SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 206

Detekcija prometnih znakova u video sekvencama

Marko Pranjić

Zagreb, lipanj 2008.

Postoje ljudi čija imena nisu navedena na naslovnoj stranici, a bez kojih ne bi bilo ni ovoga rada. Ovim putem im se zahvaljujem, posebno svom mentoru prof. dr. sc. Zoranu Kalafatiću na svim savjetima koje mi je pružio prilikom izrade ovog rada.

Sadržaj

1. Uvod

Znanstvenici, a pogotovo pisci znanstvene fantastike bili su oduševljeni mogućnošću izgradnje inteligentnih strojeva, a razumijevanje vidljivog svijeta je jedan od preduvjeta koji neki traže od takvih strojeva. Ljudski mozak lako rješava vizualne probleme ali mi imamo vrlo malo znanja o kognitivnim procesima koji se odvijaju za to vrijeme. Allan Turing, jedan od očeva modernih računala i umjetne inteligencije, vjerovao je da bi inteligentno računalo moralo moći vidjeti i razumjeti ono što vidi.

Jedno od fascinantnijih područja računarstva, računalni vid, počeo je s razvojem tek u novije vrijeme (zadnjih tridesetak godina) i još je vrlo mlada znanstvena disciplina. Zadatak računalnog vida je izgradnja umjetnog sustava za prikupljanje informacija sa slika i donošenje odluka na temelju tih informacija. Na nekoliko načina možemo računalni sustav naučiti kako donositi zaključke iz ulaznog skupa slika. Tu spadaju:

1. Metode temeljene na izgledu
2. Metode temeljene na znanju
3. Podržano učenje (eng. *reinforced learning*)

2. Računala koja vide?

Postoji nekoliko metoda kojima možemo računalni sustav naučiti kako gledati i donositi zaključke na temelju slika.

2.1. Metode temeljene na izgledu

Kod ovih metoda, sustav sam uči donositi odluke na temelju pripremljenih primjera.

2.1.1. Nadzirano učenje

Karakteristika nadziranog učenja su podaci u obliku (**x,y**), tj. ulazna vrijednost **x** i ciljna vrijednost **y**. Ciljne vrijednosti su klase, odnosno razredi u koje želimo grupirati ulazne podatke ali u općem slučaju ciljna vrijednost je broj iz skupa realnih brojeva.

Metodu nadziranog učenja možemo podijeliti na:

a) **Regresija** – učenje funkcija

   Izlaz metode regresije je funkcija koju pokušavamo aproksimirati. Možemo koristiti sljedeće prisupe:

   - unaprijedne (eng. *feedforward*) neuronske mreže
   - regresijska analiza

b) **Raspoznavanje uzoraka** – klasifikacija

   Izlaz je jedna ili više klasa kojima pokušavamo podijeliti ulazne primjere. Primjer ove metode su detektori, od kojih jedan pokušavam izgraditi u ovom radu. Neki od pristupa koji nam stojena raspolaganju su:

   - unaprijedne (eng. *feedforward*) neuronske mreže
   - k-NN klasifikatori (eng. *nearest neighbourgh classifiers*)
   - stabla odluke (eng. *decision tree*)
   - diskriminantna analiza

2.1.2. Nenadzirano učenje

Razlika između ove metode i raspoznavanja uzoraka je u tome što ovom metodom računalo pokušava samo svrstati ulazne vrijednosti u klase, grupirati uzorke, dok raspoznavanjem uzoraka unaprijed uz ulaznu vrijednost dajemo i klasu u koju je treba svrstati. Pristupi korišteni kod ove metode su:

- samoorganizirajuće neuronske mreže (eng *self-organizing map*)
- grupiranje (eng. cluster analysis)

2.2. Metode temeljene na znanju

Ove metode se temelje na pravilima koje opisuju izgled predloška te koje su mu karakteristike. Obično takva pravila opisuju odnose između dijelova slike.

2.3 Podržano učenje (eng. *reinforced learning*)

Ovom metodom računalni program uči obavljati zadatak isključivo temeljem nagrade i kazne. Algoritam obavlja zadatak bez da mu kažemo kako ga obaviti. Uči se postupkom pokušaja i pogreške u kojoj program interakcijom s dinamičkom okolinom donosi zaključke. Obilno se koriste heurističke metode kao što su evolucijski algoritmi.

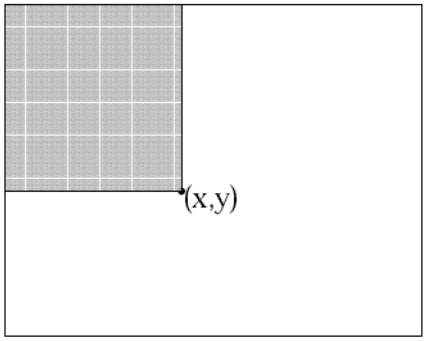
U ovom radu pokušao sam korištenjem *open source* biblioteke za rad s računalnim vidom (*OpenCV*) ostvariti automatsko prepoznavanje prometnih znakova na video sekvenci prometnih situacija iz stvarnog života korištenjem strojnog učenja za generiranje klasifikatora. Kao metodu prepoznavanja koristio sam slabe Haarove značajke koje se u jaki klasifikator spajaju AdaBoost algoritmom.

3. Klasifikacija temeljena na značajkama

Zadatak prepoznavanja objekata na slici je računalno složen zadatak. Međutim C. Papageorgiou u [1] predlaže korištenje alternativnog skupa značajki umjesto do tada korištenog intenziteta (RGB vrijednost svakog pojedinog piksela). Danas takav skup značajki znamo pod imenom Haarove značajke (*Haar-like features*)[1]. Novopredloženi skup značajki razmatra pravokutne dijelove slike i sumira vrijednosti piksela u njima. Suma se koristi za kategorizaciju slike. Ideja pristupa je da će slika šume imati drugačiju sumu piksela od slike neba i zgrada. No razlučivost ovog pristupa daje vrlo dobre rezultate i u puno manjim razlikama. Najpoznatiji i najčešći primjer korištenja Haarovih značajki je detekcija lica.

3.1. Integralna slika

Viola i P. Jones 2001. godine izdaju rad [2] kojim uvode poboljšanja za detekciju Haarovim značajkama. Pravokutne značajke mogu se izračunati vrlo brzo korištenjem posrednog prikaza slike kojeg su nazvali integralna slika.



Slika 1: Integralna slika (Viola, 2001.)

1 Alfred Haar, mađarski matematičar, poznat po Haarovoj transformaciji na kojoj se temelji postupak potpojasnog kodiranja Haarovim valićima. Haarove značajke nazvane su tako jer podsjećaju na spomenute Haarove valiće.

