Gépi látás

GKNB_INTM038

Szem felismerése képeken

Banai Alex (RV27UI)

Mérnökinformatikus Bsc szakos hallgatók

Tartalomjegyzék

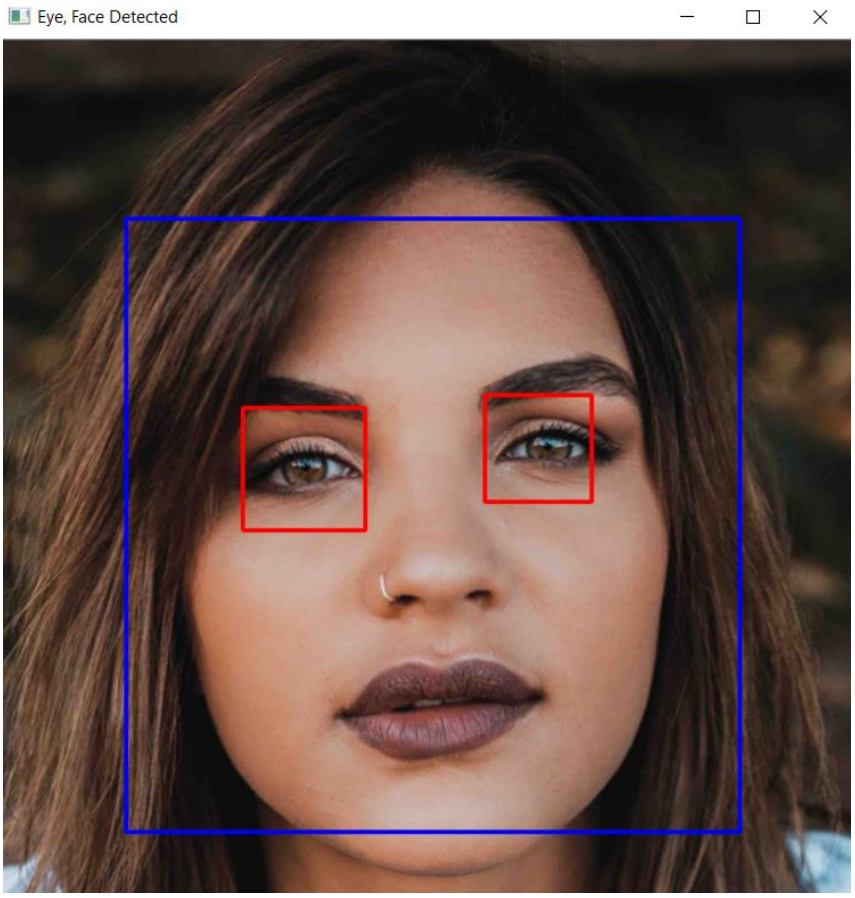
Bevezetés, megoldandó feladat kifejtése

Ahogy a téma címben is említettem, képeken történő szemfelismerést választottam feladatnak. Ha már a szemfelismerésről van szó, akkor úgy gondoltam az arc felismerést is beleviszem a témába, hogy sokkal érdekesebb legyen.

A két probléma megoldására nagyon hasonló kódot készítettem. Python programozási nyelvet használtam, ezen belül az openCV könytár volt segítségemre. Az említett könyvtáron belül található egy Haar Cascade classifier amit kifejezetten objektumok felismerésére tanítottak be, képek alapján. Ez volt ami konkrétan jól jött az én feladatomhoz, de próbálkoztam saját Haar Cascade készítésével amit 128 pozitív és 270 negatív képből tréningeztem. Ez volt az az érték ami a gépemen belátható idő alatt lefutott, a részleteket a továbbiakban kifejtem. Teszképekhez találtam egy oldalt ahonnan 1000 darabot le tudtam tölteni.

A program egy kép bemenetre a következő kimenetet adja.

Kék színnel az arc, **piros** színnel a szemek köré rajzol téglalapot.



Megoldáshoz szükséges elméleti háttér rövid ismertetése

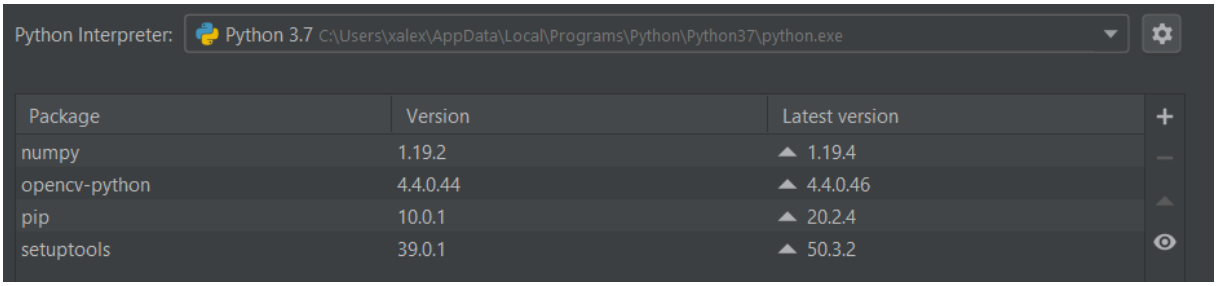
Python

Programozási nyelvnek az említett Python használtam, ez egy általános célú, magas szintű programozási nyelv, bonyolult logika van a hátterében, viszont felhasználó szinten nagyon könnyű használni (pointereket használ, mindent a heapen tárol).

