**Gépi Látás**

Security Camera

**Tartalomjegyzék:**

* Projekt Leírás
  + Definíció
  + Elvárások
* Felhasználói Dokumentáció
  + Használati utasítás
  + Extrák leírása
    - Email-küldés
* Fejlesztői Dokumentáció
  + Algoritmusok
    - HOG (Histogram Of Oriented Gradients)
    - Haarcascade algoritmus
      * AdaBoost
      * Cascade of Classifiers
        + Működése
    - Saját arckövető algoritmus
  + Tesztek
* Extrák
  + Email-küldés
  + Geolokáció meghatározása
* Források

**Projekt Leírás**

Kamera videófelvételen egyidejűleg egy mozgás felismerése és nyomon követése.

**Felhasználói Dokumentáció**

A <https://github.com/tzsombi01/Security_Camera> linkre kattintva láthatjuk a repositoryt, ahol a projekt található. Itt a A képen szöveg látható

Automatikusan generált leírás

„< > Code”-ra kattintva másoljuk ki a linket.

Parancssoron keresztül navigáljunk a mappához, amiben el szeretnénk helyezni az állományokat.

Például: cd „C:\Users\username\PycharmProjects” , itt a username helyett a saját gépünkön levő felhasználónevet írjuk be.

Futtassuk a „git clone <https://github.com/tzsombi01/Security_Camera.git>” parancsot, majd a Security\_Camera mappában találjuk meg az állományt.

A továbbiakban a PyCharm IDE-n keresztül mutatom be az indítás további lépéseit, ugyanakkor a parancssoros megfelelőit is megmutatom.

A szükséges egyéb állományokat töltsük le a „python packages” segítségével.



Töltsük le az opencv-python, numpy, illetve az urllib3 állományokat.

A képen szöveg látható

Automatikusan generált leírás



A felületen jobb felül található „Play” gombra kattintva futtathatjuk a main file-t, az elérési utat mellette adhatjuk meg.

A képen szöveg, képernyőkép, monitor, képernyő látható

Automatikusan generált leírás1: Elérési útja a main filenak, paraméter

2: Elérési útja a „haarcascade\_frontalface\_default.xml” állománynak

Meg kell adni paraméterként az elérési utat a haarcascade\_frontalface\_default.xml is, ezzel futtathatjuk a programot!

Parancssoros megfelelők:

main.py script futtatása:

username -> A saját felhasználóneved

Lépjünk be a scriptet tartalmazó mappába:

**cd C:\Users\username\PycharmProjects\Security\_Camera\src\**

Majd futtassuk a scriptet, bemeneti paraméterrel, mely az elérési útja a „haarcascade\_frontalface\_default.xml” állománynak.

**python main.py –face\_cascade „elérési\_út.xml”**

A képen szöveg, személy, beltéri látható

Automatikusan generált leírás

Már indulhat is a felvétel!

**Fejlesztői Dokumentáció**

A fejlesztés során többfajta algoritmus használata felmerült az emberi arcok, illetve az emberi testek detektálására.

Általában a sebesség volt a döntő faktor a választásban.

Fontos kiemelni, hogy mivel felvételt is készítünk, így elengedhetetlen a (relatív) magas fps szám futásnál.

**HOG (Histogram of Oriented Gradients)**

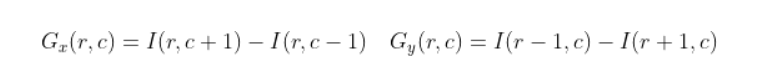
A HOG (Histogram of Oriented Gradients) algoritmus a sebessége miatt végül nem került használatba.

