Az OpenCV Machine Learning moduljának megismerése

Készítette:

Sümegi Márk, Szabó Attila

Konzulens:

Csorba Kristóf

(2017)

BSc Önálló laboratórium - Összefoglaló dokumentum

A téma leírása

A feladat az OpenCV valós idejű képfeldolgozásra kifejlesztett függvénykönyvtár gépi tanulással kapcsolatos funkcióinak áttekintése és egy közösen kiválasztott feladat megoldása ennek segítségével. A fejlesztő környezet C#.

Lehetőségek:

- Osztályozó algoritmusok felhasználása
- Geometriai alakzatok tulajdonságainak kinyerése, azok megkülönböztetése
- Klaszterező algoritmusok (tanítóhalmaz nélküli tanulás)
- Képfeldolgozó algoritmusok eredményének automatikus értékelése és ez alapján az algoritmus paramétereinek hangolása

A projekt bemutatása

A projekt az AUT által fejlesztett cv4sensorhub keretrendszer különböző felhasználási módjainak és funkcióinak Machine Learning megoldásokkal való kibővítése.

A cv4sensorhub egy interaktív kép és videófeldolgozó framework, ami azzal a céllal született, hogy képfeldolgozást is tartalmazó kutatási és kutatás-fejlesztési projektek számára nyújtson egy stabil alapot, melyben minél több komponens előre rendelkezésre áll. Így új alkalmazások fejlesztésekor nem szükséges nulláról implementálni az algoritmusokat és a kezelő felületet.

A keretrendszerre épülő modulok az orvosi felhasználású és időkritikus valós idejű alkalmazásoktól egészen a játékokig terjednek. Ennek az egyik része, amelyen első sorban dolgoztunk, egy ipari felhasználású képelemző szoftver funkcióinak kiegészítése machine learning alapú algoritmusokkal, hogy automatizálja a feldolgozást. A szoftver feladatai közé tartozik többek között, a különböző anyagcsiszolatok keresztmetszetéről készült képek analizálása (például beton szemcsésségének mérése), továbbá márványcsiszolatok mintájának elemzése.

A mi munkánk a beton belső struktúrájának elemzése, a benne található különböző anyagok, pórusok, légbuborékok és szennyeződések felismerésére és kategorizálására irányult.

OpenCV Machine Learning Modul képességei

Az ml modul a keretrendszer gépi tanulással foglalkozó könyvtára, amely osztályozási feladatokhoz, regressziós problémákhoz és clustering algoritmusokhoz nyújt függvényeket és osztályokat.

Osztályozás

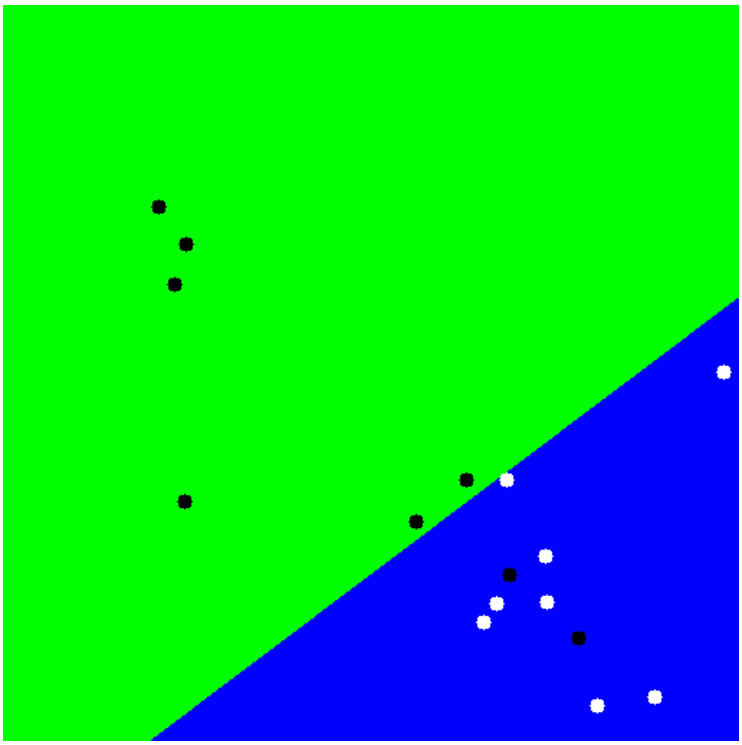
Ha ismert kategóriákba szeretnék csoportosítani az adatokat valamely tulajdonságok (feature) alapján, akkor alapvetően classification, azaz osztályozási problémáról beszélünk. Tehát jelen esetben ismert, hogy milyen szemcsék léteznek, az azokra jellemző adatok pedig kinyerhetőek. A legtöbb osztályozó algoritmus a supervised learning, azaz felügyelt tanulás elvén müködik.

