SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 316

DETEKCIJA OBJEKATA NAUČENIM ZNAČAJKAMA HISTOGRAMA ORIJENTACIJE GRADIJENTA

Siniša Louč

Zagreb, srpanj 2011.

Sadržaj:

1. Uvod

Jedan od najznačajnijih radova na području računalnog vida jest rad Paula Viole i Michaela Jonesa [1] iz 2001. g. u kojem su predstavili generalizirani algoritam detekcije objekata kojeg je moguće naučiti da detektira objekte po želji, uz pružanje slika željenog objekta kao primjera za učenje. Osnovna ideja koja stoji iza njihovog rada korištena je u mnogim drugim radovima, uključujući i ovaj. Glavna razlika jest korištenje informacije o kutu gradijenta piksela u slici umjesto osnovnog intenziteta piksela.

Zadatak programa ostvarenog u okviru ovog diplomskog rada jest detekcija trokutnih znakova u slikama. Slike su pribavljene kamerom postavljenom u osobnom vozilu. Kamera snima cestu ispred vozila u pokretu te se kasnije dobiveni video snimak rastavlja na sekvencu slika u kojima se potom traže trokutni znakovi. Detekcija trokutnih znakova bila je predmet i nekih radova koji su prethodili ovom [3][6], ali ono po čemu se ovaj rad razlikuje jest već spomenuto učenje. Naime, osim trokutnih znakova, ostvareni algoritam može detektirati i druge objekte, samo je potrebno obaviti novo učenje za svaki novi objekt koji se želi detektirati (više o ovome u poglavlju 2.4.3).

Osnovni princip rada jest kako slijedi.

Svaka pribavljena slika prvo se obrađuje Cannyjevim algoritmom detekcije rubova razvijenim u okviru preddiplomskog projekta [4]. Cannyjev algoritam detektira rubove u slici kao nagle prijelaze iz tamnih u svijetla područja i obratno te kao izlaz daje binarnu sliku u kojoj svaki piksel ili predstavlja rubni element (logička jedinica) ili ne (logička nula). Osim binarne slike, korištena implementacija Cannyjevog algoritma daje i informacije o kutu gradijenta za svaki rubni element, što je također potrebno za daljnju obradu (vidi poglavlje 2.1). Zatim se po dobivenoj binarnoj slici prolazi s pravokutnikom (oknom) koje

za svaki položaj u slici javlja da li se tamo nalazi trokutni prometni znak. Položaj okna u slici se mijenja s lijeva na desno, odozdo prema gore. Kad se u nekoj poziciji unutar okna detektira znak, obrub tog okna se iscrtava u slici crvenom bojom.

Detekcija se obavlja pomoću naučene kaskade jakih klasifikatora, koji se pak sastoje od više slabih klasifikatora (slika 1.1). Svaki klasifikator za dani položaj i dimenzije okna u ulaznoj slici daje binaran rezultat klasifikacije – u oknu se nalazi znak ili se ne nalazi. Jaki klasifikator daje konačan rezultat na temelju rezultata svih slabih klasifikatora unutar njega. Svaki jaki klasifikator je zapravo jedna razina kaskade. Ako neka razina dâ pozitivan rezultat, klasifikacija se vrši idućom razinom kaskade. Ako je rezultat negativan, okno se pomiče na idući položaj u slici.

Slika 1.1: Organizacija kaskade klasifikatora

Opis algoritma kojim se detektira prometni znak dan je u poglavlju 2. U poglavlju 3 opisana je programska izvedba algoritma. Eksperimentalni rezultati prikazani su u poglavlju 4. Konačno, u poglavlju 5 dan je završni komentar u kojem se ocjenjuje uspješnost algoritma te ističu njegove prednosti i nedostaci.

2. Korišteni algoritmi i metode

2.1 Cannyjev detektor rubova

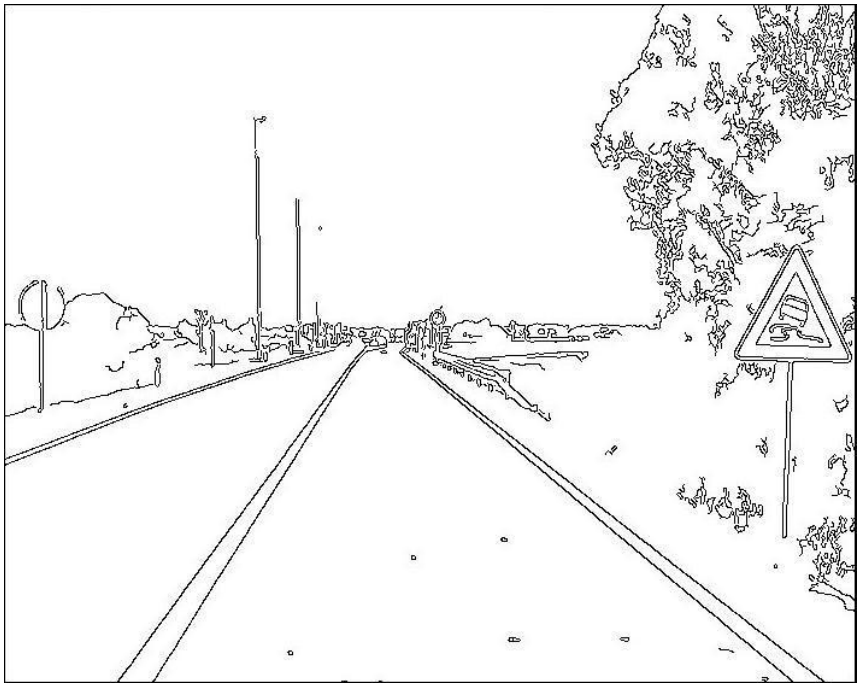
Metoda detekcije rubova koju je razvio J. Canny jedna je od najšire korištenih i najpopularnijih jer predstavlja dobar omjer složenosti i efikasnosti. Naravno, efikasnost ovisi o domeni uporabe. Cannyjev algoritam je programski ostvaren u okviru projekta koji je prethodio ovom završnom radu. Detaljni opis algoritma može se pronaći u literaturi [5] i tehničkoj dokumentaciji spomenutog projekta [4].

Cannyev algoritam sastoji se od četiri faze.

U prvoj fazi se ulazna slika transformira u sivu sliku i obrađuje Gaussovim filtrom koji uklanja uvijek prisutni šum. Parametar Gaussovog filtra je standardna devijacija (σ). Rezultat je blago zamućena slika. Druga faza algoritma za svaki piksel određuje iznos (amplitudu) vektora gradijenta i njegov smjer pružanja. Računanje gradijenta za pojedini piksel obavlja se na temelju razlike u intenzitetu susjednih piksela. U trećoj fazi potrebno je stanjiti rubove na slici gradijenta na debljinu od jednog do najviše dva piksela. Četvrta faza pomoću dva praga, visokog i niskog (t_{high}, t_{low}), za svaki piksel određuje da li on pripada rubu ili ne. Time se postiže binarizacija (nema svjetlijih i tamnijih piksela, piksel ili jest rub ili nije). Spomenuti pragovi su relativni u originalnoj implementaciji, ali za potrebe ovog rada su promijenjeni u apsolutne (vidi poglavlje 4.2).

Cannyjev algoritam kao osnovni izlaz daje binarnu sliku rubova (slika 2.1). Međutim, tokom obrade Cannyjev algoritam kao međukorak (konkretno u drugoj fazi) računa i matricu kutova - može se reći i sliku kutova, iako takva slika čovjeku vizualno nema smisla. Dakle, svaki piksel u slici osim informacije o tome

da li je rub ili nije ima i dodatnu informaciju - kut gradijenta. Pomoću kuta gradijenta dobiva se prilično dobra pretpostavka o kutu pod kojim se pojedini rubni element nalazi. Naravno, rubni element je zapravo jedan piksel i kao takav nema „kut“, ali ono na što se misli pod kut rubnog elementa jest smjer pružanja ruba na kojem se taj element nalazi. Tako bi rubni elementi na pravocrtnom horizontalnom rubu imali kut smjera pružanja 0°. U praksi se ne koristi smjer pružanja ruba nego smjer gradijenta ruba koji je okomit na smjer pružanja ruba. Tako primjerice rubovi na spomenutom horizontalnom rubu imaju kut gradijenta -90° (ako je iznad ruba tamnije područje nego ispod) ili 90° (ako je iznad ruba svjetlije područje nego ispod). Binarna slika i matrica kutova gradijenta koriste se u daljnjem postupku.



Slika 2.1: Primjer izlaza Cannyjevog algoritma. Parametri su: σ=1.5, t_{high}=0.9, t_{low}=0.7 (relativni pragovi).

2.2 Integralna slika

Algoritam detekcije znakova radi nad integralnim slikama [1]. U integralnoj slici svaki piksel sadrži zbroj vrijednosti piksela iznad i lijevo od njega. Pomoću takve slike zbroj piksela unutar bilo kojeg pravokutnog područja u slici može se izračunati uz samo četiri pristupa memoriji.

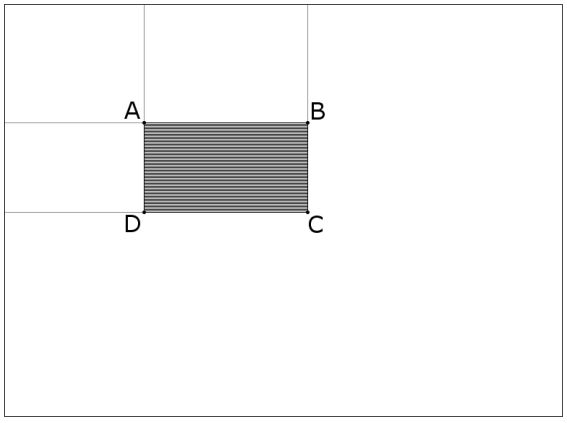
Integralna slika se može izračunati u samo jednom prolazu po slici na sljedeći način:

$$sum(x,y) = i(x,y) + sum(x-1,y) + sum(x,y-1) - sum(x-1,y-1) \qquad 2.1$$

Jednom kad je integralna slika načinjena, integral bilo kojeg pravokutnika u slici se može izračunati uz samo četiri pristupa memoriji na sljedeći način (slika 2.2):

$$\sum_{\substack{A(x)<x'<C(x) \\ A(y)<y'<C(y)}} i(x',y') = sum(A) + sum(C) - sum(B) - sum(D) \qquad 2.2$$

Računanje integrala proizvoljnog pravokutnog područja u slici je fundamentalno za daljnji postupak.



Slika 2.2: Izračun integralne slike

Za svaku ulaznu sliku računa se nekoliko integralnih slika, ovisno o broju odjeljaka u histogramu kuta gradijenta. Broj odjeljaka označava na koliko se područja dijeli raspon mogućih kutova te se izrađuje po jedna integralna slika za svako područje. Ako je npr. broj odjeljaka šest, izrađuje se sljedećih šest integralnih slika:

1. Integralna slika piksela s kutom gradijenta [0°-60]
2. Integralna slika piksela s kutom gradijenta [60°-120°]
3. Integralna slika piksela s kutom gradijenta [120°-180°]
4. Integralna slika piksela s kutom gradijenta [180°-240°]
5. Integralna slika piksela s kutom gradijenta [240°-300°]
6. Integralna slika piksela s kutom gradijenta [300°-360°]

Slika 2.3: Primjer skalirane[1] integralne slike.

[1] Da bi se mogla prikazati, slika mora biti skalirana na interval [0-255] jer vrijednosti piksela integralne slike približavanjem donjem desnom rubu mogu narasti i preko nekoliko desetaka tisuća. U samom algoritmu integralna slika se, naravno, ne skalira.

2.3 Detekcija znakova pomoću kaskade jakih klasifikatora

2.3.1 Kaskada jakih klasifikatora

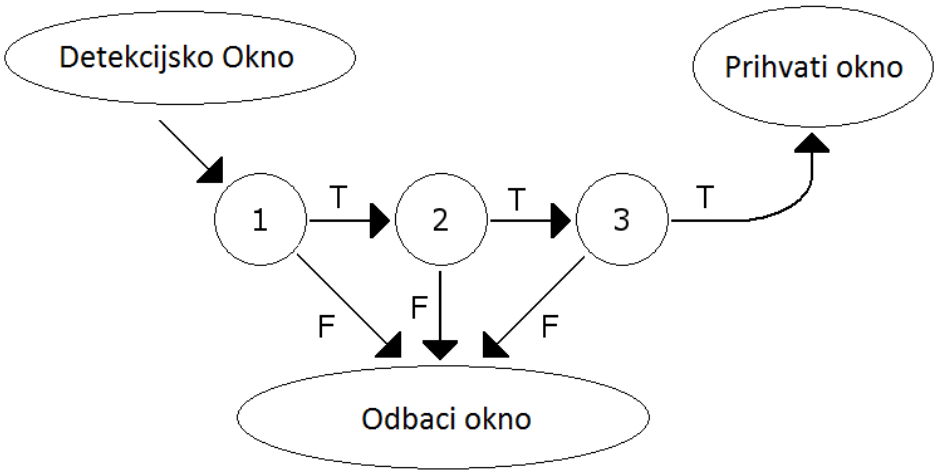
Kaskada jakih klasifikatora je serijski povezan lanac jakih klasifikatora u kojem vrijedi da se detekcija *n*-tim klasifikatorom u lancu obavlja samo ako je klasifikator s indeksom *n-1* dao pozitivan rezultat. Kako točno radi jaki klasifikator opisano je u poglavlju 2.3.2, zasad je dovoljno znati da je rezultat jakog klasifikatora binaran – ili je detekcija pozitivna (unutar okna je pronađen znak) ili je negativna. Na prvo mjesto u kaskadi postavlja se jaki klasifikator za kojeg je bitno da ima vrlo visok odziv te se zbog toga dopušta mala preciznost.

Odziv i preciznost su dvije korisne mjere koje se često koriste u problemima klasifikacije uzoraka. U ovom slučaju, odziv je veći što je više znakova na slici koji su detektirani (znakovi koji nisu detektirani ga umanjuju), a preciznost je veća što je više ispravnih detekcija u skupu svih detekcija (lažne detekcije ju umanjuju).

$$preciznost = \frac{ispravno\ pozitivni}{ispravno\ pozitivni + lažno\ \boldsymbol{pozitivni}} \tag{2.3}$$

$$odziv = \frac{ispravno\ pozitivni}{ispravno\ pozitivni + lažno\ \boldsymbol{negativni}} \tag{2.4}$$

Za prvu kaskadu je jako važno da trokutni znak bude detektiran, nema veze ako se uz to dobije i veliki broj lažnih detekcija. Zašto? Zato što se lažne detekcije filtriraju nadolazećim klasifikatorima u kaskadi. Slika 2.4 prikazuje detekciju pomoću kaskade s tri jaka klasifikatora (T označava pozitivnu detekciju, a F negativnu).



