BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM

VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR

TÁVKÖZLÉSI ÉS MÉDIAINFORMATIKAI TANSZÉK

Kép és videó rendszerezési problémák a gyakorlatban

TURBUCZ SÁNDOR

LEPOSA TAMÁS

KONZULENS

Dr. Szűcs Gábor – BME-TMIT

BUDAPEST, 2011

Kivonat

Az információs társadalom jelenleg a túlkínálat korát éli. Megszámlálhatatlan mennyiségű tartalom kerül fel az internetre nap, mint nap, aminek nagy részét már nem is szolgáltatók, hanem maguk a felhasználók töltik fel a web2.0 elterjedése óta. Ezen tartalmak a legutóbbi trendek szerint nem csupán szöveges információk, hanem hanganyag, képek és videók is. Továbbá a modern digitális mobil eszközök elterjedésének köszönhetően nem csak a weben, hanem a felhasználóknál is rengeteg multimédiás tartalom található: számítógépeken, fényképezőgépeken, okostelefonokon, táblagépeken. A nagy képhalmazok és videó gyűjtemények gyorsan átláthatóvá tétele egyre fontosabb feladat lett a mai világban. Jelen dolgozat erre a kettő témakörre koncentrál, mind elméleti, mind gyakorlati eredményeket felmutatva, és alkalmazási lehetőségeket említve.

A célunk tehát a felhasználók segítése, a képeik és videóik gyorsan áttekinthetővé tétele, különböző megoldások segítségével. Kitétel, hogy a folyamatnak automatizáltnak kell lennie, ne igényeljen felhasználói beavatkozást a kényelmes használat érdekében. Az elkészítendő megoldás egy alapvető céllal, mégis két területtel rendelkezik. A rendszerezés témán belül a TDK dolgozat a felmerülő problémáknak csak egy-egy szeletét mutatja be. Képek esetében azokat albumokba/témakörökbe kell rendezni, majd az albumon belül néhány, jellemző képet kiválasztani. Egy másik gyakran felmerülő probléma, hogy a videókat jelenetekre kell szeparálni. Így a sok ezer kép pl. a nyaralásról áttekinthető lesz néhány jellemző „témakör kép” alapján, a hosszú videókban pedig rögtön a megjelölt jelenetekhez ugorhatunk, keresgélés nélkül.

Fontos megemlíteni, hogy mind a képek rendszerezése, mind a videók szegmentálása a nyilvánvaló technikai nehézségeken túl rendkívül szubjektív feladat. Például, két ember két különböző módon válogatna szét egy képhalmazt, vagy osztana fel egy videót részekre, így az algoritmusnak meg kell találni azt a középutat, ami minden ember által elfogadható és megfelelő. Fenti okok miatt sok elméleti meggondolásra, tesztre és finomhangolásra volt szükség, amelyek a TDK dolgozatban kifejtésre kerülnek. Intelligens módszereket kellett tehát alkotni, melyekkel a felhasználók szemével nézve is megfelelő eredményeket kaphatunk. A vizsgálatok és fejlesztések több módon zajlottak: teszthalmazok felállításával, szubjektív véleményezéssel és a felállított mérőszámok és mutatók alapján visszacsatolással. A munka során az elméleti tervezés és tesztelés mellett egy működő,

Python nyelven írt prototípus program is elkészült, az általa elért eredmények ismertetésre kerülnek a dolgozatban.

A megoldás rengeteg érdekes alkalmazási lehetőséget rejt magában, mind online, mind offline környezetben: (i) Windows képek mappa ikonjánál a tartalmazott képekből megjelenhetne néhány jellemző „témakör kép” (még a 7-es verzióban is csak a betűrendben első néhány kép jelenik meg). (ii) Biztonsági kamerák felvételeinek elemzése, ahol a hosszú videó folyamban megállapítható, hogy mikor történtek események. További gyakorlati lehetőségek és részletes bemutatásuk is megtalálhatóak a dolgozatban az egyes területek sajátosságainak figyelembe vételével.

Abstract

Nowadays, there is an oversupply in information society. Countless amounts of content is uploaded to the internet every day, and the main part of it is created by the users of the web, not the service providers since the spread of Web 2.0. This content consists of sound material, pictures and videos not only of textual information. Moreover, due to the fast evolution of latest mobile devices, a great amount of this data is stored directly by the users, on their computers, digital cameras, smartphones and tablets. The perspicuity of large image sets and video collection became increasingly important in today's world. In this paper we are describing thoretical and practical results related to this topic, and show examples of how it can be used to solve real issues.

The goal is to help users by giving an overview of their image sets and videos. Criterion is that the process should be automated for convenience, instead of requiring user intervention. One case is to select a few typical images from albums. Another common problem is that the videos must be segmented into scenes. Due to these solutions we are able to overview thousands of pictures (for example from our holiday) with some „topic thumbnails" images, and we can immediately jump to marked scenes in long videos without search.

It is important to mention that both of the representative image selection and video segmentation is a very subjective task. For example different users pick different thumbnail image, and segmentate videos in different ways, so the algorithm must return an acceptable result for more than one user. For these reasons, many theoretical considerations, tests and fine-tuning were required, which are explained in this paper. Therefore we had to design intelligent techniques, which are able to produce results corresponding to the users' expectations. The developments have taken place in several ways: by setting up test sets with a subjective opinion and with the established benchmarks and indicators based on feedback. In addition the theoretical design and testing, a working prototype program was written in Python, and the obtained results are explained in this paper.

The solution has plenty of potential applications, both in online and offline environments. For example in the file manager, the thumbnail of the folder which contains photos should be selected this way, using some representative images (In windows 7, the first few picture appears in alphabetical order). Another application is the security camera recordings analysis. With automatized segmentation, we can find out when the events occured in long video streams. More applications and detailed descriptions are presented in the paper.

Tartalomjegyzék

1. Bevezető

Az információs társadalom jelenleg a bőség, a túlkínálat korát éli. Az internetet elárasztják a legkülönbözőbb forrásból származó multimédiás tartalmak, amik közül a legjellemzőbbek természetesen a képek és a videók. Ezek a legkülönbözőbb helyeken megtalálhatóak, úgy, mint blog oldalakon, közösségi portálokon, videó megosztó oldalakon, és szinte bármi egyéb helyen. Az interneten kívül, a felhasználók számítógépein, mobil eszközein is megszámlálhatatlan mennyiségű média tartalom van, a digitális kamerák és fotógépek elterjedésének köszönhetően. Ezen rengeteg kép között ma már nem is olyan egyszerű eligazodni, ráadásul rengeteg időt vesz igénybe megtalálni a minket érdeklő tartalmakat, és kiszűrni a felesleges elemeket. Így mind a webes, mind az offline alkalmazások körében megmutatkozott az igény a videók, és a képek osztályozására, vagy valamilyen módszerrel történő gyors áttekintővé tételükre.

Sok képet tartalmazó fotóalbumok, nyaralás során készített “napi 300 képes” albumok között nagyban megkönnyíti a navigálást, ha nem kell minden album minden képét végignézni. Minden album helyett annak néhány jellemző képét látjuk, így a felhasználóknak könnyen beugrik az adott album tartalma. Néhány terület, ahol ez a módszer hasznos lehet:

- Windows, képeket tartalmazó mappák
- Fotó- és fotóalbum kezelő programok (Adobe termékek, IrfanView)
- Webes képmegosztó oldalak (Flickr, Photobucket)

Egy videó körülbelüli tartalmát is könnyen és gyorsan megtudhatjuk, ha néhány jellemző képet látunk belőle: elég csak rápillantani a videó ikonjára. Egy videó tartalmának kutatása visszavezethető a képek közötti keresésre, a következő egyszerű módon. Egy videó sok állókép sorozata, így csak annyi a feladatunk, hogy az adott videót valamilyen programmal képek sorozatára bontsunk. Ezután már újra egy rendszerezési feladathoz jutunk. Videók esetén is megemlíthető néhány alkalmazási terület:

- Webes videó megosztó oldalak (Youtube, Google video, Vimeo)
- Médialejátszó alkalmazások (VLC, Media Player)

Ezeken, a viszonylag kézenfekvő példákon kívül számos egyéb, érdekes és ötletes felhasználási terület létezhet, ezeket a dokumentum végén fejtjük ki. A manapság használt

megoldások többnyire a felhasználói visszacsatoláson alapulnak. Üzleti rendszerek, például a Flickr egy olyan eljárást használnak a gyűjtemények mintavételezésére, ami a közösségi tevékenységet vizsgálja, és ez alapján határozza meg az érdeklődés mértékét egy adott kép iránt. Így az albumok készítésekor a felhasználóktól nyert információt felhasználva tud következtetni az album tartalmára, struktúrájára és szemantikájára [1]. Egy másik megoldás [2] tartalomelemzésen alapul, ahol a vizuális figyelem és az interaktív visszacsatolás mechanizmusát kombinálja a kép érdekességének kiszámításához.

Reprezentatív fénykép kiválasztására és releváns előnézeti kép generálására egy újabb módszer [3] a közel hasonló képeket illetve duplikációkat keresi meg. Ezen képek kapcsolatait egy gráffal modellezik. A legjellemzőbbet ezután a kölcsönös kapcsolatok alapján választja ki. Az előnézeti kép generálásánál a kiválasztott fényképen belül, annak legérdekesebb részét, lényegét próbálja kijelölni, mégpedig a jellemző pontok kiválasztásával; ez szakít az eddig megközelítésekkel [4][5].

A kapcsolódó kutatások a reprezentáns kép kiválasztását három különböző módon oldották meg; szöveges formájú érdeklődés [1], a képeken megjelenő arcok [6], vagy ezek kombinációja [2]. Ezen megvalósítások a képek tartalmát, annak tulajdonságait használták fel. Csak egy-két cikk tesz említést az EXIF metaadatokról (EXchangeable Image Format - ezek fényképekhez csatolt, a kép készítésének körülményeit leíró adatok), de azoknál is csak az időbélyeget és a fényképező típusát használták fel [6]. Ezen tanulmányok nem használták fel az összes EXIF metaadatot, pedig azok a tartalmat ugyanúgy jellemezhetik.

A jellemző képek kiválasztásának gyakorlati megvalósítására többféle lehetőség is létezik. A legegyszerűbb, hogy az első/utolsó képet, vagy képeket választjuk ki. Ez kézenfekvő megoldás, és gyorsan működik, azonban gyakorlati haszna nem sok van, esetleg esztétikai (bár a Windows is ezt használja). Második lehetőség, hogy véletlenszerűen választunk. Ez talán még az előzőnél is rosszabb megoldás, ugyanis teljesen megbízhatatlan információt ad a képhalmazról.

Valamilyen módon tehát rá kell jönni, milyen típusú, milyen tulajdonságú képek találhatóak az albumban, és ez alapján dönteni. Legtöbbször a képek csoportosíthatóak valamilyen tulajdonságaik szerint, így különálló osztályokba rendezhetőek. Például egy nyaralás során készült albumból jó eséllyel van sok kép egy bizonyos látványosságról, van néhány családi fotó, és több természetfotó. Így máris találtunk három csoportot, amihez további, esetleg

elrontott képek is adódhatnak. Ez már egy támpont lehet a jellemző képekhez, ugyanis a kategóriákból egy-egy kép már valamilyen módon jellemzi az adott albumot.

Az elképzelés adott, azonban ez több gyakorlati problémát vet fel. Egyrészt szoftveresen jóval nehezebb megtalálni, hogy melyik képek is tartoznak egy csoportba, mint egy ember által. Nem csak a képek változatossága miatt, ugyanis egy kép értékelése, összehasonlítása egy másikkal igen szubjektív lehet. Két különböző ember valószínűleg egészen más csoportokat alkotna ugyanabból az album képeiből, mivel mást gondol összetartozónak. A probléma eldöntése tehát még emberi intelligenciával sem könnyű, és főként nem egyértelmű feladat; a hangsúly nem is a nehézségen, hanem a szubjektivitáson van. A gépi döntés is hasonló problémákba ütközik, amit különböző módszerekkel lehet finomítani.

Az elgondolás megvalósítására az információs technológia eszköztárában több eszköz is rendelkezésre áll. Ezek általában matematikai, statisztikai alapú számítások, szoftveresen megvalósítva és gyakorlatba ültetve. Rengeteg különböző módszer létezik, és a legjobb ötlet ezek kombinálása lehet.

A klaszterezés az egyik legnépszerűbb általános módszer, különböző csoportok alkotására való, bármilyen objektumok között, mi ezt választottuk a megvalósítás kulcsaként. Ami újdonság még a dolgozatunkban, hogy a képek esetén a képtartalmi információkat és az összes EXIF adatot is felhasználjuk. Az algoritmus köré épített adatszerkezetekről, a kiegészítő módszerekről, a végrehajtás részleteiről és kiértékelésről lesz szó részletesen ebben a TDK dolgozatban. A dokumentum a problémákat nem csak elméleti oldalról közelíti, nagy hangsúlyt fektetünk a gyakorlatra is, így egy Python nyelvű prototípus program fejlesztése is megtörtént. Nem célunk kódrészletek és a programozás pontos bemutatása, így leginkább a tervezéssel kapcsolatos felvetések, kérdések és döntések lesznek láthatóak, kiegészítve a kulcsfontosságú pontok bemutatásával. Fontos még kiemelni, hogy a projekt fejlesztés alatt van, a módszereink sok részlete még kidolgozandó. Azonban már egy működő programmal rendelkezünk, kvázi egy keretrendszerrel, amely implementációs részleteinek kidolgozása folyamatban van.

2. Elméleti bevezető

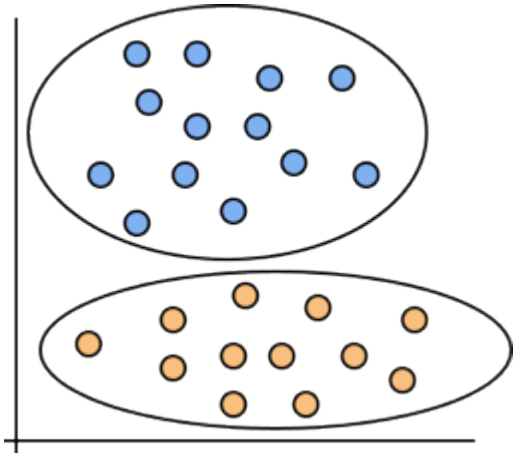
Ebben a fejezetben a munkánk megértéséhez szükséges módszerekről és technikákról adunk rövid bemutatást, kezdve adatok csoportosításának, rendezésének bevezetésével. Majd részletesen bemutatjuk az egyik legelterjedtebb adatbányászati eszközt, a klaszterezést. Ezután a képeken történő objektum felismerés következik, majd végül a videók szegmentálásával, jelenetek detekciójával zárjuk az elméleti ismertetést.

