

Prepoznavanje lica koristeći Haar Cascade algoritam i identifikacija pomoću Multi-Layered Perceptron

Stefani Kecman

Projekt iz predmeta

Digitalno procesiranje signala



Univerzitet u Sarajevu

Elektrotehnički fakultet

Odsjek za Računarstvo i informatiku

2022. godine

Bazni kod HCC

Modifikacije koda (25.05.2022)

Analiza koda

Rezultati izvršavanja koda

Nedostaci ovog rješenja

Modifikacije koda (27.05.2022)

Glavna funkcija:

Solucija za detekciju nosa

Solucija za detekciju usta

Rezultati modificiranog koda na primjerima slika sa više lica:

Uvod u neuronske mreže

Opći princip rada

Multi-Layered Perceptron

Prednosti MLP

Nedostaci MLP

Kod za prepoznavanje i identifikaciju lica

Analiza koda

Uvodne naredbe

Funkcija za konverziju slike u vektor

Funkcija za izdvajanje lica na slici

Treniranje sistema

Testiranje na novom setu slika

Procjena tačnosti rezultata

Izvještaj o uspješnosti treniranja sistema na čitavim slikama

Izvještaj o uspješnosti treniranja sistema na slikama sa izdvojenim licem:

Analiza performansi MLP za identifikaciju ličnosti

Haar Cascade algoritam za detekciju lica

Bazni kod HCC

Kod je preuzet sa stranice:
https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html

Da bi algoritam radio ispravno, potrebno je da prethodno bude treniran na velikom broju slika koje sadrže lice (pozitivne slike) i slikama bez lica na njima (negativne slike). Ulazne slike se analiziraju i računaju se Haar svojstva (“haar features”). Pri proračunu, koriste se integrirane slike, kao što je objašnjeno u seminarskom radu. Za distinkciju haar svojstava koja zaista prikazuju dio lica od onih koji to ne čine, koriste se klasifikatori.

OpenCV nudi mogućnost korištenja već istreniranih klasifikatora. Oni se nalaze u XML formatu.

```
import numpy as np
import cv2

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')

img = cv2.imread('imeslike.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

U prethodnom odjeljku su se iskoristile dvije biblioteke: **numpy** i **cv2**. **numpy** je biblioteka Python programskog jezika koja nudi različite višedimenzionalne kolekcije podataka i matematičke operacije nad njima. **cv2** je biblioteka koja povezuje OpenCV sa tekućim programom.

U ovom primjeru se koriste dva kaskadna klasifikatora. Prvi je klasifikator za detekciju lica (imenovani **face_cascade**), učitan putem xml fajla: "haarcascade_frontalface_default.xml". Drugi je klasifikator za detekciju očiju (imenovani **eye_cascade**) , učitan preko fajla: "imeslike.jpg".

Slika se učitava u "img" varijablu. U "gray" varijablu se smješta ista slika, ali u grayscale varijanti. Inače, cvtColor je metoda za konverziju boje na slici, čiji tonovi odgovaraju pretrazi slike za Haar svojstva¹.

```
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
```

Funkcija **detectMultiScale** se koristi za određivanje područja lica. U ovom slučaju prima tri parametra, što je i najčešći slučaj. To su slika koju analizira, faktor skaliranja i minimalan broj susjeda². **Faktor skaliranja** predstavlja veličinu na koju se smanjuje slika pri svakoj skali. **Minimalan broj susjeda** je broj susjeda koje trenutno pravougaonik u opticaju treba imati, da bi se zadržao u razmatranju.

```
for (x,y,w,h) in faces:  
    img = cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0),2)  
    roi_gray = gray[y:y+h, x:x+w]  
    roi_color = img[y:y+h, x:x+w]  
    eyes = eye_cascade.detectMultiScale(roi_gray)  
    for (ex,ey,ew,eh) in eyes:  
        cv2.rectangle(roi_color, (ex,ey), (ex+ew,ey+eh), (0,255,0),2)
```

U prethodnom isječku koda se nalaze ugniježdene petlje. Vanjska petlja prolazi kroz faces (kolekcija pravougaonika koje je vratila funkcija detectMultiScale), pri čemu četvorka (x, y, w, h) označavaju sljedeće parametre:

x- x koordinata početne tačke duži, odnosno gornje lijeve tačke pravougaonika.

y- y koordinata početne tačke duži, odnosno gornje lijeve tačke pravougaonika

h- dužina visine pravougaonika, tačnije dužina vertikalne ivice

¹https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html

²

<https://stackoverflow.com/questions/36218385/parameters-of-detectmultiscale-in-opencv-using-python#:~:text=detectMultiScale%20function%20is%20used%20to,is%20reduced%20with%20each%20scale.>

w- dužina širine pravougaonika, tačnije dužina horizontalne ivice

Analogno se definiraju i kombinacije varijabli **_x**, **_y**, **_h**, **_w** (pri čemu crta mijenja proizvoljno slovo).

Metoda **rectangle**, iz biblioteke cv2, crta pravougaonike. Prima 5 parametara koji respektivno znače: slika na kojoj se bilježe pravougaonici, početne koordinate pravougaonika, krajnje koordinate pravougaonika, boja ivice (uzeta je (255, 0, 0), što je tuple za plavu boju), debljina ruba izražena u pikselima. Dakle, u ovoj liniji se odvija crtanje pravougaonika, tačnije, kvadrata, oko lica.

Roi - region of interest, je polje od interesa naredne dvije linije.

roi_gray = gray[y:y+h, x:x+w] - ova naredba će izrezati lice iz gray niza. Tačnije, uzet će dio niza red: od y do y+h koordinate, a kolone: od x do x+w.

roi_color = img[y:y+h, x:x+w] - radi isto, samo na originalnoj slici, u boji.

Zatim se poziva funkcija detectMultiScale na kaskade očiju i to samo za razmatrano lice u dotoj iteraciji vanjske for petlje. Za rezultirajuću kolekciju "eyes" se kreira ista for petlja, koja će iscrtavati pravougaonike oko očiju u trenutku razmatranog lica. Metoda za iscrtavanje pravougaonika, koja s poziva i u ugniježdenoj petlji, je već objašnjena.

```
cv2.imshow('img', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Funkcija **imshow** kreira sliku od numpy niza. **waitKey** prikazuje sliku za zadani broj milisekundi ili do pritiska na neku tipku. Kada joj se proslijedi za parametar nula, kao što je u kodu iznad, onda je u pitanju drugi navedeni slučaj. **destroyAllWindows** zatvara sve otvorene prozore. To će se izvršiti po pritisku na tipku, jer slijedi waitKey naredbu.

Modifikacije koda (25.05.2022)

Kod naveden u prethodnom odjeljku je poslužio kao kostur programa koji izrađujem u sklopu ovog projekta.

Bilo je potrebno uvesti nekoliko izmjena. Prvenstveno, implementirati detekciju usta i nosa. S obzirom da već postoji istrenirana kaskada za lociranje usta, što nije slučaj sa detekcijom nosa, preuzeala sam njenu .xml datoteku. Problem pronašlaska nosa na licu sam riješila računskim putem, što će biti objašnjeno u nastavku.

Slijedi prikaz cjelokupnog koda koji se nalazi u odgovarajućem notebook-u:

```
!wget
https://raw.githubusercontent.com/opencv/opencv/4.x/data/haarcascades/haar
cascade_frontalface_default.xml
!wget
https://raw.githubusercontent.com/opencv/opencv/master/data/haarcascades/h
aarcascade_smile.xml
!wget
https://raw.githubusercontent.com/opencv/opencv/3.4/data/haarcascades/haar
cascade_eye.xml

import numpy as np
import cv2
from google.colab.patches import cv2_imshow
from copy import deepcopy
import matplotlib.pyplot as plt

face_cascade= cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
mouth_cascade = cv2.CascadeClassifier('haarcascade_smile.xml')

!wget
https://1vw4gb3u6ymmllev2sp2nlclf-wpengine.netdna-ssl.com/wp-content/upload
s/shutterstock_149962697-946x658.jpg
!mv shutterstock_149962697-946x658.jpg lice1.jpg
```

```

!wget
https://as1.ftcdn.net/v2/jpg/01/97/11/64/1000_F_197116416_hpfttXSoJMvMqU99
n6hGP4xX0ejYa4M7.jpg
!mv 1000_F_197116416_hpfttXSoJMvMqU99n6hGP4xX0ejYa4M7.jpg lice2.jpg
!wget https://live.staticflickr.com/388/19259120435_66df748ee3_b.jpg
!mv 19259120435_66df748ee3_b.jpg lice3.jpg
!wget
https://assets.vogue.com/photos/619e841aad34e792927f7518/master/w_1600,c_1
imit/IndigenousModels-RafaelMartinez-009.jpg
!mv IndigenousModels-RafaelMartinez-009.jpg lice4.jpg

img1 = cv2.imread('lice1.jpg')
img2 = cv2.imread('lice2.jpg')
img3 = cv2.imread('lice3.jpg')
img4=cv2.imread('lice4.jpg')

def face_features_detection(image):
    image_copy = deepcopy(image)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    for (x,y,w,h) in faces:
        image_copy = cv2.rectangle(image_copy, (x,y), (x+w,y+h), (255,0,0),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = image_copy[y:y+h, x:x+w]
        eyes = eye_cascade.detectMultiScale(roi_gray)
        i=0
        for (ex,ey,ew,eh) in eyes:
            i+=1
            if(i==3): break
            cv2.rectangle(roi_color, (ex,ey), (ex+ew,ey+eh), (0,255,0),2)

        mouth = mouth_cascade.detectMultiScale(roi_gray, scaleFactor=1.4,
minNeighbors=5,
minSize=(50, 50), flags=cv2.CASCADE_SCALE_IMAGE)

        j=0;
        for (mx,my,mw,mh) in mouth:
            if (my>(ey+eh)):
                mouth = [mouth[j]]

```

```

        break
    j+=1
for (mx,my,mw,mh) in mouth:
    cv2.rectangle(roi_color, (mx,my), (mx+mw,my+mh), (0,0,255),2)

(ex1,ey1,ew1,eh1) = eyes[0]
(ex2,ey2,ew2,eh2) = eyes[1]
(mx1,my1,mw1,mh1) = mouth[0]
nx = min(ex1 + ew1 + 2, ex2 + ew2 + 2)
ny = min(ey1 + eh1 // 2, ey2 + eh2 // 2)
ex = min(ex1, ex2)
ex_max = max(ex1, ex2)
if (ex == ex1):
    ew = ew1
else:
    ew = ew2
nw = np.abs(ex + ew - ex_max)
nh = np.abs(ny - my1)
cv2.rectangle(roi_color, (nx, ny), (nx+nw,ny+nh-10), (255,0,255),2)

return image_copy

```

```

l1_oznaceno=face_features_detection(img1)
l2_oznaceno=face_features_detection(img2)
l3_oznaceno=face_features_detection(img3)
l4_oznaceno=face_features_detection(img4)

cv2_imshow(l1_oznaceno)
cv2_imshow(l2_oznaceno)
cv2_imshow(l3_oznaceno)
cv2_imshow(l4_oznaceno)

```

Analiza koda

Najnovija verzija koda, u cjelini, je dostupna u notebook-u:

https://colab.research.google.com/drive/1rbmuA_ERz7JCrcY9BQaCAzt8Zy7UPEk5
koji se nalazi u drive folderu podijeljenim s Vama.