Slika 2.4 Detekcija pomoću kaskade klasifikatora

2.3.2 Jaki klasifikator

Jaki klasifikator se sastoji od više slabih klasifikatora koji daju binaran rezultat – pozitivnu ili negativnu detekciju. Slabi klasifikatori se povezuju u linearnu kombinaciju koja predstavlja jaki klasifikator:

$$h(x) = \begin{cases} 1, & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0, & inače \end{cases} \qquad 2.5$$

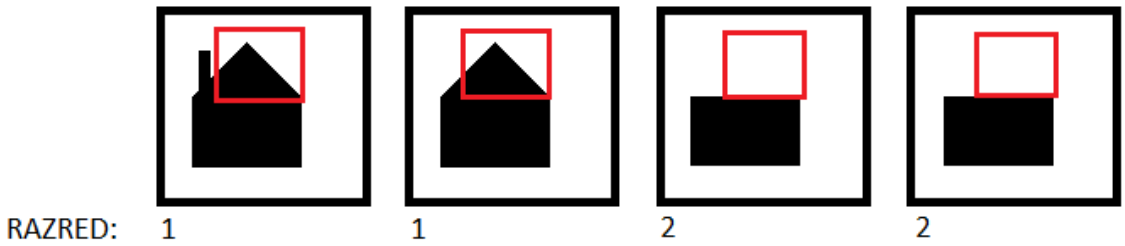
Jaki klasifikator određuje svoj izlaz pomoću izlaza slabih klasifikatora. Pritom se izlaz svakog pojedinog slabog klasifikatora množi s odgovarajućim težinskim faktorom *α*. Težinski faktori se računaju u fazi učenja i koriste se zato što nisu svi slabi klasifikatori jednako relevantni za konačni izlaz jakog klasifikatora. Ako je zbroj izlaza slabih klasifikatora pomnoženih odgovarajućim težinskim faktorima veći od polovice sume svih težinskih faktora, rezultat jakog klasifikatora je pozitivan. Koristi se polovica sume jer rezultati slabih klasifikatora iznose 0 ili 1. Valja pripaziti na to da se ne traži da barem polovica slabih

klasifikatora vrati pozitivan rezultat – čak i samo jedan slabi klasifikator s pozitivnim izlazom može rezultirati pozitivnim izlazom za cjelokupni jaki klasifikator ako mu je težinski faktor dovoljno velik.

2.3.3 Slabi klasifikator

Detekcija znaka se dakle obavlja tako što se za svaki položaj okna računa izlaz kaskade. Izlaz kaskade ovisi o izlazima jakih klasifikatora koji se ispituju redom dok jedan od njih ne vrati negativan rezultat ili dok se svi ne ispitaju (slika 2.4). Izlaz pojedinog jakog klasifikatora ovisi o izlazima njegovih slabih klasifikatora (izraz 2.5). Još je potrebno objasniti kako slabi klasifikatori računaju svoje izlaze.

Svaki slabi klasifikator ima četiri osnovne informacije: koordinate podokna, korišteni kut, prag integrala te orijentacija. Koordinate podokna su četiri para x,y vrijednosti koji označavaju točke koje čine podokno (pravokutnik unutar kojeg se računa integral). U fazi učenja se od svih mogućih podokana bira ono koje najbolje razdvaja pozitivne primjere od negativnih.



Slika 2.5. Biranje najboljeg podokna za klasifikaciju

Na slici 2.5. prikazano je biranje najboljeg podokna za odvajanje dvaju razreda[2]. Način na koji se odabire podokno opisan je u poglavlju 2.4.2, ovdje će biti objašnjeno samo zašto ovakav odabir podokna funkcionira.

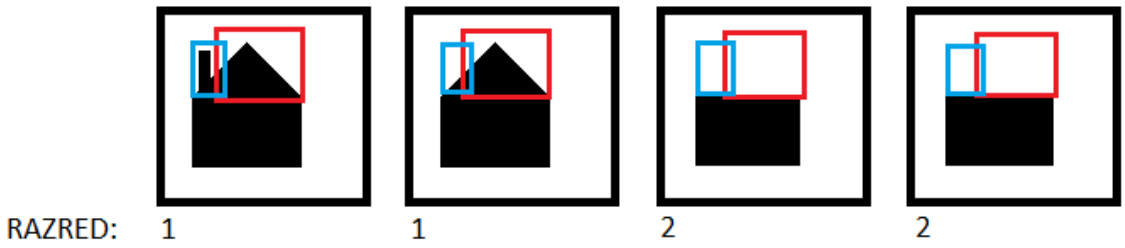
Na slici je podokno prikazano crvenim pravokutnikom. Vidljivo je kako je podokno odabrano tako da uzima u obzir onaj dio slike koji jasno razdvaja pozitivne primjere od negativnih. Ispitivanjem broja piksela unutar podokna (u daljnjem tekstu integral podokna) lako se može utvrditi da li se radi o pozitivnom ili negativnom primjeru. Integral podokna se brzo izračunava iz integralne slike napravljene u fazi pretprocesiranja. Svaki slabi klasifikator sadrži jedno podokno, premda su moguće i implementacije u kojima jedan klasifikator sadrži više podokana.

Prvi slabi klasifikator za primjer na slici 2.5. bi tako sadržavao podokno označeno crvenim pravokutnikom. Idući slabi klasifikator bi sadržavao sljedeće najbolje podokno, npr. isti pravokutnik kao na slici 2.5., ali pomaknut nekoliko piksela udesno. Važno je da jednom odabrano podokno više nikad ne može biti odabrano jer bi u suprotnom svi slabi klasifikatori u fazi učenja odabrali jedno te isto najbolje podokno.

Postupnim pridjeljivanjem podokana slabim klasifikatorima u jednom trenutku bi se iscrpila sva podokna koja u sebi sadrže krov te bi kao iduće podokno bio odabran plavi pravokutnik sa slike 2.6. Takvo podokno ima veću pogrešku od podokna sa slike 2.5. jer klasificira drugi primjer kao negativan, što je neispravno. U procesu učenja takav će klasifikator imati manju težinu jer ima veću pogrešku na primjerima za učenje.

[2] U problemima klasifikacije uzoraka u dvije skupine potpuno je nevažno da li jednu skupinu proglasimo negativnom, a drugu pozitivnom ili ih označimo numerički.

Važno je napomenuti da se ovdje radi samo o ilustrativnom primjeru. U konkretnoj implementaciji odabira najboljeg podokna algoritam staje ukoliko je greška klasifikacije odabranog podokna nula. U primjeru na slici 2.5 pronađeno podokno sve uzorke klasificira ispravno pa zapravo nikad ne bi došlo do odabira okna sa slike 2.6.



Slika 2.6. Biranje najboljeg podokna za klasifikaciju

Veličina okna za detekciju varira. U početku se slikom prolazi s manjim oknom, da bi se postupno veličina okna povećavala. Kako onda točno odrediti položaj podokna koji je zapisan u memoriji pomoću koordinata vrhova? Jednostavno – dimenzije okna podijele se s dimenzijama podokna kako bi se dobio faktor uvećanja te se koordinate vrhova podokna pomnože s tim faktorom. Tako je primjerice u ostvarenom algoritmu raspon dimenzija podokna [0-6]. Neka su koordinate gornjeg lijevog kuta podokna (3,3). To znači da će u oknu dimenzija 100 x 100 koordinate gornjeg lijevog kuta podokna biti (50,50).

Sad kad je jasno što je to podokno, potrebno je objasniti preostala tri podatka slabog klasifikatora.

Prag integrala je granica koja određuje izlaz klasifikatora. Primjer sa slike 2.5 je jednostavan jer pozitivni primjeri imaju krov, a negativni nemaju. U realnoj primjeni rijetko se događaju takvi trivijalni slučajevi. Češće će se dogoditi da npr. prvi uzorak ima vrijednost integrala unutar podokna 200, drugi uzorak ima vrijednost 150, treći uzorak ima vrijednost 100 i četvrti uzorak ima vrijednost 50.

Tada se za prag integrala može postaviti npr. 125. Jednostavnom usporedbom integrala unutar podokna s pragom može se utvrditi izlaz klasifikatora – u ovom slučaju bi bio izlaz 1 ako je vrijednost integrala veća (ili jednaka) od 125, a izlaz 0 ako je vrijednost integrala manja od 125.

Moguća je i obratna situacija – da klasifikator daje izlaz 0 za vrijednosti integrala veće od 125, a izlaz 1 za vrijednost integrala manje od 125. O kojem slučaju se radi određuje treći podatak kojeg sadrži slabi klasifikator – orijentacija. Drugim riječima, orijentacija određuje smjer znaka ≥ pri usporedbi izračunatog integrala i praga integrala u klasifikaciji.

Konačno, korišteni kut (zapravo, korišteni odjeljak) označava koji pikseli se uzimaju u obzir pri računanju integrala. Naime, ono što ovaj rad najviše razlikuje od rada Viole i Jonesa [1] jest činjenica da se pri računanju integrala u obzir uzimaju samo rubni elementi s određenim kutom gradijenta. Iz ulazne slike se nakon obrade Cannyjevim algoritmom računa onoliko integralnih slika na koliko je odjeljaka podijeljeno područje kutova, po jedna za svaki odjeljak (vidi poglavlje 2.2). Svaki slabi klasifikator nakon faze učenja sadrži podatak o tome koja integralna slika se uzima u obzir prilikom računanja integrala u podoknu. Ako je, primjerice, područje kutova podijeljeno na šest odjeljaka kao u primjeru u poglavlju 2.2, a korišteni odjeljak za neki slabi klasifikator iznosi 2, to znači da se za računanje integrala uzima treća integralna slika (indeksi idu od nule). Za spomenuti primjer iz poglavlja 2.2 to konkretno znači da se za taj klasifikator uzimaju u obzir samo pikseli s kutom gradijenta između 120° i 180°.

Pritom se mora napomenuti da se klasifikatori s različitim odjeljcima smatraju različitim klasifikatorima, čak i ako imaju identične koordinate podokna. Tako je zapravo proširen prostor slabih klasifikatora i samim time i poboljšana preciznost detekcije. Za detekciju nije dovoljno pronaći određene brojeve piksela na određenim mjestima u slici – potrebno je pronaći određene brojeve piksela pod

određenim kutovima na određenim mjestima u slici. Ako se na mjestu koje najbolje razdvaja pozitivne od negativnih primjera (kao npr. područje krova na slici 2.5) pojavljuju pikseli koji spadaju u različite odjeljke, u jaki klasifikator će možda biti dodano više slabih klasifikatora koji će imati iste koordinate podokna, ali će imati različite odjeljke. Drugim riječima, jedan slabi klasifikator može unutar određenog područja u slici zahtijevati x piksela pod kutem α, a drugi slabi klasifikator može unutar istog tog područja zahtijevati y piksela pod kutem β.

Ukoliko korisnik želi koristiti ostvareni algoritam tako da se prilikom detekcije ne uzima u obzir iznos kuta gradijenta, dovoljno je da se učenje obavi s jednim odjeljkom čime će svi mogući kutevi biti smješteni unutar tog jednog odjeljka.

2.4 Učenje kaskade jakih klasifikatora

2.4.1 Metoda adaBoost za učenje jakog klasifikatora

Učenje jakog klasifikatora pomoću slabih klasifikatora može se ostvariti na nekoliko načina – ECOC (Error Correcting Output Codes), stacking, bagging, boosting itd [7]. U literaturi [1] je korišten algoritam adaBoost [9], modifikacija osnovnog boosting algoritma (slika 2.8).

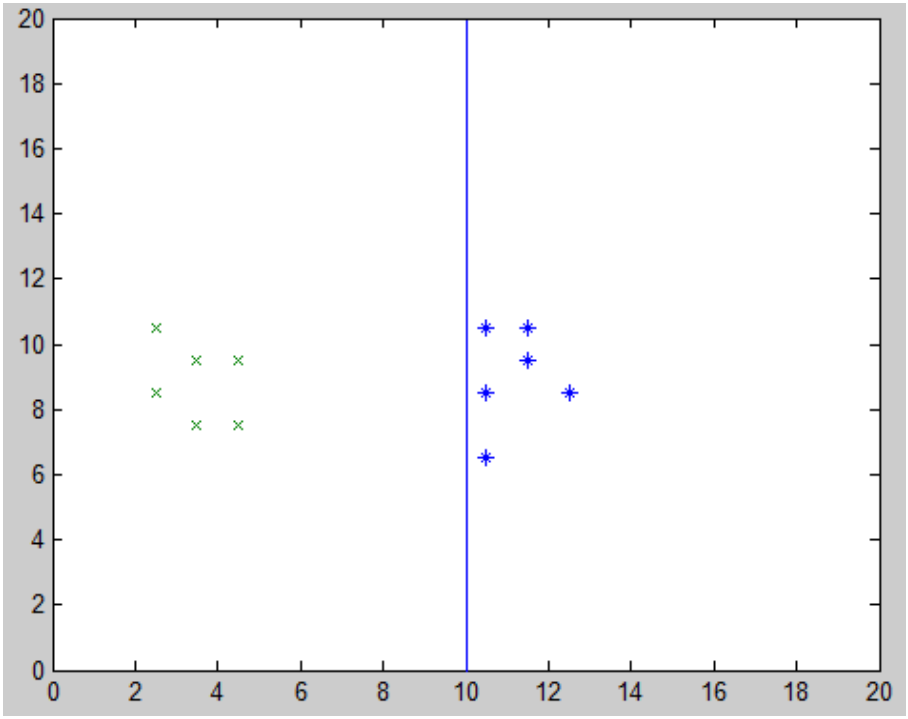
Prije opisa implementacije učenja jakog klasifikatora u ostvarenom algoritmu, potrebno je dati opis osnovnog principa metode učenja na jednostavnom primjeru. Primjer koji slijedi napisan je u programskom jeziku Matlab kao dio diplomskog rada.

Primjer je smješten u dvodimenzionalni prostor, dakle uzorci su zapravo točke u prostoru s koordinatama *x* i *y*. Uzorci su podijeljeni u dva razreda, „**+**“ i „**x**“. Neka „**+**“ budu pozitivni, a „**x**“ negativni. Potrebno je odvojiti pozitivne uzorke od

negativnih pomoću slabih klasifikatora koji su predstavljeni linijama paralelnim s osima. Svaki slabi klasifikator je, naime, samo jedna linija koja je paralelna ili s osi *x* ili s osi *y* te svi uzorci iznad, odnosno desno od nje pripadaju jednom razredu, a svi uzorci ispod, odnosno lijevo od nje drugom razredu (koji su pozitivni, a koji negativni određuje se pomoću varijable orijentacije slično kao što je opisano u prethodnom poglavlju). Slabi klasifikatori se mogu nalaziti samo na cjelobrojnim koordinatama kako bi se smanjio broj klasifikatora.

Ideja učenja je sljedeća: Postoje dva skupa za učenje, skup koji sadrži pozitivne uzorke i skup koji sadrži negativne uzorke. Svaki uzorak ima svoju težinu koja označava koliko taj uzorak pridonosi ukupnom računanju greške. Kako algoritam odmiče, uzorci koji su često ispravno klasificirani će imati manje težine tako da se daljnji tijek učenja sve više koncentrira na primjere koji su neispravno klasificirani. Moguće je koristiti sve uzorke iz skupa za učenje ili nasumično odabrati samo neke. Ako se odabiru samo neki uzorci iz skupa (to se naziva uzorkovanje skupa za učenje) tada vjerojatnost odabira pojedinog uzorka mora biti proporcionalna s njegovom težinom.

Potrebno je pronaći slabi klasifikator koji najbolje razdvaja pozitivne od negativnih uzoraka. U jednostavnom primjeru na slici 2.7 prikazan je slabi klasifikator koji najbolje razdvaja pozitivne od negativnih primjera. Klasifikator koji bi se nalazio na koordinatama x=9,8,7,6 ili 5 bio bi jednako dobar.