Ekkor minden adathoz ismert egy címke (label), amely megmondja, hogy melyik osztályhoz tartozik. A tanulás során az algoritmus a megadott tulajdonságok alapján dolgozik, azokban keres olyan jellemzőket, amelyek alapján a későbbiekben már a címke nélküli adatokra próbálja meg kitalálni.

Az OpenCV több megoldást is támogat, pl. Döntési fák, k-NearestNeighbour, illetve SVM rendszerek. Mi az utóbbival foglalkoztunk mélyebben.

Support Vector Machine

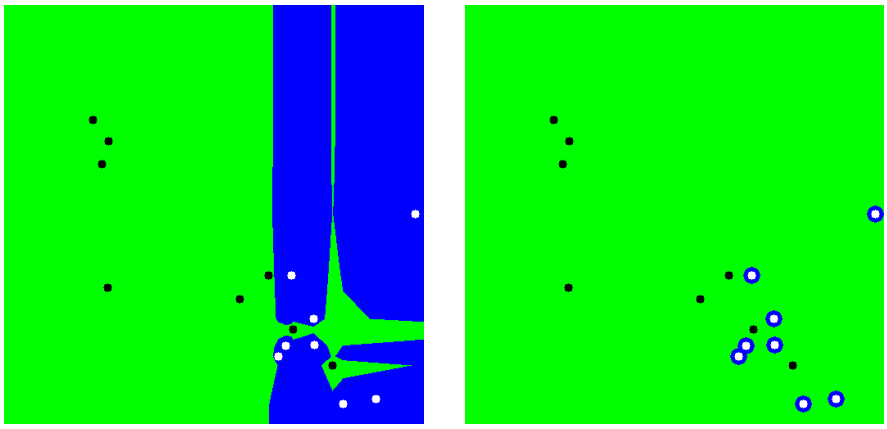
A support vector machine (SVM) egy supervised learning modell, amivel adatokat tudunk osztályozni. Adott egy már “felcímkézett” adatvektorok halmaza, amik között az algoritmus a legkissebb veszteségű elvágó hipersíkot találja meg. Ezt könnyen vizualizálhatjuk abban a speciális esetben amikor egy síkon lévő pontok halmazát szeretnénk két csoportba osztani:



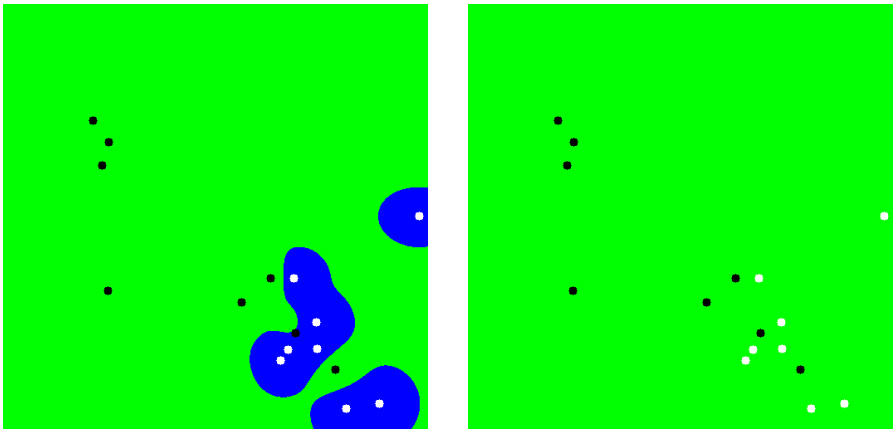
A fenti példában a fekete pontok tartoznak “A” kategóriába, a fehér pontok pedig “B“ kategóriába. A zöld felület jelzi azon pontok halmazát amit a SVM az “A” kategóriába sorolna, a kék pedig amit “B”-be.