Pythonból több verzió is elérhető, a 2.x és 3.x verzió közül én az újabbat választottam, az ok pedig az Open CV támogatás miatt volt.

A 3.7 vezióra lett az amivel sikerült a legújabb Open CV-t telepítenem. (4.4.0.44)

Programozás során próbáltam több környezetet is, (Thonny, Jupiter notebook, Visual Studio, Pycharm) de a Pycharm mellett döntöttem, számomra a könnyen kezelhetőség, átláthatóság, debug volt a fő szempont.



Open CV

OpenCV az Open Source Computer Vision Library-nek a rövidítése, ez gyakorlatilag egy fejlesztői könyvtár ami rengeteg algoritmust tartalmaz (kb. 2500). Gépilátás és gépitanulásban van fő szerepe. Ez pont témába illő volt számomra, nagyon sok munkát spóroltam vele. Több interfésszel is rendelkezik, és a legtöbb platformra elérhető. (MacOS, Windows, Linux és Android)

Az aktuálisan legújabb verziót használtam (4.4.0.46), persze azóta már megjelent a 4.5.0-ás verziója.

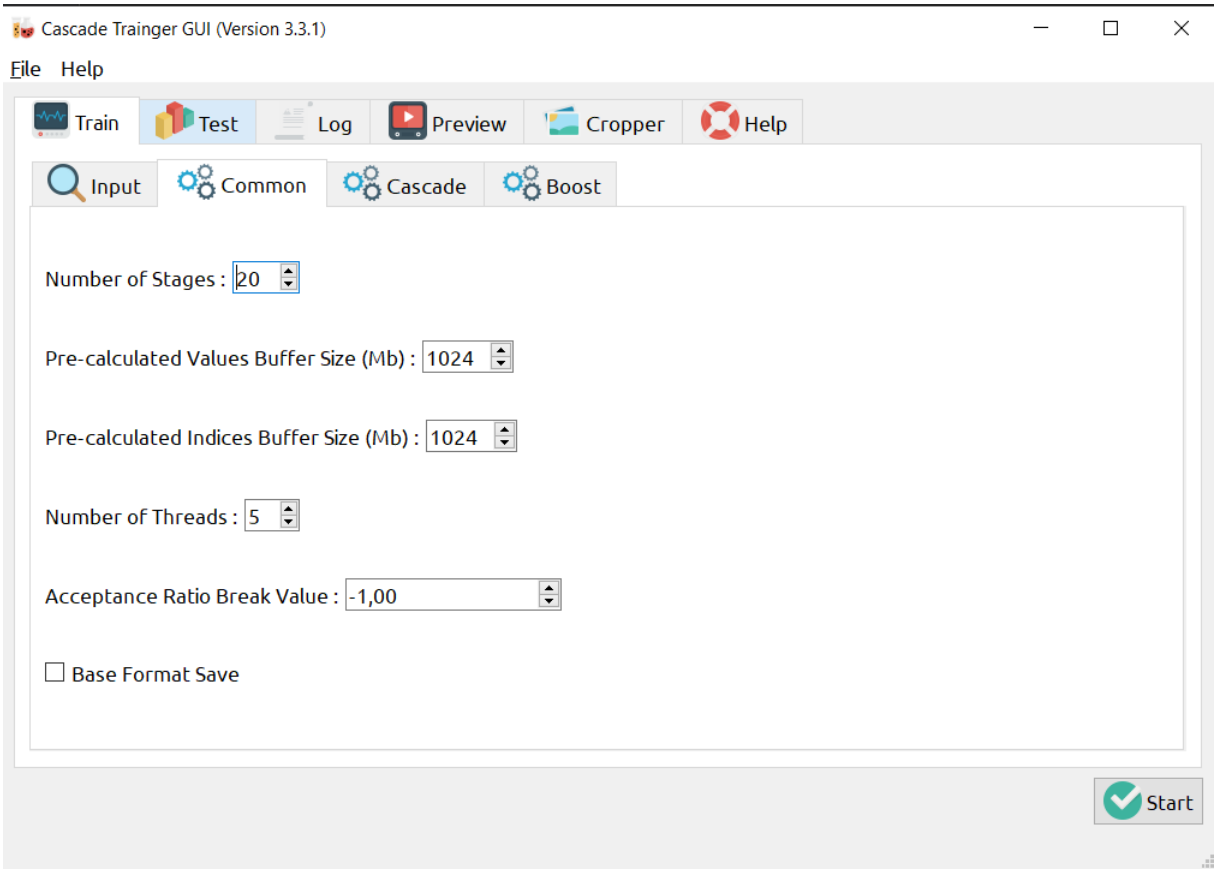
Haar Cascade

A Haar Cascade alapvetően egy classifier, amelyet objektumok felismerésére használnak (gépi tanulással készítik). A Haar Cascade úgy tanítják, hogy a pozitív képet egymásra helyezi a negatív képek halmazával. A pozitív képek amelyek tartalmazzák a felismerni kívánt objektumot, az én esetemben arcokat, illetve szemeket. A negatív képeken minden másnak kell lennie ezeken kívül, olyan dolgokra gondolok itt, mint egy arc felismerés esetén bármiféle háttér ami lehet az alany mögött. (természet, város, utca, szoba, tárgyak stb.)