Slika 2.7. Trivijalan primjer razdvajanja uzoraka metodom adaBoost.

Slabi klasifikator koji najbolje razdvaja pozitivne od negativnih uzoraka u ovoj jednostavnoj 2D okolini s malim ukupnim brojem mogućih klasifikatora moguće je jednostavno pronaći tako da se ispitaju svi mogući klasifikatori te se odabere onaj koji radi minimalnu pogrešku pri klasifikaciji. Jednom kad je pronađen najbolji slabi klasifikator, isti se uklanja iz popisa dostupnih slabih klasifikatora. Pomoću pogreške odabranog klasifikatora (za koju se zna da je minimalna moguća jer je odabran najbolji klasifikator) određuje se faktor promjene težina uzoraka. Na početku algoritma svi uzorci imaju jednake težine koje se potom korigiraju kako algoritam odmiče. Kao što je već rečeno, ideja je da uzorci koji se ispravno klasificiraju manje doprinose izračunu pogreške od uzoraka koji se neispravno klasificiraju. Time se naglasak učenja stavlja na pogrešno klasificirane primjere kako bi se naučilo i njih ispravno klasificirati. Ako se radi uzorkovanje skupa za učenje, potrebno je osvježiti i vjerojatnosti odabira uzoraka koje moraju biti proporcionalne s težinama.

Faktor promjene težine uzoraka se računa kao

$$\beta_j = \frac{\varepsilon_j}{1 - \epsilon_j} \tag{2.6}$$

Na temelju faktora promjene težine (u nastavku beta faktor) određuje se promjena težine pojedinih uzoraka kao

$$p_{j+1,t} = \beta_j^{1-\delta_{jt}} * p_{jt}, \forall t \tag{2.7}$$

pri čemu je *t* indeks uzorka, a *j* označava iteraciju algoritma. Dobivene težine je još potrebno normalizirati tako što se svaku težinu podijeli sa sumom svih težina.

Težine uzoraka su potrebne samo za vrijeme učenja. Beta faktori koji određuju kako se mijenjaju težine uzoraka potrebni su i nakon učenja te se stoga spremaju u jaki klasifikator. Svaki slabi klasifikator u jakom klasifikatoru mora imati pripadajući beta faktor. Zašto? Jer beta faktor označava koliko taj slabi klasifikator utječe na ukupnu odluku jakog klasifikatora. Naime, pomoću faktora *β* se određuje faktor *α* iz izraza 2.5. kao

$$\alpha = \log_{10} \frac{1}{\beta} \tag{2.8}$$

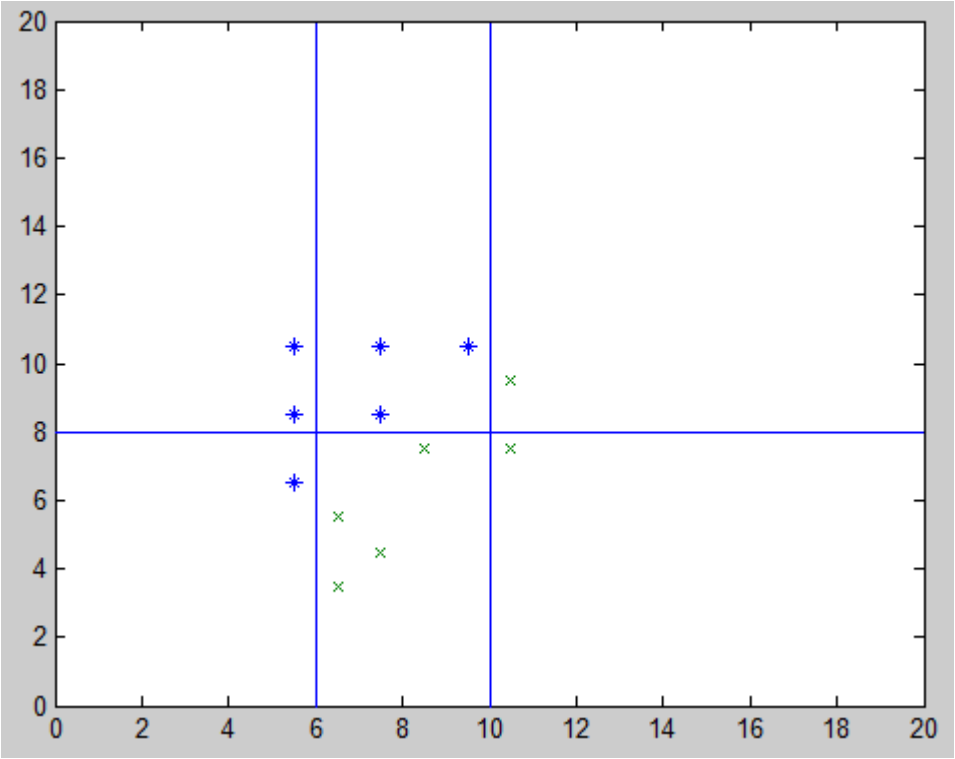
Nakon ažuriranja težina uzoraka, odabrani slabi klasifikator i pripadajući beta faktor spremaju se u jaki klasifikator te proces kreće ispočetka – traži se novi najbolji slabi klasifikator pri čemu je prethodni izbačen iz skupa klasifikatora da ne bi bio ponovno odabran. Potom se računa pogreška, određuje se beta faktor, ažuriraju se težine uzoraka te se slabi klasifikator i beta faktor spremaju u jaki klasifikator. Taj proces se ponavlja dok nije ispunjen zadani broj iteracija ili dok

dobiveni jaki klasifikator ne bude u stanju sve primjere iz skupa za učenje ispravno klasificirati. Pseudokod algoritma za učenje dan je na slici 2.8.

```
# X = {x_t, r_t}_{t=1}^N ... ukupni skup za učenje
# p = {p_jt} ... distribucija za odabir X_j
# X_j ... skup za učenje j-tog algoritma d_j
# δ_jt ∈ {0,1} ... da li je t-ti rezultat d_j pogrešan?
# ε_j ∈ [0,1] ... ponderirana ukupna greška algoritma d_j
# β_j ∈ [0,1⟩ ... faktor promjene p_jt za ispravno klasificirane x_t
def AdaBoostLearn(X):
  p_1t=1/N, ∀t
  for j=1,...,L:
    # nauči klasifikator d_j na ponderiranim uzorcima
    X_j=uzorkuj(X,p)
    d_j=nauči(X_j)

    # odredi ponderiranu pogrešku ε_j
    δ_jt=(d_j(x_t)≠r_t), ∀t
    ε_j=∑_t p_jt δ_jt
    if ε_j>1/2:
      L=j-1; break;

    # odredi faktor promjene težina uzoraka β_j
    β_j=ε_j/(1-ε_j)

    # ažuriraj i normaliziraj težine uzoraka
    p_{j+1,t}=β_j^{1-δ_jt} p_jt, ∀t
    p_{j+1,t}=p_{j+1,t}/∑_t p_{j+1,t}
  return {d_j, β_j}
```

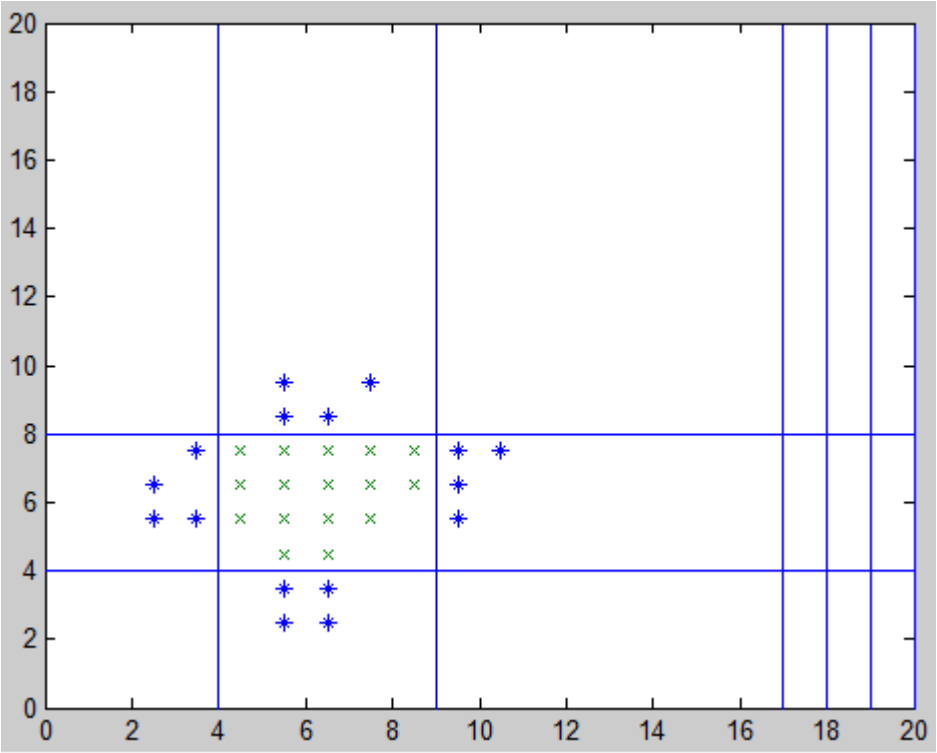
Slika 2.8. Algoritam adaBoost: učenje

Na slici 2.9 prikazan je nešto složeniji skup uzoraka.



Slika 2.9: Primjer razdvajanja uzoraka metodom adaBoost.

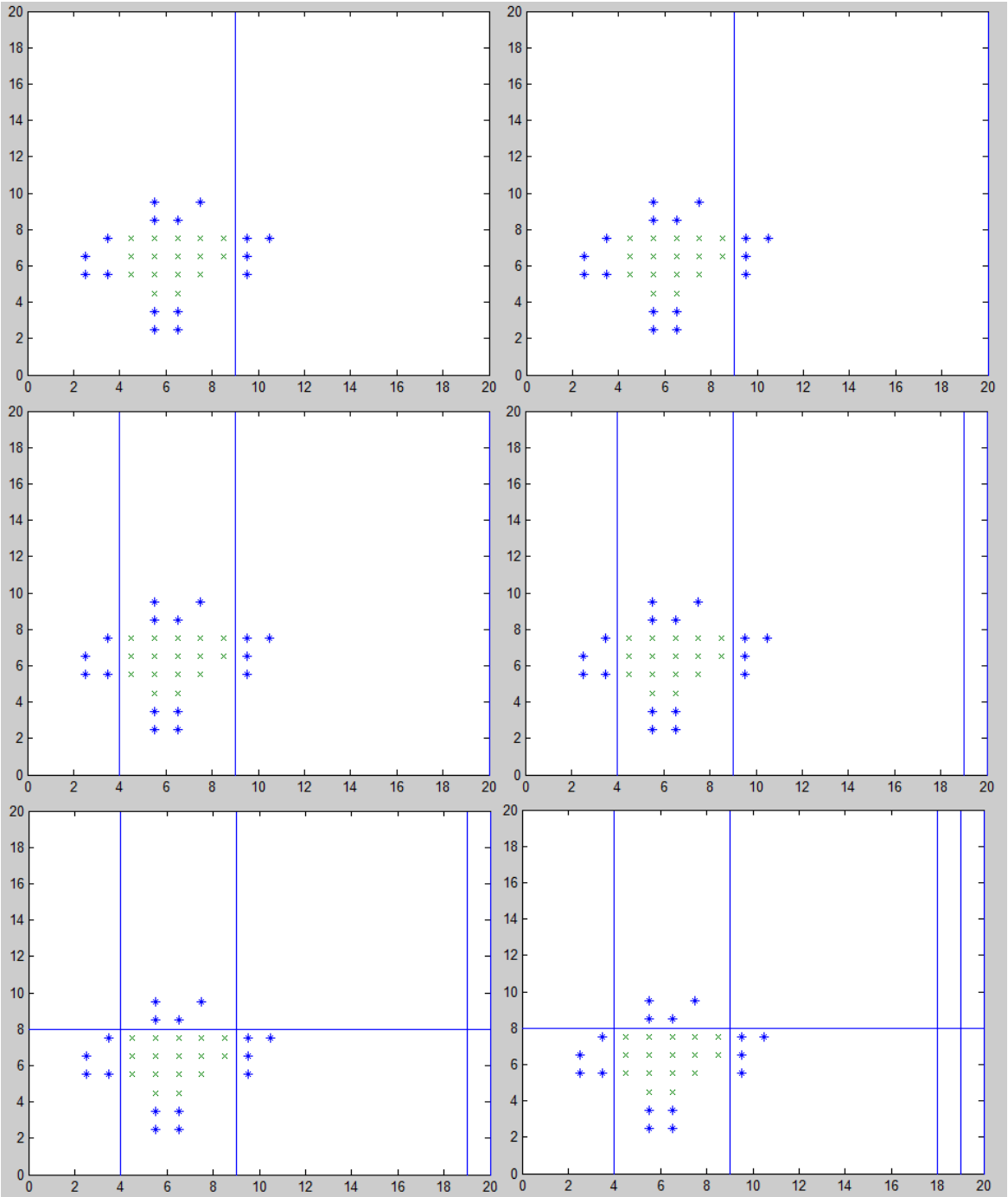
Na slici 2.10 prikazan je još složeniji skup uzoraka, skup koji nije linearno razdvojiv. Naime, na primjeru sa slike 2.9. moguće je povući kosi pravac između pozitivnih i negativnih uzoraka i time ih savršeno razdvojiti. Jaki klasifikator raspolaže samo s vodoravnim i okomitim pravcima pa su mu stoga trebala tri slaba klasifikatora da odvoji uzorke. Međutim, na slici 2.10 nije moguće povući bilo kakav pravac kojim bi se pravilno razdvojilo pozitivne i negativne uzorke. Algoritam adaBoost je svejedno savršeno točno klasificirao sve uzorke pomoću jakog klasifikatora sastavljenog od 8 slabih klasifikatora. Kako?