Na početku je potrebno importovati potrebne kaskade. Na internetu sam pronašla dostupne kaskade za detekciju lica, očiju i usta. Importujem ih naredbon **!wget**, nakon čega slijedi link odgovarajućeg fajla.

```
!wget
https://raw.githubusercontent.com/opencv/opencv/4.x/data/haarcascades/haar
cascade_frontalface_default.xml
!wget
https://raw.githubusercontent.com/opencv/opencv/master/data/haarcascades/h
aarcascade_smile.xml
!wget
https://raw.githubusercontent.com/opencv/opencv/3.4/data/haarcascades/haar
cascade_eye.xml
```

Uključujem potrebne biblioteke:

```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
from copy import deepcopy
import matplotlib.pyplot as plt
```

numpy biblioteka koja omogućuje manipulaciju višedimenzionalnim nizovima.

cv2 povezuje OpenCV sa programom.

cv2_imshow se preuzima iz `google.colab.patches` i naredba je za prikaz proslijeđene joj slike.

deepcopy je naredba preuzeta iz `copy` koja se koristi za pravljenje duboke kopije objekta.

matplotlib.pyplot biblioteka je učitana u slučaju potrebe za prikazivanjem slike na grafiku.

```
face_cascade= cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
mouth_cascade = cv2.CascadeClassifier('haarcascade_smile.xml')
```

U ovom odjeljku sam definirala kaskadne klasifikatore varijablama face_cascade, eye_cascade, mouth_cascade svrsishodnim imenima. To se postiže naredbom cv2.CascadeClassifier('ime.xml'), pri čemu je "ime" naziv potrebnog .xml fajla u našem repozitoriju.

```
!wget
https://1vw4gb3u6ymm1ev2sp2n1cxf-wpengine.netdna-ssl.com/wp-content/uploads/shutterstock_149962697-946x658.jpg
!mv shutterstock_149962697-946x658.jpg lice1.jpg
!wget
https://as1.ftcdn.net/v2/jpg/01/97/11/64/1000_F_197116416_hpfttXSoJMvMqU99n6hGP4xx0ejYa4M7.jpg
!mv 1000_F_197116416_hpfttXSoJMvMqU99n6hGP4xx0ejYa4M7.jpg lice2.jpg
!wget https://live.staticflickr.com/388/19259120435_66df748ee3_b.jpg
!mv 19259120435_66df748ee3_b.jpg lice3.jpg
!wget
https://assets.vogue.com/photos/619e841aad34e792927f7518/master/w_1600,c_limit/IndigenousModels-RafaelMartinez-009.jpg
!mv IndigenousModels-RafaelMartinez-009.jpg lice4.jpg
```

Prethodio je kod za učitavanje slika za prvobitno testiranje koda sa interneta. Koriste se naredbe **!wget** i **!mv**, pri čemu je prva već objašnjena, a druga služi za premještanje te slike u fajl sa imenom koje slijedi u nastavku. Slike respektivno prikazuju bjelkinju, afroamerikanca, azijatkinju i djevojku sa porijeklom američkih domorodaca. Ovo je površan test, a služio je samo za indikaciju da li, bez detaljnijeg testiranja, kod radi na licima sa različitim histogramima kože i drukčijim izgledom facijalnih karakteristika.

```
img1 = cv2.imread('lice1.jpg')
img2 = cv2.imread('lice2.jpg')
img3 = cv2.imread('lice3.jpg')
img4=cv2.imread('lice4.jpg')
```

Te fotografije sam potom učitala u varijable sa kojima će raditi u nastavku.

Glavna funkcija

```
def face_features_detection(image):
    image_copy = deepcopy(image)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

U prethodnom dijelu sam napravila duboku kopiju proslijedene slike (naredbom `deepcopy` (slika)) i grayscale sliku originala. Kopija slike je potrebna da bi se izbjeglo mijenjanje izvorne slike i omogućila njena ponovna upotreba.

```
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
```

Uzimam kolekciju lica koju vraća funkcija `detectMultiScale`, primijenjena na grayscale fotografiju. Ova funkcija je objašnjena u prvom dijelu ovog dokumenta. Ostali parametri koji su joj proslijedeni su preporučeni.

```
for (x,y,w,h) in faces:
    image_copy = cv2.rectangle(image_copy, (x,y), (x+w,y+h), (255,0,0), 2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = image_copy[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)
    i=0
    for (ex,ey,ew,eh) in eyes:
        i+=1
        if(i==3): break
        cv2.rectangle(roi_color, (ex,ey), (ex+ew,ey+eh), (0,255,0), 2)
```

Navedeni dio je ostao nepromijenjen u odnosu na bazni kod, koji je već objašnjen.

```
mouth = mouth_cascade.detectMultiScale(roi_gray, scaleFactor=1.4,
minNeighbors=5,
minSize=(50, 50), flags=cv2.CASCADE_SCALE_IMAGE)
```

Unutar “glavne” for petlje sam dodala i varijablu za primanje kolekcije usta, koja se prosljeđuje kao rezultat funkcije detectMultiScale. Kao i ranije, proslijedjeni parametri su preporučeni.

```
j=0;  
for (mx,my,mw,mh) in mouth:  
    if (my>(ey+eh)):  
        mouth = [mouth[j]]  
        break  
    j+=1  
for (mx,my,mw,mh) in mouth:  
    cv2.rectangle(roi_color, (mx,my), (mx+mw,my+mh), (0,0,255), 2)
```

Testiranjem ovog dijela koda sam uočila problem da klasifikatori ponude višebrojna područja označena kao da su usta. Da bih taj problem ručno riješila, vodila sam se logikom da, za svaki element kolekcije mouth ispitujem da li vrijedi da je $my > (ey + eh)$, odnosno da li se naznačena usta nalaze ispod očiju. Ukoliko je to slučaj, prihvatom prvo takvo rješenje.

```
(ex1,ey1,ew1,eh1) = eyes[0]  
(ex2,ey2,ew2,eh2) = eyes[1]
```

Pošto se u eyes kolekciji nalazi par očiju, definiram prvo i drugo oko.

```
(mx1,my1,mw1,mh1) = mouth[0]
```

Mouth je idalje kolekcija, sa jednim članom, tako da parametrima ($mx1, my1, mw1, mh1$) prosljeđujem odgovarajuće vrijednosti prvog i jedinog člana mouth kolekcije.

U nastavku sam, zbog nedostatka kaskada za detekciju nosa, improvizirala pronalaženje nosa na slici. Isječak koda za detekciju nosa predstavlja vlastiti kod koji nije zasnovan na vanjskim izvorima. Vodila sam se logikom da bi se obilježena

površina nosa trebala nalaziti između dva oka, počevši od sredine očiju i prostirati se do ispod njih, a iznad usta.

```
nx = min(ex1 + ew1 + 2, ex2 + ew2 + 2)
ny = min(ey1 + eh1 // 2, ey2 + eh2 // 2)
```

Na ovom mjestu sam određivala x i y koordinate krajnje lijeve tačke pravougaonika za nos. Uzimala sam da je to za 2 piksela pomjerena gornja desna tačka lijevog oka. Funkcija min služi da se osiguram da će to zaista biti lijevo oko (na slici, a desno oko te osobe u stvarnosti), da bi se izbjegao slučaj crtanja nosa nakon desnog oka i slične situacije.

```
ex = min(ex1, ex2)
ex_max = max(ex1, ex2)

if (ex == ex1):
    ew = ew1
else:
    ew = ew2
```

ex predstavlja x koordinatu gornje lijeve tačke lijevog oka. **ex_max** je x koordinata desnog oka. **ew** je širina lijevog oka.

```
nw = np.abs(ex + ew - ex_max)
nh = np.abs(ny - my1)
cv2.rectangle(roi_color, (nx, ny), (nx+nw, ny+nh-10), (255, 0, 255), 2)
```

nw je širina nosa i može se dobiti kao ex+ew (što daje x koordinatu gornje lijeve tačke pravougaonika za nos) -ex_max što oduzima početak drugog, desnog oka. **nh** je visina nosa i može se izračunati kao razlika između y koordinate početka nosa i y koordinate početka usta. Uzimajući ove parametre u obzir, crta se pravougaonik koji označava nos.