A képzést általában szerveren és különféle stageken végzik (A megemelkedett erőforrás igény miatt). Jobb eredményt érhetünk el a jobb minőségű képek felhasználásával, illetve a Stagek számainak növelésével.

Cascade Trainer GUI

Egy olyan program ami Cascade classifier modellek tanításához, teszteléséhez, továbbfejlesztéséhez használható. Rendelkezik grafikai intefésszel, könnyen lehet paraméterezni, jól átlátható. Ebben készítettem el a pozitív, és negatív képek alapján a modellemet, amit az arc felismeréshez használtam. Nem volt annyira pontos mint, az OpenCV-ben található Cascada, de jóval kevesebb kép és erőforrás állt rendelkezésemre. (mint ahogy említettem az előző pontban ezeken múlik, a képen látható Stagek növelésével a pontosság is jobb lesz viszont a futási idő drasztikusan megemelkedik miatta)



A megvalósítás terve és kivitelezése

Saját Haar Cascade létrehozása

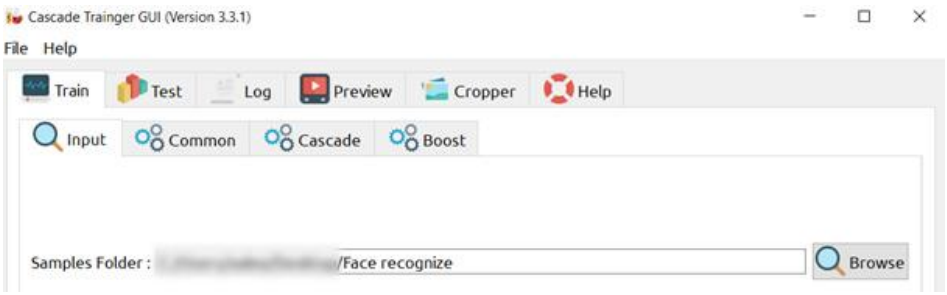
Az arcfelismeréshez készítettem sajátot, igazából azért mert ezekhez könnyebb volt pozitív és negatív képeket találni a tanításhoz. (mint a szemfelismeréshez)

Első körben a program indulása előtt, létre kell hozni egy mappát a könyvtárunkon belül tetszőleges helyen, és ezen belül még kettő:

Az egyiknek a neve „p” – ide jöttek a pozítiv képek (arcot tartalmazó képek)

A másiknak a neve „n” – ide jöttek a negatív képek (Mindent tartalmazhat, kivéve arcot (még részben se)! De a valóság az az, hogy leginkább olyan képeket érdemes, ami háttér szokott lenni egy portré esetében)

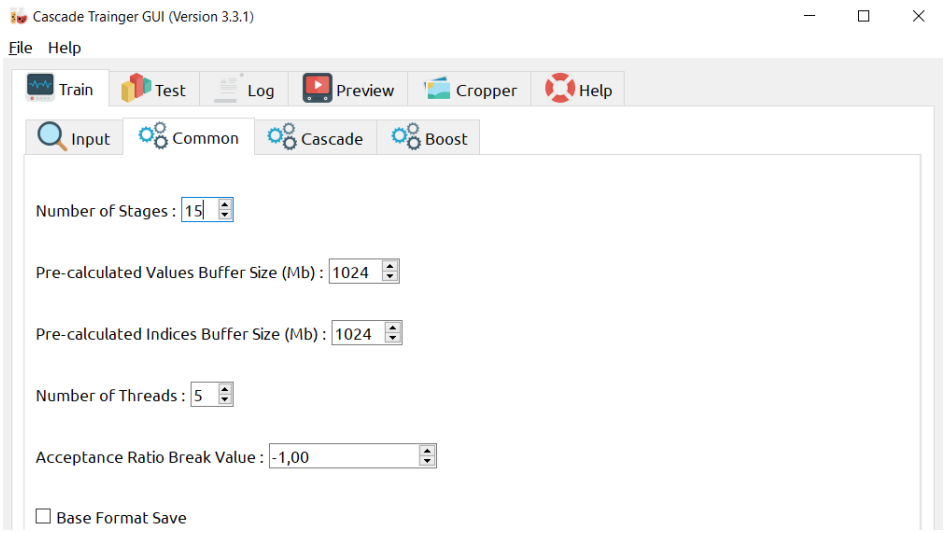
Az Input oldalon a Browse-ra kattintva a létrehozott mappát beadjuk neki.



A Common oldalon a következőket tudjuk állítani:

Stages: Az iterációk száma a tréning során, minél nagyobb a szám annál tovább tart. Én 15-20 között teszteltem. Elméletileg növelve, a pontosság is emelkedik, de a tesztek alapján más jött ki nálam.