2.1. Klaszterezés

A képek és videók rendszerezéséhez azokat valamilyen módon csoportokba kell szerveznünk; erre több megoldás is létezik. Ezek közül a klaszterezés az egyik leggyakoribb eszköz. Szinte minden nagyméretű adathalmaz modellezésére alkalmas, az adatok típusától függetlenül. Matematikai és halmazelméleti alapokon nyugvó módszer, több fajtája létezik.

2.1.1. Általános bemutatás

A klaszterezés egy olyan folyamat, amely során az adathalmaz elemeit csoportokba rendezzük úgy, hogy az egy csoportban lévő elemek egymáshoz viszonyított hasonlósága nagy, a különböző csoportokban levőké pedig kicsi. A létrehozott csoportokat klasztereknek nevezzük. Ezeket az algoritmus hozza létre, általában csak a csoportok számát adjuk meg előre. Egy klaszterezésre az 1. ábrán láthatunk példát, ahol a pontok viszonylag jól elkülönült klaszterekben vannak. A valóságban sajnos nem különíthetőek el ennyire jól az objektumok.



1. ábra – Példa klaszterezésre

A klaszterezés egyik központi eleme a csoportosítandó objektumok között definiált hasonlóság. Legyen az adathalmaz $X = \{x_1, x_2, \ldots x_N\}$, ami N darab objektumot tartalmaz. Mindegyik objektum n db attribútummal jellemezhető. Két tetszőleges klaszterezendő elem hasonlóságát $s(x_i, x_j)$ jelöli. Feltesszük továbbá, hogy $0 \le s(x_i, x_j) \le 1$ valamint $s(x_i, x_j) = s(x_j, x_i)$ teljesül minden x_i, x_j elempárra és $s(x_i, x_i) = 1$ minden x_i elemre. Általában nem az elemek hasonlóságával számolunk, hanem annak inverzével, a távolsággal. A $d(x_i, x_j)$ függvény távolságfüggvény, ha teljesülnek rá a következő feltételek:

- minden x_i, x_j esetén $0 \le d(x_i, x_j) \le \infty$; (nem negatív)
- minden x_i objektumra $d(x_i, x_i) = 0$; (saját magától 0 távolságra van)
- minden x_i, x_j esetén $d(x_i, x_j) = d(x_j, x_i)$ (szimmetrikus)

A gyakorlatban több távolság-számítási módszer is elterjedt, ilyen például az Euklidészi, Manhattan vagy Minkowszki távolság. Az objektumok attribútumait, amik alapján történik az összehasonlítás, két nagy csoportba rendezhetjük. Ez egyik a kategória típusú (pl. szín), ahol az elemek között csupán az egyenlőség megléte vizsgálható, egyezés esetén a hasonlóság értéke 1, különbözőség esetén pedig 0. A tulajdonságok másik nagy csoportja a kvantitatív típusú, ahol az értékek között egyértelműen definiálhatunk egy sorrendezést, így konkrét távolságot számíthatunk. Ilyen például az életkor.

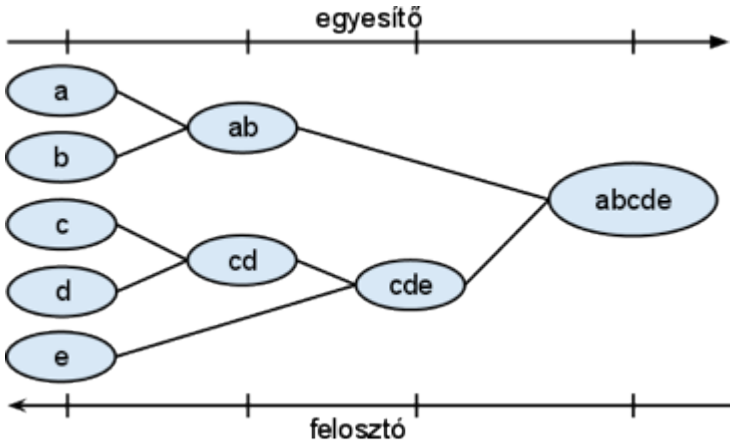
2.1.2. Módszerek

A klaszterezőket működés alapján három csoportra oszthatjuk. Ezek mindegyike eltérő hozzáállást mutat, és különböző tulajdonságokkal rendelkezik. A köztük való választáskor több szempontot is mérlegelni kell, úgy mint megvalósítás költsége, futási idő, használat egyszerűsége, vagy éppen az alkalmazás által megkövetelt tulajdonságok. A legelterjedtebb módszerek a következők:

2.1.2.1. Hierarchikus módszerek

Az első a hierarchikus módszerek csoportja, melyek az adathalmaz elemeit hierarchikus adatszerkezetbe rendezik. Ezeket a módszereket még két további csoportra oszthatjuk, aszerint, hogy hogyan történik a felosztás. Az egyesítő megközelítés (lentről felfelé építkező) úgy működik, hogy kezdetben minden objektum egy önálló klaszterbe tartozik, ezután az egymáshoz közeli klasztereket egyesítjük, a hierarchiában egy szinttel feljebb egy új klasztert

kialakítva ezzel. Ezután folyamatosan egyesítjük az egymáshoz közeli klasztereket, felfelé lépkedve a hierarchiaszinteken egészen addig, amíg minden objektum egy klaszterbe nem tartozik, vagy amíg a leállási feltétel nem teljesül. Ezzel szemben a felosztó megközelítés (fentről lefelé építkező) pontosan fordítva működik; egyetlen, minden objektumot tartalmazó objektumból indul ki, amit minden hierarchiaszinten külön klaszterekre partícionál mindaddig, amíg minden objektum egy külön klasztert nem alkot, vagy teljesül a leállási feltétel. Leállási feltétel lehet például egy elért klaszterszám. A 2. ábrán az egyesítő és felosztó megközelítés közötti különbséget láthatjuk.



2. ábra - Az egyesítő és felosztó hierarchikus klaszterezés

2.1.2.2. Partícionáló módszerek

A partícionáló módszerek ezzel szemben előre meghatározott számú csoportba partícionálják az elemeket (kezdeti klaszterezés), majd a klaszterezést javítják iteratív módon a csoportok alakításával. Fontos megemlíteni, hogy a partícionáló módszereknél a klaszterek száma előre adott. Ezen számot a felhasználónak kell megadni, azonban léteznek egyéb módszerek is, például az algoritmust többször lefuttatjuk különböző klaszterszámmal, majd ezeket hasonlítjuk össze és választjuk ki a számunkra legmegfelelőbbet.

A „jó” klaszterszám megadása mellett fontos még a „jó” kezdeti klaszterezés megtalálása. Ha ez nem sikerül, akkor előfordulhat, hogy az algoritmus csak egy lokális optimum felderítésére képes. Ez esetben szükséges többször, különböző kezdeti halmaz alapján elvégezni, majd az eredmények közül a számunkra legmegfelelőbbet kiválasztani. A kiválasztás történhet intuíció vagy egyszerű heurisztikák alapján, azonban léteznek algoritmikus módszerek is, mint például a k-közép++, amelyet később bemutatunk.

A partícionáló módszerek másik fő kérdése a klaszterezés minőségének definiálása, és az ennek megfelelő célfüggvény kiválasztása. A különböző algoritmusok különböző célfüggvényeket használnak. A célfüggvény javítására egy adott lépésben több lehetőség is kínálkozik, az algoritmusok általában „mohón” a legjobbat választják ezek közül. A leggyakrabban alkalmazott minőségi kritérium az ún. négyzetes hiba vizsgálatán alapul (1). Egy objektum (x_i) négyzetes hibája a klaszter (C_j) centroidjától (c_j) mért távolságának négyzete.

$$MSE = \left\| x_i - c_j \right\|^2 \text{, ahol } x_i \in C_j \tag{1}$$

A klaszter centroidja, vagy más néven középpontja a klaszter síkjának súlypontja, egyszerűbben fogalmazva pontjainak átlaga.

$$\vec{c}_j = \frac{1}{\left\| C_j \right\|} \sum_{x_i \in C_j} \vec{x}_i \tag{2}$$

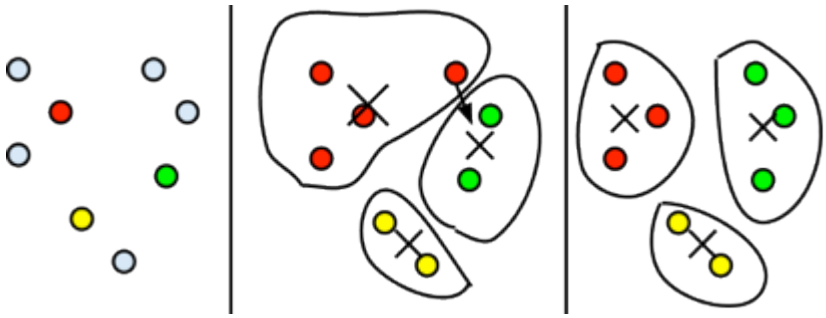
A négyzetes hibán alapuló algoritmusok célja a klaszterezés teljes négyzetes hibájának ($E(C)$) minimalizálása.

$$E(C) = \sum_{j=1}^{k} \sum_{x_i \in C_j} MSE_{ij} \tag{3}$$

Látható, hogy a klaszterek centroidjainak kiszámítása megkötés arra nézve, hogy objektumainkat kizárólag vektortérben adhatjuk meg.

Az előzőekben említett partícionáló módszerek közé tartozik a k-átlag (k-means) algoritmus. Ez az egyik legelterjedtebb módszer, részben a könnyű megvalósítás miatt. Első lépésként a felhasználó által megadott k számú tetszőleges pontot választunk az objektumaink által alkotott vektortérben, ezek lesznek az első iterációban a klaszterek reprezentánsai. Tehát az első lépés után k darab, egy elemű klaszterünk van. A második iterációban minden objektumot a hozzá legközelebb lévő reprezentánssal rendelkező klaszterhez soroljuk. Mindig, amikor egy klasztert módosítunk (azaz új elemet adunk hozzá), újra kell számolni annak centroidját. Ez az algoritmus a klaszterek súlypontját használja erre, azaz a pontjainak az átlagát, hasonlóan a fizikában használt súlypont számításhoz. A második iteráció végén már az összes objektum be lesz sorolva valamelyik klaszterba. Az összes többi lépés már csak az eredmény finomítására való, ugyanis a centroid értékek változásával lehetséges,

hogy a korábban megvizsgált pontok már egy másik klaszter centroidjához esnek közelebb. Akkor állunk meg,amikor a a klaszterek pontjai, illetve a középpontjai már nem változnak. Az algoritmus egyik legnagyobb előnye a skálázhatóság, mivel a futási ideje lineárisan függ az elemek számától (pontosabban $O(nkt)$, ahol n az elemek száma, k a klaszterek száma és t az iterációk száma). Egyik nagy hátránya, hogy a végeredmény függ a kezdeti ponthalmaz kiválasztásától. Előfordulhat, hogy végeredményként csak lokális optimumot kapunk, azonban ahogy már említettük, ez javítható az algoritmus többszöri lefuttatásával.



3. ábra - A k-átlag algoritmus működése

A k-átlag algoritmus működésére a 3. ábrán látható egy példa. Az első lépésben a kezdeti klaszterezésként kiválasztott három pont lesz a kiindulási három halmaz (piros, sárga és zöld). Látható, hogy a második iterációban még az egyik piros pontot áthelyezi a zöldbe, mivel közelebb van annak a középpontjához. Ez a lépés után az újraszámolt középpontok alapján nincs változtatás, az algoritmus kész.

A k-átlag algoritmus kiindulási feltétele egy kezdeti klaszterezés. Elterjedt módszer, hogy a klaszterszámnak megfelelő mennyiségű pontot véletlenszerűen kiválasztunk. Így a kezdeti klaszterezés k db egyelemű klaszterből áll. Ha véletlen kiválasztást alkalmaznánk, rossz esetben előfordulhat, hogy közeli pontokat választunk, így megnő a szükséges iterációk száma, az algoritmus futása lényegesen több ideig is eltarthat. A véletlenszerű kiválasztás helyett jobb eredményre vezethet a k-átlag++ algoritmus [7]. Ez nem egy másik klaszterezési mód, hanem csupán a kezdeti középpontok kiválasztására ad egy jobb megoldást. A módszer működése a következő lépésekből áll:

1. Első lépéseként véletlenszerűen, egyenletes eloszlással kiválasztunk egy pontot, ez lesz az első centroid.

2. Minden nem centroid pontra kiszámítjuk a távolságát a legközelebbi centroidtól (ez a távolságot $D(x)$-szel jelöljük), majd egy valószínűséget rendelünk hozzá a (4) képlet szerint (a valószínűség a ponttól egyre távolabb lévő többi pontnál egyre növekszik):

$$\frac{D(x')^2}{\sum_{x \in X} D(x)^2} \tag{4}$$

3. Ezután a nem centroid pontok közül kiválasztunk egyet a hozzájuk rendelt valószínűség alapján; a távolabbi pontok nagyobb valószínűséggel lesznek kiválasztva.

4. Addig ismételjük a 2. és 3. lépéseket, amíg a megfelelő számú pontot ki nem választottuk.

Ezzel a módszerrel egymástól megfelelő távolságra lévő, a ponthalmazban jól elszórt pontokat tudunk kiválasztani a térben, így ezek kiválóan alkalmasak a k-közép algoritmus kezdeti klaszterezésére. A k-közép++ szemléletes bemutatására a későbbiekben még visszatérünk.

2.1.2.3. Sűrűség alapú módszerek

Egy következő csoportba alkotnak azok a módszerek, amik klasztereket az adott térrészen belül lévő pontok sűrűsége alapján határozza meg. A legelterjedtebb ilyen módszer a DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [8] algoritmus, ami iteratívan vizsgálja, hogy egy adott pont valamilyen sugarú környezetében van-e elegendő pont (tehát megfelelő-e a sűrűség) a klaszter további kiterjesztéséhez. Általában kevesebb dimenziójú esetekre használják (többnyire 2 illetve 3).

