Skach Bálint
VZXBJD

Integrált Rendszerek

Beadandó

Kameraalapú arcfelismerés OpenCV segítségével

Készítette

Skach Bálint
VZXBJD

Skach Bálint
VZXBJD
2018.12.09

Tartalomjegyzék

Skach Bálint
VZXBJD

1. Arcfelismerés

Az arcfelismerés az informatikai biztonság egyik fontos részét képezi, mivel a hagyományos azonosítási megoldások, mint például a jelszó alapú azonosítás, kezdenek elavulttá válni, ugyanis könnyű visszaélni velük, ha valaki megszerzi. A biometrikus azonosítás lényege, hogy az alany egy olyan tulajdonságát vagy tulajdonságait használjuk azonosításra, amellyel csak ő rendelkezik. Ilyen például az ember hangja, ujjlenyomata vagy éppen az arca. Természetesen ezekkel is vissza lehet élni, de ennek az esélye sokkal kisebb.

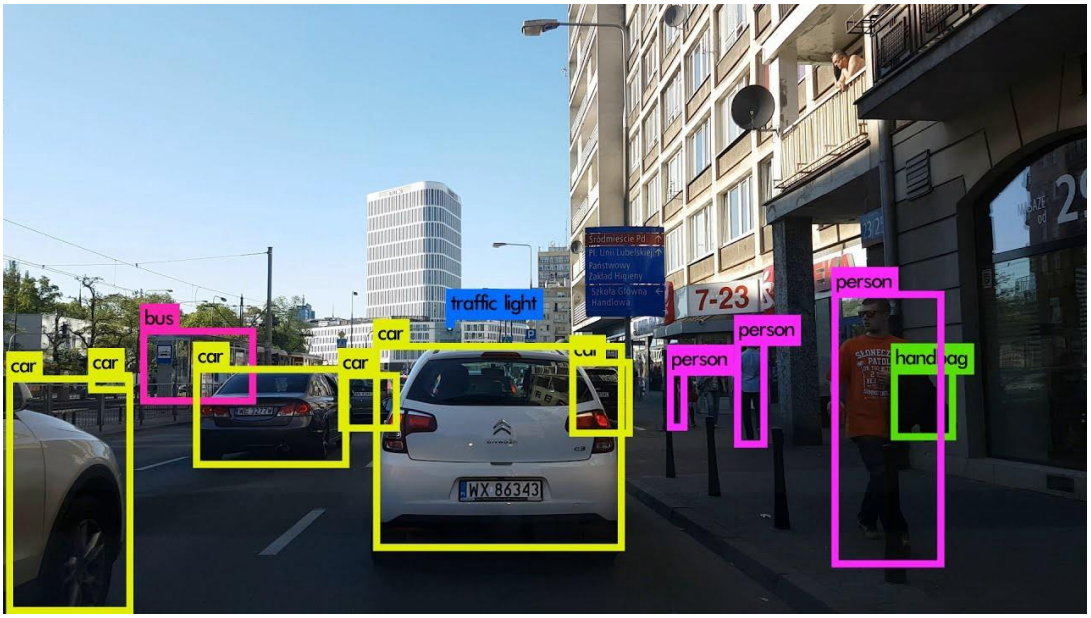
Dolgozatomban az arcfelismerés első lépését fogom bemutatni, melyet az angol Face Detection szó ír le a legjobban, magyarul talán az arcérzékelés lenne a megfelelő szó rá.

Skach Bálint
VZXBJD

Face Detection

Ahhoz, hogy azonosítani tudjunk egy képen lévő arcot, előbb fel kell ismernünk, hogy van-e a képen arc. Nekünk, az embereknek ez egy triviális folyamat, születésünk óta rendelkezünk ezzel a képességgel. Ellenben egy számítógépnek ez a feladat egy komoly kihívást jelent.

Az arcfelismerés (Face Detection) az objektumfelismerés egy részhalmazába tartozik, melynek felada, hogy egy adott képen meghatározza a keresendő osztályba tartozó objektum pozícióját és méretét. Általános eljárás, hogy ilyenkor egy színes téglalapot rajzol az objektum köré, jelezve nekünk, hogy megtalálta.