Ezen az oldalon lehet még erőforrást is emelni (CPU, RAM).



A Cascade oldalon tudjuk állítani a mintavételezés szélességét és magasságát. Fontos megjegyezni itt, hogy a mintaképek képarányának meg kell egyeznie az ittenivel. (én 600x600 képeket használtam aminek huszonötöd része a 24. (ez az alapértelmezett érték)

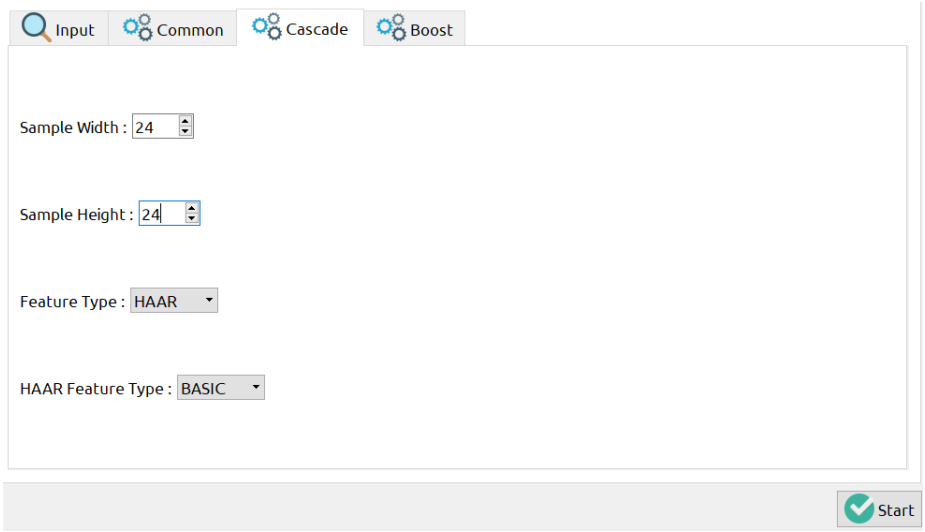
Növelni a méretet itt is nagyban lassítja a program futási idejét.

Itt lehet választani több classifiers közül is. (HAAR, LBP, HOG)

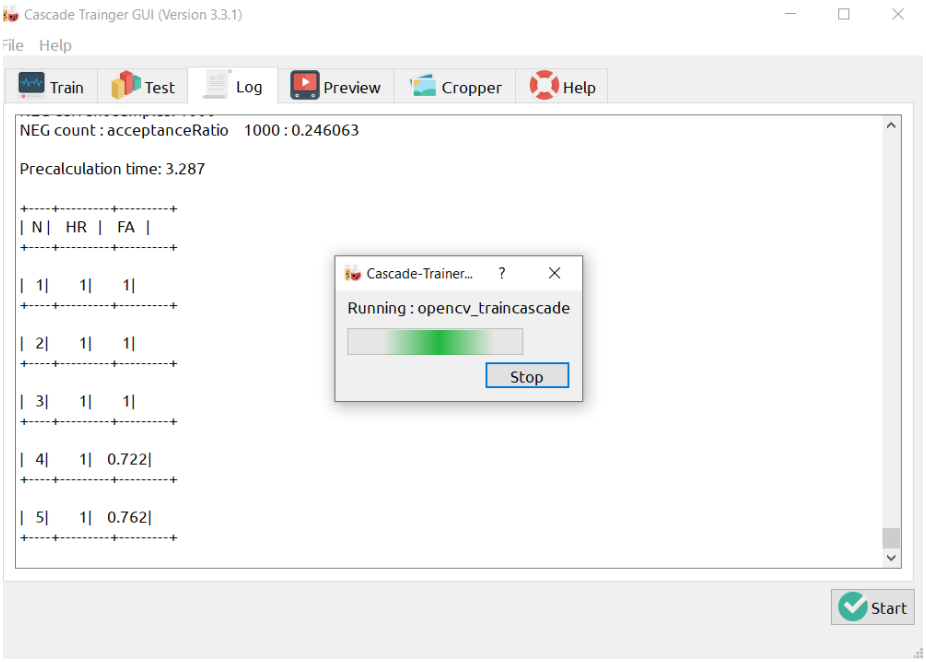
HOG-ot OpenCV 3.1 és újabb vezió felett ajánlják, nagyon pontos de tovább tart tanítani is.

LBP ez kevésbé pontos,viszont gyorsabb tanítani, és majdnem háromszor olyan gyorsan fut le az észlelés.

Én csak a HAAR teszteltem, nálam ez működött. (ez az alapértelmezett is)



Az utolsó fülnél (boost) csak a nagyon haladó embereknél ajánlották a módosítást, én emiatt békén is hagytam azt a részt. Ha minden meg van akkor a Start gombra kattintva elindul a tanítás.



Miután lefutott a program létrehoz a megjelölt könyvtárban egy classifier mappát, és azon belül lesz cascade.xml néven az általunk tréningezett file.

