Hardverska implementacija Viola-Jones algoritma

Risto Pejašinović

Sadržaj

Deo I
Viola-Jones algoritam

1 Uvod

Namena algoritma je detekcija i lokalizacija objekata na slici. Osmišljen od strane Paul Viola i Michael Jones 2001. godine [1].

Dugo godina je zbog brze i pouzdane detekcije bio standardan način detekcije lica na slici. I danas je prisutan u velikom broju mobilnih telefona i digitalnih kamera, ali danas postaje polako zamenjen konvolucionim neuronskim mrežama.

Pouzdanost i brzina su postignuti uvođenjem tri ključna doprinosa:

- **Integralna slika** omogućava brzo izračunavanje obeležja.
- **AdaBoost** algoritam za učenje, odabiranjem obeležja povećava brzinu i pouzdanost detekcije.
- **Kaskadni klasifikator** organizovanjem obeležja u kaskadama omogućava brzo odbacivanje pozadine slike.

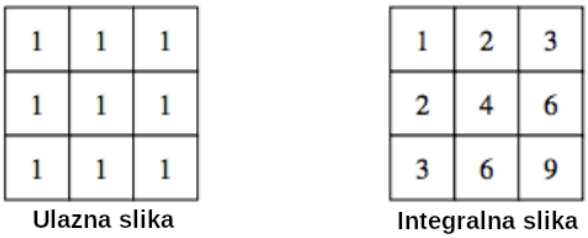
1.1 Integralna slika

Kao jedan od ključnih delova algoritma, integralna slika omogućava izračunavanje površine svakog pravouganog obeležja u konstantnom vremenu.

Intenzitet piksela u integralnoj slici na poziciji x,y je zbir svih piksela koji se nalaze gore i levo od pozicije x,y.

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{1}$$

Gde je ii(x,y) integralna slika, a i(x,y) originalna slika.

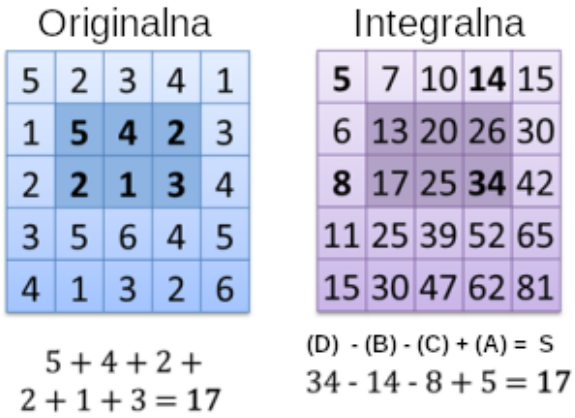


Slika 1: Primer integralne slike

Piksele integralne slike je moguće računati u paraleli, ili sekvencijalno. Izbor algoritma za računanje integralne slike značajno utiče na performanse i potrebne hardverske resurse.

U paralelnoj implementaciji cena je više pristupa memoriji i više potrebnih sabirača, dok je kod sekvencijalne implementacije manja brzina proračunavanja.

Osobina koja integralnu sliku čini pogodnu za korišćenje u Viola-Jones algoritmu je da je za računanje bilo koje pravougaone površine unutar integralne slike potrebno 2 oduzimanja i 1 sabiranje.



Slika 2: Primer računanja površine pravougaonika [2]

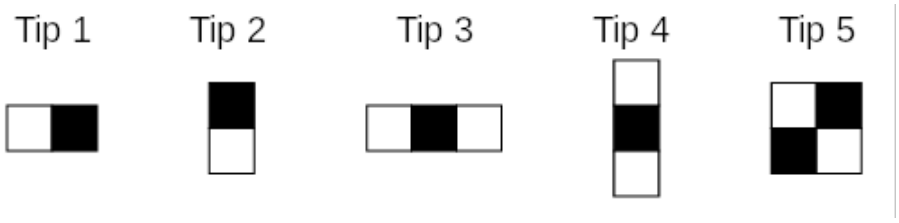
Na slici (2) je prikazano računanje površine pravougaonika na originalnoj slici i na integralnoj slici. Kao što se može videti za površinu pravougaonika MxN na originalnoj slici nam je potrebno MxN-1 sabiranja.
Dok je kod integralne slike broj operacija 2 oduzimanja i 1 sabiranje i ne zavisi od dimenzija pravougaonika.