Látható, hogy az adathalmaz nem linárisan szeparálható, tehát egy egyenes vonallal nem lehet tökéletesen kettéosztani (vagyis, hogy minden fekete a zöld felületen ls minden fehér a kéken legyen). Ilyenkor különböző kernel funkciókkal pontosíthatjuk az osztályozást, amivel a tanítóadatainkra pontosabb eredményt érhetünk el, de ezzel lehet, hogy a jövőbeli prediction pontossága csökken (lásd overfitting).

Beépített OpenCV kernelfüggvények:



(Bal: inter, Jobb: rbf)



(Bal: Chi2, Jobb: sigmoid)

Clustering

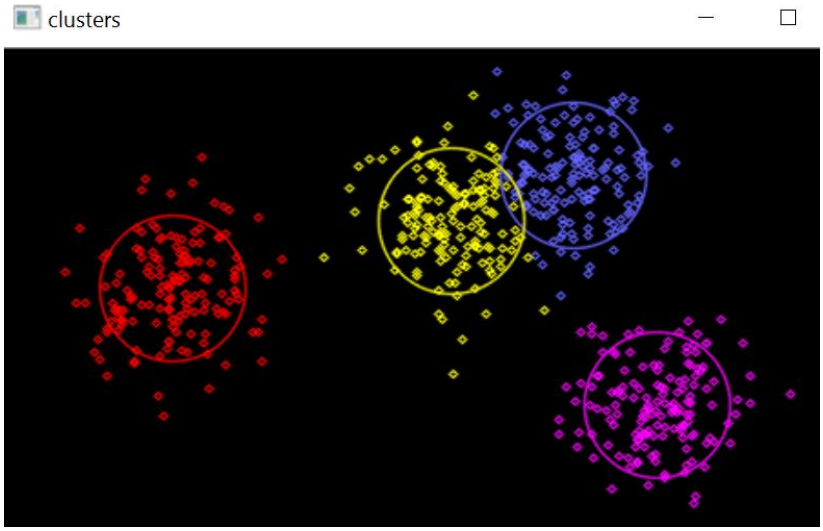
Ha előre nem ismerjük a kategóriákat, akkor az adatokban található hasonlóságok alapján létrehozhatjuk azokat a csoportokat, amelyekre felbontható az adathalmaz. Eredményül megkapjuk, hogy hány osztályunk lett, illetve ezekbe be is osztja az algoritmus az adatokat.

Emelett nincsenek címkék, tehát unsupervised learning, azaz felügyelet nélküli tanulás. Ezeket az eljárásokat nevezzük clusteringnek, Az OpenCV keretrendszerben erre példa a k-means algoritmus.

K-means

Ez egy clustering machine learning algoritmus, amely az N méretű adathalmazt k darab (k előre adott szám) csoportba osztja az alapján, hogy melyik cluster középpontjához van az éppen vizsgált adat a legközelebb.

Az eljárás kulcsa abban van, hogy mindig finomítja ezeket az úgy nevezett centroidokat, hogy a ténylegesen egy csoportot alkotó adatok "közepén" helyezkedjen el. Ezt úgy éri el, hogy véletlenszerűen kiválaszt k darabot az adathalmazból, ezek lesznek a centroidok, illetve minden további adatot hozzárendel a legközelebbi középponthoz. Majd kiszámolja ezekre a clusterekre az új centroidot az átlagos távolságok alapján és természetesen újraosztályozza az adatokat. Ezt egészen addig ismétli a rendszer, míg egy stabil állapotba ér és nem változnak már a centroidok.

