1	-1	-1	-1	-1	-1	-1	1	1
-1	-1	-1	-1	-1	-1	-1	-1	1

Tabela 3.1: Primer maske za 60° kot.

3.3 Težišče kota

Slike v neidealnem svetu nimajo idealnih kotov. Zato se pri detekciji z masko pojavi več odzivov (več navideznih kotov) za isti kot. Ti koti sicer ustrezajo pogojem maske, vendar predstavljajo samo en realen kot. Analiza vseh odzivov bi občutno upočasnila algoritem, saj bi moral upoštevati mnogo več možnosti kakor sicer. Zato uporabimo algoritem 3.2 za iskanje težišča kota, ki za vsak skupek navideznih kotov izračuna težišče, ki se nato uporabi v nadaljnjih korakih.

```
vsotaX = 0
vsotaY = 0
vsotaN = 0

for each tocka in slika
  if tocka.verjetnost > 0 and not tocka.obiskana then
    vsotaX = 0
    vsotaY = 0
    vsotaN = 0

    tezisce( tocka, 0 )
```

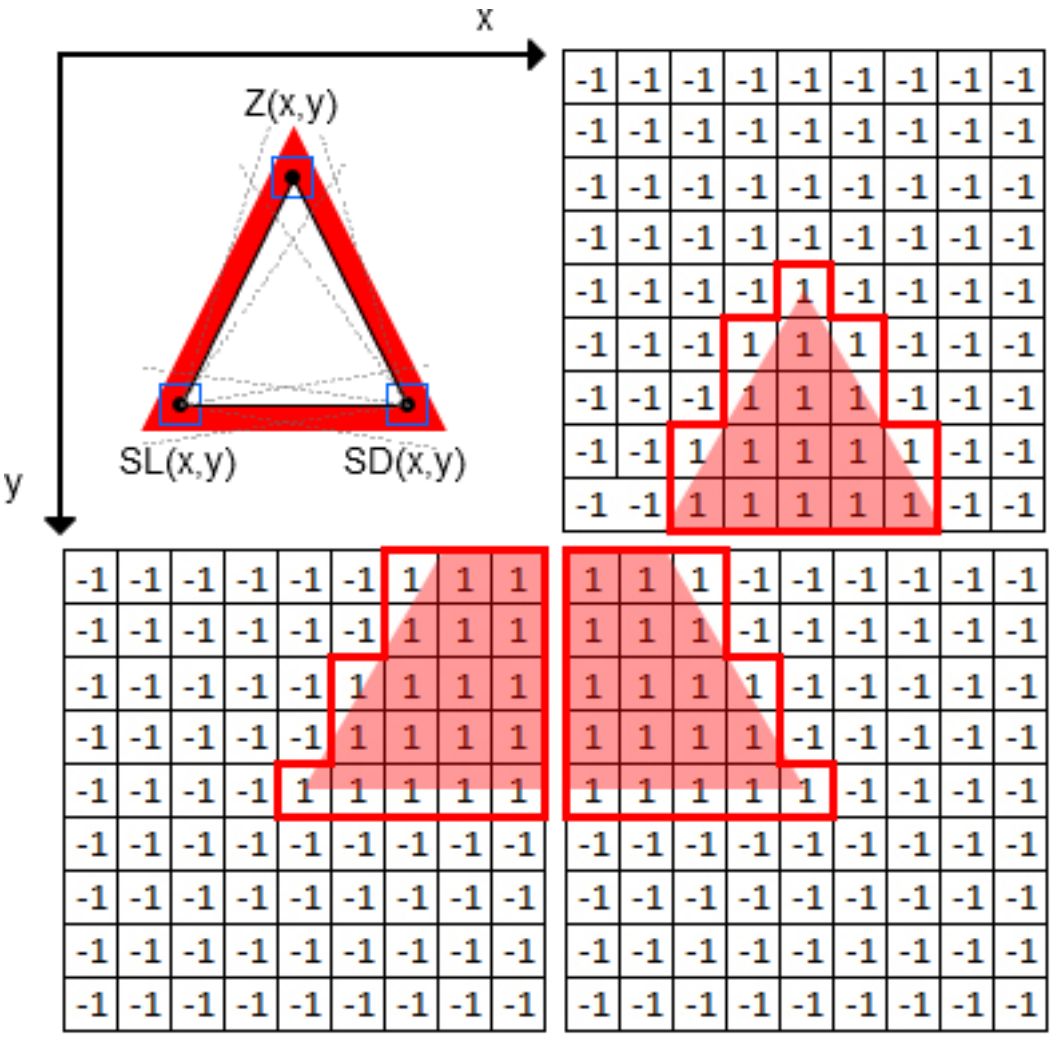
```
    x = vsotaX / vsotaN
    y = vsotaY / vsotaN

    dodaj x,y v seznam dejanskih kotov
  end
end

function tezisce( tocka, prazne )
  if tocka.verjetnost = 0 then
    prazne = prazne + 1

  vsotaX += tocka.verjetnost * tocka.x
  vsotaY += tocka.verjetnost * tocka.y
  vsotaN += tocka.verjetnost

  if tocka ni rob and prazne < 2 then
    tocka.obiskana = true
    tezisce( slika[ tocka.x, tocka.y + 1 ], prazne )
    tezisce( slika[ tocka.x, tocka.y - 1 ], prazne )
    tezisce( slika[ tocka.x + 1, tocka.y ], prazne )
    tezisce( slika[ tocka.x - 1, tocka.y ], prazne )
  end
end
```

Algoritem 3.2: Algoritem za izračun težišča kotov

Rezultat konvolucije je matrika, v kateri vsako polje predstavlja verjetnost, s katero se na sliki nahaja pripadajoč kot. Ničle pomenijo, da tam ni kotov. Te verjetnosti lahko uporabimo kot uteži, s katerimi lahko izračunamo težišče kota. Algoritem gre skozi vse točke v sliki z verjetnostjo večjo od nič. Za vsako tako točko, ki jo najde, rekurzivno preveri še njene sosede. Tako sešteva utežene koordinate. Rekurzija se prekine, če algoritem naleti na dve točki z verjetnostjo nič. Uteženo vsoto nato delimo z vsoto verjetnosti in dobimo dejanske koordinate kota, ki jih shranimo v seznam.

3.4 Detekcija kotov znakov za nevarnost

Notranji rob znakov za nevarnost ima obliko enakokrakega trikotnika. Kot tega trikotnika med krakoma in osnovnico ima 55°, medtem, ko ima kot med krakoma 70°. Maske za detekcijo lahko določimo glede na orientacijo in naklon kota. Za enakokraki trikotnik dobimo maske, ki jih prikazuje slika 3.3.



Slika 3.3: Kotne maske znakov za nevarnost.

Te maske nato konvoluiramo s sliko, ki je rezultat odstranjevanja šuma. Z upoštevanjem praga dobimo lokacije vseh kotov, ki ustrezajo tem maskam. Našo sliko sedaj sestavlja več navideznih kotov, zato moramo uporabiti algoritem za

izračun težišča za vsak tip kota posebej. Na voljo imamo tri sezname:

- SSL - seznam spodnjih levih kotov,
- SSD - seznam spodnjih desnih kotov in
- SZ - seznam zgornjih kotov.

Iz teh treh seznamov moramo sestaviti smiselne trikotnike. Preverimo vse možne kombinacije kotov po naslednjih korakih:

- iz seznama SZ izberi kot,
- iz seznama SSD in SSL izberi kot tako, da ustrezata pogojem
  - naklon premice skozi Z in SD mora biti med $55° - \theta$ in $55° + \theta$ stopinj,
  - naklon premice skozi Z in SL mora biti med $-55° - \theta$ in $-55° + \theta$ stopinj,
  - dolžini daljice med Z in SD, in daljice med Z in SL morata biti med l_{min} in l_{max},
  - dolžini daljice med Z in SD, in daljice med Z in SL morata biti približno enaki in
  - premica skozi SD in SL mora biti približno vodoravna.

3.5 Detekcija robov znaka za križišče s prednostno cesto

Ti znaki so podobni znakom za nevarnost. Torej so trikotne oblike, vendar z razliko od znakov za nevarnost, obrnjeni na glavo. Uporabimo lahko kar maske znakov za nevarnost (slika 3.3), ki jih zavrtimo za 180°. Te maske nato konvoluiramo s sliko, ki je rezultat odstranjevanja šuma. Z upoštevanjem praga dobimo lokacije vseh kotov, ki ustrezajo tem maskam. Našo sliko sedaj sestavlja več navideznih kotov, zato moramo uporabiti algoritem za izračun težišča za vsak tip kota posebej. Na voljo imamo tri sezname:

- SS - seznam spodnjih kotov,
- SZD - seznam zgornjih desnih kotov in
- SZL - seznam zgornjih levih kotov.

Tudi sestavljanje tega znaka je podobno sestavljanju znaka za nevarnost, zato lahko uporabimo kar podobna pravila:

- iz seznama SS izberi kot,
- iz seznamov SZD in SZL izberi kota tako, da ustrezata pogojem
  - naklon premice skozi S in ZD mora biti med $-55° - \theta$ in $-55° + \theta$ stopinj,
  - naklon premice skozi S in ZL mora biti med $55° - \theta$ in $55° + \theta$ stopinj,
  - dolžina daljice med S in ZD, in daljice med S in ZL morata biti med l_{min} in l_{max},
  - dolžini daljice med S in ZD, in daljice med S in ZL morata biti približno enaki in
  - premica skozi ZD in ZL mora biti približno vodoravna.

Oblika znakov za križišče s prednostno cesto je enolična. To nam poenostavi algoritem, saj nam ni potrebno klasificirati simbola v notranjosti znaka. Enostavno le v izvirni sliki, znotraj območja znaka, preštejemo število točk bele barve. Tu znova naletimo na problem spremenljivih svetlobnih pogojev. Prostor RGB teh točk moramo preslikati v prostor HSL. Ker potrebujemo samo svetlost (L), lastnost barve, ki omogoča razlikovanje med bolj svetlo in bolj temno barvo, ni potrebna uporaba nelinearnih funkcij. Velja, da je točka bela, če je svetlost večja od 0,9.