Az algoritmus működése:

A kép újraméretezése után (általában méretcsökkentés)

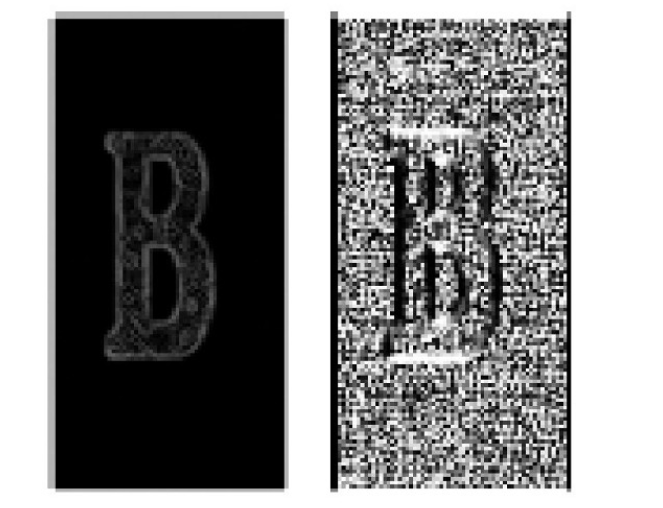
Grayscale (szürkeskálás) kép készítése

Kiszámoljuk a **Gradiens**eket, a következő képlet segítségével:



r a row (sor)-ra, míg c a column (azaz oszlop)-ot jelenti.

Ezután Magnitúdot számolunk.



8x8-as blokkokat formázunk, és minden blokkra egy 9 pontos hisztogram kerül kiszámításra.

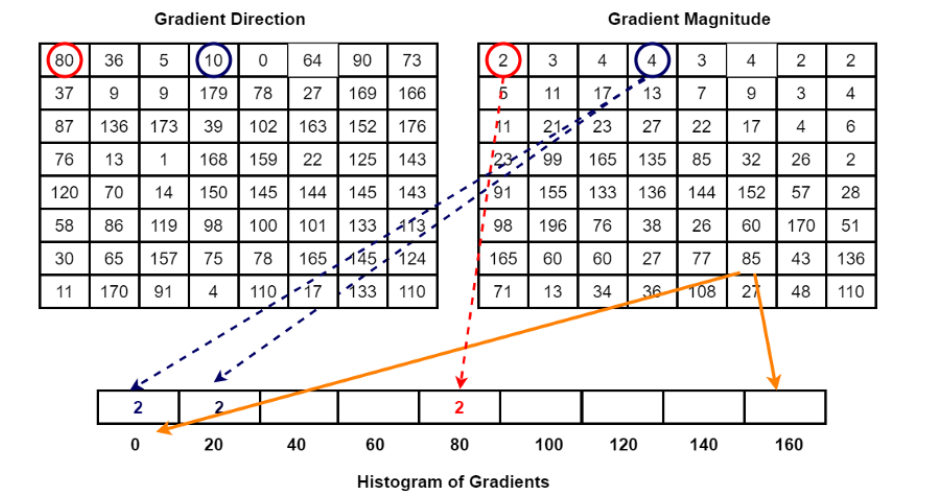
A képen szöveg látható

Automatikusan generált leírás

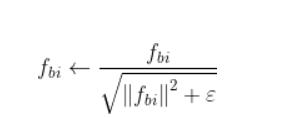
Ezeken belül egy ún. „bin”-t számolunk, és a j-edik bin-t illetve a j+1-edik bin-t felhasználva egy számolással, az értékeket pedig egy arrayben (listában) tároljuk.

Egy array-re az adott blokk (8x8) binjeként tekintünk, ahol az array j és j+1edik értéke a bin j és j+1edik kalkulált értékével egyezik meg.

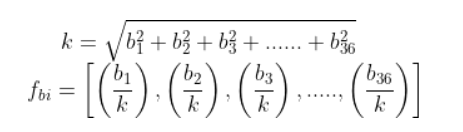
Ha minden blokk készen van, 4 blokk a 9 pontos hisztogramból együtt fognak egy 2x2-es blokkot formálni.



Ezután normalizálunk azL2 normalizációs formulával.



Ehhez segítségül ki kell számolnunk a k-t, hogy megkapjuk fbi-t.



A normalizáció kell, hogy csökkentsük a kontrasztot.

