Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

Ismeretlen képhalmaz hasonló objektumainak összerendelése automatikus klaszterezéssel

TDK dolgozat - 2014

Papp Dávid

Konzulens: Dr. Szűcs Gábor

Absztrakt

A képfeldolgozás egyik fontos területe a képi tartalmak csoportosítása, más néven klaszterezése. Ennek feladata a képek csoportosítása úgy, hogy a hasonlóak azonos csoportba, míg az eltérőek különböző csoportba kerüljenek. A probléma nehézségét jól szemlélteti, hogy nem létezik olyan algoritmus, amely tetszőleges bemenet esetén megoldaná a feladatot. Éppen ezért a témában intenzív kutatómunka folyik mind a mai napig.

Jelen esetben nem álltak rendelkezésre metaadatok, így kizárólag a képek tartalmi információit használtam fel a klaszterezéshez. A kép tartalmának matematikai reprezentálása egy újabb problémát vet fel, amely a magas dimenziószámnak köszönhető. Célszerű tehát a képeket szegmentálni, hogy csak a lényeges tartalmak reprezentálása legyen szükséges. További nehézséget jelent, hogy a legtöbb klaszterező eljárás bemeneteként meg kell adni a klaszterszámot, tehát előismeret szükséges a képhalmazról.

Dolgozatomban bemutatom azt az általam készített szemantikus képklaszterező rendszert, amely ismeretlen képhalmaz esetén is hatékonyan működik. A képek szegmentálása nem igényel előzetes tanulási folyamatot, valamint a klaszterező algoritmusom automatikusan meghatározza az optimális klaszterszámot. A szegmentált képek szemantikai elemzéséhez state-of-the-art képfeldolgozási módszereket használok, mint például a GMM (Gaussian Mixture Model) alapú Fisher-vektor.

Az elkészített rendszert ismeretlen képeken tesztelem, melynek lépéseit részletesen áttekintem, eredményeit pedig kiértékelem. A teszt képeket az ImageCLEF idén meghirdetett, úgynevezett LifeCLEF versenyének weboldaláról töltöttem le. A versenyfeladat eredetileg a képek osztályozása, viszont a klaszterezés hatékonyságának méréséhez az osztálycímkék tökéletesen használhatók.

1 BEVEZETÉS

Manapság, a digitális világ növekedésének köszönhetően rengeteg fénykép található különböző adathordozókon, szerte a világban. Az interneten fellelhető képi tartalmak száma pedig megbecsülhetetlen. Korunk egyik alapvető problémája ennek a hatalmas adatmennyiségnek a rendszerezése. Ez rendkívül fontos számtalan alkalmazás számára, amelyek képekkel foglalkoznak, gondoljunk csak egy képkereső programra, vagy képnézegetőre. Ezen alkalmazások többféle módszert használnak a felhasználók segítése érdekében, hogy minél egyszerűbben találjanak számukra minél relevánsabb fényképeket, valamint több típusú részfeladatot kell megoldaniuk, mint például a képek osztályozása, rangsorolása, csoportosítása.

A munkám motivációját az adta, hogy eddig képek osztályozásával foglalkoztam, és ezen a téren még egy nemzetközi képosztályozási versenyen is részt vettem, de a képek csoportosítása újabb izgalmas kihívásnak tűnt számomra, így belefogtam ennek a kutatásába. A nemzetközi verseny az ImageCLEF idén meghirdetett, úgynevezett LifeCLEF versenye volt, ahol növényekről készült fényképek alapján kellett faj besorolást végezni. A verseny nyáron (2014 nyarán) fejeződött be és a többedmagammal elkészített megoldásunkról szeptemberben egy konferencia cikk készült [36]. A cikkünkben leírt képfeldolgozási módszereket felhasználva egy másik cél (a csoportosítás) érdekében továbbfejlesztettem a munkámat, melynek eredményét ebben a TDK dolgozatban ismertetem.

Képek csoportosítása (más néven klaszterezése) egy fontos és nehéz képfeldolgozási probléma. Ennek feladata, hogy csoportokat – klasztereket - alakítson ki úgy, hogy az azonos klaszterbe tartozó képek jobban hasonlítsanak egymásra, mint más klaszterekben lévőkre. Többféle alkalmazási területe van, mint például földrajzi területek csoportosítása, ujjlenyomatok rendszerbe szervezése, képek szegmensre bontása és még sorolhatnám. Fontos előfeldolgozási lépése is lehet egy nagyobb képfeldolgozási folyamatnak, például karakterfelismerés esetén a különböző karaktereket klaszterezve, már csak ez egyes csoportokat kell felismerni. Valamint lehet végső lépés, például internetes keresők találati listáit csoportosítva jeleníthetjük meg. A feladat megoldása

nagyban függ az aktuális körülménytől, mint, hogy milyen típusú képekről van szó, vagy milyen adatok állnak rendelkezésünkre az egyes képekről.

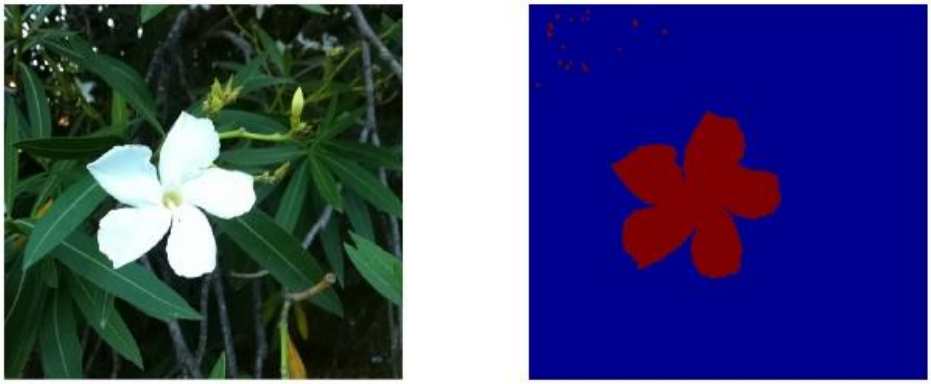
1.1 SZEMANTIKUS KÉPKLASZTEREZÉS

Az előzőekben már intuitívan definiáltam a klaszterezés fogalmát, melyet most formálisan is megadok. Legyen adott egy $I = \{I_1, I_2, \dots, I_N\}$ képhalmaz, valamint a K pozitív egész szám. Az $I_{R1} \rightarrow C_1, I_{R2} \rightarrow C_2, \dots, I_{RK} \rightarrow C_K$ hozzárendelések halmaza az I képhalmaz egy K számú klaszterbe történő klaszterezése, amennyiben a következő fennáll: $\forall X \in I_{Ri}, \forall Y \in I_{Rj} : T(X,Y)_{i=j} > T(X,Y)_{i \neq j} \quad (i,j = 1 \dots K)$, ahol I_{Ri} az I képhalmaz egy részhalmaza, C_i egy klaszter, $T(X,Y)$ pedig egy hasonlóságot kifejező tulajdonságfüggvény amely mentén a képek megkülönböztetése történik. Jól látható tehát, hogy a klaszterezés eredménye nagyban függ a választott T függvénytől. Ez általában valamilyen hasonlóságmérték (fordított reláció esetén távolságmérték) szokott lenni, amelynek meghatározása nem triviális, hiszen számos jellemzője definiálható az egyes képeknek. A hatékony T választás érdekében szükséges egy reprezentációs tér definiálása, amely minden képre azonos tulajdonságokat foglal magában. Abban az esetben, ha metaadatok nem állnak rendelkezésre, tehát nincs megadva a képekhez semmilyen plusz adat, akkor kizárólag azok tartalmi információiból tudjuk ezeket a jellemzőket, tulajdonságokat kinyerni. Ilyenkor a képen ábrázolt tartalom dönti el, hogy az melyik csoportba tartozik, vagyis ebben az esetben szemantikus képklaszterezésről beszélünk.

Ezek a jellemzők (jellemző vektorok) matematikai módszerek segítségével kaphatók meg a kép pixeleinek, színeinek, textúráinak feldolgozásával. Jellemző kinyerő eljárásokkal rengeteg féle információt kapunk, így a reprezentációs tér dimenzió száma nagymértékben megnőhet. A jellemzők számának csökkentése érdekében a képek előterét elválasztom a háttértől, így megkapom annak fontos részét, és a klaszterezési döntés kizárólag ezt felhasználva születik. Az elválasztást képszegmentálási módszerrel valósítottam meg.

1.2 KÉPSZEGMENTÁLÁS

A képszegmentálás is felfogható egy csoportosítási feladatnak, mely során a kép pixeleit rendeljük az egyes csoportokhoz valamilyen módszer alapján, amely alkalmas a pixelek közti hasonlóság mérésére. Az azonos csoportba tartozó pixelek azonos értéket kapnak, a szomszédos, de eltérő csoportba tartozók pedig lényegesen eltérőt. Amennyiben ezek a hozzárendelt értékek színkódok, a pixeleket ezekkel megjelenítve látványos az eredmény, lásd 1. ábra. Az említett módszer eltérő lehet, például egy küszöbérték segítségével végigvizsgálhatunk minden egyes pixelt, vagy élek detektálásával régiókra oszthatjuk a képet. A szegmentáció abban is különbözhet, hogy minden egyes objektumot meg akarunk-e különböztetni az összes többitől, vagy esetleg minket nem érdekel az egyes objektumok közti különbség, azokat kizárólag a háttértől akarjuk elválasztani. Az utóbbi eset az úgynevezett előtér-háttér szegmentáció, amelyet a dolgozatban én is alkalmaztam. Erre egy példa az 1. ábrán tekinthető meg.



1. ábra: Egy virág szegmentálása a háttértől

Jól látható, hogy ez egy bináris csoportosítási feladat, azaz egy pixel vagy háttér, vagy objektum. Formálisan, adott egy X kép $P = \{p_1, p_2, \dots, p_N\}$ pixeleinek halmaza, illetve egy $T(p_i)$ bináris módszer. Az X kép egy előtér-háttér szegmentációja (továbbiakban *bináris maszk*) alatt P következő felosztását értjük:

$$p_i = \begin{cases} háttér, ha\ T(P_i) = false\ (0) \\ objektum, ha\ T(P_i) = true\ (1) \end{cases} \quad (i = 1 \dots N) \qquad (1)$$

1.3 A DOLGOZAT FELÉPÍTÉSE

A dolgozat célja, hogy bemutassam azt az általam készített intelligens képfeldolgozó rendszert, amelynek feladata, hogy egy ismeretlen képhalmazt klaszterezzen. Ez azt jelenti, hogy a teszt képekről semmilyen előzetes információ nem áll rendelkezésre, az általam tervezett és implementált klaszterező eljárás pedig automatikusan meghatározza az optimálisnak vélt klaszterszámot, majd be is sorolja a képeket a megfelelő klaszterekbe.

A második fejezetben bemutatom, hogyan történik a képek előfeldolgozása, tehát a szegmentálás, majd a harmadik fejezetben a szegmentált képekről alkotott leírók előállításának menetét tárgyalom. Ezt követően a kidolgozott klaszterező eljárás áttekintése következik, végül pedig egy tesztelés ismeretlen képeken, melynek eredményeit kiértékelem.

2 ELŐTÉR-HÁTTÉR SZEGMENTÁCIÓ

Ebben a fejezetben egy létező eljárás lépéseit fogom ismertetni, melyet továbbfejlesztettem és integráltam a saját rendszerembe. A CALVIN kutatócsoport[1] az ImageNet [9] több millió képből álló gyűjteményét kísérelte meg feldolgozni. Ezen képeknek egy csekély része tartalmaz manuális annotációt, ezekből kiindulva, teljesen automatizált módon látták el befoglaló dobozokkal, szegmentációkkal és annotációkkal a képeket. Az általuk kidolgozott rendszert az ImageNet Auto-annotation projekt [24][23][25] keretein belül valósították meg. A projekt magában foglal olyan lépéseket is, melyeket én nem alkalmaztam, tehát csak a szükséges részeket valósítottam meg. A következő alfejezetekben lépésről lépésre bemutatom, hogyan történik egy kép szegmentálása, miközben átfogó képet adok az általam integrált alrendszer működéséről.

2.1 OPTIMÁLIS CÍMKEKÉP MEGHATÁROZÁSA

A bevezetésben már említettem, hogy egy X kép előtér-háttér szegmentációja megfelel a p_i pixelek bináris címkézésének. A pixelek c_i címkéiből előállítható egy C címkekép, amely a kép egy szegmentációjaként értelmezhető. Ezek alapján a pixelek és a hozzájuk tartozó címkék fölött egy úgynevezett *energiafüggvény* definiálható. Az optimális címkézést az energia minimalizálásával kapjuk meg[22][24][25]. A minimalizálandó energiafüggvény $E(C)$ két összetevőből épül fel:

$$E(C) = U(C) + V(C) \tag{2}$$

Itt $V(C)$ az úgynevezett simaságot, összefüggőséget méri a teljes képen úgy, hogy bünteti azokat a pixeleket, amelyeknek a szomszédjai eltérő címkével rendelkeznek. $U(C)$ pedig minden p_i pixel esetén egy meghatározott modell alapján kiszámítja, hogy az mekkora valószínűséggel veszi fel a c_i címkét, majd összegzi azokat. Ennek a modellnek a paramétereit valahogy becsülni kell, ami általában manuálisan történik (pl. interaktív felületen négyzet rajzolása az objektum köré). Az automatikus szegmentáláshoz szükség

[1] http://groups.inf.ed.ac.uk/calvin/index.html

van egy módszerre, amely megadja a paramétereket. A továbbiakban ezt a módszert fejtem ki, amely két fő lépésből áll, a befoglaló dobozok készítéséből és a folytonos maszk előállításából.

2.2 BEFOGLALÓ DOBOZOK KÉSZÍTÉSE

Első lépésként a képen szereplő objektumok *befoglaló doboz*ait kell meghatározni, amely nem más, mint egy téglalap, ami tartalmazza a kérdéses objektum egészét. Ez akkor optimális, ha az összes ilyen téglalap (továbbiakban ablak) közül ez a legszűkebb, ennek a legkisebb a területe. Az ideális az lenne, ha ilyen optimális ablakokat sikerülne meghatározni, viszont ez általában csak manuálisan lehetséges. A befoglaló dobozok számításánál feltételezzük, hogy egy objektum valamilyen önálló dolog a képen, zárt határvonallal rendelkezik vagy eltérő a háttértől, míg az behatárolhatatlan, formátlan terület. Az eljárás lényege, hogy átlapoló ablakok ezreit helyezzük el a bemeneti képen, majd minden egyes ablakhoz meghatározunk egy *objektumosság (objectness)* [2][3]pontszámot, amely négy különböző eljárás eredményének aggregáltja. A befoglaló dobozok a legnagyobb pontszámmal rendelkező ablakok lesznek, ezekre a legvalószínűbb, hogy objektumot tartalmaznak. A pontszámító algoritmusok a következők:

- *Multi-scale saliency (MS):* Az eljárás során a kép spektrális maradéka alapján egy globális eltérésvizsgálatot végzünk, amely a képen található egyedi régióknak kedvez [16]. A spektrális maradékból inverz fourier transzformáció segítségével előállítható az úgynevezett eltérés térkép (saliency map, lásd 2. ábra). Ennek segítségével az egyes objektumok egyedi karakterisztikája jól mérhető és azok egymástól elkülöníthetőek. Az eltérés térképet több léptékben számítjuk ki (multi-scale), és ezek mindegyékét önálló képként dolgozzuk fel. Az említett átlapoló ablakokat az MS módszer során kapott eltérés térkép alapján helyezzük el a képen úgy, hogy ahol az eltérés intenzitása nagyobb, az ablakok sűrűbbek lesznek azon a részen.