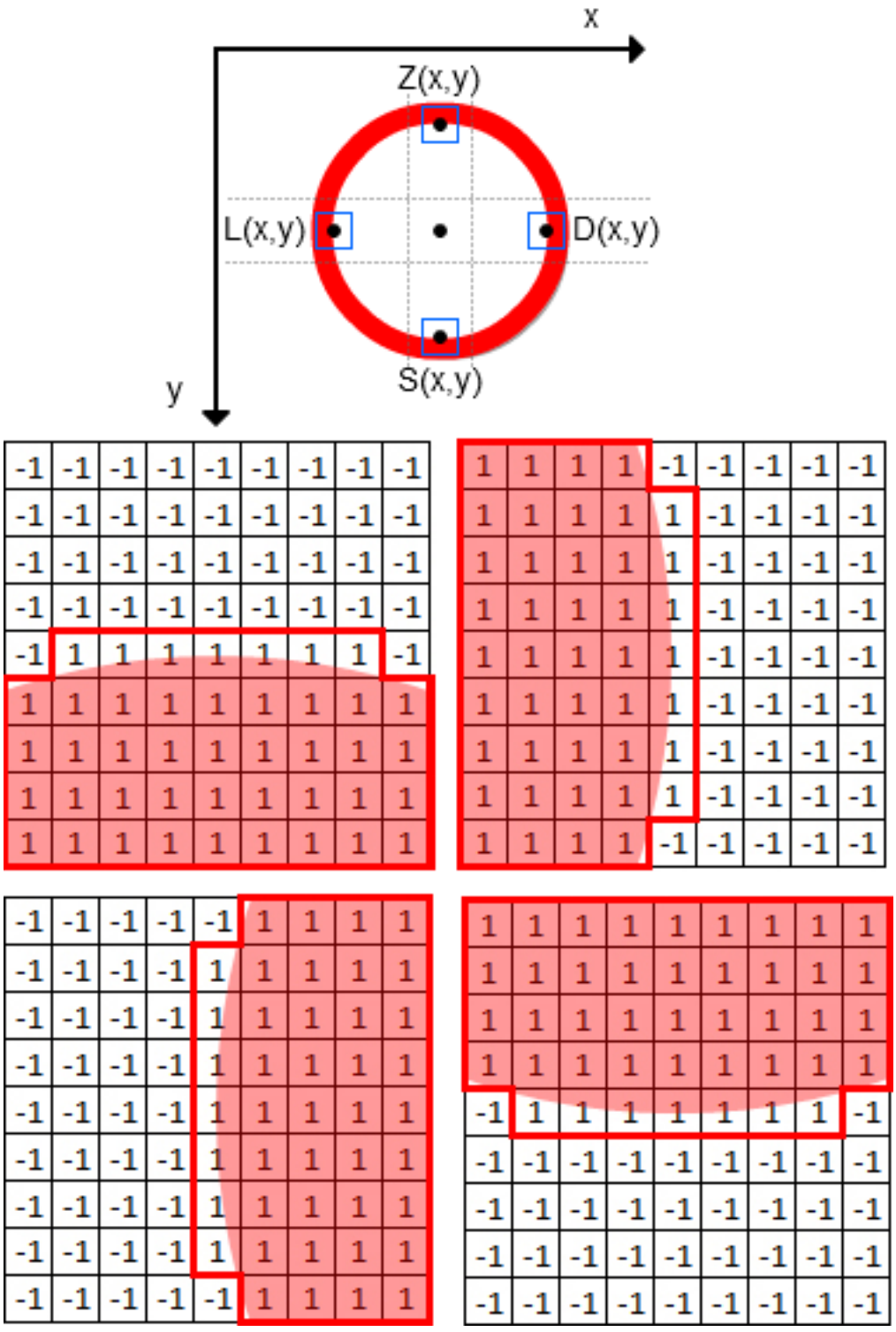
3.6 Detekcija robov znakov za prepoved

Znaki za prepoved so okrogli in nimajo kotov. Dovolj približan rob kroga izgleda kot skupek daljic z majhno spremembo naklona. Torej lahko te skrajne dele kroga aproksimiramo s pomočjo simetričnih premic z majhnim naklonom. Maske za detekcijo skrajnih delov kroga, lahko določimo glede na orientacijo in naklon teh premic. Dobimo maske, ki jih prikazuje slika 3.4. Te maske nato konvoluiramo s sliko, ki je rezultat odstranjevanja šuma. Z upoštevanjem praga dobimo lokacije vseh kotov, ki ustrezajo tem maskam. Našo sliko sedaj sestavlja več navideznih kotov, zato moramo uporabiti algoritem za izračun težišča za vsak tip kota posebej. Na voljo imamo štiri sezname:

- SL - seznam skrajnih levih delov kroga,
- SD - seznam skrajnih desnih delov kroga,
- SZ - seznam skrajnih zgornjih delov kroga in
- SS - seznam skrajnih spodnjih delov kroga.

Iz teh štirih seznamov moramo sestaviti smiselne kroge. Preverimo vse možne kombinacije kotov po naslednjih korakih:

- izberi kota iz SL in SD in preveri, če ležita približno na horizontalni premici,
- preveri, če kot L leži na levi strani kota D,
- izberi kot iz SZ, ki leži pod in približno na sredini horizontalne daljice skozi L in D,
- izberi kot iz SS, ki leži nad in približno na sredini horizontalne daljice skozi L in D,
- preveri, če je razdalja kotov S in Z, do horizontalne daljice skozi L in D, približno enaka in
- preveri, če je razdalja kotov S in Z, do horizontalne daljice skozi L in D, manjša ali enaka polovici te daljice.



Slika 3.4: Kotne maske znakov za prepoved.

3.7 Detekcija robov stop znakov

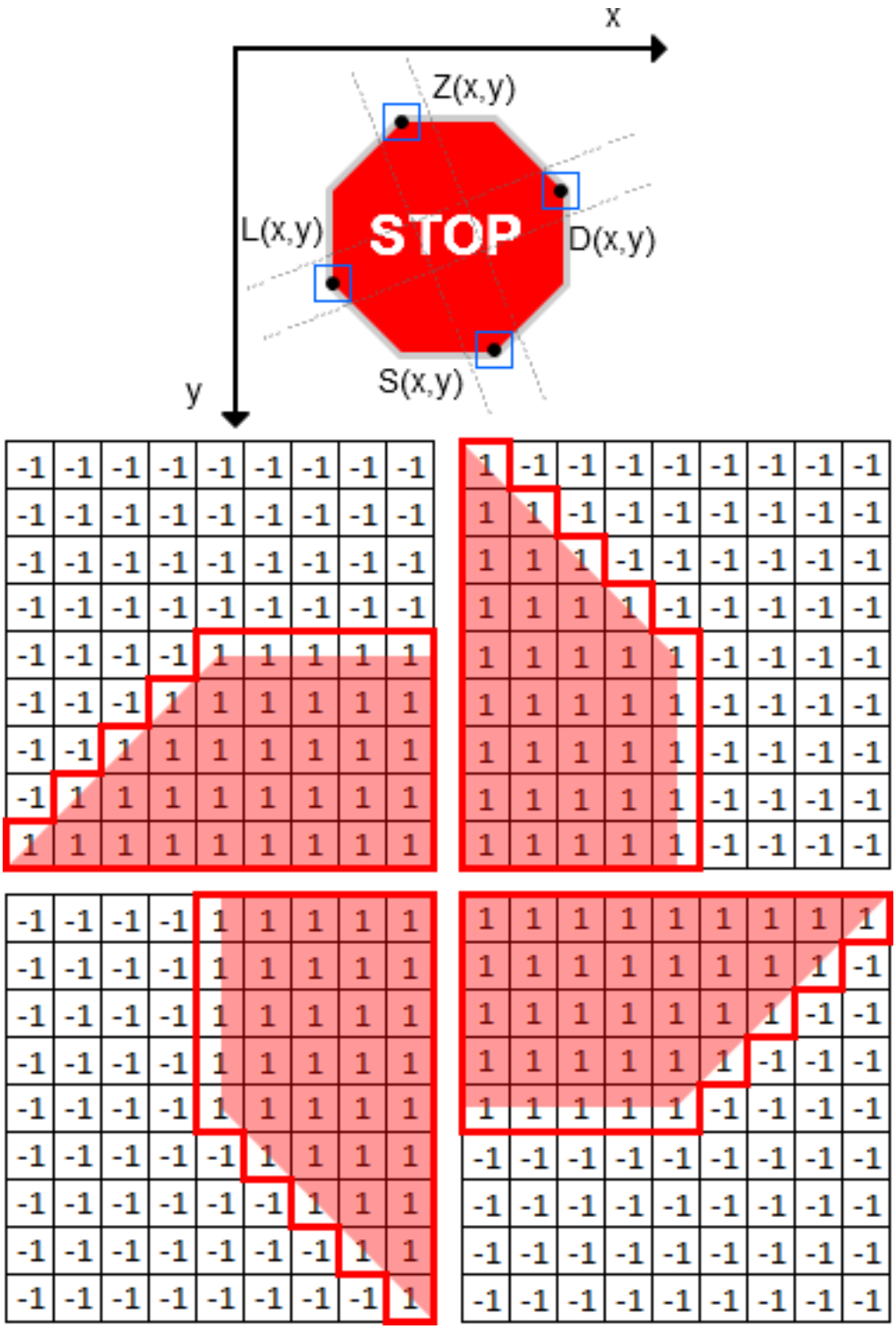
Stop znaki so nestandardne oblike (osemkotnik) in nimajo bele notranjosti. Torej ne moremo uporabiti prehoda med belimi in rdečimi točkami za detekcijo kotov. Lahko pa uporabimo mejo med okolico in rdečo notranjostjo znaka. Za detekcijo teh 135° kotov potrebujemo osem mask, kar zahteva veliko računanja. Algoritem lahko poenostavimo tako, da določimo samo štiri maske, kar je dovolj, da prepoznamo obliko osemkotnika. Maske za detekcijo lahko določimo glede na orientacijo in 135° naklon teh kotov. Dobimo maske, ki jih prikazuje slika 3.5. Te maske nato, podobno kot pri ostalih znakih, konvoluiramo s sliko, ki je rezultat odstranjevanja šuma. Z upoštevanjem praga dobimo lokacije vseh kotov, ki ustrezajo tem maskam. Našo sliko sedaj sestavlja več navideznih kotov, zato moramo uporabiti algoritem za izračun težišča kota za vsak tip kota posebej. Na voljo imamo štiri sezname:

- SL - seznam skrajnih levih delov osemkotnika,
- SD - seznam skrajnih desnih delov osemkotnika,
- SZ - seznam skrajnih zgornjih delov osemkotnika in
- SS - seznam skrajnih spodnjih delov osemkotnika.