A képen szöveg látható

Automatikusan generált leírás

A HOG jellemzők így egy 128x64-es kezdőképre: 7 x 15 x 36 = **3780 db** jellemző.

Ugyanolyan eredményes volt a testek detektálásában mint a Haarcascade detektáló algoritmus, ugyanakkor az FPS megszenvedte volna a használatát.

HOG általam használt paraméterei:

* winStride: (10, 10)
* padding: (20, 20)
* scale: 1.075

A HOG algoritmushoz az opencv package-ben elérhető cv2.HOGDescriptor\_getDefaultPeopleDetector()-t használtam.



**Haarcascade algoritmus**

Végül erre az algoritmusra esett a választás az egyszerűsége, illetve a sebessége miatt.

Az algoritmus elmélete 3 dolgon alapul:

* Edge Features (Éljellemzők)
* Line Features (Vonaljellemzők)
* Four-Rectangle Features (Négyszögjellemzők)

A képen szöveg látható

Automatikusan generált leírás

**AdaBoost (Adaptive Boosting)**

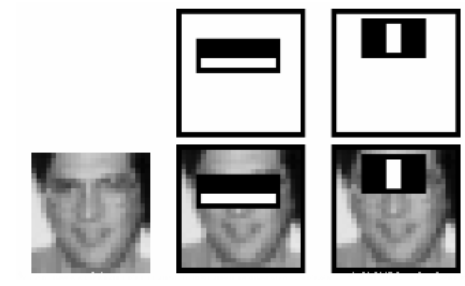
Lényegében a legjobb jellemzőket (feature) segít kiválasztani a képenkénti akár 16000+ jellemzőből.

„Weak classifier”-ekből (gyenge klasszifikáló) állít össze egy „Strong classifier”-t (erős klasszifikálót).

A „gyenge klasszifikálók” lényegében az individuális feature-ök, azonban ha ezeknek vesszük a súlyozott összegét, akkor már erős klasszifikáció-t alkotnak, alkalmasak lesznek a képek azonosítására.

Ennek segítségével csökkenti az algoritmus az „overfitting”-et, vagyis a „túlillesztést”.

Overfitting esetében túlságosan a teszt képekre alakítja a rendszer a súlyokat és nem képes (nagy hibaszázalékkal) azonosítja a kívánt alakzatokat a valós bemeneteken.



Ezen felül az algoritmus Classifierekből nem csak egyet, vagy különálló egyedeket alkalmaz. Úgynevezett **Cascade of Classifiereket**, vagyis Classifierek egy csoportját használja fel a felismerésre.

**De hogy is működik ez?**

Ha van egy 24x24-es képünk, és 6000 tulajdonságra akarjuk vizsgálni, méghozzá úgy, hogy meg tudjuk, mondani, hogy arc-e vagy sem.

A tulajdonságokat csoportokba (szintekbe) rendezzük, ahol minden szinten van 1, 10, 15, 20, 25 … tulajdonság.

Majd a képet minden egyes szinten átküldjük a tulajdonságokon (filtereken).

Amennyiben akár egy szinten is megakad, nem megy át a kép, és nem értékeljük arcnak.

Így gyorsítható a nem-arcoknak a kiértékelése.

A Haarcascade algoritmus paraméterei a következők voltak:

* scale\_factor: 1,3
* overlap: 6

Az opencv-modul tartalmaz előre súlyozott modelleket, melyet felhasználva tudunk alakzatokat felismerni.

Én a haarcascade\_frontalface\_default.xml, illetve a haarcascade.fullbody.xml állományokat használtam.

**Tesztek**

Teszteket implementáltam az arcdetektálásra, a dátum formátumára, illetve arra, hogy a felvétel jókor érjen véget illetve voltak tesztek a végül fel nem használt HOG, illetve Haarcascade emberdetektálásra de ezek itt nem kerülnek bemutatásra.

*Futtatásuk*:

PyCharm IDE-ből:

A képen szöveg, képernyőkép, monitor, fekete látható

Automatikusan generált leírás

A konfigurálás utána a megfelelő opciót kiválasztva (Unittests), futtassuk a „play” gombbal.