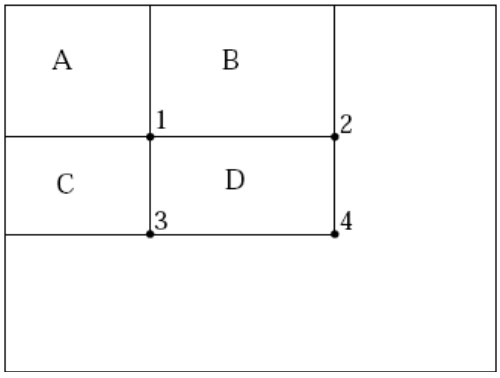
$$ii(x,y)=\sum_{x'\le x,\,y'\le y} i(x',y')$$

Integralna slika na poziciji *(x,y)* sadržava vrijednost sume svih piksela koji se nalaze iznad i lijevo od te pozicije.

Koristeći rekurzivne formule:

$$s(x,y)=s(x,y-1)+i(x,y)$$
$$ii(x,y)=ii(x-1,y)+s(x,y)$$
$$s(x,-1)=0,\ ii(-1,x)=0$$

Uz $s(x,y)$ koji predstavlja kumulativnu sumu po retcima, vrijednost integralne slike može se izračunati u jednom prolazu kroz sliku. Koristeći takvu sliku vrijednost sume bilo kojeg pravokutnika originalne slike može se lako izračunati korištenjem vrijednosti sume u vrhovima pravokutnika.



Slika 2: Vrijednost pravokutnika D dobivamo sa: D=4-3-2+1 (Viola, 2001.)

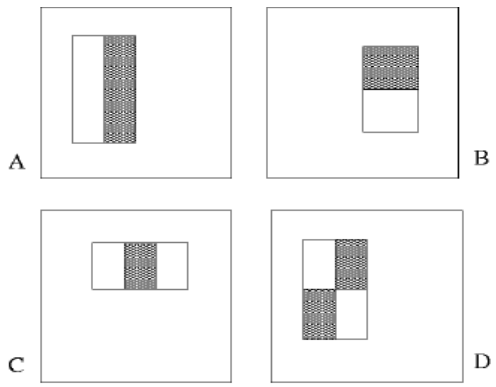
3.2. Klasifikatori

U matematici klasifikatori su preslikavanja prostora značajki u skup simbola. Nihov zadatak je svrstati objekte koji imaju slične zajedničke vrijednosti značajki u iste klase. Linearni klasifikatori to postižu odlukom o klasifikaciji koja se temelji na linearnoj kombinaciji vrijednosti značajki.

$$y = f(\vec{x} \cdot \vec{w}) = f(\sum_j x_j w_j)$$

Gdje f predstavlja funkciju klasifikatora kojoj je argument skalarni produkt vektora težina $\vec{w}$ i ulaznog vektora $\vec{x}$. Vektor težine klasifikator odredi postupkom učenja na skupu primjera.

U originalnom radu [2], Viola i Jones definiraju četiri tipa različitih značajki. Značajke su jednostavne jer ih tako možemo puno brže proračunati korištenjem integralne slike. Suma piksela koji se nalaze unutar bijelog pravokutnika oduzimaju se od vrijednosti sume sivog pravokutnika.



Slika 3: Haarove značajke (Viola, 2001.)

Značajke s dva pravokutnika osjetljive su na promjenu intenziteta duž iste linije slike, pa detektiraju rubove objekata. Značajke s tri pravokutnika detektiraju linije, a s četiri pravokutnika dijagonale objekata.

U prozoru veličine 24×24 postoji oko 180 tisuća različitih pravokutnih značajki. Iako je svaku značajku moguće proračunati vrlo efikasno, izračunavanje svih značajki prilikom detekcije je neizvedivo za praktičnu primjenu. Zbog tako velikog skupa značajki Viola i Jones u [2] bilježe:

> "Naša hipoteza, vođena eksperimentom, je da je moguće koristiti vrlo mali broj značajki za uspješnu detekciju. Glavni izazov je odrediti te značajke."

Zadatak određivanja značajki dovoljnih za uspješnu detekciju riješili su korištenjem AdaBoost algoritma. On pokušava složiti kombinaciju značajki i pripadnih težina koje najbolje odvajaju skupove pozitivnih i negativnih primjera za učenje, odnosno one koje imaju najniži broj pogrešnih klasifikacija. Ovim postupkom znatno smanjujemo broj značajki, ali nam svejedno preostaje nekoliko stotina značajki koje bi morali izražunavati za svaku sliku i za svaki prozor te slike prilikom detekcije. Iako je napredak u odnosu na početnih 180 tisuća, niz od nekoliko stotina značajki je još uvijek prevelik za praktičnu primjenu. Rješenje ovog problema je korištenje kaskade klasifikatora koji izbacuju velike dijelove slike koristeći jednostavne klasifikatore koji odbacuju negativno klasificirane dijelove. Nakon njih koriste se sve složeniji klasifikatori s više značajki i postiže se niska stopa lažne detekcije.

3.3. AdaBoost algoritam

AdaBoost ili Adaptive Boost je jedan od postupaka korištenih za poboljšanje bilo kojeg algoritma računalnog učenja a prvi puta je opisan u [3]. Ti postupci uspješno kombiniraju nekoliko grubih i umjereno uspješnih pravila u vrlo uspješna pravila.

AdaBoost algoritam poziva osnovni postupak kojim nalazi slabe klasifikatore, odnosno značajke i pripadajuće pragove. Zadatak slabog klasifikatora je dati hipotezu h_t koja ulazne podatke klasificira u dvije skupine.

$$h_t : X \rightarrow \{-1, +1\}$$

Ulazni podaci imaju određene težine w_i koje se tijekom učenja mijenjaju, a na početku su jednake za sve primjere.

$$w_i = \frac{1}{i}$$

Dobrota nekog slabog klasifikatora određena je njegovom pogreškom ϵ_t.

$$\epsilon_t = \sum_i w_i \left[h_t(x_i) \neq y_i \right]$$

Konačno, jaki klasifikatora dobiva se linearnom kombinacijom slabih klasifikatora.

$$\sum_{t=1}^{T} \alpha_t h_t(x)$$

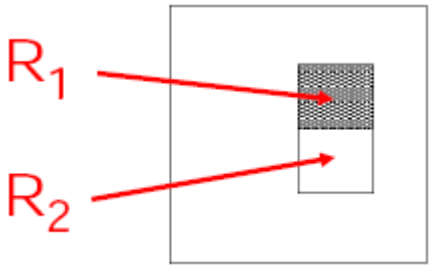
Gdje su slabi klasifikator h_t i jaki klasifikator H predstavljeni sa:

- slabi klasifikator $h_t(\vec{x}) \in \{-1, 1\}$, za vektor podataka $\vec{x}$
- jaki klasifikator $H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$

$$H(x) = sign(\alpha_1 h_1(x) + \alpha_2 h_2(x) + \alpha_3 h_3(x) + \ldots + \alpha_n h_n(x))$$

Konačna klasifikacija nije obična suma glasova jer neki slabi klasifikatori bolje aproksimiraju rješenje od drugih. Stoga se vrijednosti težina pojedinih glasova, moraju dodatno proračunati.

Za primjer značajke definirajmo polja R1 i R2



Slika 4: (Zisserman, 2004.)