Iz teh štirih seznamov moramo sestaviti smiselne osemkotnike. Preverimo vse možne kombinacije kotov po naslednjih korakih:

- izberi kota iz seznama SL in SD tako, da ustrezata pogojema
  - naklon premice skozi L in D mora biti med $160° - \theta$ in $160° + \theta$ stopinj ter
  - kot L mora biti na levi strani kota D,
- izberi kota iz SZ in SS tako, da ustrezata pogojem
  - naklon premice skozi Z in S mora biti med $70° - \theta$ in $70° + \theta$ stopinj,
  - kota Z in S morata ležati med kotoma L in D ter
  - kot Z mora ležati pod kotom S,
- izračunaj središče osemkotnika in preveri, če je razdalja vseh kotov do središča približno enaka.

Oblika stop znakov je enolična. To nam poenostavi algoritem, ker nam ni potrebno klasificirati simbola v notranjosti znaka. Enostavno le v izvirni sliki, znotraj območja znaka, preštejemo število točk rdeče barve. Uporabimo lahko isto metodo kot pri koraku segmentacije slike. Točka je torej rdeča, če je delež rdeče barve proti vsoti vseh barvnih komponent prostora RGB večji od 0,4.



Slika 3.5: Kotne maske stop znaka.

Poglavje 4

Nevronske mreže

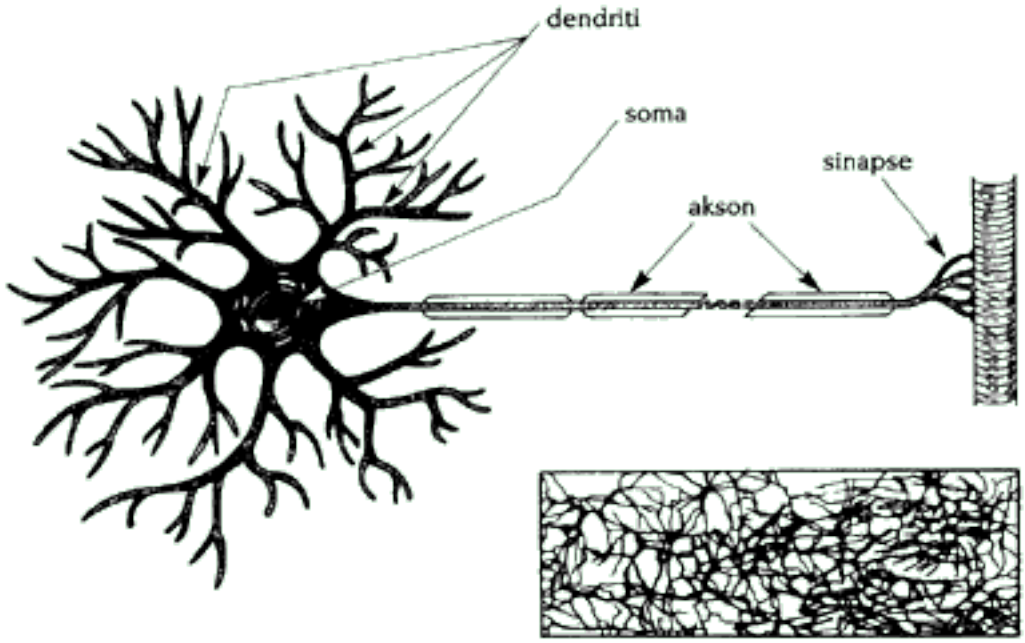
Umetne nevronske mreže so koncept procesiranja informacij, ki so ga navdihnili biološki sistemi. Ključni element teh sistemov je struktura, ki jo sestavlja veliko število med seboj povezanih elementov ali nevronov. Ti z medsebojno interakcijo poskušajo rešiti nek problem. Nevronske mreže se, tako kot ljudje, učijo s pomočjo izkušenj in se morajo reševanja specifičnih problemov, kot je na primer prepoznavanje vzorcev, najprej naučiti. Učenje v bioloških procesih poteka kot prilagajanje povezav med nevroni.

4.1 Uporaba

Nevronske mreže so sposobne narediti veliko stvari. Predvsem se dobro obnesejo v primeru kompleksnih ali nenatančnih podatkov, za kar običajni algoritmi niso primerni. Ti uporabljajo drugačno tehniko reševanja problemov. Običajni algoritmi sledijo vnaprej določenemu naboru ukazov pri reševanju problemov. To pa predstavlja precejšen problem, saj morajo biti ti ukazi znani, kar omejuje te algoritme na reševanje problemov, za katere vemo kako jih rešiti.
Nevronske mreže procesirajo informacijo podobno kot človeški možgani. Mreža je sestavljena iz velikega števila med seboj povezanih elementov ali nevronov, ki delujejo paralelno. Za reševanje problemov moramo mrežo najprej naučiti s pomočjo primerov. Ti morajo biti pazljivo izbrani, saj s tem pridobimo na hitrosti in kvaliteti učenja. Nevronska mreža se torej sama nauči kako rešiti problem. S tem seveda postane nepredvidljiva, kar je ena izmed manjših slabih lastnosti.

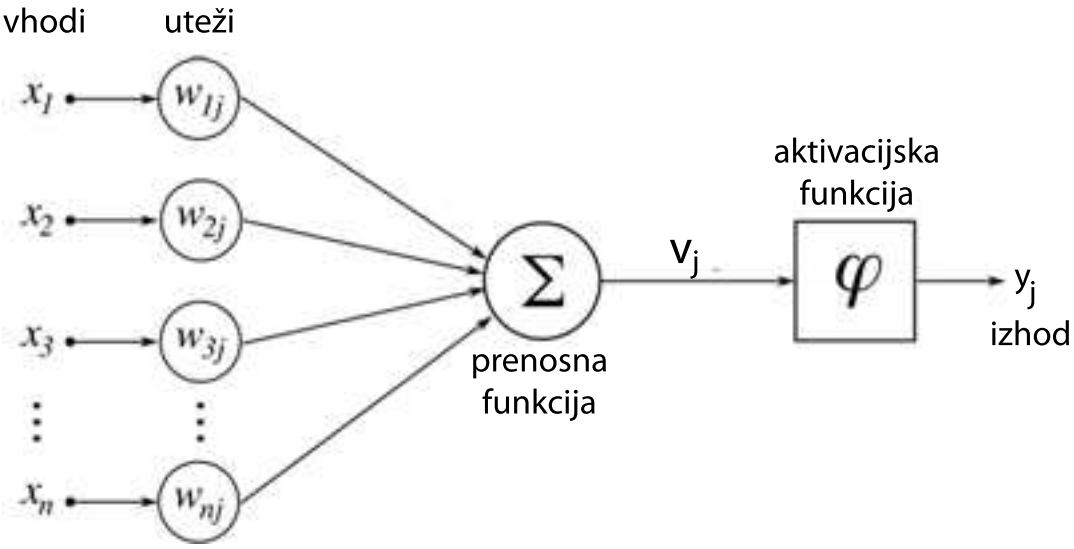
4.2 Nevron

Delovanje človeških možganov je v veliki meri še neraziskano. V možganih nevron sprejema signale od sosednjih nevronov po tankih strukturah, ki jih imenujemo dendriti. Nevron pošlje električni signal čez dolgo tanko strukturo, znano kot akson, ki se razdeli na tisoče drobnih vej. Na koncu vsake se nahaja sinapsa. Ta pretvarja aktivnosti iz aksona v električne impulze, ki inhibirajo ali vzdražijo na njih povezane dendrite sosednjih nevronov. Ko nevron postane dovolj vzbujen, pošlje aktivnost po aksomu.



Slika 4.1: Nevron (zgoraj) in naravna nevronska mreža (spodaj).

Nevron lahko predstavimo v obliki matematične funkcije, ki sta jo definirala McCulloch in Pitts [8] leta 1943. Prikazuje jo slika 4.2.



Slika 4.2: Matematični model nevrona.

Nevron sprejme množico vhodov, ki predstavljajo dendrite, jih pomnoži s pripadajočimi utežmi ter sešteje. To vsoto nato pošlje skozi aktivacijsko funkcijo, katere rezultat definira izhod nevrona. S pomočjo uteži nevron vhodom daje pomen oziroma moč. Z negativno utežjo vhode inhibira, medtem, ko jih s pozitivno ojači. Funkcija nevrona je definirana s sledečima enačbama

$$v_j = \sum_{i=0}^{n} w_{ij} x_i \tag{4.1}$$

ter

$$y_j = \sigma(v_j). \tag{4.2}$$

Vhod x_0 ponavadi fiksiramo na vrednost 1. Utež tega vhoda predstavlja prag aktivacijske funkcije. Model nevrona vključuje aktivacijsko funkcijo, ki je lahko linearna ali nelinearna. Izberemo jo glede na problem, ki ga poskušamo rešiti. Obstaja več aktivacijskih funkcij, vendar se zaradi prevelikega števila omejimo na dve, ki sta uporabljeni v našem algoritmu:

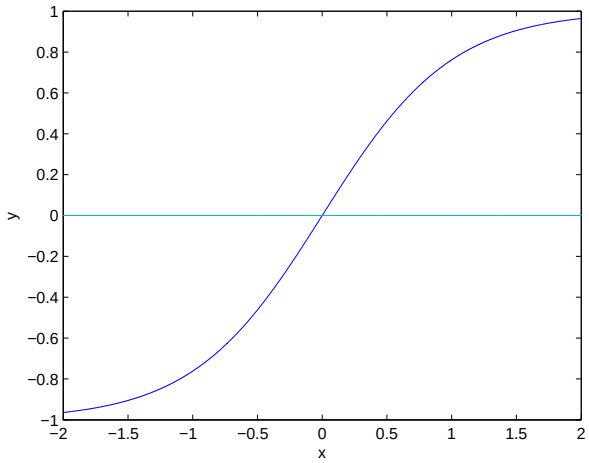
- nasičena linearna funkcija, ki jo uporabljamo v izhodni plasti in jo predstavlja enačba

$$y_j = \begin{cases} 1, & \text{če } v_j > 1 \\ v_j & \text{če } 0 < v_j < 1 \\ 0, & \text{če } v_j < 0 \end{cases} \tag{4.3}$$

- hiperbolični tangens, ki ga uporabljamo v skriti plasti in ga predstavlja enačba

$$y_j = \frac{e^{v_j} + e^{-v_j}}{e^{v_j} - e^{-v_j}}, \tag{4.4}$$

kjer je v_j utežena vsota vhodov in y_j izhod nevrona na poziciji j. Funkcija vhodno vrednost stisne med -1 in 1. Hiperbolični tangens prikazuje slika 4.3.



