Gépi látás

GKNB_INTM038

Szem felismerése képeken

Banai Alex (RV27UI)

Mérnökinformatikus Bsc szakos hallgatók

Tartalomjegyzék

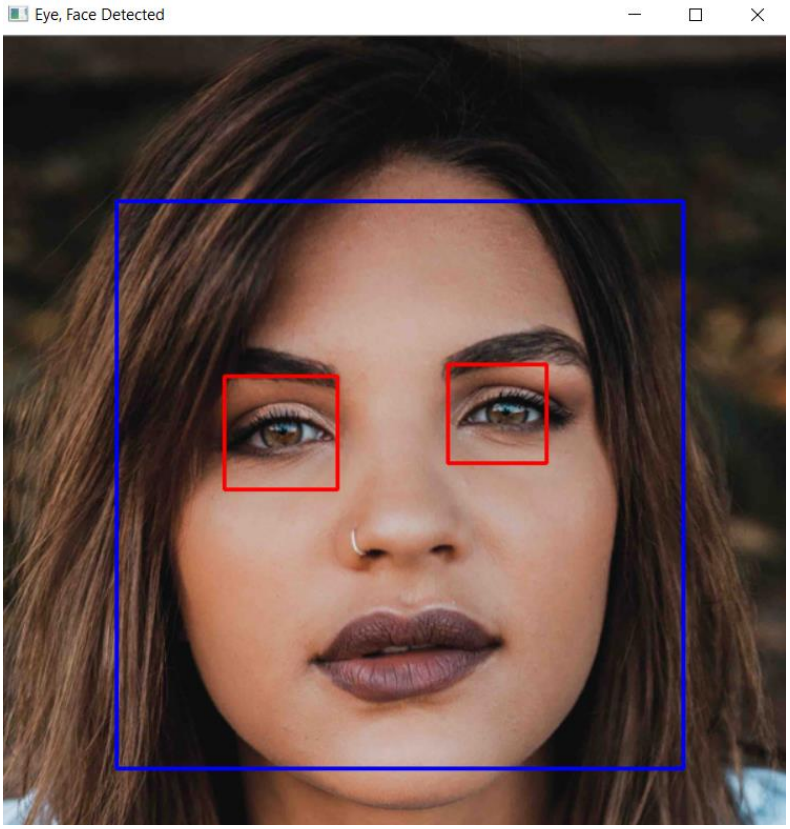
Bevezetés, megoldandó feladat kifejtése

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. (1. ábra)

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



1. ábra: Arc és szem felismerése

Az eredeti kép forrása: *https://www.kaggle.com/ciplab/real-and-fake-face-detection*

Megoldáshoz szükséges elméleti háttér rövid ismertetése

Haar Cascade

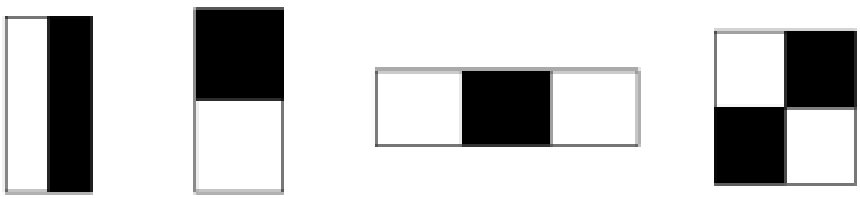
A Haar Cascade alapvetően egy classifier, amelyet objektumok felismerésére használnak (gépi tanulással készítik). A Haar Cascade úgy tanítják, hogy a pozitív képet egymásra helyezi a negatív képek halmazával. A pozítiv képek, amelyek tartalmazzák a felismerni kívánt objektumot, az én esetemben arcokat, 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.) [3]

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. [2]

Napjainkban ez az egyik leghatékonyabb és legelterjedtebb módszere az általános objektumok felismerésére. Az objektumok tetszőlegesek lehetnek, lehet, ember, arc, szem vagy tárgyak is. Azt megjegyezném, hogy olyan tárgyakat tud csak felismerni, amit 24× 24 pixel méretben, 255 fokozatú szürkeárnyalatosan is egy ember számára felismerhető. [2]

