Problem prepoznavanja lica na slici

Romac, Josip

Sveučilište u Splitu

Prirodoslovno-matematički fakultet

Odjel za informatiku

Josip Romac

PROBLEM PREPOZNAVANJA LICA NA SLICI

Diplomski rad

Split, 2020.

Temeljna dokumentacijska kartica

Diplomski rad

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za informatiku
Ruđera Boškovića 33, 21000 Split, Hrvatska

PROBLEM PREPOZNAVANJA LICA NA SLICI

Josip Romac

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 54 stranica, 36 grafičkih prikaza i 25 literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor: **Dr. sc. Saša Mladenović,** *izvanredni profesor Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Ocjenjivači: **Dr. sc. Goran Zaharija,** *viši asistent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Divna Krpan, *viši predavač Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Rad prihvaćen: **09 2020**

Basic documentation card

Thesis

University of Split
Faculty of Science
Department of informatics
Ruđera Boškovića 33, 21000 Split, Croatia

FACE RECOGNITION PROBLEM ON PICTURE

Josip Romac

Thesis deposited in library of Faculty of science, University of Split

Thesis consists of: 54 pages, 36 figures and 25 references

Original language: Croatian

Mentor: **Saša Mladenović, Ph.D.** *Associate Professor of Faculty of Science, University of Split*

Reviewers: **Goran Zaharija, Ph.D.** *Assistant Professor of Faculty of Science, University of Split*

Divna Krpan, *Senior Lecturer of Faculty of Science, University of Split*

Thesis accepted: **09 2020**

Sadržaj

1. UVOD

Nove tehnologije razvijaju se svakodnevno i ulaze u sve aspekte ljudskog života. Iako je tehnologija već desetljećima sveprisutna, još uvijek postoje područja i načini primjene koji još uvijek ili nisu razvijeni ili su još u ranoj fazi razvoja. Jedno od najatraktivnijih područja današnjice jest umjetna inteligencija. Ovo područje privlači veliku pozornost zajednice, ne samo zbog činjenice kako bi razvoj istog doveo do revolucionarnih otkrića nego i straha od potencijalnih negativnih posljedica koje bi takva tehnologija mogla donijeti. Računalna moć današnjice i dalje je u eksponencijalnom porastu što dovodi do stvaranja sve inteligentnijih računalnih sustava. Zahvaljujući tome pa i mnogim istraživačkim projektima, u današnje vrijeme imamo proizvode koji su sposobni komunicirati sa ljudima naizgled na prirodnoj razini. Primjerice promatranjem ljudi kroz kameru i slušajući ljudske glasove kroz mikrofon, računalo je sposobno donekle razumjeti ljude i adekvatno reagirati.

Jedna od temeljnih tehnika koja omogućava prirodnu interakciju čovjeka i računala jest detekcija lica. Detekcija lica temelj je svih algoritama za analizu lica uključujući poravnanje lica, modeliranje lica, prepoznavanje izraza lica, prepoznavanje spola i dobi, prepoznavanje lica i mnoge druge. Dakle, računala mogu jasno razumjeti lice te su kao takva sposobna razumjeti ljudske misli i namjere. U smislu digitalne slike, primarni cilj detekcije lica jest utvrđivanje postoji li lice na nekoj slici ili ne. Iako se iz ljudske perspektive ovo čini kao vrlo jednostavan zadatak, za računalo je to pak pravi izazov. Posljednjih nekoliko desetljeća, detekcija lica na slikama bila je jedna od najpoznatijih istraživačkih tema računarstva. Detekcija lica postigla je značajan napredak u posljednjem desetljeću dovodeći detekciju na zavidnu razinu. Iako su još uvijek istraživanja detekcije lica široko zastupljena, sve je veći broj istraživanja usmjerenih na prepoznavanje lice nego na detekciju. Prepoznavanje lica novi je izazov istraživačima diljem svijeta i zasigurno će ostati u desetljećima koji slijede prvenstveno zbog povezanosti sa umjetnom inteligencijom sa kojom će se paralelno razvijati.

I dok su se ranije poteškoće povezane sa detekcijom lica pripisivale mnogim varijacijama u mjeri, položaju, orijentaciji, pozi i izrazu lica, uvjetima osvjetljenja i slično, problemi prepoznavanja lica usmjerene su prema biometriji. Problem prepoznavanja lica je u pronalaženju jedinstvenih vrijednosti lica pojedinca koje će biti pripisane samo njemu. Sakupljanje ovakvih podataka i predviđanje lica na temelju prikupljenih podataka, temelj su današnjih tehnika prepoznavanja lica i općenito biometrije.

Računalni vid jedno je od polja koje spada u područje umjetne inteligencije. Ovaj će rad ponuditi rješenje problema prepoznavanja lica na slici. Kako u mozaiku piksela koje sadrži jedna slika detektirati pojavu objekta, u ovom slučaju lica, samo je jedan problem u nizu do konačnog prepoznavanja lica. Kako bi zahtjev bio ostvaren, biti će potrebno koristiti različite računalne mehanizme uključujući biblioteke sa podrškom za računalnu vid i strojno učenje kao i modele potpomognute umjetnom inteligencijom. Konačno rješenje biti će ostvareno kroz faze detekcije objekta, detekcije crta lica (oči, nos i usne) i općenito lica, uvježbavanje modela prepoznavanja i prepoznavanje lica. U radu će biti sadržano i nekoliko mogućnosti primjene ovakve tehnologije. Pokazat će se kako prepoznavanje lica može biti korisno od svakodnevnih životnih prilika do neprilika.

Sustavi temeljeni na prepoznavanju lica danas imaju široku primjenu i mnogo su zastupljeni već neko vrijeme. Novije generacije mobilnih uređaja dolaze sa značajkom otključavanja zaslona skeniranjem lica. Sličnim principom koriste se i proizvođači u autoindustriji. Neki modeli imaju mogućnost spremanja personaliziranih postavki automobila. Kada vozač pristupi vozilu, skeniranjem lica vozilo se prilagođava trenutnom vozaču i aktiviraju se sve ranije spremljene postavke. Unošenje lozinke prilikom pristupa osobnim podacima sve je manje potrebno jer sustavi postepeno uvode mogućnost prijave putem skeniranja lica.

2. KONKRETAN PROBLEM

Problem ovog rada prvenstveno se odnosi na mogućnost prepoznavanja lica na slici i načina izvedbe rješenja. Lice na slici može se promatrati kao objekt, stoga bi se rješenje ovog rada trebalo zasnivati na detekciji objekta. Karakteristike lica, kao što su oči, nos i usne, čine lice tipom objekta. Pronalazak takvih karakteristika na slici zahtijeva realizaciju detekcije lica. Svako lice jedinstveno je na svoj način. Prepoznavanje nekog lica podrazumijeva poznavanje tog lica od ranije. Kako bi neko lice postalo poznato, potrebno je učenje lica, odnosno pronalaženje i pamćenje jedinstvenih karakteristika tog lica. Procesom učenja lica, kreira se model poznatih lica sa pripadajućim jedinstvenim karakteristikama. Kod pokušaja prepoznavanja nekog lica, karakteristike tog lica uspoređuju se sa onima iz modela. Ukoliko se karakteristike tog lica podudaraju sa jedinstvenim karakteristikama jednog od poznatih lica iz modela, tada je došlo do prepoznavanja lica. U protivnom, lice neće biti prepoznato te će biti potrebno naučiti to lice prije prepoznavanja. Postepeno obavljanje ovih zadaća rezultirat će rješenjem problema ovog rada.

2.1 Detekcija objekta

Detekcija objekta je pronalaženje nekih karakteristika koje su jedinstvene tom tipu objekta u nekom području. Karakteristike su vrlo važne jer pridonose klasificiranju objekta. Primjerice, kada se traži lice, provjerava se udaljenost između očiju, pozicija nosa i usana te ostale značajke, poput boje kože i slično. Detekcija objekta je računalna tehnologija koja je usko povezana sa računalnim vidom i obradom slike. Uloga ove tehnologije jest otkriti jedinke semantičkih objekata neke klase kao na primjer ljudi, automobili ili u ovom slučaju lice. Detekcija objekta zastupljena je u mnogim područjima računalnog vida. Jedna od najbolje istraženih područja detekcije objekta su detekcija lica i detekcija pješaka.

Metode detekcije objekta temelje se na strojnom učenju ili dubokom učenju te ih kao takve možemo svrstati u područje AI. Kod strojnog učenja, potrebno je prvo definirati karakteristike pomoću jedne od metoda a zatim izvršiti klasifikaciju. Pristupom dubokog učenja detekciju objekta vrši se bez definiranja karakteristika. Poznatiji pristupi strojnog učenja su Viola-Jones (Haarove značajke) te histogram orijentiranih gradijenata (engl. *histogram of oriented gradients*, skraćeno HOG). Najpoznatiji primjer pristupa dubokog učenja jest konvolucijska neuronska mreža [1].

[1] Wikipedia https://en.wikipedia.org/wiki/Object_detection

2.2 Detekcija lica

Detekcija lica prvi je korak rješavanja problema prepoznavanja lica u slici. Uz to, ovo je prvi korak pri rješavanju i drugih problema zasnovanih na licu, primjerice praćenje i analiza lica.

Detekcija lica računalna je tehnologija temeljena na AI koja može identificirati i locirati pojavu ljudskih lica na digitalnim slikama i videozapisima. Može se smatrati posebnim slučajem otkrivanja objekta neke klase, gdje je zadatak pronaći lokacije i odrediti veličine svih objekata koji pripadaju određenoj klasi. U ovom slučaju lica unutar određene slike. Zbog napretka u tehnologiji detekcije lica, sada je moguće otkriti lica na slici ili u videozapisu, bez obzira na pozu glave, uvjete osvjetljenja i boju kože.

Aplikacije za detekciju lica koriste algoritme koji razlikuju pozitivne slike, odnosno slike na kojima se pojavljuje lice, i negativne slike bez pojave lica. Kako bi algoritmi bili precizni i pouzdani, moraju biti uvježbani na ogromnim skupovima podataka. Takav skup podataka sadrži tisuće različitih primjeraka slika sa licem i bez lica. Za dobro uvježban algoritam potrebna je pozamašna računalna moć te je iste često teško i gotovo nemoguće stvoriti na prosječnom osobnom računalu. Jednom uvježban algoritam uspješno će provjeriti nalazi li se lice na slici te u slučaju potvrdnog odgovora dostaviti i mjesto pronalaska. Ovisno o postavkama i aplikaciji, algoritam će najčešće postaviti jednobojni kvadrat preko regije prepoznatog područja.

Ovakvi algoritmi u prošlosti su se temeljili na strojnom učenju te je uspješnost detekcije imala vrlo nisku toleranciju odstupanja. Primjerice, lica je bilo nemoguće pronaći ukoliko je bivalo zakrenuto previše u jednu stranu ili previše nagnuto prema gore ili dolje. Također, uspješnost pronalaska ovisila je previše i o razini osvjetljenja slike u odnosu na ona algoritmom naučena. Danas uglavnom prevladavaju mnogo pouzdanije metode dubokog učenja koje koriste širok raspon različitih scena detekcije lica.

U kontekstu analize lica, algoritmi za otkrivanje lica otkriti će na koje dijelove slike se treba usredotočiti prilikom utvrđivanja dobi, prepoznavanja spola i analize emocija na temelju izraza lica [2].

[2] Sightcorp https://sightcorp.com/knowledge-base/face-detection/

2.2.1 Viola-Jones sa Haarovim značajkama

Viola-Jones algoritam nazvan je po svojim autorima, Paul Viola i Michael Jones, koji su ga osmislili 2001. godine. Iako je prvotno bio namijenjen automatskoj detekciji lica u realnom vremenu, može se koristiti pri detekciji bilo kakvih objekata. Algoritam je baziran na strojnom učenju gdje se uz pomoć mnogo primjera pozitivnih i negativnih slika uvježbava kaskadna funkcija, odnosno model. Nakon uvježbavanja, isti se model koristi za detekciju na drugim slikama.

Algorithm: Viola-Jones Face Detection Algorithm

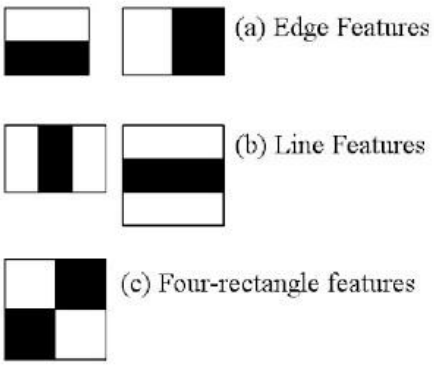
```
Input: original test image
Output: image with face indicators as rectangles
for i ← 1 to num of scales in pyramid of images do
    Downsample image to create image_i
    Compute integral image, image_ii
    for j ← 1 to num of shift steps of sub-window do
        for k ← 1 to num of stages in cascade classifier do
            for l ← 1 to num of filters of stage k do
                Filter detection sub-window
                Accumulate filter outputs
            end for
            if accumulation fails per-stage threshold then
                Reject sub-window as face
                Break this k for loop
            end if
        end for
        if sub-window passed all per-stage checks then
            Accept this sub-window as a face
        end if
    end for
end for
```

Slika 2.1 Pseudokod algoritma Viola-Jones sa Haarovim značajkama [3]

[3] Irgens, Peter, et al. "An efficient and cost effective FPGA based implementation of the Viola-Jones face detection algorithm." *HardwareX* 1 (2017): 68-75.

Viola-Jones algoritam izvodi se u četiri glavna koraka:

- izračunavanje Haarovih značajki
- stvaranje integralne slike
- Adaboost algoritam
- kaskadni klasifikator



Slika 2.2 Vrste Haar značajki [4]

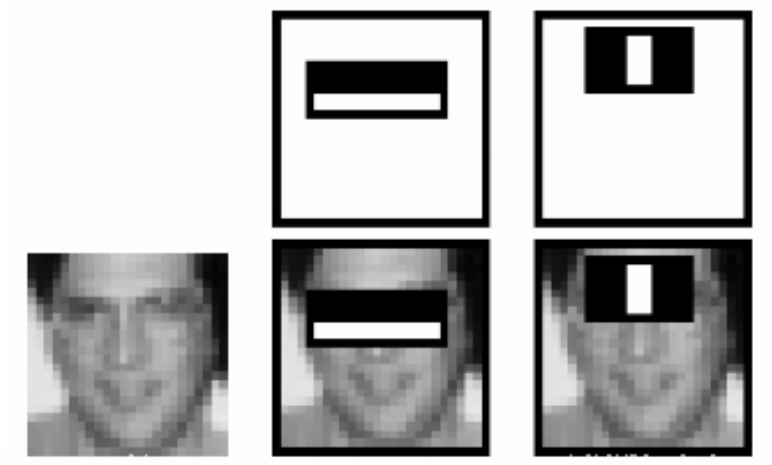
Izračunavanje Haarovih značajki podrazumijeva analizu susjednih pravokutnih područja, zbrajanje inteziteta piksela svakog područja te izračun razlika. Koriste se tri vrste značajki:

- Značajka dva pravokutnika (engl. *Edge features*), odnosno razlika između zbroja piksela unutar dva pravokutnika
- Značajka tri pravokutnika (engl. *Line features*), odnosno zbroj dva vanjska pravokutnika umanjen za zbroj središnjeg pravokutnika
- Značajka četiri pravokutnika (engl. *Four rectangle features*), odnosno razlika između dijagonalnih parova pravokutnika [5][6]

Za izračun svake značajke potrebno je pronaći zbroj piksela dva pravokutnika. Kako bi se proces ubrzao, predstavljena je integralna slika. Značajke pravokutnika moguće je vrlo brzo izračunati koristeći integralnu sliku. Integralna slika svodi izračune piksela originalne fotografije na operaciju koja uključuje samo četiri piksela.

[4] OpenCV https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html
[5] Padilla, R., C. F. F. Costa Filho, and M. G. F. Costa. "Evaluation of haar cascade classifiers designed for face detection." *World Academy of Science, Engineering and Technology* 64 (2012): 362-365.
[6] Viola, Paul, and Michael Jones. "Rapid object detection using a boosted cascade of simple features." *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition.* CVPR 2001. Vol. 1. IEEE, 2001.

Po završetku izračuna svih značajki, većina ih nije od pretjeranog značaja. Primjerice, u gornjem redu slike 1.3 prikazan je primjer dvije dobre značajke. Prva se značajka fokusira na svojstvo da je područje očiju često tamnije u odnosu na područje nosa i obraza. Druga se pak značajka oslanja na svojstvo da su oči tamnije od područja nosa između očiju. Međutim, primjena ovih značajki na bilo kojem drugom području, poput obraza, je beznačajna.



Slika 2.3 Primjeri značajki [7]

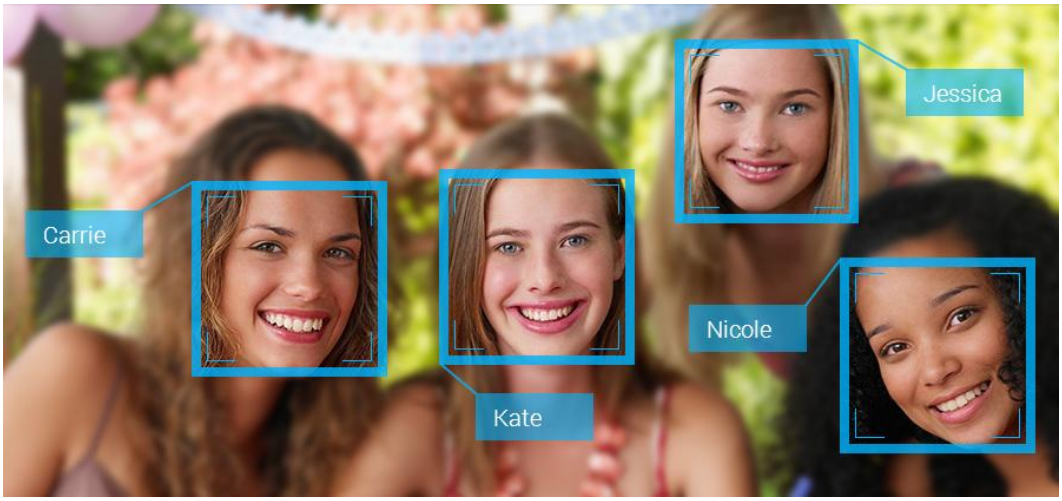
Postavlja se pitanje kako odabrati najbolje značajke. Izbor najboljih značajki postiže se primjenom Adaboost algoritma. U tu svrhu svaka značajka se primjenjuje na svim uvježbanim slikama. Za svaku se značajku pronalazi najbolji prag koji će klasificirati objekte na pozitivne i negativne primjere. Očito je kako pritom nastaju pogrešne klasifikacije. Odabiru se značajke sa najmanjom stopom pogreške, odnosno značajke koje najpreciznije klasificiraju pozitivne i negativne primjere objekta. Obično, na slici je većim dijelom zastupljeno područje koje nije traženi objekt. Primjerice, udaljena slika lica gdje je lice traženi objekt. Stoga je potrebno imati jednostavnu metodu koja provjerava je li odabrano područje ujedno i područje lica. Ukoliko nije, područje se odbacuje i više se ne provjerava. Ako je riječ o području lica, fokusirati se na to područje. Na ovaj način se posvećuje više vremena provjeri moguće regije lica.