Tada je vrijednost slabog klasifikatora $h(x)$ određena pomoću:

$$h(x)=\begin{cases} 1 & \text{ako je } f(I)>\theta \\ -1 & \text{inače} \end{cases}$$

Gdje je θ prag klasifikatora, a $f(I)$ razlika integralnih suma R2 i R1:

$$f(I)=\int_{R2} I(x,y)\,dx\,dy-\int_{R1} I(x,y)\,dx\,dy$$

Algoritam se izvodi u iteracijama *(1..t)* i svaku iteraciju možemo logički podijeliti u dva dijela. Prvi dio služi za odabir slabih klasifikatora, odnosno značajki i pridruženih pragova, a drugi procjenjuje pogrešku i ponovno evaluira težine.

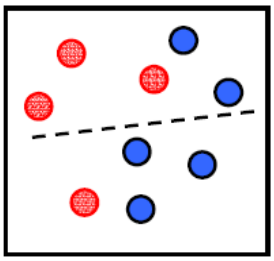
1. Odabir slabih klasifikatora

   Na primjeru slike X_i i klasifikacije $Y_i=\{-1,1\}$ evaluira se pogreška svake značajke ϵ_j i odabire najbolji prag[2] kako bi se smanjila pogreška detekcije.

   $$\epsilon_j=\sum_i \omega_i\left[h_j(x_i)\neq y_i\right] \text{ gdje su } \omega_i \text{ težine}$$

2 U originalnom algoritmu prag se bira tako da se minimizira broj krivo klasificiranih u oba skupa. Viola i Jones su u [2] modificirali algoritam za odabir praga tako da se minimizira samo broj krivo klasificiranih pozitivnih znakova. Modifikacija je uvedena radi kasnijeg slaganja klasifikatora u kaskadu. Nije provjereno jesu li uz ovakvo odstupanje od formalnog algoritma očuvana dobra teoretska svojstva AdaBoost algoritma.

Najbolji klasifikator, tj. značajka i pridruženi prag, onaj je koji daje detekciju s najmanjom pogreškom.



Slika 5

2. Procjena pogreške i izmjena težina

Pogreška klasifikatora određena je sa:

$$\epsilon_t = \sum_i w_i \left[h_t(x_i) \neq y_i \right]$$

Korekcija težine računa se pomoću:

$$\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$$

Nakon čega se mijenjaju težine pogrešno klasificiranim uzorcima.

$$w_{t+1,i} = w_{t,i} e^{-\alpha_t y_i h_t(x_i)}$$

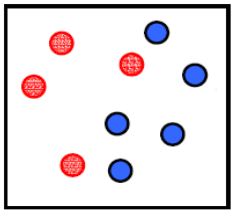
S novim težinama algoritam se vraća na prvi korak i ponavlja dok nismo zadovoljni dobivenim jakim klasifikatorom.

Konačni jaki klasifikator je oblika:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

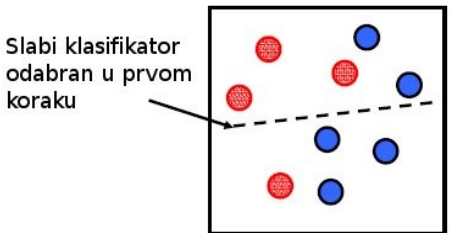
3.3.1. Primjer izvođenja AdaBoost algoritma.

Na početku svi primjeri iz skupa za učenje imaju jednake težine:



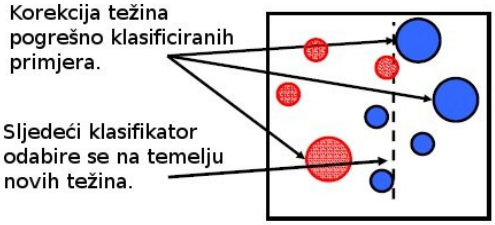
Slika 6

Odabire se klasifikator s najmanjom pogreškom.

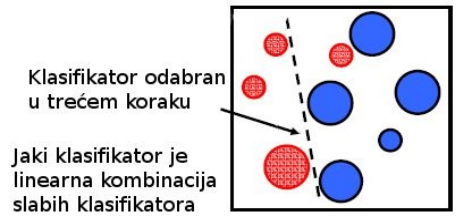


Slika 7

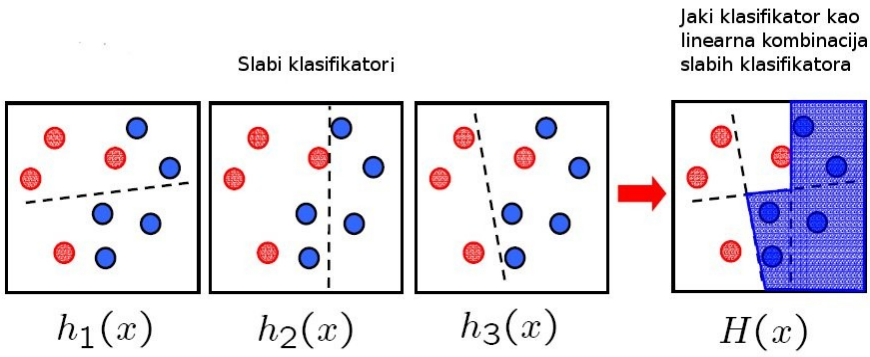
Mijenjaju se težine i odabiru se novi klasifikatori s najmanjim pogreškama.



Slika 8



Slika 9

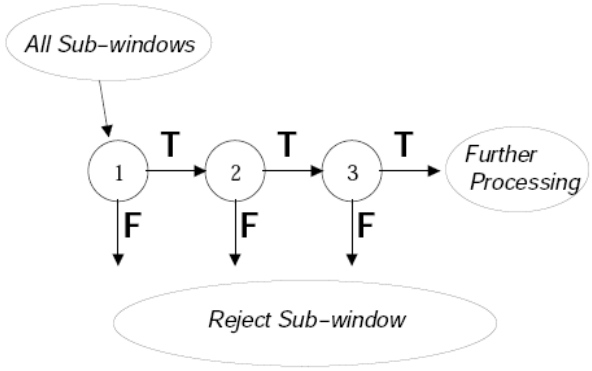


Slika 10: (Zisserman, 2004.)

4. Kaskade klasifikatora

Jaki klasifikator koji bi postizao dobre rezultate morao bi biti izgrađen od velikog broja slabih klasifikatora. Dodavanjem slabih klasifikatora izravno povećavamo broj značajki koje treba izračunati prilikom detekcije odnosno algoritamsku složenost detekcije koja se očituje u većem vremenu izračunavanja. Ovaj problem možemo zaobići korištenjem kaskade jakih klasifikatora.

Kaskada klasifikatora je stablo odlučivanja kod kojeg se tek nakon pozitivnog rezultata u nekom stupnju kaskade uzorak evaluira u sljedećem stupnju kaskade. Pozitivan rezultat prvog stupnja kaskade uzrokuje evaluaciju uzorka nad drugim stupnjem. Pozitivan rezultat drugog stupnja uzrokuje evaluaciju trećeg, itd. Negativan rezultat u bilo kojem stupnju kaskade rezultira odbacivanjem uzorka tako da se negativni uzorci odbacuju već na nižim stupnjevima kaskade, no uzorak koji će se pozitivno evaluirati mora proći sve stupnjeve kaskade.