A program maga lehetővé teszi a tesztelést is, ki is próbáltam azt a funciót benne, ugyanazon elv alapján müködik, mint az általam használt.

A következő paraméterekkel futattam:

128 pozítiv 280 negatív kép:

15 stage 24*24 pixel	3 perc
16 stage 24*24 pixel	5 perc
17 stage 24*24 pixel	10-15 perc
18 stage 24*24 pixel	18-20 perc
19 stage 24*24 pixel	30-35 perc
20 stage 24*24 pixel	60-80 perc
14 stage 32*32 pixel	~1-2 óra futási idő

Forráskód

Összefoglalva

A programot a könyvtárból egy tetszőleges képet futtatva, felismerés esetén téglalapot rajzol a szemek és arc köré. Amennyiben nem sikerül felismernie, elmossa az adott képet és kiírja középre a következőket „Nem sikerült a felismerés!”.

Kód felépítése

```
import cv2
#Itt betöltöm a OpenCV könyvtárat

# A project könyvtárából az egyik tesztképet itt töltöm be egy változóba.
image = cv2.imread("real_00034.jpg")
blurImg = cv2.blur(image,(5,5))
# A cv2.blur a kép elmosásért felelős, itt töltöm be egy változóban a tesztkép elmosott verzióját, amit abban az esetben jelenítek meg, ha az eredeti képen nem sikerült helyesen detektálni a szemeket, és az arcokat.
# Gyakorlatilag ez az Avarage Blur-nak felel meg. (esetemben 5x5 pixel alatt átlagot von és a középső elemet lecseréli az átlagra)


# Itt a saját Haar Cascade Classifiert töltöm be egy változóba amit már a könytáramban lemásoltam „haarcascade_eye.xml” illetve "haarcascade_frontalface_default.xml" néven.
A kikommentelt sorban az általam tréningezett Haar Cascade található „own_cascade.xml” néven. Ez mint említettem 128 pozítív és 270 negatív képből tanítottam.
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
face_cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
#face_cascade = cv2.CascadeClassifier("own_cascade.xml") #trained by 128
```

```
positive image, 270 negative image

# A quit egy logika változó amit a következő ciklusokhoz használtam, hogy a kilépésüket szabályozzam.
# Az i egy iterátor, amit szintén a ciklusokhoz használtam.
quit = False
i = 1

# Itt kezdődik az első ciklus, amiben a szem detektálást végeztem eyes = eye_cascade.detectMultiScale(image, scaleFactor = 1.1, minNeighbors = i) résszel.

# Alapjáraton az arc detektálást tűnik első ciklusnak logikusabbnak, de a szemünket jóval nehezebben lehet felismerni mint az arcot ezért ezt raktam előre, ha már a szemet fel se ismeri akkor az arcfelismerés le se fusson.

#Az eye_cascade.detectMultiScale 2 * 4 integert fog visszaadni,  numpy.ndarray konténerbe.

# Az első 4 az egyik szemhez, a második 4 a másik szemhez tartozik. Az első kettő szám az x (1.) és y (2.) kordinátája a szem kezdőpontja (bal felső sarok). Az x-hez hozzáadva a 3. és az y-hoz hozzáadva a 4. számot megkapjuk a jobb alsó sarkot (szem végpontját). Ezzel el is mondtam a kirajzolásnak a logikáját is. (későbbiekben fogom megvalósítani)
while  quit == False and i < 200:
    eyes = eye_cascade.detectMultiScale(image, scaleFactor=1.1, minNeighbors=i)
    print(f"I értéke: {i}")
    print(f"Szem mennyisége: {len(eyes)}")
    if len(eyes) == 2:
        quit = True
    if len(eyes) == 0:
        quit = True
    i+=1
#Az eye_cascade.detectMultiScale paraméterei:

# image – a detektálni kívánt kép neve az első paraméter.

# ScaleFactor – a sebességért és a pontosságért felel. Növelve csökken a pontosság, viszont rövidül a futási sebesség.

# minNeighbors – ezt az értéket szabályozom a ciklusban az iterátor segítségével, 1ről növelem és közben kiíratom a látványosság kedvért, hogy hány arcot és szemet talált.

A ciklus végén van két elágazás is.
Az első a helyes detektálás utáni kilépésért felel.
A második a felesleges futásért felel.

# Itt újra kezdőértékre állítom az iterátor és a logika változót a következő ciklus elindulása érdekében.
i = 1
quit = False

# Ez a ciklus teljesen ugyan az csak arcra, egyedül a logika vizsgálat tér el a ciklus elején amivel azt vizsgáljuk, hogy ha nem talált szemeket (vagy nem reálisat) akkor arcot se
```

keressen.

```
while  quit == False and i < 200 and len(eyes) == 2:
    faces = face_cascade.detectMultiScale(image, scaleFactor=1.1,
minNeighbors=i)
    print(f"I értéke: {i}")
    print(f"Arc mennyisége: {len(faces)}")
    if len(faces) == 1:
        quit = True
    if len(eyes) == 0:
        quit = True
    i+=1
```