Az algoritmus a bemeneti ponthalmazon kívül igényel még egy ε és egy „minimum pontok száma” paramétert. Segítségükkel definiálhatunk egy szomszédossági relációt a _ képlet szerint.

$$N_\varepsilon = \{(p,q) \in P \times P \,|\, dist(p,q) \leq \varepsilon\} \tag{5}$$

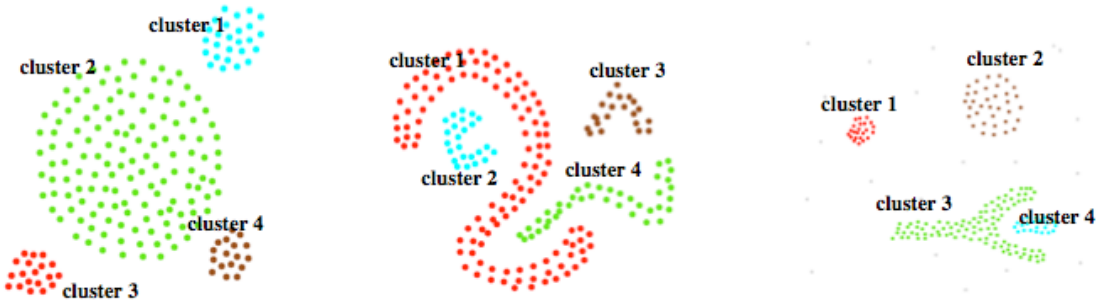
Tehát p és q akkor szomszédosak, ha ε távolságon belül vannak egymástól. Egy adott pont szomszédjainak száma a (6) képlet szerint definiált.

$$N_\varepsilon(p) = \{q \in P \,|\, (p,q) \in N_\varepsilon\} \tag{6}$$

Az algoritmus szerint egy pont magpont, ha legalább annyi szomszédja van, mint a paraméterként megadott „minimum pontok száma”. A következő definíciókra van szükség az algoritmus megértéséhez:

- A *p* pont **közvetlenül elérhető** *q* pontból, ha *p* a *q*-nak szomszédja és *q* magpont.
- A *p* pont **elérhető** *q* pontból, ha létezik egy olyan pontsorozat, ami *q*-ból indul és *p*-ben végződik úgy, hogy minden pont közvetlenül elérhető az előzőből.
- A *p* és *q* pontok **összekapcsoltak**, ha létezik egy olyan *r* pont, hogy *p* és *q* elérhető *r*-ből.
- Egy ponthalmaz ($C \subseteq P$) **összefüggő**, ha $\forall p,q \in C$ összekapcsolt egymással, tehát minden pontja páronként összekapcsolt.
- Egy ponthalmaz ($C \subseteq P$) **maximális**, ha $\forall p \in P, q \in C$ pontra, ha *p* elérhető *q*-ból, akkor $p \in C$, azaz nincs olyan külső pont, amiből elérhető bármely klaszterbeli pont.

Egy ponthalmaz akkor klaszter az algoritmus szerint, ha a fenti definíciók alapján összefüggő és maximális egyszerre. Egy példát láthatunk a 4. ábrán a módszerről.



4. ábra - A DBSCAN algoritmus által felfedezett klaszterek [8]

2.2. Objektum felismerés képeken

Manapság már nem elegendőek a régen megszokott tartalomelemzési módszerek. A szövegekre jó ideje készítenek különböző cégek kereső, indexelő szolgáltatásokat, az internetes keresők is elég jól megoldják már ezt a problémakört. Azonban a képek és videók kezeléséhez más módszerek szükségesek. Ezen tartalmakról információt kinyerni/kereshetővé tenni teljesen más technikákat és megoldásokat igényel, mint szövegek esetén. A sokkal összetettebb és nagyobb adathalmazok miatt természetesen ezek

nehezebb, és nagyobb számítási igényű feladatok, és új problémákat, azokra új megoldásokat jelentenek. Míg szövegek esetén főként szavakat, mondatokat nyerhetünk ki adatbányászati módszerek segítségével, addig képeknél a kép tartalmára vagyunk kíváncsiak. Jó lenne a kép által ábrázolt dolgokat felismerni, és valamilyen módon kereshetővé tenni. A témakörben már léteznek kutatások és megoldások, ám széleskörű elterjedése és integrálása még nem teljesen megoldott.

2.2.1. Áttekintés, előnyök

A képekből való tartalomkinyerés témakörét gyűjtőnéven CBIR (Content Based Image Retriveal - Tartalom alapú kép visszakeresés) technikák néven emlegetjük. Ez egy rendkívül szerteágazó és sok módszert magába foglaló témakör. A lényege, hogy a képről információkat gyűjtsön, és azt valóban a kép tartalmából szerezze meg. A régebbi módszerek, sőt, még manapság is, a legtöbb webes képkereső szolgáltatás csak a fellelhető képekhez tartozó szöveges adatok alapján tud következtetni a kép információtartalmára. Például a képhez csatolt leíró címkék, a képet körülvevő html információk (meta tagek, alt attribútumok, stb.) alapján. Ezek azonban megbízhatatlanok és pontatlanok lehetnek, vagy egyáltalán nem is állnak rendelkezésre, így a keresés sokszor pontatlan eredményeket ad.

A CBIR segítségével azonban a kép valós tartalmát tudjuk elemezni (színek, formák, alakok, stb.), sőt, haladó technikákkal akár konkrét objektumokat, tárgyakat is felismerhetünk. Ezen módszerekkel sokkal több információhoz juthatunk a képek tartalmáról, a kereséseket is sokkal hatékonyabbá tehetjük, és intelligens alkalmazások készíthetőek, mivel ez a gondolkodásmód sokkal közelebb áll az emberi agy általi képfeldolgozáshoz. Rendkívül sokoldalúan alkalmazható technológia: képfelismerés és -keresés, videó követés, mozgások felismerése, és további alkalmazások készültek és készülnek a különböző módszerek segítségével. Embertársaink segítésére is egy megoldás lehet, például vakok számára is léteznek lehetőségek [9].

2.2.2. Néhány módszer

A témában rengeteg kutatás és matematikai elemzés (pl. [10]) született, ezeknek megfelelően sok eltérő algoritmus létezik. Ezen anyagok nagyrészt angolul, vagy egyéb nyelveken elérhetőek, magyarul nem sok irodalma fellelhető a témának.

A David Lowe által publikált SIFT (Scale Invariant Feature Transform) [11] lényege, hogy minden képből leírókat, úgynevezett kulcspontokat nyer ki, különböző matematikai

módszerek segítségével. Az algoritmus olyan módon készíti el ezeket a leírókat, hogy a kép különböző torzításai (nagyítás-kicsinyítés, nyújtás, elforgatás) esetén is nagy eséllyel megtalálhatóak legyenek a módosított képen. Ezen kulcspontok mindegyike egy 128 dimenziós vektor, és egy kép akár több ezer ilyen vektort is tartalmazhat. A SIFT egy igen jó hatásfokú megoldás képi egyezések keresésére, ám számítási igénye nagy/bonyolult képek esetén nagy lehet, amit optimalizálásokkal lehet javítani. Megemlítendő még, hogy a SIFT algoritmus csak a leírók generálását, annak matematikai módszereit definiálja. Az azon túli, ezekre épülő módszerek, akár csak a leírók egyezésének vizsgálata, többféleképpen megoldhatóak, ezekre a szerző csak ajánlást tesz.

A SURF (Speeded Up Robust Feature) [12] egy új skála-invariáns, jellmző-kinyerő módszer képekre. Számítási igénye alacsonyabb a legtöbb módszernél, mégis nagy hatásfokkal működik. Ezt úgy éri el, hogy a képek integráltját használja fel a konvolúciós lépés során, építve az eddigi módszerekre és egyszerűsíti azokat. Alapötletét a SIFT szolgáltatta, hasonlóan sok másik algoritmushoz, azonban Haar-like leírókat használ a képek jellemzésére.

A GLOH (Gradient Location and Orientation Histogram) [13] módszer szintén egy robosztus képleíró. Hasonló a SIFT-hez, azonban inkább a területekre, régiókra koncentrál a hisztogramok előállításánál. A leírók itt 64 dimenziósak.

2.3. Videó szegmentálás

A videók egy nagy csoportot képeznek a digitális tartalmakon belül, ezért fontos foglalkoznunk velük. A feldolgozásuk (azaz annak egyik aspektusa) a dolgozat másik nagy témája, ami nem is áll olyan messze a képektől, mint ahogy az elsőre tűnik. Ugyanis minden videó képek gyors egymásutánjából áll, így felbontható sok képkockára, amikre már az általunk használt képfeldolgozási módszerek alkalmazhatóak.

2.3.1. A probléma leírása

A megoldandó feladat a címből sejthető - videók részekre bontásáról van szó. A videók szegmentálása, mint probléma, sok helyen előfordulhat, ahol egy videó folyamot szét kell választani bizonyos szempontok alapján, és cél érdekében. Ennek leggyakoribb esete, - a dolgozat is ezt a célt tűzte ki maga elé - amikor a videót jelenetekre, azaz egymástól elkülöníthető történésekre bontunk. A munkánkban automatikus jelenet detektációt

valósítunk meg, ami képes emberi beavatkozás nélkül megtalálni az egyes „shot”-ok, azaz jelenetek határait. Ennek rengeteg hasznos alkalmazása lehet.

A feladattal azonban rögtön több probléma is akad. A technikai részleteken kívül, még nagyobb gond a szubjektivitás kérdése. Ugyanis ahogy egy képhalmazból való jellemző képek kiválasztásánál, úgy a videók jelenetekre bontása esetén sincs egyetlen tökéletes megoldás. Különböző emberek különböző helyeken húznák meg a jelenetek határát, például egy átlagos videó vágó programmal, és ezek a határok nem is minden esetben egyértelműek. Példaként említve, két jelenet közötti átúsztatásos hatás esetén nehéz megállapítani, hol van az egyik jelenet vége, és hol a másik eleje, vagy esetleg hogy egy jelenetbe tartozik-e a két rész. Így az elkészítendő programnak általánosan elfogadható megoldásokat kell adnia, jó közelítésekkel az emberek által gondolt megoldásokhoz.

Ugyancsak kérdéses magának a feladatnak a definiálása: mit értünk egy jelenet alatt? Egyértelműnek tűnhet: egymástól elkülöníthető történéseket. Azonban mégis több kérdés vetődhet fel, amelyek nagy része a szubjektivitásra vezethető vissza:

- A videó folyamán többször ismétlődő (vagy nagyon hasonló) jeleneteket egynek vehetjük?
- A „lényegtelen” jelenetek (teljesen sötét/világos képkockák, áttünést megjelenítő események, stb.) is valódi jelenetnek számítanak?
- Sok, nagyon rövid (< 1sec), egymás után pörgő jelenet esetében nem biztos, hogy jó ötlet rengeteg kis jelenetként kezelni őket; milyen esetben, és melyek vonhatóak össze egy nagyobb jelenetté?

2.3.2. Jelenleg használt módszerek, programok

Az automatikus jelenet detektálás témájában már több éve születnek kutatások, publikációk (pl. [14]), különböző módszereket, algoritmusokat ajánlva. A módszereket implementáló, kész programok is készültek, erre példák a [15], [16] és [17] szoftverek, de az Adobe is elkészítette saját megoldását a Premiere Pro csomag részeként [18]. Ezek természetesen zárt forráskódú, kereskedelemben lévő és megvásárolható termékek. Azonban általánosan elmondható, hogy mind a szegmentálási módszerekre irányuló kutatások, mind a szoftverek legtöbbje ugyanazon alapötletet használja: az egymást követő képkockák (frame-ek) közötti eltérést vizsgálják.

Ez a legegyszerűbb hozzáállás, ezért könnyen megvalósítható. Azonban számításigénye nagy lehet, főként nagy felbontású videók esetén, így az elfogadható futásidő miatt nagyfokú optimalizáló módszerek szükségesek. Még nagyobb hátrány, a [19] dokumentum által tárgyalt, és általunk is említett, átúsztatásos probléma. Két képkocka összehasonlítása esetén a közöttük lévő eltéréseket vizsgáljuk, ám ezen esetekben az egymást követő képkockák között nagyon kicsi eltérés található, így nehézségekbe ütközik a két jelenet elválasztása. Ez valós probléma, és általános megoldást találni rá nehéz feladat.

2.3.3. Miben más a mi megoldásunk

Az általunk vázolt videó szegmentálási módszer az eddigiektől több pontban különbözik. Ezek közül az első és leglényegesebb a programunk fejlődéséből adódik. Ugyanis először képosztályozó, klaszterező megoldásban gondolkodtunk, és ez után jött az ötlet, hogy a megvalósított módszert alkalmazzuk videókra is. Így, a megoldásunk az eddigi gyakorlattól eltérően nem lineárisan lépked végig a videók képkockáin, azokat összehasonlítva; ehelyett a videót előbb képkockákra bontjuk, majd ezekből klasztereket képezünk, a fentebb bemutatott módszer segítségével. Ezután további javító algoritmusok következnek a jelenet szegmentálás, így a klaszterezés javítására, ami később bővebben kifejtésre kerül. Azonban már most elmondható, hogy a módszerünk, a klaszterezés jellegéből adódóan sokkal jobban kezeli az átúszásos problémát, mint a kockáról kockára lépegető eddigi megoldások, mivel az átúsztatásos jelenetek képkockáit külön klaszterként megtalálja, és egybe fogja.

3. Kép és videó rendszerezési megoldásunk

Ahogy a bevezetőben említettük a megoldási koncepciónkat, a koncepció alapján egy Python nyelvű prototípus program fejlesztése is megtörtént. A kódrészletek és a programozás pontos bemutatása helyett a tervezéssel kapcsolatos felvetések, kérdések és kulcsfontosságú döntések bemutatásával folytatjuk dolgozatunkat. A dolgozatunkban elért eredmények kép-rendszerezési részét benyújtottuk egy lektorált, rangos nemzetközi konferenciára, ahol az elbírálás után elfogadták a cikket [23].

3.1. Képek beolvasása

A megvalósítandó alkalmazás első lépésként beolvassa a megadott képeket, majd ezután analízist végez rajtuk, a szükséges minőségi jelzők kinyerése érdekében. A képeket, és a kinyert információt egy később ismertetett adatstruktúra szerint tárolja. Ezután az adatokon klaszterezést végez, majd egy egyéni eljárás szerint kiválasztja a képhalmazt reprezentáló képeket.

