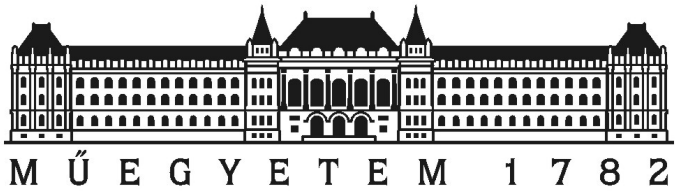


Budapest Műszaki és Gazdaságtudmányi Egyetem

Valós idejű objektum felismerés gépi látás segítségével

Varga Márton Bálint
TIBS2H

Tenk Milán
DL4O2F

Konzulens:
Dr. Csorba Kristóf

2013. október 25.

Kivonat

Az utóbbi években részben a korszerű okostelefonok és tabletek tömeges elterjedése miatt egyre inkább megjelentek hétköznapjainkban olyan alkalmazások, melyek a valós életben látottakat többlet információval egészítik ki. Ennek alapja, hogy eszközünk a kameraképen felismer egy megadott objektumot, majd annak megfelelően a valóságot virtuálisan kibővíti. Például ha ránézünk egy étterem logójára (tipikusan mobilunk vagy tabletünk segítségével, vagy akár például otthon webkamera segítségével), eszközünk kiírja, hogy hogyan juthatunk el oda a leghamarabb, és akár az étlapra is vethetünk virtuálisan egy pillantást.

Munkánk célja ezen folyamat első felének taglalása, azaz hogy miként tud eszközünk bizonyos objektumokat felismerni. Ennek során az egyes módszerek elméleti hátterének ismertetésén túl kitérünk olyan gyakorlati részletekre is, mint például ezen módszereknek az OpenCV gépi látás könyvtár felhasználásával való alkalmazása, és az erőforrásigényük vizsgálata. Ez utóbbi azért is fontos szempont, mert egyik célunk az okostelefonokon kielégítő teljesítménnyel történő futás elérése több lehetséges felismerhető objektum detektálása során. A dolgozatban három főbb objektum felismerésre alkalmas módszert vizsgálunk meg: a korrelációs mintaillesztés alapú felismerést, a kaszkádosított osztályozó alapú felismerést illetve a jellegzetesség megfeleltetés alapján való felismerést. A kaszkádosított osztályozó alapú detektor tárgyalása során külön kitérünk a Haar-szerű jellegzetességek és az LBP (Local Binary Patterns) jellegzetességek jellemzőire.

Dolgozatunk valós életből vett példák alapján bemutatja az elterjedt detektáló algoritmusok alkalmazhatóságát, mérési eredmények alapján szemlélteti azoknak korlátait, bemutatja azok Android platformon történő használatát. Részletezi, hogyan lehet az OpenCV által biztosított eszközök segítségével kaszkádosított osztályozó alapú detektorokat készíteni, majd az általunk fejlesztett környezetben tesztelni és értékelni a kapott eredményt. Bemutatja ennek kapcsán a felismerendő objektum betanítási folyamatának elméleti és gyakorlati menetét.

Az elvégzett munkánk alapja lehet objektum felismeréshez kapcsolódó további kutatási és oktatási feladatoknak.

Abstract

In the last few years partly because of the spreading of advanced smartphones and tablets there are more and more applications that provide extra information to things seen in real life. The base of this is that our device recognizes an object on the camera view and then it augments reality accordingly. For example if we look at a restaurant's logo (typically with our mobile or tablet device, or maybe with a webcamera) our device shows us how we can get there quickly and we can even take a virtual look at the menu.

Our goal is to cover the first part of this process which is about how to recognize specific objects with our devices. We will review not only the theoretical backgrounds for these methods but also practical details like how to utilize these methods using the OpenCV computer vision library and investigating hardware requirements for them. This last part is critical because one of our targets is to achieve decent performance even when running on smartphones while detecting multiple recognizable objects. In this paper we examine three major methods that can be used for recognizing objects: the template matching method based on correlation, the cascade classifier based recognition, and the feature matching based recognition. Related to the cascade classifier based detector we will also cover the attributes of Haar-like features and LBP (Local Binary Patterns) features.

Based on real life examples our paper demonstrates the applications of widespread detection algorithms, illustrates the limitations of them based on measurement results and it shows how to use them on the Android platform. It details how to create cascade classifier based detectors using tools provided by OpenCV, then how to test them and review the obtained results in our in-house developed test environment. It presents the theoretical and practical parts of the related training process for the recognizable objects.

Our work can be used as a base of additional research and educational activities related to object recognition.

HALLGATÓI NYILATKOZAT

Alulírott *Varga Márton Bálint és Tenk Milán*, hallgató kijelentem, hogy ezt a dolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelmûen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerzõ(k), cím, angol és magyar nyelvû tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhetõ elektronikus formában, a munka teljes szövegét pedig az egyetem belsõ hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik.

Budapest, 2013. október 25.

Varga Márton Bálint	*Tenk Milán*
hallgató	hallgató

Tartalomjegyzék

1. Bevezetés

A dolgozatban először áttekintjük több meghatározó objektumok felismerésére felhasználható módszer elméleti alapjait és ezeknek az Open Source Computer Vision Library (OpenCV) gépi látás könyvtár segítségével történő alkalmazását. Az elméleti áttekintés alapján a legalkalmasabbnak tekintett módszert részletesen teszteljük, és ezek eredményét közöljük. Végül összegezzük a tapasztalatainkat, és rávilágítunk arra, hogy a kiválasztott módszer mennyire alkalmas valós idejű objektum felismerési feladatok ellátására, és mennyire tudna például egy kiterjesztett valóságot implementáló szoftver alapjaként szolgálni.

A dolgozatban az elméleti áttekintést Varga Márton Bálint foglalja össze, mely 2. fejezettől a 4. fejezetig olvasható. A kaszkádosított osztályozókkal kapcsolatos gyakorlati vonatkozásokat - a tanítást és tesztelést támogató programok fejlesztését valamint a mérések elvégzését - Tenk Milán valósította meg. Az ezekkel kapcsolatos tapasztalatokat az 5. fejezettől a 7. fejezetig terjedően foglalja össze. Mivel ez egy közös dolgozat, többes szám első személyben mutatjuk be a tapasztalatainkat.

2. Mintaillesztés alapú objektum felismerés

A mintaillesztés (Template Matching) az egyik legegyszerűbb módszer, amelyet objektumok felismerésére fel lehet használni. Egyszerű működése következtében azonban leginkább csak bizonyos, ideálishoz közeli körülmények között használható megfelelő hatékonysággal, általánosabb esetekben összetettebb objektumok felismerésére nem kifejezetten alkalmas.

2.1. A módszer ismertetése

A módszer működésének alapvető lényege az, hogy egy képen meghatározza, hogy annak részei egy előre meghatározott mintaképhez mennyire illeszkednek, azaz mennyire hasonlóak. Ez a hasonlóság több korrelációt felhasználó módszer segítségével hatékonyan kiszámítható.

Két képre van tehát szükség az illesztés elvégzéséhez: egy mintaképre (T - Template Image), melyhez hasonló területet keresünk és egy forrásképre (I - Source Image), melyen a keresést végeznénk el. A módszer a mintakép hasonlóságának mértékét a forráskép egyes területeihez úgy határozza meg, hogy azon úgymond végigcsúsztatja a forrásképet. Ez úgy történik, hogy a forráskép egyik sarkából kiindulva a mintaképet pixelről pixelre vízszintesen és függőlegesen elmozgatja egészen addig amíg a forrásképet teljesen be nem járta, és minden vizsgált helyen meg nem határozta a mintaképnek és a forráskép aktuálisan vizsgált részének a hasonlóságát. Az egyes területeken a hasonlóság meghatározása sokféleképpen történhet. Az eddigiekből adódik, hogy a forrásképnek legalább akkorának kell lennie szélességben és magasságban is, mint a mintaképnek. A forráskép egyes területeihez tartozó illeszkedések mértékét a módszer egy eredmény mátrixban (R) tárolja el. Az eredmény mátrixban tárolt értékek tehát arra utal-

nak, hogy a hozzájuk tartozó, a mintaképpel azonos méretű területei a forrásképnek mennyire hasonlóak a mintaképhez. Amennyiben a forráskép és mintakép mérete sorban $W \times H$ illetve $w \times h$, az eredmény mátrix mérete szintén a fenti egyszerű csúsztatásos módszerből adódóan $(W - v + 1) \times (H - h + 1)$. Az algoritmus lefutása után a legjobb illeszkedést az eredmény mátrix globális minimuma vagy maximuma adja az illeszkedés meghatározásához felhasznált konkrét módszertől függően.

2.2. Az OpenCV implementáció

Az módszer OpenCV implementációja a *matchTemplate* nevű függvényben van megvalósítva, amelynek paraméterei a bemeneti forráskép, a bemeneti mintakép, a kimeneti kép és az illeszkedés mértékének meghatározására felhasznált módszer azonosítója. [1] A bemeneti képeket 8 bites képek vagy 32 bites képek formájában várja, és fontos hogy a két kép ugyanolyan formátumú legyen. Az eredményeket az eredménymátrixban 32 bites lebegőpontos számokként tárolja el, függetlenül attól, hogy egycsatornás vagy színes (több csatornás) képeket használtunk fel bemenetként. Az eredmény mátrixból a legjobb találat kiválasztása hatékonyan elvégezhető a *minMaxLoc* nevű függvénnyel. [2]

Az illeszkedések mértékének meghatározására az OpenCV három alapvetően különböző módszert támogat. A hasonlóságot az egyik módszer a két kép közti négyzetes különbségek alapján számítja (square difference) ki. Ekkor a 0 jelenti a legnagyobb hasonlóságot, és a minél nagyobb értékek a minél rosszabb hasonlóságot. Egy másik módszer a korrelációs illesztést (correlation matching) használja fel, amely a a két képet összeszorozza, ekkor nagy értékek jelzik a jó hasonlóságot, és 0 a rendkívül rossz hasonlóságot. A harmadik módszer a korrelációs együttható illesztést (correlation coefficient matching) használja, amely a képek átlagait felelteti meg egymáshoz képest, ekkor 1 jelzi a jó hasonlóságot, 0 jelzi ha nincs hasonlóság, a -1 pedig a rossz hasonlóságot jelzi. Az OpenCV támogatja továbbá ezen három módszer normalizált változatait is, melyek sokszor jobban felhasználhatóak és vizuálisan tisztábban ábrázolhatóak. Az alábbiakban a módszerek pontos képletei is ismertetésre kerülnek, ennek során a képek és az eredmény a fent bevezetett jelölésekkel vannak azonosítva, az összegzésekkor felhasznált x' és y' változók pedig értelemszerűen $x' = 0..w - 1$ és $y' = 0..h - 1$ értékeken futnak végig. A támogatott módszerek tehát az alábbiak:

- CV_TM_SQDIFF:

$$R(x, y) = \sum_{x',y'} (T(x', y') - I(x + x', y + y'))^2 \tag{1}$$

- CV_TM_SQDIFF_NORMED

$$R(x, y) = \frac{\sum\limits_{x',y'} (T(x', y') - I(x + x', y + y'))^2}{\sqrt{\sum\limits_{x',y'} (T(x', y')^2 \cdot \sum\limits_{x',y'} I(x + x', y + y'))^2}} \tag{2}$$

- CV_TM_CCORR:

$$R(x,y) = \sum_{x',y'} (T(x',y') \cdot I(x+x', y+y')) \tag{3}$$

- CV_TM_CCORR_NORMED:

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') \cdot I(x+x', y+y'))}{\sqrt{\sum_{x',y'} (T(x',y')^2 \cdot \sum_{x',y'} I(x+x', y+y'))^2}} \tag{4}$$

- CV_TM_CCOEFF:

$$R(x,y) = \sum_{x',y'} (T'(x',y') \cdot I'(x+x', y+y')) \tag{5}$$

ahol

$$T'(x',y') = T(x',y') - 1/(w \cdot h) \cdot \sum_{x'',y''} T(x'',y'') \tag{6}$$

$$I'(x+x', y+y') = I(x+x', y+y') - 1/(w \cdot h) \cdot \sum_{x'',y''} I(x+x'', y+y'') \tag{7}$$

- CV_TM_CCOEFF_NORMED:

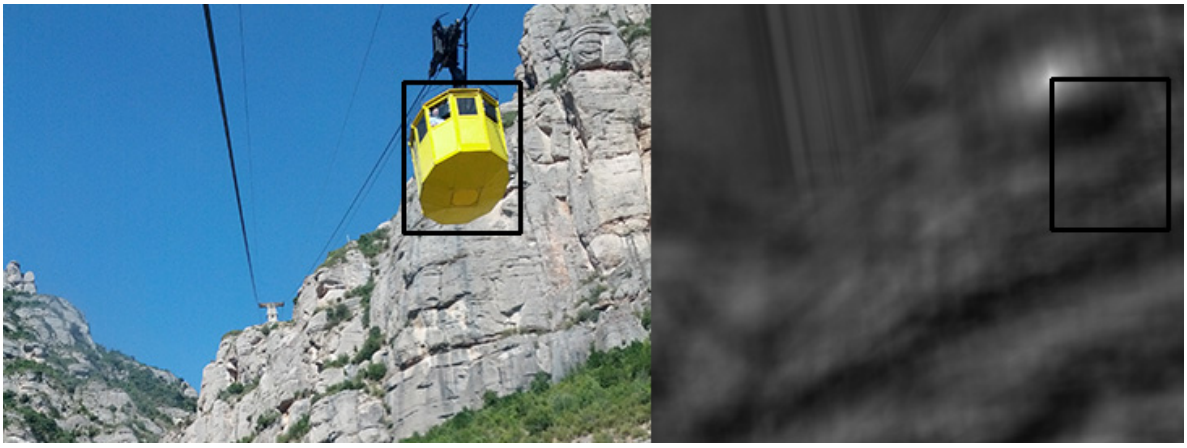
$$R(x,y) = \frac{\sum_{x',y'} (T'(x',y') \cdot I'(x+x', y+y'))}{\sqrt{\sum_{x',y'} (T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x', y+y'))^2}} \tag{8}$$

Színes bemeneti képek esetén a nevezőben a mintaképre vonatkozó szummázás, és a számlálóban szereplő minden szummázás külön lefut minden csatornára, így minden csatornára egy külön átlag kerül kiszámításra, illetve felhasználásra. Az módszer OpenCV implementációjával kapcsolatos további információk megtalálhatóak az OpenCV hivatalos weboldalán. [3]

2.3. Objektumfelismerés a módszer felhasználásával

Ahhoz, hogy a módszert objektumok felismerésére használjuk fel, mintaképként a felismerendő objektum képét, forrásképként pedig a kamerából kapott képet használhatjuk fel. A mintaképet (vagy mintaképeket) fontos, hogy jól válasszuk meg, és a felismerendő objektum jól látszódjon (megfelelő méretben, nézetben, orientációban, megvilágításban) hiszen a felismerés során csak ezekre a mintaképekre hagyatkozhatunk. Azt, hogy a felismerendő objektum szerepel-e a kamera képen, és ha szerepel akkor hol, az eredmény mátrixból a legnagyobb illeszkedésre utaló érték nagysága és mátrixban lévő pozíciója alapján becsülhetjük meg. Több felismerendő objektum esetén több mintaképet használhatunk, amelyeket egyenként kereshetünk a kamerából kapott képen.