Slika 11: Kaskada klasifikatora (Viola, 2001.)

Stopu netočne detekcije čitave kaskade možemo izraziti kao umnožak netočnih detekcija svih stupnjeva, odnosno:

$$F=\prod_{i=1}^{K} f_i$$

dok je stopa pozitivne detekcije:

$$D=\prod_{i=1}^{K} d_i$$

gdje je K broj stupnjeva kaskade, f_i stopa netočne detekcije a d_i stopa pozitivne detekcije pojedinog stupnja kaskade.

Kaskada se gradi u skladu sa željenim budućim performansama koje možemo izračunati iz gore navedenih izraza. Na primjer, stopu detekcije od 0.9 moguće je postići kaskadom od 10 stupnjeva u kojoj svaki stupanj ima stopu detekcije 0.99.

$$D=\prod_{i=1}^{10} 0.99=0.99^{10}\approx 0.9$$

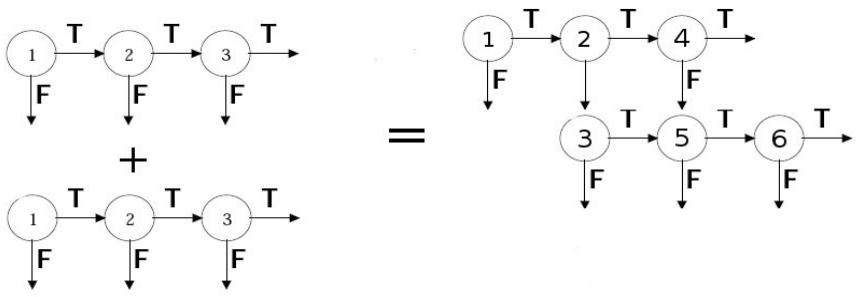
Stopa netočne detekcije kaskade uz stopu detekcije pojedinog stupnja od čak 0.5 iznosila bi:

$$F=\prod_{i=1}^{10} 0.5=0.5^{10}\approx 10^{-3}$$

Postojeći sustavi postižu jako dobre stope detekcije (>85%), zadržavajući nisku stopu pogrešne detekcije (reda veličine 10^{-6}). Broj stupnjeva kaskade kao i rezultati pojedinog stupnja moraju biti pažljivo odabrani kako bi se postigli slični rezultati. Više stupnjeva u kaskadi rezultiralo bi nižom stopom detekcije, dok bi smanjenjem stupnjeva kaskade dobili veću stopu pogrešne detekcije.

Ako postoji velika varijabilnost unutar klase koje je moguće podijeliti na podklase jednostavna kaskada nije više efikasna. Lienhart, Liang i Kuranov u [7] predlažu stablo kao efikasniju strukturu od jednostavne kaskade. Naivno rješenje je podijeliti uzorke u skupine i izgraditi kaskadu klasifikatora za svaku skupinu. Pri detekciji, ako dani uzorak uspješno prolazi bilo koju od kaskada, proglašava se objektom. U OpenCV implementaciju uključena je izgradnja strukture stabla. Ona spaja početne stupnjeve posebnih kaskada i stvara grane samo u slučaju ako je to povoljno za konačnu računsku složenost.

Svaki stupanj kaskade za učenje koristi skup primjera koji mu prosljeđuje roditelj. Rezultat je jaki klasifikator čija računska složenost je linearno ovisna o broju slabih klasifikatora.



Slika 12: Stablo klasifikatora

4.1. Izgradnja kaskade klasifikatora korištenjem biblioteke OpenCV

OpenCV je biblioteka funkcija otvorenog koda razvijena u Intelu. U njoj možemo naći mnoge algoritme i prigodne strukture podataka često korištene na području računalnog vida. Među ostalim, sadržava i funkcije za izgradnju i korištenje kaskade klasifikatora izgrađenih na Haarovim značajkama. Iako je namijenjena korištenju u programskom jeziku C++ postoje različite biblioteke i moduli kao podrška za ostale jezike.

U ovom radu pokušao sam implementirati prepoznavanje prometnih znakova u programskom jeziku Python[3] pod operacijskim sustavom Linux[4]. Sama instalacija je vrlo jednostavna jer možemo sve potrebne pakete za instalaciju dohvatiti iz repozitorija programa na Internetu. Jedini preduvjet korištenju funkcija za rad s video datotekama je instaliran i ispravno konfiguriran program FFmpeg.

Instalacija OpenCV-a uključuje module za Python i potrebnu dokumentaciju, no upozorava se da projekt uklapanja OpenCV-a u Python još nije završen što se očituje u smanjenom skupu funkcija koje su nam na raspolaganju.