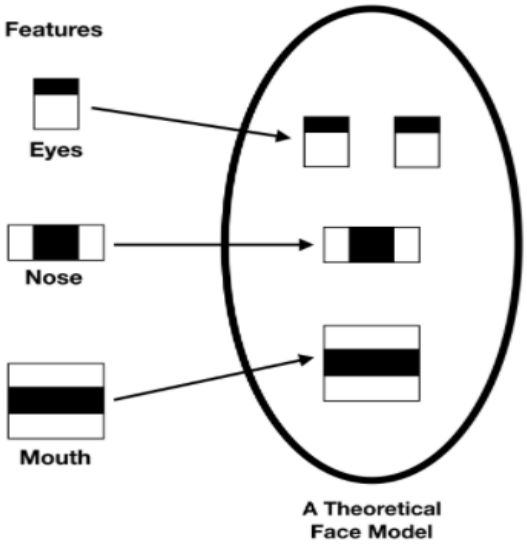
A felismerés több dolog alapján működhet. A következők lehetnek: Haar like features, LBP (Local Binary Patterns), és HOG (Histogram of Oriented Gradients)

Én a “Haar like feature” tárgyalnám mert ezt használtam a programom elkészítéséhez.



2. ábra: “Haar like feature”-hoz használt objektum detektáló eszközök

Az eredeti kép forrása: https://tdk.bme.hu/VIK/DownloadPaper/Valos-ideju-objektum-felismeres-gepi-latas



3. ábra: Az arc felismeréshez használt eszközök (haar like feature)

Az eredeti kép forrása: https://netacademia.hu/pages/a-haar-cascade-megertese

Na de hogyan is működik?

Az adott képet átméretezi 24 × 24 pixel méretre (azt gondolnánk, hogy egy ekkora kép nem tartalmazhat sok információt, de meglepően, több min százezer jellegzetességgel rendelkezik), és a kép színét átállítja fekete fehérre (ez azért előnyös mert így nem fog annyira számítani, a bőrszín vagy, hogy ferde a szeme az illetőnek és a szem nyitottsága). Az így keletkezett nagyon kis felbontású pixeles képen fehér, és fekete közötti árnyalatok lesznek. Ahogy a 4. ábrán is láthatni szemünk, szájunk és orrunk sötétebb lesz arcunkhoz képest. Maga a mintázatokat amiket keresni fog arcunkon, azt az 3. ábrán láthatjuk. [2]

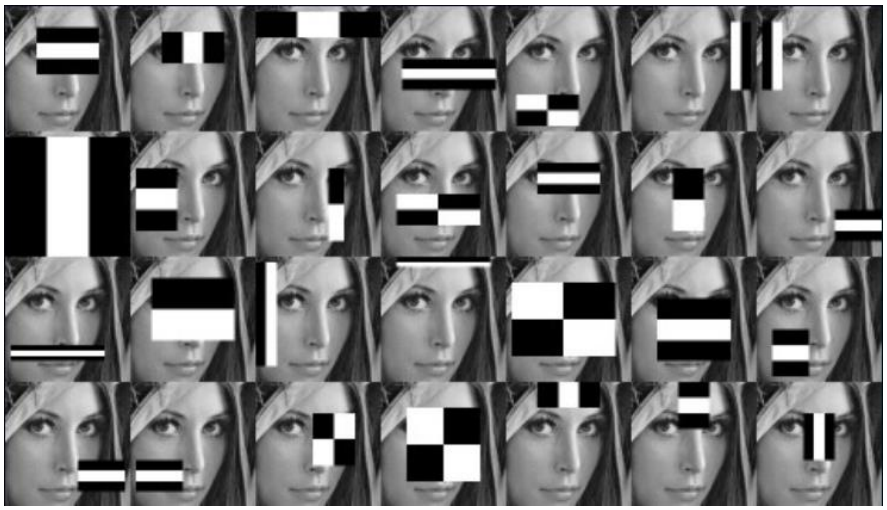
4. ábra: 24 × 24 pixeles portré (fekete- fehér)

Az eredeti kép forrása: https://netacademia.hu/pages/a-haar-cascade-megertese

Amit még fontos megjegyezni, hogy ez a módszer nem arcot ismeri fel, mert az jóval nehezebb lenne és pontatlanabb folyamat lenne, hanem kiszortírozza a kép azon részét, ami nem tartalmaz arcokat. (jelen esetembe arcfelismerő, de ha valami tárgyra használjuk a felismerést például egy órára, akkor minden olyat elvet, ami nem óra). Tulajdonképpen egy negatív felismerő, és a kidobált elemek után csak a detektálni kívánt rész marad a halmazban. (Az én esetemben az arc, és ezt fogom majd körbe rajzoltatni az eredeti képen) [2]

A módszer hátránya, hogy annyira jól működhet, hogy mosolygós, tárgyakra is arcot fog felismerni, viszont itt fontos szerepet játszik a stagek (iterációk), hogy még jó néhányszor (az

én esetemben 14-20 közöttivel próbálkoztam) iterálunk az arcon, és kiszórja a hibásan detektált arcokat. Ebből már jól látható miért számít a stage értéke annyira a futási idő szempontjából.