A mintaillesztés egyszerű működésének közvetlen következménye, hogy objektumfelisme-



1. ábra. Mintaillesztés a normalizált korrelációs együttható módszerrel. Balra a forrás kép, jobbra az eredmény mátrix látható, mindkettőn be van jelölve a legjobb megtalált illeszkedés.

résre csak rendkívül speciális körülmények között alkalmazható kellő hatékonysággal. Mivel a képek statikusan pixelenként kerülnek összehasonlításra, ezért kritikus, hogy a kameraképen a felismerendő objektum közel ugyanolyan méretben, közel ugyanolyan nézetből, közel ugyanolyan orientációban szerepeljen, mint ahogyan a mintaképen. Ha ez ugyanis nem így történik, az tipikusan azt eredményezi, hogy a két képen nem a megfelelő pixelek kerülnek összehasonlításra, ekkor pedig a módszer eredménye jó eséllyel használhatatlan lesz. Ezen lehet úgy segíteni, hogy az objektumfelismerés során több mintaképet is felhasználunk, melyek az objektumot több méretben, esetleg több nézetből, több orientációban tartalmazzák, ez azonban jelentősen lassítaná a felismerés gyorsaságát. Ezek miatt leginkább csak az olyan speciális helyzetekben alkalmazható objektumfelismerésre, ahol biztosítani tudjuk, hogy a kameraképen az objektum éppen a megfelelő körülmények között szerepel. Általános tárgyaknál, épületeknél például nagyon kevés az esélye, hogy azokat éppen a megfelelő szögből, megfelelő orientációban próbálnánk felismerni, ezért ezeknek a felismerésére nem is igazán alkalmas. A módszer leginkább kétdimenziós, síkbeli tárgyak (mint például könyvborítók, filmplakátok) felismerésére lehet alkalmas, amennyiben biztosítani tudjuk, hogy azokat szemből, álló helyzetben, a megfelelő távolságból szeretnénk felismerni.

A mintaillesztés futási ideje és tárhely igénye abból adódóan, hogy a módszer a forrásképen egyenként keresi a mintaképeket, amelynek során pixelenként csúsztatja végig azokat egymáson, alapvetően az illeszkedéshez felhasznált képek (a mintakép/mintaképek, illetve a forráskép) méretével és számával vannak egyenes arányban. Ugyan a képek területeinek összehasonlításával kapcsolódóan elvégzett számítások egyszerűek és gyorsan elvégezhetők, azonban nagyméretű vagy nagyszámú képek esetén azokat rendkívül sokszor el kell végezni, ezáltal a futási idő is hosszúvá válhat.

3. Kaszkádosított osztályozó alapú objektum felismerés

A kaszkádosított osztályozó (Cascade Classifier) alapú felismerés manapság az egyik leghatékonyabb és legelterjedtebb módszer általános objektumok valós idejű detektálására. Jellemzője, hogy felhasználásával egy hosszabb tanítási folyamat után nehezen leírható, összetett, komplex objektumok felismerésére is nagy sebességgel lehetővé válhat. Ezek az objektumok szinte tetszőlegesek lehetnek, például arcok, emberek, épületek vagy éppen használati tárgyak, és megfelelő tanítás esetén változó felismerési körülmények között is robusztus maradhat a felismerés. A felismerés konkrét folyamata több jellegzetesség típus segítségével is történhet, ilyenek például a Haar-szerű jellegzetességek (Haar-like features), az LBP (Local Binary Patterns) jellegzetességek illetve a HOG (Histogram of Oriented Gradients) jellegzetességek. Ezekből az OpenCV eszközei elsősorban az előbbi kettőt támogatják, így ezen dolgozatban is ezekre térünk részletesebben ki.

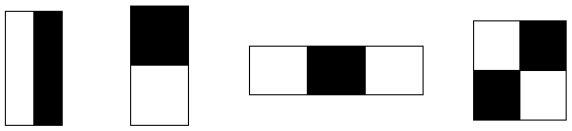
3.1. Haar-szerű jellegzetességek alapján történő objektum felismerés

A kaszkád osztályozó alapú objektum felismerésnél felhasznált a Haar-szerű jellegzetességek (Haar-like features) nevüket a Haar wavelet-ekhez való hasonlóságuk kapcsán kapták. A módszer eredeti változatát Paul Viola és Michael Jones publikálták 2001-ben. [4] A technikát elsősorban az arcfelismerés motiválta, és ezzel is demonstrálták, és ezzel a technikával sikerült az első valós idejű arcfelismerők egyikét is megvalósítani, amely annak idején egy Intel Pentium III processzoron 15 képkocka/másodperc sebességgel futott. A publikált objektum felismerési keretrendszer a Haar-szerű jellegzetességek és három főbb ötlet felhasználásával képes elérni a gyors és hatékony objektum felismerést. Az első ilyen egy új képábrázolás, az integrál kép, amely felhasználásával lehetővé válik a felismerő által felhasznált jellegzetességek rendkívül gyors kiértékelése. A második főbb közreműködés egy osztályozók létrehozására felhasznált módszer, amelynek során AdaBoost segítségével kis számú fontos vizuális jellegzetesség kerül kiválasztásra egy nagyobb készletből, így nagyon hatékony osztályozók létrehozására alkalmas. A harmadik fő hozzájárulás egy fokozatosan növekvő összetettségű osztályozók egy úgynevezett kaszkád szerkezetben való egyesítésére szolgáló módszer, amely lehetővé teszi a kép háttér területeinek gyors kiszűrését, ezáltal több számítást felhasználva az ígéretesnek tűnő, objektum-szerű régiókra.

3.1.1. Haar-szerű jellegzetességek

A technika nem közvetlenül kép intenzitás értékekkel dolgozik, hanem olyan jellegzetességek készletét használja fel, amelyek a Haar-wavelet-ekre emlékeztetnek, ezek a Haar-szerű jellegzetességek. A felismerési eljárás a képeket egyszerű jellegzetességek értékei alapján osztályozza. A jellegzetességek felhasználásának számos előnye van a hagyományos, a pixelek RGB értékeinek (intenzitásainak) közvetlen felhasználásához képest. Kizárólag pixeleken dolgozni

meglehetősen számításigényes folyamat, amelynél a jellegzetesség alapú rendszerek jelentősen gyorsabbak. Ezen túl a jellegzetességek rögzíthetik az olyan alkalmi ismereteket is, melyeket nehéz megtanulni véges számú tanítási adat alapján. A módszerben felhasznált egyszerű jellegzetességek a Haar wavelet-ekhez hasonlítanak. Egy Haar-szerű jellegzetesség szomszédos, azonos méretű és alakú téglalap alakú területeket vizsgál, az ilyen területeken kiszámítja a pixel intenzitások összegeit, és ezeknek az összegeknek a különbségeit vizsgálja (egymáshoz képest). Ezen különbségek alapján az egyes területeket (egymáshoz képest) sötétnek vagy világosnak nyilvánítja, mely alapján a kép részeit kategorizálni tudja. Az eredeti publikációban három fajta jellegzetesség szerepel: úgynevezett kettő-téglalap, három-téglalap és négy-téglalap jellegzetesség, melyek 2, 3 illetve 4 téglalap alakú területet vizsgálnak. A kettő-téglalap jellegzetesség két horizontálisan vagy vertikálisan szomszédos területen vizsgálja a pixelek összegeinek a különbségét. A három-téglalap jellegzetesség két szélső terület összegének és a köztük lévő, középső terület összegének a különbségét vizsgálja. A négy-téglalap két-két átlós terület összegei közti különbséget vizsgálja. Ezen jellegzetességek működésének szemléltetésére jó példa,



2. ábra. Haar-szerű jellegzetesség típusok

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

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

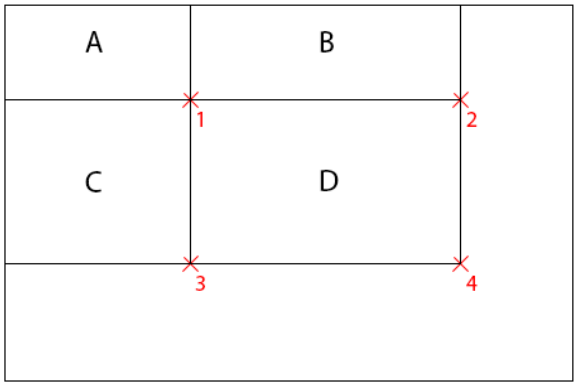
3.1.2. Integrál kép

A felismerési módszer rendkívüli sebessége nagyban az integrál képek felhasználásából adódik. Az integrál kép (II - integral image) egy köztes reprezentációja az eredeti képnek (I - image), amely a Haar-szerű jellegzetességek a gyors kiszámítását teszi lehetővé. Az integrál kép egy

kétdimenziós, a kép méretével megegyező mátrix, mely x, y mezőn az attól fentre és balra lévő pixelek összegét tartalmazza, azaz:

$$II(x,y) = \sum_{x' \leq x, y' \leq y} I(x', y') \tag{9}$$

Ez az eredeti kép pixelein mindössze egyszer végighaladva, pixelenként néhány művelettel kiszámítható, és miután ki lett számítva, segítségével a detektor által felhasznált Haar-szerű jellegzetességek bármely pozíción és méretben konstans időben kiszámíthatóvá válnak. Az eredeti képen bármely téglalap alapú területen lévő pixelek összege az integrál kép négy értékének kiolvasásával megkapható (3. ábra).



3. ábra. A D-vel jelölt téglalap területén lévő pixelek összegének kiszámítása az integrál kép segítségével. Ezen téglalap csúcsai 1, 2, 3, 4 pozíción vannak. Ekkor $II(1) = A$, $II(2) = A+B$, $II(3) = A+C$, $II(4) = A+B+C+D$. Ezekből $D = II(4)+II(1)-II(2)-II(3)$.

Ebből adódik, hogy két tetszőleges téglalap alapú terület összegei közti különbség 8 érték kiolvasásával kapható meg. Ez alapból is rendkívül alacsony mennyiség tekintve hogy tetszőlegesen sok pixel összegét néhány érték kiolvasásával megkaphatjuk, azonban a módszerben felhasznált jellegzetességekhez még ennyi érték kiolvasása sem szükséges. Mivel a Haar-szerű jellegzetességek egymás melletti téglalapokból állnak, melyeknek közös sarkaik vannak, így kevesebb érték kiolvasásával is megkaphatóak a kérdéses összegek. Így tehát a kettő-téglalap típusú jellegzetességeknél 6, a három-téglalap típusúaknál 8, a négy-téglalap típusúaknál pedig 9 érték kiolvasása szükséges.

Rainer Lienhart és Jochen Maydt a Haar-szerű jellegzetességek egy kibővített változatát publikálták 2002-ben. [5] Ebben a változatban az objektum felismerést olyan módon próbálták meg fejleszteni, hogy bevezették a jellegzetességek 45°-al elforgatott változatait. Ennek a kibővített változatnak a felhasználásával sok helyzetben pontosabban leírhatóak a felismerendő objektumok, így meglehetősen hatékonynak bizonyult. Az elforgatott jellegzetességek hatékony kiszámításához az integrál képek eredeti formájukban nem voltak felhasználhatóak, ezért ennek egy módosított változatát definiálták. Az integrál kép elforgatott változatában (RII

- rotated integral image) a mezők értéke az eredeti tőlük bal-felső régióba eső pixelek helyett annak a régiónak az óramutató járásával ellentétesen 45°-al elforgatott verziójába eső pixeleket összegzi. Képlettel:

$$RII(x,y) = \sum_{x' \leq x, x' \leq x-|y-y'|} I(x',y') \tag{10}$$

Ezt az elforgatott integrál képet az eredeti kép pixelein kétszer végighaladva lehet kiszámítani, tehát az eredeti integrál képhez képest körülbelül kétszer lassabban számolható, ez azonban még mindig nagyon gyorsnak mondható.

3.1.3. Tanulási algoritmus

Egy kép bármely részterületéhez kapcsolódó Haar-szerű jellegzetességek száma rendkívül magas, sokkal magasabb, mint ugyanezen a területen a pixelek száma. Emiatt a gyors osztályozás érdekében a tanuló folyamatnak az elérhető jellegzetességeknek nagyon nagy részét ki kell szűrnie, és alacsony számú, kritikus jellegzetességekre kell koncentrálnia. Kísérletek szerint ebből a rengeteg lehetséges jellegzetességből csak nagyon kevés kombinálható úgy, hogy egy hatékony osztályozót alkossanak. A tanulás során a fő feladat ezeknek a megtalálása. Az AdaBoost (Adaptive Boosting) egy gépi tanulás algoritmus, amely felhasználható más egyszerű (sokszor gyengének nevezett) tanuló algoritmusokkal kapcsolatban azok teljesítményének a javítására. Nevében az adaptív szó arra utal, hogy egymás után létrehozott osztályozók kerülnek tökéletesítésre előnyben részesítve azokat az eseteket, amelyeket előző osztályozók félreosztályoztak. A zajos és kiugró adatokra érzékeny, azonban kevésbé érzékeny a túltanítás (overfitting) problémára, mint más tanuló algoritmusok. Az algoritmus minden menetben generál és meghív egy új gyenge osztályozót. Minden meghíváskor frissül a súlyok eloszlása, amely a példák fontosságát jelzi az osztályozáshoz felhasznált adathalmazon. Az algoritmus minden menetben minden rosszul osztályozott példa súlyát megnöveli, és minden helyesen osztályozott példa súlyát csökkenti. Ez által az új osztályozó azokra a példákra koncentrál, amelyek korábban elkerülték a helyes osztályozást. A Haar-szerű jellegzetességek gyors kiválasztása az AdaBoost algoritmus egy szimpla módosításával történik: minden megkapott gyenge osztályozó csak egyetlen jellegzetességtől függhet. A gyenge tanulási algoritmus úgy lett tervezve, hogy azt az egy jellegzetességet válassza ki, amely a legjobban különválasztja a pozitív példákat a negatívoktól. A gyenge tanuló minden jellegzetességhez megállapítja az optimális határértéket úgy, hogy a lehető legkevesebb példát osztályozza félre. Emiatt a boosting folyamat minden olyan fokozata, amely egy új gyenge osztályozót választ ki, úgy tekinthető, mint egy jellegzetességet kiválasztó folyamat. A módszer tanítási folyamata során a jellegzetességek jelentős többsége elvetésre kerül, és tipikusan mindössze néhány száz vagy ezer jellegzetesség kerül megtartásra. Az AdaBoost hatékony tanuló algoritmust, és erős korlátokat biztosít a teljesítmény általánosítására.