$$\sum_{(x,y)\in ABCD} i(x,y) = ii(D) + ii(A) - ii(B) - ii(C)\text{[3]} \qquad (2)$$

1.2 AdaBoost i HAAR obeležja

1.2.1 HAAR obeležja

Detekcija na slici se vrši na prozorima manjih dimenzija. Na ovim prozorima računaju se HAAR obeležja koja se sastoje od dva ili više pravougaonika, kao na slici (3).
Svako obeležje se računa sabiranjem piksela crnih pravougaonika potom oduzimanjem zbira piksela belih pravougaonika.
Reprezentacija prozora pomoću integralne slike omogućava da se obeležja izračunavaju u konstantnom vremenu.



Slika 3: HAAR obeležja [4]

Dimenzije prozora se razlikuju od modela klasifikatora. Jedan često korišćeni model koristi prozor dimezije 24x24.

Slika 4: Primer obeležja za detekciju lica [1]

Oblik odabranih obeležja će zavisiti od namene detektora, na slici(4) se mogu videti dva tipična obeležja koja su od interesa za detekciju lica.
Prvo obeležje namenjeno je merenju razlike intenziteta regiona čela i očiju. Obeležje koristi činjenicu da je oblast čela svetlija od očiju.
Dok drugo obeležje poredi intenzitet regiona mosta nosa sa očima.

Kako su obeležja od interesa koja se koriste u modelima na slici(3). Za datu dimenziju prozora kombinacije svih mogućih varijacija oblika i pozicija datih obeležja čini skup od 160.000 različitih obeležja. Kako je većina ovih obeležja slična i davaće slične rezultate, ovaj broj se može drastično smanjiti korišćenjem algoritma za učenje AdaBoost.

1.2.2 AdaBoost

Kako je već rečeno može se dobiti oko 160.000 obeležja za prozor dimenzije 24x24. Od ovog broja samo neka obeležja mogu dati dobre rezultate prilikom detekcije, kao na slici(4) u primeru detektora lica.

Kako bi se odabrao skup korisnih obeležja može se koristiti neki od algoritama mašinskog učenja. Viola i Jones predlažu modifikovani AdaBoost algoritam.

Ideja AdaBoost-a je kombinovanje više *weak learner*-a kako bi se dobila pouzdana detekcija.
Weak learner je kalsifikator koji ima pouzdanost pogađanja malo bolju nego nasumičnu. Odnosno pouzdanost *weak learner*-a mora biti bar malo iznad 50%.

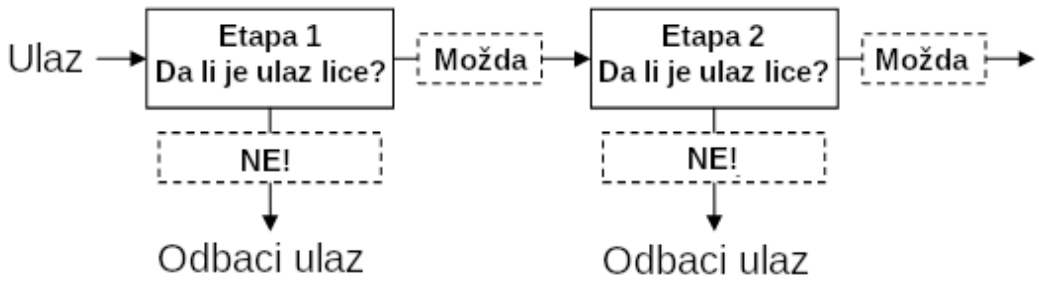
Kombinacijom ovako dobijenih *weak learner*-a može se dobiti *strong classifier*

Kao rezultat AdaBoost algoritma dobićemo skup obeležja, u ovom radu se koristi model sa skupom od 2913 obeležja.