Iz tog razloga, uveden je koncept kaskadnog klasifikatora. Umjesto primjene svih značajki na odabranom području, značajke se grupiraju u različite faze klasifikatora te se pojedinačno primjenjuju. Iako već poprilično star, ovaj algoritam se pokazao kao vrlo efikasan pri detekciji u realnom vremenu [8].

[7] OpenCV https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html
[8] Ibid.

2.3 Prepoznavanje lica

Detekcija lica neophodna je kako bi algoritmi prepoznavanja lica znali koje dijelove slike koristiti za generiranje biometrije lica. Taj uzorak kasnije će usporediti sa svakom biometrijom lica sadržanom u ranije uvježbanoj kolekciji lica kako bi se utvrdilo postoji li podudaranje [9]. Prepoznavanje lica je način prepoznavanja ljudskog lica putem tehnologije koristeći biometriju za mapiranje kontura lica sa slike ili videozapisa. Najjednostavnije rečeno ovaj način se odnosi na uspoređivanje informacije sa bazom podataka poznatih lica kako bi se pronašlo podudaranje.



Slika 2.4 Primjer prepoznavanja više lica na slici [10]

Sustavi prepoznavanja lica koriste različite računalne algoritme za odabir specifičnih i prepoznatljivih karakteristika lica osobe. Potom, karakteristike lica se matematički obrađuju te uspoređuju sa podacima o drugim licima prikupljenim u bazi podataka unutar sustava za prepoznavanje lica. Prikupljeni podaci često se nazivaju i predloškom lica i razlikuju se od izvorne slike lica. Ovakvi predlošci dizajnirani su tako da sadrže samo one detalje neophodne za razlikovanje tog lica od ostalih.

Neki sustavi su dizajnirani tako da umjesto identifikacije nepoznate osobe računaju vjerojatnost podudaranja između nepoznate osobe i određenih lica unutar baze podataka. Ovako dizajniran sustav naposlijetku nudi nekoliko potencijalnih podudaranja lica poredanih po redoslijedu vjerojatnosti identifikacije. Sustavi prepoznavanja lica razlikuju se po svojoj spodobnosti prepoznavanja ljudi u ekstremnim uvjetima kao što su loše osvjetljenje, neoptimalan kut gledanja, slike male razlučivosti ili pak niske kvalitete.

[9] Norton https://us.norton.com/internetsecurity-iot-how-facial-recognition-software-works.html
[10] Towards data science https://towardsdatascience.com/face-recognition-for-beginners-a7a9bd5eb5c2

Nerijetka je pojava pogrešnih rezultata nakon procesa prepoznavanja lica. Takvi rezultati prepoznavanja često se manifestiraju u dva oblika:

- lažni negativni, odnosno kada sustav ne uspijeva pronaći podudaranje slike testiranog lica sa slikom lica koje se nalazi unutar baze podataka te kao odgovor na upit pogrešno vraća prazan skup rezultata
- lažni pozitivni, odnosno kada sustav uspijeva pronaći podudaranje slike testiranog lica sa nekom slikom lica koje se nalazi unutar baze podataka te kao odgovor na upit vraća neispravan rezultat, odnosno pogrešno je prepoznato lice koje zapravo ne odgovara stvarnom licu

Prilikom testiranja sustava prepoznavanja lica, vrlo je važno sagledati stopu pojavljivanja „lažnih negativnih“ i „lažnih pozitivnih“ rezultata kako bi se pouzdanost prepoznavanja skalirala na odgovarajuću razinu. Primjerice, kod korištenje sustava prepoznavanja lica za otključavanja zaslona pametnog telefona povoljan je slučaj kada uređaj ne prepoznaje vlasnika iz prvog pokušaja (rezultat „lažno negativan“). Slučaju kada uređaj prepoznaje lice neke druge osobe kao lice vlasnika (rezultat „lažno pozitivan“) omogućava drugim osobama neovlašteno otključavanje vlasnikova uređaja. Sa druge strane, primjena u sudstvu zahtjeva sustave dizajnirane na način da rezultiraju sa što manje „lažno pozitivnih“ rezultata kako bi se pogrešna identifikacija nedužne osobe svela na minimum [11].

U svijetu danas postoji mnogo različitih algoritama prepoznavanja lica a najpoznatiji su:

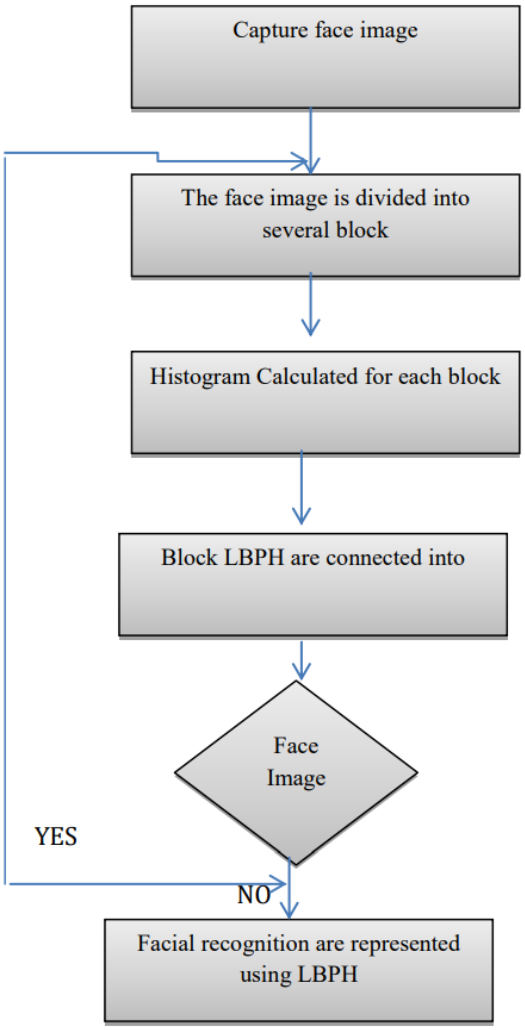
- Eigenfaces
- Fisherfaces
- LBPH
- SIFT
- SURF

Svaki algoritam ima drugačiji pristup pri sakupljanju podataka sa slike te traženju podudaranja sa ulaznom slikom. Međutim, algoritmi Eigenfaces i Fisherfaces imaju sličan pristup, kao i algoritmi SIFT i SURF.

[11] EFF https://www.eff.org/pages/face-recognition

2.3.1 Histogrami lokalnih binarnih značajki

Lokalna binarna značajka (engl. *local binary pattern*, skraćeno LBP) je jednostavan, ali vrlo učinkovit operator obrade teksture koji postavljanjem graničnih vrijednosti susjedstva označava piksele slike a rezultat prikazuje kao binarni broj.



Slika 2.5 Dijagram tijeka LBP algoritma [12]

Prvi je put opisan 1994. godine i od tada se koristi kao moćna značajka pri klasifikaciji teksture. Kasnije je utvrđeno kako LBP u kombinaciji sa HOG rezultira značajnim poboljšanjem performansi detekcije na nekim skupovima podataka. Kombiniranjem LBP sa histogramom, postiže se prikaz slika lica jednostavnim vektorom podataka. LBP se kao vizualni deskriptor može koristiti pri zadacima prepoznavanja lica. U tu svrhu, kreiran je algoritam histograma lokalnih binarnih značajki (engl. *local binary patterns histograms*, skraćeno LBPH).

[12] Khusbu Rani, Sukhbir "Face recognition technique using ICA and LBPH" , IRJET 2017.

Algoritam LBPH sastoji se od pet koraka:

1. parametri
2. uvježbavanje algoritma
3. primjena LBP operacije
4. izdvajanje histograma
5. izvođenje prepoznavanja lica

2.3.1.1 Parametri

Algoritam LBPH koristi četiri parametra:

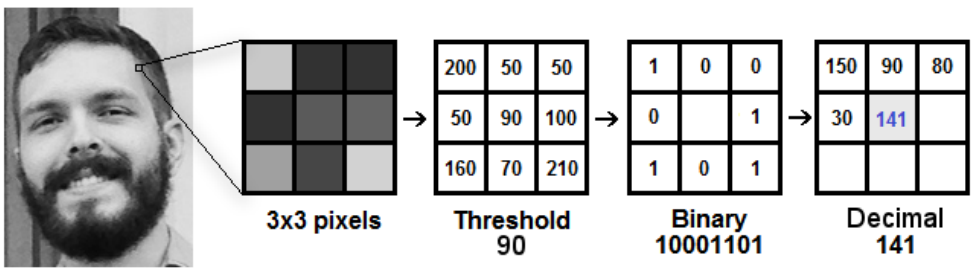
- radijus – koristi se za izradu kružne lokalne binarne značajke i predstavlja radijus oko središnjeg piksela
- susjedstvo – broj točki uzorka pri izradi kružne lokalne binarne značajke
- os x – broj vodoravnih ćelija
- os y – broj okomitih ćelija

2.3.1.2 Uvježbavanje algoritma

Kako bi se algoritam uvježbao potrebno je prvenstveno imati skup podataka sa slikama lica ljudi koje želimo prepoznati. Također je potrebno postaviti identifikator (engl. *identifier*, skraćeno ID) za svaku sliku. Algoritam koristi taj ID pri prepoznavanju ulazne slike te pri davanju izlaznog rezultata. Slike iste osobe moraju imati isti ID.

2.3.1.3 Primjena LBP operacije

Prvi korak izvršavanja LBPH algoritma jest stvaranje srednje slike koja će isticanjem karakteristika lica bolje opisati izvornu sliku. Da bi to učinio, algoritam se koristi konceptom kliznog prozora zasnovanog na parametrima radijusa i susjedstva. Slika 1.6 prikazuje ovaj postupak.



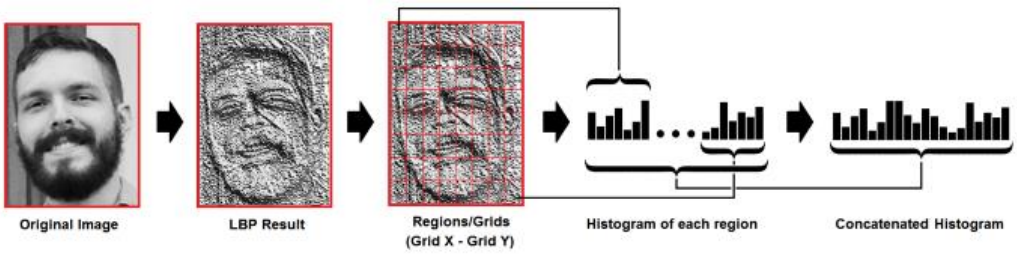
Slika 2.6 Proces stvaranja srednje slike [13]

[13] Towards data science https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b

Primjerice, na ulazu je slika u sivim tonovima. Dio ove slike može se dobiti u obliku prozora veličine 3x3 piksela. Također, dio slike se može predstaviti i kao 3x3 matrica koja sadrži intezitet svakog pojedinog piksela na ljestvici od 0 do 255. Zatim je potrebno uzeti središnju vrijednost matrice koja će se koristiti kao prag. Ova se vrijednost koristi pri definiranju novih vrijednosti za svaki od osam susjeda. Svakom susjedu središnje vrijednosti, odnosno praga (engl. *threshold*), postavlja se nova binarna vrijednost. Postavlja se 1 za vrijednosti jednake ili veće od praga, odnosno 0 za vrijednosti niže od praga. Nakon toga, matrica sadrži samo binarne vrijednosti, izuzev središnje vrijednosti. Nadalje, potrebno je spojiti svaku binarnu vrijednost sa svake pozicije u matrici, redak po redak, u novu binarnu vrijednost (primjerice 10001101). Treba napomenuti kako smjer spajanja nije bitan ukoliko se od početka do kraja primjenjuje ista analogija. Pred sam kraj, binarna vrijednost pretvara se u decimalnu te se postavlja na središnju vrijednost matrice koja je zapravo piksel izvorne slike. Rezultat ovog LBP postupka je nova slika koja bolje opisuje karakteristike izvorne slike.

2.3.1.4 Izdvajanje histograma

Izdvajanju histograma pristupa se korištenjem generirane slike iz prethodnog koraka. Sliku je potrebno rešetkasto podijeliti korištenjem parametara osi x te osi y. Primjer sa slike 1.7 prikazuje način izdvajanja histograma svake regije.



Slika 2.7 Izdvajanje histograma [14]

Kako je riječ o slici u sivim tonovima, histogram svake regije (engl. *histogram of each region*) sadrži 256 pozicija koji predstavljaju pojavu svakog inteziteta piksela. Stvaranje novih i većih histograma postiže se uzastopnim spajanjem pojedinog histograma. Konačan histogram predstavlja karakteristike izvorne slike.

[14] Towards data science https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b

2.3.1.5 Izvođenje prepoznavanja lica

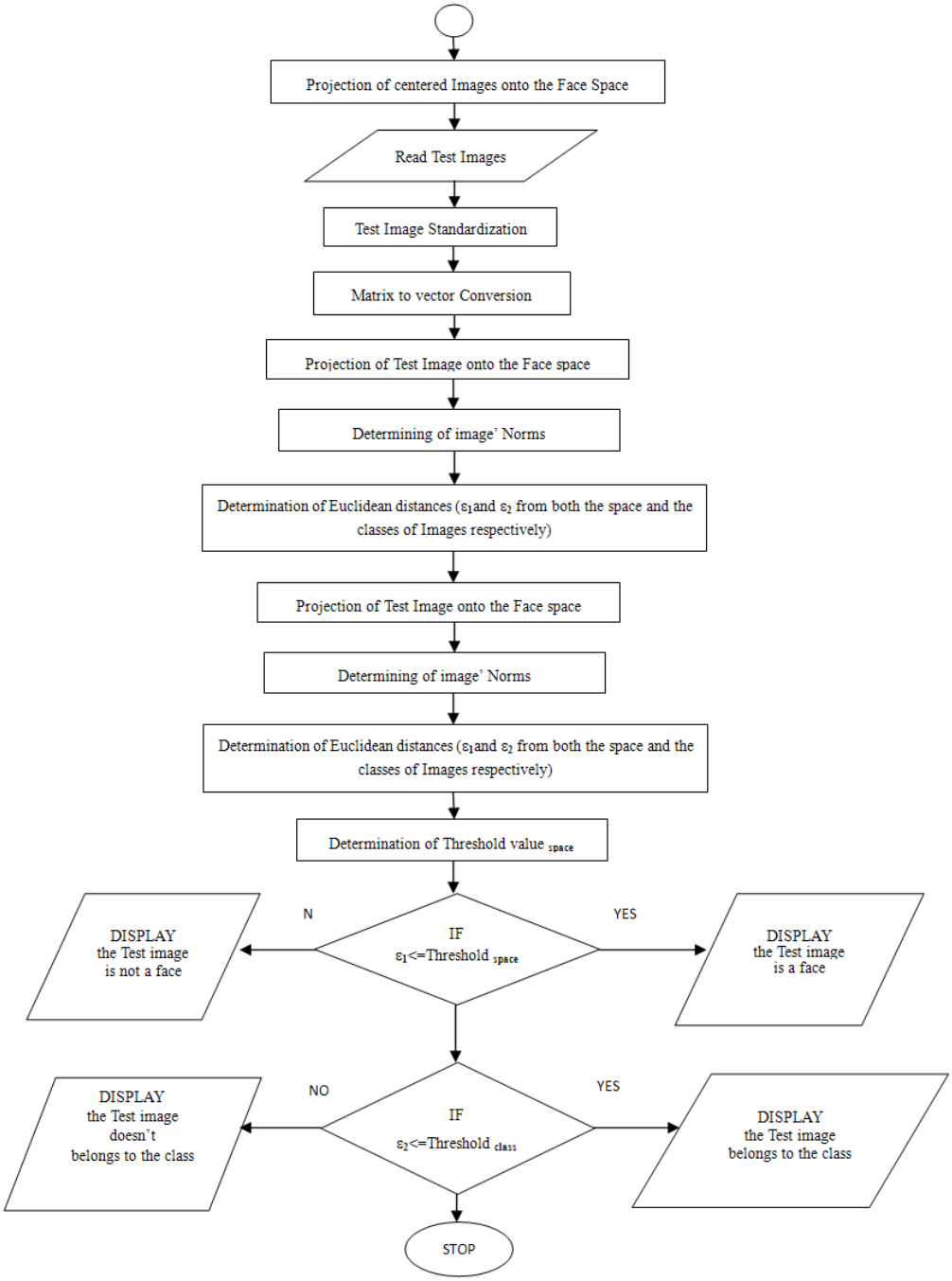
Do ovog koraka, algoritem je već uvježban. Svaki stvoreni histogram predstavlja pojedinu sliku unutar uvježbanog skupa podataka. S obzirom na ulaznu sliku, za istu je potrebno ponovno izvršiti prethodne korake te kreirati histogram koji će ju predstavljati. Kako bi se pronašla slika koja odgovara ulaznoj slici, potrebno je usporediti dva histograma i vratiti sliku sa najbližim histogramom. Mogu se koristiti razne metode pri usporedbi histograma. Primjerice, euklidska udaljenost, hi-hvadrat, apsolutna vrijednost i druge metode. Po pronalasku najbližeg histograma, algoritam vraća njegov ID. Algoritam bi također trebao vratiti i izračunatu udaljenost koja se može koristiti kao mjera pouzdanosti. Pri tome je bitno napomenuti kako je niža pouzdanost bolja jer predstavlja manju udaljenost između dva histograma. Vrijednost praga i pouzdanost mogu se koristiti za automatsku procjenu ispravnosti prepoznavanja slike. Može se pretpostaviti kako je algoritam uspješno preopoznao ako je pouzdanost niža od definiranog praga [15][16].