```
    return image_copy
```

Pošto su se sve promjene, tačnije obilježavanja, vršila na kopiji slike, ona se vraća iz funkcije.

Nastavak programa:

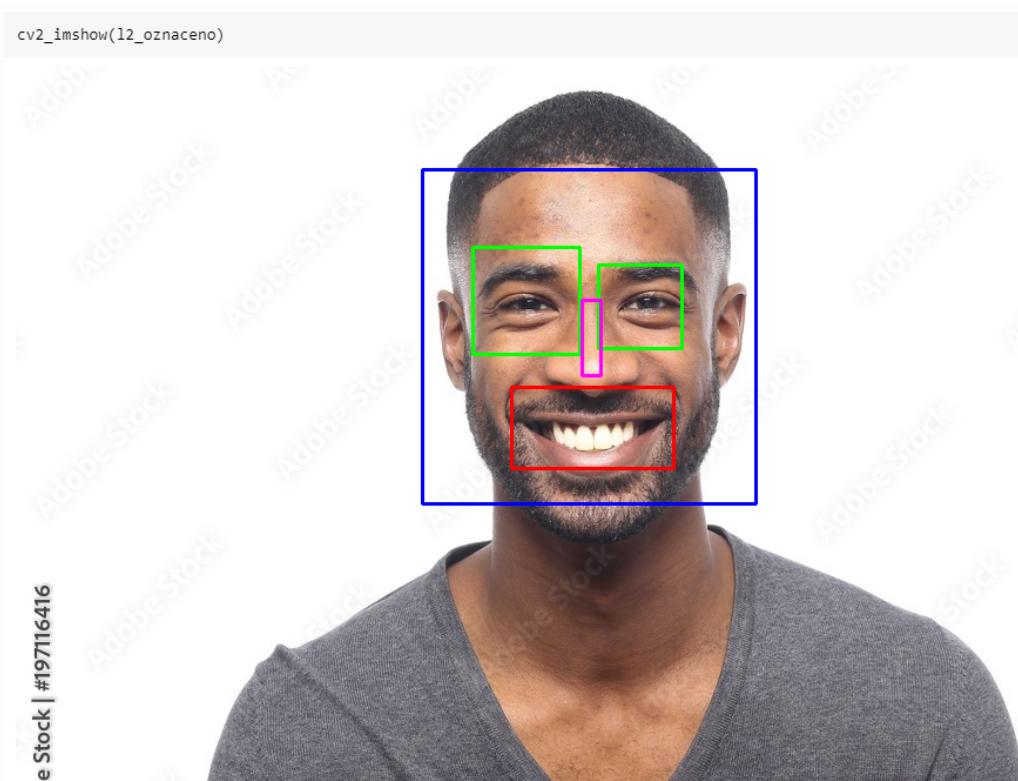
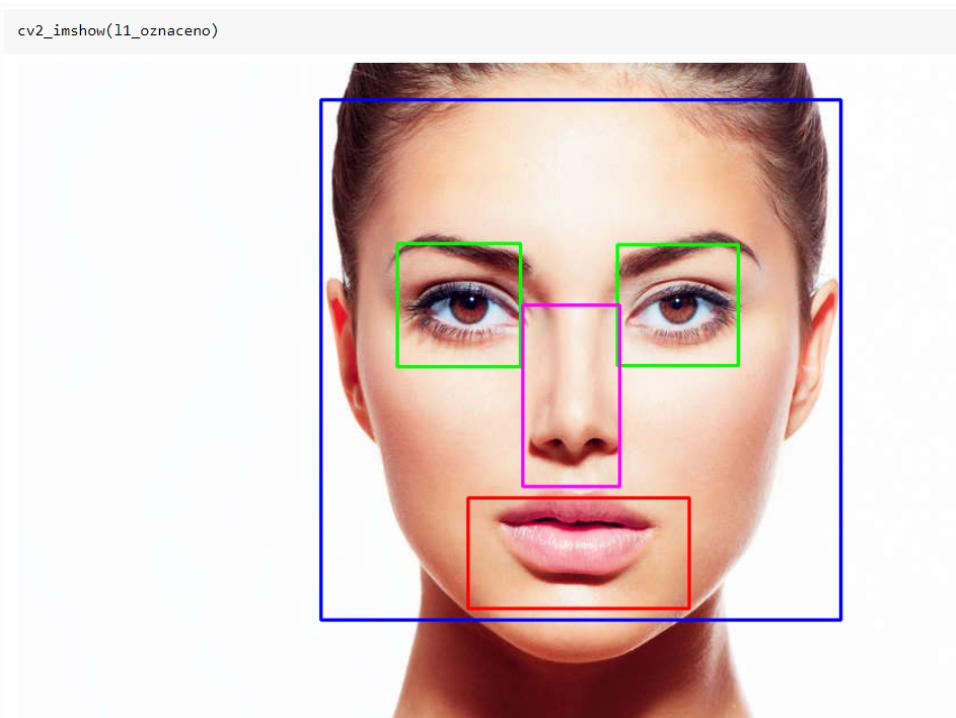
```
l1_oznaceno=face_features_detection(img1)
l2_oznaceno=face_features_detection(img2)
l3_oznaceno=face_features_detection(img3)
l4_oznaceno=face_features_detection(img4)
```

Pozivam maločas objašnjenu funkciju na sve četiri testne fotografije respektivno.

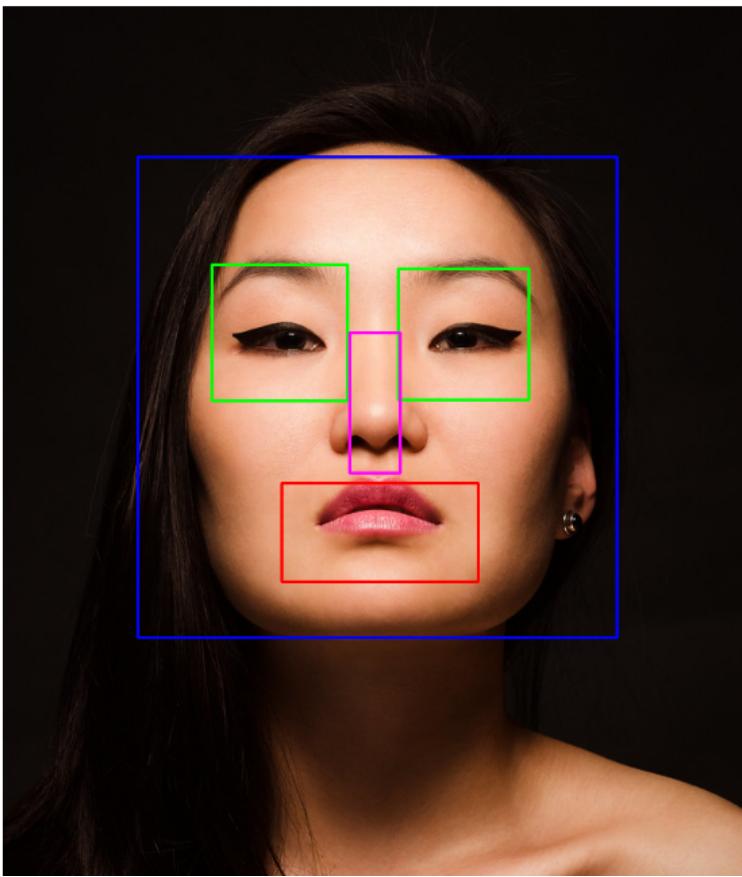
```
cv2_imshow(l1_oznaceno)
cv2_imshow(l2_oznaceno)
cv2_imshow(l3_oznaceno)
cv2_imshow(l4_oznaceno)
```

Ovim naredbama prikazujem slike na kojima su obilježeni potrebni elementi.

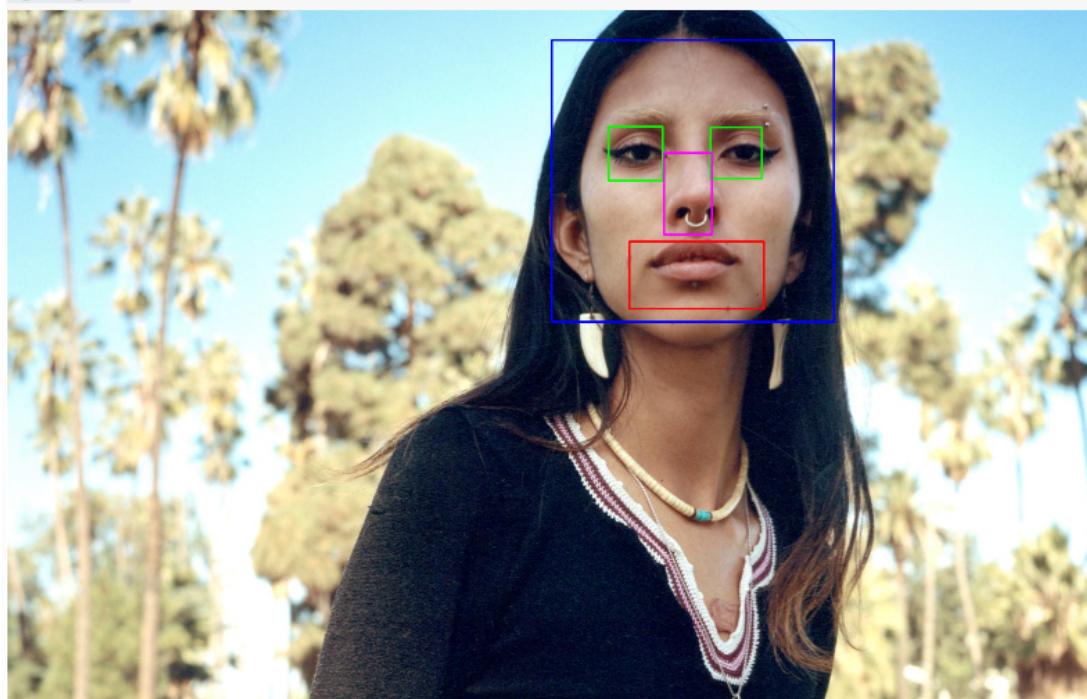
Rezultati izvršavanja koda



```
cv2_imshow(13_oznaceno)
```



```
cv2_imshow(14_oznaceno)
```



Nedostaci ovog rješenja

Primijetila sam da programski kod iz prethodnog odjeljka ne daje zadovoljavajuće rezultate ukoliko se primjeni na slikama na kojima se nalazi više osoba, bez obzira što su im lica frontalno orijentirana na fotografiji.

Naime, lica i oči bi se korektno detektovale i treba imati na umu da se njihova detekcija vrši preko klasifikatora i naredbi preuzetih sa baznog koda objašnjеног na samom početku ovog dokumenta. Dakle, njihov uspjeh je bio očekivan.

Problem je nastaje sa klasifikatorom usta, koji na slikama sa umanjenim licima ponekad detektuje usta, a ponekad ne. Čim se usta ne detektuju, nemoguće je izračunati njih vrijednost za nos, te se ni on ne može odrediti i nacrtati.

U međuvremenu se, vjerovatno, ažurirao folder sa HCC kaskadama, što je za posljedicu imalo probleme pri radu sa posljednjom slikom (native american djevojka).

Srećom, dokumentovala sam sliku u periodu kada se uspješno izvršavala

Modifikacije koda (27.05.2022)

Navedene nedostatke prethodne verzije koda sam riješila računski, izmijenivši glavnu funkciju. Slijedi prikaz nove verzije glavne funkcije, uz objašnjene promjene u kodu.

Glavna funkcija:

```
def face_features_detection(image, sc=1.3, nn=5):
    image_copy = deepcopy(image)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, sc, nn)

    for (x,y,w,h) in faces:
        image_copy = cv2.rectangle(image_copy, (x,y), (x+w,y+h), (255,0,0), 2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = image_copy[y:y+h, x:x+w]
        eyes = eye_cascade.detectMultiScale(roi_gray)
        i=0
        for (ex,ey,ew,eh) in eyes:
            i+=1
            if(i==3): break
            cv2.rectangle(roi_color, (ex,ey), (ex+ew,ey+eh), (0,255,0), 2)

            mouth = mouth_cascade.detectMultiScale(roi_gray, scaleFactor=1.4,
minNeighbors=5,
            minSize=(50, 50), flags=cv2.CASCADE_SCALE_IMAGE)

        j=0;
        for (mx,my,mw,mh) in mouth:
            if (my>(ey+eh)):
                mouth = [mouth[j]]
                break
            j+=1
        for (mx,my,mw,mh) in mouth:
```

```

cv2.rectangle(roi_color, (mx,my) , (mx+mw,my+mh) , (0,0,255) , 2)
#if len(mouth) < 1:
# continue
(ex1,ey1,ew1,eh1) = eyes[0]
(ex2,ey2,ew2,eh2) = eyes[1]
if len(mouth)>=1: (mx1,my1,mw1,mh1) = mouth[0]
nx = min(ex1 + ew1 + 2, ex2 + ew2 + 2)
ny = min(ey1 + eh1 // 2, ey2 + eh2 // 2)
ex = min(ex1, ex2)
ex_max = max(ex1, ex2)
if (ex == ex1):
    ew = ew1
else:
    ew = ew2
nw = np.abs(ex + ew - ex_max)

if len(mouth)>=1: nh = np.abs(ny - my1)
else: nh=int(np.abs(ny-3*eh1))

if (len(mouth)==0):
    mx=ex+ew//2
    my=ny+nh+2
    mh=eh1
    mw=ew+nw+5
    cv2.rectangle (roi_color, (mx,my) , (mx+mw,my+mh) , (0,0,255) , 2)
    cv2.rectangle(roi_color, (nx, ny) , (nx+nw,ny+nh-10) , (255,0,255) , 2)

return image_copy

```

Navedeni problemi su riješeni tako što sam napisala poseban metod proračuna vrijednosti nh, u slučaju da nema detektovanih usta. U tom slučaju bih usta “ručno” crtala koristeći se logikom vođenom anatomijom ljudskog lica.

Smatram da postoje bolji i profesionalniji načini rješavanja navedenih situacija, ali je i ova poslužila svrsi i predstavlja moje vlastito rješenje.

Solucija za detekciju nosa

U ranijoj verziji koda, jedina dimenzija koja je pravila problem bila je visina nosa, nh, pošto se računala po formuli $nh = np.abs(ny - my1)$.

Očigledno je da je za proračun nh bila potrebna y koordinata početne tačke pravougaonika za usta.

Izmjena koju sam uvela je sljedeća:

```
if len(mouth)>=1: nh = np.abs(ny - my1)
else: nh=int(np.abs(ny-3*eh1))
```

Ovo znači da u slučaju detektovanih usta, nh se računa po formuli od ranije, dok u drugim slučajevima koristi cjelobrojnu vrijednost razlike ny koordinate i trostrukе vrijednosti visine lijevog oka na slici.

Lijevo oko je proizvoljno odabранo. Razlika , iz koje se uzima absolutna vrijednost znači da se dužina nosa prostire od y koordinate naniže, pri čemu je ona, oprilike, tri puta duža nego visina područja oka te osobe.

Solucija za detekciju usta

Iako klasifikator nije utvrdio položaj usta na licu, moguće ga je računski odrediti ukoliko znamo položaj ostalih facijalnih značajki.