3.1.4. Kaszkád osztályozók

A fokozatosan növekvő komplexitású osztályozók egy kaszkád szerkezetben való egyesítése elképesztő mértékben megnöveli a felismerés sebességét. A kaszkád tekinthető úgy, mint egy objektum-specifikus, figyelmet összpontosító mechanizmus, amely a korábbi megközelítésekkel ellentétben statisztikai garanciát biztosít arra, hogy a figyelmen kívül hagyott régiók valószínűtlen, hogy tartalmaznák a felismerendő objektumot. A figyelmet összpontosító megközelítések elképzelése szerint gyakran gyorsan megállapítható, hogy egy képen egy objektum hol fordulhat elő, és így az összetettebb számítások csak az ígéretes régiók számára vannak fenntartva. Az ilyen megközelítések legfőbb jellemzője a folyamat hamis negatív (false negative) aránya. Elvárható, hogy ez az érték minimális legyen, hogy ezáltal közel minden felismerendő objektum példány kiválasztásra kerüljön a folyamat során. A publikált objektum felismerési keretrendszerben ismertetett tanítási folyamat során létrehozott, rendkívül egyszerű és hatékony osztályozó egy ilyen figyelmet összpontosító módszerként használható fel. Ennek a szűrőnek a hatása, hogy kevesebb, mint felére csökkenti azoknak a helyeknek számát, ahol a végső detektort ki kell értékelni.

A kaszkád felépítése azt fejezi ki, hogy bármely képen a részterületek túlnyomó többsége negatív, azaz nem tartalmazza az objektumot. A kaszkád így próbál minél több negatívot minél korábbi szinten elvetni. A pozitív példák a kaszkád minden osztályozóján átjutnak, ám ez egy rendkívül ritka eset a többihez viszonyítva. Ahogyan egy döntési fában, az egymás utáni osztályozók olyan példák alapján vannak betanítva, amik az előző szinteken átjutottak. Emiatt a későbbi szinteken lévő osztályozók egyre nehezebb feladattal néznek szembe, és azok a példák amik mélyebbre jutnak a kaszkádban, "nehezebbek" mint más példák. A nehezebb példák, amelyek eljutnak a mélyebb szintű osztályozókig, nagyobb hamis pozitív arányt eredményeznek.

3.1.5. A felismerési folyamat

A felismerés során egy adott méretű cél ablak kerül mozgatásra a bemeneti képen, és a Haar-szerű jellegzetesség az ablak minden helyén kiszámításra kerül. Annak érdekében, hogy több méretben is felismerhessük az objektumot, az osztályozó könnyen átméretezhetőre lett tervezve. Ahhoz tehát, hogy egy ismeretlen méretű objektumot is felismerjen a folyamat, a pásztázást iteratívan többször el kell végezni az osztályozó különböző méretű változataival. A felismerés során a képen azok a részterületek, amelyek nem kerültek elvetésre a kaszkád első osztályozója által, olyan további osztályozók sorozatával kerülnek feldolgozásra, amelyek fokozatosan egyre összetettebbek. Ha a kaszkádban bármely osztályozó elveti a részterületet, a feldolgozás véget ér azzal a következtetéssel, hogy az objektum nem szerepel a képen. Ha a részterület sikeresen keresztüljut minden osztályozón, az objektum felismerése sikeresnek minősülhet. A kaszkádosított felismerési folyamat felépítése így lényegében egy degenerált döntési fához hasonlítható, ahol minden szülő csomóponthoz csak egy gyerek csomópont tartozik.

3.1.6. Az OpenCV implementáció

Az OpenCV-ben a kaszkád osztályozók alapján való felismerés elvégzéséhez kapcsolódó funkciók a *CascadeClassifier* nevű osztályban találhatóak meg [6]. A *load* függvénnyel tölthetjük be az OpenCV eszközeivel korábban betanított XML formátumú kaszkád osztályozót, ez lehet Haar-szerű vagy LBP jellegzetesség típusú is. Ez után a detektálást a *detectMultiScale* függvénnyel végezhetjük el. A függvény paraméterei a bemeneti kép, a felismert objektumok helyét tartalmazó téglalapok listája (kimeneti paraméter), a skálázási faktor, a szomszédok minimális száma, egy flags mező, és a felismerni kívánt objektum minimális és maximális mérete. Az objektum a megadott minimális és maximális méret között kerül keresésre, a keresett objektum mérete minden iterációban a skálázási faktorral változik meg. A szomszédok minimális száma azt jelöli, hogy az egyes téglalap jelöltek hány szomszéddal kell hogy rendelkezzenek ahhoz, hogy ne kerüljenek elvetésre. Egy nagyobb érték itt nagyobb megbízhatóságot, de kisebb pontosságot eredményez. A flags mezőben igény esetén egyéb beállításokat is végezhetünk.

Osztályozók létrehozására az OpenCV az *opencv_traincascade* nevű eszközzel rendelkezik, amely a bemeneti képek és a megadott paraméterek alapján el tudja végezni egy kaszkád osztályozó betanítását. Ennek a programnak és további, a betanítás során felhasználható eszközöknek a bővebb ismertetésével a dolgozat egy későbbi részében foglalkozunk (5. fejezet).

3.1.7. A módszer tulajdonságai

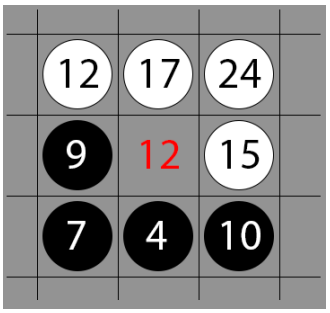
A módszerről elmondható, hogy megfelelő betanítás esetén rendkívül sok fajta objektum felismerésére lehet változó körülmények között is stabilan alkalmas. Az integrál képek és egyéb hatékony megoldásoknak köszönhetően a futási idő kifejezetten gyors, okostelefonokon is alkalmas valós idejű alkalmazásokban való felhasználásra. A módszer tárhelyigénye a felismerni kívánt objektumokat leíró kaszkád osztályozók méretéből adódik, amelyek tipikusan nem kifejezetten nagy méretűek.

3.2. Local Binary Patterns jellegzetességek

A Local Binary Patterns (LBP) [7] a Haar-szerű jellegzetességek (lásd: 3.1. fejezet) mellett egy másik gépi látásban való osztályozásra felhasználható jellegzetesség típus. Elsősorban kétdimenziós textúrák osztályozására tervezték. Az eredeti változata leginkább lokális információk leírására alkalmas, felhasználásával egy képnek leginkább a mikrostruktúrái írhatóak le hatékonyan. Egyik nagy előnye, hogy rendkívül hatékonyan számítható ki, ami a valós idejű felhasználásokban kritikus jelentőségű. Másik nagy előnye, hogy lokális jellegéből adódóan meglehetősen érzéketlen a monoton szürkeárnyalati változásokra, így például a fényviszonyokra is. Az LBP jellegzetességeket a Haar-szerű jellegzetességekhez nagyon hasonlóan felhasználhatjuk kaszkád osztályozók létrehozásához, melynek segítségével rendkívül hatékony objektum felismerés érhető el.

3.2.1. Az eredeti LBP operátor

Az LBP fő ötlete a kép lokális struktúráinak összegzése a pixelek környezetükkel történő összehasonlítása alapján. Kiszámítása a legegyszerűbb megközelítésben a pixelek egy fix, például 3×3 pixel méretű környezete alapján történik, a következőkben ismertetett módon. A vizsgált ablakon belül minden pixelt hasonlítsunk össze környezetében lévő másik pixelekkel. Ez a 3×3 pixel környezetes példánál a pixelt körülvevő 8 másik pixel. Ahol a középső pixel értéke (intenzitása) nagyobb vagy egyenlő, mint a szomszédjáé, oda rendeljünk 1-et, egyébként pedig 0-át. Ez bináris értékeket fog rendelni minden pixelhez az ablakban (példánkban 8 darabot), amely egy pixel esetében azt fejezi ki, hogy a pixelnél mely szomszédjai nagyobbak, és melyek kisebbek. A pixelekhez hozzárendelt értékekre címke néven is hivatkoznak. Számoljuk össze, hogy az ablakban az összes (példánkban $2^8 = 256$) fajta lehetséges címke közül melyik hányszor szerepel, és ebből készítsünk egy hisztogramot. Ezzel megkaptuk a keresett jellegzetesség vektort: a vizsgált ablak LBP leíróját. Objektum felismerés céljából a megkapott leírók feldolgozhatóak egy gépi tanulás algoritmussal, hogy képek osztályozására fel lehessen őket használni.



4. ábra. Az LBP működése. A sötétebb pixelű szomszédokhoz 0-át rendel, a többihez 1-et.

3.2.2. Kiterjesztett LBP variánsok

Az eredeti LBP hatékonyan jellemzi a képeket, és nagyon finom részletek leírására kiválóan alkalmas, azonban a pixelek fix méretű szomszédosságával való összehasonlítása miatt a változó méretű részletek leírására már nem alkalmas, sok célra túlságosan lokális ahhoz, hogy robusztus működést biztosítson. Emiatt a módszert számos módon kiterjesztették, elsősorban azért, hogy alkalmasabbá váljon bizonyos célok megvalósításához. Így olyan változatai is születtek, ahol például a pixelek változó méretű szomszédságukkal, tetszőleges számú szomszédukkal kerültek összehasonlításra [8].

Egy másik népszerű kiterjesztése az operátornak az úgynevezett uniform minták használata, amely segítségével lerövidíthető a jellegzetesség vektorok hossza és egy forgatás invariáns leíró implementálható [9]. A módszer alapötletét az adta, hogy bizonyos bináris minták jelentősen gyakrabban fordulnak elő, mint mások. Egy LBP minta uniform, amennyiben a bináris leíróit körbejárva a 0-ások és 1-esek közti átmenetek száma legfeljebb 2. Az 11111111 (0 át-

menet) és 00001110 (2 átmenet) például uniformis minta, a 01101100 (4 átmenet) azonban nem az. A módszerben a feldolgozás során az uniform, forgatás invariáns minták külön egyedi címkékbe sorolódnak, a nem uniform minták azonban egy közös, "egyéb" címke alá kerülnek csoportosításra.

Az arcfelismerés, illetve általánosabban objektum felismerés felhasználásában az eredeti LBP egyik legnagyobb hiányossága, hogy a térbeli információk leírását nem igazán támogatja, és a pixelek közvetlen összehasonlításából adódóan meglehetősen érzékeny a zajokra. Ezen okok miatt definiálták egy módosított változatát, Multiscale Block Local Binary Pattern (MB-LBP) néven [10]. Az MB-LBP-ben a számítások nem az egyes pixelek alapján történnek, hanem részterületek átlagos értékei alapján. Ez több előnnyel is jár. Egyrészt robusztusabb, másrészt nem csak a kép mikrostruktúráinak, hanem makrostruktúráinak a leírására is alkalmas, harmadrészt rendkívül hatékonyan kiszámítható integrál képek segítségével. A módszert ismertető publikáció a módosított jellegzetességekből való hatékony osztályozó létrehozásához javasol AdaBoost alapú algoritmust is. A módszert a Haar-szerű jellegzetességekhez hasonlóan elsősorban az arcfelismerés motiválta. Kísérletek alátámasztják, hogy az MB-LBP jellegzetességeket felhasználó arcfelismerés jelentősen jobb eredményekre képes, mint a legtöbb másik LBP alapú arcfelismerő algoritmus. Az OpenCV is ezt a módosított változatú LBP-t támogatja kaszkád osztályozók jellegzetesség típusaként.

3.2.3. OpenCV implementáció

Az OpenCV az LBP jellegzetesség típusú kaszkád osztályozókat ugyanazokkal a tanítási és felismerési eszközökkel támogatja, mint a Haar-szerű jellegzetesség típust felhasználó kaszkád osztályozókat (lásd: 3.1.6. fejezet).

3.2.4. A módszer tulajdonságai

Az LBP-t felhasználó kaszkád osztályozók tulajdonságai nagyon hasonlítanak a Haar-szerű jellegzetesség alapú kaszkád osztályozók tulajdonságaihoz (3.1.7. fejezet). Az LBP jellegzetességek a Haar-szerű jellegzetességekkel ellentétben egész számokból állnak, emiatt objektum felismerésre való felhasználás során a tanítási és felismerési fázis is többszörösen gyorsabban elvégezhető a segítségével. Ez még tovább növeli az egyébként is rendkívül hatékony Haar-szerű jellegzetesség alapú módszer teljesítményét, ami hordozható eszközökön történő objektum felismerés során meghatározó tényező lehet. A felismerés minősége mindkét jellegzetesség esetében leginkább a tanítástól függ: elsősorban a tanításhoz felhasznált képektől, továbbá a tanítási paraméterektől. Megfelelő tanítás esetén alig rosszabb, vagy közel ugyanolyan minőségű felismerés érhető el megfelelő variánsú LBP jellegzetességeket felhasználó osztályozók esetén, mint Haar-szerű jellegzetességet alkalmazó osztályozókkal.

4. Jellegzetesség megfeleltetés alapú objektum felismerés

Képek jellegzetességeit közvetlenül is felhasználhatjuk objektumok felismerésére a jellegzetesség megfeleltetés (Feature Matching) módszerével. A módszer működési elve, hogy jellegzetes pontokat keres a bemeneti képeken, és az egymáshoz hasonló jellegzetes pontokat megpróbálja feleltetni egymásnak. Ha kellően sok pontot sikerült megfeleltetni a képek között egymásnak, az utalhat arra, hogy a képeken ugyanaz az objektum szerepel. Ezt tovább valószínűsítheti, ha a megfeleltetett pontok elhelyezkedése is konzisztens.