Objektum detektálás eredménye

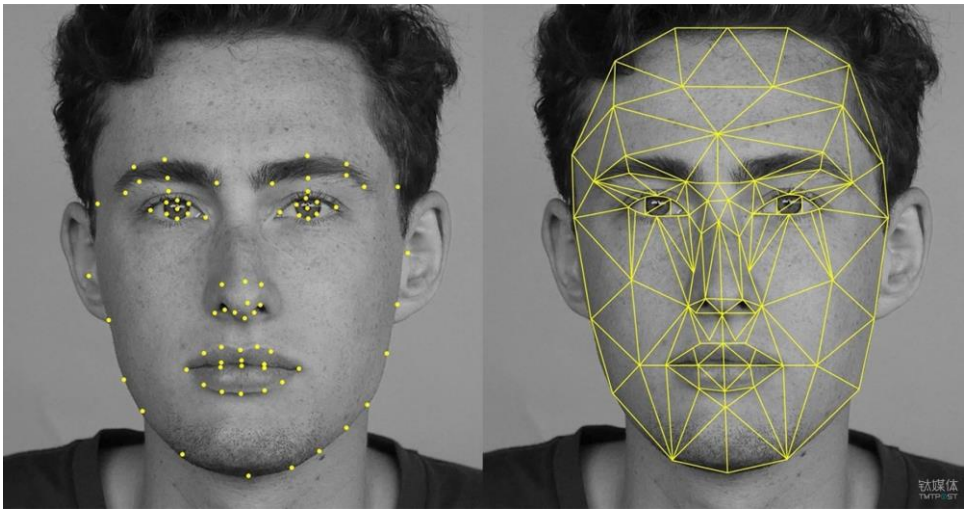
(forrás: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b)

Egyik leggyakoribb felhasználási területe, amivel már mindenki találkozhatott, az a kézi digitális kamerák arcfelismerő képessége, mely képes automatikusan fókuszálni a képen lévő arcokra.

Skach Bálint
VZXBJD
2018.12.09

Face Recognition

A Face Recognition, azaz a tényleges arcfelismerés szerepe, hogy megkülönböztessük az előző (Face Detection) lépésben felismert arcokat, az arcok tulajdonságait felismerjük és összevessük az adatbázisunkban található ismert arcok tulajdonságaival, így eldöntve, hogy a személy például jogosult-e a belépésre, vagy bejelentkezésre.



Face Recognition eredménye, mely jelöli az arc tulajdonságait
(forrás: https://blog.rapidapi.com/top-facial-recognition-apis)

OpenCV

Az OpenCV egy nyílt forráskódú gépi látást és gépi tanulást tartalmazó szoftver gyűjtemény. Több, mint 2500 optimalizált algoritmust tartalmaz, a régi klasszikusoktól kezdve az legújabb (state-of-the-art) algoritmusokig. Az OpenCV natív C++ környezetben lett feljeszve, és képes a hardveres gyorsításra is az OpenCL-nek köszönhetően. Számos interface elérhető hozzá, így használhatjuk Java vagy akár Python programunkban is. Több mint 47000-en használják, köztük a nagyobb cégek is, mint például a Google, a Microsoft, az IBM, a Honda és a Toyota.

Dolgozatomhoz én Python környezetben használtam az OpenCV-t.

Környezet kialakítása és telepítés

Mivel én Ubuntu operációs rendszert használok munkám során és otthoni személyes használatra is, ezért ennek a környezetnek a kialakítását fogom bemutatni.

Amire szükségünk van, az egy számítógép, egy webkamera és a következő programok:

- Ubuntu 18.04 LTS
- Python 3.6.7
- Numpy 1.15.4
- OpenCV 3.4.3

Az Ubuntu 18.04 LTS alapból tartalmazza a Python 3.6.7-et, így csak az OpenCV-t és a Numpy-t kell feltelepítenünk ezzel a két paranccsal:

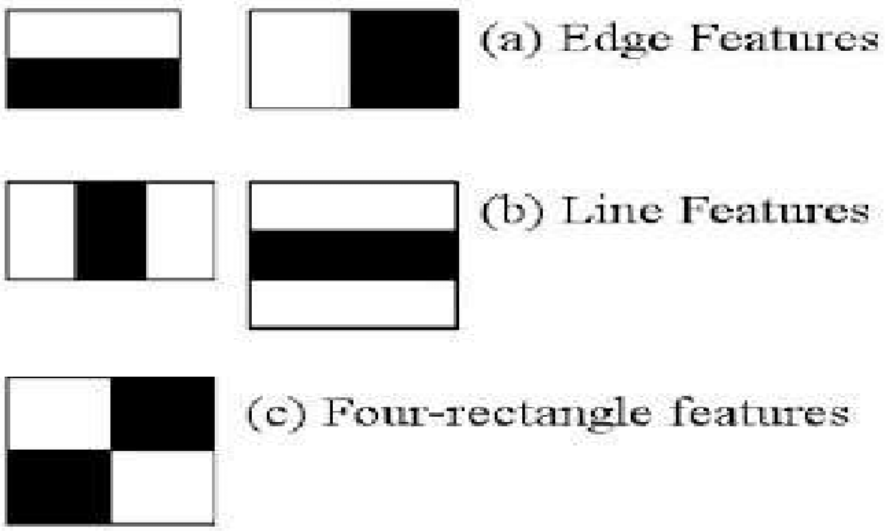
```
sudo apt install python-opencv
```

```
python3 –m pip install numpy
```

2. Haar-Cascade Classifier

Hogyan működik

A Haar-Cascade Classifier egy általános objektumfelismerő metódus, ami Haar tulajdonság-alapú kaszkás osztályozót (Haar feature-based cascade Classifier) használ. Ezt még 2001-ben Paul Viola és Micheal Jones találta ki, amit publikáltak is "Rapid Object Detection using a Boosted Cascade of Simple Features" néve.