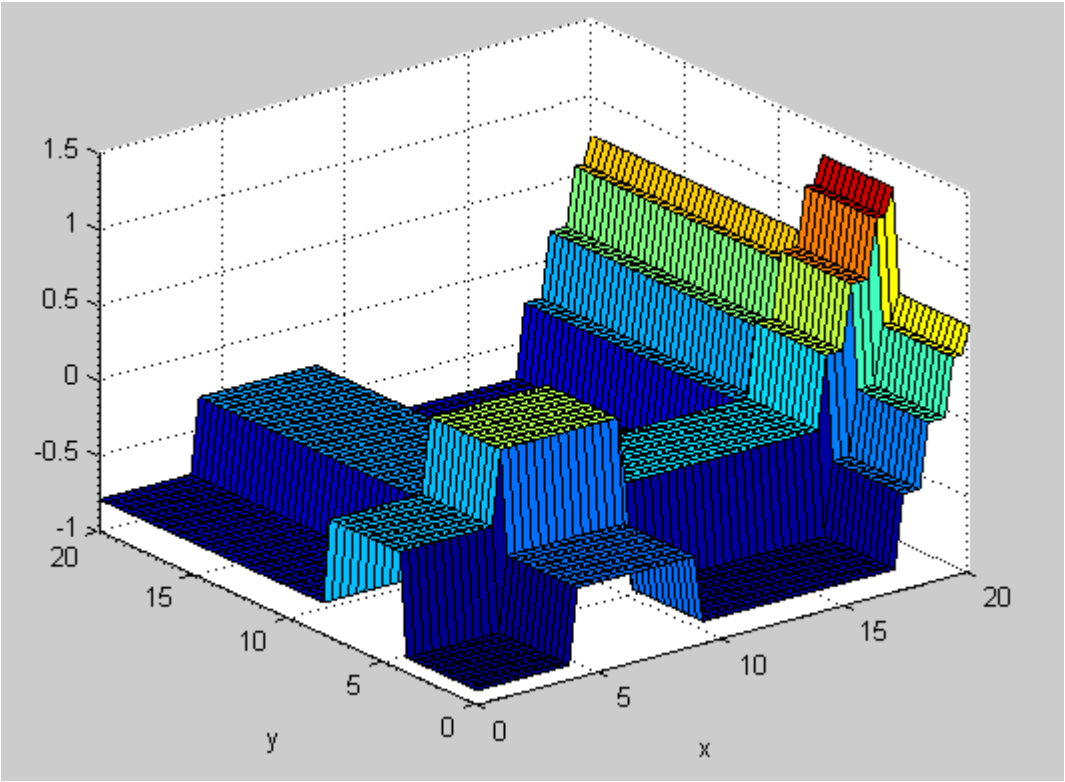
Slika 2.10: Primjer razdvajanja linearno nerazdvojivih uzoraka metodom adaBoost.

Neka četiri skupine od po četiri pozitivna uzorka budu „lijeva“, „desna“, „gornja“ i „donja“. Prvo je pronađen slabi klasifikator x=9 (slika 2.11) koji proglašava sve uzorke sa svoje desne strane pozitivnim, a sve sa svoje lijeve strane negativnim. To znači da je krivo klasificirano 12 uzoraka (lijeva, gornja i donja skupina) kojima su zbog toga porasle težine. Budući da uzorci s većom težinom više doprinose izračunu pogreške, algoritam će u idućem koraku naglasak staviti na ispravno klasificiranje „otežanih“ uzoraka. Stoga je odabran klasifikator x=20 koji sve uzorke proglašava pozitivnim. Težine uzoraka koji su bili „najteži“ su se smanjile, ali su zato povećane težine negativnih uzoraka jer oni su neispravno klasificirani. Postupak se ponavlja sve dok kombinacijom klasifikatora svi uzorci ne budu klasificirani ispravno.

Ilustracija prvih nekoliko iteracija ovog postupka prikazana je na slici 2.11. Linearna kombinacija slabih klasifikatora sa slike 2.11 prikazana je u tri dimenzije na slici 2.12.



Slika 2.11. Ilustracija učenja jakog klasifikatora metodom adaBoost za prvhi šest iteracija



Slika 2.12. Linearna kombinacija slabih klasifikatora sa slike 2.10 u tri dimenzije.

2.4.2 Učenje jakog klasifikatora

Sama implementacija učenja jakog klasifikatora u ostvarenom algoritmu detekcije trokutnih prometnih znakova ponešto se razlikuje od osnovnog modela opisanog u prethodnom poglavlju. Izračun težinskih faktora uzoraka te beta faktora je potpuno isti, razlika je samo u načinu odabira najboljeg slabog klasifikatora.

Podaci od kojih se sastoji slabi klasifikator opisani su u poglavlju 2.3.3. Prije učenja svi slabi klasifikatori moraju biti stvoreni i spremljeni u listu slabih klasifikatora. Stvaranje slabih klasifikatora je proces u kojem se za svaku

kombinaciju dimenzija podokna, orijentacije i odjeljka izrađuje po jedan klasifikator.

Podokno je veličine koja može iznositi od 1×1 do 6×6[3] i može se nalaziti na svim mogućim položajima u oknu. Prag integrala u ovoj fazi ostaje prazan te se popunjava tokom učenja. Budući da ima puno mogućih vrijednosti za prag integrala, bilo bi neefikasno stvoriti po jedan klasifikator za svaku moguću vrijednost jer bit to ukupan broj klasifikatora uvećalo onoliko puta koliko je mogućih pragova. Umjesto toga, obavlja se sljedeći postupak za određivanje optimalnog praga integrala.

Pri evaluaciji svakog pojedinog slabog klasifikatora u fazi učenja potrebno je odrediti optimalan prag za promatrani klasifikator te odrediti koliku pogrešku isti klasifikator radi pri klasifikaciji uzoraka za učenje. Optimalan prag za slabi klasifikator se određuje tako što se za svaki uzorak za učenje izračuna vrijednost integrala u podoknu tog slabog klasifikatora te se uzorci poredaju uzlazno prema vrijednosti integrala. Prag se tada može odrediti u jednom prolazu kroz listu sortiranih uzoraka. Za svaki element u listi računaju se vrijednosti 4 varijable:

- T^+: ukupna suma težina pozitivnih uzoraka,
- T^-: ukupna suma težina negativnih uzoraka,
- S^+: suma težina pozitivnih uzoraka koji se nalaze ispod trenutno promatranog uzorka,
- S^-: suma težina negativnih uzoraka koji se nalaze ispod trenutno promatranog uzorka

[3] Koordinate vrhova podokna se naknadno skaliraju na koordinate okna kako je objašnjeno u poglavlju 2.3.3 na stranici 12.

Pri prolasku kroz listu sortiranih uzoraka, vrijednosti varijabli S^+ i S^- se osvježavaju u svakoj iteraciji. Ovakav prolazak kroz listu sortiranih uzoraka zapravo simulira iterativno postavljanje pragova. Naime, za svaki uzorak u listi prag se postavlja na vrijednost koja se nalazi negdje između vrijednosti integrala trenutno promatranog uzorka u listi i prethodno promatranog uzorka u listi. Točan iznos nije bitan za proces učenja, uzorke suprotnih razreda koji imaju vrijednosti integrala npr. 1000 i 1500 jednako će dobro klasificirati iznos praga 1200 i iznos praga 1300. Budući da je nemoguće znati koja granica bi bila optimalna za primjere koji će kasnije biti klasificirani, po zakonu vjerojatnosti najispravnije je postaviti prag na polovicu vrijednosti, dakle za dani primjer 1250.

Za svaki uzorak u listi, izraz

$$e = min\left(S^+ + (T^- - S^-), S^- + (T^+ - S^+)\right) \qquad 2.9$$

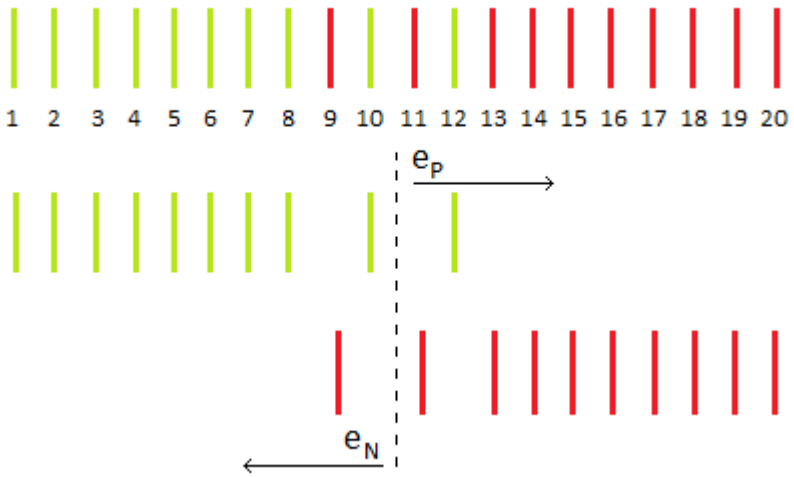
govori kolika bi greška bila napravljena da se prag postavi na danu vrijednost, dakle između vrijednosti integrala promatranog uzorka i vrijednosti integrala uzorka koji se u listi nalazi prije njega.

Nakon što se prođe cijela lista potrebno je vidjeti gdje je greška e bila minimalna. Ako je *n* indeks uzorka u listi za koji je e bila minimalna, tada se vrijednost praga integrala za slabi klasifikator kao što je već rečeno postavlja na aritmetičku sredinu vrijednosti integrala za uzorak s indeksom *n* i vrijednosti integrala za uzorak s indeksom *n-1*.

Ilustracija ovog postupka može se vidjeti na slici 2.13. Ukupan broj uzoraka je 20, pri čemu je 10 pozitivnih i 10 negativnih. Kad se poredaju uzlazno prema vrijednosti integrala, uzorci čine listu koja se vidi u gornjoj polovici slike. Minimalna greška e ostvarena je za 11. uzorak:

$$
\begin{aligned}
e &= min\left(S^{+} + (T^{-} - S^{-}), S^{-} + (T^{+} - S^{+})\right) \\
&= min\left(9 + (10 - 1), 1 + (10 - 9)\right) \\
&= min(10,2) \\
&= 2
\end{aligned}
\tag{2.10}
$$

U prethodnom računu radi jednostavnosti nisu korištene težine uzoraka. U implementaciji se uzimaju u obzir težine uzoraka tako da pojedina varijabla ne iznosi samo broj uzoraka već sumu težina uzoraka koji se uzimaju u obzir.



Slika 2.13. Vizualizacija greške optimalnog praga

Kad je ustanovljeno da je greška $e = 2$ nastupila za 11. uzorak u listi, prag se postavlja na polovicu vrijednosti integrala 11. uzorka i 10. uzorka (crtkana linija). Na slici se vidi da je greška $e = 2$ uistinu minimalna - nemoguće je povući liniju između bilo koja dva druga uzorka, a da broj pogrešno klasificiranih uzoraka bude jednak ili manji od dva.

Objašnjeno je kako se pronalazi najbolji klasifikator za dani skup primjera za učenje te kako se računa njegova pogreška. Ostatak učenja potpuno je isti kao u poglavlju 2.4.1. Valja uočiti kako i učenje opisano u poglavlju

2.4.1 i učenje opisano u ovom poglavlju (dakle učenje koje je implementirano u ostvarenom programu za detekciju znakova) slijede pseudokod sa slike 2.8, a razlikuju se jedino u načinu kako implementiraju liniju $d_j = nauči(x_j)$. Ima još jedna manja razlika – opisane metode učenja ne koriste uzorkovanje kao što je navedeno u pseudokodu, nego rade sa svim dostupnim primjerima za učenje, ali to nema nikakav utjecaj na sâm proces učenja. Uzorkovanje se koristi kad je skup primjera za učenje prevelik te bi učenje predugo trajalo.

2.4.3 Učenje kaskade jakih klasifikatora

Kaskada jakih klasifikatora radi na način da prvi klasifikator u kaskadi propusti velik broj detekcija. Velika većina tih detekcija će biti lažna, ali time se smanjuje rizik od odbacivanja ispravne detekcije. Potom se taj skup detekcija filtrira tako što svaki novi klasifikator u kaskadi odbacuje određene rezultate klasifikatora prije njega. Ovakav pristup rezultira brzom detekcijom jer će se u većini slučajeva vršiti klasifikacija s prvim ili tek nekoliko prvih klasifikatora. Tek će u rijetkim situacijama doći do klasifikacije s nešto većim brojem klasifikatora (situacije u kojima se unutar okna nalazi objekt koji sliči na trokutni prometni znak).

Međutim, ovaj pristup ima i jednu potencijalnu manu. Ako je okno postavljeno iznad trokutnog prometnog znaka, a prvi klasifikator u kaskadi dâ negativan rezultat, nema ispitivanja ostalih klasifikatora već se okno odmah pomiče dalje. Naizgled se čini da bi tako moglo doći do velikog broja lažno negativnih rezultata koji su u ovoj domeni primjene mnogo opasniji od lažno pozitivnih (veća je pogreška ne detektirati znak nego detektirati znak i još nekoliko objekata uz cestu koji nisu znakovi). Stoga se u fazi učenja nastoji minimizirati broj lažno negativnih rezultata koliko god je moguće. Manji broj lažno negativnih rezultata se postiže većom „propusnošću“ klasifikatora, što znači da će se povećati broj

lažno pozitivnih rezultata, ali to je očekivano – upravo zbog toga se i radi kaskadiranje.

Budući da kaskada radi na upravo opisani način, i učenje mora biti prilagođeno tome. Način kako to ostvariti je sljedeći:

Na samom početku odabiru se dva glavna skupa primjera, pozitivni i negativni. Skup pozitivnih primjera sastoji se od malih slika na kojima su samo trokutni prometni znakovi (slika 2.14). Skup negativnih primjera sastoji se od kompletnih slika scene ispred vozila (nazovimo ih slike pozadina) kakve će stizati na ulaz programa, ali bitno je da na tim slikama nema trokutnih prometnih znakova (slika 2.15).

Slika 2.14. Primjer uzoraka u pozitivnom skupu za učenje

Slika 2.15. Primjer slike pozadine u negativnom skupu za učenje

Pri učenju svakog pojedinog stupnja kaskade za pozitivne primjere koriste se svi uzorci iz skupa pozitivnih uzoraka. Međutim, za negativne primjere potrebno je oponašati izgled slike unutar okna koja mora dati negativan rezultat klasifikacije. Stoga je iz slika pozadina potrebno izrezati manje slike (nazovimo ih podslike) sličnih dimenzija kao kod pozitivnih primjera (slika 2.16). U početku se svaka podslika izrezuje iz nasumično odabrane slike pozadine na nasumično odabranom mjestu. Pritom se mora paziti na jednu važnu stvar – u prvom stupnju učenja kaskade sama kaskada još ne sadrži nijedan klasifikator pa je moguće odabrati bilo kakve podslike, međutim u fazama nakon prve potrebno je da dotad stvorena kaskada klasificira sve odabrane podslike kao pozitivne (dakle namjerno se biraju slike koje kaskada krivo klasificira). Time se učenje koncentrira na prepoznavanje i odbacivanje lažno pozitivnih primjera. Nakon određenog broja iteracija učenja kaskade postaje teško nasumično pronaći podsliku koja se klasificira pozitivno. Tada biranje podslika prelazi u drugi način

rada gdje se više slika i pozicija ne odabiru nasumično, nego se po redu pretražuju sve slike i sve pozicije dok se ne nakupi dovoljan broj podslika.

Slika 2.16. Primjer podslika u negativnom skupu za učenje

3. Programska implementacija

3.1. Struktura programske implementacije

U programskoj implementaciji algoritma korištena je ljuska `cvsh`[4] (computer vision shell). Ljuska je razvijena kako bi se olakšalo i pojednostavnilo eksperimentiranje s algoritmima računalnog vida u programskom jeziku C++. Dodatne informacije i detalji o korištenju ljuske dostupni su u tehničkoj dokumentaciji projekta koji je prethodio ovom radu (Cannyjev detektor rubova) [6].

Programski kôd je raspodijeljen u dvije osnovne cjeline: učenje i detekcija.