2. ábra: Eredeti kép (bal), eltérés térkép (jobb) (forrás: [16])

- *Color Contrast (CC):* Minden ablakot megvizsgál, hogy milyen mértékű a közvetlen környezetéhez képest a kontrasztbeli különbsége [28]. Ez jól használható, hiszen ha egy ablak objektumot tartalmaz, akkor annak környezetében húzódó háttér eltérhet az objektumtól. Ezek alapján, egy ablak minél szűkebben takar egy objektumot, annál jelentősebb lesz az eltérés, illetve annál nagyobb lesz az ablak CC pontszáma. Ezzel az eljárással nagyon pontosan meg lehet határozni a befoglaló dobozokat, viszont előfordulhat, hogy egy teljes objektum kimarad, ha a környezetének színeloszlása hasonló (pl.: kaméleon). A 3. ábrán látható egy példa a helyes működésre, mikor a vizsgált ablaknak (kék) és környezetének (sárga) nagy a kontrasztja, így a CC pontszáma is magas lesz, tehát valószínű, hogy befoglaló doboz lesz belőle.

3. ábra: CC pontszámítás: kékkel a megtalált ablak, sárgával annak környezete (forrás: [2])

- *Edge Density (ED):* Canny éldetektor [6] segítségével létrehozunk egy bináris éltérképet, majd minden ablak szélének közvetlen környezetében (míg a CC estén az ablak külső, itt belső környezetéről van szó) meghatározzuk az oda eső élpontok sűrűségét [2]. Egy élpont nem más, mint egy pixel, amelyet az éldetektor valamilyen él egy pontjaként azonosít. Ez alapján határozzuk meg az ablakok ED pontszámát, minél több élpont, annál magasabb. A 4. ábra jobb képe mutatja a legmagasabb ED pontszámú ablakot (kék) és annak belső környezetét (sárga). Látható, hogy itt a legsűrűbbek az élpontok, így a módszer szerint erre az ablakra a legvalószínűbb, hogy befoglaló doboz lesz, ami a bal képen pont a repülőgépet fedi. Az eljárás felismeri a zárt határvonalú objektumokat, viszont olykor hibásan is felismerhet egyet (false positive).

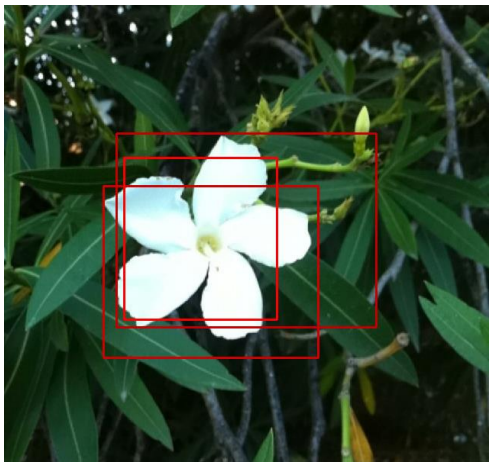
4. ábra: Eredeti kép (bal), Canny él-térkép (jobb) (forrás: [2])

- *Superpixels Straddlings (SS):* Elkészítjük a kép szuperpixeles reprezentációját, amelyet a következő módon tehetünk meg: első lépésben minden pixel egy külön csoportot alkot, majd minden további lépésben összevonjuk azokat a csoportokat, amelyek egyforma színűek, textúrájúak. Ezt egészen addig tesszük, amíg van változás egy iteráció alatt, vagy elérünk egy maximális lépésszámot. A kapott csoportok lesznek a szuperpixelek, melyek a kép egy lehetséges szegmentált felbontását adják. Példaként tekintsük az 5. ábra bal képén látható hajónak a fenti eljárással kapott szuperpixeles felbontását, amelyet a jobb oldali képen láthatunk. Fontos, hogy egy szuperpixelhez tartozó minden pixel ugyanannak az objektumnak a része (a konstrukció módjából adódóan). A SS módszer minden ablakot megvizsgál, mennyi szuperpixelt metsz [2]. Egy ablak akkor metsz egy

szuperpixelt, ha annak legalább egy-egy pixele van az ablak belső és külső környezetében. Mivel egy metszéspont adott esetben objektum vágását jelentheti, ezért az az ablak kap magasabb SS pontszámot, amelyre ez minimális. Ezek lesznek azok, amelyek legszűkebben tartalmaznak objektumot, hiszen az objektumot is szuperpixel határolja, még ha az háttér is, tehát az érintő ablak az egyetlen, amely sem az objektum, sem a háttér szuperpixelét nem metszi.

5. ábra: Egy hajó és annak szuperpixeles felbontása (forrás: [2])

A bemutatott eljárások eredményeként kapott pontszámokból minden egyes ablakra egy aggregált végső értéket kapunk naiv Bayes osztályozó segítségével [2]. Ezek közül a legnagyobb pontszámú ablakokat választjuk ki, így kapjuk a jósolt befoglaló dobozokat (lásd 6. ábra). Általában száz ilyen ablak elegendő, hogy biztosan és pontosan lefedje az összes objektumot, még bonyolult képeken is.



6. ábra: A 3 legmagasabb pontszámú ablak a képen

2.3 FOLYTONOS MASZK ELŐÁLLÍTÁSA

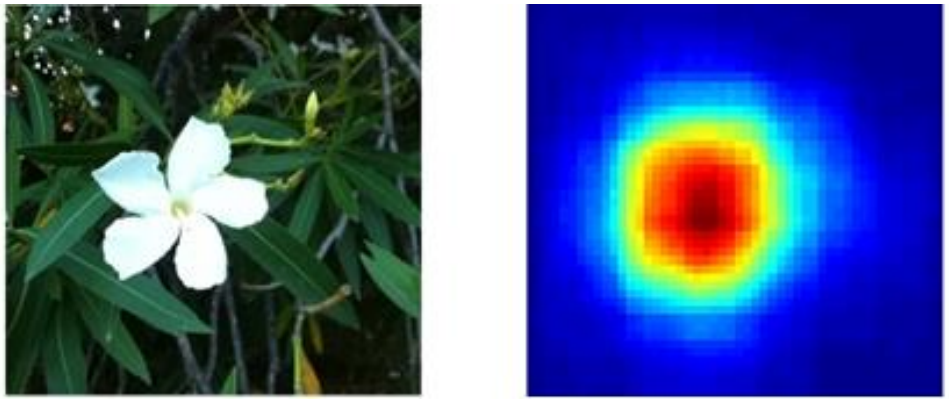
A második lépés a folytonos maszk előállítása, mely során a teszt képre számított ablakokhoz (befoglaló dobozok) egyenként meghatározzuk a hozzájuk tartalmilag legközelebbi tanító ablakokat, melyek a tanító képekre számított befoglaló dobozok összességéből kerülnek ki. Ez elárulja, hogy a módszerhez szükséges tanítási folyamat, mely során a tanító képek és a hozzájuk tartozó etalon bináris maszkok (*ground truth*) segítségével felállítunk egy úgynevezett *forráshalmaz*t, ami tartalmazza a tanító ablakokat és az ablakok bináris maszkjait. Az elképzelés lényege, hogy hasonló ablakoknak hasonló a bináris maszkja is, emellett nem szükséges, hogy a tanító kép globálisan egyezzen a teszt képpel, elég, ha van hasonló tartalmú ablakuk (továbbiakban szomszéd). Éppen ezért ez egy kritikus lépése az algoritmusnak, hiszen megfelelő szomszédok választása esetén a szegmentálás pontosan megvalósítható. A szomszédos ablakok kereséséhez, az összes tanító ablakkal össze kell hasonlítani egy adott teszt ablakot, hisztogramok [15], alakleírók és SIFT leírók [29] alapján[2]. Azért ablakok közt keressük a szomszédokat, mert globális esetben, két megegyező tartalmú kép sem biztos, hogy ugyanolyan bináris maszkkal rendelkezik. Ez lehet az eltérő nézőpont, pozíció, vagy változatos háttér miatt. Másrészt, a teszt kép olyan ablakai, melyek nem tartalmazzák az objektumot (esetleges pontatlan számítás miatt), nem rontják el a folytonos maszkot, hiszen azok szomszédjai is szintén, objektumot vélhetőleg nem tartalmazó tanító ablakok lesznek.

2.3.1 INFORMÁCIÓ ÁTVITEL

Az információ átvitel azt jelenti, hogy a teszt kép egy ablakának bináris maszkját a szomszédjainak bináris maszkjai alapján képezzük, tehát a meglévő tudást, információt kiterjesztjük, átvisszük a teszt ablakra [22]. Egy w teszt ablak M_w bináris maszkja a szomszédjaihoz tartozó M_{N_k} bináris maszkok összege, ahol k a szomszédos ablakok számát jelöli. Ehhez szükséges előbb minden M_i maszk azonos méretűvé transzformálása. Ezt a teszt képen meghatározott összes ablakra megtesszük, majd az átfedő ablakokat egy globális maszkká egyesítve a teszt kép egy úgynevezett *folytonos*

[2] A SIFT leíró kifejtésére később kerül sor.

*maszk*ját kapjuk (lásd 7. ábra), amelyet gráfvágás segítségével határozhatunk meg [14]. A kapott folytonos maszk lehetővé teszi, hogy kiszámítsuk a fent említett $U(C)$ modell paramétereit [24][25].



7. ábra: Információ terjesztéssel kapott folytonos maszk

2.3.2 SZOMSZÉDSZÁM

Fontos megemlíteni azt is, hogy érzékeny pontja a rendszernek a k paraméter (szomszédok száma) meghatározása, hiszen a szomszédok alapján hozom létre a folytonos maszkot. Kézenfekvőnek tűnhet, hogy minél több szomszédból következtessünk a teszt ablakra, viszont ez nem minden esetben szerencsés. A szomszédok meghatározásánál az eljárás egy rangsorolt listában tárolja egy teszt ablak szomszédjait, melyben a legelső elem a leghasonlóbb, majd a következő valamivel kevésbé hasonló, és így tovább. Gondoljunk bele, hogy abban az esetben, mikor túl sok szomszédot szeretnénk meghatározni egy ablakhoz, ennek a listának egy túl hosszú prefixét vesszük, és bekerülhetnek olyan képek is az információ átvitelbe, melyek nem is igazán hasonlóak (mivel a listában minden ablak fel van sorolva, a nem hasonlóak is). Ezzel akár ronthatunk is, mint javítunk, így fontos, hogy ez a paraméter megfelelően legyen megválasztva. A k-nak az implementáció során használt értékéről a dolgozat végén, a Tesztelés fejezetben lesz szó.

2.4 TANÍTÁSI FOLYAMAT KIIKTATÁSA

Az előző részekben már áttekintettem a szegmentáló alrendszer működésének főbb lépéseit, ebben az alfejezetben egy fontos tulajdonságáról ejtek szót. Felmerül a kérdés, hogy miért lesz ez jó tetszőleges bemeneti képhalmaz esetén, amelyről semmilyen

előzetes információnk nincs. Ahogy említettem, egy forráshalmazt mindenképpen fel kell építeni a működéshez. Tehát akkor mégis csak szükség van teszt halmaz specifikus tanításra? A válasz nem. A miérthez pedig tekintsünk a 8. ábra képeire. Balról az első kép egy macska befoglaló doboza, majd azt követi annak 5 legközelebbi szomszédja. Szembetűnő, hogy a legközelebbi szomszédok egyike sincs szemantikai kapcsolatban a macskát ábrázolóval. Ez azért van így, mivel a szomszédkeresés során nem szemantikus kapcsolatot keresünk, nem osztályozunk, hanem alakleírók, elrendezésbeli megegyezések alapján határozzuk meg azokat. Tehát nem túlzás azt állítani, hogy a forráshalmaz teljesen független a bemeneti képektől, hiszen bármilyen képeink is vannak, az azokon meghatározott ablakokhoz, elegendő méretű forrás esetén biztosan tudunk találni hasonló ablakokat. Ezek alapján kijelenthetem, hogy ismeretlen képhalmazhoz nem szükséges tanítási folyamat, csupán egyszer kell egy forráshalmazt felállítani, mely segítségével bármikor szegmentálhatóak tetszőleges képek.

8. ábra: A macska befoglaló doboz, és 5 legközelebbi szomszédja (forrás: [1])

2.5 MASZKOLT KÉP LÉTREHOZÁSA

A fenti eljárás eredményeként minden teszt képhez megkapom annak bináris szegmentációját. Ezek alapján képzek egy úgynevezett *maszkolt kép*et, amely nem más, mint az eredeti képből kinyert előtér, objektum (lásd 9. ábra).

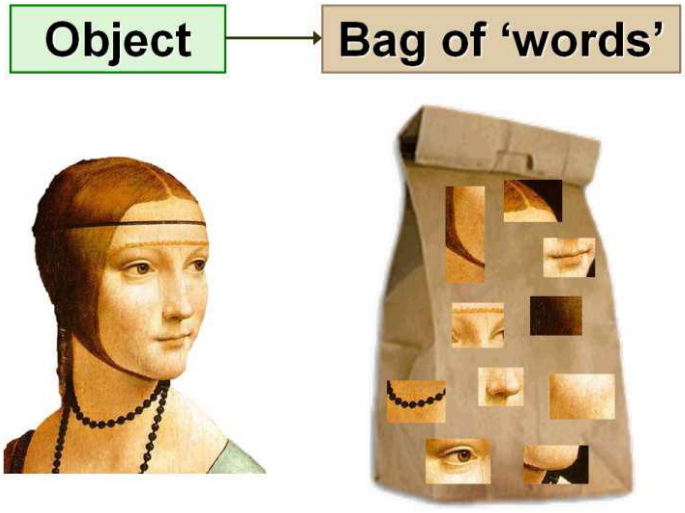
9. ábra: Maszkolt kép

3 OBJEKTUM REPREZENTÁCIÓ

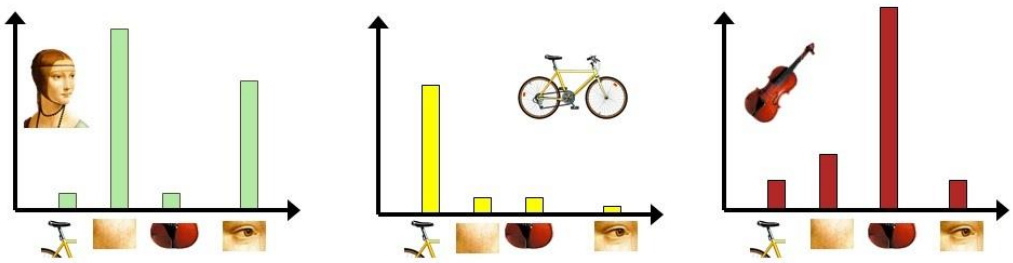
Az előző fejezetben ismertettem, hogyan készítek egy maszkolt képet az eredeti képből. A most bemutatásra kerülő alrendszer már kizárólag ezekkel a maszkolt változatokkal dolgozik tovább, és feladata, hogy a teljes bemeneti (ismeretlen) képhalmazból elkészített maszkolt képek objektumait reprezentálja. Ezek a matematikai leírók fognak alapot nyújtani a későbbi klaszterezéshez, ezért kiemelkedően fontos, hogy a lehető legpontosabban határozzam meg azokat. A következő néhány alfejezetben részletesen kifejtem, hogyan működik az algoritmus, amit erre a feladatra terveztem és implementáltam.

