Dokumentáció

Készítette: Geiger Boldizsár (RP6OUG)

Előkészületek, bemutatás

Egyetemi beadandó feladatomnak egy olyan alkalmazásra esett a választás, amelyben az adott arc érzékelésére és feldolgozására kerül sor. A bemeneti érték lehet egy állókép, de igény szerint lehetőség van camera feedből kinyert információval dolgozni (ennek természetesen feltétele valamilyen USB-n csatlakoztatható vagy beépített rögzítő eszköz, webcam).

A projekt Python 3.6-ban készült, az alábbi könyvtárak felhasználásával:

- OpenCV
- NumPy
- dlib
- os
- glob
- PIL
- matplotlib

A megfelelő futás érdekében ajánlott az általános könyvtárakon kívül telepíteni a külsőket is, amik a listában más színnel kerültek jelölésre. A telepítés legegyszerűbben terminalon keresztül ajánlatos, mindegyik könyvtár elérhető hivatalos forrásból.

Először telepítsük a „PIP” (Pip Installs Packages)-et, egy cross-platform csomagkezelőt ami segít a Python csomagok installálásában:

Ezután „pip” által telepítsük a szükséges könyvtárakat, figyelve arra, hogy a megfelelő Python verzió mappájába kerüljön:

```
# pip3 install opencv-python numpy dlib pillow matplotlib
```

Következő lépésként klónozzuk a github repo-t, hogy az állomány futtatható legyen:

```
# git clone https://github.com/gboldi19/Background_removal
```

Amint ezekkel megvagyunk, a program futásra kész. Működésével kapcsolatos információkkal a dokumentum további részében találkozhat.

(Opcionális): Bár a repository tartalmazza, lehetőség van saját HAAR Cascade dataset letöltésére, ha nem feltétlenül a frontális arcdetektálás a cél.

Megvalósítás módszere

Mintaillesztés az egyik legegyszerűbb módszer, amely objektumok felismerését teszi lehetővé. Egyetlen problémája, hogy főként ideális-, majdnem laborkörnyezetben használható hatalmas pontossággal. Több tényező is ronthat a végeredményen, köztük zaj, távolság, fényviszonyok, egyéb objektum a kép látószögében, ami takar, hogy a detektálni kívánt felület vagy mozgásban van, élessége kérdéses, stb.

A mintaillesztés folyamán 2 képet hasonlítunk össze, az egyik a mintakép ami tartalmazza azt, amit keresünk, a másik pedig az a kép, ami azt tartalmazza amin keresünk.

Arc és arcélek detektálása tanított modellek (dataset) segítségével is lehetséges, minél nagyobb méretű egy ilyen állomány, annál jobban segítheti elő a végeredmény precízségét.

2 ilyen datasetet is alkalmaztam a program vázaként, az egyik Haar-szerű jellegzetességek alapján ismeri fel a kívánt objektumot, a másik HOG (Histogram of Oriented Gradients) jellegzetesség által.

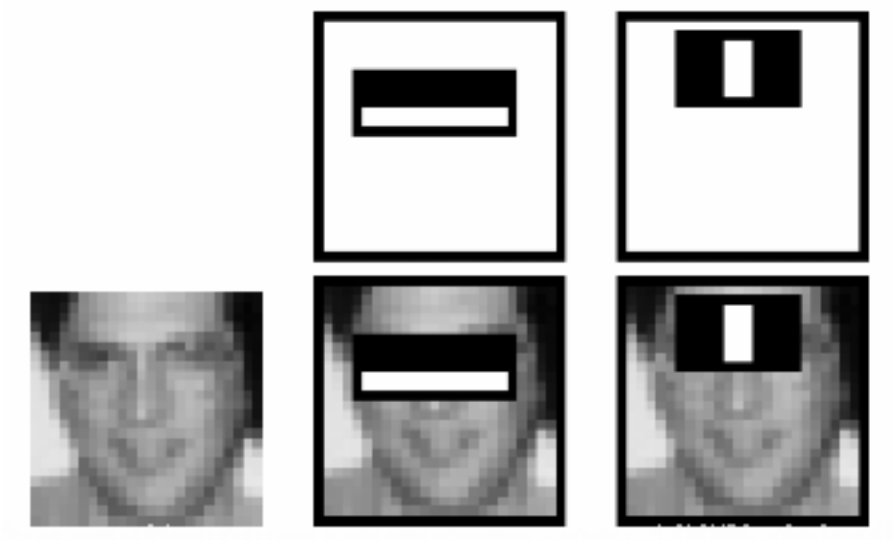
A kaszkád osztályozó alapú objektum felismerésnél felhasznált a Haar-szerű jellegzetességek (Haar-like features) nevüket a Haar wavelet-ekhez való hasonlóságuk kapcsán kapták. A módszer eredeti változatát Paul Viola és Michael Jones publikálták 2001-ben.