#Ez egy logika vizsgálat a kirajzolás előtt, kordináták alapján azt vizsgálom, hogy ha az arcon kívül esik a szem akkor, az elmosott képet jelenítse meg.

```
# notreal - ezzel a logikai változóval szabályozom
notreal = False

if  len(eyes) > 0 and len(faces) > 0 :
    # Jobb szem
     # Starting point
     # x kordináta x kordináta  vagy  y kordináta y kordináta
    if faces[0,0] > eyes[0,0] or faces[0,1] > eyes[0,1]:
        notreal = True
        print("Hibás pozíció 1")
    # End point
    if [faces[0,0] + faces[0,2]] < [eyes[0,0] + eyes[0,2]] or [faces[0,1] +
faces[0,3]] < [eyes[0,0] + eyes[0,3]]:
        notreal = True
        print("Hibás pozíció 2")
    # Bal szem
    # Starting point
    if faces[0, 0] > eyes[1, 0] or faces[0, 1] > eyes[1, 1]:
        notreal = True
        print("Hibás pozíció 3")

    # End point
    if [faces[0, 0] + faces[0, 2]] < [eyes[1, 0] + eyes[1, 2]] or [faces[0,
1] + faces[0, 3]] < [eyes[1, 0] + eyes[1, 3]]:
        notreal = True
        print("Hibás pozíció 4")


# Itt történik a kiíratás azon része amikor nem 2 szemet vagy nem 1 arcot, vagy az arc nem tartalmazza a szemet szenárióval találkozunk.
if len(eyes) != 2 or len(faces) != 1 or notreal:
    # betűtípusa a kiíratásnak
    font = cv2.FONT_HERSHEY_SIMPLEX
    # pozíciója a kiíratásnak (500x500-as képekkel dolgoztam, itt nagyjából középen van a szöveg)
    org = (40, 320)
    # Szöveg mérete
    fontScale = 1.2
    # Szöveg színe
    color = (255, 255, 255)
    # Szöveg vastagsága
    thickness = 4
```

```
    # Az említett paraméterekkel „Nem sikerult a felismeres” szöveg  kiíratása a kép közepére.
    image = cv2.putText(blurImg, 'Nem sikerult a felismeres!', org, font,
                        fontScale, color, thickness, cv2.LINE_AA)
# Ez az érdekesebb ág, amennyiben a képen jól detektáltuk a szemet és arcot a cv.rectangle függvénnyel kirajzoljuk a pozíciójukat. (téglalapot rajzol a szem és arc köré)
else:
    for (ex, ey, ew, eh) in eyes:
        cv2.rectangle(image, (ex, ey), (ex + ew, ey + eh), (0, 0, 255), 2)