A fontos algoritmus paramétereket – úgy, mint az EXIF adatok neve, és az attribútumok súlya, maximális értéke - egy külön settings.py fájlban tároljuk a könnyebb paraméterezhetőség érdekében; továbbiakat a parancssori futtatáskor paraméterként kell megadni. A beolvasást és a képfeldolgozást a *Python Imaging Library* (későbbiekben: PIL) [20] függvényei segítségével vittük véghez.

3.2. Analizálás, képfeldolgozás, felhasznált képadatok

Az alapötlet az volt, hogy a képeket csoportokba rendezzük hasonlóság alapján, majd e csoportokból választunk egy jellemző képet. A csoportokba rendezésre hatásos módszernek bizonyult a klaszterezés. Fontos kérdés azonban, hogy mi alapján lesz a hasonlóság (azaz a távolság) definiálva a képek között.

Az első nagy attribútum-csoport a kép pixeleiből készített statisztikai adatokból áll. Ilyen mérőszámok például a pixelek színeinek négyzetes összege, ami a kép világosságára utal.

Felhasználtuk továbbá az átlagos színt, illetve a medián színt. Hasonló mérőszám a pixelek négyzetes átlaga (RMS - Root Mean Square), illetve a színek varianciája.

A felhasználandó attribútumok másik nagy csoportját a képekhez rendelt meta adatok (az EXIF adatok) alkotják. Ezek általában csak a digitális fényképezőgéppel készített képeknél vannak jelen, ami leszűkíti a felhasználhatóságot, azonban a kitűzött cél éppen a személyes fényképekről szól, amit manapság szinte kizárólag ilyen módszerrel rögzítenek.

A következő lépés a beolvasott képek analizálása, a releváns attribútumok kinyerése. Az attribútumok egy részét a PIL-ben található ImageStat osztály segítségével számítottuk minden képre. A megoldásunkban ezeket választottuk ki:

- sum2: pixelek négyzetes összege
- mean: átlagos szín
- median: szín medián
- rms: négyzetes átlag
- var: variancia

Ezek az attribútumok nem skalár értékek, mindegyik egy hármassal írható le. A három RGB csatorna szerint. Így pl a mean helyett *mean_R*, *mean_G* és *mean_B* attribútumokat használtunk.

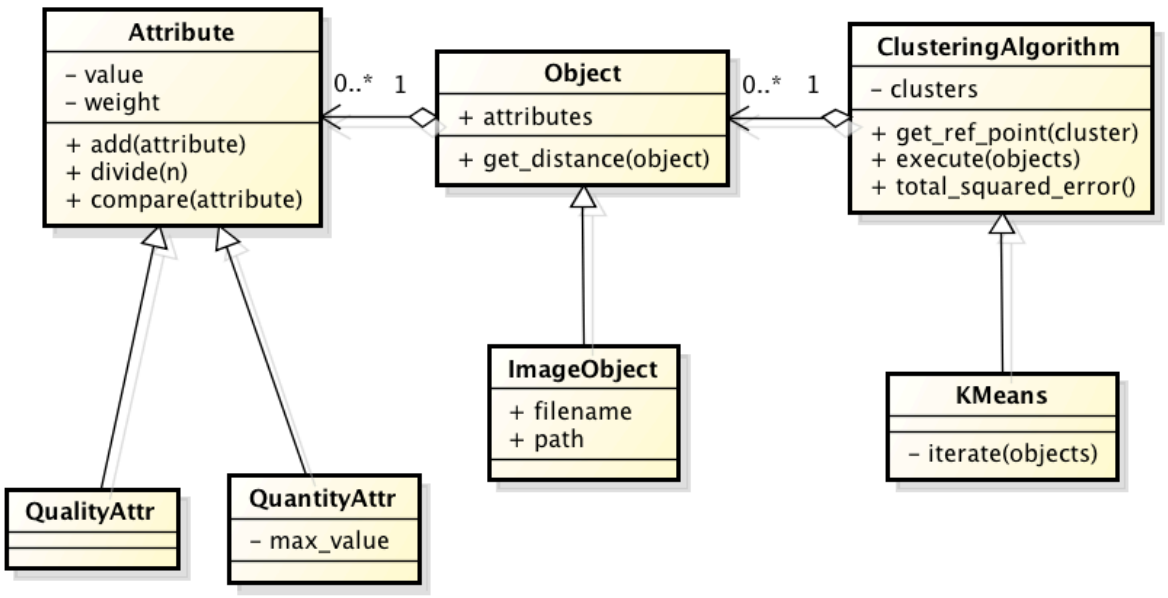
Ezeken a tulajdonságokon felül megvizsgáltuk a képekhez rendelt meta adatokat; (EXIF adatok) a következők kerültek felhasználásra (Az információk kinyeréséhez a [21] linken elérhető programot használtuk):

- EXIF Contrast: kontraszt
- EXIF ExposureProgram: program mód
- EXIF Flash: a készítés során használt vaku beállításai
- EXIF LightSource: a megállapított fényforrás
- EXIF MeteringMode: fénymérés módja
- EXIF Saturation: kép szaturációja

- EXIF SceneCaptureType: scene mód
- EXIF Sharpness: kép élességi beállítás
- EXIF WhiteBalance: fehéregyensúly
- Image Orientation: a kép orientációja
- EXIF ExposureTime: expo idő
- EXIF FNumber: F szám, azaz apertúra nagyság
- EXIF FocalLengthIn35mmFilm: a fókusztávolság a 35mm-es filmhez számítva
- EXIF ISOSpeedRatings: ISO érték

3.3. Adatszerkezetek

Az adatok tárolásához, és a számításhoz objektum-orientált szemlélettel megtervezett struktúrát készítettünk. Az egyszerűsített osztálydiagram a legfontosabb tagváltozókkal és metódusokkal az 5. ábrán látható; elemeiről ebben a fejezetben olvashatunk.



5. ábra - Az alkalmazás egyszerűsített osztálydiagramja

3.3.1. Objektumok

Az objektumot az *Object* osztály reprezentálja. Az attribútumokat az attributes tagváltozójában tárolja, ami egy tömb, (a következő részben ismertetett) Attribute példányokkal. Az osztály rendelkezik egy tagfüggvénnyel is (*get_distance*), ami egy attribútum listát vár paraméterül, majd kiszámítja ezen attribútum lista által reprezentált objektum távolságát ettől az (*self*) objektumtól. Ez a függvény alap esetben az *euklidészi távolságot* számítja ki az attribútumok, mint koordináták alapján, a módszerre egyszerűsége és gyorsasága miatt esett a választásunk.

Az *ImageObject* az *Object* leszármazottja. Egy speciális klaszterezendő objektum, ami az attribútum listán felül tárol egy elérési utat, és egy fájlnevet, és egyéb tulajdonságokat is. A képeket *ImageObject*-ként tárolja a rendszer.

3.3.2. Attribútumok

A képekhez rendelhető attribútumokat objektumokként kezeljük. Az Attribute főosztály két attribútuma az adott kép attribútum értékét, és súlyát tárolja. Ha a compare függvényt meghívjuk egy paraméterül átadott másik attribútummal, kiszámolja a két attribútum közti különbséget. Ezt természetesen két ugyanolyan típusú attribútum közt van értelme használni. Ez az összehasonlítás azonban korántsem egyértelmű, kvalitatív és kvantitatív (továbbá dátum) típusú kép attribútumok esetén jelentősen különbözik, így a konkrét összehasonlítás függvényt a leszármaztatott osztályok definiálják, amiket be is mutatunk:

3.3.2.1. Kvantitatív attribútumok

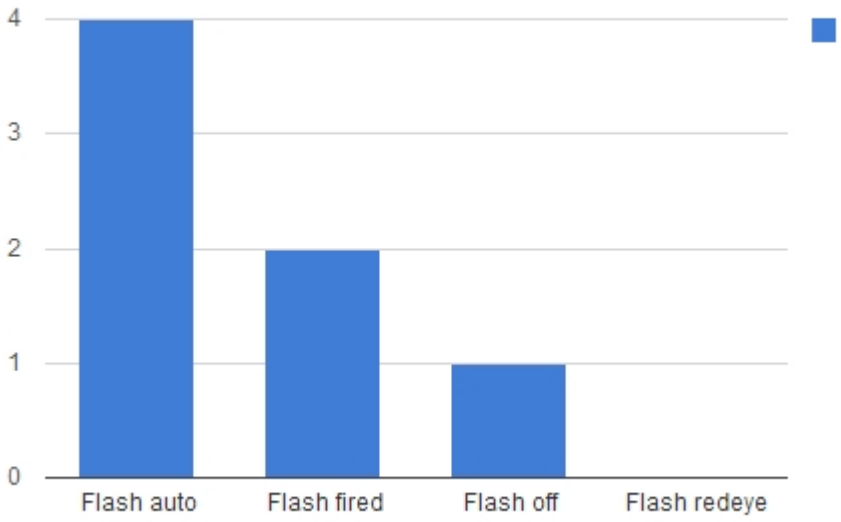
Ezek a mennyiségi adatokat tároló kép attribútumok (pl. átlagszín, F-szám, záridő, stb.). amelyek könnyen összehasonlíthatóak, így a compare függvény csupán a két érték egymásból való kivonása. A *divide* függvény egyszerű osztás, a klaszter centroidjának kiszámolásakor vesszük hasznát, kvantitatív értékek esetében; ahogy az add függvénynek is. Fontos még a *max_value*, ami az adott attribútum által felvehető maximum érték; erre azért van szükség, hogy minden értéket normálhassunk a [0..1] tartományra, így ezek később súlyozhatóak. A normálás az objektum létrejöttekor történik meg, a max_value-t a konstruktor kapja, mint paramétert. Megemlítendő, hogy a színösszeg-négyzet (*max_sum2*) maximumának megállapítása automatikusan történik képbeolvasáskor, ugyanis a kép méretétől is függ.

3.3.2.2. Kvalitatív attribútumok

Néhány kép attribútum olyan adatot tárol, amelyek esetén nem értelmezhető az értékek közötti összehasonlítás a megszokott módon, ezek nem sorrendezhető tulajdonságok. Ilyenek pl. a fehéregyensúly, vaku használata, fotózási mód; itt csak az értékek egyezését vagy nem egyezését tudjuk vizsgálni. A compare függvény is bonyolultabb ez esetben. Két pont közötti „távolság" számítás esetén, ha az adott kvalitatív attribútum megegyezik (pl. mindkét képen „*flash off*", azaz nem használtunk vakut), 0-t ad vissza, ha nem, 1-et.

A bináris eredmény helyett azonban van egy másik lehetőség is amit kidolgoztunk. A 6. ábrán egy hisztogram látható, ami egy 7 elemű klaszter elemeinek az eloszlását szemlélteti, a vaku használata, mint kvalitatív paraméter alapján. Tehát a klaszterban 4 kép van, ahol automata vaku volt használva, 2 ahol manuálisan használták, 1 ahol ki volt kapcsolva, továbbá egy képnél sem volt vörösszem-vaku. Ez alapján, egy pont távolsága ettől a klasztertől a következőképpen alakul. Ha a képen auto flash volt, akkor a távolsága a klasztertől 0, mivel ebből van a legtöbb a klaszterben is. Ha bármely másik érték, akkor a klaszterben lévő, vele azonos attribútum értékű elemek számával arányosan nő, egészen a lenormált maximális attribútum-értékig (a kvalitatív attribútumoknak nem értelmezett a maximális értéke, és normálni sem kell őket). Akkor lesz a maximum a távolság, ha a klaszterben egy ugyanolyan értékű elem sincs.

Tehát, 1 értékű lenormált maximális távolság esetén, ha a vizsgálandó pont esetén a vaku manuálisan volt használva, akkor a pont eme klasztertől a távolsága ½, ha nem volt vaku, távolsága ¾, ha pedig vörösszem-vakut használt, a távolság 1 (a maximális érték).



6. ábra - Egy klaszterben levő elemek egy konkrét attribútum szerinti eloszlása

3.4. Klaszterezés

A feladat kulcsfontosságú lépése a klaszterezés megvalósítása a kapott képhalmazon. A képek halmazát a k-átlag algoritmus szerint klaszterekre bontjuk. A klaszterezés a képekből kinyert információkat használja fel, mint attribútumokat. Így a képek tekinthetők úgy is, mint egy n dimenziós térben elhelyezett ponthalmaz, ahol n a egy kép attribútumainak a száma. Két pont közötti hasonlóságot úgy definiáltuk, hogy egy adott attribútum esetén értéke 0, ha a két attribútum megegyezik, és minél jobban különböznek, annál nagyobb ez az érték. Mindegyik attribútum maximális értékét normáltuk 1-re, hogy súlyozhatóak legyenek. Tehát tulajdonképpen egyfajta távolságot határoztunk meg, egy maximális értékkel. Ez azonban egy adott attribútum (pl sum2) mentén a képek közti különbségek arányát megtartja, így nem sérülnek ezek az információk.

A dokumentumban a „pont” és a „kép” elnevezéseket szinonimaként használom, ugyanis a képekre a klaszterezés során, mint klaszterező pontokra tekintünk. A következő részben a választott algoritmusokat és módszereket tárgyaljuk.

3.4.1. A k-átlag, és a kiválasztás okai

A klaszterezéshez az elméleti bevezetőben bemutatott k-átlag algoritmust használtuk. A döntés mellett főként az algoritmus egyszerűsége, sebessége, könnyen implementálhatósága állt.