4.1. Jellegzetesség leírók

A jellegzetességek a képeken bizonyos szerkezeteket, alakokat jelölnek. Ezek lehetnek egyszerűbb dolgok is, mint például pontok vagy élek, de akár összetettebb szerkezetek is, mint például objektumok. Egy adott jellegzetesség konkrét ábrázolása számos különböző módon történhet. Nagyon egyszerű közelítésben jelölhetjük akár egy egyszerű logikai értékkel, hogy a kép egy pontján épp van-e él vagy nincs. Egy másik megközelítésben leírhatjuk a jellegzetességet például a pont körüli régió átlagos színe alapján, vagy a pont környezetéhez viszonyított kontrasztja alapján. A lehetőségek száma közel korlátlan, és ezeket akár kombinálhatjuk is. Egy jellegzetesség konkrét reprezentációját a jellegzetesség leírójának hívjuk. Időnként nem elegendő csak egy fajta jellegzetesség leíró ahhoz, hogy kellő információnk legyen a képről, így több jellegzetesség leíró is tartozhat a kép egy-egy pontjához. Ezeknek az információit gyakran egyetlen vektorba szervezik, melyre jellegzetesség vektor néven hivatkoznak. Egy leíró lehet alacsonyabb szintű (amely például csak egyszerű élek jelenlétére utal), vagy összetettebb (amely például a pont nagyobb környezetét is pontosan jellemzi színek, alakok, vagy több egyéb szempontból). Kritikus, hogy a megoldandó feladatunkhoz megfelelő leírót alkalmazzunk. Bizonyos típusú feladatokhoz elég alacsony szintű leírók alkalmazása, más feladatok viszont magasabb szintű leírókat követelnek. Az alacsonyabb szintű leírókat általában könnyű kiszámítani és ábrázolni is, míg az összetettebbek több adattal dolgoznak, számításigényesebbek, és nehezebb őket ábrázolni is.

4.2. A jellegzetesség megfeleltetés folyamata

A jellegzetesség leírók megfeleltetésének főbb lépései az alábbiak:

1. A képeken a kiugró (várhatóan érdekes) pontok, azaz kulcspontok megtalálása egy jellegzetesség detektor felhasználásával
2. A kulcspontokhoz jellegzetesség leírók rendelése egy jellegzetesség kivonatoló felhasználásával
3. A jellegzetesség leírók egymásnak való megfeleltetése egy jellegzetesség megfeleltető felhasználásával

Bár vannak módszerek, amelyek magukba foglalják mindkettőt, fontos megkülönböztetnünk a jellegzetességek detektálásának és jellegzetességek kivonatolásának a folyamatát. Detektálás során valamilyen módszer szerint mindössze annyi a feladat, hogy várhatóan érdekes, a környezetükből kiugró pontokat találjunk meg a képen, melyek majd alkalmasak lesznek rá, hogy a képen lévő jellegzetességeket jól jellemezzék. Az így megtalált pontok leírókkal való jellemzése egy teljesen különálló feladat, és kritikus, hogy olyan leírókat generáljunk ezekhez a pontokhoz, melyek jól jellemzik ezeket a pontokat. Természetesen a detektálás során a pontok kiválasztása is kritikus, hiszen ha nem a megfelelő pontokat választottunk ki a képen, azokat hiába tudjuk jól jellemezni, ha például nem eléggé kiállóak és túlságosan általánosak, nem fogjuk tudni azok alapján a képen szereplő objektumokat egymásnak megfeleltetni. Mind a jellegzetességek detektálása, mind pedig a leírók kivonatolása számos különböző módszerrel történhet, melyek különböző tulajdonságokkal rendelkeznek. A legfontosabb ilyen tulajdonságok többek között a futási idő, a találatok száma és a különböző (például skálázási, forgatási, affin) invarianciák.

Amennyiben már rendelkezünk az illesztendő képek jellegzetességeinek leíróival, a következő feladat azoknak az egymással való megfeleltetése. A cél itt a kellően hasonló leírókkal rendelkező jellegzetességek megtalálása. A jellegzetességek megfeleltetésének elvégzésére két gyakori módszer a brute force alapú, és egy másik, approximációs megoldás. A brute force alapú megfeleltető működése nagyon egyszerű: az egyik halmazba tartozó leírókhoz úgy találja meg a legjobb illeszkedéseket a másik halmaz leírói közül, hogy egyszerűen végignézi az összeset. Biztosan optimális eredményt ad, azonban nagy számú leírók megfeleltetésénél lassú működésű lehet. Egy másik megoldás a Fast Approximate Nearest Neighbors elnevezésű approximációs módszer [11]. A megoldás a Fast Library for Approximate Nearest Neighbors nevű könyvtárat használja, melyre gyakran csak FLANN néven hivatkoznak. A könyvtár olyan algoritmusok egy gyűjteményét tartalmazza, amelyek nagy adathalmazokon és sokdimenziós jellegzetességek között a legközelebbi szomszéd gyors keresése problémára vannak optimalizálva. Gyors működésű, de nem biztos hogy mindig a legjobb eredményt adja. Akkor lehet érdemes használni, ha kellően sok leírót akarunk egymással megfeleltetni.

4.3. Az OpenCV implementáció

Az OpenCV-ben a leírók többdimenziós tér vektoraiként vannak ábrázolva. A könyvtárban két főbb típusú leíró van: lebegőpontos és bináris. Az OpenCV közös interfészeket biztosít a folyamat minden lényeges lépéséhez: a jellegzetesség detektáláshoz, a jellegzetesség kivonatolóhoz és a jellegzetesség megfeleltetéshez is.

4.3.1. A FeatureDetector interfész

A FeatureDetector az OpenCV jellegzetesség detektorainak közös interfésze [12]. Lényegi funkciója a detect függvény, amely elvégzi a jellegzetességek keresését a megadott képen vagy

képeken. A függvény kimenete egy KeyPoint típusú adatokat tartalmazó halmaz. A KeyPoint egy speciális adathalmaz a megtalált pontok (ezen fázisban még nem jellegzetességek) leírásához. Mezői többek között a pontok koordinátáinak, méretének, orientációjának és illeszkedésének mértékének ábrázolására biztosítanak lehetőséget. A függvény opcionálisan maszkok megadását is lehetővé teszi. A felismerés elvégzéséhez a könyvtár a következő típusú detektorokat támogatja: FAST, STAR, SIFT, SURF, ORB, BRISK, MSER, Good Features To Track, Good Features To Track Harris detektorral, Dense, SimpleBlob.

4.3.2. A DescriptorExtractor interfész

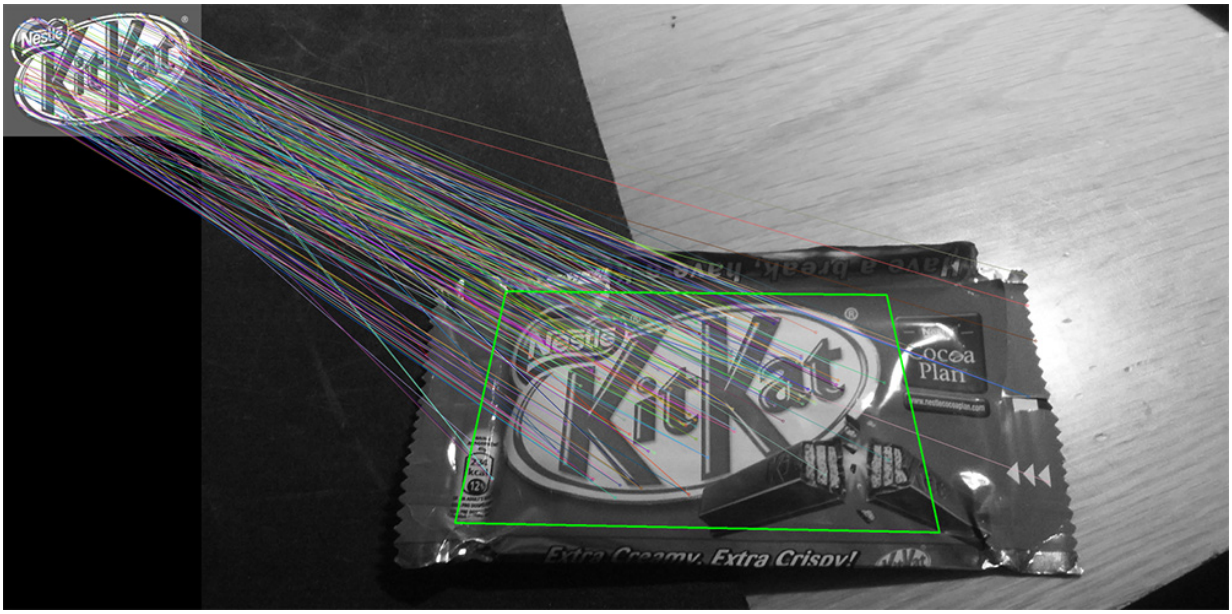
A könyvtár leírók kivonatolásához biztosított interfésze a DescriptorExtractor [13]. Legfontosabb funkciója a detect metódus, amely a megadott képen (vagy képeken) a KeyPoint típusokként megadott kulcspontok helyein kiszámítja a jellegzetességek leíróit. A függvény a leírókat egy mátrix formájában adja vissza, amelynek minden sora egy kulcsponthoz tartozó leíró vektora. A metódus a kulcspontok listáját is megváltoztathatja, ennek fő oka, hogy a leírók kiszámítása nem minden esetben lehetséges, az ilyen esetekben a kulcspontok eltávolításra kerülnek a halmazból. Bizonyos esetekben új kulcspontok is kerülhetnek a halmazba. A könyvtár által támogatott jellegzetesség kivonatolók a következők: SIFT, SURF, ORB, BRISK, BRIEF.

4.3.3. A DescriptorMatcher interfész

Az OpenCV interfésze a jellegzetességek leíróinak megfeleltetésének elvégzéséhez a DescriptorMatcher [14]. Lényegi metódusa a match és ennek variánsai, melyek a leírók egyik halmazában lévő elemekhez megtalálják a másik halmazból a legközelebbi leírókat. A függvény paraméterei az egymásnak megfeleltetendő leírók halmazai, a találatok listája (kimeneti paraméter) és egy opcionális maszk paraméter. A találatokat a függvény DMatch adattípusként adja vissza, amely tartalmazza a találathoz kapcsolódó két leíró azonosítóját, a kép azonosítóját és a távolságot, amely a találat minőségére utal. A függvénynek a két másik variánsa a knnMatch és a radiusMatch. A knnMatch a leírók egyik halmazában lévő leírókhoz a másikból a legjobb k-t találja meg, a radiusMatch csak a megadott távolságon belül lévő találatokat adja vissza. Az OpenCV-ben támogatott jellegzetességmegfeleltetők a BFMatcher és a FlannBasedMatcher. Az egyes leírók megfeleltetése során nem mindegy, hogy a köztük lévő távolságot (különbséget) milyen módszerrel számítjuk ki, főleg mivel a leírók lehetnek bináris és valós típusúak. A módszert a BFMatcher használatakor annak konstruktorának a normType paraméterében állíthatjuk be. Támogatott paraméterei a valós típusú (például SIFT, SURF) leírókhoz használható 1-normán és 2-normán alapuló távolság (azaz Manhattan távolság illetve euklideszi távolság), és a bináris típusú (például ORB, BRISK, BRIEF) leírókhoz használható Hamming távolság és annak egy módosított változata. A FlannBasedMatcher alapértelmezetten 2-normán alapuló távolságot használ, amelyet más típusú indexek használatával tudunk módosítani. Bináris leírókhoz például használhatjuk az LshIndexParams indexet.

4.4. Objektumfelismerés a módszer felhasználásával

A fent ismertetett módszert felhasználhatjuk objektumok felismerésére, ha bemenetének a felismerni kívánt objektum és a kamera képeit adjuk meg. A módszer kimenetei egymásnak megfeleltetett leíró párok, és az illeszkedés mértékére utaló távolságok. Az, hogy ezeket az információkat miként használjuk fel objektumok felismerésére, nem triviális feladat. Első megközelítésben kijelölhetünk egy határ távolságot, amely alatt hasonlónak nyilvánítjuk a megfeleltetett leírókat, a többit pedig eldobjuk. Ezután megszámolhatjuk az hasonló pontok számát, és azt összehasonlíthatjuk egy előre kijelölt (vagy például dinamikusan az átlagos távolságokhoz viszonyított) határértékkel, amely felett a felismerést sikeresnek, egyébként sikertelennek nyilvánítjuk. Kifinomultabb módszerek minden leíróhoz a 2 legjobban illeszkedő leírót keresik meg (az OpenCV-ben a knnMatch függvénnyel), és ha azoknak a távolsága túl közel vannak egymáshoz, azt feltételezik, hogy a megtalált illeszkedés nem megbízható (ha két ennyire közeli illeszkedést talált az algoritmus, akkor jelentős eséllyel nem a legjobbat találta meg), és eldobják azt. A megtalált illeszkedések pozícióinak konzisztensségét is számos módon felhasználhatjuk a felismerés megbízhatóságának javítására, hogy a teljesen különböző objektumokból származó megfeleltetéseket kiszűrjük. Ha síkbeli objektumot keresünk, és kellően sok, kellően konzisztens megfeleltetésünk van az objektumok között, az OpenCV findHomography [15] függvénye segítségével az azok közti perspektív transzformációt is kiszámíthatjuk (5. ábra).



5. ábra. Jellegzetesség megfeleltetés SURF detektorral és kivonatolóval, FLANN megfeleltetővel, perspektív megfeleltetés ábrázolásával.

4.4.1. Objektum felismerés során a módszer tulajdonságai

A módszer segítségével megvalósított objektum felismerés hatékonyságát és sebességét döntő mértékben meghatározzák a folyamat egyes fázisai (detektálás, leíró készítés, megfelelte-

tés) során felhasznált konkrét módszerek tulajdonságai és a folyamat kimenetének (egymásnak megfeleltetett leíró párok) konkrét felhasználási módja is. Amit általánosan el lehet mondani, hogy módszer leginkább a fixen sok jellegzetes, jól leírható ponttal rendelkező objektumok esetén működhet jól. Ilyenek lehetnek leginkább síkbeli jellegzetes objektumok: például könyvborítók, moziplakátok, márkalogók. Kevésbé működhet jól viszont általánosabb, változó tulajdonságú tárgyak és objektumok esetén, például erősen térbeli tárgyak esetén amelyek más szögekből nézve más jellegzetességeket mutatnak.

Megfelelő módszerek felhasználásával a felismerés futási sebessége kielégítő lehet okostelefonokon való közel valós idejű felhasználáshoz is. A tárhely igény a felismerendő objektumokat tartalmazó adathalmaztól függ: amennyiben azokat képek formájában tároljuk akkor azok mennyiségétől és méretétől, amennyiben pedig közvetlenül a leíróirat tároljuk akkor pedig azoknak a típusától is.

5. Kaszkádosított osztályozó alapú detektorok készítése

A főbb módszerek elméleti ismertetése után most áttérünk a objektumok általános felismerésére általunk legalkalmasabbnak ítélt módszer gyakorlati áttekintésére. Először részletesen ismertetjük a módszerhez szükséges tanítási folyamatot, majd bemutatjuk a tesztkörnyezetünket, végül pedig ismertetjük az elvégzett méréseink eredményeit.