1.3 Kaskadni klasifikator

Osnovni princip Viola-Jones algoritma je da se na osnovu svih obeležja u modelu dobije informacija da li se na trenutnom položaju prozora nalazi traženi objekat (npr. lice).
Kako na slici većina skeniranih regiona ne sadrži lice, računanje svih obeležja na svakoj poziciji bi bilo suvišno. Tako da je korišćenje jednog jakog klasifikatora neefikasno.

Ideja obrazovanja kaskadnog klasifikatora je da se prozori na kojima se očigledno ne nalazi lice odbace brzo, nakon samo nekoliko izračunatih obeležja.

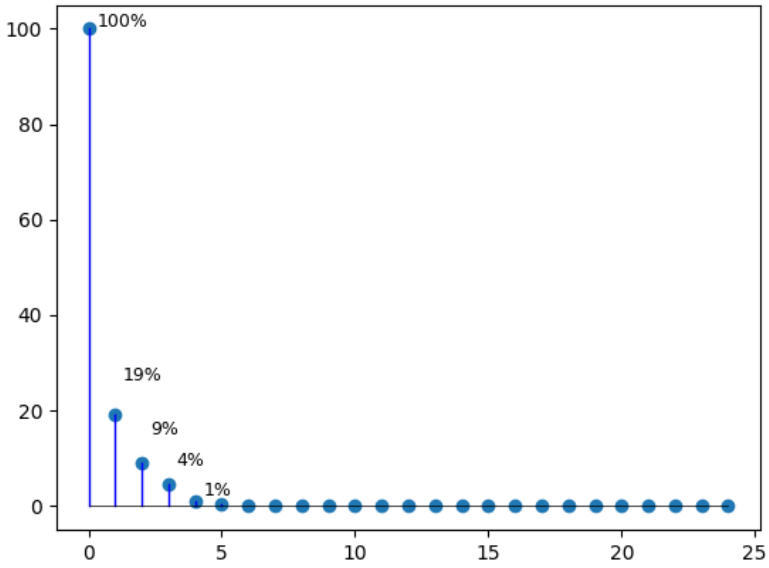


Slika 5: Kaskadni klasifikator [4]

Kako bi se prozori bez lica brzo odbacili predlog je da se jaki klasifikatori grupišu u etape (eng. *stage*). Svaka etapa treba da bude dobra u odlučivanju da li se na analiziranom prozoru definitivno ne nalazi lice. Ukoliko je to slučaj taj prozor će se brzo odbaciti.
Ukoliko rezultat etape ukazuje na to da se na prozoru možda nalazi lice, preći će se na izvršavanje sledeće etape.

Konačno ukoliko sve etape u klasifikatoru na analiziranom prozoru daju rezultat da se možda nalazi lice, može se zaključiti da se na toj poziciji zaista nalazi lice.
Zahvaljujući ovome postiže se veoma pouzdan klasifikator sa malim procentnom pogrešno negativnih (eng. *false negative*) rezultata na krajnjim etapama.

Kao primer u ovom radu će se koristiti model kaskadnog klasifikatora za prepoznavanje lica, sa 25 etapa i 2913 obeležja raspoređenih po etapama.
U prvoj etapi se nalazi samo 9 obeležja, dok taj broj raste do 211 u kasnijim etapama.



Slika 6: Procenat prolaska etapa

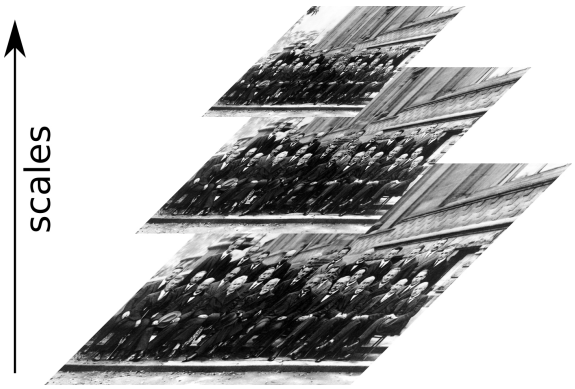
Na slici(6) je prikazana statistika izvršavanja etapa na *Caltech Dataset-u*[5], koji sadrži 450 slika od 27 različitih ljudi pod različitim osvetljenjima, izrazima i pozadinama.