```
if len(mouth)>=1: (mx1,my1,mw1,mh1) = mouth[0]
```

Ovaj dio koda uvodi dodatnu provjeru da se ne desi indeksiranje niza bez elemenata, što bi bio mouth u slučaju da nisu pronađena nijedna potencijalna usta na slici.

```
if (len(mouth)==0):
    mx=ex+ew//2
    my=ny+nh+2
    mh=eh1
    mw=ew+nw+5
    cv2.rectangle (roi_color, (mx,my), (mx+mw,my+mh), (0,0,255), 2)
```

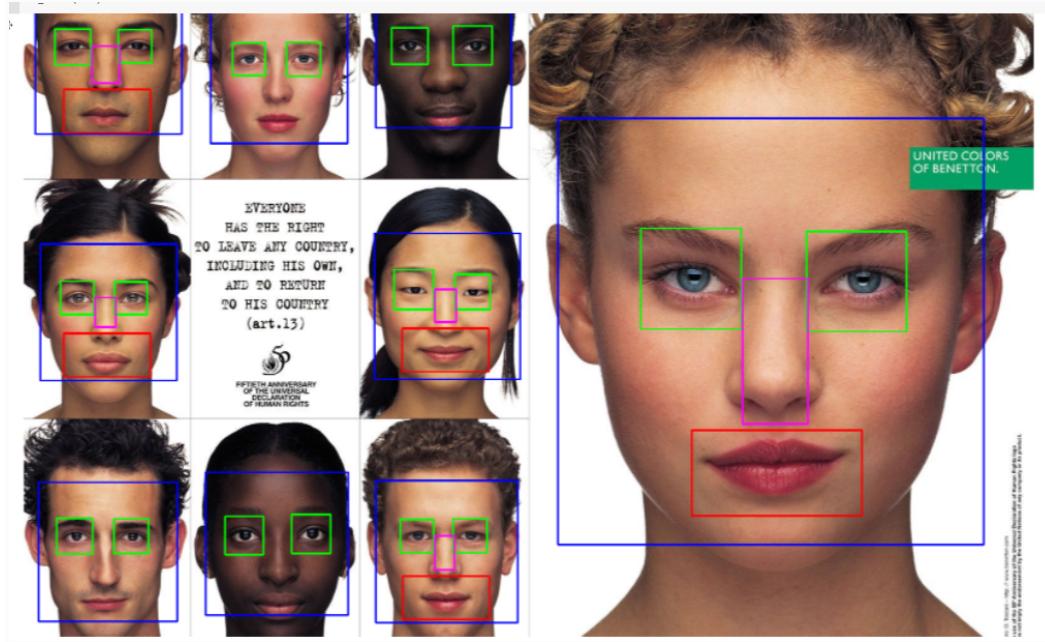
U tom slučaju, potrebne dimenzije računam intuitivno, logičkim rezonovanjem. x koordinata početne tačke pravougaonika oko usta bi se trebala, na apscisi, nalaziti na sredini između početka i kraja regije oka. Y koordinata se na ordinati nalazi ispod nosa ($ny+nh$), pomjerena za 2 piksela niže. Broj piksela je proizvoljno uzeta mala veličina.

Nakon izvršenih proračuna, pozivam naredbu za crtanje pravougaonika sa zadanim parametrima, što će rezultirati označavanjem mjesta gdje se nalaze usta osobe.

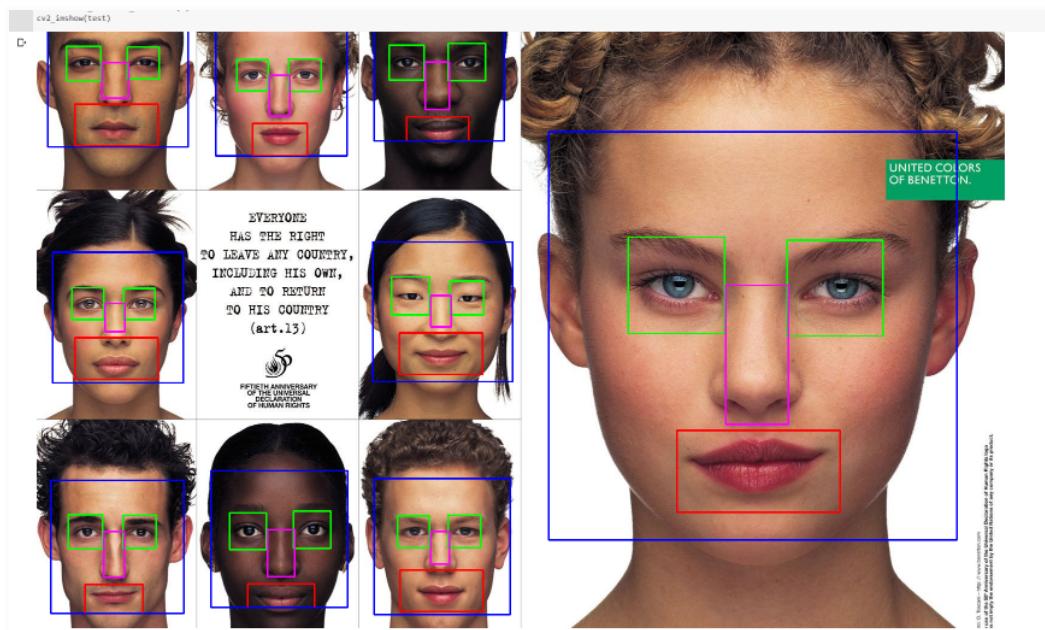
Demonstrirani kod daje izvjesno rješenje problema, ali ne i pretjerano precizno. Naime, u ovom slučaju se ne koriste klasifikatori za detekciju već se ona računa po intuitivno iskonstruiranoj formuli. Samim tim, podložna je nepreciznosti. Dalje, čak je i za ovu formulu potrebna vrijednost dimenzija nosa. Te dimenzije su u svakoj varijanti ručno izračunate, a formula za njihov proračun je još manje precizna ukoliko ne koristim koordinatu usta (koju ne mogu ni koristiti jer u ovoj varijanti je mi nemamo). To bi bili razlozi za varijabilno neprecizne rezultate detekcije usana (i nosa). Međutim, mislim da je i ovaj metod bolji od ranijeg jer barem daje neke rezultate.

Rezultati modificiranog koda na primjerima slika sa više lica:

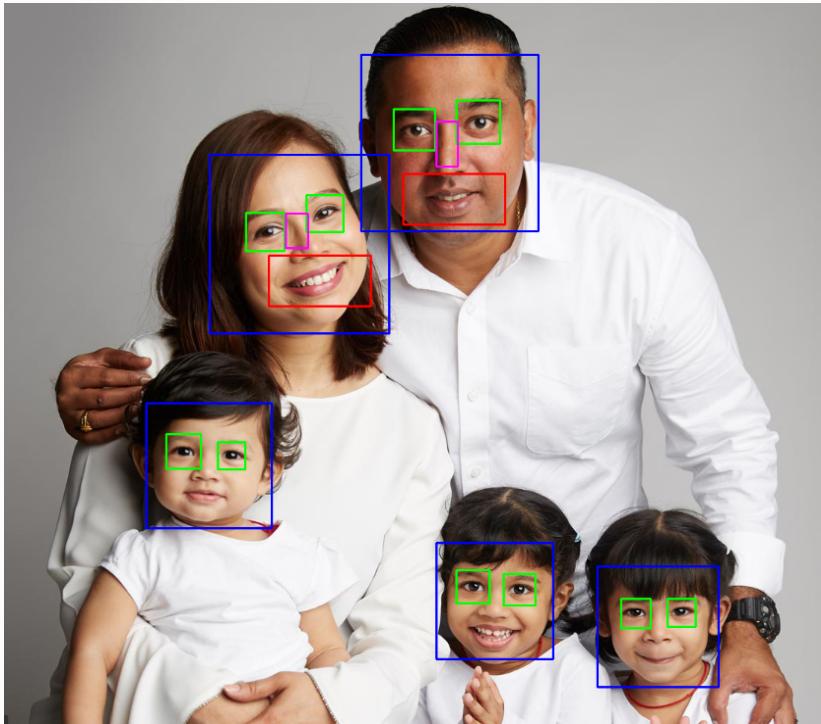
Bez modifikacije:



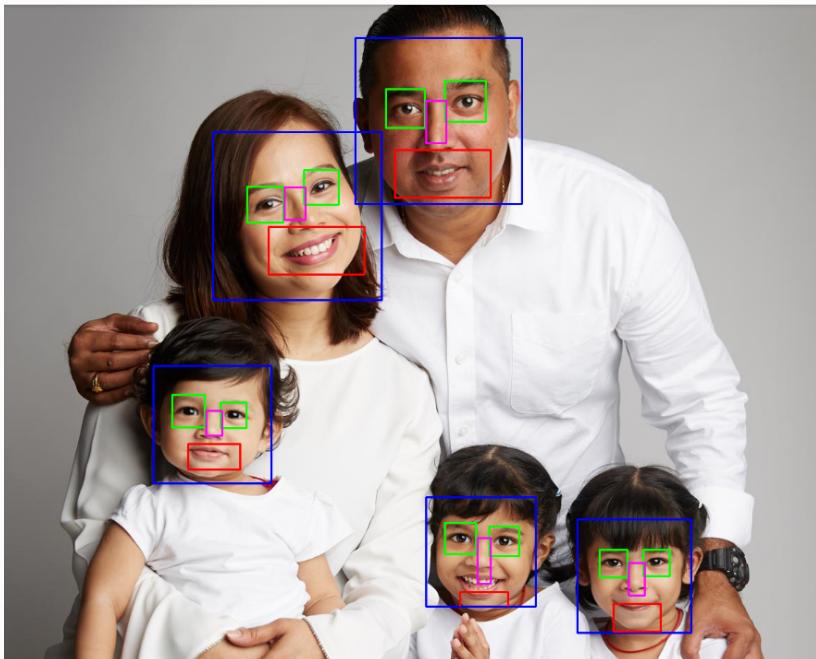
Sa modifikacijom:



Bez modifikacije:



Sa modifikacijom:



Na ovoj fotografiji nos majke nije korektno lociran, uslijed njenog položaja glave. Nos i usta krajanog lijevog djeteta su tačno locirani, dok se odstupanja primijete kod preostala dva djeteta.

Neuronske mreže

Uvod u neuronske mreže

Neuronska mreža je skup algoritama čiji je osnovni zadatak analizirati pozadinske povezanosti seta podataka. Oni se zovu tako s obzirom da proces analize koju izvršavaju nalikuje procesima u ljudskom mozgu.

Generalno govoreći, dio su znanosti **umjetne inteligencije**. Konkretno, ukoliko imaju višeslojnu strukturu (sa dva ili više slojeva), nazivaju se dubokim mrežama te su dio **dubokog učenja**³.

Opći princip rada

Neuron, u neuronskoj mreži, je matematička funkcija koju koristi algoritam. Njen zadatak je da sakuplja i klasificira informacije o slici, što radi po postupku specifičnom za tip neuronske mreže u sklopu koje radi.

Glavni elementi neuronske mreže su sljedeći:

Ulazni sloj predstavlja set podataka koji se prosljeđuje algoritmu.

Procesirajući sloj čine čvorovi i veze među njima, koje principijelno nalikuju neuronima i sinapsama

Izlazni sloj prosljeđuje rezultate analize ostatku programa.

³

<https://www.investopedia.com/terms/n/neuralnetwork.asp#:~:text=A%20neural%20network%20is%20a,organic%20or%20artificial%20in%20nature.>

Multi-Layered Perceptron

Multi-Layered Perceptron (**MLP**) je algoritam superviziranog učenja, koji uči funkciju $f(\cdot) = R^m \rightarrow R^o$ kroz treniranje na setu podataka koji mu se zada. Značenje eksponenata je m broj dimenzija ulaznog seta, a o broj dimenzija izlaznog seta⁴.

Sistemu se daje ulazni set $X = x_1 + x_2 + \dots + x_n$ i target (cilj) y .

Zatim, algoritam dolazi do zaključka o nelinearnoj funkciji aproksimacije putem klasifikacije ili regresije.

Kao što je ranije navedeno, sastoji se od tri glavna dijela. Neuroni imaju za zadatku da transformiraju vrijednosti prethodnog sloja funkcijom težinskih koeficijenata, $w_1 x_1 + w_2 x_2 + \dots + w_n x_n$ i jednim vidom tangensne funkcije⁵.

Prednosti MLP

Osnovna prednost MLP su sposobnost učenja na nelinearnim sistemima. Također, uz pomoć parcijalnog fitovanja, ovaj algoritam može učiti kontinuirano učiti u real-time smislu.

Nedostaci MLP

S druge strane, rezultati izvršavanja MLP su osjetljivi na promjenu faktora skaliranja. Skriveni slojevi sistema mogu imati funkcije gubitaka koje su u slučajevima višestrukosti minimuma nekonveksne. Samim tim, različite težinske vrijednosti daju različite preciznosti pri validaciji. Naposlijetku, potrebno je ručno usaglasiti broj neurona, skrivenih slojeva i iteracija.

⁴ https://scikit-learn.org/stable/modules/neural_networks_supervised.html

⁵ https://scikit-learn.org/stable/modules/neural_networks_supervised.html

Kod za prepoznavanje i identifikaciju lica

U odvojenom notebook-u sam implementirala algoritam za identificiranje osoba. Kao osnovu sam koristila MLP.

Bazu slika za treniranje i validaciju sam preuzela sa internet stranice www.kaggle.com

Baza, koju sam koristila, sam preuzela sa sljedećeg linka: <https://www.kaggle.com/datasets/dansbecker/5-celebrity-faces-dataset>

Iz foldera sam izbacila Mindy Kaling te sistem trenirala sa slikama 4 poznate ličnosti: Elton John, Madonna, Ben Affleck i Jerry Seinfeld.

Analiza koda

Kod u potpunosti je dostupan u notebook-u:
<https://colab.research.google.com/drive/1JteZNpbCWq0GCFm7Z5zmvrOFNxXjJXnI>
koji je podijeljen s Vama u drive folderu.

Uvodne naredbe

Početni korak je preuzimanje baze slika. To se izvršava naredbom:

```
!gdown --id 1vg7_Lw6N_OJ9eoCtHtH5wndKSVjKKplR
```

A u narednoj liniji sam je raspakovala (unzip-ovala):

```
!unzip -qq celeb_ds.zip
```

Isto tako je potrebno i preuzeti haar klasifikator za frontalno lice (njegova primjena će biti objašnjena u nastavku):

```
!wget  
https://raw.githubusercontent.com/opencv/opencv/4.x/data/haarcascades/haar  
cascade_frontalface_default.xml
```

Iskorištene naredbe za navedene svrhe su, redom, !gdown, !unzip i !wget.

```
import os  
import cv2  
import numpy as np  
from google.colab.patches import cv2_imshow  
from copy import deepcopy  
from sklearn.neural_network import MLPClassifier
```

U ovom odjeljku sam pozivala biblioteke i importovala naredbe koje će koristiti u daljoj implementaciji. cv2, np, cv2_imshow i deepcopy su već objašnjene.
os služi za povezivanje koda sa operativnim sistemom. Iz **sklearn.neural_network** (biblioteka koja nudi funkcionalnosti za mašinsko učenje u Pythonu - sklearn, u ovom slučaju konkretno za rad sa neuronskim mrežama) sam uključila u svoj program **MLPClassifier**. Ime mu je svrshodno.

Funkcija za konverziju slike u vektor

Uzela sam kod koji konvertuje sliku u vektor sa stranice: <https://stackoverflow.com/questions/28390614/opencv-hogdescriptor-python> i smjestila ga u funkciju pod nazivom “convert_image_to_vector”, koja na ulazu prima sliku.

```
def convert_image_to_vector(img):
    winSize = (64,64)
    blockSize = (16,16)
    blockStride = (8,8)
    cellSize = (8,8)
    nbins = 9
    derivAperture = 1
    winSigma = 4.
    histogramNormType = 0
    L2HysThreshold = 2.0000000000000001e-01
    gammaCorrection = 0
    nlevels = 64
                                                hog =
cv2.HOGDescriptor(winSize,blockSize,blockStride,cellSize,nbins,derivAperture,winSigma,
histogramNormType,L2HysThreshold,gammaCorrection,nlevels)
    #compute(img[, winStride[, padding[, locations]]]) -> descriptors
    winStride = (8,8)
    padding = (8,8)
```

```
locations = ((10, 20),)
hist = hog.compute(img, winStride, padding, locations)
return hist.transpose().tolist()[0]
```

Analiza pojedinačnih isječaka:

```
winSize = (64, 64)
blockSize = (16, 16)
```

Ove dvije naredbe postavljaju vrijednosti veličine prozora i bloka. **Blok** je broj celija koje se nalaze u jednom dijelu slike.

```
blockStride = (8, 8)
```

Postavlja raskorak pri slidingu bloka.

```
cellSize = (8, 8)
```

Definira dimenzije jedne celije bloka, u pikselima.

```
nbins = 9
```

Postavlja broj smjerova gradijenata unutar jedne celije.

```
derivAperture = 1
winSigma = 4.
histogramNormType = 0
L2HysThreshold = 2.000000000000001e-01
```

```
gammaCorrection = 0
```

Nivo dodatnog osvjetljenja slike (što je veći broj, slika je tamnija sa većim kontrastom).

```
nlevels = 64
```

```
hog = cv2.HOGDescriptor(winSize,blockSize,blockStride,cellSize,nbins,derivAperture,winSigma,
histogramNormType,L2HysThreshold,gammaCorrection,nlevels)
```

Deklarira se HOG deskriptor sa proslijedjenim definiranim parametrima.

```
winStride = (8,8)
```

Raskorak pri slidingu prozora. Algoritam kreće od lijeve gornje tačke i sljedeće tačke uzima krećući se dole-desno (dijagonalno) sa definiranim raskorakom.

```
padding = (8,8)
```

Nije potrebno dodatno objašnjenje

```
locations = ((10,20),)
```

Niz od 2D tačaka na kojima se računaju HOG deskriptori.

```
hist = hog.compute(img,winStride,padding,locations)
```

Deklariram histogram preko naredbe **hog.compute**. Ova funkcija vraća blok HOG deskriptora, izračunatih za datu sliku, uvezvi proslijedene parametre.

Funkcija za izdvajanje lica na slici

Imala sam ideju da testiram performanse MLP na zadanoj bazi slika u slučajevima kada se analiziraju cijele slike i kada se analiziraju samo lica. Iz tog razloga sam napisala funkciju “**select_face**” koja se bazira na sličnom setu naredbi kao dio funkcije iz koda za implementaciju HCC algoritma.

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

def select_face(image, sc=1.3, nn=5):
    image_copy = deepcopy(image)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, sc, nn)

    for (x,y,w,h) in faces:
        roi_color = image_copy[y:y+h, x:x+w]
        return roi_color

    return 0
```

Prvenstveno se inicijalizira kaskada za detekciju lica, nakon čega prelazimo na samu funkciju. U sklopu nje se pravi duboka kopija proslijedene slike.

Kao što se vidi u kodu, postavljene su i dvije defaultne ulazne vrijednosti, parametri detectMultiScale, koji su po default-u preporučene vrijednosti, a ostavljaju prostora da se mijenjaju i samim tim da se analiziraju performanse funkcije pri različitim vrijednostima.

Varijabla gray je grayscale varijanta ulazne slike.

Faces je kolekcija svih lica pronađenih na slici.

Pošto se ponavljaju već objašnjeni dijelovi koda, nisam sada ulazila u detalje. Petljom prolazimo kroz faces i vraćamo prvo lice (izdvojeno) na koje najđemo.

U slučaju da nijedno lice nije detektovano, što se dešavalo pri radu sa bazom slika, funkcija vraća nulu. Ta vrijednost nam je indikator da nijedno lice nije pronađeno, a ispitivat će se kroz dodatne uslove kasnije.

Treniranje sistema

```
ROOT = "/content/data"

train_dir = ROOT + "/train"
val_dir = ROOT + "/val"
```

Povezala sam train_dir i val_dir sa folderima iz baze za treniranje i validaciju.

```
#for directories in os.listdir(train_dir):
#    print(directories)
```

Provjera sadržaja direktorija. Pošto nije od važnosti za ostatak programa, zakomentirala sam je.

```
classes = list(os.listdir(train_dir))
print(classes)
```

U classes učitavam listu klasa slika, koje nose nazine poznate ličnosti koja se nalazi na njenim slikama te ih ispisujem radi provjere ispravnosti učitavanja.

```
clf = MLPClassifier(solver='adam', alpha=1e-5, hidden_layer_sizes=(1764,
500, 250, 100, 50, 1), random_state=1, max_iter=100)
```

Clf je varijabla preko koje će pozivati MLP klasifikator. Njene parametre sam uzela po preporukama s interneta. To su:

Solver - algoritam za težinsku optimizaciju čvorova. Postoji nekoliko različitih algoritama. To su: **lbfgs** (kvazi-Njutnovski optimizator), **sgd** (stohastički spust gradijenta), **adam** (optimizacija stohastičkog spusta baziran na gradijentima).

Hidden_layer_sizes - postavlja broj slojeva i broj neurona koje želimo imati u neuronskoj mreži.

Max_iter - maksimalan broj iteracija

Random_state - definira broj nasumičnih generiranja težinskih inicijalizacija i inicijalizacija sklonosti (bias)

```
x_train = []
y_train = []
```

Inicijalizacija nizova koji će čuvati podatke o slikama na kojima se sistem trenira. X_train je lista slika konvertovanih u vektor. y_train čuva podatke na i-toj poziciji o tome kojem folderu (klasi tj. poznatoj ličnosti) pripada i-ta slika.

```
for klasa in classes:
    current_class = classes[classes.index(klasa)]

    for file in os.listdir(train_dir + "/" + current_class):
        #print(file)
        image = cv2.imread(train_dir + "/" + current_class + "/" + file)
```

Ugniježdenim petljama prolazimo kroz pojedinačne foldere i učitavamo slike iz njih. Pri tome, pojedinačne slike učitavam u file varijablu rasponske for petlje, koristeći funkciju iz biblioteke os, pod nazivom **listdir**. Ova funkcija vraća listu imena svih dokumenata koji se nalaze u direktoriju proslijedene putanje. U svakoj iteraciji, po jednu sliku učitavam u varijablu image, naredbom **cv2.imread()**, koj učitava u varijablu fajl iz proslijedene putanje.

```
face=select_face(image)
        #if      (isinstance(face,      int)==False):      vec      =
convert_image_to_vector(face)
#else:
#  vec = convert_image_to_vector(image)
vec = convert_image_to_vector(image)
```

U face varijablu smještam isječenu sliku, tako da se na njoj nalazi samo detektovano lice. Naredne linije koda su zakomentarisane, a njihova svrha slijedi u nastavku.