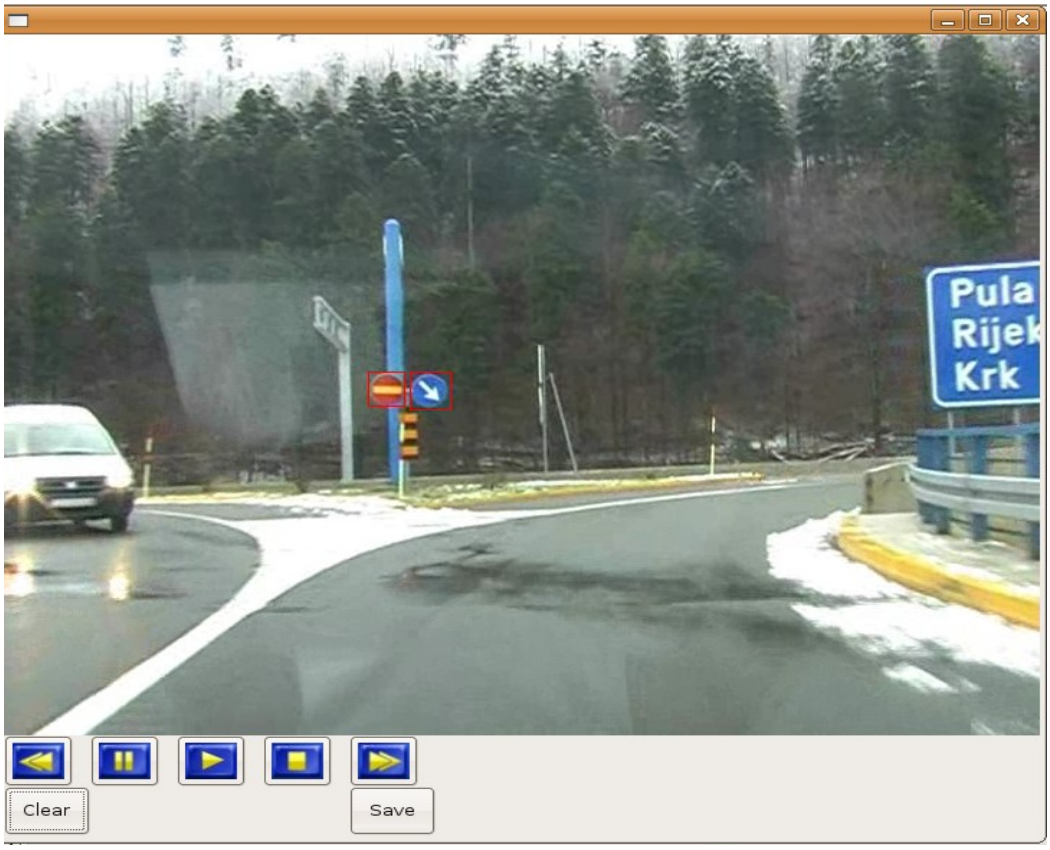
3 Python v. 2.5.1

4 Linux, Ubuntu 7.10, kernel 2.6.22-14, i686

4.1.1. Priprema primjera za učenje

Prvi korak u izgradnji kaskade klasifikatora je priprema skupa pozitivnih i negativnih primjera. Pozitivni primjer je primjer slike objekta kojeg želimo naučiti detektirati i u našem slučaju to su slike prometnih znakova dobivenih kamerom iz stvarnih prometnih situacija. Negativni primjeri su slike pozadine i na njima ne smije biti pozitivnih primjera. Za negativne primjere koristio sam slike ostalih prometnih scena koje ne sadrže prometne znakove.

Objekti koje želimo naučiti detektirati moraju biti označeni. Za potrebe označavanja napisao sam skriptu u Pythonu kojom možemo pregledavati video snimku te označiti i snimiti pozitivne i negativne primjere koje uočimo.



Slika 13: Primjer označavanja pozitivnih primjera

Ako smo zadovoljni označenim možemo pohraniti isječke u direktorij s pozitivnim primjerima i kasnije ih koristiti za treniranje kaskade[5].

Primjer isječaka s prošle snimke:

Nakon pohranjivanja skripta bilježi naziv pod kojim je isječak pohranjen te njegove dimenzije u posebnu datoteku.

Primjer zapisa:

```
frame-659-0.bmp 2 12 10 23 23 45 38 20 20
frame-659-1.bmp 1 0 0 21 21
```

Na prvom mjestu je naziv datoteke u koju je pohranjen isječak (*frame-659-0.bmp*), na drugom je broj pozitivnih elemenata (*2*) iza čega za svaki objekt slijede pozicija gornjeg lijevog kuta (*12,10*) te dimenzije objekta[6] (*23,23*).

Ako nismo ništa označili na snimci, program tretira čitavu sliku kao pozadinu i pohranjuje ju. Slično kao i za pozitivne primjere, pohranjujemo informaciju o nazivu datoteke u posebnu datoteku, no kako se čitava slika tretira kao negativan primjer, nije potrebno pohranjivati informacije o dimenzijama slike. Negativni primjeri su isječci tih slika i stvaraju se prilikom pokretanja AdaBoost algoritma.

Za treniranje kaskade OpenCV koristi slike pohranjene u posebnom formatu.

---

5 Za treniranje kaskade se koriste sive slike spremljene u posebnom formatu. Ove isječke koristimo kako bi stvorili datoteku iz koje klasifikator može učiti.

6 Format datoteke je prilagođen označavanju objekata na slikama. U ovom radu sam radio s video sekvencom i snimanje čitave slike svaki puta kada bih naišao na znak činilo mi se je nepraktično. Odlučio sam snimati samo isječke, tako da je broj pozitivnih elemenata (prometni znakovi) za svaki isječak jednak jedan. Gornji lijevi kut objekta je u (0,0) a visina i širina su dimenzije čitave slike (isječka).

Sve slike su povezane u istoj datoteci i sve su jednakih dimenzija. Ovaj format slika možemo dobiti programom *opencv-createsamples* uključenim u biblioteku funkcija. Program čita datoteku u koju smo pohranili informacije o pozitivnim primjerima i na temelju toga izgrađuje datoteku pogodnu za treniranje kaskade.

Neki važniji argumenti koje možemo proslijediti programu *opencv-createsamples* su:

`-info <naziv_datoteke>`

Naziv datoteke za parametar *info* dajemo datoteku u kojoj smo bilježili nazive i dimenzije pozitivnih primjera.

`-vec <naziv_datoteke>`

Ovaj parametar govori programu kao treba nazvati izlaznu datoteku.

`-show`

je zadan ovaj argument, primjeri za učenje prikazuju se na ekranu. Pomoću takvog prikaza lako je uočiti pogreške u označavanju.

`-num <broj_uzoraka>`

Broj pozitivnih primjera koje smo označili. ne zadamo broj primjera, program pretpostavlja 1000 uzoraka.

`-w <širina_uzoraka>`

Širina na koju će se skalirati uzorci, izražena u pikselima.

`-h <visina_uzoraka>`

Visina na koju će se skalirati uzorci, izražena u pikselima. ne zadamo željenu širinu i visinu uzoraka pretpostavljene vrijednosti su 24 piksela.

Potpun popis argumenata možemo dobiti pokretanjem programa bez argumenata, a njihova objašnjenja u korisničkoj dokumentaciji programskog paketa OpenCV.

Slika 14: Primjer negativne slike

Slika 15: Primjer negativne slike

4.2. Treniranje kaskade

Treniranje kaskade klasifikatora izvodi se još jednim programom uključenim u programski paket OpenCV. Riječ je o programu *opencv-haartraining* koji uz pomoć pozitivnih primjera generiranih pomoću *opencv-createsamples* i proizvoljnih negativnih primjera stvara kaskadu jakih klasifikatora kombinirajući ih AdaBoost algoritmom.

Neki važniji argumenti koje možemo proslijediti programu *opencv-createsamples* su:

`-data <naziv_kaskade>`

Argument programa koji određuje naziv buduće kaskade. Program sprema kaskadu kao niz direktorija označenih brojevima koji predstavljaju stupanj kaskade. Osim niza direktorija, kaskada se sprema i u formatu XML.

`-vec <naziv_datoteke>`

Naziv i put do datoteke stvorene *opencv-createsamples* programom u kojoj se nalaze pozitivni primjeri.

`-bg <naziv_datoteke>`

Parametar koji govori programu iz koje datoteke će čitati popis slika koje nam služe kao negativni primjeri.

`-npos <broj_primjera>`

Broj pozitivnih primjera stvorenih s programom *opencv-createsamples*.

`-nneg <broj_primjera>`

Broj negativnih primjera koje smo priredili za učenje.

`-nstages <broj_stupnjeva>`

Koliko stupnjeva kaskade želimo trenirati.

`-minhitrate <stopa_detekcije>`

Minimalna stopa detekcije za svaki stupanj kaskade.

`-maxfalsealarm <stopa_detekcije>`

Maksimalna dopuštena stopa lažnih detekcija za svaki stupanj.