# cv2. rectangle Paraméterei:
# image – a rajzolni kívánt kép neve az első paraméter.
# (ex, ey) – kezdő x és y kordináta (bal felső sarok)
# (ex + ew, ey + eh) – vég x és y kordináta (jobb alsó sarok)
# (0, 0, 255) – a vonal színe RGB kódban
# és az utolsó kordináta a vonal vastagsága

    for x, y, w, h in faces:
        cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2) #
# A végső kép megjelenítése „Eye, Face Detected” szöveggel
# In this line of code we want to show our image
cv2.imshow("Eye, Face Detected", image)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Hibaforrás

Az arc és szem felismerés korlátozott, a csukott szemeket nem igazán ismeri fel, valószínűleg a opencv által tartalmazott haar cascada tanulása során nem találkozott ilyen képekkel. Arcoknál ha egy részét eltakarjuk a kezünkkel, szintén nem fogja felismerni.

Rossz detektálás: Le van kezelve ha az arcon kívül szemet ismerne fel, az hamis detektálás, viszont ha az arcon belül máshol talál (például: orrot szemnek érzékeli) akkor azt sajnos hibásan jeleníti meg.

Tesztelés

Szerencsére találtam jó pár képet a Kaggle-on, ez egy olyan oldal ahol dataseteket lehet letölteni. Egy 1000 db-os packot vettem le, ami a tanításban is szerepet játszott.

Létrehoztam több arcfelismerő Haar Cascadat is a következő paraméterekkel:

128 pozítiv 280 negatív kép:

Paraméterek	Futási idő
15 stage 24*24 pixel	3 perc futási idő
16 stage 24*24 pixel	5 perc futási idő
17 stage 24*24 pixel	10-15 perc futási idő
18 stage 24*24 pixel	18-20 perc futási idő
19 stage 24*24 pixel	30-35 perc futási idő
20 stage 24*24 pixel	60-80 perc futási idő
14 stage 32*32 pixel	~1-2 óra futási idő

<u>openCV által tartalmazott (a fájból olvastam ki):</u>

Eye detection

??? pozitív és negatív képből:

24 stage 20*20 pixel	**???**

Face detection

??? pozitív és negatív képből:

25 stage 24*24 pixel	**???**

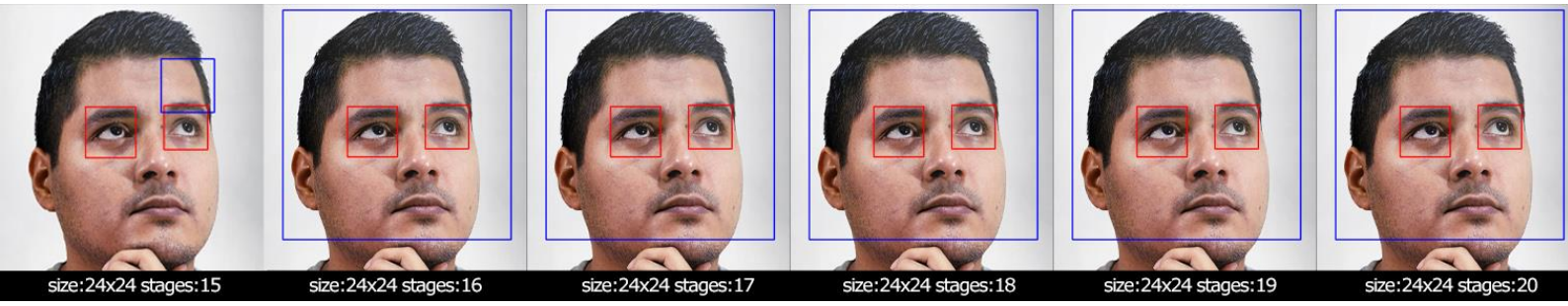
A tesztek kedvéért kikapcsoltam azt a funkciót ami tartalmazza, hogy a fejen belül kell megtalálni a szemeket, különben hibás detektálás.

Amennyiben arcot nem talál akkor a szemet se jeleníti meg. (fordítva is igaz)

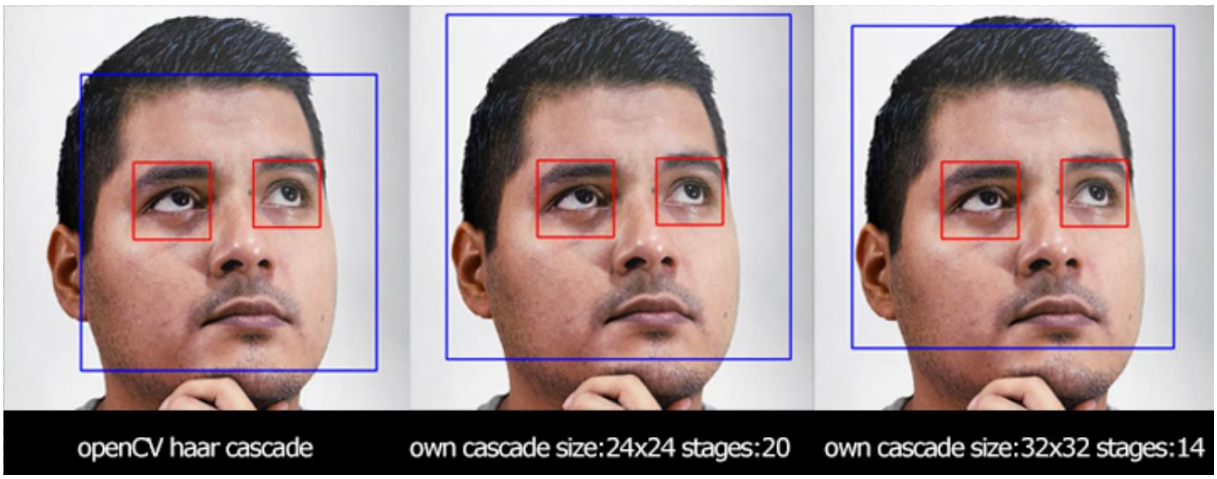