Konvolúciós kernelek amiket a Haar Cascade Classifier használ

(forrás: https://www.researchgate.net/figure/Feature-Extraction-in-Haar-Cascade-Algorithm_fig2_325736109)

Az algoritmus úgy működik, hogy a konvolúciós kernelt rávetíti a kép bizonyos pontjaira, összeadja a fehér téglalapban lévő pixelek értéket és kivonja a fekete négyzet alatt lévő pixelek összegéből. Mivel az emberi arcnak vannak bizonyos általános tulajdonságai, mint például az, hogy a szem sötétebb, mint a körülötte lévő bőr, ezért egy ilyen kernellel könnyen azonosítani lehet az ilyen pontokat egy képen. Ha sok ilyen tulajdonságot összegyűjtünk, és készítünk hozzá kernelt, ami képes felismerni, akkor kapunk egy olyan eredményhalmazt, ami tartalmazza, hogy melyik tulajdonságra milyen eredményt kaptunk. Ezek után még lehet normalizálni ezt az adatot, például ha kapunk egy eredményt egy lehetséges szem pozíciójáról, és egy orr pozíciójáról, akkor ha megnézzük ezeknek az elhelyezkedését, és az orr alja a szem fölött helyezkedik el, akkor tudhatjuk, hogy vagy fordítva van az arc, vagy nem is arcot találtunk.

Skach Bálint
VZXBJD

Mi kell hozzá

Ahhoz, hogy egy Haar Cascade osztályozót készítsünk, szükségünk van minta képekre. Ezeket a képeket két csoportra oszthatjuk. Vannak a pozitív mintáink, melyek tartalmazzák a felismerendő objektumot, mely a mi esetünkben emberek arcai. A másik csoport a negatív minták, amit háttér mintáknak is szokás hívni. Ezek a képek bármit tartalmazhatnak, csak a felismerendő objektumot nem. Ideális esetben kétszer annyi pozitív mintának kell lennie, mint negatívnak.

Negatív minták

Negatív mintáknak én sima szobákról gyűjtöttem képeket, majd manuális töröltem azokat, amelyek tartalmaztak embereket.

A minták összegyűjtésére a https://image-net.org nevű oldalt használtam. Ezen az oldalon rengeteg kategória van, és ingyenesen le is tölthetőek egy szövegfájl formájában, mely az url-eket tartalmazza. Ezután már csak egy “wget” parancsot kellett kiadnom, és máris rendelkezésemre állt 500 negatív minta.

Pozitív minták

Pozitív minták a felismerendő objektumot tartalmazzák. Mivel ez a mi esetünkben emberi arcok, ezért én erről az oldalról töltöttem le az adatokat: http://vis-www.cs.umass.edu/lfw/ . Az itt elérhető zip fájl több, mint 18000 képet tartalmaz. Mivel nekem ennyire nem volt szükségem, ezért véletlenszerűen kiválasztottam belőle 1000 képet. Ezek a képek nem csak a személy arcát tartalmazzák, hanem a felsőtestüket is. Mivel nekem csak az arcokról kell a kép, ezért az OpenCV-hez tartozó alap Haar Cascade osztályozóval detektáltam őket, és körbe vágtam. Ez az 1000 képből 960-on talált arcot.

Minták előkészítése

Miután összegyűjtöttem a mintáimat, normalizáltam őket. Ez alatt azt értem, hogy a képek méretét egységesre állítottam, a negatív mintákat 100x100 pixelesre, a pozitívakat pedig 50x50 pixelesre. Ezután elkészítettem a szükséges fájlokat, melyek tartamazzák a pozitív és negatív minták számunkra szükséges adatait, amik a pozitív minták esetén a kép elérési útvonala, a képen található objektum helye, ebben a formában: X Y SZÉLESSÉG HOSSZÚSÁG. Mivel a pozitív mintáimat már leszűrtem, ezért mindegyik sornál csak az elérési útvonal különbözött. Egy teljes sor így néz ki:

pos/1.jpg 0 0 50 50

Ezeket az info.dat vagy info.list nevű fájlba kell tárolni. A negatív mintáknál még egyszerűbb a helyzet, ott csak az elérési útvonalat kell egy bg.txt nevű fájlba lementeni, tehát egy sora így néz ki:

neg/1.jpg

Vektor file készítése

A Vektor file tartalmazza a pozitív és negatív mintáinkból alkotott közös mintákat. Az OpenCV biztosít számunkra egy parancsot, amivel egyszerűen el tudjuk készíteni ezt a filet.

```
Usage: opencv_createsamples
  [-info <collection_file_name>]
  [-img <image_file_name>]
  [-vec <vec_file_name>]
  [-bg <background_file_name>]
  [-num <number_of_samples = 1000>]
  [-bgcolor <background_color = 0>]
  [-inv] [-randinv] [-bgthresh <background_color_threshold = 80>]
  [-maxidev <max_intensity_deviation = 40>]
  [-maxxangle <max_x_rotation_angle = 1.100000>]
  [-maxyangle <max_y_rotation_angle = 1.100000>]
  [-maxzangle <max_z_rotation_angle = 0.500000>]
  [-show [<scale = 4.000000>]]
  [-w <sample_width = 24>]
  [-h <sample_height = 24>]
  [-maxscale <max sample scale = -1.000000>]
  [-rngseed <rng seed = 12345>]
```

opencv_createsamples parancs, mely segítségével létrehozhatjuk a vector fájlt

Láthatjuk, hogy sok mindenre képes, de számunkra most csak az info, vec, bg, w és a h kapcsolók a fontosak. Az info kapcsolónak az info.dat fájlt adjuk meg, ami a pozitív mintáinkat tartalmazza. A vec kapcsoló lesz a kimenetünk. A bg kapcsolónak a negatív mintáinkat tartalmazó fájlt adtam meg, ami a bg.txt. A w és h kapcsolók pedig a kernel szélességét és hosszúságát adják meg, mindkettőt 24-re állítottam, mert az alap OpenCV-s osztályozó is ezeket a paramétereket használja.

Haar Cascade Classifier elkészítése

Hasonlóan az előzőhöz, az OpenCV ennél a lépésnél is biztosít számunkra egy parancsot, az opencv_traincascade-et.

```
Usage: opencv_traincascade
  -data <cascade_dir_name>
  -vec <vec_file_name>
  -bg <background_file_name>
  [-numPos <number_of_positive_samples = 2000>]
  [-numNeg <number_of_negative_samples = 1000>]
  [-numStages <number_of_stages = 20>]
  [-precalcValBufSize <precalculated_vals_buffer_size_in_Mb = 1024>]
  [-precalcIdxBufSize <precalculated_idxs_buffer_size_in_Mb = 1024>]
  [-baseFormatSave]
  [-numThreads <max_number_of_threads = 4>]
  [-acceptanceRatioBreakValue <value> = -1>]
--cascadeParams--
  [-stageType <BOOST(default)>]
  [-featureType <{HAAR(default), LBP, HOG}>]
  [-w <sampleWidth = 24>]
  [-h <sampleHeight = 24>]
--boostParams--
  [-bt <{DAB, RAB, LB, GAB(default)}>]
  [-minHitRate <min_hit_rate> = 0.995>]
  [-maxFalseAlarmRate <max_false_alarm_rate = 0.5>]
  [-weightTrimRate <weight_trim_rate = 0.95>]
  [-maxDepth <max_depth_of_weak_tree = 1>]
  [-maxWeakCount <max_weak_tree_count = 100>]
--haarFeatureParams--
  [-mode <BASIC(default) | CORE | ALL
--lbpFeatureParams--
--HOGFeatureParams--
```

