DETEKCIJA LJUDSKOG LICA U VIDEOTIJEKU U REALNOM VREMENU

Marošević, Javor

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

DETEKCIJA LJUDSKOG LICA U VIDEOTIJEKU U REALNOM VREMENU

Javor Marošević

Zagreb, siječanj 2018.

Student vlastoručno potpisuje završni rad na prvoj stranici ispred predgovora s datumom i oznakom mjesta završetka rada te naznakom:

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada.“

U Zagrebu, datum.

Ime Prezime

Predgovor

Zahvaljujem mentoru dr. sc. Goranu Đambiću na iskazanom povjerenju, korisnim sugestijama i vodstvu tijekom izrade ovog rada. Nadalje, želim zahvaliti svojim roditeljima, prijateljima, kolegama i šefu koji su me podupirali i izlazili u susret tijekom studija. Također, zahvaljujem svim profesorima, asistentima i djelatnicima Visokog učilišta Algebra koji su svojim radom pomogli u stjecanju mog znanja i razvoju profesionalnih sposobnosti.

Javor Marošević

Prilikom uvezivanja rada, umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada koji ste preuzeli u studentskoj referadi

Sažetak

Profesionalna motivacija predlaganja ove teme je na stvarnom projektu istražiti mogućnost zadovoljavajuće točne detekcije lica osoba u realnom vremenu koristeći biblioteke otvorenoga koda. Ovaj je rad zamišljen kao prva faza u pokušaju izrade sustava koji bi sa zadovoljavajućom točnošću mogao prepoznati prisutnost studenata u predavaonici što bi moglo dovesti do njegova korištenja u automatskom ažuriranju dolaznosti studenata na predavanja ili vježbe. Ispitivat će se scenarij detekcije u kojem pokušavamo odrediti točnost detekcije s obzirom na broj osoba u prostoru i njihovu udaljenost od kamere te ocijeniti je li razina detekcije zadovoljavajuća. Ovakav sustav imao bi primjenu i u detekciji prisutnosti osoba u tzv. pametnim kućama te također mogao služiti za praćenje kretanja osoba kroz motreni prostor. Koristit će se metode izrade prototipa softvera, testiranje funkcionalnosti prema utvrđenom planu testiranja, mjerenje karakteristika sustava u radu te analiza rezultata.

Ključne riječi: detekcija objekata, detekcija lica, HOG, konvolucijske neuronske mreže.

Abstract

Professional motivation for nominating this subject is to explore the possibility of satisfactorily accurate human face detection in real time on a real-life project using open source libraries. This project is envisioned as a first phase in an attempt to build a system which could, with satisfactory accuracy, detect student presence in the classroom which could lead to its use in automatic student class or laboratory practice attendance registration. A detection scenario will be tested in which we are trying to determine detection accuracy considering number of subjects in the detection area and their distance from the camera and assess if the level of detection is at an acceptable rate. This type of system would be applicable in subject presence detection in „Smart home“ environments and could be used to track subjects movement trough an observer area. Methods that will be used are construction of a software prototype, functionality testing in accordance to a predetermined testing plan, measurement of system performance and result analysis.

Keywords: object detection, face detection, HOG, convolutional neural networks.

Sadržaj

1. Uvod

Detekcija objekata unutar slike pomoću računala ima sve veću primjenu u mnogim segmentima industrije, transporta i svakodnevnog života. Ona je ključan proces u većini aplikacija kao što su samovozeći automobili, detekcija pješaka i vozila u sustavima pametnih gradova, brojači osoba u trgovačkim centrima, kontrola proizvodnje u proizvodnim pogonima, pretraživanje *online* slika, sigurnosni sustavi i, dakako, detekcija lica. Dakle, detekcija lica kao takva spada pod poseban slučaj detekcije objekata.

Detekcija ljudskog lica unutar neke slike ima razne primjene, od automatskog prepoznavanja lica na fotografijama objavljenim na društvenim mrežama, do detekcije ljudske prisutnosti u sustavima pametnih kuća.

Iako ljudima detekcija objekata nije velik problem i vrlo nam je lako odrediti pojedine objekte unutar neke slike, kod računala to nije slučaj. Kada bismo pokušali odrediti neki proces ili algoritam, vjerojatno bismo vrlo brzo upali u problem jer kod računala imamo na raspolaganju sliku sastavljenu od mnoštva malih elemenata, piksela (engl. *pixel*), od kojih svaki ima neku određenu vrijednost. Ako bismo rekli da neka kombinacija piksela na toj slici predstavlja određeni objekt, računalo bi bilo u stanju prepoznati taj objekt na ostalim slikama, ali samo pod uvjetom da je na drugoj slici objekt identične veličine, u identičnoj pozi i u identičnim uvjetima osvjetljenja. Ovakav scenarij nije vjerojatan i zbog toga detekciji objekata moramo pristupiti na drugačiji način, bliži načinu na koji detekciju objekata obavlja i sam čovjek.

Isto tako, prilikom korištenja računala, metode detekcije moramo svesti na elementarne matematičke operacije jer računalo u svojoj srži poznaje samo takvu vrstu obrade podataka te nas to dovodi do idućeg problema – brzine obrade pojedine slike. Za detekciju je potrebno obaviti velik broj kalkulacija i makar su današnji procesori nevjerojatno brzi strojevi, broj operacije i dalje je problem. Prilikom obrade pojedinačnih slika to možda ne dolazi do izražaja, ali prilikom obrade videotijeka, gdje se radi o nekoliko desetaka slika u sekundi, detekcija se mora obaviti u nekoliko milisekundi.

Iako ćemo napraviti pregled glavnih algoritama detekcije objekata, u ovom radu nećemo se baviti implementacijom samih algoritama detekcije već ćemo koristiti gotove implementacije biblioteka otvorenoga koda.

2. Teorija i praksa detekcije lica pomoću računala

Kod zadavanja bilo kakve upute računalu moramo je opisati algoritamski, tj. moramo dobro definirati korake rješavanja tog problema do najsitnijih detalja. U ovom dijelu rada opisat ćemo tri poznate tehnike i algoritme detekcije ljudskog lica pomoću računala. Pod tehnikom smatramo način pristupa i skup rješavanja danog problema.

2.1. Pregled tehnika i algoritama detekcije lica

Detektor lica, kada mu se preda slika ili video, trebao bi moći identificirati i locirati prisutna lica bez obzira na njihovu poziciju, veličinu, orijentaciju, starost i izraz lica. Nadalje, detekcija bi trebala biti odrađena bez utjecaja uvjeta osvjetljenja i ostalog sadržaja slike. [1]

Procesiranje slike odvija se na sljedeći način: ulazna slika skenira se na svim mogućim lokacijama u svim veličinama koristeći potprozor (engl. *subwindow)* manje veličine. [1] To znači da potprozor povlačimo preko ulazne slike krenuvši od gornjeg lijevoga kuta pomičući ga za određeni broj piksela udesno sve dok ne dođemo do kraja tog retka slike. Nakon toga vraćamo potprozor na početak retka, pomaknemo ga za određeni broj piksela dolje te cijeli proces ponavljamo dok nismo skenirali cijelu ulaznu sliku. Zatim povećamo ili smanjimo veličinu potprozora i krenemo skeniranje slike ispočetka. Cijeli postupak ponavljamo dok nismo obradili sve željene veličine potprozora. Veličina korištenog potprozora direkto utječe na veličinu lica koje će biti detektirano na ulaznoj slici. Veći potprozor znači detekciju većeg lica i obrnuto. Problem detekcije objekata možemo smatrati kao problem klasifikacije uzorka koji se nalazi unutar trenutnog potprozora. Kod nekih vrsta detekcije pokušavamo prepoznati više klasa objekata. Npr. pitamo se je li u potprozoru mačka, pas ili ptica. Pripada li uzorak jednoj od te tri klase objekata? Detekcija lica obavlja se kao klasifikacija uzorka u potprozoru koji svrstavamo u jednu od dvije klase: ili je lice ili nije lice. [1] Kod same detekcije najvažnije obilježje ljudskog lica su oči te svaka tehnika detekcije lica kao glavnu značajku detekcije traži upravo ljudske oči. [2]

Svaka tehnika detekcije objekata ima dvije faze. Prva je treniranje modela, gdje računalu dajemo mnoštvo već klasificiranih slika u kojima smo pokazali gdje se na slici nalazi traženi objekt. Na temelju tih informacija računalo izvlači obilježja svojstvena za te objekte i stvara model po kojem će kasnije obavljati detekciju na novim, nepoznatim slikama. Pod modelom

smatramo skup naučenih informacija koje računalo koristi da bi moglo prepoznati takav objekt u kasnijim pokušajima detekcije.

Druga faza je postupak detekcije objekata na nepoznatim slikama u kojoj računalo pokušava pronaći svojstvena obilježja traženog objekta unutar slike. Prilikom detekcije može doći do dvije vrste pogrešaka: neodrađene detekcije koja se događa kada računalo ne prepozna traženi objekt unutar slike te lažne detekcije koja se događa kada računalo napravi detekciju koja nije istinita, tj. traženi objekt se ne nalazi na detektiranome mjestu.

Iako je u prošlosti bilo pokušaja za izradu algoritma za efikasnu detekciju lica u realnom vremenu, najveći uspjeh imali su Paul Viola i Michael Jones 2001. godine. Iako se njihov model može upotrijebiti za detekciju drugih vrsta objekata, glavna motivacija za razvoj algoritma bila je upravo detekcija ljudskog lica.

2.1.1. Viola-Jonesov algoritam

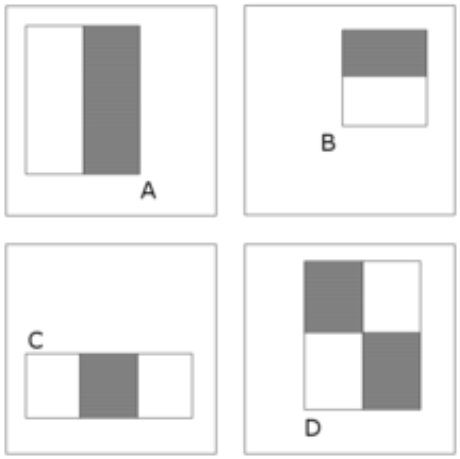
Viola-Jonesov algoritam temelji se na nekoliko koncepata:

- Haarovim značajkama
- integralnoj slici
- Adaboostu
- kaskadi klasifikatora.

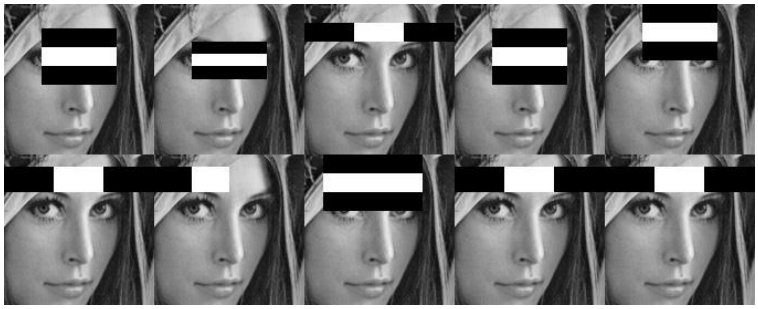
Haarove značajke

Viola-Jonesova procedura detekcije klasificira slike temeljeno na vrijednostima jednostavnih značajki (Slika 2.1.). U izvornom radnom okviru korištene su četiri vrste značajki. Vrijednost svake značajke dobije se tako da se oduzmu ukupne vrijednosti piksela koji se nalaze pod svakom vrstom pravokutnika u značajki. Npr. kod značajke „A“ oduzela bi se ukupna vrijednost svih piksela koji se nalaze pod bijelim pravokutnikom od ukupne vrijednosti piksela koji se nalaze pod crnim pravokutnikom. [3]

Ova metoda donosi vrlo dobre rezultate jer se izdvojene vrijednosti značajki odnose na većinu ljudskih lica koje želimo detektirati. Npr. na slici (Slika 2.2.) vidimo primjenu značajki na sliku lica. Ako primijenimo Haarovu značajku C u područje nosa na bilo koje lice, dobit ćemo sličan rezultat jer je razlika ukupne vrijednosti piksela pod pravokutnikom koji prekriva sredinu nosa i dva pravokutnika koji prekrivaju područje lijevo i desno (oba oka) slična (sredina nosa je svjetlija od očiju).



Slika 2.1. Haarove značajke[1]



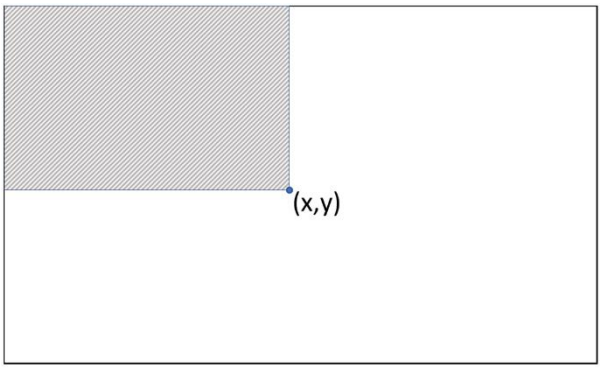
Slika 2.2. Haarove značajke primijenjene na sliku[2]

Integralna slika

Integralna slika je jedan od glavnih razloga uspjeha Viola-Jonesova radnog okvira. Ona omogućuje vrlo brz izračun ukupne vrijednosti piksela u zadanom pravokutniku, bez obzira na to gdje se on nalazio u samoj slici. Integralna slika je trenutna reprezentacija izvorne slike u kojoj vrijednost na nekoj koordinati odgovara zbroju vrijednosti svih piksela u pravokutniku iznad i lijevo (Slika 2.3.).

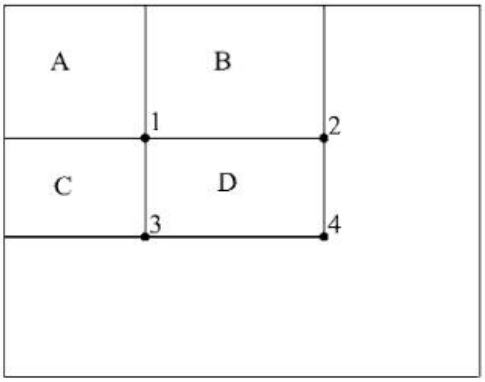
Integralna slika računa se jednom za svaku sliku u kojoj želimo pronaći lica te se potom ona koristi za računanje vrijednosti piksela ispod Haarovih značajki. Vrijednost piksela pod bilo kojim pravokutnikom na slici može se izračunati pomoću vrijednosti četiri točke na integralnoj slici.