3.1 VIZUÁLIS KÓDSZAVAK

Az eljárásom alapját egy általános, hasonló feladatokban gyakran használt módszer képezi, ami nem más, mint a szózsák modell (*Bag of Words*) képeken alkalmazott változata [34][26]. Az elképzelés lényege, hogy egy képet a rajta szereplő vizuális elemek, más néven *vizuális kódszavak* összességével reprezentálunk, és ezek térbeli elhelyezkedésétől eltekintünk (ezt szemlélteti a 10. ábra). Tehát egy képet csupán a vizuális kódszavak eloszlásával jellemzünk, és ebből próbálunk meg következtetni a szemantikus tartalmára (lásd 11. ábra).



10. ábra: BoW modell (forrás: [26])



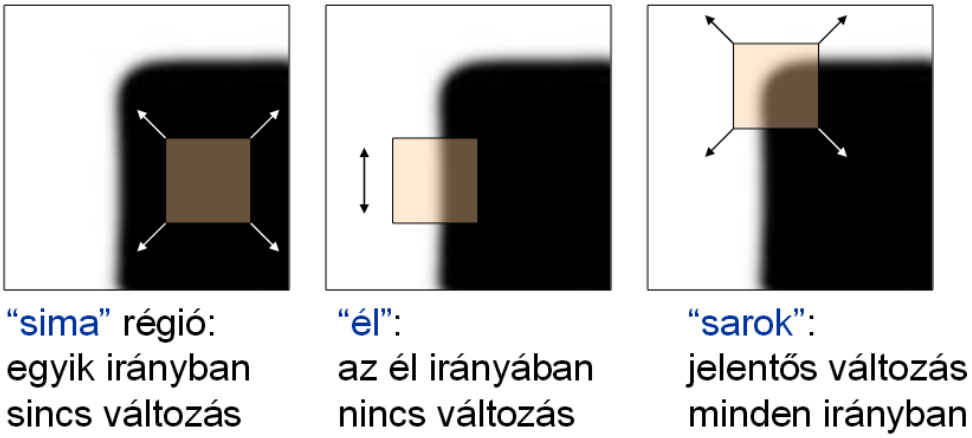
11. ábra: Vizuális kódszavak eloszlása (forrás: [26])

A vizuális kódszavak meghatározásához úgynevezett *alacsony szintű leíró*kra (röviden: leíró) van szükségünk, mely a kép egy adott pontjának környezetében előforduló lokális tulajdonságokat foglalja magában. Ezek lehetnek színek, textúrák, alakok és még sok más. Az algoritmus első feladata tehát, hogy meghatározza azokat a pontokat, melyek érdemesek leírók számítására, más néven a *kulcspontok*at.

3.1.1 KULCSPONTOK MEGHATÁROZÁSA

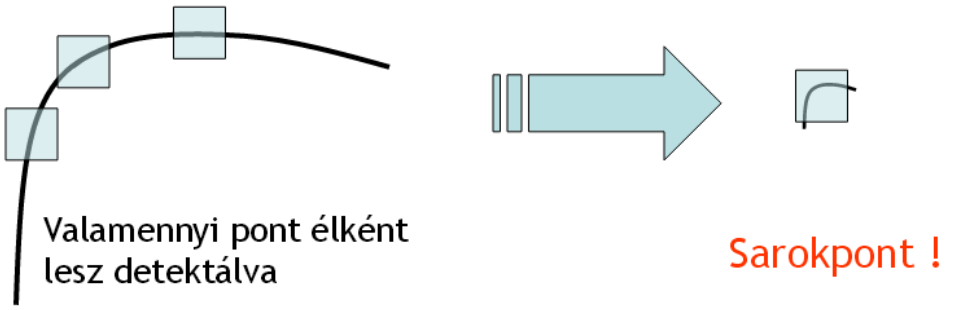
A legegyszerűbb megoldás az lenne, ha minden egyes pixelre kiszámítanám annak leíróját, viszont ez nem célra vezető, hiszen ahogy az a 9. ábrán is látszik, a háttér fehér színűre lett cserélve, így a fehérre maszkolt pixelek helyén számított leírók teljesen használhatatlanok, sőt akár még ronthatnak is a hatékonyságon. Ezért egy olyan rácsot illesztettem a képre, amely szintén az előtérként megjelölt képrészletre szorítkozik, azaz egy úgynevezett *maszkolt rács*ot. Azért rácsként említem, mert nem minden egyes pixelét veszem az előtérnek, hanem egy megadott lépésközzel haladok az egy sorban lévő pixelek között, illetve az oszlopok között is (tehát egy rácsszerű formát definiálok). Fontos megemlíteni, hogy egy kulcspontnak kiterjedése is van, tehát nem csak az adott pixelt értem alatta, hanem (ahogy említettem) annak lokális környezetét. Ez azt eredményezi, hogy egyes kulcspontok átlapolódnak, továbbá az olyan kulcspontokat, amelyek középpontja kívül esik az előtéren, viszont a környezetében megtalálható előtér pixel, azokat hozzáveszem a kulcspontok listájához. Ezzel azt érem el, hogy az objektum körvonala is biztosan jellemzője lesz az objektumot leíró vektorok összességének. Minden maszkolt képre két különböző rácsot illesztettem a fentiek alapján, rendre 6 és 18-as lépésközökkel.

A fenti kulcspont meghatározó módszeren kívül, még a Harris-Laplace detektort is használtam, melynek alapja a Harris detektor, vagy más néven kombinált sarok és él detektálás [5][21]. Az eljárás lényege, hogy egy képen azonosítja az éleket, ahol a kép pontjainak intenzitásértékeiben nagy változás következik be. Két él találkozásánál (sarokpontnál) pedig az intenzitásváltozás, azaz a derivált két irányban is nagy abszolút értékű lesz. Ennek meghatározására egy csúszó ablakot vizsgál, ezt szemlélteti a 12. ábra.



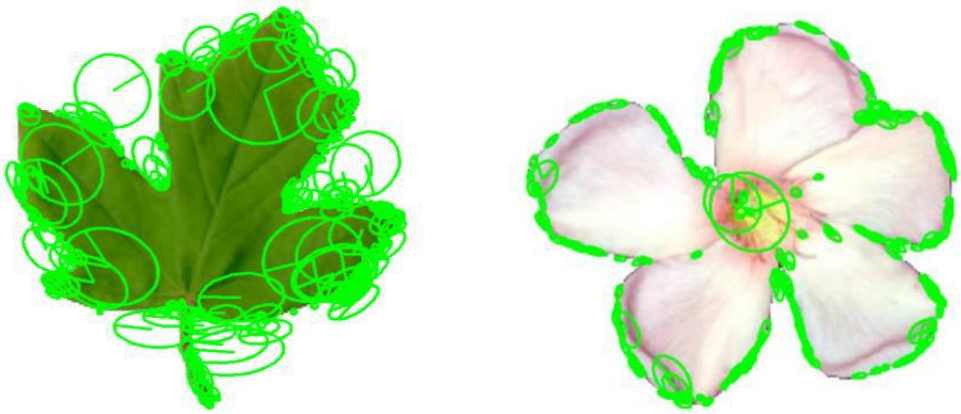
12. ábra: Sarokpont detektálás csúszó ablakot vizsgálva (forrás: [21])

Mivel minden számításhoz deriváltakat használ, ezért azok invariánsak az eltolásra és az intenzitás változására, viszont csak részben invariánsak a skálázásra. A 13. ábra bal oldalán látható, hogy egy vonal mentén éleket detektál az algoritmus, viszont egy másik skálázásban ugyanaz a vonal sarokként lesz detektálva, mivel a vizsgált ablakba belefér az egész.



13. ábra: Harris sarokdetektor nem invariáns a skálázásra (forrás: [21])

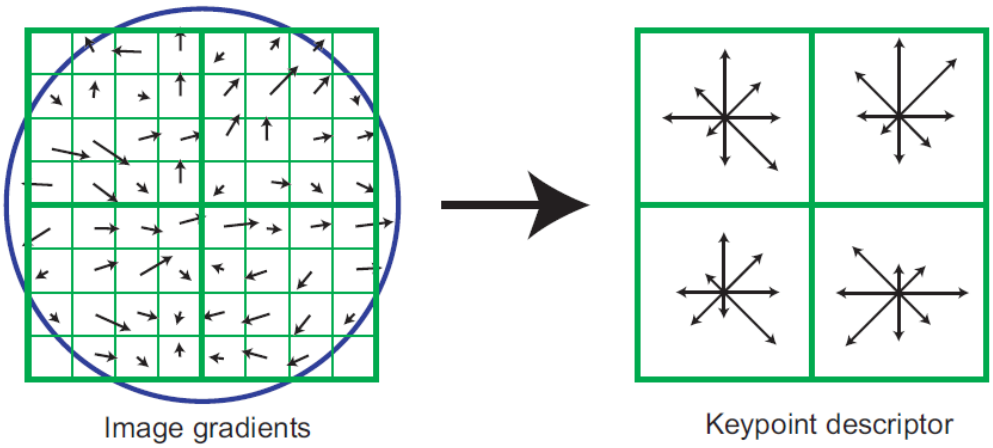
A Harris detektort K. Mikolajczyk és C. Schmid fejlesztette tovább, úgy, hogy az invariáns legyen a skálázásra is, ezt nevezik Harris-Laplace detektornak [31]. Minden maszkolt képre lefuttatom a detektort (lásd 14. ábra), majd a kapott kulcspontokat hozzácsatolom az eddig készített listához, melyet az általam tervezett kulcspont meghatározó módszerrel kaptam. Ezzel minden kép mellé elkészül egy lista, amely tartalmazza azokat a fontos pontokat, ahol leíró vektort akarok számítani.



14. ábra: Harris-Laplace detektorral kapott kulcspontok

3.1.2 A SIFT LEÍRÓ

A második főbb lépés, hogy minden kulcspontot SIFT (Scale Invariant Feature Transform) leíróval jellemzek, melyet David G. Lowe fejlesztett ki [29]. Ez tulajdonképpen egy 128 dimenziós vektor, amely egy pont lokális környezetében lévő gradiensek orientációjából számolható. A 15. ábra segítségével részletesen bemutatom a SIFT működését, melyet a szegmentáló alrendszerben, a szomszédok hasonlóságának mérése során is használtam.



15. ábra: SIFT leíró működése (forrás: [29])

Elsőnek a gradiens irányát és nagyságát számoljuk ki minden egyes mintavételi pontban. A mintavételi pontok meghatározásához $n \times n$ cellát jelölünk ki egy fontos pont körül. Minden egyes cellát további $k \times k$ részre osztunk. Ez alkotja a mintavételi pont körüli vizsgált környezetet, ahogy az az ábra bal oldalán is látható. A számított gradiensek nagyságát egy Gauss-függvény szerint súlyozzuk, majd a súlyozott gradiensekből cellánként készítünk egy-egy orientáció szöghisztogramot, d irányra. Ennek eredménye látható az ábra jobb oldalán. Az ábrán $n = 2, k = 4, d = 8$. A SIFT standard paraméterezése a következő: $n = 4, k = 4, d = 8$. Végül a hisztogramok egymásutánjából kapunk egy $n^2 \times d$ dimenziós vektort. Ez az alapbeállítások mellett 128 dimenziót jelent, ahogy én is használtam.

A módszer invariáns a fényviszonyokra, amit a vektor normalizálásával érhetünk el, mivel ha egy kép pixeleit szorozzuk egy konstanssal, a gradiensek nagyságában fellépő változást a normalizálás semlegesíti. A fényerő változások esetén egy konstanst adunk a pixelekhez, viszont ez nem befolyásolja a gradienseket, mivel azt a pixelek különbségeiből számítjuk. A nem-lineáris megvilágítási változásokkal már nagyobb probléma van, mivel nem lehet egységesen kezelni a gradiensek változását. Ezek nagy különbségeket tudnak okozni a gradiensek nagyságában, viszont az irányukat nem változtatják. Tehát ennek a befolyását úgy tudjuk csökkenteni, ha meghatározunk egy küszöb értéket, amellyel korrigáljuk a gradiensek nagyságát. Ennek hatására a gradiensek nagysága nem kap akkora szerepet, mint az orientációja. Ehhez a leíró vektor

minden 0,2-nél nagyobb elemét 0,2-re cseréljük, majd újra normalizálunk. Ezt a 0,2-es értéket Lowe javasolta [29], miután kikísérletezte azt ugyanazon képek megvilágításbeli változtatgatásával.

A fent ismertetett eljárással tehát minden maszkolt kép minden kulcspontjában kiszámítom a SIFT leírókat. Az egymáshoz hasonló leírók jelentenek egy vizuális kódszót. Ezek meghatározásához klaszterezem az összes képből kinyert SIFT leírót a GMM (Gaussian Mixture Model) módszer segítségével [33].

3.1.3 GAUSSIAN MIXTURE MODEL

A GMM egy generatív módszer az alacsony leírók klaszterezésére, tehát az egyes klaszterekhez egy-egy valószínűségi modellt rendel [33]. Ebben az esetben egy pont tartozhat több klaszterbe is, bizonyos valószínűséggel. A pontok valószínűségi sűrűségfüggvényét K darab Gauss-féle függvény súlyozott összegével írja le:

$$p(X|\lambda) = \sum_{j=1}^{K} \omega_j g\left(X\middle|\mu_j, \sigma_j\right) \quad (3)$$

Itt az $X = \{x_1, x_2, \dots, x_N\}$ jelöli az M dimenziós adatvektorokat (SIFT leírók), λ a keresendő paramétert, $g(X|\mu_j, \sigma_j)$ a Gauss-féle függvényeket (ahol μ_j a várható értéket, σ_j a szórást jelöli), valamint ω_j pedig a súlyozó együtthatókat, amelyekre a következő megkötésnek kell teljesülnie:

$$\sum_{j=1}^{K} \omega_j = 1 \quad (4)$$

A GMM λ paraméterét ML (Maximum Likelihood) becsléssel határozzuk meg. Ennek célja egy olyan λ paraméter jóslása, amely maximalizálja a valószínűségét annak, hogy a megadott ponthalmaz keletkezett. Az x_i adatvektorok közt függetlenséget feltételezve a 3. egyenlet a következő módon alakul:

$$p(X|\lambda) = \prod_{i=1}^{N} p(x_i|\lambda) \quad (5)$$

Ez a kifejezés nem-lineáris a λ paraméterre nézve, tehát a direkt maximalizálása nem lehetséges. Erre egy speciális iteratív módszert szokás használni, az EM (Expectation Maximization) algoritmust [7][8]. Ennek lényege, hogy induljunk ki egy kezdeti λ paraméterből, majd ebből becsüljünk minden lépésben egy újat (λ'), amire $p(X|\lambda') \geq p(X|\lambda)$ teljesül. Ez után az új paraméter lesz a következő iteráció kezdeti paramétere, és

így tovább. Egészen addig folytatjuk az iterációt, amíg konvergenciát nem tapasztalunk vagy elértük a maximálisan megengedett iterációk számát. A kezdeti paraméter meghatározásához pedig K-means[3] klaszterezést használtam [35].

A GMM eredményeként megkapom a vizuális kódszavakat (az egyes Gauss függvények), melyekre tekinthetek úgy, mint a teszt képhalmaz tömör reprezentálására. A célom az, hogy a hasonló objektumokat összerendeljem, klaszterezzem, tehát mindenképpen szükség van egy olyan magasabb szintű képi leíró megalkotására, amely nem csak egy kulcspontot jellemez, hanem egy egész objektumot. Erre a feladatra a Fisher-vektort használom.