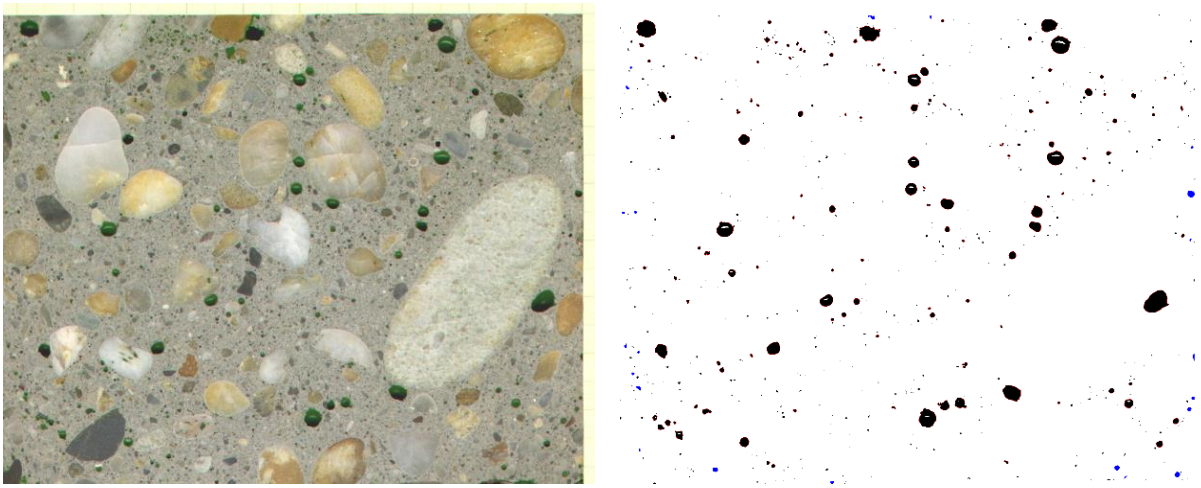


(4 clusterre bontás)

Erre készítettünk egy demo programot, amely 2D adatpontokat generál, majd az OpenCv-be beépített kmeans függvénnyel megkeresi a clustereket. Ez az eljárás több dimenzióra is alkalmazható. Például a szemcsék n darab tulajdonságai, paraméterei alapján egy n-dimennziós térben megtalálhatóak lesznek a hasonlóságok és clusterek. Sajnos ehhez túl nagy a kinyerhető feature-ök értékkészlete, de kevés adat áll rendelkezésre, ezért a továbbiakban az SVM alapú megoldást folytatjuk.

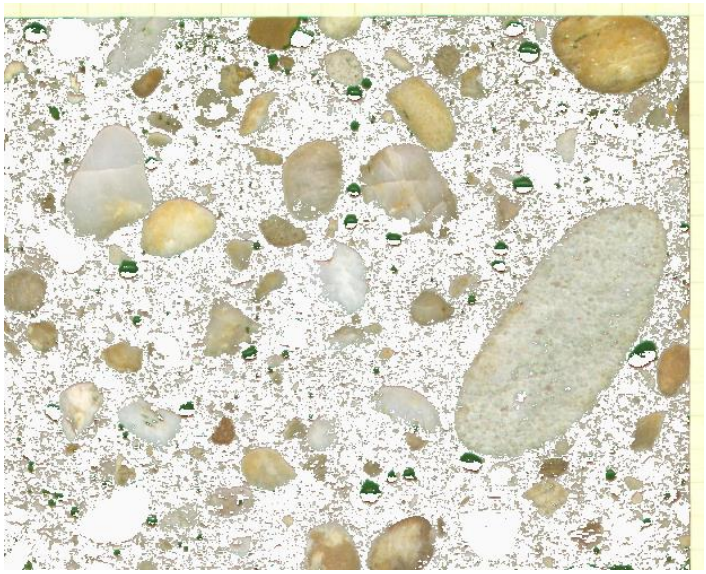
Per Pixel Classification

A feladathoz rendelkezésünkre álltak betoncsiszolat képek, valamint hozzá tartozó kontrollképek, amelyeken be voltak jelölve a szemcsék (Grain)



(Bal: eredeti, Jobb: szemcsék)

Első lépésként pixelenként próbáltuk osztályozni a képet. A pixeleket egy háromdimenziós térre vetítettük, amit egy síkkal próbál meg a SVM két részre szelni. A három dimenzió egy-egy pixel RGB értékei voltak. Az alábbi képen ennek a megoldásnak a kimenete látható.