[1] Preuzeto 21. prosinca 2017. sa https://en.wikipedia.org/wiki/Viola-Jones_object_detection_framework#/media/File:Prm_VJ_fig1_featureTypesWithAlpha.png
[2] Preuzeto 25. siječnja 2018. sa http://comp3204.ecs.soton.ac.uk/cw/c6223_coursework2.html



Slika 2.3 Vrijednost integralne slike u točki (x, y) jednaka je sumi vrijednosti svih piksela gore i lijevo

Ukupna vrijednost piksela u pravokutniku D prikazanom na slici (Slika 2.4.) računa se na sljedeći način: vrijednost integralne slike u točki 1 odgovara ukupnoj vrijednosti piksela u pravokutniku „A“. Vrijednost u točki 2 odgovara ukupnoj vrijednosti piksela A + B, u točki 3 ukupnoj vrijednosti A + C te u točki 4 ukupnoj vrijednosti A + B + C + D. Iz toga izvodimo da je ukupna vrijednost piksela unutar pravokutnika D jednaka 4 + 1 – (2 + 3).



Slika 2.4. Računanje ukupne vrijednosti piksela unutar pravokutnika D[3]

AdaBoost

U originalnom radnom okviru Viola i Jones koriste potprozor veličine 24 x 24 piksela. Unutar svakog potprozora nalazimo više od 160.000 vrijednosti značajki koje možemo analizirati. Da bismo detektirali ljudsko lice, potrebno je analizirati samo nekoliko od mnoštva značajki. Odabir tih ciljanih značajki omogućuje nam AdaBoost algoritam.

AdaBoost (*Adaptive Boost*) je algoritam strojnog učenja pri kojem nalazimo najbolje od 160.000 mogućih značajki. Nakon pronalaska najvažnijih značajki, njihova ponderirana

[3] VIOLA, P., JONES, M. Local Rapid Object Detection using a Boosted Cascade of Simple Features. Conference on Computer Vision and Pattern Recognition, 2001.

kombinacija koristi se u odlučivanju sadrži li trenutni potprozor lice ili ne. U odlučivanje se uključuju samo one značajke koje su u pravu u više od 50 % slučajeva, makar taj postotak bio i vrlo malo iznad 50 % (npr. 51 %). Ove značajke još se nazivaju i slabi klasifikatori. AdaBoost koristi linearnu kombinaciju ovih slabih klasifikatora da bi stvorio jak klasifikator koji se potom koristi u odlučivanju nalazi li se u potprozoru lice. (1) „F(x)“ predstavlja jaki klasifikator, „fn(x)“ slabi klasifikator, a „αn“ predstavlja ponder pridružen slabom klasifikatoru. Pojednostavljeno, ponder predstavlja važnost koju je AdaBoost algoritam pridodao nekoj značajki. Što je značajka važnija u detekciji lica, to će pridruženi ponder biti veći. [4]

$$F(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \alpha_3 f_3(x) + \ldots + \alpha_n f_n(x) \quad (1)$$

Nakon analize pojedinog potprozora, kombinacija slabih klasifikatora unutar jakoga klasifikatora daje jedinstvenu izlaznu vrijednost. Ako ta izlazna vrijednost prelazi ranije dogovorenu vrijednost koju nazivamo vrijednosni prag (engl. *threshold*), možemo reći da se u trenutnom potprozoru nalazi lice.

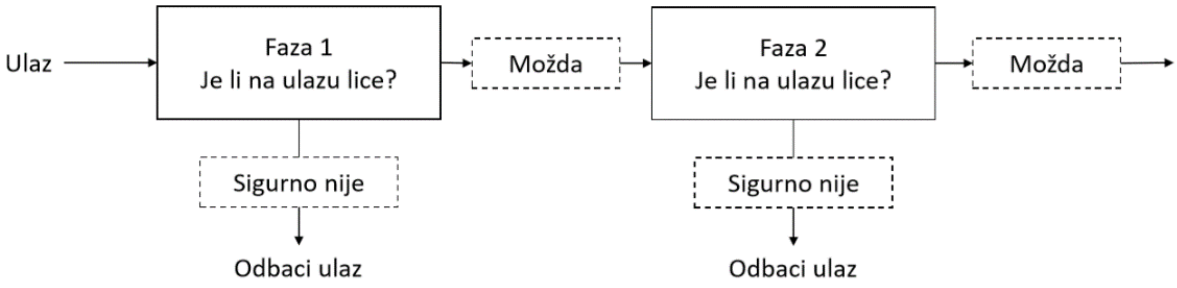
Iako AdaBoost eliminira većinu nepotrebnih značajki i ostavlja samo najznačajnije (u izvornom Viola-Jonesevu okviru radi se o 200 najvažnijih značajki), takav klasifikator je vrlo efikasna tehnika detekcije lica, no ne dovoljno efikasna za primjenu kod detekcije u realnom vremenu (za obradu jedne slike veličine 384 x 288 piksela trebalo je 0,7 sekundi). Također, prilikom povećanja broja evaluiranih značajki radi povećanja preciznosti detektora povećava se i vrijeme potrebno za obradu pojedine slike. Zbog toga Viola-Jones koristi kaskadu klasifikatora pri samom postupku detekcije.

Kaskada klasifiktora

Sliku analiziramo na način da pomičemo potprozor manje veličine preko slike sve dok nismo analizirali svaki dio slike i to činimo u nekoliko prolaza, svaki put tražeći lice veće ili manje veličine. Makar slika sadržavala i nekoliko lica, nedvojbeno je da se u njoj nalazi puno više negativnih potprozora (onih koji ne sadržavaju lice) od pozitivnih. Algoritam detekcije bi, dakle, trebao biti fokusiran na što brže odbacivanje potprozora za koje je poprilično siguran da ne sadrže lice. Zbog toga korištenje jednog velikog jakog klasifikatora nije pogodno za ovakav oblik analize jer je taj proces računalno preskup da bi ga se primijenilo na svaki potprozor unutar slike.

Da bi se ovaj problem ublažio, Viola i Jones koriste kaskadu klasifikatora gdje je svaka faza kaskade izvedena kao jedan jaki klasifikator. Sve važne značajke detekcije grupirane su u

nekoliko faza u kojima svaka od njih evaluira određeni broj značajki. Zadatak svake od tih faza je za određeni potprozor odrediti ili da se u njemu sigurno ne nalazi lice ili da se u njemu možda nalazi lice. (Slika 2.5.) Koristeći ovu tehniku, određeni potprozor biva odbačen ako ga se u bilo kojoj fazi kaskade proglasi negativnim, tj. odredi da ne sadrži lice te tako višestruko ubrzava obrada pojedine slike. Velika većina potprozora proglasit će se negativnima već prilikom prve faze kaskade što će koštati oko 60 mikroprocesorskih instrukcija ako koristimo klasifikator koji evaluira dvije značajke [3]. Tek kada određeni potprozor prođe sve faze kaskade klasifikatora proglašava ga se pozitivnim i detekcija lica u tom potprozoru je uspješna.



Slika 2.5. Kaskada klasifikatora

Prilikom treniranja kaskade klasifikatora moramo biti svjesni dva tipa kompromisa. U većini slučajeva klasifikatori koji sadrže više značajki ostvarit će veću točnost detekcije i manji broj lažno pozitivnih rezultata. Isto tako, za evaluaciju klasifikatora koji sadrže veći broj značajki potrebno je više računalnog vremena. Tako da prilikom optimizacije okvira za detekciju moramo naći balans između 3 faktora:

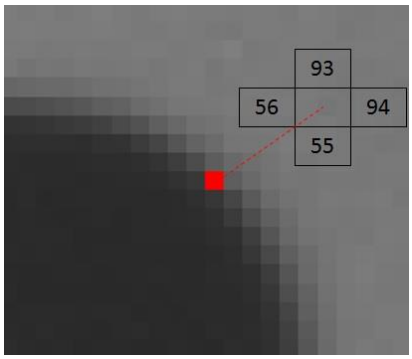
- broja faza kaskade klasifikatora
- broja značajki unutar svake faze
- praga vrijednosti svake od faza.

2.1.2. Histogram orijentiranih gradijenata

Detektori temeljeni na histogramima orijentiranih gradijenata (HOG detektori) postali su jedna od klasičnih metoda računalnog vida od objavljivanja znanstvenog rada N. Dalala i B. Triggsa 2005. godine. [6] Ova tehnika temelji se na prebrojavanju pojavljivanja orijentacija gradijenata u dijelovima slike predstavljene pomoću mreže ćelija.

Vektor gradijenata

Vektor gradijenata jedan je od temeljnih koncepata u području računalnog vida. On predstavlja promjenu u vrijednosti uzastopnih piksela gledajući prema x-osi i y-osi. Ako želimo izračunati vrijednost vektora gradijenta piksela prikazanog na slici (Slika 2.6.), moramo izračunati promjenu u vrijednostima po obje osi. Za izračun po x-osi uzet ćemo vrijednosti piksela desno i lijevo od željenog piksela i te dvije vrijednosti oduzeti. U ovom slučaju promjena po x-osi iznosi 38. Isto ćemo napraviti i za y-os te također dobiti vrijednost 38. U ovom slučaju vektor gradijenta označenog piksela je $\begin{bmatrix}38\\38\end{bmatrix}$.

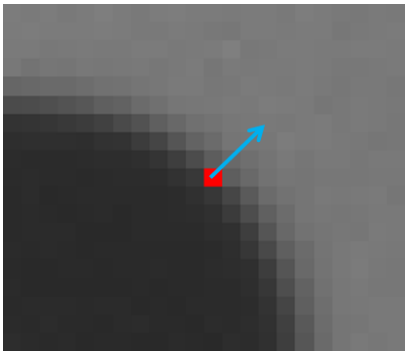


Slika 2.6. Vrijednosti dodirnih piksela po obje osi[4]

Nadalje, možemo izračunati veličinu (2) i smjer (3) vektora te ga takvog prikazati na samoj slici (Slika 2.7.). U našem slučaju veličina iznosi 53,74, a smjer 45°.

$$|\overrightarrow{XY}| = \sqrt{\Delta X^2 + \Delta Y^2} \tag{2}$$

$$\Theta = tan^{-1}\left(\frac{\Delta Y}{\Delta X}\right) \tag{3}$$



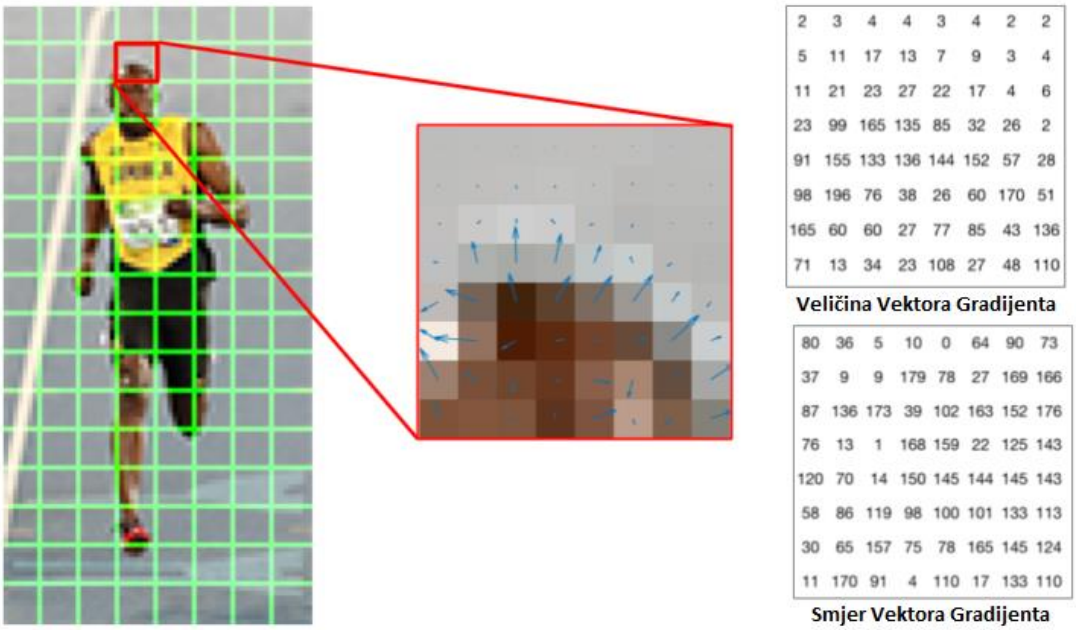
Slika 2.7. Prikaz veličina i smjera vektora gradijenta označenog piksela[5]

[4] Preuzeto 2. siječnja 2018. sa http://mccormickml.com/2013/05/07/gradient-vectors/
[5] Preuzeto 2. siječnja 2018. sa http://mccormickml.com/2013/05/07/gradient-vectors/

Jedna od glavnih primjena vektora gradijenta je u otkrivanju ruba objekata na slikama jer je vektor usmjeren prema samom rubu objekta, a sama veličina vektora govori nam o veličini promjene vrijednosti piksela koja je kod rubova objekata obično velika. [7]

Izračun histograma

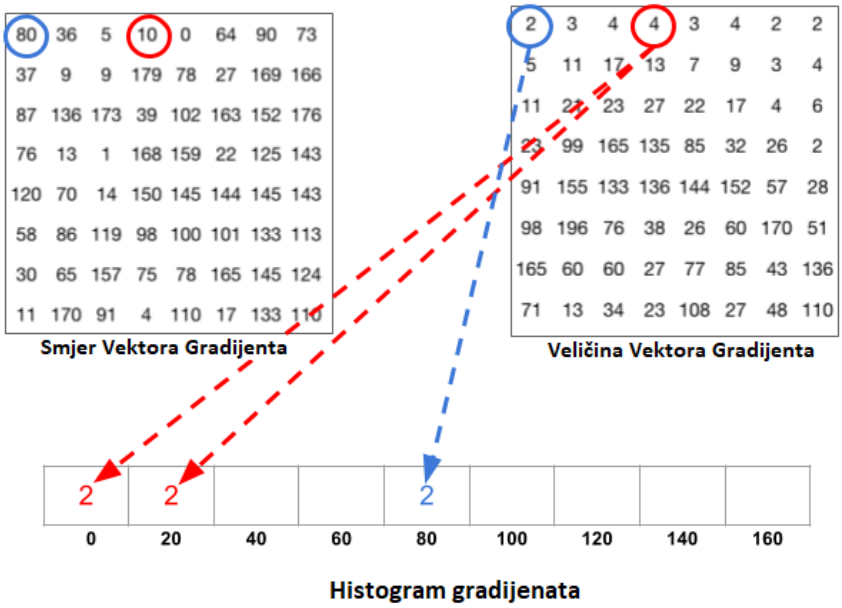
U originalnom radu Dalal i Triggs koriste ovu metodu za detekciju pješaka unutar slike te koriste prozor detekcije veličine 64 x 128 piksela [5]. Za svaki piksel unutar prozora detekcije potom se izračuna vektor gradijenta. Prozor se podijeli na mrežu u kojoj je svaka ćelija dimenzija 8 x 8 piksela. (Slika 2.8.)



Slika 2.8. Prozor detekcije podijeljen na mrežu ćelija te vrijednosti veličine i smjera vektora gradijenata piksela u jednoj ćeliji[6]

Nakon podjele, računa se histogram za svaku pojedinu ćeliju i to na način da se vrijednosti veličina vektora pojedinih piksela razvrstavaju u jedan od 9 razreda ovisno o smjeru vektora. Razredi odgovaraju kutevima 0, 20, 40, 60, 80, 100, 120, 140 i 160 stupnjeva. Ako je vrijednost smjera vektora između vrijednosti dva razreda, vrijednost njegove veličine dijeli se na oba razreda u omjeru ovisno o tome koliko je vrijednost smjera blizu vrijednosti razreda. (Slika 2.9.).

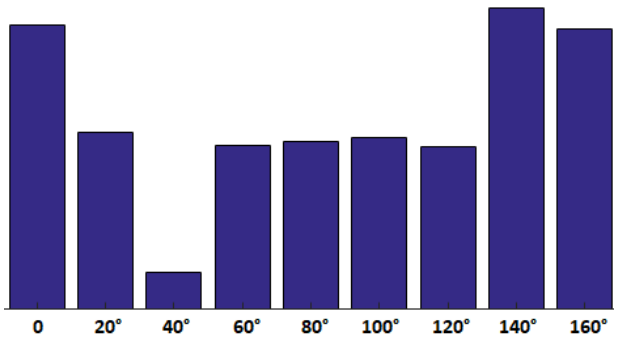
[6] Preuzeto 2. siječnja 2018. sa https://www.learnopencv.com/histogram-of-oriented-gradients/



Slika 2.9. Razvrstavanje vrijednosti veličine vektora piksela u razrede[7]

Nakon razvrstavanja vrijednosti u razrede u označenoj ćeliji dobijemo histogram orijentiranih gradijenata prikazan na slici (Slika 2.10.) iz kojeg je vidljivo da imamo veći broj orijentacija gradijenata vrijednosti bliže 0 i 180 stupnjeva što znači da su gradijenti usmjereni gore ili dolje što je i vidljivo na slici (Slika 2.8.).

Važno je napomenuti da se u ovoj implementaciji HOG detektora vrijednosti smjera gradijenata kreću u rasponu od 0 do 180 stupnjeva, tj. gradijenti suprotnih smjerova smatraju se gradijentima istog smjera. Takav oblik zapisa naziva se nepotpisanim (engl. *unsigned*) gradijentima. Razlog tome je što je empirijski dokazano da kod detekcije pješaka bolje rezultate donosi korištenje nepotpisanih gradijenata u odnosu na potpisane.



Slika 2.10. Histogram orijentiranih gradijenata odabrane ćelije[8]

[7] Preuzeto 2. siječnja 2018. sa https://www.learnopencv.com/histogram-of-oriented-gradients/
[8] Preuzeto 2. siječnja 2018. sa https://www.learnopencv.com/histogram-of-oriented-gradients/

Blok normalizacija

Proces normalizacije vrijednosti potreban je da bi se dodatno smanjila osjetljivost na promjene u osvjetljenju. Iako je sam vektor gradijenta otporan na linearne promjene u osvjetljenju (npr. svim pikselima slike povećamo vrijednost za 50), nije otporan na višestruko povećanje vrijednosti piksela (npr. sve vrijednosti piksela pomnožimo faktorom 2) jer se magnituda vektora također višestruko poveća. Rješenje ovog problema donosi euklidska ili L2 norma vektora u kojoj vrijednost svake dimenzije vektora podijelimo magnitudom tog istog vektora te dobijemo vrijednosti vektora neovisne o promjenama sveukupnog osvjetljenja.

Iako bismo mogli normalizirati svaki histogram posebno, bolje performanse dobiju se ako normaliziramo blokove ćelija. Dalal i Triggs grupiraju ćelije u blokove veličine 2 x 2 te pomiču potprozor izbora grupe ćelija uvijek za jednu ćeliju u potrebnom smjeru (Slika 2.11.). Ovim načinom osigurava se da se svaka ćelija pojavljuje više puta u finalnom deskriptoru, ali svaki put normalizirana različitim setom susjednih ćelija.

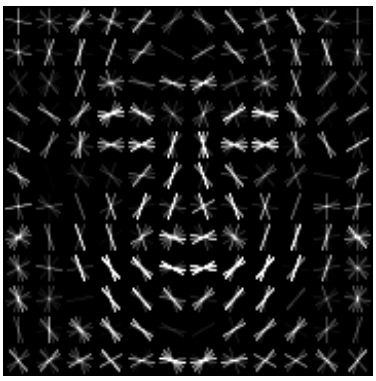
Slika 2.11. Pomicanje normalizacijskog bloka ćelija[9]

Finalni deskriptor dobiven analizom i normalizacijom bilo kojeg prozora detekcije veličine 64 x 128 piksela sadrži u sebi 3780 vrijednosti. Svaki prozor detekcije dijeli se u mrežu blokova veličine 7 x 15 što iznosi 105 različitih blokova ćelija. Svaki blok sadrži 4 ćelije od kojih svaka ćelija sadrži histogram od 9 vrijednosti, tj. u svakom bloku nalazi se 36 vrijednosti histograma orijentiranih gradijenata. Množenjem ovih vrijednosti dolazimo do ukupnog broja vrijednosti sadržanih u svakom finalnom deskriptoru prozora detekcije.

Osim za detekciju pješaka, za što su HOG detektore koristili Dallal i Triggs u svom radu, pokazalo se da ovi detektori mogu biti trenirani za detekciju bilo kakvih objekata. Jednu od implementacija nalazimo u biblioteci otvorenoga koda „dlib“ Davisa Kinga u kojoj je moguće trenirati HOG detektor na bilo kojem setu slika. Biblioteka omogućuje i

[9] Preuzeto 2. siječnja 2018. sa http://mccormickml.com/2013/05/09/hog-person-detector-tutorial/

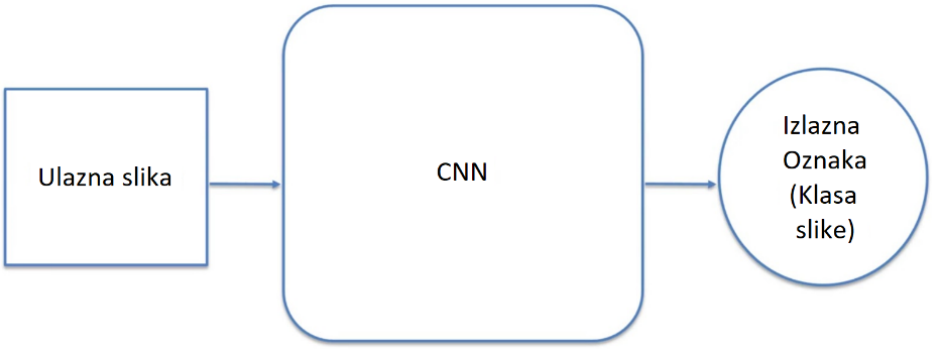
vizualizaciju samog detektora. Na slici (Slika 2.12.) vidimo detektor treniran na 18 slika ljudskog lica koji stvarno izgleda kao ljudsko lice. [6][8][9]



Slika 2.12 HOG detektor treniran za detekciju ljudskog lica[10]

2.1.3. Konvolucijske neuronske mreže

Konvolucijske neuronske mreže (CNN) su mehanizmi koji imaju zadaću da na svom izlazu daju klasifikaciju ulazne slike (navedu koje je vrste ulazna slika) (Slika 2.13.). Mogu se trenirati da na svom izlazu daju klasifikaciju bilo koje vrste objekta pa tako i za potrebe detekcije lica gdje ih treniramo da slike klasificiraju u samo dvije klase, prikazuje li ulazna slika lice ili ne. [10]



Slika 2.13. Zadaća konvolucijske neuronske mreže[11]

Konvolucijske neuronske mreže obično gradimo od nekoliko vrsta slojeva:

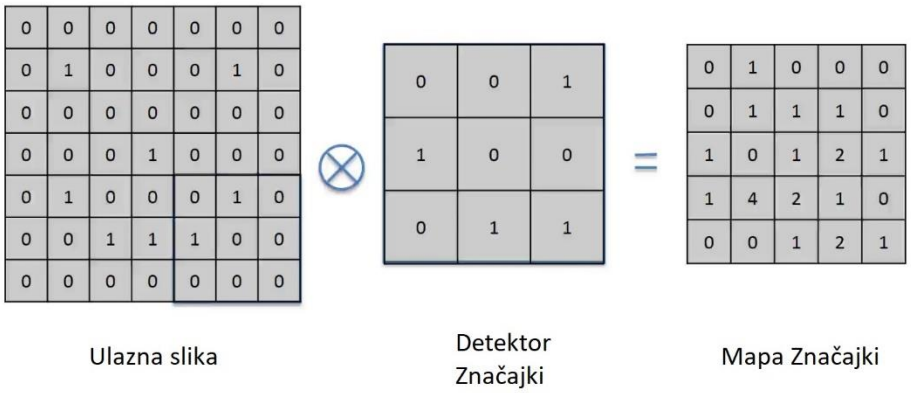
- konvolucijskog sloja
- sloja udruživanja (engl. *Pooling layer*)
- potpuno spojenog sloja (engl. *Fully-connected layer*).

[10] Preuzeto 2. siječnja 2018. sa http://blog.dlib.net/2014/02/dlib-186-released-make-your-own-object.html
[11] Preuzeto 5. siječnja 2018. sa https://www.udemy.com/machinelearning/learn/v4/t/lecture/6761138?start=0

Konvolucijski sloj

Konvolucijski sloj ime dobiva po operaciji konvolucije koja u konvolucijskim neuronskim mrežama služi za izdvajanje značajki unutar neke slike (Slika 2.14.).

Proces izdvajanja značajki unutar CNN-a počiva na korištenju „detektora značajki“ koji se još nazivaju i filteri ili jezgre CNN-a. Ulazna slika obrađuje se „povlačenjem“ detektora značajki preko nje te kombiniranjem vrijednosti piksela ulazne slike i detektora značajki te produciranjem izlazne „mape značajki“.

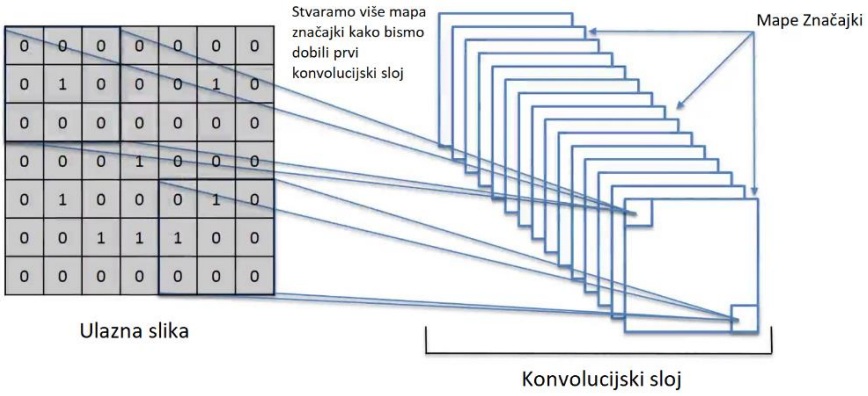


Slika 2.14. Izdvajanje značajki operacijom konvolucije u CNN-u[12]

Dimenzije mape značajki uvijek su manje veličine nego ulazna slika, a faktor smanjenja ovisi o više čimbenika kao što su veličina detektora značajki i korak (engl. *stride*) kojim povlačimo detektor preko slike. Mapa značajki je zapravo matrica vjerojatnosti pojave određene značajke unutar ulazne slike (što je vrijednost na mapi veća, to je vjerojatnost da se na tom mjestu nalazi značajka koju traži detektor). Također valja napomenuti da proces dobivanja mape značajki zadržava prostorne odnose ulazne slike što je važno svojstvo u detekciji objekata.

U jednom sloju konvolucije CNN-a proces izdvajanja značajki odvija se više puta, jednom za svaki detektor značajki pa je izlaz iz konvolucijskog sloja mnoštvo mapa značajki (Slika 2.15). Broj primijenjenih detektora naziva se dubina (engl. *depth*) te kažemo da je dubina konvolucijskog sloja jednaka broju detektora. [11]

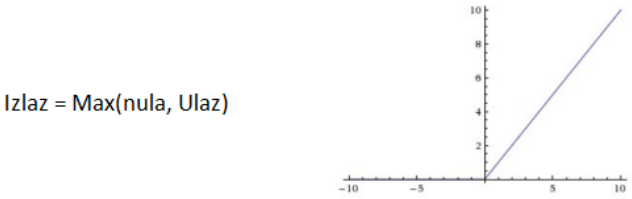
[12] Preuzeto 5. siječnja 2018. sa https://www.udemy.com/machinelearning/learn/v4/t/lecture/6761140?start=0



Slika 2.15. Stvaranje konvolucijskog sloja u CNN-u[13]

ReLU operacija

ReLU (engl. *Rectified Linear Unit*) operacija (Slika 2.16.) u CNN-u služi za „razbijanje" linearnosti. Konvolucija je linearna operacija jer se sastoji od množenja i zbrajanja matrica pa izlaz iz konvolucijskog sloja ima visok stupanj linearnosti između vrijednosti uzastopnih piksela. Ovo svojstvo nije dobro za prepoznavanje značajki objekata pa koristimo neku od funkcija za uvođenje nelinearnosti. Empirijski je dokazano da ReLU funkcija daje bolje rezultate u većini situacija. ReLU je operacija koja obrađuje svaki piksel te zamjenjuje negativne vrijednosti piksela mape značajki nulom. Primjer rezultata ReLU operacije vidljiv je na slici (Slika 2.17.). [12]

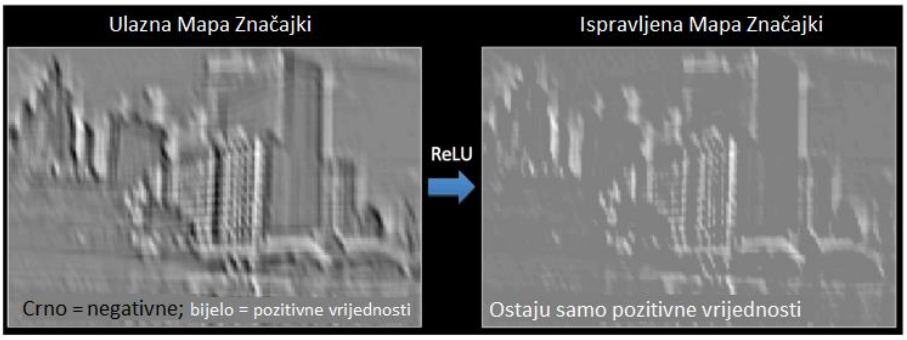


Slika 2.16. ReLU operacija[14]

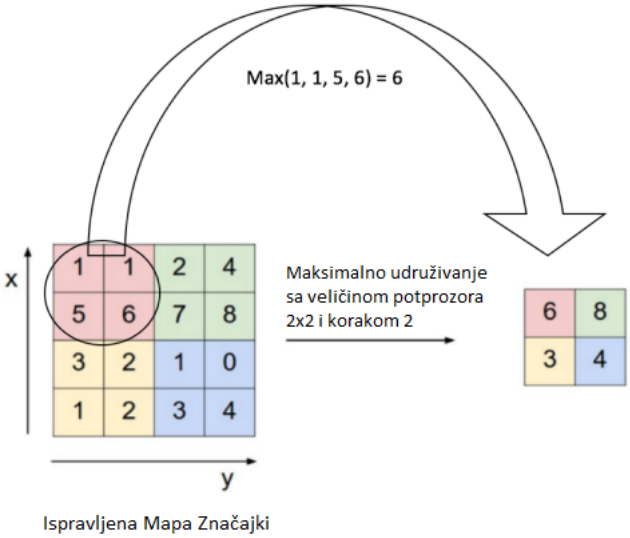
Sloj udruživanja

Svrha prostornog udruživanja (engl. s*patial pooling*), koje još nazivamo i poduzorkovanje (engl. *downsampling, subsampling*) je smanjiti dimenzije mapa značajki, a pritom očuvati najvažnije informacije koje značajku čine značajnom. Postoji više vrsta udruživanja: maksimalno, prosječno, sumirano itd. U CNN-ovima obično koristimo maksimalno udruživanje jer se same značajke očituju kao maksimalne vrijednosti procesa konvolucije te je bitno te vrijednosti očuvati. Proces udruživanja (Slika 2.18) obavlja se na način da se

[13] Preuzeto 5. siječnja 2018. sa https://www.udemy.com/machinelearning/learn/v4/t/lecture/6761140?start=0
[14] Preuzeto 5. siječnja 2018. sa https://www.udemy.com/machinelearning/learn/v4/t/lecture/6761142?start=0



Slika 2.17. Primjer rezultata ReLU operacije[15]



Slika 2.18. Proces maksimalnog udruživanja[16]

odredi veličina potprozora udruživanja te se taj prozor povlači preko mape značajki određenim korakom. Prilikom svakog pomaka uzimamo maksimalnu vrijednost piksela iz tog područja i kopiramo je u novu mapu koja služi kao izlaz sloja udruživanja. Rezultat je nova mapa smanjenih dimenzija, ali s očuvanim najvažnijim značajkama ulazne mape. Proces ponavljamo za svaku ulaznu mapu pa je broj izlaznih mapa sloja udruživanja jednak broju ulaznih mapa. Udruživanje je važno i za postizanje prostorno-translacijske invarijancije što predstavlja otpornost CNN-a na prostorna izobličenja kao što su rotacija, iskrivljenje i promjena relativne veličine ulazne slike. Npr. prilikom blage rotacije lica, pikseli koji predstavljaju značajke neće biti na potpuno istim mjestima već će biti pomaknuti u smjeru rotacije. Ako pak promatramo područja značajki (oči, vrh nosa, usne itd.) i ona će biti rotirana, no prilikom primjene procesa udruživanja na izlaznoj mapi maksimalne

[15] Preuzeto 5. siječnja 2018. sa https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
[16] Preuzeto 5. siječnja 2018. sa https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

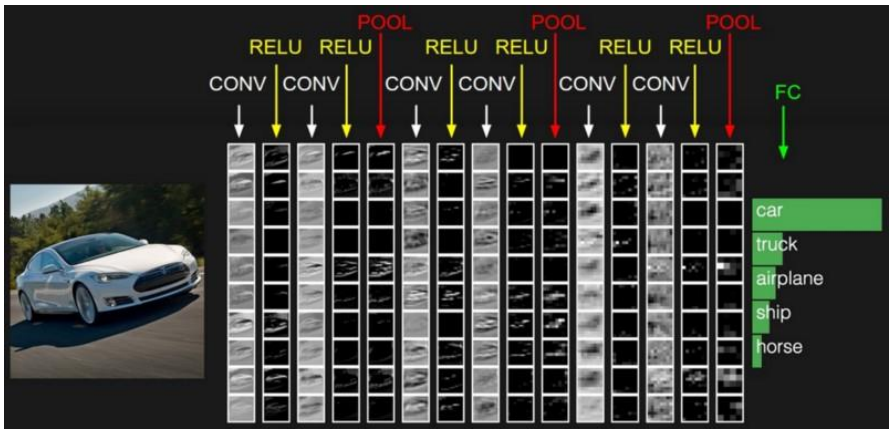
vrijednosti značajki pojavit će se na približno jednakim mjestima kao i kod lica koje nije rotirano. [13]

Još jedno važno svojstvo udruživanja je odbacivanje informacija. U većini slučajeva odbacivanje informacija smatramo lošom pojavom, ali kod CNN-ova ovo svojstvo sprječava pretreniranje (engl. *overfitting*) neuronske mreže. Pojava pretreniranja je situacija kada neuronska mreža predobro nauči podatke na kojima trenira i ne generalizira ih što dovodi do situacije da mreža radi s visokom točnošću na podacima na kojima je učila, ali ne obavlja posao dobro na novim, testnim podacima. Pretreniranje umanjujemo jer samim udruživanjem smanjujemo ukupan broj parametara koji ulaze u sljedeći sloj CNN-a, tj. ostavljamo samo bitne parametre.

Isto tako, smanjivanjem veličine mapa uvelike povećavamo brzinu rada same mreže. Na slici (Slika 2.18.) prikazan je proces udruživanja koristeći prozor veličine 2 x 2 i s veličinom koraka 2. Proces za svoj izlaz daje mapu koja je smanjena 75 % u odnosu na ulaznu mapu što predstavlja ogromno smanjenje podataka koje mreža obrađuje, a zadržavajući pritom značajke važne za točnost CNN-a. [13]

Potpuno spojeni sloj

Potpuno spojeni sloj CNN-a je sloj obične neuronske mreže dodan na cijeli lanac slojeva konvolucije i udruživanja. Vrlo je sličan skupu skrivenih slojeva neuronskih mreža, no razlika je u tome da su u potpuno spojenom sloju svi neuroni prethodnog sloja spojeni sa svim neuronima sljedećeg sloja, što kod običnih skrivenih slojeva ne mora biti tako. Svrha ovog sloja je spojiti značajke prethodnih slojeva i stvoriti dodatne atribute koji potom predviđaju klasu ulazne slike s povećanom točnošću. Važno je napomenuti da i bez potpuno spojenog sloja već imamo velik broj značajki koje bi mogle poprilično dobro predvidjeti klasu ulazne slike, no kako je neuronska mreža struktura dizajnirana sa svrhom da uzima značajke, od njih stvara dodatne atribute i samokorektivno odabire koji od tih atributa su presudni za povećanje točnosti predikcije, njezinim korištenjem uvelike povećavamo željeni učinak detekcije. [14]



Slika 2.19. CNN sastavljen od više uzastopnih slojeva[17]

CNN-ovi koji se koriste za detekciju objekata grade se od više uzastopnih slojeva te performanse samog CNN-a ovise o odabranoj kombinaciji slojeva (Slika 2.19.). Valja napomenuti da treniranje CNN-ova traje višestruko duže nego treniranje modela drugih tehnika. Ove stvari treba uzeti u obzir kod izrade sustava koji obradu slika mora obavljati u realnom vremenu. [15]

2.2. Komercijalne aplikacije za detekciju lica

Za razliku od biblioteka otvorenoga koda, komercijalne aplikacije za detekciju lica dolaze u obliku SDK (engl. *Software Development Kit*), koji omogućuje jednostavnije korištenje i laku integraciju u vlastite aplikacije i sustave. U ovim proizvodima, detekcija lica obično je samo mali dio mogućnosti koje proizvod nudi. Većina ih uz detekciju lica nudi i mogućnosti detekcije ključnih točaka lica, prepoznavanje lica, praćenje lica u videotijeku, detekciju raspoloženja osobe, detekciju spola i starosti i još mnoštvo mogućnosti. Također, proizvodi su popraćeni detaljnom dokumentacijom i primjerima korištenja što uvelike olakšava izgradnju vlastitih aplikacija.

2.2.1. Acsys FRS

Acrys Biometrics, uz gotove proizvode nudi i SDK za razvoj vlastitih sigurnosnih aplikacija koji omogućuje kontrole za praćenje, upis, verifikaciju, klasifikaciju, rad s bazom podataka,

[17] Preuzeto 5. siječnja 2018. sa https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

komunikaciju i rad s multimedijom. Kao ključne kvalitete svojih proizvoda kojima se razlikuju od ostalih biometrijskih tehnologija ističu brzinu, preciznost i inteligenciju svojih sustava. [19]

2.2.2. Betaface Face Detection and Recognition

Betaface razvijateljima sustava za detekciju prepoznavanje i analizu lica nudi SDK za operativne sustave Windows i Linux. Podržana je analiza učestalih slikovnih i videodatoteka kao i analiza videotijekova. Za razvoj se može koristiti C++ i C# programski jezik, a dostupno je i mnoštvo primjera korištenja te cijeli projekti s podrškom za videomaterijale i pripremljenom bazom lica osoba.

Kao neke od svojih klijenata Betaface ističe: 20th Century Fox, BAE Systems, BETC (Evian), B-Reel, Canon, Disney, Warsaw University of Technology i mnoge druge. [20]

2.2.3. Neurotechnology VeriLook

VeriLook tehnologija tvrtke Neurotechnology namijenjena je za razvijatelje i integratore biometrijskih sustava. Proizvođač navodi visok stupanj pouzdanosti i performansi kod detekcije lica, istodobnog prepoznavanja više lica te sparivanja lica u „1 na 1“ i „1 na više“ modovima rada kao odliku VeriLook tehnologije. Za razvoj vlastitih aplikacija na VeriLook tehnologiji dostupan je SDK i to za platforme Microsoft Windows, Mac OS X, iOS i Android. Neurotechnology svoja rješenja temelji na dubokim neuronskim mrežama.

Sustavi temeljeni na Neurotechnology rješenjima koriste se za registraciju birača na biračkim mjestima u Bangladešu, Venezueli, Sierra Leoneu i Keniji te u mnogim drugim zemljama kao dio biometrijskih sustava identifikacije osoba. [21]

2.2.4. Luxand FaceSDK

Luxand FaceSDK koristi se u izgradnji 32-bitnih i 64-bitnih aplikacija koje koriste funkcionalnosti prepoznavanja lica i identifikacije korisnika na temelju njegova lica u programskim jezicima Microsoft Visual C++, C#, Objective C, VB, Java i Delphi. FaceSDK koristi se u stotinama aplikacija za identifikaciju i autentifikaciju korisnika *web*-kamerom, pretragu podudarajućih lica u bazama podataka i na fotografijama, detekciju lica na statičnim slikama i u videu u realnom vremenu i slično.

Kao glavne klijente Luxand ističe: Universal Studios, Samsung, Boeing, Ford, LG, Badoo, Citrix, P&G, Danone, Diebold i FIS. [22]

2.3. Programski okviri za detekciju lica otvorenoga koda

Programski okviri za detekciju lica otvorenoga koda obično su programski okviri koji se bave problemima računalnog vida općenito. Detekcija ljudskog lica samo je jedna od mogućnosti koje pružaju. Također, detekcija ljudskog lica poseban je slučaj detekcije objekata pa je pomoću većine biblioteka moguće izraditi detektore za bilo koju vrstu objekata.

2.3.1. Open CV

„Biblioteka OpenCV (Open Source Computer Vision Library) je biblioteka otvorenog koda koja se bavi problemom računalnog vida i strojnog učenja. OpenCV je izgrađen da bi pružio zajedničku infrastrukturu za primjenu računalnog vida i za ubrzanje korištenja strojnog vida u komercijanim proizvodima. Biblioteka se nalazi pod BSD licencom što omogućava kompanijama da na jednostavan način koriste i mijenjaju njezin kod.“ [16]

Biblioteka sadrži više od 2500 optimiziranih algoritama, uključujući klasične i nove, moderne algoritme strojnog učenja i računalnog vida. Algoritmi imaju širok raspon primjena kao što su detekcija i prepoznavanje ljudskog lica, identifikacija objekata, klasifikacija ljudskih radnji u videu, praćenje pokreta kamere, praćenje objekata u pokretu, stvaranje 3D modela objekata, spajanje više slika radi dobivanja slike cijele scene u višoj rezoluciji (panoramske snimke), pronalazak sličnih objekata u slikama, uklanjanje crvenih očiju stvorenih korištenjem blica na fotografijama, praćenje pokreta očiju, prepoznavanje okoline i postavljanje markera radi preklapanja okoliša s objektima obogaćene stvarnosti (engl. *augmented reality*).

Biblioteka ima C++, C , Python, Java i MATHLAB sučelja i podržava operacijske sustave Windows, Linux, Android i Mac OS. Također iskorištava MMX i SSE instrukcijske setove kada su oni dostupni. [16]

2.3.2. Dlib

„Dlib je moderan C++ set alata koji sadrži algoritme strojnog učenja i alate za stvaranje kompleksnog softvera u C++ programskom jeziku radi rješavanja problema u stvarnom

svijetu. Koristi se i u poslovne i u akademske svrhe i to u velikom rasponu područja uključujući robotiku, ugrađene uređaje, mobilne telefone i u računalnim okruženjima visokog učinka. Dlibova licenca otvorenog koda omogućuje korištenje za bilo koju primjenu bez naknade.“ [17]

Za razliku od mnogo projekata otvorenoga koda, biblioteka je popraćena vrlo kvalitetnom dokumentacijom koja pokriva sve klase i sve funkcije. Također nudi mnoštvo primjera korištenja klasa i funkcija. Biblioteka nudi prenosiv kod visoke kvalitete redovno testiran na operativnim sustavima MS Windows, Linux i Mac OS X. Ne zahtijeva bilo kakve druge softverske pakete za rad te ne zahtijeva postupak instalacije ili konfiguracije prije korištenja.

Autor biblioteke je Davis King koji redovito objavljuje nove mogućnosti i primjere primjene biblioteke s detaljnim rezultatima na *web*-stranicama službenog dlib bloga. [17]

2.3.3. CCV

CCV je nastao, kako autor biblioteke navodi, „zbog frustracije sa bibliotekom računalnog vida koju je tada koristio“ te je CCV bio namijenjen „lakšem razvoju te jednostavnijem organiziranju koda sa dozom opreza u vezi zavisnosti s ostalim bibliotekama“. Jednostavnost i minimalistička priroda biblioteke olakšale su mu i pojednostavile integraciju u bilo koje poslužiteljsko okruženje.

Autor biblioteke navodi najvažnije značajke biblioteke:

- prijenosan i ugradiv kod
- moderni algoritmi računalnog vida
- čisto sučelje za predprocesiranje predmemorije slika [18].

3. Arhitektura programskog rješenja za detekciju lica

Programsko rješenje sustava detekcije ljudskog lica u realnom vremenu zahtijeva takav tip arhitekture koji omogućuje maksimalnu optimizaciju brzine izvođenja, zadržavajući pritom zadovoljavajuću razinu detekcije. Arhitektura mora omogućiti korisniku podešavanje parametara sustava s obzirom na udaljenost na kojoj želi detektirati lica jer taj faktor direktno utječe na brzinu izvođenja programskog rješenja.

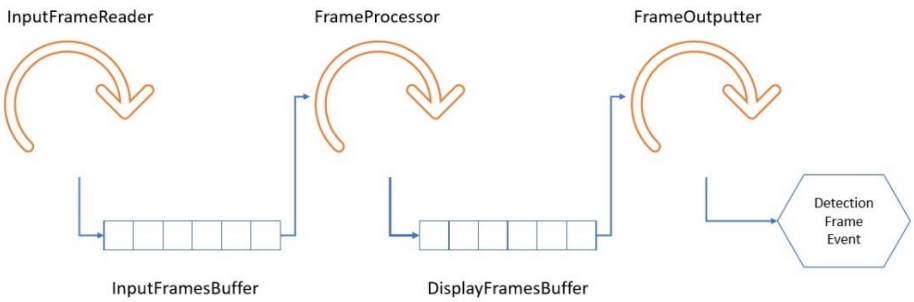
3.1. Projektni zadatak i opis funkcionalnosti sustava

Ideja projektnog zadatka je izraditi modul sustava koji bi bio u mogućnosti sa zadovoljavajućom točnošću detektirati ljudska lica unutar danog videotijeka. U izradi modula koristit će se samo biblioteke otvorenoga koda te istražiti mogućnost njihove dostatnosti u izradi kvalitetnog sustava detekcije koji bi imao primjenu u stvarnom svijetu. Ovaj je modul zamišljen kao prva faza u izgradnji sustava koji bi se primjenjivao u svrhu automatskog ažuriranja dolaznosti studenata na predavanja i vježbe. Neke od ostalih primjena ovakvog sustava bile bi praćenje kretanja osoba kroz motreni prostor, analiza navika kupaca u supermarketima, detekcija prisutnosti osoba u sustavima pametnih kuća te njihovo smještanje u određeni prostor. Ovaj modul realizirat će se kao zasebna konzolna aplikacija koja će kao svoj ulaz dobiti videotijek, bilo kao videotijek direktno iz kamere ili videomaterijal, a kao izlaz će dati slijed sličica s popratnim informacijama o detektiranim licima i njihovoj poziciji na sličici. Takav izlaz mogao bi se koristiti kao ulaz u neku od ostalih faza kompletnog sustava kao što je modul za prepoznavanje lica. Kako je prvobitna svrha ovog sustava detekcija prisutnosti studenata u učionici, obavljat će se optimizacija sustava prilagođena tom scenariju primjene. Testiranje sustava bit će obavljeno na predsnimljenom videomaterijalu iz predavaonice gdje će ispitivati sposobnost detekcije s obzirom na udaljenost osoba od kamere. Parametri koji će se koristiti za ocjenjivanje uspješnosti sustava su broj detekcija lica na pojedinoj sličici videotijeka te vrijeme potrebno za izvršenje detekcije. Radit će se analiza detekcije lica temeljena na dvije vrste detektora: HOG detektor gdje će se analizirati rezultati dobiveni izvršavanjem aplikacije na dva računala različite procesorske snage i detektor temeljen na konvolucijskim neuronskim mrežama koji koristi GPU za izvršavanje. Sustav je realiziran u višedretvenom okruženju

gdje svaka dretva ima svoju zadaću. Izvedba je klasičan *producer/consumer* sustav gdje je pojedina dretva proizvođač za sljedeću dretvu koja je potrošač podataka. Prva dretva je zadužena za dohvat podataka iz videotijeka i njegovu pripremu za konzumaciju od strane dretve zadužene za obradu sličice. Nakon što dretva zadužena za obradu sličice napravi detekciju, prosljeđuje rezultat dretvi za izlaz koja podiže događaje (engl. *event)* za svaku pojedinu obrađenu sličicu. Sustav je izveden na ovakav način da bi se mogle ostvariti pojedine optimizacije koje omogućuju izvršavanje detekcije u realnom vremenu. Optimizacije se sastoje u tome da se sama detekcija ne obavlja na svakoj sličici videotijeka već sustav parametrizira korisnik atributom *FrameSkipCount*, koji govori koliko će sličica biti preskočeno između svake detekcije. Razlog ovome je činjenica da detekcija lica uzima puno procesorskog vremena koje se povećava s udaljenosti od kamere na kojoj pokušavamo detektirati lice. Ta maksimalna udaljenost detekcije može se parametrizirati atributom *ImageScaleFactor* prema kojem će sustav povećati ili smanjiti ulaznu sličicu za zadani faktor. Realno, nije potrebno raditi detekciju na svakoj sličici, pogotovo prilikom primjene sustava u situacijama gdje se pozicija lica ne mijenja često, što je slučaj sa studentima u predavaonici. Međutim, potrebno je detektirati vrlo mala lica na sličici koja korespondiraju s licima studenata koji sjede u posljednjim redovima učionice pa faktor skaliranja slike mora biti velik. Potrebno je naći balans između ovih parametara kako bi se pružilo ugodno korisničko iskustvo te zadovoljavajući stupanj detekcije.

U sličicama na kojima se ne izvodi detekcija izvodi se praćenje prethodno detektiranih lica koristeći implementaciju korelacijskog *trackera* (engl. *correlation tracker*). Prilikom svake detekcije lica, za svako lice se nanovo stvara novi korelacijski *tracker* koji pokušava pratiti zadano lice u sljedećim sličicama sve dok „faktor korelacije“ ne postane premali ili sustav pokrene detekciju lica na nekoj daljnjoj sličici. Sam proces korelacijskog *trackinga* je manje zahtjevan od procesa detekcije, a pokazao se kao pouzdan način označavanja pozicije lica u sličicama u kojima se ne obavlja detekcija te uvelike ubrzava cijeli proces, a da pri tome daje dojam da se detekcija obavlja na svakoj sličici videotijeka te povećava ukupno korisničko iskustvo rada sustava.

Za međudretvenu komunikaciju koristi se implementacija dretveno sigurne (engl. *thread safe*) programske strukture „red“ otvorenoga koda prilagođene radu s jednim proizvođačem i jednim potrošačem izvedene u obliku međuspremnika u koje spremamo sličice za sljedeću fazu obrade. Prikaz izvedbe procesa detekcije vidljiv je na slici (Slika 3.1.).



Slika 3.1. Prikaz izvedbe procesa detekcije

3.2. Opis odabranog programskog okvira

Programsko rješenje bit će, najvećim djelom, realizirano korištenjem biblioteke otvorenoga koda dlib. Koristit će se i određeni dijelovi programskog okvira OpenCV, ali samo u segmentima vezanim za dohvaćanje videotijeka s kamere ili učitavanjem videotijeka iz datoteke. Razlog tome je to što dlib nema adekvatnu podršku za rad s videotijekovima, ali ima podršku za rad s formatima podataka koje koristi OpenCV. OpenCV ima punu podršku za rad s videotijekovima te jednostavnim pozivom funkcije vraća svaku sljedeću sličicu videomaterijala u svom vlastitom tipu podatka, bio videoulaz *web*-kamera ili datoteka na disku računala. Konverzija iz tipa podatka koju vraća OpenCV u tip podatka s kojim radi dlib vrlo je jednostavna i optimizirana pa ovakvo rješenje nudi lak način za realizaciju rada s videotijekovima bez mnogo dodatnog trošenja resursa. Za samu izvedbu detekcije dlib biblioteka je odabrana prvobitno zbog svoje opširne dokumentacije i mnoštva primjera koji uvelike olakšavaju rad. Nudi podršku za rad s detektorima temeljenim na HOG tehnologiji kao i s detektorima temeljenim na konvolucijskim neuronskim mrežama.

3.3. Izvedba sustava za detekciju lica u odabranom programskom okviru

Sustav detekcije lica izveden je u programskom jeziku C++ te je funkcionalnost zatvorena u klasu *VideoStreamFaceDetector* koju njezin korisnik može parametrizirati. Nakon pokretanja detekcije sustav vraća rezultate detekcije u obliku događaja koje je korisnik klase dužan obraditi na način na koji mu odgovara. Korisnik se mora pretplatiti na događaj ako želi obrađivati rezultate detekcije. Pretplata se obavlja pozivom metode `RegisterDetectionFrameListener` kojoj se kao parametar predaje funkcija koja

će biti pozvana za svaku sličicu koja izađe iz sustava detekcije. Kao parametre svakog događaja korisnik dobiva sličicu na kojoj se izvršila detekcija, listu detekcija lica na samoj sličici u obliku koordinata pravokutnika koji govori gdje se na sličici nalazi lice, listu praćenih lica iz prethodne sličice, ako praćena lica postoje te ostale razne informacije za testiranje poput vremena obrade pojedine sličice koje mogu služiti za testiranje performansi sustava. Ove informacije zapakirane su u jedinstvenoj klasi naziva `DetectionResult` (Kod 3.1).

```
class DetectionResult   {
public:
      cv::Mat image;
      std::vector<dlib::rectangle> detections;
      std::vector<dlib::correlation_tracker>
tracked_faces;
      DetectionResultDebugInfo debugInfo;
      bool hasDetections;
      bool hasTrackedFaces;
      DetectionResult();
      ~DetectionResult();};
```

Kod 3.1. Klasa DetectionResult koja sadrži rezultate detekcije pojedine sličice

Sustav također ima mogućnost slanja poruka korisniku kroz mehanizam *event/listener*. Pozivom metode `RegisterMessageListener` i predavanjem funkcije koja će obrađivati događaj korisnik se pretplaćuje na sustav poruka koje se šalju te ih je slobodan obraditi na način koji smatra prikladnim.

Korištenje sustava detekcije prikazano je sljedećim izvatkom koda (Kod 3.2.) gdje inicijaliziramo detektor, predamo mu izvor videotijeka (u ovom slučaju prethodno spremljen u varijablu `VIDEO_FILE_NAME`), pretplatimo se na događaje detekcije i prosljeđivanja poruka te parametriziramo detektor pozivima metoda `SetDetectionFrameSkipCount` i `SetImageScaleFactor`. Parametri su spremljeni u varijablama `frameSkipCount` i `imageScaleFactor` koje možemo prethodno tražiti od korisnika ili samo odrediti prema situaciji u kojoj se koristi detektor.

```
faceTools::VideoStreamFaceDetector detector;
detector.SetVideoSource(VIDEO_FILE_NAME);
detector.RegisterDetectionFrameListener(&showFrame);
detector.RegisterMessageListener(&onMessage);
detector.SetDetectionFrameSkipCount(frameSkipCount);
```

```
detector.SetImageScaleFactor(imageScaleFactor);
detector.StartDetection();
```

Kod 3.2. Korištenje sustava detekcije lica

Svojstvo koje je također moguće parametrizirati je i željeni broj sličica u sekundi na izlazu detektora. Svojstvo se određuje pozivanjem metode `SetDesiredFrameRate` i predavanjem cjelobrojne vrijednosti koja predstavlja željeni, tj. maksimalni broj sličica u sekundi koje će detektor isporučivati.

Metoda koja pokreće detekciju (Kod 3.3.) jednostavna je i sastoji se od toga da izračuna vrijeme koliko dretve moraju čekati da bi se ostvario željeni broj sličica u sekundi te pokretanja dretvi koje dohvaćaju sličice, njihovo procesiranje i njihovu predaju na izlaz detektora. Nakon obrade cijelog videotijeka, sustav obavještava korisnike porukom koja upućuje da je obrada videotijeka završena.

```
void faceTools::VideoStreamFaceDetector::StartDetection() {
	_sleepTimeBetweenFrames = 1000 / _desiredFrameRate;
	std::thread inputFrameReader(
&faceTools::VideoStreamFaceDetector::InputFrameReader, this
	);
	std::thread frameProcessor(
&faceTools::VideoStreamFaceDetector::FrameProcessor, this
	);
	std::thread frameOutputter(
&faceTools::VideoStreamFaceDetector::FrameOutputter, this
	);
	inputFrameReader.join();
	frameProcessor.join();
	frameOutputter.join();
	_fireMessage("Video stream finished.");
}
```

Kod 3.3. Metoda pokretanja detekcije

Svaka od metoda koja obavlja jednu od faza detekcije vrti se u `while` petlji koja ima dva uvjeta zaustavljanja. Jedan je eksplicitno zaustavljanje detekcije postavljanjem varijable `_detectionStopped` na vrijednost `true`. Drugi uvjet je završetak obrade materijala i događa se implicitno, tj. kada sustav prepozna da više nema sličica za obradu. Svaka od metoda mjeri vrijeme svojeg izvršavanja koristeći `chrono` bibiloteku koja je dio standardne biblioteke predložaka koja dolazi ugrađena u sam C++ programski jezik. Vrijeme se mjeri

na način da se zapamti trenutak u vremenu kada je izvršavanje metode započelo i trenutak kada je izvršavanje metode završilo. To vrijeme upotrebljava se za analizu brzine izvođenja samih metoda ili kao temelj za izračun vremena čekanja metode da bi se ostvarilo izvršavanje u željenom broju sličica u sekundi. Ovakav način rada je ključan za ispravan rad sustava iz dva razloga. Prvi je taj što se metoda za dohvaćanje sličica izvršava višestruko brže od metode za obradu te vjerojatno brže i od željenog broja sličica u sekundi na izlazu te dolazi do bržeg punjenja ulaznog međuspremnika nego što ga je moguće isprazniti te rezultira odbacivanjem sličica ulaznog tijeka ako se međuspremnik napuni do maksimalnog kapaciteta. Drugi razlog je postizanje željenog broja sličica u sekundi na izlazu iz sustava. Zbog korištenja izmjeničnih metoda detekcije i praćenja lica dolazi do nesrazmjera u vremenu obrade sličica i njihova spremanja u izlazni međuspremnik te je vrijeme između izlaza pojedinih sličica potrebno stabilizirati ovisno o broju željenih sličica u sekundi.

Metoda za dohvaćanje sličica s videoulaza obavlja dvije funkcionalnosti. Pokušava dohvatiti sličicu s videoulaza te ako je dohvaćanje uspješno sprema je u međuspremnik za daljnju obradu. Ako dohvaćanje nije uspješno, postavlja zastavicu da je čitanje videoulaza završeno (Kod 3.4.). Dohvaćamo koristeći `VideoCapture` iz radnog okvira OpenCV koji kao svoj ulaz prima varijablu u koju sprema dohvaćenu sličicu, a kao izlaz vraća `boolean` vrijednost koja označava je li dohvaćanje bilo uspješno ili nije. Ulazna varijabla je tipa podatka `Mat` koji predstavlja spremnik za bilo kakvu vrstu slikovnog materijala u OpenCV-u.

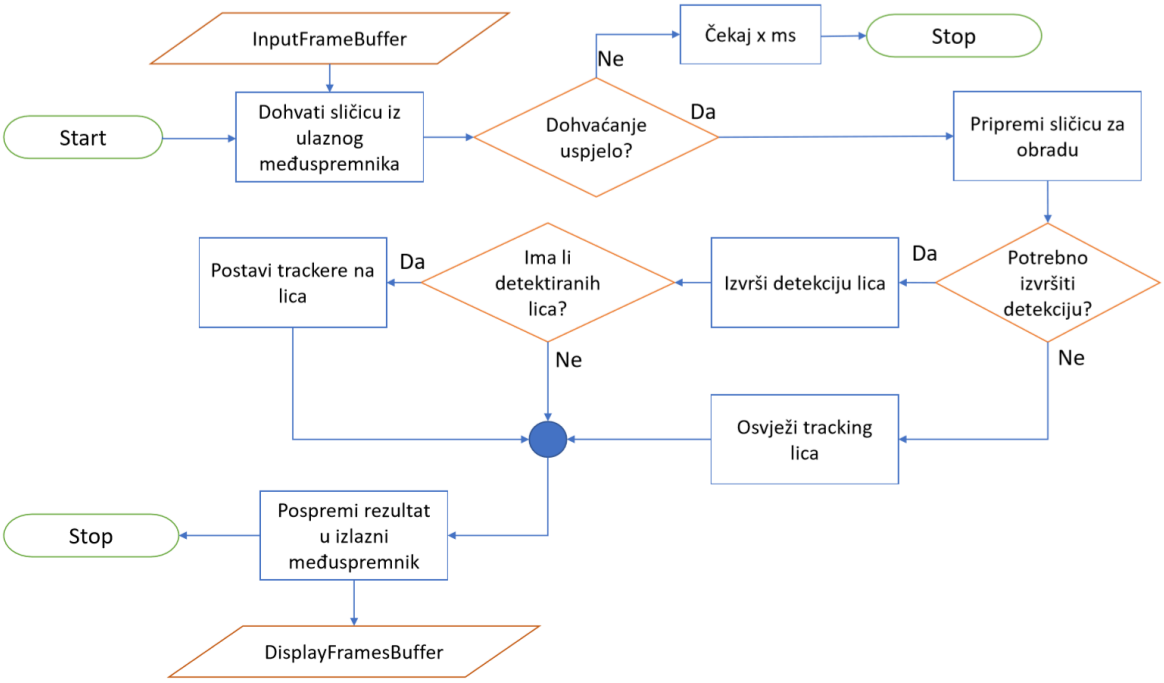
```
while (!_detectionStopped && !_inputDone) {
            cv::Mat temp;
            if (!cap.read(temp)){
                  _inputDone = true;
                  break;
            }
            bool success =
_inputFramesBuffer.try_enqueue(temp);
            }
```

Kod 3.4. Metoda za dohvaćanje sličica s videoulaza

Metoda za obradu sličica je skup koraka koji su potrebni da bi se što uspješnije obavio proces detekcije lica u zadanoj sličici (Slika 3.2.).

Nakon pokretanja, metoda pokušava dohvatiti sljedeću sličicu iz ulaznog međuspremnika. Ako dohvaćanje nije uspjelo, metoda čeka određeno vrijeme. Ako je dohvaćanje uspjelo, sličica se priprema za obradu. Priprema se sastoji od pretvorbe sličice u boji u sliku sivih

tonova (engl. *grayscale*), skaliranja veličine sličice za željeni faktor te pripreme dodatnih varijabli za potrebe procesa detekcije ili praćenja lica (Kod 3.5).



Slika 3.2. Algoritam metode obrade sličice

```
        cv::Mat temp_grey;
        cv::cvtColor(temp, temp_grey, CV_BGR2GRAY);
        cv::Mat temp_resized;
        cv::resize(temp_grey, temp_resized, cv::Size(), 1.0 /
_imageScaleFactor, 1.0 / _imageScaleFactor);

        dlib::cv_image<unsigned char> trackImg(temp_grey);
        dlib::cv_image<unsigned char> cnnTemp(temp_grey);
```

Kod 3.5. Priprema sličice za daljnju obradu

Ako se na trenutnoj sličici izvršava proces detekcije, radi se konverzija pripremljene sličice u format pogodan za detekciju, tj. u tip podatka biblioteke *Dlib*. Konverzija se obavlja predavanjem instance klase `Mat` OpenCV-a konstruktoru klase `cv_image` (Kod 3.6.).

```
if (++_imageCount % _detectionFrameSkipCount == 0){
            detections.clear();
            trackers.clear();

            dlib::cv_image<unsigned char> cimg(temp_resized);
```

```
dlib::matrix<dlib::rgb_pixel> matrix;
std::vector<dlib::mmod_rect> netDetections;
```

Kod 3.6. Konverzija sličice u format pogodan za detekciju

Ovisno o odabranom tipu detekcije izvršava se sama detekcija i rezultati spremaju u listu rezultata tipa `DetectionResult`. U biblioteci dlib detekcija se obavlja pozivom jedne metode na instanci klase koja obavlja detekciju. Prije poziva metode detekcije klase potrebno je instancirati što se u slučaju HOG detektora svodi na poziv metode koja vraća detektor lica (Kod 3.7.). Ovo je moguće iz razloga što dlib dolazi s predtreniranim detektorom za detekciju ljudskog lica. Ako bismo željeli napraviti svoj detektor, proces instanciranja detektora bio bi nešto složeniji.

```
dlib::frontal_face_detector detector =
dlib::get_frontal_face_detector();
```

Kod 3.7. Dohvaćanje detektora lica u biblioteci Dlib

U slučaju detektora temeljenog na CNN-u instanciranje je složenije jer moramo definirati neuronsku mrežu koja će obavljati detekciju. Za definiranje mreže koristimo C++ *templateing* (Kod 3.8.).

```
template <long num_filters, typename SUBNET> using con5d =
dlib::con<num_filters, 5, 5, 2, 2, SUBNET>;
template <long num_filters, typename SUBNET> using con5 =
dlib::con<num_filters, 5, 5, 1, 1, SUBNET>;

template <typename SUBNET> using downsampler =
dlib::relu<dlib::affine<con5d<32,
dlib::relu<dlib::affine<con5d<32,
dlib::relu<dlib::affine<con5d<16, SUBNET>>>>>>>>;
template <typename SUBNET> using rcon5 =
dlib::relu<dlib::affine<con5<45, SUBNET>>>;

using net_type = dlib::loss_mmod<dlib::con<1, 9, 9, 1, 1,
rcon5<rcon5<rcon5<downsampler<dlib::input_rgb_image_pyramid<d
lib::pyramid_down<3>>>>>>>>;
```

Kod 3.8. Definiranje CNN-a

Prvo definiramo 4 pomoćne strukture; `con5d` koja predstavlja konvolucijski sloj koji ujedno obavlja i *downsampling* i radi se na grupama od 5 x 5 piksela s korakom pomaka od 2 piksela; `con5` koji predstavlja konvolucijski sloj koji ne radi *downsampling* i radi se na grupama od 5 x 5 piksela s pomakom od 1 piksela, `downsampler` koji je struktura složena od nekoliko

cjelina `con5d` slojeva nad kojima se gradi vrsta *fully connected* sloja i sve se obrađuje ReLU slojem te `rcon5` koji predstavlja `con5` strukturu nad kojom je izgrađen potpuno spojeni sloj i obrađen ReLU slojem. [23]

Nakon definiranja pomoćnih struktura definiramo CNN koji se sastoji od ulaznog sloja koji očekuje sliku u boji koja se prosljeđuje u strukturu `downsampler`, u 3 uzastopne strukture `rcon5`, završni konvolucijski sloj veličine 9 x 9 piksela s pomakom od 1 piksela te je na cijelu mrežu primijenjena funkcija gubitka.

Važno je napomenuti da u ovom slučaju koristimo predtrenirani model biblioteke dlib treniran za otkrivanje ljudskog lica koji je javno dostupan te je treniranje obavljano na točno ovako definiranom CNN-u. Korištenje ovog modela nije moguće na drugačije definiranom CNN-u. Kada bismo željeli trenirati svoj model, tada bismo bili slobodni definirati vlastitu strukturu CNN-a i eventualno mogli dobiti bolje rezultate detekcije ili bolje vrijeme izvođenja detekcije.

Učitavanje predtreniranog modela u definirani CNN obavlja se pozivom metode `deserialize` (Kod 3.9.) kojoj prosljeđujemo putanju do datoteke koja sadržava model.

```
dlib::deserialize("mmod_human_face_detector.dat") >> _net;
```

Kod 3.9. Učitavanje predtreniranog modela CNN-a

Metodu detekcije biramo parametriziranjem klase `VideoStreamFaceDetector` kroz metodu `SetDetectionType` kojoj predajemo enumeraciju `DetectionType` i ona može biti HOG ili CNN.

Ako je odabrana metoda detekcije HOG, izvršava se detekcija pomoću HOG detektora (Kod 3.10.), a ako je odabrana metoda CNN, koristi se CNN detektor (Kod 3.11.).

```
detections = detector(cimg);
```

Kod 3.10. Detekcija HOG detektorom

```
dlib::assign_image(matrix, cimg);
netDetections = _net(matrix);
```

Kod 3.11. Detekcija CNN detektorom

Kako bi pravokutnici detekcije odgovarali veličinom i pozicijom, na izvornoj slici potrebno je napraviti korekciju pozicije i veličine ovisno o faktoru skaliranja kojim smo povećali ili

smanjili ulaznu sliku (Kod 3.12). U ovom koraku procesa također dodjeljujemo svakoj detekciji i vlastiti korelacijski *tracker* kojim ćemo pratiti lice u sličicama u kojima ne obavljamo detekciju.

```
for (size_t i = 0; i < detections.size(); i++){
            detections[i].set_left(
      detections[i].left()*_imageScaleFactor);
            detections[i].set_bottom(
      detections[i].bottom()*_imageScaleFactor);
            detections[i].set_right(
      detections[i].right()*_imageScaleFactor);
            detections[i].set_top(
      detections[i].top()*_imageScaleFactor);
            dlib::correlation_tracker ct =
      dlib::correlation_tracker(
      5, 5, 23, 0.001, 0.025, 0.001, 0.025, 1.02);
            ct.start_track(trackImg, detections[i]);
            trackers.push_back(ct);
      }
```

Kod 3.12. Korekcija pozicije i veličine pravokutnika detekcije ovisno o faktoru skaliranja i dodjeljivanje korelacijskog *trackera* svakoj detekciji

Ako se na ulaznoj slici ne obavlja detekcija lica, na njoj radimo osvježavanje svih postojećih korelacijskih *trackera* iz prethodne sličice. Prilikom poziva metode `update()` korelacijski *tracker* kao povratnu vrijednost vraća vrijednost korelacije što predstavlja faktor pouzdanosti da se praćeni potprozor nalazi na otkrivenome mjestu unutar trenutne slike. Empirijski je određeno da u našem slučaju faktor pouzdanosti manji od 5 znači nepouzdano praćenje i u tom trenutku odbacujemo taj *tracker* i brišemo ga s liste *trackera* (Kod 3.13.).

```
if (!result.hasDetections){
      if (trackers.size() > 0){
      double correlationValue;
      for (int i = trackers.size() - 1; i >= 0; i--){
            correlationValue = trackers[i].update(trackImg);
            if (correlationValue < 5.0){
                  trackers.erase(trackers.begin() + i);
                  }
            }
      result.tracked_faces = trackers;
```

```
            result.hasTrackedFaces = true;
            }
        }
```

Kod 3.13. Ažuriranje *trackera*

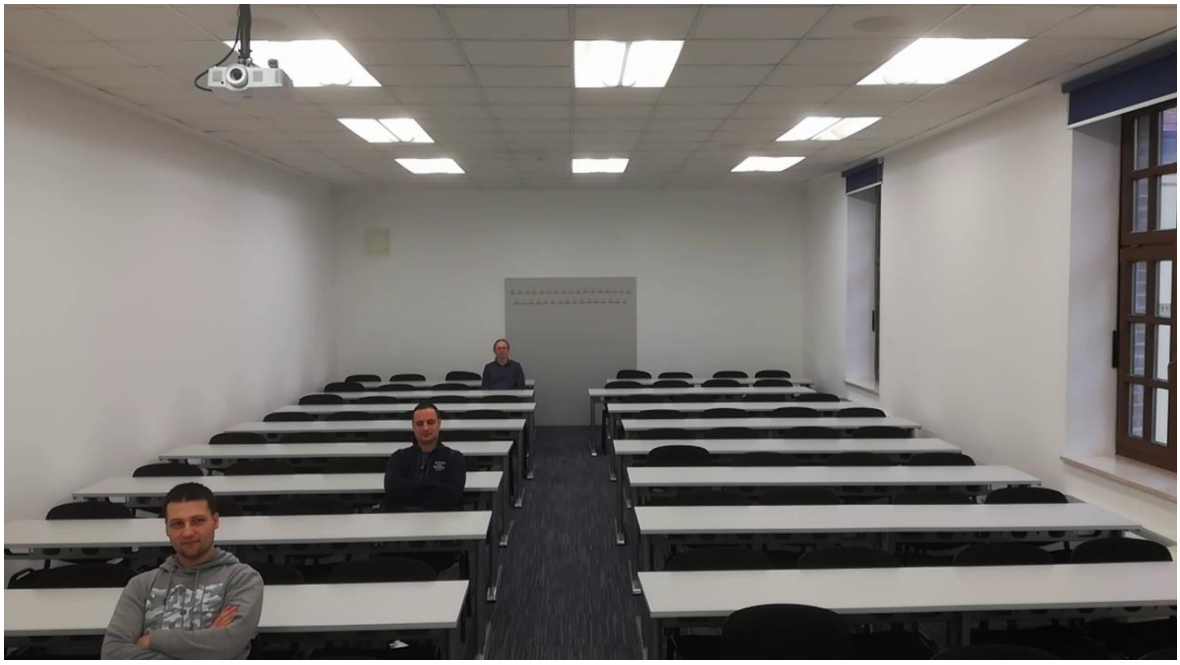
Izlazna metoda zadužena je za podizanje *evenata* koji sadrže informacije o detekcijama te samo sličicu na kojoj je obavljana detekcija. Izlazni događaji moraju se podizati u vremenskim intervalima tako da je maksimalni *frame rate* jednak željenom *frame rateu* parametriziranom od strane korisnika. To znači da se izlaz iz modula ne smije odvijati brže od željenog *frame ratea.* Metoda je također zadužena za ispravljanje razlike u vremenu dolaska sličica u izlazni međuspremnik zbog optimizacije detekcije njezinim neobavljanjem na svakoj sličici. To radimo tako da postavimo minimalni broj sličica u izlaznom međuspremniku prije kojeg ne počinjemo slati događaje pretplaćenim metodama. Ova vrijednost može se parametrizirati pozivom metode *SetBufferMinimumSize()*. Također, ako se izlazni međuspremnik isprazni, čekamo dok se ne napuni do te vrijednosti te tek tada ponovno krenemo slati događaje. Samo slanje događaja sastoji se od toga da uzmemo sljedeći rezultat iz izlaznog međuspremnika i pošaljemo ga preplaćenim metodama (Kod 3.14.).

```
if (_displayFramesBuffer.size_approx() >= _bufferMinimumSize){
            _videoRendering = true;
            while (_videoRendering){
                  DetectionResult result;
                  bool dequeueSuccess =
      _displayFramesBuffer.try_dequeue(result);
                  if (dequeueSuccess){
                        _fireFrame(result);
                  }
                  else {
                        _videoRendering = false;
                        }
            }
      }
```

Kod 3.14. Izlazna metoda

4. Testiranje sustava detekcije

Testiranje sustava detekcije obavljeno je na predsnimljenom videomaterijalu u jednoj od predavaonica. Na snimci se nalaze 3 subjekta koje pokušavamo detektirati. Udaljenosti subjekta su 3 m, 5 m i 14 m i predstavljaju najbolji, najgori i učestali slučaj u kojem se subjekti mogu nalaziti (Slika 4.1.).



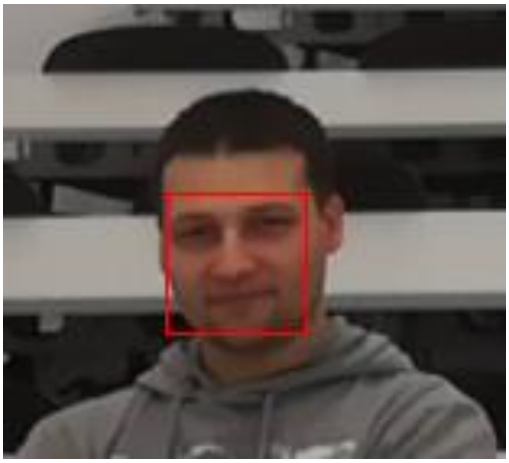
Slika 4.1 Sličica testne videosnimke

Ukupno trajanje videosnimke je 21 sekundu te sadrži ukupno 637 sličica. Razlučivost videa je 1280 x 720 piksela, a *Frame rate* snimke 30 FPS. Za testni scenarij odabrano je da se detekcija obavlja dva puta u sekundi, tj. na svakoj 15. sličici. Na ostalim sličicama samo pratimo detektirana lica. Važno je napomenuti da i samu detekciju i samo praćenje lica uzimamo kao uspješnu detekciju. U ovoj izvedbi aplikacije uspješnu detekciju označavamo crvenim pravokutnikom oko lica subjekta (Slika 4.2.) dok uspješno praćenje označavamo svijetloplavim pravokutnikom (Slika 4.3.). Testiranje je obavljano na dvije računalne konfiguracije (Tablica 4.1.). Testiranje CNN-a obavljeno je samo na Konfiguraciji 2 jer obrada putem CNN-a zahtijeva Nvidia GPU s podrškom za Cuda platformu. Prilikom testiranja prate se tri parametra: ukupan broj detekcija u pojedinoj sličici, prosječno vrijeme obrade pojedine sličice u grupi detekcije AFT (engl. *Average Frame Time*) i vrijeme obrade sličice detekcije ADFT (engl. *Average Detection Frame Time*). AFT treba uzeti s rezervom

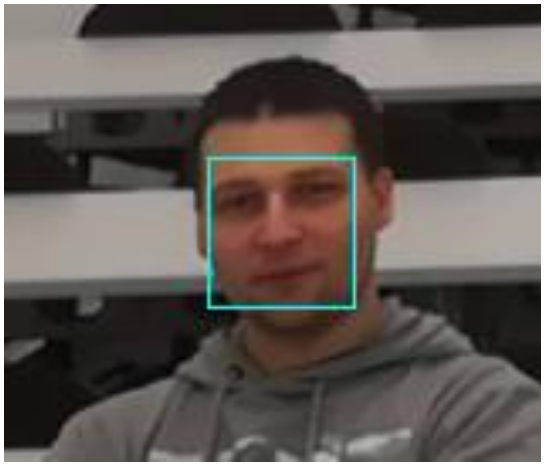
jer predstavlja mjeru koja je podložna promjenama s odabirom češćeg obavljanja detekcije. U našem slučaju AFT računa se kroz 15 uzastopnih sličica.

Tablica 4.1 Specifikacije testnih konfiguracija

	CPU	GPU	RAM
Konfiguracija 1	Intel Core i5-6300HQ 2.3GHz	Intel HD Graphics 530	16GB
Konfiguracija 2	Intel Core i5-6600 3.30GHz	NVIDIA GeForce GTX 1070 8Gb 1556 MHz	16GB



Slika 4.2. Lice s pozitivnom detekcijom

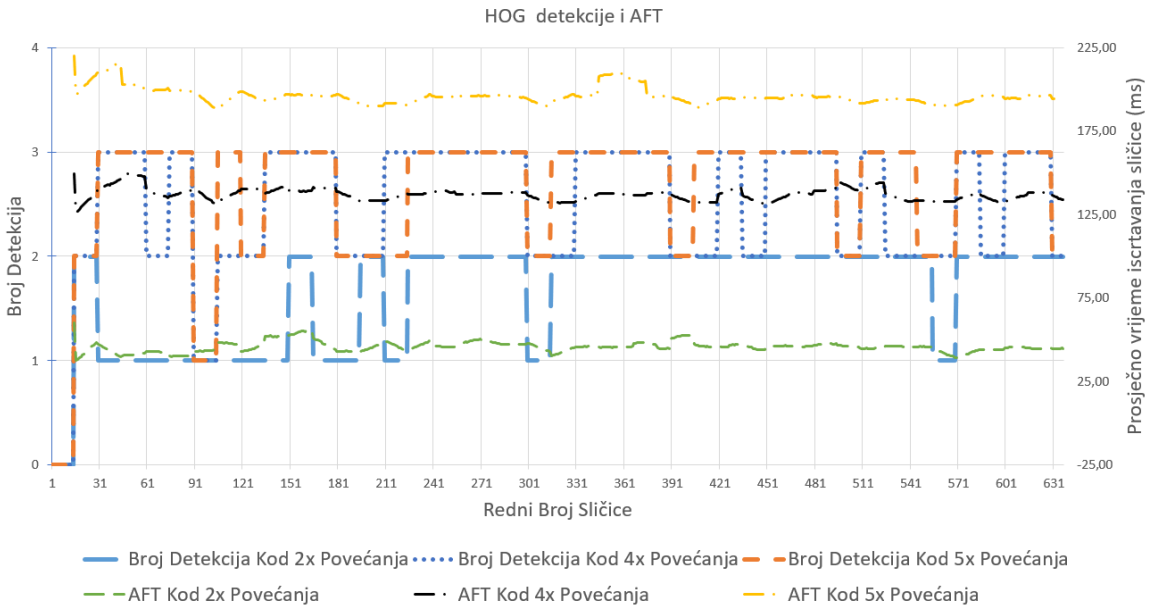


Slika 4.3. Lice s pozitivnim praćenjem

4.1. Rezultati testiranja sustava detekcije lica

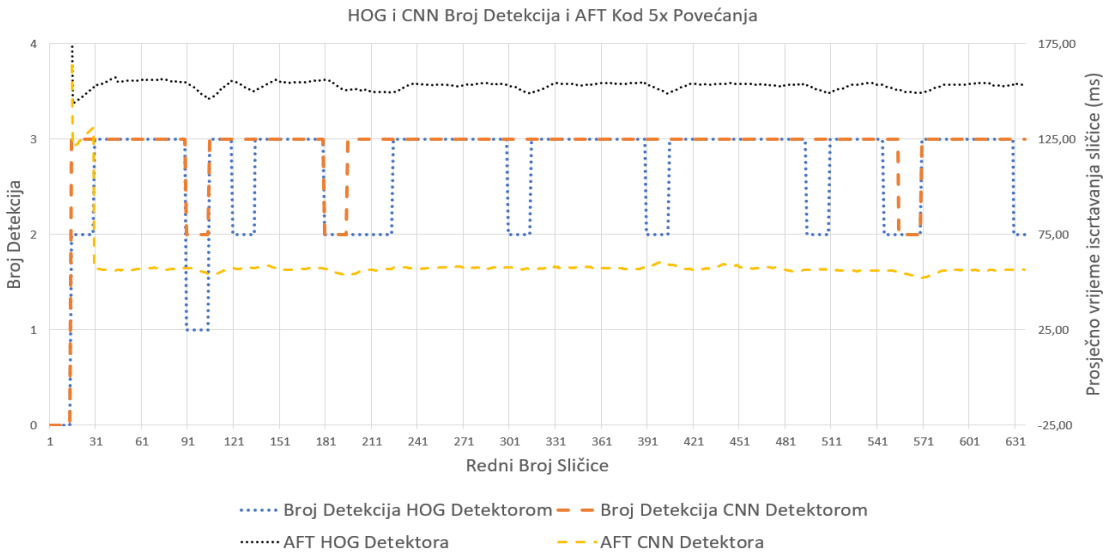
Na grafikonima (Slika 4.4., Slika 4.5.) promatramo broj detekcija u pojedinoj sličici prikazan ravnijim linijama raspona vrijednosti od 0 do 3 i prosječno vrijeme iscrtavanja sličice (AFT *Average Frame Time*) ulaznog materijala prikazano nepravilnijim linijama raspona od 25 do 225 ms. Na grafikonu (Slika 4.4.) vidljivi su rezultati dobiveni HOG detektorom koji pokazuju da pri povećanju ulazne sličice od 2 x detektor prepoznaje između jednog i dva lica u svakoj sličici ulaznog videa s ATF-om oko 45 ms. Detektor počinje detektirati treće lice tek nakon uvećanja ulazne slike za faktor 4 što znači da tada detekciju obavljamo na slici veličine 5120 x 2880 piksela. Pri povećanju slike na faktor 5 nije vidljiv velik pomak u

broju detekcija dok se ATF povećao s prosječnih 140 ms na 200 ms. Iz ovog je vidljivo da je za uspješnu detekciju u ovoj primjeni detekcije dostatan faktor povećanja 4.



Slika 4.4. Broj detekcija u pojedinim sličicama te AFT (ms)

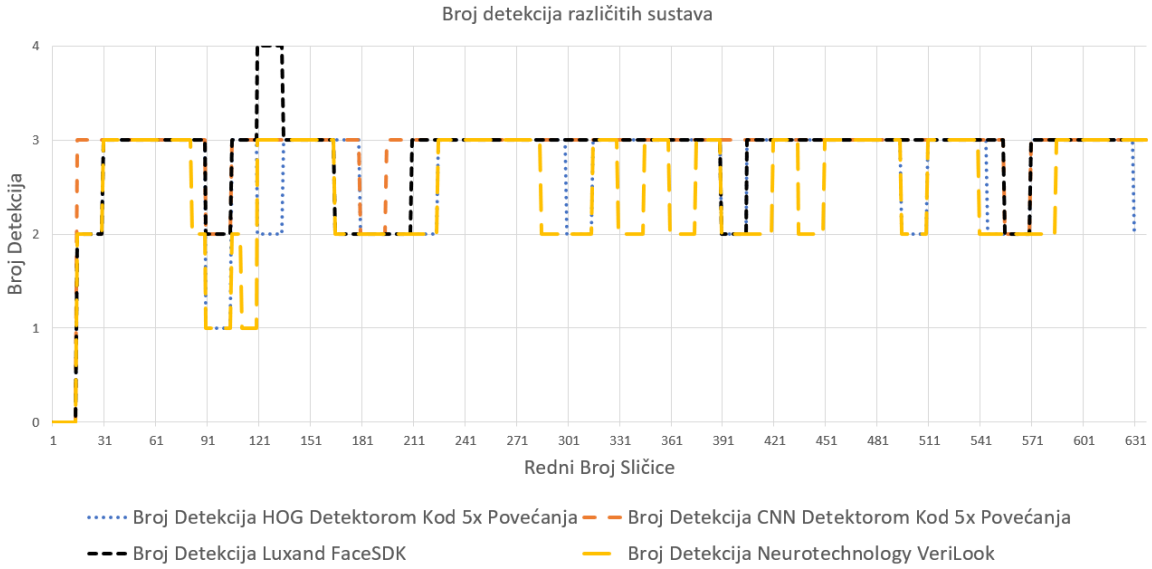
Grafikon (Slika 4.5.) prikazuje usporedbu rezultata CNN i HOG detektora te njihov AFT. Vidljivo je osjetno poboljšanje broja detekcija korištenjem CNN detektora. Kroz gotovo čitav videobroj detekcija odgovara broju subjekata koji se nalaze na snimci. To znači da je CNN detektor otporniji na pokrete subjekata koji zakrivaju dijelove lica te na zakretanja glava kod kojih HOG detektor više nije u stanju prepoznati lice subjekta. AFT CNN detektora otprilike je 3 x manji, ali to je dijelom zbog činjenice da se CNN detektor izvršava na GPU-u pa je takav rezultat i očekivan.



Slika 4.5. Usporedba broja detekcija HOG i CNN detektora te njihov AFT (ms)

4.2. Usporedba rezultata izrađenog sustava s komercijalnim sustavima

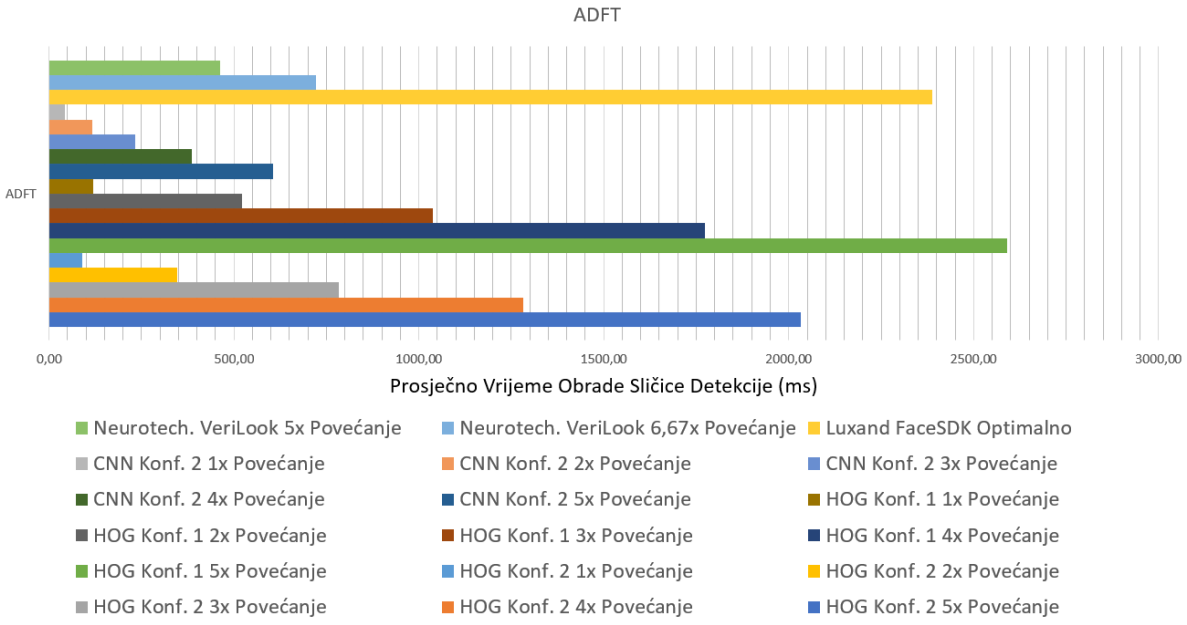
Izvršeno je testiranje 2 komercijalno dostupna sustava za detekciju ljudskog lica, Luxand FaceSDK i Neurotechnology VeriLook te su rezultati uspoređeni s našim sustavom. Komercijalne sustave nije bilo moguće parametrizirati na jednak način kao i naš sustav pa su oni baždareni da ostvare optimalnu točnost detekcije pazeći pri tome na što brže vrijeme obrade sličica. Rezultati detekcije komercijalnih sustava u poretku s dvije metode detekcije našeg sustava prikazani su na grafikonu (Slika 4.6.).



Slika 4.6. Broj detekcija s obzirom na korišteni sustav detekcije

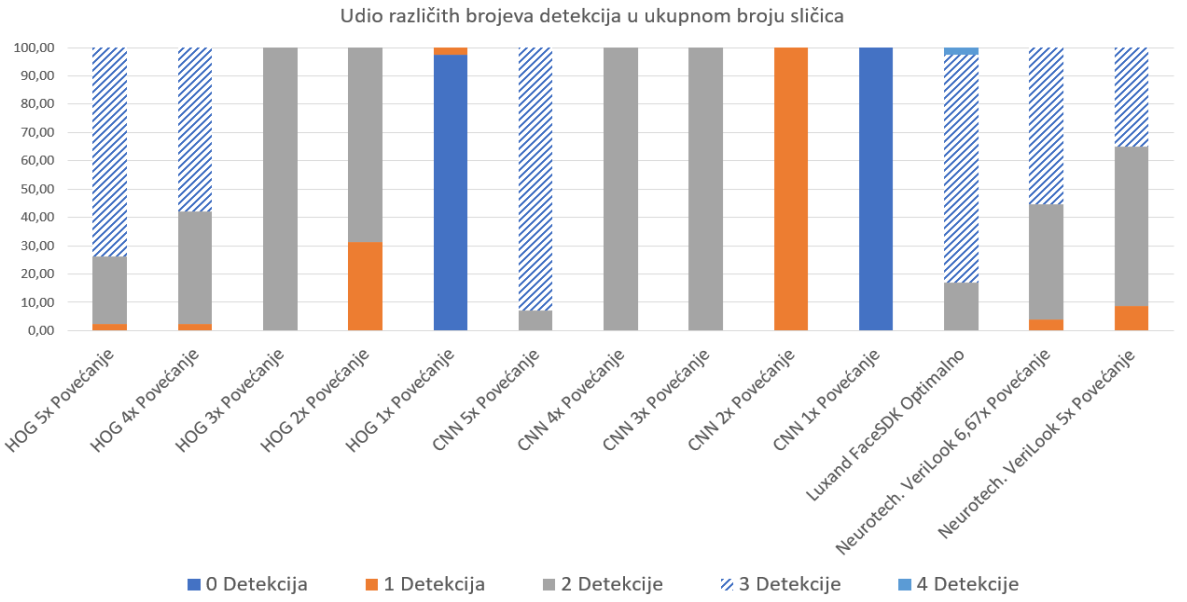
Vidljivo je da svi testirani sustavi odrađuju detekciju lica svih prisutnih subjekata većinu vremena. Također je zanimljivo da Luxand FaceSDK u jednoj grupi sličica prepoznaje 4 lica što znači da prepoznaje nepostojeće lice, tj. da očitava lažno pozitivan rezultat. Nijedan od ostalih testiranih sustava nije imao lažno pozitivnih rezultata.

Promatranjem vremena potrebnog za izvršavanje detekcije na sličici (Slika 4.7.) dolazimo do rezultata koji pokazuju da najbolje vrijeme detekcije pri kojem se prepoznaju sva 3 subjekta na snimci ima Neurotechnology VeriLook, slijedi ga CNN detektor pa HOG detektor. Zanimljivo je da je Luxand FaceSDK jedan od najsporijih u brzini detekcije na našem testnom scenariju.



Slika 4.7. ADFT testiranih sustava

Promatrajući udio pojedinog broja detekcija na sličicama u odnosu na ukupan broj sličica (Slika 4.8.) možemo vidjeti da je svaki od detektora, ako dovoljno povećamo dimenzije ulazne slike, sposoban uspješno detektirati ljudska lica na udaljenostima iz našeg testnog scenarija i to na više od 50 % od ukupnog broja sličica u videomaterijalu. Zanimljivo je promatrati CNN detektor koji, s povećanjem faktora skaliranja slike, ima skoro 100-postotno prepoznavanje u svim slučajevima. To znači da kada se lice na sličici pojavi u veličini dostatnoj za prepoznavanje, CNN detektor ga prepoznaje u svim pozama bez obzira na zakretanje glava subjekata i na djelomična zakrivanja dijela lica.



Slika 4.8. Udio broja detekcija u ukupnom broju sličica testnog materijala

U tablici su vidljivi kompletni rezultati testiranja koji uključuju testiranje na obje konfiguracije, korištenje obje vrste detektora s različitim faktorima skaliranja te testiranje performansi komercijalnih sustava detekcije.

Tablica 4.2. Rezultati testiranja sustava

					Broj detekcija				
					0	1	2	3	4
Tip Detekcije	Hardver	Faktor povećanja	AFT (ms)	ADFT (ms)	%	%	%	%	%
HOG	Konf. 2	5 x	154,56	2033,25	0,00	2,41	24,56	73,03	0,00
		4 x	98,90	1281,80	0,00	2,41	39,81	57,78	0,00
		3 x	62,30	782,79	0,00	0,00	100,00	0,00	0,00
		2 x	30,57	347,02	0,00	31,30	68,70	0,00	0,00
		1 x	6,76	89,97	97,59	2,41	0,00	0,00	0,00
	Konf. 1	5 x	199,78	2591,58	0,00	2,43	23,82	73,74	0,00
		4 x	138,47	1773,52	0,00	2,41	39,81	57,78	0,00
		3 x	82,23	1038,38	0,00	0,00	100,00	0,00	0,00
		2 x	46,27	521,02	0,00	31,30	68,70	0,00	0,00
		1 x	8,88	118,23	97,59	2,41	0,00	0,00	0,00
CNN	Konf. 2	5 x	59,03	605,07	0,00	0,00	7,22	92,78	0,00
		4 x	37,00	385,84	0,00	0,00	100,00	0,00	0,00
		3 x	25,48	233,81	0,00	0,00	100,00	0,00	0,00
		2 x	13,40	115,83	0,00	100,00	0,00	0,00	0,00
		1 x	3,51	42,55	100,00	0,00	0,00	0,00	0,00
Luxand FaceSDK	Konf. 1	Optimalno	173,91	2387,10	0,00	0,00	16,85	80,74	2,41
Neurotech. VeriLook	Konf. 1	6,67 x	80,41	721,37	0,00	3,85	40,77	55,38	0,00
		5 x	53,31	462,57	0,00	8,67	56,34	34,99	0,00

Zaključak

Kako se proširuju područja u kojima koristimo računala, tako se populariziraju kompleksne računalne metode te sele iz znanstvenih okruženja u svakodnevnu uporabu. Detekcija objekata pa tako i detekcija lica postaje sve prisutnija u mnogim uređajima koje koristimo.

Istraživanjem mogućnosti korištenja biblioteka otvorenoga koda u svrhu izrade upotrebljivog sustava detekcije ljudskog lica zadovoljavajuće točnosti pokazali smo da je itekako moguće takav sustav ostvariti. Opširna dokumentacija, kao i mnoštvo primjera uporabe koji prate korištene biblioteke, uvelike su olakšali izgradnju sustava, a performanse implementiranih algoritama su na razini komercijalno dostupnih sustava detekcije lica. Također, važno je napomenuti da je autoru ovog rada bilo puno lakše implementirati korištena rješenja otvorenoga koda nego rješenja SDK-ova testiranih komercijalnih aplikacija.

Izgrađeni sustav je u videotijeku dimenzija 1280 x 720 u mogućnosti detektirati ljudska lica na udaljenostima od 2 m do 14 m, uz prethodnu pripremu pojedine sličice skaliranjem na potrebnu veličinu. Optimizacijom sustava također je moguće ostvariti da se detekcija obavlja u realnom vremenu što nas dovodi do zaključka da bi se ovakav sustav mogao uspješno koristiti u scenariju detekcije studenata na predavanjima i kao priprema za kompletan sustav automatskog ažuriranja prisutnosti studenata na predavanjima i vježbama.

Najtočnijim detektorom pokazao se detektor temeljen na konvolucijskim neuronskim mrežama, a kako se izvodi na GPU-u, brzina detekcije koju postiže najbolja je od svih testiranih sustava detekcije. Njegova otpornost na promjene u pozama i na rotacije glava subjekata čine ga boljim izborom od detektora temeljenog na HOG metodi, ako računalni sustav podržava njegovo izvršavanje.

Promatrajući ostale rezultate vidljivo je da bi se na bibliotekama otvorenoga koda mogao izraditi detektor lica pogodan za pronalaženje lica na manjim udaljenostima koji bi se mogao koristiti na slabijim hardverskim konfiguracijama ili u situacijama gdje je jedno računalo zaduženo za obavljanje detekcije na više videotijekova te bi se mogao koristiti u sustavima pametnih kuća za detekciju prisutnosti osoba u promatranom prostoru.

Sljedeći korak u izradi sustava automatskog ažuriranja prisutnosti bio bi izgradnja modula prepoznavanja ljudskog lica koji bi koristio modul izrađen u ovom radu kao svoj ulaz i na temelju njega pokušavao identificirati detektirane osobe.

Popis kratica

CNN	*Convolutional Neural Network*	Konvolucijska neuronska mreža
SDK	*Software Development Kit*	Skup alata za razvoj softvera
AFT	*Average Frame Time*	Prosječno vrijeme sličice
ADFT	*Average Detection Frame Time*	Prosječno vrijeme sličice detekcije
GPU	*Graphics Processing Unit*	Grafička procesorska jedinica

Popis slika

Popis tablica

Popis kodova

Literatura

[1] LI, S.Z., JAIN A.K. *Handbook of Face Recognition*. London: Springer, 2011.

[2] DELAC, K., GRGIC, M. *Face Recognition*. I-TECH Education and Publishing 2007.

[3] VIOLA, P., JONES, M. *Local Rapid Object Detection using a Boosted Cascade of Simple Features. Conference on Computer Vision and Pattern Recognition*, 2001.

[4] RUTHVIK, V., KUMAR, M.N., GOPAL, S., KARNANI, N. *Raspberry Pi Based GPS Tracking System and Face Recognition System. Diplomski rad*. National Institute of Technology Calicut, 2014.

[5] Dalal, N., Triggs, b. *Histograms of Oriented Gradients for Human Detection. Conference on Computer Vision and Pattern Recognition*, 2005.

[6] DLIB BLOG, Dlib 18.6 released: Make your own object detector, http://blog.dlib.net/2014/02/dlib-186-released-make-your-own-object.html/, pristupljeno: 23. prosinca 2017.

[7] CHRIS MCCORMICK, Gradient Vectors, http://mccormickml.com/2013/05/07/gradient-vectors/, pristupljeno 2. siječnja 2018.

[8] LEARN OPENCV, Histogram of Orianted Gradients, https://www.learnopencv.com/histogram-of-oriented-gradients/, pristupljeno 2. siječnja 2018.

[9] CHRIS MCCORMICK, HOG Person Detector Tutorial, http://mccormickml.com/2013/05/09/hog-person-detector-tutorial/, pristupljeno 2. siječnja 2018.

[10] KIRILL EREMENKO, Machine Learning A-Z, CNN – What are CNNs , Udemy https://www.udemy.com/machinelearning/learn/v4/t/lecture/6761138?start=0/, pristupljeno 5. siječnja 2018.

[11] KIRILL EREMENKO, Machine Learning A-Z, CNN – Convolution Operation, Udemy https://www.udemy.com/machinelearning/learn/v4/t/lecture/6761140?start=0, pristupljeno 5. siječnja 2018.

[12] KIRILL EREMENKO, Machine Learning A-Z, CNN – ReLU Layer, Udemy https://www.udemy.com/machinelearning/learn/v4/t/lecture/6761142?start=0, pristupljeno 5. siječnja 2018.

[13] KIRILL EREMENKO, Machine Learning A-Z, CNN – Pooling , Udemy https://www.udemy.com/machinelearning/learn/v4/t/lecture/6761144?start=0/, pristupljeno 5. siječnja 2018.

[14] KIRILL EREMENKO, Machine Learning A-Z, CNN – Full Connection , Udemy https://www.udemy.com/machinelearning/learn/v4/t/lecture/6761148?start=0/, pristupljeno 5. siječnja 2018.

[15] THE DATA SCIENCE BLOG, An Intuitive Explanation of Convolutional Neural Networks, https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/, pristupljeno 5. siječnja 2018.

[16] OPENCV, About OpenCV, https://opencv.org/about.html, pristupljeno 6. siječnja 2018.

[17] DLIB C++ LIBRARY, http://dlib.net/, pristupljeno 6. siječnja 2018.

[18] CCV, A Modern Computer Vision Library, http://libccv.org/, pristupljeno 6. siječnja 2018.

[19] ACSYS BIOMETRICS, Software Developers Kit, http://www.acsysbiometrics.com/product_sdk.html, pristupljeno 12. siječnja 2018.

[20] BETAFACE, Face Detection and Recognition SDK https://www.betaface.com/wpa/index.php/products, pristupljeno 12. siječnja 2018.

[21] NEUROTECHNOLOGY, VeriLook SDK, http://www.neurotechnology.com/verilook.html, pristupljeno 12. siječnja 2018.

[22] LUXAND, Detect and Recognize Faces with Luxand FaceSDK, https://www.luxand.com/facesdk/, pristupljeno 12. siječnja 2018.

[23] DLIB C++ LIBRARY, Running a CNN Based face detector using Dlib http:// http://dlib.net/dnn_mmod_face_detection_ex.cpp.html/, pristupljeno 18. siječnja 2018.