3.2 FISHER-VEKTOR

A Fisher-vektor [19] a képfeldolgozási témakörök közül a képosztályozásban a legelterjedtebb [13], viszont az én algoritmusomba tökéletesen beleillik. Ez egy rendkívül komplex leíró, amely arra használható, hogy egy teljes képet jellemezzünk vele egyetlen vektor formájában. Esetemben ezek a maszkolt képek lesznek, vagyis az eredeti képek előterei, objektumai. Kiszámításához szükség van a SIFT leírókra, illetve a vizuális kódszavakra, tehát a GMM-re. Valójában azt fogja ez megmondani, hogy egy előtér objektum milyen eloszlásban tartalmaz vizuális elemeket, ahol ezek a vizuális elemek a teszt képhalmaz egészéből kerülnek ki. Ez tehát egy egységes és egyértelmű reprezentációja lesz az objektumoknak. A következőkben pontosan megadom, hogyan épül fel egy ilyen Fisher-vektor.

A GMM tárgyalásánál bevezetett jelöléseknek megfelelően legyen $p(X|\lambda)$ a valószínűségi sűrűségfüggvény, amelynek λ a paramétere ($\lambda = \{\omega_j, \mu_j, \sigma_j | j = 1 \dots K\}$), legyenek $X = \{x_1, x_2, \dots, x_N\}$ az M dimenziós adatvektorok, melyek itt nem az összes SIFT leírót jelentik, csak azokat, melyek egy adott objektumra vonatkoznak. A sűrűségfüggvény gradiense, azaz a logaritmusának deriváltja megadja, hogyan írja le a legjobban a modell az adatvektorokat, tehát az objektumot, így tulajdonképpen ez a mennyiség a Fisher-vektor:

$$\nabla_\lambda \log p(X|\lambda) \tag{6}$$

[3] A K-means klaszterezés működését később fejtem ki.

A kiszámítását a következő néhány lépésben mutatom be. A továbbiakban vezessük be az $L(X|\lambda) = \log p(X|\lambda)$ jelölést. Amennyiben feltesszük, hogy az adatvektorok egymástól kölcsönösen függetlenek, akkor $L(X|\lambda)$ felírható a következőképpen:

$$L(X|\lambda) = \sum_{i=1}^{N} \log p(x_i|\lambda) \tag{7}$$

Ez azért lesz igaz, mert a logaritmusok összege a szorzatok logaritmusával egyenlő, tehát az 5. egyenletben használt produktum felcserélhető summára. Annak a valószínűsége, hogy az x_i adatvektort a GMM generálta a következő:

$$p(x_i|\lambda) = \sum_{j=1}^{K} \omega_j g(x_i|\mu_j, \sigma_j) \tag{8}$$

Továbbá jelölje $\gamma_i(j)$ annak a valószínűségét, hogy az x_i adatvektort a *j*-edik Gauss-függvény generálta:

$$\gamma_i(j) = p(j|x_i, \lambda) = \frac{\omega_j p_j(x_i|\lambda)}{\sum_{jj=1}^{K} \omega_{jj} p_{jj}(x_i|\lambda)} \tag{9}$$

A súlyozási együtthatókra pedig most is érvényes a 4. egyenlet által leírt megkötés. Ezzel felbontottuk az $L(X|\lambda)$-et és kifejeztük a Gauss-függvények súlya, várható értéke és szórása alapján, melyek értékét tudjuk a GMM-ből. Az utolsó lépés, hogy deriváljuk az $L(X|\lambda)$ függvényt, méghozzá rendre a λ paraméter elemei szerint egyenként. A következő egyenletekben megadom, hogyan írhatóak fel az egyes deriváltak:

$$\frac{\partial L(X|\lambda)}{\partial \omega_j} = \sum_{i=1}^{N} \left[\frac{\gamma_i(j)}{\omega_j} - \frac{\gamma_i(1)}{\omega_1} \right] \tag{10}$$

$$\frac{\partial L(X|\lambda)}{\partial \mu_j^m} = \sum_{i=1}^{N} \gamma_i(j) \left[\frac{x_i^m - \mu_j^m}{(\sigma_j^m)^2} \right] \tag{11}$$

$$\frac{\partial L(X|\lambda)}{\partial \sigma_j^m} = \sum_{i=1}^{N} \gamma_i(j) \left[\frac{(x_i^m - \mu_j^m)^2}{(\sigma_j^m)^3} - \frac{1}{\sigma_j^m} \right] \tag{12}$$

Itt $j = 1, 2, \dots, K$ jelöli a megfelelő Gauss-függvényt, és $m = 1, 2, \dots, M$ jelöli az adott dimenziót. Ezek után a gradiens vektorok dimenziónkénti eltérő nagysága miatt még egy további normalizálás [13][20] is szükséges, melynek menetét nem részletezem. A Fisher-vektor tehát a következők szerint alakul ki:

- Az $L(X|\lambda)$-et deriváljuk egyenként a K darab Gauss-függvény súlyai szerint. Mivel a 4. egyenlet miatt, például ω_1 kiszámítható a többi súlyozási együttható ismeretében, ezért csak $K-1$ szabad paraméter van. Tehát ez $K-1$ elemet ad a Fisher-vektorba.
- Az $L(X|\lambda)$-et deriváljuk egyenként a K darab Gauss-függvény várható értékei szerint. Mivel ezek M dimenziósak, ezért ez $K \times M$ elemet fog adni a Fisher-vektorba.
- Az $L(X|\lambda)$-et deriváljuk egyenként a K darab Gauss-függvény szórásai szerint. Ez szintén $K \times M$ elem.

Ebből következik, hogy a Fisher-vektor összesen $K(2M+1)-1$ dimenziós. Az én implementációmban $K = 256$, mivel ennyi Gauss-függvényt definiálok (tesztek után ez bizonyult átlagosan a legjobbnak), $M = 128$, mivel a SIFT leírók 128 dimenziósak. Ez azt jelenti, hogy $K(2M+1)-1 = 65791$ az én esetemben, tehát egy Fisher-vektor 65791 dimenziós. Ez tehát egyértelműen meghatároz egy adott objektumot, így a továbbiakban ezeket a rendkívül magas dimenziójú vektorokat fogom klaszterezni. A kapott klaszterek lesznek az összerendelések, és a Fisher-vektorokat visszahelyettesítve az objektumokkal megkapom az egymáshoz hasonló objektumokat csoportokba rendezve.

4 KLASZTEREZÉS

A korábban leírtaknak megfelelően, a rendszer futásának ezen a pontján rendelkezésre állnak a Fisher-vektorok, melyeket a reprezentáló alrendszerem készít el a létrehozott maszkolt képekből. A Fisher-vektorok lesznek az adatpontok, melyeket klaszterezni fogok. A tervezés során rengeteg féle klaszterező eljárással találkoztam, mint például a sűrűség alapú DBSCAN [10], vagy a modell alapú GMM [33][7], de léteznek rács alapú, illetve hierarchikus módszerek is. A lényege természetesen mindegyiknek ugyanaz, vagyis az adatpontokon hasonlósági adatcsoportok definiálása úgy, hogy a hasonló adatpontok azonos, a különbözőek pedig különböző csoportba kerüljenek. A nehézség az említett hasonlóság mérése, melyre több különböző algoritmus létezik.

4.1 K-MEANS

Az általam alapul választott klaszterező eljárás a K-means [35][27][18], melyet az objektum reprezentációnál a GMM kezdeti paramétereinek meghatározására is használtam. Jelen esetben az eljárást úgy használtam, hogy az két adatpont közti hasonlóságot azok Euklideszi távolságával mérjen. Ezek alapján partíciókra osztom a teljes mintahalmazt, majd a partíciókat iteratívan frissítem. Legyenek adottak az $X = \{x_1, x_2, \dots, x_n\}$ adatpontok (a Fisher-vektorok), és a K pozitív egész szám. Az algoritmus menete a következő:

1. Válasszunk K darab kezdeti klaszterközéppontot $(z_1, z_2, \dots, z_k)$, egyenletes eloszlással.
2. Az m-edik iteratív lépésben osszuk el az $\{x_i\}$ adatvektorokat a klaszterközéppontok között a következő reláció szerint:

$$x_i \in C_l(m), ha\ \forall j - re\colon \|x_i - z_l(m)\| < \left\|x_i - z_j(m)\right\| \qquad (13)$$

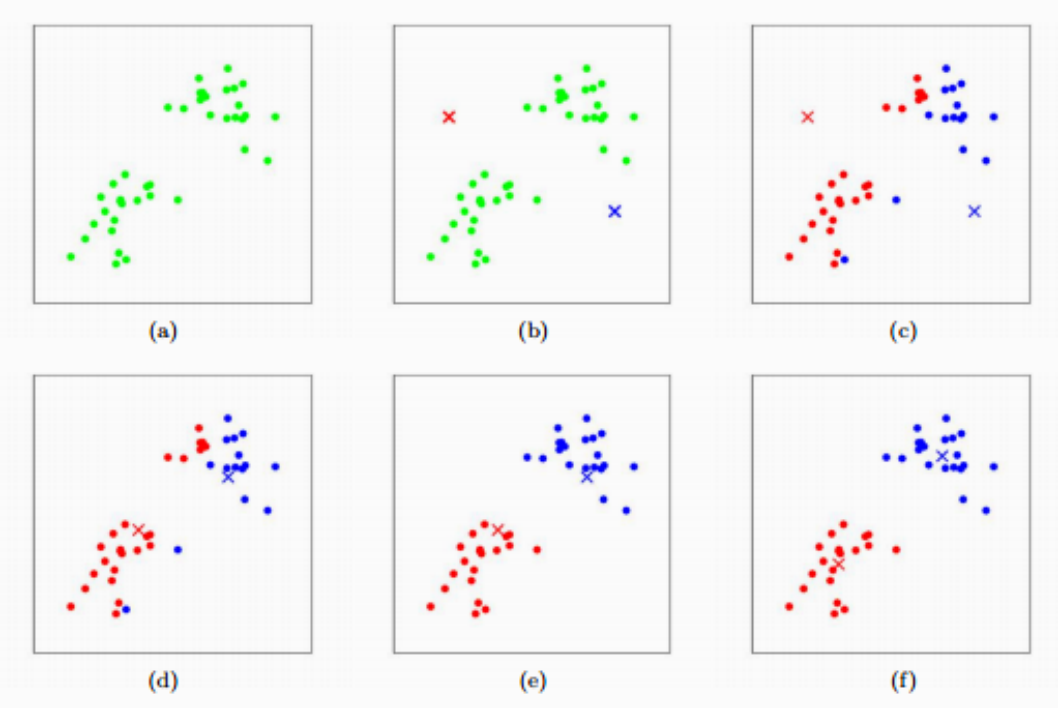
ahol $i = 1 \dots n;\ j, l = 1 \dots K;\ j \neq l$, illetve $C_l(m)$ jelöli azon adatvektorok halmazát, melyek klaszterközéppontja $z_l(m)$.

3. Ezt követően kiszámítjuk az új klaszterközéppontokat: $z_l(m+1), l = 1 \dots K$ úgy, hogy a $C_l(m)$ által jelölt adatvektorok és a hozzájuk rendelt új $z_l(m+1)$ középpont közti távolságok négyzetösszege minimális legyen. Ez pont akkor érhető el, ha az új középpontot a $C_l(m)$ adatvektorainak átlagából képezzük:

$$z_l(m+1) = \frac{1}{N_l}\sum_{x_i \in C_l(m)} x_i\,, \quad i = 1 \dots n, \quad l = 1 \dots K \tag{14}$$

   ahol N_l az adatvektorok száma a $C_l(m)$ halmazban.

4. Amennyiben $z_l(m+1) = z_l(m)$, $l = 1 \dots K$, akkor az algoritmus konvergáló folyamata befejeződött[4] és a végső klaszterközéppontok az utolsó iterációban rendelkezésre állnak, a klaszterek pedig a hozzájuk rendelt halmazok lesznek. Egyéb esetben visszalépünk a 2. pontra.



16. ábra: K-means lépései

Az eljárás lépéseit a 16. ábra szemlélteti, ahol az (a) képen látható a kiindulási ponthalmaz, majd a (b) képen két klaszterközéppontot határozunk meg ($K = 2$), ez felel meg az algoritmus 1. lépésének $(z_1(1), z_2(1))$. Ez után, a (c) képen a

[4] Előfordulhat, hogy a konvergencia túl sok iteráció után áll be, ilyenkor szabályozhatjuk azokat egy limit megadásával.

klaszterközéppontok alapján kijelöljük az egyes klasztereket $(C_1(1), C_2(1))$, 2. lépés. A (d) képen látható a középpontok frissítése $(z_1(2), z_2(2))$, 3. lépés. Ezek nem egyeznek az előző középpontokkal, ezért vissza a 2. lépésre. Ezt mutatja az (e) kép, újabb elosztását a pontoknak $(C_1(2), C_2(2))$, végül (f) egy ismételt középpontfrissítést szemléltet $(z_1(3), z_2(3))$. Ezt addig folytatjuk, amíg már nem tapasztalunk eltérést, vagy el nem érjük a megadott iterációs limitet.

A fenti leírásból jól látszik, hogy a végső klaszterezés erősen függ a kezdetben meghatározott klaszterközéppontoktól, illetve a K számtól. Ez utóbbi különösen érzékeny paraméter, hiszen ahhoz, hogy ennek pontos értékét át tudjuk adni az algoritmusnak, mindenképpen szükség van a mintahalmaz előzetes ismeretére, erre pedig valós problémák esetén kevés esély van. Ezek szerint, amennyiben ismeretlen képhalmazzal van dolgunk, valahogy meg kell becsülni K lehetséges értékeit. Erre a legegyszerűbb megoldás, hogy a felhasználótól bekérünk egy minimális és egy maximális K-t, majd e kettő közt futtatjuk a klaszterezést és kiválasztjuk a „legjobbat”. A következő részben ennek menetét fejtem ki.