Vrednosti različitih tačaka na grafu predstavlja procenat izvršavanja date etape na svih 450 slika. Prva etapa će se naravno uvek izvršiti, dok će se druga etapa izvršiti samo u 19% analiziranih prozora, druga 9% itd...
Vidimo da je posle pete etape procenat izvršavanja manji od 1%.

1.4 Invarijantnost veličine

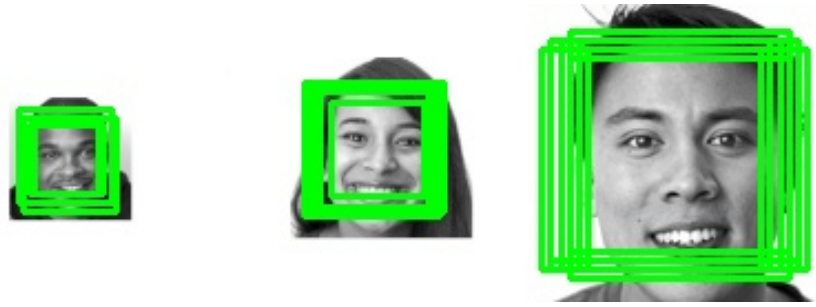
Kako lica na slikama mogu biti bliže ili dalje kameri odnosno mogu biti različitih dimenzija, potrebno je obezbediti da se ona detektuju nezavisno od veličine.

Pošto su obeležja istrenirana da detektuju samo lica koja su iste dimenzije kao i veličina obeležja potrebno je obezbediti skaliranje slike ili obeležja kako bi mogli da detektujemo lica veća od dimenzije obeležja.
Pri čemu je najmanja dimenzija lica koje je moguće detektovati jednaka dimenziji obeležja.



Slika 7: Piramida slike[6]

Ovo je rešeno uvođenjem skaliranja slike. Kao na slici(7) prvo je potrebno odraditi detekciju na originalnoj slici, zatim se slika smanjuje sa nekim faktorom i ponovo se vrši detekcija. Skaliranje se vrši sve dok je dimenzija skalirane slike veća od dimenzije obeležja.



Slika 8: Različite veličine lica

Na slici(8) može se videti rezultat klasifikatora za 3 lica različitih dimenzija.

1.5 Invarijantnost osvetljaja

Lica se mogu naći pod raznim osvetljenjima što uzrokuje problem ovom algoritmu.

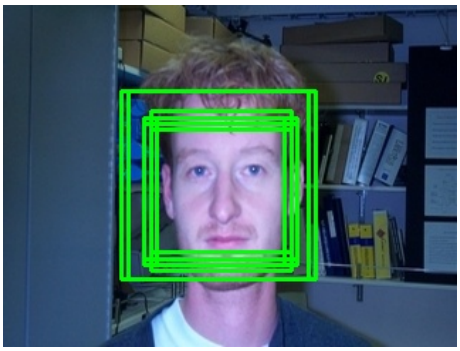
Slika 9: Previše osvetljaja[5]

Slika 10: Premalo osvetljaja[5]

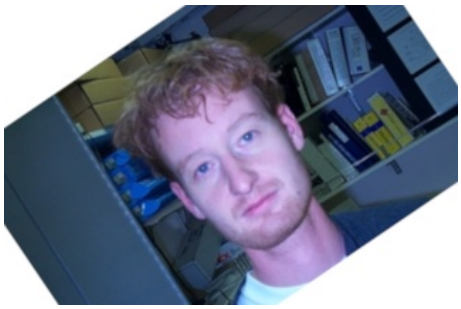
Na slikama(9, 10) su prikazane dve slike koje zbog nepovoljnog osvetljenja nisu detektovane od strane klasifikatora. Slika(9) nije detektovana zbog prevelikog osvetljenja lica, dok Slika(10) nije detektovana zbog slabog osvetljenja.

