Detekcija i prepoznavanje lica koristeći Raspberry Pi računalo

Dubovečak, Mario

Master's thesis / Diplomski rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University North / Sveučilište Sjever**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:122:145017

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-07-25**

Repository / Repozitorij:

University North Digital Repository

SVEUČILIŠTE SJEVER
SVEUČILIŠNI CENTAR VARAŽDIN

DIPLOMSKI RAD br.005/MMD/2020

DETEKCIJA I PREPOZNAVANJE LICA KORISTEĆI RASPBERRY PI RAČUNALO

Mario Dubovečak

Varaždin, srpanj 2020.

SVEUČILIŠTE SJEVER

SVEUČILIŠNI CENTAR VARAŽDIN

Studij Multimedija

DIPLOMSKI RAD br.005/MMD/2020

DETEKCIJA I PREPOZNAVANJE LICA KORISTEĆI RASPBERRY PI RAČUNALO

Student:
Mario Dubovečak, 0772/336D

Mentor:
doc. dr. sc. Emil Dumić

Varaždin, srpanj 2020.

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR-42000 Varaždin

Prijava diplomskog rada

Definiranje teme diplomskog rada i povjerenstva

ODJEL: Odjel za multimediju

STUDIJ: diplomski sveučilišni studij Multimedija

PRISTUPNIK: Mario Dubovečak | MATIČNI BROJ: 0772/336D

DATUM: 16.06.2020. | KOLEGIJ: Računalni vid

NASLOV RADA: Detekcija i prepoznavanje lica koristeći Raspberry Pi računalo

NASLOV RADA NA ENGL. JEZIKU: Face detection and recognition using Raspberry Pi

MENTOR: Emil Dumić | ZVANJE: doc.dr.sc.

ČLANOVI POVJERENSTVA:

1. doc.art.dr.sc. Mario Periša - predsjednik
2. doc.art. Robert Geček - član
3. doc.dr.sc. Emil Dumić - mentor
4. doc.dr.sc. Andrija Bernik - zamjenski član
5.

Zadatak diplomskog rada

BROJ: 005/MMD/2020

OPIS:

U ovom radu će biti napravljen sustav za detekciju lica te potom prepoznavanje lica iz unaprijed definirane baze, koristeći Raspberry Pi računalo.
Opisat će se razvojno okruženje koje će biti korišteno za sam rad: Raspberry Pi i pripadajuća oprema. Potom će biti opisani različiti algoritmi za detekciju lica, poput Haarovih kaskada (Viola-Jones), histograma orijentiranih gradijenata (HOG) i metode potpornih vektora (SVM), te metode dubokog učenja. Nakon toga, biti će dan primjer poravnavanja lica pomoću stabala odlučivanja (ERT). Zatim će biti dan prikaz nekih od postojećih metoda koje se često koriste za prepoznavanje lica: Fisherfaces (koristeći LDA), eigenfaces (koristeći PCA), histogrami lokalnih binarnih uzoraka (LBPH), metode temeljene na SIFT i SURF deskriptorima te metode dubokog učenja.
Praktični dio rada će biti ispitivanje različitih algoritama za detekciju lica te potom za prepoznavanje lica, koristeći ograničene mogućnosti Raspberry Pi računala. U sklopu toga će biti razvijena i baza sa unaprijed snimljenim licima i pripadajućim značajkama, koje će ovisiti o algoritmu koji će se koristiti.

ZADATAK URUČEN: 18.6.2020. | POTPIS MENTORA: Emil Dumić

REPUBLIKA HRVATSKA
SVEUČILIŠTE SJEVER

Zahvaljujem mentoru doc. dr.sc. Emilu Dumiću koji je pratio proces izrade diplomskog rada, savjetovao me i usmjeravao prilikom rješavanja problema, pružao potrebne izvore radi prikupljanja informacija, kao i na profesionalnom i nesebičnom provođenju vezanih kolegija.

Mario Dubovečak

Sažetak

Ovim radom će se prikazati sustav za detekciju lica, te prepoznavanje lica iz unaprijed definirane baze koristeći Raspberry Pi računalo.

Za detekciju lica mogu se koristiti različiti algoritmi poput Haarovih kaskada (Viola - Jones), histogrami orijentiranih gradijenata (HOG), metode potpornih vektora (SVM), te metode dubokog učenja koji će u teoretskom dijelu biti prikazani. Kao nastavak biti će prikazane neke postojeće metode koje se koriste za prepoznavanje lica poput Fisherfaces, Eigenfaces, histogrami lokalnih binarnih uzoraka, metode temeljene na SIFT i SURF deskriptorima i metode dubokog učenja zajedno sa primjerima koda u Python programskom jeziku.

Kao završni dio ovog rada biti će prikazan praktični dio, odnosno uporaba Raspberry Pi računala sa pripadajućom opremom i programskom podrškom u svrhu detekcije i prepoznavanja lica uz pomoć unaprijed definirane baze.

Ključne riječi: *računalni vid, Raspberry Pi, detekcija, prepoznavanje lica*.

Abstract

This paper will show a face detection system, and face recognition from a predefined database using a Raspberry Pi computer.

Different algorithms can be used for face detection, such as Haar cascades (Viola - Jones), oriented gradient histograms (HOG), support vector methods (SVM), and deep learning methods, which will be presented in the theoretical part. As a continuation, some existing methods used to recognize faces will be presented such as Fisherfaces, Eigenfaces, histograms of local binary patterns, methods based on SIFT and SURF descriptors and deep learning methods along with code examples in Python programming language.

As the final part of this paper, the practical part will be presented, ie the use of Raspberry Pi computers with associated equipment and software for the purpose of face detection and recognition with the help of a predefined database.

Keywords: *computer vision, Raspberry Pi, detection, face recognition.*

Popis korištenih kratica

CNN	Convolutional Neural Network	Konvolucijska neuronska mreža
DBM	Deep Boltzmann Machine	Duboki Boltzmannov stroj
DBN	Deep Belief Network	Mreža dubokog uvjerenja
DNN	Deep Neural Network	Duboka neuronska mreža
DOG	Difference of Gaussians	Gaussova razlika
ERT	Ensemble of Regression Trees	Stablo odlučivanja
FPPW	False Positive Per Window	Lažno pozitivno po prozoru
GUI	Graphical User Interface	Grafičko korisničko sučelje
HOG	Histogram of Oriented Gradients	Histogram orijentiranih gradijenata
IP	Internet Protocol	Internet protokol
LAB	L – Lightness from black; a - Green to red; b – Blue to yellow	Označava prostor boja – L-svjetlina, a – crveno-zeleno i b -žuto-plavo
LBP	Local binary pattern	Lokalni binarni uzorak
LDA	Linear Discriminant Analysis	Linearna diskriminatorna analiza
PCA	Principal Component Analysis	Analiza glavnih komponenti
RGB	Red Green Blue	Označava prostor boja – crvena, zelena i plava
RNN	Recurrent Neural Network	Povratna neuronska mreža
SdAs	Stacked denoising Autoencoders	Složeni autoenkoderi za uklanjanje šum
SIFT	Scale-Invariant Feature Transform	Skalarno invarijantna transformacija značajke
SSH	Secure Shell	Označava protokol za sigurno udaljeno spajanje
SURF	Speed Up Robust Features	Ubrzanje robusnih značajki
SVM	Support Vector Machine	Stroj potpornog vektora
VNC	Virtual Network Computing	Označava kontrolu udaljenog računala

Sadržaj

1. Uvod 8
2. Razvojno okruženje za rad 9
3. Algoritmi za detekciju lica 11
   - 3.1. Haarove kaskade (Viola - Jones) 11
   - 3.2. Histogram orijentiranih gradijenata (HOG) 16
     - 3.2.1. Normalizacija game i boja 17
     - 3.2.2. Računanje gradijenata 17
     - 3.2.3.Prostorno/orijentacijsko vezivanje 18
     - 3.2.4. Normalizacija i deskriptor blokova 18
     - 3.2.5. Prozor detekcije 20
   - 3.3. Metoda potpornih vektora (SVM) 20
4. Metode i razvoj dubokog učenja 24
   - 4.1. Konvolucijske neuronske mreže 24
     - 4.1.1. Konvolucijski slojevi 25
     - 4.1.2. Slojevi za udruživanje 25
     - 4.1.3. Potpuno povezani slojevi 25
   - 4.2. Duboke mreže vjerovanja i Deep Boltzmann strojevi 26
     - 4.2.1. Ograničeni Boltzmann-ov stroj 26
     - 4.2.2. Duboke mreže vjerovanja 27
   - 4.3. Složeni autoenkoderi 27
     - 4.3.1. Autoenkoderi za uklanjanje šumova 28
     - 4.3.2. Složeni autoenkoderi za uklanjanje šuma 28
5. Primjer poravnavanja lica pomoću stabla odlučivanja (ERT) 30
6. Metode za prepoznavanje lica 33
   - 6.1. Fisherfaces 33
   - 6.2. Eigenfaces 37
   - 6.3. Histogrami lokalnih binarnih uzoraka (LBPH) 39
   - 6.4. Metode temeljene na SIFT i SURF deskriptorima 42
   - 6.5. Metode dubokog učenja 49
7. Praktični dio 52
8. Zaključak 63
9. Literatura 65
10. Popis slika 66

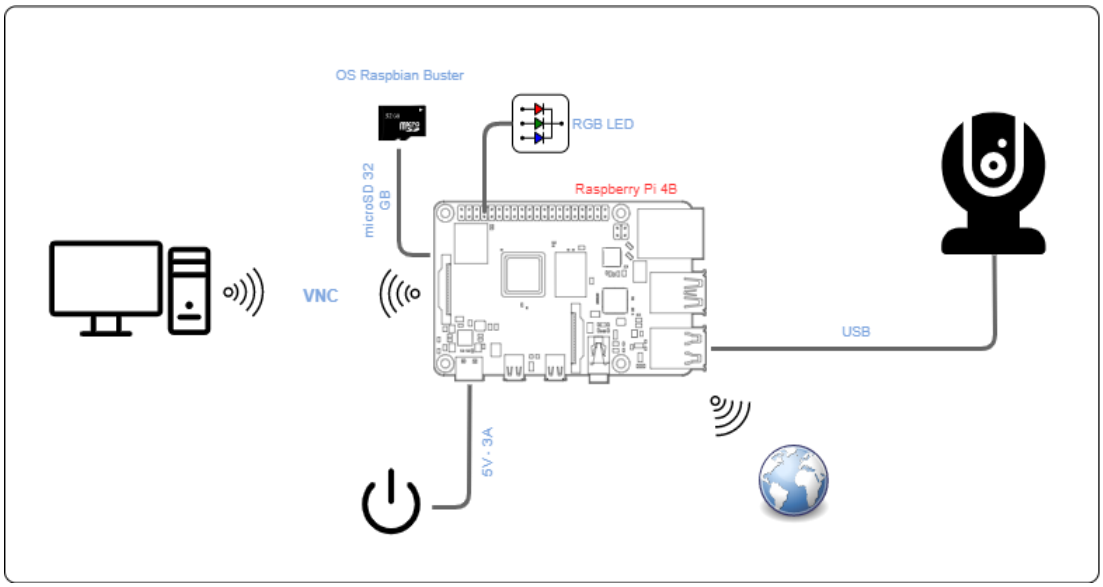
1. Uvod

Razvojem novih tehnologija razvija se želja, ali i potreba za razvojem umjetne inteligencije koja bi olakšala i unaprijedila čovjekovu svakodnevnicu. Računalne tehnologije sežu u gotovo sve grane djelatnosti kako bi se optimizirali radni procesi, ali i unaprijedili u vidu izrade novih inovativnih rješenja. Računalni vid dobrim dijelom pripada umjetnoj inteligenciji s ciljem da se određeni procesi automatiziraju kako bi bili što sličniji čovjekovoj prirodi. Prepoznavanje elemenata po kojima je svaki čovjek jedinstven, ali opet i dio jedne cjeline već se određeno vrijeme koristi bilo da je to u obliku prepoznavanja papilarnih linija, govora i ostalih fizikalnih obilježja ili da je to u obliku prikupljanja ostalih podataka radi analize podataka kao što su dob, spol, osobne navike i slično. Poznato je da se takvi podaci već uvelike koriste u virtualizaciji u svrhu analize tržišta, učenja umjetne inteligencije, izrade ciljanih proizvoda, ali i nadzora s ciljem unaprijeđenja sigurnosti i zaštiti podataka. Kao jedan od najzastupljenijih primjera može se navesti otključavanje pametnih telefona uporabom biometrijskih podataka (otključavanje otiskom prstiju ili prepoznavanjem lica) što je i u toj namjeri započelo da bi se kasnije ta mogućnost proširila na neke druge mobilne aplikacije u kategoriji mobilnog bankarstva, sigurnosti i pohrane podataka, prijave na e-servise i slično.

Računalni vid nudi niz mogućnosti sa stalnim težištem prema novim programskim inovacijama uporabom dostupnog hardware-a. Većina njih je javno dostupna putem interneta sa mogućnošću unaprijeđenja, proširenja i primjene. Kao primjer možemo navesti OpenCV biblioteku otvorenog koda koja nudi niz funkcionalnosti iz segmenta računalnog vida, a koja podržava niz programskih jezika poput Pythona, Jave C++ , te je dostupna za različite platforme poput Windows, Linux, iOS, Android i druge, Ovaj rad prikazuje programsku i hardversku povezanost u jednu zajedničku cjelinu, javno dostupnih alata i uređaja koje ne spadaju u kategorijeu „osobnog računala“ nego modularnog, a koje podržava i omogućuje sve potrebne funkcije u svrhu prepoznavanja.

Kao fizikalna i vizualna bića nije nam teško prepoznati i označiti određeni objekt, ali isto prikazati kroz računalni vid zahtjeva malo drugačiji pristup.

2. Razvojno okruženje za rad



Slika 2.1 Shematski prikaz razvojnog okruženja

Za ovaj rad koristit ćemo zadnju verziju *Raspberry Pi* uređaja sa svim potrebnim dijelovima međusobno povezane kao što je prikazano na slici 2.1, a kao osnova poslužit će uređaj oznake *Raspberry Pi 4B* sljedećih specifikacija:

- **SoC:** Broadcom BCM2711B0 quad-core A72 64-bit – 1.5 GHz
- **GPU:** Broadcom VideoCore VI
- **RAM:** 2 GB LPDDR4 SDRAM
- **Povezivanje mreža:** 2.4 GHz i 5 GHz – 802.11 b/g/n/ac WLAN, Gigabit LAN, Bluetooth 5.0 BLE
- **Spremište:**MicroSD
- **GPIO:** 40-pin GPIO
- **Povezivanje:** 2 x microHDMI 2.0, 3.5 mm analogni audio-video, 2 x USB 2.0, 2 x USB 3.0, CameraSerial Interface (CSI), Display Serial Interface (DSI), USB-C priključak za napajanje

Za vizualizaciju koristit će se VNC način povezivanja za koji može poslužiti osobno računalo priključeno na zajedničku mrežu. Isto se postavlja odmah nakon snimanja *slike* operativnog sustava na microSD karticu na način da se u *boot* particiji kreira prazna datoteka naziva *SSH*, a čiji će detalji biti tekstualno opisani u nastavku rada, odnosno u praktičnom dijelu ovog rada.

Kao spremište poslužit će microSD kartica ukupnog kapaciteta od 32 GB sa instaliranim najnovijim Raspbian Buster OS na kojeg će se potom instalirati sve potrebne datoteke za provedbu praktičnog dijela.

Za dovoljnu energiju potreban je strujni adapter sa izlaznim USB-C priključkom izlazne specifikacije od 5V i 3A. Stariji modeli Raspberry Pi uređaja koriste strujne adaptere sa slabijom izlaznom strujom (1,5 – 2 A).

Kako bi naš rad bio izvediv potrebna je i video kamera. Moguće su dvije varijante od koje je jedna spajanje odgovarajućeg modula - kamere na pripadajući CSI priključak, ali za naš rad koristit ćemo web kameru koju ćemo povezati USB priključkom.

3. Algoritmi za detekciju lica

U procesu detekcije lica koriste se različiti algoritmi, a u nastavku ćemo opisati one najznačajnije.

- Haarove kaskade (Viola - Jones)
- Histogram orijentiranih gradijenata (HOG)
- Metoda potpornih vektora (SVM)
- Metoda dubokog učenja

3.1. Haarove kaskade (Viola - Jones)

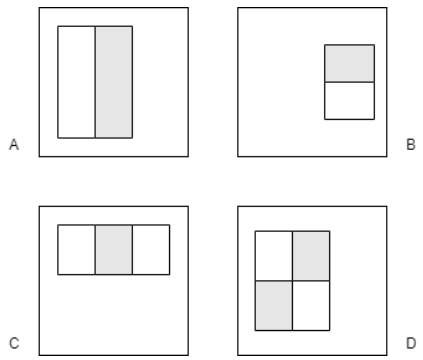
Algoritam za automatsko otkrivanje objekata ima namjenu da u realnom vremenu detektira objekte na slici ili video-zapisu. Algoritam koji je nastao 2001. godine od strane Paul Viole i Michael-Jonesa. Vezano na značajku da se radi o starijoj aplikaciji, algoritam Viola-Jones se pokazao kao moćan alat za prepoznavanje lica u realnom vremenu, a ima i mogućnost prepoznavanja nekih ostalih objekata. [1]

Algoritam ima 4 glavna koraka:

- Računanje Haarovih značajki
- Stvaranje integralne slike
- Adaboost trening
- Kaskadni klasifikatori

Računanje Haarovih značajki

Sva lica dijele nekoliko sličnih svojstava kao na primjer: očna regija je tamnija od gornjih obraza, područje nosa je svijetlije od očiju, a pozicija i veličina očiju, usta i nosa formiraju podudarne crte lica.



Slika 3.1 Četiri različite vrste značajki koje se koriste u okviru

Kod ovog koraka se uzimaju u obzir susjedna pravokutna područja na određenom mjestu za detekciju, zbraja se intenzitet piksela u svakoj regiji i izračunava razlika zbrojeva *(Slika 3.1).*

Viola-Jones koristi tri vrste značajki:

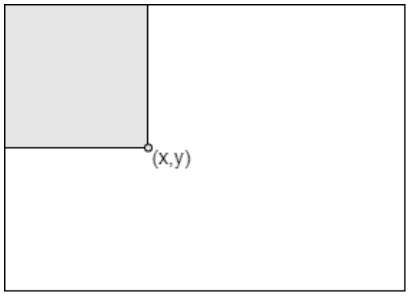
Značajka dva pravokutnika (Edge Features)- je razlika između suma piksela unutar dva pravokutna područja

Značajka tri pravokutnika (Line Features) - računa sumu dva vanjska pravokutnika oduzete od sume centralnog pravokutnika

Značajka četiri pravokutnika (Four Recktangle Features) - računa razliku između dijagonalnih parova pravokutnika

Stvaranje integralne slike

Značajke pravokutnika se mogu izračunati veoma brzo koristeći posredni prikaz za sliku koja se naziva integralna slika.Integralna slika *(Slika 3.2)* na poziciji *x,y* sadrži sumu piksela koji se nalaze iznad i prema lijevo od točke *x,y.*



Slika 3.2 Vrijednost integralne slike na točki (x,y)

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x', y') \qquad (3.1)$$

gdje u *(Formula 3.1) ii(x,y)* je integralna slika, a *i(x,y)* je originalna slika.

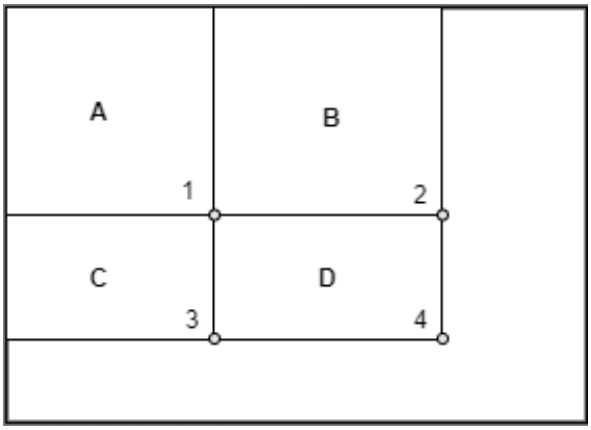
Integralna slika se može izračunati u jednom prolazu preko originalne slike uporabom formule:

$$s(x,y) = s(x,y-1) + i(x,y) \quad (3.2)$$

$$ii(x,y) = ii(x-1,y) + s(x,y) \quad (3.3)$$

gdje je *(Formule 3.2, 3.3) s(x,y)* kumulativni zbroj redaka, $s(x,-1) = 0$ i $ii(-1,y) = 0$.

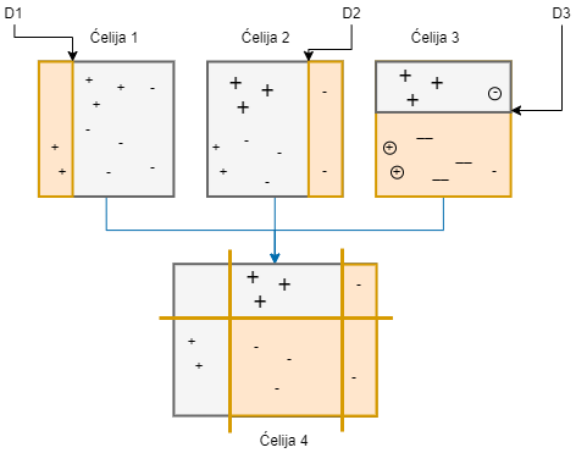
Uporabom integralne slike svaka pravokutna suma može se izračunati u četiri reference niza, prema tome razlika između dvije pravokutne sume može se izračunati u osam referenci. Značajke od dva pravokutnika koja uključuju susjedne pravokutne zbrojeve mogu se izračunati u šest referenci polja, tri pravokutnika u osam referenci polja, a u primjeru značajki sa četiri pravokutnika treba devet referenci polja.



Slika 3.3 Primjer značajki sa 4 pravokutnika

Suma piksela unutar pravokutnika oznake *D* se može izračunati kroz četiri reference niza *(Slika 3.3)*. Vrijednost integrala u točki 1 je suma piksela u pravokutniku oznake *A*. Vrijednost u točki 2 je suma *A+B*, u točki 3 je suma *A+C* i u točki 4 je *A+B+C+D*. Prema tome suma unutar pravokutnika oznake *D* je *4+1-(2+3)*. Integralna slika je u biti dvostruki integral slike (prvo duž redaka, a potom duž stupaca).

Adaboost (Adaptive Boosting)



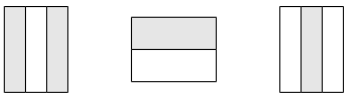
Slika 3.4 Rekonstrukcija primjera jakog klasifikatora

Adaboost je meta-algoritam strojnog učenja razvijen od strane Yoav Freunda i Roberta Schapirea i koji se može koristiti u kombinaciji s drugim vrstama algoritama učenja radi poboljšanja performansi. Izlaz ostalih algoritama učenja kombinira se u zbroj koji predstavlja konačan izlaz jakog klasifikatora *(Slika 3.4)*.

Algoritam uči iz slika na ulazu,čime se mogu odrediti lažni pozitivni i stvarni negativni što mu omogućava da bude što precizniji. Kada bi se pregledali svi mogući položaji i kombinacije tada bi dobili jako precizan model. Takav trening može biti veoma opsežan (vremenski) zbog velikog broja mogućnosti i kombinacija koje se moraju provjeriti za svaki pojedinačni kadar ili sliku. Primjerice, u standardnom potprozoru veličine 24x24 piksela nalazi se ukupno 162 336 mogućih značajki. Okvir za otkrivanje koristi Adaboost algoritam učenja za odabir najboljih značajki kao i za trening i izgradnju jakog klasifikatora.

Kaskadni klasifikatori

Sastoje se od zbirke faza gdje je svaka zbirka skup slabih klasifikatora. Svaka faza se trenira tehnikom pojačanja. Ona pruža mogućnost treniranja preciznog klasifikatora uz pomoć prosječnih odluka donešenih od strane slabijih klasifikatora.



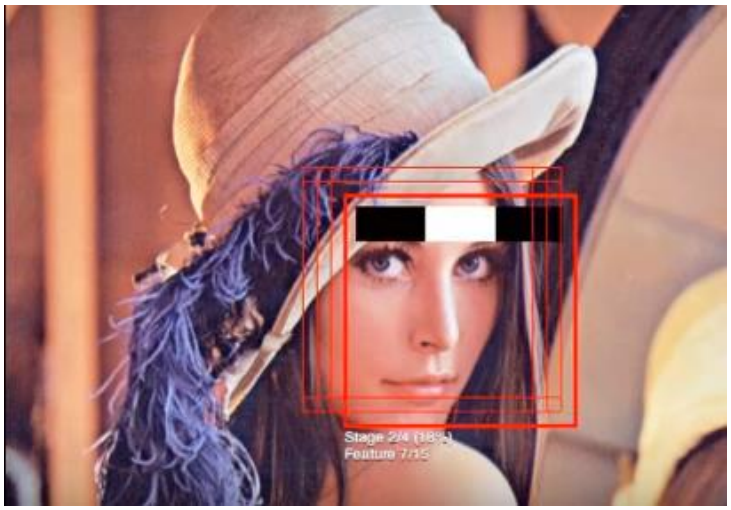
$$F(x) = a_1 f_1(x) + a_1 f_1(x) + a_1 f_1(x) + \ldots$$

Svaka faza klasifikatora označava određenu regiju definiranu trenutnom pozicijom prozora pozitivnom ili negativnom. Pozitivna oznaka označava da je neki objekt pronađen u prozoru dok negativna oznaka označava da treženi objekt u prozoru nije pronađen. Ako je oznaka negativna tada je klasifikacija ovog prostora završena i detektor pomiče prozor na sljedeće mjesto, a ako je oznaka pozitivna tada klasifikator prebacuje prostor u sljedeću fazu. Detektor označava objekt koji je pronađen na trenutnom mjestu prozora kada završna faza klasificira područje kao pozitivno. Faze su izrađene tako da se u što kraćem vremenskom roku odbace negativni uzorci. S obzirom da velika većina prozora ne sadrži traženi objekt, potrebno je dovoljno vremena da se nađu stvarni pozitivni rezultati.

Ukupno postoje tri stanja uzoraka:

- Pravi pozitivan – kada je pozitivan uzorak ispravno klasificiran
- Lažni pozitivan – kada je negativan uzorak pogrešno klasificiran kao pozitivan
- Lažni negativan – kada je pozitivan uzorak pogrešno klasificiran kao negativan

Dodavanjem više faza se smanjuje sveukupni broj lažno pozitivnih uzoraka za što je potrebno osigurati skup pozitivnih slika s određenim prostorima koje će se koristiti kao pozitivni uzorci, a uz pomoć alata za označavanje slika se mogu okvirima označiti željeni objekti iz čega se kreira tablica pozitivnih uzoraka. Uz navedeno potrebno je osigurati i set negativnih slika iz kojih funkcija automatski generira negativne uzorke kako bi se postigla prihvatljiva točnost detektora.



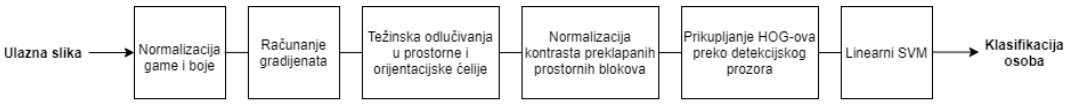
Slika 3.5 Rezultat Haarovih kaskada (izvor. https://vimeo.com/12774628)

3.2. Histogram orijentiranih gradijenata (HOG)

Deskriptor značajki koristi se u računalnom vidu i obradi slike u svrhu otkrivanja objekata. Koncept HOG-a prvi je napisao Robert K. McConnell 1986. godine, a 2005. godine Navneer Dalal i Bill Tiggs predstavili su dopunski rad na deskriptorima HOG-a u kojem su se usredotočili na detekciju pješaka na statičkim slikama, te potom i otkrivanje ljudi u video zapisima. [2]

S obzirom da je otkrivanje ljudi na slikama veoma zahtjevan posao prvenstveno zbog različitih izgleda i poza pa je iz tog razloga prvo potreban skup značajki koje omogućavaju da se izdvoji ljudski oblik od pozadine. HOG metoda slična je metodama histogramima rubne orijentacije i SIFT deskriptorima, no razlika je u tome što se HOG metoda izračunava na gustim mrežama ravnomjerno raspoređenih ćelija, a koristi se i preklapanje lokalnog kontrasta za poboljšanje točnosti. Osnovni klasifikator koristi linearni SVM što rezultira jednostavnošću i brzinom. Osnovna ideja HOG deskriptora je da se izgled i oblik nekog objekta unutar slike može opisati uz pomoć raspodjele intenziteta gradijenata ili smjerova rubova. Slika se podijeli u mala povezana područja koja se zovu ćelije, za piksele unutar svake ćelije se sastavlja HOG, pa se takvo spajanje histograma naziva deskriptor. Kako bi se poboljšala točnost, lokalni histogrami mogu normalizirati kontrast računanjem intenziteta većeg područja slike (bloka) koji se potom koristi za normalizaciju svih ćelija unutar bloka.

HOG djeluje na lokalnim ćelijama, invarijantan je geometrijskim i fotometrijskim transformacijama izuzev orijentacije objekata.



Slika 3.6 Prikaz slijeda izdvajanja značajki i prepoznavanja objekata

3.2.1. Normalizacija game i boja

Normizacija slike svjetline (*grayscale*), RGB i Lab prostora boja sa gama postavkama kod ulaznih piksela rezultira skromnim napretkom. RGB i Lab modeli boja daju usporedivu vrijednost, ali u usporedbi sa grayscale-om, performanse se smanjuju za 1.5 % na 10^{-4} FPPW-a (*False Positive Per Window*). Korijen gama kompresije svakog kanala boje povećava učinkovitost na niskim FPPW-ovima (za 1 % na 10^{-4} FPPW-a), ali log kompresija je prejaka, te smanjuje učinkovitost za 2 % na 10^{-4} FPPW-a. [2]

3.2.2. Računanje gradijenata

Prethodni korak se može i izostaviti jer normizacija ovog deskriptora na kraju postiže isti rezultat. Za računanje gradijenata korišteno je Gaussovo uglađivanje sa nastavnim diskretnim derivacijskim maskama. Testirano je nekoliko skala za uglađivanje uključujući i σ=0 (nijedan). Testne maske su sadržavale različite 1-D derivate točaka (necentrirani [-1,1], centrirani [-1,0,1], kubno korigirani [1,-8,0,8,-1]) kao i 3x3 Sobel maske i 2x2 dijagonale $\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$ kao najkompaktnija centrirane 2-D derivacijske maske. Korištenjem većih maski smanjuju se performanse i uglađivanja. Za Gaussovu derivaciju, promjenom vrijednosti σ=0 na vrijednost σ=2 smanjuje se povratna vrijednost sa 89 na 80 % na 10^{-4} FPPW-a. Na σ=0 kubno korigirani 1-D filter je otprilike 1 % lošiji nego [-1,0,1] na 10^{-4} FPPW-a, a kod 2x2 dijagonalnih masaka je za 1,5 % lošije. Korištenje [-1,1] derivacijskih maski isto tako smanjuje performanse za 1,5 %. Za slike u boji izračunati su zasebni gradijenti za svaki kanal boje, te je uzet onaj s najvećom normom vektora gradijenta piksela. [2]

3.2.3.Prostorno/orijentacijsko vezivanje

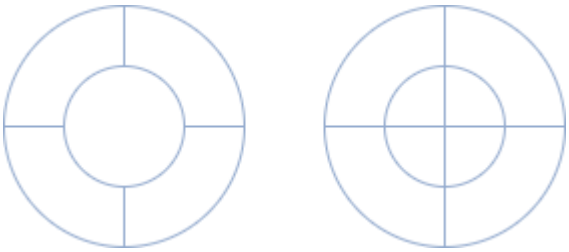
Svaki piksel računa težinsku vrijednost za kanal rubno-orijentiranog histograma temeljenog na orijentaciji elemenata gradijenata usmjerenog na njega, a vrijednosti se zatim sakupljaju u orijentacijske ćelije, a koje mogu biti pravokutne ili radijalne (log-polarne). Orijentacijske kutije su ravnomjerno raspoređene na0-180˚(gradijent bez predznaka) ili 0-360˚ (gradijent s predznakom). Nazubljenost se reducira na način da se vrijednosti interpoliraju bilinearno između susjednih ćelija u orijentaciji i poziciji. Vrijednost je funkcija veličine gradijenata piksela, bilo sama veličina, kvadrat, korijen ili isječen oblik veličine koja predstavlja meku prisutnost/odsutnost ruba na pikselu. Dalal i Triggs zaključili su da su nepotpisani gradijenti korišteni u 9 kanala histograma najbolji kod detekcije ljudi.[2]

3.2.4. Normalizacija i deskriptor blokova

Jačina gradijenata varira zbog lokalnih varijacija u osvjetljenju i kontrastu pozadine, zbog čega je veoma bitna normalizacija kontrasta radi dobre učinkovitosti. Većina ispitanih shema normalizacije vršene su na temelju grupiranja ćelija u veće blokove i na temelju normalizacije kontrasta svakog bloka zasebno. Završni deskriptor je vektor svih komponenti normaliziranih ćelija iz svih blokova u prozoru za detekciju. U praksi se blokovi obično preklapaju, tako da svaki skalar ćelije doprinosi završnom deskriptoru vektora nekoliko komponenti i svaki normaliziran u odnosu na drugi blok. Kada se poveća preklapanje tada se poveća i učinkovitost za 4 % na 10^{-4} FPPW-a ako promijenimo preklapanje sa 0 na 16-struko područje preklapanja ili četverostruko linearno preklapanje. Za testiranje se koriste dvije klase geometrije blokova. Oni pravokutni su podijeljeni na rešetke pravokutnih oblika prostornih ćelija, te kružni blokovi koji su podijeljeni u log-polarne oblike iz čega je izvedeno – R-HOG(Rectangular) i C-HOG (Circular).

R-HOG – blokovi imaju mnogo sličnosti sa SIFT metodologijom, no računaju se u gustim mrežama na jednoj ljestvici bez dominantnih orijentacijskih poravnanja i koriste se kao dio većeg kodnog vektora koji implicitno kodira prostorni položaj u odnosu na detekcijski prozor. Predstavljene su sa 3 parametra: broj ćelija po bloku, broj piksela po ćeliji i broj kanala po ćelijskom histogramu. [2]

C-HOG – prikazuju se u dvije varijante: varijanta sa jednom središnjom ćelijom i varijanta s kutno podijeljenom središnjom ćelijom.



Slika 3.7 C-HOG varijante

Predstavljeni su sa 4 parametra: brojem kutnih i radijalnih ćelija, polumjerom središnje ćelije i faktorom ekspanzije radijusa dodatnih radijalnih ćelija. Središnji radijus od 4 piksela i faktor ekspanzije 2 daje najbolji rezultat.

Postoje 4 različite metode za normalizaciju blokova *(Formule 3.4, 3.5 i 3.6)*, ako je *v* nenormirani vektor koji sadrži sve histograme u određenom bloku, $\|v\|_k$ njegova *k* norma za *k=1,2* i ϵ mala konstanta iz čega je faktor normalizacije jedan od navedenih:

$$\text{L2 norma: } f = \frac{v}{\sqrt{\|v\|_2^2 + e^2}} \qquad (3.4)$$

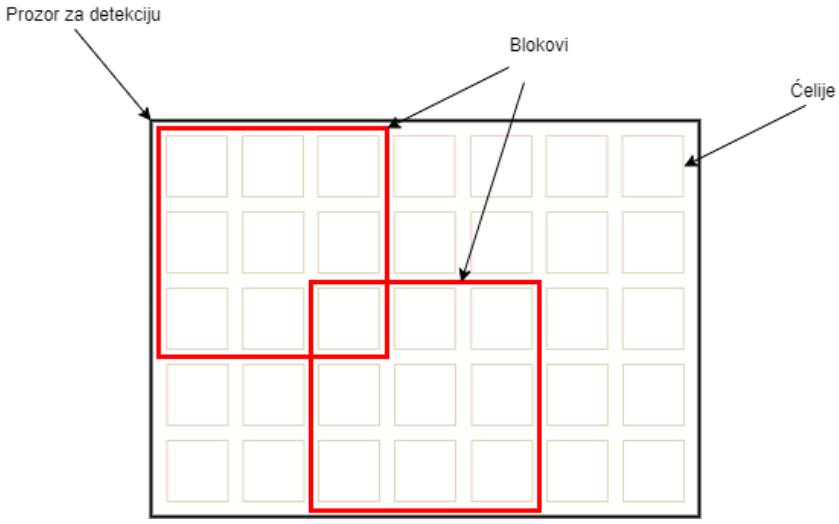
L2-hys: L2 norma s ograničenim maksimalnim vrijednostima *v* na 0,2 i renormizacijom

$$\text{L1 norma: } f = \frac{v}{(\|v\|_1 + e)} \qquad (3.5)$$

$$\text{L1-sqrt: } f = \sqrt{\frac{v}{(\|v\|_1 + e)}} \qquad (3.6)$$

3.2.5. Prozor detekcije

Prozor detekcije veličine 64x128 piksela sadrži oko 16 piksela na marginama oko osobe i na sve 4 strane.[2] Smanjenjem sa 16 na 8 piksela (48x112 pikselni prozor detekcije), smanjuju se performanse za 6 % na 10^{-4} FPPW-a. Primjer prozora za detekciju, blokova i ćelija prikazan je na slici *3.8*.

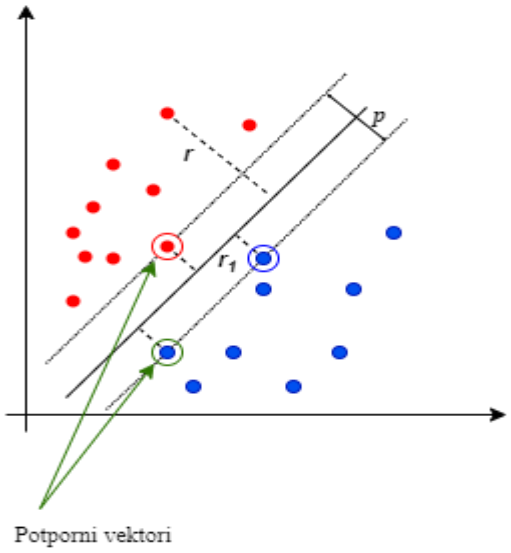


Slika 3.8 Grafički prikaz prozora za detekciju, blokova i ćelija

3.3. Metoda potpornih vektora (SVM)

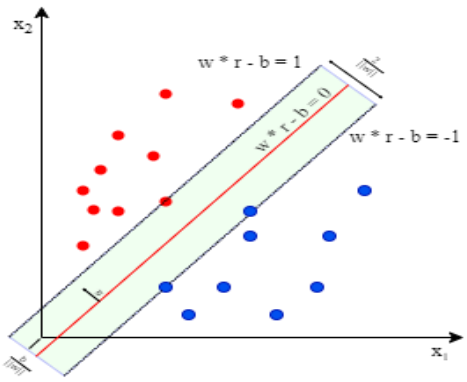
Označava pristup klasifikaciji gdje granicu klase čine potporni vektori. Prilikom detektiranja anomalija, jednoklasni SVM se koristi za definiranje normalne klase, dok se sve točke koje se nalaze van granica definiraju kao anomalije. SVM spada u kategoriju strojnog učenja s nadzorom povezanih algoritmima učenja koji analiziraju podatke korištene za klasifikaciju i regresijsku analizu koristeći analizu podataka i prepoznavanje uzoraka. Metode ovise o strukturi i atributima klasifikatora. Linearni SVM je najpoznatiji klasifikator koji predviđa svaku ulaznu klasu uzorka između dvije moguće klasifikacije. SVM gradi hiper-ravninu ili skup hiper-ravnina kako bi klasificirao sve ulaze u više dimenzionalni ili u beskonačni prostor. Najbliže vrijednosti klasifikacijskih margina nazivaju se potporni vektori čiji je cilj povećati marginu između hiper-ravnina i potpornih vektora *(Slika 3.9)*. Kvalitetno odvajanje postiže se kada hiper-ravnina ima najveću udaljenost do podatkovne točke bilo koje klase (funkcionalne margine) kod koje vrijedi pravilo što veća udaljenost to manja

generalizacijska greška klasifikatora. Potporni vektori su u biti podatkovne točke na rubu ravnine koje su najbliže hiper-ravnini koja ih razdvaja. [3]



Slika 3.9 Grafički prikaz potpornih vektora

Kod setova podataka za trening *n* točaka forme $(\vec{x}_i, Y_1), \ldots, (\vec{x}_n, Y_n)$ gdje je Y_i ili 1 ili -1 ovisno o klasi kojoj $\vec{x}_i$ pripada. Svaki $X_i$je p-dimenzionalan vektor. Cilj je naći maksimalnu hiper-ravninu koja odvaja grupu točaka $X_i$za $Y_i=1$ i za $Y_i=-1$ gdje je najveća udaljenost između hiper-ravnine i najbliže točke $\vec{x}_i$. Svaka hiper-ravnina se može opisati kao skup točaka $\vec{x}$ kroz $\vec{w} * \vec{x} - b = 0$ gdje je $\vec{w}$ vektor normale hiper-ravnine koji nije nužno jedinični vektor, dok parametar $\frac{b}{||\vec{w}||}$ pokazuje pomak hiper-ravnine od ishodišta duž normale vektora $\vec{w}$ *(Slika 3.10)*.



Slika 3.10 Hiper-ravnine i margine

Prema vrsti razdvajanja postoje dvije vrste:

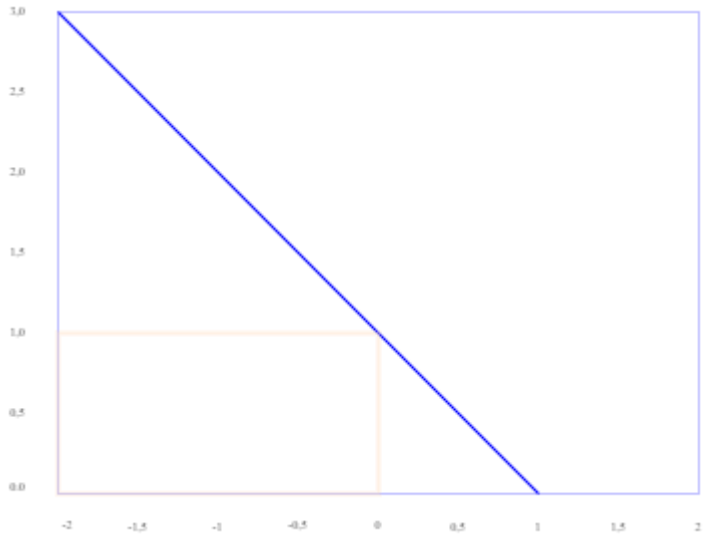
- odvajanje tvrdom marginom
- odvajanje mekom marginom

Odvajanje tvrdom marginom

Kod podataka koje je moguće odvojiti koriste se dvije paralelne hiper-ravnine koje odvajaju dvije klase podataka tako da je udaljenost između njih što veća. Područje graničenja tih dviju hiper-ravnina naziva se margina, a maksimalna margina hiper-ravnine je margina koja leži na polovini udaljenosti između njih. Hiper-ravnine se mogu opisati jednadžbom $\vec{w} * \vec{x} - b = 1$ i $\vec{w} * \vec{x} - b = -1$. Geometrijski prikazano, udaljenost između dvije hiper-ravnine je $\frac{2}{||\vec{w}||}$ tako da maksimiziramo udaljenost između ravnina koje želimo minimizirati $||\vec{w}||$, te se takva udaljenost izračunava korištenjem udaljenosti između točke i ravnine. Pritom se mora spriječiti da točke upadaju u marginu pa iz toga se dodaje ograničenje da je svaki $\vec{w} * \vec{x}_i - b \geq 1$ za svaki $1 \leq i \leq n$.

Odvajanje mekom marginom

Kod podataka koji ne dopuštaju odvajanje hiper-ravninom koristi se razdvajanje mekom marginom što opisuje hiper-ravninu koja odvaja mnoge, ali ne i sve točke podataka i to primjenom funkcije gubitka *(Slika 3.11)*.



Slika 3.11 Grafički prikaz gubitka

Iz navedenog proizlazi funkcija *max(0.1- Y_i ($\vec{w} * \vec{x} - b$))* gdje je Y_i*i*-ti cilj, a $\vec{w} * \vec{x} - b$ trenutni izlaz *(Formula 3.7)*. Pritom, ako $\vec{x}_i$ leži na ispravnoj strani margine tada je ova funkcija jednaka nuli. Za podatke s pogrešne strane margine, vrijednost funkcije je proporcionalna udaljenosti od margine koju želimo svesti na minimum uz pomoć:

$$\left[\frac{1}{n}\sum_{i=1}^{n} max\left(0, 1 - y_i(\vec{\omega} * \vec{x}_i - b)\right)\right] + \lambda||\vec{\omega}||^2 \qquad (3.7)$$

gdje λ prikazuje kompromis između povećanja veličine margine i da $\vec{x}_i$ leži na ispravnoj strani margine. Gubitak u funkciji postaje zanemariv čime će se ponašati kao SVM sa tvrdom marginom.

4. Metode i razvoj dubokog učenja

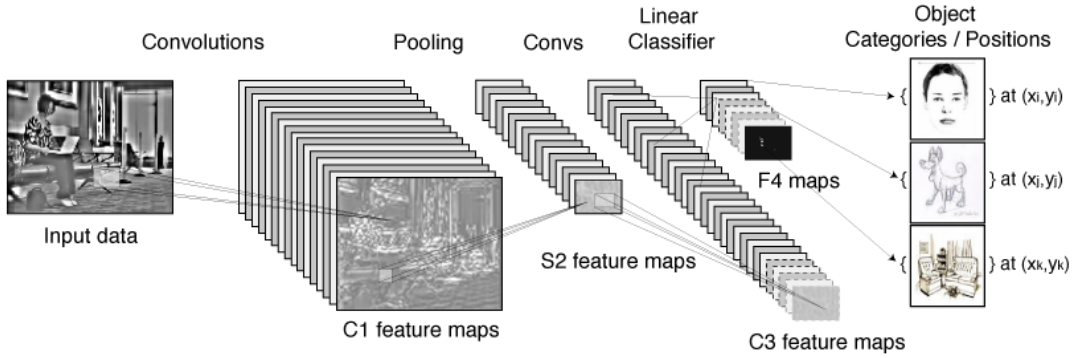
Metode dubokog učenja koriste se za prepoznavanje predmeta, lica, čovjekove aktivnosti, čovjekovih poza, skupova podataka (rad sa slikama). [4]

Glavne metode dubokog učenja su:

- Konvolucijske neuronske mreže
- Duboke mreže vjerovanja i Deep Boltzmann-ovi strojevi
- Složeni autoenkoderi

4.1. Konvolucijske neuronske mreže

Inspirirane su strukturama vizualnog sustava, a sastoje se od tri glavne vrste neuronskih slojeva: konvolucijskih slojeva, slojeva za udruživanje, te potpuno povezanih slojeva. Svaki od slojeva ima drugačiju ulogu, a svaki od slojeva pretvara ulazni volumen u izlazni volumen aktivacije neurona, naposlijetku vodeći prema potpuno povezanim slojevima, rezultirajući preslikavanjem ulaznih podataka u 1D svojstvo vektora. Konvolucijske neuronske mreže *(Slika 4.1)* pokazale su se veoma uspješne u aplikacijama računalnog vida za prepoznavanje lica, detekciju objekata, računalnog vida u robotici, te autonomnih vozila.



Slika 4.1 Prikaz arhitekture konvolucijske neuronske mreže (izvor. https://i.stack.imgur.com)

4.1.1. Konvolucijski slojevi

Ulaz u konvolucijski sloj je tenzor oblika (broj slika) x (širina slike) x (visina slike) x (dubina slike). Slika nakon prolaska kroz konvolucijski sloj generira mape značajki, s oblikom (broj slika) x (širina značajke) x (visina značajke) x (broj kanala) mape značajki.

4.1.2. Slojevi za udruživanje

Slojevi za udruživanje su zaduženi za smanjenje prostornih dimenzija ulaznog volumena za slijedeći konvolucijski sloj, a da pritom ne utječu na dimenziju dubine volumena. Smanjivanje veličine dovodi i do gubitka podataka koji koriste mrežu na način da se smanjenjem veličine smanjuje i vrijeme računskih operacija za nadolazeće slojeve mreže.

4.1.3. Potpuno povezani slojevi

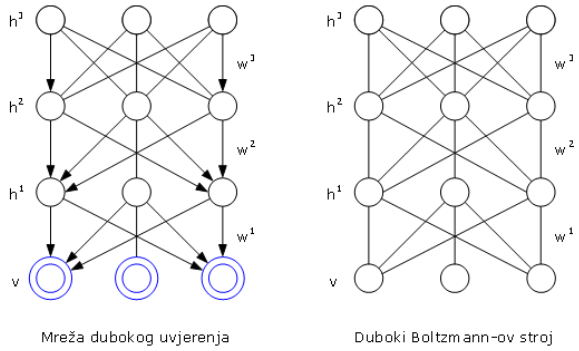
Nakon nekoliko slojeva konvolucijskih slojeva i slojeva za udruživanje, zaključivanje visoke razine u neuronskoj mreži vrši se preko potpuno povezanih slojeva. Neuroni u ovom sloju imaju potpune veze sa svim aktivacijama u prethodnom sloju pa se njihova aktivacija može izračunati matričnim množenjem praćena promjenom nagiba. Potpuno povezani slojevi na kraju pretvaraju mape 2D značajki u 1D vektor značajki.

Arhitektura konvolucijskih neuronskih mreža koristi tri konkretne ideje: lokalna receptivna polja, vezane važnosti i prostorno poduzorkovanje. Na temelju lokalnog receptivnog polja, svaka jedinica u konvolucijskom sloju prima ulaze iz skupa susjednih jedinica koje pripadaju prethodnom sloju što daje neuronima sposobnost izvlačenja vizualnih osobina poput rubova ili uglova. Ove značajke se zatim kombiniraju sa sljedećim konvolucijskim slojevima kako bi se otkrile značajke višeg reda. Koncept vezanih važnosti ograničava skup jedinica koji imaju jednake važnosti. Jedinice konvolucijskog sloja organizirane su u ravninama, a sve jedinice u ravnini dijele isti skup važnosti. Svaka ravnina je odgovorna za konstrukciju određene

značajke. Izlazi iz ravnina se nazivaju mape značajki, a svaki sloj se sastoji od više ravnina tako da se na svakoj lokaciji mogu konstruirati višestruke mape.

4.2. Duboke mreže vjerovanja i Deep Boltzmann strojevi

Duboke mreže vjerovanja (Deep Belief Network, DBN) i Deep Boltzmann strojevi (Deep Boltzmann Machines, DBM) su modeli dubokog učenja,te koriste ograničeni Boltzmann-ov stroj kao modul učenja kao generativna stohastička neuronska mreža. Duboke mreže vjerovanja imaju neizravne veze na gornja dva sloja koje tvore ograničeni Boltzmann-ov stroj i usmjerene veze na donje slojeve dok Deep Boltzmann-ovi strojevi imaju neizravne veze između svih slojeva mreže*(Slika 4.2)*.



Slika 4.2 Shematski prikaz DBN-a i DBM-a (gronja dva sloja)

4.2.1. Ograničeni Boltzmann-ov stroj

Ograničeni Boltzmannov stroj je neusmjerni grafički model sa stohastičkim vidljivim varijablama $v \epsilon \{0, 1\}^D$ i stohastičkim skrivenim varijablama $h \epsilon \{0, 1\}^F$, gdje je svaka vidljiva varijabla povezana sa svakom skrivenom varijablom. Ograničeni Boltzmannov stroj je varijanta Boltzmannovog stroja, s ograničenjem da su vidljive jedinice skrivene jedinice i moraju činiti dvostrani graf.

4.2.2. Duboke mreže vjerovanja

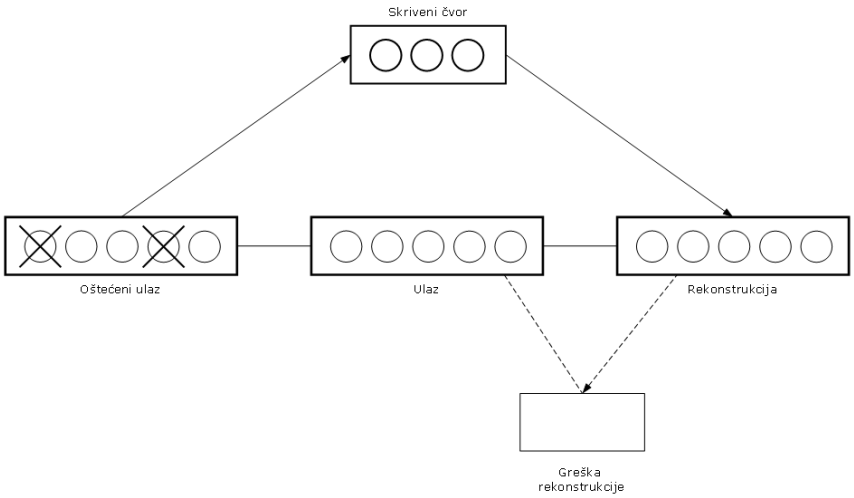
Duboke mreže vjerovanja su modeli koji osiguravaju zajedničku raspodjelu vjerojatnosti na promatranim podacima i oznakama koji se formiraju slaganjem ograničenih Boltzmannovih strojeva. Duboke mreže vjerovanja u početku koriste sloj po sloj kao način učenja kako bi inicijalizirale duboku mrežu i prilagodile važnost zajedno sa željenim rezultatima. To su grafički modeli koji uče izvlačenje dubokog hijerarhijskog prikaza podataka o treningu.

Značajan nedostatak dubokih mreža vjerovanja je što se ne uzimaju u obzir dvodimenzionalne strukture ulazne slike što može značajno utjecati na performanse i primjenjivost.

4.3. Složeni autoenkoderi

Složeni autoenkoderi koriste autoenkoder kao glavnu komponentu za izradu duboke mreže vjerovanja.

Autoenkoder ima zadatak kodirati ulaz *x* u reprodukciju *r(x)*, iz čega proizlazi da je traženi izlaz u biti njegov ulaz. Kod ovog postupka, greška rekonstrukcije se minimizira, a odgovarajući kod je naučena značajka *(Slika 4.3)*.



Slika 4.3 Uklanjanje šuma u autoenkoderu

Kada se koristi jedan skriveni linearni sloj sa srednjim kvadratom pogreške za treniranje mreže, tada se *k* skrivene jedinice uče za reprodukciju ulaza u rasponu prvih *k* vrijednosti glavnih komponenti podataka. Ako je skriveni sloj nelinearan tada se autoenkoder ne ponaša kao PCA. Kvadrat pogreške računa se prema:

$$L = ||x - f(r(x))||^2 \qquad (4.1)$$

gdje je f dekoder, a $f(r(x))$ reprodukcija napravljena pomoću modela *(Formula 4.1)*.

4.3.1. Autoenkoderi za uklanjanje šumova

Predstavljaju „nasumičnu“ verziju autoenkodera gdje je ulaz „nasumično“ oštećen dok se neoštećeni ulaz koristi za ciljani izlaz rekonstrukcije. Kod ovog procesa su bitna dva koraka:

- Autoenkoder pokušava kodirati ulaz i pritom zapamtiti podatke o unosu
- Autoenkoder pokušava poništiti utjecaj nasumičnog oštećenja primjenjenog na ulaz

Proces započinje postavljanjem broja ulaza na nulu nakon čega autoenkoder pokušava predvidjeti oštećene vrijednosti u odnosu na neoštećene vrijednosti.

Sposobnost predviđanja bilo koje podskupine varijabli od preostalih nudi dovoljan uvjet za potpuno dohvaćanje zajedničke raspodjele između skupa varijabli.

4.3.2. Složeni autoenkoderi za uklanjanje šuma

Duboka mreža može se formirati slaganjem autoenkodera za uklanjanje šuma (Stacked Denoising Autoencoders, SdAs) dovođenjem izlaznog koda nižeg sloja kao ulaz u trenutni sloj. Predtrening bez nadzora takve strukture vrši se sloj po sloj, gdje je svaki sloj istreniran kao autoenkoder za uklanjanje šuma koji minimizira grešku rekonstrukcijom ulaza (izlaz prethodnog sloja je ulaz trenutnog). Kada je prvih *k* slojeva istrenirano tada je moguće istrenirati i *(k+1)* sloj pošto je moguće izračunati izlaz sloja ispod. Nakon što završi pretraga svih slojeva, mreža prolazi kroz drugu fazu treninga za fino podešavanje odnosno optimiziranje predviđanja pogreške. Sloj logističke regresije se dodaje na izlazni kod izlaznog sloja mreže, nakon čega se

izvedena mreža trenira kao višeslojni perceptron uzimajući u obzir samo kodirane dijelove svakog autoenkodera. Princip treninga složenih autoenkodera je sličan dubokim mrežama vjerovanja. U nastavku *(Slika 4.4)* slijedi usporedba značajki modela za konvolucijske neuronske mreže, duboke mreže vjerovanja/Deep Boltzmann strojeva i složenih sutoenkodera za uklanjanje šuma.

Značajke modela	CNN	DBN/DBM	SdAs
Učenje bez nadzora	-	+	+
Učinkovitost treninga	-	-	+
Učenje značajki	+	-	-
Skaliranje, rotacija, translacija	+	-	-
Generalizacija	+	+	+

Slika 4.4 Usporedba CNN, DBN/DBM i SdAs-a

5. Primjer poravnavanja lica pomoću stabla odlučivanja (ERT)

Poravnavanje lica pomoću stabla odlučivnja (Ensemble of Regression Trees, ERT) je algoritamkoji provodi poravnavanje lica u milisekundama uz visoku preciznost. Poravnavanje lica može se riješiti kaskadom regresijskih funkcija i to kroz dva ključna elementa:

Kod prvog se radi o indeksiranju intenziteta piksela u odnosu na trenutnu procjenu oblika. Značajke kod vektorskog prikaza slike lica mogu varirati zbog deformacije oblika, ali i ostalih čimbenika poput promjene osvjetljenja što negativno utječe na preciznost procjene. Slika se transformira u normirani koordinatni sustav na temelju trenutne procjene oblika. Ovaj postupak se ponavlja više puta sve do konvergencije.

Kod drugog elementa se radi o rješavanju problema zaključivanja ili predviđanja. Kod testiranja, algoritam za poravnavanje mora procijeniti oblik u vidu višedimenzijskog vektora koji se podudara sa slikovnim podacima i modelom oblika. Određena klasa regresa stvara predviđanja koja se nalaze u linearnom podprostoru definiranom oblicima za trening. Svaki regresor uči pomoću pojačanja gradijenata s kvadratnom funkcijom gubitka pogreške. Set rijetkih piksela se koristi kao ulaz regresora, te se odabire kombinacijom algoritama za pojačanje gradijenata i prethodne vjerojatnosti udaljenosti između parova ulaznih piksela. Prethodna raspodjela pruža algoritmu pojačanu mogućnost pretraživanja velikog broja relativnih značajki. Rezultat predstavlja kaskadu regresora koja može lokalizirati orijentaciju lica. [5]

Ako je $X_I \epsilon R^2$ *x,y* koordinate *i*-te oznake lica na slici *I*, tada vektor $S=(X_1^T, X_2^T ... X_P^T) \epsilon R^{2p}$ označava koordinate svih *p* oznaka lica u slici *I*, a isti vektor se može nazvati i oblikom. $\hat{S}(t)$ označava trenutnu procjenu *S*, a svaki regresor $r_z(.,.)$ predviđa novi vektor iz slike i $\hat{S}^{(t)}$ koji se dodaje trenutnoj procjeni oblika $\hat{S}^{(t)}$ radi poboljšanja procjene.

$$\hat{S}^{(t+1)} = \hat{S}^{(t)} + r_t (I, \hat{S}^{(t)}) \qquad (5.1)$$

Ključna točka kaskade je ta da regresor $r_t$svoja predviđanja temelji na značajkama poput intenziteta piksela izračunatih iz I i indeksiranih relativno u odnosu na trenutnu procjenu oblika $\hat{S}^{(t)}$*(Formula 5.1)*. Pretpostavka da $\hat{S}^{(0)}$ pripada linearnom podprostoru podataka za trening. Početni oblik se odabire kao srednji oblik podataka za trening, centriranog i skaliranog prema izlazu graničnog okvira generičkog detektora lica. Za trening svakog r_t koristi se algoritam povećanja gradijenata stablima odluke sa zbrojem kvadratnih gubitaka.

Kod podataka za trening $(I_1 , S_1), \dots ,(I_n , S_n)$ gdje je svaki I_i slika lica i S_i njezin vektor oblika. Za regresijsku funkciju u kaskadi r_0, iz trening podataka stvara se trostruka slika lica

$$\begin{gathered} \pi \in \{1,\dots,n\} \\ \hat{S}_i^{(0)} \in \{S_1,\dots, S_n\} \setminus S_{\pi i} \\ \Delta S_i^{(0)} = S_{\pi i} - \hat{S}_i^{(0)} \end{gathered} \qquad (5.2)$$

gdje je ukupan broj trostrukih lica postavljen prema $N = nR$ gdje R označava broj korištenih inicijalizacija po slici I_i. Svaka procjena početnog oblika slike se uzorkuje ravnomjerno bez zamjene $\{S_1,\dots,S_n\}$. Dobiveni podaci *(Formula 5.2)* koriste se za učenje regresijeske funkcije r_0i to uporabom pojačanja stabla gradijenta sa zbrojem kvadrata greške gubitka. Navedeni podaci se tada dopunjuju kako bi pružili podatke za treniranje $(I_\pi , \hat{S}_i^{(0)}, \Delta S_i^{(0)})$ i to za regresor r_1 kod postavki sa $t=0$ *(Formula 5.3)*

$$\begin{gathered} \hat{S}_i^{(t+1)} = \hat{S}_i^{(t)} + r_t (I_\pi , \hat{S}_i^{(t)}) \\ \Delta S_i^{(t+1)} = S_{\pi i} - \hat{S}_i^{(t+1)} \end{gathered} \qquad (5.3)$$

Ovaj proces se ponavlja sve dok se ne nauči kaskada T regresora za $r_0 , r_1 \dots r_{t-1}$ koje u kombinaciji daju dovoljnu razinu točnosti.

Svaki regresor r_t se uči uz pomoć algoritma povećanja gradijenta stablima odluke. Na svakom podijeljenom čvoru u regresijskom stablu se donosi odabir na temelju razlike između intenziteta dvaju piksela. Pikseli koji se koriste u testiranjima, nalaze se na pozicijama u i v definiranih u koordinatnom sustavu za srednju sliku. Kod slika lica proizvoljnog oblika poželjno je indeksiranje točaka koje imaju istu poziciju

relativnu prema izvornom obliku kao i kod *u* i *v* pozicija za srednju sliku. Da bi se isto postiglo, slika se može iskriviti na srednji oblik na temelju trenutačne procjene oblika prije izdvajanja značajki, što se može poboljšati ako se koriste prekrivanja samo točaka umjesto cijele slike *(Formula 5.4)*. Ako je k_u indeks oznaka lica u srednjem obliku koji je najbliži *u* i koji definira svoj pomak od *u* kao:

$$\delta x_u = u - \bar{x}_{ku} \tag{5.4}$$

Prema tome oblik S_i definiran u slici I_i , te pozicija u I_i koja je slična poziciji u srednjoj slici oblika je dobiven iz:

$$u' = x_{i,ku} + ((1/S_i)R_i^T \delta x_u) \tag{5.5}$$

gdje sus$_i$ i R_i matrice razmjera i rotacije sličnosti koje transformiraju $s_i u\hat{S}$ srednjeg oblika *(Formula 5.5)*. Razmjer i rotacija mogu smanjiti zbroj kvadrata između točaka oznake lica srednjeg oblika $\bar{x}_j$ i iskrivljenog oblika *(Formula 5.6)*, a slično je definirana i vrijednost *v'*.

$$\sum_{i=0}^{n} || \bar{x}_j - (s_i R_i x_{i,j} + t_i)||^2 \tag{5.6}$$

Svaka podjela uključuje 3 parametra $\theta = (\pi\ u,\ v)$ i primjenjuje se na svaki test i trening prema

$$h(I_{\pi_i}, \hat{S}_i^{(t)}, \theta = \begin{cases} 1 & I_{\pi_i}(u') - I_{\pi_i}(v') > \tau \\ 0 & suprotno \end{cases} \tag{5.7}$$

gdje su vrijednosti *u'* i *v'* definirane upotrebommatrica skaliranja koje prema formuli *(Formula 5.7)* najbolje iskrivljuje $S_i^{(t)}$ u $\bar{S}$. Izračunavanje transformacije sličnosti u testnom vremenu je najzahtjevniji dio koji se vrši samo jednom na kraju svake razine kaskade.

6. Metode za prepoznavanje lica

Jedne od najznačajnijih metoda koje se koriste za prepoznavanje lica, a čiji je primjer dostupan i u Python programskom jeziku su:

- Fisherfaces (koristeći LDA)
- Eigenfaces (koristeći PCA)
- Histogrami lokalnih binarnih uzoraka (LBPH)
- Metode temeljene na SIFT i SURF deskriptorima
- Metode dubokog učenja

6.1. Fisherfaces

Kao i kod svih ostalih sustava za prepoznavanje lica izlaz je slika ili video. Izlaz je identifikacija ili verifikacija subjekata ili subjekta koji se pokazuje u slici ili videu *(Slika 6.1)*.

Slika 6.1 Koraci u prepoznavanju

Fisherfaces metoda uči specifičnu klasu transformacije matrice tako da one ne zahvaćaju i osvjetljenje kao u Eigenfaces metodi, te je veoma učinkovita kod slika koja imaju velike varijacije u osvjetljenju i kod različitih izraza lica. Performanse ove metode ovise i o ulaznim podacima, a rekonstrukcija projicirane slike vrši se kao i kod Eigenfaces metode. Koristi se i metoda glavnih komponenti (Principal Component Analysis, PCA) za linearno projiciranje prostora slike u značajku prostora s malim dimenzijama. [6]

Ako je *X* slučajni vektor sa uzorcima izvučen iz *c* klasa *(Formula 6.1)*

$$X = \{X_1, X_2, \ldots, X_c\} \tag{6.1}$$
$$X_i = \{X_1, X_2, \ldots, X_n\}$$

matrice rasipanja S_Bi S_W izračunavaju se prema formulama *(Formula 6.2, 6.3)*

$$S_B = \sum_{I=1}^{C} N_i (\mu_i - \mu)(\mu_i - \mu)^T \tag{6.2}$$
$$S_W = \sum_{I=1}^{C} \sum_{X_j \epsilon X_i} N_i \quad (x_y - \mu_i)(x_i - \mu_i)^T \tag{6.3}$$

gdje je μukupna srednja vrijednost prema *(Formula 6.4)*

$$\mu = \frac{1}{N} \sum_{i=1}^{N} X_i \tag{6.4}$$

i $\mu_i$je srednja vrijednost klase ϵ *{1, ... ,c} (Formula 6.5)*

$$\mu_i = \frac{1}{|X_i|} \sum_{i=1}^{N} X_j \epsilon X_i \, {}^{X_j} \tag{6.5}$$

Sada algoritam traži projekciju *W (Formula 6.6)* koja maksimizira kriterij razdvajanja klase

$$W_{opt} = argmax_W \frac{|W^T S_B W|}{|W^T S_W W|} \tag{6.6}$$

a rješenje za taj optimizacijski problem nudi rješenje dobiveno

$$S_B V_i = \lambda_i S_w V_i \tag{6.7}$$
$$S_w^{-1} S_B V_i = \lambda_i V_i$$

iz čega dobivamoW_{pca} *(Formula 6.8)* i W_{fld} *(Formula 6.9)*

$$W_{pca} = argmax_W |W^T S_T W| \tag{6.8}$$

$$W_{fld} = argmax_W \frac{|W^T W_{pca}^T S_B W_{pca} W|}{|W^T W_{pca}^T S_w W_{pca} W|} \tag{6.9}$$

matrica transformacije *(Formula 6.10)* koja projicira uzorak u C-1 dimenzijski prostor iz čega je:

$$W = W_{fld}^T W_{pca}^T \tag{6.10}$$

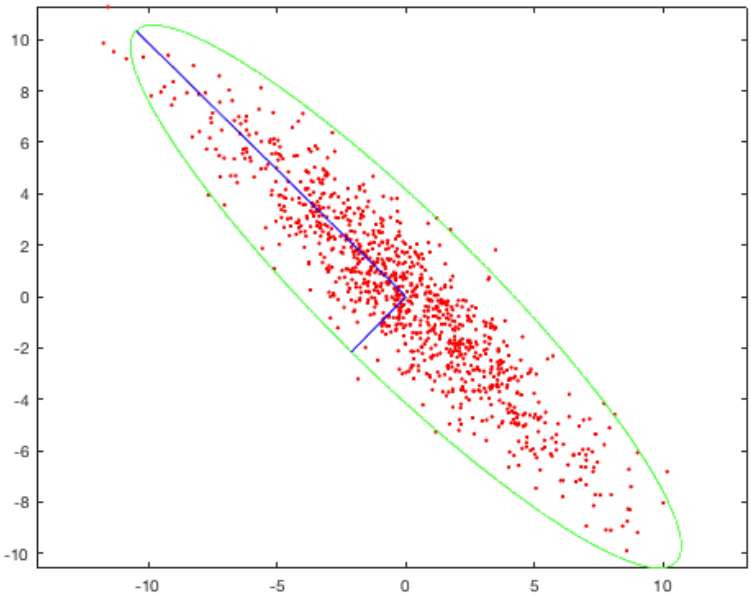
Primjer *Fisherfaces* Python koda (prikazanog, te prilagođenog na *https://answers.opencv.org/question/205605/*):

```
# potrebnih biblioteka
        import cv2
        import os
        import numpy as np

# trening podataka iz podmapa, funkcija za prepoznavanje lica uporabom OpenCV-a
        def detect_face(img):

# konverzija testne slike u grayscale, učitavanje OpenCV detektora (LBP)
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        face_cascade=cv2.CascadeClassifier('opencv-files/
        lbpcascade_frontalface.xml')
        faces =face_cascade.detectMultiScale(gray, scaleFactor=1.2, minNeighbors=5);

# uvijet ako se ne učita niti jedna slika tada se vraća originalna, korištenje samo
dijela slike koji sadržava lice
        if (len(faces) == 0):
        return None, None
         (x, y, w, h) = faces[0]
        returngray[y:y+w, x:x+h], faces[0]

# dohvaćanje mapa za svaki subjekt pojedinačno, slike i nazivi
        dirs = os.listdir(data_folder_path)
        faces = []
        labels = []

# prolazak kroz sve mape i učitavanje slika
        for dir_nameindirs:

# izrada putanje direktorija za subjekt
subject_dir_path = data_folder_path + "/" + dir_name
subject_images_names = os.listdir(subject_dir_path)
         for image_nameinsubject_images_names:
ifimage_name.startswith("."):
continue;

# izrada putanje za sliku
image_path = subject_dir_path + "/" + image_name

# učitavanje slika
image = cv2.imread(image_path)
```

```
# prikaz prozora za prikaz slike
            cv2.imshow("Trening slike...", cv2.resize(image, (400, 500)))
            cv2.waitKey(100)

# prepoznavanje lica
            face, rect = detect_face(image)

# postavljanje uvjeta ako lica nisu prepoznata
if face isnot None:
faces.append(face)
labels.append(label)
    cv2.destroyAllWindows()
    cv2.waitKey(1)
    cv2.destroyAllWindows()
returnfaces, labels
        print("Priprema...")
        faces, labels = prepare_training_data("training-data")
        print("Priprema završena")

# print ukupna lica i oznake
        print("Ukupno lica: ", len(faces))
        print("Ukupne oznake: ", len(labels))

# uporaba Fisherfacesrekognizera
        face_recognizer = cv2.face.FisherFaceRecognizer_create()

# treniranje
        face_recognizer.train(faces, np.array(labels))

#prepoznavanje – iscrtavanje pravokutnika na slici uporabom x,y koordinata definirane
dužine i širine
        def draw_rectangle(img, rect):
                (x, y, w, h) = rect
                cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)
#funkcija za ispis teksta

        def draw_text(img, text, x, y):
        cv2.putText(img, text, (x, y), cv2.FONT_HERSHEY_PLAIN, 1.5, (0, 255, 0), 2)

# funkcija kod prepoznavanja za iscrtavanje i označavanje
        def predict(test_img):
        img = test_img.copy()
        face, rect = detect_face(img)
                label, confidence = face_recognizer.predict(face)
                label_text = subjects[label]
        draw_rectangle(img, rect)
        draw_text(img, label_text, rect[0], rect[1]-5)
                returnimg
        print("Prepoznavanje slika...")

# učitavanje testnih slika, vršenje predviđanja
        test_img1 = cv2.imread("test-data/test1.jpg")
        test_img2 = cv2.imread("test-data/test2.jpg")
        predicted_img1 = predict(test_img1)
        predicted_img2 = predict(test_img2)
        print("Predviđanje završeno..")

# prikaz obje slike
        cv2.imshow(subjects[1], cv2.resize(predicted_img1, (400, 500)))
        cv2.imshow(subjects[2], cv2.resize(predicted_img2, (400, 500)))
        cv2.waitKey(0)
        cv2.destroyAllWindows()
        cv2.waitKey(1)
        cv2.destroyAllWindows()
```

6.2. Eigenfaces

Kod ove metode ulaz je slika, dok je izlaz ime osobe. Kod ulaza je bitno da slika ne bude prevelika. Što više slika imamo u bazi i što su ulazne slike veće to je sustav sporiji. Da bi isto radilo učinkovito, potrebno je smanjiti prevelike slike Slika dimenzije *m x n* je u stvari *m x n x 1* vektor. [6] Jedna tehnika za smanjenje dimenzija je analiza glavnih komponenata (*PCA*) koja se temelji na odabiru hiper-površine na koju se projiciraju sve točke koje su maksimalno rasipane, odnosno odabire se linija koja presjeca točke dijagonalno tj. os na kojoj su točke najviše rasipane.



Slika 6.2 Grafički prikaz hiperravnina-os
(izvor. https://pythonmachinelearning.pro/face-recognition-with-eigenfaces/)

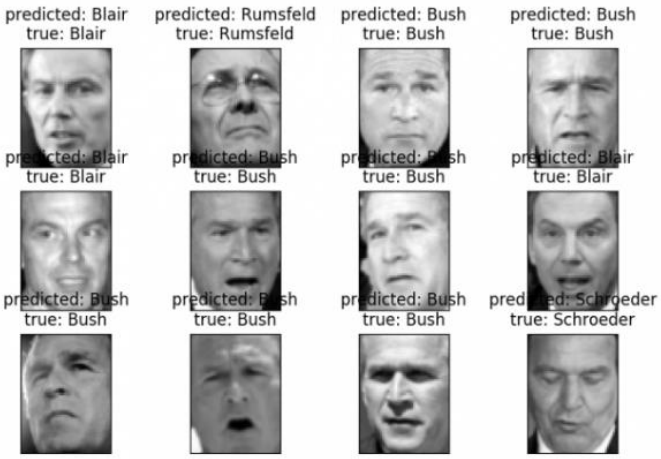
Iz grafa je vidljivo da je duga plava os ispravna jer obuhvaća sve točke koje su maksimalno raširene što omogućava klasifikatoru da zadrži najvažnije aspekte ulaznih podataka što u praksi znači da je lakše prepoznati lica kada su točke rasipane nego već skupljene *(Slika 6.2)*. Kod procesa prepoznavanja lica to se može zamisliti kao *m x n* sliku u *m x n* dimenzionalnom prostoru nakon čega se uporabom *PCA* smanjuje *m x n* prostor u manji prostor što rezultira brzinom provedbe i otpornost na šumove i promjene. Prilikom ulazne slike koristi se kaskadni klasifikator kako bi se izuzelo lice

od ostatka slike, s obzirom da lice na ulaznoj slici nije uvijek centrirano, nakon čega se na istoj koristi *eigenfaces* funkcija. Uporabom *PCA* se izrađuju *eigen*-vektori koji se mogu iskoristiti za vizualizaciju *(Slika 6.3).*

Primjer *Eigenfaces*Python koda (prikazanog, te prilagođenog na *https://pythonmachinelearning.pro/face-recognition-with-eigenfaces/*):

```
# importiranje potrebnih biblioteka
import matplotlib.pyplot as plt
fromsklearn.model_selection import train_test_split
fromsklearn.datasets import fetch_lfw_people
fromsklearn.metrics import classification_report
fromsklearn.decomposition import PCA
fromsklearn.neural_network import MLPClassifier

 # učitavanje podataka 100 slika
lfw_dataset=fetch_lfw_people(min_faces_per_person=100)

_, h, w =lfw_dataset.images.shape
X =lfw_dataset.data
y =lfw_dataset.target
target_names=lfw_dataset.target_names

# dioba na set podataka za trening i za testiranje
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.3)

# u ovomkorakuvrši se selekcija za osobekojenemajuminimalno 100 lica nakon čega
slijedi
  izračunavanje PCA
n_components= 100
pca=PCA(n_components=n_components, whiten=True).fit(X_train)

# primjena PCA transformacije
X_train_pca=pca.transform(X_train)
X_test_pca=pca.transform(X_test)

# u ovom koraku se primjenjuje transformacija kako bi se slike uvele u 100-
dimenzionalan  prostor  i to samo napodatke namijenjene za trening nakon čega
slijedi korak treniranja neuronske mreže

print("Prilagođavanje na trening set")
clf=MLPClassifier(hidden_layer_sizes=(1024,), batch_size=256, verbose=True,
early_stopping=True).fit(X_train_pca, y_train)

# u zadnjem koraku vršimo izradu izvješća o kvaliteti za svaku klasu gdje oznaka
točnosti nije pouzdana mjera, gdje oznaka podrške prikazuje koliko se puta pojavila
oznaka istine u testnom setu

y_pred=clf.predict(X_test_pca)
print(y_test, y_pred, target_names=target_names)

# nakon treninga klasifikatora slijedi klasifikacija slika - vizualizacija
def plot_gallery(images, titles, h, w, rows=3, cols=4):
    plt.figure()
    for i in range(rows * cols):
        plt.subplot(rows, cols, i+ 1)
        plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
        plt.title(titles[i])
        plt.xticks(())
        plt.yticks(())
```

```
def titles(y_pred, y_test, target_names):
    for i in range(y_pred.shape[0]):
        pred_name=target_names[y_pred[i]].split(' ')[-1]
        true_name=target_names[y_test[i]].split(' ')[-1]
        yield 'predicted: {0}\ntrue: {1}'.format(pred_name, true_name)

prediction_titles=list(titles(y_pred, y_test, target_names))
plot_gallery(X_test, prediction_titles, h, w)
```

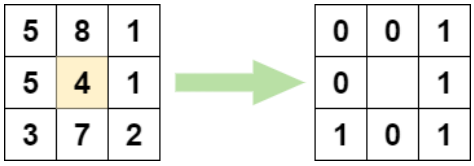


Slika 6.3 Slikovni prikaz Eigenfaces izlaza
(izvor. https://pythonmachinelearning.pro/face-recognition-with-eigenfaces/)

6.3. Histogrami lokalnih binarnih uzoraka (LBPH)

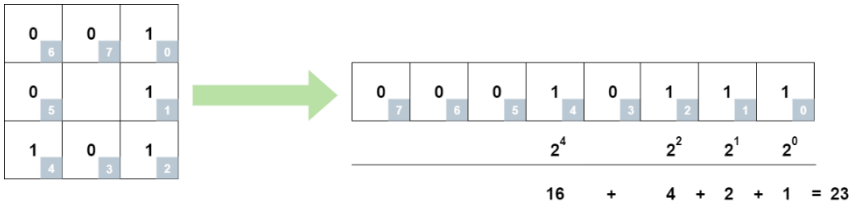
Jedna je od najstarijih metoda prepoznavanja koja je veoma jednostavna, ali i učinkovita, te koja označava susjedne piksele kao granične, a kao rezultat predstavlja binarni broj. [7] Proces se vrši na način da se ispitivani prozor podijeli u ćelije (primjerno 16 x 16 piksela za svaku ćeliju), a svaki piksel u ćeliji se uspoređuje sa njegovih 8 susjeda, slijedivši piksele u krug (u smjeru kazaljke na satu ili obrnuto). Ako je vrijednost središnjeg piksela veća od vrijednosti susjednog tada se dodaje vrijednost *0* , a u suprotnom slučaju vrijednost *1*. Dobiveni 8-znamenkasti broj se potom zbog praktičnosti pretvara u decimalni. Računanje histograma ćelije frekvencije svakog broja prikazano je kao 256-dimenzionalan vektor nakon čega slijedi optimizacija svih histograma, a potom i spajanje normaliziranih histograma svih ćelija što daje vektor značajki za cijeli prostor.

Primjer LBP deskriptora *(Slika 6.4)*za 3 x 3 piksela:



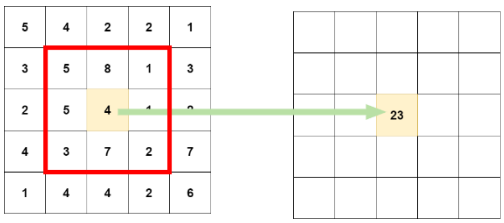
Slika 6.4 Izrada LBP-a uzimanjem susjednih 8 piksela oko centralnog piksela i pretvorba u 8 binarnih brojeva

nakon čega se binarni broj pretvara u decimalni *(Slika 6.5)*,



Slika 6.5 Kalkulacija 8 binarnih brojeva u decimalni

zatim se iz originalne slike sprema izračunata LBP vrijednost *(Slika 6.6)* u izlaznom redu sa istom visinom i širinom.



Slika 6.6 Izračunata LBP vrijednost se sprema u izlaznom redu sa istom visinom i širinom kao i originalna slika

Primjer koda za prepoznavanje tekstura koristeći *Python* i *OpenCV* (prikazanog, te prilagođenog na *https://iq.opengenus.org/lbph-algorithm-for-face-recognition/*).

```
# importiranje potrebnih biblioteka
import numpy as np
import cv2
import os

# izrada baze podataka sa imenima osoba koje želimo dodati
database = ["Osoba1", "Osoba2"]

# detektiranje lica uporabom Haarovih kaskada, obrezivanja slika i pretvorba lica u
grayscale
def face_detection(image):
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
haar_classifier=cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
face=haar_classifier.detectMultiScale(image_gray,scaleFactor=1.3,minNeighbors=5)
    (x,y,w,h) = face[0]
returnimage_gray[y:y+w, x:x+h], face[0]

# priprema podataka sa oznakama i licima
def prepare_data(data_path):
folders = os.listdir(data_path)
labels = []
faces = []
    for folder infolders:
label = int(folder)
training_images_path = data_path + '/' + folder
        for imageinos.listdir(training_images_path):
image_path = training_images_path + '/' + image
training_image = cv2.imread(image_path)
            face, bounding_box = face_detection(training_image)
faces.append(face)
labels.append(label)

    print ('Trainiranje zavrseno')
returnfaces, labels
faces, labels = prepare_data('training')
print ('Total faces = ', len(faces))
print ('Total labels = ', len(labels))

# izrada LBPH modela i treniranje istog uporabom pripremljenih podataka
model = cv2.face.createLBPHFaceRecognizer()
model.train(faces, np.array(labels))

# testiranje iztreniranog modela uporabom testne slike
def predict_image(test_image):
img = test_image.copy()
    face, bounding_box = face_detection(img)
label = model.predict(face)
label_text = database[label-1]
    print (label)
    print (label_text)
    (x,y,w,h) = bounding_box
    cv2.rectangle(img, (x,y), (x+w, y+h), (0,255,0), 2)
    cv2.putText(img, label_text, (x,y), cv2.FONT_HERSHEY_PLAIN, 1.5, (0, 255, 0), 2)
returnimg

test1 = cv2.imread("test/osoba1.jpg")
predict1 = predict_image(test1)
cv2.imshow('Prepoznavanje', predict1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

6.4. Metode temeljene na SIFT i SURF deskriptorima

SIFT (***S**cale-**I**nvariant **F**eature **T**ransform*) metoda nudi rješenje kada je u pitanju rotacija slike, skaliranje, intenzitet i promjena točke gledanja, te promjena osvjetljenja. [8]

SIFT algoritam ima 4 koraka:

- Detekcija skale ekstremnih razmjera
- Lokalizacija ključnih točaka
- Dodjela orijentacije
- Generiranje opisa

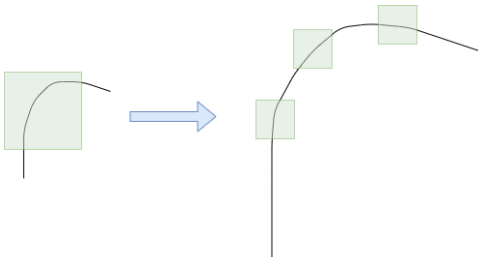
1. U prvom koraku se identificira lokacija i razmjer ključnih točaka uporabom skale ekstremnih razmjera u DoG funkciji korištenjem različitih vrijednosti σ gdje se DoG funkcija sastoji od slika Gaussovog zamućenja ulazne slike, različitim derivacijama tj, faktorom *k* prema jednadžbi *(Formula 6.11)*

$$D(x, y, \sigma) = L(x, y, k_i\sigma) - L(x, y, k_i\sigma) \tag{6.11}$$

gdje $L(x, y, k\sigma)$ predstavlja konvoluciju originalne slike $I(x,y)$ sa Gaussovim zamućenjem
$G(x, y, k\sigma)$ na skali $k\sigma$ iz čega proizlazi*(Formula 6.12)*:

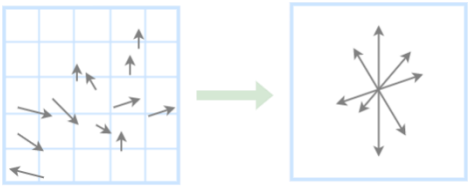
$$L(x, y, k\sigma) = G(x, y, k\sigma) * I(x, y) \tag{6.12}$$

2. U ovom koraku se lokaliziraju ključne točke *(Slika 6.7)*, te se one pročišćuju eliminacijom ključnih faktora gdje one odbacuju ključne točke niskog kontrasta.



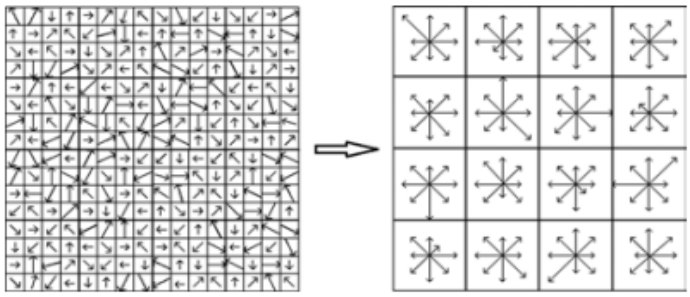
Slika 6-7 Lokalizacija ključnih točaka

3. U koraku dodjele orijentacije *(Slika 6.8)* dobiva se orijentacija ključne točke na gradijentu slike.



Slika 6.8 Orijentacija ključne točke

4. Ovdje se izračunava lokalni deskriptor slike za svaku ključnu točku na temelju veličine i orijentacije gradijenta slike u svakoj točki uzorka slike u području centriranog na ključnu točku koje izrađuju 3D histogram lokacije i orijentacije gradijenta s 4x4 lokacijskom mrežom *(Slika 6.9)* i 8-orijentacijskom ćelijom za svaki uzorak što je dimenzija deskriptora ključne točke sa 128 elemenata.



Slika 6.9 Primjer 4x4 lokacijske mreže
(izvor. https://www.researchgate.net/figure/The-process-of-building-a-single-SIFT-keypoint-descriptor-a-A-Single-SIFT keypoint_fig6_237090165)

SIFT primjer u OpenCV-u uz pomoć Python-a

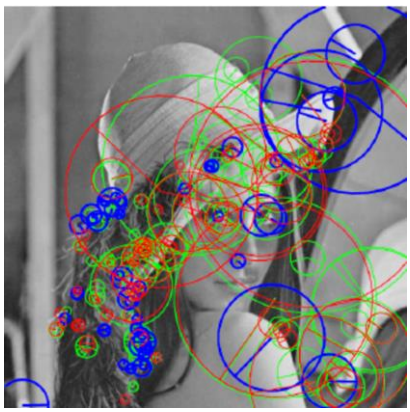
Prvo slijedi konsttrukcija SIFT objekta (primjer koda dostupan na *https://opencv-pythontutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html*)

```
import cv2
import numpy as np
img = cv2.imread('test.jpg')
gray= cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
sift = cv2.SIFT()
kp = sift.detect(gray,None)
img=cv2.drawKeypoints(gray,kp)
cv2.imwrite(test_tocke.jpg',img)
```

sift.detect() funkcija pronalazi ključnu točku u slici. Svaka ključna točka posebna je struktura koja ima više atributa poput *(x,y)* koordinata, veličine smislenog susjedstva, kuta koji određuje orijentaciju, odaziv koji određuje važnost ključne točke.

```
img=cv2.drawKeypoints(gray,kp,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv2.imwrite(test_tocke.jpg',img)
```

cv2.drawKeyPoints() funkcija crta male krugove na pozicijama ključnih točaka. *cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINT* funkcija na koju ako se proslijedi zastavica iscrtava krug s veličinom ključne točke, a ujedno će pokazati i orijentaciju.



Slika 6.10 Grafički prikaz SIFT primjera (izvor. https://www.researchgate.net/figure)

Slijedi izračunavanje deskriptora kod kojih OpenCV nudi dvije metode:

Ako su pronađene ključne točke tada se poziva *sift.compute()* funkcija koja izračunava deskriptore ključnih točaka koje su pronađene

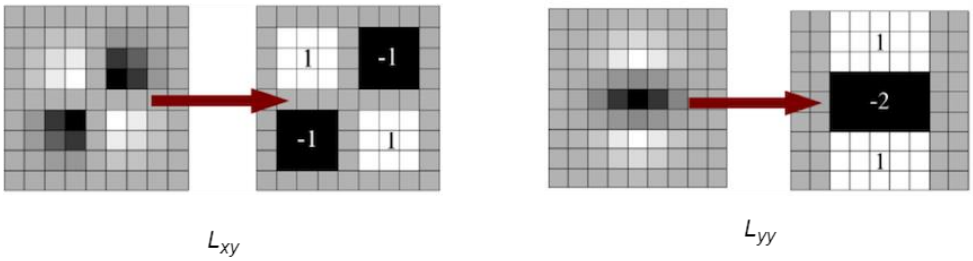
Ako nisu pronađene ključne točke, tada se iste pronalaze izravno zajedno s deskriptorima uz pomoć funkcije *sift.detectAndCompute()*.

SURF

Algoritam **SURF (**Speed Up Robust Features**)** koji se temelji na više-skalnom prostoru, a detektor značajki temelji se na Hessian-ovoj matrici koja ima odlične performanse i preciznost. [9] U slici *I* je određena točka *x=(x,y)*, a Hessian matrica *H(x,σ)* u točki*x* na skali *σ* se može definirati kao*(Formula 6.13)*:

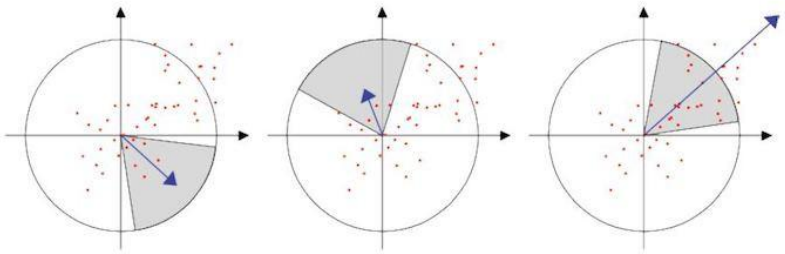
$$H(x,\sigma) = \begin{bmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{yx}(x,\sigma) & L_{yy}(x,\sigma) \end{bmatrix} \tag{6.13}$$

Vezano na SIFT, SURF metoda aproksimira *LoG* s *BOX* filtrom uporabom integralnih slika *(Slika 6.11)*, a može raditi i paralelno za različita skaliranja. SURF se oslanja na determinantu Hessian-ove matrice za skaliranje i poziciju.



Slika 6.12 Gaussova parcijalna derivacija

(izvor.https://dsp.stackexchange.com/questions/30262/determinant-of-hessian-approximation-surf)



Slika 6.11 Grafički prikaz dominantne orijentacije

(izvor. https://ieeexplore.ieee.org/document/5609874)

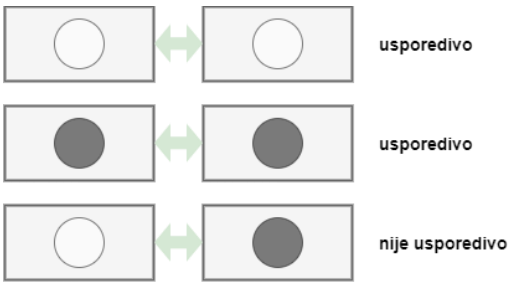
Za dodjelu orijentacije SURF metoda *(Slika 6.12)* koristi wavelet filtre u vodoravnom i okomitom smjeru težinski normirane Gaussovom funkcijom, te se zatim iscrtavaju u prostoru.

Dominantna orijentacija se procjenjuje izračunavanjem zbroja svih odgovora unutar kliznog orijentacijskog prozora pod kutom od 60˚. Za većinu aplikacija nije potrebno pronaći orijentaciju, što dodatno ubrzava proces, a takva funkcija se naziva i *U-SURF (uspravni surf).* OpenCV podržava obje ovisno o zastavici, pa tako ako je vrijednost *0* tada se izračunava orijentacija, a ako je vrijednost *1* tadase orijentacija ne izračunava i proces je samim time brži. Kod ključne točke se uzima okolina veličine 20s x 20s gdje *s* označava veličinu, nakon čega se dijeli u 4x4 podregije. Za svaku podregiju uzimaju se vodoravni i okomiti odzivi Wavelet-a nakon čega se formira vektor prema

$$v = \left(\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|\right) \tag{6.14}$$

Ovo je primjer SURF deskriptora sa ukupno 64 dimenzije, dok se za veću prepoznatljivost koristi SURF deskriptor proširene verzije od 128 dimenzija. Zbroj d_x i $|d_x|$ se izračunava odvojeno za $d_y < 0$ i $d_y \geq 0$ prema tome zbrojevi d_y *i* $|d_y|$ se dijele prema d_xčime se duplicira broj značajki *(Formula 6.14).*

Drugo važno poboljšanje je uporaba oznake *Laplace* za temeljnu interesnu točku koja izdvaja svijetle mrlje na tamnoj pozadini od obrnute situacije. U fazi prepoznavanja uspoređuju se značajke samo ako imaju isti tip kontrasta *(Slika 6.13).* Takav minimalni podatak omogućuje brže podudaranje bez smanjenja performansi deskriptora.



Slika 6.13 Primjer Laplace

SURF metoda je otprilike 3 puta brža od SIFT metode dok su performanse gotovo jednake. SURF metoda je dobra kod slika sa zamućenjem i rotacijom, ali manje dobra ukoliko se radi o promjeni osvjetljenja i točke gledišta.

SURF primjer u OpenCV-u uz pomoć Python-a

Isto kao i u SIFT metodi moguće je koristiti funkcije *SURF.detect()*; *SURF.sompute()* za pronalaženje ključnih točaka i deskriptora.

U nastavku slijedi demo primjer koji pokazuje način na koji se pronalaze SURF ključne točke i deskriptori, te kako ih nacrtati. Za primjer se koristi Python terminal, isto kao i za SIFT metodu (primjer koda dostupan na *https://opencv-python tutroals.readthedocs.io/en/latest/ py_tutorials/py_feature2d/py_surf_introl*)

```
img = cv2.imread('test.png',0)
# izrada SURF objekta
surf = cv2.SURF(400)
# pronalazak ključnih točki i deskriptora
kp, des = surf.detectAndCompute(img,None)
len(kp)
699
```

Kod demo slike je prikazano 1199 ključnih točaka, što je previše. Njih je potrebno smanjiti na 50 koje je zatim potrebno nacrtati na slici *(Slika 6.14)*upotrebom funkcije

```
# provjera Hessa
print surf.hessianThreshold
400.0
surf.hessianThreshold = 50000
# ponovno izračunavanje ključnih točaka
kp, des = surf.detectAndCompute(img,None)
print len(kp)
47
  img2=cv2.drawKeypoints(img,kp,None,(255,0,0),4)
plt.imshow(img2),plt.show()
```

Rezultat vidljiv u nastavku

Slika 6.14 Ključne točke

(izvor. izvor. https://opencv-python tutroals.readthedocs.io/en/latest/ py_tutorials/py_feature2d/py_surf_introl)

Slijedi primjena U-SURF*(Slika 6.15)*, bez traženja orijentacije

```
# Provjera i namještanje orijentacije na uspravni položaj
print surf.upright
False
surf.upright = True
# ponovno računanje i crtanje
kp = surf.detect(img,None)
img2 = cv2.drawKeypoints(img,kp,None,(255,0,0),4)
plt.imshow(img2),plt.show()
```

U nastavku je vidljivo da su sve orijentacije sada usmjerene uspravno

Slika 6.15 Uspravna orijentacija ključnih točaka

(izvor.https://opencv.ythontutroals.readthedocs.io/en/latest/pytutorials/p_feature2d)

Na kraju se provjerava veličina deskriptora koja se mijenja u 128 ako je 64 dimenzionalan.

```
# pronalaženje veličine deskriptora i promjena veličine
print surf.descriptorSize()
64
surf.extended
False
surf.extended = True
kp, des = surf.detectAndCompute(img,None)
print surf.descriptorSize()
128
print des.shape
(47, 128)
```

6.5. Metode dubokog učenja

Kod metode dubokog učenja možemo spomenuti FaceNet metodu koja je u biti neuronska mreža koja uči mapiranje slika lica u kompaktni prostor gdje udaljenosti predstavljaju mjeru sličnosti lica što u praksi znači da što su lica sličnija to je manja udaljenost između njih. Za izračunavanje gubitka koristi se tzv. metoda trostrukog gubitka koja minimizira udaljenost između ulazne slike i slika koje sadrže isti identitet dok s druge strane maksimizira udaljenost između ulazne slike i slika koje sadrže različiti identitet. FaceNet se može nazvati i sijamskom mrežom jer predstavlja neuronsku mrežnu arhitekturu koja uči razlikovati dva ulaza što im omogućava da nauče koje su slike slične, a koje nisu što može poslužiti i za prepoznavanje lica. Sijamske mreže sastoje se od dvije identične neuronske mreže svaka sa jednakim značajkama. Prvo svaka mreža uzme jednu ili dvije ulazne slike kao ulaz, a izlazi zadnjih slojeva svake mreže se potom šalju u funkciju koja određuje da li slike sadrže isti identitet, a isto se dobiva računanjem udaljenosti između dva izlaza. [10] Za primjer će se koristiti *Keras*, *Tensorflow* i *Python* programski jezik. Za provedbu su prvo potrebne dvije javno dostupne datoteke *fr_utils.py* koja sadrži funkcije za proslijeđivanje slika u mrežu i za dobivanje kodiranja slike, te datoteku naziva *inception_blocks_v2.py* koja sadrži funkcije za pripremu i sastavljanje FaceNet mreže. Za početak je potrebno sastaviti FaceNet mrežu koja će prepoznavati lica uz pomoć koda (prikazanog, te prilagođenog na: https://missinglink.ai/guides/tensorflow/tensorflow-face-recognition-three-quick-tutorials/):

```
import os
import glob
import numpy as np
import cv2
import tensorflow as tf
fromfr_utils import *
from inception_blocks_v2 import *
fromkeras import backend as KK.set_image_data_format('channels_first')FRmodel =
faceRecoModel(input_shape=(3, 96, 96))def triplet_loss(y_true, y_pred, alpha = 0.3):
anchor, positive, negative = y_pred[0], y_pred[1], y_pred[2]
pos_dist = tf.reduce_sum(tf.square(tf.subtract(anchor,
positive)), axis=-1)
neg_dist = tf.reduce_sum(tf.square(tf.subtract(anchor,
              negative)), axis=-1)
basic_loss = tf.add(tf.subtract(pos_dist, neg_dist), alpha)
loss = tf.reduce_sum(tf.maximum(basic_loss, 0.0))
returnlossFRmodel.compile(optimizer = 'adam', loss = triplet_loss, metrics =
['accuracy'])
load_weights_from_FaceNet(FRmodel)
```

Zatim slijedi inicijalizacija mreže ulaznim oblikom (3,96,96) što znači da su *RGB* kanali prva dimenzija slike koja se dostavlja mreži, te da su sve slike koje se dostavljaju mreži veličine 96x96 piksela.

Nastavno slijedi korak definiranja funkcije trostrukog gubitka oslanjajući se na formulu*6.15*

$$Loss = \sum_{i=1}^{N} \left[||f_i^a - f_i^p||_2^2 - ||f_i^a - f_i^n||_2^2 + \alpha \right] \quad (6.15)$$

gdje se *f(a)* odnosi na izlazno kodiranusliku neke osobe, $f_i(p)$odnosi na izlazno kodiranusliku iste osobe, $f_i(n)$ odnosi na izlazno kodiranu sliku različite osobe, dok α predstavlja konstantu da mreža ne optimizira prema *f(a) – f(p) = f(a) – f(n) = 0* .

Nakon što se dobije funkcija gubitka može se postaviti model prepoznavanja lica uz pomoć *Keras*-a, a ujedno se koristi *Adam*-ov alat za optimizaciju kako bi se dodatno smanjio gubitak. Slijedi priprema baze podataka osoba za koje želimo da ih sustav prepozna. Koriste se sve slike u direktoriju slika za svakog pojedinca (po jedna slika za svakog pojedinca).

```
def prepare_database():
database = {}    for file inglob.glob("images/*"):
identity = os.path.splitext(os.path.basename(file))[0]
database[identity] = img_path_to_encoding(file, FRmodel)
returndatabase
```

Podatke svake slike potrebno je pretvoriti u 128-brojčane kodove uz pomoć funkcije *img_path_to_encoding*. Ova funkcija uzima putanju slike i dostavlja je u mrežu za prepoznavanje lica, a zatim vraća izlaz iz mreže što predstavlja kodiranu sliku. Kada je u bazu podataka dodano kodiranje, za svaku sliku slijedi prepoznavanje.

```
def who_is_it(image, database, model):
encoding = img_to_encoding(image, model)
min_dist = 100
identity = None
    for (name, db_enc) indatabase.items():
dist = np.linalg.norm(db_enc - encoding)
print('distance for %s is %s' %(name, dist))
ifdist<min_dist:
min_dist = dist
identity = name
ifmin_dist> 0.52:
return None
else:
returnidentity
```

Prikazani kod dostavlja novu sliku u funkciju *img_to_encoding* koja obrađuje sliku uz pomoć FaceNet-a i vraća kodiranu sliku. Uz pomoć kodiranih slika se može pronaći individuu kojoj slika najvjerojatnije i pripada i to na način da se izračunaju udaljenosti između nove slike i svakog pojedinca u bazi podataka. Osoba koja ima najmanju udaljenost bira se kao najvjerojatniji rezultat. Da bi se isto potvrdilo služi

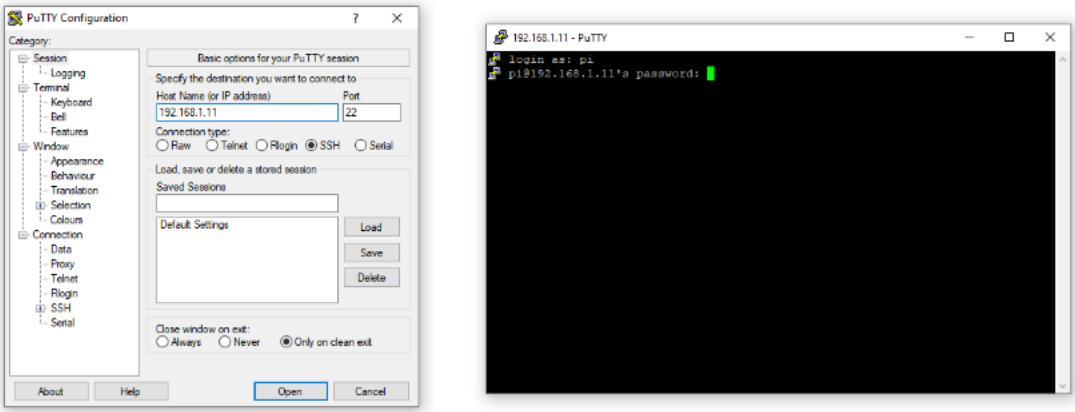
```
ifmin_dist> 0.52:
return None
else:
    return identity
```

iz čega je vidljivo da ako je navedena udaljnost veća od 0.52 tada je utvrđeno da osoba na novoj slici ne postoji u bazi podataka, a ako je navedena udaljenost jednaka ili manja od 0.52 tada je utvrđeno da su osobe iste. Za što bolje rezultate za različite skupove podataka u bazi podataka je potrebno navedenu vrijednost podesiti na manju ili veću.

7. Praktični dio

U ovome dijelu ćemo prikazati korak-po-korak cijeli postupak pripreme našeg uređaja sa svim pripadajućim elementima prikazanim na početku ovog rada. Da bi sve ispravno radilo, potrebno je instalirati odgovarajuću programsku podršku počevši od instalacije odgovarajućeg operacijskog sustava (OS-a). Javno je dostupno sve više operacijskih sustava namijenjenih za Raspberry uređaje, a neki od njih su: *DietOS (DebianOS), LIBREELEC (JEOS), OSMC (KodiOS), Windows IoT, RetroPie(DebianOS), KaliLinux, Ubuntu Core* i dr.Za ovaj projekt koristit ćemo službeni OS, zadnju verziju naziva *Raspbian Buster* koji je besplatno dostupan za preuzimanje na službenim web stranicama adrese *www.raspberrypi.org*.

Nakon što preuzmemo *.iso* datoteku sa *Raspbian Buster* OS-om, uz pomoć besplatnog alata *Win32DiskImager*-a učitavamo i postavljamo OS na microSD karticu. S obzirom da za upravljanje uređajem nećemo koristiti direktni način povezivanja putem monitora i pripadajuće periferije (tipkovnica + miš) nego ćemo isto raditi putem bežićnog povezivanja uporabom *VNC*-a (*Virtual Network Computing*) i *SSH* (*Secure Shell*) za osnovnu konfiguraciju uređaja, tom prilikom je potrebno na *boot* particiji microSD kartice kreirati novu datoteku naziva *SSH*, bez ekstenzije i sadržaja koja će nam omogućiti udaljeno povezivanje Raspberry uređaja sa osobnim računalom, a u svrhu dijeljenja zaslona. Nakon završetka ovog koraka potrebno je umetnuti microSD karticu u Raspberry uređaj, te potom istog priključiti na odgovarajući izvor napajanja radi pokretanja.

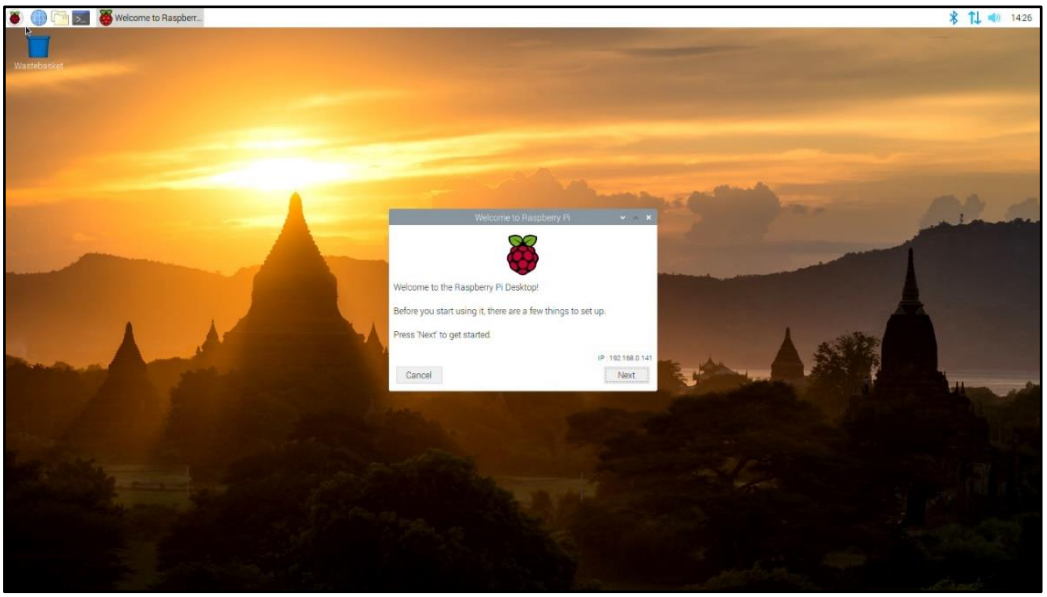


Slika 7.1 SSH povezivanje sa Raspberry uređajem

Na osobnom računalu potrebno je izvršiti skeniranje LAN mreže kako bi saznali IP adresu našeg uređaja potrebnu za osnovnu konfiguraciju. Istu potom upisujemo u *PUTTY* aplikaciju instaliranu na našem osobnom računalu i kliknemo na „*Open*“ kako bi ostvarili vezu sa Raspberry uređajem, nakon čega je potrebno upisati odgovarajuće korisničko ime i lozinku koji su po osnovnim postavkama *pi* i *raspberry*.

Nakon uspješnog povezivanja potrebno je izvršiti konfiguraciju Raspberry uređaja na način da se u novootvoreni prozor terminala upiše naredba *sudo raspi-config* , a zatim odabirom opcije *5. Interfacing Options* uključujemo opciju*VNC* radi grafičkog pristupa uređaju, odnosno dijeljenjem radne površine. Podešavanje *VNC*-a za povezivanje vršimo na način da se na osobnom računalu pokrene „*VNCViewer*“ u kojeg upisujemo iste pristupne podatke kao i za *SSH* povezivanje, IP adresu a potom korisničko ime i lozinku (*pi* i *raspberry*) nakon čega dobivamo dijeljeni ekran/radnu površinu, te je samo još potrebno podesiti vrijeme i lokacijske poistavke u novootvorenom prozoru.

Nakon uspješne konfiguracije uređaja slijedi instalacija sve potrebne programske podrške poput OpenCV-a i ostalih paketa.



Slika 7.2 VNC - Desktop RaspberryPi uređaja

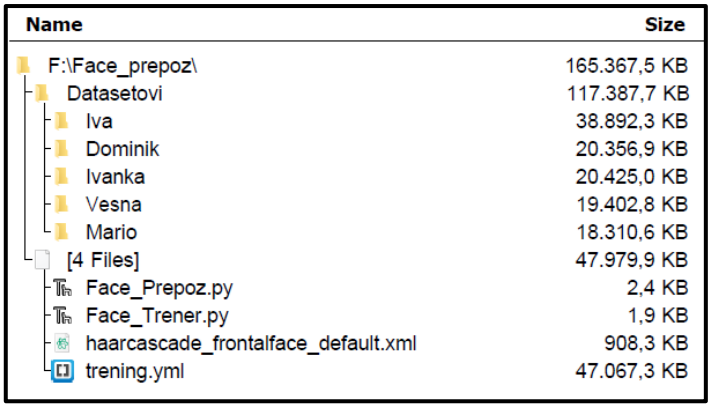
Otvaramo prozor LX terminala u koji redom upisujemo potrebne naredbe počevši od:

```
# Svaka linija je naredba za sebe koju je nakon upisa potrebno izvršiti

# prvo je potrebno provjeriti nove verzije svih paketa i ako ih ima instalirati
iste kao bi OS bio ažuran
$ sudoapt-getupdate&&sudoapt-getupgrade

# slijedi instalacija razvojnih alata poput CMake-a koji služi za konfiguraciju
procesa izgradnje OpenCV-a
$ sudoapt-getinstallbuild-essentialcmakepkg-config

# slijedi instalacija paketa za rukovanje različitim slikovnim formatima poput PNG,
JPEG, TIFF i dr
$ sudoapt-getinstalllibjpeg-dev libtiff5-dev libjasper-devlibpng-dev

# slijedi instalacija paketa za rukovanje različitim video formatima
$ sudoapt-getinstalllibavcodec-devlibavformat-devlibswscale-dev libv4l-dev
$ sudoapt-getinstalllibxvidcore-dev libx264-dev

# OpenCV biblioteka dolazi s pod-modulom naziva highgui koji je namijenjen za
prikazivanje slika na zaslonu kao i izgradnju osnovnih GUI-a pa je za isto potrebno
instalirati GTK biblioteku sa uvjetima
$ sudoapt-getinstall libfontconfig1-dev libcairo2-dev
$ sudoapt-getinstall libgdk-pixbuf2.0-dev libpango1.0-dev
$ sudoapt-getinstall libgtk2.0-dev libgtk-3-dev

# instalacija uvjeta za optimizaciju operacija unutar OpenCV-a (operacije matrica i
sl.)
$ sudoapt-getinstalllibatlas-base-devgfortran

# instalacija ovisnosti za optimizaciju operacija unutar OpenCV-a
$ sudoapt-getinstall libhdf5-dev libhdf5-serial-dev libhdf5-103
$ sudoapt-getinstall libqtgui4 libqtwebkit4 libqt4-test python3-pyqt5

# instalacija Python-a 3 za izgradnju OpenCV-a sa Python poveznicama
$ sudoapt-getinstall python3-dev

# instalacija opencv i opencv_contrib-a
$ cd ~
$ wget -O opencv.zip https://github.com/opencv/opencv/archive/4.0.0.zip
$ wget -O opencv_contrib.zip
https://github.com/opencv/opencv_contrib/archive/4.0.0.zip

#raspakiravanje arhiva
$ unzip opencv.zip
$ unzip opencv_contrib.zip

#preimenovanje direktorija
$ mv opencv-4.0.0 opencv
$ mv opencv_contrib-4.0.0 opencv_contrib

# instalacija Python paketa i preduvjeta za OpenCV je NumPy
$ pipinstallnumpy

# kreiranje build direktorija
$ cd ~/opencv
$ mkdirbuild
$ cd build

# pokretanje CMake za konfiguriranje OpenCV-a
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
```

```
    -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
    -D ENABLE_NEON=ON \
    -D ENABLE_VFPV3=ON \
    -D BUILD_TESTS=OFF \
    -D OPENCV_ENABLE_NONFREE=ON \
    -D INSTALL_PYTHON_EXAMPLES=OFF \
    -D BUILD_EXAMPLES=OFF ..

# prije izgradnje poželjno je povećati swap prostor kako bi se taj proces što više
ubrzao
$ sudonano /etc/dphys-swapfile

# promijeniti vrijednost na
CONF_SWAPSIZE=2048

# restartati swap servis
$ sudo /etc/init.d/dphys-swapfile stop
$ sudo /etc/init.d/dphys-swapfile start

# slijedi izrada OpenCV-a 4 uporabom svih 4 jezgri procesora
$ make -j4

# ako je izrada protekla bez greški tada slijedi instalacija
$ sudo make install
$ sudoldconfig

# nakon završenog procesa potrebno je vratiti swap vrijednost na osnovnu i
restartati servis
CONF_SWAPSIZE=100
$ sudo /etc/init.d/dphys-swapfile stop
$ sudo /etc/init.d/dphys-swapfile start

# slijedi testiranje
$ python3
>>>import cv2
>>>cv2.__version__
'4.1.0'
>>>exit()
```

Navedeni postupak isproban je na dva različita uređaja radi usporedbe. Izrada OpenCV zahtjeva mnogo vremena ovisno o jačini uređaja, pa tako je na starijem/slabijem uređaju Raspberry Pi 2B izrada OpenCV-a trajala otprilike 16 sati dok je za isti proces na novom uređaju Raspberry Pi 4 trajala svega 2 sata. Verzija Python2 od 01.01.2020. više nema podršku pa se tako treba koristiti nova verzija Pythona – Python3. Nakon uspješne provedbe svih prethodnih koraka, naš uređaj je spreman za izradu odgovarajućeg programa za prepoznavanje i usporedbu lica.

Glavne značajke ovog programa smještene su u dvije datoteke, jedna namijenjena za testiranje, a druga za prepoznavanje. Datoteka za testiranje uzima fotografije smještene u mapama u direktoriju *Datasetovi*, indeksira imena mapi (nazivi osoba), te od istih kreira datoteku naziva *trening.yml* sa svim potrebnim vrijednostima koje će poslužiti za usporedbu putem programa za prepoznavanje lica, dok datoteka naziva *haarcascade_frontalface_default.xml* sadrži kaskadne klasifikatore povezane sa obje datoteke.



Slika 7.3 Datotečni hijerarhijski prikaz

U nastavku je prikaz Python programskog koda za program namijenjenog za trening – *Face_Trener.py*

```
#Program za treniranje slika iz direktorija Face_Images i kreiranje trening.yml datoteke
#importiranje potrebnih biblioteka za obradu slika, za konverziju slika u numerički niz, za baratanje mapama
import cv2
import numpy as np
import os
from PIL import Image
# uporaba OpenCV LBPH Recognizera
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
recognizer = cv2.face.LBPHFaceRecognizer_create()

# definicija uvjeta, spremišta slika, navigiranje do spremišta slika, provjera svih direktorija u mapi
Lice_ID = 1
posoba = ""
y_ID = []
x_train = []
```

```
Datasetovi = os.path.join(os.getcwd(), "Datasetovi")
print (Datasetovi)
for root, dirs, files in os.walk(Datasetovi):
        for file in files:

# definicija tipova slika/ekstenzija jpeg, jpg, png
        iffile.endswith("jpeg")orfile.endswith("jpg")orfile.endswith("png"):
                        path = os.path.join(root, file)
                        ime_osobe = os.path.basename(root)
                        print(path, ime_osobe)

# provjera ako je došlo do promjene osobe, ako da povećaj
                        if posoba!=ime_osobe:
                                Lice_ID=Lice_ID+1
                                posoba = ime_osobe

# pretvorba slike u grayscale pomoću Pillow-a i rezanje slike na 550*500 px
                        Gery_Image = Image.open(path).convert("L")
                        Crop_Image = Gery_Image.resize( (550,550) , Image.ANTIALIAS)
                        Final_Image = np.array(Crop_Image, "uint8")
# detekcija lica u svim uzorcima
                        faces = face_cascade.detectMultiScale(Final_Image,
scaleFactor=1.2, minNeighbors=5)
                        print (Lice_ID,faces)

#rezanje slike prema području interesa)
                        for (x,y,w,h) in faces:
                                roi = Final_Image[y:y+h, x:x+w]
                                x_train.append(roi)
                                y_ID.append(Lice_ID)

#izrada matrice iz podataka za treniranje
recognizer.train(x_train, np.array(y_ID))

# spremanje u .yml datoteku
recognizer.save("trening.yml")
```

Nakon što se navedeni program pokrene isti kreira datoteku oznake *trening.yml* iz slika spremljenih u direktoriju *Datasetovi* na temelju koje će se kasnije uspoređivati podaci sa videa web kamere.

U nastavku je prikaz Python programskog koda za program namijenjenog za prepoznavanje – *Face_Prepoz.py*

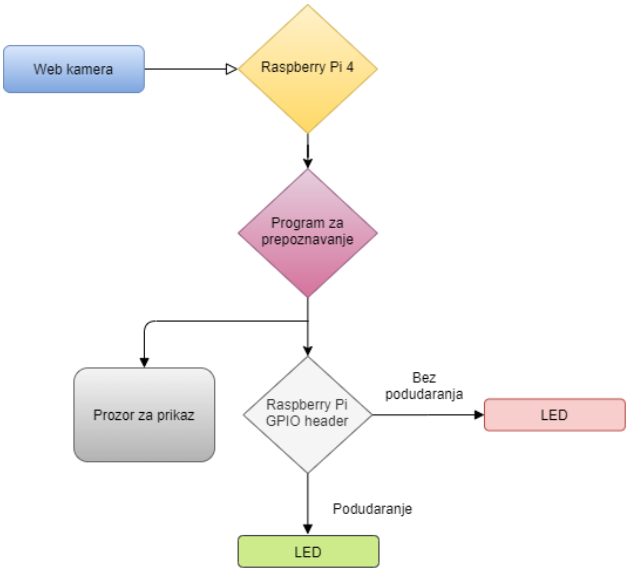
```
#Program za prepoznavanje lica i osobe usporedbom podataka iz datoteke trening.yml