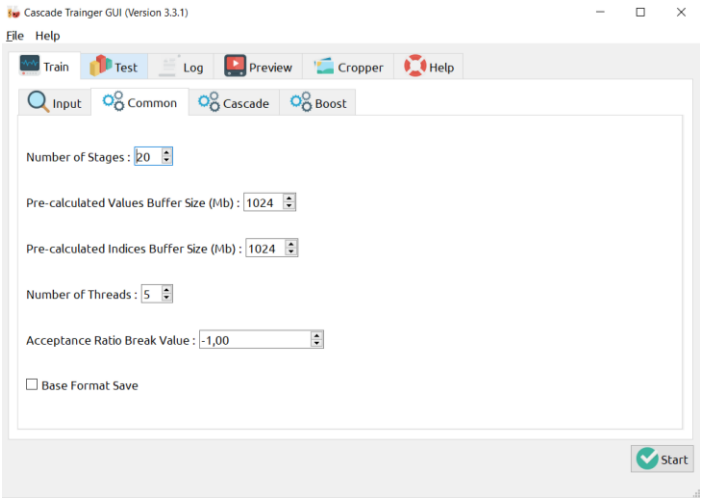
5. ábra: Stagek közti műveletek (mindent megtart, ami arc, és a fals pozitív részeket kidobja)

Az eredeti kép forrása: https://netacademia.hu/pages/a-haar-cascade-megertese

Ez a folyamat meglepően gyorsan fut, ez mutatja azt is, hogy az én személyi számítógépemen is lefutott belátható időn belül. (gépi tanulások nagy részére nem lenne alkalmas a gépem)

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 pozítiv, 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) [4]



6. ábra: A tréningező program Train - Common menüpontja

Fejlesztői környezet

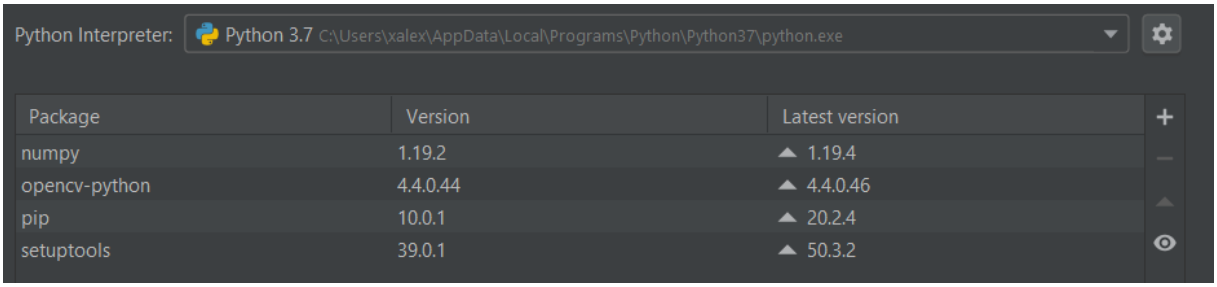
Python

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). [1]

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 verzió az az amivel sikerült a legújabb Open CV-t telepítenem. (A 2. ábrán látható)

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.



7. ábra: Pycharm menüje

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) [2]

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

A megvalósítás terve és kivitelezése

Saját Haar Cascade létrehozása

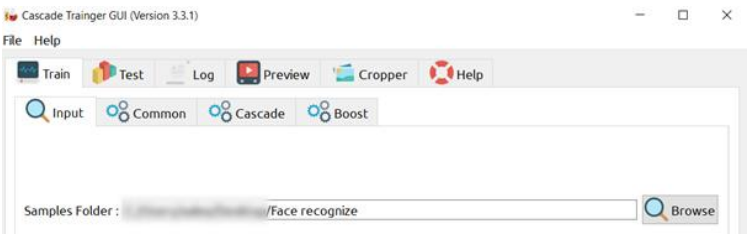
Az arcfelismeréshez készítettem sajátot, igazából azért, mert ezekhez könnyebb volt pozitív és negatív képeket találni a tanításhoz. (mint a szemfelismeréshez)

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.