Kao delimično rešenje ovog problema uvedeno je računanje standardne devijacije prozora koji se detektuje. Ovo može poboljšati detekciju u nekim slučajevima, ali ne i ekstremnim kao u prethodnom primeru.

1.6 Varijantnost rotacije



Slika 11: Ne rotirana[5]



Slika 12: Rotirana[5]

2 OpenCV modeli

OpenCV sadrži alat za treniranje kaskada i detekciju objekata pomoću Viola-Jones algoritma. Takođe dolazi sa istreniranim i testiranim klasifikatorima[1] [7]

Rezultat treniranja se smešta u .xml fajl koji sadrži sledeće informacije:

- Dimenzija obeležja (*height, width*)
- Broj etapa (*stageNum*)
- Maksimalan broj obeležja u etapi (*maxWeakCount*)
- Informacije o etapama (*stages*)
  - Broj obeležja u etapi (*maxWeakCount*)
  - Prag etape (*stageThreshold*)
  - Informacije o obeležjima (*weakClassifiers*)
    - Prag obeležja (*internalNodes*)
    - Vrednosti listova (*leafValues*)
- Informacije o obeležjima (*features*)
  - Koordinate i težine tačaka pravougaonika (*rects*)
    Svako obeležje može imati 2 ili 3 pravougaonika
    Gde su prve 2 vrednosti x i y koordinate gornje leve tačke,
    Treća i četvrta vrednost širina i visina pravougaonika
    Poslednja vrednost težina pravougaonika

2.1 OpenCV model za frontalna lica

Često korišćeni model za detekciju lica je `haarcascade_frontalface_default.xml`[2].
Ovaj model se koristi za frontalnu detekciju lica.
Neke njegove karakteristike su:

- Dimenzija obeležja: 24x24
- Broj etapa: 25
- Maksimalan broj obeležja u etapi: 211
- Ukupan broj obeležja: 2913

Rezultati detekcije ovog modela mogu se videti na slikama(8,9,10,11,12) iz sekcije 1.