A képen szöveg, eszköz, méter, mérce látható

Automatikusan generált leírás

Parancssorból (a példákban git bash-t használok, de cmd-ra írtam a parancsokat):

Navigáljunk a mappához, amiben a tesztek vannak, jelen esetben ez a src (source mappa).

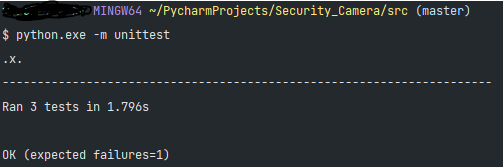
***cd src/***

A képen szöveg látható

Automatikusan generált leírás

Majd futtassuk a:

***python3.X -m unittest test.py*** parancsot, ahol a python 3.X helyett a saját python verziónkat írjuk be, például Python3.8.



Ha ki akarjuk írtani részletesen a tesztek eredményeit, használjuk a -v (verbose) kiegészítést:

***python3.X -m unittest -v test.py***

A képen szöveg látható

Automatikusan generált leírás

**Arcdetektálás**

Minden teszt képen átmegy az algoritmus, kivéve ott, ahol a fej oldalt, vagy enyhén lefelé hajtva van.

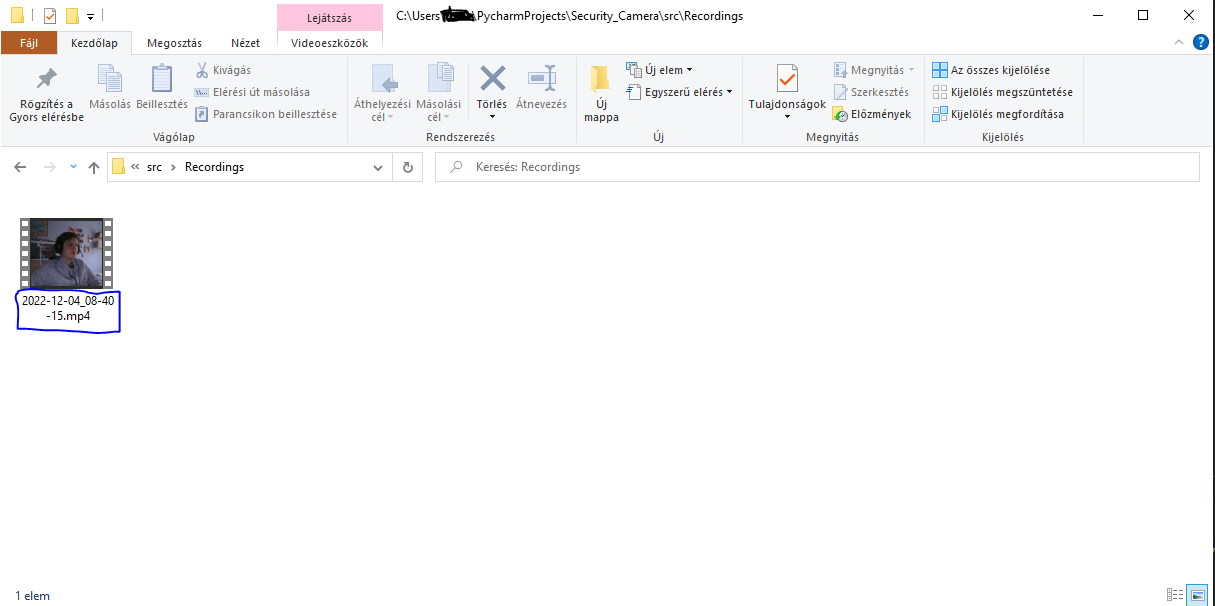
Próbálkoztam különböző beállításokkal, de végül ezek a paraméterek bizonyultak a leghasznosabbnak a leggyorsabb teljesítménnyel ötvözve.

**Dátumformátum**

Nem lehetnek a lekért dátumformátumban benne illegális karakterek.