Napomena: indentacija u drugoj liniji se poremetila uslijed kopiranja teksta, takva nije u izvornom kodu.

Testirat će uspješnost treniranja sistema u dva slučaja, kada se sistem trenira na čitavoj slici i, u drugom slučaju, samo na izdvojenom licu sa slike, i porediti preciznost.

U slučaju da lice nije pronađeno, funkcija select_face vraća nulu. Stoga u if uslovu ispitujem da li je vraćena vrijednost int.

Ukoliko jeste, to znači da u varijablu vec trebamo proslijediti, vektorski konvertovanu, čitavu sliku. Da nema ove naredbe, desila bi se greška, a i bez nje, to bi značilo da ne šaljemo ništa u vec, čime gubim podatke za treniranje, a to nije od koristi. Isto tako, nisam mogla ovaj uslov ispitivati na jednakost s nulom, zbog slučajeva kada se u faces zaista nalazi slika, što bi također javljalo grešku. Ako nije pronađeno lice, u else dijelu smještam samo izvornu sliku u dalje metode u vec. Ispod toga je ponovljena naredba iz else-a, koja će se zakomentarisati ako se otkomentariše prethodni dio, da ne dođe do duplog spasavanja slika. Također, nije bilo dovoljno samo else zakomentarisati zbog Pythonove osjetljivosti na indentacije.

```
X_train.append(vec)
y_train.append(classes.index(klasa))
```

Sta kod naposlijetku smjestila u vec, to dodajem u listu X_train, a u y_train smještam redni broj klase, tj. poznate ličnosti, kojoj slika pripada.

```
print(len(X_train))
```

Radi provjere ispisujem dužinu X_train liste.

```
clf.fit(X_train, y_train)
```

U ovom dijelu treniram sistem, tako što fit-ujem, odnosno uklapam, X_train vrijednosti sa vrijednostima iz y_train.

Testiranje na novom setu slika

Nakon što je sistem istreniran, potrebno je procijeniti njegove performanse na novom setu slika. To će biti slike istih poznatih ličnosti kao i u setu slika za treniranje, ali pojedinačne slike će biti nikad ranije viđene. Algoritam će imati zadatak da, na osnovu dotad naučenog, pokuša ispravno prepoznati osobu na slici.

S obzirom da je kod gotovo identičan ranije objašnjrenom, istaći će samo napravljene razlike.

```
ROOT = "/content/data"

val_dir = ROOT + "/val"

#for directories in os.listdir(train_dir):
# print(directories)

classes = list(os.listdir(val_dir))
print(classes)

X_val = []
y_val = []

x_val = []
y_val = []
```

U ovom dijelu sada umjesto listi X_train i y_train, deklariram X_val i y_val.

```
for klasa in classes:
    current_class = classes[classes.index(klasa)]

    for file in os.listdir(val_dir + "/" + current_class):
        #print(file)
        image = cv2.imread(val_dir + "/" + current_class + "/" + file)
        face=select_face(image)
            #if      (isinstance(face,      int)==False):      vec      =
convert_image_to_vector(face)
        #else:
        #  vec = convert_image_to_vector(image)
```

```
vec = convert_image_to_vector(image)

X_val.append(vec)
y_val.append(classes.index(klasa))
```

Učitavam vrijednosti slika iz foldera za validaciju u X_val te njihove redne brojeve u y_val.

```
print(len(X_val))
```

Radi provjere ispisujem dužinu X_val liste.

```
print(y_pred)
print(y_val)
```

Ispisujem predviđene y vrijednosti (indekse klase kojima je sistem zaključio da ulazni podaci pripadaju) i očekivanih, radi interne evaluacije uspješnosti procjene.

Procjena tačnosti rezultata

Koristila sam `sklearn.metrics.classification_report`⁶ za ispisivanje izvještaja o klasifikacijskim metrikama.

U svoj program sam ga uključila sljedećom naredbom:

```
from sklearn.metrics import classification_report
```

Ispisivanje izvještaja o tome koliko su dobro izvršene procjene:

```
print(classification_report(y_val, y_pred, target_names=classes))
```

⁶ https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

Izvještaj o uspješnosti treniranja sistema na čitavim slikama

Svi izvještaji u ovom odjeljku se odnose na analizu čitavih slika, bez izdvajanja samo lica za analizu.

Pored koda koji Vam je dostupan u notebook-u, slijede slike izvještaja o preciznosti treniranja i validacije ovog sistema. Ako želite, možete preskočiti na 60. stranicu, odakle se nastavlja dokumentacija projekta.

hidden_layer_sizes=(1764, 500, 250, 100, 50, 1):

max_iter=100:

Za program sa parametrima kakvi su navedeni u ranijoj analizi koda, slijedi da je Ben Affleck precizno prepoznao sistem u 25% slučajeva, dok je za ostale ličnosti ta vrijednost 0%. Ovo je loša preciznost.

	precision	recall	f1-score	support
jerry_seinfeld	0.00	0.00	0.00	5
ben_afflek	0.25	1.00	0.40	5
madonna	0.00	0.00	0.00	5
elton_john	0.00	0.00	0.00	5
accuracy			0.25	20
macro avg	0.06	0.25	0.10	20
weighted avg	0.06	0.25	0.10	20

Preciznost i tačnost treniranja sistema se provjerava u nastavku:

```
y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))
```

	precision	recall	f1-score	support
jerry_seinfeld	0.00	0.00	0.00	0
ben_afflek	1.00	0.20	0.33	71
madonna	0.00	0.00	0.00	0
elton_john	0.00	0.00	0.00	0
accuracy			0.20	71
macro avg	0.25	0.05	0.08	71
weighted avg	1.00	0.20	0.33	71

Odavde vidim da je tačnost pri treniranju dostignula 100% za Ben Afflecka, dok je za ostale 0%.

max_iter=150:

```
▶ from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

→      precision    recall   f1-score   support
elton_john      0.00     0.00     0.00      5
jerry_seinfeld   0.25     1.00     0.40      5
madonna         0.00     0.00     0.00      5
ben_afflek       0.00     0.00     0.00      5

accuracy          -        -      0.25     20
macro avg        0.06     0.25     0.10     20
weighted avg     0.06     0.25     0.10     20

▶ y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

      precision    recall   f1-score   support
elton_john      0.00     0.00     0.00      0
jerry_seinfeld   1.00     0.32     0.48     66
madonna         0.16     0.60     0.25      5
ben_afflek       0.00     0.00     0.00      0

accuracy          -        -      0.34     71
macro avg        0.29     0.23     0.18     71
weighted avg     0.94     0.34     0.47     71
```

max_iter=300:

```
▶ from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

→      precision    recall   f1-score   support
elton_john      0.00     0.00     0.00      5
jerry_seinfeld   0.25     1.00     0.40      5
madonna         0.00     0.00     0.00      5
ben_afflek       0.00     0.00     0.00      5

accuracy          -        -      0.25     20
macro avg        0.06     0.25     0.10     20
weighted avg     0.06     0.25     0.10     20
```

```

y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

      precision    recall  f1-score   support

elton_john       0.00     0.00     0.00      0
jerry_seinfeld    1.00     0.32     0.48     66
  madonna       0.16     0.60     0.25      5
ben_afflek        0.00     0.00     0.00      0

accuracy           -         -     0.34     71
macro avg       0.29     0.23     0.18     71
weighted avg     0.94     0.34     0.47     71

```

max_iter=900

```

▶ from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

      precision    recall  f1-score   support

elton_john       0.00     0.00     0.00      5
jerry_seinfeld    0.25     1.00     0.40      5
  madonna       0.00     0.00     0.00      5
ben_afflek        0.00     0.00     0.00      5

accuracy           -         -     0.25     20
macro avg       0.06     0.25     0.10     20
weighted avg     0.06     0.25     0.10     20

```

```

▶ y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

      precision    recall  f1-score   support

elton_john       0.00     0.00     0.00      0
jerry_seinfeld    1.00     0.32     0.48     66
  madonna       0.16     0.60     0.25      5
ben_afflek        0.00     0.00     0.00      0

accuracy           -         -     0.34     71
macro avg       0.29     0.23     0.18     71
weighted avg     0.94     0.34     0.47     71

```

max_iter=1900:

```
[61] from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

      precision    recall  f1-score   support

elton_john       0.00     0.00     0.00      5
jerry_seinfeld   0.25     1.00     0.40      5
madonna          0.00     0.00     0.00      5
ben_afflek        0.00     0.00     0.00      5

accuracy         -         -         0.25     20
macro avg       0.06     0.25     0.10     20
weighted avg     0.06     0.25     0.10     20
```



```
▶ y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

      precision    recall  f1-score   support

elton_john       0.00     0.00     0.00      0
jerry_seinfeld   1.00     0.32     0.48     66
madonna          0.16     0.60     0.25      5
ben_afflek        0.00     0.00     0.00      0

accuracy         -         -         0.34     71
macro avg       0.29     0.23     0.18     71
weighted avg     0.94     0.34     0.47     71
```

hidden_layer_sizes=(1764, 1000, 500, 250, 100, 50, 1):

max_iter=100:

```
▶ from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

      precision    recall  f1-score   support

elton_john       0.00     0.00     0.00      5
jerry_seinfeld   0.00     0.00     0.00      5
madonna          0.25     1.00     0.40      5
ben_afflek        0.00     0.00     0.00      5

accuracy         -         -         0.25     20
macro avg        0.06     0.25     0.10     20
weighted avg     0.06     0.25     0.10     20
```

```
▶ y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

      precision    recall  f1-score   support

elton_john       0.00     0.00     0.00      0
jerry_seinfeld   0.00     0.00     0.00      0
madonna          1.00     0.27     0.42     71
ben_afflek        0.00     0.00     0.00      0

accuracy         -         -         0.27     71
macro avg        0.25     0.07     0.11     71
weighted avg     1.00     0.27     0.42     71
```

max_iter=500:

```
▶ from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

      precision    recall  f1-score   support

elton_john       0.00     0.00     0.00      5
jerry_seinfeld   0.00     0.00     0.00      5
madonna          0.25     1.00     0.40      5
ben_afflek        0.00     0.00     0.00      5

accuracy         -         -         0.25     20
macro avg        0.06     0.25     0.10     20
weighted avg     0.06     0.25     0.10     20
```

```

y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

      precision    recall  f1-score   support

elton_john       0.00     0.00     0.00      0
jerry_seinfeld   0.00     0.00     0.00      0
madonna          1.00     0.27     0.42     71
ben_afflek        0.00     0.00     0.00      0

accuracy           -         -     0.27     71
macro avg       0.25     0.07     0.11     71
weighted avg     1.00     0.27     0.42     71

```

max_iter=1500:

```

from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

      precision    recall  f1-score   support

elton_john       0.00     0.00     0.00      5
jerry_seinfeld   0.00     0.00     0.00      5
madonna          0.25     1.00     0.40      5
ben_afflek        0.00     0.00     0.00      5

accuracy           -         -     0.25     20
macro avg       0.06     0.25     0.10     20
weighted avg     0.06     0.25     0.10     20