[1] `https://github.com/opencv/opencv/tree/master/data/haarcascades`

[2] `https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml`

Deo II
Projekat

3 Sažetak

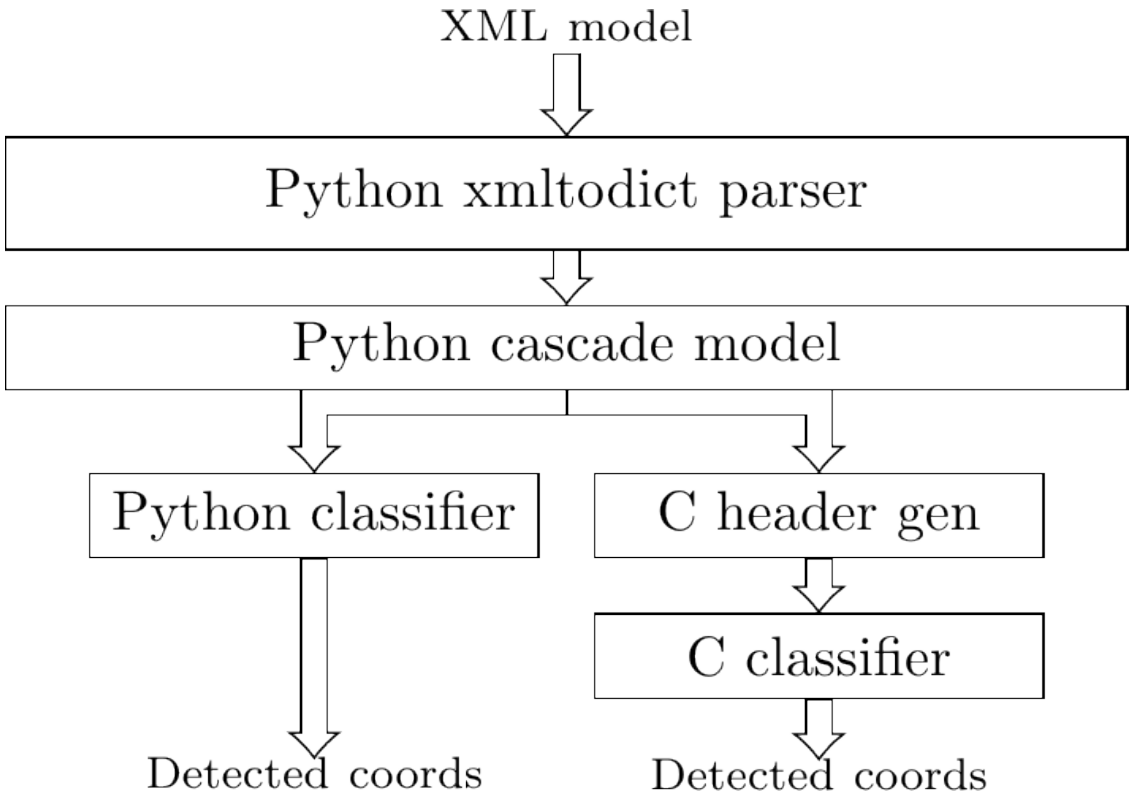
Ovaj projekat sadrži:

- Pisanje specifikacije u Python i C programskom jeziku za izvršavanje i pomoć pri particionisanju i projektovanju hardvera.
- Projektovanje arhitekture hardverskog akceleratora za Viola-Jones algoritam opisam u delu I.
- Pisanje HDL modela za sintezu u SystemVerilog RTL metodologiji i PyGears[3] metodologiji.
- Pisanje verifikacionog okruženja u SystemVerilog UVM[4] i Python PyGears okruženju.
- Poređenje dve metodologije i analiza prednosti i mane obe metodologije.
- Poređenje komercijalnog Questa Sim[5] simulatora i besplatnog open-source Verilator[6] simulatora.
- Implementacija projektovanog IP jezgra na MYIR Z-Turn Board[7] sa Zynq 7020 SoC.
- Pisanje Linux Kernel drajvera i korisničke aplikacije za korišćenje jezgra za detekciju lica na Xilinx Zynq platformi.

[3] https://github.com/bogdanvuk/pygears
[4] https://www.accellera.org/downloads/standards/uvm
[5] https://www.mentor.com/products/fv/questa/
[6] https://www.veripool.org/wiki/verilator
[7] http://www.myirtech.com/list.asp?id=502

4 Specifikacije za izršavanje

Kako bi se hardver efikasno projektovao i particionisao potrebno je početi od softverske implementacije na višem nivou apstrakcije.
Jezici visokog nivoa kao što je Python su veoma dobri za ovu namenu.



Slika 13: Veza Python modela sa XML modelom i C specifikacijom

Na slici(13) prikazana je struktura koda u slučaju Python i C klasifikatora.