A tesztek biztosítják, hogy ha később módosítani is szeretnénk a formátumon, ne tehessük azt hibásan.

Ez főleg a fájlok eltárolásánál fontos, mivel ha például van egy illegális karakter, nem dob megfelelő errort a gép, így nehéz megállapítani a hiba okát.



**Felvétel vége**

Az utolsó frametől számítva, ahol arcot detektáltunk, pontosan a beállított idő elteltével állítsa le a felvételt.

**Arckövető algoritmus**

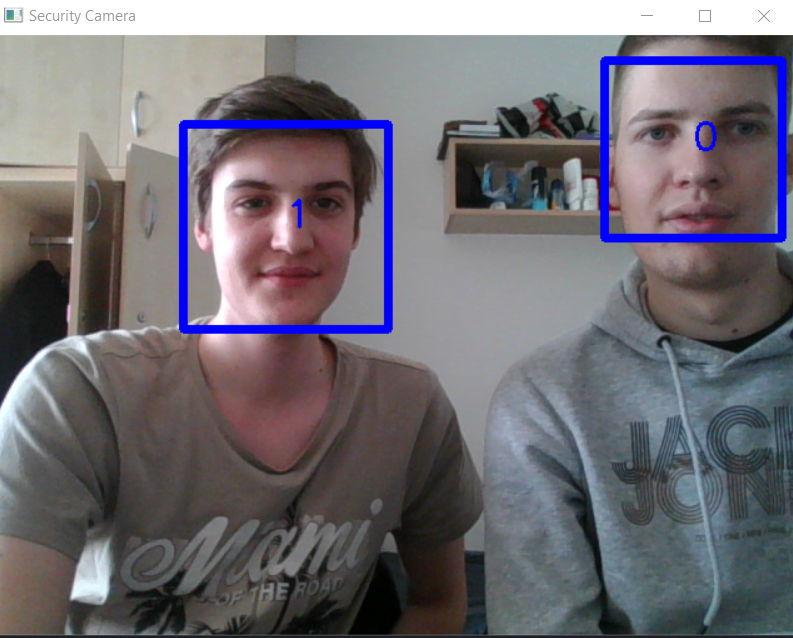
A kitűzött cél eléréséhez implementálni kellett egy arckövető algoritmust is. Úgy döntöttem, hogy nem egy „létező”, előre elkészített algoritmust használok fel, hanem sajátot készítek.

A logikája rendkívül egyszerű, ellenben pont ezért relatív gyors is.

Ha felismertünk egy arcot, annak a középpontját, illetve az „ID”-ját eltároljuk.

A következő frame kezdetekor újabb arcfelismerés, majd összehasonlítjuk az előző frame-el a kapott középpontokat. Amennyiben egy középpont a thresholdnál, azaz egy beállított értéknél többel mozdul el, új ID-t kap, amennyiben nem egyezik(~threshold-nyi távolságba esik) egy másik arctól.

Probléma az algoritmus esetében a több framen való arckövetés, ez egy hasznos feature lenne a továbbfejlesztés szempontjából.



**Extrák**

A funkciókat külön mappában helyeztem el, a jól látható elkülönítés céljából.

**Email küldés**

Implementáltam egy funkciót, mely segítségével a „kamera” behatoló (emberi arc) észlelése esetén küld egy figyelmeztető emailt timestamppel (időbélyeggel) ellátva.

Ehhez töltsük le a package managerrel az smtplib, illetve az email.message packageket, ebből az Emailmessage állomány lesz felhasználva.

**Geolokáció meghatározása**

A végleges programba nem került bele, ugyanakkor említést teszek a geolokációt IP cím alapján meghatározó függvényről.

Az emailben ennek segítségével kaphattunk helyzeti információt a kameráról.

A funkció használatához a package managerrel töltsük le a geopy csomagot, ebből majd a Nominatim API-t fogjuk használni.

**Források:**

* **Opencv Documentation**
  + Ana, Huamán, (2018)
  + cv::CascadeClassifier Library
  + Cascade Classifier