A technikát első sorban az arcfelismerés motiválta, ezzel is demonstrálták, és ezzel a technikával sikerült az első valós idejű arcfelismerők egyikét is megvalósítani, amely annak idején egy Intel Pentium III processzoron 15 képkocka/másodperc sebességgel futott. A publikált objektum felismerési keretrendszer a Haar-szerű jellegzetességek és három főbb ötlet felhasználásával képes elérni a gyors és hatékony objektum felismerést.

A technika nem közvetlenül kép intenzitás értékekkel dolgozik, hanem olyan jellegzetességek készletét használja fel, amelyek a Haar-wavelet-ekre emlékeztetnek, ezek a Haar-szerű jellegzetességek. A felismerési eljárás a képeket egyszerű jellegzetességek értékei alapján osztályozza. A jellegzetességek felhasználásának számos előnye van a hagyományos, a pixelek RGB értékeinek (intenzitásainak) közvetlen felhasználásához képest. Kizárólag pixeleken dolgozni 10 meglehetősen számításigényes folyamat, amelynél a jellegzetesség alapú rendszerek jelentősen gyorsabbak. Ezen túl a jellegzetességek rögzíthetik az olyan alkalmi ismereteket is, melyeket nehéz megtanulni véges számú tanítási adat alapján. A módszerben felhasznált egyszerű jellegzetességek a Haar wavelet-ekhez hasonlítanak. Egy Haar-szerű jellegzetesség szomszédos, azonos méretű és alakú téglalap alakú területeket vizsgál, az ilyen területeken kiszámítja a pixel intenzitások összegeit, és ezeknek az összegeknek a különbségeit vizsgálja (egymáshoz képest).

Ezen különbségek alapján az egyes területeket (egymáshoz képest) sötétnek vagy világosnak nyilvánítja, mely alapján a kép részeit kategorizálni tudja. Az eredeti publikációban három fajta jellegzetesség szerepel: úgynevezett kettő-téglalap, három-téglalap és négy-téglalap jellegzetesség, melyek 2, 3 illetve 4 téglalap alakú területet vizsgálnak. A kettő-téglalap jellegzetesség két horizontálisan vagy vertikálisan szomszédos területen vizsgálja a pixelek összegeinek a különbségét. A három-téglalap jellegzetesség két szélső terület összegének és a

köztük lévő középső terület összegének a különbségét vizsgálja. A négy-téglalap két-két átlós terület összegei közti különbséget vizsgálja.



Ezen jellegzetességek működésének szemléltetésére jó példa, hogy mivel egy emberi arcon a szemek régiója általában sötétebb az az alatti orca régiójánál, egy gyakori Haar-szerű jellegzetesség lesz egy kettő-téglalap, amelynek a felső része egy szem területén helyezkedik el és sötétebb, az alsó része pedig az arc területén van és világosabb. Egy adott osztályozóban felhasznált jellegzetesség jellemzői az alakja (az előbbi készletből), mérete és pozíciója. A detektor alapértelmezett 24×24 pixeles felbontásán elhelyezhető teljes készlete ezeknek a jellegzetességeknek így több, mint százezer.

A téglalap jellegzetességek meglehetősen primitívek sok más alternatívához képest. Ezek a jellegzetességek bár érzékenyek a sávokra, élekre és egyéb egyszerű struktúrákra, viszonylag durvának mondhatóak, önmagában gyenge osztályozók és nincs sok információtartalmuk. Téglalap jellegzetességek megfelelően nagy halmaza azonban már képes az objektum részletes, pontos jellemzésére. A Viola–Jones objektum-felismerési keretrendszerben az osztályozók halmazai kaszkádokba szerveződnek, hogy így együtt egy erős osztályozót alkossanak. A jellegzetességeknek a nagy száma és az integrál képek segítségével is biztosított hatékonysága kompenzálja az egyszerűségüket.