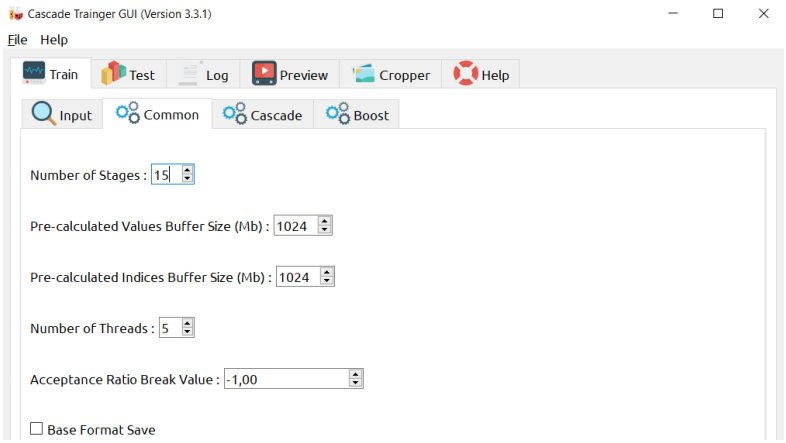


8. ábra: A tréningező program Train – Input menüpontja

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).



9. ábra: A tréningező program Train – Common menüpontja

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)

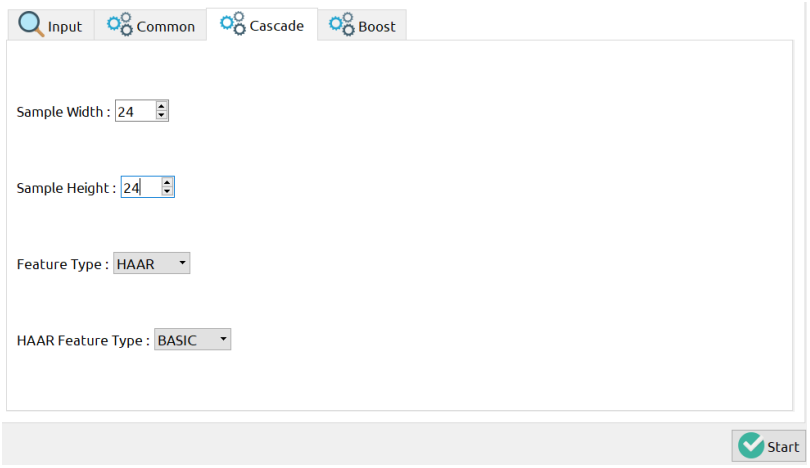
Növelni a méretet itt is nagyban lassítja a program futási idejét.

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.

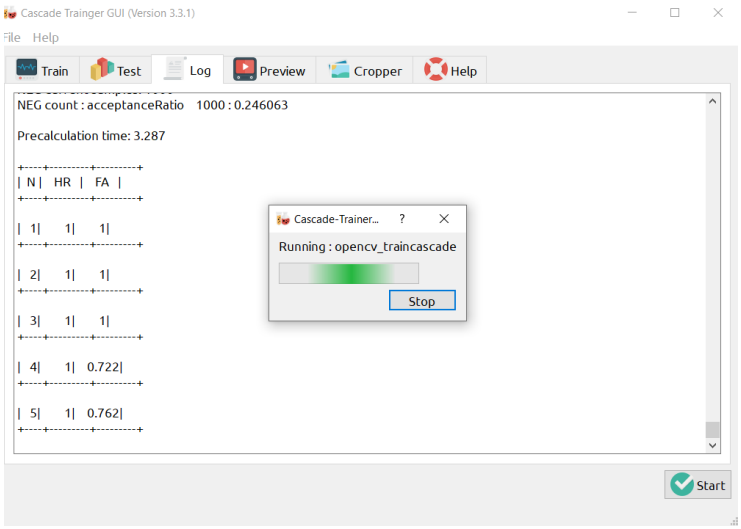
LBP ez kevésbé pontos,viszont gyorsabb tanítani, és majdnem háromszor olyan gyorsan fut le az észlelés.