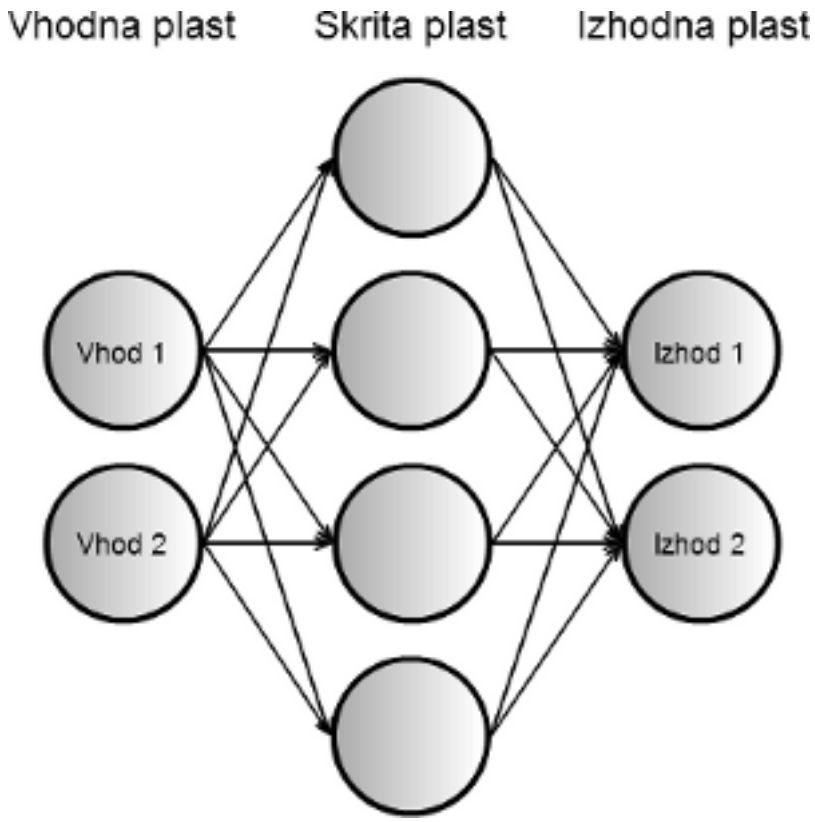
Slika 4.3: Hiperbolični tangens.

4.3 Večplastne nevronske mreže

Način organizacije nevronov je povezan z algoritmom, ki se uporabi za učenje nevronske mreže. Obstaja več tipov nevronskih mrež, ki se med seboj ločijo po tem, kako se nevroni med seboj povezujejo. V nadaljevanju se bomo omejili na večplastne nevronske mreže ter reševanju problemov, ki so povezani z razpoznavanjem vzorcev.
Mreže so običajno organizirane v več plasteh (*multi-layer networks*). V tem primeru notranje plasti imenujemo skrite plasti. Mreža v skriti plasti izbira kateri vhod je aktiven s pomočjo spreminjanja uteži. S tem lahko izbira pomen vhodnih podatkov. Preprosta oblika večplastnih mrež ima tri plasti:

- vhodno, ki predstavlja vhodne podatke,
- notranjo ali skrito, v kateri se izvrši procesiranje vhodnih podatkov, ter
- izhodno, ki predstavlja izhodno vrednost za določene vhodne podatke.

Nevronsko mrežo, v kateri so v vsaki plasti vsi nevroni povezani z vsemi iz predhodne plasti, lahko označimo kot polno povezano. V primeru manjkajočih povezav pa mrežo označimo kot delno povezano. Primer polno povezane mreže prikazuje slika 4.4.



Slika 4.4: Preprosta večplastna nevronska mreža.

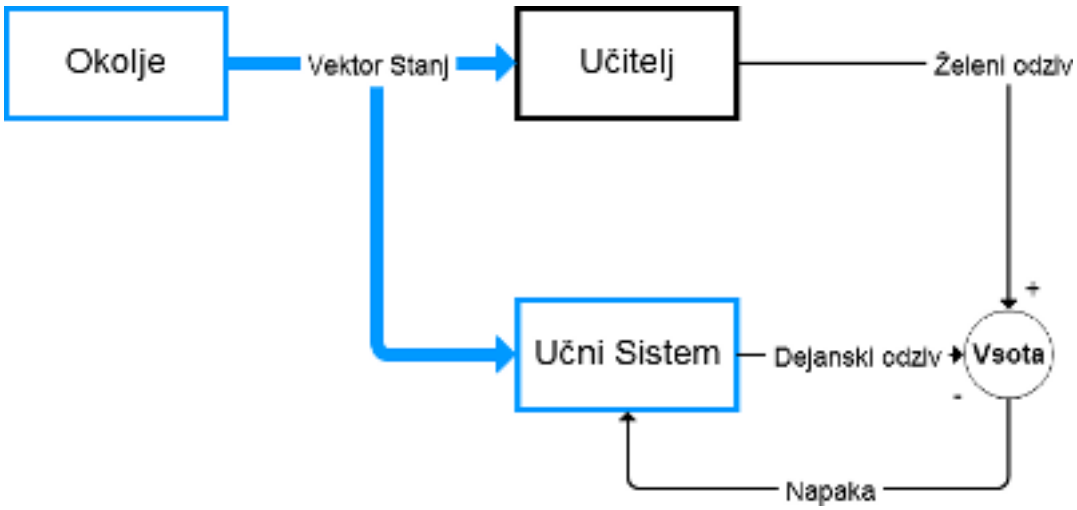
Signal v takih mrežah potuje iz vhodne proti izhodni plasti, medtem ko signal napake potuje v obratni smeri.

4.4 Učenje

Obstaja več načinov učenja nevronskih mrež. Omejili se bomo na obravnavanje nadzorovanega učenja, ki je najbolj primeren za naš problem.

4.4.1 Nadzorovano učenje

Naš cilj je najti preslikavo med vhodom in izhodom, oziroma pri danem vhodu mora nevronska mreža podati izhod, ki tem vhodom ustreza. Nadzorovano učenje poteka pod nadzorom učitelja. Učitelj preverja dejanski odziv nevronske mreže z želenim. Mreža deluje dobro, če je odziv mreže kar najbolj podoben želenim izhodom. Mreža se prilagaja ali uči toliko časa, dokler razlika med dejanskim in želenim odzivom mreže ali napaka, ne postane dovolj majhna. Znanje, ki ga posreduje učitelj, se v nevronski mreži shrani v obliki sinaptičnih uteži. Te, tako kot v človeških možganih, ponazarjajo spomin. Ob koncu učenja je nevronska mreža sama sposobna procesirati podobne naloge.



Slika 4.5: Nadzorovano učenje.

Opisan način učenja spada pod kategorijo učenja s popravljanjem napak. Na sliki 4.5 vidimo, da je proces sestavljen iz zaprte zanke, neznano okolje pa se nahaja zunaj nje.

4.4.2 Asociacija vzorcev

Asociativen spomin je podoben kot človeški možgani, ki se učijo z asociacijami. Poznamo dve obliki: avto asociacija in hetero asociacija.
Pri avto asociaciji nevronska mreža pomni nabor vzorcev. Njena naloga je popraviti oziroma dopolniti izkrivljen vzorec, ki se pojavi na vhodu mreže. Primer tega so ravno delno zakriti prometni znaki. Nevronska mreža hrani določen nabor znakov in v primeru zakritega znaka poda celoten znak.
Hetero asociacija se od avto asociacije razlikuje v funkciji preslikave. Ta vzame poljuben vhodni vzorec in ga preslika v poljubni izhodni vzorec. Avto asociacija ne potrebuje nadzorovanega učenja, medtem ko hetero asociacija potrebuje učenje z učiteljem.
Naj bo x_k vzorec, ki ga mreža dobi na vhod in y_k odziv mreže na ta vhod. Asociacijo, ki jo predstavlja nevronska mreža lahko zapišemo s

$$x_k \rightarrow y_k, \quad k = 1, 2, ..., N \quad , \tag{4.5}$$

kjer je q število shranjenih vzorcev v mreži. Vhodni vzorec I_k igra vlogo ključa, ki v mreži najde primeren izhodni vzorec y_k. Pri asociativnem pomnjenju je x_k kar enak y_k, tako imata vhodni in izhodni vektor enako dimenzijo. Pri hetero asociativnem spominu pa to ne velja. Asociativno pomnjenje razdelimo na dve fazi:

- faza pomnjenja (*ang. storage*), ki vsebuje učenje nevronske mreže skladno z enačbo 4.5 ter
- faza delovanja (*ang. recall*), ki za vsak popačen vzorec na vhodu, poišče temu primeren izhodni vzorec.

Naj bo x različica vhodnega vzorca x_j s šumom. Mreža na ta vzorec reagira z vzorcem y, ki bi moral biti enak naučenemu vzorcu y_j. V primeru neenakosti vemo, da je bila narejena napaka v delovanju mreže.
Število vzorcev q, ki jih lahko shranimo v mrežo, nam poda merilo o njeni kapaciteti.

4.4.3 Postopek vzvratnega učenja nevronskih mrež

Nevronske mreže so veliko pridobile na popularnosti po iznajdbi učinkovitega postopka za učenje, ki se imenuje vzvratni postopek učenja nevronske mreže. Postopek je sestavljen iz dveh faz:

- faze procesiranja in
- faze učenja.