Program mechanizmusa

A program írása közben az egyik legnagyobb problémát számomra a mappaművelet jelentette, amire terminal-ban alkalmazható UNIX kódokkal orvosoltam.

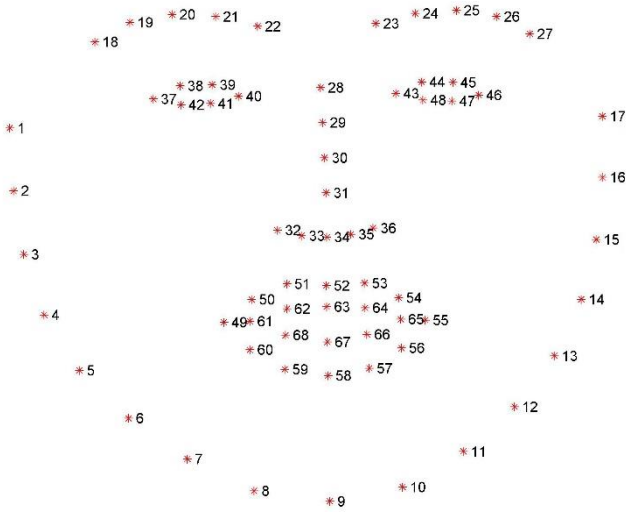
Ehhez elsősorban importáltam az „os” könyvtárat, ami segíti a kommunikációt a rendszerrel.

```
1  import cv2
2  import numpy as np
3  import dlib
4  import os
5  import glob
6  from PIL import Image
7  from matplotlib import pyplot as plt
```

Az importálás után deklaráltam a 2 mappa abszolút elérési útvonalát, illetve a dataseteket is includeoltam:

```
9  detector = dlib.get_frontal_face_detector()
10 predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
11 cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
12
13 sourcefold = "/home/gboldi19/Background_removal/gallery_src/"
14 destfold = "/home/gboldi19/Background_removal/gallery_dest/"
```

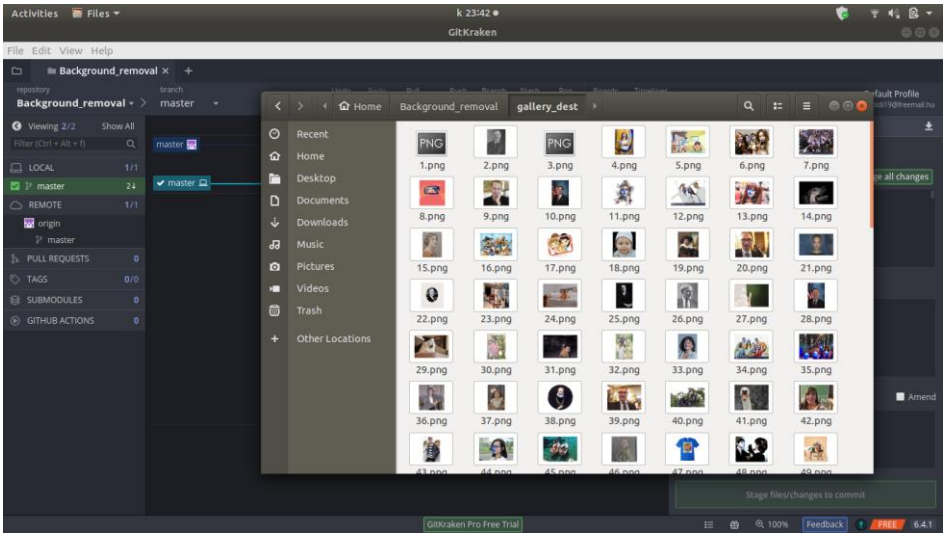
HOG jellegzetességhez a korábban is említett dlib-et használtam, amelynek van külön detektáló és illesztő függvénye is. Az illesztés arcon található bázispontok alapján működik, ami egy 68 pontból leírt halmaz, elemei a képen található mátrix pontjait és az azonos pixelek koordinátáit tartalmazzák:



Az alapvető dolgok előtöltése vagy inicializálása után a program ellenőrzi, hogy létezik-e célmappa, amibe az eredmények kerülnek másolásra. Amennyibe nem, abban az esetben az létrehozásra kerül.

```
118 if not os.path.exists(destfold):
119     os.makedirs(destfold)
```

A felhasználó eldöntheti, hogy tart igényt típus konverzióra vagy nem, ezt a program elején található **want_conversion** logikai változóval állíthatja. A konverzió folyamán az adott fájlnév és kiterjesztés elszeparálásra kerül, majd a kiterjesztés .png-re a fájlnév pedig az aktuális sorszámra változik.



Ezután indítható a fő program, ahol egy menü fogad bennünket, itt 4 féle funkció közül választhatunk. A kettő első a HOG féle detektálással kapcsolatos, míg a másik 2 HAAR féle tulajdonságok alapján észlel.

A legnagyobb különbség a kettő között, hogy míg a HAAR világos és sötét, mondhatni kontraszt-béli eltérést vizsgál, addig a HOG a színátmenet (grádiens) beli különbséggel számol, ami segít megadni az él határpontjai körüli irányt is. Ebből adódóan sokkal pontosabb, az elfordulásra, transzformációra és az objektum hely/helyzetváltoztatására nem annyira érzékeny, mint az előbbi metódus. A másik probléma, hogy mivel a detektálás HAAR cascade esetében csak 2 vagy 3 pont vizsgálatával történik nagyon sokszor ad false-positive eredményt.

Tesztelés

A program futásában a legtöbb időt a konverzió veszi igénybe. A futási idő jelentősen csökkenthető a detektálás során, ha a képek méretét redukáljuk, a képarány megtartásával, azonban fix magasság vagy szélesség megadásával.

A tesztelés folyamán törekedtem arra, hogy ne csak ideális körülményeknek megfelelő képet rakjak a forrásfájlok közé, ezzel tesztelve bármilyen false-positive eredményt. Ez lehetett nagyon rossz környezeti viszonyok között készült, filterezett vagy teljesen más tematikájú kép is.

A képeket Bing API segítségével fetcheltem 3 kereső szót alkalmazva, amik az alábbiak voltak: selfie, group selfie, portrait.

3 féle státuszt különböztet meg a program, az egyik amikor a kép színprofiljában található flag miatt nem olvasható, amikor az olvasás megtörtént, de arcot nem tudott észlelni az algoritmus, illetve amikor sikeresen olvasta be a képet és talált arcot is.

Az arcon található bázispontok felismerése után a program készített statisztikát az adott folyamatról, ami számunkra sok információval szolgál(hat).

```
Face_count:  213
On_count:  146 Face2_count 210
Face2_count 213
146 képen talált arcot a(z) 150 képből. Összesen 213 arcot talált!
Ez 97.33 %-os sikeresseget mutat!
```