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



10. ábra: A tréningező program Train – Cascade menüpontja

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.



11. ábra: A tréningező program tanítás közben

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. (9.ábra)

Paraméterek	Futási idő (perc)	128 pozítiv 280 negatív (db)
15 stage 24 × 24 pixel	5	128 pozítiv 280 negatív
16 stage 24 × 24 pixel	10 - 15	128 pozítiv 280 negatív
17 stage 24 × 24 pixel	18 - 20	128 pozítiv 280 negatív
18 stage 24 × 24 pixel	30 - 35	128 pozítiv 280 negatív
19 stage 24 × 24 pixel	60 - 80	128 pozítiv 280 negatív
20 stage 24 × 24 pixel	60 - 120	128 pozítiv 280 negatív
14 stage 32 × 32 pixel	60 - 120	128 pozítiv 280 negatív

12. ábra: Táblázat az általam tréningezett beállításokból – 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

```
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 jelenítem 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).

```
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 saját Haar Cascade Classifiert töltöm be egy változóba, amit már a könytáramban lemásoltam „haarcascade_eye.xml” illetve "haarcascade_frontalface_default.xml" néven.

```
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
```

Alapjáraton az arc detektálást tűnik első ciklusnak logikusabbnak, de a szemeket 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)

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, 1-ről 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.

```
# 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")
```

Kordináták alapján megnézem, hogy az arcon belül található-e mindkét szem, ha nem akkor pozitívra állítom át a „notreal” logikai változót, ennek a kiíratásban lesz szerepe.

```
if len(eyes) != 2 or len(faces) != 1 or notreal:
    font = cv2.FONT_HERSHEY_SIMPLEX
    org = (40, 320)
    fontScale = 1.2v
```

```
    color = (255, 255, 255)
    thickness = 4

    image = cv2.putText(blurImg, 'Nem sikerult a felismeres!', org, font,
                        fontScale, color, thickness, cv2.LINE_AA)
```

Itt történik a kiíratás azon része amikor nem 2 szemet vagy nem 1 arcot, vagy az arc nem tartalmazza a szemet szenárióval találkozunk.

```
else:
    for (ex, ey, ew, eh) in eyes:
        cv2.rectangle(image, (ex, ey), (ex + ew, ey + eh), (0, 0, 255), 2)
```

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).

A 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), 2)*** – a vonal színe RGB kódban és az utolsó kordináta a vonal vastagsága

Hibaforrás

Az arc és szem felismerés korlátozott, a csukott szemeket nem igazán ismeri fel, valószínűleg az 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. (10. ábra)

Paraméterek	Futási idő (perc)	Pozítiv és negatív képek száma (db)
15 stage, 24 × 24 pixel	3	128 pozítiv 280 negatív
16 stage, 24 × 24 pixel	5	128 pozítiv 280 negatív
17 stage, 24 × 24 pixel	10 - 15	128 pozítiv 280 negatív
18 stage, 24 × 24 pixel	18 - 20	128 pozítiv 280 negatív
19 stage, 24 × 24 pixel	30 - 35	128 pozítiv 280 negatív
20 stage, 24 × 24 pixel	60 - 80	128 pozítiv 280 negatív
14 stage, 32 × 32 pixel	60 - 120	128 pozítiv 280 negatív

13. ábra: A tréningezési beállítások részletei

Típus	Paraméterek	Futási idő (perc)	Pozítiv és negatív képek száma (db)
Szem felismerés	24 stage, 20 × 20 pixel	Ismeretlen	Ismeretlen
Arc felismerés	25 stage, 24 × 24 pixel	Ismeretlen	Ismeretlen

14. ábra: openCV által tartalmazott fájl részletei

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)