Prvi prehod je potreben za računanje vrednosti nevronov y_j in se začne pri vhodni strani mreže. Izhodne vrednosti nevronov računamo z uporabo enačbe 4.2. Pri tem sinaptične uteži ostanejo nespremenjene. Izhodne vrednosti nevronov v izhodni plasti se nato primerjajo z želenimi vrednostmi, s katerimi dobimo napako nevronov.
Drugi prehod se, z razliko od prvega, začne pri izhodu mreže in vrednost napake pošilja proti vhodu in računa spremembo uteži za vsak nevron. Naj bo funkcija napake posameznega izhodnega nevrona j, v iteraciji n, definirana kot

$$e_j(n) = d_j(n) - y_j(n), \tag{4.6}$$

kjer je d_j željena in y_j dejanska vrednost nevrona. Definiramo še skupno izhodno napako ρ iteracije n, ki je vsota vseh posameznih napak nevronov v izhodni plasti:

$$\rho(n) = \frac{1}{2}\sum_j e_j^2(n). \tag{4.7}$$

Cilj učnega procesa je minimizirati skupno napako ρ čez vse iteracije vzorcev, ki je definirana z enačbo

$$\sigma = \sum_n \rho(n). \tag{4.8}$$

Uporabimo lahko preprosto metodo posodabljanja uteži nevronov, ki za vsak vzorec posebej uteži popravi za vrednost, ki je sorazmerna negativnemu gradientu napake. Na spremembo uteži pa seveda vpliva funkcija napake vzorca. Naj bo v_j utežena vsota vseh vhodov nevrona

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n)x_i(n), \tag{4.9}$$

kjer je m število vhodov nevrona j. Izhod nevrona j definiramo kot

$$y_j(n) = \sigma(v_j(n)). \tag{4.10}$$

Izračunati moramo spremembo uteži, ki je pri vzvratnem postopku sorazmerna z delnim odvodom funkcije napake $\rho(n)$. Spremembo uteži, ki povezuje nevron j z nevronom i, definiramo po pravilu delta kot

$$\triangle w_{ji}(n) = -\eta \frac{\delta \rho(n)}{\delta w_{ji}(n)}, \tag{4.11}$$

kjer je η hitrost učenja algoritma. Iščemo take spremembe vrednosti uteži, ki zmanjšajo vrednost napake ρ. Po verižnem pravilu lahko zapišemo

$$\frac{\delta \rho(n)}{\delta w_{ji}(n)} = \frac{\delta \rho(n)}{\delta e_j(n)} \frac{\delta e_j(n)}{\delta y_j(n)} \frac{\delta y_j(n)}{\delta v_j(n)} \frac{\delta v_j(n)}{\delta w_{ji}(n)}. \tag{4.12}$$

Delne odvode lahko dobimo z odvajanjem zgornjih enačb. Enačbo 4.7 odvajamo po $e_j(n)$ in dobimo

$$\frac{\delta \rho(n)}{\delta e_j(n)} = e_j(n). \tag{4.13}$$

Z odvajanjem enačbe 4.6 po $y_j(n)$ dobimo

$$\frac{\delta e_j(n)}{\delta y_j(n)} = -1 \quad . \tag{4.14}$$

Odvajanje enačbe 4.10 po $v_j(n)$ nam poda delni odvod

$$\frac{\delta y_j(n)}{\delta v_j(n)} = \sigma'(v_j(n)), \tag{4.15}$$

kjer $\sigma'(v_j(n))$ označuje odvod enačbe 4.10 po $v_j(n)$. Ostane nam le še odvajanje enačbe 4.9 po $w_{ji}(n)$,

$$\frac{\delta v_j(n)}{\delta w_{ji}(n)} = y_i(n). \tag{4.16}$$

Vse to vstavimo v enačbo 4.12 in dobimo

$$\frac{\delta \rho(n)}{\delta w_{ji}(n)} = -e_j(n)\sigma'(v_j(n))y_i(n). \tag{4.17}$$

Enačbo 4.17 vstavimo v enačbo 4.11 in dobimo

$$\triangle w_{ji}(n) = -\eta \theta_j(n) y_i(n), \tag{4.18}$$

kjer je gradient θ definiran z

$$\theta_j(n) = e_j(n)\sigma'(v_j(n))y_i(n) \tag{4.19}$$

in kaže v nasprotno smer od potrebne spremembe uteži. Po enačbi 4.19 je lokalni gradient nevrona j definiran kot produkt pripadajoče funkcije napake in odvoda aktivacijske funkcije. V odvisnosti od lokacije nevrona ločimo dve možnosti.

- Nevron se nahaja v zunanji plasti. V tem primeru lahko uporabimo enačbo 4.6 za računanje funkcije napake. Enostavno lahko izračunamo tudi gradient $\theta_j(n)$.
- Nevron se nahaja v skriti plasti. V tem primeru nimamo na voljo želenih vrednosti d_j in ne moremo izračunati funkcije napake ρ_j. Morali bi rekurzivno izračunati vrednost napake vseh nevronov, ki so s tem nevronom povezani. Po enačbi 4.19 definirajmo gradient nevrona v skriti plasti z

$$\theta_j(n) = -\frac{\delta\rho(n)}{\delta y_j(n)}\sigma'(v_j(n))y_i(n). \tag{4.20}$$

Za izračun delnega odvoda $\frac{\delta\rho(n)}{\delta y_j(n)}$ uporabimo napako nevronov iz izhodne plasti

$$\rho(n) = \frac{1}{2}\sum_k e_k^2(n), \tag{4.21}$$

kjer je e_k napaka nevrona v izhodni plasti. Uporabimo verižno pravilo

$$\frac{\delta\rho(n)}{\delta y_j(n)} = \frac{\delta\rho(n)}{\delta e_k(n)}\frac{\delta e_k(n)}{\delta y_j(n)} \tag{4.22}$$

in dobimo

$$\frac{\delta\rho(n)}{\delta y_j(n)} = \sum_k e_k\frac{\delta e_k(n)}{\delta y_j(n)}, \tag{4.23}$$

kjer spet uporabimo verižno pravilo

$$\frac{\delta e_k(n)}{\delta y_j(n)} = \frac{\delta e_k(n)}{\delta v_k(n)}\frac{\delta v_k(n)}{\delta y_j(n)} \tag{4.24}$$

in dobimo

$$\frac{\delta\rho(n)}{\delta y_j(n)} = \sum_k e_k(n)\frac{\delta e_k(n)}{\delta v_k(n)}\frac{\delta v_k(n)}{\delta y_j(n)}. \tag{4.25}$$

Za rešitev te enačbe potrebujemo delni odvod funkcije napake nevrona v izhodni plasti (odvod $e_k(n)$) po uteženi vsoti povezav nevrona v izhodni plasti ($v_k(n)$). Potrebujemo tudi delni odvod utežene vsote povezav nevrona v izhodni plasti (odvod $v_k(n)$) po izhodu nevrona v skriti plasti ($y_j(n)$). Vidimo, da je

$$\frac{\delta e_k(n)}{\delta v_k(n)} = -\sigma_k'(v_k(n)) \tag{4.26}$$

in

$$\frac{\delta v_k(n)}{\delta y_j(n)} = w_{kj}(n). \tag{4.27}$$

To vstavimo v enačbo 4.20 in dobimo

$$\theta_j(n) = \sigma'(v_j(n)) \sum_k \theta_k(n) w_{kj}(n) \quad . \tag{4.28}$$

Vidimo, da je računanje gradienta skrite plasti odvisno od gradienta izhodne plasti. Odvod $\sigma'(v_j(n))$ je odvisen samo od aktivacijske funkcije nevronov v trenutni plasti, medtem ko je gradient $\theta_k(n)$ odvisen tudi od nevronov naslednje plasti. To daje algoritmu tudi ime, saj moramo uteži spreminjati vzvratno (*ang. backpropagation*).

4.4.4 Izboljšave

Na hitrost učenja pri vzvratnem učenu vplivamo z učnim parametrom η. Manjši je parameter, manjše so spremembe sinaptičnih uteži in posledično počasnejše je učenje, ki se lahko ustavi v lokalnem minimumu. Vendar se v primeru prevelikega η, sinaptične uteži spreminjajo tako hitro, da postane mreža nestabilna (oscilacije napake). Preprosta izboljšava je uvedba novega parametra momenta α, s katerim nadziramo občutljivost na trenutni vzorec. Vpeljemo ga v enačbi 4.18.

$$\triangle wji(n) = -\eta \theta_j(n) y_i(n) + \alpha \triangle w_{ji}(n-1) \quad . \tag{4.29}$$

S tem zagotavljamo približevanje optimalni rešitvi v primeru oscilacij.

4.4.5 Levenberg-Marquardtov postopek

Obstajajo še boljši algoritmi učenja nevronskih mrež. Eden izmed njih je Levenberg-Marquardtov postopek [9], ki je hitrejša in bolj zapletena različica

vzvratnega algoritma. Podobno kot pri vzvratnem učenju, tudi ta poskuša minimizirati povprečno kvadratno napako, ki je definirana z enačbo 4.7. Z razliko od vzvratnega učenja, ta algoritem direktno minimizira funkcijo napake. Tudi ta algoritem nas ne pripelje do optimalne rešitve. Zaradi uporabe približkov pri računanju minimuma funkcije napake, se optimalni rešitvi samo približamo. Algoritem je mnogo hitrejši od vzvratnega učenja in zagotavlja hitro konvergenco v bližini minimuma.

4.4.6 Merilo uspešnosti

Algoritem nima dobro določenega merila, po katerem bi lahko dokazali konvergenco proti rešitvi, vendar ga lahko določimo približno z minimumom globalne ali lokalne napake. Mreža je dovolj dobro naučena takrat, ko je odvod vektorja napake po utežeh enak nič. Torej lahko definiramo merilo uspešnosti. Algoritem je konvergiral, če je gradient vektorja napake dovolj majhen, kar je zelo računsko potratna operacija, saj je potrebno v vsakem koraku računati gradient. Pravilo lahko poenostavimo z definicijo konvergence, če je absolutna sprememba napake dovolj majhna. Slaba stran tega kriterija je predčasni konec algoritma, saj ne dosežemo optimalne rešitve, le približek.

4.4.7 Ustavitveni kriterij

Mreža se lahko na učnih vzorcih preveč nauči. V tem primeru je srednja kvadratna napaka za učne vzorce zelo majhna, medtem, ko je napaka vzorcev, ki jih nismo uporabili za učenje, zelo velika.
Če želimo optimalno naučeno mrežo, moramo vzorce razdeliti v dve skupini. Prva skupina vzorcev se imenuje množica vzorcev za učenje, druga pa množica vzorcev za validacijo. Prvo uporabljamo za spreminjanje uteži mreže, druga pa služi za validacijo teh uteži. V vsakem koraku, po spremembi uteži, izračunamo srednjo kvadratno napako vzorcev za učenje in vzorcev za validacijo. Če v naslednjih korakih začne napaka vzorcev za validacijo rasti, učenje ustavimo.