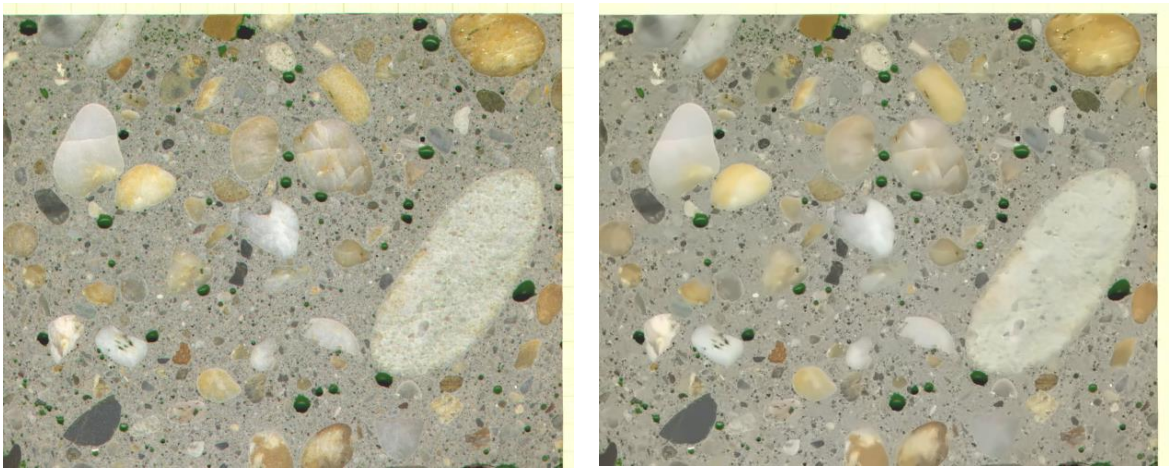
(Lineáris kernel, [r,g,b] vektor)

Az eredmény nem meglepő módon korántsem optimális. Mivel a kép nem túl absztrakt valamint elég zajos, ezért egy-egy pixelről eldönteni, hogy az egy szemcse részét alkotja-e nagyon

komplikált. Ugyan ezt a módszert alkalmazva a képpontok árnyalat-telítettség értékeire (hsv) hasonló eredményt produkál.

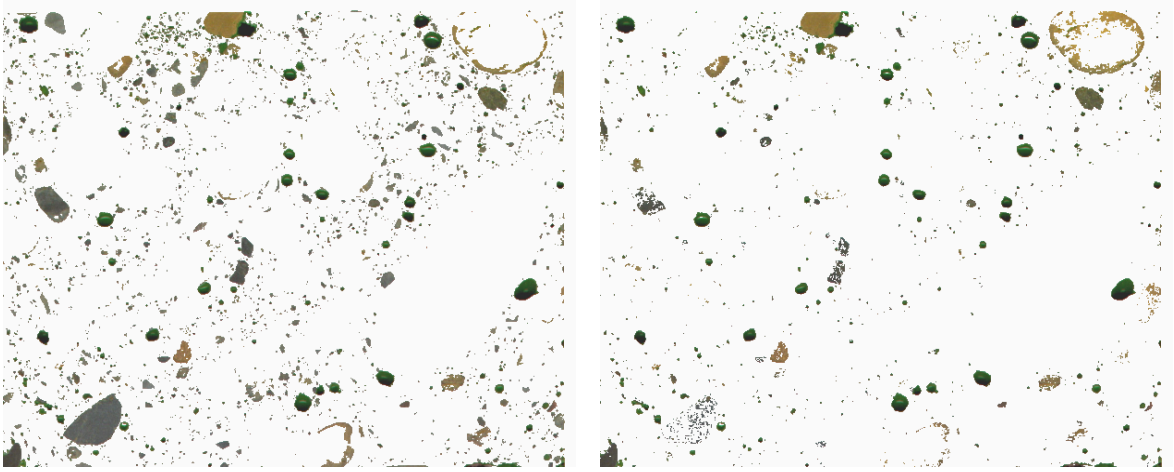
A kép absztrakciójával a block-subdivide megoldásunk foglalkozik bőveben, itt a továbbiakban is pixelenként osztályozunk de különböző képmanipulálási módszereket alkalmazunk a jobb eredmény elérése érdekében.

Alkalmazzunk zajszűrést az inputadatokon (non-local means denoising). Az alábbi képeken emberi szemmel szinte alig látható különbség a gépi képfeldolgozásban drasztikális javulást eredményez.



(Bal: eredeti, Jobb: zajszűrő alkalmazása után)