Pokretanjem *opencv-haartraining* dobivamo niz ispisa sličnih ovome:

```
Tree Classifier
Stage
+---+---+---+---+---+---+---+---+---+---+---+---+
|  0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11|
+---+---+---+---+---+---+---+---+---+---+---+---+

   0---1---2---3---4---5---6---7---8---9--10--11


Parent node: 11

*** 1 cluster ***
POS: 223 228 0.978070
NEG: 223 0.00013485
BACKGROUND PROCESSING TIME: 29.00
Precalculation time: 1.00
+----+----+-+---------+---------+---------+---------+
|  N |%SMP|F|  ST.THR |    HR   |    FA   | EXP. ERR|
+----+----+-+---------+---------+---------+---------+
|   1|100%|-|-0.808000| 1.000000| 1.000000| 0.273543|
+----+----+-+---------+---------+---------+---------+
|   2|100%|+|-1.187073| 1.000000| 1.000000| 0.266816|
+----+----+-+---------+---------+---------+---------+
|   3|100%|-|-0.944304| 1.000000| 0.654709| 0.172646|
+----+----+-+---------+---------+---------+---------+
|   4| 91%|+|-1.138482| 1.000000| 0.730942| 0.192825|
+----+----+-+---------+---------+---------+---------+
|   5| 87%|-|-0.593820| 0.995516| 0.475336| 0.132287|
+----+----+-+---------+---------+---------+---------+
Stage training time: 41.00
Number of used features: 5

Parent node: 11
Chosen number of splits: 0

Total number of splits: 0
```

Iz njega možemo pročitati kako napreduje učenje naše kaskade.

```
Stage
+---+---+---+---+---+---+---+---+---+---+---+---+
|  0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11|
+---+---+---+---+---+---+---+---+---+---+---+---+
```

Ovaj ispis nam govori da se kaskada trenutno sastoji iz dvanaest stupnjeva, ispod čega je dan grafički prikaz stabla kaskade. U ovom slučaju to je jednostavno stablo bez grananja.

```
0---1---2---3---4---5---6---7---8---9--10--11
```

Nadalje, dobivamo informacije o trenutnoj kvaliteti kaskade klasifikatora.

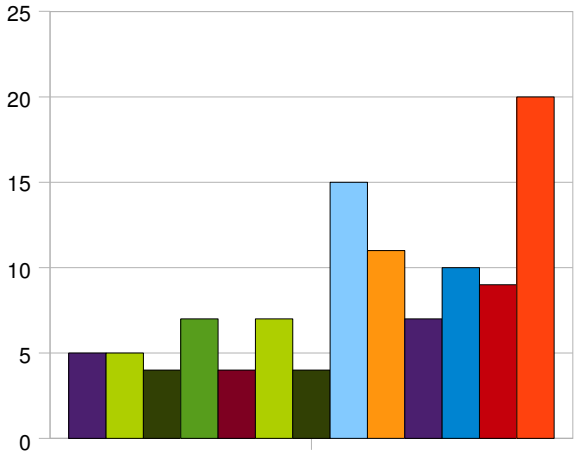
```
POS: 223 228 0.978070
NEG: 223 0.00013485
```

U našem slučaju kaskada prepoznaje 97.8% primjera iz skupa za učenje uz očekivano prihvaćanje pogrešnih detekcija do 0.0135%. Kako su skupovi za učenje ograničeni najčešće nemamo dovoljno primjera da bi ove mjere bile realne. Naš cilj je prikupiti što više primjera iz stvarnih situacija kako bi klasifikator što točnije mogao aproksimirati razliku između ta dva skupa te bio točniji u prepoznavanju.

Vrijednosti `BACKGROUND PROCESSING TIME` i `Precalculation time` izražene su u sekundama a označavaju vrijeme potrebno za prikupljanje negativnih primjera odnosno vrijeme potrebno za izračunavanje vrijednosti značajki.

Svaki redak tablice predstavlja jednu značajku. U stupcima su navedene informacije o rednom broju, postotku primjera za učenje na kojima se traže značajke, informacija o simetriji značajke, prag slabog klasifikatora izgrađenog od

te značajke stopu pozitivne i netočne pozitivne detekcije te postotak primjera koji bi bili pogrešno klasificirani uz prag stupnja jednak nuli.



Stvorena kaskada klasifikatora imala je četrnaest stupnjeva, a broj korištenih značajki na svakom stupnju učenja prikazan je grafom. Interpretacija grafa je povećanje složenosti korištenih klasifikatora s povećanjem stupnja kaskade. Ovaj graf potvrđuje komentar iz rada [2] Viole i Jonesa da bi se broj značajki kojima se odjeljuju pozitivni i negativni primjeri trebao povećavati sa stupnjem kaskade.

Naučena kaskada klasifikatora sprema se kao niz direktorija i kao datoteku u formatu XML. Prvi način sprema znanje o svakom stupnju u posebni direktorij čija imena označavaju stupanj kaskade. Tako imamo direktorije "*0*", "*1*", "*2*" pa sve do "*13*". U svakom direktoriju nalazi se datoteka s popisom korištenih klasifikatora, odnosno značajki i pridruženih pragova za taj stupanj. Datoteka u formatu XML je drugi način spremanja znanja o radu klasifikatora. Zapisi u direktorijima su ekvivalentni zapisu u XML datoteci, jedina prednost prvog zapisa je da program može jednostavno nastaviti s učenjem ako je postojao prekid u prijašnjem izvođenju, no i XML zapis ima svojih prednosti – veća čitljivost.

4.3. Primjer zapisa značajki i njihovo tumačenje

```
1
2
1 10 7 12 0 -1
1 14 7 4 0 3
haar_y3
1.565163e-03 0 -1
-4.535058e-01 4.116098e-0
```

Nakon broja čvorova (`1`) i broja pravokutnika (`2`) od kojih se sastoji ovaj slabi klasifikator, možemo iščitati dodatne informacije. To su pozicije vrhova (`1 10` i `1 14`) i dimenzije pravokutnika (`7 12` i `7 4`) značajke te vrijednosti koeficijenata (`-1` i `3`) s kojima se množi vrijednost značajke. Nakon toga slijedi tip značajke, otkrivajući nam da je to značajka s tri vodoravna pravokutnika (`haar_y3`). Ako je vrijednost značajke manja od praga (`1.565163e-03`) klasifikator poprima vrijednost (`-4.535058e-01`) odnosno (`4.116098e-0`) ako je vrijednost značajke veća od praga. Za primjer znaka i gore navedenog klasifikatora možemo odrediti koje područje slike je prepoznatljivo u tom stupnju kaskade.

Slika 16: Znak i pridružena značajka

5. Rezultati

Za provjeru rezultata koristio sam skriptu u Pythonu *facedetect.py* koja dolazi uz programski paket OpenCV. Kao argumente prilikom pokretanja skripta prima kaskadu u XML formatu i video sekvencu nad kojom želimo provesti detekciju. Rezultat izvršavanja je prikazivanje druge video sekvence s označenim mjestima koje detektor nije odbacio kao dijelove pozadine, tj. dijelovima za koje je odredio da su traženi objekti.