4.2 MIN-MAX K-MEANS

A K-means módszerben arra törekszünk, hogy minimalizáljuk minden adatvektor és annak klaszterközéppontja közti távolságok négyzetösszegét. Ez azt jelenti, hogy tömör, kompakt klasztereket kell kapnunk, így használhatjuk az adatvektorok és a klaszterközéppontjaik távolságát egy adott klaszter kompaktságának mérésére. Erre a célra az úgynevezett *klaszteren belüli távolság* mérőszámot (*intra-cluster distance, [38]*) használom, ami nem más, mint az átlaga ezeknek a távolságoknak:

$$intra = \frac{1}{n}\sum_{l=1}^{K}\sum_{x_i \in C_l}\|x_i - z_l\|^2 \qquad (15)$$

ahol n az adatvektorok száma, K a klaszterek száma, és z_l pedig a C_l klaszter középpontja. Tehát a 15. egyenletben lévő kifejezést szeretném minimalizálni. Definiálható továbbá egy úgynevezett *klaszterek közti távolság* (*inter-cluster distance, [38]*) is, mely az egyes klaszterközéppontok páronkénti távolságát jelöli. Minden páronkénti távolságot külön kezelni nem szükséges, elég, ha legkisebbet szeretném a

lehető legnagyobbra növelni. Tehát elég a legkisebb ilyen klaszterek közti távolságot kiválasztanom, melyet a következőképpen határozok meg:

$$inter = \min(\|z_l - z_j\|^2) \tag{16}$$

ahol $l = 1 \dots K-1;\ \ j = l+1 \dots K$. A fenti két mérőszám együtt fog segíteni abban, hogy meghatározzam egy klaszterezés „jóságát” (*validity*), így ezek hányadosát veszem:

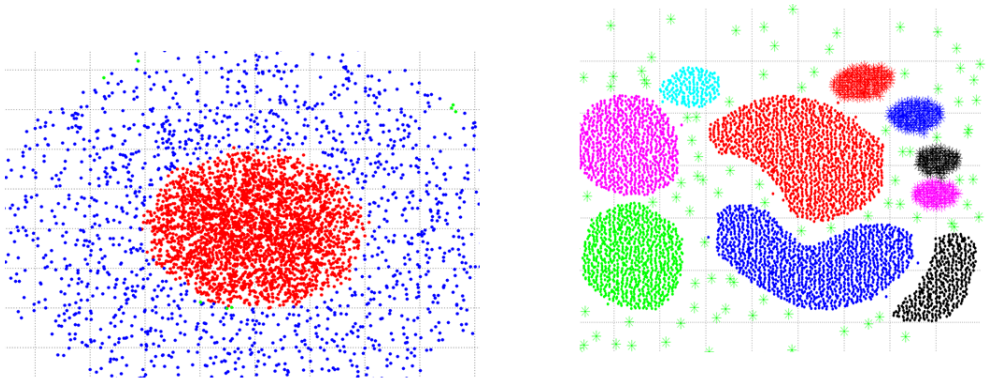
$$validity = \frac{intra}{inter} \tag{17}$$

Mivel az $intra$-t minimalizálni szeretném, ezért az kerül a számlálóba, a maximalizálandó $inter$ pedig a nevezőbe, ebből az is kiderül, hogy a $validity$ mérőszámot is minimalizálni szeretném. Ennek segítségével egy olyan eljárást hoztam létre, amellyel K alsó és felső határértéke között ki tudom választani az optimálist. Elindulok K alsó értékétől, és futtatom a klaszterezést, majd mindig eggyel növelem K-t, és újrafuttatom a klaszterezést. Minden iterációban csak a K változik, és kiszámítom a $validity$ mérőszámot. Az utolsó iteráció után az eltárolt $validity$ értékek közül a legkisebbhez tartozó K szám lesz az, amely a legmegfelelőbb az $\{x_i\}$ adatvektorok klaszterezéséhez. Az említett alsó és felső korlát változó lehet, amennyiben az összes lehetőséget számba akarjuk venni, akkor $K \in \{2, n-1\}$. Erre persze ritkán van szükség, illetve a futási időt is jelentősen növeli.

A min-max K-means eljárás tökéletesen használható 2,3 vagy akár több dimenziós vektorok esetén is, viszont jelen esetben a Fisher-vektorok 65791 dimenziósak. Sok próbálkozás után kiderült, hogy a K-means nem elég hatékony, hogy megoldja ezt a feladatot, így újabb módszer után, továbbfejlesztések után kezdtem kutatni, hogy javítani tudjak az eredményeken. A következő alfejezetben ezt a továbbfejlesztést mutatom be.

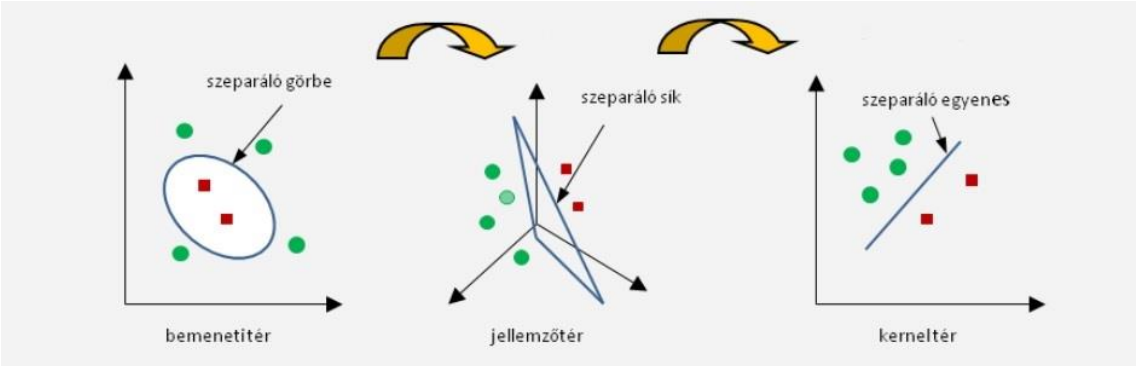
4.3 KERNEL K-MEANS

A K-means jól működik olyan esetekben, mikor a klaszterek lineárisan szeparálhatóak, illetve tömör, jól elkülöníthető klasztereink vannak. Ellenben, lineárisan nem szeparálható klaszterek esetén, vagy ha az adatvektorok egyedi „formájú” klaszterekből származnak, akkor a K-means már kevésbé hatékony, esetleg használhatatlan. Ezeket a példákat szemlélteti a 17. ábra.



17. ábra: K-means nehézségei (forrás: [32])

A fenti ábrán különböző színekkel vannak megjelenítve a valós klaszterek, viszont ezeket a K-means használatával nem tudjuk reprodukálni. Ilyen esetben célszerű az úgynevezett *kernel-trükk* használata, mely segítségével egy (elképzelt) jellemző térben megoldható a klaszterezés. A 18. ábrán látható, hogy a piros illetve zöld $\{x_i\}$ mintapontok nem szeparálhatóak lineárisan, viszont egy $x_i \to \vartheta(x_i)$ nemlineáris dimenziónövelő transzformációval olyan jellemző teret (18. ábra középső képe) kapunk melyben már a feladat megoldható lineárisan. A jellemző térben való számítások nem triviálisak, ezért használjuk a kernel teret. A kettő közti kapcsolatot a skaláris szorzás teremti meg: $Ker(x_i, x_j) = \vartheta_i \times \vartheta_j$, ahol Ker egy alkalmasan választott kernel függvény. Látható, hogy a kernel térben (18. ábra jobb képe) a mintapontok lineárisan szeparálhatóakká váltak, miközben a jellemzőtérbeli számításokat kiküszöböltük, mivel a kernel függvény argumentumai a bementi tér mintapontjai (ezt hívjuk kernel-trükknek). Fontos megjegyezni, hogy itt a bemeneti teret a Fisher-vektorok alkotják, a jellemző tér pedig egy implicit módon definiált tér, melyet a kernel-trükk használatával átugrunk.



18. ábra: Transzformáció a bemeneti tértől a kernel térig (forrás: [4])

A kernel függvények, más néven magfüggvények az adatvektorok közti hasonlóságot mérik. Kernel függvényként csak olyan függvény használható, amely belső szorzatból származtatható, valamint a következő tulajdonságokkal rendelkezik:

- a függvény értéke nem lehet negatív és radiálisan szimmetrikus kell, hogy legyen,
- azonos argumentumok esetén az értéke maximális,
- két argumentum távolságának monoton csökkenő függvénye.

A következő táblázatban összefoglaltam a leggyakrabban használt kernel függvényeket. Ezek közül én az RBF változatot használtam a klaszterező algoritmusban.

Lineáris	$Ker(\vartheta_i, \vartheta_j) = \vartheta_i \times \vartheta_j$
Polinomiális (d fokszámú)	$Ker(\vartheta_i, \vartheta_j) = (\vartheta_i \times \vartheta_j + 1)^d$
Gauss-féle (radiális bázisfüggvények, RBF)	$Ker(\vartheta_i, \vartheta_j) = e^{-\frac{\|\vartheta_i - \vartheta_j\|^2}{2\sigma^2}}$
Tangens hiperbolikusz (k és θ konstansok)	$Ker(\vartheta_i, \vartheta_j) = tanh(k(\vartheta_i \times \vartheta_j) + \theta)$

1. táblázat: Leggyakoribb kernel függvények

Ahhoz, hogy az előbb leírtakat be tudjam építeni az algoritmusba, a K-means lépéseiben alkalmaznom kell a jellemző térbe képzést. Továbbra is legyenek adottak az $X = \{x_1, x_2, \dots, x_n\}$ adatvektorok és a K pozitív egész szám. Ahogy már említettem, K-means esetén a cél az, hogy minimalizáljuk az adatvektorok és klaszterközéppontjaik közti távolságok négyzetösszegét:

$$\min\left(\sum_{l=1}^{K} \sum_{x_i \in C_l} \|x_i - z_l\|^2\right) \quad (18)$$

ahol K a klaszterek száma, C_l jelöli az l-edik klaszterbe tartozó adatvektorokat, melynek középpontja z_l. Kernel K-means esetén ezt a minimalizálási problémát a jellemző térben szeretnénk megvalósítani [32][30], tehát használjuk a $x_i \to \vartheta(x_i)$ jellemző térbe transzformálást:

$$\min\left(\sum_{l=1}^{K} \sum_{x_i \in C_l} \|\vartheta(x_i) - z_l\|^2\right) \quad (19)$$

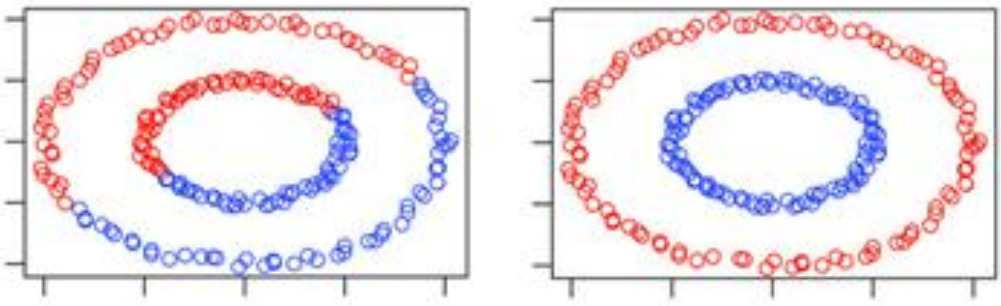
Mivel z_l itt is a klaszterközéppont, ezért behelyettesíthetjük a 19. egyenletbe és a következőt kapjuk:

$$\min\left(\sum_{l=1}^{K}\sum_{x_i\in C_l}\left\|\vartheta(x_i)-\frac{\sum_{x_j\in C_l}\vartheta(x_j)}{N_l}\right\|^2\right) \tag{20}$$

ahol N_l a C_l klaszter számosságát jelöli. A két tag négyzetét kifejtve a következőt kapjuk:

$$\vartheta(x_i)\times\vartheta(x_i)-\frac{2\times\sum_{x_j\in C_l}\vartheta(x_j)\times\vartheta(x_i)}{N_l}+\frac{\sum_{x_j,x_k\in C_l}\vartheta(x_j)\times\vartheta(x_k)}{{N_l}^2} \tag{21}$$

Ezen a ponton látszik, hol tudjuk alkalmazni a kernel trükköt. A kifejtést követően a jellemző térbeli skaláris szorzatok felválthatóak a megfelelő, bemeneti térbeli adatvektorok kernel függvényeivel ($Ker(x_i,x_j)=\vartheta_i\times\vartheta_j$)[5]. Ezt követően a minimalizálási feladat már megoldható, ennek menetét a dolgozatban nem fejtem ki, megtekinthető a megjelölt írásokban [32][30][17]. A 19. ábra szemlélteti, hogy az előbb ismertetett módszer mennyivel hatékonyabb az eredeti K-means módszernél, ha a klasztereink lineárisan nem szeparálhatóak. A bal képen a K-means eredménye látható a jobb képen pedig a Kernel K-means eredménye. Szembetűnő, hogy még ebben a nem lineáris esetben is milyen tökéletesen szétválasztotta az adatpontokat. Nekem, a Fisher-vektorok klaszterezéséhez pontosan erre van szükségem.



19. ábra: K-means és Kernel K-means ugyanazon pontokon (forrás: [32])

[5] A ϑ_i *és* $\vartheta(x_i)$ jelölést ekvivalensen használom.

4.4 MIN-MAX KERNEL K-MEANS

A 4.2-es alfejezetben már említettem, hogy K-means alapú módszerrel kapott végső klaszterezés függ a K klaszterszámtól, és a kezdetben meghatározott klaszterközéppontoktól. Az optimális K meghatározásához az előző alfejezetben tárgyalt Kernel K-means eljárást építettem be a 4.3-ban bemutatott K paramétert kereső módszerbe. Amint ez megvan, megpróbálom megkeresni azt a kezdeti klaszterközéppont disztribúciót, amely optimális az adatvektorok klaszterezéséhez. Ehhez egymás után többször is futtatom a klaszterezést, és kiválasztom a „legjobb" eredményt. Itt nem a K érték megfelelőségét értem a jóság alatt, hanem az egész klaszterezését, az adott K mellet. A továbbiakban jelölje $E(X)$ a következőt:

$$E(X) = \min\left(\sum_{l=1}^{K}\sum_{x_i\in C_l}\left\|\vartheta(x_i) - \frac{\sum_{x_i\in C_l}\vartheta(x_i)}{N_l}\right\|^2\right) \tag{22}$$

ami az $X = \{x_1, x_2, \dots, x_n\}$ adatvektorok klaszterezése során kapott távolságok négyzetösszege a minimalizálás után. Minden egyes klaszterezés után megkapom annak $E(X)$ értékét, és ezek közül fogom a minimálist választani, így határozom meg az előbb említett legjobb klaszterezést.

Megterveztem, és kifejtettem, hogyan fognak a klaszterező eljárás egyes részfeladatai működni, most összefoglalom, hogyan működik pontosan a teljes algoritmus. Legyenek továbbra is adottak az $X = \{x_1, x_2, \dots, x_n\}$ adatvektorok, a $minK, maxK$ pozitív egész számok ($maxK > minK$), és az m, l pozitív egész számok:

1. Az m-edik iterációs lépésben a következőt hajtom végre:
    a. Minden $K' = minK, minK+1, \dots, maxK-1, maxK$ értékre:
        i. Futtatom a Kernel K-means klaszterezést az $\{x_i\}$ adatvektorokon K' klaszterszámmal
        ii. A kapott klaszterezésre kiszámítom a $validity$ mérőszámot, és eltárolom, a hozzá tartozó K'-t
    b. Megkeresem a minimális $validity$-hez tartozó K'-t, és eltárolom egy M tömb m-edik elemében

2. Mivel a K-means során a kezdeti klaszterközéppontok kiválasztása minden egyes futásnál különböző, ezért kis mértékben az eredmény is mindig változik. Tehát a $validity$ is mindig változik, így nem biztos, hogy minden iterációban ugyanaz a K' lesz az optimum. Az így létrejövő variációk közül ki kell szűrnöm a megfelelőt, ennek segítése érdekében hoztam létre az M tömböt. Következő lépésként az M tömbben megkeresem, hogy melyik érték fordul elő a legtöbbször (leggyakoribb K'), ez lesz a végső K
3. Legyen $minE = \infty$, és az l-edik iterációs lépésben:
    a. Futtatom a Kernel K-means klaszterezést az $\{x_i\}$ adatvektorokon K klaszterszámmal
    b. A kapott klaszterezésre vonatkozó $E(X)$ értéket összehasonlítom $minE$ értékével:
        i. ha kisebb, akkor ez lesz az új $minE$, és eltárolom a klaszterezés adatait
        ii. egyébként haladok tovább
4. Az l-edik iteráció után kiolvasom az eltárolt klaszterezést, és ez lesz a végső klaszterezése $\{x_i\}$-nek

A fenti algoritmussal tehát megvalósítottam a klaszterezését a Fisher-vektoroknak, ezekbe visszahelyettesítve a maszkolt képeket pedig megkapom a hasonló objektumok összerendelését. Az eddig bemutatott rendszer hatékonyságát több szempontból, több bemeneti képhalmazon teszteltem. Ezekről a következő fejezetben számolok be.

5 A RENDSZER TESZTELÉSE

Az elkészített rendszert több szempontból, illetve több különböző bemenettel teszteltem. Ebben a fejezetben a kapott eredményeket kiértékelem és elemzem is. A tesztelés során az előfeldolgozást végző modul hatékonyságára és fontosságára is kitérek.