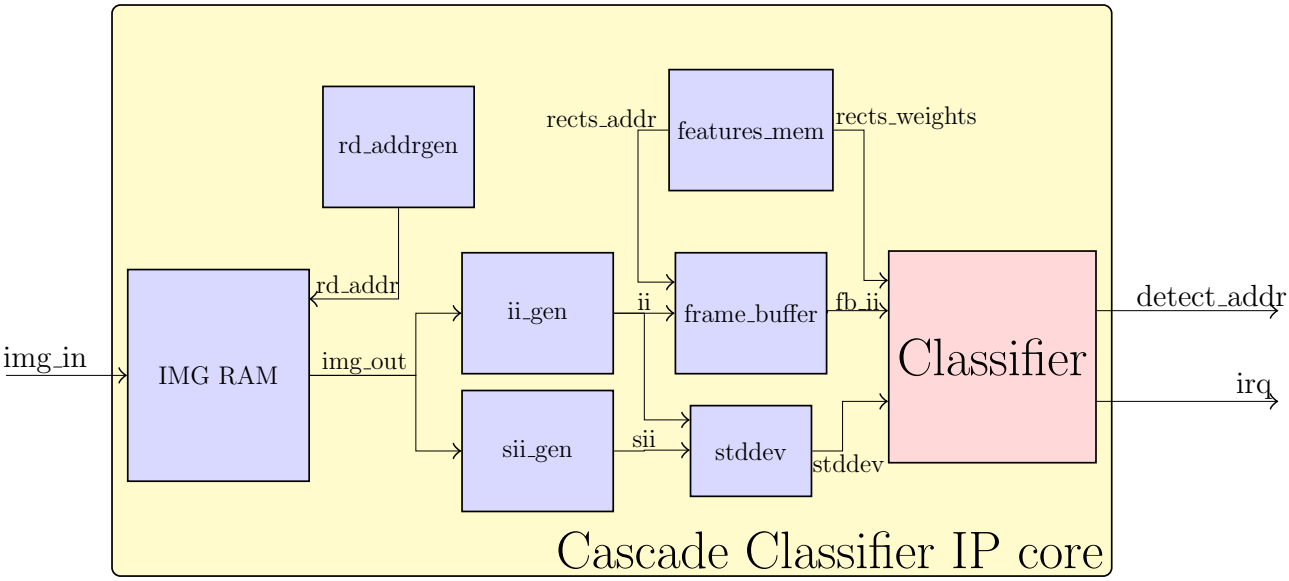
Na ulazu se nalazi XML model dobijen treningom pomoću OpenCV opisan u sekciji 2.
Kao jezik za parsiranje XML fajla se koristi Python. Zbog velikog broja Python paketa dostupnih sa gotovim rešenjima za većinu softverskih problema, problem parsiranja XML fajla se može rešiti korišćenjem paketa xmltodict[8].
Xmltodict parsira XML fajl i skladišti ga u Python dictionary.
Klase *CascadeClass*, *StageClass*, *FeatureClass* i *RectClass* [9] napisane u Pythonu predstavljaju Python model kaskadnog klasifikatora.

Napisan je i Python specifikacija za isvršavanje koja direktno koristi Python klase za detekciju objekata.
Veza između Python modela i C specifikacije realizovana je preko *C Header* fajla koji je generisan pomoću Python-a.

[8] `https://pypi.org/project/xmltodict/`
[9] `cascade_classifier/python_model/cascade.py`

5 Arhitektura hardvera

asdasd



Slika 14: Arhitektura hardvera kaskadnog klasifikatora

Na slici(15) prikazani su krupni moduli arhitekture hardvera. Sa izostavljenim detaljima impementacije pojedinačnih modula.

5.1 Interfejsi IP jezgra

IP jezgro se povezuje pomoću 3 interfejsa:

- Ulazni interfejs **img_in** sastoji se od 8-bitnog podatka vrednosti piksela slike u *grayscale* formatu (nijanse sive).
- Izlazni interfejs **detect_addr** ukoliko jezgro detektuje lice na slici postaviće x i y koordinate na ovom interfejsu.
- Izlazni interfejs **irq** je signal koji označava završetak obrade slike i signalizira da je jezgro spremno za novu sliku. Namenjen da se koristi kao prekidni signal za procesor.

5.2 Modul IMG RAM

Modul **IMG RAM** je zadužen za skladištenje slike koja se obrađuje. Sastoji se od RAM memorije i brojača za generisanje adrese za upis.
Nakon primljenog podatka na ulazu brojač adrese upisa se poveća za 1.
Skladištena slika je u 8-bitnom formatu i predstavlja grayscale vrednost piksela originalne slike.