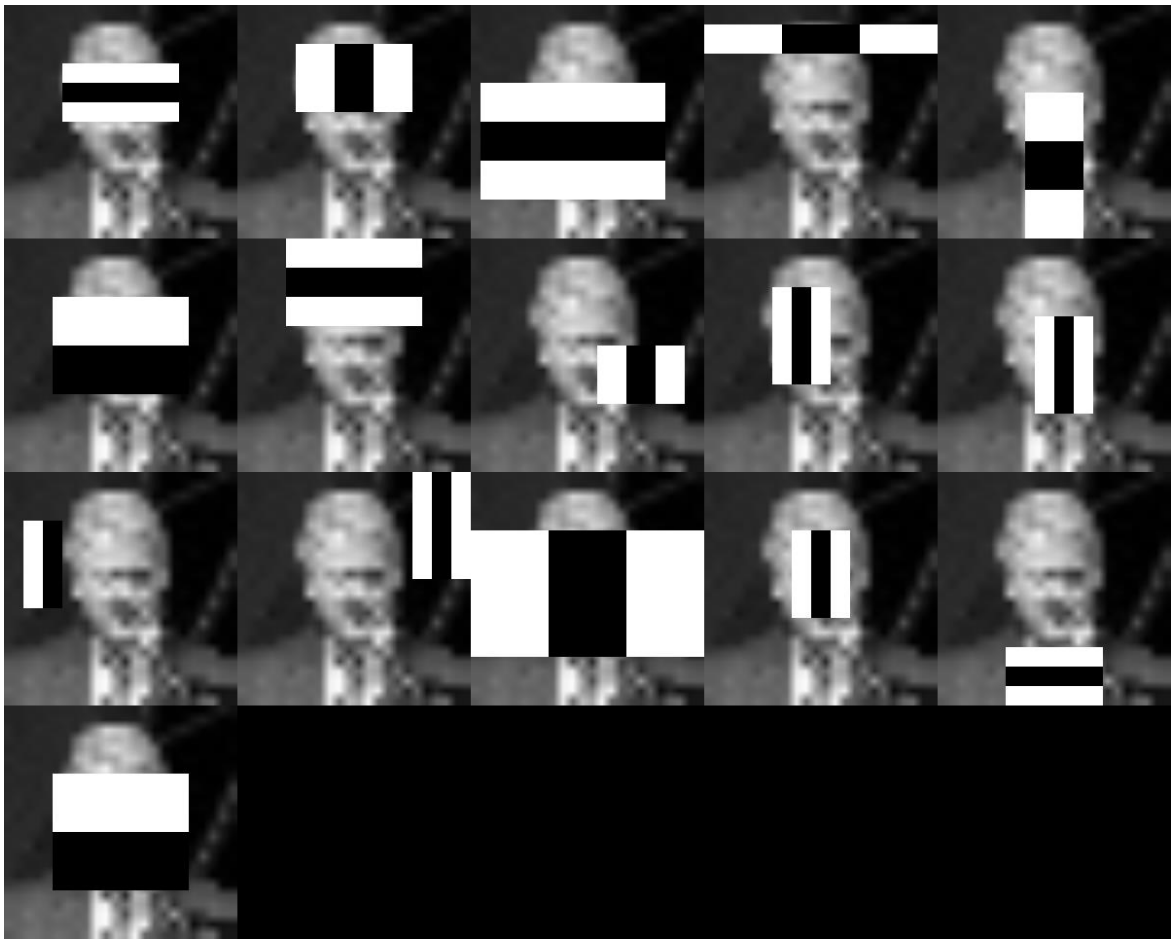
Opencv_traincascade parancs, mely segítségével létrehozhatjuk a modellünket

Ennél a lépésnél a következő kapcsolókat használtam. A data kapcsolót, melynél megadjuk, hogy hova készítse a modellt, tehát a kimenetünket. A vec kapcsolót, mely az előbb kreált vector fájlt használja. A bg kapcsolót, mely a negatív mintáink elérési útvonalát tartalmazza. A numPos kapcsolót, ez a pozitív mintáim száma. A numNeg kapcsolót, ami a negatív mintáim száma. Itt célszerű figyelni arra, hogy 2:1 arányú legyen a pozitív és negatív minták száma. A numStages kapcsolót, ami az iterációk számát adja meg. A w és h kapcsolókat, amik megint a kernel szélességét és hosszúgását adja meg, ezt megint 24x24-re állítottam.

Ennél a parancsnál figyelni kell, hogy milyen értékeket adunk meg, mert drasztikusan megnövelheti a modell generálásának idejét. Amikor a numStages-t 7-re állítottam, akkor a modellt pár másodperc alatt megalkotta, viszont magasabb értékeknél ez exponenciálisan növekszik. Elkezdtem egyet generálni 24-es numStages értékkel, de a 17. iteráció már 53 percig tartott, ezért ott leállítottam. Ha hagytam volna végig menni, akkor elvileg 9 napig tartott volna. Amit nagyon hasznosnak találtam ennél a parancsnál az az, hogyha félbeszakítok egy generálást, akkor ha átállítom a numStages értékét az utoljára elkészült iteráció értékére, akkor csinál abból egy modellt pár másodperc alatt. Ugyanígy folytatni is lehet egy félbeszakított generálást, nem kell újra generálni a már kész iterációkat.

Haar Cascade Classifier vizualizálása

Van még egy parancs, ami elérhető az OpenCV telepítésével, ami az opencv_visualisation. Ennek a célja, ahogy a nevéből is látható, hogy vizualizáljuk a modellünket. Kimenete egy videó fájl, amelyen látható, hogy iterációnként milyen tulajdonságokat keres egy képen. Ugyanezt lementi képkockánként is. Használata akkor a legeredményesebb, ha a mintakép mérete, megegyezik a kernel méretével. A mi esetünkben ez 24x24 pixel.