```

```

y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

      precision    recall  f1-score   support

elton_john       0.00     0.00     0.00      0
jerry_seinfeld   0.00     0.00     0.00      0
madonna          1.00     0.27     0.42     71
ben_afflek        0.00     0.00     0.00      0

accuracy           -         -     0.27     71
macro avg       0.25     0.07     0.11     71
weighted avg     1.00     0.27     0.42     71

```

```
hidden_layer_sizes=(1764, 1000, 750, 500, 250, 200, 100, 50, 1)
```

max_iter=100:

```
▶ from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

      precision    recall  f1-score   support

elton_john       0.25     1.00     0.40      5
jerry_seinfeld    0.00     0.00     0.00      5
madonna          0.00     0.00     0.00      5
ben_afflek        0.00     0.00     0.00      5

accuracy           -         -     0.25     20
macro avg         0.06     0.25     0.10     20
weighted avg      0.06     0.25     0.10     20
```

```
▶ y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

      precision    recall  f1-score   support

elton_john       1.00     0.24     0.39     71
jerry_seinfeld    0.00     0.00     0.00      0
madonna          0.00     0.00     0.00      0
ben_afflek        0.00     0.00     0.00      0

accuracy           -         -     0.24     71
macro avg         0.25     0.06     0.10     71
weighted avg      1.00     0.24     0.39     71
```

max_iter=500:

```
▶ from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

      precision    recall  f1-score   support

elton_john       0.25     1.00     0.40      5
jerry_seinfeld    0.00     0.00     0.00      5
madonna          0.00     0.00     0.00      5
ben_afflek        0.00     0.00     0.00      5

accuracy           -         -     0.25     20
macro avg         0.06     0.25     0.10     20
weighted avg      0.06     0.25     0.10     20
```

```

y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

      precision    recall  f1-score   support

elton_john       1.00     0.24     0.39      71
jerry_seinfeld    0.00     0.00     0.00       0
madonna          0.00     0.00     0.00       0
ben_afflek        0.00     0.00     0.00       0

accuracy           -         -     0.24      71
macro avg       0.25     0.06     0.10      71
weighted avg     1.00     0.24     0.39      71

```

max_iter=1500:

```

from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

      precision    recall  f1-score   support

elton_john       0.25     1.00     0.40       5
jerry_seinfeld    0.00     0.00     0.00       5
madonna          0.00     0.00     0.00       5
ben_afflek        0.00     0.00     0.00       5

accuracy           -         -     0.25      20
macro avg       0.06     0.25     0.10      20
weighted avg     0.06     0.25     0.10      20

```

hidden_layer_sizes=(1764, 1000, 500, 200, 100, 1)

max_iter=100

```

from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

      precision    recall  f1-score   support

elton_john       0.00     0.00     0.00       5
jerry_seinfeld    0.00     0.00     0.00       5
madonna          0.25     1.00     0.40       5
ben_afflek        0.00     0.00     0.00       5

accuracy           -         -     0.25      20
macro avg       0.06     0.25     0.10      20
weighted avg     0.06     0.25     0.10      20

```

```

y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

 $\Rightarrow$  precision recall f1-score support

elton_john 0.00 0.00 0.00 0
jerry_seinfeld 0.00 0.00 0.00 0
madonna 1.00 0.27 0.42 71
ben_afflek 0.00 0.00 0.00 0

accuracy 0.27 71
macro avg 0.25 0.07 0.11 71
weighted avg 1.00 0.27 0.42 71

```

max_iter=500

```

[173] from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

precision recall f1-score support

elton_john 0.00 0.00 0.00 5
jerry_seinfeld 0.00 0.00 0.00 5
madonna 0.25 1.00 0.40 5
ben_afflek 0.00 0.00 0.00 5

accuracy 0.25 20
macro avg 0.06 0.25 0.10 20
weighted avg 0.06 0.25 0.10 20

```

```

y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

 $\Rightarrow$  precision recall f1-score support

elton_john 0.00 0.00 0.00 0
jerry_seinfeld 0.00 0.00 0.00 0
madonna 1.00 0.27 0.42 71
ben_afflek 0.00 0.00 0.00 0

accuracy 0.27 71
macro avg 0.25 0.07 0.11 71
weighted avg 1.00 0.27 0.42 71

```

max_iter=1500

```
[179] from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

      precision    recall  f1-score   support

elton_john       0.00     0.00     0.00      5
jerry_seinfeld   0.00     0.00     0.00      5
  madonna       0.25     1.00     0.40      5
ben_afflek        0.00     0.00     0.00      5

accuracy          -         -     0.25     20
macro avg       0.06     0.25     0.10     20
weighted avg     0.06     0.25     0.10     20
```



```
y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

      precision    recall  f1-score   support

elton_john       0.00     0.00     0.00      0
jerry_seinfeld   0.00     0.00     0.00      0
  madonna       1.00     0.27     0.42     71
ben_afflek        0.00     0.00     0.00      0

accuracy          -         -     0.27     71
macro avg       0.25     0.07     0.11     71
weighted avg     1.00     0.27     0.42     71
```

hidden_layer_sizes=(1764, 1000, 500, 200, 100, 50, 1),
max_iter=300, random_state=5)

```
[1]: from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))



|                | precision | recall | f1-score | support |
|----------------|-----------|--------|----------|---------|
| elton_john     | 0.00      | 0.00   | 0.00     | 5       |
| jerry_seinfeld | 0.25      | 1.00   | 0.40     | 5       |
| madonna        | 0.00      | 0.00   | 0.00     | 5       |
| ben_afflek     | 0.00      | 0.00   | 0.00     | 5       |
| accuracy       |           |        | 0.25     | 20      |
| macro avg      | 0.06      | 0.25   | 0.10     | 20      |
| weighted avg   | 0.06      | 0.25   | 0.10     | 20      |


```
[2]: y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

	precision	recall	f1-score	support
elton_john	0.00	0.00	0.00	0
jerry_seinfeld	1.00	0.30	0.46	71
madonna	0.00	0.00	0.00	0
ben_afflek	0.00	0.00	0.00	0
accuracy			0.30	71
macro avg	0.25	0.07	0.11	71
weighted avg	1.00	0.30	0.46	71


```


```

Izvještaj o uspješnosti treniranja sistema na slikama sa izdvojenim licem:

hidden_layer_sizes=(1764, 500, 250, 100, 50, 1):

max_iter = 100:

```
▶ from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

→      precision    recall   f1-score   support
elton_john       0.00     0.00     0.00      5
jerry_seinfeld   0.25     1.00     0.40      5
madonna          0.00     0.00     0.00      5
ben_afflek        0.00     0.00     0.00      5

accuracy           0.25      20
macro avg         0.06     0.25     0.10      20
weighted avg      0.06     0.25     0.10      20
```

```
▶ y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

→      precision    recall   f1-score   support
elton_john       0.00     0.00     0.00      0
jerry_seinfeld   1.00     0.30     0.46     70
madonna          0.05     1.00     0.10      1
ben_afflek        0.00     0.00     0.00      0

accuracy           0.31      71
macro avg         0.26     0.33     0.14     71
weighted avg      0.99     0.31     0.46     71
```

max_iter =150:

```
▶ from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

▷          precision    recall  f1-score   support

elton_john      0.00     0.00     0.00      5
jerry_seinfeld   0.25     1.00     0.40      5
madonna         0.00     0.00     0.00      5
ben_afflek       0.00     0.00     0.00      5

accuracy           -         -     0.25     20
macro avg        0.06     0.25     0.10     20
weighted avg     0.06     0.25     0.10     20
```

```
▶ y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

▷          precision    recall  f1-score   support

elton_john      0.00     0.00     0.00      0
jerry_seinfeld   1.00     0.30     0.46     70
madonna         0.05     1.00     0.10      1
ben_afflek       0.00     0.00     0.00      0

accuracy           -         -     0.31     71
macro avg        0.26     0.33     0.14     71
weighted avg     0.99     0.31     0.46     71
```

max_iter=300:

```
▶ from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

→      precision    recall  f1-score   support

elton_john       0.00     0.00     0.00      5
jerry_seinfeld   0.25     1.00     0.40      5
madonna          0.00     0.00     0.00      5
ben_afflek       0.00     0.00     0.00      5

accuracy          —       0.25     0.25      20
macro avg        0.06     0.25     0.10      20
weighted avg     0.06     0.25     0.10      20
```

```
▶ y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

→      precision    recall  f1-score   support

elton_john       0.00     0.00     0.00      0
jerry_seinfeld   1.00     0.30     0.46     70
madonna          0.05     1.00     0.10      1
ben_afflek       0.00     0.00     0.00      0

accuracy          —       0.31     0.31     71
macro avg        0.26     0.33     0.14     71
weighted avg     0.99     0.31     0.46     71
```

max_iter=900:

```
▶ from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

→      precision    recall  f1-score   support

elton_john       0.00     0.00     0.00      5
jerry_seinfeld   0.25     1.00     0.40      5
madonna          0.00     0.00     0.00      5
ben_afflek       0.00     0.00     0.00      5

accuracy          —       0.25     0.25      20
macro avg        0.06     0.25     0.10      20
weighted avg     0.06     0.25     0.10      20
```

```
▶ y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))
```

```
→      precision    recall   f1-score   support

elton_john      0.00     0.00     0.00      0
jerry_seinfeld   1.00     0.30     0.46     70
madonna        0.05     1.00     0.10      1
ben_afflek       0.00     0.00     0.00      0

accuracy          -         -     0.31     71
macro avg       0.26     0.33     0.14     71
weighted avg    0.99     0.31     0.46     71
```

max_iter=1900:

```
▶ from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))
```

```
→      precision    recall   f1-score   support

elton_john      0.00     0.00     0.00      5
jerry_seinfeld   0.25     1.00     0.40      5
madonna        0.00     0.00     0.00      5
ben_afflek       0.00     0.00     0.00      5

accuracy          -         -     0.25     20
macro avg       0.06     0.25     0.10     20
weighted avg    0.06     0.25     0.10     20
```

```
▶ y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))
```

```
→      precision    recall   f1-score   support

elton_john      0.00     0.00     0.00      0
jerry_seinfeld   1.00     0.30     0.46     70
madonna        0.05     1.00     0.10      1
ben_afflek       0.00     0.00     0.00      0

accuracy          -         -     0.31     71
macro avg       0.26     0.33     0.14     71
weighted avg    0.99     0.31     0.46     71
```

hidden_layer_sizes=(1764, 1000, 500, 250, 100, 50, 1):

max_iter=100

```
▶ from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

→      precision    recall  f1-score   support

elton_john       0.00     0.00     0.00      5
jerry_seinfeld   0.00     0.00     0.00      5
madonna          0.25     1.00     0.40      5
ben_afflek        0.00     0.00     0.00      5

accuracy         -         -       0.25     20
macro avg        0.06     0.25     0.10     20
weighted avg     0.06     0.25     0.10     20
```

```
▶ y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

→      precision    recall  f1-score   support

elton_john       0.00     0.00     0.00      0
jerry_seinfeld   0.00     0.00     0.00      0
madonna          1.00     0.27     0.42     71
ben_afflek        0.00     0.00     0.00      0

accuracy         -         -       0.27     71
macro avg        0.25     0.07     0.11     71
weighted avg     1.00     0.27     0.42     71
```

max_iter=500

```
▶ from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

      precision    recall  f1-score   support

elton_john       0.00     0.00     0.00      5
jerry_seinfeld   0.00     0.00     0.00      5
madonna          0.25     1.00     0.40      5
ben_afflek        0.00     0.00     0.00      5

accuracy         -         -       0.25     20
macro avg        0.06     0.25     0.10     20
weighted avg     0.06     0.25     0.10     20
```

```

y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

    precision    recall   f1-score   support
elton_john      0.00     0.00     0.00      0
jerry_seinfeld   0.00     0.00     0.00      0
madonna        1.00     0.27     0.42     71
ben_afflek      0.00     0.00     0.00      0

accuracy          -         -       0.27     71
macro avg       0.25     0.07     0.11     71
weighted avg     1.00     0.27     0.42     71