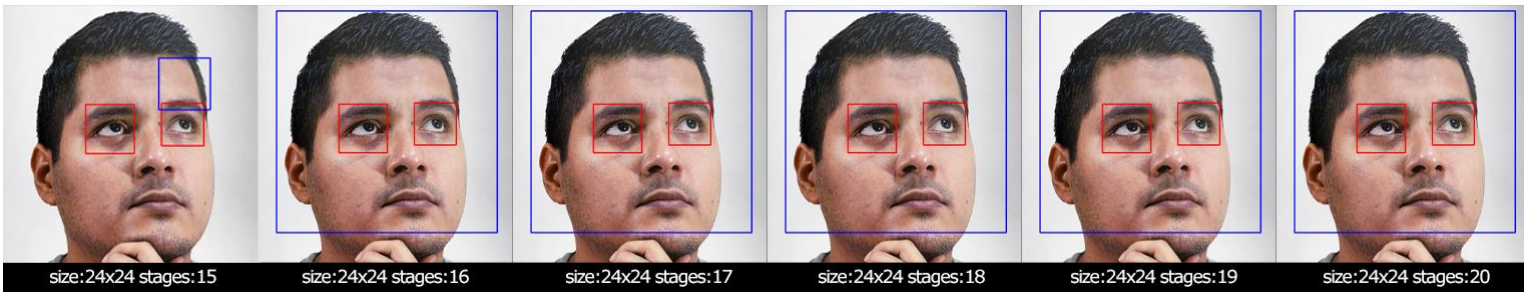
Jobb eredményt érhetünk el a pixel (width height) és a stageek növelésével. Azonban a gyakorlatban nálam valamiért a stagek növelésével egy idő után romlott az algoritmus pontossága.

15. ábra: Az arc és szem felismerése az általam tréningezettel

Az eredeti kép forrása: *https://www.kaggle.com/ciplab/real-and-fake-face-detection*

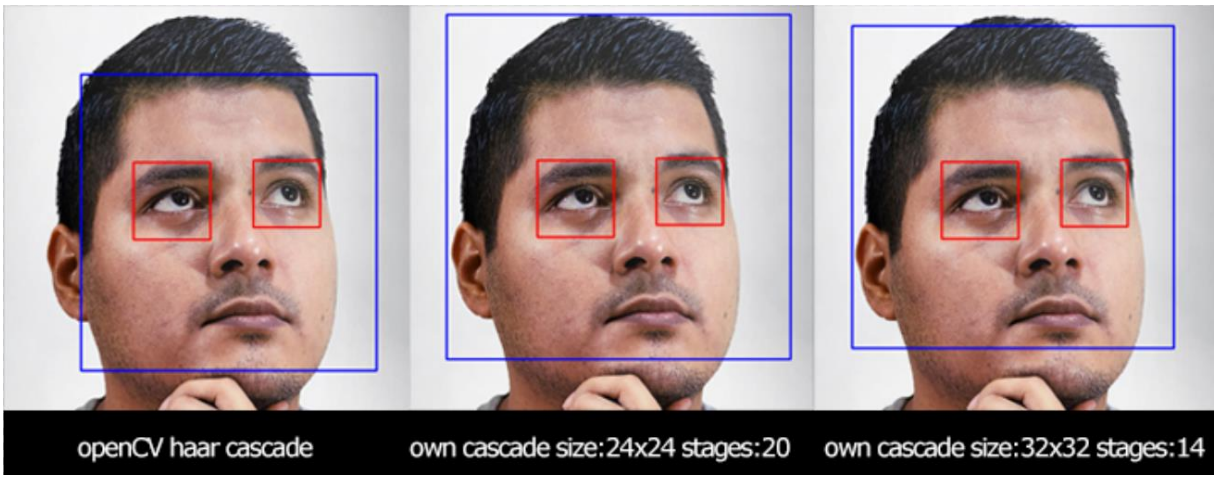
Ahogy a 12. ábrán láthatni, az utolsó 4 képen abszolút 1 arcot se talált, a 16 iterációs érte el a legjobb eredményt, ott az arc közepét betalálta, a 15 stagesen a háttér bokehját érzékelte arcnak ami szintén hibás.

Viszont volt kép ahol pont fordítva volt:



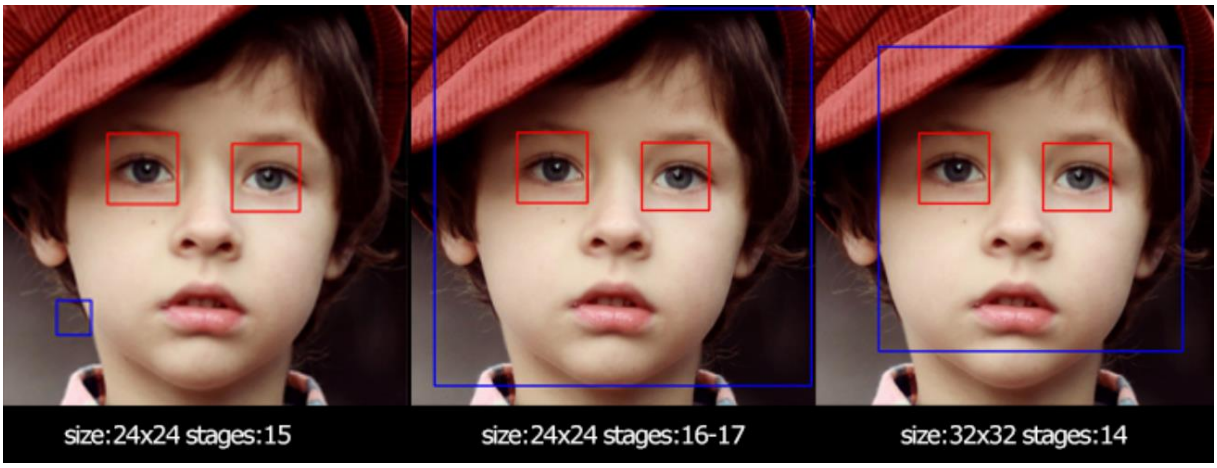
16. ábra: Az arc és szem felismerése az általam tréningezettel
Az eredeti kép forrása: *https://www.kaggle.com/ciplab/real-and-fake-face-detection*

A 13. ábrán láthatóan a 15-ös a legrosszabb, a több teljesen ugyanolyannak mondhatni.



17. ábra: Az arc és szem felismerése az általam tréningezettel
Az eredeti kép forrása: *https://www.kaggle.com/ciplab/real-and-fake-face-detection*

A 14. ábrán láthatóan belevettem a nagyobb felbontásut, illetve a openCV által tartalmazottat. Nem is kérdés, hogy az működött a legjobban, a másik két eredmény pedig nagyon hasonló.



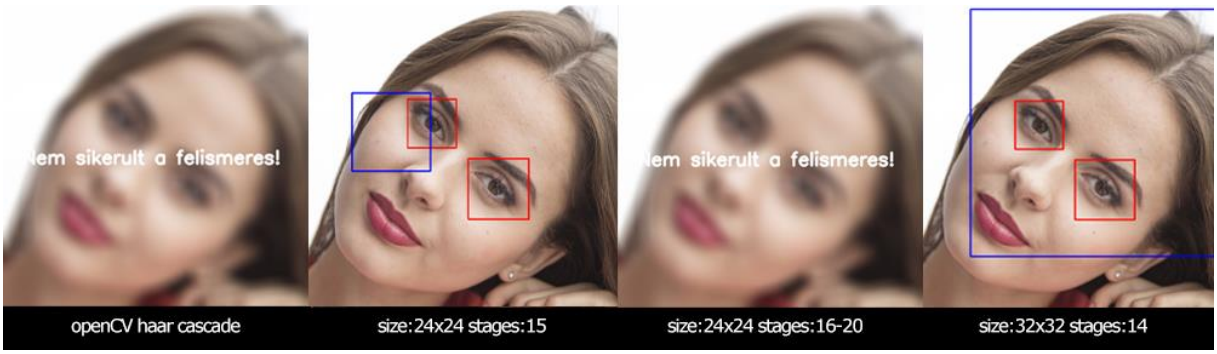
18. ábra: Az arc és szem felismerése az általam tréningezettel

Az eredeti kép forrása: *https://www.kaggle.com/ciplab/real-and-fake-face-detection*

A 15. ábrán láthatón csak az általam tréningezetteket tettem bele, a 15 stages 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 működött egyértelműen a legjobban.

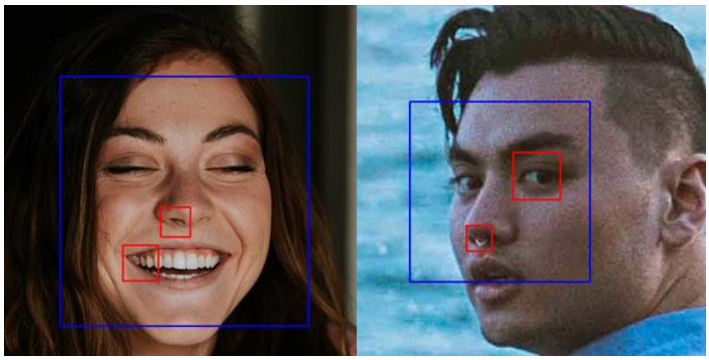
19. ábra: Az arc és szem felismerése az általam tréningezettel
Az eredeti kép forrása: *https://www.kaggle.com/ciplab/real-and-fake-face-detection*

A 16. ábrán látható, hogy a 15 stages 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.