LBPH jedan je od najjednostavnijih algoritama za prepoznavanje lica. Sposoban je predstavljati lokalne značajke na slikama. U kontroliranom okruženju mogu se postići izvrsni rezultati. Algoritam je otporan na monotone transformacije sive ljestvice te je podržan unutar OpenCV biblioteke.

[15] Dr. G.S.Anandha Mala , A. Vijay Akash , S. Shobika "Tracking of Criminal Through CCTV Using Face Detection" *International Journal of Advanced Science and Technology*, v. 29, n. 06, p. 6542 - 6551, 2020.
[16] Towards data science https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b

2.3.2 Eigenfaces

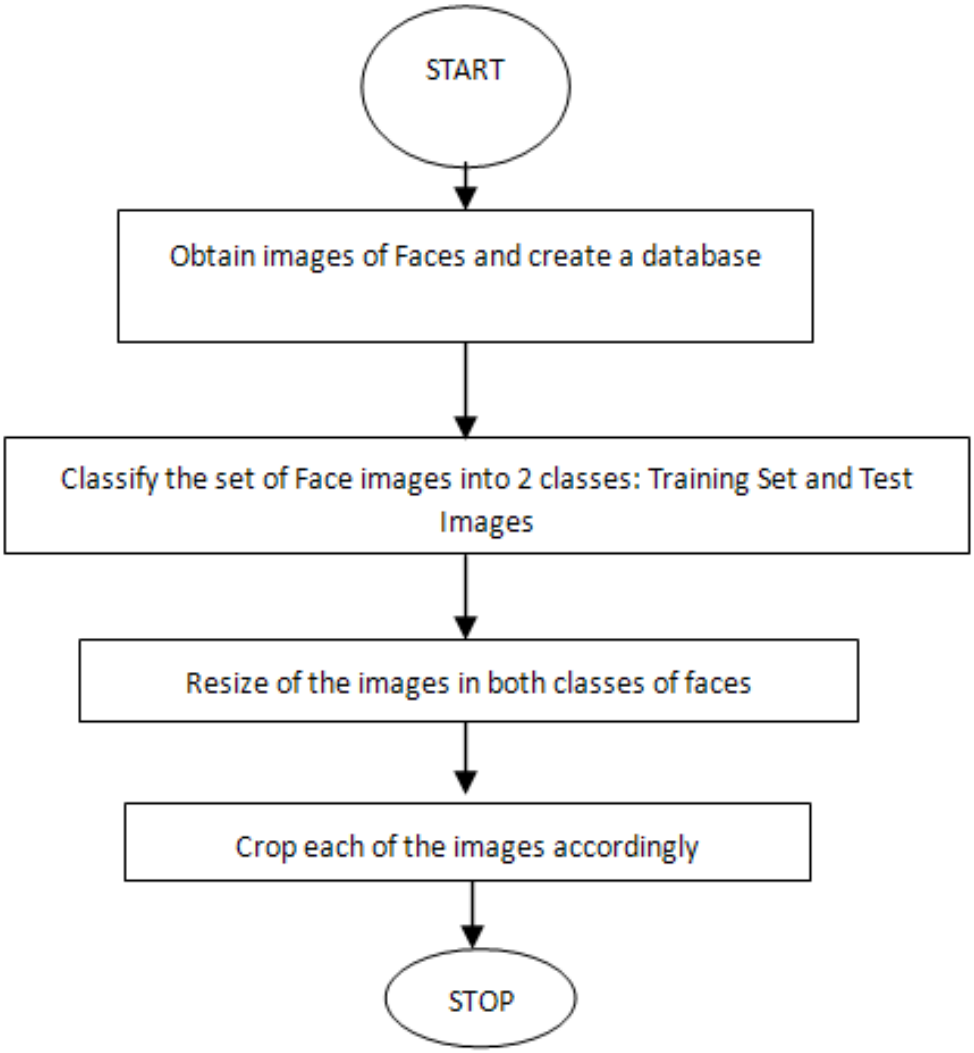
Eigenfaces jedan je od prvih a ujedno i najčešće korištenih algoritama prepoznavanja lica. Nastao je 1987. godine od strane inženjera Matthew Turk te informatičara Alex Pentland. Značajka ovog algoritma leži u istoimenoj značajci svojstvenog lica (engl. *eigenface*), odnosno nizu svojstvenih vektora koji su pogodni pri rješavanju problema prepoznavanja lica.



Slika 2.8 Dijagram tijeka prepoznavanja lica [17]

[17] Scientific & Academic Publishing http://article.sapub.org/10.5923.j.ajsp.20150502.02.html#Sec1

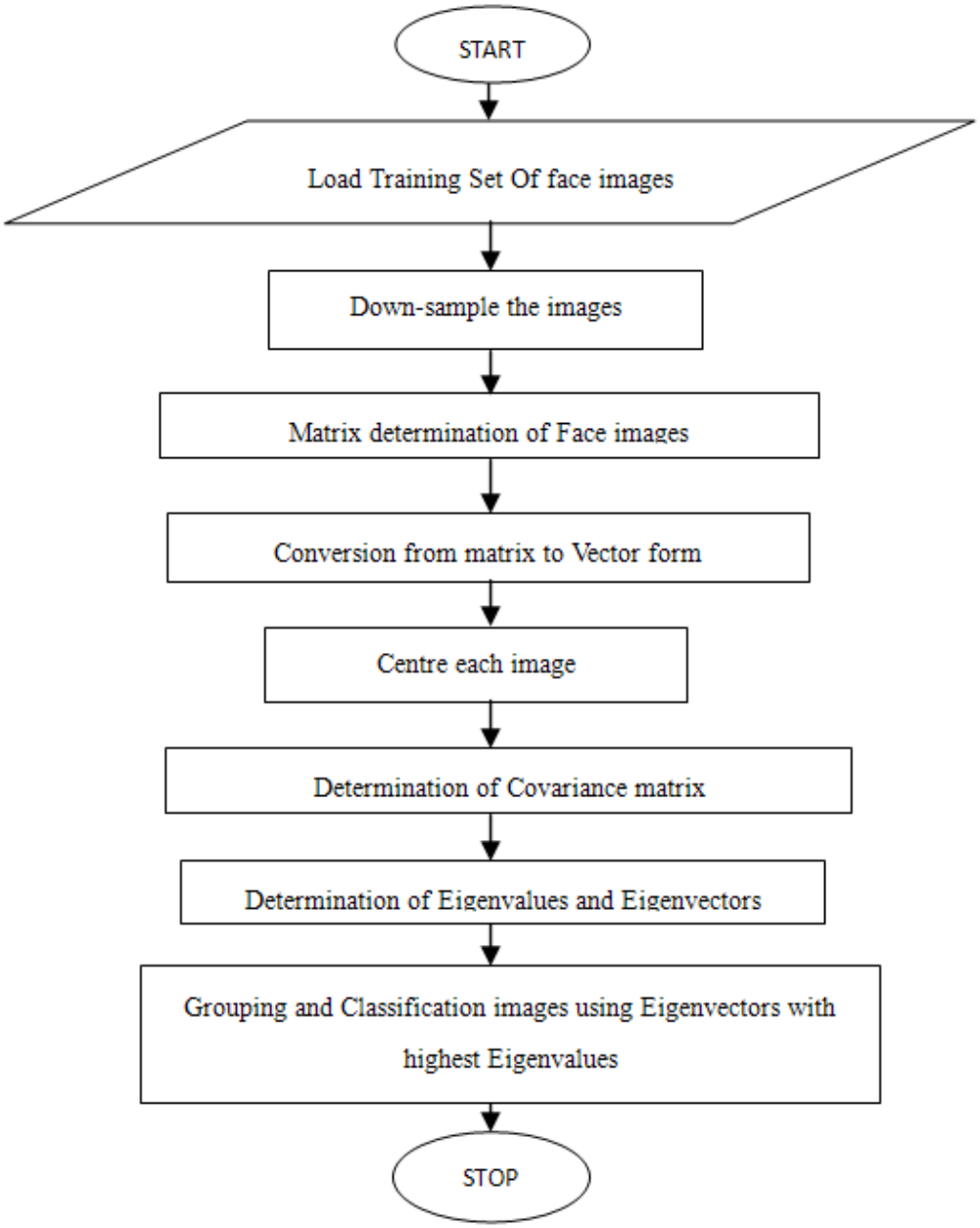
Kako bi se kreirao zadani niz vektora, potrebno je sakupiti velik uzorak slika lica. Pritom je važno da su sve slike uzorka jednake svjetline i veličine te da se elementi očiju i usana na svim uzorcima poravnati.



Slika 2.9 Dijagram tijeka predobrade [18]

[18] Scientific & Academic Publishing http://article.sapub.org/10.5923.j.ajsp.20150502.02.html#Sec1

Svojstveno lice sastoj se od područja različitog osvjetljenja. Algoritam ocjenjuje svjetlija područja lica. Karakteristike koje se ocjenjuju između ostalog su: simetrija lica, visina čela, veličina usana i nosa.



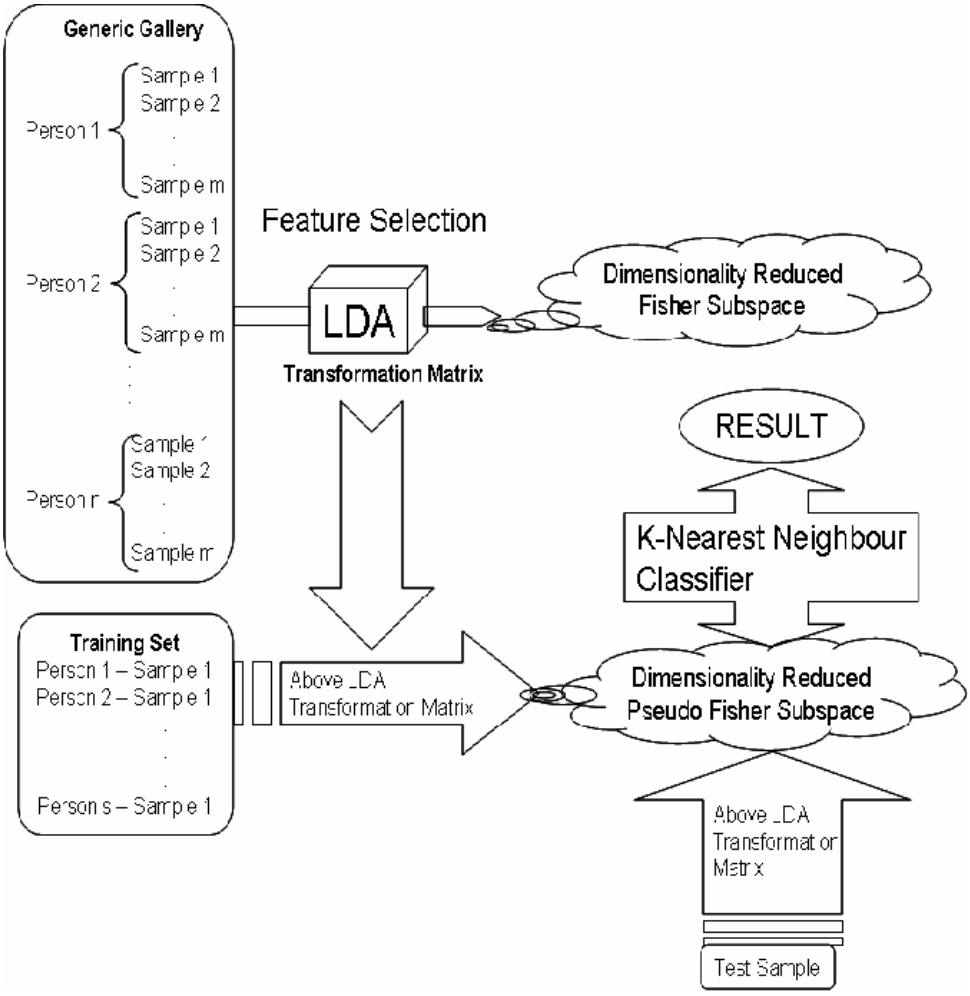
Slika 2.10 Dijagram uvježbavanja slika lica [19]

Suvremeni algoritmi koji se baziraju na svojstvenom licu su puno brži i efikasniji u odnosu na ostale algoritme. Najveći nedostatak ovo algoritma jest ovisnost o jednakom osvjetljenju, položaju glave te poravnanju među svim slikama [20].

[19] Scientific & Academic Publishing http://article.sapub.org/10.5923.j.ajsp.20150502.02.html#Sec1
[20] Turk, Matthew, and Alex Pentland. "Eigenfaces for recognition." *Journal of cognitive neuroscience* 3.1 (1991): 71-86.

2.3.3 Fisherfaces

Fisherfaces je uz Eigenfaces jedan od najpopularnijih algoritama prepoznavanja lica. Ovaj algoritam uvjetuje poznavanje Eigenfaces algoritma iz razloga što se nadovezuju jedan na drugi. Dok se Eigenfaces algoritam koristi analizom glavnih komponenti pri prepoznavanju lica, Fisherface algoritam koristi Fisherfaces linearnu diskriminantnu analizu. Fisherfaces analiza temelji se na svođenju d-dimenzionalnog vektora značajki na jednu dimenziju, što je potrebno kako bi se izvršila klasifikacija [21].

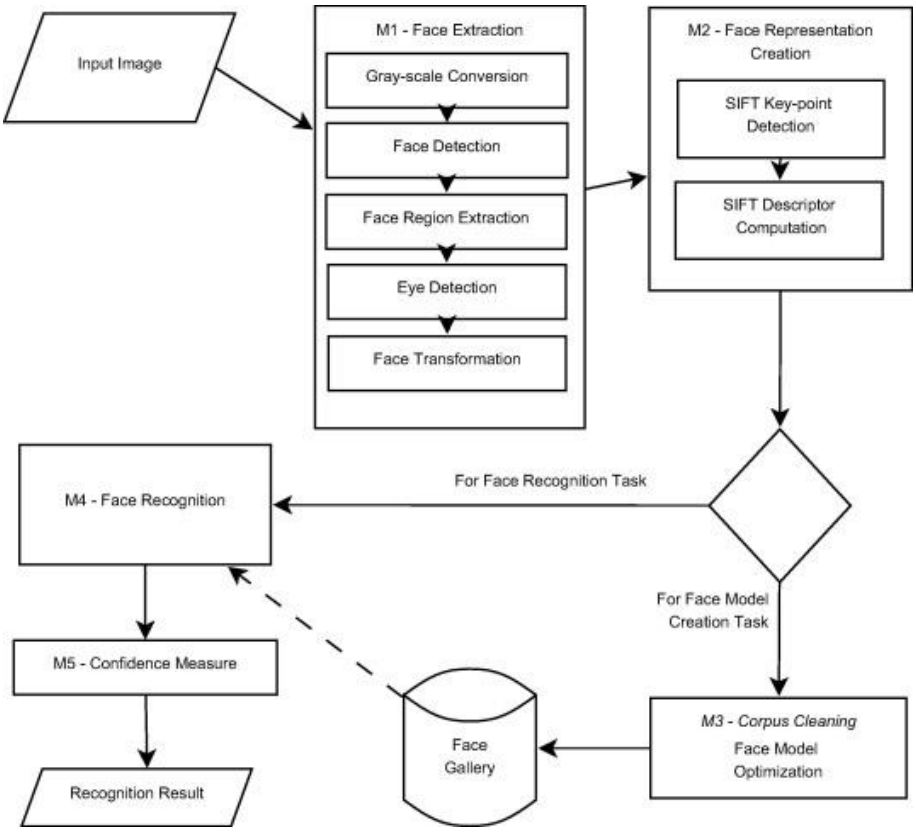


Slika 2.11 Dijagram tijeka prepoznavanja lica Fisherfaces algoritmom [22]

[21] Anggo, Mustamin, and La Arapu. "Face recognition using fisherface method." *Journal of Physics* 1028 (2018).
[22] Majumdar, A. and R. Ward. "Pseudo-Fisherface method for single image per person face recognition." *2008 IEEE International Conference on Acoustics, Speech and Signal Processing* (2008): 989-992.

2.3.4 Transformacija značajki neovisne o skali

Transformacija značajki neovisne o skali (engl. *scale-invariant feature transform*, skraćeno SIFT) je algoritam računalnog vida koji se koristi pri traženju i opisivanju lokalnih značajki na slici. Algoritam je prvotno razvio David Lowe 1999. godine a 2004. godine izdana je poboljšana verzija. Jedne od najbitnijih značajki ovog algoritma su invarijantnost pri rotaciji i promjeni veličine slike te dobra otpornost na promjene osvjetljenja, zaklonjenost objekta i promjene točke gledišta. Ovaj algoritam krasi i sposobnost uspoređivanja sa velikim bazama podataka u realnom vremenu. SIFT se koristi za različita prepoznavanja objekata, najčešće lica te pri trodimenzionalnom rekonstrukciji različitih vrsta scena [23].