5.1. Tanítási környezet elkészítése PC-n

A tesztkörnyezet használatát Windows 7 platformon mutatjuk be. Ahhoz, hogy a tanítást megkezdhessük, szükségünk lesz magára a tanító programra és ezen kívül néhány kiegészítőre, melyek elengedhetetlenek az eredményes detektorkészítéshez. Ezek a következők:

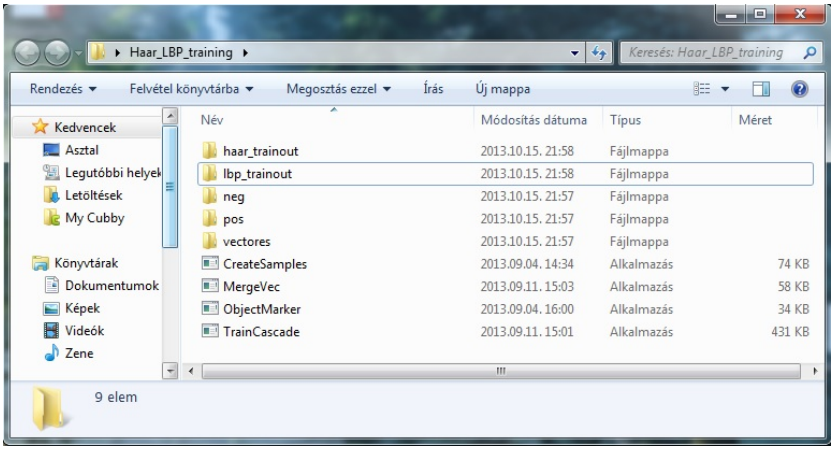
- TrainCascade.exe: Ez maga a detektortanító.
- CreateSamples.exe: Ezzel a programmal tudjuk előállítani azt a .vec kiterjesztésű fájlt, mely tartalmazza a pozitív képeket. A TrainCascade.exe kizárólag ilyen kiterjesztésű fájlokat tud pozitív tanítóhalmazként kezelni.
- ObjectMarker.exe: A képek annotálását végezhetjük el vele.
- MergeVec.exe: Ezzel több .vec fájlt tudunk összefűzni eggyé.

A fentiek közül az OpenCV library a TrainCascade és a CreateSamples forrását tartalmazza. Ezek használatához először le kell töltenünk, majd megfelelően telepítenünk az OpenCV-t. Erről bővebb információ a könyvtár hivatalos honlapján található. [16] Készítsük el a két programot a könyvtárban található megfelelő források lefordításával. A létrehozott .exe fájlokat rakjuk egymás mellé egy külön mappába. Azért érdemes egy helyre rakni ezeket a programokat, mert szorosan kapcsolódnak egymáshoz, az egyik outputja a másik inputjaként szolgál.

Az OpenCV nem tartalmaz beépített annotálót, de különböző weboldalakon találhatunk erre a célra nyílt forráskódú programot. Az általunk használt annotáló program forrása a [17] hivatkozáson érhető el. A letöltött .h és .cpp fájlt az előzőekhez hasonlóan lefordítható, majd az .exe-t tegyük az előző kettő mellé.

A MergeVec.exe létrehozásához a forrásfájlokat a [18] hivatkozás weboldalán érhetjük el. A forrásfájl lefordításának részletezése a megadott hivatkozáson olvasható. Ezt a programot is helyezzük el a többi mellett.

Miután elkészült mind a négy .exe, hozzunk létre a mappában további öt mappát: egyet a negatív képeket tartalmazó tanítóhalmaz számára, egy másikat a pozitív tanítóhalmazok részére, egyet az elkészített haar detektorok számára, egyet az lbp detektortanítások kimenetének és egyet a generált .vec fájlok számára. Ezekre azért van szükség, hogy a későbbiekben, mikor már nagy mennyiségű adattal dolgozunk, környezetünk továbbra is átlátható maradjon.



6. ábra. A javasolt könyvtárszerkezet

5.2. A tanításhoz használt programok bemutatása

A tanítási folyamat a következő lépésekből áll:

1. Pozitív és negatív képek készítése
2. Képek annotálása és/vagy szoftveres pozitív kép generálás
3. Több .vec fájl esetén ezek összefűzése
4. Tanítás

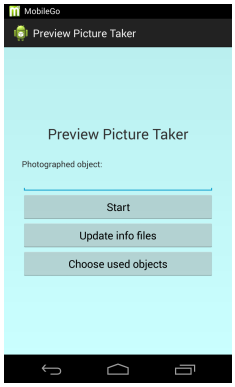
A programok használata ezen sorrend alapján kerül bemutatásra.

A tanítást támogató programok közül a az Android oldalon futó PreviewPictureTaker saját készítésű.

5.2.1. PreviewPictureTaker képkészítő Android alkalmazás

Mivel a teszteket telefon oldali preview képen futtatjuk, így a tanítóhalmazt is ilyen felbontású képek képezik, továbbá felesleges nagyméretű képeket erre a célra használni, mert azok növelik a futási időt. Ez az alkalmazás preview méretű képeket készít. Erre a saját készítésű programra azért van szükség, hogy a készített képeket struktúráltan, a PC oldali környezethez illeszkedően tudjunk elkészíteni.

Az applikáció menüjében beírhatjuk, hogy milyen objektumról készítünk képet, vagy ha egy meglévő képhalmazt szeretnénk bővíteni, az előzmények közül kiválaszthatjuk azt. Ha újat írunk be, a megadott néven létrehoz egy mappát a telefon SD kártyáján a PhotosForResearch mappában, és ide rakja számozva a készített képeket. Ha az előzményekből választottunk, akkor az előzmény mappájába helyezi az újabb fotókat. Képkészítés során a preview-ra koppintva tudunk képet készíteni, illetve lehetőség van automatizált képkészítésre, ilyenkor közelítőleg



(a) A menü

(b) Képkészítés közben

7. ábra. A PreviewPictureTaker applikáció

három készül másodpercenként. Mivel egy adott objektumról sok nézőpontból kell kép, ez a funkció jelentősen felgyorsítja a folyamatot.

Miután végeztünk a fotózással, elkészíttetjük applikációnkkal az adott képhalmazt leíró info.txt fájlt, mely tartalmazza a fájlneveket. (Erre a számítógép oldali programoknak szüksége van.) A mappát áthelyezhetjük a PC-n lévő környezet megfelelő mappájába (ha pozitív képeket készítettünk, akkor a *pos* mappába, ha negatívat, akkor a *neg*-be).

5.2.2. Annotálás ObjectMarker segítségével

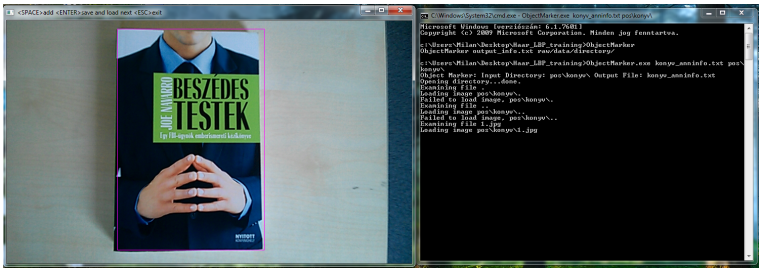
Amikor megvannak a pozitív képeink, ezzel a programmal lehet viszonylag gyorsan minden egyes képen megjelölni a betanítandó objektumot, mert megjelölés után azonnal betölti a következő annotálandó képet. Ez egy fáradtságos és nagyon monoton művelet, a betanításhoz több száz kép szükséges. Nem automatizálható, hisz ahhoz kész detektorra lenne szükségünk, és mi épp azt akarjuk elkészíteni.

Az annotálást tartalmazó info.txt fájlt mindenképp az .exe fájlok mellé tegyük, különben a következő lépésben a CreateSamples nem fog működni. Majd a CreateSamples lefuttatása után áthelyezhető a megfelelő helyre. További nagy hátránya ennek a programnak, hogy bal alsó sarokból kezdve a jobb felső sarokba húzva az annotáló téglalapot, illetve a jobb felső sarokból a bal alsóba húzva nem menti az annotálást és erről nem figyelmezteti a felhasználót! Ha nagyon sok tanítást akarunk elvégezni, érdemes ehelyett a külső forrásból származó program helyett egy sajátot írni, mely jobban illeszkedik a környezetbe. Azért nem írtuk még meg a sajátot, mert először ki akartuk mérni ezen felismerési módszernek határait, hogy alkalmas-e objektumfelismerési feladatokra.

A program használatára részletesebben a függelékben térünk ki.

5.2.3. A minták elkészítése CreateSamples használatával

A CreateSamples három funkciójú, ezeket a megfelelő parancssori argumentumok átadásával tudjuk elérni.



8. ábra. Annotáló program használat közben

Első funkciója .vec fájl létrehozása az annotálásokat tartalmazó .txt fájl alapján. Átméretezi és szürkeárnyalatossá teszi a megjelölt képrészletet, hogy a TrainCascade bemeneteként tudjon szolgálni.

Második funkciója egyetlen pozitív képből több kép generálása szoftveres elforgatással, megvilágítással, ezzel mesterségesen megközelítve a különböző szögből és fényviszonyból készült képeket. Amikor egyszerűbb logóról készítünk detektort, ezzel is egész jó eredményt érhetünk el. Bizonyos esetekben az a legeredményesebb, ha van egy .vec fájlunk, mely annotált képek alapján készült és van egy, melyet ezzel a szoftverrel generáltunk és a kettőt MergeVec-el egyesítjük. Erre a mérési eredmények bemutatása kapcsán térünk ki bővebben.

A harmadik funkció lehetővé teszi egy meglévő .vec fájl tartalmának megtekintését.

A program használatának részletes bemutatása a függelékben található.

5.2.4. CreateSamples kimeneti fájlok összefűzése MergeVec programmal

A CreateSamples nem támogatja azt, hogy több bemeneti képből generáljunk szoftveres forgatással és megvilágítással tanító halmazt. Ez azért probléma, mert egy jellegzetes logó egész jól betanítható úgy, hogy kiindulunk például 5 forrásképből, ezek mindegyikéből generálunk 200-at, majd végül ezeket adjuk át a TrainCascade-nek. Ám a TrainCascade csak egyetlen .vec fájlt kaphat paraméterként, így a generált 5 .vec fájlt egyesíteni kell. Egy másik eset, mikor a tanítóhalmaz egy részét annotált forrásból szeretnék felhasználni másikat pedig szoftveresen generálni. Ez esetben is .vec fájlok egyesítése szükséges. Ezt végzi el a MergeVec program.

A használandó parancssori argumentumok leírása a függelékben található.

5.2.5. Tanítás TrainCascade használatával

Miután a fentebb írt programok segítségével előkészítettük a tanítóhalmazt, megkezdődhet a detektortanítás. Tanítás végén egy .xml fájt kapunk, melyet közvetlenül fel tudunk használni programunkban. A TrainCascade szintén parancssori argumentumok segítségével használható. Paraméterezése összetett, az egyes paraméterek megértéséhez az elméleti háttér ismerete elengedhetetlen (3. fejezet). A használandó paraméterek részletes bemutatása a függelékben található.

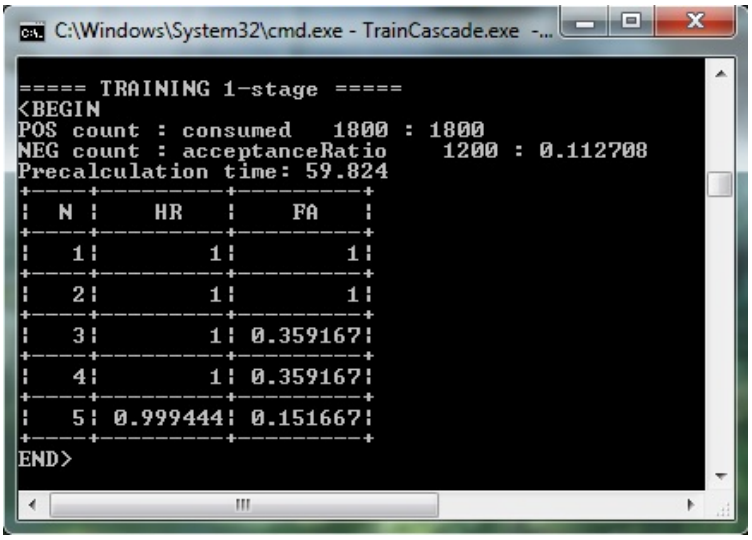
A tanítás során minden egyes befejezett szint adatai mentésre kerülnek. Így ha félbeszakad

a folyamat, legutóbbi kész szinttől folytatható. Ehhez ugyanazokkal a paraméterekkel kell elindítani, mint előtte, így érdemes mindig a tanítás megkezdésekor az átadott argumentumokat menteni.

Mivel a tanítás nagyon erőforrás és időigényes, előfordul, hogy a végeredmény napok, sőt akár hetek múlva készül el. Naponta mindenképp érdemes a részeredmények alapján összerakni egy kész kaszkádot, és kipróbálni. Ha túl sok a hamis felismerés, akkor érdemes tovább futtatni, ha már megfelelő az eredmény, vagy már magát az objektumot sem ismeri fel, akkor leállíthatjuk a tanítást. Utóbbi esetben a részeredmények alapján össze tudunk rakni olyan kaszkádot, mely kevesebb szintből áll, így meg tudjuk keresni azt a szintet, amelyik a legideálisabb.

A [19] számú hivatkozáson olvasható cikk teszteredményei azt mutatják, hogy az Adaboost típusú tanulóalgoritmusok közül a Gentle Adaboost szerepel a legjobban. Így tanítás során ezt a tanulóalgoritmust használtuk.

Futás során akkor tekinti a kaszkád egy szintjét befejezettnek, amikor elérte a paraméterben átadott maximális hamis találati arányt, vagy a maximális gyenge tanuló fa (weak tree) értékét. Ezek a bináris döntési fák a kaszkád egy szintjén belül találhatók meg. Mélységük alapértelmezetten 1 értékű, tehát alapbeállítás esetén egy szinten belül olyan fák vannak, melyek pusztán egy pontból állnak, tehát egy gyenge osztályozó egy megtanult jellegzetességet jelent.



9. ábra. Tanítás közben a konzol: N a gyenge osztályozó fák száma, HR a találati arány, FA a maximális hamis találati arány. Látható, miként javul a hamis találati arány újabb jellegzetességek megtanulásával.

Mivel az egyes paraméterek mindig a betanítandó objektum függvényében mások, így az ezekkel kapcsolatos gyakorlati tapasztalatokat a 7. fejezetben fejezetben taglaljuk.

6. Az általunk fejlesztett tesztkörnyezet bemutatása

A tesztelésre használt programok mindegyikét magunk implementáltuk. A tesztelés első lépéseként okostelefonon kipróbáljuk, hogy mennyire sikerül a preview képen megtalálni az adott objektumot. Ha ezen a teszten jól szerepel, akkor folytatjuk a PC oldali programmal történő tesztelést, melynek segítségével a statisztikai értékeket számolhatunk arról, hogy detektorunk mennyire jó minőségű.