3.4.2. Klaszterszám, kezdeti ponthalmaz megállapítása

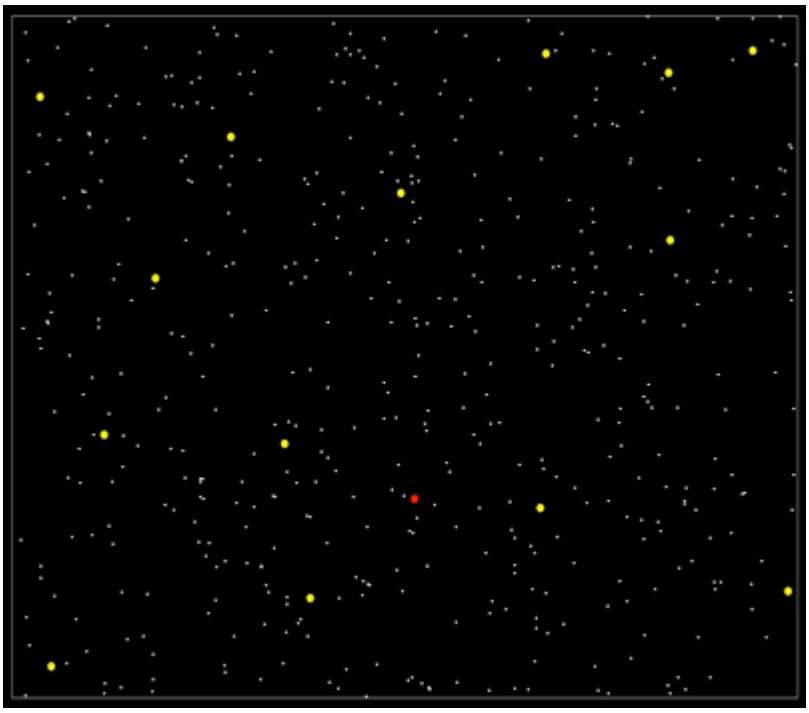
A k-átlag algoritmusnál szükséges előre megadni a klaszterek számát. Ez általában nem könnyű feladat. Ezért erre ökölszabályként a _ képletben leírt összefüggést alkalmazzák széles körben, ahol n a pontok száma, k pedig a klaszterszám.

$$k \approx \sqrt{n / 2_{\varepsilon}} \tag{7}$$

Az általunk fejlesztett alkalmazás is ezt az összefüggést használja, ám futás közben egyénileg is megadható, hogy hány klasztert szeretnénk alkotni a képhalmazból.

Az algoritmushoz szükséges kezdeti ponthalmaz meghatározására a k-átlag++ algoritmust választottuk, ezzel javítva a klaszterezés végeredményének jóságát; pontosabban a futási időt: a szükséges iterációk számát. Amint már említve volt, ez a kezdeti, kiinduló klaszterekre ad egy jó megoldást.

A módszer demonstrálására elkészítettünk egy egyszerű parancssoros demó programot, ami jól szemlélteti a működést (*kmeansplusplusdemo.py*), a generált kimenet egy kép, amit a 7. ábrán láthatunk.



7. ábra - A k-átlag++ algoritmust szemléltető tesztprogram

A program által generált képen a kis pontok a tér pontjai (azaz a képek), a színes pontok az algoritmus által kiválasztott kezdő, egy elemű klaszterek; a piros jelöli az első kiválasztott pontot. A tér most csak 2 dimenziós, a pontok két attribútuma az x és y koordinátáik. A módszert beépítettük a programunkba is, és megfelelően működik több dimenzióban is, jobb kezdőpontokat, így kiindulási klasztereket szolgáltat, mint a véletlenszerű kijelölés.

3.4.3. Klaszterezés végrehajtása

A klaszterezés néhány jól elkülönülő részből áll. Az első a kezdeti pontok kiválasztása. Majd ez után következik az iterációk végrehajtása, (azaz minden pont a hozzá legközelebbi klaszter centroidhoz tartozó klaszterbe kerül) addig, amíg már nincs változás a pontok elrendeződésében. Mi ezt kiegészítettük azzal, hogy a klaszterezést többször lefuttatjuk, és a legkisebb négyzetes hibájú megoldást választjuk ki, mint legjobb klaszterezést. A programunk így legfelső szinten egymásba ágyazott ciklusokból áll.

Az iterációk a korábban már ismertetett módon folynak. A program írása során nem volt kimondott cél az optimális futási idő és erőforrás-felhasználás elérése, sokkal inkább a prototípus elkészítése, a probléma megoldása. Az iterációk során a program információkat képes kiírni a standard kimenetre az algoritmus futásának állapotáról:

- hányadik iterációnál járunk

- hány változás történt a pontok besorolásában az adott iteráció után
- melyik klaszterben épp mennyi elem található
- az iterációk befejezése után kiírja a végső klaszterek méretét és a minimum méretet, ami felett már „elég nagy” egy klaszter; azaz a centroidjához legközelebb lévő kép már egy kiválasztandó jellemző képet jelent
- az algoritmus többszöri lefutása után kiírja a legjobbnak Ítélt, legkisebb négyzetes hibájú klaszterezés hibáját

```
iteration 0
297 changes
[43, 54, 16, 7, 42, 1, 26, 19, 28, 23, 1, 7, 41]
iteration 1
23 changes
[47, 43, 16, 7, 49, 1, 26, 18, 30, 25, 1, 7, 38]
iteration 2
6 changes
[49, 45, 16, 7, 45, 1, 26, 18, 30, 25, 1, 7, 38]
iteration 3
1 changes
[50, 44, 16, 7, 45, 1, 26, 18, 30, 25, 1, 7, 38]
iteration 4
0 changes
cluster size:  50  min size:  36.96
cluster size:  44  min size:  36.96
cluster size:  16  min size:  36.96
cluster size:  7  min size:  36.96
cluster size:  45  min size:  36.96
cluster size:  1  min size:  36.96
cluster size:  26  min size:  36.96
cluster size:  18  min size:  36.96
cluster size:  30  min size:  36.96
cluster size:  25  min size:  36.96
cluster size:  1  min size:  36.96
cluster size:  7  min size:  36.96
cluster size:  38  min size:  36.96
clustering error: 91.7222425188
----------
```

8. ábra - Az algoritmus egy futásának a szöveges kimenete

A 8. ábrán láthatjuk a futás közben kiírt adatokat, amiből jól látszik, hogy gyorsan konvergál a megoldás felé, általában néhány iteráció után kész az k-közép algoritmus.

3.5. Objektum felismerés

Következzen a képek tartalmi elemzésével kapcsolatos tervezői döntések, és megvalósítási részletek bemutatása. A SIFT módszert választottuk az elérhető algoritmusok közül, és sikeresen integráltuk a programba.

3.5.1. SIFT

A SIFT, mint módszer már jó néhány éve publikálásra került, ennek köszönhetően elég elterjedt, és sok helyen használt. A legújabb algoritmusok is ezen alapszanak, továbbfejlesztve az ötletet, így a SIFT a maga nemében mérföldkőnek számít. Választásunk főként a robosztussága, és a képtorzításokra való érzéketlensége miatt esett rá. A futási ideje nem a legjobb, de a prototípus programhoz bőven elegendő, későbbiekben lecserélhető, például az újabb SURF algoritmusra. Nem utolsó sorban, a SIFT-nek elérhető a szerző, Lowe által publikált C nyelvű programja [11] is. Ezt a parancssoros programot használjuk fel a kulcspontok generálására, Python nyelvű és parancsokkal és megfelelő paraméterezéssel.

Fontos megjegyezni, hogy a programunk jelenlegi verziójában még nem implementáltuk konkrét objektumok osztályokba sorolását. Ez azt jelenti, hogy a megtalált, két kép közötti egyezésről nem tudjuk, hogy valójában micsoda is, csak azt, hogy mindkét képen megtalálható. Konkrét objektumok (pl. cipő, labda, ház) felismeréséhez (osztályozásához) tanuló algoritmusok, tanító halmazok és módszerek szükségesek, amik a projekt jövőbeni fejlesztései között szerepelnek. Ám a jelenleg használt módszer is segíti a képek még jobb csoportosítását, és így a programunk működését. A generált kulcspontok az elforgatott, torzított képen is megtalálhatóak, vagy egy másik képen lévő képrészleten is. További számítások, például az első lépésként elvégzendő kulcspont-egyezések vizsgálata már nem képezi a SIFT részét. A kulcspont-egyezéseket mi a NNS (Nearest Neighbour Search) módszerrel vizsgáltuk, azonban a közeljövőben tervezzük az áttérést a hatékonyabb Best-bin-first algoritmusra.

3.5.2. Egyezés mértéke (QoM), eredete, fejlesztések

Két kép közötti tartalom-egyezés vizsgálatához valamilyen jósági mutatót kell felállítanunk annak érdekében, hogy eldönthető legyen, hogy találtunk-e valamilyen egyezést két kép között. Szükség van egy mértékre, ami minél pontosabban megadja két kép közötti egyezés tulajdonságait, egyben a két kép „távolságát”. A távolságot itt egészen más módon számoljuk és értelmezzük, mint a klaszterezés esetében. A távolság azonban itt is egy számérték, minél nagyobb, annál jobb a két kép közötti egyezés; a neve QoM, azaz Quality of Match.

Ennek a mértéknek alapötlete a [22] dokumentumban található, ahol különböző mérések és tesztek igazolják annak létszükségét, és alapelgondolásait. Lássunk néhány alap felvetést.
Legyenek:

- Ks : az első kép kulcspontjainak száma

- Kc : a második kép kulcspontjainak száma
- Km : a megtalált, két képen egyező kulcspontok száma

Ekkor:

- Ks >> Kc = az egyező találatok nagyon megbízhatatlanok
- Ks << Kc = az egyező találatok nagyon megbízhatóak

A futtatáskor mindig két képet hasonlítunk össze: egy „első” és egy „második” képet. A találatok megbízhatatlansága azt jelenti, hogy hiába talált egyező kulcspontokat a program, azok lehet, hogy hibásak, tehát rossz objektum felismerés történt. Tehát, ha az első kép kulcspontjaiból sokkal több van, mint a második kép leíróiból, akkor megbízhatatlan lehet a felismerés, ezért ezt a QoM-ben megfelelő módon „büntetni” kell; az ellenkező esetben pedig „jutalmazni”. Ezen felismerések a [22] dokumentumban található, tesztekkel alátámasztott méréseken alapulnak; további megfontolások is olvashatóak az írásban. A tesztek során mi is futottunk problémákba és anomáliákba. Például két, szemmel láthatóan különböző képen találtam kulcspont-egyezéseket, vagy épphogy nem voltak egyezések, a QoM érték mégis magas lett. Ezen tapasztalatok, tesztek alapján finomítottuk a képletet:

- Az R = Ks/Kc arány a tesztjek során nem rontotta le annyira a találatok megbízhatóságát; nem volt indokolt a QoM olyan mértékű büntetése, ami az írásban lévő képletben volt. Ráadásul a számítás előtt a programunk leméretezi a képeket, ez további javuláshoz vezet. Így leszabályoztuk a „büntetés” és a „javítás” mértékét, hogy bizonyos értékeknél ne legyen nagyobb; ez a *settings.py* fájlban állítható.
- A MatchPercent érték arra szolgál, hogy megmutassa, az első kép kulcspontjaiból hány százalékuk volt találat a másik kép leírói között. Ha ez az érték kicsi, az lerontja a QoM értékét. Mivel mindkét irányú egyezést kell nézni, megeshet, hogy az első kép esetében ez az arány rossz, azonban a második kép kulcspontjainak nagy százaléka egyezett. Így mi a két érték (Ks/Km és Kc/Km) közül a nagyobbikat vesszük a képlet során figyelembe.
- A MatchPercent érték nagy lehet akkor is, ha egy nagyon kevés kulcspontot tartalmazó képről van szó. Ha pl. egy kép 15 leírót tartalmaz, abból már 5 egyezés is 33%-ot jelent; holott 5 egyező kulcspont két kép között nagyon alacsony érték, csak véletlen, esetleges kisebb hiba műve lehet. Így, a settings.py fáljban egy beállítható

paraméter található, aminél kevesebb megtalált egyezés esetén exponenciálisan „büntetődik” a QoM értéke.

A QoM paraméter, ámbár javított és tapasztalatokkal kiegészített, mégis tovább finomítható, és a bemutatott paraméterek helyes megadása is kérdés lehet; a mostani értéküket gyakorlati úton állítottuk be. Ez egy, a feladat jövője során megoldandó téma, és további kutatásokat igényel.

3.5.3. A módszer beépítésének lehetőségei

Felmerül a kérdés, hogy a képeken talált egyezésekkel hogyan is tudnánk segíteni a jellemző kép kiválasztását. Az integrálást több nézőpontból is meg kell vizsgálni; melyik milyen hatással van a működésre:

- futási idő elfogadható legyen
- minél többet javítson a klaszterezésen
- könnyű használat
- implementálás egyszerű legyen

A jelenleg használt séma egy köztes megoldás, próbál a fent említett négy kritérium mindegyikének eleget tenni, azok között a középutat keresve. Több módszer létezhet, a választott megoldást ismertetnénk.

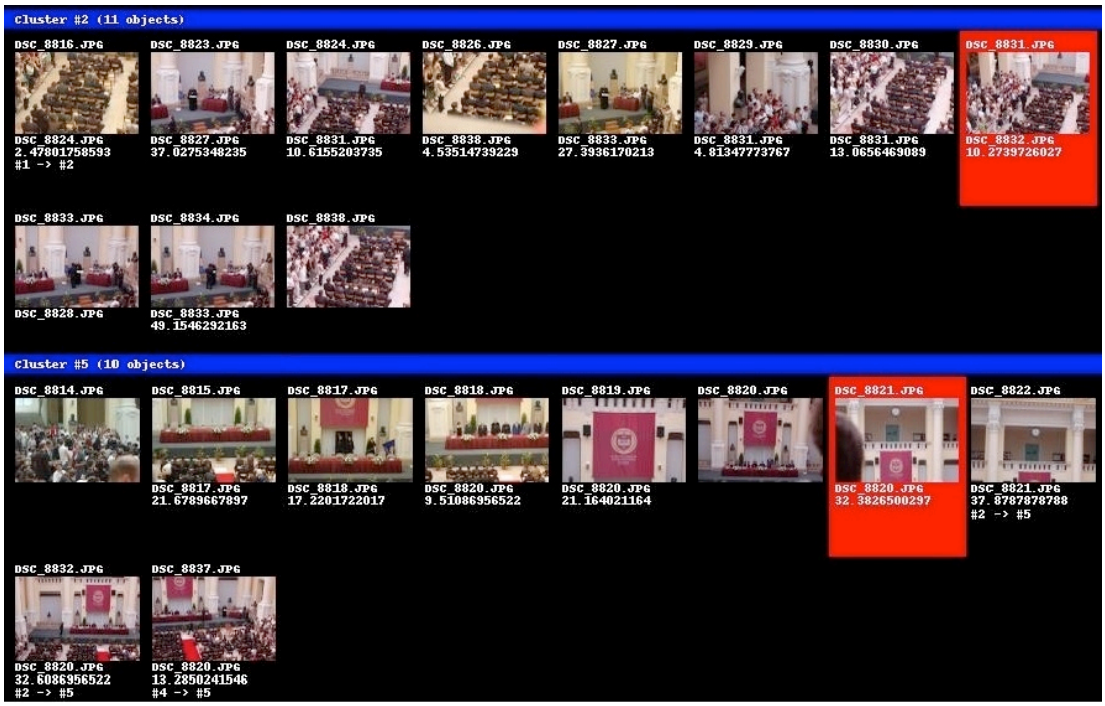
3.5.3.1. Klaszterezés módosítása

A legkézenfekvőbb megoldásnak először az tűnhet, ha az eddig használt (kvalitatív és kvantitatív) attribútumokhoz hozzávéve, egy újabbat adnánk minden képhez, megfelelően súlyozva. Így eggyel több dimenziós vektorokkal dolgoznánk; ez egy átlátható és szép megoldás lenne: implementálása egyszerű, és a klaszterezés javítása is hangolható a súlyozás beállításával. Azonban egy fontos elvi akadály adódik: a „régi” attribútumok egyetlen képre vonatkoznak, egy képet jellemeznek és arra kiszámíthatóak. Azonban a SIFT általi képegyezés (a tény, hogy találtunk valamit, ami két különböző képen is szerepel) kettő kép közös tulajdonsága, és mint ilyen, nem rendelhető attribútumként egyetlen képhez. Így távolságot sem tudunk számolni, és semmilyen egyéb műveletet, amivel a klaszterező módszerünk lefuttatható.