5.1 KÉPSZEGMENTÁLÓ ALRENDSZER ELEMZÉSE

Ahhoz, hogy eredményes legyen a hasonló objektumok csoportosítása, elengedhetetlen, hogy azokat az objektumokat jól ki tudjuk ragadni a képből. Ezért a képszegmentálásnak elfogadható módon kell teljesítenie, különben akár többet árthat, mint használ. Az általam használt képszegmentáló eljárás nagy része azonos egy már korábban is létezővel, amelynek hatékonysága bizonyított [22]. Ennek ellenére egy saját kiértékelést is végeztem, mely segítségével megbizonyosodtam a helyes működésről a módosításaim után is, illetve be tudtam állítani néhány fontos paramétert.

5.1.1 KIÉRTÉKELT MÉRŐSZÁMOK

Most néhány olyan leírót szeretnék bemutatni, melyek jól jellemzik a szegmentálás hatékonyságát, így ezek segítségével végzem a kiértékelést. Az egyik ilyen az úgynevezett *F-measure*, amely a szegmentálás pontosságát jellemzi úgy, hogy súlyozott átlagát számítja a *precision* és *recall* mutatóknak. Az általános formula a következő:

$$F_\beta = (1+\beta^2)\cdot\frac{precision\cdot recall}{(\beta^2\cdot precision)+recall} \tag{23}$$

Itt β egy tetszőleges pozitív valós számot jelöl. Én $\beta = 1$ értékkel számolok, mely esetén az F_1 érték a precision és recall mutatók harmonikus közepe lesz. Ennek számításához előbb az említett pontosság és fedés értékeket kell meghatároznom, melyek a konfúziós mátrixból származtathatók a következőképpen:

$$precision = \frac{tp}{tp+fp} \tag{24}$$

$$recall = \frac{tp}{tp+fn} \tag{25}$$

A tp jelenti a helyesen igaz (true positive), fp a helytelen igaz (false positive), illetve fn jelöli a helytelen hamis (false negative) eredményeket. Ezeket pixel-szinten számítom ki, vagyis minden pixelt megvizsgálok, hogy az említett esetek melyike igaz rá. A meghatározásukhoz szükségem van manuális, etalon szegmentációkra, melyek minden tesztelt képre adottak, így tudom meghatározni, hogy valójában egy kép pixelének milyennek kellene lennie.

További fontos mutatók még a *pontosság* (*accuracy,* röviden: *acc)*, illetve az *Intersection Over Union (IOU)*. Az accuracy nem más, mint a pixel-szintű egyezések számának normálisa. Megvizsgálom, mennyi pixelt találtam el helyesen, és ezek számát osztom a kép pixeleinek teljes számával. Ez a mutató is felírható a konfúziós mátrix elemeivel:

$$acc = \frac{tp+tn}{tp+fp+fn+tn} \tag{26}$$

A tn pedig a helyesen hamis (true negative) pixelek számát felöli. Az IOU kiszámításához a hányadosát képezem az általam jósolt szegmentáció és a valós szegmentáció metszetének és uniójának. A szegmentáció annál pontosabb, minél jobban takarásban van a kettő, tehát az unió nem „lóg” túl nagyon a metszeten. Az említett mutatók közül mindegyiknek az értékkészlete a $[0, 1]$ intervallum, és minél jobban közelít egy mutató értéke az 1-hez, annál pontosabb a szegmentáció annak szempontjából.

5.1.2 A HASZNÁLT KÉPHALMAZ

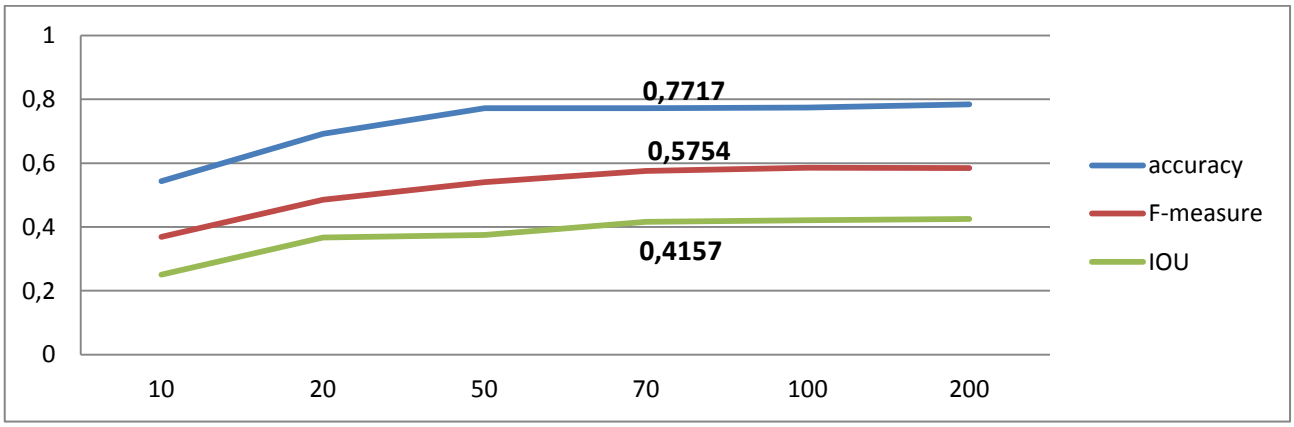
A fenti mérőszámokból kiderül, hogy mindenképpen olyan teszthalmazra van szükségem, amely rendelkezik manuális szegmentációval, különben a kiértékelés nem valósítható meg. A tesztelés során egy 900 képből álló, manuális szegmentációkkal rendelkező képhalmazt használtam, melyet a CALVIN csoport weboldaláról[6] töltöttem le. Ezek változatos, több objektumot tartalmazó képek, és előfordulnak olyanok is, ahol az objektumok lelógnak a képről vagy takarják egymást. Fontos megjegyezni, hogy azért ekkora méretű a teszthalmaz, mert a paraméterek finomításához többször is le kellett futtatni a szegmentálást, ami nagyobb méretek esetén lényegesen tovább tartott volna. A

[6] http://groups.inf.ed.ac.uk/calvin/proj-imagenet/page/index.html

2.3-as szakaszban említett forráshalmaz elkészítéséhez a PASCAL 2012-es évben publikált képeiből azokat használtam fel, melyekhez adottak voltak az etalon szegmentációk [11][12]. Ezek összesen 20 különböző típusú objektumot tartalmaznak, viszont ahogy azt kifejtettem, a szegmentálás szempontjából irrelevánsak ezek a típusok. Így a forráshalmaz elkészítéséhez 2913 képet használtam, melyen 6929 objektum található összesen.

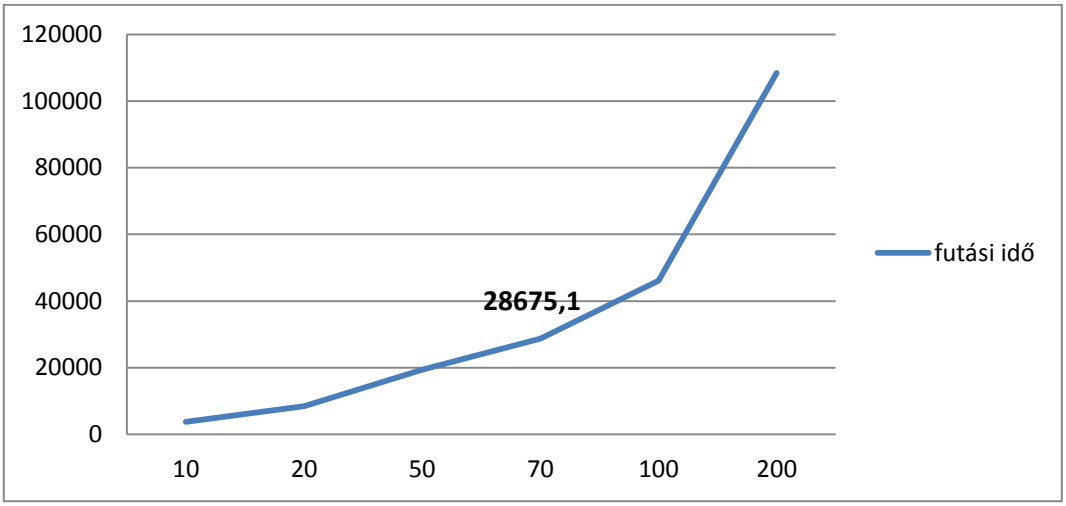
5.1.3 EREDMÉNYEK BEMUTATÁSA

Az egyik legfontosabb paramétere a szegmentálási folyamatnak az, hogy egy tesztképen mennyi befoglaló dobozt határozunk meg. Ezt követően ezekkel dolgozik a rendszer, tehát ez mindenre befolyással van az egész eljárásban. Ahhoz, hogy ennek megfelelő számát meg tudjam határozni, több különböző értékkel is kipróbáltam, ezek rendre 10, 20, 50, 70, 100 és 200. A fentebb említett mérőszámokat kiértékeltem minden pontban, ennek eredménye látható a 20. ábrán.



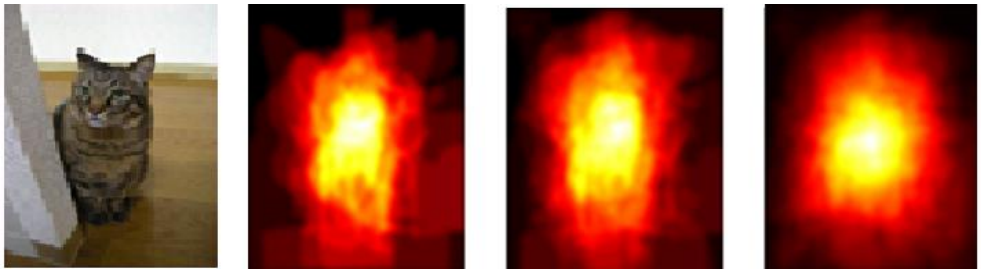
20. ábra: A fontosabb mérőszámok a befoglaló dobozok számának függvényében

A 20. ábrán látszik, hogy 70 befoglaló doboz után már jelentős javulás nem következett be (így 70-nél utólag feltűntettem a mérőszámok pontos értékét is az ábrán. Emellett még lemértem az egyes futási időket is, melyet a 21. ábrán jelenítek meg. Jól látszik, hogy a 100 befoglaló doboztól növelve már a futási idő rendkívül hosszú, így együttesen a mérőszámok és a futási idők alapján úgy döntöttem, hogy 70 befoglaló dobozt készítek képenként. Ebben az esetben egy kép szegmentálásának ideje átlagosan 31,86 másodperc, a teljes folyamat pedig 28675,1 másodpercig tartott.



21. ábra: Futási idő a befoglaló dobozok számának függvényében

További fontos paraméter még az, hogy a bináris maszkokat milyen (azonos) méretűre transzformáljam, ezt kipróbáltam 30×30, 50×50 illetve 120×120 méretekkel. A kiértékelt mutatókon nem sokat változtatott, viszont a 120×120 esetben már észrevehető volt a futási idő növekedése, ezért az 50×50 méretet választottam. A 2.3.2-es részben tárgyalt szomszédszám meghatározása is fontos a hatékonyság szempontjából. A 22. ábra macskát ábrázoló képe mellett annak folytonos maszkjai láthatók rendre 15, 30 majd 100 legközelebbi szomszéd alapján. Az első két képen látható, hogy a forma jobban illeszkedik az eredeti objektumra, az utolsó esetben viszont már annyi szomszédot vizsgálunk, hogy minden irányban van egy kis eltérés az eredetitől, így tulajdonképpen annak a formáját elveszítjük. Tehát $k = 30$-ig mindenképpen érdemes elmenni, viszont én az implementáció során $k = 50$-el próbálkoztam, ami szintén megfelelt. Ez az a határ, ahonnan szerintem már jelentős javulás nem várható, viszont romlás igen, a teszt ablak folytonos maszkját illetően.



22. ábra: A vizsgált kép és annak folytonos maszkjai a szomszédszám függvényében (forrás: [1])

5.2 KLASZTEREZŐ ALRENDSZER ELEMZÉSE

Ebben az alfejezetben a klaszterező algoritmusom elemzésére koncentrálok. Fontos megjegyezni, hogy a legtöbb klaszterező eljárást „csak” a klaszterezés eredménye szempontjából szokás vizsgálni, a választott feladatom esetén azonban még a klaszterszám meghatározását is meg kell figyelnem. Ezért a klaszterező eljárásomat több szempontból is tesztelem, külön a klaszterszám meghatározását, illetve magát a klaszterezést is. Mivel hasonló objektumok összerendeléséről van szó, ezért előfordulhat, hogy nem teljesen egyező, de hasonló objektumok is bekerülnek ugyanabba a klaszterbe, így a klaszterszám nem feltétlenül egyezik meg minden esetben az objektumtípusok számával.

5.2.1 KIÉRTÉKELT MÉRŐSZÁMOK

Ahogy a szegmentáló alrendszer elemzése esetén, most is meghatározok néhány mutatót, melyek alkalmasak a klaszterezés hatékonyságának mérésére. Ezek a *pontosság* (*accuracy*), illetve a *tisztaság* (*purity*) [37]. Az accuracy-t hasonlóan a 26. egyenletben leírtakhoz, most is a konfúziós mátrixból származtatom, viszont jelen esetben a mátrix elemei, bizonyos klaszterezési döntések számosságát jelentik. Egy döntés két objektum viszonyát határozza meg, tehát az összes döntés száma: $\binom{n}{2}$. A tp jelöli azon döntések számát, mikor két valóban hasonló objektumra a klaszterező algoritmus is azt mondja, hogy hasonlóak, vagyis azonos klaszterbe helyezi azokat. A tn azt jelenti, hogy két különböző objektumot két különböző klaszterbe sorol a rendszer. Ennek a két számnak az összege a helyes klaszterezési döntések száma. A hibásak pedig: fp, ami azt jelenti, hogy két különböző objektum azonos klaszterbe került, illetve fn, mikor azonos klaszterbe helyez a rendszer két különböző objektumot. Ez tehát az algoritmus helyes döntéseinek százalékát adja meg, így az egész eljárás hatékonyságát jellemzi. Szokás még *Rand-measure*, esetleg *Rand-index* néven is hívni.

Ezzel szemben a tisztaság meghatározható csak egy adott klaszterre, illetve az egész klaszterezésre is. Ehhez minden C_l klaszterhez hozzárendelek egy $\mathcal{O}_l$ számot, amely annak az objektumnak számossága lesz, amelyből a legtöbb található a C_l-ben. Ezzel a bevezetéssel a tisztaság a következőképpen írható fel egyetlen klaszterre:

$$purity(C_l) = \frac{\mathcal{O}_l}{N_l} \quad (24)$$

ahol N_l-el szokásosan a C_l klaszter számosságát jelölöm. Ezen túl kiszámítom a tisztaságot a teljes klaszterezésre is, amely n darab Fisher-vektor esetén a következő:

$$purity(\{C_1 \dots C_l\}) = \frac{1}{n} \times \sum_{l=1}^{K} \mathcal{O}_l \quad (25)$$

A tisztasággal tehát az határozható meg, hogy egy klaszter (illetve a klaszterezés során létrejött klaszterek összessége) milyen mértékben tartalmaz elemeket ugyanabból az osztályból (vagyis ugyanabból a típusú objektumból).