3.2. Učenje

Datoteka **alg_learnCascade.cpp** je glavna komponenta koja obavlja učenje kaskade jakih klasifikatora. Komponenta, između ostalog, sadrži klasu `alg_learnCascade` koja posjeduje metodu `alg_learnCascade::process():`

```
void alg_learnCascade::process(
      const img_vectorAbstract& src,
      const win_event_vectorAbstract&,
      int)
```

Metoda `process()` prvo učitava slike za učenje pomoću funkcije `std::vector<img_wrap> getImages(std::string filePath)` definirane u datoteci `ext_load_images.cpp` koja učitava sve slike koje se nalaze u

[4] Izvorni kôd ljuske dostupan je na adresi http://www.zemris.fer.hr/~ssegvic/src/cvsh_src_090310.tar.gz

zadanom direktoriju i sprema ih u radnu memoriju kao vektor slika. Tip podatka `img_wrap` definiran je u samoj ljusci.

Ovdje se stvara i skup svih slabih klasifikatora `std::vector<Classifier> classifiers` iz kojeg se biraju najbolji slabi klasifikatori pri izgradnji jakih klasifikatora. Potrebno je zadati broj odjeljaka za kut i veličinu podokna.

Potom se pokreće osnovna petlja za učenje kaskade. Prvo se popunjava negativni skup za učenje podslikama izrezanim iz skupa slika pozadina. Pomoću trenutne iteracije petlje određuje se da li se podslike traže nasumično ili potpuno. Kad je pribavljeno dovoljno podslika pokreće se učenje jakog klasifikatora. Broj slabih klasifikatora u jakom klasifikatoru je parametar postupka.

Kad je jaki klasifikator napravljen, sprema se u kaskadu te petlja ide ispočetka. Broj iteracija petlje, dakle broj jakih klasifikatora u kaskadi zadaje se kao parametar postupka.

Funkcija za pronalaženje jakog klasifikatora je `findSC()`:

```
StrongClassifier findSC(
      vector<Classifier> classifiers,
      int numOfBins,
      vector<img_wrap> imgsVecP,
      vector<img_wrap> imgsVecN,
      int numOfLearnIters);
```

definirana u datoteci `ext_findStrongClassifier.cpp`. U njoj se iz primljenih slika izrađuju objekti klase `Sample` definirane u datoteci `ext_adaBoostLearn.hpp`, a potom se za skup slabih klasifikatora primljen kao parametar i izrađene objekte tipa Sample poziva funkcija `learn()` uz zadani

broj iteracija. Funkcija `learn()` definirana je u datoteci ext_adaBoostLearn.cpp na sljedeći način:

```
StrongClassifier learn(
      vector<Sample> samples,
      vector<Classifier> classifiers,
      int numOfIterations);
```

Samo učenje ne radi sa sirovim slikama, nego s objektima klase `Sample` zato što je za svaki uzorak za učenje potrebno pamtiti i neke dodatne atribute. Klasa `Sample` izgleda ovako:

```
class Sample{
public:
      vector<img_wrap*> imagesP;
      vector<img_wrap> images2;
      int numOfImages;
      int classOfSample;
      double probability;
      double rouletteWheelSum;
      double weight;
      double featureValue;
      bool operator<(Sample &rhs) {
return featureValue < rhs.featureValue;
}
};
```

Od važnijih atributa klase valja spomenuti vektor slika koji sadrži integralne slike podijeljene po odjeljcima, razred u koji spada uzorak (pozitivni ili negativni), težina uzorka i iznos integrala u podoknu (eng. feature value).

Funkcija `learn()` obavlja učenje opisano u poglavlju 2.4.2. Od važnijih funkcija valja spomenuti `evalClassifiers()`:

```
Classifier evalClassifiers(
      vector<Classifier> classifiers,
      vector<Sample> data);
```

definiranu u datoteci `ext_adaBoostEvaluate.cpp`. U algoritmima koji koriste metodu učenja adaBoost implementacija ove funkcije može se bitno razlikovati -

primjerice, ista funkcija napisana u Matlab-u pri izradi jednostavnog 2D primjera učenja opisanog u poglavlju 2.4.1 je potpuno drukčija u implementaciji. Međutim, bitno je da funkcija za zadani skup slabih klasifikatora i primjera za učenje funkcija vraća klasifikator koji najbolje razdvaja pozitivne od negativnih primjera.

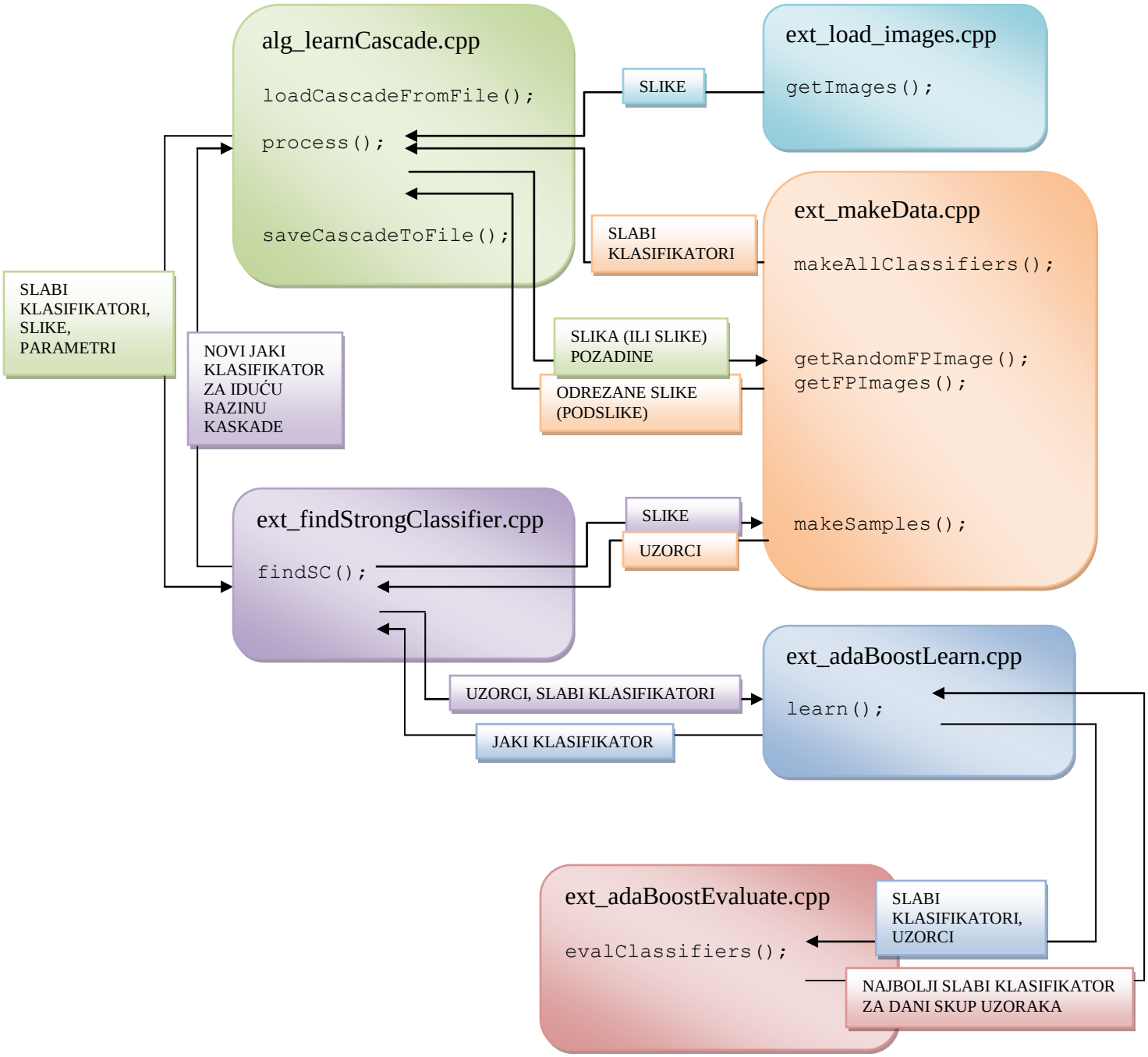
Druga važna funkcija korištena u funkciji `learn()` je `simpleClassifyImage()` također definirana u datoteci `ext_adaBoostEvaluate.cpp`:

```
int simpleClassifyImage(
      Classifier cl,
      img_wrap* img);
```

Funkcija za najbolji klasifikator dobiven pomoću funkcije `evalClassifiers()` vraća rezultat klasifikacije nad zadanom slikom. Funkcija služi za osvježavanje težina uzoraka jer se pomoću nje određuje koji uzorci su krivo klasificirani i kao takvi trebaju u nastavku učenja dobiti veću težinu.

U datoteci `ext_adaBoostClassify.cpp` definirane su pomoćne funkcije koje klasificiraju primljeni objekt tipa `Sample` koristeći jaki klasifikator ili kaskadu. Funkcije se ne koriste u konačnoj implementaciji jer se kaskada primjenjuje izravno na isječke slike unutar okna, a ne na `Sample` objekte radi bržeg izvođenja.

Pojednostavljeni dijagram učenja (bez pomoćnih klasa i privatnih metoda) prikazan je na slici 3.1.



Slika 3.1. Dijagram toka učenja s osnovnim klasama i funkcijama

3.3. Detekcija

Kad učenje završi, naučena kaskada se sprema u datoteku. Da bi se mogla obavljati detekcija, potrebno je promijeniti treći argument koji se predaje ljusci pri pokretanju (prvi argument je putanja do mape sa slikama, drugi argument je naredba izvršavanja, a treći je algoritam koji se koristi). Za učenje je potrebno predati niz znakova „`-a=alg_learnCascade`“, dok je za detekciju potrebno predati niz znakova „`-a=alg_detectSignUsingCascade`“.

Detekcija je mnogo jednostavnija od učenja. Datoteka `alg_detectSignUsingCascade.cpp` sadrži klasu istog naziva koja između ostalog sadrži i metodu `process()`[5]. Pri konstrukciji objekta te klase prima se ulazna slika u kojoj se u daljnjem postupku detektiraju trokutni znakovi.

U metodi `process()` se prvo iz željene datoteke učitava kaskada pomoću funkcije `loadCascadeFromFile()` koja je definirana kao javna metoda klase `Cascade` (dakle, stvara se prazan objekt tipa `Cascade` te se poziva njegova metoda za popunjavanje kaskade iz datoteke). Potom se slika obrađuje Cannyjevim detektorom rubova pomoću funkcije `processImage()` definirane u `ext_makeData.cpp`. Kao rezultat obrade dobivaju se binarna slika rubova i slika kuteva gradijenta. Na temelju njih se izrađuju integralne slike podijeljene po kutevima pomoću funkcije `makeAngleImgs()` definirane također u `ext_makeData.cpp` te funkcije `integralImage()` definirane u `ext_integralImage.cpp`.

Dobivene integralne slike se zajedno s kaskadom predaju funkciji `signDetectionBoost()` definiranoj u `ext_signDetectionBoost.cpp` kao:

[5] Datoteka alg_learnCascade.cpp također sadrži klasu istog naziva koja između ostalog posjeduje i metodu process(). Naime, svi algoritmi korišteni u ljusci nasljeđuju klasu alg_base koja zahtjeva implementaciju metode process(). Na taj način ljuska „zna“ kako pozvati algoritam.

```
void signDetectionBoost(
      int margin,
      win_ann& annDetections,
      vector<img_wrap> intImgs,
      Cascade& cascade)
```

pri čemu se margina dobiva u Cannyjevom algoritmu, a `annDetections` je adresa anotacija (crvenih pravokutnika pomoću kojih se prikazuju detekcije u slici).

Funkcija detekciju obavlja pomoću čak pet ugnježđenih petlji:

U vanjskoj petlji se u svakoj iteraciji povećava veličina detekcijskog okna.
Unutar nje se u dvostrukoj petlji svaki piksel slike postavlja kao donji lijevi kut detekcijskog okna.
Za dani položaj okna se u novoj petlji prolazi kroz vektor jakih klasifikatora kaskade.
Konačno, za svaki jaki klasifikator se u konačnoj petlji prolazi kroz sve njegove slabe klasifikatore i za svakog od njih se računa integral unutar podokna koji se potom uspoređuje s pragom te se kao rezultat dobiva 0 ili 1. Rezultat se množi s faktorom α koji se računa pomoću vektora koji sadrži β faktore i dodaje u zbroj (vidi izraz 2.5). Konačan rezultat linearne kombinacije (*h(x)* u izrazu 2.5) predstavlja izlaz trenutno promatranog jakog klasifikatora. Ako je izlaz pozitivan, traži se rezultat idućeg jakog klasifikatora u vektoru (četvrta petlja), a ako je negativan prelazi se na idući piksel (druga i treća petlja).

Ako svi jaki klasifikatori za dani položaj okna daju pozitivan rezultat, iscrtava se crveni pravokutnik na toj poziciji u slici.

4. Eksperimentalni rezultati

4.1. Organizacija rezultata

Kod testiranja algoritama koji služe za prepoznavanje objekata, eksperimentalni rezultati se dijele na tri skupine:

- Eksperimenti u kojima algoritam uspješno pronalazi željene objekte (pozitivni rezultati)
- Eksperimenti u kojima algoritam kao detekcije prijavljuje neželjene objekte (lažno pozitivni rezultati)
- Eksperimenti u kojima algoritam ne pronalazi željene objekte (lažno negativni rezultati)

Testiranje je obavljeno na 678 znakova.

4.2. Opis izvođenja s parametrima

4.2.1 Obrada slike

Prvotni parametri Cannyjevog algoritma su σ =2.0, t_{high} = 26.0, t_{low} = 2.0[6]. Pragovi nisu relativni kako je u početku planirano, već su promijenjeni na apsolutne. Naime, ako se velika slika obradi Cannyjem s relativnim parametrima i iz nje se uzme odsječak, taj odsječak neće izgledati onako kako bi izgledao da je prvo izvađen iz slike, a potom obrađen Cannyjem. Relativni pragovi označavaju postotak, a sâm iznos praga ovisi o udjelu rubova u slici. S apsolutnim pragovima sve slike dobivaju "jednak tretman". Broj odjeljaka, dakle

[6] Kasnije su opisane dodatne kaskade koje su naučene na drugačijim parametrima.

broj integralnih slika iznosi 6. Početna ispitivanja su imala broj odjeljaka jednak 3 s početnim kutom 90°, što daje iznose gradijenta od 90°, 210° i 330°. Učenje sa 6 odjeljaka, dakle s gradijentima iznosa 30°, 90°, 150°, 210°, 270° i 330° je puno brže napredovalo (prije su se iscrpili svi lažno pozitivni uzorci iz skupa slika pozadina).

4.2.2 Učenje

U pozitivnom skupu za učenje nalazilo se 1400 slika trokutnih prometnih znakova, dok se za negativni skup u svakoj iteraciji biralo novih 2000 lažno pozitivnih uzoraka isječenih iz slika pozadina kojih je ukupno 400.

Prvi jaki klasifikator u kaskadi sastoji se od samo dva slaba klasifikatora. Broj slabih klasifikatora se potom za svaki novi jaki klasifikator u kaskadi povećava 2 puta. Kad dođe do 32, broj slabih klasifikatora se više ne uvećava. U praksi se pokazalo da daljnje udvostručavanje broja klasifikatora ne doprinosi rezultatima, a uvelike povećava trajanje učenja.