Fenti probléma miatt egy másik megközelítést alkalmaztunk. A k-átlag klaszterezés, az algoritmus természeténél fogva nem mindig pontosan ugyanazt az eredményt adja, lehetnek

kisebb eltérések ugyanazon képhalmazon való újbóli futtatáskor. Továbbá, mivel univerzálisan „jó”, mindenféle albumra tökéletes súlyozást és beállításokat lehetetlen találni, hibák is előfordulhatnak, például egy kép nem abba a klaszterbe kerül, ahova valóban illene.

Így adódik a lehetőség, hogy a klaszterezés hibáit utólag javítsuk egy másik módszerrel, jelen esetben az objektum felismeréssel. Ez az általunk jelenleg használt metódus, működése a következő. Először lefuttatjuk a „normál” klaszterezést, majd az összes képet páronként összehasonlítjuk a SIFT segítségével, objektum-egyezéseket keresve. Ha két képen megfelelő mértékű (QoM) egyezést találtunk, és a két kép nem ugyanazon klaszterben van, akkor áthelyezzük a neki jobban megfelelő klaszterba. A módszer működését a következő _ ábra szemlélteti:



9. ábra - A klaszterezés javítása SIFT-tel

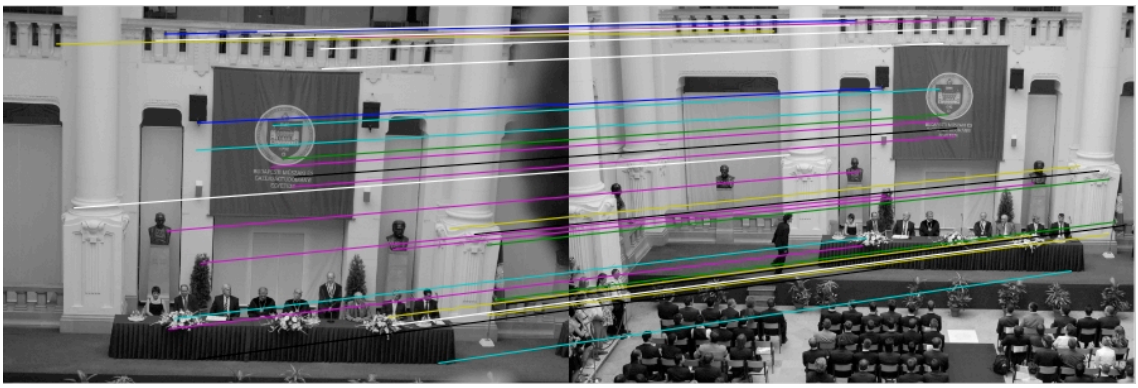
A fenti ábrán látható a program lefutása egy albumon, azaz az indexképei, a képek klaszterekre bontásával. A #2 és az #5 számú klaszterek láthatóak kivágva, mert ezek a lényegesek most számunkra. Minden kép felett látható a fájl neve. A kép alatt pedig a talált SIFT egyezés eredményei vannak, ha történt ilyen:

1. sor: kép fájlneve, amivel az egyezést találtuk

2. sor: egyezés mértéke, azaz a QoM

3. sor: ha a kép át lett helyezve az eredeti k-átlag klaszterezés által meghatározott helyéről, #HONNAN -> #HOVA formátumban látszik, hogy ez milyen módon történt

Amint látható, több kép esetén is egyezést találtunk. A legfontosabbak a #5 klaszterben lévő DSC_8822.jpg, DSC_8832.jpg és DSC_8837.jpg nevű képek. Az első kettőnél látható, hogy a #2 klaszterből lettek áthelyezve ide, a harmadik pedig a #4 klaszterből került át (ami jelen képen nem látható). Az egyezés tárgyai a képen mindhárom esetben a BME zászló, és az azt körülvevő oszlopok és pulpitus volt, ahogyan az alábbi 10. ábrán is látszik:



10. ábra - Magyarázat a SIFT általi klaszter javításhoz

Itt a SIFT egyezéseket demózó programunk kimenete látszik, két kérdéses képen futtatva, (DSC_8820.jpg, DSC_8832.jpg) amik az előző, _ ábra képén is szerepeltek. Megjegyzendő, hogy több kulcspont-egyezés is található a kép között, de az átláthatóság kedvéért csak néhány, különböző színnel jelölt vonalat jelenítettünk meg. Minden vonal két végén látható, hogy valóban ugyanazon pontokat jelölték ki, két különböző képen.

A módszerrel tehát láthatóan jól működik, azonban több kérdés is felmerülhet a gyakorlati alkalmazásával kapcsolatban:

- Mely képeket hasonlítsuk össze egymással? Nagy képhalmaz esetén a futásidő kritikus lehet a „mindet minddel” hozzáállás, és a teljes gráfok élszámát mutató n*(n-1)/2 képlet miatt. 30 képes album esetén ez már 435db SIFT összehasonlítást jelent.
    - Véletlenszerűen néhány képet néhány másikkal?
    - Minden képet csak a klaszter centroidokkal?
- Mekkora legyen az egyezés, azaz a QoM határ, ami felett áthelyezzük a képet?
- Hogyan helyezzük át a képeket?

- Kisebb klaszterban lévő képet a nagyobb klaszterbe, vagy fordítva?
- A klaszter centroidtól lévő (klaszterezés által számolt) távolságok alapján döntsünk? (Amelyik kép a kettő közül közelebb van az eredeti centroidjához, azt hagyjuk helyben)
- Csináljunk új klasztert a két képnek?
- Egyszerűen helyezzük át feltétel nélkül?

- Hogyan szüntessük meg a képek „ugrálását” klaszterek között? (Azaz többszöri áthelyezgetését)

A kérdések megválaszolása sok, ugyancsak kiterjedt tesztelést és hangolást igényel. A programunkban a QoM határ a *settings.py* fájlban állítható, jelenleg a tesztjeink által jónak ítélt értéken van. Az áthelyezés minden esetben megtörténik; a tesztek általi tapasztalatok azt mutatták, hogy pl. a centroidtól való távolság mérése sokszor hibás eredményhez vezet. Az összehasonlítások száma pedig ugyancsak a *settings.py* fájlban állítható, a „mindet mindennel” módszertől kezdve a „minden másodikat minden másodikkal” metóduson át, és így tovább. A metódusok további hangolása és fejlesztése a projekt későbbi szakaszában várható.

3.5.3.2. Egyéb lehetőségek

A másik megközelítés a fent említett, klaszterezésbe való szerves integrálás, az utólagos javítás helyett. A már vizsgált, újabb attribútumként való bevezetés nem járható út, azonban egyéb lehetőségek szóba jöhetnek:

- Konkrét objektumok felismerése a képeken, ezek már kvalitatív attribútumként egyetlen képhez is hozzárendelhetőek (pl.: 1.jpg-n található egy ház)
- Egyéb módon megoldani, hogy az objektum-felismerés (kép-egyezés) ne két kép közös tulajdonsága legyen, hanem egy-egy képhez hozzárendelhető egyéni attribútum

3.6. Videó szegmentálás

A projektünk fejlesztése során jött az ötlet, hogy a képeket osztályozó módszereink kiválóan alkalmazhatóak videók esetén is. Ahogy egy képalbumból, úgy egy videóból is kinyerhetőek az azt jellemző képek. De amíg a képeknél a különböző klaszterek “középpontjai” megfelelő

választások, addig videóknál más megközelítést kell alkalmazni. Mivel a videó egymás utáni jelenetek sorozata, érdemes ezek határai megtalálni, így a jelenetek kezdő képkockái jól jellemezhetik az adott videót. Továbbá, a jelenetek határainak megtalálása egyéb lehetőségeket, például a gyors keresést/ugrást is magában rejti.

3.6.1. Összekapcsolás a kép klaszterezéssel, integráció

Tehát, a módszer hasonló, mint a képek esetében: a videót először képkockákra bontjuk, majd ezen alkalmazzuk a kép klaszterező algoritmust. Megjegyzendő, hogy jelen esetben is használható, de nem ajánlott a SIFT objektum-egyezések vizsgálata az eddigi módon. Ugyanis egy videóból kinyert képkockák között rengeteg hasonló képet, így egyezést találnánk fölöslegesen. Másrészt a képhalmaz mérete is nagy lesz, ami a futási idő miatt kritikus. Ehelyett más, a videók természetéhez jobban illeszkedő módszereket dolgoztunk ki a klaszterezés javítására, de a SIFT-et is meghagytuk, mint lehetőséget.

A kódszintű integráláshoz, az egyszerű használat érdekében sok helyen módosításokra volt szükség a programban. Ennek eredményeképpen rendkívül egyszerűen használható a parancssoros szoftverünk videó módban is, egyszerűen a --video, --fps, és a --mode (és egyéb opcionális) kapcsolók használatával, ahogy a 11. ábrán is látható.

```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\_school\pic-vid-organizr_beta>python cluster.py -iv --video test0.avi --fps 4
 --mode scenes
video_frames, Are you sure (Y/N)? y
ffmpeg version git-N-29961-g9763420, Copyright (c) 2000-2011 the FFmpeg develope
rs
  built on May 16 2011 18:25:25 with gcc 4.5.3
  configuration: --enable-gpl --enable-version3 --enable-memalign-hack --enable-
runtime-cpudetect --enable-avisynth --enable-bzlib --enable-frei0r --enable-libo
pencore-amrnb --enable-libopencore-amrwb --enable-libfreetype --enable-libgsm --
enable-libmp3lame --enable-libopenjpeg --enable-librtmp --enable-libschroedinger
 --enable-libspeex --enable-libtheora --enable-libvorbis --enable-libvpx --enabl
e-libx264 --enable-libxavs --enable-libxvid --enable-zlib --pkg-config=pkg-confi
g
  libavutil    51.  2. 1 / 51.  2. 1
  libavcodec   53.  5. 0 / 53.  5. 0
  libavformat  53.  1. 0 / 53.  1. 0
  libavdevice  53.  0. 0 / 53.  0. 0
  libavfilter   2.  5. 0 /  2.  5. 0
  libswscale    0. 14. 0 /  0. 14. 0
  libpostproc  51.  2. 0 / 51.  2. 0

Seems stream 0 codec frame rate differs from container frame rate: 30000.00 (300
00/1) -> 29.97 (30000/1001)
Input #0, avi, from 'test0.avi':
  Metadata:
    encoder         : Lavf52.13.0
  Duration: 00:01:33.85, start: 0.000000, bitrate: 730 kb/s
    Stream #0.0: Video: mpeg4, yuv420p, 352x240 [PAR 200:219 DAR 880:657], 29.97
 fps, 29.97 tbr, 29.97 tbn, 30k tbc
    Stream #0.1: Audio: mp2, 44100 Hz, stereo, s16, 64 kb/s
```

11. ábra - Videó mód és az ffmpeg használata

A videók kezeléséhez, kódolásához és dekódolásához az ffmpeg nevű GNU-LGPL licenszű szoftvert használtuk. Ez egy rendkívül sokoldalú és könnyen használható parancssoros program.

3.6.2. Megvalósítási lehetőségek, kérdések

A jelenlegi szoftverek nagy része elejétől végéig, képkockánként végignézi a videófolyamokat, így keresve köztük különbségeket, és megtalálva a jelenet határokat. Ezzel egyrészt futásidejű gondok akadhatnak, másrészt az átúsztatásos jelenetek detektálására nem igazán alkalmas - nem fogja megtalálni a határokat. Ehelyett mi a klaszterezést alkalmaztuk, mint megoldást, különböző módszerekkel kombinálva. Ennek hátránya, hogy nem alkalmas valósidejű jelenet-detektálásra, mivel meg kell várni a klaszterezés lefutásának végét. Azonban nem is célunk „realtime” alkalmazás megalkotása, így a döntésünk megfelelőnek bizonyul.

Egy videóból generált képkockák halmaza egészen más tulajdonságokkal bír, mint egy átlagos, eddig elemzett képalbum. A viszonylag sokféle különböző kép helyett most sok, nagyon hasonló képet kapunk, amik egy klaszterbe kell, hogy kerüljenek, mivel nagy valószínűséggel egy jelenethez tartozó képkockákról van szó. Ez az alapfelvetés, azonban

sok új kérdés vetődik fel, amik a képalbum osztályozás során eddig nem jelentek meg, ezek egy részét a bevezetőben már említettem:

- Mekkora legyen egy jelenet minimum hossza? (Egy fél másodpercre beugró kép egy külön jelenetnek számít?)
- A túl rövid, vagy lényegtelen (pl. sötét képernyő) jeleneteket törölhetjük?
- Az ismétlődő, de időben távol lévő jeleneteket egybe szabad venni?
- Átúszásos hatások esetén az átúszás hova kerüljön? Esetleg külön jelenet legyen?

Ezek a kérdések szubjektív válaszokat feltételeznek, így megválaszolásuk nem egyértelmű. A minél jobb meghatározásukhoz egy megoldás lehet embereket, tesztalanyokat megkérdezni, és a kapott válaszok alapján átlagolni. A programunkban ezeket a lehetőségeket könnyen módosíthatóvá tettük, a *settings.py* fájlban állítható paraméterekkel, és választható lefutási módokkal.