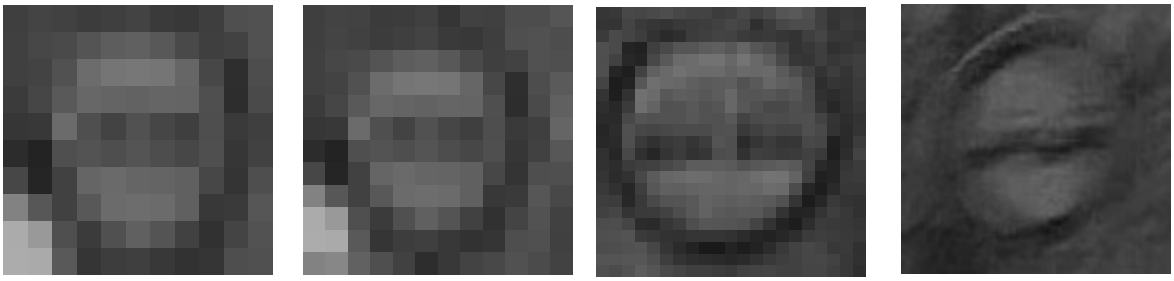
Prilikom izrade rada primijetio sam nekoliko problema koji su ometali uspješnu detekciju. Na jednom dijelu snimke padala je kiša.

Slika 17: Problem detekcije po kiši

Ljudskom oku koje je naučeno na izled prometnog znaka kapljica kiše ne predstavlja veći problem, no računalu svaka smetnja može znak učiniti neprepoznatljivim.

Kvaliteta video zapisa također ima veliki utjecaj. Ako je prometni znak daleko, ne vidimo ga jasno i ne možemo napraviti uspješnu detekciju. Ako je pak znak bliže, postoji opasnost da zbog velike brzine kretanja vozila, a time i kamere, dođe do razmazivanja slike uslijed pokreta (eng. *motion blur*).

Kako ovaj tip detektora ne raspoznaje boje, slike znakova koje detektor vidi izgledaju ovako.



Iz priloženog vidimo da detekcija znakova nije ni približno jednostavan postupak. Čak ni sam, pregledom isječaka nekada ne mogu odlučiti jesam li označio ispravne dijelove slike.

Računao sam s time da će osvjetljenje i vremenski uvjeti utjecati na konačni rezultat. Znao sam da je detekcija ovisna o kvaliteti snimke. Ono na što nisam računao je refleksija unutrašnosti vozila na prednje staklo, a time i na snimku. Ovisno o promjenama u intenzitetu svjetla, reflektirano svjetlo je oblikovalo svakojake oblike i bilo je uzrok brojnih netočnih detekcija. Na sljedećoj slici vidimo refleksiju unutrašnjosti vozila koja mijenja scenu koju je zabilježila kamera.

Slika 18: Refleksija unutrašnjosti vozila

6. Opis razvijenog programa i upute za korištenje

Kao praktično ostvarenje, za ovaj rad napisao sam skriptu u programskom jeziku Python koja olakšava izradu kaskade klasifikatora i pojednostavljuje postupke vezane za njenu izradu. Program olakšava stvaranje skupova za učenje tako što dopušta označavanje primjera prilikom gledanja video zapisa. Nakon stvaranja skupova i izgradnje kaskade klasifikatora rezultat detekcije možemo provjeriti u istom programu.

Argumenti poziva programa određuju njegovo ponašanje. Mogući argumenti su:

`--video=<naziv_video_sekvence>`

Ovaj argument koristimo kako bi programu dali do znanja s kojom vide sekvencom radimo.

`--cascade=<naziv kaskade>`

Ako je zadan argument --cascade prilikom poziva, skripta učitava kaskadu klasifikatora i prikazuje rezultat detekcije na snimci zadanoj pomoću --video argumenta. Zadana kaskada mora biti u XML formatu

`--dump`

Svaki prepoznati objekt se pohranjuje.

Jedini obavezni argument prilikom poziva programa je `--video`, ako je to ujedno i jedini argument, program služi kao alat za označavanje pozitivnih i negativnih primjera na video sekvenci. Dodatkom `--cascade` argumenta program učitava

zadanu kaskadu klasifikatora u XML obliku i prikazuje snimku s detektiranim objektima. Ako uočimo da kaskada nije prepoznala neki znak, možemo ga označiti i koristiti za sljedeće učenje kaskade. Ali čak i nakon nekoliko popravanja skupova za učenje ne možemo biti sigurni da će klasifikator raditi dobro. U mom slučaju puno prepoznatih objekata na snimci su bili lažni pozitivi jer je teško zabilježiti negativne slike sa svim varijacijama koje se mogu dogoditi. Upravo zato sam dodao `--dump` argument. Ako njega koristimo snimaju se svi prepoznati objeki na snimci. Kasnije možemo lako ukloniti stvarne pozitivne primjere slika i ostaviti samo skup koji ćemo pridružiti skupu negativnih primjera. Ovime dodajemo u skup za učenje primjere koje nije odbacio u prošlom izvođenju tjerajući program da se usredotoči na teže primjere.

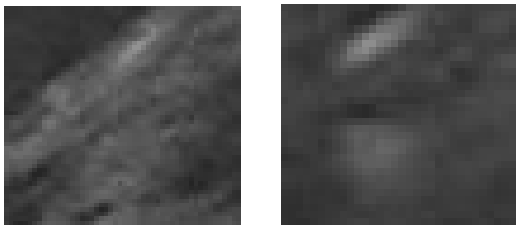
U izradi praktičnog dijela ovog rada, odnosno programa za označavanje i prikaz detekcije, i sam sam izgradio nekoliko kaskada klasifikatora. Dobivao sam raznovrsne rezultate koji su ovisili i poboljšavali se usporedno s brojem primjera koje sam priredio za učenje. Zapazio sam da kvaliteta skupova pozitivnih i negativnih primjera znatno utječe na kvalitetu detekcije. Ako su primjeri međusobno slični, AdaBoost algoritam za učenje usredotočit će se na zajedničke karakteristike koje nisu nužno biti bitne za prepoznavanje šireg skupa znakova. Algoritam i ne zna da postoji širi skup znakova, on je učio nad specifičnim skupom kojeg smo mu priredili. U tom slučaju dobili bi kaskadu koja je stručnjak za prepoznavanje znakova iz skupa pozitivnih primjera, ili u rječniku učenja: štreber. Skup nad kojim učimo kaskadu treba biti što raznovrsniji, sa što više varijacija i različitih primjera.

6.1. Primjer prepoznavanja kaskade

Za primjer prepoznavanja i grešaka pri prepoznavanju odlučio sam koristiti kaskadu koja točno prepoznaje znak u više od 70% slučajeva. Iako prepoznaje više lažnih znakova nego stvarnih, lažne detekcije su obično na neobičnim pozicijama ili nemaju vremenski kontinuitet, tj. rezultat su trenutnog osvjetljenja.