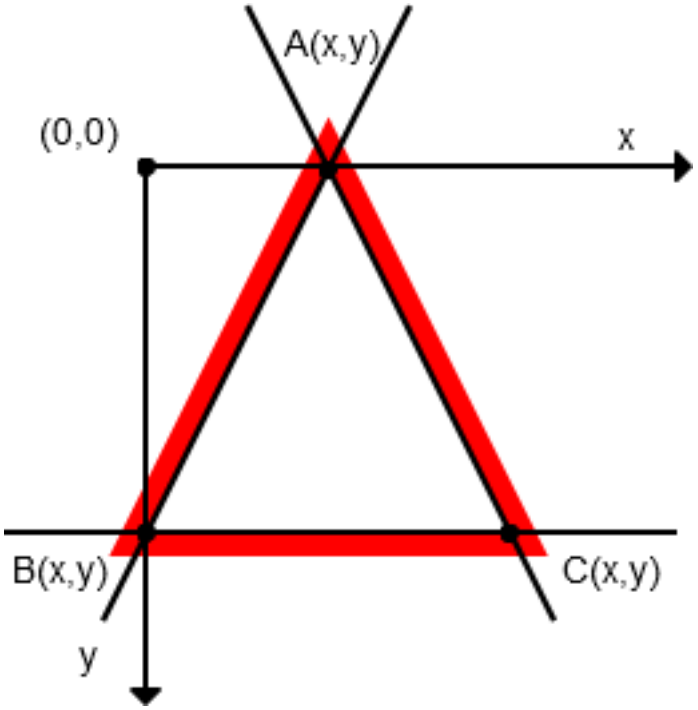
Poglavje 5

Prepoznavanje vzorcev

Ljudje smo pri prepoznavanju vzorcev zelo učinkoviti, saj brez napora lahko prepoznamo na primer obraz osebe, ki se je, od zadnjega obiska, zelo postarala. Prepoznavanje vzorcev je formalno definirano kot proces klasificiranja vzorcev v razrede (kategorije). Nevronska mreža se mora seveda najprej naučiti prepoznavanja vzorcev. Mrežo učimo z naborom vzorcev in razredov, ki tem vzorcem pripadajo. Učenje take mreže poteka s povezovanjem vhodnih vzorcev z izhodnimi. Uporabnost te mreže se kaže z identificiranjem vzorca, ki ni bil naučen. V tem primeru mreža na izhodu postavi vrednosti tistega vhoda iz učne množice, ki je najbližje.
Prepoznavanje vzorcev z nevronskimi mrežami je statistične narave, kjer so vzorci predstavljeni kot točke v večdimenzionalnem odločitvenem prostoru. Ta je razdeljen na območja, ki predstavljajo razrede (kategorije). Meje teh območij določimo z učenjem. V tem delu se bomo osredotočili na dve skupini znakov. Na znake za prepoved in znake za nevarnost. Prvi so trikotne oblike, medtem, ko so drugi okrogli. Že v fazi lociranja prepoznamo obliko znakov, zato lahko za uvrščanje uporabimo dve nevronski mreži, vsako za svojo skupino znakov. Pred postopkom uvrščanja znakov v razrede jih je potrebno pretvoriti v primerno obliko.

5.1 Predprocesiranje

Vsi izbrani znaki imajo rdeč rob, kar izdatno pripomore k učinkovitosti prepoznavanja, saj se lahko osredotočimo na prehod med rdečim robom in belo notranjostjo. V fazi lociranja znakov, v kateri določimo lokacije znakov na podlagi tega prehoda, iz slike izvlečemo posnetek znaka. Ta obsega pravokotnik, ki vsebuje notranjost znaka s simbolom, del roba in del okolice znaka. Rob in okolica znaka nista zaželena in ju moramo odstraniti.
Znaki za nevarnost imajo obliko trikotnika. Notranjost znaka lahko definiramo kot prostor med dvema premicama, ki ju lahko določimo s pomočjo robnih točk.



Slika 5.1: Znaki za nevarnost.

Naj bo premica med točkama A in B na sliki 5.1 določena z enačbo

$$y = k_1 x + n_1, \tag{5.1}$$

premica med A in C pa z enačbo

$$y = k_2 x + n_2. \tag{5.2}$$

Naklona k_1, k_2 in odseka n_1, n_2 lahko izračunamo iz robnih točk trikotnika

$$k_1 = \frac{x_A - x_B}{y_A - y_B}, \tag{5.3}$$

$$k_2 = \frac{x_A - x_C}{y_A - y_C}, \tag{5.4}$$

$$n_1 = y_A - x_A * k_1 \tag{5.5}$$

in

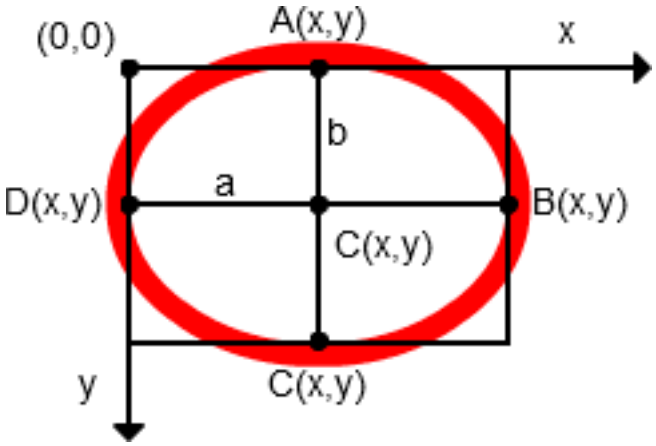
$$n_2 = y_A - x_A * k_2. \tag{5.6}$$

Zanemarimo lahko vse točke, ki ne ležijo med tema dvema premicama.
Pri znakih za prepoved je postopek podoben. Ti znaki imajo obliko kroga, ki je lahko stisnjen, zato jih lahko opišemo s kanonično enačbo elipse

$$\frac{(x - X_c)^2}{a^2} + \frac{(y - Y_c)^2}{b^2} = 1, \tag{5.7}$$

kjer sta X_c in Y_c koordinati središča, a in b pa parametra elipse. Po sliki 5.2 vidimo, da ima središče elipse koordinato $C(a, b)$. V notranjosti elipse so torej vse točke, ki zadoščajo sledečemu pogoju

$$\frac{(x - a)^2}{a^2} + \frac{(y - b)^2}{b^2} < 1. \tag{5.8}$$



Slika 5.2: Znaki za prepoved.

Sedaj imamo na voljo pravokotnik, ki na sredini vsebuje notranjost znaka. Število točk, ki jih lahko posredujemo nevronski mreži je seveda omejeno, zato

moramo sliko skrčiti na primerno velikost. Uporabimo funkcijo *resize* [13], ki nam na enostaven način omogoča zmanjšanje slike na 30x30 točk. To naredi z metodo bilinearne interpolacije, ki v vsaki točki izračuna linearno interpolacijo točk v x in y smeri. Rezultat je nato linearna interpolacija x in y smeri.
Sliko nato binariziramo, s pomočjo sledeče enačbe [14]

$$f(x,y) = \begin{cases} 1 & \text{če } 0,3f_r(x,y) + 0,59f_g(x,y) + 0,11f_b(x,y) < T \\ 0 & \text{drugod} \end{cases}, \quad (5.9)$$

kjer so f_r, f_g in f_b komponente prostora RGB in T parameter algoritma, ki je v našem primeru 128. Sliko znaka najprej pretvorimo v sivinsko črno-bel prostor ter ga prepolovimo glede na parameter T. Točke z vrednostjo manjšo od T naj imajo vrednost 1, ostale pa vrednost 0. Ostane nam 900 binarnih točk, kar pomeni 900 $*$ št. skritih nevronov + št. skritih nevronov $*$ št. izhodov učnih parametrov v nevronski mreži, kar je še vedno preveč za učinkovito učenje z nevronsko mrežo.
Večina izmed 900 točk se skozi vzorce spreminja zelo malo, zato lahko uporabimo metodo PCA. PCA je matematično orodje, ki z uporabo ortogonalnih transformacij strni vhodne podatke na podlagi medsebojne odvisnosti. Rezultat transformacije je množica podatkov, ki so med seboj neodvisni. Matlab omogoča enostavno izvedbo te metode z uporabo funkcije *processpca*, ki sprejme dva parametra

- X, vhodno matriko, ki jo želimo strniti ter
- I, izguba, ki je še sprejemljiva in je v našem primeru 0.001.

Rezultat te funkcije je nova strnjena matrika in nastavitve, ki povejo kako vektor strniti na podano število vhodov. Pri procesiranju vzorcev znakov za nevarnost dobimo vektorje dolžine 20, medtem, ko pri procesiranju znakov za dolžina vektorjev znaša 17.

5.2 Priprava podatkov za učenje

Za učenje potrebujemo množico preslikav med vhodi in izhodi, zato moramo zgraditi bazo znakov, ki bo predstavljala učne vzorce. Na internetu [11] najdemo idealne slike izbranih znakov. Najboljšo mrežo dobimo z učenjem na vzorcih iz realnega sveta, ki so pogosto popačeni (šum, zamegljenost...). Ti so trenutno nedostopni, zato vzamemo slike idealnih znakov, jih primerno popačimo in po zgoraj opisanem postopku obdelamo. Tako dobimo vektorje, ki predstavljajo učno množico.