A kezdeti klaszterezés elég jól elkülöníti a hasonló, így valószínűleg egy jelenetbe tartozó képeket külön-külön klaszterekbe. Azonban sok, a képeknél jó megoldás most hibás eredményt adhat. A képkockákat az ffmpeg segítségével úgy állítjuk elő, hogy a fájlnevek növekvő sorrendben legyenek, így az első képkocka pl. img00000001.jpg, a második img00000002.jpg, és így tovább. Tehát, ha a klaszterezés során egy folytonosan eggyel növekvő sorszámú klaszter képei közé kerül egy nagyobb sorszámú, sorrendben nem oda illő kép, akkor az egy másik, későbbi jelenet képe, és itt rossz helyen van, hiába hasonló a képi világa az ebben a klaszterben lévő képekéhez. Így a klaszterezés a videó szegmentálásra nézve hibákat véthet, amiket megfelelő módszerekkel javítani kell, akárcsak a SIFT-tel tettük.

3.6.3. Szegmentáló módszerek elemzése, megvalósítása

A javítások tehát abból állnak, hogy a lefuttatott klaszterezés által létrehozott klasztereket szükség esetén módosítsuk utólag, olyan módon, hogy az új klaszterek már valóban egy-egy jelenet képkockáit tartalmazzák, azaz csak egymás után következő képeket. Fontos megemlíteni, hogy a terminológiában itt a klaszter és jelenet szavak ugyanazon fogalmat jelentik. A 12. ábrán látható egy klasszikus, klaszterezés által vétett hiba, aminek a javítása szükséges:

12. ábra - Klaszterezési hiba videó szegmentálás esetén

Látható, hogy a 197-es kép után a 203 következik, azaz megszakad a folytonosság, itt kimaradt néhány képkocka. Továbbá, a klaszter végére került két, egyáltalán nem ide való kép is (370, 371 számúak), amik mind az időbeli távolság, mind a képi világ miatt biztosan egy másik klaszterbe, így jelentbe kellene, hogy tartozzanak. Az általunk kitalált javító módszereket két fő szempont szerint osztályozzuk:

- Műveletek a jelenetekkel (törlés, egyesítés, stb.)
- Mérőszám a műveletekhez (jelenet hossz, képek távolsága, stb.)

1. Műveletek a jelenetekkel

A 12. ábrán látható, és hasonló hibák javítását néhány alapvető művelettel végezhetjük el, amik használata program szinten történik:

- delete - egy jelenet törlése
- cut - egy jelenet szétvágása
- join - két jelenet egyesítése
    - bármely jelenetet bármelyikkel
    - csak egymást követő jeleneteket
- check_repeated - két hasonló jelenet keresése

Ezek az alapműveletek implementálásra kerültek a programban, tetszőleges sorrendben használhatóak, akár többször is egymás után. Megjegyzendő, hogy a szétvágás művelet használata esetén a _ ábrán látott, folytonossági hibák mindenképpen először automatikusan javításra kerülnek, azaz mérőszámtól függetlenül, ha egy nem a sorban következő képkocka jön egy klaszterben, a klaszter ott mindenképp szétvágásra kerül. A sorból kicsit kilóg az utolsó művelet, ugyanis az nem javításra való. Egyszerűen megnézi, hogy melyek az

egymáshoz hasonló jelenetek, és ezt a lefutás végén megjeleníti; így célszerű legutolsó műveletként használni, amikor már kialakítottnak tekintjük a jeleneteket.

2. Mérőszám a műveletekhez

A bemutatott négy alapművelet használatához meg kell adni a feltételeket is, azaz hogy milyen feltételek mellett töröljük a jelenetet, avagy egyesítsünk kettőt. Ezek a feltételek mérőszámok, és azok kapcsolatát jelentik. A lehetséges mérőszámok:

- *time_diff* - két képkocka közötti időkülönbség
- *dist* - két képkocka közötti (klaszterezés szerinti) távolság
- *length* - egy jelenet hossza
- *sift* - két képkocka közötti SIFT egyezéssel vizsgált QoM érték

Ezen mértékegységek határai a *settings.py* fájlban találhatóak, így könnyen állíthatóak igények, és tesztek eredményei alapján. Minden mértékből két verzió is megtalálható a fájlban, a videó szegmentálásnak két működési módját alkottuk meg. Az egyik módban jelenet határokat keresünk, a másikban pedig különböző videókból összeollózott videó esetén, a videó határokat. A működési mód a –mode kapcsolóval szabályozható, jelenleg használt értékei a „scenes” vagy a „joined”.

A két képkoca közötti mértékeket mindig jelenethatárokon vizsgáljuk, azaz például a #2 jelenet utolsó képkockája, és a #3 jelenet első képkockája között. Így például „join” művelet és „dist” mérték használata esetén megállapítható, hogy a két jelenetet a klaszter-távolság alapján egyesíteni kell-e vagy nem. A kódszintű használatára egy példa:

```
join_near_scenes_following(
     fps,program_mode=mode,
     check_mode=['OR','length','dist'])
delete_short_scenes(fps,program_mode=mode)
```

Amint látható, egy logikai művelet is megadandó az egyes alapműveletekhez, a még jobb eredmények és hangolhatóság érdekében. „OR” esetén, ha bármelyik, utána felsorolt mérték eléri a megadott határt, a művelet végrehajtásra kerül, azaz az éppen vizsgált két jelenet egyesítődik egy jelenetté. Értelemszerűen „AND” használatakor minden mértéknek el kell érnie a határt, hogy a művelet lefusson.

Látható tehát, hogy a „minden képkockán végigszaladunk” módszer helyett mi másféle megközelítést alkalmaztunk. Egy elég jó jelenetfelbontásból indulunk ki, amit a klaszterezés

adott. Ezt javítva, a jeleneteket módosítva (tologatva) alakítjuk ki a végső felbontást és jelenethatárokat az általunk fejlesztett műveletekkel. Így jelöljük ki a videót jellemző képeket.

3.7. Végeredmény kiértékelése

A klaszterezést többször is lefuttatjuk. Minden kiszámított klaszterezésre kiszámoljuk a teljes négyzetes hibát, ez lesz az elsődleges minőségi mutató. A legkisebb hibával rendelkező klaszterezést vesszük, mint legjobb felosztást. Ebből a klaszterezésből kiválasztjuk a „megfelelő" méretű klasztereket: egy klaszter akkor számít megfelelő méretűnek, ha a pontjainak a száma legalább a teljes halmaz számának egy megadott hányada, pl.: 15%-a (Ez az érték beállítható a konfigurációs fájlban.). Ahány megfelelő méretű klaszter kiválasztásra került, annyi jellemző képet kapunk eredményként, mivel utolsó lépésként ezekből a klaszterekből egy-egy kép kerül kiválasztásra, mégpedig a klaszter-centroidhoz legközelebb lévők.

Videók esetén a klaszterezés után lefutnak a szegmentáló metódusok, majd a javított klaszterek lesznek a megtalált jelenetek. Mindegyiknek a legelső képei a videót jellemző képek lesznek, továbbá kiírásra kerülnek a jelenetek, és azok körülbelüli kezdő időpontja.

Mind a kép klaszterezés, mind a videó szegmentálás futása során, és a futás végén sok információ jelenik meg a parancssorban. Szemléltetésként a videó mód futásnak végeredménye látható a 13. ábrán. ahol megfigyelhetőek a megtalált jeleneteket, kezdési idejükkel és kezdőképükkel, valamint a hasonló jelenetek.

```
SIMILAR SCENES: # 3  - # 5
SIMILAR SCENES: # 3  - # 7
SIMILAR SCENES: # 4  - # 6
SIMILAR SCENES: # 5  - # 7

######## Scenes (time: hh:mm:ss) ########
# 1 start time: 00:00:00 ; start img: img0000001.jpg
# 2 start time: 00:00:02 ; start img: img0000012.jpg
# 3 start time: 00:00:32 ; start img: img0000130.jpg
# 4 start time: 00:00:37 ; start img: img0000151.jpg
# 5 start time: 00:00:42 ; start img: img0000173.jpg
# 6 start time: 00:00:47 ; start img: img0000191.jpg
# 7 start time: 00:00:51 ; start img: img0000208.jpg
# 8 start time: 00:00:56 ; start img: img0000229.jpg
```

13. ábra - Videó szegmentálás lefutásának eredménye a konzolban

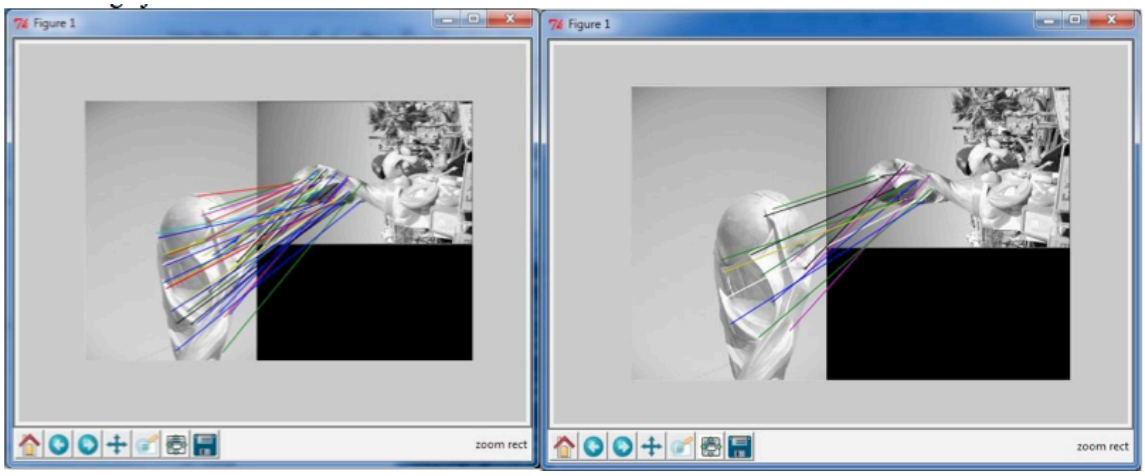
3.8. Megjelenítés - demózás

A konzolban történő adatok megjelenítésén túl a programhoz többféle demózási lehetőséget is készítettünk. Ezek mind teszteléshez, mind a megoldás működésének demonstrálásához kiválóak; ezek ismertetése következik.

3.8.1. Objektum felismerés szemléltetése

Az alkalmazáshoz elkészült egy olyan modul, melynek segítségével az objektum felismerés szemléltethető. Lehetőség van egy kép kulcspontjainak kirajzolására, illetve két kép összehasonlítására a talált kulcspontok alapján.

A 14. ábrán két kép összehasonlítása, egyező részek keresése látható a SIFT által. Az első képen 139 kulcspont, míg a másodikon 2127 található. Ezekből 57-et egyezőnek talált a program. A bal oldali összehasonlításon mindegyik megtalált pont, a jobb oldalin pedig a „less” paraméterrel szabályozva, csak minden 4. egyező kulcspont látható összekötve, a láthatóság kedvéért. Bár a csak serleget tartalmazó kép elforgatott, blur és grayscale effektekkel torzított, a program mégis megtalálta a serleget, azaz annak adott pontjait a jobb oldali képen.



14. ábra – Két kép összehasonlítása a SIFT alapján

A korábban említett QoM érték jelen esetben 45.5635, ami igen jónak számít, ugyanis a tapasztalataink és a tesztek szerint kb. 2.7 felett már szemmel látható egyezések vannak két kép között.

3.8.2. Indexkép rajzolása képalbumból

A program tartalmaz egy klaszter-kirajzoló függvényt, ami jelentősen segíti a tervezést, implementációt és tesztelést. Ez a klasztereket, és a hozzá tartozó képeket könnyen átlátható formában, indexképként megjeleníti, ahogy az a 15. ábrán alant látható.

15. ábra - Egy teszthalmazból generált indexkép

Amint látható, a választott jellemző képek pirossal vannak bekeretezve. Öt klaszter jött létre, ebből három elég nagy, így három indexképet kaptunk. Jól láthatóak a különböző csoportok is: a fenti egyedüli kép egy túl sötétre sikerült fotó, a #2 a növényzetet ábrázoló zöld képek. A #3 klaszter a hotel sárgás képei, a #4 egy kevertebb csoport, míg az #5 néhány hasonló fotó a falról.

3.8.3. Videó jelenetek szeparálásának demózása

A videó jelenet detektáláshoz is készült egy bemutató program, ami ugyancsak mind teszteléshez és fejlesztéshez, mind a program demózásához nagyon hasznos; az 16. ábrán látható a kimenete:

16. ábra - Videó jelenet szeparálás demózása

Ez valójában egy videó, amiből egy kivágott részt láthatunk. A program a futás után a meglévő képkockákból újra összeállítja a videót, az ffmpeg segítségével, .avi formátumba. Azzal a különbséggel, hogy a generált videón a bal felső sarokban látható mindig hogy éppen melyik jelenetnél tartunk, a jobb alsóban pedig az ehhez hasonló jelenet(-ek) sorszáma.

4. Elemzés

A módszerek bemutatása után a most következő fejezetben több szempont alapján elemezzük az elkészült megoldásunkat. Az elemzés első részében a jelennel foglalkozunk, a program jelenlegi verziójáról, annak működéséről teszünk megállapításokat, a működés jóságát vizsgáljuk. A második szekcióban a jövő problémái, feladatai kerülnek bemutatásra: a lehetséges, és folyamatban lévő továbbfejlesztési lehetőségeket soroljuk fel, ezután megemlítünk néhány ígéretes gyakorlati alkalmazási lehetőséget.

4.1. Értékelés valós teszthalmaz alapján

A módszer jóságának bemutatására egy tesztet dolgoztunk ki. A feladat tehát: fényképalbumokból kell kiválasztani a legjellemzőbb képeket. Mivel ez szubjektív választás, ezért emberi véleményekkel vetjük össze a gépi eredményeket.

4.1.1. Teszthalmaz

Összegyűjtöttünk egy 20 albumból álló képhalmazt. Az albumok egyenként 30 képet tartalmaznak, így összesen 600 képből áll. Ezen albumok különböző témájúak (nyaralás, buli stb.), azonban albumon belül valamely szempontból kapcsolódnak egymáshoz. A képek nagy része EXIF információkat is tartalmaz.