```

max_iter= 1500

```

from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

    precision    recall   f1-score   support
elton_john      0.00     0.00     0.00      5
jerry_seinfeld   0.00     0.00     0.00      5
madonna        0.25     1.00     0.40      5
ben_afflek      0.00     0.00     0.00      5

accuracy          -         -       0.25     20
macro avg       0.06     0.25     0.10     20
weighted avg     0.06     0.25     0.10     20

```

```

y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

    precision    recall   f1-score   support
elton_john      0.00     0.00     0.00      0
jerry_seinfeld   0.00     0.00     0.00      0
madonna        1.00     0.27     0.42     71
ben_afflek      0.00     0.00     0.00      0

accuracy          -         -       0.27     71
macro avg       0.25     0.07     0.11     71
weighted avg     1.00     0.27     0.42     71

```

`hidden_layer_sizes=(1764, 1000, 750, 500, 250, 200, 100, 50, 1)`

`max_iter=100:`

```
▶ from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

→      precision    recall  f1-score   support

elton_john       0.25     1.00     0.40      5
jerry_seinfeld    0.00     0.00     0.00      5
madonna          0.00     0.00     0.00      5
ben_afflek        0.00     0.00     0.00      5

accuracy           -         -     0.25     20
macro avg       0.06     0.25     0.10     20
weighted avg     0.06     0.25     0.10     20

▶ y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

→      precision    recall  f1-score   support

elton_john       1.00     0.24     0.39     71
jerry_seinfeld    0.00     0.00     0.00      0
madonna          0.00     0.00     0.00      0
ben_afflek        0.00     0.00     0.00      0

accuracy           -         -     0.24     71
macro avg       0.25     0.06     0.10     71
weighted avg     1.00     0.24     0.39     71
```

`max_iter=500:`

```
→      precision    recall  f1-score   support

elton_john       0.25     1.00     0.40      5
jerry_seinfeld    0.00     0.00     0.00      5
madonna          0.00     0.00     0.00      5
ben_afflek        0.00     0.00     0.00      5

accuracy           -         -     0.25     20
macro avg       0.06     0.25     0.10     20
weighted avg     0.06     0.25     0.10     20
```

```

▶ y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

precision    recall   f1-score   support

elton_john      1.00      0.24      0.39      71
jerry_seinfeld   0.00      0.00      0.00       0
madonna          0.00      0.00      0.00       0
ben_afflek       0.00      0.00      0.00       0

accuracy           -         -      0.24      71
macro avg        0.25      0.06      0.10      71
weighted avg     1.00      0.24      0.39      71

```

max_iter=1500:

```

▶ print(classification_report(y_val, y_pred, target_names=classes))

precision    recall   f1-score   support

elton_john      0.25      1.00      0.40       5
jerry_seinfeld   0.00      0.00      0.00       5
madonna          0.00      0.00      0.00       5
ben_afflek       0.00      0.00      0.00       5

accuracy           -         -      0.25      20
macro avg        0.06      0.25      0.10      20
weighted avg     0.06      0.25      0.10      20

```

```

▶ y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

precision    recall   f1-score   support

elton_john      1.00      0.24      0.39      71
jerry_seinfeld   0.00      0.00      0.00       0
madonna          0.00      0.00      0.00       0
ben_afflek       0.00      0.00      0.00       0

accuracy           -         -      0.24      71
macro avg        0.25      0.06      0.10      71
weighted avg     1.00      0.24      0.39      71

```

hidden_layer_sizes=(1764, 1000, 500, 200, 100, 1)

max_iter=100

```
▶ from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

precision    recall   f1-score   support
elton_john      0.00      0.00      0.00       5
jerry_seinfeld   0.00      0.00      0.00       5
madonna          0.25      1.00      0.40       5
ben_afflek        0.00      0.00      0.00       5

accuracy           -         -         -        20
macro avg          0.06      0.25      0.10      20
weighted avg       0.06      0.25      0.10      20
```



```
▶ y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))

precision    recall   f1-score   support
elton_john      0.00      0.00      0.00       0
jerry_seinfeld   0.00      0.00      0.00       0
madonna          1.00      0.27      0.42      71
ben_afflek        0.00      0.00      0.00       0

accuracy           -         -         -        71
macro avg          0.25      0.07      0.11      71
weighted avg       1.00      0.27      0.42      71
```

max_iter=500

```
[153] from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))

precision    recall   f1-score   support
elton_john      0.00      0.00      0.00       5
jerry_seinfeld   0.00      0.00      0.00       5
madonna          0.25      1.00      0.40       5
ben_afflek        0.00      0.00      0.00       5

accuracy           -         -         -        20
macro avg          0.06      0.25      0.10      20
weighted avg       0.06      0.25      0.10      20
```

```
▶ y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))
```

```
↳      precision    recall   f1-score   support

elton_john       0.00     0.00     0.00      0
jerry_seinfeld   0.00     0.00     0.00      0
madonna          1.00     0.27     0.42     71
ben_afflek        0.00     0.00     0.00      0

accuracy           -         -     0.27     71
macro avg        0.25     0.07     0.11     71
weighted avg     1.00     0.27     0.42     71
```

max_iter=1500

```
▶ from sklearn.metrics import classification_report

print(classification_report(y_val, y_pred, target_names=classes))
```

```
↳      precision    recall   f1-score   support

elton_john       0.00     0.00     0.00      5
jerry_seinfeld   0.00     0.00     0.00      5
madonna          0.25     1.00     0.40      5
ben_afflek        0.00     0.00     0.00      5

accuracy           -         -     0.25     20
macro avg        0.06     0.25     0.10     20
weighted avg     0.06     0.25     0.10     20
```

```
▶ y_pred_train = clf.predict(X_train)

print(classification_report(y_pred_train, y_train, target_names=classes))
```

```
↳      precision    recall   f1-score   support

elton_john       0.00     0.00     0.00      0
jerry_seinfeld   0.00     0.00     0.00      0
madonna          1.00     0.27     0.42     71
ben_afflek        0.00     0.00     0.00      0

accuracy           -         -     0.27     71
macro avg        0.25     0.07     0.11     71
weighted avg     1.00     0.27     0.42     71
```

Analiza performansi MLP za identifikaciju ličnosti

U ranije navedenim izvještajima stoje procenti preciznosti treniranja i validacije prepoznavanja navedenih poznatih ličnosti na slikama.

Pri testiranju, mijenjala sam broj slojeva, neurona po slojevima, random_state i maksimalan broj iteracija.

Ono što se može primijetiti je da se pri različitim kombinacijama gore navedenih parametara ostvaruje izvjesni uspjeh validiranja sistema na novom setu slika, ali samo za jednu ličnost te isključivo 25%. U tim slučajevima, izvještaj o uspješnosti treniranja pokazuje 100% uspjeha za 25% validiranu ličnost. Postoje situacije kada je uspješnost treniranja sistema zabilježena kod više od jedne osobe. To je primjećeno u sljedećiminstancama:

Treniranje sistema na čitavim slikama:

```
hidden_layer_sizes=(1764, 500, 250, 100, 50, 1), max_iter>=150:
```

Preciznost validacije: Jerry Seinfeld =0.25

Preciznost treniranja: Jerry Seinfeld=1

Madonna=0.16

Treniranje sistema na slikama sa izdvojenim licem:

```
hidden_layer_sizes=(1764, 500, 250, 100, 50, 1), sve testne iteracije
```

Preciznost validacije: Jerry Seinfeld =0.25

Preciznost treniranja: Jerry Seinfeld=1

Madonna=0.05

Sumirane preciznosti validacije sistema:

(broj naveden u tablici označava broj pojavljivanja pozitivnih vrijednosti u izvještaju o uspješnosti treniranja sistema)

	cjelovite slike	izdvojeno lice
Ben Affleck	1	0
Madonna	6	6
Jerry Seinfeld	5	5
Elton John	3	4

Ukupan broj testiranja u oba slučaja: 15

Zaključila sam da se sve ličnosti, sumarno, u testnim slučajevima prepoznaju za treniranje i validaciju što je znak da nije došlo do greške pri učitavanju.

Prvi zaključak koji izvodim je da treniranje MLP-a, nad ovom bazom slika, ali i generalno, zavisi od broja ulaza za pojedinačnu ličnost. Broj slika, po folderima je:

	Train	Value
Madonna	19	5
E. John	17	5
B. Affleck	14	5
J. Seinf...	21	5

U svakom folderu slika za validiranje ima po 5 slika, ali broj slika za treniranje nije ravnomjerno raspoređen. Ako se u obzir uzme mogućnost postojanja slika na kojima sistem uopće nije prepoznao lice (što sam primijetila kod E. Johna), onda se neravnomjernost pojačava. Samim tim, sistem je imao više šansi da se istrenira za prepoznavanje pojedinih ličnosti, u odnosu na druge.

Uspješnost treniranja i prepoznavanja zavisi i od karakterističnosti slika.

Ben Affleck ima najviše slika u bazi koje su iz profila ili “ $\frac{3}{4}$ point of view”, a primjetno je da se sistem najslabije istrenirao na njegovim slikama, gotovo nikako. Elton John nosi naočale i to direktno utiče na gradijente fotografije. Jerry Seinfeld uglavnom ima frontalne fotografije, što objašnjava razlog njegovog učestalog pojavljivanja u izvještajima kao uspješno prepoznate/istrenirane osobe. Madonnine fotografije su raznolike, ali s obzirom da joj se izgled lica nije mijenjao od slike do slike, te je većina fotografija frontalno, razumljivo je što je sistem uspije prepoznati. Isto tako, zaključujem da se sistem najbolje trenirao upravo na njenim slikama, što se vidi iz odgovarajuće tabele.

Također, primjetno je da se performanse ne poboljšavaju povećanjem broja iteracija, što znači da analiza prirodno terminira u okviru od 100 iteracija. Dakle, problem nije nedostatak iteracija za treniranje. S druge strane, ni povećanje ili smanjivanje broja slojeva i neurona nije igralo veliku ulogu u poboljšanju preciznosti.

Loše rezultate koje daje izvještaj mogu prepisati svojoj neupućenosti u optimalne parametre za rad MLP na primjeru jedne baze kakva je uključena u ovaj projekt. Međutim, smatram da sam sprovela dovoljno testiranja da bih utvrdila varijacije u rezultatima. Šta god promijenila, ne pravi znatnu razliku u odnosu na preciznost prije promjena. Isto tako, postoji mogućnost da MLP nije idealan izbor za rad na ovom zadatku.

Pretražujući sam naišla na podatak da postoje druge vrste neuronskih mreža, kao što je CNN (Konvolucijska neuronska mreža), koja pokazuje bolje performanse u odnosu na MLP.