5.3 Učenje mreže

Za klasifikacijo imamo na voljo množico preslikav med vhodi in izhodi, zato lahko uporabimo nadzorovano učenje s pomočjo Levenberg-Marquardtovega postopka. Za lažjo implementacijo uporabimo troslojno polno povezano nevronsko mrežo, v kateri uteži predstavimo z matrikami. Imamo dve kategoriji znakov, zato uporabimo dve enaki nevronski mreži. Za učenje uporabimo Matlab orodje *nftool*, ki sprejme tri parametre:

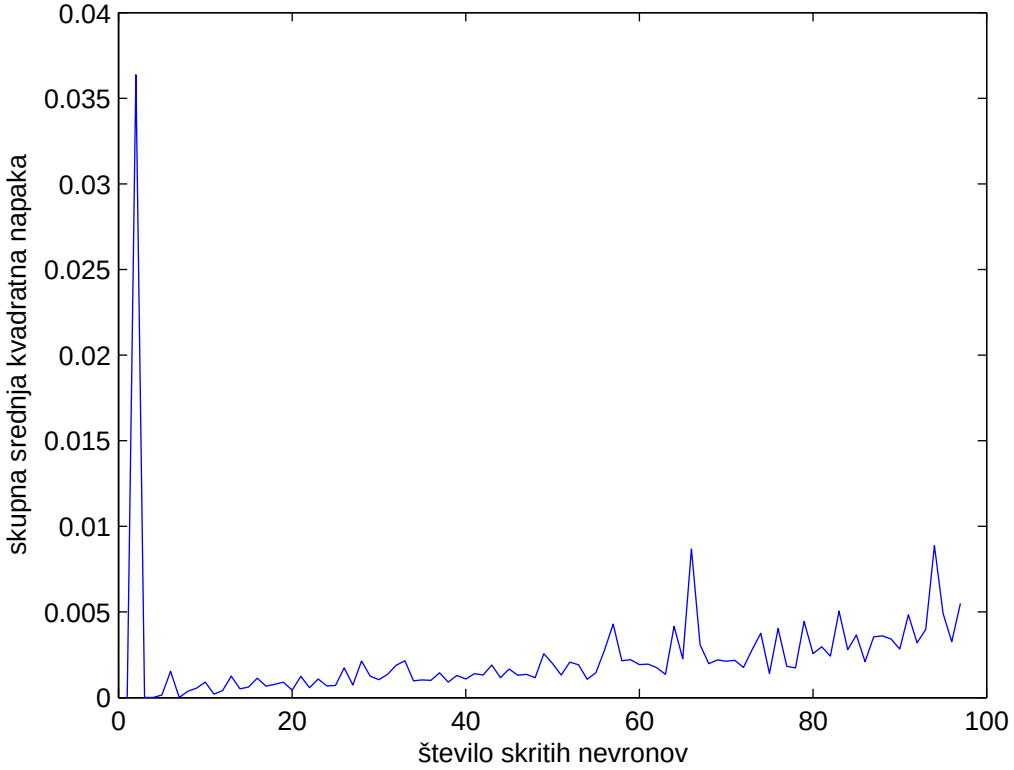
- vhodne vzorce v obliki matrike
- željene izhode v obliki matrike in
- število nevronov v skriti plasti.

Število stolpcev vhodne matrike določa število vhodov, število vrstic izhodne matrike pa število izhodov matrike. Vhodna matrika se razdeli na tri dele:

- vzorci za učenje (40%), s pomočjo katerih se prilagaja uteži med učenjem,
- vzorci za validacijo (30%), s pomočjo katerih se med učenjem testira uspešnost mreže in preprečuje pojav fotografskega spomina, ter
- vzorci za testiranje (30%), s pomočjo katerih se testira uspešnost mreže ob koncu učenja.

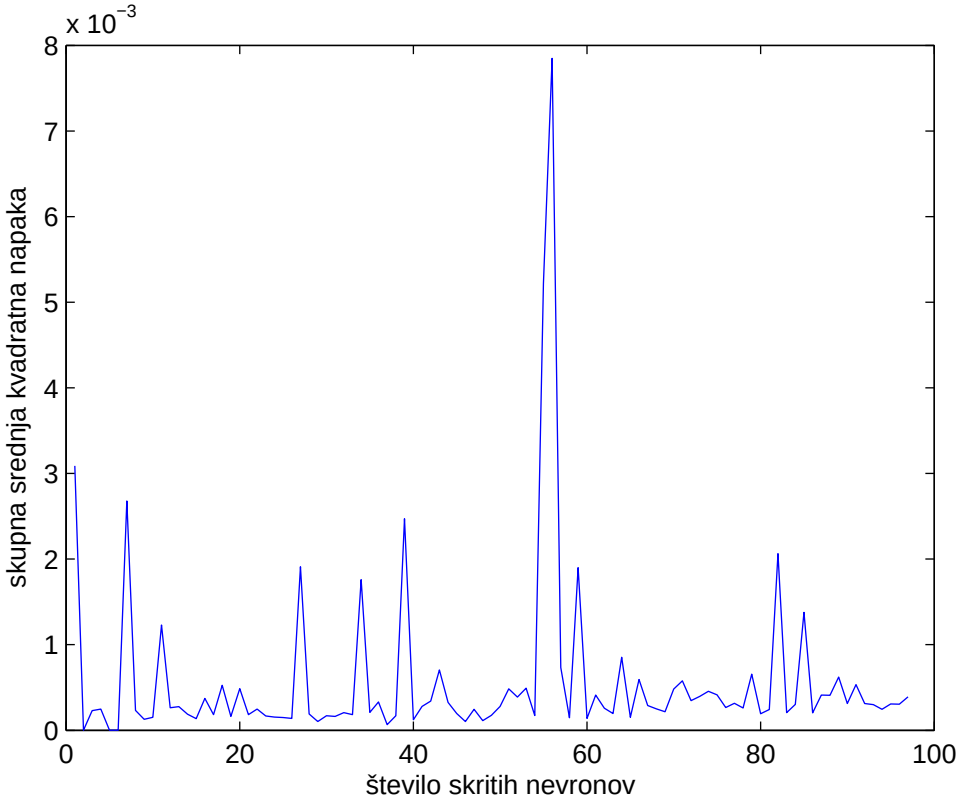
Določiti moramo še število nevronov v skriti plasti, ki je ena pomembnejših karakteristik nevronske mreže. V primeru nezadostnega števila skritih nevronov, mreža ne more modelirati kompleksnih podatkov, medtem, ko v primeru prevelikega števila nevronov čas učenja postane zelo dolg in mreža lahko razvije fotografski spomin. To pomeni, da postane nestabilna za vse vrednosti,

ki niso v učni množici. Rezultati učenja so v tem primeru odlični, vendar se ob simuliranju na novih vzorcih obnesejo zelo slabo. Število nevronov tako določimo edino z empiričnim preverjanjem. Mrežo učimo z razponom števila skritih nevronov od 1 do 100. Za vsako število skritih nevronov mrežo naučimo desetkrat. Najboljše uteži ima tista mreža pri kateri je skupna kvadratna napaka vzorcev za učenje, validacijo in testiranje, najmanjša. Slika 5.3 prikazuje spreminjanje srednje kvadratne napake z večanjem števila skritih nevronov pri učenju mreže znakov za nevarnost.



Slika 5.3: Srednja kvadratna napaka učenja znakov za nevarnost.

Vidimo, da srednja kvadratna napaka najmanjšo vrednost doseže pri številu skritih nevronov 5. Slika 5.4 prikazuje spreminjanje srednje kvadratne napake z večanjem števila skritih nevronov pri učenju mreže znakov za prepoved.



Slika 5.4: Srednja kvadratna napaka učenja znakov za prepoved.

Vidimo, da srednja kvadratna napaka najmanjšo vrednost doseže pri številu skritih nevronov 4. Na obeh grafih se lepo vidi rast srednje kvadratne napake z večanjem števila skritih nevronov, saj se s tem veča tudi verjetnost pojava fotografskega spomina mreže.

5.4 Uporaba mreže

Uporaba naučene mreže je zelo enostavna. Binarno sliko moramo najprej strniti z uporabo *processpca* funkcije, vendar z drugačnimi parametri kot pri vzorcih. Funkcija v tem primeru sprejme tri parametre:

- konstanto apply, ki pove funkciji, da mora uporabiti obstoječe nastavitve za krčenje matrike,
- matriko, ki jo želimo skrčiti, v našem primeru je to binarni vektor dolžine 900, ter
- transformacijsko matriko, s katero moramo pomnožiti vhodni vektor.

Rezultat funkcije nato posredujemo funkciji *sim*, ki simulira delovanje mreže in sprejme dva parametra:

- konstanto 'apply', ki pove funkciji, da mora uporabiti obstoječe nastavitve za krčenje matrike,
- matriko, ki jo želimo skrčiti, v našem primeru je to binarni vektor dolžine 900, ter
- nastavitve, ki povejo funkciji kako skrčiti podano matriko.

Rezultat je vektor, sestavljen iz števil s plavajočo vejico in predstavlja ujemanje določenega znaka. Določiti moramo še prag, ki predstavlja prepovedano območje. To pomeni, da ne moremo natančno določiti kateri znak predstavlja vektor. Naj bo vrednost izhoda določena z enačbo

$$O(x) = \begin{cases} 1, & \text{če } 0{,}75 <= \text{y(x)} \\ 0, & \text{če y(x)} <= 0{,}25 \\ X, & \text{drugod} \end{cases}, \tag{5.10}$$

kjer je $O(x)$ končna vrednost in y(x) izhodna vrednost nevronske mreže.

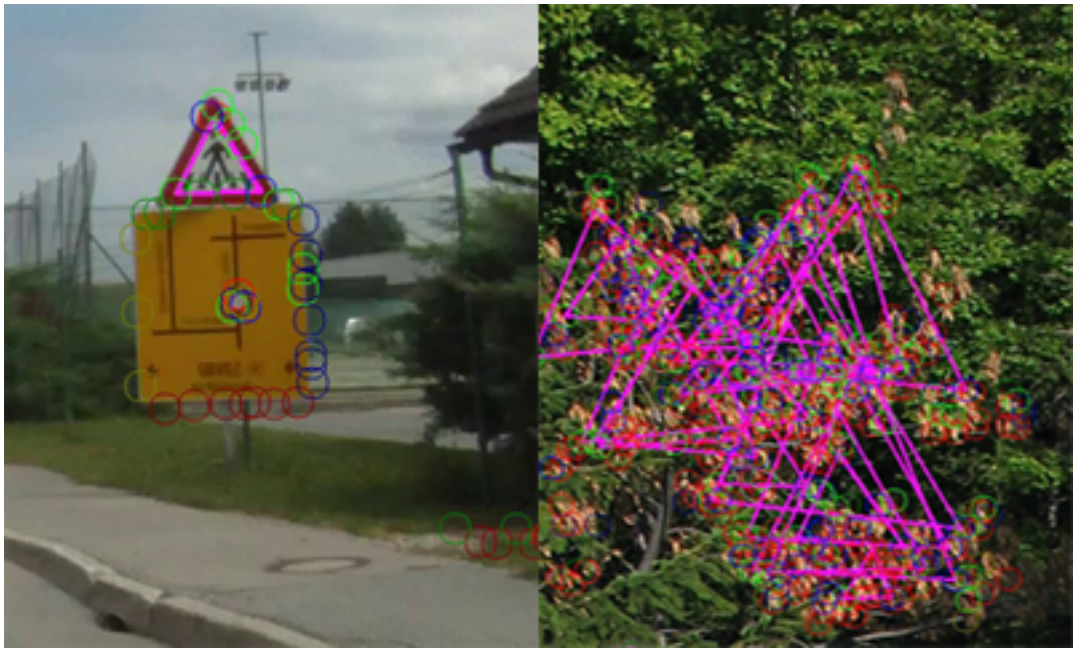
Poglavje 6

Testiranje

V tem razdelku je opisano testiranje prepoznavanja prometnih znakov na podlagi slik iz realnega okolja.