6.1. Android oldali tesztelő program

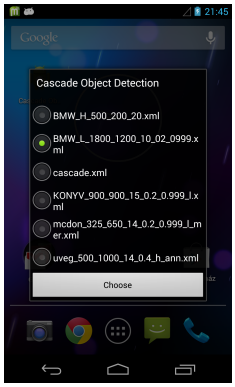
Ez a program valós időben teszteli az osztályozót. Futtatja a detektort, a megtalált objektumokat bekeretezi/bekarikázza. Amikor egy tesztfuttatást leállítunk az SD kártyára mentésre kerül, hogy minden egyes frame-en hány milliszekundumig tartott detektáló algoritmus lefutása. Ezen kívül menti, hogy átlagosan mennyi volt a futási idő, mikor megtalálta az adott objektumot, és mennyi, amikor nem találta meg. (Ezen nagyon pici időintervallumok pontos meghatározására hardveres performance counter-ek értékeit használja fel.) Így első körben könnyen eldönthető, érdemes-e a tesztelt kaszkádot további részletes teszt alá vetni.

Indításakor ki kell választani az előugró listából a tesztelendő .xml fájlt. Ezeket az SD kártyán az XmlForTest mappában keresi, tehát a tesztelendő kaszkádokat ide kell tennünk. Bal felső sarokban írja az aktuális FPS-t, a bal alsóban egy másik tesztelendő fájlt tölthetünk be. A mérések során kísérletet tettünk arra, hogy mennyire háttér-függetleníthető egy objektum detektálása, ha úgy tanítjuk be a detektort, hogy a tanítóhalmazon Canny [20] éldetektorral megkeressük a képen lévő éleket, majd detektálás során is először a képen lévő éleket detektáljuk, majd azon próbáljuk keresni a betanított objektum éleit. Erre nyújt lehetőséget a jobb felső sarokban a *use canny* gomb. Ekkor a preview-n a detektált élek jelennek meg. Erre példa a 10. ábrán látható.

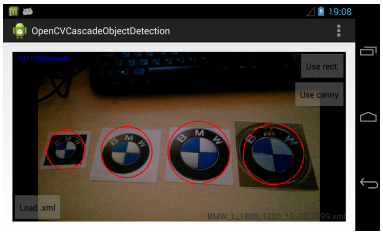
10. ábra. Canny éldetektor szemléltetése a láthatóság kedvéért inverzben

Ezzel a programmal ki lehet közelítőleg mérni, hogy mekkora távolságról illetve milyen szögből képes detektorunk felismerni az objektumot.

A programot LG Nexus 4 típusó telefonon futtattuk, a teszteredményeinkből a futási idő mérések erre a telefonra specifikusan jellemzők.



(a) A tesztelendő kaszkádok listája



(b) Tesztelés közben

11. ábra. A tesztprogram futás közben

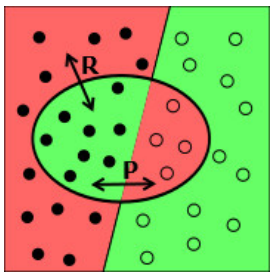
6.2. PC oldali tesztelő és tanítást támogató program

Ahhoz, hogy megértsük, hogy ez a program mivel támogatja a detektor értékelését, két fogalom ismerete szükséges, ezek az F-measure [21] és a 4-fold cross validation [22].

Az F-measure érték a vizsgált detektor minőségét jellemzi. Precision és recall harmonikus közepeként számítható ki:

$$F = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} \tag{11}$$

Ahol a precision a jó találatok és az összes találat hányadosaként, míg a recall a jól megtalált objektumok számának és az összes megtalálandó objektum számának hányadosaként kapható meg. Ezt egy szemléletes példán bemutatva:



12. ábra. F-measure szemléltetés [21]

A fenti képen a megtalálandó objektumhalmaz a fekete, a megtalált halmaz a középen bekarikázott rész. Ennek megfelelően a számított értékek:

$$precision = \frac{8}{12} \tag{12}$$

$$recall = \frac{8}{20} \tag{13}$$

$$F = \frac{2}{\frac{1}{\frac{8}{12}} + \frac{1}{\frac{8}{20}}} = \frac{2}{4} = \frac{1}{2} \tag{14}$$

Míg a precision vagy a recall elmehet valamilyen véglet irányába, az F-measure mentes ettől. Például ha a detektornak egyetlen jó találata van, és semmit nem detektált rosszul, akkor a precision értéke 1, ám mellette van még 9 amit nem ismert fel. Ekkor ha pusztán a precision-el jellemeznénk a detektort, a legjobb eredményt kapnánk, pedig a valóságban nem állja meg helyét megfelelően, hisz 10-ből csak 1-et ismert fel. Itt jön képbe a recall, ebben az esetben ennek az értéke $\frac{1}{10}$, tehát két eredmény harmonikus közepe már megfelelően jellemzi a detektort. (Ebben az esetben F = $\frac{2}{11}$).

A 4-fold cross validation egy olyan módszer, melynél nem különítjük el szigorúan a tanító és teszthalmazt, mikor a képek készülnek. A tanító és teszthalmazokat a 13. ábra szerint alakítjuk ki:

13. ábra. 4-fold cross validation halmazai

Tehát mindig a képek másik negyede a teszthalmaz, és a maradék háromnegyed pedig a tanító halmaz. Ez a művelet 4 detektortanítást és tesztelést jelent. Azért érdemes elvégezni, mert ezzel igazolni tudjuk, hogy nem azért lettek a teszteredményeink jók, mert épp jól választottuk meg a tanítóhalmazhoz a teszthalmazt.

Program három funkcióval támogatja a detektortanítási és tesztelési folyamatot.

- F-measure értékek számításához adatokat szolgáltat. Ezt úgy teszi, hogy átadjuk neki a teszthalmazt, és a tesztelendő kaszkádot, majd a képeken lefuttatja és bekeretezi a találatokat. A teszt végén találatokról statisztikát készít egy .csv fáljba, azt kell ellenőriznünk, hogy a találatok valóban jók voltak-e. Például ha van a tesztképen egy sötétebb papucs és egy sörösüveg, és a sörösüveget szeretnénk detektálni, akkor nem-e a papucsot detektálta véletlenül helyette. Ezeket hamis találatokat korrigálva egy Excel táblázatban könnyen ki tudjuk számolni az F-measure értéket.
- Egy meglévő annotált képhalmazt 4 részre bont a 4-fold cross validation elvégzéséhez. Kimenetként megkapjuk mind a négy teszthalmazt és a hozzájuk tartozó annotálást tartalmazó info fájlokat a pozitív tanítóhalmazokról. Az utóbbiak alapján a kaszkádok legyárthatók, majd tesztelhetők az előbbi teszthalmazokon.

- Egy megadott képhalmazon lefuttatja a Canny éldetektort, ezzel megteremtve annak a lehetőségét, hogy olyan képeink képezzék a tanítóhalmazt, melyen csak az élek látszanak. (Ehhez kapcsolódik a telefonon a canny alapján történő detektálás).

7. Tanítási tapasztalatok és teszteredmények bemutatása

Minden esetben az elkészült kaszkádot először a telefon preview képén próbáltuk ki. Azért nem volt érdemes rögtön statisztikát készíteni, mert az időigényesebb, és gyakran előfordult, hogy kaszkádunk túltanított lett (azaz nem ismeri fel magát az objektumot sem), vagy túl sok mindent ismert fel. A preview képen jól teljesítők közül kiválasztottunk 3-at vagy 4-et, és ezekről statisztikát készítve választottuk ki a legjobban sikerült kaszkádot.

A detektálási idő mérésénél összehasonlítottuk, hogy mennyi a különbség abban az esetben, amikor felismeri az objektumot, és mennyi, amikor nem látható a preview képen az objektum. Általánosan elmondható, hogy nincs nagy különbség köztük, a detektálás általában 3-4 milliszekundummal több. Várható volt, hogy több lesz, hisz ekkor a felismerés során a kaszkád minden szintjén végighalad, ami kicsit időigényesebb, mint amikor nem kell végighaladni rajta. Viszont az meglepő volt, hogy csak pár milliszekundum a különbség, többre számítottunk.

A mérések során a távolság és szögmérési eredmények közelítő értékek, a lehetőségekhez mérten igyekeztünk a lehető legpontosabban kimérni őket.

Hogy a felismerési módszer hétköznapi életben való használhatóságát teszteljük, olyan objektumtípusokat tanítottunk be, melyekkel a hétköznapok során számtalanszor találkozunk: logókat, tárgyakat és épületet.

7.1. BMW logó

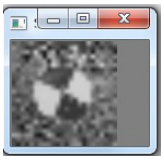
14. ábra. A felismerendő BMW logó

Első kísérletünk a BMW logó felismerését tűzte ki célul. Az OpenCV library dokumentációjában olvasható [23], hogy jellegzetes logókhoz készíthető úgy is eredményesen detektor, hogy a CreateSamples segítségével generálunk egyetlen képből néhány száz képet tartalmazó tanítóhalmazt, majd ennek segítségével tanítunk.

Az első próbálkozások során alapértelmezett tanítási paramétereket használtunk, 15-20 szintes kaszkád készítéséhez. Eredményként nagyon gyenge minőségű detektorok születtek, szinte mindent felismertek logónak a preview képen. Ennek oka az volt, hogy tanítóhalmazt 50-100 körüli pozitív és negatív képek képezték, mely nagyon kevésnek számít.

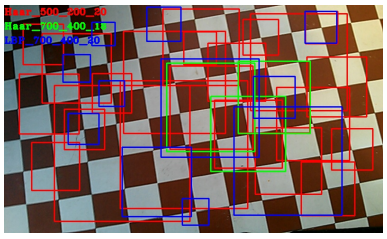
A pozitív és negatív képek számát növelve (közelítőleg 200-200-ra) egész jó kaszkádot sikerült készíteni, mely olyan képek esetén jól működött, ahol nagyjából ugyanolyan fényviszonyok között készültek, mint a betanításhoz használt alapkép. Például ha az alapkép egy olyan logó, mely ellenőrzött körülmények között készült (nem fénylik), akkor az a valóságban egy kocsin lévő fénylő logón már kevésbé működött jól.

Ennek kiküszöbölése érdekében a CreateSamples programmal generáltunk 5 különböző kiinduló képből (egyik standard logó, melyet plakáton használnak, másik kocsin készült stb.) 5 .vec fájt, melyeknek mindegyike 150-150 pozitív képet tartalmazott, majd ezt fűztük össze egy 750 képet tartalmazó tanítóhalmazzá. Ennek kapcsán ütköztünk abba a hibába, hogy ha a CreateSamples programnál nem kapcsoljuk ki az átlátszóság használatát és háttérképet használunk a generálás során, akkor a kapott képek eltorzulnak, ezzel hátráltatva az eredményes tanítást.



15. ábra. A CreateSamples által eltorzított logó

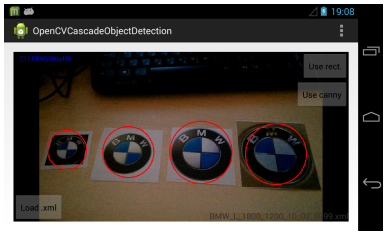
Tehát 750 pozitív (argumentumként ebből 700-at átadva) és 400 negatív képet használva, a kaszkádot 20 szintesre beállítva a tanítás eredménye már elfogadható volt. További fontos tapasztalat, hogy a negatív képek közé érdemes olyan képeket is beválogatni, mellyel a detektor számára könnyen összetéveszthető az adott objektum. Ilyen volt például a BMW logó esetében a járólap, melynél két sötétebb és két világosabb csempe átellenesen találkozik a logó közepéhez hasonlóan. Mivel a tanítás során épp a logó közepén lévő feature pontokat lettek betanítva, így a járólap ezen részeit is logónak detektálta a tesztprogram.



16. ábra. Három kaszkád találatai, a negatív tanítóhalmazban nem szerepel a járólap

Miután a negatív tanítóhalmaz kiegészült a járólapról készült képekkel, ez a hiba is kiküszöbölődött. Próbaként az egyes alapképekből 150 helyett 400-at generáltunk a CreateSamples-el, majd ezeket egy .vec-é alakítva próbát tettünk, hogy javul-e a detektor minősége. Ezzel gyakorlatilag a tanítási idő nőtt, de ebben az esetben nem hozott jobb eredményt.

Egy másik próbaként a tanítás során a szintek számát kevesebbre választottuk, ám az egyes szinteken szigorúbb paramétereket szabtunk meg. Például az alapértelmezett szintenkénti 0.5 értékű maxFalseAlarmRate helyett 0,2-t, a 0,95 értékű minHitRate helyett pedig 0,999-et használva 10 szintet definiálva nagyon jó eredményt kaptunk.



17. ábra. A megfelelően működő detektor

A pozitív képeket tartalmazó teszthalmazban 132 egy kocsin lévő logóról készült, 131 pedig papírra nyomtatott logókat tartalmazott, továbbá 900 negatív képen is futtatva lettek a detektorok. A legjobban sikerült kaszkádok statisztikai eredményei a 1. táblázatban láthatók.

1. táblázat. Teszteredmények

Detektor	Precision	Recall	F-measure	Átlagos detektálási idő
Haar_500_200_20	0,6154	0,9430	0,7447	536 ms
LBP_700_400_20	0,7458	0,8479	0,7936	49 ms
Haar_700_400_18	0,8087	0,9164	0,8592	58 ms
LBP_1800_1200_10_0.2_0.999	0,9868	0,8551	0,9165	55 ms

Ahol a detektornevek a tanítás paramétereit tartalmazzák az alábbi mintát követve:
<Leíró típusa>_<pozitív képek száma>_<negatív képek száma>_<szintek száma>_
<maxFalseAlarmRate>_<minHitRate>
A névben meg nem említett paraméterek alapértelmezettek. A továbbiakban is ehhez hasonlóan értelmezendőek a detektornevek.

A fenti táblázatban feltűnő, hogy a Haar_500_200_20-as detektálási ideje majdnem tízszerese a többinek. Ez azért van, mert a tanítási folyamat során már az első néhány szinten nagyon sok jellegzetességet jegyzett meg, majd a magasabb szinteken még ezeknél is többet, tehát feleslegen jegyzett meg túl sok jellegzetességet.