Prva naučena kaskada sastoji se od 8 razina (kasnije su naučene još tri dodatne kaskade, što je opisano u poglavlju 4.5). Algoritam je dodavao razine kaskade dok je imao primjera za učenje, a kad su iscrpljeni svi lažno pozitivni uzorci iz slika pozadina učenje je stalo. Valja napomenuti da su Viola i Jones u svom radu [1] imali 38 razina, što bi dalo naslutiti da informacija o kutu gradijenta rubnih elemenata mnogo doprinosi procesu klasifikacije pa je potrebno manje razina u kaskadi. Međutim, valja imati na umu da je ovdje broj slabih klasifikatora udvostručavan sa svakom novom razinom kaskade do pete razine (dakle broj klasifikatora po razinama je 2, 4, 8, 16, 32 te ostaje 32 za sve daljnje razine), dok su Viola i Jones imali drugačiju raspodjelu (2, 10, 25, 25, 50, 50, 50...) pa kaskade nisu baš usporedive. Osim toga, valja imati na umu da su

Viola i Jones radili detekciju lica osoba, što je kudikamo zahtjevniji zadatak od trokutnih znakova.

Učenje se odvijalo u tri faze:

1. faza: Lažno pozitivni uzorci se iz slika pozadina biraju nasumično. Prvo se nasumično odabere slika pozadine, potom se nasumično odaberu koordinate te se naposlijetku nasumično odabere veličina uzorka. Proces se ponavlja dok se ne pribavi dovoljan broj lažno pozitivnih uzoraka.
2. faza: Lažno pozitivni uzorci se iz slika pozadina biraju slijedno, ali veličina uzorka se bira nasumično. Kad se prođu sve koordinate u slici pozadine, prelazi se na iduću dok se ne pribavi dovoljan broj lažno pozitivnih uzoraka. U idućoj iteraciji učenja obrada kreće od zadnje obrađene slike.
3. faza: Lažno pozitivni uzorci se iz slika pozadina biraju potpunim pretraživanjem. Kreće se od minimalne veličine uzorka i iz slika pozadina se slijedno (kao u prethodnoj fazi) pribavljaju svi lažno pozitivni uzorci. Ako se za danu veličinu okna ne pronađe dovoljno lažno pozitivnih uzoraka, veličina se uvećava za faktor skaliranja i obrada kreće ispočetka od prve slike. U idućoj iteraciji učenja obrada kreće od zadnje obrađene slike i zadnje veličine okna.

Prva faza je trajala tokom prve dvije iteracije učenja kaskade (dakle učenja prva dva jaka klasifikatora). Već tada je postalo poprilično teško pronaći nasumične lažno pozitivne uzorke pa je postupak prešao na fazu 2. Nakon što je obrađena zadnja slika pozadine, postupak je prešao na fazu 3. Valja primjetiti da u fazi 1 algoritam prije ili kasnije dobavi dovoljno slika, pitanje je samo koliko će to potrajati. U fazama 2 i 3 to ne mora biti slučaj. Tako u fazi 2 u početku svi uzorci mogu biti izvađeni iz jedne ili eventualno nekoliko slika pozadina, ali kad kaskada postane preciznija isti broj lažno pozitivnih uzoraka će morati biti pribavljen iz nekoliko desetaka slika pozadina. Kad se obradi zadnja slika pozadine u fazi 2, vrlo vjerojatno neće biti pribavljen dovoljan broj lažno pozitivnih uzoraka. Tada postupak transparentno prelazi u fazu 3 pri čemu

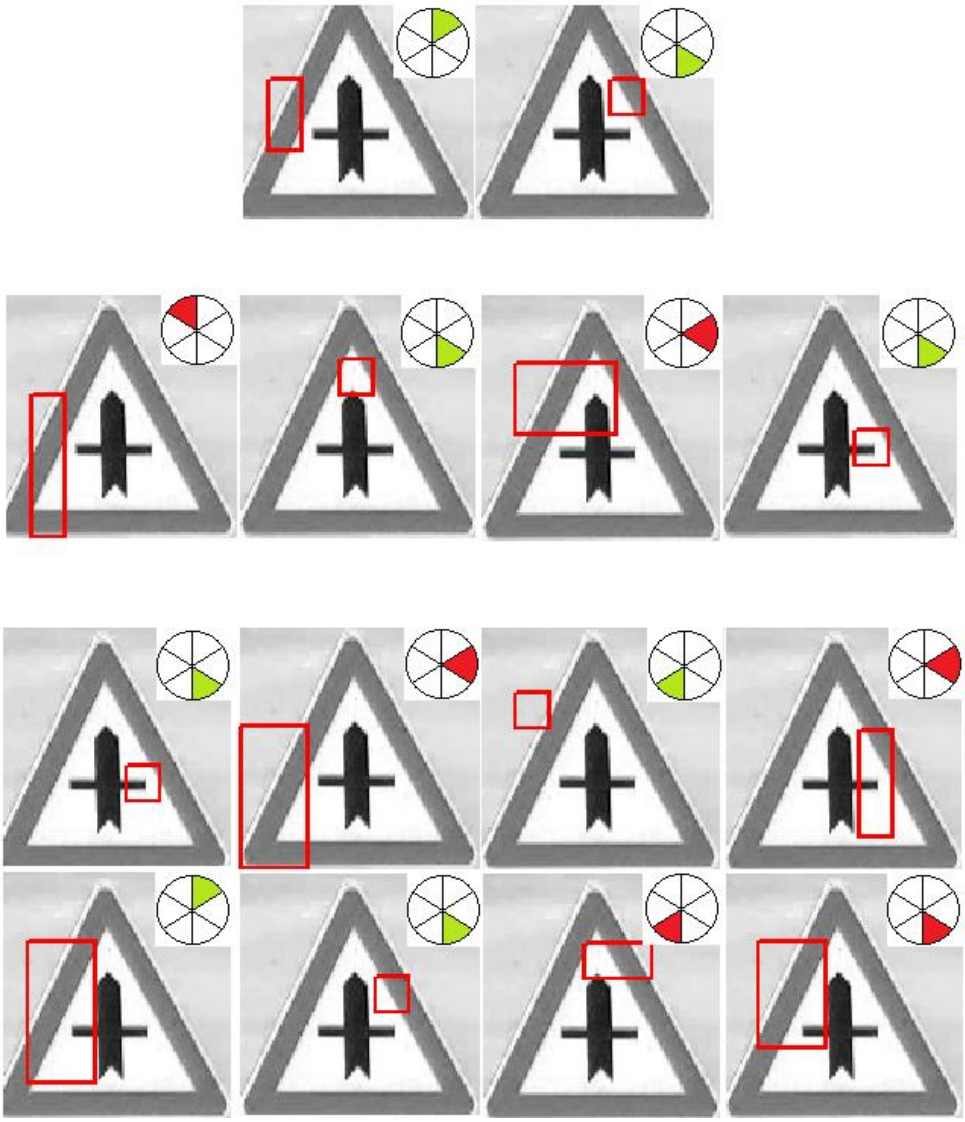
zadržava dotad pribavljene uzorke iz faze 2. Isto vrijedi i za obradu zadnje slike pozadine za određenu veličinu okna u fazi 3 – postupk zadržava pribavljene uzorke, ali kreće od prve slike s uvećanim oknom.

Nakon što se u fazi 3 iz zadnje slike pozadine za maksimalnu veličinu okna pribave sve lažno pozitivni uzorci, obavlja se učenje jakog klasifikatora s pribavljenim uzorcima i algoritam staje.

Minimalna i maksimalna veličina uzorka iznose 30 i 90. Faktor uvećanja u 3. fazi iznosi 1.1 (dakle, nakon dobavljanja svih lažno pozitivnih uzoraka veličine 30 veličina uzorka se povećava na 30 x 1.1 = 33 itd.).

Glavni utjecaj na trajanje učenja imaju broj odjeljaka, broj uzoraka u skupovima za učenje te broj slabih klasifikatora unutar jakog. Za ranije navedene parametre - 6 odjeljaka, 1400 odnosno 2000 uzoraka te udvostručavanje broja slabih klasifikatora krenuvši od 2 - učenje traje otprilike 10 sati.

Podokna za prve tri razine naučene kaskade prikazana su na slici 4.1. Odsječci kruga u gornjem desnom uglu svake slike znaka označavaju nad kojim odjeljkom radi pojedini slabi klasifikator, pri čemu valja naglasiti da je označen kut ruba, a ne kut gradijenta. Zelena boja označava orijentaciju, dakle da li integral mora biti veći od praga (zeleno) ili manji (crveno).

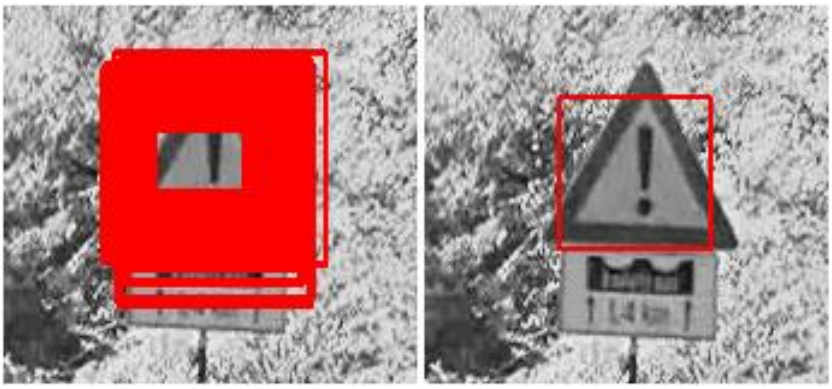


Slika 4.1. Podokna u prve tri razine kaskade

4.2.3 Detekcija

U detekciji moraju biti korišteni isti parametri kao u učenju (početna veličina okna = 30, faktor skaliranja = 1.1, broj odjeljaka = 6 te oni parametri Cannyjevog algoritma s kojima je obavljeno učenje pojedine kaskade).

Rezultat detekcije su desetine i ponekad stotine detekcija koje se preklapaju (slika 4.2 lijevo). Zbog toga je kao oblik postprocesiranja obavljeno grupiranje detekcija. Sve detekcije koje se preklapaju spadaju u jednu grupu. Postojanje preklapanja između dviju danih detekcija je lako utvrditi jer su poznate koordinate pravokutnika koji čine detekcije. Kad se utvrde sve grupe u slici, računaju se prosječne koordinate vrhova svih detekcija u pojedinoj grupi te se kao konačna detekcija prijavljuje okno s takvim usrednjenim koordinatama (slika 4.2 desno).



Slika 4.2. Eliminacija preklapanja detekcija.

4.3. Demonstracija rezultata

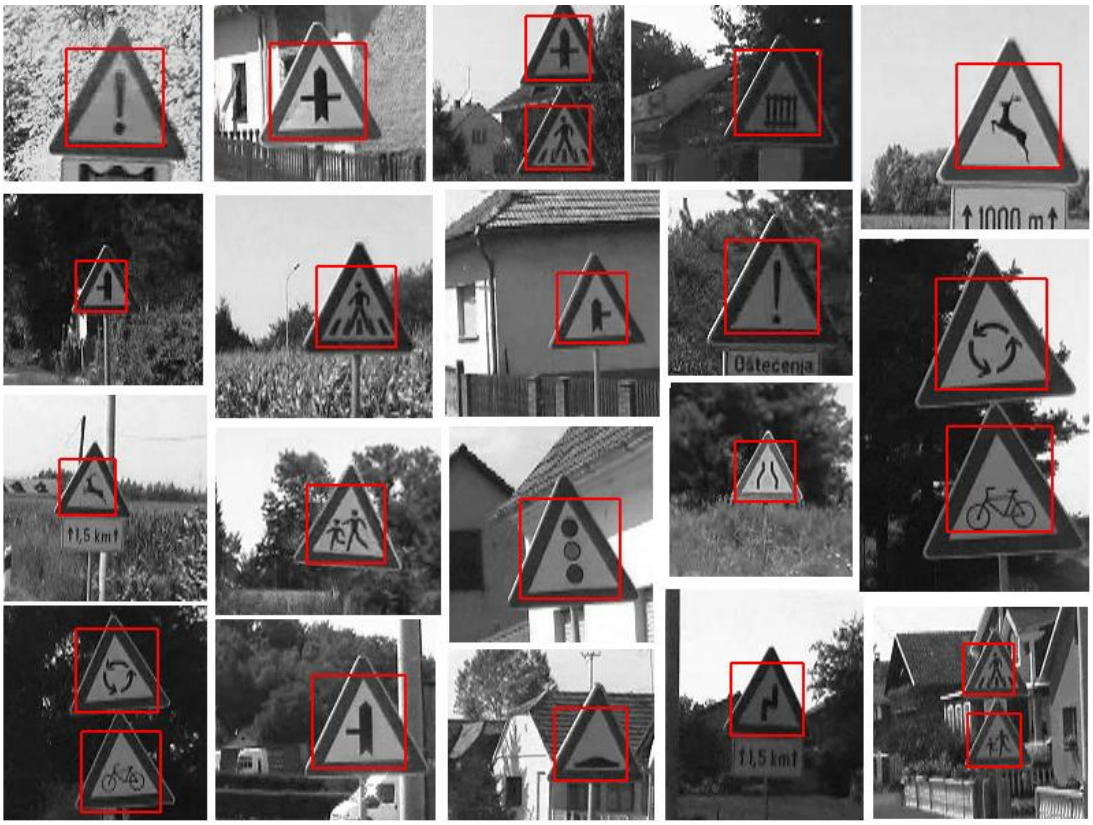
4.3.1 Pozitivni rezultati

Pozitivni rezultati su oni u kojima je detektiran prometni znak. Prikazana su četiri pozitivna rezultata (slike 4.3, 4.4 i 4.5) te još jedna skupina pozitivnih rezultata (slika 4.6).

Slika 4.3

Slika 4.4

Slika 4.5



Slika 4.6

4.3.2 Lažno pozitivni rezultati

Na slikama 4.7 i 4.8 prikazani su primjeri lažno pozitivnih rezultata. Lažno pozitivne rezultate moguće je izbjeći ili barem svesti na minimum detaljnijim učenjem:

- povećanjem broja slika pozadina dobit će se veći skup različitih uzoraka koje se mora naučiti razlikovati od prometnih znakova
- povećanjem broja uzoraka u skupu za učenje dobit će se „općenitiji“ jaki klasifikatori koji su u stanju razlikovati prometne znakove od mnoštva negativnih uzoraka
- povećanjem broja slabih klasifikatora dobit će se preciznije određivanje karakteristika znaka pa će ih se moći bolje razlučiti od negativnih primjera (povećanje ne smije biti preveliko – klasifikator bi se mogao previše usredotočiti na dane uzorke, a želimo zadržati određenu općenitost kako bi radio dobro i na uzorcima koje nije „vidio“)
- povećanjem broja jakih klasifikatora dobit će se dodatne razine kaskade koje bi mogle biti u stanju razlučiti teške primjere kao na slici 4.8 koji poprilično nalikuju na trokutni prometni znak

Nameće se pitanje – zašto onda nisu provedena spomenuta poboljšanja u ostvarenom programu? Odgovor je jednostavan - navedena poboljšanja poprilično povećavaju trajanje učenja (posebice povećanje broja uzoraka u skupu za učenje), a nije moguće egzaktno odrediti što točno povećati i koliko puta te koliki će to točno imati utjecaj na preciznost i odziv. Prvi rezultati su bili vrlo loši, ali su povećavanjem broja slika pozadina, primjera za učenje, slabih klasifikatora itd. u konačnici postignuti rezultati koji su predstavljeni u radu. Opća praksa u ovakvim postupcima jest povećavati parametre dok se ne postignu rezultati koji su zadovoljavajući za potrebe domene primjene algoritma.