Az alap OpenCV Haar Cascade Classifier első iterációja vizualizálva

3. Haar Cascade Classifier alkalmazása

Miután elkészítettem a két modellem, nem maradt más hátra, mint használni őket. Használatuk rendkívül egyszerű, csak be kell őket költeni az opencv segítségével:

modell = cv.CascadeClassifier('eleresiutvonal_a_fajlhoz')

Ez után már csak használni kell egy képen, amit a modell.detectMultiScale(img, 1.3, 5) metódussal tudunk megtenni. Visszatérési értéke egy több dimenziós tömb, mely a felismert arcokat és a hozzájuk tartozó X, Y, SZÉLESSÉG, HOSSZÚSÁG értékeket tartalmazza. Ezeket használva két alkalmazást készítettem.

Képen történő arcfelismerés

Ez az alkalmazás egy input és egy output fájlt kér paraméterként. Célja, hogy az input fájlon található arcokat felismerje és köré rajzoljon egy téglalapot. Az alkalmazás három modellt használ, az OpenCV alap arcfelismerőjét és a sajátjaimat, amiket 7 és 17 iterációig futtattam. Az OpenCV-s piros, a saját 7 iterációs kék és a saját 17 iterációs zöld keretet kap.

```python
import cv2 as cv
import numpy as np
import sys

def recognize(img, out_path):

    default_faces = default.detectMultiScale(img, 1.3, 5)
    cascade_7_faces = cascade_7.detectMultiScale(img, 1.3, 5)
    cascade_17_faces = cascade_17.detectMultiScale(img, 1.3, 5)

    for (x,y,w,h) in default_faces:
        cv.rectangle(img, (x,y), (x+w, y+h), (0,0,255),2)

    for (x,y,w,h) in cascade_7_faces:
        cv.rectangle(img, (x,y), (x+w, y+h), (255,0,0),2)

    for (x,y,w,h) in cascade_17_faces:
        cv.rectangle(img, (x,y), (x+w, y+h), (0,255,0),2)

    cv.imwrite("results/" + out_path, img)

default = cv.CascadeClassifier('/home/skach/dev/facedet/haarcascade_frontalface_default.xml')
cascade_7 = cv.CascadeClassifier('/home/skach/dev/facedet/cascade.xml')
cascade_17 = cv.CascadeClassifier('/home/skach/dev/facedet/cascade_17.xml')

if (len(sys.argv) < 3):
    print("Usage: image_faceRec.py <image> <outfile>")
    sys.exit(2)
else:
    img_path = sys.argv[1]
    out_path = sys.argv[2]
    img = cv.imread(img_path, 1)
    recognize(img, out_path)
    sys.exit(0)
```