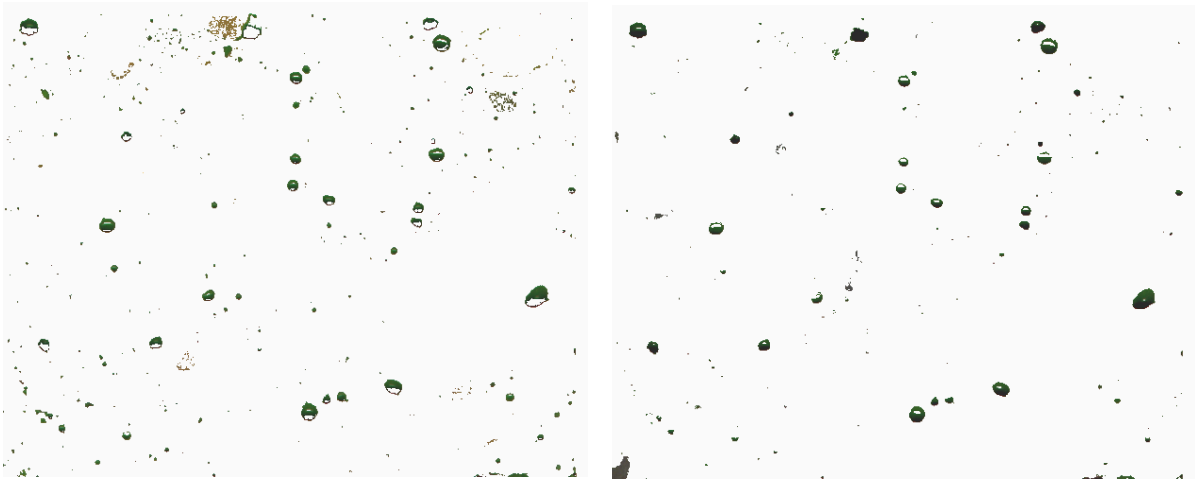
Az új kép pixelenkénti klasszifikációja RGB és HSV színtérben már sokkal jobban hasonlít az elvárt kimenethez, de sejthető, (a probléma bonyolultságából adódóan), hogy a vektorok nem lineárisan szeparálhatóan helyezkednek el a térben.



(RGB denoise, HSV denoise)

A pontosságot növelni tudjuk kernelfüggvényekkel amik a nemlinearitást beleviszik a tanulásba. Ilyenek alkalmazásával az adott tanulóadatra pontosabb kimenetet kapunk, de ezzel sérthetjük a rendszerünk predikáló képességét más bemeneti adatokra.

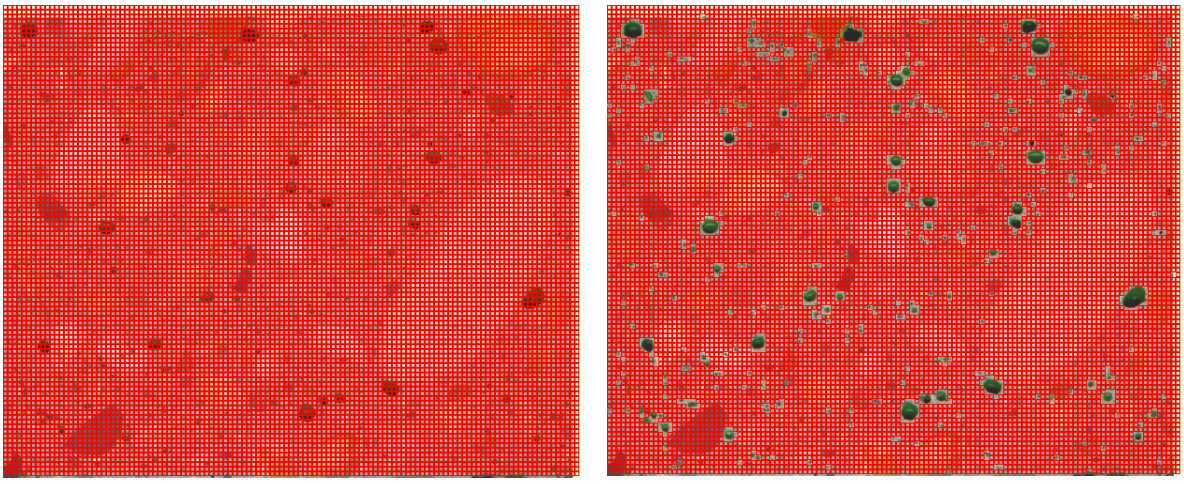
Az alábbi képeken egy enyhén overfittelő kernel (histogram intersection kernel) kimenetei láthatóak RGB és HSV színtérben.