Slika 2.12 Dijagram tijeka prepoznavanja lica korištenjem SIFT algoritma [24]

[23] Wikipedia https://en.wikipedia.org/wiki/Scale-invariant_feature_transform

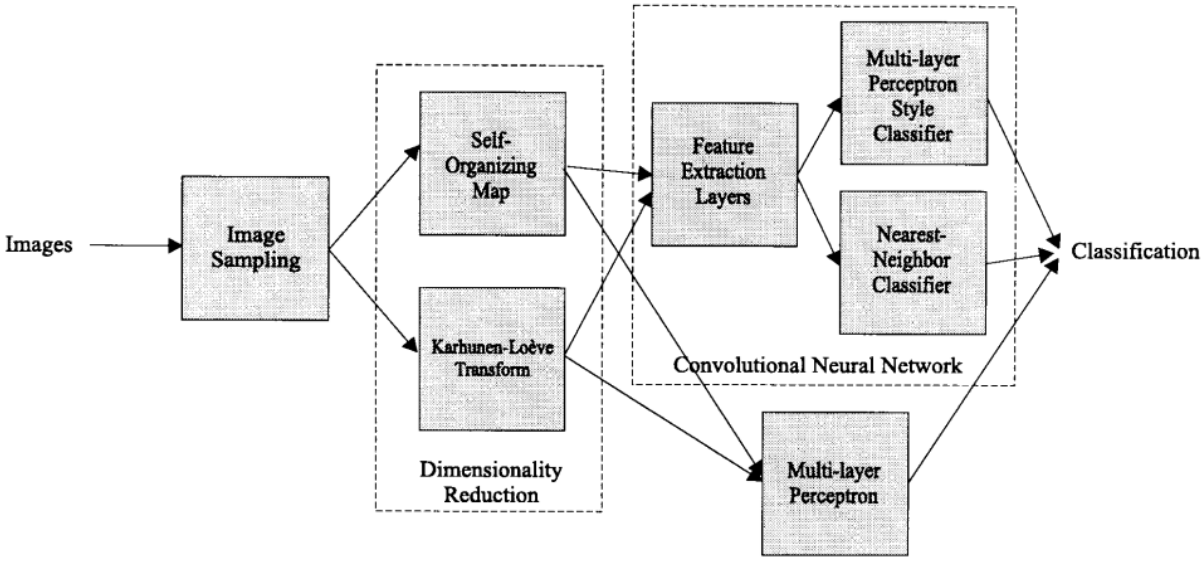
[24] Lenc, Ladislav, and Pavel Král "Automatic face recognition system based on the SIFT features." *Computers & Electrical* Engineering 46 (2015): 256-272.

2.3.5 Konvolucijska neuronska mreža

Konvolucijska neuronska mreža (engl. *convolutional neural network*, skraćeno CNN) je klasa dubokih neuronskih mreža koja se najčešće primjenjuje za analizu vizualnih slika, u području dubokog učenja. CNN u velikoj su mjeri zastupljene u području računalnog vida. Najčešće se koriste kod detekcije objekata i prepoznavanja lica. Ovakva neuronska mreža sastoji se od tri glavna neuronska sloja:

- konvolucijski sloj – generiranje značajki
- sloj udruživanja – redukcija prostornih dimenzija ulaznih podataka
- potpuno povezani sloj – zaključivanje na visokoj razini

Uloga svakog pojedinog sloja različita. No, svaki sloj pretvara ulazne podatke u izlazne podatke. Po pretvaranju istih, aktivira se neuronska mreža. U konačnici, dolazi do potpunog povezivanja slojeva te preslikavanja podataka u vektore [25].



Slika 2.13 Dijagram tijeka prepoznavanja lica korištenjem CNN [26]

[25] Lawrence, Steve, et al. "Face recognition: A convolutional neural-network approach." *IEEE transactions on neural networks* 8.1 (1997): 98-113.
[26] Ibid.

3. ODABIR ALATA

U svrhu realizacije rješenja problema prepoznavanja lica na slici, što je konkretno zadatak ovog rada, neophodno je koristitit odgovarajuće razvojno okruženje. S obzirom da je tema ovog rada vezana uz računalni vid, potrebno je pronaći i koristiti odgovarajuće biblioteke sa podrškom za razvoj aplikacija temeljenih na računalnom vidu.

Za realizaciju rješenja odabran je programski jezik Python. Prvenstveno zbog jednostavne i čiste sintakse a potom i zbog bogate kolekcije podržanih biblioteka. Kao razvojno okruženje odabran je alat Microsoft Visual Studio Code IDE. Ovo razvojno okruženje odabrano je iz više razloga. Visual Studio Code je interaktivan i pregledan, ne zahtijeva instalaciju te nudi brzu i jednostavnu instalaciju dodataka putem terminala. OpenCV je odabran kao biblioteka sa podrškom za računalnu vid. Ova biblioteka koristi jednostavnu sintaksu i sadrži veliku broj algoritama. Ovaj rad ponudit će rješenje problema primjerom korištenja jedne klasične i jedne moderne metode. Klasičan primjer izrađen je kroz OpenCV biblioteku a za realizaciju modernog rješenja odabrana je biblioteka Tensorflow. Iako je riječ o vrlo moćnom alatu, Tensorflow je poslužio tek kao podrška za Keras u ovom projektu. Keras i njegova konvolucijska neuronska mreža korištene su pri realizaciji modernog rješenja problema.

3.1 Visual Studio Code

Visual Studio Code je tekstualni uređivač i proizvod tvrtke Microsofta. Dostupan je na operacijskim sustavima Windows, Linux i macOS operacijske sustave. Uključuje podršku za ispravak pogrešaka u kodu, označavanje sintakse, inteligentno nadopunjavanje te preuređivanje koda. Besplatan je za korištenje i alat je otvorenog koda. Visual Studio Code je prilagodljiv što znači da krajnji korisnik može personalizirati izgled i funkcionalnosti alata prema vlasitim željama i potrebama. Alat također omogućava jednostavnu instalaciju proširanja koja pružaju dodatne funkcionalnosti. Visual Studio Code je fleksibilan, brz i pouzdan te pruža podršku za širok spektar programski jezika kao što su Python, C#, Java i drugi [27]. Upravo zahvaljujući svojim mogućnostima, ovaj alat je odabran kao razvojno okruženje ovog rada.

[27] Visual Studio Code https://code.visualstudio.com/

3.2 OpenCV

OpenCV je biblioteka programske podrške otvorenog koda za računalni vid, strojno učenje te obradu slike. Stvorena je kako bi pružila jedinstven pristup razvoju aplikacija računalnog vida te popularizirala upotrebu strojne percepcije. Projekt OpenCV službeno je pokrenula tvrtka Intel Research 1999. godine koju je predvodio Intelov tim stručnjaka za optimizaciju u Rusiji. Biblioteka danas sadrži više od 2500 optimiziranih algoritama, uključujući i skup klasičnih i najsuvremenijih algoritama za računalni vid i strojno učenje. Ovi se algoritmi mogu koristiti za otkrivanje i prepoznavanje lica, prepoznavanje objekata, klasificiranje ljudskih radnji u videozapisima, praćenje objekata u pokretu, izradu 3D modela predmeta, spajanje više fotografija u jednu kako bi se dobila šira slika visoke rezolucije, pronalaženje slične slike u bazi podataka, uklanjanje efekta crvenih očiju sa fotografija snimljenih bljeskalicom, detekciju usmjerenosti pogleda oka i slično. OpenCV ima više od 47 tisuća korisnika zajednice i preko 18 milijuna preuzimanja. Biblioteka je danas dostupna na gotovo svim računalnim platformama te je besplatna za upotrebu u komercijalne i akademske svrhe [28].

3.3 TensorFlow i Keras

Uska je povezanost TensorFlowa i Kerasa. TensorFlow je u potpunosti platforma strojnog učenja otvorenog koda. Najbolje ju je promatrati kao jedinstven infrastrukturni sloj za diferencijabilno programiranje. Četri su ključne značajke koje sadrži ova platforma:

- Efikasno izvršavanje operacija niske razine tenzora na CPU, GPU, ili TPU.
- Računanje gradijenta proizvoljnih diferencijabilnih izraza
- Skaliranje računanja na više uređaja
- Izvoz programa i grafova na vanjski tijek izvršavanja

Keras je API za TensorFlow i sučelje za rješavanje problema strojnog učenja sa naglaskom na moderno duboko učenje. Pruža važne apstrakcije i elemente potrebne za razvoj i isporuku rješenja strojnog učenja. Inženjerima i istraživačima omogućuje da u potpunosti iskoriste značajke skalabilnosti i dostupnosti na više platformi. Keras je kompatibilan sa TPU i GPU klasterima te nudi mogućnost izvoza Keras modela za pokretanje u pregledniku ili na pametnom telefonu [29][30].

[28] OpenCV https://opencv.org/about/
[29] Tensorflow https://www.tensorflow.org/about
[30] Keras https://keras.io/

3.3.1 TensorFlow

TensorFlow je jedna od najjednostavnijih i najpraktičnija platforma otvorenog koda koja se koristi za strojno učenje. Osnovan je od strane Google Brain tima 2011. godine u Google centru a prvotno je nazivan DistBelief.

Karakterizira ga sveobuhvatan i fleksibilan sustav alata, biblioteka te resursa čitave zajednice koja doprinosi korištenju najnovije tehnologije strojnog učenja. Zajednica omogućava i razvojnim inženjerima jednostavnu izradu i implementaciju aplikacija zasnovanih na strojnom učenju. Prvotno se koristio pri strojnom učenjenju kako prepoznao karakteristične uzorke na slikama. Prilikom obrade koristio se pozitivnim pojačanjem prihvaćajući samo točne rezultate. U slučaju pogrešnog rezultata sustav se prilagođavao kako bi mogao dodatno prepoznati i neki drugi uzorak na slici. Na ovaj način poboljšan je broj točnih rezultata. Kasnije, Tenserflow nasljeđuje taj koncept kroz duboko učenje odnosno konvolucijsku neuronsku mrežu. Prva verzija pod nazivom TensorFlow dostupna je od 11. veljače 2017. Naziv TensorFlow leži načinjenici da za računanje koristi multidimenzionalne matrice koje se nazivaju tenzorima [31].

Tensorflow je poslužio razvoju čitavog niza biblioteka koje služe za rješavanje konkretnih problema zasnovanih na neuronskim mrežama, između ostalih i Keras biblioteke koja se koristi u ovom radu. Tenserflow ne igra veoma bitnu ulogu u ovom radu te se više koristi u pozadini.

3.3.2 Keras

Keras je biblioteka ili API za duboko učenje. Napisan je na programskom jeziku Python. Dostupan je kao biblioteka u sklopu platforme za strojno učenje TensorFlow a može se koristiti zajedno sa Microsoft Cognitive Toolkit, Theano ili MXNet. Osmišljen je i razvijen kako bi omogućio brzo izvođenje eksperimenata. Kreatori Kerasa ističu kako je ključ dobrog istraživanje što prije prijeći sa ideje na rezultate [32].

U ovom radu korištena je Keras biblioteka u svrhu pred-procesiranja ulaznih podataka te u svrhu prepoznavanja lica, odnosno detekciju nošenja zaštitne maske.

[31] Wikipedia https://en.wikipedia.org/wiki/TensorFlow
[32] Wikipedia https://en.wikipedia.org/wiki/Keras

4. REALIZACIJA RJEŠENJA

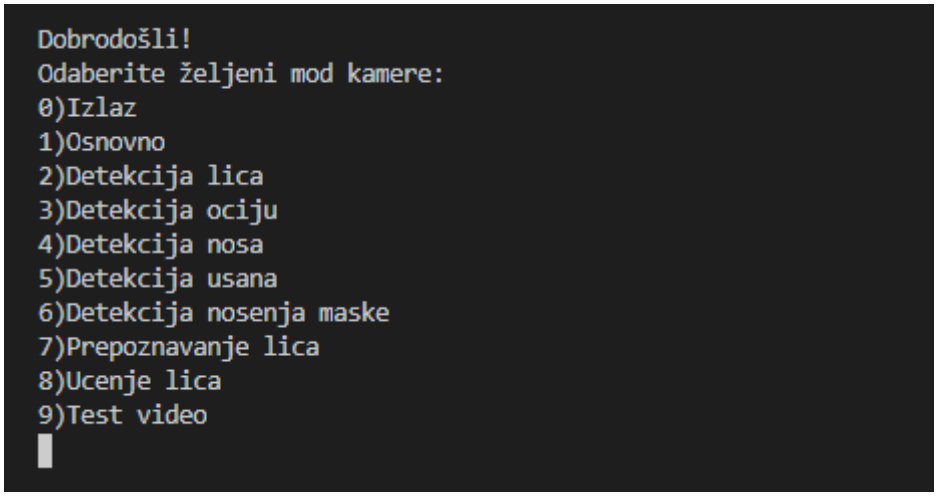
Ideja demonstracije rješenja problema prepoznavanja lica na slici zamišljena je kroz realizaciju jednostavne aplikacije. Odlučeno je kako je jednostavna aplikacija najprikladnije rješenje iz više razloga. S obzirom da je ovaj rad osmišljen kako bi riješio problem prepoznavanja lica na slici, prvenstveno je potrebno prikazati složenost realizacije rješenja tog problema. Iz tog razloga je nepotrebno rješenju pridodavati dodatne značajke kao što su primjerice transformacije lica, uklanjanje pozadine i slične. Uz to, zamišljeno je da rad bude razumljiv ne samo iskusnim čitateljima nego i onima koji to tek žele postati. Cilj ovog rada jest da se uz pomoć računalne tehnologije kreira jednostavan nativan sustav koji će kao ulazni podatak imati proizvoljnu digitalnu sliku a kao izlaz originalnu sliku obogaćenu značajkom prepoznavanja. Nativan pristup rješavanju problema izuzetno je važan. I ne samo iz razloga što će digitalna imitacija načina ljudskog razmišljanja i donošenja odluka pozitivno utjecati na razumijevanje procesa prepoznavanja kod pojedinca. Već i zbog činjenice da je takav pristup primjene temelj gotovo svih tehnoloških dostignuća kako kroz povijest tako i današnjice.

Aplikacija opisana ovim radom zasniva se na web kameri ili bilo kakvoj vrsti kamere povezanoj sa samom aplikacijom. Odabrana je web kamera ispred obične slike zbog toga što web kamera nudi prikaz u realnom vremenu. Dinamičniji prikaz pridonosi jasnijem prikazuje novonastalih promjena. Krajnji korisnik ovako može testirati efikasnost rezultata mijenjanjem pozicije, nagiba, udaljenosti lica kao i mijenjanjem razine osvjetljenja okoline pa i samog lica.

Funkcionalnost aplikacije opisana je kroz devet načina rada kamere:

1) Osnovni
2) Detekcija lica
3) Detekcija očiju
4) Detekcija nosa
5) Detekcija usana
6) Detekcija nošenja maske
7) Prepoznavanje lica
8) Učenje lica
9) Test video

Demonstracija rješenja podijeljena je na ove načine rada kamere kako bi se korisnik mogao postepeno upoznavati i biti ukorak sa tijekom izvršavanja pojedine faze. Prilikom pokretanja aplikacije, prikazuje se ispis na konzoli (slika 4.1) te slika kamere. Kako korisnik mijenja ili uključuje načine rada kamere sa većim predznakom tako rješenje postaje složenije. Ovakav dizajn aplikacije, gdje korisnik pritiskanjem odgovarajućeg broj aktivira pripadajući način rada kamere, konstruiran je upravo iz razloga da se korisniku omogući što veća manipulacija. Sve u svrhu boljeg testiranja rješenja.



Slika 4.1 Početni izged konzole

Prikaz kamere uz sliku, sadrži i nekoliko tekstualnih oznaka. Ove oznake služe kako bi informirale korisnika o trenutačnim aktivnostima. Krajnje gornja oznaka izvještava korisnika o posljednje poduzetoj akciji u smislu trenutno aktivnih načina rada kamere. U nekim načinima rada prisutna je i druga oznaka. Ova oznaka koristi se kao brojač i prebrojava sve značajke pojedinog načina. Za prikaz je odabrana žarko zelena boja kako bi se oznake što bolje istjecale na različitim slikama.

4.1 Osnovno

Osnovni način rada odnosi se na običan prikaz kamere te kao takav nije prilično zanimljiv. Ovim se načinom željela pokazati prva faza rješavanja problema. Ulazni i izlazni podaci, odnosno niz digitalnih slika na ulazu i izlazu, osigurani su ovim načinom. Izgled osnovnog načina rada kamere prikazan je na slici 4.2.

Slika 4.2 Izgled osnovnog načina rada

Očito je kako su kod ovog načina rada ulazne slike jednake izlaznim slikama. Po prelasku sa drugog načina rada kamere na ovaj, brišu se sve do tog trenutka aktivne značajke.