5.2.2 TESZTELÉSHEZ HASZNÁLT KÉPEK

Most olyan teszthalmazra van szükségem, amelyhez adottak a valós osztálycímkék, tehát, hogy az egyes képek, objektumok közül ténylegesen melyek hasonlóak. Erre azért van szükség, mert csak így tudom kiértékelni az eredményeket. Ez persze valós probléma esetén nem lehetséges, hiszen ott pont az az elvárás, hogy megadjuk ezeket az összerendeléseket, viszont a rendszer teszteléséhez tökéletesen használható, így megállapítható annak elvárt hatékonyága egy hasonló bemeneti képhalmaz esetén.

A választott képhalmaz az ImageCLEF idén megrendezett, úgynevezett LifeCLEF[7] versenyének egyik részfeladatához kiadott képekből áll. A versenyfeladat eredetileg növények osztályozása volt, amelyben magam is részt vettem, és a csapatommal az előkelő 3. helyen végeztünk [36]. Ezek megfelelnek az elvárásoknak a klaszterezés teszteléséhez, hiszen minden képhez adott annak osztálycímkéje. A képhalmazban rengeteg féle növény található, illetve azoknak részei, mivel az esetek többségében nem a teljes növényt ábrázolja a kép, annak éppen csak a virágát, termését, levelét, törzsét, stb. Az általam felhasznált képek mindegyike virágot vagy levelet ábrázol (lásd 23. ábra).

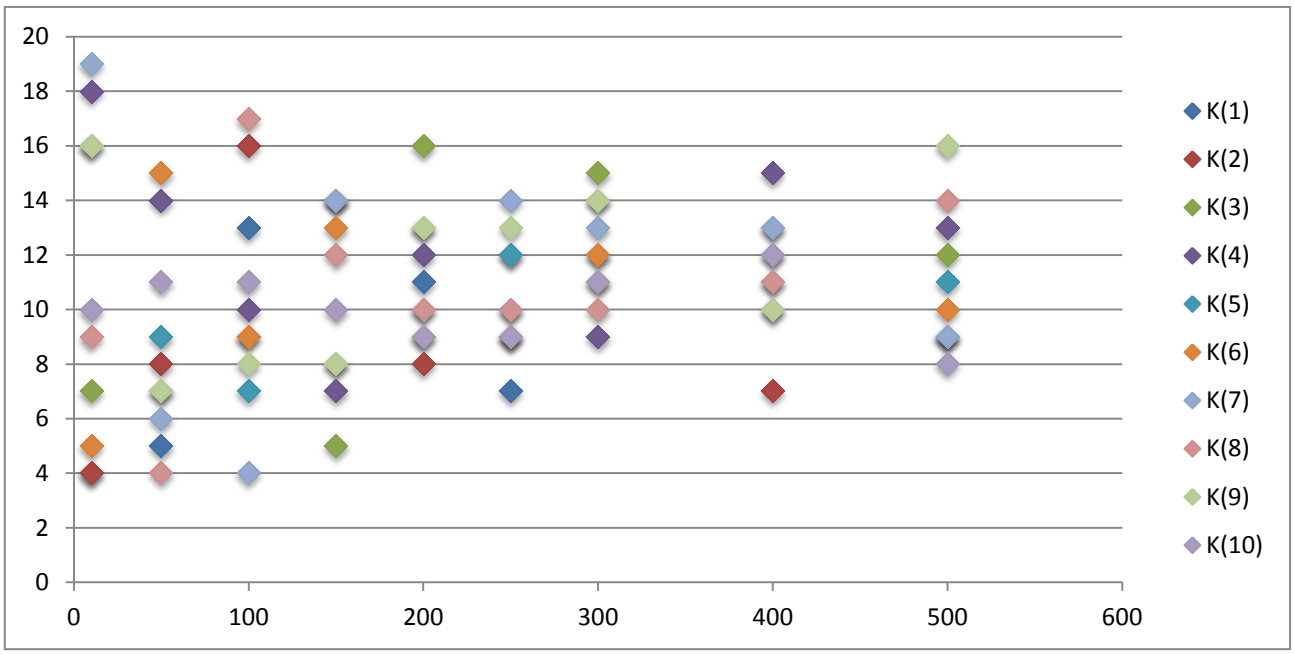
23. ábra: Néhány, a klaszterezéshez használt képek közül

[7] http://www.imageclef.org/2014/lifeclef/plant

Összesen 1194 képet gyűjtöttem ki a több százezres képhalmazból, 12 különböző típusú objektummal, ezeken teszteltem az klaszterező eljárást. A különböző objektumokból nem azonos számút választottam, van, amelyikből 18, illetve 232 szerepel a teszthalmazban.

5.2.3 KLASZTERSZÁM MEGHATÁROZÁSÁNAK TESZTELÉSE

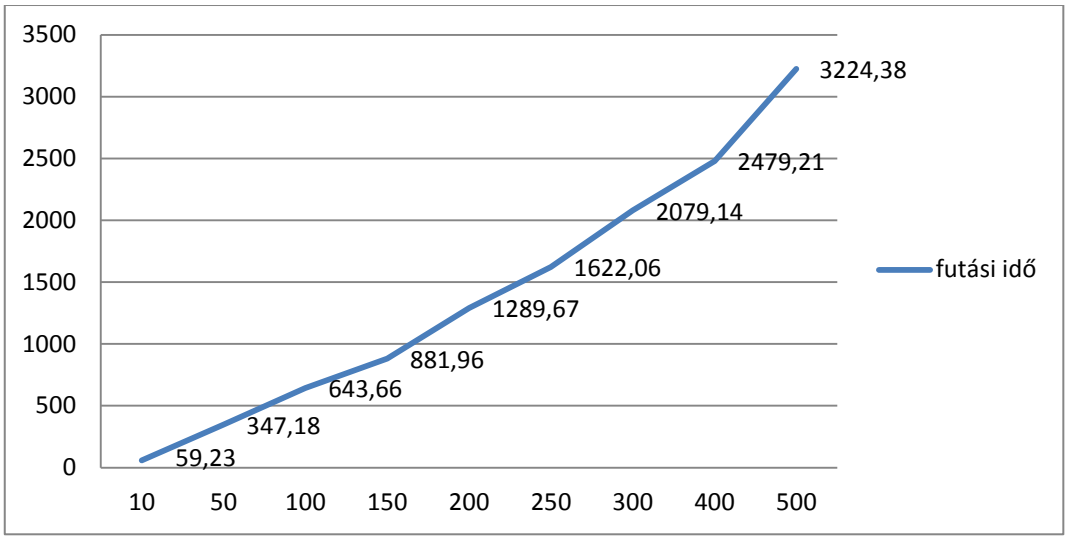
Az algoritmus fontos lépése, hogy megbecsüli azt, hogy az adott teszthalmaz mennyi klaszterből áll. Ehhez a 4.4 részben ismertetett K-t kell meghatároznom, a finomítható paramétereim pedig a $minK, maxK, m$ pozitív egész számok. Ezek mindegyike tulajdonképpen iterációs lépésszámot jelöl ki. A klaszterszám meghatározásához egy m hosszú tömböt töltök ki, majd annak a leggyakrabban előforduló elemét választom ki, mint K. Az elképzelésem az, hogy az eljárást sokszor lefuttatva, még néhány helytelen eredmény esetén is várhatóan egyre többször kapok „megfelelő” klaszterszámot. Tehát magán a klaszterszám meghatározáson nem változtatok, azt a 4.2 részben leírtak alapján végzem, kizárólag az m paramétert próbálom meg beállítani. Ehhez 10-szer lefuttattam a klaszterezést, és minden futtatásnál kipróbáltam m következő értékeit: 10, 50, 100, 150, 200, 250, 300, 400 és 500. Ennek eredménye látható a 24. ábrán.



24. ábra: Becsült klaszterszámok az m paraméter függvényében

A fenti ábrán $K(1) \dots K(10)$ jelöli az egyes futtatások számát, és m minden próba értékéhez megjelenítettem, hogy a 10 futás esetén milyen klaszterszámokat kaptam. Jól látszik, hogy az m növelésével a futtatásonként kapott K-k sűrűsödnek a 12-es valós klaszterszám köré. Megjegyzem, hogy a 24. ábrán látható rombuszok némelyike ugyanazon ponton van, mint egy másik rombusz, így csak a legfelsők láthatók.

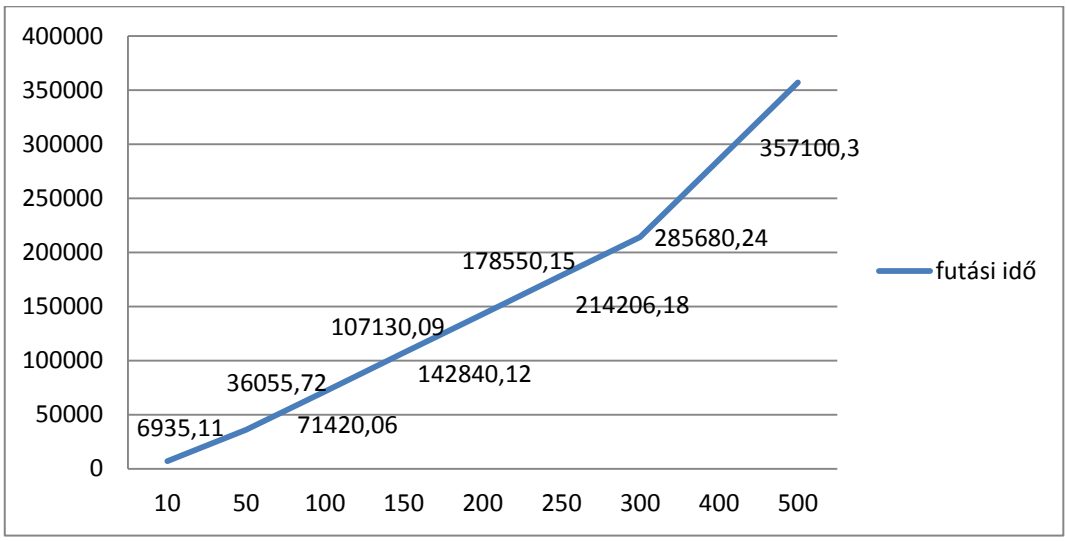
A kapott eredmények értékeléséhez még megvizsgáltam a futási idejüket is. Erre nagy befolyással bír a $minK$ *és* $maxK$ értéke, hiszen egyetlen iteráción belül, még $maxK - minK$ további iteráció fut, mely mindegyikében klaszterezem egyszer az adathalmazt. A 24. ábrán látott eredmények kiszámításához manuálisan megadtam a $minK = 2$, $maxK = 25$ értékeket, annak érdekében, hogy az algoritmusnak ne kelljen túl széles intervallumot végigpásztáznia. Az így kapott futtatások idejét a 25. ábrán foglaltam össze. Megjegyzem, kizárólag az M tömb kitöltésének idejét jelenítem meg a méretétől függően, tehát az algoritmus többi részével most nem foglalkozom Látható, hogy $m = 500$ esetén nem sokkal kevesebb, mint egy óra alatt lefut az M tömb kitöltése. Erre viszont általában nincs lehetőség, hiszen teljesen automatikus esetben elvárt, hogy ne kelljen segítségül megadni a $minK, maxK$ paraméterek értékét.



25. ábra: Futási idők másodpercen az m paraméter függvényében ($maxK = 25$ értékkel)

Felhasználói segítség nélkül a legjobb, amit az algoritmus tenni tud, hogy $minK = 2$-től $maxK = \left\lceil \frac{n}{2} \right\rceil$-ig fut, ami jelen esetben $\frac{1194}{2} = 597$. Így biztosan nem hagyok ki egyetlen

lehetséges klaszterszámot sem, feltéve, hogy egyik klaszter sem áll egyetlen elemből. Ebben az esetben az M egyetlen elemének kitöltéséhez átlagosan ≈714,2 másodpercre van szükség, ami abból a mérésből adódik, hogy egyszer futtattam az eljárást $m = 100$ mellett. Feltéve, hogy a futási idő az m további növelésével lineárisan nő (ahogy eddig), megbecsültem mennyi idő lett volna, ha futtatom az algoritmust annak további értékeire is (lásd 26. ábra).



26. ábra: Becsült futási idők az m paraméter függvényében, segítség nélküli esetben

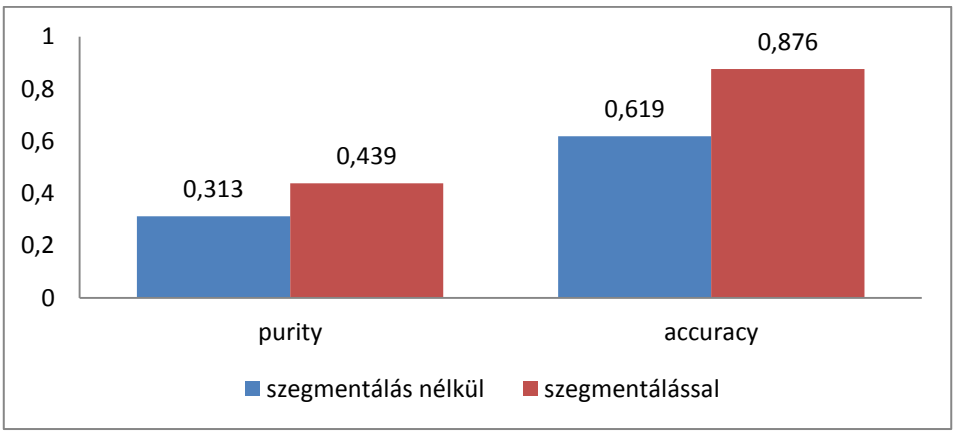
A fenti ábrán látszik, hogy a futási idő $m = 10$ esetén is több mint kétszeresen meghaladja a 25. ábrán látható $m = 500$ esetet. Tehát a minK, maxK paraméterek rendkívül befolyásolóan hatnak a futási időre. Ez egy olyan pontja a rendszernek, amit továbbfejlesztésre szorul, hiszen pár ezer kép tesztelése esetén nem várható, hogy 2-3 napot is megvárjunk, mire megkapjuk az eredményt. A fentiek alapján a végső implementációban $m = 150$ választás mellett maradtam, hiszen ez már viszonylag kevés szórással határozza meg a K paramétert, és a futási ideje is „mérsékelt” (107130,9 másodperc).

5.2.4 VÉGSŐ KLASZTEREZÉS

A tesztelés eddigi fázisaiban arra koncentráltam, hogy az egyes algoritmusok paramétereit optimálisan tudjam beállítani. Ebben a részben a kapott paraméterek értékeit felhasználva egy teljes futtatást fogok elemezni, melybe beletartozik a

szegmentálás, az objektum reprezentáció majd a klaszterezés is. Az egyetlen ismeretlen paraméter az l maradt, amely a kiszámított K klaszterszámmal történő Kernel K-means futtatások számát határozza meg, melyek segítségével az $E(X)$ kifejezést minimalizálom. Mivel itt már nincsenek további belső ciklusok, és egyértelműen kijelenthető, hogy minél többször iterálok, annál nagyobb a valószínűsége, hogy megtalálom az optimális klaszterezést, ezért $l = 100000$ értékkel dolgoztam.