Jobb eredményt érhetünk el a pixel (width height) és a stageek növelésével. Azonban a gyakorlatban nálam valamiért a stagek növelésével egy idő után romlott az algoritmus pontossága.

Ahogy a fenti képen láthatni, az utolsó 4 képen abszolút 1 arcot se talált, a 16 iterációs érte el a legjobb eredményt, ott az arc közepét betalálta, a 15 stagesen a háttér bokehját érzékelte arcnak ami szintén hibás.

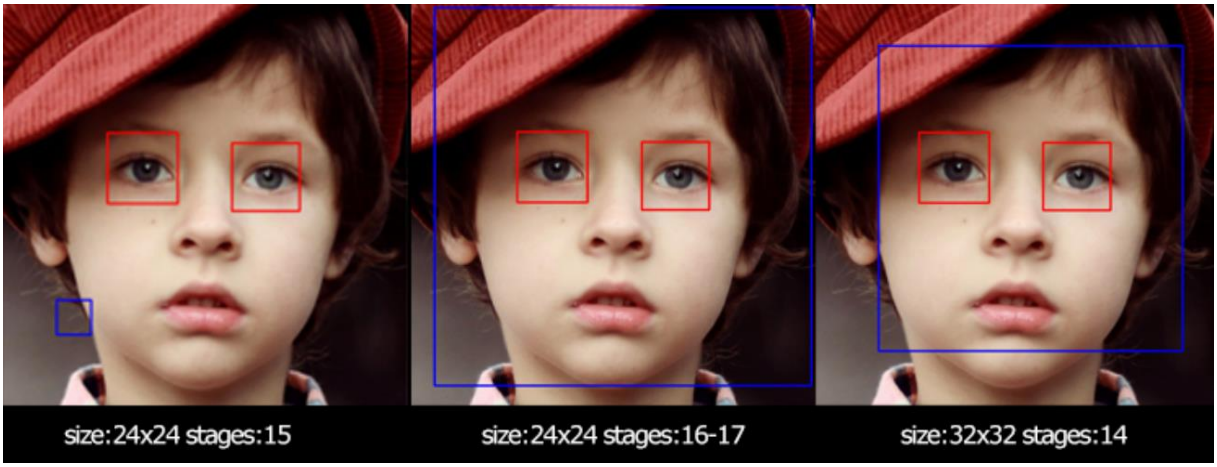
Viszont volt kép ahol pont fordítva volt:



Itt láthatóan a 15-ös a legrosszabb, a több teljesen ugyanolyannak mondhatni.

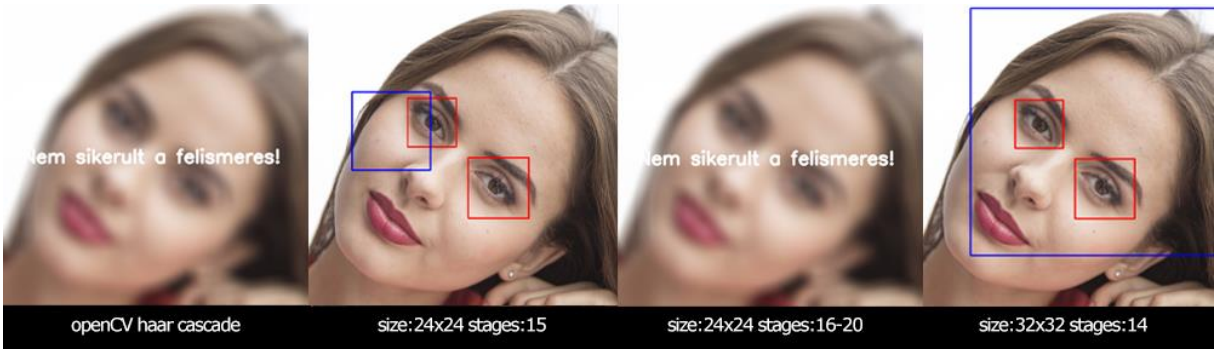


Itt már belevettem a nagyobb felbontásut, illetve a openCV által tartalmazottat. Nem is kérdés, hogy az müködött a legjobban, a másik két eredmény pedig nagyon hasonló.

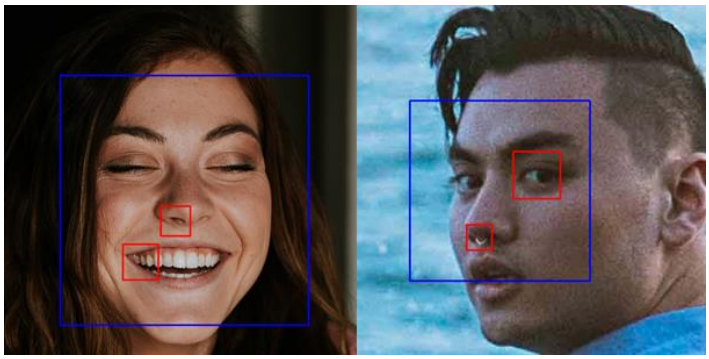


Itt csak az általam tréningezetteket tettem bele, a 15-ös használhatatlan, a 16-17 egész jó megoldást adott, a 18-20 nem ismert fel arcot, emiatt nem is tettem bele, és a végén a nagyobb felbontású 32x32-es 14 iterációt tartalmazó müködött egyértelműen a legjobban.

Itt a 15-ös talált valamit de hibásan, a 16 és 20 közötti egyetlen arcot sem tudott detektálni, viszont ami nagy meglepetés a nagyobb felbontású 14 stages jól eltalálta.



Ami számomra meglepetést okozott, hogy néhány képen jobban működött az általam tréningezett mint az openCV által tartalmazott. (megemlíteném, hogy a tréningezés során nem használtam ezt a képet).



Amit a programba nem tudtam lekezelni az, az ha a szem pozícióját az arcon belül rossz helyre teszi. A csukott szemek azok amit még nem igazán ismer fel, valószínűleg a opencv által tartalmazott haar cascada tanítása alatt nem szerepelt a pozitív képek között.

Összegzés

Összeségében kijelenthetem, hogy a 32x32-es működőtt a legjobban (az openCV-et leszámítva, vele hasonlítani nem lenne igazságos, hiszen jóval kevesebb erőforrás állt rendelkezésemre), ezek után talán a 16-17 iterációs, a 15-ös gyakran érzékelt olyan esetekben amiben a többi nem talált, viszont hibásan.

Irodalomjegyzék

Pyton

OpenCV

Kaggle - képek

Haar Cascade

Haar Cascade 2

Eye Detection

Avering Blur

Cascade Traning GUI