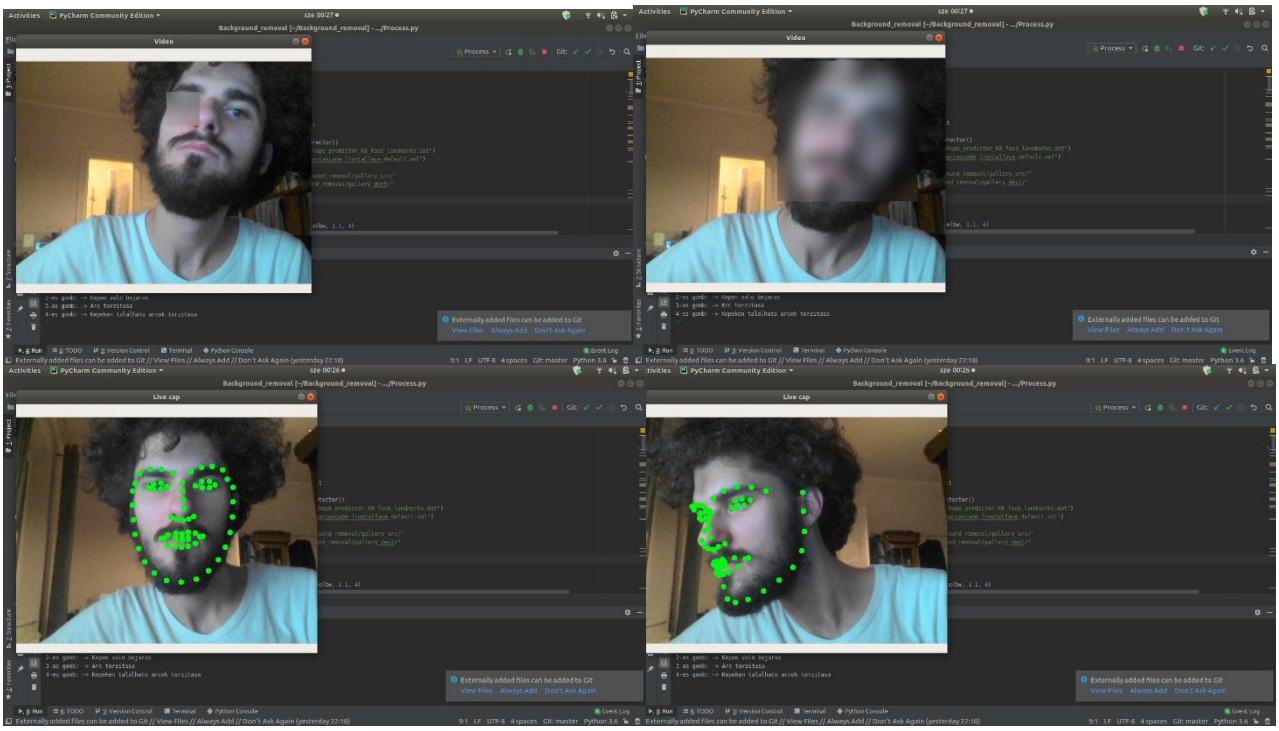
HAAR cascados észleléssel és a dlib segítségével is végigiteráltam a forrásmappán, pár érdekes eredményt csatolnék.

Balról jobbra, fentről lefele: 1. Forráskép, 2. HOG feature based detection, 3. HAAR cascade alapú észlelés majd Gaussian Blur-rel maszkolva. Itt a végeredmény majdnem azonos, azonban akad egy különbség. Jobb oldalon a lány keze és a másik leányzó feje/haja között figyelő urat a HOG-os megoldás nem észlelte. Ez amiatt következett be, mivel a teljes arcél nem volt kivehető, és csak akkor detektál, amennyibe mind a 68 pont rendelkezésre áll.

Egy másik eset, amikor az arcok jól kivehetőek, azonban a fényviszonyok vagy elfordulás akadályozhatja az észlelés pontosságát. Ilyenkor a HAAR Cascade teljesített rosszabbul:

A másik problémát a többször is említett false-positive találat jelentette HAAR tulajdonságok esetében, amely a jobb oldali képen, a leányzó nyakánál látható.

Az eset reprodukálható camera feeden keresztül történő detektálásnál is:



Felhasználói útmutató

A program futása előtt telepítsük a bevezetőben bemutatott állományokat. Telepítést követően terminálban vagy powershellben navigáljuk a mutatót a projektmappára. Amennyiben ez megtörtént utána írjuk be, hogy

```
#   python3 Project.py
```

A program elindul, és a célmappa létrehozása majd a képek másolása a célmappába automatikusan elindul, kérem várjon türelemmel a folyamat befejeztéig, egy üzenet értesíti amint végzett az adott procedúra.

Ezután a menüből szabadon kiválaszthatja az Ön számára ideális alpogramot. A választás számát (1 – 4) írja be, majd üssön Enter-t!

Amennyiben a kamerás észlelés vagy arc felületének torzítása mellett döntött, az adott ablak addig fog futni, ameddig az ’x’ gomb nem kerül megnyomásra a billentyűzeten. Ha a forrásképek detektálását vagy torzítását választotta, „gallery_dest” nevezetű mappában fogja találni a program futásának végeredményét.

Források, irodalom

- https://github.com/davisking/dlib
- https://www.pyimagesearch.com/
- https://www.learnopencv.com/histogram-of-oriented-gradients/
- https://docs.opencv.org/
- https://www.geeksforgeeks.org/python-haar-cascades-for-object-detection/