(Inter HSV, Inter RGB)

Block-Subdivide Method

A kép pixelenkénti osztályozását eleinte igen komplexnek találtuk, ezért egyszerűsíteni akartuk a problémát több pixel összevonásával. Mivel ebben a pillanatban még nem elérhető az OpenCV superpixel funkciója C#-ból, ezért blokkokra osztottuk a képet. (A superpixel megoldást részletesebben a kitekintés a jövőre fejezetben tárgyaljuk)



(Bal: blokkosított kép, Jobb: elvárt eredmény a prediction után)

A fenti képeken látható módon a képet 5x5 pixeles blokkokra osztottuk fel, majd mindegyik blokkot egy vektorral jellemeztünk.

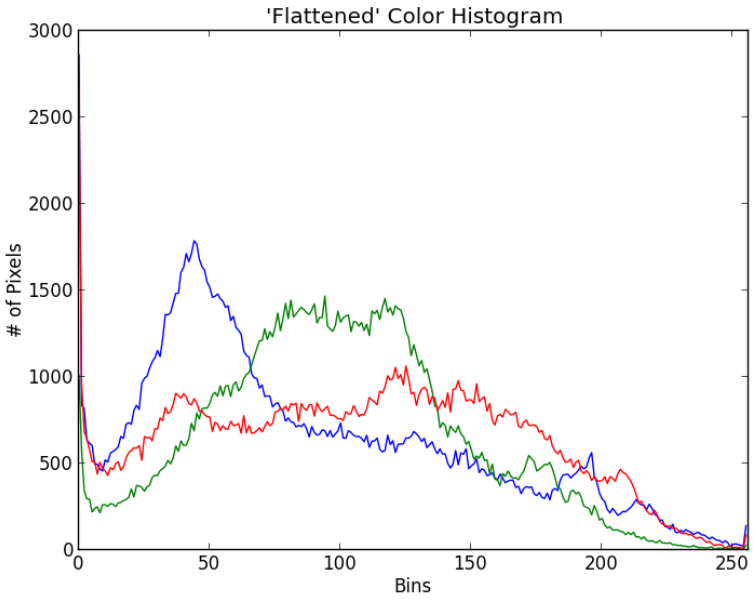
Feature Engineering

Amíg a per pixel alapú osztályzásban a felhasználható jellemző a három (RGB) csatorna volt, azzal, hogy nagyobb egységeket dolgozunk fel új tulajdonságokat is bevezethetünk, ezzel növelve az osztályozás pontosságát. A szemcsék, szennyeződések és pólusok pontos meghatározásával felhasználható a méret és a kerület / terület arány is, mint feature. Ellenben a négyzetrácsos felosztással ezek egyformák lesznek, ezért más feature-ök kellenek. Erre nyújtanak megoldást a hisztogrammok

Hisztogrammok

Színhisztogram: a képen (képrészleten) statisztikát végzünk az egyes színcsatornák előfordulásain és megszámoljuk, hogy az egyes értékekből hány darab van. Ez az érték RGB esetén [0-255] között van, amelyeket kisebb csoportokba sorolhatunk, hogy ne legyen feleslegesen nagy a featurevector. A mi implementációnkban ezt a tartományt 8 részre bontottuk fel, ez a szám már kellő mennyiségű adatot tartalmaz, illetve a jellemzők száma is kezelehető marad. (Minden csatornára 8 csoport: 3 * 8 = 24 elemű a vektor).

HSV színtér esetén a Hue és Saturation értékekre készítünk hisztogrammot, előbbi [0-180], utóbbi [0-255] közötti értékeket vehet fel.



http://www.pyimagesearch.com/wp-content/uploads/2014/01/flattened_color.png

A színhisztogram az objektum elhelyezkedésétől, irányától és betekintési szögtől nagyjából független, azaz hasonló eredményt ad az adott tárgyra különböző pozíciókban, ezért alkalmas

a felismerésre. Az OpenCV keretrendszer erre is nyújt megoldást, a clacHist függvényt használhatjuk fel erre a célra.

Irány-gradiens hisztogram: a gradiens (x és y szerinti deriváltak) hasznos az alakzatok detektálásnál, mert a sarkokon és az élek mentén az értéke nagy, így több dolgot is elárul a formáról. A gradiens nagysága mellett fontos még az iránya is. Ezzel a két tulajdonságával könnyen készíthetünk belőle hisztogrammot, erre az OpenCv a HOGDescriptor osztályt és a hozzátartozó függvényeket kínálja

(24 dim RGB színhisztogramm eredmény)

A fenti kép egy lineáris kernel használatával és RGB színhisztogrammal készült eredmény, ami minőségben nagyjából a zajszűrésen átesett pixelenkénti klasszifikációval egyszintű. Ez a megoldás nem annyira SVM, inkább deep neural network tanítására lenne alkalmas, amihez viszont nem rendelkezünk elég tanítóadattal.