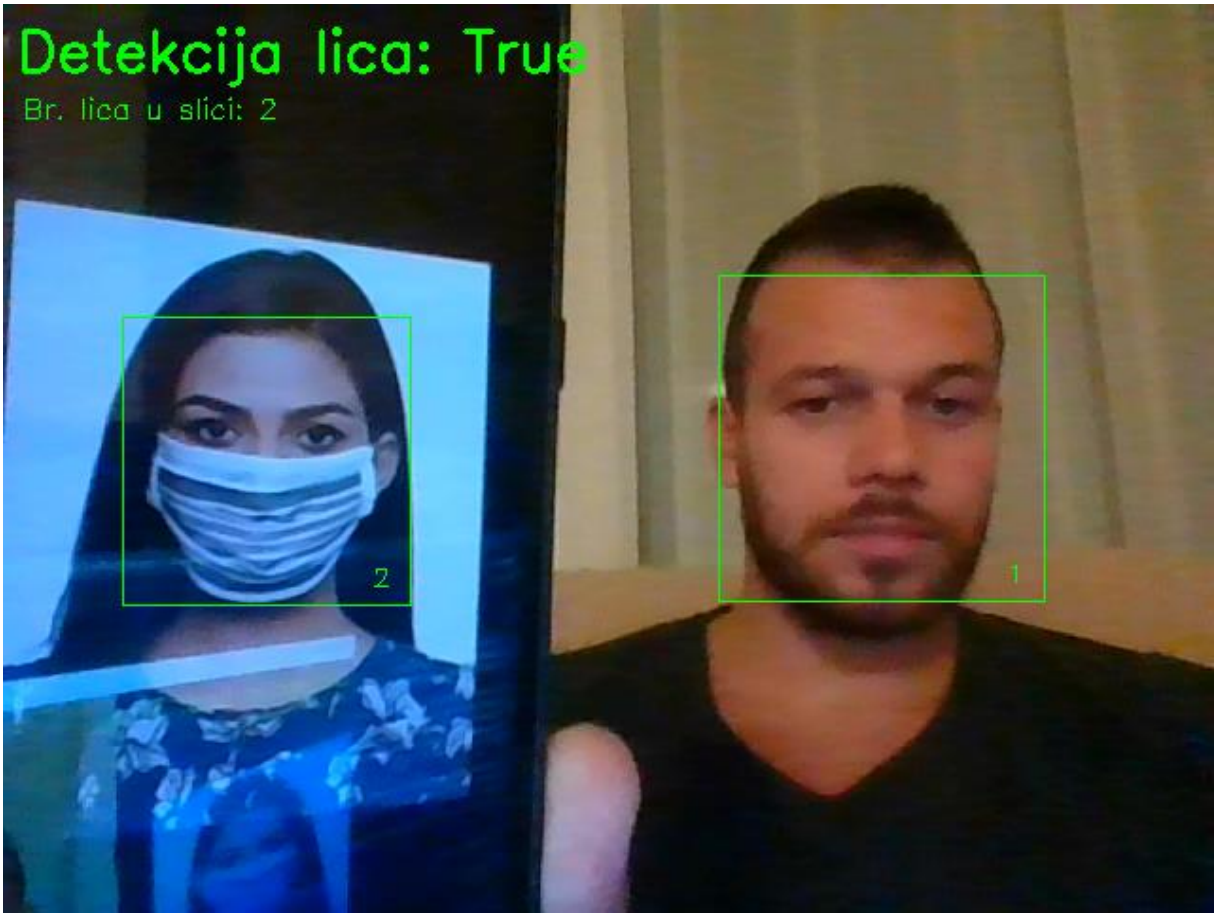
```
1.  import cv2
2.  cap = cv2.VideoCapture(0)
3.  while True:
4.      if cv2.waitKey(1) == ord('0'):
5.          break
6.      ret, frame = cap.read()
7.      cv2.imshow('frame', frame)
8.  cap.release()
9.  cv2.destroyAllWindows()
```

Kôd 4.1 „Kostur“

Ovaj način rada se može nazvati i „kosturom“ aplikacije jer je osnova drugim načinima rada.

4.2 Detekcija lica

Detekcija lica odnosi se na prikaz kamere sa značajkom pronalaska lica na slici. Ovim se načinom željela pokazati druga faza rješavanja problema. Ukoliko je lice pronađeno, isto će biti istaknuto zelenim kvadratom. Ovakvi kvadrati ističu obrube prepoznate regije lica.



Slika 4.3 Detekcija lica

Također, istaknut je broj unutar prepoznate regije. Ovaj broj predstavlja poredak prepoznate regije u odnosu na ostale regije, ukoliko je detektirano više lica. Ovaj način rada kamere, kao i svi ostali načini opisani ovim radom, sposoban je prikazati značajke na proizvoljnom broju lica. Odnosno, prikaz značajki nije ograničen isključivo na jedno lice.

Slika 4.3 prikazuje izgled aplikacije sa uključenim načinom rada za detekciju lica. Vidljivo je kako su u situaciji na slici uspješno prepoznate dvije regije lica. Zanimljivo je primjetiti kako je desno lice detektirano prije lijevog lica. Također, zanimljivo je kako je uspješno detektirano lice koje nosi zaštitnu masku. Razlog tomu leži u odabiru algoritma pretrage te uvježbanog modela detekcije lica.

Da bi ovaj način rada mogao uspješno detektirati lice potrebno je koristiti uvježbani model. Kako je riječ o detekciji lica nužno je koristiti i model zasnovan na licu. Za potrebu ovog rada, kao model detekcije lica korišten je popularni kaskadni model frontalnog lica sa Haar značajkom. Prvotno, ideja je bila kreirati vlastiti model. Međutim, ranijim navodima o tome kako je za dobro uvježban algoritam potrebna pozamašna računalna moć, odlučeno je kako će se ipak koristiti pouzdan i provjeren model.

```
1.  face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

Kôd 4.2 Učitavanje modela za detekciju lica

OpenCV biblioteka podržava korištenje gotovih modela. Za učitavanje modela koristi se metoda *CascadeClassifier* na način prikazan u kôdu 4.2. S obzirom da je model zasnovan na kaskadama, iste je potrebno kaskadno klasificirati.

```
1.  gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
2.  faces = face_cascade.detectMultiScale(gray, scaleFactor = 1.3, minNeighbors = 5, min
    Size = (50, 50))
3.  labele[2] = 0
4.  for (x, y, w, h) in faces:
5.      labele[2] += 1
6.      if lice:
7.          cv2.rectangle(frame, (x,y), (x+w, y+h), (0, 255, 0), 1)
8.          cv2.putText(frame, str(labele[2]), (x+w-20, y+h-
    10), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 255, 0), 1)
9.
10. if lice:
11.         labele[1] = "Br. lica u slici: " + str(labele[2])
```

Kôd 4.3 Detekcija lica

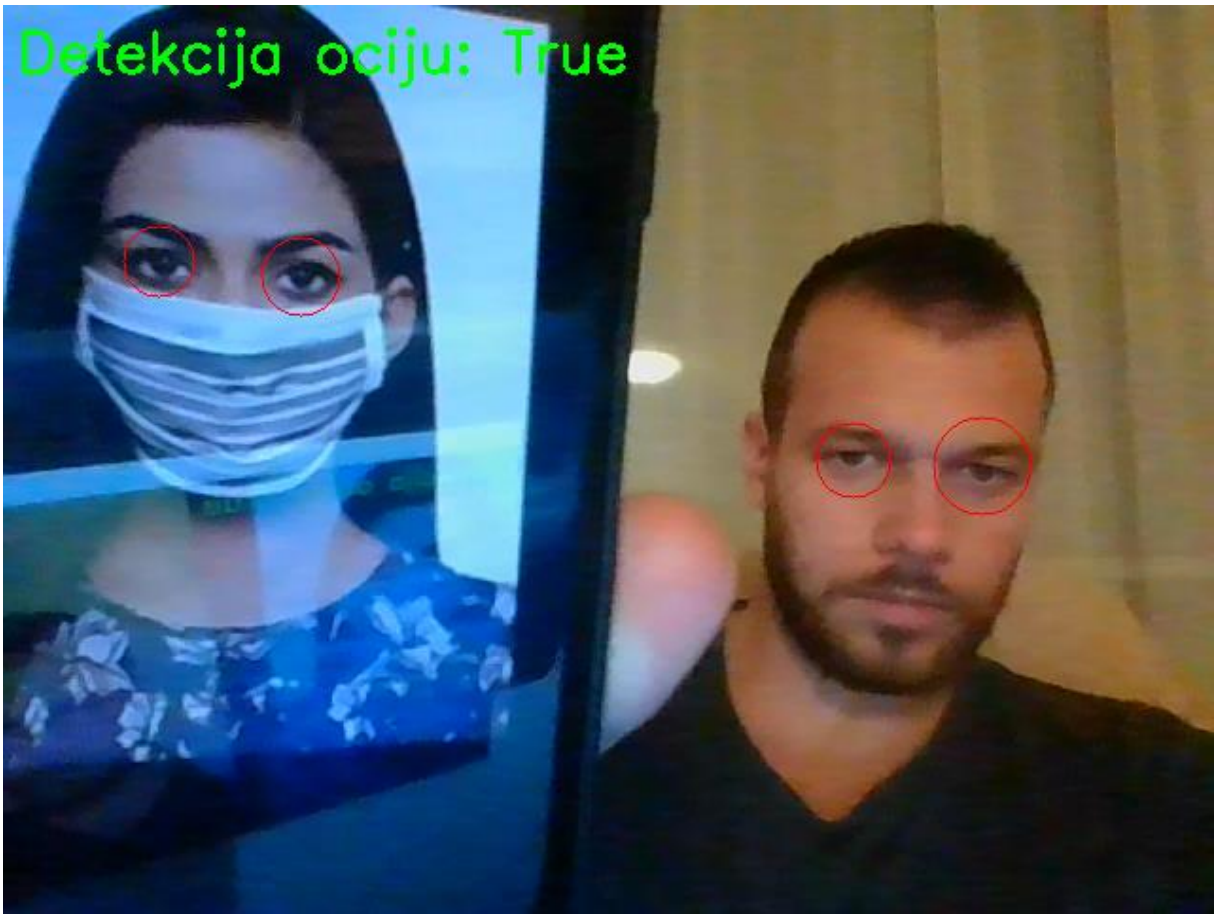
Za detekciju potencijalnih regija lica na slici koristi se metoda *detectMultiScale* koja za parametre uzima:

- sliku u sivim tonovima na kojoj se vrši detekcija
- scaleFactor – redukcija veličine slike
- minNeighbors – broj potrebnih susjeda za validaciju
- minSize – minimalna veličina prepoznate regije; odbacuju se regije manje od zadane veličine

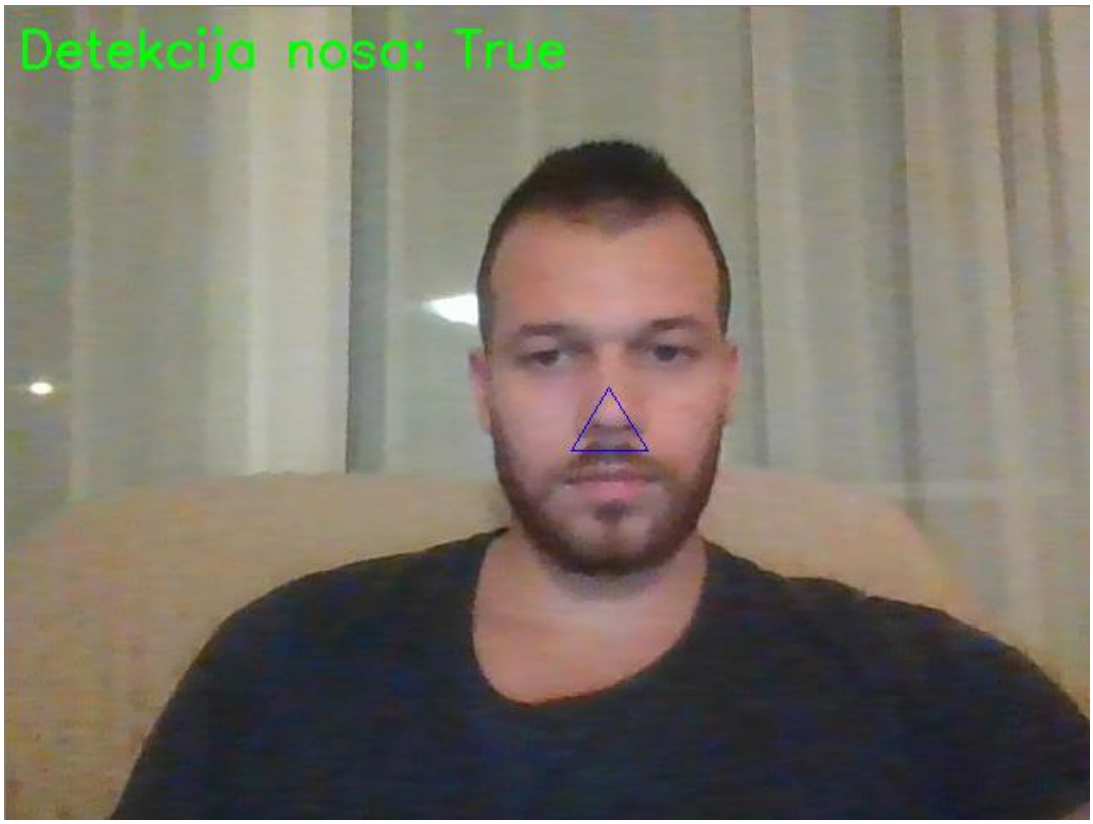
Metodi je inače dovoljan parametar slike no u radu su uključeni dodatni u svrhu postizanja što kvalitetnijeg rješenja. Prepoznata regija lica predstavljena je koordinatama ishodišne točke te visine i širine u jediničnom sustavu koji odgovara slici na kojoj je izvršena detekcija. Kôd 4.3 uz kôd 4.2 predstavlja značajku detekcije lica. Isti je potrebno uključiti unutar *while True* petlje ranije navedenog „kostura“ aplikacije kako bi značajka bila funkcionalna.

4.3 Detekcija očiju, nosa i usana

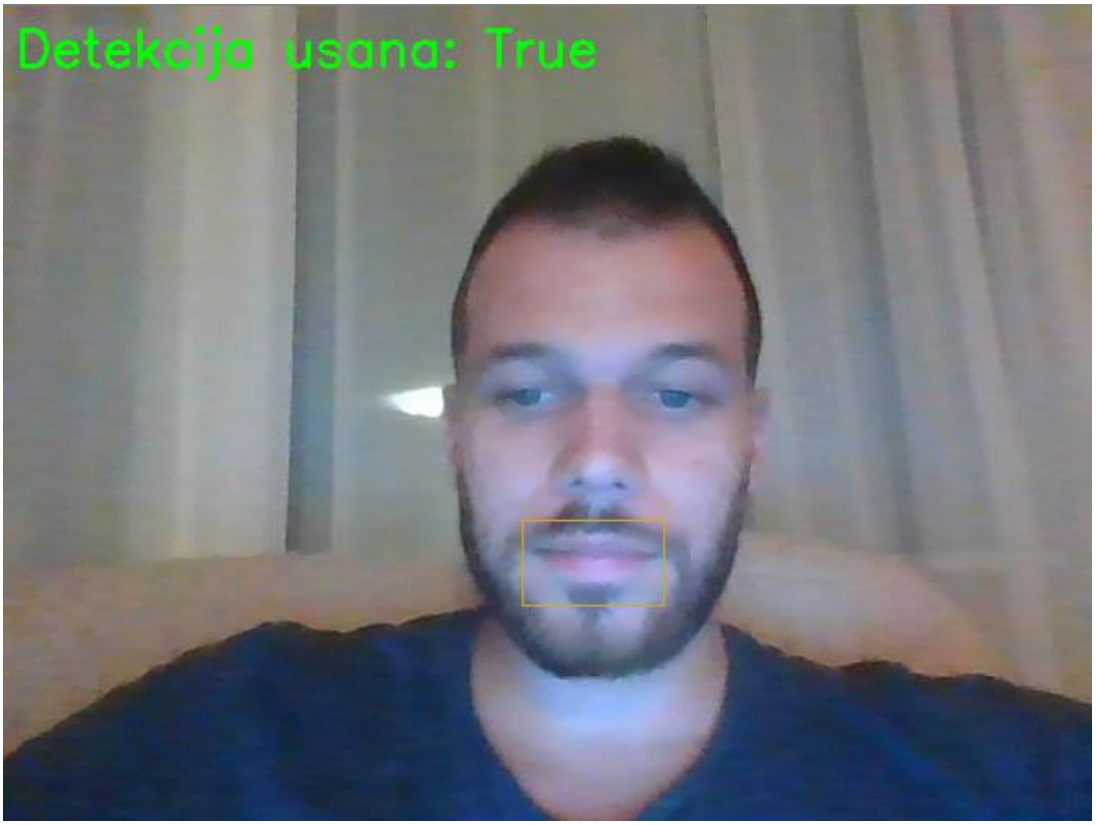
Detekcija očiju, nosa i usana nije ništa drugo nego detekcija konture odnosno crta lica. Ovi načini su također predstavnici druge faze rješavanja problema. S obzirom da su slično implementirana, ove tri značajke su predstavljene zajedno. U svrhu bolje preglednosti i lakšeg razmijevanja ove značajke su različito prikazne. Tako će prepoznate regije očiju biti naznačene crvenim kružnicama, regije nosa plavim trokutom a regije usana zlatnim pravokutnikom. U svrhu estetike odabrana je baš ovakva kombinacija kako bi sami geometrijski oblici pripadajućih regija nalikovale na lice.



Slika 4.4 Detekcija očiju



Slika 4.5 Detekcija nosa



Slika 4.6 Detekcija usana

Za ova tri načina rada kamere postupak dizajniranja te implementacije gotovo je identičan prethodno objašnjenom načinu detekcije lica. Isti su obuhvaćeni ovim radom za potrebe usporedbe i demostracije drugih značajki lica. Slično kao kod načina detekcije lica, i ova tri načina rada zahtijevaju prethodno uvježban model (kôd 4.4). Takvi modeli trebaju biti zasnovani na očima, odnosno nosu ili usnama.