4.1.2. Összehasonlítás

Három emberi alany a szubjektív véleménye alapján kiválasztotta mindegyikből a legjellemzőbb képeket, pontosabban sorba rendezték azokat. Ez úgy történt, hogy minden albumnál el kellett dönteniük, hogy melyik a legjellemzőbb, a második legjellemzőbb stb. Így minden albumnál személyenként kaptunk egy rendezést a képekre. A megvalósított alkalmazásunk végeredményét (albumonként egy legjellemzőbb kép) összehasonlítottuk ezekkel. Az összehasonlítás az emberi rendezés összegzett eredménye alapján történt. Az összegzéshez Borda módszerét használtuk. A gépi eredmény nem minden esetben egyezett meg a szubjektív teszt végeredményével, azonban az albumok 30%-ánál (p5=30%) az első 5 legjellemzőbb kép között volt. Számítást végeztünk továbbá, hogy a gépi eredmények az esetek 60%-ában az első 10 (p10=60%) reprezentatív kép között van. Ezen eredmények természetesen nem tökéletesek, azonban elegendőek, és a szubjektív véleményezés

ellenére is értékelhetőek. Megvizsgáltuk az ember általi sorbarendezéseket, és azt tapasztaltuk, hogy ezeknek viszonylag nagy a szórása. Kereszt-validációval két alany rendezését ellenőriztük és összegeztük, majd összehasonlítottuk a harmadik fél eredményével. Az eredményt a következő táblázat szemlélteti:

	1	2	3	Gépi eredmény
P5	30%	40%	45%	30%
P10	85%	75%	70%	60%

1. táblázat – A keresztvalidáció eredménye

Az összehasonlítás eredménye az elsőnél p5=30%, p10=85%, a másodiknál p5=40%, p10=75%, a harmadiknál pedig p5=45%, p10=70%. Ezen számok alapján a megvalósított automatikus eljárásunk majdnem olyan „jó”, mint az emberek választása.

4.2. Továbbfejlesztési lehetőségek

A program jól működik, a klaszterezést meglepően jó hatásfokkal elvégzi. Több képhalmazra is teszteltük, és a paramétereket jól beállítva kiváló eredményeket kaptunk. További fejlesztések természetesen még váratnak magukra, amik még tovább javítják a működési hatékonyságot; ezek egy része kutatást és teszteket igényel:

- Még több jellemző kinyerésével egy képből. Ezek lehetnek bonyolultabbak is, például valamilyen témafelismerés, objektumok vagy arcok detektálása.
- Az attribútum-súlyok az alkalmazott próbálgatós módszer helyett meghatározhatók valamilyen tanuló algoritmussal, amely során tanuló és validációs képhalmazokat használunk fel. Esetleg „közvélemény-kutatás” jelleggel emberek által is elkészíthető a tanító halmaz; ennek során képek közötti hasonlóságot pontozhatnának a tesztalanyok, majd ez, és a képek vizsgálata alapján jól közelíthetőek a megfelelő súlyok; azaz hogy mely tulajdonságok mennyire számítanak a képhalmazok osztályozása során. A klaszterezés jósága a validációs halmaz alapján történhet a tévesztési mátrix vizsgálatával.
- A kezdeti klaszterszám meghatározása is fontos feladat, erre az ökölszabály helyett további módszerek kereshetőek. Egy megoldás lehet az egész algoritmus többszöri lefuttatása, néhány különböző kezdeti klaszterszámmal, majd jósági mutatók alapján a legjobb kiválasztása.

- Optimalizálások, a futás gyorsítása (pl. jobb NNS kulcspont-egyezést vizsgáló algoritmus helyett Best-bin-first)
- Sok, a settings.py-ben tárolt paraméter és mértékhatár további finomhangolása, nagy teszthalmazok és kérdőívek eredményei alapján
- A QoM mérték további finomítása kép másik klaszterbe való áttevésének, mikéntjének fejlesztése.
- Az objektum felismerés egyéb módon történő beépítése a programba

Mivel elég szubjektív döntésekről beszélünk a feladat megoldása során, több problémával is szembesülünk: egyrészt nem lehet kijelenteni, egy adott klaszterezésre és képkiválasztásra általánosan, hogy „jó”, mivel másoknak mást jelenthet a jó felosztás, és a jó indexkép. Másrészt nagyon különbözőek lehetnek a képhalmazok, ami megnehezíti a feladatunkat. Ezen problémák megoldására többek között adaptív módszerek alkalmazhatóak, vagyis a program minden fontos paramétert és futási jellemzőt az adott képhalmazhoz optimalizálva állít be. Ezek az attribútumok következőek:

- kvantitatív tulajdonságok maximális értéke (a normalizáláshoz)
- klaszterek száma
- „elég nagy” klaszterméret értékének beállítása
- tulajdonságok súlyozása

Videók esetén ugyancsak alkothatóak adaptív, az adott videóhoz igazodó beállítások és módszerek, főként a klaszterezést javító szegmentáló műveletek esetében. Ilyen módon rendkívül flexibilis, és intelligens megoldásokkal bővíthető a szoftverünk.

4.3. Gyakorlati alkalmazási területek

Az általunk elkészített megoldásnak, és a klaszterezés köré épülő módszereknek rengeteg alkalmazási területe van mind online, mind offline környezetben. Nem csak a nagyközönség számára elérhető szolgáltatásokról beszélünk, egyéb alternatív módok is elképzelhetőek. Az egyes területek számára egyedi módosítások lehetnek szükségesek a programban, az igényeknek megfelelően.

Képek esetén magától értetődik a webes és offline képkezelő, album rendező alkalmazásokba való beépítés lehetősége. Így a sok száz, vagy ezer készített családi képek rendezve és könnyen áttekinthetővé válnak. Például egy album mappájára rátekintve azon látható néhány, kiválasztott jellemző kép, így azonnal láthatjuk az album tartalmát. Amint a bevezetőben is említettem, a megoldás egy egyszerűsített, optimalizált verziója akár operációs rendszerek részeként is megjelenhet: pl. Windows-ban a képeket tartalmazó mappákon megjelenhetne a releváns, kiválasztott néhány fotó, a jelenlegi „első négy" helyett.

A videó szeparálás, azaz jelenet detektálás ugyancsak sok lehetőséget rejt magában. Leghasznosabb az online videómegosztó portálokon lehetne, például a Youtube-on. A videók listájánál a videó neve alatt nem csak egy képet láthatnánk, hanem az egérkurzor odavitele esetén néhány jellemző, kiválasztott képet egymás után váltogatva, így azonnal következtethetünk a videók tartalmára; még mielőtt betöltöttük volna azt, időt és sávszélességet foglalva. A kiválasztott képek a megtalált jelenetek első képkockái lennének. Egy másik felhasználási mód a gyors tekerés/ugrás megoldása. Egy hosszabb videó, például híradó esetén megeshet, hogy a felhasználó nem kíváncsi az egész anyagra, csak az őt érdeklő részekre. Ez esetben, a lenti, videó folyamatát mutató kereső csíkon a szoftver jelölőket helyezhet el, a megtalált jelenetek elejére. Ezek a jelölők akár fel is címkézhetőek. Így a felhasználó azonnal az őt érdeklő részhez ugorhat, a videóban való felesleges keresgélés nélkül.

A videó elemzésnek sok további alternatív felhasználási módja lehet, ezek közül is említenék egyet. Célirányos optimalizációkkal és módosításokkal a megoldásunk alkalmassá tehető például biztonsági videokamerák tartalmának elemzésére. Adott egy éjszakai, bejárati ajtóra néző kamera. Amennyiben betörés történt, le lehet futtatni az aznapi videó anyagon az elemzést. A jelenet detektáló a klaszterezés segítségével hamar megtalálja a sok órányi eseménytelen képkocka között azokat, ahol történt valami, a videófolyamot jelenetekre bontva. Rögtön megtudható az is, hogy mikor történt az adott esemény, így elkezdhető a nyomozás. Ezen módszer esetén a célirányos optimalizálás azért szükséges, hogy a hosszú videó miatti sok képkockát megfelelő sebességgel tudjuk kezelni. Továbbá, valószínűleg sok ugyanolyan, eseménytelen képkockánk lesz, így ügyes módszerekkel sokat spórolhatunk a futási időn.

5. Összefoglalás

A munkánk során a kapcsoló szakirodalom áttekintése után, tervezői döntések mentén létrehoztunk egy új módszert, amely segítségével jellemző képet választhatunk ki albumokból és videókból. Jelen dolgozat egy éves időtartamú munkát foglal magában, amelynek felénél az addigi eredményeink egy lektorált, rangos nemzetközi konferenciára kerültek be, ahol 7 bíráló visszajelzései alapján döntöttek a konferenciára való bejutásról [23].

A kifejlesztett, klaszterzésen alapuló módszerünk képes a kép- és videóadatokat beolvasni és elemezni a statisztikai jellemzők, a hozzárendelt metaadatok és a tartalma alapján. A beolvasást követően intelligensen csoportokat alkot, és jellemző képeket választ ki. Az alkalmazás szöveges kimenetén felül képes szemléltető indexképeket is generálni, amely jó demonstrációs lehetőségeket nyújt. A módszer helytállóságának bizonyítására egy szubjektív tesztet hajtottunk végre, amely igazolta az elvárásokat, és további fejlesztésekre ösztönöz minket. A szoftver videó elemzés területén is jól teljesít már a jelenlegi fázisban is, nagy potenciált rejt magában a jövőre nézve.

Az alkalmazás jelen állapotában felhasználható, a parancssoros felület és szöveges kimenet által könnyen integrálható, azonban a munkánk nem ért véget, folyamatosan dolgozunk a továbbfejlesztésen mind a képek, mind a videók irányában. A projekt méretei, bonyolultsága és a szubjektív problémakör miatt sok, magunknak feltett kérdés megválaszolása és fejlesztési irányok kijelölése még folyamatban van. Ezek egy része tervezői döntéseket, másik hányaduk teszthalmazokat és külső véleményeket, kérdőíveket igényelhet. Azonban a lehető legtöbb ilyen beállítást könnyen változtathatóvá tettük, és úgy alkottuk meg a programot, hogy már a jelenlegi verziójában is jól működjön, iteratív fejlesztést lehetővé téve.

6. Rövidítésjegyzék

EXIF	Exchangeable Image file Format
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
CBIR	Content Based Image Retriveal - Tartalom alapú kép visszakeresés
SIFT	Scale Invariant Feature Transform
SURF	Speeded Up Robust Feature
GLOH	Gradient Location and Orientation Histogram
PIL	*Python Imaging Library*
RMS	Root Mean Square
RGB	Red, Green, Blue
NNS	Nearest Neighbour Search
QoM	Quality of Match

7. Irodalomjegyzék

[1] S. Boll, P. Sandhaus, and U. Westermann, “Semantics, content, and structure of many for the creation of personal photo albums”, ACM Multimedia „07, Proceedings of the 15th international conference on Multimedia, Augsburg, Germany, September 24-29, 2007, pp. 641- 650, DOI: 10.1145/1291233.1291385

[2] K. Vaiapury and M. S. Kankanhalli, “Finding interesting images in albums using attention", Journal of Multimedia, Vol. 3., Num. 4., October 2008, pp. 2-13.

[3] W.-T. Chu and C.-H. Lin, “Automatic selection of representative photo and smart thumbnailing using near-duplicate detection” MM „08, Proceeding of the 16th ACM international conference on Multimedia, Vancouver, Canada, October 26-31, 2008, pp. 829-832, DOI: 10.1145/1459359.1459498.

[4] L. Itti and C. Koch, “A saliency-based search mechanism for overt and covert shifts of visual attention” Vision Research 40, 2000, pp. 1489–1506

[5] L. Itti, C. Koch, and E. Niebur, “A model of saliency-based visual attention for Rapid Scene Analysis”, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 20, Issue 11, 1998, pp. 1254- 1259, DOI: 10.1109/34.730558.

[6] E. Potapova, M. Egorova, and I. Safonov, “Automatic Photo Selection for Media and Entertainment Applications”, GraphiCon”2009, Proceedings of The 19th International Conference on Computer Graphics and Vision, October 5-9, 2009, Moscow, Russia, pp. 117-124.

[7] D. Arthur, S. Vassilvitskii “k-means++: The Advantages of Careful Seeding”

[8] J. Sander, M. Ester, H. Kriegel, Xiaowei Xu, “Density-Based Clustering in Spatial Databases:The Algorithm GDBSCAN and its Applications”

[9] Mobile OCR, Face and Object Recognition for the Blind - http://www.seeingwithsound.com/ocr.htm

[10] Peter M. Roth, M. Winter “Surevey of Appearance-Based Methods for Object Regognition”

[11] David G. Lowe “Distinctive Image Features from Scale-Invariant Keypoints”

[12] H. Bay, T. Tuytelaars, Luc Van Gool “SURF: Speeded Up Robust Features”

[13] K. Mikolajczyk, C. Schmid "A performance evaluation of local descriptors", IEEE Transactions on Pattern Analysis and Machine Intelligence, 10, 27, pp 1615--1630, 2005.

[14] Chong-Wah Ngo, Yu-Fei Ma, Hong-Jiang Zhang “Video summarization and scene detection by graph modeling”

[15] HandySaw DS - Powerful Automatic Optical Video Scene Detection Tool http://www.davisr.com/cgi-bin/content/products/handysaw/description.htm

[16] Scenalyzer Live http://www.scenalyzer.com

[17] Scene Detector http://www.scene-detector.com/

[18] Adobe Premiere Pro CS3 - Help Resource Center http://livedocs.adobe.com/en_US/PremierePro/3.0/help.html?content=WS1c9bc5c2e465a58a91cf0b1038518aef7-7f9f.html

[19] Rainer Lienhart “Comparison of Automatic Shot Boundary Detection Algorithms”

[20] Python Imaging Library Handbook - http://www.pythonware.com/library/pil/handbook/

[21] The **EXIF.py** project - http://exif-py.sourceforge.net

[22] Thomas Bakken “An evaluation of the SIFT algorithm for CBIR”

[23] Gábor Szűcs, Tamás Leposa, Sándor Turbucz „Representative Picture Selection from Albums.” In: The Third International Conferences on Advances in Multimedia (MMEDIA 2011). Budapest, Magyarország, 2011.04.17 - 04.22.

8. Ábrák jegyzéke

9. Táblázatok jegyzéke