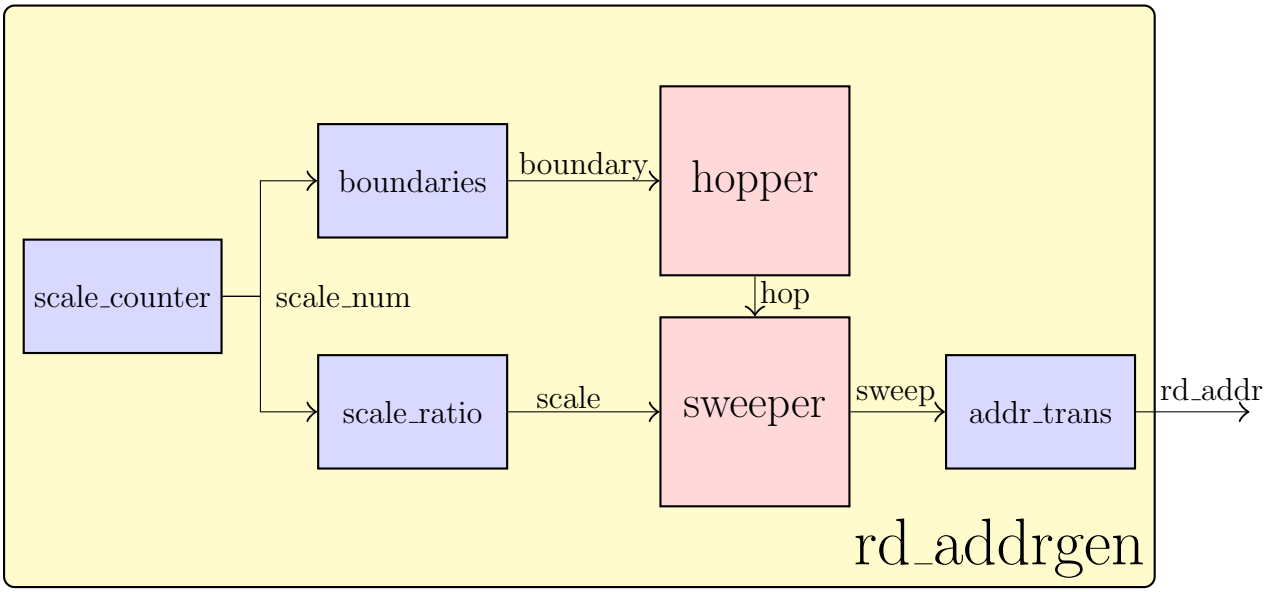
Veličina memorije je $width * height * 8$ bit. Za dimenziju slike 240x320 veličina memorije je 614400 bita.

Komunicira sa okolinom pomoću 3 interfejsa:

- Ulazni interfejs **img_in** ekvivalentan je sa istoimenim interfejsom na višem nivou hijerarhije.
- Ulazni interfejs **rd_addr** predstavlja linearnu adresu generisanu od strane **rd_addrgen** modula. Koristi se za čitanje podataka iz RAM memorije.
- Izlazni interfejs **img_out** predstavlja iščitane podatke iz RAM memorije.

5.3 Modul rd_addrgen

asd



Slika 15: Arhitektura hardvera kaskadnog klasifikatora

Literatura

[1] P. A. Viola and M. J. Jones, "Rapid object detection using a boosted cascade of simple features," in *CVPR*, 2001.

[2] A. Jain, "Computer vision – face detection," 2016. [Online]. Available: https://vinsol.com/blog/2016/06/28/computer-vision-face-detection/

[3] K. Cen, "Study of viola-jones real time face detector," 2016.

[4] O. Jensen, "Implementing the viola-jones face detection algorithm," 2008.

[5] M. Weber, "Frontal face dataset," 1999. [Online]. Available: www.vision.caltech.edu/Image_Datasets/faces/faces.tar

[6] Z. Ye, "5kk73 gpu assignment 2012," 2012. [Online]. Available: https://sites.google.com/site/5kk73gpu2012/assignment/viola-jones-face-detection

[7] OpenCV, "Opencv$_d ocs.''[Online].Available$: