SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Dominik Šulić

DETEKTIRANJE LICA NA VIDEOZAPISIMA

ZAVRŠNI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Dominik Šulić

Matični broj: 45123/16–R

Studij: Informacijski sustavi

DETEKTIRANJE LICA NA VIDEOZAPISIMA

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Miroslav Bača

Varaždin, lipanj 2020.

Dominik Šulić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Cilj ovog rada je napisati program koji će detektirati ljudska lica na videozapisima te usporediti lica i dati neki postotak sličnosti. Uz pomoć programskog jezika python i biblioteki koje su uključene u njega, napraviti ću detekciju lica i usporedbu između lica na slikama koje korisnik odluči spremiti ili na slikama koje je korisnik unaprijed pripremio u nekom direktoriju.

Ključne riječi: python; skripte; biblioteke; ljudsko lice; karakteristične točke; viola-jones algoritam;

Sadržaj

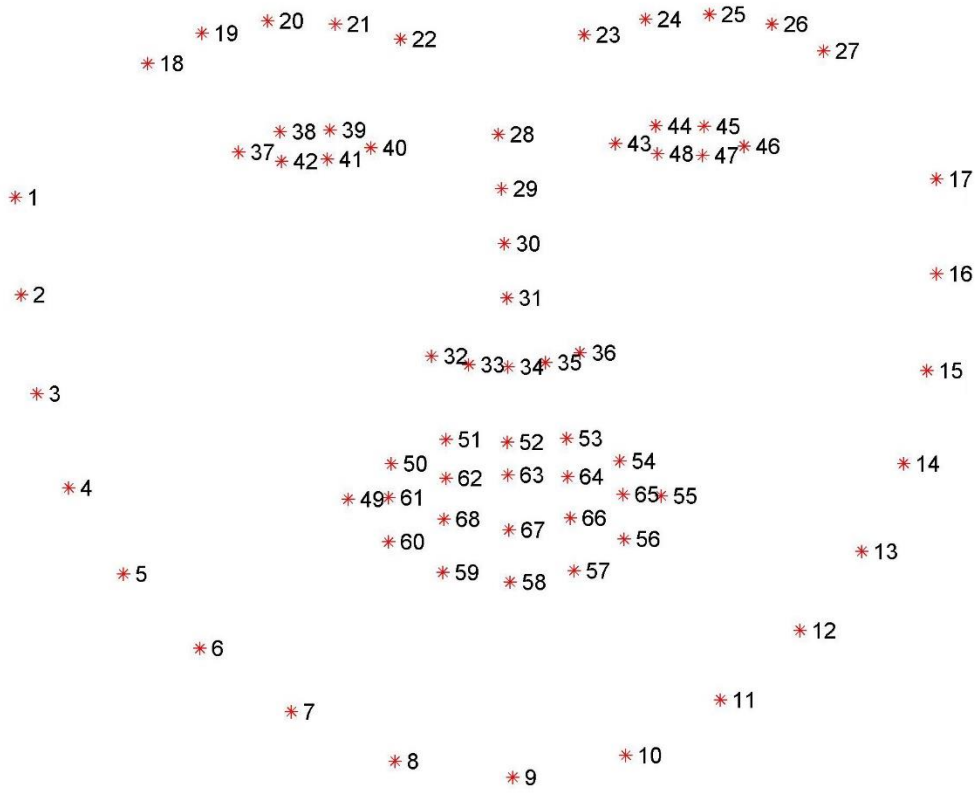
1. Uvod

Za detektiranje lica na videozapisima u ovom radu korišten je programski jezik python (specifične razloge zašto baš python spomenuti ću kasnije). Kako bi ova aplikacija ispravno radila, potrebno je imati ili pripremiti neki videozapis (testirano sa .mp4 datotekama) te ga učitati u aplikaciji (koristeći gumb „load file“). Pritiskom na gumb „detect faces“ započinje detekcija lica na učitanom videozapisu. Detektiranje lica sam realizirao tako da, nakon što se izvrši određeni python kod, se lica „uokvire“ zelenim kvadratom tijekom reproduciranja učitane video datoteke.

Prilikom reproduciranja videozapisa korisnik može pritisnuti tipku s i tipku c. Tipkom s će se spremiti trenutno prikazana slika na videozapisu, dok će se tipkom c zaustaviti izvođenje programa. Treća opcija, „compare saved faces“, izvlači ljudska lica iz prije spremljenih slika (od strane korisnika) te, uz pomoć euklidske distance, uspoređuje ih i daje postotak sličnosti. Četvrta opcija, „compare faces in your chosen folder“, uspoređuje lica iz direktorija kojeg korisnik sam može odabrati (samo od slika koje su .jpg ekstenzije). Generalno, koristeći euklidsku distancu, može se približno odrediti sličnost lica. Smatra se da, ako je razlika između lica veća od 0.6, lice ne pripada istoj osobi.

2. Ljudsko lice

Osnovne karakteristike ljudskog lica se mogu pokazati skupom karakterističnih točaka . Postoje algoritmi koji iz detektiranog lica mogu izvući karakteristične točke, koje reprezentiraju oči, obrve, nos, usta i generalan oblik lica. Ti algoritmi se nalaze u popularnim bibliotekama današnjice, pa tako za python bi to bile dlib, opencv, caffe itd.

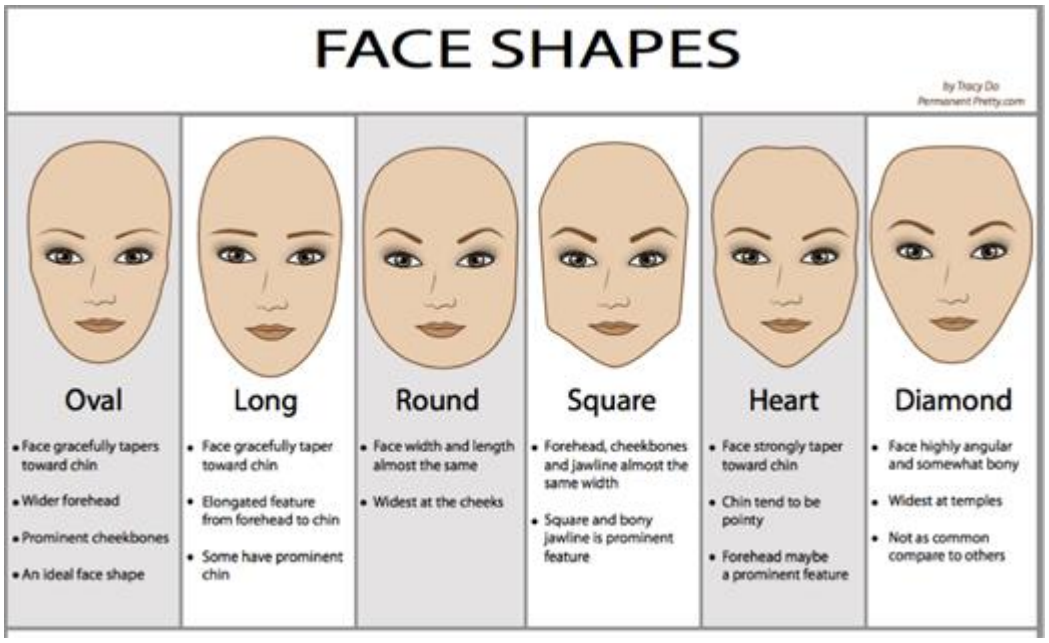


Slika 1: Karakteristične točke ljudskog lica (izvor: https://medium.com/@aanilkayy/extract-eyes-in-face-i%CC%87mages-with-dlib-and-face-landmark-points-c45ef480c1)

Tako je na gornjoj slici vidljivo da je generalno prikazano neko lice, ali samo koristeći skup točaka. Te točke su strogo određene te, npr. točke 37-42 prikazuju lijevo oko (što zna biti vrlo korisno u radu na ovom području).

2.1 Karakteristične točke lica

Za početak, definirati ću generalne oblike lica. Postoje lica ovalnog oblika, duguljasta lica, okrugla, četvrtasta, lica u obliku srca i lica u obliku dijamanta.

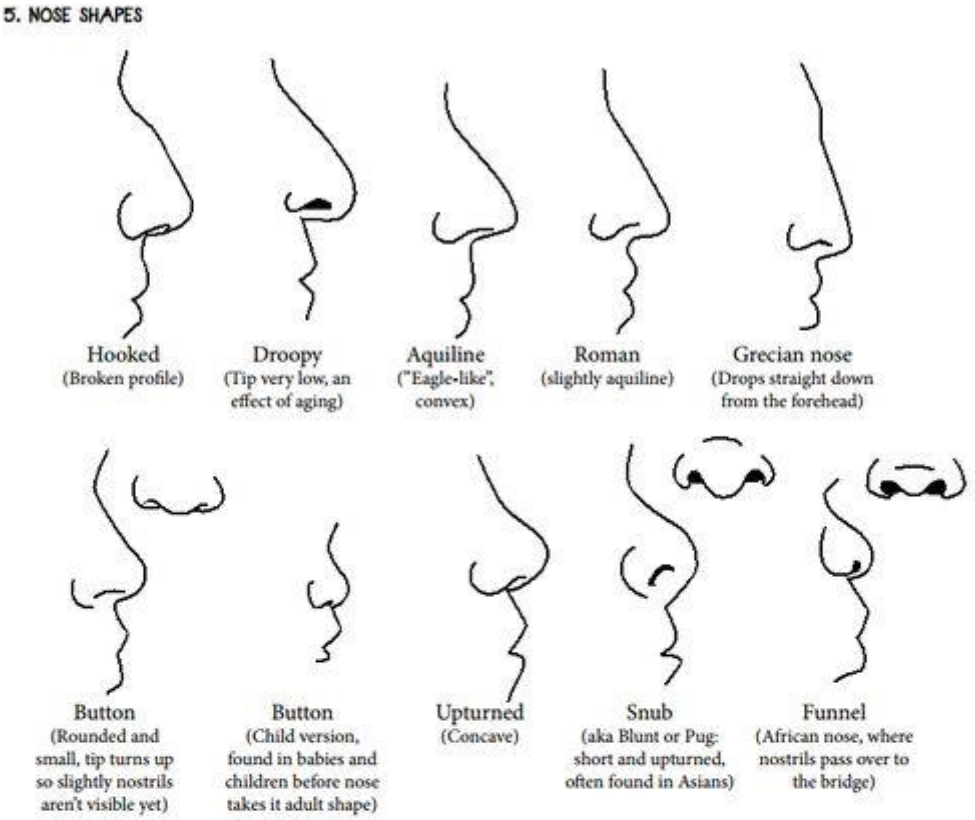


Slika 2: Oblici lica (izvor: https://kamdora.com/2015/10/03/get-the-best-eyebrows-for-your-face-shape/)

Ovalni oblik lica karakteriziran je širim čelom, izraženim jagodicama i taj oblik lica se sužava prema bradi. Duguljasta lica se također sužavaju prema bradi, neka od njih sadrže izraženu bradu i produžena su oblika od čela do brade. Okrugla lica su najširа na obrazima i njihova duljina i širina su skoro pa iste dužine. Lica četvrtastog oblika karakterizira skoro ista dužina čela, jagodica i vilice te je vilica izražena. Lica u obliku srca često sadrže šiljastu bradu, čelo koje može biti izraženo te se jako sužavaju prema bradi. Lica u obliku dijamanta nisu pretjerano česta a sadrže „ćoškasto“ i pomalo koščato lice koje je najšire na „temples“ (područje između čela i oba uha – ravno).

Oblici obrva koji postoje su ravne obrve, zakrivljene, zakrivljene s malenim lukom i velikim lukom, obrve u s obliku i obrve koje su usmjerene prema gore.

Slika 3: Oblici obrva (izvor: https://www.pinterest.com/pin/548805904580647970/)



Slika 4: Oblici nosa i usana (izvor: https://www.pinterest.com/pin/270778996321050059/)

Prvi oblik nosa je u obliku kuke, drugi je „obješen“, tj prema dolje je usmjeren, treći je sličan kao i orlov, poslije toga rimski koji je sličan prijašnjem, pa grčki koji naglo „pada“, pa u obliku gumba, u smislu da je malen i okrugao, nakon toga tip koji je usmjeren prema gore, pa prćast i zadnji je afrički oblik. Oblici usana koji postoje su prirodne, šiljaste prirodne, tanke, u obliku kupidovog luka, pa uni-lip (bez nekog očitog vrha), usne koje su uske i punašne, nakon njih su usne koje su široke i imaju 2 razmaknuta „vrha“ te glamurozne.

Oblici očiju koji postoje su: oči u obliku badema, oči okruglog oblika, „monolid“ oči koje su karakterizirane time da nemaju nabor na očnome kapku, „hooded“ koje su kao da imaju kapuljaču, u smislu da imaju više kože na očnome kapku na gornjem rubu, usmjerene prema gore i prema dolje, oči koje su „duboko usađene“ u smislu da su smještene malo dublje u licu, izbočene oči koje su suprotne duboko usađenima, te oči koje su smještene međusobno blizu i one koje su smještene međusobno daleko. Naravno, oči imaju različito obojane šarenice, no to u mom radu nema smisla uspoređivati zato jer se lica traže na slikama koje su sive, bez boja.

EYE SHAPE CHART

SHILPA ahuja

ALMOND

ROUND

MONOLID

HOODED

UPTURNED

DOWNTURNED

DEEP SET

PROTRUDING

CLOSE SET

WIDE SET

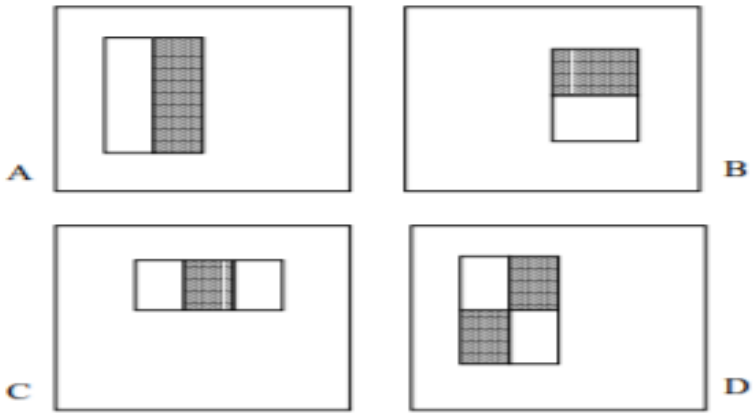
Slika 5: Oblici očiju (izvor: https://shilpaahuja.com/whats-your-eye-shape-makeup/)

Ukratko, to su svi dijelovi lica. Naravno, detekciju lica dodatno otežava činjenica da je većina lica međusobno različita, činjenica da ljudi koriste dodatke kao što su npr. sunčane naočale, kape itd.

3. Viola-Jones algoritam

Viola-Jones algoritam je algoritam koji se koristi za brzu detekciju objekata, koji su objavili Paul Viola i Michael Jones 2001. Iako se na prvi pogled taj algoritam čini starim, on se još i danas koristi zbog toga što ima veliku uspješnost u detekciji objekata i zato što to čini brzo. Algoritam se može, ugrubo, podijeliti na 2 dijela, na 2 stadija, na treniranje i detekciju.

Prvi stadij koji ću objasniti će biti detekcija, zato jer je tako lakše objasniti treniranje, u protivnom samo uskačem u brdo neobjašnjenih pojmova. Za detektiranje, algoritam „prolazi“ kroz sliku koristeći neki četverokut (kvadrat, pravokutnik – može se specificirati veličina) i cilj mu je saznati sadrži li ta slika neko lice. Da bi algoritam odredio sadrži li slika lice ili ne, on traži haarove značajke (eng. *haar features*) u tom četverokutu. Viola i Jones su „odredili“ 3 vrste takvih značajki: značajka s dva pravokutnika (eng. *two-rectangle feature*), značajka s tri pravokutnika (eng. *three-rectangle feature*) i značajka s četiri pravokutnika (eng. *four-rectangle feature*) i te značajke su prikazane na donjoj slici. Nazvane su tako jer sadrže taj broj pravokutnika, no oni se ne moraju nužno javljati u oblicima koji su prikazani dolje (npr. može se javiti pravokutnik koji je isti kao i pravokutnik na slici A ali je postavljen u ležećem položaju).



Slika 6: Haarove značajke (izvor: https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf)

Na gornjoj slici su prikazani pravokutnici s tamnim i svjetlim dijelovima i, npr. pravokutnik B, bi potencijalno mogao prikazivati dio usta ili cijela usta (ili recimo obrvu i dio čela da su tamna i svjetla strana zamijenjene). Svaka ta značajka ima svoju vrijednost. Tu vrijednost se može izračunati tako da se sumira vrijednost pojedinačnih piksela i na tamnom i na svjetlom dijelu i nakon toga se od sume tamnog dijela oduzme suma svjetlog dijela. Taj proces je krajnje jednostavan tako da ga ne ću posebno prikazivati (uključuje samo operacije zbrajanja i oduzimanja). No, poanta ovog algoritma je da je brz i učinkovit te zbog toga se ne računa vrijednost svake značajke posebno (iako to nisu kompliciranе matematičke operacije, na nekoj većoj slici sam broj operacija koje bi procesor trebao izvesti bi bio prevelik). Upravo da bi se izbjeglo intenzivno računanje brojeva, Viola i Jones su prezentirali integralnu sliku (eng. *Integral image*).

Tablica 1: Obična slika

1	2	1	2
1	1	2	1
5	5	6	7
8	8	8	8

Tablica 2: Integralna slika

1	3	4	6
2	5	8	11
7	15	24	34
15	31	48	66

U gornjim tablicama prikazani su neki brojevi. Ti brojevi (u prvoj tablici) predstavljaju „vrijednost“ piksela, te u ovom slučaju će manji brojevi predstavljati svjetliju pozadinu a veći

brojevi tamniju pozadinu. Brojeve sam odabrao nasumično i ne reflektiraju stvarnost ali će poslužiti u demonstrativne svrhe. Prva tablica relativno odgovara A kvadratu na haarovim značajkama i recimo da ona prestavlja čelo i obrvu, u smislu da će donji dio biti obrva a gornji dio će biti dio čela (direktno iznad obrve). Za primjer ću izračunati vrijednost u 2. redu i 3. stupcu. Da bi dobili vrijednost tog piksela, moramo zbrojiti vrijednosti svih piksela u toj manjoj matrici, u matrici koja sadrži 2 reda i 3 stupca, počevši od gornjeg lijevog kuta. Dakle, vrijednost tog polja će biti (1 + 2 + 1 + 1 + 1 + 2) = 8. Postupak je isti za izračun svih ostalih vrijednosti.

Sada dolazi glavni dio, pošto bi za svaku značajku inače trebalo konstantno izračunavati njenu vrijednost, algoritam uvodi jednostavniju alternativu izračuna vrijednosti te značajke. Prvo se kreira integralna slika sa svim njezinim vrijednostima, pa nakon toga ako nas zanima vrijednost neke specifične podmatrice, potrebno je saznati njezine vrijednosti u kutevima (gornji lijevi i desni te donji lijevi i desni). Na gornjoj tablici označio sam četiri polja nekom svjetlo plavom bojom, te ću za nju izračunati vrijednost. Vrijednost se računa vrlo jednostavno, zbroje se gornji lijevi i donji desni kut i od tog zbroja se oduzme vrijednost od donjeg lijevog i gornjeg desnog kuta. (7 + 31) – (15 + 15) = 1. Kao što je već spomenuto, ovaj način izračuna drastično smanjuje broj potrebnih operacija za izračun vrijednosti svake značajke, što se možda ne čini toliko bitnim, ali na nekoj slici koja je recimo 2000 x 2000, ovaj algoritam će uštediti mnogo vremena kod izračuna tih značajki.

Sada slijedi treniranje, a treniranje zapravo „trenira“ računalo da identificira ove značajke. Pod treniranjem se podrazumijeva da se računalu moraju dati neke slike koje zapravo sadrže specifični objekt (u ovom slučaju ljudsko lice) i one bi trebale biti tako i označene, te ostale slike koje ne sadrže ljudsko lice. Kada se algoritmu daju te slike i kada on izvrši svoj trening, algoritam bi trebao odrediti koje značajke najbolje reprezentiraju ljudsko lice. Treniranje je često dugotrajan proces zato jer algoritam (više puta) mora provjeriti svaki dio slike a postoji vrlo velik broj mogućih kombinacija koje se mogu pojaviti. Ugrubo, proces treniranja se svodi na to da se odabere neka značajka koja bi trebala dobro označavati što je lice a što nije. Naravno, lice se ne može predvidjeti samo na temelju jedne male značajke pa algoritam počinje nadodavati ostale značajke. Ovisno o odabranim značajkama, one se zadržavaju ili zamjenjuju, ovisno o postignutim rezultatima. Taj proces se ponavlja sve dok algoritam ne dosegne neku prihvatljivu razinu uspješnosti.

Nakon što algoritam prođe tu fazu učenja, može se početi primjenjivati, ali da bi u potpunosti opisali proces treniranja, postoji još jedna faza koja ubrzava cjelokupni proces detekcije. Ta faza se zove kaskadiranje (eng. *cascading*). Algoritam pregledava neki manji dio slike, i on „uzme“ najvažniju značajku (koju je odredio tijekom treninga – Viola i Jones napominju da su uspjeli kreirati jedan odličan klasifikator (eng. *classifier*) koristeći samo 2 značajke) i ako te značajke nema (moguće je uzeti više tih značajki), taj dio slike se odbacuje kao potencijalni kandidat koji bi mogao sadržavati lice ili dio lica. U fazi kaskadiranja, svaki dio slike mora proći kroz više klasifikatora da bi se tretirao kao važan dio slike, u protivnom, ako jedan klasifikator ne propusti taj dio slike, dio slike u pitanju se automatski odbacuje i ne tretira se kao neko područje interesa. Kaskadiranje dalje ubrzava proces detekcije zato jer su klasifikatori tu da odbace lažne pozitive što štedi vrijeme (a naravno i povećava preciznost algoritma).

4. Kod i objašnjenje koda

Prije sam spomenuo da ću nešto kasnije objasniti zašto sam odabrao python pa ću to napraviti ovdje, kao nekakav uvod u kod. Unatoč tome što su biblioteke koje sam koristio u pythonu dostupne i za neke druge programske jezike (npr. bio sam zainteresiran za c++ programski jezik), imao sam problema s uključivanjem tih biblioteki u programski jezik. Nakon nekoliko neuspješnih pokušaja dodavanja tih biblioteki u c++, odlučio sam probati isto napraviti u pythonu i kroz par minuta sam pripremio cijelo okruženje. No, to nisu jedini razlozi, python je programski jezik dosta visoke razine, koji ima već mnogo biblioteki koje se vrlo lako mogu dodati u njega, i python je u današnje vrijeme odličan izbor za radove koji se bave nekim područjem strojnog učenja ili umjetne inteligencije općenito, uvelike zato što ima puno dostupnih vodiča na internetu te zato jer se veliki, kompleksni, dijelovi koda skrivaju iza javno dostupnih funkcija koje omogućuju rad.

Samo kao napomena, kod za ovu aplikaciju je sadržan u dvije datoteke. Dvije datoteke zato jer je tako preglednije, u smislu da je grafičko sučelje sadržano u jednoj datoteci a funkcionalnost aplikacije u drugoj datoteci, te je ovako osigurana „nezavisnost“ datoteke u kojoj je sadržana funkcionalnost. Nezavisnost u smislu da ne ovisi o prvoj datoteci, nego da se može relativno lako implementirati i u neki drugi sustav.

4.1 Kod

GUI.py datoteka:

```
1.  import tkinter
2.  import tkinter.filedialog
3.  from face_detection import detectFaces, compareFaces

4.  filePath = ""

5.  root = tkinter.Tk()
6.  root.minsize(800, 600)
7.  frame = tkinter.Frame(root, bg = "#0b0c10")
8.  frame.place(relx = 0, rely = 0, relwidth = 1, relheight = 1)
```

```
9.  def btnLoadFileIsClicked():
10.     global filePath
11.     filePath = tkinter.filedialog.askopenfilename()

12. def btnDetectIsClicked():
13.     global filePath
14.     if(filePath != ""):
15.          detectFaces(filePath)

16. def btnCompareIsClicked():
17.     compareFaces("")

18. def btnCompareUserPicturesIsClicked():
19.     path = tkinter.filedialog.askdirectory(title = 'Select Folder')
20.     compareFaces(path)

21.
    btnLoadFile = tkinter.Button(frame, text = "Load file", bg = "#1f28
33", command = btnLoadFileIsClicked, fg = "#ffffff", activebackground =
"#45a29e")

22.
    btnDetect = tkinter.Button(frame, text = "Detect faces", bg = "#1f2
833", command = btnDetectIsClicked, fg = "#ffffff", activebackground = "
#45a29e")

23.
    btnCompare = tkinter.Button(frame, text = "Compare saved faces", bg
 = "#1f2833", command = btnCompareIsClicked, fg = "#ffffff", activebackg
round = "#45a29e")

24.
    btnCompareUserPictures = tkinter.Button(frame, text = "Compare face
s in your chosen folder", bg = "#1f2833", command = btnCompareUserPictur
esIsClicked, fg = "#ffffff", activebackground = "#45a29e")

25. def drawHomeScreen():
26.
         btnLoadFile.place(relx = 0.35, rely = 0.15, relwidth = 0.3, re
lheight = 0.1)
27.
         btnDetect.place(relx = 0.35, rely = 0.35, relwidth = 0.3, relh
eight = 0.1)
```

```
28.
        btnCompare.place(relx = 0.35, rely = 0.55, relwidth = 0.3, rel
height = 0.1)
29.
        btnCompareUserPictures.place(relx = 0.35, rely = 0.75, relwidt
h = 0.3, relheight = 0.1)
30.     root.mainloop()


31.  drawHomeScreen()
```

Face_detection.py datoteka:

```
1. from cv2 import cv2
2. from imutils.video import FileVideoStream
3. import dlib
4. import os
5. import glob
6. import numpy
7. import math


8. imageCounter = 0
9. scaleFactor = 1.2
10.  minNeighbors = 3
11.  minSize = (50, 50)
12.  cascadePath = "../haarcascade_frontalface_alt.xml"
13.  predictorPath = "../shape_predictor_5_face_landmarks.dat"
14.  faceRecognitionModelPath = "../dlib_face_recognition_resnet_model_v
   1.dat"
15.  savedImagesPath = os.getcwd() + "/saved_images"
16.  detectedFacesPath = savedImagesPath + "/detected_faces"


17.  if not os.path.exists("saved_images"):
18.      os.mkdir("saved_images")


19.  os.chdir(savedImagesPath)


20.  if not os.path.exists("detected_faces"):
21.      os.mkdir("detected_faces")


22.  def detectFaces(filePath):
23.      cascadeClassifier = cv2.CascadeClassifier(cascadePath)
24.      video = FileVideoStream(filePath).start()
```

```
25.       global imageCounter

26.       while video.more():
27.           frame = video.read()
28.           keyPress = cv2.waitKey(1)

29.           if keyPress & 0xFF == ord('c'):
30.                  print("Stopping the detection...")
31.                  break
32.           if keyPress & 0xFF == ord('s'):
33.                  print("Saving the image...")
34.                  imageCounter += 1
35.                  cv2.imwrite('image #' + str(imageCounter) + '.jpg',
    frame)

36.           grayFrame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
37.           detectedFaces = cascadeClassifier.detectMultiScale(grayFra
   me, scaleFactor = scaleFactor, minNeighbors = minNeighbors, minSize =
    minSize)

38.           if len(detectedFaces) > 0:
39.                 for (x, y, w, h) in detectedFaces:
40.                      cv2.rectangle(frame, (x, y), (x + w, y + h), (0
   , 255, 0), 2)

41.           cv2.imshow('Face detection', frame)

42.       video.stop()

43.  def compareFaces(filePath):
44.       faceDetector = dlib.get_frontal_face_detector()
45.       shapePredictor = dlib.shape_predictor(predictorPath)
46.       faceRecognitionModel = dlib.face_recognition_model_v1(faceReco
   gnitionModelPath)
47.       faceDescriptorsForComparison = []
48.       faceMatchThreshold = 0.6
49.       faceNumber = 0

50.       if(filePath == ""):
51.            os.chdir(detectedFacesPath)
```

```
52.            for f in glob.glob(os.path.join(savedImagesPath, "*.jpg")
):
53.                 print("File: {}".format(f))
54.                 image = dlib.load_rgb_image(f)
55.                 detectedFaces = faceDetector(image, 1)
56.                 print("Faces detected: {}".format(len(detectedFaces
)))


57.                 for face in detectedFaces:
58.                       croppedImage = image[face.top():face.top() +
abs(face.top()-face.bottom()), face.left():face.left() + abs(face.left()-
face.right())]
59.                       cv2.imwrite('face #' + str(faceNumber+1) + '.
png', croppedImage)
60.                       faceNumber += 1
61.                       shape = shapePredictor(image, face)
62.                       faceDescriptor = faceRecognitionModel.compute
_face_descriptor(image, shape)
63.                       faceDescriptorsForComparison.append(numpy.asa
rray(faceDescriptor))


64.                 os.chdir(savedImagesPath)
65.       else:
66.           os.chdir(filePath)
67.           if not os.path.exists("detected_faces"):
68.                 os.mkdir("detected_faces")
69.           os.chdir(filePath + "/detected_faces")


70.           for f in glob.glob(os.path.join(filePath, "*.jpg")):
71.                 print("File: {}".format(f))
72.                 image = dlib.load_rgb_image(f)
73.                 detectedFaces = faceDetector(image, 1)
74.                 print("Faces detected: {}".format(len(detectedFaces
)))


75.                 for face in detectedFaces:
76.                      croppedImage = image[face.top():face.top() + a
bs(face.top()-face.bottom()), face.left():face.left() + abs(face.left()-
face.right())]
77.                       cv2.imwrite('face #' + str(faceNumber+1) + '.p
ng', croppedImage)
78.                       faceNumber += 1
79.                       shape = shapePredictor(image, face)
```

```
80.                         faceDescriptor = faceRecognitionModel.compute_
face_descriptor(image, shape)
81.                         faceDescriptorsForComparison.append(numpy.asar
ray(faceDescriptor))


82.           os.chdir(savedImagesPath)


83.       for i in range(0, len(faceDescriptorsForComparison)):
84.             euclideanDistance = numpy.linalg.norm(faceDescriptorsFor
Comparison[0] - faceDescriptorsForComparison[i])


85.             if euclideanDistance > faceMatchThreshold:
86.                   if(euclideanDistance >= 1.0):
87.                         print("The distance between face #1 and face
#{} is {} and they are {}% similar.".format(i+1, euclideanDistance, 0))
88.                   else:
89.                         rangeValue = (1.0 - faceMatchThreshold)
90.                         linearValue = (1.0 - euclideanDistance) / (ra
ngeValue * 2.0)
91.                         similarityPercentage = linearValue * 100
92.                         print("The distance between face #1 and face
#{} is {} and they are {}% similar.".format(i+1, euclideanDistance, round(s
imilarityPercentage, 2)))
93.             else:
94.                   rangeValue = faceMatchThreshold
95.                   linearValue = 1.0 - (euclideanDistance / (rangeVal
ue * 2.0))
96.                   linearValue += ((1.0 - linearValue) * math.pow((li
nearValue - 0.5) * 2, 0.2))
97.                   similarityPercentage = linearValue * 100
98.                   print("The distance between face #1 and face #{} i
s {} and they are {}% similar.".format(i+1, euclideanDistance, round(simila
rityPercentage, 2)))
99.  cv2.destroyAllWindows()
```

4.2 Objašnjenje

U ovome poglavlju biti će objašnjene obje skripte koje sam napravio, liniju po liniju koda, sve do kraja face_detection.py datoteke. Ovaj pristup sam odabrao zato jer mislim da je preglednije imati sav kod na jednom mjestu pa ga objašnjavati postepeno nego da nadodajem liniju po liniju i objašnjavam tako. Neke važnije linije ću kopirati pri objašnjavanju (koje su iz druge skripte).

4.2.1 GUI.py

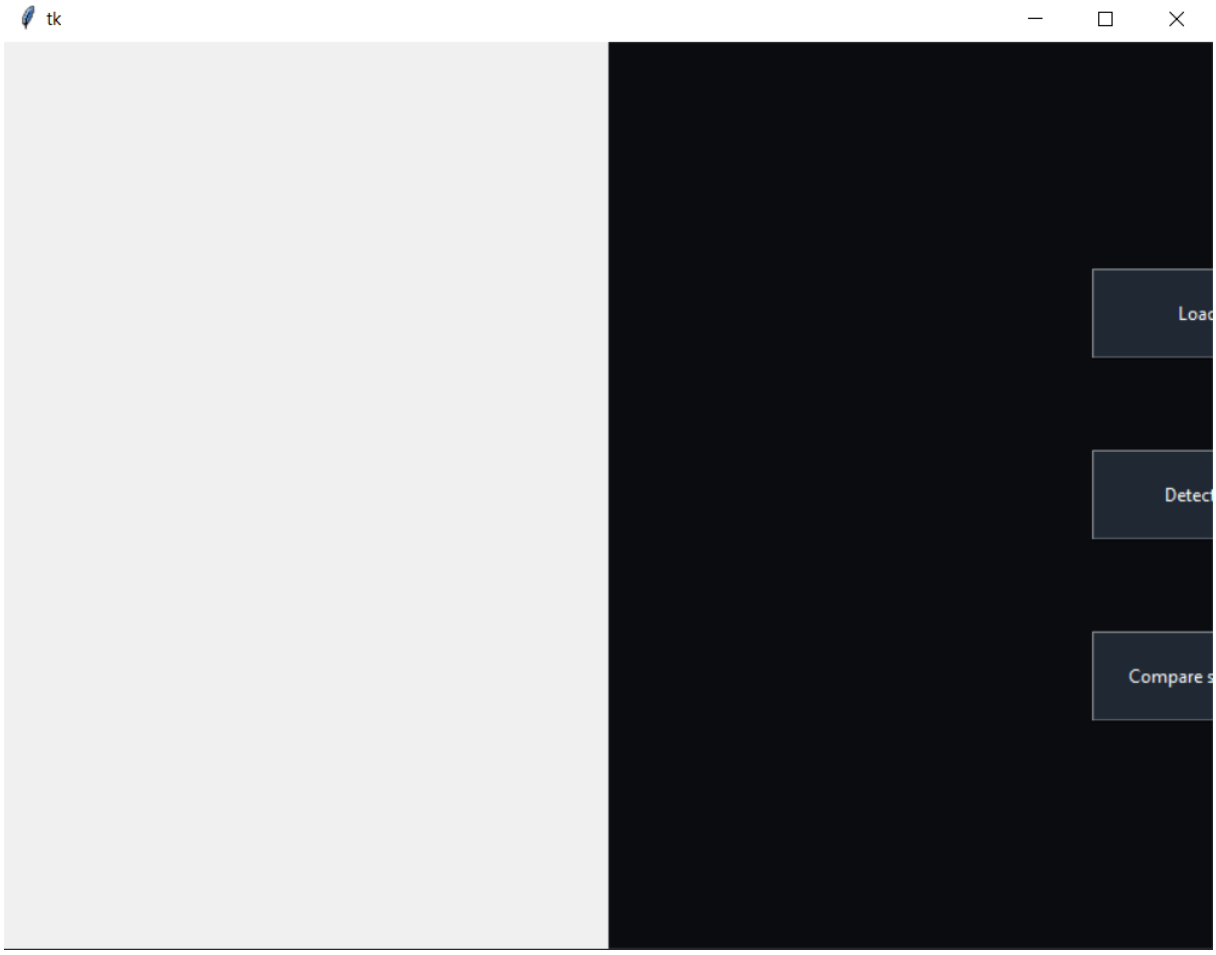
GUI.py datoteka sadrži kod vezan uz grafičko sučelje aplikacije. Koristeći ključnu riječ „import“ u pythonu se u trenutnu skriptu može uključiti ili dodati određeni dio koda. Taj dio koda je prethodno potrebno dodati u python biblioteke, koristeći „pip install“ ili neku drugu metodu dodavanja biblioteki, no u to ne ću ulaziti. Nakon što se uspješno uključi biblioteka u python, pristup njenom kodu je omogućen korištenjem ključne riječi „import“ nakon koje slijedi ime biblioteke ili skripte. Tako je u prvoj liniji koda uključen „tkinter“.

Tkinter je pythonova standardna biblioteka za rad sa grafičkim sučeljima. U drugoj liniji koda se uključuje specifični dio tkinter biblioteke, „filedialog“. Taj dio sam uključio u skriptu zato jer pruža vrlo jednostavan način pristupanja programu za upravljanje datotekama (eng. *file explorer* ili *windows explorer*), a taj dio je potreban korisniku da bi mogao odabrati videozapis na kojem želi detektirati lica. Treći i zadnji import uključuje funkcije iz druge datoteke koja sadrži kod (skripte). Specifično, uključuju se dvije funkcije, jedna za detektiranje lica (imena „detectFaces“) i druga za usporedbu lica (imena „compareFaces“). Funkcija je zapravo skup linija koda kojima je dano neko ime kako bi se kasnije mogla „pozvati“ tj. koristiti pod tim imenom. Funkcije se izvršavaju samo kada ih se pozove. Riječ „from“ označava da se iz specifične skripte uključuje neki specifični dio koda, i treba točno imenovati određenu skriptu.

Na četvrtoj liniji koda deklarirana je varijabla imena „filePath“ i njoj je dodijeljen prazan znakovni niz (eng. *string* - koristiti ću engleski naziv). Ta varijabla je, u biti, globalna varijabla, što znači da njoj sve funkcije (za koje je tako specificirano) imaju pristup. Tu varijablu koristim kako bi spremio putanju do videozapisa kojeg korisnik odabere koristeći upravitelja datotekama. Peta linija koda u varijablu imena „root“ zapravo pohranjuje običan prozor s osnovnim funkcionalnostima (minimiziraj, maksimiziraj i zatvori), naslovnu traku i prazan prostor gdje će uskoro biti neki grafički elementi, koristeći tkinterovu funkciju „Tk“. Na sljedećoj liniji koda, šestoj, na spremljeni prozor u varijabli imena „root“ se, funkcijom „minsize“, stavlja ograničenje. To ograničenje određuje minimalnu širinu i visinu prozora programa u pikselima. Program će, u mom slučaju, uvijek biti najmanje širok 800 piksela i visok 600 piksela i ne će se nikako moći staviti na širinu/dužinu manju od specificirane, osim programskim putem, ali će se prozor uvijek moći povećati (korištenjem miša).

Sedma linija koda koristi još jedan grafički element (eng. *widget*) i sprema ga u varijablu imena „frame“. „tkinter.Frame“ funkcija uzima kao prvi parametar „parent window“. Taj „roditeljski“ prozor je zapravo „root“ varijabla koju sam prije deklarirao. Ostali parametri su samo opcije koje programer želi koristiti, tako sam ja specificirao samo pozadinsku boju (u heksadecimalnom obliku) i ja sam, više-manje nasumično, odabrao neku crnu. Osim pozadinske boje mogu se koristiti i neke druge opcije, npr. „height“ i „width“ za visinu i širinu, „cursor“ koji se može staviti na točku (eng. *dot*), strelicu (eng. *arrow*) ili nešto treće, a promjeniti će izgled miša na ekranu kada pređete njime preko otvorenog prozora itd.

Sljedeća linija koda doslovno stavlja varijablu „frame“ na prozor („root“). Parametri koje sam specificirao su „relx“ i „rely“ koji su oba jednaki 0, te „relwidth“ i „relheight“ koji su oba jednaki 1. Relx i rely su pomaci (eng. *offset*), x je horizontalni pomak a y je vertikalni. Relx je doslovno pomak cijele frame varijable na glavnom prozoru za specificirani postotak prozora. Relwidth i relheight određuju koji postotak glavnog prozora („root“) bi „frame“ varijabla trebala zauzimati. Tu sam stavio 1, tj. treba pokrivati cijeli prozor, neovisno o njegovoj širini i visini.



Slika 7: Grafičko sučelje aplikacije kada je relx = 0.5

Deveta linija koda koristi ključnu riječ „def“ kako bi, pa, definirala funkciju i ista stvar je sa 12.,16. i 18. linijom koda. Sve četiri spomenute linije koda kreiraju funkcije koje se pozivaju kada je pritisnut neki gumb, do čega ću uskoro doći. U prvoj toj funkciji, „btnLoadFileIsClicked“, na desetoj liniji koda, deklarira se varijabla imena „filePath“ kao globalna, što omogućava čitanje i mijenjanje te varijable unutar ove funkcije. Na 11. liniji koda koristi se funkcija imena „askopenfilename“ koja zapravo pokreće upravitelja datotekama i daje korisniku opciju da izabere neki videozapis. Nakon što je korisnik odabrao videozapis, funkcija automatski vraća apsolutnu putanju do tog videozapisa i sprema ga u „filePath“ varijablu.

12. linija koda opet definira funkciju. Ta funkcija je imena „btnDetectIsClicked“ i u njoj je opet definirana globalna varijabla „filePath“ te tu zapravo mogu objasniti zašto je to globalna varijabla. Zbog načina na koji sam napisao ostatak programa, odlučio sam napraviti globalnu varijablu (imena „filePath“) tako da joj može pristupiti više funkcija. Razlog tome je što za

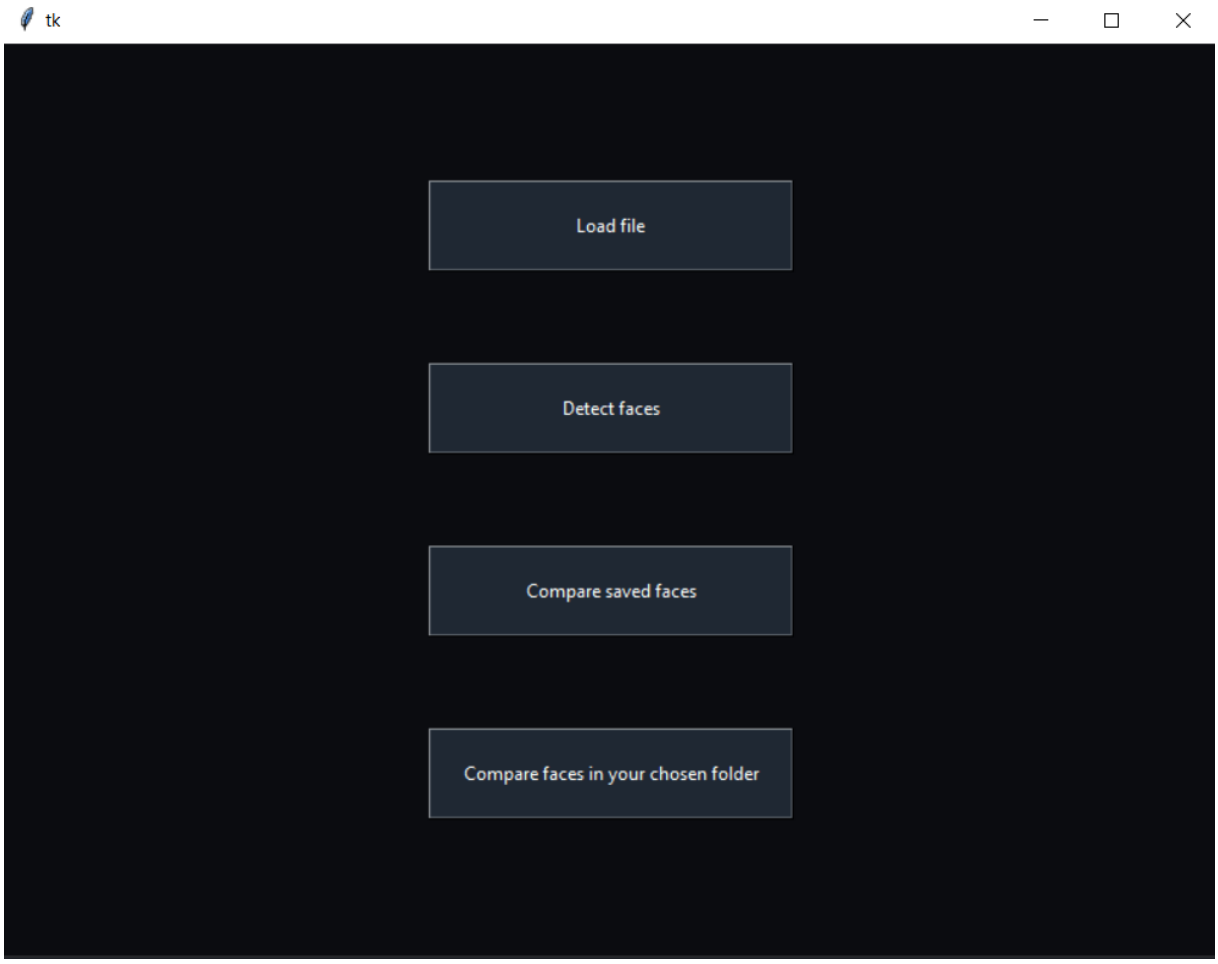
određene gumbe (do kojih kasnije dolazim) morate specificirati određene funkcije koje se pozivaju kada su oni pritisnuti. Najjednostavniji način koji sam uočio je bio taj da specificiram „fliePath“ kao globalnu varijablu tako da joj mogu pristupiti obje funkcije („btnDetectIsClicked“ i „btnLoadFileIsClicked“). Odlučio sam koristiti 4 gumba, prvi od kojih služi za učitavanje puta do videozapisa, zato jer je tako nekako bolje podijeljeno sučelje, hijerarhijski, i da korisniku dam mogućnost aktiviranja svakog dijela programa po želji (u protivnom bi bio slučaj da imam samo tri gumba a na prvom gumbu je spojeno učitavanje videozapisa i detekcija lica).

Pošto sada i funkcija „btnDetectIsClicked“ ima pristup varijabli „filePath“, ona može pročitati njenu vrijednost, što je cijela poanta. 14. linija koda provjerava sadrži li varijabla „filePath“ ikoji znak, te, samo ako sadržava, izvodi se 15. linija koda koja je ranije uključena naredbom „import“. Za tu funkciju potreban je argument koji je string, a taj string treba sadržavati putanju do videozapisa. 16. linija koda definira funkciju imena „btnCompareIsClicked“ a u njoj je sadržana samo jedna linija koda, a ta linija koda je zapravo funkcija koja je uključena u program preko ključne riječi „import“ na početku ove skripte. Ta funkcija sadrži parametar ali taj parametar je prazan string. Na 18. liniji koda nalazi se još jedna funkcija, „btnCompareUserPicturesIsClicked“, koja koristi istu uključenu funkciju ali ju poziva sa parametrom koji sadrži apsolutnu putanju do direktorija u kojem korisnik želi usporediti slike.

21., 22., 23. i 24. linija koda su u biti skoro identične, kreiraju grafičke elemente (gumbove specifično) a funkcija uzima kao prvi parametar „parent window“ (što je u ovom slučaju „frame“, gumbovi se dodaju na „frame“, a „frame“ na „root“ prozor) te nakon toga niz raznih opcija. Opcije koje sam ja koristio su: „text“ što stavlja određeni tekst na gumb, „bg“, „fg“ i „activebackground“ što samo mjenjaju boje, te najvažnija opcija, „command“ koja određuje koja funkcija će se izvršiti kada je taj gumb pritisnut. Svaki gumb je prikladno imenovan te su imena usklađena s funkcijama koje oni pozivaju. 25. linija koda definira funkciju, a ta funkcija zapravo stavlja gumbove na određene pozicije i počinje izvršavanje programa.

Funkcija place uzima iste parametre koje sam već spomenuo („relx“, „rely“, „relheight“ i „relwidth“ te ih ne ću ponovno objašnjavati – jedino vrijedno spomena je da se u obzir uzima „parent window“ kod izračuna), a ona doslovno stavlja već prije deklarirane gumbove na „frame“, na određenu, specificiranu, lokaciju. Zadnja linija koda u toj funkciji je „root.mainloop()“ koja pokreće sve, započinje beskonačnu petlju te čeka na nekakav događaj koji se mora obraditi (a što je prethodno potrebno specificirati – gumbovi i funkcije koje se pozivaju kada su

oni pritisnuti). Jedino što je preostalo u ovoj skripti je poziv funkcije „drawHomeScreen“ koja izvršava prije spomenute stvari.



Slika 8: Grafičko sučelje aplikacije

4.2.2 Face_detection.py

Face_detection.py je skripta koja sadrži svu pozadinsku logiku programa. Njene funkcije se pozovu kada je na grafičkom sučelju pritisnut gumb za detektiranje lica ili usporedbu lica.

Prvih sedam linija koda u ovoj skripti su samo uključene biblioteke. Prva od njih je „cv2“ i ona je zapravo openCV biblioteka koja se bavi strojnim učenjem i stvarima kao što su detekcije lica (tj. objekata općenito) i sl. Sljedeća biblioteka je „imutils“ (tj. „imutils.video“) koja sadrži mnogo funkcija koje olakšavaju osnovne funkcionalnosti obrade slika (npr. rotacija,

promjena dimenzija itd.). „Dlib“ je biblioteka koju sam koristio za usporedbu lica te ona također sadrži mnogo algoritama koji se primjenjuju u strojnom učenju. „Os“ biblioteka omogućava korištenje funkcija koje su ovisne o operacijskom sustavu (korišteno za kreiranje direktorija u mom slučaju). „Glob“ se koristi za traženje datoteka sa specifičnim nastavkom (npr. jpg, png itd.). „Numpy“ pruža mnogo funkcionalnosti za znanstvene izračune, što joj je i primarna namjena. „Math“ sadrži matematičke funkcije (uključujući trigonometrijske funkcije, logaritmičke itd.) i par matematičkih konstanti.

Sljedeće linije koda su tu samo da se odrede neke konstantne vrijednosti. Tako je „imageCounter“ varijabla jednaka 0, „scaleFactor“ je 1.2, „minNeighbours“ = 3 i „minSize“ = (50,50). „minSize“ je kolekcija čiji se elementi ne mogu mjenjati i poredani su (eng. *Tuple*). U ovome slučaju, „minSize“ sadrži dva elementa, oba su jednaka 50. „CascadePath“, „predictorPath“ i „faceRecognitionModelPath“ su string varijable koje sadrže relativne putanje do tri specifične datoteke koje ću kasnije objasniti. „savedImagesPath“ sadrži putanju do „saved_images“ direktorija. „Os.getcwd“ je funkcija koja vraća apsolutnu putanju do trenutnog aktivnog direktorija (direktorija u kojem se ova aplikacija pokreće) u obliku stringa. Na to sam još nadodao „/saved_images“. Pojašnjeno, „saved_images“ direktorij je direktorij u kojem će se spremati sve slike koje korisnik odluči spremiti (pritiskom na tipku s prilikom detektiranja lica na odabranom videozapisu). „detectedFacesPath“ se nadovezuje na „savedImagesPath“ dodavajući „/detected_faces“. U „detected_faces“ direktoriju će biti spremljena lica koja se uspoređuju.

17. linija koda koristi ključnu riječ „if“ kako bi provjerila postoji li direktorij imena „saved_images“ u trenutnom aktivnom direktoriju te ako on ne postoji, kreira se 18. linijom koda. „Os.mkdir“ je skraćeno od kreiraj direktorij (eng. *make directory*) i ta linija koda se aktivira samo ako ne postoji taj direktorij u trenutnom aktivnom direktoriju. Poanta ovih linija je ta da, ako neki novi korisnik prvi puta pokreće aplikaciju ili ako je neki stariji korisnik obrisao taj direktorij, automatski mu se kreiraju oba potrebna direktorija za rad. 19. linija koda, „os.chdir“, mijenja trenutni aktivni direktorij i mijenja ga na vrijednost varijable „savedImagesPath“ (koja je jednaka putanji ovog novog direktorija). 20. i 21. linija koda rade isto što i 17. i 18., samo za direktorij imena „detected_faces“ (koji se nalazi unutar direktorija imena „saved_images“).

Prva funkcija definirana je na 22. liniji koda. Ta funkcija se poziva kada korisnik odabere videozapis koji želi reproducirati i na kojem želi detektirati lica. Parametar koji se prosljeđuje toj funkciji je string i on mora sadržavati apsolutnu putanju do videozapisa koji se želi reproducirati (to je funkcija koja se poziva na pritisak gumba – napravljeno u GUI.py). Na 23. liniji koda se definira varijabla imena „cascadeClassifier“ i u nju se zapravo sprema taj klasifikator koji je prije spomenut. Argument za tu funkciju je putanja do klasifikatora (koji se nalazi u istom direktoriju u kojem su i skripte) i u ovom slučaju korištena je relativna putanja.

Na liniji ispod se deklarira objekt imena „video“ i u taj objekt se preko imutils biblioteke učitava videozapis koji se nalazi na specificiranoj putanji. Klasa FileVideoStream sadrži nekoliko metoda, od kojih ja većinu koristim. Za početak, pri kreiranju objekta, ta klasa „inicijalizira“ videozapis i jednu boolean varijablu koja označava bi li se dretva (koja se uskoro kreira) trebala zaustaviti ili ne. Također pokreće podatkovnu strukturu red (eng. *Queue*) i prije spomenutu dretvu. Metodom „start“, ta biblioteka pokreće prethodno kreiranu dretvu koja počinje čitati slike (eng. *frame*) i stavlja ih u red. Još preostaje globalna varijabla imena „imageCounter“ koja je samo tu da prati broj trenutne slike (jer kada korisnik pritisne tipku s, spremaju se slike pa se ova varijabla koristi kao pomoć u numeraciji i imenovanju tih slika).

26. linija koda provjerava sadrži li (u objektu imena „video“) red ikoju sliku za obraditi te vraća „true“ ako sadrži. Koristeći ključnu riječ „while“ kreirana je petlja koja će se vrtjeti sve dok će funkcija „more“ vraćati true (ili dok se ručno ne prekine izvođenje). Prva stvar koja se izvodi unutar te petlje je da se u varijablu „frame“ učitava sljedeća slika iz reda. Na 28. liniji koda funkcijom „waitKey“ se čeka na pritisak tipke (u trajanju od 1 milisekunde) te, ako je detektiran pritisak tipke, u varijablu „keyPress“ se sprema šifra pritisnute tipke. U protivnom se u tu varijablu sprema vrijednost -1.

Sljedeća linija koda je malo kompliciranija, no ona samo provjerava koju tipku je korisnik pritisnuo i odgovara li specificiranim tipkama u kodu. Ukratko, u varijabli „keyPress“ sadržan je 32-bitni integer koji reprezentira tipku koju je korisnik pritisnuo i ta vrijednost se modificira & operatorom na način da u tom 32-bitnom integeru preostane samo zadnjih 8 bitova. Potrebno je ovako smanjiti vrijednost tog integera zato jer „ord('c')“ vraća vrijednost od 0 do 255 (koja je reprezentirana sa 8 bitova).

Na kraju se uspoređuje vrijednost koju je vratio „ord('c')“ i 8-bitni integer i samo ako su one jednake, kod ulazi u if petlju. Pa tako se prvo provjerava je li korisnik pritisnuo tipku c i ako je u konzoli se naredbom „print“ ispiše poruka koja označava da se videozapis prestaje reproducirati i na kraju se izlazi iz while petlje. Još se provjerava za tipku s, te, ako ju je korisnik pritisnuo, u konzoli se ispiše da se trenutna slika sprema, nakon toga se inkrementira vrijednost varijable „imageCounter“ i pomoću naredbe „imwrite“, trenutna slika se sprema sa specificiranim nazivom u trenutni aktivni direktorij.

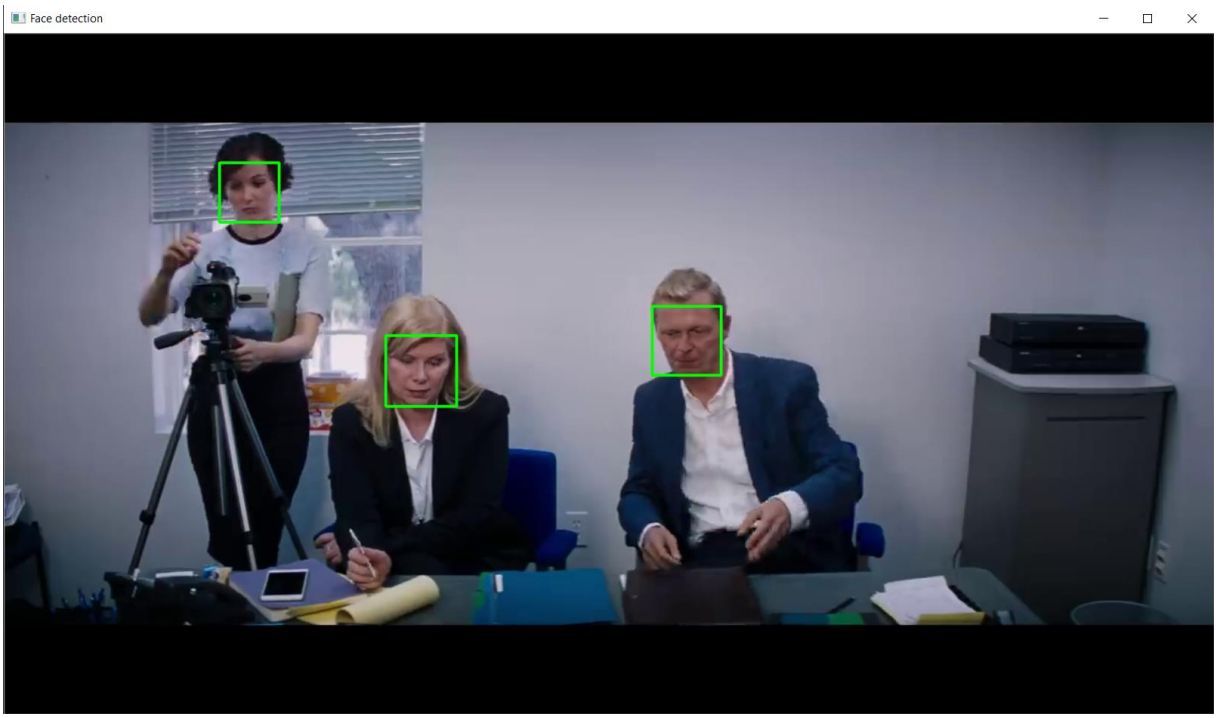
Nakon svega ovoga, na red dolazi sama detekcija lica, koja je poprilično laka za korištenje. Prvo se, na 36. liniji koda, trenutačna slika koja se obrađuje (spremljena u varijabli „frame“) funkcijom „cvtColor“ konvertira u sliku sivih tonova (BGR -> Gray konverzija). Algoritmu boja ništa ne znači pa je ona kompletno maknuta iz slike što automatski ubrzava rad programa (zato jer ima manje stvari za procesirati – svaki piksel nije reprezentiran sa 3 vrijednosti za RGB nego sa 1 vrijednošću).

```
37.  detectedFaces = cascadeClassifier.detectMultiScale(grayFrame, scale
   Factor = scaleFactor, minNeighbors = minNeighbors, minSize = minSize)
```

Gornja linija koda je kopirana zato jer je to najvažnija linija koda u cijelome programu. Samo da istaknem, ova funkcija se poziva nad klasifikatorom za prednji dio lica. Prvi argument gornje funkcije je slika nad kojom se detekcija vrši te je tu prosljeđena slika sivih tonova. Drugi parametar je „scaleFactor“ koji određuje koliko će se slika smanjiti (to bi trebalo poboljšati proces detektiranja) prilikom izvođenja algoritma. Sljedeći parametar je „minNeighbors“ koji određuje koliko „susjeda“ svaki pravokutnik treba imati da bi se on zadržao. Ovaj parametar se uvodi zato da bi se izbjegli lažni pozitivi (zato jer bi neke značajke koje nisu dio ljudskog lica mogle proći klasifikatore – zato se ti lažni pozitivi odbacuju ako ih je malo tj. ako nemaju susjeda.

O ovome je možda bolje razmišljati na način da će se teško detektirati neko lice koje bi se sastojalo od 1 ili 2 značajke). Zadnji parametar koji funkcija uzima je „minSize“ i on određuje najmanju veličinu objekta koju taj objekt može poprimiti, u smislu da su svi objekti manji od te veličine ignorirani.
Rezultat gornje funkcije se sprema u varijablu „detectedFaces“. Nakon toga se pomoću ključne riječi „if“ provjerava je li duljina od „detectedFaces“ veća od 0, i, u slučaju da je, algoritam obrubi to lice ili ta lica zelenim pravokutnicima (eventualno kvadratima). Na samome kraju te

petlje se prikaže trenutna slika i ako su na njoj detektirana neka lica, ta lica će biti obrubljena. Zadnja linija u toj funkciji je „video.stop“ koja jednostavno zaustavlja reprodukciju videozapisa.