```
1.  eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
2.  nose_cascade = cv2.CascadeClassifier('haarcascade_mcs_nose.xml')
3.  smile_cascade = cv2.CascadeClassifier('haarcascade_mcs_mouth.xml')
```

Kôd 4.4 Učitavanje modela za detekciju očiju, nosa i usana

Ono što je bitno kod detekcije očiju, nosa i usana jest da se detekcija vrši unutar prepoznate regije lica. Na ovaj način postignut je višestruk benefit. Mogućnost pogrešne detekcije očiju, nosa i usana svedena je na minimum jer se ne pretražuje čitava slika. Uz to, ubrzan je i proces detekcije. Kako bi ovakav dizajn bio efikasan, detekcija lica mora biti što je moguće bolja.

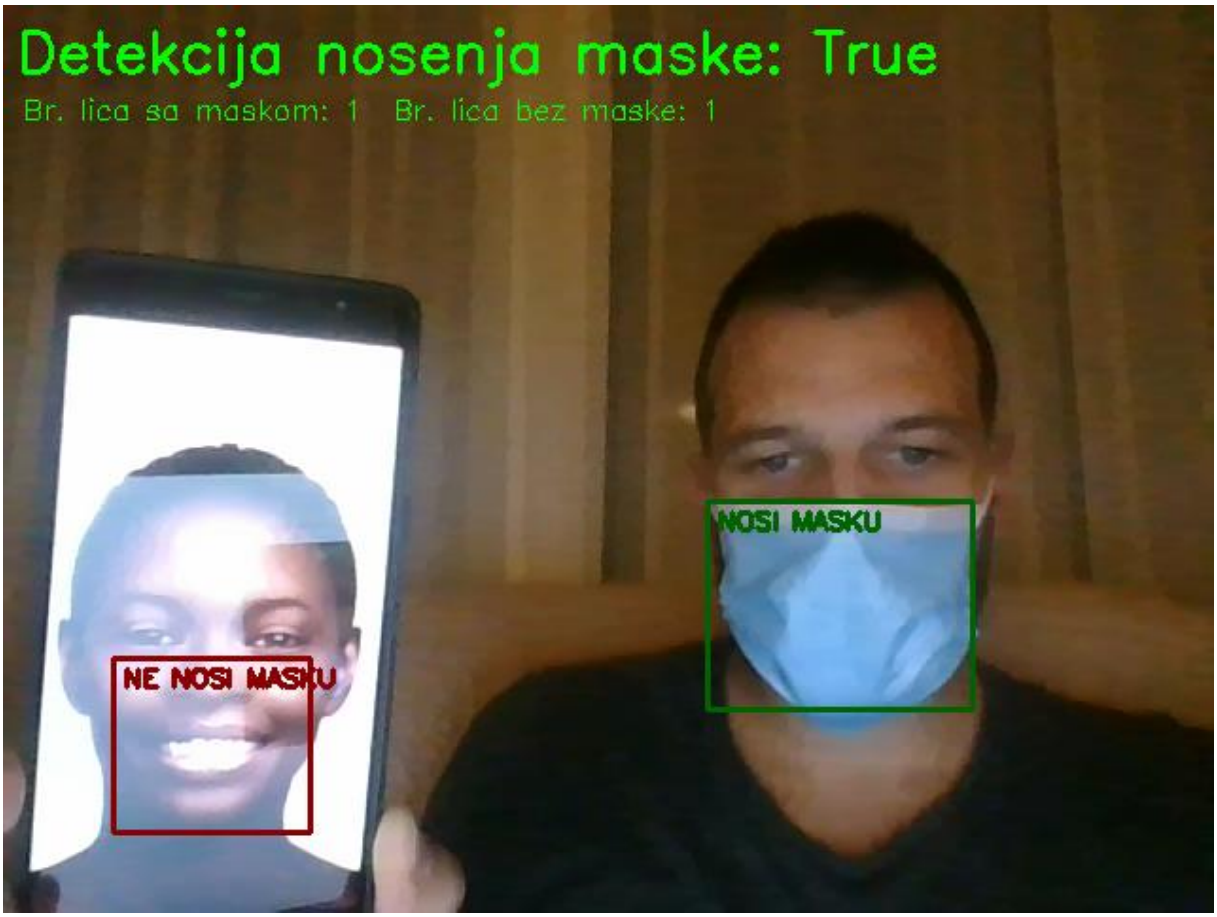
```
1.  if oci:
2.      eye = eye_cascade.detectMultiScale(gray[y:y+h,x:x+w], 1.3, 5)
3.      for (ax, ay, aw, ah) in eye:
4.          if(y+ay < y+(h/2)):
5.              cv2.circle(frame, (x+ax+aw//2, y+ay+ah//2), aw//2, (0, 0, 255), 1)
6.  if nos:
7.      nose = nose_cascade.detectMultiScale(gray[y:y+h, x:x+w], 1.3, 5)
8.      for (ax, ay, aw, ah) in nose:
9.          t1, t2, t3 = (x+ax+aw//2, y+ay), (x+ax, y+ay+ah), (x+ax+aw, y+ay+ah)
10.         cv2.drawContours(frame, [np.array( [t1, t2, t3] )], 0, (255, 0, 0), 1)
11. if usne:
12.     smile = smile_cascade.detectMultiScale(gray[y:y+h, x:x+w], 1.3, 5)
13.     for (ax, ay, aw, ah) in smile:
14.         if(y+ay > y+(h/2)):
15.             cv2.rectangle(frame, (x+ax, y+ay),(x+ax+aw, y+ay+ah), (32, 165, 218), 1)
```

Kôd 4.5 Detekcija očiju, nosa i usana

Kôd 4.5 uz kôd 4.3 predstavlja značajke detekcije očiju nosa i usana. Isti je potrebno uključiti unutar *for* petlje ranije navedenog koda 4.3 detekcije lica kako bi značajke bile funkcionalne. Posebna pozornost usmjerena je ka koordinatama detektiranih područja. Podsjetimo, kod ovih triju značajki ishodišne točke te visine i širine predstavljene su jediničnim sustavom koji odgovara prepoznatoj regiji lica. Stoga je od iznimne važnosti uskladiti jedinične sustave kako bi se osigurao ispravan prikaz na slici.

4.4 Detekcija nošenja maske

Detekcija nošenja maske odnosi se na prikaz kamere sa značajkom prepoznavanja ljudskog lica koje nosi odnosno ne nosi zaštitnu masku. Ovaj način primjer je treće faze rješavanja problema. Metoda i tip modela ovog načina mogu se koristiti i za prepoznavanje lica iako se konkretno kod ovog načina ne prepoznaje lice. Upotreba metode i tipa modela za prepoznavanje u kontekstu detekcije može djelovati konfuzno. No detekcija nošenja maske nije ništa drugo nego prepoznavanje lica sa maskom odnosno lica bez maske. U kontekstu načina prepoznavanja lica predviđa se čijem točno licu pripadaju karakteristike uzorka. Ipak, kod ovog se načina predviđa kojoj kategoriji lica pripadaju karakteristike uzorka. Predviđa se je li uzorak sličniji onim uzorcima u kategoriji sa zaštitnom maskom ili pak onima unutar kategorije bez zaštitne maske. Primjer prepoznavanja lica slijedi u nastavku ovog rada.



Slika 4.7 Detekcija nošenja maske

Ukoliko je detektirana upotreba zaštitne maske na licu, maska će biti istaknuta zelenim kvadratom popraćena tekstualnom oznakom o statusu. Ukoliko upotreba nije detektirana, predviđeno područje upotrebe također će biti istaknuto kvadratom i oznakom ali u crvenoj boji. Doduše, prepoznata regija originalno uključuje čitavo lice. U svrhu boljeg pregleda rezultata

obrubi regije i boja su prilagođeni. Ovaj način sadrži i oznaku brojača koji prebrojava primjere detekcije na slici. Slika 4.7 prikazuje izgled načina detekcije nošenja maske na nekom primjeru.

U svrhu ostvarivanja ovog načina korištene su značajke biblioteke TensorFlow. Iako je ovaj način mogao biti ostvaren i putem biblioteke OpenCV, korištena je druga biblioteka kako bi se pokazalo postojanje i drugih alata pri rješavanju istog problema. Kao što je ranije navedeno, konkretno prepoznavanje lica korištenjem OpenCV biblioteke slijedi u nastavku.

```
1.  mask_model = load_model("mask_model.h5")
```

Kôd 4.6 Učitavanje modela detekcije nošenja maske

Kako bi ovaj način rada mogao uspješno detektirati nošenje maske potrebno je koristiti uvježbani model. Za potrebe rada, kao model detekcije nošenja maske korišten je model zasnovan na Kerasovom unaprijed uvježbanom modelu MobileNet u sklopu biblioteke TensorFlow. Odabrani model stvoren je uvježbavanjem unaprijed uvježbanog modela MobileNet. I kod ovog načina korišten je gotovi model. Za učitavanje modela koristi se metoda *load_model* na način prikazan u kôdu 4.6.

```
1.      if maska:
2.          uzorak = frame[y:y+h, x:x+w]
3.          uzorak = cv2.resize(uzorak,(224, 224))
4.          uzorak = np.array(uzorak)
5.          uzorak = np.expand_dims(uzorak, axis=0)
6.          uzorak = preprocess_input(uzorak)
7.          pred = mask_model.predict(uzorak)
8.          (sa, bez) = pred[0]
9.          br_maski = br_maski + 1 if sa>bez else br_maski
10.         labele[3] = "NOSI MASKU" if sa>bez else "NE NOSI MASKU"
11.         boja = (0, 100, 0) if sa>bez else (0, 0, 139)
12.         cv2.rectangle(frame, (x+20, y+h//2), (x+w-20, y+h+20), boja, 2)
13.         cv2.putText(frame, labele[3], (x+25, y+h//2+15), cv2.FONT_HERSHEY_SIMPLEX, 0
    .45, boja, 2)
14. if maska:
15.     labele[1] = "Br. lica sa maskom: " + str(br_maski) + "  Br. lica bez maskom: " +
     str(labele[2]-br_maski)
```

Kôd 4.7 Detekcija nošenja maske

Za uzorak se uzima regija lica. Kod ovog načina gdje se prepoznavanje vrši preko TensorFlow biblioteke, uzorak ne mora biti u sivim tonovima. Uzorak je zatim potrebno svesti u odgovarajući oblik za operaciju predviđanja. Redom se izvodi dovođenje u odgovarajuću veličinu, pretvaranje uzorka u niz i proširenje te predobrada. Predviđanje se izvodi metodom *predict*. Metoda kao argument prima uzorak nad kojim je potrebno izvršiti predviđanje. Rezultat predviđanja iskazan je na ljestvici od 0 do 1 ovisno o tome koliko je predviđanje pouzdano. S obzirom da ovaj način treba detektirati i prikazati primjere korištenja i primjere ne korištenja

maske, rezultat predviđanja treba imati dvije vrijednosti. Odabrani model uvježban je primjerima nošenja i primjerima ne nošenja maske, te je na taj način osiguran rezultat predviđanja sa dvije vrijednosti. Prva vrijednost predstavlja razinu pouzdanosti predviđanja nošenja maske, odnosno ne nošenja maske za drugu vrijednost. Na ovaj način uzorku se predviđa pouzdanost pripadanja za oba slučaja. Ovisno koje je predviđanje pouzdanije, prepoznava se pripada li uzorak slučaju sa maskom ili slučaju bez maske.

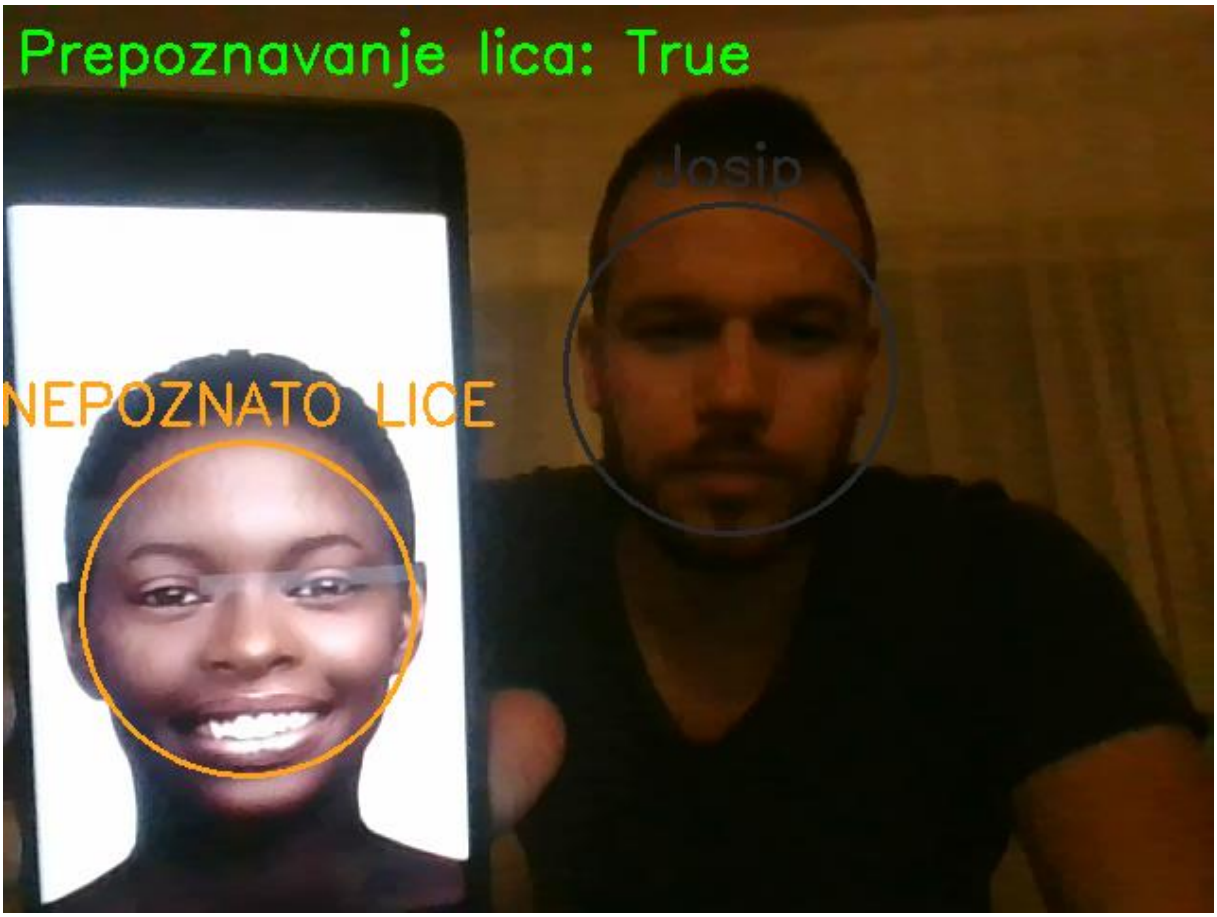
Kako bi značajka bila funkcionalna, kôd 4.7 potrebno je uključiti unutar ranije navedene *while True* petrlje, a prvi *if maska* uvjet dodatno unutar *for* petlje ranije navedenog koda 4.3 detekcije lica.

4.4.1 Primjer primjene I – Prevencija pandemije

Prvi primjer primjene inspiriran je pandemijom virusa COVID-19 koji je uzrokovao globalnu zdravstvenu krizu. Od prve pojave u prosincu 2019. godine do rujna 2020. godine evidentirano je preko 30 milijuna slučajeva zaraze. Svjetska zdravstvena organizacija kao prevenciju daljnjeg širenja virusa preporuča korištenje zaštitnih maski te držanje socijalne distance. Korištenje zaštitne maske na mnogim je mjestima postalo obavezno. Odluka o obaveznom korištenju maske zahtjeva odgovarajuće kontrolne mehanizme. Jedan primjer takvog mehanizma zasniva se na prepoznavanju lica sa maskom, odnosno detekciji nošenja maske. Ovaj primjer primjene osmišljen je kao sustav detekcije nošenja maske. Sustav podrazumijeva korištenje kamera povezanih sa računalom. Kod ovakvog sustava kamere bi trebale biti postavljene unutar mjesta u kojemu je obavezno korištenje zaštitne maske. Korištenjem računala vršila bi se automatska detekcija nošenja maske. Po detekciji nepoštivanja mjere obaveznog korištenja maske, odgovorne institucije dobile bi obavijest o istom. Takva obavijest može sadržavati sliku lica te slične informacije koje će pomoći pri donošenju odluke o sankciji prijavljene osobe. Ovakav sustav uvelike pomaže odgovornim institucijama pri kontroli a ima i potencijala pri reduciranju broja nepoštivanja mjera. Kao takav, može biti od iznimne koristi kod prevencije pandemije.

4.5 Prepoznavanje lica

Prepoznavanje lica odnosi se na prikaz kamere sa značajkom prepoznavanja ljudskog lica na slici. U ovom radu prepoznavanje lica zasnovano je na OpenCV biblioteci. Ovim se načinom željela pokazati treća, ujedno i posljednja faza rješavanja problema prepoznavanja lica na slici. U ovoj fazi uspoređuju se trenutno detektirane značajke lica sa značajkama svih lica unutar uvježbanog modela lica.



Slika 4.8 Prepoznavanje lica

Ukoliko se trenute značajke podudaraju sa značajkama nekog lica unutar modela, tada je došlo do prepoznavanja lica. Slika 4.8 prikazuje primjer jednog prepoznatog i jednog nepoznatog lica na slici. Ovisno o uspješnosti prepoznavanja, regija prepoznatog lica biti će istaknuta sivom kružnicom, odnosno narančastom kružnicom ukoliko detektirano lice nije prepoznato. Također, ukoliko je lice prepoznato, kružnica će sadržavati i oznaku sa imenom lica. U protivnom će oznaka sadržavati poruku o nepoznatom licu. I kod ovog načina, a u svrhu boljeg razumijevanja i pregleda rezultata, boje kružnice i oznake ovise o ishodu prepoznavanja.

Ovaj način mora sadržavati uvježban model kako bi bio sposoban prepoznavati lica. Uvježban model sadrži značajke svih prethodno naučenih lica i njihove ID-jeve. Učenje novih lica slijedi

u nastavku rada. S obzirom da ID unutar modela lica može biti isključivo broj, popis imena naučenih lica sprema se u tekstualnu datoteku. Datoteka sa imenima i sve popratne datoteke ovog rada moraju se nalaziti na istoj lokaciji.