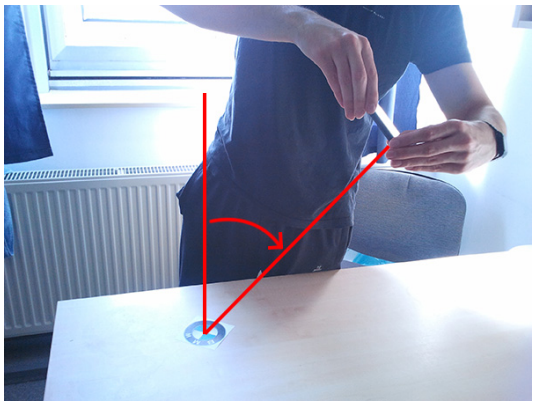
Látható hogy a legjobb eredményt a 10 szintes LBP érte el. Itt nem a nagy elemszámú pozitív tanító halmaz, hanem a gondosabban választott negatív tanítóhalmaz által sikerült jobb eredményt elérni. Ezen kaszkád detektálási szögeinek mérési eredménye az alábbi:

2. táblázat. Távolság és szögmérési eredmények

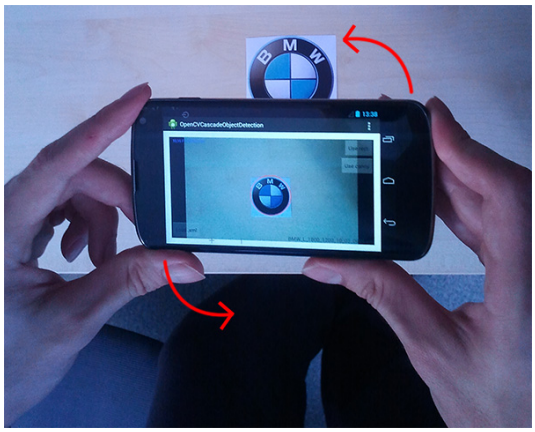
Szögmérés távolsága	50 cm
Kamera szöge a logó közepétől előre és hátra körívesen mozgatva	-70° és 70° között
Kamera szöge a logó közepétől balra és jobbra körívesen mozgatva	-45°-tól 45°-ig
Adott szögből telefon oldal irányú dönthetősége	25°
Szemből nézve maximális távolság	1 m

A fentebb írt határokon kívül a detektor elveszíti a célpontot. A szögek értelmezése:

(a) Kamera szöge a logó közepétől előre és hátra körívesen mozgatva



(b) Kamera szöge a logó közepétől balra és jobbra körívesen mozgatva



(c) Adott szögből telefon oldal irányú dönthetősége

18. ábra. A szögek értelmezése

(a) Épületen

(b) Piros háttéren

(c) Fehér háttéren

(d) Felirat a logóban

19. ábra. A McDonald's logó hátterének sokszínűsége

7.2. McDonald's logó

Ezen logó betanítása komplexebb feladat, mint a BMW logóé, ugyanis a háttere mindig más. Erre néhány személetes példa a 19. ábrán látható.

Először a BMW logónál bevált módszerrel próbálkoztunk, a fenti képeket alapul véve generáltunk 300-300 képet mindegyikből, majd egyesítettük a kapott fájlokat. Ezzel a módszerrel a tanítás eredményeként nagyon rossz detektorokat kaptunk, mind Haar, mind LBP leíró használatával. Ennél a gondot egyrészt a változatos háttér jelentette, másrészt hogy maga a logó sem mindig ugyanolyan (Pl. valamelyikben felirat van). Így a tanító algoritmus nem tudta megjegyezni a logót egyértelműen azonosító jellegzetességeket.

Ennek kapcsán merült fel az ötlet, hogy a Canny éldetektor segítségével detektáljuk a logó körvonalait, ezzel háttér-függetlenítve a felismerést. Ez esetben a tanító halmaz elemei is csak a detektált éleket tartalmazzák, valamint az objektum detektálása előtt is lefut az éldetektor, majd ezen keresi a megtanult jellegzetességeket. Először egyetlen pozitív képet felhasználva a CreateSamples segítségével próbáltunk képeket generálni, majd ezzel tanítani, de ez nem volt eredményes, ugyanis túl kevés információt tartalmaztak a képek ahhoz, hogy az egyes szinteken elérjük a megfelelő maxFalseAlarmRate értékét. Következő lépésként egy annotált képhalmazon futtattuk le a Canny-t, majd ez alapján próbáltuk meg a tanítást. Itt a TrainCascade már sikerrel járt, ám a kaszkád valós időben nem szerepelt jól. A Canny detektor egy újabb bizonytalanságot vitt felismerési folyamatba, ugyanis sokszor magát a körvonalat sem találta meg, és így mikor a kaszkád lefutott, a keresett objektum körvonala nem volt látható a képen, továbbá amikor felismerhető volt körvonal, akkor is nagy bizonytalansággal ismerte csak fel. Így ezen módszer használatát elvetettük.

A detektortanítás szürkeárnyalatos képek alapján történik (a CreateSamples átméretezi és szürkeárnyalatossá teszi), valamint a detektálás során is ilyeneken fut a detektor. A szürkeárnyalatos képeken a fehér háttéren lévő sárga logó emberi szemmel is nagyon nehezen felismerhető, így ezeket kizártuk a tanítóhalmazból. 375 annotált képet használva, a pozitív képek közé csak piros hátterűeket beválogatva egész jó minőségű detektort sikerült csinálni, melyek ellenőrzött körülmények között nagyon jól szerepeltek (tehát a háttér nem bármilyen, hanem összefüggően

sötétebb színű).

A teszthalmazba egy McDonald's-os műanyagpoháron és egy a krumplit tartalmazó kartonon lévő logót használtunk, tehát elmondható, hogy a teszthalmaz képei ellenőrzött körülmények között készültek. A teszteléshez 195 pozitív képet és 900 negatívat használtunk. Ennek eredménye:

3. táblázat. Teszteredmények

Detektor	Precision	Recall	F-measure	Átlagos detektálási idő
Haar_325_650_10_0.2_0.999	0,7276	0,8358	0,7780	71 ms
LBP_325_650_14_0.2_0.999	0,9938	0,8307	0,9050	56 ms

Ha a teszthalmazba olyan logókat tennénk, melynek változatos a háttere, az eredmények drasztikusan rosszabbodnának.

Az LBP_325_650_14_0.2_0.999 távolság és szögmérési eredménye:

4. táblázat. Távolság és szögmérési eredmények

Szögmérés távolsága	20 cm
Kamera szöge a logó közepétől előre és hátra körívesen mozgatva	-45° és 30° között
Kamera szöge a logó közepétől balra és jobbra körívesen mozgatva	-45°-tól 45°-ig
Adott szögből telefon oldal irányú dönthetősége	20°
Szemből nézve maximális távolság	50 cm

Konklúzióként elmondható, hogy ezt a logót egyelőre csak részben sikerült ezzel a módszerrel detektálni. Általános eset megoldására érdemes lenne kipróbálni a jellegzetesség megfeleltetés alapú objektumfelismerés módszerét.

7.3. Könyv

20. ábra. A felismerendő könyv borítója

Első próbálkozásként a fentebb látható könyv borítóját a CreateSamples által generált tanítóhalmaz segítségével tanítottuk. Ehhez egy alapképet felhasználva 900 képet generáltunk odafigyelve arra, hogy a generált kép oldalarányai megegyezzenek a könyv oldalarányaival (ez

esetben 14x42). Már ezzel a módszerrel is egész jó detektor született, de nem voltak teljesen magabiztosak. Ezért a tanítóhalmazhoz 202 annotált képet csatoltunk, és kipróbáltuk, hogy javít-e a minőségen. Ezzel a módszerrel sikerült elérni a legjobb eredményt. Az így készült detektorok közül valósidejű tesztből a legjobbat kiválasztottuk, melyet 330 pozitív és 900 negatív képet teszthalmaz segítségével teszteltünk. Ennek az eredményei:

5. táblázat. Teszteredmények

Detektor	Precision	Recall	F-measure	Átlagos detektálási idő
LBP_900_900_15_0.2_0.999	0,9944	0,5454	0,7045	57 ms

Ennél tesztnél a teszthalmazból a pozitív képek picit több mint a felét ismerte csak fel. Az eredmény meglepő volt, ugyanis a telefonon futtatott valós idejű teszten nagyon jól szerepelt a kaszkád. Valószínűleg a tesztképek egy része épp olyan szögből készült a könyvről, mely nézetből a könyv nem volt felismerhető a detektor számára.

6. táblázat. Távolság és szögmérési eredmények

Szögmérés távolsága	40 cm
Kamera szöge a logó közepétől előre és hátra körívesen mozgatva	-45° és 45° között
Kamera szöge a logó közepétől balra és jobbra körívesen mozgatva	-45°-tól 30°-ig
Adott szögből telefon oldal irányú dönthetősége	15°
Szemből nézve maximális távolság	1,5 m

A könyv esetében az annotált és generált képek közösen képezték a tanítóhalmazt, ennek segítségével viszonylag hamar megfelelő minőségű kaszkádot sikerült készíteni.

7.4. Sörösüveg

21. ábra. Négy kép a tantó/teszt halmazból

Ennél a tanításnál nem egy specifikus sörmárka felismerését tűztük ki célul, hanem általánosan a barna sörösüveg felismerését. Így a tanítóhalmazba különböző címkéjű üvegeket válogattunk be, ezzel elkerülve, hogy ezek jellegzetességeit használja fel a kaszkád. (Ha konkrétan egy sörmárkát szeretnénk felismerni, akkor azt a címke alapján tudnánk megtenni.)

Először annotált képhalmaz segítségével tanítottuk a detektort. Ehhez 600 pozitív kép készült, vegyes háttérrel, mindenféle szögből, más-más üvegekről. Ezek mindegyikét annotáltuk, majd lefuttattuk a tanító programot. Az eredmény elsőre a vártál jobb lett.

További javításra úgy tettünk kísérletet, hogy a CreateSamples-el generáltunk 450 képet úgy, hogy 150-150 képeket generáltunk a sört körbeforgatva, ezzel elkerülve azt, hogy túl sokszor szerepeljen a tanítóhalmazban ugyanaz a címke. A tanítóhalaz generálása során használt képarány 14×42.

Az így kapott eredmények a következők lettek:

7. táblázat. Teszteredmények

Detektor	Precision	Recall	F-measure	Átlagos detektálási idő
LBP_900_1000_17_0.3_0.999_mixed	1	0,3724	0,5427	26 ms
Haar_900_1000_15_mixed	0,7763	0,8138	0,7946	32 ms
Haar_500_1000_14_0.4_ann	0,945	0,8345	0,8864	29 ms

A táblázatban a tanítási paraméterek után a „mixed” arra utal, hogy az annotált és a generált képek keverékét tartalmazta a tanító halmaz. Meglepő, hogy ebben az esetben az annotált tanítóhalmazhoz csatolt generált halmaz rontott az eredményeken. Ez épp ellentettje annak, amit a könyvnél tapasztaltunk. A 17 szintes LBP-nél a kevesebb szintet használva a félreismerések száma drasztikusan nőtt, az F-measure értéket bár nem számoltuk ki, de nem sokat javult volna a precision romlása miatt. Mindháromról elmondható, ha az üveg hátterébe sötétebb dolog kerül, mely az üveg színéhez hasonlít, akkor egyik sem tudja felismerni.

A Haar_500_1000_14_0.4_ann kaszkád szögmérési eredményei a következők:

8. táblázat. Távolság és szögmérési eredmények

Szögmérés távolsága	50 cm
Kamera magasságának szöge az üveg közepétől	-15° és 30° között
Adott szögből telefon oldal irányú dönthetősége	15°
Szemből nézve maximális távolság	1 m

Az üveget 360°-ot körbeforgatva az adott tartományban folyamatosan, magabiztosan felismeri.

Az eredmények alapján az annotált halmazon érdemesnek láttuk a 4-fold cross validation kipróbálását, ezzel igazolva azt, hogy nem a szerencsés teszthalmazválasztás miatt lett ilyen jó eredményünk. Ennek elvégzéséhez kiegészítettük a PC oldali tesztelőprogramot egy olyan funkcióval, mely egy annotált képhalmaz alapján 4 tanító és teszthalmazt készít a 13. ábra alapján. Az eredeti képhalmaz első felében csak egy fajta üvegről készült képek voltak, a második felében egy másikról és így tovább. Ezért a 4 halmazt úgy hoztuk létre, hogy az első képtől kezdve minden negyedik kép került az első halmazba, majd a másodiktól kezdve minden negyedik a következőbe és így tovább. Ezzel elkerültük azt, hogy előálljon olyan eset a teszt

során, hogy a tanítóhalmazban nem szerepel az az üveg, ami épp a teszthalmazba került. A tanítás során ugyanazokat a paramétereket használtuk, mint az eredetinél (a pozitív tanítóhalmazt leszámítva). Az így létrehozott négy kaszkád teszteredménye:

9. táblázat. 4-fold cross validation teszteredményei

Halmazok	Precision	Recall	F-measure
Első	0,9219	0,7867	0,8489
Második	0,9417	0,7533	0,8370
Harmadik	0,9375	0,8000	0,8633
Negyedik	0,9672	0,7919	0,8708

Látható, hogy a kaszkád ezen a teszten is megállta a helyét, hisz mind a négy halmaznál magasabb lett az F-measure értéke 0,8-nál. Így ez a tanítás sikeresnek mondható.

7.5. Épület

22. ábra. A Parlament [24]

Egy épület detektálhatóságának vizsgálata során kísérletet tettünk az Országház felismerésére. Ehhez 1650 képet készítettünk az épületről különböző szögekből a budai oldalról. Ennek közelítőleg a felét terveztük tanítóhalmazként felhasználni, másik felét pedig teszthalmazként. Így 950 annotált kép képezte a tanítóhalmazt, valamint 1200 kép a negatív tanítóhalmazt, a használt képarány 42×14. Különféle paraméterekkel tanítva arra az eredményre jutottunk, hogy a detektor kizárólag azokon a képeken működik jól, mely teszthalmazban volt, tehát ami akkor készült, amikor a tanítóhalmaz is. Ez azzal magyarázható, hogy azokon a képeken az épület ugyanolyan fényviszonyok közt lett lefényképezve, így ezeken sikerült megtalálni a megfelelő jellegzetességeket. Ám ha egy máskor készült képen próbáltuk detektálni az épületet, azon már nem sikerült megtalálni. Így az előkészített teszthalmazon lefuttatva a detektorokat a statisztikák félrevezetőek lennének, ezért azokat nem készítettük el.

Ezzel a detektálási módszerrel úgy lehetne javítani ezen a hibán, hogy nagyon sokszor kellene képeket készíteni az épületről különböző fényviszonyokban, különböző szögekből, és mindet annotálni kellene. Ez nagyon fáradtságos és hosszadalmas munka lenne, így ez esetben alternatív megoldást kell találni.

Telefonok esetén egy megoldás lehetne jellegzetesség megfeleltetés és a GPS koordináták adatainak közös használata. A GPS adatok alapján nagyon leszűkülne a lehetséges felismeren-

dő épületek száma, majd az épületen néhány jellegzetes pontot megtalálva már egyértelműen beazonosítható lenne, hogy melyik épületről van szó.

8. Összegzés

A dolgozat elkészítése kapcsán megismerkedtünk a széles körben elterjedt objektum felismerő algoritmusok elméleti hátterével. Ezek közül a kaszkádosított osztályozó alapú detektálási módszert választottuk ki és vizsgáltuk meg részletesen a gyakorlati képességeit valós idejű alkalmazásokra. A tesztek során hétköznapi életből vett objektumokon próbáltuk ki az algoritmus hatékonyságát.

Az időmérési eredmények alapján elmondható, hogy a kiválasztott módszer egy objektum felismerése esetén teljes mértékben alkalmas okostelefonon futtatva is a valós idejő detektálás megvalósítására. Ha több különböző objektumot szeretnénk egyidejűleg felismerni, akkor a detektorokat egymás után futtatva a futási idő lineárisan nő, az általunk tesztelt készüléken körülbelül 4-5 különböző detektor futtatása esetén vált a működés nem valós idejűvé, de egyéb célokra ilyen esetben is alkalmas marad a módszer. A futási teljesítményen javítani több magos processzorok esetén párhuzamosítással még lehetne, ha az egyes detektorok külön szálakon futnának az egyes magokon, akkor egy ilyen környezetben több detektorral is elérhető lehetne a valós idejű futás.

Ha a módszert kiterjesztett valóság alapú szoftverben szeretnénk felhasználni, túl sok objektum detektálását nem tervezhetjük vele, mert azzal drasztikusan romlana a futási idő, romlana a kép folyamatossága és ezzel a felhasználói élmény. De ha programunknak csak néhány objektumot kell felismernie, akkor bátran használható.

9. Függelék

9.1. Az ObjectMarker program használata

Az .exe fájl parancssori argumentumok formájában 2 paramétert kap (ebben a sorrendben):

1. A kimeneti .txt fájl nevét, mely az annotálási információkat tartalmazni fogja.
2. Az annotálandó képek elérési útvonalát.

Egy példa erre: *ObjectMarker.exe konyv_anninfo.txt pos\konyv*

Használat során SPACE gombbal menthetjük a jelölést, és ENTER gombbal tölthetjük be a következőt. [17]

9.2. A CreateSamples program használata

Annotált képhalmazból .vec fájl készítése a következő parancssori argumentumok segítségével valósítható meg[23]:

- -info <*info_file_neve*> Annotálást tartalmazó infó fájl neve.
- -vec <*vector_file_neve*> A kimeneti .vec fájl neve.
- -num <*kép_szám*> A .vec-be rakandó pozitív képek száma. Figyeljünk, hogy ne lépje túl az összes képünk számát, mert ha több, akkor hibával fejeződik be a programvégrehajtás. Ez akkor jelent problémát, ha a MergeVec nevű programmal össze akarjuk fűzni egy másik .vec fájllal.
- -w <*szélesség*> A .vec-ben tárolt kimeneti kép szélessége pixelben.
- -h <*magasság*> A .vec-ben tárolt kimeneti kép magassága pixelben.
- -show Megmutatja a generálás során generált képeket a .vec fájlban.

Magasságnak és szélesség alapértéke 24×24 pixel. Ezek által leírt területnél sokkal nagyobbat nem érdemes használni, mert növeli a futási időt. Viszont arra figyeljünk, hogy a betanítandó objektum arányaihoz illeszkedjen a magasság és szélesség. Továbbá mindig érdemes a -show paranccsal a generált halmazt megnézni, ezzel ellenőrizve az arányokat és a minőséget. (A SPACE gombot nyomva tartva hamar végig lehet pörgetni pár száz képet, így ez nem időigényes).
Példa argumentumok átadására: *CreateSamples.exe -info pos\konyv\ann_info.txt -vec vectores\ann_vec -num 100 -show -w 20 -h 26*

Egyetlen képből több generálása szoftveres úton - mely pozitív tanítóhalmazként szolgál - az alábbi argumentumok segítségével érhető el: [23]

- -img <*képfile_neve*> A szoftveresen elforgatandó kép elérési útvonala.
- -vec <*vector_file_neve*> A kimeneti .vec fájl neve.

- -bg <*háttér_képeket_leíró_file_neve*> A leíró .txt fájlban szereplő képeket háttérként használja a generálás során. Körültekintően válasszuk meg ezeket, mert rossz háttér és áttetszőség esetén teljesen eltorzulhat a pozitív kép.
- -bgcolor <*háttér_szín*> A háttér színe szürkeárnyalatos kép esetén. Értéke 0-255 között lehet.
- -bgthresh <*háttér_szín_threshold*> Áttetszőség beállítása, 0 esetén nincs áttetszőség.
- -inv Invertál minden elkészült képet.
- -randinv Véletlenszerűen invertál képeket.
- -maxxangle <*maximális_x_irányú_forgatás*> Maximális x irányú elforgatás értéke radiánban.
- -maxyangle <*maximális_y_irányú_forgatás*> Maximális y irányú elforgatás értéke radiánban.
- -maxzangle <*maximális_z_irányú_forgatás*> Maximális z irányú elforgatás értéke radiánban.
- -num <*kép_szám*> A .vec-be rakandó pozitív képek száma.
- -w <*szélesség*> A .vec-ben tárolt kimeneti kép szélessége pixelben.
- -h <*magasság*> A .vec-ben tárolt kimeneti kép magassága pixelben.
- -show Megmutatja a generálás során generált képeket a .vec fájlban.

Amikor így generálunk pozitív tanítóhalmazt, feltétlenül nézzük végig a készülő képeket ugyanis több esetben is olyannyira eltorzulhat a kimeneti kép, hogy gyakorlatilag teljesen felismerhetetlenné válik. Így ha ilyenek előfordulnak a tanítóhalmazunkban, nagyon rossz detektort kapunk. A magasság és szélesség arányának itt is illeszkedniük kell a betanítandó objektumhoz. Fontos, hogy a felhasznált kép kizárólag a felismerendő objektumot tartalmazza.
Példa argumentumok átadására: *CreateSamples.exe -img pos\konyv 1.jpg -vec vectores\test_vector -num 100 -bgcolor 200 -show*

Egy meglévő .vec fájl tartalmának megtekintése esetén csak a -vec <*vector_file_neve*> paramétert adjuk át a programnak, ezzel megadva egy már létező .vec fájlt. Példa argumentumok átadására: *CreateSamples.exe -vec vectores\test_vector*

9.3. A MergeVec program használata

Ahhoz, hogy egyesíteni tudjunk a CreateSamples kimeneti fájljait, készíteni kell erre a célra egy info.txt-t. Fontos, hogy ez az egyesítendő fájlok elérési útvonalát tartalmazza, nem csak az adott .vec fájlok nevét!

Az használandó argumentumok a következők (az első kettő sorrendje kötött) [18]:

- Bemeneti .vec fájlok elérési útvonalát tartalmazó info fájl neve.
- Az összefűzött kimeneti .vec fájl neve.
- -show Megnézhetjük az egyesített .vec tartalmát.

- -w <*szélesség*> A .vec-ben tárolt kép szélessége pixelben.
- -h <*magasság*> A .vec-ben tárolt kép magassága pixelben.

Fontos, hogy az előzmény .vec fájlokban használt magasság és szélesség értéket adjuk át itt is argumentumként.

9.4. A TrainCascade parancssori argumentumai

A tanításhoz használt program argumentumai a következők [23]:

1. Átalános paraméterek

    - -data <*kimeneti_mappa_neve*> A mappa neve, ahová a tanítás részeredményei és a végeredmény kerül. Fontos, hogy létező mappát adjunk meg, a TrainCascade nem hoz létre mappát, ha nem létezik, hanem hibával kilép.
    - -vec <*bementi_vec fájl_neve*> A .vec fájl elérési útvonala, melyben a pozitív képek vannak.
    - -bg <*negatív_képek_leíró_info_fájl_neve*> Negatív képhalmazt leíró fájl elérési útvonala.
    - -numPos <*pozitív_minták_száma*> Tanítás során használt pozitív képek száma. Az itt megadott számon felül is használ az algoritmus pozitív képeket a megadott .vec fájlból. Ha a .vec fájlunk nem tartalmaz kellő számút, a program hibával kilép. Így közelítőleg a .vec-fájlban található képek számának 0.8 - 0.9 szorosát érdemes megadni, így biztosan nem ütközünk ebbe a hibába.
    - -numNeg <*negatív_példák_száma*> Negatív képek száma.
    - -numStages <*szintek_száma*> A kaszkád szintjeinek száma.
    - -precalcValBufSize <*puffer_méret*> Puffer mérete megabyte-ban jellegzetesség értékek számára.
    - -precalcIdxBufSize <*puffer_méret*> Puffer mérete megabyte-ban jellegzetesség indexek számára.

2. Kaszkád paraméterei

    - -stageType <*BOOST(default)*> Szintek típusa, egyelőre csak BOOST lehet.
    - -featureType <*HAAR,LBP*> A leíró típusa.
    - -w <*szélesség*> A .vec-ben tárolt kép szélessége pixelben.
    - -h <*magasság*> A .vec-ben tárolt kép magassága pixelben.

3. Boost paraméterei

    - -bt <*DAB, RAB, LB, GAB(default)*> A boosting algoritmus típusa. A paraméterek jelentése: *DAB* - Discrete AdaBoost, *RAB* - Real AdaBoost, *LB* - LogitBoost,

GAB - Gentle AdaBoost. A fenti AdaBoost típusok és azok összehasonlítása a [19] számú hivatkozáson olvasható. Ennek tanulsága alapján az alapértelmezett Gentle Adaboost-ot használjuk tanítás során.

- -minHitRate <*min_hit_rate_értéke*> Az osztályozó minden szintjén a minimális találati arány.
- -maxFalseAlarmRate <*max_false_alarm_rate_értéke*> Maximális hamis találat aránya minden egyes szinten.
- -weightTrimRate <*weight_trim_rate_értéke*> Súlyozás értéke.
- -maxDepth <*gyenge_osztályozó_ maximális_mélysége*> Egyetlen gyenge osztályozó fa (weak tree) maximális mélysége. Ezek a bináris döntési fák a kaszkád egy szinten belül találhatók meg. Mélységük alapértéke 1, tehát alapbeállítás esetén belül olyan fák vannak, melyek pusztán egy pontból állnak.
- -maxWeakCount <*maximális_gyenge_osztályozók_száma*> Tanítás során maximális gyenge osztályozó fa száma. (Tanítási folyamat során a konzolon N.)

4. Haar esetén paraméterek

   - -mode <BASIC (default) | CORE | ALL> Meghatározza, hogy a Haar jellegzetességek közül melyeket használja.
   - -baseFormatSave Régi formátumba ment, mely kompatibilis a haartraining.cpp-vel.

Hivatkozások

[1] OpenCV documentation: matchTemplate. Accessed: 2013-10-19. [Online]. Available: http://docs.opencv.org/modules/imgproc/doc/object_detection.html?highlight=matchtemplate#matchtemplate

[2] OpenCV documentation: minMaxLoc. Accessed: 2013-10-19. [Online]. Available: http://docs.opencv.org/modules/core/doc/operations_on_arrays.html?highlight=minmaxloc#minmaxloc

[3] OpenCV documentation: Template matching. Accessed: 2013-10-19. [Online]. Available: http://docs.opencv.org/doc/tutorials/imgproc/histograms/template_matching/template_matching.html

[4] P. Viola and M. Jones, „Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1. IEEE, 2001, pp. I–511.

[5] R. Lienhart and J. Maydt, „An extended set of haar-like features for rapid object detection," in *Image Processing. 2002. Proceedings. 2002 International Conference on*, vol. 1. IEEE, 2002, pp. I–900.

[6] OpenCV documentation: CascadeClassifier. Accessed: 2013-10-23. [Online]. Available: http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html?highlight=cascadeclassifier#cascadeclassifier

[7] T. Ojala, M. Pietikäinen, and D. Harwood, „A comparative study of texture measures with classification based on featured distributions," *Pattern recognition*, vol. 29, no. 1, pp. 51–59, 1996.

[8] T. Ahonen, A. Hadid, and M. Pietikäinen, „Face recognition with local binary patterns," in *Computer Vision-ECCV 2004*. Springer, 2004, pp. 469–481.

[9] T. Ojala, M. Pietikainen, and T. Maenpaa, „Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 7, pp. 971–987, 2002.

[10] S. Liao, X. Zhu, Z. Lei, L. Zhang, and S. Z. Li, „Learning multi-scale block local binary patterns for face recognition," in *Advances in Biometrics*. Springer, 2007, pp. 828–837.

[11] M. Muja and D. G. Lowe, „Fast approximate nearest neighbors with automatic algorithm configuration." in *VISAPP (1)*, 2009, pp. 331–340.

[12] OpenCV documentation: Common interfaces of feature detectors. Accessed: 2013-10-25. [Online]. Available: http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_feature_detectors.html

[13] OpenCV documentation: Common interfaces of descriptor extractors. Accessed: 2013-10-25. [Online]. Available: http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_descriptor_extractors.html

[14] OpenCV documentation: Common interfaces of descriptor matchers. Accessed: 2013-10-25. [Online]. Available: http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_descriptor_matchers.html

[15] OpenCV documentation: findhomography. Accessed: 2013-10-25. [Online]. Available: http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=findhomography#findhomography

[16] OpenCV documentation: installation. Accessed: 2013-10-15. [Online]. Available: http://docs.opencv.org/doc/tutorials/introduction/windows_install/windows_install.html

[17] ObjectMarker program source. Accessed: 2013-10-15. [Online]. Available: http://www.technolabsz.com/2012/07/how-to-do-opencv-haar-training-in.html

[18] MergeVec program source. Accessed: 2013-10-15. [Online]. Available: http://note.sonots.com/SciSoftware/haartraining/mergevec.cpp.html

[19] V. P. Rainer Lienhart, Alexander Kuranov, „Empirical analysis of detection cascades of boosted classifiers for rapid object detection," in *Pattern Recognition*. Springer, 2003, pp. 297–304.

[20] Canny edge detector. Accessed: 2013-10-17. [Online]. Available: http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html

[21] FMeasure. Accessed: 2013-10-18. [Online]. Available: http://en.wikipedia.org/wiki/Precision_and_recall

[22] Cross validation. Accessed: 2013-10-18. [Online]. Available: http://en.wikipedia.org/wiki/Cross-validation_%28statistics%29

[23] OpenCV cascade classifier training. Accessed: 2013-10-15. [Online]. Available: http://docs.opencv.org/doc/user_guide/ug_traincascade.html

[24] Parlament fotó. Accessed: 2013-10-24. [Online]. Available: http://upload.wikimedia.org/wikipedia/commons/c/c2/Parlament_%28Budapest,_HU%29.JPG