Cv4SensorHub integráció

Tag és AppearanceCommand

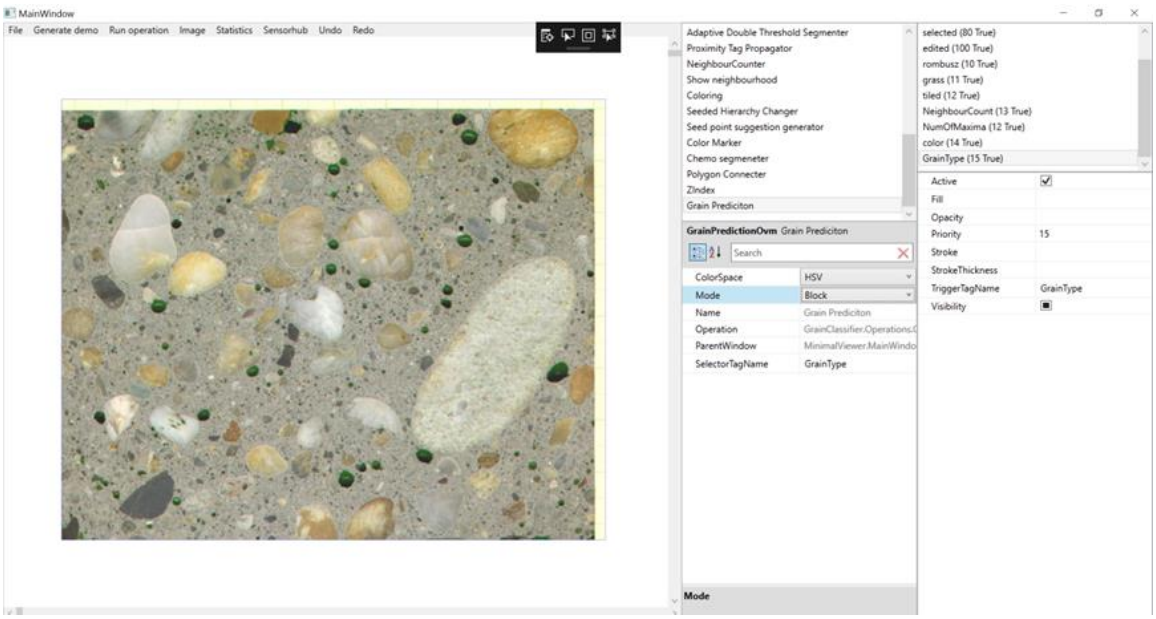
A keretrendszerben az egyes osztályokat a GrainType nevű tagben kezeljük, ennek az értékei adják meg, hogy melyik osztályba tartozik az adott szemcse, szennyeződés vagz pólus. Akár a Tagger operationnel manuálisan, akár majd az osztályozó által megjelölt poligonokat egy appearance command automatikusan kiszínezi azonos színűre az azonos osztályba tartozókat

AuxFloatArray

A Hisztogrammok különböző kimenetet generálnak, de az egységesség és a könnyebb kezelés érdekében mindegyiket azonos formára konvertáltuk, float tömbökké. Természetesen ezek tárolása is lehetséges, erre van szükség egy aux-ra.

RGBColorHistrogram, HSVColorHistogram

Ezen osztályok és függvényeik segítségével lehet generálni a megfelelő hisztogrammokat, amelyeket a poligonokhoz aux-ban lehet tárolni



Jövőbeli tervek

Superixel megközelítés:

További megoldási lehetőséget nyújtana az úgynevezett “superpixelek” alkalmazása négyzetrács helyett. Az algoritmus lényege, hogy a hasonló színű szomszédos pixeleket összevonja egy régióba, de emellett ügyel arra is, hogy a kijelölt felületek nagyjából hasonló méretűek legynek. Ez nagyban segítené a felismerendő kép absztrahálását, ami a predikciós algoritmusok hatékonyságát nagyban növelné.

http://doi.ieeecomputersociety.org/cms/Computer.org/dl/trans/tp/2012/11/figures/ttp20121122741.gif

(superpixelekre bontott képek)