Image_facedet.py

Skach Bálint
VZXBJD

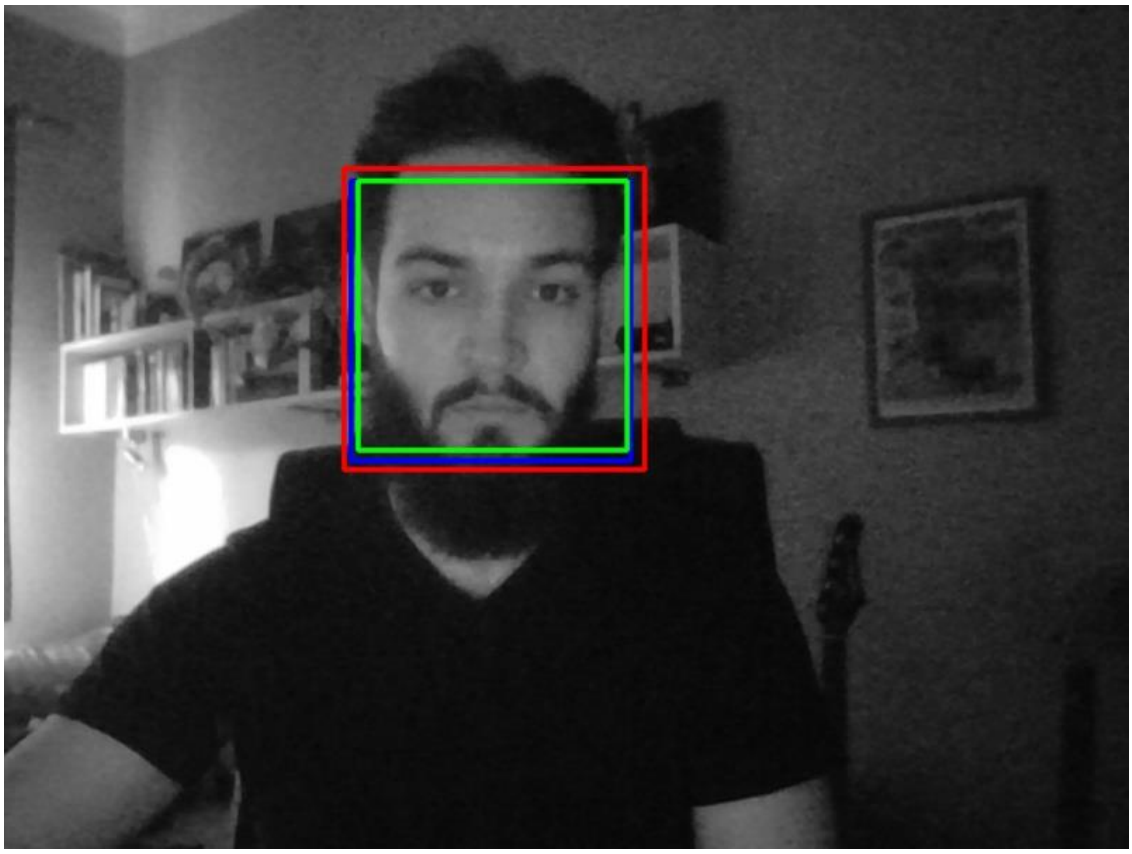
Real-Time kameraalapú arcfelismerés

Ez az alkalmazás a webkamera képét használja inputként, végrehajtja rajta mindhárom modell arcfelismerését, majd megjeleníti a képet. A p gomb megnyomásával készíthetünk pillanatképet, a q gombbal pedig kiléphetünk.

```
import cv2 as cv
import numpy as np
import os

def recognize(ret, frame):
    gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    colored_gray=cv.cvtColor(gray, cv.COLOR_GRAY2BGR)

    default_faces = default.detectMultiScale(gray, 1.3, 5)
    cascade_7_faces = cascade_7.detectMultiScale(gray, 1.3, 5)
    cascade_17_faces = cascade_17.detectMultiScale(gray, 1.3, 5)

    for (x,y,w,h) in default_faces:
        cv.rectangle(colored_gray, (x,y), (x+w, y+h), (0,0,255),2)

    for (x,y,w,h) in cascade_7_faces:
        cv.rectangle(colored_gray, (x,y), (x+w, y+h), (255,0,0),2)

    for (x,y,w,h) in cascade_17_faces:
        cv.rectangle(colored_gray, (x,y), (x+w, y+h), (0,255,0),2)


    resized = cv.resize(colored_gray, (800,600))
    cv.imshow('cap', resized)
    if cv.waitKey(1) & 0xFF == ord('p'):
        image_name= str(len(os.listdir('screenshots')))
        print('screenshot name:' + image_name)
        cv.imwrite('screenshots/' + image_name + '.jpg', resized)

default = cv.CascadeClassifier('/home/skach/dev/facedet/haarcascade_frontalface_default.xml')
cascade_7 = cv.CascadeClassifier('/home/skach/dev/facedet/cascade.xml')
cascade_17 = cv.CascadeClassifier('/home/skach/dev/facedet/cascade_17.xml')

cap = cv.VideoCapture(0)

while True:

    ret, frame = cap.read()
    recognize(ret, frame)

    if cv.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv.destroyAllWindows()
```

video_facedet.py

Skach Bálint
VZXBJD
2018.12.09



Egy pillanatkép, ahol látszik, hogy mindhárom modell felismerte az arcomat

4. Eredmények

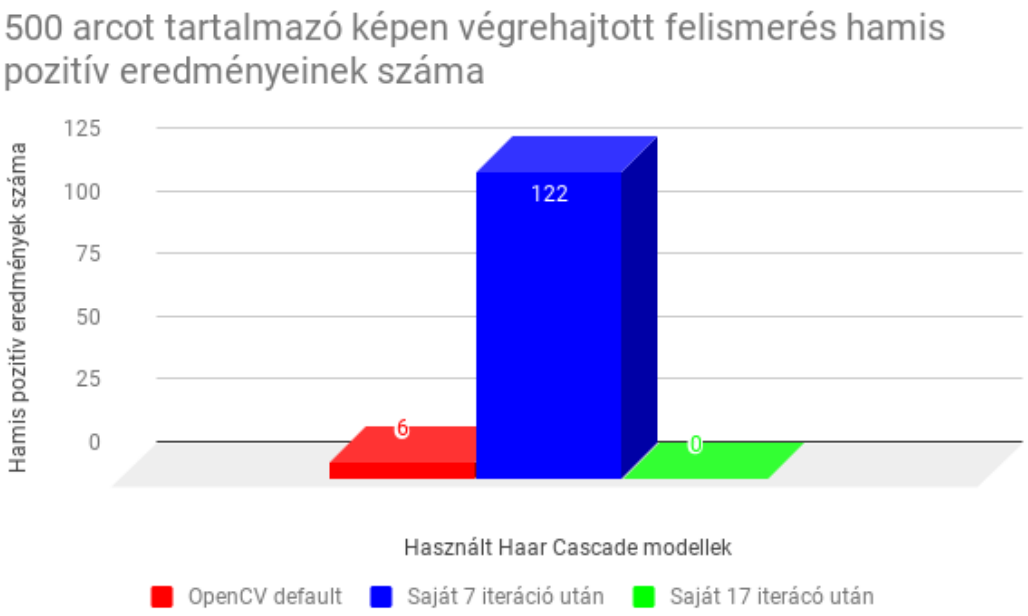
Modelljeim tesztelésére az eredeti pozitív mintáimhoz nyúltam. A 18000 közül kiválasztottam véletlenszerűen 500 darabot. Az image_facedet.py alkalmazásomból kiindulva írtam egy másikat, ami egy mappában lévő képekre futtatja az arcfelismerést, és eredményeit a színes keretekkel ellátott képek mellett egy csv fájlba is eltárolja. Ebben a csv fájlban azt irattam ki, hogy melyik képnél melyik modell hány arcot talált. Ezek után manuálisan ellenőriztem, ahol hamis pozitív eredmény volt, ott strigulát húztam.