```
1.  if(not path.exists(url) or not path.exists(url2)):
2.      print("Ne postoji kreiran model. Najprije naučite lica.")
3.  else:
4.      if prepoznavanje:
5.          dic = {}
6.          recognizer = cv2.face.LBPHFaceRecognizer_create()
7.          recognizer.read(url)
8.          with open(url2) as i:
9.              imenik = (i.read()).split(" ")
10.             imenik.pop()
11.         for i in imenik:
12.             dic[i] = 0
13.         print(imenik)
```

Kôd 4.8 Učitavanje modela i imenika za prepoznavanje lica

Pri realizaciji načina prepoznavanja lica prvenstveno je potrebno uvjeriti se u postojanje imenika te uvježbanog modela. Ukoliko isti nisu dostupni, najprije je potrebno naučiti neko lice kroz način učenja lica. U protivnom nema potrebe vršiti prepoznavanje lica ako nije poznato niti jedno lice. Ukoliko su isti dostupni, potrebno je inicijalizirati LBPH model te učitati uvježbani model. Također je potrebno učitati i imena iz imenika kao u kôdu 4.8.

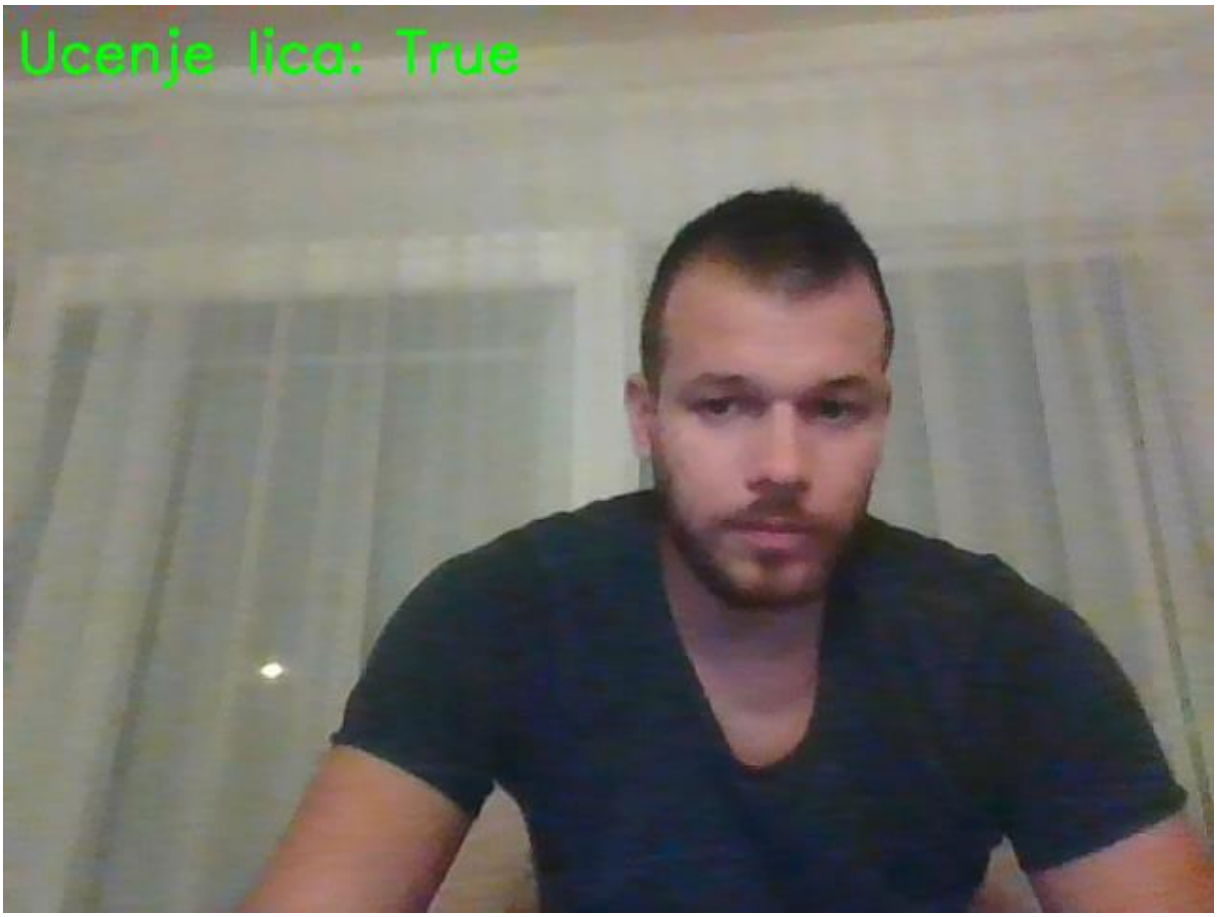
```
1.  if prepoznavanje:
2.      Id, confidence = recognizer.predict(gray[y:y+h,x:x+w])
3.      labele[4] = imenik[Id] if confidence<70 else "NEPOZNATO LICE"
4.      vel = cv2.getTextSize(labele[4],cv2.FONT_HERSHEY_SIMPLEX, 1, 2)
5.      boja2 = (41, 41, 41) if confidence<70 else (26, 163, 255)
6.      cv2.circle(frame, (x+w//2, y+h//2), w//2, boja2, 2)
7.      cv2.putText(frame, labele[4], (x+w//2-vel[0][0]//2,
           y-10), cv2.FONT_HERSHEY_SIMPLEX, 1, boja2, 2)
```

Kôd 4.9 Prepoznavanje lica

Prepoznavnje lica vrši se na način da se LBPH modelu putem metode *predict* kao parametar pošalje lice u sivom tonu. Metoda potom vrši provjeru te kao rezultat vraća ID prepoznatog lica i razinu pouzdanosti. Razina pouzdanosti je vrijednost od 0 do 100 gdje se 0 smatra savršenom pouzdanošću. Metoda će uvijek vratiti ID sa najmanjom vrijednosti pouzdanosti. Iz tog razloga potrebno je definirati granicu pouzdanosti. Za granicu pouzdanosti odabrana je vrijednost 70 iz razloga što većina naučenih lica daje nižu vrijednost pouzdanosti. Ukoliko je pouzdanost veća ili jednaka 70, uglavnom je riječ o nepoznatom licu. Ukoliko nije riječ o nepoznatom licu, oznaka poprima odgovarajuće ime iz imenika. Kako bi značajka prepoznavanja lica bila funkcionalna, kôd 4.8 potrebno je uključiti unutar *while True* petrlje, a kôd 4.9 dodatno unutar *for* petlje ranije navedenih kodova.

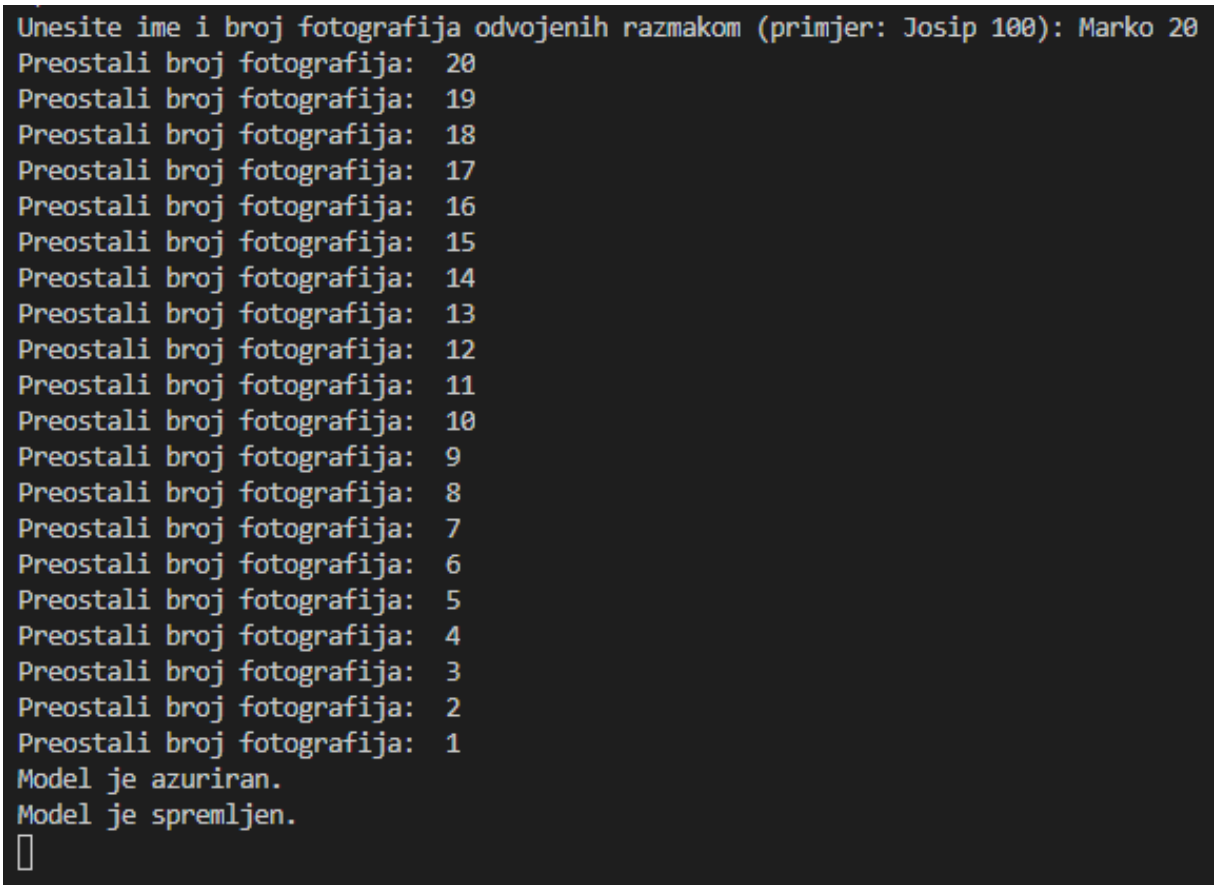
4.6 Učenje lica

Učenje lica odnosi se na na postupak prikupljanja slika lica putem kamere i uvježbavanja modela koji je neophodan pri prepoznavanju lica. Ovaj način rada je specifičan i privremen. Najprije zahtijeva korištenje konzole, a potom skeniranje lica te na samom kraju uvježbavanje modela. Po završetku ovog procesa, učenje lica je završeno te se automatski aktivira način običnog prikaza.



Slika 4.9 Učenje lica

Po aktivaciji ovog načina potrebno je unijeti ime i broj slika odvojeni razmakom te po završetku potvrditi svoj unos. Ime se odnosi na ime osobe nad kojom se planira vršiti proces učenja lica. Broj slika je prirodan broj i odnosni se na proizvoljan broj slika koje će biti prikupljene putem kamere. Ukoliko se u trenutku prikupljanja slika putem kamere ne detektira lice na slici, iste će biti odbačene. Odnosno, prikupljene će biti samo one slike na kojima je detektirano lice. Prikupljanje slika će se zaustaviti svaki put kada lice ne bude detektirano. Proces prikupljanja se odvija sve dok nije ostvaren prethodno zadan broj slika. Po završetku slijedi uvježbavanje modela na temelju ranije prikupljenih slika. Slika 4.10 prikazuje primjer izgleda konzole od trenutka aktivacije načina učenja lica do završetka.



Slika 4.10 Izgled konzole tijekom učenja lica

```
1.  if ucenje:
2.      recognizer = cv2.face.LBPHFaceRecognizer_create()
3.      x = input("Unesite ime i broj fotografija odvojenih razmakom (primjer: Josip 100
    ): ")
4.      ime, br = x.split(" ")[0], int(x.split(" ")[1])
5.      imenik = []
6.      if(path.exists(url2)):
7.          with open(url2) as i:
8.              imenik = (i.read()).split(" ")
9.              imenik.pop()
10.     if(ime in imenik):
11.         ID = imenik.index(ime)
12.     else:
13.         ID = len(imenik)
14.         imenik.append(ime)
15.     with open(url2, 'w') as i:
16.         for el in imenik:
17.             i.writelines(el+" ")
```

Kôd 4.10 Unos podataka i osvježavanje imenika pri učenju lica

Način učenja lica zahtijeva prethodnu inicijalizaciju LBPH modela. Metoda ovog modela u nastavku će biti korištena pri procesu učenja lica. Slijedi unos podataka te učitavanje imena iz imenika. Kako bi se izbjeglo višestruko definiranje istog lica, potrebno je provjeriti nalazi li se ime u imeniku naučenih lica. Licu koje ulazi u proces učenja biti će dodijeljen odgovarajući ID iz imenika ukoliko se ime već nalazi u imeniku. U ovom slučaju, riječ je o licu koje je već izvršilo proces učenja. Ukoliko se ime ne nalazi u imeniku, licu će biti dodjeljen prvi slobodni ID. Nakon što je imenik osvježen, pristupa se spremanju istog. Naknadna učenja istog lica su dozvoljena sve dok koriste isti ID. U svrhu unapređenja prepoznavanja moguće je proizvoljan broj puta učiti isto lice. Korištenje istog ID-ja pri učenju različitih lica rezultirat će kreiranjem neispravnog modela. Ne preporuča se kreiranje novog ID-ja pri učenju ranije naučenog lica jer rezultati mogu djelovati zbunjujuće.

```
1.  if ucenje:
2.      if br>0:
3.          kontura = np.array(gray[y:y+h,x:x+w],'uint8')
4.          galerija.append(kontura)
5.          IDs.append(ID)
6.          print("Preostali broj fotografija: ", br)
7.          br-=1
8.          if(br==0):
9.              if(path.exists(url)):
10.                 recognizer.update(galerija, np.array(IDs))
11.                 print("Model je azuriran.")
12.             else:
13.                 recognizer.train(galerija, np.array(IDs))
14.                 print("Model je kreiran.")
15.             recognizer.save(url)
16.             print("Model je spremljen.")
```

Kôd 4.11 Učenje lica

Prikupljanje slika lica podrazumijeva dodavanje slika lica u sivim tonovima unutar liste, odnosno galerije slika. ID lica dodaje se u vlastitu listu po dodavanju slike u galeriju. Ove dvije liste neophodne su pri procesu učenja lica. Nakon kompletiranja navedenih listi, pristupa se učenju lica. Učenje lica izvršava se putem metode *train* unutar modela LBPH ukoliko nije dostupan uvježbani model. Ako pak postoji postoji uvježban model, učenje se izvršava putem metode *update*. Obje metode kao parametar primaju listu slika i listu odgovarajućih ID-jeva. Po završetku operacija učenja potrebno je sačuvati uvježbani model. Kako bi značajka učenja lica bila funkcionalna, kôd 4.10 potrebno je uključiti unutar *while True* petrlje, a kôd 4.11 dodatno unutar *for* petlje ranije navedenih kodova.

4.7 Test video

Posljednji u nizu načina rada kamere je test video. Ovaj način služi u svrhe testiranja značajki i demonstracije primjera primjene rješenja ovog rada. Po aktivaciji ovog načina, prikaz slike kamere u stvarnom vremenu zamijenjen je primjerom videozapisa. Videozapis se prikazuje beskonačno do deaktivacije ovog načina kada ponovno započinje prikaz slike kamere. Primjer videozapisa popraćen je odgovarajućim testnim uvježbanim modelom i testnim imenikom. Ovisno o aktivaciji ovog načina, mijenja se i lokacija sa koje se učitava uvježbani model te imenik.

4.7.1 Primjer primjene II – Evidencija radnog vremena

Drugi primjer primjene realiziran je ovim načinom rada kamere. Iako još uvijek u nekim situacijama radnici samostalno vode vlastitu evidenciju radnog vremena, u suvremenom je svijetu evidencija je digitalizirana. Jedan od danas najpopularnijih sustava evidencije baziran je na beskontaktnim karticama. Kod ovakvog sustava, svaki se zaposlenik evidentira prislanjanjem vlastite beskontaktne kartice na čitač prilikom svakog prolaska pored čitača. Čitač je uglavnom smješten neposredno uz sam ulaz u mjesto obavljanja posla i povezan je sa računalom. Računalo automatski kreira evidenciju za svakog zaposlenika.

Ovaj primjer primjene ide korak dalje i predlaže uvođenje sustava evidencije baziranog na kamerama. Kod ovakvog sustava kamere bi trebale biti postavljene neposredno unutar ulaza u mjesto obavljanja posla i povezane sa računalom. Ovisno o realizaciji sustav može uključivati više kamera. Primjerice, kamere na različitim visinama i sa različitim usmjerenjima, ili kamere na različitim pozicijama unutar radnog mjesta ako se želi precizirati kretanje zaposlenika. Računalo bi vršilo detekcije i prepoznavanja lica na slikama kamere u stvarnom vremenu. Također, putem računala bi se obavljao popis zaposlenika sa njihovim osobnim detaljima, uključujući i sliku lica. Prepoznavanjem lica, računalo bi evidentiralo vrijeme dolaska i odlaska zaposlenika. Na taj način ukinula bi se potreba za korištenjem beskontaktnih kartica a čitav sustav dodatno bi se automatizirao. Primjena ovakvog sustava bila bi od višestruke koristi. U smislu sigurnosti, ovaj sustav može se koristiti kao video nadzor. Po potrebi, ovakav sustav može vršiti evidenciju svih posjetitelja pomoću detekcije lica.

Kroz test video način demonstriran je primjer ovakve primjene. Omogućavanjem načina prepoznavanja lica, računalo uspješno prepoznava lica iz imenika (Slika 4.11).

Slika 4.11 Prepoznavanje lica

Ovaj primjer se može poistovjetiti sa sustavom evidencije baziranog na kamerama. Videozapis je u ulozi slike kamere u realnom vremenu. Imenik iz primjera je ustvari popis zaposlenika, a lice iz videozapisa je lice zaposlenika.

```
Perisic se prvi put pojavljuje u 16:12:19
Vida se prvi put pojavljuje u 16:12:29
Brozovic se prvi put pojavljuje u 16:12:36
Strinic se prvi put pojavljuje u 16:12:44
Rebic se prvi put pojavljuje u 16:12:51
```

Slika 4.12 Konzola u ulozi evidencije

Nakon što je lice prepoznato, u konzoli se ispisuje vrijeme prve pojave prepoznatog lica. Konzola se ovdje može poistovjetiti sa evidencijom radnog vremena u koju se upisuje vrijeme dolaska, odnosno odlaska zaposlenika.

5. ZAKLJUČAK

Naučiti računalo da detektira objekte jedno je od prvih razloga nastanka čitave današnje znanosti koja se naziva računalna vidi. Do sada su uloženi veliki napori te su provedena brojna istraživanja u svrhu razvijanja sustava prepoznavanja lica. Kao i u mnogim drugim nastojanjima da se računalo nauči obavljati čovjeku već poznatu vještinu, tako se i kod prepoznavanja lica koristio nativan pristup. Računalo se pokušalo naučiti prepoznavanju lica imitiranjem ljudskih kognitivnih procesa prepoznavanja. Klasični pristupi temeljili su se uglavnom na osvjetljenju i poravnanjima. Razvojem umjetne inteligencije i računalne moći, posebice u zadnjem desetljeću, računalni vid i strojno učenje također su unaprijeđeni. Moderni pristupi temelje se na velikom broju značajki te u kombinaciji sa neuronskom mrežom postižu vrlo zadovoljavajuće rezultate.

Ovim radom pokazalo se kako problem prepoznavanja lica nije jednostavan, već da je to složen proces koji uključuje detekciju i učenje lica. Klasične metode, iako već dovoljno stare i nedovoljno precizne, i dalje su vrlo popularne i zastupljene. Prvenstveno iz razloga jer se temelje na jednostavnoj analogiji koja ne zahtjeva veliku računalnu moć te se kao takve vrlo brzo izvode. Učenje pa i prepoznavanje lica ovisi o osvjetljenju, broju različitih slika, kutu lica i slično. Sa druge strane, moderne metode su dosta preciznije i ne zahtjevaju stroge uvjete pri prepoznavanju lica. Međutim, upravo zbog svoje složenosti, često zahtjevaju veliku računalnu moć, pogotovo za uvježbavanje modela.

Umjetna inteligencija doživila je rapidan rast u posljednjih nekoliko godina te se i dalje ubrzano razvija. Jedna je od vodećih znanstvenih disciplina današnjice i svakim je danom sve više zastupljena u svim aspektima života. Neosporivo je kako će upravo umjetna inteligencija biti glavni faktor razvoja prepoznavanja lica u budućnosti. Ipak, razvoj prepoznavanja lica biti će uvjetovan razvoju računalnog sklopovlja. Bolje rečeno, razvoj prepoznavanja lica u budućnosti slijedit će razvoju računalne moći te razvoju digitalne kamere.

Sa druge strane, postavlja se pitanje primjene ovakve tehnologije. Bilo to u automobilima pri testiranju umora vozača ili pri otključavanju tipkovnice pametnog telefona, očito je kako prepoznavanje lica teži upotrebi u sigurnosne svrhe. Upravo se ta jedinstvenost značajki lica i biometrija pokazuje kao pravi potencijal prepoznavanja lica. Iako prepoznavanje lica pomaže pri provjeri osobnog identiteta, ovakva upotreba dovodi u pitanje privatnost pojedinca. Ukoliko je pojedincu važna privatnost, postavlja se pitanje kontrole vlastitih podataka s obzirom da je biometrija lica, kao i svaki drugi oblik biometrije, podatak.

Literatura

1. Anggo, Mustamin, and La Arapu. "Face recognition using fisherface method." *Journal of Physics* 1028 (2018).
2. Dr. G.S.Anandha Mala , A. Vijay Akash , S. Shobika "Tracking of Criminal Through CCTV Using Face Detection" *International Journal of Advanced Science and Technology*, v. 29, n. 06, p. 6542 - 6551, 2020.
3. EFF https://www.eff.org/pages/face-recognition
4. Irgens, Peter, et al. "An efficient and cost effective FPGA based implementation of the Viola-Jones face detection algorithm." *HardwareX* 1 (2017): 68-75.
5. Keras https://keras.io/
6. Khusbu Rani, Sukhbir "Face recognition technique using ICA and LBPH", IRJET 2017.
7. Lawrence, Steve, et al. "Face recognition: A convolutional neural-network approach." *IEEE transactions on neural networks* 8.1 (1997): 98-113.
8. Lenc, Ladislav, and Pavel Král "Automatic face recognition system based on the SIFT features." *Computers & Electrical* Engineering 46 (2015): 256-272.
9. Majumdar, A. and R. Ward. “Pseudo-Fisherface method for single image per person face recognition.” *2008 IEEE International Conference on Acoustics, Speech and Signal Processing* (2008): 989-992.
10. Norton https://us.norton.com/internetsecurity-iot-how-facial-recognition-software-works.html
11. OpenCV https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html
12. OpenCV https://opencv.org/about/
13. Padilla, R., C. F. F. Costa Filho, and M. G. F. Costa. "Evaluation of haar cascade classifiers designed for face detection." *World Academy of Science, Engineering and Technology* 64 (2012): 362-365.
14. Scientific & Academic Publishing http://article.sapub.org/10.5923.j.ajsp.20150502.02.html#Sec1
15. Sightcorp https://sightcorp.com/knowledge-base/face-detection/
16. Tensorflow https://www.tensorflow.org/about
17. Towards data science https://towardsdatascience.com/face-recognition-for-beginners-a7a9bd5eb5c2
18. Towards data science https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b
19. Turk, Matthew, and Alex Pentland. "Eigenfaces for recognition." *Journal of cognitive neuroscience* 3.1 (1991): 71-86.
20. Viola, Paul, and Michael Jones. "Rapid object detection using a boosted cascade of simple features." *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition.* CVPR 2001. Vol. 1. IEEE, 2001.
21. Visual Studio Code https://code.visualstudio.com/
22. Wikipedia https://en.wikipedia.org/wiki/Keras
23. Wikipedia https://en.wikipedia.org/wiki/Object_detection
24. Wikipedia https://en.wikipedia.org/wiki/Scale-invariant_feature_transform
25. Wikipedia https://en.wikipedia.org/wiki/TensorFlow

Sažetak

Problem prepoznavanja lica na slici podrazumijeva usporedbu jedinstvenih značajki lica na slici sa značajkama ranije poznatih lica. Lice je poznato nakon procesa učenja lica, odnosno po pronalaženju jedinstvenih značajki. Kako bi značajke bile pronađene, potrebno je najprije pronaći lice. Detekcija objekata kojom se traže crte lica naziva se detekcija lica. Pronalazak crta lica ne zahtjeva veliku preciznost. Stoga se za detekciju lica koriste se uglavnom brži klasični pristupi poput Viola-Jones algoritma sa Haarovim značajkama. Sa druge strane, pronalaženje i prepoznavanje jedinstvenih značajki zahtijeva veliku preciznost. Zato se za učenje i prepoznavanje lica pretežito koriste moderni pristupi kao što su konvolucijske neuronske mreže. Rješenje je realizirano kroz Visual Studio Code IDE. Aplikacija se sastoji od više načina rada kamere zasnovanih na OpenCV biblioteci. Predstavljena su rješenja detekcije (oči, nos, usne, lice), učenja i prepoznavanja (lice, lice sa maskom). Opisana su dva primjera primjene. Evidencija radnog vremena temelji se na klasičnom pristupu i realizirana je testnim videozapisom i prepoznavanjem lica. Prevencija pandemije primjer je modernog pristupa a realizirana je detekcijom nošenja maske, zasnovanoj na TensorFlow biblioteci i Keras značajkama.

Ključne riječi: detekcija, prepoznavanje, lica

Summary

Face recognition problem in picture involves comparing the unique features of the face in the picture with the features of previously known faces. The face is known after the process of learning the face, by finding unique features. For the features to be found, we need to find the face first. To find the face we use face detection. Face detection is detection of objects that looks for facial features. Finding facial features does not require much precision. Therefore, for face detection are mostly used faster classical approaches such as the Viola-Jones algorithm with Haar features. On the other hand, finding and recognizing unique features requires great precision. Therefore, modern approaches such as convolutional neural networks are mainly used for training and facial recognition. The solution was implemented through the Visual Studio Code IDE. The application consists of several camera modes based on the OpenCV library. Solutions for detection (eyes, nose, lips, face), training and recognition (face, face with a mask) are presented. Two application examples are described. The record of working hours is based on the classical approach and was realized by a test video and face recognition. Pandemic prevention is an example of a modern approach and was realized by mask wearing detection, based on the TensorFlow library and Keras features.

Key words: facial, detection, recognition

Popis slika i kodova

Skraćenice

AI	artificial intelligence	umjetna inteligencija
API	application programming interface	aplikacijsko programsko sučelje
CNN	convolutional neural network	konvolucijska neuronska mreža
COVID	coronavirus disease	koronavirus bolest
CPU	central processing unit	središnja jedinica za obradu
GPU	graphics processing unit	grafička procesorska jedinica
HOG	histogram of oriented gradients	histogram orijentiranih gradijenata
ID	identifier	identifikator
IDE	integrated development environment	integrirano razvojno okruženje
LBP	local binary pattern	lokalna binarna značajka
LBPH	local binary patterns histograms	histogram lokalnih binarnih značajki
SIFT	scale-invariant feature transform	transformacija značajki neovisne o skali
SURF	speeded up robust features	ubrzane robusne značajke
TPU	tensor processing unit	tenzorska procesorska jedinica

Privitak

Dodatne datoteke

Pri izradi programskog dijela ovog rada, korištene su različite datoteke trećih strana.

Model za detekciju nošenja maske
https://drive.google.com/file/d/16uMH4YwdkA8sdnMlJNE7nv_tBJkX5eNe/view

Model prepiznavanja lica
https://raw.githubusercontent.com/opencv/opencv/master/data/haarcascades/haarcascade_frontalface_default.xml

Model prepoznavanja očiju
https://raw.githubusercontent.com/opencv/opencv/master/data/haarcascades/haarcascade_eye.xml

Model prepoznavanja nosa
https://raw.githubusercontent.com/adobe/SimpleSensor/master/simplesensor/collection_modules/demographic_camera/classifiers/haarcascades/haarcascade_mcs_nose.xml

Model prepoznavanja usana
https://raw.githubusercontent.com/adobe/SimpleSensor/master/simplesensor/collection_modules/demographic_camera/classifiers/haarcascades/haarcascade_mcs_mouth.xml

Videozapis https://youtu.be/9ytQsz20dJM

Programsko rješenje

Cjelokupno programsko rješenje

https://drive.google.com/file/d/1Z-JMalit0u8RYImlVZO_qsNVbIoiQfKv/view

Izvorni kod

```
import cv2
import numpy as np
from os import path
from datetime import datetime
from tensorflow.keras.models import load_model
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
nose_cascade = cv2.CascadeClassifier('haarcascade_mcs_nose.xml')
smile_cascade = cv2.CascadeClassifier('haarcascade_mcs_mouth.xml')
mask_model = load_model("mask_model.h5")
labele = ["Osnovno", "", 0, "", ""]
lice, oci, nos, usne, maska, prepoznavanje, ucenje, test = False, False, False, Fals
e, False, False, False, False
imenik, galerija, IDs, dic, br, ID, url, url2 = [], [], [], {}, 0, 0, 'model.xml', '
imenik.txt'
cap = cv2.VideoCapture(0)
print("Dobrodošli!\nOdaberite željeni mod kamere:\n0)Izlaz\n1)Osnovno\n2)Detekcija l
ica\n3)Detekcija ociju\n4)Detekcija nosa\n5)Detekcija usana\n6)Detekcija nosenja mas
ke\n7)Prepoznavanje lica\n8)Ucenje lica\n9)Test video")
while(True):
    if cv2.waitKey(1) == ord('0'):
        break
    if cv2.waitKey(1) == ord('1'):
        labele[0] = "Osnovno"
        labele[1] = ""
        lice, oci, nos, usne, maska, prepoznavanje, ucenje = False, False, False, Fa
lse, False, False, False
    if cv2.waitKey(1) == ord('2'):
        lice = not lice
        labele[0] = "Detekcija lica: " + str(lice)
        labele[1] = "" if lice == False else ""
    if cv2.waitKey(1) == ord('3'):
        oci = not oci
        labele[0] = "Detekcija ociju: " + str(oci)
        labele[1] = "" if oci == False else ""
    if cv2.waitKey(1) == ord('4'):
        nos = not nos
        labele[0] = "Detekcija nosa: " + str(nos)
        labele[1] = "" if nos == False else ""
    if cv2.waitKey(1) == ord('5'):
        usne = not usne
        labele[0] = "Detekcija usana: " + str(usne)
        labele[1] = "" if usne == False else ""
    if cv2.waitKey(1) == ord('6'):
        maska = not maska
        labele[0] = "Detekcija nosenja maske: " + str(maska)
        labele[1] = "" if maska == False else ""
    if cv2.waitKey(1) == ord('7'):
        if(not path.exists(url) or not path.exists(url2)):
            print("Ne postoji kreiran model. Najprije naučite lica.")
        else:
            prepoznavanje = not prepoznavanje
            labele[0] = "Prepoznavanje lica: " + str(prepoznavanje)
            labele[1] = "" if prepoznavanje == False else ""
            if prepoznavanje:
                dic = {}
                recognizer = cv2.face.LBPHFaceRecognizer_create()
                recognizer.read(url)
                with open(url2) as i:
                    imenik = (i.read()).split(" ")
                    imenik.pop()
```

```
58.                 for i in imenik:
59.                     dic[i] = 0
60.                 print(imenik)
61.     if cv2.waitKey(1) == ord('8'):
62.         ucenje = not ucenje
63.         labele[0] = "Ucenje lica: " + str(ucenje)
64.         labele[1] = "" if ucenje == False else ""
65.         if ucenje:
66.             recognizer = cv2.face.LBPHFaceRecognizer_create()
67.             x = input("Unesite ime i broj fotografija odvojenih razmakom (primjer: J
    osip 100): ")
68.             ime, br = x.split(" ")[0], int(x.split(" ")[1])
69.             imenik = []
70.             if(path.exists(url2)):
71.                 with open(url2) as i:
72.                     imenik = (i.read()).split(" ")
73.                     imenik.pop()
74.             if(ime in imenik):
75.                 ID = imenik.index(ime)
76.             else:
77.                 ID = len(imenik)
78.                 imenik.append(ime)
79.             with open(url2, 'w') as i:
80.                 for el in imenik:
81.                     i.writelines(el+" ")
82.     if cv2.waitKey(1) == ord('9'):
83.         test = not test
84.         labele[0] = "Test video: " + str(test)
85.         labele[1] = "" if test == False else ""
86.         if test:
87.             url = 'model2.xml'
88.             url2 = 'imenik2.txt'
89.             cap = cv2.VideoCapture('anthem_of_Croatia_final_2018.mp4')
90.         else:
91.             url = 'model.xml'
92.             url2 = 'imenik.txt'
93.             cap = cv2.VideoCapture(0)
94.     ret, frame = cap.read()
95.     if not ret:
96.         cap = cv2.VideoCapture('anthem_of_Croatia_final_2018.mp4')
97.         ret, frame = cap.read()
98.     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
99.     faces = face_cascade.detectMultiScale(gray, scaleFactor = 1.3, minNeighbors = 5,
     minSize = (50, 50))
100.            cv2.putText(frame, labele[0], (7, 35), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 2
    55, 0), 2)
101.            cv2.putText(frame, labele[1], (10, 60), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0
    , 255, 0), 1)
102.            labele[2] = 0
103.            br_maski = 0
104.            for (x, y, w, h) in faces:
105.                labele[2] += 1
106.                if lice:
107.                    cv2.rectangle(frame, (x,y), (x+w, y+h), (0, 255, 0), 1)
108.                    cv2.putText(frame, str(labele[2]), (x+w-20, y+h-
    10), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 255, 0), 1)
109.                if oci:
110.                    eye = eye_cascade.detectMultiScale(gray[y:y+h,x:x+w], 1.3, 5)
111.                    for (ax, ay, aw, ah) in eye:
112.                        if(y+ay < y+(h/2)):
113.                            cv2.circle(frame, (x+ax+aw//2, y+ay+ah//2), aw//2, (0, 0,
     255), 1)
114.                if nos:
115.                    nose = nose_cascade.detectMultiScale(gray[y:y+h, x:x+w], 1.3, 5)

116.                    for (ax, ay, aw, ah) in nose:
```

```
                        t1, t2, t3 = (x+ax+aw//2, y+ay), (x+ax, y+ay+ah), (x+ax+aw, y+ay+ah)
                        cv2.drawContours(frame, [np.array( [t1, t2, t3] )], 0, (255, 0, 0), 1)
                if usne:
                    smile = smile_cascade.detectMultiScale(gray[y:y+h, x:x+w], 1.3, 5)
                    for (ax, ay, aw, ah) in smile:
                        if(y+ay > y+(h/2)):
                            cv2.rectangle(frame, (x+ax, y+ay), (x+ax+aw, y+ay+ah), (32, 165, 218), 1)
                if maska:
                    uzorak = frame[y:y+h, x:x+w]
                    uzorak = cv2.resize(uzorak,(224, 224))
                    uzorak = np.array(uzorak)
                    uzorak = np.expand_dims(uzorak, axis=0)
                    uzorak = preprocess_input(uzorak)
                    pred = mask_model.predict(uzorak)
                    (sa, bez) = pred[0]
                    br_maski = br_maski + 1 if sa>bez else br_maski
                    labele[3] = "NOSI MASKU" if sa>bez else "NE NOSI MASKU"
                    boja = (0, 100, 0) if sa>bez else (0, 0, 139)
                    cv2.rectangle(frame, (x+20, y+h//2), (x+w-20, y+h+20), boja, 2)
                    cv2.putText(frame, labele[3], (x+25, y+h//2+15), cv2.FONT_HERSHEY_SIMPLEX, 0.45, boja, 2)
                if prepoznavanje:
                    Id, confidence = recognizer.predict(gray[y:y+h,x:x+w])
                    labele[4] = imenik[Id] if confidence<70 else "NEPOZNATO LICE"
                    vel = cv2.getTextSize(labele[4],cv2.FONT_HERSHEY_SIMPLEX, 1, 2)
                    boja2 = (41, 41, 41) if confidence<70 else (26, 163, 255)
                    cv2.circle(frame, (x+w//2, y+h//2), w//2, boja2, 2)
                    cv2.putText(frame, labele[4], (x+w//2-vel[0][0]//2, y-10), cv2.FONT_HERSHEY_SIMPLEX, 1, boja2, 2)
                    if dic[imenik[Id]]<10 and confidence<70:
                        dic[imenik[Id]] += 1
                        if dic[imenik[Id]] == 10:
                            now = datetime.now()
                            print(imenik[Id] + " se prvi put pojavljuje u " + now.strftime("%H:%M:%S"))
                if ucenje:
                    if br>0:
                        kontura = np.array(gray[y:y+h,x:x+w],'uint8')
                        galerija.append(kontura)
                        IDs.append(ID)
                        print("Preostali broj fotografija: ", br)
                        br-=1
                        if(br==0):
                            if(path.exists(url)):
                                recognizer.update(galerija, np.array(IDs))
                                print("Model je azuriran.")
                            else:
                                recognizer.train(galerija, np.array(IDs))
                                print("Model je kreiran.")
                            recognizer.save(url)
                            print("Model je spremljen.")
            if lice:
                labele[1] = "Br. lica u slici: " + str(labele[2])
            if maska:
                labele[1] = "Br. lica sa maskom: " + str(br_maski) + "  Br. lica bez maske: " + str(labele[2]-br_maski)
            cv2.imshow('frame', frame)
        cap.release()
        cv2.destroyAllWindows()
```