A klaszterszám kiválasztás feladatánál az algoritmus a $K = 9$ választás mellett döntött. Az eredményeket kiértékeltem, a fentebb definiált mérőszámok a következők: $purity(\{C_1 \dots C_K\}) = 0{,}439$, illetve $acc = 0{,}876$. Ahogy az accuracy is mutatja, $\approx 88\%$-os pontossággal tudja eldönteni a klaszterező, hogy két objektum hasonló-e vagy sem. A purity az egyes klaszterek tisztaságát méri. Ez várható volt, hogy nem lesz túl magas, hiszen eleve 9 klaszterbe sorolta az algoritmus a bemenetként adott 12 valódi csoportot (12 külön növényfajt) tartalmazó képhalmazt, tehát minimális esetben is van 3 olyan klaszter, melynek tisztasága alacsony. Az algoritmus összesen 139103,6 másodpercig futott, tehát átlagosan $\approx 116{,}5$ másodpercbe került egyetlen képnek a feldolgozása. Ebből a megtalált K-val történő végső klaszterezés ($l = 100000$-el) 2043,2 másodpercig tartott, a maszkolt képek előállítása pedig 36114,8 másodperc volt, ami meghaladja a 10 órát. Kérdés, hogy szükséges-e egy ilyen hosszú előfeldolgozás? Ennek kihagyásával a futási időt jelentősen lehetne csökkenteni ebben az esetben. Annak érdekében, hogy választ kapjak erre a kérdésre, lefuttattam a teljes eljárást a nyers, szegmentáció nélküli képeken. Ennek eredményeit a 27. ábrán hasonlítom össze az előbbi eredményekkel.



27. ábra: Eredmények szegmentálással és anélkül

A fenti ábrán jól látszik, hogy a szegmentálás jó hatással van a klaszterezésre, hiszen mindkét mutatóban jelentős növekedést értem el vele. Ezért a válasz az, hogy igen, szükség van az előfeldolgozásra a pontosabb, hatékonyabb munka érdekében. A 28. ábrán látható 8 kép közül a felső 4 az eredeti, klaszterezni kívánt kép, az alsó négy, pedig az azokból maszkolt képek. Ez a 4 kép a szegmentálás nélküli esetben (tehát mikor a felső 4 képet klaszterezte az algoritmus) ugyanabba a csoportba került. Ez helytelen, mivel az első két kép a *Pittosporum tobira (Thunb.) W.T.Aiton*, a második kettő pedig a *Nerium oleander L* nevű növényfaj egy-egy virága. Ez azért történt így, mert a képek globálisan, a hátterükkel együtt jobban hasonlítanak, hiszen ilyenkor a háttérről is készül reprezentáció, ami belekerül a Fisher-vektorba, ez által bezavar a klaszterezésnél. A szegmentálást is használva, a klaszterező eljárás az alsó 4 képet klaszterezi. Szembetűnő, hogy ezeken a képeken már nincs semmilyen zavaró plusz elem, kizárólag a lényeges részek, az objektumok (jelen esetben virágok). Éppen ezért sikeresen szét is választja a két különböző növényfaj virágait, így az első kettő és a második kettő kép különböző klaszterekbe kerülnek.

28. ábra: Két különböző növényfaj virágai, illetve azok maszkolt változatai

A klaszterezéshez kapcsolódó tesztek eredményeit a 2. táblázatban foglaltam össze. Az első 9 sorban a „becsült K értékek” oszlopban megjelenítettem minden különböző értéket, melyet az adott m 10-szeri futtatásával kaptam.

$minK$	$maxK$	m	becsült K értékek	l	$purity$	$accurac$	megjegyzés	idő (h)
2	25	10	4,5,7,9,10,16,18,19	-	-	-	szegmentálással	0,016
2	25	50	4,5,6,7,8,9,11,14,15	-	-	-	szegmentálással	0,096
2	25	100	4,7,8,9,10,11,13,16,17	-	-	-	szegmentálással	0,179
2	25	150	5,7,8,10,12,13,14	-	-	-	szegmentálással	0,245
2	25	200	8,9,10,11,12,13,16	-	-	-	szegmentálással	0,358
2	25	250	7,9,10,12,13,14	-	-	-	szegmentálással	0,451
2	25	300	9,10,11,12,13,14,15	-	-	-	szegmentálással	0,578
2	25	400	7,10,11,12,13,15	-	-	-	szegmentálással	0,689
2	25	500	8,9,10,11,12,13,14,16	-	-	-	szegmentálással	0,896
2	579	150	9	100000	**0,439**	**0,876**	szegmentálással	38,64
2	579	150	6	100000	0,313	0,619	szeg. nélkül	32,35

2. táblázat: A klaszterezési tesztek eredményeinek összefoglalása

Szeretném megemlíteni, ahogy a táblázatban is látszik, a korábban kalkulált 10 órás különbség a szegmentálás nélküli esethez képest, 6,3 órára csökkent. Ez annak köszönhető, hogy ilyenkor a teljes képből sokkal több jellemző (SIFT) leírót állít elő a rendszer, az azokkal történő további részfeladatok ideje pedig ez által megnő. Ezzel a tapasztalattal az is kijelenthető, hogy a szegmentálás a klaszterezés hatékonyságán túl, annak futási idején is javít.

6 ÖSSZEFOGLALÁS

A dolgozatban elkészítettem egy olyan automatikus klaszterező rendszert, amelynek feladata, hogy egy ismeretlen képhalmazban található hasonló objektumokat összecsoportosítsa, egymáshoz rendelje. A rendszer három fő alrendszerből épül fel. Az első az előfeldolgozást végzi, ami egy előtér-háttér szegmentálását jelenti minden egyes bemeneti képnek. A feldolgozott képeket, úgynevezett maszkolt képeket reprezentálja matematikai módszerek segítségével a következő modul. Ennek során egy olyan komplex leírót alkotok, a Fisher-vektort, mely egy objektum egészét képes leírni egyetlen vektorral. Ezek egyértelműek és megkülönböztethetőek egymástól. Az utolsó alrendszer a klaszterezés, mely a Fisher-vektorokat kapja bementként, ezeket klaszterezi. Az általam tervezett és implementált klaszterező eljáráshoz nem szükség előismeret a bementről, mivel automatikusan meghatározza az optimálisnak vélt klaszterszámot. Ehhez több módszert ötvöztem, illetve saját tervezésű lépésekkel is kiegészítettem ezt a komplex eljárást. A meghatározott klaszterszámmal majd egy végső klaszterezést futtatok, melynek eredménye a hasonló Fisher-vektorok csoportosítása. A Fisher-vektorok helyébe az objektumokat visszatéve megkaphatók az összerendelések.

A munkám utolsó fázisa egy több lépcsős tesztelés volt, mely keretein belül elsőként optimalizáltam a fontosabb paramétereit a rendszernek. Ehhez külön teszteltem az alrendszereket, és finomítottam azok beállításait. A kapott paraméterekkel egy teljes kiértékelést is végeztem, melyből kiderült, hogy a klaszterezés $\approx 88\%$ pontossággal meg tudja mondani két objektumról, hogy azok hasonlóak-e vagy sem (az adott bemeneti képek mellett). A tesztelés közben kiderült a rendszer legnagyobb hátránya is, ami a futási idő. Ezen a további munkám során tervezek javítani úgy, hogy a hatékonyság rovására ne menjen.

A TDK dolgozatom azon részéből, amely a képek leíróját állítja elő osztályozási feladatokhoz, egy nemzetközi konferencia cikk készült [36]. Ezen kívül a dolgozatom objektum felismerési részéből egy folyóirat cikk van készülőben: egy impact factor-os folyóirathoz benyújtott kétszerzős cikkünk ugyanis túljutott az első bírálati körön.

A rendszert továbbfejlesztem a jövőben, hogy képes legyen olyan képek kezelésére is, melyek több objektumot is tartalmaznak egyszerre. Továbbá tervezés alatt van egy olyan klaszterezési módszer is, mely egy hasonlósági konfidenciát javasol minden képpárra, és úgy próbálja meg elvégezni a csoportosítást. A későbbi munkám során ezt is integrálom a most bemutatott rendszerbe.

7 IRODALOMJEGYZÉK

[1] A. Rosenfeld and D. Weinshall. *Extracting foreground masks towards object recognition*. In ICCV, 2011

[2] Alexe, B., Deselaers, T., Ferrari V.: *Measuring the objectness of image windows*, IEEE Transactions on Pattern Analysis and Machine Intelligence, November 2012.

[3] Alexe, B., Deselaers, T., Ferrari V.: *What is an object ?*, IEEE Computer Vision and Pattern Recognition (CVPR), San Francisco, June 2010.

[4] Altrichter Márta, Horváth Gábor, Pataki Béla, Strausz György, Takács Gábor, Valyon József: *Neurális hálózatok,* PANEM, 2006, http://mialmanach.mit.bme.hu/neuralis/ch06s03, (letöltés dátuma: 2014.10.12.)

[5] C. Harris, M. Stephens: *A combined corner and edge detector*. In C. J. Taylor, editors, Proceedings of the Alvey Vision Conference, pages 23.1-23.6. Alvey Vision Club, September 1988. doi:10.5244/C.2.23.

[6] Canny, J.: *A Computational Approach to Edge Detection*, Pattern Analysis and Machine Intelligence, Pages: 679 – 698, ISSN : 0162-8828, 1986.

[7] Carlo Tomasi: *Estimating Gaussian Mixture Densities with EM – A Tutorial,* Duke University. http://www.cs.duke.edu/courses/spring04/cps196.1/handouts/EM/tomasiEM.pdf (letöltés dátuma: 2014.10.07.)

[8] Dempster, A., Laird, N., Rubin, D.: *Maximum Likelihood from Incomplete Data via the EM Algorithm*, Journal of the Royal Statistical Society 39(1), 1977, pp. 1–38.

[9] Deng, J., Dong, W., Socher, R., Li, L., Li, K., Fei-Fei, L.: *ImageNet: A Large-Scale Hierarchical Image Database*, Computer Vision and Pattern Recognition, 2009, ISBN:978-1-4244-3992-8.

[10] Ester M., Kriegel H.-P., Sander J., Xu X.: "*A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*", Proc. 2nd int. Conf. on Knowledge Discovery and Data Mining (KDD '96), Portland, Oregon, 1996, AAAI Press, 1996.

[11] Everingham, M. and Van Gool, L. and Williams, C. K. I. and Winn, J. and Zisserman, A.: *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*.

[12] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J. and Zisserman, A.: *The PASCAL Visual Object Classes (VOC) Challenge,* International Journal of Computer Vision, 88(2), 2010, pp. 303-338.

[13] Florent Perronnin, Chris Dance: *Fisher kernel on visual vocabularies for image categorization*, CVPR, Computer Vision and Pattern Recognition, 2007.

[14] Greig, D.M., Porteous, B.T. and Seheult, A.H.: *Exact maximum a posteriori estimation for binary images,* Journal of the Royal Statistical Society Series B, 1989.

[15] Guillaumin, M., Mensink, T., Verbeek, J., Schmid, C.: TagProp: *Discriminative metric learning in nearest neighbor models for image auto-annotation*, Computer Vision, IEEE 12th International Conference, E-ISBN : 978-1-4244-4419-9 Print ISBN: 978-1-4244-4420-5, 2009.

[16] Hou, X. Z.: Saliency Detection: *A Spectral Residual Approach*. Department of Computer Science, Shanghai Jiao Tong University: In CVPR, 2007.

[17] Inderjit Dhillon, Yuqiang Guan, Brian Kulis, Kernel k-means, *Spectral Clustering, and Normalized Cuts*, In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2004.

[18] J.T. Tou and R.C. Gonzalez, *Pattern Recognition Principles*, Massachusetts: Addison-Wesley, 1974.

[19] Jaakkola TS, Haussler D.: *Exploiting generative models in discriminative classifiers*. Advances in Neural Information Processing Systems (NIPS), Vol. 11, 1998, pp. 487-493.

[20] Jorge Sánchez, Florent Perronnin, Thomas Mensink: *Improved Fisher Vector for Large Scale Image Classification*, COMPUTER VISION – ECCV 2010. Lecture Notes in Computer Science, 2010, Volume 6314/2010, pp. 143-156.

[21] Kató Zoltán: *Sarokpontok detektálása*, Képfeldolgozás és Számítógépes Grafika tanszék SZTE. http://www.inf.u-

szeged.hu/~kato/teaching/DigitalisKepfeldolgozasTG/07-CornerDetection.pdf (letöltés dátuma: 2014.10.07.)

[22] Kuettel, D., Ferrari V.: *Figure-ground segmentation by transferring window masks*, IEEE Computer Vision and Pattern Recognition (CVPR), Providence, June 2012.

[23] Kuettel, D., Guillaumin M., Ferrari V., Vezhnevets, A.: *ImageNet Auto-annotation*, a projekt weboldala: http://www.vision.ee.ethz.ch/~mguillau/imagenet.html?calvin, megtekintés dátuma: 2014.10.05

[24] Kuettel, D., Guillaumin M., Ferrari V.: *ImageNet Auto-annotation with Segmentation Propagation*, Technical Report, Internation Journal of Computer Vision, 2013.

[25] Kuettel, D., Guillaumin, M., Ferrari, V.: *Segmentation Propagation in ImageNet*, European Conference on Computer Vision (ECCV), Firenze, Italy, October 2012.

[26] L. Fei-Fei, R. Fergus, and A. Torralba: *Recognizing and Learning Object Categories*, Part1 – Single object classes: *Bag of Words models, Part-based models, and Discriminative models,* 2009, pp. 2-16.

[27] Lior Rokach and Oded Maimon: *Clustering methods*. In The Data Mining and Knowledge Discovery Handbook, pages 321–352. 2005.

[28] Liu, T., Sun, J., Zheng, N., Tang, X., Shum, H.: *Learning to detect a salient object*. In CVPR, 2007.

[29] Lowe, D. G.: *„Distinctive Image Features from Scale-Invariant Keypoints”*, International Journal of Computer Vision, 60, 2, pp. 91-110, 2004.

[30] Max Welling, Kernel K-means and Spectral Clustering, University of Toronto, https://www.ics.uci.edu/~welling/teaching/273ASpring09/SpectralClustering.pdf, (letöltés dátuma: 2014.10.12.)

[31] Mikolajczyk, K., Schmid, C.: *Scale & affine invariant interest point detectors,* International Journal on Computer Vision 60(1), 2004, pp. 63-86.

[32] Radha Chitta, *Partitional Algorithms to Detect Complex Clusters*, 2003. http://www.cse.msu.edu/~cse802/Kernel%20k-means&Spectral_Clustering.pptx, (letöltés dátuma: 2014.10.12.)

[33] Reynolds, D. A.: *Gaussian Mixture Models*, Encyclopedia of Biometric Recognition, Springer, Journal Article, February 2008. http://www.ll.mit.edu/mission/communications/ist/publications/0802_Reynolds_Biometrics-GMM.pdf (letöltés dátuma: 2014.10.07.)

[34] S. Lazebnik, A. Torralba, L. Fei-Fei, D. Lowe, C. Szurka: *Bag-of-Words models*, Lecture 9, 2012, pp. 1-32.

[35] Szegedi Tudományegyetem - Informatikai Tanszékcsoport, Mesterséges Intelligencia II: *Felügyelet nélküli tanulás – A K-means klaszterező*, http://www.inf.u-szeged.hu/~ormandi/ai2/03-k-means.pdf (letöltés dátuma: 2014.10.09.)

[36] Szűcs Gábor, Papp Dávid, Lovas Dániel: *Viewpoints Combined Classification Method in Image-based Plant Identification Task*, In: (eds.) Cappellato L., Ferro N., Halvey M., Kraaij W., Working Notes for CLEF 2014 Conference, (Vol-1180, ISSN: 1613-0073), Sheffield, UK, September 15-18, 2014., pp. 763-770.

[37] Tan, P-N., Steinbach, M., Kumar, V.: *Adatbányászat*, Panem Kft., ISBN: 9786155186097, 2012.

[38] Turi, R.H., Ray, S., 2000, *Determination of the Number of Clusters in Colour Image Segmentation*, SCSSE Monash University, Clayton Vic Australia.