Hamis pozitív eredmények

Az 500 kép közül az OpenCV-s modell összesen hatszor hibázott, azaz olyan objektumot ismert fel arcként, ami nem arc.

A 17 iterációs modellem egyet sem hibázott, bár nem is talált meg mindent, de ezt a következő pontban kifejtem.

A 7 iterációs modellem az 500 képen 573 arcot ismert fel, ez ösztönzött arra, hogy manuálisan ellenőrizzem az eredményeimet. Ez a modell 122 hamis pozitív eredményt adott.

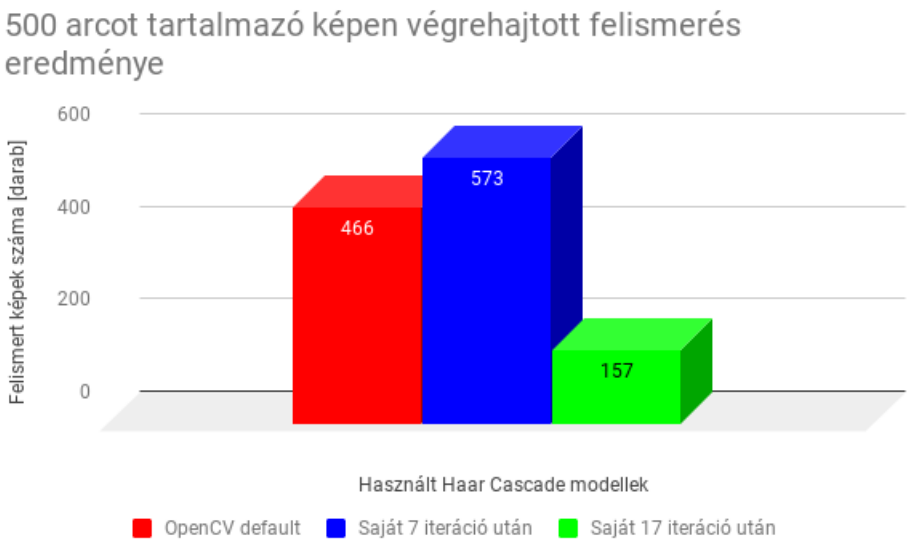


Pontosság

Az OpenCV-s modell az 500 képből 460-at jelölt meg helyesen, így 92%-os pontossággal rendelkezik. Ha a hamis pozitív/összeset nézzük, akkor 98,7%.

A 17 iterációs modellem bár nem jelölt meg egy képes sem hamisan, de helyesen nem teljesített jól. Az 500 képből 157-et jelölt meg helyesen, így 31,4%-os pontossággal bír. Ha a hamis pozitív/összeset nézzük, akkor 100%.

A 7 iterációs modellem, ha levonom az összes megtaláltból a hamis pozitívakat, akkor 451 képet jelölt meg helyesen, így 90,2%-os pontossággal bír. Ha a hamis pozitív/összeset nézzük, akkor 78,7%.



5. Összegzés

Ahhoz képest, hogy milyen kevés adatot és iterációt használtam, a 7 iterációs modellem egész szépen használható, bár nagyon érzékeny a környezetére.

A 17 iterációs modellem pedig nagyon tompa a környezetére, viszont olyan objektumot nem talál meg, ami nem arc.

A továbbiakban célszerű lenne más pozitív és negatív adatokkal kipróbálni, hogy csak az iterációbeli különbség milyen változásokat okozhat. Azt is érdemes lenne kipróbálni, hogy nagyobb mintákkal csökkenthető-e az alacsony iterációjú modellek érzékenysége.

6. Irodalmi jegyzék

- Face Detection and Recognition: Theory and Practice (Asit Kumar Datta, Madhura Datta, Pradipta Kumar Banerjee) 2015
- https://pythonprogramming.net

Skach Bálint 2018.12.09
VZXBJD