20. ábra: Az arc és szem felismerése az általam tréningezettel
Az eredeti kép forrása: *https://www.kaggle.com/ciplab/real-and-fake-face-detection*

Ami számomra meglepetést okozott, hogy néhány képen jobban müködött az általam tréningezett, mint az openCV által tartalmazott. Megemlíteném, hogy a tréningezés során nem használtam ezt a képet. (17. ábrán látható)



21. ábra: Az arc és szem felismerése az általam tréningezettel
Az eredeti kép forrása: *https://www.kaggle.com/ciplab/real-and-fake-face-detection*

Amit a programba nem tudtam lekezelni az, az, ha a szem pozícióját az arcon belül rossz helyre teszi. (18. ábrán látható)

A csukott szemek azok, amit nem igazán ismer fel, valószínűleg a opencv által tartalmazott haar cascada tanítása alatt nem szerepelt a pozitív képek között. (18. ábrán látható)

Összegzés

A legvégén ***100 képre*** leteszteltem az általam tréningezetteket:

OpenCV Haar Cascade (OpenCV által tartalmazott):

- 74 %-os pontosságot ért el.
- Azokra a képekre nem működött jól, ahol forgatva volt az arc, vagy nagy részben eltakarta valami, illetve a csukott szemeket sem ismerte fel, a legtöbb szcenárióban.

Size 24 × 24, Stage 16:

- 35 %-os pontosságot ért el.
- Az eredményből látszik, hogy ilyen kevés képből tréningezve, nem lesz túl pontos a programunk.

Size 32 × 32, Stage 14:

- 40 %-os pontosságot ért el.
- Ezekkel a paraméterekkel sikerült a legjobb eredményt kihoznom, viszont még itt is kijön, hogy mennyire kevés a dataset a program mögött.

Azért azt megjegyezném itt a végén, hogy elég vegyes képeket használtam teszteléshez (változatos bőrszín, különböző háttér és fejtartás), próbáltam minél reálisabb eredményt kapni.

Összeségében kijelenthetem, hogy a 32x32-es működött a legjobban (az openCV-et leszámítva, vele hasonlítani nem lenne igazságos, hiszen jóval kevesebb erőforrás állt rendelkezésemre), ezek után talán a 16-os iterációs, a 15-ös gyakran érzékelt hibásan arcokat. (A 17,18,19,20 nem mutatott kiemelkedőt)

Irodalomjegyzék

[1] A. Kamdar, „GeeksforGeeks - Haar Cascade for Object Detection,” 25 november 2019. [Online]. Available: https://www.geeksforgeeks.org/python-haar-cascades-for-object-detection/.

[2] T. M. Varga Márton Bálint, „https://tdk.bme.hu/,” 13 október 2013. [Online]. Available: https://tdk.bme.hu/VIK/DownloadPaper/Valos-ideju-objektum-felismeres-gepi-latas. [Hozzáférés dátuma: 14 01 2020].

[3] N. Académia, „ Net Académia,” [Online]. Available: https://netacademia.hu/pages/a-haar-cascade-megertese. [Hozzáférés dátuma: 2020].

[4] A. Ahmadi, „ Cascade Trainer GUI,” [Online]. Available: https://amin-ahmadi.com/cascade-trainer-gui/. [Hozzáférés dátuma: 2020].

[5] „Wikipédia,” 26 október 2020. [Online]. Available: https://hu.wikipedia.org/wiki/Python_(programoz%C3%A1si_nyelv).

[6] O. t. Copyright © 2020, „OpenCv,” [Online]. Available: https://opencv.org/about/. [Hozzáférés dátuma: 2020].

[7] Kaggle, „Kaggle - real and fake face detection,” Kaggle, [Online]. Available: https://www.kaggle.com/ciplab/real-and-fake-face-detection. [Hozzáférés dátuma: 2020].

[8] Parwiz, „Codeloop,” [Online]. Available: https://codeloop.org/python-eye-detection-with-opencv/. [Hozzáférés dátuma: 2020].