Slika 9: Detekcija lica na videozapisu

43. linijom koda definira se nova funkcija imena „compareFaces“ koja uzima kao parametar putanju do direktorija u kojem su spremljene slike. Ovu funkciju koristim kako bih usporedio lica i probao kvantificirati postotak sličnosti između njih. Na početku te funkcije se definira „faceDetector“ i koristi se funkcija „get_frontal_face_detector“ da bi se učitao detektor lica (slično kao i u funkciji „detectFaces“). Ovaj detektor lica nije toliko važan i ne ću ga detaljno objašnjavati pošto ga koristim samo kao alat da bih došao do cilja koji želim ispuniti a to je usporedba lica (uostalom, već prije je objašnjen glavni algoritam ovog rada). Ispod detektora definiran je „shapePredictor“ a to je dlibova funkcija koja se koristi za učitavanje tog objekta u memoriju. Taj objekt se koristi, u ovom slučaju, za detekciju karakterističnih točaka lica. U liniji ispod se učitava „faceRecognitionModel“, a on se koristi za „preslikavanje“ (eng. *map*) ljudskog lica u 128D vektor. Uz pomoć tog vektora moguće je euklidskom distancom odrediti sličnost lica. U liniji ispod je definirano prazno polje i nakon toga su definirane još dvije varijable koje ću kasnije spomenuti.

50. linija koda koristi ključnu riječ „if“ kako bi razdvojila tijek koda, a taj tijek koda se razdvaja na temelju vrijednosti „filePath“ varijable koja je argument „compareFaces“ funkcije. Ako je vrijednost te varijable jednaka praznome stringu, izvesti će se prvi dio, u protivnome će se izvesti drugi dio, a drugi dio ne ću uopće objašnjavati pošto je skoro identičan prvome dijelu, razlike su male. Direktno ispod te ključne riječi, mijenja se trenutni aktivni direktorij u vrijednost varijable „detectedFacesPath“. Ako smo ušli u taj „if“, to znači da je funkciji proslijeđen prazan string, što zapravo znači da korisnik nije odabrao svoj direktorij za usporedbu lica nego je koristio standardni (direktorije koji se automatski kreiraju kod skripti). Sljedeća linija koda koristi „for“ petlju kako bi iterirala kroz sve datoteke u specificiranom direktoriju s određenom ekstenzijom (.jpg). Za to je potrebna putanja do direktorija („savedImagesPath“ – standardni, nije odabrano od strane korisnika) i ekstenzija koja označava tip datoteke.

Prva stvar koja se izvodi u toj petlji je ispis imena datoteke koja se trenutno obrađuje u konzolu, koristeći „print“ funkciju („format“ se koristi kako bi mogao ljepše formatirati ispis). Nakon toga se u varijablu imena „image“ učitava, pomoću dliba, slika (koja je RGB – ne sivih tonova) koja kao argument zahtijeva put do te slike (a koji je dobiven u „for“ petlji – koristim samo „f“ kao taj argument pošto je to moguće u pythonu). U varijablu „detectedFaces“, „faceDetector“ sada vraća koordinate detektiranih lica. Parametri za poziv su slika nad kojom se detektira i povećanje slike (eng. *upsampling*), u smislu da, ako se prosljedi 1, slika će se povećati jednom. Pošto za ovaj dio programa nije toliko bitna brzina izvođenja, to sam mogao i povećati na 2 ili 3 ali pošto su sve detekcije bile točne, nisam vidio potrebu za time. Nakon toga se pomoću funkcije „print“ u konzolu ispisuje broj detektiranih lica.

57. linijom koda aplikacija ulazi u još jednu „for“ petlju koja iterira kroz „detectedFaces“ tako da mogu obraditi svako detektirano lice. Prvo što se izvodi u toj petlji je spremanje specifičnih dijelova slike („image) u varijablu „croppedImage“ zato da bih detektirana lica mogao spremiti u mapu „detected_faces“ kasnije. To napravim tako da iz originalne slike izrežem specifične dijelove pomoću koordinata koje sam dobio preko „faceDetector“. Lice dobijem tako da izvučem gornju koordinatu lica, te izvučem duljinu lica na slici koristeći malo matematike. Oduzeo sam od najgornje koordinate lica najdonju koordinatu lica i to sam pretvorio u apsolutnu vrijednost koristeći „abs“. Istu stvar sam napravio s lijevom i desnom koordinatom te sam tako efektivno odredio 2 dimenzije slike. Na sljedećoj liniji koda sam to

lice spremio u trenutni aktivni direktorij i inkrementirao vrijednost varijable „faceNumber“ koja tu služi samo kao brojač detektiranih lica – pomaže pri imenovanju.

Na liniji koda nakon toga sam u varijablu „shape“, koristeći „shapePredictor“, spremio detekciju cijelog objekta (eng. *full object detection*) što su zapravo karakteristične točke lica – to se postiže tako da se koodrinate tog lica proslijede funkciji zajedno sa slikom na kojoj se ono nalazi. Sljedeća linija koda koristi funkciju „compute_face_descriptor“ koja uzima kao parametre sliku i taj „full_object_detection“ kako bi to konvertirala u 128-dimenzionalni deskriptor lica (eng. *face descriptor* - opis) i spremila ga u varijablu „faceDescriptor“. Ovo je potrebno kako bih mogao precizno izračunati euklidsku udaljenost tj. sličnost između lica.

Nakon toga se taj deskriptor sprema u već prije definirano polje koristeći funkciju „append“ – nadodaj. Kao napomena: koristio sam numpy funkciju „asarray“ da spremim taj deskriptor u polje – radi usporedbe. Preostalo je još mijenjanje trenutnog aktivnog direktorija na staru putanju, „savedImagesPath“ i „else“ dio. Taj dio se aktivira ako je u prosljeđen string koji zapravo sadrži putanju do nekog direktorija u kojem korisnik želi detektirati lica. Taj dio je potpuno isti kao i u „if“ dijelu koda, samo što je kod prve „for“ petlje umjesto „savedImagesPath“ korištena varijabla koja je prosljeđena kao parametar funkcije „compareFaces“ – da bi usporedila lica u zadanome direktoriju.

Zadnji dio ove funkcije se koristi samo za određivanje sličnosti uz pomoć euklidske distance (eng. *euclidean distance*). Opisati ću samo neke dijelove zato jer je ovo osnovna matematika. Dio započinje s „for“ petljom koja se vrti broj puta koji je jednak broju spremljenih deskriptora lica. „numpy.linalg.norm“ je funkcija koja vraća euklidsku distancu, u ovom slučaju, između 2 lica i sprema ju u varijablu istog imena. Nakon toga se vrši par „if“ provjera i malo matematike, no ono što je zapravo bitno je da sam stavio prag za usporedbu lica. Taj prag iznosi 0.6 i ako je razlika između 2 lica veća od 0.6 onda se smatra da ta dva lica ne pripadaju istoj osobi, a ako je manja onda bi trebalo pripadati istoj osobi. Taj prag se može još malo smanjiti ili povećati, no 0.6 je dobra mjera. „round“ je funkcija koja samo zaokružuje specificirani broj na zadani broj decimala. „math.pow“ je funkcija koja uzima kao parametre 2 broja i vraća broj koji je jednak prvom parametru na potenciju drugog parametra. Na samom kraju ostaje funkcija „cv2.destroyAllWindows“ koja briše sve prije kreirane prozore.

5. Zaključak

U ovome radu napravio sam jednu malenu aplikaciju koja omogućuje korisniku da učita videozapis na kojem kasnije može detektirati ljudska lica, spremiti ih te usporediti ih. Aplikacija je napravljena uz pomoć Visual Studio Code tekstualnog editora koji pruža korisnicima mnogo mogućnosti (npr. syntax highlighting, debugging itd.). Programski jezik koji sam koristio je python, programski jezik visoke razine koji olakšava programiranje grafičkih sučelja i koji je jedan od najvažnijih jezika današnjice u polju umjetne inteligencije. Svrha rada je pokazati razumijevanje programskih jezika i njihovih biblioteki te pokazati koliko je zapravo kompleksno sve što se događa u pozadini.

Popis literature

[1] https://www.britannica.com/science/face-anatomy

[2] https://design.tutsplus.com/tutorials/human-anatomy-fundamentals-basics-of-the-face--cms-20417

[3] https://www.tutorialspoint.com/python/tk_frame.htm

[4] https://www.geeksforgeeks.org/python-gui-tkinter/

[5] https://docs.python.org/3/library/glob.html

[6] https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf

[7] https://github.com/jrosebr1/imutils/blob/master/imutils/video/filevideostream.py

[8] http://dlib.net/python/index.html

[9] https://github.com/ageitgey/face_recognition/wiki/Calculating-Accuracy-as-a-Percentage

Popis slika

Popis tablica