#importiranje potrebnih biblioteka za obradu slika, za konverziju slika u numerički
niz, za rukovanje mapama, za rukovanje GPIO sučeljem - vrsta i oznaka
import cv2
import numpy as np
import os
from PIL import Image
import RPi.GPIO as GPIO
from time
import sleep
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(21,GPIO.OUT)
GPIO.setup(23,GPIO.OUT)



# uporaba OpenCV LBPH Recognizera, učitavanje trening.yml datoteke
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read("trening.yml")
id = 0
labels = ['Ivanka','Vesna','Iva','Mario','Dominik']

# import videa iz Web kamere, definiranje omjera slike
cam = cv2.VideoCapture(0)
cam.set(3, 640)
cam.set(4, 480)

# Definiranje minimalne veličine prozora za prepoznavanje lica
minW = 0.1*cam.get(3)
minH = 0.1*cam.get(4)

# definiranje uvjeta ako je točno
while True:
    ret, img =cam.read()#učitavanje frameova
    img = cv2.flip(img,1) #okretanje kamere
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)



#podešavanje skaliranja, susjeda TP i FP
```

```
    faces = faceCascade.detectMultiScale(gray,scaleFactor =1.2,minNeighbors=5,
minSize=(int(minW),int(minH)),)
    for(x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
        id, confidence = recognizer.predict(gray[y:y+h,x:x+w])

# ispisivanje naziva/imena prema confidence vrijednostima, za max vrijednost izlaz
na zeleni LED preko GPIO headera
        if (confidence < 100):
            id = labels[id]
            GPIO.output(21,1)
            GPIO.output(21,0)
            time.sleep(0.75)
            confidence = "{0}%".format(round(100 - confidence))
# ispisivanje nepoznato ako nije prepoznato, ako nije prepoznato izlaz na crveni LED
preko GPIO headera
        else:
            id = "nepoznato"
            GPIO.output(21,0)
            GPIO.output(23,1)
            confidence = "{0}%".format(round(100 - confidence))

        cv2.putText(img,str(id),(x+10,y-10),font,1,(255,255,255),2)
        cv2.putText(img,str(confidence),(x+10,y+h-10),font,1,(255,255,0),1)

    cv2.imshow('Web - kamera',img)
    if cv2.waitKey(10) & 0xff==ord('q'): # Tipka'q' za izlaz
        break
# resetiranje
print("Izlaz i reset")
cam.release()
cv2.destroyAllWindows()
GPIO.cleanup()
```

Nakon pokretanja programa za trening pomoću kojeg su kreirane i učitane vrijednosti od svih slika iz pripadajućih mapa, te iste uspoređene sa video prikazom dostavljenog sa web kamere, prepoznatoje lice osobe u kadru i ako se ono podudara sa onim u bazi podataka tada se na okviru za prepoznavanje ispisuje i ime.

Slika 7.4 Grafički prikaz programa za prepoznavanje sa rezultatom

Ako program prepozna osobu putem web kamere odnosno ako se osoba podudara sa osobom iz baze, tada se ispisuje naziv osobe u prozoru za prikaz i ujedno šalje signal na izlaz Raspberry Pi uređaja u obliku gašenja crvenog svjetla na LED modulu i paljenja zelenog u ciklusnom trajanju od 0.75 sekunde. Sve dok nema podudaranja svijetli crveno svijetlo, a u slučaju podudaranja gasi se crveno i pali zeleno svjetlo.



Slika 7.5 Dijagram tijeka procesa prepoznavanja

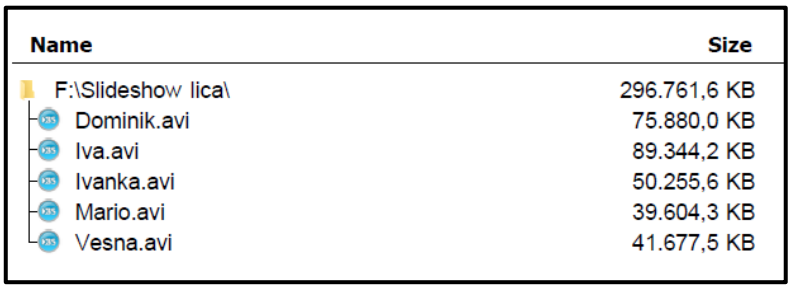
Za ovaj rad izrađeni su datasetovi fotografija za 5 različitih osoba. uzevši u obzir karakteristike Raspberry Pi 4 uređaja, izrađeni su datasetovi od 100 fotografija (dimenzija 640x480 piksela sa *.png* ekstenzijom i u realnim/dnevnim uvjetima) za svaku osobu odnosno ukupno 500 fotografija sveukupno iz čega je izdvojeno po 50 fotografija za svaku osobu na kojoj su prepoznate Haarove kaskade sukladno postavkama za treniranje testnog programa. Iz pripremljenih datasetova izvršen je trening pokretanjem *Face_Trener.py* datoteke kako bi istrenirali naš program sa kreiranim datasetovima. Sljedeći korak bio je kopiranje cijelog dataseta fotografija (ukupno 250 fotografija) iz čega je izrađen *slideshow* (video datoteka *.avi* ekstenzije) za svaku od 5 osoba sa postavkama trajanja svake fotografije od 1 sekundu i sa tranzicijom izmjene fotografija u pojedinačnom trajanju od pola sekunde. Zasebne video datoteke tako neće previše opteretiti uređaj. Svaka od pripadajućih datoteka se potom zasebno učita u program za prepoznavanje na način da se dio koda programa za prepoznavanje *Face_Prepoz.py* promjenio iz

```
cam = cv2.VideoCapture(0)
```

gdje vrijednost *0* predstavlja ulaz kamere (ako bi bilo više od jedne kamere tada bi vrijednosti bile nastavne 1, 2, 3..), a kako bi ulaz promijenili na video datoteku potrebno je izmjeniti prema

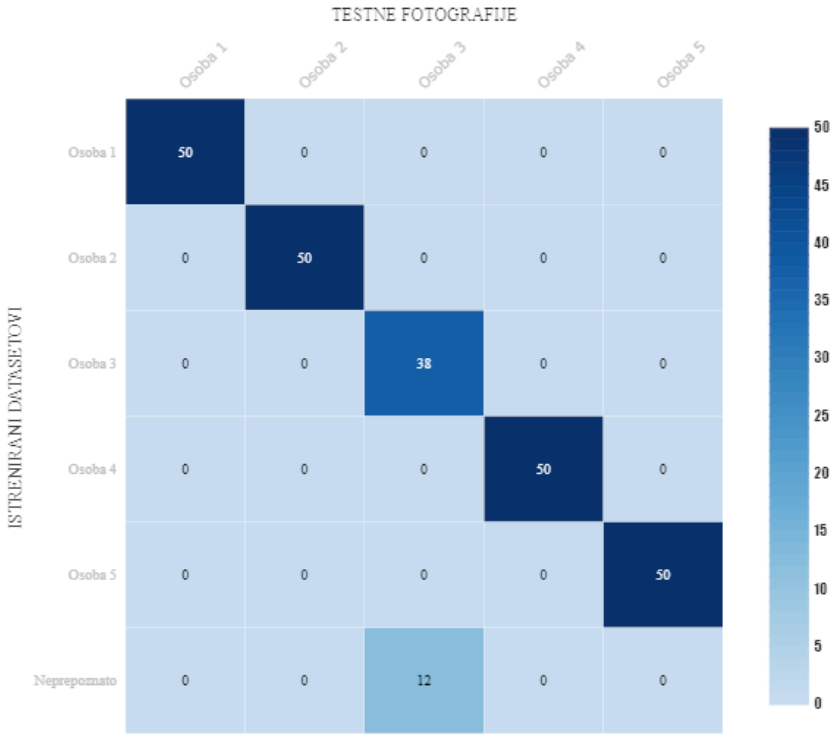
```
cam = cv2.VideoCapture('Ime_video_datoteke.avi')
```

gdje kao ime datoteke navodimo naziv datoteke koju želimo učitati sa ekstenzijom iste redom kao što je prikazano na slici ispod *(Slika 7-6).*



Slika 7.6 Datotečni prikaz slideshow video datoteka

Učitavanjem navedenih video datoteka kroz sustav prepoznavanja rezultiralo je očekivanim rezultatom gdje su na svakoj od fotografija prepoznata lica iako je bilo i povremenih netočnih u usporedbi testnih fotografija sa onima predefiniranim.



Slika 7.7 Matrica prikaza podudarnosti

Iz prikazane matrice *(Slika 7.7)* vidljivo je da je za osobu oznake *„Osoba 3“* ukupno 12 slika koje nisu prepoznate odnosno na istima su prepoznata lica uz pomoć Haarovih kaskada, ali iste nisu prepoznate u usporedbi sa osobama koje su prethodno istrenirane u bazi – nema podudaranja. Kao razlog može se navesti da se radi o ženskoj osobi duže kose koja na navedenim neprepoznatim fotografijama ima nagnutu glavu u stranu prema kutu od otprilike 45 °.

8. Zaključak

Čovjeku je veoma lako prepoznati i definirati neki objekt, dok je u računalnom vidu taj proces mnogo zahtjevniji. S razvojem računalnih tehnologija radi se i na usavršavanju umjetne inteligencije kako bi ona bila što sličnija čovjeku i njegovim navikama s ciljem unaprijeđenja. Potrebno je niz složenih matematičkih operacija i značajki kako bi umjetna inteligencija konkurirala čovjekovoj instinktivnosti, za što je potrebno osigurati i dovoljno potrebnih resursa.

Kao uvod u ovaj diplomski rad prikazano je razvojno okruženje namijenjeno za rad u kojem su nabrojane i opisane pojedine komponente radi uspješne provedbe cjelokupnog procesa. U teoretskom dijelu prikazani su i različiti algoritmi za detekciju lica, da bi se vezano na isto dao uvid u neke od postojećih metoda koje se često koriste za prepoznavanje lica. Za svaku od postojećih metoda prikazan je i opisan programski kod načinjen u Python programskom jeziku.

Nastavno na teoretski dio, korak po korak opisan je postupak pripreme, postavljanja i puštanja u rad Raspberry Pi računala zajedno sa svim komponentama uključujući i one programske, a radi detektiranja i prepoznavanja lica osoba. Za testni dio uzete su Haarove kaskade (Viola-Jones) kao algoritam za detektiranje lica dok je za prepoznavanje iskorištena metoda histograma lokalnih binarnih uzoraka (LBPH). Naveden je i opisan popratni Python programski kod koji jednim dijelom služi za treniranje uz upotrebu prethodno definirane baze podataka (baze lica osoba) kako bi se upotrijebio i za prepoznavanje, odnosno za usporedbu, a po dobivenom rezultatu podudarnosti (točno-netočno) dao i odgovarajući signal na izlazu iz Raspberry Pi uređaja (GPIO) u obliku signala namijenjenog za paljenje odgovarajuće LED lampice što signalno odgovara ukoliko bi se koristila varijanta sa električnom bravom. Vezano na provjeru točnosti zamijećeno je da se točnost detekcije lica može razlikovati zbog raznih parametara kao naprimjer osvjetljenje, rotacija lica u svim smjerovima, udaljenost od kamere, kvaliteta kamere. Iako su fotografije veće oštrine pogodnije za prepoznavanja one su i zahtjevnije za baratanje što je u praktičnom dijelu rezultiralo dodatnim zagrijavanjem Raspberry Pi4 uređaja do povremenih 79 °C bez uporabe dodatnih hladila.

Zbog modularnosti, Raspberry Pi uređaj nudi temelj za daljnji proces u smislu kontrole prolaska odabranih osoba kroz neka vrata. Ideja je da se iskoristi postojeće i

nadogradi u smislu da se uz kontrolu prolaska omogući mrežno prikupljanje podataka (vrijeme, mjesto, osoba) i udaljena kontrola.

U Varaždinu, 10.07.2020. godine.

UNIVERSITY NORTH

Sveučilište Sjever

SVEUČILIŠTE SJEVER

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, MARIO DUBOVEČAK *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom Detekcija i pogađanje lica korištedi Raspberry Pi računala *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)

MARIO DUBOVEČAK M.Dubovečak
(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, MARIO DUBOVEČAK *(ime i prezime)* neopozivo izjavljujem da sam suglasan/na s javnom objavom završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom Detekcija i pogađanje lica korištedi Raspberry Pi računala *(upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)

MARIO DUBOVEČAK M.Dubovečak
(vlastoručni potpis)

9. Literatura

[1] Paul Viola, Michael Jones: Robust Real-time Object Detection, Vancouver-Canada, 13.07.2001. godine, st. 4-13,

[2] Navneet Dalal, Bill Triggs: Histograms of Oriented Gradients for Human Detection, Montbonnot – Francuska, str. 2-7,

[3] Corinna Cortes, Vladimir Vapnik: Support-Vector Networks, Machine Learning br. 20, 1995. godina, str. 273-297,

[4] A. Voulodimos, N. Doulamis: Deep Learning for Computer Vision, Computational Intelligence and Neuroscience Volume 2018, 2018. godina, str. 2-8,

[5] Vahid Kazemi, Josephine Sullivan: One Milisecond Face Alignment with Ensemble of Regression Tree, Computer Vision and Pattern Recognition (CVPR), Columbus Ohio, SAD, 2014. godina,

[6] P.N. Belhumeur, J.P. Hespanha i D.J. Kriegman: Eigenfaces vs. Fisherfaces: Recognition Using Class Spectic Linear Projection, IEEE Transaction on Pattern Analysis and Machine Intelligence Vol.19, br. 7, 2017. godina,

[7] T. Ojala, M. Pietikäinen i D. Harwood: A Comparative Study of Texture Measures with Classification based on Feature distribution, Pattern Recognition Vol.29, br. 1, 1996. godina,

[8] David G. Lowe: Object Recognition from Local Scale-Invariant Features, International Computer Vision Conference, Corfu – Grčka, 1999. godina, str. 2-7,

[9] Herbert Bay, Tinnie Tuytelaars i Luc Van Gool: SURF – Speed Up Robust Features, Lecture Notes in Computer Science, 2006. godina, str. 404-417,

[10] F. Schroff, D. Kalenichenko i J. Philbin: FaceNet – A Unified Embedding for Face Recognition and Clustering, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015. godina, str. 815-823.

10. Popis slika

Slika 2.1 Shematski prikaz razvojnog okruženja 9
Slika 3.1 Četiri različite vrste značajki koje se koriste u okviru 11
Slika 3.2 Vrijednost integralne slike na točki (x,y) 12
Slika 3.3 Primjer značajki sa 4 pravokutnika 13
Slika 3.4 Rekonstrukcija primjera jakog klasifikatora 14
Slika 3.5 Rezultat Haarovih kaskada (izvor. https://vimeo.com/12774628) 16
Slika 3.6 Prikaz slijeda izdvajanja značajki i prepoznavanja objekata 17
Slika 3.7 C-HOG varijante 19
Slika 3.8 Grafički prikaz prozora za detekciju, blokova i ćelija 20
Slika 3.9 Grafički prikaz potpornih vektora 21
Slika 3.10 Hiper-ravnine i margine 21
Slika 3.11 Grafički prikaz gubitka 23
Slika 4.1 Prikaz arhitekture konvolucijske neuronske mreže (izvor. https://i.stack.imgur.com) 24
Slika 4.2 Shematski prikaz DBN-a i DBM-a (gronja dva sloja) 26
Slika 4.3 Uklanjanje šuma u autoenkoderu 27
Slika 4.4 Usporedba CNN, DBN/DBM i SdAs-a 29
Slika 6.1 Koraci u prepoznavanju 33
Slika 6.2 Grafički prikaz hiperravnina-os 37
Slika 6.3 Slikovni prikaz Eigenfaces izlaza 39
Slika 6.4 Izrada LBP-a uzimanjem susjednih 8 piksela oko centralnog piksela i pretvorba u 8 binarnih brojeva 40
Slika 6.5 Kalkulacija 8 binarnih brojeva u decimalni 40
Slika 6.6 Izračunata LBP vrijednost se sprema u izlaznom redu sa istom visinom i širinom kao i originalna slika 40
Slika 6-7 Lokalizacija ključnih točaka 42
Slika 6.8 Orijentacija ključne točke 43
Slika 6.9 Primjer 4x4 lokacijske mreže 43
Slika 6.10 Grafički prikaz SIFT primjera (izvor. https://www.researchgate.net/figure) 44
Slika 6.11 Grafički prikaz dominantne orijentacije 45

Slika 6.12 Gaussova parcijalna derivacija 45
Slika 6.13 Primjer Laplace 46
Slika 6.14 Ključne točke 47
Slika 6.15 Uspravna orijentacija ključnih točaka 48
Slika 7.1 SSH povezivanje sa Raspberry uređajem 52
Slika 7.2 VNC - Desktop RaspberryPi uređaja 53
Slika 7.3 Datotečni hijerarhijski prikaz 56
Slika 7.4 Grafički prikaz programa za prepoznavanje sa rezultatom 60
Slika 7.5 Dijagram tijeka procesa prepoznavanja 60
Slika 7.6 Datotečni prikaz slideshow video datoteka 61
Slika 7.7 Matrica prikaza podudarnosti 62