Slika 4.7

Slika 4.8

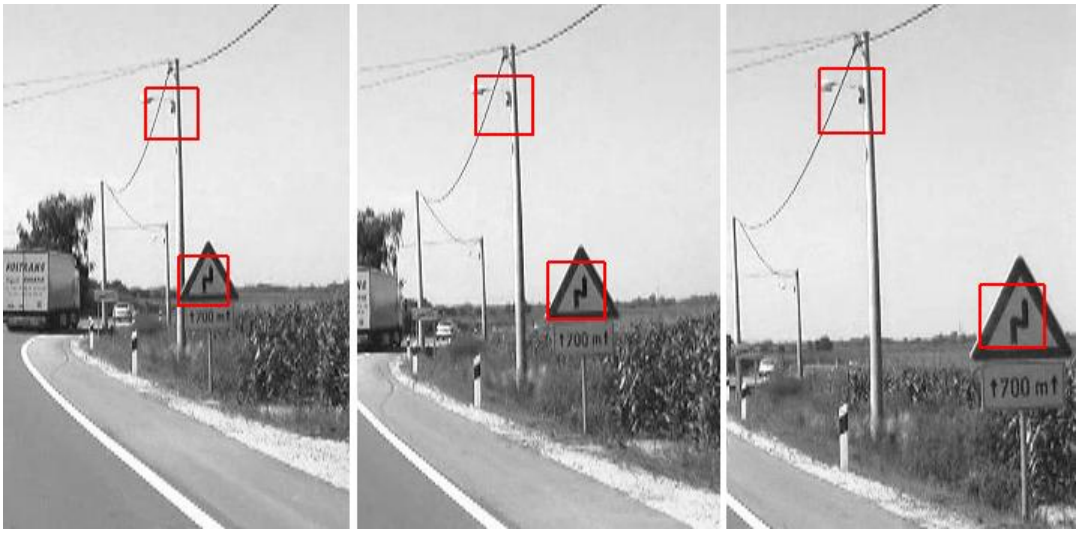
Ovdje valja spomenuti jedno zanimljivo opažanje: na slici 4.7 je dobiven lažno pozitivan rezultat kojeg je bez sumnje moguće izbjeći preciznijim učenjem, ali što je s lažno pozitivnim rezultatom na slici 4.8? Radi se o trokutnom objektu svijetle boje okruženom s tamnom granicom koji u sredini ima manje objekte tamne boje. Kako detektor može „znati" da se ne radi o prometnom znaku? Ovdje je vidljivo koliko je ljudski mozak superioran u odnosu na računala – i malo dijete će na danoj slici bez problema pronaći prometni znak, pri čemu neće imati ni najmanje dileme oko spornog trokutnog dijela kuće jer vidi da je to kuća, a ne znak. Bilo bi potrebno ugraditi određene semantičke informacije u detektor objekata, što je problematično, ali ne zbog toga **kako** to napraviti nego **što** točno napraviti.

Slika 4.9. Većina lažno pozitivnih detekcija su pročelja kuće koja su trokutnog oblika.

Što se tiče konkretnog primjera s kućom, većina lažno pozitivnih detekcija su zapravo takvi slučajevi (slika 4.9) pa je za rješenje tog specifičnog problema moguće naučiti posebnu razinu kaskade kojoj bi kao skup negativnih primjera za učenje bio predan skup slika na kojima su sporni objekti (pročelje kuće). Sličan slučaj je i s okruglim znakovima (slika 4.10), premda su takvi slučajevi mnogo rjeđi od onih s krovovima.

Neki scene su snimljene u kratkom vremenskom razmaku pa se zapravo radi o vrlo sličnim slikama. Tada je moguće da se lažno pozitivne detekcije jave u svim slikama iste scene (slika 4.11). Naravno, pri evaluaciji rezultata svaka takva lažno pozitivna detekcija je zasebno brojana.

Slika 4.10. Nekoliko lažno pozitivnih detekcija su okrugli znakovi

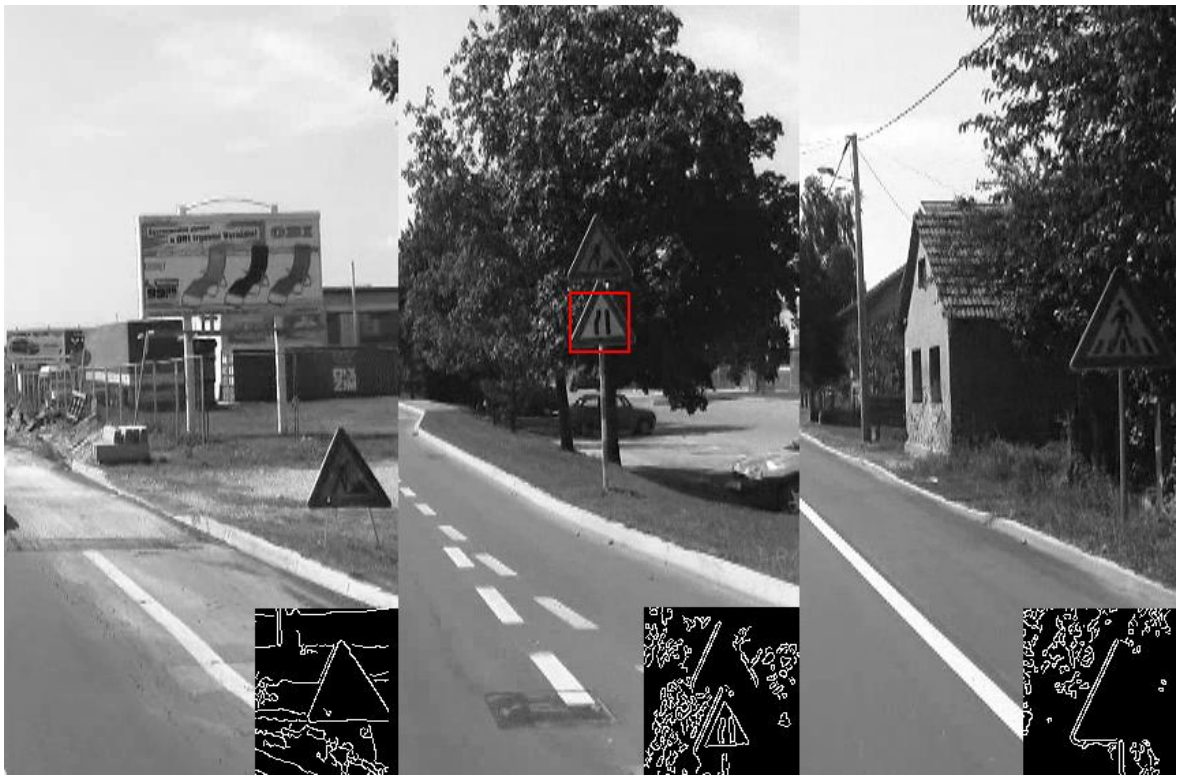


Slika 4.11. Lažno pozitivna detekcija na tri uzastopne slike iste scene

4.3.3 Lažno negativni rezultati

Kad algoritam ne detektira znak u slici govori se o lažno negativnoj detekciji.

Na slici 4.12 prikazana su tri lažno negativna rezultata koji su nastali kao posljedica neispravnog izlaza Cannyjevog algoritma detekcije rubova. U donjem desnom kutu pojedinog dijela slike vidi se da na unutar područja znaka nisu detektirani rubni elementi. Ovaj problem se može riješiti izbjegavanjem faze usporedbe s pragom u Cannyjevom algoritmu (vidi poglavlje 4.5).



Slika 4.12

Slika 4.13. Lažno negativna detekcija malog znaka

Na slici 4.13 prikazan je lažno negativan rezultat detekcije znaka malih dimenzija. Premda je znak dovoljno velik da bi ga okno moglo detektirati (naučena kaskada je namijenjena za znakove minimalne veličine 30x30 piksela), znak svejedno nije detektiran. Uočeno je da mali znakovi znaju biti problematični za algoritam detekcije, vjerojatno zato što piksel ima puno veće značenje unutar okna 30x30 nego unutar okna 90x90 pa je moguće da zbog samo nekoliko „zalutalih“ piksela rezultat klasifikacije bude neispravan.

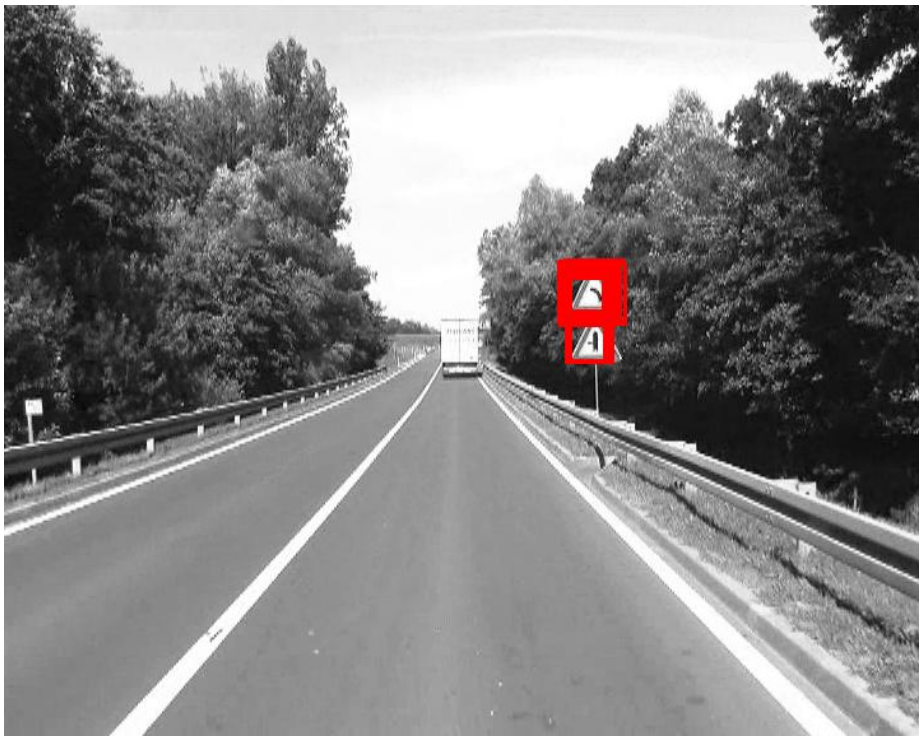
Na slici 4.14 prikazan je lažno negativan rezultat čiji uzrok nije sasvim jasan. Znak jest nešto manji, a kao što je upravo rečeno mali znakovi nerijetko predstavljaju problem, ali nije tako malen kao na slici 4.13 već spada u skupinu nešto većih znakova koji se većinom uspješno detektiraju (slika 4.18). Moja pretpostavka je da je razlog nedetekcije to što znakova ovog tipa ima jako malo u skupu za učenje, a većina ostalih znakova u skupu za učenje ima sliku negdje

u središtu znaka. Klasifikatori su zbog toga naučili da rubni elementi moraju postojati i oko središta znaka, a ne samo pri dnu.

Slika 4.14. Lažno negativan rezultat čiji uzrok nije poznat

Na slici 4.15 prikazan je lažno negativan rezultat do kojeg je došlo zbog neispravnog grupiranja rezultata. Na slici 4.16 može se vidjeti da su na oba prometna znaka javljene detekcije, ali su se malo preklapale te ih je algoritam grupiranja svrstao u istu grupu. Dakle, ovaj lažno negativan rezultat nije „krivnja“ samog detektora, već postupka postprocesiranja rezultata.

Slika 4.15 Lažno negativan rezultat kao posljedica grupiranja detekcija

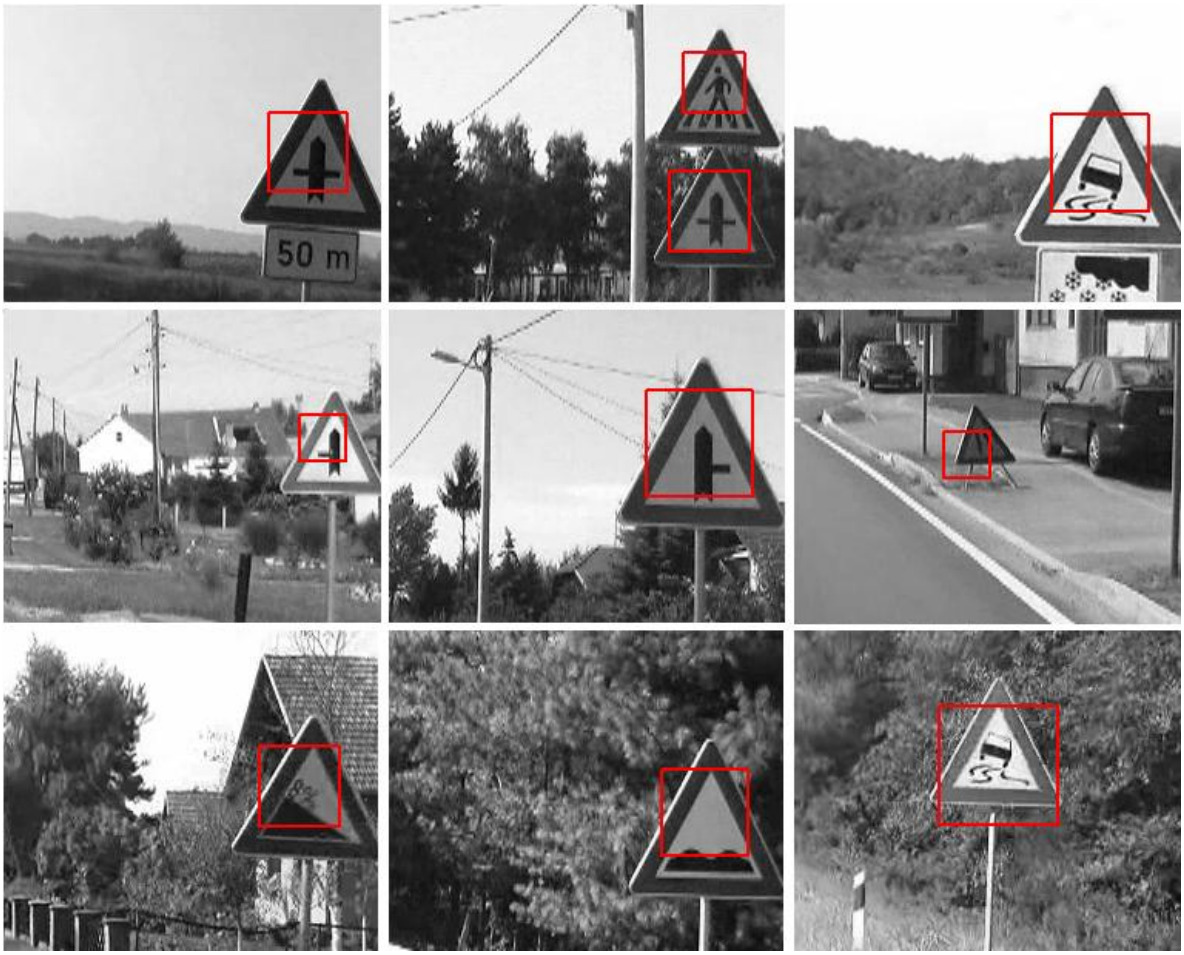


Slika 4.16 Detekcije prije grupiranja za primjer 4.15

4.4. Poteškoće pri detekciji

4.4.1 Lokalizacija detekcije

Ostvareni detektor trokutnih prometnih znakova ima visok odziv i vrlo dobru preciznost. Ono što je ponekad malo problematično jest lokalizacija detekcije. Iako se u većini slučajeva radi o dobroj lokalizaciji kakva je prikazana na slikama 4.3 - 4.6, ponekad se zna pojaviti loše lokalizirana detekcija. Najekstremniji primjeri loše lokalizacije prikazani su na slici 4.17.



Slika 4.17. Neprecizne pozitivne detekcije

Svi primjeri na slici 4.17, kao i oni što nisu prikazani svrstani su u pozitivne rezultate jer logički gledano ne radi se ni o lažno pozitivnim, ni o lažno negativnim detekcijama, ali za neku realnu primjenu bilo bi potrebno poboljšati lokalizaciju pomoću neke optimizacijske metode ili možda pomoću dodatne razine kaskade.

4.5 Statistika uspješnosti algoritma

Testiranje je obavljeno na 677 znakova. Kao što je rečeno u prethodnom poglavlju, naučene kaskade su namijenjene za detekciju minimalnom veličinom okna od 30 x 30 piksela.

Zašto je granica baš 30 piksela? Da je 25, moglo bi se postaviti pitanje zašto baš 25. U praksi bi granicu bilo potrebno odrediti ovisno o domeni primjene detektora – primjerice, za obradu u stvarnom vremenu je poželjno da se znak što prije uoči, dok je za obradu unaprijed prikupljenih slika možda dovoljno vršiti obradu samo nad slikama pribavljenim iz blizine. Odlučio sam da detektor bude namijenjen za znakove veličine između 30 x 30 piksela i 94 x 94 piksela (kao gornja granica je u algoritmu zadana vrijednost 90, ali zapravo se okno uvećava za 10% počevši od 30 sve dok ne prijeđe granicu, a to se dogodi kad naraste na konkretno 94 piksela). Slika 4.18 prikazuje ovisnost odziva algoritma o veličini okna.

Naučene su četiri kaskade čiji su rezultati dani u tablicama 4.1 - 4.4.

Prva kaskada naučena je s parametrima Cannyjevog detektora rubova σ=2.0, gornji prag=26.0, donji prag=2.0.

Pozitivni rezultati	661
Lažno pozitivni rezultati	74
Lažno negativni rezultati	17
Preciznost	89.93%
Odziv	97.49%

Tablica 4.1. Statistika rezultata testiranja za kaskadu s parametrima praga (26, 2).

Slika 4.18. Ovisnost odziva o veličini okna

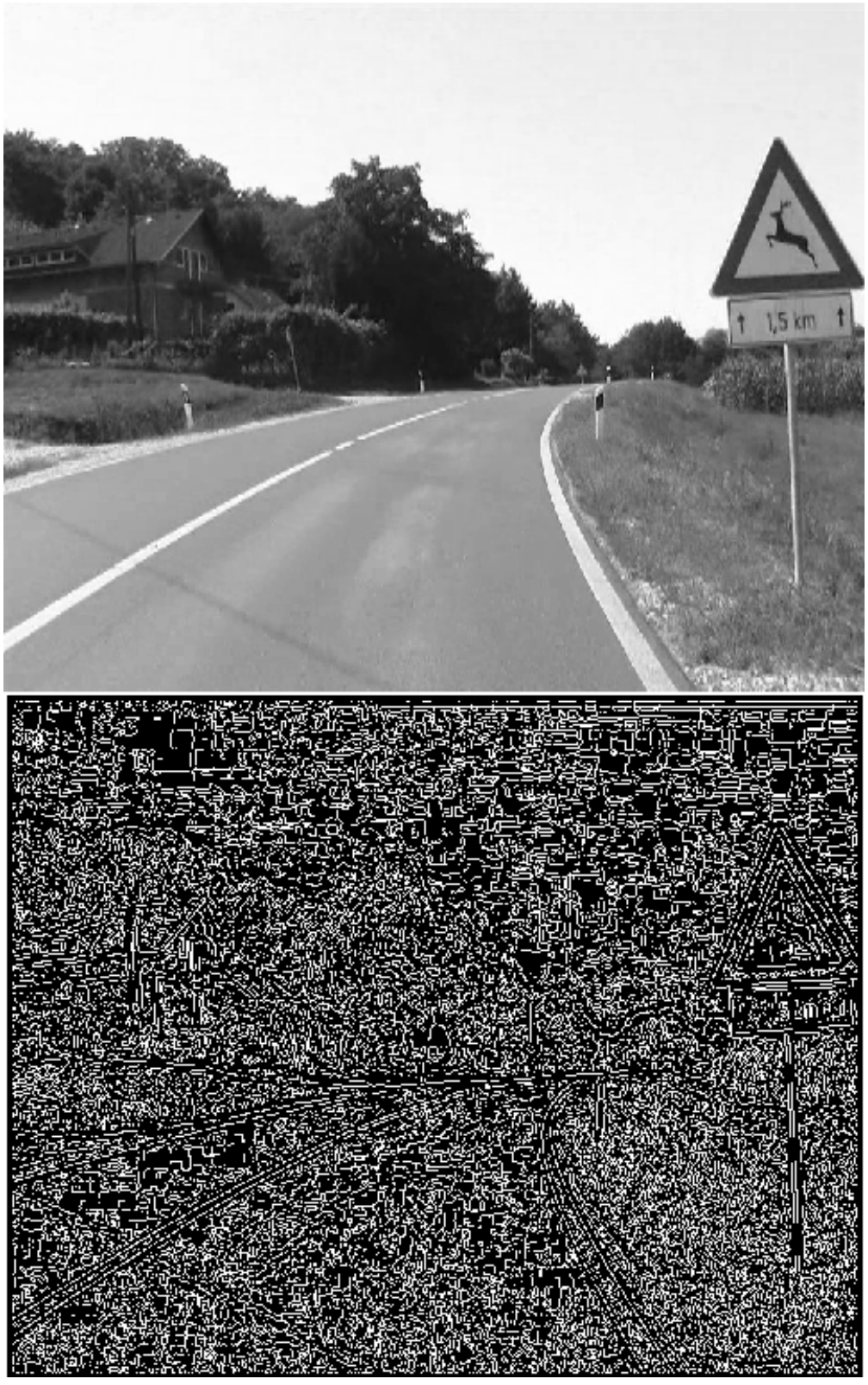
Lažno negativne detekcije su važnije od lažno pozitivnih te je stoga moguća poboljšanja logično tražiti prvenstveno u smjeru smanjivanja lažno negativnih rezultata. Promatrajući dobivene lažno negativne rezultate, lako je primijetiti da bi se određen dio njih eliminirao „popravljanjem“ Cannyjevog detektora rubova koji povremeno daje neispravan izlaz. Drugi parametri detektora rubova bi možda bili dobri za dane primjere, ali bi loše utjecali na neke druge. Međutim, što ako bi se parametri detektora rubova sveli na minimum, tj. ako bi se izbacili gornji i donji prag pri čemu bi ostao samo parametar σ (sigma)?

Kao odgovor na navedeno pitanje naučena je **druga kaskada** koja ima parametre gornjeg i donjeg praga u Cannyjevom algoritmu jednake nuli. To znači da se propušta maksimalan broj rubnih elemenata. Statistika za ovu kaskadu je dana u tablici 4.2.

Pozitivni rezultati	673
Lažno pozitivni rezultati	151
Lažno negativni rezultati	5
Preciznost	81.67%
Odziv	99.26%

Tablica 4.2. Statistika rezultata testiranja za kaskadu s parametrima praga (0, 0).

Propuštanje većeg broja rubnih elemenata za posljedicu ima smanjenje broja lažno negativnih rezultata jer nema problema s rubnim elementima koji nedostaju. S druge strane, veliki broj rubova u slici rezultira i većim brojem lažno pozitivnih rezultata jer u slici jednostavno nema „praznog prostora“ pa je veća vjerojatnost da će se negdje dogoditi razmještaj rubnih elemenata koji klasifikatori prepoznaju kao znak. Primjer slike rubova za parametre pragova jednake nuli prikazan je na slici 4.20.



Slika 4.20. Primjer obrade Cannyjevim algoritmom s parametrima praga (0,0)

Treća kaskada naučena je za rad nad slikama stanjenog gradijenta, što je izlaz treće faze Cannyjevog algoritma (vidi poglavlje Cannyjev algoritam). To znači da nema izvlačenja rubnih elemenata iz slike pa se integralna slika izrađuje izravno nad vrijednostima intenziteta piksela. Naravno, informacija o kutu gradijenta je još uvijek prisutna, bez nje ne bi bio moguće razlučiti koji piksel pridonosi kojoj integralnoj slici. Statistika za navedenu kaskadu dana je u tablici 4.3.

Pozitivni rezultati	666
Lažno pozitivni rezultati	132
Lažno negativni rezultati	12
Preciznost	83.45%
Odziv	98.23%

Tablica 4.3. Statistika rezultata testiranja za kaskadu s parametrima praga (0, 0).

Dok je kaskada naučena nad Cannyjevim algoritmom s apsolutnim pragovima (26, 2) imala najveću preciznost, a kaskada naučena nad Cannyjevim algoritmom s apsolutnim pragovima (0, 0) najveći odziv, kaskada naučena nad stanjenim gradijentom nalazi se negdje između jer ni preciznost ni odziv nisu najbolji, ali niti najgori u usporedbi s ostalim dvjema kaskadama.

Slika 4.21. Slika stanjenih gradijenata

Konačno, **četvrta kaskada** je ekvivalentna prvoj, ali s dvostruko manjim brojem pozitivnih uzoraka u skupu za učenje. Rezultati su vrlo bliski, ali kaskada koja je naučena na manjem broju pozitivnih uzoraka ima jednu razinu više. Razlog je jasan – manji broj primjera za učenje rezultira manjom preciznošću za isti broj razina, a algoritam obavlja učenje dok ne iscrpi sve lažno pozitivne uzorke u skupu slika pozadina pa je bila potrebna jedna razina više.

Pozitivni rezultati	663
Lažno pozitivni rezultati	87
Lažno negativni rezultati	15
Preciznost	88.40%
Odziv	97.78%

Tablica 4.4. Statistika rezultata testiranja za kaskadu s istim parametrima kao u tablici 4.1, ali uz dvostruko manje pozitivnih primjera u skupu za učenje

5. Zaključak

Razvijeni algoritam za detekciju prometnih znakova pokazao se iznimno robustan u usporedbi s nekim drugim radovima [3][6], ali i nešto manje precizan pri lokalizaciji samog znaka. U usporedbi sa sličnim radovima [1], korištenje informacije o kutu gradijenta smanjuje broj potrebnih razina kaskade.

U ostvarenoj implementaciji detektora objekata ima mjesta za napredak. Lokalizaciju prometnog znaka bi se moglo poboljšati pomoću neke od metoda optimizacije, slično kao što je obavljeno u jednom od prethodnih radova [6] ili naučiti dodatnu razinu kaskade (stroj s potpornim vektorima ili možda umjetna neuronska mreža) koja bi među primljenim detekcijama odabrala najbolju. Nadalje, pri izradi integralne slike bilo bi zanimljivo vidjeti kakvi bi bili rezultati da pojedini piksel može svojim iznosom pridonositi više integralnih slika, a ne samo jednoj, pri čemu bi većim dijelom svoje vrijednosti pridonosio slici za onaj kut gradijenta kojem je bliži (primjerice, piksel s kutom gradijenta 80° više bi pridonosio integralnoj slici za kut 90° nego onoj za 45°). Konačno, Viola i Jones su u svom radu [1] opisali pomicanje pragova pri određivanju optimalnog praga kako bi se smanjila vjerojatnost lažno negativnih rezultata uz neizbježno povećanje broja lažno pozitivnih rezultata. Opisano pomicanje pragova nije implementirano jer nije jasno kako ono utječe na matematički dokazana svojstva algoritma adaBoost [1] te je ostavljeno kao zanimljiv eksperiment za neki budući rad.

6. Literatura

[1] Viola, P., Jones, M. J.: *Robust Real-Time Face Detection*, International Journal of Computer Vision 57(2), 137–154, 2004

[2] Bonači, I., Kovaček, I., Kusalić, I.: *Detekcija i raspoznavanje prometnih znakova u videosnimci*, Fakultet elektrotehnike i računarstva, Zagreb 2009

[3] Dostal, D., Louč, S.: *Detekcija prometnih znakova pomoću integralnih slika*, Fakultet elektrotehnike i računarstva, Zagreb 2010

[4] Bašić, Š., Čepo, P. Š., Dodolović, I., Dostal, D., Grbić, S., Gulić, M., Horvatin, I., Louč, S., Sučić, I. *Cannyjev detektor rubova*, tehnička dokumentacija, Fakultet elektrotehnike i računarstva, Zagreb 2008

[5] Canny, J. *A Computational Approach To Edge Detection'*, 1986., PAMI, 8. svezak, 6. izdanje

[6] Louč, S. : Pronalaženje granica objekata ulančavanjem rubnih elemenata, Završni rad, Fakultet elektrotehnike i računarstva, Zagreb 2009

[7] Graczyk M., Lasota T., Trawiński B., Trawiński K.: *Comparison of Bagging, Boosting and Stacking Ensembles Applied to Real Estate Appraisal*, ACIIDS (2), 2010, pp.340-350.

[8] Rojas, R: *AdaBoost and the Super Bowl of Classifiers - A Tutorial Introduction to Adaptive Boosting*, Freie Universiat at Berlin, 2009

[9] Freund, Y., Schapire, R. E.: *A decision-theoretic generalization of on-line learning and an application to boosting*. Journal of Computer and System Sciences, 55(1):119–139, 1997.

Detekcija objekata naučenim značajkama histograma orijentacije gradijenta

Sažetak

Opisuje se postupak detekcije trokutnih prometnih znakova u slici. Kutevi gradijenta (0°-360°) se podijele u odjeljke te se izrađuje po jedna integralna slika za svaki odjeljak. Pomoću dobivenih integralnih slika te skupa pozitivnih i negativnih primjera za učenje izrađuje se kaskada klasifikatora metodom adaBoost. Dobivena kaskada se koristi za detekciju trokutnih prometnih znakova u slikama iz skupa za testiranje. Opisana je razvijena programska implementacija u programskom jeziku C++. Konačno, prikazani su i komentirani dobiveni eksperimentalni rezultati primjene algoritma na stvarnim slikama.

Ključne riječi

Računalni vid, kut gradijenta, integralna slika, adaBoost, kaskada klasifikatora, detekcija objekata.

Object detection using learned features of histogram of oriented gradients

Summary

We describe a procedure for triangular traffic sign detection. Gradient angles (0°-360°) are divided into bins and a set of integral images (one for each bin) is constructed. Using the integral images and sets of positive and negative learning examples a cascade of classifiers using adaBoost method is constructed. Constructed cascade is used for detecting triangular traffic signs in a set of testing images. The developed software implementation in C++ is described. Finally, experimental results of applying the algorithm on real images are provided and discussed.

Keywords

Computer vision, gradient angle, integral image, adaBoost, cascade of classifiers, object detection.