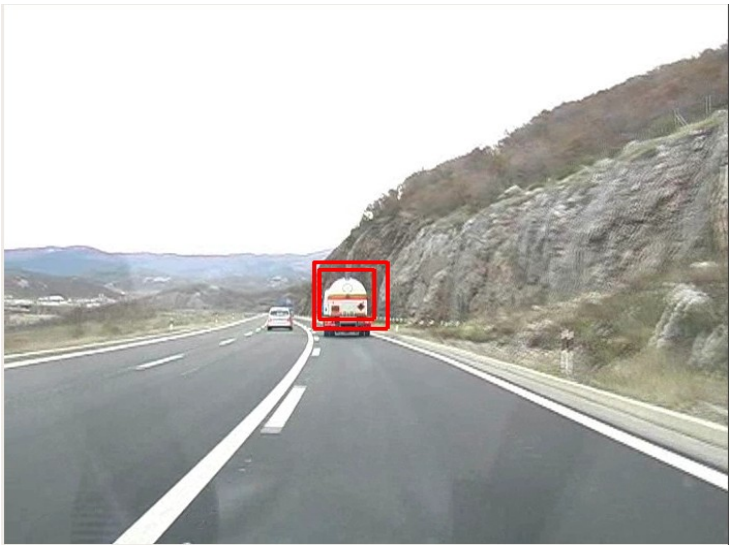
Slika 19: Pogrešne detekcije

Pozicije koje je detektor prepoznao kao moguće znakove zapravo izgledaju ovako:



Primjećujemo da su na obje slike kružni oblici i možemo vidjeti kako su uspjeli zavarati detektor.

Prilikom detekcije susreo sam se i sa sljedećim krivo prepoznatim slikama.



Slika 20: Pogrešna detekcija

Slika 21: Pogrešna detekcija

Ovakve pogrešne detekcije nalazit će se vjerojatno na svakoj snimci, no njihovo izbjegavanje je jako teško. Za odbaciti ovakve lažne znakove program bi morao moći razumijeti kontekst, a to prelazi obujam ovoga rada.

7. Zaključak

U izradi ovoga rada upoznao sam se s OpenCV programskim paketom i metodom detekcije objekata temeljenoj na Haarovim značajkama. Detekcija se zasniva na računanju vrijednosti Haarovih značajki i traženju onih koje najbolje razdvajaju skupove pozitivnih i negativnih primjera. Korištene su samo značajke predložene u radu Viole i Jonesa [2]. Za ubrzanje postupka računanja vrijednosti pravokutnih značajki koristi se integralna slika. Slabi klasifikatori kombiniraju se AdaBoost algoritmom kako bi dobili jaki klasifikator kao linearnu kombinaciju slabih klasifikatora. Dobiveni jaki klasifikatori potom se slažu u kaskadu kako bi dobili što bržu i točniju detekciju.

Dobre strane ovoga pristupa su brzina detekcije, neovisnost o varijacijama objekata koje želimo detektirati i mogućnost učenja sustava na vlastitim pogreškama. No detekcija ovoga tipa ima i neke nedostatke. Kao najveći nedostatak vidim korištenje sivih slika prilikom učenja i detekcije. Vjerujem da se tako gubi jako puno informacija pogotovo kod slika prometnih znakova. Njihove boje su namjerno izrazite kako bi ih vozači lakše uočili, a gubitkom tih informacija ograničeni smo na oblike. Sljedeća mana je da za uspješno naučen detektor trebamo imati puno označenih primjera. U literaturi se spominju brojevi od desetak tisuća označenih primjera za uspješnu detekciju. Uz navedene nedostatke i samo nekoliko stotina označenih primjera u sklopu ovoga rada naučen je detektor prometnih znakova koji ispravno prepoznaje znakove u više od 70% slučajeva. Iako postoji dosta lažno detektiranih znakova, oni nemaju vremenski kontinuitet dok se stvarni znakovi prepoznaju na sličnim lokacijama u bliskim vremenskim

trenutcima. Za postupke koji ovo zapažanje uzimaju u obzir navedeno je u [6] i do 30% uspješnija detekcija, čija implementacija može poslužiti kao daljne proširenje teme ovoga rada.

8. Literatura

[1] Papageorgiou, C., Oren, M., Poggio T. A general framework for object detection. Proceedings of the International Conference on Computer Vision, Bombay, India, 1998.

[2] Viola, P., Jones, M. Robust Real-time Face Detection. Second International Workshop on Statistical and Computational Theories of Vision – Modeling, Learning, Computing and Sampling. Vancouver, Canada, 2001.

[3] Freund, Y., Shapiro, R. E. A Short Introduction to Boosting: A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences. 55(1):119–139.

[4] Zampieron, J. M. D., Self-Localization in Ubiquitous Computing using Sensor

Fusion. Rochester Institute of Technology, Rochester, New York, 2006.
[5] Shapiro, L. G., Stockman, G., Computer Vision. Prentice-Hall, 2000.

[6] Zisserman, A. Object Recognition: Face Detection. Computer Vision Lectures. University of Oxford, 2004.

[7] Lienhart, R., Liang, L., Kuranov, A. A Detector Tree of Boosted Classifiers for Real-Time Object Detection and Tracking. IEEE ICME2003, Vol. 2, pp. 277-280, 2003.

[8] Open Computer Vision Library. *http:/sourceforge.net/projects/opencvlibrary*

[9] Dalbelo-Bašić, B. Strojno učenje: Uvod, Bilješke za predavanja UI ak. g. 2007./2008., FER, 2008.

[10] Van Brakel, R. B. J., Hanckmann, P. Eye Tracking: Additional component computer graphics, Eindhoven University of Technology, Faculty of Mathematics and Computer Sciences, 2008.

9. Sažetak

Detekcija prometnih znakova u video sekvencama

U radu je dan pregled nekih od metoda strojnog učenja te detaljnije obrađena metoda računalnog vida temeljena na pravokutnim Haarovim značajkama definiranim u radu Viole i Jonesa [2]. Dan je opis Haarovih značajki i AdaBoost algoritma kombiniranja niza slabih klasifikatora u jaki klasifikator. Objašnjen je postupak pripreme primjera za treniranje kaskade klasifikatora i izgradnje kaskade temeljem programa uključenih u *open-source* biblioteku programskog paketa OpenCV. U sklopu rada napisana je i opisana implementacija programskog sustava za detekciju prometnih znakova izgrađenog u programskom jeziku Python temeljem programskog paketa OpenCV. Analizirana je dobivena kaskada klasifikatora i objašnjeni najčešći problemi prilikom detekcije prometnih znakova na snimkama prometnih situacija.

Ključne riječi:

Računalni vid, strojno učenje, detekcija objekata, detekcija prometnih znakova, Haarove značajke, AdaBoost algoritam, OpenCV biblioteka

9. Abstract

Road Sign Detection in Video Sequences

This paper gives overview of some of machine learning algorithms and discusses computer vision method based on Haar-like features defined in paper by Viola and Jones [2]. It describes Haar-like features and AdaBoost algorithm which selects a small number of critical visual features and yields efficient classifiers. Process of preparing examples for teaching classifier cascades and the construction of these cascades based on the programs included in OpenCV, open-source computer vision library, is explained. As part of the work, Python script based on OpenCV library for road sign detection was written and described. Resulting classifier cascade was analysed and most common problems in road signs detection based on Haar-like features were discussed.

Keywords:

Computer vision, machine learning, object detection, road sign detection, Haar-like features, AdaBoost algorithm, OpenCV library