6.1 Testiranje lociranja znakov

Testna množica lociranja znakov obsega 20 slik znakov za križišče s prednostno cesto, 41 slik znakov za nevarnost in 63 slik znakov za prepoved. Slike temeljijo na posnetkih znakov v realnem okolju. Na vsaki sliki je natanko en znak. Na teh slikah so bili izvedeni zgoraj navedeni postopki v sledečem vrstnem redu: glajenje, barvna segmentacija, detekcija kotov in prepoznavanje oblik znakov. Slika 6.1 prikazuje rezultat lociranja znakov. Na levi strani slike je prikazana pravilno prepoznana lokacija znaka brez lažnih znakov, na desni pa slika z napačno lociranimi znaki.
Uspešnost lociranja prometnih znakov prikazuje tabela 6.1. Rezultati so razdeljeni na tri kategorije, izmed katerih vsaka predstavlja svoj tip znaka. Za vsako kategorijo je prikazano število testnih slik (C), število pravilno (S) in število lažno (F) zaznanih znakov. Lažni znaki so področja na sliki, na katerih dejansko ni prometnih znakov.



Slika 6.1: Primera lociranja prometnih znakov.

	C	S	F
križišče s prednostno cesto	26	21	60
nevarnost	41	39	64
prepoved	63	61	39

Tabela 6.1: Uspešnost lociranja znakov.

Delež pravilno zaznanih znakov je zelo visok, prav tako tudi delež lažno zaznanih znakov. Temu gre pripisati visoko število najdenih kotov v sliki, kar je posledica barvne segmentacije. Ta ima majhno toleranco na barve, ki so podobne rdeči (rumenorjava, zelenorjava...).

6.2 Testiranje klasifikacije znakov

Testirali smo klasifikacijo znakov, ki obsega 21 slik z enim znakom za nevarnost in 25 slik z enim znakom za prepoved. Izbrane so tiste slike, v katerih je algoritem pravilno lociral prometni znak. V vsaki sliki se nahaja natanko en pravilno lociran prometni znak, lahko pa tudi več lažno lociranih znakov. Število teh je v veliki meri odvisno od tipa slike in ni enakomerno porazdeljeno po slikah. V slikah z znaki za nevarnost se nahaja 25 lažno lociranih znakov, medtem, ko je število teh v slikah z znaki za prepoved 27.
Uspešnost lociranja prometnih znakov prikazuje tabela 6.2. Podobno kot pri testiranju lociranja znakov so rezultati razdeljeni na dve kategoriji. Za vsako kategorijo je prikazano število slik znakov v testni množici (C), delež pravilno klasificiranih znakov (S), število lažno lociranih znakov v slikah (FC) in delež nepravilno klasificiranih lažnih znakov(FF).

	C	$S(\%)$	FC	$FF(\%)$
nevarnost	21	85.7	25	92
prepoved	25	60	27	88.9

Tabela 6.2: Uspešnost lociranja znakov.

Kljub uporabi prepovedanega območja, mreža napačno klasificira večino lažnih znakov. Razlog zato je nepravilno naučena nevronska mreža, ki pričakuje na vhodu dejanski in ne lažni znak. Brez poseganja v že naučeno mrežo to lahko rešimo z uporabo dodatne nevronske mreže, ki služi za izočevanje lažnih znakov. Učenje te mreže je potekalo z dvema skupinama vzorcev. Prva skupina vsebuje vzorce, v katerih so dejanski znaki, druga pa poljubne vzorce okolice znakov. Mreža na izhodu poda 1, če je vhodni vzorec dejanski znak, in 0, če je vhodni vzorec lažni znak. Uporaba te mreže pripomore k manjšemu deležu nepravilno klasificiranih lažnih znakov, vendar na ta račun žrtvujemo uspešnost klasificiranja dejanskih znakov, saj mreža zavrže nekaj pravilno lociranih prometnih znakov. Tabela 6.2 prikazuje uspešnost klasificiranja prometnih znakov z dodatno nevronsko mrežo. Enako kot v prejšnjem primeru je za vsako kategorijo prikazano število slik znakov v testni množici (C), delež pravilno klasificiranih znakov (S), število lažno lociranih znakov v slikah (FC) in delež nepravilno klasificiranih lažnih znakov(FF).

	C	$S(\%)$	FC	$FF(\%)$
nevarnost	21	71.4	25	4
prepoved	25	52	27	14.8

Tabela 6.3: Uspešnost lociranja znakov z dodatno nevronsko mrežo.

Uspešnost klasficikacije bi se izboljšala z uporabo boljše baze prometnih znakov, ki v trenutni implementaciji temelji na posnetkih idealnih znakov. Ti so v nekaterih primerih drugačni od tistih v realnem okolju, zato bi boljše rezultate dosegli z bazo prometnih znakov, ki bi temeljila na slikah znakov v realnem okolju, vendar je ta trenutno nedostopna.

Poglavje 7

Zaključek

V delu je predstavljena implementacija sistema za prepoznavanje prometnih znakov, ki temelji na Daimler-Benzovem algoritmu [1]. Preprosta metoda barvne segmentacije zavrže barve, ki nas ne zanimajo. Maske za detekcijo kotov učinkovito najdejo lokacije kotov znakov. Na podlagi medsebojne lokacije kotov prepoznamo obliko znaka. Nevronska mreža nato točke znotraj znaka klasificira glede na tip znaka. Med implementacijo algoritma so se pojavljale težave navedene v nadaljevanju.

- Računanje težišča kotov, ki lahko povzroči premik dejanske lokacije kota. To povzroči problem posebej pri manjših znakih, kjer je natančnost bolj pomembna.
- Barvna segmentacija, ki močno vpliva na število zaznanih kotov in hitrost algoritma. Algoritem pri nekaterih slikah dajeje boljše rezultate z enimi nastavitvami, medtem, ko pri drugih slikah boljše z drugimi nastavitvami.
- Klasifikacija znaka, ki z mrežo, naučeno na idealnih vzorcih, nepravilno prepozna znake.
- Težavnost pri razhroščevanju kode. Pri kompleksnih slikah število zaznanih kotov naraste, kar otežuje iskanje tistih kotov, ki povzročajo težave.
- Z večanjem števila parametrov algoritma se veča tudi težavnost odkrivanja optimalnih vrednosti teh parametrov.
- Optimizacija kode.

Algoritem bi lahko izboljšali na veliko področjih, predvsem v koraku barvne segmentacije in prepoznavanje oblik znakov, kjer bi lahko uporabili boljše iskalne algoritme. Bolj učinkovit način konvolucije bi bila uporaba diskretne

Fourierjeve transformacije ter manjših in posplošenih mask. Ideja teh mask je v uporabi ene maske za kote s približno enako usmerjenostjo (npr. zgornji kot pri znakih za nevarnost in zgornji kot pri znakih za prepoved), s čimer bi povečali hitrost in rahlo poslabšali kvaliteto algoritma. Ne smemo pozabiti tudi na klasifikacijo simbolov znotraj znakov. Nevronske mreže so odlično orodje za prepoznavanje znakov, vendar ob predpostavki, da imamo na voljo primerno podatkovno bazo učnih vzorcev. Mnogo boljše rezultate bi dosegli z uporabo baze, ki temelji na slikah prometnih znakov iz realnega okolja, ki pa trenutno ni na voljo.

Vse več navigacijskih naprav v avtomobilih vsebuje prikaz prometnih znakov na podlagi podatkovne baze. Tak način prikaza znakov je mnogo bolj zanesljiv, saj ni potrebno imeti dodatne kamere in zapletenih algoritmov, ampak temelji na vnaprej shranjenih koordinatah znakov. Ima pa tudi slabe lastnosti, saj ne izraža trenutnega stanja na cesti, zato ga je potrebno pogosto posodabljati. V tem primeru je bolje uporabljati postopek, opisan v tem delu.

Za nadaljnji razvoj bi bilo zanimivo prilagoditi algoritem za prepoznavanje prometnih znakov na mobilnih telefonih z vgrajeno kamero. Voznik bi mobilni telefon namestil na vetrobransko steklo s posebnim nastavkom, sistem pa bi ga ob zaznanem prometnem znaku primerno opozoril.

Slike

Tabele

Algoritmi

Literatura

[1] S. Estable, J. Schick, F. Stein, R. Janssen, R. Ott, W. Ritter, Y. Zheng, CNRS, Aubiere, France, *"A real-time traffic sign recognition system,"* v Proceedings of the Intelligent Vehicles '94 Symposium, okt. 1994, str. 24-30.

[2] K. Rangarajan, M. Shah, and D. Van Brackle, *"Optimal corner detector,"* v Computer Vision, Graphics and Image Processing, nov. 1989, str. 230-245.

[3] J. Canny, *"A Computational Approach to Edge Detection,"* v IEEE Transactions on Pattern Analysis and Machine Intelligence, št. 8, zv. 6, nov. 1986, str. 679-698.

[4] W. McCulloch, W. Pitts, *"A logical calculus of the ideas immanent in nervous activity,"* v Bulletin of Mathematical Biology, št. 52, zv. 1, dec. 1990, str. 99-115.

[5] A. Dobnikar, B. Šter, *Mehko računanje za modeliranje razpoznavanje in regresijo*, Založba FRE in FRI, sep. 2008.

[6] S. Haykin, *Neural Networks and Learning Machines*, Prentice Hall, nov. 2008.

[7] Donald O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*, Psychology Press, jun. 2002.

[8] M. Bratina, *Modeliranje nelineranih dinamičnih sistemov z metodami teorije informacij*, doktorska disertacija, Univerza v Ljubljani, 2009.

[9] U. Lotrič, *Uporaba valčne analize in nevronskih mrež pri napovedovanju časovnih vrst*, doktorska disertacija, Univerza v Ljubljani, 2000.

[10] (2011) Neural Networks. Dostopno na:
http://library.thinkquest.org/C007395/tqweb/intro.html.

[11] (2011) Slike prometnih znakov. Dostopno na:
http://www.signaco.si/pznaki.htm.

[12] (2011) Barvni prostori. Dostopno na:
http://en.wikipedia.org/wiki/HSL_and_HSV.

[13] (2011) OpenCV. Dostopno na:
http://opencv.willowgarage.com/wiki/.

[14] (2011) Sivine. Dostopno na:
http://en.wikipedia.org/wiki/Grayscale.